A Monte Carlo tree search approach to learning decision trees

Cecília Nunes
Universitat Pompeu Fabra,
Barcelona, Spain
Philips Research Medisys
Paris, France
cecilia.nunes@upf.edu

Mathieu De Craene
Philips Research Medisys
Paris, France

Hélène Langet
Philips Research Medisys
Paris, France

Oscar Camara
Universitat Pompeu Fabra,
Barcelona, Spain

Anders Jonsson
Universitat Pompeu Fabra,
Barcelona, Spain

***Abstract*—Decision trees (DTs) are a widely used prediction tool, owing to their interpretability. Standard learning methods follow a locally-optimal approach that trades off prediction performance for computational efficiency. Such methods can however be far from optimal, and it may pay off to spend more computational resources to increase performance. Monte Carlo tree search (MCTS) is an approach to approximate optimal choices in exponentially large search spaces. Since exploring the space of all possible DTs is computationally intractable, we propose a DT learning approach based on MCTS. To bound the branching factor of MCTS, we limit the number of decisions at each level of the search tree, and introduce mechanisms to balance exploration, DT size and the statistical significance of the predictions. To mitigate the computational cost of our method, we employ a move pruning strategy that discards some branches of the search tree, leading to improved performance. The experiments show that our approach outperformed locally-optimal search in 20 out of 31 datasets, with a reduction in DT size in most of the cases.**

***Index Terms*—Decision trees, Monte Carlo tree search, Interpretability**

I. Introduction

Decision trees (DTs) are one of the most widely used tools in data mining, specially in critical domains such as medicine where predictions need to be understood. Although recent methods offer higher accuracy and stability, their output is often a black box prediction [1]. Interpretability is all the more necessary, as European regulation has been recently enacted to secure the right to an explanation of algorithmically-made decisions [2].

An optimal DT minimizes the number of decisions, while maximizing prediction accuracy. Learning an optimal DT is NP-complete, owing to the discrete and sequential nature of the splits [3]. Standard learning approaches, such as C4.5 [4] and CART [5], learn a DT by following a top-down locally-optimal strategy. They are extensively used in practice as they offer a good trade-off of interpretability, computational complexity, and prediction performance. They are not however guaranteed to yield an optimal DT. Global DT optimization methods have been proposed, but they either use multivariate test functions, which limits interpretability [6], or they impose a DT structure *a priori* [7]. Evolutionary DT learning showed performance improvements in several datasets, suggesting the potential to improve on locally-optimal search [8].

Monte Carlo tree search (MCTS) is an algorithm that learns the next best action for domains modeled as Markov decision processes (MDPs), by taking samples of the decision space [9]. An MDP is a Markovian process of *state* variables, where the probability distribution is conditioned on *actions*. The method builds a *search tree*, where each node represents a state of the MDP, and each branch represents an action. MCTS works by estimating the *value* of the state at each node, which denotes the expected long-term utility of being in that state. The value estimates are then used to guide the exploration of the search space. The MCTS algorithm has had success in problems with high branching factor, and its benefit is best realized when adapted to suit the domain at hand [10]. This includes playing Atari games such as Ms Pac-Man [11], and Total War: Rome II [12], and computer Go [13].

We propose a non-greedy DT learning approach, that uses MCTS to perform a guided exploration of the space of DTs. The method outputs the entire search tree, from which a DT can be chosen to suit the constraints of the domain. We focus on the performance of a single DT, as opposed to an ensemble of trees, targeting applications in which interpretability is crucial. Our method achieves improvement in prediction performance compared to C4.5, for 20 of the 31 evaluated datasets. The results indicate that the benefit of action pruning, and the cases where it is beneficial, needs further investigation.

II. Proposed approach

Consider the input variable $\mathbf{X}$ with dimensions X_j, $j = 1, ..., M$, and the categorical output Y. Each DT node displays a test function $t(\mathbf{x})$ that partitions the input domain. We consider univariate test functions to ensure interpretability of the output DTs. For example, if a test function $t(\mathbf{x})$ at a node concerns a numeric attribute X_j, we have :

$$t(\mathbf{x}) = \mathbf{1}(x_j < \tau), \tag{1}$$

where the $0/1$-outcome direct the instances $\mathbf{x}$ to the left or right child node. Each DT leaf contains a prediction about Y.

The learning problem is to build a DT from the training dataset $\mathcal{D}$ that predicts Y for a sample $\mathbf{x} \notin \mathcal{D}$. Locally-optimal methods build a DT top-down by selecting the test function $t(\mathbf{x})$ that maximizes an objective for the training subset at each DT node [14]. The instances are divided between the resulting nodes, which are then evaluated with respect to a stop criterion. If the stop criterion is not fulfilled, learning proceeds recursively on those nodes. C4.5 uses the *information gain* as a split criterion [4].

1) Decision tree learning as a Markov decision process: We start by modeling DT learning as the MDP where each state s represents a DT, and an action $a = \big(\ell, t(\mathbf{x})\big)$ is the univariate test $t(\mathbf{x})$ applied at DT leaf ℓ, that partitions samples between two new leaves. The reward $r(s)$ is 0 for all states, except when the state s is terminal. In this case, the state s is a finalized DT and $r(s)$ is a measure of its prediction performance.

2) Characterizing the search tree: Using this MDP, MCTS builds a *search tree* where each node n contains:

- $s(n)$: a state or a DT
- $a(n)$: the incoming action that generated n from its parent
- $U(n)$: the set of actions that have not yet been applied to $s(n)$. A new action is popped from $U(n)$ every time n is expanded.
- $N(n)$: the number of simulations started in n or one of its descendants.
- $Q(n)$: the cumulative reward of simulations started in n or its descendants.

These components are illustrated in Fig. 1. Note that *search tree* is not to be confused with the *decision tree*, present at each search tree node.

At each MCTS iteration, the search tree is traversed according to a *tree policy*, composed by the *selection* and *expansion* phases. A child selection policy is applied starting at the MCTS root node, in order to choose the most promising child. The selected MCTS node is returned if it represents a terminal state. If the node is not terminal and not fully expanded, then one action is applied to its state, returning a new node. A *simulation* or *rollout* is then run from the returned node following a *rollout policy*, obtaining an estimate of its reward. The reward is backpropagated to update the value estimates of the ancestors.

The remainder of Section II describes how MCTS is used to learn DTs, as summarized in Fig. 2. The method takes as input a *training* dataset, split into an *induction* set and a *validation* set. The former is used in the expansion phase, while the latter is used to compute the rewards in the simulation phase. The constructed DTs are evaluated on an independent *test* set, after the search is finished.

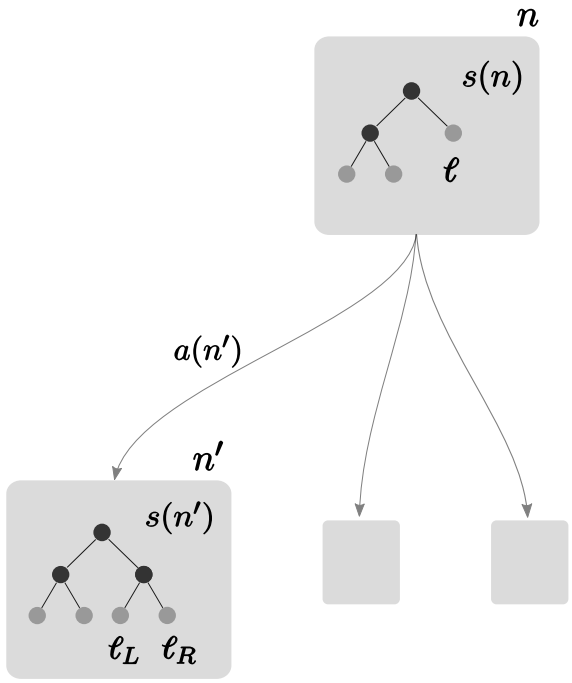

Fig. 1. Illustration of the nodes of the search tree, n and n', and their states, $s(n)$ and $s(n')$. We remark that the *search tree* is not to be confused with the *decision tree* associated to each search tree node. Executing the incoming action $a(n')$ on state $s(n)$ adds a split to leaf ℓ, resulting in two new leaves ℓ_L and ℓ_R. The depth of a search tree node corresponds to the number of upstream actions that led to it, and is equal to the number of splits of its DT. E.g. node n' has search depth equal to 3, and its state $s(n')$ has 3 internal nodes and 4 leaves.

A. Tree policy

The tree policy contains all the definitions for iteratively exploring and expanding the state space, and is often used with domain-specific enhancements. During *selection*, the value estimates are used to select the most promising MCTS node. During *expansion*, a new nodes are added to the search tree.

1) Selection: The selection policy defines how to choose the search tree nodes to be explored. In MCTS, node selection is treated as a multi-armed bandit problem where the rewards are the value estimates of each node [15].

The multi-armed bandit problem consists in choosing the best action out of a set of actions with unknown reward distributions. The player has to follow a behavior or policy that balances out the *exploration* of new actions with the *exploitation* of previous actions with known reward. In such problems, the *regret* is the loss caused by the policy not always choosing the best action. For a given policy, we are interested in finding an Upper Confidence Bound (UCB) on the probability that its regret will be lower than a given value. The UCB1 selection policy solves the exploration-

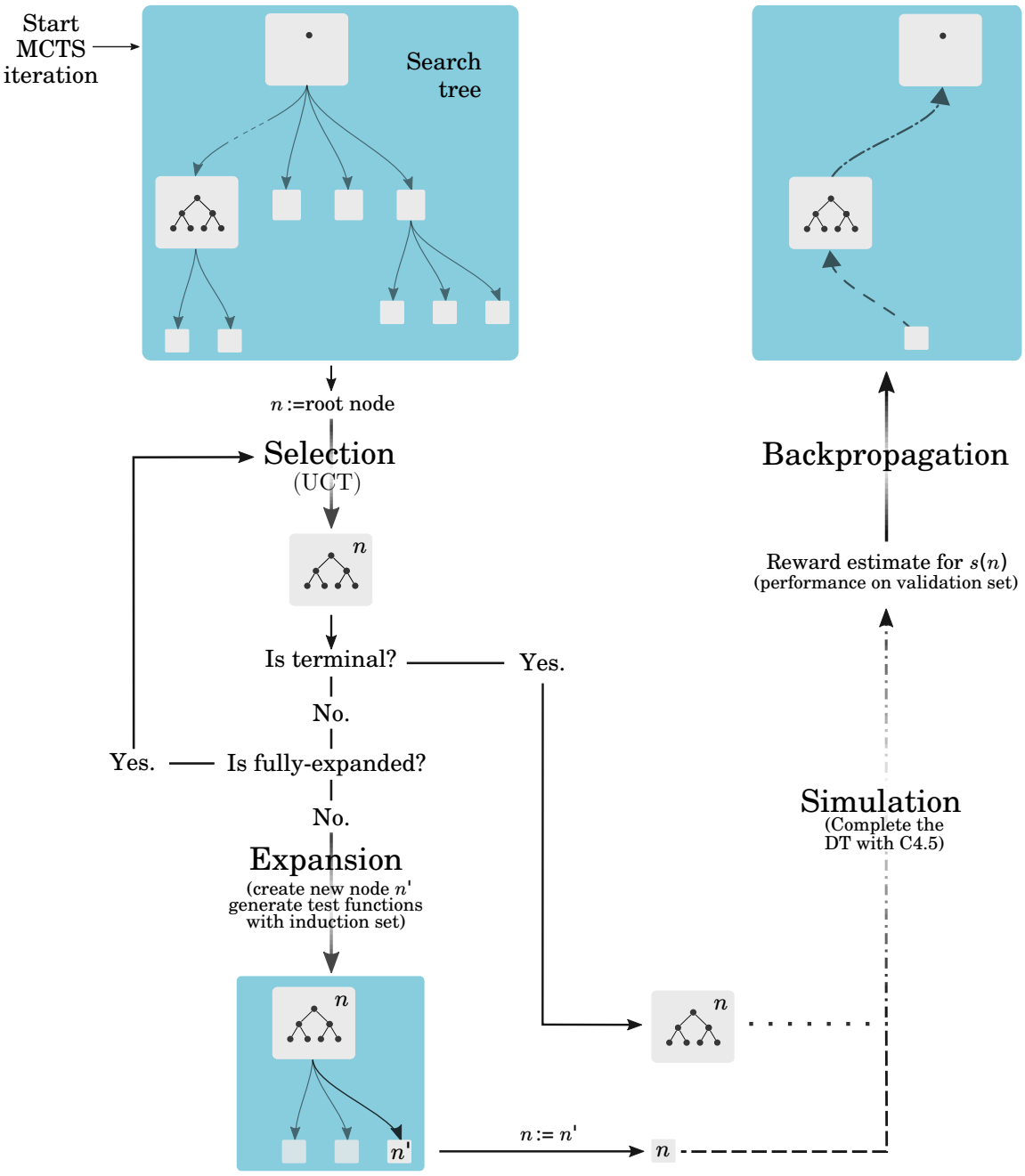

Fig. 2. Illustration of one iteration of the proposed MCTS approach.

exploitation dilemma by selecting the arm that maximizes the aforementioned upper bound [16].

MCTS employs UCB1 as a selection policy. This algorithm is known as Upper Confidence Bound for Trees (UCT) [15], and consists in choosing the child of n which maximizes:

$$\underset{n' \in c(n)}{\arg\max} \frac{Q(n')}{N(n')} + 2C_p \sqrt{\frac{\log N(n)}{N(n')}},$$

where $c(n)$ is the set of children of n and C_p is a constant. The first term $\frac{Q(n')}{N(n')}$ estimates the value of $s(n')$, promoting the exploitation of high rewards. The second term accounts for the uncertainty about less visited states, promoting their exploration. When no simulation has been started from n or one of its descendants, there are not enough statistics to employ UCB1. In this case, we move on to expand n.

2) Expansion: During *expansion*, new nodes are added to the search tree. As illustrated in Fig. 1, a selected MCTS node n is expanded by performing an action $a \in U(n)$, creating a new node n'. When applying a, two new leaves ℓ_L and ℓ_R are added to $s(n)$. If ℓ_L and ℓ_R are not terminal, candidate split functions are generated for each of them. We choose the top K functions that maximize the information gain at each leaves'

training subset. Combining each new leaf with its candidate functions yields the new set of actions, $U(n')$. K test functions are generated for each new leaf.

We impose restrictions on DT size, required to define terminal leaves and states. A DT leaf ℓ is terminal if either the number of samples in $\mathcal{D}(\ell)$ is smaller or equal to $\eta_{instances}$, or the DT branch of ℓ is deeper than the maximum depth η_{depth}. A state is terminal if all leaves of the its DT are terminal.

B. Simulation and backpropagation

The simulation policy, also called rollout policy, consists in estimating the predictive performance of the DT at the search node that is returned by the tree policy. This performance is estimated on the validation set, unseen during expansion. Since there is no random component in this policy, no more than one rollout is done for each visited MCTS node.

After the simulation, $N(n)$ is incremented by 1 and the reward estimate is added to $Q(n)$. The $Q(.)$ and $N(.)$ quantities are equally updated in the ancestors of n, in what is called the backpropagation phase.

C. Decision-tree statistical pruning

DT pruning acts as a DT regularizer. Using MCTS, we approximate the pruning method used in C4.5 with the following strategy. Each time a rollout is done, we complete the DT of the search node n returned by the tree policy, and check if the incoming action $a(n)$ still exists in the completed DT, or if it was removed by pruning. If it was removed, node n is eliminated from the search tree.

D. Search-tree move pruning

Move pruning is a technique that tackles this exponential growth of the search tree by removing suboptimal nodes, allowing more time to be employed on better choices [10]. We employ an optional move pruning approach based on the average value estimates at each node, $\frac{Q(n)}{N(n)}$. Every n_p iterations, of all the search branches with depth greater than d_p, only the path with highest average reward is kept.

E. Output

MCTS is often applied in games to find the next best move for a player, after which the search tree is discarded. In DT learning, however, there is a single start state, which is the root DT node containing the complete training set. The proposed approach runs MCTS once, and outputs the entire search tree. The performance of the DTs present at each MCTS node are then computed on the test dataset. The target application is to then choose the DT that best fits the constraints of application.

III. Experiments

We perform experiments comparing the performance of (1) the DT learned using C4.5 with its locally-optimal approach, (2) the DT built using the evolutionary approach by Kretowski and Grzes [8], (3) the best DT learned using the proposed MCTS approach, and (4) the best DT learned using the proposed MCTS approach with the move pruning method described in Section II-D. Improving performance consists in achieving better predictions on unseen data, while minimizing DT complexity, assessed by its number of leaves.

The approaches were evaluated on 23 publicly-available and 8 synthetic datasets. Sources include the UCI repository [17], KEEL repository [18], and the MIMIC-II database [19]. The synthetic datasets were generated using an adaptation of the method by Guyon [20]. Each dataset was split into a training set (70%) and a test (30%) set. For MCTS, the training dataset was further divided into two induction (70%) and validation (30%) sets. Since MCTS generates multiple DTs, the greedy and evolutionary DTs are compared with the DT obtained by MCTS with highest $F1$ on the validation set. The performance is computed on the test set.

A grid-search was done on each training set to tune the C4.5 DT pruning confidence factor through cross-validation (CV), kept constant in all experiments. We also tuned the value of the α parameter of the evolutionary approach [8], which controls the size of the output DT. The MCTS parameters are set as $C_p = 1$, $K = 3$, and $\eta_{instances} = 4$. The maximum DT depth η_{depth} is set to the maximum depth achieved by C4.5 through CV, with the selected confidence factor. When employed, move pruning is performed every $3e3$ iterations at depth $d_p = 5$. Each experiment had a budget of $2e5$ iterations.

IV. Results and discussion

The results are displayed in Table I, and complemented by Figs. 4 and 3. In Table I, we first observe that the evolutionary approach resulted in larger DTs compared to C4.5 for most of the datasets. The method led to an increased $F1$ in 12 datasets, most of which had a rise in DT size. E.g. in the *Car Evaluation* and *Solar flare* datasets, the number of leaves grew from 32 to 218 and from 59 to 121, respectively. Depending on the application, such DT sizes can hinder interpretability. We remark that most of the 12 cases that benefited from the evolutionary search were among the largest of the datasets, indicating that a more exhausting search may be appropriate, and the splits found for larger DTs are likely to be more statistically significant.

The $F1$ performance of the best DT learned by MCTS without move pruning was superior that of C4.5 in $11/31$ datasets, as depicted by the blue dots of Fig. 3(a). The method resulted in smaller DTs in the majority of the cases, as shown in Table I. When examining the 20 datasets where MCTS did worse than C4.5, we see that in many cases the maximum search depth reached by MCTS was considerably smaller than number of leaves of the C4.5 solution. Fig. 1 clarifies the relation between the search depth and the number of DT leaves. E.g. the *BHP* dataset had a greedy DT with 27 leaves, which would correspond to 26-level deep search tree. In this case, the MCTS tree only reached 9 levels, and the best MCTS solution had only 9 leaves. Similarly, C4.5 built a 33-leaf DT for the *Tic-tac-toe* dataset, while the corresponding MCTS tree was only 14 levels deep. This suggests that the search tree did not reach the necessary depth to find a good solution, assuming that the number of leaves of a good solution, or its search

TABLE I

RESULTS OF THE (1) C.5, (2) EVOLUTIONARY, (3) MCTS, AND (4) MCTS WITH MOVE PRUNING (MCTSP) APPROACHES FOR LEARNING DTS. PERFORMANCE METRICS ARE ACCURACY (ACC), F1 SCORE AVERAGED OVER ALL CLASSES (F1), AND NUMBER OF DT LEAVES (L). THE RESULT WITH HIGHEST $F1$ IS HIGHLIGHTED IN BOLD FONT. THE RESULT WITH THE SMALLEST NUMBER OF LEAVES IS UNDERLINED. WE ALSO DISPLAY THE MAXIMUM DEPTH OF THE SEARCH TREE WHEN USING MCTS (D), AND THE TOTAL MCTS RUNTIME (T) IN HH:MM. RUNTIME FOR C4.5 AND THE EVOLUTIONARY APPROACH IS NOT GIVEN BECAUSE THE METHODS ARE FAST I.E RUN IN LESS THAN A MINUTE. A DT WITH L LEAVES CORRESPONDS TO A SEARCH DEPTH OF $L-1$.

Dataset	(1) C4.5 DT			(2) Evol. DT			(3) Best MCTS DT					(4) Best MCTSp DT				
	Acc	F1	L	Acc	F1	L	Acc	F1	L	D	T	Acc	F1	L	D	T
Balance Scale	81.6	81.1	30	85.6	85.6	35	87.4	**87.4**	12	13	5:29	84.5	84.4	7	35	2:02
Banknote auth.	97.6	97.6	14	93.0	92.8	29	98.1	**98.0**	8	11	24:17	98.1	**98.0**	8	17	14:21
BHP	94.1	93.8	27	74.2	73.5	3	86.3	85.8	9	9	29:39	97.1	**96.9**	28	30	13:37
Biodegradation	81.4	77.6	13	78.5	74.0	21	82.7	**79.9**	9	11	56:53	81.7	78.5	14	19	46:52
Breast cancer	91.9	91.1	5	90.7	89.7	3	92.4	**91.9**	3	5	0:04	92.4	**91.9**	3	5	0:05
Car evaluation	89.4	73.8	32	93.8	**83.8**	218	82.9	71.0	12	14	17:48	88.3	78.1	19	42	5:00
Contraceptive	48.3	45.8	43	55.1	49.5	57	52.4	51.6	8	10	30:22	56.7	**53.4**	28	53	36:56
Credit approval	85.5	85.1	6	87.0	**86.9**	17	80.7	80.6	9	12	25:46	81.6	81.3	10	18	24:23
Solar flare	71.5	58.9	59	72.4	**63.0**	121	70.7	58.8	11	13	21:00	74.2	62.3	40	55	34:26
German credit	75.4	**65.7**	20	72.7	61.8	81	69.9	63.4	12	13	27:58	73.4	63.9	22	31	15:00
Indian liver	71.0	**62.7**	24	64.2	57.2	321	68.8	60.9	8	10	17:58	61.4	58.5	21	40	11:47
Arterial catheter	87.8	87.7	6	88.0	**87.9**	26	86.9	86.8	7	12	29:08	87.1	87.0	5	13	13:59
Language	78.3	60.7	8	79.1	57.8	9	75.1	**61.8**	2	7	2:04	75.1	**61.8**	2	7	3:19
Localization	96.3	96.3	23	96.5	96.5	13	96.8	**96.8**	9	12	17:50	96.5	96.5	17	29	30:44
Mammographic	82.0	**82.0**	11	78.5	78.5	22	80.0	80.0	10	12	3:20	80.3	80.3	9	12	2:11
Diabetic	66.5	66.2	11	67.6	**67.6**	7	65.9	65.9	10	12	86:16	66.8	66.6	18	24	56:01
Phishing	86.3	83.4	34	89.9	**89.7**	96	84.3	76.2	12	12	15:30	86.1	81.3	21	32	14:35
Pima indians	74.0	70.7	24	74.5	68.9	3	77.5	74.0	8	11	17:35	80.1	**77.4**	17	28	14:36
Student	75.0	**74.6**	12	75.0	73.5	9	74.0	72.5	9	12	18:37	69.4	68.9	15	26	26:33
Synthetic 1	70.8	70.9	24	60.7	59.8	29	67.3	67.2	9	10	64:25	73.5	**73.7**	20	29	85:27
Synthetic 2	70.1	**70.0**	27	62.6	61.8	43	63.0	62.7	10	10	59:09	63.0	62.7	10	10	60:33
Synthetic 3	81.1	80.3	21	77.6	76.6	13	74.5	74.0	8	9	38:22	81.6	**80.9**	26	28	31:08
Synthetic 4	79.6	72.8	3	74.0	60.5	9	77.4	**73.1**	5	7	0:05	77.4	**73.1**	5	7	0:10
Synthetic 5	48.4	48.3	78	46.9	46.9	15	47.3	44.1	8	10	78:53	55.1	**55.3**	50	85	105:13
Synthetic 6	51.2	49.8	22	50.2	49.8	17	52.1	50.5	8	11	72:48	51.2	**51.1**	26	37	87:37
Synthetic 7	57.1	56.4	43	64.8	**58.6**	23	55.6	52.1	10	11	70:54	55.6	52.1	10	11	57:22
Synthetic 8	61.9	**55.6**	29	59.1	52.2	95	66.3	53.9	10	12	23:36	63.0	55.1	32	38	6:47
Tic-tac-toe	91.7	90.3	33	78.1	74.8	193	82.6	77.5	13	14	20:25	93.8	**92.9**	30	49	11:44
Titanic	78.7	70.0	8	78.1	**71.9**	3	75.6	71.3	8	9	0:15	75.6	71.3	5	9	0:17
Wine quality	70.5	70.5	107	71.5	**71.3**	117	70.5	69.6	9	9	42:10	70.5	69.6	9	9	46:06
Yeast	67.5	66.4	9	66.0	65.8	23	65.3	65.2	8	12	17:27	68.3	**68.2**	9	20	14:47

complexity, is expected to be in the same order of magnitude as the number of leaves of locally-optimal solution.

A deeper search was achieved by running MCTS with move pruning, as depicted in Fig. 4(b). This approach boosted the $F1$ performance of MCTS for 18 of the datasets, as shown in Fig. 3(c). In total, MCTS with move pruning outperformed C4.5 in 20 of the datasets. In most of those cases, the output DT was actually smaller compared to C4.5. E.g. the *Synthetic 5* dataset had a greedy solution with 78 leaves, while the MCTS solution had 50 leaves. This indicates that increasing the search depth by move pruning effectively was able to remove sub-optimal states, allowed more resources to be spent on the exploration of good states.

In the datasets where MCTS with move pruning outperformed C4.5 in terms of $F1$ with an increase in DT size were the *BHP*, *Diabetic*, *Synthetic 3*, *Synthetic 4*, *Synthetic 6* and *Yeast* datasets. However, the increases in number of leaves were moderate, leading to DTs that can still be interpretable. This is contrary to the evolutionary approach, where the performance improvements were mostly achieved by significantly increasing DT size. For example, MCTS was able to outperform C4.5 in the *Solar flare* dataset with 40 leaves, while the evolutionary DT has 121 leaves.

V. CONCLUSIONS

This manuscript describes a novel approach for learning DTs using MCTS. We evaluated the hypothesis that better prediction performance can be achieved compared to locally-optimal search, by performing a MCTS in the space of DTs. Our hypothesis was validated by achieving DTs with higher $F1$ in 20 out of 31 datasets, the majority of which had reduced DT size. We also evaluated an evolutionary DT learning approach which outperformed locally-optimal search in 12 of the datasets, but at the expense of building larger DTs.

Datasets of complex decision problems, for which finding the optimal DT is likely to require a more complex search, may require a greater MCTS depth. The characterization of such datasets is a topic to be extended in further research. For those cases, move pruning may be necessary to for MCTS to outperform greedy search. Move pruning is however

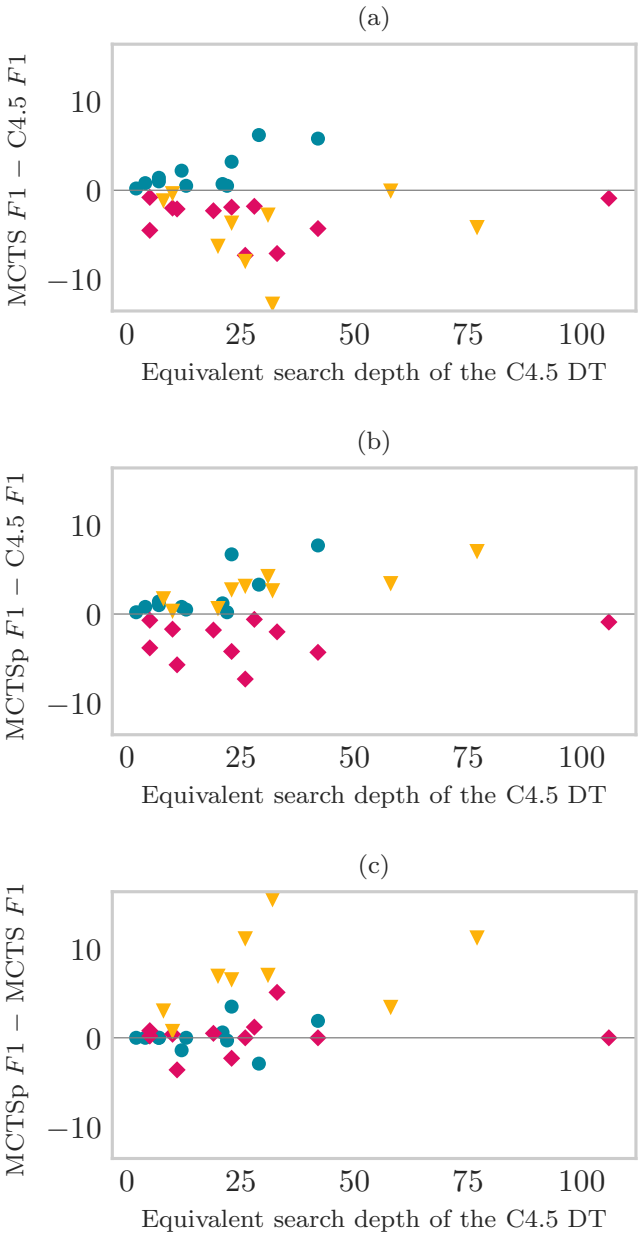

Fig. 3. Difference in average $F1$ between (a) C4.5 and MCTS, (b) C4.5 and MCTS with move pruning (MCTSp), and (c) MCTS and MCTSp. Each point represents a dataset. Blue dots are the cases where MCTS did better than C4.5. Yellow triangles are the datasets where MCTS did not improve C4.5, but MCTSp did. Rose diamonds correspond to the datasets where neither MCTS or MCTSp outperformed C4.5.

recommended for any dataset, as it allows deeper states to be reached faster. The search can thus be quickened by using a smaller number of iterations.

Future work directions include a comprehensive analysis of the parameters of the algorithm. In particular, the employment of better move pruning strategies would allow a reduction in the algorithm runtime, as there is a high degree of redundancy in the output DTs. It would also facilitate scaling up to larger datasets and more complex domains. The proposed

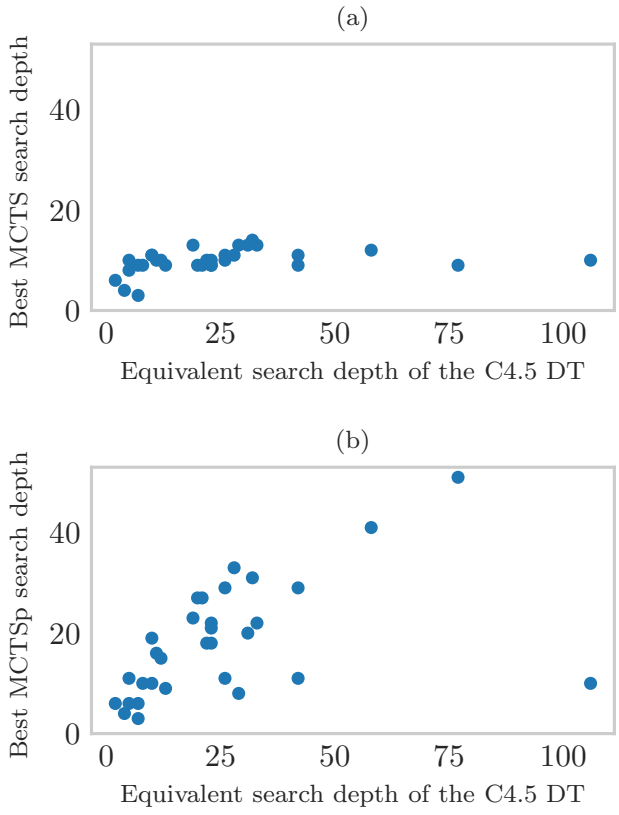

Fig. 4. Search-tree depth of the node with the best DT build by (a) MCTS and (b) MCTS with move pruning (MCTSp). Each point represents a dataset.

approach is general and can be employed with other functions to generate actions, or other estimators of DT value. The maximum depth reached for a given number of iterations depends on the rewards seen by the selection algorithm. Using other simulation policies may therefore improve the prediction of how good a DT split is considering potential interactions with other variables, a central problem in DT learning.

Finally, the method needs to be used with a post-processing methodology to select the DT that best meets the constraints of the domain, including missing data and varying feature costs.

Acknowledgments

This work was supported by the European Union Horizon 2020 research and innovation programme (grant 642676 - Cardiofunxion), and by the Spanish Ministry of Economy and Competitiveness (grant TIN2014-52923-R; Maria de Maeztu Units of Excellence Programme - MDM-2015-0502). Acknowledgments to Dr. Bart Bijnens for the discussions that motivated and improved the study.

References

[1] S. F. Weng, J. Reps, J. Kai, J. M. Garibaldi, and N. Qureshi, "Can machine-learning improve cardiovascular risk prediction?" *PLOS ONE*, vol. 12, no. 4, 2017.

[2] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016, repealing Directive 95/46/EC," 2016 O.J. L 119, 4.5.:1–88.

[3] 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.

[4] R. Quinlan, *C4.5: Programs for Machine Learning*. CA: Morgan Kaufmann, 1993.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.

[6] M. Norouzi, M. D. Collins, M. Johnson, D. J. Fleet, and P. Kohli, "Efficient non-greedy optimization of decision trees," *Nips*, pp. 1–9, 2015.

[7] K. Bennett, "Global tree optimization: A non-greedy decision tree algorithm," in *Computing Science and Statistics*, 1994, pp. 156–160.

[8] M. Kretowski and M. Grzes, "Global learning of decision trees by an evolutionary algorithm," *Information Processing and Security Systems*, pp. 401–410, 2005.

[9] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[10] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[11] X. Gan, Y. Bao, and Z. Han, "Real-time search method in nondeterministic game—ms. pac-man," *ICGA Journal*, vol. 34, no. 4, pp. 209–222, 2011.

[12] A. J. Champandard, "Monte-carlo tree search in total war: Rome ii's campaign ai," AIGameDev.com, 2014. [Online]. Available: http://aigamedev. com/open/coverage/mcts-rome-ii

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

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

[15] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.

[16] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[17] M. Lichman, "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[18] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2011.

[19] J. Lee, D. J. Scott, M. Villarroel, G. D. Clifford, M. Saeed, and R. G. Mark, "Open-access mimic-ii database for intensive care research," in *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*. IEEE, 2011, pp. 8315–8318.

[20] I. Guyon, "Design of experiments for the nips 2003 variable selection benchmark," 2003. [Online]. Available: clopinet.com/isabelle/Projects/NIPS2003