# A Generic Complete Anytime Beam Search for Optimal Decision Tree

Harold Kiossou[1], Siegfried Nijssen[2], and Pierre Schaus[1]

[1] UCLouvain, ICTEAM, Belgium
{first.last}@uclouvain.be
[2] KU Leuven, DTAI, Belgium
siegfried.nijssen@kuleuven.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.

**Keywords:** Optimal Decision Trees · Beam Search · Anytime.

## 1 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 [**?**] and C4.5 [**?**] 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 [?,?], constraint programming [?], SAT [?], and dynamic programming [?,?]. While these methods offer strong generalization properties [?,?], they tend to suffer from poor anytime behavior: when interrupted before convergence, they often return poor-quality solutions.

The DL8.5 algorithm [?], 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.

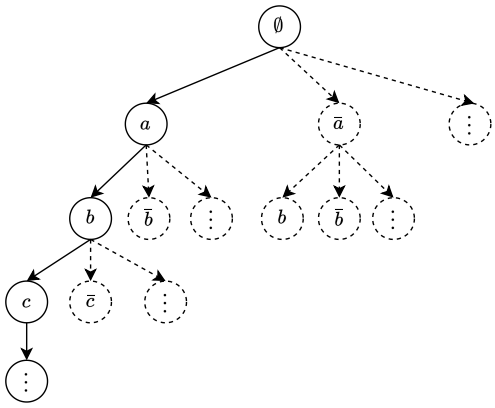


**Fig. 1.** DL8.5 explores the leftmost branches first, often leading to poor anytime performance when interrupted early.

To improve the anytime performance or scalability of exact methods, three notable works have been proposed. LDS-DL8.5 [?] integrates limited discrepancy search (LDS) into DL8.5, resulting in an algorithm that is both anytime and complete. Top-$k$-DL8.5 [?] 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 [?] 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 possibility of highly unbalanced trees when interrupted early as in DL8.5. Blossom is guaranteed to find the optimal tree given sufficient time. Despite their 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 [**?**], 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 [**?**], 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 outperform 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.

## 2 Related Works and Background

*Greedy approaches.* Traditional algorithms like CART [**?**] and C4.5 [**?**] build trees greedily using local information. While efficient, they lack optimality guarantees. Optimization-based approaches using MILP [**?**], SAT [**?**], MaxSAT [**?**], and CP [**?**] provide guarantees but struggle to scale.

*Dynamic Programming Approaches and DL8.5.* Dynamic Programming (DP) methods [**?**,**?**,**?**] improve scalability of exact tree learning. DL8.5 [**?**], the foundation of our work, learns optimal decision trees on binary datasets $\mathcal{D}$ under minimum support and maximum depth constraints. A binary dataset $(\mathcal{D}, \mathcal{C})$ contains examples $\mathcal{D} \subseteq \prod_{f \in \mathcal{F}} \{f, \bar{f}\}$ over Boolean features $\mathcal{F}$, with $\mathcal{C}(x)$ the class of $x$. For $\mathcal{S} \subseteq \mathcal{D}$ and feature $f$, define $\mathcal{S}(f) = \{x \in \mathcal{S} \mid f \in x\}$ and analogously $\mathcal{S}(\bar{f})$. A binary decision tree assigns features to internal nodes and labels edges with $f$ or $\bar{f}$, each root-to-leaf path encoding a conjunction of tests. For a branch $b$, $\mathcal{S}(b) = \bigcap_{f \in b} \mathcal{S}(f)$, and its classification error is: $|\mathcal{S}(b)| - \max_{c \in \mathcal{C}} |\mathcal{S}_c(b)|$.

DL8.5 explores the search space via a depth-first traversal of an *OR–AND* search tree. At each OR node, a feature is selected and the two corresponding AND nodes ($\{f, \bar{f}\}$) are explored in order: first the examples satisfying

the feature, then the remaining ones. This recursive filtering naturally enforces minimum-support and maximum-depth constraints, and the errors of the two subtrees are combined to compute the node's error. Efficiency comes from upper-bound pruning, which discards branches that cannot improve the current best solution, and from a trie-based cache that avoids recomputing identical subproblems, together greatly reducing the search space.

*Anytime complete approaches.* DL8.5's depth-first search prioritizes leftmost branches (Figure 1), potentially delaying exploration of promising subtrees. With many features or large depths, the search may remain confined to early subtrees. If interrupted, DL8.5 yields unbalanced trees that may underperform greedy heuristics.

LDS-DL8.5 [**?**] applies iterative Limited Discrepancy Search to prioritize trees close to a greedy baseline. A discrepancy is choosing a feature differing from the greedy choice. Gradually increasing allowed discrepancies transitions from greedy to exhaustive search, identifying good trees early. Blossom [**?**] expands nodes layer by layer, processing the non-expanded node closest to the root. Its first solution is the greedy tree, improving anytime performance. However, it may exhibit poor diversification near the root. No empirical comparison exists between Blossom and LDS-DL8.5.

*Anytime incomplete approaches.* Top-$k$ [**?**] restricts branching to the $k$ highest-ranked features at each OR node, trading off between greedy C4.5 and full DL8.5 search.

LGDT [**?**] combines greedy strategies with limited-depth lookahead, improving quality without exhaustive search cost.

## 3 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 [**?**] introduced Complete Anytime Beam Search (CABS), based on the principle of *iterative weakening* [**?**].

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 CABS 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 1 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 quickly finding initial solutions and ensuring optimality through complete exploration.

**Table 1.** Rule definitions for CADL8.5

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

CADL8.5 extends DL8.5 with an outer loop (Lines 6–11) that repeatedly invokes `BeamDL8.5` while progressively relaxing constraints. Starting with strict pruning to rapidly find a feasible solution and establish an upper bound, the `relax` function weakens rules after each iteration, expanding the search space. This continues until a perfect tree is found, no further relaxation is possible, or time 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:

- `update`$(t : \texttt{T}, c : \texttt{Context}) \rightarrow$ `T`: modifies a node's state based on its context, including error metrics, dataset size, and feature selection information.
- `prune`$(t : \texttt{T}, r : \texttt{R}) \rightarrow$ `boolean`: evaluates whether a node should be pruned based on its current state and rule parameters.

**Algorithm 1:** Complete Anytime DL8.5 (CADL8.5)

```
   Input  : 𝒟, rule, minsup, maxdepth
   Output: Best tree satisfying minsup and maxdepth
 1 Struct Best{error : float, ub : float, tree : Tree, state : T}
 2 cache ← Trie < branch, Best >
 3 ub ← +∞
 4 c_0 ← { i: 0, e: error(∅), s: |𝒟| }
 5 s_0 ← initial_state(c_0)
 6 do
 7     sol ← BeamDL8.5(∅, ub, c_0, s_0)
 8     ub ← sol.error
 9     rule ← relax(rule)
10     s_0 ← sol.state
11 while sol is not optimal or rule is relaxable
12 return sol.tree
13 Procedure BeamDL8.5(b, ub, c_p, s_p)
14     e ← error(b)
15     if |b| = maxdepth or e = 0 then
16         return Best{e, ub, leaf(b), terminal_state(s_p)}
17     if time limit reached or prune(s_p, rule) then
18         return Best{e, ub, leaf(b), s_p}
19     node ← cache.get(b)
20     if node ≠ ∅ and ub ≤ node.ub and ¬prune(node.state, rule) then
21         return node
22     τ ← ∅, child_ub ← ub, optimal ← true
23     foreach (i, f) in ℱ sorted by heuristic do
24         if |𝒟(b ∪ {f})| < minsup or |𝒟(b ∪ {f̄})| < minsup then
25             continue
26         c_l ← { i: i, e: error(b ∪ {f̄}), s: |𝒟(b ∪ {f̄})| }
27         s_l ← update(s_p, c_l)
28         left ← BeamDL8.5(b ∪ {f̄}, child_ub, c_l, s_l)
29         if left.tree = ∅ then
30             continue
31         c_r ← { i: i, e: error(b ∪ {f}), s: |𝒟(b ∪ {f})| }
32         s_r ← update(s_p, c_r)
33         right ← BeamDL8.5(b ∪ {f}, child_ub − left.error, c_r, s_r)
34         if right.tree = ∅ then
35             continue
36         e ← left.error + right.error
37         if prune(left.state, rule) or prune(right.state, rule) then
38             optimal ← false
39         if e < child_ub then
40             child_ub ← e
41             τ ← Tree(left.tree, right.tree)
42             c_p.error ← e
43             s_p ← update(s_p, c_p)
44         if e = 0 then
45             break
46     if optimal then
47         s_p ← terminal_state(s_p)
48     node ← Best{child_ub, ub, τ, s_p}
49     cache.save(b, node)
50     return node
```

– `relax`($r$ : `R`) → `R`: incrementally weakens rule constraints to permit wider exploration in subsequent iterations.

- `terminal_state`($t$ : `T`) $\rightarrow$ `T`: generates a special state marking a node as fully explored and exempt from future pruning.
- `initial_state`($c$ : `Context`) $\rightarrow$ `T`: creates the initial state for the root node based on its context.

The search begins by constructing an initial context $c_0$ at the root using the dataset's error and size. This context is passed to `initial_state` to generate the root's initial state $t_0$ (Line 5), which starts the first `BeamDL8.5` iteration with the initial upper bound.

During `BeamDL8.5`, each node's state is evaluated before expansion. The `update` function computes an updated state $(s_l, s_r)$ based on the node's context (Lines 27 and 32), which is then evaluated using `prune` (Line 17) to check if the node violates active constraints. If so, the node is not expanded.

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 compatible cached entry exists—one whose upper bound is consistent with the 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.

### 3.1 Purity rule

The purity rule defines a minimum purity threshold in `R`. Node expansion stops when purity meets or exceeds this threshold. Weakening incrementally increases the threshold by $\delta$ until 1.0. If purity remains below the threshold, search continues until maximum depth. The purity of branch $b$ is $purity(b) = 1 - error(b)/|\mathcal{S}(b)|$.

### 3.2 Gain rule

The gain rule restricts feature expansion using an information gain threshold. At a node, let $\tau^*$ be the highest gain and $\tau(f)$ the gain of feature $f$. The cumulative gap is $\texttt{cum\_gap} = \texttt{cum\_gap}_{\text{parent}} + (\tau^* - \tau(f))$. A feature expands only if $\texttt{cum\_gap} \leq \texttt{max\_gap}$. Setting $\texttt{max\_gap} = 0$ yields greedy C4.5-like behavior

**Table 2.** Average primal integral on depth 6

| 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 |

while larger values allow broader exploration. The cumulative nature of the constraint enforces a stricter selection in deeper parts of the tree, preventing getting stuck in lower parts of the search space.

### 3.3 Discrepancy rule

The Discrepancy rule employs the same principle of Limited Discrepancy Search (LDS) as in the LDS-DL8.5 algorithm [**?**], 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 `max_discr` $= 0$) results in a greedy tree. Increasing the discrepancy budget expands the search space and enables exploring other parts of the search space.

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 $cost(b) = cost(\bar{b}) = 1$.

**Table 3.** Average primal integral on depth 7

| 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 |

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.

### 3.4 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. 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 reducing the time spent in the deeper parts of the search space in early iterations.

## 4 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

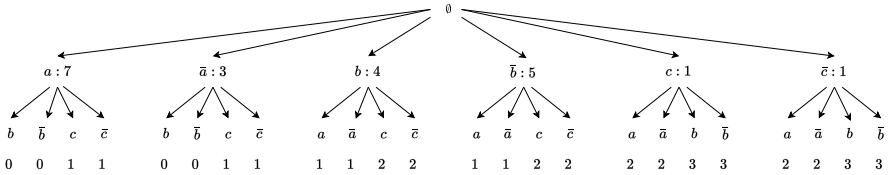


**Fig. 2.** Search tree for three features. Values at depth 1 show node error before expansion; at depth 2, discrepancy costs.

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[3], DL8.5, and Blossom implementations. We compare CADL8.5 using the *average primal integral* [**?**], which measures anytime behavior by tracking the primal bound's convergence toward the optimal solution. The *primal gap* of solution $x(t)$ at time $t$ is $\gamma(x(t)) = |x(t) - x_{\text{opt}}|/|x(t)|$. The primal function $p(t)$ equals 1 if no solution is found by time $t$, and $\gamma(x(t))$ otherwise. The primal integral is $P(T) = \int_0^T p(t)\,dt = \sum_{i=1}^{n} p(t_{i-1}) \cdot (t_i - t_{i-1})$, where $t_i$ denotes when a new solution is found. The ratio $P(t_{\max})/t_{\max}$ represents average solution quality during search; smaller values indicate better anytime performance.

Tables 2 and 3 show average primal integral evolution across time budgets (15-300s) for depths 6 and 7, excluding datasets where DL8.5 solves in under 1s. We evaluate three relaxation strategies: *Monotonic* (threshold increased by 1), *Exponential* (multiplied by 2), and *Luby* [**?**]. All complete anytime strategies outperform DL8.5. CA-Discrepancy (Luby/Monotonic) and CA-Top-$k^*$ achieve best results, especially at longer timeouts. At depth 6, CA-Discrepancy with monotonic relaxation achieves 7.4 at 300s, while CA-Top-$k^*$ reaches 7.7.

Under short timeouts ($15 - 30$s), 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,

[3] `https://scikit-learn.org/`

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.

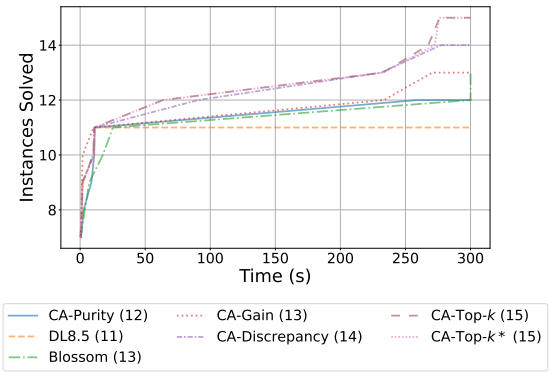


**Fig. 3.** Cumulative number of instances solved as a function of time with the total number of instances solved by each approach within 300s

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 do not compromise the ability to reach optimal solutions.

## 5 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.