**计算概论小课题结题报告**

多种类数独计算器

**仝鑫**

**刘茁**

**吴天海**

**Jan.14, 2016**

目录

**1 概述** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**2 数独简介及种类**

2.1 数独简介 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2 数独种类

2.2.1 普通数独 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2.2 对角线数独 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2.3 窗口数独. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2.2.4 杀手数独. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**3 数独算法**

3.1 优化深度优先搜索算法 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.1.1 相关头文件 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.1.2 SudokuData类及相关函数

3.1.2.1 类的声明 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.1.2.2 DFS实现 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2 Dancing Links算法

3.2.1 相关头文件 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2.2 KillerSudokuData类及相关函数 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3.2.3 DancingLinks类及相关函数 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**4 可视化实现**

4.1 选择Qt原因 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2 相关类及相关函数

4.2.1 Dialog类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.2 Cell类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.3 SudokuTable类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.4 MainWindow类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.5 SettingsDialog类 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

4.2.6 QFile类实现文件读取 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**5 总结**

5.1 主要优点 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5.2 主要不足 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5.3 现实意义及未来展望

5.3.1 现实意义 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5.3.2 未来展望 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5.4 课题分工及感想

5.4.1 仝鑫 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5.4.2 刘茁 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

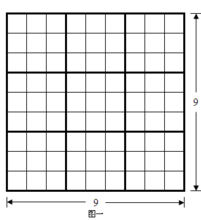
5.4.3 吴天海 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

**1 概述**

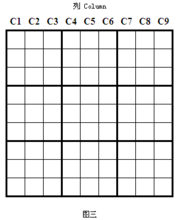
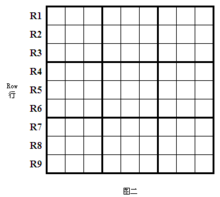
数独是一种风靡全球的数字游戏，其种类多样，难度梯度大，解法也较多，而且其解决方法往往是许多其他问题解法的基础或衍生。与此同时，网上也有着许多比较流行的解题软件（如数独计算器、数独助手等），其宗旨便是仿照人脑思考模式解决数独问题。而本课题组旨在通过对各种数独不同算法的探究，寻找到相对较为快速高效的方法，结合Qt实现可视化，制作成多种类数独计算器，并加入一些功能，使得其能解决四种基本类型的数独，并将所用算法进行拓展，寻觅解决其他数独方法的思路，并探索这些算法其他更加广泛的应用领域。

**2 数独简介及种类**

**2.1 数独简介**

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/c8177f3e6709c93dc7663c559f3df8dcd00054ff?fr=lemma&ct=single)数独是一种逻辑游戏。玩家需要根据9×9盘面上的已知数字，推理出所有剩余空格的数字，并满足**每一行、每一列、每一个粗线宫内的数字均含1-9，不重复**。

图一 九宫格

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/e61190ef76c6a7ef4bdb9a84fdfaaf51f2de6696?fr=lemma&ct=single)[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/d4628535e5dde71112757154a7efce1b9c1661e8?fr=lemma&ct=single)数独表格的水平方向有九横行，垂直方向有九纵列的矩形，画分八十一个小[正方形](http://baike.baidu.com/view/56164.htm" \t "_blank)，称为**[九宫格](http://baike.baidu.com/subview/230179/5038177.htm" \t "_blank)**（Grid），如图一所示，是数独的作用范围。

图二 行

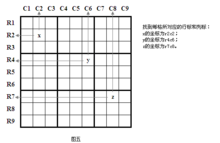
数独表格水平方向的每一横行有九格，每一横行称为**行**（Row），如图二所示。

数独表格垂直方向的每一纵列有九格，每一纵列称为**列**（Column），如图三所示。

图三 列

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/f9198618367adab44d2e29e18bd4b31c8601e491?fr=lemma&ct=single) 数独表格三行与三列相交之处有九格，每一单元称为**小九宫**（Box、Block），简称**[宫](http://baike.baidu.com/subview/3385/5264919.htm" \t "_blank)**，如图四所示（在杀手数独中，宫往往用单词Nonet表示）。

图四 小九宫

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/377adab44aed2e73820cb48e8701a18b86d6faae?fr=lemma&ct=single) 上述行、列、宫统称为**单元**（Unit）。

数独表格的格位按所处的行列单元赋予**坐标值**，如图五所示。

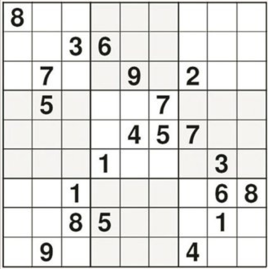
[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/9825bc315c6034a8cb048a23cb134954082376a9?fr=lemma&ct=single)坐标有多种**标示法**，最为广泛使用的是横行R1~R9，纵列C1~C9的标示法。

图五 坐标值

在九宫格的格位填上一些数字，作为填数判断的[线索](http://baike.baidu.com/subview/932625/16700467.htm" \t "_blank)，称为**提示数**（Clue），如图六所示。

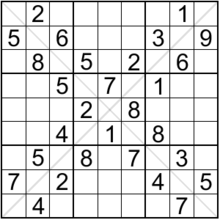
图六 提示数

**2.2 数独种类**

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/0dd7912397dda144d35ab299b1b7d0a20cf4864e?fr=lemma&ct=single)**2.2.1 普通数独**

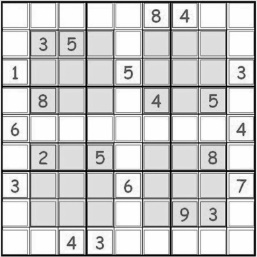
要求每一行、每一列、每一个粗线宫内的数字均含1-9，不重复。

图七 普通数独

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/622762d0f703918f86b10068513d269759eec42b?fr=lemma&ct=single)**2.2.2 对角线数独**

在普通数独规则基础上，两条大对角线的数字不重复。

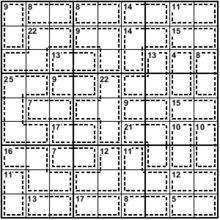
图八 对角线数独

**2.2.3 窗口数独**

在普通数独规则基础上，四个阴影区域内的数字无重复。

图九 窗口数独

**2.2.4 杀手数独**

[](http://baike.baidu.com/pic/%E6%95%B0%E7%8B%AC/74847/0/bba1cd11728b47100662c06ec3cec3fdfd0323c6?fr=lemma&ct=single)在普通数独规则的基础上，每个虚线框左上角的数字表示虚线框内所有数字之和，每个虚线框内数字无重复。

图十 杀手数独

**3 数独算法**

**3.1 优化深度优先搜索算法**

**3.1.1 相关头文件**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <ctime>

#include <algorithm>

#include “stdlib.h”

**3.1.2 SudokuData类及相关函数**

**3.1.2.1 类的声明**

class SudokuData{

public:

SudokuData();

void clear();

void erase();

void getInputData(int(\*data)[9]) const;

void setInputDataNum(int row, int column, int num);

void setUserAnswerNum(int row, int column, int num);

bool checkUserAnswer();

bool hasValidData() const;

void getInvalidDataRecord(int(\*data)[9]) const;

void getPreviousAnswer(int(\*data)[9]);

void getNextAnswer(int(\*data)[9]);

void getNewPuzzle(int(\*data)[9]);

bool hasNext() const;

bool hasPrevious() const;

int getSolutions() const;

void setSolutions(int num);

private:

void initial();

int getAnswerNum(int max, int time\_limit = 100000000);

bool dfsNum(int order, int &ans, int max, time\_t beg, int time\_limit);

bool dfsPrevious(int order);

bool dfsNext(int order);

bool isEnd() const;

bool isValidNum(int(\*data)[9], int row, int column, int num) const;

bool canContinue() const;

void tryNum(int row, int column, int num);

void checkDataValid(int(\*data)[9]);

bool findOrederPos(int &row, int &column, int order) const;

bool findBestPos(int &row, int &column) const;

void fillAnswerNum(int(\*data)[9], int &row, int &column, int num, int order);

void unfillAnswerNum(int(\*data)[9], int row, int column);

int inputData[9][9];

int invalidDataRecord[9][9];

bool dataIsValid;

int userAnswerData[9][9];

int previousAnswerData[9][9];

int currentAnswerData[9][9];

int nextAnswerData[9][9];

int solutions;

int currentAnswerOrder;

bool dfsInitialed;

bool hasPreviousAnswer;

bool hasNextAnswer;

int filledBoxs;

int fillOrder[9][9];

int numCanTry[82][9][9][10];

};

**3.1.2.2 DFS实现**

bool SudokuData::dfsNum(int order, int &ans, int max, time\_t beg, int time\_limit){

clock\_t now = clock();

if (now - beg > time\_limit)

return true;

if (!canContinue())

return false;

int row, column;

if (!findBestPos(row, column)){

if (isEnd()){

ans++;

if (ans == max)

return true;

else

return false;

}

else

QMessageBox::critical(0, QString("error"), QString("error in SudokuData::dfsNum"));

}

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

if (!numCanTry[filledBoxs][row][column][i]) continue;

fillAnswerNum(currentAnswerData, row, column, i, order);

if (dfsNum(order + 1, ans, max, beg, time\_limit)) return true;

unfillAnswerNum(currentAnswerData, row, column);

}

return false;

}

bool SudokuData::dfsPrevious(int order){

if (!canContinue())

return false;

int row, column;

bool hasBegin = false;

if (!dfsInitialed){

hasBegin = true;

if (findOrederPos(row, column, order)){

if (dfsPrevious(order + 1))

return true;

}

else{

dfsInitialed = true;

return false;

}

}

else if (!findBestPos(row, column)){

if (isEnd())

return true;

else

QMessageBox::critical(0, QString("error"), QString("error in SudokuData::dfsPrevious"));

}

for (int i = 9; i >= 1; i--){

if (hasBegin){

hasBegin = false;

i = previousAnswerData[row][column];

unfillAnswerNum(previousAnswerData, row, column);

continue;

}

if (!numCanTry[filledBoxs][row][column][i]) continue;

fillAnswerNum(previousAnswerData, row, column, i, order);

if (dfsPrevious(order + 1)) return true;

unfillAnswerNum(previousAnswerData, row, column);

}

return false;

}

bool SudokuData::dfsNext(int order){

if (!canContinue())

return false;

int row, column;

bool hasBegin = false;

if (currentAnswerOrder != 0 && !dfsInitialed){

hasBegin = true;

if (findOrederPos(row, column, order)){

if (dfsNext(order + 1))

return true;

}

else{

dfsInitialed = true;

return false;

}

}

else if (!findBestPos(row, column)){

if (isEnd())

return true;

else

QMessageBox::critical(0, QString("error"), QString("error in SudokuData::dfsNext"));

}

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

if (hasBegin){

hasBegin = false;

i = nextAnswerData[row][column];

unfillAnswerNum(nextAnswerData, row, column);

continue;

}

if (!numCanTry[filledBoxs][row][column][i]) continue;

fillAnswerNum(nextAnswerData, row, column, i, order);

if (dfsNext(order + 1)) return true;

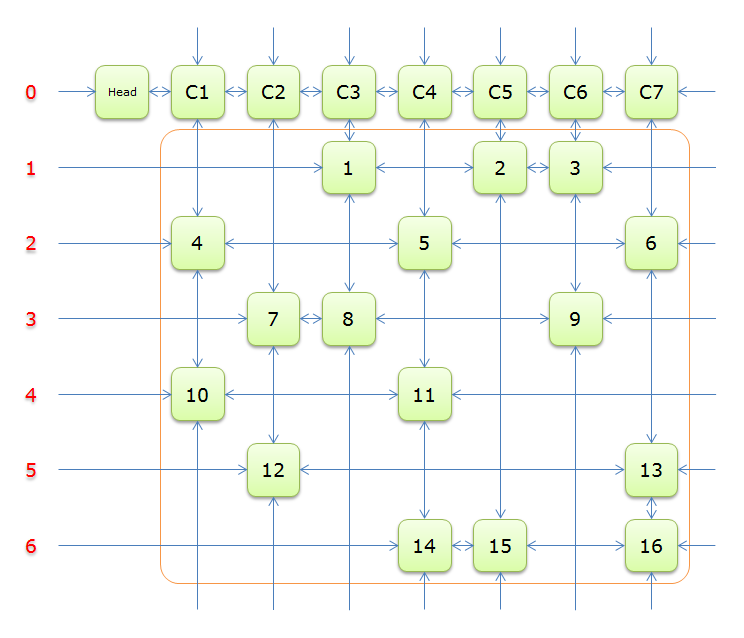
unfillAnswerNum(nextAnswerData, row, column);

}

return false;

}

**3.2 Dancing Links算法**

[](http://images.cnitblog.com/blog/93227/201306/29144604-86559ddc694840869aebb129c2874afb.png)算法大师Donald E.Knuth提出了DLX（Dancing Links X）算法。实际上，他把上面求解的过程称为X算法，而他提出的舞蹈链（Dancing Links）实际上并不是一种算法，而是一种数据结构。一种非常巧妙的数据结构，他的数据结构在缓存和回溯的过程中效率惊人，不需要额外的空间，以及近乎线性的时间。Dancing Links的核心是基于双向十字交叉链表的方便操作（移除、恢复加入）。

图十一 双向循环十字交叉链表

利用舞蹈链（Dancing Links）算法求解数独问题，实际上就是下面一个流程：

1.把数独问题转换为精确覆盖问题

2.设计出数据矩阵

3.用舞蹈链算法求解该精确覆盖问题

4.把该精确覆盖问题的解转换为数独的解

**3.2.1 相关头文件**

#include <iostream>

#include <limits.h>

#include <vector>

#include <math.h>

#include <fstream>

#include <algorithm>

#include <time.h>

**3.2.2 KillerSudokuData类及相关函数**

class KillerSudokuData{

public:

void gen\_1(vector<vector<int> > &ans, vector<int> &temp, int n, int s, int current = 0);

void gen\_2(vector<vector<int> > &check,int s,int i);

void gen\_pair(int blocks, int sum, vector<vector<int> > &check);

bool StrtoMatrix(vector<vector<int> > &matrix, int &total\_blocks, int &total\_sum, istream &infile, vector<int> &check);

vector<int> MatrixtoSudoku(vector<vector<int> > &matrix, vector<int> result)

private:

int total\_blocks;

int total\_sum;

vector<int> check;

vector<vector<int> > matrix;

};

**3.2.3 DancingLinks类及相关函数**

struct Node{

Node \*up, \*down, \*left, \*right, \*colRoot, \*rowRoot; //上下左右四个指针以及指向行列对象的指针

int Num; //行对象特有,记录行数

int Size; //列对象特有,记录该列元素

Node(int i = -1 ): Num(i),Size(0) {}

};

class DLX{

public:

DLX(vector<vector<int> > &matrix);

~DLX();

void init();

void link(vector<vector<int> > &matrix);

void cover(Node \*cRoot);

void recover(Node \*cRoot);

bool Search(int k = 0);

vector<int> getResult() const { return result;}

int getUpdates() const { return \_updates;}

private:

Node \*Head;

vector<int> result; //结果存放在这里

int \_row, \_col, \_updates; //记录行列数,更新次数

};

**4 可视化实现**

**4.1 选择Qt原因**

本课题重点放在对算法的理解和应用，而对可视化图形绘制的要求并不高，因此我们选择了Qt Creater进行可视化的制作。而选择Qt也自有其优势：

第一，便于操作。Qt Creator有两大关键益处：提供首个专为支持跨平台开发而设计的IDE，并确保首次接触Qt框架的开发人员能迅速上手和操作。因时间有限，无法花太多时间学习可视化相关内容，所以我们看中了Qt这一易上手的特性。加之CSDN等论坛对于Qt编程的教学内容相当丰富，因此知识来源非常广泛。而事实证明，它确实是一个便捷且人性化的IDE，在进行接口连接的时候相当自然且方便。

第二，面向对象。Qt的良好封装机制使得Qt的模块化程度非常高，可重用性较好，对于用户开发来说是非常方便的。Qt提供了一种称为signals/slots（信号-槽机制）的安全类型，对于数独计算器这种与这使得各个元件之间的协同工作变得十分简单。

第三，具有丰富的API。Qt包括多达250个以上的 C++类，还提供基于模板的collections，serialization，file，I/O device，directory management，date/time 类，甚至还包括正则表达式的处理功能。不仅如此，Qt还同时支持2D/3D图形渲染，支持OpenGL和XML，这让我们看到了其丰富的实用价值，本着通过课题学习知识的原则，我们最终选择了Qt。

**4.2 相关类及相关函数**

**4.2.1 Dialog类**

该类是各数独计算器界面的构造类。我们将该类选择为QMainWindow的子类，目的是让其拥有更多的功能和更多的API，但与此同时也导致了打开以后后台运行占用内存的情况。

class Dialog : public QMainWindow{

Q\_OBJECT

public:

Dialog(QWidget \* parent = 0);

void createToolBar();

void createActions();

public slots:

void setPreviousActionEnabled(bool flag);

void setNextActionEnabled(bool flag);

void setEyeMode();

void settings();

void help();

void about();

private:

SettingsDialog \* settingsdialog;

SudokuTable \* sudokuTable;

QToolBar \* toolBar;

QAction \* newPuzzleAction;

QAction \* previousAction;

QAction \* nextAciton;

QAction \* eraseAction;

QAction \* clearAction;

QAction \* eyeModeAction;

QAction \* settingsAction;

QAction \* helpAction;

QAction \* aboutAction;

};

**4.2.2 Cell类**

该类是QTableWidgetItem的子类，是单元格绘制类。

class Cell : public QTableWidgetItem{

public:

enum BackgroundColor { WHITE, GREY, EYECOLOR, WINDOWCOLOR };

enum TextColor { DEFAULT, RIGHT, WRONG };

Cell();//构造函数

Cell \* *clone*() const;//常复制构造函数

QVariant *data*(int role) const;

void *setData*(int role, const QVariant &value);//value的值赋给role

bool editEnabled() const;

void setEditEnabled(bool flag);

bool dirty() const;

void setDirty(bool flag);

void setTextColor(TextColor color);

void setBackgroundColor(BackgroundColor color);

private:

bool dataIsDirty;

};

**4.2.3 SudokuTable类**

该类是QTableWidget的子类，是将SudokuData里的计算函数、Cell里的单元格构造函数与Dialog相结合的类。

class SudokuTable :public QTableWidget{

Q\_OBJECT

public:

SudokuTable(QWidget \*parent /\* = 0 \*/);

public slots:

void newPuzzle();

void previous();

void next();

void erase();

void clear();

int dataSolutions() const;

void setDataSolutions(int num);

bool eyeMode() const;

void setEyeMode(bool flag);

bool godMode() const;

void setGodMode(bool flag);

void drawBasicBackgroud();

void drawEyeModeBackground(int row, int column);

void somethingChanged(int row, int column);

signals:

void setPreviousEnabled(bool flag);

void setNextEnabled(bool flag);

private:

Cell \* cell(int row, int column);//参数表示行列坐标

SudokuData \*sudokuData;

bool isGodMode;

bool isEyeMode;

};

**4.2.4 MainWindow类**

该类是主界面构造类，是真正的MainWindow。我们通过该类实现各Dialog之间的转换。

namespace Ui {

class mainwindow;

}

class mainwindow : public QMainWindow{

Q\_OBJECT

public:

explicit mainwindow(QMainWindow \*parent = 0);

~*mainwindow*();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

private:

Ui::mainwindow \*ui;

Dialog1 dialog1;

Dialog2 dialog2;

Dialog3 dialog3;

Dialog4 dialog4;

};

**4.2.5 SettingsDialog类**

该类是QDialog的子类，适用于普通数独、对角线数独以及窗口数独，目的是实现解个数的控制。

class SettingsDialog :public QDialog{

Q\_OBJECT

public:

SettingsDialog(QWidget \* parent /\* = 0 \*/);

int solutions() const { return comboBox->currentIndex() + 1; }

void setSolutions(int num) { comboBox->setCurrentIndex(num - 1); }

private:

QLabel \* label;

QComboBox \* comboBox;

QPushButton \* okButton;

QPushButton \* cancelButton;

};

**4.2.6 QFile类实现文件读取**

该类主要用于杀手数独，实现题目的读入。因为杀手数独类型的特殊性，其题目的构造有一定难度，而且题目的输入也相当复杂。我们考虑选用文本文档输入的形式来读入数据，格式如下：

∑

**1≤i,j≤9**

ij **=** sum

对应代码实现如下：

void SudokuTable4::StrtoMatrix(QVector<QVector<int> > &matrix, int &total\_blocks, int &total\_sum, QString fileName, QVector<int> &check){

QFile infile(fileName);

if(infile.*open*(QIODevice::ReadOnly)){

QTextStream in(&infile);

while(!in.atEnd()){

QString str1 = in.readLine();

std::string str = str1.toStdString();

QVector<QVector<int> > pairs;

int mul = 1;

int sum = 0;

int blocks;

int ix = 0;

if (total\_blocks == 81 && total\_sum == 405)

break;

if (total\_blocks >= 81 || total\_sum >= 405)

return;

while(1){

++ix;

if (str[str.size()-ix] == '=')

{

blocks = (str.size()-ix+1)/3;

break;

}

sum += (str[str.size()-ix] - '0') \* pow(10,ix - 1);

}

if (sum < blocks\*(1+blocks)/2 || sum > blocks\*(19-blocks)/2){

QMessageBox::critical(NULL, QString("错误"), QString("和数超出范围！"));

return;

}

for(int i = str.size() -1 ; i >= blocks \* 3; i--){

add += (str[i] - '0') \* mul;

mul \*= 10;

}

……

QVector<int> row,col;

for (int jx = 0; jx < blocks; ++jx)

{

QVector<int>::iterator it = std::find(check.begin(), check.end(), (str[jx\*3] - '0' ) \* 9 + (str[jx\*3+1] - '0'));

if (it != check.end()){

QMessageBox::critical(NULL, QString("错误"), QString("重复输入！"));

return;

}

row.push\_back(str[jx \* 3] - '0');

col.push\_back(str[jx \* 3 + 1] - '0');

check.push\_back((str[jx \* 3] - '0') \* 9 + (str[jx\*3 + 1] - '0'));

}

for (int jx = 0; jx < pairs.size(); ++jx){

for (int kx = 0; kx < blocks; ++kx){

int \_row = row[kx]-1;

int \_col = col[kx]-1;

int val = pairs[jx][1] -1;

QVector<int> current\_row(729,0);

current\_row[9\*\_row+\_col] = 1;

current\_row[81+9\*\_row+val] = 1;

current\_row[162+9\*\_col+val] = 1;

current\_row[243+9\*(\_row/3\*3+\_col/3)+val] = 1;

for (int lx = 0; lx <= val; ++lx)

current\_row[324+total\_sum+pairs[jx][0]+lx] = 1;

matrix.push\_back(current\_row);

}

}

total\_blocks += blocks;

total\_sum += sum;

}

}

}

**5 总结**

**5.1 主要优点**

**·**DancingLinks以及优化DFS算法具有优越性及普适性，适用于许多问题的求解

**·**采用DancingLinks和双向十字交叉链表解决杀手数独问题，效率高

**·**采用优化DFS算法解决普通数独问题，效率较高，且具有一般性，稍加修改后可适用于所有数独

**·**实现三种数独的随机生成及条件判定

**·**适用于多解问题，借助于算法的优化，可快速解出所有解

**·**UI界面简洁且较为美观

**·**代码编写可读性较好，且具备极大扩充及修改可能

**·**极大锻炼了建模能力，建立了一个较为基础的理论模型

**5.2 主要不足**

**·**窗口数独中出题时间不稳定，解的个数增加后部分题目出题时间过长

**·**杀手数独随机出题功能尚不具备，且其UI界面直观程度尚有欠缺

**·**因时间精力有限，种类较少，部分计划中的其他种类数独有待解决

**·**界面之间切换后仍保持后台运行，导致占用内存较大

**·**因为多人合作而写，故风格各异，编程代码格式略显紊乱

**·**功能较少，更多功能尚待开发

**5.3 现实意义及未来展望**

**5.3.1现实意义：**

**·**数独计算器加以完善优化后可直接应用，对解决常见数独问题较为有力

**·**算法完善后原则上可以解决大多数较为典型的数独问题

**·**杀手数独所用DancingLinks算法是精确覆盖问题的一个特化，精确覆盖问题还可用于以下领域：

**·**解决八皇后问题及其衍生问题

**·**从精确问题入手探讨NP-完全问题，可以此为切入点继续研究模拟退火算法、遗传算法以及神经网络算法等，从而解决更多问题

**·**更广阔领域：以此为切入点进一步深化对精确覆盖问题的讨论以及完善，将适用于求解精确覆盖问题的O（1.414^n）链数DNA计算机算法等问题

**·**优化DFS算法可应用于各种回溯问题的优化求解

**5.3.2 未来展望**

**·**将完善杀手数独算法，使其具备随机出题及求解所有杀手数独能力

**·**将完善其他功能，并解决更多种类数独问题

**·**将进一步对DancingLinks以及DFS、BFS等算法进行探讨，寻找其更加普遍的应用

**·**分析其时间复杂度，绘制相关折线图并进行优化

**·**尽量将该程序与OpenGL相结合，将UI界面美观化，并进一步绘制相关图形

**5.4 课题分工及感想**

**5.4.1 仝鑫**

因为竞赛经验的缺乏，也因为个人能力的有限，在物院的这第一学期过得相当充实且有压力。与此同时，也注定了放在各科目上的时间都会有所均衡和折扣。但是我们还是坚持完成了课题。也许这个课题不是那么的完美，也许有不少的漏洞，但我们通过这段经历获得了许多：对编程的深刻认识，对c++语言的进一步掌握，对建立模型的深刻认知，以及对自主命题、自主完成课题能力的极大锻炼。回想每一次Debug时报错的近乎狂躁的失落以及无意间柳暗花明时的欣喜若狂，我深刻地感受到了这个课题带给我的成就感以及我与同学们之间存在的差距。谢谢老师，谢谢在电脑前码代码的时光。

**5.4.2 刘茁**

**5.4.3 吴天海**