An optimal algorithm based on the solution to the coarse-grained arc consistency algorithms of the Constraint satisfaction Problems

Gang Yang, Huifeng Li, Can Cao, Siyuan Chen, and Yubo Zhao
College of Computer Science and Technology
Jilin University
Changchun, China, 130012

Abstract. Constraint satisfaction problem are widely used in the artificial intelligence field. We study the coarse-grained arc consistency algorithms of the Constraint satisfaction Problems. We find that variables whose degrees or domains are 1 have invalid revises which can be removed in the process. We demonstrate that these revises are redundant. We also provide an improved method to avoid the redundancy. The improved framework of the constraint problemsAC3 algorithms contain the ignorance of degree is 1, and do main is 1.The improved framework can be used on all of the coarse-grained arc consistency algorithms. The result of the test shows that we can save up to 65% revise times and 10% execution time after using the improved algorithms AC3 algorithms containing the ignorance of degree is 1, and domain is 1.

Keywords: Constraint satisfaction problem; Maintaining arc consistency; the Coarse-grained algorithms; Revise; degree is 1; domain is 1

I. INTRODUCTION

Many problems in our daily life can be solved by establishing a constraint satisfaction problem (CSP) [1] model, for instance, the resources allocation and the scheduling problem. Also, applications like the spatial graphic representation, the design constraint of hardware and the check plot, are all simulations of the CSP. This characteristic has made CSP widely use in the artificial intelligence field. A solution to CSP problem is to assign values to all the variables in their domains, making the instantiated variables satisfy the given constraints. While it was recognized that solving CSPs was in general NP-hard, a variety of analytical techniques were brought to bear to evaluate, predict or compare algorithm performance and relate problem complexity to problem structure.[1] Among those techniques, backtracking algorithm (BT) is a complete algorithm. [2] While because the basic BT is not very efficient, usually, people use constraint propagation before and during search to reduce the scale of problems, eliminating some of the inexistent variables to narrow the search space and cut down search times to improve the efficiency. Arc Consistency (AC) is the earliest and the most widely used technique among all the existed constraint propagation. Maintaining Arc Consistency (MAC) is regarded as the most sufficient method to solve large-scale of hard problems. [4] There are two types of AC classified by different types of constraint propagation, one is the coarse-grained arc consistency algorithms, which propagate on the arc, and the other one is fine-grained arc consistency algorithms, which propagate among the variables. [5] Using less cost of space, the coarse-grained algorithms are more widely used in the practice. Mackworth (1977) raised the AC3, a classical arc consistency coarse-grained algorithm, which points out that the coarse-grained arc consistency algorithm is combined of two parts: basic framework and revise.

This paper analyses the process of the coarse-grained arc consistency algorithms, and finds that there are insufficient correction check in the arc when the degree or value equals 1. We testify that this kind of revise which appears rarely in the preprocessing of the arc consistency and mostly in the processing is redundant. We provide a method called Ignore for a degree and for a domain (IDD), avoiding the redundant revise and improving the basic framework of the coarse-grained arc consistency algorithms. The improved framework AC3_IDD can be used to better all the coarse-grained MAC algorithms presented. Finally, we apply our method into the recently popular coarse-grained arc consistency algorithms, MAC3rm. The result of the experiment shows that the improved algorithms can save up to 65% of the revise times, and raise the efficiency of solving to 10%.

II. PRELIMINARIES

Definition 1(Constraint Satisfaction Problem, CSP) [1] A CSP P is a triple P = <X,D,C> where X is an n-tuple of variables X = {x1, x2, . . . , xn}, D is a corresponding n-tuple of domains D = {D1,D2, . . . ,Dn}, Di∈D is the finite domain of xi, when xi∈X, C is a t-tuple of constraints C = {C1,C2,…,Ck}. Any Cj∈C represents the constraint relation of the variables.

Definition 2 （arc consistency, AC)） [6]For a given binary CSP P, (x,Cxy) is an arc in P, if any a, of which domain is x, can find a b, of which domain is y, to make (a,b) satisfy Cxy, then arc (x,Cxy) is arc consistency. Only when each arc of the CSP is AC, can the CSP be AC.

Values which cannot find supporting from their adjacent arcs have to be removed, when processing the arc constraint check. If all the values in the domain of a variable are removed, then the problem P has no solution, return arc consistency check

fail.

Maintaining Arc Consistency(MAC) is a highly efficient basktrack search algorithms for solving CSP problems. MAC is to embed AC algorithms under the framework of BT algorithms. In the search process maintaining arc consistency situation, using AC algorithms to constraint propagation each time after assigning a value to a variable, if we get an arc consistency situation, then the assignment is successful; otherwise, the assignment is failed and either select the next assignment or backtrack. Currently used MAC is the MAC3rm algorithms which are coarse-grained algorithms rose by Lecoutre in 2007.

Definition 3 Degree TheDegree of a variable is the number of other variables when there are constraints between the variable and the others.

Definition 4 Domain Eachdomain has its own domain. We concern not only the value of the domain, but also whether the domain is 1.

Algorithm 1 is the basic framework of the coarse-grained AC algorithms. *Q* is the collection need to be propagated, keeping all the arcs that are yet to be checked; initialize *Q* is to initialize the collection to be propagated, if AC algorithms are used in the preprocessing, then we need to add all the arcs in the problem into *Q* to initialize *Q*. In the process of arc consistency check, if the domain of any variables changed, then we have to add all the arcs pointing to that variable to *Q* and check the arcs again. If the domain of any variable is empty, then return check fail; otherwise when Q is empty, return arc consistency check successful. In algorithm 2, function *revise* is the check and revise process of the coarse-grained algorithm. *Dx* is the finite domain of *x*, *Revise(x,c)* is to find supports for all the unremoved value in the domain of *x* on the arc. All the AC algorithms mentioned below are coarse-grained arc consistency algorithms.

Algorithm 1:

```
AC3_frame
    begin
    initialize Q;
    while Q is not empty do
        select and remove the arc(xi,c) from Q;
        if Revise(xi,c) then
        if Di=∅ then return fail;
            else for each c′ such that xi,xj∈X(c′)
                if c′≠c then
                Q←Q∪(xj,c′);
return success;
end
```

Algorithm 2:

```
procedure  Revise(x,c)
begin
change ← false ;
        for each v belong to D[x] ,do
                If  v  has  no  support  on  c
then
                                remove  v  from  D[i] ;
                                change=true;
return  change;
```

- End

III. REDUNDANCY CHECK

Before starting the MAC algorithms, we need to process the arc consistency preprocessing through AC algorithms. If the preprocessing failed, the problem will have no solution; otherwise, we can begin to backtrack search. Therefore, the problem is in an AC situation before the backtrack search. Each time we assign a value, we need to process the constraint propagation through AC algorithms. If the AC algorithms are successful, then we can do the next assignment; otherwise, the backtrack happens. Thus, in the process of MAC algorithms search, the problem must be in an AC situation before assignment. To assign value *a* to variable *x* is to remove all the values in the domain of *x* except *a*. Therefore, each time after assignment, only the domain of the current variable will change. We only need to use all the arcs pointing to *x* to initiate the collection *Q* to be propagated, and then execute AC algorithms to process the constraint propagation.

Further, when the degree of a variable is 1, we can make sure whether the variable is arc consistency by checking the related variables in the collection because of the bidirectional support of the constraints. Therefore, the check of this variable is redundant, and ignorance of the variable will not affect the solver. We can then ignore the variables whose degrees are 1. All the related variables need to be compared with a certain value of the variable, when the domain of the variable is 1. The problem of comparing different values does not exist. Thus, ignorance of the variables whose domains are 1 will not affect the solver, when the preprocessing has passed.

As a result, the idea that a new variable whose degree is 1 can be generated when certain variable whose domain is 1 is ignored and its related variables are also ignored comes to our mind. Because of the constraint propagation, the solution of the problem can be greatly simplified. We then design a new algorithm AC3 rm_IDD:

AC3rm_IDD:

```
AC3rm_IDD  frame
    begin
    initialize Q;
    while Q is not empty do
        select and remove the arc(xi,c) from Q;
        if(xi->degree)=1  then
          xjsuch that xi,xj∈X(c)xj->num--;  continue;
        else  if  Newrevise(xi,c) then
                if Di=∅ then return fail;
else for each c′ such that xi,xj∈X(c′)
                        if c′≠c then
                        Q←Q∪(xj,c′);
return success;
End
```

AC3rm_IDD Revise:

```
procedure  Revise(x,c)
begin
change ← false ;
 if |D(x)|=1  then
foreach x'  such that x,x'∈X(c′)and  c′≠c
```

- x'->num--; return false;
- else for each v belong to D[x] ,do
- If v has no support on c then
- remove v from D[i] ;
- change←true;
- return change;
- End

In Figure 1, the ovals represent the initial domains of certain variables, the lines between values show that the two values satisfy the constraint. Before solving the problem, the result of the arc consistency preprocessing is showed in Figure 2. During the process, we first assign 0 to *x0*, add it into *Q* ,*and*then execute AC algorithms *revise*(*x0*,*c0*), after that, 0 is removed from the domain of *x2*, and the assignment of (*x0*,0) is successful. Next step, *x0* can be ignored because its domain has changed to 1 and there is no need to put it into the collection. Meanwhile, *x3* can also be ignored because its degree becomes 1 after minus by 1. Therefore, we can avoid the repetition of algorithm *revise* andsave the processing time. Figure 3 shows the process of solving the constraint problems.

Fig. 1 Previous constraint problem

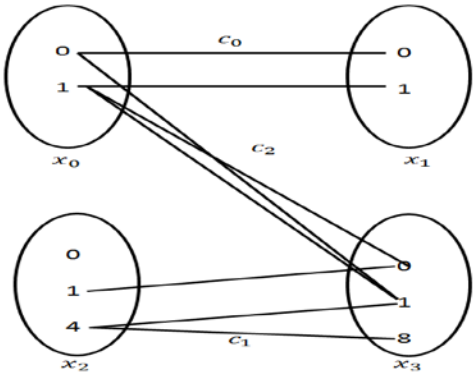


Fig. 2 Arc consistency preprocessed

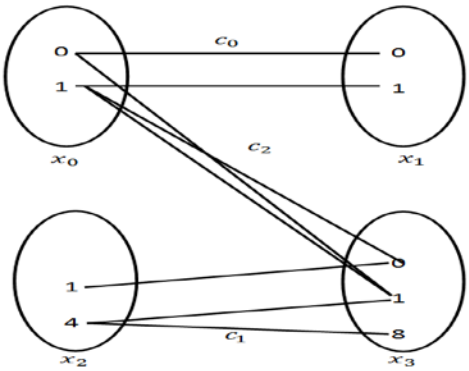
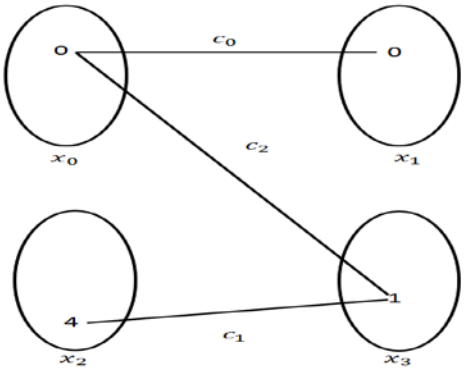


Fig. 3 Solution of constraint problem



IV. RELATED WORK AND DISCUSSION

We find that we can determine the variables in the arc before we get the arc in the collection by AC3rm algorithms and analyse. If the degree of the variable is 1, then only one relation is between the variable and other variables (the number of the constraint related to the variable is only one). According to the bidirectional support of the constraints, we can determine whether the arc is consistent when judging the variable on the other side of the constraint of the arc. So the removal of the revises can be remained, but putting back all the arcs which are related to the variable on the other side of the constraint can be deleted. That is the ignorance of the degree of 1. This method reduces the additions to the arcs in the collection, and simplifies the solution. It is the same with the checking function in AC3rm algorithms. According to the bidirectional support of the constraints, we can determine whether the arc is consistent when checking the variable on the other side of the constraint of the arc. So the check can be ignored and the efficiency of solution is improved. Also, all the variables related to the variable need to minus 1 after the ignorance of the variable whose degree is 1. The ignorance of 1 degree variables can reduce the process of supporting check, save the time of solution, and optimize the whole process. In summary, we combine the theories of degree is 1 and domain is 1. Degree is 1 can improve domain is 1. Therefore, the theory can increase the solving efficiency of the problems and reach the optimization effect. However, if the concepts of degree is1 and domain is 1 are added, we will inevitably add related data structures and functions to maintain the data structure. This will cost more spending which can impede the improvement of solving time. The reduction of the times of constraint check illustrates that it is correct to add the theory of degree is 1 and domain is 1 into the basic framework of the coarse-grained algorithm. We believe that as long as the maintenance of the data structures is briefly enough, the skill of process is of high quality, the time of the CPU execution will be shortened largely, and the save of time is very obvious.

V. TEST RESULT

We use two different algorithms to test the improvement of efficiency of the coarse-grained algorithms by IDD method. The basic algorithm is the currently international prevalent coarse-grained algorithms Mac3rm; Mac3rm_IDD is the improved algorithm which adds degree is 1 and domain is 1 skill to the basic of Mac3rm. Using IDD to improve Mac3rm, Mac3rm_IDD is a new method that used the framework of AC3_frame_IDD to improve the framework of Mac3rm. The use

case of test is the benchmark in the standard use case. Figure 4 is the line chart of the contrast of CPU execution time between MAC3rm and Mac3rm_IDD. Figure 5 is the line chart of the times of revises between MAC3rm and Mac3rm_IDD. The result of the test shows that comparing those with the coarse-grained arc consistency algorithm MAC3rm , the efficiency of CPU improved by 10% and the times of revises reduced by 65% after adding the IDD method Mac3rm_IDD.

Table 1

Test index	**CPU execution time(ms)**			**Times of revises**		
Test file	***MAC3rm***	***Mac3rm_IDD***	***Improved CPU (%)***	***MAC3rm***	***Mac3rm_IDD***	***Reduced revisetimes(%)***
frb30-15-1-bis	1091	712	34.73877177	12896	4423	65.70254342
frb30-15-2-bis	813	688	15.37515375	13454	4517	66.42634161
frb30-15-3-bis	919	853	7.18171926	13206	4736	64.13751325
frb30-15-4-bis	1110	1105	0.45045045	13144	4621	64.8432745
frb30-15-5-bis	820	794	3.170731707	13020	4437	65.92165899
frb35-17-1-bis	727	661	9.078404402	18864	6947	63.17324003
frb35-17-2-bis	1667	1475	11.51769646	18720	6058	67.63888889
frb35-17-3-bis	1509	1311	13.12127237	19152	6639	65.33521303
frb35-17-4-bis	1749	1489	14.86563751	18864	6559	65.23006785
frb35-17-5-bis	1226	1068	12.88743883	19656	6397	67.45522996
frb40-19-1-bis	1771	1467	17.16544325	26322	9192	65.07864144
frb40-19-2-bis	2232	2008	10.03584229	26322	9545	63.73755794
frb40-19-3-bis	2059	1895	7.965031569	25256	8710	65.51314539
frb40-19-4-bis	2286	2008	12.16097988	26732	8745	67.28639832
frb40-19-5-bis	1941	1631	15.97114889	26732	9006	66.3100404
frb45-21-1-bis	2707	2472	8.681196897	36248	12612	65.20635621
frb45-21-2-bis	2315	2091	9.676025918	34960	11229	67.88043478
frb45-21-3-bis	2313	2123	8.214440121	33948	10887	67.93036409
frb45-21-4-bis	2470	2179	11.78137652	34776	11571	66.72705314
frb45-21-5-bis	2451	2123	13.38229294	34500	12279	64.40869565
frb50-23-1-bis	3025	2582	14.6446281	43860	14676	66.53898769
frb50-23-2-bis	3789	3317	12.45711269	45390	16235	64.23220974
frb50-23-3-bis	2870	2609	9.094076655	46512	14732	68.32645339
frb50-23-4-bis	2999	2742	8.569523174	44370	15840	64.30020284
frb50-23-5-bis	2988	2588	13.38688086	43656	14730	66.25893348
frb53-24-1-bis	3253	2904	10.72855825	51300	17660	65.57504873
frb53-24-2-bis	3271	2991	8.560073372	51192	16717	67.34450695
frb53-24-3-bis	3308	2946	10.94316808	50760	16987	66.53467297
frb53-24-4-bis	3754	3327	11.37453383	51084	17199	66.33192389
frb53-24-5-bis	3563	3363	5.613247264	51408	17445	66.0655929
frb56-25-1-bis	3337	2846	14.7138148	58938	20559	65.11758119
frb56-25-2-bis	3804	3297	13.32807571	58254	18957	67.45802863
frb56-25-3-bis	3398	2983	12.21306651	58254	19631	66.30102654
frb56-25-4-bis	3801	3499	7.945277559	59850	20651	65.49540518

frb56-25-5-bis	2718	2522	7.211184695	58368	20524	64.83689693
frb59-26-1-bis	4415	3979	9.875424689	66720	22151	66.80005995
frb59-26-2-bis	4224	3551	15.93276515	65880	22341	66.08834244
frb59-26-3-bis	4049	3543	12.49691282	65640	21899	66.6377209
frb59-26-4-bis	4010	3702	7.680798005	67080	20985	68.71645796
frb59-26-5-bis	4224	3849	8.877840909	64680	21985	66.00958565

Fig. 4 Comparison of the CPU execution time between MAC3rm and Mac3rm_IDD

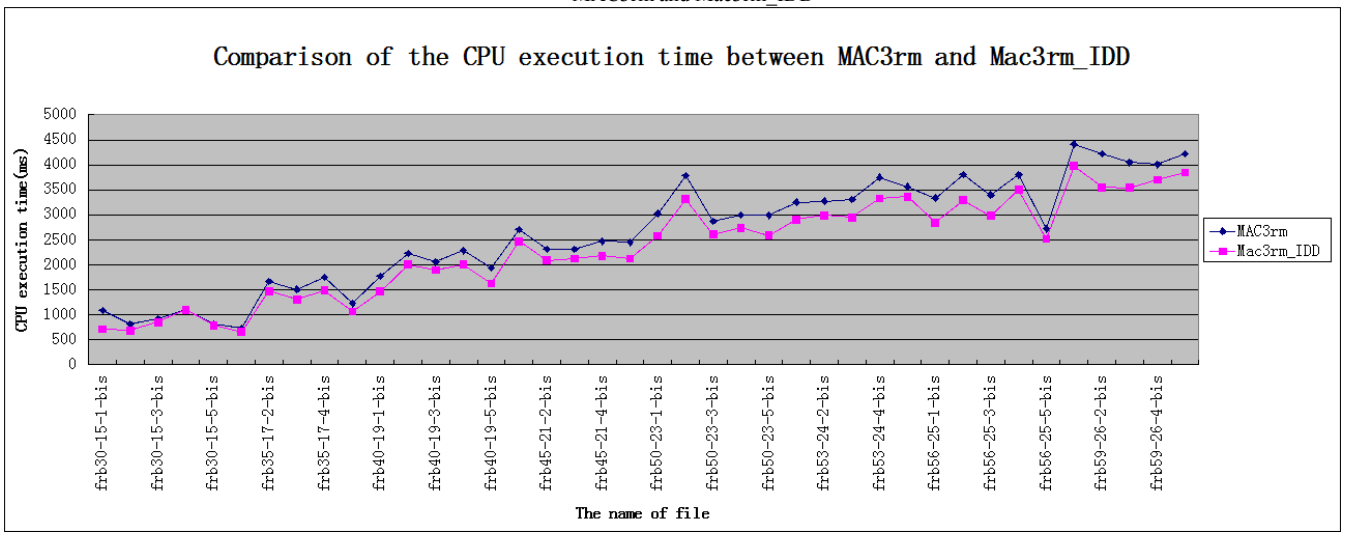
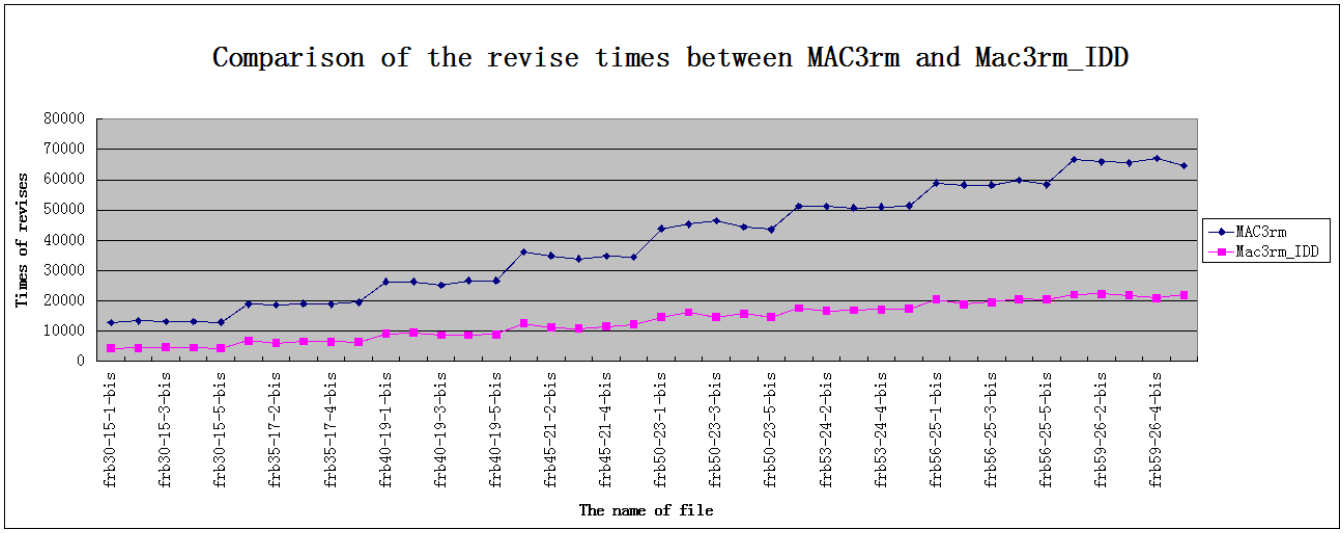


Fig. 5 Comparison of the revise times between MAC3rm and Mac3rm_IDD



VI. CONCLUSION

Revise is the essential core part of the coarse-grained arc consistency algorithms. The worst time complexity of the execution of revise is $O(d^2)$. Therefore, avoiding the useless revise can increase the efficiency of algorithms. Most of the studies of the coarse-grained algorithms are concentrated on the improvement of the revise, and few of them have improved the framework of the algorithms. We study the execution of the coarse-grained maintaining arc consistency algorithms, finding that there are some invalid revises in them. These revises are mainly focused on the variables whose degree is 1 or domain is 1. By using the bidirectional support of the constraints, we can ignore those variables whose degree or domain is 1, reducing the added arcs in the collection and the times of revise. Therefore, the workload can be reduced, and the previous algorithms are optimized.

REFERENCES

[1]. Freuder EC, Mackworth AK. Constraint satisfaction: An emerging paradigm. Rossi F, van Beek P, Walsh T, eds. Handbook ofConstraint Programming.Amsterdam:Elservier,2006.13−27.[doi: 10.1016/S1574-6526(06)80006-4]

[2]. vanBeek P. Backtracking search algorithms. In: Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming.Amsterdam: Elservier, 2006. 85−134. [doi: 10.1016/S1574-6526(06)80008-8]

[3]. Bessière C. Constraint propagation. Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam:Elservier, 2006. 29−84.

[4]. Sabin D, Freuder EC. Contradicting conventional wisdom in constraint satisfaction. In: Cohn AG, ed. Proc. of the 11th EuropeanConf. on Artificial Intelligence. Amsterdam: John Wiley & Sons, 1994. 125−129.

[5]. Bessière C, Regin JC, Yap RHC, Zhang YL. An optimal coarse-grained arc consistency algorithm. Artificial Intelligence, 2005, 165(2):165−185. [doi: 10.1016/j.artint.2005.02.004]

[6]. Mackworth AK. Consistency in networks of relations. Artificial Intelligence, 1977,8(1):99−118. [doi: 10.1016/0004-3702(77)90007-8]