

**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： 计算机23xx班**

**学 号： U2023xxxxxxxxxxxx**

**姓 名： xxxxxxxxxx**

**指导教师： 李剑军**

**报告日期： 2024.9.14**

**计算机科学与技术学院**

**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-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to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%)

**目录**

[1 引言 1](#_Toc6350)

[2. 系统总体设计 4](#_Toc14547)

[2.1系统需求分析 4](#_Toc22503)

[2.1.1 SAT求解器 4](#_Toc29486)

[2.1.2 对角线数独游戏 4](#_Toc5637)

[2.2系统结构分析 5](#_Toc29936)

[3 系统详细设计 7](#_Toc20877)

[3.1有关数据结构的定义 7](#_Toc8328)

[3.1.1 向量类 7](#_Toc21076)

[3.1.2 文字类 8](#_Toc14637)

[3.1.3 子句类 9](#_Toc18072)

[3.1.4 合取范式类 9](#_Toc29797)

[3.2 函数设计与算法实现 10](#_Toc21765)

[3.2.1 核心DPLL模块 10](#_Toc28796)

[3.2.2 CNF解析模块 11](#_Toc13248)

[3.2.3 数独生成模块 12](#_Toc32745)

[3.2.4 数独归约模块 14](#_Toc5311)

[3.2.5 数独检错模块 15](#_Toc30907)

[3.3 DPLL算法优化策略 16](#_Toc31231)

[3.3.1 策略1：选择第一个子句的第一个文字 16](#_Toc28797)

[3.3.2 策略2：随机选择一个非空子句中的一个文字 17](#_Toc22616)

[3.3.3 策略3：选择出现次数最多的文字 17](#_Toc20534)

[3.3.4 策略4：选择权重最大的文字（不考虑正负） 18](#_Toc26109)

[3.3.5 策略5：选择权重最大的文字（考虑正负） 19](#_Toc18802)

[3.3.6总结 20](#_Toc6634)

[3.4 系统开发环境 21](#_Toc15470)

[4 系统测试 22](#_Toc19404)

[4.1 SAT求解 22](#_Toc24806)

[4.1.1 功能测试 22](#_Toc8233)

[4.1.2 性能测试 24](#_Toc19993)

[4.1.3 用户交互测试 25](#_Toc2773)

[4.2 数独游戏测试 26](#_Toc9196)

[4.2.1 数独生成测试 26](#_Toc16883)

[4.2.2 数独交互测试 27](#_Toc26217)

[4.3 数独游戏归约为SAT问题求解 30](#_Toc3602)

[5 总结与体会 31](#_Toc26087)

[5.1全文总结 31](#_Toc2089)

[5.2个人体会 31](#_Toc8593)

[参考文献 33](#_Toc7294)

[附录A main.cpp 34](#_Toc16183)

[附录B DPLL-Solver.cpp 43](#_Toc8638)

[附录C GenerateXSudoku.cpp 51](#_Toc19078)

[附录D SudokuToCnf.cpp 54](#_Toc3530)

**1引言**

**1.1课题背景与意义**

在逻辑和计算机科学中，布尔可满足性问题（有时称为命题可满足性问题，缩写为SATISFIABILITY、SAT或B-SAT）是一个经典问题，其核心在于判断一个布尔公式是否存在一个变量赋值使其为真。SAT问题不仅在理论计算机科学中占据重要地位，而且在实际应用中也发挥着关键作用，如在硬件验证、软件测试、人工智能等领域。随着技术的不断发展，SAT问题的求解变得越来越重要。

由于它的应用广泛，且可以规约为 SAT 的问题种类繁多，这个困难的组合问题一直吸引着研究人员的关注。就理论研究而言，许多组合问题都可以用命题公式表示，并通过运行 SAT 求解器来解决，例如图着色、顶点覆盖和团检测等问题。它还是自动定理证明的有用工具，其中一个典型案例就是凯勒猜想的解决。此外 SAT 求解还有很多工业应用，如电路设计中的有界模型检查、配置管理和等效性检查。它也是逻辑综合中的一个重要组成部分，许多 SAT 求解器都是专门为此任务而设计的。因此，SAT 求解不仅有重要的研究意义，还能帮助工业界实现更经济的工作流程。

由于P/NP问题仍未解决，研究人员对SAT问题的难度心怀敬畏，并努力设计高效的 SAT 求解器。在SAT问题中，DPLL算法的引入和应用具有重要的历史意义。该算法在布尔公式的求解中，通过对公式进行递归分解和回溯搜索，有效地减少了计算复杂度。DPLL算法引入了单位传播和分裂启发式等技术，这些技术极大地提升了算法的性能。研究人员不断优化DPLL算法，以应对越来越复杂的SAT实例，从而推动了SAT求解技术的发展。

本课题研究基于DPLL算法的SAT求解器并寻找优化策略，其具有重要的理论和实际意义。首先，DPLL算法作为SAT求解的基础算法，其性能优化直接影响到SAT问题的解决效率。通过深入研究DPLL算法，可以揭示其在处理大规模SAT实例时的表现，为改进算法提供理论支持。其次，DPLL算法的研究和应用不仅推动了SAT求解技术的进步，还为相关领域的算法研究和工程应用提供了宝贵的经验。尤其是在硬件设计验证、自动推理等实际应用中，DPLL算法的有效实现能够显著提高问题解决的效率和准确性。

**1.2国内外研究现状**

SAT问题自1960年逐渐受到关注，由于其在理论和实际应用中的重要性，成为了国内外研究的热点。自2002年起，每隔一两年举行的SAT竞赛旨在寻找高性能的求解器，以实现快速求解SAT问题，并公开高性能求解器的源代码。这一措施有效推动了SAT问题的快速发展。以下是SAT问题研究的发展历程。

1960 年，Davis 和 Putnam提出 DPLL(Davis-Putnam-Logemann-Loveland) 算法,它是首个求解 SAT 问题的完备算法。大部分现有的精确算法的SAT 求解器都是基于DPLL算法.

1962年，Martin Davis, George Logemann and Donald W. Loveland共同提出了改进版本。它提供了一个基于树搜索和回溯方法来枚举可能的赋值。

1971 年，Stephen Arthur. Cook 证明了 SAT 问题是 NP 完全问题。

1992 年，Bart Selman[11]等人提出贪心搜索算法（简称 GSAT 算法）。

1997年，李初民教授开发出了 著 名 的satZ 求 解 器，首次将单子句传播技术应用在分支启发式策略中

1998 年作者梁东敏提出了改进的子句加权 WSAT 算法。

1999 年，João Marques Silva 等人在 DPLL 算法的基础上提出了 GRASP 算法，首次引入了冲突学习回溯策略。

2000年金人超和黄文奇提出的并行 Solar 算法，2002 年作者张德富在文献中，提出模拟退火算法。

2003 年，Eén[18]研发出 MiniSAT 求解器，大多数求解器都是基于 MiniSAT 求解器上的改进，是高性能的求解器之一。

2005 年，Niklas Eén 等人提出的 SatEliteE 首次实现了预处理简化问题的规模和复杂性。

2007 年，Bart Selman 和 Henry Kautz 对当前 SAT 问题的现状做了全面的整理和叙述。

2009 年，Simon, Laurent 等人在2009年提出基于MiniSATSAT求解器CDCL(Conflict Driven Clause Learning，CDCL)

在此之后，每年的SAT竞赛上，许多改进的算法不断被提出，都对SAT问题的研究和求解起了大大的推动作用。

2009 年，Audemard 等人研发出 Glucose 求解器，首次提出文字距离块(Literals Blocks Distance，LBD)的概念，它被用于评估学习子句的质量。

2014 年，Oh基于 Glucose2.3求解器研发出 SWDiA5BY 求解器。

2016 年，Liang等人研发出 MapleCOMSPS 求解器，在 2016 年 SAT 竞赛中的Main Track 组获得第一名。

2018 年，Devriendt研发出 InIDGlucose 求解器，主要考虑了初始变量的选择。

2022年，华中科技大学计算机学院何琨教授团队提出将随机游走策略与决策树模型相结合的创新方法，使求解器在面对具有不同特征的算例时采用不同的随机游走策略辅助搜索，很好地提升了SAT求解器的鲁棒性。

**1.3课程设计的主要研究工作**

本次课设主要研究SAT问题的理论知识，并学习基于DPLL算法对SAT进行求解和算法优化，在此基础上，设法将对角线数独游戏转化为SAT问题并集成到上面的求解器进行求解。具体而言，课程设计的主要研究工作包括：

1. 对SAT问题的概念，背景，研究意义和研究现状进行了解和研究，学习理论知识。
2. 了解DPLL算法的基本原理和关键技术，如单位传播、分裂启发式，然后设计并实现一个基于DPLL算法。
3. 通过测试不同规模和复杂度的SAT实例，评估实现程序的性能，包括求解速度和准确性。基于测试结果，对算法和程序进行优化，提出改进策略，以提高求解效率和扩展适用范围。
4. 将对角线数独游戏进行问题建模，将对角线数独的约束条件转化为布尔公式。对于每个数独格子，我们引入布尔变量表示其可能的值，并为行、列、九宫格以及对角线约束构建相应的逻辑公式，将构建的布尔公式输入到SAT求解器中，利用DPLL算法求解这些公式，以确定数独的解。

# 2. 系统总体设计

## **2.1系统需求分析**

## **2.1.1 SAT求解器**

SAT求解器应具有以下功能：

1. 输入功能：用户可以输入参数决定要使用的功能（读取文件，输出CNF内容，DPLL求解，返回等）。用户可以输入cnf文件的路径，程序将从中读取内容到CNF的数据结构中。程序应具备检错功能以防止用户的错误输入，例如在没有读取文件的情况下就进行求解。
2. 输出功能：程序可以将对cnf算例文件求解的结果保存到同名的文件中并将运行结果打印到屏幕上（是否满足，可满足时各变量的赋值状态，运行时间，优化率等）。
3. 求解功能：通过调用DPLL函数，对读入的cnf算例文件进行求解，用户应能在这一步时选择变元选取策略。
4. 界面功能：系统应提供简单的界面功能用于交互，通过合理的排版，清屏功能提供良好的用户体验。

## **2.1.2 对角线数独游戏**

对角线数独游戏部分应具备以下功能：

生成数独：通过挖洞法，随机生成一个数独。

游玩交互：用户可以输入单元格坐标在指定位置填入数字，撤销操作，重新开始游戏，查看答案，或调用SAT求解器对数独进行求解。

检错功能：程序能用户游玩时的输入进行检查，如输入的坐标和数据是否在有效范围内，输入是否违反了数独游戏的规则，输入的坐标是否是初始数独棋盘已经提供的。

界面功能：通过合理布局和各种符号的使用，提供一个视觉效果良好的数独面板，并具备清屏功能，返回功能等。

DPLL求解功能：程序应能将生成的数独游戏转化为SAT问题，即将局面转化为一个cnf文件，并调用之前的SAT求解器进行求解，将最终各单元格的赋值结果打印到屏幕上。

## **2.2系统结构分析**

本系统要求实现一个具有简单菜单功能的SAT求解器和对角线数独游戏。本系统应具备以下功能模块，除主控模块外，每个模块由对应的.h头文件和.cpp文件组成。

1. 主控与显示模块(Display)：系统在main.cpp声明了需要使用的变量，通过多个while,switch,system(“pause”)和system(“cls”)语句实现了菜单功能和交互功能，根据用户输入的参数，调用相关的功能模块。
2. cnf解析模块(CNFParser)：本模块主要由CNF类和CNFParser类构成，通过成员函数parser，传入文件名和对象cnf，将文件内容读取到cnf中，当读取错误时，模块能够返回错误信息。
3. 核心DPLL模块(DPLL-Solver)：本模块实现了DPLL求解功能，主要由三个函数构成，即DPLL主框架，变元决策函数和分裂函数（传播函数）。此外，还设置了两个输出函数用于将求解结果显示在屏幕上或保存到同名的.res文件中。
4. 对角线数独游戏模块(XSudoku Game):本模块主要处理数独游戏的生成，归约，求解和对于用户输入检错功能。因此本模块又可分为四个子模块：
5. 数独游戏生成模块：本项目采用了递归回溯的算法生成对角线数独格局，其功能函数包括：随机生成第一行，检查当前位置的可能性，递归主函数。
6. 数独游戏归约模块：本模块需要将生成的数独游戏格局归约为cnf格式，因此在资源文件中设置了对于的cnf文件，通过传入数独游戏格局，将转化后的内容输出并保存到对应的cnf文件中。
7. 数独游戏求解模块：该模块实际只需调用之前的DPLL模块即可，但需要对输出函数进行改进，只需要显示正文字为真的赋值结果（即每个单元格对应的数字）。
8. 游戏检错模块(Check Game)：本模块能够对用户的游玩局面进行实时检错，当用户输入的内容将导致数独游戏的规则冲突时，程序会提醒用户输入错误，重新输入。



图2-1 程序流程图

**3 系统详细设计**

**3.1有关数据结构的定义**

本系统主要要处理的数据包括cnf(合取范式),其中cnf又包括子句和文字；DPLL时各变元的赋值状态；数独游戏格局。由于课设要求不能使用std现有的vector类，因此我们可以先自行实现一个vector类，再据此设计cnf类（因为子句数量和子句长度是非固定的，因此我们需要一个可变长的数组），尽管该存储结构在处理某些cnf时会陷入劣势。

**3.1.1 向量类**

该类主要仿照std标准库中的vector类，根据需要只实现了其基础功能。

1. template <class T>
2. class vector {
3. public:
4. *// 数据*
5. T\* data;
6. *// 大小*
7. int Size;
8. *// 容量*
9. int capacity;
10. *// 构造函数*
11. vector() : Size(0), capacity(1), data(new T[1]) {}
12. *// 析构函数*
13. ~vector() {
14. delete[] data;
15. }
16. *//含参构造函数(初始化大小,元素)*
17. vector(int n, T t) : Size(n), capacity(n), data(new T[n]);
18. *// 拷贝构造函数*
19. vector(const vector& v) : Size(v.Size), capacity(v.capacity), data(new T[v.capacity]) {
20. std::copy(v.data, v.data + v.Size, data);
21. }
22. *// 赋值运算符*
23. vector& operator=(const vector& v);
24. *// 添加元素*
25. void push\_back(T t) ;
26. *// 删除元素*
27. void pop\_back() *// 删除最后一个元素*
28. {
29. if (Size > 0) Size--;
30. }
31. *// 返回大小*
32. int size() const {
33. return Size;
34. }
35. *// 返回元素（非 const 版本）*
36. T& operator[](int index);
37. *// 返回元素（const 版本）*
38. const T& operator[](int index) const;
39. *//判断是否为空*
40. bool empty() const;
41. *//清空*
42. void clear();
43. *// 调整容量*
44. void resize(int new\_capacity);
45. };

**3.1.2 文字类**

文字类主要记录布尔变元的编号，定义如下。

1. class Literal
2. {
3. public:
4. *//布尔变量*
5. int var;
6. Literal();
7. *//构造函数*
8. Literal(int var);
9. *//析构函数*
10. ~Literal();
11. *//是否是负文字*
12. bool isNegative();
13. *//返回绝对值*
14. int getAbs();
15. *//打印文字*
16. void print();
17. *//不等号重载*
18. bool operator!=(const Literal& l)const;
19. *//等号重载*
20. bool operator==(const Literal& l)const;
21. };

**3.1.3 子句类**

子句类主要成员为文字的向量，同时根据DPLL算法设计了removeLiteral等成员函数

1. class Clause
2. {
3. public:
4. *//文字集合*
5. vector<Literal> literals;
6. *//构造函数*
7. Clause();
8. *//析构函数*
9. ~Clause();
10. *//添加文字*
11. void addLiteral(Literal l);
12. *//删除所有特定文字*
13. void removeLiteral(const Literal& l);
14. *//打印子句*
15. void print();
16. *//判断是否是单子句*
17. bool isUnit()const;
18. };

**3.1.4 合取范式类**

CNF类主要成员为子句的向量，设计了removeClause和removeLiteral两个成员函数以便于DPLL算法中传播函数的实现。

1. class CNF
2. {
3. public:
4. *//子句集合*
5. vector<Clause> clauses;
6. *//构造函数*
7. CNF();
8. *//添加子句*
9. void addClause(Clause c);
10. *//移除所有包含单子句L的子句*
11. void removeClause(const Clause& clause);
12. *//移除剩下所有子句中的~L*
13. void removeLiteral(const Literal& L);
14. bool isEmpty() const {
15. return clauses.empty();
16. }
17. bool hasNext() const {
18. *// 如果列表中至少有一个子句，且不是空子句，则返回true*
19. return !clauses.empty() && !clauses[0].literals.empty();
20. }
21. void print();
22. };

**3.2 函数设计与算法实现**

由于各模块设计的函数过多，这里仅对各个模块中主要函数的实现进行介绍，源码位于附录中。

**3.2.1 核心DPLL模块**

1. bool DPLL(CNF& cnf,int way, int\* assignments)

(1) 输入：

CNF& cnf：布尔公式，合取范式（CNF）形式。

int way：启发式搜索策略，用于选择变量。

int\* assignments：当前变量赋值数组。

(2) 输出：如果公式可满足返回true，否则返回false。

(3) 算法思想描述：DPLL算法通过递归和回溯方法解决SAT问题。算法主要包括单子句传播、选择变量赋值、递归求解和回溯。

(4) 算法处理步骤：

1. 单子句传播：处理所有单子句，传播其赋值。
2. 检查公式：如果所有子句满足，返回true。
3. 选择并尝试赋值：选择一个变量，尝试赋值为true或false，递归求解。
4. 回溯：如果赋值导致冲突，回溯并尝试其他赋值。

(5) 时间复杂度： O(2^n)，n为变量数量，因需要探索所有可能赋值组合。

(6) 空间复杂度：O(n + m)，n为变量数量，m为子句数量，包括递归栈和CNF副本的存储。

2.bool propagate(CNF& cnf, const Literal& literal,int \*assignments);

(1) 输入：CNF& cnf：当前的布尔公式，以合取范式（CNF）形式表示。

const Literal& literal：要传播的文字（变量及其取值）。

int\* assignments：记录变量赋值的数组。

(2) 输出：bool：如果传播成功且CNF无冲突，返回true；如果出现空子句，返回false。

(3) 算法思想描述：该函数用于传播文字literal的赋值，更新CNF公式，并记录该赋值。如果传播过程中出现空子句（即存在无法满足的子句），则返回false，否则返回true。

(4) 算法处理步骤：

1. 遍历子句：检查CNF中的每个子句。
2. 检查文字：如果子句中包含literal，该子句已被满足，不需要进一步处理。如果子句中包含~literal（literal的否定），删除这个文字。
3. 更新子句：
4. 如果删除~literal后子句仍不为空，将其加入到新的CNF中。如果出现空子句，返回false。
5. 记录赋值：更新变量赋值数组assignments，记录literal的赋值。更新CNF为新的CNF。

(5) 时间复杂度： O(m \* k)，其中m为子句数量，k为每个子句中文字的数量。每个子句都可能需要遍历和修改。

(6) 空间复杂度：

O(m \* k)，需要存储新的CNF子句。

**3.2.2 CNF解析模块**

1.bool parse(CNF& cnf,const string& filename, int& NumVars, int& NumClauses)

(1) 输入：

CNF& cnf：要填充的布尔公式（CNF）对象。

const string& filename：包含CNF公式的文件名。

int& NumVars：引用参数，用于返回CNF公式中的变量数量。

int& NumClauses：引用参数，用于返回CNF公式中的子句数量。

(2) 输出：bool：成功读取并解析文件返回true，失败返回false。

(3) 算法思想描述：该函数从指定的文件中读取CNF格式的布尔公式，并解析变量数量、子句数量以及子句内容，填充到CNF对象中。如果文件无法打开或格式错误，返回false。

(4) 算法处理步骤：

1. 打开文件：尝试打开指定文件。如果打开失败，输出错误信息并返回false。
2. 清空现有CNF：如果CNF对象已有子句，先清空它以准备加载新的数据。
3. 逐行读取文件：逐行读取文件内容，跳过注释和空行。如果行以p开头，解析变量数和子句数。对于其他行，解析出子句中的文字，并将其添加到CNF对象中。
4. 关闭文件：读取完成后，关闭文件并返回true表示成功。

(5) 时间复杂度：O(m \* k)，其中m为文件中的行数（对应子句数量），k为每行的文字数量（每个子句中变量的数量）。每行都需要解析和存储。

(6) 空间复杂度：O(m \* k)，需要存储解析出的所有子句和文字

**3.2.3 数独生成模块**

1. bool solveSudoku(vector<int>& board)

(1) 输入：vector<int>& board：一个一维数组，表示数独棋盘，其中0表示空格，1至9表示已填入的数字。

(2) 输出：bool：如果成功解决数独返回true，否则返回false。

(3) 算法思想描述：该函数使用回溯法解决数独生成问题。它在棋盘上寻找空格，然后尝试填入数字。如果某个数字有效且填入后可以继续生成数独，函数将递归调用自身；如果不行，则进行回溯（撤销当前填入的数字并尝试下一个数字）。

(4) 算法处理步骤：

1. 遍历棋盘：使用两层循环遍历棋盘的每个位置。
2. 寻找空格：如果找到空格（即值为0），进入下一步。
3. 尝试填入数字：对每个数字（1到SIZE）进行检查：使用isSafe(board, row, col, num)函数检查该数字在当前位置是否安全。如果安全，将数字填入棋盘。
4. 递归调用：递归调用solveSudoku(board)以继续解决数独。
5. 如果递归成功（返回true），则完成解决。
6. 回溯：如果填入数字后无法继续解决，撤销该填入（将该位置重置为0）。
7. 返回结果：如果棋盘已完全填满且没有空格，则返回true；如果遍历完所有位置都没有找到可以填入的数字，则返回false。

(5) 时间复杂度： 在最坏情况下，时间复杂度为O(9^(n^2))，n为数独的行列数，考虑每个空位都可能填入1至9中的任何数字。

(6) 空间复杂度：O(n^2)，主要是递归调用栈的空间，n为数独的行列数。

1. void GenerateFirstLine(vector<int>& a)

(1) 输入：vector<int>& a：一个一维数组，用于存储生成的第一行。

(2) 输出：void：无返回值。函数直接修改输入的数组a。

(3) 算法思想描述：该函数生成一个数独的第一行。确保该行中的数字是1至SIZE的排列，且没有重复。利用随机数生成数字，并通过检查避免重复。

(4) 算法处理步骤：

1. 遍历数组：遍历数组的每个位置，以生成数字。
2. 生成随机数字：使用rand() % SIZE + 1生成1到SIZE范围内的随机数。
3. 检查重复：检查当前生成的数字是否与之前位置的数字重复。
4. 如果重复，重新生成数字，并从头开始重新检查（设置j = 0）。
5. 继续检查：如果没有重复，继续检查下一个位置。

(5) 时间复杂度： 最坏情况下为O(SIZE^2)，因为每次生成数字后可能需要检查所有之前的位置以确保无重复。

(6) 空间复杂度：O(1)，仅使用了常量级的额外空间，修改原数组a。

1. bool isSafe(const vector<int>& board, int row, int col, int num)

(1) 输入：const vector<int>& board：一个一维数组，表示数独棋盘。

int row：待检查的行索引。

int col：待检查的列索引。

int num：待检查的数字。

(2) 输出：bool：如果在指定位置放置数字num是安全的（即不违反数独规则），返回true；否则返回false。

(3) 算法思想描述：该函数检查在数独棋盘的指定位置放置数字num是否满足数独的所有规则

(4) 算法处理步骤：

1. 检查行：遍历当前行的所有列，确保num不在该行中出现。
2. 检查列：遍历当前列的所有行，确保num不在该列中出现.
3. 检查九宫格：计算3x3子网格的起始行列索引。遍历子网格中的所有位置，确保num不在该子网格中出现。
4. 检查主对角线：如果当前位置在主对角线上，遍历主对角线的所有位置，确保num不在主对角线中出现。
5. 检查副对角线：如果当前位置在副对角线上，遍历副对角线的所有位置，确保num不在副对角线中出现。

(5) 时间复杂度： O(SIZE)：由于需要检查行、列、九宫格以及对角线的位置，时间复杂度为O(SIZE)，其中SIZE通常为9（数独棋盘的行列数）。

(6) 空间复杂度：O(1)：只使用了常量级的额外空间来存储变量，不依赖于输入数据规模。

**3.2.4 数独归约模块**

1.bool XSudokuToCnf(const vector<int>& board, int empty)

(1) 输入：const vector<int>& board：一个一维数组表示数独棋盘，长度为81（9x9），数组中的值代表每个位置的数字（0表示空白）。

int empty：表示空白位置的数量，用于计算最终的子句数量。

(2) 输出：bool：如果成功将生成的CNF文件写入磁盘，返回true；否则返回false。

(3) 算法思想描述：该函数将数独问题转换为CNF（合取范式）格式以供SAT求解器使用。它通过读取源CNF文件，生成与数独规则对应的CNF子句，并将这些子句写入目标CNF文件。规则包括主对角线和副对角线的数字不能重复，以及数独棋盘上已有数字的约束。

(4) 算法处理步骤：

1. 打开文件：打开源文件SudokuBase.cnf进行读取。打开目标文件XSudoku.cnf进行写入。如果目标文件不存在则创建。
2. 写入文件头：写入CNF文件头信息，包括问题类型p cnf，变量数量（729），子句数量（8829+648+81-空白位置）。
3. 生成主对角线的约束：对于每个数字，确保主对角线上的不同位置不含相同数字。写入相应的CNF子句。
4. 生成副对角线的约束：对于每个数字，确保副对角线上的不同位置不含相同数字。写入相应的CNF子句。
5. 生成棋盘已有数字的约束：根据棋盘上已有的数字，生成约束子句确保这些数字在相应的位置上。
6. 复制基础约束内容：从源文件SudokuBase.cnf中复制已有的基础约束子句并写入目标文件。
7. 关闭文件：关闭源文件和目标文件。

(5) 时间复杂度：O(N^2)，其中N为数独的维度（通常为9），因为需要生成和写入主对角线、副对角线和已有数字的约束，这些操作都是依赖于N^2的。

(6) 空间复杂度：O(N^2)，主要用于存储生成的CNF子句，以及在内存中处理文件读写操作。这也包括临时缓冲区的空间。

**3.2.5 数独检错模块**

1.bool CheckBoard(const vector<int>& board, int x, int y, int num)

(1) 输入：

1. const vector<int>& board：一个一维数组，表示数独棋盘，长度为81（9x9），数组中的值表示每个位置的数字（0表示空白）。
2. int x：要检查的行索引（从1开始计数）。
3. int y：要检查的列索引（从1开始计数）。
4. int num：要放置在位置(x, y)的数字。

(2) 输出：

(3) 算法思想描述：该函数检查在数独棋盘的指定位置(x, y)放置数字num是否仍然符合数独规则。具体包括：

1. 检查所在行是否有重复数字。
2. 检查所在列是否有重复数字。
3. 检查所在3x3子网格是否有重复数字。

(4) 算法处理步骤：

1. 创建当前棋盘的副本：将输入棋盘board复制到一个新的数组nowboard。在副本中，将位置(x, y)的数字设置为num。
2. 检查行：调用CheckRow函数检查修改后的棋盘在指定行是否有重复的数字。
3. 检查列：调用CheckColumn函数检查修改后的棋盘在指定列是否有重复的数字。
4. 检查3x3方框：计算3x3子网格的起始行和列。调用CheckBox函数检查修改后的棋盘在指定3x3子网格内是否有重复的数字。
5. 返回结果：如果行、列和3x3子网格检查都通过，则返回true；否则返回false。

(5) 时间复杂度： O(N)，其中N为数独的维度（通常为9）。检查行、列和3x3子网格的操作都是常数时间的操作，因此时间复杂度为O(N)。

(6) 空间复杂度：O(N^2)，主要用于存储棋盘的副本nowboard，其中N为数独的维度（通常为9）。

**3.3 DPLL算法优化策略**

在DPLL算法（Davis-Putnam-Logemann-Loveland）中，选择合适的决策变量策略对算法的效率和性能有很大影响、在本项目中，决策变量选择函数提供了五种不同的策略来选择决策变量。以下是对这些策略的分析：

**3.3.1 策略1：选择第一个子句的第一个文字**

1. if (way == 2)
2. {
3. if (!cnf.clauses.empty() && !cnf.clauses[0].literals.empty()) *//如果子句不为空*
4. {
5. return cnf.clauses[0].literals[0];
6. }
7. }

优点：实现简单，代码直观。

缺点：

(1)选择的文字可能不具备代表性。它只选择第一个子句的第一个文字，可能导致决策不佳。

(2)对于复杂的CNF公式，这种策略可能会导致较高的回溯次数。

适用场景：适用于小型算例或测试基础功能时，对于初学者来说是最容易想到的策略。

**3.3.2 策略2：随机选择一个非空子句中的一个文字**

1. else if (way == 1)
2. {
3. int clauseIndex = rand() % cnf.clauses.size();
4. while (cnf.clauses[clauseIndex].literals.empty()) {
5. clauseIndex = rand() % cnf.clauses.size();
6. }
7. int literalIndex = rand() % cnf.clauses[clauseIndex].literals.size();
8. return cnf.clauses[clauseIndex].literals[literalIndex];
9. }

优点：

(1)引入随机性，可以避免某些特定结构导致的困境。

(2)实现简单，能够有效减少局部最优解的陷阱。

缺点：

(1)随机性可能导致选择不具代表性的文字，从而可能导致较高的回溯次数。

(2)不保证一定能找到最优解或高效解。

适用场景：适用于求解复杂问题时，作为启发式方法补充使用。

**3.3.3 策略3：选择出现次数最多的文字**

1. else if (way == 3)
2. {
3. vector<int> cnt(numVars \* 2 + 1, 0);
4. for (const Clause& clause : cnf.clauses) {
5. for (const Literal& literal : clause.literals) {
6. if (literal.var > 0) {
7. cnt[literal.var]++;
8. }
9. else {
10. cnt[numVars - literal.var]++;
11. }
12. }
13. }
14. int maxBool = 0;
15. int maxTimes = 0;
16. for (int i = 1; i <= numVars; ++i) {
17. if (cnt[i] > maxTimes) {
18. maxTimes = cnt[i];
19. maxBool = i;
20. }
21. }
22. if (maxTimes == 1) {
23. for (int i = numVars + 1; i <= numVars \* 2; ++i) {
24. if (cnt[i] > maxTimes) {
25. maxTimes = cnt[i];
26. maxBool = numVars - i;
27. }
28. }
29. }
30. return Literal(maxBool);
31. }

优点：

(1) 选择出现次数最多的文字可以提高启发式选择的质量，因为这些文字在更多的子句中出现，可能对许多子句有影响。

(2)可以减少回溯次数，实现快速传播，提高算法的效率。

缺点：

(1)在某些情况下，频繁出现的文字可能被过度选择，导致某些子句被不必要地“填满”。

(2)需要额外的计数和计算开销，在某些复杂算例中可能陷入困境。

适用场景： 适用于需要优化决策变量选择的情况，尤其是在处理复杂的CNF公式时，在中型算例的求解上具备较大优势。

**3.3.4 策略4：选择权重最大的文字（不考虑正负）**

1. else if (way == 4)
2. {
3. vector<double> weight(numVars + 1, 0.0);
4. for (const Clause& clause : cnf.clauses) {
5. double clauseWeight = pow(2.0, -static\_cast<double>(clause.literals.size()));
6. for (const Literal& literal : clause.literals) {
7. if (literal.var != 0) {
8. int varIndex = abs(literal.var);
9. if (varIndex <= numVars) {
10. weight[varIndex] += clauseWeight;
11. }
12. }
13. }
14. }
15. double maxWeight = 0.0;
16. int maxBool = 0;
17. for (int i = 1; i <= numVars; ++i) {
18. if (weight[i] > maxWeight) {
19. maxWeight = weight[i];
20. maxBool = i;
21. }
22. }
23. return Literal(maxBool);
24. }

优点：

(1) 权重的计算考虑了子句的数量，这样可以选择那些影响力较大的文字。

(2) 通常比单纯的计数方法更有效，因为它考虑了子句的长度。

缺点：

(1)计算复杂度较高，特别是当子句较多时。

(1)没有考虑文字的正负符号。

适用场景：适用于需要权重计算的情况，能够帮助选择对更多子句有影响的文字。

**3.3.5 策略5：选择权重最大的文字（考虑正负）**

1. else
2. {
3. vector<double> weight(numVars \* 2 + 1, 0.0);
4. for (const Clause& clause : cnf.clauses) {
5. double clauseWeight = pow(2.0, -static\_cast<double>(clause.literals.size()));
6. for (const Literal& literal : clause.literals) {
7. int varIndex = std::abs(literal.var);
8. if (literal.var > 0) {
9. weight[varIndex] += clauseWeight;
10. }
11. else {
12. weight[numVars - literal.var] += clauseWeight;
13. }
14. }
15. }
16. double maxWeight = 0.0;
17. int maxBool = 0;
18. for (int i = 1; i <= numVars; ++i) {
19. double totalWeight = weight[i] + weight[i + numVars];
20. if (totalWeight > maxWeight) {
21. maxWeight = totalWeight;
22. maxBool = i;
23. }
24. }
25. if (weight[maxBool] < weight[maxBool + numVars]) {
26. maxBool = -maxBool;
27. }
28. return Literal(maxBool);
29. }

优点：

(1)综合考虑了文字的正负符号，可以更准确地选择最具影响力的文字。

(2)权重计算可以更好地反映文字对CNF公式的整体影响。

缺点：

(1) 计算复杂度较高，尤其是在处理大量文字时。

(2) 需要额外的计算来确定正负符号。

适用场景：适用于需要精准选择影响力较大的文字的情况，尤其是在复杂的CNF公式中。

**3.3.6总结**

每种策略都有其适用场景和优缺点。选择合适的策略通常取决于具体的应用需求和问题的复杂性。策略1和策略2适合简单或实验性场景，策略3适合需要启发式选择的场景，策略4和策略5则适用于需要权重计算的更复杂的情况。

**3.4 系统开发环境**

本次实验中使用的环境配置如下：

（1）操作系统版本：Microsoft Windows 11 家庭中文版 X64

（2）平台工具集：Visual Studio 2022 (v143)

（3）C++语言标准： ISO C++14 标准

（4）运行库：多线程调试 DLL (/MDd)

**4 系统测试**

随着软件的不断发展和复杂化，潜在的漏洞、错误和性能问题也越来越多，软件测试就成了确保软件质量和可靠性的关键步骤。常见的软件测试方法包括单元测试，集成测试，系统测试，安全测试，用户界面测试等。考虑到本系统的功能和要求，这里主要对功能，性能，用户界面进行测试.

在main函数中构建功能演示菜单，并对系统进行测试，如图

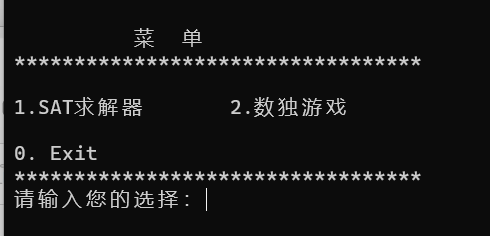


图4-1 功能演示菜单

**4.1 SAT求解**

**4.1.1 功能测试**

表4-1 基准算例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例名称 | 预期结果 | 实际结果 |
| 20 | 91 | sat-20.cnf | 可满足 | 可满足 |
| 30 | 420 | unsat-5cnf-30.cnf | 不可满足 | 不可满足 |

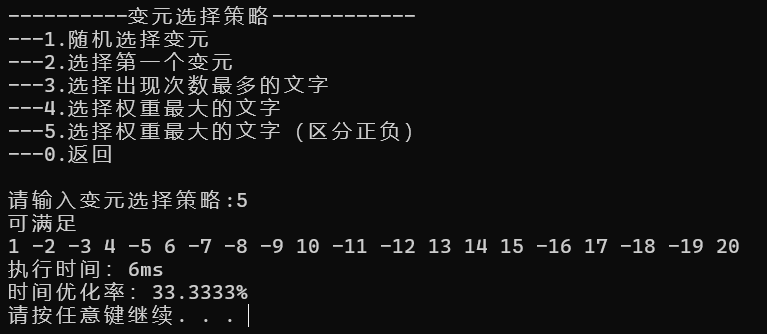


图4-2 基准算例1的运行结果

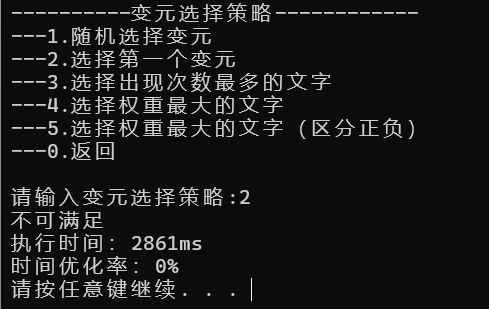


图4-3 基准算例2的运行结果

调用打印CNF文件功能，并与源文件比较，结果与预期相符，表明cnf文件的内容被成功读取

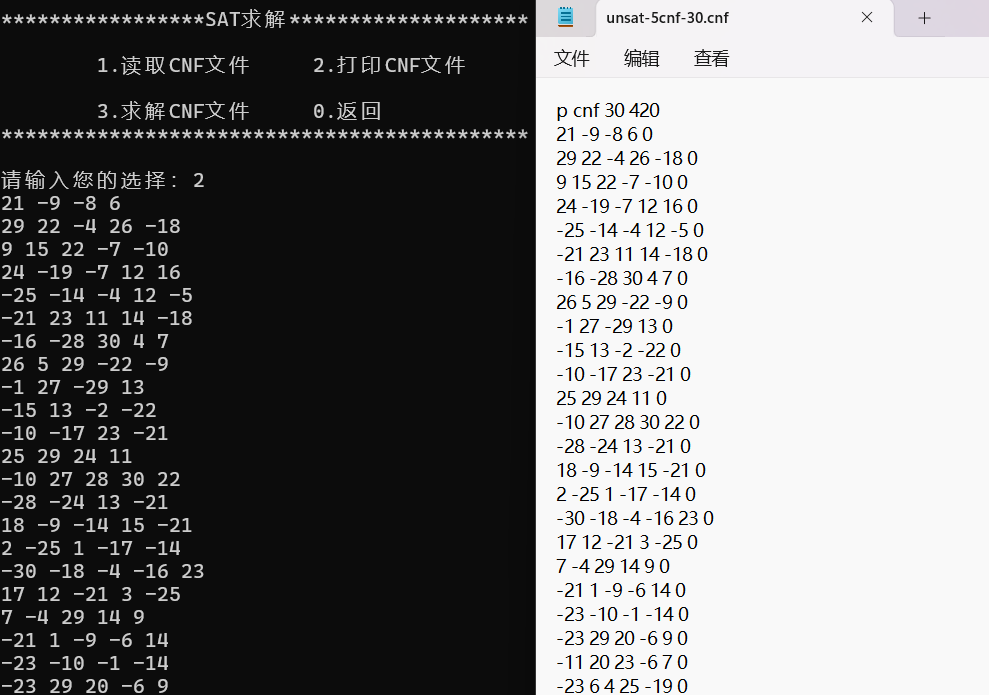


图4-4 读取并打印CNF文件的功能演示

通过以上测试，可见程序功能基本无误。

**4.1.2 性能测试**

表4-2 性能测试算例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例名称 | 优化前 | 优化后 | 优化率 |
| 181 | 3151 | ais10.cnf | 179918ms | 1200ms | 99.33% |
| 303 | 2851 | sud00009.cnf | 13986ms | 659ms | 95.2881% |
| 100 | 600 | problem11-100.cnf | 630 | 12637 | -1900.059% |
| 100 | 200 | problem9-100.cnf | 1342 | 380 | 71.6841% |
| 297 | 2721 | sud00861.cnf | 8170 | 2603 | 68.14% |
| 308 | 2911 | sud00021.cnf | 83515 | 20063 | 75.97% |

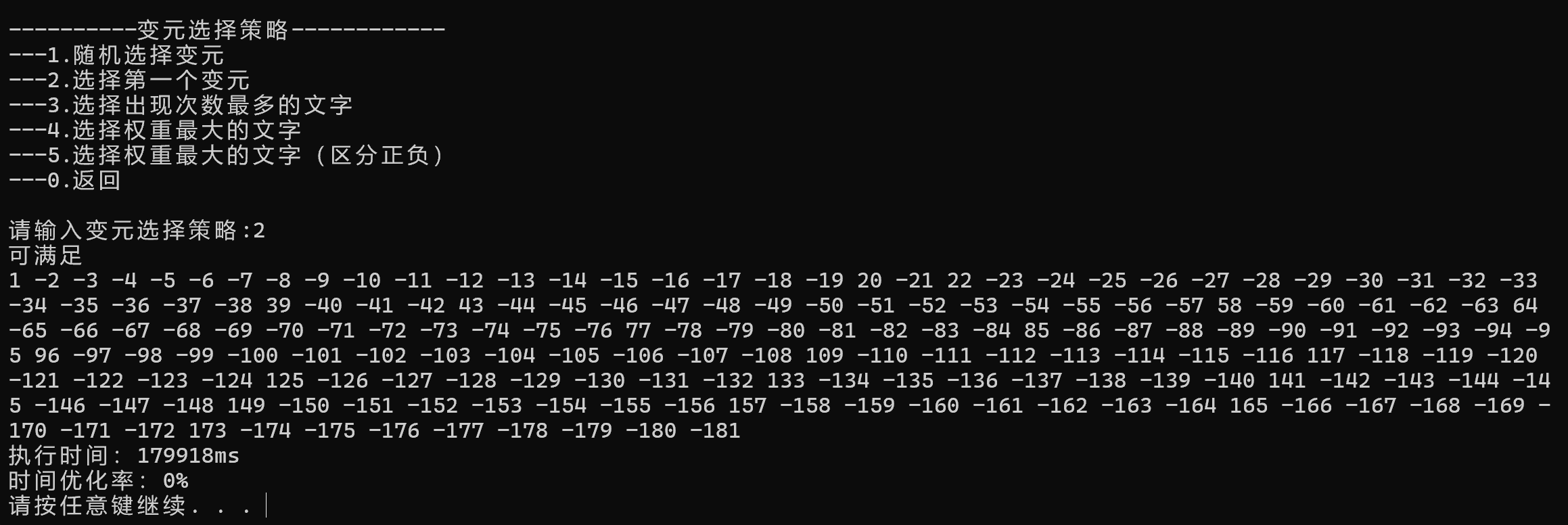


图4-5 性能测试算例1优化前的运行结果

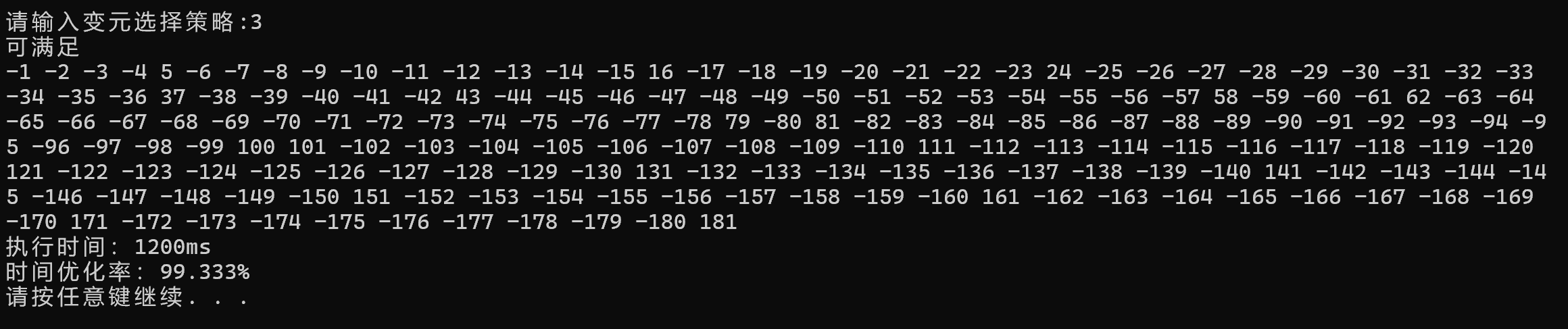


图4-6 性能测试算例1优化后的运行结果

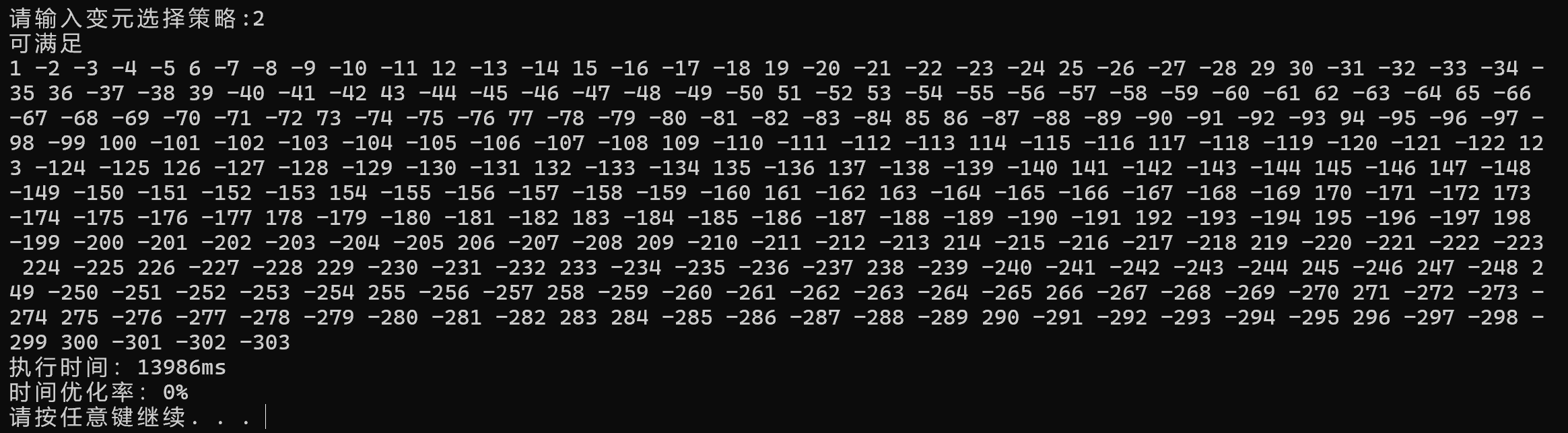


图4-7 性能测试算例2优化前的运行结果

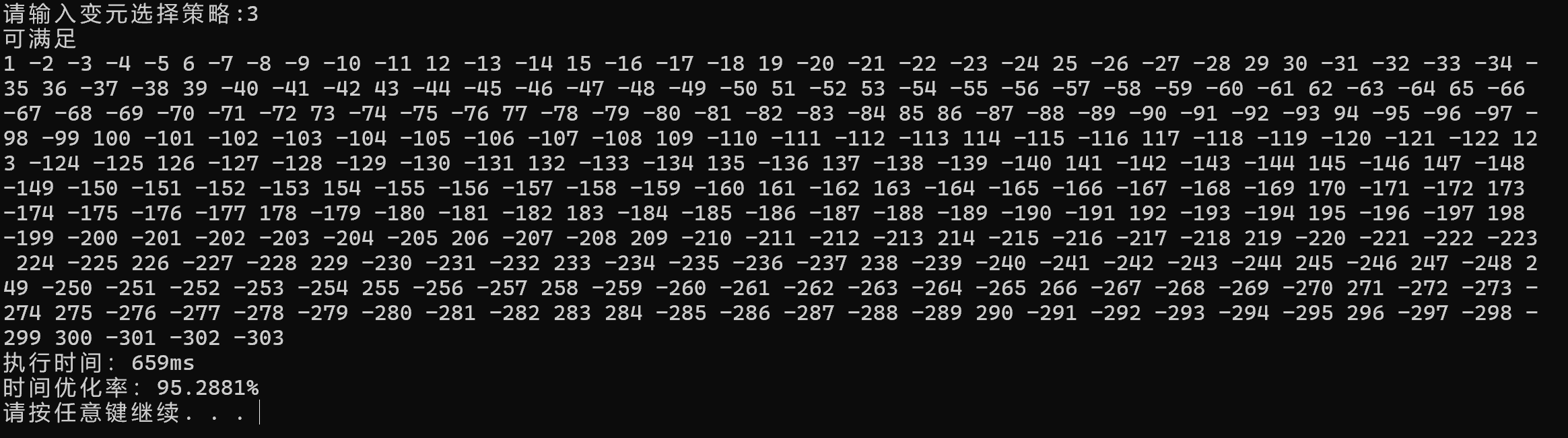


图4-8性能测试算例2优化后的运行结果

性能测试的效果良好，优化后。的决策选择函数大幅度提升了算法速度，两个算例的优化率均超过95%。

**4.1.3 用户交互测试**

经测试，系统能正常接收用户参数进行页面切换或相应功能调用，并对用户可能出现的错误输入均进行了相应应对机制。

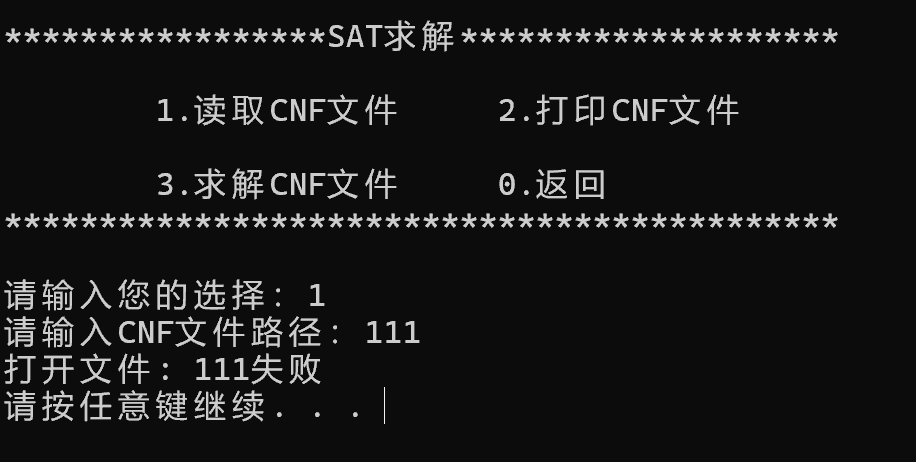


图4-9读取文件的用户交互测试

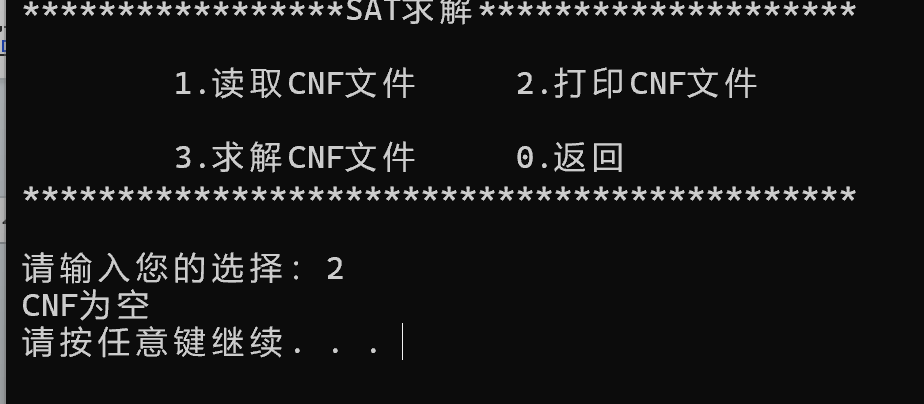


图4-10打开文件的用户交互测试

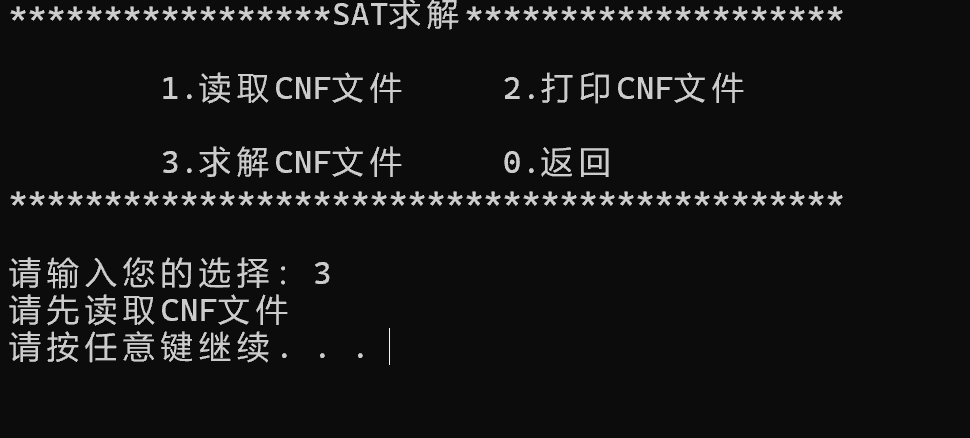


图4-11求解文件的用户交互测试

## 4.2 数独游戏测试

该部分，主要测试程序能否生成符合要求的数独游戏棋盘以及用户能否正常游玩游戏，对程序的交互界面进行测试。

**4.2.1 数独生成测试**

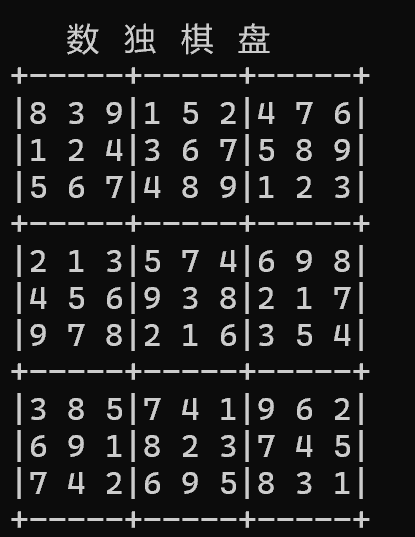
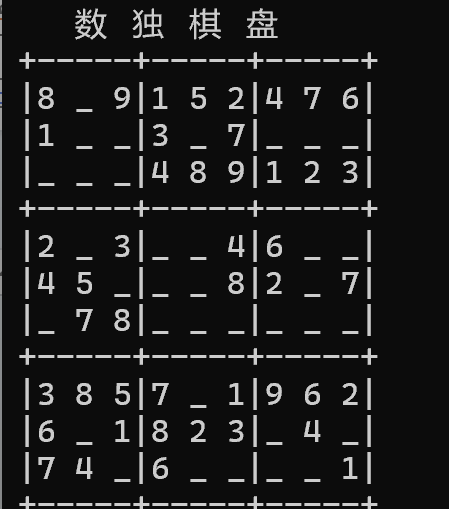
****

图4-12 数独生成测试1

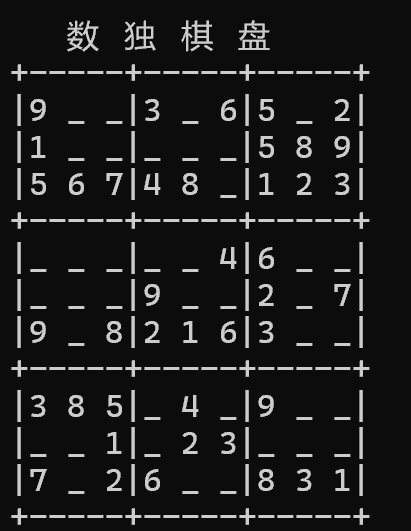


图4-13 数独生成测试2

经验证，经挖洞法自动生成的棋盘符合对角线数独的游戏要求，且能提供良好的视觉效果。

**4.2.2 数独交互测试**

1. 填入操作
2. 填入的位置已有数字

输入 1 1 1 4

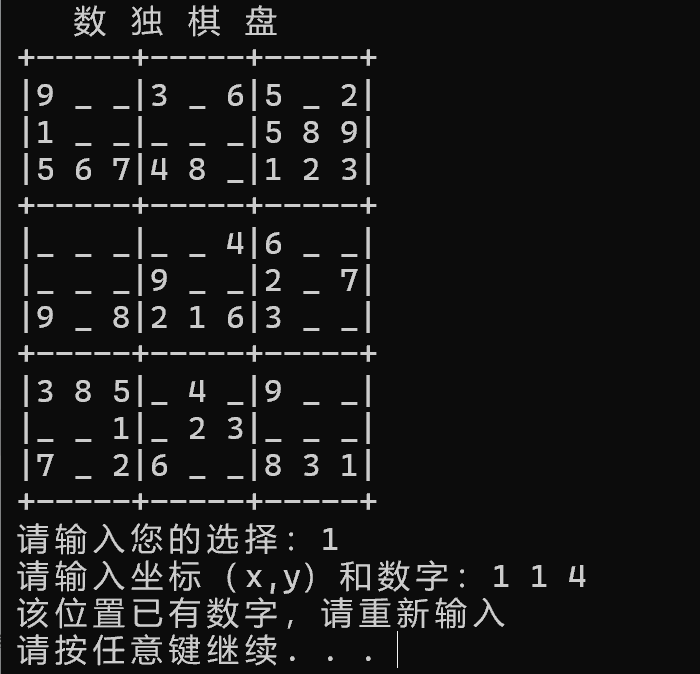


图4-14 数独填入操作测试1

1. 填入数字将违反规则

输入 1 1 2 3

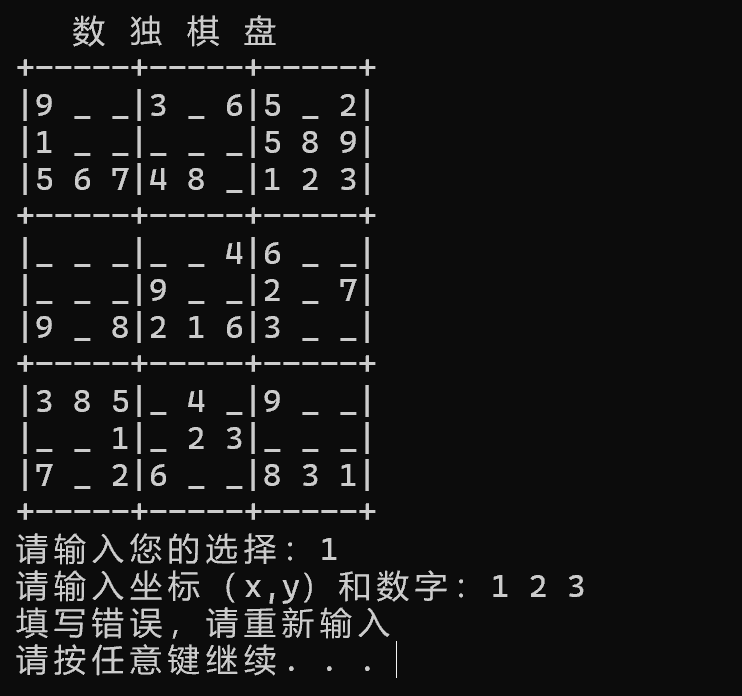


图4-15数独填入操作测试2

1. 正常填入

输入 1 1 24



图4-16数独填入操作测试3

经测试，各类操作均在程序考虑范围之类，用户拥有良好的交互体验

1. 删除操作

如图，在(1,1)位置填入4后，输入1 1 1 0删除该位置数字



图4-17数独删除操作测试

1. 重置操作

如图，在已经输入若干数的情况下输入3可恢复棋盘到初始化状态。

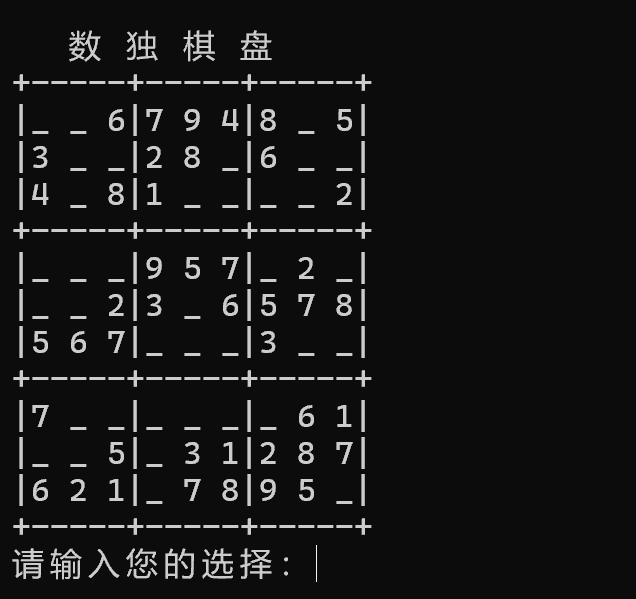
****

图4-18数独重置操作测试

## 4.3 数独游戏归约为SAT问题求解

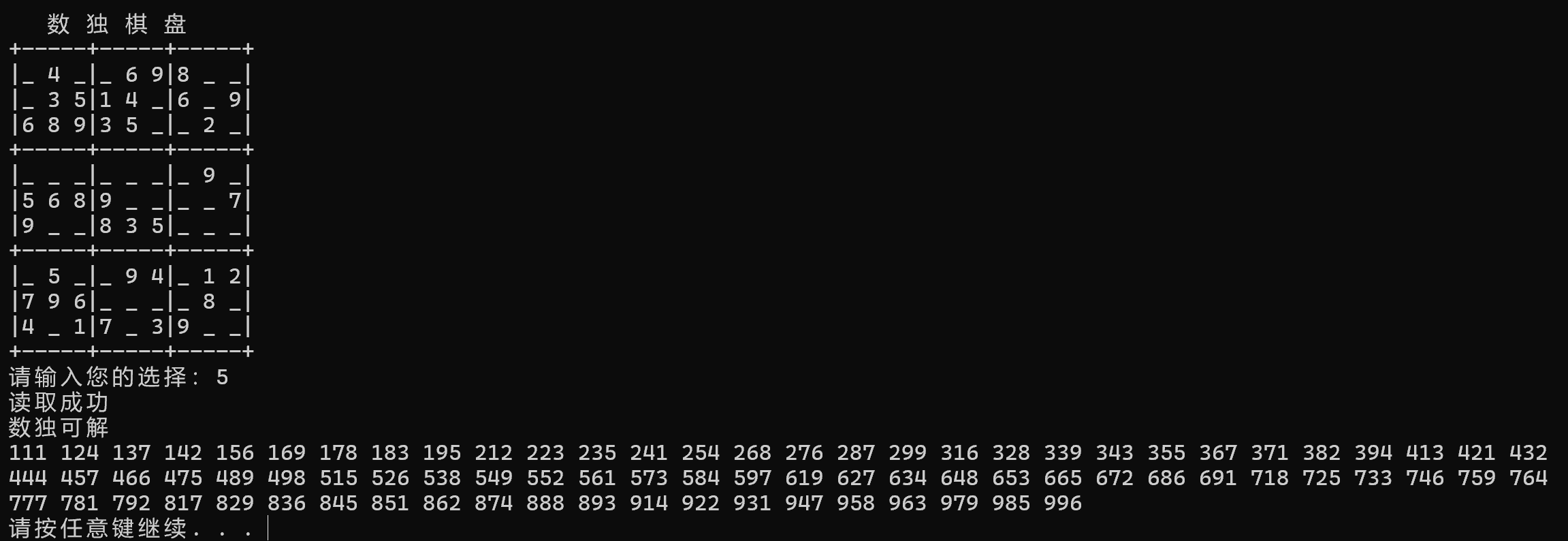
****

图4-19数独DPLL求解



如图，随机生成的数独经转换为cnf文件后调用SAT求解器求解，最终计算结果与最初生成的数独一致，可见程序满足最初的设计目标。

图4-20数独DPLL求解标准答案

**5 总结与体会**

**5.1全文总结**

SAT问题作为经典的NP难问题，具有重要的理论研究和实用价值，学习和研究SAT问题的求解对我们具有重要意义。经过两周的学习研究，本文实现了基于SAT（布尔可满足性问题）求解的对角线数独求解程序的设计与实现过程。通过深入学习SAT问题的理论和背景知识，理解DPLL算法并提出优化策略，将数独问题归约为SAT问题，本文实现了一个能够解决对角线数独问题的求解器。尽管本文只实现了一个初级SAT求解器，但也展示了SAT理论与算法在实际问题中的应用，也为自己积累了解决复杂问题的实践经验。

本文的主要工作包括：

（1）学习SAT问题的理论背景，了解其概念

（2）学习DPLL算法的应用，提出优化算法

（3）设计数独到SAT的归约算法，并运用SAT求解器进行对角线数独问题求解。

**5.2个人体会**

作为一名计算机专业大二学生，在此之前的编程经验仅限于基本编程模型与技术的验证性训练，这次课设对我来说毫无疑问是个挑战。第一次来实验课的时候，我发现同学们好像都早已知道课设的题目，有的人已经做了部分准备甚至已经完成了项目（可能由于我是转专业后分流到计科1班的，并没有加入实验群，收到相关文件），当我看到课设的两个设计问题：SAT问题，抽象语法树的时候，简直是一头雾水，简单了解后，在学长的建议下我还是选择了SAT问题。然后第一节实验课花了整整三个小时才基本理解SAT问题和DPLL的概念，项目还没开始建。在不断学习了解下，浏览相关的技术博客和论文之后，最终在第六节实验课时完成了项目并通过了检查。不过在写此次课设报告仍然发现了数个bug，这说明我之前对程序的测试仍有不足。总的来说，本次课设让我遇到了很多此前实验课没遇到的问题，也积攒了设计程序，组织代码的经验。我们的程序不仅仅需要解决问题，还需要良好的使用体验和可维护性，可检验性，这是在大一的实验课很少体会到的。最后，这里简单列出一些本次课设新遇到的问题，新积累的经验：

1. **LNK链接错误：**在平常的编程的练习中，我们大多遇到的都是些语法错误，但在本次课设中，由于文件，变量名繁多，头文件包含混乱，我遇到了多次编译链接错误。有些是函数名声明和定义不符，有些是编译lib库和引用lib库选项不一致，有些是头文件的重复引用……造成了大量“无法解析的外部命令”、“已经在.obj中定义”等报错。经过反复debug，相关解决方案的学习，我积累了许多开发、组织项目的经验。
2. **变量的反复使用：**为减少变量的使用，项目中部分变量是可以反复利用的，然而由于忘记在重复使用前进行memset操作，导致经常出现莫名其妙的错误，由于习惯性先检查功能函数，容易忽视变量原因。
3. **交互界面设计：**数独的交互和界面设计也是一个比较复杂的部分，首先数独游戏的棋盘就存在三类，完整棋盘，挖洞后生成的初始棋盘，和用户正在游玩的棋盘，在编写程序时很容易混淆。此外，还要考虑用户的各种错误操作，实时判断当前用户想要填入的位置，数字是否合法。只有自己反复体验，测试，才能发现这些错误。
4. **博客撰写：**本次课设参照学习了很多学长博客中留下的经验，因此，本人也尝试将本次课设学习过程记录到博客中，目前有90的浏览量(大概都是同学)。网址如下：[Losyi - 博客园 (cnblogs.com)](https://www.cnblogs.com/losyi)

# 参考文献

[1]杨晗. 求解SAT问题中分支策略与删除策略的研究[D].西南交通大学,2020.DOI:10.27414/d.cnki.gxnju.2019.001679.

[2] 陈稳. 基于DPLL的SAT算法的研究及应用[D].电子科技大学,2011.

[3]肖凡. SAT及其相关问题的精确算法研究[D].华中科技大学,2020.DOI:10.27157/d.cnki.ghzku.2019.000280.

[4] 维基百科：SAT [SAT | encyclopedia article by TheFreeDictionary](https://encyclopedia.thefreedictionary.com/SAT)

[5] 美剧之神.DPLL算法求解CNF SAT 与 数独求解程序(C++ 实现).CSDN.检索2022.12,https://blog.csdn.net/M1170780140/article/details/128053901

[6]建鼎呓语.随机数独局面的生成算法.CSDN.检索2015.8,https://blog.csdn.net/nibiewuxuanze/article/details/47679927?spm=1001.2014.3001.5506

[7] 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

[8] 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A main.cpp**

1. #include"DPLL-Solver.h"
2. #include"Literal.h"
3. #include"Clause.h"
4. #include"CNF.h"
5. #include"CNFParser.h"
6. #include"vector.hpp"
7. #include"definition.h"
8. #include"GenerateXSudoku.h"
9. #include"CheckSudokuBoard.hpp"
10. bool SudokuToCnf(const vector<int>& board, int empty);*//将数独转化为CNF*
11. bool XSudokuToCnf(const vector<int>& board, int empty);*//将对角线数独转化为CNF*
12. int Count = 0;*//递归次数*
13. int numVars = 0, numClauses = 0;*//变元数和子句数*
14. int main() {
15. int choice = 1, choice2 = 1, choice3 = 1, choice4 = 1, way = 1;*//选择*
16. int isRead = 0;*//是否读取CNF文件*
17. std::string filename;*//文件名*
18. CNF Cnf; *//CNF*
19. clock\_t start, end;*//计时*
20. double time1 = 0, time2 = 0;*//时间比较*
21. int\* assignments = nullptr;*//布尔变元的赋值状态*
22. Sodoku\* sodoku = new Sodoku();
23. vector<int> NormalBoard(81, 0);*//数独完整棋盘*
24. vector<int> GameBoard(81, 0);*//挖洞生成的数独游戏棋盘*
25. vector<int> GamePlayBoard(81, 0);*//玩家填写的数独游戏棋盘*
26. int empty = 0;*//挖洞数目*
27. int x = 0, y = 0, num = 0;*//玩家输入的坐标和数字*
28. srand(static\_cast<unsigned int>(time(nullptr)));*//随机数种子*
29. while (true) {
30. cout << endl << endl;
31. cout << "\t  菜  单" << endl;
32. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
33. cout << endl;
34. cout << "1.SAT求解器\t  2.数独游戏"<< endl;
35. cout << endl;
36. cout << "0. Exit" << endl;
37. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
38. cout << "请输入您的选择：";
39. cin >> choice;
40. system("cls");
41. switch (choice) {
42. case 1:
43. choice2 = 1;
44. while (choice2)
45. {
46. way = 1;
47. cout << endl;
48. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SAT求解\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;
49. cout << "\t1.读取CNF文件     2.打印CNF文件" << endl<<endl;
50. cout << "\t3.求解CNF文件     0.返回" << endl;
51. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl<<endl;
52. cout << "请输入您的选择：";
53. cin >> choice2;
54. switch (choice2)
55. {
56. case 1:
57. cout << "请输入CNF文件路径：";
58. cin >> filename;
59. if (CNFParser::parse(Cnf, filename, numVars, numClauses))
60. {
61. isRead = 1;
62. if (assignments != nullptr) delete[] assignments;
63. assignments = new int[numVars + 1];
65. cout << "读取成功" << endl;
66. }
67. *//按任意键继续*
68. system("pause");
69. system("cls");
70. break;
71. case 2:
72. if (Cnf.isEmpty()) cout << "CNF为空" << endl;
73. else Cnf.print();
74. *//按任意键继续*
75. system("pause");
76. system("cls");
77. break;
78. case 3:
79. while (isRead&&way) {
80. Count = 0;
81. cout << endl;
82. cout << "----------变元选择策略------------" << endl;
83. cout << "---1.随机选择变元" << endl;
84. cout << "---2.选择第一个变元" << endl;
85. cout << "---3.选择出现次数最多的文字" << endl;
86. cout << "---4.选择权重最大的文字" << endl;
87. cout << "---5.选择权重最大的文字（区分正负）" << endl;
88. cout << "---0.返回" << endl;
89. cout << endl;
90. cout << "请输入变元选择策略:";
91. cin >> way;
92. if (way == 0) { system("cls"); break; }
93. *//计算执行时间*
94. start = clock();
95. bool result = DPLL(Cnf, way, assignments);
96. if (result)
97. {
98. cout << "可满足" << endl;
99. printAssignments(assignments,1);
100. }
101. else cout << "不可满足" << endl;
102. end = clock();
104. if (way == 2) time1 = (double)(end - start);
105. time2 = (double)(end - start);
106. saveResultToFile(filename, result, assignments, time2);
107. cout << "执行时间：" << time2 << "ms" << endl;
108. if (time1 == 0 || time1 == time2 || time2 == 0) cout << "时间优化率：0%" << endl;
109. else cout << "时间优化率：" << (time1 - time2) / time1 \* 100 << "%" << endl;
110. system("pause");
111. system("cls");
112. }
113. if (isRead == 0)
114. {
115. cout << "请先读取CNF文件" << endl;
116. system("pause");
117. system("cls");
118. }
119. break;
120. case 0:
121. system("cls");
123. break;
124. default:
125. break;
126. }
127. }
128. break;
129. case 2:
130. choice3 = 1;
131. while (choice3)
132. {
134. cout << endl;
135. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*数独游戏\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;
136. cout << "\t1.普通数独     2.对角线数独" << endl << endl;;
137. cout << "\t0.返回" << endl;
138. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;;
139. cout << "请输入您的选择：";
140. cin >> choice3;
141. switch (choice3)
142. {
143. case 1:
144. system("cls");
145. StartSodokuGenerate(sodoku, NormalBoard);
146. empty=generateGameBoard(NormalBoard, GameBoard);
147. GamePlayBoard = GameBoard;
148. choice4 = 1;
149. while (choice4)
150. {
151. system("cls");
152. cout << endl;
153. cout << "-----------------数独游戏-----------------" << endl << endl;
154. cout << "\t1.填写数字     2.查看答案" << endl << endl;
155. cout << "\t3.重新开始     4.清屏" << endl << endl;
156. cout << "\t5.DPLL求解     0.返回" << endl;
157. cout << "-------------------------------------------" << endl << endl;
158. PrintBoard(GamePlayBoard);
159. cout << "请输入您的选择：";
160. cin >> choice4;
161. switch (choice4)
162. {
163. case 1:
164. cout << "请输入坐标（x,y）和数字：";
165. cin >> x >> y >> num;
166. if (x < 1 || x>9 || y < 1 || y>9 || num < 0 || num>9)
167. {
168. cout << "输入错误，请重新输入" << endl;
169. system("pause");
170. break;
171. }
173. if (GameBoard[(x - 1) \* 9 + y - 1] != 0)
174. {
175. cout << "该位置不可填写，请重新输入" << endl;
176. system("pause");
177. break;
178. }
179. *//检测当前局面是否合法*
180. if (CheckBoard(GamePlayBoard, x, y, num) == false)
181. {
182. cout << "填写错误，请重新输入" << endl;
183. system("pause");
184. break;
185. }
186. GamePlayBoard[(x - 1) \* 9 + y - 1] = num;
187. cout << endl;
188. PrintBoard(GamePlayBoard);
189. break;
190. case 2:
191. cout << endl;
192. PrintBoard(NormalBoard);
193. system("pause");
194. break;
195. case 3:
196. GamePlayBoard = GameBoard;
197. system("cls");
198. PrintBoard(GamePlayBoard);
199. break;
200. case 4:
201. system("cls");
202. break;
203. case 5:
204. SudokuToCnf(GamePlayBoard,empty);
205. if (CNFParser::parse(Cnf, "NormalSudoku.cnf", numVars, numClauses))
206. {
207. isRead = 1;
208. numVars = 999;
209. if (assignments != nullptr) delete[] assignments;
210. assignments = new int[numVars + 1];
211. cout << "读取成功" << endl;
212. }
213. if (DPLL(Cnf, 2, assignments))
214. {
215. cout << "数独可解" << endl;
216. printAssignments(assignments,0);
217. }
218. else cout << "数独不可解" << endl;
219. system("pause");
220. break;
221. case 0:
222. system("cls");
223. break;
224. default:
225. break;
226. }
227. }
228. system("pause");
229. break;
230. case 2:
231. system("cls");
232. generateDiagonalSudoku(NormalBoard);
233. empty = generateGameBoard(NormalBoard, GameBoard);
234. GamePlayBoard = GameBoard;
236. choice4 = 1;
237. while (choice4)
238. {
239. system("cls");
240. cout << endl;
241. cout << "-----------------数独游戏-----------------" << endl << endl;
242. cout << "\t1.填写数字     2.查看答案" << endl << endl;
243. cout << "\t3.重新开始     4.清屏" << endl << endl;
244. cout << "\t5.DPLL求解     0.返回" << endl;
245. cout << "-------------------------------------------" << endl << endl;
246. PrintBoard(GamePlayBoard);
247. cout << "请输入您的选择：";
248. cin >> choice4;
249. switch (choice4)
250. {
251. case 1:
252. cout << "请输入坐标（x,y）和数字：";
253. cin >> x >> y >> num;
254. if (x < 1 || x>9 || y < 1 || y>9 || num < 1 || num>9)
255. {
256. cout << "输入错误，请重新输入" << endl;
257. system("pause");
258. break;
259. }
261. if (GameBoard[(x - 1) \* 9 + y - 1] != 0)
262. {
263. cout << "该位置不可填写，请重新输入" << endl;
264. system("pause");
265. break;
266. }
267. *//检测当前局面是否合法*
268. if (CheckBoard(GamePlayBoard, x, y, num) == false)
269. {
270. cout << "填写错误，请重新输入" << endl;
271. system("pause");
272. break;
273. }
274. GamePlayBoard[(x - 1) \* 9 + y - 1] = num;
275. cout << endl;
276. PrintBoard(GamePlayBoard);
277. break;
278. case 2:
279. cout << endl;
280. PrintBoard(NormalBoard);
281. system("pause");
282. break;
283. case 3:
284. GamePlayBoard = GameBoard;
285. system("cls");
286. PrintBoard(GamePlayBoard);
287. break;
288. case 4:
289. system("cls");
290. break;
291. case 5:
292. XSudokuToCnf(GamePlayBoard, empty);
293. if (CNFParser::parse(Cnf, "XSudoku.cnf", numVars, numClauses))
294. {
295. isRead = 1;
296. numVars = 999;
297. if (assignments != nullptr) delete[] assignments;
298. assignments = new int[numVars + 1];
299. cout << "读取成功" << endl;
300. }
301. if (DPLL(Cnf, 2, assignments))
302. {
303. cout << "数独可解" << endl;
304. printAssignments(assignments,0);
305. }
306. else cout << "数独不可解" << endl;
307. system("pause");
308. break;
309. case 0:
310. system("cls");
311. break;
312. default:
313. break;
314. }
315. }
316. system("pause");
317. break;
318. case 0:
319. system("cls");
320. break;
321. default:
322. break;
323. }
325. }
326. break;
327. case 0:
328. cout << "退出程序" << endl;
329. return 0;
330. default:
331. cout << "无效的选择，请重新尝试\n";
332. }
333. }
334. return 0;
335. }

**附录B DPLL-Solver.cpp**

1. #include "DPLL-Solver.h"
2. #include <chrono>
3. extern int  numVars;*//变元个数，在main.cpp中定义*
4. extern int Count;*//调试用，检测递归次数*
5. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
6. \*函数名称：propagate
7. \* 函数功能：根据给定的文字（literal）对 CNF 公式进行传播，更新 CNF 公式和赋值数组。
8. \* 注释：- 遍历 CNF 公式中的所有子句。
9. - 检查子句是否已被给定文字（literal）满足。
10. - 如果子句已被满足，则忽略该子句。
11. - 如果子句未被满足，删除子句中与给定文字相反的文字。
12. - 如果删除文字后子句为空，则返回 False，表示冲突。
13. - 如果子句非空，将其添加到新的 CNF 公式中。
14. - 更新赋值数组以记录文字的赋值。
15. \* 返回值：bool 类型，如果传播成功，则返回 True，若出现空子句，则返回 False。
16. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
17. bool propagate(CNF& cnf, const Literal& literal,int \*assignments) {
18. CNF newCNF;
19. bool hasChanged = false;
20. for (int i = 0; i < cnf.clauses.size(); ++i) {
21. const Clause& clause = cnf.clauses[i];*//取出一个子句*
22. *//检查字句是否包含文字*
23. bool satisfied = false;
24. for (int j = 0; j < clause.literals.size(); ++j) {
25. if (clause.literals[j] == literal) *//文字相等*
26. {
27. satisfied = true;
28. break;
29. }
30. }
31. if (satisfied) {
32. *//删除已满足的字句，即不添加到新CNF*
33. continue;
34. }
35. *//删除子句中的~L文字*
36. Clause newClause = clause;
37. Literal negLiteral(-literal.var);
38. newClause.removeLiteral(negLiteral);
39. *//如果删除~L后子句不为空，添加到新CNF*
40. if (!newClause.literals.empty()) {
41. newCNF.addClause(newClause);
42. }
43. else {
44. *// 如果出现空子句，返回false*
45. return false;
46. }
47. hasChanged = true;*//标记已经改变*
48. }
49. assignments[abs(literal.var)] = (literal.var > 0) ? 1 : -1; *//记录赋值情况*
50. cnf = newCNF;*//更新CNF*
51. return true;
52. }
53. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
54. \*函数名称：selectLiteral
55. \* 函数功能：根据指定的策略选择一个文字。
56. \* 注释：- 策略1：选择第一个子句的第一个文字。
57. - 策略2：随机选择一个非空子句中的一个文字。
58. - 策略3：选择出现次数最多的文字。
59. - 策略4：选择权重最大的文字（不考虑正负）。
60. - 策略5：选择权重最大的文字（考虑正负）。
61. \* 返回值：Literal 类型，返回选择的文字。
62. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
63. Literal selectLiteral(const CNF& cnf,int way) {
64. *// 策略1：选择第一个子句的第一个文字*
66. Count++;
67. if (way == 2)
68. {
69. if (!cnf.clauses.empty() && !cnf.clauses[0].literals.empty()) *//如果子句不为空*
70. {
71. return cnf.clauses[0].literals[0];
72. }
73. }
74. *// 策略2：随机选取一个文字*
75. else if (way == 1)
76. {
78. *// 找到一个非空子句*
79. int clauseIndex = rand() % cnf.clauses.size();
80. while (cnf.clauses[clauseIndex].literals.empty()) {
81. clauseIndex = rand() % cnf.clauses.size();
82. }
83. *// 选择子句中的一个文字*
84. int literalIndex = rand() % cnf.clauses[clauseIndex].literals.size();
85. return cnf.clauses[clauseIndex].literals[literalIndex];
86. }
87. *// 策略3：选择出现次数最多的文字*
88. else if (way == 3)
89. {
90. *// 初始化计数数组*
91. vector<int> cnt(numVars \* 2 + 1, 0);
92. *// 计算每个变元的出现次数*
93. for (const Clause& clause : cnf.clauses) {
94. for (const Literal& literal : clause.literals) {
95. if (literal.var > 0) {
96. cnt[literal.var]++;
97. }
98. else {
99. cnt[numVars - literal.var]++;
100. }
101. }
102. }
103. *// 选择出现次数最多的变元（正面）*
104. int maxBool = 0;
105. int maxTimes = 0;
106. for (int i = 1; i <= numVars; ++i) {
107. if (cnt[i] > maxTimes) {
108. maxTimes = cnt[i];
109. maxBool = i;
110. }
111. }
112. *// 如果正面变元的出现次数都是 1，则选择反面变元*
113. if (maxTimes == 1) {
114. for (int i = numVars + 1; i <= numVars \* 2; ++i) {
115. if (cnt[i] > maxTimes) {
116. maxTimes = cnt[i];
117. maxBool = numVars - i;
118. }
119. }
120. }
121. return Literal(maxBool);
122. }
123. *//策略4：选择权重最大的文字（不考虑正负）*
124. else if (way == 4)
125. {
126. *// 使用 vector 替代裸指针*
127. vector<double> weight(numVars + 1, 0.0);
128. *// 计算每个变元的权重*
129. for (const Clause& clause : cnf.clauses) {
130. double clauseWeight = pow(2.0, -static\_cast<double>(clause.literals.size()));
131. for (const Literal& literal : clause.literals) {
132. if (literal.var != 0) {
133. int varIndex = abs(literal.var);
134. if (varIndex <= numVars) {
135. weight[varIndex] += clauseWeight;
136. }
137. }
138. }
139. }
140. *// 找到最大权重的变元*
141. double maxWeight = 0.0;
142. int maxBool = 0;
143. for (int i = 1; i <= numVars; ++i) {
144. if (weight[i] > maxWeight) {
145. maxWeight = weight[i];
146. maxBool = i;
147. }
148. }
149. return Literal(maxBool);
150. }
151. *//策略5：选择权重最大的文字（考虑正负）*
152. else
153. {
154. *// 使用 std::vector 替代裸指针*
155. vector<double> weight(numVars \* 2 + 1, 0.0);
156. *// 计算每个变元的权重*
157. for (const Clause& clause : cnf.clauses) {
158. double clauseWeight = pow(2.0, -static\_cast<double>(clause.literals.size()));
159. for (const Literal& literal : clause.literals) {
160. int varIndex = std::abs(literal.var);
161. if (literal.var > 0) {
162. weight[varIndex] += clauseWeight;
163. }
164. else {
165. weight[numVars - literal.var] += clauseWeight;
166. }
167. }
168. }
169. *// 找到最大权重的变元*
170. double maxWeight = 0.0;
171. int maxBool = 0;
172. for (int i = 1; i <= numVars; ++i) {
173. double totalWeight = weight[i] + weight[i + numVars];
174. if (totalWeight > maxWeight) {
175. maxWeight = totalWeight;
176. maxBool = i;
177. }
178. }
179. *// 确定最终变元是正面还是反面*
180. if (weight[maxBool] < weight[maxBool + numVars]) {
181. maxBool = -maxBool;
182. }
183. return Literal(maxBool);
184. }
185. return Literal(); *// 返回空文字*
187. }
188. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
189. \*函数名称：DPLL
190. \* 函数功能：使用 DPLL 算法对给定的 CNF 公式进行求解。
191. \* 注释：- 不断查找单子句，并进行传播。
192. - 如果找到单子句且传播成功，则继续处理。
193. - 如果 CNF 公式为空，说明所有子句都满足，返回 True。
194. - 否则，选择一个文字进行分支搜索。
195. - 尝试给文字赋值为真或假，并递归地调用 DPLL 函数。
196. - 如果任一分支成功，则返回 True；否则，返回 False。
197. \* 返回值：bool 类型，如果 CNF 公式可满足，则返回 True，否则返回 False。
198. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
199. bool DPLL(CNF& cnf,int way, int\* assignments) {
200. while (true) {
201. bool unitClauseFound = false;
202. for (int i = 0; i < cnf.clauses.size(); ++i) {
203. const Clause& clause = cnf.clauses[i];
204. if (clause.isUnit()) {
205. *//如果是单子句，传播*
206. Literal unitLiteral = clause.literals[0];*//取出单子句的文字*
207. if (!propagate(cnf, unitLiteral, assignments)) {
208. return false;*//如果传播失败，返回false*
209. }
211. unitClauseFound = true;
212. break;
213. }
214. }
215. if (!unitClauseFound) {*//没有单子句，跳出循环*
216. break;
217. }
218. }
219. if (cnf.isEmpty()) {
220. return true; *// 所有子句都满足,返回true*
221. }
222. *//如果cnf中还有子句，选择文字*
223. if (cnf.hasNext())
224. {
225. *// 选择文字来搜索*
226. Literal literal = selectLiteral(cnf,way);
227. if (literal.var == 0) {
228. return false; *//没有可供选择的文字*
229. }
230. *// 尝试文字的两种赋值*
231. CNF cnfTrue = cnf;
232. Literal posLiteral(literal.var);
233. if (propagate(cnfTrue, posLiteral, assignments)) {
234. if (DPLL(cnfTrue,way, assignments)) {
235. return true;
236. }
237. }
238. CNF cnfFalse = cnf;
239. Literal negLiteral(-literal.var);
240. if (propagate(cnfFalse, negLiteral, assignments)) {
241. if (DPLL(cnfFalse,way, assignments)) {
242. return true;
243. }
244. }
245. }
246. return false; *// 两种分支都失败*
247. }
249. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
250. \*函数名称：printAssignments
251. \* 函数功能：打印给定赋值数组中的变量赋值。
252. \* 注释：-遍历赋值数组，打印赋值为 1 的变量。
253. - 如果 choice 为 True，且赋值为 -1，则也打印负变量。
254. \* 返回值：void
255. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
256. void printAssignments(const int\* assignments,int choice) {
257. for (int i = 1; i <= numVars; ++i) {
258. if (assignments[i] == 1) {
259. std::cout << i << " ";
260. }
261. else if (choice&&assignments[i] == -1) {
262. std::cout << -i << " ";
263. }
264. }
265. std::cout << std::endl;
266. }
267. *// 将结果保存到文件中*
268. void saveResultToFile(const std::string& baseFilename, bool result, const int\* assignments,double duration) {
269. string filename = baseFilename;
270. filename.erase(filename.end() - 4,filename.end());
271. filename += ".res";
272. ofstream outFile(filename);
273. if (!outFile) {
274. cerr << "打开文件: " << filename << "失败" << endl;
275. return;
276. }
277. *// 写入求解结果*
278. outFile << "s " << (result ? "1" : "0") << std::endl;
279. *// 写入变量赋值*
280. outFile << "v ";
281. if (result) {
282. for (int i = 1; i <= numVars; ++i) {
283. if (assignments[i] == 1) {
284. outFile << i << " ";
285. }
286. else if (assignments[i] == -1) {
287. outFile << -i << " ";
288. }
289. }
290. outFile << std::endl;
291. }
292. *// 写入执行时间*
293. outFile << "t " << duration << std::endl;
294. outFile.close();
295. }

**附录C GenerateXSudoku.cpp**

1. #include"GenerateXSudoku.h"
2. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
3. \*函数名称：isSafe
4. \* 函数功能：检查在给定的行、列和数字的情况下，数字是否可以安全地放置在数独棋盘上。
5. \* 注释：用于生成数独时，检查数字是否可以放置在数独棋盘上。
6. \* 返回值：如果数字可以安全地放置在数独棋盘上，则返回true，否则返回false。
7. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
8. bool isSafe(const vector<int>& board, int row, int col, int num) {
9. for (int x = 0; x < SIZE; x++) {
10. *// 检查行*
11. if (board[row \* SIZE + x] == num)
12. return false;
13. *// 检查列*
14. if (board[x \* SIZE + col] == num)
15. return false;
16. }
17. *// 检查九宫格*
18. int startRow = row - row % 3;
19. int startCol = col - col % 3;
20. for (int i = 0; i < 3; i++) {
21. for (int j = 0; j < 3; j++) {
22. if (board[(startRow + i) \* SIZE + (startCol + j)] == num)
23. return false;
24. }
25. }
26. *// 检查主对角线*
27. if (row == col) {
28. for (int i = 0; i < SIZE; i++) {
29. if (board[i \* SIZE + i] == num)
30. return false;
31. }
32. }
33. *// 检查副对角线*
34. if (row + col == SIZE - 1) {
35. for (int i = 0; i < SIZE; i++) {
36. if (board[i \* SIZE + (SIZE - 1 - i)] == num)
37. return false;
38. }
39. }
40. return true;
41. }
42. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
43. \*函数名称：solveSudoku
44. \* 函数功能：随机生成数独棋盘
45. \* 注释：利用递归回溯的思想，生成数独棋盘。
46. \* 返回值：bool类型，如果成功生成数独棋盘，则返回true，否则返回false。
47. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
48. bool solveSudoku(vector<int>& board)
49. {
50. for (int row = 0; row < SIZE; row++) {
51. for (int col = 0; col < SIZE; col++) {
52. if (board[row \* SIZE + col] == 0) { *//如果当前位置为空*
53. for (int num = 1; num <= SIZE; num++) {
54. if (isSafe(board, row, col, num)) {
55. board[row \* SIZE + col] = num; *// 放置数字*
56. if (solveSudoku(board))
57. return true; *// 递归*
58. *// Backtrack*
59. board[row \* SIZE + col] = 0; *//回溯*
60. }
61. }
62. return false; *// 递归失败*
63. }
64. }
65. }
66. return true; *// 生成成功*
67. }
68. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
69. \*函数名称：GenerateFirstLine
70. \* 函数功能：随机产生数独棋盘的第一行
71. \* 注释：solveSudoku函数的辅助函数，用于生成数独棋盘的第一行，这决定了后续数独棋盘的生成。
72. \* 返回值：void
73. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
74. void GenerateFirstLine(vector<int>& a) {
75. for (int i = 0; i < SIZE; ++i) {
76. a[i] = rand() % SIZE + 1;
77. int j = 0;
78. while (j < i) {
79. if (a[i] == a[j]) {*//如果生成的数字重复*
80. a[i] = rand() % SIZE + 1;*//重新生成*
81. j = 0;*//重新检查*
82. }
83. else {
84. j++;
85. }
86. }
87. }
88. }
89. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
90. \*函数名称：generateDiagonalSudoku
91. \* 函数功能：随机生成对角线数独棋盘
92. \* 注释：传入board数组，生成对角线数独棋盘。
93. \* 返回值：void
94. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
95. void generateDiagonalSudoku(vector<int>& board) {
97. GenerateFirstLine(board);
98. solveSudoku(board);
100. }

**附录D SudokuToCnf.cpp**

1. #pragma once
2. #include"definition.h"
3. #include"vector.hpp"
4. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
5. \*函数名称：sudokuToCnf
6. \* 函数功能：：将数独棋盘转换为 CNF 格式，并将其写入文件。
7. \* 注释：- 从源文件 "SudokuBase.cnf" 复制基础约束内容。
8. - 添加数独棋盘上已有数字的约束。
9. - 写入主对角线和副对角线不能重复的规则。
10. - 写入 CNF 文件头。
11. \* 返回值：bool类型，如果成功生成 CNF 文件，则返回 true，否则返回 false。
12. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
13. bool SudokuToCnf(const vector<int>& board,int empty)
14. {
15. string sourceFilename = "SudokuBase.cnf";
16. string destinationFilename = "NormalSudoku.cnf";
17. *// 打开源文件进行读取*
18. ifstream sourceFile(sourceFilename, ios::binary);
19. if (!sourceFile) {
20. cerr << "无法打开源文件 " << sourceFilename << endl;
21. return 0;
22. }
23. *// 打开目标文件进行写入，如果文件不存在则新建*
24. ofstream destinationFile(destinationFilename, std::ios::out | std::ios::trunc);
25. if (!destinationFile) {
26. cerr << "无法打开目标文件 " << destinationFilename << endl;
27. return 0;
28. }
29. destinationFile << "p" << " " << "cnf" << " " << 729 << " " << 8829 + 81 - empty << endl;
30. *// 从源文件读取内容并写入到目标文件*
32. for (int i = 1; i <=9; ++i) {
33. for (int j = 1; j <=9; ++j) {
34. if (board[(i- 1) \* 9 + j - 1]) {
35. destinationFile << i << j << board[(i - 1) \* 9 + j - 1] << " 0" << endl;
36. }
37. }
38. }
39. char buffer[1024];
40. while (sourceFile.read(buffer, sizeof(buffer))) {
41. destinationFile.write(buffer, sourceFile.gcount());
42. }
43. *// 写入剩余的内容（如果有的话）*
44. if (sourceFile.gcount() > 0) {
45. destinationFile.write(buffer, sourceFile.gcount());
46. }
47. *// 关闭文件*
48. sourceFile.close();
49. destinationFile.close();
50. *//cout << "文件内容复制完成。" << endl;*
51. }
52. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
53. \*函数名称：XsudokuToCnf
54. \* 函数功能：：将数独棋盘(X数独)转换为 CNF 格式，并将其写入文件。
55. \* 注释：- 从源文件 "SudokuBase.cnf" 复制基础约束内容。
56. - 添加数独棋盘上已有数字的约束。
57. - 写入主对角线和副对角线不能重复的规则。
58. - 写入 CNF 文件头。
59. \* 返回值：bool类型，如果成功生成 CNF 文件，则返回 true，否则返回 false。
60. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
61. bool XSudokuToCnf(const vector<int>& board, int empty)
62. {
63. string sourceFilename = "SudokuBase.cnf";
64. string destinationFilename = "XSudoku.cnf";
65. *// 打开源文件进行读取*
66. ifstream sourceFile(sourceFilename, ios::binary);
67. if (!sourceFile) {
68. cerr << "无法打开源文件 " << sourceFilename << endl;
69. return 0;
70. }
71. *// 打开目标文件进行写入，如果文件不存在则新建*
72. ofstream destinationFile(destinationFilename, std::ios::out | std::ios::trunc);
73. if (!destinationFile) {
74. cerr << "无法打开目标文件 " << destinationFilename << endl;
75. return 0;
76. }
77. *// 写入文件头*
78. destinationFile << "p" << " " << "cnf" << " " << 729 << " " << 8829 +648+81 - empty << endl;
79. *// 主对角线不能重复的规则*
80. int mainDiagonal[9] = { 11, 22, 33, 44, 55, 66, 77, 88, 99 };
81. for (int z = 1; z <= 9; ++z) {
82. for (int i = 0; i < 9; ++i) {
83. for (int j = i + 1; j < 9; ++j) {
84. destinationFile << 0 - (mainDiagonal[i] \* 10 + z) << " " << 0 - (mainDiagonal[j] \* 10 + z) << " " << 0 << endl;
85. }
86. }
87. }
88. *// 副对角线不能重复的规则*
89. int antiDiagonal[9] = { 19, 28, 37, 46, 55, 64, 73, 82, 91 };
90. for (int z = 1; z <= 9; ++z) {
91. for (int i = 0; i < 9; ++i) {
92. for (int j = i + 1; j < 9; ++j) {
93. destinationFile << 0 - (antiDiagonal[i] \* 10 + z) << " " << 0 - (antiDiagonal[j] \* 10 + z) << " " << 0 << endl;
94. }
95. }
96. }
97. *//棋盘上已有数字产生的约束----------------------------------------*
98. for (int i = 1; i <= 9; ++i) {
99. for (int j = 1; j <= 9; ++j) {
100. if (board[(i - 1) \* 9 + j - 1]) {
101. destinationFile << i << j << board[(i - 1) \* 9 + j - 1] << " 0" << endl;
102. }
103. }
104. }
105. *//棋盘上已有数字产生的约束---------------------------------------*
106. *//复制基础约束内容-----------------------------------------------*
107. char buffer[1024];
108. while (sourceFile.read(buffer, sizeof(buffer))) {
109. destinationFile.write(buffer, sourceFile.gcount());
110. }
111. *// 写入剩余的内容（如果有的话）*
112. if (sourceFile.gcount() > 0) {
113. destinationFile.write(buffer, sourceFile.gcount());
114. }
115. *//复制基础约束内容-----------------------------------------------*
116. *// 关闭文件*
117. sourceFile.close();
118. destinationFile.close();
119. }