

华中科技大学

课程设计报告

题目： 基于 SAT 的数独游戏求解程序

课程名称： 综合程序设计课程设计

专业班级： CS1703

学 号： U201714607

姓 名： 钟子琛

指导教师： 袁凌

报告日期： 2019.3.27

计算机科学与技术学

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： $[(t-t_0)/t]*100\%$ ，其中 t 为未对 DPLL 优化时求解基准算例的执行时间， t_0 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问题求解，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%)

参考文献

- [1] 张健著. 逻辑公式的可满足性判定—方法、工具及应用. 科学出版社, 2000
- [2] Tanbir Ahmed. An Implementation of the DPLL Algorithm. Master thesis, Concordia University, Canada, 2009
- [3] 陈稳. 基于 DPLL 的 SAT 算法的研究与应用. 硕士学位论文, 电子科技大学, 2011
- [4] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243
- [5] 360 百科: 数独游戏 <https://baike.so.com/doc/3390505-3569059.html>
- [6] Tjark Weber. A sat-based sudoku solver. In 12th International Conference on Logic 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] Robert Ganian and Stefan Szeider. Community Structure Inspired Algorithms for SAT and #SAT. International Conference on Theory and Applications of Satisfiability Testing (SAT 2015), 223–237 360

目 录

任务书	I
1 引言	1
1.1 课题背景与意义	1
1.2 国内外研究现状	1
1.3 课程设计的主要研究工作	1
2 系统需求分析与总体设计	2
2.1 系统需求分析	2
2.2 系统总体设计	2
3 系统详细设计	3
3.1 有关数据结构的定义	3
3.2 主要算法设计	5
4 系统实现与测试	7
4.1 系统实现	7
4.2 系统测试	10
5 总结与展望	23
5.1 全文总结	23
5.2 工作展望	23
参考文献	25
附录	26

1 引言

1.1 课题背景与意义

可满足性问题 (Satisfiability Problem) 即 SAT 问题, 是对一个以合取范式 (Conjunctive Normal Form, 常简称 CNF) 的形式给出的命题逻辑公式进行判断, 以找出是否存在一个真值指派, 使得该命题逻辑公式的值为真。SAT 问题看似简单, 但它却是计算机领域和人工智能领域所要研究的中心问题, 被称为理论计算机科学和数理逻辑中的第一问题, 在硬件验证、人工智能、电子设计自动化、自动化推理、组合等式检测等领域具有非常重要的理论和实践意义。

1.2 国内外研究现状

从 1960 年至今, SAT 问题一直备受人们的关注, 世界各国的研究人员在这方面都做了大量的工作, 提出了许多求解算法。每年可满足性理论和应用方面的国际会议都会组织一次 SAT 竞赛以求找到一组最快的 SAT 求解器, 而且会详细展示一系列的高效求解器的性能。2003 年的 SAT 竞赛中, 就有 30 多种解决方案针对从成千上万的基准问题中挑选出的一些 SAT 问题实例同台竞争。国内也经常组织一些 SAT 竞赛及研讨会, 这些都促进了 SAT 算法的飞速发展。尽管命题逻辑的可满足性问题理论研究已趋于成熟, 但在 SAT 求解器被越来越多地应用到各种实际问题领域的今天, 探寻解决 SAT 问题的高效算法仍然是一个吸引人并且极具挑战性的研究方向。

1.3 课程设计的主要研究工作

- (1) 对 SAT 问题的研究背景、意义及现状进行了简要总结, 学习了命题逻辑可满足性问题的基本理论知识。
- (2) 基于 DPLL 算法, 研究了关于 SAT 问题的求解相关的知识, 并自编写了一个 SAT 问题的求解器。
- (3) 将 SAT 问题应用到实际问题中, 本文中 choice 的是数独问题, 并编写了运用 SAT 求解器来解决数独问题的程序。

2 系统需求分析与总体设计

2.1 系统需求分析

能够读取已有的 SAT 算例 cnf 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。

在此基础上，还可以使用该系统随机生成数独问题，并将数独问题转换为 SAT 问题，再由 SAT 求解器解出最后的结果。

2.2 系统总体设计

该系统主要功能是，读取 cnf 文件，输出 cnf 文件的解，能够求解数独，因此包含了一下几个模块。

- (1) 允许用户使用系统读取 cnf 文件。
- (2) 运用基于 DPLL 算法的 SAT 求解器，得到该 SAT 问题的解。
- (3) 以创建 res 文件的方式，输出该 cnf 文件的解。
- (4) 能够随机生成一个有解的数独。首先创建一个完整的数独，然后在该数独的基础上挖空，从而形成一个新的数独问题。
- (5) 将该数独问题转化为 SAT 问题。
- (6) 运用板块 1 中的 SAT 求解器，输出创建的 SAT 问题的解，再将解转化为数独的解。

完整系统模块结构图如图 2-1 所示。

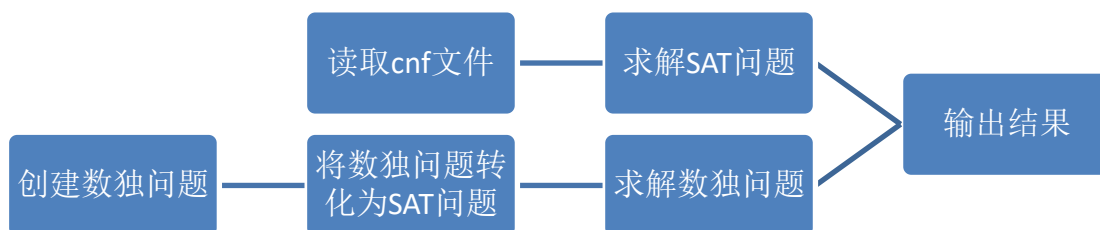


图 2-1 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

每种数据结构的具体定义如下：

Formula:

sta: 标识公式是否可满足，0为不满足，1为满足，-1为待定

num_st: 表示子句数

num_v: 变元数

Statement* root: 指向第一个子句

Statement:

num_lit: 文字数

elem: 指向子句中的第一个文字

next: 指向下一个子句

Literal:

pos: 表示文字状态 1代表正文字 0代表负文字

num: 表示文字是第几号变元

next: 指向下一个文字

Stack_F:

V: 储存公式中确定的变元

F: 栈中储存的公式

next: 下一个节点

hole:

x: 行

y: 列

dig_num: 可能的数字数

elem: 指向第一个可能的数字

digital:

num: 数字

v_num: 数字所对应的变元编号

next: 下一个数字

总结如表3-1所示：

数据结构	说明
Formula	公式，表示CNF文件中保存的CNF公式
Statement	子句，表示公式中的子句
Literal	表示每一个子句中的变元文字
Stack_F	用来储存公式的栈
hole	数独中挖出来的洞
digital	数独中每一个格子内存放的数字

表3-1

各个数据结构之间的关系：

Formula保存了所有的Statement，每一个Statement由若干Literal组成。Stack_F是在关于Formula的函数中起到储存Formula的作用。hole是数独的基本单位，每一个hole中存在若干个可能填在该空格内的数字。

数据结构之间的关系如图3-1所示。

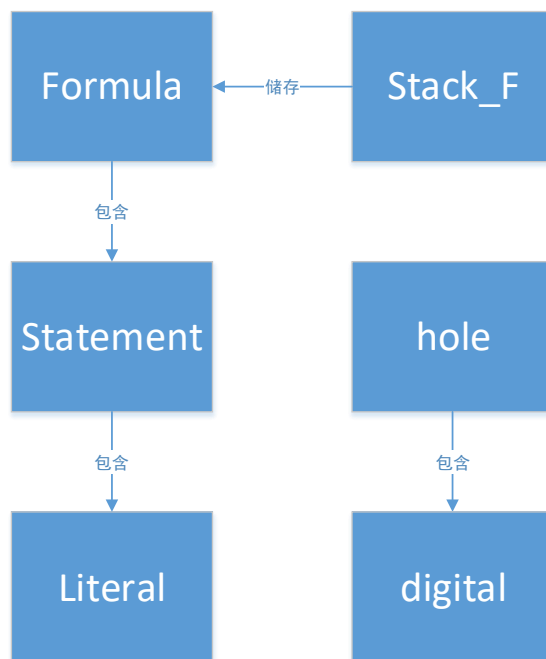


图3-1

3.2 主要算法设计

这部分主要描述系统中的模块实现的流程,可采用文字配合流程图的方式表示各模块的算法思想及流程。

程序主要分为两大模块:

(1) SAT

该模块能够读取cnf文件生成cnf范式并求解cnf范式。

在求解cnf范式的过程中主要运用的是DPLL算法:

DPLL 算法是一种基于树的回溯算法,主要使用两种基本处理策略:

a.单子句规则。如果子句集 S 中有一个单子句 L ,那么 L 一定取真值,于是可以从 S 中删除所有包含 L 的子句(包括单子句本身),得到子句集 S_1 ,如果它是空集,则 S 可满足。否则对 S_1 中的每个子句,如果它包含文字 $\neg L$,则从该子句中去掉这个文字,这样可得到子句集合 S_2 。 S 可满足当且仅当 S_2 可满足。单子句传播策略就是反复利用单子句规则化简 S 的过程。

b.分裂策略。按某种策略选取一个文字 L 。如果 L 取真值,则根据单子句传播策略,可将 S 化成 S_2 ;若 L 取假值(即 $\neg L$ 成立)时, S 可化成 S_1 。

根据上述规则可不断对公式化简,并最终达到终止状态,其执行过程可表示为一棵二叉搜索树,如下图 3-2 所示。

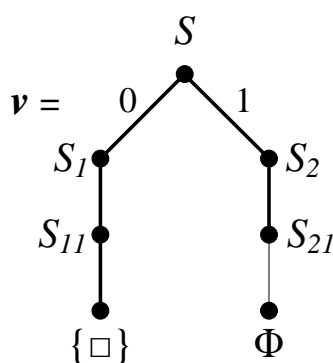


图 3-2

考虑到执行效率的问题,使用循环的方法而非递归。

运用这个算法最终将cnf范式解出来。

(2) Sudoku

该模块能够创建一个数独问题。

创建数独的算法:首先使用一个二维数组来储存这个数组,然后使用random

函数来生成一个随机的已解数独，再通过随机挖洞的方式来创建这个数独游戏。

该模块还能够将这个数独问题转化为cnf范式，转化为cnf文件，这样我们就把数独问题转化成了SAT问题，由此我们能够使用模块一通过DPLL算法得到该SAT问题的解，最终得到这个数独的解。

大致流程图如图3-3所示：

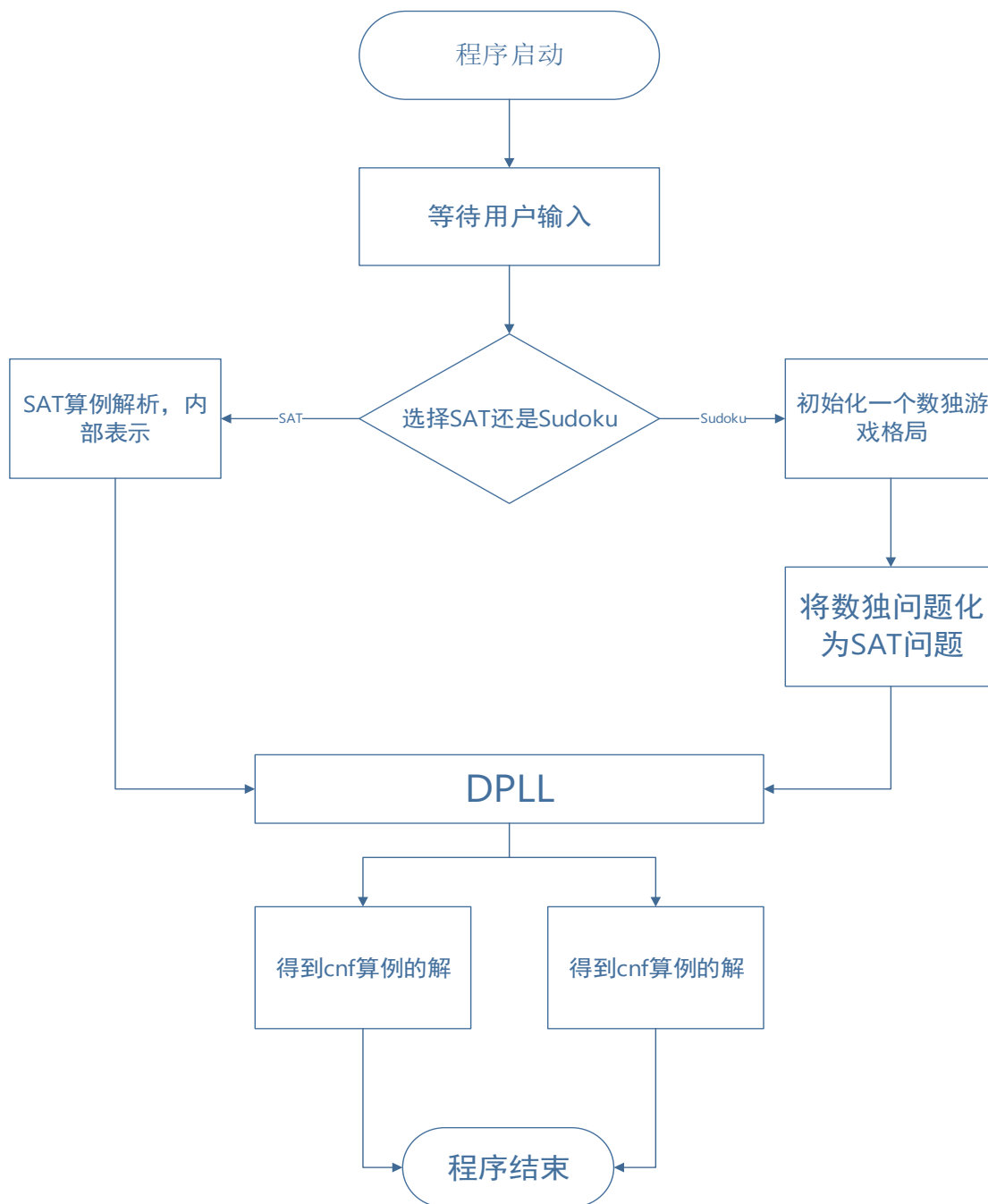


图3-3

4 系统实现与测试

4.1 系统实现

运行环境：Windows10、16G 运行内存、Inter(R) Core(TM) i5-7300HQ

数据结构定义：

```
typedef struct Literal
{
    int pos;//表示文字状态 1代表正文字 0代表负文字
    int num;//表示文字是第几号变元
    struct Literal* next;//指向下一个文字
}Literal;

typedef struct Statement
{
    int num_lit;//文字数
    struct Literal* elem;//指向子句中的第一个文字
    struct Statement* next;//指向下一个子句
}Statement;

typedef struct Formula
{
    int sta;//标识公式是够可满足，0为不满足，1为满足，-1为待定
    int num_st;//表示子句数
    int num_v;//变元数
    struct Statement* root;//指向公式中的第一个子句
}Formula;

typedef struct Stack_F
{
    int* V;//储存已确定信息的变元
    struct Formula* S;//表示栈中存储的公式
    struct Stack_F* next;//用于指向下一个节点
}Stack_F;
```

```
typedef struct hole
{
    int x;//标识行
    int y;//标识列
    int dig_num;//标识可能的数字数
    struct digital* elem;//指向第一个可能的数字
}hole;

typedef struct digital
{
    int num;//数字
    int v_num;//数字所对应的变元编号
    struct digital* next;
}digital;
```

所运用的函数：

主控、交互与显示模块（display）：

1、main()

首先创建变量time_start和time_end，用来记录程序运行的时间。然后让用户选择是进行cnf求解还是生成数独并求解。

若选择cnf求解，则调用SAT函数。若选择生成数独并求解，则调用Sudoku函数。

CNF解析模块（cnfparser）以及核心DPLL模块（solver）：

2、void SAT()

该函数能够读取cnf文件生成cnf范式并求解cnf范式。

- (1) 让用户输入文件路径，按照路径打开对应的cnf文件，打开文件成功后调用CNF_Reader函数，读取对应的cnf文件并由此生成cnf范式。
- (2) 读取文件结束后，首先对应文件中的每个变元创建一个数组Varies[i]，用来储存它们的解。之后开始计时并调用SAT_Solver中的Solver函数，该方法能够通过传递的cnf范式将对应的解解出来并储存到Varies[i]数组中。

- (3) 最后, 经过Check函数的检测, 依据Varies[i]数组中储存的内容, 生成.res文件。

3、void CNF_Reader(FILE* fp, Formula* formula)

通过该函数能够读取cnf文件来生成cnf范式。

- (1) 若每一行开头为c, 则判断该行为注释, 忽略该行。
- (2) 若某一行开头为cnf, 则说明之后为cnf公式, 将之后的变量数和子句数保存到formula->num_v以及formula->num_st中。
- (3) 以Statement数据结构创建储存子句的首节点, 赋值到传入参数的formula的root中, 之后在子句中每读取完一个子句, 就将该子句储存到Statement链表中。
- (4) 每读取到一个子句, 创建一个Literal首节点, 赋值给Statement->elem, 用来储存子句中的每一个字。
- (5) 当读取到最后一个字后, 该cnf文件读取完毕, 返回到SAT函数中。

4、status Solver(Formula* formula, int* Varies)

该函数能够通过传入的 cnf 范式得到该范式的解。主要是运用 DPLL 算法, 但是为了优化执行效率将递归换成了循环。采取的策略为: 选出长度最短的子句, 在子句中选择出现次数最多的变元。

5、bool Check(int* Varies_res, Formula* formula)

该函数能够检测得到的 cnf 算例的解是否正确。在传输的参数 Varies_res 中存储着该 cnf 算例的解, 从 formula 中开始以此检测每一个子句, 如果每一个子句中都有一个文字满足, 那么说明结果正确, 否则, 结果错误, 重新读取 cnf 文件, 重新求解。

数独模块, 包括数独生成、归约、求解(Sudoku):

6、void Sudoku()

该函数主要用来处理数独游戏的问题。

- (1) 首先调用 CreateFinalSud 函数生成完全数独格局, 然后再调用 CreateSudoku 来对完全数独格局挖空, 从而生成一个数独游戏。
- (2) 生成数独游戏之后将它输出到控制台上。
- (3) 调用 TransfSud 函数, 将这个数独游戏化为 cnf 范式。

(4) 将生成的 cnf 范式输出为 cnf 文件，通过之前 SAT 函数进行求解，得到这个数独游戏的解。

(5) 将数独游戏的解化为可视化的解输出到控制台上。

7、void CreateFinalSud(int(*sud)[9])

该函数能够创建一个完整的数独格局储存到传入的数组中。

8、int CreateSudoku(int(*sud)[9])

该函数对之前创建的完全数独格局进行挖空，从而生成一个完整的数独游戏。

9、int* TransfSud(int(*sud)[9], Formula*S, hole* holes)

该函数将生成的数独游戏问题转化为 cnf 范式。算法：根据每个空格内有且仅有一个数字、1~9 中的数字在每行、每列和每个宫内出现且仅出现一次创建子句，通过邻接表记录每个空格中可能出现的数字。具体步骤：

(1) 生成空格及其可能的数字的邻接表。

(2) 空格内必须有一个数字，由此创建子句。

(3) 1~9 每个数字在每行最多出现一次，最少出现一次，由此创建子句。

(4) 1~9 每个数字在每列最多出现一次，最少出现一次，由此创建子句。

(5) 1~9 每个数字在每个小九宫格内最多出现一次，最少出现一次，由此创建子句。

(6) 将创建的所有子句组合成 cnf 范式。

10、void ShowSudResult(int* Varies, hole* holes, int(*sud)[9])

在由 SAT_Solver 将这个数独生成的 cnf 范式解出来之后，得到的解储存在 Varies 中，该函数能够将 Varies 中的解反映到数独中，并将解出来的数独输出到控制台。

程序能够实现的所有功能到此结束。程序具体代码见附录。

4.2 系统测试

常用软件测试方法：

(1) 静态测试：指测试不运行的部分，例如测试产品说明书，对此进行检查和审阅。静态方法是指不运行被测程序本身，仅通过分析或检查源程序的文法、结构、过程、接口等来检查程序的正确性。

- (2) 动态测试：是指通过运行软件来检验软件的动态行为和运行结果的正确性。
- (3) 单元测试：是最微小规模的测试;以测试某个功能或代码块。
- (4) 集成测试：是指一个应用系统的各个部件的联合测试，以决定它们能否在一起共同工作并没有冲突。
- (5) 系统测试：是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不相符合或与之矛盾的地方。
- (6) 性能测试：是在交替进行负荷和强迫测试时常用的术语。理想的“性能测试”(和其他类型的测试)应在需求文档或质量保证、测试计划中定义。性能测试一般包括负载测试和压力测试。
- (7) 随机测试：没有书面测试用例、记录期望结果、检查列表、脚本或指令的测试。主要是根据测试者的经验对软件进行功能和性能抽查。随机测试是根据测试说明书执行用例测试的重要补充手段，是保证测试覆盖完整性的有效方式和过程。
- (8)

对于本程序，主要使用动态测试、集成测试、性能测试和随机测试。

SAT测试方法：

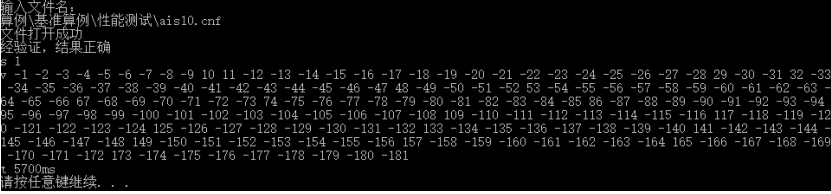
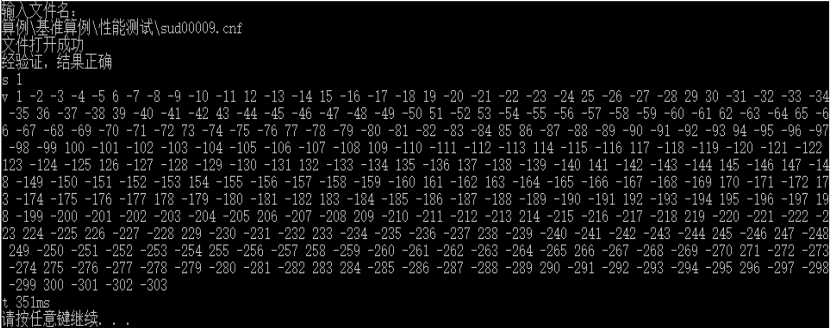
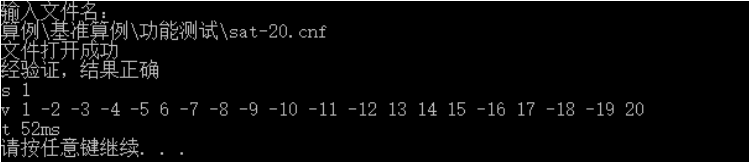
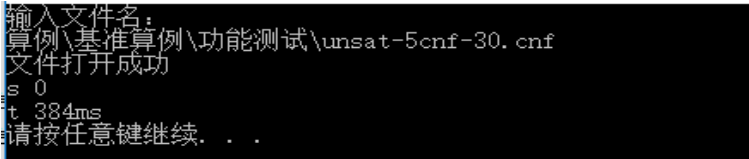
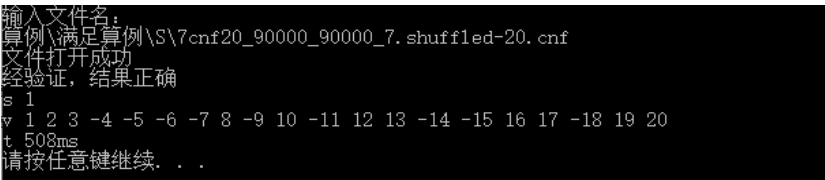
- (1) 让程序读取基准算例中的功能测试，检测程序的基础功能是否正确。
- (2) 基准算例中的性能测试计算运行时间，除此之外，再使用未采用优化策略时的程序来读取这些算例，通过对比，得到程序的优化率。
- (3) 随机抽取20个满足算例中的S规模以及M规模算例，5个不满足算例，进行测试。

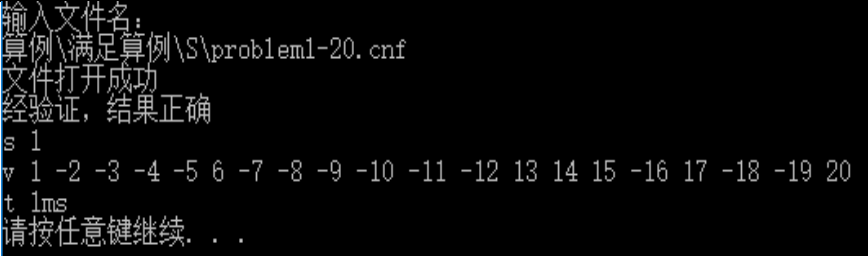
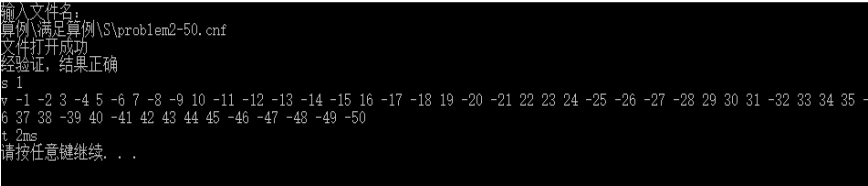
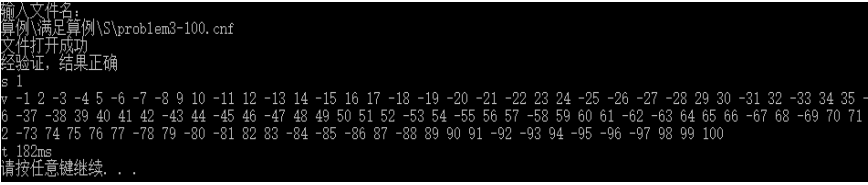
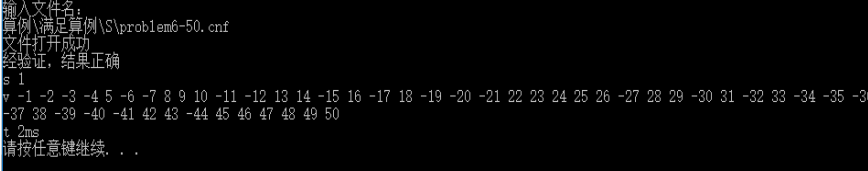
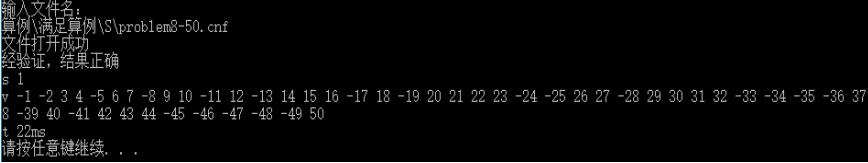
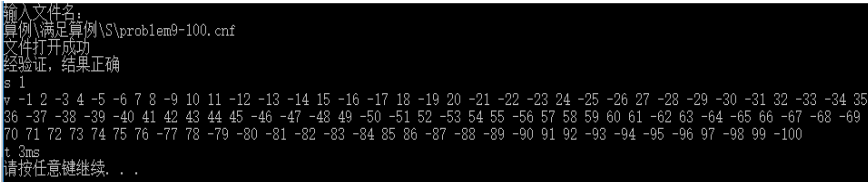

Sudoku测试方法：

运行Sudoku模块，随机生成5组有20空格的数独，检查结果的正确性，记录运行时间。

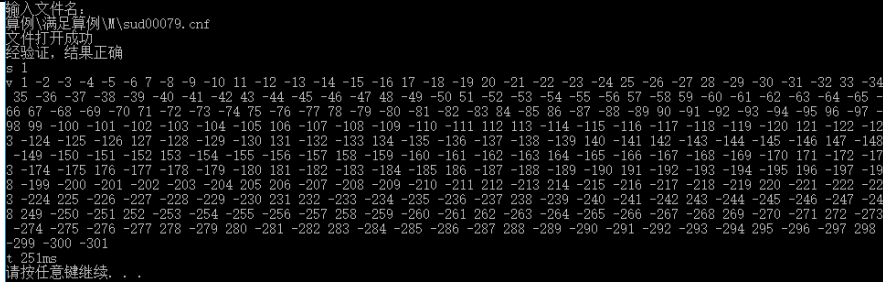
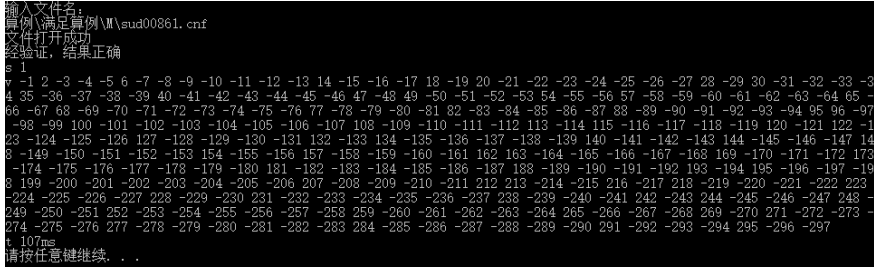
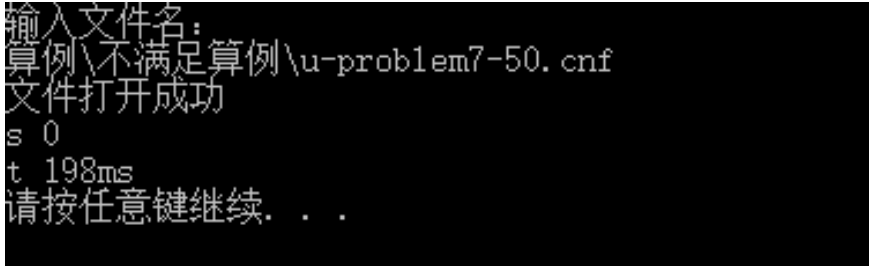
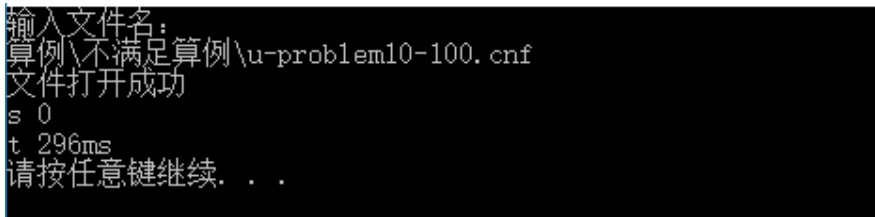
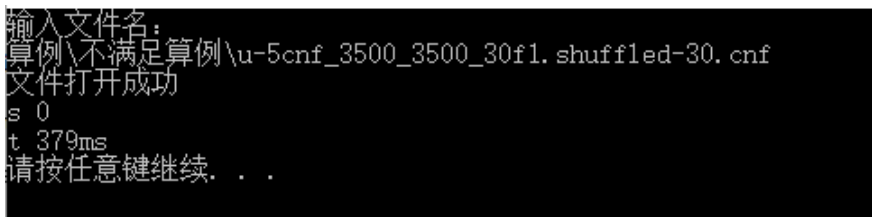

SAT测试结果:

- (1) 未优化程序采取策略: 统计所有子句中每个变元的数量, 选取其中数量最多的变元。测试结果如下表4-1所示:

	算例运行截图	时 间 /ms
性能测试	ais10.cnf 	5700
	sud00009.cnf 	351
功能测试	sat-20.cnf 	52
	unsat-5cnf-30.cnf 	384
	7cnf20_90000_90000_7.shuffled-20.cnf 	508
	problem1-20.cnf	1

S 满 足 算 例	 <pre> 输入文件名: 算例\满足算例\S\problem1-20.cnf 文件打开成功 经验证, 结果正确 s 1 v 1 -2 -3 -4 -5 6 -7 -8 -9 -10 -11 -12 13 14 15 -16 17 -18 -19 20 t 1ms 请按任意键继续. . . </pre>	
	 <pre> 输入文件名: 算例\满足算例\S\problem2-50.cnf 文件打开成功 经验证, 结果正确 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 t 2ms 请按任意键继续. . . </pre>	2
	 <pre> 输入文件名: 算例\满足算例\S\problem3-100.cnf 文件打开成功 经验证, 结果正确 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 t 182ms 请按任意键继续. . . </pre>	182
	 <pre> 输入文件名: 算例\满足算例\S\problem6-50.cnf 文件打开成功 经验证, 结果正确 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 t 2ms 请按任意键继续. . . </pre>	2
	 <pre> 输入文件名: 算例\满足算例\S\problem8-50.cnf 文件打开成功 经验证, 结果正确 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 t 22ms 请按任意键继续. . . </pre>	22
	 <pre> 输入文件名: 算例\满足算例\S\problem9-100.cnf 文件打开成功 经验证, 结果正确 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 t 3ms 请按任意键继续. . . </pre>	3
	 <pre> 输入文件名: 算例\满足算例\S\problem11-100.cnf 文件打开成功 经验证, 结果正确 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 t 3ms 请按任意键继续. . . </pre>	35

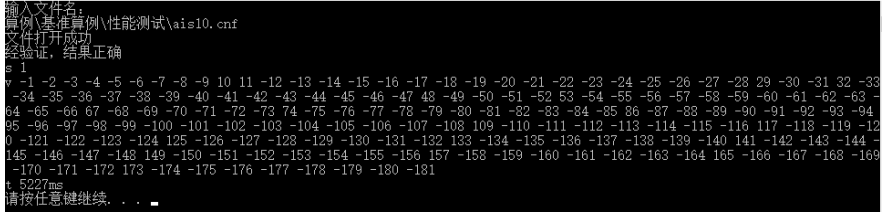
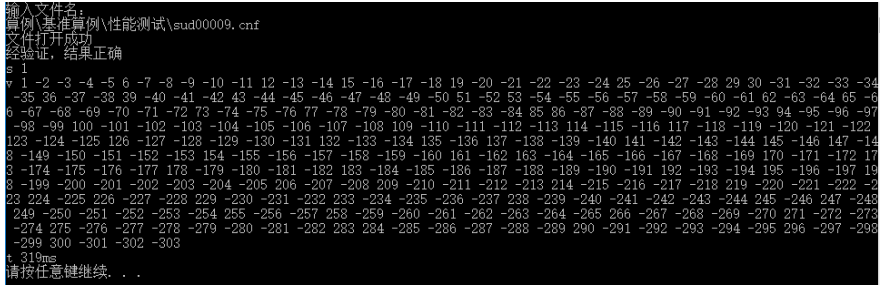
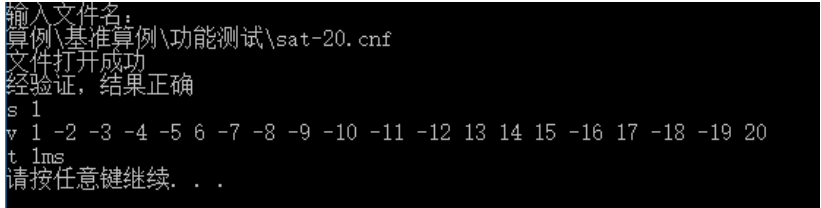
M 满 足 算 例	<p>输入文件名: 算例\满足算例\S\problem11-100.cnf 文件打开成功 经验证, 结果正确</p> <pre>s 1 v 1 -2 3 4 5 -6 -7 -8 -9 -10 -11 -12 -13 14 -15 -16 -17 18 19 -20 21 -22 -23 24 25 -26 27 28 29 30 31 -32 -33 34 -35 -36 37 38 -39 -40 -41 -42 -43 -44 45 -46 -47 -48 49 50 -51 -52 53 54 -55 -56 -57 -58 59 -60 -61 -62 -63 64 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 t 35ms 请按任意键继续. . .</pre>	
	<p>tst_v25_c100.cnf</p> <p>输入文件名: 算例\满足算例\S\tst_v25_c100.cnf 文件打开成功 经验证, 结果正确</p> <pre>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 t 1ms 请按任意键继续. . .</pre>	1
	<p>problem12-200.cnf</p> <p>输入文件名: 算例\满足算例\M\problem12-200.cnf 文件打开成功 经验证, 结果正确</p> <pre>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 284ms 请按任意键继续. . .</pre>	284
	<p>sud00001.cnf</p> <p>输入文件名: 算例\满足算例\M\sud00001.cnf 文件打开成功 经验证, 结果正确</p> <pre>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 -201 -202 -203 -204 -205 -206 -207 -208 209 -210 -211 -212 213 -214 -215 -216 -217 -218 219 -220 221 -222 -223 -224 -225 -226 -227 -228 229 -230 -231 -232 -233 234 -235 236 -237 -238 -239 -240 -241 242 -243 -244 245 -246 -247 -248 -249 -250 -251 252 253 -254 -255 -256 -257 -258 -259 260 -261 -262 -263 -264 265 -266 -267 -268 269 -270 -271 -272 273 -274 -275 -276 -277 278 -279 -280 281 -282 283 -284 -285 -286 287 -288 -289 -290 -291 -292 -293 -294 -295 296 -297 298 -299 -300 -301 t 156ms 请按任意键继续. . .</pre>	156
	<p>sud00009.cnf</p> <p>输入文件名: 算例\满足算例\M\sud00009.cnf 文件打开成功 经验证, 结果正确</p> <pre>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 -201 -202 -203 -204 -205 206 -207 -208 209 -210 -211 -212 -213 214 -215 -216 -217 -218 219 -220 -221 -222 -223 224 -225 226 -227 -228 229 -230 -231 -232 233 -234 -235 -236 -237 238 -239 -240 -241 -242 -243 -244 245 -246 247 -248 249 -250 -251 -252 -253 -254 255 -256 -257 258 -259 -260 -261 -262 -263 -264 -265 266 -267 -268 -269 -270 271 -272 -273 -274 275 -276 -277 -278 -279 -280 -281 -282 283 284 -285 -286 -287 -288 -289 290 -291 -292 -293 -294 -295 296 -297 -298 -299 300 -301 -302 -303 t 359ms 请按任意键继续. . .</pre>	359
	<p>sud00012.cnf</p> <p>输入文件名: 算例\满足算例\M\sud00012.cnf 文件打开成功 经验证, 结果正确</p> <pre>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 -201 -202 203 -204 -205 -206 207 -208 -209 -210 211 -212 -213 -214 215 -216 -217 -218 219 -220 -221 -222 -223 224 -225 226 -227 228 -229 -230 231 -232 t 81ms 请按任意键继续. . .</pre>	81
	<p>sud00079.cnf</p>	251

不 满 足 算 例		
		107
		198
		296
		379
		1

		
	<p>php-010-008.shuffled-as.sat05-1171</p> 	24564

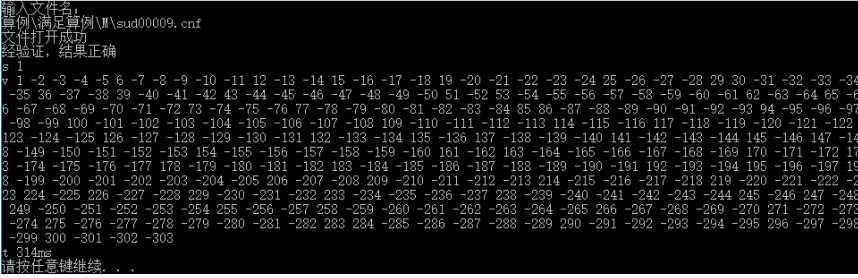
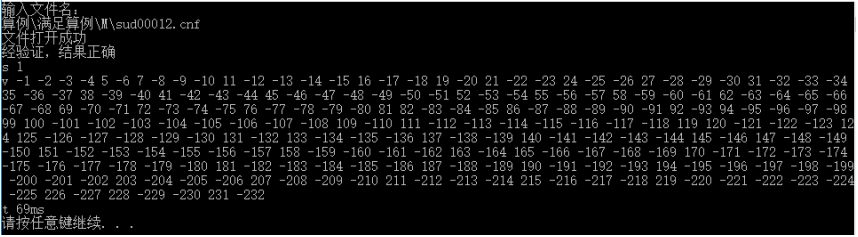
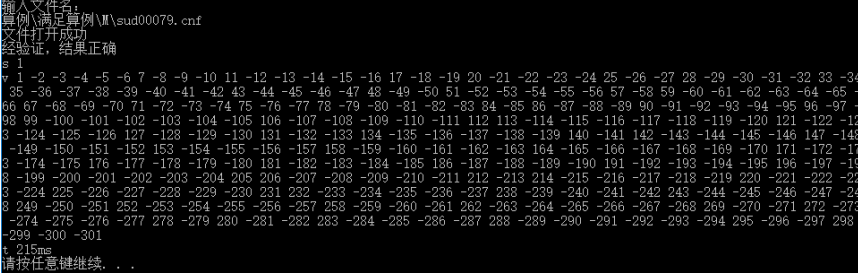
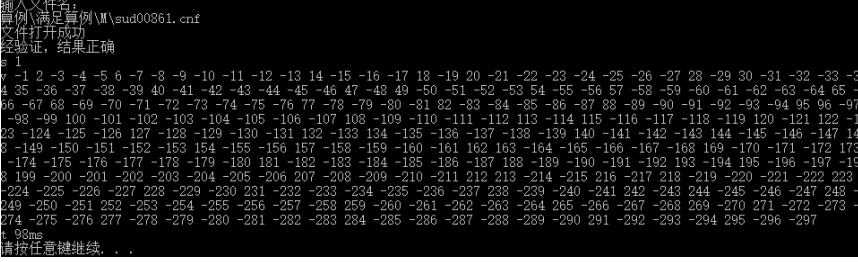
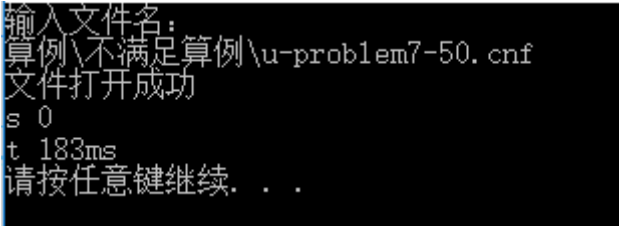
表4-1

(2) 优化程序采取策略：首先选出长度最短的子句，在子句中选择出现次数最多的变元。测试结果如下表4-2所示：

	算例运行截图	时 间 /ms
性能测试	<p>ais10.cnf</p> 	5227
	<p>sud00009.cnf</p> 	319
功能测试	<p>sat-20.cnf</p> 	1

S 满 足 算 例	unsat-5cnf-30. cnf	342
	<pre> 输入文件名: 算例\基准算例\功能测试\unsat-5cnf-30. cnf 文件打开成功 s 0 t 342ms 请按任意键继续. . . </pre>	
	7cnf20_90000_90000_7. shuffled-20. cnf	443
	<pre> 输入文件名: 算例\满足算例\S\7cnf20_90000_90000_7. shuffled-20. cnf 文件打开成功 经验证, 结果正确 s 1 v 1 2 3 -4 -5 -6 -7 8 -9 10 -11 12 13 -14 -15 16 17 -18 19 20 t 443ms 请按任意键继续. . . </pre>	
	problem1-20. cnf	1
	<pre> 输入文件名: 算例\满足算例\S\problem1-20. cnf 文件打开成功 经验证, 结果正确 s 1 v 1 -2 -3 -4 -5 6 -7 -8 -9 -10 -11 -12 13 14 15 -16 17 -18 -19 20 t 1ms 请按任意键继续. . . </pre>	
	problem2-50. cnf	3
	<pre> 输入文件名: 算例\满足算例\S\problem2-50. cnf 文件打开成功 经验证, 结果正确 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 o 37 38 -39 40 -41 42 43 44 45 -46 -47 -48 -49 -50 t 3ms 请按任意键继续. . . </pre>	
	problem3-100. cnf	164
	<pre> 输入文件名: 算例\满足算例\S\problem3-100. cnf 文件打开成功 经验证, 结果正确 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 o -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 o 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 t 164ms 请按任意键继续. . . </pre>	
	problem6-50. cnf	3
	<pre> 输入文件名: 算例\满足算例\S\problem6-50. cnf 文件打开成功 经验证, 结果正确 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 o -37 38 -39 -40 -41 42 43 -44 45 46 47 48 49 50 t 3ms 请按任意键继续. . . </pre>	
	problem8-50. cnf	19

M 满 足 算 例	<p>输入文件名: 算例\满足算例\S\problem8-50.cnf 文件打开成功 经验证, 结果正确 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 t 19ms 请按任意键继续. . .</p>	
	<p>problem9-100.cnf</p> <p>输入文件名: 算例\满足算例\S\problem9-100.cnf 文件打开成功 经验证, 结果正确 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 t 4ms 请按任意键继续. . .</p>	4
	<p>problem11-100.cnf</p> <p>输入文件名: 算例\满足算例\S\problem11-100.cnf 文件打开成功 经验证, 结果正确 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 t 21ms 请按任意键继续. . .</p>	31
	<p>tst_v25_c100.cnf</p> <p>输入文件名: 算例\满足算例\S\tst_v25_c100.cnf 文件打开成功 经验证, 结果正确 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 t 2ms 请按任意键继续. . .</p>	2
	<p>problem12-200.cnf</p> <p>输入文件名: 算例\满足算例\M\problem12-200.cnf 文件打开成功 经验证, 结果正确 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 251ms 请按任意键继续. . .</p>	251
M 满 足 算 例	<p>sud00001.cnf</p> <p>输入文件名: 算例\满足算例\M\sud00001.cnf 文件打开成功 经验证, 结果正确 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 -201 -202 -203 -204 -205 -206 -207 -208 209 -210 -211 -212 213 -214 -215 -216 -217 -218 219 -220 221 -222 -223 224 -225 -226 -227 228 229 -230 -231 -232 -233 234 -235 236 -237 -238 -239 -240 -241 242 -243 -244 245 -246 -247 -248 -249 -250 -251 252 253 -254 -255 -256 -257 -258 -259 260 -261 -262 -263 -264 265 -266 -267 -268 269 -270 -271 -272 273 -274 -275 -276 -277 278 -279 -280 281 -282 283 -284 -285 -286 287 -288 -289 -290 -291 -292 -293 -294 -295 296 -297 298 -299 -300 -301 t 140ms 请按任意键继续. . .</p>	140
	<p>sud00009.cnf</p>	314

	
<p>sud00012.cnf</p> 	69
<p>sud00079.cnf</p> 	215
<p>sud00861.cnf</p> 	98
<p>u-problem7-50.cnf</p> 	183
<p>u-problem10-100.cnf</p>	265

不 满 足 算 例	输入文件名: 算例\不满足算例\u-problem10-100.cnf 文件打开成功 s 0 t 265ms 请按任意键继续. . .	
	u-5cnf_3500_3500_30f1.shuffled-30.cnf 输入文件名: 算例\不满足算例\u-5cnf_3500_3500_30f1.shuffled-30.cnf 文件打开成功 s 0 t 345ms 请按任意键继续. . .	345
	tst_v10_c100.cnf 输入文件名: 算例\不满足算例\tst_v10_c100.cnf 文件打开成功 s 0 t 1ms 请按任意键继续. . .	1
	php-010-008.shuffled-as.sat05-1171 输入文件名: 算例\不满足算例\php-010-008.shuffled-as.sat05-1171.cnf 文件打开成功 s 0 t 22039ms 请按任意键继续. . .	22039

对比可见，优化率大致为5%左右

Sudoku测试结果:

生成的数独	时间/ms
-------	-------

	<pre> 6 7 8 2 3 1 8 5 1 9 7 1 6 5 4 3 2 9 3 1 2 8 9 3 6 5 7 4 5 6 1 4 9 3 8 3 7 2 3 9 7 8 1 6 5 1 3 8 7 6 4 5 2 9 8 9 7 6 4 7 5 3 2 5 4 3 8 9 2 7 1 6 </pre> <p>请输入保存文件名: EasySudoku1 c in 2019/3/3 20:13:57 c p cnf 27 110 1 2 3 -2 -3 4 5 6 7 -5 -6 -5 -7 -6 -7 8 9 10 11 12 -11 -12 13 14 -13 -14 15 16 17 18 19 -18 -19 20 21 22 23 -23 -23 24 2 5 26 27 1 2 5 -1 -2 -1 5 -1 5 -5 3 6 4 7 -4 -7 10 9 8 11 13 -11 -13 12 14 -12 -14 15 16 18 21 -18 -21 19 20 17 22 24 -22 -24 23 27 25 26 25 26 1 17 22 27 -22 -27 2 23 -2 -23 3 8 24 15 18 -15 -18 19 9 11 -9 -11 20 12 10 21 4 5 13 -5 -13 6 14 -6 -14 7 16 -7 -16 1 2 -1 -2 3 9 11 -9 -11 8 12 10 5 13 -5 -13 6 14 -6 -14 4 7 -4 -7 17 15 18 -15 -18 19 20 21 16 22 27 -22 -27 25 26 23 24 文件打开成功!</p> <p>*****</p> <p>数独的解为:</p> <pre> 6 4 5 2 8 1 9 3 7 9 7 8 5 6 9 7 1 4 3 1 2 8 9 7 6 5 3 4 5 6 1 4 9 3 8 3 7 8 9 4 5 6 9 2 1 1 2 3 4 7 1 8 6 5 8 6 4 3 7 2 5 1 9 2 3 1 9 7 6 4 8 5 </pre> <p>t: 8515ms 请按任意键继续. . .</p>	8515
简	<pre> 9 7 8 3 6 9 1 2 5 1 2 6 9 7 4 5 8 3 5 4 5 1 3 6 7 8 9 7 3 9 1 4 5 8 9 1 4 5 6 1 7 8 3 3 3 8 9 1 2 2 3 4 5 2 3 1 5 9 7 4 8 2 5 6 1 9 7 4 8 2 </pre> <p>请输入保存文件名: EasySudoku2 c in 2019/3/3 20:15:21 c p cnf 24 97 1 2 3 4 5 -4 -5 6 7 8 -6 -7 -6 -8 -7 -8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 -23 -24 1 2 6 9 -6 -9 4 7 -4 -7 5 8 -5 -8 3 10 11 12 14 13 16 17 15 18 19 23 22 24 -22 -24 20 21 12 2 18 20 13 15 21 10 19 16 3 4 5 22 -5 -22 17 6 23 -6 -23 7 14 -7 -14 3 24 -8 -24 9 1 11 2 3 6 9 -6 -9 4 7 -4 -7 5 8 -5 -8 1 12 13 10 14 11 18 20 15 16 19 21 23 22 24 -22 -24 17 文件打开成功!</p> <p>*****</p> <p>数独的解为:</p> <pre> 9 7 8 3 1 2 5 6 4 3 1 2 6 4 5 8 9 7 6 4 5 9 7 8 3 2 1 7 8 9 1 2 5 3 6 4 1 2 3 4 5 6 7 8 9 4 5 6 7 8 9 1 2 3 3 8 9 7 2 3 1 1 5 2 3 1 5 6 4 8 7 5 6 4 8 9 7 3 1 2 </pre> <p>t: 18072ms 请按任意键继续. . .</p>	18072
单	<pre> 4 2 3 1 4 7 1 8 5 9 7 1 9 3 6 7 6 2 5 6 1 1 9 3 6 3 8 4 7 7 7 8 9 4 5 1 1 6 4 5 3 3 1 2 9 3 1 2 9 8 5 6 3 7 </pre> <p>请输入保存文件名: EasySudoku3 c in 2019/3/3 20:16:33 c p cnf 26 105 1 2 3 4 5 -4 -5 6 7 -6 -7 8 9 10 -9 -10 11 12 13 14 -13 -14 15 16 17 18 19 20 21 22 -21 -22 23 24 25 26 -25 -26 1 2 3 8 4 6 -4 -6 5 7 -5 -7 9 10 11 -10 -11 12 16 13 14 15 -14 -15 17 18 20 19 21 23 -21 -23 22 25 24 26 -24 -26 9 1 4 -1 -4 5 7 0 21 -5 -10 -5 -21 -10 -21 22 2 17 23 13 6 14 -6 -14 7 11 -7 -11 12 15 24 16 8 19 3 25 18 20 26 -20 -26 9 1 4 -1 -4 2 5 10 -5 -10 6 7 11 -7 -11 8 3 17 12 13 14 15 -14 -15 16 18 21 23 -21 -23 22 24 25 20 26 -20 -26 19 文件打开成功!</p> <p>*****</p> <p>数独的解为:</p> <pre> 5 6 4 2 3 1 9 7 8 3 9 1 5 6 9 7 4 2 4 3 8 1 2 3 7 8 3 4 5 5 1 2 3 7 8 3 8 9 4 4 5 6 1 2 3 1 2 3 7 8 9 4 5 6 6 4 5 3 1 2 8 9 7 3 9 8 6 4 5 9 2 1 3 1 2 9 7 8 5 6 4 </pre> <p>t: 10890ms 请按任意键继续. . .</p>	10890

数

```

生成的数独:

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

请输入保存文件名:
EasySudoku4
c
in 2019/3/3 20:17:15
c
p cnf 23 94
1 2 3 4 -3 -4 5 6 7 8 -7 -8 9 10 11 12 13 14 15 16 -15 -16 17 18 19 20 21 22 23 2 1 3 -1 -3 5 4 6 7 -6 -7 8 12 11 9 10 1
3 14 15 16 17 -16 -17 18 19 20 21 22 23 22 14 6 18 9 21 10 23 1 19 2 20 3 7 11 -3 -7 -3 -11 -7 -11 4 15 8 16 -8 -16 13 1
2 5 17 6 9 2 1 10 12 3 7 11 -3 -7 -3 -11 -7 -11 5 4 6 14 18 19 20 13 15 16 17 -16 -17 22 21 23 文件打开成功!

*****

数独的解为:

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

t: 6354ms
请按任意键继续. . .
    
```

6354

独

```

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

请输入保存文件名:
EasySudoku5
c
in 2019/3/3 20:18:6
c
p cnf 27 110
1 2 3 4 5 6 -5 -6 7 8 9 -8 -9 10 11 12 13 -12 -13 14 15 -14 -15 16 17 18 19 20 -19 -20 21 22 23 24 -23 -24 25 26 27 -26
-27 1 2 3 5 -3 -5 4 6 7 8 9 10 -9 -10 14 11 12 -11 -12 13 15 -13 -15 18 17 16 19 21 20 22 -20 -22 23 25 -23 -26 24 25 -2
4 -25 27 16 23 11 24 -11 -24 19 12 20 25 -12 -20 -12 -25 -20 -25 13 3 7 4 21 26 -21 -26 27 22 14 15 17 -15 -17 18 8 1 9
-1 -9 2 5 -2 -5 6 10 2 1 11 12 -11 -12 13 3 4 7 5 14 -5 -14 6 15 -6 -15 8 9 10 -9 -10 19 23 20 24 25 -20 -24 -20 -25 -24
-25 16 21 26 -21 -26 22 27 18 17 文件打开成功!

*****

数独的解为:

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

t: 13594ms
请按任意键继续. . .
    
```

13594

程序检测结束，基本满足要求。

5 总结与展望

5.1 全文总结

对自己的工作做个总结，主要工作如下：

(1) 准备工作：通过自己区看论文和查资料，了解了 SAT 问题的解法、学习了 CNF 文件的阅览方式、掌握了 DPLL 算法的使用以及使用 SAT 问题的技巧来解决数独问题。

(2) 开始编写程序：首先一边整理程序需要的模块，一边构思需要实现的函数以及数据结构，然后将所有模块之间的关系确定下来。运用模块化编程，将自己构思的程序思路一块一块地实现，比如首先实现 cnf 文件地读取，转化为 cnf 范式，然后将 cnf 范式传入 DPLL 模块中进行求解，最后输出结果并生成文件。将所有模块基本编写完毕后，开始构建模块与模块之间的关系，编写程序与用户的交互界面，最终完成这项课程设计。

(3) 调试阶段：在基本完成课程设计之后，程序依然存在许多问题。运用 Visual Studio 2017，通过单步调试、变量监控，一步一步修复 bug，最终形成了一个能基本达成课设要求，成功运行的程序。

(4) 遇到的问题：在整个过程中，遇到了可许多问题，首先是在构思程序时，因为是第一次接触如此之大的 C 语言工程，如何管理项目文件是我必须解决的第一个问题。因此，我学习了 C 语言的分文件编程，对于以后编辑大规模程序有很大的帮助吧。在思考数据结构的时候也是在一边编程时一边完善。在思考算法时，因为要查询大量的英文资料和论文，阅读和理解这些文字也花费了很多时间。最后就是在调试阶段，一开始有大量的 bug，一时间有很多都弄不清楚，只有在自己慢慢总结尝试才调试完成。

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) 尽量多的去尝试参加这样的大型项目。一是为了锻炼自己的编程能力、增加自己的工作经验；二是为了习惯以后与其他人一起合作的工作环境，虽然本次实验是由自己完成，但是参加工作以后要和其他人一起完成一个项目，这样每个人编程习惯不同，对于代码的可扩展性以及兼容性都要多多考虑。

(2) 需要多多了解国外的前沿技术。通过这次课程设计，让我知道了国内的学习资源确实受到了很大的限制。因此，能够多多参考学校外的资料对我的学习来说十分重要。就像这次了解DPLL算法以及了解数独游戏的规约都是参考了国外的论文了解到的。

参考文献

- [1] 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
- [2] Carsten Sinz. Visualizing SAT Instances and Runs of the DPLL Algorithm. J Autom Reasoning (2007) 39:219–243
- [3] 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.
- [4] 百度百科: 软件测试方法
<https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E6%B5%8B%E8%AF%95%E6%96%B9%E6%B3%95/1850037?fr=aladdin#1>

附录

head.h

```
#pragma once

#ifndef HEAD_H_INCLUDED
#define HEAD_H_INCLUDED

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>


//一些定量
#define TURE 1
#define FALSE 0
#define CONFUSE -1


//status用于做状态变量TRUE、FALSE等的类型
typedef int status;


/*
int pos; 表示文字状态 1代表正文字 0代表负文字
int num; 表示文字是第几号变元
struct Literal* next; 指向下一个文字
*/
typedef struct Literal
{
    int pos;//表示文字状态 1代表正文字 0代表负文字
    int num;//表示文字是第几号变元
    struct Literal* next;//指向下一个文字
```

```

}Literal;

/*

int num_lit;          文字数

struct Literal* elem; 指向子句中的第一个文字

struct Statement* next; 指向下一个子句

*/

typedef struct Statement
{
    int num_lit;//文字数

    struct Literal* elem;//指向子句中的第一个文字

    struct Statement* next;//指向下一个子句

}Statement;

/*

int sta; 标识公式是够可满足，0为不满足，1为满足，-1为待定

int num_st; 表示子句数

int num_v; 变元数

struct Statement* root 指向公式中的第一个子句

*/

typedef struct Formula
{
    int sta;//标识公式是够可满足，0为不满足，1为满足，-1为待定

    int num_st;//表示子句数

    int num_v;//变元数

    struct Statement* root;//指向公式中的第一个子句

}Formula;

/*

int* V; 储存已确定信息的变元

struct Formula* S; 表示栈中存储的公式

struct Stack_F* next; 用于指向下一个节点

*/

```

```
typedef struct Stack_F
{
    int* V;//储存已确定信息的变元
    struct Formula* S;//表示栈中存储的公式
    struct Stack_F* next;//用于指向下一个节点
}Stack_F;

/*-----用于数独格局转cnf范式的数据结构-----*/
//通过邻接表储存数独格局中的每个空格，空格由可能的数字组成，每个可能的
//数字对应于一个变元号
//hole用于储存每一个空格
typedef struct hole
{
    int x;//标识行
    int y;//标识列
    int dig_num;//标识可能的数字数
    struct digital* elem;//指向第一个可能的数字
}hole;

//digital用于储存每个空格中可能出现的数字
typedef struct digital
{
    int num;//数字
    int v_num;//数字所对应的变元编号
    struct digital* next;
}digital;

//原函数
void SAT(long, long);
void CNF_Reader(FILE*, Formula*);
```



```
status Solver(Formula*, int*);  
bool Check(int* Varies_res, Formula* formula);  
Statement* SearchStatement(int k, Formula* formula);  
Literal* SearchLiteral(int k, Statement* statement);
```

```
status HaveUnitClause(Formula*);  
void RemoveClause(Literal, Formula*);  
void ClearClause(Literal, Formula*);  
status HaveEmptyClause(Formula*);  
Formula* AddClause(Literal, Formula*);  
void DestoryFormula(Formula*);  
Literal Choose_L(Formula*);  
Formula* Copy_F(Formula*);  
int* Copy_V(int*, int);
```

```
void Sudoku(long time_start, long time_end);  
void CreateFinalSud(int(*)[9]);  
int CreateSudoku(int(*)[9]);  
int* TransfSud(int(*)[9], Formula*, hole*);  
void ShowSudResult(int *, hole*, int(*)[9]);  
void CnfToFile(Formula*);
```

```
#endif // HEAD_H_INCLUDED
```

FinishedProject.cpp

```
#include "head.h"
#include "conio.h"
void main(void) {
    int choice = 1;
    while (choice != 0) {
        fflush(stdin);

        long time_start = 0; //记录程序开始时间
        long time_end = 0;   //记录程序结束时间

        printf("1、通过文件生成cnf范式并求解\n\n");
        printf("2、随机生成一个数独并求解\n\n");
        printf("0、退出\n\n");
        printf("-----\n");
        printf("输入相应序号:\n");
        scanf("%d", &choice);
        getchar();
        switch (choice)
        {
            case 1: SAT(time_start, time_end); break;
            case 2: Sudoku(time_start, time_end); break;
            case 0: break;
            default: printf("输入非法\n");
        }
        // setbuf(stdin, NULL);
        system("pause");
        system("cls");
    }
}
```

SAT.cpp

```
#include "head.h"

/*SAT问题求解：通过读取cnf文件，求出其生成的cnf范式的解*/

void SAT(long time_start, long time_end) {

    system("cls");

    FILE* fp;

    char filename[100];

    Formula formula;    //保存生成的cnf范式

    int SAT;            //cnf可满足性标记，0为不满足，1为满足

    int* Varies = NULL; //保存所有变元信息，0为假，1为真，-1为不确定

    //formula初始化

    formula.sta = -1;

    formula.num_st = 0;

    formula.root = NULL;

    /*****CNF文件读取*****/

    printf("输入文件名：\n");

    scanf("%s", filename);

    getchar();

    fp = fopen(filename, "r");

    if (!fp) {

        printf("文件打开失败\n");

        return;

    }

    printf("文件打开成功\n");

    CNF_Reader(fp, &formula);

    fclose(fp);
```

```
/******CNF文件读取完毕******/
```

```
/****运用SAT_Solver验证cnf范式的可满足性*****/
```

```
Varies = (int*)malloc((formula.num_v + 1) * sizeof(int));
```

```
int i;
```

```
//初始化Varies
```

```
for (i = 1; i <= formula.num_v; i++) {
```

```
    Varies[i] = -1;
```

```
}
```

```
//开始计时
```

```
time_start = clock();
```

```
SAT = Solver(&formula, Varies);
```

```
time_end = clock();//记录程序结束时间
```

```
/*******/
```

```
if (SAT == 1) {
```

```
    if (Check(Varies, &formula) == true) {
```

```
        printf("经验证，结果正确\n");
```

```
    }
```

```
    else {
```

```
        printf("经验证，结果错误\n");
```

```
    }
```

```
}
```

```
/*输出验证结果*/
```

```
//将文件名改为以.res结尾
```

```
int str_length;
```

```
str_length = strlen(filename);
```

```
char str[] = ".res";
```

```

strcpy(filename + str_length - 4, str);
fp = fopen(filename, "w");//打开一个与算例同名的res文件
if (!fp)
{
    printf("文件打开失败！ \n");
}
//输出可满足性
printf("s %d\n", SAT);
//向文件中写入可满足性
char c = 's';
fprintf(fp, "%c\t", c);
fprintf(fp, "%d\n", SAT);
//文件可满足时，输出一组真值，并向文件中写入一组真值
if (SAT == 1) {
    c = 'v';
    printf("%c ", c);
    fprintf(fp, "%c\t", c);
    for (i = 1; i <= formula.num_v; i++) {
        if (Varies[i]) {
            printf("%d ", i);
            fprintf(fp, "%d ", i);
        }
        else {
            printf("-%d ", i);
            fprintf(fp, "-%d ", i);
        }
    }
    printf("\n");
    fprintf(fp, "\n");
}

```

```

//输出程序运行时间

long k;

k = (time_end - time_start);

c = 't';

printf("%c %ldms\n", c, k);

fprintf(fp, "%c\t", c);

fprintf(fp, "%ld\n", k);

fclose(fp);

}

bool Check(int* Varies_res, Formula* formula) {
    //找每一个子句
    for (int i = 0; i < formula->num_st; i++) {
        //找到子句的每一个文字
        for (int j = 0; j < SearchStatement(i, formula)->num_lit; j++) {
            if (SearchLiteral(j, SearchStatement(i, formula))->pos ==
Varies_res[SearchLiteral(j, SearchStatement(i, formula))->num]) {
                //一个文字是真的，直接下一条语句
                break;
            }
            else {
                //已经检索到最后一个文字
                if (j == SearchStatement(i, formula)->num_lit - 1) {
                    return false;
                }
            }
        }
    }
    return true;
}

```

```
Statement* SearchStatement(int k, Formula* formula) {  
    Statement* res = formula->root;  
    for (int i = 0; i <= formula->num_st; i++) {  
        if (i == k) {  
            return res;  
        }  
        else {  
            res = res->next;  
        }  
    }  
    printf("ERROR!Beyond the range!");  
    return NULL;  
}
```

```
Literal* SearchLiteral(int k, Statement* statement) {  
    Literal* res = statement->elem;  
    for (int i = 0; i <= statement->num_lit; i++) {  
        if (i == k) {  
            return res;  
        }  
        else {  
            res = res->next;  
        }  
    }  
    printf("ERROR!Beyond the range!");  
    return NULL;  
}
```

CNF_Reader.cpp

```
#include "head.h"

/*通过cnf文件生成cnf范式*/

void CNF_Reader(FILE* fp, Formula* formula) {

    char buff[1000];//存储注释

    //将注释存到buff数组中

    while (fgetc(fp) == 'c') {

        fgets(buff, 1000, fp);

    }

    //通过fscanf读取变元数和子句数

    int i;

    //读取“cnf”字符串

    for (i = 1; i <= 5; i++) {

        fgetc(fp);

    }

    //将cnf文件中给出的变元数和子句数写入公式中

    fscanf(fp, "%d %d", &(formula->num_v), &(formula->num_st));

    //读取换行符

    fgetc(fp);

    /*****根据文件创建公式*****/

    Statement* statement;//创建子句

    formula->root = (Statement*)malloc(sizeof(Statement));

    statement = formula->root;

    statement->next = NULL;

    statement->num_lit = 0;

    for (i = 1; i <= formula->num_st; i++) {

        if (formula->root->num_lit != 0) {

            statement->next = (Statement*)malloc(sizeof(Statement));

            statement = statement->next;

        }

    }

}
```



```

statement->next = NULL;

statement->num_lit = 0;

}

//创建第一个文字
Literal* literal;

statement->elem = (Literal*)malloc(sizeof(Literal));

literal = statement->elem;

literal->next = NULL;

int k;

//将文件中目前读取的文字写入k
fscanf(fp, "%d", &k);

//完成后续文字的读取
while (k != 0) {
    if (statement->num_lit != 0) {
        literal->next = (Literal*)malloc(sizeof(Literal));
        literal = literal->next;
        literal->next = NULL;
    }
    if (k > 0) {
        literal->pos = 1;
        literal->num = k;
    }
    else {
        literal->pos = 0;
        literal->num = (-k);
    }
    (statement->num_lit)++; //每读取一个文字，子句的文字数加一
    fscanf(fp, "%d", &k);
}

```

```
fgetc(fp);//读取每行子句尾的换行符
    }
}
```

SAT_Solver.cpp

```
#include "head.h"
```

```
/*
```

在提供cnf范式的前提下判断公式的可满足性。

```
*/
```

```
status Solver(Formula* formula, int* Varies)
```

```
{
```

```
    Formula* formula_temp;    //用于临时储存生成的公式
```

```
    Formula* formula_current = formula; //用于表示当前公式
```

```
    Literal literal;           //用于记录分裂策略选出的文字
```

```
    int* current_Varies = 0;    //用于表示当前变元真值状况
```

```
    bool token = true;         //用于标记是否为第一次循环
```

```
    int i;
```

```
    Stack_F* top;
```

```
    Stack_F* p;
```

```
    top = (Stack_F*)malloc(sizeof(Stack_F));
```

```
    top->S = Copy_F(formula);
```

```
    top->V = Copy_V(Varies, formula->num_v);
```

```
    top->next = NULL;
```

```
    /*****DPLL求解*****/
```

```
    while ((formula_current->sta != 1) && (top != NULL)) {
```

//不是第一次进行DPLL，则进行删除公式和真值关系操作

```
        if (token == true) {
```

```
            token = false;
```

```
        }
```

```
        else {
```

```
            DestoryFormula(formula_current);
```

```
            free(current_Varies);
```

```
}
```

//栈头公式出栈并作为当前公式，栈头变元真值关系出栈作为当前变元真值关系

```
formula_current = top->S;
```

```
current_Varies = top->V;
```

```
p = top;
```

```
top = top->next;
```

```
free(p);
```

//当存在单子句时，用单子句传播规则处理公式

```
while (HaveUnitClause(formula_current)) {
```

```
    //选择一个单子句，并记录文字
```

```
    Literal L;
```

```
    Statement* p_s; //用于遍历公式
```

```
    p_s = formula_current->root;
```

```
    // 找到单子句
```

```
    while (p_s->num_lit != 1) {
```

```
        p_s = p_s->next;
```

```
    }
```

```
    L = *(p_s->elem);
```

//记录此单子句所含文字的变元使单子句为真

```
    if (L.pos == 1)
```

```
        current_Varies[L.num] = 1;
```

```
    else if (L.pos == 0)
```

```
        current_Varies[L.num] = 0;
```

//删除所有包含L的子句

```
    RemoveClause(L, formula_current);
```

//对S1中的每个子句，如果它包含文字 非L,则从该子句中去掉这个

文字

```

    if (L.pos == 1)
        L.pos = 0;
    else if (L.pos == 0)
        L.pos = 1;
    //清除公式中L记录文字的负文字
    ClearClause(L, formula_current);

    //如果它是空集，当前CNF范式满足
    if (formula_current->root == NULL) {
        //当公式可满足时，复制当前变元真值关系
        formula_current->sta = 1;
        for (i = 1; i <= formula->num_v; i++) { //复制此时的真值状态
            Varies[i] = current_Varies[i];
        }
    }
    //如果不是空集，当前CNF范式不满足
    else if (HaveEmptyClause(formula_current)) {
        formula_current->sta = 0;
    }
}

//当不存在单子句时，且不知道范式是否满足，则使用分裂策略将S分
为S1, S2
//策略：选出长度最短的子句，在子句中选择出现次数最多的变元
if (formula_current->sta == -1) {
    literal = Choose_L(formula_current);
    //先向当前范式的子句集合中加入由选出的文字，组成S2
    formula_temp = AddClause(literal, formula_current);
    p = (Stack_F*)malloc(sizeof(Stack_F));

```

```

        p->S = formula_temp;
        p->V = Copy_V(current_Varies, formula->num_v);
        p->next = top;
        top = p;

        //再向子句集中加入选出文字的负文字，组成S1
        literal.pos = 0;
        formula_temp = AddClause(literal, formula_current);
        p = (Stack_F*)malloc(sizeof(Stack_F));
        p->S = formula_temp;
        p->V = Copy_V(current_Varies, formula->num_v);
        p->next = top;
        top = p;
    }
}

/*****DPLL结束*****/

if (formula_current->sta == 1) {
    return TURE;
}
else if (formula_current->sta == 0) {
    return FALSE;
}
else {
    return CONFUSE;
}
}

```

Solver_Functions.cpp

```
#include "head.h"

/*-----solver中需要的函数-----*/
/*拷贝当前的公式*/
Formula* Copy_F(Formula* formula) {
    Formula* S1;
    S1 = (Formula*)malloc(sizeof(Formula));
    //复制当前公式中的所有信息
    S1->sta = formula->sta;
    S1->num_st = formula->num_st;
    S1->num_v = formula->num_v;
    S1->root = (Statement*)malloc(sizeof(Statement));
    S1->root->next = NULL;
    //复制公式链表
    Statement* p_s1;
    Statement* p_s;
    p_s1 = S1->root;
    p_s = formula->root;
    while (p_s != NULL) {
        p_s1->num_lit = p_s->num_lit;
        p_s1->elem = (Literal*)malloc(sizeof(Literal));
        p_s1->elem->next = NULL;
        Literal* p_l1;
        Literal* p_l;
        p_l1 = p_s1->elem;
        p_l = p_s->elem;
        while (p_l != NULL) {
            p_l1->num = p_l->num;
```

```

        p_l1->pos = p_l->pos;
        if (p_l->next != NULL) {
            p_l1->next = (Literal*)malloc(sizeof(Literal));
            p_l1 = p_l1->next;
        }
        p_l1->next = NULL;
        p_l = p_l->next;
    }
    if (p_s->next != NULL) {
        p_s1->next = (Statement*)malloc(sizeof(Statement));
        p_s1 = p_s1->next;
    }
    p_s1->next = NULL;
    p_s = p_s->next;
}
return S1;
}

```

/*拷贝当前变元真值关系*/

```

int* Copy_V(int* V, int num_v) {
    int* V1; //拷贝者
    int i;
    V1 = (int*)malloc((num_v + 1) * sizeof(int));
    for (i = 1; i <= num_v; i++) {
        V1[i] = V[i];
    }
    return V1;
}

```


/*判断当前公式是否含有单子句*/

```
status HaveUnitClause(Formula* formula) {
    int token = 0; //用于标识公式是否含有单子句，如有则标志为1，如没有则标志为0

    Statement* p_s;
    p_s = formula->root;
    //遍历公式检验公式中是否含有单子句
    while ((token != 1) && (p_s != NULL)) {
        if (p_s->num_lit == 1)
            token = 1;

        p_s = p_s->next;
    }
    //当标志为1的时候返回有单子句，不为1时返回无单子句
    if (token == 1) {
        return TURE;
    }
    else {
        return FALSE;
    }
}
```

/*去除含有指定文字的子句*/

```
void RemoveClause(Literal lit, Formula* formula) {
    int token = 0; //用于标记当前子句是否含有特定文字,1代表含有
    int token_root = 0; //用于标记根节点是否被检验,1表示已检验
    Statement* p_s = NULL; //用于遍历公式
    Statement* p_sf = NULL; //用于删除子句
    Literal* p_l = NULL; //用于遍历子句
    Literal* p_lf = NULL; //用于删除文字
```

//遍历所有子句，碰到含有特定文字的子句则删除之

//先判断根子句是否含有特定文字

```
while ((token_root != 1) && formula->root != NULL)
{
    p_l = formula->root->elem;
    while ((token != 1) && (p_l != NULL)) {
        if ((p_l->pos == lit.pos) && (p_l->num == lit.num))
            token = 1;

        p_l = p_l->next;
    }
    if (token == 1) {
        p_sf = formula->root;
        formula->root = formula->root->next;
        p_l = p_sf->elem;
        free(p_sf);
        p_sf = NULL;
        while (p_l != NULL) {
            p_lf = p_l->next;
            free(p_l);
            p_l = p_lf;
        }
        (formula->num_st)--;//子句数减一
    }
    else {
        token_root = 1;
    }
    token = 0;
}

if (formula->root) {
```

//遍历剩余公式

```
p_sf = formula->root;
p_s = p_sf->next;
while (p_s != NULL) {
    p_l = p_s->elem;
    while ((token != 1) && (p_l != NULL)) {
        if ((p_l->pos == lit.pos) && (p_l->num == lit.num))
            token = 1;
        p_l = p_l->next;
    }
}
```

//删除含有特定文字的子句

```
if (token == 1) {
    p_s = p_s->next;
    p_l = p_sf->next->elem;
    free(p_sf->next);
    while (p_l != NULL) {
        p_lf = p_l->next;
        free(p_l);
        p_l = p_lf;
    }
    p_sf->next = p_s;
    (formula->num_st)--;//子句数减一
}
else {
    p_s = p_s->next;
    p_sf = p_sf->next;
}
token = 0;//重置标记
}
```

```
}
```

/*清除子句中的特定文字*/

```
void ClearClause(Literal lit, Formula* formula) {
    Statement* p_s = NULL; //用于遍历公式
    Literal* p_l = NULL; //用于遍历子句
    Literal* p_lf = NULL; //用于删除文字
    int token = 0; //用于标记根文字是否被检验,1表示已检验
    p_s = formula->root;
    while (p_s != NULL) {
        while ((token != 1) && (p_s->elem != NULL)) { //先检验根文字
            p_l = p_s->elem;
            if ((p_l->pos == lit.pos) && (p_l->num == lit.num)) { //根文字为特定
                文字
                p_s->elem = p_s->elem->next;
                free(p_l); //删除根文字
                p_l = NULL;
                (p_s->num_lit)--; //子句文字数减一
            }
            else {
                token = 1; //根文字已检验
            }
        }
        if (p_s->elem) {
            p_lf = p_s->elem;
            p_l = p_lf->next;
            while (p_l != NULL) { //检验其他文字
                if ((p_l->pos == lit.pos) && (p_l->num == lit.num)) { //此文字为特
                    定文字
                    p_l = p_l->next;

```

```

        free(p_lf->next);//删除根文字
        (p_s->num_lit)--;//子句文字数减一
        p_lf->next = p_l;
    }
    else {
        p_l = p_l->next;
        p_lf = p_lf->next;
    }
}
}
p_s = p_s->next;
token = 0;//下一句根文字未检验
}
}

```

/*判断当前公式是否含有空子句，有则返回TRUE*/

```

status HaveEmptyClause(Formula* formula) {
    int token;
    token = 0;
    Statement* p;
    p = formula->root;
    while ((token != 1) && (p != NULL)) {
        if (p->num_lit == 0) {
            token = 1;
        }
        p = p->next;
    }
    if (token == 1)
        return TURE;
    else

```

```

        return FALSE;
    }

    /*
    在公式中选择一个特定的正文字并返回
    遍历公式选出长度最短的子句，在子句中选择出现次数最多的变元，返回其正
    文字
    */

    Literal Choose_L(Formula* formula) {
        Literal lit;          //用于返回选择变元的正文字
        Statement* shortest; //用于指向最短的子句
        Literal* p_l;         //用于遍历最短子句
        Statement* p;         //用于遍历公式
        int v[2];             //用于保存出现次数最多的变元及出现次数
        int i;

        //遍历公式找出最短的子句
        p = formula->root;
        shortest = p;
        while (p != NULL) {
            if ((p->num_lit) < (shortest->num_lit))
                shortest = p;
            p = p->next;
        }

        //在最短子句中找出出现次数最多的变元
        int** num_v; //保存变元及其出现数目的二维数组
        num_v = (int**)malloc((shortest->num_lit) * sizeof(int*));
        for (i = 0; i < shortest->num_lit; i++) {
            num_v[i] = (int*)malloc(2 * sizeof(int));
            num_v[i][0] = 0;
            num_v[i][1] = 0;
        }
    }
}

```

```

    }

    p_l = shortest->elem;
    i = 0;
    while (p_l != NULL) {
        for (i = 0; (num_v[i][0] != p_l->num) && (num_v[i][0] != 0) && (i <
shortest->num_lit); i++);
        if (num_v[i][0] == p_l->num)
            num_v[i][1]++;
        else if (num_v[i][0] == 0) {
            num_v[i][0] = p_l->num;
            num_v[i][1] = 1;
        }
        p_l = p_l->next;
    }
    v[0] = num_v[0][0];
    v[1] = num_v[0][1];
    for (i = 0; i < shortest->num_lit; i++) {
        if (num_v[i][1] > v[1])
        {
            v[0] = num_v[i][0];
            v[1] = num_v[i][1];
        }
    }
    lit.pos = 1;
    lit.num = v[0];
    return lit;
}

/*

```

将一个给定的文字加入一个公式中

输入一个文字和一个公式

返回一个新的公式(该公式的另外开辟储存空间)

*/

```
Formula* AddClause(Literal lit, Formula* formula) {
```

```
    Formula* S1;           //用于返回生成的公式
```

```
    Statement* p_s;        //用于生成一个单子句
```

```
    Statement* p;          //用于遍历公式
```

```
    S1 = Copy_F(formula);  //拷贝公式
```

```
    //根据文字生成一个单子句
```

```
    p_s = (Statement*)malloc(sizeof(Statement));
```

```
    p_s->elem = (Literal*)malloc(sizeof(Literal));
```

```
    p_s->elem->num = lit.num;
```

```
    p_s->elem->pos = lit.pos;
```

```
    p_s->elem->next = NULL;
```

```
    p_s->num_lit = 1;
```

```
    p_s->next = NULL;
```

```
    //将单子句纳入公式中
```

```
    p = S1->root;
```

```
    while (p->next != NULL)
```

```
        p = p->next;
```

```
    p->next = p_s;
```

```
    S1->num_st++; //子句数加一
```

```
    return S1;
```

```
}
```

```
/*
```

销毁输入指针所指向的公式所占用的内存区域

```
*/
```

```
void DestoryFormula(Formula* formula) {
```



```

Statement* p_s; //用于遍历公式
Statement* p_sf; //用于删除子句
Literal* p_l; //用于遍历子句
Literal* p_lf;    //用于删除文字
//删除公式，记录根子句位置
p_s = formula->root;
free(formula);
//外层循环遍历删除子句，内层循环遍历删除文字
while (p_s != NULL) {
    p_sf = p_s;
    p_s = p_s->next;
    p_l = p_sf->elem; //记录根文字位置，删除子句
    free(p_sf);
    while (p_l != NULL) {
        p_lf = p_l;
        p_l = p_l->next;
        free(p_lf);
    }
}
}

```

Sudoku.cpp

```
#include "head.h"

/*
生成一个数独格局，并用SAT相关知识解决问题
*/

void Sudoku(long time_start, long time_end) {
    int sud[9][9];

    int i;

    int j;

    Formula formula;

    int *Varies;

    Varies = NULL;

    hole* holes;

    int hole_num;

    hole_num = 0;

    system("cls");

    printf("\n随机生成一个数独格局并转化为SAT问题解决\n\n");
    printf("-----\n\n");

    //创建数独数组9×9格局
    for (i = 0; i < 9; i++) {
        for (j = 0; j < 9; j++) {
            sud[i][j] = 0; //初始格局为空
        }
    }

    CreateFinalSud(sud); //生成数独最终格局

    //生成完成了一个数独

    hole_num = CreateSudoku(sud);
```

//展示生成的数独问题格局

```
system("cls");
printf("\n生成的数独: \n\n");
printf("      ----- \n");
for (i = 0; i < 9; i++)
{
    printf("\t|");
    for (j = 0; j < 9; j++)
    {
        if (sud[i][j])
            printf("%d|", sud[i][j]);
        else
            printf(" |");
    }
    printf("\n");
}
printf("      ----- \n\n");
```

//数独问题转化为SAT问题

```
holes = (hole*)malloc(hole_num * sizeof(hole));
formula.num_st = 0;
formula.num_v = 0;
formula.root = NULL;
formula.sta = -1;
//开始计时
time_start = clock();
Varies = TransfSud(sud, &formula, holes);
//输出cnf文件
CnfToFile(&formula);
```

```
//利用dpll算法解决数独产生的SAT问题
Solver(&formula, Varies);
//输出dpll算法求出的解
ShowSudResult(Varies, holes, sud);
//结束计时
time_end = clock();//记录程序结束时间
long k;
k = (time_end - time_start);
char c = 't';
printf("%c %ldms\n", c, k);
}
```

Sudoku_Functions.cpp

```
#include "head.h"
```

```
void CreateFinalSud(int(*sud)[9]) {
```

```
    int i;
```

```
    int j;
```

```
    int k;
```

```
    int a;
```

```
    //在数独格局中给定最初的位于数独格局中央的 3×3 格局
```

```
    k = 1;
```

```
    while (k <= 9) { //由 1 到 9 依次赋值
```

```
        for (i = 3; i < 6; i++) {
```

```
            for (j = 3; j < 6; j++) {
```

```
                sud[i][j] = k++;
```

```
            }
```

```
        }
```

```
    }
```

```
    //得到初始 3×3 格局左右 3×3 位置中的数字
```

```
    srand((unsigned)time(NULL));
```

```
    k = rand() % 2; //通过随机数决定完全行变换产生的 3×3 格局的左右位置
```

```
    if (k == 1) {
```

```
        //初始化左边
```

```
        for (j = 0; j < 3; j++) {
```

```
            sud[3][j] = sud[5][j + 3];
```

```
        }
```

```
        for (i = 4; i < 6; i++) {
```

```
            for (j = 0; j < 3; j++) {
```

```
                sud[i][j] = sud[i - 1][j + 3];
```

```
            }
```

```
        }
```

```
        //初始化右边
```

```
        for (j = 6; j < 9; j++) {
```

```
            sud[5][j] = sud[3][j - 3];
```

```
        }
```

```

    for (i = 3; i < 5; i++) {
        for (j = 6; j < 9; j++) {
            sud[i][j] = sud[i + 1][j - 3];
        }
    }
}
else {
    //初始化左边
    for (j = 0; j < 3; j++) {
        sud[5][j] = sud[3][j + 3];
    }
    for (i = 3; i < 5; i++) {
        for (j = 0; j < 3; j++) {
            sud[i][j] = sud[i + 1][j + 3];
        }
    }
    //初始化右边
    for (j = 6; j < 9; j++) {
        sud[3][j] = sud[5][j - 3];
    }
    for (i = 4; i < 6; i++) {
        for (j = 6; j < 9; j++) {
            sud[i][j] = sud[i - 1][j - 3];
        }
    }
}
//通过列变换得到其他 3×3 格局
for (a = 0; a < 9; a += 3) {
    k = rand() % 2; //每个列变换具有两种可能
    if (k == 1) {
        //初始化上边
        for (i = 0; i < 3; i++) {
            sud[i][a] = sud[i + 3][a + 2];

```

```

    }
    for (j = a + 1; j < a + 3; j++) {
        for (i = 0; i < 3; i++) {
            sud[i][j] = sud[i + 3][j - 1];
        }
    }
    //初始化下边
    for (i = 6; i < 9; i++) {
        sud[i][a + 2] = sud[i - 3][a];
    }
    for (j = a; j < a + 2; j++) {
        for (i = 6; i < 9; i++) {
            sud[i][j] = sud[i - 3][j + 1];
        }
    }
}
else {
    //初始化上边
    for (i = 0; i < 3; i++) {
        sud[i][a + 2] = sud[i + 3][a];
    }
    for (j = a; j < a + 2; j++) {
        for (i = 0; i < 3; i++) {
            sud[i][j] = sud[i + 3][j + 1];
        }
    }
    //初始化下边
    for (i = 6; i < 9; i++) {
        sud[i][a] = sud[i - 3][a + 2];
    }
    for (j = a + 1; j < a + 3; j++) {
        for (i = 6; i < 9; i++) {
            sud[i][j] = sud[i - 3][j - 1];
        }
    }
}

```

```

        }
    }
}
}

/*
creatSudProblem 用于产生数独问题
*/
int CreateSudoku(int(*sud)[9]) {
    int level;
    int hole_num;
    int i;
    //记录挖洞位置
    int hole_x;
    int hole_y;

    //生成挖洞数
    printf("-----\n");
    printf("输入数字选择难度：【1~3】\n");
    printf("1、easy\n2、normal\n3、evil\n\n");

    scanf("%d", &level);
    while (level > 3 || level < 1)
        printf("输入非法,重新输入：\n");
    hole_num = level * 20;

    //挖洞，利用随机数生成挖洞位置
    srand((unsigned)time(NULL));
    for (i = 1; i <= hole_num; i++) {
        hole_x = rand() % 9;
        hole_y = rand() % 9;
        while (sud[hole_x][hole_y] == 0) {

```



```

        hole_x = rand() % 9;
        hole_y = rand() % 9;
    }
    sud[hole_x][hole_y] = 0;
}
return hole_num;
}

```

/*

将给定的 cnf 范式输入到文件中

输入为给定的 cnf 范式，输出一个 cnf 公式

*/

```

void CnfToFile(Formula* formula) {
    FILE* fp;
    char filename[100];
    Statement* statement;
    Literal* literal;

    printf("请输入保存文件名: \n");
    scanf("%s", filename);
    fp = fopen(filename, "w");
    char str[] = ".cnf";
    strcpy(filename + strlen(filename), str); //为文件名加上后缀
    if (!fp)
        printf("文件打开失败!\n");
    else {
        time_t t; //用于输出当前时间
        struct tm* lt;
        time(&t);
        lt = localtime(&t);
        fprintf(fp, "c          in %d/%d/%d %d:%d:%d\n", lt->tm_year + 1900,
lt->tm_mon, lt->tm_mday, lt->tm_hour, lt->tm_min, lt->tm_sec);
        printf("c          in %d/%d/%d %d:%d:%d\n", lt->tm_year + 1900,

```

```
lt->tm_mon, lt->tm_mday, lt->tm_hour, lt->tm_min, lt->tm_sec);

//输入变元数和子句数等信息
fprintf(fp, "p cnf %d %d\n", formula->num_v, formula->num_st);
printf("p cnf %d %d\n", formula->num_v, formula->num_st);

//输入子句部分
statement = formula->root;
while (statement) {
    literal = statement->elem;
    while (literal) {
        if (literal->pos) {
            fprintf(fp, "%d ", literal->num);
            printf("%d ", literal->num);
        }
        else {
            fprintf(fp, "%d ", -literal->num);
            printf("%d ", -literal->num);
        }
        literal = literal->next;
    }
    fprintf(fp, "0\n");
    statement = statement->next;
}

printf("文件打开成功! \n");
}

fclose(fp);
}
```

/*

将数独问题转化为 SAT 问题

算法：根据每个空格内有且仅有一个数字、1~9 中的数字在每行、每列和每个宫内出现且仅出现一次创建子句，通过邻接表记录每个空格中可能出现的数字

优化思路：用 9×10 的数组保存每个数字可能出现的位置

*/

```

int* TransfSud(int(*sud)[9], Formula*S, hole* holes) {
    int* Varies; //用于表示变元真值关系的数组
    int hole_num = 0; //用于记录空格数
    digital* p_d; //用于生成空格中可能出现的数字,在生成子句中用于遍历
    digital* p_df; //用于指向空格可能的数字的尾位置,在生成子句中用于遍历
    Statement* p_s; //用于创建子句
    Statement* p_sf = NULL; //用于遍历公式
    Literal* p_l; //用于创建文字
    Literal* p_lf = NULL; //用于遍历子句
    //用作循环变量
    int i;
    int j;
    int k;
    //用于保存行列
    int x;
    int y;

    //生成空格及其可能的数字邻接表
    for (i = 0; i < 9; i++) { //遍历格局找到每一个空格
        for (j = 0; j < 9; j++) {
            if (sud[i][j] == 0) {
                (holes + hole_num)->x = i;
                (holes + hole_num)->y = j;
                hole_num++;
            }
        }
    }
    for (i = 0; i < hole_num; i++) { //找到每个空格对应的数字
        holes[i].elem = NULL;
        p_df = NULL;
        holes[i].dig_num = 0;
        for (j = 1; j <= 9; j++) { //遍历检查 1~9 中哪些数可能出现在于此空格

```

```

bool found = false; //标志此数是否已出现
for (k = 0; k < 9 && (found == false); k++) { //在行中查找
    if (sud[holes[i].x][k] == j)
        found = true;
}
for (k = 0; k < 9 && (found == false); k++) { //在列中查找
    if (sud[k][holes[i].y] == j)
        found = true;
}
x = (holes[i].x) / 3; //用于记录空格所在的宫
y = (holes[i].y) / 3;
for (int a = 3 * x; (a >= 3 * x) && (a < 3 * (x + 1)) && (found == false);
a++) { //在宫中查找
    for (int b = 3 * y; (b >= 3 * y) && (b < 3 * (y + 1)) && (found ==
false); b++) {
        if (sud[a][b] == j)
            found = true;
    }
}
//为空格内的数字生成变元编号
if (found == 0) {
    p_d = (digital*)malloc(sizeof(digital));
    p_d->num = j; //记录该数字
    p_d->v_num = ++(S->num_v); //给数字一个变元编号
    p_d->next = NULL;
    holes[i].dig_num++; //空格可能出现的数字数目加一
    if (holes[i].elem == NULL) {
        holes[i].elem = p_d;
        p_df = p_d;
    }
    else {
        p_df->next = p_d;
    }
}

```

```

        p_df = p_d;
    }
}
}
}

```

//由空格内最多一个数字，最少一个数字 创建子句

```
for (i = 0; i < hole_num; i++) {
```

```
    p_d = holes[i].elem;
```

```
    p_df = p_d;
```

```
    p_s = (Statement*)malloc(sizeof(Statement));
```

```
    p_s->elem = NULL;
```

```
    p_s->next = NULL;
```

```
    S->num_st++;
```

```
    if (i == 0 && (S->root == NULL)) { //创建子句,当为第一个子句时，优先
```

考虑公式的根子句指针

```
        S->root = p_s;
```

```
    }
```

```
    else {
```

```
        p_sf->next = p_s;
```

```
    }
```

```
    p_sf = p_s;
```

```
    while (p_d) { //根据每个空格最少一个数字创建子句
```

```
        p_l = (Literal*)malloc(sizeof(Literal));
```

```
        p_l->next = NULL;
```

```
        p_sf->num_lit++; //子句文字数加一
```

```
        if (p_s->elem == NULL) {
```

```
            p_s->elem = p_l;
```

```
        }
```

```
        else {
```

```
            p_lf->next = p_l;
```

```
        }
```

```
        p_lf = p_l;
```

```

        p_lf->pos = 1; //文字设为正，以保证析取之后至少有一个为 1
        p_lf->num = p_d->v_num;
        p_d = p_d->next;
    }
    while (p_df) { //根据每个空格最多一个数字创建子句
        p_d = p_df->next;
        while (p_d) {
            p_s = (Statement*)malloc(sizeof(Statement)); //生成一个新子句
            p_s->next = NULL;
            S->num_st++;
            p_sf->next = p_s;
            p_sf = p_s;
            p_sf->num_lit = 2; //每个子句只有两个文字以代表只有一个文字
为真

            p_sf->elem = (Literal*)malloc(sizeof(Literal));
            p_sf->elem->num = p_df->v_num;
            p_sf->elem->pos = 0; //两个文字只有一个为真，故都是负文字
            p_sf->elem->next = (Literal*)malloc(sizeof(Literal));
            p_sf->elem->next->num = p_d->v_num;
            p_sf->elem->next->pos = 0;
            p_sf->elem->next->next = NULL;
            p_d = p_d->next;
        }
        p_df = p_df->next;
    }
}

//由每个数字在每行最多出现一次，最少出现一次 创建子句
j = 0;
for (i = 0; i < 9; i++) { //遍历每一行
    x = j; //用于记录每一行的开始空格位置
    for (k = 1; k <= 9; k++) { //遍历每个数字

```

```

int v[10]; //用于储存在行的空格上每个数字可能出现位置的变元号
v[0] = 0; //首行用于存储数字在此行出现次数

int a; //用于遍历 v 数字

int b;
a = 1;
j = x;

while ((holes[j].x == i) && j < hole_num) { //找到每个数字对应的空格
出现位置保持相应变元号

    p_df = holes[j].elem;
    while (p_df) {
        if (p_df->num == k) {
            v[a] = p_df->v_num;
            v[0]++;
            a++;
            break;
        }
        p_df = p_df->next;
    }
    j++;
}

if (v[0] == 0)
    continue;

p_s = (Statement*)malloc(sizeof(Statement)); //根据一个数字在一行内
最少出现一次生成公式

S->num_st++; //子句数加一

p_s->num_lit = v[0];
p_s->next = NULL;
p_sf->next = p_s;
p_sf = p_s;
for (a = 1; a <= v[0]; a++) {
    p_l = (Literal*)malloc(sizeof(Literal));
    p_l->next = NULL;
    if (a == 1) {

```

```

        p_sf->elem = p_l;
    }
    else {
        p_lf->next = p_l;
    }
    p_lf = p_l;
    p_lf->pos = 1;//保证只有数字最少出现一次，故此子句的文字全
    为正
    p_lf->num = v[a];
}
for (a = 1; a <= v[0]; a++) { //根据一个数字在一行内最多出现一次生
    成公式

```

```

        for (b = a + 1; b <= v[0]; b++) {
            p_s = (Statement*)malloc(sizeof(Statement));
            S->num_st++; //子句数加一
            p_s->num_lit = 2;
            p_s->next = NULL;
            p_sf->next = p_s;
            p_sf = p_s;
            p_sf->elem = (Literal*)malloc(sizeof(Literal));
            p_sf->elem->num = v[a];
            p_sf->elem->pos = 0; //两个文字只有一个为真，故都是负文字
            p_sf->elem->next = (Literal*)malloc(sizeof(Literal));
            p_sf->elem->next->num = v[b];
            p_sf->elem->next->pos = 0;
            p_sf->elem->next->next = NULL;
        }
    }
}
}

```

//由每个数字在每列最多出现一次，最少出现一次 创建子句

```

for (j = 0; j < 9; j++) {

```



```

for (k = 1; k <= 9; k++) { //遍历每个数字

    int v[10]; //用于储存在列的空格上每个数字可能出现位置的变元号
    v[0] = 0; //首行用于存储数字在此列出现次数

    int a; //用于遍历 v 数字
    a = 1;
    int b;

    for (i = 0; i < hole_num; i++) { //找到每个数字对应的空格出现位置保
保持相应变元号

        if (holes[i].y == j) {
            p_df = holes[i].elem;
            while (p_df) {
                if (p_df->num == k) {
                    v[a] = p_df->v_num;
                    v[0]++;
                    a++;
                    break;
                }
                p_df = p_df->next;
            }
        }
    }

    if (v[0] == 0)
        continue;

    p_s = (Statement*)malloc(sizeof(Statement)); //根据一个数字在一行内
最少出现一次生成公式

    S->num_st++; //子句数加一
    p_s->num_lit = v[0];
    p_s->next = NULL;
    p_sf->next = p_s;
    p_sf = p_s;
    for (a = 1; a <= v[0]; a++) {
        p_l = (Literal*)malloc(sizeof(Literal));

```

```

p_l->next = NULL;
if (a == 1) {
    p_sf->elem = p_l;
}
else {
    p_lf->next = p_l;
}
p_lf = p_l;
p_lf->pos = 1; //保证只有数字最少出现一次，故此子句的文字全

```

为正

```

p_lf->num = v[a];
}
for (a = 1; a <= v[0]; a++) { //根据一个数字在一行内最多出现一次生

```

成公式

```

for (b = a + 1; b <= v[0]; b++) {
    p_s = (Statement*)malloc(sizeof(Statement));
    S->num_st++; //子句数加一
    p_s->num_lit = 2;
    p_s->next = NULL;
    p_sf->next = p_s;
    p_sf = p_s;
    p_sf->elem = (Literal*)malloc(sizeof(Literal));
    p_sf->elem->num = v[a];
    p_sf->elem->pos = 0; //两个文字只有一个为真，故都是负文字
    p_sf->elem->next = (Literal*)malloc(sizeof(Literal));
    p_sf->elem->next->num = v[b];
    p_sf->elem->next->pos = 0;
    p_sf->elem->next->next = NULL;
}
}
}
}

```

//由每个数字在每个子宫内最多出现一次，最少出现一次 创建子句

```
int place;//用于遍历宫
for (place = 0; place < 9; place++) {
    int ub_x;//表示宫的上界
    int ub_y;
    int lb_x;//表示宫的下界
    int lb_y;
    lb_x = (place / 3) * 3;
    ub_x = (place / 3 + 1) * 3;
    switch (place) { //确定当前宫对应的上下界
        case 0:
        case 3:
        case 6:
        {
            lb_y = 0;
            ub_y = 3;
            break;
        }
        case 1:
        case 4:
        case 7:
        {
            lb_y = 3;
            ub_y = 6;
            break;
        }
        case 2:
        case 5:
        case 8:
        {
            lb_y = 6;
            ub_y = 9;
            break;
        }
    }
```

```

    }
    for (k = 1; k <= 9; k++) //遍历每个数字
    {
        int v[10]; //用于储存在列的空格上每个数字可能出现位置的变元号
        v[0] = 0; //首行用于存储数字在此列出现次数
        int a; //用于辅助遍历
        int b;
        a = 0;
        b = 1;
        for (a = 0; a < hole_num; a++) { //找到每个数字对应的空格出现位置
保持相应变元号
            if ((holes[a].x >= lb_x) && (holes[a].x < ub_x) && (holes[a].y >=
lb_y) && (holes[a].y < ub_y)) { //当空格在当前宫内时
                p_df = holes[a].elem;
                while (p_df) { //遍历空格中可能出现的数字
                    if (p_df->num == k) { //如果恰为当前数字则计入数组
                        v[b] = p_df->v_num;
                        v[0]++;
                        b++;
                        break;
                    }
                    p_df = p_df->next;
                }
            }
        }
        if (v[0] == 0)
            continue;
        p_s = (Statement*)malloc(sizeof(Statement)); //根据一个数字在一行内
最少出现一次生成公式
        S->num_st++; //子句数加一
        p_s->num_lit = v[0];
        p_s->next = NULL;
    }
}

```

```

p_sf->next = p_s;
p_sf = p_s;
for (a = 1; a <= v[0]; a++) {
    p_l = (Literal*)malloc(sizeof(Literal));
    p_l->next = NULL;
    if (a == 1) {
        p_sf->elem = p_l;
    }
    else {
        p_lf->next = p_l;
    }
    p_lf = p_l;
    p_lf->pos = 1; //保证只有数字最少出现一次，故此子句的文字全
    p_lf->num = v[a];
}

```

为正

```

for (a = 1; a <= v[0]; a++) { //根据一个数字在一行内最多出现一次生

```

成公式

```

for (b = a + 1; b <= v[0]; b++) {
    p_s = (Statement*)malloc(sizeof(Statement));
    S->num_st++; //子句数加一
    p_s->num_lit = 2;
    p_s->next = NULL;
    p_sf->next = p_s;
    p_sf = p_s;
    p_sf->elem = (Literal*)malloc(sizeof(Literal));
    p_sf->elem->num = v[a];
    p_sf->elem->pos = 0; //两个文字只有一个为真，故都是负文字
    p_sf->elem->next = (Literal*)malloc(sizeof(Literal));
    p_sf->elem->next->num = v[b];
    p_sf->elem->next->pos = 0;
    p_sf->elem->next->next = NULL;
}

```

```

    }
}
}

//由变元数生成变元真值关系数组
Varies = (int*)malloc((S->num_v + 1) * sizeof(int));
for (i = 1; i < (S->num_v + 1); i++)
    Varies[i] = -1;
return Varies;
}

/*
将 cnf 变元真值关系转化为数独的解并输出
输入为一组 cnf 变元真值关系
输出为数独的解
算法思想：输出数独元素，当此元素为 0 时，找到空格数组中对应的空格，遍历
查看空格中那个数字对应的变元为真则输出此数字
*/
void ShowSudResult(int* Varies, hole* holes, int(*sud)[9]) {
    int i;//用于遍历数独格局
    int j;
    int k = 0;//用于遍历空格数组
    digital* p;

    printf("\n*****\n\n");
    printf("数独的解为： \n\n");
    printf("-----\n");
    for (i = 0; i < 9; i++) {
        printf("\t");
        for (j = 0; j < 9; j++) {
            if (sud[i][j] == 0) { //当此处为空时
                p = holes[k++].elem;
            }
        }
    }
}

```

```
        while (p) {
            if (Varies[p->v_num]) {
                sud[i][j] = p->num;
                break;
            }
            else
                p = p->next;
        }
        printf("%d", sud[i][j]);
    }
    printf("\n");
}
printf("-----\n\n");
}
```