成　绩	
评卷人	

研究生	李雪
学　号	2015363039

武漢紡織大學

研 究 生 课 程 论 文

论文题目	约束满足问题
完成时间	2020. 11. 08
课程名称	人工智能
专　　业	电子信息
年　　级	2020 级

武汉纺织大学研究生处制

一、 引言

约束满足问题(Constraint Satisfaction Problem ， CSP)是人工智能的一个重要研究方向，其研究结果在符号推理、系统诊断、真值维护系统、车间作业调度、资源分配和产品配置等问题中有广泛的应用。由于 CSP 一般都是 NP-hard 问题，为了更好地对其进行求解，1977 年 Mackworth 将弧相容概念引入到 CSP 中，提出了 AC-1 算法，继而又对其进行改进，提出了 AC-2 算法和 AC-3 算法，其中 AC-3 算法直到现在还被广泛使用。1986 年 Mohr 利用支持的概念对 AC-3 算法进行了改进，提出了 AC-4 算法。

2001 年 Bessière 等提出 AC-2000 算法和 AC-2001 算法。后来相继提出了 AC-311，AC-312 和 AC-313 等算法，它们都是对 AC-3 算法的改进。将 AC 算法用于 BT(backtrack)框架之中，就是目前主流的 CSP 求解算法 BT+MAC。在用 BT+MAC 算法处理问题之前，加入一个预处理过程，可以减小搜索空间，提高整个算法的效率。上面提到的弧相容算法都可以与 BT+MAC 相结合，使整个求解问题的效率得到提高。启发式(heuristic)在问题的求解过程中扮演着十分重要的角色，常见的启发式有变量启发式和值启发式。应用变量启发式的求解算法可显著提高算法的效率，使其能更快地找到解或者更早地发现该问题无解；值启发式则使算法优先在有解可能性大的空间中进行搜索，避免将不参与解的值赋予变量，从而达到提高搜索效率的目的。在一般 BT+MAC 算法中采用的启发式是一种静态启发式策略，但问题在求解过程中变量与变量的取值对求解的影响会发生变化。

因此本文在吸取静态值启发式优点的基础上，利用预处理阶段和回溯过程相容检查中的有用信息，提出了一种动态值启发式算法———回溯动态值启发式(Backtrack-Dynamic Value Heuristic ， BT-DVH) 算法，通过随机问题和标准库(benchmark)测试，表明 BT-DVH 算法的效率是维持弧相容(Maintaining Arc Consistency ， MAC)算法的数倍，在求解 CSP 算法上具有很大的优势。

二、 相关工作

2.1 BT-DVH 算法

首先简述 BT+MAC 算法的执行过程如图 2.1 所示：

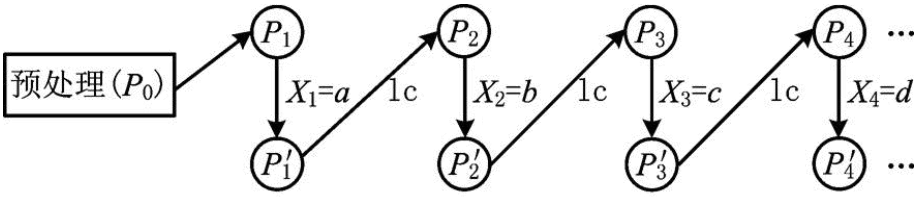


图 2.1 BT+MAC 算法的执行流程图

为提高效率，BT-DHV 算法在 BT+MAC 算法的基础上，加入了变量启发式和动态的值启发式。加入变量启发式是为了优先实例化能够尽早发现冲突或者利于减少回溯次数的变量。

BT-DHV 算法执行过程中需要输入一个 CSP，最后输出一个可行解或者无解退出。pre_process()是一个预处理过程，一般使用 SAC 算法，该算法可以剔除冗余值，缩短算法的执行时间。在实例化开始前，调用 getvariable()函数和 getvalue()函数分别选取一个变量和一个值。保存现场后，对子问题进行相容性检查，目前效率较高的是弧相容技术。为体现 BT-DVH 算法的优越性，举一个简单的例子，分别用 BT-DVH 算法和传统的 BT+MAC 算法求解，比较它们的回溯次数。

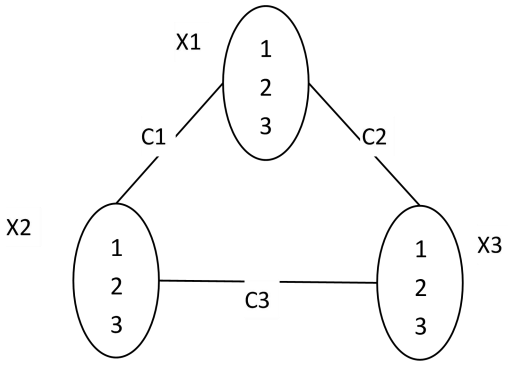


图 2.2 一个 CSP

如图 2.2 所示为一个 CSP，X1，X2，X3 的值域都是{1，2，3}。约束 C1 是(1，1)，(1，2)，(2，2);约束 C2 是(3，1)，(3，2);约束 C3 是(1，3)。

按照 BT-DVH 算法，计算出的 turnups 分别：(X1，1)=3，(X1，2)=1，(X1，3)=0;(X2，1)=1，(X2，2)=2，(X2，3)=2;(X3，1)=1，(X3，2)=1，(X3，3)=1。在变量 X1 中，由于值 3 的 turnups 最小，则先把 X1 实例化为 3，再把 X2 实例化为 turnups 值最小的 1，同理 X3 也实例化为 1。这样就得到了该 CSP 的解{3，1，1}。在此过程中，回溯次数为 0。如果按照传统的 BT+MAC 算法，X1 实例化为 1，X2 和 X3 都只能实例化为 3，则导致冲突，产生了回溯，直

到最后得出解{2，1，1}，回溯次数为 2。由此可见，BT-DVH 算法比传统的 BT+MAC 算法具有更少的回溯次数和更高的效率。

2.2 非二元 CSP 转换成二元 CSP

二元化的优点在于已有大量现成的启发式搜索算法以及当前仅适用于二元 CSP 的算法(如最小正向检查法)。这里简述两种转化的方法:对偶图法(dualgraph)和隐藏变量法(hiddenvariable)。在对偶图转换中，原始问题中的约束变为新的表示中的变量。我们把这些表示约束的变量称为 c-variable，而原始的变量仍称为变量。每个 c-variable 的论域恰好是满足原始约束的元组的集合。两个 c-variable 间存在二元约束，当且仅当原始约束共享一些变量。该二元约束要求那些共享变量取相同的值。见图 2.3CSP 的对偶图表示中，有 4 个 c-variableY1，Y2，…，Y4，每个代表原始问题中的一条 3 元约束(即子句)。例如，c-variableY1 对应非二元约束 R{X1，X2，X6}，Y1 的论域是约束元组集合{(0，0，1)，…，(1，1，1)}。二元约束强制出现在多于一个 c-variable 中的变量有相同的值。

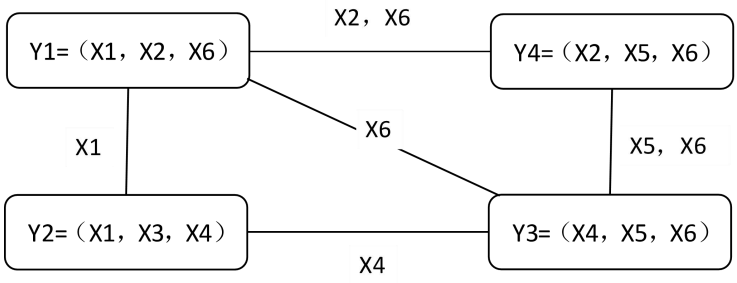


图 2.3 由对偶图法产生的二元 CSP

在隐藏变量表示中,变量集包括原始问题中所有的变量(它们的域没有改变)加上一个新的隐藏变量 h-variable 集。对于原始问题中的每条约束 Ci，增加一个 h-variableHi。Hi 的论域由 Ci 中每个元组的唯一标识符组成。新的表示只包含二元约束,构造如下:对于每个 h-variableHi,在 Hi 和 Vars(Ci)中每个变量间施加一条二元约束。Hi 和 xk 是这样约束的，Hi 中每个值对应于 Vars(Ci)中变量的值元组，则定义了 xk 的唯一值。因此，Hi 和 xk 间的二元约束由 xk 的唯一值与 Hi 中每个值组成(注意约束在另一方向不起作用，xk 的一个值可能与 Hi 的许多值相容)。

三、 算法描述（伪代码）

3.1 CSP 回溯法

回溯法使用约束函数来消除无效子树的搜索，相当于对解搜索树进行了剪枝处理，因而大大提高了效率。但是，回溯法本质上还是解空间上的深度优先搜索，对大多数问题而言，其时间复杂度仍然是指数级的。回溯法也是一种完备的搜索算法，这意味着，如果问题有解的话，它一定能够找到解。

如下为 CSP 回溯法的伪代码：

```
def Consistent(Xi, v(i)):
    for each (Xj ,v(j))∈ Solution:
        if Rij ∈ R and (v(i), v(j)) /∈ Rij :
            return False
     return True

def Backtracking(V ars):
    Select a variable Xi ∈ V ars
     for each value v(i) ∈ Di:
        if Consistent(Xi, v(i)):
              Solution ← Solution + (Xi, v(i))
        if Xi is the only variable in V ars:
            return True
        else:
            if Backtracking(V ars\{Xi}):
                 return True
            else:
                Solution ← Solution \ (Xi, v(i))
     return False

def CSP-BT():
    Solution ← ∅
    return Backtracking(X)
```

3.2 AC-3 算法

AC-1 算法比较低效，每次值域更新后，需要重新检查所有的弧。实际上，只需要重新检查所有与之相关的弧。AC-3 算法则是一种约束传播算法，即在得到约束条件和值域的情况下，算出所有变量满足约束条件的取值范围。AC-3 算法能够处理的只有二元约束，即弧相容的情况，或者说在约束条件中任何一个约束条件都只包含了两个以下的变量。

如下为 AC-3 算法的伪代码：

```
def ArcConsistent(Xi,Xj ):
    Changed ← False
    for each v(i) ∈ Di:
        Found ← F alse
        for each v(j) ∈ Dj :
            if (v(i), v(j)) ∈ Rij :
                Found ← True
                break
        if not Found:
            Di ← Di ← v(i)
            Changed ← T rue
    return Changed

    def AC-3():
        ...
        while Arcs = ∅ :
            Select and remove (Xi, Xj ) from Arcs
            if ArcConsisten(Xi,Xj ):
                if Di = ∅ :
                    return F alse
                else:
                        for each variable Xk ∈ X such that k = j:
                            if Rki ∈ R:
```

Arcs ← Arcs ∪ (Xk, Xi)

return T rue

四、 算法验证（结果与分析）

4.1 8 皇后问题

8 皇后(8-Queens)问题是一个经典的 CSP 问题。该问题是，在国际象棋棋盘上，如何放置 8 个皇后，使得任意 2 个皇后都无法互相攻击对方。换句话说，同一行与同一列上只允许放置 1 个皇后，同一对角线上至多只能放置 1 个皇后。如图 4.1 所示，展示了 8 皇后问题的一个合法解。

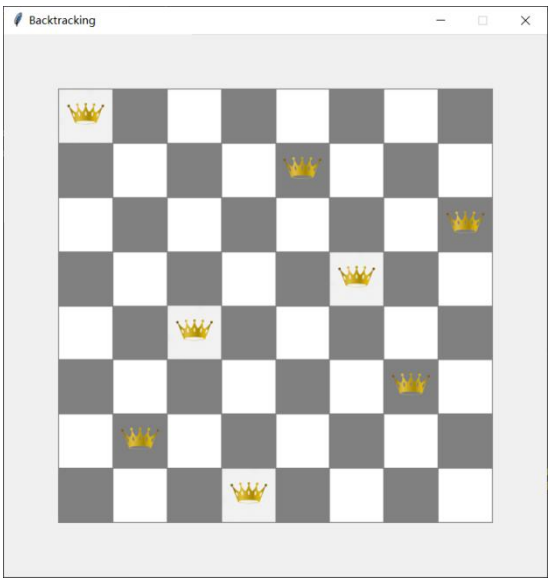


图 4.1 8 皇后问题运行图

分析：以四皇后为例分析，4 皇后 CSP 回溯法执行示意图如图 4.2 所示。在该图中，被大圆圈框住的节点表示回溯法找到的一个解 X = {X1 = 2, X2 = 4, X3 = 1, X4 = 3}。此外，图中还标注了算法访问的总节点个数、约束检测函数的总执行次数以及每个节点的约束检测次数。

八皇后问题算法思路分析：

(1) 第一个皇后先放第一行第一列；

(2) 第二个皇后放在第二行第一列、然后判断是否 OK，如果不 OK，继续放在第二列、第三列、依次把所有列都放完，找到一个合适；

(3) 继续第三个皇后，还是第一列、第二列.....直到第 8 个皇后也能放在一个不冲突的位置，算是找到了一个正确解；

(4) 当得到一个正确解时，在栈回退到上一个栈时，就会开始回溯，即

将第一个皇后，放到第一列的所有正确解，全部得到.

(5) 然后回头继续第一个皇后放第二列，后面继续循环执行 1,2,3,4 的步骤。

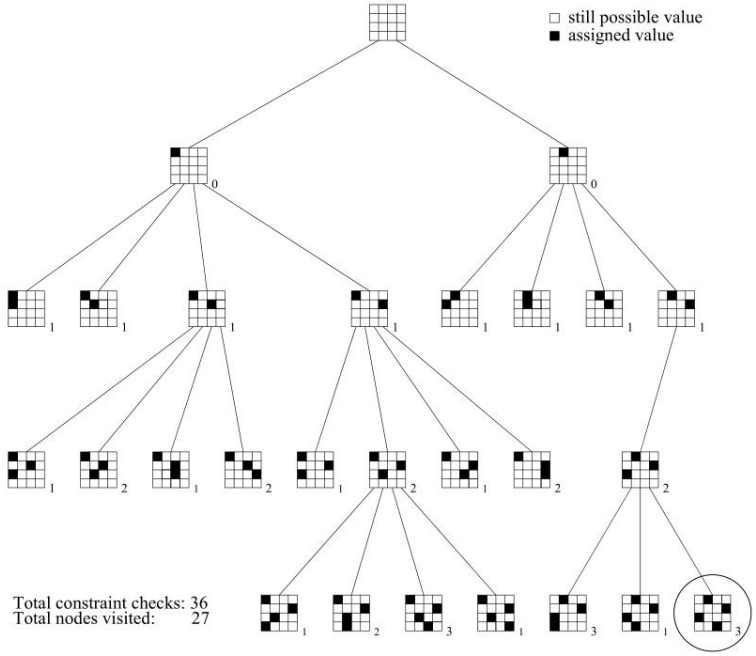
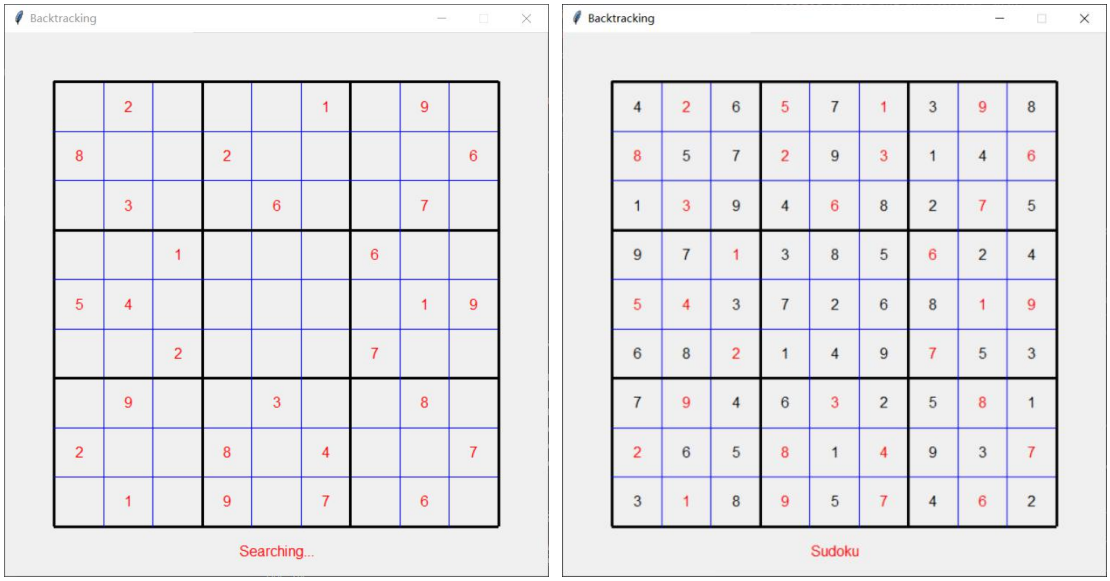


图 4.2 4 皇后 CSP 回溯法执行示意图

4.2 9×9 数独游戏

数独游戏它的每一行或每一列或每个 3 × 3 方块内的所有变量必须取不同的值。如图 4.3 所示，展示了一个典型的数独游戏。左图表示数独游戏的初始状态，右图表示该数独游戏的目标状态或解状态。



（a） 数独游戏的初始状态 （b）数独游戏的目标状态

图 4.3 9×9 数独游戏运行图

分析：数独游戏求解过程如下。

(1) 根据横列、竖列和方格的限制条件排除各个点不可能的数字，并从1~9将各个可能的数字用小字体逐个写进每个空白的格子。

(2) 审视第一步骤的结果，如果发现某个空格只有一个数字，即确定该空格为这个数字。并根据该数字审视其相关的横行、竖列和方格，并划除相同的数字。

(3) 审视各个横行、竖列和方格中罗列出可能的数字结果，若发现某一个数字在各个横行、竖列或方格中出现的次数仅一次，则可以确定该空格的解为此数字。并根据第二条的方法排除与此空格相关列或方格中相同的数字。

(4) 审视各个横行、竖列和方格中罗列的各个可能的结果，找出相对称的两个数组合的空格（或3个、4个组合），并确定这两个空格（或3个、4个）的数字只可能为这两个数字，即两个数字在这两个空格的位置可以交换，但不可能到该行、该列或该方格的其他位置。根据此结果可以排除相关列或方格罗列出相关数字的可能，并缩小范围。

(5) 反复使用2、3、4提到的步骤，逐步得到一个一个空格的解，并将先前罗列的各种可能的结果一个一个排除，使可能的范围越来越小，直至得到最后结果。

五、 算法应用

CSP在许多领域都有着重要的应用，例如人工智能(Artificial Intelligence)、运筹学(Operations Reserch)、调度(Scheduling)、供应链管理(Supply Chain Management)、图算法(Graph Algorithms)、计算机视觉(Computer Vision)及计算语言学(Computational Linguistics)等。

分布式约束满足特别适用于Agent间需要协作的问题。因为约束是可以很好地表示Agent间行为依赖性的一种形式，而且分布式约束满足算法的结构就是用来局部化Agent间的通信的，所以，很多以协作性为主的多主体系统中的应用问题都可以形式化为分布式约束满足问题。

分布式约束满足的另一个应用领域是由多个单一相同Agent构成的网络，比如分布式传感网络。为了解决分布式传感网络中的有限资源和实时需

求的限制，使用资源低开销和高实时性的分布式算法是相当必要的，因此，可将这类问题映射成分布式约束满足问题进行求解。

另一类可形式化为分布式约束满足的问题是资源分配。资源分配中将任务或资源分配给 Agent，资源的有限性导致 Agent 间存在约束关系。在形式化为分布式约束满足问题时，将每个任务或资源看作变量，对任务或资源相应的指派看作赋值，它们彼此间的制约就是约束。

还有很多其他寻找 Agent 间行动决策的一致组合的问题，比如值班表安排、分布式日程安排等，都可以形式化为分布式约束满足问题来求解。

六、 结论与展望

通过学习 CSP 算法以及回溯法、约束传播的求解方法，认识到约束满足问题（CSP）的状态是变量/值对的集合，解条件通过一组变量上的约束表示。许多重要的现实世界问题都可以描述为 CSP。很多推理技术使用约束来推导变量/值对是相容的，哪些不是。这些可以是结点相容、弧相容、路径相容和 k-相容。此外回溯搜索也是深度优先搜索的一种形式，经常用于求解 CSP，推理和搜索可以交织进行。

在求解 CSP 的过程中，启发式仍然具有不可替代的作用，但是目前研究的重点仅在于变量启发式和静态的值启发式。本文吸收了静态值启发式的优势，利用预处理阶段和回溯过程的相容检查中的有用信息，提出了一种动态的值启发式算法——BT-DVH 算法。该算法随着每次相容性检查对启发参数进行动态更新，从而达到先将更有可能成为解的值赋予变量，对有解的 CSP 求解效率的提高具有明显的效果。测试结果表明，无论是随机 CSP 还是标准库问题，此算法的执行效率都高于目前主流算法，能达到其 2～10 倍。但是本文所提出的动态值启发式主要针对有解的 CSP，下一步工作将针对全部的 CSP 进行研究，重点放在无解的问题上，以求得到更优的求解算法。

七、 参考文献

[1] 王孜文. (0). 基于启发式的约束满足问题研究. (Doctoral dissertation, 吉林大学).

[2] 曹伟伟;李铁克;;基于约束满足的 Job Shop 调度算法中的启发式规则[A];全国第八届工业工程与企业信息化学术会议论文集[C];2004 年

[3] 刘洋;贺仁杰;陈英武;;基于动态约束满足的一类含时间窗口的多资源动态调度模型与方法[A];中国运筹学会第七届学术交流会论文集（中卷）[C];2004 年

[4] 王秦辉;约束满足及其分布式求解和应用研究[D];中国科学技术大学;2007 年

[5] 黄晶;基于多 Agent 分布式约束优化问题求解方法研究[D];吉林大学;2008 年

[6] 谷学强;陈璟;王克波;;面向目标区域规划的分布式约束满足求解方法[J];兵工自动化;2009 年 01 期

[7] 孙伟,马绍汉;约束满足问题并行弧相容算法[J];计算机工程与科学;1997 年 01 期

[8] 李占山;王孜文;艾阳;李宏博;;基于 AC-4 的动态值启发式约束满足问题求解算法[J];吉林大学学报(工学版);2011 年 05 期

[9] 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[J]. 软件学报, 2012(07):001816-1823.

[10] 杜小勤。《人工智能》课程系列，Part I: Python 程序设计基础，2018/06/13。

[11] 杜小勤。《人工智能》课程系列，Part II: Python 算法基础，2018/07/31。

[12] 杜小勤。《人工智能》课程系列,Chapter 6:约束满足问题，2018/10/23。

[13] 贺前华, 韦岗. 多声道音频编码 AC-3 算法原理[J]. 计算机工程, 1998, 24(12):44-46.

[14] 李彦涛, 陈玉健, 孙家广. 混合式几何约束满足的研究[J]. 计算机学报, 2001(04):000347-353.