```
        if dom(x_i)=∅ then
            return fail;
    return success;
end
procedure searchPCsupport(x_i,a,c_ij)
begin
    for each value b∈dom(x_j) do
        if (x_j,b) is an AC-support of (x_i,a) on c_ij then
            if searchPCwitness(x_i,a,x_j,b) then
                return true;
    return false;
end
procedure searchPCwitness(x_i,a,x_j,b)
begin
    for each variable x_k constrained with both x_i and x_j do
        if checkPath(x_i,x_j,x_k) then
            findPCwit←false;
            for each value c∈dom(x_k) do
                if (a,c) satisfies c_ik and (b,c) satisfies c_jk then
                    findPCwit←true;
                    break;
            if ¬findPCwit then
                return false;
    return true;
end
```

算法 1 中,Q 是传播队列,保存所有需要执行检查的变量,如果该局部相容算法用于求解前预处理阶段,在初始化 Q 时,要将 X 中所有的变量加入 Q.如果是在回溯搜索过程中,则初始化 Q 时只将当前赋值变量加入到 Q.在约束传播过程中,每次弹出 Q 中一个变量 x_j,然后对所有与 x_j 相连的变量 x_i 进行检查,为 $dom(x_i)$中每个值寻找 PC-support,并将没有 PC-support 的值删除.任意变量 x_i 的值域如果发生改变,都将被加入 Q.如果检查过程中发现某一个变量值域为空,则返回 maxRPC 检查失败;否则,当 Q 为空时,maxRPC 检查成功.

searchPCsupport 函数在为一个变量值(x_i,a)在约束 c_{ij} 寻找 PC-support 时,首先找到它的一个 *AC-support*(x_j,b),然后调用 *searchPCwitness* 函数检查(x_i,a)与(x_j,b)是否为路径相容的:若二者是路径相容的,(x_j,b)就是(x_i,a)的 PC-support;否则,尝试下一个 AC-support.其中,*checkPath* 函数是用来判断是否需要在某条路径检查 PC-witness 的存在.当算法为 maxRPC 算法时,*checkPath* 函数返回值恒为真,则每次都要检查 PC-witness.当算法为 AC 算法时,*checkPath* 函数返回值恒为假,每次都无需检查 PC-witness.实际上,该函数只有后面介绍的 PmaxRPC 算法使用,对于 PmaxRPC 算法,*checkPath* 函数返回值可能为真,也可能为假.

2 概率最大受限路径相容

给定 3 个变量 x_i,x_j,x_k 和 3 条约束 c_{ij},c_{ik},c_{jk},(x_j,b)是(x_i,a)在约束 c_{ij} 上的 AC-support,假设(x_i,a)在约束 c_{ik} 上存在 m 个 AC-support,(x_j,b)在约束 c_{jk} 上存在 n 个 AC-support,那么从 PC-support 的定义可知:(x_j,b)和(x_i,a)在 x_k 上存在 PC-witness 的一个充要条件就是(x_j,b)的 m 个 AC-support 和(x_i,a)的 n 个 AC-support 中,至少有一个是重复的.

例:$dom(x_i)=dom(x_j)=dom(x_k)=\{1,2,3,4\}$,约束 $c_{ij}=\{(1,2),(1,3),(2,2),(3,4)\}$,约束 $c_{ik}=\{(1,1),(1,3),(3,1),(4,2),(4,4)\}$,约束 $c_{jk}=\{(1,2),(2,2),(3,3),(4,4)\}$,这 3 条约束中分别记录了满足它们的元组.假设只有 x_k 是和 x_i,x_j 之间都存在约束的变量,当为$(x_i,1)$在约束 c_{ij} 上寻找 PC-support 时,我们首先要找到它的第 1 个 AC-support,因为元组(1,2)满足约束 c_{ij},所以$(x_j,2)$是它的第 1 个 AC-support.然后,我们要在 $dom(x_k)$中找到一个值 c,使得$(1,c)$满足 c_{ik},并且$(2,c)$满足 c_{jk},遍历 $dom(x_k)$后我们发现,没有这样一个值 c,因此不存在对应的 PC-witness,此时,$(x_j,2)$只是$(x_i,1)$在约束 c_{ij} 上的 AC-support,而不是 PC-support,所以要尝试下一个 AC-support.然后,我们找到$(x_i,1)$在约束 c_{ij} 上的第 2 个 *AC-support*$(x_j,3)$.搜索 $dom(x_k)$后可以发现,$(x_k,3)$可以作为 PC-witness,因为(1,3)满足约束 c_{ik} 并且(3,3)满足约束 c_{jk},因此,$(x_j,3)$是$(x_i,1)$在约束 c_{ij} 上的 PC-support.此例中,$(x_i,1)$在约束 c_{ik} 上有$(x_k,1)$和$(x_k,3)$这 2 个 AC- support,而$(x_j,2)$在约束 c_{jk} 上有$(x_k,2)$这 1 个 AC-support,并且这些 AC-support 中没有重复的,因此,$(x_j,2)$不是$(x_i,1)$的 PC-support.$(x_j,3)$在约束 c_{jk} 上虽然只有$(x_k,3)$这 1 个 AC-support,但是$(x_i,1)$在约束 c_{ik} 上也有 1 个 AC-support 是$(x_k,3)$,此时,$(x_k,3)$就成为了二者的 PC-witness,因此,$(x_j,3)$是$(x_i,1)$的 PC-support.

注意:在这个例子中,如果除 x_k 以外还有其他变量 x_p 与 x_i,x_j 之间都存在约束,那$(x_j,3)$必须也在 $dom(x_p)$中找到 PC-witness,它才能成为$(x_i,1)$在约束 c_{ij} 上的 PC-support.以下 $C(n,m)$表示从 n 个不同元素中取出 m 个元素的组合数.

定理 1. 给定变量 x_i,x_j 和 x_k,约束 c_{ij},c_{ik} 和 c_{jk},(x_j,b)是(x_i,a)在约束 c_{ij} 上的 AC-support,并且(x_i,a)在约束 c_{ik} 上存在 m 个 AC-support,(x_j,b)在约束 c_{jk} 上存在 n 个 AC-support,$|dom(x_k)|=d$,那么(x_i,a)与(x_j,b)在 $dom(x_k)$中存在 PC-witness 的概率为 $1-\frac{C(d-m,n)}{C(d,n)}$.

证明:$dom(x_k)$中有 d 个值,在 d 个值中存在 m 个(x_i,a)的 AC-support,那么这 m 个 AC-support 可能的组合方式个数为 $C(d,m)$;同理,(x_j,b)的 n 个 AC-support 的可能组合方式个数为 $C(d,n)$.因此,二者所有 AC-support 的组合方式共有 $C(d,m)\times C(d,n)$种.在(x_i,a)的 m 个 AC-support 已知的情况下,(x_j,b)的 n 个 AC-support 与前者的 m 个 AC-support 中没有重复的可能组合方式有 $C(d-m,n)$种,因此二者的 AC-support 中没有重复的可能组合方式共有 $C(d,m)\times C(d-m,n)$种.没有重复的 AC-support 即是二者之间不存在 PC-witness.

由以上论述可知:二者的 AC- support 中没有任何两个相同的概率为 $\frac{C(d,m)\times C(d-m,n)}{C(d,m)\times C(d,n)}=\frac{C(d-m,n)}{C(d,n)}$,所以二者的 AC-support 中至少有一个相同的概率,即,PC-witness 存在的概率为 $1-\frac{C(d-m,n)}{C(d,n)}$. □

推论 1. 当 $m+n>d$ 时,PC-witness 一定存在.

由推论 1 可知:当 $d=1$ 时,若 $m>0,n>0$,则 PC-witness 一定存在.若 $m=0$,则(x_i,a)在约束 c_{ik} 不存在 AC-support,因此它不是弧相容的,我们将它从 $dom(x_i)$中删除;同理,若 $n=0$,(x_j,b)将由于弧相容检查失败而被我们从 $dom(x_j)$中删除.

定理 2. 给定变量 x_i,x_j 和 x_k,约束 c_{ij},c_{ik} 和 c_{jk},(x_j,b)是(x_i,a)在约束 c_{ij} 上的 AC-support,当$|dom(x_k)|=1$ 时,若(x_j,b)与(x_i,a)没有因为弧相容检查失败而被删除,则(x_j,b)与(x_i,a)在 x_k 上存在 PC-witness.

证明:maxRPC 比 AC 的检查更严格,因此,若(x_j,b)没有因为弧相容检查失败而被删除,则(x_j,b)在所有包含 x_j 的约束上都存在至少一个 AC-support.当$|dom(x_k)|=1$ 时,假设此时 $dom(x_k)=\{c\}$,(x_j,b)在约束 c_{jk} 上的 AC-support 一定是(x_k,c),此时,(b,c)一定满足约束 c_{jk};同理,(a,c)一定满足约束 c_{ik}.因此,(x_k,c)就是(x_j,b)与(x_i,a)在 x_k 上的 PC-witness. □

由定理 2 可知:当$|dom(x_k)|=1$ 时,我们可以避免在 x_k 上寻找 PC-witness 的过程.根据定理 1、定理 2 和推论 1,在已知 m,n 和 d 的情况下,我们可以在 maxRPC 算法中避免一些冗余的 PC-witness 检查.但在实际应用中,只有 d 可以用较低的计算代价得到,而 m 和 n 则分别需要$|dom(x_k)|$次约束检查才能得到,其耗时甚至可能超过寻找 PC-witness 的过程.因此在回溯搜索过程中,每次计算准确的 m 和 n 用来估计 PC-witness 存在的概率是不实际的.但是我们可以利用约束的解密度来估计 m 和 n.给定约束 c_{ij},变量 x_i,x_j,c_{ij} 的解密度为 $sd(c_{ij})$,由解密度的定义

© 中国科学院软件研究所 http://www.jos.org.cn

可知:$|dom(x_i)|\times|dom(x_j)|\times sd(c_{ij})$是当前满足 c_{ij} 的元组个数,平均到 $dom(x_i)$的每个值(x_i,a)之后,其在约束 c_{ij} 上的 AC-support 个数的估计值为$\lfloor|dom(x_j)|\times sd(c_{ij})\rfloor$.但每次计算二元约束的解密度要经过 d^2 次约束检查,使用准确的解密度仍然是不实际的.因此,我们可以在求解开始前计算出每条约束的解密度,在搜索过程中,每条约束使用预先计算好的静态解密度来为每个值估计 m 和 n.使用静态解密度为 $dom(x_i)$中每个值(x_i,a)在约束 c_{ij} 上估计的 AC-support 个数都是相同的,因此我们可以使用 m 和 n 的估计值来判断某个路径上任意两个值之间存在 PC-witness 的概率.由于约束解密度是静态的,因此影响这个概率的参数中动态变化的就只有 $dom(x_k)$的大小.

性质 1. 给定变量 x_i,x_j 和 x_k,约束 c_{ij},c_{ik} 和 c_{jk},(x_j,b)是(x_i,a)在约束 c_{ij} 上的 AC-support,当$|dom(x_k)|$足够大时,我们使用静态解密度估计的(x_i,a)与(x_j,b)在 $dom(x_k)$中存在 PC-witness 的概率无限接近 1.

证明:令 $d=|dom(x_k)|$,那么 $m=d\times sd(c_{ik}),n=d\times sd(c_{jk})$,此时,$1-\dfrac{C(d-m,n)}{C(d,n)}=1-\dfrac{C(d\times(1-sd_1),d\times sd_2)}{C(d,d\times sd_2)}$, 其中,$sd_1$ 代表 $sd(c_{ik})$,sd_2 代表 $sd(c_{jk})$.将组合数公式展开消去相同分母后,得:

$$1-\prod_{i=0,1,2,\ldots,\lfloor d\times sd_2\rfloor}\frac{d\times(1-sd_1)-i}{d-i}=1-\prod_{i=0,1,2,\ldots,\lfloor d\times sd_2\rfloor}\left(1-sd_1-\frac{i\times sd_1}{d-i}\right).$$

上式中,每一项是一个从 0~1 之间的小数,当 d 足够大时,相当于无穷多个 0~1 之间的小数相乘,结果将无限接近 0.因此,当$|dom(x_k)|$足够大时,PC-witness 的概率无限接近 1. □

在实际应用中,$|dom(x_k)|$不一定能够达到足够大的程度,因此我们在图 1 中展示了实验所得的 PC-witness 存在的概率与$|dom(x_k)|$的关系,其中:横轴表示 x_k 值域大小,坐标为 2,3,4,…,9,10,20,30,…,500;纵轴为 PC-witness 存在的概率,每条折线对应一对解密度得到的对应概率,例如,0.1~0.9 表示约束 c_{ik} 与 c_{jk} 的解密度分别为 0.1 和 0.9.观察图中的趋势可知:当解密度一定时,随着$|dom(x_k)|$的增大,PC-witness 存在的概率也在逐渐增加(虽然不是严格的单调递增,但是总的趋势是在逐渐增加),即使两条约束的解密度非常小时(例如 0.1~0.1),当$|dom(x_k)|$增大到 400 以上时,PC-witness 存在的概率也会无限接近 1.在概率 maxRPC 算法中,预先设定一个阈值 ct,当估计的 PC-witness 存在概率大于 ct 时,我们不在这条路径寻找 PC-witness.随着$|dom(x_k)|$的增大,PC-witness 存在的概率也在逐渐增加,因此可以在 2 到 $d(d$ 为 x_k 初始值域大小)的范围内找到值 $cut[x_i][x_j][x_k]$,使得当$|dom(x_k)|<cut[x_i][x_j][x_k]$时,PC-witness 存在的概率小于 ct,此时,我们就在对应的路径上执行 PC-witness 检查.

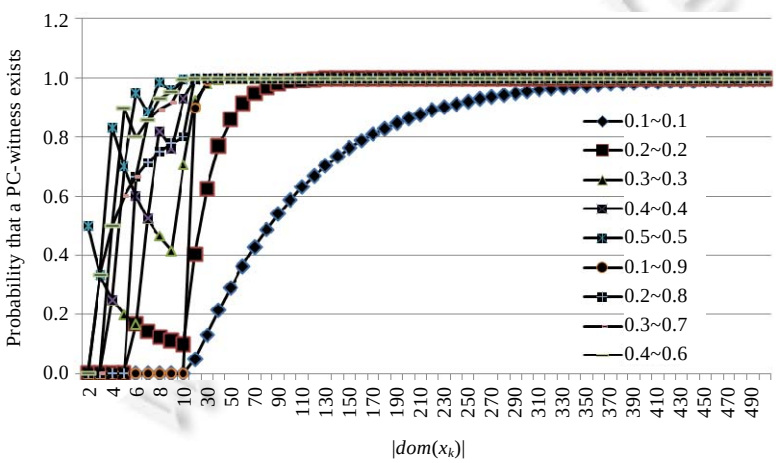

Fig.1 Experiments on the relation between the probability of a PC-witness exists and $|dom(x_k)|$

图 1 PC-witness 存在概率与 x_k 值域大小关系实验

PmaxRPC 算法的 *checkPath* 函数流程如图 2 所示.我们使用数据结构 $cut[x_i][x_j][x_k]$记录对于变量 x_i 与 x_j,在路径 x_k 上需要执行路径检查时的值域大小(是需要检查的最大值,大于这个值时不需要检查).因为 PC-witness 存在的概率随 x_k 值域的增大也在逐渐增长,所以 *checkPath* 函数中的 compute $cut[x_i][x_j][x_k]$在计算 $cut[x_i][x_j][x_k]$

© 中国科学院软件研究所 http://www.jos.org.cn

时不必为 2 到 d 范围内的每一个值都计算概率,而是在 2 到 d 的范围内使用二分搜索策略来计算 $cut[x_i][x_j][x_k]$.所有 $cut[x_i][x_j][x_k]$都被初始化为−1,只有第一次使用时,我们计算对应路径上的 $cut[x_i][x_j][x_k]$.第 1 次计算后不再重复计算.

```
procedure checkPath (x_i,x_j,x_k)
begin
  if dom(x_k)=1 then
    return false;
  if cut[x_i][x_j][x_k]=−1 then
    compute cut[x_i][x_j][x_k];
  if cut[x_i][x_j][x_k]>dom(x_k) then
    return true;
  return false;
end
```

Fig.2 The checkpath procedure of probabilistic maxRPC algorithm

图 2 概率受限最大路径相容算法的路径检查过程

3 讨 论

我们首先讨论与 maxRPC 算法相比,PmaxRPC 需要的额外计算代价.在给定 m,n 和 d 的情况下,计算 PC-witness 存在的概率时间复杂度为 $m+n$.给定变量 x_i,x_j 和 x_k,计算 $cut[x_i][x_j][x_k]$所需要的计算时间为 $d\times(sd(c_{ij})+sd(c_{jk}))\times log(d)$,其中,$d$ 是 x_k 的初始值域大小,$sd(c_{ij})$与 $sd(c_{jk})$是常量,因此,计算 $cut[x_i][x_j][x_k]$时间复杂度为 $O(d\log(d))$.最坏情况下,我们要为任意两个变量 x_i,x_j 计算其在第 3 个变量 x_k 上的 $cut[x_i][x_j][x_k]$,而变量 x_i,x_j 的组合个数最多为 e,e 是 C 中约束个数,因此最坏情况下,要计算 $e\times n$ 次 $cut[x_i][x_j][x_k]$,所以,这里最坏时间复杂度为 $O(endlog(d))$,其中,n 为 X 中变量个数.这一时间复杂度仍小于目前最优的 maxRPC3^{rm} 算法的时间复杂度 $O(en^2d^4)$,因此,PmaxRPC 算法和 maxRPC 算法的时间复杂度相同.数据结构 $cut[x_i][x_j][x_k]$消耗的空间为 $O(en)$,而目前最优的 maxRPC3^{rm} 算法的空间复杂度为 $O(ed)$,因此,概率 maxRPC 算法的空间复杂度为 $O(en+ed)$.

除本文介绍的方法外,2012 年提出的一种基于启发式的 Partial-maxRPC[21]可以节省一些路径检查.在为 $dom(x_i)$中所有值在约束 c_{ij} 上寻找支持时,Partial-maxRPC 首先为 x_i 和 x_j 分别打分,打分的原则是:x_i 与 x_j 中,更容易导致局部相容检查失败的那个变量分值低一些.其中介绍两种较有效的打分方式:一是直接使用变量值域进行打分,二是使用变量值域与变量的加权度的比值[22].当判断在 $dom(x_j)$中是要寻找 AC-support 还是 PC-support 时,如果 x_j 的分值低于 x_i,则需要寻找 PC-support;否则,只寻找 AC-support.

4 实验结果

我们的测试环境为 Intel Core(TM) i5-3210M CPU 2.5GHz/4G RAM,JDK 1.7.本文所用的所有 benchmark 测试用例均可从 http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html 下载.测试使用的 maxRPC 算法是目前最适用于求解的轻量级 maxRPC3^{rm} 算法.文献[23]中介绍的 maxRPC^{bit} 算法虽然效率高于 maxRPC3^{rm},但是它并不是通用算法,并且其效率提高主要是来自位运算方法的编程技巧.

首先在几个随机模型 Model RB[13,14]的 Benchmark 用例上(包括 frb-30-15-1,frb-35-17-1,frb-40-19-1)测试了实际的 PC-witness 存在概率的准确性.Model RB 可以保证生成的问题用例是对应的问题规模上的处于相变阶段的难解用例[13].这里我们不考虑 cpu 耗时,每次计算准确的 m,n 和 d,将计算所得概率按照 0-1 划分为以 0.1 为步长的 10 个区间.我们执行所有的路径检查并同时计算该路径上 PC-witness 存在的概率,统计所有概率区间内每次路径检查时 PC-witness 存在与不存在的次数,进而得到每个概率区间内执行路径检查时 PC-witness 存在的比例,结果如图 3 所示.

图 3 中:x 轴表示 PC-witness 存在的概率区间,其中,0.2 表示概率在 0.2~0.3 之间;对应的 y 轴表示所有 PC-witness 概率落在这一区间时,执行所有的 PC-witness 检查中,PC-witness 存在的比例.由图 3 中数据可知:当我们计算所得的 PC-witness 存在概率较大(0.9~1.0)或较小(0.0~0.3)时,概率比较准确.而概率 maxRPC 中,主要就

© 中国科学院软件研究所 http://www.jos.org.cn

是对计算所得概率较小的路径进行检查,因此,我们的计算概率足以支持概率 maxRPC 算法.

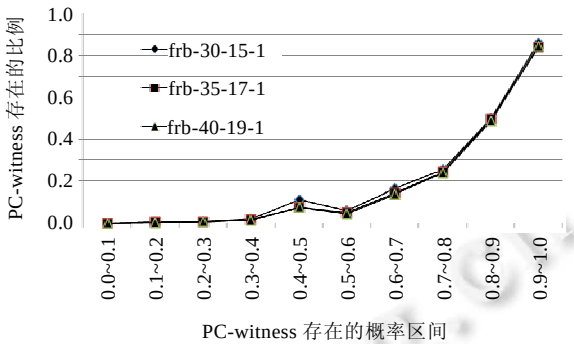

Fig.3 Accuracy of the probability of PC-witness exist

图 3 PC-witness 存在概率准确度

然后,我们测试了概率 maxRPC 算法用于求解的效率,并比较了 PmaxRPC3^{rm} 算法、maxRPC3^{rm} 算法和 MAC 算法的求解效率.PmaxRPC 算法使用的 *ct* 为 0.7,即估算的 PC-witness 存在概率小于 0.3 时,执行路径检查,否则不执行路径检查.测试的数据指标为 cpu 耗时,回溯搜索树节点个数(nodes)和约束检查次数(cc).在所有算法中都使用了避免冗余修正检查的技术[24],变量赋值顺序启发式采用 Dom/Wdeg[22].

首先,我们对 benchmark 库中的几组 Model D[14]模型进行测试.每组 Model D 模型由⟨*n*,*d*,*c*,*t*⟩这 4 个参数表示,其中,*n* 表示变量个数,*d* 表示变量值域大小,*c* 表示约束个数,*t* 表示约束解密度.这几组 Model D 模型涵盖了不同约束解密度的随机问题,并且他们都是对应问题规模上处于相变阶段的难解实例.每组参数包含 100 个实例,平均结果见表 1.

Table 1 Results on random instances

表 1 随机问题测试结果

模型参数	衡量指标	MAC	maxRPC	PmaxRPC
⟨40,8,753,0.9⟩	cpu	1.96	10.99	2.52
	nodes	29,648	17,683	29,648
	cc	2.77E+07	5.06E+08	2.62E+07
⟨40,11,414,0.8⟩	cpu	2.6	5.99	3.28
	nodes	50,996	29,043	50,996
	cc	4.15E+07	2.57E+08	3.76E+07
⟨40,16,250,0.65⟩	cpu	2.57	3.63	3.11
	nodes	54,440	28,147	54,440
	cc	5.21E+07	1.37E+08	4.49E+07
⟨40,25,180,0.5⟩	cpu	3.76	4.08	4.33
	nodes	70,818	32,741	58,050
	cc	1.00E+08	1.61E+08	8.25E+07
⟨40,40,135,0.35⟩	cpu	2.63	2.97	2.84
	nodes	40,218	21,688	38,731
	cc	1.03E+08	1.49E+08	8.45E+07
⟨40,80,103,0.2⟩	cpu	2.34	2.36	1.97
	nodes	22,935	11,677	14,658
	cc	1.51E+08	1.87E+08	1.02E+08
⟨40,180,84,0.1⟩	cpu	3.48	2.96	2.65
	nodes	18,003	7,314	9,164
	cc	3.78E+08	3.91E+08	2.64E+08

表 1 中数据显示:当约束解密度较高时,MAC 算法的求解效率明显高于 maxRPC 算法,而 PmaxRPC 算法无法发现需要检查的路径,此时退化为 AC 算法;但是随着约束解密度的逐渐降低,PmaxRPC 算法的效果逐渐显现,和我们预想的一样,PmaxRPC 算法在求解时,它产生的回溯搜索树的节点个数介于 MAC 与 maxRPC 之间;当约

© 中国科学院软件研究所 http://www.jos.org.cn

束解密度较低时,maxRPC 算法的求解效率高于 MAC 算法;而当解密度低于 0.2 时,PmaxRPC 算法的求解效率高于MAC与maxRPC.除此以外,在这些随机问题结果中我们还观察到另外一个现象,就是随着约束解密度的逐渐降低,maxRPC 的求解效率逐渐增加;当约束解密度非常低时,maxRPC 的求解效率高于 MAC.

最后,我们对 benchmark 库中的一组实际应用问题 radio link frequency assignment problem (RLFAP)[25]进行了测试.RLFAP 问题是为一些无线电装置分配频率,使得这些频率同时满足一些特殊要求的约束并且尽量使用较少的不同频率.RLFAP 问题中既包含解密度较高的约束,又包含解密度较低的约束.结果见表 2.

Table 2 Results on RLFAP instances

表 2 RLFAP 问题测试结果

测试用例	衡量指标	MAC	maxRPC	PmaxRPC
graph8-f10	cpu	1.63	0.42	0.34
	nodes	17,573	2,120	2,704
	cc	1.95E+07	2.44E+07	1.56E+07
graph8-f11	cpu	0.13	0.14	0.08
	nodes	1,209	260	326
	cc	2.23E+06	1.29E+07	8.48E+06
graph9-f9	cpu	2.25	1.73	0.17
	nodes	18,330	4,894	1,419
	cc	2.27E+07	9.12E+07	1.41E+07
graph9-f10	cpu	0.29	0.29	0.21
	nodes	1,472	329	520
	cc	3.42E+06	2.30E+07	1.75E+07
scen11-f3	cpu	>1800	>1800	1508
	nodes	–	–	–
	cc	–	–	–
scen11-f6	cpu	32.59	21.27	32.07
	nodes	259,802	41,351	146,074
	cc	4.84E+08	1.41E+09	1.81E+09
scen11-f8	cpu	3.17	5.39	2.28
	nodes	21,876	7,005	8,325
	cc	5.50E+07	3.85E+08	1.84E+08
scen11-f10	cpu	0.14	0.31	0.11
	nodes	720	212	291
	cc	1.66E+06	2.02E+07	1.24E+07

从表 2 中数据可知:在解 RLFAP 这类实际问题时,maxRPC 的求解效率与 AC 相比无明显优势,有时高于 AC,有时低于 AC.但是使用 PmaxRPC 后,避免了一些冗余的检查,PmaxRPC 效率高于 AC 与 maxRPC;并且在 scen11-f3 问题上,AC 与 maxRPC 无法在半小时内求解.与前面 Model D 的结果中观察到的现象一样,PmaxRPC 在这组实际问题上的搜索树节点个数处于 AC 与 maxRPC 之间.但是 PmaxRPC 的耗时比二者更少,说明了 PmaxRPC 在这组问题上避免了大量无效的路径检查.

5 结 论

本文首先提出一种方法来估计对于任意两个值,在第 3 个变量上存在 PC-witness 的概率,并基于这一概率提出了一种基于概率的 maxRPC 算法——PmaxRPC.PmaxRPC 可以避免在那些 PC-witness 存在概率较高的变量上进行路径检查.实验结果表明:在约束解密度较低的问题上,PmaxRPC 求解效率高于 AC 与 maxRPC.在一组包含不同解密度约束的实际问题上,PmaxRPC 的求解效率仍有着明显的优势,最高可以比 AC 与 maxRPC 节省 10 倍以上的求解耗时.

References:

[1] Freuder EC, Mackworth AK. Constraint Satisfaction: An Emerging Paradigm. In: Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam: Elsevier, 2006. 13−28.

© 中国科学院软件研究所 http://www.jos.org.cn

[2] Li H, Shen H, Li Z, Guo J. Reducing consistency checks in generating corrective explanations for interactive constraint satisfaction. Knowledge-Based Systems, 2013,43:103−111. [doi: 10.1016/j.knosys.2013.01.024]

[3] Bessière C. Constraint Propagation. Rossi F, van Beek P, Walsh T, eds. Handbook of Constraint Programming. Amsterdam: Elsevier, 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] Debruyne R, Bessière C. Some practicable filtering techniques for the constraint satisfaction problem. In: Pollack E, ed. Proc. of the IJCAI'97. Nagoya: Morgan Kaufmann, 1997. 412−417.

[6] Montanari U. Networks of constraints: Fundamental properties and applications to picture processing. Information Science, 1974,7: 95−132. [doi: 10.1016/0020-0255(74)90008-5]

[7] Debruyne R, Bessière C. From restricted path consistency to max-restricted path consistency. In: Smolka G, ed. Proc. of the CP'97. Linz: Springer-Verlag, 1997. 312−326. [doi: 10.1007/BFb0017448]

[8] Lecoutre C, Prosser P. Maintaining singleton arc consistency. In: Dongen MRC, Lecoutre L, eds. Proc. of the 3rd Int'l Workshop on Constraint Propagation and Implementation Held with CP 2006. Nantes: Springer-Verlag, 2006. 47−61.

[9] Sabin D, Freuder EC. Contradicting conventional wisdom in constraint satisfaction. In: Cohn AG, ed. Proc. of the ECAI'94. London: John Wiley and Sons, 1994. 125−129. [doi: 10.1007/3-540-58601-6_86]

[10] Haralick RM, Elliott GL. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 1980,14: 263−313. [doi: 10.1016/0004-3702(80)90051-X]

[11] Balafoutis T, Paparrizou A, Stergiou K, Walsh T. Improving the performance of maxRPC. In: David C, ed. Proc. of the CP 2010. St Andrews: Springer-Verlag, 2010. 69−83. [doi: 10.1007/978-3-642-15396-9_9]

[12] Vion J, Debruyne R. Light algorithms for maintaining Max-RPC during search. In: Bulitko V, Beck JC, eds. Proc. of the SARA 2009. California: AAAI Press, 2009. 167−174.

[13] Xu K, Li W. Exact phase transitions in random constraint satisfaction problems. Journal of Artificial Intelligence Research, 2000, 12:93−103.

[14] Xu K, Boussemart F, Hemery F, Lecoutre C. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. Artificial Intelligence, 2007,171:514−534. [doi: 10.1016/j.artint.2007.04.001]

[15] Gao J, Wang J, Yin M. Experimental analyses on phase transitions in compiling satisfiability problems. Science China Information Sciences, 2014. [doi: 10.1007/s11432-014-5154-0]

[16] Cai S, Su K. Comprehensive score: Towards efficient local search for SAT with long clauses. In: Proc. of the IJCAI 2013. Beijing: AAAI Press, 2013. 489−495.

[17] Cai S, Su K. Local search for boolean satisfiability with configuration checking and subscore. Artificial Intelligence, 2013,204: 75−98. [doi: 10.1016/j.artint.2013.09.001]

[18] Huang P, Yin M. An upper (lower) bound for max (min) CSP. Science China Information Sciences, 2014,57:1−9. [doi: 10.1007/s11432-013-5052-x]

[19] Grandoni F, Italiano G. Improved algorithms for max-restricted path consistency. In: Wallace M, ed. Proc. of the CP 2003. Kinsale: Springer-Verlag, 2003. 858−862. [doi: 10.1007/978-3-540-45193-8_67]

[20] Lecoutre C, Hemery F. A study of residual supports in arc consistency. In: Sangal R, Mehta H, Bagga RK, eds. Proc. of the IJCAI 2007. San Francisco: M. Kaufmann, 2007. 125−130.

[21] Guo J, Li Z, Li H. Partial max-restricted path consistency. In: Proc. of the ICTAI 2012. Athens: IEEE, 2012. 186−190. [doi: 10.1109/ICTAI.2012.33]

[22] Boussemart F, Hemery F, Lecoutre C, Sais L. Boosting systematic search by weighting constraints. In: Lopez de Mantaras R, Saitta L, eds. Proc. of the ECAI 2004. Amsterdam: IOS Press, 2004. 146−150.

[23] Guo J, Li Z, Zhang L, Geng X. MaxRPC algorithms based on bitwise operations. In: Lee J, ed. Proc. of the CP 2011. Perugia: Springer-Verlag, 2011. 373−384. [doi: 10.1007/978-3-642-23786-7_29]

© 中国科学院软件研究所 http://www.jos.org.cn

[24] Li HB, Li ZS, Wang T. Improving coarse-grained arc consistency algorithms in solving constraint satisfaction problems. Ruan Jian Xue Bao/Journal of Software, 2012,23(7):1816−1823 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4129.htm [doi: 10.3724/SP.J.1001.2012.04129]

[25] Cabon B, Givry S, Lobjois L, Fcabon L, Schiex T. Radio link frequency assignment. Constraint, 1999,4:79−89. [doi: 10.1023/A:1009812409930]

附中文参考文献:

[24] 李宏博,李占山,王涛.改进求解约束满足问题粗粒度弧相容算法.软件学报,2012,23(7):1816−1823. http://www.jos.org.cn/1000-9825/4129.htm [doi: 10.3724/SP.J.1001.2012.04129]

李宏博(1985－),男,吉林省吉林市人,博士生,主要研究领域为约束满足问题求解,约束传播,启发式搜索,蛋白质结构预测.

李占山(1966－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为约束优化与约束求解,基于模型的诊断,智能规划与调度,机器学习.

梁艳春(1953－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为机器学习,生物信息学.

© 中国科学院软件研究所 http://www.jos.org.cn