Variable-Oriented Adaptive Singleton Consistency

Yaling Wu[1] , **Hongbo Li**[2*] , **Minghao Yin**[1]

[1]School of Information Science and Technology, Northeast Normal University, Changchun, China.
[2]School of Computing and Artificial Intelligence, Shanghai University of Finance and Economics, Shanghai, China.
lihb905@nenu.edu.cn

Abstract

Singleton consistency significantly reduces the search space of backtracking search for solving constraint satisfaction problems (CSP). However, it is too expensive to be used throughout the search, and it has not been successfully applied in solving general CSPs. In this paper, we propose a variable-oriented adaptive singleton consistency, namely VOASC, that could be applied in general-purpose constraint solvers. It calculates the filtering efficiency of local consistency enforced by the default propagation engine and the singleton-based consistency for each variable, and then selects an appropriate consistency to propagate the decisions of the variable in subsequent searches. We performed extensive experimentation with the MiniZinc benchmark suite. VOASC demonstrates superior solving performance on both CSP and COP, compared to the two candidate local consistencies and several existing adaptive propagation methods.

1 Introduction

Constraint propagation is the cornerstone of solving constraint satisfaction problems. It enforces local consistencies to prune the search space of backtracking search. Generalized arc consistency (GAC) [Bessière and Régin, 1997] has been the most popular level of local consistency in constraint solvers. In the past decades, a number of consistencies that are stronger than GAC have been developed [Debruyne and Bessière, 1997; Bennaceur and Affane, 2001; Bessière *et al.*, 2008; Lecoutre *et al.*, 2013; Karakashian *et al.*, 2010]. However, these stronger consistencies are generally too expensive to be used during the whole search. Thus, some efforts have been made on adaptive consistency [Stergiou, 2008; Paparrizou and Stergiou, 2012; Balafrej *et al.*, 2013; Balafrej *et al.*, 2014; Woodward *et al.*, 2014; Balafrej *et al.*, 2015; Woodward *et al.*, 2018]. These studies demonstrate the potential of adaptive consistencies for enhancing the performance of backtracking search. However, most of them focus on particular kinds of problems defined with specific constraints, such as binary constraints [Stergiou, 2008; Balafrej *et al.*, 2013; Balafrej *et al.*, 2015], and table constraints [Paparrizou and Stergiou, 2012; Woodward *et al.*, 2014]. To the best of our knowledge, none of them has been applied in general-purpose constraint solvers.

In constraint solvers, the default propagation engines usually employ GAC and bound consistency (BC) [Collavizza *et al.*, 1998] for most of the constraints. It is non-trivial to implement the stronger consistencies concerning multiple constraints, such as maxRPC [Debruyne and Bessière, 1997] and maxRPWC [Bessière *et al.*, 2008]. On the contrary, singleton consistency is relatively easier to implement in general-purpose solvers.

In this paper, we investigate how to improve the performance of a general-purpose constraint solver by utilizing the stronger filtering ability of singleton consistency. We propose a variable-oriented adaptive singleton consistency (namely VOASC) that can be applied in general-purpose constraint solvers. It considers switching between the local consistency enforced by the default propagation engine, denoted by default local consistency (DLC), and the singleton consistency (SC) [Debruyne and Bessière, 1997] built on DLC. We evaluate the filtering efficiency of a candidate consistency by the number of values filtered by it per unit time. At each subsequent search tree node, we select an unassigned variable and then determine whether the DLC or the SC should be used to propagate the assignment of the variable according to their filtering efficiencies. Furthermore, we adapt the method to apply it in solving constraint optimization problems (COP). We performed extensive experimentation with the MiniZinc benchmark suite. The experiments were conducted in Choco solver under three efficient variable ordering heuristics, $dom/wdeg^{ca.cd}$ [Wattez *et al.*, 2019], FRBA [Li *et al.*, 2021] and pick/dom [Audemard *et al.*, 2023]. VOASC is compared with the DLC, the SC, and some adaptive propagation methods, including an existing adaptive singleton-based consistency [Balafrej *et al.*, 2014], a reinforcement-learning-based adaptive propagation method [Woodward *et al.*, 2014], and a reactive propagation strategy PrePeak+ [Woodward *et al.*, 2018]. The results demonstrate the superior solving performance of VOASC on both CSP and COP.

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 $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,

*corresponding author

$\mathcal{D}$ is a set of domains $\mathcal{D}$ =$\{dom(x_1), dom(x_2), ..., dom(x_n)\}$, where $dom(x_i)$ is a finite set of possible values for variable x_i, and $\mathcal{C}$ is a set of e constraints $\mathcal{C}$=$\{c_1, c_2, ..., c_e\}$. Each constraint c consists of two parts, an ordered set of variables $scp(c) = \{x_{i1}, x_{i2}, ..., x_{ir}\}$ and a subset of the Cartesian product $dom(x_{i1}) \times dom(x_{i2}) \times ... \times dom(x_{ir})$ that specifies the allowed combinations of values for the variables $\{x_{i1}, x_{i2}, ..., x_{ir}\}$. An element of $dom(x_{i1}) \times dom(x_{i2}) \times ... \times dom(x_{ir})$ is called a tuple on $scp(c)$, denoted by τ. $\tau[x_i]$ is the value of x_i in τ. A tuple τ is valid iff for each $x_i \in scp(c)$, $\tau[x_i]$ is in $dom(x_i)$. We use $\mathcal{P}(x_i = v)$ to denote the CSP obtained by restricting $dom(x_i)$ to $\{v\}$ in $\mathcal{P}$. An *assignment* $\mathcal{A}$ to variables $X \subseteq \mathcal{X}$ is a set of *instantiations* of the form $(x_i = v)$, one for each $x_i \in X$ to assign v to x_i where $v \in dom(x_i)$. An assignment to $\mathcal{X}$ satisfying all constraints is a solution to $\mathcal{P}$.

A CSP is satisfiable if it has at least one solution; otherwise unsatisfiable. Solving a CSP involves either finding a solution or determining that no solution exists. Backtracking search is widely used in solving CSPs. It performs a depth-first traversal of a search tree. At each search tree node, an unassigned variable is selected, typically using a variable ordering heuristic (VOH), and a new node is generated after the instantiation of this variable; then a propagation algorithm is applied to filter those inconsistent values [Bessière, 2006]. If the propagation of the current decision leads to failure, e.g., a domain wipeout, then one or more decisions must be canceled, and backtracking occurs.

When an objective function $\mathcal{F}$ is added to a CSP, we obtain a constraint optimization problem (COP). The objective function $\mathcal{F}$ maps every feasible solution to a real number, called the objective. Solving a COP $\mathcal{P}$ involves finding an optimal solution, which is the feasible solution of $\mathcal{P}$ with the best objective. However, for many real-world COP scenarios, finding a high-quality solution quickly is often the primary requirement. When solving a COP, backtracking search is adapted into branch-and-bound search (B&B). The decision process and propagation process in solving CSPs and COPs are similar, the difference is that whenever a new feasible solution $\mathcal{S}$ is found by B&B, $\mathcal{F}(\mathcal{S})$ is used to bound subsequent search to ensure the next feasible solution found must be better than $\mathcal{S}$. Search continues until no more feasible solutions are found after exhausting the search space. The last solution found is an optimal solution of $\mathcal{P}$.

Definition 1 (Generalized Arc Consistency)**.** *Given a CSP $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, a constraint $c \in \mathcal{C}$, and a variable $x_i \in scp(c)$,*

- *A value $(x_i = v)$ is consistent with c iff there exists a valid tuple τ allowed by c and $\tau[x_i] = v$.*
- *A constraint c is generalized arc consistent iff $\forall x_i \in scp(c)$, $dom(x_i) \neq \emptyset$ and $\forall v \in dom(x_i)$, $(x_i = v)$ is consistent with c.*
- *$\mathcal{P}$ is generalized arc consistent iff all the constraints of $\mathcal{C}$ are generalized arc consistent.*

While GAC checks the consistency of all the values in the domain of each variable, bound consistency checks only the lower bound and upper bound of the domain of each variable. The DLC of a CSP solver usually contains different consistency levels. For example, it may enforce BC for linear constraints, and GAC for table constraints [Lecoutre, 2011] and some global constraints [Régin, 1994].

Singleton consistency was first proposed as singleton arc consistency (SAC)[1][Debruyne and Bessière, 1997]. SAC can be extended to a more general singleton consistency that replaces the underlying consistency, AC, with other local consistencies, such as GAC or even the DLC of a constraint solver. In this work, we use DLC as the underlying consistency. In the following, singleton consistency (SC) denotes the singleton consistency built on the DLC of a constraint solver. DLC($\mathcal{P}(x_i = v)$) is the CSP after assigning v to x_i and running the DLC. Given a CSP $\mathcal{P}$, for each variable-value pair (x_i, v), the SC checks and removes the v from $dom(x_i)$ if DLC($\mathcal{P}(x_i = v)$) leads to a failure.

Besides constraint propagation, the restart technique [Gomes *et al.*, 1998] makes constraint solvers more robust in solving various types of problems. It terminates the run of the current search and restarts a new search from the root node when a predefined restart condition is reached. The aim is to switch search entrances to prevent the search from being trapped in exploring a part of the search space far from a solution. It has been shown that a sequence of short runs instead of a single long run may be a more effective use of computational resources [Gomes *et al.*, 2000].

3 Variable-Oriented Adaptive Singleton Consistency

In constraint solvers, singleton consistency (SC) is more powerful in filtering inconsistent values than the default local consistency (DLC). However, in some cases, SC costs much more computational resources than DLC, yet achieves only marginally better filtering performance. Thus, the key issue is to determine when singleton consistency should be applied, ensuring computational resources are allocated exclusively to effective domain filtering. To address the issue, we develop a variable-oriented adaptive singleton consistency. The method dynamically evaluates the performance of different consistencies propagating the decisions of each variable, and determines the most effective consistency for the variable, which could yield better filtering performance. In the following, we first introduce the evaluation of the filtering performance of different consistencies, and then propose the adaptive singleton consistency based on the evaluation.

3.1 Evaluation of Filtering Performance

To evaluate the filtering performance of a local consistency LC, we consider the number of values filtered by LC per unit time, called filtering efficiency (FE). When a local consistency LC is applied to propagate the decision of a variable x_i, we count the total number of values filtered by the propagation (marked by *filterNum*) and record the time cost (marked by $duration$) of the propagation, then we can calculate the FE of the propagation by $\frac{filterNum}{duration}$. For each variable x_i, we use a counter $proNum_{LC}[i]$ for the number of propagations on x_i with LC, and an accumulator $accFE_{LC}[i]$ to aggregate the FEs generated during these propagations. Then

[1]Arc consistency (AC) is the GAC in binary CSPs.

Algorithm 1: ADAPTIVE PROPAGATION

Input: x_i: a variable; v: a value in $dom(x_i)$
Output: *isConsistent: boolean*

1 *before* ← sum of the domain size of all variables;
2 *scTriggered* ← ISSINGLETONCONSISTENCY(x_i);
3 *startTime* ← current timestamp;
4 **if** *scTriggered* **then**
5 *isConsistent* ← SCPROPAGATE(x_i, v);
6 **else**
7 *isConsistent* ← DLCPROPAGATE(x_i, v);
8 $duration$ ← current timestamp - $startTime$;
9 *after* ← sum of the domain size of all variables;
10 *filterNum* ← *before* - *after*;
11 *FE* ← $\frac{filterNum}{duration}$;
12 **if** $scTriggered$ **then**
13 $proNum_{SC}[i] \leftarrow proNum_{SC}[i] + 1$;
14 $accFE_{SC}[i] \leftarrow accFE_{SC}[i]$+ *FE*;
15 **else**
16 $proNum_{DLC}[i] \leftarrow proNum_{DLC}[i] + 1$;
17 $accFE_{DLC}[i] \leftarrow accFE_{DLC}[i]$+ *FE*;
18 **return** $isConsistent$;

we can calculate the average FE of propagating x_i with LC by $\frac{accFE_{LC}[i]}{proNum_{LC}[i]}$, marked by $aveFE_{LC}[i]$.

Given two local consistencies of a constraint solver, DLC and SC. We can calculate the scores for them on each variable x_i, e.g., $aveFE_{DLC}[i]$ and $aveFE_{SC}[i]$, and the one with the larger score is preferred to propagate the decisions of x_i.

3.2 The Adaptive Propagation Algorithm

The adaptive propagation algorithm is presented in Algorithm 1 and Algorithm 2. The Algorithm 1 is called at each search tree node to adaptively select a local consistency to propagate the decision of x_i.

Algorithm 1 selects a candidate consistency at line 2 by the function ISSINGLETONCONSISTENCY presented in Algorithm 2, and then the selected consistency is employed to propagate the decision of x_i at line 5 (by SC) or line 7 (by DLC). The propagation result, e.g., a failure is detected or not, is also recorded here. The duration of the propagation is obtained at line 8, the number of filtered values is calculated at line 10, and the FE of the propagation is calculated at line 11. The counter $proNum_{SC}[i]$ is updated at line 13, and the accumulator $accFE_{SC}[i]$ is aggregated at line 14. The corresponding data of DLC are updated at lines 16-17. Finally, the propagation result is returned at line 18.

Algorithm 2 determines whether SC should be applied to propagate the decision of x_i. Our study originates from tackling the problem instances that are hard for DLC to solve, so we employ a preprocessing phase at the beginning of the resolution to eliminate the easy instances, so that only the hard instances proceed to the next phase that activates the adaptive singleton consistency.

In the preprocessing phase, the search always selects DLC to propagate the decisions of all variables. The preprocessing phase terminates when either a limited restart number *preLimit* (a parameter) is reached or all variables have the FE information of DLC. In Algorithm 2, the *preprocessing* (initialized to true) and *last* (initialized to 0) are global variables. The *restartNum* is also a global variable that records the restart number. If *restartNum* is larger than *preLimit*, the *preprocessing* label is set to false at line 7; otherwise, it checks whether there are variables lacking FE information at lines 3-6. If $proNum_{DLC}[j]$ is 0, it indicates that the variable x_j has never been propagated by DLC and has no FE information, so the resolution is still in the preprocessing phase and x_i should be propagated by DLC (false is returned at line 6). Note that the *last* records the last variable identified as lacking FE information and it is used globally. When checking whether there are variables lacking FE information, we start from the last identified one. If all variables already have FE information, the *preprocessing* label will be set to false at line 7. After the preprocessing phase is terminated, given a variable x_i, the adaptive selection is divided into four cases:

Algorithm 2: ISSINGLETONCONSISTENCY

Input: x_i: a variable
Output: *true* for SC; *false* for DLC;

1 **if** *preProcessing* **then**
2 **if** *restartNum* < *preLimit* **then**
3 **for** $j = last$ *to* $|\mathcal{X}| - 1$ **do**
4 **if** $proNum_{DLC}[j] = 0$ **then**
5 $last \leftarrow j$;
6 **return** *false*;
7 *preProcessing* ← *false*;
8 **if** $proNum_{DLC}[i] > 0$ ***and*** $proNum_{SC}[i] > 0$ **then**
9 $aveFE_{DLC}[i] \leftarrow \frac{accFE_{DLC}[i]}{proNum_{DLC}[i]}$;
10 $aveFE_{SC}[i] \leftarrow \frac{accFE_{SC}[i]}{proNum_{SC}[i]}$;
11 **return** $aveFE_{DLC}[i] < aveFE_{SC}[i]$;
12 **else if** $proNum_{DLC}[i] > 0$ ***and*** $proNum_{SC}[i] = 0$ **then**
13 **return** *true*;
14 **else if** $proNum_{DLC}[i] = 0$ ***and*** $proNum_{SC}[i] > 0$ **then**
15 **return** *false*;
16 **else**
17 *dlcBetter* ← 0;
18 *scBetter* ← 0;
19 **for** $k = 0$ *to* $|\mathcal{X}| - 1$ **do**
20 **if** $proNum_{DLC}[k] > 0$ ***and*** $proNum_{SC}[k] > 0$ **then**
21 $aveFE_{DLC}[k] \leftarrow \frac{accFE_{DLC}[k]}{proNum_{DLC}[k]}$;
22 $aveFE_{SC}[k] \leftarrow \frac{accFE_{SC}[k]}{proNum_{SC}[k]}$;
23 **if** $aveFE_{DLC}[k] \geq aveFE_{SC}[k]$ **then**
24 *dlcBetter* ← *dlcBetter* + 1;
25 **else**
26 *scBetter* ← *scBetter* + 1;
27 **return** *dlcBetter* < *scBetter*;

(1) both SC and DLC have FE information; (2) DLC has FE information and SC does not; (3) SC has FE information and DLC does not; (4) neither of them has the information.

Whether a candidate has FE information on a variable x_i can be reflected by the $proNum_{LC}[i]$, i.e., $proNum_{LC}[i] > 0$ indicates that the candidate LC has FE information. The first case is confirmed at line 8, and then we calculate the $aveFE_{DLC}[i]$ and $aveFE_{SC}[i]$ at line 9 and line 10, respectively. The selection will be made at line 11.

In cases (2) and (3), we prefer to select the one lacking FE information. Thus, in the second (or third) case confirmed at line 12 (or line 14), our strategy is to directly select SC (or DLC) at line 13 (or line 15).

In the last case, none of the candidates has FE information on x_i. This happens only when the preprocessing phase is terminated by the restart limit (the condition at line 2) and x_i has never been assigned during the preprocessing phase. We use the FE information of other variables to determine which candidate should be tried first on x_i. Specifically, we choose the variables that have learned FE information for both SC and DLC (the variables identified at line 20) to vote for the decision. We use two counters *dlcBetter* and *scBetter* to record the number of variables voting for the candidates, respectively. For each voter variable x_k, the $aveFE_{DLC}[k]$ and $aveFE_{SC}[k]$ are calculated at line 21 and line 22, respectively. The voting choice is made at lines 23 to 26, and the final decision is made at line 27.

In binary branching [Hwang and Mitchell, 2005], Algorithm 1 is used to propagate only positive decisions, and the candidate consistency used to propagate each positive decision is recorded. If the search backtracks and the corresponding negative decision is generated, then the recorded consistency will be used to do the propagation directly.

3.3 Adaptation for COP

The method introduced before is designed for solving CSPs. It can be directly applied in solving COPs. However, solving a COP involves solving a series of CSPs. With the bounds of the objective being tightened, the newly generated CSPs become increasingly difficult to solve. Consequently, the FE information gathered through the initial preprocessing procedure may lose its effectiveness. To address the issue, we rerun the preprocessing procedure when we find it difficult to find a new solution. There are various criteria for determining when a problem enters a hard-to-solve phase. We employ a simple strategy here, which triggers the preprocessing procedure when no new feasible solution is found after q restarts, where q is a parameter.

4 Related Work

The following approaches can be considered as constraint-oriented adaptive constraint propagation, since they associate each constraint with a local consistency. (1) Stergiou proposed adaptive propagation approaches [Stergiou, 2008] that use some heuristics to dynamically switch between enforcing a weak local consistency (AC) and a strong local consistency (maxRPC). The heuristics monitor the DWO (domain wipe out) and the value deletions led by each constraint. If a constraint leads to a DWO in current propagation or it deletes at least a value, it will be propagated by the strong local consistency at the next time. Later, Paparrizou and Stergiou extended the approach to non-binary CSPs [Paparrizou and Stergiou, 2012], e.g., the weak local consistency is GAC, and the strong local consistency is maxRPWC. (2) Stamatatos and Stergiou proposed to run a random probing preprocess [Stamatatos and Stergiou, 2009] to learn which local consistency should be used on each constraint. It gathers information regarding the percentage of fruitful revisions for each constraint and the values filtered by different propagation methods, e.g., BC, AC and maxRPC. Consequently, the constraints can be clustered and the solver could automatically choose the local consistency for each constraint.

The following approaches can be considered as node-oriented adaptive constraint propagations, since they choose to enforce a specific local consistency at certain search tree nodes according to some heuristic. (1) The adaptive partition-one-AC (APOAC) [Balafrej *et al.*, 2014] is an adaptive singleton consistency which is closely related to our method. It interrupts the propagation process (in particular, the singleton tests) when the pruning effect of the singleton tests has faded. Specifically, APOAC alternates between a learning phase and an exploitation phase during search. In the learning phase, it tries to learn a value k for the maximum times that a series of singleton tests on all values of a variable should be performed in the exploitation phase. The main idea of APOAC can be applied to other singleton consistencies, and we will compare its adaptation of SC with our method in the experiments. (2) The adaptive propagation method [Balafrej *et al.*, 2015] based on reinforcement learning switches among AC, maxRPC and POAC at each depth of the search tree. It associates each depth with a multi-armed bandit (MAB) selector that prefers the local consistency *i* that maximizes:

$$\rho(i) = \overline{R}_i + \sqrt{\frac{2 \ln m}{m_i}} \tag{1}$$

where $\overline{R}_i$ is the mean of the past rewards of local consistency *i* at the current depth, m_i is the number of times local consistency *i* was selected and m is the current number of all trials. The reward is the CPU time needed to enforce local consistency *i* at the m-th visit of a node at the given depth plus the time to explore the subtree rooted there. (3) The last one we should mention is a reactive strategy that triggers strong local consistency when the search starts thrashing, namely PrePeak [Woodward *et al.*, 2018]. Specifically, it records the backtrack times of each search tree level and uses a threshold θ to determine whether the strong local consistency should be triggered at the current level. The θ is initialized to the maximum value of the recorded backtrack times in the first n^2 backtracks, and it is increased (or decreased) if applying the strong local consistency leads to a failure (or not). A variant of PrePeak, namely PrePeak+ employs a strategy to terminate the propagation of the strong local consistency either by the size of the propagation queue or by the run time.

5 Experiments

To examine how the variable-oriented adaptive singleton consistency performs in general-purpose constraint solvers, we

perform extensive experimentation with the MiniZinc benchmark suite [Stuckey *et al.*, 2014]. The following are some detailed settings of the experiments:

- **Environment.** The environment is JDK21 under Ubuntu 24.04 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 with 1 core and 8GB RAM.
- **Solver.** The experiments were conducted in an efficient and well-architected CP solver, Choco [Prud'homme *et al.*, 2017]. The source code of our algorithm is available at https://github.com/tiger510520/VOASC
- **Baselines.** We compare VOASC against the two candidate consistencies: the DLC of Choco and the SC built on the DLC. Some adaptive propagation methods including the MAB-based adaptive propagation (denoted by MABAP) [Balafrej *et al.*, 2015], the existing adaptive singleton-based consistency (denoted by ASC) [Balafrej *et al.*, 2014] and PrePeak+ [Woodward *et al.*, 2018], are also involved. These adaptive propagation methods are adopted to make a selection between DLC and the SC.
- **Instances.** We have considered two CSP instance sets from the MiniZinc benchmark suite. The first set contains 103 instances[2] used in recent MiniZinc challenges (from 14 problems). The second set contains 362 instances (from 46 problems) selected from the MiniZinc instance set[3] available at https://github.com/MiniZinc/MiniZinc-benchmarks, which will be used in Tables 5 to 6. For the COP experiments, we considered the instances used in the 2024 and 2025 MiniZinc Challenges.
- **Search Strategies.** We have used three variable ordering heuristics, $dom/wdeg^{ca.cd}$ [Wattez *et al.*, 2019], FRBA [Li *et al.*, 2021] and pick/dom [Audemard *et al.*, 2023]. The value selector for CSP always selects the minimum value. For COP, the value selector is solution-based phase saving [Demirovic *et al.*, 2018] with the underlying value selector that selects the minimum value [4]. All the search strategies are equipped with the Luby [Luby *et al.*, 1993] restart with an initial cutoff of 100 and the restart condition is the number of failures.
- **Nogood-Recording.** All the compared search strategies are equipped with the *Reduced nld-Nogoods* [Lecoutre *et al.*, 2007].
- **Random seed.** The random seed 0 is used throughout the experiments, mainly used by variable ordering heuristics to break ties.

[2]There are 105 CSP instances used in 2014-2023 years, but two of them are duplicate ones used in two years.

[3]After eliminating some large instances which cannot be flattened in 1 hour, there are 46 problems with 1876 instances. For each problem, if there are more than 10 instances, 10 instances are randomly selected; otherwise, all the instances are selected.

[4]We also tested the RLARF [Delecluse and Schaus, 2024] heuristic as the underlying value selector, and the value selectors without solution-based phase saving. The presented combination gets the best performance.

- **Metrics.** For CSP, the performance of searching for the first solution or proving unsatisfiable are measured by the number of instances solved (#Sol in the tables) in a timeout limit of 3600 seconds and the PAR2 score[5] of runtime. For COP, we have used the primal gap [Berthold, 2013] to evaluate the performance of finding feasible solutions and finding solutions with better objectives. For each tested instance, it gives a score between 0 and 1 to each candidate method defined as follows: if the candidate did not find a feasible solution or obj is opposite in sign to the obj_{best}, 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, it gets 1 score; otherwise, it gets a score $\frac{|obj_{best}-obj|}{max\{|obj_{best}|,|obj|\}}$. The smaller score indicates a better performance. We also considered the number of instances where optimality is proved.
- In the following tables, the best one in each comparison is highlighted in bold.

	Restart	Full	Lightweight
SC	no	49	54
	yes	54	**57**
ASC	no	55	–
	yes	**59**	–
MABAP	no	60	61
	yes	60	**63**
PrePeak	no	61	62
	yes	62	**63**

Table 1: Whether restart and lightweight SC perform better. The number of instances solved by different strategies is conducted with $dom/wdeg^{ca.cd}$. The Full and Lightweight columns indicate the full version of SC that propagates until a fixed point and a lightweight version that checks each variable-value pair only once.

	DLC	SC	VOASC	Virtual
#Sol	58	57	**63**	67
PAR2	3245	3482	**2879**	2672

Table 2: Examination of the effectiveness of FE. For each instance, we run each candidate offline for 1 hour to learn the FE information on each variable. The virtual column is the strategy that selects the better one of DLC and SC in hindsight. With the pre-learned FE information, we compared VOASC against DLC and SC.

preLimit	10	50	100	300	500	800
#Sol	64	64	**66**	65	63	63
PAR2	2820	2859	**2743**	2831	2885	2908

Table 3: How $preLimit$ affects the performance of VOASC

[5]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.

Table 1 presents the results of whether restart and lightweight SC perform better. We can see that both the lightweight version of SC and the restart strategy improve the performance of the naive version of the baselines. Thus, in the following experiments, we use the lightweight version of SC for singleton consistency.

Secondly, we examine whether the filtering efficiency is effective in making the selection, e.g., if the FE information is pre-computed offline, how does VOASC perform? The results are presented in Table 2. We can see that although VOASC is outperformed by the virtual strategy, it performs much better than the two candidates. The results show that the FE is effective in making the selection.

Thirdly, we investigate the impact of the parameter *preLimit* on the performance of VOASC. While a total of 11 values were tested under the $dom/wdeg^{ca.cd}$, for the sake of brevity and to highlight the key performance trends, Table 3 presents the results for 6 representative values. We can observe that the optimal performance is achieved at 100. Thus, we set *preLimit* to 100 for all subsequent experiments.

Next, we evaluate the performance of VOASC against the baseline methods under two distinct variable ordering heuristics: FRBA and pick/dom, providing a more rigorous test of the algorithms' generalizability. The results are presented in Table 4. In the FRBA columns, we can see that both MABAP and PrePeak+ perform better than DLC and SC in both the number of solved instances and PAR2 score. VOASC brings significant improvement over the existing approaches. It solves 11 more instances than the best baseline, PrePeak+. A similar phenomenon can be observed in the pick/dom columns. Although PrePeak+ works well when combined with pick/dom heuristic, it is still outperformed by VOASC. With both of the variable ordering heuristics, PrePeak+ constructs identical search trees to DLC in much more instances than VOASC, indicating that PrePeak+ prefers to leverage the solving power of DLC. There are only 5 unsatisfiable instances, which may be insufficient to capture the performance characteristics. In addition, since the parameter setting *preLimit* = 100 was derived from the 103 instance set, we further evaluate the performance of VOASC on the second instance set to verify its generalization capability in Table 5.

The second instance set contains more than 30 unsatisfiable

	FRBA						pick/dom					
	ALL(82)		SAT(77)		UNSAT(5)		ALL(82)		SAT(77)		UNSAT(5)	
	#Sol	PAR2	#Sol	PAR2	#Sol	PAR2	#Sol	PAR2	#Sol	PAR2	#Sol	PAR2
DLC	64	2838	59	1826	5	**104**	68	2674	63	1595	5	294
SC	58	3342	54	2392	4	1766	54	3640	50	2762	4	2222
ASC	63	3017	59	1959	4	1731	55	3554	51	2644	4	2250
MABAP	66	2838	61	1814	5	285	63	2968	58	1971	5	541
PrePeak+	66(42)	2681	61	1609	5	212	73(53)	2398	68	1211	5	503
VOASC	**77**(26)	**2063**	**72**	**783**	**5**	193	**77**(38)	**2095**	**72**	**823**	**5**	**253**

Table 4: Comparing VOASC against the baselines under FRBA and pick/dom heuristics. The integers in the brackets in the second row are the numbers of instances involved in the corresponding categories. We have tested 103 instances, and 21 of them are not solved by any of the strategies, so they are not involved in the results. The integers in the brackets of the #Sol columns are the number of instances where the corresponding strategy builds a search tree identical to DLC, i.e., the SC has not been activated when solving these instances.

VOH	Category		DLC	SC	ASC	MABAP	PrePeak+	VOASC
$dom/wdeg^{ca.cd}$	ALL (299)	#Sol	256	250	249	265	263(218)	**278**(208)
		PAR2	2173	2322	2394	2001	2035	**1760**
	SAT (268)	#Sol	239	221	219	235	244(209)	**248**(199)
		PAR2	857	1378	1486	967	713	**625**
	UNSAT (31)	#Sol	17	29	**30**	**30**	19(9)	**30**(9)
		PAR2	3339	566	481	**376**	2960	515
FRBA	ALL (299)	#Sol	260	256	254	267	262(225)	**284**(203)
		PAR2	2084	2233	2305	1958	2037	**1616**
	SAT (268)	#Sol	242	227	225	238	244(215)	**254**(193)
		PAR2	761	1264	1351	891	699	**452**
	UNSAT (31)	#Sol	18	29	29	29	18(10)	**30**(10)
		PAR2	3125	519	600	532	3109	**337**
pick/dom	ALL (299)	#Sol	247	252	240	259	248(212)	**281**(194)
		PAR2	2361	2338	2542	2146	2344	**1700**
	SAT (268)	#Sol	230	222	211	230	231(201)	**251**(186)
		PAR2	1106	1409	1658	1129	1088	**550**
	UNSAT (31)	#Sol	17	30	29	29	17(11)	**30**(8)
		PAR2	3381	484	720	668	3328	**467**

Table 5: Comparing VOASC against the baselines in the second instance set containing 362 instances

	$dom/wdeg^{ca.cd}$						FRBA						pick/dom					
	DLC	SC	ASC	MAB	PP	VOA	DLC	SC	ASC	MAB	PP	VOA	DLC	SC	ASC	MAB	PP	VOA
ODLC	**30**	0	0	13	29	27	**27**	0	0	13	**27**	26	**29**	0	0	14	27	26
OSC	0	**24**	**24**	23	7	22	0	**23**	22	20	1	**23**	0	**34**	28	31	1	30
BOTH	**226**	**226**	222	224	**226**	224	**233**	**233**	230	231	**233**	**233**	**218**	**218**	212	214	**218**	**218**
NONE	0	0	3	**5**	1	**5**	0	0	2	**3**	1	2	0	0	0	0	2	**7**

Table 6: Number of solved instances in four categories, where ODLC indicates the instances solved by only DLC, OSC indicates the instances solved by only SC, BOTH indicates the instances solved by both DLC and SC and NONE indicates the instances solved by neither DLC nor SC. The acronyms MAB, PP, and VOA stand for MABAP, PrePeak+, and VOASC, respectively.

instances. We can see that SC, MABAP and ASC perform much better than the other two baselines in solving unsatisfiable instances under all the three variable ordering heuristics. While PrePeak+ outperforms the other baselines on satisfiable instances, it is only marginally better than DLC on unsatisfiable ones. This observation is consistent with PrePeak+'s tendency to leverage DLC's solving power. When using $dom/wdeg^{ca.cd}$, VOASC loses a little in solving unsatisfiable instances in terms of PAR2 score. However, it gets the best performance in all the other comparisons. Its overall performance remains better than that of all the baselines.

Furthermore, we divided the second instance set into four categories and examined how these propagation algorithms perform. The results are presented in Table 6. From the pick/dom columns, we can see that ASC works well in solving the OSC instances, but it cannot solve any ODLC instance, indicating that ASC tends to utilize SC's solving ability. PrePeak+ works well in solving the ODLC instances, but it solves only 1 OSC instance, further confirming that PrePeak+ tends to leverage DLC's solving ability. MABAP and VOASC make a good balance between the utilization of DLC and SC. Both of them successfully solved subsets of instances within ODLC and OSC sets. When comparing VOASC with MABAP, we observe mixed results under the three variable ordering heuristics. However, a notable observation is that VOASC leverages DLC's solving power much more effectively than MABAP, and it solves much more ODLC instances than MABAP. Considering the instances solved by none of DLC and SC, VOASC gets the best performance when using $dom/wdeg^{ca.cd}$ and pick/dom heuristics, and it is the second best one when using FRBA. Thus, VOASC strikes a superior balance between leveraging the solving power of DLC and SC compared to the existing adaptive approaches.

Finally, we examine how the adaptation of VOASC performs in solving COPs. The results are presented in Table 7. The table presents the results of VOASC adaptation with q = 1000, which is the best parameter (among {100, 200, 500, 1000}) for the adaptation. We can see that SC, ASC and MABAP perform poorly in solving the COP instances. They are significantly outperformed by the other three methods. PrePeak+ is the best one among the baselines, which is reasonable because it tends to leverage DLC's solving ability and the θ is dynamically updated when the newly generated CSP becomes more difficult to solve. However, VOASC still gets a better primal gap than PrePeak+ and proves optimality for more instances than PrePeak+. Note that the direct application of the VOASC algorithm to COPs performs worse than the DLC, further confirming that the FE information gathered through the initial preprocessing procedure loses its effectiveness in the newly generated CSPs.

		DLC	SC	ASC	MAB	PP	VOA
FRBA	gap	11.4	50.4	58.2	38.3	9.1	**7.0**
	opt	56	54	51	50	61	**62**
pick/dom	gap	13.9	53.2	60.9	40.1	11.9	**9.1**
	opt	57	52	51	52	57	**60**

Table 7: Comparing VOASC against the baselines on the COP instance set. The gap denotes the primal gap, and opt is the number of instances where the optimality is proved.

In summary, the filtering efficiency is effective in selecting an appropriate one between DLC and SC to propagate the decision of a variable. VOASC with the parameter tuned on the $dom/wdeg^{ca.cd}$ heuristic also works well when the heuristics switch to FRBA and pick/dom, which demonstrates the strong generalization capability of our method. VOASC makes better choices about which propagation algorithm to invoke, and it is more robust than the existing approaches in solving both CSPs and COPs.

6 Conclusion

In this paper, we propose a variable-oriented adaptive singleton consistency for solving constraint satisfaction problems. The filtering efficiency of a local consistency is evaluated by the number of total values filtered per unit time. The approach learns the filtering efficiency of singleton consistency and the default local consistency in propagating the decisions of each variable. In the subsequent search, it determines which local consistency should be used to propagate the variable according to the filtering efficiencies. The experimental results show that the filtering efficiency is effective in selecting an appropriate consistency to propagate the decision of a variable. VOASC exploits the potential filtering ability of singleton consistency more effectively than MABAP and PrePeak+. It is also effective to utilize the solving ability of the default propagation engine. Additionally, the adaptation of VOASC also demonstrates superior performance in solving COP compared to existing approaches. In general, VOASC is more efficient and more robust than the existing propagation strategies and it could be an efficient and effective adaptive propagation scheme for general-purpose constraint solvers.

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, 62532014 and CCF-Huawei Populus Grove Fund.

References

[Audemard *et al.*, 2023] 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.

[Balafrej *et al.*, 2013] Amine Balafrej, Christian Bessière, Remi Coletta, and El Houssine Bouyakhf. Adaptive parameterized consistency. In *International Conference on Principles and Practice of Constraint Programming*, pages 143–158. Springer, 2013.

[Balafrej *et al.*, 2014] Amine Balafrej, Christian Bessière, El Houssine Bouyakhf, and Gilles Trombettoni. Adaptive singleton-based consistencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[Balafrej *et al.*, 2015] Amine Balafrej, Christian Bessière, and Anastasia Paparrizou. Multi-armed bandits for adaptive constraint propagation. In *Proc. IJCAI'15*, pages 290–296. AAAI Press, 2015.

[Bennaceur and Affane, 2001] Hachemi Bennaceur and Mohamed-Salah Affane. Partition-k-ac: an efficient filtering technique combining domain partition and arc consistency. In *Principles and Practice of Constraint Programming—CP 2001: 7th International Conference, CP 2001 Paphos, Cyprus, November 26–December 1, 2001 Proceedings 7*, pages 560–564. Springer, 2001.

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

[Bessière and Régin, 1997] Christian Bessière and Jean-Charles Régin. Arc consistency for general constraint networks: preliminary results. In *IJCAI*, pages 398–404. Morgan Kaufmann, 1997.

[Bessière *et al.*, 2008] Christian Bessière, Kostas Stergiou, and Toby Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6):800–822, 2008.

[Bessière, 2006] Christian Bessière. Constraint propagation. In *Foundations of Artificial Intelligence*, volume 2, pages 29–83. Elsevier, 2006.

[Collavizza *et al.*, 1998] Hélène Collavizza, François Delobel, and Michel Rueher. A note on partial consistencies over continuous domains. In *International Conference on Principles and Practice of Constraint Programming*, pages 147–161. Springer, 1998.

[Debruyne and Bessière, 1997] Romuald Debruyne and Christian Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI'97*, pages 412–417. Morgan Kaufmann, 1997.

[Delecluse and Schaus, 2024] 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.

[Demirovic *et al.*, 2018] 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.

[Gomes *et al.*, 1998] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI'98*, pages 431–437. AAAI, 1998.

[Gomes *et al.*, 2000] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1-2):67–100, 2000.

[Hwang and Mitchell, 2005] Joey Hwang and David G. Mitchell. 2-way vs d-way branching for csp. In *Proc. of CP'05*, pages 343–357. Springer, 2005.

[Karakashian *et al.*, 2010] Shant Karakashian, Robert Woodward, Christopher Reeson, Berthe Choueiry, and Christian Bessière. A first practical algorithm for high levels of relational consistency. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):101–107, 2010.

[Lecoutre *et al.*, 2007] Christophe Lecoutre, Lakhdar Saïs, Sebastien Tabary, and Vincent Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4):147–167, 2007.

[Lecoutre *et al.*, 2013] Christophe Lecoutre, Anastasia Paparrizou, and Kostas Stergiou. Extending str to a higher-order consistency. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1):576–582, 2013.

[Lecoutre, 2011] Christophe Lecoutre. Str2: optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, October 2011.

[Li *et al.*, 2021] Hongbo Li, Minghao Yin, and Zhanshan Li. Failure Based Variable Ordering Heuristics for Solving CSPs. In *Proc. CP'21*, pages 9:1–9:10, 2021.

[Luby *et al.*, 1993] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[Paparrizou and Stergiou, 2012] 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.

[Prud'homme *et al.*, 2017] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documenta-*

tion. TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, AAAI'94, page 362–367. AAAI Press, 1994.

[Stamatatos and Stergiou, 2009] Efstathios Stamatatos and Kostas Stergiou. Learning how to propagate using random probing. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 263–278. Springer, 2009.

[Stergiou, 2008] Kostas Stergiou. Heuristics for dynamically adapting propagation. In *ECAI 2008*, pages 485–489. IOS Press, 2008.

[Stuckey *et al.*, 2014] Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, Jun. 2014.

[Wattez *et al.*, 2019] Hugues Wattez, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. Refining constraint weighting. In *Proc. of ICTAI'19*, pages 71–77. IEEE, 2019.

[Woodward *et al.*, 2014] Robert J. Woodward, Anthony Schneider, Berthe Y. Choueiry, and Christian Bessière. Adaptive parameterized consistency for non-binary csps by counting supports. In *Proc. of CP'14*, pages 755–764, 2014.

[Woodward *et al.*, 2018] Robert J. Woodward, Berthe Y. Choueiry, and Christian Bessière. A reactive strategy for high-level consistency during search. In *IJCAI*, volume 2018, pages 1390–1397, 2018.