INTERPRETABLE REINFORCEMENT LEARNING VIA EVOLUTIONARY DECISION TREES

Vinícius Garcia Silva da Costa

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Carlos Eduardo Pedreira

Rio de Janeiro
Maio de 2025

INTERPRETABLE REINFORCEMENT LEARNING VIA EVOLUTIONARY DECISION TREES

Vinícius Garcia Silva da Costa

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Carlos Eduardo Pedreira

Aprovada por: Prof. Carlos Eduardo Pedreira
Prof. Gerson Zaverucha
Prof. Sancho Salcedo Sanz
Prof. Frederico Gadelha Guimarães
Prof. Amit Bhaya

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2025

Costa, Vinícius Garcia Silva da

INTERPRETABLE REINFORCEMENT LEARNING VIA EVOLUTIONARY DECISION TREES/Vinícius Garcia Silva da Costa. – Rio de Janeiro: UFRJ/COPPE, 2025.

XI, 104 p.: il.; 29, 7cm.

Orientador: Carlos Eduardo Pedreira

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2025.

Referências Bibliográficas: p. 97 – 102.

1. Árvores de decisão. 2. Aprendizado por reforço. 3. Interpretabilidade. I. Pedreira, Carlos Eduardo. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Acknowledgments

First and foremost, I would like to thank my parents, Patrícia and Francisco, for the unconditional support throughout my whole life. It would be an understatement to say that I would not be where I am today without you two; this thesis is as much your achievement as it is mine.

To my wife, Silvia, thank you for being there during every step of the journey. After years of reviewing drafts and hearing me talk for hours about decision trees, I'm confident you could present this thesis as well as I could. Thank you for your unwavering patience and support, and for being such an integral part of my life.

To my advisor, Carlos Pedreira, thank you for your guidance and support throughout these years. I will be forever grateful to you for believing in me since the beginning, and for offering me so many wonderful opportunities; without them this doctorate would not even exist, and they have shaped my life for the better.

To Sancho Salcedo Sanz and Silvia Jiménez Fernández, thank you for receiving me in UAH, and for supporting and teaching me so much throughout my stay. You have built something very special in GHEODE, and I'm profoundly grateful to have been part of it for a while. To Carolina Marcelino, I could not thank you enough for making the entire experience possible in the first place.

To all the wonderful people I met in Alcalá – Cosmin, Eugenio, Alberto, Dušan, Luismi, Francesca, Jorge, Alex, Bogdan, Cesar, Adrián, Andrei, David, Daniel, even Simo –, thank you for your friendship and for our conversations. You have made this Brazilian kid feel very welcome, and he will forever carry in his heart the time he spent with you.

To my colleagues from PESC – Amanda Camacho and Arthur Santana – thank you for your friendship and support. Our conversations made the pandemic much more bearable, which was no easy feat, and lightened the days I spent in UFRJ. To Gabriel Matos, I'm deeply grateful for welcoming me in Alcalá, for the light-hearted talks, and for the various discussions and insights which have undoubtedly shaped this thesis.

To my friends Bruno Ogava, Diego Jesus, Horácio Macêdo, Pedro Marques and Rafael Katopodis, thanks for hearing me out when I needed to vent, and helping to put everything into perspective.

To professors Gerson Zaverucha, Sancho Salcedo Sanz, Frederico Gadelha Guimarães and Amit Bhaya, thank you for accepting to take part in the examination committee.

I'm grateful for all the financial support received from the CAPES research agency. I would not be here without it.

Finally, I would like to offer a special thanks to professor Mario Benevides, who in 2019 convinced me to get a Master's degree by assuaging my fears and saying that "*worst case scenario, you're going to be a 2-years-older kid with a Master's*". It took a little bit longer than that and I didn't get a Master's, but I have no regrets.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

INTERPRETABLE REINFORCEMENT LEARNING VIA EVOLUTIONARY DECISION TREES

Vinícius Garcia Silva da Costa

Maio/2025

Orientador: Carlos Eduardo Pedreira

Programa: Engenharia de Sistemas e Computação

Recentemente, algoritmos de *Reinforcement Learning* (RL) foram aplicados com sucesso em uma vasta gama de tarefas, desde direção autônoma a tratamentos médicos dinâmicos. No entanto, a maioria dos modelos bem-sucedidos neste campo são construídos com redes neurais profundas, e portanto possuem um baixo nível de interpretabilidade que limita suas aplicações no mundo real. Nesta tese, nós abordamos esse problema substituindo as redes neurais profundas por um modelo de aprendizado de máquina altamente interpretável: árvores de decisão. Primeiramente, nós conduzimos uma revisão exaustiva dos últimos 10 anos de pesquisa em árvores, preenchendo uma lacuna na literatura ao conectar linhas de pesquisa isoladas sob uma perspectiva única e determinar o estado-da-arte destes modelos. Segundamente, nós separamos os Algoritmos Evolutivos como a abordagem de otimização a ser utilizada, e propomos uma codificação matricial inovadora que reduz em até 20 vezes o custo computacional de evoluir árvores de decisão para problemas de classificação. Em seguida, nós partimos para o campo de Reinforcement Learning ao propormos um algoritmo capaz de evoluir, com sucesso, árvores de decisão para problemas de RL; este algoritmo é a peça-chave da nossa tese. Sua essência está no uso de técnicas de *Imitation Learning* para a inicialização do processo evolutivo, assim como no uso de um método de poda inovador chamado *Reward Pruning*, capaz de reduzir o tamanho das árvores sem comprometer seu desempenho. A abordagem como um todo foi testada em três *benchmarks* populares de RL, assim como numa tarefa de fertilização de colheita baseada em um problema do mundo real. Até onde sabemos, esta abordagem é a primeira a resolver o *benchmark Lunar Lander* mantendo tanto interpretabilidade quanto alto desempenho (90% de taxa de sucesso), além de ser a primeiro a resolver o *benchmark Mountain Car* com apenas 7 nós

(aproximadamente metade do melhor resultado anteriormente reportado na literatura), e na tarefa de fertilização inspirada no mundo real, é a primeira a produzir resultados interpretáveis enquanto alcança o mesmo desempenho que redes neurais profundas. Analisamos as melhores soluções mais a fundo e demonstramos que não apenas elas são interpretáveis, como também são diversas, o que empodera o usuário final ao fornecer a ele a escolha de qual modelo melhor se encaixa nos seus critérios e requisitos próprios. De maneira geral, os resultados desta tese avançam o estado-da-arte de RL interpretável, abrindo caminho para o uso de RL em domínios onde confiança e segurança são preocupações cruciais.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

INTERPRETABLE REINFORCEMENT LEARNING VIA EVOLUTIONARY DECISION TREES

Vinícius Garcia Silva da Costa

May/2025

Advisor: Carlos Eduardo Pedreira

Department: Systems Engineering and Computer Science

Recently, Reinforcement Learning (RL) algorithms have been successfully applied to a wide range of tasks, from autonomous driving to dynamic medical treatments. However, most successful models in this domain are built using deep neural networks, which gives them a level of uninterpretability that limits their real-world applications. In this thesis, we address this problem by replacing the complex neural networks with an interpretable learning model: Decision Trees (DTs). First, we conduct a comprehensive survey of the last 10 years of DT research, filling a gap in the literature by determining the current state-of-the-art on these models and connecting their different lines of research under a unified perspective. Second, we decide upon Evolutionary Algorithms as the optimization approach to be used, and propose a novel matrix-based encoding for Evolutionary DTs that reduces the computational cost of their training on classification tasks by up to 20 times compared to the traditional encoding. Third, we move onto the Reinforcement Learning arena by proposing a framework capable of successfully evolving DTs for RL tasks, which serves as the centerpiece of this thesis. The key idea behind this framework is to warm-start the evolutionary process by employing Imitation Learning techniques (in particular, the DAgger algorithm) and a novel pruning method that we propose called Reward Pruning, which reduces tree size without compromising on performance. The overall approach is tested on three popular benchmarks environments from the OpenAI Gym library and on a crop fertilization task inspired by real-world constraints; to the best of our knowledge, this approach is the first to solve the Lunar Lander benchmark with both interpretability and high confidence (90% of success rate), the first to solve the Mountain Car benchmark with only 7 nodes, and on the real-world fertilization environment, it is the first to be able to

achieve the same performance as the best deep neural networks while having the added bonus of interpretability. We further analyze the best solutions generated by our approach and demonstrate that they are not only interpretable, but also diverse, which empowers the end-user with the ability to choose the model that best suits their requirements. Overall, the results in this thesis push the state-of-the-art in Interpretable RL, facilitating the usage of RL in domains where trust and security are key concerns.

Contents

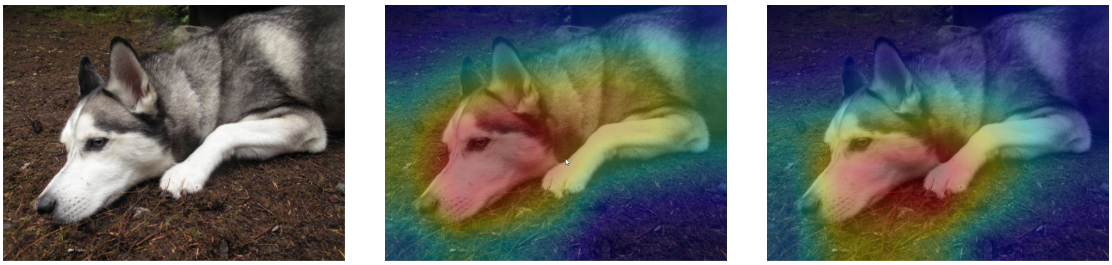
List of Figures

Chapter 1

Introduction

At the time of this writing, the world is experiencing an "*Artificial Intelligence Revolution*" [2, 3]. Although research in the field dates back to the 1950s [4], only in the past decade has AI risen to its current position of true cultural phenomenon, capable of creating and disrupting entire industries in short periods of time. Owing to several technological advancements and some material conditions (such as the emergence of Big Data and powerful graphics cards), AI can now generate high-quality images based on text descriptions [5, 6], produce complex and comprehensive answers to question prompts [7, 8], synthesize natural-sounding speech from text [9], accurately transcribe text from speech [10], beat humans at a wide range of board and video games [11–13], drive cars autonomously [14, 15], among many other successes that seemed impossible only a few decades ago.

Successes like these provide an unmistakable affirmative answer to the age-old question of "*Can we make AI that works?*". Recent years, however, have seen the rise of a second hard question: "How *do these AI models work?*". It may seem strange that such a question needs to be asked in the first place – after all, how could we have built such successful models without explicitly knowing how they work? –, but because of the way most modern AI is constructed, the answer to this question is anything but straightforward. A large parcel of state-of-the-art AI is based on Deep Neural Networks (DNNs, [16]), which are complex models composed of dozens of interconnected layers and millions or even billions of parameters (for instance, OpenAI's GPT-3 has 175 billion parameters [7]). It is possible for us to understand the algorithms that assign these parameters, and it is even feasible to examine the particular values each parameter takes, but the overall scale of these models is so large that it is humanly impossible to understand what these parameters are *doing as a whole*; in other words, the models are bonafide "black-boxes". Such a model can learn to distinguish dog breeds in a photo, and it can even learn to do it perfectly, but if we pick a photo and ask why did the model made a certain choice, there is no answer – and inspecting its guts reveals no further information.



(a) Test image, containing a Siberian husky

(b) Evidence for image being of a Siberian husky

(c) Evidence for image being of a transverse flute

Figure 1.1: Representation of saliency maps produced for picture (a), in a case where the prediction is correct (b), and in a case where the prediction is incorrect (c). Images reproduced from [1].

Of course, this is not necessarily a problem: there are several domains where successfully deploying a model does not require us to understand its reasoning (e.g. generating images for a newspaper, synthesizing speech for an audiobook, etc); in such cases, it does not matter how the output was produced, only whether it is good or not. But in many other domains (e.g. applications in healthcare, finance, civil and criminal law, smart cities, etc), it is so essential to understand why (and how) the models make their decisions that the absence of such justifications can compromise any intention to use the model in real-world scenarios. Indeed, under the European Union's General Data Protection Regulation (GDPR, [17]), systems that affect citizen's lives must provide some level of interpretability according to the so-called "right to explanation" principle (even though its details have been a source of heated discussion [18, 19]).

In trying to solve this problem, the literature has focused on two distinct approaches: *explaining* and *interpreting*[1]. In Explainable AI, the goal is to take a black-box model, use it to make predictions, and then use external techniques to try to *explain* why the model has made such predictions. One example would be to take an input and then apply tiny tweaks to it until the prediction made by the black-box changes; this would give us some level of insight into the question "why did the model make this prediction instead of a different one?" [22, 23]. Another popular example are the saliency maps of Convolutional Neural Networks (see Figure 1.1): a technique which uses the network weights to highlight the regions of the input image that the model paid attention to while making its prediction [24].

Interpretable AI, on the other hand, avoids the use of black-boxes altogether, favoring the usage of transparent models that can be inspected and *interpreted*

[1]We note that this terminology, although used by some authors [1, 20], is not universally employed; some authors instead call the distinction "intrinsic interpretability" vs. "post-hoc interpretability", others call it "transparency" vs. "post-hoc interpretability", and others even draw no distinction at all. For a more in-depth discussion on terminology, we refer to [21].

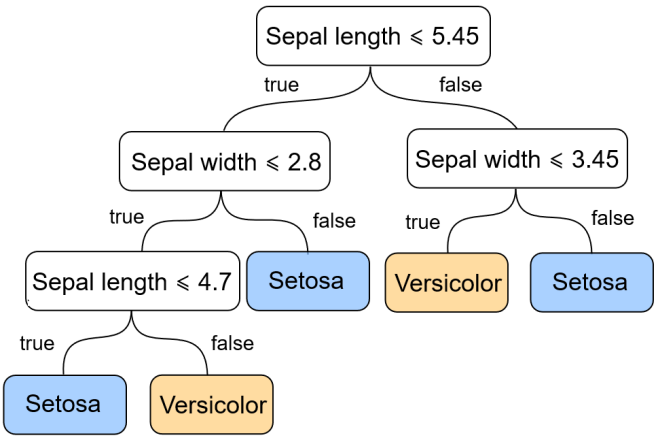


Figure 1.2: A decision tree for a simplified version of the Iris dataset, in which the goal is to use a flower's characteristics (petals and sepals) to determine its species (*Iris setosa* or *Iris versicolor*).

without the need of any external tool – the so-called "white-boxes". A simple example would be the Decision Rules model [25, 26], which makes predictions via a list of IF-THEN statements that is not too dissimilar from a basic computer program; for instance, "IF age $>$ 60 AND temperature $>$ 41 THEN sick ELSE healthy". No external tool is needed to explain such a model, because it is its own explanation. A more popular and sophisticated interpretable approach would be the Decision Tree (DT) family of models ([27, 28], see Figure 1.2), which organizes such IF-THEN statements into a hierarchical, flowchart-like structure, making it easier to add more conditions while preserving a sense of order.

There has been some debate about whether interpretability or explainability is preferable [1, 29]. At its best, the explainable approach allows us to use the already successful black-boxes while giving us valuable insights into how they work. At its worst, these insights are so incomplete as to be useless (e.g., looking at the saliency map of an incorrect prediction gives us no clue as to why the error was made, as shown in Figure 1.1) or even misleading (e.g. the explanation of a decision made by a black-box is not necessarily consistent with the actual "reasoning" followed by the black-box, so it is possible to take an unfair model and explain away each of its individual predictions using fair justifications, a process that has been called *fairwashing* [30]). Interpretable models avoid all of these problems, but they also come with their own major drawback: creating models that are both interpretable and high-performing is harder than creating models that are just high-performing, and in fact it is sometimes not even clear if there *exists* a white-box model that can match the performance of a particular black-box.

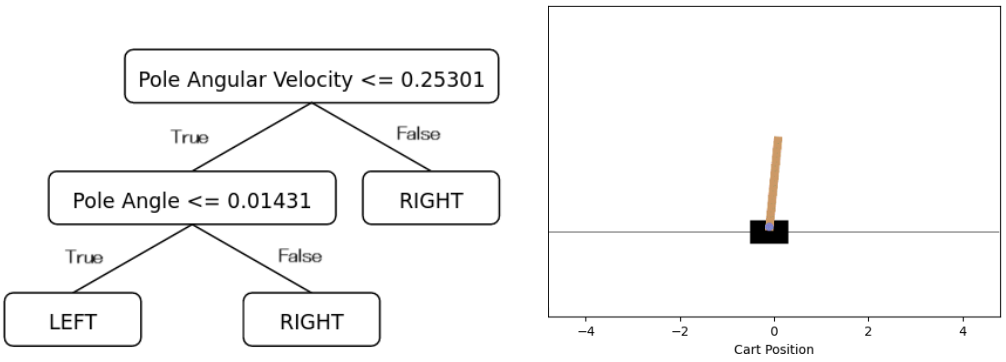
It is in this general context that this thesis is situated. Our overall goal is to push the state-of-the-art of interpretable models in a particular subfield of AI, called Reinforcement Learning (RL). The application of interpretability to RL has

been identified as one of the major challenges for interpretable AI as a whole [31], and there have been several recent works and surveys dedicated to it [21, 32–34]. Furthermore, we believe we have also provided other important contributions to the field of interpretable AI as a whole.

In RL, the goal is to create an intelligent agent that observes the current state of the world, processes it, and decides the best possible action to take according to a previously defined goal. The classic example is a game like chess, where the RL agent is a player who considers the current state of the board and decides which piece to move in order to maximize its chances of winning. Naturally, the chosen move generates a response from the other player (be they human or machine) and results in a new game state that must be similarly evaluated to produce a new move – a process that repeats all the way to the end of the game, when the agent receives a numerical feedback indicating how well it did (e.g. +1 if the agent won, -1 if it lost, and 0 if it tied). The RL framework fits well with games of this kind, but it can be applied to basically any problem where one can define a state that evolves over time, a set of actions that the agent can take to affect that state evolution, and a numerical feedback that can guide the agent towards the desired behavior. Indeed, the framework has found great success in diverse domains such as robotics [35], resource management [36], recommendation systems [37], dynamic medical treatments [38], and even language models like the aforementioned GPT-3 [8].

As is common with AI successes in recent years, most of these RL applications use black-box models such as DNNs, which effectively renders the resulting agents uninterpretable. In addition to the disadvantages of uninterpretable AI that were previously discussed, uninterpretable RL has its own share of specific disadvantages. First, it is difficult or even impossible to guarantee that the model will not perform undesirable and potentially harmful actions, since its "decision making process" is opaque to humans [31]. Second, uninterpretability complicates troubleshooting, since it is difficult to understand why the model chose one action over another [31, 32]. Finally, in high-stakes domains where safety is critical (such as healthcare [38, 39] and autonomous driving [40]), the opaqueness of DNN-based RL approaches can create challenges to their real-world deployment, be they legislative (such as GDPR's "right to explanation") or organizational (certain stakeholders might be resistant to employ systems with rules that they cannot explicitly audit).

An intuitive way to introduce interpretability into RL is to use existing RL algorithms but replace the opaque DNNs with interpretable models, such as DTs (Figure 1.3 shows how DTs can be used to represent agents in RL tasks). But while this replacement may sound intuitive, there is no obvious way to carry it out: traditional DT algorithms operate in a batch framework that requires the entire dataset to be available at once, while RL algorithms typically operate on an online,



(a) Tree agent (b) RL environment

Figure 1.3: An example of RL agent representation with DTs. In this benchmark environment (*Cartpole*), the goal is to keep the pole balanced upright by moving the cart left and right; if the pole falls down, the agent fails. Every timestep the agent finds the leaf which corresponds to the current state of the environment, and takes the action defined by it (LEFT indicates that the cart should move left, while RIGHT means the opposite).

sample-by-sample basis (i.e. the agent can only access the next state after it has decided how to act in the current state, making it impossible for the dataset to be available since the start). Although this gap can be bridged in intuitive ways, other complications soon appear, such as how to guarantee that the tree is large enough to account for all relevant possibilities, while at the same time making the tree small enough to still be interpretable by humans (after all, a tree with thousands of nodes is arguably as uninterpretable as a DNN).

The conflict inherent to this integration has been tackled in the literature through a few different approaches. The earliest one is the *greedy* approach, which takes inspiration from traditional DT algorithms (like CART [41] and C4.5 [42]) and strives, like them, to build trees iteratively from scratch. All works from this line of research [43–46] follow the same general idea: a tree agent is executed on the environment, and each one of its leaves store a history of what happened after that leaf was visited (i.e. a history of rewards, of value updates, etc). When that history fulfills a certain criterion (e.g. when it reaches a significantly high variance), the leaf turns into an inner node and splits into two other leaves; the process is repeated with this new tree. Although there are important differences between algorithms in the greedy approach, they all neglect the interpretability aspect: since these methods only include procedures to grow the trees, the final model is bound to be much larger than necessary, which runs counter to the goals of Interpretable RL.

Other works take the *gradient-based* route. By drawing parallels between the graph-like structures of DTs and neural networks, this line of research aims to combine the best attributes of both, creating models that are not only easy to interpret (like DTs) but also easy to train (like neural networks). Although this hybridization

started out in supervised learning [47], two works have used this approach for RL [48, 49]. In both cases, the final results are difficult to interpret: Liu et al. [48] proposed a framework in which the tree leaves contain not actions (like in Figure 1.3), but linear models that predict the values of each state-action pair. This in theory improves performance since the tree model has a higher representation capacity, but at a heavy cost to interpretability, since by looking at a tree leaf it is not at all clear which action should be taken. In contrast, Silva et al. [50] adapted not the tree's leaves, but its inner nodes: each one of them now contains a linear combination of every attribute, instead of a single attribute (as in traditional trees like the one in Figure 1.3). This enables gradient-based optimization, but at the same time turns the tree's inner nodes into the uninterpretable nodes of a neural network. The authors proposed a way to convert this new tree back into its traditional interpretable format, however it resulted in critical loss of performance; it is safe to assume that better solutions may be found if interpretable solutions are seeked from the start.

Finally, the *evolutionary* approach trains DTs for RL by using Evolutionary Algorithms (EAs, [51]). From a general perspective, these algorithms are metaheuristics that draw inspiration from the Darwinian theory of natural selection in order to find good solutions for optimization problems. This approach is a natural fit for this kind of problem because while it is quite simple to determine how well a DT performs on a given RL task (i.e. to calculate its fitness), usually it is not clear at all how one should modify said tree in order to increase its performance (i.e. to perform a greedy step). This means that given the goal of producing trees for RL that are both interpretable and highly-performing, one cannot do much better than randomly modifying the trees until high-quality solutions emerge, which is at the core of the evolutionary approach.

To the best of our knowledge, there have been two works in this line of research so far. Dhebar et al. [52] used EAs to evolve a structure known as non-linear DTs, which are trees where the inner splits contain non-linearities (for example $w_1 x_1^{-3} + w_2 x_2 x_1 - w_3 \leq 0$). This has the positive impact of increasing the representation capacity of the DT model, but at the same time results in uninterpretable trees, since these formulas are much harder to understand than the traditional univariate splits. Custode et al. [53] avoided this issue by keeping the traditional tree structure unchanged; in their work, a hybrid approach was proposed in which the inner splits are defined by an EA, while the leaves are determined by traditional RL algorithms. Although in theory this separation of duties accelerates training, it can also bias the results towards trees that are larger than necessary, since it has been previously shown in the literature that the tree structure needed to obtain an optimal policy is sometimes larger than the tree structure needed to represent it [44]. All in all, these two works have crucial issues that can be improved.

1.1 Thesis Statement

Given this broad context, our thesis introduces a new framework for achieving Interpretable RL through DTs. This framework arose after a sequential process of investigation that can be divided into three steps:

1. Research on DT spans more than six decades, but the most popular models are still those that were published in the 80s. What is the current state-of-the-art in DT research? Can we exploit any of these novel developments for RL?

2. Evolutionary DTs appear to be indeed the better approach for RL problems, however they have a large computational cost. Can we reduce this cost by exploring other data structures, or perhaps by adopting parallelism?

3. It is quite simple to adapt Evolutionary DTs for RL tasks. Does the naive implementation work? How can we improve upon this implementation? Is there something we can exploit from those uninterpretable models we rejected?

Concisely, our thesis can be summarized in the following central statement: "*How can we create interpretable and high-performing Decision Trees for Reinforcement Learning?*".

In trying to answer question 1, we noticed that there was no clear summary of the state-of-the-art of DTs, since the field lacked a wide-ranging survey of the works published in its last 10 years (despite this decade being the most prolific one). In an attempt to fill this gap, we published a survey paper at the *Artificial Intelligence Review* journal (impact factor 12.0), which already has 150 citations by other authors in JCR journals.

Concerning question 2, we have developed a novel matrix-based encoding for DTs that allows their entire training process to be represented as a series of matrix operations. By exploiting the computational efficiency of numerical computation and the ease with which matrix operations can be parallelized, the proposed technique speeds up the computational cost of evolving DTs in up to 20 times on supervised learning problems, greatly alleviating one of the key disadvantages of the evolutionary tree approach [54]. Although it does not provide the same speedups for RL tasks (since the bottleneck there is usually environment emulation), it represents an important innovation for evolutionary DT learning as a whole. This work has been published at the *Applied Soft Computing* journal (impact factor 8.7).

Finally, regarding question 3, we propose a novel framework that does not sacrifice interpretability nor performance while evolving DTs for RL. To achieve this, we use techniques from the Imitation Learning literature – alongside a novel procedure we propose called Reward Pruning – to generate a pool of decent solutions. Then,

these solutions are passed onto an evolutionary process (in our implementation, the Coral Reef Optimization metaheuristic [55, 56]) to further optimize and fine-tune them; the final trees are both interpretable and highly-performing. We tested our approach on three popular benchmarks from the OpenAI Gym library and on a crop fertilization task inspired by real-world maize management. To the best of our knowledge, this framework is the first to solve the Lunar Lander benchmark with interpretability and consistency (90% of success rate), the first to solve the Mountain Car benchmark with only 7 nodes, and on the real-world fertilization problem, it is the first interpretable model to achieve the same performance as the best DNNs. Further analyses show that the best solutions produced by our framework possess not only interpretability but also diversity, therefore empowering the user with the ability to pick which model they feel best captures their intended and expected behavior. This work has been published at the *Artificial Intelligence* journal (impact factor 14.4).

The following chapters present these contributions in detail through the published papers themselves, with interstitial sections offering context and connecting them. They are structured as follows: Chapter 2 contains "Recent advances in Decision Trees: an updated survey", Chapter 3 includes the second contribution, "Efficient Evolution of Decision Trees via Fully Matrix-based Fitness Evaluation", and Chapter 4 contains "Evolving Decision Trees for Interpretable Reinforcement Learning". Finally, Chapter 5 concludes the thesis with a discussion of limitations and possible future directions for the research.

Chapter 2

Recent advances in Decision Trees: an updated survey

As previously established, the overarching goal of this thesis is to achieve high-quality models for RL that are able to both perform well and be humanly understandable. To achieve this, many different approaches could have been taken, but we have opted to use Decision Trees (DTs): popular machine learning models that are well-known for their effectiveness and transparency. To make the most out of these models, we delved deep into the literature, trying to understand not only the seminal techniques that are used to this day (such as CART [41] and C4.5 [42]), but also the new advancements that have been developed since then.

During this research, however, we noted that there were no wide-ranging surveys of the DT field published in the last decade. Furthermore, several different lines of DT research had little communication between each other, despite having similar goals. Trying to fill this gap in the literature, we carried out a comprehensive survey of the latest advancements in the field, structuring the contributions in a novel way that had not been proposed before (i.e. a separation by approximation, generalization, and interpretability). The end result was published in *Artificial Intelligence Review* and has received a lot of interest since then, which we take as an indication of its usefulness and clarity. The remainder of this chapter presents this survey as published.

Artificial Intelligence Review (2023) 56:4765–4800
https://doi.org/10.1007/s10462-022-10275-5

Recent advances in decision trees: an updated survey

Vinícius G. Costa[1] · **Carlos E. Pedreira**[1]

Published online: 10 October 2022
© The Author(s), under exclusive licence to Springer Nature B.V. 2022

Abstract
Decision Trees (DTs) are predictive models in supervised learning, known not only for their unquestionable utility in a wide range of applications but also for their interpretability and robustness. Research on the subject is still going strong after almost 60 years since its original inception, and in the last decade, several researchers have tackled key matters in the field. Although many great surveys have been published in the past, there is a gap since none covers the last decade of the field as a whole. This paper proposes a review of the main recent advances in DT research, focusing on three major goals of a predictive learner: issues regarding the fitting of training data, generalization, and interpretability. Moreover, by organizing several topics that have been previously analyzed in isolation, this survey attempts to provide an overview of the field, its key concerns, and future trends, serving as a good entry point for both researchers and newcomers to the machine learning community.

Keywords Decision trees · Machine learning · Interpretable models · Classification algorithms

1 Introduction

In recent years, there has been a growing wealth of various data sources: social media, smartphones, Internet of Things (IoT) devices, health monitors, and many others. By leveraging this new resource, the *machine learning* field has grown rapidly, allowing an increasing number of intelligent applications to learn their functions directly from data instead of having them explicitly programmed. This has resulted in systems capable of controlling self-driving cars (Bojarski et al. 2016), recognizing speech (Amodei et al. 2016) and surpassing humans at the game Go (Silver et al. 2016), among others. Underlying these results is not only an abundance of available data, but also important advances in the learning models themselves (Sarker 2021).

✉ Carlos E. Pedreira
pedreira56@gmail.com

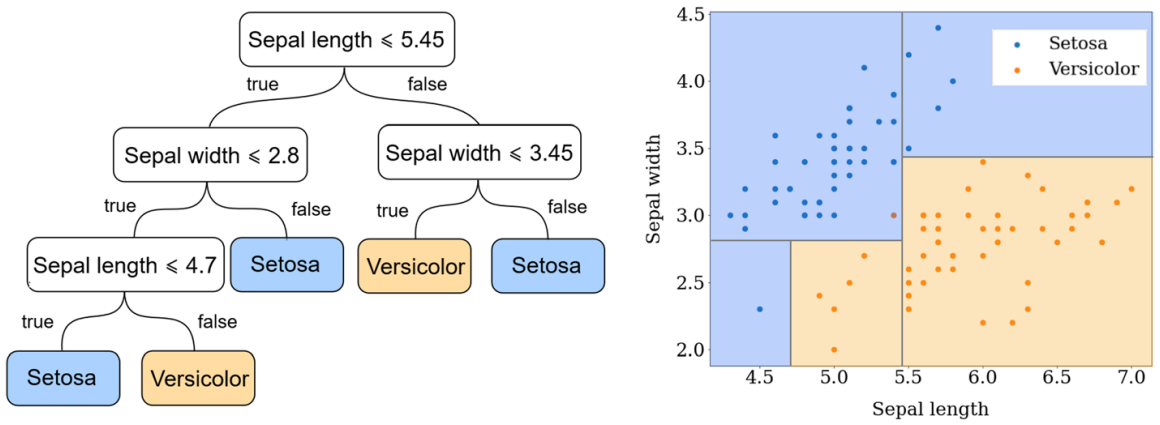
Vinícius G. Costa
vgcosta@cos.ufrj.br

[1] Systems Engineering and Computer Science Department, Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

One of the most popular learning models are Decision Trees (DTs). These models are usually represented in a flowchart-like structure, in which every internal node is a logical test (called a *split*), and every leaf is a prediction (see Fig. 1a). During inference, each observation starts at the root and ends in one of the leaves, following a path that is completely transparent to the user. The simplicity of this approach belies its robustness: DTs are known not only as an interpretable model that is easy to grasp, but also as an accurate method that has stood the test of time, ever since their original proposal in the 1960s (Morgan and Sonquist 1963). Trees present many other advantages as well, such as low computational cost, being able to deal with missing values, and out-of-the-box handling of mixed data (Hastie et al. 2009).

Although DTs have been widely-used for many decades, recently these models have attracted more attention than usual. With the growing ubiquity of machine learning and automated decision systems, there has been a rising interest in *explainable machine learning*: building models that can be, in some sense, understood by humans (Rudin 2019; Roscher et al. 2020). Since DTs are known for their interpretability, many researchers have turned their attention towards this field. The following uptick in interest, in addition to the usual flow of publishing, has resulted in a rich collection of contributions that may benefit from a broad review (see Fig. 2 for an illustration of this increasing interest in the field). Several great reviews have already been written on the subject, such as the ones by Murthy (1998), Rokach and Maimon (2007), Kotsiantis (2013) and Loh (2014), however, the last decade of research remains uncovered. More recent surveys do exist, but they focus on specific tree approaches, like fuzzy trees (Sosnowski and Gadomer 2019) and mathematical programming trees (Carrizosa et al. 2021), therefore not covering the field as a whole. In an attempt to fill this gap, this survey aims for two contributions: first, to review the recent work in DT research, and second, to position those works in relation to well-established previous developments in the field.

The platform Web of Science was used to locate current research papers on the topic of DTs. To further refine the results and guarantee both quality and relevance, the authors defined inclusion and exclusion criteria, such that a paper must satisfy all inclusion criteria and none of the exclusion criteria:



(a) Tree visualization **(b)** Partitioning visualization

Fig. 1 A classification tree induced on a reduced version of the Iris dataset

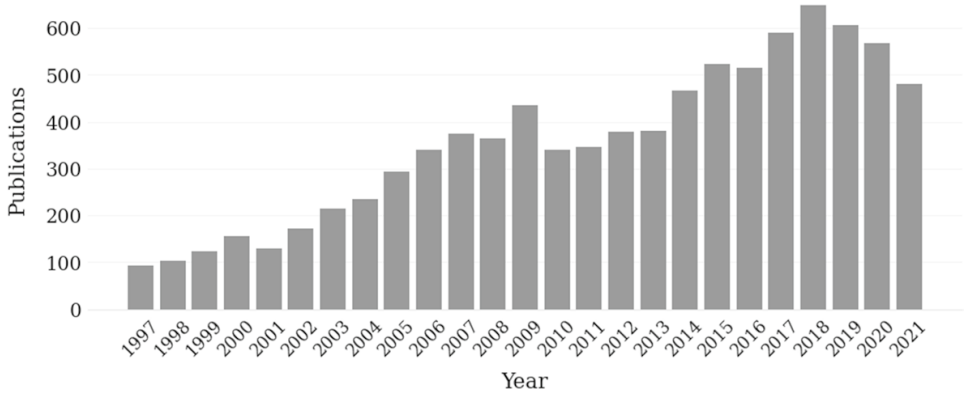


Fig. 2 Number of publications found through the search term "TI=(tree*) AND TS=(classif* OR regress* OR decision)", organized by year of publication. These results only consider the search term and a filter by venue (Computer Science or Engineering categories), therefore applications are also included. The graph was obtained through the Web of Science database

Inclusion criteria

- The paper proposes a method in the DT family.
- The paper was published within the last 10 years in a high-standard journal (as their findings provide the most relevant domain knowledge), or within the last 3 years in a well-reputed conference (as they present the latest advancements). Both sources were given the same relevance.

Exclusion criteria

- The paper was not written in English.
- The paper was not published in a venue related to Computer Science or Engineering.
- The paper only presents an application of a DT method.

Furthermore, certain subjects were not considered, such as fuzzy trees, trees for online learning, and ensembles of trees, because although these fields were clearly linked to DT research at first, they have since become sizeable subjects with their own sets of considerations and challenges. Additionally, throughout the process of writing the survey, two other types of papers were included: conference papers older than 3 years that were relevant enough to be frequently cited by other papers in the survey (e.g. Frosst and Hinton 2017), and papers that either directly continue or precede other papers in the survey (e.g. Demirović et al. 2021 directly expands on Aglin et al. 2020). At the end of this process, 83 recent papers were selected for discussion, in addition to several older seminal papers that were also selected to provide context.

Given the high number of relevant papers, the authors aimed to keep the survey concise by giving more detail to approaches that could be more complex to understand, while giving less detail to approaches that are either more straightforward or based on well-established methods. The level of detail given to a certain paper does not correlate to its quality or novelty, and interested readers are referred to the original papers for further mathematical and technical details.

It also must be noted that this survey does not empirically compare the cited methods. Such comparisons would be valuable since, to the best of the authors' knowledge,

there is no large-scale study comparing different DT methods in a wide range of datasets. However, there are two great obstacles to such a study: first, the majority of papers on the topic do not make the code openly available nor provide implementations for the proposed methods, and second, the papers frequently use different datasets. These two issues make it very difficult to compare a large number of DT methods to each other, and it is the opinion of the authors that comparing only a small subset of these methods would lead to weaker and possibly misleading conclusions with regards to the superiority of a method or family of methods. As such, this survey refrains from empirical comparisons and focuses instead on comparing the approaches from a conceptual and algorithmic point of view, aiming to identify recent trends and research gaps while leaving comparative analyses to future studies.

Next, an outline is presented for the topics and key questions that support the structure of this survey (see also Fig. 3).

- *Approximation* (Sect. 3): Issues regarding the fitting of training data.
  - *Multivariate splits* (Sect. 3.1): Traditionally, the internal nodes of a tree are simple and involve a single variable. Is there any valuable gain in using more complex splits?
  - *Model trees* (Sect. 3.2): In regression tasks, each leaf in the tree corresponds to a constant number, and consequently the tree's outputs are both limited and discontinuous. However, most real-life regression tasks require more complex representations. Are there advantages in employing functions, instead of constants, at the leaves?
  - *Trees for data with specific characteristics* (Sect. 3.3): Traditional trees work well with most kinds of data. But what if the dataset is unlabeled, or if certain classes are more important than others? These additional considerations sometimes require a different approach.
  - *Non-greedy trees* (Sect. 3.4): Traditional DTs are constructed in a greedy way that is both efficient and easy to implement. Can one build better trees by using other optimization approaches?

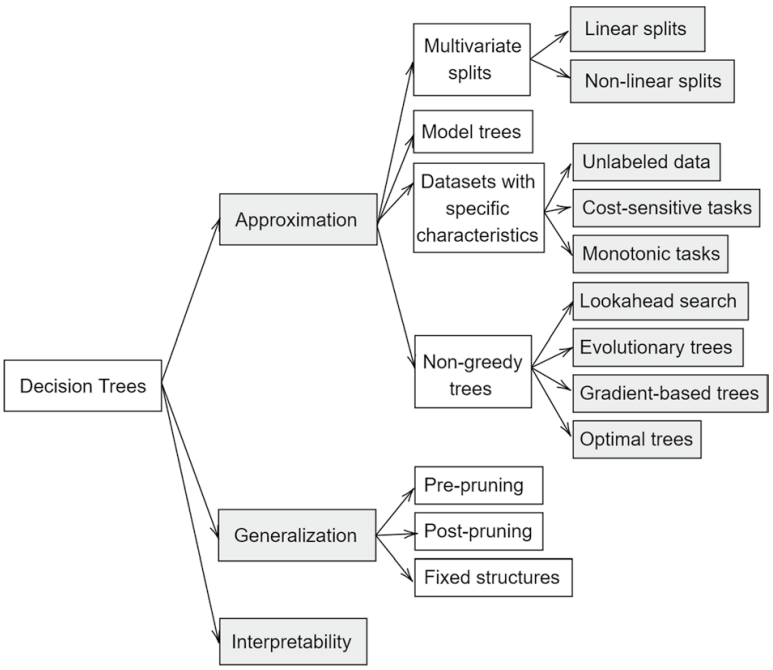


Fig. 3 Visual outline of the survey

- *Generalization* (Sect. 4): Left unattended, most tree construction algorithms will produce a large tree that overfits the training data.
  - *Pre-pruning* (Sect. 4.1): Should one interrupt the algorithm before the tree grows too large?
  - *Post-pruning* (Sect. 4.2): Should one reduce the final tree by eliminating some of its branches?
  - *Fixed structure* (Sect. 4.3): Should one fix the tree structure a priori, therefore eliminating growth altogether?
- *Interpretability* (Sect. 5): Indeed, DTs are often praised for their interpretability. But are they always interpretable? Which design decisions affect interpretability?

As can be seen from the outline, this survey is split into two levels. At the outer level, sections are organized by three major goals of a predictive learner: approximation, generalization, and interpretability. At the inner level, the survey is organized by different design decisions in DT algorithms, either regarding the tree structure (e.g. What are the splits? What are the leaves?) or the induction process (e.g. How is the tree constructed? How is it pruned?).

This organization was chosen to propose a view of the area and ease the flow of reading, but it is not without its problems. The outer levels sometimes overlap since some decisions contribute to more than one major goal (e.g. by reducing tree size, post-pruning enhances both generalization and interpretability). Furthermore, since many of the design decisions at the inner level are not mutually exclusive, they have occasionally been combined in interesting ways, which leads the survey to cite some works in multiple sections. In truth, it is difficult to provide a clear taxonomy for the field since many topics are interconnected, and it is noteworthy that Murthy (1998) identified similar difficulties in his seminal survey.

2 Basic concepts

Intending to make this text as self-contained as possible, this section offers a brief description of how traditional DT algorithms work. By "traditional", this survey refers to the greedy univariate trees built by CART (Classification and Regression Trees) (Breiman et al. 1984), which remains one of the most popular algorithms in the field, and also serves as the base for several other contributions. A DT of this kind is illustrated in Fig. 1a.

Regarding terminology, this survey denotes that the goal of predictive models is to navigate a *hypothesis set* (e.g. the set of possible trees) and select a *hypothesis* (e.g. a particular tree) that is close to the *target function* that one wants to predict. To this end, a *training dataset* consisting of *observations* is used, with each observation being composed of *attributes* (that admit numerical, categorical, or binary *values*) and a *target attribute* (a *label*, in classification tasks, or a constant number, in regression tasks). An ideal hypothesis not only *fits* the training data (i.e. approximates), but also has similar performance in out-of-sample data taken from the same distribution (i.e. generalizes).

Structurally, traditional DTs are composed of two elements: *internal nodes* and *leaves*. Each internal node contains a logical test, called a *split*, of the form "attribute $\leq$ value" (for numerical attributes) or "attribute = value" (for categorical attributes). Since every split has a binary outcome of true or false, every node has two outgoing branches and the

Table 1 Example dataset for a medical diagnosis task with three individuals, and attributes "temperature" and "age"

Temperature	Age	Label
39	58	Sick
40	19	Sick
36	50	Healthy

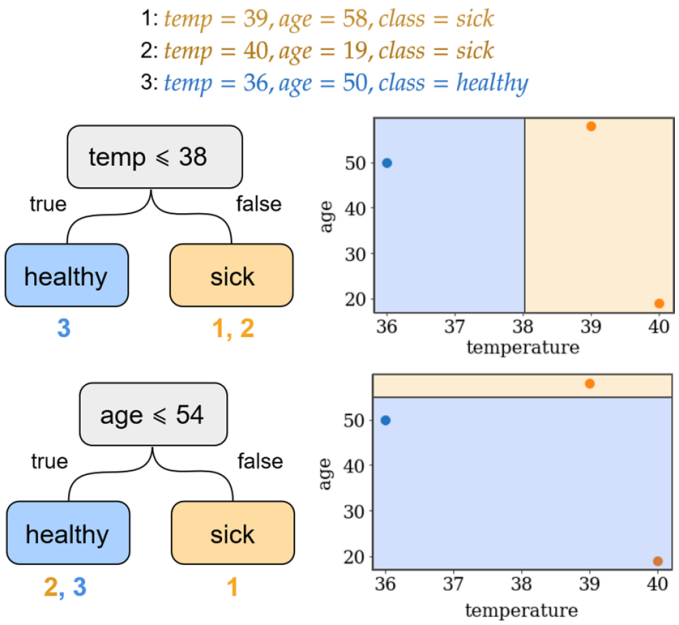


Fig. 4 Comparison between two candidate splits for a medical diagnosis example

tree itself is also binary. The leaves, on the other hand, contain either a label or a constant number, and they serve as the tree's possible outcomes, selected during the inference step through successive application of the splits.

The process of building DTs from training data is usually called *decision tree induction*, and to understand how it is traditionally done, it is useful to visualize a DT as a specific partitioning of the input space. This interpretation is more readily apparent with numerical attributes, and it is illustrated in Fig. 1b alongside the usual flowchart depiction. The root node splits the space into two disjoint partitions, which are passed along to its left and right child nodes. Each of these nodes further splits the partitions they received by applying their own tests and then passing them along to their children. This process is repeated until a leaf is reached, as the leaves correspond to the partitions themselves and contain the predictions associated with each region of the input space. In this context, the induction process can be seen as an optimization problem, in which the goal is to partition the input space in a way that correctly predicts the target function, while using as few partitions as possible.

Traditional algorithms solve this problem through a greedy approach that iteratively selects the most informative splits. To illustrate this notion of *informativeness*, consider a medical diagnosis example in which the attributes are "temperature" and "age", the prediction is either "healthy" or "sick", and the dataset is composed of the three individuals displayed in Table 1. In this case, the split "temperature ≤ 38" is more informative than the split "age ≤ 54", because the first divides the dataset perfectly into healthy and sick individuals, while the latter cannot explain why individuals 2 and 3 have different labels (see Fig. 4). In other words, the first split is more informative because it divides the dataset into *pure partitions* (i.e. every element has the same label), whereas the second split does

not. This impurity is captured by a measure called *splitting criterion*, which for classification tasks is usually either entropy-based information gain (Quinlan 1986) or Gini impurity (Breiman et al. 1984).

From this measure, an induction algorithm follows naturally: the tree can be built one node at a time, in a greedy way that always selects the best split according to the splitting criterion. Because each node considers only the data that has been passed to it from its parent, the procedure fits well with a recursion, and it is aptly called *recursive partitioning*. For classification tasks, it can be summarized as follows:

(1) Let X be a dataset with attributes $(x_1, \ldots, x_n)$
(2) If X is a pure partition with label y:
  (a) Create a leaf that predicts label y, and return.
(3) Otherwise:
  (a) For each attribute x_i:
    (i) For each possible cutoff point or value c:
      (a) Select the best candidate split $(x_i \leq c)$ or $(x_i = c)$ according to the splitting criterion.
      (b) Create an internal node with (x_i, c) being the split.
      (c) Divide X into two datasets X_{left} and X_{right}, according to the split (x_i, c).
      (d) Create a child node for each of the two newly-created datasets.
      (e) For each child node, execute step 1.

Since this procedure will not stop until all the leaves contain pure partitions, it is guaranteed to induce a tree that perfectly classifies the training data. This is great news for the fitting capacity of the model, as the tree will adapt to the complexity of the target function and grow as large as it is necessary to represent it. However, it is also terrible news concerning generalization: in the worst case, pure partitions can only be provided by an exponentially-large tree that has one leaf for every observation in the dataset, which is analogous to fitting N points using a polynomial function of the $(N-1)$th degree. For this reason, the procedure presented above usually results in overfitting, which is one of the reasons why DTs have been accused of having low bias and high variance (Breiman et al. 1984, Sect. 3.1). Fortunately, this weakness of DT algorithms (sometimes called *overpartitioning*) has been extensively studied, and indeed one of the reasons for CART's popularity is precisely the way by which it alleviates this problem: instead of stopping the procedure before the tree overfits (called *pre-pruning*), the algorithm lets the overfitting occur and then eliminates some of the branches from the tree (called *post-pruning*). These topics are explored in Sect. 4.

Furthermore, it can be seen that the above procedure contains some arbitrary decisions which could be reasonably modified. For instance, instead of numerical splits following the form $x_i \leq c$, they could involve a linear combination of attributes such as $w_i x_i + w_j x_j \leq c$. Or perhaps splits could be stochastic, instead of deterministic. Or, maybe, the stopping criterion could be minimum partition size, instead of pure partitions. The possibilities are huge, and in a certain sense, much of DT research can be understood as extensions of this framework. They are the topic of the rest of the survey.

3 Approximation

A fundamental topic in machine learning is approximation: the ability of a model to select a hypothesis that can fit a training dataset by minimizing the associated error. A model with low approximation capacity is unable to approximate complex target functions, regardless of the size of the training dataset (e.g., a linear model is unable to adequately approximate a sinusoidal target function). On the other hand, a model with an excessive high approximation capacity, for a given training set size, can end up overfitting the training data.

There are many aspects of DTs that impact their ability to approximate. The most immediate one is size: a tree with a single split (also called a *stump*) can only represent a single line that is orthogonal to its associated attribute; on the other hand, a tree with many nodes is capable of representing this single line and much more. Although growing larger trees has a positive effect concerning approximation, several other aspects may impact this ability. The following sections consider multivariate splits, model leaves, trees for datasets with particular data characteristics, and non-greedy trees.

3.1 Multivariate splits

Traditionally, the splits in a DT are *univariate*: they evaluate a single attribute at a time, generally in the form of $x_i \leq c$ for numerical attributes and $x_i = c$ for categorical ones. As a result, these univariate splits correspond to hyperplanes that are orthogonal to the tested attributes and parallel to all others, which renders them the alternative name *axis-parallel splits*. Splits of this kind are employed because they are simple to interpret and optimize, but they have their drawbacks: a DT with univariate splits (called a *univariate tree*) is a composition of orthogonal lines, and as such, it is restricted to representing hyper-rectangles in the attribute space, an effect illustrated in Fig. 5a. When facing target functions such as a simple linear combination between two attributes, univariate DT algorithms may end up inducing unnecessarily large trees that may be not only hard to interpret, but also unable to adequately approximate the target function.

The most popular solution to this problem is the induction of trees that consider multiple attributes in each split, called *multivariate trees*. Since the splits are no longer limited to orthogonal lines, the tree can represent much more than hyper-rectangles, therefore enhancing its approximation capacity. However, this improvement also brings its own set of disadvantages: multivariate splits are considered to be less interpretable than their univariate counterparts (Breiman et al. 1984, Sect. 5.2.2), and they are significantly more time-consuming to induce (since there are many more attributes to consider). Consequently, the use of multivariate trees should be evaluated depending on the complexity of the task and the size of the available sample. For an in-depth survey of different aspects related to multivariate trees, please refer to Brodley and Utgoff (1995).

The next two subsections explore how recent contributions deal with some of these issues. These contributions are divided into two categories: those which employ linear combinations of the attributes at each split, and those which employ non-linearities.

3.1.1 Linear splits

Most multivariate DTs employ a linear combination of attributes in some (or all) of their splits, resulting in the so-called *linear* or *oblique* trees. Accordingly, their splits follow the

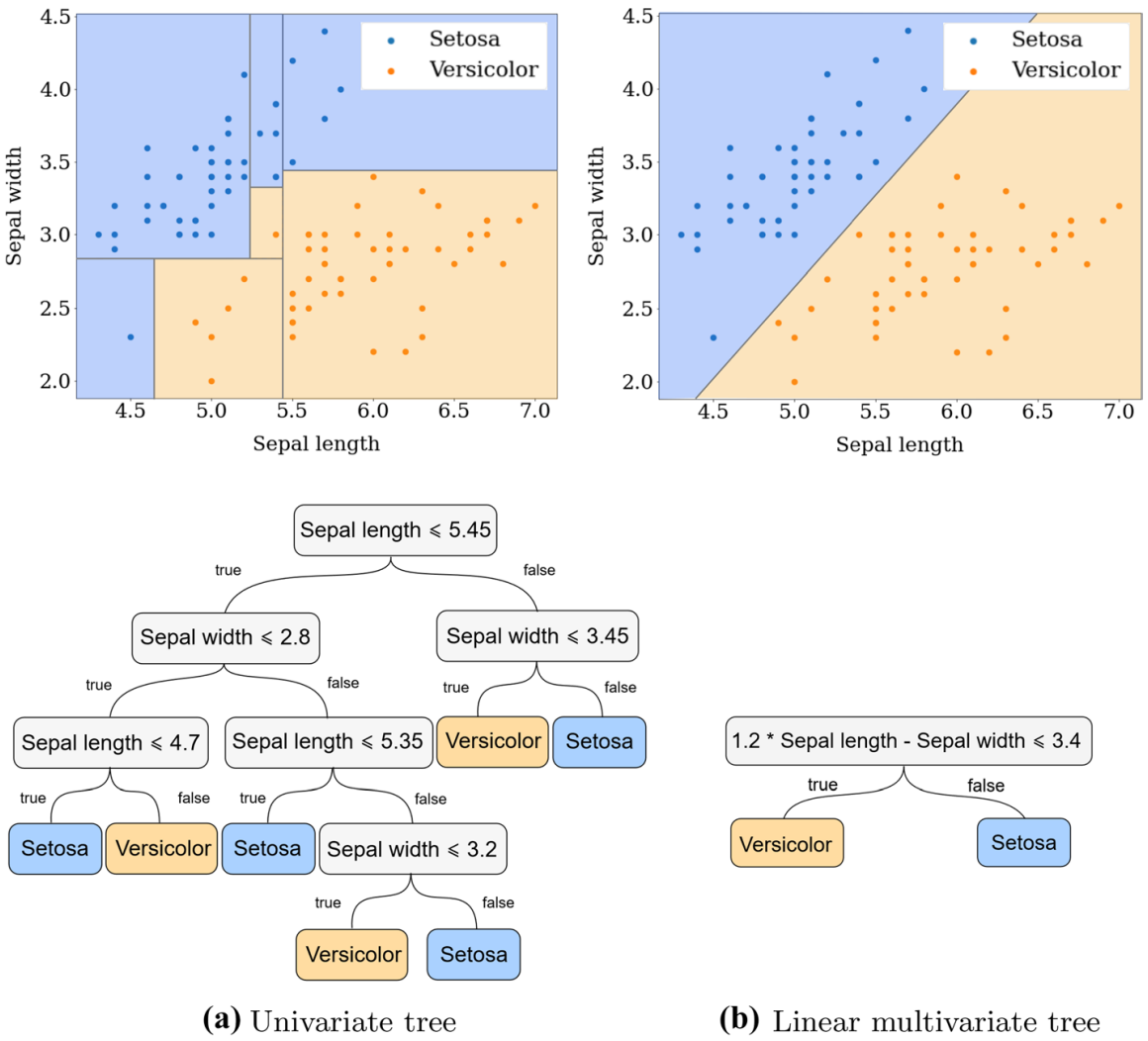


Fig. 5 Two classification trees induced on a reduced version of the Iris dataset

form $\sum_{\forall i} w_i x_i \leq c$, where w_i is the weight associated with the ith attribute x_i, and c is a threshold—please see Fig. 5b for an example of such a linear split. In this context, finding the best linear split amounts to finding a set of weights and a threshold such that the splitting criterion is maximized, but since this problem is NP-Complete for many criteria (Heath et al. 1993), heuristics are often employed. Solutions to this problem can be divided into three main approaches: optimization techniques, linear discriminant analysis, and geometrically inspired heuristics.

The *optimization* approach is exemplified by CART-LC (Breiman et al. 1984), a variant of CART that incorporates linear splits induced by a deterministic hill-climbing algorithm. Since it is not straightforward to linearly combine categorical data, only numerical attributes are considered for linear splits, like most other linear tree algorithms. Although the authors reported that CART-LC is competitive with other non-DT methods in tasks where the class structure depends on a linear combination of attributes, some researchers later argued that the deterministic approach of CART-LC makes it vulnerable to getting stuck in low-quality local optima (Heath et al. 1993; Murthy et al. 1993). This motivated the use of randomized search heuristics, such as SADT (Heath et al. 1993), which employs simulated annealing optimization, and OC1 (Murthy et al. 1993), which combines CART-LC and SADT by employing hill-climbing with random perturbations.

Other multivariate DTs use *Linear Discriminant Analysis* (LDA; Tharwat et al. 2017) techniques to find linear splits. Traditionally, these techniques are used in statistics to find a linear combination of attributes that separates classes, through the construction of a lower-dimensional space that maximizes the between-class variance and minimizes the within-class variance. Because they directly output a linear split, LDA methods fit naturally within the traditional greedy DT framework as a replacement for evaluating every candidate split. Among DTs that employ LDA family methods, this paper highlights FACT (Loh and Vanichsetakul 1988), CRUISE (Kim and Loh 2001), QUEST (Loh and Shih 1997), and GUIDE (Loh 2009).

More recent contributions to the LDA family are Sparse ADTrees (Sok et al. 2015) and Fisher's ADTree (Sok et al. 2016). Both are multivariate extensions of the Alternating Decision Tree model (a type of DT that uses boosting, proposed by Freund and Mason 1999), but they differ in the type of LDA used. Sparse ADTrees employ Sparse LDA (Clemmensen et al. 2011) to induce linear splits with few attributes, whereas Fisher's ADTree uses the more traditional Fisher's linear analysis to determine the linear splits at each node, akin to another DT in the LDA family, Fisher's Decision Tree (López-Chau et al. 2013). Since ADTrees are constructed using boosting, both extensions modify their LDA methods to allow for sample weighting—this is the main difference in the splitting procedures of Fisher's Decision Tree and Fisher's ADTree.

The third group of heuristics obtains the linear split by bringing inspiration from the problem's *geometric* structure. The GDT algorithm (Manwani and Sastry 2012) does this by determining two 'clustering hyperplanes' that are as close to one class and as far to the other one as possible; then, only two splits are ever considered at each node, those being the two angles that bisect the hyperplanes. This considerably lowers the computational cost of finding the splits and is shown to produce intuitive results in many scenarios. Wickramarachchi et al. (2016) proposed a different geometric approach in the HHCART algorithm, which creates a Householder matrix from the vector orientations of each class and uses this matrix to rotate the input space. Similar to LDA techniques, the oblique split in the input space is obtained by finding an axis-parallel split in the rotated space. Subsequently, Wickramarachchi et al. (2019) combined HHCART and GDT into the HHCART(G) algorithm: the induction process is similar to the original HHCART's, but instead of creating a rotated space based on eigenvectors, it bases the rotation on a modified version of GDT's angle bisector.

Up to this point, all the cited linear DT algorithms have followed the traditional greedy and iterative approach to constructing trees. However, more recently, some authors have proposed *non-greedy* trees that employ linear splits, using integer programming (Bertsimas and Dunn 2017; Blanquero et al. 2021; Zhu et al. 2020; Aghaei et al. 2019) and gradient-based optimization (Norouzi et al. 2015). Approaches of this sort are discussed in Sects. 3.4.3 and 3.4.4.

3.1.2 Non-linear splits

Aiming to further enhance DT approximation capacity, some contributions pointed at non-linear splits. Fuzzy Rule-Based DTs (Wang et al. 2015c) employ non-linear splits by representing them as fuzzy conjunction rules, such that the non-linear split $f_1(x_1)f_2(x_2) \leq 0.6$ (where f_1 and f_2 are fuzzy membership functions) can be expressed in the much more interpretable form "does the observation have short sepal length and short sepal width?". Although the reported accuracy was not significantly higher than other multivariate

approaches, FRDT stands out for its superior interpretability resulting from the fuzzy rule usage. Subsequently, Mu et al. (2020) proposed an extension to FRDT called MR-FRBDT, which applies several fuzzy rules at each split instead of a single one, and employs the parallel framework of Map-Reduce to speed up tree induction. Although accuracy and scalability were improved, it may be noted that interpretability was diminished due to the use of multiple fuzzy rules.

Paez et al. (2019) proposed a strategy by which linear and non-linear splits can be introduced to any DT algorithm. This is done by adding Interactive Basis Functions (IBFs) as artificial attributes to the original dataset, such that the final tree may include splits like $e^{x_1} + e^{x_2} \leq 2$, or $x_1 x_2 \leq 7$. Since the user needs to specify which IBFs will be considered, the strategy turns DTs into a semi-parametric approach, therefore requiring domain knowledge that may not always be present. The authors reported that IBFs have better performance in datasets with more attributes, more observations, and fewer labels. It can be noted that FRDT's non-linearity may be seen as a particular case of the IBF strategy.

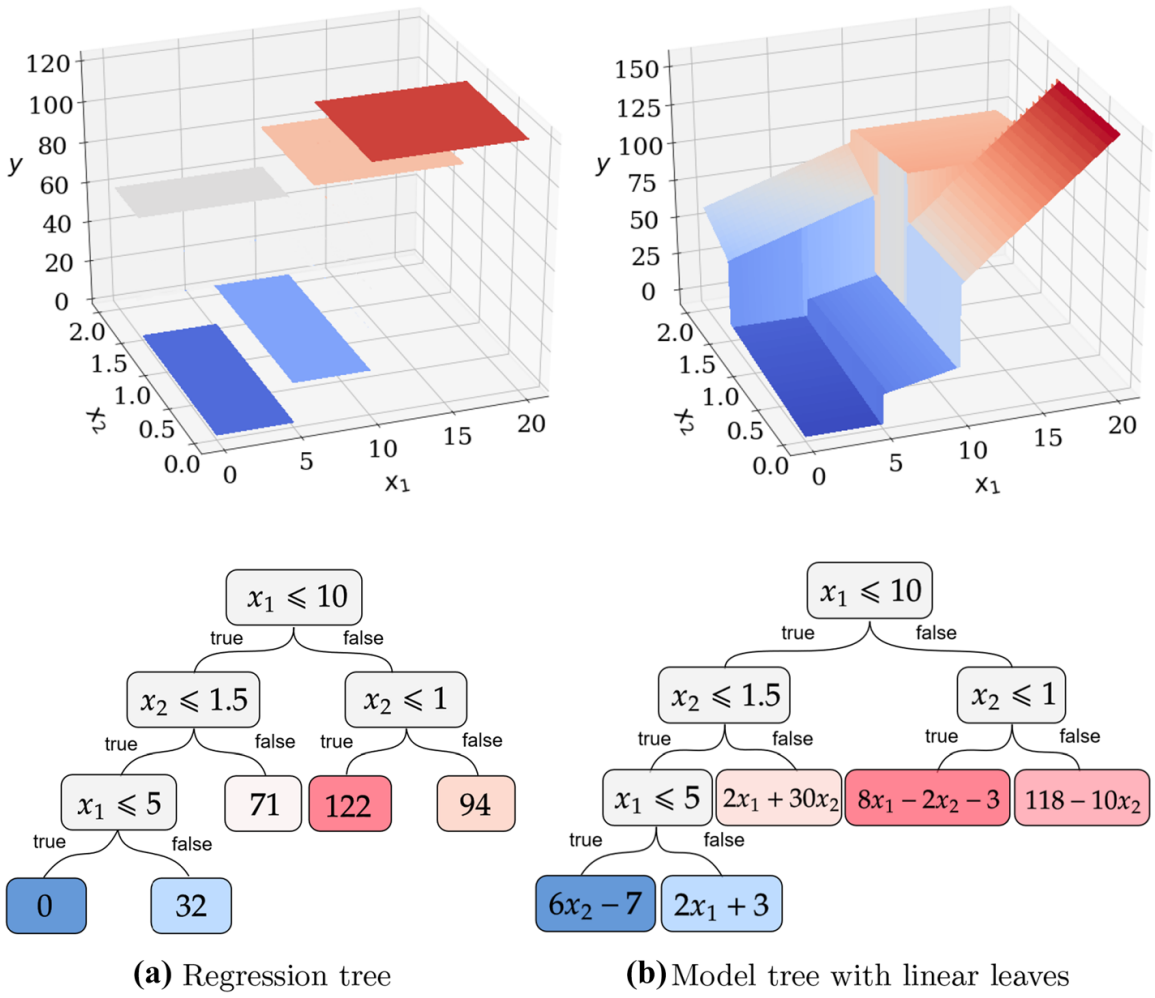
Finally, almost every DT trained with gradient-based optimization employs non-linear splits in some way, either by applying a sigmoid function over a linear combination of input attributes (Suarez and Lutsko 1999; Irsoy et al. 2012, 2014; Norouzi et al. 2015; Frosst and Hinton 2017), or by employing entire neural networks at internal nodes (Hehn et al. 2020; Tanno et al. 2019). These approaches are further explored in Sect. 3.4.3.

3.2 Model trees

While multivariate trees employ more complex splits, other trees employ more complex leaves. *Model trees* are DTs with predictive models in their leaves, instead of labels or constant numbers. Although these trees were originally presented as an extension to regression trees that would not be limited to representing piecewise constant functions (please see Fig. 6) (Quinlan 1992; Breiman et al. 1984, Sect. 8.8), they have since been applied to classification tasks as well, through the use of predictive models such as Naive Bayes (Kohavi 1996) and logistic regression (Landwehr et al. 2005). More recently, research has aimed at inducing more accurate and efficient model trees (Wang et al. 2015b; Zhou and Yan 2019; Czajkowski and Kretowski 2016; Yang et al. 2017; Broelemann and Kasneci 2019), as well as integrating this approach with other DT frameworks (Ikonomovska et al. 2011; Frank et al. 2015).

One of the main challenges of model trees is efficiency: if the greedy procedure is naively employed, then two predictive models (one for each child node) have to be trained to evaluate the quality of a single candidate split (Loh 2011). This is intractable for most real-world tasks, as the number of candidate splits usually scales with the size of the training dataset and the dimensions of its input space. Often, researchers tackle this problem either by growing a standard tree and adding models in post-processing (Quinlan 1992), or by inducing a model for each internal node and then using this model to warm-start the node's children, therefore reducing the cost of training a new model (Landwehr et al. 2005). Both approaches are reflected in recent work.

Broelemann and Kasneci (2019) followed the *warm-starting* approach and proposed an algorithm for inducing trees with any differentiable predictive model for leaves. The authors argued that traditional splitting criteria are inadequate for model trees since they directly attempt to obtain the best partitions; instead, they proposed a novel gradient-based splitting criterion that explicitly considers the model loss of each leaf, reportedly achieving better accuracy than other model trees. After applying the criterion and obtaining a split,



(a) Regression tree **(b)** Model tree with linear leaves

Fig. 6 An illustrative comparison between traditional regression trees and model trees

the algorithm trains a model for each new child node and warm-starts it by using the parent's trained model. Warm-start is also used in FIMT-DD (Ikonomovska et al. 2011), an online algorithm for inducing model trees capable of dealing with concept drift. Each leaf has a corresponding perceptron that uses the Delta rule to update its weights with every observation that falls into that path. Every time a concept drift is detected, the leaf is split further, and its perceptron is copied over to both its children in a warm-starting procedure.

The *post-processing* approach is reflected in the work of Zhou and Yan (2019): their algorithm constructs a traditional C4.5 tree and then, after the post-pruning step, replaces the newly-created pruned leaves with ELM classification models (Huang et al. 2006) trained on the corresponding data partitions. The authors reported that this approach was more effective than an earlier model tree algorithm called ELM-trees (Wang et al. 2015b), which instead employed ELM models during pre-pruning and additionally tackled efficiency by employing parallelism in the evaluation of candidate splits.

Other approaches have been proposed to avoid training an intractable number of models. Alternating Model Trees (Frank et al. 2015) consider only the median value of each attribute as a potential split, therefore severely reducing the number of models that need to be trained. Furthermore, the predictive models themselves are univariate linear functions of the form $ax + b$, and consequently can be quickly trained using linear regression. In contrast, MPTrees (Yang et al. 2017) employ a mixed-integer optimization (MIO) formulation

to find, given an input attribute, both the optimal split point and the linear regression coefficients of its two children. This offers an optimality guarantee that is absent in other model tree approaches, and since only the input attributes need to be iterated, the efficiency of the overall algorithm depends only on MIO optimization. It bears to note, however, that only datasets with less than 5000 observations were included in the paper, perhaps due to the well-known expensive computational cost of guaranteeing optimality—a similar discussion is presented in Sect. 3.4.4.

Finally, Czajkowski and Kretowski (2016) avoided training multiple models by using an evolutionary algorithm that induces model trees in a global way. After evaluating different combinations of DT structure decisions, the authors found that the best model is a fully mixed tree, in which splits can be either uni- or multivariate, and leaves can be either constants or linear predictors.

3.3 Datasets with specific characteristics

Although traditional DT algorithms were proposed as a general tool for predictive learning, they may not be well-suited or applicable to datasets with certain characteristics, such as unlabeled data and monotonic constraints. Therefore, some authors proposed extensions to better fit these situations, usually by introducing new splitting criteria and pruning mechanisms. The following subsections bundle together some of these contributions.

3.3.1 Unlabeled data

As seen in Sect. 2, classification DT algorithms evaluate candidate splits based on their ability to separate the labels. However, there are situations where the data is either completely unlabeled (unsupervised tasks) or partially unlabeled (semi-supervised tasks). In these situations, it is difficult to use traditional splitting criteria, as they depend on knowing the correct prediction for any observation. To effectively tackle these tasks, the traditional DT framework needs to be modified.

The *unsupervised* task has been tackled in the past by DT algorithms such as CLTrees (Liu et al. 2000) and PCT (Blockeel et al. 1998). More recently, Fraiman et al. (2013) proposed Clustering Unsupervised Binary Trees (CUBT), a CART-based algorithm that uses a deviance splitting criterion based on a heterogeneity measure derived from the input's covariance matrices. After using this criterion to construct a CART tree, CUBT prunes the model by merging pairs of leaves with low dissimilarity, according to another proposed measure based on the Euclidean distance. In contrast to traditional pruning procedures, CUBT includes a procedure for merging even non-adjacent leaves, as this enables different leaves throughout the tree to be labeled as part of the same cluster. It bears to note that both deviance and dissimilarity were defined only for continuous attributes, which is why Ghattas et al. (2017) extended this approach to use entropy-based splitting criteria when the attributes are categorical.

Semi-supervised DT learning, on the other hand, had not received much attention until recent years. Tanha et al. (2017) proposed tree induction through a self-training framework: at each iteration, a DT is induced using the labeled data, and the resulting tree is used to classify the remaining unlabeled data. The unlabeled observations with the most confident predictions are incorporated into the labeled set, and the process is repeated until a stopping criterion is reached. Since correctly estimating the confidence of a prediction is a crucial part of self-training, the authors employed techniques that have been previously

reported to enhance the probability estimation capacity of DTs, such as Laplacian Correction (Provost and Domingos 2003), no-pruning (Provost and Domingos 2003), and grafting (Webb 1997). The resulting approach is found to surpass traditional DT algorithms in the self-training semi-supervised task.

However, since self-training is computationally costly and prone to error propagation, Levatić et al. (2017) employed an alternative approach to semi-supervised trees, namely an extension of the previously-cited PCT (Blockeel et al. 1998). The proposed algorithm exploits both labeled and unlabeled data through a splitting criterion that considers not only class impurity (like traditional algorithms) but also attribute impurity. Unlabeled data were used only for the latter type, while labeled data were used for both. A hyper-parameter w weighs these two sources of impurity, effectively controlling the relevance of supervised vs. unsupervised learning. This approach was subsequently extended to multi-target regression (Levatić et al. 2018).

Finally, a similar weighted supervised-vs.-unsupervised splitting criterion was employed by Kim (2016) to produce semi-supervised DTs. However, the goal of the author was not to take advantage of additional unlabeled data (as in a traditional semi-supervised task), but to produce well-separated partitions in which other models could be constructed (as in a model tree environment, discussed in Sect. 3.2). The author argued that labels may be distributed differently in different subspaces, therefore a criterion based only on the output distribution may fail to produce adequate partitions. As such, the input space distribution has to be considered to enhance the subspace partitioning capability of trees.

3.3.2 Cost-sensitive learning

Several real-world problems include some type of specific cost, e.g. the cost of incorrectly classifying an observation (the misclassification cost), or the cost of obtaining an attribute value (the test cost). DTs that consider these two costs have been extensively proposed in the past, and a comprehensive survey on the topic has been written by Lomax and Vadera (2013).

More recently, some contributions have aimed at balancing cost-sensitive learning with *resource constraints*. Chen et al. (2016) proposed trees in which every test costs a time resource and every path from the root to a leaf must not exceed a maximum time. Splitting is done by minimizing not only test and misclassification costs, but also the time used by the node, with the final tree then being post-processed to further grow the nodes that did not exploit the maximum time allotted. Subsequently, the authors extended this idea to consider not only time but multiple arbitrary resource constraints (Wu et al. 2019), though by a different approach: at first, a frequent-pattern extraction method mines rules from the training data that satisfy the resource constraints, and then, a tree is induced using the extracted rules as possible splits. This guarantees that the final tree satisfies all constraints without explicitly considering them during the growing step.

Other authors tackle the issue of *efficiently* building cost-sensitive trees. The ACSDT algorithm (Li et al. 2015) employs two procedures with this aim: first, it considers only a subset of candidate splits by using a gradient-descent-inspired method, and second, it employs feature selection by removing attributes that present low splitting criterion values at some point in the tree construction. This latter idea is also explored in the Batch-Deleting Adaptive DT algorithm (BDADT; Zhao and Li 2017), an algorithm that uses the same feature selection step but considers class imbalance by incorporating weights during splitting.

Finally, Correa Bahnsen et al. (2015) have focused on *example-dependency*: a third type of cost in which the penalty for misclassification depends not only upon the classes but also upon which individual observation is being considered. To tackle this problem, the authors proposed novel splitting criteria and pruning mechanisms that allow for example-specific weights.

3.3.3 Monotonicity constraints

In certain tasks (called *monotonic*), it is known that the target attribute is a monotonic function of the input attributes for at least some subset of the data. Since traditional splitting criteria fail to address such constraints, several authors have proposed extensions over the years, such as Potharst and Bioch (1999) and Cao-Van and De Baets (2003). More recently, Hu et al. (2010) introduced Rank Mutual Information (RMI), a new information measure that combines Shannon's entropy with dominance rough set theory in an attempt to capture the ordinal consistency of the data's partitions. Based on this measure, the authors then proposed a DT algorithm called REMT (Hu et al. 2012) which was reported to be less sensitive to noise than previously proposed algorithms.

Subsequently, Marsala and Petturiti (2015) generalized the RMI approach to splitting criteria other than Shannon's entropy, formally defining a group of rank discrimination measures that includes RMI and a newly proposed Rank Gini Impurity (RGI). In the same paper, the authors also proposed RMDT(H), a tree induction algorithm that evaluates splits based on any rank discrimination measure H from this group of measures. Finally, Pei et al. (2016) proposed a linear multivariate DT called MMT that can employ both RMI and RGI to tasks with partially monotonic constraints. Such constraints are handled by projecting the observations unto the multivariate split vector, consequently creating a space in which non-monotone samples can be seen as monotonic. The splits themselves were obtained in a manner reminiscent of the linear discriminant analysis approach discussed in Sect. 3.1.1, with the difference being that the non-negative least squares method is used, so as to guarantee monotonicity throughout the tree.

3.4 Non-greedy trees

As discussed in Sect. 2, DT induction can be seen as an optimization problem, in which the goal is to construct a tree that correctly fits the data while using a minimal number of splits. However, direct optimization is not a convenient approach, since this task has been identified as NP-Complete in certain cases, such as for binary trees (Hyafil and Rivest 1976). Therefore, heuristics are often employed to efficiently navigate the search space, with top-down greedy algorithms being the most popular by a large margin (e.g., CART, ID3, C4.5, GUIDE).

However, there has been some debate over the effectiveness of greedy approaches. In particular, it can be argued that such algorithms are only one-step optimal and not overall optimal, since the construction procedure only considers the quality of the next split and not of future splits on the same path (Breiman et al. 1984, Sect. 2.8.2). Consequently, several authors have proposed non-greedy heuristics for navigating the DT search space more appropriately, as well as optimizing techniques that improve the tractability of direct optimization. The following subsections explore lookahead search, evolutionary methods, gradient-based optimization, and optimal trees.

3.4.1 Lookahead trees

Lookahead search is an intuitive approach to reduce the shortsightedness of greedy algorithms: instead of selecting the optimal split with regards to the next iteration, a k-lookahead selects the optimal split relating to the next k iterations. Satisfactory results with this approach were initially reported by Norton (1989) and Ragavan and Rendell (1993), with the latter showing particular success in tasks with high attribute interaction. Apparently, the only downside of lookahead search was the increase in computational cost (since a greater number of splits need to be considered).

Nevertheless, Murthy and Salzberg (1995a) demonstrated that one-level lookahead exhibits *pathology*: not only does it fail to produce significantly better trees, but also it may produce trees that are worse than the purely greedy methods. The authors hypothesized that this happens because optimizing common splitting criteria (such as information gain) does not necessarily improve the tree as a whole, therefore a better local optimization can lead to even more sub-optimal solutions. In face of these results, interest in the lookahead approach waned, and other non-greedy methods were explored in the literature.

However, it is not a consensus whether lookahead is overall harmful or if it may have some kind of potential. Esmeir and Markovitch (2007) argued that Murthy and Salzberg's paper only studied datasets with simple target functions, while lookahead is more advantageous when there is high attribute interaction (an observation reflected by Ragavan and Rendell's analysis). They then proposed LSID3, an anytime DT algorithm that constructs trees using a dynamic lookahead and employs a splitting criterion that is biased towards shallower trees. This criterion explicitly considers the size of the subtree under each candidate path, which is estimated via a Monte Carlo sampling of the possible trees. LSID3 differs from the approach studied by Murthy and Salzberg in two main ways: it employs a dynamic lookahead instead of a fixed one-level, and it is used in datasets with high attribute interaction. Perhaps due to these differences, the authors reported that LSID3 outperforms greedy algorithms, especially when a larger time budget is available.

A related and more recent approach is the UCT-DT algorithm (Nunes et al. 2020), which combines Monte Carlo sampling with an anytime property in a different way: namely, it uses the Monte Carlo Tree Search (MCTS) algorithm to navigate the space of possible DTs. In MCTS the search space is represented as a game tree, which is iteratively built by selecting the node with the highest value and expanding it. UCT-DT applies the same idea, but the search space is composed of possible DTs instead of game states, and the value of a particular tree is defined as its accuracy on a validation set. Lookahead search is applied in two ways: first, in the usage of the tree-structured approach of MCTS itself, and second, in the experimental usage of C4.5 to complete the tree before measuring its accuracy. Although the authors found that UCT-DT outperforms greedy approaches on datasets larger than 1000 observations, they also reported that using C4.5 to complete the tree was not superior to using the performance of the current DT, which can be taken to indicate that at least part of the lookahead approach was not beneficial. Overall, there seems to be space for further investigating the effects of lookahead search, especially since the computational constraints are progressively less limiting.

3.4.2 Evolutionary trees

A different approach for avoiding sub-optimality is inducing trees through *evolutionary* algorithms (EAs; Mitchell 1998). Instead of building a single tree, these methods build several trees and randomly modify and combine them using "genetic operators" until a satisfactory solution is found. Moreover, instead of using a splitting criterion to compare candidate splits at each internal node, these methods compare multiple DTs at each iteration using a "fitness function" (usually a weighted sum of tree size and accuracy). Overall, it has been argued that by evaluating the performance of entire trees at once, EAs avoid the sub-optimality trap, and therefore produce more accurate trees (Barros et al. 2012). The more complex search comes with two main downsides: a higher computational cost, and a large number of hyperparameters that need to be somehow adequately determined.

Barros et al. (2012) provided a comprehensive survey on the usage of EAs for the induction of DTs. Since this contribution, many authors have introduced new algorithms to the field, most of which can be adequately analyzed through the taxonomic lenses proposed in that survey. For example, Barros et al. noted the popular usage of multi-objective fitness functions, and some of the more recent algorithms follow the same approach (Brunello et al. 2017; Chabbouh et al. 2019; Karabadji et al. 2017). Furthermore, the survey noted how many EAs employ a random initial population, and this is repeated in some recent contributions (Chabbouh et al. 2019; Karabadji et al. 2017).

Some advances in evolutionary trees deserve to be emphasized: Barros et al. highlighted the need for *efficient* implementations that alleviated the computational cost of EAs, and more recent contributions tackled this goal through parallelism. Czajkowski et al. (2015) combined the parallel approaches of shared memory and message passing to induce DTs using the GDT framework (Kretowski and Grzes 2007), attaining a speedup close to x15 for CPUs with 64 cores. In a subsequent paper (Jurczuk et al. 2017), the same authors improved this speedup by using general-purpose GPUs for evaluating the fitness of each tree, which they argued to be the most time-consuming step of the evolutionary approach. Speedups ranging from x148 to x781 were reported, depending on the dataset and GPU used. Notably, this approach induced trees in less than 10 min for datasets with close to 20 million observations.

Another aspect that has been recently receiving considerable attention is the usage of EAs that also deal with *hyperparameter optimizations*, such as BiLeGA (Adibi 2019), a two-level EA that alternates between feature selection and DT induction. During the feature selection phase, the algorithm generates groups of attributes by applying genetic operators, while in the DT induction phase, a tree is evolutionarily constructed using only the attributes of the corresponding attribute group. Karabadji et al. (2017) similarly considered feature selection, but went one step further by providing genes for many other hyperparameters, such as which subset of training and testing data are to be used, and even which induction algorithm itself (e.g., CART, C4.5, etc). This hyperparameter-evolution approach reached its most ambitious version with HEAD-DT (Barros et al. 2015), an algorithm that aims to evolve entire DT induction methods by including genes for design decisions such as splitting criterion, stopping criterion, post-pruning method, and tree branching factor. By doing this, the authors aimed to obtain not only a better-optimized tree, but a better-optimized DT algorithm for certain domains.

3.4.3 Gradient-based trees

Gradient-based optimization has been increasingly explored as an alternative to greedy DT induction. Although this idea is not new (Jordan and Jacobs 1994; Suarez and Lutsko 1999), the last decade's renaissance in deep learning has prompted many members of the artificial neural networks (ANNs) community to experiment with using backpropagation-inspired approaches in other methods, such as DTs. Most of these contributions are motivated by the sub-optimality argument—greedy algorithms produce sub-optimal trees, therefore the non-greedy gradient-based optimization may help (Suarez and Lutsko 1999; Irsoy et al. 2012, 2014; Norouzi et al. 2015), but with the recent interest in explainable AI, an increasing number of works also report being inspired by the potential of a hybrid model that combines DT's interpretability with ANN's accuracy (Frosst and Hinton 2017; Yang et al. 2018; Silva et al. 2020; Hehn et al. 2020; Wan et al. 2020).

Regardless of the motivation, all such approaches face a common challenge: gradient-based optimization requires continuous and differentiable functions, but traditional DTs are defined by discrete and non-linear decisions, such as selecting which attribute will be used in a split or if an observation goes left or right. To resolve this contradiction, gradient-based methods replace the traditional *hard* DT architecture with a *softer* version that is more amenable for differentiation. This is done via two major modifications: the nature of the splits, and how they are interpreted.

Instead of using the traditional non-differentiable splits $x_i \leq c$, soft DTs usually employ the differentiable function $\sigma(f(\mathbf{x}))$, with $\sigma(\cdot)$ being the sigmoid function and $f(\mathbf{x})$ being the linear combination plus bias $\sum_{\forall i} w_i x_i + b$ (Suarez and Lutsko 1999; Irsoy et al. 2012, 2014; Frosst and Hinton 2017; Wan et al. 2020). The inspiration for this is clear: the tree's internal nodes now mimic the network's neurons, and a backpropagation-inspired derivation is even more immediate. In some cases this analogy is eschewed since f is defined as a convolutional ANN (Hehn et al. 2020; Tanno et al. 2019), however, the overall idea remains the same since these networks are also differentiable.

The sigmoid function brings differentiability, but also creates another problem: how to use its output (a real number between 0 and 1) to decide if an observation goes left or right? The most popular solution is to interpret $\sigma(f(\mathbf{x}))$ (resp. $1 - \sigma(f(\mathbf{x}))$) as the probability of an observation $\mathbf{x}$ going right (resp. left), therefore turning the traditional mutually exclusive splits (called *crisp*) into weighted ones (called *soft*). There are three ways in which these soft splits can be used to output a prediction. In the first way, the outputted prediction is determined only by the leaf with the highest probability (Frosst and Hinton 2017; Wan et al. 2020), whereas in the second way, the prediction is determined by an aggregation of all leaves, for example through a weighted sum (Suarez and Lutsko 1999; Irsoy et al. 2012, 2014; Hehn et al. 2020). These two different modes are illustrated in Fig. 7. The third way differs from the previous two by virtue of being stochastic: the probabilities are sampled at each split, and a path throughout the tree is chosen (Tanno et al. 2019). It may be noted that the first two approaches bring additional computational costs since they require visiting every node in the tree, whereas the last approach avoids this problem altogether and introduces a new potential disadvantage (stochasticity).

There are exceptions to the soft splits solution. Norouzi et al. (2015) managed to use gradient-based optimization in a DT without resorting to soft splits, by establishing a link between traditional DT optimization and structured prediction with latent attributes. This provided their method with a continuous upper bound that serves as a surrogate

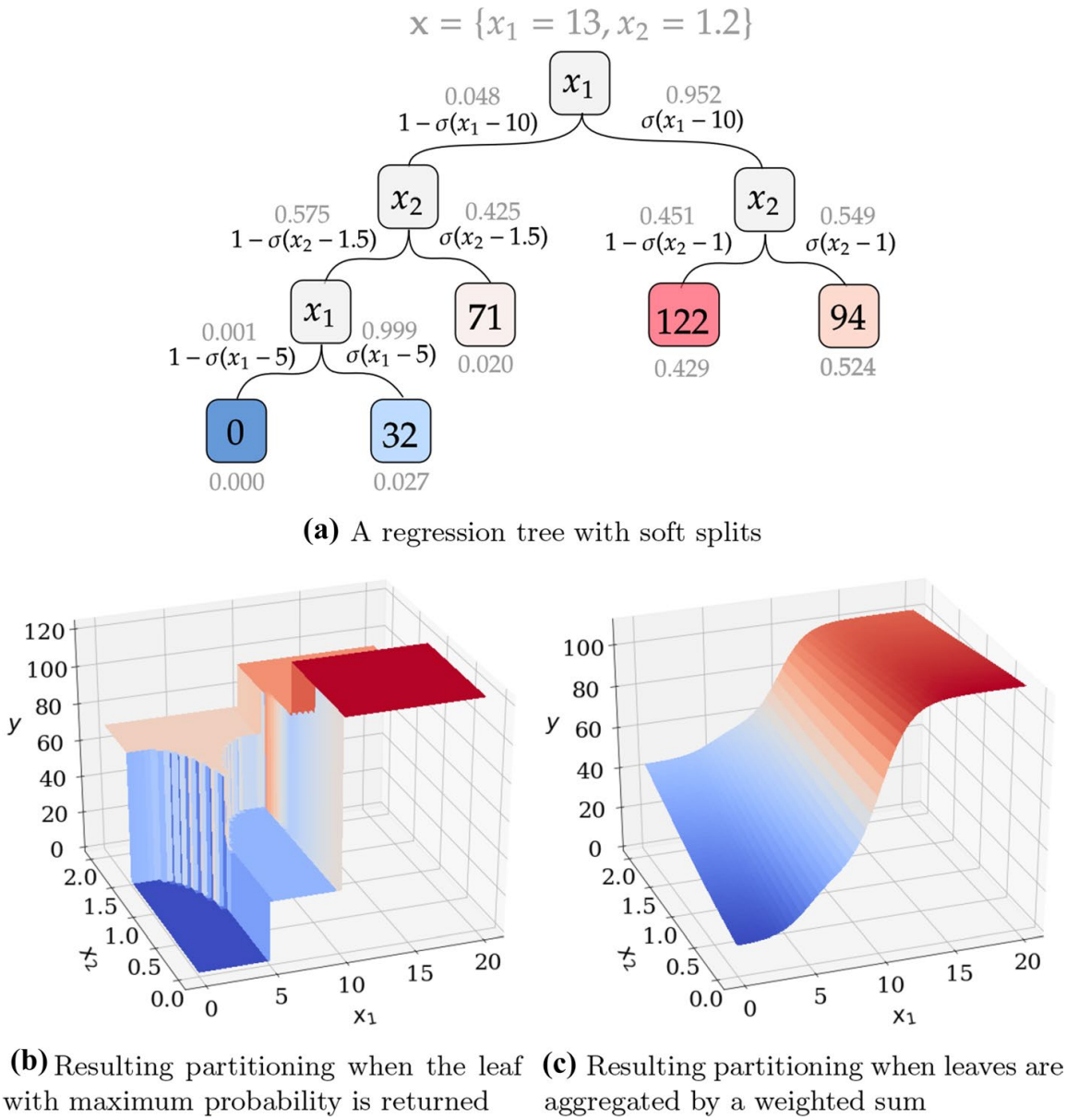


Fig. 7 An illustrative comparison between two ways to interpret soft splits from a regression tree. Gray values correspond to an example observation $\mathbf{x} = (x_1 = 13, x_2 = 1.2)$

objective for gradient descent, therefore bypassing the discontinuous error inherent to traditional crisp splits. Hehn et al. (2020) took a different approach and employed soft splits only during training time, by combining the Expectation-Maximization approach with an annealing scheme that makes the nodes increasingly deterministic. Therefore, at test time, all nodes produce either 0 or 1 as output (for left or right).

Another aspect that differs between these methods is how the *tree induction* occurs. Most algorithms are divided into two phases: greedy tree construction, and fine-tuning. In the first phase, a tree is greedily constructed as usual, except that after each split, the parameters of the new subtree are updated through gradient-based optimization, while the rest of the tree's parameters are fixed. In the second phase (fine-tuning), the algorithm runs a final gradient update through the entire tree, updating all of its weights while fixing the structure. Some algorithms employ both phases (Irsoy et al. 2012; Tanno et al. 2019; Hehn et al. 2020), while others restrict themselves to the fine-tuning step, and assume that a tree structure is given instead of greedily constructing it (these "fixed tree structure"

approaches are later discussed in Sect. 4.3). Still other approaches exist. NBDT (Wan et al. 2020) applies fine-tuning, but induces a tree differently: first, it generates the tree's leaves based on the weights at the output layer of a neural network, and then, it constructs the rest of the tree by merging nodes in a bottom-up process (akin to hierarchical agglomerative clustering). Budding Trees (Irsoy et al. 2014) eschew both greedy induction and fine-tuning by considering every node to be both a leaf and an internal node, therefore conflating tree construction with pruning and updating all weights at every iteration.

Furthermore, one of the benefits of employing gradient-based optimization is the possibility of explicitly using *regularization* to guide the training process. This idea has been explored by a few authors: Norouzi et al. (2015) used norm regularization to constrain the learned weights in the nodes, Hehn et al. (2020) followed a similar approach but through a spatial regularization particular to image inputs, and Frosst and Hinton (2017) employed a penalty that encourages internal nodes to make equal use of both its subtrees.

Overall, gradient-based optimization has been reported to result in trees that are more accurate and shallower, apparently delivering on their promise of less sub-optimal DTs. However, this advantage comes with a cost: by employing all input attributes at every node, it becomes humanly impossible to understand why the model made a certain decision, therefore giving up the traditional notion of DT transparency. This problem and proposed solutions are discussed in Sect. 5.4 on interpretability.

3.4.4 Optimal trees

When taken to the extreme, the idea of avoiding sub-optimality leads to optimal trees: algorithms that search the space of possible models and find the tree that maximizes some measure (e.g. accuracy) while under a size constraint (e.g. maximum depth). Since the search space is impossibly large, methods in this category are more computationally expensive than their heuristical counterparts, which is why despite similar ideas having been explored in the past (Meisel and Michalopoulos 1973; Bennett and Blue 1996), the approach only started to flourish with the hardware advancements of recent years (Bertsimas and Dunn 2017). Currently, there are three main lines of research being investigated: mathematical programming, Boolean satisfiability, and specialized approaches. This subsection explores each in turn and then comments on issues common to all three.

In the *mathematical programming* (MP) approach, DT induction is formulated as a mathematical program, which is then passed to generic commercial solvers such as Gurobi or CPLEX who return the optimal tree. Since the DT induction problem involves several discrete choices, the formulations themselves are usually classified as either MIO (Verwer and Zhang 2017; Bertsimas and Dunn 2017; Rhuggenaath et al. 2018; Aghaei et al. 2019; Firat et al. 2020; Günlük et al. 2021) or purely binary integer programming (BIP; Verwer and Zhang 2019). Although approaches of this line of research can be traced back to the '90s (Bennett 1992; Bennett and Blue 1996), the idea has only gained steam with the technical advances of modern optimization solvers (Bertsimas and Dunn 2017).

This recent interest was ignited in 2017, with the publication of OCT (Bertsimas and Dunn 2017) and DTIP (Verwer and Zhang 2017), two algorithms that induce univariate trees using MIO. DTIP employs fewer variables and restrictions than OCT, which makes it more scalable with regards to sample size; OCT, on the other hand, finds the tree structure in a more sophisticated way and is more thoroughly investigated in relation to the traditional greedy approaches. Both algorithms were subsequently built upon: OCT was extended for regression (Dunn 2018) and prescriptive tasks (Bertsimas et al. 2019), while

DTIP was extended for fuzzy trees (Rhuggenaath et al. 2018) and its authors proposed a novel binary formulation called BinOCT (Verwer and Zhang 2019). BinOCT stands out for replacing the dependency on the dataset size with a dependency on the number of different valued attributes present in the dataset, therefore improving the scalability of optimal trees even further. For a detailed view of MP for DT induction, please refer to Carrizosa et al. (2021).

A second line of research poses DT induction as a *Boolean satisfiability* (SAT) problem, in which the goal is to assign values to Boolean variables such that a certain set of clauses is satisfied. This approach is similar to the previous one in the sense that generic solvers are also employed (SAT solvers in this case), but it notably differs in the definition of optimality that is often used: in the SAT approach, a tree is optimal if it is consistent (i.e. perfectly predicts all observations) while having a minimal size. This focus on consistency is inherited from technical considerations of the SAT framework, and brings with it some downsides: it makes the approach infeasible for noisy datasets (where inconsistent data often appears) and for datasets that have many observations (in which case the consistent trees would be impossibly large). It has also has been reported to result in overfitting, even when the trees are small (Hu et al. 2020).

This line of work started with Bessiere et al. (2009), who proposed an initial formulation for DT induction as an SAT problem, and further showed that it was only practical for small datasets with less than a hundred observations. The works that followed all tried to tackle this issue of scalability: Narodytska et al. (2018) built upon their work and provided a tighter formulation that solved these small instances orders of magnitude faster; Avellaneda (2020) proposed an iterative strategy that considers incrementally larger subsets of the training data, until either the whole training set is considered, or no optimal tree can be found under the size constraints; Janota and Morgado (2020) proposed a formulation that explicitly considers the tree's paths, and was reported to outperform Narodytska et al.'s approach. However, despite this chain of improvements, scalability remains a challenge. More recently, works have considered the SAT approach from other angles: Schidler and Szeider (2021) used SAT techniques to progressively prune a greedy-induced tree, while Hu et al. (2020) used the MaxSAT framework to maximize accuracy instead of capturing it perfectly, resulting in an algorithm closer to the MP approach.

Instead of passing formulations to generic solvers, a third group of works is concerned with more *specialized* approaches. Two lines of research are highlighted.

DL8 (Nijssen and Fromont 2010) frames DT induction as a frequent itemset mining problem: each path corresponds to an itemset, and ideas similar to dynamic programming can be employed to find the optimal tree. This is made possible due to the decomposable nature of DT induction: if a tree is optimal, then any subtree it contains is also optimal. Subsequently, Aglin et al. (2020) built upon these ideas to propose DL8.5, an algorithm that achieves higher efficiency by using more sophisticated caching techniques and bounds to prune the search space. MurTree (Demirović et al. 2021) is currently the last word in this line of research: caching and bounding are further refined, and the problem is explicitly framed in the conventional dynamic programming framework.

A parallel line of specialized algorithms started with OSDT (Hu et al. 2019). Leveraging their previous work on optimal rule lists (Angelino et al. 2017), the authors established several bounds for tree accuracy (e.g. if a tree has a lower bound on its accuracy, then further splitting the tree will maintain the bound, etc.) which the algorithm then uses in an explicit branch-and-bound approach to prune the search space and search it more efficiently. A collection of specialized data structures was also developed to reduce computational costs, similar to MurTree. An extension called GOSDT (Lin et al. 2020) generalized

OSDT to objective measures other than accuracy and further adapted it to treat continuous attributes more efficiently.

This subsection concludes with comments on optimal trees as a whole. An advantage of these algorithms is *flexibility*: since the objective functions are explicitly defined, it is easy to adapt them to domains where accuracy is not the measure to be optimized. This is not the case for traditional greedy trees, where the global objective is not tackled directly but indirectly through the splitting criterion (due to the local optimality itself Breiman et al. 1984, Sect. 4.1). So far, the flexibility of optimal trees has been showcased for objectives like discrimination measures (Aghaei et al. 2019), imbalanced accuracy (Verwer and Zhang 2017; Lin et al. 2020; Günlük et al. 2021), and general combinations of accuracy and size (Bertsimas and Dunn 2017; Hu et al. 2020; Lin et al. 2020; Demirović et al. 2021). A limitation of this approach is that most optimal trees are limited to representing linear measures, as their methods are either based upon linear programming or exploit mathematical properties of linear measures. Non-linear goals require alternative approaches like the bi-objective optimization considered by Demirović and Stuckey (2021).

Another theme commonly discussed is the required pre-processing of *attribute types*. Because splits of the form "attribute $\leq$ value" can be easily formulated as linear constraints, the majority of MP approaches assume that all attributes are numerical, relying on transformations such as one-hot encoding to handle categorical data (Bertsimas and Dunn 2017; Verwer and Zhang 2017, 2019; Firat et al. 2020; Blanquero et al. 2021). This works well but is in stark contrast to traditional greedy algorithms like CART, which handle categorical and numerical data out-of-the-box. The closest optimal tree algorithm in this regard is the formulation proposed by Aghaei et al. (2019), which proposes an MIO problem that handles mixed data.

Inspired by similar technical considerations, other algorithms assume that all attributes are categorical (Günlük et al. 2021) or, more often, binary (Bessiere et al. 2009; Nijssen and Fromont 2010; Narodytska et al. 2018; Verwer and Zhang 2019; Aglin et al. 2020; Janota and Morgado 2020; Demirović et al. 2021). For categorical attributes the encoding is straightforward, but numerical attributes require special consideration, because encoding every different value is infeasible. A proposed solution is "bucketization": the process of ordering the observed values and only considering splits between pairs with different labels (Verwer and Zhang 2019; Aglin et al. 2020). However, although this idea is attractive, Lin et al. (2020) have shown that it sacrifices optimality. More sophisticated approaches include efficiently considering every possible threshold via specialized approaches (Lin et al. 2020), using supervised discretization algorithms (Demirović et al. 2021), or decile binning (Günlük et al. 2021).

A major challenge to all optimal tree algorithms is *scalability*: the high computational cost of searching the entire tree space (albeit implicitly) significantly increases the time necessary to process datasets with more attributes and observations. As previously noted, this is particularly problematic in the SAT line of research, where certificates of optimality are restricted to datasets with hundreds of observations. However, scalability is also an issue in the MP approach, where algorithms only tackle datasets with less than 5000 observations. This effect can also be seen in the depth of the trees considered: most works consider depths of at most 4 or 5 (Bertsimas and Dunn 2017; Verwer and Zhang 2017; Rhuggenaath et al. 2018; Verwer and Zhang 2019; Aglin et al. 2020; Avellaneda 2020; Firat et al. 2020).

Several authors have proposed ways of sidestepping the scalability issue, for example by inducing trees on a subsample of the training data (Bessiere et al. 2009; Narodytska et al. 2018; Avellaneda 2020) or by reducing the input space (Narodytska et al.

2018; Firat et al. 2020). However, these techniques often sacrifice optimality, which seems counter to the initial idea of optimal trees. The line of specialized algorithms seem to fare better in this regard: by avoiding the use of generic solvers in favor of highly specialized implementations that exploit characteristics of the DT induction problem, recent proposals like MurTree and GOSDT can handle datasets with tens of thousands of observations and dozens of attributes, often achieving certificates of optimality.

Interestingly, the scalability issue of optimal trees might not be so problematic. Murthy and Salzberg (1995b) empirically found that the gap between greedy trees and their optimal counterparts is most significant when the target function is complex but there are few observations—the gap diminishes when the training dataset grows larger. If true, this would mean that optimal trees are most useful especially when there are few observations, which is where they already flourish. It remains to be seen if the hypothesis holds for more thorough investigations of this kind.

Lastly, although the previous discussion has focused on univariate trees, it is noteworthy that multivariate trees (seen in Sect. 3.1) have also been adapted to the optimal framework, such as OCT-H (Bertsimas and Dunn 2017), ORCT (Blanquero et al. 2021, 2020), SVM-1 ODT (Zhu et al. 2020) and the formulation by Aghaei et al. (2019). Additionally, Aghaei et al. also considered optimal trees with model leaves, such as the ones seen in Sect. 3.2.

4 Generalization

The previous sections focused on different ways of enhancing the approximation capacity of DTs, but generalization is also a key concern. There is an intuitive trade-off between approximation and generalization: if a model has high representational power, it has a higher chance of perfectly representing the training data (overfitting) possibly at the expense of a poor generalization performance on the out-of-sample data. Conversely, if a model has low representational power, overfitting is less probable and the out-of-sample results will be probably comparable to the in-sample accuracy, although one should be aware that the models may end up underfitted. This situation is analogous to the bias–variance trade-off, and it is also related to the number of degrees of freedom involved in the models.

In the context of DTs, there is a strong connection between generalization and tree size: shallower trees have a lower representational capacity and tend to generalize better, whereas deeper trees correspondingly tend to generalize worse. Because of this, any technique that directly or indirectly reduces tree size can be seen as partly beneficial to generalization, which includes the previously seen multivariate splits (by enhancing the representational capacity of the nodes, the final tree may end up smaller) and non-greedy trees (a more optimal tree may have a lower number of nodes). But generalization concerns more than tree size: many of the topics presented as advantageous to approximation (Sect. 3) can also be understood as contributing negatively to generalization, since they enhance the representational capacity of the tree as a whole. Amid these multiple lenses through which generalization can be seen, this section constrains itself only to approaches that aim at generalization by directly involving tree size, namely: pre-pruning, post-pruning, and fixed structures.

4.1 Pre-pruning

Possibly the most intuitive approach to tree size reduction is simply to stop growing the tree when it becomes too large (according to a pre-determined criterion). This approach is called *pre-pruning*, and it can be seen as an instance of the traditional early stopping procedure. In its most basic form, pre-pruning algorithms amount to applying a minimum threshold on the splitting criterion (Gleser and Collen 1972; Quinlan 1986), but more complex methods have also been employed.

Recently, Wu et al. (2016) proposed SCDT (Size Constrained DT), a DT induction algorithm that grows the tree until all leaves are pure or until the number of leaves reaches the user-specified maximum. As a result, the order of node evaluation becomes highly relevant, and nodes are evaluated in ascending splitting criterion order. This is also the case for other pre-pruning algorithms, such as the recently proposed MSI (Garcia Leiva et al. 2019). In MSI, Kolmogorov complexity is employed during induction to estimate the complexity of the current tree, and when the tree starts to accumulate unnecessary complexity, the growing procedure is interrupted. The authors reported that this approach leads MSI to induce trees that are slightly less accurate but significantly shorter than traditional algorithms, all the while having no hyper-parameters. This trade-off appears to be in line with the reported "horizon effect" of pre-pruning algorithms: since good splits may be hiding behind bad splits, local information is occasionally unable to find more accurate trees (Breiman et al. 1984, Sect. 3.1). This effect is responsible for the inconsistent accuracy of the pre-pruning approach, and it is also why pre-pruning has been largely abandoned in favor of post-pruning methods (Breslow and Aha 1997).

4.2 Post-pruning

The most popular tree simplification approach is post-pruning, sometimes simply called *pruning*. The main idea is to allow the DT to grow unimpeded, and then eliminate the subtrees that are introducing needless complexity to the model; this complexity is often measured on a validation set. Although slower than other methods such as pre-pruning, post-pruning has consistently been shown to improve accuracy, especially in tasks with a high level of noise (Mingers 1989; Quinlan 1987). One classic pruning method is CART's cost-complexity pruning (Breiman et al. 1984), which has two phases: in the first phase, the complete tree is used to create several trees with increasing levels of simplification, and in the second phase, one of these simplified trees is selected according to a measure that combines its accuracy (cost) with its number of nodes (complexity).

Although post-pruning remains an essential element of DT learning, the issue has not received much innovation recently. Most recent algorithms employ traditional post-pruning methods, such as the previously cited cost-complexity pruning (Loh and Shih 1997; Barros et al. 2015; Wickramarachchi et al. 2016), ID3's pessimistic error pruning (López-Chau et al. 2013), C4.5's error-based pruning (Esmeir and Markovitch 2007), or even no post-pruning at all (Hehn et al. 2020; Wang et al. 2015c; Bertsimas and Dunn 2017; Verwer and Zhang 2019).

An exception to this trend is visual pruning (Iorio et al. 2019). In this method, the tree is visualized as a dendrogram-like structure, with the branches having a length proportional to the impurity reduction they bring. The pruning is done by selecting a threshold beyond which all subtrees are pruned to leaves, akin to applying a minimum splitting criterion to

an already constructed tree. The process renders a direct visual interpretation and can notably be employed without the use of a separate validation set, which frees up data that can be used for training. Although advantageous at first, this idea can bring in its own set of problems, since pruning approaches that exclusively use the training data have been found to be more optimistic and therefore prone to overfitting (Mingers 1989). For this reason, pruning approaches of this kind usually employ some type of pessimistic correction (Quinlan 1987). Because such corrections are absent in visual pruning, it remains to be seen if the approach could be enhanced with such corrections.

4.3 Fixed structure

Fixed structure methods take a tree structure as input and search for an accurate set of splits that fits the given structure. Seen from another angle, this approach obtains simplified models by removing trees with undesirable structures from the hypothesis space. Suarez and Lutsko (1999) stated that this procedure is analogous to fixing the architecture and then finding the weights in an ANN.

This idea is common in gradient-based (Sect. 3.4.3) and optimal trees (Sect. 3.4.4), but for different reasons. In gradient-based trees, fixed structures appear to be a consequence of bringing inspiration from ANNs, where the architectures are also fixed. Meanwhile, in optimal trees, fixing the structure is both a technical consideration (the mathematical programming formulations are simpler) and a practical one (optimizing a fixed structure is already computationally costly, therefore iterating over many structures is not prioritized).

Regardless of origin, a key issue is how to determine the fixed structure itself, which either involves user-defined parameters or some sort of automated procedure. User-centric approaches include optimizing the maximal tree of user-specified depth (Verwer and Zhang 2017; Frosst and Hinton 2017; Verwer and Zhang 2019; Blanquero et al. 2021), assuming that the entire structure is provided by the user (Aghaei et al. 2019), or using cross-validation to select a tree from a small group of pre-determined fixed structures (Günlük et al. 2021). Automatic procedures, on the other hand, usually run a greedy induction algorithm like CART and employ the resulting tree's structure as the one to be fixed (Suarez and Lutsko 1999; Silva et al. 2020; Norouzi et al. 2015). A more sophisticated approach can be found in the OCT algorithm (Bertsimas and Dunn 2017), which combines the two ideas: it selects fixed structures from a pool of CART trees trimmed to various depths, up to a maximum depth specified by the user.

In general, fixed structure algorithms are employed more out of necessity than out of concern for generalization, though they are nevertheless positive in that regard (since the provided tree structures are often shallow). However, it has been argued that if the goal is to reduce tree size, then fixing the number of nodes is too rigid a strategy, since the model may become unaware of reasonable trade-offs (e.g. adding a single node may double accuracy) (Freitas 2014). Overall, fixed structures can be seen as both a boon and a barrier to be surpassed.

5 Interpretability

DTs are closely related to interpretability: not only are they often regarded as a particularly interpretable model (Freitas 2014), but also interpretability itself is regarded as the "biggest single advantage of the tree-structured approach" (Breiman and Friedman 1988).

However, not all DTs are equally interpretable, or even interpretable at all, e.g. a tree with thousands of nodes is much more difficult to grasp than a shallower version of it. Consequently, several authors have proposed ways to create more interpretable trees, exploring aspects such as tree size, multivariate splits, soft splits, and others. This section bundles such works.

5.1 Tree size

Tree size is the aspect most commonly associated with interpretability: shallow trees are regarded as very interpretable, whereas deep trees are not (Lipton 2018; Molnar 2022). Using the interpretability framework proposed by Lipton (2018), the issue is *simulatability*: it is difficult for a user to mentally simulate the tree as a whole when there are too many nodes involved. For trees with modest complexity (3 to 12 leaves), this claim was empirically observed in the survey done by Piltaver et al. (2016), in which a strong negative correlation was found between the number of leaves in a tree and a "comprehensibility" score given by the respondents.

But although tree size is important to interpretability, it does not capture this notion entirely. As argued by Freitas (2014), users may find a larger tree to be more interpretable than a shallower version of it, if the splits of the larger model better reflect the user's domain knowledge. This exemplifies the fact that size is only a syntactic notion, and that the semantic aspects of the model (its contents) must be considered for a fuller picture of interpretability.

This complex relation between tree size and interpretability was also identified in the survey by Piltaver et al.: the authors found that if the respondent does not necessarily need to use the whole tree (e.g. they are manually using the tree for prediction), then the number of leaves is not as important as the depth of the deepest leaf required to answer the respondent's question (as measured by both a subjective "question difficulty" rating and the time required to answer a question). Therefore, depending on the task, a large tree may not be as uninterpretable if the most common use cases are closer to the root. Recently, Hwang et al. (2020) proposed a splitting criterion that tackles this same issue: instead of aiming to reduce the combined impurity of two partitions (as is usual in the traditional methods), their criterion focuses on only one of the partitions having low impurity and large observation coverage. As a result, the broadest and most accurate rules are pushed to the top of the tree, reducing the user's need to investigate deeper leaves.

5.2 Multivariate splits

Another aspect that affects interpretability is the presence of multivariate splits: logical tests that involve more than one variable (discussed in Sect. 3.1 and illustrated in Fig. 5b). At first, their presence seems to harm interpretability, since multivariate splits are less interpretable than univariate ones (as can be easily illustrated by two hypothetical splits in a medical task: "temperature ≤ 38" and "$1.2 \times \text{age} + 0.3 \times \text{temperature} \leq 50$"). However, as noted by Brodley and Utgoff (1995), there is a flip-side to the inclusion of multivariate splits: the higher approximation ability of multivariate tree splits may make the trees much shallower than their univariate counterparts (as illustrated in Fig. 5), which would consequently increase the interpretability related to tree size. These opposite results complicate any claim to the general interpretability of these splits. However, it can also be argued that even a single multivariate split is enough to compromise the interpretability of the entire

tree, because if the included multivariate function is uninterpretable, then every path that passes through it will contain an uninterpretable step—therefore harming what has been called the *decomposability* aspect of interpretable predictions, meaning that every part of the model must admit an intuitive explanation (Lipton 2018).

This negative impact on interpretability can be mitigated in several ways, the most common of which is to use multivariate splits that are simpler and more interpretable, such as linear functions. The splits can be further simplified through the use of "feature sparseness" techniques that reduce the number of attributes involved: for example, CART-LC (Breiman et al. 1984) employs a backward elimination process to remove attributes that do not contribute much to accuracy, whereas GUIDE (Loh 2009) considers only bivariate splits from the beginning. More recently, Sok et al. (2015) tackled sparse multivariate splits by using a variant of LDA called Sparse LDA (Clemmensen et al. 2011), while Wang et al. (2015a) employed sparse additive models. Another way to improve the interpretability of these trees is to focus on alternative ways of split interpretation: Fuzzy Rule-based DTs (Wang et al. 2015c) represent non-linear splits by using interpretable fuzzy rules, while Paez et al. (2019) proposed a visualization tool called "decision charts" that allows the user to visually interpret such splits.

5.3 Soft splits

Another design decision that affects interpretability is the possible inclusion of soft splits: logical tests that attribute a weight to each outgoing branch, instead of resulting in mutually exclusive outcomes (like the ones seen in traditional trees, called *crisp*). As discussed in Sect. 3.4.3 and illustrated in Fig. 7, the prediction step of such trees usually involves calculating the weight of every leaf, and either averaging the results or selecting the leaf with maximum weight. As a consequence, soft splits deal a huge blow to the *simulatability* aspect of interpretability (Lipton 2018), because to manually use such a tree for prediction, a user has to simulate every path in the tree while keeping track of all the weights.

However, soft splits also have a positive contribution: if the prediction is done via averaging, then soft splits restore a notion of locality that is absent in traditional splits (Suarez and Lutsko 1999). For example, consider the first tree in Fig. 4: a patient with a temperature of "38.00" will be classified as healthy, whereas the same patient with a temperature of "38.01" would be classified as sick. This difference could plausibly reduce the user's trust in the model, because similar inputs are expected to result in similar outputs (Alvarez-Melis and Jaakkola 2018). A soft split, in contrast, would avoid this problem entirely, because the slight temperature difference would be manifested as a slight change in the tree's weights, and therefore in its final continuous output. As such, if the domain of interest cannot be delimited by sharp boundaries, the soft approach might be more interpretable to the users. It is noteworthy that this is the domain where fuzzy logic is advantageous, as it captures the cognitive uncertainty of categories (Yuan and Shaw 1995), and indeed, the sub-field of DT research known as fuzzy trees (Sosnowski and Gadomer 2019) also employs soft splits (Yuan and Shaw 1995; Janikow 1998; Suarez and Lutsko 1999).

5.4 Other issues

Some trees, such as the ones obtained via gradient-based optimization (seen in Sect. 3.4.3), employ splits that are both soft and multivariate. This results in a model similar to a neural network which cannot be interpreted in the traditional DT way. An intuitive solution is to

simply revert to the traditional structure: Silva et al. (2020) proposed inducing a multivariate soft tree using gradient-based techniques and then transforming it into a univariate crisp tree by selecting the attributes with higher weights. A more common solution is aiming at *post hoc interpretations* (Lipton 2018) of the resulting trees through two main approaches: either by focusing on computer vision tasks, in which the multivariate weights map to features that can be visually interpreted (Frosst and Hinton 2017; Hehn et al. 2020), or by labeling the tree's paths so that each path can be interpreted as representative of a concept (e.g. the left subtree handles man-made objects, while the right handles animals) (Tanno et al. 2019; Wan et al. 2020).

Another issue that affects interpretability is the instability of traditional DT algorithms (Li and Belford 2002): the notion that small changes in the training data might result in the induction of substantially different trees. Instability occurs for two main reasons: first, there might be near-ties during the selection of each split, which means that a split being selected depends upon a small fraction of the data. Second, a single different split during induction results in different subtrees, therefore propagating the difference throughout the model. Instability affects interpretability because it might decrease trust in the training algorithm, since humans expect that similar inputs result in similar outputs (Alvarez-Melis and Jaakkola 2018).

To tackle instability, Wang et al. (2014) proposed a splitting criterion that better disambiguates nearly-tied splits. This is done by introducing the concept of "segments", which allows the criterion to consider not only the purity of each partition but also the label distributions. Their criterion reportedly achieves less unstable trees with higher generalization and smaller size. Subsequently, Yan et al. (2019) extended this approach by proposing two novel splitting criteria, SPES1 and SPES2, both based on the idea of surpassing the previous approach through a better balance of "traditional splitting measures" and "number of segments". On a more theoretical front, Baranauskas (2015) showed through simulation that for traditional splitting criteria, the number of nearly-tied splits grows with the number of classes, with instability also depending on the imbalance of the dataset.

Several other issues fall under the general notion of interpretability. Since traditional trees do not capture causality, they are not useful for providing insight into the causal relationships of the data. To tackle this, Li et al. (2017) proposed a splitting criterion for inducing causal DTs. On the other hand, Johansson et al. (2018) aimed at improving trust in regression trees by guaranteeing a probability of error for each inference, through a framework called conformal prediction. Lastly, Carreira-Perpinan and Hada (2021) proposed an exact algorithm for finding counterfactual explanations in both univariate and multivariate DTs, therefore allowing the user to understand how the input could be changed to obtain a more desirable outcome.

6 Future directions

As shown throughout the paper, the field of DT research is still being actively researched, with proposals being made across many different fronts. This section points out some future trends and interesting problems.

- *Applications in Industry 4.0* With the growth of the IoTs and Industry 4.0 as a whole, there will be increasingly more opportunities to implement automated decision processes in day-to-day life, and it is expected that DTs will play an important role in many of

those applications, especially those in which interpretability is a key concern. Recent applications of DTs in the industry range from validating data from smart energy meters (Elsisi et al. 2021) to detecting motor faults and cyber-attacks (Tran et al. 2021), and many more are expected to be seen.

- *Optimal trees* Currently, optimal trees are one of the most researched topics in the field. These methods sidestep the sub-optimality concerns that have always been present in DT research, and therefore offer an exciting new paradigm with limits that are still being actively explored. The major challenge is scalability: since guaranteeing optimality is much more expensive than settling for a heuristical solution, optimal trees are usually infeasible for larger datasets, a limitation that traditional trees do not have. Specialized algorithms such as GOSDT and DL8.5 appear to be paving the way in this regard, in contrast to approaches that employ generic commercial solvers.
- *Gradient-based trees* It is expected that gradient-based trees will continue to receive some attention, as researchers of the neural networks community attempt to find new ways to combine the interpretability of DTs with the recent developments in neural networks. Although there is fertile ground to be found in inducing more accurate trees by proposing novel gradient-based algorithms, the biggest challenge is not in reaching the accuracy of a neural network, but instead in maintaining the interpretability of the tree, as most gradient-based approaches employ soft multivariate splits that preclude any traditional notion of interpretability. Therefore, an interesting problem is how to take advantage of gradient-based optimization while keeping the final model interpretable.
- *Improving accuracy without compromising interpretability* As discussed throughout the survey, some DT design decisions generally improve approximation capacity while reducing interpretability (e.g. multivariate splits, model leaves, and larger tree size). This conflict hints at an accuracy-interpretability trade-off, but it is not clear that this trade-off is inevitable; indeed, in many situations, it is known to be more myth than fact (Rudin 2019). Therefore, an interesting challenge going forward is how to improve the accuracy of DT models without compromising that quality which is usually lauded as their key advantage: their transparency. This challenge can be handled in two ways: either by inducing more accurate models without significantly changing the structure of traditional trees, or by proposing new ways to interpret non-traditional tree structures. The former is the most common, exemplified through proposals of new splitting criteria, pruning techniques, or optimization methods (such as optimal trees). The latter is less explored, but with much potential nonetheless: recent works have explored new visualization tools (Paez et al. 2019), post hoc labeling techniques (Wan et al. 2020), restricting multivariate splits to user-provided functions (Paez et al. 2019), and exploiting novel representations for multivariate splits (Wang et al. 2015c). Future research could follow along both these paths.
- *Large-scale comparison studies* As pointed out throughout the survey, many different lines of DT research share the same goals: for example, many evolutionary, optimal, and gradient-based algorithms define their main motivation as "surpassing the sub-optimality of greedy decision trees". However, these methods are rarely compared against each other: most papers provide comparisons only to traditional methods such as CART and C4.5, and sometimes to a few other methods in the same family. Because of this lack of communication between different approaches, it is hard to ascertain what are the best algorithms in the field as a whole, which is why most users end up defaulting to traditional algorithms like CART and C4.5. As noted in the introduction, a comprehensive study of this kind is difficult, since many papers do not make their code openly available and/or provide empirical results on different groups of datasets.

These points were brought up by Rusch and Zeileis (2014) and they continue to be an issue, even though some researchers in the field have started to publish code alongside the papers themselves. Still, a large-scale study across many datasets would be valuable going forwards, as the community would be better able to understand the limitations and advantages of each family of algorithms.

7 Conclusion

Since the introduction of the first regression tree algorithm in the seminal paper by Morgan and Sonquist, close to six decades have passed; nevertheless, DTs have remained a widely-used model and an actively researched subject. This paper attempted to review the advances of the last decade in the field, since the most recent surveys do not feature topics like optimal trees and interpretability, which have gained steam in the last few years. In this scenario where increasingly more attention is given to interpretable AI and DT research, the authors hope that this review serves as a good entry point for both researchers and newcomers to the machine learning community.

Acknowledgements Both authors are grateful to the anonymous reviewers for their valuable suggestions and feedback.

Funding This work was partially funded by the Brazilian research agencies CNPq—National Council for Scientific and Technological Development (Grant Number 306258/2019-6), FAPERJ—Foundation for Research Support of Rio de Janeiro State (Grant Number E-26/200.840/2021), and a Ph.D. Scholarship from CAPES (PROEX).

Data availibility Not applicable.

Code availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Adibi MA (2019) Single and multiple outputs decision tree classification using bi-level discrete-continues genetic algorithm. Pattern Recognit Lett 128:190–196. https://doi.org/10.1016/j.patrec.2019.09.001

Aghaei S, Azizi MJ, Vayanos P (2019) Learning optimal and fair decision trees for non-discriminative decision-making. In: Proceedings of the AAAI conference on artificial intelligence, vol 33(01), pp 1418–1426. https://doi.org/10.1609/aaai.v33i01.33011418

Aglin G, Nijssen S, Schaus P (2020) Learning optimal decision trees using caching branch-and-bound search. In: Proceedings of the AAAI conference on artificial intelligence, vol 34(04), pp 3146–3153. https://doi.org/10.1609/aaai.v34i04.5711

Alvarez-Melis D, Jaakkola TS (2018) On the robustness of interpretability methods. arXiv:1806.08049 [cs, stat]

Amodei D, Ananthanarayanan S, Anubhai R et al (2016) Deep speech 2: end-to-end speech recognition in English and Mandarin. In: International conference on machine learning, PMLR, pp 173–182

Angelino E, Larus-Stone N, Alabi D et al (2017) Learning certifiably optimal rule lists. https://doi.org/10.1145/3097983.3098047

Avellaneda F (2020) Efficient inference of optimal decision trees. In: Proceedings of the AAAI conference on artificial intelligence, vol 34(04), pp 3195–3202. https://doi.org/10.1609/aaai.v34i04.5717

Baranauskas JA (2015) The number of classes as a source for instability of decision tree algorithms in high dimensional datasets. Artif Intell Rev 43(2):301–310. https://doi.org/10.1007/s10462-012-9374-7
Barros RC, Basgalupp MP, de Carvalho ACPLF et al (2012) A survey of evolutionary algorithms for decision-tree induction. IEEE Trans Syst Man Cybern C 42(3):291–312. https://doi.org/10.1109/TSMCC.2011.2157494
Barros RC, de Carvalho ACPLF, Freitas AA (2015) Automatic design of decision-tree induction algorithms. Springer Briefs in computer science. Springer. https://doi.org/10.1007/978-3-319-14231-9
Bennett KP (1992) Decision tree construction via linear programming, Technical report. University of Wisconsin-Madison Department of Computer Sciences
Bennett KP, Blue JA (1996) Optimal decision trees. Technical report, R.P.I. Math Report No. 214. Rensselaer Polytechnic Institute
Bertsimas D, Dunn J (2017) Optimal classification trees. Mach Learn 106(7):1039–1082. https://doi.org/10.1007/s10994-017-5633-9
Bertsimas D, Dunn J, Mundru N (2019) Optimal prescriptive trees 1(2):164–183. https://doi.org/10.1287/ijoo.2018.0005
Bessiere C, Hebrard E, O'Sullivan B (2009) Minimising decision tree size as combinatorial optimisation. In: Gent IP (ed) Principles and practice of constraint programming—CP 2009. Lecture notes in computer science, vol 5732. Springer, Berlin, pp 173–187. https://doi.org/10.1007/978-3-642-04244-7_{1}6
Blanquero R, Carrizosa E, Molero-Río C et al (2020) Sparsity in optimal randomized classification trees. Eur J Oper Res 284(1):255–272. arXiv: 2002.09191
Blanquero R, Carrizosa E, Molero-Río C et al (2021) Optimal randomized classification trees. Comput Oper Res 132(105):281. https://doi.org/10.1016/j.cor.2021.105281
Blockeel H, Raedt LD, Ramon J (1998) Top-down induction of clustering trees. In: Proceedings of the fifteenth international conference on machine learning, 1998, pp 55–63
Bojarski M, Del Testa D, Dworakowski D et al (2016) End to end learning for self-driving cars. arXiv preprint. arXiv:1604.07316
Breiman L, Friedman JH (1988) Tree-structured classification via generalized discriminant analysis: comment. J Am Stat Assoc 83(403):725–727
Breiman L, Friedman J, Stone CJ et al (1984) Classification and regression trees. Taylor & Francis, Boca Raton
Breslow LA, Aha DW (1997) Simplifying decision trees: a survey. Knowl Eng Rev 12(01):1–40. https://doi.org/10.1017/S0269888997000015
Brodley CE, Utgoff PE (1995) Multivariate decision trees. Mach Learn 19(1):45–77. https://doi.org/10.1007/BF00994660
Broelemann K, Kasneci G (2019) A gradient-based split criterion for highly accurate and transparent model trees. In: Proceedings of the twenty-eighth international joint conference on artificial intelligence, 2019, pp 2030–2037. https://doi.org/10.24963/ijcai.2019/281
Brunello A, Marzano E, Montanari A et al (2017) Decision tree pruning via multi-objective evolutionary computation. Int J Mach Learn Comput 7(6):167–175. https://doi.org/10.18178/ijmlc.2017.7.6.641
Cao-Van K, De Baets B (2003) Growing decision trees in an ordinal setting. Int J Intell Syst 18(7):733–750. https://doi.org/10.1002/int.10113
Carreira-Perpinan MA, Hada SS (2021) Counterfactual explanations for oblique decision trees: exact, efficient algorithms. In: Proceedings of the AAAI conference on artificial intelligence, 2021, vol 35(8), pp 6903–6911
Carrizosa E, Molero-Río C, Romero Morales D (2021) Mathematical optimization in classification and regression trees. TOP 29(1):5–33. https://doi.org/10.1007/s11750-021-00594-1
Chabbouh M, Bechikh S, Hung CC et al (2019) Multi-objective evolution of oblique decision trees for imbalanced data binary classification. Swarm Evol Comput 49:1–22. https://doi.org/10.1016/j.swevo.2019.05.005
Chen YL, Wu CC, Tang K (2016) Time-constrained cost-sensitive decision tree induction. Inf Sci 354:140–152. https://doi.org/10.1016/j.ins.2016.03.022
Clemmensen L, Hastie T, Witten D et al (2011) Sparse discriminant analysis. Technometrics 53(4):406–413. https://doi.org/10.1198/TECH.2011.08118
Correa Bahnsen A, Aouada D, Ottersten B (2015) Example-dependent cost-sensitive decision trees. Expert Syst Appl 42(19):6609–6619. https://doi.org/10.1016/j.eswa.2015.04.042
Czajkowski M, Kretowski M (2016) The role of decision tree representation in regression problems—an evolutionary perspective. Appl Soft Comput 48:458–475. https://doi.org/10.1016/j.asoc.2016.07.007
Czajkowski M, Jurczuk K, Kretowski M (2015) A parallel approach for evolutionary induced decision trees. MPI+OpenMP implementation. In: Rutkowski L, Korytkowski M, Scherer R et al (eds)

Artificial intelligence and soft computing. Lecture notes in computer science, vol 9119. Springer, Cham, pp 340–349. https://doi.org/10.1007/978-3-319-19324-3_31
Demirović E, Stuckey PJ (2021) Optimal decision trees for nonlinear metrics. In: Proceedings of the AAAI conference on artificial intelligence, 2021, vol 35(5), pp 3733–3741
Demirović E, Lukina A, Hebrard E et al (2021) MurTree: optimal classification trees via dynamic programming and search. arXiv:2007.12652 [cs, stat] ArXiv: 2007.12652
Dunn JW (2018) Optimal trees for prediction and prescription. PhD Thesis, Massachusetts Institute of Technology
Elsisi M, Mahmoud K, Lehtonen M et al (2021) Reliable industry 4.0 based on machine learning and IoT for analyzing, monitoring, and securing smart meters. Sensors 21(2):487
Esmeir S, Markovitch S (2007) Anytime learning of decision trees. J Mach Learn Res 8:891–933
Firat M, Crognier G, Gabor AF et al (2020) Column generation based heuristic for learning classification trees. Comput Oper Res 116(104):866. https://doi.org/10.1016/j.cor.2019.104866
Fraiman R, Ghattas B, Svarc M (2013) Interpretable clustering using unsupervised binary trees. Adv Data Anal Classif 7(2):125–145. https://doi.org/10.1007/s11634-013-0129-3
Frank E, Mayo M, Kramer S (2015) Alternating model trees. In: Proceedings of the 30th annual ACM symposium on applied computing, Salamanca, Spain. ACM, pp 871–878. https://doi.org/10.1145/2695664.2695848
Freitas AA (2014) Comprehensible classification models: a position paper. ACM SIGKDD Explor Newsl 15(1):1–10
Freund Y, Mason L (1999) The alternating decision tree learning algorithm. In: Proceedings of the sixteenth international conference on machine learning, 1999, pp 124–133
Frosst N, Hinton G (2017) Distilling a neural network into a soft decision tree. arXiv:1711.09784 [cs, stat]
Garcia Leiva R, Fernandez Anta A, Mancuso V et al (2019) A novel hyperparameter-free approach to decision tree construction that avoids overfitting by design. IEEE Access 7:99978–99987. https://doi.org/10.1109/ACCESS.2019.2930235
Ghattas B, Michel P, Boyer L (2017) Clustering nominal data using unsupervised binary decision trees: comparisons with the state of the art methods. Pattern Recognit 67:177–185. https://doi.org/10.1016/j.patcog.2017.01.031
Gleser MA, Collen MF (1972) Towards automated medical decisions. Comput Biomed Res 5(2):180–189. https://doi.org/10.1016/0010-4809(72)90080-8
Günlük O, Kalagnanam J, Li M et al (2021) Optimal decision trees for categorical data via integer programming. J Glob Optim. https://doi.org/10.1007/s10898-021-01009-y
Hastie T, Tibshirani R, Friedman JH et al (2009) The elements of statistical learning: data mining, inference, and prediction, vol 2. Springer, New York
Heath D, Kasif S, Salzberg S (1993) Induction of oblique decision trees. J Artif Intell Res 1993:1002–1007
Hehn TM, Kooij JFP, Hamprecht FA (2020) End-to-end learning of decision trees and forests. Int J Comput Vis 128(4):997–1011. https://doi.org/10.1007/s11263-019-01237-6
Hu Q, Guo M, Yu D et al (2010) Information entropy for ordinal classification. Sci China Inf Sci 53(6):1188–1200. https://doi.org/10.1007/s11432-010-3117-7
Hu Q, Che X, Zhang L et al (2012) Rank entropy-based decision trees for monotonic classification. IEEE Trans Knowl Data Eng 24(11):2052–2064. https://doi.org/10.1109/TKDE.2011.149
Hu X, Rudin C, Seltzer M (2019) Optimal sparse decision trees. In: Advances in neural information processing systems (NeurIPS)
Hu H, Siala M, Hebrard E, et al (2020) Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In: Proceedings of the twenty-ninth international joint conference on artificial intelligence, pp 1170–1176. ISSN: 1045-0823. https://doi.org/10.24963/ijcai.2020/163
Huang GB, Zhu QY, Siew CK (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1–3):489–501
Hwang S, Yeo HG, Hong JS (2020) A new splitting criterion for better interpretable trees. IEEE Access 8:62762–62774. https://doi.org/10.1109/ACCESS.2020.2985255
Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is NP-complete. Inf Process Lett 5(1):15–17. https://doi.org/10.1016/0020-0190(76)90095-8
Ikonomovska E, Gama J, Džeroski S (2011) Learning model trees from evolving data streams. Data Min Knowl Discov 23(1):128–168. https://doi.org/10.1007/s10618-010-0201-y
Iorio C, Aria M, D'Ambrosio A et al (2019) Informative trees by visual pruning. Expert Syst Appl 127:228–240. https://doi.org/10.1016/j.eswa.2019.03.018
Irsoy O, Yıldız OT, Alpaydın E (2012) Soft decision trees. In: Proceedings of the 21st international conference on pattern recognition (ICPR2012), 2012, pp 1819–1822

Irsoy O, Yildiz OT, Alpaydin E (2014) Budding trees. In: 2014 22nd international conference on pattern recognition, Stockholm, Sweden, 2014. IEEE, pp 3582–3587. https://doi.org/10.1109/ICPR.2014.616
Janikow C (1998) Fuzzy decision trees: issues and methods. IEEE Trans Syst Man Cybern B 28(1):1–14. https://doi.org/10.1109/3477.658573
Janota M, Morgado A (2020) SAT-based encodings for optimal decision trees with explicit paths. In: Theory and applications of satisfiability testing—SAT 12178, pp 501–518. https://doi.org/10.1007/978-3-030-51825-7_35
Johansson U, Linusson H, Löfström T et al (2018) Interpretable regression trees using conformal prediction. Expert Syst Appl 97:394–404. https://doi.org/10.1016/j.eswa.2017.12.041
Jordan MI, Jacobs RA (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Comput 6(2):181–214. https://doi.org/10.1162/neco.1994.6.2.181
Jurczuk K, Czajkowski M, Kretowski M (2017) Evolutionary induction of a decision tree for large-scale data: a GPU-based approach. Soft Comput 21(24):7363–7379. https://doi.org/10.1007/s00500-016-2280-1
Karabadji NEI, Seridi H, Bousetouane F et al (2017) An evolutionary scheme for decision tree construction. Knowl Based Syst 119:166–177. https://doi.org/10.1016/j.knosys.2016.12.011
Kim K (2016) A hybrid classification algorithm by subspace partitioning through semi-supervised decision tree. Pattern Recognit 60:157–163. https://doi.org/10.1016/j.patcog.2016.04.016
Kim H, Loh WY (2001) Classification trees with unbiased multiway splits. J Am Stat Assoc 96(454):589–604
Kohavi R (1996) Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. In: Proceedings of the second international conference on knowledge discovery and data mining. KDD'96, 1996. AAAI Press, pp 202–207
Kotsiantis SB (2013) Decision trees: a recent overview. Artif Intell Rev 39(4):261–283. https://doi.org/10.1007/s10462-011-9272-4
Kretowski M, Grzes M (2007) Evolutionary induction of mixed decision trees. IJDWM 3:68–82. https://doi.org/10.4018/jdwm.2007100104
Landwehr N, Hall M, Frank E (2005) Logistic model trees. Mach Learn 59(1):161–205. https://doi.org/10.1007/s10994-005-0466-3
Levatić J, Ceci M, Kocev D et al (2017) Semi-supervised classification trees. J Intell Inf Syst 49(3):461–486. https://doi.org/10.1007/s10844-017-0457-4
Levatić J, Kocev D, Ceci M et al (2018) Semi-supervised trees for multi-target regression. Inf Sci 450:109–127. https://doi.org/10.1016/j.ins.2018.03.033
Li RH, Belford GG (2002) Instability of decision tree classification algorithms. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '02, New York, NY, USA. Association for Computing Machinery, pp 570–575. https://doi.org/10.1145/775047.775131
Li X, Zhao H, Zhu W (2015) A cost sensitive decision tree algorithm with two adaptive mechanisms. Knowl Based Syst 88:24–33. https://doi.org/10.1016/j.knosys.2015.08.012
Li J, Ma S, Le T et al (2017) Causal decision trees. IEEE Trans Knowl Data Eng 29(2):257–271. https://doi.org/10.1109/TKDE.2016.2619350
Lin J, Zhong C, Hu D et al (2020) Generalized and scalable optimal sparse decision trees. In: Proceedings of the 37th international conference on machine learning, 2020. PMLR, pp 6150–6160. ISSN: 2640-3498
Lipton ZC (2018) The mythos of model interpretability: in machine learning, the concept of interpretability is both important and slippery. Queue 16(3):31–57
Liu B, Xia Y, Yu PS (2000) Clustering through decision tree construction. In: Proceedings of the ninth international conference on information and knowledge management, CIKM '00, New York, NY, USA, 2000. Association for Computing Machinery, pp 20–29. https://doi.org/10.1145/354756.354775
Loh WY (2009) Improving the precision of classification trees. Ann Appl Stat. https://doi.org/10.1214/09-AOAS260
Loh WY (2011) Classification and regression trees. Wiley Interdiscip Rev Data Min Knowl Discov 1(1):14–23
Loh WY (2014) Fifty years of classification and regression trees. Int Stat Rev 82(3):329–348. https://doi.org/10.1111/insr.12016
Loh WY, Shih YS (1997) Split selection methods for classification trees. Stat Sin 7(4):815–840
Loh WY, Vanichsetakul N (1988) Tree-structured classification via generalized discriminant analysis. J Am Stat Assoc 83(403):715–725. https://doi.org/10.1080/01621459.1988.10478652
Lomax S, Vadera S (2013) A survey of cost-sensitive decision tree induction algorithms. ACM Comput Surv (CSUR). https://doi.org/10.1145/2431211.2431215

López-Chau A, Cervantes J, López-García L et al (2013) Fisher's decision tree. Expert Syst Appl 40(16):6283–6291. https://doi.org/10.1016/j.eswa.2013.05.044
Manwani N, Sastry PS (2012) Geometric decision tree. IEEE Trans Syst Man Cybern B 42(1):181–192. https://doi.org/10.1109/TSMCB.2011.2163392
Marsala C, Petturiti D (2015) Rank discrimination measures for enforcing monotonicity in decision tree induction. Inf Sci 291:143–171. https://doi.org/10.1016/j.ins.2014.08.045
Meisel W, Michalopoulos D (1973) A partitioning algorithm with application in pattern classification and the optimization of decision trees. IEEE Trans Comput C–22(1):93–103. https://doi.org/10.1109/T-C.1973.223603
Mingers J (1989) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4(2):227–243. https://doi.org/10.1023/A:1022604100933
Mitchell M (1998) An introduction to genetic algorithms. MIT Press, Cambridge
Molnar C (2022) Interpretable machine learning, 2nd edn. christophm.github.io/interpretable-ml-book/
Morgan JN, Sonquist JA (1963) Problems in the analysis of survey data, and a proposal. J Am Stat Assoc 58(302):415–434. https://doi.org/10.1080/01621459.1963.10500855
Mu Y, Liu X, Wang L et al (2020) A parallel fuzzy rule-base based decision tree in the framework of Map-Reduce. Pattern Recognit 103(107):326. https://doi.org/10.1016/j.patcog.2020.107326
Murthy SK (1998) Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min Knowl Discov 2(4):345–389. https://doi.org/10.1023/a:1009744630224
Murthy S, Salzberg S (1995a) Lookahead and pathology in decision tree induction. In: Proceedings of the 14th international joint conference on artificial intelligence, IJCAI'95, vol 2. Morgan Kaufmann Publishers Inc., San Francisco, pp 1025–1031
Murthy SK, Salzberg S (1995b) Decision tree induction: how effective is the greedy heuristic? p 6
Murthy S, Kasif S, Salzberg S et al (1993) OC1: a randomized induction of oblique decision trees. In: AAAI, Citeseer, pp 322–327
Narodytska N, Ignatiev A, Pereira F et al (2018) Learning optimal decision trees with SAT. In: Proceedings of the twenty-seventh international joint conference on artificial intelligence. International Joint Conferences on Artificial Intelligence Organization, Stockholm, pp 1362–1368. https://doi.org/10.24963/ijcai.2018/189
Nijssen S, Fromont E (2010) Optimal constraint-based decision tree induction from itemset lattices. Data Min Knowl Discov 21(1):9–51. https://doi.org/10.1007/s10618-010-0174-x
Norouzi M, Collins M, Johnson MA et al (2015) Efficient non-greedy optimization of decision trees. In: Advances in neural information processing systems, vol 28. Curran Associates, Inc., Red Hook
Norton SW (1989) Generating better decision trees. In: IJCAI, pp 800–805
Nunes C, De Craene M, Langet H et al (2020) Learning decision trees through Monte Carlo tree search: an empirical evaluation. WIREs Data Min Knowl Discov. https://doi.org/10.1002/widm.1348
Paez A, López F, Ruiz M et al (2019) Inducing non-orthogonal and non-linear decision boundaries in decision trees via interactive basis functions. Expert Syst Appl 122:183–206. https://doi.org/10.1016/j.eswa.2018.12.041
Pei S, Hu Q, Chen C (2016) Multivariate decision trees with monotonicity constraints. Knowl Based Syst 112:14–25
Piltaver R, Luštrek M, Gams M et al (2016) What makes classification trees comprehensible? Expert Syst Appl 62:333–346. https://doi.org/10.1016/j.eswa.2016.06.009
Potharst R, Bioch JC (1999) A decision tree algorithm for ordinal classification. In: Goos G, Hartmanis J, van Leeuwen J et al (eds) Advances in intelligent data analysis. Lecture notes in computer science, vol 1642. Springer, Berlin, pp 187–198. https://doi.org/10.1007/3-540-48412-4_16
Provost F, Domingos P (2003) Tree induction for probability-based ranking. Mach Learn 52(3):199–215. https://doi.org/10.1023/A:1024099825458
Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106. https://doi.org/10.1007/BF00116251
Quinlan JR (1987) Simplifying decision trees. Int J Man–Mach Stud 27(3):221–234. https://doi.org/10.1016/S0020-7373(87)80053-6
Quinlan JR (1992) Learning with continuous classes. In: 5th Australian joint conference on artificial intelligence. World Scientific, pp 343–348
Ragavan H, Rendell LA (1993) Lookahead feature construction for learning hard concepts. In: Proceedings of the tenth international conference on international conference on machine learning, ICML'93, 1993. Morgan Kaufmann Publishers, Inc., San Francisco, pp 252–259
Rhuggenaath J, Zhang Y, Akcay A et al (2018) Learning fuzzy decision trees using integer programming. In: 2018 IEEE international conference on fuzzy systems (FUZZ-IEEE), pp 1–8. https://doi.org/10.1109/FUZZ-IEEE.2018.8491636

Rokach L, Maimon OZ (2007) Data mining with decision trees: theory and applications. World Scientific, Singapore
Roscher R, Bohn B, Duarte MF et al (2020) Explainable machine learning for scientific insights and discoveries. IEEE Access 8:42200–42216. https://doi.org/10.1109/ACCESS.2020.2976199
Rudin C (2019) Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1(5):206–215. https://doi.org/10.1038/s42256-019-0048-x
Rusch T, Zeileis A (2014) Discussion on fifty years of classification and regression trees. Int Stat Rev 82(3):361–367
Sarker IH (2021) Machine learning: algorithms, real-world applications and research directions. SN Comput Sci 2(3):1–21
Schidler A, Szeider S (2021) SAT-based decision tree learning for large data sets. In: Proceedings of the AAAI conference on artificial intelligence, vol 35(5), pp 3904–3912
Silva A, Gombolay M, Killian T et al (2020) Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In: Proceedings of the twenty third international conference on artificial intelligence and statistics, 2020. PMLR, pp 1855–1865. ISSN: 2640-3498
Silver D, Huang A, Maddison CJ et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489
Sok HK, Ooi MPL, Kuang YC (2015) Sparse alternating decision tree. Pattern Recognit Lett 60–61:57–64. https://doi.org/10.1016/j.patrec.2015.03.002
Sok HK, Ooi MPL, Kuang YC et al (2016) Multivariate alternating decision trees. Pattern Recognit 50:195–209. https://doi.org/10.1016/j.patcog.2015.08.014
Sosnowski ZA, Gadomer Lu (2019) Fuzzy trees and forests—review. Wiley Interdiscip Rev Data Min Knowl Discov. https://doi.org/10.1002/widm.1316
Suarez A, Lutsko J (1999) Globally optimal fuzzy decision trees for classification and regression. IEEE Trans Pattern Anal Mach Intell 21(12):1297–1311. https://doi.org/10.1109/34.817409
Tanha J, van Someren M, Afsarmanesh H (2017) Semi-supervised self-training for decision tree classifiers. Int J Mach Learn Cybern 8(1):355–370
Tanno R, Arulkumaran K, Alexander D et al (2019) Adaptive neural trees. In: Proceedings of the 36th international conference on machine learning, 2019. PMLR, pp 6166–6175. ISSN: 2640-3498
Tharwat A, Gaber T, Ibrahim A et al (2017) Linear discriminant analysis: a detailed tutorial. AI Commun 30(2):169–190. https://doi.org/10.3233/AIC-170729
Tran MQ, Elsisi M, Mahmoud K et al (2021) Experimental setup for online fault diagnosis of induction machines via promising IoT and machine learning: towards industry 4.0 empowerment. IEEE Access 9:115429–115441
Verwer S, Zhang Y (2017) Learning decision trees with flexible constraints and objectives using integer optimization. In: Salvagnin D, Lombardi M (eds) Integration of AI and OR techniques in constraint programming. Lecture notes in computer science, vol 10335. Springer, Cham, pp 94–103. https://doi.org/10.1007/978-3-319-59776-8_8
Verwer S, Zhang Y (2019) Learning optimal classification trees using a binary linear program formulation. In: Proceedings of the AAAI conference on artificial intelligence, vol 33(01), pp 1625–1632. https://doi.org/10.1609/aaai.v33i01.33011624
Wan A, Dunlap L, Ho D et al (2020) NBDT: neural-backed decision trees. arXiv:2004.00221
Wang R, Kwong S, Wang XZ et al (2014) Segment based decision tree induction with continuous valued attributes. IEEE Trans Cybern 45(7):1262–1275
Wang J, Fujimaki R, Motohashi Y (2015a) Trading interpretability for accuracy: oblique treed sparse additive models. In: Proceedings of the 21st ACM SIGKDD international conference on knowledge discovery and data mining, 2015, pp 1245–1254
Wang R, He YL, Chow CY et al (2015b) Learning ELM-Tree from big data based on uncertainty reduction. Fuzzy Sets Syst 258:79–100. https://doi.org/10.1016/j.fss.2014.04.028
Wang X, Liu X, Pedrycz W et al (2015c) Fuzzy rule based decision trees. Pattern Recognit 48(1):50–59. https://doi.org/10.1016/j.patcog.2014.08.001
Webb GI (1997) Decision tree grafting. In: Proceedings of the fifteenth international joint conference on artificial intelligence, IJCAI'97, vol 2. Morgan Kaufmann Publishers, Inc., San Francisco, pp 846–851
Wickramarachchi D, Robertson B, Reale M et al (2016) HHCART: an oblique decision tree. Comput Stat Data Anal 96:12–23. https://doi.org/10.1016/j.csda.2015.11.006
Wickramarachchi DC, Robertson BL, Reale M et al (2019) A reflected feature space for CART. Aust NZ J Stat 61(3):380–391. https://doi.org/10.1111/anzs.12275
Wu CC, Chen YL, Liu YH et al (2016) Decision tree induction with a constrained number of leaf nodes. Appl Intell 45(3):673–685. https://doi.org/10.1007/s10489-016-0785-z

Wu CC, Chen YL, Tang K (2019) Cost-sensitive decision tree with multiple resource constraints. Appl Intell 49(10):3765–3782. https://doi.org/10.1007/s10489-019-01464-x

Yan J, Zhang Z, Xie L et al (2019) A unified framework for decision tree on continuous attributes. IEEE Access 7:11924–11933. https://doi.org/10.1109/ACCESS.2019.2892083

Yang L, Liu S, Tsoka S et al (2017) A regression tree approach using mathematical programming. Expert Syst Appl 78:347–357. https://doi.org/10.1016/j.eswa.2017.02.013

Yang Y, Morillo IG, Hospedales TM (2018) Deep neural decision trees. arXiv:1806.06988 [cs, stat]

Yuan Y, Shaw MJ (1995) Induction of fuzzy decision trees. Fuzzy Sets Syst 69(2):125–139. https://doi.org/10.1016/0165-0114(94)00229-Z

Zhao H, Li X (2017) A cost sensitive decision tree algorithm based on weighted class distribution with batch deleting attribute mechanism. Inf Sci 378:303–316. https://doi.org/10.1016/j.ins.2016.09.054

Zhou X, Yan D (2019) Model tree pruning. Int J Mach Learn Cybern 10(12):3431–3444. https://doi.org/10.1007/s13042-019-00930-9

Zhu H, Murali P, Phan D et al (2020) A scalable MIP-based method for learning optimal multivariate decision trees. In: Larochelle H, Ranzato M, Hadsell R et al (eds) Advances in neural information processing systems, vol 33. Curran Associates, Inc., Red Hook, pp 1771–1781

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Chapter 3

Efficient Evolution of Decision Trees via Fully Matrix-based Fitness Evaluation

In the previous chapter, we presented a comprehensive survey of the recent developments in DT research, which included not only proposals that fall under the traditional greedy approach, but, more importantly, works that innovated upon that original framework – such as multivariate trees, cost-sensitive trees, and trees with soft splits. Given the overarching goal of this thesis, which is to create interpretable models for RL, a question naturally emerges: *Which line of Decision Tree research, among the many that were explored, is the most promising to adapt for RL environments?*

The most immediate answer would be the traditional greedy approach (such as CART and C4.5), since to this day they are the most popular algorithms used to build DTs. However, this strategy is hampered due to a fundamental mismatch between greedy DT algorithms and DTs for RL: in essence, the greedy approach to building DTs requires tree leaves to be independent from each other, which in RL they are not. To clarify this, it is useful to focus on the role played by the leaves. In supervised learning, the leaves are fully independent, so that modifying one leaf affects only the predictions made by that specific leaf; the performance of the rest of the tree remains unchanged. This allows one to intuitively use greedy algorithms to create DTs for supervised learning, since all changes are purely local, and at each stage one can make the best optimal choice for that region of the tree. In an RL task, however, this notion of independence is overturned by the time dimension: while changing one leaf still affects only the predictions that fall under that specific leaf, in RL trees each leaf corresponds to an action (see Figure 1.3), so that one local leaf change has the effect of changing the action taken by the agent when visiting a

certain state, which affects the next visited state as well, and the next, and so on. That is, changing a single leaf can impact the performance of the entire tree, since it may change the entire set of states that the agent visits; in other words, the leaves in a RL tree effectively decide the states that will be visited, and any changes made to them are anything but local. As a result, the greedy approach as a whole does not fit RL tasks very well.

Given this context, from the research that was conducted we identified Evolutionary DTs as the most promising approach to train DTs for RL. As briefly stated in Chapter 1, this approach portrays the tree induction process as an optimization problem, which can therefore be solved by the meta-heuristics known as Evolutionary Algorithms [57, 58]. Instead of building single trees node-by-node like is done by traditional greedy algorithms, Evolutionary DTs consider entire populations of trees and operate on them all at once. The general approach can be outlined as such: first, each tree is modified randomly by a set of pre-defined operations. Then, the trees are evaluated by a *fitness function*, which measures their quality and ranks them. Finally, the lowest-quality trees are removed, while the highest-quality trees are recombined to produce the next population. This process is repeated until a stopping criterion is reached, such as a fixed running time or a minimum solution quality.

There are three main reasons why the evolutionary approach was identified as a natural fit for the task of building DTs for RL. First, it is straightforward to calculate the fitness function of a particular tree solution for RL, since to measure its quality one can simply run several simulations using this tree solution as an RL agent, and use the rewards obtained across these simulations. Second, the evolutionary approach uses stochastic operators to modify its solutions from generation to generation, and while there are several optimization problems for which these operators are inefficient (such as problems of linear or integer programming), the task of building DTs for RL is not one of them, since no general heuristic for improving RL trees has been discovered. Thirdly and finally, the aforementioned interdependence between leaves in RL trees is not an issue for evolutionary algorithms, since each solution is evaluated as a whole (i.e. global evaluation), instead of having its leaves and inner nodes separately evaluated (i.e. local evaluation, as in a greedy algorithm).

However, despite this natural fit with our stated goal, evolutionary algorithms have a well-known drawback: high computational cost. Since the technique considers entire populations of solutions and operates on them for hundreds or thousands of iterations, it is often orders of magnitude slower than other approaches, especially ones that consider a single tree at a time [54]. In the case of RL, the situation is further exacerbated by the fact that to evaluate how an agent performs (i.e. to

calculate its fitness), one cannot simply run a single simulation, but must run dozens or hundreds of them in order to obtain metrics that are statistically meaningful. Though one simulation is often fast, the cost quickly compounds.

Given this context, the next step of this thesis' research comprised an investigation on how the computational cost of evolving DTs could be alleviated. In particular, we leveraged the efficiency of numerical computation by proposing a novel matrix-based encoding that represents trees as matrices and the tree training process as a series of matrix-based operations. Most crucially, this investigation restricted itself to the notion of supervised learning, since it is a simpler context on which the validity of the approach could be first tested; the subject of building DTs specifically for RL will resume in Chapter 4. The remaining of this chapter presents the proposed matrix-based encoding and CRO-DT, the evolutionary algorithm presented to exemplify this encoding.

Applied Soft Computing Journal 150 (2024) 111045

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Efficient evolution of decision trees via fully matrix-based fitness evaluation

Vinícius G. Costa [a,1], Sancho Salcedo-Sanz [b], Carlos E. Pedreira [a,*]

[a] Systems Engineering and Computer Science Department, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[b] Department of Signal Processing and Communications, Universidad de Alcalá, Alcalá de Henares, 28805, Madrid, Spain

ARTICLE INFO

Keywords:
Decision trees
Interpretability
Problem encoding
Coral reef optimization
Evolutionary computation

ABSTRACT

Decision Trees (DTs) are a class of supervised learning models that are widely used for both classification and regression applications. They are well-known for their interpretability and robustness, which have led them to remain popular even 60 years after they were first proposed. However, because traditional tree algorithms use greedy methods that are prone to suboptimality, several works have explored the usage of evolutionary algorithms instead. Although these algorithms are often reported to outperform the traditional greedy approach, their computational cost is much higher, since the evolutionary component requires a large number (millions or billions) of function evaluations in order to produce a single tree. Aiming to reduce this computational cost, in this work we propose an encoding that allows the training and evaluation of DTs using only matrix operations. The proposed procedure is shown to be much faster than the traditional tree implementation for complete trees with depths ranging from 2 to 6, and for datasets ranging in size from 100 to 100,000 observations. In particular, the results show speedups of nearly up to 20 times, especially when the dataset is large and the desired tree is small enough to be interpretable. The proposed procedure also benefits from GPU parallelization, although it is still highly performing without it. Furthermore, we propose an evolutionary algorithm, called Coral Reef Optimization for Decision Trees (CRO-DT), that integrates this encoding with a pre-existing ensemble algorithm to evolve better univariate trees. The results obtained show that the proposed CRO-DT is competitive with traditional and modern tree algorithms, consistently producing models of good quality across 14 tested UCI Datasets. We conclude that for most relevant situations, the proposed matrix encoding provides significant speedups over the traditional implementation, and also may serve as a basis for high quality evolutionary DT algorithms.

1. Introduction

Decision Trees (DTs) are supervised learning models, typically represented as flowcharts, that are popular for both their interpretability and performance across a wide range of domains. Although these models have been widely used for several decades (the first contributions date back to the sixties [1]), the field has recently seen a resurgence of attention [2], in part due to the increased interest in interpretable machine learning [3,4]. One particular direction of DT research that has gained traction is that of Evolutionary Decision Trees (EDTs) [5]: methods that use evolutionary algorithms to train trees of small size and high performance. Because these methods optimize trees globally, they often outperform traditional greedy algorithms like CART [6] and C4.5 [7], whose local decisions are more likely to produce suboptimal models [5].

However, this advantage comes at a price: EDTs frequently report much higher training times than their greedy counterparts, since the evolutionary approach evaluates a large number of solutions in order to train a single model. This can strongly limit the applicability of EDT algorithms in large datasets (which are increasingly common in the age of Big Data), as well as the performance of the trained models (since the quality of the final solutions often depends on how many iterations are executed). Indeed, back in 2011, [5] highlighted the computational cost of EDTs as a key issue in the field. Since then, most of the work in this direction has tackled the problem either through parallelization [8–11] or fitness inheritance [8,12], two important fronts that have led to impressive speedups.

In this paper, we propose a new approach to reduce the computational cost of evolutionary trees, namely by encoding the trees as matrices and evaluating their fitness through a series of matrix operations. This is in stark contrast to the traditional `if`-based implementation of DTs, which introduces a computational overhead that can scale poorly — especially when the same dataset needs to be

* Corresponding author.
E-mail address: pedreira@cos.ufrj.br (C.E. Pedreira).
[1] This work was partially done while VGC was in a stay at the Universidad de Alcalá.

https://doi.org/10.1016/j.asoc.2023.111045
Received 6 February 2023; Received in revised form 16 August 2023; Accepted 8 November 2023
Available online 11 November 2023
1568-4946/© 2023 Elsevier B.V. All rights reserved.

evaluated millions or billions of times, such as in evolutionary algorithms. Conceptually, our approach parallels the successful applications of deep neural networks [13] to large datasets, which have only been made possible due to their highly efficient numerical representation as matrices. By exploiting this same efficiency for DTs, we have achieved significant speedups in the EDT approach, especially when training small trees on large datasets. The proposed procedure is compatible with and benefits from GPU parallelization, but still displays high performance even in a CPU-only environment. Matrix encodings have been proposed in the past [14,15], but to the best of our knowledge, this is the first proposal of a fully matrix-based representation for the entire fitness evaluation procedure.

Furthermore, we also propose an EDT algorithm built with this approach that is based on the Coral Reef Optimization with Substrate Layers (CRO-SL) [16,17] technique. CRO-SL is an evolutionary multi-method ensemble algorithm [18] capable of applying different search methods within a single population. It has already been successfully applied to very different optimization problems in Science and Engineering fields, including energy and microgrids design [19,20], computer science [21], logistics [22], mechanical and structural design [23–25] and electrical engineering [26,27], among others. In this paper, we apply CRO-SL with distinct variants of Differential Evolution to evolve trees through their matrix-based representations. The resulting algorithm, which we call CRO-DT, explores the space of multivariate trees but produces univariate solutions with low depth, thus ensuring interpretability. We evaluated the proposed algorithm on benchmarks from the UCI Machine Learning Repository [28] and found that the proposed CRO-DT is competitive with traditional and modern DT algorithms. This demonstrates not only the validity of the matrix approach as a basis for EDTs, but also that the reduced computational cost allows EDTs to search wider and obtain better solutions.

The rest of the paper is structured as follows. First, Section 2 covers the relevant background knowledge and provides an overview of the related works. Section 3 describes the matrix encoding proposed in this paper, while Section 4 focuses on the proposed EDT algorithm, CRO-DT. Both the proposed methods are tested and evaluated in Section 5. Finally, Section 6 closes the paper with our concluding remarks.

2. Background and related work

To make the paper self-contained, in this section we discuss key notions from DTs, EDTs, and the CRO line of evolutionary algorithms. At the end, we provide a review of the related works.

2.1. Decision trees

Decision Trees (DTs) [2,29] are machine learning models used for classification and regression tasks. These models are composed of inner nodes, which contain logical tests, and leaves, which contain predictions (labels for classification and numerical values for regression). In the inference step, an observation follows a path from the root to one of the leaves, determined by the results of each logical test applied to the tested observation. These tests (often called "splits") can be either univariate or multivariate:

- **Univariate splits**: involve a single attribute, usually of the format $x_i \leq c$, where x_i is an attribute and c is a threshold;
- **Multivariate splits**: involve multiple attributes, usually in a linear combination such as $w_1x_1 + w_2x_2 + \cdots + w_Px_P \leq c$, where P is the number of attributes and c the threshold. Such linear tests are also called "oblique splits". Note that if we define a dummy attribute $x_0 = 1$, we can rewrite this test in the simpler version $\sum_{i=0}^{P} w_i x_i \leq 0$, where $w_0 = -c$.

Besides the different types of logical tests, there are also distinct types of tree structures, e.g. trees can be binary (each node has two children) or multiway, trees can be complete (each leaf has the same depth) or non-complete, trees can be crisp (each observation is assigned to a single leaf) or soft, and so on. In this paper, we only deal with complete binary crisp classification trees, although the proposed matrix scheme can be extended to non-complete binary trees of any kind.

The traditional way to build a DT of this sort is through a greedy top-down procedure called recursive partitioning, in which nodes are created iteratively with the goal of better partitioning the dataset. Typically, all possible splits are considered at each step, and the node created is the one that best improves accuracy. This is the approach taken by classic algorithms widely used to this day, such as CART [6], ID3 [30] and C4.5 [7]. However, since decisions are made greedily, there is no guarantee that the final tree is the one with the best global accuracy — it may well be that the optimal tree contains inner nodes that are not actually the locally best. This is the "suboptimality problem", often used as an argument against greedy DT algorithms [31–33].

2.1.1. Evolutionary decision trees

Evolutionary DTs (EDTs) [5] have been proposed as a way to avoid the suboptimality problem of greedy DT algorithms. By treating DT construction as an optimization problem, EDTs use techniques from a field of optimization methods called Evolutionary Algorithms (EAs; [34]) to obtain high-quality trees. A superficial and general description of the evolutionary process is as follows. First, at each iteration (called a "generation"), the EDT algorithm considers a whole group of trees (called a "population"), instead of a single one. Then, each tree is given a score based on its quality (called its "fitness"), which is usually a linear combination of its accuracy and its size. Next, the trees with the highest fitness are recombined with each other using various operators, so that the resulting solutions make up the population of the next generation (the low-fitness solutions are usually discarded or kept only with a very small probability). This process is repeated until a predetermined stopping criterion is reached, for example a maximum number of generations. At this point, the algorithm stops and the best tree of the last generation is returned as the final solution. This overall approach has been implemented with several different evolutionary algorithms, such as Differential Evolution [15,35,36], Particle Swarm Optimization [37,38], Ant Colony Optimization [39], and Grammatical Evolution [40], among others. In this paper, the usage of the Coral Reef Optimization (CRO) algorithm is investigated, since it has been shown to be effective in a wide range of problems, including those related to machine learning [41,42]. Furthermore, the substrate layers version of CRO is an ensemble algorithm that can combine other optimization approaches and extract the best from each of them, through application of its self-adaptive strategies. This is discussed in the following section.

2.2. The PCRO-SL algorithm

In this section, we briefly describe the Probabilistic Coral Reef Optimization with Substrate Layers (PCRO-SL, [43]) algorithm, which is the basis of the CRO-DT algorithm proposed in this paper. Since PCRO-SL is the result of a series of iterative refinements, we will first describe the basic CRO, then its successor CRO-SL, and finally the latest PCRO-SL.

2.2.1. CRO: The Coral Reef Optimization algorithm

The Coral Reef Optimization algorithm [44], first published in 2014, is an evolutionary algorithm inspired by the natural processes of reef formation and coral reproduction. Its characteristics makes it a meta-heuristic algorithm with similarities to genetic algorithms and simulated annealing. In the CRO the reef is represented as a square grid of dimensions $M \times N$, while the corals (i.e., solutions to the optimization problem) correspond to the cells of said grid. During initialization, a

fraction ρ_0 of the cells of the reef are filled with arbitrary initial solutions (e.g. randomly generated). Then, the coral reproduction process is simulated as follows:

1. **Larvae formation**: in this step, new individuals (called *larvae*) are created through two processes that mimic the external and internal reproduction of real-life corals:
   (a) **Broadcast spawning**: A fraction F_b of the reef's corals are selected as broadcast spawners. These individuals are then separated into pairs, such that each pair produces a new larva through a crossover mechanism (or other strategy).
   (b) **Brooding**: The corals not selected for broadcast spawning (a fraction $(1 - F_b)$ of the reef) use a mutation mechanism to produce one new larva each.
2. **Larvae setting**: after the larvae have been created, each tries to find a position in the reef where they can settle down to grow. This is done by selecting a random cell and checking whether (1) the spot is empty, or (2) the larva's fitness is higher than the fitness of the coral in the current cell. If either of these conditions is met, the larva moves to the cell and becomes a new coral; otherwise, the larva can select another random cell and try again. Each larva can try a maximum of k times.
3. **Asexual reproduction**: all corals on the reef are sorted according to their fitness, and a small fraction F_a of the fittest are selected for asexual reproduction. Each of these individuals will mutate and produce a new larva that will try to colonize the reef in exactly the same way as described in the previous step.
4. **Depredation**: in this step, some corals become candidates for elimination. A fraction F_d of the lowest-fitness individuals is selected, and each one has a probability P_d of being predated and freeing up their coral reef cell.

The evolutionary process is executed until a stopping criterion is fulfilled, usually a maximum number of function evaluations or of generations. This decision, along with several others (such as the parameters, which crossover and mutation mechanisms to use, etc.), is left open to each implementation.

2.2.2. CRO with substrate layers

The CRO-SL algorithm builds on the basic CRO process by introducing a new concept: substrates. Marine studies have shown that the success of coral larvae is related to the seafloor substrate in which they fall after reproduction. Inspired by this, the reef grid in CRO-SL is divided into T substrates (i.e. regions) of approximately equal size, so that each coral on the reef performs the broadcast spawning phase differently depending on the substrate it is currently occupying. For example, in one substrate this might mean performing a 1-point crossover between two corals to obtain a new larva, while in another it might mean applying a Gaussian perturbation to a single coral. All other parts of the evolutionary process remain the same, and the algorithm continues as before.

Due to its mixture of several different exploration strategies, CRO-SL can be considered as a multi-method ensemble algorithm [18], and thus belongs to a more sophisticated and different category than CRO. Furthermore, this approach also leads CRO-SL to distance itself from simple metaheuristics, and take it a step closer towards hyper-heuristics: algorithms that attempt to select and combine several simpler heuristics in order to obtain better solutions [45].

2.2.3. Probabilistic CRO-SL

A second improvement on the CRO formula can be found in Probabilistic CRO-SL (PCRO-SL) [43], a modification of CRO-SL that replaces the spatiality of the substrates with a stochastic component. Each substrate has a probability p_i of being selected. Then, during the broadcast spawning phase, each coral samples these probabilities in some way (e.g. via roulette selection) so that the selected substrate determines the reproductive mechanism to be executed. This has the effect of spreading the influence of a substrate over the entire reef, rather than localizing it to a specific region.

In its simplest form, each p_i is set uniformly so that all substrates have the same probability of being selected. However, the original PCRO-SL paper also describes a *dynamic* scheme that updates these probabilities according to the success of each substrate. For example, in the "larvae success rate" scheme, the probability p_i is updated at each generation to reflect the fraction of successful larvae produced using the ith substrate (i.e. larvae that managed to settle on the reef). After obtaining the larval success values for each substrate, a softmax function is applied to normalize them and produce the final probabilities. The original paper also proposed other metrics, such as the average fitness of each larva produced by a substrate and the improvement in fitness of those larvae.

In general, the adoption of probabilistic substrate selection allows the algorithm to generate more diverse solutions, since each coral can be affected by numerous modification mechanisms during its lifetime. Furthermore, the dynamic scheme guarantees that the selection of these mechanisms is commensurate with their quality, resulting in a self-adapting procedure that rewards high-performing substrates while reducing the impact of low-performing ones.

2.3. Related works

Some papers in the literature also use *matrix representations* for DTs. In [15], the authors evolved a multivariate tree model called Perceptron Decision Trees (PDTs, [46]) using a popular evolutionary algorithm called Differential Evolution (DE, [47]). Since DE requires that its solutions be represented as vectors of real numbers, the authors encoded the tree using a matrix scheme similar to the one proposed here so as to adequately interface with the evolutionary algorithm. Their proposal differs from ours in three important ways: first, it uses three different matrices to represent each tree (one for weights, one for constants, and one for leaves), while we use only a single one. Second, their approach also evolves the label-leaf assignments, while in our work these assignments are obtained by a procedure that guarantees their optimality. Third, their method evolves multivariate trees, while our proposal produces univariate trees. This last point is a fundamental distinction that prevents the algorithms from being compared in a fair way, since a multivariate tree of depth d has more fitting capacity than an univariate tree of the same depth, while simultaneously being less interpretable.

In terms of particulars, [14] contains the encoding that is most similar to the one used in our paper, as the authors also represented DTs with a single matrix of weights and classified observations using matrix operations. However, despite these similarities, the two approaches differ in several important ways: because the authors' goal was to train DTs using stochastic gradient descent, their matrix approach stops after classifying individual observations, and thus refrains from classifying entire datasets, optimizing label leaves, computing accuracy, or processing entire sets of trees at the same time — all procedures handled by our approach. Furthermore, because their work was proposed in a very different context, there is no mention that these matrix computations might be useful in the EDT family of algorithms, nor is there any discussion of the speedups that result from using the matrix encodings. Finally, their work considers multivariate trees (like the PDTs mentioned previously), which prevents direct comparisons with our method. Other authors have explored *vector-based representations* of DTs (as opposed to matrix-based ones), such as [35,48]. However, these vector-based representations were converted into traditional tree structures to compute accuracy, while in our work the entire fitness evaluation procedure involves only the matrix encoding.

There are several works that aim to *reduce the computational cost* of EDTs, but they explore either fitness inheritance [8,12] or parallelization [8] instead of the matrix representation seen here. In *fitness inheritance* techniques, the hierarchical nature of DTs is used to identify situations where the fitness of certain subtrees does not need to be evaluated, since it can be inferred (or simply copied) from previous computations. [12] were the first to apply this idea to EDTs, with an approach that stores the observation indices at each leaf and determines when certain genetic operators require recalculation. The authors analytically computed an upper and lower bound for the speedup gained and reported a training time savings of 30%–40%, averaged over 11 datasets. A similar idea is used in the Global Decision Tree (GDT) framework proposed by [8], but instead of storing observation indices at the leaves, an index table is computed for each tree, mapping nodes to training data.

In addition to fitness inheritance techniques, several works have aimed to reduce the computational cost of EDTs by *parallelizing* the evaluation of DTs. Most of these contributions revolve around the GDT framework, and together form a long line of iterative improvements to the system. [49] used a distributed approach in which different processors evaluated different subsets of the global population (called "islands"), with periodic communication between them. [9] built on this approach, combining the OpenMP [50] and MPI [51] paradigms to achieve further speedups. The focus then shifted to GPU-based optimization, with [11] designing 6 procedures amenable to GPU parallelization that together comprise most of the tree induction process. In the latter paper, the authors reported speedups of up to 200x for some combinations of datasets and graphics cards. An approach that simultaneously exploits both fitness inheritance and GPUs can be found in [10]. Although the approach proposed in this paper is also compatible with parallelization and does benefit from GPU usage, this is not the driving force behind the highest speedups obtained, and the method still has a high performance when GPUs are not available.

The *PCRO-SL method*, which serves as the basis for the CRO-DT proposed here, belongs to a line of evolutionary algorithms that have found great success in several domains. These algorithms have been used for positioning turbines in offshore wind farms [52], estimating country-wide energy consumption [53], extracting information from web-based resources [54], among others. In addition, the algorithm has been integrated in the past with machine learning models, such as in [41], where it is used to select features to be passed to a neural network, and in [42], where it is used to compress convolutional neural networks. It is relevant to point out that the algorithm has not been used before to evolve DTs.

3. Fully matrix-based fitness evaluation of classification trees

This section contains the main methodological contribution of this paper. We will show how DTs can be trained and evaluated using only matrix operations, complete with a numerical example to aid comprehension. For details on how DTs are usually implemented, we refer to Appendix A.

3.1. Methodological aspects

We begin by establishing some basic notation and concepts. Let $\mathcal{X} \in \mathbb{R}^{N\times(P+1)}$ be a matrix representing a dataset of N observations, each with P numerical attributes, such that the kth observation is denoted by $\mathbf{x}_k = \begin{bmatrix} x_{k,0} & x_{k,1} & \dots & x_{k,P} \end{bmatrix}$ (where $x_{k,0}$ is a dummy attribute added to handle the bias, always taking the value 1). Let $y \in \mathbb{R}^N$ be a vector containing the numerical labels for each observation in $\mathcal{X}$, such that $y_k \in \{1, \dots, C\}$.

Let T be a complete binary tree of depth d, consisting of S_i inner nodes and $S_\ell = S_i + 1$ leaves. Without loss of generality, we assume that the splits are oblique, such that the ith inner node (in pre-order traversal) contains the split $\sum_{j=0}^{P} w_{i,j}x_{k,j} \leq 0$ (for a generic data point $\mathbf{x}_\mathbf{k}$).[2] Given this configuration, we can encode T as a weight matrix $W \in \mathbb{R}^{S_i\times(P+1)}$, such that the ith inner node of the tree (in the pre-order traversal) corresponds to the ith row in W. Fig. 1 illustrates this encoding with respect to a tree of depth 2, for which the weight matrix is:

$$W = \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,P} \\ w_{1,0} & w_{1,1} & \dots & w_{1,P} \\ w_{2,0} & w_{2,1} & \dots & w_{2,P} \end{bmatrix} \tag{1}$$

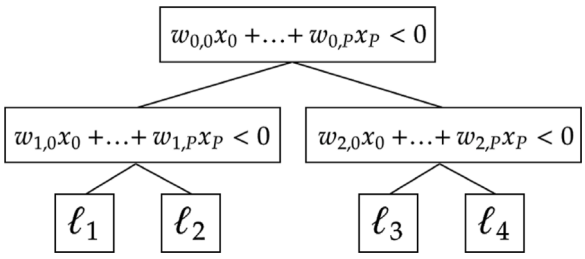


Fig. 1. A multivariate decision tree with depth 2, for a dataset with P attributes.

To fully define this decision tree, it is necessary to define not only its splits, but also the label for each of its leaves, which will be represented as a vector $L = \begin{bmatrix} \ell_1 & \ell_2 & \dots & \ell_{S_\ell} \end{bmatrix}$. Note, however, that there is no real choice involved in deciding L, since given a weight matrix W and a dataset $(\mathcal{X}, y)$, there is a single vector L^* that optimizes the accuracy of the entire tree: it is the one formed by running the tree over the entire dataset and setting ℓ_k as the most frequent class among the observations that fell into the kth leaf (i.e., majority vote). Thus, one can evaluate the fitness of the tree simply by having a dataset $(\mathcal{X}, y)$ and a weight matrix W, since by counting we can determine the optimal label distribution at the leaves and hence its accuracy.

Given this discussion, two goals can be outlined: first, to obtain L^* given W, $\mathcal{X}$, and y; second, to obtain the accuracy of the tree given W, L^*, $\mathcal{X}$, and y.

Algorithm 1 MatrixEval($\mathcal{X}, y, W, S_\ell, S_i, d$): fully matrix-based evaluation of a DT

$N \leftarrow \#\mathcal{X}$
$M \leftarrow$ CreateMask(d) ▷ Described in Algorithm 3
$\sigma(\cdot) \leftarrow \text{sign}(\cdot)$ ▷ Defined element-wise
$Z \leftarrow \sigma(W X^T)$
$\phi(\cdot) \leftarrow \max(0, \min(1, \cdot))$ ▷ Defined element-wise
$Q \leftarrow \phi(MZ - (d-1))$
$Y \leftarrow [y^T | \dots | y^T]^T$ ▷ Such that $Y \in \mathbb{R}^{S_\ell \times N}$
$\rho(\cdot) \leftarrow \left(\prod_{j=1;j\neq\cdot}^{C} \frac{(j-Y\odot Q)}{(j-\cdot)}\right)\mathbb{1}$
$R \leftarrow [\rho(1)| \dots |\rho(C)]$
$L^* \leftarrow$ row-wise argmax(R)
Accuracy $\leftarrow \sum$(row-wise max(R))$/N$
return L^*, Accuracy

To compute L^*, note that if $\mathbf{x}_k$ follows the left (resp. right) path at the ith inner node, then by definition $\sum_{j=0}^{P} w_{i,j}x_j \leq 0$ (resp. $\sum_{j=0}^{P} w_{i,j}x_j > 0$). Let σ be the sign operator, defined so that $\text{sign}(0) = -1$ (to break any possible ties). Thus, $Z = \sigma(W\mathcal{X}^T)$ is a matrix such that if $\mathbf{x}_k$ would follow the left path at the ith inner node, then $Z_{i,k} = -1$, otherwise $Z_{i,k} = +1$. Therefore, Z can be used to determine the full

[2] Note that we do not lose any generality by using oblique splits, since this scheme is also able to represent univariate trees: the ith split is univariate if $\mathbf{w}_i = \begin{bmatrix} w_{i,0} & w_{i,1} & \dots & w_{i,P} \end{bmatrix}$ is 0 except for two elements: the bias $w_{i,0}$ and the weight $w_{i,j}$ corresponding to the selected attribute j. In this case, the oblique split can be rewritten as the univariate $x_j \leq -\frac{w_{i,0}}{w_{i,j}}$ if $w_{i,j} > 0$, or $x_j \geq -\frac{w_{i,0}}{w_{i,j}}$ otherwise.

path that an observation $\mathbf{x}_k$ would take, if we combine it with a mask matrix M of dimensions $(S_\ell \times S_i)$ where:

$$M_{j,i} = \begin{cases} -1 & \text{if } j\text{th leaf is on the } \textbf{left} \text{ path of } i\text{th inner node} \\ +1 & \text{if } j\text{th leaf is on the } \textbf{right} \text{ path of } i\text{th inner node} \\ 0 & \text{if } j\text{th leaf is } \textbf{not} \text{ on the path of } i\text{th inner node} \end{cases} \tag{2}$$

An algorithm for creating a matrix M given any depth is presented in Appendix B.

Therefore, if $M_{j,i}$ and $Z_{i,k}$ have the same sign, it follows that $\mathbf{x}_k$ took the correct path at the ith inner node to reach the jth leaf, and if they differ, $\mathbf{x}_k$ took a different path and cannot reach the jth leaf. By multiplying these two matrices, one obtains $MZ \in \mathbb{R}^{S_\ell \times N}$, a matrix in which the element $(MZ)_{j,k}$ corresponds to the total number of correct steps taken by the observation $\mathbf{x}_k$ with respect to arriving at the jth leaf. Consequently, $\mathbf{x}_k$ belongs to the jth leaf if $(MZ)_{j,k} = d$. Let ϕ be a clipping operator such that $\phi(x) = \max(0, \min(x, 1))$. Then $Q = \phi(MZ - (d-1))$ is a matrix of one-hot columns indicating the leaf index reached by each observation in $\mathcal{X}$.

Since Q already provides the assignment of observations to leaves, to obtain L^* one simply has to assign the class to each observation and then take the majority vote. This is done by concatenating S_ℓ repetitions of the y vector to construct a Y matrix of dimensions $(S_\ell \times N)$. Then the Hadamard product $Y \odot Q$ is applied to obtain a matrix where each row corresponds to a different leaf, such that the kth element of the jth row is either 0 (if $\mathbf{x}_k$ does not belong to the jth leaf) or y_k (if it does). This operation has the effect of "coloring" Q with the class of each observation. Finally, L^* can be obtained simply by selecting the most common element from each row of $Y \odot Q$, therefore effectively making the majority vote itself. To implement this using matrix operations, a matrix $R \in \mathbb{R}^{S_\ell \times C}$ can be constructed, where $R_{j,k}$ is the number of observations of class k that belong to the jth leaf. Given R, obtaining L^* simply involves taking the row-wise argmax of R. Note that R can be constructed either by methods such as `bincount` in `numpy`, or by using the following operation:

$$R_k^T = \left(\prod_{j=1; j \neq k}^{C} \frac{(j - Y \odot Q)}{(j-k)} \right) \mathbb{1} \tag{3}$$

Where R_k^T is the kth column of R (corresponding to class k), $\prod$ is the sequential application of the Hadamard product, and $\mathbb{1}$ is a one-column of size $(N \times 1)$.

Once L^* is obtained, the task of calculating the final accuracy is straightforward: because the jth row of R describes the number of elements of each class at the jth leaf, the maximum value in the row equals the number of correct classifications at said leaf. Therefore, by summing the maximum element for each row and dividing by the total number of observations N, one can find the accuracy of T on the dataset $(\mathcal{X}, y)$. As a result, the fitness of the tree is fully evaluated and its optimal leaves are known. The complete algorithm is shown in Algorithm 1.

Note that this matrix-based algorithm can be easily extended to a tensor-based form, in which a new dimension is added to represent different trees. With this alternative form, entire sets of trees can be evaluated all at once, reducing computational overhead by batching matrix operations. This is in stark contrast to the more immediate iterative implementation, which calculates the fitness of the tree set by iterating through every tree and calling the matrix-based evaluation for each. In particular, the *batch evaluation* approach fits naturally within an evolutionary computation environment, where each generation has a set of trees that can be evaluated completely independently. The impact of this batch implementation is analyzed more in-depth on Section 5.1.1.

3.2. A numerical example

Let $(\mathcal{X}, y)$ be a dataset with $N = 6$ observations, $P = 3$ attributes and $C = 2$ classes:

$$\mathcal{X} = \begin{bmatrix} 1 & -2 & -4 & -4 \\ 1 & -2 & -1 & -4 \\ 1 & 0 & 4 & 4 \\ 1 & 0 & 0 & 0 \\ 1 & -1 & -1 & -1 \\ 1 & 3 & -2 & -3 \end{bmatrix} \tag{4}$$

$$y = \begin{bmatrix} 1 & 1 & 2 & 1 & 2 & 1 \end{bmatrix} \tag{5}$$

Furthermore, let T be a complete tree of depth $d = 2$, with $S_i = 2^d - 1 = 3$ inner splits and $S_\ell = S_i + 1 = 4$ leaves. Its corresponding matrix $W \in \mathbb{R}^{S_i \times (P+1)}$ is defined as:

$$W = \begin{bmatrix} 0.1 & 0.2 & 0.8 & 0.8 \\ -0.4 & -0.1 & -0.9 & 0.7 \\ 0.7 & -0.4 & -0.9 & -0.4 \end{bmatrix} \tag{6}$$

Given its depth, the mask matrix M is:

$$M = \begin{bmatrix} -1 & -1 & 0 \\ -1 & +1 & 0 \\ +1 & 0 & -1 \\ +1 & 0 & +1 \end{bmatrix} \tag{7}$$

Then, the following intermediary matrices are:

$$Z = \sigma(W\mathcal{X}^T) = \begin{bmatrix} -1 & -1 & +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & -1 & -1 & -1 \\ +1 & +1 & -1 & +1 & +1 & +1 \end{bmatrix} \tag{8}$$

$$MZ = \begin{bmatrix} 0 & +2 & 0 & 0 & +2 & +2 \\ +2 & 0 & -2 & -2 & 0 & 0 \\ -2 & -2 & +2 & 0 & -2 & -2 \\ 0 & 0 & 0 & +2 & 0 & 0 \end{bmatrix} \tag{9}$$

$$Q = \phi(MZ - (d-1)) = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{10}$$

$$Y = \begin{bmatrix} 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 \end{bmatrix} \tag{11}$$

$$Y \odot Q = \begin{bmatrix} 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{12}$$

$$R = \begin{bmatrix} 2 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{13}$$

By taking an `argmax` of each row of R, one finds that the optimal leaf distribution is $L^* = \begin{bmatrix} 1 & 1 & 2 & 1 \end{bmatrix}$. Furthermore, by summing up the maximum of each row and dividing by N, the accuracy of T for $(\mathcal{X}, y)$ is calculated to be $\frac{5}{6} \approx 0.833$.

3.3. Analysis of time complexity

In this section, we give a brief big-O analysis of the time complexity of the matrix encoding. In particular, we are interested in the Algorithm 1, which computes both the optimal labels L^* of a given tree and its accuracy on a dataset $(\mathcal{X}, y)$. We leave the creation of mask M and of matrix Y out of our analysis, since they are independent of W and can be computed just once and then used for each generation. Furthermore,

we assume the use of a naive matrix multiplication algorithm, so that the operation AB between two matrices $A \in \mathbb{R}^{a\times b}$ and $B \in \mathbb{R}^{b\times c}$ has a time complexity of $O(abc)$. There are faster methods (such as the Strassen algorithm), but for the purposes of our analysis this is sufficient.

First, obtaining Z has a time complexity of $O(S_i PN)$, since it is dominated by the multiplication of $W \in \mathbb{R}^{S_i\times P}$ by $X \in \mathbb{R}^{N\times P}$. Note that both the sign function σ and the clipper function ϕ have a complexity of $O(1)$ when applied to a single value, since they check only a small number of conditionals (one and two, respectively).

Then, obtaining Q has a time complexity of $O(S_i S_\ell N)$, since it mainly involves the multiplication of $M \in \mathbb{R}^{S_\ell\times S_i}$ and $Z \in \mathbb{R}^{S_i\times N}$. The element-wise subtraction by $(d-1)$ does not affect the time complexity, similar to using the clipper function ϕ.

Next, the algorithm executes the bincount function ρ. To analyze this function, note that it starts by performing $\frac{(j-Y\odot Q)}{(j-\cdot)}$ for each class j (except one). The complexity of this operation is completely dominated by the Hadamard product $Y \odot Q$, which is $O(S_\ell N)$ since $(Y, Q) \in \mathbb{R}^{S_\ell\times N}$. Next, the $\prod$ function computes the Hadamard product of the $C-1$ matrices, which has complexity $O(S_\ell NC)$. Finally, it multiplies the result by a vector $\mathbb{1} \in \mathbb{R}^N$, leaving the final complexity of the ρ function as $O(S_\ell NC)$. Since R consists of a collection of C applications of ρ, obtaining R has a time complexity of $O(S_\ell NC^2)$.

The computation of L^* is just a row-wise argmax of R, so it has complexity $O(S_\ell C)$, since $R \in \mathbb{R}^{S_\ell\times C}$. Similarly, the precision can be calculated in $O(S_\ell C)$.

Summing up all the time complexities gives the final value:

$$\begin{aligned} O(\cdot) &= O(S_i PN) + O(S_i S_\ell N) + O(S_\ell NC^2) + O(S_\ell C) + O(S_\ell C) \\ &= O(S_i N(P + S_\ell)) + O(S_\ell NC^2) \end{aligned}$$

Assuming a complete tree of depth d, it follows that $S_i = 2^d - 1$ and $S_\ell = 2^d$, therefore:

$$O(\cdot) = O(2^d N(P + 2^d + C^2)) \tag{14}$$

Note that this time complexity is exponential in depth. This is in contrast to traditional tree encodings, which follow a linear time complexity of $O(Nd)$ (for more details, see Appendix A). Observe, however, that although the algorithmic complexity is greater, the implementation itself may be more efficient — this will be discussed further in Section 5.

Finally, we note that a detailed analysis of *memory complexity* is omitted, since both the proposed matrix-based encoding and the traditional tree implementation have low memory requirements. To illustrate this, note that the most memory-intensive step in the proposed method is the element-wise multiplication of matrices Y and Q, both of which have dimensions $(S_\ell \times N)$. Considering a very large dataset with 10,000,000 observations and a tree depth of 5, the memory complexity is on the order of $(2 \times 32 \times 10^7)$, which amounts to 2.56 GB of memory (assuming each element is a 4-byte integer). If the program is to be executed on a CPU, this memory requirement can be easily met by today's home PCs, while if the program is to be executed on a GPU, this requirement is also feasible, since modern GPUs are equipped with 8 GB to 12 GB of memory. Memory could only become a problem if a large number of trees were to be evaluated at once, such as in the tensor-based batch evaluation procedure described earlier; however, our experiments show that for large datasets there is no observable difference between the batch approach and the less memory-intensive usual approach (see Section 5.1.1). Given this discussion, we conclude that memory requirements are not a challenge for the proposed method, even for large datasets.

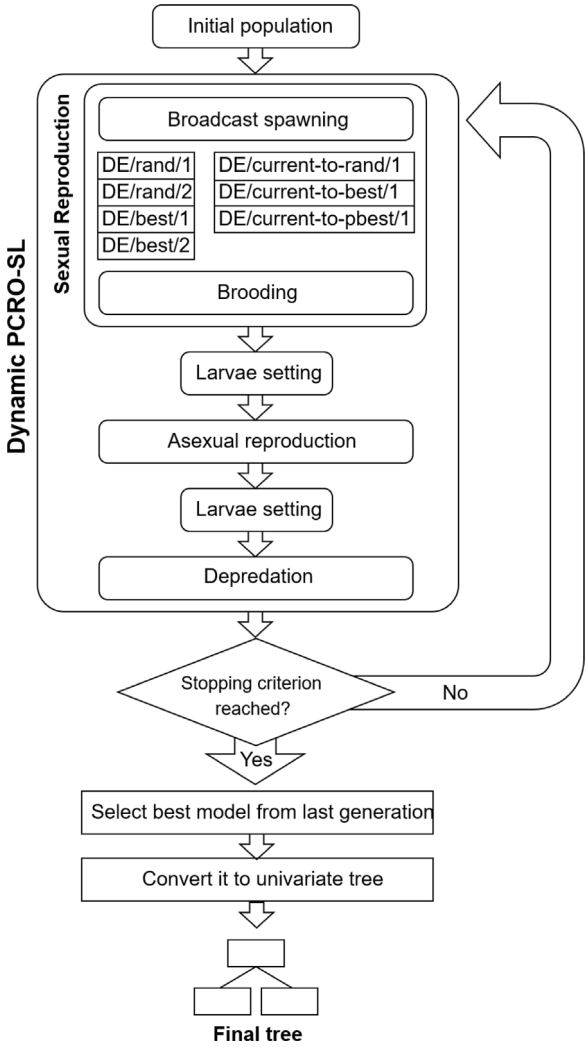


Fig. 2. Flowchart diagram of the CRO-DT algorithm.

4. Coral Reef Optimization for Decision Trees (CRO-DT)

In this section we describe the proposed algorithm CRO-DT, which uses the dynamic version of PCRO-SL to evolve classification trees. In general, CRO-DT follows a very similar structure to the PCRO-SL algorithm, with some modifications to adapt it to the evolution of DTs. Fig. 2 illustrates how CRO-DT works.

The algorithm is fed with an input dataset $(\mathcal{X}, y)$ and a depth d, the latter one being responsible for determining the size of the considered solutions (since each solution is encoded as a matrix $W \in \mathbb{R}^{(2^d-1)\times(P+1)}$). In this way, CRO-DT falls under the umbrella of the so-called "fixed structure" tree algorithms [2], which do not grow or prune the trees during execution. It is worth noting that fixing the tree structure is merely a design decision of CRO-DT, and does not necessarily follow from the use of matrix encoding. By using genetic operators capable of increasing the size of the matrices, it is possible to develop a fully matrix-based EDT that would also produce trees of varying depth. In CRO-DT, we have chosen to fix the tree structure because this has interesting consequences. For one, it still allows the algorithm to build trees of depth less than d, since by assigning the same label to sibling leaves, it is able to effectively eliminate their parent split. Second, when d is set to a sensible value, it reduces the risk of overfitting (because the tree is not allowed to grow and potentially fit the noise in the data, such as in traditional greedy algorithms). Finally,

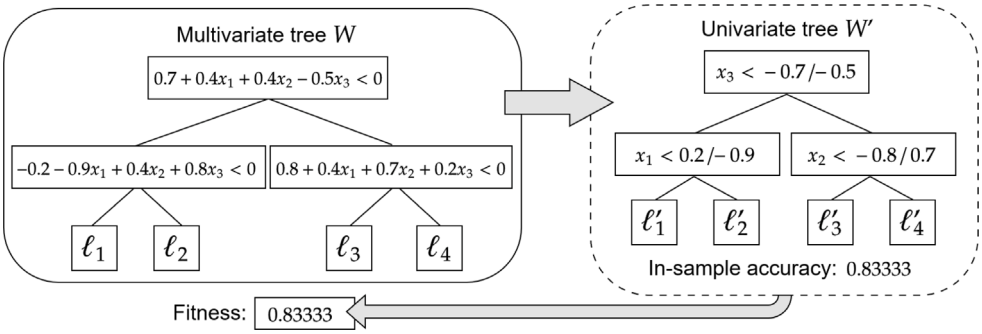


Fig. 3. Illustration of the mechanism that guarantees the evolution of univariate DTs.

it gives more control to the end user, who is able to adjust the depth d and tune the final results according to their own interpretability and performance concerns.

Besides the solution representation, a key decision when using PCRO-SL is the definition of its substrates: mechanisms for mutation and crossover that are executed during the broadcast spawning phase of the algorithm. Although substrates can be defined as any kind of mutation or crossover scheme, in CRO-DT they are all derived from the well-established Differential Evolution (DE) [47]. DE is a population-based metaheuristic that aims to evolve individuals represented as numerical vectors. Since our solutions are encoded as matrices, DE is a natural fit. At each iteration, solutions are updated by combining themselves with other solutions, such that every ith element in a solution T is updated as follows:

$$T_i = \begin{cases} V_i & \text{if rand}(0,1) \leq CR \\ T_i & \text{otherwise} \end{cases}$$

In which $CR \in [0,1]$ is a parameter that determines the probability of crossover between each element of the current solution T and a mutated solution V. The construction of V can be defined in several ways, each corresponding to a different substrate. In CRO-DT the following variants are used:

1. DE/rand/1:

$$V_i \leftarrow R1_i + F(R2_i - R3_i)$$

2. DE/rand/2:

$$V_i \leftarrow R1_i + F(R2_i - R3_i) + F(R4_i - R5_i)$$

3. DE/best/1:

$$V_i \leftarrow B_i + F(R1_i - R2_i)$$

4. DE/best/2:

$$V_i \leftarrow B_i + F(R1_i - R2_i) + F(R3_i - R4_i)$$

5. DE/current-to-rand/1:

$$V_i \leftarrow T_i + \text{rand}(0,1)(T_i - R1_i) + F(R2_i - R3_i)$$

6. DE/current-to-best/1:

$$V_i \leftarrow T_i + F(B_i - T_i) + F(R1_i - R2_i)$$

7. DE/current-to-pbest/1:

$$V_i \leftarrow T_i + F(PB_i - T_i) + F(R1_i - R2_i)$$

Here, F is a scaling factor, Rj is a random solution sampled from the population, B is the best solution in the population (i.e., highest fitness), and PB is a random solution from the top 10% in the population.

Although DE is well-suited for optimizing the matrix encoding proposed here, it should be noted that direct application of these substrates would cause CRO-DT to evolve multivariate DTs instead of the univariate ones that are considered interpretable. This is because that for a split to be univariate, its corresponding row in the matrix encoding must consist of all zeros, with the sole exceptions of the bias and the weight of a single attribute (corresponding to the selected attribute at said split). Since DE imposes no such restriction, virtually all solutions produced by its subtrates are guaranteed to be multivariate. One way to avoid this would be to abandon the use of DE and employ only custom-built genetic operators capable of preserving such structure, but in doing so one would lose the ability to use numerical optimization substrates like DE that already have a track record of producing high-quality solutions. Thus, CRO-DT takes an alternative approach: instead of defining fitness as the accuracy of the multivariate DT encoded in W, it defines it as the fitness of the closest univariate DT, which is denoted as W'. This is illustrated in Fig. 3 and can be defined as follows:

$$W'_{i,j} = \begin{cases} W_{i,j} & \text{if } j = 0 \text{ or } j = \arg\max_k \{|W_{i,k}|\}_{k=1}^{P+1} \\ 0 & \text{otherwise} \end{cases}$$

By using this mechanism, CRO-DT guarantees that it will move towards high quality univariate solutions, since a solution is only as good as its univariate counterpart. It is important to keep in mind that such univariate conversion is done only for the purpose of fitness evaluation, and that solutions retain their original multivariate encoding. From the point of view of evolutionary computation, we can say that the multivariate matrix encoding is the "genotype", while the univariate tree is the "phenotype". In fact, this scheme takes advantage of the good performance of numerical optimization algorithms (in particular, DE) to produce better solutions in a discrete space. Note also that if one does not include this technique, CRO-DT turns into a multivariate tree algorithm like OC1 [55] or OCT-H [31].

In all other aspects, CRO-DT closely follows the scheme dictated by dynamic PCRO-SL. After reaching a stopping criterion (which in our experiments was a maximum number of function evaluations), the algorithm selects the best model in the last generation and returns its corresponding univariate tree as the final solution.

5. Experimental design and results

In this section, three questions are addressed:

- How fast is the matrix encoding, when compared to the more traditional implementation of DTs? (Section 5.1)
- What is the impact of GPU usage and batch evaluation on the training time of the matrix-based approach? (Section 5.1.1)
- How accurate is CRO-DT, when compared to other interpretable DT algorithms? (Section 5.2)

To carry out the experiments, both synthetic and real data were employed. The synthetic data is used to evaluate training time and

Table 1
Datasets used throughout the paper. At the top, artificial datasets, while on the bottom, real datasets taken from UCI Machine Learning Repository [28].

Dataset name	# of observations (N)	# of attributes (P)	# of labels (C)
Artificial_1	100	2	3
Artificial_2	1000	2	3
Artificial_3	1000	10	3
Artificial_4	1000	10	10
Artificial_5	10 000	2	3
Artificial_6	10 000	10	3
Artificial_7	100 000	10	10
Breast cancer	683	9	2
Car evaluation	1728	6	4
Banknote auth.	1372	4	2
Balance scale	625	4	3
Acute inflammations 1	120	6	2
Acute inflammations 2	120	6	2
Blood transfusion	748	4	2
Climate crashes	540	18	2
Connectionist sonar	208	60	2
Optical recognition	3823	64	10
Drybeans	13 611	16	7
Avila bible	10 430	16	11
Wine quality red	1599	11	6
Wine quality white	4898	11	7

was generated using the `make_classification` routine in Scikit Learn [56]. Meanwhile, the real data is used to evaluate the accuracy and consists of 14 classification tasks from the widely used UCI Machine Learning Repository [28]. All datasets are described in Table 1. This distinction is made because artificial datasets allows for a more precise relationship between speedups and dataset parameters, since the computational cost of EDTs does not depend on the content of the dataset, but rather only on the values of N, P and C. On the other hand, the performance of a given machine learning algorithm depends largely on the contents of the dataset, so it is fundamental to evaluate the accuracy on real data. Furthermore, only datasets with numerical attributes were selected, as this is the domain to which the proposed matrix encoding lends itself more naturally. Nevertheless, categorical attributes could also be handled through techniques such as one-hot encoding.

All code was implemented using Python as base, since it is the programming language of choice for machine learning applications. In particular, the CRO-SL algorithm was implemented with the PyCRO-SL library [43], while the matrix-based encoding was implemented with TensorFlow [57]. In order to evaluate the computational cost of evolving EDTs using the proposed matrix approach, the traditional tree encoding was implemented, as described by Algorithm 2 in Appendix. Because TensorFlow uses C as a backend, we implemented this traditional tree encoding similarly using C, interfacing it with the Python code via the Cython framework. All experiments were performed on an Intel i9-12900 K 12th Gen with 32 GB of RAM and a Nvidia RTX 3080 GPU.[3]

5.1. Computational cost

To evaluate the computational cost of evolving EDTs using the proposed matrix encoding, we compare its training times with the ones obtained by the traditional tree encoding. In particular, the selected matrix-based approach employs both batch evaluation (described in Section 3) and GPU parallelization. For each of the artificial datasets described in Table 1, 5 runs of both encodings were carried out for depths ranging from 2 to 7, such that all runs were performed for a fixed number of 10^4 fitness evaluations (100 generations with 100 fitness evaluations each).

The results of the best configuration are shown in Table 2. In general, it can be seen that the matrix encoding outperforms the traditional encoding for all depths from 2 to 6, with speedups reaching close to 20 times. The sole exception is the smallest dataset evaluated ($N = 100$), for which the proposed approach obtains speedups up to depth 5.

Next, we analyze the results in more detail. A first observation that can be made is that there is little difference in training time when varying P or C: datasets Artificial_2, Artificial_3 and Artificial_4 always show very similar speedups, as do datasets Artificial_5 and Artificial_6. This is expected for the traditional tree encoding, since its time complexity is independent of P and C, but it is wholly unexpected for the proposed matrix encoding, as its time complexity includes both parameters. The reason why P and C do not seem to affect the matrix encoding is because they are largely overshadowed by the value of N, which exceeds them by several orders of magnitude in every dataset considered. Because of this difference in magnitude, matrix encoding times are effectively driven by the value of N, resulting in similar-looking results for these two sets of artificial data. While in theory it would be possible to generate datasets with larger values for P and C, real datasets behave as the ones displayed here, with a large number of observations and at most a few dozen attributes and/or classes. Therefore, in our analyses, we have refrained from experimenting with extremely large values of P or C, and decided to stick to more realistic values where we can see that these parameters have almost no impact on the training time.

Furthermore, it can clearly be seen that the speedups increase as the datasets get larger. This is evident in depth 2, where a small dataset ($N = 100$) has a negligible training time for both encodings, whereas a large dataset ($N = 100,000$) takes 4 min with the traditional implementation and just over 10 s with the proposed representation. Such a difference is quite intuitive. A large N means more iterations of the inference procedure, where small overheads in the traditional implementation accumulate into higher training times. Meanwhile, by exploiting the efficiency of numerical computation, the proposed matrix encoding greatly reduces this overhead and thus achieves better speedups at larger values of N.

Finally, we can see that the speedups decrease as the depth increases. For example, the matrix encoding offers a speedup of nearly 20 times on the Artificial_7 dataset for trees of depth 2, while it

[3] The source code is available at https://github.com/vgarciasc/CRO-DT.

Table 2
Average training time (in seconds) for tree and matrix encodings relating to DTs of depths 2 to 7, plus or minus standard deviation. The matrix results refer to the Batch GPU configuration, except for (*), which refers to the Iterative GPU configuration due to lack of memory in GPU.

Dataset name	Depth 2			Depth 3		
	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup
Artificial_1	0.4 ± 0.00	0.1 ± 0.00	5.1x	0.4 ± 0.00	0.1 ± 0.00	4.6x
Artificial_2	2.7 ± 0.01	0.2 ± 0.00	14.5x	2.7 ± 0.00	0.5 ± 0.00	6.1x
Artificial_3	2.7 ± 0.01	0.2 ± 0.00	14.5x	2.8 ± 0.00	0.5 ± 0.01	6.3x
Artificial_4	2.7 ± 0.01	0.2 ± 0.00	14.4x	2.8 ± 0.01	0.5 ± 0.00	6.1x
Artificial_5	24.9 ± 0.01	1.6 ± 0.01	15.7x	25.6 ± 0.02	3.1 ± 0.02	8.3x
Artificial_6	25.4 ± 0.01	1.6 ± 0.01	16.0x	26.5 ± 0.04	3.1 ± 0.02	8.5x
Artificial_7	250.4 ± 0.38	12.7 ± 0.29	19.7x	259.5 ± 0.07	26.6 ± 0.31	9.8x
Dataset name	Depth 4			Depth 5		
	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup
Artificial_1	0.4 ± 0.00	0.2 ± 0.01	2.7x	0.5 ± 0.00	0.3 ± 0.01	1.4x
Artificial_2	2.8 ± 0.01	0.8 ± 0.01	3.4x	2.9 ± 0.01	1.3 ± 0.01	2.2x
Artificial_3	2.9 ± 0.01	0.8 ± 0.03	3.4x	3.0 ± 0.01	1.3 ± 0.01	2.3x
Artificial_4	2.9 ± 0.01	0.8 ± 0.04	3.5x	3.0 ± 0.01	1.4 ± 0.02	2.2x
Artificial_5	26.1 ± 0.02	3.6 ± 0.05	7.3x	26.7 ± 0.02	7.8 ± 0.07	3.4x
Artificial_6	27.3 ± 0.02	3.6 ± 0.15	7.7x	28.2 ± 0.02	7.8 ± 0.04	3.6x
Artificial_7	268.2 ± 0.13	33.8 ± 0.88	7.9x	278.0 ± 0.14	70.8 ± 3.32	3.9x
Dataset name	Depth 6			Depth 7		
	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup	Tree encoding Avg. training time (s)	Matrix encoding Avg. training time (s)	Speedup
Artificial_1	0.5 ± 0.00	0.9 ± 0.02	0.5x	0.6 ± 0.00	1.2 ± 0.01	0.5x
Artificial_2	3.0 ± 0.00	1.8 ± 0.01	1.6x	3.1 ± 0.00	4.7 ± 0.01	0.7x
Artificial_3	3.2 ± 0.00	1.8 ± 0.06	1.7x	3.3 ± 0.00	4.7 ± 0.10	0.7x
Artificial_4	3.2 ± 0.00	1.8 ± 0.07	1.7x	3.3 ± 0.00	4.7 ± 0.08	0.7x
Artificial_5	27.3 ± 0.02	18.4 ± 0.14	1.5x	27.8 ± 0.02	50.8 ± 3.32	0.5x
Artificial_6	29.2 ± 0.01	18.4 ± 1.26	1.6x	30.1 ± 0.03	50.7 ± 1.18	0.6x
Artificial_7	286.8 ± 0.06	165.3 ± 0.93	1.7x	295.5 ± 0.21	443.6 ± 0.07*	0.7x

provides a speedup of only 1.7x on the same dataset for depth 6. This trend reaches its critical point at depth 7, where the traditional tree encoding actually outperforms the proposed matrix procedure for all datasets, becoming the faster representation for EDTs. This behavior can be easily explained by considering the time complexity analysis given in Section 3.3. Since the proposed method has an exponential complexity, its training time is bound to increase more than that of the traditional tree encoding, which has a linear complexity. Although this means that the proposed matrix encoding is only useful in some situations, we point out that these situations are very similar to those in which DTs are used in real life — DTs are often used for their interpretability, so their desired depth is usually very low. Indeed, at depth 7 (where the traditional tree implementation overtakes the proposed encoding), trees have 255 nodes and can hardly be called interpretable. In these situations, one would be better off employing other uninterpretable methods such as SVMs or ANNs, instead of DTs. As a result, we conclude that for most of the situations that matter, the matrix encoding does indeed provide significant speedups compared to the traditional implementation.

5.1.1. Impact of batch evaluation and GPU usage

As previously stated, the matrix-based implementation of Table 2 uses both batch evaluation and GPU processing. However, to fully understand the proposed approach, it is key to investigate if these two components are fundamental to the speedups observed, or merely auxiliary. Fig. 4 shows the results for all four possible combinations of these two components, through which it is possible to evaluate their individual impact.

First, we compare the results for the configurations without batch evaluation, which we call “Iterative CPU” and “Iterative GPU”. From Fig. 4, it can be seen that these two configurations are usually very close to one another, with the GPU only bringing occasional improvements: it is faster than the CPU when the matrices are large (more data, deeper trees), and slower when the matrices are small (fewer data, shallower trees). Only when the datasets are sufficiently large ($N \geq 10^4$) that the GPU implementation outperforms its CPU counterpart for all depths, which is a result typical of GPU implementations, since loading data on GPU memory has an overhead that must be compensated by the cost of the computations themselves. Furthermore, it can clearly be seen that for the first four datasets evaluated, these two configurations are actually always *slower* than the traditional encoding — possibly due to inherent costs related to TensorFlow usage, which are not being compensated by the small scale of the matrix operations. The balance shifts somewhere between $N = 10^3$ and $N = 10^4$, where these configurations start to offer benefits at small tree depths. In fact, for $N = 10^5$, the Iterative GPU configuration is faster than the traditional encoding for depths from 2 to 6, while its CPU counterpart is faster for depths from 2 to 5. As a whole, these results suggest that an iterative implementation of the matrix-based approach is not sufficient to guarantee widespread speedups, even with GPU usage.

Next, we compare the two configurations that employ batch evaluation, called “Batch CPU” and “Batch GPU”. The usage of batch evaluation should, in theory, reduce overheads related to matrix computation and speed up the process, which Fig. 4 shows that is clearly being achieved. Both batch configurations are faster than their iterative counterparts for every dataset analyzed, even being faster than the traditional encoding under the circumstances discussed in the previous section. Moreover, although GPU usage is almost always beneficial, it is not the driving force behind these speedups: even with a CPU implementation, the batch matrix-based procedure is able to outperform tree encoding, often being just 1 unit of tree depth behind its GPU counterpart (i.e. Batch GPU is often faster for ranges 2 to 6, while Batch CPU for depths 2 to 5). These results suggest that batch evaluation is crucial for the general speedups obtained by the matrix-based approach, being more important than GPU parallelization itself.

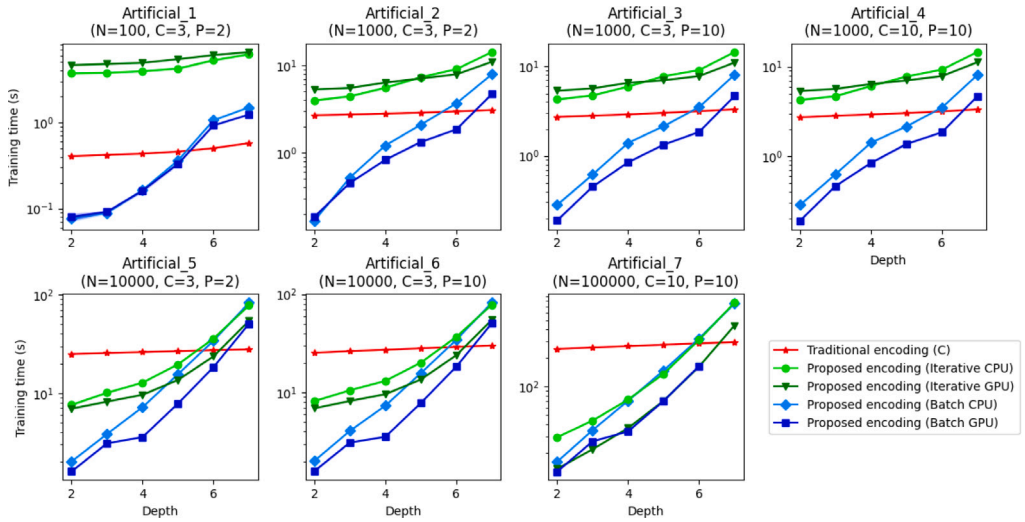


Fig. 4. Comparison of training time (in seconds) for tree encoding and matrix encoding configurations, evaluated on artificial datasets.

Interestingly, the benefits of batch evaluation diminish with increasing matrix size: for the largest depths of the largest dataset ($N = 10^5$), there is no visible difference between Batch CPU and Iterative CPU, or between Batch GPU and Iterative GPU. This is possibly due to the matrices being large enough that their computation effectively overshadows any overhead reduced by the inclusion of batch evaluation.

As a final point, we note that for the largest configuration evaluated ($N = 10^5$, $d = 7$), the Batch GPU configuration was unable to run since the graphics card did not have enough memory to calculate every individual at the same time. Given that this is exactly the situation in which the batch evaluation and the iterative approach have the same training time, we argue that in these situations the iterative approach should be used instead, without any additional computational cost.

5.2. Accuracy of CRO-DT

To assess the accuracy of the CRO-DT algorithm, we compared it with six other algorithms from four different lines of research. The algorithms were selected based not only on having a diversity of approaches, but also on code availability and the capacity of each algorithm to restrict depth. This last point is particularly relevant as CRO-DT is a fixed-structure DT algorithm, i.e. it takes a tree depth as input and does not exceed that depth throughout training. Therefore, in order to make a fair comparison, it would be necessary to compare CRO-DT with other trees of the same depth produced by other algorithms. After reviewing the literature and applying these criteria, the algorithms selected were CART [6], C4.5 [7], OCT [31], GOSDT [58], TAO [59], and CMA-ES [60].

- CART and C4.5 represent the traditional greedy top-down approach that is still widely used today. CART was implemented with the `scikit-learn` library [56] and C4.5 with the `imodels` library [61]. Both libraries allow the depth of the models to be restricted by pre-pruning, but although this is the most direct way, it is also known to produce lower quality solutions [62]. Since the CART implementation allows for post-pruning, we limited its depth using a more sophisticated approach: the pruning coefficient α is started at 0 and then increased by 0.005 until a tree of the desired depth (or less) is produced. This is the tree that is reported.
- GOSDT and OCT represent the optimal tree approach [63] that has recently received a lot of attention in DT research. For GOSDT, we used the available ensemble-guessing implementation [58], which is the latest development in a series of algorithms dating back to OSDT [64]. We used all the default parameters supplied with the software, along with a 0.001 regularization and a depth constraint. No code was available for OCT, but rather than not include this algorithm, we decided to report the results from the original paper as most of the datasets overlapped [31]. For comparison, the original OCT paper used the same dataset partitioning as we do (50%–25% – 25%), but they only ran 5 simulations for each result, whereas we run 100.
- TAO represents the line of standalone DT algorithms. We used the implementation for TAO available in the `imodels` package, with the arguments being the desired depth, a number of iterations of 100 (five times the default of 20 to encourage better solutions), and the remaining parameters as defaults. Note that the TAO implementation in `imodels` is restricted to producing univariate splits, so all the algorithms mentioned in this section produce univariate trees.
- Finally, CMA-ES represents the population-based line of research. This entry deviates from the others in the sense that CMA-ES is not a DT algorithm, but a more general evolution-based method that can evolve any kind of solution vector. This decision was made because, to the best of our knowledge, there is no published EDT that meets the requirements established above. In this context, we used the CMA-ES implementation from the `pycma` [65] library (which is actively maintained by the algorithm's original author) to evolve matrix-based DTs, as in CRO-DT. All default parameters were employed, while the number of evaluations was fixed to be the same as in CRO-DT, so as to make comparisons fair.

The metric used to compare these algorithms was out-of-sample accuracy. Although accuracy is not always the best measure to use (especially when the datasets are unbalanced), it is often employed as a standard comparison metric, and it is also easy to understand and reproduce. In addition, all reported accuracies are the average of 100 different simulations, so that in each simulation the datasets are split 50%–25%–25% for training-validation-test. During training, the algorithms have access to both training and validation sets, and their performance on the test set is reported.

Note that we do not compare the training times of the algorithms. This is because the goal is for CRO-DT to surpass these algorithms in

Table 3
Parameters employed in both CRO-DT and CRO-DT (CS).

Parameter name	Parameter value
Population size	200
ρ_0	0.8
F_b	0.98
F_d	0.1
P_d	0.4
k	3
F	1
CR	0.8
Maximum number of evaluations	10^6

terms of performance, not computational cost; it is impossible for an evolutionary algorithm like CRO-DT to reach the speed of a greedy algorithm like CART, nor is it our intention to do so. For an idea of how long CRO-DT takes to run for a certain dataset, we refer to Section 5.1, where these training times are extensively reported for a variety of configurations.

Two versions of CRO-DT are presented here for evaluation: CRO-DT and CRO-DT (CS). The only difference between them is that CRO-DT starts with a random population of trees, while CRO-DT (CS) is initialized with CART trees (CS stands for CART Seeding). This initialization was done as follows: one third of the initial population consisted of CART trees trained on the same training set, another third consisted of the same trees but mutated (using a simple Gaussian perturbation), and the last third consisted of completely random trees. This seeding ensures that CRO-DT starts with solutions as good as CART, while having enough diversity in the population to avoid getting stuck in the same local optima as its greedy counterpart. This idea of seeding the initial population of an EDT with greedy trees can also be found in [66–68].

Because both CRO-DT and CRO-DT (CS) are based on CRO-SL, they have several parameters that control different aspects of the evolutionary algorithm. These parameters include spawning rate, predation rate, and predation probability, among others. As is common in the field, we determined them by conducting preliminary experiments to compare different sets of parameterizations, each defined based on the authors' previous experience. The final parameterization chosen for CRO-DT is shown in Table 3, with the substrates defined in Section 2.2.

Table 4 shows the performance of all 8 algorithms considering trees of depth 2, with their corresponding rankings in parentheses (with tied algorithms receiving their mean ranking). First of all, it is worth noting that no algorithm takes the absolute first place, as they all shine on some datasets while losing out on others. However, the ranking gives us valuable information: even though CRO-DT (CS) reports the best test accuracy on only 5 of the 14 datasets, its average ranking is the highest of all algorithms (2.6), with CRO-DT following close behind (3.5). This indicates that both algorithms produce consistently good solutions, which is a highly desirable property for off-the-shelf classification models. Furthermore, we can note that CRO-DT (CS) has the lowest standard deviation in ranking, which makes its consistency stand out even more. Moreover, it can be noted that there are some datasets where CART outperforms CRO-DT (CS) (e.g., Car evaluation, Blood transfusion), which may sound unexpected since CRO-DT (CS) is initialized with CART solutions and therefore should, in principle, produce equal or greater accuracies than CART. However, we argue that such behavior can be easily explained as overfitting. Indeed, Table C.7 in the Appendix shows that the training accuracy of CRO-DT (CS) is always greater than that of CART, which is what the initialization effectively guarantees. A similar reasoning can explain why CRO-DT sometimes has higher accuracy than CRO-DT (CS), although the average ranking shows that CART seeding has a clear positive impact on the algorithm. In general, since the two highest ranked algorithms are the CRO-DT variants, it can be said that under the conditions considered, the proposed algorithm tends to outperform others in the construction of extremely shallow trees.

Tables 5 and 6 show the results for depths 3 and 4. In both cases, CRO-DT does not perform as well as before, as it respectively achieves the 4th and 5th highest average rankings. However, CRO-DT (CS) still manages to maintain its position as the top performer, having both the highest average ranking and the lowest standard deviation for the two depths. This cements its place as a competitive alternative to its greedy and optimal counterparts. Furthermore, the growing discrepancy between CRO-DT and CRO-DT (CS) effectively demonstrates the general importance of a good initialization for evolutionary algorithms, while the difference between CRO-DT (CS) and CART shows the value of evolutionary algorithms as fine-tuning approaches capable of going beyond their initial solutions. Finally, it should be noted that just as before, there are some datasets where CART outperforms CRO-DT (CS) (e.g. Climate crashes for depth 3 and Blood transfusion for depth 4). However, this can be explained by the same reasoning as the one proposed for depth 2.

Fig. 5 summarizes the information from Tables 4–6, highlighting the effect of depth. Again, we can see that CRO-DT (CS) is consistently among the best solutions, although rarely the best. Most interestingly, the figure highlights that increasing depth sometimes has a negative impact on test accuracy (e.g. CRO-DT (CS) on the Climate crashes dataset), which is not intuitive since a tree of depth $d+1$ can represent anything that a tree of depth d is able to. We propose two mutually exclusive explanations for this phenomenon. The first is that the algorithm has trouble navigating the larger search spaces of deeper trees, and therefore finds worse solutions. The second is that with a higher number of parameters, the algorithm has more degrees of freedom and sometimes produces models that overfit. Although both explanations are theoretically plausible, we can see from Fig. C.6 in the Appendix that the training accuracy is in fact always stagnant or increasing, thus giving more credence to the overfit hypothesis.

Finally, we point out that CRO-DT (CS) does not get stuck in the local optima with which it is initialized. Tables 4–6 show that among the 42 dataset depth configurations evaluated, there is none where CRO-DT (CS) reports the same result as CART (we exclude configurations where both reached 100% accuracy). Although these results represent an average behavior over 100 runs, and a more careful verification would involve comparing each individual run, the general superiority of CRO-DT (CS) over CART serves as a strong indication that CRO-DT (CS) more often than not improves on the local optima with which it is initialized. The final solutions may indeed be different local optima, but in this case they are optima of high quality, since CRO-DT (CS) has been shown to outperform its alternative approaches on average over all depths.

6. Conclusions and future work

In this paper, we have proposed an encoding for decision trees that allows their evaluation via a sequence of matrix operations. The proposed encoding has been shown to be faster than the traditional tree implementation for complete trees with depths ranging from 2 to 6, and for datasets with sizes ranging from 100 to 100,000 observations. In the considered scenarios, speedups of nearly up to 20 times were obtained, with the proposed method shining when the datasets are large and the desired trees are small. Since this combination is increasingly common in the era of Big Data and given the broader trends towards interpretable models, we find that the proposed encoding is widely useful in real-world scenarios. Furthermore, we have shown that these speedups owe in large part to the application of techniques that cannot be applied to the traditional tree implementation (i.e. batch evaluation and parallelization), which exemplify that there are benefits particular to a matrix-based tree evaluation approach.

Table 4
Out-of-sample accuracy for trees with depth 2. The highest average accuracy for each dataset is highlighted in bold, and the average ranking is between parentheses. (*): As reported in [31].

Dataset name	OCT*	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	94.5 (3)	93.8 ± 1.7 (6)	93.1 ± 2.0 (7)	95.3 ± 1.8 (2)	94.2 ± 1.6 (4)	81.3 ± 14.1 (8)	93.9 ± 2.0 (5)	**95.5 ± 1.8 (1)**
Car evaluation	73.7 (7)	77.8 ± 1.3 (2.5)	**79.1 ± 1.3 (1)**	77.6 ± 1.3 (4.5)	77.8 ± 1.3 (2.5)	71.0 ± 2.7 (8)	77.6 ± 1.3 (4.5)	77.5 ± 1.3 (6)
Banknote auth.	90.1 (3)	89.4 ± 1.7 (5)	88.7 ± 1.5 (6)	**91.1 ± 1.3 (1)**	89.8 ± 1.6 (4)	86.3 ± 2.1 (8)	88.5 ± 2.5 (7)	90.4 ± 2.0 (2)
Balance scale	67.1 (4)	64.7 ± 3.5 (7)	61.5 ± 3.8 (8)	69.1 ± 2.6 (3)	66.4 ± 3.6 (5)	65.5 ± 7.4 (6)	**69.2 ± 2.5 (1.5)**	**69.2 ± 2.3 (1.5)**
Acute inflam. 1	**100.0 (2)**	86.7 ± 8.3 (8)	97.3 ± 4.9 (5)	99.7 ± 2.1 (4)	91.2 ± 5.1 (7)	93.3 ± 6.7 (6)	**100.0 ± 0.0 (2)**	**100.0 ± 0.0 (2)**
Acute inflam. 2	**100.0 (1.5)**	99.0 ± 2.6 (4.5)	98.8 ± 3.3 (7)	98.9 ± 2.8 (6)	99.0 ± 2.6 (4.5)	96.5 ± 7.4 (8)	**100.0 ± 0.0 (1.5)**	99.6 ± 2.7 (3)
Blood transfusion	75.5 (6.5)	76.1 ± 0.7 (2)	75.4 ± 1.8 (8)	75.5 ± 1.8 (6.5)	75.6 ± 1.7 (5)	**76.3 ± 0.3 (1)**	75.9 ± 1.3 (3)	75.7 ± 1.7 (4)
Climate crashes	90.5 (7)	91.3 ± 1.2 (4.5)	90.1 ± 1.4 (8)	**91.9 ± 1.8 (1)**	90.6 ± 2.2 (6)	91.5 ± 0.4 (2.5)	91.5 ± 2.0 (2.5)	91.3 ± 2.2 (4.5)
Conn. sonar	71.2 (3)	69.4 ± 5.8 (6)	66.4 ± 6.5 (8)	69.5 ± 5.9 (5)	68.7 ± 6.0 (7)	72.4 ± 5.7 (2)	**72.8 ± 5.9 (1)**	70.4 ± 5.8 (4)
Optical recognition	29.4 (6)	25.0 ± 6.5 (8)	25.8 ± 3.3 (7)	**38.4 ± 0.3 (1)**	31.5 ± 3.0 (5)	34.9 ± 0.7 (3)	34.7 ± 0.8 (4)	38.0 ± 0.5 (2)
Drybeans	–	55.1 ± 4.8 (6)	60.3 ± 6.4 (4)	62.1 ± 1.4 (3)	**65.2 ± 0.6 (1)**	50.1 ± 10.7 (7)	56.9 ± 2.8 (5)	64.4 ± 0.9 (2)
Avila bible	–	50.2 ± 0.5 (7)	50.8 ± 0.5 (5)	50.5 ± 2.2 (6)	51.0 ± 0.5 (4)	53.0 ± 0.8 (3)	**53.6 ± 0.4 (1.5)**	**53.6 ± 0.4 (1.5)**
Wine quality red	–	55.2 ± 1.8 (2)	54.4 ± 2.7 (4)	50.9 ± 5.6 (6)	**55.0 ± 2.1 (3)**	50.1 ± 3.2 (7)	51.0 ± 2.1 (5)	**55.8 ± 2.2 (1)**
Wine quality white	–	51.1 ± 1.1 (3)	49.2 ± 2.5 (4)	46.7 ± 2.5 (7)	**51.5 ± 1.3 (1)**	46.9 ± 1.5 (6)	48.3 ± 1.3 (5)	51.4 ± 1.2 (2)
Average rank	4.3 ± 2.1	5.1 ± 2.1	5.9 ± 2.1	4.0 ± 2.2	4.2 ± 1.9	5.4 ± 2.5	3.5 ± 1.8	2.6 ± 1.5

Table 5
Out-of-sample accuracy for trees with depth 3. The highest average accuracy for each dataset is highlighted in bold, and the average ranking is between parentheses. (*): As reported in [31].

Dataset name	OCT*	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	**95.3 (1.5)**	94.7 ± 1.7 (4.5)	94.6 ± 1.8 (6)	95.2 ± 1.6 (3)	94.7 ± 1.6 (4.5)	87.5 ± 11.8 (8)	94.1 ± 1.8 (7)	**95.3 ± 1.7 (1.5)**
Car evaluation	77.4 (7)	77.8 ± 1.3 (6)	**85.8 ± 1.0 (1)**	80.9 ± 1.2 (2)	79.3 ± 1.5 (5)	75.2 ± 5.1 (8)	80.5 ± 1.3 (3)	80.3 ± 1.3 (4)
Banknote auth.	89.6 (8)	91.8 ± 2.2 (4)	91.5 ± 2.3 (5.5)	**96.5 ± 1.1 (1)**	94.9 ± 1.4 (2)	90.6 ± 1.9 (7)	91.5 ± 2.3 (5.5)	92.4 ± 2.3 (3)
Balance scale	68.9 (6)	68.8 ± 3.2 (7)	61.8 ± 3.8 (8)	72.8 ± 2.5 (2)	71.7 ± 2.9 (4)	71.4 ± 3.6 (5)	72.6 ± 2.8 (3)	**73.0 ± 2.6 (1)**
Acute inflam. 1	**100.0 (3)**	**100.0 ± 0.0 (3)**	98.5 ± 3.2 (8)	99.7 ± 2.1 (6.5)	99.7 ± 1.3 (6.5)	**100.0 ± 0.0 (3)**	**100.0 ± 0.0 (3)**	**100.0 ± 0.0 (3)**
Acute inflam. 2	**100.0 (2.5)**	99.0 ± 2.6 (5.5)	98.8 ± 3.3 (8)	98.9 ± 2.8 (7)	99.0 ± 2.6 (5.5)	**100.0 ± 0.0 (2.5)**	**100.0 ± 0.0 (2.5)**	**100.0 ± 0.0 (2.5)**
Blood transfusion	77.0 (2)	76.5 ± 1.4 (4)	75.1 ± 2.2 (8)	**77.8 ± 2.1 (1)**	76.9 ± 2.1 (3)	76.3 ± 0.3 (5)	75.6 ± 1.6 (7)	76.1 ± 1.9 (6)
Climate crashes	91.4 (4)	91.6 ± 1.5 (2.5)	88.9 ± 2.5 (8)	**91.9 ± 1.8 (1)**	90.5 ± 2.3 (6.5)	91.6 ± 0.7 (2.5)	90.5 ± 2.4 (6.5)	91.1 ± 2.0 (5)
Conn. sonar	69.6 (7)	70.1 ± 6.3 (6)	66.3 ± 6.4 (8)	71.2 ± 6.1 (3)	70.9 ± 6.1 (4)	**74.5 ± 5.1 (1)**	71.4 ± 5.7 (2)	70.6 ± 7.2 (5)
Optical recognition	41.6 (6)	34.6 ± 8.0 (7)	28.3 ± 4.5 (8)	**59.9 ± 2.4 (1)**	49.9 ± 4.8 (4.5)	49.9 ± 2.6 (4.5)	50.0 ± 3.7 (3)	53.8 ± 1.9 (2)
Drybeans	–	76.6 ± 1.4 (3)	63.3 ± 4.6 (6)	77.5 ± 2.5 (2)	**78.2 ± 1.2 (1)**	60.7 ± 3.9 (7)	65.2 ± 3.4 (5)	74.3 ± 3.3 (4)
Avila bible	–	52.8 ± 0.6 (6)	51.1 ± 0.7 (7)	53.0 ± 2.2 (5)	53.6 ± 0.6 (4)	55.5 ± 0.8 (3)	56.9 ± 0.8 (2)	**57.0 ± 0.6 (1)**
Wine quality red	–	55.3 ± 2.1 (3)	54.8 ± 2.0 (4)	51.1 ± 5.4 (7)	**56.1 ± 2.3 (1)**	51.9 ± 2.1 (5)	51.4 ± 2.2 (6)	55.7 ± 2.4 (2)
Wine quality white	–	51.9 ± 1.3 (2)	49.3 ± 2.6 (4)	46.9 ± 2.5 (7)	**52.0 ± 1.5 (1)**	48.6 ± 1.5 (6)	48.7 ± 0.9 (5)	50.9 ± 1.4 (3)
Average rank	4.7 ± 2.4	4.5 ± 1.7	6.4 ± 2.1	3.5 ± 2.5	3.8 ± 1.9	4.8 ± 2.2	4.3 ± 1.9	3.1 ± 1.6

Table 6
Out-of-sample accuracy for trees with depth 4. The highest average accuracy for each dataset is highlighted in bold, and the average ranking is between parentheses. (*): As reported in [31].

Dataset name	OCT*	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	**95.3 (1.5)**	94.7 ± 1.8 (4.5)	94.4 ± 2.0 (6)	**95.3 ± 1.7 (1.5)**	94.7 ± 1.7 (4.5)	92.7 ± 6.6 (8)	94.3 ± 1.7 (7)	95.2 ± 1.6 (3)
Car evaluation	78.8 (8)	84.3 ± 1.4 (6)	**87.9 ± 1.0 (1)**	86.4 ± 1.3 (2)	84.5 ± 1.5 (5)	82.1 ± 5.8 (7)	85.5 ± 1.6 (4)	86.1 ± 1.3 (3)
Banknote auth.	90.7 (8)	93.6 ± 2.2 (6)	97.1 ± 1.3 (2)	**97.8 ± 0.9 (1)**	96.6 ± 1.3 (3)	93.0 ± 2.1 (7)	94.5 ± 2.3 (5)	95.2 ± 2.2 (4)
Balance scale	71.6 (7)	74.9 ± 3.6 (5)	63.9 ± 4.0 (8)	**78.3 ± 2.6 (1)**	77.4 ± 3.1 (3)	74.4 ± 3.6 (6)	76.5 ± 3.1 (4)	77.8 ± 3.0 (2)
Acute inflam. 1	**100.0 (3)**	**100.0 ± 0.0 (3)**	98.5 ± 3.2 (8)	99.7 ± 2.1 (6.5)	99.7 ± 1.2 (6.5)	**100.0 ± 0.0 (3)**	**100.0 ± 0.0 (3)**	**100.0 ± 0.0 (3)**
Acute inflam. 2	**100.0 (2)**	99.0 ± 2.6 (5.5)	98.8 ± 3.3 (8)	98.9 ± 2.8 (7)	99.0 ± 2.6 (5.5)	**100.0 ± 0.0 (2)**	**100.0 ± 0.0 (2)**	99.8 ± 1.3 (4)
Blood transfusion	77.0 (3)	77.1 ± 1.8 (2)	74.4 ± 2.7 (8)	**77.3 ± 2.3 (1)**	76.7 ± 2.2 (4)	76.3 ± 0.3 (5)	75.8 ± 1.5 (7)	76.1 ± 2.2 (6)
Climate crashes	91.4 (4)	91.8 ± 1.8 (3)	88.5 ± 2.8 (8)	**92.3 ± 1.8 (1)**	90.1 ± 2.6 (6)	91.9 ± 0.9 (2)	89.8 ± 2.4 (7)	90.7 ± 2.6 (5)
Conn. sonar	69.6 (7)	70.6 ± 6.6 (6)	66.4 ± 6.4 (8)	71.6 ± 6.4 (4)	70.9 ± 5.8 (5)	72.1 ± 5.9 (2)	**72.6 ± 5.7 (1)**	71.7 ± 6.7 (3)
Optical recognition	54.7 (5)	53.2 ± 3.2 (6)	30.3 ± 4.8 (8)	63.6 ± 5.2 (3)	64.6 ± 6.5 (2)	51.2 ± 3.2 (7)	60.0 ± 3.1 (4)	**65.2 ± 2.0 (1)**
Drybeans	–	80.5 ± 1.9 (2)	64.1 ± 3.7 (7)	65.1 ± 5.9 (6)	**83.2 ± 1.5 (1)**	66.7 ± 3.4 (5)	71.5 ± 4.8 (4)	77.9 ± 4.7 (3)
Avila bible	–	54.0 ± 1.3 (5)	51.7 ± 1.0 (7)	53.7 ± 1.8 (6)	55.8 ± 0.8 (4)	58.5 ± 1.0 (3)	59.4 ± 1.0 (2)	**59.6 ± 0.7 (1)**
Wine quality red	–	55.9 ± 2.3 (2)	54.7 ± 2.2 (4)	50.8 ± 5.2 (7)	**56.9 ± 2.5 (1)**	52.6 ± 1.8 (5)	51.5 ± 2.3 (6)	54.8 ± 2.9 (3)
Wine quality white	–	52.0 ± 1.3 (2)	49.5 ± 2.8 (4)	47.1 ± 2.6 (7)	**52.3 ± 1.4 (1)**	49.2 ± 1.2 (6)	49.4 ± 1.0 (5)	50.7 ± 1.4 (3)
Average rank	4.9 ± 2.5	4.1 ± 1.7	6.2 ± 2.5	3.9 ± 2.6	3.7 ± 1.9	4.9 ± 2.1	4.4 ± 1.9	3.1 ± 1.4

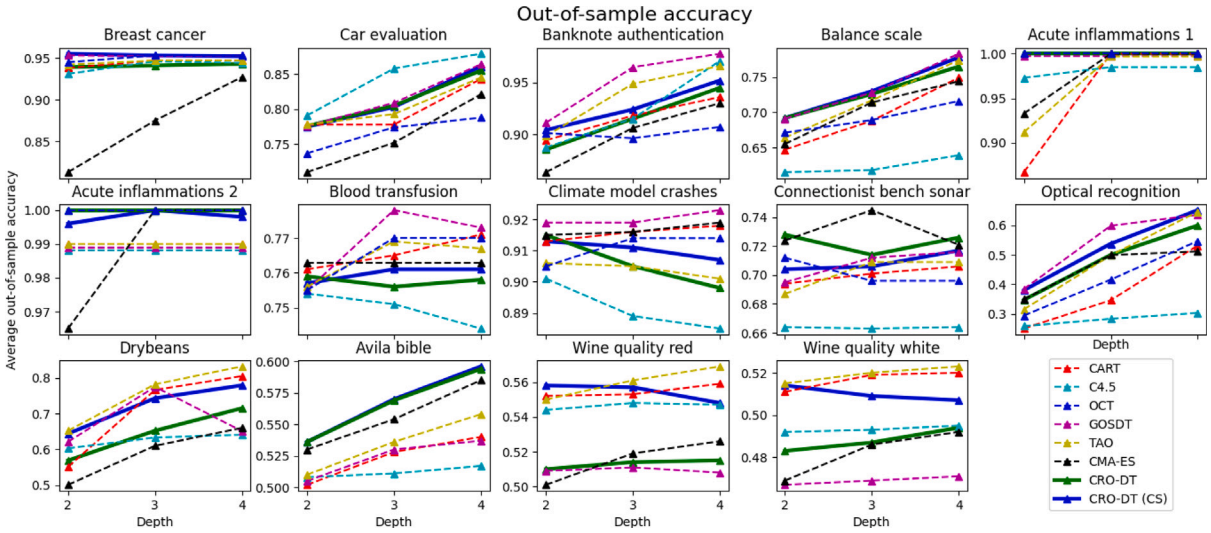


Fig. 5. Out-of-sample accuracies for different DT algorithms over the datasets.

To evaluate how the benefits of this encoding extend to accuracy, we also proposed CRO-DT, an algorithm that evolves decision trees using the Probabilistic Coral Reef Optimization with Substrate Layers algorithm. Although CRO-DT did not often have the best accuracy, it is noteworthy that when initialized with CART solutions, it had the highest average rank and lowest standard deviation for all depths and datasets analyzed. This suggests that it consistently produces good solutions — something that is particularly useful in real-world scenarios where not much is known about the domain.

Finally, as future work, we point out a number of possibilities for extending what has been proposed in this paper. For example, the matrix encoding could be extended to store the results of identical nodes in the population and calculate each of them only once, therefore reducing evaluation time. The encoding could also be adapted to include measures other than accuracy, such as F1 score or cost-sensitive measures. There are also several avenues to explore in CRO-DT, such as using different substrates (other than the Differential Evolution ones that we have used) and even different initialization methods (such as using GOSDT instead of CART for the initial population). An interesting approach would be to eschew interpretability and evaluate the value of CRO-DT as a multivariate tree algorithm, such as OC1 and OCT-H, as well as the speedups obtained by the matrix-based approach in a multivariate environment.

CRediT authorship contribution statement

Vinícius G. Costa: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Sancho Salcedo-Sanz:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision. **Carlos E. Pedreira:** Conceptualization, Methodology, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Carlos E. Pedreira reports financial support was provided by National Council for Scientific and Technological Development. Carlos E. Pedreira reports financial support was provided by Carlos Chagas Filho Foundation for Research Support of Rio de Janeiro State. Vinícius G. Costa reports financial support was provided by Coordination of Higher Education Personnel Improvement. Sancho Salcedo-Sanz reports financial support was provided by Spain Ministry of Science and Innovation.

Data availability

The source code's repository is mentioned in the Experiments section.

Acknowledgments

We are grateful to the anonymous reviewers for carefully reading the manuscript and offering valuable feedback, which greatly improved the quality of this paper and the fairness of the results obtained. We also thank Alberto Palomo-Alonso for help with the TensorFlow implementation.

Funding

This work was supported in part by the Brazilian research agencies CNPq—National Council for Scientific and Technological Development (Grant Number 306258/2019-6); FAPERJ—Foundation, Brazil for Research Support of Rio de Janeiro State (Grant Number E-26/200.840/2021); Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), Brazil - Finance Code 001; and by project PID2020-115454GB-C21595 of the Spanish Ministry of Science and Innovation (MICINN).

Appendix A. Traditional implementations of Decision Trees

To emphasize the uniqueness of the proposed matrix encoding, in this section we describe how DTs are typically implemented in programming environments. In particular, we are interested in the *inference* procedure, where an observation is assigned to a leaf after being tested by a sequence of splits.

Given the hierarchical and sequential nature of DTs, the most natural approach is to use a sequence of `if` statements not unlike those found in the structure of the tree itself. Starting at the root, the following process can be executed: if the logical test is satisfied, pass control to the left child node, otherwise pass it to the right child node. This process is repeated until a leaf node gets control of the procedure, at which point its label is returned and the inference process is marked as done. To the best of our knowledge, this is the approach taken by all popular implementations of DT algorithms, such as the widely used Scikit-Learn [56], Weka's J48 [69], and even Ross Quinlan's original C4.5 code [7].

Algorithm 2 contains a pseudocode of the inference process as described above. Assuming a tree of depth d, the time complexity of

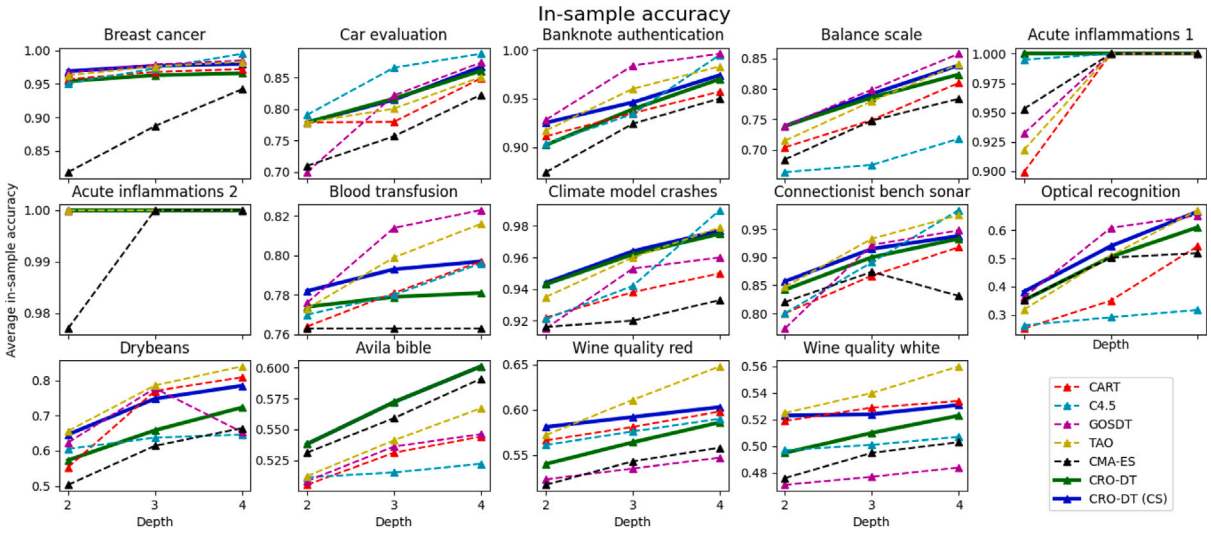


Fig. C.6. In-sample accuracies for different DT algorithms over the datasets.

this algorithm is $O(d)$, since it evaluates d conditions to infer the label of an observation. If the goal is to evaluate an entire dataset consisting of N observations, the complexity is simply $O(Nd)$.

Furthermore, note that this algorithm can be implemented either with a recursive approach (where the inference routine is called each time) or with a `while` loop (where the stopping condition would be that the current node is a leaf). Nevertheless, it remains essentially the same. Because recursive procedures are usually more expensive to implement than iterative approaches, all implementations of the traditional encoding in this paper use the latter method.

Algorithm 2 TreeInference(**x**, Node) : Illustration of DT traditional implementation

```
if IsLeaf(Node) then
    return Node.label
else
    if x[Node.attribute_index] ≤ Node.threshold then
        return TreeInference(x, Node.left)
    else
        return TreeInference(x, Node.right)
    end if
end if
```

Appendix B. Producing the mask matrix M

In Section 3 of the main text, a matrix encoding for DTs was proposed. This encoding employs a mask matrix M that, although easy to describe, is not immediate to produce. With the goal of making the paper self-contained, Algorithm 3 includes the pseudocode for generating this mask matrix for any depth d.

Appendix C. In-sample results for CRO-DT

In this section of the Appendix, we report the in-sample results for CRO-DT and CRO-DT (CS). Table C.7 summarizes the training data for all depths, while Fig. C.6 displays it visually.

Algorithm 3 CreateMask($\mathcal{X}, y, W, S_\ell, S_i, d$): creating the mask matrix M

```
M ← a matrix of zeroes of size (S_ℓ × S_i)

for j = 1 ... d do
    M_{1,j} ← −1                ▷ Initialize first row with path to leftmost leaf
end for

for i = 2 ... S_ℓ do
    M_i ← M_{i−1}               ▷ Initialize current row with previous row
    last_pos ← 0
    for j = 1 ... S_i do
        if M_{i−1,last_pos} = −1 then
            M_{i,last_pos} ← +1  ▷ Inverts last element if it is not inverted
            continue
        end if
    end for

    inv_count ← 0
    for j = S_i ... 1 do
        if M_{i−1,j} = +1 then
            M_{i,j} ← 0
            if last_pos + 1 + inv_count < S_i then
                M_{i,last_pos+1+inv_count} ← −1      ▷ Invert last element
                inv_count ← inv_count + 1            ▷ and continue inverting
            end if
        else
            break
        end if
    end for

    for j = last_pos ... 1 do
        if M_{i,j} = −1 then
            M_{i,j} ← +1
        end if
    end for
end for
return M
```

Table C.7
In-sample accuracy for trees with depths from 2 to 4. The highest average accuracy for each dataset is highlighted in bold, and the average ranking is between parentheses. OCT values are not included since training values are not reported in the original paper.

	Depth 2						
Dataset name	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	95.7 ± 1.3 (4)	95.1 ± 1.4 (6)	**96.9 ± 0.7 (1.5)**	96.3 ± 0.7 (3)	81.8 ± 14.3 (7)	95.4 ± 0.9 (5)	**96.9 ± 0.6 (1.5)**
Car evaluation	77.9 ± 0.7 (4)	**79.1 ± 0.6 (1)**	77.9 ± 0.7 (4)	77.9 ± 0.7 (4)	71.0 ± 2.6 (7)	77.9 ± 0.7 (4)	77.9 ± 0.7 (4)
Banknote auth.	91.1 ± 1.3 (4)	90.3 ± 0.9 (5)	**93.1 ± 0.6 (1)**	91.7 ± 1.1 (3)	87.4 ± 1.3 (7)	90.2 ± 1.9 (6)	92.5 ± 1.0 (2)
Balance scale	70.4 ± 2.8 (5)	66.3 ± 4.2 (7)	**73.9 ± 1.1 (2)**	71.5 ± 2.5 (4)	68.4 ± 8.0 (6)	**73.9 ± 1.1 (2)**	**73.9 ± 1.1 (2)**
Acute inflam. 1	89.9 ± 4.5 (7)	99.5 ± 1.6 (3)	93.2 ± 2.0 (5)	91.8 ± 2.5 (6)	95.3 ± 4.1 (4)	**100.0 ± 0.0 (1.5)**	**100.0 ± 0.0 (1.5)**
Acute inflam. 2	**100.0 ± 0.0 (3.5)**	**100.0 ± 0.0 (3.5)**	**100.0 ± 0.0 (3.5)**	**100.0 ± 0.0 (3.5)**	97.7 ± 4.3 (7)	**100.0 ± 0.0 (3.5)**	**100.0 ± 0.0 (3.5)**
Blood transfusion	76.4 ± 0.6 (6)	77.0 ± 0.9 (5)	**78.7 ± 0.7 (1)**	77.3 ± 1.2 (4)	76.3 ± 0.1 (7)	77.4 ± 0.9 (3)	78.2 ± 0.7 (2)
Climate crashes	92.2 ± 1.2 (5)	92.1 ± 0.4 (6)	93.5 ± 1.2 (3.5)	93.5 ± 1.1 (3.5)	91.6 ± 0.4 (7)	94.3 ± 0.6 (2)	**94.4 ± 0.5 (1)**
Conn. sonar	80.0 ± 4.9 (6.5)	80.0 ± 4.8 (6.5)	85.5 ± 1.8 (2)	84.7 ± 2.6 (3)	82.0 ± 2.5 (5)	84.2 ± 2.2 (4)	**85.7 ± 2.1 (1)**
Optical recognition	25.1 ± 6.6 (7)	26.2 ± 3.4 (6)	**38.6 ± 0.2 (1)**	31.8 ± 3.1 (5)	35.3 ± 0.4 (3)	35.1 ± 0.5 (4)	38.2 ± 0.4 (2)
Drybeans	55.3 ± 4.8 (6)	60.5 ± 6.4 (4)	62.2 ± 1.4 (3)	**65.6 ± 0.4 (1)**	50.3 ± 10.8 (7)	57.3 ± 2.8 (5)	64.6 ± 0.7 (2)
Avila bible	50.5 ± 0.4 (7)	51.1 ± 0.4 (5)	50.8 ± 2.1 (6)	51.2 ± 0.3 (4)	53.1 ± 0.8 (3)	**53.8 ± 0.3 (1.5)**	**53.8 ± 0.2 (1.5)**
Wine quality red	56.6 ± 1.3 (3)	56.1 ± 2.6 (4)	52.3 ± 5.2 (6)	57.2 ± 1.5 (2)	51.7 ± 3.5 (7)	54.0 ± 1.1 (5)	**58.1 ± 1.1 (1)**
Wine quality white	51.9 ± 0.9 (3)	49.7 ± 2.3 (4)	47.2 ± 2.3 (7)	**52.5 ± 0.8 (1)**	47.6 ± 1.5 (6)	49.5 ± 1.1 (5)	52.3 ± 0.8 (2)
Average rank	5.1 ± 1.5	4.7 ± 1.6	3.3 ± 2.0	3.4 ± 1.4	5.9 ± 1.5	3.7 ± 1.5	1.9 ± 0.9
	Depth 3						
Dataset name	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	96.8 ± 0.9 (5)	97.3 ± 1.1 (4)	**97.9 ± 0.6 (1)**	97.6 ± 0.6 (3)	88.7 ± 12.3 (7)	96.3 ± 0.9 (6)	97.7 ± 0.5 (2)
Car evaluation	78.0 ± 0.8 (6)	**86.6 ± 0.5 (1)**	82.2 ± 0.6 (2)	80.1 ± 1.0 (5)	75.7 ± 5.6 (7)	81.7 ± 0.6 (3)	81.5 ± 0.7 (4)
Banknote auth.	93.5 ± 1.5 (5)	93.4 ± 1.6 (6)	**98.4 ± 0.4 (1)**	96.0 ± 0.9 (2)	92.4 ± 1.5 (7)	93.9 ± 1.9 (4)	94.6 ± 1.7 (3)
Balance scale	74.9 ± 2.8 (5)	67.5 ± 4.3 (7)	**79.9 ± 1.0 (1)**	78.0 ± 1.3 (4)	74.8 ± 3.4 (6)	78.6 ± 1.2 (3)	79.2 ± 1.1 (2)
Acute inflam. 1	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**
Acute inflam. 2	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**
Blood transfusion	78.1 ± 2.1 (4)	78.0 ± 1.5 (5)	**81.4 ± 1.2 (1)**	79.9 ± 1.4 (2)	76.3 ± 0.3 (7)	77.9 ± 1.1 (6)	79.3 ± 0.9 (3)
Climate crashes	93.8 ± 1.8 (6)	94.2 ± 1.5 (5)	95.3 ± 0.9 (4)	96.0 ± 0.9 (3)	92.0 ± 1.3 (7)	96.2 ± 0.6 (2)	**96.4 ± 0.6 (1)**
Conn. sonar	86.7 ± 6.2 (7)	89.1 ± 3.9 (5)	92.2 ± 1.8 (2)	**93.3 ± 2.2 (1)**	87.4 ± 3.6 (6)	90.0 ± 2.5 (4)	91.5 ± 2.1 (3)
Optical recognition	35.0 ± 8.1 (6)	29.1 ± 4.8 (7)	**60.9 ± 1.8 (1)**	51.0 ± 4.9 (3)	50.3 ± 2.4 (5)	50.6 ± 3.8 (4)	54.4 ± 1.6 (2)
Drybeans	77.0 ± 1.3 (3)	63.7 ± 4.6 (6)	77.8 ± 2.4 (2)	**78.6 ± 1.1 (1)**	61.3 ± 4.0 (7)	65.8 ± 3.3 (5)	74.8 ± 3.4 (4)
Avila bible	53.1 ± 0.4 (6)	51.5 ± 0.5 (7)	53.6 ± 1.9 (5)	54.1 ± 0.4 (4)	56.0 ± 0.7 (3)	**57.2 ± 0.7 (1.5)**	**57.2 ± 0.5 (1.5)**
Wine quality red	58.1 ± 2.0 (3)	57.6 ± 1.6 (4)	53.5 ± 5.1 (7)	**61.1 ± 1.6 (1)**	54.3 ± 1.6 (6)	56.4 ± 1.0 (5)	59.2 ± 1.3 (2)
Wine quality white	52.9 ± 1.0 (2)	50.1 ± 2.4 (5)	47.7 ± 2.8 (7)	**54.0 ± 0.9 (1)**	49.2 ± 1.2 (6)	51.0 ± 1.0 (4)	52.4 ± 0.8 (3)
Average rank	4.7 ± 1.4	5.0 ± 1.6	3.0 ± 2.2	2.7 ± 1.4	5.9 ± 1.4	4.0 ± 1.3	2.8 ± 1.0
	Depth 4						
Dataset name	CART	C4.5	GOSDT	TAO	CMA-ES	CRO-DT	CRO-DT (CS)
Breast cancer	97.2 ± 0.8 (5)	**99.5 ± 0.5 (1)**	98.5 ± 0.6 (2)	98.3 ± 0.6 (3)	94.2 ± 6.7 (7)	96.6 ± 0.8 (6)	98.0 ± 0.5 (4)
Car evaluation	84.9 ± 1.2 (6)	**88.8 ± 0.6 (1)**	87.4 ± 0.6 (2)	85.1 ± 1.1 (5)	82.3 ± 5.9 (7)	86.1 ± 1.2 (4)	86.7 ± 0.7 (3)
Banknote auth.	95.7 ± 2.0 (6)	99.5 ± 1.0 (2)	**99.6 ± 0.2 (1)**	98.3 ± 1.0 (3)	95.0 ± 1.3 (7)	97.0 ± 1.6 (5)	97.4 ± 1.6 (4)
Balance scale	81.0 ± 3.0 (5)	71.8 ± 4.1 (7)	**85.8 ± 0.8 (1)**	84.1 ± 1.3 (2)	78.4 ± 1.6 (6)	82.4 ± 1.6 (4)	83.9 ± 1.2 (3)
Acute inflam. 1	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**
Acute inflam. 2	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**	**100.0 ± 0.0 (4)**
Blood transfusion	79.7 ± 2.3 (3.5)	79.6 ± 1.9 (5)	**82.3 ± 1.2 (1)**	81.6 ± 1.4 (2)	76.3 ± 0.1 (7)	78.1 ± 1.0 (6)	79.7 ± 1.1 (3.5)
Climate crashes	95.0 ± 2.1 (6)	**99.0 ± 1.3 (1)**	96.0 ± 1.2 (5)	97.9 ± 0.9 (2)	93.3 ± 2.2 (7)	97.5 ± 0.7 (4)	97.7 ± 0.6 (3)
Conn. sonar	91.8 ± 6.3 (6)	**98.4 ± 2.1 (1)**	94.8 ± 1.7 (3)	97.5 ± 1.7 (2)	83.1 ± 2.2 (7)	93.3 ± 2.5 (5)	93.8 ± 2.5 (4)
Optical recognition	54.4 ± 14.0 (5)	31.7 ± 5.1 (7)	65.3 ± 5.6 (3)	**67.2 ± 6.6 (1)**	51.9 ± 2.9 (6)	61.1 ± 3.0 (4)	66.7 ± 1.5 (2)
Drybeans	80.9 ± 2.0 (2)	64.6 ± 3.6 (7)	65.3 ± 6.0 (6)	**84.0 ± 1.5 (1)**	67.3 ± 3.2 (5)	72.3 ± 4.7 (4)	78.5 ± 4.6 (3)
Avila bible	54.4 ± 1.3 (6)	52.2 ± 0.8 (7)	54.6 ± 1.5 (5)	56.7 ± 0.7 (4)	59.1 ± 0.9 (3)	**60.1 ± 0.8 (1.5)**	**60.1 ± 0.7 (1.5)**
Wine quality red	59.8 ± 2.8 (3)	59.0 ± 1.7 (4)	54.7 ± 5.4 (7)	**64.8 ± 1.3 (1)**	55.8 ± 1.1 (6)	58.6 ± 1.3 (5)	60.3 ± 1.6 (2)
Wine quality white	53.4 ± 1.1 (2)	50.7 ± 2.5 (5)	48.4 ± 3.5 (7)	**56.0 ± 0.7 (1)**	50.3 ± 0.8 (6)	52.3 ± 1.1 (4)	53.1 ± 0.8 (3)
Average rank	4.5 ± 1.5	4.0 ± 2.4	3.6 ± 2.1	2.5 ± 1.3	5.9 ± 1.4	4.3 ± 1.1	3.1 ± 0.8

References

[1] J.N. Morgan, J.A. Sonquist, Problems in the analysis of survey data, and a proposal, J. Am. Stat. Assoc. 58 (302) (1963) 415–434.

[2] V.G. Costa, C.E. Pedreira, Recent advances in decision trees: An updated survey, Artif. Intell. Rev. (2022) 1–36.

[3] D.V. Carvalho, E.M. Pereira, J.S. Cardoso, Machine learning interpretability: A survey on methods and metrics, Electronics 8 (8) (2019) 832.

[4] C. Molnar, Interpretable Machine Learning, second ed., 2022, URL https://christophm.github.io/interpretable-ml-book.

[5] R.C. Barros, M.P. Basgalupp, A.C. De Carvalho, A.A. Freitas, A survey of evolutionary algorithms for decision-tree induction, IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.) 42 (3) (2011) 291–312.

[6] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, CRC Press, 1984.

[7] J.R. Quinlan, C4.5: Programs for Machine Learning, Elsevier, 1993.

[8] M. Kretowski, Evolutionary Decision Trees in Large-Scale Data Mining, Springer, 2019, http://dx.doi.org/10.1007/978-3-030-21851-5.

[9] M. Czajkowski, K. Jurczuk, M. Kretowski, A parallel approach for evolutionary induced decision trees. MPI + OpenMP implementation, in: International Conference on Artificial Intelligence and Soft Computing, Springer, 2015, pp. 340–349.

[10] K. Jurczuk, M. Czajkowski, M. Kretowski, Fitness evaluation reuse for accelerating GPU-based evolutionary induction of decision trees, Int. J. High Perform. Comput. Appl. 35 (1) (2021) 20–32.

[11] K. Jurczuk, M. Czajkowski, M. Kretowski, GPU-based acceleration of evolutionary induction of model trees, Appl. Soft Comput. 119 (2022) 108503.

[12] D. Kalles, A. Papagelis, Lossless fitness inheritance in genetic algorithms for decision trees, Soft Comput. 14 (9) (2010) 973–993.

[13] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[14] M. Norouzi, M. Collins, M.A. Johnson, D.J. Fleet, P. Kohli, Efficient non-greedy optimization of decision trees, Adv. Neural Inf. Process. Syst. 28 (2015).

[15] R.A. Lopes, A. Freitas, R. Silva, F.G. Guimarães, Differential evolution and perceptron decision trees for classification tasks, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2012, pp. 550–557.

[16] S. Salcedo-Sanz, C. Camacho-Gómez, D. Molina, F. Herrera, A coral reefs optimization algorithm with substrate layers and local search for large scale global optimization, in: 2016 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2016, pp. 3574–3581.

[17] S. Salcedo-Sanz, A review on the coral reefs optimization algorithm: new development lines and current applications, Progress Artif. Intell. 6 (1) (2017) 1–15.

[18] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms–A survey, Swarm Evol. Comput. 44 (2019) 695–711.

[19] S. Jiménez-Fernández, C. Camacho-Gómez, R. Mallol-Poyato, J.C. Fernández, J. Del Ser, A. Portilla-Figueras, S. Salcedo-Sanz, Optimal microgrid topology design and siting of distributed generation sources using a multi-objective substrate layer coral reefs optimization algorithm, Sustainability 11 (1) (2019) 169.

[20] J. Pérez-Aracil, D. Casillas-Pérez, S. Jiménez-Fernández, L. Prieto-Godino, S. Salcedo-Sanz, A versatile multi-method ensemble for wind farm layout optimization, J. Wind Eng. Ind. Aerodyn. 225 (2022) 104991.

[21] I. Moya, E. Bermejo, M. Chica, Ó. Cordón, Coral reefs optimization algorithms for agent-based model calibration, Eng. Appl. Artif. Intell. 100 (2021) 104170.

[22] C.-S. Lin, M.-C. Chiang, C.-S. Yang, A co-evolution coral reefs optimization approach for multi-objective vehicle routing problem with time windows, in: 2019 IEEE International Conference on Systems, Man and Cybernetics, SMC, IEEE, 2019, pp. 2001–2006.

[23] S. Salcedo-Sanz, C. Camacho-Gómez, A. Magdaleno, E. Pereira, A. Lorenzana, Structures vibration control via tuned mass dampers using a co-evolution coral reefs optimization algorithm, J. Sound Vib. 393 (2017) 62–75.

[24] C. Camacho-Gómez, X. Wang, E. Pereira, I. Díaz, S. Salcedo-Sanz, Active vibration control design using the coral reefs optimization with substrate layer algorithm, Eng. Struct. 157 (2018) 14–26.

[25] J. Pérez-Aracil, C. Camacho-Gómez, A.M. Hernández-Díaz, E. Pereira, S. Salcedo-Sanz, Submerged arches optimal design with a multi-method ensemble meta-heuristic approach, IEEE Access 8 (2020) 215057–215072.

[26] R. Sánchez-Montero, C. Camacho-Gómez, P.-L. López-Espí, S. Salcedo-Sanz, Optimal design of a planar textile antenna for industrial scientific medical (ISM) 2.4 GHz wireless body area networks (WBAN) with the CRO-SL algorithm, Sensors 18 (7) (2018) 1982.

[27] C. Camacho-Gomez, R. Sanchez-Montero, D. Martínez-Villanueva, P.-L. López-Espí, S. Salcedo-Sanz, Design of a multi-band microstrip textile patch antenna for LTE and 5G services with the CRO-SL ensemble, Appl. Sci. 10 (3) (2020) 1168.

[28] A. Asuncion, D. Newman, UCI machine learning repository, 2007, URL http://archive.ics.uci.edu/ml.

[29] W.-Y. Loh, Fifty years of classification and regression trees, Internat. Statist. Rev. 82 (3) (2014) 329–348.

[30] J.R. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106.

[31] D. Bertsimas, J. Dunn, Optimal classification trees, Mach. Learn. 106 (7) (2017) 1039–1082.

[32] J. Lin, C. Zhong, D. Hu, C. Rudin, M. Seltzer, Generalized and scalable optimal sparse decision trees, in: International Conference on Machine Learning, PMLR, 2020, pp. 6150–6160.

[33] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, P.J. Stuckey, MurTree: Optimal classification trees via dynamic programming and search, 2020, arXiv preprint arXiv:2007.12652.

[34] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, second ed., Springer Publishing Company, Incorporated, 2015, http://dx.doi.org/10.1007/978-3-662-44874-8.

[35] R. Rivera-Lopez, J. Canul-Reich, A global search approach for inducing oblique decision trees using differential evolution, in: Canadian Conference on Artificial Intelligence, Springer, 2017, pp. 27–38.

[36] R. Rivera-Lopez, J. Canul-Reich, Construction of near-optimal axis-parallel decision trees using a differential-evolution-based approach, IEEE Access 6 (2018) 5548–5563.

[37] C. Veenhuis, M. Koppen, J. Kruger, B. Nickolay, Tree swarm optimization: an approach to PSO-based tree discovery, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 2, IEEE, 2005, pp. 1238–1245.

[38] J.E. Fieldsend, Optimizing decision trees using multi-objective particle swarm optimization, in: Swarm Intelligence for Multi-Objective Problems in Data Mining, Springer, 2009, pp. 93–114.

[39] F.E. Otero, A.A. Freitas, C.G. Johnson, Inducing decision trees with an ant colony optimization algorithm, Appl. Soft Comput. 12 (11) (2012) 3615–3626.

[40] L.L. Custode, F. Mento, F. Tursi, A. Smargiassi, R. Inchingolo, T. Perrone, L. Demi, G. Iacca, Multi-objective automatic analysis of lung ultrasound data from COVID-19 patients by means of deep learning and decision trees, Appl. Soft Comput. 133 (2023) 109926.

[41] S. Salcedo-Sanz, A. Pastor-Sánchez, L. Prieto, A. Blanco-Aguilera, R. García-Herrera, Feature selection in wind speed prediction systems based on a hybrid coral reefs optimization–extreme learning machine approach, Energy Convers. Manage. 87 (2014) 10–18.

[42] A. Martin, V.M. Vargas, P.A. Gutiérrez, D. Camacho, C. Hervás-Martínez, Optimising convolutional neural networks using a hybrid statistically-driven coral reef optimisation algorithm, Appl. Soft Comput. 90 (2020) 106144.

[43] J. Pérez-Aracil, C. Camacho-Gómez, E. Lorente-Ramos, C.M. Marina, L.M. Cornejo-Bueno, S. Salcedo-Sanz, New probabilistic, dynamic multi-method ensembles for optimization based on the CRO-SL, Mathematics 11 (7) (2023) 1666.

[44] S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, J. Portilla-Figueras, The coral reefs optimization algorithm: A novel metaheuristic for efficiently solving optimization problems, Sci. World J. 2014 (2014).

[45] J.H. Drake, A. Kheiri, E. Özcan, E.K. Burke, Recent advances in selection hyper-heuristics, European J. Oper. Res. 285 (2) (2020) 405–428.

[46] K.P. Bennett, O.L. Mangasarian, Multicategory discrimination via linear programming, Optim. Software 3 (1–3) (1994) 27–39.

[47] K.V. Price, Differential evolution, in: Handbook of Optimization, Springer, 2013, pp. 187–214.

[48] R. Rivera-Lopez, J. Canul-Reich, J.A. Gámez, J.M. Puerta, OC1-DE: A differential evolution based approach for inducing oblique decision trees, in: International Conference on Artificial Intelligence and Soft Computing, Springer, 2017, pp. 427–438.

[49] P.P. M Kretowski, Global induction of decision trees: From parallel implementation to distributed evolution, in: International Conference on Artificial Intelligence and Soft Computing, Springer, 2008, pp. 426–437.

[50] B. Chapman, G. Jost, R.v.d. Pas, Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation), The MIT Press, 2007.

[51] P.S. Pacheco, Parallel Programming with MPI, Morgan Kaufmann Publishers Inc., 1996.

[52] S. Salcedo-Sanz, D. Gallo-Marazuela, A. Pastor-Sánchez, L. Carro-Calvo, A. Portilla-Figueras, L. Prieto, Offshore wind farm design with the coral reefs optimization algorithm, Renew. Energy 63 (2014) 109–115.

[53] S. Salcedo-Sanz, J. Muñoz-Bulnes, M.J. Vermeij, New coral reefs-based approaches for the model type selection problem: A novel method to predict a nation's future energy demand, Int. J. Bio-inspired Comput. 10 (3) (2017) 145–158.

[54] P. Jiménez, J.C. Roldán, R. Corchuelo, A coral-reef approach to extract information from HTML tables, Appl. Soft Comput. 115 (2022) 107980.

[55] S.K. Murthy, S. Kasif, S. Salzberg, R. Beigel, OC1: A randomized algorithm for building oblique decision trees, in: Proceedings of AAAI, vol. 93, Citeseer, 1993, pp. 322–327.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[57] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, URL https://www.tensorflow.org/. Software available from tensorflow.org.

[58] H. McTavish, C. Zhong, R. Achermann, I. Karimalis, J. Chen, C. Rudin, M. Seltzer, Fast sparse decision tree optimization via reference ensembles, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 9604–9613.

[59] M.A. Carreira-Perpinán, P. Tavallali, Alternating optimization of decision trees, with application to learning sparse oblique trees, Adv. Neural Inf. Process. Syst. 31 (2018).

[60] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195.

[61] C. Singh, K. Nasseri, Y.S. Tan, T. Tang, B. Yu, Imodels: A Python package for fitting interpretable models, J. Open Source Software 6 (61) (2021) 3192, http://dx.doi.org/10.21105/joss.03192.

[62] L.A. Breslow, D.W. Aha, Simplifying decision trees: A survey, Knowl. Eng. Rev. 12 (01) (1997) 1–40.

[63] E. Carrizosa, C. Molero-Río, D. Romero Morales, Mathematical optimization in classification and regression trees, Top 29 (1) (2021) 5–33.

[64] X. Hu, C. Rudin, M. Seltzer, Optimal sparse decision trees, Adv. Neural Inf. Process. Syst. 32 (2019).

[65] N. Hansen, Y. Akimoto, P. Baudis, CMA-ES/pycma on Github, 2019, http://dx.doi.org/10.5281/zenodo.2559634, Zenodo.

[66] Z. Fu, B.L. Golden, S. Lele, S. Raghavan, E. Wasil, Diversification for better classification trees, Comput. Oper. Res. 33 (11) (2006) 3185–3202.

[67] M. Krętowski, M. Grześ, Evolutionary induction of cost-sensitive decision trees, in: International Symposium on Methodologies for Intelligent Systems, Springer, 2006, pp. 121–126.

[68] D. Haizhou, M. Chong, Study on constructing generalized decision tree by using DNA coding genetic algorithm, in: 2009 International Conference on Web Information Systems and Mining, IEEE, 2009, pp. 163–167.

[69] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: An update, ACM SIGKDD Explor. Newslett. 11 (1) (2009) 10–18.

Chapter 4

Evolving Decision Trees for Interpretable Reinforcement Learning

In the previous chapter, we explored the evolutionary approach to create DTs and proposed a novel matrix-based encoding that greatly sped up the evolution of classification trees, as well as a novel evolutionary DT algorithm (called CRO-DT) that is competitive with other modern DT approaches on classification tasks. In this chapter, we will build upon these approaches to propose an algorithm that evolves interpretable and highly-performing trees for RL problems.

The most immediate way to adapt the proposed methods to a RL context is to run CRO-DT as-is, but making two basic changes: one to the fitness function (by replacing the supervised learning *accuracy* with the reinforcement learning *average reward*), and one to the leaves (by replacing the supervised learning *labels* with the reinforcement learning *actions*, as shown in Figure 1.3). This results in an algorithm that is very similar to CRO-DT, but that can be run on RL tasks. However, although this basic approach works (as in, it produces trees that evaluate the current state of the RL environment and take actions accordingly), preliminary results showed that this version of the algorithm was unable to produce solutions of reasonable quality, which led three other core changes to be introduced.

First, a more complex initialization procedure was included. The best results obtained in Chapter 3 were from a CRO-DT variant that did not start from scratch, but that was in fact seeded with a small population of CART trees; this allowed the evolutionary algorithm to focus purely on fine-tuning, which in turn led to it being competitive and achieving the best overall solutions. In an attempt to reproduce this idea, we sought ways to jumpstart CRO-DT with decent initial solutions for RL, which led us to employing procedures from the RL subfield of Imitation Learning

(thoroughly explained in Appendix A). As will be shown, this was instrumental in solving more complex tasks.

Second, a more sophisticated fitness function was introduced. Employing average reward, as previously stated, is sufficient to make the algorithm work for RL tasks, since it will guide the algorithm towards solutions that have higher average reward; however, this function is also overly simplistic, and does not consider other dimensions that are also important to obtain the desired outcomes (such as consistency and interpretability). To tackle those issues, we modified the fitness function to include two other components, and we demonstrated through ablation studies that removing either of those components results in worse solutions according to the established criteria.

Finally, and perhaps most crucially, the algorithm's encoding was changed. While CRO-DT's novel matrix-based encoding is much faster for classification tasks, preliminary experiments demonstrated that it does not hold the same benefits when RL environments are considered. This is due to full parallelization becoming impossible: while in supervised learning every prediction is independent and therefore the proposed procedure is able to evaluate every tree in the population at once, in RL the predictions are *not* independent, since these tasks are divided into timesteps that always depend on their previous installments. In other words, a single matrix operation is sufficient to evaluate an entire classification dataset, but not to run an entire RL episode, since you only know the next state after predicting the previous one. Parallelizing everything else is possible, but is not enough to attain any significant speedups. For these reasons, and in order to make the algorithm more general and easy to grasp, we employed a more traditional tree encoding with simpler genetic operators instead of the matrix encoding proposed in the previous Chapter; every other aspect of the evolutionary algorithm was preserved.

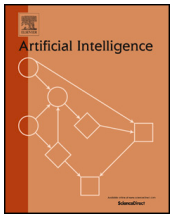
The resulting algorithm, called MENS-DT-RL, is therefore a modified form of CRO-DT that incorporates several adaptations specific to RL. It is a robust algorithm that can produce trees with both high performance and interpretability, and that achieves novel results in standard RL benchmarks. The remainder of this chapter is dedicated to our published paper which discusses it in detail.

Artificial Intelligence 327 (2024) 104057

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Evolving interpretable decision trees for reinforcement learning

Vinícius G. Costa [a,1], Jorge Pérez-Aracil [b], Sancho Salcedo-Sanz [b], Carlos E. Pedreira [a,*]

[a] Systems Engineering and Computer Science Department, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[b] Department of Signal Processing and Communications, Universidad de Alcalá, Alcalá de Henares, 28805, Madrid, Spain

ARTICLE INFO

Keywords:
Interpretability
Reinforcement learning
Decision trees
Evolutionary-based algorithms
Multi-method ensembles

ABSTRACT

In recent years, reinforcement learning (RL) techniques have achieved great success in many different applications. However, their heavy reliance on complex deep neural networks makes most RL models uninterpretable, limiting their application in domains where trust and security are important. To address this challenge, we propose MENS-DT-RL, an algorithm capable of constructing interpretable models for RL via the evolution of decision tree (DT) models. MENS-DT-RL uses a multi-method ensemble algorithm to evolve univariate DTs, guiding the process with a fitness metric that prioritizes interpretability and consistent high performance. Three different initializations for the MENS-DT-RL are proposed, including the use of Imitation Learning (IL) techniques, and a novel pruning approach that reduces solution size without compromising performance. To evaluate the proposed approach, we compare it with other models from the literature on three benchmark tasks from the OpenAI Gym library, as well as on a fertilization problem inspired by real-world crop management. To the best of our knowledge, the proposed scheme is the first to solve the Lunar Lander benchmark with both interpretability and a high confidence rate (90% of episodes are successful), as well as the first to solve the Mountain Car environment with a tree of only 7 nodes. On the real-world task, the proposed MENS-DT-RL is able to produce solutions with the same quality as deep RL policies, with the added bonus of interpretability. We also analyze the best solutions found by the algorithm and show that they are not only interpretable but also diverse in their behavior, empowering the end user with the choice of which model to apply. Overall, the findings show that the proposed approach is capable of producing high-quality transparent models for RL, achieving interpretability without losing performance.

1. Introduction

In recent years, deep Reinforcement Learning (RL) algorithms have been applied with great success to various tasks, such as robotics [1], resource management [2], recommendation systems [3], and playing Go at superhuman levels [4]. While these results are impressive, deep RL algorithms typically use Deep Neural Networks (DNNs) with complex multi-layer structures and thousands (if not millions) of parameters. For this reason, the resulting models are generally considered to be uninterpretable, which is in line with the popular understanding of DNNs as "black boxes" [5–7].

* Corresponding author.
E-mail address: pedreira@cos.ufrj.br (C.E. Pedreira).

[1] This work was partially done while VGC was in a stay at the Universidad de Alcalá.

https://doi.org/10.1016/j.artint.2023.104057
Received 7 August 2023; Received in revised form 9 December 2023; Accepted 10 December 2023

Available online 16 December 2023
0004-3702/© 2023 Elsevier B.V. All rights reserved.

The uninterpretability of deep RL models has several drawbacks. The first of those is related to the current great concern about potentially harmful effects that can be caused by artificial intelligent systems: since the "model's decision-making process" is incomprehensible to humans, it is hard or perhaps impossible to guarantee that it will not execute undesired and potentially harmful actions [8]. Second, uninterpretability complicates troubleshooting, since it is hard to understand why the model chose one action over another [8,9]. Third, there are legal concerns, since transparency and accountability are increasingly being legislated as prerequisites for the deployment of autonomous solutions in real-world environments [10]. Finally, for high-stakes domains where safety is critical (such as healthcare [11,12] and autonomous driving [13]), the lack of transparency of DNN-based approaches may lead to solutions that are impossible to trust and therefore to deploy. These drawbacks have recently led to a great deal of attention being paid to interpretable RL: several recent works and surveys have been dedicated to it [9,10,14,15], and the task has been identified as one of the major challenges for interpretable ML as a whole [8].

An intuitive way to incorporate interpretability into RL is to replace the DNNs with more interpretable models, such as the widely popular decision trees (DTs): rule-based models, usually presented in a flowchart-like structure that consists of logical tests and predictions [16]. Because their inner workings are completely transparent and simulatable by the end user, DTs are often considered the go-to techniques in interpretable ML, sometimes referred to as "white boxes" [17–19]. But while mixing DTs with RL may seem intuitive, there is no obvious way to perform this integration, mainly because traditional DT algorithms require that the entire dataset be available at once, while RL algorithms typically work on an online, sample-by-sample basis. The literature has attempted to reconcile this conflict by introducing several different approaches, ranging from the gradient-based [20] to the greedy [21]. However, most works end up sacrificing either interpretability or model performance, resulting in trees that cannot directly compete with more complex and uninterpretable approaches.

To address this challenge, we propose the "Multi-method ENSemble for Decision Trees in Reinforcement Learning" (MENS-DT-RL), an evolutionary-based algorithm capable of producing interpretable and high-performing DTs for RL tasks. The algorithm works by adapting a multi-method ensemble algorithm (CRO-SL [22,23]) that has been successful at optimization problems both in machine learning [24,25] and engineering [26,27]. We extend the considered multi-method ensemble with a set of genetic operators capable of handling DT models, as well as with a fitness metric that rewards trees with consistent behaviors and smaller sizes. Although the basic MENS-DT-RL presents high-quality results for simpler tasks, its true value appears when it is initialized with good solutions from an Imitation Learning (IL) approach, especially when these solutions are pruned using a novel technique that adapts traditional post-pruning algorithms to the RL environment. The main contributions of this work can be summarized as follows:

- We propose a novel evolutionary algorithm for interpretable RL, based on a multi-method ensemble optimization algorithm.
- We develop a method capable of pruning DTs in RL environments, inspired by traditional DT pruning methods from supervised learning.
- We evaluate the proposed algorithm over three popular benchmarks from the OpenAI Gym environment [28] and a crop management task based on real-world fertilization problems [29]. To the best of our knowledge, this algorithm surpasses the state-of-the-art with regards to interpretable DTs for RL, finding the new smallest DT capable of solving the Mountain Car environment with a perfect success rate, being the first to solve the Lunar Lander task with a high success rate ($\geq$90% of successful trials), and surpassing all previous expert policies for the real-world fertilization task.
- We demonstrate not only that the resulting solutions are interpretable, but also that they are diverse, and therefore empower the user with the ability to choose between distinct behaviors for the desired task.

The rest of the paper is organized as follows: Section 2 discusses the necessary knowledge and offers a review of the related works in the literature. Section 3 describes the MENS-DT-RL algorithm and its initialization procedures. Section 4 contains the experiments and discussion, and Section 5 presents our concluding remarks.

2. Background and related works

With the goal of making the paper self-contained, in this section we briefly cover the topics of Reinforcement Learning and Imitation Learning, as well as briefly introduce the multi-method ensemble algorithm that serves as basis for the proposed MENS-DT-RL technique. At the end of the section, we offer a review of the related works.

2.1. Reinforcement learning

Reinforcement Learning (RL) [30] is an area of machine learning that aims to create intelligent agents that observe the current state of the world, process it, and decide what is the best possible action to take according to a previously defined goal. Traditionally, there are three key components to an RL problem: *states*, which contain the environmental data that the agent can observe at a given moment; *actions*, which encapsulate the ways in which the agent can affect the environment; and *rewards*, which are the feedback that the agent receives after taking a certain action in a certain state. In an ideal learning process, the agent starts by taking random actions and receiving poor rewards, but eventually learns how to use the current state to figure out the best action to take, thus learning a "behavior function" that maximizes the prospective reward. Such functions are called *policies*, and are usually denoted by $\pi : S \rightarrow A$.

Using the immediate reward is typically not enough to learn good policies: often the consequences of an action are felt only several timesteps in the future, which complicates the issue of learning which actions were well taken and which were not. In the

most extreme cases, a reward is given only after a terminal state is reached (i.e., at the end of so-called *episodes*, finite sequences that begin in an initial state and end in a terminal one). The traditional example is a chess game, in which the agent collects a reward of $+1$ if it wins, -1 if it loses, and 0 if it draws; until an outcome is reached, the reward is continuously 0, which severely limits the feedback the agent receives throughout the learning process.

These and other problems can be addressed to some extent by techniques such as the traditional Q-learning algorithm [31]. In this algorithm, the agent maintains a Q-table, which is a table where each entry contains a Q-*value* (i.e. the expected cumulative reward of taking a given action in a given state). Through iterative updates based on observed rewards and transitions between states, Q-learning modifies these Q-values to approximate the optimal policy, continuing until either the Q-table converges or the agent reaches a desired level of performance. Although Q-learning is a cornerstone of RL literature and spawned several extensions [32–34], there are other fronts to the field that do not build directly upon it, such as Imitation Learning.

2.1.1. Imitation learning

Imitation Learning (IL) [35] is a sub-field of RL in which the goal is not to train an agent from scratch, but instead to imitate another agent called an *expert*. These techniques find great use in tasks where it is easier to demonstrate the correct behavior than to implicitly define it through a reward function, such as in autonomous driving [36], human-like locomotion [37] and object manipulation [38], among others. The expert is usually taken to be a human demonstrator, but most IL techniques can also be applied when the target of imitation is another RL agent.

The most basic IL approach, Behavioral Cloning, simply collects a dataset of demonstrations $\mathcal{D} = \{(s, a)\}$ and fits a supervised learning algorithm to the data [39]. The result is a model capable of predicting an action given an input state, and that therefore can be used in much the same way as an RL agent. Although intuitive, this naive approach has been shown to be susceptible to "covariate shift": a condition in which the imperfect imitator, due to its imperfection, sometimes takes non-expert actions and ends up in states that the expert would not visit. Since the expert would not visit these states, they are not included in the dataset $\mathcal{D}$ used to train the imitator, which leads it to take even more non-expert actions and end up in even more unfamiliar states. Because this effect often leads to bad results, more sophisticated algorithms were proposed, such as DAgger [40].

In DAgger, covariate shift is tackled by extending the dataset $\mathcal{D}$ with these distinct situations brought by the imitator's imperfection. At every i-th iteration of DAgger, a new imitator π_i is produced by fitting a supervised learning model on the dataset $\mathcal{D}$, just like in Behavioral Cloning. Then, the dataset is expanded: first, the imitator π_i freely interacts with the environment and visits a set of states, which are stored by the algorithm. Next, this set of states is passed on to the expert agent, who labels them with the expert action it would take in each state. The result is a dataset $\mathcal{D}_i$ that contains imitator states but expert actions, which is then aggregated unto $\mathcal{D}$ for the next $(i+1)$-th iteration. Through this iterative approach, DAgger finds much more success than the simpler Behavioral Cloning, since covariate shift is addressed by having $\mathcal{D}$ contain a wider array of states. The key disadvantage of DAgger, however, is that it requires continuous access to an expert – which may be costly or impossible, depending on the domain.

Since IL effectively reduces RL problems into a supervised learning framework, DTs for RL can be easily trained by using off-the-shelf traditional DT algorithms as imitators. However, there is no guarantee that these trees will be interpretable, since in trying to faithfully imitate the experts they can end up being unreasonably large. Therefore, more specialized solutions are necessary, such as the ones proposed in this paper.

2.2. Multi-method ensemble algorithm for optimization

In optimization problems, an *ensemble* method for optimization refers to an algorithm that combines different types of alternative methods, search strategies or operators, to obtain high-quality solutions [41]. The application of ensemble approaches to solve optimization problems has been very important in the last few years, due to the good results obtained by these combinations of techniques in hard optimization problems and real applications. Following [41], there are different types of ensemble approaches, but the most used are those trying to obtain an optimal combination of different types of search strategies or operators within a single algorithm (multi-method ensembles). The main idea behind these ensemble approaches is to exploit the capacity of different search methods/operators by combining them in several possible ways, in order to improve the search ability of the final approach in optimization problems.

Different types of multi-method ensembles for optimization have been described in the literature. An example of multi-method competitive ensemble is [42], in which different operators are applied in a single evolutionary algorithm-based ensemble. An approach with a similar idea was proposed in [43]. Multi-method approaches have also been applied to improve the performance of meta-heuristics in multi-objective optimization problems [44]. There are also multi-method algorithms which work on different sub-populations such as [45], and ensembles of multi-strategy algorithms based on algorithms such as Differential Evolution [46,47].

In this paper we consider the CRO-SL multi-method ensemble [22], an algorithm that has obtained excellent results in hard optimization problems in the past [48,25,49,50]. Specifically, we apply the probabilistic-dynamic version of the algorithm [51]. Details on the implementation of the multi-method ensemble, as well as a Python implementation of the algorithm, can be found in [51].

2.3. Related works

In this section, we review those works that similarly tackle the issue of interpretability in RL. Given the vastness of the field, a comprehensive review is outside the scope of the current paper; instead, we focus on contributions that also tackle interpretable RL

through the DT angle. However, given that DTs are not the only representation that has been explored in this literature, we start out by taking a non-exhaustive approach to position DTs within the larger context of policy structures that have been previously explored in interpretable RL.

Formulas are a common representation in interpretable RL. In these models, each action is expressed as a function of the attributes in the state s; for example, in an air conditioner task, the temperature of the unit could be represented as $f(x) = 20.175 + 0.05temp - 0.0045temp^2$, where *temp* is the current temperature in the room. There is no well-defined way to obtain formulas of this kind: for instance, [52] used genetic programming techniques to generate formulas from a historical dataset of environmental state-action trajectories, while [53] used a novel method called evolutionary feature synthesis to create complex features from state attributes, which were then assembled into functions through linear models. In general, these formula-based solutions are not readily comparable to DT agents: although formulas can be clear, concise, and particularly insightful when the underlying model lends itself well to mathematical modeling, DTs are easier to visualize and interpret without number crunching, a property that may be more desirable to the end user. One model is not more interpretable than the other – each has its own flavor of interpretability.

Another structure used in interpretable RL is that of programmatic policies. These models, which can be seen as a generalization of the formulas described above, use a mix of Boolean, algebraic, and conditional operators to create a system that closely resembles human-made computer programs. In the previous air conditioning task, an example agent of this type would be "***if*** *(temperature* $>$ *30)* ***and*** *(humidity* $>$ *20%)* ***then*** *set air conditioner to (41.66* $-$ *0.66* $\times$ *temperature)* ***else*** *turn it off*". As with formulas, a wide variety of techniques have been used in the past to obtain these models: [54] built them by searching the latent space generated by variational autoencoders, [55] did the same by combining Bayesian optimization with imitation learning, and [56] derived a Bayesian inference algorithm to optimize agents from few-shot datasets. Again, comparing these types of solutions to DTs is difficult: programmatic policies are more flexible, can handle more complex logic, and are closer to human computer programs, which tend to be interpretable. However, due to their flexibility, these machine-generated programs can also quickly become unclear and difficult to understand, a problem that DTs do not face with the same intensity due to their more constrained and visual structure.

Fuzzy control systems, which consist of a series of fuzzy IF-THEN rules, are a third type of representation used in several works. An example for an air conditioner control problem would be "***IF*** *temperature is hot* ***THEN*** *turn A/C to cold*", where "hot" and "cold" are defined by fuzzy membership functions and can take a predefined range of values (in contrast to a "crisp" logical test like "IF temperature > 35 THEN turn A/C on", which would do nothing if the temperature was 34.9 degrees). Although fuzzy control systems were originally proposed in the '70s [57], they were usually defined by hand rather than learned from data, leaving a gap which researchers tried to fill with a wide range of techniques. [58] used genetic algorithms to evolve individuals as rules, combining them to produce a complete solution in a framework they called "symbiotic." [59] mixed traditional RL algorithms with modern fuzzy techniques to solve the problem in an online manner, avoiding the need to relearn as the environment changes. [60] used the Particle Swarm Optimization (PSO) metaheuristic to generate these rules from a batch dataset, similar to their posterior work on evolving formulas [52]. Compared to DT models, fuzzy rule sets are often more powerful, have conditions that may be easier to understand (due to the use of concepts as thresholds), and have fewer rules than a DT would have if converted to the same rule representation. However, DTs have a clearer hierarchical structure, can be represented visually, and do not require external analysis of membership function graphs in order to be mentally simulated, which makes them more straightforward to interpret. As with the other models discussed previously, the choice ultimately comes down to personal taste, as each representation has its own approach to interpretability.

Next, we move on to a more in-depth discussion of works that tackle interpretable RL through the use of DTs. In this line of research, four different approaches can be discerned: greedy, imitation-based, gradient-based and evolutionary learning.

The *greedy* approach attempts to build the DTs iteratively from scratch, similarly to traditional greedy DT algorithms like CART [61] and C4.5 [62]. The first contribution in this line of work was proposed by Pyeatt and Howe [63]: DTs were used to predict the Q-value of every (state, action) observation so that each leaf contains a history of the ΔQ updates applied to it. If this sample history reaches a sufficiently high variance, the leaf is split into two, since it supposedly contains more than one relevant region of the input space. The original paper does not report tree sizes, but given that the procedure is strictly additive (the tree only increases in size), it seems safe to assume that the produced trees are not particularly small, which can severely impact the interpretability of the final model. A related approach can be found in U-Trees [64] and its extension, Continuous U-Trees [65]. Both these algorithms also store online information in the leaves and split them when a condition is fulfilled, but in their case, the leaf-action mapping is defined by solving the DT's corresponding Markov Decision Process, while the splitting criterion is either a Kolmogorov-Smirnov test or a check for Markov property violation. These two algorithms suffer from the same issue as the proposal by Pyeatt and Howe, in the sense that they are strictly additive and tree size is not reported. This issue also impacts Fitted Q-Iteration (FQI; [66]), a more popular approach to mixing DTs with RL. At each iteration, a new regression tree is trained on a dataset composed of (state, action) pairs as inputs and Q-values as outputs, which in theory gradually approximates the optimal policy. While FQI can produce DT's for RL, the final model cannot be easily interpreted – not only because of the additive issue mentioned above, but also because each leaf represents a single Q-value (instead of an action). This means that in order to choose an action with an FQI tree, it is necessary to simulate the tree for every possible action and remember which action had the highest Q-value (assuming actions are taken greedily). Naturally, this process is much less interpretable than simply having actions for leaves and simulating the tree once, which is the approach taken in this paper.

Another branch of contributions involves *imitation-based* approaches, which borrow ideas from the IL literature to train DTs based on uninterpretable models. In [67], authors proposed an algorithm called VIPER, which extends the widely-used IL algorithm DAgger [40] and differentiates itself by learning not the actions but instead the Q-values themselves. The authors show that by

representing the policy as a DT structure, it is possible to verify logically that certain states are never reached, which is key for achieving trustworthy models. However, since the authors were not focused on interpretability, it is not clear if VIPER is capable of producing small and interpretable trees (indeed, the original paper reports trees with close to 700 nodes). Another approach in the same line is [68], which does not employ a traditional IL algorithm, but instead builds upon a method proposed by [69] to distil a DNN into the structure of a DT. Although the final models obtained high average reward in a challenging task, the authors employed a variant of DTs called "soft trees", which replaces the traditional univariate logical tests in the inner nodes with a full linear combination of parameters that are then passed onto a non-linear function (akin to a DNN's neuron). For this reason, it is debatable if the proposed solution achieves traditional notions of interpretability.

The *gradient-based* approach trains DTs for RL by replacing the traditional greedy approach with online, gradient-based algorithms. This can be seen as part of a broader movement in DT research to keep the interpretability of a DT while reaping the benefits of DNNs' flexible training algorithms, creating hybrid models that have been called gradient-based trees [16] or neural trees [70]. Following this idea, [71] proposed Linear Model U-Trees (LMUT), an extension of the greedy Continuous U-Trees that employs linear models at the leaves and trains them via stochastic gradient descent. Although the model obtains some degree of success, the produced trees are not interpretable in a traditional sense, since each leaf contains not a single comprehensible action but an entire linear model itself. A final gradient-based method was proposed by [20]. The authors built upon an earlier algorithm by [72], which used stochastic gradient descent to train soft trees; by adapting this algorithm, training DTs for RL becomes straightforward. However, as was pointed previously when discussing [68,69], this "soft tree" approach arguably lacks interpretability, and although the authors provided a solution to make the model more interpretable by converting the soft splits into univariate ones, this process resulted in some loss of performance. Thus, better trees may be found by aiming directly for univariate solutions from the start.

Finally, *evolutionary* approaches employ evolutionary algorithms (EAs) to train DTs for RL tasks. In [73], the authors use a Grammatical Evolution approach to evolve the tree's inner splits, while the leaves are determined by running Q-learning for a low number of episodes. Although this hybrid procedure reduces the number of parameters that need to be evolved by the EA (which usually has the positive effect of accelerating the algorithm), it can also bias the results towards trees that are larger than necessary, since it has been shown in the literature that the tree size needed to obtain an optimal policy is sometimes larger than the tree size needed to represent it [64]. Furthermore, the proposed approach is not able to obtain small univariate trees for more sophisticated environments such as Lunar Lander, which the algorithms proposed here are able to. Finally, [74] uses EAs to evolve a structure known as non-linear DTs, in which the inner splits are non-linear inequalities such as $w_1 x_1^{-3} + w_2 x_2^3 - w_3 \leq 0$. This augments the representation capacity of DTs and therefore reduces their size, but it also limits the traditional notions of interpretability for which DTs are known.

3. Multi-method ensemble for decision trees in reinforcement learning (MENS-DT-RL)

In this section we describe the MENS-DT-RL algorithm, which is applied to evolve interpretable DTs for RL tasks. In essence, the MENS-DT-RL closely follows the base structure of the CRO-SL algorithm described in [51], such that each individual in the ensemble corresponds to a different tree agent, and each mutation operator applies a distinct modification to the tree structure (e.g. removing a node, adding a node, etc). Then, when an individual fitness must be calculated, the algorithm takes the corresponding tree and runs it over a set of N episodes, using the total reward of each episode to calculate the individual's fitness through a certain function. The overall approach is illustrated in Fig. 1, and the fitness evaluation is illustrated in Fig. 2. In what follows, we provide a detailed look at the components of the MENS-DT-RL.

The first component is the *solution representation*. In the original multi-method ensemble in [51], solutions are represented as a vector or matrix of real numbers, which are then modified in order to produce a good solution. This allows for the usage of numerical optimization algorithms as operators, such as Differential Evolution and Harmony Search (both of which are already implemented in the open-source implementation from [51]). However, although matrix representations of DTs have been proposed in the past [75–77], in this work we employ a more straightforward and intuitive representation of each solution as a traditional univariate DT. Since numerical operators are not applicable under this representation, new ones were proposed for MENS-DT-RL, allowing an input tree to be transformed into any other possible univariate tree. The operators are illustrated in Fig. 3, and can be described as follows:

1. *Replace with child*: select a random inner node and uniformly select either its left or right subtree. Replace the selected inner node with its selected subtree.
2. *Truncate*: select a random inner node and uniformly select either its left or right subtree. Replace the selected subtree with a randomly generated leaf.
3. *Insert inner node*: select a random inner node from the tree, and between it and its parent, insert a new random inner node. Assign the selected inner node as the left or right child of the newly generated inner node with 50-50 probability.
4. *Expand leaf*: select a random leaf and turn it into an inner node with two random leaves for children.
5. *Reset split*: select a random inner node. Replace its attribute with a randomly selected one, and replace its threshold with a uniform value between -1 and 1.
6. *Modify threshold*: select a random inner node and apply to its threshold a Gaussian perturbation of mean 0 and standard deviation of 0.1.
7. *Modify leaf (discrete)*: select a random leaf and replace its action with a new randomly selected one.
8. *Modify leaf (continuous)*: select a random leaf and replace its action with a random value in the range of possible continuous actions.

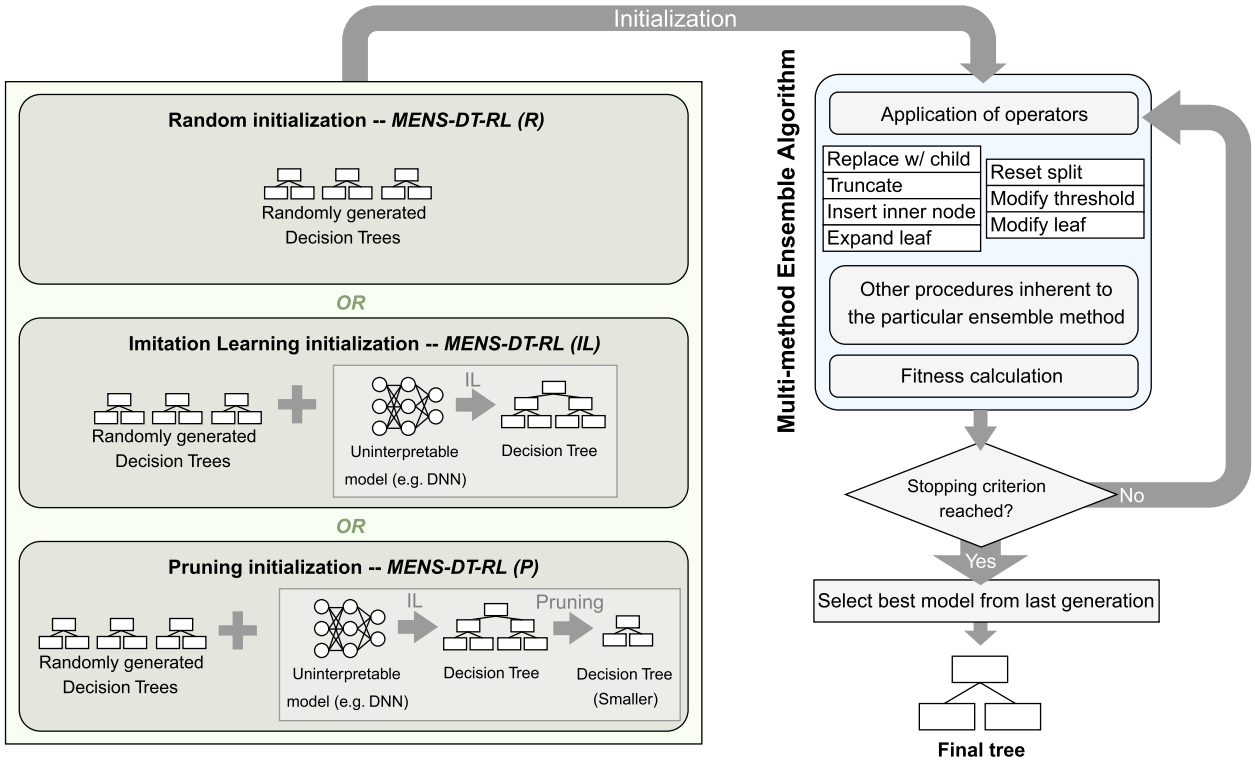


Fig. 1. An illustration of the overall MENS-DT-RL algorithm. First, an initialization procedure is selected (either random trees, random trees and trees obtained via Imitation Learning, or random trees and trees obtained via Imitation Learning that were pruned using Reward Pruning). Then, the initial population is passed onto a multi-method ensemble algorithm in order to produce an improved solution, which is ideally a small and high-performing tree agent. In this paper, the particular multi-method ensemble used is the Probabilistic CRO-SL [51].

The second key component in the MENS-DT-RL algorithm is the *fitness*. As is usual in evolutionary computation, each individual is evaluated in accordance with a fitness metric that corresponds to the quality of each solution. Since we are dealing with RL problems, the most natural metric for performance would be the average cumulative reward, since it is the one most often used to compare different algorithms. However, using only the average reward might bias the algorithm towards models that are needlessly large: for instance, there might be a situation where a tree is half the size of its predecessor, but has only a slightly lower average reward – an algorithm that focuses only on reward would ignore this more interpretable solution. For this reason, it might be interesting to include tree size in the fitness function, in addition to the average reward. It must be said, however, that a lower tree size does not always imply a more interpretable tree: for instance, end-users might find a larger tree more interpretable than a smaller one if the splits of the larger one are closer to how the user thinks about the problem [78]; in addition, trees with multivariate functions in their splits are usually smaller than their univariate counterparts, but not exactly more interpretable [16]. Indeed, it is impossible to express interpretability with a single metric, given the subjectivity of the topic. However, since a tree with 5 nodes is almost guaranteed to be more interpretable than a tree with 100 nodes that tackles the same task, we opt to use the number of nodes as a proxy for interpretability, even though it is an imperfect one.

Yet, a myopic focus on reward does not only leave out interpretability: it may also bias the algorithm toward models that are inconsistent. A tree with an average reward of 99 and a standard deviation of 20 will be considered inferior to a tree with an average reward of 100 but a standard deviation of 50, while in practice they should be considered at least on an equal level. Given these considerations, the MENS-DT-RL algorithm attempts to build more consistent and interpretable models by using the following fitness metric:

$$fitness(M) = \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)} - \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(G_0^{(M,j)} - \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)}\right)^2} - \alpha||M|| \tag{1}$$

Where M is a DT, $||M||$ is the number of nodes in the tree, N is the number of episodes runs to evaluate fitness, $G_0^{(M,i)}$ is the undiscounted sum of all rewards received in the i-th episode of the tree M (also called "total reward" or simply "reward"), and α is a regularization parameter that penalizes larger tree sizes. From another perspective, this measure is effectively the average reward minus the standard deviation of rewards minus the tree size weighted by α. Including these two components leads to the acceptance of more consistent models that are also more aware of the tradeoffs between performance and size. An ablation study investigating the effects of removing these two components is presented in Appendix A.

A third key aspect of the MENS-DT-RL algorithm is *attribute normalization*. At each step in the RL environment, the attributes for whose minimum and maximum values are known are normalized to a [-1, 1] range, otherwise, they remain unaltered. This step

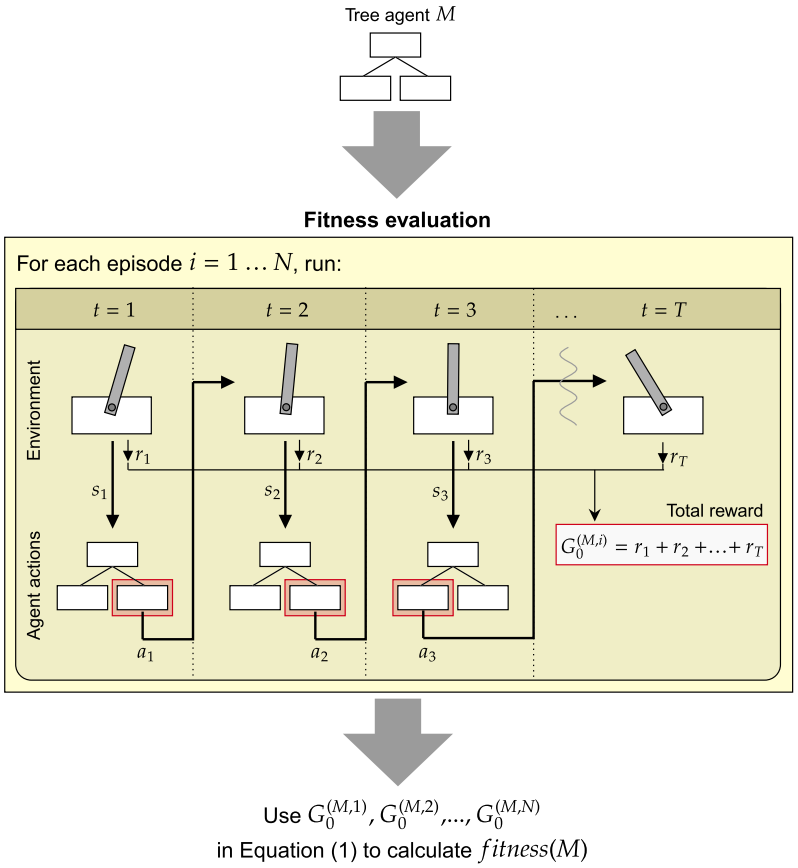


Fig. 2. An illustration of the fitness evaluation procedure. In this example, a tree agent of depth 1 is used on the Cartpole environment. The specifics of how to aggregate $G_0^{(T,i)}$ in order to compute fitness are described in Equation (1).

was empirically observed to greatly improve the quality of obtained solutions since this is the range in which the "modify split" and "reset split" operators work; if this is not included, the algorithm has trouble applying fine-tuning changes to the thresholds in the splits, which end up being either too small to be noticed or too large and catastrophic. A similar normalization was employed in the multivariate DTs of [73].

The final component of the MENS-DT-RL is its *initialization*. The most straightforward way to initialize any evolutionary algorithm is randomly, however, it is known that jumpstarting them with good solutions can greatly enhance the final results [79]. In this work, we experiment with three different initializations for the MENS-DT-RL, which make up its three main configurations:

- *MENS-DT-RL (R)*: random initialization. A depth d is selected and a complete tree with this depth is created by uniformly selecting attributes, thresholds, and action leaves.
- *MENS-DT-RL (IL)*: IL initialization. CART is employed under the DAgger framework to obtain a tree that imitates a high-performing DNN trained on the task; this tree is then added to the initial population for the MENS-DT-RL.
- *MENS-DT-RL (P)*: pruning initialization. This approach takes the previous IL tree and reduces its size without decreasing its performance, in a novel procedure we propose called Reward Pruning. This should provide the MENS-DT-RL with a better initial solution in terms of fitness.

Next, we describe these two last initializations in more detail. The *IL initialization* starts by assuming the existence of an uninterpretable model (called an "expert") that ideally has good performance on the task. Although any model would suffice, we argue that DNNs are particularly fit for the job: they already have widespread usage across the RL community and there are several resources dedicated to facilitating their training [80], which as a whole greatly alleviates the complexity of the requirement. With the expert in hands, this initialization distils it into a DT using an IL technique: this paper specifically uses the DAgger algorithm [40] to distil the expert into a CART tree model [61], with both DAgger and CART being chosen because they are widely used in their own fields and have a long track record of good-quality results.

An important decision in this phase is the regularization parameter of CART (i.e. the cost-complexity coefficient). If this parameter is too low, the resulting tree will be close to the expert but may have an excessive number of nodes, needlessly making the initial fitness worse. On the other hand, if the regularization parameter is too high, the resulting tree will be overly simplified and badly performing, leading to poor results. It is crucial to find a trade-off that maximizes fitness by finding solutions with high performance but low size. In this work, we define this parameter with a heuristic similar to the elbow method of unsupervised learning: namely, to increase the regularization until the model's performance is negatively affected. Crucially, the algorithm does not compromise

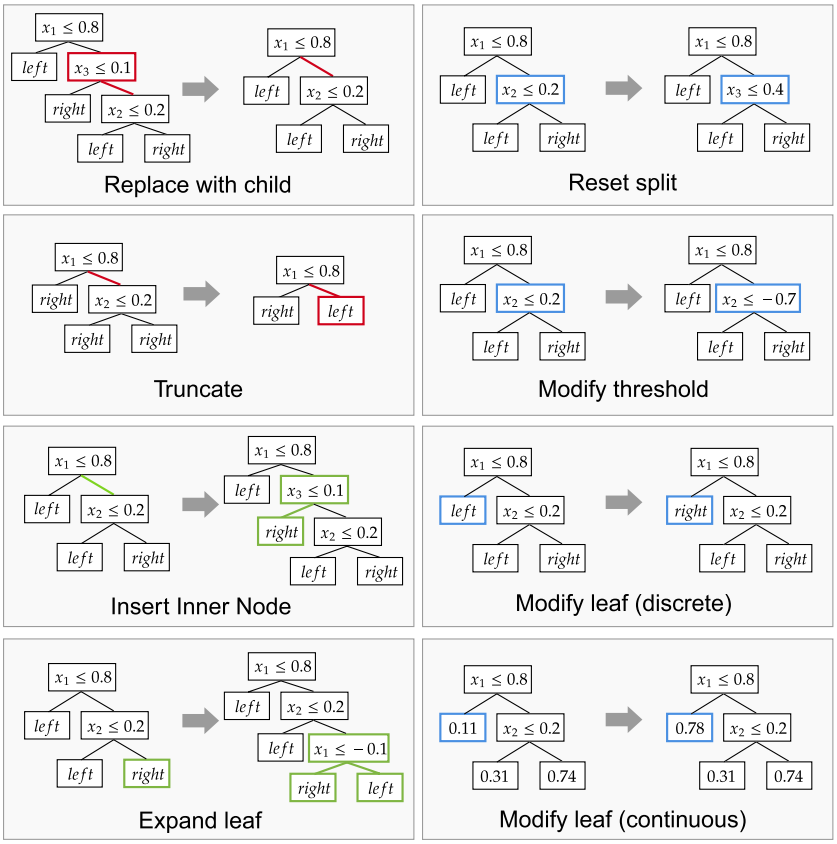


Fig. 3. Operators employed in MENS-DT-RL. In these examples, "left" and "right" are two actions to be taken (the agent moving either left or right).

performance at this stage, as this is already done by the pruning initialization. Further note that, in theory, there is nothing to prevent either DAgger or CART from producing small trees of high quality at this step, but as will be seen in section 4.2, this usually does not happen in practice.

Finally, we describe the *pruning initialization*, which aims to increase the fitness of the initial IL solutions by reducing their size without compromising their performance. The overall approach, which we call Reward Pruning, can be seen as an RL extension of post-pruning algorithms from the DT literature (e.g. CART's cost-complexity pruning and C4.5's error-based pruning). However, while these post-pruning algorithms were proposed in a supervised learning environment where the goal was to reduce overfitting, our goal is instead to reduce tree size and increase interpretability, with any improvement in performance being somewhat incidental. The technique is described in Algorithm 1.

Algorithm 1 Reward Pruning.

Require: Expert agent π^*
  //Imitation phase
  $M \leftarrow$ Apply DAgger to imitate π^* with CART

  //Pruning phase
  fitness$(\cdot) \leftarrow$ Defined in Equation (1)
  SR$(\cdot) \leftarrow$ Fraction of successful episodes by model "$\cdot$" over N episodes
  for round $\leftarrow 1 \dots R$ **do**
    for each node m in M **do** ▷ bottom-up, left-right
      $M' \leftarrow$ Replace m in M with its left child
      if fitness$(M') \geq$ fitness(M) or SR$(M') \geq$ SR(M) **then**
        $M \leftarrow M'$
      else
        $M'' \leftarrow$ Replace m in M with its right child
        if fitness$(M'') \geq$ fitness(M) or SR$(M'') \geq$ SR(M) **then**
          $M \leftarrow M''$
        end if
      end if
    end for
  end for

Essentially, Reward Pruning works by iteratively reducing the tree while monitoring its performance on the task. At each step, the algorithm selects an inner node and replaces it with its left child – if the performance of the tree is maintained or improved, the

Table 1
Parameters used in each environment for the MENS-DT-RL.

Parameter	Cartpole	Mountain Car	Lunar Lander	Maize Fertilization
ρ	0.6	0.8	0.6	0.6
F_b	0.98	0.98	0.5	0.98
F_d	0.05	0.1	0.05	0.05
P_d	0.2	0.4	0.2	0.2
k	1	3	5	1
K	1	1	1	1
α	0.1	0.1	2	0.1
Initial tree depth	3	3	3	3
Generations	200	200	200	50

replacement is kept, otherwise it is discarded and the same procedure is performed for the right child. If neither child proves to be an adequate replacement for the parent, the current step is terminated and the next node is selected. This process is repeated for all nodes in a bottom-up order until it reaches the root. At this point, every inner node has been evaluated at least once, so the procedure can either stop and return the resulting tree, or it can use this resulting tree as input for another "round" of pruning, starting from the bottom once again.

Reward Pruning employs two tests to decide whether a given pruning operation should be retained or reversed. Let M' be a smaller tree obtained by replacing one of the inner nodes of M with one of its children. The simplest way to decide whether to replace M with M' would be to check iff $fitness(M') \geq fitness(M)$, since this would mean that performance has not decreased. However, RL tasks are often stochastic, which means that there is noise in the fitness calculation – even if M and M' were the same, their fitness could be different due to slight variations in episodes. For this reason, Reward Pruning considers not only the fitness but also the success rate of the model: that is, the fraction of episodes among the N executed that have reached the task's predetermined minimum reward threshold. If M has either a lower fitness or success rate than M', it is replaced. This provides a more reliable way to determine whether a particular pruning operation resulted in a better model.

Finally, note that the stochasticity of RL extends to both phases of Reward Pruning, making it a stochastic algorithm: given the same initial uninterpretable model, different runs of Reward Pruning may yield different intermediate trees and different final trees. This is consistent with common RL models and provides another key difference between Reward Pruning and traditional post-pruning algorithms.

4. Experiments and discussion

In this section, we want to answer the following questions:

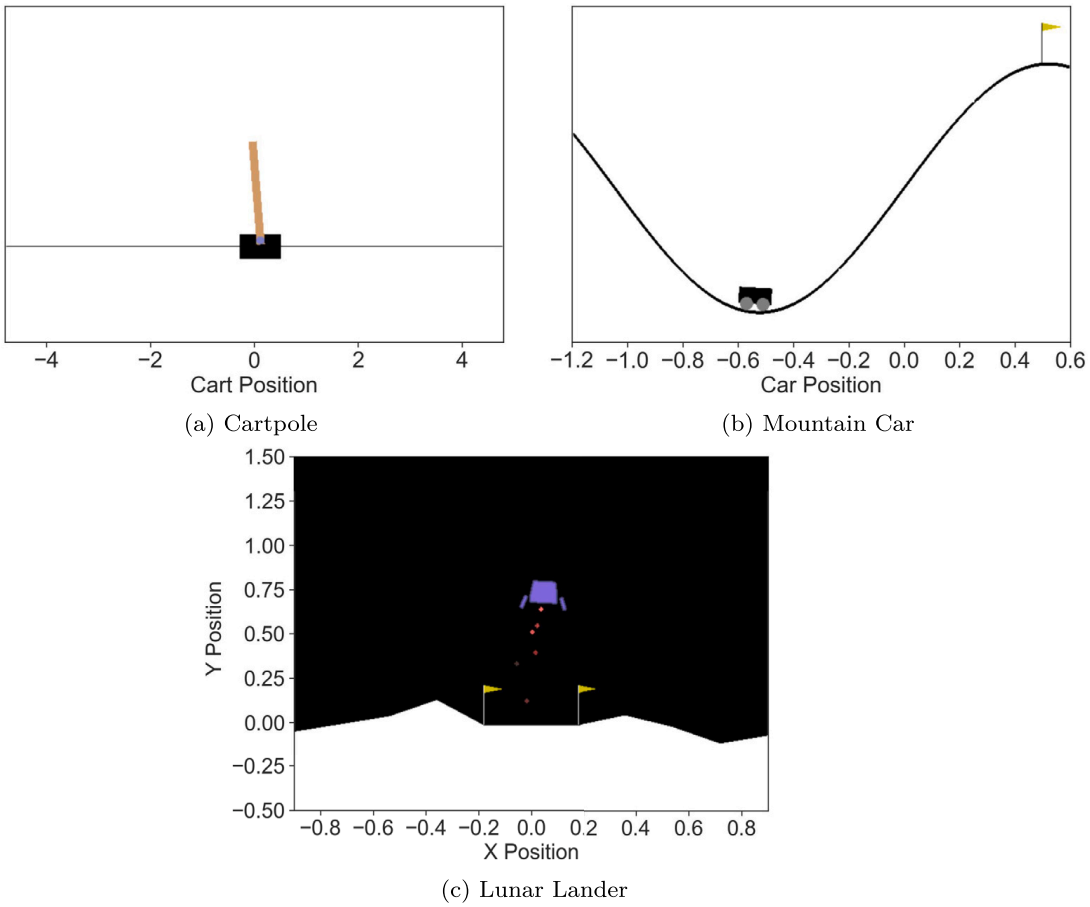
1. How does MENS-DT-RL compare to other algorithms, in terms of produced tree size and performance?
2. What effect does the initialization have on the MENS-DT-RL's results?
3. Can the resulting trees be interpreted?
4. Is there diversity in the solutions produced?

To this end, we conduct experiments with the MENS-DT-RL across four different RL environments: three benchmarks from the widely used OpenAI Gym framework [28], and an agronomical environment based on real-world crop management [29]. The results are compared with other methods from the literature, primarily through three different measures: episodic reward (which measures the performance of the agent in the task), tree size (which serves as a proxy for the interpretability of the resulting agent, with smaller trees being more interpretable), and success rate (which condenses the reward information by assessing the proportion of episodes that resulted in success). For every OpenAI Gym benchmark, a perfect success rate of 1.0 is attainable, as shown in the leaderboards available in the official documentation.[2] For the Maize Fertilization environment, to the best of our knowledge, the highest performance was achieved by the PPO agent released by the environment's own authors, which reached a success rate of around 0.8.

Each algorithm was run for 50 simulations (30 in the case of crop management due to the simulator's high computational cost), with each simulation resulting in a single univariate tree. To calculate the fitness of each individual during the execution of the algorithm, 100 episodes are run. All values reported were obtained by re-running the solutions on another 1,000 episodes. Furthermore, since the environments are stochastic, each tree has an average and a standard deviation for its reward and success rate, which means that the values reported here are the average of these averages and standard deviations. However, while the average behavior of each algorithm is useful for comparison, it is also important to understand that in practice only a single solution is needed, which is why we also report the best solution rather than the average. Table 1 contains the parameterizations for each environment, defined on the basis of preliminary tuning.

As described in Section 3, three different configurations for the MENS-DT-RL are considered, differing primarily in their initialization. The first configuration, called "MENS-DT-RL (R)", is randomly initialized with randomly generated complete trees of

[2] https://github.com/openai/gym/wiki/Leaderboard.



(a) Cartpole (b) Mountain Car

(c) Lunar Lander

Fig. 4. Illustration of the three OpenAI Gym environments (Maize Fertilization is not pictured since it has no visual component).

depth 3 (i.e. 15 nodes) – this provides the algorithm with maximum diversity, but also no solution from which it can jumpstart its performance. The second configuration, referred to as "MENS-DT-RL (IL)", initializes the population with trees obtained via IL. In particular, the i-th simulation of "MENS-DT-RL (IL)" is initialized with a population consisting of randomly generated trees of depth 3 and the i-th tree obtained from IL. The third and final evaluated configuration, called "MENS-DT-RL (P)", follows the same scheme as the previous one but applies 10 rounds of Reward Pruning to the IL tree before adding it to the initial population. In addition, the experts used in IL are dense neural networks with 2 hidden layers of 32 nodes each, trained with deep Q-Learning and experience replay – the only exception is the crop environment, where we use the PPO model included in the library itself [29].

All three configurations are compared with different solutions from the literature. Given the novelty of interpretable AI and interpretable RL specifically, there are a scant few works which produce interpretable agents for the three environments employed here. The authors from [73], when faced a similar situation in their work on evolutionary trees for RL, remedied this by proposing an interpretability score that could be applied not only to DTs but to any machine learning model, therefore creating a common ground in which any pair of algorithms could be compared in terms of interpretability. Although this sounds intuitive, we avoid this approach for two main reasons. The first is that this does not provide any additional information of particular importance: in terms of interpretability, the trees indeed almost always surpass any of the neural network-based models that are popular in RL, but this is already known since DTs are widely known to be more interpretable than the alternatives. The second reason is that even in the rare case where the trees are indeed surpassed by some other method in terms of interpretability, for example, a medium-sized tree against a very small neural network, it is debatable whether the neural network truly could be considered "interpretable" in the common sense of the term, or if it is only achieving a better interpretability score simply because it is "gaming the metric" (e.g. it has fewer mathematical operations involved, while still having a hard-to-grasp multi-layered structure). As such, we opt to compare the DTs obtained here only with similar DTs from the literature.

All experiments were implemented on Python 3.10 programming language. The code is fully available at `https://github.com/vgarciasc/CRO_DT_RL`.

4.1. Environments

4.1.1. Cartpole

The Cartpole environment (illustrated on Fig. 4a) involves balancing a pole on a cart by moving the cart horizontally. The state consists of four different attributes: the cart's position and velocity, and the pole's angle and angular velocity. The actions involve either pushing the cart left or right. The reward is always +1 until the episode terminates, rewarding every step that the pole is kept

upright. The termination is achieved when the cart reaches the edge of the region of interest, if the episode reaches 500 steps, or if the pole's angle is greater than 12 degrees in any direction (in which case it is considered to have fallen). As such, the accumulated maximum reward for this environment is 500 points. In this work, we consider that an episode is successful if the accumulated reward is larger than 495 points.

4.1.2. Mountain Car

In the Mountain Car environment (illustrated in Fig. 4b) [81], the agent controls a car in a valley between two mountains, such that the goal is to make the car reach a flag on top of the rightmost mountain. Since the car does not have enough traction to climb this mountain by itself, it must carefully build up momentum to achieve a higher speed and reach the flag. Two attributes are available: the car's velocity, and its position along the X-axis. With regards to actions, there are three: accelerating to the left, to the right, or not accelerating at all. The reward is -1 for every step during which the flag is not reached, awarding agents that reach the flag faster. The episode terminates when the flag is reached or if 200 steps are executed, leading the worst possible solution to be an accumulated reward of -200 points. In this work, a successful episode of Mountain Car involves not only reaching the flag but doing so with an accumulated reward lesser than -110 points.

4.1.3. Lunar Lander

The Lunar Lander environment (illustrated in Fig. 4c) tasks the agent with landing a rocket upright on a flat surface. To this end, the agent must carefully balance each of its four actions: doing nothing, firing the main engine (located at the bottom of the ship), the left engine, or the right engine. Eight attributes are made available: the X and Y coordinates of the agent, its X and Y linear velocities, its angle and angular velocity, and two boolean attributes that signal whether or not each leg is touching the ground. Rewards for Lunar Lander involve several components. Crashing results in -100 points, while landing correctly results in +100 points. Each leg that contacts the ground results in +10 points. Furthermore, the agent must save fuel: it is penalized for -0.3 points each frame it fires the main engine, as well as -0.03 points for every side engine. The episode terminates if the agent crashes, if it lands correctly, or after 1000 timesteps. The episode is considered to be successful if the accumulated reward is larger than 200 points.

4.1.4. Maize Fertilization

The Maize Fertilization environment is a task modeled in `gym-DSSAT` [29], a Python RL wrapper for the widely celebrated Decision Support System for Agrotechnology Transfer (DSSAT; [82,83]) crop model. In this task, the goal is to design a nitrogen fertilization policy for a maize crop field, taking into account each day's data in order to determine how much nitrogen should be added to the soil. Adding no nitrogen may result in nitrogen deficiency and a reduced harvest, while adding excessive nitrogen involves a high financial cost and may result in water pollution, among other undesired effects. The agent must take all this into consideration to decide both (1) when to add fertilizer, and (2) how much fertilizer to add. Each episode corresponds to a growing season and contains around 160 days, with each day being a timestep. There are 12 different attributes each day, ranging from weather data (rainfall and temperature degrees) to current plant information (total biomass and growth stage). The reward is a function of both the plant nitrogen uptake and the quantity of fertilizer added, while the action at each step is a continuous value that denotes the quantity of fertilizer to add (from 0 to 200 kg/ha). For a more detailed description of the environment, we refer to [29]. In our paper, we define the success rate to be a final cumulative reward of 60.

4.2. Results

4.2.1. Cartpole

Cartpole is the simplest of the three environments, and it is expected that most approaches should perform well on it. Indeed, as it can be seen in Table 2, all three configurations of MENS-DT-RL were able to find a tree with 5 nodes and a perfect reward of 500.0 ± 0.0, which means that for every episode out of 1000 tested, these best agents managed to balance the pole for the maximum amount of time (500 timesteps). Although all three configurations reached this solution, there are slight differences in the evolutionary process: Fig. 5(a) shows that the IL and pruning initializations start very close to the final solutions, while the randomly initialized MENS-DT-RL (R) needs some generations to catch up, since it starts from scratch. On average, the three configurations have already converged by generation 100, and simply maintain their results from then on. Furthermore, Fig. 5(c) shows that MENS-DT-RL does not get stuck on the initialized solutions, since both MENS-DT-RL (IL) and MENS-DT-RL (P) reduce their initial average tree sizes over time. In addition, this figure also shows that MENS-DT-RL (R) actually ends up with a smaller average tree size than the other two approaches, a fact reflected in Table 2 – this may be due to the small size of the best solution, which actually benefits from the random search aspect of this configuration. In terms of average reward and success rate, there is little difference between the averages of these three configurations, as they are all close to the perfect reward of 500.0 ± 0.0. Given these results, MENS-DT-RL (R) appears to be the best configuration of the three, as it is able to reliably produce the best solution without incurring the cost of requiring an expert model.

Curiously, the existence of an optimal tree size does not imply the existence of a single optimal tree. By analyzing the results from the MENS-DT-RL configurations, we can see that there are different trees of size 5 that obtain a perfect success rate in the task: Fig. 6 shows three of them. The first one relies on Pole Angle and Pole Angular Velocity, however, the last two do not rely on the Pole Angular Velocity at all and instead use the Cart Velocity, differing between themselves with regards to how these two attributes are used. By analyzing the execution of these three solutions, it is indeed possible to see that there are differences in how they behave: as is shown in Fig. 7, trees A and B keep the pole in a tight range of [-0.05, 0.05] radians, while tree C lets the pole reach larger angles

Table 2
Results for the Cartpole task. (*): as reported in the original paper. (†): re-executed on 1000 episodes based on the reported best solution from the original paper. (‡): solution taken from [84]. Best solutions are highlighted in bold.

	Average results			
	Avg. Reward	Avg. Stdev. Reward	Average SR	Avg. #Nodes
Neural Network Expert	500.00	0.00	1.00	—
MENS-DT-RL (R)	**499.92**	**0.33**	**1.00**	**6.40**
MENS-DT-RL (IL)	499.98	0.27	1.00	6.84
MENS-DT-RL (P)	499.99	0.18	1.00	6.88
Custode and Iacca, 2023 [73]*	497.34	5.19	1.00	–

	Highlighted solutions			
	Avg. Reward	Std. Dev. Reward	Success Rate	#Nodes
MENS-DT-RL (R)	**500.00**	**0.00**	**1.00**	**5**
MENS-DT-RL (IL)	**500.00**	**0.00**	**1.00**	**5**
MENS-DT-RL (P)	**500.00**	**0.00**	**1.00**	**5**
Custode and Iacca, 2023 [73]*†	**499.67**	**5.98**	**1.00**	**5**
Silva et al., 2020 [20] †‡	499.80	2.71	1.00	13

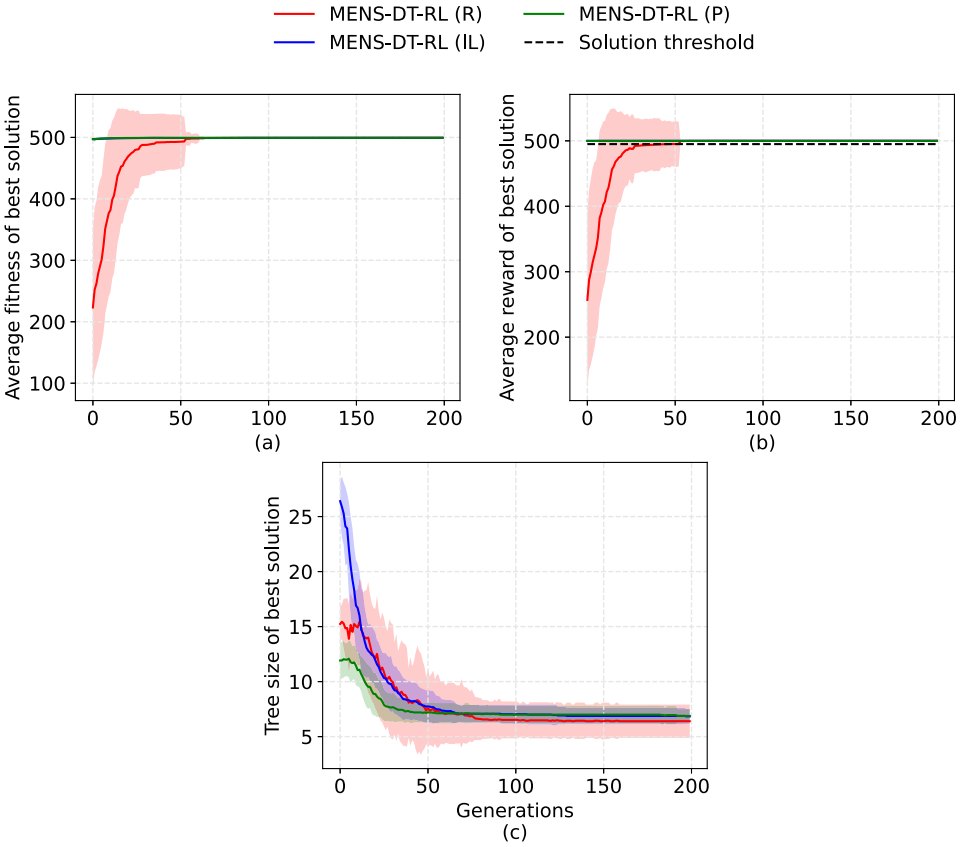


Fig. 5. Average metrics across generations for the fittest individual of each generation in Cartpole, across the three configurations of MENS-DT-RL. Each plot shows the evolution of a particular metric: (a) Fitness, (b) Reward, (c) Tree size. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

before balancing it again. Furthermore, tree A has a slight preference for a right tilt, while tree B is the opposite. This exemplifies one of the key advantages of the evolutionary approach, which is the variety of produced solutions: due to the stochastic nature of evolutionary algorithms, the end-user has the power to choose an agent based on their own preferences of size, performance, and even features used.

To the best of our knowledge, two works report univariate DTs for the Cartpole task: [73] and [20]. The former does not report the average number of nodes of the produced trees but does report the average reward and standard deviation, which also closely approximate the values reported by the MENS-DT-RL (while being slightly lower). In addition, the authors report the best

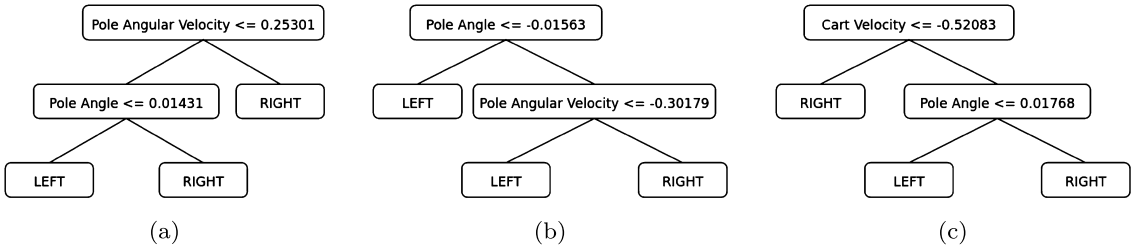


Fig. 6. Three of the best solutions obtained for Cartpole by the MENS-DT-RL approaches.

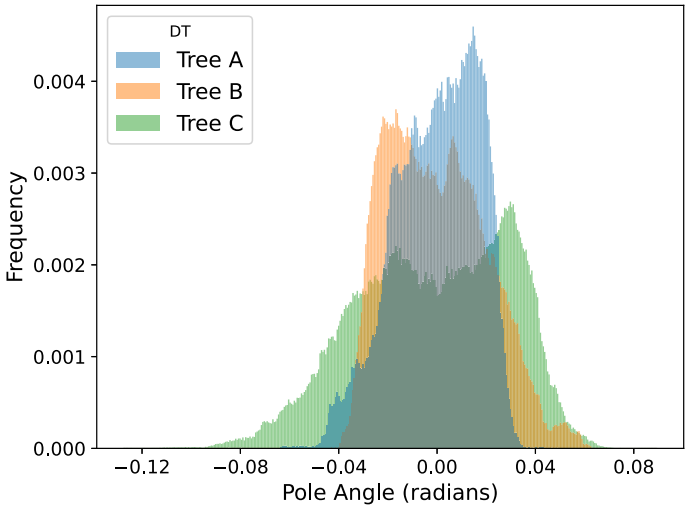


Fig. 7. Distribution of Pole Angle across 1,000 episodes by the three trees in Fig. 6.

solution found, which we re-implemented and executed for another 1000 test simulations to report here; this result not only achieves the maximum success rate but also contains 5 nodes, which reflects the best solutions found by MENS-DT-RL. Therefore, it seems highly likely that this is indeed the lowest size for which a univariate tree can achieve maximum reward for the Cartpole problem. Meanwhile, [20]'s gradient-based approach also obtained a solution with perfect reward, but with 13 nodes – higher than the solutions found by MENS-DT-RL and [73].

4.2.2. Mountain Car

Mountain Car is a simpler problem than Cartpole in terms of state attributes, but due to its sparse rewards it is actually harder to solve: a good solution that almost touches the flag and a bad solution that never leaves the starting position both have the minimum reward of -200. Fig. 8 compares the average performance of the three configurations over the evolutionary process. As it can be seen, MENS-DT-RL (R) is able to greatly improve its fitness by increasing the average reward and reducing both the tree size and the standard deviation of the reward, but although its final trees are only slightly larger than those of the other two configurations, its average reward pales in comparison and fails to reach the task solution threshold.

On the other hand, both MENS-DT-RL (IL) and MENS-DT-RL (P) are able to produce good quality results. Table 3 shows that the two initializations produce solutions with an average success rate of 0.74, a value that greatly exceeds the success rate of 0.28 obtained by the MENS-DT-RL (R). Moreover, note that this result is obtained even though the DNN expert has a success rate of only 0.39, indicating that the initial solutions do not need to be perfect to benefit from MENS-DT-RL. This greatly reduces the effort required to train the expert, which in turn reduces the computational cost of using IL-based initialization and makes it easier to use. Fig. 8 further shows that MENS-DT-RL improves these two initializations, as the average fitness increases even though it starts in a good range. This improvement is mainly due to a reduction of the average standard deviation, which means that the solutions become more consistent in their performance, even though the average reward is maintained. Note that if the fitness did not include the standard deviation, MENS-DT-RL would ignore it and such an improvement in consistency would not take place.

Table 3 shows the best solutions obtained in this environment. As it can be seen, all three configurations are able to produce a tree with 9 nodes, an approximate reward of -102 $\pm$ 0.7, and a success rate of 1.0 – even the randomly initialized MENS-DT-RL (R), which has a worse performance on average. Note that this solution is not only small enough to be highly interpretable, but also has a better success rate than the neural network expert: given the previous remark that both MENS-DT-RL (IL) and MENS-DT-RL (P) have a higher average performance than MENS-DT-RL (R), it can be concluded that the evolutionary algorithm effectively takes the decent solutions from the expert and improves them towards an optimal behavior. In addition, these algorithms also find some solutions that although not the best in terms of fitness, are also very noteworthy: trees that also have a perfect success rate of 1.0 but with a smaller size of 7 nodes (see Table 3). Their average reward of about -106 $\pm$ 1.3 is slightly worse than that of the fittest solutions, but depending on the user's priorities, they might actually be preferable due to their smaller size and similar success rate.

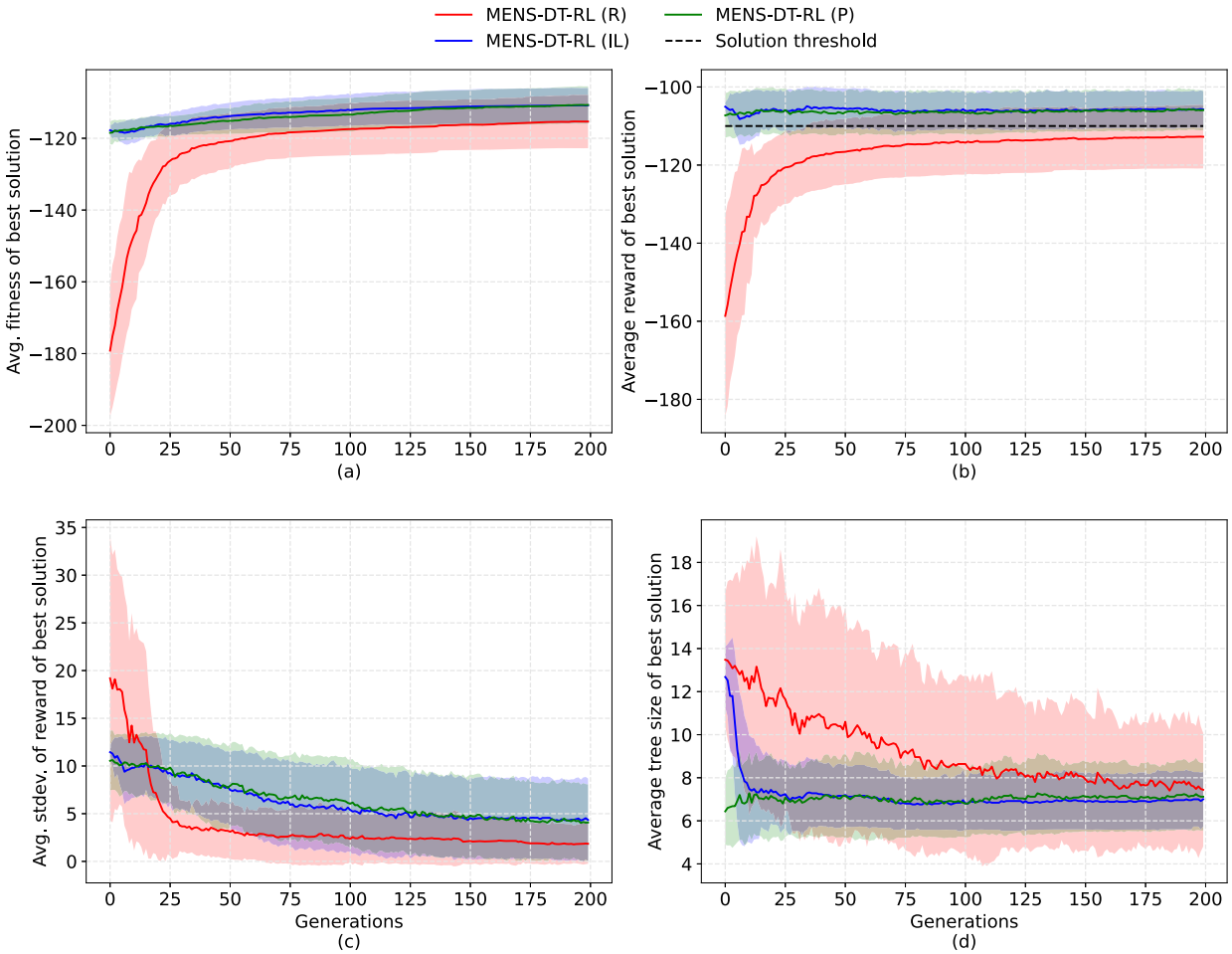


Fig. 8. Average metrics across generations for the fittest individual of each generation in Mountain Car, across the three configurations of the MENS-DT-RL. Each plot shows the evolution of a particular metric: (a) Fitness, (b) Reward, (c) Standard Deviation of Reward, (d) Tree size.

Table 3
Results for the Mountain Car task. (*): as reported in the original paper. (**): re-executed on 1000 episodes based on the reported best solution in the original paper. Best solutions are highlighted in bold.

	Average results			
	Avg. Reward	Avg. Stdev. Reward	Average SR	Avg. #Nodes
Neural Network Expert	-113.03	24.08	0.39	–
MENS-DT-RL (R)	-112.98	2.08	0.28	7.44
MENS-DT-RL (IL)	**-106.30**	**4.49**	**0.74**	**7.00**
MENS-DT-RL (P)	**-106.68**	**5.94**	**0.74**	**7.12**
Custode and Iacca, 2023 [73] *	-108.16	–	0.70	–

	Highlighted solutions			
	Avg. Reward	Std. Dev. Reward	Success Rate	#Nodes
MENS-DT-RL (R)	**-102.53**	**0.90**	**1.00**	**9**
MENS-DT-RL (IL)	**-102.79**	**0.71**	**1.00**	**9**
MENS-DT-RL (P)	**-102.79**	**0.70**	**1.00**	**9**
MENS-DT-RL (IL)	-106.54	1.22	1.00	7
MENS-DT-RL (P)	-106.46	1.33	1.00	7
Custode and Iacca, 2023 [73] **	-101.72	3.04	1.00	13
Dhebar et al. 2020 [74] *	-105.82	21.77	0.83	–

This is another example of the diversity of solutions obtained by the MENS-DT-RL approach, and of the choices that this gives to the end user. These solutions are shown in Fig. 9.

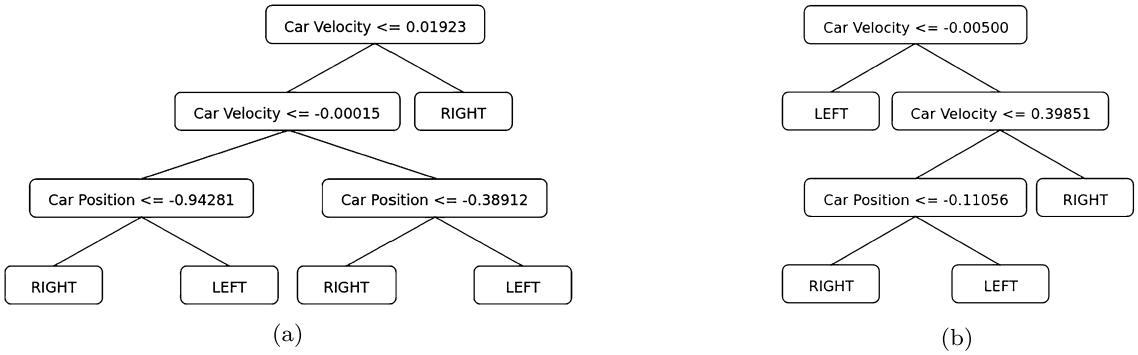


Fig. 9. Two of the best solutions obtained for Mountain Car by the MENS-DT-RL approaches (obtained by MENS-DT-RL (R) and MENS-DT-RL (P), respectively).

Finally, we compared our results with those reported in the literature. Our main point of comparison is the work of [73], which, to the best of our knowledge, is the only contribution that has experimented with univariate trees for the Mountain Car environment. The authors reported both a lower average reward and a lower success rate than those obtained by our two most sophisticated initializations, which indicates the advantage of using MENS-DT-RL, but it must be noted that in their work only 10 simulations were performed, while our results are the average of 50. Given this situation, it might be more informative to look at the best solution. The best solution found by [73] was a tree with 13 nodes and a perfect success rate of 1.00, which can be directly compared with the trees obtained in Fig. 9, both of which have a perfect success rate while having a smaller tree size (4 and 6 fewer nodes, respectively). This indicates that MENS-DT-RL is able to produce more interpretable solutions without compromising performance, even when randomly initialized. Moreover, we note that in [74] the authors also experimented with DTs for the Mountain Car environment, but instead of using univariate splits, they used highly nonlinear splits, which simply do not have the same level of interpretability as traditional univariate DTs. To the best of our knowledge, the trees reported here with maximum success rate and 7 or 9 nodes are the best interpretable DT solutions for the Mountain Car benchmark.

4.2.3. Lunar Lander

Lunar Lander is the most complicated environment of the benchmarks presented here, with 8 attributes, 4 different actions, and a movement dynamic that is much more complex than that of the previous two tasks. Such complexity may not be a problem for a model like the expert DNN (which achieves a success rate of 0.97), but it poses a clear challenge to the simplicity of univariate DTs, especially in terms of tree size: good solutions require much larger trees than those developed for Cartpole and Mountain Car.

Table 4 shows the results for this environment, where it can be seen that MENS-DT-RL (P) outperforms both the random and the IL initializations. While the randomly initialized MENS-DT-RL (R) is able to produce good solutions for Cartpole and Mountain Car, in Lunar Lander it is unable to evolve any tree with a success rate higher than 0.1 and ends up with an average success rate of 0. Such a negative result is a direct consequence of the complexity of the environment: since good solutions have more nodes and are easily broken by even small changes, it is strikingly difficult for the evolutionary approach to find such solutions when starting from scratch; that is, the topology of the solution space is not amenable to encountering good solutions based on what amounts to random search. Fig. 11(b) shows that while the average reward of this configuration improves over time, it is far from the task solution threshold of 200.

Table 4
Results for the Lunar Lander task. Best solutions in terms of fitness are highlighted in bold.

	Average results			
	Avg. Reward	Avg. Stdev. Reward	Average SR	Avg. #Nodes
Neural Network Expert	254.32	31.71	0.97	—
MENS-DT-RL (R)	-52.39	32.39	0.00	5.76
MENS-DT-RL (IL)	242.13	55.68	0.92	278.76
MENS-DT-RL (P)	**223.46**	**77.10**	**0.84**	**50.04**

	Highlighted solutions			
	Avg. Reward	Std. Dev. Reward	Success Rate	#Nodes
MENS-DT-RL (R)	-54.93	128.12	0.06	7
MENS-DT-RL (IL)	245.12	40.50	0.93	187
MENS-DT-RL (P)	**237.70**	**72.51**	**0.91**	**37**
Silva et al. 2020 [20]	-78.40	32.20	0.00	19
Dhebar et al. 2020 [74]	234.98	22.25	0.99	–

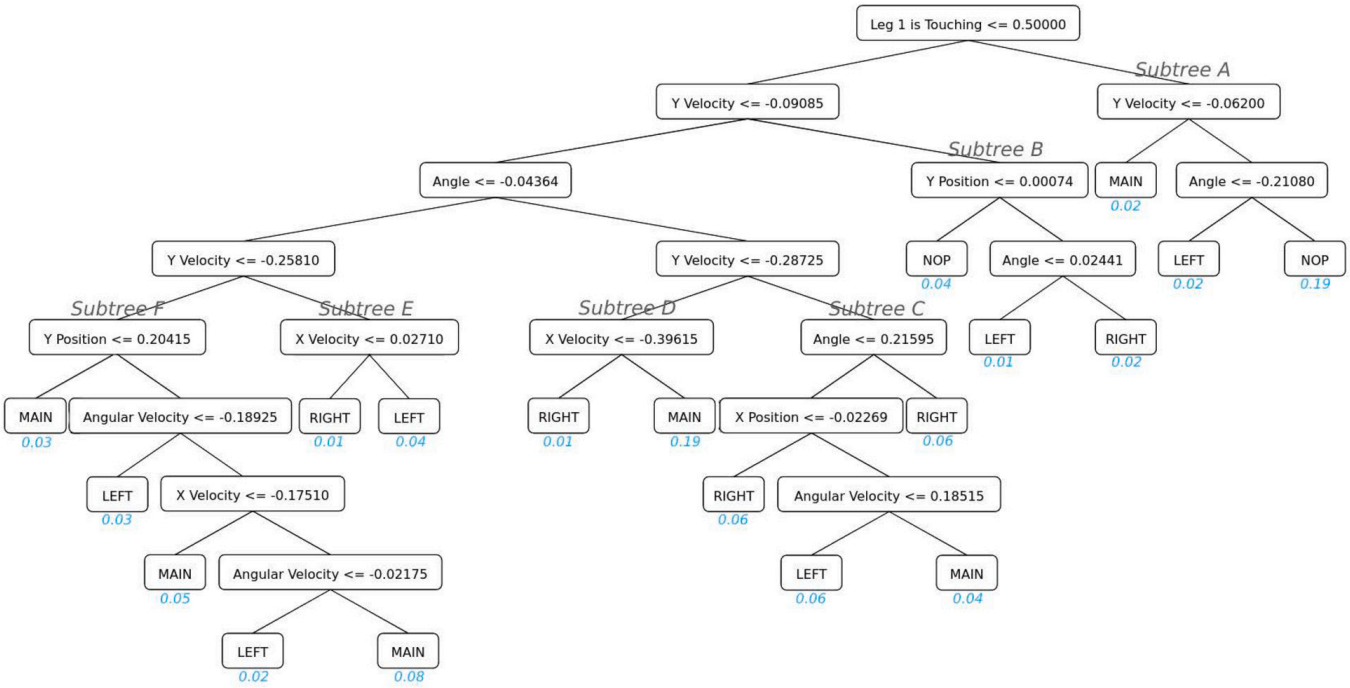


Fig. 10. Best solution found for the Lunar Lander environment, obtained by MENS-DT-RL (P). Labels in blue are the percentage of visits for every leaf.

Initializing MENS-DT-RL with IL solutions, however, yields more interesting results: the average success rate is 0.92 with an average tree size of 278.76, which means that MENS-DT-RL reduces the tree size of the IL trees while sacrificing some small amount of performance. This is illustrated in Fig. 11, where it can be seen that the average reward of MENS-DT-RL (IL) is mostly stagnant over time, while its size continues to be reduced. This shows that MENS-DT-RL is indeed capable of handling solutions to a more complex task like Lunar Lander, but it needs good initial solutions from which to start its search. However, the trees generated by this configuration are too large to be interpreted, which explains why Fig. 11(a) shows that MENS-DT-RL (IL) has a much lower average fitness than the poorly performing MENS-DT-RL (R).

This challenge can be overcome by using a pruned initialization. Since Reward Pruning reduces the size of the IL trees while trying to maintain their performance, MENS-DT-RL (P) is given a much better fitness from the start, leading it to outperform the other two configurations. Furthermore, Fig. 11 shows that this fitness continues to improve over time, mainly due to further reductions in the tree size and the standard deviation of the reward, i.e. the solutions become smaller and more consistent. While the average success rate of MENS-DT-RL (P) is slightly lower than that of MENS-DT-RL (IL) (0.84 vs. 0.92), its average tree size is an order of magnitude smaller (50.04 vs. 278.76), resulting in trees that not only frequently solve the task, but are also in a better range of interpretability. Furthermore, it is also important to look at the best solution, since this is what the end user will actually use: this solution, shown in Fig. 10, has a 91% success rate and a tree size of 37 nodes, a combination of interpretability and performance that, to the best of our knowledge, has not yet been achieved by any other work.

Three papers in the literature have tackled the Lunar Lander environment with DTs. Silva et al. [20] applied univariate trees to the task in the form of rule lists, but were unable to obtain solutions that solved the problem, reporting an average reward of -78.4 with a standard deviation of 32.2. Similarly, Custode and Iacca [73] attempted to extend their evolutionary approach to this task, but did not find a configuration that gave satisfactory results, reporting only a multivariate tree instead. Finally, Dhebar et al. [74] reported a DT with a success rate of 0.99 that consistently solves the problem, but it employs several nonlinearities that call into question the interpretability aspect. To the best of our knowledge, this is the first time that a high-performing interpretable solution using univariate DTs has been proposed for this benchmark.

4.2.4. Maize Fertilization

The last of the four simulated environments is Maize Fertilization, a task made difficult not only by its large state space (12 different attributes) and complex internal dynamics (controlled by the DSSAT crop management system) but also by the continuity of its actions: at each step of the simulation, instead of choosing from a list of predetermined actions, the agent must specify a quantity (how much fertilizer to place in the soil that day). Although this is quite natural for a complex neural network to represent, it poses a challenge for DTs: while it is possible to modify the leaves of the tree to represent continuous values instead of categorical ones (as in Regression Trees), this limits the possible outputs of the tree to the number of different leaves it has, and thus puts smaller trees at a disadvantage with regards to action sensitivity. Some authors try to bypass this restriction by including linear models in the leaves (the so-called Model Trees [16]), however since this approach compromises interpretability, in this paper we chose to restrict every leaf to a single value and grow the tree if needed.

Table 5 contains the simulation results. As it can be seen, the neural network expert is able to achieve a 0.80 success rate, with an average reward of 59.22 which is very close to the success threshold of 60. This is expected of deep RL models and indeed closely

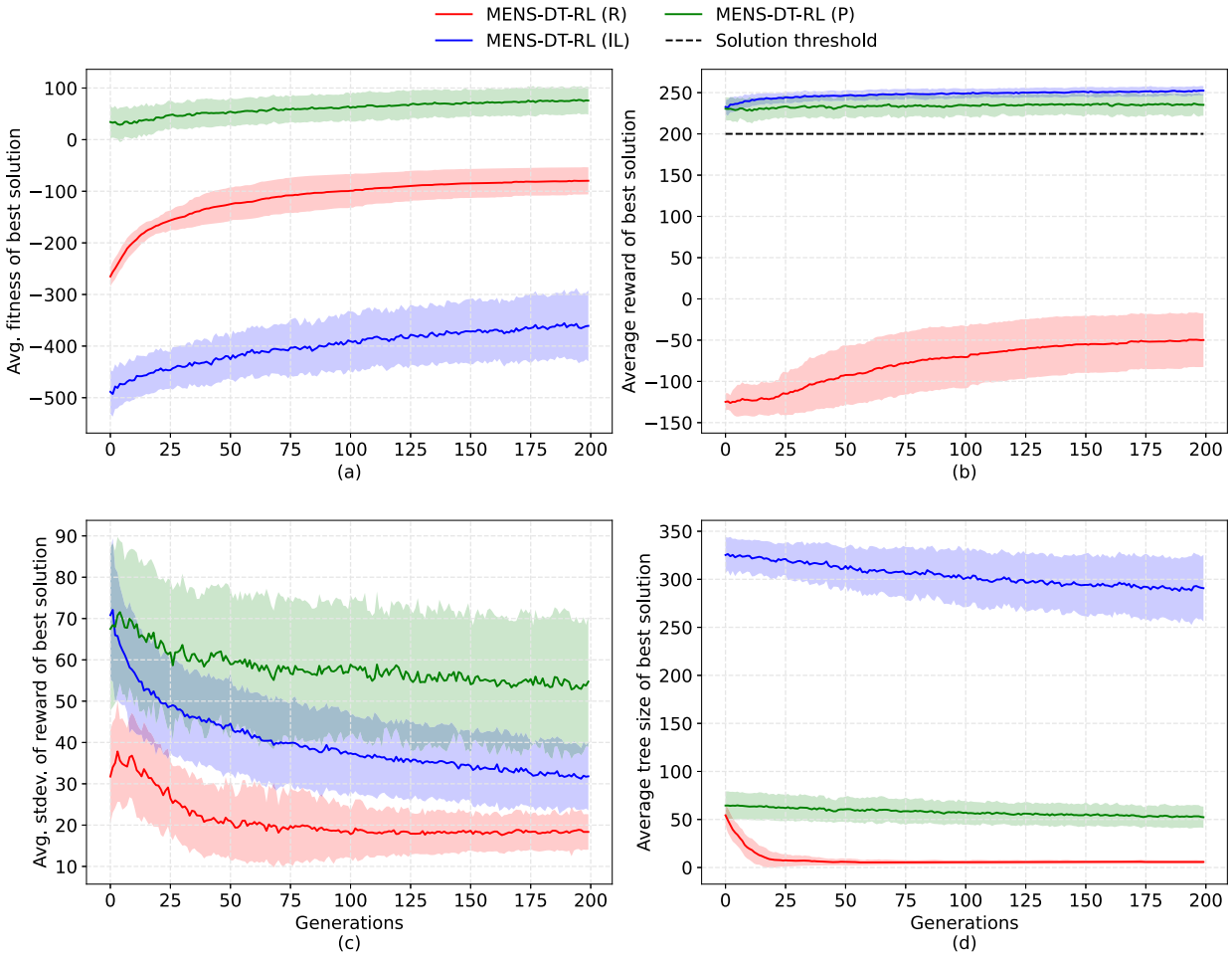


Fig. 11. Average metrics across generations for the fittest individual of each generation in Lunar Lander, across the three configurations of MENS-DT-RL. Each plot shows the evolution of a particular metric: (a) Fitness, (b) Reward, (c) Standard Deviation of Reward, (d) Tree size.

mirrors the benchmark results showed in [29]. In contrast, MENS-DT-RL is unable to achieve such a high-performing behavior by itself: its random initialization version has an average reward of 2.64 and an average success rate of 0.35, both of which fall far below the solution quality established by the expert. As it has been shown in previous environments, leveraging this expert is key to achieving much better solutions. By using Imitation Learning, MENS-DT-RL (IL) is able to obtain a similar performance as the expert while bringing the policy to the DT domain, attaining a success rate of 0.80 and an average reward of 57.33. However, it is still questionable if these solutions can be clearly interpreted since the average tree size is 69.73. MENS-DT-RL (P) is able to bring a final improvement through the Reward Pruning approach, maintaining a near-identical success rate as MENS-DT-RL (IL) and a very similar average reward (56.74 against 57.33), all the while achieving an average tree size of 9.93 that is much more within the realm of ordinary DT interpretability. Fig. 12 reinforces these conclusions: MENS-DT-RL (P) has a very similar reward to MENS-DT-RL (IL), all the while having a similar tree size to MENS-DT-RL (R). Furthermore, the fitnesses of all three configurations have different rates of increase over time: MENS-DT-RL (R) is the fastest-growing of the three, since it has so much to improve, and conversely MENS-DT-RL (P) is the slowest, since the Reward Pruning solutions it starts from are already of very high quality. MENS-DT-RL (IL) falls squarely in the middle, since its initial solutions have good reward but high tree size.

The same relationship between models can be found by looking at the best solutions obtained by each configuration. Although MENS-DT-RL (R) has a low average performance, its best solution is quite decent: a tree of 5 nodes and success rate of 0.70. Considering the results from previous environments, we get a further indication that MENS-DT-RL (R) is able to eventually find good-quality solutions given that their tree size is not too high (as happens for Cartpole and Mountain Car, but not for Lunar Lander). As for MENS-DT-RL (IL), its best solution has a slightly higher success rate as the expert, with a tree size of 13 nodes. The best solution overall comes from MENS-DT-RL (P), which maintains this high success rate while having a reduced tree size of only 5 nodes. This tree is depicted in Fig. 13.

We compare this DT agent against the two baseline policies proposed in the original paper: Null Policy and Expert Policy [29]. *Null Policy*, as the name suggests, consists of adding no fertilizer throughout the period, therefore incurring no fertilizer costs at the expense of a worse harvest – this can be clearly seen in Fig. 14, where the policy's cumulative reward is shown to be entirely non-decreasing. *Expert Policy*, on the other hand, adds a fixed amount of fertilizer on three pre-specified days of the growing season (based on the experiment done by [85]): this results in the three drops in Fig. 14, which decrease reward in the short term but end up resulting in a final larger cumulative reward than the Null Policy. These two policies can be easily represented by DTs of size 1

Table 5
Results for the Maize Fertilization task. Best solutions in terms of fitness are highlighted in bold. (*): Number of tree nodes of the policies if they were represented as DTs.

	Average results			
	Avg. Reward	Avg. Stdev. Reward	Average SR	Avg. #Nodes
Neural Network Expert	59.22	13.77	0.80	—
MENS-DT-RL (R)	2.64	27.71	0.35	12.13
MENS-DT-RL (IL)	57.33	12.83	0.80	69.73
MENS-DT-RL (P)	**56.74**	**12.19**	**0.79**	**9.93**

	Highlighted solutions			
	Avg. Reward	Std. Dev. Reward	Success Rate	#Nodes
MENS-DT-RL (R)	56.16	18.15	0.70	5
MENS-DT-RL (IL)	57.38	11.74	0.83	13
MENS-DT-RL (P)	**57.70**	**11.23**	**0.83**	**5**
Null Policy [29]	42.49	5.95	0.08	1*
Expert Policy [29,85]	55.22	29.44	0.59	7*

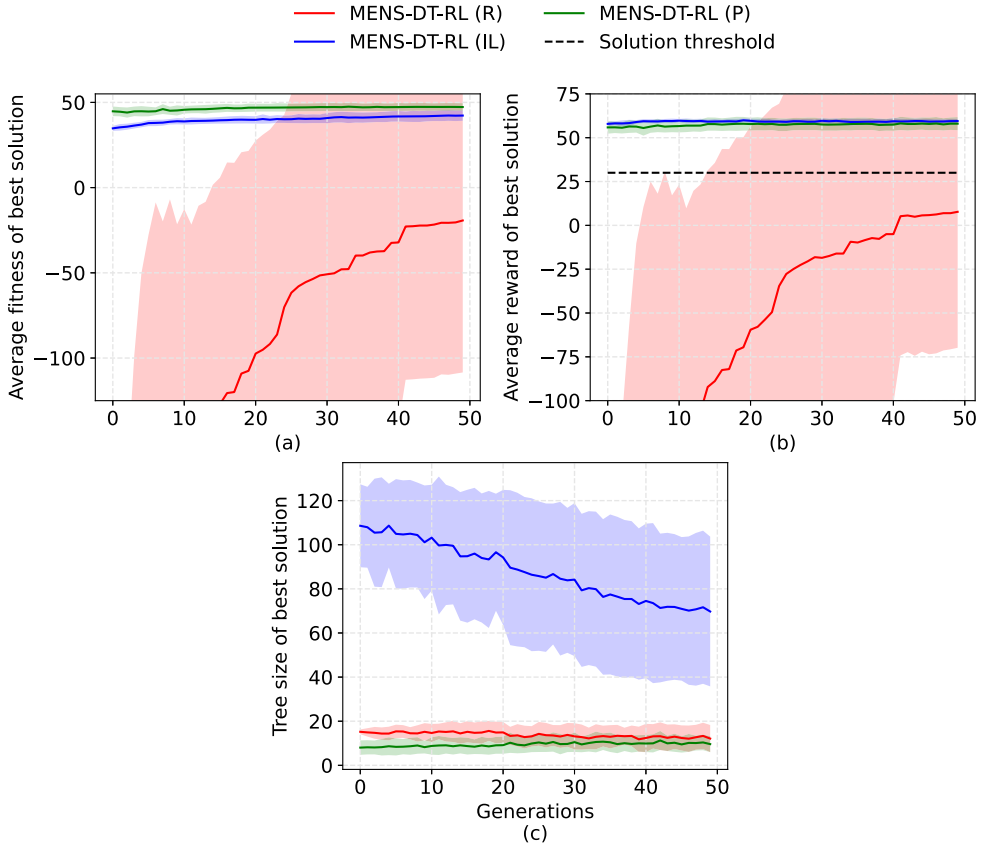


Fig. 12. Average metrics across generations for the fittest individual of each generation in the Maize Fertilization environment, across the three configurations of MENS-DT-RL. Each plot shows the evolution of a particular metric: (a) Fitness, (b) Reward, (c) Tree size.

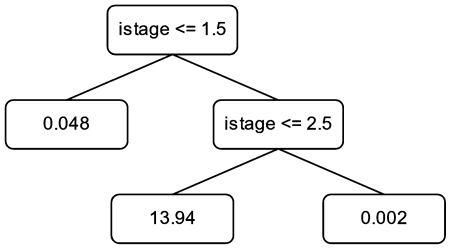


Fig. 13. Best solution found for the Maize Fertilization environment (obtained by the MENS-DT-RL (P)).

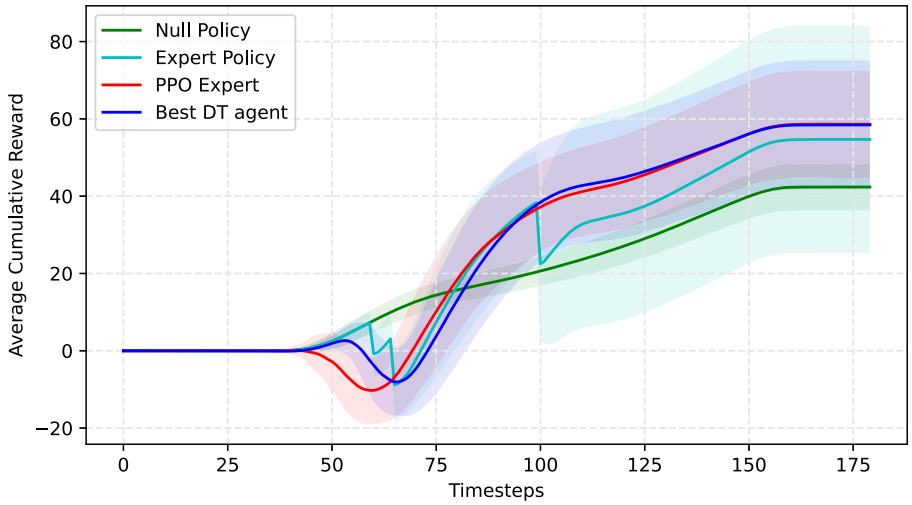


Fig. 14. Average cumulative reward over time of four different policies on the Maize Fertilization environment. This figure is based on a similar analysis done in the original `gym-DSSAT` paper [29]. Metrics were measured over 1,000 episodes.

and 7, respectively. Also shown in the figure is the *PPO Expert*, which refers to the model we have used so far as an expert; instead of adding fixed amounts on fixed days, it takes a subtler approach by adding gradual quantities over a large number of days, resulting in a smoother curve and a higher cumulative reward over time. Finally, the *Best DT Agent* has a curve whose smoothness closely mirrors the PPO's, although it is clear that this agent begins fertilization sooner. Despite this difference, the two end up with very similar final cumulative rewards, with the key distinction that the DT agent is interpretable, while the PPO is not.

4.3. Interpreting the models

With the goal of demonstrating the interpretability of the univariate tree approach to RL, this section provides interpretations for the best solutions obtained in each environment. These interpretations have been supported by a visualization tool (included in the previously-mentioned public repository) that displays the DT agent and the rendered environment side-by-side, allowing the user to see in real-time which nodes are activated at each time step (see Fig. 15). Editing the tree is also possible, paving the way for interactions typical of human-in-the-loop machine learning [86]. We argue that tools like this highlight the interpretability of DT models and their usefulness in the RL domain, allowing user insights that are impossible in uninterpretable models such as deep neural networks.

4.3.1. Cartpole

For the Cartpole task, we have chosen Tree B in Fig. 6 to interpret. Since the model is small, the interpretation is quite simple: if the angle of the pole is less than or equal to -0.015, the pole is tilted to the left, so the cart must move to the left to correct the pole. On the other hand, if the pole's angle is greater than -0.015, then it is either perfectly balanced or tilted to the right. In this case, the tree uses the angular velocity to decide what to do: if the pole is increasingly tilted to the left (Angular Velocity $\leq$ -0.30), then the cart must move to the left to correct it since just trusting the angle and moving to the right would increase the velocity even more and result in a difficult-to-recover scenario. Otherwise, the cart must move to the right. This behavior results in a left tilt bias, which can be seen in Fig. 7.

4.3.2. Mountain Car

For the Mountain Car task, we have chosen to interpret Tree A in Fig. 9. The first check made by the agent is to see if the car is moving to the right (Car Velocity $\leq$ 0.019): if it is, then it maintains this momentum by moving even further to the right, which means it either reaches the final flag or reaches a slope where it loses momentum and starts moving in the opposite direction. If this test fails, then the movement must be defined by whether the car is stationary or moving to the left, two situations distinguished by the following "Car Velocity $\leq$ -0.0" split. The car then moves to the left in two situations: when it is already moving to the left but has not yet reached position -0.94, or when it is stationary and its position is greater than -0.38. Fig. 4b shows where these two positions are. They both correspond to points on the slopes just *before* the car's momentum drops to zero, so by explicitly reversing direction in these two situations, the agent is able to start moving in the opposite direction before the slope forces it to do so, thus optimizing the time needed to build up enough momentum to reach the flag.

4.3.3. Lunar Lander

The Lunar Lander agent is the most difficult of the four to interpret, mainly due to its larger size. To do this, it is crucial to identify which subtrees are responsible for handling which situations, which can be done by visualizing the agent in action and observing the leaf activation patterns. Fig. 10 encapsulates this idea by both highlighting specific subtrees and annotating each leaf with its fraction of visits, indicating which are more frequently activated.

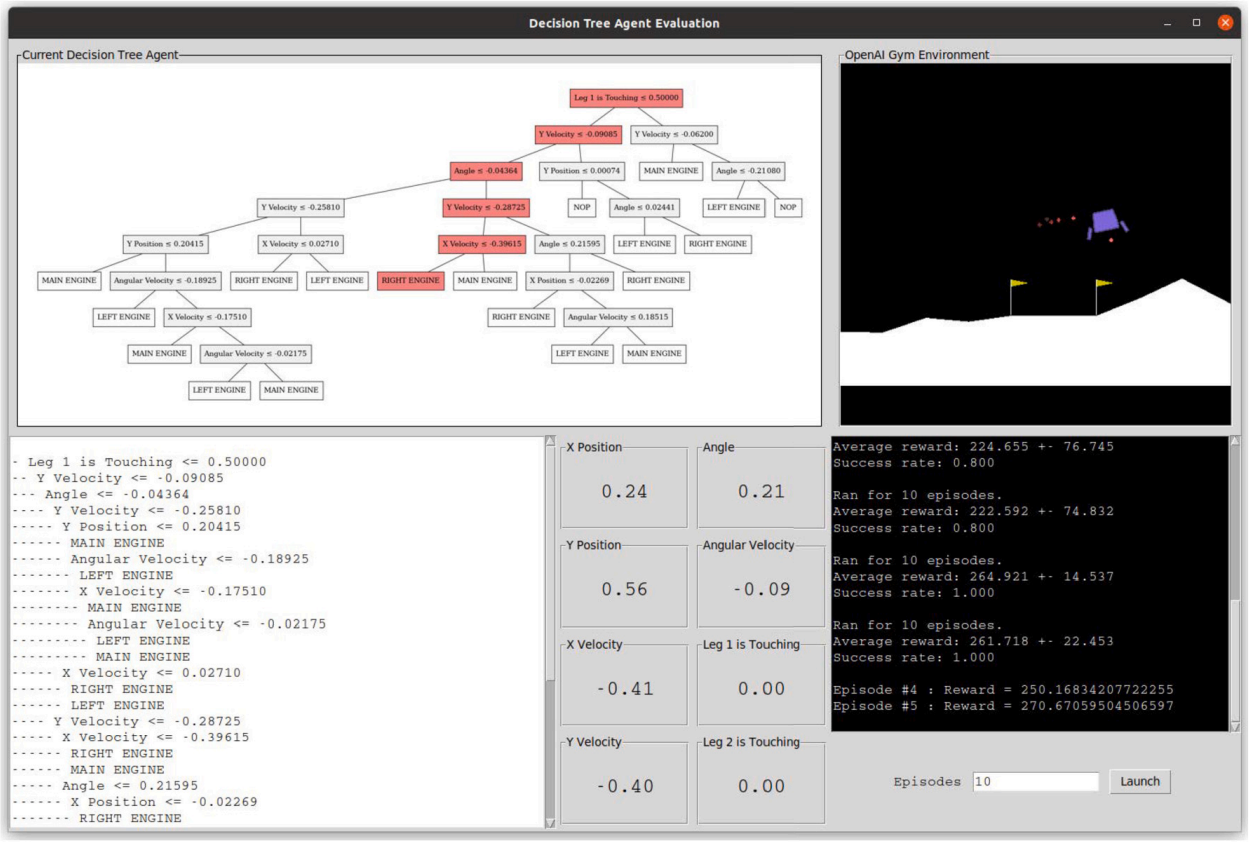


Fig. 15. Visualization tool that facilitates investigating the agent's reasoning and interacting with its rules.

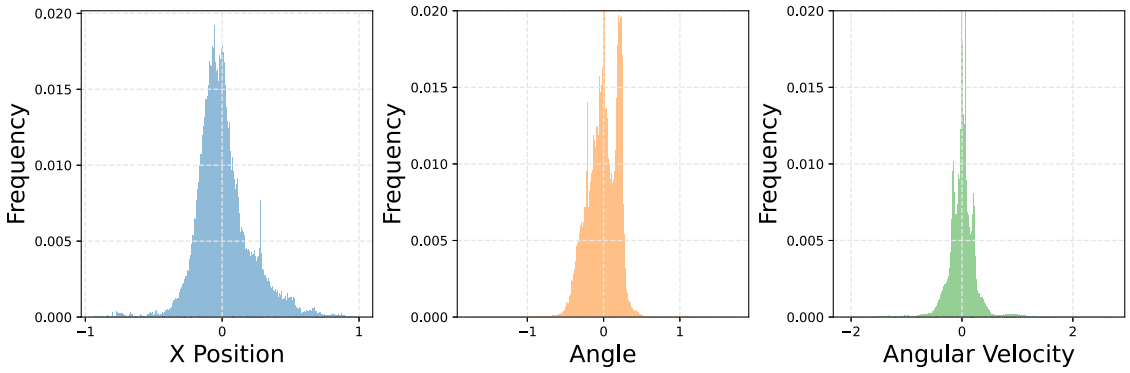


Fig. 16. Distribution of attributes from the states visited by the Lunar Lander agent in Fig. 10. Note that there is an asymmetry in the angles, as well as a more prevalent presence in the positive values of X than in the negative ones.

The root split in the tree checks whether Leg 1 touches the ground, leading to Subtree A if the result is positive. It then becomes clear that this subtree is responsible for the final landing procedures, activating the main engine if the agent is falling too fast, correcting with the left engine to land the other leg if the ship is tilted too much to the right, and otherwise doing nothing once the landing is complete. Because the task does not end immediately in this latter case, the agent spends some time doing nothing on the landing pad, resulting in the high frequency of the NOP leaf in this subtree.

Subtree B, on the other hand, is activated when the agent is not touching the ground but has a vertical velocity greater than -0.09: that is, it is falling slowly. As such, it handles a pre-landing phase in which the agent has started to decelerate but is not yet close enough to the ground for Subtree A to handle the situation. Quite simply, Subtree B either fires the left or right engines symmetrically based on the current angle of the ship, or it does nothing if the vertical position is less than 0.0 (i.e. the ship is extremely close to touching the ground, but has not yet done so).

Then, the remaining subtrees are divided into two groups based on the split Angle $\leq$ -0.04: the left group handles situations where the angle is mostly negative (i.e., right-tilt), and the right group handles situations where the angle is mostly positive (i.e., left-tilt). The latter responds for 42% of the actions taken by the agent, while the former responds for only 26%, suggesting a slight angle asymmetry during flight. This suspicion is further supported by Fig. 16, which shows the distributions of the states visited by

the agent – there is a slight tendency for positive angles, as well as a better distributed presence in the rightmost half of the scene. In this context, it is easier to interpret Subtrees C and D. Subtree D is activated when the vertical velocity is less than -0.28 (i.e. the agent is falling too fast). In this situation, the most sensible thing to do is to decelerate by firing the main engine as soon as possible – and indeed, the corresponding main engine leaf in Subtree D is one of the two most frequent nodes in the tree, serving as the main way to mitigate the fall. However, if there is no danger of free fall (Y Velocity > -0.28), then Subtree C corrects the angle and position: it fires the right engine to correct either a highly positive angle (Angle > 0.21) or an off-center position (X Position ≤ -0.02). Otherwise, this subtree corrects the natural tendency to tilt left by firing the left engine if the angular velocity allows it (Angular Velocity ≤ 0.185).

The last pair of subtrees is also responsible for deceleration, angle correction, and position correction, but in a slightly different way to account for the fact that it only handles negative angles. If the vertical speed is high enough (i.e., the agent is not falling too fast), Subtree E corrects the horizontal speed in a remarkably symmetrical way, firing the engine in the opposite direction to the current one. However, if the agent is in free fall, Subtree F is activated to slow down, just like Subtree D – although it performs this role in a slightly more sophisticated way. The subtree not only fires the main engine to reduce the rate of fall, but it can also fire the left engine in two situations: first, if the angular velocity is extremely skewed to the left (Angular Velocity ≤ -0.18), or if it is slightly skewed to the left and there is no danger of going too far away from the landing pad (Angular Velocity ≤ -0.02 and X Velocity > -0.17). In all other situations, the main engine is actually fired for deceleration. Empirically, firing the left engine in Subtree F usually results in the angle being tilted to the other side, thus ceding control to Subtrees C and D.

4.3.4. Maize Fertilization

Finally, we interpret the best DT produced for the Maize Fertilization environment (see Fig. 13). The model has three possible actions: add a very small amount of nitrogen to the soil (0.048), add a larger amount of nitrogen (13.94), or add close to no nitrogen (0.002). In deciding which of these amounts to add, the model only considers the `istage` attribute, which corresponds to the growing stage of the corn; more specifically, the model adds a very small amount of nitrogen when the plants are at the end of their juvenile stage (i.e., when `istage` = 1), a larger amount of nitrogen when 50% of them have completed their flowering initiation (i.e., when `istage` = 2), and almost no fertilizer in all other cases. This strategy bears some similarities to the expert policy described in Section 4.2.4 (which also adds fertilizer only during the first half of the crop's development), but both differ significantly in terms of nitrogen distribution: rather than following the expert policy's approach of adding large amounts of fertilizer over a few days, the DT policy opts to add smaller amounts of fertilizer interspersed throughout all of the crop's early growth stages. Significantly, no information other than the current growth stage, not even weather or soil data, is required to achieve this level of performance; this provides insight not only into the problem but also into the inner workings of the simulator itself. We emphasize that this kind of insight comes naturally when dealing with interpretable models such as DTs, while it is much more difficult or even impossible for uninterpretable models such as deep neural networks.

5. Conclusion

In this work, we proposed a novel algorithm called MENS-DT-RL to produce interpretable DTs for RL tasks. It works by extending the CRO-SL optimization ensemble algorithm with novel operators capable of handling DT structures, as well as with a fitness metric that prioritizes interpretable and high-performing models. We also proposed three different initializations for MENS-DT-RL, including the usage of IL techniques to provide good initial solutions, and a novel pruning approach called Reward Pruning that reduces RL trees while maintaining their performance.

The algorithm and its configurations have been evaluated on four environments: three benchmarks from the widely used OpenAI Gym taskset, and a crop fertilization environment, based on a real-world problem. The results showed that the proposed MENS-DT-RL algorithm was able to produce high-quality interpretable models for all environments, to the best of our knowledge outperforming the state-of-the-art of DTs for RL on three of them (Mountain Car, Lunar Lander and Maize Fertilization). In particular, the MENS-DT-RL with random initialization was able to achieve the best solution for Cartpole and Mountain Car (1.0 success rate with 5 and 7 nodes, respectively), while the proposed Reward Pruning approach was required to achieve the best solutions for Lunar Lander (0.9 success rate with 37 nodes) and Maize Fertilization (0.8 success rate with 5 nodes). The best models for each environment were discussed in detail and it was shown that they are indeed interpretable, indicating that the proposed approach is capable of achieving interpretability without losing performance. In addition, it was shown that the best solutions obtained by the algorithm sometimes differ in their approach to the problem, effectively empowering the user with the choice of which model to choose. Overall, the results show that MENS-DT-RL excels at tasks both artificial and real-world-based, handling simple and complex problems, either with discrete or continuous action spaces.

As future work, it would be interesting to extend the MENS-DT-RL to other types of RL problems, such as those with visual attributes as input (instead of tabular data), and with multiple agents (instead of a single one). In addition, it would be worthwhile to study Reward Pruning in more detail and to experiment with other designs, since this approach was crucial in solving the more complex Lunar Lander task. Finally, it might be interesting to extend the MENS-DT-RL to use other DT models, such as allowing splits that contain small and interpretable combinations of attributes (such as logical operators between boolean attributes).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Vinícius G. Costa reports financial support was provided by Coordination of Higher Education Personnel Improvement. Carlos E. Pedreira reports financial support was provided by National Council for Scientific and Technological Development. Carlos E. Pedreira reports financial support was provided by Carlos Chagas Filho Foundation for Research Support in the State of Rio de Janeiro. Sancho Salcedo-Sanz reports financial support was provided by Spain Ministry of Science and Innovation. Jorge Pérez-Aracil reports financial support was provided by Spain Ministry of Science and Innovation.

Data availability

The code is in a public repository shared in the paper.

Acknowledgements

This work was supported in part by the Brazilian research agencies CNPq — National Council for Scientific and Technological Development (Grant Number 306258/2019-6); FAPERJ—Foundation for Research Support of Rio de Janeiro State (Grant Number E-26/200.840/2021); Coordination of Higher Education Personnel Improvement - Finance Code 001; and by the Ministry of Science and Innovation (MICINN) through a National Project (PID2020-115454GB-C21).

Appendix A. Ablation study

In Section 3, we specified how to use the total rewards of each episode in order to calculate a tree agent's fitness (Equation (1)). This calculation involves three components: the *average reward*, the *standard deviation of the reward*, and the *tree size*. We argued that though the average reward is effectively what drives the success rate, including the standard deviation is key to producing trees with consistent behavior, while including tree size guides the process towards smaller, more interpretable trees. In this section, we provide an ablation study to justify the inclusion of these two components, showing that removing them from the equation results in worse solutions.

To achieve this goal, we modify the fitness calculation of the MENS-DT-RL to create four versions:

1. *Fitness A*: the full fitness, with standard deviation and tree size, as used in the paper:

$$fitness(M) = \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)} - \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(G_0^{(M,j)} - \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)}\right)^2} - \alpha||M|| \tag{A.1}$$

2. *Fitness B*: the full fitness, but without standard deviation:

$$fitness(M) = \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)} - \alpha||M|| \tag{A.2}$$

3. *Fitness C*: the full fitness, but without tree size:

$$fitness(M) = \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)} - \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(G_0^{(M,j)} - \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)}\right)^2} \tag{A.3}$$

4. *Fitness D*: using only the average reward as fitness:

$$fitness(M) = \frac{1}{N}\sum_{i=1}^{N} G_0^{(M,i)} \tag{A.4}$$

In order to compare these four models, we focus on the Mountain Car environment, which strikes a good balance between simplicity and complexity (not as simple as Cartpole, but complex enough for there to be a reasonable difference between algorithms). The three different initializations of the MENS-DT-RL are executed for 50 simulations each, with the same parameterization as in the main text. The results are displayed in Table A.6.

As the results show, varying the fitness function can greatly affect the final solution: the configurations with smaller tree size are usually "A" and "B" (where tree size is part of the fitness function), while the configurations with lower standard deviation are usually "A" and "C" (where standard deviation is taken into account). To better see this effect, it is useful to depict the results visually. Fig. A.17 shows the boxplots of all four configurations for MENS-DT-RL (R), where it can be seen that, at the extremes, fitness A leads to solutions with lower tree size and standard deviation, while fitness D goes in the opposite direction. Interestingly, this has no clear effect on success rate, since this metric has such a large variance for each configuration; this is typical of random initializations like that of MENS-DT-RL (R), which can sometimes jumpstart the model with a slightly decent solution, and other times leave it stranded with a range of 0.00 success rate solutions that need to be completely modified by the genetic operators. Therefore, for the MENS-DT-RL (R), fitness A and B are clearly better than C and D (since they have similar success rates and lower

Table A.6
Results for the ablation study done for the Mountain Car task. Fitnesses C and D do not include tree size, while fitnesses B and D do not include standard deviation of reward.

Initialization	Config.	Avg. Reward	Avg. Stdev. Reward	Average SR	Avg. #Nodes
MENS-DT-RL (R)	Fitness A	-112.98	2.08	0.28	7.44
	Fitness B	-108.77	5.61	0.41	9.00
	Fitness C	-114.82	2.37	0.25	25.20
	Fitness D	-107.37	6.30	0.45	23.16
MENS-DT-RL (IL)	Fitness A	-106.30	4.49	0.74	7.00
	Fitness B	-103.98	11.14	0.40	7.20
	Fitness C	-103.53	4.95	0.54	25.48
	Fitness D	-101.76	9.27	0.81	21.84
MENS-DT-RL (P)	Fitness A	-106.68	5.94	0.74	7.12
	Fitness B	-106.87	10.32	0.38	6.40
	Fitness C	-105.78	3.12	0.76	22.44
	Fitness D	-104.36	8.73	0.59	19.80

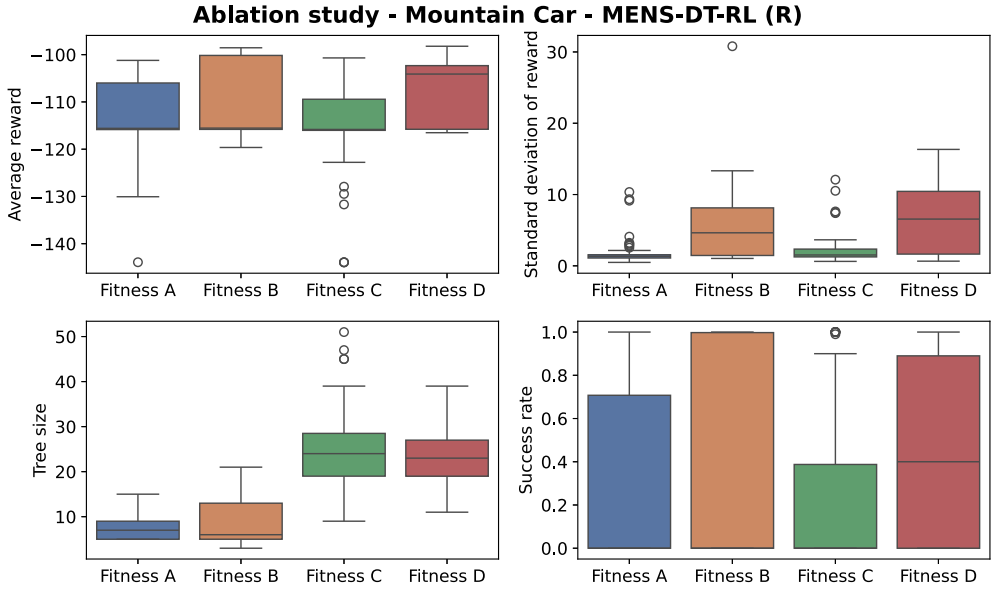


Fig. A.17. Ablation comparison between results produced by the MENS-DT-RL (R) when using different fitness functions on the Mountain Car task. Fitnesses C and D do not include tree size, while fitnesses B and D do not include standard deviation of reward.

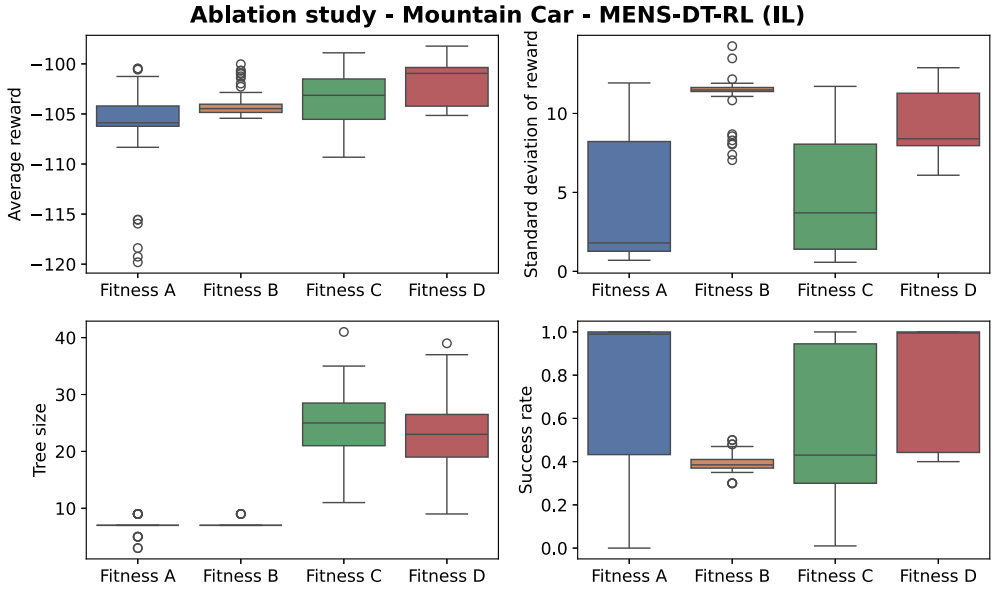


Fig. A.18. Ablation comparison between results produced by MENS-DT-RL (IL) when using different fitness functions on the Mountain Car task. Fitnesses C and D do not include tree size, while fitnesses B and D do not include standard deviation of reward.

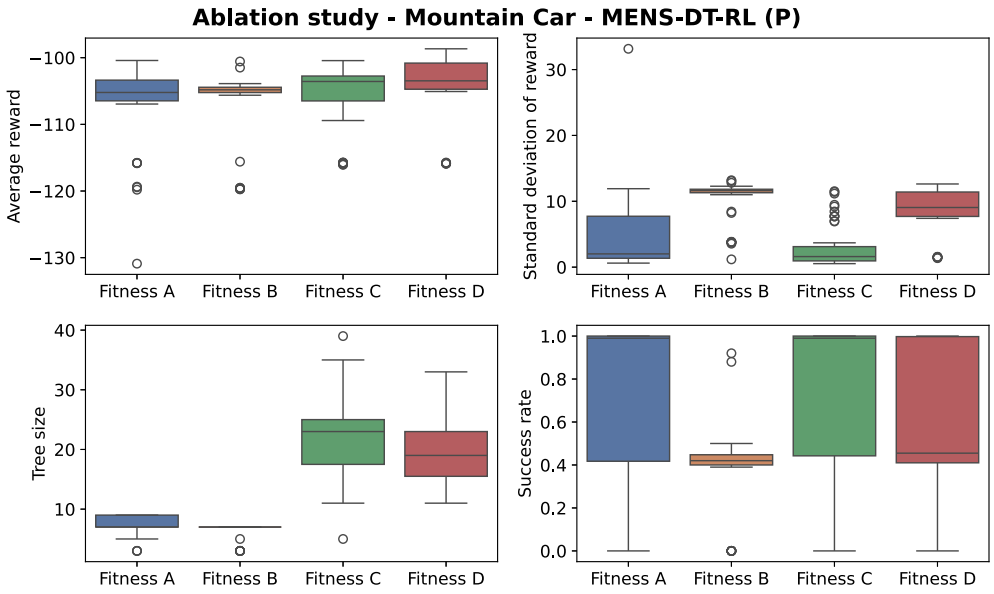


Fig. A.19. Ablation comparison between results produced by MENS-DT-RL (P) when using different fitness functions on the Mountain Car task. Fitnesses C and D do not include tree size, while fitnesses B and D do not include standard deviation of reward.

tree size), but A is not definitely better than B (A's standard deviation is lower, but this has no conclusive impact on success rate, given the high variance).

To see how including standard deviation drives success rate up, we must turn ourselves to MENS-DT-RL (IL), see Fig. A.18. As in the previous initialization, configurations "A" and "B" continue to stand out in terms of low tree size, while "A" and "C" do the same for standard deviation; however, there is also a marked difference in success rate that was absent in the previous initialization: now, fitness A has considerably higher success rate than fitness B, with little overlap between them. This serves as strong indication that including standard deviation in the fitness leads to models with higher success rates. The exact same behavior can be seen in Fig. A.19, which displays the boxplots for MENS-DT-RL (P). Overall, these results indicate that including both standard deviation and tree size in the fitness function leads to solutions with lower tree size and higher success rate.

References

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al., Solving Rubik's cube with a robot hand, arXiv preprint, arXiv:1910.07113, 2019.
[2] H. Mao, M. Alizadeh, I. Menache, S. Kandula, Resource management with deep reinforcement learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 50–56.
[3] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N.J. Yuan, X. Xie, Z. Li, DRN: a deep reinforcement learning framework for news recommendation, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 167–176.
[4] 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 529 (7587) (2016) 484–489.
[5] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nat. Mach. Intell. 1 (5) (2019) 206–215.
[6] G. Alain, Y. Bengio, Understanding intermediate layers using linear classifier probes, arXiv preprint, arXiv:1610.01644, 2016.
[7] E.M. Kenny, C. Ford, M. Quinn, M.T. Keane, Explaining black-box classifiers using post-hoc explanations-by-example: the effect of explanations and error-rates in XAI user studies, Artif. Intell. 294 (2021) 103459.
[8] C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, C. Zhong, Interpretable machine learning: fundamental principles and 10 grand challenges, Stat. Surv. 16 (2022) 1–85.
[9] C. Glanois, P. Weng, M. Zimmer, D. Li, J. Hao, T. Yang, W. Liu, A survey on interpretable reinforcement learning, IEEE Trans. Neural Netw. Learn. Syst. (2021).
[10] E. Puiutta, E. Veith, Explainable reinforcement learning: a survey, in: International Cross-Domain Conference for Machine Learning and Knowledge Extraction, Springer, 2020, pp. 77–95.
[11] C. Yu, J. Liu, S. Nemati, G. Yin, Reinforcement learning in healthcare: a survey, ACM Comput. Surv. 55 (1) (2021) 1–36.
[12] A. Coronato, M. Naeem, G. De Pietro, G. Paragliola, Reinforcement learning for intelligent healthcare applications: a survey, Artif. Intell. Med. 109 (2020) 101964.
[13] B.R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A.A. Al Sallab, S. Yogamani, P. Pérez, Deep reinforcement learning for autonomous driving: a survey, IEEE Trans. Intell. Transp. Syst. 23 (6) (2021) 4909–4926.
[14] P. Sequeira, M. Gervasio, Interestingness elements for explainable reinforcement learning: understanding agents' capabilities and limitations, Artif. Intell. 288 (2020) 103367.
[15] A. Alharin, T.-N. Doan, M. Sartipi, Reinforcement learning interpretation methods: a survey, IEEE Access 8 (2020) 171058–171077.
[16] V.G. Costa, C.E. Pedreira, Recent advances in decision trees: an updated survey, Artif. Intell. Rev. (2022) 1–36.
[17] O. Loyola-Gonzalez, Black-box vs. white-box: understanding their advantages and weaknesses from a practical point of view, IEEE Access 7 (2019) 154096–154113.
[18] X. Meng, P. Zhang, Y. Xu, H. Xie, Construction of decision tree based on C4.5 algorithm for online voltage stability assessment, Int. J. Electr. Power Energy Syst. 118 (2020) 105793.
[19] G. Ciravegna, P. Barbiero, F. Giannini, M. Gori, P. Lió, M. Maggini, S. Melacci, Logic explained networks, Artif. Intell. 314 (2023) 103822.

[20] A. Silva, M. Gombolay, T. Killian, I. Jimenez, S.-H. Son, Optimization methods for interpretable differentiable decision trees applied to reinforcement learning, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 1855–1865.
[21] A.M. Roth, N. Topin, P. Jamshidi, M. Veloso, Conservative Q-improvement: reinforcement learning for an interpretable decision-tree policy, arXiv preprint, arXiv:1907.01180, 2019.
[22] S. Salcedo-Sanz, C. Camacho-Gómez, D. Molina, F. Herrera, A coral reefs optimization algorithm with substrate layers and local search for large scale global optimization, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 3574–3581.
[23] S. Salcedo-Sanz, A review on the coral reefs optimization algorithm: new development lines and current applications, Prog. Artif. Intell. 6 (2017) 1–15.
[24] Y.-C. Wang, C.-W. Tsai, An efficient coral reef optimization with substrate layers for clustering problem on spark, in: 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2018, pp. 2814–2819.
[25] L. García-Hernández, J. Garcia-Hernandez, L. Salas-Morera, C. Carmona-Muñoz, N.S. Alghamdi, J.V. de Oliveira, S. Salcedo-Sanz, Addressing unequal area facility layout problems with the coral reef optimization algorithm with substrate layers, Eng. Appl. Artif. Intell. 93 (2020) 103697.
[26] C. Marcelino, J. Pérez-Aracil, E. Wanner, S. Jiménez-Fernández, G. Leite, S. Salcedo-Sanz, Cross-entropy boosted CRO-SL for optimal power flow in smart grids, Soft Comput. 27 (10) (2023) 6549–6572.
[27] E. Bermejo, M. Chica, S. Damas, S. Salcedo-Sanz, O. Cordón, Coral reef optimization with substrate layers for medical image registration, Swarm Evol. Comput. 42 (2018) 138–159.
[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI gym, arXiv preprint, arXiv:1606.01540, 2016.
[29] R. Gautron, E.J. Padrón, P. Preux, J. Bigot, O.-A. Maillard, D. Emukpere, gym-DSSAT: a crop model turned into a reinforcement learning environment, Ph.D. thesis, Inria Lille, 2022.
[30] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[31] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292.
[32] P.Y. Glorennec, L. Jouffe, Fuzzy q-learning, in: Proceedings of 6th International Fuzzy Systems Conference, vol. 2, IEEE, 1997, pp. 659–662.
[33] H. Hasselt, Double q-learning, Adv. Neural Inf. Process. Syst. 23 (2010).
[34] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[35] A. Hussein, M.M. Gaber, E. Elyan, C. Jayne, Imitation learning: a survey of learning methods, ACM Comput. Surv. 50 (2) (2017) 1–35.
[36] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., End to end learning for self-driving cars, arXiv preprint, arXiv:1604.07316, 2016.
[37] J. Merel, Y. Tassa, D. TB, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, N. Heess, Learning human behaviors from motion capture by adversarial imitation, arXiv preprint, arXiv:1707.02201, 2017.
[38] C. Finn, S. Levine, P. Abbeel, Guided cost learning: deep inverse optimal control via policy optimization, in: International Conference on Machine Learning, PMLR, 2016, pp. 49–58.
[39] D.A. Pomerleau, ALVINN: an autonomous land vehicle in a neural network, Adv. Neural Inf. Process. Syst. 1 (1988).
[40] S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.
[41] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms–a survey, Swarm Evol. Comput. 44 (2019) 695–711.
[42] J.A. Vrugt, B.A. Robinson, Improved evolutionary optimization from genetically adaptive multimethod search, Proc. Natl. Acad. Sci. 104 (3) (2007) 708–711.
[43] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2) (2008) 243–259.
[44] W.K. Mashwani, A. Salhi, Multiobjective evolutionary algorithm based on multimethod with dynamic resources allocation, Appl. Soft Comput. 39 (2016) 292–309.
[45] Y. Xue, S. Zhong, Y. Zhuang, B. Xu, An ensemble algorithm with self-adaptive learning techniques for high-dimensional numerical optimization, Appl. Math. Comput. 231 (2014) 329–346.
[46] X. Wang, C. Li, J. Zhu, Q. Meng, L-SHADE-E: ensemble of two differential evolution algorithms originating from L-SHADE, Inf. Sci. 552 (2021) 201–219.
[47] J. Yao, Z. Chen, Z. Liu, Improved ensemble of differential evolution variants, PLoS ONE 16 (8) (2021) e0256206.
[48] S. Salcedo-Sanz, R. García-Herrera, C. Camacho-Gómez, E. Alexandre, L. Carro-Calvo, F. Jaume-Santero, Near-optimal selection of representative measuring points for robust temperature field reconstruction with the CRO-SL and analogue methods, Glob. Planet. Change 178 (2019) 15–34.
[49] J. Pérez-Aracil, C. Camacho-Gómez, A.M. Hernández-Díaz, E. Pereira, D. Camacho, S. Salcedo-Sanz, Memetic coral reefs optimization algorithms for optimal geometrical design of submerged arches, Swarm Evol. Comput. 67 (2021) 100958.
[50] J. Pérez-Aracil, D. Casillas-Pérez, S. Jiménez-Fernández, L. Prieto-Godino, S. Salcedo-Sanz, A versatile multi-method ensemble for wind farm layout optimization, J. Wind Eng. Ind. Aerodyn. 225 (2022) 104991.
[51] J. Pérez-Aracil, C. Camacho-Gómez, E. Lorente-Ramos, C.M. Marina, L.M. Cornejo-Bueno, S. Salcedo-Sanz, New probabilistic, dynamic multi-method ensembles for optimization based on the CRO-SL, Mathematics 11 (7) (2023) 1666.
[52] D. Hein, S. Udluft, T.A. Runkler, Interpretable policies for reinforcement learning by genetic programming, Eng. Appl. Artif. Intell. 76 (2018) 158–169.
[53] H. Zhang, A. Zhou, X. Lin, Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis, Complex Intell. Syst. 6 (2020) 741–753.
[54] D. Trivedi, J. Zhang, S.-H. Sun, J.J. Lim, Learning to synthesize programs as interpretable and generalizable policies, Adv. Neural Inf. Process. Syst. 34 (2021) 25146–25163.
[55] A. Verma, V. Murali, R. Singh, P. Kohli, S. Chaudhuri, Programmatically interpretable reinforcement learning, in: International Conference on Machine Learning, PMLR, 2018, pp. 5045–5054.
[56] T. Silver, K.R. Allen, A.K. Lew, L.P. Kaelbling, J. Tenenbaum, Few-shot Bayesian imitation learning with logical program policies, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 10251–10258.
[57] E.H. Mamdani, Application of fuzzy algorithms for control of simple dynamic plant, in: Proceedings of the Institution of Electrical Engineers, vol. 121, IET, 1974, pp. 1585–1588.
[58] C.-F. Juang, J.-Y. Lin, C.-T. Lin, Genetic reinforcement learning through symbiotic evolution for fuzzy controller design, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 30 (2) (2000) 290–302.
[59] J. Huang, P.P. Angelov, C. Yin, Interpretable policies for reinforcement learning by empirical fuzzy sets, Eng. Appl. Artif. Intell. 91 (2020) 103559.
[60] D. Hein, A. Hentschel, T. Runkler, S. Udluft, Particle Swarm Optimization for generating interpretable fuzzy reinforcement learning policies, Eng. Appl. Artif. Intell. 65 (2017) 87–98.
[61] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Routledge, 2017.
[62] J.R. Quinlan, C4.5: Programs for Machine Learning, Elsevier, 2014.
[63] L.D. Pyeatt, A.E. Howe, et al., Decision tree function approximation in reinforcement learning, in: Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models, Cuba, vol. 2, 2001, pp. 70–77.
[64] A.K. McCallum, Reinforcement Learning with Selective Perception and Hidden State, University of Rochester, 1996.
[65] W.T. Uther, M.M. Veloso, Tree based discretization for continuous state space reinforcement learning, in: AAAI/IAAI, vol. 98, 1998, pp. 769–774.

[66] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, J. Mach. Learn. Res. 6 (2005).
[67] O. Bastani, Y. Pu, A. Solar-Lezama, Verifiable reinforcement learning via policy extraction, Adv. Neural Inf. Process. Syst. 31 (2018).
[68] Y. Coppens, K. Efthymiadis, T. Lenaerts, A. Nowé, T. Miller, R. Weber, D. Magazzeni, Distilling deep reinforcement learning policies in soft decision trees, in: Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence, 2019, pp. 1–6.
[69] N. Frosst, G. Hinton, Distilling a neural network into a soft decision tree, arXiv preprint, arXiv:1711.09784, 2017.
[70] H. Li, J. Song, M. Xue, H. Zhang, J. Ye, L. Cheng, M. Song, A survey of neural trees, arXiv preprint, arXiv:2209.03415, 2022.
[71] G. Liu, O. Schulte, W. Zhu, Q. Li, Toward interpretable deep reinforcement learning with linear model U-trees, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 414–429.
[72] A. Suárez, J.F. Lutsko, Globally optimal fuzzy decision trees for classification and regression, IEEE Trans. Pattern Anal. Mach. Intell. 21 (12) (1999) 1297–1311.
[73] L.L. Custode, G. Iacca, Evolutionary learning of interpretable decision trees, IEEE Access 11 (2023) 6169–6184.
[74] Y. Dhebar, K. Deb, S. Nageshrao, L. Zhu, D. Filev, Towards interpretable-AI policies induction using evolutionary nonlinear decision trees for discrete action systems, arXiv preprint, arXiv:2009.09521, 2020.
[75] R.A. Lopes, A. Freitas, R. Silva, F.G. Guimarães, Differential evolution and perceptron decision trees for classification tasks, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2012, pp. 550–557.
[76] R. Rivera-Lopez, J. Canul-Reich, A global search approach for inducing oblique decision trees using differential evolution, in: Canadian Conference on Artificial Intelligence, Springer, 2017, pp. 27–38.
[77] V.G. Costa, S. Salcedo-Sanz, C.E. Pedreira, Efficient evolution of decision trees via fully matrix-based fitness evaluation, Appl. Soft Comput. (2023) 111045.
[78] A.A. Freitas, Comprehensible classification models: a position paper, ACM SIGKDD Explor. Newsl. 15 (1) (2014) 1–10.
[79] B. Kazimipour, X. Li, A.K. Qin, A review of population initialization techniques for evolutionary algorithms, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2585–2592.
[80] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, Stable-Baselines3: reliable reinforcement learning implementations, J. Mach. Learn. Res. 22 (1) (2021) 12348–12355.
[81] A.W. Moore, Efficient memory-based learning for robot control, Tech. Rep., University of Cambridge, 1990.
[82] J.W. Jones, G. Hoogenboom, C.H. Porter, K.J. Boote, W.D. Batchelor, L. Hunt, P.W. Wilkens, U. Singh, A.J. Gijsman, J.T. Ritchie, The DSSAT cropping system model, Eur. J. Agron. 18 (3–4) (2003) 235–265.
[83] G. Hoogenboom, C.H. Porter, K.J. Boote, V. Shelia, P.W. Wilkens, U. Singh, J.W. White, S. Asseng, J.I. Lizaso, L.P. Moreno, et al., The DSSAT crop modeling ecosystem, in: Advances in Crop Modelling for a Sustainable Agriculture, Burleigh Dodds Science Publishing, 2019, pp. 173–216.
[84] L.L. Custode, G. Iacca, Evolutionary learning of interpretable decision trees, arXiv preprint, arXiv:2012.07723, 2020.
[85] L. Hunt, K. Boote, Data for model operation, calibration, and evaluation, in: Understanding Options for Agricultural Production, 1998, pp. 9–39.
[86] E. Mosqueira-Rey, E. Hernández-Pereira, D. Alonso-Ríos, J. Bobes-Bascarán, Á. Fernández-Leal, Human-in-the-loop machine learning: a state of the art, Artif. Intell. Rev. 56 (4) (2023) 3005–3054.

Chapter 5

Conclusions

The world is currently experiencing a boom in Artificial Intelligence (AI) research, spearheaded by the Deep Learning advancements seen in the last decade. However, most of this attention has been devoted to "black-box" models, which cannot be directly understood by humans and that therefore have limited applicability on certain domains. Because of this, a growing body of research has been dedicated to explore "white-boxes" – models that are inherently interpretable –, in a movement that has been called *Interpretable AI*.

Interpretable models bring with them several advantages, such as being generally easier to understand, explain, and troubleshoot than their uninterpretable counterparts. However, these systems also have a significant drawback: it is often hard to create a model that is both high-performing and interpretable, such that sometimes it is not clear at all if there even exists a model that can fulfill both of these criteria. This challenge has been tackled in many ways as researchers turn their attentions to this field, but there are certain areas, such as Reinforcement Learning (RL), that still have a lot left to explore.

In this thesis, we tackled the goal of achieving Interpretable RL by using the "white-box" models known as Decision Trees (DTs). While aiming for this goal, several novel results were attained in Interpretable AI as a whole. First, we conducted an extensive review of recent advancements in the DT literature, positioning these contributions relative to well-established developments of the field. In doing so, the evolutionary approach was identified as the most promising way to create trees for RL, which led us to delve deeper into an exploration of these models.

Second, we proposed a way to alleviate the high computational cost of evolving DTs for supervised learning, specifically by using a novel matrix-based encoding that allows one to train classification trees through a series of matrix operations. The proposed encoding leverages the efficiency of numerical computation and resulted in speed-ups of up to 20 times, relative to the traditional programmatic approach that is normally used. To test the effects of these speedups, we also proposed a

novel evolutionary algorithm for supervised learning that employs this encoding, and was demonstrated to outperform other modern DT algorithms across a series of well-established classification benchmarks. The algorithm was called CRO-DT since it is built upon the bio-inspired Coral Reef Optimization (CRO) meta-heuristic, which has a successful history of applications in parameter optimization for machine learning applications.

Finally, we applied these previous lessons to the RL field, and proposed an algorithm called MENS-DT-RL that mixes Imitation Learning with the evolutionary approach to create trees for RL that are both interpretable and high-performing. More specifically, this algorithm was able to solve the well-known Mountain Car benchmark with almost half the number of nodes of the previous best tree-based solutions on the task, and was also the first to solve the complex Lunar Lander benchmark with both high interpretability and consistency. On a crop management simulation based on the real-world problem of growing maize, MENS-DT-RL was able to perfectly match the performance of deep RL models with only 5 nodes. These results demonstrate not only the validity of our proposed method, but also of tree-based solutions in general for interpretable RL tasks of various different kinds.

5.1 Limitations and future work

Although MENS-DT-RL has obtained several key results in popular RL benchmarks, there are limitations to this approach that would be interesting to address in future works. Those are the topic of this final section.

The first limitation concerns the notion of *robustness*: in essence, users expect that similar inputs will produce similar outputs, in such a way that interpretability is compromised when this is not the case [59]. Given this definition of robustness, it can be argued that MENS-DT-RL is not robust, since due to its stochastic nature, it may produce two entirely different solutions given the exact same environment and parameters, provided that the random seed is different (as seen in Cartpole's results in Chapter 4). However, this lack of robustness can also be seen as a positive, since it produces diverse solutions that the end-user can choose from, empowering them to select the one that best suits their goals. Furthermore, it's worthwhile to note that even though the evolutionary algorithm as a whole may present this lack of robustness, each tree produced by the algorithm is as robust as any traditional DT, given that their transparent inner structure does not change between predictions. In light of this discussion, it would be interesting to quantify the diversity and robustness of MENS-DT-RL in future works, connecting them to the parameters employed by the evolutionary algorithm.

A more crucial limitation is the *state representation*. In its current form, MENS-

DT-RL assumes low-dimensional and structured state attributes, such as the "Pole Angle" and "Pole Angular Velocity" of Cartpole; however, there are several RL tasks where the state is high-dimensional and less structured, most notably those tasks that depend on raw visual input (e.g. autonomous driving which largely employs cameras). These are tasks where black-box DNNs have historically excelled (since they can transform the raw visual input into relevant features through the training and application of their hidden layers), but it would be entirely possible to carry out this feature engineering more explicitly as well, in conjunction with more interpretable models. In this sense, it would be interesting to couple MENS-DT-RL with computer vision algorithms for object detection and segmentation (such as IODINE [60]), and evaluate its performance over tasks with raw visual data.

A similar limitation is the use of *univariate nodes*: since all inner nodes contain only a single attribute, MENS-DT-RL may produce solutions that are unnecessarily large (or "verbose"). A clear example can be found in the Lunar Lander environment, where there are two symmetric boolean attributes: "Is Leg 1 touching the ground?" and "Is Leg 2 touching the ground?". Though the attributes are different, it is reasonable to assume that an optimal agent would carry out the same behavior independent of which leg is touching the ground (e.g. turn off all engines to allow the ship to land). However, since there is no way to create an inner node with the logical expression "Is Leg 1 OR Leg 2 touching the ground?", the only way to achieve this would be to use two different inner nodes and duplicate the entire behavior subtree, which would increase solution size (possibly by a large margin). It would be interesting to extend MENS-DT-RL with the capacity to evolve multivariate features such as this and compare the results.

Another way to tackle this last point would be to look through the lens of *modularity*. In Chapter 4, certain parts of the best Lunar Lander tree were interpreted as "modules" (the landing module, the deceleration module, etc), and could therefore be abbreviated during visualization. Some questions arise: given a particular tree agent, is there a way to detect these modules automatically, instead of having to label them manually? If so, could these modules be taken into account by the evolutionary process, and possibly repeated throughout the tree? This notion might allow trees to grow larger without incurring a high interpretability cost, and would perhaps allow trees to succeed in even more complex RL environments.

A final limitation concerns the *interpretability metric* used. Throughout our work, we refer to tree size (i.e. number of tree nodes) as a measure of the solution interpretability, such that smaller trees are always deemed more interpretable than their larger counterparts. However, this is an over-simplification: as discussed by [61] and mentioned in Chapter 2, users may find larger trees to be *more* interpretable if their splits are a better representation of the user's domain knowledge. In this

sense, it would be interesting to experiment with other proxies for interpretability, such as the "average root-to-leaf path length" discussed by [62]: according to this notion, users do not interpret *models* but *predictions*, so that a very large tree can still be highly interpretable if its most frequently visited leaves are close to the roots. Trees evolved by MENS-DT-RL to optimize this metric would look very different from the ones obtained in our research, and comparing them would be important to understand how to better balance performance and interpretability.

Finally, we highlight that Reward Pruning merits more in-depth study. It would be interesting to evaluate the impact of the number of rounds, tree traversal order, and other parameters on the algorithm's results, especially since it was one of the key components behind MENS-DT-RL's successful and competitive results.

References

[1] RUDIN, C. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead", *Nature Machine Intelligence*, v. 1, n. 5, pp. 206–215, maio 2019. ISSN: 2522-5839. doi: 10.1038/s42256-019-0048-x.

[2] HARARI, Y. N. "Reboot for the AI revolution", *Nature*, v. 550, n. 7676, pp. 324–327, 2017.

[3] MAKRIDAKIS, S. "The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms", *Futures*, v. 90, pp. 46–60, 2017.

[4] HAENLEIN, M., KAPLAN, A. "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence", *California management review*, v. 61, n. 4, pp. 5–14, 2019.

[5] RAMESH, A., PAVLOV, M., GOH, G., et al. "Zero-shot text-to-image generation". In: *International Conference on Machine Learning*, pp. 8821–8831. PMLR, 2021.

[6] SAHARIA, C., CHAN, W., SAXENA, S., et al. "Photorealistic text-to-image diffusion models with deep language understanding", *Advances in Neural Information Processing Systems*, v. 35, pp. 36479–36494, 2022.

[7] BROWN, T., MANN, B., RYDER, N., et al. "Language models are few-shot learners", *Advances in neural information processing systems*, v. 33, pp. 1877–1901, 2020.

[8] OUYANG, L., WU, J., JIANG, X., et al. "Training language models to follow instructions with human feedback", *Advances in Neural Information Processing Systems*, v. 35, pp. 27730–27744, 2022.

[9] REN, Y., RUAN, Y., TAN, X., et al. "Fastspeech: Fast, robust and controllable text to speech", *Advances in neural information processing systems*, v. 32, 2019.

[10] RADFORD, A., KIM, J. W., XU, T., et al. "Robust speech recognition via large-scale weak supervision". In: *International Conference on Machine Learning*, pp. 28492–28518. PMLR, 2023.

[11] MNIH, V., KAVUKCUOGLU, K., SILVER, D., et al. "Playing atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[12] SILVER, D., HUBERT, T., SCHRITTWIESER, J., et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm", *arXiv preprint arXiv:1712.01815*, 2017.

[13] SCHRITTWIESER, J., ANTONOGLOU, I., HUBERT, T., et al. "Mastering atari, go, chess and shogi by planning with a learned model", *Nature*, v. 588, n. 7839, pp. 604–609, 2020.

[14] BADUE, C., GUIDOLINI, R., CARNEIRO, R. V., et al. "Self-driving cars: A survey", *Expert Systems with Applications*, v. 165, pp. 113816, 2021.

[15] GUPTA, A., ANPALAGAN, A., GUAN, L., et al. "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues", *Array*, v. 10, pp. 100057, 2021.

[16] LECUN, Y., BENGIO, Y., HINTON, G. "Deep learning", *nature*, v. 521, n. 7553, pp. 436–444, 2015.

[17] EUROPEAN PARLIAMENT, COUNCIL OF THE EUROPEAN UNION. "Regulation (EU) 2016/679 of the European Parliament and of the Council". Disponível em: <`https://data.europa.eu/eli/reg/2016/679/oj`>.

[18] WACHTER, S., MITTELSTADT, B., FLORIDI, L. "Why a right to explanation of automated decision-making does not exist in the general data protection regulation", *International Data Privacy Law*, v. 7, n. 2, pp. 76–99, 2017.

[19] SELBST, A., POWLES, J. ""Meaningful information" and the right to explanation". In: *conference on fairness, accountability and transparency*, pp. 48–48. PMLR, 2018.

[20] RAI, A. "Explainable AI: From black box to glass box", *Journal of the Academy of Marketing Science*, v. 48, pp. 137–141, 2020.

[21] PUIUTTA, E., VEITH, E. "Explainable reinforcement learning: A survey". In: *International cross-domain conference for machine learning and knowledge extraction*, pp. 77–95. Springer, 2020.

[22] RIBEIRO, M. T., SINGH, S., GUESTRIN, C. "" Why should i trust you?" Explaining the predictions of any classifier". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[23] WACHTER, S., MITTELSTADT, B., RUSSELL, C. "Counterfactual explanations without opening the black box: Automated decisions and the GDPR", *Harv. JL & Tech.*, v. 31, pp. 841, 2017.

[24] SIMONYAN, K., VEDALDI, A., ZISSERMAN, A. "Deep inside convolutional networks: Visualising image classification models and saliency maps", *arXiv preprint arXiv:1312.6034*, 2013.

[25] HOLTE, R. C. "Very simple classification rules perform well on most commonly used datasets", *Machine learning*, v. 11, pp. 63–90, 1993.

[26] APTÉ, C., DAMERAU, F., WEISS, S. M. "Automated learning of decision rules for text categorization", *ACM Transactions on Information Systems (TOIS)*, v. 12, n. 3, pp. 233–251, 1994.

[27] MURTHY, S. K. "Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey", v. 2, n. 4, pp. 345–389, 1998. doi: 10.1023/a:1009744630224.

[28] LOH, W.-Y. "Fifty Years of Classification and Regression Trees", *International Statistical Review*, v. 82, n. 3, pp. 329–348, 2014. ISSN: 1751-5823. doi: 10.1111/insr.12016.

[29] LIPTON, Z. C. "The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, v. 16, n. 3, pp. 31–57, 2018.

[30] AÏVODJI, U., ARAI, H., FORTINEAU, O., et al. "Fairwashing: the risk of rationalization". In: *International Conference on Machine Learning*, pp. 161–170. PMLR, 2019.

[31] RUDIN, C., CHEN, C., CHEN, Z., et al. "Interpretable machine learning: Fundamental principles and 10 grand challenges", *Statistics Surveys*, v. 16, pp. 1–85, 2022.

[32] GLANOIS, C., WENG, P., ZIMMER, M., et al. "A Survey on Interpretable Reinforcement Learning", *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[33] SEQUEIRA, P., GERVASIO, M. "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations", *Artificial Intelligence*, v. 288, pp. 103367, 2020.

[34] ALHARIN, A., DOAN, T.-N., SARTIPI, M. "Reinforcement learning interpretation methods: A survey", *IEEE Access*, v. 8, pp. 171058–171077, 2020.

[35] AKKAYA, I., ANDRYCHOWICZ, M., CHOCIEJ, M., et al. "Solving rubik's cube with a robot hand", *arXiv preprint arXiv:1910.07113*, 2019.

[36] MAO, H., ALIZADEH, M., MENACHE, I., et al. "Resource management with deep reinforcement learning". In: *Proceedings of the 15th ACM workshop on hot topics in networks*, pp. 50–56, 2016.

[37] ZHENG, G., ZHANG, F., ZHENG, Z., et al. "DRN: A deep reinforcement learning framework for news recommendation". In: *Proceedings of the 2018 world wide web conference*, pp. 167–176, 2018.

[38] YU, C., LIU, J., NEMATI, S., et al. "Reinforcement learning in healthcare: A survey", *ACM Computing Surveys (CSUR)*, v. 55, n. 1, pp. 1–36, 2021.

[39] CORONATO, A., NAEEM, M., DE PIETRO, G., et al. "Reinforcement learning for intelligent healthcare applications: A survey", *Artificial Intelligence in Medicine*, v. 109, pp. 101964, 2020.

[40] KIRAN, B. R., SOBH, I., TALPAERT, V., et al. "Deep reinforcement learning for autonomous driving: A survey", *IEEE Transactions on Intelligent Transportation Systems*, v. 23, n. 6, pp. 4909–4926, 2021.

[41] BREIMAN, L., FRIEDMAN, J., STONE, C. J., et al. *Classification and Regression Trees*. Taylor & Francis, jan. 1984. ISBN: 978-0-412-04841-8.

[42] QUINLAN, J. R. "Learning with continuous classes". In: *5th Australian joint conference on artificial intelligence*, v. 92, pp. 343–348. World Scientific, 1992.

[43] PYEATT, L. D., HOWE, A. E., OTHERS. "Decision tree function approximation in reinforcement learning". In: *Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models*, v. 2, pp. 70–77. Cuba, 2001.

[44] MCCALLUM, A. K. *Reinforcement learning with selective perception and hidden state*. University of Rochester, 1996.

[45] UTHER, W. T., VELOSO, M. M. "Tree based discretization for continuous state space reinforcement learning", *Aaai/iaai*, v. 98, pp. 769–774, 1998.

[46] ERNST, D., GEURTS, P., WEHENKEL, L. "Tree-based batch mode reinforcement learning", *Journal of Machine Learning Research*, v. 6, 2005.

[47] LI, H., SONG, J., XUE, M., et al. "A Survey of Neural Trees", *arXiv preprint arXiv:2209.03415*, 2022.

[48] LIU, G., SCHULTE, O., ZHU, W., et al. "Toward interpretable deep reinforcement learning with linear model u-trees". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 414–429. Springer, 2019.

[49] SILVA, A., GOMBOLAY, M., KILLIAN, T., et al. "Optimization Methods for Interpretable Differentiable Decision Trees Applied to Reinforcement Learning". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 1855–1865. PMLR, jun. 2020. ISSN: 2640-3498.

[50] SILVA, A., GOMBOLAY, M., KILLIAN, T., et al. "Optimization methods for interpretable differentiable decision trees applied to reinforcement learning". In: *International conference on artificial intelligence and statistics*, pp. 1855–1865. PMLR, 2020.

[51] EIBEN, A. E., SMITH, J. E. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2015. ISBN: 3662448734. doi: 10.1007/978-3-662-44874-8.

[52] DHEBAR, Y., DEB, K., NAGESHRAO, S., et al. "Towards Interpretable-AI Policies Induction using Evolutionary Nonlinear Decision Trees for Discrete Action Systems", *arXiv preprint arXiv:2009.09521*, 2020.

[53] CUSTODE, L. L., IACCA, G. "Evolutionary learning of interpretable decision trees", *IEEE Access*, 2023.

[54] BARROS, R. C., CARVALHO, A. C. P. L. F. D., FREITAS, A. A. *Automatic Design of Decision-Tree Induction Algorithms*. SpringerBriefs in Computer Science. Springer International Publishing, 2015. ISBN: 978-3-319-14230-2. doi: 10.1007/978-3-319-14231-9.

[55] SALCEDO-SANZ, S., DEL SER, J., LANDA-TORRES, I., et al. "The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems", *The Scientific World Journal*, v. 2014, 2014.

[56] PÉREZ-ARACIL, J., CAMACHO-GÓMEZ, C., LORENTE-RAMOS, E., et al. "New Probabilistic, Dynamic Multi-Method Ensembles for Optimization Based on the CRO-SL", *Mathematics*, v. 11, n. 7, pp. 1666, 2023.

[57] BÄCK, T., FOGEL, D. B., MICHALEWICZ, Z. "Handbook of evolutionary computation", *Release*, v. 97, n. 1, pp. B1, 1997.

[58] MITCHELL, M., TAYLOR, C. E. "Evolutionary computation: an overview", *Annual Review of Ecology and Systematics*, pp. 593–616, 1999.

[59] ALVAREZ-MELIS, D., JAAKKOLA, T. S. "On the Robustness of Interpretability Methods", *arXiv:1806.08049 [cs, stat]*, jun. 2018.

[60] GREFF, K., KAUFMAN, R. L., KABRA, R., et al. "Multi-object representation learning with iterative variational inference". In: *International conference on machine learning*, pp. 2424–2433. PMLR, 2019.

[61] FREITAS, A. A. "Comprehensible classification models: a position paper", *ACM SIGKDD explorations newsletter*, v. 15, n. 1, pp. 1–10, 2014.

[62] PILTAVER, R., LUŠTREK, M., GAMS, M., et al. "What makes classification trees comprehensible?" *Expert Systems with Applications*, v. 62, pp. 333–346, nov. 2016. ISSN: 09574174. doi: 10.1016/j.eswa.2016.06.009.

[63] ROSS, S., GORDON, G., BAGNELL, D. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

[64] HUSSEIN, A., GABER, M. M., ELYAN, E., et al. "Imitation learning: A survey of learning methods", *ACM Computing Surveys (CSUR)*, v. 50, n. 2, pp. 1–35, 2017.

[65] BOJARSKI, M., DEL TESTA, D., DWORAKOWSKI, D., et al. "End to end learning for self-driving cars", *arXiv preprint arXiv:1604.07316*, 2016.

[66] MEREL, J., TASSA, Y., TB, D., et al. "Learning human behaviors from motion capture by adversarial imitation", *arXiv preprint arXiv:1707.02201*, 2017.

[67] POMERLEAU, D. A. "Alvinn: An autonomous land vehicle in a neural network", *Advances in neural information processing systems*, v. 1, 1988.

[68] SUTTON, R. S., BARTO, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Appendix A

Imitation Learning and DAgger

This Appendix explains in more detail the Imitation Learning paradigm and the DAgger algorithm [63], which are key components behind the success of MENS-DT-RL (presented in Chapter 4).

Imitation Learning (IL) [64] is a branch of Reinforcement Learning (RL) where the objective is not to learn behaviors from scratch (by interacting with the environment, failing, and learning from such failures – as in traditional RL), but rather to replicate the behaviors of a reference agent, known as the *expert*. IL techniques are particularly valuable in scenarios such as autonomous driving [65] and realistic locomotion [66], where demonstrating the desired behavior is more practical than defining it through a reward function. While in the literature the expert is more often a human demonstrator, most IL methods are equally applicable when the expert is another machine agent.

One of the simplest forms of IL is *Behavioral Cloning* [67]. In this approach, one lets the expert interact with the environment for a certain number of episodes, and collects every pair of (state visited by the expert, action taken by the expert), resulting in a dataset $\mathcal{D} = \{(s, a)\}$. Then, one applies a supervised learning model to $\mathcal{D}$, creating a model that can map states to actions and which can then be used just like a traditional RL agent produced via more classic algorithms, such as Q-learning or value iteration [68].

Although simple and intuitive, Behavioral Cloning faces the key problem known as *covariate shift*. In essence, the imitator trained on the demonstration dataset $\mathcal{D}$ will be an inherently imperfect copy of the expert, and therefore will eventually take imperfect actions (i.e. actions that the expert would not take in that same scenario). These imperfect actions will eventually lead the imitator to states that are of lower value, and that therefore the expert itself would never have visited in the first place. Since the expert did not visit these states, they are absent from the training dataset $\mathcal{D}$, so the imitator has no information on how the expert would recuperate from this situation. This leads the imitator further and further away from

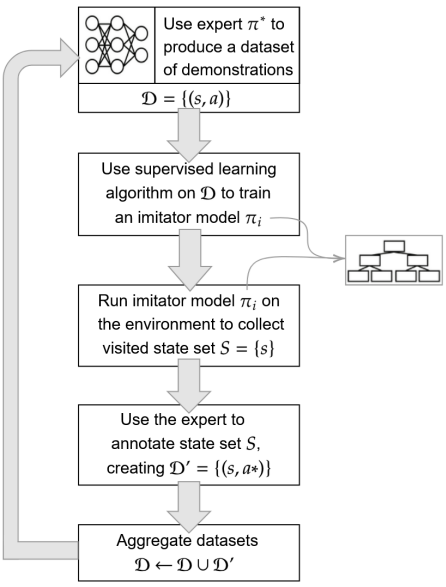


Figure A.1: An illustration of the DAgger algorithm, with a Neural Network as the expert and a Decision Tree as the imitator.

the expert's trajectory and towards states of increasingly lower quality, resulting in a poor performance that is a far cry from the expert's. Hence, Behavioral Cloning often produces agents that are very brittle.

One algorithm proposed to solve this issue is *DAgger*. DAgger (Dataset Aggregation, [63]) handles covariate shift by expanding the training dataset with new states encountered by the imitator during execution. At first, DAgger runs exactly the same as Behavioral Cloning, by observing the states visited by an expert, collecting a dataset $\mathcal{D}$, and training an imitator agent on $\mathcal{D}$ using supervised learning algorithms. However, instead of stopping there, DAgger runs the imitator on the environment and collects the states visited by said imitator; then, it queries the expert for the actions that it would take on each of these states, generating a series of pairs of (state visited by the imitator, action that the expert would take on such a state), which compose the new dataset $\mathcal{D}'$. Finally, $\mathcal{D}'$ is aggregated into $\mathcal{D}$, which is then used to train a new imitator, which is then used to collect a new $\mathcal{D}'$, and so on and so forth. A visual representation of this iterative process is shown in Figure A.1.

By using this approach, at each iteration the algorithm produces imitators that have increasingly more data on how the expert acts. Crucially, this expanded data is much more diverse and actually contains information on how to recuperate from a series of low-value states – seeing that the states are provided not only by the expert but also by the imperfect imitators themselves. This leads DAgger to strongly outperform Behavioral Cloning. Such an approach does, however, come with the drawback of needing continuous access to the expert, which might be impractical or expensive in various contexts, especially where the expert is human.