A Generic Complete Anytime Beam Search for Optimal Decision Tree

Harold Silvère Kiossou[1], **Siegfried Nijssen**[1,2], **Pierre Schaus**[1]

[1]UCLouvain, ICTEAM, Louvain-la-Neuve, Belgium
[2]KU LEUVEN, DTAI, Leuven, Belgium
harold.kiossou@uclouvain.be , siegfried.nijssen@kuleuven.be, pierre.schaus@uclouvain.be

Abstract

Finding an optimal decision tree that minimizes classification error is known to be NP-hard. While exact algorithms based on MILP, CP, SAT, or dynamic programming guarantee optimality, they often suffer from poor anytime behavior—meaning they struggle to find high-quality decision trees quickly when the search is stopped before completion—due to unbalanced search space exploration. To address this, several anytime extensions of exact methods have been proposed, such as LDS-DL8.5, Top-k-DL8.5, and Blossom, but they have not been systematically compared, making it difficult to assess their relative effectiveness. In this paper, we propose CA-DL8.5, a generic, complete, and anytime beam search algorithm that extends the DL8.5 framework and unifies some existing anytime strategies. In particular, CA-DL8.5 generalizes previous approaches LDS-DL8.5 and Top-k-DL8.5, by allowing the integration of various heuristics and relaxation mechanisms through a modular design. The algorithm reuses DL8.5's efficient branch-and-bound pruning and trie-based caching, combined with a restart-based beam search that gradually relaxes pruning criteria to improve solution quality over time. Our contributions are twofold: (1) We introduce this new generic framework for exact and anytime decision tree learning, enabling the incorporation of diverse heuristics and search strategies; (2) We conduct a rigorous empirical comparison of several instantiations of CA-DL8.5—based on Purity, Gain, Discrepancy, and Top-k heuristics—using an anytime evaluation metric called the primal gap integral. Experimental results on standard classification benchmarks show that CA-DL8.5 using LDS (limited discrepancy) consistently provides the best anytime performance, outperforming both other CA-DL8.5 variants and the Blossom algorithm while maintaining completeness and optimality guarantees.

Code — https://anonymous.4open.science/r/cadl85/
Datasets —
https://dtai-static.cs.kuleuven.be/CP4IM/datasets/

Introduction

Decision trees are a fundamental machine learning model, widely adopted for their interpretability and solid performance in domains such as healthcare, finance, and education. Classic algorithms like CART (Breiman et al. 1984) and C4.5 (Quinlan 2014) induce decision trees greedily, selecting splits in a top-down manner based on local heuristics. These methods are fast but lack optimality guarantees, and they often produce suboptimal trees.

Recent years have seen growing interest in exact decision tree learning algorithms, which aim to find globally optimal trees, typically minimizing classification error or another loss function. These algorithms leverage combinatorial optimization techniques from MILP (Bertsimas and Dunn 2017; Aghaei, Gómez, and Vayanos 2021), constraint programming (Verhaeghe et al. 2020), SAT (Narodytska et al. 2018), and dynamic programming (Aglin, Nijssen, and Schaus 2020; Demirović et al. 2022). While these methods offer strong generalization properties (Bertsimas and Dunn 2017; van der Linden et al. 2024), they tend to suffer from poor anytime behavior: when interrupted before convergence, they often return poor-quality solutions.

The DL8.5 algorithm (Aglin, Nijssen, and Schaus 2020), based on dynamic programming and efficient caching, is a state-of-the-art method for optimal tree induction. DL8.5 explores the search space in a depth-first fashion, which often causes it to become stuck in unpromising regions, as illustrated in Figure 1. As a result, it may return poor trees when stopped early. Greedy methods like C4.5 provide quick results but lack the capacity to improve or guarantee optimality over time. Neither approach offers the benefits of a true anytime algorithm, which should return a good initial solution quickly and improve it continuously as time allows.

To improve the anytime performance or scalability of exact methods, three notable work have been proposed. LDS-DL8.5 (Kiossou et al. 2022) integrates limited discrepancy search (LDS) into DL8.5, resulting in an algorithm that is both anytime and complete. Top-k-DL8.5 (Blanc et al. 2024) modifies DL8.5 by restricting the candidate features at each node to the Top-k according to a ranking heuristic. This is a compromise between C4.5 and DL8.5: faster and more scalable, but unable to guarantee convergence to the optimal tree. Finally, the Blossom algorithm (Demirović, Hebrard, and Jean 2023) follows a fundamentally different search strategy. It uses a depth-first approach that expands decision tree nodes level by level, offering improved anytime behavior by avoiding the possible result of highly unbalanced trees when interrupted early as for DL8.5. Blossom is guaranteed to find the optimal tree given sufficient time. Despite their

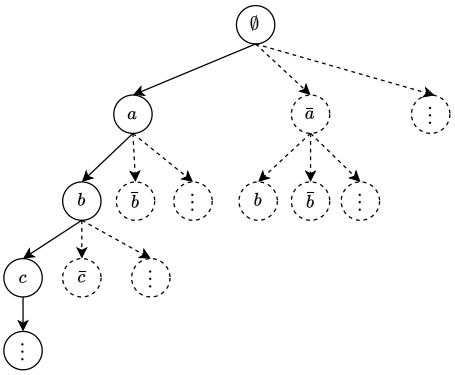

Figure 1: DL8.5 explores the leftmost branches first, often leading to poor anytime performance when interrupted early.

promise, these three approaches have not been systematically compared in prior work, making it difficult to assess their relative strengths.

This paper makes two main contributions. First, we introduce CA-DL8.5, a novel algorithm for decision tree learning that is: **complete** (guarantees optimality when given sufficient time), **anytime** (produces high-quality solutions early and improves them over time), and **generic** (easily instantiated with different heuristics and strategies). CA-DL8.5 builds upon the Complete Anytime Beam Search (CABS) framework (Zhang 1998), extending DL8.5 with an iterative weakening strategy that gradually relaxes pruning constraints across restarts. It generalizes LDS-DL8.5 and extends Top-k-DL8.5 to a complete method.

Second, we perform a rigorous empirical study of CA-DL8.5 under four heuristic strategies: Purity, Gain, Discrepancy, and Top-k. We evaluate these variants using the primal gap integral metric (Berthold 2013), a principled anytime evaluation measure that captures performance over time. Our results show that CA-DL8.5 instantiated with discrepancy-based search (equivalent to LDS-DL8.5) and Top-k consistently outperforms other instantiations and also improves upon the Blossom algorithm.

Overall, our work provides a unified and extensible framework for designing anytime exact decision tree algorithms, offering both theoretical guarantees and strong empirical performance. The CA-DL8.5 algorithm opens up new possibilities for combining optimality with real-time responsiveness in interpretable machine learning.

Related Works and Background

Greedy approaches Traditional tree-based algorithms, such as CART (Breiman et al. 1984) and C4.5 (Quinlan 2014), construct decision trees using greedy, top-down splits based on local information. These methods are efficient and scalable but cannot guarantee optimality. To address this limitation, several optimization-based approaches have been proposed: Bertsimas and Dunn (Bertsimas and Dunn 2017) formulated tree learning as a mixed-integer program, while others leveraged SAT (Narodytska et al. 2018), MaxSAT (Hu et al. 2020), and constraint programming (Verhaeghe et al. 2020). Although these methods provide theoretical guarantees, they struggle to scale to large datasets or deeper trees.

Dynamic Programming Approaches and DL8.5. Dynamic Programming (DP) methods (Nijssen and Fromont 2007; Aglin, Nijssen, and Schaus 2020; Demirović et al. 2022) improve the scalability of exact decision tree learning. DL8.5 (Aglin, Nijssen, and Schaus 2020), the foundation of our work, learns optimal decision trees on binary datasets $\mathcal{D}$ under minimum support and maximum depth constraints. A binary dataset $(\mathcal{D}, \mathcal{C})$ consists of data $\mathcal{D} \subseteq \prod_{f \in \mathcal{F}} \{f, \bar{f}\}$ over boolean features $\mathcal{F}$; $\mathcal{C}(x)$ indicates class labels for all $x \in \mathcal{D}$. For $\mathcal{S} \subseteq \mathcal{D}$ and feature f, define $\mathcal{S}(f) = \{x \in \mathcal{S} \mid f \in x\}$ and $\mathcal{S}(\bar{f})$ analogously. A binary decision tree assigns features to internal nodes and labels edges with f or $\bar{f}$, with each branch representing a root-to-leaf path. For branch b, $\mathcal{S}(b) = \bigcap_{f \in b} \mathcal{S}(f)$, and its classification error is $|\mathcal{S}(b)| - \max_{c \in \mathcal{C}} |\mathcal{S}_c(b)|$, where $\mathcal{C}$ indicates all class labels.

DP-based algorithms such as DL8.5 perform a depth-first exploration of an *OR-AND* search tree. As shown in Algorithm alg:dl85, at each node, it selects a feature (OR node) and recursively explores the two branches $\{f, \bar{f}\}$ (AND nodes). The left branch is evaluated first (Line 16) by filtering examples not matching the feature, followed by the right branch (Line 18) with examples that do. This recursion progressively reduces the dataset and enforces the minimum support (Line 14) and maximum depth (Line 7) constraints. Subtree errors are computed and combined to obtain the current tree's error. Efficiency is enhanced through two mechanisms: upper-bound pruning (Lines 17 and 19) discards unpromising branches, and a trie-based cache (Lines 9 and 28) stores previously solved subproblems to avoid redundant computations.

Anytime complete approaches As illustrated in Figure 1, the recursive depth-first exploration of DL8.5 prioritizes the leftmost branches of the search tree (solid lines), potentially delaying the exploration of other promising regions. Consider three binary features a, b, and c: the algorithm may fully explore the path abc and its variations before considering alternative feature combinations. With many features and larger maximum depths, the search can remain trapped in early subtrees for extended periods, leaving large regions of the search space such as $ab\bar{c}$, $a\bar{b}$, and $\bar{a}$ (dashed lines) unexplored until late in the process. This behavior introduces a key weakness: if interrupted, the algorithm typically produces an incomplete, unbalanced decision tree, that can be of lower quality than one built by simple greedy heuristics. To address this limitation and improve anytime performance, we propose integrating a complete anytime beam search strategy into DL8.5, preserving optimality guarantees while producing high-quality solutions throughout execution.

Recent work has explored improving the anytime behavior of exact decision tree algorithms. Kiossou et al. (Kiossou et al. 2022) introduced LDS-DL8.5, which applies iterative Limited Discrepancy Search (LDS) to prioritize solutions close to a heuristic baseline tree. A discrepancy corre-

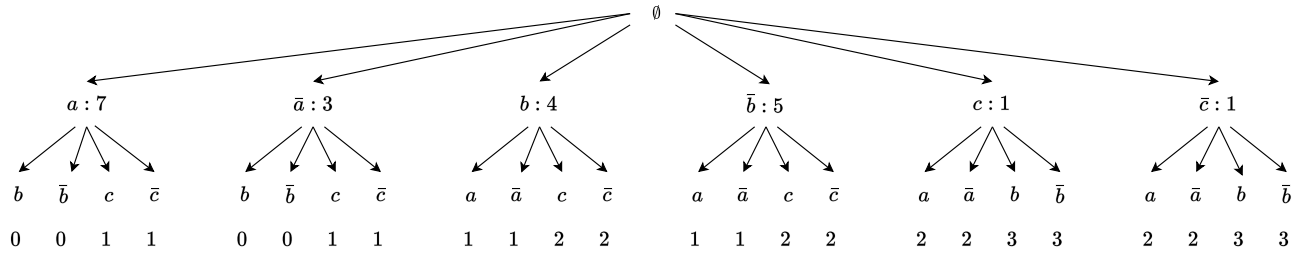

Figure 2: Search tree for three features. The values at depth 1 represent node error before expansion; at depth 2, they reflect discrepancy costs.

sponds to selecting a feature different from the one chosen by a greedy algorithm (e.g., the feature with maximum information gain). By gradually increasing the allowed number of discrepancies along a root-to-leaf path, the algorithm transitions from the greedy tree (zero discrepancies) toward an exhaustive search. This approach makes the algorithm more efficient in discovering the best tree. Early discovery of good decision trees helps prune the search space using the branch-and-bound techniques of DL8.5. In the worst case, the best tree found coincides with the greedy tree.

The Blossom algorithm, introduced in (Demirović, Hebrard, and Jean 2023), employs a different search approach to enhance the anytime behavior of the search. While it also utilizes depth-first search, it proceeds layer by layer within the tree, always expanding the non-expanded node that is closest to the root. Similar to LDS-DL8.5, the first solution found in the leftmost leaf node of the depth-first search exploration corresponds to the decision tree that would be identified by a purely greedy strategy. This characteristic makes it an improvement in terms of anytime behavior compared to DL8.5.

However, like DL8.5, the Blossom algorithm can also suffer from poor diversification of the search space, particularly concerning features of nodes close to the root. These features are reconsidered last during backtracking, despite being selected first and somewhat blindly when limited information was available for making an informed decision. To the best of our knowledge, no experiments have yet been conducted to compare Blossom and LDS-DL8.5.

Anytime incomplete approaches The Top-k method, introduced in (Blanc et al. 2024) is an extension of DL8.5 that aims at improving its scalability. It shares many similarities with LDS-DL8.5. Specifically, it heuristically restricts the set of features tried at each OR-node to the Top-k features, as determined by some ranking heuristic. The search space to explore is thus voluntarily limited and therefore this method can be considered as a trade-off between the pure greedy C4.5 and the complete search of DL8.5.

Other hybrid methods like LGDT (Kiossou et al. 2024) balance greedy efficiency with limited depth lookahead based on exact method to improve solution quality without the full computational cost of exact methods.

Complete Anytime Beam Search DL8.5 (CADL8.5)

Beam search is an informed search algorithm that explores a space by keeping only the most promising nodes at each level. While it improves upon breadth-first and depth-first search through heuristic pruning, overly aggressive pruning can cause it to miss solutions. To overcome this, Zhang (Zhang 1998) introduced Complete Anytime Beam Search (CABS), based on the principle of *iterative weakening* (Provost 1993). CABS performs successive beam search iterations with progressively relaxed pruning constraints. It starts with strict pruning, quickly identifying initial solutions, and then gradually weakens the pruning rules to explore larger portions of the search space. This process enables the algorithm to discover solutions that would have been pruned in earlier iterations.

This iterative relaxation approach gives CABS two valuable properties: (1) it provides anytime behavior by quickly finding initial solutions that improve over time, and (2) it ensures completeness by gradually reducing pruning constraints until an optimal solution is found. These properties make CBS particularly well-suited for complex optimization problems like decision tree learning, where balancing solution quality with computational efficiency is important.

In this work we propose Complete Anytime DL8.5 (CADL8.5), which adapts the original DL8.5 algorithm by including CABS ideas. It is a generic framework that guarantees high-quality solutions even when terminated early. By integrating principles from CABS, CADL8.5 guides tree construction using adaptive pruning rules. These rules are relaxed over multiple iterations, allowing the algorithm to prioritize promising regions initially while progressively exploring more diverse parts of the solution space.

Algorithm 2 presents the structure of CADL8.5. In addition to the standard inputs required by DL8.5 (dataset $\mathcal{D}$, minimum support `minsup`, and maximum depth `maxdepth`), CADL8.5 requires one additional parameter: `r`, the rule or the set of rules guiding the search space exploration. This allows the algorithm to balance between quickly finding initial solutions and ensuring optimality through complete exploration.

A key difference between DL8.5 and CADL8.5 is the introduction of an outer loop (Lines 6–11) that repeatedly invokes the `BeamDL8.5` search procedure while progressively

Algorithm 1: DL8.5

Input : $\mathcal{D}$, minsup, maxdepth
Output: Best tree under the maxdepth and minsup constraints

1 **struct** Best$\{error : \texttt{float}, ub : \texttt{float}, tree : \texttt{Tree}\}$
2 $cache \leftarrow Trie < branch,\ \texttt{Best} >$
3 $solution \leftarrow \texttt{DL85} - \texttt{Search}(\emptyset,\ ,+\infty)$
4 **return** $solution$
5 **Procedure** $\texttt{DL85} - \texttt{Search}\ (b,\ ub)$
6 $e \leftarrow \texttt{error}(b)$
7 **if** $|b| = \texttt{maxdepth}$ ***or*** $e = 0$ **then**
8 **return** $\texttt{Best}\{e,\ ub,\ \texttt{leaf}(b)\}$
9 $node \leftarrow cache.\texttt{get}(b)$
10 **if** $node \neq \emptyset$ ***and*** $ub \leq node.ub$ **then**
11 **return** $node$
12 $(\tau,\ best\_error,\ child\_ub) \leftarrow (\emptyset,\ +\infty,\ ub)$
13 **for** f **in** $\mathcal{F}$ *sorted by a heuristic* **do**
14 **if** $|\mathcal{D}(b \cup \{f\})| <$ minsup **or** $|\mathcal{D}(b \cup \{\bar{f}\})| <$ minsup **then**
15 **continue**
16 $left \leftarrow \texttt{DL85} - \texttt{Search}(b \cup \{\bar{f}\}, child\_ub)$
17 **if** $left.tree = \emptyset$ **then continue**
18 $right \leftarrow \texttt{DL85} - \texttt{Search}(b \cup \{f\}, child\_ub - left.error)$
19 **if** $right.tree = \emptyset$ **then continue**
20
21 $e \leftarrow left.error + right.error$
22 **if** $e < child\_ub$ **then**
23 $best\_error \leftarrow e$
24 $child\_ub \leftarrow e$
25 $\tau \leftarrow \texttt{Tree}(left.tree, right.tree)$
26 **if** $e = 0$ **then break**
27 $node \leftarrow \texttt{Best}\{best\_error,\ ub,\ \tau\}$
28 $cache.save(b, node)$
29 **return** $node$

relaxing constraints. The algorithm begins with strict pruning rules to rapidly identify a feasible solution that establishes an initial upper bound on the classification error. After each iteration, the rules are weakened using the `relax` function, expanding the search space. This process continues until one of three conditions is met: a perfect tree is found (zero error), no further rule relaxation is possible, or the allocated time budget is exhausted.

To implement this strategy, CADL8.5 introduces two generic types: `T` and `R`. Type `T` encapsulates state information required to apply pruning decisions, while `R` stores the parameters that define the current rule constraints. These types are manipulated through five core functions:

- $\texttt{update}(t : \texttt{T}, c : \texttt{Context}) \rightarrow \texttt{T}$: modifies a node's state based on its context, including error metrics, dataset size, and feature selection information.
- $\texttt{prune}(t : \texttt{T}, r : \texttt{R}) \rightarrow \texttt{boolean}$: evaluates whether a node should be pruned based on its current state and rule parameters.
- $\texttt{relax}(r : \texttt{R}) \rightarrow \texttt{R}$: incrementally weakens rule constraints to permit wider exploration in subsequent iterations.
- $\texttt{terminal\_state}(t : \texttt{T}) \rightarrow \texttt{T}$: generates a special state marking a node as fully explored and exempt from future pruning.
- $\texttt{initial\_state}(c : \texttt{Context}) \rightarrow \texttt{T}$: creates the initial state for the root node based on its context.

Rule	State & Constraint	Key Functions
Purity	T{purity : float}	update : purity $= 1 - \frac{\texttt{ctx.e}}{\lvert\texttt{ctx.S}\rvert}$
	R{min_purity : float}	prune : purity $\geq$ min_purity
		relax : min_purity $+= \delta$
Gain	T{cum_gap : float}	update : cum_gap $+=$ best_gain $-$ gain
	R{max_gap : float}	prune : cum_gap $>$ max_gap
		relax : max_gap $+= \delta$
Discrepancy	T{tot_discr : int}	update : tot_discr $+=$ ctx.i
	R{max_discr : int}	prune : tot_discr $\geq$ max_discr
		relax : max_discr $+= \delta$
Top-k	T{feat_idx : int}	update : feat_idx $=$ ctx.i
	R{k : int}	prune : feat_idx $\geq$ k
		relax : k $+= \delta$

Table 1: Rule definitions for CADL8.5

The search process begins by constructing an initial context c_0 at the root, incorporating the dataset's error measure and size. This context is passed to `initial_state` to generate the root's initial state t_0 (Line 5), which is then used alongside the initial upper bound to start the first `BeamDL8.5` search iteration.

At the root of the search space, the initial context c_0 is constructed using the root error and dataset size. This context is then passed to `initial_state` to produce the initial state t_0 (Line 5), which is used to start the first call to `BeamDL8.5` along with the initial upper bound.

During the `BeamDL8.5` procedure, each node's state is evaluated against the current rule configuration before expansion. The `update` function computes an updated state t based on the node's context (Lines 27 and 43). This state is then evaluated using the `prune` predicate (Line 17) to determine if the node violates any active constraints. If pruning conditions are met, the node is not expanded further and is returned as-is.

Nodes are also not expanded when they reach terminal conditions: either the maximum tree depth is attained or perfect classification (zero error) is achieved. In these cases, the node is explicitly marked as fully explored by applying the `terminal_state` function (Line 15). This ensures that terminal nodes are treated as optimal in future caching operations and pruning decisions, preventing unnecessary recomputation of already optimal subtrees.

CADL8.5 extends DL8.5's caching strategy to avoid redundant computations across iterations. Each branch b is associated with a `Best` object that stores the optimal subtree, its error bound, and its state. Before expanding any node, the cache is queried for previous results. If a com-

Algorithm 2: Complete Anytime DL8.5 (CADL8.5)

Input : $\mathcal{D}$, `rule`, minsup, maxdepth
Output: Best tree satisfying minsup and maxdepth

1 **Struct** `Best`{`error` : float, `ub` : float, `tree` : Tree, `state` : `T`}
2 $cache \leftarrow$ `Trie` $<$ `branch`, `Best` $>$
3 $ub \leftarrow +\infty$
4 $context_0 \leftarrow$ { i: 0, e: `error`$(\emptyset)$, s: $|\mathcal{D}|$ }
5 $state_0 \leftarrow$ `initial_state`$(context_0)$
6 **do**
7 $\quad sol \leftarrow$ `BeamDL8.5`$(\emptyset, ub, context_0, state_0)$
8 $\quad ub \leftarrow sol.error$
9 $\quad$ `rule` $\leftarrow$ `relax`(`rule`)
10 $\quad state_0 \leftarrow sol.state$
11 **while** *sol is not optimal* ***or*** `rule` *is relaxable*
12 **return** $sol.tree$
13 **Procedure** `BeamDL8.5`$(b, ub, context_p, state_p)$
14 $\quad e \leftarrow$ `error`(b)
15 $\quad$ **if** $|b| = maxdepth$ ***or*** $e = 0$ **then**
16 $\quad\quad$ **return** `Best`$\{e, ub,$ `leaf`$(b),$ `terminal_state`$(state_p)\}$
17 $\quad$ **if** *time limit reached* ***or*** `prune`$(state_p,$ `rule`$)$ **then**
18 $\quad\quad$ **return** `Best`$\{e, ub,$ `leaf`$(b), state_p\}$
19 $\quad node \leftarrow cache.$`get`$(b)$
20 $\quad$ **if** $node \neq \emptyset$ ***and*** $ub \leq node.ub$ ***and*** $\neg$`prune`$(node.state,$ `rule`$)$ **then**
21 $\quad\quad$ **return** $node$
22 $\quad \tau \leftarrow \emptyset, child\_ub \leftarrow ub, optimal \leftarrow$ `true`
23 $\quad$ **foreach** (i, f) ***in*** $\mathcal{F}$ *sorted by heuristic* **do**
24 $\quad\quad$ **if** $|\mathcal{D}(b \cup \{f\})| < minsup$ ***or*** $|\mathcal{D}(b \cup \{\bar{f}\})| < minsup$ **then**
25 $\quad\quad\quad$ **continue**
26 $\quad\quad context_l \leftarrow$ { i: i, e: `error`$(b \cup \{\bar{f}\})$, s: $|\mathcal{D}(b \cup \{\bar{f}\})|$ }
27 $\quad\quad state_l \leftarrow$ `update`$(state_p, context_l)$
28 $\quad\quad l \leftarrow$ `BeamDL8.5`$(b \cup \{\bar{f}\}, child\_ub, context_l, state_l)$
29 $\quad\quad$ **if** $l.tree = \emptyset$ **then**
30 $\quad\quad\quad$ **continue**
31 $\quad\quad context_r \leftarrow$ { i: i, e: `error`$(b \cup \{f\})$, s: $|\mathcal{D}(b \cup \{f\})|$ }
32 $\quad\quad state_r \leftarrow$ `update`$(state_p, context_r)$
33 $\quad\quad r \leftarrow$ `BeamDL8.5`$(b \cup \{f\}, child\_ub - l.error, context_r, state_r)$
34 $\quad\quad$ **if** $r.tree = \emptyset$ **then**
35 $\quad\quad\quad$ **continue**
36 $\quad\quad e \leftarrow l.error + r.error$
37 $\quad\quad$ **if** `prune`$(l.state,$ `rule`$)$ ***or*** `prune`$(r.state,$ `rule`$)$ **then**
38 $\quad\quad\quad optimal \leftarrow$ `false`
39 $\quad\quad$ **if** $e < child\_ub$ **then**
40 $\quad\quad\quad child\_ub \leftarrow e$
41 $\quad\quad\quad \tau \leftarrow$ `Tree`$(l.tree, r.tree)$
42 $\quad\quad\quad context_p.error \leftarrow e$
43 $\quad\quad\quad state_p \leftarrow$ `update`$(state_p, context_p)$
44 $\quad\quad$ **if** $e = 0$ **then**
45 $\quad\quad\quad$ **break**
46 $\quad$ **if** $optimal$ **then**
47 $\quad\quad state_p \leftarrow$ `terminal_state`$(state_p)$
48 $\quad node \leftarrow$ `Best`$\{child\_ub, ub, \tau, state_p\}$
49 $\quad cache.$`save`$(b, node)$
50 $\quad$ **return** $node$

patible cached entry exists—one whose upper bound is consistent with current requirements and not pruned under the current rule set—the cached result is immediately reused (Line 20). Additionally, nodes are marked as fully explored only when all their children have been optimally processed (Lines 47–49).

To control the exploration cost of nodes in CADL8.5, we implement four different pruning strategies: the *Purity* rule, the *Gain* rule, the *Discrepancy* rule, and the *Top-k* rule. Table 1 summarizes these rules using the generic type structures introduced earlier.

Purity rule

The purity rule defines a minimum purity threshold for the tree stored in `R`. A node expansion is stopped when its purity meets or exceeds this threshold. Weakening the rule involves incrementally increasing the threshold by a value δ until it reaches a maximum of 1.0. If a node's purity remains below the current threshold (`prune`), the search continues along that branch until the maximum depth is reached, during which purity may improve. Without a depth constraint, this strategy would attempt to construct a perfect decision tree. The purity of a branch b is defined as

$$purity(b) = 1 - \frac{error(b)}{|\mathcal{S}(b)|}.$$

For example, in the search tree of Figure 2, assume 10 examples fall into each branch at depth 1, with $purity(a) = 0.3$ and $purity(\bar{a}) = 0.7$. With a threshold of 0.5, branch a is expanded, whereas $\bar{a}$ is not, since it is pure enough.

Gain rule

The gain rule restricts feature expansion using an information gain threshold. At a node, let τ^* be the highest gain and $\tau(f)$ the gain of a feature f. The local gap is $\tau^* - \tau(f)$, and each feature maintains a cumulative gap along the path from the root:

$$\texttt{cum\_gap} = \texttt{cum\_gap}_{\text{parent}} + (\tau^* - \tau(f)).$$

A feature is expanded only if $\texttt{cum\_gap} \leq \texttt{max\_gap}$. Setting $\texttt{max\_gap} = 0$ yields a greedy strategy similar to C4.5, while larger values allow broader exploration. The cumulative constraint naturally tightens at deeper levels, focusing the search and avoiding excessive exploration of suboptimal branches.

Discrepancy rule

The Discrepancy rule employs the same principle of Limited Discrepancy Search (LDS) as in the LDS-DL8.5 algorithm (Kiossou et al. 2022), to control deviations from a preferred exploration order. Each node tracks the total discrepancy `tot_discr` accumulated from the root, where each feature is assigned an index i based on its position in the candidate list. The discrepancy of a path thus reflects how many times the search deviated from the leftmost option.

At each node, only features whose associated cost does not exceed the threshold `max_discr` are considered. For example, exploring only the leftmost feature at each split (with

Approach	Sub	Runtime (s) 15	30	60	120	240	300
C4.5	–	64.3	64.3	64.3	64.3	64.3	64.3
Top-3	–	46.3	45.0	44.4	44.1	44.0	44.0
Top-5	–	37.5	35.2	34.3	33.8	33.6	33.5
DL8.5	–	46.6	40.4	34.6	31.5	29.5	28.8
Blossom	–	**27.1**	24.7	19.6	14.5	9.6	8.5
CA-Purity	–	48.8	42.5	36.8	31.6	27.7	26.7
CA-Gain	exponential	43.7	36.3	32.2	29.2	26.4	24.7
	luby	42.8	35.9	31.5	25.5	22.0	21.2
	monotonic	43.2	36.6	32.3	26.2	22.5	21.5
CA-Discrepancy	exponential	29.1	23.2	18.6	15.8	13.0	11.4
	luby	27.2	**20.7**	**16.7**	13.9	9.2	7.8
	monotonic	27.4	20.8	16.7	14.0	8.8	**7.4**
CA-Top-k	exponential	34.9	30.6	24.9	20.0	17.5	16.8
	luby	32.7	25.4	20.5	17.3	14.0	12.0
	monotonic	31.1	23.5	19.0	16.3	13.4	11.3
CA-Top-k^*	exponential	34.4	27.2	23.3	17.0	12.7	11.4
	luby	31.0	25.4	18.0	**12.2**	**8.5**	7.7
	monotonic	31.2	25.6	18.4	12.2	8.6	7.7

Table 2: Average primal integral on depth 6

Approach	Sub	Runtime (s) 15	30	60	120	240	300
C4.5	–	69.3	69.3	69.3	69.3	69.3	69.3
Top-3	–	52.9	51.7	51.1	50.8	50.7	50.6
Top-5	–	46.8	41.0	38.5	37.2	36.6	36.5
DL8.5	–	58.3	52.0	47.2	43.9	40.4	39.5
Blossom	–	37.1	30.5	24.7	21.5	18.6	17.2
CA-Purity	–	55.9	50.0	45.7	43.2	40.7	39.7
CA-Gain	exponential	50.5	45.3	39.1	35.5	33.0	32.5
	luby	54.7	51.4	45.6	41.7	38.0	36.9
	monotonic	55.1	51.6	46.2	41.9	39.3	38.5
CA-Discrepancy	exponential	33.0	28.3	25.0	22.8	19.3	17.8
	luby	**31.3**	26.9	22.8	19.0	15.1	**14.0**
	monotonic	31.5	**26.5**	**22.5**	**19.0**	**15.1**	14.0
CA-Top-k	exponential	41.8	34.5	28.5	25.1	22.4	21.1
	luby	39.4	31.2	25.3	22.1	19.1	18.0
	monotonic	37.7	29.7	24.2	20.8	17.6	16.6
CA-Top-k^*	exponential	41.8	36.7	33.9	31.3	27.7	26.6
	luby	39.6	34.7	30.7	26.7	23.4	22.3
	monotonic	41.8	35.8	31.3	26.9	23.8	22.6

Table 3: Average primal integral on depth 7

`max_discr` $= 0$) results in a greedy tree. Increasing the discrepancy budget expands the search space and enables exploring other parts of the space search.

As illustrated in Figure 2, if feature A is explored first, then $cost(a) = cost(\bar{a}) = 0$. Choosing B instead at the same level requires $cost(b) = cost(\bar{b}) = 1$. Similarly, deeper paths such as ba and $b\bar{a}$ have $cost = 1$ since A is the first successor of b, and $cost = 2$ for branches like bc since C is the second.

Top-k rule

The *Top-k rule* controls the breadth of the search by limiting the number of features explored at each node. It considers only the k best candidates, where the position of a feature in the sorted list determines its index `ctx.i`. When $k = 1$, the algorithm behaves greedily, producing trees similar to CART or C4.5 depending on the heuristic used. As k increases, more features are evaluated per node, expanding the search space and allowing corrections to early decisions. We also propose a new variant, denoted as Top-k* in the results, where the beam width k is halved at each level of the tree, with a minimum value of one. This allows to reduce the time spent in the deeper parts of the search space in early iterations.

Unlike the Discrepancy rule, Top-k and Top-k^* rules do not accumulate costs across the tree. The feature index is local to the node and does not depend on the path. A node is pruned if its feature index exceeds the current threshold k, unless marked as terminal. The relaxation function increments k, progressively weakening the pruning condition. The state tracks the index of the selected feature (`feat_idx`), and pruning is bypassed when this index is set to ∞, used as a sentinel for terminal nodes.

Results

To evaluate the performance of CADL8.5, we conducted a series of experiments. This section presents the results. We begin by analyzing the anytime behavior of CADL8.5 using the previously mentioned rules, followed by a comparison of its performance in finding optimal solutions. All experiments were conducted on 25 datasets from CP4IM, with a minimum support threshold of 1. The algorithms were executed on a server equipped with an Intel Xeon Platinum 8160 CPU and 320 GB of RAM, running Rocky Linux version 8.4. For comparison, we include a scikit-learn implementation of C4.5[1], DL8.5 and Blossom implementations. We compare CADL8.5 to the other algorithms using the *average primal integral*, as introduced in (Berthold 2013) to measure the anytime behavior of optimization solvers. The primal integral aims to measure the progress of an algorithm's primal bound convergence toward the optimal (or best known) solution over the entire solving time. It is based on the *primal function* $p(t)$, which represents the gap between the current solution $x(t)$ at time t and the optimal or best known solution x_{opt}. The *primal gap* of a solution $x(t)$ is defined as

$$\gamma(x(t)) = \frac{|x(t) - x_{\text{opt}}|}{|x(t)|}.$$

The function $p(t)$ equals 1 if no feasible solution has been found by time t, and $\gamma(x(t))$ otherwise. The function $p(t)$ is a step function that changes whenever a new feasible solution is found. It is monotonically decreasing and becomes

[1]https://scikit-learn.org/

zero once the optimal solution is reached. The primal integral $P(T)$ is defined as the integral of the primal gap function $p(t)$ over the time horizon T:

$$P(T) = \int_0^T p(t)\,dt = \sum_{i=1}^{n} p(t_{i-1}) \cdot (t_i - t_{i-1}),$$

where each t_i denotes a time point at which a new incumbent solution is found. The primal integral encourages the early discovery of high-quality solutions. If a better solution is found at the same time, $P(t_{\max})$ decreases. Similarly, if the same solution is found earlier, $P(t_{\max})$ also decreases. The ratio $P(t_{\max})/t_{\max}$ can be interpreted as the average quality of the solution during the search process. A smaller value indicates a higher expected quality of the current solution if the algorithm is interrupted at an arbitrary point in time.

Tables 2 and 3 report the evolution of the average primal integral across various time budgets (from 15 to 300 seconds) for tree depths 6 and 7. To ensure meaningful comparisons, we exclude datasets where DL8.5 finds the optimal solution in under 1 second. For the Gain, Discrepancy, and Top-k rule strategies, we evaluate three approaches to relax the rules between restarts: *Monotonic*, where the threshold is increased by a fixed amount (set to 1 in our experiments); *Exponential*, where the threshold is multiplied by a constant factor (2 in our experiments). We also use *Luby*, where the increment follows the Luby sequence from (Lorenz 2021). Across both depths, all complete anytime strategies outperform DL8.5, highlighting the benefits of rule-based restarting. The best overall results are achieved by CA-Discrepancy (Luby and Monotonic) and CA-Top-k^*, especially under longer time budgets. At depth 6, CA-Discrepancy with monotonic relaxation achieves the lowest average primal integral of 7.4 at 300s, while CA-Top-k^* reaches 7.7.

Under short timeouts (15–30s), Blossom produces high quality early solutions, often outperforming CADL8.5 variants. However, CADL8.5 quickly catches up and surpasses Blossom as runtime increases. This trend becomes more pronounced at depth 7 (Table 3), where CA-Discrepancy with Monotonic and Luby relaxation achieves average primal integral values of 14.0, better than Blossom's 17.2 at 300s. These improvements are largely due to the increased diversification in its search strategy, exploring more parts of the search space, whereas Blossom tends to remain focused on optimizing the deeper layers of a specific tree before moving elsewhere.

Among all rules, *Discrepancy* consistently outperforms the others. CA-Top-k and its variant CA-Top-k^* also perform well, particularly at large timeouts. CA-Gain and CA-Purity lag behind: the Gain rule often selects larger subtrees, leading to longer subsearches; Purity may require several ineffective relaxations before contributing to meaningful diversification.

Greedy baselines such as C4.5, Top-3, and Top-5 deliver quick but static solutions. Among them, Top-5 performs best under tight time budgets (15–30s), briefly outperforming DL8.5. However, none of the greedy methods improve their solutions over time.

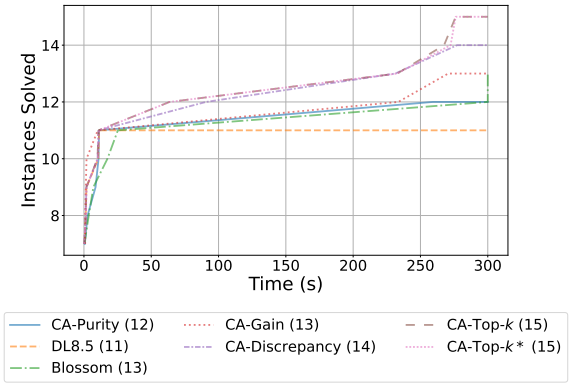

Figure 3: Cumulative number of instances solved as a function of time with the total number of instances solved by each approach within 300s

Figure 3 illustrates the cumulative termination count of each algorithm to find and prove optimality within a time budget of 300s and a depth constraint of 5. Overall, the CADL8.5 variants demonstrate superior solving power compared to DL8.5 and Blossom. Notably, CA-Top-k^* solves the most instances (15 out of 25) and does so more quickly than the other methods across most of the timeline. CA-Discrepancy and CA-Top-k also perform strongly, solving 14 and 15 instances respectively, and surpassing other approaches beyond the 50-second mark. Blossom shows a steep initial rise, indicating strong early performance, but plateaus sooner than the CADL8.5 variants. CA-Gain achieves a similar final count as Blossom (13 instances) but shows slower progress in the early phase. DL8.5 and CA-Purity underperform both in terms of speed and total solved instances, solving only 11 and 12 datasets respectively. This shows that CADL8.5 variants especially Top-k(*) and Discrepancy does not compromise the ability to reach optimal solutions.

Conclusion

In this paper, we introduced CADL8.5, a complete anytime framework for decision tree learning that generalizes DL8.5, LDS-DL8.5, and Top-k. It combines DL8.5's efficient branch-and-bound pruning and trie-based caching with a restart-based search that progressively relaxes pruning criteria, guided by rule based strategies such as node purity, Information Gain gap, Discrepancy and Top-k. Our experiments show that CADL8.5 variants, especially CA-Top-k^* and CA-Discrepancy, deliver strong anytime performance without sacrificing the ability to reach optimal solutions. They solve more instances to optimality than DL8.5 and Blossom and perform at least on par with greedy approaches, while improving solution quality over time. Future work includes exploring combined rule strategies such as Gain and Discrepancy.

References

Aghaei, S.; Gómez, A.; and Vayanos, P. 2021. Strong optimal classification trees. *arXiv preprint arXiv:2103.15965*.

Aglin, G.; Nijssen, S.; and Schaus, P. 2020. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of AAAI*, volume 34, 3146–3153.

Berthold, T. 2013. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614.

Bertsimas, D.; and Dunn, J. 2017. Optimal classification trees. *Machine Learning*, 106: 1039–1082.

Blanc, G.; Lange, J.; Pabbaraju, C.; Sullivan, C.; Tan, L.; and Tiwari, M. 2024. Harnessing the power of choices in decision tree learning. In *Advances in Neural Information Processing Systems*, volume 36.

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and regression trees*, volume 37. Wadsworth Int. Group.

Demirović, E.; Hebrard, E.; and Jean, L. 2023. Blossom: an anytime algorithm for computing optimal decision trees. In *International Conference on Machine Learning*, 7533–7562.

Demirović, E.; et al. 2022. MurTree: Optimal Decision Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23: 1–47.

Hu, H.; et al. 2020. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In *IJCAI-PRICAI 2020*.

Kiossou, H.; et al. 2022. Time constrained dl8.5 using limited discrepancy search. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 443–459.

Kiossou, H.; et al. 2024. Efficient Lookahead Decision Trees. In *International Symposium on Intelligent Data Analysis*, 133–144.

Lorenz, J.-H. 2021. Restart strategies in a continuous setting. *Theory of Computing Systems*, 65(8): 1143–1164.

Narodytska, N.; et al. 2018. Learning Optimal Decision Trees with SAT. In *IJCAI*, 1362–1368.

Nijssen, S.; and Fromont, E. 2007. Mining optimal decision trees from itemset lattices. In *KDD*, 530–539.

Provost, F. 1993. Iterative weakening: Optimal and near-optimal policies for the selection of search bias. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 749–755.

Quinlan, J. 2014. *C4.5: Programs for machine learning*. Elsevier.

van der Linden, J. G.; Vos, D.; de Weerdt, M. M.; Verwer, S.; and Demirović, E. 2024. Optimal or Greedy Decision Trees? Revisiting their Objectives, Tuning, and Performance. *arXiv e-prints*, arXiv–2409.

Verhaeghe, H.; et al. 2020. Learning optimal decision trees using constraint programming. *Constraints*, 25: 226–250.

Zhang, W. 1998. Complete anytime beam search. In *AAAI/IAAI*, 425–430.