Advances in Intelligent Systems Research, volume 134

2nd International Conference on Control, Automation, and Artificial Intelligence (CAAI 2017)

lmaxRPC^{ls}: An Algorithm Utilizing Light Symmetry for Approximating maxRPC in Constraint Programming

Zhiying Xu[2], Shihui Song[3] and Zhanshan Li[1,3]

[1]Laboratary of Symbol Computation and Knowledge Engineering, Education Ministry, China

[2]School of Computer Science, Jilin University, Changchun, China

[3]School of Software, Jilin University, Changchun, China

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 multidirectionality 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 Grandoni, 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 maxRPCrm[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, maxRPCrm 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 maxRPCrm 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 =

$\{c_1, c_2, \dots, c_e\}$ is a set of e constraints. Each constraint c is a tuple (scp(c),rel(c)) where scp(c) = {x1, x2, … , x_r } is an ordered subset of X, rel(c) is a subset of Cartesian product D(x_1), D(x_2),..., D(x_r) which represents the allowed tuples for the variables in scp(c). In this paper, we focus on binary CSPs, and a constraint can be expressed by c_{ij} that specifies a scope scp(c_{ij}) = { x_i , x_j }. A tuple $\tau \in$ rel(c_{ij}) ={ v_i , v_j }, where $v_i \in$ D(x_i) and $v_j \in$ D(x_j), is valid iff both 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 $v_i \in$ 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_i in D(x_i) 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 performance is similar to lmaxRPC3rm due to its inability to exploit symmetry. Now we combine these two important techniques together.

Algorithm 1: lmaxRPC^{tr}: Boolean

```
begin
   initialization
   P = currently assigned variables
   while (P ≠ ∅)
       P = P – {xj}
       for each xi ∈ X, cij ∈ C
           for each vi ∈ di
               if ( not checkPCSupLoss(vi, xj) )
                   delete vi
                   P = P ∪ {xi}
           if (di = ∅)
               return false
   return true
end
```

Algorithm 2: checkPCSupLoss: Boolean

```
   begin
1  if ( LastPC(xi, vi, xj) ∈ dj )
2      return true
3   vstart = LastStart(xi, vi, xj)
4   for each vj ∈ dj, vj > vstart
5       if isCnst(vi, vj) and checkPCWitLoss(vi, vj)
6           LastPC(xi, vi, xj) = vj
7           LastAC(xi, vi, xj) = vj
8           LastStart(xi, vi, xj) = vj
9       if LastPC(xj, vj, xi) ∉ di
10          LastPC(xj, vj, xi) = vi
   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 search start v_{start} (line 3). After that, it will check each value behind v_{start} (line4). The algorithm will call isCnst to check whether the current value v_j is consistent with v_i . The implementation of isCnst is easy and omitted here. For v_j compatible with v_i, the algorithm will call checkPCWitLoss to look for PC-witnesses (line 5).

If checkPCWitLoss returns true, v_j is a PC-support for vi. At this point, the algorithm will update LastPC(x_i, v_i, x_j) 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). 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_i. In order to ensure the correctness of the algorithm, since the PC-support is obtained by symmetry, the LastStart(x_j , v_j , x_i) will no longer be updated. Thus, when a PC-support fails, the corresponding search will start from the previous position pointed by LastStart(x_j , v_j , x_i), 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 I. EXPERIMENTAL RESULTS: TIME (T) AND CHECKS (CC)

instance	$lmaxRPC3^{rm}$		$lmaxRPC^{ls}$	
	t	cc	t	cc
BH-4-4-e-3	3.303	40768	3.6	42032
BH-4-4-e-8	3.911	40768	4.2	42032
composed-25-10-20-3	27.2	23053	22.8	17965
composed-25-10-20-5	25.5	18163	21.0	17137
composed-25-10-20-9	28.2	19467	22.1	18790
frb30-15-5	14.5	60562	9.4	59767
langford-3-11	162.2	276159	110.1	285144
rand-23-23-253	18.0	63819	18.4	60891
rand-26-26-325	43.4	90725	25.0	84528
rand-2-30-15-306	9.7	37863	3.4	34767
tightness0.1	163.3	59109	160.8	57768
tightness0.2	135.8	56412	129.9	50749
tightness0.35	421.5	77800	315.8	66634
tightness0.8	687.1	30240	492.3	30156
tightness0.9	1016.1	16480	605.9	16377

It can be seen that the performance of lmaxRPCls 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, lmaxRPCls 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 exploits more lightweight multidirectionality to ensure significantly less checks. Experimentation shows that lmaxRPCls 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

[1] E.C. Freuder, A.K. Mackworth. Constraint satisfaction: An emerging paradigm, J. Foundations of Artificial Intelligence. 2 (2006): 13-27.

[2] C. Bessiere. Constraint propagation. Foundations of Artificial Intelligence, J. 2 (2006): 29-83.

[3] A.K. Mackworth. Consistency in networks of relations, J. Artificial intelligence 8.1 (1977): 99-118.

[4] R. Debruyne, C. Bessiere. Some practicable filtering techniques for the constraint satisfaction problem, C. In Proceedings of IJCAI’97. 1997.

[5] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing, J. *Information sciences* 7 (1974): 95-132.

[6] R. Debruyne, C. Bessiere. From restricted path consistency to max-restricted path consistency, C. *Principles and Practice of Constraint Programming-CP97* (1997): 312-326.

[7] K. Xu, F. Boussemart, F. Hemery, C. Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances, J. *Artificial Intelligence* 171.8 (2007): 514-534.

[8] J. Gao, J. Wang, M. Yin. Experimental analyses on phase transitions in compiling satisfiability problems, J. *Science China Information Sciences* 58.3 (2015): 1-11.

[9] S. Cai, K. Su. Comprehensive Score: Towards Efficient Local Search for SAT with Long Clauses, C. *IJCAI*. 2013.

[10] S. Cai, K. Su. Local search for Boolean Satisfiability with configuration checking and subscore, J. *Artificial Intelligence* 204 (2013): 75-98.

[11] C. Bessiere. Arc-consistency and arc-consistency again, J. *Artificial intelligence* 65.1 (1994): 179-190.

[12] F. Grandoni, G. Italiano. Improved algorithms for max-restricted path consistency, C. Principles and Practice of Constraint Programming–CP 2003. Springer Berlin/Heidelberg, 2003.

[13] C. Bessière, J.C. Régin. Refining the Basic Constraint Propagation Algorithm, C. *IJCAI*. Vol. 1. 2001.

[14] J. Vion, R. Debruyne. Light algorithms for maintaining max-RPC during search, C. *SARA-2009 Eighth Symposium on Abstraction, Reformulation, and Approximation*. 2009.

[15] C. Lecoutre, F. Hemery. A Study of Residual Supports in Arc Consistency, C. *IJCAI*. Vol. 7. 2007.

[16] T. Balafoutis, A. Paparrizou, K. Stergiou, T. Walsh. Improving the performance of maxRPC, C. *International Conference on Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg, 2010.

[17] C. Likitvivatanavong, Y. Zhang, S. Shannon, J. Bowe, E.C. Freuder. Arc Consistency during Search, C. *IJCAI*. Vol. 7. 2007.

[18] T. Balafoutis, A. Paparrizou, K. Stergiou, T. Walsh. New algorithms for max restricted path consistency, J. *Constraints* 16.4 (2011): 372-406.

[19] J. Guo, Z. Li, L. Zhang, X. Geng. MaxRPC algorithms based on bitwise operations, C. *International Conference on Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg, 2011.