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：大数据 2302
学　　号：U202315673
姓　　名：夏祺风
指导教师：李贤芝
报告日期：2024 年 9 月 12 日

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 的高效 SAT 求解器的实现，并用提供的算例做相应测试。

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． 将对角线数独游戏问题转化为 SAT 问题，并集成到实现的 SAT 求解器进行数独游戏求解，数独游戏需要具有一定的交互性，使得用户可玩。

5. 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)

6. 将各个模块整合成为项目，并且进行调试直至能够正常运行，并且具有友好完善的异常处理机制。

2 系统需求分析与总体设计

2.1 系统需求分析

设计问题中变元、文字、子句、公式等有效的物理存储结构，基于 DPLL 过程实现一个高效 SAT 求解器，对于给定的中小规模算例进行求解，输出求解结果，统计求解时间。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。

（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。

（3）DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。

（4）时间性能的测量：基于相应的时间处理函数，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）程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略等某 一方面进行优化设计与实现。

（6）SAT 应用：将对角线数独游戏问题转化为 SAT 问题,并集成到上面的求解器进行数独游戏的求解，游戏可玩，具有一定的、简单的交互性。

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，各自模块下面还有一些小的功能，大致介绍如下:

1．基于 DPLL 算法的 SAT 求解器，能够完成如下功能:

（1） CNF 文件的指定与读取解析;

（2） DPLL 求解，计算求解时间并显示，将结果检查输出并且保存到同名.res 文件里。

2．对角线数独游戏，能够完成如下功能:

（1）随机数独游戏的创建（有难度选择）,并可以实现一定程度上的交互;

（2）读入数独游戏文件（txt），转化为 CNF 文件 DPLL 进行求解，再可视化地将结果打印到屏幕上;

系统总体设计流程图如图 2.1 所示:

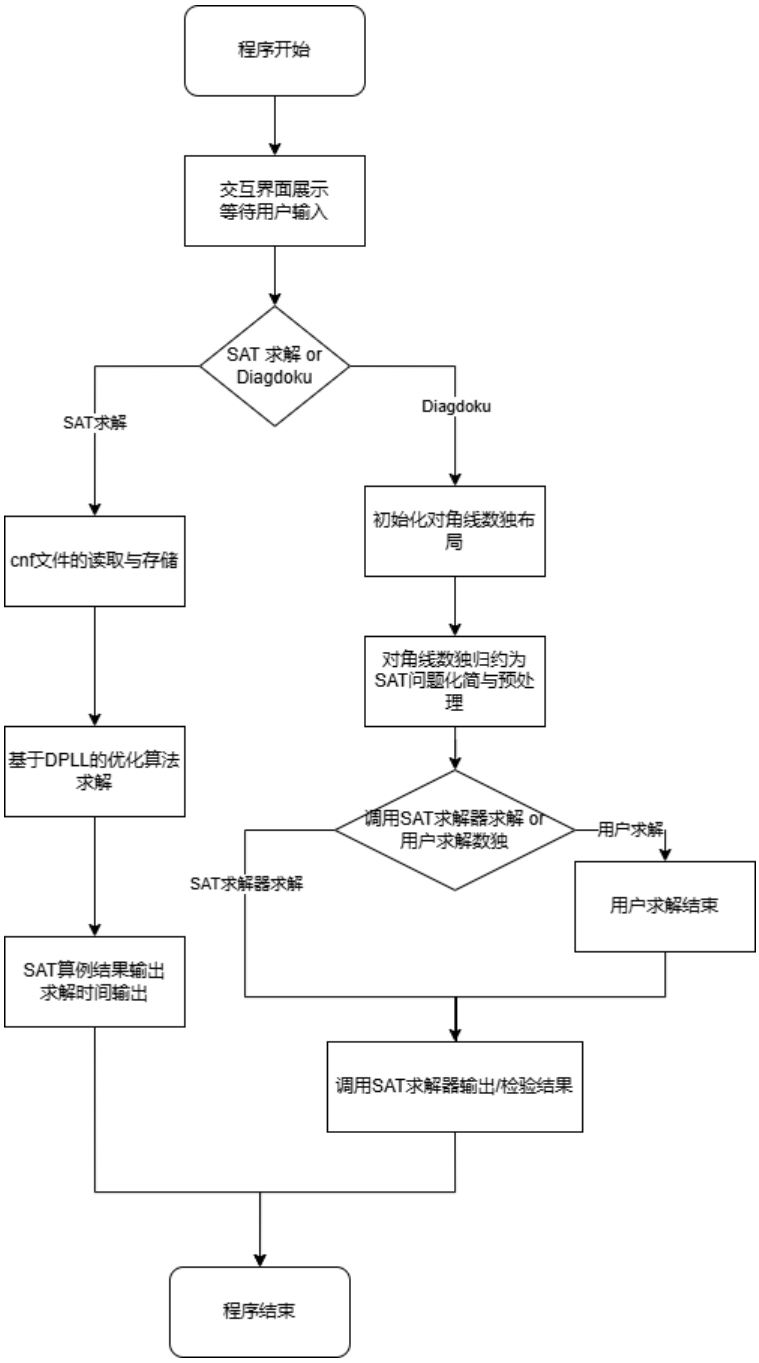

程序开始
交互界面展示
等待用户输入
SAT 求解 or
Diagdoku
SAT求解
Diagdoku
cnf文件的读取与存储
基于DPLL的优化算法
求解
SAT算例结果输出
求解时间输出
初始化对角线数独布
局
对角线数独归约为
SAT问题化简与预处
理
调用SAT求解器求解 or
用户求解数独
SAT求解器求解
用户求解
用户求解结束
调用SAT求解器输出/检验结果
程序结束

图 2-1 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

在 SAT 求解模块中，需要对每个子句和子句包含的文字进行存储。由于不同的 cnf 文件中子句数目及子句中文字的数目并不相同，所以采用类似十字链表的 结构进行存储：以 crossCNF 为起始，crossNode 为节点进行连接。如图 3-1 所示。

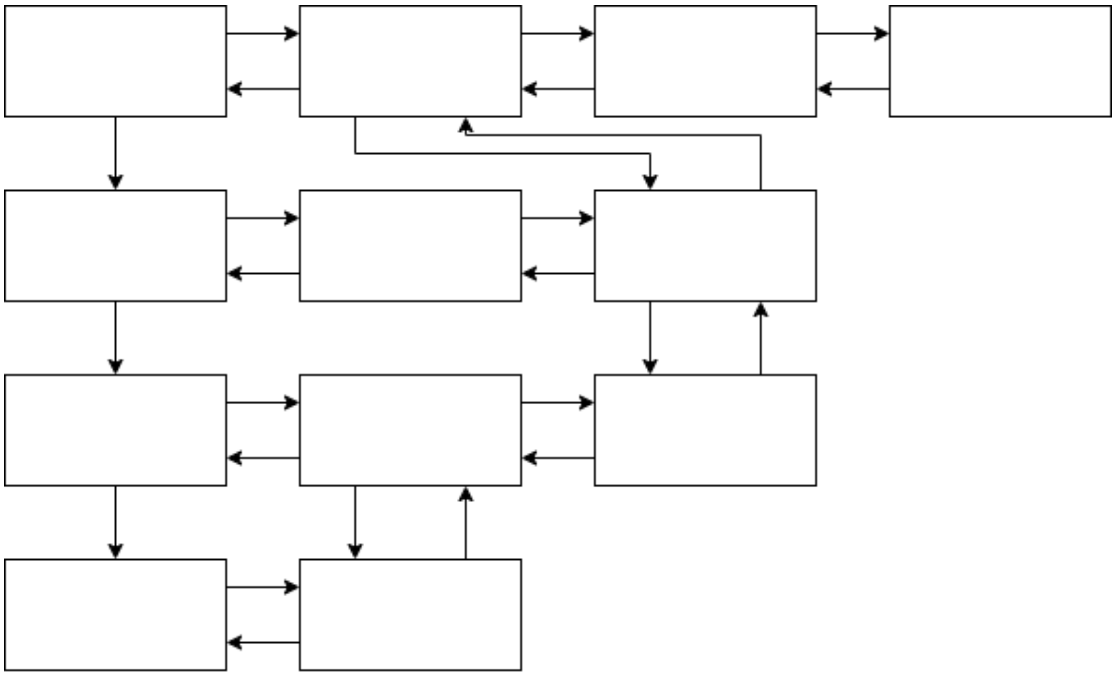


图 3-1 cnf 公式存储结构图

crossNode	*up	*down	*next	*left
节点	指向上一行相同文字	指向下一行相同文字	指向下一个节点	指向上一个节点

图 3-2 十字链表节点存储结构图

这部分要写的：（1）首先描述系统中要处理那些数据，每种类型的数据包括哪些数据项，每个数据项的数据类型，最后可用一个表格表示出来；（2）描述这多种数据在系统中如何关联，可通过图直观的说明这多种数据间的关联。

文字节点中，设置有 up,down,next,left 指针。up 指针用来指向上一行相同的文字，down 指针用来指向下一行相同的文字，next 指针用来指向下一个节点，left 指针用来指向上一个节点，并且设置了一个标记的布尔变量，负责标记这个文字是否被删除，且设置一个整形变量负责记录这个文字当前是真还是

假。

在子句节点中，设计 boolNum 来记录包含的变元的个数，设置 clauseNum 来记录子句个数，remainClauseNum 来记录剩下的子句个数，定义 sum 来记录每个子句剩余文字个数。在 Diagdoku 模块中，定义了一个二维数组 map 来记录数独中的信息。

3.2 主要算法设计

3.2.1 cnf 文件读取

cnf 文件操作时，文件名构造一个 crosscnf 的类，并利用其中的构造函数，传入文件的地址，将文件中的信息存储到十字链表中。

（1）读取.cnf 文件

利用 new 申请空间，将.cnf 文件中数据读入十字链表，各子句和文字的标志 flag 设置为 0，同时在问题规模结构体中记录每个文字的出现次数。

（2）输出 cnf 文件

遍历整个十字链表，并将关键信息在屏幕上输出。

3.2.2 DPLL 算法

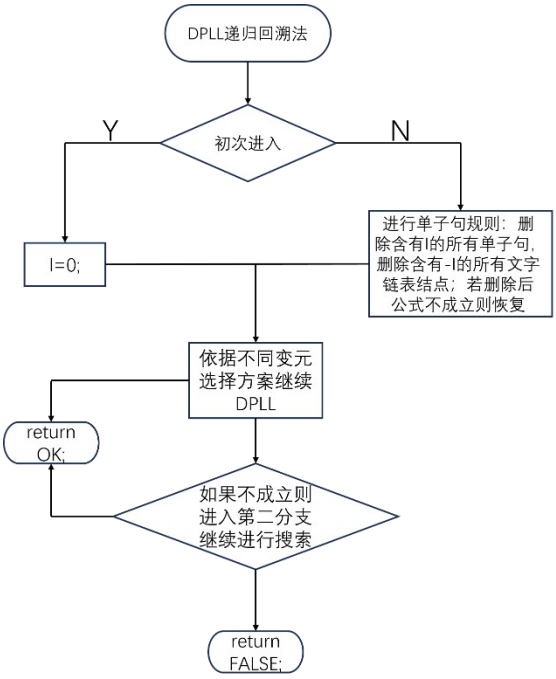


图 3-3 递归算法 DPLL 函数

（1）单子句规则

在子句集中寻找单子句，假设找到子句集 S 中单子句为 L，L 取真值，更新答案数组。再根据单子句规则，屏蔽子句集中所有包含 L 的句子（即将结点的标志赋值为递归深度，初始深度为 1），同时屏蔽所有¬L 的文字，更新子句中文字的个数。执行屏蔽操作时，更新各个变元出现的次数。

（2）分裂策略

首先基于不同的变元选择策略选取一个文字 L。假设 L 取真值，则将其作为下 一次递归中的目标单子句。根据单子句传播策略递归往复，直到无法找到可赋值的变元时函数执行完毕；如果在执行过程中出现空子句，则表示出现冲突。此时 回溯至上一级，恢复在上一级中删除的文字和子句，修改子句中文字的个数。接着假设¬L 取真值，再次进入递归，直到求解成功。若所有情况都无法求解成功，则原范式是不可满足的。

（3）答案保存与检验

求解完成后，显示范式是否满足和求解所用时间，并将问题结构体中数组所保存的答案录入同名但以.res 结尾的文件中，文件中同时保存求解结果，求解答案与求解时间。

3.2.3 对角线数独游戏

（1）主要的思路在于生成满的数独盘，将数独存储到 cnf 文件里面，利用挖洞法生成符合要求提示个数的数独盘，以及验算用户所填写的答案是否正确。

（2）cnf 范式归约

cnf 约束重要性在于可以利用一个写着 cnf 范式归约的文件来求解并以此来生成一个数独解。对角线数独是在普通数独的基础上衍生出的变种，要求在 9×9 的大九宫格中填入数字 1～9，使他们满足如下规则：每个数字在每个小九宫格内不能出现一样的数字，在每行、每列和每条大对角线中也不能出现一样的数字。每个方格中可以填 1~9 中任意数字，故文字数为 81×9=729 个。在 cnf 文件中布尔变元采用逐行从左到右、从上到下由 1 至 729 存储，其中每个方格含有编号为 9n+1~9n+9 的九个布尔变元。

（3）数独格局生成

调用 dpll 算法去求解事先准备的写好 cnf 范式的 cnf 文件，求解出一组解，并将 这组解存储到一个 res 文件里面，再利用一个 map 数组读取这组解，并按照玩家需求，显示相应个数的数独盘。

（4）数独盘的填写

填写数独的时候需要输入想要填写位置的横坐标和纵坐标以及想要填入的数字，填写的数独内容先存储到 map 数组里面，再通过一个负责展示的函数，实时将命令行中的数独盘更新。

（5）数独的验证

将 map 数组中填写的内容和 res 文件中存好的答案进行对比，如果一样的话 说明答案正确，不过不能保证只有唯一解的情况。

3.3 程序优化

相较于普通链表的存储结构和朴素 DPLL 算法，本程序对结构和算法层面做 了以下优化。

3.2.1 结构优化

在 cnf 文件的存储与 DPLL 算法操作中，采用十字链表存储归约，以一种更直观和紧凑的方式表示布尔公式，通过 up、down、left 和 right 指针，可以轻松地访问和遍历与相同布尔变量、相同子句或相同文字相关的节点，而无需额外的复杂操作；设置 flag 标记是否删除和记录递归深度。这样可以在子句和文字的删除操作时，通过改变标志使删除对象在后续不进行遍历，从而避免了删除和修改指针带来的时间消耗与复杂操作，也减少了出现错误的几率；在读取 cnf 文件并存储其中信息的时候，记录每个文字是正还是负出现的次数，方便在后续进行假设的时候优先假设（如果一个文字是正数出现的次数多，就优先假设正为真值，反之亦然），且在统计次数时也有技巧，将负数通过公式转换成成一种较大的正数，分开了正负文字的统计，使其他函数处理负数文字更加方便。

3.2.2 算法优化

在 DPLL 求解模块中，朴素 DPLL 算法在分裂策略选取变元采用的是随机变元选择，即选取第一个子句的第一个变元，本程序在朴素的 DPLL 算法的基础上引进了最短子句出现频率最大优先策略(MOM)和 cdcl(Conflict-Driven Clause Learning)算法。

MOM 策略：在一个子句中，只要一个文字得到满足，那么这个子句就得到满足了。所以，如果一个子句的长度越长（含有字母数越多），那么可以使得该子句满足的文字数目也就越多，这个子句也就越容易满足，所以它就不是所谓的“矛盾的主要方面”，我们不需要过于关注这个子句；然而，如果一个子句长度很小，那它就很不容易被满足，所以我们要优先考虑它们，给予它们更高的权重，这样做的目的就是让那些不容易被满足的子句优先得到满足。

Conflict-Driven Clause Learning：直译过来是“冲突驱动的子句学习”，也就是从冲突中吸取教训，做出更合理的猜测。所谓冲突，就是出现某个子句的所有文字都为 false，使得该子句不可能被满足。出现这种情况时，我们可以从中总结出关于文字取值的一些限制，根据这些限制进行决策，进而提升算法效率。CDCL 相比于 DPLL 最大的特点是“non-chronological back-jumping”，即“非时序性回溯”，换言之就是回溯时不一定回到上一层，而可能回到上几层。从冲突中吸取教训的过程称为子句学习。当冲突发生时，我们分析冲突发生的原因，学习一个新的子句并加入 CNF 中，该子句可以使得这个冲突被避免，并且其他类似的冲突也可能被避免。完成子句学习后，我们进行回溯，回溯到哪一层取决于刚才分析的结果。

在引入了两个优化策略后，SAT 求解器在中大型算例中表现更加出色，而在小型算例中，由于朴素的 DPLL 算法不需要额外的时间消耗，故朴素的 DPLL 算法在小型的算例中有着更优的表现。

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：AMD Ryzen 7 6800H with Radeon Graphics　　3.20 GHz

机带 RAM：16.0 GB

系统类型：64 位操作系统, 基于 x64 的处理器

2. 软件环境：

Windows11 下 Microsoft Visual Studio Community 2022 (64 位) 17.2.6

4.1.2 数据类型定义

```
#define MAXN 10000
enum BOOLVALUE { UNSURE, FALSE, TRUE };

struct crossNode
    //十字链表节点
{
    int Bool;//表示自己的布尔变元
    int Clause;//表示自己的子句
    bool del;
      crossNode* up, * down, * right, * left, * next;
      crossNode()
      {
          Bool = 0;
          Clause = 0;
          up = NULL;
          down = NULL;
          right = NULL;
          left = NULL;
          next = NULL;
          del = false;
      }

};

```

```
25.  struct intStack//手动实现int栈，这里使用数组栈
26.
27.  {
28.      int S[MAXN];
29.      int i = 0;
30.      void push(int value)
31.      {
32.          S[i++] = value;
33.      }
34.
35.      int pop()
36.      {
37.          //assert(i>0);
38.          return S[--i];
39.      }
40.
41.      bool empty()
42.      {
43.          return i == 0;
44.      }
45.
46.  };
47.
48.  class crossCNF
49.      //十字链表cnf
50.  {
51.  public:
52.      crossCNF(const char* const filename);//构造函数
53.      crossCNF(FILE* fp);//构造函数（文件指针版）
54.      ~crossCNF();//析构函数
55.      bool calculate(const char* const filename);//检验
56.      void print(FILE*);//打印
57.      bool solve(const char* const filename, bool disp
     lay);//名义解
58.      bool solve(const char* const filename, const cha
     r* const filename0, bool display);
59.      int randomSolve(int Ans[]);
60.      int randomBelieve(int L, int level);
61.      int randomInnerSolve(int level);
62.
63.  private:
64.      int boolNum;//布尔变元个数
65.      int clauseNum;//子句个数（原来的）
```

```
66.      int remainClauseNum;//删了一些之后剩下的子句个数
67.      int* sum;//每个子句的剩余文字个数
68.      int* tendency;//每个文字的“倾向”
69.      crossNode* bools;//编号从1开始
70.      crossNode* clauses;//编号从1开始
71.      BOOLVALUE* hypo;//搜索时使用的“假设”
72.      intStack single;//存单子句的栈
73.      void addNode(int, int);//添加结点，生成cnf对象时要
         用
74.      int changeBool(int);//把[-boolNum,-
         1]U[1,boolNum]翻译成[1,2*boolNum]的函数
75.      int innerSolve();//实际解
76.      int believe(int);//“相信”，具体含义请看solver.cpp

77.      void restore(int L, int startNum, crossNode* Hea
         d);//恢复被believe函数修改过的cnf
78.
79.  };
```

4.1.3 SAT 求解模块主要函数

Class crossCNF 这个类中负责 cnf 文件的存储，DPLL 求解和验算。

（1）crossCNF(const char * const filename);

构造函数，将构造根据传入的文件名称，读取 cnf 信息并存储到十字链表。

（2）bool solve(const char* const filename,bool display);

负责调用 DPLL 进行求解存储的 cnf 范式归约。

（3）void print(FILE *);

将存储的 cnf 输出到命令行。

（4）bool calculate(const char * const filename);

负责检验所求解的答案是否正确，先将 res 文件中的答案读取到数组里面，再 对每一个子句进行遍历，当遇到第一个为真的文字时候，这个子句为真，进行下 一个子句的遍历，当全部遍历完的时候，即所求答案为正确的。

4.1.4 Diagdoku 模块主要函数

class Diagdoku 这个类中的成员变量和成员函数负责主要的数据的存储和 功能实现

（1）Diagdoku();

这是一个构造函数，负责初始化装着数独信息的二维数组，都初始化为空。

（2）void load(const char* const filename);

这是一个参数为字符串指针的函数，接收传进来的参数，读取指定 res 文

件 里面以及填好的答案，并将其存储到二维数组里面。

（3）void print();

将存储好的数独信息的数组显示到命令行窗口，形成一个 ui 界面，供用户交互功能使用。

（4）void generate_cnf(const char* const filename);

将数独中已经填写的内容存储到本地的 cnf 文件里面，其实就是在原来的 cnf 范式归约上再加上数独盘中已经填写个数的单子句规则。

（5）int content_num();

计算数独中已经填写的位置的个数，并将其返回。

（6）void randomGenerate(int n);

这是一个负责随机生成数独题目的成员函数，先将负责存储数独的二维数组 初始化为空，然后调用 dpll 求解器求解指定的 cnf 文件中的 cnf 范式归约，并将完整的数独盘的信息存储到我们自己定义的一个二维数组里面，最后再通过挖洞法得到最后的数独题。

4.2 系统测试

4.2.1 交互系统展示

主交互系统界面如图 4.2.1-1 所示，SAT 交互系统界面如图 4.2.1-2 所示，数独游戏交互系统界面如图 4.2.1-3 所示：

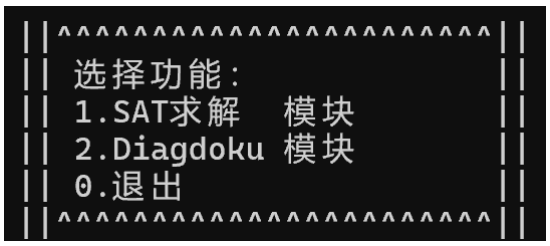


图 4.2.1-1 主交互系统

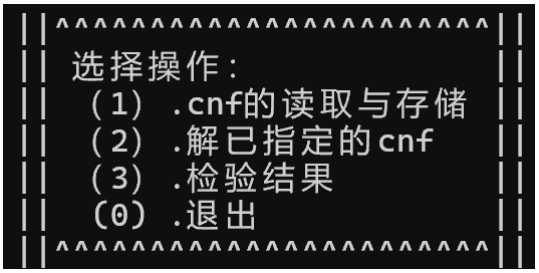


图 4.2.1-2 SAT 交互系统

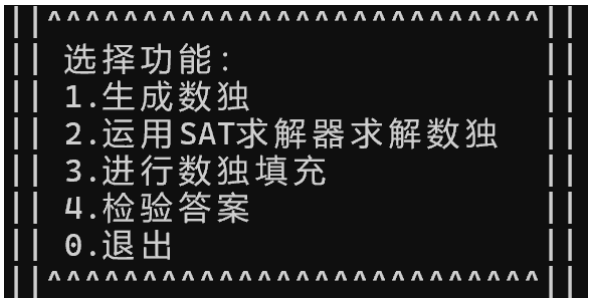


图 4.2.1-3 数独游戏交互系统

4.2.2 SAT 求解模块测试

（1）cnf 文件读取功能测试

此功能可读取 cnf 文件，将数据信息保存在问题规模结构体中。本测试将测试程序是否能成功读入文件并显示子句数和文字数。测试 1 为正确文件名，测试 2 为错误文件名，可测试程序能否对异常输入做出相应的提示。测试结果如下表格及图片所示：

测试输入	目标文件	预期结果	实际结果
1	算例/1.cnf	文件指定成功	1 指定需要求解的cnf文件名： 算例/1.cnf 输出文件被指定为算例/1.res 请按任意键继续. . .
1	1.cnf	打开文件失败，重新输入	1 指定需要求解的cnf文件名： 1.cnf 打开文件失败！重试！

表 4.2.2-1 cnf 文件读取功能测试

（2）cnf 文件求解功能测试

过程简要介绍：在主交互系统界面选择 1.SAT 求解模块，选择 1 指定想要求解的 CNF 文件名。选择 2.求解已经指定的 cnf 算例，有变元随机选取策略、最短子句出现频率最大优先策略和 cdcl 冲突驱动的子句学习策略三个选项，求解完成后会输出求解耗时并将求解结果保存在与 CNF 文件同名的.res 文件中，可以在 SAT 交互系统输入 3 输出求解结果；

测试算例 1：算例/1.cnf；先读入所要求解的 CNF 文件名（如图 4.2.2-1 所示），使用三种策略求解测试算例，结果分别如图 4.2.2-2，图 4.2.2-3，图 4.2.2-4 所示，遍历测试算例 CNF 文件，保存 SAT 的求解结果，求解结果和保存结果分别如图 4.2.2-5，图 4.2.2-6 所示（测试算例 CNF 文件的遍历结果将省略）：

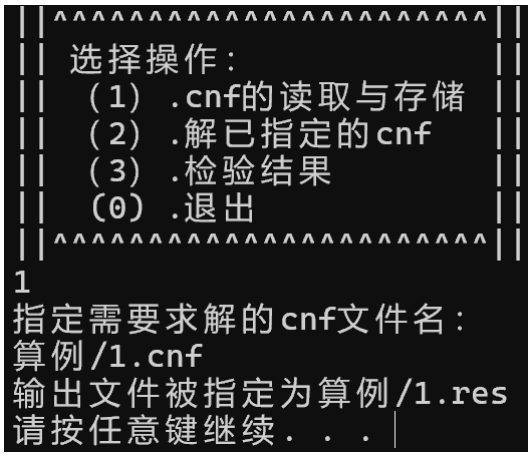


图 4.2.2-1 读入所要求解的 CNF 文件名

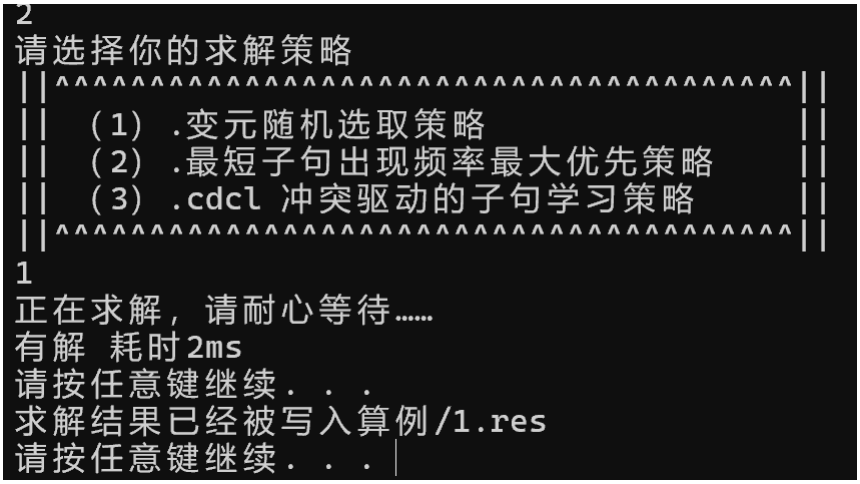


图 4.2.2-2 采用变元随机选取策略求解

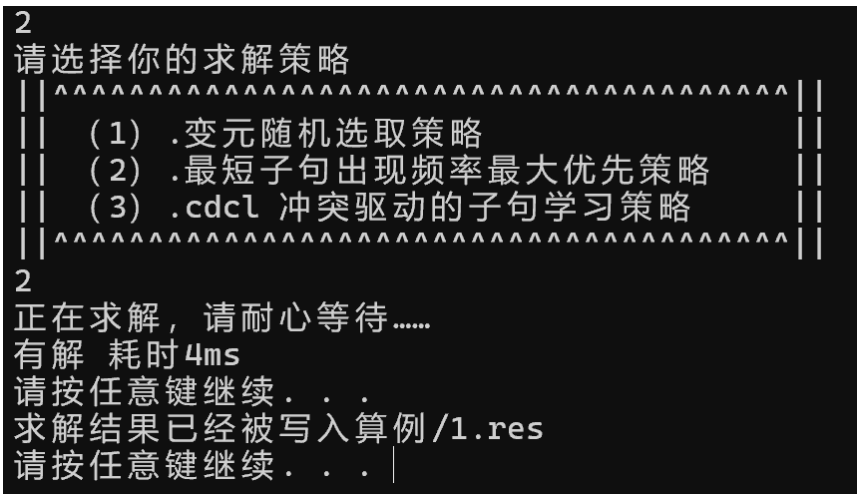


图 4.2.2-3 采用 MOM 策略求解

```
2
请选择你的求解策略
||^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^||
||  (1) .变元随机选取策略                     ||
||  (2) .最短子句出现频率最大优先策略         ||
||  (3) .cdcl 冲突驱动的子句学习策略          ||
||^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^||
3
正在求解，请耐心等待……
有解 耗时36ms
请按任意键继续. . .
求解结果已经被写入算例/1.res
请按任意键继续. . .
```

图 4.2.2-4 采用 cdcl 策略求解

```
res=[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 -28 -29 -30 -31 32 33 -34
35 -36 37 -38 -39 40 41 -42 -43 -44 45 -46 47 48 -49 50 -51 -52 53 54 -55 -56 -57 58 59 60 61 62 63 64 -65 66 67 -68 -69
 -70 -71 72 73 -74 -75 -76 -77 -78 -79 80 -81 -82 -83 84 -85 -86 -87 -88 -89 90 91 -92 -93 94 95 -96 97 98 99 -100 101 -
102 -103 -104 105 106 107 108 109 -110 -111 -112 -113 -114 115 -116 -117 118 119 -120 -121 122 -123 -124 125 126 -127 -1
28 -129 -130 131 -132 -133 -134 135 136 137 -138 -139 -140 -141 -142 -143 144 -145 146 147 148 149 -150 -151 -152 153 -1
54 -155 -156 -157 158 -159 -160 -161 -162 163 164 165 166 -167 168 169 170 -171 -172 173 -174 -175 -176 -177 -178 -179 1
80 181 -182 -183 -184 -185 186 -187 -188 -189 -190 -191 -192 -193 194 -195 196 -197 -198 199 200 ]
```

图 4.2.2-5 输出求解结果

```
s 1
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 -28 -29 -30 -31 32 33 -34 35 -36 37 -38 -39 40 41 -42 -43 -44
45 -46 47 48 -49 50 -51 -52 53 54 -55 -56 -57 58 59 60 61 62 63 64 -65 66 67 -68 -69 -70 -71 72 73 -74 -75 -76 -77 -78 -79 80 -81 -82 -83
84 -85 -86 -87 -88 -89 90 91 -92 -93 94 95 -96 97 98 99 -100 101 -102 -103 -104 105 106 107 108 109 -110 -111 -112 -113 -114 115 -116 -117 118
119 -120 -121 122 -123 -124 125 126 -127 -128 -129 -130 131 -132 -133 -134 135 136 137 -138 -139 -140 -141 -142 -143 144 -145 146 147 148
149 -150 -151 -152 153 -154 -155 -156 -157 158 -159 -160 -161 -162 163 164 165 166 -167 168 169 170 -171 -172 173 -174 -175 -176 -177 -178 -179
180 181 -182 -183 -184 -185 186 -187 -188 -189 -190 -191 -192 -193 194 -195 196 -197 -198 199 200
t 36
```

图 4.2.2-5 SAT 求解结果保存

采用 verify.exe 验证，上述求解均正确。

（3）DPLL 优化性能测试

测试 DPLL 算法的优化性能，主要是针对变元选取策略进行优化。本程序在变元随机选取策略的基础上，对中大规模算例引进了最短子句出现频率最大优先策略和 cdcl 冲突驱动的子句学习策略两种优化。

不同的算例寻求减少 DPLL 的求解时间时，采用的变元选取策略未必相同。随着算例规模的增大，这种区分会愈发显著。因此，在不改变 DPLL 算法的递归结构下，并不存在一种普适的算法能够实现绝对的优化效果。即使采用寻找子句 集中的首个变元这一策略，在处理某些特定算例时也可能耗费极短的时间。故在 测试过程中可能出现负优化的情况，属于正常现象。

算例名	文字数	子句数	朴素 DPLL 求解时间 (ms)	MOM 策略求解时间 (ms)	Cdcl 策略求解时间 (ms)	优化率
1.cnf	200	1200	2	4	36	-100%
2.cnf	1075	3152	84	64	92	24%
3.cnf	301	2780	2	1289	5	-150%

4（unsatisfied）.cnf	512	9685	84	397	60	29%
5.cnf	20	1532	21	7	112	67%
6.cnf	265	5666	3324	9734	85	97%
7（unsatisfied）.cnf	3000	8881	1444580	841354	324156	77%
11（unsatisfied）.cnf	60	936	31128	1585	21456	95%

注：优化率采用优化后最优情况与朴素 DPLL 算法求解时间计算

表 4.2.2-2 DPLL 优化性能测试

4.2.3 对角线数独游戏测试

数独部分简要介绍：数独部分分为两个功能，一是基与挖洞法随机生成数独游戏盘，并且通过控制挖洞数量可选择生成不同难度的数独盘，可以提供给用户自行求解；二是将生成或读取的原始数独盘进行归约，转化为 CNF 文件,再利用 SAT 求解器求解 CNF 文件，最终将求解结果转换为数独数字打印到屏幕上显示。

（1）生成数独测试

本测试将测验程序能否正常生成空数独和有提示数的数独。生成空数独的目 的是生成对角线数独的 cnf 归约，以便输入初始合法格局进行求解。测试结果如图 4.2.3-1 和图 4.2.3-2 所示：

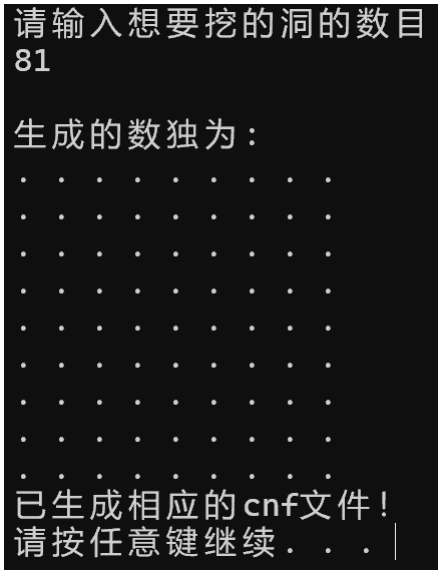


图 4.2.3-1 生成空数独

图 4.2.3-2 生成有提示的数独

（2）自动求解功能测试

本测试将测验程序能否对对角线数独进行自动求解。测试结果如下图所示：

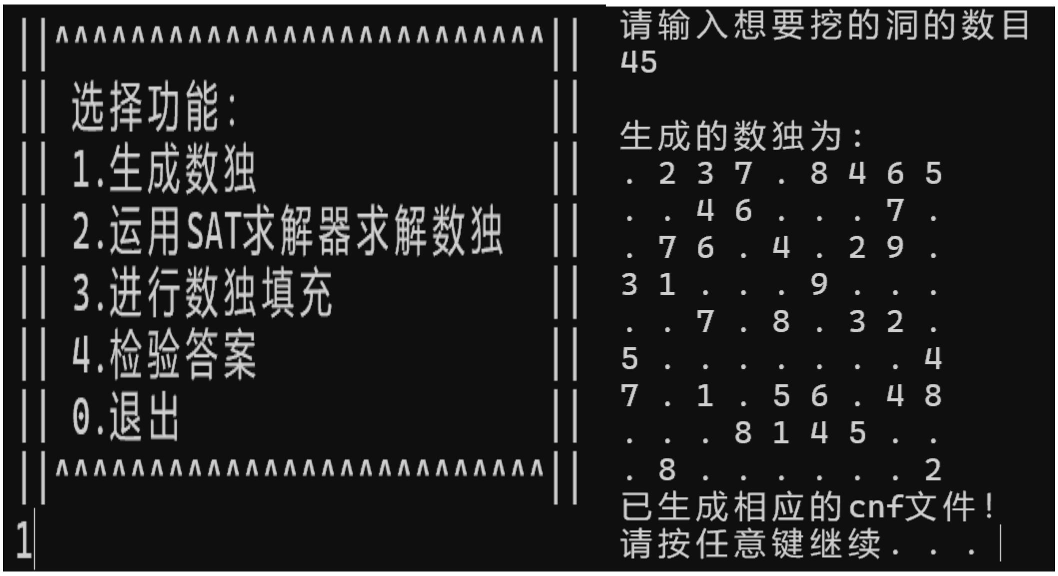
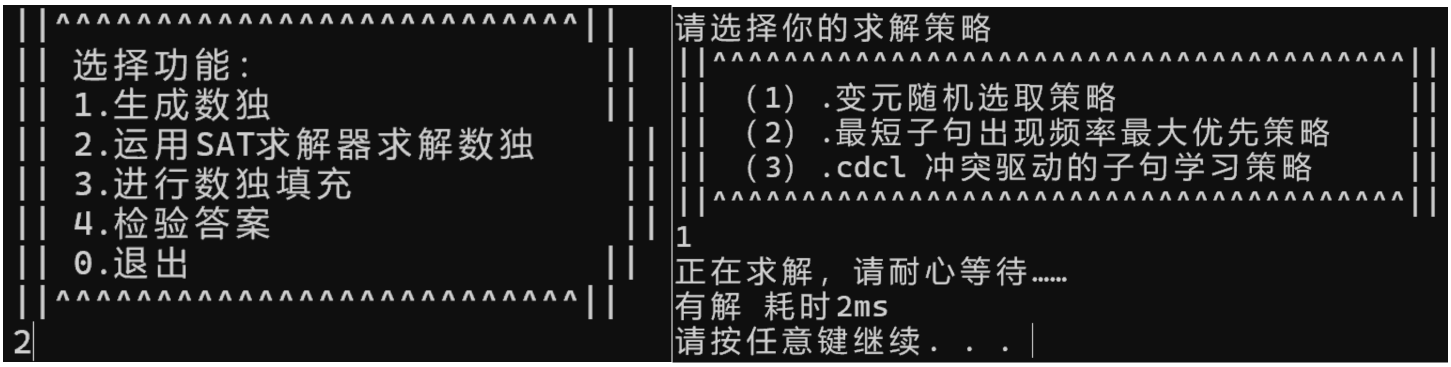


图 4.2.3-3 生成数独



```
请按任意键继续. . .
标准答案:
1 2 3 7 9 8 4 6 5
9 5 4 6 3 2 8 7 1
8 7 6 1 4 5 2 9 3
3 1 8 4 2 9 7 5 6
6 4 7 5 8 1 3 2 9
5 9 2 3 6 7 1 8 4
7 3 1 2 5 6 9 4 8
2 6 9 8 1 4 5 3 7
4 8 5 9 7 3 6 1 2
SAT求解答案:
1 2 3 7 9 8 4 6 5
9 5 4 6 3 2 8 7 1
8 7 6 5 4 1 2 9 3
3 1 8 4 2 9 7 5 6
6 4 7 1 8 5 3 2 9
5 9 2 3 6 7 1 8 4
7 3 1 2 5 6 9 4 8
2 6 9 8 1 4 5 3 7
4 8 5 9 7 3 6 1 2
```

图 4.2.3-4 自动求解数独

（3）用户交互功能测试

此功能使用户通过输入行列数和填充内容在对角线数独中填入数字，实现数独的手动求解。测试将测验程序能否由用户输入手动求解对角线数独，以及能否对用户不合理输入进行纠正。

```
当前数独如下
3 . . . 6 . 8 . 4
. . 5 8 . 1 . . 6
9 . . 4 . 7 . 2 .
4 5 . . 7 . 6 . .
6 2 7 9 . . . 5 .
. . . 6 4 . . 1 2
1 6 . . . 2 9 . 5
5 . 4 . 9 . 2 8 .
2 9 . 3 5 4 1 . .
请输入你想填入数独的位置（i，j），以及填入的数字num（只需输1个0退出填写）
```

图 4.2.3-5 挖洞法生成数独 手动求解前数独盘状态

```
请输入你想填入数独的位置（i，j），以及填入的数字num（只需输1个0退出填写）
1 3 2 0
当前数独如下
3 . 2 . 6 . 8 . 4
. . 5 8 . 1 . . 6
9 . . 4 . 7 . 2 .
4 5 . . 7 . 6 . .
6 2 7 9 . . . 5 .
. . . 6 4 . . 1 2
1 6 . . . 2 9 . 5
5 . 4 . 9 . 2 8 .
2 9 . 3 5 4 1 . .
```

图 4.2.3-6 手动输入后数独盘状态

（4）手动求解数独检验测试

图 4.2.3-7 对手动求解的数独进行检验

5 总结与展望

5.1 全文总结

整个课程设计流程中，主要工作如下：

（1）研究与理解可满足性问题： 首先，需要深入学习和理解可满足性问题 的概念、研究背景以及其在计算机科学中的重要性。通过文献资料的查阅，掌握 可满足性问题的历史发展、国内外研究现状以及我校在这一领域的研究成果。这 也包括了对可满足性问题的研究前景有清晰的认识。

（2）构建 CNF 数据结构和 DPLL 算法实现：自行设计和建立用于存储 CNF 公式的数据结构，并实现 DPLL 算法以解决可满足性问题。

（3）模块化算法开发： 将算法的整体框架进行模块化处理，分阶段编写和 调试各个模块的代码。确保每个模块的功能正常运作。

（4）整合核心算法和 SAT 问题转化代码：将编写的模块整合成 DPLL 算法的主体部分，并根据提供的 SAT 测试案例对 DPLL 算法进行调试。同时，编写代码将两类数独问题转化为 SAT 问题的形式。

（5）构建完整工程并设计界面： 组织已编写的模块和代码，构建一个完整 的工程。此外，设计用户界面，以便用户能够使用和测试算法。

（6）代码注释与文档编写： 对所有代码进行分类和组织，编写注释以解释 每个头文件的作用以及包含的代码块。同时，编写文档，记录课程设计的整个流 程、算法细节和界面说明。

以上工作的完善和执行将有助于顺利完成课程设计，同时确保代码的可维护 性和可理解性，以及对可满足性问题的深入理解和应用。

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）多线程优化：我将深入学习多线程编程，以提高 DPLL 算法的时间性能。通过并行化处理，将尝试减少求解时间，使算法更高效。

（2）CDCL 算法探索：我将持续研究冲突驱动子句学习（CDCL）算法，以进一步优化 DPLL 算法的回溯过程。这将有助于提高算法在复杂问题上的解决能 力。

(3) 处理大型算例：我将致力于解决计算机设备的限制，以克服 DPLL 算法在处理大型算例时的局限性以及不稳定性。这包括优化内存使用和算法设计，以处理更大规模的问题。

(4) Qt 界面设计学习：我将学习 Qt 界面设计技巧，以改进算法的用户界 面。通过提高菜单和数独游戏的交互性，使用户更容易使用和理解算法。

(5) 代码简化和美化：我将进行代码审查，去除冗余设计，以使代码更简洁和易于维护。这将有助于提高代码的可读性和可维护性。

通过这些改进和研究方向，我希望能够进一步提升 DPLL 算法的性能、功能 和用户体验，以更好地应对现实世界中的可满足性问题。

6 体会

在本次课程设计的过程中我收获到了很多编程知识，也认识到了自己的许多不足，比如实现的 SAT 求解器性能较低，数独的归约逻辑也相对冗繁等等。以下是我按照完成整个课设的时间顺序写的一些心得体会：

1．万事开头难。在最开始的时候需要阅读很多次任务书才能较为清晰地感受到实现程序的大致方法，比如数独棋盘创建 CNF 文件的多个约束，将变元语义编码转换为自然顺序编码的方式等等。当自己慢慢上手摸索时，经历了多次失败才逐渐开始懂得如何去构建 SAT 求解器，实现 DPLL 递归函数和构建数独系统；

2．在实现基于 DPLL 的 SAT 求解器的过程中，我上网找了很多资料来理解递归、回溯等算法思想。由于设计了较复杂的结构体，所以回溯的时候都要考虑到。在此过程中，经历了多次失败和屡次调试后我才明白问题出在哪，需要耐心细致地对程序进行 Debug。最终实现了 DPLL 算法，但是变元的选择策略还未太掌握清楚。

3．在数独游戏的实现过程中，查阅了很多资料才知道挖洞法的具体创建方法，即对已知数独终盘进行挖洞，第一次挖洞后数独盘的解一定是唯一的，所以不用考虑唯一性。之后每次挖洞需要 DPLL 且只能求解出唯一解，否则挖洞失败。整个过程一一罗列后逐步实现，需要足够的耐心和细致的观察思考；

4．最后就是将各部分整合为项目的问题，由于从来没有做课程设计的经验，以为项目整合并非难事，然而在网上看了教程，自己实际操作一遍后才发现有很多的问题。主要问题是在刚开始写各个模块的时候，变量、数据结构等定义得比较随意，导致项目整合的时候进行了诸多调整，浪费了许多时间。还有一些类似于重复定义的问题，起初无法解决，也是通过上网查阅相关资料才慢慢理解和完善这些问题。

5．总而言之，在实现这次课程设计的过程中我收获了非常多的编程知识和实操经验，懂得了实践出真知的真理。代码之路需要充足的细致与耐心，路漫漫而其修远兮，吾将上下而求索。

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

display.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<stdexcept>
#include<cstdio>
#include<cstdlib>
#include "solver.h"
#include "Diagdoku.h"
#include <ctime>
#include <vector>
#include <algorithm>
#include <time.h>
#include <numeric>
#include<string>
using namespace std;
extern int random;
#define N 9  // 9x9 数独
int inputOrder(int stt, int end, const char* const text = NULL)

//这个函数用来输入选项，选择范围是[stt,end]
//将此单独设置成函数，可以避免重复的检查输入是否正确
//以返回值的形式输出结果
//会把字符串 text 输出在屏幕上，以提示用户输入
{

    if (end < stt)throw runtime_error("in inputOrder():end<stt\
n");//遇到输入问题时报错。如真报错，肯定是代码问题而非输入问题
    int i, ans = -1;
    /*
    i:循环变量
    ans:待返回的值

    */
    char s[1000];//输入的一行，应该没有人会一行打 1000 个字符吧
    bool ok = false;//用于标记输入是否合法。
    if (text)printf("%s", text);//输出提示文字
    while (1)//主循环。如输入合法，返回输入，否则再次循环
    {
        fgets(s, sizeof(s), stdin);//“安全地”从屏幕上读入一行字符串
```

```
37.         ok = true;
38.
39.         for (i = 0; s[i] != '\n'; i++)//检查输入是否含有非数字字符
40.         {
41.             if (s[i] < '0' || s[i]>'9')
42.             {
43.                 printf("输入含有非数字字符(%c,%d)，请重试\n", s[i],
    s[i]);
44.                 ok = false;
45.                 break;//跳出 for 循环
46.             }
47.         }
48.         s[i] = '\0';//标记字符串的末尾
49.         if (ok == false)continue;
50.         if (strlen(s) == 0)continue;//这是为了应对用户输入一个空行的
    情况
51.
52.         ans = atoi(s);// atoi:把字符串形式的整数转化为 int
53.         if (ans<stt || ans>end)// 判断是否超出范围
54.         {
55.             printf("输入超出范围，请重试\n");
56.             ok = false;
57.             continue;
58.         }
59.         return ans;//如未超出范围，返回
60.     }
61.
62.
63.
64. }
65.
66. void useDPLL()
67. //此函数用于演示 dpll 模块
68. {
69.     srand(clock());
70.
71.     char filename[500];//cnf 文件名
72.     char filename2[500];//res 文件名
73.     crossCNF* cnf = NULL;//待定的 cnf 对象
74.     int choice;//选项
75.     FILE* fp = NULL;//待定文件指针
76.     bool solved = false;//表示当前 cnf 是否被解过。被解过的 cnf 对象数据
    结构会被破坏，不能再解，再解需要重新从文件中读取
77.     while (1)//主循环
```

```
78.     {
79.         system("cls");//清屏
80.         choice = inputOrder(0, 4,
81.             "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
82.             "|| 选择操作:              ||\n"
83.             "|| （1）.cnf 的读取与存储 || \n"
84.             "|| （2）.解已指定的 cnf    ||\n"
85.             "|| （3）.检验结果         ||\n"
86.             "||  (0) .退出             ||\n"
87.             "||^^^^^^^^^^^^^^^^^^^^^^^^||\n");
88.
89.         switch (choice)
90.         {
91.         case 0: {//退出
92.             if (cnf != NULL)delete cnf;//释放 cnf
93.             return;
94.         }
95.         case 1: {//指定 cnf
96.             //除了指定 cnf 文件名之外，这个块还需要重新初始化其他内容
97.             if (cnf != NULL)
98.             {
99.                 printf("cnf 已经定义，正在释放旧 cnf\n");
100.                delete cnf;//释放 cnf
101.                cnf = NULL;
102.                fclose(fp);
103.                fp = NULL;
104.            }
105.
106.
107.            while (fp == NULL)
108.            {
109.                printf("指定需要求解的 cnf 文件名：\n");
110.                scanf("%s", filename);
111.
112.                fp = fopen(filename, "r");
113.                if (fp == NULL)printf("打开文件失败！重试！\n");
114.            }
115.            //现在已经打开文件成功
116.            cnf = new crossCNF(fp);//重新初始化
117.            solved = false;
118.            //然后还要重新赋值 filename2
119.            strcpy(filename2, filename);
120.            int i = 0;
```

```
121.                while (filename2[i] != '.')i++;
122.                filename2[i + 1] = 'r';
123.                filename2[i + 2] = 'e';
124.                filename2[i + 3] = 's';
125.                filename2[i + 4] = '\0';
126.
127.                printf("输出文件被指定为%s\n", filename2);
128.                break;
129.            }
130.
131.            case 2: {//求解已指定的cnf
132.                if (cnf == NULL)
133.                {
134.                    printf("还未指定cnf!\n");
135.                    break;
136.                }
137.                if (solved)
138.                {
139.                    printf("这个cnf已经被解过， 数据结构被破坏，请重新
    初始化！\n");
140.                    break;
141.                }
142.
143.                cnf->solve(filename2, filename, true);
144.                solved = true;
145.                printf("求解结果已经被写入%s\n", filename2);
146.                break;
147.            }
148.
149.            case 3: {//验证并显示
150.                if (cnf == NULL)
151.                {
152.                    printf("还未指定cnf!\n");
153.                    break;
154.                }
155.                if (solved == false)
156.                {
157.                    printf("这个cnf还未被求解过！\n");
158.                    break;
159.                }
160.                cnf->calculate(filename2);
161.                break;
162.            }
163.
```

```
164.
165.          }
166.          system("pause");
167.      }
168.
169.
170.
171.  }
172.
173.
174.  // 将 (i, j, k) 转换为唯一的整数变量
175.  int var(int i, int j, int k) {
176.      return (i - 1) * N * N + (j - 1) * N + k;
177.  }
178.  void inverse_var(int v, int& i, int& j, int& k) {
179.      v = v - 1;  // 转换到从 0 开始的索引
180.      i = v / (N * N) + 1;
181.      j = (v % (N * N)) / N + 1;
182.      k = v % N + 1;
183.  }
184.  void toCnf(vector<vector<int>>& sudoku, int holes) {
185.      FILE* fp = fopen("Diagdoku.cnf", "w");
186.
187.      if (!fp) {
188.          printf("Cannot open file for writing!\n");
189.          return;
190.      }
191.      fprintf(fp, "c\nc\n");
192.      fprintf(fp, "p cnf 729 %d\n", 9558 - holes);     //
    9477+81-holes 对角线数独限制条件 81-holes 填入的数
193.
194.      // 单子句
195.      for (int row = 0; row < sudoku.size(); ++row) {
196.          for (int col = 0; col < sudoku[row].size(); ++col) {
197.              int num = sudoku[row][col];
198.              if (num != 0)
199.              {
200.                  fprintf(fp, "%d %d\n", var(row + 1, col + 1,
    num), 0);
201.              }
202.          }
203.      }
204.      // 每行每列每块的每个数字只能出现一次
205.      for (int x = 1; x <= 9; x++) {
```

```
206.          for (int y = 1; y <= 9; y++) {
207.              for (int z = 1; z <= 9; z++) {
208.                  fprintf(fp, "%d ", (x - 1) * 81 + (y - 1) * 9
    + z);
209.              }
210.              fprintf(fp, "0\n");
211.          }
212.      }
213.      // 行
214.      for (int y = 1; y <= 9; y++) {
215.          for (int z = 1; z <= 9; z++) {
216.              for (int x = 1; x <= 8; x++) {
217.                  for (int i = x + 1; i <= 9; i++) {
218.                      fprintf(fp, "-%d -%d 0\n", (x - 1) * 81 +
    (y - 1) * 9 + z, (i - 1) * 81 + (y - 1) * 9 + z);
219.                  }
220.              }
221.          }
222.      }
223.      // 列
224.      for (int x = 1; x <= 9; x++) {
225.          for (int z = 1; z <= 9; z++) {
226.              for (int y = 1; y <= 8; y++) {
227.                  for (int i = y + 1; i <= 9; i++) {
228.                      fprintf(fp, "-%d -%d 0\n", (x - 1) * 81 +
    (y - 1) * 9 + z, (x - 1) * 81 + (i - 1) * 9 + z);
229.                  }
230.              }
231.          }
232.      }
233.      // 3x3子网格
234.      for (int z = 1; z <= 9; z++) {
235.          for (int i = 0; i <= 2; i++) {
236.              for (int j = 0; j <= 2; j++) {
237.                  for (int x = 1; x <= 3; x++) {
238.                      for (int y = 1; y <= 3; y++) {
239.                          for (int k = y + 1; k <= 3; k++) {
240.                              fprintf(fp, "-%d -%d 0\n", (3 * i
    + x - 1) * 81 + (3 * j + y - 1) * 9 + z, (3 * i + x - 1) * 81 +
    (3 * j + k - 1) * 9 + z);
241.                          }
242.                      }
243.                  }
244.              }
```

```
245.          }
246.      }
247.      for (int z = 1; z <= 9; z++) {
248.          for (int i = 0; i <= 2; i++) {
249.              for (int j = 0; j <= 2; j++) {
250.                  for (int x = 1; x <= 3; x++) {
251.                      for (int y = 1; y <= 3; y++) {
252.                          for (int k = x + 1; k <= 3; k++) {
253.                              for (int l = 1; l <= 3; l++) {
254.                                  fprintf(fp, "-%d -%d 0\n", (3
    * i + x - 1) * 81 + (3 * j + y - 1) * 9 + z, (3 * i + k - 1) *
    81 + (3 * j + l - 1) * 9 + z);
255.                              }
256.                          }
257.                      }
258.                  }
259.              }
260.          }
261.      }
262.      // 主对角线
263.      for (int z = 1; z <= 9; z++) {
264.          for (int i = 1; i <= 8; i++) {
265.              for (int j = i + 1; j <= 9; j++) {
266.                  fprintf(fp, "-%d -%d 0\n", (i - 1) * 81 + (i
    - 1) * 9 + z, (j - 1) * 81 + (j - 1) * 9 + z);
267.              }
268.          }
269.      }
270.      // 副对角线
271.      for (int z = 1; z <= 9; z++) {
272.          for (int i = 1; i <= 8; i++) {
273.              for (int j = i + 1; j <= 9; j++) {
274.                  fprintf(fp, "-%d -%d 0\n", (i - 1) * 81 + (9
    - i) * 9 + z, (j - 1) * 81 + (9 - j) * 9 + z);
275.              }
276.          }
277.      }
278.      fclose(fp);
279.  }
280.
281.
282.
283.  // 打印数独
284.  void print_sudoku(const vector<vector<int>>& sudoku) {
```

```
285.      for (const auto& row : sudoku) {
286.          for (int num : row) {
287.              cout << (num == 0 ? "." : to_string(num)) << " ";

288.          }
289.          cout << endl;
290.      }
291.  }
292.
293.  // 检查数字 num 是否可以放在(row, col)位置上
294.  bool is_safe(const vector<vector<int>>& sudoku, int row, int
   col, int num) {
295.      // 检查行
296.      for (int x = 0; x < N; x++) {
297.          if (sudoku[row][x] == num) return false;
298.      }
299.
300.      // 检查列
301.      for (int x = 0; x < N; x++) {
302.          if (sudoku[x][col] == num) return false;
303.      }
304.
305.      // 检查 3x3 子方格
306.      int start_row = row - row % 3;
307.      int start_col = col - col % 3;
308.      for (int i = 0; i < 3; i++) {
309.          for (int j = 0; j < 3; j++) {
310.              if (sudoku[i + start_row][j + start_col] == num)
   return false;
311.          }
312.      }
313.
314.      // 检查对角线
315.      if (row == col) {  // 主对角线
316.          for (int i = 0; i < N; i++) {
317.              if (sudoku[i][i] == num) return false;
318.          }
319.      }
320.      if (row + col == N - 1) {  // 副对角线
321.          for (int i = 0; i < N; i++) {
322.              if (sudoku[i][N - 1 - i] == num) return false;
323.          }
324.      }
325.
```

```
326.      return true;
327.  }
328.
329.  // 使用递归回溯算法填充数独
330.  bool fill_sudoku(vector<vector<int>>& sudoku, int row, int co
   l) {
331.      // 如果已经填满，返回 true
332.      if (row == N - 1 && col == N) return true;
333.
334.      // 如果列数超出，进入下一行
335.      if (col == N) {
336.          row++;
337.          col = 0;
338.      }
339.
340.      // 如果当前位置已经有值，递归下一个位置
341.      if (sudoku[row][col] != 0) return fill_sudoku(sudoku, row
   , col + 1);
342.
343.      // 尝试填充 1 到 9 的数字
344.      vector<int> numbers(N);
345.      iota(numbers.begin(), numbers.end(), 1);  // 生成 1 到 9 的数
   字
346.      random_shuffle(numbers.begin(), numbers.end());  // 随机打
   乱数字顺序
347.
348.      for (int num : numbers) {
349.          if (is_safe(sudoku, row, col, num)) {
350.              sudoku[row][col] = num;
351.
352.              // 递归填充下一个位置
353.              if (fill_sudoku(sudoku, row, col + 1)) return tru
   e;
354.
355.              // 回溯，如果填充失败，恢复 0
356.              sudoku[row][col] = 0;
357.          }
358.      }
359.
360.      return false;
361.  }
362.
363.  // 生成一个完整的 9x9 对角线数独
364.  void generate_sudoku(vector<vector<int>>& sudoku) {
```

```
365.      fill_sudoku(sudoku, 0, 0);
366.  }
367.
368.  // 挖洞法
369.  void remove_digits(const vector<vector<int>>& original_sudoku
   , vector<vector<int>>& modified_sudoku, int holes) {
370.      // 复制原始数独到修改后的数独
371.      modified_sudoku = original_sudoku;
372.
373.      // 生成所有位置的列表
374.      vector<int> positions;
375.      for (int i = 0; i < N; ++i) {
376.          for (int j = 0; j < N; ++j) {
377.              positions.push_back(i * N + j);
378.          }
379.      }
380.
381.      // 打乱位置列表
382.      random_shuffle(positions.begin(), positions.end());
383.
384.      // 挖洞
385.      for (int k = 0; k < holes; ++k) {
386.          int pos = positions[k];
387.          int row = pos / N;
388.          int col = pos % N;
389.
390.          modified_sudoku[row][col] = 0; // 将指定位置设为0，表示
   挖洞
391.      }
392.  }
393.  void useDiagdoku()
394.  //此函数用于使用蜂窝数独模块
395.  {
396.      int choice;
397.      choice = inputOrder(0, 4,
398.          "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n"
399.          "|| 选择功能:                  ||\n"
400.          "|| 1.生成数独                  ||\n"
401.          "|| 2.运用SAT求解器求解数独     ||\n"
402.          "|| 3.进行数独填充              ||\n"
403.          "|| 4.检验答案                  ||\n"
404.          "|| 0.退出                      ||\n"
405.          "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n"
406.      );
```

```
407.      // 初始化一个 9x9 空的数独
408.      vector<vector<int>> sudoku(N, vector<int>(N, 0));
409.      vector<vector<int>> modified_sudoku(N, vector<int>(N, 0))
   ;
410.      vector<vector<int>> newsudoku;
411.      int row, col, num;
412.      crossCNF* cnf = NULL;//待定的 cnf 对象
413.      FILE* file = fopen("Diagdoku.cnf", "r");
414.      FILE* file2 = NULL;
415.      char line1[10], s[2];
416.      int temp1, i, j, k;
417.      int ans[N][N], cnt = 1;
418.      while (choice != 0)
419.      {
420.          system("cls");
421.          switch (choice)
422.          {
423.          case 1:
424.              srand(time(0)); // 使用当前时间作为随机数种子
425.              // 生成对角线数独
426.              generate_sudoku(sudoku);
427.
428.              // // 挖洞前的数独
429.              // cout << "完整的数独:\n";
430.              // print_sudoku(sudoku);
431.
432.              // 挖洞操作，假设我们要挖 20 个洞
433.              int holes;
434.              cout << "请输入想要挖的洞的数目\n";
435.              cin >> holes;
436.              remove_digits(sudoku, modified_sudoku, holes);
437.              newsudoku = modified_sudoku;
438.              // 挖洞后的数独
439.              cout << "\n 生成的数独为:\n";
440.              print_sudoku(modified_sudoku);
441.
442.              // 生成数独 CNF
443.              toCnf(modified_sudoku, holes);
444.
445.              printf("已生成相应的 cnf 文件！\n");
446.              system("pause");
447.              break;
448.
449.          case 2:
```

```
450.                cnf = new crossCNF(file);//重新初始化
451.                cnf->solve("Diagdoku.res", "Diagdoku.cnf", true);

452.                file2 = fopen("Diagdoku.res", "r");
453.                // 读取第一行
454.                fgets(line1, sizeof(line1), file2);
455.                // 读取第二行头两个字符
456.                fgets(s, sizeof(s), file2);
457.                while (cnt <= 81)
458.                {
459.                    fscanf(file2, "%d ", &temp1);
460.                    if (temp1 > 0)
461.                    {
462.                        inverse_var(temp1, i, j, k);
463.                        ans[i - 1][j - 1] = k;
464.                        cnt++;
465.                    }
466.                }
467.                // 关闭文件
468.                fclose(file2);
469.                cout << "标准答案:\n";
470.                print_sudoku(sudoku);
471.                printf("SAT 求解答案: \n");
472.                for (int i = 0; i < N; i++)
473.                {
474.                    for (int j = 0; j < N; j++)
475.                        printf("%d ", ans[i][j]);
476.                    printf("\n");
477.                }
478.                printf("程序结束");
479.                break;
480.            case 3:
481.                newsudoku = modified_sudoku;
482.                cout << "当前数独如下" << endl;
483.                print_sudoku(newsudoku);
484.                cout << "请输入你想填入数独的位置（i，j），以及填入的数字
    num（只需输 1 个 0 退出填写）" << endl;
485.                cin >> row;
486.                while (row != 0)
487.                {
488.                    cin >> col >> num;
489.                    newsudoku[row - 1][col - 1] = num;
490.                    cout << "当前数独如下" << endl;
491.                    print_sudoku(newsudoku);
```

```
                cout << "继续填写则输入(i,j,num),不填写请输入0" <
< endl;
                cin >> row;
            }
            break;
        case 4:
            if (newsudoku == sudoku)
                cout << "你真是个天才！" << endl;
            else
                cout << "填写不对哦，请再试一次吧！";
            cout << "你填写的结果" << endl;
            print_sudoku(newsudoku);
            cout << "标准答案" << endl;
            print_sudoku(sudoku);
            break;

        default:
            break;
        }
        choice = inputOrder(0, 4,
            "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n"
            "|| 选择功能:                  ||\n"
            "|| 1.生成数独                  ||\n"
            "|| 2.运用SAT求解器求解数独     ||\n"
            "|| 3.进行数独填充              ||\n"
            "|| 4.检验答案                  ||\n"
            "|| 0.退出                      ||\n"
            "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n"
        );
    }
}


int main()
{
    int choice;
    while (1)
    {
        system("cls");
        choice = inputOrder(0, 2,
            "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
            "|| 选择功能:              ||\n"
```

```
535.                "|| 1.SAT求解  模块       ||\n"
536.                "|| 2.Diagdoku 模块       ||\n"
537.                "|| 0.退出                 ||\n"
538.                "||^^^^^^^^^^^^^^^^^^^^^^^^||\n"
539.            );
540.
541.            switch (choice)
542.            {
543.            case 0: {
544.                return 0;//此时退出就是结束整个程序
545.            }
546.            case 1: {
547.                useDPLL();
548.                break;
549.            }
550.
551.            case 2: {
552.                useDiagdoku();
553.                break;
554.            }
555.            }
556.
557.        }
558.
559.
560.  }
```

solver.cpp

```
1.  //这个文件是对 solver.h 中 CNF 求解器部分的实现
2.  #define _CRT_SECURE_NO_WARNINGS
3.  #include<cstdlib>
4.  #include<cstdio>
5.  #include<ctime>
6.  #include<new>
7.  #include<cassert>
8.  #include "solver.h"
9.  #include "cdcl.hpp"
10. using namespace std;
11.
12. int random = 0;//如果 random==0，会使用“倾向”求解策略，否则会使用“随机”求解策略
13. //倾向：在分类讨论布尔变元 L 为真还是为假时，如果一开始的 cnf 中+L 的出现次数比-L 多则先讨论真，否则先讨论假
```

```
14. //随机：在分类讨论布尔变元 L 为真还是为假时，随机先讨论真还是先讨论假
15. int my_choice;
16. bool crossCNF::calculate(const char* const filename)
17. //验证 cnf 和 res 文件是否匹配
18. {
19. 
20.     int* ans = new int[boolNum + 1];//res 文件的内容会被写入 ans 再与 cnf 进行比较
21.     int i, n;
22. 
23.     FILE* fp = fopen(filename, "r");
24.     if (fp == NULL)
25.     {
26.         printf("文件打开失败，不能检验\n");
27.         delete[] ans;
28.         return 0;
29.     }
30.     fscanf(fp, "%*s%d%*s", &n);
31.     if (n == 0)
32.     {
33.         printf("无解，不可验证\n");
34.         return 0;
35.     }
36. 
37.     printf("res=[");
38.     for (i = 0; i < boolNum; i++)
39.     {
40.         fscanf(fp, "%d", &n);
41.         if (n < 0)ans[-n] = 0;
42.         else ans[n] = 1;
43.         printf("%d ", n);
44.     }
45.     bool ok;
46.     crossNode* p;
47.     printf("]\n 检验开始\n");
48.     for (i = 1; i <= clauseNum; i++)//在每个子句中找到第一个为 True 的文字，找到了，这个子句就算 True
49.     {
50.         ok = false;
51.         p = clauses[i].right;
52.         while (p)
53.         {
54.             n = p->Bool;
55.             printf("%d ", n);
```

```
56.             if (n < 0 && ans[-n] == 0 || n>0 && ans[n] == 1)
57.             {
58.                 printf("True");
59.                 ok = true;
60.                 break;
61.             }
62.
63.
64.             p = p->right;
65.         }
66.         printf("\n");
67.         if (!ok)
68.         {
69.             printf("clause:False");//如果发现哪个子句为False，则整个cnf也就为False
70.             fclose(fp);
71.             delete[] ans;
72.             return false;
73.         }
74.     }
75.
76.
77.     delete[] ans;
78.     fclose(fp);
79.     return true;
80. }
81.
82. bool crossCNF::solve(const char* const filename, bool display)
83. //名义解，主要工作是为innerSolve函数创造环境，把结果写成文件什么的
84. {
85.
86.     FILE* fp = NULL;
87.     if (filename)fp = fopen(filename, "w");
88.     if (random)printf("当前求解策略：随机\n");
89.     else printf("当前求解策略：倾向\n");
90.
91.
92.     int start_time = clock();//开始计时
93.     bool ans = innerSolve();
94.     int delta_time = clock() - start_time;//结束计时
95.
96.     if (ans)
97.     {
98.         //有解
```

```
99.          if (display)printf("有解 耗时%dms\n", delta_time);
100.            hypo[0] = TRUE;
101.            if (fp)
102.            {
103.                fprintf(fp, "s 1\nv ");
104.
105.                for (int i = 1; i <= boolNum; i++)
106.                {
107.                    if (hypo[i] == TRUE)fprintf(fp, "%d ", i);
108.                    else if (hypo[i] == FALSE)fprintf(fp, "-%d ",
    i);
109.                    else fprintf(fp, "-%d ", i);
110.                }
111.
112.                fprintf(fp, "\nt %d", delta_time);
113.            }
114.
115.
116.        }
117.        else
118.        {
119.            //无解
120.            if (display)printf("无解 耗时%d\n", delta_time);
121.            hypo[0] = FALSE;
122.            if (fp)fprintf(fp, "s 0\n");
123.
124.        }
125.        fclose(fp);
126.
127.
128.
129.        return ans;
130.
131.    }
132.
133.    bool crossCNF::solve(const char* const filename, const char*
    const filename0, bool display)//filename 为输出指定文件
134.    //名义解，主要工作是调用 innerSolve 函数求解，并输出结果到指定文件
135.    {
136.
137.        FILE* fp = NULL;
138.        if (filename)fp = fopen(filename, "w");
139.        printf("请选择你的求解策略\n"
140.            "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n"
```

```
        "||  （1）.变元随机选取策略              || \n"
        "||  （2）.最短子句出现频率最大优先策略    ||\n"
        "||  （3）.cdcl 冲突驱动的子句学习策略      ||\n"
        "||^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^||\n");
    scanf("%d", &my_choice);
    bool ans;
    cdcl::Solver2 solver2;
    int start_time = clock();//开始计时
    printf("正在求解，请耐心等待……\n");
    switch (my_choice)
    {
    case 1:
    case 2:
        ans = innerSolve();
        break;
    case 3:
        solver2.parseDimacsProblem(filename0);
        ans = solver2.solve();
        if (ans)
            ans = 0;
        else if (ans == 0)
        {
            ans = 1;
            for (int i = 0; i < solver2.assigns.size(); i++)
            {
                if (solver2.assigns[i] == 0) {
                    hypo[i + 1] = TRUE;
                }
                else {
                    hypo[i + 1] = FALSE;
                }
            }
        }
    default:
        break;
    }
    int delta_time = clock() - start_time;//结束计时

    if (ans)
    {
        //有解
        if (display)printf("有解 耗时%dms\n", delta_time);
        hypo[0] = TRUE;
        if (fp)
```

```
185.          {
186.              fprintf(fp, "s 1\nv ");
187.
188.              for (int i = 1; i <= boolNum; i++)
189.              {
190.                  if (hypo[i] == TRUE)fprintf(fp, "%d ", i);
191.                  else if (hypo[i] == FALSE)fprintf(fp, "-%d ",
    i);
192.                  else fprintf(fp, "-%d ", i);
193.              }
194.
195.              fprintf(fp, "\nt %d", delta_time);
196.          }
197.
198.
199.      }
200.      else
201.      {
202.          //无解
203.          if (display)printf("无解 耗时%dms\n", delta_time);
204.          hypo[0] = FALSE;
205.          if (fp) {
206.              fprintf(fp, "s 0\n");
207.              fprintf(fp, "\nt %d", delta_time);
208.          }
209.
210.      }
211.      system("pause");
212.      fclose(fp);
213.
214.
215.
216.      return ans;
217.
218.  }
219.
220.  int crossCNF::believe(int L)
221.  //相信 L==true，这个函数会和 innerSolve 相互调用
222.  {
223.
224.      //在 hypothesis 中注册结果
225.      if (L > 0)hypo[L] = TRUE;
226.      else hypo[-L] = FALSE;//将负数转化成对应的正数 bool 变元编号，让
    它为假，即-L 为真（临时 result）
```

```
227.      //设置“被删除”栈
228.      crossNode* DeleteNodesHead = NULL;//被删除的bool变元栈
229.      //设置临时指针变量
230.      crossNode* p, * q;
231.      int i, target;
232.      int startNum = remainClauseNum;
233.
234.      //删去所有含L的子句
235.      for (q = bools[changeBool(L)].down; q; q = q->down)//正数
    不动，负数化为对应的大正数，顺着向下的指针删
236.      {
237.
238.          if (q->del)continue;
239.          target = q->Clause;
240.          sum[target]++;
241.          remainClauseNum--;
242.          for (p = &clauses[target]; p; p = p->right)
243.          {
244.              if (p->del)continue;
245.              p->del = true;       // 已经被删除了
246.
247.              //把p加入DeleteNodes栈
248.              p->next = DeleteNodesHead;
249.              DeleteNodesHead = p;
250.              sum[target]--;
251.          }
252.      }
253.
254.      //如果导致剩余子句数量为0 ,就认为这是真解
255.      if (remainClauseNum == 0)return 1;
256.      //需要删去所有子句中的-L
257.      for (p = bools[changeBool(-L)].down; p; p = p->down)
258.      {
259.          if (p->del)continue;
260.          p->del = true;
261.          p->next = DeleteNodesHead;
262.          DeleteNodesHead = p;
263.          sum[p->Clause]--;
264.          // 删除多了怎么办
265.          //可以在这里检查空子句
266.          if (sum[p->Clause] == 0)
267.          {
268.              restore(L, startNum, DeleteNodesHead);
269.              return 0;
```

```
270.           }
271.           if (sum[p->Clause] == 1)single.push(p->Clause);//如果
    删出了单子句，就把它加到“单子句”栈中去
272.       }
273.       if (innerSolve())return 1;
274.       restore(L, startNum, DeleteNodesHead);
275.       return 0;
276.   }
277.
278.
279.   void crossCNF::restore(int L, int startNum, crossNode* Head)
280.   //把believe中做出的修改复原
281.   {
282.
283.       hypo[abs(L)] = UNSURE;
284.       while (Head)//复原一些结点
285.       {
286.           Head->del = false;
287.           sum[Head->Clause]++;
288.           Head = Head->next;
289.       }
290.       remainClauseNum = startNum;
291.   }
292.
293.   int crossCNF::innerSolve()
294.   //实际解，会和believe相互调用
295.   {
296.       int i, L;
297.       crossNode* p = NULL;
298.       while (!single.empty())//试图从“单子句”栈中获得单子句
299.       {
300.           L = single.pop();
301.           if (!clauses[L].del && sum[L] == 1)//找到单子句（可能不
    在子句链表开头，需要查找）
302.           {
303.               p = clauses[L].right;
304.               while (p->del)p = p->right;
305.               return believe(p->Bool);//返回这个单子句中的bool变元
    序号
306.           }
307.       }
308.       long long *cnt = new long long[boolNum + 10];
309.       long long max = 0, index = 0;
310.       long long min = boolNum + 1;
```

```
311.      switch (my_choice)
312.      {
313.      case 1: //选取“第一行第一个” 文字进行分类讨论 ， 这个策略是最普
      通的
314.          //第一文字选取策略
315.          for (i = 1; i <= clauseNum; i++)
316.          {
317.              if (sum[i] == 0) continue;   // 还剩余的文字为 0
318.              p = clauses[i].right;
319.              while (p && p->del)p = p->right;
320.              break;
321.          }
322.          L = abs(p->Bool);    // 转化为正数
323.          break;
324.      case 2: //最短子句出现频率最大优先策略(MOM)
325.          for (int i = 1; i <= boolNum; i++)
326.              cnt[i] = 0;
327.          for (i = 1; i <= clauseNum; i++)
328.          {
329.              if (sum[i] != 0)
330.              {
331.                  if (sum[i] < min)
332.                      min = sum[i];
333.              }
334.          }
335.          for (i = 1; i <= clauseNum; i++)
336.          {
337.              if (sum[i] == min)
338.              {
339.                  p = clauses[i].right;
340.                  while (p)
341.                  {
342.                      if (!p->del)
343.                          cnt[abs(p->Bool)]++;
344.                      p = p->right;
345.                  }
346.              }
347.          }
348.          for (int i = 1; i <= boolNum; i++)
349.          {
350.              if (cnt[i] > max)
351.              {
352.                  max = cnt[i];
353.                  index = i;
```

```
354.                 }
355.             }
356.             L = index;
357.             break;
358.         default:
359.             break;
360.         }
361.
362.
363.         //开始分类讨论，首先要决定先考虑正的还是先考虑负的
364.         if (random)L *= 2 * (rand() % 2) - 1;//随机化选取策略
365.         else L *= tendency[L];//倾向选取策略
366.
367.         if (believe(L))return 1;
368.         return believe(-L);
369.
370.
371.  }
```

cnfparser.cpp

```
1.  #define _CRT_SECURE_NO_WARNINGS
2.  #include<new>
3.  #include "solver.h"
4.  int crossCNF::changeBool(int x)
5.  //把负数翻译成一种比较大的正数的函数，以便其他函数处理负数文字
6.  {
7.      return (x > 0) ? x : (2 * boolNum + 1 + x);
8.  }
9.
10. crossCNF::crossCNF(FILE* fp)//构造函数，可根据 fp 构造 CNF 对象
11. {
12.     char get;
13.
14.     //跳过注释:
15.     while (1)
16.     {
17.         get = fgetc(fp);
18.         if (get == 'c')while (fgetc(fp) != '\n'); // 跳过注释
19.         else break;
20.     }
21.
22.     //理论上，现在 get=='p'
23.     //取元数据
```

```
24.     fscanf(fp, "%*s%d%d", &boolNum, &clauseNum);
25.
26.     remainClauseNum = clauseNum;
27.
28.
29.     //开辟数据空间
30.     clauses = new crossNode[clauseNum + 1];//头链表
31.     bools = new crossNode[2 * boolNum + 2];//节点块
32.     hypo = new BOOLVALUE[boolNum + 1];//每个节点的假设
33.     sum = new int[clauseNum + 1];//存储每个子句的剩余变元数
34.
35.
36.     int i, nowClause = 1, temp;
37.     for (i = 0; i <= clauseNum; i++)clauses[i].Bool = 0;
38.     for (i = 0; i <= boolNum; i++)hypo[i] = UNSURE;
39.     for (i = 0; i < 2 * boolNum + 2; i++)bools[i].Bool = 0;
40.     for (i = 0; i <= clauseNum; i++)sum[i] = 0;
41.     //逐个存储
42.     while (nowClause <= clauseNum)
43.     {
44.         fscanf(fp, "%d", &temp);
45.         if (temp == 0)nowClause++;
46.         else addNode(nowClause, temp);// 在此处添加十字链表结点
47.     }
48.     fclose(fp);
49.
50.
51.
52.
53.     //初始化“单子句”栈
54.     for (i = 1; i <= clauseNum; i++)if (sum[i] == 1)single.push
   (i);//逐个检查子句，将单子句序号纳入栈中
55.
56.     //初始化“倾向”
57.     tendency = new int[boolNum + 1];//创建所有布尔变元的“倾向”数组
58.     int posi, nega;
59.     crossNode* p;
60.     for (i = 1; i <= boolNum; i++)
61.     {
62.         posi = 0;
63.         nega = 0;
64.         // 记录正数出现多还是负数出现多
65.         for (p = bools[i].down; p; p = p->down)posi++;
```

```
66.         for (p = bools[changeBool(-i)].down; p; p = p->down)nega++;
67.         if (posi >= nega)tendency[i] = 1;
68.         else tendency[i] = -1;
69.     }
70.
71.     return;
72. }
73.
74.
75. crossCNF::crossCNF(const char* const filename)//构造函数，根据文件名构造 CNF 对象
76. {
77.     FILE* fp = fopen(filename, "r");
78.     new (this)crossCNF(fp);
79.     fclose(fp);
80. }
81.
82. crossCNF::~crossCNF()//析构函数
83. {
84.     crossNode* p, * q;
85.     int i;
86.     //逐行删除
87.     for (i = 1; i <= clauseNum; i++)
88.     {
89.         p = clauses[i].right;
90.         while (p)
91.         {
92.             q = p->right;
93.             delete p;
94.             p = q;
95.         }
96.     }
97.     delete[] bools, clauses, hypo, sum, tendency;//删去手动分配的数组空间
98. }
99.
100.
101.  void crossCNF::print(FILE* fp = stdout)//依次打印目前 cnf 十字链表中的未被删除的内容
102.  {
103.      crossNode* p;
104.      for (int i = 1; i <= clauseNum; i++)
105.      {
```

```
106.          if (clauses[i].del)continue;
107.          fprintf(fp, "%d:", i);
108.          for (p = clauses[i].right; p != NULL; p = p->right)
109.          {
110.              if (p->del)continue;
111.              fprintf(fp, "%d ", p->Bool);
112.          }
113.          fprintf(fp, "\n");
114.      }
115.  }
116.
117.
118.
119.  void crossCNF::addNode(int Clause, int Bool)
120.  {
121.      crossNode* p = new crossNode(), * p1, * p2;
122.      p->Bool = Bool;
123.      p->Clause = Clause;
124.      Bool = changeBool(Bool);//正负数坐标转化为物理坐标
125.      p1 = &clauses[Clause];
126.      p2 = &bools[Bool];
127.
128.      while (p1->right != NULL && changeBool(p1->right->Bool) <
    Bool)p1 = p1->right;
129.      while (p2->down != NULL && p2->down->Clause < Clause)p2 =
    p2->down;
130.      if (p1->right)p1->right->left = p;
131.      p->right = p1->right;
132.      p1->right = p;
133.      p->left = p1;
134.
135.      if (p2->down)p2->down->up = p;
136.      p->down = p2->down;
137.      p2->down = p;
138.      p->up = p2;
139.
140.      sum[Clause]++;
141.
142.  }
```