Transparent AI by Design

Search Algorithms for Supervised Learning, Control Policies, and Combinatorial Certification

Demirović, Emir

DOI
10.3233/FAIA250778

Publication date
2025

Document Version
Final published version

Published in
ECAI 2025 - 28th European Conference on Artificial Intelligence, including 14th Conference on Prestigious Applications of Intelligent Systems, PAIS 2025 - Proceedings

Citation (APA)
Demirović, E. (2025). Transparent AI by Design: Search Algorithms for Supervised Learning, Control Policies, and Combinatorial Certification. In I. Lynce, N. Murano, M. Vallati, S. Villata, F. Chesani, M. Milano, A. Omicini, & M. Dastani (Eds.), *ECAI 2025 - 28th European Conference on Artificial Intelligence, including 14th Conference on Prestigious Applications of Intelligent Systems, PAIS 2025 - Proceedings* (pp. 2-9). (Frontiers in Artificial Intelligence and Applications; Vol. 413). IOS Press. https://doi.org/10.3233/FAIA250778

ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA250778

Transparent AI by Design: Search Algorithms for Supervised Learning, Control Policies, and Combinatorial Certification

Emir Demirović

Delft University of Technology, The Netherlands

Abstract. AI methods—such as those used in supervised learning, controller synthesis, and combinatorial optimisation—have demonstrated immense value across many domains. However, their practical adoption is hindered by reliability concerns, particularly when these systems are designed as black boxes. Two key challenges arise for black-box AI: (1) lack of performance guarantees—when AI fails, it is unclear whether the task is infeasible or the underlying algorithm is simply inadequate; and (2) lack of confidence—results may be difficult to interpret or trust. While post-hoc interpretability techniques offer partial remedies, we advocate for a different paradigm: building AI systems that are transparent by design. Rather than explaining opaque decisions after the fact, we synthesise outputs that are intrinsically understandable and verifiable. This shifts the focus from doubting AI to questioning whether we are solving the right problem. We apply this approach across three distinct domains: supervised learning, controller synthesis, and infeasibility certification for combinatorial optimisation problems. Although these tasks involve exponentially large search spaces, recent advances demonstrate that designing for transparency is increasingly practical—often without sacrificing performance—making it a compelling alternative to opaque AI systems.

1 Introduction

AI is widely applied across a broad range of domains, including scheduling, hardware verification, healthcare, criminal justice, and protein engineering. The societal impact of AI is profound.

Despite the success of AI, there is a natural scepticism towards its practical use due to the sheer complexity of the underlying algorithms and the problems being addressed. Across many domains, two common concerns arise from a lack of:

1. *Performance guarantees*: AI may fail to achieve a desirable outcome, even though the desirable outcome is theoretically possible.
   - Consider training a machine learning model that reflects historical data while addressing potential discrimination against marginalised groups. When AI produces a model that falls short of our expectations—such as an unsatisfactory balance between faithfulness to historical data and fairness—it can be unclear whether a better model truly does not exist or if current techniques are simply unable to discover it.
2. *Confidence*: We may doubt whether AI outputs are sensible.
   - A complex policy that is difficult to interpret can be risky to deploy, even if it performs well in the training environment. The policy may rely on spurious correlations in the training data, leading to correct outcomes for the wrong reasons.
   - AI may claim that meeting the expected service requirements is infeasible; however, this conclusion could stem from a software bug, even if the underlying techniques are conceptually sound.

These issues pose major barriers to the adoption of AI.

Numerous efforts have been made to address the aforementioned issues. Increased research into better algorithmic heuristics to improve AI, software testing, and post-hoc explanations all contribute to mitigating the issues. Note, however, that such approaches do not *fully eliminate the concerns*. Unless we fundamentally resolve the problems, there will always be room for concern.

There is a common belief that using black-box AI is necessary because we need complex functions to capture the intricate interactions of our real world. The unfortunate consequence of this view is that it leaves out modern transparent AI approaches from the start, even though such approaches may very well be the most appropriate choice for the given application.

In contrast, this paper advocates an alternative approach: focusing on AI algorithms that synthesise outputs we can truly understand—achieving transparency by design. The core idea is to precisely *specify* the properties that we believe lead to transparent outputs, and then employ *search algorithms* to systematically explore the space of all possible transparent outputs and identify the best one.

By employing search over interpretable outputs, we make our goals explicit and enable provable performance guarantees. **This shifts our focus from doubting AI to critically evaluating whether we are addressing the right problems.**

This process is inherently iterative, functioning as a feedback loop. In each iteration, we may revise the specification based on insights gained from examining the current output. Upon receiving a solution, we either accept it if it meets our expectations or identify flaws in the specification, make necessary adjustments, and repeat the process.

The process outlined above is a general framework that needs to be tailored to the particular domain. The specification of a desirable output may differ significantly across applications, and we may need to leverage domain-specific algorithmic ideas to enhance the performance of the search algorithm. This brings its own challenges.

In the following, we show instantiations of these ideas across three very different domains:

1. Supervised learning, where we summarise historical data using models represented as *decision trees*—hierarchical models that label data through simple decision rules—taking into account a variety of objectives and constraints. For interpretability reasons, we focus on *small* decision trees, which typically require only three or four questions to label a single data point.
2. Controller synthesis, where we construct policies also in the form of *small* decision trees to control deterministic black-box systems.
3. Certificates that prove the infeasibility of a combinatorial optimisation problem, which can be independently verified to validate these claims, regardless of the algorithm that generated them.

We review recent success in these areas, reflect on common concerns and misunderstandings, and highlight the key ideas that enable these algorithms to work in practice.

The main message is that, although synthesising transparent AI is challenging—and may seem implausible due to the exponentially large search space—carefully designed techniques have made it practical to solve problems while addressing concerns about performance and confidence. While not all problems are yet amenable to transparent approaches, the significant benefits underscore the importance of pursuing approaches that are inherently transparent.

For the cases discussed in this paper, tremendous progress has been made in recent years—advancements that would have been unimaginable just a few years ago. This highlights the critical importance of pursuing approaches that are inherently transparent.

2 Search for Supervised Machine Learning with Optimal Decision Trees

The goal of supervised machine learning is to synthesise ("learn") a prediction function that estimates a quantity of interest by observing a dataset of labelled input-output pairs. For high-stakes domains, such as health care and criminal justice, interpretable prediction functions are crucial.

It has been shown that for many applications—in particular those involving well-structured data—'simpler' models that are easily interpretable can represent the data equally well [24, 13, 6] and in some cases even outperform more complex models (such as neural networks or ensembles) by incorporating additional objectives and constraints during training.

Surprisingly, despite substantial evidence supporting interpretable models—particularly decision trees—such models are often met with scepticism and not considered *by default*.

In the following, we start off by illustrating the value of decision trees for a cancer treatment application, show the broad range of objectives and constraints that may be considered when training decision trees, discuss common concerns, and explain the main ideas for synthesising small decision trees in practice that offer superior performance whilst maintaining interpretability.

2.1 Illustrative example: oncology (cancer treatment)

Consider a supervised machine learning task related to oncology (cancer treatment), where the goal is to predict whether a mutation in a human protein is considered oncogenic (dangerous) or benign.

Following the substantial effort of data collection and data processing [26], researchers trained a complex boosted tree ensemble [27] to make these predictions, resulting in a success story given the high performance of the final model.

They also explored decision trees as an interpretable model, but the performance was unsatisfactory—largely due to the use of a widely adopted heuristic based on CART, an algorithm developed in the 1980s. In contrast, recent algorithms based on search have shown significantly better performance.

For demonstration purposes, we used a modern search-based algorithm—STreeD[1] [30]—to compute a decision tree optimised for performance (F1-score in this case). Unlike typical heuristic approaches employed in machine learning (e.g., CART), STreeD exhaustively searches for the best-performing small decision tree amongst all small decision trees. The model was trained and evaluated using the same dataset and folds as the black-box model.

And the result: our decision tree had comparative performance to the black-box model (both models achieving an F1-score around 93%), whilst only requiring *three predicate nodes* (Figure 1a)! It is evident that our decision tree is the more desirable model. In this case, we did not have to choose a trade-off between accuracy and interpretability: we could have both.

It is worth noting that achieving high performance does not necessarily imply that the model is good, regardless of the form of the model. An expert is required to validate the model and conclude that the model is indeed making predictions based on sensible features. The interpretable nature of decision trees allows us to understand how the predictions are made and further iteratively refine the data if needed, eventually leading to a model that we can trust. This is much harder to achieve using opaque models such as neural networks.

The above example is not an isolated case. Similar situations have been observed by many researchers on a wide range of datasets [24, 13, 6]. When considering well-structured datasets—such as tabular datasets as in the oncology example—interpretable models such as small decision trees can match the performance of more complex models whilst offering the advantage of greater interpretability.

2.2 Optimal decision trees

If we accept that *small* decision trees—defined here as those with a depth of three or four, or consisting of only a few nodes—are interpretable, it is natural to ask for the tree that *best* represents the dataset amongst all possible decision trees. In other words, we seek the *optimal decision tree*.

It is *crucial* to recognise that no decision tree is *universally optimal*. Instead, optimality is to be understood given a *specification*, which consists of:

1. Constraints, which implicitly define the (exponentially large) feasible set of decision trees. Examples include imposing restrictions on the number of nodes in the tree or fairness considerations.
2. An objective function, which defines the quality of the tree, typically with respect to a training dataset. Examples of objective functions are accuracy, sparsity (a trade-off between accuracy and the number of nodes of the tree), the F1-score, the average decision length of a query, and may even consider optimising multiple objectives simultaneously.

We may observe that decision trees are not necessarily 'simple': these are nonlinear functions that may consider a variety of requirements as part of training, *and* they also happen to be interpretable!

Optimal decision trees have recently received significant attention, e.g., see the recent survey from 2023 [11], with the caveat that many papers have been published since.

A key advantage over traditional heuristic-based approaches (e.g., CART, C4.5) is that optimal approaches are *guaranteed* to find the

[1] https://github.com/AlgTUDelft/pystreed

best representation of the data and support constraints, which can be notoriously difficult to include in heuristic algorithms, e.g., constraints regarding fairness. Since such trees are typically also *small*, we obtain the best of both worlds for many real-world datasets: highly performant models and interpretability.

2.2.1 *Flexibility of decision trees*

Before discussing algorithms for optimal decision trees, we first provide a non-exhaustive list of different applications for decision trees that take into account a variety of constraints and objectives. This demonstrates the flexibility of decision trees as nonlinear functions that offer considerable power in capturing complex relationships in data in a fairly interpretable manner.

Classification [13] (Figure 1a) This is the classic application of decision trees, used to discriminate data points between a set of discrete classes based on input features.

Regression [28] (Figure 1b) These trees generalise classical linear regression by adding a nonlinear layer: given an instance, the regression tree uses a set of rules, organised in a tree-like manner, to determine which internal linear regression model to consider. This is particularly suitable for data that can be explained by linear regression after performing data partitioning.

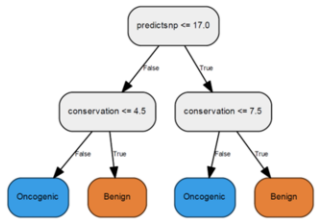
Demographic Parity (Group Fairness) [29] We may seek the decision tree that best reflects historical data while ensuring similar outcomes across different groups. Fairness, in this case, is quantified as a numerical measure of violation, with a strict limit imposed on acceptable disparity. Rather than producing a single tree based on a fixed, potentially arbitrary fairness threshold, we may instead generate a set of trees representing the full trade-off spectrum between faithfulness to historical data (accuracy) and fairness—the Pareto front (see Figure 1c)—allowing experts to select the appropriate balance.

Survival analysis [19] (Figure 1d) In applications where the goal is to predict a singular, non-repeating event—such as death or mechanical failure—the data may be (right-)censored, i.e., the data only records the time of the event up to a certain point, beyond which the exact time of occurrence is unknown. Using nonlinear functions that model the probability of the event occurring at each time point can be appropriate, which can be directly incorporated into decision trees to enhance predictive accuracy.

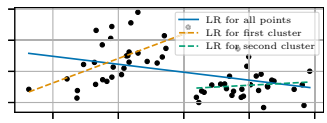
Imbalanced data [12] For datasets where one class significantly outweighs the others, accuracy alone may not be appropriate. High accuracy can be achieved simply by predicting the majority class in every case, which is especially problematic in applications where detecting the minority (e.g., a defect or illness) class is crucial. In such scenarios, it is more appropriate to optimise for nonlinear objectives that better capture the trade-off between detecting true positives and avoiding false alarms, such as the F1-score. We may then seek a tree that maximises such metrics.

Cost-sensitive classification [30] In some applications, it may be desirable to simultaneously optimise accuracy and the cost of evaluating features, e.g., performing a medical test may incur a cost. Optimal decision trees can optimise over both of these criteria.

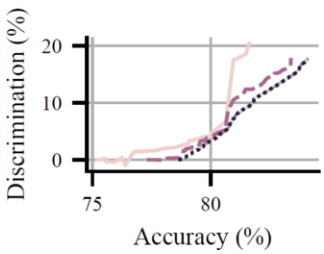
These are just some examples of applications for which we have algorithms that can effectively synthesise small optimal decision trees in practice. Other applications can be supported by adapting existing techniques or by leveraging our general decision tree framework, STreeD [30], which can be customised to meet diverse requirements without additional research efforts.



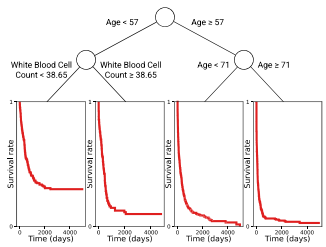
(a) A classification tree predicting whether a mutation is benign or oncogenic; trained on oncology data [26] using STreeD [30].



(b) Graphical representation of a regression tree. A single linear regression model cannot capture two separate clusters (middle line), whereas a tree can make a case distinction to decide which of the two regression models to use (each cluster has its own regression model).



(c) Pareto front representing all trade-offs between accuracy and discrimination. Each point is a distinct decision tree, while different lines correspond to trees of depths two, three, and four.



(d) A survival tree may use nonlinear functions in its leaves to estimate the longevity of a patient based on historical data.

Figure 1: Examples of decision tree use cases from Section 2.2.1.

2.3 *Concerns and limitations*

Decision trees offer clear value but are not suitable for every application. In our experience, we observed many situations where concerns surrounding decision trees were based on misunderstandings of optimal decision trees, apples-to-oranges comparisons, or deeper questions that go beyond decision trees and relate broadly to machine learning. We address a subset of these concerns to shed light on these common issues and provide a more nuanced perspective on the role and value of decision trees.

Why do we need decision trees when we can use opaque models and rely on explainable AI? The problem with explainable AI is the lack of guarantees: no method can reliably explain the behaviour of opaque models. This is by definition, otherwise we could replace the opaque model with its explanations, which would lead to an interpretable model! For high-stakes domains, interpretative models should be the primary choice (see [24] for more discussion).

Are decision trees are too simple, do we not need more complex models? This is a common concern; however, there are two main points. First, small decision trees are interpretable, but not necessarily simple, as illustrated in Section 2.2.1. Given that we may directly optimise for a variety of objectives and constraints, this may make small decision trees more appealing than other models in practice. Second, while decision trees may be less suitable for unstructured data—such as image-based vision tasks—there are many documented cases where decision trees perform comparably to complex, opaque models [24] on well-structured data (e.g., tabular datasets), which is common in applications requiring interpretability.

Does synthesising the best decision tree for the training dataset

lead to overfitting? First, *small* decision trees have less opportunity for overfitting. Second, this is a deeper problem that spans across machine learning. For example, we have also observed in our work [29] that a model that is fair on the training data may exhibit different fairness performance on the test set. However, given that we consider learning predictive models as an optimisation problem for which we use search algorithms to compute the *best model* according to the formulated problem, we have a unique advantage: If the outcome is not desirable, then this prompts us to revise the way we formulated our optimisation problem. In other words, rather than worrying about *how* we will compute our model, we can direct our attention to defining *what* we need to solve.

Is training optimal decision trees too costly? This is a concern that largely depends on the application. In many cases, we can afford to spend more computational time offline, but when this is not the case, then indeed heuristics may be a better option. We note that this concern may also come from a time when optimal decision tree methods were much slower; nowadays, computing optimal decision trees for datasets that took hours a few years ago takes seconds.

Are decision trees too brittle? Owing to the interpretability of decision trees, we can indeed understand how the model operates and identify exactly the conditions under which the model is brittle. This can be seen as an advantage, since opaque models may also be brittle, but we simply may not be aware of it, which is a more dangerous situation. A recent promising idea related to robustness has been to consider all decision trees that exhibit optimal or close-to-optimal performance [34], which provides a more comprehensive view of the data compared to a single decision tree.

Do decision tree methods require binarising features, does this lead to lower accuracy? First, even with binarisation, optimal decision tree methods may still offer greater accuracy and additional functionality compared to heuristics. Second, recently we have seen methods that do not require binarisation, e.g., [8, 22].

I tried optimal decision trees, and they do not work! This concern is common; for instance, it arose in the oncology application discussed in Section 2.1. The standard approach to decision tree induction still relies on heuristic algorithms developed in the 1980s, such as CART. While these methods remain useful, modern optimal decision tree algorithms can achieve superior performance—but are not yet widely adopted. Unlike conventional machine learning methods, optimal decision tree algorithms explore an exponentially large search space exhaustively. This shift introduces new challenges in understanding and working with such models. For example, it may come as a surprise that training an optimal depth-four tree with seven nodes can take longer than training a tree with fifteen nodes, due to the expanded search space. Similarly, much larger heuristic trees may sometimes outperform small optimal trees in terms of accuracy, but such trees offer little in terms of interpretability. Adapting to these differences requires a mindset shift in how such models are developed and used. Over time, as tooling improves, optimal decision trees may become accessible to practitioners without requiring deep knowledge of the underlying algorithms. For a detailed comparison of heuristic and optimal trees—focusing on accuracy and model size—see our recent study [31].

2.4 How to compute optimised decision trees?

Computing an optimal decision is an $\mathbf{NP}$-hard problem [20]. Nevertheless, owing to many recent advances, we can compute *small* optimal decision trees in practice reasonably quickly, which are precisely the decision trees we are interested in from an interpretability perspective. These are decision trees up to depth four, which could be understood as our definition of decision tree interpretability, but in some cases, larger trees may also be computed.

The key insight is to exploit the unique decision tree structure as part of the synthesis algorithm. When searching for an optimal decision tree, in many use cases, there is a *divide-and-conquer* structure that can be applied recursively. Once a node has been assigned a particular predicate, each child node gives rise to a smaller optimal decision tree problem, and crucially, these two subproblems can be solved independently from one another.

This high-level algorithmic idea may be formulated in a single recursive formula [13]. For simplicity, consider binary classification. Let $T(\mathcal{D}, d)$ be the misclassification score of the optimal decision tree of depth d on the dataset $\mathcal{D}$, $\mathcal{F}$ be the set of considered predicates (binary features), $|\mathcal{D}^+|$ and $|\mathcal{D}^-|$ be the number of positive and negative instances (binary classification), and $\mathcal{D}_f$ and $\mathcal{D}_{\bar{f}}$ be the datasets that contain instances from dataset $\mathcal{D}$ that satisfies and does not satisfy predicate f, then we may compute the minimum number of misclassifications by using the following formula:

$$T(\mathcal{D}, d) = \begin{cases} \min\{|\mathcal{D}^+|, |\mathcal{D}^-|\} & d = 0 \\ \min_{f \in \mathcal{F}} \{\ T(\mathcal{D}_f, d-1)\ +\ \ T(\mathcal{D}_{\bar{f}}, d-1)\} & d > 0 \end{cases} \tag{1}$$

The state-of-the-art makes use of a range of techniques to speed up the computation of the above formula, e.g., specialised subroutines for small depths [13], lower bounding techniques to avoid recursive calls [13, 21], and caching techniques [13] for binary features.

A generalised version of the above equation covers more objectives and constraints, but we omit details for simplicity [30]. Not all decision tree problems are amenable to the above formulation. Formally, it is important to ensure that subtrees may be optimised independently by checking a *separability* property [30], but this property holds for all supervised learning examples discussed in this paper.

We note that optimal decision tree algorithms relied on the user restricting the pool of allowed predicates (datasets with binary features) to maintain performance, but recently there have been several works that lift this requirement and allow optimising directly with continuous features [8, 22].

We stress the importance of designing specialised algorithms that leverage the unique structure of decision trees. This structural leverage is the key reason state-of-the-art optimal decision tree algorithms achieve orders-of-magnitude runtime improvements over generic approaches based on mixed-integer programming, constraint programming, and SAT-based methods, making optimal decision trees a valuable asset in our machine learning toolbox.

3 Search for Controller Synthesis with Optimal Decision Trees

We now discuss how the same high-level idea of searching for interpretable models may be transferred to a very different setting, namely, controlling (deterministic black-box) dynamical systems.

As an example, consider *cart-pole*, a classical control problem. The task is to balance an upright pole standing on a cart for as long as possible while only being able to control the cart by nudging it to the left or to the right. The mechanics of the system are governed by nonlinear physical equations, see Figure 2.

The key difference compared to the supervised machine learning setting is that, by default, there is no ground truth that could be used

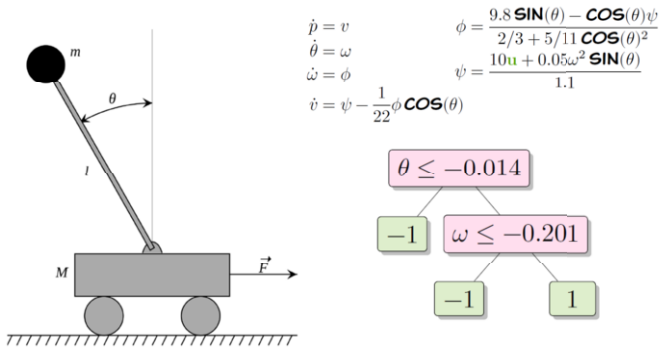


Figure 2: Cart-pole, a classical control problem, equations describing the evolution of the system as a function of the control action *u*, and an example decision tree controller.

to train the controller. Instead, the controller must be constructed directly based on the evolution of the dynamical system over time.

In the following, we take a step further and consider the dynamical system a *deterministic black-box*. This captures the scenario where the system is either inconvenient to represent as a set of equations (e.g., given by a procedural algorithm) or otherwise too complex.

Although there may be many valid controllers (policies), we may be interested in a subset of those controllers that we can understand. Given the black-box nature of the system, having an interpretable controller is paramount to understand system behaviour.

Unfortunately, interpretable controllers are comparatively underexplored in the literature when compared to opaque options, e.g., reinforcement learning. Decision trees are a good alternative.

The surprising observation is that even though the system dynamics are black-box, we can still compute interpretable controllers that are, in a sense, optimal. This requires specialised techniques, and although the techniques are in an early stage, the results are promising.

3.1 Optimal Decision Tree Controllers

The first step is to determine which controllers are considered interpretable. Following our approach of using search to find the best output (controllers) amongst outputs that we can understand, we first need to define the setting we are considering, since only then may we talk about interpretability and optimality.

A *black-box dynamical system* takes as input the current *state* (a d-dimensional real-value vector) and a discrete *action* (typically given by the controller or policy given a state), and computes the next state. Note that *how* the black-box performs its computation is abstracted away; we merely assume that it is possible to compute.

In the cart-pole example, after applying force to nudge the cart to the left or right side (an action), the system returns the new position and velocity of the cart and pole (the state).

Given an initial state, a controller, and a black-box dynamical system, we may compute the trajectory of the system by repeatedly applying the controller to the system. The goal is to keep the system in a *safe state for as long as possible*. We may also be interested in reaching a goal state as quickly as possible.

In the cart-pole example, a state is considered safe if the angle of the pole with respect to the cart is within a small range.

For interpretability reasons, we restrict our attention to controllers that take the form of a decision tree. We then seek to compute an *optimal tree controller*, i.e., a controller that keeps the system in a *safe state* for as long as possible given an initial state. In other words, there is no other controller that maintains the safe state for longer.

Multiple initial states may be simultaneously considered, in which case we seek a single controller that maximises the minimum performance amongst all initial states. The minimisation problem may be analogously defined, i.e., find the controller that leads the system towards a target state as quickly as possible.

To make the search procedure manageable, we restrict the definition of our controller to use a discrete set of actions (e.g., two actions) and a set of predicates defined a priori. We consider predicates of the form $[x_i \geq c]$, where x is a component of the state vector and c is a constant. The predicates may be chosen to uniformly cover the state.

We may now observe that the search space of all decision tree controllers, as defined above, is finite but exponentially large. We may proceed to search for the optimal tree controller, despite the system being represented as a black-box. In fact, the black-box representation is a powerful abstraction that allows for easy incorporation of constraints and other requirements without needing to develop specialised algorithms, apart from the general search procedure.

As a result, we obtain decision tree controllers with the best performance that is possible to achieve amongst the defined set of feasible decision tree controllers, and their small size make it easier to understand the decision making process of the controller.

3.2 Concerns and limitations

Decision tree controllers offer a novel approach to automated control. However, tree controllers are a relatively new concept, the research is in its early phases, and tree controllers are not applicable in every situation, so it is natural that there are many concerns regarding their applicability. We discuss common issues below.

Can small decision tree controllers capture high-dimensional environments? This is a fundamental limitation of small trees: if the system is highly-dimensional, and all dimensions must be taken into account for decision making, then indeed a small tree may not be appropriate. However, when the system *is* controllable by a small tree, then training a small and optimal tree controller is highly beneficial.

Are decision tree controllers too simple? Similar points hold as discussed in Section 2.3. On the one hand, small tree controllers are interpretable and, in that sense, simple. On the other hand, we may synthesise tree controllers to support black-box environments, which may encode a variety of complex behaviours that other approaches would have difficulties dealing with. A more holistic view is required when assessing tree controllers for a given application.

Is the runtime to train optimal tree controllers too high? This largely depends on the size of the tree and the application. We believe this will become less of an issue as techniques advance.

3.3 How to synthesise optimal tree controllers?

Algorithms to synthesise *optimal* decision tree controllers for black-box systems are a recent invention [14]. Given the definition of the tree controller from the previous section, the idea is to exhaustively explore the set of such controllers, evaluate each tree with respect to the black-box system, and take the best tree.

There are two key insights that make the approach possible in practice. First, given that the systems considered are deterministic, the sequence of states obtained by applying a decision tree controller to an initial state is a sufficient representation of the behaviour of the system, which allows us to reason about the system even though it is a black-box. Second, when enumerating decision trees, by analysing previously encountered states, we may determine sets of decision trees that have identical performance, which leads to pruning a large

portion of the search space. This makes it possible to synthesise optimal small trees in practice for certain problems, despite the exponentially large search space.

Beyond the setting considered here, small tree controllers have seen increasing interest, e.g., optimal [33] and heuristic controllers [2, 3] for stochastic white-box systems, heuristic tree controllers based on an explicit representation of all state-action pairs [4, 5], and verification of tree policies in continuous time [25].

Optimising small tree controllers is a promising research direction for interpretable AI. As the techniques mature, we believe tree controllers will become standard in automated control, offering unique advantages in interpretability compared to classical approaches.

4 Search for Certificates of Combinatorial Optimisation Problems

When we discussed using search to synthesise interpretable models (decision trees), we were mainly focused on being able to understand the output, i.e., the resulting decision tree model.

However, if we push the interpretability idea further, we may not settle for only obtaining a model, but we may also be interested in understanding *why* the generated model is indeed the best. In other words, how can we be sure that a better model does not exist?

This is a general question that relates to all search algorithms that deal with combinatorial problems. There are two interconnected reasons to address this question. First, for interpretability reasons, we may be interested in understanding the main bottleneck of the underlying problem. Second, given that search algorithms require both conceptual advancements and well-executed engineering efforts to work in practice, it is reasonable to suspect that there could be an error in some part of the algorithm or its implementation. Considering search broadly, there have been many documented cases where even well-established tools reported incorrect results [10, 9, 16, 1].

The answer to this problem is to follow the idea of synthesising an interpretable output: we want a *certificate* that can be used to explain that no better solution exists in the defined search space. Ideally, the certificate is something that we can easily understand, so that we can easily convince ourselves that the final conclusion is indeed correct.

Certificates effectively removes any doubts regarding the performance limits of the search algorithm, both conceptual and implementation mistakes. Since a certificate is a mathematical proof with respect to the problem definition, after validating the certificate, we may be certain about the certified claims, even if the algorithm producing the certificate may have unsoundness issues.

We discuss constraint programming as a paradigm for modelling and solving combinatorial optimisation problems, the form of certificates that could be used to explain the inexistence of better solutions for constraint programming, common criticisms and open research questions, and give a high-level idea of certificate generation.

4.1 Constraint Programming

To facilitate trust, it is important to have a modelling language that defines problems using high-level concepts: the higher the level, the better. Certificates may then operate over this high-level definition.

Constraint Programming is a paradigm for modelling and solving combinatorial optimisation problems matching precisely this requirement. The feasible space is defined implicitly through a set of constraints, which are defined as predicates over discrete variables. For optimisation problems, we may specify an objective function that allows discriminating between feasible solutions.

A key feature of constraint programming is that the form of the constraints is not predefined by the paradigm; essentially, arbitrary relations over variables may be defined. Examples of common constraints include the *all-different* constraint, which imposes that no two integer variables may take the same value, and the *disjunctive* constraint, which defines that tasks must be scheduled in time such that no overlap appears. There are over 400 constraints, with new constraints being introduced to suit particular application needs.

Constraint programming not only allows for an expressive paradigm for specifying combinatorial problems, but also includes solving algorithms that directly exploit the high-level structure to prune the search space (propagators). This makes constraint programming an appealing paradigm for modelling and solving, and provides opportunities from an interpretability perspective.

4.2 Certificates

The certificates [15] we discuss for constraint programming leverage the high-level description of the combinatorial problem and corresponding algorithms that exploit this explicit structure.

As an example, consider the following infeasible problem and its certificate of infeasibility. The certificate has been designed to be intuitive and easy to follow—regardless of how it was generated.

$$
\begin{aligned}
& x, z \in \{0, 1\} \\
& y \in \{0, 1, 2\} \\
c_1 \colon\ & 2x + y + 2z \geq 2 \\
c_2 \colon\ & 2x + y - 2z \geq 0 \\
c_3 \colon\ & 2x - y + 2z \geq 0 \\
c_4 \colon\ & 2x - y - 2z \geq -2 \\
c_5 \colon\ & -2x + y + 2z \geq 2 \\
c_6 \colon\ & -2x + y - 2z \geq 0
\end{aligned}
$$

Step #	Implied by	Reasoning
1	Constraint c_3	$[x \leq 0] \land [y \geq 2] \longrightarrow [z \geq 1]$
2	Constraint c_4	$[x \leq 0] \land [y \geq 2] \longrightarrow [z \leq 0]$
3	Steps 1 & 2	$[x \leq 0] \land [y \geq 2] \longrightarrow \bot$
4	Constraint c_1	$[x \leq 0] \land [y \leq 1] \longrightarrow [z \geq 1]$
5	Constraint c_2	$[x \leq 0] \land [y \leq 1] \longrightarrow [z \leq 0]$
6	Steps 4 & 5	$[x \leq 0] \land [y \leq 1] \longrightarrow \bot$
7	Step 6	$[x \leq 0] \longrightarrow [y \geq 2]$
8	Step 3	$[x \leq 0] \longrightarrow [y \leq 1]$
9	Steps 7 & 8	$[x \leq 0] \longrightarrow \bot$
10	Constraint c_5	$[x \geq 1] \land [y \leq 2] \longrightarrow [z \geq 1]$
11	Constraint c_6	$[x \geq 1] \land [y \leq 2] \longrightarrow [z \leq 0]$
12	Steps 10 and 11	$[x \geq 1] \land [y \leq 2] \longrightarrow \bot$
13	Step 9	$\top \longrightarrow [x \geq 1]$
14	Steps 13 and 12	$[y \leq 2] \longrightarrow \bot$
15	Step 14	$\top \longrightarrow [y \geq 3]$
16	Initial domain	$\top \longrightarrow [y \leq 2]$
17	Steps 15 & 16	$\top \longrightarrow \bot$ (`Infeasible`)

A key point, which aids interpretability, is that the certificate is designed to break down the final statement (e.g., 'infeasible') into smaller parts that could be understood independently (separated by horizontal lines in the example). Steps that reason over *individual* constraints (e.g., Steps 1 and 2) are interleaved with deriving implied constraints, which are obtained by reasoning over *multiple* constraints (e.g., Step 3).

In the end, we obtain a trivially implied constraint that asserts that the problem indeed has no feasible solutions, or a bound on the objec-

tive function (for optimisation problems, not shown in the example). Since each step of the certificate is kept simple, we may easily convince ourselves of its correctness. We may even conclude additional information, e.g., based on the certificate, the domain of the variable z plays no role in the infeasibility of the problem.

In the example, we used linear inequalities, but any arbitrary constraints may also be used. This could influence the interpretability of the certificate, e.g., using high-level constraints specialised for a particular domain may make the certificate easier to understand.

For any solvable constraint programming instance, we can, in principle, generate a certificate. In this sense, the certificate is precisely the interpretable output that helps us understand why the problem is infeasible or why no better solution exists.

4.3 Concerns

Certificates for combinatorial problems have a long history [32, 18, 17], but using high-level constraints as part of the certificates is only recent [15]. This opens new opportunities, but also raises questions.

Are certificates too detailed to be interpreted by a human? Humans may reason differently than solving algorithms. On the one hand, high-level certificates precisely capture the reasoning used in the solving algorithm in a mathematically correct manner without direct redundancies, given current solving techniques. In this sense, this matches our notion of interpretability, since we can inspect the certificate to understand the reasoning process. On the other hand, it is true that a large certificate may be difficult to grasp. In these cases, devising even higher-level constraints or adding another layer of abstraction (e.g., minimal unsatisfiable subsets of constraints [7]) on top could support human interpretability.

Does using high-level constraints make it more complicated to check the correctness of the certificates, compared to using low-level constraints? Encoding combinatorial problems using simpler constraints has the advantage of a simpler formalism, relevant for implementation reasons. However, this comes with the downsides that modelling of the combinatorial problem is more complicated, which is a serious barrier in practice, and we may lose computational and interpretability benefits of high-level constraints.

4.4 How to generate certificates?

The discussed certificates [15] have been designed to trace the behaviour of solvers. In this sense, solving algorithms may directly produce the certificate by recording their internal reasoning steps.

However, solving algorithms perform a large number of operations, and due to $\mathbf{NP}$-hardness of the problems, many such operations may be redundant. This can result in prohibitive runtime and memory overhead for real-world problems.

To avoid the overhead, the core idea is to only record a proof sketch at runtime, and as a post-processing step, remove redundant parts and expand the remaining sketch into a certificate. In this manner, many of the redundancies inherent to $\mathbf{NP}$-hard problems may be avoided.

The sketch may be obtained directly from the search algorithm. In short, constraint programming solvers use backtracking search to exhaustively explore all possible assignments to the problem. This is complemented by propagation, which removes variable assignments that cannot be part of a feasible assignment given the current partial assignment. Each time an infeasible assignment is detected because of propagation (i.e., propagation removes all possible options), prior to backtracking, the solver performs a conflict analysis procedure [23] to extract a *nogood*, i.e., a conjunction of atomic statements that logically imply the conflict. A nogood may also be seen as a rule that has been derived by jointly reasoning over multiple constraints, e.g., in the example certificate, Step 3 is a nogood.

Nogoods encountered during the search compose the sketch of the certificate, based on which a full certificate may be obtained as a post-processing step. Determining which nogoods to include in the certificate is a core challenge, which is handled by the search algorithm, but once the nogoods have been successfully determined, it is easy to reconstruct a derivation of individual nogoods, e.g., the nogood in Step 3 may be derived by assuming its left-hand side and then following the propagations that lead to a conflict (Steps 1 and 2). We defer the interested reader to the paper for more details [15].

5 Discussion and Conclusion

AI has been establishing itself as a core part of society. Although this has clear benefits, there are fundamental concerns about AI when it comes to performance guarantees and our confidence in the technology, in particular when the underlying AI algorithm is black-box.

To address these issues, in this paper, we advocated for an approach that achieves transparency by design: we precisely define what we consider interpretable, and then use search algorithms to find the best possible output. This allows us to focus on understanding the problem that needs to be solved, rather than doubting the AI.

We discussed this idea in three very different contexts with vastly different solving algorithms: supervised machine learning, control policies, and certificates for combinatorial problems. Although in each of these cases, the proposed approach leads to an exponentially large search space, specialised search algorithms could be designed that make this feasible in practice, addressing the aforementioned concerns of performance and confidence in a principled way.

The focus was on the algorithmic aspects of interpretable AI. However, it is important to recognise that algorithms represent only one part of the solution when addressing real-world problems.

Interpretability is a central concept in many applications of AI, but it remains an elusive term—its meaning depends heavily on both the domain and the context, with no universally accepted definition of interpretability exists. Applying interpretable AI requires close collaboration with domain experts to validate and refine the approach.

However, within the AI community, this human-centred aspect has received significantly less attention compared to algorithmic development. For example, research on understanding what makes decision trees interpretable for particular applications is scarce. Similarly, it remains unclear how to best explain that no better solution exists for a combinatorial problem. To fully realise the potential of interpretability, we must obtain a better understanding of effectively integrate them into practical workflows.

We must also be mindful not to over-interpret interpretable AI models. For example, while a small decision tree may be easy to understand, it can still capture spurious correlations in the data. Moreover, a single decision tree represents just one of potentially many possible perspectives on the dataset. A more nuanced approach—one involving multiple models and actively challenging their conclusions—is necessary, even when the models are interpretable.

Lastly, we considered a static scenario. However, we acknowledge that the environment in which we use AI is constantly changing. For example, a model that is considered fair today may no longer be fair in a year. As another example, our problem definitions are merely approximation of reality; we cannot hope to capture every aspect of the problem. It is therefore essentially that we remain vigilant in monitoring (interpretable) AI models after they have been deployed.

Acknowledgements

We are grateful to Anna Lukina, Stanislav Mazurenko, Mimi Jasarevic, Koos van der Linden, Elif Arslan, and Maarten Flippo for their reviews, which played a valuable role in refining this paper. We further thank Koos van der Linden for training and providing the image of the decision tree for the oncology dataset, and Anna Lukina for the cart-pole figure.

References

[1] Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Nightingale. Metamorphic testing of constraint solvers. In *Conference on the Principles and Practice of Constraint Programming*, 2018.

[2] R. Andriushchenko, M. Ceska, D. Chakraborty, S. Junges, J. Kretinsky, and F. Macák. Symbiotic local search for small decision tree policies in mdps. In *Conference on Uncertainty in Artificial Intelligence*, 2025.

[3] R. Andriushchenko, M. Češka, S. Junges, and F. Macák. Small decision trees for mdps with deductive synthesis. In *International Conference on Computer Aided Verification (CAV)*, 2025.

[4] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínskỳ, M. Weininger, and M. Zamani. dtcontrol: decision tree learning algorithms for controller representation. In *International Conference on Hybrid Systems: Computation and Control*, 2020.

[5] P. Ashok, M. Jackermeier, J. Křetínskỳ, C. Weinhuber, M. Weininger, and M. Yadav. dtcontrol 2.0: explainable strategy representation via decision tree learning steered by experts. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2021.

[6] D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 2017.

[7] I. Bleukx, R. Boumazouza, T. Guns, N. Laage, and G. Poveda. Modeling and explaining an industrial workforce allocation and scheduling problem. In *International Conference on Principles and Practice of Constraint Programming*, 2025.

[8] C. E. Briţa, J. G. M. van der Linden, and E. Demirović. Optimal classification trees for continuous feature data using dynamic programming with branch-and-bound. In *AAAI*, 2025.

[9] R. Brummayer, F. Lonsing, and A. Biere. Automated testing and debugging of SAT and QBF solvers. In *International Conference on the Theory and Applications of Satisfiability Testing*, 2010.

[10] W. Cook, T. Koch, D. E. Steffy, and K. Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 2013.

[11] V. G. Costa and C. E. Pedreira. Recent advances in decision trees: an updated survey. *Artificial Intelligence Review*, 2023.

[12] E. Demirović and P. J. Stuckey. Optimal decision trees for nonlinear metrics. In *AAAI*, 2021.

[13] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research (JAIR)*, 23(26):1–47, 2022.

[14] E. Demirović, C. Schilling, and A. Lukina. In search of trees: Decision-tree policy synthesis for black-box systems via search. In *AAAI*, 2025.

[15] M. Flippo, K. Sidorov, I. Marijnissen, J. Smits, and E. Demirović. A multi-stage proof logging framework to certify the correctness of cp solvers. In *International Conference on Principles and Practice of Constraint Programming*, 2024.

[16] X. Gillard, P. Schaus, and Y. Deville. Solvercheck: Declarative testing of constraints. In *International Conference on the Principles and Practice of Constraint Programming*, 2019.

[17] S. Gocht, C. McCreesh, and J. Nordström. An auditable constraint programming solver. 2022.

[18] M. J. Heule. Proofs of unsatisfiability. In *Handbook of Satisfiability*. IOS Press, 2021.

[19] T. Huisman, J. G. M. van der Linden, and E. Demirović. Optimal survival trees: a dynamic programming approach. In *AAAI*, 2024.

[20] H. Laurent and R. L. Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[21] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning (ICML)*, 2020.

[22] R. Mazumder, X. Meng, and H. Wang. Quant-bnb: A scalable branch-and-bound method for optimal decision trees with continuous features. In *International Conference on Machine Learning (ICML)*, 2022.

[23] O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 2009.

[24] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 2019.

[25] C. Schilling, A. Lukina, E. Demirović, and K. Larsen. Safety verification of decision-tree policies in continuous time. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[26] J. Stourac, S. Borko, R. Khan, P. Pokorna, A. Dobias, J. Planas-Iglesias, S. Mazurenko, G. Pinto, V. Szotkowska, J. Sterba, O. Slaby, J. Damborsky, and D. Bednar. Training and test datasets for the predictonco tool [data set], 2023. URL https://doi.org/10.5281/zenodo.10013764.

[27] J. Stourac, S. Borko, R. T. Khan, P. Pokorna, A. Dobias, J. Planas-Iglesias, S. Mazurenko, G. Pinto, V. Szotkowska, J. Sterba, et al. Predictonco: a web tool supporting decision-making in precision oncology by extending the bioinformatics predictions with advanced computing and machine learning. *Briefings in Bioinformatics*, 25(1), 2024.

[28] M. van den Bos, J. G. M. van der Linden, and E. Demirović. Piecewise constant and linear regression trees: an optimal dynamic programming approach. In *International Conference on Machine Learning (ICML)*, 2024.

[29] J. van der Linden, M. de Weerdt, and E. Demirović. Fair and optimal decision trees: A dynamic programming approach. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[30] J. van der Linden, M. de Weerdt, and E. Demirović. Necessary and sufficient conditions for optimal decision trees using dynamic programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[31] J. G. M. van der Linden, D. Vos, M. M. de Weerdt, S. Verwer, and E. Demirović. Optimal or greedy decision trees? revisiting their objectives, tuning, and performance. *arXiv preprint arXiv:2409.12788*, 2024.

[32] M. Veksler and O. Strichman. A proof-producing csp solver. In *AAAI*, 2010.

[33] D. Vos and S. Verwer. Optimal decision tree policies for markov decision processes. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

[34] R. Xin, C. Zhong, Z. Chen, T. Takagi, M. Seltzer, and C. Rudin. Exploring the whole rashomon set of sparse decision trees. *Advances in neural information processing systems (NeurIPS)*, 2022.