

**课程设计报告**

**题目： 基于SAT的数独游戏求解程序**

**课程名称： 程序设计综合课程设计**

**专业班级： 计算机1705班**

**学 号： U201714726**

**姓 名： 王明明**

**指导教师： 李丹**

**报告日期： 2019.3.20**

**计算机科学与技术学院**

目 录

[任 务 书 2](#_Toc3892992)

[设计内容 2](#_Toc3892993)

[设计要求 2](#_Toc3892994)

[1 引 言 3](#_Toc3892995)

[1.1 问题背景 3](#_Toc3892996)

[1.2 研究意义 4](#_Toc3892997)

[1.3 本文摘要 4](#_Toc3892998)

[2. 系统总体设计 5](#_Toc3892999)

[2.1 有关常量和变量定义 5](#_Toc3893000)

[2.2 一些函数的算法设计 6](#_Toc3893001)

[2.3 程序源代码 11](#_Toc3893002)

[3. SAT问题求解 11](#_Toc3893003)

[3.1 总体构想 11](#_Toc3893004)

[3.2 存储结构 11](#_Toc3893005)

[3.3 DPLL算法思想 13](#_Toc3893006)

[3.4 DPLL改进思路 14](#_Toc3893007)

[4. 算例测试 16](#_Toc3893008)

[4.1 CNF算例求解 16](#_Toc3893009)

[4.2 数独游戏测试 29](#_Toc3893010)

[4.3 分析 29](#_Toc3893011)

[5. 感悟与总结 30](#_Toc3893012)

[6. 致谢 31](#_Toc3893013)

[参考文献 32](#_Toc3893014)

[附录A main.c 33](#_Toc3893015)

[附录B SAT.h 34](#_Toc3893016)

[附录C satSolve.c 37](#_Toc3893017)

[附录D Sudoku.h 56](#_Toc3893018)

[附录E Game.c 57](#_Toc3893019)

# 任 务 书

## 设计内容

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

## 设计要求

要求具有如下功能：

1. **输入输出功能：**包括程序执行参数的输入，SAT算例cnf文件的读取，执行结果的输出与文件保存等。(15%)
2. **公式解析与验证：**读取cnf算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。数据结构的设计可参考文献[1-3]。(15%)
3. **DPLL过程：**基于DPLL算法框架，实现SAT算例的求解。(35%)
4. **时间性能的测量：**基于相应的时间处理函数（参考time.h），记录DPLL过程执行时间（以毫秒为单位），并作为输出信息的一部分。(5%)
5. **程序优化：**对基本DPLL的实现进行存储结构、分支变元选取策略[1-3]等某一方面进行优化设计与实现，提供较明确的性能优化率结果。优化率的计算公式为：[(t-to)/t]\*100%,其中t 为未对DPLL优化时求解基准算例的执行时间，to则为优化DPLL实现时求解同一算例的执行时间。(15%)
6. **SAT应用：**将数独游戏[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%)

# 1 引 言

## 问题背景

SAT问题又称命题逻辑公式的可满足性问题（satisfiability problem），是判断对合取范式形式给出的命题逻辑公式是否存在一个真值指派使得该逻辑公式为真。SAT问题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。看似简单，却可广泛应用于许多实际问题如人工智能、电子设计自动化、自动化推理、硬件设计、安全协议验证等，具有重要理论意义与应用价值。对于SAT问题的研究从没有停止过，在1997年和2003年，H.Kautz与B.Selman两次列举出SAT搜索面临的挑战性问题，并于2011年和2007年，两度对当时的SAT问题研究现状进行了全面的综述。黄文奇提出的Solar算法在北京第三届SAT问题快速算法比赛中获得第一名。对SAT问题的求解主要有完备算法和不完备算法两大类。不完备算法主要是局部搜索算法，这种算法不能保证一定找到解，但是求解速度快，对于某些SAT问题的求解，局部搜索算法要比很多完备算法更有效。完备算法出现的时间更早，优点是可以正确判断SAT问题的可满足性，在算例无解的情况下可以给出完备的证明。对于求解SAT问题的优化算法主要有启发式算法、冲突子句学习算法、双文字监视法等。

## 研究意义

SAT问题是第一个被证明的NP完全问题，而NP完全问题由于其极大的理论价值和困难程度，破解后将会在许多领域得到广泛应用，从而在计算复杂性理论中具有非常重要的地位。由于所有的NP完全问题都能够在多项式时间内进行转换，那么如果SAT问题能够得到高效解决，所有的NP完全问题都能够在多项式时间内得到解决。对SAT问题的求解，可用于解决计算机和人工智能领域内的CSP问题（约束满足问题）、语义信息的处理和逻辑编程等问题，也可用于解决计算机辅助设计领域中的任务规划与设计、三维物体识别等问题。SAT问题的应用领域非常广泛，还能用于解决数学研究和应用领域中的旅行商问题和逻辑算数问题。许多实际问题，例如数据库检索、积木世界规划、超大规模集成电路设计、人工智能等都可以转换成SAT问题进而进行求解。可见对SAT问题求解的研究，具有重大意义。

## 本文摘要

依据Davis和Putnam在1960年提出的DPLL算法来求解合取范式，并基于原始的DPLL算法提出优化方案，并对优化前后的DPLL算法进行比较，做出一些思考。本文首先介绍关于SAT问题的背景、研究意义等，接着介绍系统总体的设计，有关常量和数据元素类型的定义等，接着给出一些函数的设计思路，紧接着对未改近的DPLL算法和改进之后的DPLL算法思路进行介绍，源代码注重模块化，各有分工，最后进行分析与总结与致谢。

# 2. 系统总体设计

## 2.1有关常量和变量定义

有关常量的定义：

#define MaxNumVar 4000

#define PreassignVar 18

//预先分配的变元数目

#define CONFLICT 0

#define SATISFIABLE 1

#define UNSATISFIABLE 0

#define OTHERS 2

#define SINGLE -1

//UNKNOWN表示该变元未知，NONE表示该变元不存在，FALSE表示该变元为假，TRUE表示该变元为真

#define FALSE -1

#define TRUE 1

#define UNKNOWN 0

#define NONE 2

有关变量的定义：

int numVar;

int knownVar;

int numBranch;

int firstBranch[MaxNumVar];

有关数据结构的定义：

typedef struct varWatch {

struct varList \*pos;

struct varList \*neg;

} Var\_watch;

typedef struct varList {

struct clause \*p;

struct varList \*next;

} VarList;

typedef struct clause {

struct clauseLiteral \*p;

struct clause \*nextClause;

} Clause;

typedef struct clauseLiteral {

int data;

struct clauseLiteral \*next;

} ClauseLiteral;

typedef struct satAnswer {

int branchLevel[MaxNumVar + 1];

int value[MaxNumVar + 1];

int searched[MaxNumVar + 1];

int singleClause[MaxNumVar + 1];

} SatAnswer;

## 2.2一些函数的算法设计

(1)函数名称：InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision)

操作结果：初始化数据元素类型。

设计思想：逐行清空，直到指针为空，流程图如图2-1 InitSat所示。



图2-1 InitSat简易流程图

(2)函数名称：LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp)

操作结果：加载cnf文件中的内容，读取变元个数，子句数目等

设计思想：先找到标志cnf文件内容开始的符号“p cnf ”，接着调用函数GetNum，直到文件结尾标志出现。流程图如图2-2 LoadCnf所示。



图2-2 LoadCnf简易流程图

(3)函数名称：PrintAnswer(SatAnswer \*answer, int result, char filename[100], int duration);

操作结果：将求解SAT问题的结果打印并保存

设计思想：首先设置文件指针创建一个与源文件同名不同后缀的文件，紧接着根据value值的不同选择打印的值为正数还是负数，直到文件结束。流程图如图2-3所示。



图2-3 PrintAnswer简易流程图

(4)函数名称：NextBranch(int branchDecision[], SatAnswer \*answer)

操作结果： 选出下一个决策变量。

设计思想： 遍历子句，统计每个文字出现次数并记录到对应的计数器数组中。

(5)函数名称：ChectAnswer(Clause \*S, SatAnswer \*answer)

操作结果：检查计算结果是否正确，并将结果保存。

设计思想：传入计算结果和子句结构S，将计算结果代入检查，输出检查结果并创建文本文件用来保存结果。  
 (6)函数名称：PutClause(Clause \*ctemp,int var, Var\_watch var\_watch[])

操作结果：在文字邻接表最后添加子句。

设计思想：先找到文字邻接表的末尾，传入子句，指针初始化。流程图如图2-4 所示。



图2-4 PutCluse流程图

## 2.3程序源代码

见《附录A main.c》、《附录B SAT.h》、《附录C satSolve.c》、《附录D Sudoku.h》、《附录E Game.c》。

# 3. SAT问题求解

## 3.1总体构想

整个工程包含main.c、SAT.h、satSolve.c、Sudoku.h、Game.c五个文件，其中main.c用于程序主界面的显示和主要函数的调用、SAT.h是声明关于求解SAT问题有关的函数以及一些变量的声明、数据元素类型的定义、satSolve.c是求解SAT问题有关函数的实现、Sudoku.h是关于求解数独以及数独游戏函数的声明、Game.c是数独游戏的实现。

五个文件中关于求解SAT问题的函数有InitSat、Sat、LoadCnf、GetNum、DPLL、PrintAnswer、PutClause、Deduce、SingleClauseDeduce、NextBranch、Analyse\_conflict、ChectAnswer。其中InitSat用于数据元素类型等的初始化、Sat用于调用函数、LoadCnf用于cnf文件的调用、GetNum以读取字符的形式得到cnf文件的内容如变元数子句数等、DPLL用于求解SAT问题的框架、PrintAnswer是用来打印和保存结果的、PutClause用于添加子句、Deduce和SingleClauseDeduce类似于一颗树和树上的节点便于回溯、NextBranch是确定下一分支变量、Analyse\_conflict、ChectAnswer是用来回溯的。

关于数独游戏的函数主要有Sudoku、NewSudoku、GenerateSudoku、DigHole、SolveSudoku。其中Sudoku用于函数调用、NewSudoku 和GenerateSudoku用于生成新的数独、DigHole用于挖洞、SolveSudoku用于将数独转化为cnf文件并求解。

## 3.2 存储结构

存储结构及其注释如下：

typedef struct varWatch {

struct varList \*pos; //文字邻接表，正文字邻接表，负文字邻接表，

struct varList \*neg;

} Var\_watch;

typedef struct varList {

struct clause \*p; //指向一个子句

struct varList \*next; //指向下一个包含该文字的子句

} VarList;

typedef struct clause {

struct clauseLiteral \*p; //指向一个子句中的下一个文字

struct clause \*nextClause; //这个在邻接表中无意义，要从读取CNF文件时看出效果

} Clause;

typedef struct clauseLiteral {

int data;//文字的值

struct clauseLiteral \*next; //指向子句中的下一个文字

} ClauseLiteral;

typedef struct satAnswer {

int branchLevel[MaxNumVar + 1]; //赋值时的决策树高度

int value[MaxNumVar + 1]; //TRUE or FALSE or UNKNOWN or NONE

int searched[MaxNumVar + 1]; //已被搜索的情况数

int singleClause[MaxNumVar + 1]; //标记是否存在该变量的单子句

} SatAnswer;

该存储结构为文字邻接表，数据元素类型有文字、变元、子句、表格。首先，每一个变元都有正负两个文字，用pos表示正文字，neg表示负文字，变元用数组进行存储，每一个变元具有两个类似于表格的结构，分别存储出现该变元正文字的所有子句和出现该变元负文字的所有子句，表格的每一行代表一条子句，将子句的开头连接起来构成list。

可用下图来表示：

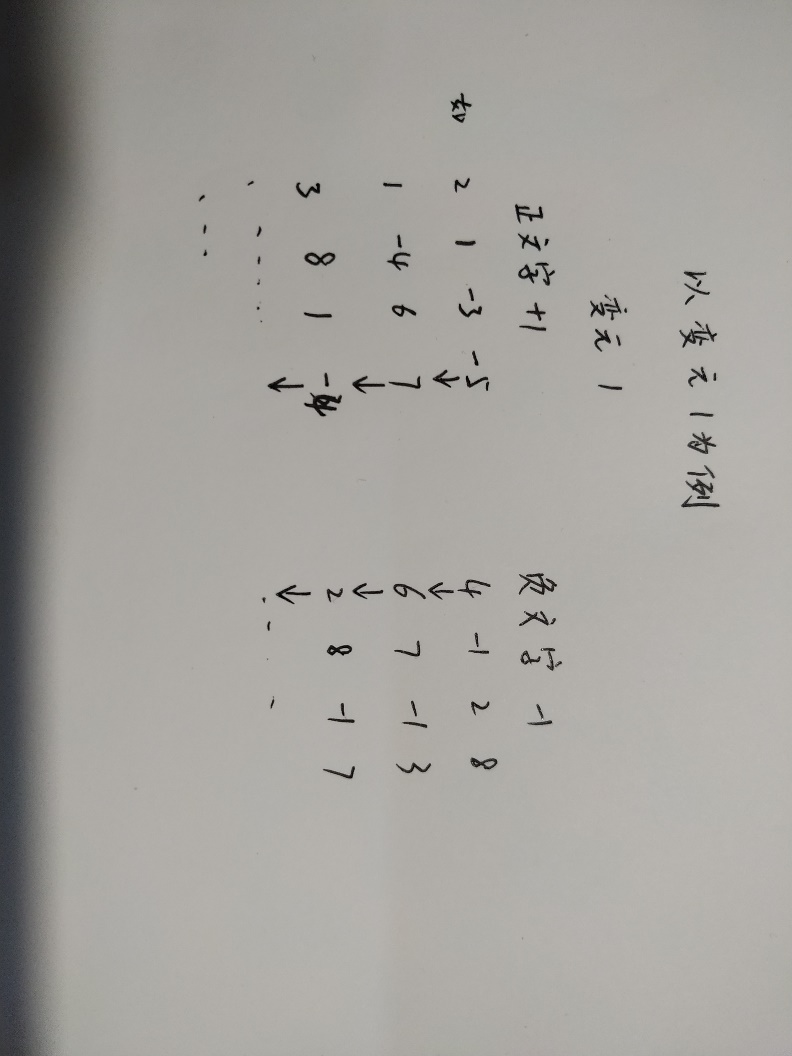


图3-1 数据结构图

## 3.3 DPLL算法思想

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：

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*¬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。

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*¬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2.2所示。



图3-2 DPLL算法搜索树

基于单子句传播与分裂策略的DPLL算法可以描述为一个如后所示的递归过程DPLL( *S* ),为了优化执行效率，一般用非递归实现。

DPLL( *S*) :

/\* *S*为公式对应的子句集。若其满足，返回TURE；否则返回FALSE. \*/

{

while(*S*中存在单子句) {//单子句传播

在*S*中选一个单子句*L*；

依据单子句规则，利用*L*化简*S*；

if *S* = Φ return(TRUE);

else if (*S*中有空子句 ) return（FALSE）；

}//while

基于某种策略选取变元*v*；//策略对DPLL性能影响很大

if DPLL（*S* ∪*v* ）return(TURE);

return DPLL(*S* ∪¬*v*);

}

## 3.4 DPLL改进思路

我们首先假定，在初始状态下，不存在已经被赋值的变量，我们称未被赋值的变量为自由变量。主循环通过给一个自由变量赋值，从而进入一个分支，这种对变量进行赋值的操作被称为判定，这个被赋值的变量也就有了一个与之对应的判定级。判定级从1开始，随之后的分支变量出现而增长。这些操作由迭代算法中的 NextBranch()方法完成。

进入分支后，问题可以简化为由这一判定及其推论决定。deduce()方法完成一些必要的推理，以便得出在当前的判定级下，为了使被判定实例满足而需要进行的变量赋值。在分支后由该分支判定的推论得到的被赋值变量，被定义为与该分支变量处于同一判定级。

推理结束后，如果所有变量已经被赋值且所有子句都被满足，则返回 SATISFIABLE：如果在推理后的赋值状态下，存在矛盾子句，则当前判定分支及其子分支都不可能是能够使实例被满足的变量赋值，这时求解器会进行回退操作。回退到0判定级则表示即使未进入任何一个判定分支，该输入实例仍然是不可满足的，在这种情况下，求解器将给出该被求解实例不可满足的结论。其中回溯操作回退到上一判定级，如果选定的赋值变量的两个文字均已经推理完毕，则再次回退到上一判定级。

迭代的DPLL算法中的NextBranch()方法决定下一个分支方向。文字统计启发式算法，该算法对给定的变量在两种赋值状态(true或false)下，会导致的未被满足子句数量进行统计。在实际应用中，这种启发式算法，即通过计算同一变量文字的不同逻辑值的动态最大组合总和来选定分支变量的启发式算法，在标准测试数据上可以得到很好的结果。

关于NextBranch()方法的具体算法描述如下

(1)每一变量的两个文字(true和false)都对应一个计数器，它们的初始值为0。

(2)当一个子句被加入到子句库中时，子句中每个文字对应的计数器值增加。

(3)每次判定时，拥有最高计数器值的未被赋值变量及它的分支逻辑值会被选出作为下一个分支

(4)默认情况下，这种数值会被随机破坏。这一点是可以改变默认配置的。

(5)所有计数器的值会被周期性地减少一个常量。

# 4. 算例测试

程序采用简易界面，如图所示：

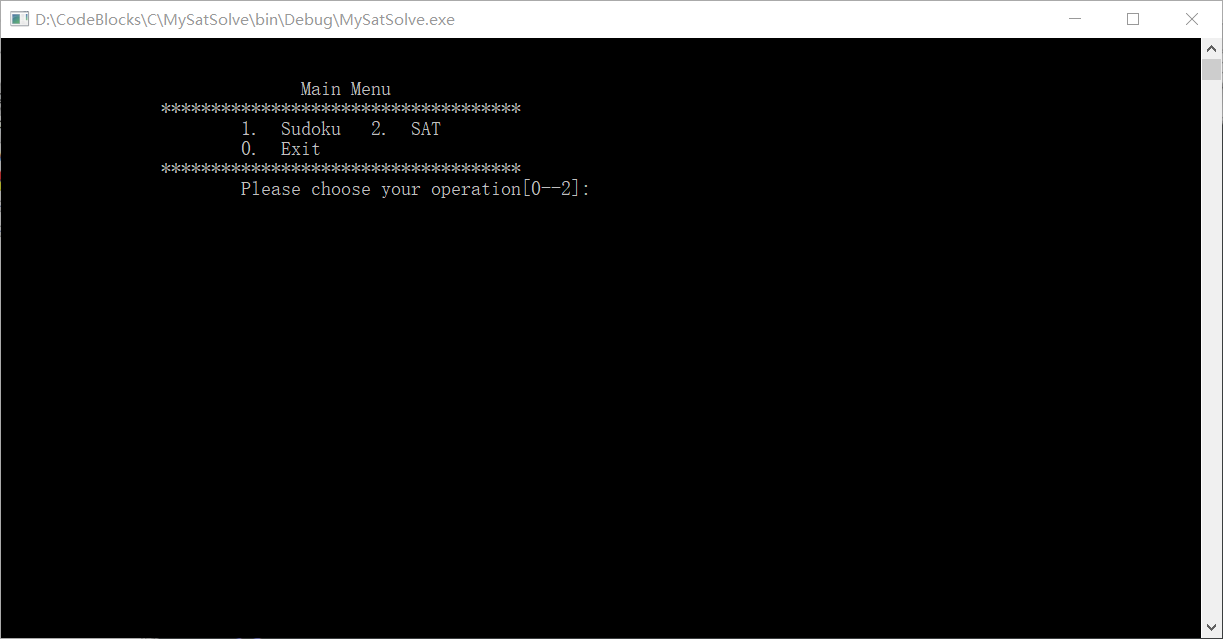


图4-1 程序主界面

## 4.1 CNF算例求解

CNF基准算例 表4-1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例文件夹 | 算例名称 | 优化前(ms) | 优化后(ms) | 优化率 |
| 20 | 91 | 功能测试 | sat-20.cnf | 0 | 0 | 0 |
| 30 | 420 | 功能测试 | unsat-5cnf-30.cnf | No solution | No solution |  |
| 181 | 3151 | 性能测试 | ais10.cnf | 0 | 50 |  |
| 303 | 2851 | 性能测试 | sud00009.cnf | 4 | 26 | -550% |

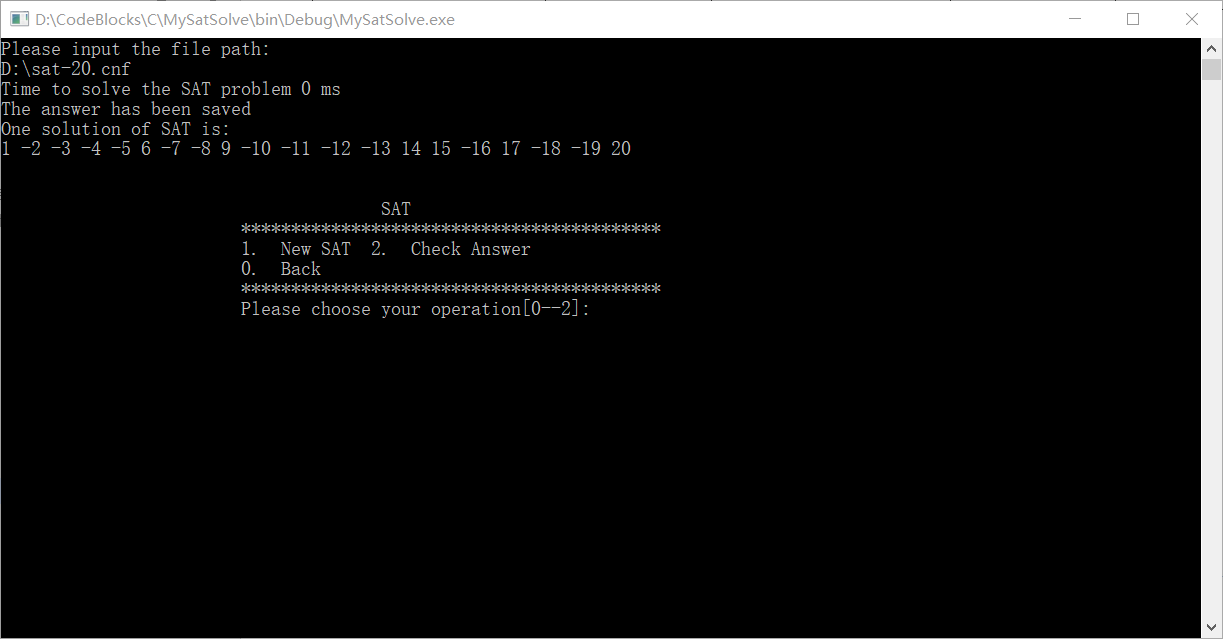


图4-2 算例sat-20.cnf未优化

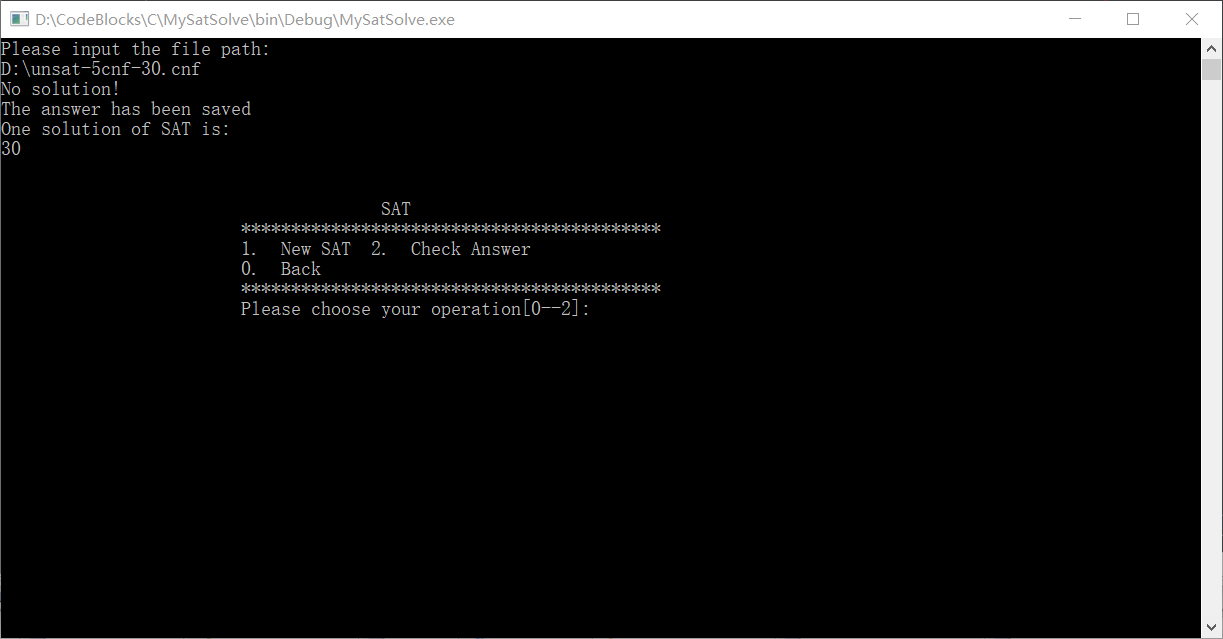


图4-3 unsat-5cnf-30.cnf算例未优化

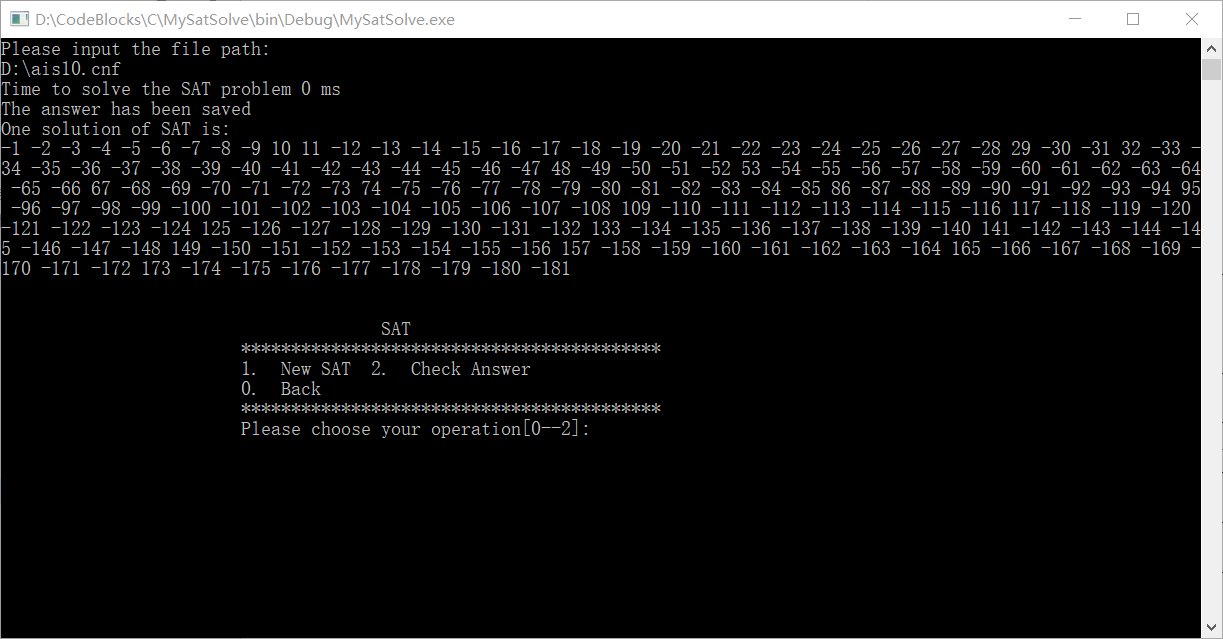


图4-4 ais10.cnf算例未优化

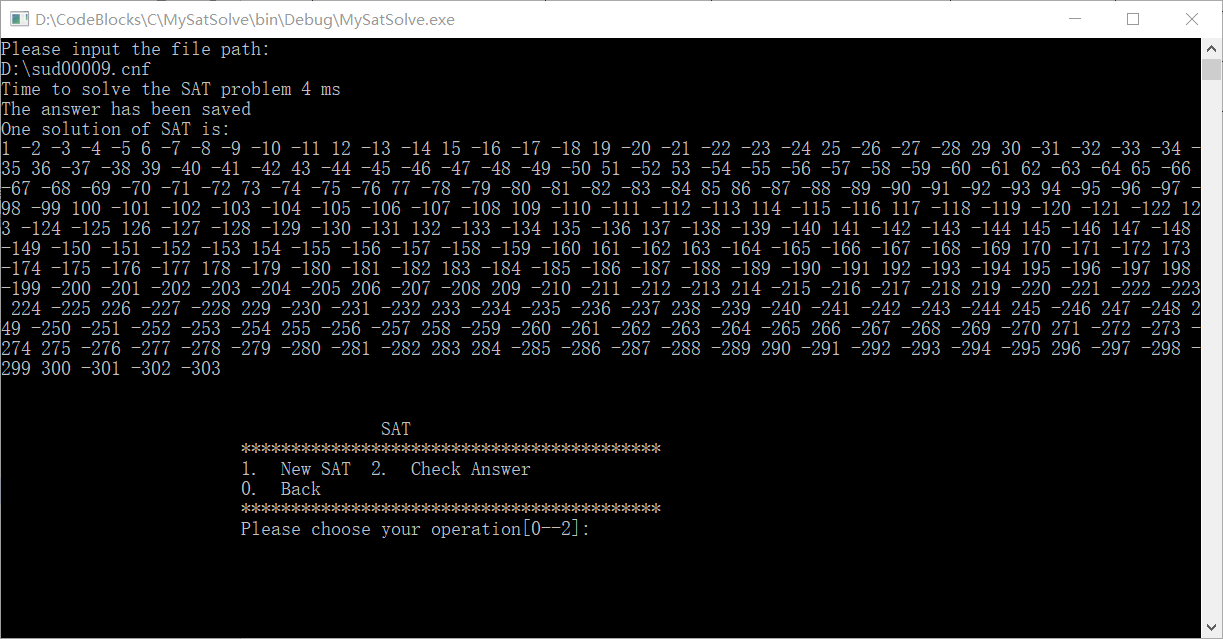


图4-5 sud00009.cnf算例未优化

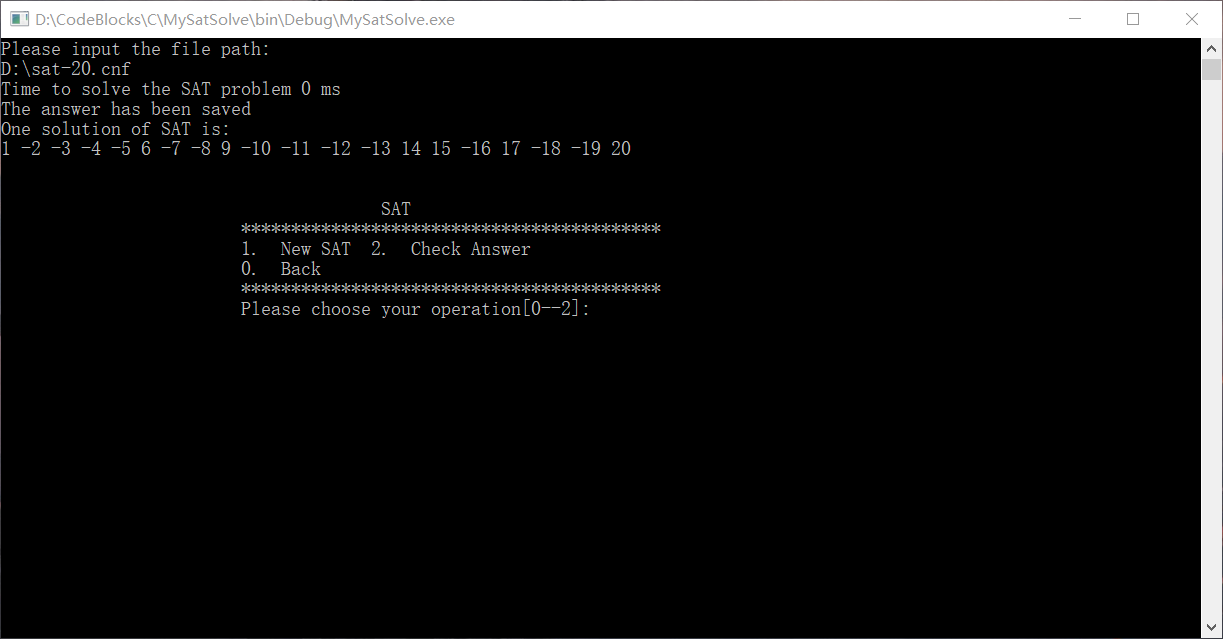


图4-6 算例sat-20.cnf优化后

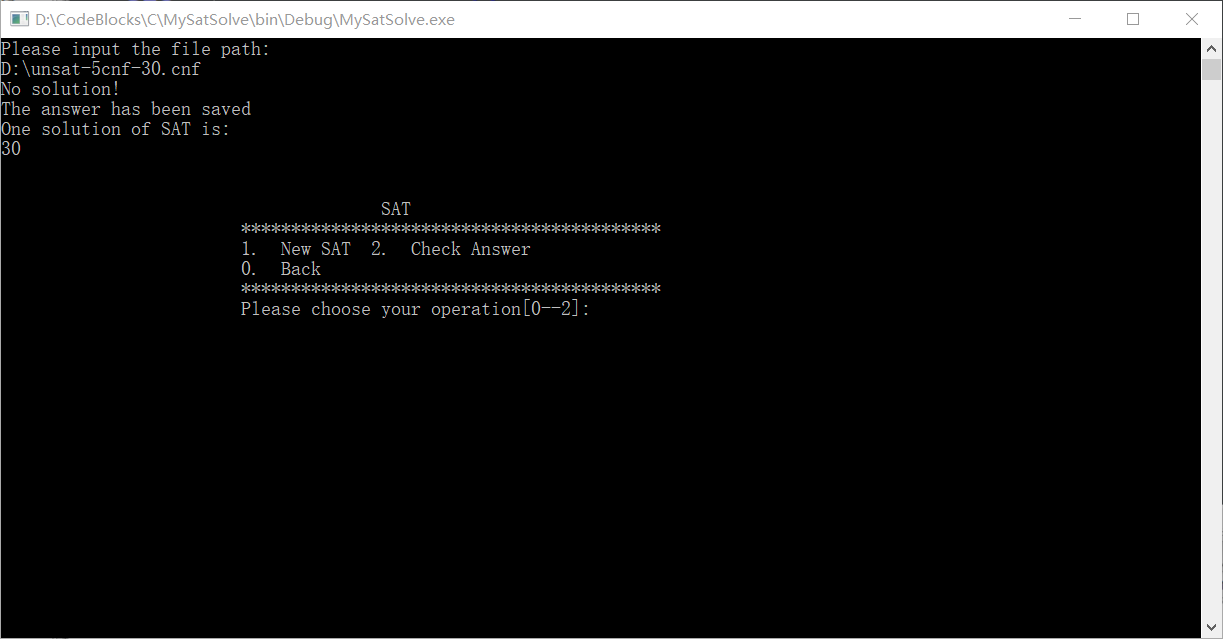


图4-7 unsat-5cnf-30.cnf算例优化后

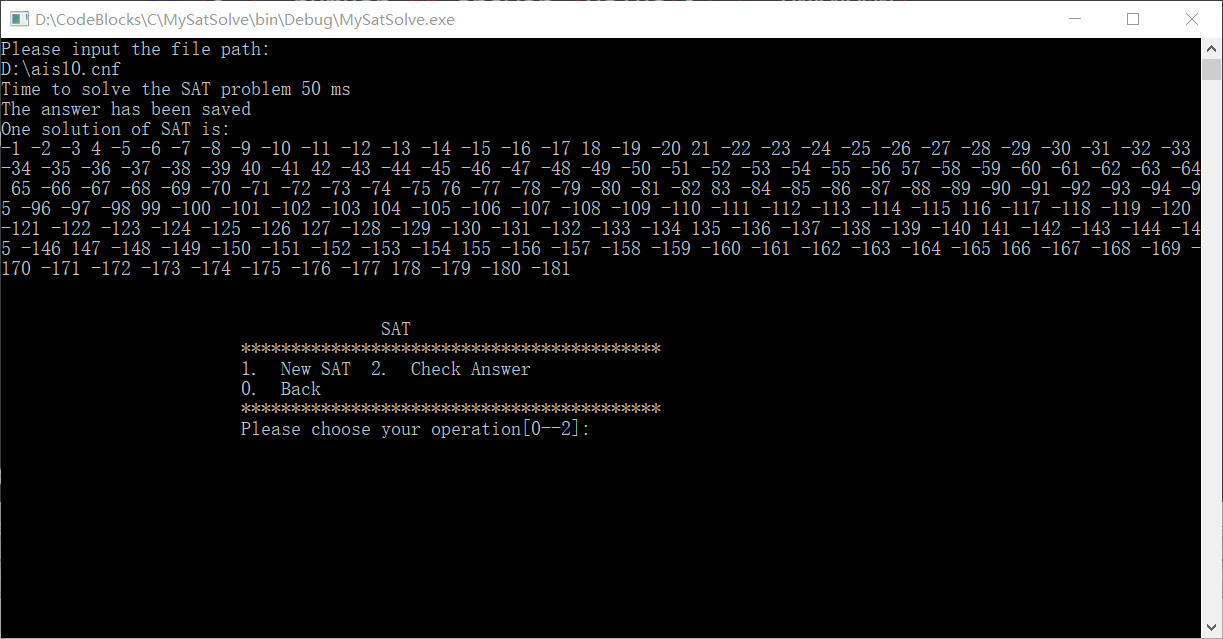


图4-8 ais10.cnf算例优化后

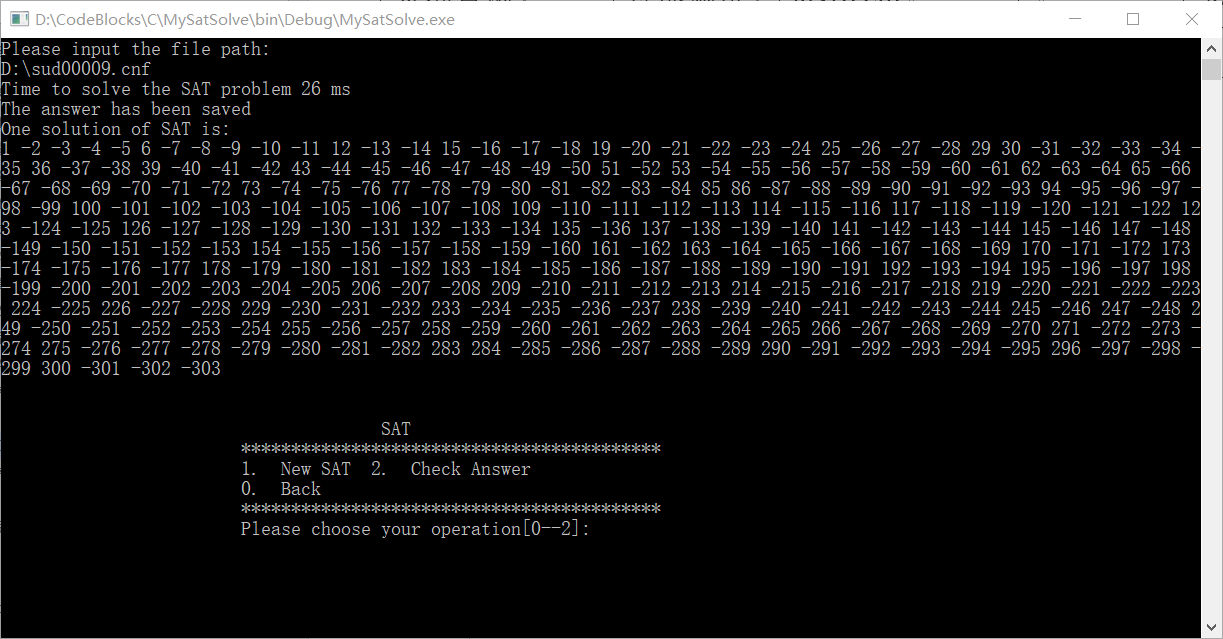


图4-9 sud00009.cnf算例优化后

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例属性 | 算例名称 | 优化前（ms） | 优化后（ms） | 优化率 |
| 20 | 91 | S | problem1-20.cnf | 0 | 1 |  |
| 50 | 80 | S | problem2-50.cnf | 2 | 0 | 100% |
| 100 | 340 | S | problem3-100.cnf | 434 | 3 | 99% |
| 50 | 100 | S | problem6-50.cnf | 8 | 0 | 100% |
| 50 | 300 | S | problem8-50.cnf | 0 | 0 |  |
| 100 | 200 | S | problem9-100.cnf | 76765 | 0 | 100% |
| 100 | 600 | S | problem11-100.cnf | 2 | 0 | 100% |
| 25 | 100 | S | tst\_v25\_c100.cnf | 0 | 0 |  |
| 20 | 1532 | S | 7cnf20\_90000\_90000\_7  .shuffled-20.cnf | 111 | 48 | 57% |

CNF算例 表4-2

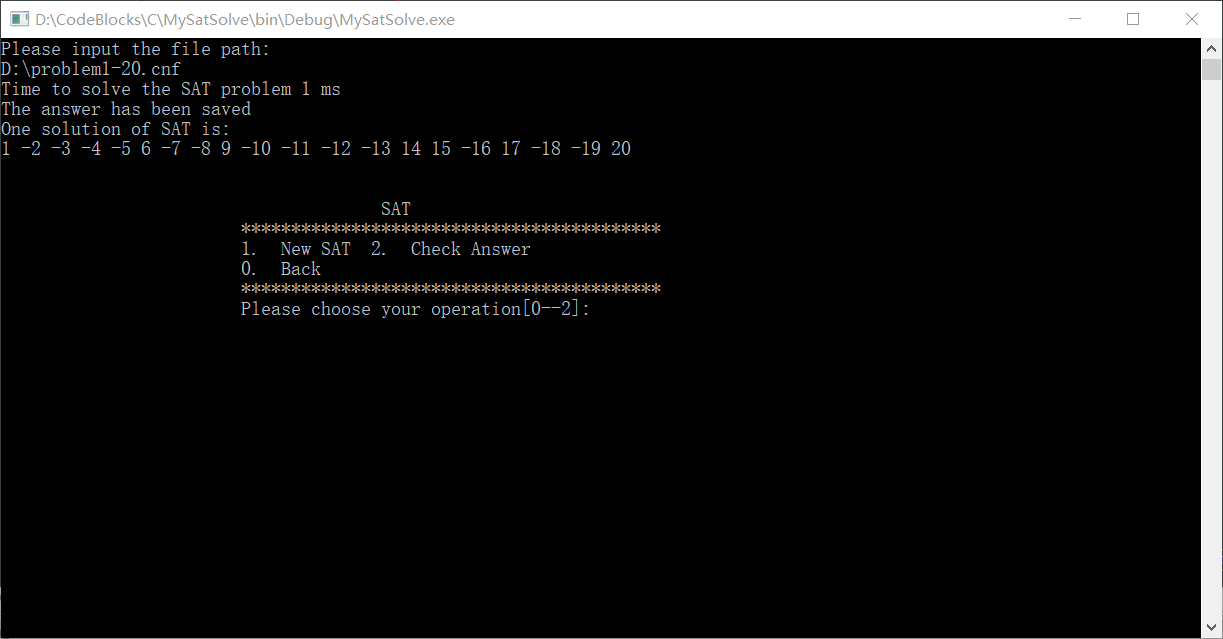


图4-10 problem1-20.cnf算例优化后

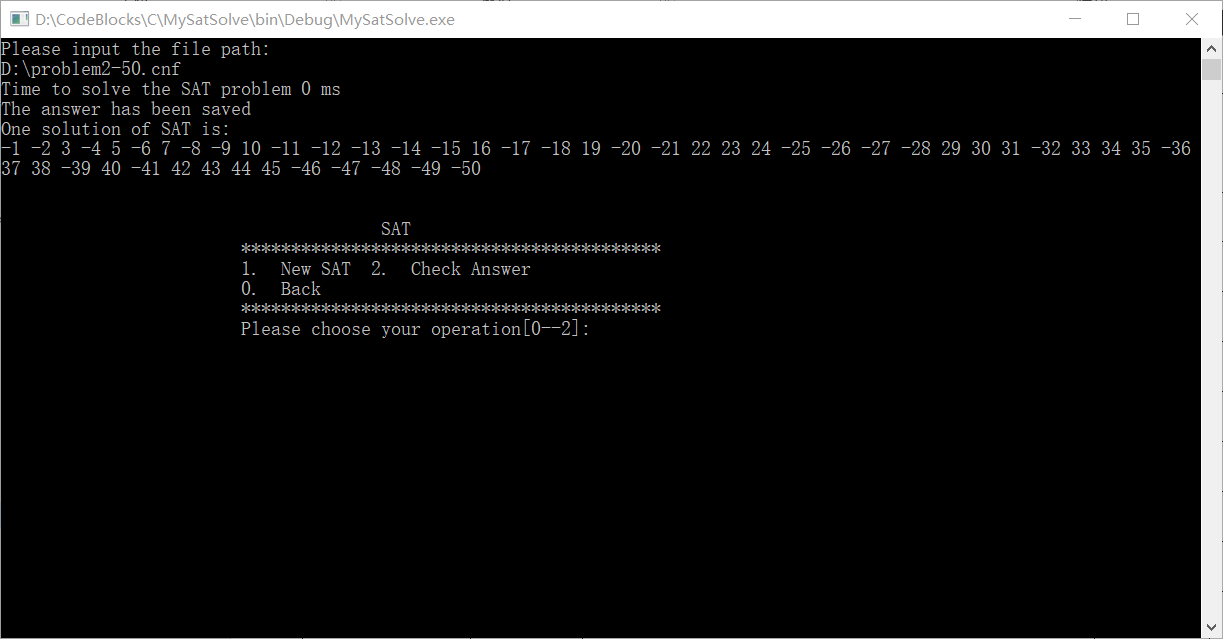


图4-11 problem2-50.cnf算例优化后

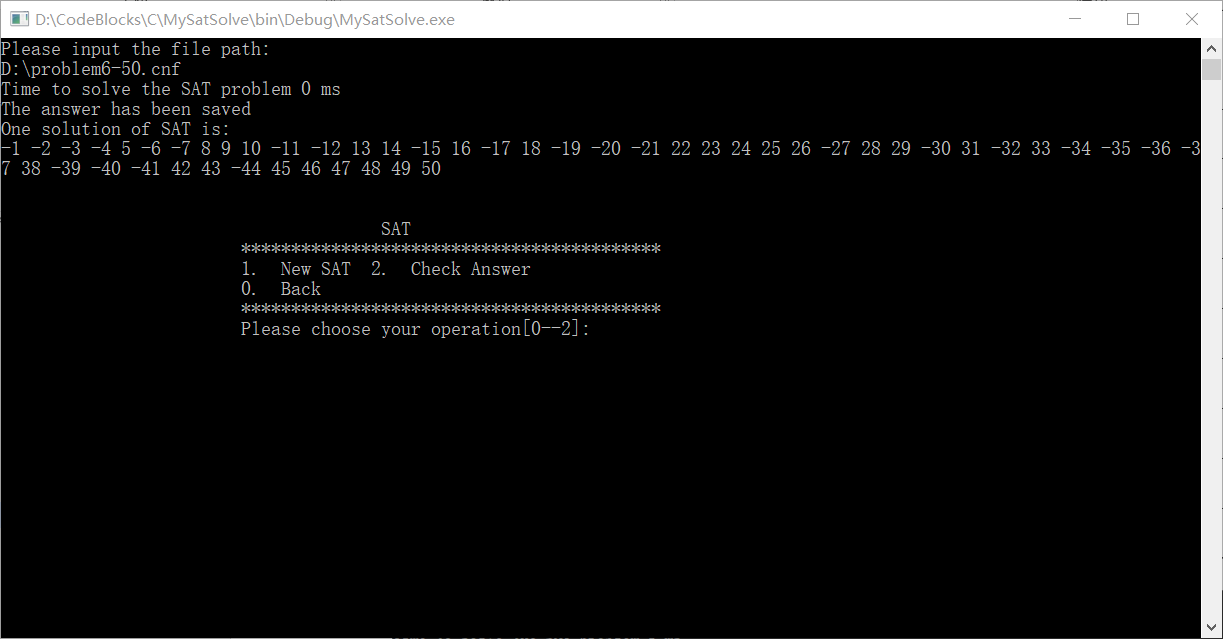
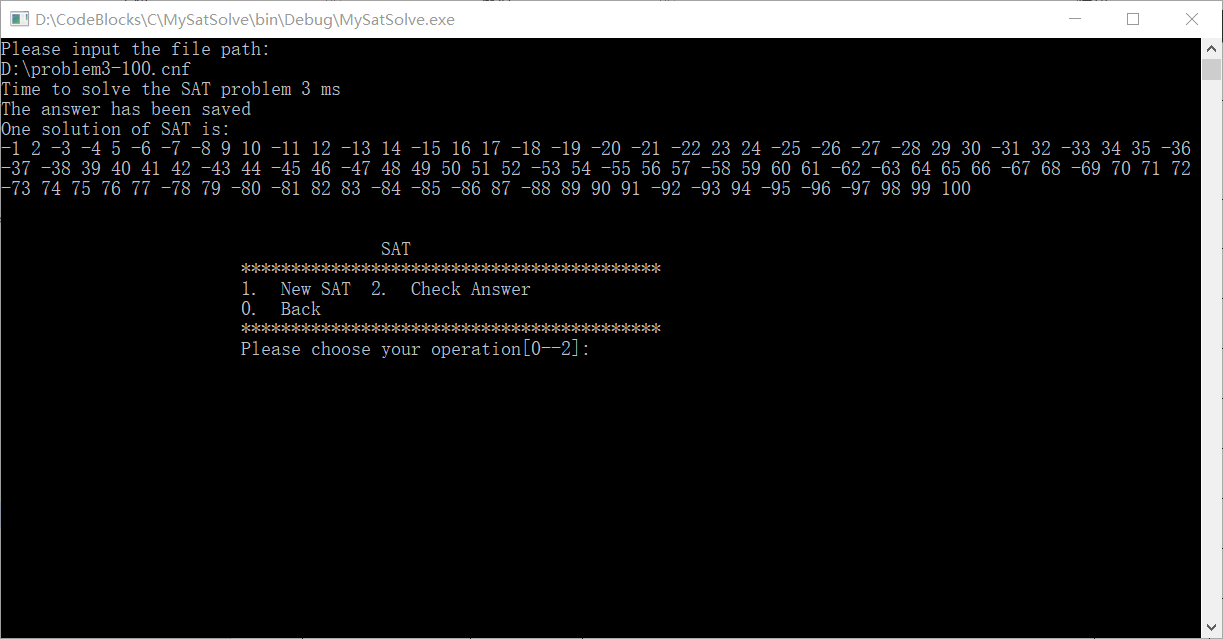


图4-12 problem6-50.cnf算例优化后

图4-13 problem3-100.cnf算例优化后

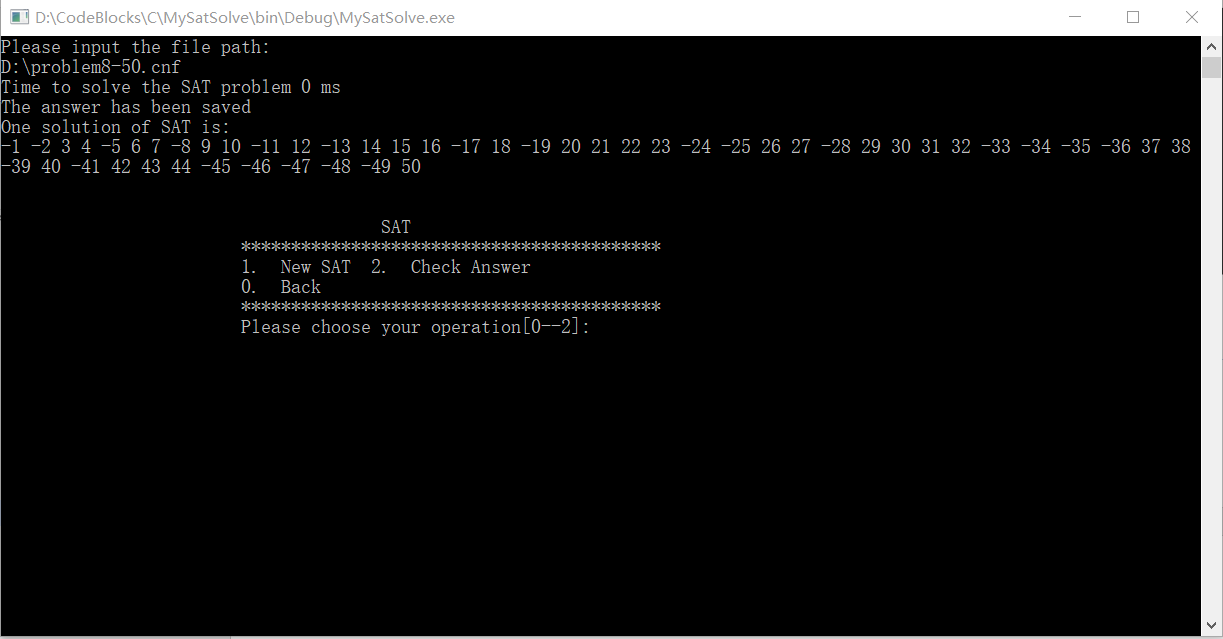


图4-14 problem8-50.cnf算例优化后

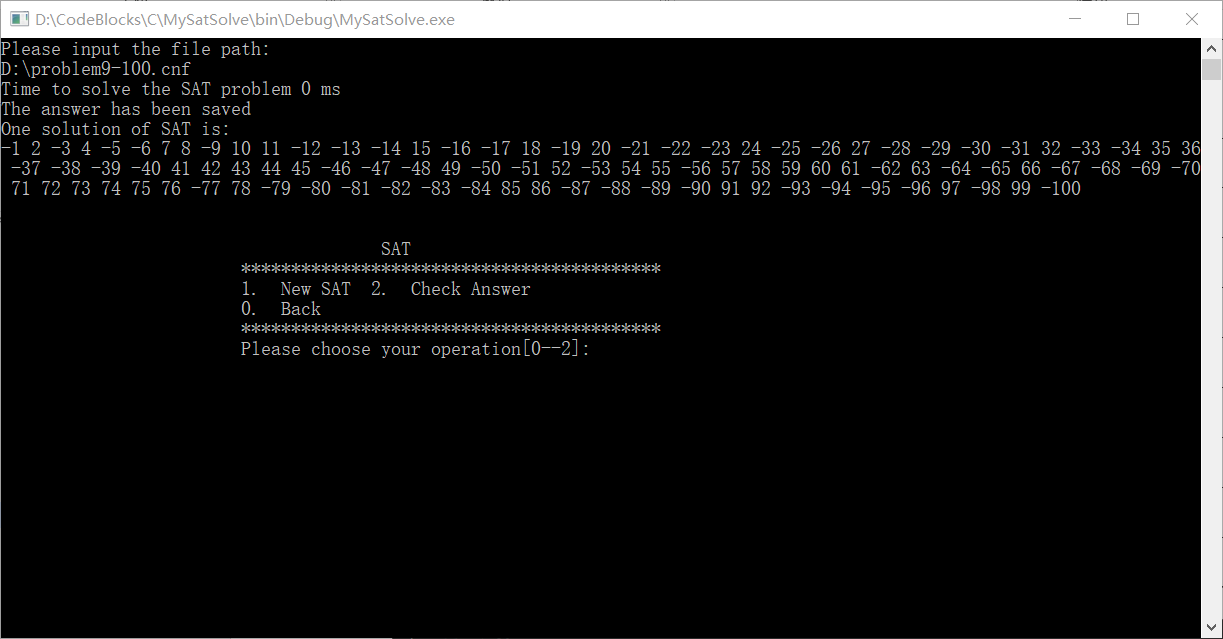


图4-15 problem9-100.cnf算例优化后

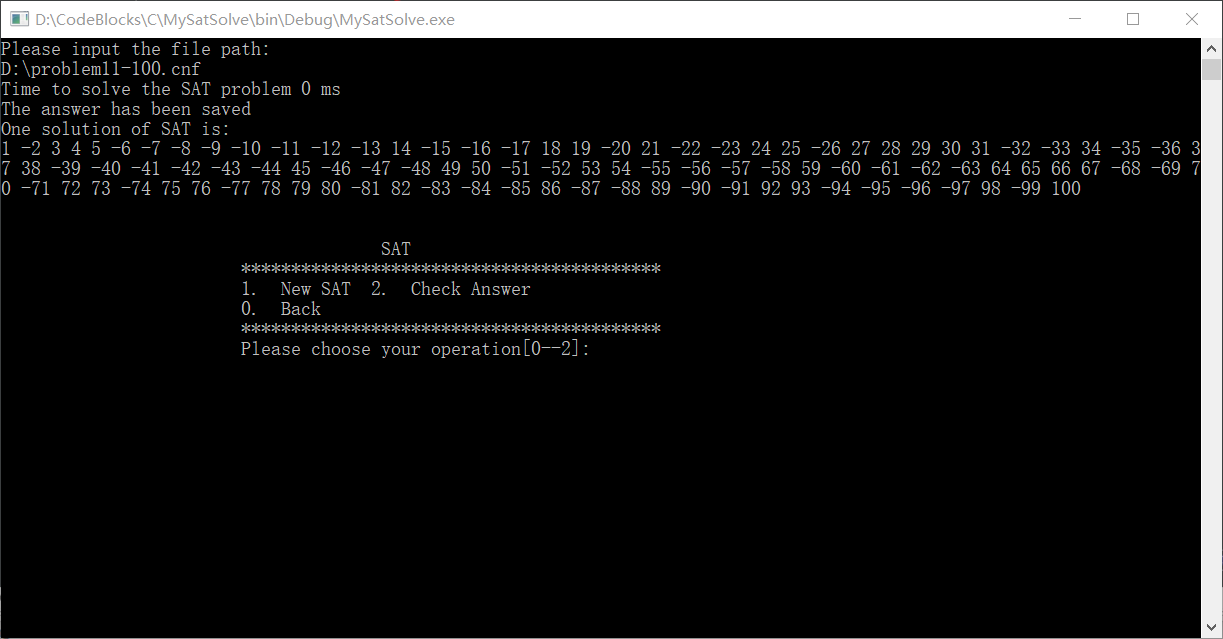


图4-16 problem11-100.cnf算例优化后

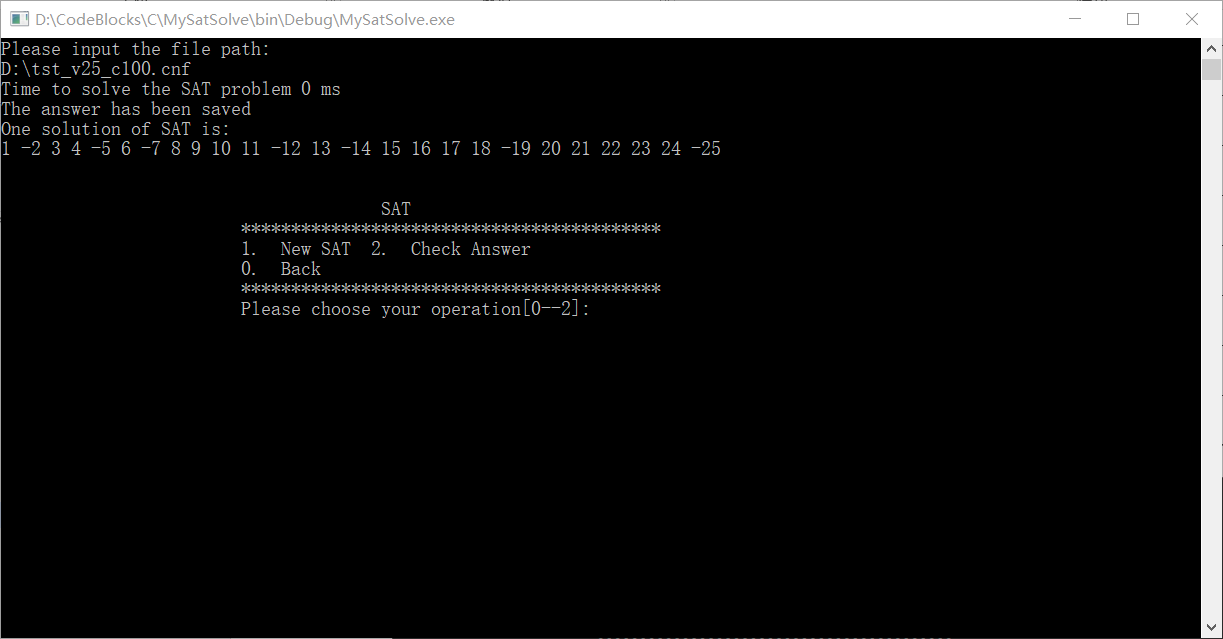


图4-17 tst\_v25\_c100.cnf算例优化后

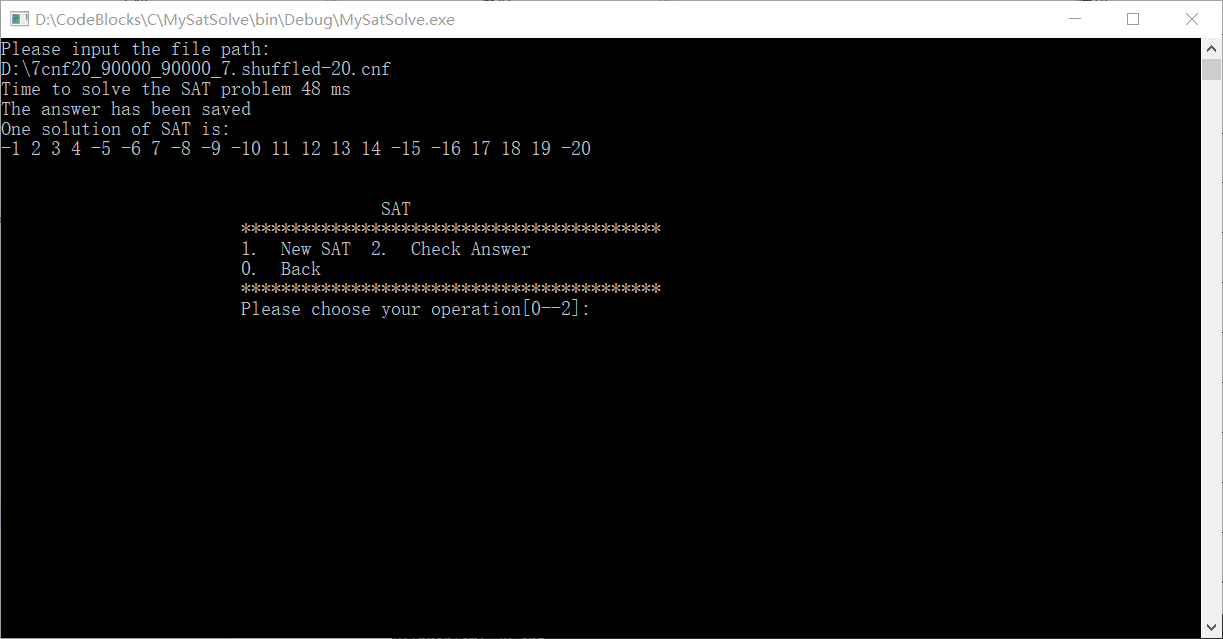


图4-18 7cnf20\_90000\_90000\_7.shuffled-20.cnf算例优化后

CNF算例 表4-3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 变元数 | 子句数 | 算例属性 | 算例名称 | 优化前（ms） | 优化后（ms） | 优化率 |
| 200 | 320 | M | problem5-200.cnf |  | 10 |  |
| 200 | 1200 | M | problem12-200.cnf | 1738 | 20 | 99% |
| 301 | 2780 | M | sud00001.cnf | 24 | 0 | 100% |
| 303 | 2851 | M | sud00009.cnf | 3 | 26 | -766% |
| 232 | 1901 | M | sud00012.cnf | 4 | 1 | 75% |
| 308 | 2911 | M | sud00021.cnf | 5 | 0 | 100% |
| 301 | 2810 | M | sud00079.cnf | 5 | 3 | 40% |
| 224 | 1762 | M | sud00082.cnf | 1 | 7 | -600% |
| 297 | 2721 | M | sud00861.cnf | 1 | 1 | 0 |
| 200 | 210 | M | tst\_v200\_c210.cnf | 0 | 1 |  |
| 231 | 1166 | M | bart17.shuffled-231.cnf | 52 | 57 | -10% |

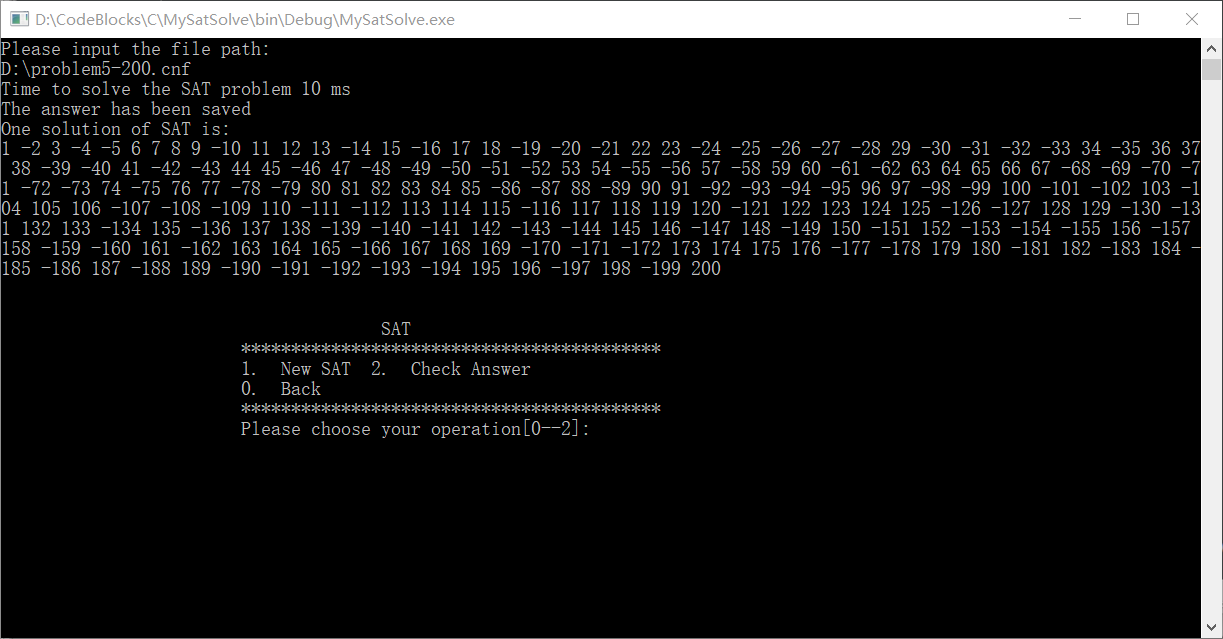


图4-19 problem5-200.cnf算例优化后

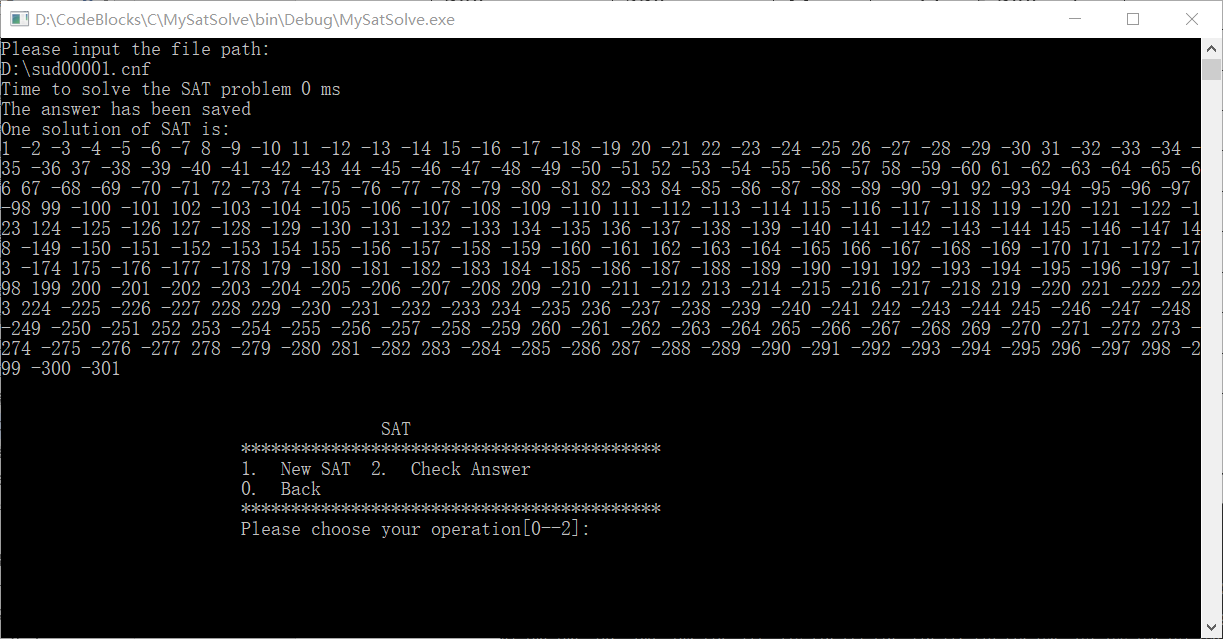


图4-20 problem12-200.cnf算例优化后

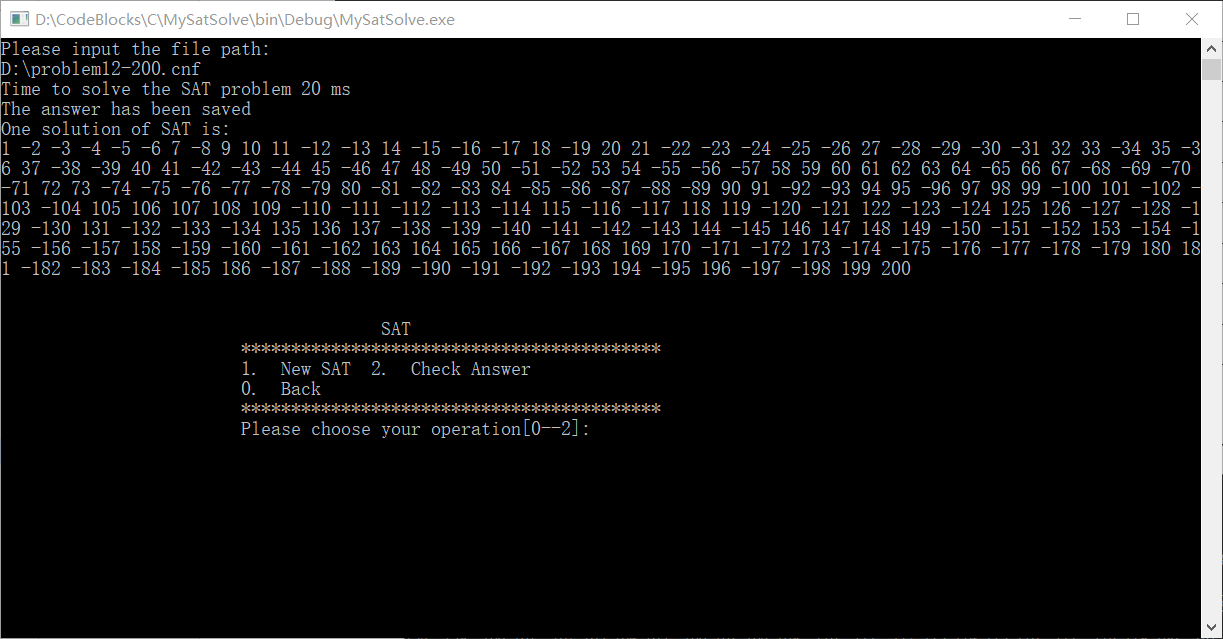
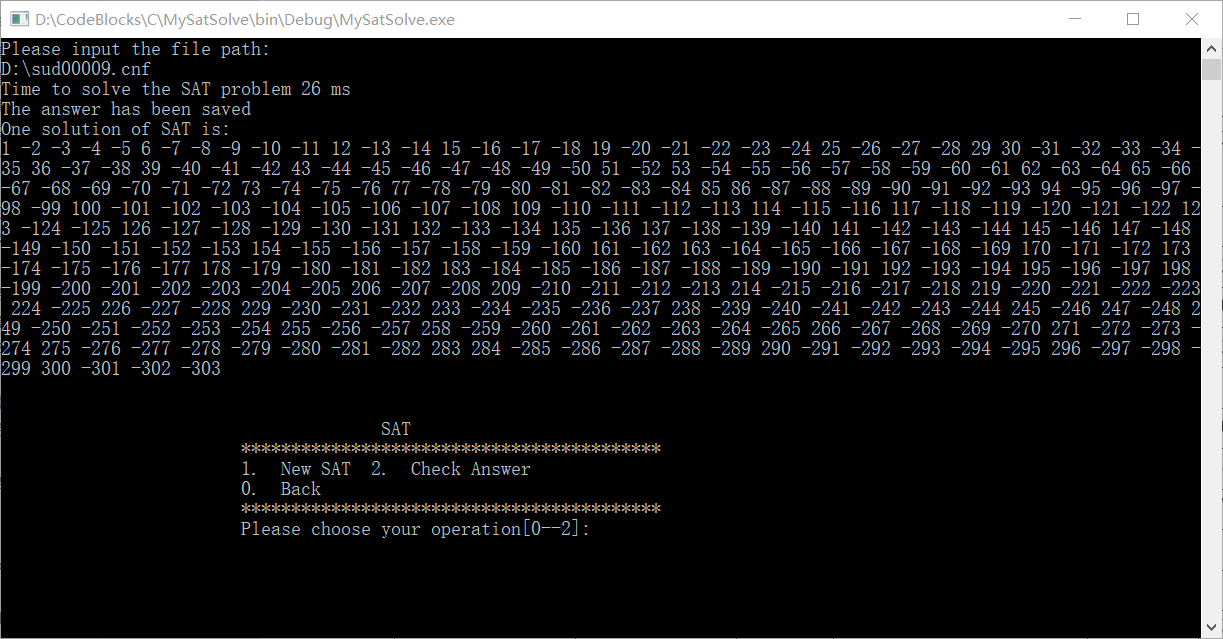
图4-21 sud00001.cnf算例优化后  


图4-22 sud00009.cnf算例优化后

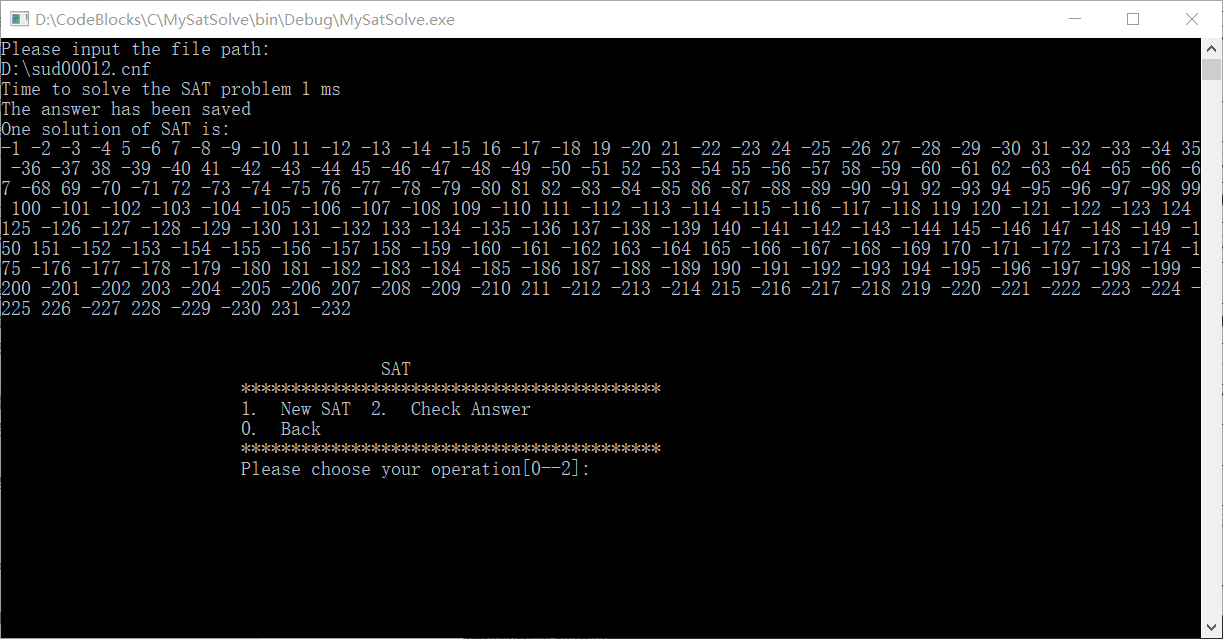


图4-23 sud00012.cnf算例优化后

## 4.2 数独游戏测试

****

图4-24 数独生成截图

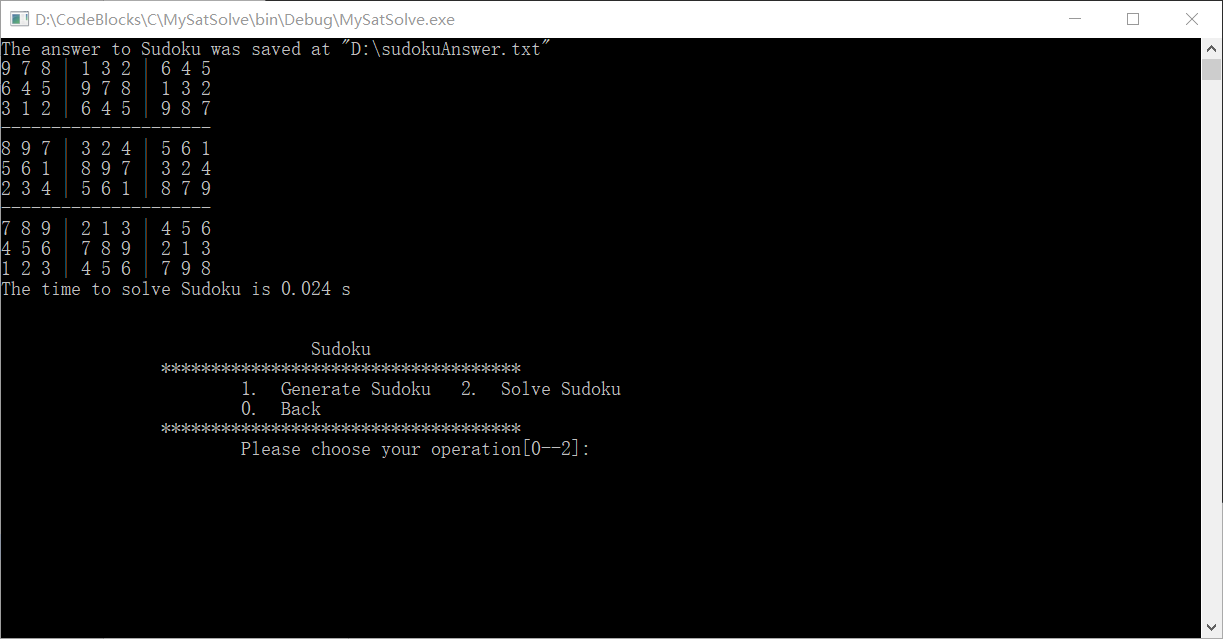


图4-25 数独求解截图

## 4.3 分析

（1）对于SAT问题：

通过上面的表格可以大致看出：未优化之前的算法在基准算例中的性能测试环节更占优势，随着变元数目与子句数目的增多优化过后的算法更占优势。通过计算同一变量文字的不同逻辑值的动态最大组合总和来选定分支变量的启发式算法，在标准测试数据上可以得到很好的结果。当子句的数目不是很多时，优化之后的算法优势不明显，因为优化需要遍历子句中的文字，得到出现次数最多的文字，在这种情况下，子句数目在一个合适的范围时不至于优化等于没优化，或者优化之后更慢。

（2）对于数独游戏：

对于数独游戏的测试，不难发现存在一个重要的问题，那就是生成初始棋盘的时间太长，但是求解的速度很快，我想这与求解SAT问题的策略有关。经过多次的测试，统计得出生成数独的平均耗时在40秒左右，求解均为几十毫秒。

（3）不同的算法适应不同的算例。优化之前的算法更适合小型算例，字句与变元数目均不多的情况。优化之后的算法更适合文字出现次数比较悬殊且变元数目和子句数目为M的情况。

# 5. 感悟与总结

通过课程设计教学与实践环节，我进一步正确理解与应用了专业知识，增强和提高了分析问题与解决问题的综合能力，加深了对于求解实际问题的基本科研步骤的体会与理解，增强和提升了信息搜索和分析技能，培养了技术总结的基本技能，锻炼了课程设计报告的撰写的能力。

课程设计这门课程只进行了八次，共两个星期，但我觉得这门课的重量不止这些学时，在我看来虽然学时少，但是收获大。这次课设与以往的实验不同，以往都是一个文件就可以搞定，但是这次试验更偏向于实际，注重工程中的模块化编程，这一点是我的收获之一。

怎么说呢，我感觉这次的课程设计课程的难度超过以往，毕竟往年只是C语言的课设或者数据结构课设，但是今年是两者的结合，难度上提升很大。很多人都是从寒假看到题目就开始构思撰写的，我当时在寒假看到题目的时候是真的很懵，看不懂要干嘛，毕竟也没有模板什么的，所以只有问同学问一些代码基础比较好的同学，问一问他们有什么思路。在学习的过程中，有一点是必须要具有的，那就是勤于请教，不懂就要问，我在这些天里问了很多人关于课设中遇到的难题，他们也都乐于回答，这使我很感激。

若问收获，我想我最大的收获就是意识到自己代码能力薄弱，需要锻炼吧。孔子说学而不思则罔，思而不学则殆。过去的学习中，我总是拘泥于书本，并没有过多重视实践，这次课设使我变得清醒，毕竟以后的工作也要和代码打交道，代码能力不能是短板。这次课设也使我意识到，学习的过程中需要同伴，互相监督学习，互相鼓励支持，一个人是很难克制自己的，希望以后寻找一些学习上的同伴，共同进步。

最后，希望自己谨记经验教训，坚持下来。

# 6. 致谢

在我完成课程设计的期间里，我遇到了很多困难，在我陷入困境无法解决的时候，有很多人给予了我无私的帮助，比如班长，比如郑翰浓，比如张智璐等等。更重要的是，老师和助教给予了我很多建议，指正了我一些错误，我想如果没有他们的帮助，我不太可能在这两个星期内完成课程设计这个间艰巨的任务。在此，我向在课程设计期间帮助过我的老师、同学表示最真挚的感谢，谢谢你们对我的帮助！再次感谢！

# 参考文献

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

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

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

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

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

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

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

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

[9] Sudoku Puzzles Generating: from Easy to Evil.

http://zhangroup.aporc.org/images/files/Paper\_3485.pdf

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

[11] 赵伟楠. 对可满足性（SAT）问题求全解的算法研究及实现. 硕士学位论文. 北京交通大学

# 附录A main.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include "SAT.h"

#include "Sudoku.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*文件名称:main.c

\*文件描述:主界面的显示、函数调用

\*日 期:2019-2-21

\*版 本：V1.0

\*@\_@|||||||||||

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main() {

int op = 1;

while (op) {

printf("\n\n");

printf("\t\t\t Main Menu \n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. Sudoku 2. SAT\n");

printf("\t\t\t0. Exit\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\tPlease choose your operation[0--2]:");

scanf("%d", &op);

system("cls");

switch (op) {

case 1:

Sudoku();

break;

case 2:

Sat();

break;

case 0:

exit(0);

default:

printf("\t\t\tPlease choose your operation again[0--2]:");

scanf("%d", &op);

}

}

return 0;

}

# 附录B SAT.h

#ifndef SAT\_H\_INCLUDED

#define SAT\_H\_INCLUDED

#define MaxNumVar 4000

#define PreassignVar 18

//预先分配的变元数目

#define CONFLICT 0

#define SATISFIABLE 1

#define UNSATISFIABLE 0

#define OTHERS 2

#define SINGLE -1

//UNKNOWN表示该变元未知，NONE表示该变元不存在，FALSE表示该变元为假，TRUE表示该变元为真

#define FALSE -1

#define TRUE 1

#define UNKNOWN 0

#define NONE 2

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*文件名称:SAT.h

\*文件描述:关于求解SAT问题的函数声明、数据结构定义

\*日 期:2019-2-21

\*版 本：V1.0

\*@\_@|||||||||||

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//该结构参考双文字监视方法，可看作两张表格

typedef struct varWatch {

struct varList \*pos; //文字邻接表，正文字邻接表，负文字邻接表，

struct varList \*neg;

} Var\_watch;

typedef struct varList {

struct clause \*p; //指向一个子句

struct varList \*next; //指向下一个包含该文字的子句

} VarList;

typedef struct clause {

struct clauseLiteral \*p;//指向一个子句中的下一个文字

struct clause \*nextClause;//这个在邻接表中无意义，要从读取CNF文件时看出效果

} Clause;

typedef struct clauseLiteral {

int data;//文字的值

struct clauseLiteral \*next; //指向子句中的下一个文字

} ClauseLiteral;

typedef struct satAnswer {

int branchLevel[MaxNumVar + 1]; //赋值时的决策树高度

int value[MaxNumVar + 1]; //TRUE or FALSE or UNKNOWN or NONE

int searched[MaxNumVar + 1]; //已被搜索的情况数

int singleClause[MaxNumVar + 1]; //标记是否存在该变量的单子句

} SatAnswer;

int InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision);

int Sat();

int LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp);

int GetNum(FILE \*fp);

int DPLL(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int op, int firstBranch);

int PrintAnswer(SatAnswer \*answer, int result, char filename[100], int duration);

int PutClause(Clause \*ctemp, int var, Var\_watch var\_watch[]);

int Deduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*root);

int SingleClauseDeduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*\*vp);

int NextBranch(int branchDecision[], SatAnswer \*answer);

int Analyse\_conflict(int \*blevel, int var, SatAnswer \*answer);

int ChectAnswer(Clause \*S, SatAnswer \*answer);

int numVar;

int knownVar;

int numBranch;

int firstBranch[MaxNumVar];

#endif // SAT\_H\_INCLUDED

# 附录C satSolve.c

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "SAT.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*文件名称:satSolve.c

\*文件描述:解决SAT问题的函数实现

\*日 期:2019-2-21

\*版 本：V1.0

\*@\_@|||||||||||

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InitSat(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]) {

Clause \*cfront = \*S, \*crear; //子句的前后指针，用于清空

ClauseLiteral \*lfront, \*lrear; //文字的前后指针

//子句初始化

while (cfront) {

crear = cfront->nextClause; //这两行可以理解为一个表格从左上角出发

lfront = cfront->p;

while (lfront) {

lrear = lfront->next;

free(lfront);

lfront = lrear; //逐一清空子句中的文字，清空表格中的一行

}

free(cfront);

cfront = crear; //清除完毕之后，由于free了前指针。。。

}

\*S = NULL;

//解初始化

numVar = 0; //变元数目为0

knownVar = 0; //已知变元数目为0

numBranch = 0; //分支数目为0

\*answer = (SatAnswer \*) malloc(sizeof(SatAnswer)); //给answer结构分配空间

for (int i = 1; i <= MaxNumVar; ++i) {

//解初始化

(\*answer)->value[i] = NONE; //表示变元不存在

(\*answer)->branchLevel[i] = 0; //赋值时决策树高度为0

(\*answer)->searched[i] = 0; //已被搜索的情况数是0

(\*answer)->singleClause[i] = 0; //该变量的单子句数目为0

//文字邻接表初始化

var\_watch[i].pos = (VarList \*) malloc(sizeof(VarList)); //为正文字邻接表指针分配空间

var\_watch[i].pos->next = NULL; //正文字邻接表指针指向空

var\_watch[i].neg = (VarList \*) malloc(sizeof(VarList)); //为负文字邻接表指针分配空间

var\_watch[i].neg->next = NULL; //负文字邻接表指针指向空

}

//分支决策计数器初始化

for (int j = 1; j <= 2 \* MaxNumVar; ++j)

branchDecision[j] = 0; //分支决策计数器初始化为0

}

int Sat() {

Clause \*S = NULL; //子句指针

SatAnswer \*answer; //SAT解的指针

Var\_watch var\_watch[MaxNumVar + 1]; //最大变元数目加一个空间

FILE \*fp; //文件指针

char filename[100]; //文件名

int branchDecision[2 \* MaxNumVar + 1]; //决策树最大分支 应该是变元数目的二倍加一（一个变元有两种可能）

int op = 1,result;

clock\_t start, finish; //调用的time.h头文件

int duration; //用于表示耗时

while (op) {

printf("\n\n");

printf("\t\t\t\t SAT\n");

printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. New SAT 2. Check Answer\n");

printf("\t\t\t0. Back\n");

printf("\t\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\tPlease choose your operation[0--2]:");

scanf("%d", &op);

system("cls");

switch (op) {

case 1://case1意味着要读取cnf文件，首先要初始化，因此有下一行

InitSat(&S, &answer, var\_watch, branchDecision);

printf("Please input the file path:\n");

scanf("%s", filename);

fp = fopen(filename, "r");

if (fp == NULL) {

printf("fail to open the file!\n ");

break;

} else LoadCnf(&S, answer, var\_watch, fp); //调用LoadCnf 函数，读取文件

start = clock(); //调用time.h，记录开始时间

result = DPLL(answer, var\_watch, branchDecision, 1, 1);

finish = clock(); //调用time.h，记录结束时间

duration = (finish - start);//得到解决问题的耗时

if (result == SATISFIABLE)

{

printf("Time to solve the SAT problem %d ms\n", duration);

PrintAnswer(answer, 1, filename, duration);

}

else

{

printf("No solution!\n");

PrintAnswer(answer, 0, filename, duration);

}

break;

case 2:

ChectAnswer(S, answer);

break;

case 0:

return 0;

default:

printf("\t\t\tPlease choose your operation again[0~2]:");

scanf("%d", &op);

}

}

}

int LoadCnf(Clause \*\*S, SatAnswer \*answer, Var\_watch var\_watch[], FILE \*fp) {//加载CNF文件

char c;

Clause \*ctemp, \*cp = NULL; //两个指针，加载CNF文件中的子句会用到

ClauseLiteral \*lp, \*ltemp; //两个指针，加载CNF文件中的文字会用到

int var; //变量的值

int numClauseVar; //每个子句的变元数

fscanf(fp, "%c", &c);

while (c == 'c') { //注释段

while (c != '\n' && c != '\r') //

fscanf(fp, "%c", &c); //

fscanf(fp, "%c", &c);

if (c == '\n') //如果读取到换行符，意味着要再重复一次

fscanf(fp, "%c", &c);

}

fscanf(fp, " cnf "); //开始读到文件的信息段

numVar = GetNum(fp); //读取变量数，并赋值给numVar

GetNum(fp); //读取子句数

var = GetNum(fp); //再次调用，读取下一行第一个变量值，把值赋给var

while (1) {

numClauseVar = 0;

ctemp = (Clause \*) malloc(sizeof(Clause)); //为ctemp指针分配空间

lp = ctemp->p; //lp为文字指针，，可以想象一下表格长什么样

while (var) {

++numClauseVar; //计数器功能，统计每个子句的变元数目

if (answer->value[abs(var)] == NONE)

answer->value[abs(var)] = UNKNOWN;

ltemp = (ClauseLiteral \*) malloc(sizeof(ClauseLiteral));

ltemp->data = var; //文字的值域改成var，赋值

ltemp->next = NULL; //

if (numClauseVar == 1) { //储存子句中首个变量

ctemp->p = lp = ltemp;

} else { //储存子句中非首个变量

lp->next = ltemp;

lp = lp->next;

}

if (var > 0)

++firstBranch[var]; //初始分支决策计数增加

else

++firstBranch[numVar - var];

PutClause(ctemp, var, var\_watch); //储存各变量的子句地址

var = GetNum(fp);

}

if (numClauseVar == 1) { //输入单子句，则该子句必须满足，无需存入

answer->value[abs(lp->data)] = lp->data / abs(lp->data);

++knownVar; //已知变元数目加1

} else if (\*S == NULL) {

\*S = cp = ctemp;

cp->nextClause = NULL;

} else { //想想表格的形式，是列的头

cp->nextClause = ctemp;

cp = cp->nextClause;

cp->nextClause = NULL;

}

var = GetNum(fp); //若到达文件尾，再执行一次z'z 读文件操作时，设置文件结束标志

if (feof(fp))

break;

}

}

int GetNum(FILE \*fp) {//这个函数用来读取cnf文件时得到变量数、子句数、后面的内容

char c;

int sign = 1, num = 0; //num 用来得到文字的值,sign用来标记文字是正或者负

fscanf(fp, "%c", &c);

if (c == '-') {

sign = -1; //sign变为-1，表示为负文字

fscanf(fp, "%c", &c);

} else if (c == '0') { //表示该条子句结束

fscanf(fp, "%c", &c);

if (c == '\r') //表示换行

fscanf(fp, "%c", &c);

return num; //

} else if (feof(fp)) //如果是结束标记

return 0;

while (c != ' ' && c != '\n' && c != '\r') {

num = num \* 10 + c - '0'; //得到文字的值

fscanf(fp, "%c", &c);

}

if (c == '\r')

fscanf(fp, "%c", &c);

return sign \* num; //用来得到文字（包括正负和值）

}

int DPLL(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int op, int firstBranch) {

int status, var, blevel = 0; //初始判定级为0

VarList \*vp; //邻接表指针，用于

while (1) {

if (numBranch++ == 0) { //第一次分支决策

if (op == 1) //1:SAT求解、2:生成数独的检验求解

var = NextBranch(branchDecision, answer);

else

var = firstBranch;

} else

var = NextBranch(branchDecision, answer); //下一分支决策

++blevel; //判定级加1

answer->value[abs(var)] = var / abs(var); //进入一分支

answer->branchLevel[abs(var)] = blevel;

++answer->searched[abs(var)]; //已被搜索情况数加1

++knownVar; //已知变元数目加1

while (1) {

if (var > 0)

vp = var\_watch[var].neg->next; //var为TRUE，则搜索var为FALSE的子句

else

vp = var\_watch[-var].pos->next; //var为FALSE，则搜索var为TRUE的子句

status = Deduce(blevel, answer, var\_watch, branchDecision, vp);//单子句传播，返回子句的状态

if (status == SATISFIABLE) //满足的情况

return SATISFIABLE;

else if (status == CONFLICT) {

var = Analyse\_conflict(&blevel, var, answer);//var > 0，矛盾，开始回溯

if (blevel == 0)

return UNSATISFIABLE;

else { //进入另一分支，不满足

answer->value[var] = -answer->value[var];//则值进行反转，正变成负，负变成正

++answer->searched[var]; //被搜索情况数目加1

if (answer->value[var] < 0)

var = -var;

}

} else if (status == OTHERS) break; //已知条件不足，进入下一层

}

}

}

int PrintAnswer(SatAnswer \*answer,int result, char filename[100], int duration) {

FILE \*fp;

fp = filename;

int p = 0;

while (filename[p] != 0) p++;

while (filename[p] != '.') p--;

p++;

filename[p] = 'r';

p++;

filename[p] = 'e';

p++;

filename[p] = 's';

p++;

filename[p] = 0;

fp = fopen(fp, "w");

if(result == 1)

fprintf(fp, "s 1\r\n");

else

fprintf(fp, "s 0\r\n");

fprintf(fp, "v ");

for(int i = 1; i < MaxNumVar; i++)

{

if(answer->value == TRUE)

fprintf(fp, "%d ", i);

else

fprintf(fp, "-%d ", i);

}

fprintf(fp, "\r\n");

fprintf(fp, "t %d\r\n", duration);

fclose(fp);

printf("The answer has been saved\n");

printf("One solution of SAT is:\n");

for (int i = 1; i <= MaxNumVar; ++i)

if (answer->value[i] == TRUE)

{

printf("%d ", i);

//fprintf(file, "%d ", i);

}

else if (answer->value[i] == FALSE)

printf("-%d ", i);

printf("\n");

//fflush(file);

}

int PutClause(Clause \*ctemp, int var, Var\_watch var\_watch[]) {

VarList \*wp;

if (var > 0) //判断var是否大于零，从而归纳到相应的表格中

wp = var\_watch[var].pos;

else

wp = var\_watch[-var].neg;

while (wp->next)

wp = wp->next; //循环，找到VarList的尾部，将var添加到尾部

wp->next = (VarList \*) malloc(sizeof(VarList));//分配空间

wp = wp->next; //

wp->p = ctemp; //将子句放在末尾

wp->next = NULL;

}

int Deduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*root) {

int top = 0, status; //status表示状态

VarList \*stack[MaxNumVar], \*vp = root; //栈，栈的最大长度为最大变元数目

stack[top++] = vp;

while (top) {

vp = stack[top - 1]; //访问栈顶元素

status = SINGLE;

while (status == SINGLE && vp) { //左子树搜索

status = SingleClauseDeduce(blevel, answer, var\_watch, branchDecision, &vp);

stack[top++] = vp; //左孩子入栈

}

--top; //空指针退栈

if (status == CONFLICT)

return CONFLICT;

if (top) { //右子树搜索

vp = stack[--top]; //根节点出栈

if (vp->next)

stack[top++] = vp->next; //右孩子入栈

}

}

if (knownVar < numVar)

return OTHERS;

else return SATISFIABLE;

}

int SingleClauseDeduce(int blevel, SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], VarList \*\*vp) {

Clause \*cp; //子句指针

ClauseLiteral \*lp; //文字指针

int unknownNum, firstUnknown, satisfiable; //

//初始化

unknownNum = 0;

firstUnknown = 0;

satisfiable = 0;

cp = (\*vp)->p; //将cp指向vp所指向的子句

lp = cp->p; //文字指针指向cp所指向的文字

if (lp == NULL)

return OTHERS;

while (lp) {

if (lp->data > 0)

++branchDecision[lp->data]; //分支决策计数增加

else

++branchDecision[numVar - lp->data];

if (answer->value[abs(lp->data)] \* lp->data > 0) { //子句中存在值为TRUE的文字，子句成立

satisfiable = 1;

break;

}

if (answer->value[abs(lp->data)] == UNKNOWN) {

++unknownNum; //计数子句中未被赋值的文字，

if (firstUnknown == 0)

firstUnknown = lp->data; //记录第一个未知的文字

}

lp = lp->next;

}

if (unknownNum == 0 && satisfiable == 0) //该子句文字均已知并且都为FALSE，为矛盾句

return CONFLICT;

else if (unknownNum == 1 && satisfiable == 0) { //该子句无值为TRUE的文字，且只有一个未知文字，为单子句

answer->singleClause[abs(firstUnknown)] = 1; //标记，单子句出现的位置

answer->value[abs(firstUnknown)] = firstUnknown / abs(firstUnknown);

answer->branchLevel[abs(firstUnknown)] = blevel;

++knownVar; //已被赋值的变元数目加1,即确定变元数目加1

if (firstUnknown > 0)

\*vp = var\_watch[firstUnknown].neg->next; //var为TRUE，则检索var为FALSE的子句

else

\*vp = var\_watch[-firstUnknown].pos->next; //var为FALSE，则检索var为TRUE的子句

return SINGLE;

} else if (knownVar < numVar) {

\*vp = NULL;

return OTHERS; //判断条件不足，返回OTHERS

} else return SATISFIABLE;

}

int NextBranch(int branchDecision[], SatAnswer \*answer) {//下一分支函数

int maxVar = numVar, maxCount = 0;

int \*branch;

++numBranch;

branch = numBranch == 1 ? firstBranch : branchDecision;

for (int i = 1; i <= 2 \* numVar; ++i) {

if (i <= numVar && answer->value[i] != UNKNOWN)

continue;

if (i > numVar && answer->value[i - numVar] != UNKNOWN)

continue;

if (maxCount <= \*(branch + i)) {

maxVar = i;

maxCount = \*(branch + i);

}

// maxVar = i;

}

return maxVar > numVar ? numVar - maxVar : maxVar;

}

int Analyse\_conflict(int \*blevel, int var, SatAnswer \*answer) {//回溯函数

int fore = abs(var);

while (\*blevel != 0) {

for (int j = 1; j <= numVar; ++j)

if (j != fore && answer->branchLevel[j] == \*blevel) { //将由var赋值产生的单子句重置

answer->value[j] = UNKNOWN;

answer->branchLevel[j] = 0;

answer->searched[j] = 0;

answer->singleClause[j] = 0;

--knownVar;

}

if (\*blevel != 1) {

if (answer->searched[fore] == 2) { //var的TRUE和FALSE分支均搜索过，进行回溯

--(\*blevel);

answer->value[fore] = UNKNOWN;

answer->branchLevel[fore] = 0;

answer->searched[fore] = 0;

--knownVar;

for (int i = 1; i <= numVar; ++i)

if (answer->branchLevel[i] == \*blevel && answer->singleClause[i] == 0) {

fore = i;

break;

}

} else break; //搜索另一分支

} else if (answer->searched[abs(fore)] == 2)//blevel1全部搜索完

--(\*blevel);

else break; //搜索blevel1的另一分支

}

return fore;

}

int ChectAnswer(Clause \*S, SatAnswer \*answer) {//解的正误检查

FILE \*fp;

Clause \*cp = S;

VarList \*error = NULL, \*ep;

ClauseLiteral \*lp;

int ok = 0, s = 1, num = 0;

fp = fopen("D:\\Satcheck.txt", "w");//在D盘建立一个txt文件

error = ep = (VarList \*) malloc(sizeof(VarList));//分配空间

error->next = NULL;

fprintf(fp, "Answer:\r\n");

for (int i = 1; i <= MaxNumVar; ++i)

if (answer->value[i] == TRUE)//标记的第i个值为正文字

fprintf(fp, "%d ", i);

else if (answer->value[i] == FALSE)//标记的第i个值为负文字

fprintf(fp, "-%d ", i);

fprintf(fp, "\r\n\r\n");

while (cp) {

ok = 0;

lp = cp->p;

fprintf(fp, "C%-4d ", ++num);

while (lp) {

fprintf(fp, "%4d ", lp->data);

if (answer->value[abs(lp->data)] \* lp->data > 0)

ok = 1;

lp = lp->next;

}

if (ok == 1)

fprintf(fp, "T\r\n");

else {

s = 0;

fprintf(fp, "F\r\n");

ep->next = (VarList \*) malloc(sizeof(VarList));

ep = ep->next;

ep->p = cp;

ep->next = NULL;

}

cp = cp->nextClause;

}

if (s == 1)

printf("Check: Satisfiable!\n");

else printf("Check: Unsatisfiable!\n");//检查的结果

printf("The detail was saved at \"D:\\Satcheck.txt\"\n");//保存在哪里了

ep = error->next;

if (ep) {

fprintf(fp, "\r\n\r\nError:\r\n");

num = 0;

}

while (ep) {

fprintf(fp, "C%-4d ", ++num);

lp = ep->p->p;

while (lp) {

fprintf(fp, "%4d ", lp->data);

lp = lp->next;

}

fprintf(fp, "F\r\n");

ep = ep->next;

}

fclose(fp);

}

# 附录D Sudoku.h

#ifndef SUDOKU\_H\_INCLUDED

#define SUDOKU\_H\_INCLUDED

#include "SAT.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*文件名称:Sudoku.h

\*文件描述:用来定义数独游戏相关的函数

\*日 期:2019-2-21

\*版 本：V1.0

\*@\_@|||||||||||

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Sudoku();

int NewSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]);

int DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int SolveSudoku(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]);

int dig\_watch(int sudokuTable[9][9]);

int dig\_test(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision);

#endif // SUDOKU\_H\_INCLUDED

# 附录E Game.c

#ifndef GAME\_C\_INCLUDED

#define GAME\_C\_INCLUDED

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "Sudoku.h"

#include "SAT.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*文件名称:Game.c

\*文件描述:数独游戏部分的代码实现

\*日 期:2019-2-21

\*版 本：V1.0

\*@\_@|||||||||||

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Sudoku() {

Clause \*S = NULL;

SatAnswer \*answer;

FILE \*fp;

char filename[100];

Var\_watch var\_watch[MaxNumVar + 1];

int branchDecision[2 \* MaxNumVar + 1];

int sudokuTable[9][9];

int op = 1;

clock\_t start, finish;

double duration;

srand((unsigned) time(NULL));

while (op) {

printf("\n\n");

printf("\t\t\t Sudoku\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. Generate Sudoku 2. Solve Sudoku\n");

printf("\t\t\t0. Back\n");

printf("\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\tPlease choose your operation[0--2]:");

scanf("%d", &op);

system("cls");

switch (op) {

case 1:

start = clock();

NewSudoku(&S, &answer, var\_watch, branchDecision, sudokuTable);

finish = clock();

duration = (double) (finish - start) / 1000.0;

printf("The time to generate Sudoku is %.3f s\n", duration);

break;

case 2:

InitSat(&S, &answer, var\_watch, branchDecision);

/\*printf("Input the file path:\n");

scanf("%s", filename);

fp = fopen(filename, "r");\*/

fp = fopen("D:\\sudoku\_rule.txt", "r");

LoadCnf(&S, answer, var\_watch, fp);

fclose(fp);

start = clock();

SolveSudoku(answer, var\_watch, branchDecision, sudokuTable);

finish = clock();

duration = (double) (finish - start) / 1000.0;

printf("The time to solve Sudoku is %.3f s\n", duration);

break;

case 0:

return 0;

default:

printf("\t\t\tPlease choose your operation again[0~2]:");

scanf("%d", &op);

}

}

}

int NewSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

clock\_t a, b;

double d;

a = clock();

if (GenerateSudoku(S, answer, var\_watch, branchDecision) == 0)

return 0;

b = clock();

d = (double) (b - a) / 1000.0;

printf("The time to generate the final Sudoku is %.3f s\n\n", d);

DigHole(\*answer, var\_watch, branchDecision, sudokuTable);

}

int GenerateSudoku(Clause \*\*S, SatAnswer \*\*answer, Var\_watch var\_watch[], int branchDecision[]) {

int x, y, z, i, j, k, l;

int dig\_order[82], index;

FILE \*fp;

fp = fopen("D:\\sudoku\_rule.txt", "w");

fprintf(fp, "p cnf 729 10287\r\n");

for (x = 0; x < 9; ++x)

for (y = 0; y < 9; ++y)

for (z = 1; z <= 8; ++z) {

for (i = z + 1; i <= 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* x + 9 \* y + i));//每个位置，数字1~9至多出现一次

}

for (x = 0; x < 9; ++x)

for (z = 1; z <= 9; ++z)

for (y = 0; y < 8; ++y) {

for (i = y + 1; i < 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* x + 9 \* i + z));//每一行，数字1~9至多出现一次

}

for (y = 0; y < 9; ++y)

for (z = 1; z <= 9; ++z)

for (x = 0; x < 8; ++x) {

for (i = x + 1; i < 9; ++i)

fprintf(fp, "%d %d 0\r\n", -(81 \* x + 9 \* y + z), -(81 \* i + 9 \* y + z));//每一列，数字1~9至多出现一次

}

for (z = 1; z <= 9; ++z)

for (i = 0; i < 3; ++i)

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

for (x = 0; x < 3; ++x)

for (y = 0; y < 3; ++y)

fprintf(fp, "%d ", 81 \* (3 \* i + x) + 9 \* (3 \* j + y) + z);//数字1~9在每个3×3数独中至少出现一次

fprintf(fp, "0\r\n");

for (x = 0; x < 3; ++x) {

for (y = 0; y < 3; ++y) {

for (k = x + 1; k < 3; ++k)

for (l = 0; l < 3; ++l)

if (l != y)

fprintf(fp, "%d %d 0\r\n", -(81 \* (3 \* i + x) + 9 \* (3 \* j + y) + z),

-(81 \* (3 \* i + k) + 9 \* (3 \* j + l) + z));//数字1~9在每个3×3数独中至多出现一次

}

}

}

fclose(fp);

do {

fp = fopen("D:\\sudoku\_rule.txt", "r");

// fp = fopen("C:\\Users\\dell\\Desktop\\base.txt", "r");

if (fp == NULL) {

printf("Opening \"D:\\sudoku\_rule.txt\" failed.\n ");

return 0;

}

InitSat(S, answer, var\_watch, branchDecision);

LoadCnf(S, \*answer, var\_watch, fp);

fclose(fp);

for (j = 1; j <= 81; ++j)

dig\_order[j] = j;

for (j = 81; j > 1; --j) { //随机生成初始化顺序

index = rand() % j + 1;

if (j != index) {

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

dig\_order[index] = dig\_order[index] ^ dig\_order[j];

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

}

}

for (k = 0; k < 11;) { //在棋盘中随机选11个格子随机填入1~9

x = (dig\_order[j] - 1) / 9;

y = (dig\_order[j] - 1) % 9;

z = rand() % 9 + 1;

for (l = 1; l <= 9; ++l)

if (l == z)

(\*answer)->value[81 \* x + 9 \* y + l] = TRUE;

else

(\*answer)->value[81 \* x + 9 \* y + l] = FALSE;

++k;

}

knownVar = k;

} while (DPLL(\*answer, var\_watch, branchDecision, 2, -(rand() % 729 + 1)) == UNSATISFIABLE);

return 1;

}

int DigHole(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

int x, y, z, i, j, k;

int dig\_order[82], index, dig = 1;

int firstBranch;

FILE \*fp;

clock\_t a, b;

a = clock();

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

answer->branchLevel[i] = 0;

answer->searched[i] = 0;

answer->singleClause[i] = 0;

}

for (x = 0; x < 9; ++x) { //得到终盘结果

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

for (z = 1; z <= 9; ++z)

if (answer->value[81 \* x + 9 \* y + z] == TRUE) {

sudokuTable[x][y] = z;

break;

}

}

}

for (j = 1; j <= 81; ++j)

dig\_order[j] = j;

for (j = 81; j > 1; --j) { //随机生成挖洞顺序

index = rand() % j + 1;

if (j != index) {

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

dig\_order[index] = dig\_order[index] ^ dig\_order[j];

dig\_order[j] = dig\_order[j] ^ dig\_order[index];

}

}

for (j = 1; j <= 81 - PreassignVar && dig <= 81;) {

b = clock();

printf("digged = %d digging = %d time = %.3f s\n", j - 1, dig, (double) (b - a) / 1000.0);

dig\_watch(sudokuTable);

x = (dig\_order[dig] - 1) / 9;

y = (dig\_order[dig++] - 1) % 9;

z = sudokuTable[x][y];

if (z <= 0) //该位置不可挖，寻找下一个位置

continue;

knownVar = 9 \* (81 - j); //已经挖掉j个洞

numBranch = 0;

for (i = 1; i <= 9; ++i) //挖去该位置

answer->value[81 \* x + 9 \* y + i] = UNKNOWN;

if (j < 4) { //挖去个数小于4，解必定唯一

++j;

sudokuTable[x][y] = 0;

for (k = 1; k <= 9; ++k) //挖去该位置

answer->value[81 \* x + 9 \* y + k] = UNKNOWN;

continue;

}

for (i = 1; i <= 9; ++i) { //检测挖去该位置解是否唯一

if (i == z)

continue;

firstBranch = 81 \* x + 9 \* y + i;

answer->searched[81 \* x + 9 \* y + i] = 1; //锁定i的另一分支

if (DPLL(answer, var\_watch, branchDecision, 2, firstBranch) == SATISFIABLE) //挖去该位置有其他解

break;

knownVar = 9 \* (81 - j); //已经挖掉j个洞

numBranch = 0;

for (k = 1; k <= numVar; ++k) { //重置终盘

if (!answer->branchLevel[k]) //决策级为0，为初始化条件，不重置

continue;

answer->value[k] = UNKNOWN;

answer->branchLevel[k] = 0;

answer->searched[k] = 0;

answer->singleClause[k] = 0;

}

}

if (i == 10) { //挖去该位置解仍唯一

++j;

sudokuTable[x][y] = 0;

} else { //挖去该位置解不唯一

if (dig > 81)

break;

sudokuTable[x][y] = -sudokuTable[x][y]; //该位置不可挖去

for (k = 1; k <= numVar; ++k) { //重置终盘

if (!answer->branchLevel[k]) //决策级为0，为初始化条件，不重置

continue;

answer->value[k] = UNKNOWN;

answer->branchLevel[k] = 0;

answer->searched[k] = 0;

answer->singleClause[k] = 0;

}

for (k = 1; k <= 9; ++k) //填入原来的数

if (k == z)

answer->value[81 \* x + 9 \* y + k] = TRUE;

else

answer->value[81 \* x + 9 \* y + k] = FALSE;

}

}

fp = fopen("D:\\sudokuTable.txt", "a+");

printf("The new Sudoku was generated at \"D:\\sudokuTable.txt\"\n");

printf("%d known positions:\n", 81 - j + 1);

for (x = 0; x < 9; ++x) { //得到生成数独

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

sudokuTable[x][y] = abs(sudokuTable[x][y]);

if (sudokuTable[x][y] != 0) {

for (i = 1; i <= 9; ++i) {

if (i != sudokuTable[x][y])

fprintf(fp, "%d 0\r\n", -(81 \* x + 9 \* y + i));

else

fprintf(fp, "%d 0\r\n", 81 \* x + 9 \* y + i);

}

}

if (y != 0 && y % 3 == 0)

printf("| ");

printf("%d ", sudokuTable[x][y]);

}

printf("\n");

if (x != 8 && x % 3 == 2)

printf("---------------------\n");

}

fclose(fp);

}

int SolveSudoku(SatAnswer \*answer, Var\_watch var\_watch[], int branchDecision[], int sudokuTable[9][9]) {

int x, y, z;

FILE \*fp;

DPLL(answer, var\_watch, branchDecision, 1, 1);

for (x = 0; x < 9; ++x) { //得到终盘结果

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

for (z = 1; z <= 9; ++z)

if (answer->value[81 \* x + 9 \* y + z] == TRUE) {

sudokuTable[x][y] = z;

break;

}

}

}

fp = fopen("D:\\sudokuAnswer.txt", "w");

printf("The answer to Sudoku was saved at \"D:\\sudokuAnswer.txt\"\n");

for (x = 0; x < 9; ++x) { //保存并打印数独答案

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

sudokuTable[x][y] = abs(sudokuTable[x][y]);

if (sudokuTable[x][y] != 0) {

for (int i = 1; i <= 9; ++i) {

if (i != sudokuTable[x][y])

fprintf(fp, "%d 0\r\n", -(81 \* x + 9 \* y + i));

else

fprintf(fp, "%d 0\r\n", 81 \* x + 9 \* y + i);

}

}

if (y != 0 && y % 3 == 0)

printf("| ");

printf("%d ", sudokuTable[x][y]);

}

printf("\n");

if (x != 8 && x % 3 == 2)

printf("---------------------\n");

}

fclose(fp);

}

int dig\_watch(int sudokuTable[9][9]) {

int x, y;

for (x = 0; x < 9; ++x) { //保存并打印数独答案

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

printf("%2d ", sudokuTable[x][y]);

}

printf("\n");

}

printf("\n\n");

}

int dig\_test(Clause \*\*S, SatAnswer \*\*answer, Var\_watch \*var\_watch, int \*branchDecision) {

FILE \*fp;

Clause \*S1 = NULL;

Var\_watch var\_watch1[MaxNumVar + 1];

int branchDecision1[2 \* MaxNumVar + 1];

int sudokuTable[9][9];

InitSat(S, answer, var\_watch, branchDecision);

fp = fopen("C:\\Users\\dell\\Desktop\\sudoku\_rule.txt", "r"); //初始棋盘

LoadCnf(S, \*answer, var\_watch, fp);

fclose(fp);

InitSat(&S1, answer, var\_watch1, branchDecision1);

fp = fopen("D:\\sudoku\_answer.txt", "r"); //测试终盘

LoadCnf(&S1, \*answer, var\_watch1, fp);

fclose(fp);

DigHole(\*answer, var\_watch, branchDecision, sudokuTable);

}

#endif // GAME\_C\_INCLUDED

**目□□录**（黑体小2号加粗居中）

任务书 I

1□□□引言 1

1.1□□×××××× 1

1.2□□×××××× 3

1.2.1□□×××××× 7

1.3□□×××××× 10

**……**

**……**

**……**

3□□□×××××× 20

3.1□□×××××× 20

3.2□□×××××× 23

3.2.1□□×××××× 25

3.3□□×××××× 30

**……**

**……**

4□□□ 40

参考文献 44

附录□×××××× 45

（章为宋体小4号加粗，其余宋体小4号，字母、阿拉伯数字为Time New Roman小4号）