# Lightweight Look-Ahead-Based Value Heuristics for Constraint Optimization Problems

**Ziyang Yu**
School of Information Science and Technology, Northeast Normal University, Changchun, China

**Hongbo Li**[1]
School of Computing and Artificial Intelligence, Shanghai University of Finance and Economics, Shanghai, China

## Abstract

The *Bound-Impact Value Selector* (BIVS) employs a look-ahead strategy to enable black-box Constraint Optimization Problem (COP) solvers to find high-quality solutions earlier. However, its computational cost prohibits its use throughout the entire search process. To mitigate this cost, the *Restricted Fixpoint* (RF) approach considers only the constraints on the shortest paths between the selected variable and the objective, yielding better performance. In this paper, we propose a lightweight strategy from a different perspective, named *Assess Before Look-Ahead* (ABLA), to enhance the performance of look-ahead-based value heuristics for solving COPs. ABLA first assesses whether the look-ahead process can differentiate between the values of a variable, and only performs the look-ahead when this assessment passes. Experiments on benchmark instances from recent MiniZinc Challenges demonstrate that ABLA's decisions to skip redundant look-ahead processes are highly reliable, with an average accuracy of over 94%. Consequently, ABLA significantly boosts the performance of both BIVS and RF, outperforming two other baselines: the minimum value heuristic and the RLARF value heuristic.

**2012 ACM Subject Classification** Theory of computation → Constraint and logic programming

**Keywords and phrases** Constraint Optimization Problem, Value Heuristic, Look-Ahead

**Digital Object Identifier** 10.4230/LIPIcs.CP.2026.59

**Supplementary Material** *Software (Source Code)*: `https://github.com/lihb905/abla`

**Funding** The National Natural Science Foundation of China under Grant 62276060 and CCF-Huawei Populus Grove Fund

**Acknowledgements** We thank the anonymous referees for their constructive comments.

## 1 Introduction

Constraint programming (CP) is a powerful paradigm for solving discrete combinatorial optimization problems. Under the Holy Grail of CP [9], the users model real-world problems as constraint optimization problems (COP) and then apply a constraint solver to solve the COPs. A COP is NP-hard in its general form. To solve COPs, branch-and-bound search (B&B) explores the search space with a depth-first backtracking search resulting in a search tree. Developing effective general-purpose search strategies is essential to achieving the Holy Grail of CP. While some search strategies directly select a variable-value pair to make a decision at each search tree node [18, 19], a more popular decision-making process proceeds in two steps: an unassigned variable is selected first, typically using a variable ordering heuristic [4, 13], and then a value heuristic [8, 6] is employed to select a value from the domain of the variable. Over the past decades, numerous general-purpose variable ordering heuristics

[1] corresponding author

have been proposed [4, 21, 16, 10, 13, 1, 17], leading to significant progress. In contrast, general-purpose value heuristics remain relatively scarce, so the simplest minimum-value (MV) heuristic is still widely used in constraint solvers. Recently, several value heuristics based on machine learning techniques have been introduced [5, 15]. However, due to their reliance on large amounts of training data, their application in black-box constraint solvers remains limited.

To enable black-box COP solvers to find high-quality solutions earlier, the *Bound-Impact Value Selector* (BIVS) [8] employs a look-ahead strategy. For each candidate value of a selected variable, it propagates the assignment and selects the value that minimizes (or maximizes) the objective bound for minimization (or maximization) problems. However, the look-ahead process in BIVS is computationally expensive, making it less efficient to apply throughout the entire branch-and-bound search. To address this efficiency, a smart lightweight approach called *Restricted Fixpoint* (RF) [6] has been proposed, which propagates only the constraints along the shortest paths between the selected variable and the objective variable to estimate the impact of each value. Although RF is less informative than BIVS, it reduces the computational cost of the look-ahead process and strikes a good balance between informativeness and efficiency. As a result, RF enhances the overall performance of BIVS.

In this paper, we propose a lightweight look-ahead strategy from a different perspective, namely *Assess Before Look-Ahead* (ABLA), to enhance the performance of look-ahead-based value heuristics for solving constraint optimization problems. The look-ahead process in BIVS and RF iterates over all values in the domain of the selected variable, propagating each instantiation to evaluate its impact on the objective variable, and selects a value based on these impacts. However, in certain scenarios, look-ahead-based heuristics may fail to distinguish between candidate values, consequently selecting values identically to the simplest minimum-value (MV) heuristic. The main contribution of this paper is to avoid unnecessary iterations and propagations during impact evaluation. The key idea is to first examine the impacts of the lower and upper bounds of the selected variable to determine whether a full iteration is worthwhile. If the assessment indicates that the look-ahead is unlikely to be informative, the algorithm directly returns the minimum value. Based on this idea, we propose two variants of the lightweight look-ahead strategy, namely ABLA and ABLA2. The main difference between them lies in how they handle infeasible values detected during the assessment. While ABLA directly returns the infeasible value detected at either the lower bound or the upper bound, ABLA2 directly returns only the infeasible value if it is detected at the first bound checked, e.g., the lower (upper) bound for a minimization (maximization) problem. We performed extensive experimentation on the benchmark instances used in recent MiniZinc Challenges. The results show that the lightweight frameworks substantially improve the performance of both BIVS and RF. The decisions made by the lightweight strategies to skip redundant look-ahead processes are highly accurate. When the lower and upper bounds of a variable have identical impacts on the objective, the intermediate values are likely to have the same impact. The ABLA2+BIVS value heuristic outperforms the existing value heuristics under different variable ordering heuristics.

## 2 Background

A *Constraint Optimization Problem* (COP) $\mathcal{P}$ is a 4-tuple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F} \rangle$, where $\mathcal{X}$ is a set of variables, the domain $dom(x) \in \mathcal{D}$ specifies the set of possible values for each $x \in \mathcal{X}$, $\mathcal{C}$ is a set of constraints and $\mathcal{F}$ is an objective function. Each constraint $c \in \mathcal{C}$ specifies the allowed combinations of values for a subset of variables. An *assignment* $\mathcal{A}$ to variables $X \subseteq \mathcal{X}$ is a

set of *instantiations* of the form $x = v$, one for each $x \in X$ to assign $v$ to $x$ where $v \in dom(x)$. If $X = \mathcal{X}$, then $\mathcal{A}$ is a *complete assignment*; otherwise a *partial assignment*. A complete assignment $S$ that satisfies all constraints is a *feasible solution* to $\mathcal{P}$. The objective function $\mathcal{F}$ maps every feasible solution $S$ of $\mathcal{P}$ to a real number $\mathcal{F}(S)$, called the objective. Without loss of generality, we consider here minimization and a feasible solution $S^*$ is an *optimal solution* if $\mathcal{F}(S^*) \leq \mathcal{F}(S)$ for any other feasible solution $S$ to $\mathcal{P}$. In case of maximization, optimal solutions are defined simply with $\leq$ replaced by $\geq$.

A COP $\mathcal{P}$ can be solved by *branch-and-bound search* (B&B) augmented with constraint propagation [3]. This work focuses on value heuristics for B&B, so we skip the details of constraint propagation and always adopt the solver's default propagation engine. B&B explores the search space with a depth-first backtracking search, resulting in a search tree. The main idea is to use the last best feasible solution found as an upper (or lower) bound to help prune the search space of minimization (or maximization) problems. When no more solutions can be found, optimality is proved and the last feasible solution found can be returned as the optimal solution of the COP.

In backtracking search, binary branching [11] is more powerful than $k$-way branching, so it is widely used in constraint solvers and we consider binary branching in this study. Binary branching has two types of decisions: positive decisions (e.g., an instantiation $x_i = v_i$) and negative decisions (e.g., the refutation $x_i \neq v_i$). At each search tree node, an unassigned variable is selected and a value is selected from the domain of the variable. A new node will be generated after the instantiation to this variable, then a propagation algorithm is applied to filter those inconsistent values [3]. If a positive decision leads to a failure (such as a domain wipeout), the corresponding negative decision is generated immediately. Both types of decisions reduce the domain of the variable, so constraint propagation is applied after both types of decisions. If the propagation of a decision leads to a failure, then the decision must be canceled, and a backtracking occurs. If a failure is detected at the root node, it indicates that either optimality is proved or unsatisfiable is proved, then the search completes.

There are two primary types of search heuristics in backtracking search: variable ordering heuristics [4, 13] and value heuristics [8, 6]. As mentioned before, variable ordering heuristics are used to select an unassigned variable at each search tree node, and value heuristics are used to select a value to assign from the domain of the variable. This study focuses on value heuristics for COPs, so we review the details of some related value heuristics in the next section.

## 3 Related Work

The most relevant methods to our work are the *Bound-Impact Value Selector* [8] and its lightweight variant, *Restricted Fixpoint* [6]. Both heuristics operate within a common framework. At each search tree node, once an unassigned variable $x$ is selected, they iterate over all values in $dom(x)$ and evaluate each value by examining its impact on the objective variable. Specifically, they propagate the instantiation of each value on a copy of the COP. This propagation may reduce the domain of the objective variable, and the heuristic selects the value that yields the best bound, e.g., the smallest lower bound in minimization problems or the largest upper bound in maximization problems. Our work aims to improve this framework by skipping redundant look-ahead processes.

While BIVS propagates instantiations across the entire constraint set $C$, RF restricts propagation to a subset of $C$. Before searching starts, RF constructs a bipartite variable-constraint graph, where variables and constraints form two disjoint vertex sets, and an edge

connects a variable to a constraint if the variable appears in that constraint. The shortest paths from each variable to the objective variable are precomputed. When evaluating the values of a variable, RF only propagates constraints along the shortest paths. As a result, it reduces the computation cost of propagating the instantiation of each value. This lightweight variant offers a trade-off between computational efficiency and informativeness, and has been shown to enhance the overall performance of BIVS.

It is worth noting that *Solution-Based Phase Saving* (SBPS) [7] is not a standalone value heuristic but rather a preference that builds upon an underlying one. For a given variable $x$, SBPS will first attempt to assign it the value it held in the most recently found solution. This strategy only becomes active after a first solution has been discovered. If no solution has been found yet, or if the preferred value is no longer in the current domain of $x$, SBPS simply delegates the selection to its underlying heuristic, such as BIVS, RF, or the strategy proposed in this work.

## 4 A Lightweight Look-Ahead Strategy

In this section, we first provide a detailed review of the look-ahead framework employed by BIVS and RF. We then discuss the motivation behind our approach before presenting the proposed method. In the following, we assume that values in each variable domain are sorted in increasing order and, without loss of generality, consider minimization problems as representative. Several functions used in the subsequent pseudocode are defined as follows:

- the $lb(x)$ returns the lower bound of the current $dom(x)$;
- the $ub(x)$ returns the upper bound of the current $dom(x)$;
- the $nextValue(x, v)$ returns the next value of $v$ in current $dom(x)$ and it returns $null$ if $v$ is the last one;
- the $\mathcal{P}.obj$ is the objective variable of a COP $\mathcal{P}$;
- the $propagate(\mathcal{P}, x, v)$ propagates the instantiation $x = v$ in $\mathcal{P}$ and the lower bound and upper bound of $obj$ may be updated by the propagation. If the propagation leads to a failure, then it sets the two bounds of $obj$ to $\infty$.

### 4.1 The Framework of BIVS

Algorithm 1 illustrates the working mechanism of the BIVS framework. The algorithm begins by recording the current state of the COP (line 4). It then iterates over all values in the domain of the selected variable $x$ (lines 5–11). For each value, it propagates the corresponding instantiation on a copy of the current COP state and evaluates the resulting lower bound of the objective variable. Finally, it returns the value that minimizes this lower bound.

*Restricted Fixpoint* is obtained by a simple modification to BIVS. For each variable, it precomputes the shortest paths from that variable to the objective variable. Let $C_{sp}(x)$ denote the set of constraints along the paths. RF then replaces the full constraint set $C$ with $C_{sp}(x)$ when recording the COP state at line 4 of Algorithm 1. In other words, only the constraints in $C_{sp}(x)$ are considered during state recording and subsequent propagation.

### 4.2 The Motivation

The look-ahead strategy employed by BIVS and RF provides valuable guidance for value selection, but its primary limitation lies in the associated computational cost. Ideally, computational effort should be invested in evaluating all candidate values only when such evaluation yields meaningful information. However, the look-ahead strategy fails to provide

**Algorithm 1** The Framework of BIVS.

**Input:** A COP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F} \rangle$; the selected variable $x$.
**Output:** an integer $v$.

```
1  bestValue ← lb(x);
2  bestLB ← ∞;
3  v ← lb(x);
4  P ← current state of ⟨X, D, C⟩;
5  while v ≠ null do
6      P' ← makeCopyOf(P);
7      propagate(P', x, v);
8      if lb(P'.obj) < bestLB then
9          bestValue ← v;
10         bestLB ← lb(P'.obj);
11     v ← nextValue(x, v);
12 return bestValue;
```

discriminative information when propagating any of the instantiations leads to identical bounds on the objective variable. In such scenarios, the look-ahead process may consume substantial time to evaluate all values, ultimately selecting the minimum value, i.e., a choice that could have been made trivially by the near-zero-cost MV heuristic. This observation motivates our work: we aim to assess, prior to performing a full look-ahead, whether the process is likely to yield discriminative information for value selection.

If there is no path from the selected variable $x$ to the objective variable *obj* in the bipartite variable-constraint graph, then propagating an instantiation of $x$ cannot affect *obj*, and the look-ahead strategy will fail to produce discriminative information. This issue has already been addressed by the RF approach. Beyond this structural case, it is difficult to predict from the constraint network alone whether the propagation of a variable will affect the objective variable. For example, a scenario where $x$ and *obj* are loosely connected, i.e., there are few paths between them, and the constraints along those paths are very loose. In such a context, the propagation of $x$ has little chance of affecting *obj*. However, as the search progresses and more variables become assigned, these initially loose connections may tighten, potentially enabling $x$ to influence *obj*. This dynamic nature suggests that a static structural analysis is insufficient; instead, a real-time assessment should be performed before each look-ahead process to determine whether it is worthwhile.

## 4.3 Assess Before Look-Ahead

In the following, propagating a value $v$ of variable $x$ refers to assigning $x = v$ and propagating the effects of this instantiation through the constraint network.

To identify redundant look-ahead processes, we propose a new strategy that examines the impact of propagating the lower bound $lb(x)$ and upper bound $ub(x)$ on the objective variable. The core idea is to skip the full look-ahead if the impacts of the two bounds are identical. We name this method *Assess Before Look-Ahead* (ABLA), and its pseudocode is presented in Algorithm 2.

Since propagating $lb(x)$ or $ub(x)$ may lead to failures, and processing conflicts with priority may benefit the search. Thus, if either propagation fails (detected at line 5 or line 11), the corresponding value is returned directly at line 6 or line 12 and the searching engine of

binary branching will process the failure immediately, i.e., the right branch will be generated, and the refutation is propagated immediately. If both propagations succeed, we compare the bounds of the objective variable after each propagation (line 15). The variables *leftLB* and *leftUB* record the lower and upper bounds of the objective variable after propagating $x = lb(x)$ (lines 7–8), while *rightLB* and *rightUB* record those after propagating $x = ub(x)$ (lines 13–14). If the two lower bounds are equal and the two upper bounds are also equal, the look-ahead process is considered redundant and is skipped, and $lb(x)$ is returned. Otherwise, if the bounds differ, the LookAhead procedure in Algorithm 3 is invoked to evaluate all values in the domain.

**Algorithm 2** Assess Before Look-Ahead.

**Input:** A COP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{F} \rangle$; the selected variable $x$.
**Output:** an integer $v$.

```
1  leftLB, leftUB, rightLB, rightUB ← ∞;
2  P ← current state of ⟨X, D, C⟩;
3  P' ← makeCopyOf(P);
4  propagate(P', x, lb(x));
5  if lb(P'.obj) = ∞ then
6      return lb(x);
7  leftLB ← lb(P'.obj);
8  leftUB ← ub(P'.obj);
9  P' ← makeCopyOf(P);
10 propagate(P', x, ub(x));
11 if lb(P'.obj) = ∞ then
12     return ub(x);
13 rightLB ← lb(P'.obj);
14 rightUB ← ub(P'.obj);
15 if leftLB = rightLB and leftUB = rightUB then
16     return lb(x);
17 return LookAhead(P, x, leftLB, rightLB);
```

The LookAhead procedure takes as input the selected variable, along with the corresponding lower bounds of the objective variable after propagating these two values. Before evaluating the values between $lb(x)$ and $ub(x)$, it initializes *bestValue* to the $lb(x)$ and records the associated best lower bound (*bestLB*) (lines 1–2). It then iterates over the values between $lb(x)$ and $ub(x)$ (lines 4–10) to find the one that yields the best objective bound. Finally, it compares the selected value with the $ub(x)$ (lines 11-12), and returns the better one. In Algorithm 3, there exists a case where $leftLB = rightLB$. This occurs because in Algorithm 2, even if $leftLB = rightLB$, the LookAhead procedure is still invoked when $leftUB \neq rightUB$. The workflow of Algorithm 3 ensures that, in such cases where $leftLB = rightLB$, if no intermediate value yields a better objective bound, the lower bound $lb(x)$ is preferred.

As mentioned in [8], propagating the removal of infeasible values immediately may delay branching on more promising values. So we propose a revised version of ABLA, called ABLA2. The key modification is to postpone the handling of certain failures detected during the assessment phase. Specifically, ABLA2 first evaluates the impact of $lb(x)$; if $lb(x)$ is infeasible, it is returned immediately. If $lb(x)$ is feasible but $ub(x)$ is infeasible, the upper bound is not returned directly; instead, this case is treated as having different bounds, and the

**Algorithm 3** LookAhead.

**Input:** A COP $\mathcal{P}$; the selected variable $x$; integers *leftLB*, *rightLB*.
**Output:** an integer $v$.

```
1  bestLB ← leftLB;
2  bestValue ← lb(x);
3  v ← nextValue(x, lb(x));
4  while v < ub(x) do
5      P' ← makeCopyOf(P);
6      propagate(P', x, v);
7      if lb(P'.obj) < bestLB then
8          bestValue ← v;
9          bestLB ← lb(P'.obj);
10     v ← nextValue(x, v);
11 if rightLB < bestLB then
12     bestValue ← ub(x);
13 return bestValue;
```

LookAhead procedure is invoked. If both bounds are feasible, ABLA2 behaves identically to ABLA. As a result, ABLA2 can be obtained from Algorithm 2 by removing lines 11–12. The intuition behind this design is as follows. When $lb(x)$ is feasible and $ub(x)$ is infeasible, immediately returning the infeasible upper bound would cause $lb(x)$ to be re-evaluated the next time the variable is selected. If multiple infeasible values exist near the upper bound, this could lead to $lb(x)$ being re-evaluated repeatedly, reducing efficiency.

It can be seen that both ABLA and ABLA2 perform at most $O(d)$ propagations in the worst case, where $d$ is the current domain size of $x$. This is because the assessment phase checks at most two values, and the LookAhead procedure evaluates at most $O(d)$ values. For maximization problems, the algorithms can be adapted in a straightforward manner: during assessment, we always check the upper bound first, followed by the lower bound; in the look-ahead phase, we select the value that maximizes the upper bound of the objective variable.

## 5 Experiments

To examine the performance of the lightweight look-ahead strategies in solving COPs, we performed extensive experimentation on the COP instances of the recent two MiniZinc challenges. The following are some general settings in the experiments:

- **Environment.** The environment is Ubuntu 22.04 with Intel(R) Xeon(R) Platinum 8360H CPUs @3.0GHz and 512G RAM.
- **Solver.** We have used Choco (version 4.10.18), an efficient and well-architected CP solver [20].
- **Resource.** 60 test runs are running simultaneously, and each run is allocated 1 core and 8 GB RAM.
- **Instances.** There are 200 instances from 40 problems involved in the recent two MiniZinc Challenges (2024-2025). The instances are flattened offline. This study focuses on COPs defined with integer variables, so we eliminate the instances defined with set variables, the CSP instances, and some instances that failed to flatten. The remaining 173 instances from 35 problems are used in the experiments.

- **Restart.** The restart condition is the number of failures. All the compared search strategies are equipped with the default restart strategy in Choco, e.g., a Luby [14] restart strategy with an initial cutoff of 500 and a maximum restart number of 50000.
- **Nogood-Recording.** All the compared search strategies are equipped with the default *Nogood-Recording* technique built in Choco, i.e. the *Reduced nld-Nogoods* [12].
- **Variable ordering heuristics.** We have used four efficient variable ordering heuristics, including activity-based search (ABS) [16], the refined weighted degree combined with minimum domain (WDEG) [22], the pick/dom heuristic[2] (Pick) [1], and the refined version of failure-based heuristic (FRBA4) [23].
- **Random seed.** The random seed 0 is used throughout the experiments, mainly used to break ties in variable ordering heuristics.
- **Timeout.** The timeout is set to 3600 seconds, and we also present the results within 1800 seconds for reference.
- **An implementation detail.** Following the implementation details of BIVS, if the variable has a domain defined with an interval or an enumerated set with size larger than 100, then we check only the lower bound and upper bound of the variables and always skip the look-ahead process.
- **Metrics.** We employ the primal gap [2] to evaluate the overall performance of the value heuristics, which considers both finding feasible solutions and finding solutions with better objectives. For each tested instance[3], it gives a score between 0 and 1 to each candidate value heuristic defined as follows: if the candidate did not find a feasible solution, it gets 1 score; otherwise, it gets a score $\frac{|obj_{best}-obj|}{max\{|obj_{best}|,|obj|\}}$, where $obj_{best}$ is the objective of the best solution found by all candidates and $obj$ is the objective of the solution found by current candidate. The smaller score indicates a better performance. In addition, we involve the number of instances where a feasible solution is found (#Solved) and the number of instances where optimality is proved (#Optimum).

In the following tables, the best one in each comparison is highlighted in bold. The primal gap values are accumulated over all instances for which at least one of the tested value heuristics finds a feasible solution. The following value heuristics are evaluated:

- **MV.** It always selects the lower bound of $x$.
- **BIVS [8].** The original BIVS. We have used the implementation built in Choco.
- **RF [6].** The original *Restricted Fixpoint*, marked by BIVS+RF in [6].
- **RLARF [6].** The value heuristic combining RF with the reverse look-ahead strategy (RLA). For RF and RLARF, we have used the implementation shared by the authors, which was implemented in Choco.
- **ABLA + BIVS.** The lightweight BIVS using our ABLA strategy.
- **ABLA2 + BIVS.** The lightweight BIVS using our ABLA2 strategy.
- **ABLA + RF.** The lightweight RF using our ABLA strategy.
- **ABLA2 + RF.** The lightweight RF using our ABLA2 strategy.

To begin with, we compared the four variable ordering heuristics under the existing value heuristics to find more powerful baselines. The results are presented in Table 1. We can see that FRBA4 and Pick perform much better than WDEG and ABS, outperforming them by

[2] There are two implementations of the pick/dom heuristic in Choco. We tested them and have used the one performing better, e.g., PickOnFil.

[3] If none of the compared methods finds a feasible solution for an instance, then the instance is not counted.

a large margin on all three metrics. Thus, we use FRBA and Pick as the variable ordering heuristics in the following. We also observe that the BIVS value heuristic performs better than the other existing value heuristics in primal gap and finding feasible solutions, however, RF and RLARF prove optimality for more instances than the others in 3600 seconds.

**Table 1** Comparison of variable ordering heuristics for finding more powerful baselines. The primary purpose of this table is to compare the performance of different variable ordering heuristics; therefore, the best result in each column (for each metric) is highlighted.

| Metrics | Time | 1800s | | | | 3600s | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Heuristics | MV | BIVS | RF | RLARF | MV | BIVS | RF | RLARF |
| Primal Gap | ABS | 33.47 | 25.89 | 29.92 | 34.54 | 30.34 | 25.06 | 28.49 | 32.14 |
| | WDEG | 46.08 | 45.40 | 43.07 | 46.57 | 45.51 | 43.73 | 42.02 | 45.81 |
| | Pick | 29.16 | 22.87 | **24.18** | 28.86 | 26.00 | **17.21** | **22.08** | 25.06 |
| | FRBA4 | **28.41** | **22.39** | 25.18 | **26.98** | **25.22** | 18.21 | 23.34 | **24.46** |
| #Solved | ABS | 150 | 150 | 150 | 148 | 151 | 150 | 151 | 150 |
| | WDEG | 145 | 136 | 145 | 144 | 145 | 137 | 145 | 144 |
| | Pick | **156** | **155** | **157** | **156** | **157** | **159** | **158** | **158** |
| | FRBA4 | 154 | **155** | 154 | 155 | 156 | 156 | 155 | 156 |
| #Optimum | ABS | 53 | **50** | 52 | 48 | 56 | 52 | 54 | 51 |
| | WDEG | 44 | 42 | 45 | 43 | 46 | 44 | 47 | 46 |
| | Pick | 55 | 47 | 52 | 53 | 58 | 53 | 56 | 55 |
| | FRBA4 | **57** | **50** | **56** | **56** | **60** | **57** | **61** | **60** |

**Table 2** Comparing ABLA with existing value heuristics. The table compares the performance of different value heuristics, so the best one in each row is highlighted.

| Time | Metrics | Heuristics | BIVS | | | RF | | | MV | RLARF |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | ABLA | ABLA2 | BIVS | ABLA | ABLA2 | RF | | |
| 1800s | Primal Gap | Pick | 20.89 | **20.52** | 24.02 | 22.22 | 22.17 | 25.47 | 30.38 | 29.91 |
| | | FRBA4 | 21.51 | **20.75** | 24.57 | 23.38 | 23.45 | 26.97 | 30.27 | 28.63 |
| | #Solved | Pick | 156 | **157** | 155 | **157** | **157** | **157** | 156 | 156 |
| | | FRBA4 | 155 | **156** | 155 | 154 | 154 | 154 | 154 | 155 |
| | #Optimum | Pick | 51 | 54 | 47 | 52 | 51 | 52 | **55** | 53 |
| | | FRBA4 | 54 | **58** | 50 | 57 | 57 | 56 | 57 | 56 |
| 3600s | Primal Gap | Pick | 18.33 | **16.91** | 18.29 | 18.45 | 19.48 | 23.49 | 27.35 | 26.27 |
| | | FRBA4 | **17.55** | 17.79 | 20.42 | 20.92 | 21.00 | 25.19 | 27.10 | 26.17 |
| | #Solved | Pick | 157 | **159** | **159** | **159** | 158 | 158 | 157 | 158 |
| | | FRBA4 | **158** | **158** | 156 | 155 | 155 | 155 | 156 | 156 |
| | #Optimum | Pick | 56 | 56 | 53 | 55 | **58** | 56 | **58** | 55 |
| | | FRBA4 | 55 | 61 | 57 | 63 | **66** | 61 | 60 | 60 |

Next, we compare the ABLA strategies with the existing value heuristics. The results are presented in Table 2. We can see that within the relatively short time limit of 1800 seconds, ABLA and ABLA2 exhibit similar performance in terms of primal gap and the ability to find feasible solutions, with ABLA2 showing a slight advantage. Moreover, ABLA2 demonstrates a stronger capability in proving optimality. Under the time limit of 3600 seconds, we observe mixed results in primal gap and the ability to find feasible solutions. The smallest primal gap is achieved by ABLA2+BIVS under the Pick variable ordering heuristic,

and ABLA2 continues to outperform ABLA in proving optimality. Overall, ABLA2 exhibits better comprehensive performance than ABLA. When compared with existing methods, ABLA and ABLA2 improve the primal gap and feasible solution performance of BIVS and RF in almost all cases, with the only exception being ABLA under the Pick variable ordering heuristic, which shows a slight degradation. In terms of proving optimality, ABLA and ABLA2 also enhance the performance of BIVS and RF in the majority of cases. Therefore, ABLA and ABLA2 generally improve the overall solving performance of BIVS and RF, with ABLA2 slightly outperforming ABLA. Among all value heuristics compared in the table, ABLA2+BIVS achieves the best performance in primal gap and feasible solution metrics. Although it is not the best in proving optimality, its performance remains competitive.

We then present statistics in Table 3 to demonstrate the effectiveness of ABLA and ABLA2 in enhancing look-ahead-based value heuristics. In the default setting of the Choco solver, the look-ahead process is always skipped for enumeration variables with a domain size greater than 100, and for interval variables, it only checks the two bounds. In such cases, ABLA is not applicable. Hence, in the first part of the table, we first examine how many value selection decisions are applicable to ABLA, which is the proportion of full look-ahead processes out of the total number of decisions. We can see that on average, full look-ahead processes are performed in over 90% of the value selections. In the second part of the table, we show the proportion of decisions requiring a full look-ahead for which ABLA decides to skip the intermediate checks. We can see that on average, ABLA and ABLA2 skip 11%-12% full look-ahead checks of BIVS, and they skip 42%-43% full look-ahead checks of RF. This is reasonable because RF only propagates along the constraints on the shortest paths from the current decision variable to the objective variable, which is weaker than the full propagation of BIVS. Consequently, it ignores the impacts of the current decision variable on the objective variable through other paths. In the third section of the table, we present the proportion of cases where the value selected when skipping the look-ahead process is the same as that selected when performing the full look-ahead. Specifically, this is the proportion of skip decisions made by ABLA for which forcing a complete look-ahead would not change the value selection, reflecting the proportion of appropriate skipping decisions. As we can observe, when ABLA and ABLA2 determine to skip the look-ahead process, the decision is highly reliable. This high accuracy suggests that when the two bounds yield the same impact on the objective, a full look-ahead rarely yields additional discriminative information. It explains why ABLA and ABLA2 improve the performance of BIVS and RF in terms of primal gap and the ability to find feasible solutions: by skipping unnecessary look-ahead processes, they select values identical to those that would have been chosen by the full look-ahead in most cases, but at lower costs. In the last section of the table, we present the proportion of cases where all values in the variable's domain have the same impact on the objective variable when ABLA decides to skip the look-ahead process. We can see that, on average, in more than 85% of the cases where the lower and upper bounds have identical impacts on the objective, the intermediate values also have the same impact.

Moreover, distinguishing whether the superiority of ABLA comes from its different decision-making or from computational savings is nontrivial, as any decision in a complete search may completely change the search trajectory, making it difficult to draw reliable conclusions in such cases. Consequently, we select representative instances on which ABLA2+RF and RF produce identical decisions (noting that ABLA2+BIVS and BIVS rarely produce identical decisions), thus constructing the same search tree. In such cases, we can fairly compare their runtime performance. The results are presented in Table 4. We can see that when they produce identical decisions, ABLA2 significantly speeds up the search process of the RF heuristic. It is interesting that all the look-ahead processes in the cvrp instances should be skipped.

**Table 3** Accuracy of ABLA's decisions to skip look-ahead processes. We present the results obtained under the pick/dom variable ordering heuristic. The table is divided into four sections from top to bottom. The first section shows the proportion of value selection decisions made by the full look-ahead strategies (BIVS and RF) out of all value selection decisions. The second section shows, among all decisions made by BIVS and RF, the proportion of the look-ahead process that ABLA decides to skip. The third section shows, for the look-ahead process that ABLA decides to skip, the proportion of cases where the value selection decision made is the same as that of performing the look-ahead. The fourth section shows, for the look-ahead process that ABLA decides to skip, the proportion of cases where all values in the variable's domain have the same impact on the objective variable. The $n_1$ column is the number of decisions requiring a full look-ahead by BIVS or RF. The results of these instances are grouped according to the interval into which the number of $n_1$ required to solve them falls. The number of instances in each interval is in the brackets. In each cell, we present the average proportion for the instances whose $n_1$ falls within the corresponding interval. There are some instances where ABLA and RF never determine to skip the look-ahead process, which are excluded from the table, so the sums of instance numbers in the average rows are less than the total instance number of 173.

| | $n_1$ | BIVS | | RF | |
|---|---|---|---|---|---|
| | | ABLA | ABLA2 | ABLA | ABLA2 |
| Proportion of Look-Ahead Decisions | 1 - $10^4$ | 98% (28) | 99% (26) | 99% (32) | 99% (32) |
| | $10^4$ - $10^5$ | 99% (18) | 95% (22) | 97% (19) | 96% (17) |
| | $10^5$ - $10^6$ | 86% (37) | 90% (36) | 85% (34) | 85% (35) |
| | $10^6$ - $10^7$ | 91% (59) | 90% (61) | 92% (50) | 91% (53) |
| | $10^7$ - $\infty$ | 99% (29) | 99% (26) | 99% (26) | 99% (24) |
| | Average | 93% (171) | 94% (171) | 94% (161) | 93% (161) |
| Proportion of Skipping Decisions | 1 - $10^4$ | 6% (28) | 6% (26) | 28% (32) | 30% (32) |
| | $10^4$ - $10^5$ | 10% (18) | 8% (22) | 40% (19) | 35% (17) |
| | $10^5$ - $10^6$ | 17% (37) | 13% (36) | 36% (34) | 40% (35) |
| | $10^6$ - $10^7$ | 13% (59) | 13% (61) | 48% (50) | 49% (53) |
| | $10^7$ - $\infty$ | 14% (29) | 13% (26) | 58% (26) | 54% (24) |
| | Average | 12% (171) | 11% (171) | 42% (161) | 43% (161) |
| Proportion of Decision Agreement When Skipping | 1 - $10^4$ | 89% (28) | 91% (26) | 92% (32) | 95% (32) |
| | $10^4$ - $10^5$ | 97% (18) | 97% (22) | 99% (19) | 93% (17) |
| | $10^5$ - $10^6$ | 98% (37) | 98% (36) | 92% (34) | 92% (35) |
| | $10^6$ - $10^7$ | 98% (59) | 98% (61) | 95% (50) | 94% (53) |
| | $10^7$ - $\infty$ | 97% (29) | 95% (26) | 98% (26) | 98% (24) |
| | Average | 97% (171) | 97% (171) | 95% (161) | 94% (161) |
| Proportion of Uniform Impact When Skipping | 1 - $10^4$ | 85% (28) | 86% (26) | 87% (32) | 86% (32) |
| | $10^4$ - $10^5$ | 88% (18) | 88% (22) | 85% (19) | 84% (17) |
| | $10^5$ - $10^6$ | 81% (37) | 83% (36) | 82% (34) | 83% (35) |
| | $10^6$ - $10^7$ | 88% (59) | 88% (61) | 91% (50) | 90% (53) |
| | $10^7$ - $\infty$ | 81% (29) | 80% (26) | 92% (26) | 93% (24) |
| | Average | 85% (171) | 85% (171) | 88% (161) | 88% (161) |

**Table 4** Runtime (in seconds) to find the best solution on some representative instances where ABLA2+RF and RF produce identical decisions. The last two columns present the number of value selection decisions made by RF with full look-aheads (#fulls) and the number of full look-aheads skipped by ABLA2 (#skips). Note that in these instances, the value selected when skipping the look-ahead process is the same as that selected when performing the full look-ahead.

| Instance | ABLA2+RF | RF | #fulls | #skip |
|---|---|---|---|---|
| tiny-cvrp_hard_instance_18 | **1645** | 2299 | 6,726,160 | 6,726,160 |
| tiny-cvrp_medium_instance_13 | **2437** | 2984 | 11,242,937 | 11,242,937 |
| EchoSched_12-12-0-1_7 | **1653** | 1953 | 4,255,294 | 3,823,562 |
| EchoSched_15-15-0-1_3 | **1956** | 2243 | 2,812,403 | 2,450,684 |

Additionally, the restart technique and *Nogood-Recording* technique enhance the efficiency and robustness of constraint solvers, but they may also mask some poor decisions made by the value heuristics. Therefore, we conduct an ablation study that re-evaluates the performance of ABLA and existing value heuristics without these two techniques. The results are presented in Table 5. We can see that when the two techniques are disabled, under both variable ordering heuristics, ABLA and ABLA2 improve the performance of BIVS and RF in terms of the primal gap in most cases, except that ABLA2 does not improve BIVS under the FRBA4 heuristic. In terms of proving optimality, both ABLA and ABLA2 offer benefits to BIVS and RF. Overall, even when the two techniques are removed, ABLA and ABLA2 still bring noticeable improvements.

**Table 5** Comparing ABLA against existing value heuristics without restart and nogood-recording in 3600 seconds.

| Metrics | Heuristics | BIVS | | | RF | | | MV | RLARF |
|---|---|---|---|---|---|---|---|---|---|
| | | ABLA | ABLA2 | BIVS | ABLA | ABLA2 | RF | | |
| Primal Gap | Pick | **33.41** | 33.92 | 35.66 | 35.17 | 34.69 | 35.24 | 36.52 | 38.32 |
| | FRBA4 | 25.33 | 29.86 | 27.73 | **18.86** | 20.29 | 22.03 | 28.38 | 28.20 |
| #Solved | Pick | 150 | 150 | 150 | 146 | 147 | 149 | **151** | 150 |
| | FRBA4 | 147 | 143 | 147 | **154** | 152 | 153 | 150 | 150 |
| #Optimum | Pick | 48 | 48 | 47 | 47 | **49** | 47 | 47 | 48 |
| | FRBA4 | 57 | 59 | 54 | **62** | 61 | 59 | 58 | 56 |

After that, we present the result of each problem in Table 6. We can see that no single value heuristic dominates the others. Performance varies across different problems. Among all the compared heuristics, ABLA2+BIVS achieves the best performance on 19 problems, which is the highest count. A direct comparison between ABLA2+BIVS and BIVS shows that ABLA2+BIVS outperforms BIVS on 16 problems, while BIVS performs better on 8 problems. This suggests that integrating ABLA2 makes BIVS more robust across different problem types.

Finally, *Solution-Based Phase Saving* (SBPS) [7] is a simple yet efficient strategy for value selection, which can be combined with the compared value heuristics. So we present the results involving SBPS in Table 7. Specifically, each of the compared value heuristics is used as the underlying selector for SBPS. With the integration of SBPS, ABLA2 still outperforms ABLA overall. In terms of primal gap, the combination of the Pick variable ordering heuristic and the RLARF value heuristic achieves the best performance within 1800 seconds, while ABLA2+BIVS performs best within 3600 seconds. Overall, ABLA2 consistently improves the primal gap performance of BIVS and RF across different variable heuristic configurations,

**Table 6** Detailed comparison of ABLA2+BIVS and existing value heuristics across different problems. We present the primal gap in the table, with the number of solved instances in brackets. The results are conducted with Pick variable ordering heuristic. The #Best row shows the number of problems where each value heuristic gets the best performance.

| Problem(#Instances) | ABLA2+BIVS | MV | BIVS | RF | RLARF |
|---|---|---|---|---|---|
| accap(5) | **0.56(5)** | 0.91(5) | 1.1(5) | 0.6(5) | 0.59(5) |
| aircraft-disassembly(5) | 0.06(3) | **0.0(3)** | 0.05(3) | **0.0(3)** | 0.08(3) |
| cable-tree-wiring(5) | **1.39(5)** | 1.42(5) | 2.01(4) | 1.72(5) | 1.57(5) |
| community-detection(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | 0.03(5) |
| compression(5) | **0.0(5)** | 1.0(4) | 1.0(4) | 1.0(4) | 1.0(4) |
| concert-hall-cap(5) | **0.01(5)** | 0.83(5) | 0.11(5) | 0.78(5) | 0.78(5) |
| fox-geese-corn(5) | 1.06(5) | 0.89(5) | **0.43(5)** | 0.98(5) | 0.58(5) |
| graph-clear(5) | 1.7(5) | 1.53(5) | 2.1(5) | 1.49(5) | **1.48(5)** |
| hoist-benchmark(5) | 0.37(5) | **0.17(5)** | 0.77(5) | **0.17(5)** | 0.2(5) |
| monitor-placement-1id(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** |
| neighbours(5) | **0.03(5)** | 1.65(5) | 0.04(5) | 1.07(5) | 1.65(5) |
| network(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** |
| peacable(5) | 2.16(5) | **2.01(5)** | 2.16(5) | 2.02(5) | 2.09(5) |
| portal(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** |
| tiny-cvrp(5) | 0.1(5) | 0.12(5) | **0.09(5)** | 0.1(5) | 0.13(5) |
| triangular(5) | 0.39(5) | 0.88(5) | 0.53(5) | **0.18(5)** | 0.98(5) |
| word-equations(4) | **0.17(3)** | 0.35(3) | 0.35(3) | 0.35(3) | 0.35(3) |
| yumi-dynamic(5) | 1.34(4) | 2.1(3) | 0.36(5) | 1.21(4) | **0.2(5)** |
| EchoSched(5) | 0.05(5) | **0.02(5)** | 0.06(5) | 0.03(5) | 0.03(5) |
| carpet-cutting(5) | 0.04(5) | 0.02(5) | 0.1(5) | **0.01(5)** | 0.02(5) |
| cgt(5) | **0.85(5)** | 4.3(5) | 1.53(5) | 3.2(5) | 4.2(5) |
| fbd1(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** |
| groupsplitter(5) | **0.18(5)** | 0.54(5) | 0.36(5) | 0.25(5) | 0.42(5) |
| hitori(5) | **0.0(1)** | **0.0(1)** | **0.0(1)** | **0.0(1)** | **0.0(1)** |
| ihtc-2024-kletzander(5) | **0.09(5)** | 2.11(5) | 0.15(5) | 2.19(5) | 2.22(5) |
| ihtc-2024-marte(4) | 0.09(4) | 0.36(4) | **0.05(4)** | 0.16(4) | 0.37(4) |
| is(5) | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** | **0.0(5)** |
| mondoku(5) | 1.83(4) | 1.53(4) | **1.33(4)** | 1.7(4) | 1.7(4) |
| products-and-shelves(5) | **0.15(4)** | **0.15(4)** | **0.15(4)** | **0.15(4)** | 1.17(3) |
| proteindesign12(5) | 0.02(5) | **0.01(5)** | 0.02(5) | **0.01(5)** | 0.02(5) |
| skill-allocation(5) | 2.0(3) | 2.0(3) | **1.0(4)** | 2.0(3) | 2.0(3) |
| stripboard(5) | **1.0(4)** | **1.0(4)** | **1.0(4)** | **1.0(4)** | **1.0(4)** |
| tower(5) | 0.89(5) | 1.24(5) | **0.81(5)** | **0.81(5)** | 1.14(5) |
| tsptw(5) | **0.01(4)** | **0.01(4)** | 0.02(4) | 0.02(4) | 0.02(4) |
| work-task-variation(5) | 0.34(5) | **0.19(5)** | 0.61(5) | 0.28(5) | 0.25(5) |
| All(173) | **16.91(159)** | 27.35(157) | 18.29(159) | 23.49(158) | 26.27(158) |
| #Best | **19** | 16 | 15 | 15 | 9 |

and ABLA does not bring improvement only at 3600 seconds under the Pick variable ordering heuristic. Notably, RLARF stands out in this setting, yet ABLA2+BIVS still outperforms RLARF in most cases on primal gap and feasible solution metrics, although it exhibits slightly weaker performance in proving optimality when combined with the Pick variable heuristic. Therefore, even equipped with SBPS, ABLA2 continues to enhance the performance of BIVS and RF.

**Table 7** Comparing ABLA with existing value heuristics under solution-based phase saving. The table compares the performance of different value heuristics, so the best one in each row is highlighted.

| Time | Metrics | Heuristics | BIVS | | | RF | | | MV | RLARF |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | ABLA | ABLA2 | BIVS | ABLA | ABLA2 | RF | | |
| 1800s | Primal Gap | Pick | 16.20 | 15.47 | 16.89 | 15.81 | 15.46 | 16.13 | 18.25 | **14.99** |
| | | FRBA4 | 16.90 | **15.66** | 18.55 | 17.18 | 17.22 | 17.86 | 16.40 | 16.45 |
| | #Solved | Pick | 156 | **157** | 155 | **157** | **157** | **157** | 156 | 156 |
| | | FRBA4 | 155 | **156** | 155 | 154 | 154 | 154 | 154 | 155 |
| | #Optimum | Pick | 56 | 56 | 57 | **58** | **58** | **58** | **58** | 57 |
| | | FRBA4 | 58 | **61** | 57 | 60 | 59 | 60 | **61** | **61** |
| 3600s | Primal Gap | Pick | 13.13 | **11.83** | 11.97 | 11.85 | 12.65 | 13.50 | 15.17 | 12.14 |
| | | FRBA4 | 13.92 | **12.71** | 15.43 | 15.34 | 14.81 | 15.66 | 13.88 | 13.71 |
| | #Solved | Pick | 157 | **159** | **159** | **159** | 158 | 158 | 157 | 158 |
| | | FRBA4 | **158** | **158** | 156 | 155 | 155 | 155 | 156 | 156 |
| | #Optimum | Pick | 59 | 57 | 57 | 60 | 60 | 59 | 59 | **61** |
| | | FRBA4 | 61 | 63 | 60 | **64** | 62 | 62 | 62 | 63 |

In summary, none of the evaluated value heuristics dominates the others across all three metrics. The decisions made by ABLA and ABLA2 to skip redundant look-ahead processes are highly reliable, enabling ABLA2 to consistently improve the performance of BIVS and RF. Among all compared value heuristics, ABLA2+BIVS exhibits the best comprehensive performance: it outperforms the others in terms of primal gap and the ability to find feasible solutions, while also delivering competitive performance in proving optimality.

## 6 Conclusion

In this paper, we propose a lightweight strategy for look-ahead-based value heuristics. The core idea is to assess, before performing a full look-ahead, whether the process is likely to yield discriminative information for value selection. This is achieved by examining the impact of the lower and upper bounds of the selected variable on the objective variable. If the impacts of the two bounds are identical, the look-ahead process is skipped. Based on this idea, we introduce two variants of the Assess Before Look-Ahead approach: ABLA and ABLA2. Experiments conducted on benchmark instances from recent MiniZinc Challenges demonstrate that the decisions made by our lightweight strategy to skip redundant look-ahead processes are highly reliable, achieving an average accuracy of over 94%. These results confirm that examining the impact of the lower and upper bounds is an effective way to identify redundant look-ahead processes in BIVS and RF. Furthermore, ABLA2 enhances the robustness of the BIVS value heuristic. The combination ABLA2+BIVS outperforms existing value heuristics in terms of primal gap and the ability to find feasible solutions, while also improving BIVS's capability in proving optimality.

## References


1 Gilles Audemard, Christophe Lecoutre, and Charles Prud'Homme. Guiding backtrack search by tracking variables during constraint propagation. In *International Conference on Principles and Practice of Constraint Programming*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CP.2023.9.

2 Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013. doi:10.1016/J.ORL.2013.08.007.

3 C. Bessiere. Constraint propagation. In *Foundations of Artificial Intelligence*, volume 2, pages 29–83. Elsevier, 2006. doi:10.1016/S1574-6526(06)80007-6.

4 F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proc. ECAI'04*, pages 146–150, 2004.

5 Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3677–3687, May 2021.

6 Augustin Delecluse and Pierre Schaus. Black-box value heuristics for solving optimization problems with constraint programming (short paper). In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CP.2024.36.

7 Emir Demirovic, Geoffrey Chu, and Peter J. Stuckey. Solution-based phase saving and large neighbourhood search. In *Proc. CP'18*, pages 99–108. Springer, 2018.

8 J. G. Fages and C. Prud'Homme. Making the first solution good. In *Proc. ICTAI'17*. IEEE, 2017.

9 E. Freuder. In pursuit of the holy grail. *ACM Comput. Surv.*, 28(4es):63–es, December 1996. doi:10.1145/242224.242304.

10 D. Habet and C. Terrioux. Conflict history based search for constraint satisfaction problem. In *Proc. of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1117–1122. ACM, 2019.

11 J. Hwang and D. G. Mitchell. 2-way vs d-way branching for csp. In *Proc. of CP'05*, pages 343–357. Springer, 2005.

12 C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4):147–167, 2007. doi:10.3233/sat190009.

13 H. Li, M. Yin, and Z. Li. Failure Based Variable Ordering Heuristics for Solving CSPs. In *Proc. CP'21*, pages 9:1–9:10, 2021.

14 M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993. doi:10.1016/0020-0190(93)90029-9.

15 Tom Marty, Tristan François, Pierre Tessier, Louis Gautier, Louis-Martin Rousseau, and Quentin Cappart. Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:19, 2023. doi:10.4230/LIPIcs.CP.2023.25.

16 L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Proc. CPAIOR'12*, pages 228–243. Springer, 2012.

17 A. Palmieri and G. Perez. Objective as a feature for robust search strategies. In *Proc. CP'18*, pages 328–344. Springer, 2018.

18 G. Pesant, C. G. Quimper, and A. Zanarini. Counting-based search: Branching heuristics for constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 43:173–210, 2012. doi:10.1613/JAIR.3463.

19 Gilles Pesant. Counting-based search for constraint optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), March 2016.

20 C. Prud'homme, J-G. Fages, and X. Lorca. *Choco Documentation*. TASC – LS2N CNRS UMR 6241, COSLING S.A.S., 2017. URL: `http://www.choco-solver.org`.

21 P. Refalo. Impact-based search strategies for constraint programming. In *Proc. CP'04*, pages 557–571. Springer, 2004. `doi:10.1007/978-3-540-30201-8_41`.

22 H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proc. of ICTAI'19*, pages 71–77. IEEE, 2019.

23 Y. Zhang and H. Li. Right branches matter in failure-based variable ordering heuristics. In *Proc. of AAAI'26*, pages 14397–14404, 2026.