華中科技大學

课程设计报告

题目：基于 SAT 的蜂窝数独游戏求解程序

课程名称：程序设计综合课程设计

专业班级：CS2201

学　　号：U202215357

姓　　名：王文涛

指导教师：李剑军

报告日期：2023.9.27

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：$[(t-t_o)/t]*100\%$,其中 t 为未对 DPLL 优化时求解基准算例的执行时间，t_o 则为优化 DPLL 实现时求解同一算例的执行时间。(15%)

（6）SAT 应用：将数双独游戏[5]问题转化为 SAT 问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。应用问题归约为 SAT 问题的具体方法可参考文献[3]与[6-8]。(15%)

口 参考文献

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

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

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

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

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

Twodoku： https://en.grandgames.net/multisudoku/twodoku

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

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

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

[9] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191

目录

1 引言

1.1 课题背景与意义

1.1.1 课题背景

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

1.1.2 课题意义

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。

1.2 国内外研究现状

国际上已经提出了各种不同的 SAT 求解器，包括基于回溯搜索的 DPLL 算法和其改进算法，以及国际上已经提出了各种不同的 SAT 求解器，包括基于回溯搜索的 DPLL 算法和其改进算法，以及基于局部搜索的算法。这些算法可以解决不同类型和规模的 SAT 问题，包括数独游戏等逻辑谜题。国内也有一些学者和学生对基于 SAT 的数独游戏求解程序进行了研究和实现，主要采用了 DPLL 算法或其变体，并利用 QT 等工具进行了界面设计和功能实现。基于 SAT 的蜂窝数独游戏求解程序是一个相对较新的课题，目前还没有找到很多相关的文献和资料。但是该课题已经有了一定的设计思路和实现方案，包括将蜂窝数独转化为 CNF 公式，利用 DPLL 算法进行求解，以及利用 QT 进行界面设计等。

1.3 课程设计的主要研究工作

(1)了解 SAT 问题与 CNF 文件。

(2)研究 dpll 算法，包括 dpll 算法的思想，如何化简 cnf 公式，如何回溯以及如何用 C 语言实现。

(3)研究蜂窝数独。包括蜂窝数独的规则，如何生成蜂窝数独游戏格局，如何将其规约为 SAT 问题，如何将语义编码转换为自然编码。

2 系统需求分析与总体设计

2.1 系统需求分析

(1)输入输出功能：包括程序执行参数的输入，SAT 算例 cnf 文件的读取，执行结果的输出与文件保存等。

(2)公式解析与验证：读取 cnf 算例文件，解析文件，基于一定的物理结构，建立公式的内部表示；并实现对解析正确性的验证功能，即遍历内部结构逐行输出与显示每个子句，与输入算例对比可人工判断解析功能的正确性。

(3)DPLL 过程：基于 DPLL 算法框架，实现 SAT 算例的求解。

(4)时间性能的测量：基于相应的时间处理函数，记录 DPLL 过程执行时间（以毫秒为单位），并作为输出信息的一部分。

(5)程序优化：对基本 DPLL 的实现进行存储结构、分支变元选取策略等某一方面进行优化设计与实现，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。

(6)SAT 应用：将数双独游戏[5]问题转化为 SAT 问题[6-8]，并集成到上面的求解器进行数独游戏求解，游戏可玩，具有一定的/简单的交互性。

2.2 系统总体设计

2.2.1 系统功能模块

1) 主控、交互与显示模块

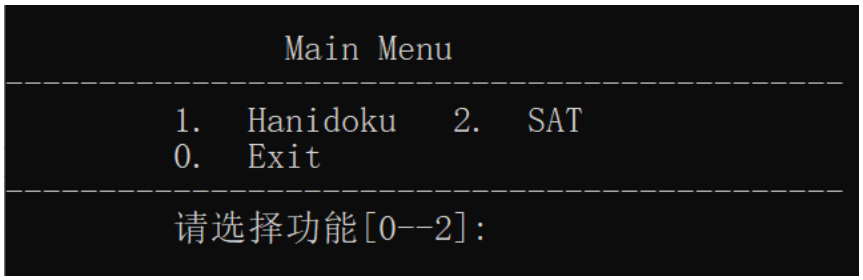


图 1 主菜单

2) CNF 解析模块

读取 cnf 文件中的信息并保存到一个二维链表中，以便之后使用 dpll 算法求解。

3) 核心 DPLL 模块

实现 dpll 算法，给出 cnf 的解。

4) 蜂窝数独模块

包括生成蜂窝数独游戏格局、将蜂窝数独问题归约为 SAT 问题、求出 SAT 问题解、给出蜂窝数独的解。

2.2.2 系统模块结构图

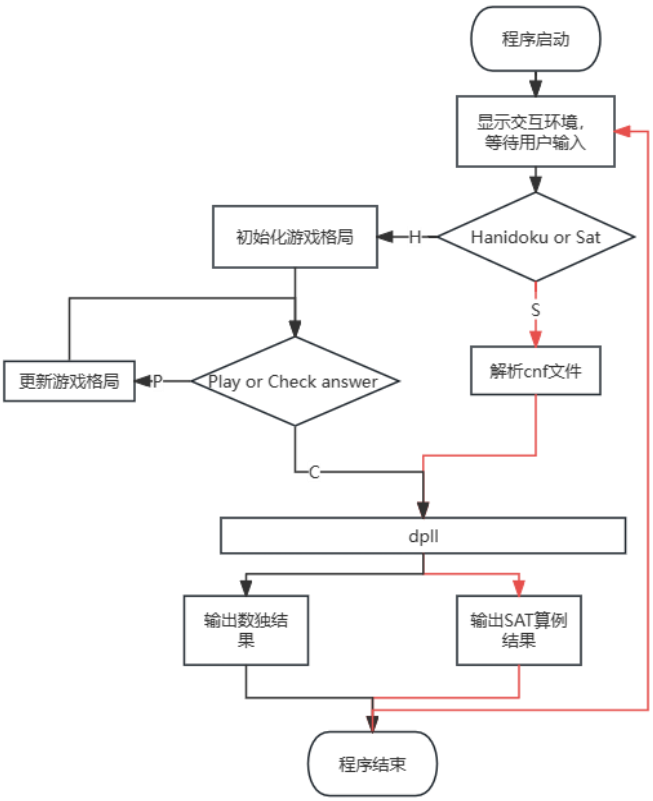


图 2 系统模块结构图

3 系统详细设计

3.1 有关数据结构的定义

3.1.1 需要处理的数据

表格 1 需要处理的数据

数据					
cnf	数据项	合取范式的第一个子句（head）	合取范式中的子句个数(size)	变元数目（varmax）	是否可满足（satisfied）
	数据类型	clause 类型的指针 clause*	整型（int）	整型（int）	整型（int）
文字(literal)	数据项	文字的值 value	指向下一个文字的指针 next		
	数据类型	整型 int	literal 类型的指针 literal*		
子句(clause)	数据项	指向第一个文字的指针 head	指向下一个子句的指针 next		
	数据类型	literal 类型的指针 literal*	clause 类型的指针 clause*		
结果(answer)	数据项	真值 value	程序用时 time		
	数据类型	整型指针 int*	整型 int		

3.1.2 数据在系统中的关联

cnf 中 head 指向第一个 clause 节点，size,varmax,与 satisfied 储存 cnf 文件的信息。clause 节点中的 head 指向第一个 literal 节点，next 指向下一个 clause 节点。litral 中 value 储存每个文字的值，next 指向下一个文字。answer 中 value 数组储存每个数的真值，用来输出答案，time 记录耗时。

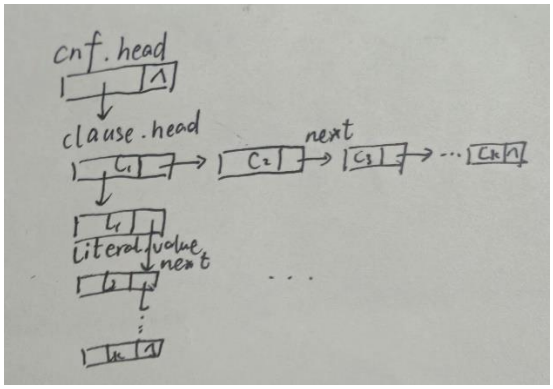


图 3 数据在系统中的关联

3.2 主要算法设计

3.2.1 各函数设计

1. cnf* read_cnf(char* filename)

1） 操作结果：将 cnf 文件中的内容储存到链表中，读取变元个数，子句数目，并返回 cnf 指针。

2） 设计思想：过滤掉信息段之前的字符，循环读取数字，赋值到链表中。

3） 流程图

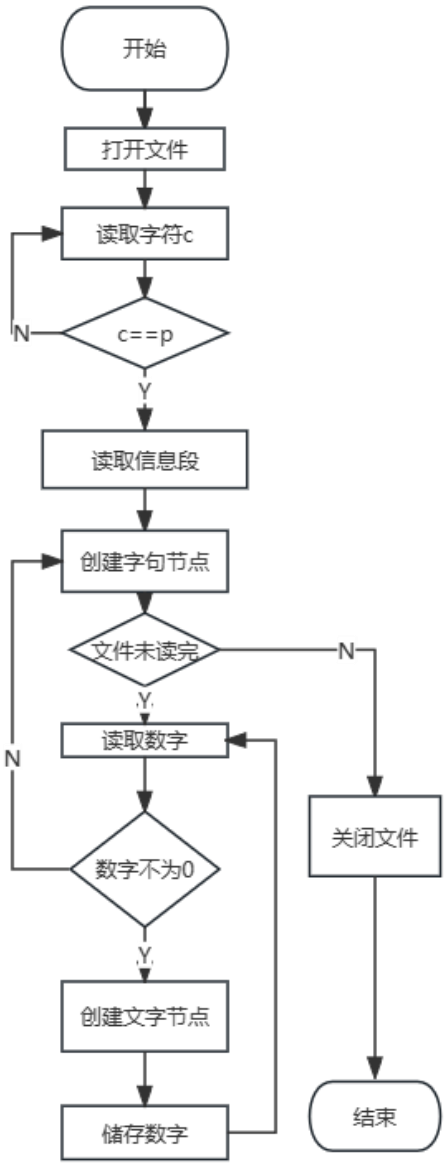


图 4 read_cnf 流程图

2. int print_answer(cnf* formula, answer ans, char* filename)

1) 操作结果：将答案生成到 res 文件中。

2) 设计思想：在原 cnf 文件的同级文件夹中创建一个.res 后缀的新文件，将 answer 结构体与 cnf 结构体中的储存的信息全部输出，最后关闭文件。

3. void checkanswer(cnf* formula, char* filename)

1) 操作结果：根据链表生成检查文件，人工查看是否与提供的 cnf 文件相同。

2) 设计思想：在原 cnf 文件的同级文件夹中创建一个.txt 后缀的新文件，将链表中的储存的信息全部输出，最后关闭文件。

4. void assign_answer(int l, answer answer, int trueth)

1) 操作结果：在答案数组中赋值。

2) 设计思想：根据传的 trueth 参数在答案数组中对文字 l 进行赋值。

3) 流程图

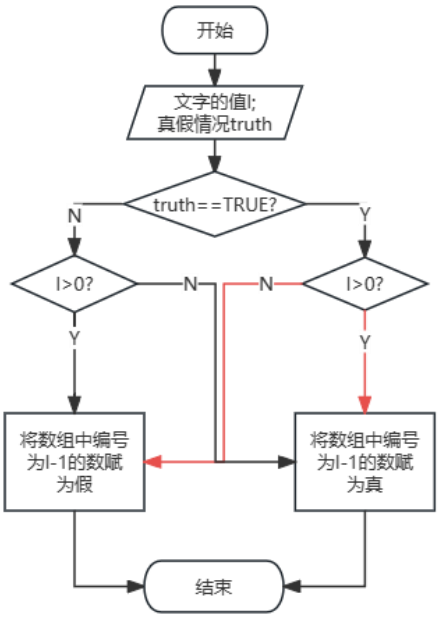


图 5 print_answer 流程图

5. int find_unit_clause(cnf* formula, answer answer)

1) 操作结果：找到单子句并返回子句的值。

2) 设计思想：遍历链表，找到第一个文字的后继为空的子句，返回文字的值。若无单子句则返回 0。

6. unit_propagation(cnf* formula, answer answer)

1) 操作结果：根据 find_unit_clause 函数找到的单子句与单子句传播规则更新链

表。

2) 设计思想：循环调用 find_unit_clause 函数直到不存在单子句，再调用 assign_cnf 函数更新链表。

7. assign_cnf(int var, cnf* formula,answer ans)

1) 操作结果：将链表中与含有相同关键字的子句删除，将相反的的关键字删除，返回新的链表。

2) 设计思想：遍历链表，如果找到与关键字相同的文字，就把子句删除，如果找到与关键字相反的文字，就把文字删除。

3) 流程图

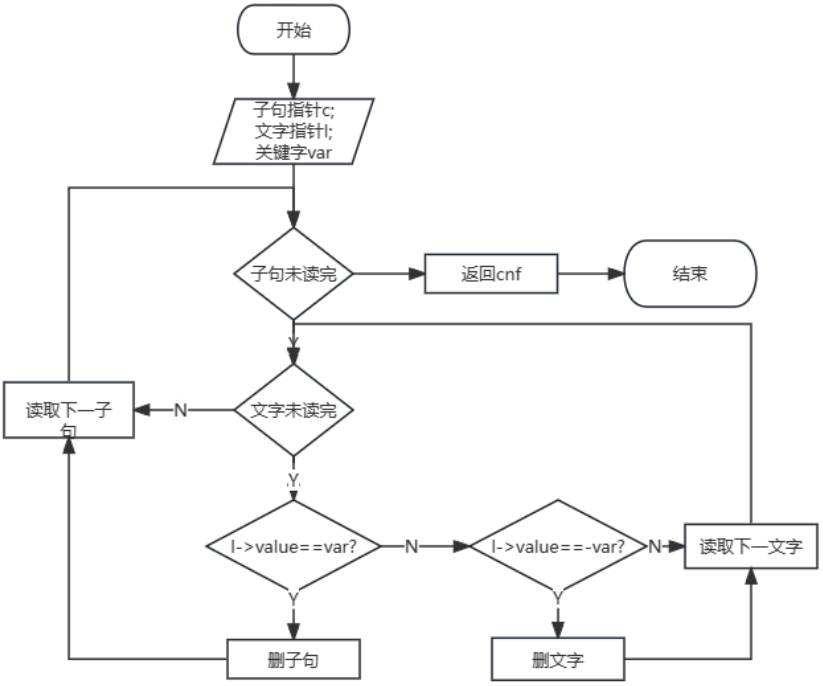


图 6 assign_cnf 流程图

8. is_formula_satisfied(cnf* formula)

1) 操作结果：判断 cnf 状态,返回 TRUE，FALSE，或者 UNKNOWN。

2) 设计思想：子句集为空说明 cnf 已满足，存在空子句说明 cnf 不可满足,若都不是，说明 cnf 可满足性未知。

9. cpycnf(cnf* formula)

1) 操作结果：将链表复制给另一个链表。

2) 设计思想：重新创建一个链表，将原链表的值传过去。

10. print_answer(cnf* formula, answer ans, char* filename)

1) 操作结果：将 answer 中的文件储存到.res 文件中。

2) 设计思想：创建文件，将 cnf 的可满足性，每个文字的真值，dpll 算法用时输出到文件中。

11. checkanswer(cnf* formula, char* filename)

1) 操作结果：生成检查文件。

2) 设计思想：创建文件，将链表中的数据按 cnf 的格式输出到文件中。

12. dpll(cnf* formula, answer answer)

1) 操作结果：返回 cnf 的可满足性，将答案写在 answer 中。

2) 设计思想：先单子句传播，再判断 cnf 的可满足性，若可满足则返回 TRUE,不可满足则返回 FALSE,再选取一个关键字赋为正，将链表复制一份，递归直到返回值，若返回 TRUE 则返回 TRUE,若返回 FALSE,则将关键字赋为负递归。

3) 流程图

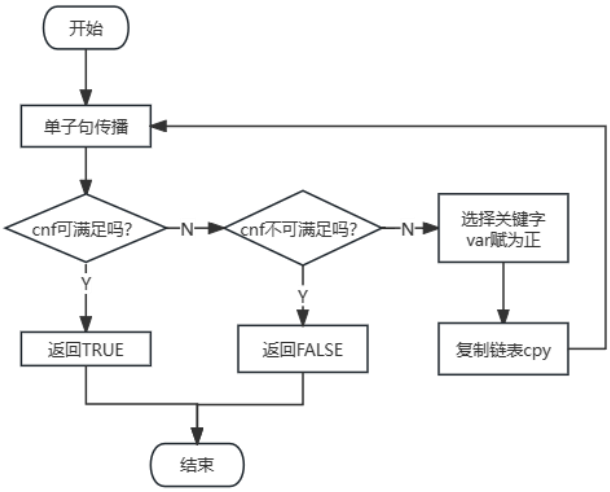


图 7 dpll 流程图

3.2.2dpll 算法的优化

优化前的算法，二维链表只读，不做任何操作，而使用一个大小为变元数的整型数组来储存每一个变元的真值，若为真则赋为 1，若为假则赋为 0，若为未知则赋为-1。单子句传播时，若子句中只有一个文字值未知，则为单子句，若有一个文字值为 1，则子句已满足，再根据单子句的文字值，将与该文字相反的文字赋为 1 或 0。判断 cnf 的满足性时，若所有子句都满足，则 cnf 满足，若有一个子句中文字都为假，则 cnf 不满足。最后输出数组中储存的每个变元的真值。

优化后的算法,直接对二维链表操作，另外使用一个数组来储存真值。单子句传播时，直接将文字或子句从链表中删除，若链表为空，则 cnf 满足，若存在空子句，则 cnf 不满足。优化后的链表随着递归层数增加，数据越来越少，时间复杂度更低，但是由于要复制整份链表，导致占用内存较大，空间复杂度更高。

4 系统实现与测试

4.1 系统实现

4.1.1 软硬件环境

1. 硬件：操作系统：Windows 10, CPU：AMD Ryzen 5 2500U,显卡： AMD Radeon™ Vega 8 Graphics,内存：RAM: 6045 Mb
2. 软件：编译器：VS2022，Win64

4.1.2 数据类型的定义

```
struct literal {
    int value = 0; //  文字的值
    literal* next = {}; //  指向下一个文字的指针
};
struct clause {
    literal* head; //  指向第一个文字的指针
    clause* next; //  指向下一个子句的指针
};
struct cnf {
    clause* head; //  合取范式的第一个子句
    int size; //  合取范式中的子句个数
    int varmax; //变元数目
    int satisfied;//是否可满足
};
struct answer {
```

```
    int* value;//记录真值
    int time;//程序用时
};
```

4.2 系统测试

主要选取 dpll 模块与蜂窝数独生成模块进行测试。

1. Dpll 模块
   1) 测试方法：选取不同大小的测试用例，使用优化前和优化后的 dpll 算法分别求解，比较两次的答案与用时。
   2) 运行结果：首先比较不同算法的结果

      文件 problem2-50

```
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
```

图 8 优化前

```
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 3ms
```

图 9 优化后

文件 sud00079

```
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 435ms
```

图 10 优化前

```
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 102ms
```

图 11 优化后

文件 tst_v10_c100

```
s 0
v 1 2 0 0 5 -6 7 8 -9 -10
t 2ms
```

图 12 优化前

```
s 0
v 1 2 0 0 5 -6 7 8 -9 -10
t 1ms
```

图 13 优化后

对比上面三组结果，可以发现，不管是可满足的算例还是不可满足的算例，优化前的算法和优化和的算法给出的结果都是一致的。说明算法能够给出正确的结果。

接下来计算算法优化后的优化率。

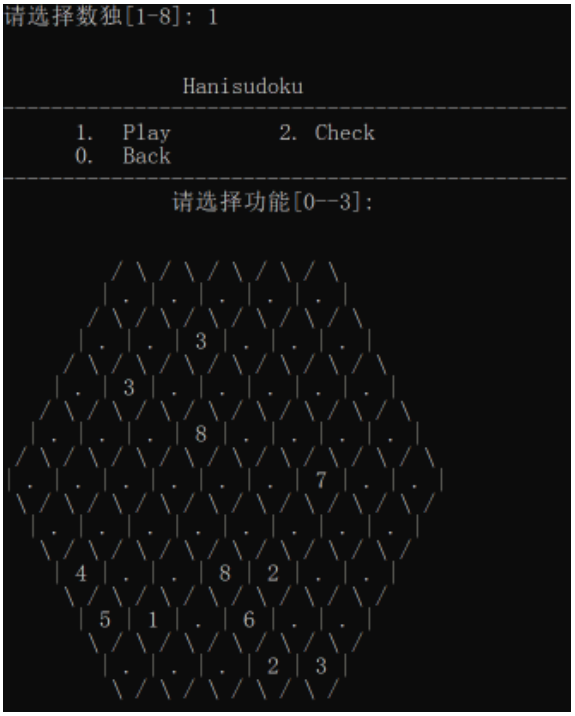
算例名称	算例属性			时间（ms）		
	子句数	变元数	是否有解	优化前	优化后	优化率
7cnf20_90000_90000_7.shuffled-20	1532	20	是	581	26	95.52%
problem1-20	91	20	是	1	0	/
problem2-50	80	50	是	2	3	-50.00%
problem3-100	340	100	是	627	9	98.56%
problem5-200	320	200	是	3997	54	98.65%
problem12-200	1200	200	是	3676	32	99.13%
sud00861	2721	297	是	180	64	64.44%
sud00082	1762	224	是	23	30	-30.43%
tst_v10_c100	100	10	否	1	1	0.00%
unsat-5cnf-30	420	30	否	542	417	23.06%

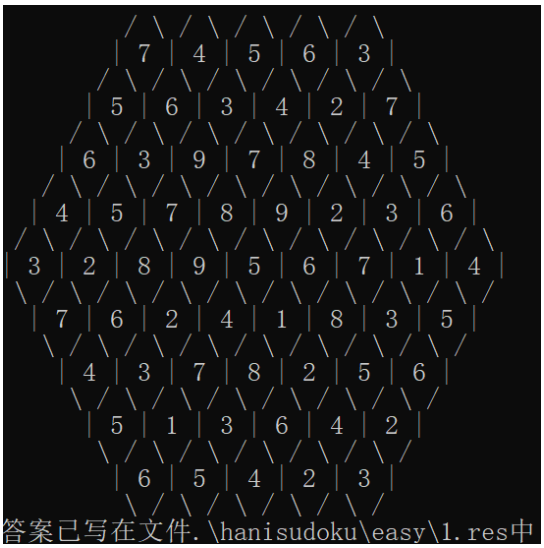
从这个表格中可以看出优化后的变元在处理大多数算例时有比较高的优化

率（90%以上），但是在处理一些算例时会有负优化。

2. 蜂窝数独生成

先根据给出的已知数字生成蜂窝数独题目，再使用SAT求解器解出答案，查看答案是否正确且唯一。





可以看出蜂窝数独的生成与求解都是没有问题的。

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）在网上查找关于 SAT 问题，dpll 算法和蜂窝数独游戏的相关资料，查阅国内外的相关文献，了解国内外对于这些问题的研究。了解 dpll 算法的数学原理，研究蜂窝数独的规则与玩法。

（2）编写 SAT 求解器，将整个功能拆分成很多的小函数，包括读取 cnf 文件，创建二维链表，复制二维链表，查找单子句，单子句传播，判断 cnf 的可满足性，维护一个储存答案的结构体，将答案以文件形式储存。

（3）创建一个可互动的蜂窝数独游戏，首先编写代码生成蜂窝数独的基本 CNF 文件，然后创建多级菜单，通过打表的方式将语义编码转换成从一开始的自然编码，将蜂窝数独打印到屏幕上，给玩家提供修改蜂窝数独的选项，将蜂窝数独转换成 cnf 文件求解，将答案储存到文件中。

5.1 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）优化 SAT 求解器，比如使用启发式是选取变元的策略，优化储存结果，使用指针栈来储存产生的变化，使用更多的化简 CNF 的算法，使 SAT 求解器的效率更高。

（2）优化蜂窝数独游戏，使游戏的操作更方便，可以自动生成不同难度的数独游戏，还可以加上给玩家的提示功能。

（3）把蜂窝数独游戏给周围的人玩，吸取他们的建议。

6 体会

这次试验应该是我目前做过难度最大，投入精力最多的一次实验了，因为在假期时想着开学后还有两个星期的时间来完成，就只做了很少的工作。结果实验的难度远远超出了我的想像，在两个星期里，我几乎把每天的空余时间都放在这上面了，很多时候连睡觉的时间也用来写代码。实际上大部分时候都在 debug，因为这种递归函数不方便追踪到具体的步骤，更因为要处理的数据太大，不可能一步步地调试，导致 debug 非常困难。为此，我在网上搜索很多如何 debug 的教程，自己也领悟了很多 debug 的方法。在构建二维链表，删除子句或者文字的时候，遇到了很多的 bug，这让我意识到自己对于链表的使用还不够熟练，在 debug 的过程中，我对链表的理解也更加深刻了。在测试算例的时候，我的电脑崩溃了好几次，我之后通过任务管理器发现应该是爆内存了，我这才深刻地理解了减少内存占用，降低空间复杂度的重要性。在写蜂窝数独的 cnf 文件时，遇了离散数学的问题，最后请教了同学才搞定，果然好学好算法，首先要学好数学。如何将语义编码和自然编码相互转换的问题也困扰了我很久，最后学到了打表的方法。总的来说，这次试验消耗了我大量的心力，但是也让我学习到了很多新的东西。完成这段试验的过程非常有成就感，看着自己程序越来越完整，功能越来越多，十分开心。

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

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

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

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

[9] Sudoku Puzzles Generating: from Easy to Evil.
http://zhangroup.aporc.org/images/files/Paper_3485.pdf

[10] 薛源海，蒋彪彬，李永卓. 基于“挖洞”思想的数独游戏生成算法. 数学的实践与认识,2009,39(21):1-7

[11] 黄祖贤. 数独游戏的问题生成及求解算法优化. 安徽工业大学学报(自然科学版), 2015,32(2):187-191