Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

Speeding up the Arc Consistency algorithm in Constraint Satisfaction Problems: A New Modification of AC-3

Yaser Shokri-Kalandaragh*[1]

[1]Department of Advanced Technologies, University of Mohaghegh Ardabili.

ABSTRACT

Dealing with constraints is always very common in real-world implementation issues. Search algorithms for real problems are also no exception. Because of the constraints in search problems (named Constraint Satisfaction Problems (CSPs)), their main solving algorithm is presented in backtracking form. The constraint propagation algorithm is an auxiliary tool to avoid facing constraint conditions as well as reducing search options. This algorithm has been presented in almost seven versions so far. In this paper, we have updated the third version of this algorithm, which is presented under the title of AC-3, from five aspects and have increased its capabilities. The most important feature of our proposed algorithm is its low time complexity. This feature has been made possible by two auxiliary criteria introduction for detecting more critical binary constraints. Faster investigation of critical constraints leads to early detection of dead-end in the search path and the search continues in this direction stops.

Keyword: Search Algorithm, Constraint Satisfaction, Constraint Propagation, Arc Consistency, Binary Constraint.

AMS subject Classification: 05C85.

ARTICLE INFO

Article history:
Research paper
Received 24, April 2022
Received in revised form 11, May 2022
Accepted 28 May 2022
Available online 01, June 2022

*Corresponding author: Y. Shokri-Kalandaragh. Email: shokri@uma.ac.ir

1 Introduction

Solving artificial intelligence problems in most cases involves informed or uninformed searching in a state space and testing the searched states. In these types of problems, information from the problem parameters is an auxiliary tool to guide the states search. One of the most common ways to add auxiliary information to the search process is to use heuristic functions. This is called informed search. Another type of informed search problems are ones in which additional information are expressed as constraints on state variables. We are going to discuss the background of the issue and express the existing challenges. A simple backtracking algorithm is one of the first algorithms presented to solve Constraint Satisfaction Problems (CSPs). This algorithm is based on the depth-first search technique [6-19]. In this method, variables are not initially assigned altogether and gradually, one variable is assigned in each step. By assigning each variable, the constraint test is performed and if the desired variable violates one of the constraints, the backward moving is done.

By default, this algorithm is uninformed, and assuming that the space has a finite variable set with discrete allowed domains, it operates randomly in two steps: 1) selecting the variable to be assigned and 2) selecting a value for the variable. In order to avoid the time complexity caused by random selections, several heuristic functions such as Minimum Remaining Values (MRV), Degree heuristic function and least-constraining-value heuristic function were presented [16-12]. However, despite the proposed heuristic functions in the backtracking algorithm, because the final check was made after the assignments, the possibility of reaching a dead-end in one direction and performing a reversal was not far from expectation [19]. Repeated reversals increased the time complexity and hence the forward checking method was presented [2-5]. In this method, with each new assignment, the violation of the constraints of the variables related to this assignment (variable) is investigated and the unauthorized items are removed from the domain of these variables so that the search algorithm is never involved in them. This feature makes the forward checking method more efficient than the simple backtracking method, but it still has its drawbacks.

The main problem is that any assignment may inevitably violate the constraint after a few more assignments for other variables, and the continuation of this path is doomed to failure. To solve this problem, a method was presented named constraints propagation [9]. The first version of these algorithms, known as the Arc Consistency (AC) algorithm was introduced to apply to binary CSPs. In this method, a constraint graph is drawn in which each variable is identified by a node and each constraint by an arc that must be consistent. In this definition, the arc that connects X_1 to X_2 is consistent if there is a member (value) $x_2 \in D_2$ for each $x_1 \in D_1$ that does not violate the constraints. In this algorithm, for the next assignment all the arcs in the constraint graph must be consistent. If an arc is not consistent, it can force to be consistent by removing values from the source variable domain [9]. Since the introduction of the AC algorithm in 1977, various versions until AC-7 have been introduced [7]. Among these versions, the AC-3, also listed in the reference book of [15], was more popular for its simplicity, flexibility,

and scalability. For these reasons, several studies have been conducted to modify AC-3, including [3] where by the introduction of two new algorithms which are performed better than the AC-6 in worst-case conditions, the AC-3 is fundamentally changed. A similar procedure has been followed in [24], except that the general form of AC-3 has been preserved. In the meantime, CSP issues with non-binary constraints were also considered, so that in [4] three new modifications in AC-3 with the approaches of arc propagation, variable propagation and constraint propagation were discussed. In both variable and constraint propagations, the authors have stated that the main goal is to be able to set up non-binary constraints. After this article, two approaches named variable-oriented and constraint-oriented propagation, became formalized [14]. Non-binary CSPs consistency algorithms are generally known as Generalized Arc Consistent (GAC) algorithms [17-10]. It is noteworthy that since the introduction of non-binary CSPs, researchers have tried to turn them into binary ones. Two well-known references in this field called binary encoding are [13] and [11]. These conversion efforts were due to the fact that practical researches were done and were being done on binary CSP issues [22-18]. Shortcomings elimination in this field has occupied the minds of researchers tile today [20-8-21]. Researches are ongoing to update the (G)AC algorithm [23] and they are also being used to solve new problems [1].
Despite the numerous methods proposed to improve the performance of the AC-3 algorithm, or in other words, the constraint propagation algorithm, the issue at stake here is a compromise between the performed calculations and the struck shortcuts. This compromise point can be shifted many times and potentially valuable results can be achieved. In this paper, a new method is considered to improve the performance of AC-3 algorithm so that the algorithm complies with the backtracking algorithm and operates faster than all AC-i versions.
We have used the "Arc Consistency Factor" and source node "Crisis Factor" ideas, in order to identify and address inconsistent arcs faster. The existence of these two factors speeds up the CSP solving process with the help of the arc propagation method and makes it more effective.

2 Principles

Definition: In the classical definition, a constraint satisfaction problem is represented by a multiple in the form $CSP = \langle X, D, C \rangle$, and is the assignment of values to a set of n variables $X = \langle X_1, X_2, ..., X_n \rangle$ from domains $D = \langle D_1, D_2, ..., D_n \rangle$ such that for each variable $X_i \in D_i$ and must have satisfy constraints $C = \langle C_1, C_2, ..., C_m \rangle$ where each expresses a constraint on one or more variables. An assignment in which no constraint is violated is a consistent assignment and if all variables are set, this is a complete assignment.
Before the main discussion, let us take a look at the flowchart of the backtracking algorithm in the CSPs solving in Figure 1. In this flowchart, we will go back to the block specified by II for two reasons, first, when the kth variable assignment is not compatible with its constraints, and second, if for the assignment of the kth variable, the other vari-

ables cannot have a candidate that satisfies all the constraints. Also, in this flowchart, the position of the constraint propagation algorithm, which has been specified with a gray box, is better understood. The two blocks identified by I and II are the steps mentioned earlier, for which a heuristic function is provided in references [16] and [12].

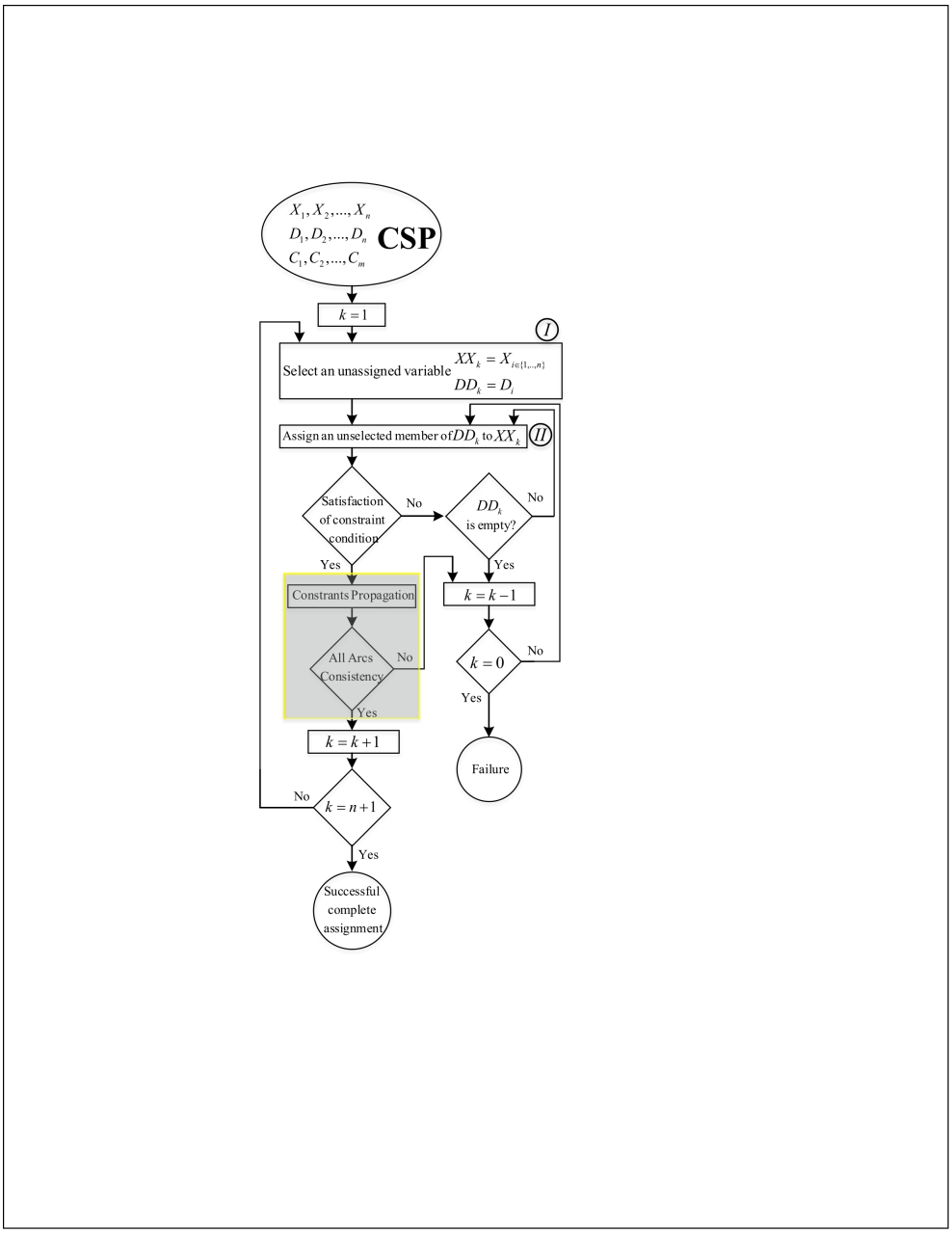

Figure 1: Backtracking algorithm flowchart in solving CSPs

In the next section, the prioritization criteria of the constraint graph arcs are introduced and the details of implementation and time complexity of the proposed constraint propagation algorithm are investigated.

3 New Proposed Algorithm

Relative criterion (first criterion): the Consistency Factor of each arc is defined as follows:

$$S_{Factor} = \frac{Distination\ node\ domain\ cardinality}{Source\ node\ domain\ cardinality} \tag{1}$$

It is clear that the larger consistency factor means there is more relative choice at the arc destination and the arc is more consistent. The arc with the lowest consistency factor will be the **critical arc** and its source will be the critical node. The critical arc is in priority in the consistency investigation. If this arc is inconsistent, it can be consistent by removing some members of its source node domain. In this case, it should be noted that the consistency factor of the constraint arc connected to that node changes. The time complexity of the Arc Consistency algorithm primarily depends on the number of arcs are investigated after each value assigned to each variable.

If the number of variables of CSP is n $(X_1, X_2, ..., X_n)$, the AC algorithm will run n-1 times. Let's consider the initial state in which any assignment has not yet been made. If the number of discrete values in the domain of each variable is assumed to be d, each (X_i, X_j) arc will be checked for the number of values in its source node (X_i), which is d and by removing each member of D_i all (X_k, X_i) arcs that are connected to X_i, enter to the cycle for reconsideration. In general, it can be said that each (X_k, X_i) arc at most enters the algorithm d times and will be checked d times each time, so each arc is checked d^2 times, which means that with the maximum number of arcs $(n^2 - n)$, the AC algorithm complexity order will be $O(d^2n^2)$.

By applying the consistency factor for each arc, the sooner checking chances of the inconsistent arc increases. In this situation, the d entry for one arc is canceled if the arcs in the algorithm queue are prevented from reentering. In this case, the complexity of the AC algorithm is reduced to $O(dn^2)$.

The second criterion will be used as a complement to the first criterion. This criterion uses the information of two heuristic functions of MRV and degrees [16-12] at the same time and can be used instead of one of them in the step specified by the symbol I in Figure **??**. In this criterion, a combination of the two approaches introduced in [14] has been used.

Absolute Criterion (Second Criterion): For each of the nodes (variables) of the constraint graph, the following crisis factor is defined:

$$Node_{CFactor} = \frac{Number\ of\ constraints\ affecting\ the\ node}{Node's\ domain\ cardinality} \tag{2}$$

The magnitude of this factor for each node indicates its criticality in the search tree and its high potential for failure or to reach to dead-end. Logically, this variable or its corresponding node could be the source of the arc with a low consistency factor.

By the introducing of these two criteria as well as some other considerations, which probably some of them are also mentioned in different versions of the AC algorithm, the proposed flowchart for the AC algorithm is described in Figure 2. This flowchart with one

input and two outputs will be placed in the location of the gray block shown in Figure 1 to complete the entire backtracking algorithm. The two introduced criteria in (1) and (2) are used in the flowchart of Figure 2 in Block I. Some other considerations have been taken into account in the initial block and stage specified by II to avoid additional calculations. In Figure 2, in order to make arcs consistent, by removing each member from D_{jj}, all the arcs that are destined to X_{jj} are reentered in the queue in block II, and the sorting operation is also performed in the queue again.

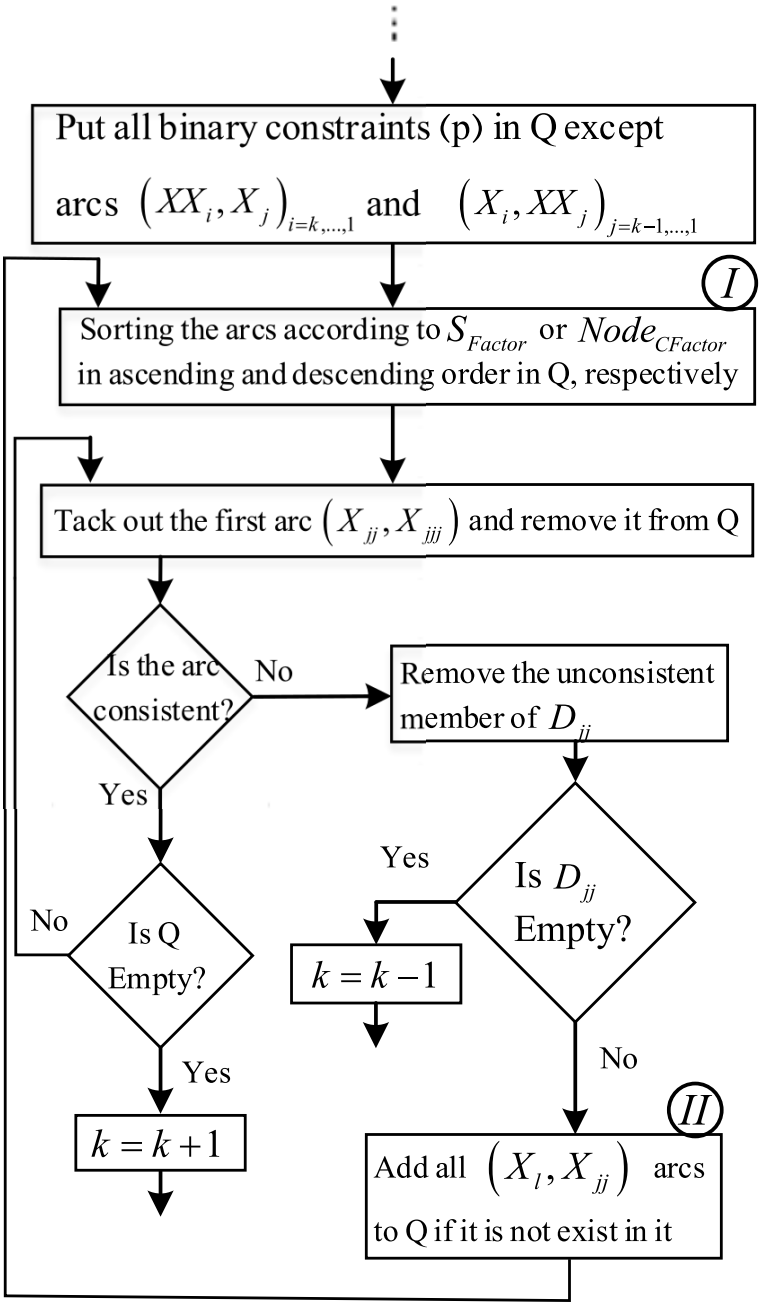

Figure 2: Proposed Arc Consistency algorithm flowchart

3.1 Constraint graph arc consistency proposed algorithm

The proposed algorithm for implementing the flowchart shown in Figure 2 is given below under the title of *AC_New* function. This algorithm can be executed many times in the structure of Figure 1.

function $AC\_New\,(CSP)$ returns CSP *possibly with reduced domains or* $failure$
  inputs : CSP, *A binary CSP with assigned variables* $\{XX_1, XX_2, ..., XX_k\} \subset \{X_1, X_2, ..., X_n\}$
  local variables : *queue*, *a queue of the Arcs*, *initially all the Arcs in* CSP *except* $(XX_i, X_j)_{i=k,k-1,...,1}$ & $(X_i, XX_j)_{j=k-1,k-2,...,1}$
  $queue = SORT\,(queue, CSP)$
  while *queue is not empty* do
    $(X_{jj}, X_{jjj}) \leftarrow$ Remove_First $(queue)$
      if Remove $\_\, Inconsistent\_\, Values\ (X_{jj}, X_{jjj})$ then
        if D_{jj} *is empty* then return $failure$
        for *each* X_l *in* NEGHBORS $[X_{jj}] - \{X_{jjj}, XX_k, XX_{k-1}, ..., XX_1\}$ do
          if $(X_l, X_{jj}) \notin queue$ then
          add (X_l, X_{jj}) *to queue*
    $queue =$ SORT $(queue, CSP)$
  return CSP

==========

function SORT $(queue, CSP)$ returns *queue*
  for $i = 1$ *to* $length\,(queue)$ do
    Insert $\_end\,(squeue) \leftarrow \mathrm{S}_{Factor}\,(queue\,(i))$
  if $i > 1$ then
    for $j = i$ *to* 1 do
      if $squeue\,(j) < squeue\,(j-1)$ then
        $x \leftarrow queue\,(j)$
        $queue(j) \leftarrow queue\,(j-1)$
        $queue\,(j-1) \leftarrow x$
      else if $squeue\,(j) = squeue\,(j-1)$ then
        if $\mathrm{Node}_{CFactor}$ (SORCE $(queue\,(j))) >$
            $\mathrm{Node}_{CFactor}$ (SORCE $(queue\,(j-1)))$ then
          $x \leftarrow queue\,(j)$
          $queue(j) \leftarrow queue\,(j-1)$
          $queue\,(j-1) \leftarrow x$
  return *queue*

==========

function REMOVE $\_INCONSISTENT\_VALUES\,(X_i, X_j)$ returns *true* *or* $false$

$removed \leftarrow false$
for *each* x *in* D_{jj} do
if *no value* y *in* D_{jjj} *allows* (x, y) *to satisfy the constraint between* X_{jj} *and* X_{jjj} then
delete x *from* D_{jj}
$removed \leftarrow true$
return *removed*

The main body of this algorithm is the same as AC-3, which has been presented in [15]. AC-New by introducing a new SORT function, receives a CSP and checks its arc consistency. This function first must be able to receive a CSP problem at any stage (meaning assigned to any number of variables) which was not possible in AC-3. In AC-New if an arc is not consistent, its source node domain is corrected and inconsistent members are removed. At this stage, if all the arcs are consistent, the problem is returned, but if the domain of any unassigned variable becomes empty, it means that the assignment has not been correct up to this stage and a backward step must be made. This situation is notified to the main algorithm when the failure value is returned. The SORT function first tries to sort the arcs of the constraint graph based on S_{Factor}, and if this criterion is equal for two arcs, it uses the $Node_{CFactor}$ criterion that compares the criticality of the source nodes of the arcs with each other.
In this paper, in the new proposed algorithm for arc consistency or constraint propagation we have improved the third version of this algorithm in several stages. The first is to prevent duplicate arcs from reentering to check queue, but since this is not very innovative in the example implementation section, it is assumed that AC-3 itself has this capability. The second case is to adjust the algorithm to coordinate with the backtracking algorithm so that it can be executed after assigning values to some variables. The third case is to remove the assigned arcs from the check queue because these arcs are once checked in the backtracking algorithm. The fourth case is the removal of the checked arcs in the previous cycle, in the other words, the arcs related to the variables assigned in the previous step are not re-investigated in the current cycle. The fifth case, which is the main innovation, is to sort the arcs according to two criteria and increase the chance of early inspection of the arc that is inconsistent. This not only prevents the arcs from re-entering the queue, but also reduces the number of times an arc is checked.
It should be noted that in the AC-New algorithm, a sorting operation has been added that will increase the computational complexity. Regarding the fact that a sort operation with the n^2 members has a computational complexity of the order $O\left(n^2 \log n^2\right) = O\left(n^2 \log n\right)$, as long as $d^2 > \log n$, the new algorithm can be more efficient.
In the next section backtracking algorithm with the constraints propagation option is implemented on the famous example of Australia's states coloring and the improvement rate when using AC-New compared to AC-3 is shown in the form of a table.

4 Example implementation

Let's consider the Australia's states coloring with red, blue, and green CSP. The constraint is that no two neighboring states can have the same color.

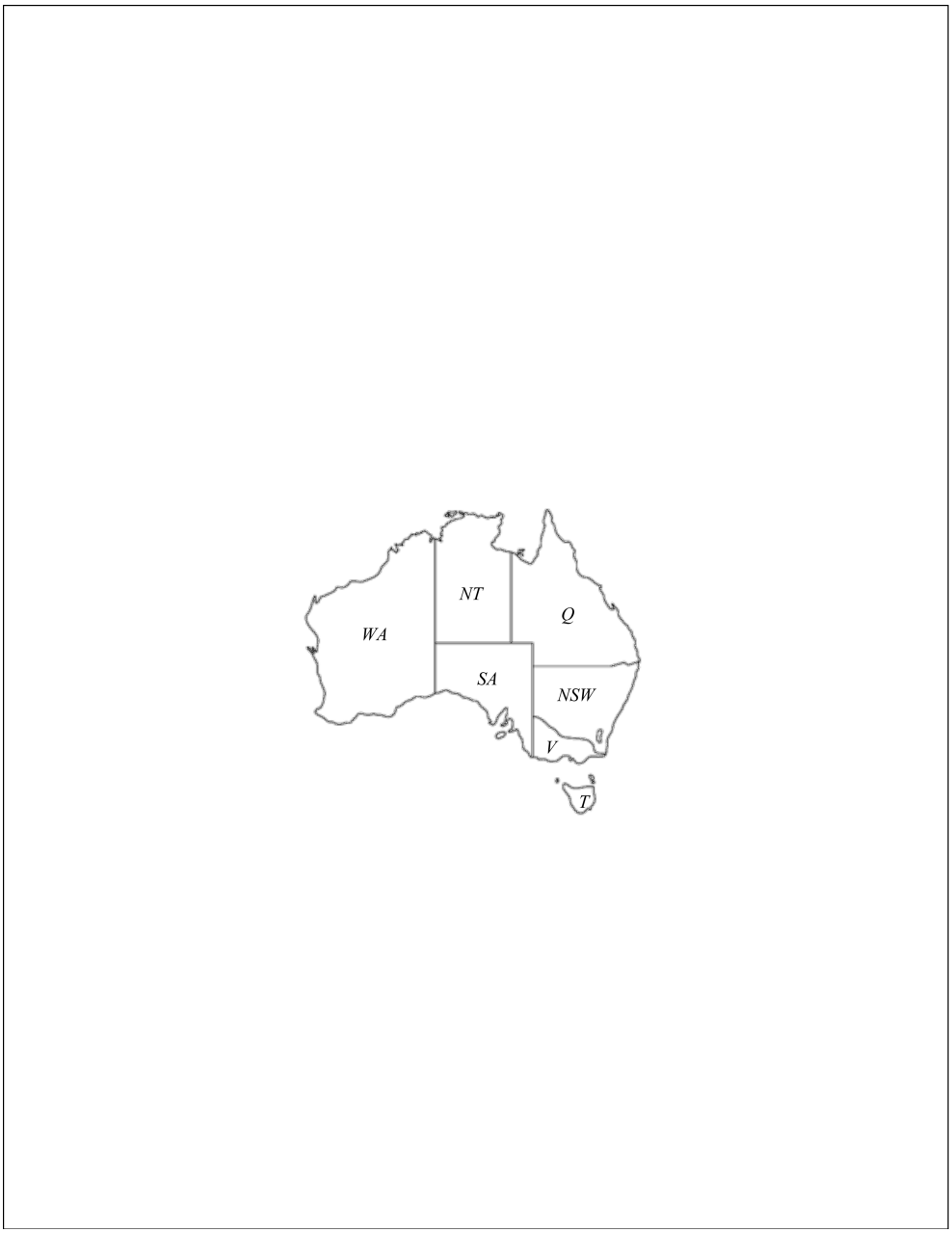

Figure 3: Map of Australia with its states

In this example, in order to be able to measure the performance of the constraint propagation or arc consistency algorithm, first we will not use any heuristic function in the overhead backtracking algorithm shown in Figure 1 at the steps specified by I and II.

Therefore, the selection of variables for coloring and also the assignment of colors to the variables will be based on the letters sort of the English alphabet.
We first assume that the backtracking algorithm with AC-3 is used to solve this problem. The staining steps and the active arcs of the constraint graph in each step are shown in Figure 4.

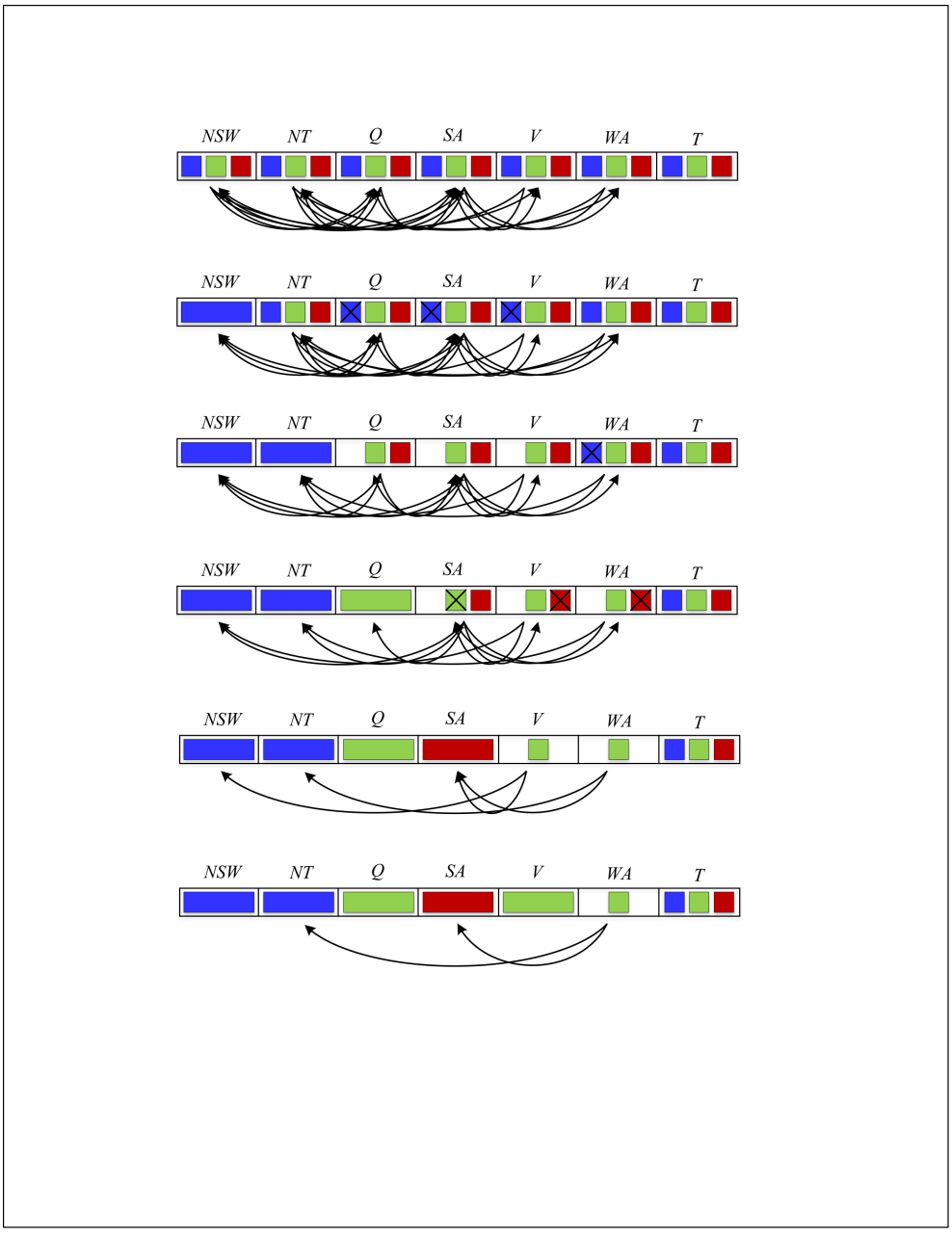

Figure 4: Steps were taken in coloring the states of Australia by backtracking algorithm and AC-3

The six steps of arc inspection are shown in Figure 4, and in the last two steps, WA and T are colored and the arcs are no longer checked.

Now let's assume to solve the same problem with the general backtracking algorithm and the AC-New auxiliary algorithm proposed in this paper. In the first cycle, the performance of this algorithm is similar to the previous one and 18 arcs each are checked three times. In the second cycle, there are 15 arcs and in this arcs investigation, priority is given to (SA, NSW) then (Q, NSW) and then (V, NSW) and in the first three checks, blue color is removed from three variables domain and no checked arc will re-enter the queue. Through these 15 arcs, eight arcs three times and seven arcs two times are checked. Details of the third cycle are given in the first part of Figure 5. At this cycle, (WA, NT) is the first arc that should be checked where the blue color is removed from it, and this is the only arc that is checked three times. Therefore, at this cycle, nine arcs are checked, of which one arc is checked three times and eight arcs are checked two times. In the fourth cycle, the details of which are given in the second part of Figure 4, first the arcs (SA, Q), (V, SA) and (W, SA) are checked, respectively and one color is removed from each of their domains.

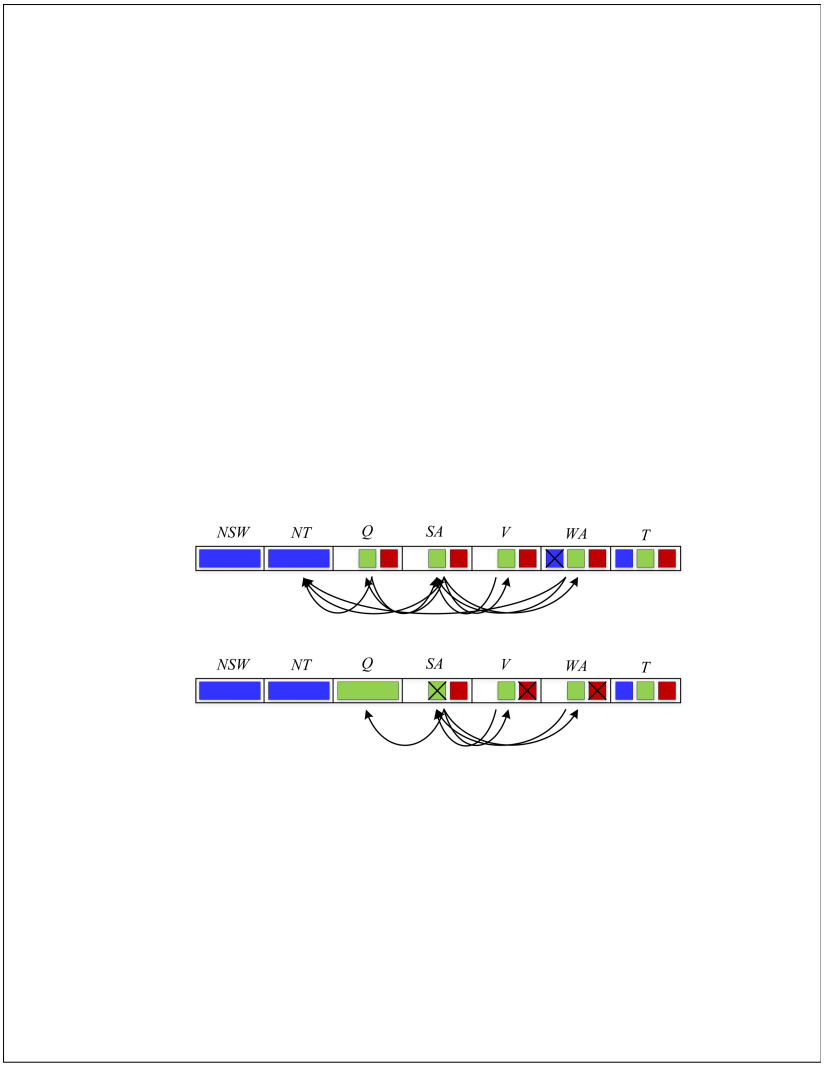

Figure 5: cycles taken in coloring the states of Australia by backtracking algorithm and

AC-New

At this cycle, five arcs are checked, of which 3 arcs are checked twice and two arcs are checked once. In the fifth cycle, red is assigned for SA, and only two arcs with the source of V and WA are checked once, and in the other cycles, no arc remains to be checked. The results of solving this example using the two algorithms AC-3 and AC-New are summarized in Table 1.

Table 1: backtracking algorithm testing by using the Ac-3 and AC-New on the coloring example of the Australian states

	AC-3 algorithm		**AC-New algorithm**	
	Arcs num	Checks num	Arcs num	Checks num
First cycle	18	54	18	54
Second cycle	19	53	15	38
Third cycle	13	27	9	19
Fourth cycle	11	18	5	8
Fifth cycle	4	4	2	2
Sixth cycle	2	2	0	0
Total	67	158	49	121

It can be seen that a tangible relative improvement has been created.

5 Conclusion

According to the result obtained in Table 1 and all the cases mentioned at the end of the two previous sections, the number of arcs and the number of times the arcs are checked has been reduced in general and the constraint propagation algorithm's time complexity has been improved. Briefly, through the five modifications which have been done over AC-3 algorithm, each of the first four modifications is seen in some way in the AC-i versions, which are integrated in this paper and we have completed the discussion with the fifth modification. Theoretically, according to the discussion has been done in section 4, since in the example $9 > \log 6$, the result obtained in the table is also justifiable.

References

[1] Alanazi, E., Arc Consistency for Constrained Lexicographic Preference Trees. IEEE Access, 8 (2020) 59694-59700.

[2] Bacchus, F., Extending forward checking. In International Conference on Principles and Practice of Constraint Programming, (2000) 35-51.

[3] Bessiere, C. and Régin, J.C., Refining the Basic Constraint Propagation Algorithm. In IJCAI, 1 (2001) 309-315.

[4] Boussemart, F., Hemery, F. and Lecoutre, C., Revision ordering heuristics for the Constraint Satisfaction Problem. In Proceedings of CPAI'04 workshop held with CP'04, (2004) 29-43.

[5] Dent, M.J. and Mercer, R.E., Minimal forward checking. In Proceedings Sixth International Conference on Tools with Artificial Intelligence, (1994) 432-438.

[6] Fillmore, J.P. and Williamson, S.G., On backtracking: a combinatorial description of the algorithm. SIAM Journal on Computing, 3 (1974) 41-55.

[7] Freuder, E.C., Using inference to reduce arc consistency computation. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, (1995) 592–598.

[8] Isli, A., A binarized-domains arc-consistency algorithm for TCSPs: its computational analysis and its use as a filtering procedure in solution search algorithms, arXiv preprint arXiv:2002.11508, (2020).

[9] Mackworth, A.K., Consistency in networks of relations. Artificial intelligence, 8 (1997) 99-118.

[10] Mackworth, A.K., On reading sketch maps, Department of Computer Science, University of British Columbia, (1997).

[11] Mamoulis, N. and Stergiou, K., Solving non-binary CSPs using the hidden variable encoding. In International Conference on Principles and Practice of Constraint Programming, (2001) 168-182.

[12] Mouhoub, M. and Jafari, B., Heuristic techniques for variable and value ordering in csps. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, (2011) 457-464.

[13] Nagarajan, S., Goodwin, S., Sattar, A. and Thornton, J., On dual encodings for non-binary constraint satisfaction problems. In International Conference on Principles and Practice of Constraint Programming, (2000) 531-536.

[14] Prud'homme, C., Lorca, X., Douence, R. and Jussien, N., A Propagation Engine Framework. Citeseer, (2012).

[15] Russell, S.J. and Peter N., Artificial Intelligence - A Modern Approach. Pearson Education, Inc., (2010).

[16] Stynes, D. and Brown, K.N., Value ordering for quantified CSPs. Constraints, 14 (2009) 16-37.

[17] Sudo, Y., Kurihara, M. and Mitamura, T., Extending Fuzzy Constraint Satisfaction Problems. J. Adv. Comput. Intell. Intell. Informatics, 10 (2006) 465-471.

[18] Ullmann, J.R., Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. Journal of Experimental Algorithmics (JEA), 15 (2011) 1-1.

[19] Van Beek, P., Backtracking search algorithms. In Foundations of artificial intelligence, 2 (2006) 85-134.

[20] Wang, R. and Yap, R.H., Arc consistency revisited. In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, (2019) 599-615.

[21] Wang, R. and Yap, R.H., Bipartite encoding: a new binary encoding for solving non-binary CSPs. In Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, (2021) 1184-1191.

[22] Yap, R.H., Xia, W. and Wang, R., Generalized arc consistency algorithms for table constraints: A summary of algorithmic ideas. In Proceedings of the AAAI Conference on Artificial Intelligence, 34 (2020) 13590-13597.

[23] Zhang, X., Gao, J., Lv, Y. and Zhang, W., Early and efficient identification of useless constraint propagation for alldifferent constraints. In Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, (2021) 1126-1133.

[24] Zhang, Y. and Yap, R.H., Making AC-3 an optimal algorithm. In IJCAI, 1 (2001) 316-321.