symmetry

MDPI

Article

Two Improved Constraint-Solving Algorithms Based on lmaxRPC3^{rm}

Xirui Pan [1,2], Zhuyuan Cheng [1,2] and Yonggang Zhang [1,2,*]

[1] College of Computer Science and Technology, Jilin University, No. 2699 Qianjin Street, Changchun 130012, China; panxr21@mails.jlu.edu.cn (X.P.); chengzy15@mails.jlu.edu.cn (Z.C.)
[2] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, No. 2699 Qianjin Street, Changchun 130012, China
* Correspondence: zhangyg@jlu.edu.cn; Tel.: +86-139-4313-1520

Abstract: The Constraint Satisfaction Problem (CSP) is a significant research area in artificial intelligence, and includes a large number of symmetric or asymmetric structures. A backtracking search combined with constraint propagation is considered to be the best CSP-solving algorithm, and the consistency algorithm is the main algorithm used in the process of constraint propagation, which is the key factor in constraint-solving efficiency. Max-restricted path consistency (maxRPC) is a well-known and efficient consistency algorithm, whereas the lmaxRPC3^{rm} algorithm is a classic lightweight algorithm for maxRPC. In this paper, we leverage the properties of symmetry to devise an improved pruning strategy aimed at efficiently diminishing the problem's search space, thus enhancing the overall solving efficiency. Firstly, we propose the maxRPC3^{sim} algorithm, which abandons the two complex data structures used by lmaxRPC3^{rm}. We can render the algorithm to be more concise and competitive compared to the original algorithm while ensuring that it maintains the same average performance. Secondly, inspired by the RCP3 algorithm, we propose the maxRPC3^{simR} algorithm, which uses the idea of residual support to cut down the redundant operation of the lmaxRPC3^{rm} algorithm. Finally, combining the domain/weighted degree (dom/wdeg) heuristic with the activity-based search (ABS) heuristic, a new variable ordering heuristic, ADW, is proposed. Our heuristic prioritizes the selection of variables with symmetry for pruning, further enhancing the algorithm's pruning capabilities. Experiments were conducted on both random and structural problems separately. The results indicate that our two algorithms generally outperform other algorithms in terms of performance on both problem classes. Moreover, the new heuristic algorithm demonstrates enhanced robustness across different problem types when compared to various existing algorithms.

Keywords: CSP; path consistency; maxRPC; lmaxRPC3^{rm}; variable ordering heuristic; symmetry and asymmetry; artificial intelligence

Citation: Pan, X.; Cheng, Z.; Zhang, Y. Two Improved Constraint-Solving Algorithms Based on lmaxRPC3^{rm}. *Symmetry* **2023**, *15*, 2151. https://doi.org/10.3390/sym15122151

Academic Editors: Lorentz Jäntschi and Sergei D. Odintsov

Received: 12 September 2023
Revised: 17 November 2023
Accepted: 23 November 2023
Published: 3 December 2023

1. Introduction

The Constraint Satisfaction Problem (CSP), as a classic problem of artificial intelligence [1,2], has garnered significant attention from researchers since it was proposed, and it encompasses numerous symmetric and asymmetric structures [3,4]. Related technologies are widely used in the fields of configuration, scheduling, and combinatorial problem solving. A CSP selects a value from a given finite domain for each variable involved in the problem to satisfy the constraints among variables. Generally, CSPs are NP-hard problems [5]. At present, a backtracking search combined with constraint propagation is a classical algorithm used to solve CSPs [6]. This algorithm aims to reduce the symmetrical structure within the problem and eliminate redundant operations. Identifying and locating these symmetric structures are critical areas of focus. Constraint propagation is a common technique that utilizes local consistency to eliminate a subset of inconsistent ethics. There are several tried-and-tested consistency algorithms, including arc consistency (AC) [7], singleton arc consistency (SAC) [8], path consistency (PC) [9], and maxRPC [10].

Enhancing the efficiency of the maxRPC algorithm has been a major focus of research for decades since the development of the algorithm. MaxRPC was initially presented by Debruyne et al. in 1997 [11], as one of the most important local consistencies, which extends RPC, and a maxRPC algorithm maxRPC1 based on AC6 was proposed [11]. Grandoni et al. proposed the maxRPC2 algorithm in 2003, which reduces the space complexity without increasing the time complexity by not storing the found witness [12]. In 2009, Vion and other scholars proposed a backtracking stable and efficient residual data structure, which can achieve the maximum maxRPC domain filtering consistency with the minimum memory consumption in the search process [13]. On the basis of the AC-3^{rm} algorithm [14], two algorithms maxRPC^{rm} and lmaxRPC^{rm} were proposed. Compared with the best maxRPC algorithm at that time, they are competitive and easy to implement. In 2010, Balafoutis et al. [15] proposed maxRPC3 and maxRPC3^{rm}, and their lightweight versions lmaxRPC3 and lmaxRPC3^{rm}. By reducing some redundant constraint inspections, the existing maxRPC algorithm was further improved. Compared with maxRPC, lmaxRPC achieves a lower consistency level, but it is still stronger than AC. The experiment shows that using lmaxRPC in the search process is more cost-effective than maxRPC. lmaxRPC3^{rm} competes with or surpasses the maintenance of AC (MAC) [16] on many problems. In recent years, little work has been carried out on maxRPC. In 2017, a new algorithm was proposed by Li et al. that utilizes light symmetry to approximate maxRPC and enhance the residual support. The algorithm also weakens multi-directional use and narrows the search scope [17].

Theoretical and experimental results demonstrate the strong pruning capability of lmaxRPC3^{rm}. However, the space–time cost does not always meet practical application requirements. Furthermore, designing an effective heuristic function for the pruning strategy is another challenge for the lmaxRPC3^{rm} algorithm. In our work, we introduce two innovative algorithms to overcome these limitations. This paper primarily leverages the concept of symmetry to prune the variables with a symmetric nature, thereby reducing redundant searches and repeated calculations and enhancing overall solving efficiency. To enhance the selection of symmetrical variables for pruning, we developed an improved variable ranking heuristic to enhance our pruning capabilities. Firstly, we propose the maxRPC3^{sim}, a concise maxRPC algorithm, which abandons the complex lastAC and lastPC data structures used in the lmaxRPC3^{rm} algorithm to store AC support and PC support. In this way, some redundant search operations can be avoided, making the algorithm efficient and easy to implement while reducing time and space costs. The experimental results show that maxRPC3^{sim} has certain competitiveness in most problem cases. Secondly, we propose another lmaxRPC algorithm, maxRPC3^{simR}. Inspired by rRPC3 [18], and unlike the lmaxRPC3^{rm} algorithm that supports storing AC and PC separately, maxRPC3^{simR} uses only one residual support to store AC and PC support. In this way, AC support, PC support, and PC witness are stored and searched, and many redundant residual support accesses are reduced, so that the algorithm can effectively reduce redundant consistent support access and reduce the overhead of time and space. Additionally, we investigate variable ordering heuristics to further enhance search performance. Dom/wdeg and ABS [19], as classic variable sorting heuristics, have been widely used since they were proposed. We propose an improved heuristic ADW (activity + dom/wdeg), which combines dom/wdeg with the ABS heuristic. The improved heuristic can not only bring dom/wdeg into full play but also keep the advantages of ABS. Experiments show that the combination of ADW heuristic and different algorithms has good performance, which further shows that ADW has strong robustness.

The paper is structured into multiple sections. In Section 2, we present an overview of recent related work. Section 3 introduces the preliminaries associated with CSPs. Sections 4 and 5 present the improvements made to the lmaxRPC3^{rm} algorithm and the variable ordering heuristic, respectively. The outcomes of experiments conducted on different problem classes are presented in Section 6. Finally, in Section 7, a summary of our work is provided along with suggestions for potential avenues of future research.

2. Related Work

Symmetry in CSP. When solving a problem with a Constraint Satisfaction Problem (CSP) model, it is common to have multiple variables representing multiple instances of the same object. For example, two variables x_i and x_j can represent two history courses that a class must attend in a week. In the case where x_i takes the value Monday at 8 a.m. and x_j takes the value Thursday at 9 a.m., or vice versa, it does not affect whether the instantiation solves the problem or not. In such cases, x_i and x_j are said to be symmetrical. Symmetry detection techniques are employed to prevent the search process from exploring symmetrical subtrees, enabling the pruning of branches in the search tree that contain solutions only if these solutions have a corresponding symmetrical solution elsewhere in the search tree. The issue of symmetry in values was initially addressed by Freuder (1991) [20] through the concept of value interchangeability, which was further extended by Cooper (1997) [21] to include value substitutability. Subsequent studies have extensively investigated various types of symmetries (Benhamou 1994; Gent et al., 2006) [22,23]. Another approach for reducing symmetries when modeling a problem is utilizing set variables instead of integer variables (Gervet and Van Hentenryck 2006) [24,25].

NSCs. Stergiou K. conducted a study to explore the balance between efficiency and ease of implementation in consistency techniques. They focused on neighborhood singleton consistencies (NSCs), which are an extension of the previously proposed neighborhood SAC (NSAC). The study proposes multiple new NSC family members and analyzes them theoretically and experimentally. Theoretical results indicate that the NSCs can provide pruning power ranging between that of RPC and (3,1)-consistency. The study reveals that specific members of the NSC exhibit exceptional competitiveness as versatile propagation techniques for binary constraints, surpassing existing methods in certain problem domains [26].

Interleaved method. The algorithms investigated by Wallace R J employ AC repeatedly under stringent local assumptions to achieve higher levels of consistency. These algorithms establish singleton arc consistency (SAC) in the neighborhood. Most of these algorithms utilize a simple interleaving of AC with the basic SAC procedure. However, the strategy of interleaving weaker and stronger forms of reasoning has not received sufficient attention in its own right. This paper investigates the effects of interleaving and presents novel methods based on this concept. It demonstrates that different problem types exhibit significant variations in their responsiveness to AC interleaving, wherein it proves beneficial in most cases but detrimental for certain algorithms and problem types. Incorporating this feature into SACQ algorithms enhances their consistent and decisive superiority over other SAC algorithms. Additionally, an approach based on AC-4 is considered for interleaving, along with the incorporation of stronger methods than AC [27].

EFCC. Li Z, Yu Z, and their colleagues took a different approach in their research from what had been carried out before. Rather than focusing on strong consistencies, they revisited the effectiveness of weak consistencies. To increase the efficiency of propagation, they proposed algorithms based on bitwise operations. Additionally, they suggested an enhanced version of forward checking consistency (EFCC), which has a stronger pruning ability than FCCs but not as strong as ACs [28].

In recent years, most studies have primarily concentrated on enhancing AC and SAC, while research has seldom been conducted on maxRPC. Nevertheless, maxRPC has demonstrated strong pruning capabilities, as shown in Table 1. Table 1 shows the related work of this paper in order of pruning capabilities (weak to strong) as provided in Reference [29]. Additionally, its lightweight variant, lmaxRPC3^{rm}, has been proven to possess a strong pruning ability, but there is still a considerable amount of compressible space in terms of the space–time cost. Therefore, we aim to leverage its pruning ability as a foundation for optimization and improvement to enhance solution accuracy while simultaneously reducing the space–time cost.

Table 1. Related work in this paper order by algorithms' pruning capability (weak to strong).

No.	Authors	Compatibility Technology	Algorithms	Effectiveness Analyses
1	Alan K. et al. [25]	AC	Arc Consistency	Propagating constraints using AC is the most widely and traditional method.
2	Lecoutre C. et al. (2005) [30]	SAC	Singleton Arc Consistency	SAC is the most promising strong compatibility algorithm capable of replacing AC.
3	Douglass C. et al. [25]	PC	Path Consistency	The idea of PC is to further eliminate illegitimate values by checking the constraint paths.
4	Berlandier P. (1995) [31]	RPC	Restricted Path Consistency	The RPC algorithm determines whether or not to prune by the constraint relationship between the three variables.
5	Debruyne R. et al. (1997) [11]	maxRPC	max-Restricted Path Consistency	The maxRPC algorithm is more compatible than the RPC algorithm and has better pruning capabilities.
6	Debruyne R. et al. (2000) [32]	PIC	Path Inverse Consistency	The PIC algorithm focuses on inverse constraint relationships to better understand constraints between variables.
7	Freuder E C. et al. (1996) [33]	NIC	Neighborhood Inverse Consistency	The NIC algorithm focuses on regional constraints, i.e., constraint relationships within local neighborhoods.
8	Stergiou, K. et al. (2019) [26]	NSCs	Neighborhood Singleton Consistencies	NSC aims to eliminate values that are inconsistent with individual constraints by focusing on them.
9	Li Z, Yu Z. et al. (2021) [27]	EFCC	Enhanced Forward Checking Consistency	EFCC strategically selects variables and values and uses advanced techniques such as domain segmentation.

3. Preliminaries

In this paper, we aim to address the issue of binary constraint satisfaction. A binary CSP is defined as a triple $p = \langle X, D, C \rangle$ where $X = \{x_1, x_2, \ldots, x_n\}$ represents a set of n variables, $D = \{D(x_1), D(x_2), \ldots, D(x_n)\}$ represents a set of domains, $D(x_i)$ denotes a discrete finite value domain of variable x_i, and $C = \{c_1, c_2, c_3, \ldots, c_e\}$ represents a set of e binary constraints. A binary constraint $c_{i,j}$ is made up of two parts: var(c) and rel(c), where $var(c) = \{x_i, x_j\}$ is an ordered set of two variables of X, and rel(c) is a subset of the Cartesian product $D(x_1) \times \ldots \times D(x_m)$ of the finite value domain of these variables. The tuples in this subset represent the value combination of variables that satisfy the constraint. A binary CSP is typically represented as a constrained graph. The nodes in the graph denote variables, while the edges represent constraints between two adjacent nodes.

Definition 1 (arc consistency, AC). *If a value a_i belongs to $D(x_i)$, it will be considered an attribute of AC only if it satisfies all constraints $c_{i,j}$. For each constraint $c_{i,j}$, there must be a value a_j in $D(x_j)$ that satisfies the constraint $c_{i,j}$ when combined with value a_i. This value a_j is called an AC support of value a_i. A variable x_i is considered AC if all values in $D(x_i)$ are AC. A problem is attributed to an AC problem if there are no empty domains in D and all variables in X are AC.*

Definition 2 (path consistency, PC). *In order for a pair of values (a_i, a_j) to be attributed to PC, it must be the case that for any third variable x_k, there exists a value $a_k \in D(x_k)$ such that a_k is an AC support of both values a_i and a_j. When these conditions are met, we say that a_i is a PC support of the value a_i in the variable x_j, and that a_k is a PC witness for the pair of values (a_i, a_j) in the variable x_k.*

Definition 3 (max-restricted path consistency, maxRPC). *To clarify, in a given problem, a value is considered a maxRPC value if it is an AC value and meets the following condition: for every constraint $c_{i,j}$, there exists a value a_j, which is an AC support of the value a_i, making the pair (a_i, a_j) PC. Similarly, a variable is considered maxRPC if all the values in its domain meet the above*

criterion. Finally, a problem is considered a maxRPC problem if there are no empty domains in D(x) and all variables in X are maxRPC.

Definition 4 (lightweight max-restricted path consistency, lmaxRPC)**.** *The lmaxRPC algorithm is a simplified version of the maxRPC algorithm that only focuses on the loss of AC supports and ignores the loss of PC witnesses. In other words, when removing a variable x_j from Q, for each $a_i \in D(x_i)$, the lmaxRPC algorithm only checks if a_i has PC support in $D(x_j)$, without taking into consideration the value in x_j as witnesses of the PCs affected by other values. This results in a faster algorithm than AC but slower than maxRPC. lmaxRPC3 and lmaxRPC3^{rm} are lightweight versions of maxRPC3 and maxRPC3^{rm}, respectively, that further simplify the process of inspecting missing PC witnesses.*

4. New Algorithms for maxRPC

4.1. maxRPC3^{sim}

The first improvement algorithm discards the two data structures LastAC and LastPC of the lmaxRPC3^{rm} algorithm. By selecting symmetric variables for removal at each step, the algorithm's efficiency and ease of implementation are enhanced by effectively reducing the search space and eliminating redundant operations. We named the new algorithm as maxRPC3^{sim}. Algorithm 1 is the main component of maxRPC3^{sim}, and the propagation list Q stores the variables of the filtering domain. Adding variables to list Q may occur during initialization or when a PC support is detected. Algorithm 1 goes through every variable in Q. When variable x_j is removed from Q (line 2), it is necessary to identify its related variable x_i to ensure that they are maxRPC. For every value a_i in $D(x_i)$, the function findPCsup (line 5) is called to verify whether each a_i value has PC support in $D(x_j)$. Below, we provide a detailed explanation of the function findPCsup. If the function returns FALSE, it means that there is no new PC support in $D(x_j)$ for a_i. In such a scenario, Algorithm 1 eliminates a_i from $D(x_i)$ (line 6) and adds x_i to Q (line 7). After checking all a_i values in $D(x_i)$, the algorithm decides whether $D(x_i)$ is empty or not. If it is, the algorithm returns FAILURE.

Algorithm 1 maxRPC3^{sim}

```
1:  while Q ≠ ∅ do
2:      Q ⇐ Q − {x_j}
3:      for each x_i ∈ X s.t. c_{i,j} ∈ C do
4:          for each a_i ∈ D(x_i) do
5:              if ¬ findPCsup(x_i, a_i, x_j) then
6:                  remove a_i
7:                  Q ⇐Q ∪ {x_i}
8:          if D(x_i) = ∅ then
9:              return FAILURE
10: return SUCCESS
```

The function of Algorithm 2 FindPCsup is used to determine if there is PC support between two variables. The function first checks the arc consistency of (a_i, a_j) by calling isTruecon at line 3 for each variable x_j and value a_j in $D(x_j)$. If isTruecon returns TRUE, then it means that (a_i, a_j) satisfies the constraint, indicating that a_j supports a_i for arc consistency. If the result of isTruecon is TRUE, function 1 then checks whether there is a PC witness a_k for (a_i, a_j) by calling the findPCwit function. The findPCwit function is explained below. If there is a new PC witness in $D(x_j)$ for a_i, then the function returns TRUE, meaning that a_j provides a PC support for a_i.

Algorithm 2 Function 1 findPCsup(x_i, a_i, x_j)

```
1: for each x_j ∈ X do
2:     for each a_j ∈ D(x_j) do
3:         if isTruecon(x_i, a_i, x_j, a_j) then
4:             if findPCwit(a_i, a_j) then
5:                 return TRUE
6: return FALSE
```

To verify the existence of a PC witness a_k for the pair (a_i, a_j), we use the function of Algorithm 3 findPCwit. First, we check whether there exists a value a_k on the third variable x_k that satisfies the constraints between (x_i, x_k) and (x_j, x_k) for both (a_i, a_k) and (a_j, a_k). To do this, we use the isTruecon function. If such a value a_k exists, it confirms that there is a PC witness between a_i and a_j.

Algorithm 3 Function 2 findPCwit(a_i, a_j)

```
1: for each x_k ∈ X s.t. c_{i,k} and c_{j,k} ∈ C do
2:     thePCwit ⇐ FALSE
3:     for each x_k ∈ X s.t. c_{i,k} and c_{j,k} ∈ C do
4:         if isTruecon(x_i, a_i, x_k, a_k) and isTruecon(x_j, a_j, x_k, a_k) then
5:             thePCwit ⇐ TRUE
6:             break
7:     if ¬ thePCwit then
8:         return FALSE
9: return TRUE
```

4.2. $maxRPC3^{simR}$

The second improved algorithm is also based on lmaxRPC3^{rm}. In this enhancement, we use a residual support S to store recently discovered compatibility support. Inspired by the concept of the residual support data structure in the rRPC3 algorithm, we name this new algorithm as maxRPC3^{simR}. The residue is a support used to prove that a given value is PC or AC, located or stored during program execution. Unlike lmaxRPC3^{rm}, which stores AC and PC support separately, maxRPC3^{simR} only adopts one residual support S to reduce some redundant access. Furthermore, we do not initialize S in the initialization step. Instead, we judge first and then assign values during the algorithm's propagation process. This targeted assignment strategy effectively minimizes redundant operations.

We omit the detailed introduction of maxRPC3^{simR} in Algorithm 4 because it is the same as Algorithm 1; the only difference is that we use findPCsupR rather than findPCsup.

Algorithm 4 maxRPC3^{simR}

```
1: while Q ≠ ∅ do
2:     Q ⇐ Q − {x_j}
3:     for each x_i ∈ X s.t. c_{i,j} ∈ C do
4:         for each a_i ∈ D(x_i) do
5:             if ¬ findPCsupR(x_i, a_i, x_j) then
6:                 remove a_i
7:                 Q ⇐Q ∪ {x_i}
8:         if D(x_i) = ∅ then
9:             return FAILURE
10: return SUCCESS
```

We also omit the findPCsupR function of Algorithm 5 for maxRPC3^{simR} as it is the same as the findPCsup function with the two differences being that we call it findPCwitR instead

of findPCwit and we store the PC support values in the corresponding data structure S to facilitate the subsequent search.

Algorithm 5 Function 3 findPCsupR(x_i, a_i, x_j)

```
1: for each x_j ∈ X do
2:     for each a_j ∈ D(x_j) do
3:         if isTruecon(x_i, a_i, x_j, a_j) then
4:             if findPCwitR(a_i, a_j) then
5:                 S_(x_i,a_i,x_j) ⇐ a_j
6:                 return TRUE
7: return FALSE
```

Next we explain the findPCwitR function of Algorithm 6 in detail. The findPCwitR function checks each value of x_k in two steps. First, it looks for a value $S_{(x_i,a_i,x_k)}$ and then checks if the pair ($S_{(x_i,a_i,x_k)}$, a_j) is arc consistent. If the pair is not arc consistent, it moves to the second step. In the second step, it checks for a valued $S_{(x_j,a_j,x_k)}$ and verifies whether the pair ($S_{(x_j,a_j,x_k)}$, a_i) is arc consistent. This process is performed at line 2. If a PC witness exists, the above condition returns TRUE and findPCwitR can jump out of this cycle ahead of time. If S does not contain a PC witness for (a_i, a_j) that includes a_k, then there is no such a_k, and then findPCwitR starts a search similar to the findPCwit function and findPCwitR also stores the PC support values into the corresponding data structure S.

Algorithm 6 Function 4 findPCwitR(a_i, x_j)

```
1: for each x_k ∈ X s.t. c_i,k and c_j,k ∈ C do
2:     if (S_(x_i,a_i,x_k) and isTruecon(x_k, S_(x_i,a_i,x_k), x_j, a_j)) or (S_(x_j,a_j,x_k) and
3:      isTruecon(x_k, S_(x_j,a_j,x_k), x_i, a_i)) then
4:         thePCwit ⇐ TRUE
5:         continue
6:     for each a_k ∈ D(X_k) do
7:         if isTruecon(x_i, a_i, x_k, a_k) and isTruecon(x_j, a_j, x_k, a_k) then
8:             thePCwit ⇐ TRUE
9:             S_(x_i,a_i,x_k) ⇐ a_k
10:            S_(x_j,a_j,x_k) ⇐ a_k
11:            break
12:        if ¬ thePCwit then
13:            return FALSE
14: return TRUE
```

Upon analyzing the previously presented pseudocode, it becomes clear that our proposed algorithm as well as the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms all have a time complexity of $O(n^2)$. However, our algorithms, maxRPC3^{sim} and maxRPC3^{simR}, are designed to eliminate redundant structures. maxRPC3^{sim} does not use the original algorithm to store AC- and PC-supported LastAC and LastPC data structures, while maxRPC3^{simR} only retains residual support to store compatibility support. We also adjusted the overall data structure of the two new algorithms. As a result, our algorithm has fewer loops and iterations, which makes it less time-consuming. Moreover, the reduction in the number of redundant accesses leads to less space consumption. Furthermore, our experiments show that our algorithm is more efficient in solving the problem as it takes less time.

5. Heuristics for maxRPC Algorithms

Mainstream constraint solvers speed up the solution by using heuristics to order variables or constraint priorities. At present, many AC algorithms use variable ordering

heuristics to select the first deleted variables from the propagation list to improve search efficiency [34,35]. The heuristic algorithm used by the AC algorithm can also be applied to the maxRPC algorithm. Although heuristics require extra computation, it is insignificant to the overall cost of the algorithm.

The variable ordering heuristic is responsible for assigning priority to the variables involved, the variables with higher priority are assigned first, while those with lower priority are assigned later. A good variable ordering can often accelerate the process of finding the solution. Classic variable ordering heuristics include dom/ddeg, dom/wdeg, ABS (activity-based Search), etc. After the ABS variable ordering heuristic was proposed, the algorithm-solving efficiency greatly improved. In this section, we employ the concept of symmetry to devise a novel variable ordering heuristic known as ADW (ABS+Dom/wdeg). ADW prioritizes variables with symmetry by sorting them, thereby enhancing the algorithm's pruning capabilities. Our experimental results demonstrated significant improvements.

(1) Dom/ddeg heuristic

Dom/ddeg heuristic is a dynamic variable ordering heuristic that combines variable domain size and variable degree. In optimization problems, Dom refers to possible variable values, while ddeg is the number of variable constraints. The dom/ddeg heuristic algorithm works by prioritizing the variable with the smallest ratio of current domain size to dynamic degree in each step. This variable is given the highest priority to be processed next.

(2) Dom/wdeg heuristic

The Dom/wdeg heuristic is a variable ordering strategy that integrates domain size and weighting, enhancing the professionalism and academic rigor of this research paper in accordance with the requirements of the journal Nature. Wdeg only considers constraints that contain two uninstantiated variables. There is a certain connection between wdeg and ddeg. When the weight counter of all variables is set to 1, wdeg becomes ddeg.

Wdeg maintains a counter for each constraint that indicates the number of constraint failures, i.e., the number of times a value from the domain of a variable has been removed throughout the propagation process. The wdeg expression of variable x_i is

$$\alpha_{wdeg}(x_i) = \sum_{c \in C} weight[c] s.t. x_i \in vars(c) x_i \wedge |(FutVars(c))| > 1 \quad (1)$$

where the expression "$FutVar(c)$" refers to the set of uninitialized variables that are involved in the constraint "c". During initialization, the weight counter of all variables is set to 1. If the constraint does not meet the requirement, its weight is incremented by 1. It is worth noting that once restarting, to keep learning all the time, the weight will not reset to one. In each step, the dom/wdeg heuristic selects the variable with the minimum ratio $\frac{|D(x_i)|}{\alpha_{wdeg}(x_i)}$ as the highest priority.

(3) ABS heuristic

The ABS heuristic algorithm is a dynamic variable ordering heuristic algorithm, which uses the activities of variables in the process of propagation to guide the search. Inspired by the VSIDS algorithm, ABS gradually forgets the oldest statistics based on a sum of attenuation. Each variable x_i is associated with a counter to measure how often the domain of x_i is reduced in the propagation process. This measure is initialized by probing the search space, which does not add any space requirements [36].

The variable activity value is initialized to 0 and given a CSP $p = \langle X, D, C \rangle$. After an assignment decision, a constraint propagation algorithm F is applied; F produces a new field $D' \subseteq D$, continuing to perform compatibly. To define the subset of variables X' that are affected, apply F to P. The formulas for the definitions are as follows: Formula (2) represents the domain change of the affected variable, while Formula (3) represents the domain change of the unaffected variable.

$$\forall x \in X' : D'(x) \subset D(x) \quad (2)$$

$$\forall x \in X/X' : D'(x) = D(x) \tag{3}$$

The activity of variable x, denoted as A(x_i), updates at each search tree node after constraint propagation. The update rules are shown in Formulas (4) and (5). Formula (4) corresponds to the unpruned variable, and Formula (5) corresponds to the pruned variable.

The activity of x_i (represented by A(x_i)) is updated as follows:

$$\forall x \in Xs.t.|D(x)| > 1 : A(x) = A(x) * \gamma \tag{4}$$

$$\forall x \in X' : A(x) = A(x) + 1 \tag{5}$$

where X' is the set of the affected variables and γ is the attenuation constant acceptable $0 \leq \gamma \leq 1$. Attenuation only affects free variables, thus slowing down the activity of deleting previously marked variables. In each step, the ABS heuristic selects the variable with the highest ratio of A(x_i) to D(x_i) as the highest priority.

(4) ADW heuristic

dom/wdeg is the variable that picks the smallest value for dom(x)/wdeg(x) and ABS is the variable that picks the largest value for A(x)/D(x). By leveraging the strengths of both dom/wdeg and ABS, we propose a novel variable ordering heuristic called ADW (activity + dom/wdeg), which is expected to achieve better results in more cases. ADW considers the domain size, variable weighting, and activity. ADW maintains a counter for each constraint similar to wdeg, and a counter for each variable similar to ABS. Our proposed ADW heuristic selects the variable that is most likely to cause failure and is most active. In each step, the heuristic sets the variable x_i with the highest ratio of (A(x_i) $\times$ wdeg (x_i))/dom(x_i) as the maximum priority. This improved heuristic not only fully utilizes the potential of dom/wdeg but also retains the advantages of ABS. The experimental results demonstrate that the new heuristic exhibits excellent performance and strong robustness.

6. Experiments and Analysis

All experiments were carried out in the environment of the Ubuntu 17.04 system, Intel i7-6700 processor, and 8 g RAM. We evaluated our proposed algorithms and heuristics on structured and stochastic CSP problems selected from C. Lecoutre's XCSP repository. A lot of the details of the problems can be found at https://www.movingai.com/benchmarks/ (accessed on 6 April 2023).

In this paper, the branching scheme of our algorithms is two-way (also known as binary). The two-way branching technique involves creating two branches: one where a variable is assigned a value and propagation is triggered, and another where the variable is left unassigned and propagation is triggered. In another branch, a value is removed from the variable's domain, triggering propagation again. If the left branch propagation fails, the right branch propagation succeeds, and the next variable can be selected. Algorithm backtracking occurs when both branches fail to propagate [18].

In all the tables below, the number of instances tested in the problem is shown in parentheses after the problem. The solution result is expressed as the average solution time of multiple instances. The best-performing algorithms are highlighted in bold and underlined. Timeout is set to 600 s, indicated by the symbol "**".

6.1. Performance Comparison of Improved Algorithms

In this section, all algorithms use dom/wdeg heuristics to order variables. For example, for Algorithm 1 maxRPC3^{sim} described in Section 4, when a value a_i is deleted due to a constraint, the heuristic changes the weight of the constraint $c_{i,j}$.

Tables 2 and 3 present the experimental outcomes of algorithm enhancement. The algorithms tested include maxRPC3^{sim}, maxRPC3^{simR}, lmaxRPC3^{rm}, and maxRPC3^{rm}. The "time" column represents the duration in seconds taken to solve the problems, while the

"node" column indicates the number of nodes. Table 2 displays the test results on 23 problems, which altogether amount to 244 structured problems. On the other hand, Table 3 illustrates the test results of 8 problem categories, summing 71 random problem instances.

From Table 2, we can see that among the 23 structured problems, a total of 10 problems took maxRPC3^{sim} the shortest time on average, maxRPC3^{simR} took the shortest time for 5 problems, and lmaxRPC3^{rm} and maxRPC3^{rm} both took the shortest time for 3 problems. However, due to the limitation of the initialization cost of the data structure, all algorithms had a high cost in large instances. Specifically, all algorithms timed out on the qcp-25 and qwh-25 (4) problem class; that is, the data items in the table with the "**" symbol in the table. In particular, lmaxRPC3^{rm} timed out on the qcp-20 problem class. In contrast, the new algorithms maxRPC3^{sim} and maxRPC3^{simR} can be used without timeout, indicating that our new algorithms perform slightly better in large instance problems. Overall, the average performance of algorithms maxRPC3^{sim} and maxRPC3^{simR} is better than that of algorithm lmaxRPC3^{rm} on 76.2% and 66.7% of problem classes, respectively.

Table 2. Mean standalone performance in structured problems.

Problem Class/Algorithms	**maxRPC3^{sim}**		**maxRPC3^{simR}**		**lmaxRPC3^{rm}**		**maxRPC3^{rm}**	
	Time	**Node**	**Time**	**Node**	**Time**	**Node**	**Time**	**Node**
qcp-10 (14)	**0.112**	99	0.135	99	0.291	99	0.496	103
qcp-15 (15)	123.388	17,076	**107.425**	21,044	147.175	12,752	142.999	7021
qcp-20 (4)	588.884	21,436	545.688	34710	**	**	**471.537**	7303
qcp-25 (4)	**	**	**	**	**	**	**	**
qwh-10 (10)	**0.066**	107	0.0814	107	0.143	106	0.261	108
qwh-15 (10)	2.948	589	**2.192**	589	4.45	589	5.252	419
qwh-20 (10)	348.211	15,348	**305.712**	20,604	372.424	9390	454.338	5765
qwh-25 (4)	**	**	**	**	**	**	**	**
rlfapgraphs (14)	**0.115**	548	0.192	548	0.18	548	0.122	548
rlfapscens (11)	**0.118**	559	0.199	559	0.186	559	0.132	559
rlfapGraphsMod (12)	**0.137**	751	0.234	751	0.242	751	0.144	751
rlfapScens11 (12)	**0.134**	680	0.232	673	0.249	680	0.165	680
subs (9)	0.0026	35	0.0032	35	0.00189	35	**0.00181**	35
coloring (19)	12.129	118,686	**11.401**	118,686	24.063	118,686	34.343	115,067
hos (14)	**0.08**	1248	0.13	1248	0.147	1248	0.086	1248
myciel (16)	0.00333	97	0.00354	97	**0.00215**	97	0.00268	97
bqwh-15-106 (10)	**0.17**	230	0.196	230	0.478	267	0.975	224
bqwh-18-141_glb (10)	0.0026	106	0.00711	141	0.0024	141	**0.00181**	106
jobShop-e0ddr1 (10)	0.00523	50	0.00607	50	**0.00405**	50	0.00458	50
queens (14)	**0.011**	70	0.0188	70	0.0226	70	0.014	70
largequeens (2)	**0.275**	350	0.555	350	0.963	350	0.457	350
queensKnights (10)	0.00183	18	0.00151	18	**0.00097**	18	0.00106	18
queenAttacking (10)	0.00705	66	**0.00669**	66	0.014	66	0.00836	66

CPU times (in secs) and nodes for the two different branching schemes. The best-performing algorithms are highlighted in bold and underlined. Timeout is set to 600 s, indicated by the symbol "**".

maxRPC3^{sim} has approximately three times the number of problems with the shortest time spent on them compared to maxRPC3^{rm} and lmaxRPC3^{rm}. maxRPC3^{sim} algorithms takes the shortest time on average in the qcp-10, qwh-10, rlfapgraphs, rlfapscens, rlfapGraphsMod, rlfapScens11, hos, bqwh-15-106, queens, and largequeens problems. Among them, the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms spend on average more than twice and four times as long as maxRPC3^{sim} on the qcp-10 problem, respectively. maxRPC3^{sim} has almost absolute dominance in the rlfap problem, and has the shortest time spent in rlfapgraphs, the rlfapscens, rlfapGraphsMod, and rlfapScens11, which are the four best problems, while lmaxRPC3^{rm} takes about twice as long as maxRPC3^{sim} in these problems. In the bqwh-15-106 problem, lmaxRPC3^{rm} and maxRPC3^{rm} take about two and

five times as long as maxRPC3^{sim}, respectively. lmaxRPC3^{rm} takes about two times as long as maxRPC3^{sim} on average in the Queens problem, and in the largequeens problem, lmaxRPC3^{rm} and maxRPC3^{rm} take about three and two times as long as maxRPC3^{sim}.

In the problems qcp-15, qwh-15, qwh-20, coloring, and queenAttacking, maxRPC3^{sim} takes the shortest time except for maxRPC3^{simR}. In the qwh-15 problem, maxRPC3^{rm} and lmaxRPC3^{rm} take about twice as long as maxRPC3^{sim}. In the coloring problem, maxRPC3^{rm} and lmaxRPC3^{rm} take about three and two times as long as maxRPC3^{simR}. In the queenAttacking problem, the lmaxRPC3^{rm} algorithm takes about twice as long as maxRPC3^{sim}. Overall, the maxRPC3^{sim} algorithm has the best results in most problems.

The number of problems where maxRPC3^{simR} works best compared to maxRPC3^{rm} and lmaxRPC3^{rm} is close to twice the number of problems of maxRPC3^{rm} and lmaxRPC3^{rm}. The maxRPC3^{simR} algorithm takes the least amount of time on average in the qcp-15, qwh-15, qwh-20, coloring, and queenAttacking problems. In the qwh-15 problem, the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms take about twice as long as the maxRPC3^{simR} algorithm on average. In qwh-20, the maxRPC3^{rm} algorithm takes about 1.5 times as long as maxRPC3^{simR}. In the coloring problem, the maxRPC3^{rm} algorithm and the lmaxRPC3^{rm} algorithm take on average more than two and three times as long as maxRPC3^{simR}, respectively.

The maxRPC3^{simR} algorithm is the most effective algorithm except for the maxRPC3^{sim} algorithm in the qcp-10, qwh-10, and bqwh-15-106 problems. In the qcp-10 problem, the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms take about two and four times as long as the maxRPC3^{simR} algorithm. In problem qwh-10, the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms take on average nearly twice and three times as long as maxRPC3^{simR}, respectively. On the bqwh-15-106 problem, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm take approximately two and five times as long as the maxRPC3^{simR} algorithm. Moreover, we can also see that in the qcp-10, subs, myciel, bqwh-18-141_glb, jobShop-e0ddr1, and queensKnights problems, lmaxRPC3^{rm} and maxRPC3^{rm} perform better. We speculate that it is because our algorithm is modified based on the symmetry property, and these problems have fewer symmetric structures and are more dependent on the redundant data structures we abandoned. However, the solving time is actually quite similar, and our algorithm saves more storage space because of the reduction in redundant operations.

In the rlfap problem, the maxRPC3^{sim} algorithm is absolutely dominant, the maxRPC3^{simR} algorithm is not dominant, and the number of nodes is close. The maxRPC3^{sim} algorithm has the best result in the Hos problem, and in qcp-15, qwh-20, and coloring, the maxRPC3^{simR} algorithm has outstanding performance. Side by side, the maxRPC3^{sim} algorithm has an advantage in solving smaller-scale problems, while maxRPC3^{simR} has an advantage in many large-scale problems. It can be seen that out of the seven problems with an average time of more than 1 s, the maxRPC3^{simR} algorithm has the least amount of time spent in four problems, while there are two problems where all four algorithms time out, and one problem where the best performer is the maxRPC3^{rm} algorithm. It shows that our proposed maxRPC3^{simR} algorithm has good pruning ability, and at the same time, saves time with simplicity and efficiency.

Table 3 shows the average performance of the algorithms for random instances from the geometric, Ehi-85, composed, random, and frb problems. The average performance of algorithms maxRPC3^{sim} and maxRPC3^{simR} are both better than that of lmaxRPC3^{rm}, except for the Ehi-85 problem. That is, the new algorithms are better than lmaxRPC3^{rm} in 85% of problem classes. Moreover, the maxRPC3^{rm} algorithm times out in the rand-2-23 and rand-2-24 problems, while none of the other three algorithms time out. Thus, the algorithm maxRPC3^{simR} works best on the randomized problem.

Table 3. Mean standalone performance in random problems.

Problem Class/Algorithms	maxRPC3^{sim}		maxRPC3^{simR}		lmaxRPC3^{rm}		maxRPC3^{rm}	
	Time	**Node**	**Time**	**Node**	**Time**	**Node**	**Time**	**Node**
geom (15)	121.873	10,811	**104.604**	11,205	144.821	7415	145.832	4478
ehi-85 (10)	0.0819	0	0.0975	0	0.037	0	**0.0272**	0
composed-25-1-2 (10)	**0.0131**	104.604	0.021	0	0.0289	0	0.0882	4478
composed-25-10-20 (10)	**0.189**	247	0.267	247	0.547	247	0.959	276
rand-2-23 (3)	515.318	83,494	**497.493**	96,784	538.527	46,365	**	52,126
rand-2-24 (3)	465.905	66,527	**434.952**	74,865	557.8	42,295	**	25,947
frb30-15 (10)	5.142	1255	**4.392**	1179	14.325	1690	17.172	1148
frb35-17 (10)	47.966	8902	**30.74**	6348	82.116	8078	119.022	6722

CPU times (in secs) and nodes for the two different branching schemes. The best-performing algorithms are highlighted in bold and underlined. Timeout is set to 600 s, indicated by the symbol "**".

Comparing the maxRPC3^{sim}, lmaxRPC3^{rm}, and maxRPC3^{rm} algorithms, the number of best-performing problems in the maxRPC3^{sim} algorithm is two, while there are none for lmaxRPC3^{rm} and only one for maxRPC3^{rm}. In the composed-25-1-2 and composed-25-10-20 problems, maxRPC3^{sim} took the least time. In the composed-25-1-2 problem, the lmaxRPC3^{rm} and maxRPC3^{rm} algorithms took twice and nearly seven times as long as the maxRPC3^{sim} algorithm, respectively. In the composed-25-10-20 problem, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm took three and five times longer than the maxRPC3^{sim} algorithm, respectively.

The maxRPC3^{sim} algorithm is the most effective algorithm except for the maxRPC3^{simR} algorithm in the geom, rand-2-23, rand-2-24, frb30-15, and frb35-17 problems. In particular, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm took about three times as long as the maxRPC3^{rm} algorithm in problem frb30-15. In problem frb35-17, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm took about twice as long as the maxRPC3^{sim} algorithm and about three times as long as the maxRPC3^{sim} algorithm, respectively. Algorithm maxRPC3^{sim} works much better than the maxRPC3^{rm} algorithm and the lmaxRPC3^{rm} algorithm.

Comparing the maxRPC3^{simR}, lmaxRPC3^{rm}, and maxRPC3^{rm} algorithms, the maxRPC3^{simR} algorithm works best on all five problems, whereas the maxRPC3^{rm} algorithm works well on only one problem and the lmaxRPC3^{rm} algorithm does not. The maxRPC3^{simR} algorithm works best in the geom, rand-2-23, rand-2-24, frb30-15, and frb35-17 problems. In the frb30-15 problem, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm took more than three and four times as long as the maxRPC3^{simR} algorithm, respectively. In the frb35-17 algorithm, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm took more than twice and three times as long as the maxRPC3^{simR} algorithm, respectively.

The maxRPC3^{simR} algorithm is the most effective algorithm except for the maxRPC3^{sim} algorithm in both problem classes of the composed problem. In particular, the maxRPC3^{rm} algorithm takes approximately four times longer than the maxRPC3^{simR} algorithm in the composed-25-1-2 problem, and in the composed-25-10-20 algorithm, the lmaxRPC3^{rm} algorithm and the maxRPC3^{rm} algorithm take about two and four times longer than the maxRPC3^{simR} algorithm, respectively. Overall, the maxRPC3^{simR} algorithm has the best results in stochastic problems. We can see from the time that is similar to the structured problems, the maxRPC3^{sim} algorithm works well for small-scale problems, and the maxRPC3^{simR} algorithm works well for larger-scale problems.

In recent years, very little work has been conducted on the maxRPC algorithm. In Reference [17], only 15 kinds of problems were tested, while we carried out experiments on 31 kinds of structured problems and random problems. In addition, Reference [17] showed that the best average performance improved by 65%, while the average performance of

algorithm maxRPC3^{sim} improved by 82.5% on problem bqwh-15-106, and the average performance of algorithm maxRPC3^{simR} improved by 74.4% on problem frb30-15.

6.2. Performance Comparison of Improved Algorithms with Heuristics.

Tables 4 and 5 compare the performance of algorithms maxRPC3^{sim}, maxRPC3^{simR}, lmaxRPC3^{rm}, and AC3 using different heuristics on nine problem classes, which are randomly selected examples of structured and randomized problems. We include results from dom/ddeg, dom/wdeg, ABS, and our proposed ADW heuristic.

Table 4. Algorithms maxRPC3^{sim} and maxRPC3^{simR} use different heuristic results.

Problem Class/Algorithms	**maxRPC3^{sim}**				**maxRPC3^{simR}**			
	dom/ddeg	**dom/wdeg**	**ABS**	**ADW**	**dom/ddeg**	**dom/wdeg**	**ABS**	**ADW**
qcp-10 (12)	47.056	0.122	5.145	**0.072**	96.339	0.15	5.437	**0.079**
qcp-20 (2)	600	577.768	600	**484.339**	600	491.376	600	**433.382**
qwh-15 (10)	5.458	**0.066**	5.578	2.181	4.034	**0.0814**	3.968	1.551
queens (14)	0.303	**0.011**	0.298	0.297	0.019	0.0188	**0.012**	0.013
rlfapGraphs (11)	0.129	**0.079**	0.082	0.116	0.128	0.13	**0.052**	0.099
rlfapScens (11)	0.128	**0.118**	0.163	0.178	0.201	0.199	**0.079**	0.137
coloring (11)	18.833	**0.023**	61.832	0.025	23.644	**0.022**	62.883	0.025
rand-2-24 (3)	**346.82**	465.905	549.067	569.309	**319.769**	434.952	522.267	536.487
frb30-15 (10)	5.498	**5.142**	13.777	6.569	5.209	**4.392**	14.246	6.55
geom (9)	**15.406**	62.514	50.028	32.084	**16.065**	46.909	42.627	28.967
composed-25-10-20 (4)	600	0.216	0.062	**0.056**	600	0.277	0.07	**0.063**
total time	1639.631	1111.964	1286.032	**1095.226**	1665.408	**978.5072**	1251.641	1007.353

CPU times (in secs) and nodes for the two different branching schemes. The best-performing algorithms are highlighted in bold and underlined.

From Table 4 we can conclude that the ADW heuristic performs best in the structured problems qcp-10 and qcp-20, and the stochastic problem composed-25-10-20 under the execution of the maxRPC3^{sim} algorithm. In the qcp-10 problem, the algorithm takes hundreds of times longer to apply dom/ddeg than to apply the ADW heuristic, dom/wdeg takes close to twice as long as ADW, and ABS takes tens of times longer than ADW. In the qcp-20 problem, only the dom/wdeg heuristic and the ADW heuristic did not time out, whereas the other two heuristics did. In the COMPOSED problem, dom/wdeg timed out, while dom/wdeg took about four times as long as ADW. Dom/wdeg worked best in the qwh-15, queens, rlfapgraphs, rlfapScens, coloring, and frb30-15 problems, and dom/ddeg worked best in the rand-2-24 and geom problems, which shows that applying the ADW heuristic to the maxRPC3^{sim} algorithm has a good bootstrap ordering of the algorithmic variables, and works better than applying the dom/wdeg heuristic in some problems, and better than applying the dom/ddeg and ABS heuristics in many problems.

In the maxRPC3^{simR} algorithm, the application of the ADW heuristic works best in the qcp-10, qcp-20, and composed-25-10-20 problems, the application of the dom/wdeg heuristic works best in the qwh-15, coloring, and frb30-15 problems, the application of the ABS heuristic works best in the queens, rlfapGraphs, and rlfapScens problems, and the Dom/Wdeg heuristic yields the best results in stochastic problems Rand-2-24 and Geom. Taken together, applying dom/wdeg, ABS and ADW can have the best results in a larger number of problems, while applying dom/ddeg has the best results in the stochastic problems rand and geom. Therefore, the application of the improved variable ordering heuristic ADW also works well for variable ordering of the maxRPC3^{simR} algorithm and is more efficient than using other heuristics in many problems.

Table 5. Algorithms lmaxRPC3^{rm} and AC3 use different heuristic results.

Problem Class/Algorithms	**lmaxRPC3^{rm}**				**AC3**			
	dom/ddeg	**dom/wdeg**	**ABS**	**ADW**	**dom/ddeg**	**dom/wdeg**	**ABS**	**ADW**
qcp-10 (12)	68.809	0.328	19.138	**0.258**	22.38	0.014	0.566	**0.004**
qcp-20 (2)	**	**	**	**	600	**168.203**	600	308.703
qwh-15 (10)	12.861	**0.143**	15.266	5.269	0.744	**0.0115**	0.28	0.088
queens (14)	0.018	0.0226	0.015	**0.005**	58.43	0.00136	**0.0008**	0.005
rlfapGraphs (11)	0.173	0.128	0.094	**0.086**	0.007	0.007	**0.004**	**0.004**
rlfapScens (11)	0.199	0.186	**0.133**	0.191	0.008	0.00925	**0.005**	**0.005**
coloring (11)	54.614	**0.037**	85.92	0.197	5.559	**0.005**	56.732	0.007
rand-2-24 (3)	580.35	**529.666**	600	600	**15.151**	29.139	21.562	21.866
frb30-15 (10)	22.315	**14.325**	80.083	25.504	0.203	0.209	0.331	**0.154**
geom (9)	**58.91**	96.637	90.744	147.326	0.66	1.857	0.824	**0.603**
composed-25-10-20 (4)	600	0.652	0.327	**0.281**	600	0.029	0.012	**0.005**
total time	1398.249	**642.1246**	891.72	779.117	703.142	31.28211	680.3168	**22.732**

CPU times (in secs) and nodes for the two different branching schemes. The best-performing algorithms are highlighted in bold and underlined. Timeout is set to 600 s, indicated by the symbol "**".

Table 5 shows the results of applying the four heuristics on the lmaxRPC3^{rm} algorithm and the AC3 algorithm. When using the lmaxRPC3^{rm} algorithm, applying the ADW heuristic worked best in the qcp-10, queens, rlfapGraphs, and composed-25-10-20 problems, the dom/wdeg heuristic was best when applied to the qwh-15, coloring, rand-2-24, and frb30-15 problems, the ABS heuristic was best when applied to the rlfapScens problem, and the dom/wdeg application was best when applied to the geom stochastic problem. Therefore applying the ADW heuristic and dom/wdeg heuristic works best.

When utilizing the AC3 algorithm, the application of the ADW heuristic yielded optimal results in solving the qcp-10, rlfapGraphs, rlfapScens, frb30-15, geom, and composed-25-10-20 problems. Conversely, employing the dom/wdeg heuristic proved to be most effective in addressing the rand-2-24 problem. Furthermore, for resolving the qcp-20, qwh-15, and coloring problems, implementing the dom/wdeg heuristic produced superior outcomes. Notably, when dealing with the queens as well as the rlfapGraphs and rlfapScens problems that required equal time allocation between ADW and ABS heuristics, applying the ABS heuristic demonstrated better performance. In summary, ADW exhibited optimal efficacy when combined with the AC3 algorithm, while both dom/wdeg and ADW heuristics showcased promising results when employed alongside lmaxRPC3^{rm}.

The overall results show that the total time of algorithms maxRPC3^{sim} and AC3 using ADW heuristics is the best, and algorithms maxRPC3^{simR} and lmaxRPC3^{rm} using ADW heuristics are in second place, but compared with other heuristics, ADW is close to the first heuristics, which means that ADW heuristics have strong robustness. This advantage makes ADW very useful in the case of uncertainty and many kinds of problems. From the industrial point of view, it has great practical significance in practical problems.

7. Conclusions

Symmetrical structures are ubiquitous in the CSP. It is crucial to identify and characterize these structures to apply effective pruning strategies, thus reducing redundant searches and improving solving efficiency. In light of this, we propose a simplified and more efficient pruning strategy based on lmaxRPC3^{rm}, which minimizes symmetry and eliminates redundant operations with less space–time cost. The paper presents the following main contributions: Firstly, we focus on the recent classic algorithm, lmaxRPC3^{rm}, and enhance its performance through simplification and optimization of the data structure. This leads to the development of two new maxRPC algorithms: maxRPC3^{sim} and maxRPC3^{simR}. These

algorithms demonstrate improved performance compared to the original lmaxRPC3^{rm}. Additionally, we introduce an enhanced variable ordering heuristic called ADW. This heuristic aids in the selection of more appropriate variables, optimizing the pruning strategy.

The algorithm maxRPC3^{sim} does not use residual records, discards the LastAC and LastPC data structures in lmaxRPC3^{rm}, and starts each search for support from the minimum value. The maxRPC3^{simR} changes the way the two data structures are used to store residual support in the lmaxRPC3^{rm} algorithm, respectively, and uses one data structure to store support. Thus the two improved algorithms avoid some redundant compatibility support and PC witness access, making the algorithms more efficient. Experimental results show that the new algorithm outperforms the original maxRPC algorithm in a large number of arithmetic cases. Specifically, the maxRPC3^{simR} algorithm shows superior performance in stochastic problems, while the maxRPC3^{sim} algorithm shows superior performance in structured problems. Additionally, from the experimental data, we found that the maxRPC3^{simR} algorithm generally works better for problems with larger problem sizes, while the maxRPC3^{sim} algorithm works better for problems with smaller problem sizes. In the timeout case, the maxRPC3^{simR} algorithm has the least number of timeout problems, and the maxRPC3^{sim} algorithm is second only to the maxRPC3^{simR} algorithm. Our two proposed algorithms demonstrate outstanding performance in the majority of stochastic and structural problems.

In addition, incorporating variable ranking heuristics into algorithms can serve as a valuable guide for variable selection during the algorithmic process. We combine the classic efficient dom/wdeg heuristic and ABS heuristic to generate the improved heuristic ADW. We apply several classic heuristics and our ADW heuristic in our proposed algorithms as well as in the comparison algorithm, and evaluate their effectiveness through experimental analysis. The experimental results show that the ADW heuristic's overall performance is in the top two, which is highly significant for practical problem solving. This implies that we can use ADW in the case of unknown or complex problem types without artificial prediction.

Although our algorithm performs well in most problems, there are still some limitations in the research. For example, the proposed pruning strategy is very effective in eliminating symmetric structures and reducing redundant searches, but it is not necessarily suitable for all types of constraint satisfaction problems. In addition, there is still room for improving algorithm optimization and combining technologies to enhance problem-solving efficiency. Additionally, the performance of the algorithm seems to be affected by the problem's size, and it is observed that maxRPC3^{simR} performs better on larger problems, while maxRPC3^{sim} is more effective on smaller problems, which indicates that the scalability of the algorithm is limited.

In future work, we will consider the study of a bitwise version of the maxRPC3^{sim} and maxRPC3^{simR} algorithms. The process of finding AC support, PC support, and PC witness is performed frequently and consumes a substantial amount of time in the algorithm. To address this, we aim to leverage bit operations to accelerate the process of identifying and storing AC support, PC support, and PC witness. This enhancement is expected to significantly improve the efficiency of lmaxRPC3^{rm}. Additionally, we intend to explore the integration of machine learning techniques into our algorithm [37]. The objective is to train constraint networks capable of solving more complex and diverse problems, thereby enhancing the algorithm's generalization ability. Furthermore, to better identify and eliminate symmetric structures in CSPs, we are investigating the potential of combining the symmetry-breaking group-theoretic representation [38].

Author Contributions: The conceptualization and design of the study were contributed to by all authors, with X.P. and Z.C. playing important roles. X.P. and Z.C. conducted the experiments and X.P. drafted the first draft. All authors critically reviewed and provided constructive feedback on earlier versions of the manuscript. Y.Z. secured the necessary funding for this research effort. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China (62076108, 61872159), and the Natural Science Foundation of Jilin Province, China (20210101172JC).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to express their gratitude to the editors and reviewers for handling and reviewing our paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ABS	activity-based search heuristic
AC	arc consistency
CSP	Constraint Satisfaction Problem
dom/wdeg	domain/weighted degree heuristic
RPC	restricted path consistency
lmaxRPC3^{rm}	classic lightweight algorithm for maxRPC
maxRPC	Max-restricted path consistency
PC	path consistency
SAC	singleton arc consistency
PIC	path inverse consistency
NIC	neighborhood inverse consistency
NSCs	neighborhood singleton consistencies
EFCC	enhanced forward checking consistency

References

1. Tang, Y.; Pan, Z.; Pedrycz, W.; Ren, F.; Song, X. Viewpoint-based kernel fuzzy clustering with weight information granules. *IEEE Trans. Emerg. Top. Comput. Intell.* **2023**, *7*, 342–356. [CrossRef]
2. Tang, Y.; Huang, J.; Pedrycz, W.; Li, B.; Ren, F. A fuzzy cluster validity index induced by triple center relation. *IEEE Trans. Cybern.* **2023**, *53*, 5024–5036. [CrossRef] [PubMed]
3. Zhou, W.; Zhou, P.; Zheng, Y.; Xie, Z. A Heuristic Integrated Scheduling Algorithm via Processing Characteristics of Various Machines. *Symmetry* **2022**, *14*, 2150. [CrossRef]
4. Han, B.; Hu, M.; Wang, X.; Ren, F. A Triple-Structure Network Model Based upon MobileNet V1 and Multi-Loss Function for Facial Expression Recognition. *Symmetry* **2022**, *14*, 2055. [CrossRef]
5. Freuder, E.C.; Mackworth, A.K. Constraint satisfaction: An emerging paradigm. *Found. Artif. Intell.* **2006**, *2*, 13–27.
6. Dlask, T.; Werner, T. Bounding Linear Programs by Constraint Propagation: Application to Max-SAT. In Proceedings of the CP: International Conference on Principles and Practice of Constraint Programming, Louvain-la-Neuve, Belgium, 7–11 September 2020; pp. 177–193.
7. Yap, R.H.C.; Xia, W.; Wang, R. Generalized arc consistency algorithms for table constraints: A summary of algorithmic ideas. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 13590–13597. [CrossRef]
8. Carbonnel, C.; Cohen, D.A.; Cooper, M.C. On singleton arc consistency for CSPs defined by monotone patterns. *Algorithmica* **2019**, *81*, 1699–1727. [CrossRef] [PubMed]
9. Kong, S.; Li, S.; Michael, S. Exploring directional path-consistency for solving constraint networks. *Comput. J.* **2018**, *61*, 1338–1350. [CrossRef]
10. Guo, J.; Li, Z.; Li, H. Partial Max-restricted Path Consistency. In Proceedings of the ICTAI: International Conference on Tools with Artificial Intelligence, Athens, Greece, 7–9 November 2012; pp. 186–190.
11. Debruyne, R.; Bessiere, C. From restricted path consistency to max-restricted path consistency. In Proceedings of the CP: International Conference on Principles and Practice of Constraint Programming, Linz, Austria, 29 October–1 November 1997; pp. 312–326.
12. Grandoni, F.; Italiano, G. Improved algorithms for max-restricted path consistency. In Proceedings of the CP: International Conference on Principles and Practice of Constraint Programming, Kinsale, Ireland, 29 September–3 October 2003; pp. 858–862.
13. Vion, J.; Debruyne, R. Light algorithms for maintaining Max-RPC during search. In Proceedings of the SARA-2009 Eighth Symposium on Abstraction, Reformulation, and Approximation, Lake Arrowhead, CA, USA, 8–10 August 2009; pp. 167–174.
14. Lecoutre, C.; Hemery, F. A study of residual supports in arc consistency. In Proceedings of the IJCAI: International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 125–130.

15. Balafoutis, T.; Paparrizou, A.; Stergiou, K.; Walsh, T. Improving the performance of maxRPC. In Proceedings of the CP: International Conference on Principles and Practice of Constraint Programming, St. Andrews, UK, 6–10 September 2010; pp. 69–83.
16. Wahbi, M. Maintaining arc consistency asynchronously in synchronous distributed search. In Proceedings of the ICTAI: International Conference on Tools with Artificial Intelligence, Athens, Greece, 7–9 November 2012; pp. 33–40.
17. Xu, Z.; Song, S. lmaxRPC (ls): An Algorithm Utilizing Light Symmetry for Approximating maxRPC in Constraint Programming. In Proceedings of the CAAI: Chinese Association for Artificial Intelligence, Sanya, China, 25–26 June 2018; pp. 382–385.
18. Stergiou, K. Revisiting restricted path consistency. *Constraints* **2017**, *22*, 377–402. [CrossRef]
19. Michel, L.; Hentenryck, P.V. Activity-Based Search for Black-Box Constraint Programming Solvers. In Proceedings of the CPAIOR: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Nantes, France, 28 May–1 June 2012; pp. 228–243.
20. Freuder, E.C. Eliminating interchangeable values in constraint satisfaction problems. In Proceedings of the AAAI'91, Anaheim, CA, USA, 14–19 July 1991; pp. 227–233.
21. Cooper, M. Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artif. Intell.* **1997**, *90*, 1–24. [CrossRef]
22. Benhamou, B. Study of symmetry in constraint satisfaction problems. In Proceedings of the PPCP'94, Seattle, WA, USA, 2–4 May 1994; pp. 249–257.
23. Gent, I.; Petrie, K.; Puget, J. Symmetry in constraint programming. In *Handbook of Constraint Programming*; Rossi, F., van Beek, P., Walsh, T., Eds.; Elsevier: Amsterdam, The Netherland, 2006; Chapter 10.
24. Gervet, C.; Van Hentenryck, P. Length-lex ordering for set CSPs. In Proceedings of the AAAI'06, Boston, MA, USA, 16–20 July 2006; pp. 48–53.
25. Marquis, P.; Papini, O.; Prade, H. *A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms*; Springer Nature: Berlin/Heidelberg, Germany, 2020; pp. 174–176.
26. Stergiou, K. Neighborhood singleton consistencies. *Constraints* **2019**, *24*, 94–131. [CrossRef]
27. Wallace, R.J. Interleaving levels of consistency enforcement for singleton arc consistency in CSPs, with a new best (N) SAC algorithm. In Proceedings of the International Conference of the Italian Association for Artificial Intelligence, Online, 25–27 November 2020; pp. 301–317.
28. Li, Z.; Yu, Z.; Li, H.; Guo, J.; Li, Z. Revisiting the efficacy of weak consistencies: A study of forward checking. *Sci. China Inf. Sci.* **2021**, *64*, 179102. [CrossRef]
29. Rossi, F.; Van Beek, P.; Walsh, T. *Handbook of Constraint Programming*; Elsevier: Amsterdam, The Netherlands, 2006; pp. 57–61.
30. Lecoutre, C. A greedy approach to establish singleton arc consistency. In Proceedings of the International Joint Conference on Artificial Intelligence, Edinburgh, UK, 30 July–5 August 2005; Morgan Kaufmann Publishers: San Francisco, CA, USA, 2005; pp. 199–204.
31. Berlandier, P. Improving domain filtering using restricted path consistency. In Proceedings of the Artificial Intelligence for Applications, Los Angeles, CA, USA, 20–23 February 1995; pp. 32–37.
32. Debruyne, R. A property of path inverse consistency leading to an optimal PIC algorithm. In Proceedings of the European Conference on Artificial Intelligence, Berlin, Germany, 20–25 August 2000; IOS Press: Amsterdam, The Netherland, 2000; pp. 88–92.
33. Freuder, E.C.; Elfe, C.D. Neighborhood Inverse Consistency Preprocessing. In Proceedings of the AAAI-96, Portland, OR, USA, 4–8 August 1996; pp. 202–208.
34. Yong, K.W.; Mouhoub, M. Using conflict and support counts for variable and value ordering in CSPs. *Appl. Intell.* **2018**, *48*, 2487–2500. [CrossRef]
35. Narodytska, N.; Walsh, T. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. In Proceedings of the IJCAI: International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 149–154.
36. Boussemart, F.; Hemery, F.; Lecoutre, C.; Sais, L. Boosting systematic search by weighting constraints. In Proceedings of the ECAI: European Conference on Artificial Intelligence, Valencia, Spain, 22–27 August 2004; pp. 146–150.
37. Bessiere, C.; Carbonnel, C.; Dries, A.; Hebrard, E.; Katsirelos, G.; Narodytska, N.; Quimper, C.G.; Stergiou, K.; Tsouros, D.C.; Walsh, T. Learning constraints through partial queries. *Artif. Intell.* **2023**, *319*, 103896. [CrossRef]
38. Bachtis, D.; Aarts, G.; Lucini, B. Adding machine learning within Hamiltonians: Renormalization group transformations, symmetry breaking and restoration. *Phys. Rev. Res.* **2021**, *3*, 013134. [CrossRef]