華中科技大學

课程设计报告

题目：基于 SAT 的对角线数独游戏求解程序

课程名称：程序设计综合课程设计
专业班级：本硕博 2301 班
学　　号：U202315763
姓　　名：王家乐
指导教师：向文
报告日期：2024/9/10

计算机科学与技术学院

目录

任务书

□ 设计内容

SAT 问题即命题逻辑公式的可满足性问题（satisfiability problem），是计算机科学与人工智能基本问题，是一个典型的 NP 完全问题，可广泛应用于许多实际问题如硬件设计、安全协议验证等，具有重要理论意义与应用价值。本设计要求基于 DPLL 算法实现一个完备 SAT 求解器，对输入的 CNF 范式算例文件，解析并建立其内部表示；精心设计问题中变元、文字、子句、公式等有效的物理存储结构以及一定的分支变元处理策略，使求解器具有优化的执行性能；对一定规模的算例能有效求解，输出与文件保存求解结果，统计求解时间。

□ 设计要求

要求具有如下功能：

（1）输入输出功能：包括程序执行参数的输入，SAT 算例 cnf 文件的读取，执行结果的输出与文件保存等。(15%)

（2）公式解析与验证：读取 cnf 算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)

（3）DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。(35%)

（4）时间性能的测量：基于相应的时间处理函数（参考 time.h），记录 DPLL 过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)

（5）程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：$[(t-t_o)/t]*100\%$,其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)

（6）SAT 应用：将数双独游戏[5]问题转化为 SAT 问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献[3]与[6-8]。(15%)

1 引言

1.1 课题背景与意义

1.1.1 SAT

SAT 问题即布尔可满足性问题，是确定是否存在满足给定布尔公式解释的问题。如果对于给定的布尔公式变量可以一致地用 TRUE 或者 FALSE 替换，那么该布尔公式可满足，相反则不满足。

SAT 问题是计算机科学与人工智能领域的经典问题，研究成果广泛应用于电子设计自动化，人工智能等领域，因此研究 SAT 问题有助于拓展知识面以及今后的实际应用。

1.1.2 数独游戏

数独游戏的历史渊源比较久远，数独是一种源自 18 世纪末的瑞士，后在美国发展并在日本得以发扬光大的数学智力拼图游戏。

早在数千年前，中国人就发明了九宫图:在 9 个方格中，横行和竖行的数字总和是相同的。"数独"也不是什么新生事物，已经存在了数百年。18 世纪，瑞士数学家莱昂哈德·欧勒发明了"拉丁方块"，但并没有受到人们的重视。直到 20 世纪 70 年代，美国杂志才以"数字拼图"的名称将它重新推出。日本随后接受并推广了这种游戏，并且将它改名为"数独"，大致的意思是"独个的数字"或"只出现一次的数字"。

现今流行的数独于 1984 年由日本游戏杂志《パズル通信ニコリ》发表并得了现时的名称。数独本是"独立的数字"的省略，因为每一个方格都填上一个非零的个位数。数独冲出日本成为英国当下的流行游戏，得归功于曾任香港高等法院法官的高乐德(Wayne Gould)。2004 年，他在日本旅行的时候，发现杂志上介绍的这款游戏，便带回伦敦向《泰晤士报》推介并获得接纳。英国《每日邮报》也于三日后开始连载，使数独在英国正式掀起热潮。数独不仅是报章增加销量的法宝，脑筋动得快的《泰晤士报》还做起手机族的生意，花 4.5 英镑就能下载 10 则数独游戏到手机上玩。渐渐，其他国家和地区受其影响也开始风靡数独。

同类似的填字游戏不同，数独受欢迎的原因之一是它既不需要丰富的百科知识，也不要掌握大量的词汇，这使其能迅速为孩子和初学者所接受。根据游戏开始时的方格中已有的数字和位置，数独难易程度不同，有些复杂的甚至令数学家

也不能完成。据著名的动游戏开发商 Astraware Ltd.预计，移动数独游戏的版本多达几十种，Palm 和 Windows Mobile 设备版本的数独游戏就各有 20 种左右。Sudokumo 推出的移动数独游戏，能够下载到大多数手机中。这家位于英国的游戏软件公司表示，已经在全球卖出了 7500 套数独游戏，而且来自用户的兴趣还在增加。

1.2 国内外研究现状

求解 SAT 问题的经典算法——DPLL 算法，它在 1962 年由马丁·戴维斯、希拉里·普特南、乔治·洛吉曼和多纳·洛夫兰德共同提出，作为早期戴维斯-普特南算法的一种改进。戴维斯-普特南算法是戴维斯与普特南在 1960 年发展的一种算法。DPLL 是一种高效的程序，并且经过 40 多年还是最有效的 SAT 解法，以及很多一阶逻辑的自动定理证明的基础。

在之后 Bart Selman 和 Henry Kautz 分别于 1997 年和 2003 年在人工智能第五届国际合作会议上提出了 SAT 问题面临的十大挑战性问题，并在 2001 年和 2007 年先后 对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出对 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。这使得越来越多的人开始关注并研究 SAT 问题，所以这段时间也涌现出了众多新的高效的 SAT 算法如 MINISAT、SATO、CHAFF、POSIT 和 GRASP 等，SAT 算法的研究成果显著，求解算法也越来越多地应用到了实际问题领域。这些 新兴的算法大都是基于 DPLL 算法的改进算法，改进的方面包括：采用新的数据结构、新的变量决策策略或者新的快速的算法实现方案。国内也涌现出了许多高效的求解算法，如 1998 年梁东敏提出了改进的子句加权 WSAT 算法，2000 年金人超和黄文奇提出了并行 Solar 算法，2002 年张德富提出了模拟退火算法。

SAT 国际竞赛从 2002 年开始每隔一到两年举办一次，这也极大地推动了 SAT 问题的研究，由此可见 SAT 求解问题仍在继续被人们所探索。

1.3 课程设计的主要研究工作

1. 首先对 DPLL 算法，SAT 求解问题的背景，原理进行深入了解，根据相关资料对于项目做一个整体的设计；

2. 设计相应数据结构与算法来完成基于 DPLL 的 SAT 求解器的实现，并用提供的算例做相应测试；

设计程序要求模块化，程序源代码进行模块化组织。主要模块包括如下：

（1） 主控、交互与显示模块(display)

（2） CNF 解析模块(cnfparser)

（3） 核心 DPLL 模块(solver)

（4） 数独模块,包括游戏格局生成、归约、求解(sudoku)

3. 从改变存储结构或者选取文字策略等方面来实现算法的优化，设计测试方案来总结优化的效果；

4. 设计问题转化策略将数独问题归约为 SAT 问题并求解。

2 系统需求分析与总体设计

2.1 系统需求分析

基于 DPLL 算法的 SAT 求解器可以求解部分布尔算例，可以对 CNF 文件进行解析和结果输出；

创建随机的对角线数独游戏并且具有一定的交互性，可以用来求解和自主求解对角线数独游戏。

2.2 系统总体设计

系统总体设计分为两个大的模块：基于 DPLL 算法的 SAT 求解器和对角线数独游戏，各自模块下的功能大致介绍如下：

1. 基于 DPLL 算法的 SAT 求解器：

（1） CNF 的读取解析，遍历输出，保存；

（2） DPLL 求解，计算求解时间并显示，将结果保存到同名.res 文件里。

2. 对角线数独游戏：

（1） 创建随机数独游戏（自行输入初始提提示数个数）,并可以实现一定程度上的交互；

（2） 将数独格局转化为 CNF 文件并调用 DPLL 进行求解，再可视化地将结果打印到屏幕上。

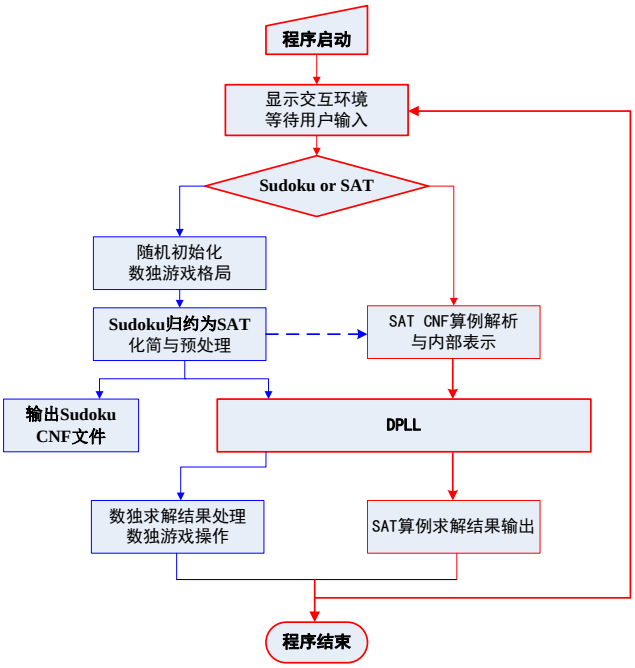

图 2.2-1 系统程序流程图

3 系统详细设计

3.1 有关数据结构的定义

1. CNF 文件读取和处理相关结构体

（1）对于文字节点需要一个结构体 literalNode 来定义：数据项有 int 型的文字值 literal；literalNode*型的指向下一个文字节点的指针 next。

（2）对于子句节点需要一个结构体 clauseNode 来定义：数据项有 literalNode*型的指向该子句第一个文字节点的指针 head；clauseNode*型的指向下一个子句节点的指针 next。

（3）用一个结构体 cnfNode 来定义 CNF 范式链表，用于存储 CNF 文件信息：数据项有 int 型的布尔变元个数 boolCount；int 型的子句个数 clauseCount；clauseNode*型的指向第一个子句节点的指针 root。

表 3.1-1 对 CNF 文件读取和处理相关结构体

数据名	用途	数据项
literalNode	文字节点	int literal; struct literalNode* next;
clauseNode	子句节点	struct literalNode* head; struct clauseNode* next;
cnfNode	CNF 范式链表	int boolCount,clauseCount; struct clauseNode* root;

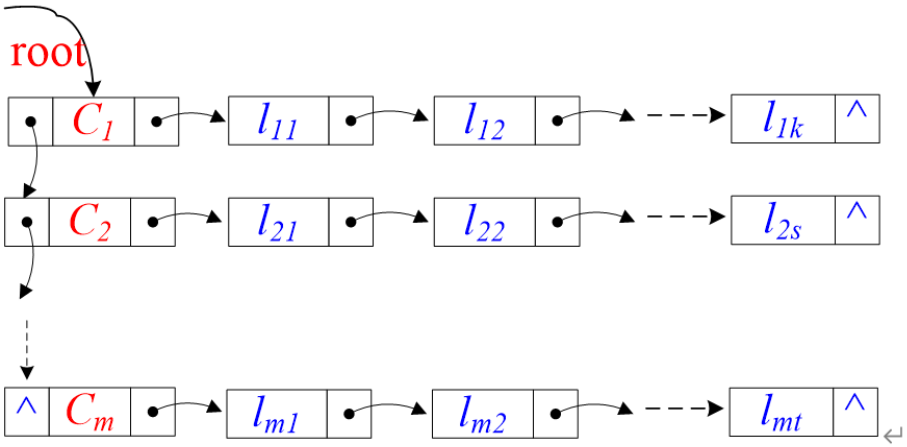

图 3.1-1CNF 文件存储结构图

2. DPLL 过程

该过程需要一个 bool 类型的。线性表(数组)来记录布尔变元的值。

3. X_Sudoku

该部分需要一个 int 类型的数组来记录数独棋盘每个位置的值。

3.2 主要算法设计

3.2.1 CNF 文件解析及处理

1. 根据 CNF 文件创建 CNF 范式链表

status ReadFile(CNF &cnf,char fileName[]);

参数: (CNF) &cnf：指向 CNF 范式链表的指针

(char) fileName：文件名

返回值：OK：创建成功

ERROR：创建失败

打开文件名为 fileName[]的文件后，先读入文件中的无关信息，再读入布尔变元个数和子句个数，随后依次创建子句节点和文字节点。

2. 销毁范式链表

status DestroyCnf(CNF &cnf);

参数：(CNF) &cnf：指向 CNF 范式链表的指针

返回值：OK：销毁成功

ERROR：销毁失败

用 while 循环销毁创建的范式链表，释放各节点存储空间。

3. 遍历范式链表

status PrintCnf(CNF cnf);

参数：(CNF) &cnf：指向 CNF 范式链表的指针

返回值：OK：遍历成功

ERROR：遍历失败

依次输出布尔变元个数，子句个数以及每个子句及文字的信息，检验 CNF 文件的解析与处理是否正确。

3.2.2 DPLL 算法处理

1. 判断是否为单子句

status IsUnitClause(literalList l)；

参数：(literalList) l：指向子句的第一个文字节点的指针

返回值：TRUE：是单子句

FALSE：不是单子句

判断该子句是否只有一个文字节点，是则返回 TRUE，否则返回 FALSE。

2．找到单子句并返回其文字节点的文字值

int FindUnitClause(clauseList cL)；

参数：（clauseList) cL：指向范式链表第一个子句节点的指针

返回值：非 0：找到单子句

0：没有找到单子句

从 cL 开始，依次将每个子句的第一个文字节点传入函数 IsUnitClause()，若值为 TRUE，返回该文字节点的文字；若每个子句都不是单子句，则返回 0。

3．根据选择的文字化简链表

status Simplify(clauseList &cL, int literal)；

参数：(clauseList) &cL：指向范式链表第一个子句节点的指针

(int) literal：要用来化简链表的文字

返回值：OK：化简成功

ERROR：化简失败

从 cL 开始，依次检查每一个子句，若该子句中出现该文字，则删除该子句；若出现该文字的负文字，则删除这个负文字节点。(默认将 literal 的值赋为真)

4．判断当前链表是否满足(DPLL 递归的出口)

status Satisfy(clauseList cL);

参数：(clauseList) cL：指向范式链表第一个子句节点的指针

返回值：OK：满足

ERROR：不满足

若 cL 为空节点，则说明所有子句都已满足，该 SAT 问题已找到解，返回 OK；否则返回 ERROR。

5. 判断当前链表是否有空子句(DPLL 的另一个递归出口)

status EmptyClause(clauseList cL);

参数：(clauseList) cL：指向范式链表第一个子句节点的指针

返回值：TRUE：有空子句

FALSE：没有空子句

从 cL 开始，依次检查每一个子句，若某一个子句的表头为空节点，则表明该子句为空子句，返回 TRUE；若都不为空节点，则表明没有空子句，返回 FALSE。

6. 将当前链表复制一份(用于 DPLL 的一个分支)

clauseList CopyCnf(clauseList cL);

参数：(clauseList) cL：指向范式链表第一个子句节点的指针

返回值：(clauseList)：指向复制的范式链表第一个子句节点的指针

从 cL 开始，用 while 循环依次为每个子句节点和文字节点分配内存，复制成功后返回指向新的范式链表的第一个子句节点的指针。

7. 没有单子句时选择变元的策略(未优化)

int ChooseLiteral_1(CNF cnf);

参数：(CNF) cnf：指向 CNF 范式链表的指针

返回值：(int)型的文字

随机在当前链表(链表中剩下的文字都未被赋值)中选择一个文字。

8. 没有单子句时选择变元的策略(优化一)

int ChooseLiteral_2(CNF cnf);

参数：(CNF) cnf：指向 CNF 范式链表的指针

返回值：(int)型的文字

遍历当前范式链表，找到出现次数最多的文字并将其返回。

9. 没有单子句时选择变元的策略(优化二：MOM 算法)

int ChooseLiteral_3(CNF cnf);

参数：(CNF) cnf：指向 CNF 范式链表的指针

返回值：(int)型的文字

遍历当前范式链表，找到最短的子句中出现的频率最高的文字并将其返回。

10.递归的 DPLL 算法

status DPLL(CNF cnf, bool value[], int flag)；

参数：(CNF) cnf：指向 CNF 范式链表的指针

(bool *) value：用来存布尔变元真值的数组(true or false)

(int) flag：用来控制选择哪种变元选取策略

返回值：OK：DPLL 成功，该 SAT 问题有解

ERROR：DPLL 失败，该 SAT 问题无解

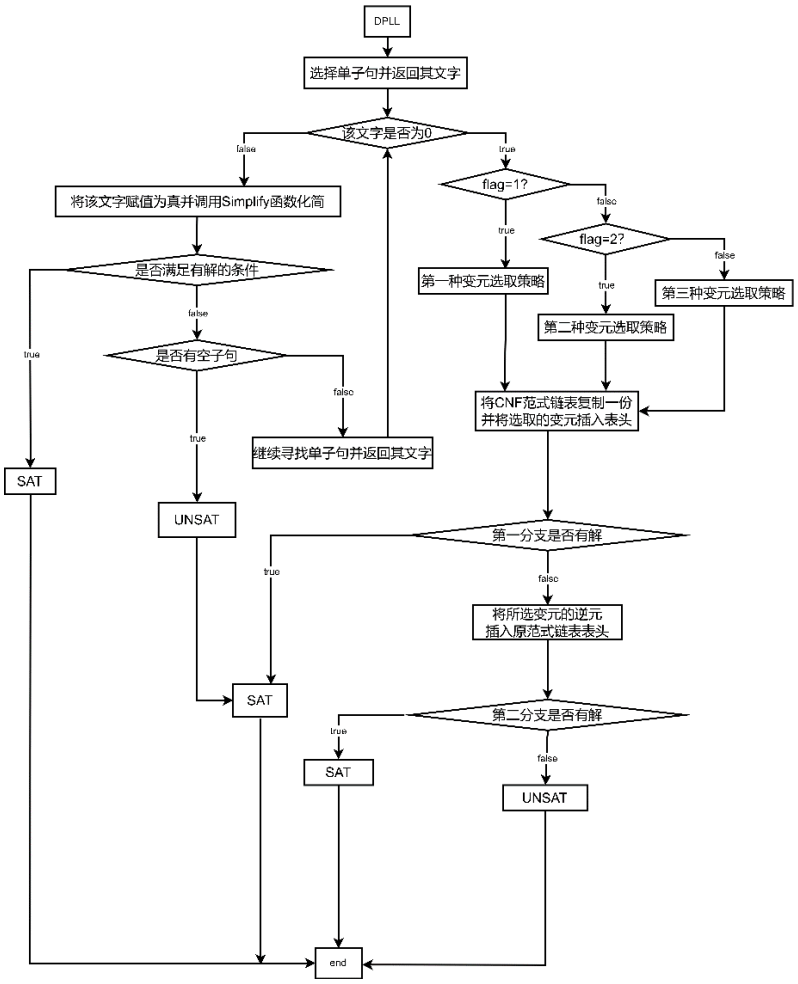

图 3.2.2-1 DPLL 函数流程图

3.2.3 对角线数独算法处理

1. 生成初始数独

status Generate_Sudoku(int board[SIZE + 1][SIZE + 1], int newBoard[SIZE + 1][SIZE + 1], int newBoard2[SIZE + 1][SIZE + 1], bool isFixed[SIZE + 1][SIZE + 1], int num, bool value[SIZE * SIZE * SIZE + 1]);

参数：(int) board[SIZE+1][SIZE+1]：保存生成的初始数独

(int) newboard[SIZE+1][SIZE+1]：用来与用户交互的数独

(int) newboard2[SIZE+1][SIZE+1]：用来保存结果的数独

(bool) isFixed[SIZE+1][SIZE+1]：用来记录谋个位置是否为提示数

(int) num: 用户输入的提示数个数

(bool) value[SIZE * SIZE * SIZE + 1]：存储数独变元真值

返回值：OK：生成成功

ERROR：生成失败

先随机生成一个初始的完整数独，再采用挖洞法生成一个有指定数目提示数的数独，并将结果保存在三个棋盘中。

2. 判断填入的数字是否有效

status Is_Valid(int board[SIZE + 1][SIZE + 1], int row, int col, int v);

参数：(int) board[SIZE + 1][SIZE + 1]：初始数独

(int) row：行号

(int) col：列号

(int) v：填入的值

返回值：OK：有效

ERROR：无效

先检查行号、列号和填入的数字是否在 1-9 之间，若不是则返回 ERROR；

再检查 v 是否满足行约束、列约束、九宫格约束及对角线约束，满足则返回 OK，否则返回 ERROR。

3. 用户玩数独的交互界面

void Play_Sudoku(int newboard[SIZE + 1][SIZE + 1], bool isFixed[SIZE + 1][SIZE + 1]);

参数：(int) board[SIZE + 1][SIZE + 1]：用来交互的数独

(bool) isFixed[SIZE + 1][SIZE + 1]：是否为提示数

返回值：void

进入该函数首先打印初始数独，再提示用户输入行号、列号和要填入的数字，先判断该位置是否为提示数，若是则对用户做出警告并让其重新输入；再判断填入的数字是否有效，无效则提示“Wrong”，否则将该数字填入该位置并打印新的数独棋盘。

4. 将数独规约为 SAT 问题并写入文件

status WriteToFile(int board[SIZE + 1][SIZE + 1], int num, char name[]);

参数：(int) board[SIZE + 1][SIZE + 1]：初始数独

(int) num：提示数个数

(char) name[]：要写入的文件名

返回值：OK：写入成功

ERROR：写入失败

先将布尔变元数目(729)、子句数目(12654+num)写入文件，再依次将提示数约束(num 行)、格子约束(2997 行)、行约束(2997 行)、列约束(2997 行)、九宫格约束(2997 行)、对角线约束(666 行)写入文件，每行以 0 结尾；注意：数字 ijk 表示第 i 行第 j 列填入 k，数字-ijk 表示第 i 行第 j 列不填入 k，再写入文件时要运用公式(i-1)*81+(j-1)*9+k 转换成以 1 起始的连续数字。

5. 求解数独

status Slove(int newboard2[SIZE + 1][SIZE + 1], bool value[SIZE * SIZE * SIZE + 1]);

参数：(int) newboard2[SIZE + 1][SIZE + 1]：保存结果的数独

(bool) value[SIZE * SIZE * SIZE + 1]：存储数独变元真值

返回值：OK：求解成功

ERROR：求解失败

调用 DPLL 函数求解规约的 SAT 问题，再根据公式(i-1)*81+(j-1)*9+k 将数独每个位置的值算出来并填入 newboard2 中。

4 系统实现与测试

4.1 系统实现

4.1.1 软硬件环境

1. 硬件环境：

处理器：12th Gen Intel Core i7-12700H

机带 RAM：16GB

2. 软件环境

Windows11—Visual Studio Code

4.1.2 数据类型定义

1.宏定义

```
#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define SIZE 9
```

2. 状态码

```
typedef int status;
```

3. 存储结构

```
/*文字节点&链表*/
typedef struct literalNode{
   int literal;   // 文字(变元)
   struct literalNode *next;   // 指向下一个文字
} literalNode, *literalList;
/*子句节点&链表*/
typedef struct clauseNode{
   literalList head;  // 指向子句中的第一个文字
   struct clauseNode *next;   // 指向下一个子句
} clauseNode, *clauseList;
/*CNF 文件*/
typedef struct cnfNode{
```

```
    clauseList root;   // 指向 CNF 的第一个子句
    int boolCount;     // 布尔变元个数
    int clauseCount;   // 子句个数
} cnfNode, *CNF;
```

4.1.3 函数声明及函数调用关系

```
/* 主交互界面 */
void DisPlay();
/* 打印菜单 */
void PrintMenu();
/* 读取文件并解析 cnf */
status ReadFile(CNF &cnf, char fileName[]);
/* 销毁当前解析的 cnf */
status DestroyCnf(clauseList &cL);
/* 打印 cnf */
status PrintCnf(CNF cnf);
/* DPLL 算法 */
status DPLL(CNF cnf, bool value[], int flag);
/* 判断是否为单子句 */
status IsUnitClause(literalList l);
/* 找到单子句并返回该文字 */
int FindUnitClause(clauseList cL);
/* 销毁子句 */
status DestroyClause(clauseList &cL);
/* 选择一个未赋值的文字(未优化) */
int ChooseLiteral_1(CNF cnf);
/* 选择一个未赋值的文字(优化) */
int ChooseLiteral_2(CNF cnf);
/* 改进 2 */
int ChooseLiteral_3(CNF cnf);
/* 根据选择的文字化简 */
void Simplify(clauseList &cL, int literal);
```

```
/* 判断文字是否满足 */
status Satisfy(clauseList cL);
/* 判断是否有空子句 */
status EmptyClause(clauseList cL);
/* 复制 cnf */
clauseList CopyCnf(clauseList cL);
/* 保存求解结果 */
status SaveResult(int result, double time, double time_, bool value[], char fileName[], int boolCount);
/* X 数独 */
void X_Sudoku();
/* 打印 X 数独菜单 */
void PrintMenu_X();
/* 生成数独 */
status Generate_Sudoku(int board[SIZE + 1][SIZE + 1], int newBoard[SIZE + 1][SIZE + 1], int newBoard2[SIZE + 1][SIZE + 1], bool isFixed[SIZE + 1][SIZE + 1], int num, bool value[SIZE * SIZE * SIZE + 1]);
/* 判断 board[row][col]是否可以填入 v */
status Is_Valid(int board[SIZE+1][SIZE+1], int row, int col, int v);
/* 打印数独 */
void Print_Sudoku(int board[SIZE+1][SIZE+1]);
/* 玩数独 */
void Play_Sudoku(int board[SIZE+1][SIZE+1], bool isFixed[SIZE+1][SIZE+1]);
/* 将数独约束条件写入文件 */
status WriteToFile(int board[SIZE+1][SIZE+1], int num, char name[]);
/* 求解数独 */
status Slove(int board[SIZE+1][SIZE+1], bool value[SIZE*SIZE*SIZE+1]);
/* 填充 3x3 的宫格 */
status Fill_Box(int board[SIZE+1][SIZE+1], int newBoard[SIZE+1][SIZE+1], int newBoard2[SIZE+1][SIZE+1], int rowStart, int colStart);
/* 打乱数组 */
```

void Shuffle(int arr[], int n);

4.2 系统测试

4.2.1 交互系统展示

主交互系统界面(SAT)和对角线数独交互系统界面分别如图 4.2.1-1 和图 4.2.1-2 所示：

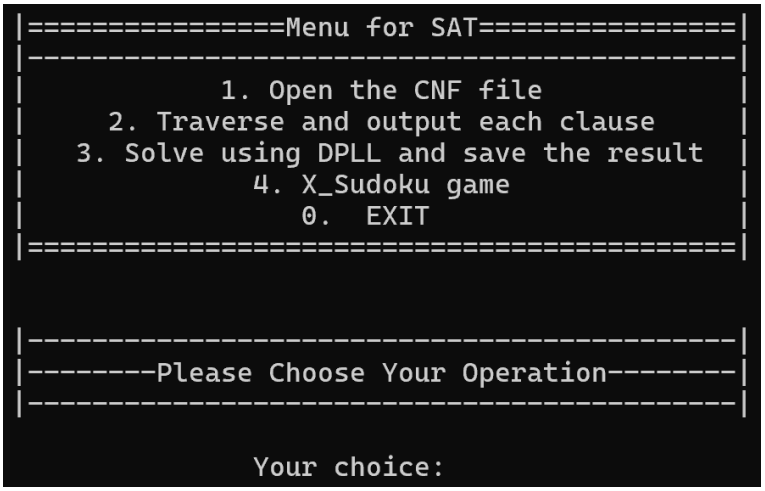

图 4.2.1-1 主交互系统(SAT)界面

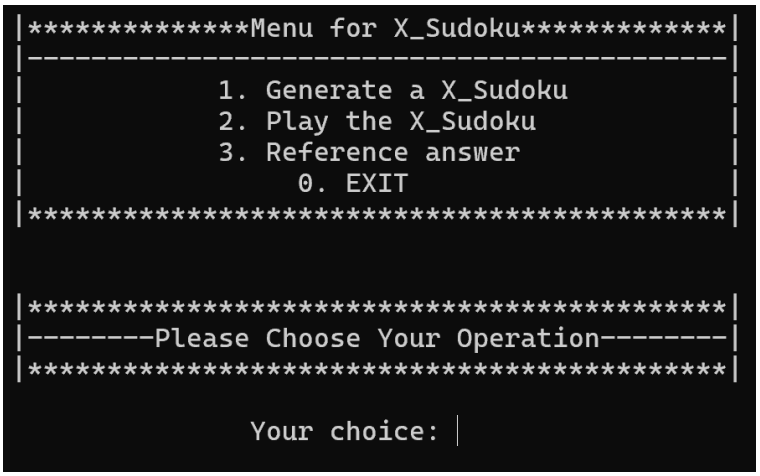

图 4.2.1-2 对角线数独交互系统界面

4.2.2 CNF 文件处理及求解模块测试

1. 读入 cnf 文件并遍历输出(测试算例:SAT 测试备选算例\满足算例 S\problem1-20.cnf)；在主交互界面输入 1，再输入文件名进行读取；再输入 2 进行遍历检查 cnf 文件解析是否正确;输入 3 求解该算例并选择是否优化和是否保存。

```
|================Menu for SAT================|
|--------------------------------------------|
|            1. Open the CNF file            |
|     2. Traverse and output each clause     |
|   3. Solve using DPLL and save the result  |
|              4. X_Sudoku game              |
|                 0.  EXIT                   |
|============================================|

Please input the file name: problem1-20.cnf
Read successfully.

|--------------------------------------------|
|--------Please Choose Your Operation--------|
|--------------------------------------------|

              Your choice:
```

图 4.2.2-1 文件读取与解析

```
The CNF is:
boolCount:20
clauseCount:91
4    -18  19   0
3    18   -5   0
-5   -8   -15  0
-20  7    -16  0
10   -13  -7   0
-12  -9   17   0
17   19   5    0
-16  9    15   0
11   -5   -14  0
18   -10  13   0
-3   11   12   0
-6   -17  -8   0
-18  14   1    0
-19  -15  10   0
12   18   -19  0
-8   4    7    0
-8   -9   4    0
7    17   -15  0
12   -7   -14  0
-10  -11  8    0
2    -15  -11  0
9    6    1    0
-11  20   -17  0
9    -15  13   0
12   -7   -17  0
-18  -2   20   0
20   12   4    0
19   11   14   0
-16  18   -4   0
-1   -17  -19  0
-13  15   10   0
-12  -14  -13  0
12   -14  -7   0
```

```
12   -14  -7   0
-7   16   10   0
6    10   7    0
20   14   -16  0
-19  17   11   0
-7   1    -20  0
-5   12   15   0
-4   -9   -13  0
12   -11  -7   0
-5   19   -8   0
1    16   17   0
20   -14  -15  0
13   -4   10   0
14   7    10   0
-5   9    20   0
10   1    -19  0
-16  -15  -1   0
16   3    -11  0
-15  -10  4    0
4    -15  -3   0
-10  -16  11   0
-8   12   -5   0
14   -6   12   0
1    6    11   0
-13  -5   -1   0
-7   -2   12   0
1    -20  19   0
-2   -13  -8   0
15   18   4    0
-11  14   9    0
-6   -15  -2   0
5    -12  -15  0
-6   17   5    0
-13  5    -19  0
20   -1   14   0
9    -17  15   0
```

图 4.2.2-2&4.2.2-3 cnf 文件遍历

```
SAT

1   : FALSE
2   : TRUE
3   : TRUE
4   : TRUE
5   : FALSE
6   : FALSE
7   : FALSE
8   : TRUE
9   : TRUE
10  : TRUE
11  : TRUE
12  : FALSE
13  : FALSE
14  : TRUE
15  : TRUE
16  : FALSE
17  : TRUE
18  : TRUE
19  : TRUE
20  : TRUE

Time: 0.310000 ms(not optimized)

Do you want to optimize the algorithm? (1/0): 1

Time: 0.254700 ms(optimized)
```

图 4.2.2-4 DPLL 求解结果及优化前后耗时

src > output > problem1-20.res

```
1  s 1
2  v -1 2 3 4 -5 -6 -7 8 9 10 11 -12 -13 14 15 -16 17 18 19 20
3  t 0.310000ms
4  t 0.254700ms(optimized)
5  Optimization Rate: 17.84%
```

图 4.2.2-5 将结果保存至同名.res 文件

2. 测试算例:SAT 测试备选算例\满足算例\S\ sud00001.cnf

```
|================Menu for SAT================|
|--------------------------------------------|
|            1. Open the CNF file            |
|     2. Traverse and output each clause     |
|   3. Solve using DPLL and save the result  |
|              4. X_Sudoku game              |
|                 0.  EXIT                   |
|============================================|

 Please input the file name: sud00001.cnf
 Read successfully.

|--------------------------------------------|
|--------Please Choose Your Operation--------|
|--------------------------------------------|

               Your choice:
```

图 4.2.2-6 文件读取与解析

```
279 : FALSE
280 : FALSE
281 : TRUE
282 : FALSE
283 : TRUE
284 : FALSE
285 : FALSE
286 : FALSE
287 : TRUE
288 : FALSE
289 : FALSE
290 : FALSE
291 : FALSE
292 : FALSE
293 : FALSE
294 : FALSE
295 : FALSE
296 : TRUE
297 : FALSE
298 : TRUE
299 : FALSE
300 : FALSE
301 : FALSE

Time: 15.407800 ms(not optimized)

Do you want to optimize the algorithm? (1/0): 1

Time: 11.964200 ms(optimized)
```

图 4.2.2-7 DPLL 求解结果及优化前后耗时

```
src > output > sud00001.res
1  s 1
2  v 1 -2 -3 -4 -5 -6 -7 8 -9 -10 11 -12 -13 -14 15 -16 -17 -18 -19 20 -21 22 -23 -24 -25 26 -27
3  t 15.407800ms
4  t 11.964200ms(optimized)
5  Optimization Rate: 22.35%
```

图 4.2.2-8 将结果保存至同名.res 文件

3. 测试算例:SAT 测试备选算例\基础算例\功能测试\unsat-5cnf-30.cnf

```
|================Menu for SAT================|
|--------------------------------------------|
|            1. Open the CNF file            |
|     2. Traverse and output each clause     |
|   3. Solve using DPLL and save the result  |
|              4. X_Sudoku game              |
|                 0.  EXIT                   |
|============================================|

Please input the file name: unsat-5cnf-30.cnf
Read successfully.

|--------------------------------------------|
|--------Please Choose Your Operation--------|
|--------------------------------------------|

               Your choice:
```

图 4.2.2-9 文件读取与解析

```
|================Menu for SAT================|
|--------------------------------------------|
|            1. Open the CNF file            |
|     2. Traverse and output each clause     |
|   3. Solve using DPLL and save the result  |
|              4. X_Sudoku game              |
|                 0.  EXIT                   |
|============================================|

UNSAT

Time: 82.846900 ms(not optimized)

Do you want to optimize the algorithm? (1/0): 1

Time: 36.472500 ms(optimized)

Save the result to file? (1/0):
```

图 4.2.2-10 DPLL 求解结果及优化前后耗时

```
src > output > unsat-5cnf-30.res
1   s 0
2   t 82.846900ms
3   t 36.472500ms(optimized)
4   Optimization Rate: 55.98%
```

图 4.2.2-11 将结果保存至同名.res 文件

4.2.3 算例测试总结

表 4-1 算例测试总结表

算例名	变元数	子句数	子句数/变元数	结果	求解时间	优化率
SAT 测试备选算例\满足算例 S\ problem1-20.cnf	20	91	0.22	满足	优化后：0.257ms 优化前：0.310ms	17.8%
SAT 测试备选算例\其它可供选择的算例\tst\tst_v10_c100.cnf	10	100	0.10	不满足	优化后：0.192ms 优化前：0.259ms	25.9%
SAT 测试备选算例\其它可供选择使用的算例\ais\ais6.cnf	61	581	0.10	满足	优化后：1.63ms 优化前：1.35ms	-17.2%

SAT 测试备选算例\其它可供选择的算例\tst\flat30-99.cnf	90	300	0.30	满足	优化后：0.86ms 优化前：1.37ms	37.2%
SAT 测试备选算例\其它可供选择使用的算例\tst\tst_v100_c425.cnf	100	425	0.24	满足	优化后：2.71ms 优化前：2.39ms	-11.8%
SAT 测试备选算例\其它可供选择使用的算例\ais\ais8.cnf	113	1520	0.07	满足	优化后：12.56ms 优化前：36.58ms	65.6%
SAT 测试备选算例\基准算例\性能测试\ais10.cnf	181	3151	0.06	满足	优化后：98.07ms 优化前：1005.9 ms	90.3%
SAT 测试备选算例\其它可供选择使用的算例\tst\tst_v200_c220.cnf	200	200	1.00	满足	优化后：5.53ms 优化前：2.81ms	-49.2%
SAT 测试备选算例\满足算例\M\sud0009.cnf	303	2851	0.106	满足	优化后：5.65ms 优化前：3.49ms	-38.2%
SAT 测试备选算例\其他可供选择的算例\ais\ais12.cnf	265	5666	0.05	满足	优化后：1801ms 优化前：34484ms	94.8%
SAT 测试备选算例\其他可供选择的算例\tst\sw100-1.cnf	500	3100	0.16	满足	优化后：59.27ms 优化前：87.57ms	32.3%
SAT 测试备选算例\其它可供选择使用的算例\tst\sw100-70.cnf	500	3100	0.16	满足	优化后：6706 ms 优化前：11728ms	-42.8%
SAT 测试备选算例\其它可供选择使用的算例\tst\qg7-09.cnf	729	22060	0.03	不满足	优化后：134.84ms 优化前：219.90ms	38.7%
SAT 测试备选算例\不满足算例\u-5cnf_3500_3500_30f1.shuffled-30.cnf	30	420	0.07	不满足	优化后：38.36ms 优化前：137.45ms	72.1%
SAT 测试备选算例\其它可供选择使用的算例\tst\qg4-08.cnf	512	9685	0.05	不满足	优化后：1349ms 优化前：1831ms	26.3%

4.2.4 数独交互界面及功能模块测试

```
|**************Menu for X_Sudoku*************|
|-----------------------------------------|
|            1. Generate a X_Sudoku          |
|            2. Play the X_Sudoku            |
|            3. Reference answer             |
|                 0. EXIT                    |
|*****************************************|

 Please enter the number of prompts(>=18): 22
 Generate successfully.

|*****************************************|
|--------Please Choose Your Operation--------|
|*****************************************|

              Your choice:
```

图 4.2.4-1 生成一个有 22 个提示数的数独

```
 . . . | . . 4 | . . 5
 9 4 . | . . . | 6 . .
 . . 2 | . 1 . | . 8 .
-------+-------+-------
 . . . | . . 6 | . 3 9
 . . 3 | 8 . . | . . 4
 . . . | 3 . . | . . .
-------+-------+-------
 4 . . | 5 . 1 | . . .
 . . 6 | . . 8 | . . .
 5 . . | . . . | . 4 .
```

图 4.2.4-2 打印该初始数独

```
. . . | . . 4 | . . 5
9 4 . | . . . | 6 . .
. . 2 | . 1 . | . 8 .
------+-------+------
. . . | . . 6 | . 3 9
. . 3 | 8 . . | . . 4
. . . | 3 . . | . . .
------+-------+------
4 . . | 5 . 1 | . . .
. . 6 | . . 8 | . . .
5 . . | . . . | . 4 .

Please enter the row, col and value(0 to EXIT): 2 1 2
This is a fixed number.
Please enter the row, col and value(0 to EXIT): 1 1 9
Wrong answer.
Please enter the row, col and value(0 to EXIT): 1 1 8
```

图 4.2.4-3 无效的输入

```
8 . . | . . 4 | . . 5
9 4 . | . . . | 6 . .
. . 2 | . 1 . | . 8 .
------+-------+------
. . . | . . 6 | . 3 9
. . 3 | 8 . . | . . 4
. . . | 3 . . | . . .
------+-------+------
4 . . | 5 . 1 | . . .
. . 6 | . . 8 | . . .
5 . . | . . . | . 4 .

Please enter the row, col and value(0 to EXIT):
```

图 4.2.4-4 输入有效,打印填入后的新数独

```
Original X_Sudoku:
. . . | . . 4 | . . 5
9 4 . | . . . | 6 . .
. . 2 | . 1 . | . 8 .
------+-------+------
. . . | . . 6 | . 3 9
. . 3 | 8 . . | . . 4
. . . | 3 . . | . . .
------+-------+------
4 . . | 5 . 1 | . . .
. . 6 | . . 8 | . . .
5 . . | . . . | . 4 .

Reference answer:
8 6 7 | 2 3 4 | 1 9 5
9 4 1 | 7 8 5 | 6 2 3
3 5 2 | 6 1 9 | 4 8 7
------+-------+------
2 8 4 | 1 5 6 | 7 3 9
1 7 3 | 8 9 2 | 5 6 4
6 9 5 | 3 4 7 | 8 1 2
------+-------+------
4 2 9 | 5 6 1 | 3 7 8
7 3 6 | 4 2 8 | 9 5 1
5 1 8 | 9 7 3 | 2 4 6

|*****************************************|
|--------Please Choose Your Operation--------|
|*****************************************|

              Your choice:
```

图 4.2.4-5 查看参考答案

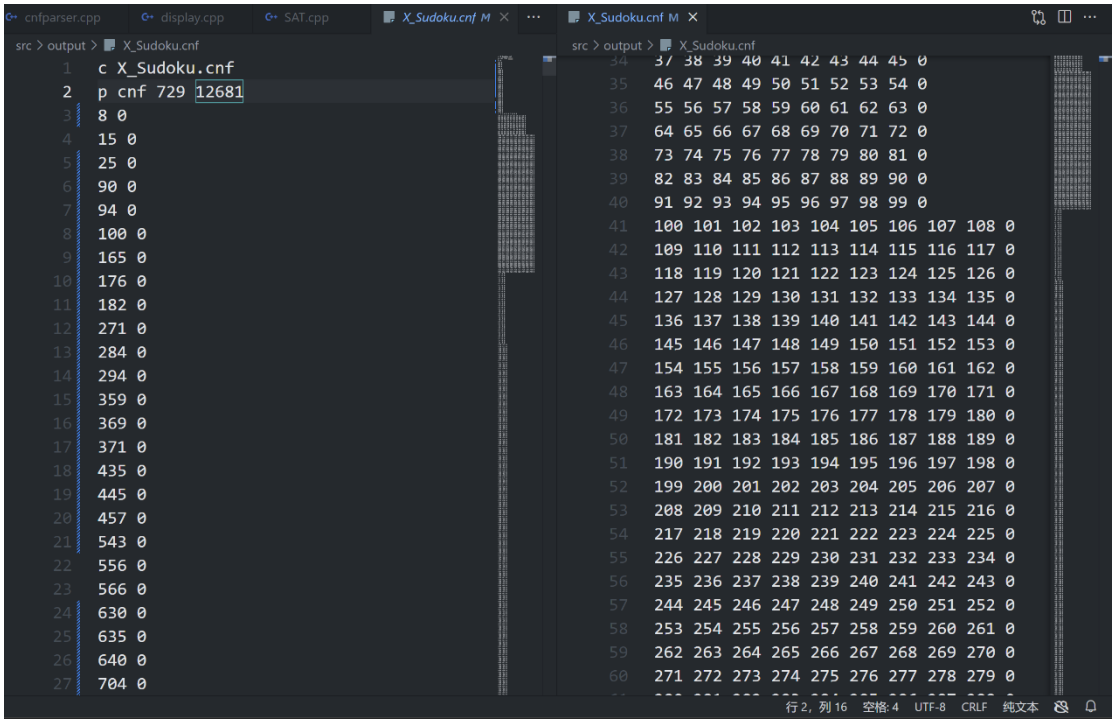

图 4.2.4-6 将数独归约后的 cnf 文件

5 总结与展望

5.1 全文总结

主要工作如下：

(1) 实现了一个基于 DPLL 算法的 SAT 求解器；

(2) 对 SAT 求解器的变元选择策略进行了优化改进；

(3) 完成了数独游戏的初始化工作，包括 CNF 文件的转换、基于挖洞法的随机数独生成、数独终盘的绘制等，并实现了一定程度的交互功能；

(4) 设计了测试方案并成功完成了多个算例的测试；

(5) 对各部分功能进行了衔接和完善，构建了一个简易系统。

5.2 工作展望

在今后的研究中，围绕着如下几个方面开展工作：

(1) **进一步优化 SAT 求解器的性能：** 虽然目前的 SAT 求解器在变元选择策略上做了一定的优化，但未来还可以考虑引入更高级的启发式算法，例如 VSIDS（Variable State Independent Decaying Sum）或 LRB（Learning Rate Branching）等现代 SAT 求解算法中常用的策略。通过结合多种启发式方法，进一步提升求解效率，并在不同规模和复杂度的算例中测试其表现；

(2) **实现更多高级数独生成算法：** 当前的数独生成采用了挖洞法，后续可以尝试基于复杂度可调节的生成算法，生成更具挑战性和多样性的数独题目。还可以加入难度分析模型，根据玩家的解题历史动态调整题目的难度级别，增加游戏的趣味性和互动性；

(3) **增强用户交互和游戏体验：** 数独游戏目前具备了一定的交互性，未来可以进一步增强用户界面设计和互动功能。可以实现更加友好的用户提示系统、解题步骤回放以及错误反馈机制。同时，添加计时器、排行榜等游戏化元素，提升玩家的沉浸感和参与度；

(4) **探索更多的应用场景：**SAT 求解器不仅可以用于数独问题，还可以用于许多现实世界中的复杂问题求解。未来可以探索其在硬件验证、逻辑电路设计、

AI 规划以及其他约束满足问题中的应用。通过实际应用案例，扩展求解器的适用范围，并针对不同领域进行专项优化；

(5) **构建更加健壮的测试与验证体系：** 虽然现阶段已经对求解器进行了多个算例的测试，但今后可以进一步完善测试体系，构建大规模自动化测试框架，涵盖不同类型、规模和复杂度的 SAT 问题。确保在多种条件下的稳定性与效率，并针对性能瓶颈和异常情况进行深入分析与改进；

(6) **开源与社区合作：** 未来可以考虑将 SAT 求解器及数独生成工具开源，发布到 GitHub 或其他开发者社区，与全球开发者共享经验和成果。通过社区合作，可以收集更多用户反馈，找到潜在的改进方向，并通过开源项目获得更多的贡献者参与，共同推动项目的发展。

6 体会

6.1 算法设计与优化的思考

在这次项目中，我对算法设计与优化有了深刻的理解。DPLL 算法的核心思想虽然相对简单，但在实际实现中，如何进行优化和设计高效的搜索策略对结果的影响巨大。这让我不仅仅停留在算法的表面实现上，而是开始深入思考算法的运行机制以及性能瓶颈。

1. 算法效率与性能权衡

在实现 DPLL 算法的过程中，选择合适的文字和处理单子句成为关键步骤。起初，我选择了一个较为简单的文字选择策略，但随着测试用例的复杂度增加，我发现这种方式在处理大规模问题时效率并不高。通过不断实验和对比不同的策略，我了解到优化不仅仅是让算法变得“更快”，还需要在性能与复杂性之间找到平衡。过于复杂的策略虽然在某些场景下提升了效率，但在一些规模较小的例子中却反而拖慢了整体速度。

2. 递归与回溯的挑战

DPLL 算法本质上是一个递归求解问题，在递归调用中，我频繁遇到性能瓶颈的挑战。由于未对递归深度进行控制，算法在复杂求解中容易陷入过深的递归。在后续的改进中，我计划引入递归深度控制，并设计更合理的剪枝策略，以避免无效的分支探索，从而减少搜索空间。这一过程让我对递归算法有了更深入的理解，也认识到在复杂问题中引入合理的优化策略是必不可少的。

6.2 调试与测试的重要性

在这次开发过程中，调试和测试的重要性让我有了深刻的体会。面对复杂的递归调用和多变的逻辑分支，小错误往往会隐藏在代码深处，导致程序意外的行为。在不断调试与完善的过程中，我深刻感受到测试对于保证代码质量的价值。

1. 单元测试与模块化调试的启示

开发初期，我意识到 SAT 求解器的每个部分都具有较强的独立性，像文字选择、子句处理、递归求解等模块相互依赖但逻辑上相对独立。因此，我首先为每个模块编写了单元测试。这样的做法不仅让我能精准定位每个模块中的问题，还让我能在局部修复错误后迅速验证效果，避免了复杂系统中常见的“牵一发而动全身”的情况。这种模块化调试的方式让我更加确信，分而治之的思想在大型

项目中是极为有效的。

2. 测试用例设计的深层思考

设计测试用例不仅仅是验证程序的正确性，更是一种提前预测算法性能的手段。在设计 SAT 求解器的测试时，我选择了从简单的小规模问题逐渐增加复杂度，尤其是将数独问题作为复杂的测试用例。

3. 代码审查与优化的心得

在调试的过程中，我意识到，不仅要解决功能上的错误，代码结构的合理性和效率同样重要。频繁的代码审查让我能够发现不必要的重复计算以及内存资源的浪费，这不仅影响了程序的运行速度，也降低了代码的可维护性。因此，在修复问题的同时，我还主动去优化代码的逻辑和结构。在这个过程中，我感受到一种成就感：不仅让程序运行得更快，也让代码变得更加清晰易懂。

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

Twodoku： https://en.grandgames.net/multisudoku/twodoku

[6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic for Programming, 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 International Symposium 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] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

附录

```
#pragma once

/*头文件*/
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
#include <windows.h>
#include <winnt.h>
/*常量*/
#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define SIZE 9
/*状态码*/
typedef int status;
/*文字节点&链表*/
typedef struct literalNode{
    int literal;               // 文字(变元)
    struct literalNode *next; // 指向下一个文字
} literalNode, *literalList;
/*子句节点&链表*/
typedef struct clauseNode{
    literalList head;         // 指向子句中的第一个文字
    struct clauseNode *next; // 指向下一个子句
} clauseNode, *clauseList;
/*CNF 文件*/
typedef struct cnfNode{
    clauseList root; // 指向 CNF 的第一个子句
    int boolCount;   // 布尔变元个数
    int clauseCount; // 子句个数
} cnfNode, *CNF;
```

```
/*函数声明*/
void DisPlay(); // 主交互界面
void PrintMenu(); // 打印菜单
status ReadFile(CNF &cnf,char fileName[]); // 读取文件并解析 cnf
status DestroyCnf(clauseList &cL); // 销毁当前解析的 cnf
status PrintCnf(CNF cnf); // 打印 cnf
status DPLL(CNF cnf,bool value[],int flag); // DPLL 算法
status IsUnitClause(literalList l); // 判断是否为单子句
int FindUnitClause(clauseList cL); // 找到单子句并返回该文字
status DestroyClause(clauseList &cL); // 销毁子句
int ChooseLiteral_1(CNF cnf); // 选择一个未赋值的文字(未优化)
int ChooseLiteral_2(CNF cnf); // 选择一个未赋值的文字(优化)
int ChooseLiteral_3(CNF cnf); // 改进 2
void Simplify(clauseList &cL, int literal); // 根据选择的文字化简
status Satisfy(clauseList cL); // 判断文字是否满足
status EmptyClause(clauseList cL); // 判断是否有空子句
clauseList CopyCnf(clauseList cL); // 复制 cnf
status SaveResult(int result, double time, double time_, bool value[],char
fileName[],int boolCount); // 保存求解结果
void X_Sudoku(); // X 数独
void PrintMenu_X(); // 打印 X 数独菜单
status Generate_Sudoku(int board[SIZE+1][SIZE+1],int
newBoard[SIZE+1][SIZE+1],int newBoard2[SIZE+1][SIZE+1],bool
isFixed[SIZE+1][SIZE+1],int num,bool value[SIZE*SIZE*SIZE+1]); // 生成数
独
status Is_Valid(int board[SIZE+1][SIZE+1],int row, int col, int v); // 判
断 board[row][col]是否可以填入 v
void Print_Sudoku(int board[SIZE+1][SIZE+1]); // 打印数独
void Play_Sudoku(int board[SIZE+1][SIZE+1],bool isFixed[SIZE+1][SIZE+1]);
// 玩数独
status WriteToFile(int board[SIZE+1][SIZE+1],int num,char name[]); // 将
数独约束条件写入文件
status Slove(int board[SIZE+1][SIZE+1],bool value[SIZE*SIZE*SIZE+1]); //
求解数独
status Fill_Box(int board[SIZE+1][SIZE+1], int
newBoard[SIZE+1][SIZE+1],int newBoard2[SIZE+1][SIZE+1],int rowStart, int
colStart); // 填充 3x3 的宫格
void Shuffle(int arr[], int n); // 打乱数组
```

```
/*---------------------------cnfparser---------------------------*/
/*
 @ 函数名称: ReadFile
 @ 接受参数: clauseList &,char[]
 @ 函数功能: 用文件指针 fp 打开用户指定的文件，并读取文件内容保存到给定参数中
 @ 返回值: status
 */
status ReadFile(CNF &cnf, char fileName[]){
    FILE *fp = fopen(fileName, "r");
    while (fp == NULL){
        printf(" File not found, please input again: ");
        scanf("%s", fileName);
        fp = fopen(fileName, "r");
    }
    char ch;
    // 跳过注释
    while ((ch = getc(fp)) == 'c'){
        while ((ch = getc(fp)) != '\n')
            continue;
    }
    // 跳过 p cnf
    getc(fp);
    getc(fp);
    getc(fp);
    getc(fp);
     fscanf(fp, "%d%d", &cnf->boolCount, &cnf->clauseCount);
     // 读取布尔变元个数和子句个数
    cnf->root = NULL; // 初始化 CNF
    clauseList lastClause = NULL; // 用于记录上一个一个子句
    for (int i = 0; i < cnf->clauseCount; i++){
        // 读取子句
        clauseList newClause = (clauseList)malloc(sizeof(clauseNode));
        newClause->head = NULL;
        newClause->next = NULL;
        literalList lastLiteral = NULL; // 用于记录上一个文字
        int number; // 读取文字
        fscanf(fp, "%d", &number);
        while (number != 0){
```

```
            literalList newLiteral =
(literalList)malloc(sizeof(literalNode));
            newLiteral->literal = number;
            newLiteral->next = NULL;
            if (newClause->head == NULL) // 如果是第一个文字
                newClause->head = newLiteral;
            else
                lastLiteral->next = newLiteral;
            lastLiteral = newLiteral; // 更新上一个文字
            fscanf(fp, "%d", &number);
        }
        if (cnf->root == NULL) // 如果是第一个子句
            cnf->root = newClause;
        else
            lastClause->next = newClause;
        lastClause = newClause; // 更新上一个子句
    }
    fclose(fp);
    return OK;
}
/*
 @ 函数名称: DestroyCnf
 @ 接受参数: clauseList &
 @ 函数功能: 销毁给定的 CNF 文件
 @ 返回值: status
 */
status DestroyCnf(clauseList &cL){
    while (cL != NULL){
        clauseList tempClause = cL;
        cL = cL->next; // 指向下一个子句
        literalList p = tempClause->head; // 指向当前子句的第一个文字
        while (p != NULL){
            literalList tempLiteral = p;
            p = p->next;
            free(tempLiteral); // 释放文字
        }
        free(tempClause); // 释放子句
    }
```

```
    cL = NULL; // cL 指向 NULL
    return OK;
}
/*
 @ 函数名称: PrintCnf
 @ 接受参数: clauseList
 @ 函数功能: 打印给定的 CNF 文件
 @ 返回值: status
 */
status PrintCnf(CNF cnf){
    clauseList p = cnf->root;
    // 如果没有子句
    if (p == NULL){
        printf(" No clauses.\n");
        return ERROR;
    }
    printf("  The CNF is:\n");
    printf("  boolCount:%d\n", cnf->boolCount);     // 打印布尔变元个数
    printf("  clauseCount:%d\n", cnf->clauseCount); // 打印子句个数
    while (p){
        literalList q = p->head; // 指向子句的第一个文字
        printf("  ");
        while (q){
            printf("%-5d", q->literal); // 打印文字
            q = q->next;
        }
        printf("0\n"); // 子句结束
        p = p->next;   // 指向下一个子句
    }
    return OK;
}
/*----------------------------display-----------------------------*/
/*
 @ 函数名称: DisPlay
 @ 函数功能: 交互界面
 @ 返回值: void
 */
void DisPlay(){
```

```
    bool *value = NULL; // 存储文字的真值
    // clauseList cL = NULL;
    CNF cnf = (CNF)malloc(sizeof(cnfNode));
    cnf->root = NULL;
    char fileName[100]; // 文件名
    PrintMenu();        // 打印菜单
    int op = 1;
    while (op){
        printf("\n|-----------------------------------------|\n");
        printf(  "|--------Please Choose Your Operation--------|\n");
        printf(  "|-----------------------------------------|\n\n");
        printf(  "               Your choice: ");
        scanf("%d", &op);
        system("cls"); // 每次进来清屏
        PrintMenu();   // 打印菜单
        switch (op){
        case 1:{
            if (cnf->root != NULL){ // 如果已经打开了 CNF 文件
                printf(" The CNF has been read.\n");
                printf(" Do you want to read another? (1/0): ");
                int choice;
                scanf("%d", &choice);
                if (choice == 0)
                    break;
                else // 重新读取
                    DestroyCnf(cnf->root); // 销毁当前解析的 CNF
            }
            printf(" Please input the file name: ");
            scanf("%s", fileName);
            if (ReadFile(cnf, fileName) == OK) // 读取文件并解析 CNF
                printf(" Read successfully.\n");
            else
                printf(" Read failed.\n");
            break;
        }
        case 2:{
            if (cnf->root == NULL) // 如果没有打开 CNF 文件
                printf(" You haven't open the CNF file.\n");
```

```
            else
                PrintCnf(cnf); // 打印 CNF 文件
            break;
        }
        case 3:{
            if (cnf->root == NULL){ // 如果没有打开 CNF 文件
                printf(" You haven't open the CNF file.\n");
                break;
            }
            else{
                CNF newCnf = (CNF)malloc(sizeof(cnfNode));
                newCnf->root = CopyCnf(cnf->root); // 复制 CNF
                newCnf->boolCount = cnf->boolCount;
                newCnf->clauseCount = cnf->clauseCount;
                value = (bool *)malloc(sizeof(bool) * (cnf->boolCount + 1));
                for (int i = 1; i <= cnf->boolCount; i++)
                    value[i] = TRUE; // 初始化，均赋为 TRUE
                LARGE_INTEGER frequency, frequency_; // 计时器频率
                LARGE_INTEGER start, start_, end, end_; // 设置时间变量
                double time, time_;
                // 未优化的时间
                QueryPerformanceFrequency(&frequency);
                QueryPerformanceCounter(&start); // 计时开始;
                int result = DPLL(newCnf, value, 1);
                QueryPerformanceCounter(&end); // 结束
                time = (double)(end.QuadPart - start.QuadPart) /
frequency.QuadPart; // 计算运行时间
                // 输出 SAT 结果
                if (result == OK){ // SAT
                    printf(" SAT\n\n");
                    // 输出文字的真值
                    for (int i = 1; i <= cnf->boolCount; i++){
                        if (value[i] == TRUE)
                            printf(" %-4d: TRUE\n", i);
                        else
                            printf(" %-4d: FALSE\n", i);
                    }
                }
```

```
                else // UNSAT
                    printf(" UNSAT\n");
                // 输出优化前的时间
                printf("\n Time: %lf ms(not optimized)\n", time * 1000);
                // 是否优化
                int ch;
                printf("\n Do you want to optimize the algorithm? (1/0): ");
                scanf("%d", &ch);
                if (ch == 0)
                    time_ = 0;
                else{
                    DestroyCnf(newCnf->root); // 销毁未优化的 CNF
                    newCnf->root = CopyCnf(cnf->root); // 复制 CNF
                    QueryPerformanceFrequency(&frequency_);
                    QueryPerformanceCounter(&start_); // 计时开始;
                    DPLL(newCnf, value, 3);
                    QueryPerformanceCounter(&end_); // 结束
                    time_ = (double)(end_.QuadPart - start_.QuadPart) /
frequency_.QuadPart; // 计算运行时间
                    printf("\n Time: %lf ms(optimized)\n", time_ * 1000);
                }
                // 是否保存
                printf("\n Save the result to file? (1/0): ");
                int choice;
                scanf("%d", &choice);
                printf("\n");
                if (choice == 1){
                    // 保存求解结果
                    if (SaveResult(result, time, time_, value, fileName,
cnf->boolCount))
                        printf(" Save successfully.\n");
                    else
                        printf(" Save failed.\n");
                }
            }
            break;
        }
        case 4:{
```

```
            X_Sudoku();  // X 数独界面
            PrintMenu(); // 跳转回来时重新打印菜单
            break;
        }
        case 0:{
            printf(" Exit successfully.\n");
            return; // 退出
        }
        default:{
            printf(" Invalid input.\n"); // 无效输入
            break;
        }
        }
    }
    if (cnf->root != NULL)
        DestroyCnf(cnf->root); // 退出时销毁 CNF
    free(cnf);
    return;
}
/*
 @ 函数名称: PrintMenu
 @ 函数功能: 打印菜单
 @ 返回值: void
 */
void PrintMenu(){
    printf("|================Menu for SAT================|\n");
    printf("|--------------------------------------------|\n");
    printf("|            1. Open the CNF file            |\n");
    printf("|     2. Traverse and output each clause     |\n");
    printf("|   3. Solve using DPLL and save the result  |\n");
    printf("|              4. X_Sudoku game              |\n");
    printf("|                 0.  EXIT                   |\n");
    printf("|============================================|\n\n");
}
/*----------------------------solver----------------------------*/
/*
 @ 函数名称: IsUnitClause
 @ 接受参数: literalList
```

```
 @ 函数功能: 判断是否为单子句
 @ 返回值: status
 */
status IsUnitClause(literalList l){
    if (l != NULL && l->next == NULL) // 只有一个文字
        return TRUE;
    else
        return FALSE;
}
/*
 @ 函数名称: FindUnitClause
 @ 接受参数: clauseList
 @ 函数功能: 找到单子句并返回该文字
 @ 返回值: int
 */
int FindUnitClause(clauseList cL){
    clauseList p = cL;
    while (p){
        if (IsUnitClause(p->head))   // 是单子句
            return p->head->literal; // 返回该文字
        p = p->next;
    }
    return 0;
}
/*
 @ 函数名称: DestroyClause
 @ 接受参数: clauseList &
 @ 函数功能: 销毁子句
 @ 返回值: status
 */
status DestroyClause(clauseList &cL){
    literalList p = cL->head;
    while (p){
        literalList temp = p;
        p = p->next; // 指向下一个文字
        free(temp);  // 释放文字
    }
    free(cL);  // 释放子句
```

```
    cL = NULL; // cL 指向 NULL
    return OK;
}
/*
 @ 函数名称: Simplify
 @ 接受参数: clauseList &, int
 @ 函数功能: 根据选择的文字化简
 @ 返回值: void
 */
void Simplify(clauseList &cL, int literal){
    clauseList pre = NULL, p = cL; // pre 指向前一个子句
    while (p != NULL){
        bool clauseDeleted = false;           // 是否删除子句
        literalList lpre = NULL, q = p->head; // lpre 指向前一个文字
        while (q != NULL){
            if (q->literal == literal){ // 删除该子句
                if (pre == NULL) // 删除的是第一个子句
                    cL = p->next;
                else // 删除的不是第一个子句
                    pre->next = p->next;
                DestroyClause(p);                    // 销毁该子句
                p = (pre == NULL) ? cL : pre->next; // 指向下一个子句
                clauseDeleted = true;                // 子句已删除
                break;
            }
            else if (q->literal == -literal){ // 删除该文字
                if (lpre == NULL) // 删除的是第一个文字
                    p->head = q->next;
                else // 删除的不是第一个文字
                    lpre->next = q->next;
                free(q);                                    // 释放该文字
                q =(lpre == NULL) ? p->head : lpre->next; // 指向下一个文字
            }
            else{ // 未删除
                lpre = q;
                q = q->next;
            }
        }
```

```
        if (!clauseDeleted){ // 子句未删除
            pre = p;
            p = p->next;
        }
    }
}
/*
 @ 函数名称: CopyCnf
 @ 接受参数: clauseList
 @ 函数功能: 复制 cnf
 @ 返回值: clauseList
 */
clauseList CopyCnf(clauseList cL){
    // 初始化新的 CNF
    clauseList newCnf = (clauseList)malloc(sizeof(clauseNode));
    clauseList lpa, lpb;  // lpa 指向新的子句,lpb 指向旧的子句
    literalList tpa, tpb; // tpa 指向新的文字,tpb 指向旧的文字
    newCnf->head = (literalList)malloc(sizeof(literalNode));
    newCnf->next = NULL;
    newCnf->head->next = NULL;
    for (lpb = cL, lpa = newCnf; lpb != NULL; lpb = lpb->next, lpa = lpa->next){
        for (tpb = lpb->head, tpa = lpa->head; tpb != NULL; tpb = tpb->next,
tpa = tpa->next){
            tpa->literal = tpb->literal;
            tpa->next = (literalList)malloc(sizeof(literalNode));
            tpa->next->next = NULL;
            if (tpb->next == NULL){ // 旧的子句中的文字已经复制完
                free(tpa->next);
                tpa->next = NULL;
            }
        }
        lpa->next = (clauseList)malloc(sizeof(clauseNode));
        lpa->next->head = (literalList)malloc(sizeof(literalNode));
        lpa->next->next = NULL;
        lpa->next->head->next = NULL;
        if (lpb->next == NULL){ // 旧的 CNF 中的子句已经复制完
            free(lpa->next->head);
            free(lpa->next);
```

```
            lpa->next = NULL;
        }
    }
    return newCnf;
}
/*
 @ 函数名称: ChooseLiteral_1(未优化)
 @ 接受参数: clauseList
 @ 函数功能: 选择文字(第一个文字)
 @ 返回值: int
 */
int ChooseLiteral_1(CNF cnf){
    return cnf->root->head->literal;
}
/*
 @ 函数名称: ChooseLiteral_2(优化)
 @ 接受参数: clauseList
 @ 函数功能: (没有单子句时的策略)选择文字(出现次数最多的文字)
 @ 返回值: int
 */
int ChooseLiteral_2(CNF cnf){
    clauseList lp = cnf->root;
    literalList dp;
    int *count, MaxWord, max; // count 记录每个文字出现次数,MaxWord 记录出现
最多次数的文字
    count = (int *)malloc(sizeof(int) * (cnf->boolCount * 2 + 1));
    for (int i = 0; i <= cnf->boolCount * 2; i++)
        count[i] = 0; // 初始化
    // 计算子句中各文字出现次数
    for (lp = cnf->root; lp != NULL; lp = lp->next){
        for (dp = lp->head; dp != NULL; dp = dp->next){
            if (dp->literal > 0) // 正文字
                count[dp->literal]++;
            else
                count[cnf->boolCount - dp->literal]++; // 负文字
        }
    }
    max = 0;
```

```
    // 找到出现次数最多的正文字
    for (int i = 1; i <= cnf->boolCount; i++){
        if (max < count[i]){
            max = count[i];
            MaxWord = i;
        }
    }
    if (max == 0){
        // 若没有出现正文字,找到出现次数最多的负文字
        for (int i = cnf->boolCount + 1; i <= cnf->boolCount * 2; i++){
            if (max < count[i]){
                max = count[i];
                MaxWord = cnf->boolCount - i;
            }
        }
    }
    free(count);
    return MaxWord;
}
/*
 @ 函数名称: ChooseLiteral_3(优化)
 @ 接受参数: clauseList
 @ 函数功能: 选择最短子句中出现次数最多的文字
 @ 返回值: int
*/
int ChooseLiteral_3(CNF cnf){
    clauseList p = cnf->root;
    int *count = (int *)calloc(cnf->boolCount * 2 + 1, sizeof(int));
    int minSize = INT_MAX; // 初始化为大于可能的最大子句长度
    int literal = 0;
    clauseList temp = NULL;
    // 遍历子句，找到最小子句并统计其文字
    while (p != NULL){
        literalList q = p->head;
        int clauseSize = 0;
        while (q != NULL){
            clauseSize++;
            q = q->next;
```

```
        }
        if (clauseSize < minSize){
            minSize = clauseSize; // 更新最小子句大小
            temp = p;
        }
        p = p->next;
    }
    // 遍历子句，统计最小子句中各文字出现次数
    literalList q = temp->head;
    while (q != NULL){
        count[q->literal + cnf->boolCount]++;
        q = q->next;
    }
    // 找到最频繁的文字
    int maxCount = 0;
    for (int i = 0; i < cnf->boolCount * 2 + 1; i++){
        if (count[i] > maxCount){
            maxCount = count[i];
            literal = i - cnf->boolCount;
        }
    }
    free(count);
    return literal;
}
/*
 @ 函数名称: Satisfy
 @ 接受参数: clauseList
 @ 函数功能: 是否满足
 @ 返回值: status
*/
status Satisfy(clauseList cL){
    if (cL == NULL)
        return OK;
    else
        return ERROR;
}
/*
 @ 函数名称: EmptyClause
```

```
 @ 接受参数: clauseList
 @ 函数功能: 是否有空子句
 @ 返回值: status
 */
status EmptyClause(clauseList cL){
    clauseList p = cL;
    while (p){
        if (p->head == NULL) // 空子句，返回 UNSAT
            return TRUE;
        p = p->next;
    }
    return FALSE;
}
/*
 @ 函数名称: DPLL
 @ 接受参数: clauseList,int[]
 @ 函数功能: DPLL 算法求解 SAT 问题
 @ 返回值: status
 */
status DPLL(CNF cnf, bool value[], int flag){
    /*1.单子句规则*/
    int unitLiteral = FindUnitClause(cnf->root);
    while (unitLiteral != 0){
        value[abs(unitLiteral)] = (unitLiteral > 0) ? TRUE : FALSE;
        Simplify(cnf->root, unitLiteral);
        // 终止条件
        if (Satisfy(cnf->root) == OK)
            return OK;
        if (EmptyClause(cnf->root) == TRUE)
            return ERROR;
        unitLiteral = FindUnitClause(cnf->root);
    }
    /*2.选择一个未赋值的文字*/
    int literal;
    if (flag == 1)
        literal = ChooseLiteral_1(cnf); // 未优化
    else if (flag == 2)
        literal = ChooseLiteral_2(cnf); // 优化
```

```
    else
        literal = ChooseLiteral_3(cnf);
    /*3.将该文字赋值为真，递归求解*/
    CNF newCnf = (CNF)malloc(sizeof(cnfNode));
    newCnf->root = CopyCnf(cnf->root); // 复制 CNF
    newCnf->boolCount = cnf->boolCount;
    newCnf->clauseCount = cnf->clauseCount;
    clauseList p = (clauseList)malloc(sizeof(clauseNode));
    p->head = (literalList)malloc(sizeof(literalNode));
    p->head->literal = literal;
    p->head->next = NULL;
    p->next = newCnf->root;
    newCnf->root = p; // 插入到表头
    if (DPLL(newCnf, value, flag) == 1)
        return 1; // 在第一分支中搜索
    DestroyCnf(newCnf->root);
    /*4.将该文字赋值为假，递归求解*/
    // newCnf = CopyCnf(cL);
    // newCnf=cL;
    clauseList q = (clauseList)malloc(sizeof(clauseNode));
    q->head = (literalList)malloc(sizeof(literalNode));
    q->head->literal = -literal;
    q->head->next = NULL;
    q->next = cnf->root;
    cnf->root = q; // 插入到表头
    status re=DPLL(cnf,value,flag); // 回溯到执行分支策略的初态进入另一分支
    // DestroyCnf(cL);
    return re;
}
/*
 @ 函数名称: SaveResult
 @ 接受参数: int,double,int[],char[]
 @ 函数功能: 保存求解结果
 @ 返回值: status
 */
status SaveResult(int result, double time, double time_, bool value[], char
fileName[], int boolCount){
    FILE *fp;
```

```
    char name[100];
    for (int i = 0; fileName[i] != '\0'; i++){
        // 修改拓展名.res
        if (fileName[i] == '.' && fileName[i + 4] == '\0'){
            name[i] = '.';
            name[i + 1] = 'r';
            name[i + 2] = 'e';
            name[i + 3] = 's';
            name[i + 4] = '\0';
            break;
        }
        name[i] = fileName[i];
    }
    if (fopen_s(&fp, name, "w")){
        printf(" Fail!\n");
        return ERROR;
    }
    fprintf(fp, "s %d", result); // 求解结果
    if (result == 1){
        fprintf(fp, "\nv");
        // 保存解值
        for (int i = 1, cnt = 1; i <= boolCount; i++, cnt++){
            if (value[i] == true)
                fprintf(fp, " %d", i);
            else
                fprintf(fp, " %d", -i);
        }
    }
    fprintf(fp, "\nt %lfms", time * 1000); // 运行时间/毫秒
    if (time_ != 0){
        fprintf(fp, "\nt %lfms(optimized)", time_ * 1000);// 运行时间/毫秒
        double optimization_rate = ((time - time_) / time) * 100;// 优化率
        fprintf(fp, "\nOptimization Rate: %.2lf%%", optimization_rate);
    }
    fclose(fp);
    return OK;
}
/*----------------------------X-Sudoku----------------------------*/
```

```
/*
 @ 函数名称: X_Sudoku
 @ 函数功能: X 数独交互界面
 @ 返回值: void
 */
void X_Sudoku(){
    system("cls");
    PrintMenu_X();
    int num;                              // 提示数的个数
    bool isFixed[SIZE + 1][SIZE + 1];  // 记录是否为提示数字
    int board[SIZE + 1][SIZE + 1];     // 生成的初始数独
    int newBoard[SIZE + 1][SIZE + 1];  // 用来玩的数独
    int newBoard2[SIZE + 1][SIZE + 1]; // 保存答案的数独
    bool value[SIZE * SIZE * SIZE + 1];// 记录 DPLL 的结果
    for (int i = 1; i <= SIZE * SIZE * SIZE; i++)
        value[i] = FALSE;
    int op = 1;   // 操作
    int flag = 0; // 是否生成数独
    while (op){
        printf("\n|*****************************************|\n");
        printf(  "|--------Please Choose Your Operation--------|\n");
        printf(  "|*****************************************|\n\n");
        printf(  "               Your choice: ");
        scanf("%d", &op);
        system("cls");
        PrintMenu_X();
        switch (op){
        case 1:{
            printf(" Please enter the number of prompts(>=18): ");
            scanf("%d", &num);
            while (num<18|| num>81){// 提示数的个数必须大于等于 18 小于等于 81
                printf(" Invalid input, please enter again: ");
                scanf("%d", &num);
            }
            if (Generate_Sudoku(board, newBoard, newBoard2, isFixed, num,
value)){
                printf(" Generate successfully.\n");
                flag = 1; // 生成成功
```

```
            }
            else
                printf(" Generate failed.\n");
            break;
        }
        case 2:{
            if (flag){
                Play_Sudoku(newBoard, isFixed);
                PrintMenu_X(); // 每次玩完跳转回来重新打印菜单
            }
            else
                printf(" Please generate the X_Sudoku first.\n");
            break;
        }
        case 3:{
            if (flag){
                printf(" Original X_Sudoku:\n");
                Print_Sudoku(board); // 打印原始数独
                printf("\n");
                if (Slove(newBoard2, value)){ // 求解数独
                    printf(" Reference answer:\n");
                    Print_Sudoku(newBoard2); // 打印答案
                }
                else
                    printf(" No answer.\n"); // 无解
            }
            else
                printf(" Please generate the X_Sudoku first.\n");
            break;
        }
        case 0:{
            system("cls"); // 退出时清屏
            break;
        }
        default:{
            printf(" Invalid input.\n");
            break;
        }
```

```
        }
    }
/*
 @ 函数名称: PrintMenu_X
 @ 函数功能: 打印 X 数独菜单
 @ 返回值: void
 */
void PrintMenu_X(){
    printf("|**************Menu for X_Sudoku*************|\n");
    printf("|-----------------------------------------|\n");
    printf("|           1. Generate a X_Sudoku          |\n");
    printf("|           2. Play the X_Sudoku            |\n");
    printf("|           3. Reference answer             |\n");
    printf("|                0. EXIT                    |\n");
    printf("|*****************************************|\n\n");
}
/*
 @ 函数名称: Generate_Sudoku
 @ 接受参数: int[][],bool[][],int
 @ 函数功能: 生成数独
 @ 返回值: status
 */
status Generate_Sudoku(int board[SIZE + 1][SIZE + 1], int newBoard[SIZE +
1][SIZE + 1], int newBoard2[SIZE + 1][SIZE + 1], bool isFixed[SIZE + 1][SIZE
+ 1], int num, bool value[SIZE * SIZE * SIZE + 1]){
    char name[100] = "X_Sudoku.cnf"; // 文件名
START:
    srand(time(NULL));
    // 初始化棋盘
    for (int i = 1; i <= SIZE; i++)
        for (int j = 1; j <= SIZE; j++){
            board[i][j] = 0;
            newBoard[i][j] = 0;
            newBoard2[i][j] = 0;
            isFixed[i][j] = TRUE; // 默认全为为提示数字
        }
    int n[SIZE]; // 1-9 的数组
    for (int i = 0; i < SIZE; i++)
```

```
        n[i] = i + 1;
    Shuffle(n, SIZE); // 打乱数组
    int index = 0;
    for (int i = 1; i <= SIZE; i++){ // 对角线填入 1-9
        board[i][i] = n[index];
        newBoard[i][i] = n[index];
        newBoard2[i][i] = n[index];
        index++;
    }
    for (int i = 1; i <= SIZE; i += 3){
        Fill_Box(board, newBoard, newBoard2, i, i); // 主对角线 3 个填充 3x3
的宫格
    }
    WriteToFile(board, 27, name); // 将数独约束条件写入文件
    CNF p = (CNF)malloc(sizeof(cnfNode));
    p->root = NULL;
    ReadFile(p, name); // 读取文件并解析 CNF
    for (int i = 1; i <= SIZE * SIZE * SIZE; i++)
        value[i] = FALSE;
    if (DPLL(p, value, 3) == ERROR) // 求解数独
        goto START;
    // 将 DPLL 的结果填入数独
    for (int i = 1; i <= SIZE * SIZE * SIZE + 1; i++){
        if (value[i] == TRUE){
            int row = (i - 1) / (SIZE * SIZE) + 1;
            int col = (i - 1) / SIZE % SIZE + 1;
            int v = (i - 1) % SIZE + 1;
            board[row][col] = v;
            newBoard[row][col] = v;
            newBoard2[row][col] = v;
        }
    }
    // 挖洞,剩下 num 个提示数
    int remove = 81 - num;
    int single = remove / 9; // single 一定小于等于 7
    int res = remove - 9 * single;
    // int c[SIZE]={9-single};
    for (int row = 1; row <= 9; row++) // 每行挖 single 个{
```

```
        int s = single;
        while (s){
            int col = rand() % SIZE + 1;
            if (board[row][col] != 0){ // 没有被挖
                board[row][col] = 0;
                newBoard[row][col] = 0;
                newBoard2[row][col] = 0;
                isFixed[row][col] = FALSE;
                s--;
            }
        }
    }
    while (res){ // 挖剩下的
        int row = rand() % SIZE + 1;
        // while(c[row-1]<=2)
        // row = rand() % SIZE + 1;
        int col = rand() % SIZE + 1;
        if (board[row][col] != 0){
            board[row][col] = 0;
            newBoard[row][col] = 0;
            newBoard2[row][col] = 0;
            isFixed[row][col] = FALSE;
            res--;
            // c[row-1]--;
        }
    }
    return OK;
}
/*
 @ 函数名称: Is_Valid
 @ 接受参数: int[][],int,int,int
 @ 函数功能: 判断 board[row][col]是否可以填入 v
 @ 返回值: status
 */
status Is_Valid(int board[SIZE + 1][SIZE + 1], int row, int col, int v){
    // 检查行和列
    for (int i = 1; i <= SIZE; i++){
        if (board[row][i] == v || board[i][col] == v) // 行或列有重复
```

```
            return FALSE;
    }
    // 检查 3x3 宫格
    int startRow = (row - 1) / 3 * 3 + 1; // 宫格的起始行
    int startCol = (col - 1) / 3 * 3 + 1; // 宫格的起始列
    for (int i = startRow; i < startRow + 3; i++){
        for (int j = startCol; j < startCol + 3; j++){
            if (board[i][j] == v) // 宫格内有重复
                return FALSE;
        }
    }
    // 检查主对角线
    if (row == col){
        for (int i = 1; i <= SIZE; i++){
            if (board[i][i] == v)
                return FALSE;
        }
    }
    // 检查副对角线
    if (row + col == SIZE + 1){
        for (int i = 1; i <= SIZE; i++){
            if (board[i][SIZE - i + 1] == v)
                return FALSE;
        }
    }
    return TRUE;
}
/*
 @ 函数名称: Print_Sudoku
 @ 接受参数: int[][]
 @ 函数功能: 打印数独
 @ 返回值: void
 */
void Print_Sudoku(int board[SIZE + 1][SIZE + 1]){
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            if (board[i][j] == 0) // 未填入
                printf(" .");
```

```
            else // 已填入
                printf("%2d", board[i][j]);
            if (j % 3 == 0 && j != SIZE) // 每 3 列打印一个竖线
                printf(" |");
        }
        printf("\n");
        if (i % 3 == 0 && i != SIZE) // 每 3 行打印一个横线
            printf("-------+-------+-------\n");
    }
}
/*
 @ 函数名称: Play_Sudoku
 @ 接受参数: int[][], bool[][]
 @ 函数功能: 玩数独的交互界面
 @ 返回值: void
 */
void Play_Sudoku(int board[SIZE + 1][SIZE + 1], bool isFixed[SIZE + 1][SIZE
+ 1]){
    system("cls");       // 清屏
    Print_Sudoku(board); // 打印初始数独
    printf("\n");
    while (1){
        int row, col, v;
        printf(" Please enter the row, col and value(0 to EXIT): ");
        scanf("%d", &row);
        if (row == 0){ // 退出
            system("cls");
            return;
        }
        scanf("%d%d", &col, &v);
        if (row < 1 || row > SIZE || col < 1 || col > SIZE || v < 1 || v >
SIZE){ // 输入不合法
            printf(" Invalid input.\n");
            continue;
        }
        if (isFixed[row][col]){ // 是提示数
            printf(" This is a fixed number.\n");
            continue;
```

```
        }
        if (!Is_Valid(board, row, col, v){ // 不符合数独规则
            printf(" Wrong answer.\n");
            continue;
        }
        else{ // 符合数独规则
            board[row][col] = v;
            system("cls");
            Print_Sudoku(board); // 打印新数独
            printf("\n");
        }
    }
}
/*
 @ 函数名称: WriteToFile
 @ 接受参数: int[][],int
 @ 函数功能: 将数独约束条件写入文件
 @ 返回值: status
 */
status WriteToFile(int board[SIZE + 1][SIZE + 1], int num, char name[]){
    FILE *fp;
    if (fopen_s(&fp, name, "w")){
        printf(" Fail!\n");
        return ERROR;
    }
    fprintf(fp, "c %s\n", name);
    fprintf(fp, "p cnf 729 %d\n", num + 12654); // 12654 是数独的约束条件
    // 数字 ijk 表示第 i 行第 j 列的数字是 k
    // 用公式(i-1)*81+(j-1)*9+k 将每个变元映射到 1-729 的变元上
    /*提示数约束(写在前面,便于单子句规则进行)*/
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            if (board[i][j] != 0)
                fprintf(fp, "%d 0\n", (i - 1) * SIZE * SIZE + (j - 1) * SIZE
+ board[i][j]);
        }
    }
    /*每个格子的约束*/
```

```
    // 每个格子必须填入一个数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                fprintf(fp, "%d ",(i-1) *SIZE * SIZE + (j - 1) * SIZE + k);
            }
            fprintf(fp, "0\n");
        }
    }
    // 每个格子不能填入两个数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                for (int l = k + 1; l <= SIZE; l++){
                    fprintf(fp, "%d %d 0\n", 0 - ((i - 1) * SIZE * SIZE + (j
- 1) * SIZE + k), 0 - ((i - 1) * SIZE * SIZE + (j - 1) * SIZE + l));
                }
            }
        }
    }
    /*行约束*/
    // 每一行必须填入 1-9
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                fprintf(fp, "%d ",(i - 1) *SIZE *SIZE + (k - 1) * SIZE + j);
            }
            fprintf(fp, "0\n");
        }
    }
    // 每一行不能填入两个相同的数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                for (int l = k + 1; l <= SIZE; l++){
                    fprintf(fp, "%d %d 0\n", 0 - ((i - 1) * SIZE * SIZE + (k
- 1) * SIZE + j), 0 - ((i - 1) * SIZE * SIZE + (l - 1) * SIZE + j));
                }
```

```
            }
        }
    }
    /*列约束*/
    // 每一列必须填入 1-9
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                fprintf(fp, "%d ", k - 1) *SIZE *SIZE + (i - 1) * SIZE + j);
            }
            fprintf(fp, "0\n");
        }
    }
    // 每一列不能填入两个相同的数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = 1; k <= SIZE; k++){
                for (int l = k + 1; l <= SIZE; l++){
                    fprintf(fp, "%d %d 0\n", 0 - ((k - 1) * SIZE * SIZE + (i
- 1) * SIZE + j), 0 - ((l - 1) * SIZE * SIZE + (i - 1) * SIZE + j));
                }
            }
        }
    }
    /*3x3 宫格约束*/
    // 每个 3x3 宫格必须填入 1-9
    for (int i = 1; i <= SIZE; i += 3){
        for (int j = 1; j <= SIZE; j += 3){
            for (int k = 1; k <= SIZE; k++){
                for (int l = 0; l < 3; l++){
                    for (int m = 0; m < 3; m++){
                        fprintf(fp, "%d ", ((i + l - 1) * SIZE * SIZE + (j
+ m - 1) * SIZE + k));
                    }
                }
                fprintf(fp, "0\n");
            }
        }
```

```
    }
    // 每个 3x3 宫格不能填入两个相同的数字
    for (int i = 1; i <= SIZE; i += 3){
        for (int j = 1; j <= SIZE; j += 3){
            for (int k = 1; k <= SIZE; k++){
                for (int l = 0; l < 3; l++){
                    for (int m = 0; m < 3; m++){
                        for (int n = k + 1; n <= SIZE; n++){
                            fprintf(fp, "%d %d 0\n", 0 - ((i + l - 1) * SIZE
* SIZE + (j + m - 1) * SIZE + k), 0 - ((i + l - 1) * SIZE * SIZE + (j + m
- 1) * SIZE + n));
                        }
                    }
                }
            }
        }
    }
    /*对角线约束*/
    // 主对角线必须填入 1-9
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            fprintf(fp, "%d ", (j - 1) * SIZE * SIZE + (j - 1) * SIZE + i);
        }
        fprintf(fp, "0\n");
    }
    // 主对角线不能填入两个相同的数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = j + 1; k <= SIZE; k++){
                fprintf(fp, "%d %d 0\n", 0 - ((j - 1) * SIZE * SIZE + (j -
1) * SIZE + i), 0 - ((k - 1) * SIZE * SIZE + (k - 1) * SIZE + i));
            }
        }
    }
    // 副对角线必须填入 1-9
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            fprintf(fp, "%d ",(j - 1) *SIZE* SIZE+ (SIZE - j + 1)*SIZE + i);
```

```
        }
        fprintf(fp, "0\n");
    }
    // 副对角线不能填入两个相同的数字
    for (int i = 1; i <= SIZE; i++){
        for (int j = 1; j <= SIZE; j++){
            for (int k = j + 1; k <= SIZE; k++){
                fprintf(fp, "%d %d 0\n", 0 - ((j - 1) * SIZE * SIZE + (SIZE
- j + 1) * SIZE + i), 0 - ((k - 1) * SIZE * SIZE + (SIZE - k + 1) * SIZE +
i));
            }
        }
    }
    fclose(fp);
    return OK;
}
/*
 @ 函数名称: Slove
 @ 接受参数: int[][],int[]
 @ 函数功能: DPLL 求解数独
 @ 返回值: status
 */
status Slove(int board[SIZE + 1][SIZE + 1], bool value[SIZE * SIZE * SIZE
+ 1]){
    // clauseList p=NULL;
    // char name[100]="X_Sudoku.cnf";
    // if(ReadFile(p,name)==OK)
    // if(DPLL(p,value)==ERROR)
    //     return ERROR;
    // 利用公式(i-1)*81+(j-1)*9+k 反解
    for (int i = 1; i <= SIZE * SIZE * SIZE + 1; i++){
        if (value[i] == TRUE){
            int row = (i - 1) / (SIZE * SIZE) + 1;
            int col = (i - 1) / SIZE % SIZE + 1;
            int v = (i - 1) % SIZE + 1;
            board[row][col] = v;
        }
    }
```

```
    return OK;
}
/*
 @ 函数名称: Fill_Box
 @ 接受参数: int[][],int[][],int[][],int,int
 @ 函数功能: 填充 3x3 的宫格
 @ 返回值: status
 */
status Fill_Box(int board[SIZE + 1][SIZE + 1], int newBoard[SIZE + 1][SIZE
+ 1], int newBoard2[SIZE + 1][SIZE + 1], int rowStart, int colStart){
    int n[SIZE - 3];       // 除了对角线以外的 6 个格子
    bool flag[SIZE] = {0}; // 标记 1-9 是否已经填入
    for (int i = 0; i < 3; i++){
        flag[board[rowStart + i][colStart + i] - 1] = 1; // 对角线的 3 个格
子上的数字标记为已填入
    }
    int index = 0;
    for (int i = 0; i < SIZE; i++){
        if (!flag[i])
            n[index++] = i + 1;
    }
    Shuffle(n, 6); // 打乱数组
    index = 0;
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++){
            if (board[rowStart + i][colStart + j] == 0){// 没有填入
                board[rowStart + i][colStart + j] = n[index];
                newBoard[rowStart + i][colStart + j] = n[index];
                newBoard2[rowStart + i][colStart + j] = n[index];
                index++;
            }
        }
    }
    return OK;
}
/*
 @ 函数名称: Shuffle
 @ 接受参数: int[],int
```

```
 @ 函数功能: 洗牌算法,打乱数组顺序
 @ 返回值: void
 */
void Shuffle(int arr[], int n){
    srand(time(NULL)); // 用时间做种子
    // 每次从后面的数中随机选一个数与前面的数交换
    for (int i = n - 1; i > 0; i--){
        int j = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```