A*算法求解 8 数码问题

摘 要

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。本实验基于 C++语言，运用 A*算法对八数码问题进行目标结点搜索，通过 Qt 实现界面的可视化及搜索树的绘制，并选择多种不同的启发函数，对其搜索效率进行了多方位的比较。

关键词：A*算法，启发函数，八数码，Qt

装 订 线

A* algorithm to solve 8 digital problems

ABSTRACT

The eight digital problem is also known as the nine house problem. On a 3 by 3 board, the pendulum has eight pieces. Each piece is marked with a number from 1 to 8. There is also a space on the board, and adjacent pieces can be moved to the space. The problem is to give an initial state and a target state, and to find a moving step with the least number of moves from the initial state to the target state. Based on C++ language, this experiment uses the A* algorithm to search the target node of eight-digit problem, uses Qt to realize the visualization of the interface and the drawing of the search tree. Also, we select a variety of different heuristic functions to make a multi-directional comparison of its search efficiency.

Key words： A* algorithm, heuristic function, octuple, Qt

装 订 线

目 录

装 订 线

1 实验概述

1.1 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。

1.2 实验基本内容及要求

1.2.1 基本内容

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)。

1.2.2 实验要求

（1）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。

（2）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。

（3）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。

（4）撰写实验报告，提交源代码、实验报告、汇报 PPT。

1.3 本文所做的工作

在本实验中，我们深入研究了 A*算法，并将其应用到八数码问题的求解，通过 C++语言，并利用 STL 模板库简化代码复杂度，我们实现了由初始状态到目标状态最短路径的寻找。程序中挑选了四种代价估计函数 f(n)，并从时间、步骤、拓展结点数等多方面分析了不同代价估计函数的效率，进而深入理解不同代价估计函数之间效率差异的原因，从而指导其他问题中 f(n)的选择。

同时，为了方便用户的使用，我们利用 Qt Company 开发的跨平台 C+图形用户界面应用程序开发框架，实现了可以人机交互的 UI 界面，在这样一个界面中，用户可以自主输入初始和目标状态，选择使用的代价估计函数，在点击开始运行的按钮后，用户可以看到可视化的运行过程、运行步骤、运行时间以及搜索树等等，在 help 工具栏中，可以查阅到使用方法以及本团队信息。

2 试验方案整体概述

2.1 总体设计思路与总体框架

总体设计思路可以分为数码问题的内核算法设计以及外部 UI 界面的设计两大部分，两部分之间的交互如图 1 所示：

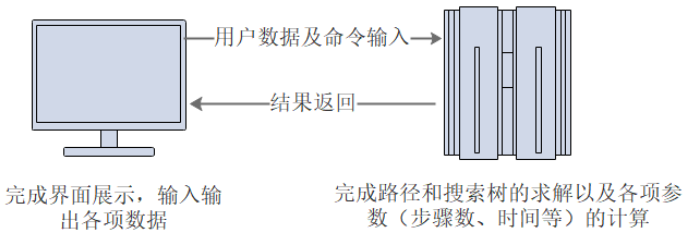

图 1 内核算法与 UI 界面交互示意图

2.1.1 内核部分设计思路

内核部分使用 A*算法，搜索树中的每个结点包括：当前矩阵、父节点、代价估计函数值等，为存储搜索树并求解下一个扩展结点，设置优先队列 open 表、map 容器 close 表和向量 path。open 表用于存储已扩展出但是还没有访问的节点，使用优先队列进行自动排序；close 表存储已经访问过的节点，避免重复访问；path 向量用于保存路径。关于寻找路径，采用以下方式：

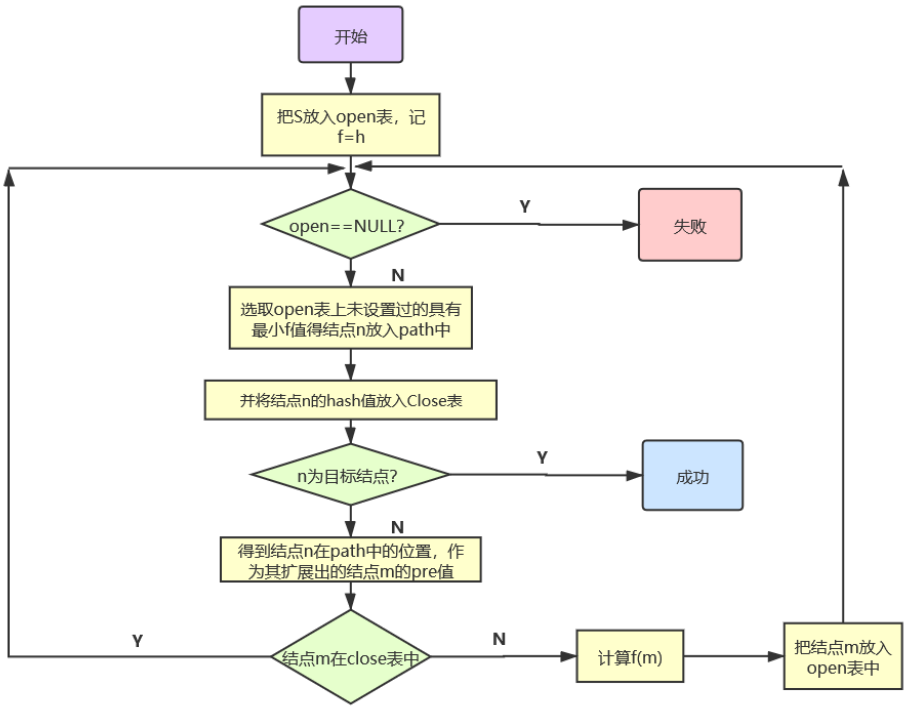

图 2 内核部分设计思路流程图

（1）先将最开始的结点放入 Open 表

（2）将 Open 表中代价估计值最小的结点取出放入 path 中，若已经为目标结点，则找到路径。否则将其放入 Close 表，并生成扩展集

（3）对于每个扩展出的结点，如果不在 Close 中，就将其放入 Open 表

（4）回到 2

若找到目标结点， 则由目标结点通过递归方式向前回溯,直到最开始的结点为止，然后递归打印。（流程图如图 2）

2.1.2 UI 界面设计思路

UI 界面初始状态如图 3，界面上从左到右、从上到下设置有：帮助工具栏、初始和目标状态输入框、运行过程演示框、f(n)选择框、开始运行键及细节展示键。

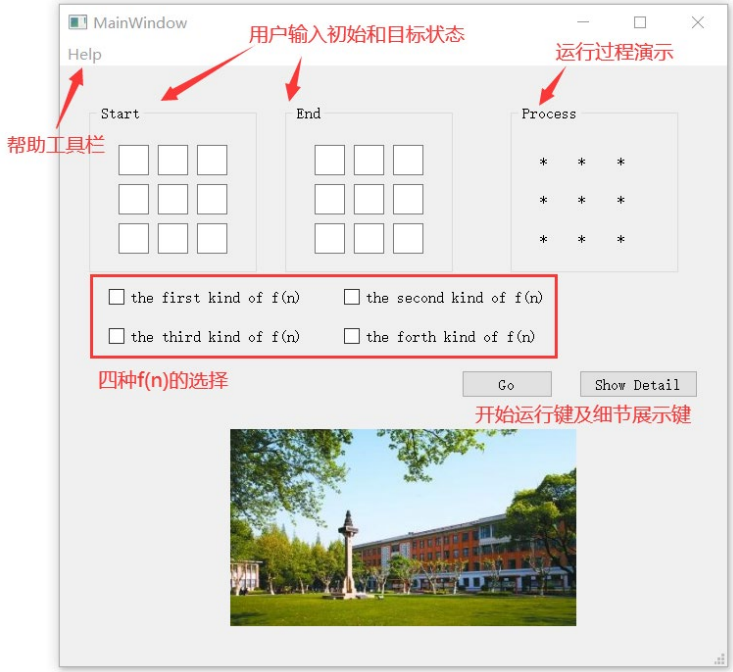

图 3 UI 界面初始状态

输入正确的初始和终止状态，并正确选择 f(n)之后点击 Go 按钮，可以看到 Process 模块内的运行情况，其设置为每隔一秒向后推移一步。

在运行过程中，Go 按钮设置为灰色不可点击，在运行完成时 Go 按钮会重新开启。

运行结束后，可以点击 Show Detail 键观察运行步骤、运行时间、总结点数等（如图 4），此时点击 Full Tree 按钮则会显示出完整的搜索树，最短路径用红色标出（如图 5，由于整个搜索树过大，只展示一部分）。

图 4 Detail 界面

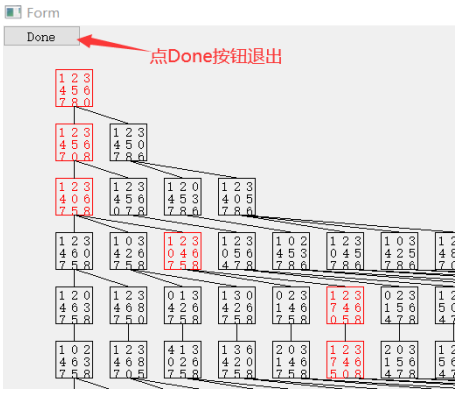

图 5 部分搜索树展示

在 Help 工具栏中可以查阅使用方法（Usage）和团队信息（About_us），如图 6

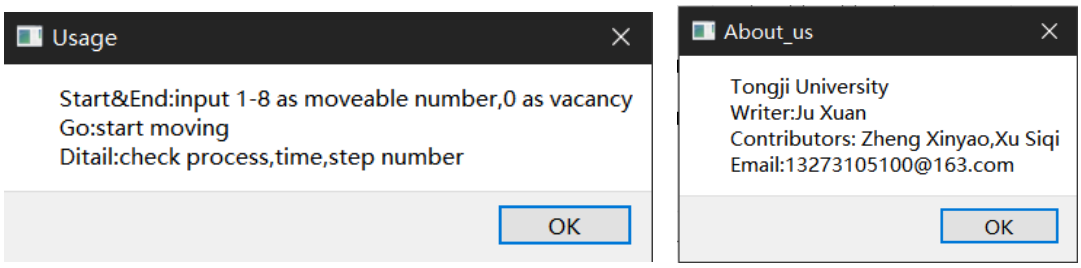

图 6 Help 工具栏信息

输入错误则会给出错误提示，如图 7

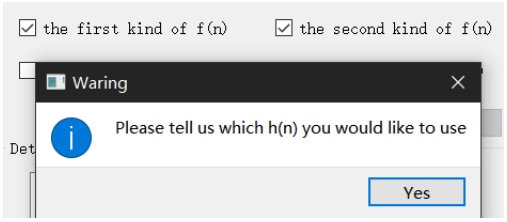

图 7 错误提示

2.2 核心算法及基本原理

A*算法是一种有序搜索算法，其特点在于对估价函数的定义上。这个估价函数 f 使得在任意结点上其函数值 f(n)能估算出从结点 S 到结点 n 的最小代价路径的代价与从结点 n 到某一目标结点的最小代价路径的代价的总和，也就是说 f(n)是约束通过结点 n 的一条最小代价路径的代价的一个估计。我们定义函数 f*，使得在任一结点 n 上其函数 f*(n)就是从结点 S 到结点 n 的一条最佳路径的实际代价加上从结点 n 到某目标结点的一条最佳路径的代价之和，即

$$f^*(n) = g^*(n) + h^*(n)$$

我们希望估价函数 f 是 f*的一个估计，此估计可由下式给出:

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

其中：g 是 g*的估计；h 是 h*的估计。对于 g(n)来说，一个明显的选择就是搜索树中从 S 到 n 的这段路径的代价，这一代价可以由从 n 到 S 寻找指针时，把所遇到的各段弧线的代价加起来给出(这条路径就是到目前为止用搜索算法找到的从 S 到 n 的最小代价路径)。这个定义包含了 $g(n)\geq g^*(n)$。对于 h*(n)的估计 h(n),它依赖于有关问题的领域的启发信息。于是称作启发函数。启发函数中，应用的启发信息(问题知识)越多,扩展的结点就较少,就能更快地搜索到目标结点。总结下来，我们以 d(n)表达状态 n 到目标状态的距离，那么 h(n)的选取大致有如下三种情况：

（1）如果 h(n)< d(n)到目标状态的实际距离，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。

（2）如果 h(n)=d(n)，即距离估计 h(n)等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。

（3）如果 h(n)>d(n)，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

在我们的程序中,我们将运用了下面四种不同的估价函数,并比较其运行效果。

- $f(n)=g(n)+h_1(n)$

其中 g(n)代表搜索树中结点(n)的深度，根结点的深度为 0。启发函数 $h_1(n)$代表与目标结点相比,结点错放棋子个数。

- $f(n)=g(n)+h_2(n)$

其中,g(n)代表搜索树中结点(n)的深度,根结点的深度为 0。启发函数 $h_2(n)$定义为每一个结点与其目标位置之间的曼哈顿距离

- $f(n)=g(n)+h_3(n)$**(仅仅作为对比，实际在本题目中并不是最合适的)**

其中 g(n)代表搜索树中结点(n)的深度，根结点的深度为 0。启发函数 $h_3(n)$定义为每一个结点与其目标位置之间的欧几里得距离

- $f(n)=g(n)+h_4(n)$**(仅仅作为对比，实际在本题目中并不是最合适的)**

其中,g(n)代表搜索树中结点(n)的深度,根结点的深度为 0。启发函数 $h_4(n)$定义为每一个结点与其目标位置之间的对角线距离

2.3 模块设计

模块设计部分同样分为内核算法部分与 UI 界面设计部分两块进行介绍。

2.3.1 内核部分模块设计

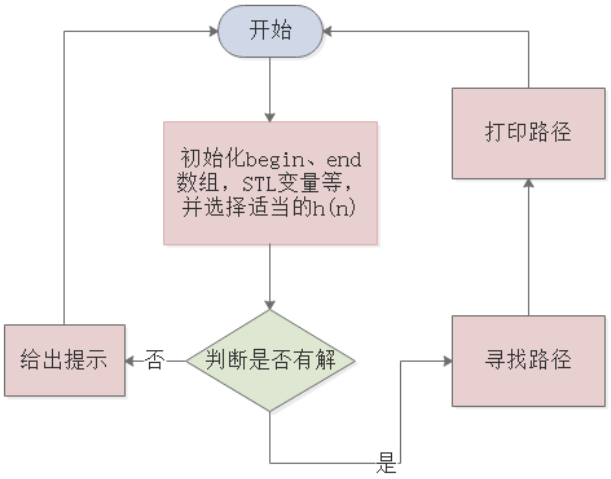

图 8 内核部分模块设计

【结点定义】Item 结构体记录每个状态结点的信息

```
struct Item{
    int matrix[MATRIX_N][MATRIX_N];   //#define MATRIX_N 3
    int pre;
    int f_n, g_n, h_n;
    Item(int matrix[MATRIX_N][MATRIX_N], int pre, int g_n, int h_n);//带参的构造函数
    bool operator <(const Item temp) const;//通过 f_n 比较大小，应用于优先队列的排序
    bool operator ==(const Item temp) const;//比较两个数组是否一样（每个元素）
};
```

matrix：每个状态的数字分布矩阵，其中有 1-8，0 代表空格子

pre：父节点在 path 中的下标(具体说明见下)

f_n，g_n，h_n：即 f(n),g(n),h(n)；f(n)=g(n)+h(n),g(n)=n,h(n)有两种选取方法

【判断是否存在解】

通过 solution_exist 函数判断解的存在性，如果开始和目标矩阵的逆序数奇偶性相同，则有解，否则无解

【最短路径寻找】

参见 2.1.1

【h(n)的选取】

h1(n)：用不在位置的棋子个数总和作为启发式函数，估价值 h(n)<= n 到目标节点的距离实际值，搜索的点数多，搜索范围大，效率低，但能得到最优解。

h2(n)： 曼哈顿距离是标准的启发式函数，其值是两个点在标准坐标系上的绝对轴距总和，在多数网格地图中的启发式算法中，曼哈顿距离能表现出较好的性能，在我们的实验所用到的四种启发式函数中，曼哈顿函数表现出的性能也最优。

h3(n)：如果单位可以沿着任意角度移动（而不是网格方向），可使用直线距离，即欧几里德距离。我们的任务中，虽然不能沿任意距离移动，但取这个 h(n)作为对比。这种方法的缺点是，直接使用 A*时将会遇到麻烦，因为代价函数 g 不会与启发函数 h 匹配。且平方根运算也将耗费一些时间。

h4(n)： 这是一种综合了直线运动与对角运动（曼哈顿距离和欧几里德距离）的方法，从实验结果可以看出，在此题中这个估价函数并没有表现出较好的性能，这是由于其并不适应本题题目要求的缘故，但在允许对角运动的情况下，此函数的性能较优。

【打印路径】

在 PrintPath 函数中通过递归的方式，从目标结点开始按照结点的 pre 变量回溯，直到回溯到初始结点，依次打印结点、跳出函数，直至打印到初始结点

2.3.2 UI 界面部分模块设计

UI 界面部分主要使用信号-槽的结构，调用 Qt 中的界面设计库，完成对各项任务的回应，其结构如图 9 所示

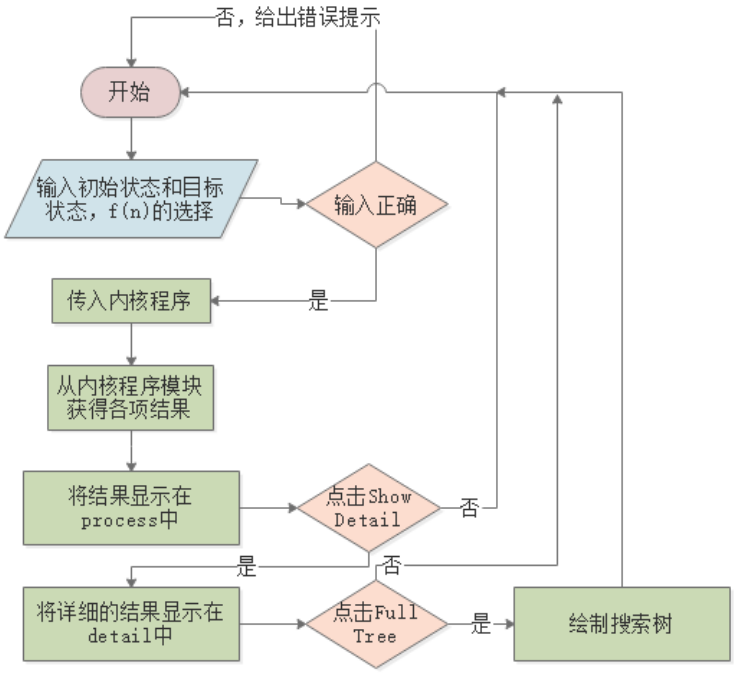

图 9 UI 界面部分模块设计

【绘制搜索树】

调用 Qt 中的 QPainter 和 QPen 库，采用递归方式进行绘制整个搜索树，通过 vector<node> 类型的 form 向量来记录父节点与子节点的关系，通过 vector<int> 类型的 present_draw_of_each_path 来记录每层已绘制的结点数，从初始结点开始递归画出孩子结点，在最优路径上的结点用红色绘制

3 实验过程

3.1 环境说明

3.1.1 操作系统

64 位 Microsoft Windows

3.1.2 开发语言

C++

3.1.2 开发环境

开发环境: Qt Creater 5.14.2 for VS2017

编译器:Microsoft Visual C++ Compiler 15.9.28307.423(x86_amd64)

调试器: Auto-detected CDB

配套构建: Desktop Qt 5.14.2 MSVC2017 64bit

3.2 源代码文件清单及主要函数清单

3.2.1 头文件

(1) head.h

包含 STL 容器的头文件 (queue,map,vector), 时间记录的头文件(time.h),包含宏定义(MATRIX_N)和每个状态节点信息记录的结构体 (Item)。

(2) mainwindow.h

程序 ui 界面的的头文件和部分函数的声明, 包括打印路径的函数声明和鼠标定位相关函数以及 h(n)实现的声明。

(3) show_tree.h

绘制最优数的头文件。

3.2.2 源文件

(1) find_path.cpp(算法核心，判断是否有解，寻找路径打印路径,包含 mainwindow.h)

solution_exist:判断初始状态目标状态逆序数奇偶性是否相等,判断八数码是否有解;
hash_index:得到当前表的 hash 值，在 find_path 中对 open、close 表的处理中会用到;
MainWindow::PrintPath:递归打印路径，用于程序主页面的八数码显示;
MainWindow::find_path:循环对结点进行扩展，对 open、close 表进行处理。

(2) hn.cpp(四个 hn 函数的代码,包含 mainwindow.h)

calculate_h_n_1：计算不在位置的棋子个数之和；
calculate_h_n_2：计算曼哈顿距离，即当前结点与目标结点横纵坐标距离和；
calculate_h_n_3：计算为欧几里得距离；
calculate_h_n_4：计算对角线距离。

(3) item.cpp(item 结构体的相关函数，包含 head.h)

Item：更新每个结点的值和 h(n),g(n),f(n)；
Item::operator <：比较两个 item 的 fn 的大小；
Item::operator ==：判断两个结点值是否相等。

(4) main.cpp(显示程序界面，包含 mainwindow.h)

(5) mainwindow.cpp(程序界面，包含 mainwindow.h,show_tree.h)

MainWindow::init_start_and_end:初始化 start 和 end 数组，并检测使用哪种 hn；
About_clicked,Usage_clicked,Show_tree_clicked:三种功能的鼠标定位。

(6) show_tree.cpp(画最优搜索树的相关函数，包含 show_tree.h)

Show_tree:画最优搜索树的 ui 界面赋值；
Show_tree::done_button_clicked:完成功能，按 done 关闭窗口；
draw_one_matrix：设置颜色，寻找位置；
Show_tree::draw_tree：找子节点坐标，递归画图；
Show_tree::paintEvent：宏观调控画图实现画树功能。

3.2.3 其他文件

mainwindow.ui：画出动态图；
show_tree.ui；画出最优搜索树；
tongji.jpg：图像界面所用到的图片。

3.3 实验结果展示

3.3.1 UI 界面展示

具体内容可以通过运行 eight_digital_problem.exe 得到,此处只展示一次运行结果(如图 10)

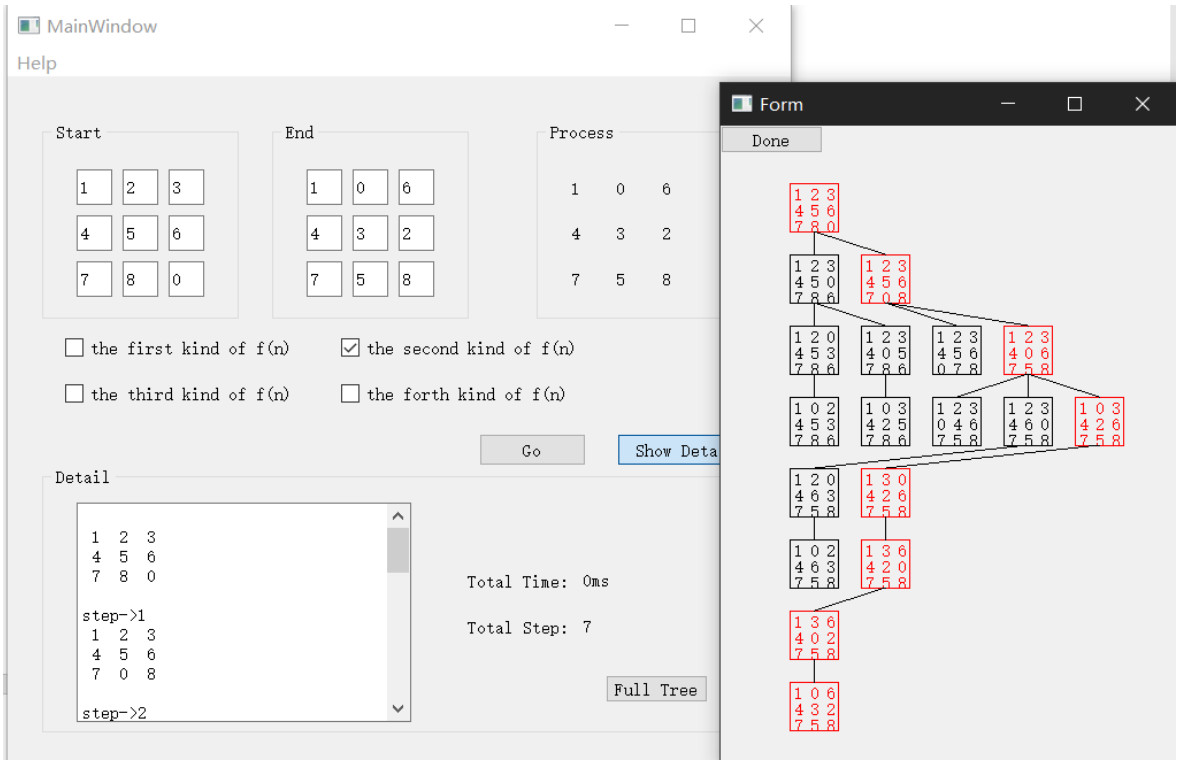

图 10 运行结果

3.3.2 效率分析

研究了 20 种不同的起始与目标状态下四种 f(n)运行时间的差异，并绘制折线图，如表 1 和图 11

批次	步骤数/步	$f_1(n)$用时/ms	$f_2(n)$用时/ms	$f_3(n)$用时/ms	$f_4(n)$用时/ms
1	15	2	2	3	1
2	15	2	5	2	4
3	16	4	3	6	9
4	18	9	17	8	9
5	18	8	8	9	8
6	19	16	15	14	24
7	20	18	19	29	17
8	20	17	20	25	17
9	21	27	30	27	27
10	22	55	53	55	53
11	22	53	53	57	56

12	22	47	40	48	42
13	22	62	56	56	56
14	23	78	76	77	76
15	23	59	56	56	58
16	24	104	96	95	95
17	24	142	154	154	135
18	26	353	346	342	341
19	27	432	405	413	463
20	27	441	431	434	461
和		1929	1885	1910	1952

表 1 运行时间对比图

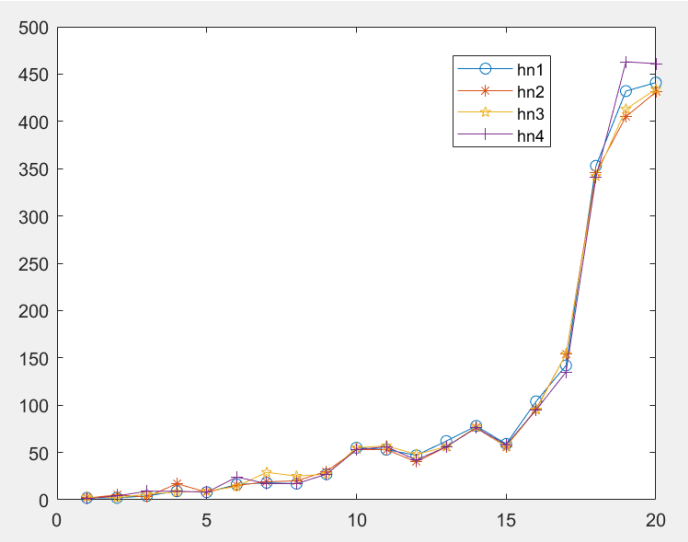

图 11 四种 f(n)运行时间对比折线图

对于不同情况，有不同的最优 f(n)。但通过折线图可看出，多数情况下 hn2 折线对应的点最低，用时最少，且从表格可看出 hn2 对应的总时间最小。综合分析，使用 hn_2 运行时间较低，搜索速度最快。选择曼哈顿距离，即所有当前结点与目标结点的横纵坐标差的绝对值的和作为 h(n)效率最高，与预期相符合。

4 总结

4.1 实验总结

通过本次实验，我们对启发式搜索有了更深入的了解。为了避免不必要的扩展，定义一个好的估价函数至关重要。我们查阅了相关文献，学习了不同的估价函数，选取了四种估价函数进行比较，并发现用曼哈顿距离作为 g(n)，效率明显较高于其他几种估价函数。

为了有较好的效果呈现，我们组员特意学习使用 QT，程序界面简洁美观，功能完备，最优搜索树直观。程序界面可以完成以下功能：展示最优步骤和步数，画出完整的最优搜索树，判断问题是否有解，比较四种不同 hn 的效率。

代码实现工程中使用了 STL 容器，结构清晰，方便 open,close 的管理。

4.2 存在问题

（1）保证找到最短路径（最优解的）条件，关键在于估价函数的选取：估价值 h(n)≤ n 到目标节点的距离实际值，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。并且如果 h(n)=d(n)，即距离估计 h(n)等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。如果 估价值>实际值,搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。 故不是所有情况下都得到的最优解。

（2）不同的 hn 适用于不同的情况，本实验得到最优 h(n)（采用曼哈顿距离）只是在已有的几种情况测试中综合得到的，第三四种 h(n)更适用于可以对角线位移的情况，但在本例中表现也较为良好，故不能绝对地说明哪一种 h(n)更好。

（3）因为八数码问题规模相对较小，没有考虑空间耗费问题，可能会牺牲一定的空间

4.3 后续改进方向

（1）考虑使用双向 A*寻路算法加快搜索速度，或者通过准确有效的剪除不符合要求的状态来减少不必要的搜索。

（2）可以增加功能，随机生成可行的初始状态，比较四种 hn 的运行时间，通过概率可得到运行效率最优的 hn。

（3）利用 k- d 树空间索引结构，动态加载节点信息，减小内存使用空间。

4.4 心得体会

（1）学习使用 A*算法解决搜索问题，实现八数码问题的求解，对启发式搜索有了深一步的认识。

（2）估价函数的选取方法有了进一步的了解，能够比较不同估价函数的性能，选择适合的估价函数。

（3）学习使用 QT 的 ui 设计，编程能力有进一步提升，能够设计出美观的程序界面。

（4）使用 STL 库里的各种容器，功能齐全，算法实现时会更简洁明了，open,close 表的扩展和管理也更方便。

参考文献

[1] Stuart J. Russell，Peter Norvig. 世界计算机教材精选 人工智能 一种现代的方法 第 3 版. 北京：清华大学出版社, 2018.07.

[2] 付宏杰,王雪莹,周健,周孙静,朱珠,张俊余.八数码问题解法效率比较及改进研究[J].软件导刊,2016,15(09):41-45.

[3] 周浩.八数码问题 DFS 和 BFS 算法的设计与实现[J].电脑知识与技术,2011,7(22):5487-5489.

[4] https://www.bilibili.com/video/BV1Hx411X7QB

装 订 线