

**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： 计算机1806**

**学 号： U201814655**

**姓 名： 杨雨鑫**

**指导教师： 纪俊文**

**报告日期： 2020.5.10**

**计算机科学与技术学院**

# 任务书

**设计内容**

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

**设计要求**

要求具有如下功能：

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

**目录**

[任务书 2](#_Toc36476695)

[1引言 5](#_Toc36476696)

[1.1课题背景与意义 5](#_Toc36476697)

[1.2国内外研究现状 5](#_Toc36476698)

[1.3课程设计的主要研究工作 5](#_Toc36476699)

[2系统需求分析与总体设计 7](#_Toc36476700)

[2.1系统需求分析 7](#_Toc36476701)

[2.2系统总体设计 7](#_Toc36476702)

[3系统详细设计 9](#_Toc36476703)

[3.1有关数据结构的定义 9](#_Toc36476704)

[3.2主要算法设计 11](#_Toc36476705)

[4系统实现与测试 13](#_Toc36476706)

[4.1系统实现 13](#_Toc36476707)

[4.2系统测试 17](#_Toc36476708)

[5. 总结与展望 27](#_Toc36476709)

[5.1全文总结 27](#_Toc36476710)

[5.2工作展望 27](#_Toc36476711)

[6体会 29](#_Toc36476712)

[参考文献 30](#_Toc36476713)

[附录A 31](#_Toc36476714)

# 1引言

## 1.1课题背景与意义

### 1.1.1 课题背景

近十多年来随着许多有效的启发式算法的提出以及一系列数据结构和实现方法上的创新使得可满足性问题(SAT)的求解水平取得了突飞猛进的提高。尽管目前可满足性问题还属于NP问题，目前还没有任何一种算法能实现在最差情况下的多项式时间复杂度，但当前的SAT问题求解器以能够轻松解决含有几万个甚至几十万个变量的可满足性问题。由于可满足性问题求解水平的巨大进步，SAT的应用范围已越来越大，从形式验证到人工智能等许多领域均以SAT求解器作为其核心计算引擎。可满足性问题的研究已经从一个计算机理论学界的纯学术问题逐渐发展成为当前许多计算机领域中具有很大应用价值的计算问题。

近十多年来，可满足性问题研究逐渐升温，已成为了国际国内的研究热点，

取得了一批相当重要的理论和实践成果，应该说当前的SAT问题研究比十多年前已取得了很大的突破，并直接或间接地推动了其他相关领域(比如形式验证，人工智能等领域)的发展。

### 1.1.2 课题意义

可满足性(SAT)问题属于命题逻辑的范畴，是约束满足问题的一个分支。在集成电路形式验证中，SAT问题求解器已被广泛应用于定界模型检验，无界模型检验和等价性检验。ATPG检验，微处理器验证，设计错误诊断。除此之外，SAT问题还应用于其它领域。例如：FPGA的布局，逻辑优化等。

可以看到，可满足性(sat)问题研究是许多重大问题的基础，有着非常广泛的应用价值。

## 1.2 国内外研究现状

当前的SAT问题的算法主要包含两类：局部搜索算法和回溯搜索算法。近十多年来，国际上已提出了各种不同的局部搜索算法和回溯搜索算法，使得SAl懈决器解决不同领域中的SAT问题的能力不断增强，能解决的问题的规模不断增大。其中局部搜索算法显示出对于随机的SAT问题特别有用，而回溯搜索算法则被用来解决大规模实际应用领域中的SAT问题。事实上，国际上已提出了一大批采用回溯搜索算法的高效的SAT问题解决器，其中绝大多数提出来的回溯搜索算法是对原始的DPLL回溯搜索算法【3，4】的改进算法。这些改进措施包括：新的变量决策策略，新的搜索空间剪除技术，新的推理和回溯技术以及新的更快的算法实现方案和数据结构等。当前水平的SAT问题求解器已能够轻松解决以前传统SAT问题解决器完全无法解决的可满足性问题。

## 1.3 课程设计的主要研究工作

本次课设以实现DPLL算法为基础的SAT求解器为核心，通过多个算例来检验求解器的性能。并在其基础上应用于数独游戏的求解，实现对一个数独游戏的自动求解。实现用SAT求解器得出正确的解的同时，尽量去完善算法，以达到提高求解器效率的目的。

# 2系统需求分析与总体设计

## 2.1系统需求分析

需要实现一个能够成功运行的SAT求解器，并使其尽量高效，并且有效地解决各种大小的算例，并且能实现对cnf文件的各种修改。打印cnf文件，修改其中内容（比如增加或删除子句或文字）并将得到的结果输出，并且验证其正确性。

当成功设计了一个SAT求解器后，将其用于二进制数独的求解。要求首先能自动随机生成数独盘，或者从外界读取数独盘，然后将数独转化为SAT问题的求解，并利用SAT求解器得到结果并输出，最后打印出数独的答案，用以验证。

## 2.2系统总体设计

SAT求解器的功能主要有5个模块：

1. 对CNF文件的读取，并打印该文件，检查数据是否放入了结构中。
2. 对子句文字的删除，添加的独立操作。
3. 运行SAT求解器进行求解。
4. 将求解结果文件保存。
5. 对结果文件进行验证结果是否正确。

数独生成器的功能主要有3个模块：

1. 随机生成数独终盘。
2. 求解数独，导入生成的数独cnf文件，并用SAT求解器求解
3. 打印最终得到结果的终盘。

系统总体模块结构图如图2-1所示：C:\Users\WidenSun\AppData\Local\Temp\ksohtml14936\wps1.png

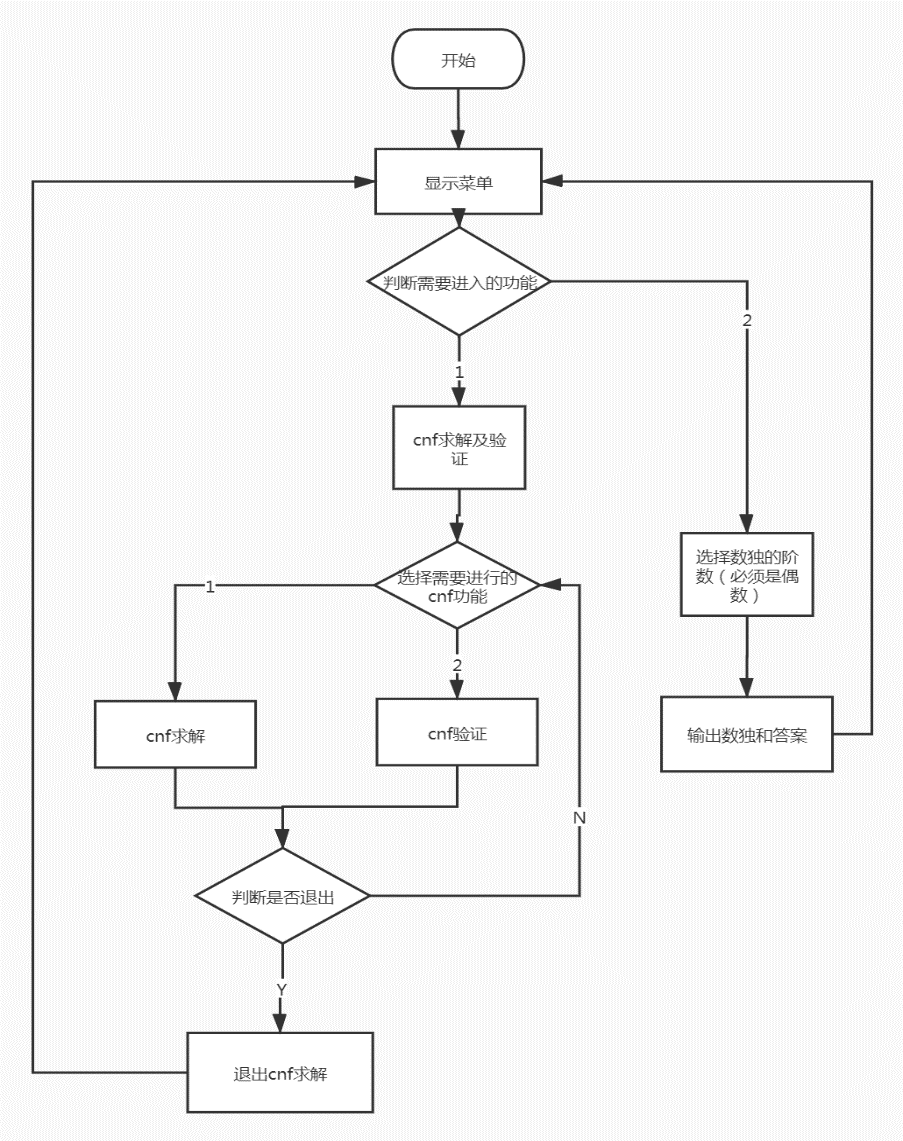


图2.1 系统总体框架图

这里我们准备设计两个菜单功能，一个是主菜单用来让用户选择是进入cnf解答或者是数独子界面。在cnf子菜单中可以选择解答cnf功能，会在屏幕上输出cnf答案和求解时间，或者是选择验证生成的res文件是否满足cnf字句文件。

在主菜单中也可以选择进入数独功能，输入数独的阶数，可以生成cnf字句并且进行求解，最后输出数独的答案和需要完成的数独。

# 3系统详细设计

## 3.1有关数据结构的定义

1. 系统中要处理的数据包括：文字结构、子句结构等。具体如表3.1所示。

表3.1 系统中的数据结构具体说明

|  |  |  |  |
| --- | --- | --- | --- |
| 数据结构定义 | 数据结构说明 | 数据元素内容 | 数据元素内容说明 |
| typedef struct varWatch  {  struct varList \*positive;  struct varList \*negative;  } Var\_watch,varWatch; | 变元观察链表 | Positive指针 | 指向对应变元的正文字在二叉链表中的地址存储链表 |
| Negative指针 | 指向对应变元负文字在二叉链表中的地址存储链表 |
| typedef struct varList  {  struct clause \*p;  struct varList \*next;  int satisfied;  } Var\_List,VarList; | 变元正负文字地址存储链表 | 指针p | 指向正文字或是负文字的所在字句的地址 |
| 指针next | 指向下一个存储该类文字地址的存储结构 |
| 变量satisfied | 记录是否因为该变元赋值而导致字句被标记为满足 |
| typedef struct clause  {  struct clauseLiteral \*p;  struct clause \*nextclause;  int word\_num;  int flag;  } clause; | 保存字句的二叉链表中的子句结构 | 指针p | 指向该字句中的第一个文字结构 |
| 指针nextclause | 指向下一个字句 |
| 变量word\_num | 用来存储该字句的文字数 |
| 变量flag | 标记该字句是否已经满足 |
| typedef struct clauseLiteral{  int data;  struct clauseLiteral \*next;  } clauseLiteral; | 存储子句中的文字 | 变量data | 用来存储文字 |
| 指针next | 用来指向该字句的下一个文字结构 |

1. 子句和变元采用二叉链表的结构进行存储。
2. 这里我设计一个监视链表将同一变元所在的子句的地址存储起来，方便在给某一个变元赋值以后在原来的二叉链表中快速删除该变元的文字。具体如图3.1所示。



图3.1 二叉链表存储结构

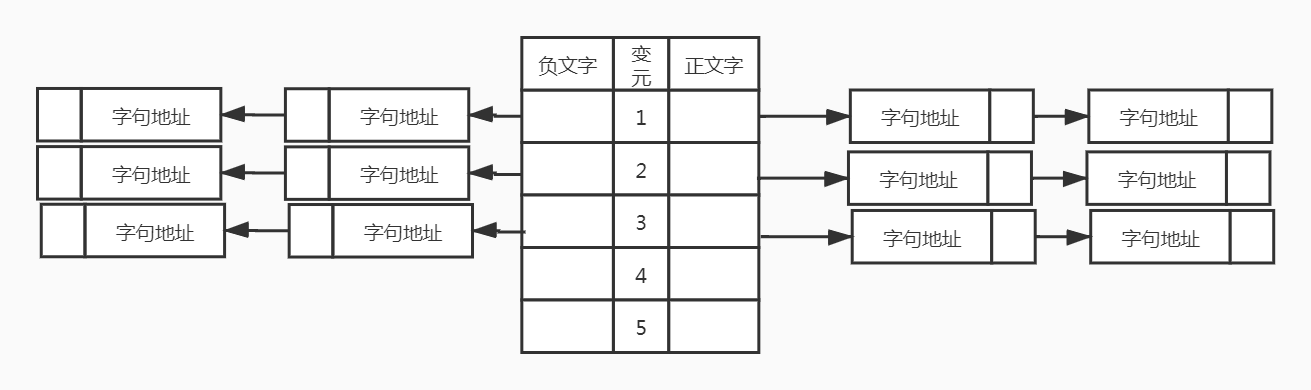


图3.2 监视链表存储结构

图3.1是我的二叉链表结构，图3.2是我的文字监视链表结构。

## 3.2主要算法设计

这部分主要描述系统中的每个模块实现的具体操作流程。

1.CNF模块：

(1)在对cnf文件进行解析时，如果读取到字符‘c’，则自动转入下一行搜索。

直到读取到首字符为‘p’的地方，然后读取后续的三个字符，进行判断，若为‘cnf’，则将接下来的第一个读取到数字记录为变元数num\_var，第二个读取的数字记录为子句数clause\_num，在这个过程中我编写了一个getnum函数专门用来把字符数字转换成十进制数字并返回。转入下一行，开始读取子句，初始化的字句flag标志都是0代表该字句未满足，之后逐个读取文字，计数器最后赋值给字句的word\_num变量代表该字句所包含的文字数。

在读取文字的同时，我们还需要构建监视链表的信息，把该文字所在字句的地址存储在相应的监视链表的位置中，方便访问文字。

(2)DPLL求解过程

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)改进后的单子句规则

通过文字监视链表，我们可以在执行单子句规则时快速的将含有该文字子句赋值为真，同时也可以把赋值相反的文字的子句中的文字删除。

(4)变元选取策略

我开始直接简单的使用了选取第一个未满足字句的第一个文字作为下一个赋值的变元，但是遇到了性能测试的ais10表现的并不出色，这引发了我对变元选取策略的优化策略的思考。

(5)DPLL回溯策略

由于回溯过程中需要擦除之前的操作，我遇到了困难，于是决定开了一个堆栈用来表示搜索路径，方便我对之前的操作进行擦除。

2.数独模块

(1)数独原始棋盘的生成：我是通过对三个规则的解读，设计出了三个函数，当确定了数独的阶数时，三个函数就会自动生成一个cnf文件，我们再利用rand函数随机选取一个变元对其赋值，再将其放入该cnf文件中，如在cnf文件中加入一行2 0。将最终的cnf文件代入我们之前写的cnf求解模块，得到的解经过变换后即为我们的原始棋盘。

(2)挖洞法：接下来我们再在保证唯一求解的情况下随机挖洞，即可生成一个可玩的数独。唯一求解的函数我并没能成功写出来，因此最后我只能随机数量随机位置进行挖洞，因此用户做出来的解可能与系统输出的解不一致。

# 4系统实现与测试

## 4.1系统实现

软件：codeblocks，windows7，core i7 4700MQ

本系统可实现的功能主要有：cnf文件读取，通过DPLL算法（优化前和优化后）求解SAT问题，求解结果在屏幕上输出，数独游戏生成，以及数独游戏答案求解与输出等。

程序在内存为8G的电脑上运行，可在短时间内求解出全部小型算例、全部中型算例。对于大型的算例，短时间内则无法计算出结果。

本系统包括主程序部分，SAT求解部分和数独部分。程序代码详见附录。

1. 主程序

主程序是对其他功能的整合，首先会给出主菜单提示用户选择操作。具体关系如图4.1所示。

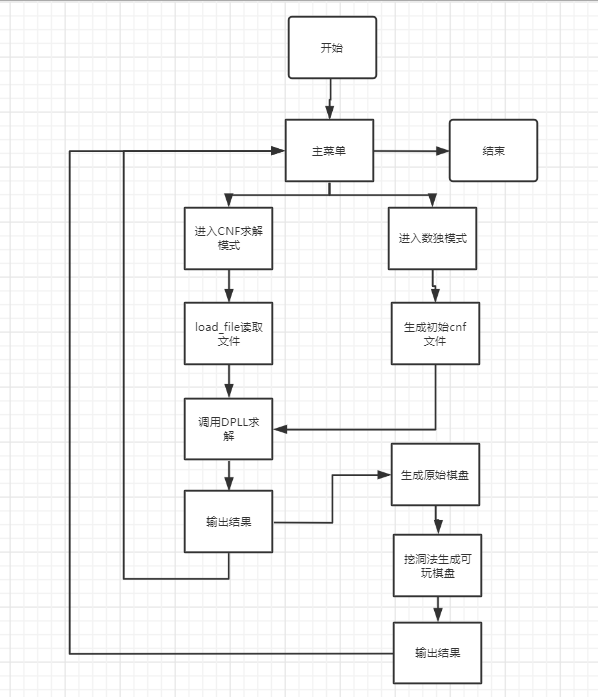


图4.1 主程序函数调用关系示意图

**函数名称：int main**

函数设计功能：用来输出主菜单，需要用户选择需要进行的功能。

1. CNF求解模式

这一部分包括了cnf文件读取、DPLL求解和时间计算等功能。

**函数名称：int InitSat(clause \*\*S, Var\_watch \*var\_watch);**

函数设计功能：用来初始化每个参数和分配指针空间，做好初始化。

**函数名称：int LoadCnf(clause \*\*S, Var\_watch var\_watch[], FILE \*fp);**

函数设计功能：用来加载cnf文件，包括建立二叉链表和监视链表。

**函数名称：int GetNum(FILE \*fp);**

函数设计功能：用来读取数字，把文件中的字符串转换成十进制数字存储起来并返回。

**函数名称：int Putclause(clause \*ctemp, int var, Var\_watch var\_watch[]);**

函数设计功能：用来建立监视链表，把每个文字所在子句的地址存在监视链表里面。

**函数名称：int abs(int a);**

函数设计功能：用来计算绝对值，主要是担心在后面的数组下标没转换成正数。

**函数名称：int DPLL(clause \*S,Var\_watch \*var\_watch);**

函数设计功能：读取二叉链表的头结点S，读取监视链表的表头Var\_watch，进行DPLL算法求解。

**函数名称：clause \*pan\_duan(clause \*S);**

函数设计功能：读取二叉链表的表头，函数用来判断二叉链表中是否存在单子句。

**函数名称：int unique(clause \*single,Var\_watch \*var\_watch);**

函数设计功能：进行单子句操作，把找到的单子句的变元赋值，并且将二叉链表中该变元的正负文字进行操作。

**函数名称：int empty\_set(clause \*q);**

函数设计功能：判断二叉链表是否为空。

**函数名称：int empty\_clause(clause \*q);**

函数设计功能：判断二叉链表中是否存在空子句。

**函数名称：int stragety(clause \*q);**

函数设计功能：变元选取策略，也是我前后优化的重点位置。

**函数名称：int reset(varWatch \*var\_watch,int temp);**

函数设计功能：把上一步变元赋值操作造成的二叉链表变化擦除。

**函数名称：int delete\_var(clause \*pt,int temp);**

函数设计功能：删除和该变元赋值相反的文字（比如该文字赋值为正，则删除二叉链表中所有为该变元的负值的文字）。

**函数名称：int search(Var\_watch \*var\_watch,int temp);**

函数设计功能：给变元赋值由temp确定后，对该变元的正负文字进行操作（比如把子句标记为满足还有把负文字从二叉链表中删除）。

**函数名称：int print\_answer();**

函数设计功能：打印出cnf文件答案。

**函数名称：int save\_answer(int result, char \*filename,int time);**

函数设计功能：把所得到的答案按照要求保存在res文件中。

**函数名称：int SAT();**

函数设计功能：sat求解器子菜单显示。

**函数名称：int answer\_test(clause \*S,FILE \*fp,varWatch \*var\_watch);**

函数设计功能：把res文件解析后回带进cnf检验是否正确。

**函数名称：int last\_branch(varWatch \*var\_watch);**

函数设计功能：回溯到上一层分支点，再进入另外一侧的子树搜索。

1. 数独

这一部分包括终盘生成，选取挖洞，结果唯一性验证（未完成）等功能。

**函数名称：status Sudoku();**

函数设计功能：数独功能主函数。

**函数名称：status InitSudoku(int level);**

函数设计功能：初始化数独，包括生成cnf文件。

**函数名称：status calculate(int n, int m);**

函数设计功能：计算字句数量函数，后面会用到。

**函数名称：status Generatecnf(int level);**

函数设计功能：初始化cnf文件，把文件头设置好。

**函数名称：status Rule1(int level, FILE \*fp);**

函数设计功能：把第一个约束条件转换为cnf字句。

**函数名称：status Rule2(int level, FILE \*fp);**

函数设计功能：把第二个约束条件转换为cnf字句。

**函数名称：status Rule3(int level, FILE \*fp);**

函数名称：void Rule2\_clause(int arr[], int level, int wordnum, int k1, int k2, FILE \*fp);

## 4.2系统测试

基准算例结果见表1-1：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 变元数 | 子句数 | 优化前  （ms） | 优化后  （ms） | 优化比率 |
| sat-20.cnf | 20 | 914 | 1 | 1 | 0% |
| unsat-5cnf-30.cnf | 30 | 42021 | 无解 | 无解 | \* |
| ais10.cnf | 181 | 31511 | 3905 | 6732 | -72.4% |
| sud00009.cnf | 303 | 2851 | 467 | 199 | 57.3% |

结果截图如下：

|  |  |
| --- | --- |
| 优化前 | 优化后 |
|  |  |
|  |  |
|  |  |
|  |  |

这里的ais算例我得到了很严重的负优化结果，打开了ais文件我才发现了这个文件的特殊性，因此我的第一种策略更加符合这个方法。

小型算例结果见表1-2：

表1-1 小型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 变元数 | 子句数 | 优化前（ms） | 优化后（ms） | 优化比例 |
| 7cnf20\_90000\_90000\_7.shuffled-20.cnf | 20 | 1532 | 184 | 126 | 31.5% |
| problem1-20.cnf | 20 | 91 | 0 | 1 | 0% |
| problem2-50.cnf | 50 | 80 | 119 | 1 | 99.1% |
| problem3-100.cnf | 100 | 340 | 627 | 13 | 97.9% |
| problem6-50.cnf | 50 | 100 | 171 | 2 | 98.8% |
| problem8-50.cnf | 50 | 300 | 36 | 4 | 88.9% |
| problem9-100.cnf | 100 | 200 | 128 | 15 | 88.2% |
| Problem11-100.cnf | 100 | 600 | 116 | 14 | 87.9% |
| tst\_v25\_c100.cnf | 25 | 100 | 1 | 1 | 0% |

由于优化决策的时候影响并不是特别大，但是由于我这里每次化简之后选取的时候都要计算词频，选取词频最高的，于是可以看出小型算例的优化程度很大，这是由于优化前的时候大量时间花在了选取高词频上了。

中型算例结果见表1-3：

表1-2 中型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 变元数 | 子句数 | 优化前（ms） | 优化后（ms） | 优化比例 |
| bart17.shuffled-231.cnf | 231 | 1166 | 130 | 20 | 84.6% |
| ec-mod2c-rand3bip-sat-250-2.shuffled-as.sat05-2534.cnf | 354 | 2596 | -1 | -1 | \* |
| m-mod2c-rand3bip-sat-220-3.shuffled-as.sat05-2490-311.cnf | 311 | 2192 | -1 | -1 | \* |
| m-mod2c-rand3bip-sat-240-2.shuffled-as.sat05-2519-338.cnf | 338 | 2376 | -1 | -1 | \* |
| m-SGI\_30\_80\_15\_90\_4-dir.shuffled-as.sat03-6-450.cnf | 450 | 28508 | -1 | -1 | \* |
| problem12-200.cnf | 200 | 1200 | 42511 | 24 | 99.9% |
| problem5-200.cnf | 200 | 320 | -1 | 46 | 100% |
| sud00001.cnf | 301 | 2780 | 98 | 80 | 18.3% |
| sud00009.cnf | 303 | 2851 | 441 | 193 | 56.2% |
| sud00012.cnf | 232 | 1901 | 164 | 61 | 62.8% |
| sud00021.cnf | 308 | 2911 | 6903 | 757 | 89.0% |
| sud00079.cnf | 301 | 2810 | 465 | 193 | 58.5% |
| sud00082.cnf | 224 | 1762 | 95 | 22 | 76.8% |
| sud00861.cnf | 297 | 2721 | 374 | 61 | 83.6% |
| tst\_v200\_c210.cnf | 200 | 210 | 13 | 4 | 69.2% |

可见对于稍微复杂的中型算例来说，优化程度比较明显，证明还是由于选取太花时间了。

大型算例结果见表1-4：

表1-3 大型算例测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 变元数 | 子句数 | 优化前  （ms） | 优化后  (ms) | 优化比率 |
| ec-iso-ukn009.shuffled-as.sat05-3632-1584.cnf | 1584 | 16587 | -1 | -1 | \* |
| ec-vda\_gr\_rcs\_w9.shuffled-6498.cnf | 6498 | 130997 | 2586 | 2227 | 16.1% |
| eh-dp04s04.shuffled-1075.cnf | 1075 | 3152 | 5265 | -1 | -00 |
| eh-vmpc\_25.renamed-as.sat05-1913-625.cnf | 625 | 76775 | -1 | -1 | \* |
| eh-vmpc\_29.renamed-as.sat05-1916-841.cnf | 841 | 120147 | -1 | -1 | \* |
| e-par32-3.shuffled-3176.cnf | 3176 | 10297 | -1 | -1 | \* |
| mc-sha0\_36\_5-50073.cnf | 50073 | 210223 | -1 | -1 | \* |
| m-vmpc\_32.renamed-as.sat05-1919-1024.cnf | 1024 | 161664 | -1 | -1 | \* |

从上述结果可以看出，只有两个大型算例可以算出结果，可能由于算例的某些特殊性才会出现这样的结果，多则6000+的变元却可以算出来，1000+多的却不能得出结果。

不可满足算例见结果表1-4：

表1-4 不可满足算例测试结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | 变元数 | 子句数 | 优化前 | 优化后 |
| eu-rand\_net60-25-10.shuffled-3000.cnf  （3000 8881） | 3000 | 8881 | -1 | -1 |
| gt-030.shuffled-as.sat05-1295.cnf  （870 24824） | 870 | 24824 | -1 | -1 |
| mm-2x2-6-6-s.1.shuffled-as.sat03-1499-400.cnf  （400 2576） | 400 | 2576 | 能 | 能 |
| php-010-008.shuffled-as.sat05-1171.cnf  （80 370） | 80 | 370 | 能 | 能 |
| SGI\_30\_70\_18\_80\_10-dir.shuffled-as.sat03-140-540.cnf  （540 40510） | 540 | 40510 | -1 | -1 |
| tst\_v10\_c100.cnf  （10 100） | 10 | 100 | 能 | 能 |
| u-5cnf\_3500\_3500\_30f1.shuffled-30.cnf  （30 420） | 30 | 420 | 能 | 能 |
| u-5cnf\_4300\_4300\_110.shuffled-110.cnf  （60 936） | 60 | 936 | -1 | -1 |

由上面不可满足的算例可以看出，有大量的算例无法得出最后的结果，这和我的变元选取策略有直接的关系，选择出现次数最多的变元作为决策方式并不能很好的解决各个不满足的算例和一些大型的算例，但是我优化之后就可以很大程度的提高了选择的时间，从而更加节约。这在之后的工作中可以再进行一定程度的修改，寻找更有效率的决策方式。在优化过程中减少了部分冗杂的代码外，将决策方式也进行了简单的优化，这也注定优化程度会由于算例的到特殊性而使优化程度不稳定，若对数据结构再进行优化的话，会有更高的优化程度和效率。

优化前的变元选取策略：选取每次化简之后频次最高的变元进行计算。

优化之后的策略，直接选取下一个未满足子句的第一个文字作为选取的变元赋值，加快了选取操作速度，大幅度提高了求解效率，并且可以解决所有的大型和绝大多数的中型算例。

Res答案验证测试：

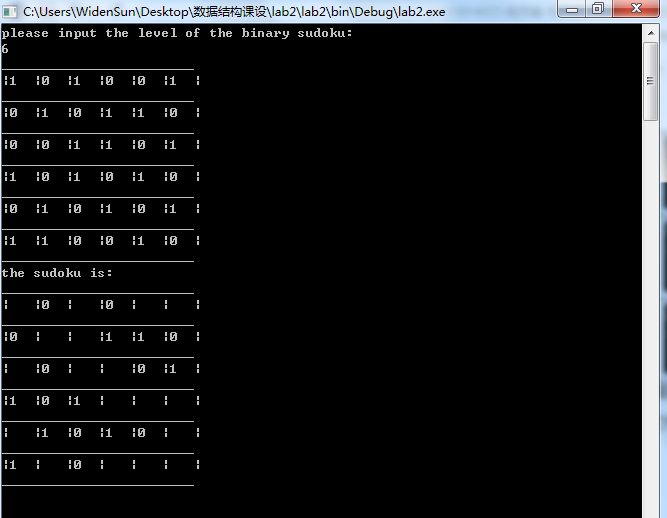
|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | sat-20.res | ais10.res | sud00009.res |
| sat-20.cnf |  |  |  |
| ais10.cnf |  |  |  |
| sud00009.cnf |  |  |  |

这里我们发现，必须是对应的res文件才能够对应名称相同的cnf文件，输出satisfied，否则，如果使用其他的res文件来验证，答案错误，会输出unsatisfied！

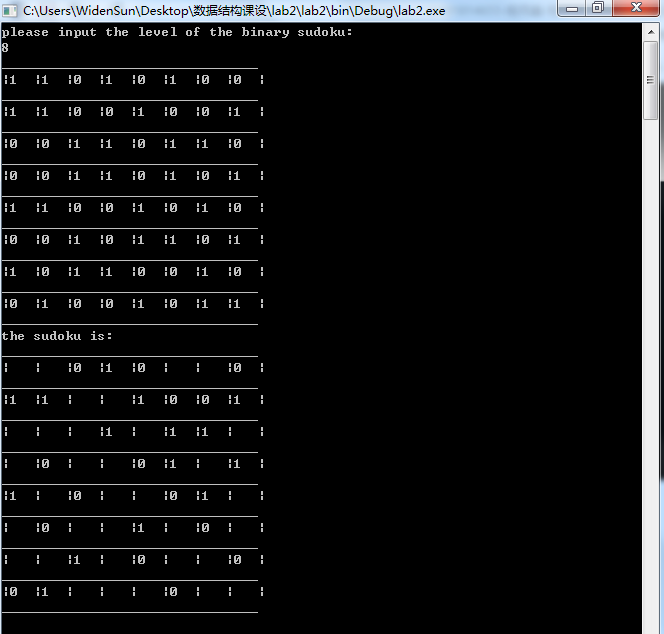
Cnf文件求解以及验证功能测试结束。

进行数独游戏测试：

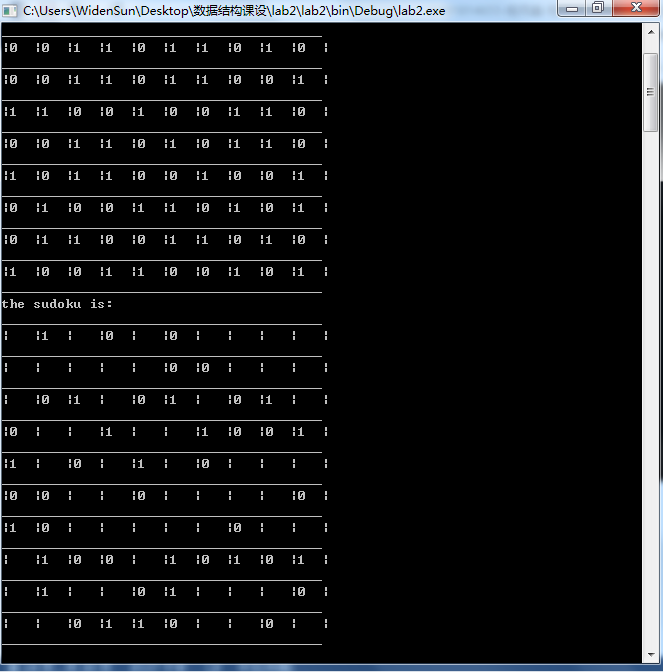
6\*6格局：



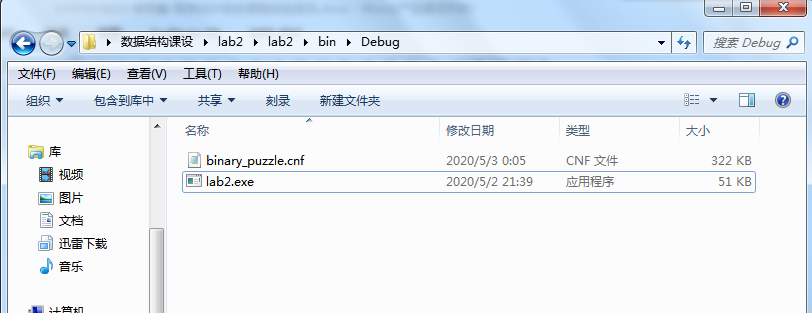
8\*8格局：



10\*10格局：



这里我的数独是直接随机生成的，并且是基于挖洞法生成的数独，上面显示的是我的数独答案，下面显示的是我的给用户玩家玩的数独游戏格局。



在exe文件目录下生成了对应的cnf文件方便我调用之前的cnf求解器。

到此数独功能测试完成。

# 5 总结与展望

## 5.1 全文总结

本次课程设计，我完成的主要工作如下：

（1）查阅相关资料，了解DPLL算法的思路。

（2）对任务书所要求的功能进行设计、分析。

（3）完成对cnf文件的读取，并将其存入设定的物理结构。

（4）用递归的方式实现了DPLL算法。4

（5）查阅参考文献，理解数独游戏生成的原理和方法。

（6）结合SAT求解，创建出阶数由用户选择的数独格局。

（7）设计交互式界面

## 5.2 工作展望

在今后的学习、研究、实践中，我将主要围绕以下几个方面开展工作：

1. 在已有的物理结构基础上，进一步改善优化数据结构，找到更合适、运算更快捷的存储结构，以进一步提高程序运行速度。
2. 学习优秀的变元选取策略，提高程序的运行速度。
3. 更深入的学习挖洞法，了解不同难度的挖洞法。
4. 思考并且能够归类不同类型的sat问题
5. 在这次的程序设计中的SAT求解部分，基本是以DPLL算法为基础进行对SAT问题的求解。但是在求解过程的变量决策的选择上还有很大的提升空间。以及数据结构上除了邻接表之外，还可以尝试其它的数据结构，比如胀缩数据结构进行新的尝试，再与现在的数据结构进行性能上的对比。我也了解到一种新的算法，WM算法是改进后的DPLL算法，在今后的工作中可以尝试利用WM算法去解决SAT问题，再观察优化程度。变量决策上还可以进行很大的改进，之前的决策过于简单导致比较依赖于算例的特殊性。
6. 这次由于自己的sat求解器还不是很成熟，我没有进行唯一解的保证，但是根据我的观察，数独生成的cnf文件也具有特性，觉得可以从这方面下手，找到适合求解数独的方法，才能快速计算保证唯一性。

# 6体会

这是我们第一个比较大的课设，我也付出了很多的时间精力去学习论文，但是其实这次老师所给的例子，个人觉得还是太具有偏向性了，我尝试了cdcl字句学习的方法居然比我之前的两个想法慢了不少，个人也感到十分的苦恼。

包括直到现在，我还是认为我优化前的频次变元选取策略应该比优化后的毫无根据的边缘选取策略效率高，但是并没有体现出来，只能说老师这次给的例子更加适合我的第二种想法。

当然我也没有全盘否定我最开始编写的频次统计变元选取策略，因为在ais.cnf这一类文件中，很明显我的优化前的频次统计策略更快，说明在遇到这类文件时候还是我的频次选取策略更靠谱。

通过大量的网络搜索和文献查找，我知道对于不同类型的cnf文件，都有适应它的算法，其实主要是因为dpll是一个完备性算法，时间开销是不可能减少太多。

其实我个人的想法还是区间搜索，在遇到比较大型的算例时更有可能在有限的时间内算出答案来。当然，现在主流的基于cdcl子句学习算法的方法我觉得必须要在更大型，并且更具有随机性的例子中才能体现出它的优势，而不是像我这里第二个撞大运的优化方法，在之后计算机性能上来之后，还是该使用具有数学证明的优化搜索策略进行优化。

总的来说，这次课设我阅读了不少中英文文献，对于我的文献检索能力有了很大的提高，希望能继续加油！

# 

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

[6]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.

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

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

# 附录A

**程序代码：见打包文件中SAT文件夹。**