Right Branches Matter in Failure-based Variable Ordering Heuristics

Yang Zhang[1] **and Hongbo Li**[1,2*]

[1]College of Information Science and Technology, Northeast Normal University, Changchun, China.
[2]Key Laboratory for Applied Statistics of Ministry of Education, Northeast Normal University, Changchun, China.
lihb905@nenu.edu.cn

Abstract

Failure-based variable ordering heuristics (VOH) are efficient general-purpose search heuristics for solving constraint satisfaction problems (CSP). They learn from the failures detected during the search and select the variables that are most likely to fail. The current failure-based VOHs, i.e. the failure-rate-based (FRBA) and failure-length-based (FLBA), focus on only the failures detected in left branches. In this paper, we investigate how the failure information from right branches affects the performance of the failure-based VOHs. Four strategies utilizing the failure information of right branches are proposed to refine the failure-based VOHs. Our experiments performed with the benchmark instances used in the recent MiniZinc challenges show that utilizing the failures detected in right branches enhances the performance of the failure-based VOHs. The unified version combining all the proposed strategies generally gets the best performance. It demonstrates remarkable superiority over several general-purpose VOHs, including activity-based search, conflict-history search, refined weighted degree, pick/dom, and the existing FRBA, which are considered state-of-the-art. Our study demonstrates that right branches matter in failure-based VOHs.

Source code —
https://github.com/zy-nesime/right-branch-frba

Introduction

Constraint programming is one of the foundations of artificial intelligence. It provides a powerful paradigm for tackling hard combinatorial search problems occurring in various fields, such as Scheduling, Routing, Configuration, Networks, etc. The basic idea in constraint programming is that the user models the real-world problems as constraint satisfaction problems (CSP) or constraint optimization problems (COP), and a general-purpose constraint solver is used to solve them. Backtracking search is a complete method for solving CSPs and COPs. At each step of backtracking search, an unassigned variable is selected to assign a value. The ordering in which the variables are assigned is crucial to the efficiency of backtracking search. It is a computationally difficult task to find an optimal ordering that results in a search tree exploring the fewest number of nodes (Liberatore 2000), thus, the ordering is usually determined by variable ordering heuristics (VOH) in practice.

The fail-first principle "to succeed, try first where you are likely to fail" (Haralick and Elliott 1980), has been a central idea for developing efficient VOHs for solving CSPs. Following the principle, many efficient VOHs have been proposed in the past decades, such as minimum domain size (Haralick and Elliott 1980), weighted degree (Boussemart et al. 2004; Li et al. 2016; Wattez et al. 2019), impact-based search (Refalo 2004), conflict-history search (CHS) (Habet and Terrioux 2019), failure-based search (Li, Yin, and Li 2021), pick/dom (Audemard, Lecoutre, and Prud'Homme 2023), and so on. Some other VOHs also use this principle to some extent, such as activity-based search (ABS) (Michel and Van Hentenryck 2012) and correlation-based search (Wang, Xia, and Yap 2017). For optimization problems, the objective has been considered as a feature for developing VOH (Palmieri and Perez 2018). Besides, some meta-heuristics are proposed to generate promising search entrances to find high-quality solutions earlier for COPs (Li et al. 2020; Li and Lee 2023). Among these efficient VOHs, the failure-based VOHs offer relatively straightforward implementations of the fail-first principle. They learn from the failures detected after propagating positive decisions in left branches of binary branching strategy, and identify the variables that are most likely to fail. However, the failures detected in right branches may provide valuable information.

In this paper, we investigate how the failure information from right branches affects the performance of the failure-based VOHs. We propose four strategies to utilize the failure information from right branches into the failure-based VOHs. Firstly, we simply add the propagation information of right branches. Secondly, we consider the consecutive failures occurring during constraint propagation. Thirdly, we involve the maximum number of consecutive failures to engage the impact from search depth. Finally, we combine the three strategies to obtain the unified version. Our experiments were performed with the benchmark instances used in recent MiniZinc challenges. The results show that utilizing the failures detected in right branches enhances the performance of the failure-based VOHs. The unified version significantly improves the performance of the existing failure-based VOHs, and outperforms several existing general-

*Corresponding author

purpose VOHs, including activity-based search (Michel and Van Hentenryck 2012), conflict-history search (Habet and Terrioux 2019), the refined weighted degree combined with minimum domain size (Wattez et al. 2019), pick/dom (Audemard, Lecoutre, and Prud'Homme 2023), and the existing failure-based VOHs (Li, Yin, and Li 2021), which can be considered as the state-of-the-art. Our study demonstrates that right branches matter in failure-based VOHs.

The paper is structured as follows. Section 2 reviews the background. Section 3 reviews failure-based VOHs and some other efficient VOHs. The refined versions of failure-based VOHs utilizing information from right branch are proposed in Section 4. Section 5 presents the experimental results and analysis, and Section 6 is the conclusion.

Background

A constraint satisfaction problem (CSP) $\mathcal{P}$ is a triple $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is a set of *n* variables $\mathcal{X} = \{x_1, x_2 \ldots x_n\}$, $\mathcal{D}$ is a set of domains $\mathcal{D} = \{dom(x_1), dom(x_2) \ldots dom(x_n)\}$, where $dom(x_i)$ is a finite set of possible values for variable x_i, and $\mathcal{C}$ is a set of *e* constraints $\mathcal{C} = \{c_1, c_2 \ldots c_e\}$. Each constraint *c* consists of two parts, an ordered set of variables $scp(c) = \{x_{i1}, x_{i2} \ldots x_{ir}\}$ and a subset of the Cartesian product $dom(x_{i1}) \times dom(x_{i2}) \times \ldots \times dom(x_{ir})$ that specifies the allowed combinations of values for the variables $\{x_{i1}, x_{i2} \ldots x_{ir}\}$. A solution to a CSP is an assignment of a value to each variable such that all the constraints are satisfied. Solving a CSP $\mathcal{P}$ usually involves either finding one (or more) solution of $\mathcal{P}$ or proving that $\mathcal{P}$ does not have a solution. A CSP is satisfiable if it has at least one solution; otherwise unsatisfiable.

A CSP is NP-hard in its general form. Backtracking search is widely used in solving CSPs. It performs a depth-first traversal of a search tree. We use *PastVar* to denote the set of fixed variables that have been assigned and *FutVar* to denote the set of future variables that have not been assigned. At each search tree node, a future variable is selected, typically by a variable ordering heuristic, and a new node is generated after the assignment to this variable; then a propagation algorithm (Bessiere 2006) is applied to filter inconsistent values from the domains of variables. If the propagation leads to a failure, e.g. a domain wipeout, one or more assignments must be canceled and backtracking occurs.

There are two branching strategies in backtracking search, k-way branching and binary branching (Hwang and Mitchell 2005). Binary branching has been shown to be more powerful than k-way branching, so it is widely used in modern constraint solvers. Binary branching considers two types of decisions: positive decisions, e.g. the instantiation $x_i = v_i$ where x_i is an unassigned variable and v_i is a value in $dom(x_i)$ and negative decisions, e.g. the refutation $x_i \neq v_i$. The search trees built by binary branching are binary trees where a left branch corresponds to a positive decision and a right branch corresponds to a negative decision. For example, given a variable x_i with $dom(x_i)$={1,2,3,4,5}, an instantiation $x_i = 1$ reduces the domain to $\{1\}$ and the refutation $x_i \neq 1$ reduces the domain to $\{2, 3, 4, 5\}$. Both types of decisions reduce the domain of the variable, so constraint propagation is applied after both types of decisions.

Related Work

Following the fail-first principle, many efficient VOHs have been developed. We will first recall the failure-based VOHs (Li, Yin, and Li 2021), which are closely related to our method, and then review some other efficient ones.

Failure-based VOHs

In the context of backtracking search, the failure rate of a variable x FR(x) is $\frac{failNum(x)}{assignNum(x)}$ where the *failNum*(x) is the total number of failures caused by the propagations of positive decisions made on x and the *assignNum*(x) is the total number of positive decisions made on x since the beginning of search. The failure-rate-based (FRB) VOH selects the variable with the largest $\frac{FR(x)}{|dom(x)|}$.

The length of a failure caused by the propagation of a positive decision of x is denoted by $|failure(x)|$, which is the number of variables fixed before propagating the decision. The failure-length-based (FLB) VOH associates a score *AFL*(x) with each variable x, and selects the variable with the largest $\frac{AFL(x)}{|dom(x)|}$. The score *AFL*(x) is initialized to 0 and is updated as follows.

$$AFL(x) = AFL(x) + \frac{1}{|failure(x)|} \tag{1}$$

The failure-based VOHs employ the strategy of CHS to give the variables that caused more recent failures more priority. For each variable x, the factor of the decaying strategy is defined as,

$$A(x) = \frac{1}{\#TotalFailure - LastFailure(x) + 1} \tag{2}$$

where *#TotalFailure* is the total number of failures detected since the beginning of the search, and *LastFailure*(x) stores the *#TotalFailure* value of the last failure caused by x. Based on the $A(x)$ factor, the FRBA VOH selects the variable with largest *FRBA*(x) defined as,

$$FRBA(x) = \frac{FR(x) + A(x)}{|dom(x)|} \tag{3}$$

The FLBA VOH selects the variable with largest *FLBA*(x) defined as,

$$FLBA(x) = \frac{AFL(x) \times A(x)}{|dom(x)|} \tag{4}$$

Other Efficient VOHs

The simplest VOH, minimum domain size, selects the variable with the smallest domain size (Haralick and Elliott 1980). It has been combined with many other VOHs, for example, the *dom/deg* (Bessière and Règin 1996) and *dom/ddeg* (Smith and Grant 1998) combine minimum domain size with the largest variable degree.

The weighted degree (Boussemart et al. 2004) associates a weight with each constraint, which records the number of failures caused by the constraint. It identifies the variables involved in the difficult parts of problems. Combined with

minimum domain size, the *dom/wdeg* VOH has been one of the most efficient general-purpose VOHs and has been used as the default VOH by some solvers, such as Choco (Prud'homme, Fages, and Lorca 2017). Its variants use different strategies to update the weights of constraints, such as constraint tightness (Li et al. 2016) and the explanation information of a failure (Hebrard and Siala 2017). Its recent refinement, $wdeg^{ca.cd}$, combines current arity and current domains to update the constraint weights, and has been shown to outperform the classic weighted degree heuristic (Wattez et al. 2019).

The impact-based search estimates the search space reduction after the propagation of the assignment of each variable (Refalo 2004). It prefers the variable that may lead to the greatest search space reduction.

The activity-based search estimates how active a variable is, e.g., how often a variable is affected by the propagation of the assignments of other variables (Michel and Van Hentenryck 2012). It prefers the most active variables. Furthermore, considering the objective as a feature, the objective-based variable selector (Palmieri and Perez 2018) combines activity with objective modifications to guide the search for solving COPs. It prefers the variables whose propagations have larger impacts on the objective variable.

The conflict history search considers the history of constraint failures (Habet and Terrioux 2019). Each constraint is associated with a reward that is higher when the constraint is frequently involved in conflicts. The CHS VOH prefers the variables involved in those constraints that have caused recent failures.

The pick/dom VOH (Audemard, Lecoutre, and Prud'Homme 2023) tracks the variables that trigger filtering operations during propagation. It records a *pick*$[x]$ for each variable, which is the number of values of a variable filtered during the propagations leading to failures. The variables with larger *pick* values are preferred.

The last-conflict-based reasoning (Lecoutre et al. 2009) can be considered as a higher level of VOH and can be combined with any VOH, called the underlying VOH. Whenever an assignment of a variable x is canceled, such as when the propagation of the assignment leads to a failure, x is stored as a last conflict variable. The strategy always selects the last conflict variable until its assignment succeeds. It makes the next selection by the underlying VOH if there is no last conflict variable stored.

Involving Right Branch in Failure-based VOH

It has been shown that the failure-rate-based VOH performs better than the failure-length-based VOH (Li, Yin, and Li 2021), so we focus on involving right branch in the FRB VOH. To calculate the failure rate of a variable x, we need to collect the *failNum*(x) and *assignNum*(x). As mentioned in the background, the existing FRB VOH, marked by FRB_0, uses the number of failures led by the propagation of positive decisions of x as *failNum*(x), and uses the number of positive decisions made on x as *assignNum*(x).

In the following, we propose four refined versions of the FRB VOH, marked by FRB_1, FRB_2, FRB_3, and FRB_4. In the rest of this section, we introduce how these refined versions calculate *failNum*(x) and *assignNum*(x) when the right branch is involved, while also explaining the fundamental principles and intuitive basis of these strategies.

FRB_1

The FRB_0 VOH considers only the decisions made at left branches. The aim is to estimate how likely the propagation of a positive decision of a variable x leads to a failure. A positive decision $x = a$ triggers a propagation because the domain of x is reduced. A negative decision $x \neq a$ also reduces the domain of s. Thus, FRB_1 involves the decisions made at right branches. The *failNum*(x) of FRB_1 is the total number of failures caused by propagating the decisions made on x (including both positive decisions and negative decisions) and the *assignNum*(x) of FRB_1 is the total number of the propagation of the decisions made on x (including both positive decisions and negative decisions).

FRB_2

In binary branching, if a positive decision leads to a failure, the corresponding negative decision will be generated and propagated immediately. If the negative decision also leads to a failure, the search will go back to the nearest positive decision and cancel it, and then generate the corresponding negative decision and propagate it. If all the propagations fail under extreme cases, the search may backtrack to the root node. Thus, a positive decision may be followed by a series of consecutive failures. Note that the VOH cannot make any decision before the consecutive propagations conclude with success, so we could consider that the last positive decision leads to all the consecutive failures. In FRB_2, when a positive decision of x leads to a failure, the *failNum*(x) is incremented by the number of consecutive failures following the decision. The *assignNum*(x) of FRB_2 is the total number of positive decisions made on x.

FRB_3

While FRB_1 considers only the single failure led by each decision, FRB_3 further considers the consecutive failures. In addition, we further involve the information of the maximum number of failures to make the search backtrack to the root node, denoted by *maxF*, which is the number of positive decisions attached from the root node to the current failure node. When a failure (led by either a positive one or a negative one) is detected, the *failNum*(x) of FRB_3 is incremented by the number of consecutive failures following the decision divided by *maxF*. The *assignNum*(x) of FRB_3 is the total number of the propagation of the decisions made on x (including both positive and negative ones).

FRB_4

FRB_4 is the strategy that combines the strategies of FRB_2 and FRB_3. When a positive decision of x leads to a failure, the *failNum*(x) is incremented by the number of consecutive failures following the decision divided by *maxF*. The *assignNum*(x) of FRB_4 is the total number of positive decisions made on x.

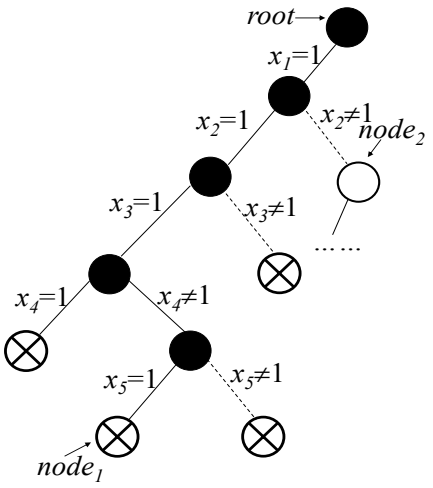

Figure 1: An example of consecutive failures.

An Example

Figure 1 shows an example of consecutive failures. The search propagates the decision of $x_5 = 1$ and backtracks to the branch of $x_2 \neq 1$ with the following process.

- The propagation of $x_5 = 1$ fails, and the corresponding negative decision $x_5 \neq 1$ is generated and propagated.
- The propagation of $x_5 \neq 1$ also fails, and the search cancels $x_4 \neq 1$ and $x_3 = 1$ directly.
- The negative decision $x_3 \neq 1$ is generated and propagated, and the propagation still fails, then the search cancels $x_2 = 1$ and propagates the negative decision $x_2 \neq 1$.
- The propagation of $x_2 \neq 1$ succeeds, and the $node_2$ is generated and the search continues.

The Table 1 presents how the failure-rate based VOHs increment the *failNum*(x) and *assignNum*(x) for these variables when the search backtracks from *node*$_1$ to *node*$_2$. Note that the table presents the number of counters incremented during the propagations from $node_1$ to $node_2$. The $failNum(x_5)$ of FRB$_3$ is incremented by 1, because the failures of $x_5 = 1$ and $x_5 \neq 1$ increment 3 and 2, respectively, and the sum is divided by 5, which is the maximum failure number from node1 to the root.

Strategies	Counters	x_1	x_2	x_3	x_4	x_5
FRB$_0$	$failNum$	0	0	0	0	1
	$assignNum$	0	0	0	0	1
FRB$_1$	$failNum$	0	0	1	0	2
	$assignNum$	0	1	1	0	2
FRB$_2$	$failNum$	0	0	0	0	3
	$assignNum$	0	0	0	0	1
FRB$_3$	$failNum$	0	0	0.2	0	1
	$assignNum$	0	1	1	0	2
FRB$_4$	$failNum$	0	0	0	0	0.6
	$assignNum$	0	0	0	0	1

Table 1: How different strategies increment the counters

Following the existing FRBA VOH, we can combine the FRB$_1$, FRB$_2$, FRB$_3$, FRB$_4$ with the decaying factor $A(x)$ to obtain FRBA$_1$, FRBA$_2$, FRBA$_3$ and FRBA$_4$.

Experiments

To examine how the information from right branches affects the performance of the failure-based VOHs, we perform extensive experimentation with the MiniZinc benchmark suite (Stuckey et al. 2014). The following are some general settings in the experiments:

- **Baselines.** We compare the refined FRBA VOHs with various efficient general-purpose VOHs, including activity-based search (Michel and Van Hentenryck 2012), conflict-history search (Habet and Terrioux 2019), dom/wdeg$^{ca.cd}$ (Wattez et al. 2019), pick/dom[1] (Audemard, Lecoutre, and Prud'Homme 2023), and the FRBA$_0$ (Li, Yin, and Li 2021),
- **Environment.** The environment is JDK23 under Ubuntu 22.04 with four Intel(R) Xeon(R) CPU E7-8890-v4 @2.20GHz and 512GB RAM. We have 96 cores in total, so 96 test runs are running simultaneously. Each run is allocated 1 core and 8GB RAM.
- **Solver.** The experiments were conducted in an efficient and well-architected CP solver, Choco (Prud'homme, Fages, and Lorca 2017), where all the baseline VOHs are already implemented.
- **Benchmark.** We have used the MiniZinc benchmark suite (Stuckey et al. 2014) in the experiments. All the instances were flattened offline.
  - The CSP instance set contains the 103 CSP instances (from 14 different problems) used in the MiniZinc Challenges of the past ten years. There are 105 instances used in the ten years (2014-2023), but two of them are duplicate ones used in two years. The challenge 2024 contains no CSP instance.
  - The COP instance set contains 186 COP instances used in MiniZinc challenges 2023 and 2024. There are 200 instances used in the two years (2023-2024), and we have eliminated 5 CSP instances, 3 instances with very large domains that can not be processed by Choco and 6 instances where we failed to flatten.
- **Restart.** The restart condition is the number of failures. A Luby restart strategy (Luby, Sinclair, and Zuckerman 1993) with an initial cutoff of 500 and the default restart strategy of Choco are used. The default restart strategy is a geometric restart strategy (Walsh 1999) with an initial cutoff of 5 and a growing factor of 1.05, and a total restart number of 50,000
- **Nogood-Recording.** All the compared search strategies are equipped with the default *Nogood-Recording* technique built in Choco, i.e. the *Reduced nld-Nogoods* (Lecoutre et al. 2007).

[1]There are two implementations of pick/dom in Choco. We tested both of them and present the result of the better one, e.g., the PickOnFil version.

- **Random seed.** Each of the compared VOH was tested with five random seeds from 0 to 4 and the numbers in Tables 2-8 are the average across 5 runs with different random seeds.
- In the following tables, the best one in each comparison is highlighted in bold.

Results on CSPs

- **Metrics.** The performance of searching for the first solution or proving unsatisfiable are measured by the number of instances solved (#Solved in the tables) in a timeout limit of 1200 seconds and the PAR2 Score [2] of runtime. The #Solved in the following tables is the average number of solved instances across 5 runs with different random seeds.
- **Value Selector.** To ensure a fair comparison, we uniformly employed the minimum value as the value selector across the experiments, which always selects the smallest value in the domain of a variable.

Firstly, we examine how the failures from right branches affect the decaying strategy factor. In the existing FRBA VOH, marked by FRBA_0, the counter *#TotalFailure* counts all the failures including those from both left branches and right branches, whereas the *LastFailure*(x) records the *#TotalFailure* value of the last failure caused by a positive decision of x. Thus, we can refine the decaying strategy by recording in *LastFailure*(x) the *#TotalFailure* value of the last failure caused by the propagation of a decision of x, either a positive one or a negative one. The refined decaying factor is marked by A'. We compared the performance of FRB_0, FRBA_0 and FRB_0+A' in Table 2. We can see that FRBA_0 performs better than FRB_0 in solving the satisfiable instances and the refined decaying strategy improves the existing decaying strategy. In unsatisfiable instances, the refined decaying strategy also improves the existing strategy, and no decaying strategy performs slightly better than the refined decaying strategy. The decaying strategies record the latest failure led by each variable. The refined version may have little impact on x_5 in the example, since the two failures are very close. However, it has a relatively large impact on x_3 whose latest failure is detected during consecutive backtracks. The results show that the refined decaying strategy gets the best performance in general. Thus, right branches matter in identifying the latest failure of each variable.

Secondly, we tested a refined initialization strategy. While the current strategy initializes the *failNum* of each variable to 0.5, the refined strategy simply initializes each *failNum* to 1, and the *assignNum* is still initialized to 1. In this way, the original failure rate of each variable is 1, which is the maximum value of failure rate. As the searching progresses, the failure rate of each variable decreases, so the variables that have never been assigned have the largest failure rate, and the strategy ensures that each variable will be tried. The results are presented in Table 3. We can see that the refined strategy loses a little when using the refined decaying strategy to solve unsatisfiable instances. In all the other cases, it improves the current initialization strategy.

[2]The PAR2 score of a solver is defined as the sum of all runtimes for solved instances + 2×timeout for unsolved instances, which is used in SAT competition.

	Metrics	FRB_0	FRBA_0	FRB_0+A'
All	#Solved	61.8	62.6	**64.6**
	PAR2	1005	1004	**957**
Sat	#Solved	56.8	58.2	**59.6**
	PAR2	787	767	**726**
Unsat	#Solved	**5.0**	4.4	**5.0**
	PAR2	**502**	744	520

Table 2: Comparison of decaying factors

		FRBA_0		FRB_0+A'	
	Metrics	0.5	1.0	0.5	1.0
All	#Solved	62.6	**65.4**	64.6	**65.6**
	PAR2	1004	**936**	957	**946**
Sat	#Solved	58.2	**60.6**	59.6	**60.6**
	PAR2	767	**689**	726	**706**
UnSat	#Solved	4.4	**4.8**	**5.0**	**5.0**
	PAR2	744	**645**	**520**	584

Table 3: Comparison of different initializations

In the following, all the refined versions of failure-rate-based VOHs (except for FRBA_0) use the refined decaying strategy and the refined initialization strategy. In Table 4, we compared the performance of the refined FRB VOHs. We can see that FRB_1, FRB_2, FRB_3 and FRB_4 enhance FRB_0, and FRBA_1, FRBA_2, FRBA_3 and FRBA_4 enhance FRBA_0. Thus, there are different strategies to enhance failure-based VOHs by utilizing the failure information from right branch. The failure information from right branch matters in failure-based VOHs. The FRBA_4 and FRB_4 achieved the best performance in their respective categories. FRBA_4 substantially improves the performance of the FRBA_0 VOH.

Next, we compare FRBA_4 with other efficient VOHs, which are considered the state-of-the-art. Besides the Luby restart strategy, the default restart strategy of Choco is involved here. The results are presented in Table 5. We can see that Luby restart is more efficient than the default restart strategy in solving these problems. The activity-based search, a classic search heuristic proposed in 2012 (Michel and Van Hentenryck 2012), is still efficient. The latest one, pick/dom (Audemard, Lecoutre, and Prud'Homme 2023), is the best one among the existing VOH. Under both of the two restart strategies, FRBA_4 demonstrates remarkable superiority over the state-of-the-art VOHs.

Finally, we investigated how these VOHs perform in each of the problems. The results are presented in Table 6. We can see that when equipped with the default restart strategy, FRBA_4 gets the best performance in 9 out of 14 problems, and it is the only best one in solving 5 of the problems. When equipped with the Luby restart strategy, FRBA_4 gets the best performance in 11 out of 14 problems, and it is the only best one in solving 6 of the problems. Note that FRBA_4 never gets the worst performance. Thus, FRBA_4 is more robust

	Metrics	$FRBA_0$	$FRBA_1$	$FRBA_2$	$FRBA_3$	$FRBA_4$	FRB_0	FRB_1	FRB_2	FRB_3	FRB_4
All	#Solved	62.6	71.6	69.4	72.0	**73.2**	61.8	69.0	69.0	65.6	69.4
	PAR2	1004	782	834	781	**754**	1005	831	852	934	844
Sat	#Solved	58.2	66.6	64.4	67.4	**68.2**	56.8	65.0	64.2	61.6	65.2
	PAR2	767	505	572	492	**474**	787	543	586	665	563
Unsat	#Solved	4.4	**5.0**	**5.0**	4.6	**5.0**	**5.0**	4.0	4.8	4.0	4.2
	PAR2	744	528	501	675	**463**	502	846	606	937	792

Table 4: Comparison among the failure-based VOHs

	Metrics	Default Restart						Luby Restart					
		$FRBA_0$	WDEG	CHS	ABS	Pick	$FRBA_4$	$FRBA_0$	WDEG	CHS	ABS	Pick	$FRBA_4$
All	#Solved	62.6	57.6	48.2	59.8	61.0	**70.2**	62.6	58.6	48.6	63.0	64.2	**73.2**
	PAR2	1007	1103	1341	1057	1042	**816**	1004	1074	1325	1007	977	**754**
Sat	#Solved	57.6	52.6	46.2	55.4	56.0	**65.2**	58.2	53.8	46.8	58.0	59.2	**68.2**
	PAR2	789	907	1120	834	827	**551**	767	865	1095	784	744	**474**
UnSat	#Solved	**5.0**	**5.0**	2.0	4.4	**5.0**	**5.0**	4.4	4.8	1.8	**5.0**	**5.0**	**5.0**
	PAR2	506	543	1715	743	588	**473**	744	623	1790	570	603	**463**

The WDEG column is for $\frac{dom}{wdeg^{ca.cd}}$. The pick column is for pick/dom.

Table 5: Comparing $FRBA_4$ against the state-of-the-art VOHs.

Problems	Default Restart						Luby Restart					
	$FRBA_0$	WDEG	CHS	ABS	Pick	$FRBA_4$	$FRBA_0$	WDEG	CHS	ABS	Pick	$FRBA_4$
sudoku_fixed(5)	2.8	3.0	3.0	3.0	3.0	**3.2**	3.0	3.0	2.8	2.8	3.0	**3.2**
rotating-workforce(15)	**4.0**	3.4	1.6	3.6	3.6	3.8	4.2	4.0	2.2	3.6	4.8	**6.0**
perfect_square(5)	0.2	0.0	0.0	0.2	0.0	**1.2**	0.0	0.0	0.0	0.2	0.0	**1.0**
pentominoes(10)	8.0	**8.6**	7.0	8.4	7.4	7.6	8.0	8.2	7.0	**8.4**	8.0	7.2
whirlpool(5)	0.0	0.0	0.4	0.0	0.0	**0.6**	0.2	0.0	**1.2**	0.0	0.0	**1.2**
ecp_xIGData(9)	4.4	0.4	0.0	2.6	3.2	**7.8**	5.0	1.0	0.0	4.0	5.2	**8.6**
amaze3(10)	8.8	8.8	7.4	6.8	8.8	**9.2**	8.6	8.6	6.8	8.2	**9.0**	**9.0**
oocsp_racks(10)	9.6	9.6	9.0	9.4	**10.0**	**10.0**	9.4	9.6	8.4	**10.0**	**10.0**	**10.0**
sb(9)	5.8	7.4	5.8	7.2	**8.2**	8.0	6.0	7.8	6.2	6.8	7.4	**8.2**
nmseq(5)	**5.0**	**5.0**	3.6	**5.0**	4.6	**5.0**	**5.0**	**5.0**	4.0	**5.0**	4.4	**5.0**
CostasArray(5)	**3.0**	2.4	2.4	2.6	2.8	2.8	2.8	2.4	2.0	**3.0**	2.4	2.8
rect_packing(5)	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**	**5.0**
mknapsack(5)	4.0	4.0	3.0	**5.0**	3.0	4.0	3.6	4.0	3.0	**5.0**	3.0	4.0
fillomino(5)	**2.0**	0.0	0.0	1.0	1.4	**2.0**	1.8	0.0	0.0	1.0	**2.0**	**2.0**
#Wins	5	3	1	3	3	**9**	2	2	2	6	4	**11**

Each number in the brackets is the instance number of the problem which is the upper bound of the cells in the corresponding row.

Table 6: Numbers of instances of each problem solved by different VOHs.

than the existing VOHs when solving these CSP instances.

Results on COPs

- **Metrics.** We follow the metrics used in the pick/dom study (Audemard, Lecoutre, and Prud'Homme 2023). The performance of each VOH h is measured by the number of times h proves optimality and the number of times h gives the best bound in a timeout limit of 1200 seconds, compared to the other VOHs. In addition, a ranking score is also used. Specifically, for each instance:
  - if the instance is unsatisfiable, 1 point is won by h if h indicates that the instance is unsatisfiable, 0 otherwise,
  - if h finds a solution with a bound worse than another one (found by another VOH), 0 point is won by h,
  - if h finds an optimal solution, and the optimality is proved, 1 point is won by S,
  - if h find a solution with the best found bound among all VOHs, while indicating no information about optimality: 1 point is won by h if no other VOH proves that this bound is optimal, 0.5 otherwise.
- **Value Selector.** We uniformly employed the solution-based phase saving (Demirovic, Chu, and Stuckey 2018) as the value selector and the underlying value selector is the minimum value, e.g. when the value in the last solution is absent, the minimum value takes over.

Heuristic	Default Restart			Luby Restart		
	Score	Optimality	Best Bound	Score	Optimality	Best Bound
$FRBA_1$	111.9	80.4	113.8	112.2	81.2	113.4
$FRBA_2$	111.7	77.8	113.6	111.7	79.4	113.6
$FRBA_3$	108.1	81	109.8	108	78.2	110.6
$FRBA_4$	**113.5**	**81.6**	**115**	112	79	114.2
$FRBA_0$	107	76	110	107	80	108.6
WDEG	81	63	86	82.3	62.6	87.4
CHS	63.9	39.6	74.2	66.7	42	75.8
ABS	103	73.8	106.4	107.7	73.6	111.6
Pick	92.3	60.4	99.6	103.2	69	109.4

The table reports the ranking score (Score), the number of instances where optimality is proved (Optimality), and the number of instances where the best bound is found (Best Bound). These scores were calculated by considering all 18 configurations, which represent the complete set of combinations between the 9 VOHs and the 2 restart strategies.

Table 7: Comparing the failure-based VOHs against the other state-of-the-art VOHs on COP instances.

Problem	$FRBA_4$	$FRBA_0$	ABS	Pick	Problem	$FRBA_4$	$FRBA_0$	ABS	Pick
accap	0.0:0.0	0.0:0.0	0.0:0.0	**1.0**:**5.0**	hoist-benchmark	**1.0**:2.0	**1.0**:1.0	**1.0**:**4.2**	0.4:0.4
portal	5.0:5.0	5.0:5.0	5.0:5.0	5.0:5.0	word-equations	2.0:3.0	**3.0**:3.0	**3.0**:3.0	2.4:3.0
harmony	**1.0**:2.6	0.2:4.2	**1.0**:**4.4**	0.0:2.2	network-50-cstr	5.0:5.0	5.0:5.0	5.0:5.0	5.0:5.0
tiny-cvrp	**2.0**:4.4	1.8:4.2	**2.0**:3.8	1.0:**4.6**	peacable-queens	1.0:2.6	1.0:**4.2**	1.0:2.6	1.0:1.4
neighbours	1.0:**4.0**	1.0:2.6	1.0:3.0	1.0:1.6	concert-hall-cap	**1.0**:0.4	0.8:1.2	**1.0**:**2.4**	0.0:2.0
triangular	1.0:3.2	1.0:3.0	1.0:**3.8**	1.0:3.6	cable-tree-wiring	0.0:1.6	0.0:0.4	0.0:**3.2**	0.0:0.0
compression	**4.0**:4.4	3.2:5.0	3.8:4.0	2.0:2.6	community-detection	4.0:4.2	**5.0**:**5.0**	**5.0**:**5.0**	4.0:**5.0**
graph-clear	0.0:1.6	0.0:**3.6**	0.0:0.0	0.0:1.0	aircraft-disassembly	0.0:**4.8**	0.0:4.2	0.0:3.0	0.0:2.0
yumi-dynamic	**1.0**:**2.2**	0.0:1.8	0.8:1.8	0.6:2.0	monitor-placement-1id	5.0:5.0	5.0:5.0	5.0:5.0	5.0:5.0
fox-geese-corn	0.0:2.8	0.0:2.6	0.0:**3.6**	0.0:0.4	#wins	**6**:**7**	3:4	**6**:3	1:3

Each of the problems contains 5 instances. For each VOH, the couple 'number of proved optima : number of best bounds found' is provided. The #win presents the number of problems where each VOH gets the best performance. Note that these scores were calculated by considering the 4 VOHs combined with the geometric restart strategy.

Table 8: Detailed results of COP instances from MiniZinc Challenge 2024.

Table 7 presents the overall comparative results. The results are the average across 5 runs with different random seeds. We can see that for these COPs, ABS outperforms dom/wdeg$^{ca.cd}$, CHS and pick/dom on every metric, however, it is outperformed by the refined failure-based VOHs. $FRBA_4$ gets the best performance in all three metrics. We present the detailed results of some problems in Table 8. Due to space limitations, we present the results of the instances from MiniZinc Challenge 2024, and four VOHs, $FRBA_4$, $FRBA_0$, ABS and pick/dom are involved. We select the four VOHs because $FRBA_4$ is the representative of the refined FRBA, and $FRBA_0$, ABS and pick/dom perform better than the other two existing VOHs. We can see that $FRBA_4$ demonstrated leading performance across both metrics. It achieved the best performance on 6 problems (ranked joint first) in proving optimality, and is the sole best performer on 7 problems in finding best bounds.

In summary, the failure information from right branches enhances the performance of the failure-based VOHs. Besides counting the failures and decisions on right branches, the information is contributive in the following two aspects: (1) recording the latest failure of each variable to refine the decay component, and (2) incrementally updating failure counts based on consecutive failures.

Conclusion

In this paper, we propose four strategies to involve the failure information from right branches in failure-based variable ordering heuristics. The experiments performed with the CSP and COP instances used in recent MiniZinc challenges show that there are various strategies to enhance the performance of failure-based VOHs by utilizing the information from right branches. Combining the proposed strategies, the $FRBA_4$ VOH significantly improves the performance of the existing FRBA VOH, and it outperforms several efficient VOHs that are considered the state-of-the-art. The failure information from right branches refines the decaying component of the failure-based VOHs, and the consecutive failures should also be taken into account. Right branches matter in failure-based variable ordering heuristics.

Acknowledgments

We thank the anonymous referees for their constructive comments. This work is supported by the National Natural Science Foundation of China under Grant 62276060, CCF-Huawei Populus Grove Fund, and Science and Technology Development Program of Jilin Province under Grant 20230101060JC.

References

Audemard, G.; Lecoutre, C.; and Prud'Homme, C. 2023. 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.

Bessiere, C. 2006. Constraint propagation. In *Foundations of Artificial Intelligence*, volume 2, 29–83. Elsevier.

Bessière, C.; and Règin, J. C. 1996. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. CP'96*, 61–75. Springer.

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

Demirovic, E.; Chu, G.; and Stuckey, P. J. 2018. Solution-based phase saving and large neighbourhood search. In *Proc. CP'18*, 99–108. Springer.

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

Haralick, R.; and Elliott, G. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14: 263–313.

Hebrard, E.; and Siala, M. 2017. Explanation-Based Weighted Degree. In *Proc. CPAIOR'17*, 167–175. Springer.

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

Lecoutre, C.; Sais, L.; Tabary, S.; and Vidal, V. 2007. Recording and Minimizing Nogoods from Restarts. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4): 147–167.

Lecoutre, C.; Saïs, L.; Tabary, S.; and Vidal, V. 2009. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18): 1592–1614.

Li, H.; and Lee, J. H. 2023. Finding Good Partial Assignments during Restart-Based Branch and Bound Search. In *Proc. AAAI'23*, 4035–4043.

Li, H.; Lee, J. H.; Mi, H.; and Yin, M. 2020. Finding Good Subtrees for Constraint Optimization Problems Using Frequent Pattern Mining. In *Proc. AAAI'20*, 1577–1584.

Li, H.; Liang, Y.; Zhang, N.; Guo, J.; Xu, D.; and Li, Z. 2016. Improving Degree-based Variable Ordering Heuristics for Solving Constraint Satisfaction Problems. *Journal of Heuristics*, 22(2): 125–145.

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

Liberatore, P. 2000. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1): 315–326.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4): 173–180.

Michel, L.; and Van Hentenryck, P. 2012. Activity-based Search for Black-box Constraint Programming Solvers. In *Proc. CPAIOR'12*, 228–243. Springer.

Palmieri, A.; and Perez, G. 2018. Objective as a Feature for Robust Search Strategies. In *Proc. CP'18*, 328–344. Springer.

Prud'homme, C.; Fages, J.-G.; and Lorca, X. 2017. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S.

Refalo, P. 2004. Impact-based Search Strategies for Constraint Programming. In *Proc. CP'04*, 557–571. Springer.

Smith, B. M.; and Grant, S. A. 1998. Trying Harder to Fail First. In *Proc. ECAI'98*, 249–253.

Stuckey, P. J.; Feydy, T.; Schutt, A.; Tack, G.; and Fischer, J. 2014. The MiniZinc Challenge 2008–2013. *AI Magazine*, 35(2): 55–60.

Walsh, T. 1999. Search in a Small World. In *Proc. IJCAI'99*, 1172–1177.

Wang, R.; Xia, W.; and Yap, R. H. C. 2017. Correlation Heuristics for Constraint Programming. In *Proc. ICTAI'17*, 1037–1041. IEEE.

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