A*算法求解 8 数码问题

摘 要

八数码是人工智能领域的经典问题。该课程设计使用 A*算法求解八数码问题，对比分析了三种不同的 h(n)启发函数对搜索算法的影响。程序基于 C++语言，通过 Qt 实现界面的可视化，能够根据始末状态展示搜索的过程、搜索树及其他与算法性能有关的信息。

关键词：八数码，A*算法，启发函数比较

装

订

线

A* algorithm for Solving 8 puzzle problem

ABSTRACT

8 puzzle is a classic problem in the field of artificial intelligence. This assignment solves eight puzzle problem by using A* algorithm. It compares and analyze three different heuristic function h(n) in solving this issue. The program is based on C++, using Qt to realize visualization interface. It can also show search process, search tree and other information about the feature of algorithm according to the start and end state.

Key words： eight puzzle，A* algorithm，comparison of heuristic function

装

订

线

目 录

1 引 言 1
1.1 八数码 1
1.2 A*算法 1
2 实验概述 2
2.1 实验目的 2
2.2 实验内容 2
2.3 本文所做的工作 2
3 实验方案设计 3
3.1 总体设计思路与总体架构 3
3.1.1 内核部分思路与框架 3
3.1.2 UI 部分思路与框架 3
3.2 核心算法及基本原理 6
3.3 模块设计 7
3.3.1 内核模块设计 7
3.3.2 UI 模块设计 8
3.3.3 内核部分与 UI 部分的联系 10
3.4 创新内容或优化算法 10
4 实验过程 11
4.1 环境说明 11
4.2 源代码文件清单及主要函数清单 11
4.2.1 头文件 11
4.2.2 源文件 11
4.2.3 表单文件 13
4.3 实验结果展示 13
4.3.1 UI 界面展示 13
4.3.2 结果分析 14
5 总结 17
5.1 实验中存在的问题及解决方法 17
5.2 后续改进方向 17
5.3 心得体会及自评 17
参考文献 18

装 订 线

1 引言

1.1 八数码

八数码问题也称为九宫问题。在 3×3 的棋盘，摆有八个棋子，每个棋子上标有 1 至 8 的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

1.2 A*算法

A*搜索算法（A* search algorithm）是一种在图形平面上，有多个节点的路径，求出最低通过成本的算法。常用于游戏中的 NPC 的移动计算，或网络游戏的 BOT 的移动计算上。

该算法综合了最良优先搜索和 Dijkstra 算法的优点：在进行启发式搜索提高算法效率的同时，可以保证找到一条最优路径（基于启发函数）。

在此算法中，如果以 g(n)表示从起点到任意顶点 n 的实际距离， h(n)表示任意顶点 n 到目标顶点的估算距离（根据所采用的评估函数的不同而变化），那么 A*算法的估算函数为：

f(n)=g(n)+h(n)

这个公式遵循以下特性：

如果 g(n)为 0，即只计算任意顶点 n 到目标的启发函数 h(n)，而不计算起点到顶点 n 的距离，则算法转化为使用贪心策略的最良优先搜索，速度最快，但可能得不出最优解；

如果 h(n)不大于顶点 n 到目标顶点的实际距离，则一定可以求出最优解，而且 h(n)越小，需要计算的节点越多，算法效率越低，此种情况也是该实验考虑的情况。

如果 h(n)为 0，即只需求出起点到任意顶点 n 的最短路径 g(n)，而不计算任何启发函数 h(n)，则转化为单源最短路径问题，即 Dijkstra 算法，此时需要计算最多的顶点

2 实验概述

2.1 实验目的

熟悉和掌握启发式搜索策略的定义、评价函数 f(n)和算法过程，并利用 A*算法求解 8 数码问题，理解求解流程和搜索顺序。

2.2 实验内容

以 8 数码问题为例，实现 A*算法的求解程序（编程语言不限），要求设计两种不同的启发函数 h(n)。

设置相同初始状态和目标状态，针对不同的评价函数求得问题的解，比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数和运行时间等。要求画出结果比较的图表，并进行性能分析。

要求界面显示初始状态，目标状态和中间搜索步骤。

要求显示搜索过程，画出搜索过程生成的搜索树，并在每个节点显示对应节点的评价值 f(n)。以红色标注出最终结果所选用的路线。

撰写实验报告，提交源代码（进行注释）、实验报告、汇报 PPT。

2.3 本文所做的工作

本实验使用 A*算法求解八数码问题，实现了从初始状态到目标状态最短路径的查找。代码使用了 C++语言编写并利用 STL 库简化了代码编写的难度。实验从扩展节点数、生成节点数和运行时间三个角度分析比较了这三种启发函数，进而加深对 A*算法的理解。

同时，使用了 Qt 这一跨平台 C++图形用户界面应用程序来实现 UI 界面。用户可以在菜单栏中获得有关八数码和该程序使用的相关知识，还可以获得实验的结论信息。用户可以利用鼠标和键盘输入八数码的起始和终止状态、选择三种启发函数之一。程序能根据用户输入判断是否符合要求、是否能得出结果，并在结果基础上展示运行步骤、运行时间、扩展节点数、生成节点数和搜索树。由于界面大小限制，搜索树可能会显示不全，用户可以通过点击、拖动来查看搜索树。

3 实验方案设计

3.1 总体设计思路与总体架构

将总体设计分为 UI 设计和内核设计两个部分，UI 部分负责与用户之间的交互和图形化的展示，内核部分主要进行搜索求解的过程。

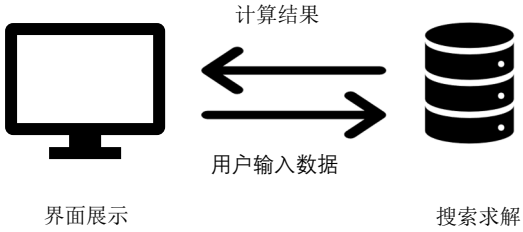

图 3.1 总体设计组成图

3.1.1 内核部分思路与框架

内核部分主要是对输入的判断和对合理输入的求解。

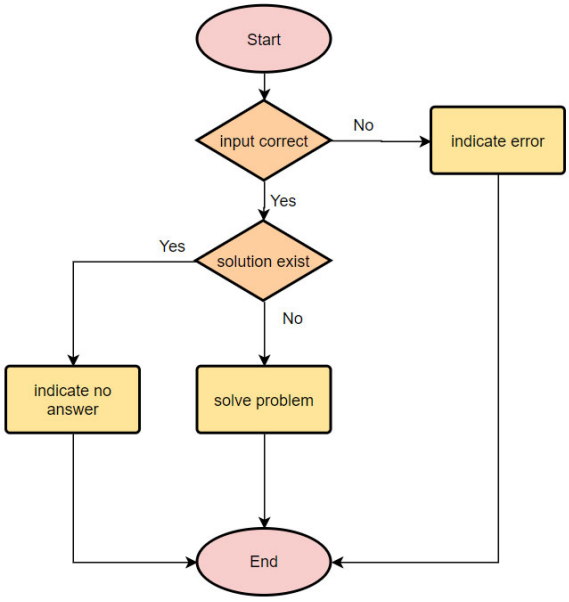

图 3.1.1 内核部分思路流程图

3.1.2 UI 部分思路与框架

UI 界面按功能分为工具栏、输入部分、展示部分，共三大块。初始界面由于没有或未有合法输入,只展示工具栏和输入部分。可以通过键盘在输入部分输入开始与结束状态，点击选择 h(n)函数。输出部分可展示移动的过程和与求解相关的信息。点击 more 可查看搜索树。绘制搜索树时展示了搜索树的每个结点和对应的 f(n)的值。

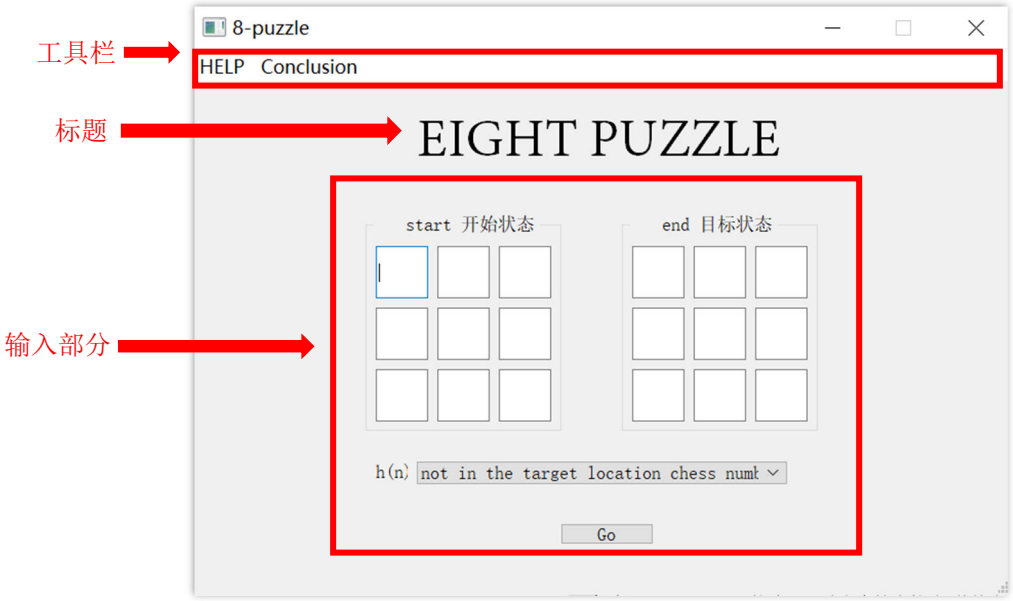

图 3.1.2.1 初始界面展示

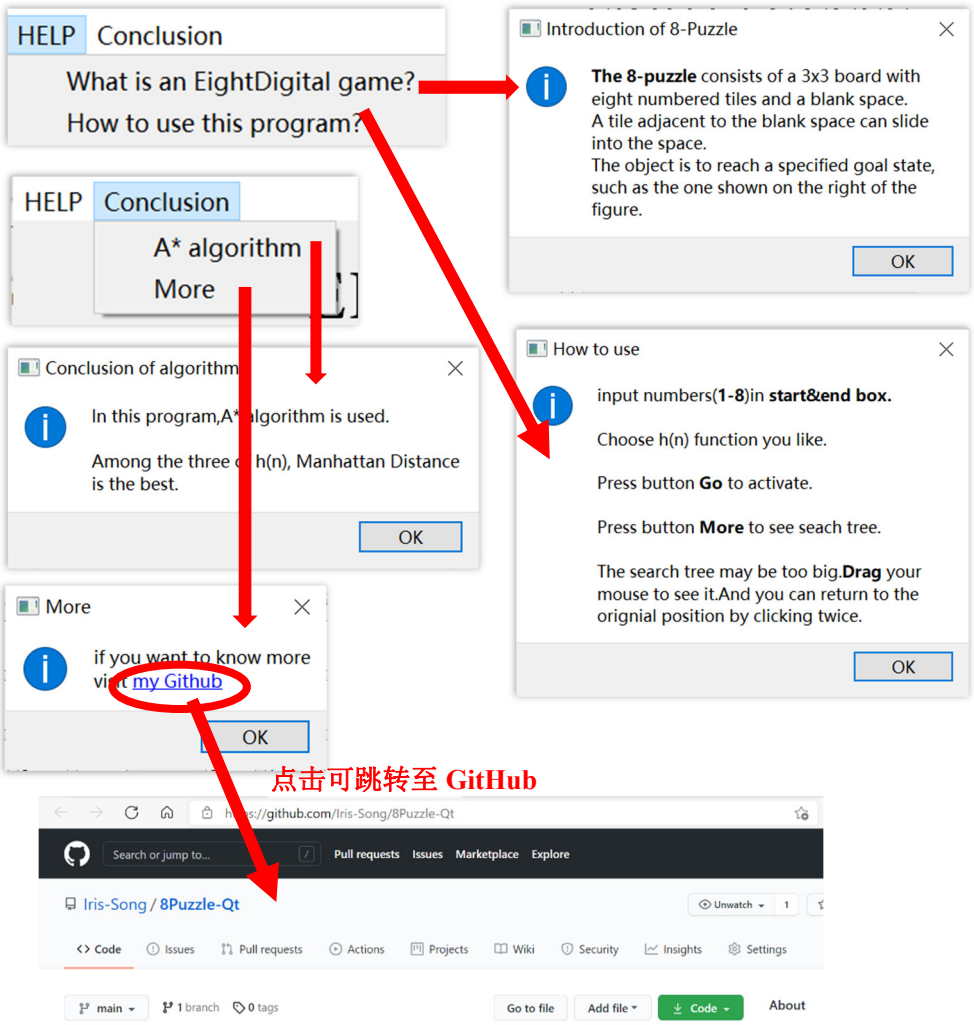

图 3.1.2.2 工具栏展示

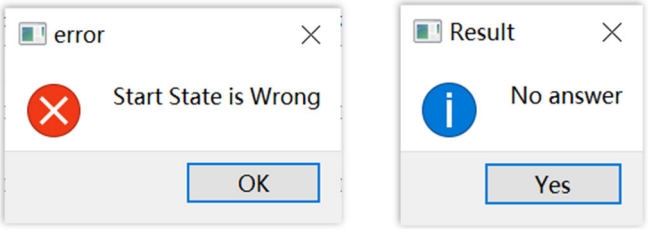

图 3.1.2.3 错误提示

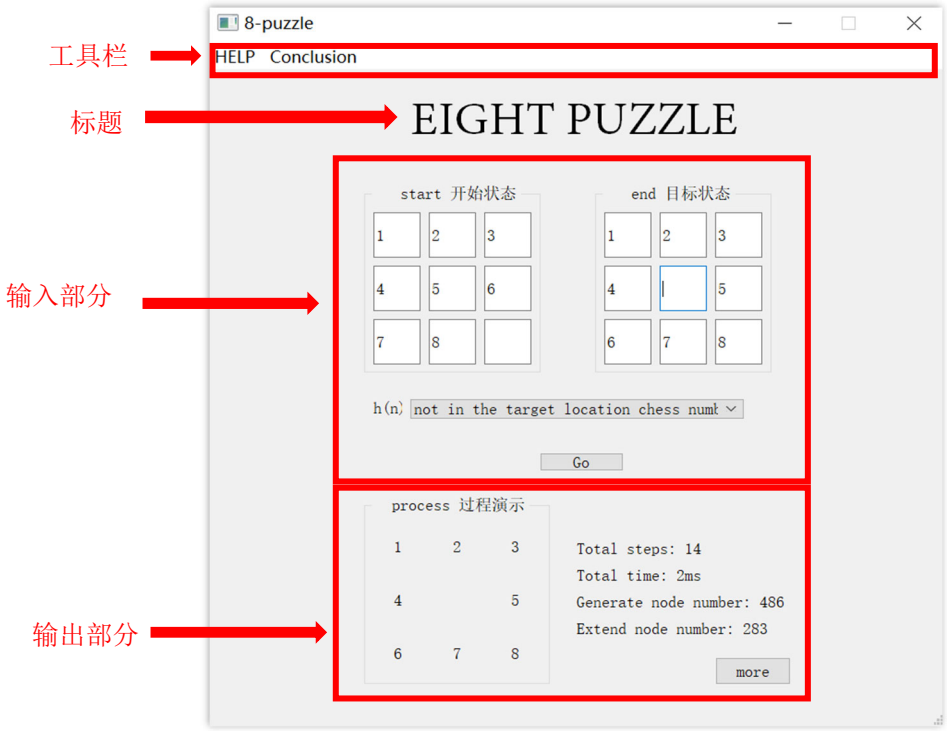

图 3.1.2.3 完整界面展示

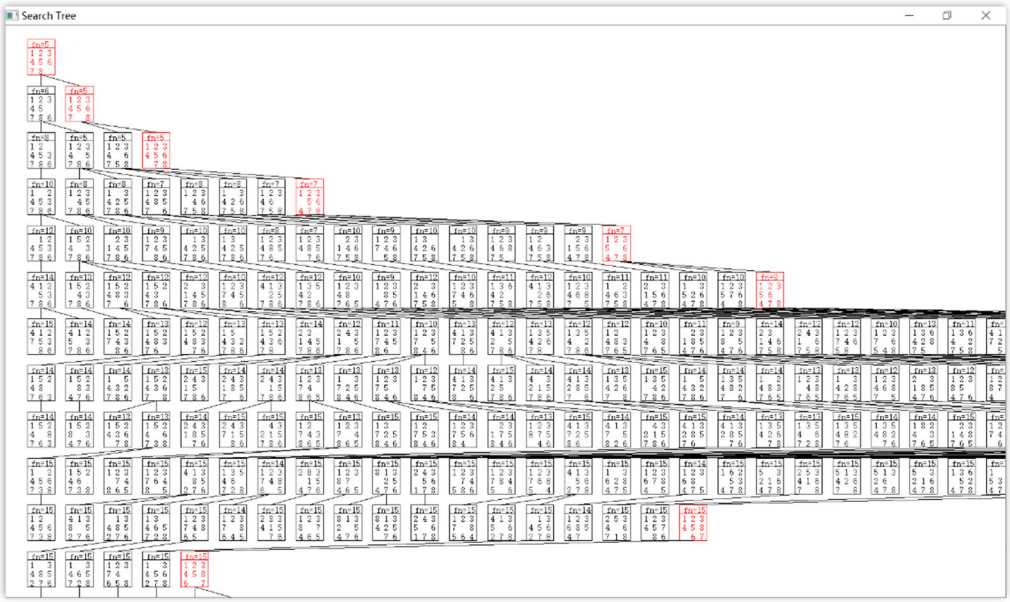

图 3.1.2.4 搜索树展示

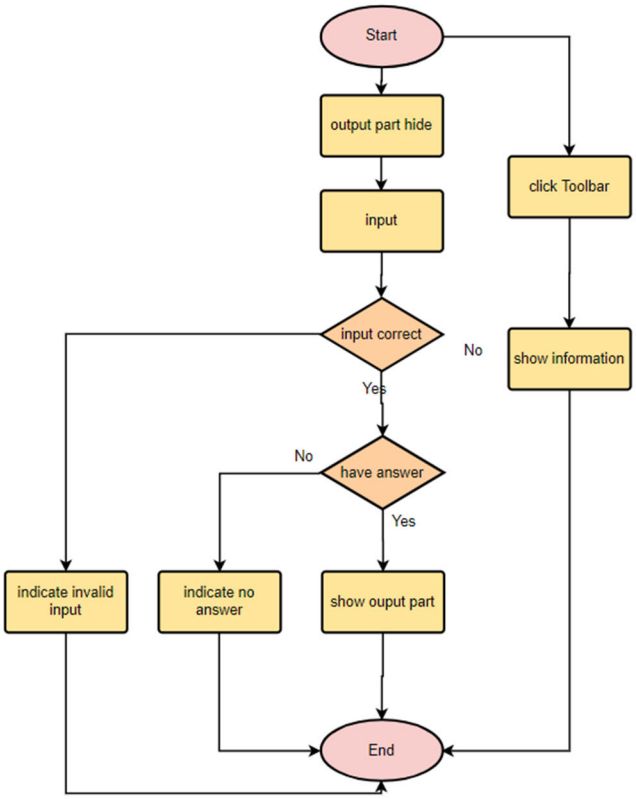

图 3.1.2.5 UI 部分思路流程图

3.2 核心算法及基本原理

搜索算法

该搜索过程是在判断了已经有解的基础上进行的，使用了 A*搜索（详见引文所述），搜索树中的每个结点包括当前矩阵、父节点、f(n)、g(n)、h(n)。该搜索算法使用 extd 优先队列来表示已经拓展出但是还没有访问的结点，使用容器 path 来保存找到的路径。使用 visited map 记录已经访问过的结点，避免对同一结点进行多次访问。

g(n)代表结点 n 的深度，h(n)代表启发函数。

算法具体的实现过程如下：

（1）将初始状态放入 extd 优先队列

（2）将 extd 队首元素弹出，将其放入 path 容器中。

（3）如果该弹出结点与目标结点矩阵相同，则搜索过程结束。否则将弹出结点并标记为已经搜索过。

（4）搜索在弹出节点基础上可以移动得到的结点，如果移动得到的结点没被搜索过，则将其放入优先队列 extd 中，

（5）回到（2）

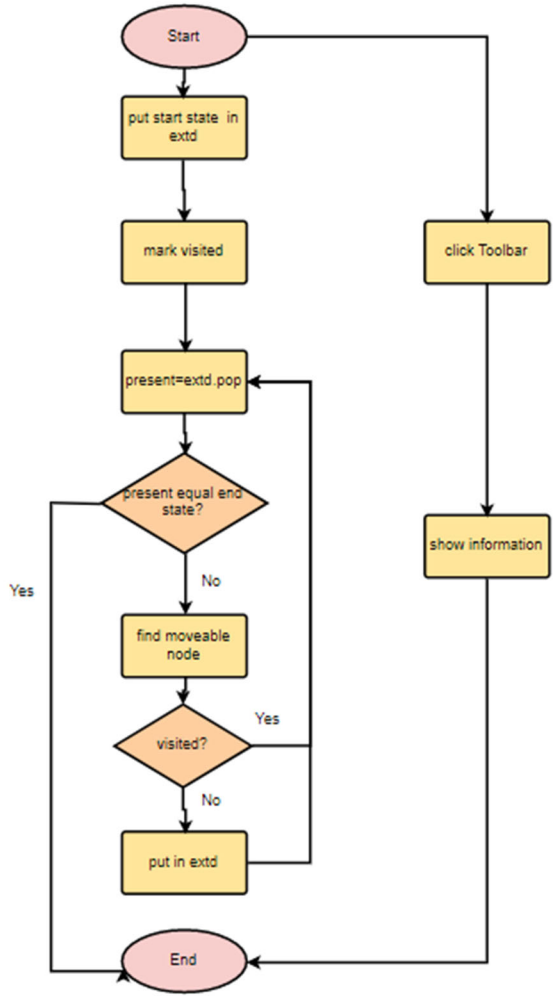

图 3.2 搜索算法流程图

3.3 模块设计

3.3.1 内核模块设计

A. 结点定义

用 Node 类记录每个结点的信息

matrix：记录每个状态的数字矩阵(1-8)，其中 0 代表空格子，SIZE=3

pre：记录父节点在 path 中的下标

f,g,h：分别代表 f(n),g(n),h(n)，f(n)=g(n)+h(n)

```
class Node
{
public:
    int matrix[SIZE][SIZE];
    int pre;
    int f, g, h;
    Node(int matrix[SIZE][SIZE], int pre, int g, int h);//带参的构造函数
    bool operator <(const Node tmp) const;//比较 f(n)大小
```

```
bool operator ==(const Node tmp) const;//比较两个 matrix 数组是否完全一样
};
```

B. 判断是否存在解

通过 SolutionExist 函数来判断解的有无，如果始末状态的奇偶逆序数相同则有解，否则无解。

C. 路径搜索

见 3.2。

D. h(n)选择

我选择了 3 种 h(n)函数。

1. h0：不在目标位置的棋子的个数
2. h1：每个棋子与目标位置的曼哈顿距离。
3. h2：每个棋子与目标位置的欧几里得距离。

D. 路径回溯

利用递归的方式回溯 pre 变量至初始结点，该方法是 UI 界面中的显示过程的内核部分。

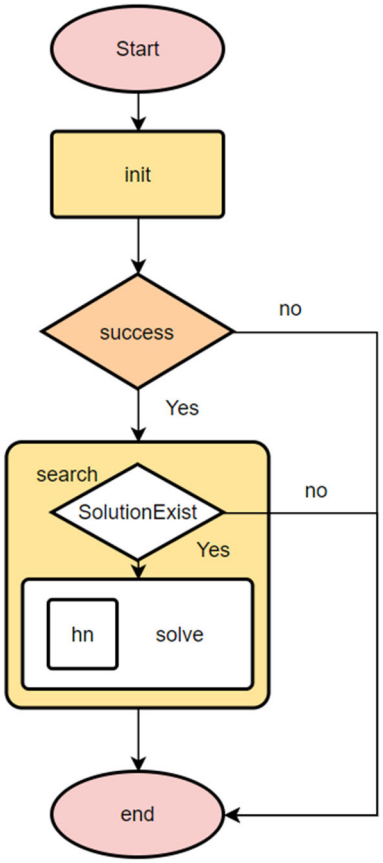

图 3.3.1 内核模块设计流程图

3.3.2 UI 模块设计

UI 模块主要通过使 Qt 的界面设计库实现，对某些函数进行了重载，并添加了信号-槽结构。利用内核模块进行相应的运算后显示在 UI 界面上。与内核模块之间的联系，详见 3.3.3。

A. 搜索树的绘制

重载了 paintEvent 函数，设计了 treenode 类，代表搜索树中每一个结点，以帮助搜索树的绘制。

```
class treenode
{
public:
    //一个节点最多能向四个方向移动，最多有 4 个孩子
    int children[4]={-1,-1,-1,-1};
    //是否为最终路径
    bool isFinal=false;
};
```

采用递归的方法，用 vector<treenode> form 记录父子结点间的关系，记录每一层的结点数量。调用 Qt 中的 QgraphicsView、Qpainter、Qpixmap 库函数从初始结点绘制搜索树，绘制时标记每个结点的数字矩阵和 f(n)的值，最优路径上的结点用红色绘制。

B. 搜索树的移动

由于窗口大小的限制，较大的搜索树不能在界面上完全显示，这时要查看搜索树的边缘部分就需要移动搜索树。我重载了鼠标的点击与释放函数来模拟拖放效果。根据其在“拖放”过程中移动的距离，通过适当的比例来计算其位移的改变，通过双缓存绘制的方式模拟搜索树的移动。

双击鼠标左键可回归原位。

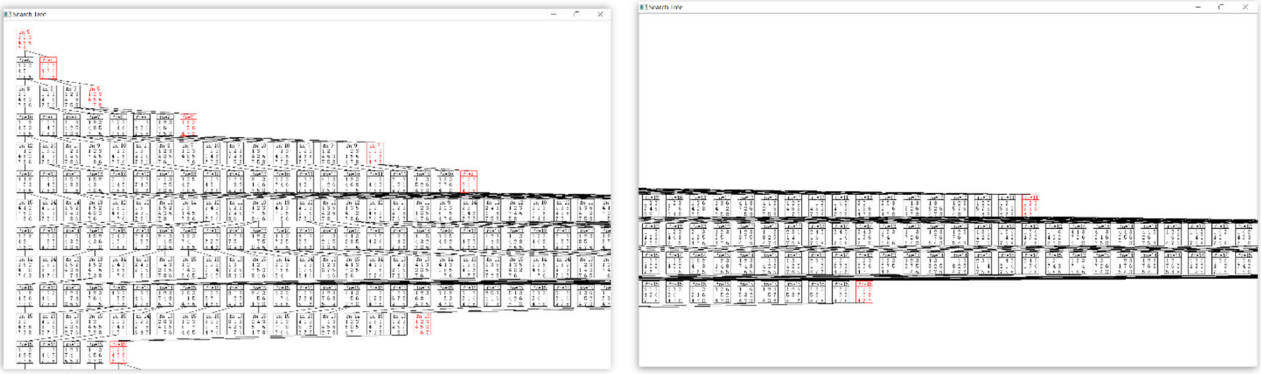

图 3.3.2.B 搜索树的移动前(左)后(右)

C. 菜单栏与按钮的实现

利用信号-槽结构实现。

D. h(n)的选择

使用 Qt 库中的 Combo Box 实现。根据 Combo Box 返回选择的下标来确定选择的 h(n)函数。默认选择 h0，避免了用户忘记选择的情况。

3.3.3 内核部分与 UI 部分的联系

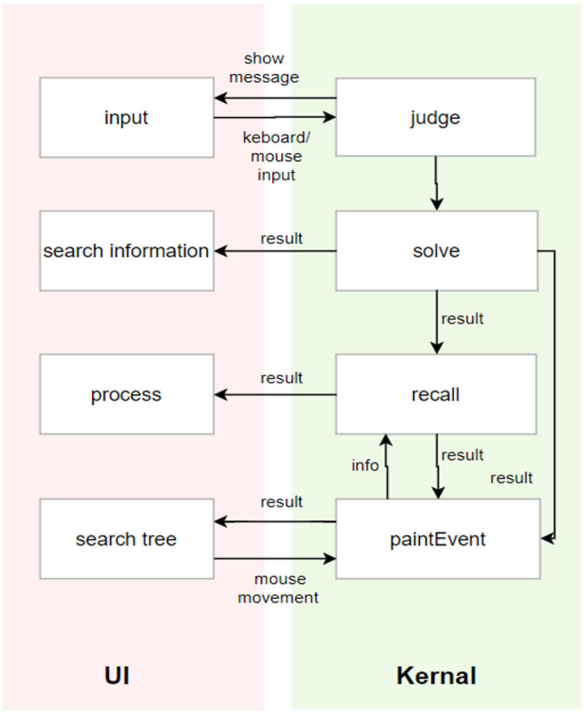

图 3.3.3 内核部分与 UI 部分的联系

3.4 创新内容或优化算法

A. 避免重复结点的访问

用 visited map 来标记该状态是否被访问过，如果被访问过则不会再被重复访问，从而大大提高了效率。

B. 搜索树的展示

详见 3.2.2B 搜索树的移动。

4 实验过程

4.1 环境说明

硬件配置：Intel Core i5-10210U CPU

操作系统：64 位 Windows10

开发语言：C++

开发环境：Qt Creator (Qt 5.12.10 MSVC2017 64bit)

Visual Studio 2019

编译器：Microsoft Visual C++ Compiler 16.5.30104.148(x86_amd64)

调试器：CDB Engine

4.2 源代码文件清单及主要函数清单

4.2.1 头文件

A. head.h

包含与 STL 有关、与时间有关、与 UI 界面有关的头文件，包含关于 Node 类的声明。

B. mainwindow.h

包含与 UI 界面有关的头文件，声明了 MainWindow 类，该类中包含与主界面有关的（与搜索相关的）变量与函数(包括 slot 函数)。

C. detailwindow.h

包含与 UI 界面有关的头文件，声明了 treenode 类、detailindow 类，detailindow 类中包含与副界面有关的（与绘制搜索树相关的）变量与函数。

D. ui_mainwindow.h

Qt 根据 mainwindow.ui 自动生成的头文件

4.2.2 源文件

A. main.cpp

包含 mainwindow.h

main 函数所在，显示主程序

B. mainwindow.cpp

包含 mainwindow.h，detailwindow.h，ui_mainwindow.h

a. MainWindow 类的构造函数和析构函数

b. bool MainWindow::**init**()

对 MainWindow 类中的变量进行初始化，并判断始末状态的输入是否合法，初始化成功返回 true，否则返回 false，并会有错误信息框提示

c. void MainWindow::**IntroClicked**()

工具栏 HELP 菜单下拉栏中关于八数码介绍的槽函数。会有消息框介绍相关信息。

d. void MainWindow:: **UsageClicked**()

工具栏 HELP 菜单下拉栏中关于该程序使用的槽函数。会有消息框介绍相关信息。

e. void MainWindow:: **AlgoClicked**()

工具栏 Conclusion 菜单下拉栏中关于 A*算法的槽函数。会有消息框介绍相关信息。

f. void MainWindow:: **MoreClicked**()

工具栏 Conclusion 菜单下拉栏中关于更多情况的槽函数。会有消息框介绍相关信息。

g. void MainWindow::**on_GoButton_clicked**()

GoButton 的槽函数，左键单击可开始搜索，在进行过程演示的时候会设为不可点击状态，结束后会设为可点击状态。

h. void MainWindow::**on_MoreButton_clicked**()

MoreButton 的槽函数，在初始化成功的条件下可左键单击展示 search tree 窗口。

C. detailwindow.cpp

包含 detailwindow.h

a.Node 类的构造函数和析构函数

b. void detailwindow::**mouseReleaseEvent**(QMouseEvent *event)

对 mouseReleaseEvent 进行重载，得到鼠标释放时的坐标，并刷新视图。

c. void detailwindow:: **mousePressEvent** (QMouseEvent *event)

对 mousePressEvent 进行重载，得到鼠标点击时的坐标，并刷新视图。

d. void **DrawOneMtrix**(Node nd,bool dst,long long centerX,long long centerY,QPainter& painter)

在以 centerX 和 centerY 为 x、y 坐标的中心上画一个结点的矩型。该矩形包括最上方的长方形和下方的正方形。长方形内标注了 f(n)的大小，正方形内是数字矩阵。若 dst 为 true，则该节点会被画成红色，为 false 则为黑色。

e. void detailwindow::**DrawTree**(vector<int> &LyrMtrxNum,int depth,int totalDepth, vector<treenode> &form,QPainter& painter,int ndNum,long long centerX,long long centerY,long long x,long long y)

递归的绘制搜索树。

f. void detailwindow::**paintEvent**(QPaintEvent *event)

对 paintEvent 进行重载，生成 painter，根据鼠标的拖动计算初始坐标，调用 DrawTree 函数绘制搜索树。

D. solution.cpp

包含 mainwindow.h，ui_mainwindow.h

a. bool **SolutionExist**(const int start[SIZE][SIZE],const int end[SIZE][SIZE])

判断输入的始末状态是否有解。

b. int **hn**(const int present[SIZE][SIZE], const int end[SIZE][SIZE])

求某个数字矩阵的估值函数。

c. void MainWindow::ShowMatrix(const Node nd)

用递归的方法，动态展示数字矩阵的变换过程。

d. int **HashSet**(int matrix[SIZE][SIZE])

将数字矩阵通过哈希变换映射为一个 int 型的值。

e. bool **range**(int i)

检查下标是否在范围内

f. void MainWindow::Search()

根据合法的起始状态进行搜索，具体算法详见 3.2

E.node.cpp

包含 head.h

a.Node 类的构造函数和析构函数

b.<和==的重载函数

<重载后比较 f 的大小，==号比较数字矩阵是否完全相等

4.2.3 表单文件

A. mainwindow.ui

主窗口图形化界面绘制

4.3 实验结果展示

4.3.1 UI 界面展示

简单起见，只展示一组结果。

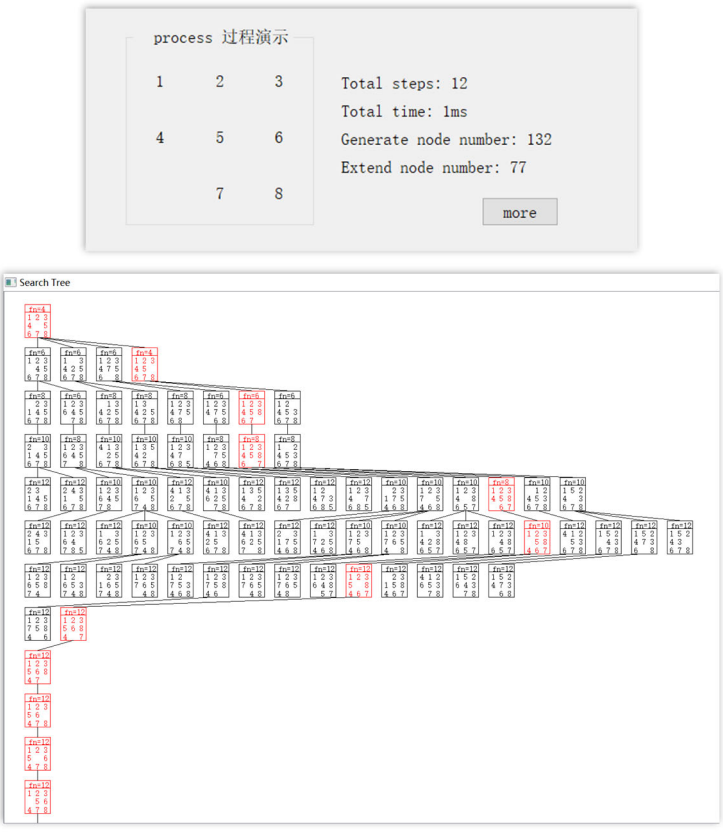

图 4.3.1 结果展示

4.3.2 结果分析

我随机测试了 20 种不同始末状态下运行时间、、的差异，对比三种不同的评估函数，并绘制了表格和折线图。

序号	总步数	h0 不在末态位置的点的数量			h1 曼哈顿距离			h2 欧几里得距离		
		运行时间(ms)	生成节点数	扩展节点数	运行时间(ms)	生成节点数	扩展节点数	运行时间(ms)	生成节点数	扩展节点数
1	22	21	10239	6487	7	2167	1365	22	6410	4001
2	21	26	9314	5731	3	1161	703	11	5041	3069
3	22	28	14247	9156	4	1892	1168	13	5224	3266
4	15	2	561	336	1	268	160	1	616	364
5	14	1	486	283	1	193	114	1	292	168
6	13	2	354	209	0	143	84	1	296	172
7	2	0	4	3	0	4	3	0	4	3
8	28	354	128534	92280	8	4129	2638	74	29179	18486
9	14	1	463	270	1	193	114	1	352	205
10	15	3	759	458	1	241	142	1	598	356
11	22	28	13746	8801	2	1316	811	15	5425	3393
12	21	18	9396	5804	5	1271	778	18	5202	3159
13	22	21	10389	6601	6	1237	760	21	6417	4001
14	12	1	236	139	0	121	72	1	214	125
15	13	2	414	241	0	144	88	0	238	137
16	14	1	551	337	1	154	91	1	416	242
17	21	19	9476	6036	3	1440	889	16	5545	3464
18	22	28	14886	9291	7	2080	1300	24	5976	3638
19	21	17	9296	5868	2	1018	630	11	4956	2918
20	27	194	81769	57175	1	593	395	57	21521	13612

图 4.3.2.1 20 个测试点对比表

从结果对比中不难发现，不论是从运行时间还是结点生成数和拓展数上 h1（曼哈顿法）的效率都是最高的，其次是 h2（欧几里得距离法），h0（根据不在末态位置的棋子数来估值）的效果最差，且 h1 远远优于 h0。

不论采用这三种哪一种估值函数，其最优路径的步数都是一致的，其三者 h(n)<h*(n),均是有效的。

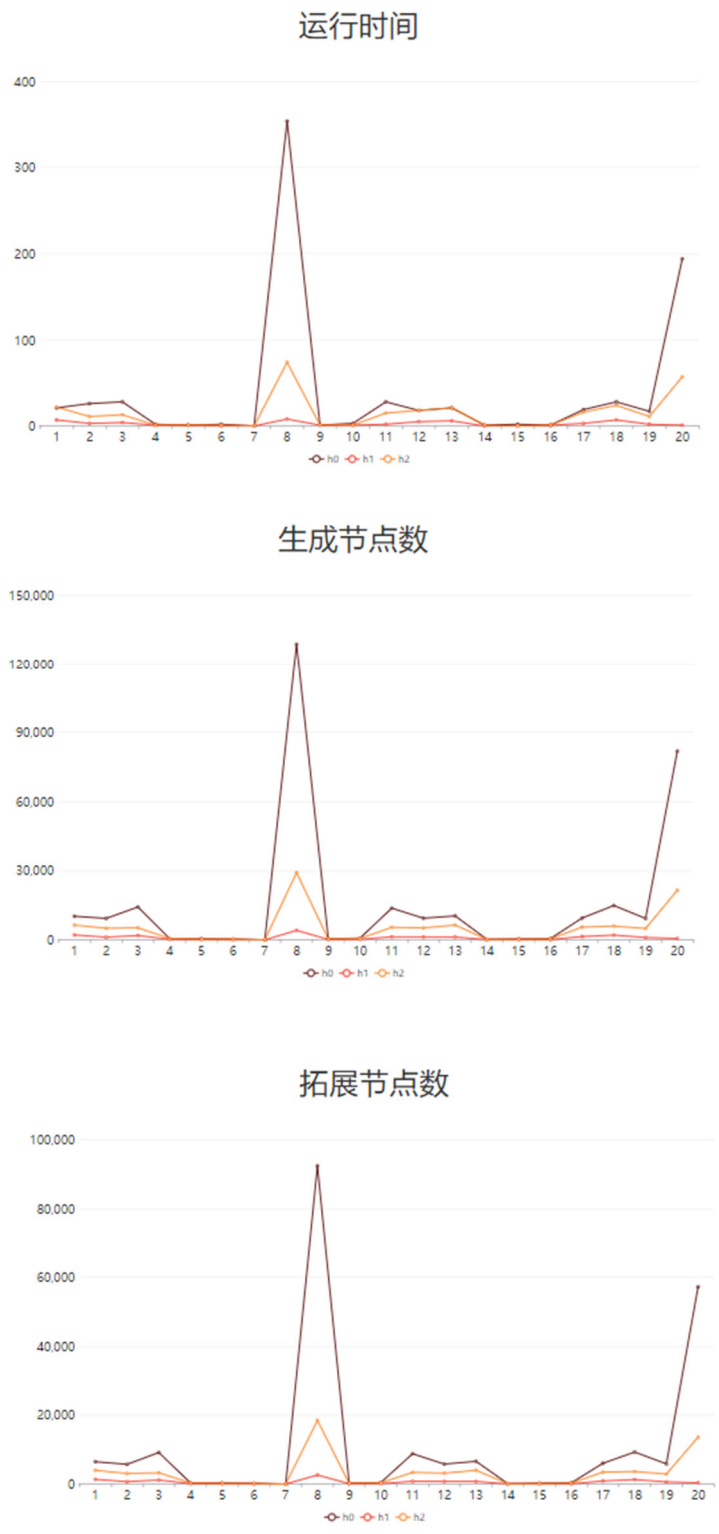

图 4.3.2.2 20 个对比折线图

装 订 线

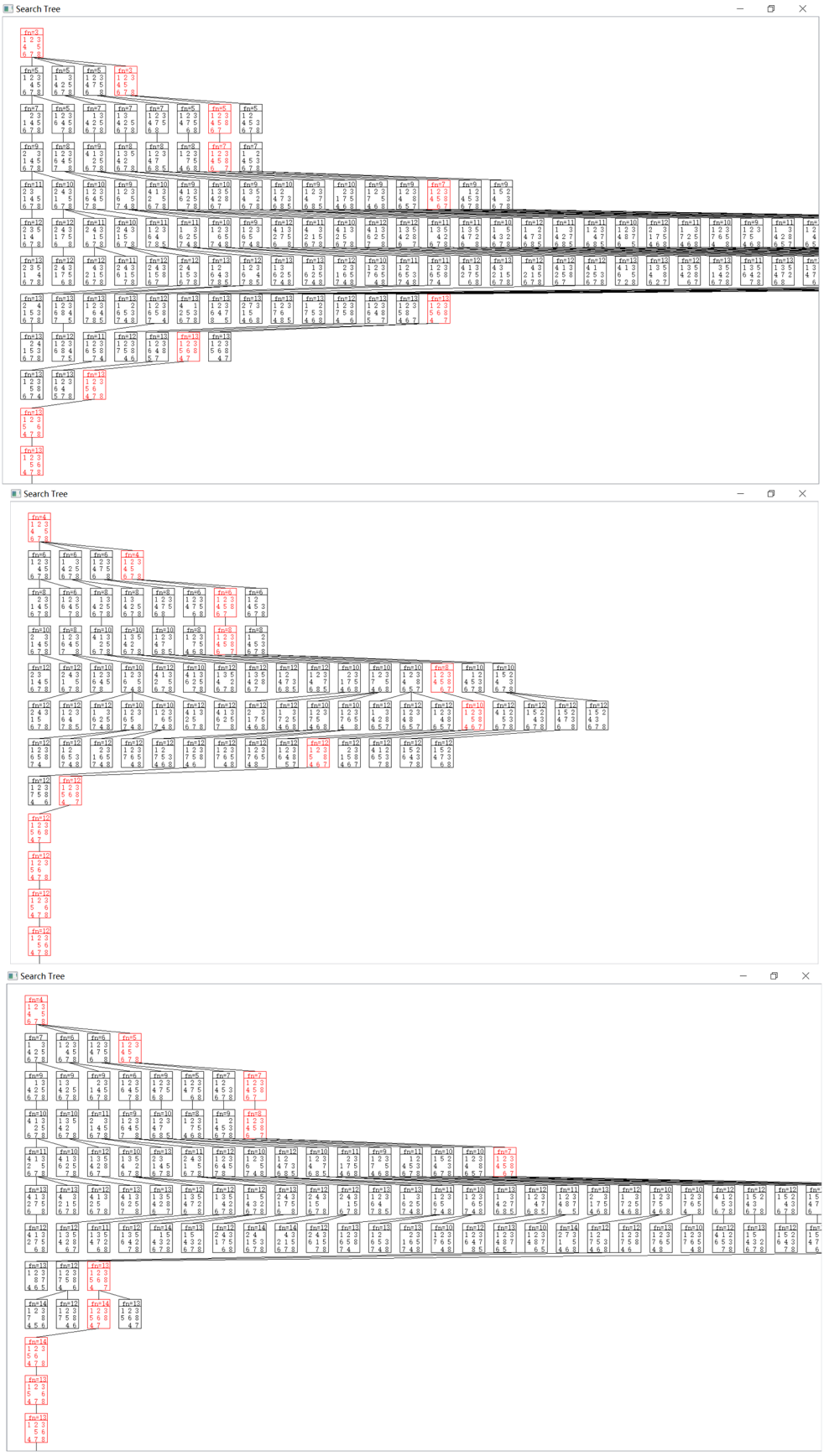

图 4.3.2.3 相同初始状态情况下的搜索树，由上到下依次为 h0、h1、h2 时的情况

5 总结

5.1 实验中存在的问题及解决方法

（1）重载 paintEvent 函数时，出现过逻辑混乱的情况，多增添注释，画流程图较好的解决了这一问题。

（2）对如何绘制搜索树，寻找结点感到茫然。参考了相关资料，想到可以新建 treenode 类来解决难以找到子节点的问题。

（3）搜索树显示不全。因为这次实验才第一次接触 Qt，在图形化的过程中中遇到了许多困难，其中之一就是搜索树，我最初采用的是 Qpainter 绘制，因其与 cmd 界面类似比较简单。由于在某些始末状态下搜索树太大，搜索树无法绘制完整，我采取了重载 mouseEvent 方法来实现界面的移动。

5.2 后续改进方向

（1）搜索树虽可以移动，但无法放大、缩小。考虑使用 Qt 中的其他库函数（如 GraphicsView）来解决这一问题

（2）考虑寻找合适的剪枝方法，加快解决问题的速度

（3）由于八数码的问题规模较小，对空间的要求不高。但仍希望，寻找合适的方法，减少内存消耗。

5.3 心得体会及自评

本次实验全部由我自己完成，在效率上比较高，但是自己可能难以发现程序中存在的一些问题。借着这次实验学习了 Qt 的相关知识，图书馆内的书籍和网上的资料都为我提供了很大帮助。实验的时间大多都耗费在图形化界面的的学习上了。我搜索了网上许多有关八数码的论文，略有遗憾的是没能发现或是想出什么创新性的 h(n)函数。但实验的效果总体很理想，也比较符合我个人的预期。本次实验也帮助我更好的掌握了 A*算法。

自我感觉较好的完成了本次课设。

参考文献

[1] Jasmin Blanchette, Mark Summerfield. C++ GUI Qt 4 编程. 北京:电子工业出版社,2018，2000(2):5-8.

[2] https://zh.wikipedia.org/wiki/A*%E6%90%9C%E5%B0%8B%E6%BC%94%E7%AE%97%E6%B3%95

[3] http://ai.stanford.edu/~latombe/cs121/2011/slides/D-heuristic-search.pdf

[4] Stuart Russell, Peter Norvig. Artificial intelligence: a modern approach 北京:人民邮电出版社, 2002

装

订

线