成　绩	
评卷人	

研究生	冯朗
学　号	2015363065

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. 7
课程名称	人工智能
专　　业	电子信息
年　　级	2020

武汉纺织大学研究生处制

一、 引言

关于搜索理论的研究是人工智能的核心领域，在人工智能领域，搜索即对某种问题的求解过程，而搜索方式就是求解该问题的方式或者模型。从提出问题（即初始状态）到解决问题（即目标状态），整个的求解过程， 事实上就是一个状态空间搜索[1]的过程。约束满足问题在人工智能中的非常重要，它在物流规划、生产调度、产品配置器、图算法、计算机视觉及计算语言学、生物信息等领域都有非常广泛的应用。

在计算机科学与人工智能领域， CSP 中的回溯搜索研究，最早可以追溯至 1965 年， Golomb 和 Baumert 在 JACM 发表了论文“Backtrack Programming”。然而，实际上，回溯搜索是 19 世纪提出的基本算法，出现于当时的娱乐数学游戏中。 CSP 中的玩具实例 8 皇后，据说在 1848 年由国际象棋选手 Max Bazzel 提出，一直是早期约束满足问题的主要实验对象之一。在二次世界大战后，回溯搜索是计算机科学与运筹学的学术研究主题。

本文旨在介绍约束满足问题的基本思想以及两种求约束满足问题的通用方法分别为回溯法和约束传播法，以及对其存在的一些问题进行讨论，并对此提出一些想法与思考，以供日后的学习和参考。

本文主要有以下几个部分组成，首先会介绍约束满足问题的基本思想，以及对算法的问题及其中的关键技术做一个讨论，然后用一个实例来进行验证和分析，接着讨论一些相关的应用和发展，最后做一个总结展望。

二、 相关工作

1. 约束满足问题

约束满足问题（Constraint Satisfaction Problem， CSP）指的是，针对给定的一组变量及其需要满足的一些限制或约束条件，找出满足这些约束关系的一个解、全部解或最优解。如果解集为空，则称该 CSP 问题是不

[1] 状态空间搜索就是将问题求解过程表现为从初始状态到目标状态寻找这个路径的过程。由于求解问题的过程中分枝有很多，主要是求解过程中求解条件的不确定性，不完备性造成的，使得求解的路径很多这就构成了一个图，我们说这个图就是状态空间。问题的求解实际上就是在这个图中找到一条路径可以从开始到结果。这个寻找的过程就是状态空间搜索。

满足的 (Unsatisfable)；如果需要找出最优解或次优解，则 CSP 被称为约束优化问题 (Constraint OptimizationProblem, COP)。

在 CSP 发展的早期，有两个主流的发展方向，语言和算法。在语言研究领域， CSP 更多地受到逻辑程序设计 (Logic Programming)——某种形式的约束逻辑程序设计的影响，以程序设计语言与库的开发工作为主。在算法研究领域， CSP 主要受到人工智能领域中搜索方法的影响，它将主要精力集中在启发式方法与推理方法——基于约束的推理(Constraint-based Reasoning) 与基于示例的推理 (Case-based Reasoning) 等研究中。

实际上，CSP 可以说是一种具有通用性的一类方法的总称，具有众多的变体，这也说明 CSP 的应用范围并不仅仅的局限于人工智能或者说计算机领域，在很多其他工学或者经济学等领域同样适用。

1.1 CSP 形式化

如果从研究对象的粒度[2]来看，基于状态空间的搜索算法是把状态看作一个黑盒子，通过领域特定的启发式对搜索进行引导与评估，并通过测试来确定状态是否为目标状态。

而 CSP 不同，它将状态进行了进一步的细分，利用一组变量及其值来表示状态。当每个变量的值满足所有关于变量的约束时，问题就得到了求解。CSP 算法充分利用了状态各变量及其内在的约束结构，所以这种策略是通用的，能够解决某一类型的问题，而不是依据问题来特定一个启发函数来求解复杂的问题。

很明显 CSP 有一下一系列的有点：

- 结构化表示；
- 使用通用策略求解；
- 通过识别违反约束的变量值及其组合，可以迅速地消除大规模的搜索子空间，或避免对这些子空间进行搜索；

1.2 CSP 定义

约束可满足问题定义为 1 个三元组（X，D，C），其中 X＝｛X_1，X_2，…，

[2] 粒度就是同一维度下，数据统计的粗细程度，计算机领域中粒度指系统内存扩展增量的最小值。粒度问题是设计数据仓库的一个最重要方面。粒度是指数据仓库的数据单位中保存数据的细化或综合程度的级别。细化程度越高，粒度级就越小；相反，细化程度越低，粒度级就越大。数据的粒度一直是一个设计问题。

X_n}是 1 个由 n 个变量组成的集合，D＝{D_1，D_2，…，D_n}是其值域的集合，每个变量对应 1 个值域；C＝ {C_1，C_2，…，C_e} 是 e 个约束组成的集合，每个约束C_i是有序对 <scope,rel>，其中 scope 表示约束中的变量组， rel 定义了这些变量取值应该满足的关系，例如，如果 X1、 X2 的值域均为{A,B}，约束条件是二者不能取相同的值，那么关系可以描述如下： <(X_1; X_2) |{(A,B)，(B,A)}>，或者<(X_1，X_2)，$X_1 \neq X_2$>。很明显，scope 就是变量组，即（X_1，X_2），而 rel 就是约束条件<$X_1 \neq X_2$>。

由此，我们定义 CSP 的状态和解的基本概念。状态指的是对部分或全部变量的一个赋值，例如 {$X_i = v_i$，$X_j = v_j$}。一个不违反任何约束条件的赋值被称为一致的或相容的 (Consistent)、合法的 (Legal)。而完整的赋值 (Complete Assignment)，指的是每个变量都已经被赋值。CSP 问题的解，指的是相容且完整的赋值。

1.3 CSP 经典实例

8 黄后问题是一个经典的 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 1-1 就是一个 8 皇后的合法解。

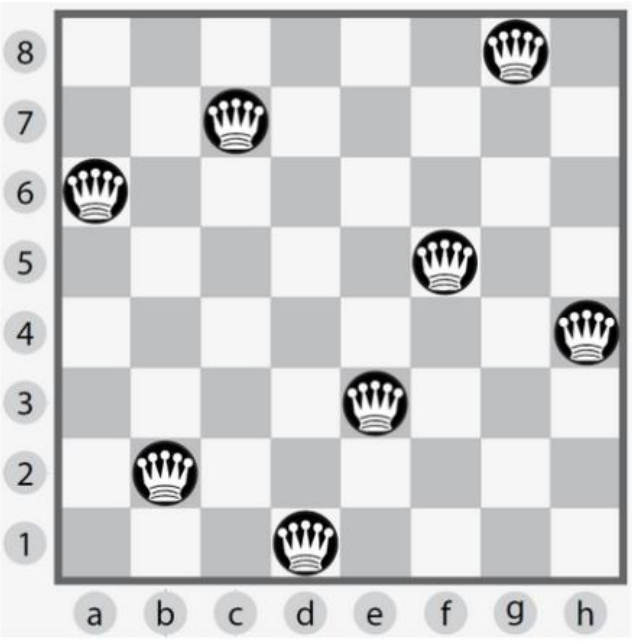

图 1-1

在 CSP 问题中如果约束条件所包含的变量个数不超过 n，那么称该约束问题是一个 n 元的约束问题。像 8 皇后就是一个 2 元的约束（也可以说是一个 8 元的约束），因为他的约束中同时只存在两个变量，即两个皇后。任意一个 n 元的约束问题最终都可以转换成 2 元的约束，简单来说，即使有 n 个变量存在，其中一个变量与其他 n-1 个变量存在约束，但是这一个约束其实可以看做 2 个变量有 n-1 个约束。所以 8 皇后问题也可以说是一个 8 元 CSP，因为它的每个变量都必须取不同的值，即 8 个皇后的位置都互相存在约束。如图 1-2 所示，每一个点与其余的所有点之间都有约束。

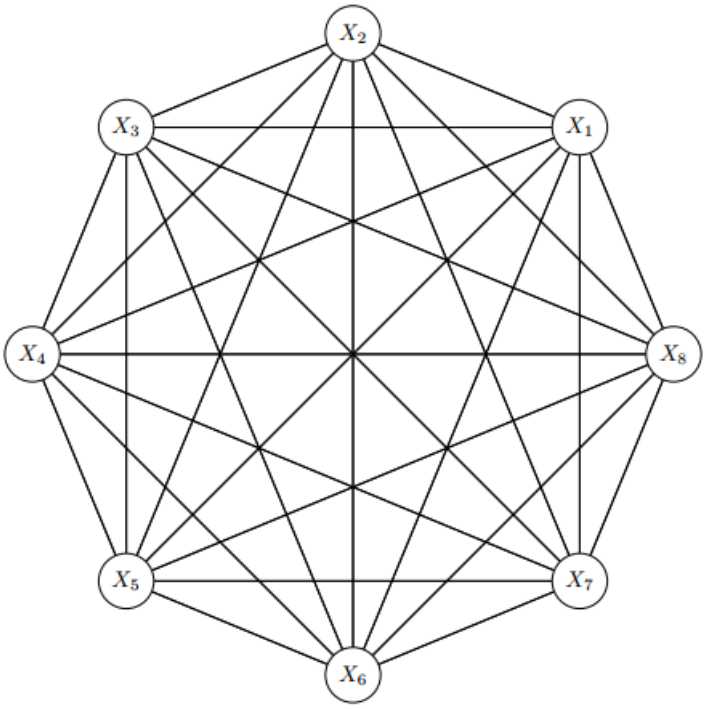

图 1-2

为了求解的便利，我们通常都将其看做 2 元 CSP。

根据 CSP 的定义，可以将 8 皇后问题形式化：

- 使用变量 Xi 表示每一行中的皇后，其中$1 \leq i \leq 8$，则 8 皇后问题中的变量集合为$X = \{X_1, X_2, \ldots, X_8\}$，表示总共有 8 个皇后。
- 每个变量 X_i 的具体取值则表示该皇后所在列的位置，即值域 D $=\{D_1, D_2, \ldots, D_8\}$。$D_1 = D_2 = \cdots = D_8 = \{1,2, \ldots,8\}$。即 i 表示行，$X_i$ 表示列。若$X_1 = 7$，则表示该皇后位于 1 行 7 列。
- 任意两个变量之间的约束关系可以表示为$<(X_i, X_j), R_{ij}>$，即变量为

X_i和X_j,约束条件为R_{ij}。顾对约束条件R_{ij}有：

1. 两个皇后不能再同一行：$\mathrm{i} \neq \mathrm{j}$

2. 两个皇后不能再同一列：$X_i \neq X_j$

3. 两个皇后不能位于对角线上：$|\mathrm{i}-\mathrm{j}| \neq |X_i - X_j|$

因此，约束关系可以表示为：

$$<(X_i, X_j), \mathrm{i} \neq \mathrm{j} \,\&\, X_i \neq X_j \& |\mathrm{i}-\mathrm{j}| \neq \left|X_i - X_j\right| >$$

将问题形式化为 CSP 以及表示为约束图，可以获得如下好处： CSP 表示问题较为自然；可以使用 CSP 通用求解系统进行问题的求解，且比其它搜索技术求解要简单快捷得多。原因在于， CSP 求解能快速消除搜索空间中的一些无用子空间，或避免对这些子空间进行搜索（这与剪枝算法很相似）。

总体而言， CSP 属于 NP-Complete 问题[3]:一方面，“不难”检测出一组给定的变量赋值是否满足约束，也“不难猜测”出一组合法的变量赋值，这表明 CSP 属于 NP 问题；另一方面， SAT(Satisfability Problem)[4]属于 NP-Hard 问题，且可以被形式化为 CSP。这说明 CSP 问题既是一个 NP 问题，又是 NP-Hard 问题，所以 CSP 是一个 NP-Complete 问题。

2. 求解 CSP 的通用方法

2.1 回溯法

回溯 (Backtracking， BT) 法是一种直接在解空间中按照深度优先搜索策略执行搜索的通用型算法，即它是一种求解问题的基本策略。为了能够

---

[3] 在计算机领域，一般可以将问题分为可解问题和不可解问题。不可解问题可以分为两类：一类如停机问题，的确无解；另一类虽然有解，但时间复杂度很高。可解问题分为多项式问(Polynomial Problem， P 问题) 和非确定性多项式问题 (Nondeterministic Polynomial Problem，NP 问题)。 P 问题是一个判定问题类，这些问题可以用一个确定性算法在多项式时间内判定或解出。 NP 问题指的是，不能确定一个复杂问题能否在多项式时间内找到答案，但是可以在多项式时间内验证答案是否正确。在 P 和 NP 问题中， P 的难度最低， NP 由于只对验证答案的时间作了限定，从而有可能包含某些无法在多项式时间内找到答案的问题，即 NP 是比 P 更困难的问题。NP-Complete(NP 完全) 问题是 NP 问题中最难的问题。它需要同时满足下面两个条件：1.它是一个 NP 问题；2.所有的 NP 问题 (在多项式时间内) 都可以约化到它。围棋或象棋的博弈问题、国际象棋的 n 皇后问题、密码学中的大素数分解问题等，都属于 NP-Complete 类问题。如果可以证明某问题有一个子问题是 NP-Hard 问题，那么该问题是一个 NP-Hard 问题。通俗的讲，已经有一个很难的问题 L',而 L 问题比 L'更难解决，那么该问题就是 NP-Hard 问题。

[4] 布尔可满足性问题(Boolean satisfiability problem;SAT)是第一个被证明属于 NP 完全的问题。SAT 问题的基本形式指给定一个命题变量集合 X 和一个 X 上的合取范式 V(X)，判 断 是 否 存在一个真值赋值 t(X)，使 V(X)为真。如果能找到，则称 V(X)是可满足的(satisfiable)，否则称 V(X)是不可满足的(unsatisfiable)。SAT 问题的模型发现形式指当 V(X)可满足时，给出使公式 V(X)可满足的一组赋值。

在解空间执行有效的搜索，需要为解空间定义合适的结构，以便将解空间中的元素组织在一起，从而允许回溯法遍历整个解空间。

基本的回溯法是遍历解空间的一种基本框架，为了提高搜索效率，避免一些无效的搜索，通常采取两种措施;

1. 使用约束函数剪去不满足约束条件的子树
2. 使用限界函数剪去得不到最优解的子树

通常情况下，回溯法使用深度优先搜索框架。为便于理解，将变量按层次表达为搜索树结构（虽然实际情况下更多的是图，但是树结构便于理解和描述）。在搜索树中，每一层的所有节点表示某种变量；某个节点的所有分支表示该变量的所有可能取值。回溯法从树的根节点开始，每次选择一个分支执行，即选取该变量的某个具体取值（实例化）。然后，算法检测新变量的取值与已有变量的取值是否满足约束条件：如果满足，表明新变量实例化成功，继续往下执行，到达下一层（即选取另一个新变量)；否则，新变量实例化失败，需要回溯重新实例化一个新值。上述过程，递归进行：

- 如果所有变量都有合法值或满足约束的值，则找到问题的解
- 如果当前变量的所有实例化回溯失败，则执行变量回溯，即回溯至上一个变量的某个取值

直至找到一个解或所有解，或者已经遍历所有的变量取值组合，没有找到解，算法停止。

从回溯法的算法描述可以看出，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。

回溯法发展至今，它存在着两个无法完全避免的问题：（反复）抖动和冗余的约束检测。抖动指的是，算法在解空间的不同部分进行搜索时总是由于同样的原因而导致搜索失败。对于这个问题，算法可以直接回跳至导致失败的变量处进行回溯来避免，相关的算法有 Backjumping（BJ）算法它可以在多个层级之间进行回跳，以避免抖动现象的发生。Conflict-directed

Backjumping（CBJ）算法，它可以执行多个回跳，这些算法的目的都是为了减少访问无效节点的数目，从而减小约束检测的个数，最终达到提高搜索效率的目的。针对冗余约束检测问题，研究者提出了 Backmarking(BM) 算法，用来阻止相同的约束检测反复发生。

2.2 约束传播

在常规的状态空间中，算法能够执行的唯一任务是搜索。但是，在 CSP 的解状态空间中，算法除了可以执行回溯搜索外（即从变量的多种可能取值中选取新值），还可以执行推断——这是一种利用约束关系进行推理的方法，被称为约束传播。具体而言，约束传播指的是，使用约束减少变量的合法或有效的取值范围，并将这种影响传播到与此变量存在约束关系的其它变量上，从而间接地压缩了状态空间，提高了求解效率。

约束传播的基本思想是，将一个 CSP 转换为等价的、更简单且易于求解的 CSP。其主要技术手段是，消去变量中无效冗余的取值，并将这种变化通过约束变量进行传播，从而也消去那些变量中的无效冗余取值。约束传播可以在不同的粒度上执行，从而形成了不同的约束传播方法。

为了方便理解，我们引入约束图的概念，即使用图的二元约束关系：用节点表示变量，用边表示约束关系。例如，节点存在一条导向自己的边，那么该节点表示的变量存在一元约束。

由于约束传播能够消去变两个中无效的冗余值，所以可以把约束传波作为回溯的预处理，或者结合近回溯算法的框架中，搜索与约束传播交替进行，从而提高回溯法的效率。

有些一致性或相容性检测算法通过将约束变量集的规模与交互层次限制在局部范围，使用效率更高的多项式算法，让约束变得更紧凑，或者使隐式的约束显式化，从而达到识别或消除抖动的效果。

2.2.1 节点一致性

节点一致性算法(Node Consistency Algorithm)：当且仅当 $D_i \subseteq R_i$ 时，我们称值域 D_i 的变量或顶点 X_i 具有节点一致性。此处R_i表示约束，根据前文 1.2 节 CSP 定义中关于三元组的定义，一元约束$C_i =< (X_i), R_i >$;二元约束$C_S =< (X_i, X_j), R_{(i,j)} >$

如果节点X_i并不具备节点一致性，那么可以执行以下操作：

$$\mathrm{D}_i' = D_i \cap R_i$$

$$D_i \leftarrow \mathrm{D}_i'$$

使得节点X_i具备节点一致性。

节点一致性算法仅仅关注约束图中的 1 元约束。如果 CSP 约束图不满足节点一致性，那么意味着至少存在一个节点 X_i，它的某个实例化值 V(i) 总是会导致回溯失败。换句话说，节点 X_i 的实例化值 V(i) 是冗余的，不会出现在任何解中。因此，可以安全地将 V(i) 去除,所以通过节点一致性算法能够很好的消除约束图中的 1 元约束。

2.2.2 弧一致性

弧一致性算法 (Arc Consistency Algorithm) 使用二元约束进一步对变量值域进行压缩处理。由于，可以将 n 元约束转换成 2 元的约束，因此，我们只定义 2 元的 CSP 求解器。

弧的一致性：对于X_i与X_j的值域D_i 和D_j 只有当且仅当 D_i 的每个元素，在 D_j中存在相应的元素，它们之间满足关系$R_{(i,j)}$时，弧（X_i, X_j）具备弧的一致性。可以表示为：

$$D_i \subset \pi_i(D_j \bowtie R_{(i,j)})$$

其中， π 是投影 (Porjection) 算子。

同样的，当弧（X_i, X_j）不满足弧的一致性时，可以通过以下操作：

$$\mathrm{D}_i' = D_i \cap \pi_i(D_j \bowtie R_{(i,j)})$$

$$D_i \leftarrow \mathrm{D}_i'$$

使得弧〈X_i, X_j〉满足弧的一致性。这样可以删掉D_i中与D_j不满足关系$R_{(i,j)}$的元素，同样的，可以弧（X_j, X_i）执行类似操作，删掉D_j中与D_i*不满足关系* $R_{(i,j)}$的元素，进一步的压缩X_j的值域D_j。

基本的弧一致性算法简单地检测所有的弧，直至没有进一步的 (变量) 值域缩减，该算法被称为 AC-1 算法。我们可以将弧一致性算法看作局部一致性的消解与传播算法：随着消解与传播的迭代，整个约束图的一致性将逐

渐地以单调的方式达到一个固定点。此时，约束图取得了一致性，算法即终止。

2.2.3 路径一致性

路径一致性算法 (Path Consistency Algorithm) 进一步增强了变量的局部约束检测能力，将二元约束检测升级为三元约束检测。

设长度为 2 的路径，以节点 X_i为起点，X_m为中间节点，X_j为终节点，则路径(X_i, X_m, X_j)是路径一致性的。当且仅当$R_{(i,j)}$允许的每对值（a,b），在变量X_m中，都存在对应的值 c，使得允许的值对(a,c)和$R_{(m,j)}$允许的值对(c,b)都存在时，满足路径一致性,即：

$$R_{(i,j)} \subset \pi_{ij}(R_{(i,m)} \bowtie D_m \bowtie R_{(m,j)})$$

同样的，如果路径(X_i, X_m, X_j)不满足一致性，可以执行操作：

$$R'_{(i,j)} = R_{(i,j)} \cap \pi_{ij}(R_{(i,m)} \bowtie D_m \bowtie R_{(m,j)})$$

$$R_{(i,j)} \leftarrow R'_{(i,j)}$$

与弧的一致性相似，只要约束图中的约束发生了变化，那么就重新检测，经过多次的约束，删掉重复的约束，从而提高搜索效率。

3. 回溯法和约束传播的结合使用

关于回溯法和约束传播的结合，通常有两种方式：

- 将约束传播作为回溯法的预处理步骤：利用约束传播对约束图中节点的值域进行压缩处理（例如使其满足弧一致性），然后再在预处理后的约束图上执行回溯搜索；
- 将约束传播作为算法组件，嵌入到回溯法框架中；

在第二种方法中，具体地，将两者结合在一起的基本想法是，在搜索树上，每当访问一个节点时，需要执行一次约束传播算法：压缩相关节点或变量的值域。一旦发现约束传播路径上任何节点的值域为空 (即意味着不可能产生有效的解)，则该节点将被裁减掉。通过剪枝的方式，减少无效的搜索，从而提高搜索的效率。

三、 算法描述（伪代码）和验证（结果与分析）

1. 回溯算法

def Consistent(X_i,$v^{(i)}$):

1. for each $(X_j,v^{(j)})\in Solution$:
2. if $R_{ij} \in R$ and $(v^{(i)}, v^{(j)}) \notin R_{ij}$:
3. return False
4. return True

def Backtracking($Vars$):

1. Select a variable $X_i \in Vars$
2. for each value $v^{(i)} \in D_i$:
3. if Consistent(X_i, $v^{(i)}$):
4. $Solution \leftarrow Solution + (X_i, v^{(i)})$
5. if X_i is the only variable in $Vars$:
6. return True
7. else:
8. if Backtracking($Vars \backslash \{X_i\}$):
9. return True
10. else:
11. $Solution \leftarrow Solution - (X_i, v^{(i)})$
12. return False

def CSP-BT():

1. $Solution \leftarrow \emptyset$
2. return Backtracking(X)

```
[(0, 0), (1, 2), (2, 4), (3, 1)]
[(0, 0), (1, 2), (2, 4), (3, 1)]
[(0, 0), (1, 2), (2, 4)]
[(0, 0), (1, 2), (2, 4), (3, 6), (4, 1), (5, 3)]
[(0, 0), (1, 2), (2, 4), (3, 6), (4, 1)]
[(0, 0), (1, 2), (2, 4), (3, 6)]
[(0, 0), (1, 2), (2, 4), (3, 6)]
[(0, 0), (1, 2), (2, 4)]
[(0, 0), (1, 2), (2, 4), (3, 7), (4, 1), (5, 3)]
[(0, 0), (1, 2), (2, 4), (3, 7), (4, 1)]
[(0, 0), (1, 2), (2, 4), (3, 7)]
[(0, 0), (1, 2), (2, 4), (3, 7)]
[(0, 0), (1, 2), (2, 4)]
[(0, 0), (1, 2)]
[(0, 0), (1, 2), (2, 5), (3, 1), (4, 6)]
[(0, 0), (1, 2), (2, 5), (3, 1)]
[(0, 0), (1, 2), (2, 5)]
[(0, 0), (1, 2), (2, 5), (3, 7), (4, 1)]
[(0, 0), (1, 2), (2, 5), (3, 7), (4, 1)]
[(0, 0), (1, 2), (2, 5), (3, 7)]
[(0, 0), (1, 2), (2, 5)]
[(0, 0), (1, 2)]
```

图 3-1. 8 皇后回溯开始过程

```
[(0, 0), (1, 4), (2, 6), (3, 1), (4, 5)]
[(0, 0), (1, 4), (2, 6), (3, 1)]
[(0, 0), (1, 4), (2, 6)]
[(0, 0), (1, 4)]
[(0, 0), (1, 4), (2, 7), (3, 1), (4, 3), (5, 6)]
[(0, 0), (1, 4), (2, 7), (3, 1), (4, 3)]
[(0, 0), (1, 4), (2, 7), (3, 1)]
[(0, 0), (1, 4), (2, 7), (3, 1), (4, 6), (5, 2)]
[(0, 0), (1, 4), (2, 7), (3, 1), (4, 6)]
[(0, 0), (1, 4), (2, 7), (3, 1)]
[(0, 0), (1, 4), (2, 7)]
[(0, 0), (1, 4), (2, 7), (3, 5), (4, 2), (5, 6), (6, 1), (7, 3)]
```

图 3-2. 8 皇后回溯最终过程

该回溯算法只找出了一个满足约束条件的解。图 3-1 为回溯算法的在 8 皇后问题中一开始的运行过程，从图中可以看到，第一个皇后的落点永远是（0,0），然后为了满足约束条件（0，i）和（i,0）皆不能放置皇后，$0 < i < 8$。所以第二个皇后的落点从(1,2)开始，然后一次往下，发现在放置第四个皇后(3,1)之后，无法满足约束条件，只能执行回溯，回溯至(2,4)，继续执行算法，放置第六个皇后(5,3)后，又无法满足约束条件，执行回溯，重复执行算法，最终多次的迭代，最后得出图 3-2 的最终满足约束的第一个结果。由于该算法满足结果后就执行返回，最终只能找出满足约束的第一个解。

def Consistent(X_i,$v^{(i)}$):
1. for each $(X_j,v^{(j)}) \in Solution$:
2. if $R_{ij} \in R$ and $(v^{(i)}, v^{(j)}) \notin R_{ij}$:
3. return False
4. return True

def Backtracking($Vars$):
1. Select a variable $X_i \in Vars$
2. for each value $v^{(i)} \in D_i$:
3. if Consistent(X_i, $v^{(i)}$):
4. $Solution \leftarrow Solution + (X_i, v^{(i)})$
5. if X_i is the only variable in $Vars$:
6. $Solutions \leftarrow Solutions + Solution$
7. else:
8. Backtracking($Vars \backslash \{X_i\}$)
9. $Solution \leftarrow Solution - (X_i, v^{(i)})$

def CSP-BT():
1. $Solutions \leftarrow \emptyset$
2. $Solution \leftarrow \emptyset$
3. return Backtracking(X)

对该算法进行改进，添加一个列表，用来存放更多解的结果。并对算法的结束条件进行约束，最终达到可以得到所有解的算法。下面来看最终的实现效果。

```
[(0, 0), (1, 4), (2, 7), (3, 5), (4, 2), (5, 6), (6, 1), (7, 3)]
[(0, 0), (1, 5), (2, 7), (3, 2), (4, 6), (5, 3), (6, 1), (7, 4)]
[(0, 0), (1, 6), (2, 3), (3, 5), (4, 7), (5, 1), (6, 4), (7, 2)]
[(0, 0), (1, 6), (2, 4), (3, 7), (4, 1), (5, 3), (6, 5), (7, 2)]
[(0, 1), (1, 3), (2, 5), (3, 7), (4, 2), (5, 0), (6, 6), (7, 4)]
[(0, 1), (1, 4), (2, 6), (3, 0), (4, 2), (5, 7), (6, 5), (7, 3)]
[(0, 1), (1, 4), (2, 6), (3, 3), (4, 0), (5, 7), (6, 5), (7, 2)]
[(0, 1), (1, 5), (2, 0), (3, 6), (4, 3), (5, 7), (6, 2), (7, 4)]
[(0, 1), (1, 5), (2, 7), (3, 2), (4, 0), (5, 3), (6, 6), (7, 4)]
[(0, 1), (1, 6), (2, 2), (3, 5), (4, 7), (5, 4), (6, 0), (7, 3)]
[(0, 1), (1, 6), (2, 4), (3, 7), (4, 0), (5, 3), (6, 5), (7, 2)]
[(0, 1), (1, 7), (2, 5), (3, 0), (4, 2), (5, 4), (6, 6), (7, 3)]
[(0, 2), (1, 0), (2, 6), (3, 4), (4, 7), (5, 1), (6, 3), (7, 5)]
[(0, 2), (1, 4), (2, 1), (3, 7), (4, 0), (5, 6), (6, 3), (7, 5)]
[(0, 2), (1, 4), (2, 1), (3, 7), (4, 5), (5, 3), (6, 6), (7, 0)]
[(0, 2), (1, 4), (2, 6), (3, 0), (4, 3), (5, 1), (6, 7), (7, 5)]
[(0, 2), (1, 4), (2, 7), (3, 3), (4, 0), (5, 6), (6, 1), (7, 5)]
[(0, 2), (1, 5), (2, 1), (3, 4), (4, 7), (5, 0), (6, 6), (7, 3)]
[(0, 2), (1, 5), (2, 1), (3, 6), (4, 0), (5, 3), (6, 7), (7, 4)]
[(0, 2), (1, 5), (2, 1), (3, 6), (4, 4), (5, 0), (6, 7), (7, 3)]
[(0, 6), (1, 0), (2, 2), (3, 7), (4, 5), (5, 3), (6, 1), (7, 4)]
[(0, 6), (1, 1), (2, 3), (3, 0), (4, 7), (5, 4), (6, 2), (7, 5)]
[(0, 6), (1, 1), (2, 5), (3, 2), (4, 0), (5, 3), (6, 7), (7, 4)]
[(0, 6), (1, 2), (2, 0), (3, 5), (4, 7), (5, 4), (6, 1), (7, 3)]
[(0, 6), (1, 2), (2, 7), (3, 1), (4, 4), (5, 0), (6, 5), (7, 3)]
[(0, 6), (1, 3), (2, 1), (3, 4), (4, 7), (5, 0), (6, 2), (7, 5)]
[(0, 6), (1, 3), (2, 1), (3, 7), (4, 5), (5, 0), (6, 2), (7, 4)]
[(0, 6), (1, 4), (2, 2), (3, 0), (4, 5), (5, 7), (6, 1), (7, 3)]
[(0, 7), (1, 1), (2, 3), (3, 0), (4, 6), (5, 4), (6, 2), (7, 5)]
[(0, 7), (1, 1), (2, 4), (3, 2), (4, 0), (5, 6), (6, 3), (7, 5)]
[(0, 7), (1, 2), (2, 0), (3, 5), (4, 1), (5, 4), (6, 6), (7, 3)]
[(0, 7), (1, 3), (2, 0), (3, 2), (4, 5), (5, 1), (6, 6), (7, 4)]
```

图 3-3

图 3-3 表示了 8 皇后问题所有解的位置表现。可以看到它的解很多，而且在每一个位置上它所拥有的解的数量都是不同的，该算法能很好的找出 8 皇后的所有解

四、 算法应用

CSP 与其相关算法广泛的应用于各个领域之中。不仅仅是在人工智能或者计算机领域，在生活中应用也非常广泛，像医疗，交通，运输，生产调度，物流等都有 CSP 相关技术的应用

1.农田灌溉规划问题

随着西北大型灌区水资源供需矛盾日益加剧,粮食生产安全逐渐存在重大威胁,传统优化渠系配水模型主要基于目标函数对作物需水量进行优化配置,而目标函数参数及约束条件较为复杂,难以实现全局优化配水，在干渠设计流量恒定条件下,以剩余流量最小原则建立回溯搜索算法优化配水方程,通过回溯搜索算法求解下级渠道阀门开启及关闭配水时间图,并通过多级比较，发现回溯搜索算法在渠系水利用率较低的西洞渠系弃水为零,而在渠系水利用率较高的小河站渠系弃水现象严重,该算法普遍适用于渠系水利用率较低地区.利用回溯搜索算法对灌区进行优化配水,在合理适用条件下,不仅能够保证灌溉时间较优,满足渠系灌溉制度要求,而且能够维持流体输送相对稳定性,达到渠系优化配水目的。

2．高效调度优化

排课管理作为高校教学管理和实施的重要工作之一,是维持高校正常教学秩序的前提,其涉及范围广、受限制的条件多,排课时需考虑上课时间、教学场所、教师及学生等多方面因素。传统的手工排课方式容易造成排课因素冲突、教学资源分配不合理、管理效率低等问题,影响了教学工作的正常运转,已无法满足现有教学管理的需要。因此，利用回溯算法来对课程的调度进行优化,回溯法的搜索过程可使用剪枝函数避免无效的搜索。剪枝函数包含两类:约束函数和限界函数,可剪去互斥约束条件的路径和不能得到最优解的路径,从而提高搜索效率。研究回溯算法的基础上将之优化并应用到高职学院排课系统的开发过程中，效果良好,提高了排课工作的运行效率,用户的满意度也同时得到了提高。

3.经济学实验分组

经济学的许多实验中都会涉及排列组合的分组问题，比如独裁者博弈、公共物品实验，以及众多经典的博弈类实验．在这些实验中，设计者需要将

参与者按照一定数量进行分组，在每轮中会多组同时实验，并进行多轮，一些实验还要求几个参与者被分人同一组的情况只出现一次. 解决这一问题，约束满足问题提供了一个很好的思路，利用 CSP 对此类分组实验进行分析规制，将分组形式拆解，对任意轮次的任意组中的每个空位构造相对应的值域(可填人此位置的参与者范围)以及限制条件，然后使用回溯算法解决问题，将经济学经典实验中分组问题归类为 CSP，并将问题拆解成变量集、值域集以及约束条件集的形式，并使用回溯算法实现分组。很好的达到了目标要求。

五、 结论与展望

因为自己算法掌握得不是很熟练，在算法的理解及验证（读代码）方面花费了大量的时间，所以本文并没有做过多深入的拓展，更多的是根据讲义内容， 简要了解了回溯算法及约束传播的相关问题，并结合相关资料做了基本实现及验证，同时，通过了解一些具体的应用，加深了 CSP 相关算法的认识和学习，在学习的过程中，我也认识到，该算法不仅在传统领域应用广泛，在目前比较火热的机器学习，人工智能领域也有着不可获缺的作用。因此，本文在扎实自己算法功底的基础上，希望能给自己以后机器学习提供一些创新思路，为以后的学习做一些参考。

六、 参考文献

[1] Stuart J. Russell， Peter Norvig，殷建平等译。《人工智能：一种现代的方法》，第 3 版，清华大学出版社。

[2]ConstraintSatisfactionProblem.https://zh.wikipedia.org/wiki/Constraint_satisfaction_problem

[3] 陈丹琳.基于 CSP 方法的经济学实验分组问题的设计与实现[J].天津师范大学学报（自然科学版）,2019,39(6):67-70,76.

[4] 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,2012,23(7):1816-1823.

[5] 蒋李鸣,吕佳宇,何哲华.基于启发式的约束满足问题 AC 系列算法改进研究[J].软件工程,2018,21(2):30-34.

[6] 张永刚,程竹元.最大受限路径相容约束传播算法的研究进展[J].计算机科学,2018,45(z1):41-45,62.

[7] 王海燕,欧阳丹彤,张永刚, 等.基于 AC 与 LmaxRPC 的自适应约束传播求解算法[J].湖南大学学报（自然科学版）,2013,40(7):86-91.

[8] 王腾飞,徐周波,古天龙.弧一致性符号 ADD 算法及在 CSP 求解中的应用[J].计算机科学,2013,40(12):243-247.

[9] 杜小勤.《人工智能》 课程系列:约束满足问题,2020/7/31

[10] 韩宇,孙志鹏,黄睿, 等.基于回溯搜索算法的灌区优化配水模型[J].工程科学与技术,2020,52(1):29-37.

[11] 柯红香.基于优先度与回溯算法的高职院校排课系统设计与实现[J].信阳农林学院学报,2019,29(3):93-96.

[12] 鲍康胜.从八皇后问题引发递归回溯算法的思考[J].电脑编程技巧与维护,2019,(5):32-34.

[13] 侯彦彪,杨天军,张爱民, 等.约束满足技术及其在维修性冲突检测中的应用[J].兵工自动化,2007,26(10):70-71,74.