软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(7):1816−1823 [doi: 10.3724/SP.J.1001.2012.04129]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

改进求解约束满足问题粗粒度弧相容算法*

李宏博[1,2], 李占山[1,2+], 王 涛[3]

[1](吉林大学 计算机科学与技术学院,吉林 长春 130012)

[2](吉林大学 符号计算与知识工程教育部重点实验室,吉林 长春 130012)

[3](长春工业大学 计算机科学与工程学院,吉林 长春 130012)

Improving Coarse-Grained Arc Consistency Algorithms in Solving Constraint Satisfaction Problems

LI Hong-Bo[1,2], LI Zhan-Shan[1,2+], WANG Tao[3]

[1](College of Computer Science and Technology, Jilin University, Changchun 130012, China)

[2](Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China)

[3](School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China)

+ Corresponding author: E-mail: zslizsli@163.com

Li HB, Li ZS, Wang T. Improving coarse-grained arc consistency algorithms in solving constraint satisfaction problems. *Journal of Software*, 2012,23(7):1816−1823 (in Chinese). http://www.jos.org.cn/1000-9825/ 4129.htm

Abstract: Constraint satisfaction problems have been widely investigated in artificial intelligence area. This paper investigates whether the coarse-grained maintaining arc is consistent, which is used to solve CSPs. The study has found that during the search for a solution, there are ineffective revisions, which revise the arcs that point to assigned variables. This study has proved that such revisions are redundant and proposed a method to avoid such redundant revisions. The paper gives the improved basic frame for the coarse-grained arc consistency algorithms, named AC3_frame_ARR. The new frame can be applied to improve all the coarse-grained AC algorithms. The experimental results show that the improved algorithms can save at most 80% revisions and 40% CPU time.

Key words: constraint satisfaction problem; maintaining arc consistency; coarse-grained algorithm; revision

摘 要: 约束满足问题在人工智能领域有着广泛的应用.研究了约束满足问题的粗粒度维持弧相容求解算法,发现在求解过程中,对于指向已赋值变量的弧存在无效的修正检查,证明了这类修正检查是冗余的.提出一种方法避免这类冗余的修正检查,给出改进后的粗粒度弧相容算法的基本框架 AC3_frame_ARR,该改进框架可用于改进所有粗粒度弧相容算法.实验结果表明,经过 AC3_frame_ARR 改进后的算法最多可以节省 80%的修正检查次数和 40%的求解耗时.

关键词: 约束满足问题;维持弧相容;粗粒度算法;修正检查

中图法分类号: TP18 **文献标识码**: A

* 基金项目: 国家自然科学基金(60773097, 60873148, 60973089); 吉林省自然科学基金(201101039, 20071106, 20080107); 国家教育部博士点专项基金(20100061110031)

收稿时间: 2011-06-29; 定稿时间: 2011-10-08

约束满足问题(constraint satisfaction problem,简称 CSP)[1]在人工智能领域有着广泛的应用,许多实际问题可以应用约束满足问题建模,如配置问题、调度问题等.约束满足问题可以分为二元约束满足问题和多元约束满足问题,其中,多元约束满足问题可以转化为等价的二元约束满足问题,因此,二元约束满足问题一直是该领域研究的热点.寻找一个约束满足问题的解是 NP-难问题[1].尽管如此,目前仍存在很多优秀的 CSP 求解算法.在众多求解算法中,基于回溯思想的 BT 算法[2]是一种完备的核心算法.但基本的 BT 算法效率较低,为了提高 BT 算法的求解效率,通常要在 BT 搜索过程中嵌入约束传播技术[3].约束传播技术主要利用约束与变量之间的制约关系,删除那些一定不在解中的变量值,缩小搜索空间.约束传播技术分为很多种,如弧相容(arc consistency,简称 AC)[4,5]、singleton 弧相容(简称 SAC)[6,7]、路径相容(path consistency,简称 PC)[3]、*k*-相容[8]等等.在已有的约束传播技术中,弧相容(AC)在求解中应用得最为成功,目前已有很多优秀的弧相容算法,如 AC3[4],AC4[5],AC6[9],AC3.1[10],AC3rm[11]等.弧相容算法根据其约束传播类型分为粗粒度算法和细粒度算法[10]两类,其中,粗粒度算法是在弧上进行约束传播,而细粒度算法是在变量值之间进行约束传播.在 BT 算法中,通过 AC 算法进行约束传播的约束求解算法称为维持弧相容算法(maintaining arc consistency,简称 MAC)[12].MAC 算法是目前求解难解的约束满足问题应用最广泛的算法.MAC 算法的效率主要受两个因素影响:一是所采用的 AC 算法本身的效率,另一个是在求解过程中需要维持的数据结构的复杂程度.细粒度算法虽然在执行单独的弧相容过程中效率优于粗粒度算法,但其本身使用精细的数据结构,使得在求解中维持这些数据结构消耗了更多的时间.因此在求解中,粗粒度算法的应用更加广泛.AC3 是 1977 年由 Mackworth 提出的经典的粗粒度算法[4],它给出了粗粒度算法的框架,即粗粒度算法主要由基本框架和修正检查两部分构成.AC3 的最坏时间复杂度为 $O(ed^3)$.为了优化时间复杂度以及减少弧相容过程中执行的约束检查次数,在 AC3 提出后的很长一段时间内,国内外学者们将研究重点放在了细粒度算法中,因为细粒度算法,如 AC4,AC6,AC7[13]等具有最坏时间复杂度 $O(ed^2)$,尤其是 AC7 通过引入约束的双向支持的特性,大幅度减少了约束检查的次数.但是由于其本身的数据结构较为复杂,因此在求解中的应用不是很广泛.2001 年,Bessiere 提出了 AC3.1,它是 AC3 的优化版本,通过记录值在弧上最后的一个支持改进了 AC3 算法的修正检查过程,具有最坏时间复杂度 $O(ed^2)$.其数据结构比细粒度算法简单,因此求解效率高于细粒度算法.至此,人们又将研究重点转移到粗粒度算法上.在 AC3.1 的基础上,Lecoutre 于 2003 年将约束双向支持的特性引入到粗粒度算法中,提出了 AC3.2[14]和 AC3.3[14],改进了 AC3.1 的 revise 过程,AC3.2 和 AC3.3 比 AC3.1 执行更少的约束检查,但最坏时间复杂度未得到改进.2007 年,Lecoutre 又提出了 AC3rm,利用约束的双向支持特性实现了残留支持的思想.AC3rm 虽然最坏时间复杂度为 $O(ed^3)$,但其实际求解效率优于之前的其他算法,并且执行的约束检查次数也较少.这些具有代表性的 AC3 的改进算法都只是改进了算法的修正检查过程,并没有改进算法的基本框架.执行一次修正检查的最坏时间复杂度是 $O(d^2)$.因此,减少修正检查的次数可以提高 AC 算法的效率.2005 年,Mehta 和 Dongen 提出了一种修正检查条件(revision condition,简称 RC)[15],可以减少求解初期阶段无效的修正检查.除此以外,Wallace[16]和 Dongen[17]的工作在从待传播队列中选择下一条执行修正检查的弧时,使用了一些启发式规则,对更有可能导致失败的弧优先执行修正检查.以上两种方法都对粗粒度算法的基本框架进行了改进.

本文分析了粗粒度求解算法的执行过程,发现其中对于指向已赋值的变量的弧存在无效的修正检查.证明了这类修正检查是冗余的.这种冗余的修正检查在弧相容预处理过程中很少出现,而在求解过程中出现得较为频繁.我们提出一种方法 avoid_redundant_revisions(ARR),避免了这类冗余的修正检查,给出改进后的粗粒度算法的基本框架 AC3_frame_ARR,这一修正框架可以用于改进目前所有的粗粒度 MAC 算法.除此以外,ARR 和修正检查条件(RC)不同,避免的不是同一类型的修正检查,因此,ARR 可以和 RC 结合使用.最后,我们将这一方法应用到目前国际上流行的粗粒度求解算法 MAC3rm 中,实验结果显示,经过 ARR 单独改进后的算法最多可节省 80%的修正检查次数,最高可提高 40%的求解效率.ARR 和 RC 结合使用,可以更好地改进 MAC3rm.

1 背景知识

定义 1(约束满足问题 constraint satisfaction problem(CSP)). 一个约束满足问题 P 由 3 部分组成:$P=\langle X,D,$

C⟩.其中,$X=\{x_1,x_2,\ldots,x_n\}$是一个有限的变量集合;$D=\{D_1,D_2,\ldots,D_n\}$,其中,$D_i\in D$ 对应 $x_i\in X$,是变量 x_i 的有限值域;$C=\{c_1,c_2,\ldots,c_k\}$是一个有限约束集合,其中,任意 $c_j\in C$ 表示变量取值之间的制约关系.

如果 C 中所有约束包含的变量个数都小于等于 2,称 P 为二元约束满足问题.本文主要讨论二元约束满足问题.一个约束满足问题的解是为每个变量在其值域中选择一个值构成集合 S,使得 S 中的所有变量取值满足 C 中所有约束.我们用 P 表示一个约束满足问题,S 表示 P 的一个解,$P.X$ 表示 P 的变量集,$P.C$ 表示 P 的约束集,$X(c)$表示 c 中包含所有变量的集合,(x,a)表示 x 的取值为 a.

在二元约束满足问题中,如果存在一条约束 c_{xy},则(x,c_{xy})称为一条弧,表示的是约束 c_{xy} 对 x 取值的制约.一条弧由 3 部分构成,其中,x 为这条弧指向的变量,y 是这条弧出发的变量,c_{xy} 是 x 和 y 之间的约束.弧的描述(x,c_{xy})只体现了 x 和 c_{xy},y 可以由这二者隐含指定.在二元 CSP 中,一条约束对应两条弧.

性质 1(约束的双向支持性[13]). 给定一条约束 c_{xy},如果(x,a),(y,b)满足 c_{xy},则(y,b)是(x,a)在弧(x,c_{xy})上的一个支持,(x,a)也是(y,b)在弧(y,c_{xy})上的一个支持.

定义 2(弧相容,arc consistency(AC))[4]. 给定一个二元约束满足问题 P,对于 P 中的某一条弧(x,c_{xy}),如果 x 值域中的每一个值 a 都能在 y 的值域中找到一个值 b,使得(a,b)满足 c_{xy},那么称弧(x,c_{xy})是弧相容的.一个 CSP 是弧相容的,当且仅当它的每一条弧都是弧相容的.

对于问题 P,在进行弧相容检查时,要将那些在相邻弧上找不到支持的值删除.如果检查过程中发现某一变量的值域中的值被全部删除,则此问题 P 无解,返回弧相容检查失败.

维持弧相容(maintaining arc consistency,简称 MAC)算法[12]是一种高效的求解 CSP 问题的回溯搜索算法.MAC 算法是在 BT 算法框架下嵌入 AC 算法,在搜索过程中维持弧相容的状态,每次变量赋值后利用 AC 算法进行约束传播,如果得到一个弧相容的状态,则赋值成功;否则,赋值失败,选择下一个赋值或者发生回溯.目前流行的 MAC 算法是由 Lecoutre 于 2007 年提出的 MAC3rm 算法,MAC3rm 是一种粗粒度算法,由于篇幅有限,这里不介绍 MAC3rm 的算法描述,有兴趣的读者可以参考文献[11].除了在求解过程中利用 AC 算法进行约束传播外,在求解前还要通过 AC 算法执行预处理.AC3 给出了粗粒度算法的经典框架[4],这一框架一直沿用至今.

图 1 所示为粗粒度算法的基本框架,常见的粗粒度算法,如 AC3,AC3.1,AC3.3,AC3rm 等都是基于这一框架的.其中,Q 是待传播队列,保存所有需要执行修正检查的弧;initialize Q 是初始化待传播队列,如果 AC 算法用于求解前预处理阶段,则要将问题中所有的弧加入 Q 来实现 initialize Q.在弧相容检查过程中,如果有变量的值域改变,则将从这个变量发出的所有弧加入 Q,对这些弧重新执行修正检查.如果检查过程中发现某一个变量值域为空,则返回弧相容,检查失败;否则,当 Q 为空时,弧相容检查成功.图 2 中的 *revise* 函数是粗粒度算法的修正检查过程,其中,D_x 是 x 的有限值域,$Revise(x,c)$是为 x 值域中所有未被删除的值寻找在这条弧上的支持.下面提到的 AC 算法都是指粗粒度弧相容算法.

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

Fig.1 Basic frame for coarse-grained AC algorithms

图 1 粗粒度 AC 算法基本框架

```
procedure Revise(x,c)
 begin
    change←false;
        for each v∈D_x do
            If v has no support on c then
                remove v from D_x;
                change←true;
    return change;
    end
```

Fig.2 Revision for coarse-grained AC algorithms

图 2 粗粒度 AC 算法的修正检查

2 冗余检查

在 MAC 算法求解开始之前,要通过 AC 算法执行一次弧相容预处理.如果预处理失败,则问题无解;否则,开

始回溯搜索.因此在开始回溯搜索之前,问题是一个弧相容的状态.而每次赋值后,都要通过 AC 算法执行约束传播,如果 AC 算法执行成功,则进行下一次赋值;否则,发生回溯.可见,在 MAC 算法搜索过程中,每次赋值之前,问题一定是弧相容的状态.为变量 x 赋值 a,相当于将 x 值域中除 a 以外的其他值全部删除.因此每次赋值后,只有当前赋值变量的值域发生了改变.所以,为变量 x 赋值后,只需用从 x 发出的所有弧来初始化待传播队列 Q,然后执行 AC 算法进行约束传播.

图 3 和图 4 中,椭圆表示对应变量的初始值域,值之间的连线表示这两个值满足这条约束.在求解前,弧相容预处理的结果如图 3 所示.假设求解过程中我们首先将 x 赋值为 1,要将(y,c_{xy})加入 Q 后执行 AC 算法,执行 $revise(y,c_{xy})$后,y 的值域没有改变,因此没有将其他弧加入 Q.此时,Q 为空,没有变量值域为空,AC 算法停止,$(x,1)$赋值成功.下一步赋值$(z,1)$,赋值后将(y,c_{yz})加入 Q 后执行 AC 算法,执行 $revise(y,c_{yz})$后,D_y 中的 3 被删除,值域改变,因此将从 y 发出且约束不是 c_{yz} 的所有弧加入 Q.此时 Q 中包含(x,c_{xy}),各变量值域和值之间的支持如图 4 所示.下次执行 $revise(x,c_{xy})$时,就是一次无效的修正检查.这是由于在为 x 赋值 1 后,执行弧相容算法成功,$(x,1)$是 y 值域中所有未被删除的值在(y,c_{xy})上的支持.由于约束的双向支持性,y 值域中所有未删除的值也一定都是$(x,1)$在(x,c_{xy})上的支持.因此,只要 y 的值域不空,$(x,1)$在弧(x,c_{xy})上就一定存在支持,弧相容算法只要确定值$(x,1)$在弧(x,c_{xy})上存在支持即可,不必知道到底是哪个值支持.所以,$revise(x,c_{xy})$就是无效的修正检查.但这种情况只适用于 x 的值域大小为 1 的情况,如果 x 的值域大小大于 2,则当 y 的值域发生改变时,无法确认刚被删除的值是 x 值域中哪个值的支持,因此需要执行 $revise(x,c_{xy})$,重新为 x 值域中所有值寻找支持.

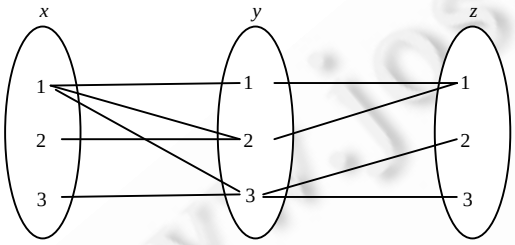

Fig.3 Domains after AC pre-processing

图 3 弧相容预处理后的值域

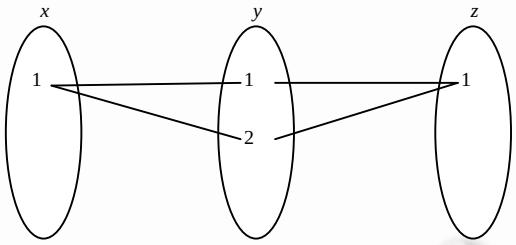

Fig.4 Domains after x and z have been assigned

图 4 x,z 赋值成功后的值域

定理 1. 在粗粒度 MAC 算法求解过程中,对指向一个已赋值变量的弧执行的修正检查是冗余的.

证明:在粗粒度 AC 算法执行过程中,对一条弧执行修正检查的前提是这条弧在待传播队列中,而将一条弧加入到待传播队列的前提条件是这条弧的出发变量的值域发生了改变.在求解过程中,每次为变量 x 赋值后要将从 x 出发的所有弧加入到待传播队列.在将这些从 x 出发的弧执行完修正检查后,如果没有变量值域为空,则说明 x 的当前赋值是那些从 x 发出的弧指向变量的值域中剩余所有值在对应弧上的支持.由于约束具有双向支持性,因此,所有指向 x 的弧的出发变量值域中剩余的所有值也都是 x 的当前赋值在对应弧上的支持.因此,当指向已赋值变量 x 的弧的出发变量值域发生改变时,如果出发变量值域不为空,则 x 的当前赋值一定在对应弧上存在支持;如果出发变量值域为空,则直接返回弧相容失败,因此无需对指向已赋值变量 x 的弧执行修正检查. □

根据定理 1,我们应该避免对那些指向已赋值变量的弧执行修正检查.对弧执行修正检查的前提条件是这条弧要被加入到待传播队列 Q 中,因此,在将弧加入到待传播队列 Q 中之前,先判断这条弧指向的变量是否已赋值:如果已赋值,则不加入 Q;否则,加入 Q.这样,就可以避免这类冗余的修正检查.我们称这一方法为 avoid_redundant_revisions,简称 ARR.图 5 给出了改进后的粗粒度算法框架.

```
AC3_frame_ARR
 begin
      Q←{(x_i,c)|c∈C,x_i∈X(c)};
      while Q is not empty do
           select and remove the arc(x_i,c) from Q;
                if Revise(x_i,c) then
                     if D_i=∅ then return fail;
                     else for each c′ such that x_i,x_j∈X(c′)
                          if x_j has not been assigned and c′≠c then
                               Q←Q∪(x_j,c′);
      return success;
 end
```

Fig.5 Improved basic frame for coarse-grained AC algorithms

图 5 改进的粗粒度 AC 算法基本框架

3 相关工作讨论

2005 年的 IJCAI 会议上,Mehta 和 Dongen 提出了修正检查条件 revision condition(RC),通过减少修正检查次数改进了粗粒度算法的基本框架.本文提出的 ARR 和 RC 方法类似,都可以避免那些无效的修正检查,但二者避免的是两种不同类型的修正检查.RC 方法通过为 x 的值域中所有值记录其在(x,c_{xy})上的支持个数,进而得到 D_x 中所有值在(x,c_{xy})上最少的支持个数 Min-*Support-Count*.当需要将弧(x,c_{xy})加入到待传播队列中时,如果当前 y 值域中被删除的值 *Current-Delete-Count* 小于 Min-*Support-Count*,则说明当前 x 值域中所有的值都可以在 y 的值域中找到至少一个支持.因此,这条弧(x,c_{xy})当前无需执行修正检查,不加入到待传播队列.只有那些 Min-*Support-Count* 小于等于 *Current-Delete-Count* 的弧才需要执行修正检查,被加入到待传播队列.RC 方法通过这一判断减少了粗粒度算法在执行过程中的 *revise* 调用次数,但 RC 方法的缺点在于,初始时需要计算每条弧上的最小支持个数,在问题容易解决的情况下,可能计算每条弧最小支持的耗时就已经超过求解耗时.RC 在回溯搜索过程刚开始时效果较明显,因为这一阶段已赋值变量较少,其他未赋值变量的值域中被删除的值也较少,因此不满足修正检查条件的弧较多,可以避免无效的修正检查.ARR 是在已赋值变量较多的情况下更有效,因为我们避免的就是对已赋值变量执行修正检查.

ARR 在求解过程中的应用效果比在弧相容预处理过程中要明显,因为在执行弧相容预处理过程中,变量值域大小为 1 的情况很少,而在求解过程中,通过变量赋值导致的变量值域大小为 1 的情况较多,可以明显减少冗余的修正检查次数.ARR 和 RC 都是改进粗粒度算法的框架,可以应用于所有的粗粒度算法,并且二者可以结合使用.除此以外,我们于 2011 年提出的一种基于环切割的方法 Cycle_Cut_Search(简称 CCS)[18]改进了 MAC3rm 算法,将 MAC3rm 的回溯搜索过程分为两个阶段,其中,第 1 阶段是通过 MAC3rm 为环切割集中变量赋值,第 2 阶段通过无回溯树搜索算法为剩余变量赋值.可见,CCS 的第 1 阶段也是粗粒度 MAC 算法,因此,ARR 同样可以用于改进 CCS 的第 1 阶段.

4 实验结果

为测试 ARR 方法对粗粒度算法效率的改进,我们共采用 4 种不同的算法进行测试.基础算法采用目前国际上流行的粗粒度算法 Mac3rm;Mac3rm_RC 是在 Mac3rm 的基础上增加了 RC 技术的改进算法;Mac3rm_ARR 是用 ARR 改进 MAC3rm,也就是用 AC3_frame_ARR 框架改进 Mac3rm 基础框架后的新算法;Mac3rm_ARR_RC 是将 RC 和 ARR 结合使用改进 MAC3rm 后得到的新算法.测试用例采用随机问题和标准库测试用例 benchmark,测试的数据指标为 CPU 耗时和修正检查次数.由于变量赋值顺序的启发式规则对 MAC 算法的求解效率影响很大,我们采用目前最流行的 Dom/Wdeg 启发式规则[19].测试环境为 Intel Core2 Duo CPU E7400 2.8GHz/1G RAM,JDK 1.6.

4.1 随机问题测试

我们选择二元随机约束满足问题经典模型 Model B[20]进行测试.Model B 问题模型由 4 个参数控制:$\langle n,m,p_1,$

p_2⟩,其中,n 是问题中的变量个数;m 是每个变量的值域大小,其中每个变量的值域大小都是相同的;p_1 表示约束密度,即在所有变量之间随机选择 $p_1\times n\times(n-1)/2$ 条约束;p_2 表示约束松紧度,对于每条约束,随机选择 $p_2\times m\times m$ 个值对作为满足这条约束的值对.我们选取 n=30,m=30,p_1=0.5 的问题进行测试,p_2 从 0.1~0.9,每组连续测试 50 次求平均值,其中,p_2 从 0.1~0.54 和 p_2 从 0.66~0.9 的随机问题都是容易解的问题,我们没有给出这两个区间的测试结果,图 6 和图 7 给出了难解问题,即 p_2 在 0.55~0.65 区间内耗时和修正检查次数的比较情况.

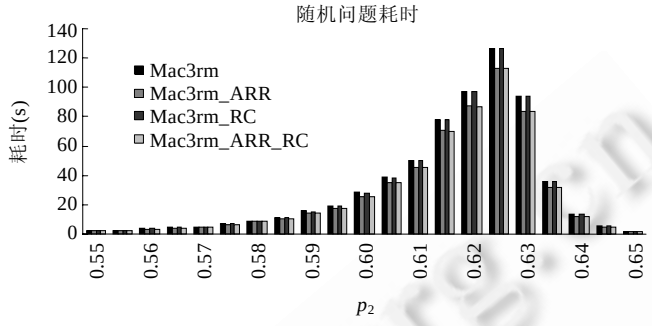

Fig.6 Time results on random instances

图 6 随机问题耗时结果

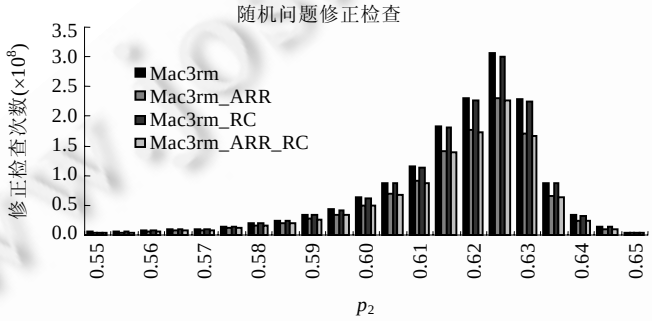

Fig.7 Revision results on random instances

图 7 随机问题修正检查结果

在图 6 和图 7 的每组问题结果中,从左到右 4 个矩形依次代表 Mac3rm,Mac3rm_ARR,Mac3rm_RC,Mac3rm_ARR_RC 的测试结果.数据显示,Mac3rm_ARR 和 Mac3rm_ARR_RC 对 Mac3rm 的改进效果较为明显,而 Mac3rm_RC 几乎对 Mac3rm 没有改进.Mac3rm_ARR 大约提高 10%的求解效率,节省大约 20%的修正检查.我们的原始数据显示,Mac3rm_ARR 减少的修正检查和 Mac3rm_RC 减少的修正检查次数之和,恰好接近于 Mac3rm_ARR_ RC 减少的修正检查次数,这也验证了之前提到的,二者避免的是不同类型的修正检查.

4.2 Benchmark测试

Benchmark 是目前国际通用的测试 CSP 求解器效率的标准库测试用例,本文选取的测试用例源文件全部来自 2005 年国际 CP 程序竞赛(http://cpai.ucc.ie/05/Benchmarks.html).我们在其中选取两组较难解的问题进行测试.表 1 给出了 frb 问题[21]的测试结果.2005 年 CP 竞赛提供的 frb 测试用例每组问题中包含 5 个子问题,每组子问题的规模相同,难度接近.因此,表 1 给出的是每组问题的平均测试结果.鸽巢问题是一组实际应用问题,鸽巢问题中两个子问题相差一个变量可能导致耗时相差 10 倍以上.为避免规模较大问题对规模较小问题的测试结果产生影响,我们在表 2 中分别给出了 5 个子问题的测试结果.

Table 1 Test results on frb instances

表 1 frb 问题测试结果

测试指标	CPU time				Revision			
测试算法	Mac3rm	Mac3rm ARR	Mac3rm RC	Mac3rm ARR_RC	Mac3rm	Mac3rm ARR	Mac3rm RC	Mac3rm ARR_RC
frb30-15	0.29	0.27	0.27	0.25	964 346	775 695	834 789	663 933
frb35-17	3.57	3.27	3.29	3.01	11 553 087	9 229 506	9 954 083	7 862 752
frb40-19	17.95	16.30	16.99	15.43	55 743 215	43 866 544	49 809 579	38 729 393
frb45-21	245.67	222.67	237.86	214.70	737 302 771	573 016 169	677 728 915	521 052 867

Table 2 Test results on pigeon instances

表 2 鸽巢问题测试结果

测试指标	CPU time				Revision			
测试算法	Mac3rm	Mac3rm ARR	Mac3rm RC	Mac3rm ARR_RC	Mac3rm	Mac3rm ARR	Mac3rm RC	Mac3rm ARR_RC
pigeons9	0.28	0.17	0.21	0.14	1 574 445	465 713	793 354	219 192
pigeons10	3.01	1.70	2.14	1.44	15 861 760	4 181 532	7 991 611	1 972 809
pigeons11	34	19	25	16	1.768E+08	4.194E+07	8.886E+07	1.973E+07
pigeons12	419	227	304	195	2.136E+09	4.614E+08	1.071E+09	2.170E+08
pigeons13	5 580	2 966	4 080	2 577	2.798E+10	5.540E+09	1.402E+10	2.604E+09

frb 问题的测试结果显示:Mac3rm_ARR 和 Mac3rm_RC 都改进了接近 10%的求解效率;Mac3rm_ARR 的效率略高于 Mac3rm_RC,二者结合使用可以改进 10%以上的求解效率.鸽巢问题的测试结果显示:Mac3rm_ARR 改进了大约 40%的求解效率;Mac3rm_RC 改进了大约 20%的求解效率;Mac3rm_ARR 的效率明显高于 Mac3rm_RC,二者结合使用可以改进 50%以上的求解效率.Mac3rm_ARR 和 Mac3rm_RC 都节省了一些修正检查,但 Mac3rm_ARR 比 Mac3rm_RC 节省更多的修正检查.在鸽巢问题中,Mac3rm_ARR 最多节省了超过 80%的修正检查,这也说明,在某些问题的求解过程中,存在着大量的冗余修正检查.frb 问题和鸽巢问题的修正检查次数统计结果又一次验证了二者避免的是不同类型的修正检查,Mac3rm_ARR 减少的修正检查和 Mac3rm_RC 减少的修正检查次数之和,接近于 Mac3rm_ARR_RC 减少的修正检查次数.

5 结 论

修正检查是粗粒度弧相容算法必不可少的核心部分,执行一次修正检查的最坏时间复杂度为 $O(d^2)$,因此,避免无效的修正检查可以提高算法的求解效率.粗粒度算法的研究一直集中在对修正检查部分的改进上,对于算法框架的改进工作则较少.本文研究了粗粒度维持弧相容求解算法的执行过程,发现其中存在一些无效的修正检查,这类修正检查主要是对指向已赋值变量的弧执行的修正检查.我们证明了这类修正检查是冗余的,给出一种方法 ARR 避免这类冗余的修正检查,并且给出修正后的粗粒度算法框架 AC3_frame_ARR,这一框架可以用于改进目前所有基于 AC3 的粗粒度算法.除此以外,ARR 还可以和另一种改进方法 RC 相结合,二者避免了两种不同类型的冗余修正检查.我们的实验结果显示,ARR 除了自身可以改进目前流行的粗粒度 MAC 求解算法以外,与 RC 方法结合共同改进粗粒度 MAC 算法的效果更好.

References:

[1] Freuder EC, Mackworth AK. Constraint satisfaction: An emerging paradigm. Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam: Elservier, 2006. 13–27. [doi: 10.1016/S1574-6526(06)80006-4]

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

[5] Mohr R, Henderson TC. Arc and path consistency revised. Artificial Intelligence, 1986,28(2):225−233. [doi: 10.1016/0004-

3702(86)90083-4]
[6] Debruyne R, Bessière C. Some practicable filtering techniques for the constraint satisfaction problem. In: Pollack ME, ed. Proc. of the IJCAI'97. Morgan Kaufmann Publishers, 1997. 412−417. http://www.emn.fr/z-info/rdebruyn/ijcai97.pdf
[7] Bessière C, Cardon S, Debruyne R, Lecoutre C. Efficient algorithms for singleton arc consistency. Constraints, 2011,16(1):25−53. [doi: 10.1007/s10601-009-9080-5]
[8] Freuder EC. A sufficient condition for backtrack-free search. Journal of ACM, 1982,29(1):24−32. [doi: 10.1145/322290.322292]
[9] Bessière C. Arc-Consistency and arc-consistency again. Artificial Intelligence, 1994,65(1):179−190. [doi: 10.1016/0004-3702(94)90041-8]
[10] 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]
[11] Lecoutre C, Hemery F. A study of residual supports in arc consistency. In: Veloso MM, ed. Proc. of the IJCAI 2007. AAAI Press, 2007. 125−130. https://www.aaai.org/Papers/IJCAI/2007/IJCAI07-018.pdf
[12] Sabin D, Freuder EC. Contradicting conventional wisdom in constraint satisfaction. In: Cohn AG, ed. Proc. of the 11th European Conf. on Artificial Intelligence. Amsterdam: John Wiley & Sons, 1994. 125−129.
[13] Bessière C, Freuder EC, Régin JC. Using constraint metaknowledge to reduce arc consistency computation. Artificial Intelligence, 1999,107(1):125−148. [doi: 10.1016/S0004-3702(98)00105-2]
[14] Lecoutre C, Boussemart F, Hemery F. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In: Francesca R, ed. Proc. of the CP 2003. Springer-Verlag, 2003. 480−494. http://www.cril.univ-artois.fr/~lecoutre/research/publications/2003/CP2003.pdf
[15] Mehta D, van Dongen MRC. Reducing checks and revisions in coarse-grained MAC algorithms. In: Kaelbling LP, ed. Proc. of the IJCAI 2005. Professional Book Center, 2005. 236−241. http://ijcai.org/papers/0820.pdf
[16] Wallace RJ, Freuder EC. Ordering heuristics for arc consistency algorithms. In: Proc. of the 9th Canadian Conf. on Artificial Intelligence. Vancouver, 1992. 163−169. http://4c.ucc.ie/web/upload/publications/misc/wallace92ordering.pdf
[17] van Dongen MRC. Saving support checks does not always save time. Artificial Intelligence Review, 2004,21(3-4):317−334. [doi: 10.1023/B:AIRE.0000007389.67810.17]
[18] Li ZS, Li HB, Zhang YG, Wang ZW. An approach of solving constraint satisfaction problem based on cycle-cut. Chinese Journal of Computers, 2011,34(8):1528−1535 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01528]
[19] Boussemart F, Hemery F, Lecoutre C, Sais L. Boosting systematic search by weighting constraints. In: Saitta L, ed. Proc. of the 16th European Conf. on Artificial Intelligence. IOS Press, 2004. 146−150. http://www.cril.univ-artois.fr/~lecoutre/research/publications/2004/ECAI2004.pdf
[20] Smith BM, Dyer ME. Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence, 1996,81(1):155−181. [doi: 10.1016/0004-3702(95)00052-6]
[21] Xu K, Li W. Exact phase transitions in random constraint satisfaction problems. Journal of Artificial Intelligence Research, 2000,12(3-4):93−103.

附中文参考文献:

[18] 李占山,李宏博,张永刚,王孜文.一种基于环切割的约束满足问题求解算法.计算机学报,2011,34(8):1528−1535. [doi: 10.3724/SP.J.1016.2011.01528]

李宏博(1985－),男,吉林公主岭人,博士生,主要研究领域为约束问题求解.

王涛(1969－),女,讲师,主要研究领域为约束问题求解.

李占山(1966－),男,博士,教授,CCF 会员,主要研究领域为基于模型的诊断,智能规划与调度,约束问题求解.