A Portfolio-Based Approach to Select Efficient Variable Ordering Heuristics for Constraint Satisfaction Problems

Hongbo Li
School of Information Science and Technology, Northeast Normal University, Changchun, China

Yaling Wu
School of Information Science and Technology, Northeast Normal University, Changchun, China

Minghao Yin
School of Information Science and Technology, Northeast Normal University, Changchun, China

Zhanshan Li[1]
College of Computer Science and Technology, Jilin University, Changchun, China

Abstract

Variable ordering heuristics (VOH) play a central role in solving Constraint Satisfaction Problems (CSP). The performance of different VOHs may vary greatly in solving the same CSP instance. In this paper, we propose an approach to select efficient VOHs for solving different CSP instances. The approach contains two phases. The first phase is a probing procedure that runs a simple portfolio strategy containing several different VOHs. The portfolio tries to use each of the candidate VOHs to guide backtracking search to solve the CSP instance within a limited number of failures. If the CSP is not solved by the portfolio, one of the candidates is selected for it by analysing the information attached in the search trees generated by the candidates. The second phase uses the selected VOH to guide backtracking search to solve the CSP. The experiments are run with the `MiniZinc` benchmark suite and four different VOHs which are considered as the state of the art are involved. The results show that the proposed approach finds the best VOH for more than 67% instances and it solves more instances than all the candidate VOHs and an adaptive VOH based on Multi-Armed Bandit. It could be an effective adaptive search strategy for black-box CSP solvers.

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

Keywords and phrases Constraint Satisfaction Problem, Variable Ordering Heuristic, Adaptive Search Heuristic, Portfolio

Digital Object Identifier 10.4230/LIPIcs.CP.2022.32

Supplementary Material *Software (Source Code)*: `https://github.com/lihb905/sevoh`
archived at `swh:1:dir:566338d2739577697c51179e2071e6bf0bea108f`

Funding *Hongbo Li*: The National Natural Science Foundation of China (61802056). The Plan of Science and Technology Development in Jilin Province (20210101470JC). Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University.
Minghao Yin: The National Natural Science Foundation of China under Grant NO. 61976050.
Zhanshan Li: Open Research Fund of Key Laboratory of Space Utilization, Chinese Academy of Sciences under Grant NO. LSU-KFJJ-2019-08.

1 Introduction

The challenge in a Constraint Satisfaction Problem (CSP) is to find an assignment of values to all variables that satisfies the constraints defined over the variables, or otherwise, to prove that there is no such an assignment. Backtracking search is a complete method for solving

[1] Corresponding author

CSPs. It performs a depth-first traversal of a search tree to solve CSPs. At each node of the search tree, 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 for solving CSPs. Thus, variable ordering heuristics (VOH) play a central role in solving CSPs.

In the past decades, much effort has been done in developing effective variable ordering heuristics [3, 21, 16, 23, 7, 25, 13]. There is no VOH dominating all the others in solving all CSP instances. In other words, different VOHs have different performances on different problems. The performances of different VOHs can vary greatly while solving the same CSP instance. Thus, if we can find the best VOHs for different CSP instances, then the overall performance of a black-box CSP solver will be significantly improved. In recent years, determining an efficient VOH for a given CSP instance has attracted much attention. For instance, a reinforcement learning technique, Multi-Armed Bandit (`MAB`), has been used to design adaptive VOHs for CSPs [27, 24]. In these approaches, `MAB` is employed to select VOHs to make decisions and the candidate VOHs are switched over during the search.

In this paper, we propose a new approach to select efficient VOHs for solving different CSP instances. The approach contains two phases. The first phase is a probing procedure that checks how each candidate VOH behaves when trying to solve the CSP instance within limited resources. It runs a portfolio strategy containing the candidate VOHs and monitors the searching process. In each run with a candidate, a search tree will be generated within a limited number of failures. If the failure number reaches the limit, it restarts the search to try the next candidate. For each candidate VOH, the maximum depth and the failure depth of the search trees are collected. If the CSP is not solved during the probing, some measurements utilizing the collected information are used to select an efficient VOH. In the second phase, the selected VOH is used to guide backtracking search until the search completes. Extensive experimentation with the `MiniZinc` benchmark suite are performed to examine the efficiency of the proposed approach. Four modern VOHs including activity-based search (`ABS`) [16], conflict-history search (`CHS`) [7], refined weighted degree (`WDEG`) [25] and failure rate based VOH (`FRBA`) [13] are used as candidates. The results show that the proposed approach finds the best VOH from four candidates for more than 67% instances and it solves more instances than all the candidates and an adaptive strategy based on `MAB`. The approach does not need an offline training and it is easy to be implemented in constraint solvers, so it could be an effective adaptive search strategy for black-box CSP solvers.

2 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 disallowed (or 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 $\mathcal{P}$ involves either finding one (or more) solution of $\mathcal{P}$ or proving that $\mathcal{P}$ is unsatisfiable.

To solve real world problems, the users model the problems as CSPs and constraint solvers solve the CSPs. Average users have little knowledge about constraint solving, so an efficient black-box solver is required. Backtracking search is a standard algorithm for solving CSPs. It performs a depth-first traversal of a search tree. At each search tree node, an unassigned variable is selected and a new node is generated after the assignment to this variable, then a propagation algorithm is applied to filter those inconsistent values from the domains of variables. If the propagation leads to a domain wipe out, then a failure is encountered, one or more assignments must be canceled and a backtracking occurs. The ordering in which the variables are assigned is crucial to the efficiency of backtracking search and it is difficult to find an optimal ordering that results in a search tree exploring the smallest number of nodes [14]. Thus, the ordering is usually determined by variable ordering heuristics (VOH).

There exists many efficient VOHs, such as the impact-based search [21], activity-based search [16], the count-based search [19], the correlation-based search [23], the conflict-history search [7], the refined weighted degree [3, 25] and the failure-based VOHs [13]. None of them dominates all the others and their performances may vary greatly in solving a same CSP instance. Thus, finding the right VOH for a given CSP is a key issue for black-box solvers.

3 Related Work

Adaptive constraint solving has been studied in the CP community. A major difference between these methods is whether an offline training phase is required. Relying on the offline training phases, these methods are effective to predict an efficient algorithm, heuristic or even a solver for CSP instances [28, 9, 5, 1]. While these methods have shown their effectiveness, they may be less efficient to solve a new unseen instance (such as a new real-world problem) if it contains some unknown characteristics or structures, e.g., it is not close to any of the instances used in the training data.

On the contrary, the online learning methods do not require an offline training phase or training data. Some of them do learning during constraint solving procedure, such as the the modern VOHs [21, 16, 3, 23, 17, 7, 13, 12], the bandit-based search strategies [15, 27, 24] and the adaptive constraint propagation techniques [18, 26, 2]. Some other methods do online learning before searching starts, such as the learning value heuristics with a linear regression method [4] and the frequent pattern mining-based search [11].

Among these online learning approaches, the closest works are the two adaptive variable ordering heuristics based on Multi-Armed Bandit. Both them associate each candidate VOH with an arm. The first one applies the Upper Confidence Bound algorithm (`UCB1`) and Thompson Sampling (`TS`) algorithm to select a candidate VOH to make a decision at each search tree node [27]. If a new node is generated by a candidate H_i, the reward of H_i will be updated with the number of children of the search tree node. The second one applies the exponentially weighted forecaster for exploration and exploitation (`EXP3`), `UCB1` and `TS` to estimate the best VOH for given CSPs [24]. It exploits a restart mechanism with the `MAB`-based framework. Besides, the candidate VOHs are switched only after restarts. Its reward function is defined by the pruned tree size (`PTS`) [24]. The two methods use the reinforcement learning technique to combine different VOH online, e.g., the candidate VOHs are switched over during search. Our approach uses an effective measurement to select the best VOHs for given CSPs. Although our approach also switches the candidate VOH at the probing phase, after the best one is selected, it will be used to guide backtracking search until the search completes.

4 Selecting An Efficient VOH for a CSP

Given a set of k candidate VOHs, H_1, H_2, ..., H_k. We propose an approach, namely Selecting Efficient Variable Ordering Heuristics (`SEVOH`), to select an efficient VOH for solving a given CSP instance. The approach consists of two phases. The first phase is a probing procedure that runs a simple portfolio strategy to collect some information of search trees built by the candidate VOHs. If the problem is not solved by the probing procedure, we analysis the collected information to select an efficient VOH for the problem. The second phase is a straightforward strategy that uses the selected VOH to guide backtracking search to solve the problem. In the following, we introduce how to design the measurements for selecting an efficient VOH and how to collect search tree information for the measurements in a portfolio-based probing procedure.

4.1 The Measurements for Selecting An Efficient VOH

To select an efficient VOH for a CSP instance, we analysis how the candidate VOHs behave when solving the instance within limited resources, e.g., a limited number of failures. Then some measurements should be used to evaluate the performance of the candidates.

Firstly, many effective VOHs are designed according to the Fail First Principle that "to succeed, try first where you are likely to fail" [8]. Therefore, one intuition is that a more efficient VOH may detect failures at higher levels of the search trees, which has been used by the failure length based VOH [13]. The intuition is reasonable, because the higher level a failure is detected, the more search space is pruned. So our first measurement is the minimum failure depth of the search trees of each candidate.

- **Minimum Failure Depth (`MinFD`)**
  Given a CSP instance and a VOH H_i, we use H_i to guide backtracking search to solve the instance and a search tree will be built. In the search tree, the number of assigned variables when a failure occurs is record as the depth of the failure. A search tree usually contains a number of failures, so we use the average of the depths of all the failures as the failure depth of the search tree. The probing procedure, introduced in next subsection, will generates *roundLimit* search trees for H_i. The minimum one of the failure depth of the *roundLimit* search trees is considered as the `Minimum Failure Depth` of H_i for the instance, denoted by `MinFD`(H_i). This measurement prefers the VOH with smaller `MinFD`.

Secondly, an efficient VOH for a satisfiable CSP instance should find a solution as early as possible. An intuition is that a more efficient VOH may explore a deeper search tree than the others within limited resources. So our second measurement is the maximum depth of the search trees.

- **Maximum Depth (`MaxD`)**
  Given a CSP instance, each candidate H_i will build *roundLimit* search trees during the probing procedure. Each of the search trees has a deepest depth which is the largest number of simultaneously assigned variables. We use the largest one of all the deepest depth of the *roundLimit* search trees as the `Maximum Depth` of H_i for the instance, denoted by `MaxD`(H_i). This measurement prefers the VOH with larger `MaxD`.

Given a CSP instance, for each candidate H_i, we collect the information during the probing procedure to calculate `MinFD`(H_i) and `MaxD`(H_i). Each of the two measurements can be used as the scoring function to evaluate the performance of H_i solving the instance. Besides, we can combine the two measurements to use $\frac{\texttt{MaxD}(H_i)}{log(\texttt{MinFD}(H_i))}$ as the scoring function to evaluate the performance of H_i, which prefers the candidate with the largest score. The logarithmic scaling is to make `MinFD`(H_i) a smaller number, because the `MinFD`(H_i) and `MaxD`(H_i) are quite close in some instances.

4.2 A Portfolio-Based Probing

The portfolio contains several candidate VOHs, H_1, H_2, ..., H_k. The probing procedure is shown in Algorithm 1.

Algorithm 1 Portfolio-Based Probing.

```
   Input: $k$ candidate VOHs: $H_1$, $H_2$, ..., $H_k$; the maximum number of rounds:
          $roundLimit$; the maximum number of failures in each call: $failLimit$
   Output: the collected MinFD and MaxD, or unsatisfiable, or a solution.
1  for $i$ = 1 to $k$ do
2      MaxD($H_i$) $\leftarrow$ 0;
3      MinFD($H_i$) $\leftarrow$ the number of variables;
4  $round \leftarrow 1$;
5  while $round \leq roundLimit$ do
6      for $i$ = 1 to $k$ do
7          $failNum \leftarrow 0$;
8          $totalFailDepth \leftarrow 0$;
9          while $failNum < failLimit$ do
10             $depth \leftarrow$ the number of fixed variables;
11             if $depth >$ MaxD($H_i$) then
12                 MaxD($H_i$) $\leftarrow depth$;
13             $x \leftarrow$ the variable selected by $H_i$;
14             $v \leftarrow$ a value selected for $x$;
15             if the propagation of the assignment $x$=$v$ fails then
16                 $totalFailDepth \leftarrow totalFailDepth + depth$;
17                 $failNum \leftarrow failNum + 1$;
18                 if unsatisfiable is proved then
19                     terminate the search and return unsatisfiable;
20                 backtracking occurs;
21             else
22                 if a solution is found then
23                     terminate the search and return the solution;
24         $failDepth \leftarrow totalFailDepth / failNum$;
25         if $failDepth <$ MinFD($H_i$) then
26             MinFD($H_i$) $\leftarrow failDepth$;
27         restart the search;
28     $round \leftarrow round + 1$;
```

The procedure runs at most *roundLimit* rounds (line 5). In each round, the candidate VOHs are called sequentially. In each call of a candidate H_i (lines 9 to 23), we run backtracking search with H_i as the variable ordering heuristic and set a restart condition to *failLimit* failures. If the failure number reaches the limit, then we record the search tree information and restart the search with next candidate VOH. Every candidate builds a search tree in each round and the corresponding information is recorded, so if the problem is not solved at the probing procedure, then the information of *roundLimit* search trees will be recorded for each candidate.

Modern VOHs usually use some learning strategies during search, so we make all candidate VOHs to learn during the entire probing procedure. In other words, during the call of candidate H_i, the information used by H_i will be learned and updated. Meanwhile, the information used by the other candidates will be learned and updated. All the information for the candidates, such as the weighted degrees [25], the activities of variables [16], the conflict history [7] and the failure rates [13], will be accumulated throughout the procedure.

The portfolio strategy runs backtracking search with different VOHs, so if a CSP instance can be easily solved by one of the candidate VOHs, then it may be solved during the probing procedure; otherwise, we analysis the recorded information of search trees and use the proposed measurements to select an efficient candidate VOH for the instance.

5 Experiments

To examine the efficiency of the proposed approach, we perform extensive experimentation with the `MiniZinc` benchmark suite. Four candidate VOHs including `ABS` [16], $dom/wdeg^{ca.cd}$ (marked by `WDEG` in the following tables) [25], `CHS` [7] and `FRBA` [13] are involved. The performance of searching for the first solution or proving unsatisfiable are measured by cpu time in seconds and numbers of instances solved in a timeout limit of 1200 seconds. In the following, we present the results of all instances (`All`), the satisfiable instances (`Sat`) and the unsatisfiable instances (`UnSat`) respectively. The best one in each comparison is in bold. More details about the experiments can be found in the appendix.

Table 1 The performance of different measurements.

		`PTS`	`MinFD`	`MaxD`	$\frac{\texttt{MaxD}}{log(\texttt{MinFD})}$
`All`	`BestOne`	15.08%	17.23%	59.69%	**67.69%**
(325)	`BestTwo`	23.69%	26.77%	72.62%	**81.85%**
`Sat`	`BestOne`	15.63%	16.67%	64.58%	**72.22%**
(288)	`BestTwo`	23.96%	20.14%	75.35%	**83.33%**
`UnSat`	`BestOne`	10.81%	21.62%	21.62%	**32.43%**
(37)	`BestTwo`	21.62%	**78.38%**	51.35%	70.27%

Firstly, we examine the percentage of instances where `SEVOH` finds the best one among the four candidate VOHs with different measurements. The reward function of the `MAB`-based VOH can be adapted as a measurement for `SEVOH`, so we further involve the pruned tree size (`PTS`) [24] as a measurement here. The results are presented in Table 1. After eliminating the instances solved by the probing procedure and the instances where all candidates result in timeout, the table contains the results of 325 instances. The `BestOne` (`BestTwo`) row is the percentage of instances where the VOH selected by `SEVOH` is the best one (one of the best two candidates). It is shown that `PTS` is not suitable for the selection here. Although `MinFD` does not work well in finding efficient VOHs for the satisfiable instances, it works well in finding a good candidate which is one of the best two for the unsatisfiable ones. On the contrary, `MaxD` works well in the satisfiable instances, but it does not work well in the unsatisfiable ones. The observation indicates that we should use different measurements for satisfiable and unsatisfiable instances. However, the satisfiability of a CSP is not determinable before solving it, so we cannot pre-select a measurement for each instance. Thankfully, combining the two measurements, $\frac{\texttt{MaxD}}{log(\texttt{MinFD})}$ performs well in finding efficient VOHs for all the instances. For both the satisfiable and the unsatisfiable, it finds a good candidate which is one of the best two for more than 70% instances. Thus, we uses $\frac{\texttt{MaxD}}{log(\texttt{MinFD})}$ as the measurement in the following experiments.

Secondly, we examine how the parameter *roundLimit* affects the percentage of instances where `SEVOH` finds good candidates. The results are presented in Table 2. The `#Solved by Portfolio` is the number of instance solved by the probing procedure and the `Probing Time` is the average time cost of the probing procedure. It is shown that different *roundLimit* makes little affection for the percentages. With the increasing of *roundLimit*, the number of instances solved by the portfolio strategy increases, as well as the probing time cost. But there is a trend that the number of increased instances solved by the probing procedure is diminishing, e.g., at the beginning we have 808-785=23, and then 16, 7, 4, 2. It indicates that increasing the *roundLimit* of the portfolio may not always solve more instances. Thus, we should find the best VOH for the hard instances which can not be easily solved by any of the candidate VOHs. The *roundLimit* 100 results in the largest percentage of `BestTwo`, so we set *roundLimit* to 100 in the following experiments.

Table 2 The performance of the probing procedure with different *roundLimits*.

	50	100	200	300	400	500
`BestOne`	66.86%	67.69%	**67.81%**	66.79%	66.79%	67.29%
`BestTwo`	79.71%	**81.85%**	80.14%	78.83%	79.48%	79.70%
`#Solved by Portfolio`	785	808	824	831	835	**837**
`Probing Time`	**63.61**	78.08	94.78	103.29	109.16	114.42

Thirdly, we compare `SEVOH` with the candidate VOHs and the `RestartMAB` (marked by $\texttt{RMAB}_{\texttt{PTS}}$) [24] which is a `MAB`-based VOH with its default reward function `PTS`. Besides the $\texttt{RMAB}_{\texttt{PTS}}$, we further involve a strategy (marked by $\texttt{RMAB}_{\frac{\texttt{Max}}{\texttt{Min}}}$) that adapts $\frac{\texttt{MaxD}}{log(\texttt{MinFD})}$ as the reward function in the `MAB`-based framework. In Table 3, we present the number of instances solved (`#Solved`), the average cpu time in solving the instances that are solved by all the compared VOHs (average time of all-solved instances, `AST`) and the average cpu time in solving all the instances which are solved by at least one of the VOHs (average time of all instances, `AllT`). The integer in the brackets after `AST` is the number of all-solved instances, so is the one after `AllT`. The time cost of a timeout run is counted as 1200 seconds. It is shown that `SEVOH` performs better than all the others in solving the satisfiable instances. Although `SEVOH` is outperformed by $\texttt{RMAB}_{\frac{\texttt{Max}}{\texttt{Min}}}$ in solving the unsatisfiable instances, it solves only 1 instance less than the bests. Thus, `SEVOH` gets the best overall performance.

Table 3 The overall performance.

		ABS	CHS	WDEG	FRBA	$\texttt{RMAB}_{\texttt{PTS}}$	$\texttt{RMAB}_{\frac{\texttt{Max}}{\texttt{Min}}}$	SEVOH
`All`	`#Solved`	630	730	700	985	894	933	**1056**
	`AST` (464)	34.91	26.67	57.91	23.65	18.29	11.87	**9.18**
	`AllT` (1131)	573.29	461.88	500.95	190.68	322.87	266.12	**142.38**
`Sat`	`#Solved`	570	679	648	919	830	867	**991**
	`AST` (416)	35.76	27.59	60.98	21.16	20.11	12.81	**8.20**
	`AllT` (1059)	590.51	467.87	509.69	186.42	333.75	273.07	**137.77**
`Unsat`	`#Solved`	60	51	52	**66**	64	**66**	65
	`AST` (48)	27.57	18.71	31.38	45.26	**2.54**	3.65	17.70
	`AllT` (72)	320.11	373.76	372.46	253.44	**162.81**	163.81	210.12

It has been shown in Table 2 that the probing procedure of `SEVOH` solves a number of instances. This is because the portfolio strategy works well in solving most of the instances that can be easily solved by at least one of the candidates. We are wondering how `SEVOH` performs in solving the hard instances that cannot be solved by the probing procedure, e.g., those cannot be easily solved by any of the candidates. Thus, in Table 4, we present the results of such instances. We can see that `SEVOH` solves the largest numbers of both satisfiable instances and unsatisfiable instances.

Table 4 Results of the instances that cannot be solved by the probing procedure of `SEVOH`.

		ABS	CHS	WDEG	FRBA	RMAB$_{\text{PTS}}$	RMAB$_{\frac{\text{Max}}{\text{Min}}}$	SEVOH
All	#Solved	121	91	101	224	133	151	**248**
	AST (53)	59.83	116.44	251.29	130.97	48.57	**38.55**	63.96
	AllT (323)	807.24	902.62	898.90	414.15	776.49	695.10	**395.78**
Sat	#Solved	87	64	75	193	97	115	**212**
	AST (29)	**64.03**	182.01	407.54	164.69	84.61	64.48	87.84
	AllT (280)	876.52	966.57	954.09	400.46	863.27	768.48	**408.00**
Unsat	#Solved	34	27	26	31	**36**	**36**	**36**
	AST (24)	54.76	37.22	62.48	90.23	**5.02**	7.23	35.10
	AllT (43)	356.07	486.19	539.55	503.26	**211.46**	217.28	316.24

Finally, we compare the performance of the three adaptive strategies. Three different combinations of candidates are considered. The results are presented in Table 5. It is shown `SEVOH` solves the largest numbers of instances in all the combinations of candidates. When combining three candidates, `SEVOH` costs more time than the others in solving the all-solved instances. This is because it has poor performance in solving some of the instances. However, its average time cost of solving all the instances which are solved by at least one of the 9 strategies is the least one.

Table 5 The performance of the adaptive strategies with different candidates.

	ABS+CHS			ABS+CHS+WDEG			ABS+CHS+WDEG+FRBA		
	RMAB$_{\text{PTS}}$	RMAB$_{\frac{\text{Max}}{\text{Min}}}$	SEVOH	RMAB$_{\text{PTS}}$	RMAB$_{\frac{\text{Max}}{\text{Min}}}$	SEVOH	RMAB$_{\text{PTS}}$	RMAB$_{\frac{\text{Max}}{\text{Min}}}$	SEVOH
#Solved	754	759	**814**	755	796	**846**	894	933	**1056**
AST (683)	26.92	23.35	**21.71**	**24.08**	28.85	28.72	26.73	24.75	**19.48**
AllT (1111)	424.87	415.07	**358.96**	423.39	398.71	**335.55**	307.08	249.31	**123.34**

6 Conclusion

In this paper, we propose a portfolio-based approach to select efficient variable ordering heuristics for CSPs. Extensive experimentations performed on `MiniZinc` benchmark suite demonstrate that the portfolio strategy is effective in solving the instances that can be easily solved by some of the candidate VOHs. Besides, the measurement combining the information of minimum failure depth and the maximum depth of search trees is effective to select good candidate VOHs for different CSPs. With the measurement, `SEVOH` finds the best one from four candidate VOHs for more than 67% instances. Consequently, `SEVOH` solves more instances than all the candidate VOHs and the adaptive VOH based on Multi-Armed Bandit. It could be an effective adaptive search strategy for black-box CSP solvers.

References

1 R. Amadini, M. Gabbrielli, and J. Mauro. Portfolio approaches for constraint optimization problems. *Annals of Mathematics and Artificial Intelligence*, 76:229–246, 2016.

2 A. Balafrej, C. Bessiere, and A. Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proc. IJCAI'15*, pages 290–296. AAAI Press, 2015.

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

4 G. Chu and P. J. Stuckey. Learning value heuristics for constraint programming. In *Proc. CPAIOR'15*, pages 108–123. Springer, 2015.

5 S. Epstein and S. Petrovic. Learning to solve constraint problems. In *Proc. ICAPS'07, Workshop on Planning and Learning*, 2007.

6 C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI'98*, pages 431–437. AAAI, 1998.

7 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.

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

9 H. Hurley, L. Kotthoff, Y. Malitsky, and B. O'Sullivan. Proteus: a hierarchical portfolio of solvers and transformations. In *Proc. CPAIOR'14*, 2014.

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

11 H. Li, J. H. Lee, H. Mi, and M. Yin. Finding good subtrees for constraint optimization problems using frequent pattern mining. In *Proc. AAAI'20*, pages 1577–1584, 2020.

12 H. Li, Y. Liang, N. Zhang, J. Guo, D. Xu, and Z. Li. Improving degree-based variable ordering heuristics for solving constraint satisfaction problems. *Journal of Heuristics*, 22(2):125–145, 2016.

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

15 M. Loth, M. Sebag, Y. Hamadi, and M. Schoenauer. Bandit-based search for constraint programming. In *Proc. CP'13*, pages 464–480. Springer, 2013.

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 Anastasia Paparrizou and Kostas Stergiou. Evaluating simple fully automated heuristics for adaptive constraint propagation. In *Proc. of ICTAI'12*, volume 1, pages 880–885, 2012. doi:10.1109/ICTAI.2012.123.

19 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.

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.

22 T. Walsh. Search in a small world. In *Proc. IJCAI'99*, pages 1172–1177, 1999.

23 R. Wang, W. Xia, and R. H. C. Yap. Correlation heuristics for constraint programming. In *Proc. ICTAI'17*, pages 1037–1041. IEEE, 2017.

24 H. Wattez, F. Koriche, C. Lecoutre, A. Paparrizou, and S. Tabary. Learning variable ordering heuristics with multi-armed bandits and restarts. In *Proc. ECAI'20*, pages 371–378. IOS Press, 2020.

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

26 R. J. Woodward, A. Schneider, B.Y. Choueiry, and C. Bessiere. Adaptive parameterized consistency for non-binary csps by counting supports. In *Proc. of CP'14*, pages 755–764, 2014.

27 W. Xia and R. H. C. Yap. Learning robust search strategies using a bandit-based approach. In *Proc. AAAI'18*, pages 6657–6665. AAAI, 2018.

28 L. Xu, F. Hutter, H. H Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

A Details of The Experiments

A.1 Environment

The experiments were run in Choco solver (version 4.10.6) [20] where all the candidate VOHs are already implemented. The environment is JDK8 under CentOS 6.4 with 4 Intel Xeon CPU E7-4820@2.00GHz processors and 58 GB RAM. Each run is allocated 1 GB RAM.

A.2 Benchmark

The benchmark suite are from `https://github.com/MiniZinc/minizinc-benchmarks`. The instances are flattened offline. After eliminating some large instances which cannot be flattened in 1 hour and the problems where unsatisfiable is proved at root node, we include 46 `MiniZinc` models with 1876 instances in the experiments.

A.3 Restart

The $failLimit$ of the probing procedure of `SEVOH` is set to the number of variables. If the instance contains less than 100 variables, the $failLimit$ is set to 100. A geometric restart strategy [6, 22] is equipped by the second phase of `SEVOH` and the candidates VOHs. The growing factor is 1.1 and the initial cutoff is 10 failures. The restart strategy of the `MAB`-based framework is the Luby restart which is the default one used in [24].

A.4 Searching

Binary branching strategy [10] is used throughout the experiments. `ABS` uses its default value selector and all the others use lexicographic ordering as the value selector. We forbid the sampling procedure of `ABS` when it is used as a candidate, because the probing procedure of `SEVOH` warms up all the candidates. When `ABS` is used alone, we run it with its default settings.

A.5 Other Details

In Table 1 and Table 2, we need to determine the best candidate for each instance. Thus, for each instance, we run each candidate H_i with 10 random seeds from 1 to 10 and use the average time cost of the 10 runs as the performance of H_i solving the instance. The best VOH for a instance is the one costing least cpu time. Then we run the probing procedure with random seed 0 to select a VOH for each instance. The results in Table 1 are obtained with *roundLimit* 100. In Tables 3, 4 and 5, we have used a unique random seed 0 in the experiments. The instances where all the seven VOHs result in timeout are eliminated.