Abstract—Constraint satisfaction problem (CSP) can be widely applied in many areas. This paper investigates the maximum restricted path consistency algorithm. There is a large quantity of useless checks in the process of searching for a PC-support with the most popular algorithm lmaxRPC3rm. Since lmaxRPC3rm has to examine the whole domain of a variable to ascertain whether a PC-support exists. The efficiency of the search can be improved by eliminating such useless checks. Firstly, this paper analyses the features which accounts for the existence of these ineffective checks. And then, this paper discusses some methods of solving these problems. Afterwards, a new data structure is put forward to strengthen residual supports and weaken the use of multidirectional to narrow the range of search. A new algorithm, lmaxRPCls, which exploits the results above is proposed and it is proved that lmaxRPCls is correct and complete. It is also proved that the time complexity of this new algorithm is better than that of lmaxRPC3rm. Experimental results show that lmaxRPCls performs better in most benchmark instances and it can improve the performance by 65% in the best case.

Keywords—constraint programming; symmetry; maxRPC

I. INTRODUCTION

Constraint satisfaction problem (CSP) is a classical branch of artificial intelligence, and many practical problems can be explained by the constraint satisfaction problems. However, the solution to a CSP is NP-hard [1]. The general strategy is to introduce the constraint propagation techniques [2] in the backtracking search. Constraint propagation is usually based on some local consistency techniques. Classical consistency levels involve arc consistency (AC) [3], singleton arc consistency (SAC) [4], path consistency (PC) [5], and max-restricted path consistency (maxRPC) [6]. The pruning ability of maxRPC lies between the SAC and AC, providing a more reasonable balance between the consistency level and computational cost. At present, many scholars’ research work mainly focus on generating the problem-solving model [7, 8], boolean satisfiability problem (SAT) [9], and maximum satisfaction CSP problem [10].

As far as we know, there are four main algorithmic frameworks enforcing maxRPC. The first algorithm, maxRPC1, was presented by Christian Bessiere [6]. maxRPC1 is based on the fine-grained algorithm AC-6 [11], with an optimal time complexity O(end3) and space complexity O(end). maxRPC2 [13], proposed by Fabrizio Grondoni, which utilizes ideas from AC2001 / 3.1 [12]. The time complexity is the same as maxRPC1 while the space complexity is O(ed). Later, Julien Vion et al. proposed a new coarse-grained algorithm maxRPCrn [14]. Similar to AC3rm [15], residual techniques were exploited to reduce redundant checks and the time and space complexity is O(en2d4) and O(end) respectively. Although maxRPC2 has the optimal time complexity and suboptimal space complexity, maxRPCrn is still advantageous during search. Recently, Thanasis Balafoutis et al. proposed two coarse-grained algorithms, maxRPC3 and maxRPC3rm [16]. maxRPC3 takes advantages of maxRPC2 and AC2001 / 3.1, while maxRPC3rm is similar to maxRPCrn and AC3rm. The time complexity of these two algorithms is O(end3) and O(en2d4) and the space complexity is O(end) and O(ed) respectively. At the same time, the most popular algorithm MAC has attracted a lot of attention. Chavalit Likitvivatanavong et al. proposed the ACS-resOpt algorithm [17], which utilizes a new data structure for the process of searching for AC-support to avoid searching the entire domain of a variable. Afterwards, Thanasis Balafoutis et al. exploited ideas from ACS-resOpt algorithm in maxRPC3rm and maxRPC3 and proposed the lmaxRPC3rm-resOpt algorithm. Although the algorithm has decreased the theoretical complexity, but experimentation shows it achieve a poor performance due to complicated operations on data structures [18]. Further experiments show that lmaxRPC3 and lmaxRPC3rm are best suited in the preprocessing phase, while lmaxRPC3rm is best suited to use during search [19].

This paper analyzes the characteristics of the popular maxRPC algorithms: the search for a PC-support must be done with iterating the entire domain. This paper then discusses the method of reducing such checks and finds that it is possible to narrow the search range if the residual technique is strengthened and the symmetry is weakened in some places. Then, a new algorithm lmaxRPCls based on this conclusion is proposed. Experimental results show that lmaxRPCls is much faster in most test cases, and can effectively reduce redundant checks with the highest performance increase by 35%. At the same time, the performance in those problems with a larger domain is more prominent, which is more valuable in many practical problems.

II. BACKGROUND

A constraint satisfaction problem (CSP) is defined as a triple (X, D, C) where: X = \{x1, x2, ..., xn\} is a set of n variables, D = \{D(x1), D(x2), ..., D(xn)\} is a set of domains, one for each variable, with maximum cardinality d, and C =
\{ c_1, c_2, \ldots, c_r \} \) is a set of \( r \) constraints. Each constraint \( c \) is a tuple \((\text{scp}(c), \text{rel}(c))\) where \( \text{scp}(c) = \{ x_1, x_2, \ldots, x_r \} \) is an ordered subset of \( X \), \( \text{rel}(c) \) is a subset of Cartesian product \( D(x_1), D(x_2), \ldots, D(x_r) \) which represents the allowed tuples for the variables in \( \text{scp}(c) \). In this paper, we focus on binary CSPs, and a constraint can be expressed by \( c_{ij} \) that specifies a scope \( \text{scp}(c_{ij}) = \{ x_i, x_j \} \). A tuple \( \tau \in \text{rel}(c_{ij}) \) = \{ \( v_i, v_j \) \}, where \( v_i \in D(x_i) \) and \( v_j \in D(x_j) \), is valid iff \( v_i \) and \( v_j \) are valid. A value \( v_i \) is valid iff it is not removed from the current domain of the corresponding variable.

In a binary CSP, a value \( v_i \in D(x_i) \) is arc consistent (AC) iff for each constraint \( c_{ij} \) there exists a value \( v_j \in D(x_j) \) called an AC-support s.t \( \{ v_i, v_j \} \) is a valid tuple. A variable is AC iff all its values are AC and a problem \( P \) is AC iff all the variables are AC.

A value \( v_i \) in \( D(x_i) \) is maximum restricted path consistent (maxRPC) iff for each constraint \( c_{ij} \), there exists a value \( v_j \in D(x_j) \) which is an AC-support for \( v_i \) and the pair \( \{ v_i, v_j \} \) is path consistent (PC). A pair \( \{ v_i, v_j \} \) is PC iff for any third variable \( x_k \), there exists a value \( v_k \in D(x_k) \) s.t \( v_k \) is an AC-support for both \( v_i \) and \( v_j \). In this case, \( v_j \) is called a PC-support for \( v_i \) and \( v_k \) is a PC-witness for the tuple \( \{ v_i, v_j \} \).

III. NEW ALGORITHM

The reason why performance is not significantly improved with lmaxRPC3rm is that the search for a PC-support always involves the entire domain and make a lot of redundant checks. At the same time, lmaxRPC3 eliminates this redundancy by ensuring that every value will be checked at most once. But its implementation is similar to lmaxRPC3rm due to its inability to exploit symmetry. Now we combine these two important techniques together.

Algorithm 1: lmaxRPC\(^2\): Boolean

```
begin
  initialization
  begin
    while \( (P \neq \emptyset) \) do
      \( P = P \setminus \{ x_j \} \)
      for each \( x_i \in X, c_{ij} \in C \)
      for each \( v_i \in D(x_i) \)
      if (not checkPCSupLoss(v_i, x_j))
        delete \( v_i \)
        \( P = P \cup \{ x_i \} \)
      return false
  end
end
```

Algorithm 2: checkPCSupLoss: Boolean

```
begin
  if (LastPC(x_i, v_i, x_j) = d_i)
    return true
  v_start = LastStart(x_i, v_i, x_j)
  for each \( v_j \in d_j, v_j \geq v_start \)
  if isCnst(v_i, v_j) and checkPCWitLoss(v_i, v_j)
    LastPC(x_i, v_i, x_j) = v_i
    LastAC(x_i, v_i, x_j) = v_j
    LastStart(x_i, v_i, x_j) = v_j
    if LastPC(x_i, v_j, x_i) \neq d_i
      LastPC(x_i, v_j, x_i) = v_i
end
```

Algorithm 1 is similar to that of lmaxRPC3rm while the main differences are in Algorithm 2. The algorithm checkPCSupLoss will firstly compute the search start position. In order to do this, we use LastStart to record the residual support. Since a PC-support is also an AC-support, the algorithm will also update LastAC(\( x_i, v_i, x_j \)) and checkPCWitLoss(\( x_i, v_j, x_i \)). Then the value of LastStart(\( x_i, v_i, x_j \)) will be modified to the current \( v_j \) position. At the same time, in order to use the symmetry, the algorithm will first detect whether LastPC(\( x_j, v_j, x_i \)) exists, and if it does not exist, it will update LastPC(\( x_j, v_j, x_i \)) for the current \( v_j \). In
order to ensure the correctness of the algorithm, since the PC-support is obtained by symmetry, the LastStart($X_i, V_j$, $X_j$) will no longer be updated. Thus, when a PC-support fails, the corresponding search will start from the previous position pointed by LastStart($X_i, V_j$, $X_j$), so that no value is ignored.

If checkPCWitLoss returns false, the algorithm will continue to try the next value in $d_j$, and if the value is already the last one in the domain, checkPCSupLoss will return false, indicating that $V_i$ has no PC-support in $X_j$.

The function checkPCWitLoss is the same as that of lmaxRPC3rm.

IV. EXPERIMENTATION

We have experimented with binary table constraint CSPs taken from C.Lecoutre’s XCSP repository which have been used in CSP competitions. Excluding those instances that are extremely hard for all algorithms, the evaluation involves 1200 instances. More details about these classes of problems could be found in C.Lecoutre’s homepage. All tests are run on a Intel(R) Core(TM) i5-6300HQ CPU @2.30GHz with 8GB RAM processing on Windows.

<table>
<thead>
<tr>
<th>instance</th>
<th>lmaxRPC3rm</th>
<th>lmaxRPC4m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t</td>
<td>cc</td>
</tr>
<tr>
<td>BH-4-4-e-3</td>
<td>3.303</td>
<td>40768</td>
</tr>
<tr>
<td>BH-4-4-e-8</td>
<td>3.911</td>
<td>40768</td>
</tr>
<tr>
<td>composed-25-10-20-3</td>
<td>27.2</td>
<td>23053</td>
</tr>
<tr>
<td>composed-25-10-20-5</td>
<td>25.5</td>
<td>18163</td>
</tr>
<tr>
<td>composed-25-10-20-9</td>
<td>28.2</td>
<td>19467</td>
</tr>
<tr>
<td>frb30-15-5</td>
<td>14.5</td>
<td>60562</td>
</tr>
<tr>
<td>langford-3-11</td>
<td>162.2</td>
<td>276159</td>
</tr>
<tr>
<td>rand-23-23-253</td>
<td>18.0</td>
<td>68319</td>
</tr>
<tr>
<td>rand-26-36-325</td>
<td>43.4</td>
<td>90725</td>
</tr>
<tr>
<td>rand-2-30-15-306</td>
<td>9.7</td>
<td>37863</td>
</tr>
<tr>
<td>tightness0.1</td>
<td>163.3</td>
<td>59109</td>
</tr>
<tr>
<td>tightness0.2</td>
<td>135.8</td>
<td>56412</td>
</tr>
<tr>
<td>tightness0.35</td>
<td>421.5</td>
<td>77800</td>
</tr>
<tr>
<td>tightness0.8</td>
<td>687.1</td>
<td>30240</td>
</tr>
<tr>
<td>tightness0.9</td>
<td>1016.1</td>
<td>16480</td>
</tr>
</tbody>
</table>

It can be seen that the performance of lmaxRPCs is superior to that of the most popular algorithm in most test cases. Most performance is improved by 10% to 35%, with performance in rand-2-23-15-306 improved by 65%. At the same time, the number of checks are largely reduced.

On tightness problems, for example, when the problem is getting harder, the improvement is more obvious. On tightness 0.1, lmaxRPCs algorithm performance is poor. This is because the problem has a low density of constraints and less ineffective PC-support, and the use of new data structures has led to a reduction in performance. On langford problem, our algorithm can be much faster than lmaxRPC3rm. Specifically, the performance increase can be up to 30%.

For the rest test cases, the reduction in checks is positively reflected on performance improvement. To be specific, when the size of domain is larger, the improvement is more obvious which can be referred from rand problems. On rand problems, the number of checks have been significantly reduced. And then this advantage is reflected on mean run time.

V. CONCLUSION

Maximum restricted path consistency has a more powerful pruning ability, but its search for a PC-support requires a lot of computational cost. This paper analyzes the combination of symmetry and residual techniques and gives a new algorithm exploiting more lightweight multidirectionality to ensure significantly less checks. Experimentation shows that lmaxRPCs can reduce checks on almost every test instance and is much faster than the most popular algorithm. When the domain of a problem gets larger, the improvement is more obvious.

REFERENCES

