An Interpretable Ensemble Heuristic for Principal-Agent Games with Machine Learning

Avinashh Kumar Baswapuram[1], Chen Chen[2], Wenbo Cai[3], and İ. Esra Büyüktahtakın[4*]

[1,4]Grado Department of Industrial and Systems Engineering, Virginia Tech, 925 Prices Fork Road, Blacksburg, 24060, VA, U.S.A

[2,3]Department of Mechanical and Industrial Engineering, New Jersey Institute of Technology, 323 Dr Martin Luther King Jr Blvd, Newark, 07102, NJ, U.S.A.

[*]Corresponding author. Email: esratoy@vt.edu;
Contributing authors: avinashh@vt.edu; cc673@njit.edu; cai@njit.edu;

April 12, 2026

Abstract

This paper addresses the challenge of enhancing public policy decision-making by efficiently solving principal-agent models (PAMs) for public-private partnerships, a critical yet computationally demanding problem. We develop a fast, interpretable, and generalizable approach to support policy decisions under these settings.

We propose an interpretable ensemble heuristic (EH) that integrates Machine Learning (ML), Operations Research (OR), and Game Theory. First, we reformulate a PAM as a mixed-integer program to improve efficiency. Next, we solve thousands of PAM instances under varying configurations to generate training data for ensemble tree-based ML models that identify key solution patterns. These patterns form a hierarchical heuristic that provides feasible and interpretable solutions. We demonstrate the EH's efficacy in managing the Emerald Ash Borer (EAB) infestation, an urgent public-policy threat to U.S. ash trees. Empirical results show that the EH produces high-quality solutions with 1–2% optimality gaps while significantly reducing computational time compared to exact optimization. Furthermore, the heuristic explains predictions using an average of 4.5 of 9 input features, enhancing transparency.

Our findings demonstrate that the EH promotes rapid, informed, and accountable policy decisions by balancing interpretability with computational efficiency. Practically, it supports real-time simulations for stakeholders without specialized ML or OR expertise. Methodologically, it demonstrates a robust integration of optimization and machine learning to solve complex policy models. Beyond the EAB application, this approach provides a scalable framework for real-time decision support where transparency and justification are paramount.

Keywords: machine learning; optimization; principal-agent; ensemble tree models; random forest;

1 INTRODUCTION

This study proposes a novel approach that leverages Machine Learning (ML) to enhance the tractability and interpretability of solutions for Principal-Agent Models (PAMs). Traditional ap-

proaches to solving PAMs often involve complex mathematical optimization, making the solutions difficult to understand and implement. To overcome these challenges, we develop an interpretable ensemble heuristic (EH) for solving PAMs by extracting and utilizing solution patterns identified by ML algorithms. This approach aims to make the solutions more understandable to decision-makers, regardless of their background in optimization or machine learning. By leveraging the power of ML, our heuristic provides efficient and easily understandable solutions to decision-makers, facilitating better decision-making in various applications.

In many real-world scenarios, decision-support models must be repeatedly solved due to the dynamic nature of the underlying problem. By predicting all or part of the solutions, our proposed methodology can either eliminate the need for repeated base problem resolution or substantially reduce computational time. Furthermore, the general trends and insights obtained from the EH can inform the design and improvement of operational decisions in various domains.

Our novel approach not only predict solutions to PAMs, but also extract and interpret meaningful insights form those solutions. This distinguishes our work from the majority of existing literature, which primarily focuses on using ML techniques, such as reinforcement learning (RL), deep reinforcement learning (DRL), and deep neural networks (DNNs), to generate agent interactions and predict optimal outcomes. While related work, such as Bang et al. (2021), has explored the use of ML (e.g., random forests, bagging models) to identify non-linear patterns (in their case, between predictor variables and likelihood of terrorist attacks), our work goes further by emphasizing the development of an interpretable heuristic, ensuring that the extracted insights are easily understandable and explainable to decision-makers.

To demonstrate our methodology, we apply it to a PAM developed by Chen et al. (2024). This PAM incentivizes private landowners to inspect and treat or remove their ash trees to mitigate the impact of Emerald Ash Borer, an invasive insect that poses a significant threat to ash trees and causes billions of dollars of damage in North America (Kıbış et al. 2021, Bushaj et al. 2022, Büyüktahtakın and Haight 2018). Game-theoretic approaches have been widely adopted for the management of invasive species, including using Shapley values to solve cooperative games based on dynamic models for the control of invasive species (Büyüktahtakın et al. 2013, 2011); implementing dynamic contractual mechanisms that involve transfer payments between landowners (Bhat and Huffaker 2007) and with payment schedules (Liu and Sims 2016); and employing Nash bargaining models to address payment negotiations between agents (Cobourn et al. 2019, Siriwardena et al. 2018). Furthering this line of research, Chen et al. (2024) develop a PAM that focuses on designing effective public reimbursement policies to address EAB infestations on private land. This work extends previous research that has utilized stochastic mixed-integer non-linear programming methods to optimize the search and control of EAB.

Chen et al. (2024) propose that local governments implement cost-sharing programs to encourage landowner participation in EAB management. They formulate two distinct policy design approaches: Infestation-Based Reimbursement (IBR) and Treatment-Based Reimbursement (TBR).

Reimbursements in these programs are linked to the extent of the infestation and the degree of treatment, respectively. Within this PAM framework, the optimal treatment levels and reimbursement amounts are determined for both designs, aiming to maximize the utility of the city forester while ensuring the landowner's willingness to participate. Although Chen et al. (2024) presents an analytical solution for the IBR model, the solutions for the TBR model are incomplete. The authors obtain numerical solutions for the TBR model through an enumeration approach, which becomes computationally expensive as the number of trees on the landowner's property increases. This limitation significantly hinders the applicability of the TBR model for real-time decision-making.

To address this challenge, this study introduces a novel approach. First, we develop an efficient reformulation of the TBR model, which can be adapted to other similar PAMs. Next, we use an ML methodology to build an interpretable ensemble heuristic (EH) for solving the re-formulated PAM. This methodology addresses the critical factors of interpretability, feasibility, and generalizability of ML predictions within the context of PAMs. We construct a hierarchical tree-based heuristic using prediction rules extracted from a random forest (RF) model trained on solved instances. The heuristic effectively captures the broader solution patterns delineated by the RF model while remaining easily explainable to individuals with limited expertise in ML or Operations Research (OR). Additionally, we demonstrate the generalizability of our heuristic by applying it to predict optimal solutions for instances with varying number of trees. The resulting predicted solutions are of high quality and maintain feasibility through the incorporation of alternative recommendations to address infeasibility. We also present improvements in computational efficiency, with significant reduction in solution time compared to traditional optimization methods. Finally, we compare the performance of our heuristic with other established heuristics and ML approaches. This analysis assesses the effectiveness and limitations of our approach and provides valuable information on the effective management of EAB in local communities in the US. To the best of our knowledge, this is the first machine learning approach developed for decision-making in controlling forest invasive insect problems.

2 LITERATURE REVIEW AND CONTRIBUTIONS

Over the past decade, there has been a noticeable increase in the application of ML algorithms in OR . However, research specifically focuses on explaining the outcomes of optimization models using ML techniques remains relatively limited.

Bertsimas and Stellato (2021) present an ML approach that shares some similarities with our work. Their method focuses on understanding the relationships between tight constraints at optimality and input parameters in both continuous- and mixed-integer convex optimization problems. Similarly to our work, they use a tree structure to explain the conditions on input parameters that lead to specific predictions. Their method involves training ML algorithms to predict tight constraints at optimality, which they define as "strategies". The final outcome of their approach is a set of interpretations derived from the ML model, which predicts the three most likely optimal strategies

for a given instance.

In contrast to their approach, we develop an EH to directly predict a unique optimal solution pattern. Furthermore, while Bertsimas and Stellato (2021) demonstrate the generalizability of their approach by presenting implementation results on different kinds of problems, we validate our method by training a heuristic on a specific problem size and then evaluating its performance on both larger and smaller problem sizes. Furthermore, they use the Machine Learning Optimizer algorithm presented by Bertsimas and Dunn (2017) to implement their methodology, while we build a heuristic using rules extracted from the trained ensemble of ML models.

Another study closely related to our work in terms of the interpretation of ensemble models is presented in Sagi and Rokach (2020). They propose a novel "decision forest" method that integrates the prediction rules of a random forest into a single interpretable tree. Their method builds a set of rule conjunctions from the given decision forest and organizes them into a tree structure, enabling faster predictions for new inputs. However, their approach imposes a limitation on tree depth, restricting it to a maximum of five features, a limitation that our method does not impose. Additionally, their node-splitting criterion is based on the features that give the highest information gain, while our approach iteratively partitions the value ranges of the features in a hierarchical order to reach unique outcomes.

Next, we present a comprehensive review of the literature that includes various applications of ML algorithms in OR and methodologies proposed to interpret ML predictions. In addition, we organize our discussions of the reviewed literature on the basis of their implementation approaches.

2.1 ML for optimization

This section discusses various implementations of ML in OR, classified by their applications: solution prediction and improvement in the solution search strategy.

Numerous studies have utilized neural networks to predict solutions to optimization problems. Vinyals et al. (2015) present pointer networks within an encoder-decoder (ED) framework for solving combinatorial optimization problems such as the traveling salesperson problem (TSP) and the planar convex hull problem, suitable for problems where the output size depends on the input sequence size. (Yilmaz and Büyüktahtakın 2023, 2024b) use recurrent neural networks (RNNs) and ED approaches to predict binary variables in capacitated lot-sizing problems and optimal solutions to sequential decision-making problems, respectively. **?** extend the ED prediction framework to learn solutions to multi-stage stochastic programs. Pan et al. (2022) propose a Feasibility-Optimized Deep Neural Network approach for AC optimal power flow problems. Bello et al. (2016) present a framework that combines neural networks and RL for TSP, optimizing the parameters of RNN through a policy gradient method with a negative tour length as a reward signal. Yilmaz and Büyüktahtakın (2024a) develop a DRL framework to solve two-stage scenario-based stochastic programs with a multi-agent structure.

Dimitris and Bartolomeo (2022) present a neural network-based approach to solve mixed-integer

programs by predicting the appropriate solution search strategies. Hottung et al. (2020) propose Deep Learning Heuristic Tree Search (DLTS), a novel method that uses DNNs to learn customized solution strategies and lower bounds for the container pre-marshaling problem (CPMP) solely by analyzing existing near-optimal solutions. Similarly, Kallestad et al. (2023) use a DRL framework to enhance existing hyperheuristics with a data-driven heuristic selection module for parameter-controlled low-level heuristics involving combinatorial optimization problems with uncertainties.

Traditional ML algorithms such as K-nearest neighbor (KNN) were also used by Liu et al. (2024) to predict whether a dynamic programming-based transition should be considered in the heuristic, improving its efficiency in solving the classic lot streaming and scheduling problem. Zhang (2017) propose a two-phase heuristic approach to the capacitated vehicle routing problem. In Phase I, a density-based clustering algorithm identifies sets of feasible and cost-effective clusters. In Phase II, the max-min ant system assigns clusters to vehicles and sequences them on each tour.

2.2 Interpreting ML algorithms

An increasing number of studies have emphasized the importance of explaining the patterns delineated by ML algorithms. Providing clear and understandable explanations is critical for building user trust and ensuring transparency, especially when using ML to improve the analytical tractability of complex models, such as PAMs in decision-making processes.

Gilpin et al. (2018) define the effectiveness of an explanation based on the "interpretability" and "completeness." Interpretability refers to providing simple explanations that are easily understood by humans, while completeness refers to the ability to accurately describe the operation of a system. Our solution prioritizes both interpretability and completeness.

The standard approaches, such as Local Interpretable Model-agnostic Explanations (LIME) and Shapley Additive Explanations (SHAP), do not completely explain the ML predictions (Le et al. 2022, Dieber and Kirrane 2020). Although Ramon et al. (2024) have improved on this employing a Classification and Regression Trees (CART) decision tree with Shapley values to extract global explanations from RF models, our methodology further advances the explainability of ML-driven results. Using a hierarchical tree model, our approach merges local and global explanatory mechanisms, capturing complex data interactions, while maintaining a high level of interpretability. This allows for detailed explanations for PAM solutions, making them accessible to individuals without expertise in ML or OR, while ensuring fidelity to the original model.

Haddouchi and Berrado (2019) use visualization tools to enhance the interpretability of the predictions, while Petkovic et al. (2018) generate summary reports. However, we believe that building a single hierarchical tree that captures global patterns while remaining simplicity for human comprehension offers a more effective approach, which we have incorporated into our methodology.

2.3 Key contributions

To the best of our knowledge, this is the first study to present a novel ML-based methodology to build an interpretable heuristic to predict solutions for PAM. Specifically, we develop an ensemble heuristic (EH) that captures global solution patterns and provides concise explanations for each prediction. Our approach uses an adapted Random Forest (RF) model, which involves extracting prediction rules from the RF and identifying broader patterns to better understand the relationships between inputs and optimal PAM solutions. This innovative methodology refines the extensive list of rules within the RF model by partitioning feature value ranges and aligning intersecting rules with these partitions, resulting in clear and interpretable solution strategies. Furthermore, we address the potential issue of infeasibility in predicted solutions, determined by the monotonicity constraints of the optimization model, by providing alternative feasible solutions.

We demonstrate the generalizability of our EH by evaluating its performance across various problem sizes, despite being initially developed for a specific problem size. We rigorously benchmark the computational performance of the EH against the state-of-the-art commercial solver, Gurobi version 11.0.1, focusing on the optimality gap and the solution time. Our results show significant speed improvements, achieving solution speeds of 28 to 69 times faster than Gurobi while maintaining a low optimality gap of $1 - 2\%$ across all test sets, ensuring that all predicted solutions remain feasible.

In addition, we compare the performance of our EH with other established heuristics and ML approaches, highlight the trade-off between the optimality gap, infeasibility, solution time, and interpretability. The proposed heuristic formulation methodology holds potential for applications where problems involving PAMs with slightly varying inputs are repeatedly solved, such as in various contexts between multiple entities.

Finally, we provide an improved formulation for the TBR model presented in Chen et al. (2024). This reformulation significantly enhances computational efficiency by eliminating the need to enumerate all possible decisions, a process that can be computationally expensive. This refined formulation has broader implications, as it can be extended to other PAMs with discrete decision variables that require the enumeration of all possible solution combinations.

3 METHODOLOGY

This section outlines the methodological approach used in this study. We begin by presenting an efficient re-formulation of the treatment-based reimbursement (TBR) model initially proposed by Chen et al. (2024). This re-formulation significantly enhances computational efficiency, enabling scalability to larger problem instances. Specifically, our re-formulated TBR model achieves scalability up to 200 trees with solution times of just a few CPU seconds, a substantial improvement over the original model established by Chen et al. (2024). The original model exhibits exponential growth in computational complexity with an increasing number of trees, requiring over two hours to solve for instances with just 10 trees. This significant improvement in computational efficiency

not only accelerates the computational process, but also enhances the practical applicability of the model in real-time decision-making scenarios. Next, we detail the re-formulation, the training of the ensemble models, the interpretation of the predicted solutions from the ensemble models, and the development of our ensemble heuristic.

3.1 An efficient re-formulation of the TBR model

Recall in the TBR model of Chen et al. (2024), the infestation level, i.e., the number of infested trees (i) is not verifiable, while the number of treated trees for each infestation level ($q(i)$) is. Consequently, the vector of treated trees for each infestation level ($\mathbf{q}$) becomes a set of decision variables. To overcome the computational challenges associated with enumerating all possible combinations of treatment decisions, our re-formulation introduces auxiliary binary variables. This re-formulation drastically improves solution time.

The notation used in the re-formulation is detailed in Appendix A, largely following the convention established in Chen et al. (2024) with the addition of the following auxiliary binary variables:

- $\mu(q(i), i)$: An auxiliary binary variable that equals 1 if $q(i) \geq i$ and 0 otherwise.
- $\bar{\mu}(q(i), i)$: An auxiliary binary varaible defined as $\bar{\mu}(q(i), i) = 1 - \mu(q(i), i)$.
- $\omega_{i,j}$: An auxiliary binary variable that equals 1 if $q(i) = j$ and 0 otherwise. This variable represents the assignment of specific treatment levels (j) to different infestation levels (i).

In addition, the re-formulation uses M as a large positive number and m as a small positive number, where $0 < m < 1$. These constants ensure the proper functioning of the auxiliary binary variables within the constraints of the re-formulated TBR model, which is presented in Eq. (1) below.

$$
\begin{aligned}
\max_{\mathbf{q},\mathbf{r}} \Psi(\mathbf{q}, \mathbf{r}|n) = \; & \textstyle\sum_{i=0}^{n} B(n, \pi, i) \cdot \big[\mu(q(i), i) \cdot \big(s \cdot w_1 - \gamma \cdot d_1\big) + \bar{\mu}(q(i), i) \cdot \big(s \cdot w_0 - \gamma \cdot d_0\big) \\
\text{s.t.} \quad & \qquad\qquad -r(q(i))\big] \\
& \Phi(\mathbf{q}, \mathbf{r}|n) \geq \Phi(a_0|n) && && \text{(IR)} \\
& \phi(q(i)|n, i) \geq \phi(j|n, i) && \forall\, i, j \in [0, n] && \text{(IC}_{ij}\text{)} \\
& r(j) \geq r(j-1) && \forall\, i, j \in [1, n] && \text{(MON}_j\text{)} \\
& r(j) \geq 0 && \forall\, i \in [0, n] && \text{(NN}_j\text{)} \\
& q(i) = \textstyle\sum_{j=0}^{n} \omega_{i,j} \cdot j && \forall\, 0 \leq i \leq n && \text{(TR}_i\text{)} \\
& \textstyle\sum_{j=0}^{n} \omega_{i,j} = 1 && \forall\, 0 \leq i \leq n && \text{(DE}_i\text{)} \\
& r(q(i)) = \textstyle\sum_{j=0}^{n} \omega_{i,j} \cdot r(j) && \forall\, 0 \leq i \leq n && \text{(RE}_i\text{)} \\
& q(i) \geq i - M \cdot \mu(q(i), i) && \forall\, 0 \leq i \leq n && \text{(MU}_i - 1\text{)} \\
& q(i) \leq i + M \cdot \bar{\mu}(q(i), i) && \forall\, i = 0 && \text{(MU}_i - 2\text{)} \\
& q(i) \leq i + M \cdot \bar{\mu}(q(i), i) - m && \forall\, i \in [1, n] && \text{(MU}_i - 3\text{)} \\
& \omega_{i,j}, \quad \mu(q(i), i) \in \{0, 1\} && \forall\, i, j \in [0, n] && \text{(BI}_{ij}\text{)}
\end{aligned}
\tag{1}
$$

where

$$B(n,\pi,i)=\tbinom{n}{i}\pi^i\bar{\pi}^{n-i},\quad w_1=\rho q(i)+(\bar{\pi}^h+\rho\pi^h)(n-i),\qquad d_1=i-\rho q(i)+\bar{\rho}\pi^h(n-i),$$
$$w_0=q(i)-\bar{\rho}i+(\bar{\pi}^l+\rho\pi^l)(n-q(i)),\quad d_0=\bar{\rho}i+\bar{\rho}\pi^l(n-q(i)),$$
$$\Phi(a_0|n)=\bar{\pi}^n\cdot\Big(\theta\bar{\pi}^l n-c\pi^l n\Big)+\textstyle\sum_{i=1}^{n}\binom{n}{i}\pi^i\bar{\pi}^{n-i}\cdot\Big(\theta\bar{\pi}^h(n-i)-c(\pi^h(n-i)+i)\Big)$$
$$\Phi(\mathbf{q},\mathbf{r}|n)=\textstyle\sum_{0}^{n}\binom{n}{i}\pi^i\bar{\pi}^{n-i}\cdot\Big[\mu_i\cdot\phi\Big(q(i),r(q(i))|n,i,q(i)<i\Big)+\bar{\mu}_i\cdot\phi\Big(q(i),r(q(i))\Big|n,i,q(i)\geq i\Big)\Big],$$
$$\begin{aligned}\phi\Big(q(i),r(q(i))\Big|n,i,q(i)\geq i\Big)=\;&\theta\cdot\big[q(i)-\bar{\rho}i+\bar{\pi}^l(n-q(i))+\rho\pi^l(n-q(i))\big]+r(q(i))-\alpha\cdot n\\&-\beta\cdot\big[q(i)+\pi^l(n-q(i))\big]-c\cdot\big[\bar{\rho}i+\bar{\rho}\pi^l(n-q(i))\big],\\\phi\Big(q(i),r(q(i))\Big|n,i,q(i)<i\Big)=\;&\theta\cdot\big[\rho q(i)+\bar{\pi}^h(n-i)+\rho\pi^h(n-i)\big]+r(q(i))-\alpha\cdot n\\&-\beta\cdot\big[q(i)+\pi^h(n-i)\big]-c\cdot\big[i-\rho q(i)+\bar{\rho}\pi^h(n-i)\big].\end{aligned}$$

The city forester aims to determine the optimal schedule of treatment decisions ($\mathbf{q}$) and their corresponding reimbursements ($\mathbf{r}$) to maximize her net expected utility ($\Phi(\mathbf{q},\mathbf{r}|n)$), as defined in the TBR formulation's objective function (1). The first four constraints are identical to those in the TBR model of Chen et al. (2024). These constraints incentivize the participation of the landowner (IR); align the landowner's treatment decisions with the city forester's objective (IC_{ij}); guarantee the reimbursements are non-decreasing with respect to number of trees treated (MON_j); and ensure that reimbursements are non-negative (NN_j).

The remaining constraints in formulation (1) are new to our re-formulation. Using the binary variable $\omega_{i,j}$, Constraint TR_i and DE_i ensure that exactly one treatment decision, ($q(i)=j$, where $j\in J$), is selected for each infestation level i. When a specific treatment decision j is selected, $\omega_{i,j}$ is set to 1, and consequently the correct reimbursement ($r(j)$) is selected. The set of constraints (MU_i) ensures that the binary variables $\mu(q(i),i)$ correctly reflect the relationship between $q(i)$ and i: $\mu(q(i),i)=1$ if $q(i)<i$ and 0 otherwise.

The approach presented by Chen et al. (2024) iteratively determines the optimal treatment decisions (OTDs) by evaluating each possible combination of OTDs for all infestation levels. For each combination, the values of $q(i)$ are fixed, and the combination that maximizes the objective function is selected. Our re-formulation, as expressed in (1), introduces the binary variable $\omega_{i,j}$. This change significantly improves computational efficiency by replacing the integer decision variable $q(i)$ with a linear combination of binary variables, each weighted by an integer constant j. Similarly, the decision variable $r(q(i))$, which is contingent on $q(i)$, is reformulated into a linear combination of binary variables, each scaled by $r(j)$ variables, allowing the model to optimize the values of q and r directly, without the need for the iterative process of fixing q-values and re-solving multiple optimization problems, as employed by Chen et al. (2024). Our computational experiments, which compared solution times for approximately 2000 instances of the TBR model with 5 trees, demonstrated a drastic improvement in computational efficiency. The re-formulation improved the average solution time by a factor of 150 compared to the original approach.

3.2 Training the ensemble models

We train the ensemble models on a data set that includes the modified parameter values derived from the TBR model and the optimal solutions of numerous solved instances using the efficient re-formulation in (1). The training dataset consists of a combination of input parameters, as detailed in Table 1. Instead of directly using the raw input parameters, we utilize a transformed set of input features. For example, for input parameters θ, c, s and ρ, we use the combined forms of $\rho \cdot (\theta$+c), $\rho \cdot (\theta$+c+s) and $\rho/\overline{\rho}$. The first two combinations are derived from the analytical framework established by Chen et al. (2024) to classify the optimal solutions of the TBR model. Among the numerous combinations of parameters tested, $\rho/\overline{\rho}$ showed the highest predictive power. Other input parameters are used in their original form as features for training the ensemble models. This approach better leverages the modified input parameters to capture the integral relationships in the optimization model.

We conduct a rigorous hyper-parameter tuning process using grid search. Among the evaluated hyper-parameters, n_estimators, max_depth, and criterion demonstrated a significant impact on performance, as measured by the unweighted F1 score, during the initial training iterations. In subsequent iterations, including those involving feature engineering, we explore an exhaustive range of values for these key hyper-parameters to ensure optimal model performance. Each instance in the training dataset is generated using parameter values derived from the individual criteria mentioned in the "Input Criterion" column of Table 1. In addition, for each instance all possible solutions for all levels of infestation are considered.

Table 1: Input Parameters for Instance Generation

Para.	**Description**	**Input Criterion**
π	The initial (or first) period's attack rate, i.e., likelihood of an ash tree infested with EAB	Discrete: U[0.2, 0.9], step size = 0.1
α	Cost of inspecting an ash tress	Continuous: U[30,60]
β	EAB treatment cost for an ash tree	Continuous: U[150, 500]
ρ	Probability of treatment success	Discrete: U[0.2, 0.8] , step size = 0.1
θ	Marginal value of a healthy tree to landowner	Continuous: U[25, 500]
s	Marginal value of a healthy tree to city forester	Continuous: U[50,250]
π^l (π^l)	likelihood of a healthy tree in one period becomes infested in the next period being high (low)	Discrete: U[0,π], step size = 0.1 (Discrete: U[π,1], step size = 0.1)
γ	Removal cost of an ash tree	Continuous: U[50,150]
f	Ratio of c/β	Discrete: U[3,10], step size 0.5
c	Penalty cost of a deceased ash tree	$c = f \cdot \beta$

To generate training instances for our ensemble models, we consider a scenario with trees $n = 5$, reflecting the limited number of ash trees that are typically found in private properties. Solving the TBR model for this scenario with 5 trees takes an average of 1.036 CPU seconds. The resulting OTDs for an individual set of inputs across all possible levels of infestation are discretized and grouped into representative patterns. This clustering approach, inspired by Chen et al. (2024), com-

bines the results of multiple levels of infestation into characteristic sequences, effectively capturing common action patterns. ML algorithms trained using clustered results show a prediction accuracy more than 10% higher than the same algorithms when trained on data sets with unclustered individual character outcomes.

The character representations of the OTDs are shown in Table 2. Let n denote the total number of ash trees on private property, i represent the infestation level, and q denote the prescribed OTD. Table 3 illustrates an example of a cluster structure. The characters N, S, I, P, and A represent prescribed treatment levels of treat zero trees, treat some (but not all) infested trees, treat all infested trees, treat all infested trees and some healthy trees, and treat all trees, respectively. The subscripts and superscripts associated with each character define the lower and upper limits of the infestation levels for which the corresponding treatment level is the same. Table 3 shows an example in which the optimal OTDs fall into the group $N_0^0 I_1^j A_{j+1}^n$, where $n = 5$ and $j = 2$. This implies that it is optimal to treat no trees when none are infested, treat all infested trees when the level of infestation (i) is between 1 and j and treat all trees when i exceeds j.

Table 2: Character Representations of OTDs

OTD (q)	Char.	Interpretation
$q = 0$	N	Treat no trees
$0 < q < i$	S	Treat some (not all) infested trees
$q = i$	I	Treat all infested trees
$i < q < n$	P	Treat all infested tres and some healthy trees
$q = n$	A	Treat all trees

Table 3: Cluster $N_0^0 I_1^j A_{j+1}^n$

Inf. Lev. (i)	OTD (q)	Char.
0	0	N
1	1	I
2	2	I
3	5	A
4	5	A
5	5	A

The generated data set has 35 unique outcomes, 16 of which are very infrequent, occurring in less than 0.02% of the total instances. To address this imbalance, each of these 16 infrequent outcomes was replaced by one of the 19 remaining high-frequent clusters. Replacement was determined based on the cluster that resulted in the minimum trade-off in the optimality gap.

3.3 Interpreting prediction strategies of Random Forest models

Random Forest (RF) models offer a degree of inherent interpretability due to their ability to extract simple and meaningful rules that can be easily understood and applied (Miraboutalebi et al. 2016). To extract the prediction strategies/rules used by the trained RF model, we use "inTrees" R-studio package proposed by Deng (2019). This package is designed to extract the conditions within each individual branch of the tree ensembles and their corresponding label outcome. The combined set of conditions on the features from the root node to the leaf node constitutes an individual rule. Table 4 presents a breakdown of the specific rules extracted from our model and their associated clusters. The 'Popularity' column quantifies each rule's prevalence within its cluster, calculated as the proportion of instances that satisfy the rule relative to all instances in that cluster. This metric helps us assess the importance and generalizability of each rule within its

respective cluster, providing a foundation for developing a simple and interpretable heuristic based on the most prevalent rules.

Table 4: Strategy Clustering and Popularity

Rule	**Cluster Label**	**Popularity**[a]
$\rho/\bar{\rho} > 0.280$ & $\pi^l \leq 0.150$	$N_0^0 I_1^{n-1} A_n^n$	0.55
$\rho/\bar{\rho} \leq 0.686$ & $\pi^l > 0.450$	$A_0^j N_{j+1}^n$	0.28
$\pi^l \leq 0.210$ & $\pi^h \leq 0.270$ & $\pi^h > 0.210$	$N_0^0 I_1^j N_{j+1}^n$	0.31
$\beta > 297.156$ & $\beta \leq 452.821$ & $\rho \cdot (\theta + c) \leq 888.707$ & $\pi^l > 0.31$ & $\pi^h > 0.475$	$N_0^0 I_1^j N_{j+1}^n$	0.11
$\rho/\bar{\rho} > 0.234$ & $\rho \cdot (\theta + c) > 1740.860$ & $\pi^l > 0.270$	A_0^n	0.15

[a] 'Popularity' is defined as the ratio of the number of instances within a cluster that satisfy a specified rule to the total number of instances within that cluster.

3.4 Developing the ensemble heuristic

Our ensemble heuristic (EH) adopts the hierarchical architecture, mirroring the inherent structure of decision trees. This approach categorizes the spectrum of feature values to effectively discern distinctive outcomes. This methodology encapsulates and assimilates predictive rules into a well-defined hierarchical order. Prioritization within this structure is informed by feature importance analysis performed on a trained random forest model, positioning the parameter of utmost significance at the top of the hierarchy, succeeded by parameters of decreasing importance.

To address the complexity of the extensive rule set extracted from the RF model, our approach simplifies interpretation by selecting a subset of rules for each output label. This subset has 99% of the data points within their respective label, capturing the broader patterns that can predict the output while excluding the extensive set of rules that explain the remaining 1% of the data, which often represent localized trends. The comprehensive set of these selected rules ($\mathcal{R}_p$) and their respective outcomes forms the foundation of our EH.

The process of constructing the EH is encapsulated within Algorithm 1. This process begins by selecting the first feature in $\mathcal{F}$ and its corresponding *partition_inputs*, a set of input parameters for partitioning the feature space. Subsequently, Algorithm 2 is invoked, which divides the value range of the selected feature into equal-sized intervals based on the provided input limits and interval size. Each interval is then associated with rules that intersect its value range. The details of the algorithms with an example are given in Appendix B.

4 IMPLEMENTATION AND EXPERIMENTATION

We begin this section by detailing the computational platform utilized for all experiments. Subsequently, we provide a comprehensive description of the computational experiments conducted using our ensemble heuristic (EH) and other comparative methods. Lastly, we describe the evaluation metrics used to measure performance.

4.1 Computational platform

The process of generating instances, training, and testing ML models was carried out using the Advanced Computing Resources (ARC) infrastructure at Virginia Tech. ARC has a high-performance computing cluster based on the Linux platform, equipped with an AMD Epyc 7742 processor running at 2.245 GHz and 64 GB of memory.

For instance generation, Python 3.9.17 was used in conjunction with the Gurobi solver version 11.0.1 build v10.0.2rc0 (mac64[arm]) (Gurobi Optimization 2023), accessed through the 'gurobipy' library API under an academic license. This setup facilitated efficient optimization and instance creation. ML models were trained using the sci-kit-learn version 1.3.0 Python package.

4.2 Computational experiments

We designed experiments to evaluate the generalizability of our ensemble heuristic (EH) and to benchmark its performance against the following heuristics:

- **Random Forest (RF) Model**: The RF model used to build the EH
- **Decision Forest (DF) Model**: A novel approach proposed by Sagi and Rokach (2020), which builds an interpretable decision tree from RF models.
- **Bertsimas Heuristic (BH)**: A relax-and-fix heuristic for solving combinatorial problems, proposed by Bertsimas and Demir (2002).
- **Chen Heuristic (CH)**: A partly analytical approach proposed by Chen et al. (2024) for predicting TBR solution clusters.

To evaluate generalizability, we train our heuristic on solved instances with a problem size of 5 trees. Subsequently, we evaluate its performance on solved instances with varying problem sizes. Similarly, we evaluate the performance of the RF and DF methods using this approach. In addition, we compare the performance of all heuristics across test sets using the same evaluation metrics.

4.3 Evaluation metrics

This section presents performance evaluation metrics designed to assess the feasibility, optimality gaps, and interpretability of the predicted solutions.

- **Accuracy (%)**: Measures the proportion of solutions predicted or computed by the heuristic that are optimal, given the set of input parameters.
- **Infeasibility (%)**: Indicates the proportion of solutions predicted by the heuristic that are infeasible for the corresponding inputs.
- **Interpretability**: Describes the capacity to explain or provide meaning in understandable terms to a human, as per the definition by Arrieta et al. (2020). We quantify interpretability by the number of unique features required to explain a prediction reflecting the heuristic's ability to elucidate its decisions Molnar (2020), Miller (2019). A concise elucidation of a decision also reflects the heuristic's effectiveness in selecting only input features with strong predictive power.

- **OptGap(%)**: Measure the optimality gap, similar to that defined by Yilmaz and Büyüktahtakın (2023). Let y^* (resp. $\hat{y}^*$) denote the optimal solution of the original instance (resp. heuristic-predicted solution) and let $Z(y^*)$ (resp. $Z(\hat{y}^*)$) represent the corresponding objective function value. The optimality gap is calculated as follows:

$$\textbf{OptGap}(\%) = \frac{|Z(\hat{y}^*) - Z(y^*)|}{|Z(y^*)|} \times 100. \tag{2}$$

- **timeGRB**: The average solution time of an optimization problem in CPU seconds, calculated using the Gurobi solver in its default setting.
- **timeMLheur**: The average CPU seconds required by the heuristic to make a prediction.
- **timeGRBresolve**: The average solution time of an optimization problem after fixing the predicted solutions, in CPU seconds, using the Gurobi solver in its default setting.
- **TimeImpMLheur**: The factor of improvement in the solution time attained by the heuristic predictions with respect to the default Gurobi solver is given by

$$\textbf{TimeImpMLheur} = \frac{timeGRB}{timeMLheur + timeGRBresolve}. \tag{3}$$

5 COMPUTATIONAL RESULTS

We begin this section by providing an illustrative example of the decision tree, the output of the ensemble heuristic (EH), highlighting the hierarchical significance of the input parameters and the iterative partitioning process. Next, we present a comprehensive evaluation of the generalizability and performance of our EH across various datasets. We benchmark its efficiency against traditional solvers and other heuristic methods. Through this examination, we demonstrate the EH's ability to significantly reduce computational time while maintaining solution feasibility and discuss the trade-offs associated with achieving greater interpretability in predictions.

5.1 Illustrative Example of the Decision Tree

First, we present a snippet of the output tree structure obtained by applying our EH to the TBR model based on the instances generated using the schema in Table 1.

As shown in Figure 1, the tree starts with a root node, which represents the feature with the highest importance according to the RF model. Among the input parameters, π^l is identified as the most important, followed by $\rho/\overline{\rho}$, β, $\rho \cdot (\theta + c)$, $\rho \cdot (\theta + c + s)$, γ, α, and π^h in decreasing order of importance. The dashed lines in the second and fourth branches that connect to node π^h represent a redacted illustration of the branches and nodes, skipping the features between π^h and its preceding nodes in the illustration. The dotted lines represent other branches in the ensemble heuristic, which are not detailed in the illustration.

Starting from the root node π^l, the value ranges of each feature are iteratively partitioned to segregate intersecting rules, thus clustering rules with similar results. Each partitioned value range creates a new branch that leads to a child node at the next hierarchy level. At each node, the range

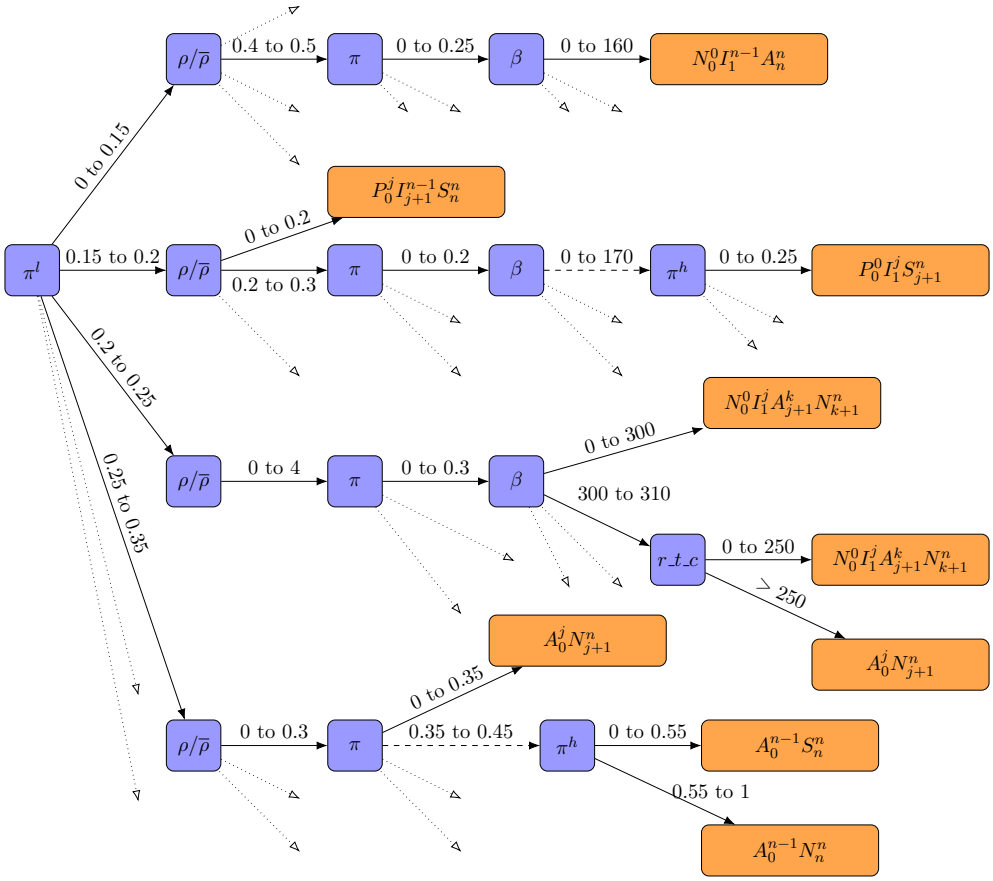


Figure 1: Illustration Example of the Decision Tree, the output of EH. Note: r_t_c represents $\rho \cdot (\theta + c)$.

of values of the corresponding feature is partitioned, with the resultant partitions varying across nodes. This iterative partitioning continues until a unique outcome is achieved. The outcome $N_0^0 I_1^{n-1} A_n^n$ in the top branch of Figure 1 shows a case where this is achieved with 4 features, while the outcome $P_0^0 I_1^j S_{j+1}^n$ illustrates a case where all input features were used to arrive at a unique outcome.

5.2 Experiment results

Next, we present results that demonstrate the generalizability and interpretability of our ensemble heuristic (EH). We benchmark its performance against various other heuristics using the aforementioned metrics, while also discussing the interpretability of these approaches. Our EH exhibits high interpretability, on average using conditions from only 4.5 out of 9 input features. This performance surpasses other approaches because some of which lack interpretability, while others require significantly more conditions to explain their predictions. For example, DF, built as an interpretable method, uses an average of more than 12 conditions.

To perform these evaluations, we create five testing datasets with varying problem sizes, that is, the number of trees. The test set with 5 trees has about 0.78 million instances, while the other test sets have about 0.52 million instances each, resulting in a total of 2.86 million test instances. We predict

solutions for a set of decision variables corresponding to treatment decisions for the TBR model. Using our heuristic, the time to generate predictions while ensuring feasibility is significantly faster compared to solving the full problem using the Gurobi solver. We measure the improvement in solution time by calculating the ratio of CPU seconds required by each heuristic method to the CPU seconds required by Gurobi.

Furthermore, we compare the performance of our ensemble heuristic (EH) with Random Forest (RF), Bertsimas Heuristic (BH), Decision Forest (DF), and Chen Heuristic (CH), as described in Section 4.2. Compared with RF, we show the trade-offs made in performance to enhance interpretability. Furthermore, we benchmark our performance against BH since BH is shown to provide decent performance for similar mixed-integer programming (MIP) formulations, such as the multi-dimensional knapsack problem (Yilmaz and Büyüktahtakın 2024b). Compared with DF, we demonstrate the effectiveness of our methodology in creating a simple and interpretable tree with rules extracted from RF models. Lastly, compared to CH, we show the effectiveness and limitations of using ML algorithms to delineate solution patterns.

5.2.1 Results on generalizability and interpretability of our ensemble heuristic

In Table 5, we present the results of the ensemble heuristic for all metrics, defined in Section 4.3. The model is trained only for problem size 5 but is used to predict solutions for instances with problem sizes 3 to 7. This shows that the ensemble heuristic can effectively predict solutions for various problem sizes if it captures the broader trends accurately. The test instances with 5 trees are solved with a mean solution time of 1.04 seconds with the Gurobi solver. Across all test sets, the average time to make predictions and ensure feasibility is significantly less than the time required by exact methods. For the test set with problem size 5, the average optimality gap is about 1.03%, as shown in Table 5. This low optimality gap was achieved with 0% infeasibility. This was possible with the provision in the EH to provide alternative solutions to infeasible predictions by exploiting the monotonicity constraints on the reimbursement amounts for all treatment decisions of the TBR model. It should be noted that the solution A_0^n ensures that all trees are treated, providing the flexibility to set the reimbursement values for other treatment decisions to any non-decreasing value in the monotonicity constraint (MON_j), making it a universally feasible solution structure.

Table 5: Performance Metrics for EH

# of trees (n)	**3**	**4**	**5**	**6**	**7**	**Average**
Accuracy (%)	96.32%	96.38%	97.84%	95.62%	95.69%	96.37%
Infeasibility (%)	0%	0%	0%	0%	0%	0%
OptGap (%)	1.12%	0.96%	1.03%	1.99%	1.21%	1.26%
TimeimpMLheur	27.9	31.2	36.0	51.4	68.9	43.1
Interpretability	4.5	4.4	4.4	4.4	4.4	4.4

Predictions for instances with different problem sizes yielded average optimality gaps between 0.96% to 2%, with 0% infeasibility. Furthermore, for all problem sizes, the total CPU-time, including

prediction and resolving using our EH, is significantly less than the Gurobi solution time. The improvement factors in the solution time increase as the size of the problem increases. For example, the improvement factor for the set with problem size 3 is about 28, which increases monotonically, achieving an improvement factor of about 69 for the set with problem size 7. As mentioned earlier, we use the distinct number of features as a quantitative measure to facilitate user interpretation. On average, our EH uses 4.5 features to obtain a prediction. However, the median use is only around 3 features, with 90% of test instances using no more than 6 features. In contrast, the TBR model requires 9 input parameters. This result demonstrates the effectiveness of the ensemble heuristic in improving the interpretability of ML predictions over exact or other heuristic approaches.

5.2.2 Performance of Random Forest (RF)

In Table 6, we present the performance metrics of the RF model, which incorporate an ensemble of 150 estimators. Similarly to our heuristic approach, the RF model is trained on instances with 5 trees and tested on problem sizes ranging from 3 to 7. The performance of RF is better than that of our heuristic with respect to accuracy, improvement of solution time, and optimality gap. This difference is expected due to the more comprehensive set of prediction rules that RF uses compared to our heuristic. Although our heuristic captures broader trends to predict individual outcomes, it excludes minor trends to improve interpretability. The difference in optimality gap between the RF and our heuristic reflects the trade-off between interpretability and performance. Our heuristic is more interpretable, featuring an average of about 4.5 conditions, whereas the Random Forest is not interpretable at all. In addition, RF shows a higher infeasibility in all test sets, as it lacks the provision to provide alternate solutions for infeasible predictions.

Table 6: Performance Metrics for RF*

# of trees (n)	**3**	**4**	**5**	**6**	**7**	**Average**
Accuracy(%)	97.66%	97.63%	99.89%	96.94%	96.83%	97.79%
Infeasibility(%)	0.15%	0.23%	0.002%	0.17%	0.20%	0.15%
OptGap(%)	0.84%	0.37%	0.09%	0.85%	0.56%	0.54%
TimeimpMLheur	64.4	71.2	64.8	100.0	113.2	82.71

*RF has an average tree depth of 31 conditions across estimators.

5.2.3 Performance of Decision Forest (DF)

In Table 7, we present the performance metrics of the DF implementation across various test sets. The results show that the accuracy is significantly lower than that of our EH. Furthermore, the average infeasibility is about 5% in all test sets, which can be partly attributed to the low precision. This performance drop may arise from the inherent constraints on the depth of individual trees in the trained ensemble model from which the rules are extracted. Sagi and Rokach (2020) impose restrictions to manage algorithmic complexity, which could impede the model's ability to effectively capture both broader and local trends. However, the methodology demonstrates a notable improvement in prediction speed, resulting in solution time improvement factors between

51 and 113, compared to using the Gurobi solver. This improvement is higher than the EH in all test sets. Furthermore, the optimality gap for the predictions ranges from 9% to 15% across the test sets, which is partly due to the lower accuracy of the predictions.

Table 7: Performance Metrics for DF

# of trees (n)	**3**	**4**	**5**	**6**	**7**	**Average**
Accuracy (%)	89.60%	89.80%	89.14%	88.96%	88.74%	89.25%
Infeasibility (%)	4.89%	4.86%	5.01%	4.97%	5.26%	5.0%
OptGap (%)	12.49%	8.90%	15.43%	13.30%	12.15%	12.45%
TimeimpMLheur	51.7	57.4	65.5	93.2	113.4	76.25
Interpretability	12.36	12.4	12.35	12.45	12.35	12.38

5.2.4 Performance of Bertsimas Heuristic (BH)

Table 8 shows the performance metrics of BH. The prediction accuracy is generally lower than EH, except for the highlighted set. Infeasibility is negligible in all test sets. However, the optimality gap is about 2-6% higher than our heuristic except for the highlighted test set. Furthermore, the solution time of BH is lower using the Gurobi solver across all test sets. Given the relax-fix-and-optimize nature of BH, the interpretation of the solutions cannot be achieved.

Table 8: Performance Metrics for BH

# of trees (n)	**3**	**4**	**5**	**6**	**7**	**Average**
Accuracy (%)	94.76%	97.31%	**99.29%**	94.92%	95.12%	96.28%
Infeasibility (%)	∼0	∼0	∼0	∼0	∼0	∼0
OptGap (%)	7.98%	**0.83%**	**1.03%**	6.36%	3.53%	3.95%
TimeimpMLheur	0.50	0.46	0.45	0.49	0.38	0.45

5.2.5 Performance of Chen Heuristic (CH)

Table 9 presents the performance metrics for Chen's heuristic (CH). Although the accuracy across all test sets is lower than that of our ensemble heuristic, the CH exhibits a lower optimality gap. These contrasting outcomes arise from CH's limitation in predicting unique and outlier solutions for all cases. Specifically, CH provides unique outcomes for the four most frequent solution clusters but predicts multiple possible solutions for the rest. In many instances, these multiple solutions are either optimal or near-optimal, leading to a lower optimality gap. In contrast, our approach predicts unique answers for any instance, occasionally resulting in suboptimal solutions and a higher optimality gap.

Our heuristic outperforms CH by identifying 19 unique solution structures for the TBR model, whereas CH recognizes only 11. Importantly, CH's solution structures are specific to the TBR model due to its combination of analytical and numerical methods. In contrast, our approach is general and thus can identify unique solution structures for any PAM without requiring adjustments.

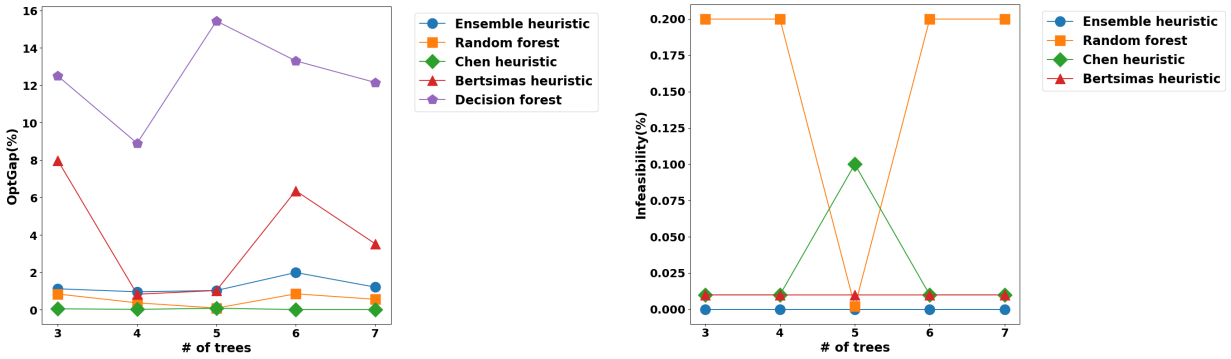
Additionally, while CH generally performs well for most TBR model instances, it fails for certain solution clusters. For instance, CH fails to identify the optimal solution cluster $N_0^0 I_1^j A_{j+1}^k N_{k+1}^n$. Instead, it predicts suboptimal solutions such as $N_0^0 I_1^j A_{j+1}^{n-1} N_n^n$ and $A_0^{n-1} N_n^n$. These predictions lead to significantly higher costs due to overtreatment. As a result, the objective function values of these suboptimal solutions are, on average, approximately 400 times lower than those of the optimal solutions. Fortunately, these suboptimal predictions of CH are limited to specific localized ranges of input parameter values.

Table 9: Performance Metrics for CH

# of trees (n)	**3**	**4**	**5**	**6**	**7**	**Average**
Accuracy (%)	94.55%	95.63%	94.96%	94.82%	95.04%	95.0%
Infeasibility (%)	~ 0	~ 0	0.1%	~ 0	~ 0	~ 0
OptGap (%)	0.05%	0.02%	0.08%	0.01%	0.01%	0.03%
TimeimpMLheur	53.3	55.7	72.7	103.3	111.6	79.3

5.2.6 Performance comparisons of heuristics

Figure 2 compares the performance of various heuristics on solving the TBR model and presents the results in terms of OptGap (%) and Infeasibility (%). Specifically, CH consistently achieves the lowest optimality gap among all methods. RF predictions closely follow CH's performance, while our EH achieves an optimality gap of 1–2%. BH exhibits optimality gaps that are comparable to or higher than our EH, except for a specific test set. As shown in Figure 2(b), EH, CH and BH yield zero or near-zero infeasible outcomes. RF has a slightly higher infeasibility (with a maximum of 0.85%). DF exhibits significantly higher infeasibility (around 5%) and is not included in Figure 2(b) to highlight the performance comparison of the other methods. The predicted solutions are deemed infeasible if they do not meet the monotonicity constraints (MON_j) specified in formulation **??**.



(a) OptGap (%)

(b) Infeasibility (%)

Figure 2: Performance Comparison

5.2.7 Prediction optimality of the EH across parameter value ranges

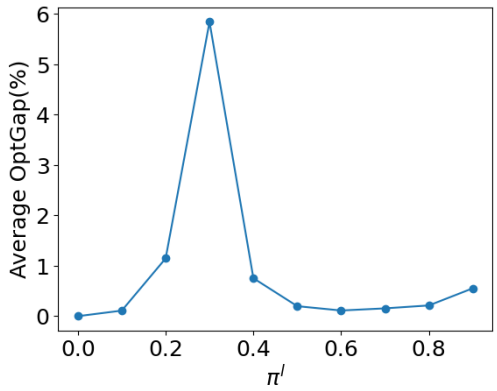
We present the variation in the optimality gap of the predictions of the EH across different input parameter ranges for the test set consisting of instances with 5 trees. As shown in Figure 3, the parameters considered are π^l, $\rho/\bar{\rho}$, π^h, c, and f. Overall, the EH algorithm demonstrates smaller optimality gaps and improved efficiency when treatment is effective, treatment and removal costs are high, and the second-period attack rate is high.

The optimality gap of the predictions varies significantly between the value ranges of all parameters. The optimality gap is higher in specific neighborhoods and drops to consistently low levels in other regions. The variation against π^l is unimodal, where instances with $\pi^l = 0.3$ exhibit a high optimality gap of 6%, which drops to less than 1% for other values of π^l, as shown in Figure 3(a). For the other parameters, the optimality gap is more evenly distributed, with approximately half of the value range showing a high optimality gap, while the other half shows significantly lower gaps. As shown in Figures 3(b), 3(c), and 3(d) the highest optimality gap is between $1-3\%$ for parameters π^l, $\rho/\bar{\rho}$, π^h, c. In contrast, the parameter f has a large gap of 6% in the lower value ranges, which consistently drops to close to 0 with increasing f as shown in Figure 3(e). This indicates that the results are highly sensitive to the π^l values, as confirmed by the feature importance analysis of the trained RF model. This high sensitivity can be attributed to the occurrence of multiple optimal solutions within specific π^l value ranges, which is more spread out for other parameters. As the variation graphs depict the significant variation of OptGap(%) across the value ranges of π^l, π^h, and ρ, it is critical for managers to ensure precise data collection for these parameters. Since these parameters represent probabilities between 0 and 1, they are particularly susceptible to rounding errors.

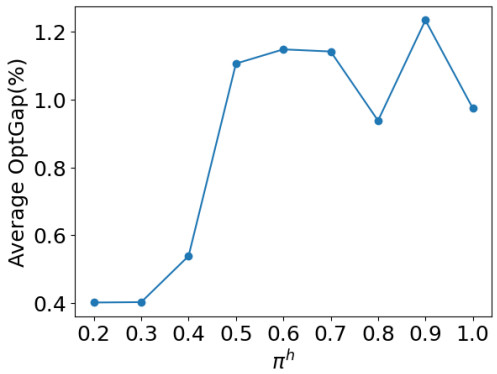
Additionally, we analyze the optimality gap of the predictions for instances with 5 trees when tested on out-of-distribution inputs of the parameter β (EAB treatment cost). The optimality gap is evaluated on two test sets: one with β values between [75, 150], and the other with β values between [500, 600]. The optimality gaps for both test sets are 0.92% and 0.23%, respectively. However, the optimality gap for the solutions predicted for instances with β values greater than 600 increases significantly. This trend highlights the limitations of our ML-based approach, as the drop can be attributed to different solution patterns when the β values exceed the trained range beyond a certain threshold.

6 CONCLUSIONS AND FUTURE WORK

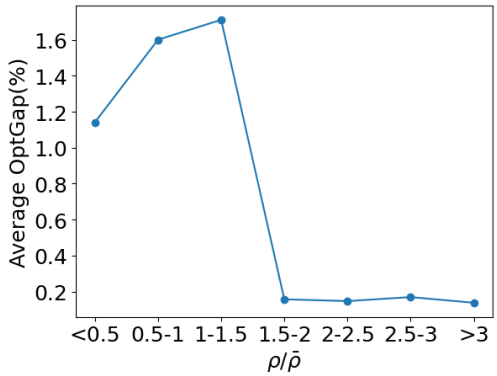
We present a novel Machine Learning (ML) approach to develop an interpretable ensemble heuristic (EH) for a principal-agent model (PAM). The EH significantly enhances prediction interpretability by requiring, on average, conditions on only 4.5 out of the 9 input features to explain its predictions. In comparison, other benchmarked approaches are either entirely non-interpretable or require a significantly larger number of conditions to explain their predictions.



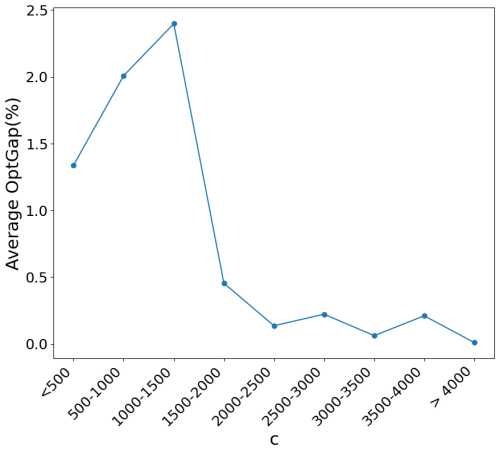
(a) OptGap (%) vs π^l



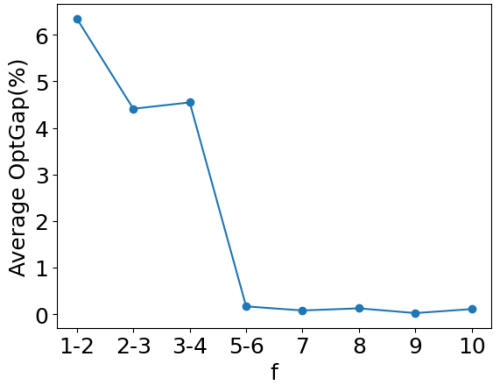
(b) OptGap (%) vs π^h



(c) OptGap (%) vs $\rho/\bar{\rho}$



(d) OptGap (%) vs c



(e) OptGap (%) vs f

Figure 3: Variation of OptGap (%) across value ranges of different input parameters

Furthermore, our approach assists managerial decision-makers by avoiding redundant optimization problem solving when faced with similar problems with slight variations in input. Specifically, we introduce a pattern-delineation methodology for interpreting predictions from ML models. Within this framework, we address infeasible predictions by performing a feasibility check and prescribing

alternative solutions.

Notably, we show that the ensemble heuristic (EH), trained on instances of a particular problem size, can effectively predict solutions for instances with larger or smaller problem sizes. Our computational results reveal a reduction in solution time - up to 60 orders of magnitude - while maintaining an optimality gap of approximately 1–2%.

Our findings demonstrate the effectiveness of our approach, which leverages ML algorithms to infer analytical solutions for an optimization model and subsequently find computationally tractable solutions. Compared to the RF model, our EH achieves high interpretability while maintaining competitive optimality gaps. Furthermore, our approach significantly improves the solution time while achieving low optimality gaps.

The principle-agent model (PAM) formulated by Chen et al. (2024) aims to solve the coordination problem between a city forester and individual landowners in the context of invasive specifies management. Given the recurrent nature of decision-making, solving the optimization model for each instance can be computationally expensive, hindering efficient decision-making. Our study aims to address this challenge by developing a tractable approach that eliminates the need to repeatedly formalize and solve the optimization model, while also providing generalizable analytical explanations, thereby streamlining the decision-making process.

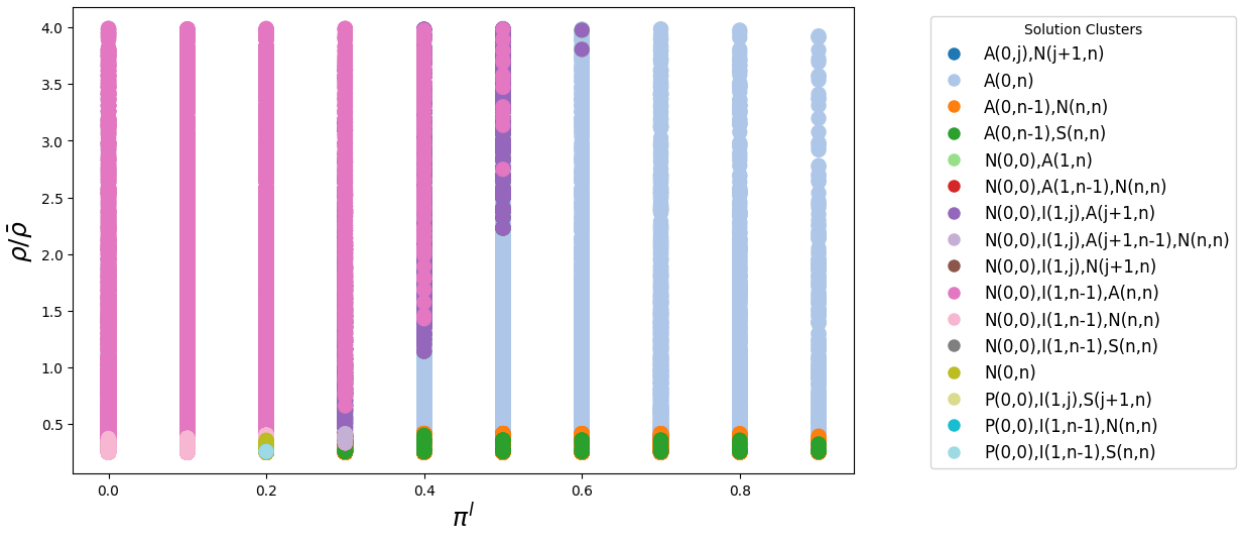


Figure 4: Incidence of Solution Clusters Across Value Ranges of π^l and $\rho/\bar{\rho}$.

Our EH demonstrates the ability to predict solutions with fewer than five inputs for approximately 90% of instances, thereby reducing the extent of data collection required. Furthermore, the discerned patterns can inform the formulation of policies. Figure 4 illustrates the incidence of different solution clusters in the value ranges of π^l and $\rho/\bar{\rho}$ using a maximum of 10,000 sample instances for each solution cluster. When the probability of a healthy tree in one period becoming infested in the subsequent period π^l is low (less than 0.2), the TBR model predominantly prescribes the solution

clusters $N_0^0 I_1^{n-1} A_n^n$ and $N_0^0 I_1^{n-1} N_n^n$. These solutions suggest that the city forester should actively induce the private landowner to treat only infested trees in the first period because the benefit of treating healthy trees in the first period is not substantial. In contrast, when π^l is greater than 0.6, the prescribed solution clusters change to A_0^n, $A_0^{n-1} N_n^n$, and $A_0^{n-1} S_n^n$. These solution clusters collectively advocate for the treatment of all trees, both infested and healthy, to achieve immunization in the subsequent period.

Furthermore, the most frequent clusters, $N_0^0 I_1^{n-1} A_n^n$ and A_0^n, collectively make up more than 95% of the training instances and are prescribed at opposite ends of the value range of π^l. The 14 other less frequent solution clusters, which collectively account for less than 5% of the instances, are prescribed sparsely as shown in Figure 4. However, these less frequent solution clusters within the specific value ranges of π^l are similar to the most frequent ones occurring in that range. For π^l less than 0.2, where $N_0^0 I_1^{n-1} N_n^n$ and $N_0^0 I_1^{n-1} A_n^n$ are prescribed, optimal treatment decisions (OTD) for the majority of infestation levels are the same in both solution clusters, with variations observed only at the highest level of infestation. In this scenario, the strategy changes from treating all trees to zero trees when the odds of treatment success ($\rho/\bar{\rho}$) are closer to zero. Similarly, when π^l is greater than 0.6, $A_0^{n-1} N_n^n$ and $A_0^{n-1} S_n^n$ are prescribed along with A_0^n. Here, OTDs are the same for most infestation levels, differing only at higher infestation levels. These less frequent solutions typically occur when $\rho/\bar{\rho}$ is close to zero in this value range of π^l.

The value range of π^l between 0.2 and 0.6 represents a transitional range in which the predominant solution shifts from $N_0^0 I_1^{n-1} A_n^n$ to A_0^n as π^l increases. This range also exhibits the highest incidence of the less frequent solutions, reflecting the broader transition in the solutions. The increase in the frequency of incidence of solution clusters $N_0^0 I_1^j A_{j+1}^n$ and A_0^n at π^l values of 0.4 and 0.5 indicates that the strategy of treating all trees becomes increasingly optimal once the infestation level exceeds a certain threshold. This transition toward treating all trees becomes more apparent at π^l values greater than 0.6.

In summary, our EH captures these broader trends while incorporating the nuances to predict less frequent solutions. These nuanced yet explainable predictions for the less frequent solutions can help local foresters ensure that only the required level of treatment is administered, thus avoiding unnecessary expenditures and improving the efficiency of the overall budget allocation.

Our findings demonstrate the potential of using ML algorithms to infer analytical solutions for optimization models, resulting in more tractable and interpretable solutions. The EH highlights this potential. Our EH can predict part of the solution, while the remaining decisions can be solved using exact approaches, offering a hybrid approach that balances interpretability with computational efficiency. This approach has the potential to solve complex problems with practical solution times, while enhancing the ease of use and interpretability of these complex optimization mechanisms.

In addition, we also demonstrate the generalizability of ML-based heuristics that predict discrete solutions across different problem sizes. Future research can explore the generalization of predictions

for continuous variables on varying problem scales. While this study focuses on supervised learning, future research could explore reinforcement learning (RL) approaches, while acknowledging the potential challenges of resource-intensive training and instability in complex settings (Barto and Dietterich 2004). A similar implementation can be explored in RL (Bushaj et al. 2023, Yilmaz and Büyüktahtakın 2024a, Bushaj and Büyüktahtakın 2024) to make policy functions interpretable to a user, especially in domains with legal requirements to explain decisions. Finally, this methodology can be applied to models that emulate contractual cooperation to facilitate the formulation of effective contractual policies for the agents involved.

DECLARATIONS

- **Funding:** The authors are grateful for the funding provided by the Grado Department of Industrial and Systems Engineering and the computing clusters offered by the Advanced Computing Resources (ARC) Center at Virginia Tech.

- **Competing Interests:** The authors have no relevant financial or non-financial interests to disclose.
- **Ethics approval and consent to participate:** Not applicable.
- **Consent for publication:** Not applicable.
- Data availability: The authors confirm that the methods for generating all data during this study are included in this published article.
- Materials availability:) Not applicable.
- Code availability: Not applicable.
- Author contribution: Not applicable.

References

Arrieta, A. B., N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Information fusion 58, 82–115.

Bang, J. T., A. Basuchoudhary, and A. Mitra (2021). Validating game-theoretic models of terrorism: Insights from machine learning. Games 12(3), 54.

Barto, A. G. and T. G. Dietterich (2004). Reinforcement learning and its relationship to supervised learning. Handbook of learning and approximate dynamic programming 10, 9780470544785.

Bello, I., H. Pham, Q. V. Le, M. Norouzi, and S. Bengio (2016). Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940.

Bertsimas, D. and R. Demir (2002). An approximate dynamic programming approach to multidimensional knapsack problems. Management Science 48(4), 550–565.

Bertsimas, D. and J. Dunn (2017). Optimal classification trees. Machine Learning 106, 1039–1082.

Bertsimas, D. and B. Stellato (2021). The voice of optimization. Machine Learning 110(2), 249–277.

Bhat, M. G. and R. G. Huffaker (2007). Management of a transboundary wildlife population: A self-enforcing cooperative agreement with renegotiation and variable transfer payments. Journal of Environmental Economics and Management 53(1), 54–67.

Bushaj, S. and İ. E. Büyüktahtakın (2024). A K-means supported reinforcement learning framework to multi-dimensional knapsack. Journal of Global Optimization 89, 655–685.

Bushaj, S., İ. E. Büyüktahtakın, and R. G. Haight (2022). Risk-averse multi-stage stochastic optimization for surveillance and operations planning of a forest insect infestation. European Journal of Operational Research 299(3), 1094–1110.

Bushaj, S., X. Yin, A. Beqiri, D. Andrews, and I. E. Büyüktahtakın (2023). A simulation-deep reinforcement learning (sirl) approach for epidemic control optimization. Annals of Operations Research 328(1), 245–277.

Büyüktahtakın, İ. E., Z. Feng, G. Frisvold, and F. Szidarovszky (2013). Invasive species control based on a cooperative game. Applied Mathematics 4(10), 54–59.

Büyüktahtakın, İ. E., Z. Feng, G. Frisvold, F. Szidarovszky, and A. Olsson (2011). A dynamic model of controlling invasive species. Computers & Mathematics with Applications 62(9), 3326–3333.

Büyüktahtakın, I. E. and R. G. Haight (2018). A review of operations research models in invasive species management: state of the art, challenges, and future directions. Annals of Operations Research 271(2), 357–403.

Chen, C., W. Cai, İ. E. Büyüktahtakın, and R. G. Haight (2024). A game-theoretic approach to incentivize landowners to mitigate an emerald ash borer outbreak. IISE Transactions 56(11), 1131–1145.

Cobourn, K. M., G. S. Amacher, and R. G. Haight (2019). Cooperative management of invasive species: a dynamic nash bargaining approach. Environmental and resource economics 72, 1041–1068.

Deng, H. (2019). Interpreting tree ensembles with intrees. International Journal of Data Science and Analytics 7(4), 277–287.

Dieber, J. and S. Kirrane (2020). Why model why? assessing the strengths and limitations of lime. arXiv preprint arXiv:2012.00093.

Dimitris, B. and S. Bartolomeo (2022). INFORMS Journal on Computing. https://doi.org/10.1287/ijoc.2022.1181.

Gilpin, L. H., D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal (2018). Explaining explanations: An overview of interpretability of machine learning. In 2018 IEEE 5th International Conference on data science and advanced analytics (DSAA), pp. 80–89. IEEE.

Gurobi Optimization, L. (2023). Gurobi optimizer reference manual.

Haddouchi, M. and A. Berrado (2019). A survey of methods and tools used for interpreting random forest. In 2019 1st International Conference on Smart Systems and Data Science (ICSSD), pp. 1–6. IEEE.

Hottung, A., S. Tanaka, and K. Tierney (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. Computers & Operations Research 113, 104781.

Kallestad, J., R. Hasibi, A. Hemmati, and K. Sörensen (2023). A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems. European Journal of Operational Research 309(1), 446–468.

Kıbış, E. Y., İ. E. Büyüktahtakın, R. G. Haight, N. Akhundov, K. Knight, and C. E. Flower (2021). A multistage stochastic programming approach to the optimal surveillance and control of the emerald ash borer in cities. INFORMS Journal on Computing 33(2), 808–834.

Le, T.-T.-H., H. Kim, H. Kang, and H. Kim (2022). Classification and explanation for intrusion detection system based on ensemble trees and shap method. Sensors 22(3), 1154.

Liu, R., C. Wang, H. Ouyang, and Z. Wu (2024). Exact algorithm and machine learning-based heuristic for the stochastic lot streaming and scheduling problem. IISE Transactions, 1–15.

Liu, Y. and C. Sims (2016). Spatial-dynamic externalities and coordination in invasive species control. Resource and Energy Economics 44, 23–38.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence 267, 1–38.

Miraboutalebi, S. M., P. Kazemi, and P. Bahrami (2016). Fatty acid methyl ester (fame) composition used for estimation of biodiesel cetane number employing random forest and artificial neural networks: a new approach. Fuel 166, 143–151.

Molnar, C. (2020). Interpretable Machine Learning. Independently Published.

Pan, X., M. Chen, T. Zhao, and S. H. Low (2022). Deepopf: A feasibility-optimized deep neural network approach for ac optimal power flow problems. IEEE Systems Journal 17(1), 673–683.

Petkovic, D., R. Altman, M. Wong, and A. Vigil (2018). Improving the explainability of random forest classifier–user centered approach. In Pacific symposium on biocomputing 2018: proceedings of the pacific symposium, pp. 204–215. World Scientific.

Ramon, Y., D. Martens, T. Evgeniou, and S. Praet (2024). Can metafeatures help improve explanations of prediction models when using behavioral and textual data? Machine Learning 113(7), 4245–4284.

Sagi, O. and L. Rokach (2020). Explainable decision forest: Transforming a decision forest into an interpretable tree. Information Fusion 61, 124–138.

Siriwardena, S. D., K. M. Cobourn, G. S. Amacher, and R. G. Haight (2018). Cooperative bargaining to manage invasive species in jurisdictions with public and private lands. Journal of Forest Economics 32, 72–83.

Vinyals, O., M. Fortunato, and N. Jaitly (2015). Pointer networks. arXiv preprint arXiv:1506.03134.

Yilmaz, D. and İ. E. Büyüktahtakın (2023). Learning optimal solutions via an LSTM-optimization framework. Operations Research Forum 4(2), 48.

Yilmaz, D. and İ. E. Büyüktahtakın (2024a). A deep reinforcement learning framework for solving two-stage stochastic programs. Optimization Letters 18, 1993–2020.

Yilmaz, D. and İ. E. Büyüktahtakın (2024b). An expandable machine learning-optimization framework to

sequential decision-making. European Journal of Operational Research 314(1), 280–296.

Zhang, J. (2017). An efficient density-based clustering algorithm for the capacitated vehicle routing problem. In 2017 international conference on computer network, electronic and automation (ICCNEA), pp. 465–469. IEEE.

Appendices

A NOTATION OF TBR FORMULATION

Here, we present the detailed notation used in the TBR re-formulation (1).

Parameters:

π: The probability of EAB infestation during the initial period.

α: The cost of surveying an ash tree for EAB infection in dollar value.

β: The cost of treating an ash tree for EAB infection in dollar value.

ρ: The probability of success of the standard insecticide treatment for EAB.

θ: The perceived marginal value of a surviving tree for a private land owner.

s: The perceived marginal value of a healthy tree for the city forester.

π^l: Infestation probability in the second period if treatment is done in the first period.

π^h: Infestation probability in the second period if treatment is not done in the first period.

γ: The penalty cost perceived by the city forester for a dying tree.

c: Cost of removing a dead tree in dollar terms.

Sets and Indices:

n: the total number of ash trees.

I: Set of possible infestation levels and $I = \{0, \ldots, n\}$.

J: Set of possible treatment decisions and $J = \{0, \ldots, n\}$.

i: Index for the infestation level $i \in I$.

j: Index for the treatment decision $j \in J$.

Decision variables:

$q(i)$: Number of trees to be treated at the infestation level i.

$r(q(i))$: The value of reimbursement to incentivize landowners to treat q trees at the infestation

level i.

Computed variables:

w_t: The number of successfully treated trees in period t.

d_t: The number of un-treated or un-successfully treated trees in period t.

$\Phi(q, r|n)$: The utility of the land owner when they decide to treat q trees for EAB infestation out of n trees and receive a reimbursement amount of r.

B DEVELOPING THE ENSEMBLE HEURISTIC

This section presents the details of the EH algorithm along with an illustrative example.

Algorithm 1 Recursive Branch Enumeration

Require: T: Base heuristic tree; N: Number of instances; H: Current hierarchy level in the tree; $\mathcal{RF}$: the Random Forest model trained on solved instances.

$\mathcal{R} \leftarrow$ Set of extracted rules and their respective outcomes.
▷ Subset of rules covering 99% of the instances for each outcome.
$\mathcal{F} \leftarrow$ List of features from $\mathcal{RF}$ sorted by importance .
function ENUMERATE_BRANCHES(T, N, $\mathcal{R}$, H=1, $\mathcal{F}$, *partition_inputs*)
    terminate_check, output = SHOULDTERMINATE(N, $\mathcal{R}$, H)
    if terminate_check **then**
        $T(H, N) \leftarrow$ output
        return
            ▷ Terminate if no further enumeration is required.
    else
        Compute feature f from $\mathcal{F}$ for level H.
        Compute *p_input* from *partition_inputs* for feature f.
        for each *branch* in $T(H, N)$ **do**
            if a hierarchy level $T(H+1, N)$ does not exist in T **then**
                Create a new hierarchy level $T(H+1)$ in tree T corresponding to $H+1$.
            end if
            Add a *new_node* to $T(H+1), N$.
            new_branches, *branch_rules*, *term_rules* ←PARTITION(*branch*,f,*p_input*,($\mathcal{R}$))
            $\mathcal{R}_n \leftarrow$ branch_rules
            Add *new_branches*, *branch_rules*, *term_rules* to $T(H+1, N)$.
            ENUMERATE_BRANCHES(T, N, $\mathcal{R}_n$, $H+1$, $\mathcal{F}$, *partition_inputs*)
                ▷ Recursive call to enumerate branches at the next level.
        end for
    end if
    return T ▷ T is the tree containing all the nodes and branches and their corresponding rules and value ranges
end function

For each rule ($\mathcal{R}$), the relevant condition on the feature is evaluated to determine the range of

values that satisfy the condition. This range is then compared to each interval to ascertain whether they intersect. If the rule's value range overlaps with the interval's range, the rule is considered to intersect with that interval and is thus affiliated with it. For instance, if a rule has the condition $\beta \geq 200$ and an interval ranges from 300 to 350, the rule intersects with this interval because their ranges overlap. During implementation, each interval maintains a set of outcomes associated with the rules that intersect it. Moreover, a single rule can be associated with multiple intervals.

Furthermore, consecutive intervals with identical outcome clusters are combined to form a broader value range, referred to as partitions. This strategy of combining intervals is also applied to scenarios where an interval lacks a definitive outcome but is either preceded by or followed by an interval with an outcome, providing a local approximation of the outcome. These partitions constitute the branches that split at that node.

This iterative refinement and rule assignment process extends down the hierarchy as described in Algorithm 1. Each branch leads to a new node at the next hierarchical level, with the rules associated with the preceding branch ($\mathcal{R}_n$) serving as the input ($\mathcal{R}$) for partitioning the new node. At each new node, the ranges of values of the corresponding feature are partitioned using Algorithm 2. This process continues recursively until either uniformity in the outcomes is reached (signifying a leaf node) or all features have been employed in the hierarchy.

Algorithm 2 Partitioning Algorithm

function PARTITION(*node*, $feature$, *partition_inputs*, $\mathcal{R}$)
    Divide the value range of feature f into intervals based on *partition_inputs*.
        ▷ *partition_inputs contains the details of value ranges of features and interval sizes for partitioning.*
    for each interval p in f's divided range **do**
        $\mathcal{R}_p \leftarrow$ rules from $\mathcal{R}$ that apply to interval p.
        Determine and assign the outcomes to p based on $\mathcal{R}_p$.
        Merge consecutive intervals p having the same outcome to form new branches.
    end for
    $\mathcal{B} \leftarrow$ Finalized branches and their corresponding value ranges
    $\mathcal{R}_n \leftarrow$ rules from $\mathcal{R}$ applicable to the new branch $\mathcal{B}$.
    Identify and list any terminating rules from $\mathcal{R}_n$ that can't be partitioned further.
    return $\mathcal{B}$, $\mathcal{R}_n$, terminating rules
end function

Despite iterative refinement, certain scenarios yielded multiple potential outcomes after exhaustive feature partitioning. In these instances, the most frequent outcome within the local value range was

designated as the definitive result, as described in Algorithm 3, ensuring the practical applicability and reliability of the ensemble heuristic. Our heuristic incorporates a mechanism for handling "terminating" rules. These rules contain conditions only on a subset of features and conclude the condition sequence before all possible features are employed. Terminating rules are cataloged separately and, when evaluating an input, serve as the primary evaluative criteria. If an input satisfies the conditions of a terminating rule, the corresponding outcome is assigned, and the traversal of subsequent child nodes are bypassed. The subsequent child nodes would be traversed only when the conditions for the terminating rules are not met, as described in Algorithm 4. This approach enhances the efficiency and accuracy of the heuristic by prioritizing the evaluation of terminating rules, which can quickly provide definitive outcomes without the need for further hierarchical evaluation.

Algorithm 3 Termination Conditions

function isLeaf(R)
    check ← False
    if the number of unique outcomes in R equals 1 **then**
        check ← True
    end if
    return *check*
end function
function ShouldTerminate(*node*, R_n, H_no, n_f)
                ▷ *check* if the current node is a leaf or if all features have been processed.
                ▷ n_f is the total number of input features
    check ← False
    output ← False
    if isLeaf(R_n) **then**
        check ← True
        output ← "unique outcome of R_n"
    else if $H\_no = n_f$ **then**
        check ← True
        output ← "Locally most frequent outcome in R_n"
    end if
    return *check*, *output* ▷ *check* is a boolean value referring to a node being leaf or not.
                ▷ *output* is the final unique solution at the leaf node.
end function

We program Algorithm 1 in Python. In our formulation, the inputs $\mathcal{T}$, R, and *partition_inputs* are dictionaries. The elements of $\mathcal{T}$ are maintained in three dictionaries. The first dictionary stores nodes with nested directionalities to represent the hierarchical relationship between nodes.

Algorithm 4 Implementation of EH

```
function IMPLEMENT_TREE(input_values, T)
    OTD ← None                                  ▷ Initialize the Optimal Treatment Decision as None
    H ← Hierarchical order of T                 ▷ Retrieve the hierarchical order from tree T
    f_i ← None                                  ▷ Initialize feature f_i to consider at each level
    for f_i in the order of H do
        Ascertain the appropriate value range for the input f_i
        if terminal rules at node are satisfied then
            OTD ← apply the terminal rule's solution
            break                               ▷ Exit loop as solution is found
        else if leaf node is reached then
            OTD ← apply the leaf node's solution
            break                               ▷ Exit loop, solution found at leaf node
        else
            continue                            ▷ Continue to the next feature
        end if
    end for                                     ▷ Validate the feasibility of the solution
    if solution is infeasible then
        OTD ← An alternate feasible solution
    end ifreturn OTD
end function
```

The second dictionary stores the value ranges of each node and the third dictionary stores the terminating rules for each node. $\mathcal{F}$ is passed as an ordered list of features, while N and H are passed as initializing entries of strings and numbers, respectively. The function returns a collection of dictionaries that collectively represent the elements of $\mathcal{T}$ as described above.

B.0.1 Illustration of the ensemble heuristic using a simple example

In this section, we present an example implementation of Algorithm 1, illustrated in Figure 5. The inputs and their corresponding notation used for this example are as follows:

- Set of input features: f_i, $\forall\, i \in \{1, 2, 3, 4\}$. A feature can be an individual input parameter or a combination of one or more parameters.
- Set of outcomes to be predicted: O_i, $\forall\, i \in \{1, 2, 3\}$. An outcome refers to a solution cluster in the TBR model.
- Branches in the tree: $B_{i_1 i_2 \ldots i_k}$, where $1 \leq i_1 \leq 4$ and $1 \leq k \leq 4$, with $i_2, i_3, \ldots, i_k \geq 1$. In this notation, k denotes the hierarchical level of the branch, and the indices $i_1, i_2, ..., i_k$ indicate the path from the root to the node, representing sequential parent-child relationships. For instance, B_1, B_2, and B_3 are branches at the first hierarchical level. Their respective child branches at the second level are denoted as B_{11}, B_{21}, and B_{31}, illustrating that B_{11} is a child

of B_1, B_{21} is a child of B_2, and so on. Each subsequent index represents a deeper level in the tree hierarchy.

- The value ranges of each of the input features, along with their corresponding partition interval sizes and priority ranks are presented in Table 10. This table provides the inputs for `partition_inputs` and the feature importance $\mathcal{F}$ mentioned in Algorithm 1.

Table 10: Feature Interval and Importance

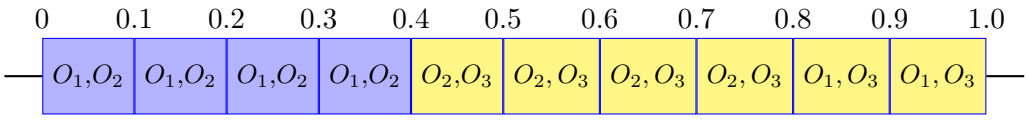
Feature	**Value Range**	**Interval Size**	**Feature Importance**
f_1	[0,1]	0.1	1
f_2	[0,300]	25	2
f_3	[0,500]	25	3
f_4	[0,1]	0.1	4

Table 11 presents the prediction rules extracted from the corresponding Random Forest (RF) model. Subsequently, these rule conditions will be referred to their assigned labels. The tree enumeration, outlined in Algorithm 1, begins by establishing the first hierarchical level and the corresponding root node. The function `ENUMERATE_BRANCHES` then generates branches from the root by partitioning the value range of the feature f_1, identified as the most important feature. According to Algorithm 2, the range of f_1 (from 0 to 1) is divided into intervals of 0.1, as specified in Table 10. This algorithm assesses the intersection of each rule with these intervals based on conditions related to f_1. Figure 5(a) visualizes these intervals post-evaluation of intersecting rules and outcomes. Finally, intervals with similar outcomes are merged, guided by the affiliated rules.

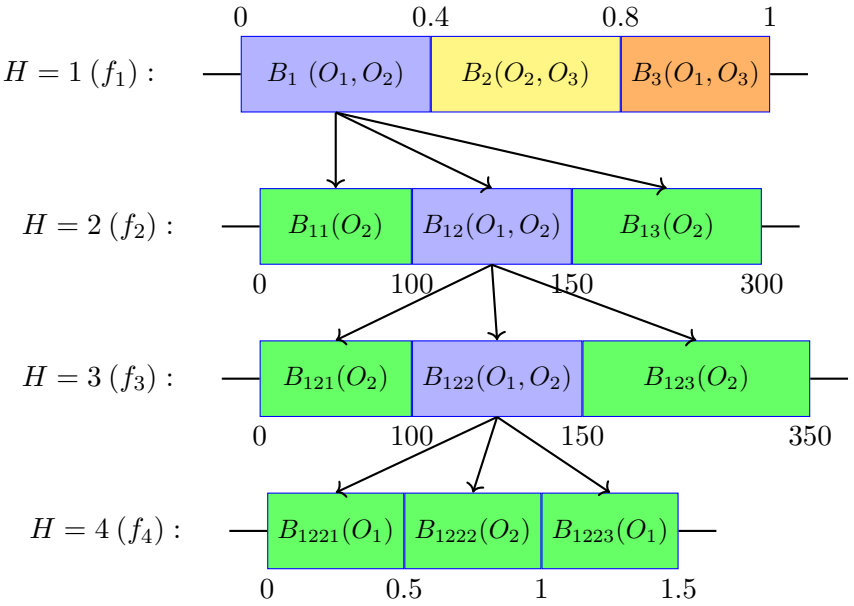
Table 11: Example of a Subset of Rules Extracted from the Random Forest Model

Rule Label	**Extracted Rules**	**Outcome**
r_1	$f_1 < 0.5$ & $f_2 > 100$ & $f_3 > 100$ & $f_4 < 0.4$	O_1
r_2	$f_1 > 0.4$ & $f_2 < 90$ & $f_3 < 150$ & $f_4 < 0.5$	O_2
r_3	$f_1 > 0.65$ & $f_1 < 0.8$ & $f_2 > 150$ & $f_3 > 250$ & $f_4 < 0.2$	O_3
r_4	$f_1 < 0.4$ & $f_2 > 80$ & $f_3 < 120$	O_2
r_5	$f_1 > 0.8$ & $f_2 > 60$ & $f_3 < 150$	O_1
r_6	$f_1 < 0.5$ & $f_2 < 150$ & $f_3 > 100$ & $f_3 < 150$ & $f_4 > 0.5$	O_2

For example, the first four intervals shown in Figure 5(a), ranging from 0 to 0.4, are associated with rules r_1, r_4, and r_6 because the allowed value ranges for f_1 in these rules intersect with these intervals. Similarly, rules r_2 and r_3 correspond to intervals from 0.4 to 0.8, while rules r_2, r_3, and r_5 are linked to intervals from 0.8 to 1. The corresponding outcomes of the intersecting rules are



(a) Intervals based on f_1 partitioned values



(b) Branches

Figure 5: Illustration of branch enumeration

also grouped with these intervals. Based on the similarities in the outcomes, the first four intervals are merged to form the branch B_1, which extends to the next hierarchical level leading to the next partition node. The intervals from 0.4 to 0.8 form branch B_2, and the intervals from 0.8 to 1 form branch B_3. The sub-figure labeled 'H=1 (f_1)' in Figure 5(b) illustrates the final branches B_1, B_2, and B_3, which are added to the first hierarchical level of the ensemble tree. Algorithm 2 also returns the corresponding rules, outcomes, and value ranges for each of the branches created.

Furthermore, the function `ENUMERATE_BRANCHES` initiates a new hierarchical level $H = 2$ due to the non-uniqueness of outcomes across all branches. At this level, the three branches from the first level leads to three corresponding partition nodes. Each of these nodes then undergoes further partitioning of the value range of f_2, following the procedure outlined in Algorithm 2, to create new child branches. The sub-figure labeled '$H = 2\ (f_2)$' in Figure 5(b) depicts the partitioned branches at the node corresponding to B_1. We adopt a naming convention for child branches, where child branches of B_1 are labeled B_{11}, B_{11} and B_{13}, indicating their extension from the parent node.

At level $H = 2$, nodes corresponding to B_{11} and B_{13} are identified as leaf nodes due to their unique outcomes. This triggers the termination of their branch enumeration, as the function `ShouldTerminate` in Algorithm 1 returns `True` for these nodes. The sub-figure labeled ' $H = 3$ (f_3)' in Figure 5(b) illustrates further partitions at the node corresponding to B_{12}. Here, the child branches B_{121} and B_{123} culminate in leaf nodes with a unique outcome O_2. At level $H = 4$, the node corresponding to B_{122} is further partitioned, resulting in three leaf nodes with outcomes O_1 and O_2. Furthermore, rule $r4$ is considered a terminating rule at this level, as it does not impose conditions on f_4. Upon achieving unique outcomes from all partitions of B_{122}, the enumeration concludes, prompted by the `ShouldTerminate` function. The detailed branches along with corresponding rules and value ranges are incorporated into the tree.