

**课程设计报告**

**题目：基于SAT的百分号数独游戏求解程序**

**课程名称：程序设计综合课程设计**

**专业班级： CS2401**

**学 号： U2024114514**

**姓 名： Bottledkzk**

**指导教师： 李剑军**

**报告日期： 2025年9月28日**

**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%)

## 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

[任务书 I](#_Toc210045380)

[设计内容 I](#_Toc210045381)

[设计要求 I](#_Toc210045382)

[参考文献 II](#_Toc210045383)

[**目录 III**](#_Toc210045384)

[**1引言 4**](#_Toc210045385)

[1.1课题背景与意义 4](#_Toc210045386)

[1.1.1理论价值 4](#_Toc210045387)

[1.1.2创新意义 4](#_Toc210045388)

[1.2国内外研究现状 5](#_Toc210045389)

[1.2.1国际研究状况 5](#_Toc210045390)

[1.2.2国内研究状况 5](#_Toc210045391)

[1.2.3数独求解技术 5](#_Toc210045392)

[1.3课程设计的主要研究工作 6](#_Toc210045393)

[**2系统需求分析与总体设计 7**](#_Toc210045394)

[2.1系统需求分析 7](#_Toc210045395)

[2.1.1 SAT求解功能 7](#_Toc210045396)

[2.1.2 百分号数独功能 7](#_Toc210045397)

[2.1.3 非功能需求 7](#_Toc210045398)

[2.2系统总体设计 8](#_Toc210045399)

[2.2.1 系统架构设计 8](#_Toc210045400)

[2.2.2 功能模块划分 8](#_Toc210045401)

[2.2.3 模块接口设计 10](#_Toc210045402)

[2.2.4 数据流设计 10](#_Toc210045403)

[2.2.5 关键技术选择 11](#_Toc210045404)

[**3系统详细设计 12**](#_Toc210045405)

[3.1有关数据结构的定义 12](#_Toc210045406)

[3.1.1 系统数据类型详述 12](#_Toc210045407)

[3.1.2 数据结构关系 14](#_Toc210045408)

[3.1.3 数据结构设计原理 15](#_Toc210045409)

[3.2 主要算法设计 15](#_Toc210045410)

[3.2.1 DPLL核心算法实现 15](#_Toc210045411)

[3.2.2 CDCL算法优化设计 17](#_Toc210045412)

[3.2.3 数独SAT编码算法 19](#_Toc210045413)

[3.2.4 图形界面交互设计 20](#_Toc210045414)

[3.2.5 性能优化策略 20](#_Toc210045415)

[3.2.6 数据流整合 20](#_Toc210045416)

[3.2.7 小结 21](#_Toc210045417)

[**4系统实现与测试 22**](#_Toc210045418)

[4.1系统实现 22](#_Toc210045419)

[4.1.1 开发环境配置 22](#_Toc210045420)

[4.1.2 函数模块说明 22](#_Toc210045421)

[4.1.3 模块调用关系 24](#_Toc210045422)

[4.1.4 模块间协作机制 26](#_Toc210045423)

[4.2系统测试 27](#_Toc210045424)

[4.2.1 常用软件测试方法 27](#_Toc210045425)

[4.2.2 关键模块测试 27](#_Toc210045426)

[**5总结与展望 32**](#_Toc210045427)

[5.1全文总结 32](#_Toc210045428)

[5.2工作展望 32](#_Toc210045429)

[5.2.1 算法深度优化 33](#_Toc210045430)

[5.2.2 工程级求解器开发 33](#_Toc210045431)

[5.2.3 图形化项目拓展 33](#_Toc210045432)

[5.2.4 完整项目开发实践 33](#_Toc210045433)

[5.2.5 技术栈扩展 33](#_Toc210045434)

[**6 体会 34**](#_Toc210045435)

[**参考文献 35**](#_Toc210045436)

# 1引言

## 1.1课题背景与意义

SAT（布尔可满足性问题）是计算机科学和人工智能领域的核心基础问题，它研究的是如何确定一个给定的命题逻辑公式是否存在满足其所有子句的变量赋值。作为第一个被证明的NP完全问题，SAT在计算复杂性理论中具有重要地位，同时在实际应用中也有着广泛的用途，包括硬件验证、软件测试、人工智能规划、协议安全验证等多个领域。

### 1.1.1理论价值

SAT问题的理论价值在于它为计算复杂性理论提供了重要的研究对象。Cook-Levin定理证明了SAT是NP完全的，这意味着任何NP问题都可以在多项式时间内归约为SAT问题。这一特性使得SAT成为研究NP问题的重要桥梁，也为算法设计和优化提供了理论基础。

在实际应用层面，随着SAT求解技术的不断发展，现代SAT求解器已经能够处理包含数百万变量和子句的大规模工业实例。特别是在集成电路设计、形式化验证、人工智能规划等领域，SAT求解器已经成为不可或缺的工具。例如，在硬件设计过程中，利用SAT求解器可以进行等价性检查、模型检测等关键任务。

### 1.1.2创新意义

数独游戏作为一种经典的逻辑谜题，具有明确的规则约束和唯一的解要求，这使其成为研究约束满足问题的理想模型。百分号数独作为数独的一种变体，在传统数独的行、列、宫约束基础上，增加了撇对角线约束和窗口约束，进一步提高了问题的复杂性和挑战性。

本课题的创新意义在于将SAT求解技术应用于百分号数独这一特定问题领域，通过设计高效的DPLL求解器及其优化版本CDCL算法，实现了一个完整的数独游戏求解系统。这不仅深化了对SAT算法原理的理解，也探索了逻辑推理在实际游戏中的应用价值，为类似约束满足问题的求解提供了可借鉴的技术方案。

## 1.2国内外研究现状

SAT求解算法的研究经历了从基础理论到高效实现的漫长发展过程。早期的研究主要集中在算法的基础框架上，其中DPLL（Davis-Putnam-Logemann-Loveland）算法作为最经典的完备SAT求解算法，自1960年代提出以来一直是研究的重点。DPLL算法基于深度优先搜索和回溯机制，通过单子句传播和分裂策略不断简化公式，其核心思想至今仍被现代SAT求解器所采用。

### 1.2.1国际研究状况

国际上，SAT求解技术的研究呈现出快速发展的态势。近年来，CDCL（冲突驱动子句学习）算法作为DPLL的重要扩展，通过引入冲突分析、子句学习、非时序回溯等机制，显著提高了求解效率。国际SAT竞赛每年都会推出新的基准测试集，促进了各研究团队之间的技术交流与竞争。欧洲的MiniSat、美国的Z3、中国的MapleSAT等求解器在国际上都具有重要影响力。

### 1.2.2国内研究状况

在国内，SAT研究也取得了显著进展。中国科学院、清华大学、北京大学等科研机构在SAT算法优化、并行求解、应用拓展等方面都做出了重要贡献。特别是在SAT与人工智能、形式化验证等交叉领域的研究，为我国在理论基础研究和工业应用方面奠定了坚实基础。

### 1.2.3数独求解技术

在数独求解技术方面，国内外研究者提出了多种求解方法。传统的数独求解算法包括回溯法、约束传播、舞蹈链算法等。而基于SAT的数独求解方法则提供了一种新的思路：通过将数独规则编码为CNF范式，利用成熟的SAT求解器进行求解。这种方法不仅保证了求解的正确性和完备性，还能够利用SAT求解器的优化技术提高求解效率。

对于百分号数独这一特定变种，目前国内外的专门研究相对较少。大多数研究集中在传统数独的生成和求解上，对于包含额外约束的变种数独的研究还有待深入。本课题正是在这一背景下，探索百分号数独的SAT编码方法和求解技术，填补了这一领域的研究空白。

## 1.3课程设计的主要研究工作

本课程设计围绕基于SAT的百分号数独游戏求解程序展开，主要完成了以下研究工作：

**（1）实现了基础的DPLL求解器**。设计并实现了CNF公式的解析模块，能够正确读取标准DIMACS格式的CNF文件，并建立相应的内部数据结构。实现了完整的DPLL算法框架，包括单子句传播、纯文字消除、分裂策略等核心组件。通过精心设计的数据结构，确保了算法的高效执行和正确性。

**（2）开发了CDCL优化算法**。在基础DPLL的基础上，引入了冲突分析机制，能够在发现冲突时分析冲突原因并学习新的子句。实现了VSIDS变量活动度评估策略，动态调整变量选择优先级。设计了非时序回溯机制，避免不必要的搜索空间探索。这些优化措施显著提高了求解器的性能。

**（3）完成了百分号数独的SAT编码**。深入分析了百分号数独的规则特点，设计了完整的约束编码方案。除了传统数独的行、列、宫约束外，重点实现了撇对角线约束和窗口约束的编码。开发了数独问题到SAT问题的自动转换模块，能够将任意百分号数独实例转化为等价的CNF公式。

**（4）构建了图形化用户界面**。使用EasyX图形库开发了友好的数独游戏界面，支持数独的生成、求解、验证等交互功能。界面设计考虑了用户体验，提供了清晰的可视化反馈和操作指引。

**（5）进行了全面的测试评估**。设计了系统的测试方案，使用多个标准SAT算例和自定义数独实例对系统进行测试。对比分析了DPLL和CDCL算法的性能差异，验证了优化措施的有效性。测试结果表明，本系统能够正确高效地求解各类SAT问题和百分号数独实例。

# 2系统需求分析与总体设计

## 2.1系统需求分析

基于课程设计要求，本系统需要实现SAT求解和百分号数独游戏两大核心功能模块。

### 2.1.1 SAT求解功能

**（1）CNF文件解析功能**：系统能够正确读取标准DIMACS格式的CNF文件，解析其中的变量声明、子句信息，并建立内部数据结构表示。

**（2）DPLL求解功能**：实现完整的DPLL算法框架，包括单子句传播、纯文字消除、分裂策略、回溯机制等核心组件。

**（3）CDCL优化功能**：在DPLL基础上实现冲突驱动子句学习机制，包括冲突分析、子句学习、非时序回溯、VSIDS变量选择策略等优化技术。

**（4）求解结果输出**：将求解结果按照规范格式输出到.res文件，包括可满足性判定结果、变量赋值序列、执行时间等信息。

### 2.1.2 百分号数独功能

**（1）数独生成功能**：支持两种数独生成方式：从文件读取已有数独格局和基于挖洞算法自动生成唯一解数独。要求生成的数独具有合法性且唯一解。

**（2）SAT编码功能**：将百分号数独问题自动转换为等价的CNF公式，包括基本约束（单元格、行、列、宫）和特殊约束（撇对角线、窗口约束）。

**（3）图形交互功能**：提供友好的图形用户界面，支持数独的数字输入、清除、求解、验证等交互操作。界面应清晰显示数独棋盘和操作按钮。

**（4）求解验证功能**：能够调用SAT求解器对用户填写的数独进行正确性验证，提供即时反馈。

### 2.1.3 非功能需求

（1）**性能需求：**对于小型算例，要求求解时间在1s以内，以满足数独求解和生成的低延迟需求。

**（2）可靠性需求**：系统应能够稳定运行，不出现崩溃或死锁，对异常输入具有容错处理能力，文件读写操作应保证数据的完整性。

**（3）可用性需求：**界面设计简洁直观，操作流程清晰，提供明确的操作提示和错误信息。

**（4）数据分析需求：**系统需要处理SAT求解的相关数据和数独游戏相关数据。SAT求解数据包括：CNF文件数据——变量数、子句数、文字序列；求解状态数据——变量赋值、决策层级、冲突信息；性能统计数据：执行时间、回溯次数、学习子句数。数独游戏相关数据包括：数独格局数据——9×9的整数矩阵；约束规则数据——基本约束和特殊约束的定义；用户交互数据——鼠标点击位置、输入数字。

## 2.2系统总体设计

### 2.2.1 系统架构设计

本系统采用核心功能模块化架构，分为SAT求解器，DPLL核心算法，数独游戏三大核心，整体架构如图2-1所示。

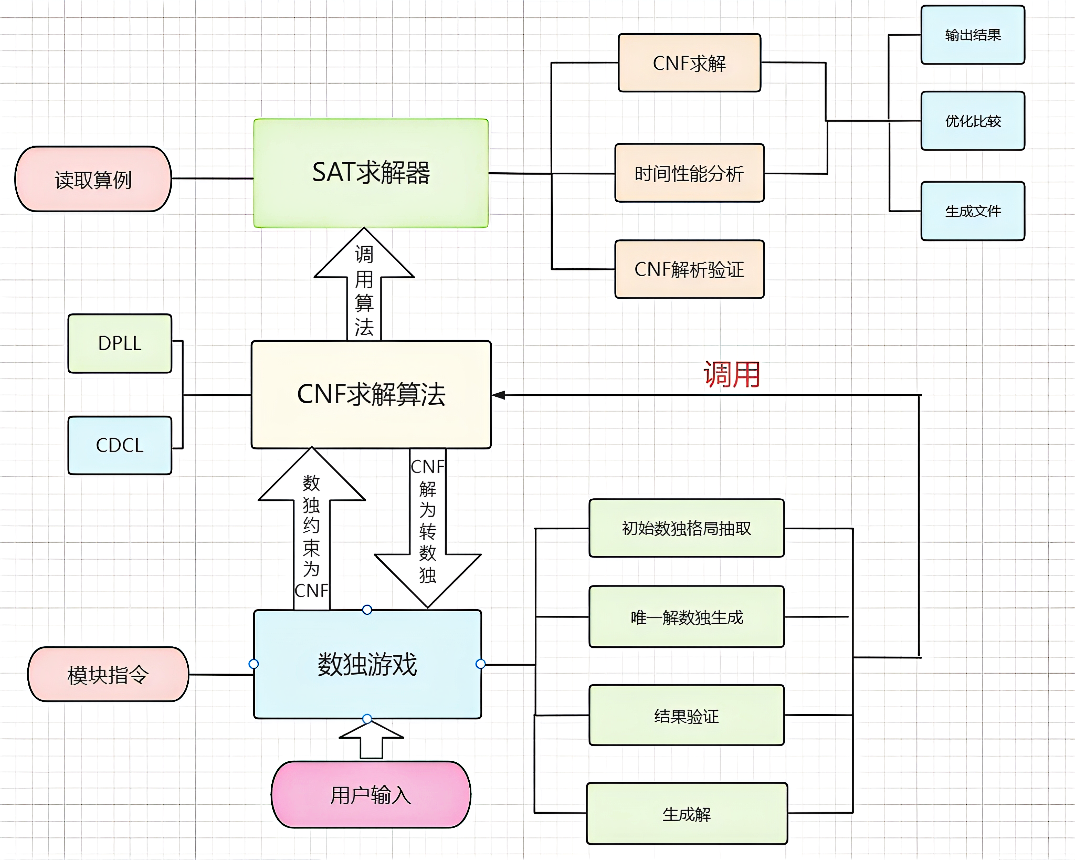


图2-1 系统架构设计

### 2.2.2 功能模块划分

根据系统需求，将系统分为五大核心模块：

**（1）主控模块（Main Controller）**

功能职责：系统入口点，负责模块间的协调调度

核心程序：SatSolver和Sudoku两大程序，初始化各模块，进行各项操作并输出结果

依赖关系：调用所有其他模块的接口函数

**2. SAT求解模块（SAT Solver）**

功能职责：实现DPLL和CDCL算法，完成SAT问题的求解

核心文件：DPLL.h, CDCL.h, SatHead.h

子模块：

CNF解析子模块：parse()函数

DPLL求解子模块：DPLL\_DpllPro()函数

CDCL优化子模块：solve()函数

结果输出子模块：print\_model()函数

**3. 数独处理模块（Sudoku Processor）**

功能职责：处理数独相关的所有业务逻辑

核心文件：sudukoH.h

子模块：

数独生成子模块：generate\_sudoku\_puzzle()函数

SAT编码子模块：generate\_cnf()函数

解提取子模块：extract\_solution()函数

验证子模块：validateSudoku()函数

**4. 图形界面模块（GUI Module）**

功能职责：提供图形化用户界面，处理用户交互

核心文件：Sudoku.cpp

子模块：

界面绘制子模块：drawBoard(), drawButton()函数

事件处理子模块：鼠标点击、键盘输入处理

动画效果子模块：按钮反馈、数字输入动画

**5. 文件管理模块（File Manager）**

功能职责：统一管理文件的读写操作

功能函数：

读取CNF文件：DPLL\_createCnf()函数，parse()函数

保存求解结果：DPLL\_generateResult()函数，print\_model()函数

数独文件操作：read\_sudoku\_from\_file()函数，generate\_cnf()函数

### 2.2.3 模块接口设计

各模块间通过清晰的接口进行通信，主要接口定义如下：

**（1）SAT求解模块接口**

**————————————————————————————————**

// CNF解析接口

int parse(Solver\* s, char\* filename);

// 求解接口

int solve(Solver\* s);

// 结果输出接口

void print\_model(Solver\* s, FILE\* fout);

**————————————————————————————————**

**（2）数独处理模块接口**

**————————————————————————————————**

// 数独生成接口

bool generate\_sudoku\_puzzle(char\* output);

// SAT编码接口

void generate\_cnf(int sudoku[SIZE][SIZE], const char\* filename);

// 解提取接口

void extract\_solution(Solver\* solver, int sudoku[SIZE][SIZE]);

**————————————————————————————————**

### 2.2.4 数据流设计

系统的典型数据处理流程如图2-2所示：

手机屏幕截图

AI 生成的内容可能不正确。

图2-2 数独求解系统流程图

### 2.2.5 关键技术选择

（1）算法选择依据：

选择DPLL作为基础算法，因其经典且易于理解实现；

选择CDCL作为优化方案，因其在实践中表现优异；

选择VSIDS变量选择策略，平衡性能和实现复杂度。

**（2）数据结构选择：**

使用链表结构存储子句，适应动态变化的公式规模；

使用数组存储变量赋值，保证随机访问效率；

使用堆结构实现变量优先级队列，支持高效更新。

**（3）图形库选择：**

选择EasyX图形库，因其简单易用且适合教学用途；

提供基本的图形绘制和事件处理功能；

在Windows平台上有良好的兼容性。

# 3系统详细设计

## 3.1有关数据结构的定义

### 3.1.1 系统数据类型详述

本系统处理的数据主要分为三大类：SAT求解相关数据、数独游戏相关数据和图形界面相关数据。每种数据类型的具体定义如下表3-1所示：

表3-1 **系统数据类型详述表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据类型 | 数据项名称 | 数据项描述 | 数据类型 | 数据项说明 |
| SAT求解数据 |  |  |  |  |
| DPLL\_Literal | order | 文字编号 | int | 正手表示正文字，负数表示逆文字 |
|  | fromClause | 所属子句编号 | int | 标识该文字属于哪个子句 |
|  | nextL | 下一个文字指针 | DPLL\_Literal\* | 链表结构，指向同一字句的下一文字 |
| DPLL\_Clause | order | 子句编号 | int | 子句的唯一标识符 |
|  | size | 子句大小 | int | 子句中包含的文字数量 |
|  | headL | 文字链表头指针 | DPLL\_Literal\* | 指向子句的第一个文字 |
|  | nextC | 下一个子句指针 | DPLL\_Clause\* | 链表结构，指向公式中的下一个子句 |
| DPLL\_Cnf | literals | 变量赋值数组 | int\* | 存储变量赋值状态（0/1） |
|  | lnum | 变量总量 | int | CNF公式中的变量数量 |
|  | cnum | 子句总数 | int | CNF公式中的子句数量 |
|  | headC | 子句链表头指针 | DPLL\_Clause | 指向公式的第一个子句 |
|  | labsnum | 文字出现次数数组 | int\* | 统计每个文字在公式中的出现次数 |
|  | lrelweight | 文字权重数组 | float\* | 存储每一个文字的相对权重值 |
|  | topA | 操作栈顶指针 | DPLL\_Action\* | 指向操作记录栈的顶部 |
| CDCL\_Clause | lbd | LBD值 | int | 学习子句的质量评估值 |
|  | size | 子句大小 | int | 子句中的文字数量 |
|  | lits | 文字数组 | int\* | 动态分配的文字数组 |
| Solver | vars | 变量数量 | int | 求解器处理的变量总数 |
|  | clauses | 子句数量 | int | 求解器处理的子句总数 |
|  | clause\_DB | 子句数据库 | Clause\* | 存储所有子句的数组 |
|  | value | 变量赋值数组 | int\* | 存储每个变量的当前赋值 |
|  | reason | 推导原因数组 | int\* | 记录每个变量赋值的推导子句 |
|  | level | 决策层级数组 | int\* | 记录每个变量的决策层级 |
|  | activity | 变量活动度数组 | double\* | 记录每个变量的VSIDS活动度 |
|  | trail | 赋值轨迹数组 | int\* | 记录变量赋值的顺序 |
|  | learnt | 学习子句数组 | int\* | 存储冲突分析得到的学习子句 |
| 数独游戏数据 |  |  |  |  |
| SudokuBoard | sudoku | 当前数独状态 | int[SIZE][SIZE] | 9\*9的整数矩阵，存储当前数独格局 |
|  | solution | 求解结果 | int[SIZE][SIZE] | 存储SAT求解器返回的数独解 |
|  | initial | 初始格局 | int[SIZE][SIZE] | 存储初始数独，用于区分固定数字 |
| 图形界面数据 |  |  |  |  |
| Button | x | 按钮x坐标 | int | 按钮在界面中的横坐标 |
|  | y | 按钮y坐标 | int | 按钮在界面中的纵坐标 |
|  | width | 按钮宽度 | int | 按钮的像素宽度 |
|  | height | 按钮高度 | int | 按钮的像素高度 |
|  | text | 按钮文本 | const char\* | 按钮上显示的文字 |
| DisplayInfo | board\_offset\_x | 棋盘x偏移 | int | 棋盘左上角的x坐标 |
|  | board\_offset\_y | 棋盘y偏移 | int | 棋盘左上角的y坐标 |
|  | cell\_size | 格子大小 | int | 每个数独格子的像素尺寸 |
|  | button\_count | 按钮数量 | int | 界面中按钮的总数 |

### 3.1.2 数据结构关系

(1)DPLL数据结构关系如图3-1所示

**图形用户界面

AI 生成的内容可能不正确。**

图3-1 DPLL数据结构关系

（2）CDCL数据结构关系如图3-2所示

**图形用户界面

AI 生成的内容可能不正确。**

图3-2 CDCL数据结构关系

（3）数独相关数据结构关系如图3-3所示

图示

AI 生成的内容可能不正确。

图3-3 数独相关数据结构关系

### 3.1.3 数据结构设计原理

（1）链表与数组的权衡：

子句使用链表存储，适应子句数量的动态变化；

文字在子句内使用链表，便于插入和删除操作；

变量赋值使用数组，保证O(1)时间的随机访问。

（2）内存管理策略：

采用动态内存分配，适应不同规模的问题实例；

实现自动扩容机制，避免频繁重新分配；

设计完善的内存释放函数，防止内存泄漏。

## 3.2 主要算法设计

### 3.2.1 DPLL核心算法实现

DPLL算法核心流程如算法3-1所示：

**————————————————————————————————**

算法3-1: DPLL求解算法

输入: CNF公式S

输出: SATISFIABLE或UNSATISFIABLE

function DPLL(S):

    while S中存在单子句 do

        L ← 选择单子句

        S ← 单子句传播(S, L)      // 应用单子句规则

        if S为空 then return SATISFIABLE

        if S包含空子句 then return UNSATISFIABLE

    end while

    if S为空 then return SATISFIABLE

    if S包含空子句 then return UNSATISFIABLE

    v ← 基于权重选择分裂变量        // 变量选择策略

    if DPLL(S ∪ {v}) = SATISFIABLE then

        return SATISFIABLE

    else

        return DPLL(S ∪ {¬v})     // 回溯尝试另一分支

    end if

end function

**————————————————————————————————**

其中，单子句传播的实现如算法3-2所示**：**

**————————————————————————————————**

算法3-2: 单子句传播算法

输入: CNF公式S

输出: SATISFIABLE / UNSATISFIABLE / 需要分裂

int DPLL\_unitClauseDiss(S) {

    while (存在单子句) {

        // 选择单子句L

        literal = 单子句的文字;

        // 删除包含L的子句

        for 每个包含L的子句C:

            DPLL\_removeClause(C, cnf);

        // 从剩余子句中删除¬L

        for 每个包含¬L的子句C:

            DPLL\_removeLiteral(¬L, C, cnf);

        // 检查终止条件

        if 公式为空: return SATISFIABLE;

        if 存在空子句: return UNSATISFIABLE;

    }

    return 需要分裂;

}

**————————————————————————————————**

在变量选择策略方面上，基于文字相对权重进行变量选择，由权重计算公式（3-1）得出，其中sign(x)在x为正文字时为+1，负文字时为-1,size(c)表示文字所在对应子句的大小。

（3-1）

### 3.2.2 CDCL算法优化设计

CDCL算法整体框架如算法3-3所示：

**————————————————————————————————**

算法3-2: CDCL求解算法

输入: CNF公式

输出: 可满足性结果

function CDCL\_Solve():

    while true:

        conflict ← UnitPropagation()    // 单元传播

        if conflict ≠ -1:               // 发现冲突

            if decision\_level = 0: return UNSAT

            learnt\_clause, backtrack\_level ← AnalyzeConflict(conflict)

            Backtrack(backtrack\_level)

            AddLearntClause(learnt\_clause)

            RestartIfNeeded()           // 重启策略

        else if 所有变量已赋值:

            return SAT

        else:

            Decide()                    // 决策新变量

        end if

    end while

end function

**————————————————————————————————**

其中，关键的问题是冲突分析和子句学习，可由下列函数实现：

**————————————————————————————————**

int analyze(Solver\* s, int conflict, int\* backtrackLevel, int\* lbd) {

    // 初始化分析状态

    s->learnt\_size = 0;

    s->time\_stamp++;

    // 寻找第一个UIP（唯一隐含点）

    do {

        Clause\* c = &s->clause\_DB[conflict];

        for 子句c中的每个文字lit:

            if lit的决策层级 > 0 且 未标记:

                标记该文字

                if lit的层级 == 当前层级:

                    待处理文字数++

                else:

                    添加到学习子句

                end if

                提升变量活动度

            end if

        end for

        // 回溯赋值轨迹找到下一个冲突点

        while 轨迹中的变量已标记:

            移动轨迹指针

    } while 待处理文字数 > 0;

    // 计算LBD（文字块距离）

    \*lbd = 计算学习子句的LBD值;

    // 确定回溯层级

    \*backtrackLevel = 计算最大回溯层级;

    return 0;

}

**————————————————————————————————**

VSIDS的选择策略可由下列函数实现：

**————————————————————————————————**

void bump\_var(Solver\* s, int var, double coeff) {

    // 提升变量活动度

    s->activity[var] += s->var\_inc \* coeff;

    // 防止浮点数溢出

    if (s->activity[var] > 1e100) {

        for 所有变量: 活动度缩放;

        s->var\_inc \*= 1e-100;

    }

    // 更新堆结构

    if 变量在堆中: heap\_update(&s->vsids, var);

}

int decide(Solver\* s) {

    // 从堆中选择活动度最高的未赋值变量

    while (!heap\_empty(&s->vsids)) {

        next = heap\_pop(&s->vsids);

        if s->value[next] == 0:

            break;

    }

    // 基于相位保存策略选择赋值

    if s->saved[next] != 0:

        next \*= s->saved[next];  // 使用保存的相位

    // 进行决策赋值

    assign(s, next, 当前层级+1, -1);

    return 0;

}

**————————————————————————————————**

### 3.2.3 数独SAT编码算法

**变量编码方案：**每个数独变量编码为三维索引（3-2）：

（3-2）

其中row, col ∈ [1,9], digit ∈ [1,9]，var为编码，row为行，col为列，digit为填入数字。

对与一个百分号数独格局的约束，包括了单元格约束，行约束，列约束，宫约束，撇对角线约束和窗口约束。

对于单元格约束，需满足两个命题，即对每一个单元格至少有一个数字且最多有一个数字。

约束每个单元格至少有一个数字可以由下列子句（3-3）实现，其中var代表文字编号，i,j表示单元格坐标。

（3-3）

约束每个单元格最多有一个数字，可以使用轮换的互斥子句（3-4），其中，k1,k2为范围内不同的数字。

（3-4）

对于其余约束，可以对其约束范围内的每对单元格，添加如（3-4）的互斥子句即可。

### 3.2.4 图形界面交互设计

绘制图形界面时，利用EasyX三方库，利用网格线绘制棋盘背景，凸显宫，撇对角线和窗口约束，将数字限制在单元格中，同时添加功能按钮，形成交互。

对于事件流程处理，当用户点击对应按钮时，执行相应功能，并给出反馈。当用户点击单元格时，弹出弹窗允许用户输入并检验输入合法性给出反馈。

### 3.2.5 性能优化策略

主要从内存，算法和缓存三个角度对性能进行优化。

**（1）内存优化：**

使用内存池管理频繁分配的小对象；

实现子句数据库的惰性删除机制；

采用增量式的数据结构扩容策略。

**（2）算法优化**：

实现观察器机制，优化单元传播效率；

使用LBD评估学习子句质量，及时清理低质量子句；

设计自适应重启策略，避免陷入局部搜索。

**（3）缓存优化：**

实现子句文字的缓存友好存储布局；

使用位操作加速文字处理；

优化堆操作，减少缓存失效。

### 3.2.6 数据流整合

对于算法，主控和各功能模块，利用核心数据之间的关联性质，有条理地进行整合分析，处理数据流的设计框架如图3-4所示。

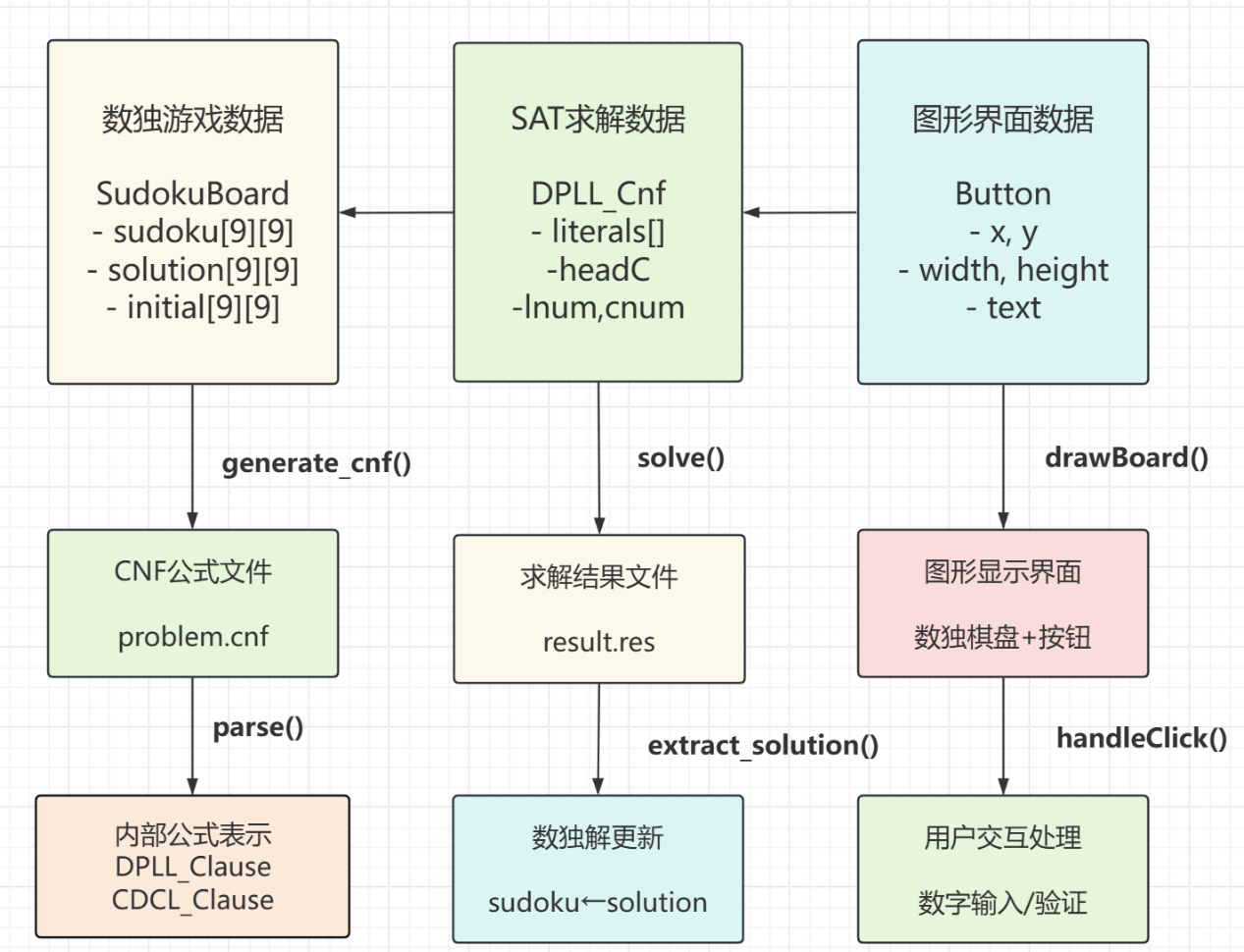


图3-4 数据流处理框架

### 3.2.7 小结

通过以上详细设计，系统具备了高效的数据结构和算法实现方案，为后续的程序实现奠定了坚实的技术基础。

# 4系统实现与测试

## 4.1系统实现

### 4.1.1 开发环境配置

（1）硬件环境

处理器：AMD Ryzen 9 7940HX;

显卡：NVIDIA GeForce RTX 4060；

内存: 16GB + 32GB DDR5内存；

存储：1TB + 2TB NVMe SSD；

显示器: 16英寸 2.5k (2560x1600) 165Hz IPS屏幕；

（2）软件环境

操作系统：Windows 11 家庭中文版 24H2；

开发工具：Visual Studio Code 1.104.2;

图形库：EasyX 2024-6-1;

编译环境：MinGW-w64 GCC 15.2.0 （x86\_64- seh-msvcrt）

调试工具：GDB 16.3

版本控制：Git 2.51.0

### 4.1.2 函数模块说明

**（1）SAT求解器核心模块**

CDCL算法模块（CDCL.h）：

init\_solver()：初始化CDCL求解器数据结构;

parse()：解析CNF文件格式的SAT问题;

solve()：执行CDCL算法进行SAT求解;

propagate()：单元传播过程;

analyze()：冲突分析，生成学习子句;

decide()：变量决策过程;

backtrack()：回溯到指定决策层级;

print\_model()：输出SAT解模型;

DPLL算法模块（DPLL.h）:

DPLL\_createCnf()：创建CNF公式数据结构;

DPLL\_initialCnf()：初始化CNF赋值;

DPLL\_DpllPro()：执行DPLL算法求解;

DPLL\_unitClauseDiss()：单子句传播过程;

DPLL\_assignWithWeight()：基于权重的变量赋值;

DPLL\_backtrack()：回溯操作;

DPLL\_generateResult()：生成求解结果文件;

**（2）数独处理模块（sudukoH.h）**

数独文件操作函数：

read\_sudoku\_from\_file()：从文件读取数独初始格局；

generate\_cnf()：生成包含基本规则和额外约束的数独CNF文件；

generate\_cnf\_with\_extra()：生成带额外排除约束的CNF文件；

数独生成与验证函数：

generate\_sudoku\_puzzle()：生成具有唯一解的数独谜题；

generate\_full\_solution()：生成完整的数独终盘；

is\_unique\_solution()：验证单个位置解的唯一性；

is\_unique\_full\_solution()：验证整个数独解的唯一性；

数独求解与输出函数：

extract\_solution()：从SAT解中提取数独解；

print\_sudoku()：格式化输出数独棋盘；

**（3）图形界面模块（Sudoku.cpp）**

界面绘制函数：

drawBoard()：绘制数独棋盘和特殊约束区域；

drawButton()：绘制功能按钮；

drawTitle()：绘制程序标题；

drawAuthor()：绘制作者信息；

交互处理函数：

isPointInButton()：检测鼠标点击位置；

getCellFromCoord()：将坐标转换为数独单元格索引；

功能实现函数：

clearUserInput()：清空用户输入的数字；

generateSudoku()：从文件调取数独；

generateUniqueSudoku()：生成唯一解数独；

solveSudoku()：求解当前数独；

validateSudoku()：验证数独解答正确性；

### 4.1.3 模块调用关系

**（1）系统模块调用**

主要程序为SatSolver(性能测试程序)和Sudoku(数独图形界面程序)，其中SatSolver调用模块关系见图4-1，Sudoku调用模块见图4-2。

SatSolver可以选择测试文件，对测试文件采取DPLL和CDCL算法模块分别求解并输出结果，测试时间性能并进行比较，如图4-1所示。

图示

AI 生成的内容可能不正确。

图4-1 SatSolver模块调用

Sudoku主要分为图形界面模块和内层数据处理和算法模块，各个模块互相联系交互，如图4-2所示。

图片包含 图形用户界面

AI 生成的内容可能不正确。

图4-2 Sudoku模块调用

**（2）核心功能调用流程**

唯一解数独初始格局生成流程：

**————————————————————————————————**

generateUniqueSudoku()

    → generate\_sudoku\_puzzle()

        → generate\_full\_solution()

            → generate\_cnf() → CDCL求解

        → 挖空验证(is\_unique\_solution)

**————————————————————————————————**

数独求解流程：

**————————————————————————————————**

solveSudoku()

    → generate\_cnf(当前数独状态)

    → init\_solver()

    → parse(CNF文件)

    → solve() [CDCL算法]

    → extract\_solution()

    → 更新界面显示

**————————————————————————————————**

SAT求解流程（CDCL）：

**————————————————————————————————**

solve()

    → propagate() [单元传播]

    → 冲突检测

    │   → analyze() [冲突分析]

    │   → backtrack() [回溯]

    → decide() [变量决策]

    → 循环直至求解完成

**————————————————————————————————**

数独验证流程：

**————————————————————————————————**

validateSudoku()

    → 检查行约束

    → 检查列约束

    → 检查宫约束

    → 检查特殊约束（对角线、窗口）

    → 返回验证结果

**————————————————————————————————**

### 4.1.4 模块间协作机制

**（1）数据流**：图形界面模块接收用户输入，调用数独处理模块生成SAT问题，通过SAT求解器模块求解后，将结果返回图形界面显示。

**（2）控制流**：用户通过图形界面触发功能，系统按预设流程调用相应函数模块，各模块通过参数传递和返回值进行通信。

**（3）文件交互**：数独模块生成CNF文件，SAT求解器读取并求解，结果通过文件或内存传递回数独模块。

**（4）错误处理**：各模块通过返回值传递状态信息，上层模块根据返回值决定后续操作流程。

该系统通过模块化设计实现了功能的清晰分离，各模块职责明确，便于维护和扩展。

## 4.2系统测试

### 4.2.1 常用软件测试方法

**（1）黑盒测试**

功能测试：验证模块功能是否符合需求规格

边界值分析：测试输入参数的边界条件

等价类划分：将输入数据划分为有效和无效等价类

错误推测法：基于经验推测可能存在的错误

**（2）白盒测试**

语句覆盖：确保每条语句至少执行一次

分支覆盖：验证每个条件分支都被执行

路径覆盖：测试所有可能的执行路径

条件覆盖：检查每个逻辑条件的真假值

**（3）性能测试**

时间性能：测试算法执行时间

资源使用：检测内存、CPU等资源消耗

负载测试：验证模块在不同负载下的表现

**（4）集成测试**

自顶向下：从主模块开始逐步集成

自底向上：从底层模块开始测试

回归测试：确保修改不会影响现有功能

### 4.2.2 关键模块测试

**（1）DPLL/CDCL求解模块**

**功能：**利用DPLL算法和CDCL算法求解SAT问题

**设计目标：**能正确解析CNF格式文件，实现高效的SAT求解，输出正确的满足解和无解判断。

**数据选取策略：**覆盖简单和复杂算例，满足和不满足算例，对两大算法的极限处理情况进行测试。

**测试大纲：**选取多个样例文件，其中，算例文件1、2、3、5为简单满足算例，4为简单不满足算例，6、12为复杂满足算例，7、9、11为复杂不满足算例。

**运行结果：**性能测试结果见图4-3。

文本

AI 生成的内容可能不正确。

图4-3 SAT求解器性能测试结果

**结果分析：**由图4-3中性能测试结果可见，DPLL仅能处理简单算例，对于复杂算例如算例6，耗时极高，效率极低。而CDCL能快速处理大部分算例，对比DPLL优化率极高，性能优秀，仅对超复杂算例如算例12处理效率不足，能够满足程序快速响应需求。

**（2）数独生成模块**

**功能：**能够利用初始数独格局文件生成谜题，也能够自动生成唯一解的随机数独谜题。

**设计目标**：生成符合约束条件的有效数独，保证生成的数独具有唯一解，包含特殊约束（对角线、窗口）。

**测试数据选取：**多次生成数独谜题。

**测试大纲：**分别多次调用数独抽取模块和数独生成模块，分析稳定性，人工检查是否符合百分号数独规则约束，并进行唯一解验证。

**运行结果：**调取数独功能结果见下图4-4。

表格, 日历

AI 生成的内容可能不正确。

图4-4 调取数独功能测试

生成数独功能测试结果见图4-5。

表格, 日历

AI 生成的内容可能不正确。表格, 日历

AI 生成的内容可能不正确。

图4-5 生成数独功能测试结果

唯一解验证如图4-6所示。

文本, 表格

AI 生成的内容可能不正确。

图4-6 唯一解验证

**结果分析：**通过多次测试，调取数独和生成数独功能稳定，经过人工验证，满足百分号数独约束规则，对生成数独初始格局进行多次求解验证，证明符合唯一解谜题。

**（3）数独求解和验证模块**

**功能：**求解数独问题并验证解答的正确性。

**设计目标：**将数独问题转换为SAT问题，调用SAT求解器获得解答，验证数独解答满足所有约束，提供详细的验证反馈。

**测试数据策略：**对可解数独——已知有解的标准数独，不可解数独——包含冲突的无效数独，边界情况——完全空数独、已填满数独，进行测试

**测试大纲：**分别进行人工输入答案和调用生成解功能，验证求解和验证解功能是否有效。

**测试结果：**对于合法初始数独格局应用求解功能如图4-7所示。

表格, 日历

AI 生成的内容可能不正确。文本, 表格

AI 生成的内容可能不正确。

图4-7 求解合法数独

对于非法初始格局应用求解功能如图4-8所示。

表格, 日历

AI 生成的内容可能不正确。表格

AI 生成的内容可能不正确。

图4-8 求解非法数独

对于非满数独或错误数独进行解验证结果如图4-9所示。

文本, 表格

AI 生成的内容可能不正确。文本, 表格

AI 生成的内容可能不正确。

图4-9 错误数独解验证

对于合法正确解的数独结果进行验证结果如4-10所示。

文本, 表格

AI 生成的内容可能不正确。

图4-10 正确数独结果解验证

**结果分析：**程序能正确求解各类合法数独格局，并对数独结果进行检验，对各种错误能给出反馈。

**（4）测试总结**

通过上述四个关键模块的系统测试，可以全面验证系统具有下面核心功能性质：

**算法正确性**——SAT求解算法的理论基础实现，能正确求解SAT问题。

**功能完整性**——数独生成、求解、验证的完整流程。

**性能指标**——在不同规模问题上的表现优异明显，能满足数独游戏所需性能。

**健壮性**——对边界情况和异常输入的处理能力足够，能给出清晰反馈。

# 5总结与展望

## 5.1全文总结

本课程设计基于SAT求解技术，成功设计并实现了一个完整的百分号数独游戏求解程序。通过对DPLL算法的深入研究和CDCL优化技术的实践应用，完成了一个功能完善、性能优良的SAT求解系统，并将其成功应用于百分号数独这一具有挑战性的逻辑游戏求解中。主要工作成果如下：

**（1）实现了完整的SAT求解器框架**

设计并实现了基于经典DPLL算法的SAT求解器，包含完整的单子句传播、纯文字消除、分裂策略等核心组件。

在DPLL基础上实现了CDCL优化算法，引入了冲突分析、子句学习、非时序回溯、VSIDS变量选择等先进技术。

开发了高效的CNF文件解析模块，支持标准DIMACS格式的读取和内部数据结构构建。

实现了完善的求解结果输出功能，符合课程设计规范要求。

**2. 构建了百分号数独游戏系统**

设计了百分号数独的完整SAT编码方案，包括基本约束（行、列、宫）和特殊约束（撇对角线、窗口约束）。

实现了数独问题的自动生成功能，支持文件读取和算法生成两种方式。

开发了友好的图形用户界面，提供数独的数字输入、求解、验证等交互功能。

建立了数独问题与SAT求解器之间的无缝连接，实现了自动求解和结果提取。

**3. 进行了系统的性能优化与测试**

对数据结构和算法进行了多方面的优化，显著提升了求解效率。

设计了全面的测试方案，验证了系统的正确性、稳定性和性能表现。

通过对比实验证明了CDCL算法相比基础DPLL的显著性能优势。

## 5.2工作展望

基于本项目的开发经验和技术积累，我对未来的学习和发展方向有了更清晰的规划，主要集中在以下几个方面的深入探索：

### 5.2.1 算法深度优化

在现有DPLL和CDCL算法基础上，我将进一步深入研究现代SAT求解器的核心技术。计划系统学习MiniSat、Glucose、CaDiCaL等主流求解器的源码，理解其高级优化技术，包括冲突分析优化、子句管理策略、重启机制等。通过复现和改进这些算法，提升对SAT求解理论的理解深度，为开发更高效的求解器奠定基础。

### 5.2.2 工程级求解器开发

基于当前的项目经验，我计划着手开发一个更加完善的SAT求解器框架。重点研究现代C++在算法实现中的应用，学习模板元编程、智能指针等高级特性，提升代码的性能和可维护性。同时，将探索如何将求解器模块化，提供清晰的API接口，使其能够方便地集成到其他应用中。

### 5.2.3 图形化项目拓展

本次项目中积累的图形界面开发经验具有很强的迁移价值。我计划将这些经验运用到更多类型的程序开发中，比如开发算法可视化工具、数据分析和展示系统、教育类软件等。特别是基于EasyX或转向Qt、ImGui等更专业的GUI框架，开发具有实用价值的桌面应用程序。

### 5.2.4 完整项目开发实践

在掌握核心算法和图形开发能力的基础上，我计划开展更具挑战性的完整项目开发。这可能包括开发一个集成多种算法的算法实验平台，或者结合SAT求解技术解决实际的工程问题。通过这些项目，系统锻炼需求分析、架构设计、编码实现、测试维护的全流程开发能力。

### 5.2.5 技术栈扩展

为了适应更复杂的开发需求，我将逐步扩展技术栈。在图形开发方面，从EasyX向跨平台的图形库过渡；在算法实现方面，学习并行计算和GPU加速技术；在工程实践方面，掌握版本控制、持续集成、单元测试等现代软件开发方法。

通过这些有计划的学习和实践，我希望能够从课程项目的层次提升到能够独立开发复杂软件系统的水平，为未来的技术生涯打下坚实基础。本项目的完成只是一个起点，后续的深入探索将帮助我在程序设计和算法实现的道路上走得更远。

# 6 体会

通过这次基于SAT的百分号数独求解程序的开发，我深刻体会到了从算法理论到工程实践的完整过程，其中既有挑战也有收获。

在算法学习阶段，DPLL和CDCL算法看似清晰的理论背后隐藏着许多实现细节的挑战。最初阅读论文时，我以为理解了算法的核心思想，但在实际编码过程中才发现，单子句传播的高效实现、冲突分析的正确性保证、学习子句的质量评估等问题都需要仔细斟酌。特别是在实现CDCL算法时，冲突分析和非时序回溯机制的理解花费了我大量时间，需要通过反复调试和查阅资料才能真正掌握其精髓。

在程序开发过程中，我认识到一个完整的系统需要兼顾多个方面。从最开始的SAT求解核心，到数独问题的SAT编码，再到图形界面的实现，每个环节都有其独特的技术要点。图形界面开发让我学会了如何设计用户友好的交互流程，而算法优化则锻炼了我的性能分析能力。特别是在处理大规模数独问题时，如何平衡求解时间和内存使用成为了一个需要不断调整的课题。

这次项目也让我体验到了算法优化的乐趣。从最初的基础DPLL实现，到加入CDCL的各种优化策略，每一次改进都能带来可见的性能提升。当我看到优化后的求解器能够快速解决那些原本需要很长时间的问题时，获得的成就感是单纯理论学习无法比拟的。这种将抽象算法转化为实际可运行代码的过程，让我对计算机科学的魅力有了更深的理解。

通过这个项目，我不仅掌握了SAT求解的相关算法，更重要的是学会了如何系统地解决复杂的技术问题。从需求分析到模块设计，从代码实现到测试优化，每个阶段都需要不同的思维方式和技术能力。这种全流程的开发经验对我未来的学习和工作都具有重要价值。

这次经历让我明白，优秀的程序员不仅需要掌握编程技巧，更需要具备将抽象问题转化为具体解决方案的能力。在未来的学习中，我将继续深入探索算法与系统开发的结合，努力提升自己的工程实践能力。

# 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