《人工智能》课程设计报告

选题名称：基于 CNF 求解器的数独游戏

专业班级：CS1705

学　　号：U201714707

姓　　名：练炳诚

指导教师：冯琪

报告日期：2020 年 7 月 30 日

计算机科学与技术学院

目录

1. 摘要

数独游戏(Sudoku Game)是一种运用纸、笔进行演算的逻辑游戏。玩家需要根据 9×9 盘面上的已知数字，推理出所有剩余空格的数字，并满足每一行、每一列、每一个粗线宫（3*3）内的数字均含 1-9，不重复。数独游戏按其逻辑规律可以翻译成合取范式（conjunctive normal form,CNF），进而转化为命题逻辑公式的可满足性问题（satisfiability problem,SAT）。SAT 问题在计算机理论研究和人工智能领域有着重要作用。

本次课程设计中，实现了基于“种子棋盘”的魔方转动法和挖洞法产生非完备的数独初盘，以及将数独初盘转换为 CNF 的算法，玩家可通过图形化界面体验数独游戏，还实现了基本的 DPLL 算法求解 CNF 的可满足性并，通过一些启发式规则进行了算法优化，最后将求解的结果映射到数独游戏棋盘上形成数独终盘。

2. 问题描述

2.1 数独求解问题的知识表示

数独求解的问题可用命题逻辑进行表示，数独格局满足的约束条件如下[6]：

(1)每个空格中每个数字至少出现一次；

(2)每个数字在每行中最多出现一次；

(3)每个数字在每列中最多出现一次；

(4)每个数字再每个 3*3 方格中最多出现一次。

将上述约束条件翻译为命题，以 Sxyz 表示 x 行 y 列为数字 z，则对应的命题如下：

(1)$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}S_{xyz}$

(2) $\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{y=1}^{8}\bigwedge_{i=y+1}^{9}(\neg S_{xyz} \vee \neg S_{xiz})$

(3) $\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigwedge_{x=1}^{8}\bigwedge_{i=x+1}^{9}(\neg S_{xyz} \vee \neg S_{iyz})$

(4) $\bigwedge_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=y+1}^{3}(\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+x)(3j+k)z})$,

$\bigwedge_{z=1}^{9}\bigwedge_{i=1}^{2}\bigwedge_{j=1}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{k=x+1}^{3}\bigwedge_{y=1}^{3}\bigwedge_{l=1}^{3}(\neg S_{(3i+x)(3j+y)z} \vee \neg S_{(3i+k)(3j+l)z})$

上述命题中的析取逻辑可等价地变为合取逻辑，得到的 CNF 子句共有 81+3*9*9*C(9,2)=81 + 3*9*9*36=8829 条，变元编号范围为[1,729]，按顺序表示的命题是第 1 行 1 列填 1，第 1 行 1 列填 2…即每个空格用 9 个变元的取值唯一确定填入的数字，每个变元取正表示命题为真，取负表示命题为假。

这样，就将一个数独的求解问题表示成了关于 729 个变元的 CNF 的可满足性问题，对这些 CNF 的可满足性的求解构成了对数独游戏的求解，由数独解的存在唯一性可知，当对 729 个变元完成赋值后，即完成了数独的求解。采用上述知识表示时，729 个变元中，只有 81 个变元赋值为正，且从 1 到 729 号变元，每 9 个变元仅 1 个为正，其余变元为负。

变元编号 n 与“第 x 行第 y 列填 z”的转换关系如下：

$$n=(x-1)*81+(y-1)*9+z$$
$$x=\lceil n/81\rceil$$
$$y=\lceil (n\%81)/9\rceil$$
$$z=n\%9$$

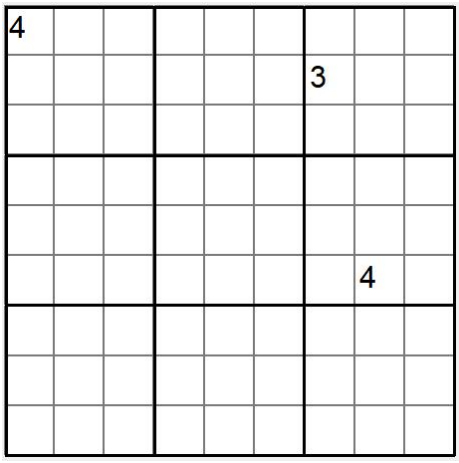

图 2-1 数独初盘中填入的数字与变元取值关系

例如，对图 2-1 所示的数独初盘，初始变元值为正的分别是第 4 号（第 1 行第 1 列填 4），第 138 号（第 2 行第 7 列填 3），第 472 号（第 6 行第 8 行填 4），其余变元均未取值或为负值。

2.2 CNF-SAT 求解问题的知识表示

对 CNF 的求解过程实际上是对 SAT 问题的求解过程，下面对 SAT 问题进行定义[3]。

CNF-SAT：设 B 为布尔变量集合 U 上的合取范式，判断是否有一组真值指派使得合取范式 B 中所有子句都是可满足的（即 B 中所有子句的取值都为真）。根据命题逻辑的基本知识可知，所有的命题逻辑公式都可以转化为 CNF 公式的形式。

以下是 SAT 问题中对 CNF 公式进行化简时常用的一些化简规则：

（1）单子句规则（单元子句规则）

若一个子句中仅有一个文字未赋值，所有其它的文字赋值都是 0（或者该子句本身只含有一个文字），那么此时可推断出该文字赋值为 1。

（2）纯文字规则

在一个合取范式中，若一个命题变量要么以全是正文字的形式出现，要么以

全是负文字的形式出现，那么该文字就称为纯文字。消除这种所有出现即为正或出现即为负的原子公式所在子句的规则就称为纯文字规则。

例如：CNF 公式的子句集

$$\varphi = (\neg p \vee q \vee r \vee \neg t) \wedge (p \vee \neg q \vee \neg r \vee \neg t) \wedge (\neg p \vee \neg q)$$

其中$\neg t$ 是纯文字，则公式φ 等价于$\varphi' = (\neg p \vee \neg q)$

（3）消除原子公式规则

假设给定的合取范式或一部分子式可以表示为以下这种形式：

$$(P \vee u) \wedge (Q \vee \bar{u}) \wedge R$$

其中， P 、Q 、R 都是析取子式，u 和 $\bar{u}$ 不在其它子句中出现，且 P 、Q 、R 都与原子公式 u 无关，那么该公式可以消减为不含原子公式 u 的形式：

$$(P \vee Q) \wedge R$$

（4）重言子句规则

如果一个子句是重言子句（一般是子句中存在互补的文字），那么就可以将它从合取范式中删除。

（5）包含规则

如果A_1，A_2 是合取范式 B 的两个子句，且 $A_1 \subseteq A_2$，则求合取范式 B 的可满足性问题时可以直接将子句 A_1 删去。

（6）分裂规则

设CNF 公式的子句集为：

$$\varphi = (A_1 \vee r) \wedge K \wedge (A_m \vee r) \wedge (B_1 \vee r) \wedge K \wedge (B_n \vee \neg r) \wedge K \wedge C_1 \wedge K \wedge C_t$$

其中A_i，B_j，C_k 均是不含文字 r 或者 $\bar{r}$ 的子句。此时公式φ 可以分裂成两个不含 r 或者 $\bar{r}$ 的子句集：

$$\varphi_1 = A_1 \wedge K \wedge A_m \wedge C_1 \wedge K \wedge C_t$$

$$\varphi_2 = B_1 \wedge K \wedge B_n \wedge C_1 \wedge K \wedge C_t$$

对 CNF 公式中每个变元进行赋值判断满足性的过程实际上是一个解空间的搜索过程，其搜索过程如图 2-2 所示。

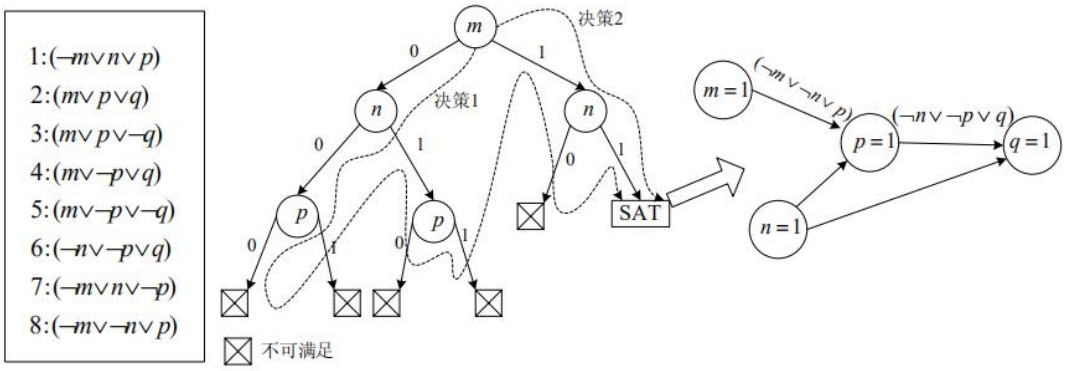

图 2-2 SAT 问题求解示例

对 SAT 问题，可采用 DPLL 算法进行求解，将在 3.3 节中进行介绍。

3. 算法选择及实现

3.1 数独初盘形成算法

一个数独棋盘的所有组合有 6,670,903,752,021,072,936,960（约为 6.67×10 的 21 次方）种[5]，要设计一个完备的数独棋盘生成算法是较为困难的，产生过程必须是随机的，且每填入一个数字，都需要判断是否冲突，时间开销较大。本次课程设计中采用了一种非完备的、简单有效的数独棋盘生成算法。

基于“种子棋盘”的魔法转动法生成数独游戏终盘。“种子棋盘”即为一个初始的数独游戏终盘，第一行为数字 1~9，第四行、第七行每一列为上一行该列加 1 对 9 取余，其余行每一列为上一行该列数字加 3 对 9 取余。最后将所有 0 替换为 9，得到种子棋盘。这样生成的种子棋盘如图 3-1 所示。

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
8	9	1	2	3	4	5	6	7
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8
3	4	5	6	7	8	9	1	2

图 3-1 一种“种子棋盘”的表示

“种子棋盘”是一个合法的数独终盘，每行、每列、每个宫格里的数字均不重复。故将任意两行、任意两列间的数进行交换，不会破坏行、列直接的唯一性，但宫格内的唯一性可能会被破坏。若将行、列之间的交换限制在一个宫格内，即将行分为 1~3、4~6、7~9 三组，列同样分组，每组内可任意进行行、列交换，这样就保证了宫格内的唯一性。而每个宫格内的交换过程与魔方进行转动时方块的变化过程相同，故将这种方法称为魔方转动法。

这种方法产生的数独棋盘组合远远小于完备的算法产生的结果，但对于一般的数独游戏来说，已经足以使用。

在产生数独棋盘后，还要去除其中的部分数字产生数独初盘，剩下的数字作

为提示让玩家能够推理出数独终盘。

数独游戏提示数的多寡与难易并无绝对关系，多提示数比少提示数难的情况屡见不鲜，同时也存在增加提示数之后题目反而变难的情形，即使是相同提示数（甚或相同谜题图形）也可以变化出各式各样的难度。提示数少对于出题的困难度则有比较直接的关系，以 20~35 提示数而言，每少一个提示数，其出题难度会增加数倍，在制作谜题时，提示数在 22 以下就非常困难，所以常见的数独题其提示数在 23~30 之间。

设计去掉的数字数量与游戏难度的关系如下：

（1）30~39：简单

（2）40~49：中等

（3）50~59：困难

对产生的数独终盘，随机选择位置将数字去除，这样的方法称为挖洞法。根据选择的游戏难度，采用随机算法获取一个固定的挖洞数，再采用随机算法进行挖洞，就产生了如图 3-2 所示的数独初盘。

			7				2	
2		3			5			7
8	7	9	1		2	6		4
9			2					
	5							
	2	4	5	7			9	
				5			7	
7		8		2			4	
4	3		6		7	2		9

图 3-2 挖洞法产生的数独终盘

3.2 数独求解问题转换 CNF 算法

按照 2.1 节中的规则，每个数独谜题的求解问题都能转换为 8829 条 CNF 子句，另外对已填入的数字，还要产生已填入数字的 CNF 子句（仅含变元的单子句，取值为 1）根据规则（1）~（4），转换 CNF 算法的伪代码描述如下。

```
//genCNF()表示生成合取范式
initial board[1..9][1..9] //board 为数独棋盘的二维矩阵表示
//将已填入的数字转换为单子句
for i = 1 to 9
   for j = 1 to 9
      if board[i][j]  已填入数字
         genCNF((i-1)*81 + (j-1)*9 + board[i][j])
//规则（1）：每空至少填一个数字
for x = 1 to 9//每行
   for y = 1 to 9//每列
      for z = 1 to 9//每个数字
         genCNF((x-1)*81 + (y-1)*9 + z)
//规则（2）：每列每个数最多出现一次
for x = 1 to 9//每行
   for z = 1 to 9//每个数字
      for m = 1 to 8//前一列
         for n = m+1 to 9//后一列
            genCNF(-(81*(x-1)+9*(m-1)+z),-(81*(x-1)+9*(n-1)+z))
//规则（3）：每行每个数最多出现一次
for y = 1 to 9//每列
   for z = 1 to 9//每个数字
      for m = 1 to 8//前一行
         for n = m+1 to 9//后一行
            genCNF(-(81*(m-1)+9*(y-1)+z),-(81*(n-1)+9*(y-1)+z))
//规则（4）：每个宫格中每个数最多出现一次
for z = 1 to 9//每个数字
   for i = 0 to 3//宫格行
      for j = 0 to 3//宫格列
         for x = 0 to 3//宫格内行
            for y = 0 to 3//宫格内前一列
               for k = y+1 to 3//宫格内后一列
                  genCNF(-(81*(3*i+x)+9*(3*j+y)+z),-(81*(3*i+x)+9*(3*j+k)+z))
for z = 1 to 9//每个数字
   for i = 0 to 3//宫格行
      for j = 0 to 3//宫格列
         for y = 0 to 3//宫格内列
            for x = 0 to 3//宫格内前一行
               for k = x+1 to 3//宫格内后一行
                  for l = 0 to 3
                     genCNF(-(81*(3*i+x)+9*(3*j+y)+z),-(81*(3*i+k)+9*(3*j+l)+z))
```

3.3 DPLL 算法及启发式策略

对 CNF 公式中的文字进行真值赋值所得出的搜索空间可以用一棵二叉树来表示，树中的每个节点对应一个命题变量，取值只能为 1 和 0，左右子树表示该变量取真值和假值的分支，从二叉树中根节点到叶子节点的一条路径就表示 CNF 公式中的一组变量赋值序列[3]。基于 DPLL 的算法都是对这棵二叉树从根节点开始进行深度优先搜索(DFS)遍历所有的通路，以找到使问题可满足的解。

设变量集合为 U ，该变量集合上的变量组成合取范式 B ，则有可满足性问题(B,U)和一个变量赋值顺序 ，变量集合 U 上的所有可能的真值赋值在下会构成一棵二叉树的搜索空间。

举例说明，设 $U=\{x_1, x_2, x_3\}$，$\{\beta_1(U)=x_1, \beta_2(U)=x_2, \beta_3(U)=x_3\}$，那么真值赋值所构成的搜索空间如图 3-3 所示。

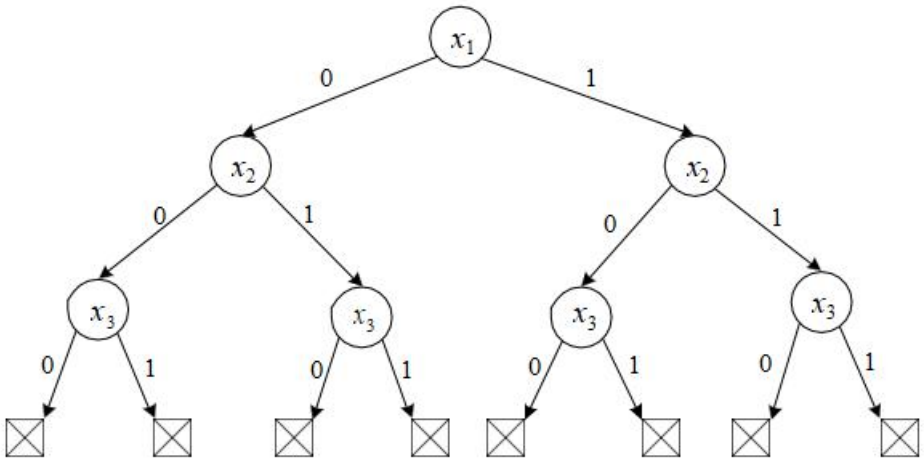

图 3-3 变量的二叉搜索空间

容易知道，这课二叉树的叶子节点只能有两种：

（1）子句集取真值，该问题是可满足的。

（2）子句集中含有一个或多个空子句，该问题不可满足。

原始 DPLL 算法的时间复杂度是 *O(2^n)的*，为加快 DPLL 算法的搜索效率，可采用以下 3 个启发式策略对变量的决策进行优化：

（1）变量决策阶段（即 DECIDE 过程）：在搜索过程的每一分支阶段，选择未赋值的一个变量为其赋值为 0 或 1，该变量称为决策变量，决策变量在赋值时所处的二叉树中的高度称为它的决策层。

（2）推理阶段（也称为 BCP 过程）：每一次已选择变量赋值之后，识别

该赋值所导致的必要的赋值或者根据已有的变量赋值对子句进行化简，即进行布尔约束传播过程。举例说明，比如子句$(x_1 \vee x_2 \neg x_3)$，如果决策阶段选取x_3为决策变量并令它取值为 0，那么该子句已可满足，就可以将其从子句集中删除，原问题得到了化简。

（3）回溯阶段（BACKTRACK）：推理过程中发生冲突时，使搜索过程从较深的变量决策层返回至较浅的决策层。冲突即指 BCP 过程中，至少出现了一个子句不可满足的情况。

采用启发式策略的 DPLL 算法的流程图如图 3-4 所示。

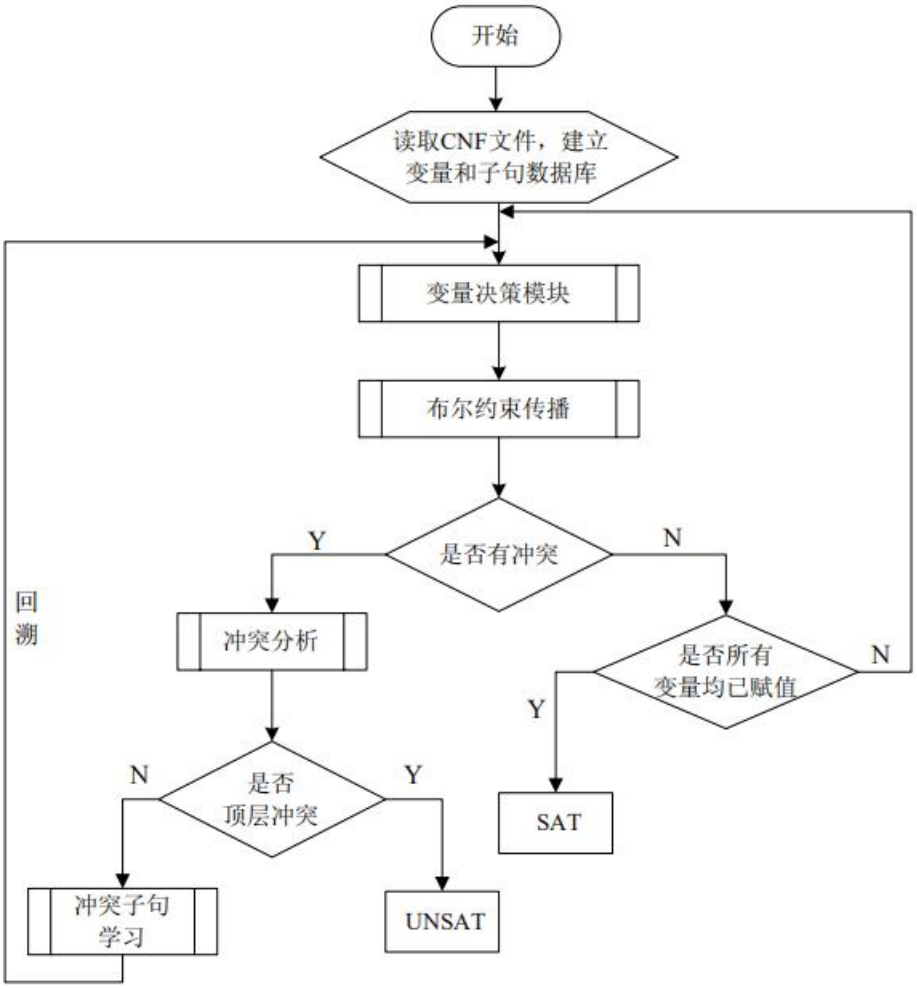

图 3-4 基本的 DPLL 算法流程图

DPLL 算法就是循环使用上文中提到的单子句规则、纯文字规则、分裂规则、重言子句规则和包含规则等化简策略对不断地对 CNF 公式进行化简直到求得问题的解的过程。最原始的 DPLL 算法可用下述伪代码来描述：

```
let B : SAT 问题的子句集
DPLL(B)
  while true
    if  B 中存在单子句 C
      对该子句中的唯一文字 β进行赋值，然后利用 β化简子句集 B
    if  B 为空 //命题可满足
      return true
    if  !variable_decide  //有未进行赋值的变量
      从 B 中任选一个没有赋值的变量为其赋值
      return true
    while !bcp_deduce  //BCP 未进行
      利用化简策略对子句集进行化简
      if  conflict  //存在冲突
        选择最近的没有翻转的决策变量 β'
        if  β'为空 所有决策变量均翻转过
          return false
```

4. 代码实现

4.1 开发工具

硬件环境：PC 机，Intel Core i7 八核 CPU 2.8GHz,16G 内存；

系统环境：Microsoft Windows 10 系统;

项目开发平台：Qt Creator 5.6;

代码框架：QT 5.6;

4.2 关键数据结构说明

数独初盘形成算法中，“种子棋盘”采用静态二维数组保存到静态变量区，不允许用户修改；由种子棋盘生成的数独初盘保存到堆栈段，采用二维数组表示，因为数独棋盘中不会出现数字 0，故可用数字 0 表示挖洞法挖去的数字。

数独求解问题转换 CNF 算法中，定义.cnf 文件的格式如图 4-1 所示。

```
p cnf 729 8869
46 0
117 0
136 0
155 0
165 0
173 0
195 0
```

图 4-1 .cnf 文件格式

其中，第一行说明了文件是.cnf 文件，后面的两个数字 x、y 表示变元的数量和 CNF 子句的数量，接下来的 y 行是一些以数字 0 结束的数字串，每个数字表示变元的编号，取值为正表示变元取真，取值为负表示变元取假。

DPLL 算法中，介绍了 CNF 子句集、CNF 子句、变元（文字）的概念，三者的集合关系如图 4-2 所示。

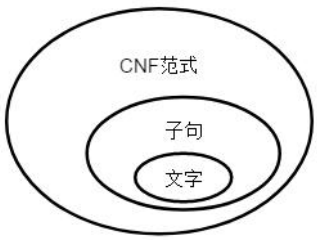

图 4-2 集合关系

即单个文字既是一条子句，又可以构成一个 CNF 公式。综合考虑三者的关系，可以将 CNF 公式的数据结构设计为一个邻接表，其中顺序表部分为范式中的每条子句，而链表部分为每条子句中所含的文字。其逻辑结构如图 4-3 所示。

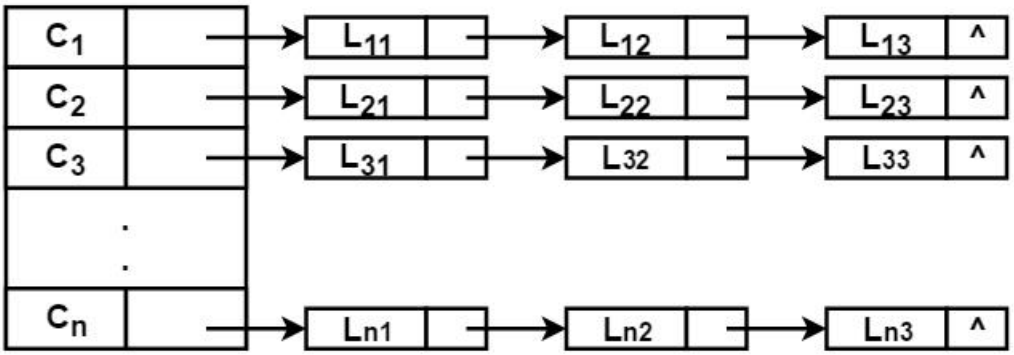

图 4-3 CNF 子句集存储结构图

这样一来，搜索CNF子句集中的子句可按顺序或随机查找的方式，而搜索每条子句中的文字时可按顺序查找的方式。考虑到DPLL算法过程会对子句集内各子句、文字有赋值操作，对每个子句存储单元和文字存储结点，内部还要有表示其赋值状态的变量等。因此子句的数据应该用一个结构体来存储，结构体内部必须包含的内容有该子句的赋值状态、指向其文字链表的指针。文字的数据也用一个结构体来存储，结构体内部必须包含的内容有该文字的赋值状态、指向下一文字结点的指针。

DPLL算法过程中，每当对一个文字进行赋值时，就将该文字作为一个单子句加入到子句集中，进行BCP传播和SAT判断，并进行自身递归调用。由于该算法是基于回溯的，因此每次递归前要保存该CNF子句集的状态，若每次递归返回时不释放掉中间用来保存状态的内存空间，则会存在内存耗尽的问题。考虑到这一情况，用堆栈表示每次决策前的状态，如图4-4所示。

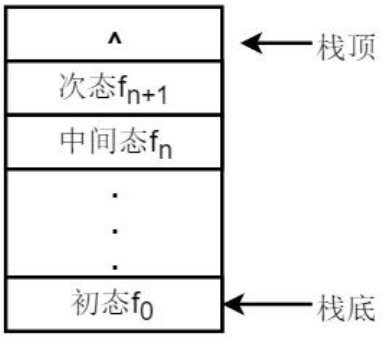

图 4-4 状态栈结构示意图

每次回溯时，通过堆栈中的状态信息对CNF子句集进行覆写即可恢复决策层状态。

在DPLL算法执行过程中，会涉及对文字的选取，因此需要一个顺序表来存放该公式的所有文字和文字的信息，如出现频率、赋值状态、赋值次数等。顺序表的长度为文字个数*2，(正负文字分别存放)，如图4-5所示。

图 4-5 文字存放顺序表示意图

正负文字 literal 的下标θ转换公式如下：

$$\theta = \text{literal} - 1 \ , \text{literal} > 0;$$

$$\theta = -\text{literal} + \text{variablenum} - 1, \text{literal} < 0$$

其中，variblenum 为该公式的变元最大值。

5. 结果演示

数独游戏启动后界面如图 5-1 所示。

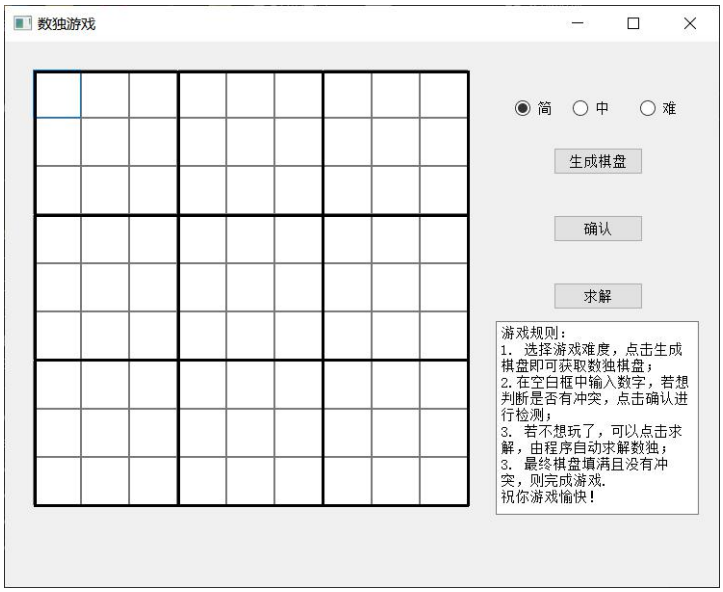

图 5-1 数独游戏界面

选择难度为“中”，点击生成棋盘，如图 5-2 所示。

图 5-2 中等难度数独初盘

可以看到第 1 宫中的(1,)位置可以填入数字 4，若在第 1 宫的(1,3)位置填入 2 会与第 3 宫内的数字 2 出现行冲突，如图 5-3 所示。

图 5-3 冲突提示

冲突提示后，用户输入的数字 2 变为红色，如图 5-4 所示。

图 5-4 冲突数字变为红色

解完数独后，提示用时，如图 5-5 所示。

图 5-5 数独解完提示

另外新建一个数独初盘，点击求解，在当前文件目录下产生 sudoku.cnf 文件和 sudoku.out 文件，分别为数独求解转换的 CNF 子句集和求解结果，两个文件的内容和求解后的棋盘分别如图 5-6、图 5-8、图 5-9 所示。

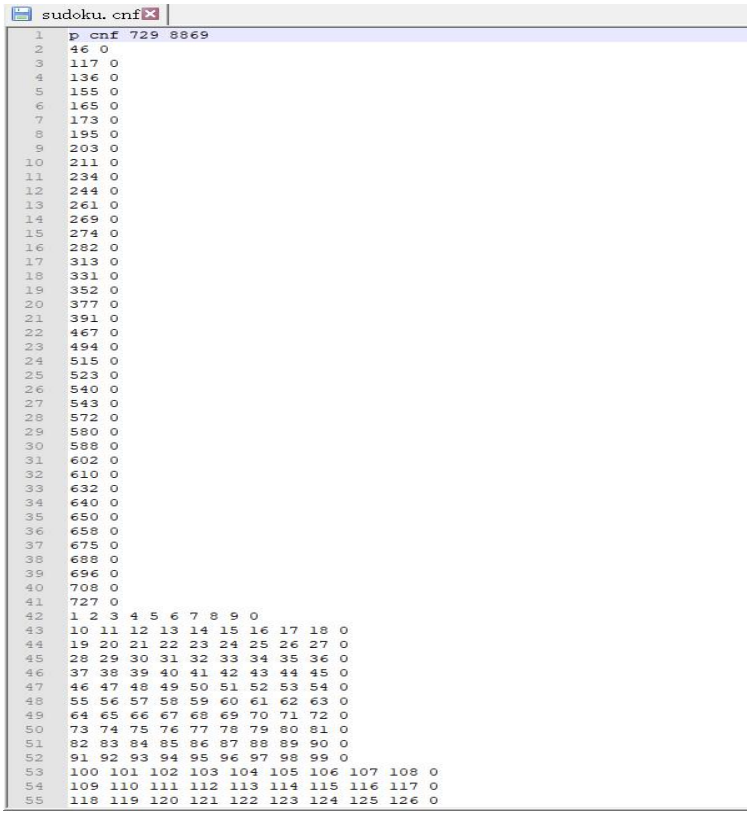

图 5-6 sudoku.cnf 文件内容

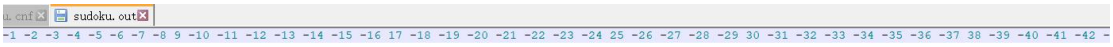

图 5-7 sudoku.out 文件内容

图 5-8 数独求解结果

6. 小结及展望

在人工智能课程设计中，我学会了将问题通过知识表示进行描述，同时也了解到 SAT 问题在人工智能领域的重要性，通过阅读文献学会了将数独求解问题转换为 CNF-SAT 问题，并实现了基于 DPLL 算法的 SAT 求解器，通过一些启发式策略对基本的 DPLL 算法进行了优化，最后将求解结果反映到数独棋盘的自动求解功能中。

在 UI 界面模块，设计了简易的用户操作逻辑，用户的非冲突输入能够显示为蓝色，冲突输入显示为红色，用户能够选择数独棋盘的难度，有计时机制和自动求解机制，具有可玩性。

但由于开发经验和知识的欠缺，以及时间的不足，在 DPLL 算法的优化上没有更进一步，后期可以添加子句学习、非时序回溯、随机重启动等机制[3]来进一步优化 DPLL 算法。

总之，这次课设让我对人工智能领域的一些算法有所了解，希望以后可以更进步学习更多的算法及应用。

参考文献

[1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社，2000

[2]Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University,Canada,2009

[3] 陈稳. 基于 DPLL 的 SAT 算法的研究与应用.硕士学位论文，电子科技大学，2011

[4]Carsten Sinz.Visualizing SAT Instances and Runs of the DPLL Algorithm.J Autom Reasoning (2007) 39:219–243

[5] 360 百科：数独游戏 https://baike.so.com/doc/3390505-3569059.html

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic forProgramming, Artificial Intelligence and Reasoning, LPAR 2005, pages 11–15, 2005.

[7]Ins Lynce and Jol Ouaknine. Sudoku as a sat problem.In Proceedings of the 9th InternationalSymposium on Artificial Intelligence and Mathematics, AIMATH 2006, Fort Lauderdale.Springer,2006.

[8] Uwe Pfeiffer, Tomas Karnagel and Guido Scheffler. A Sudoku-Solver for Large Puzzles using SAT. LPAR-17-short (EPiC Series, vol. 13), 52–57

[9] Sudoku Puzzles Generating: from Easy to Evil. http://zhangroup.aporc.org/images/files/Paper_3485.pdf

[10] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing(SAT 2015),223-237360