

本 科 毕 业 设 计（论 文）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学院(部) | 计算机科学与技术学院 | | | |
| 题 目 | 基于A\*算法的问题求解方法研究与实现 | | | |
| 年 级 | 2018级 | | 专业 | 计算机科学与技术 |
| 班 级 | 18计科2班 | | 学号 | 1827405075 |
| 姓 名 | 罗峰 | | | |
| 指导老师 | 刘全 | | 职称 | 教授 |
| 论文提交日期 | | 2022.5 | | |

苏州大学

本科毕业设计（论文）独创性声明

**本人郑重声明：所提交的本科毕业设计（论文）是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本设计（论文）不含其他个人或集体已经发表或撰写过的研究成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。**

**作者签名：  日 期： 2022.5.4**

苏州大学

本科毕业设计（论文）使用授权声明

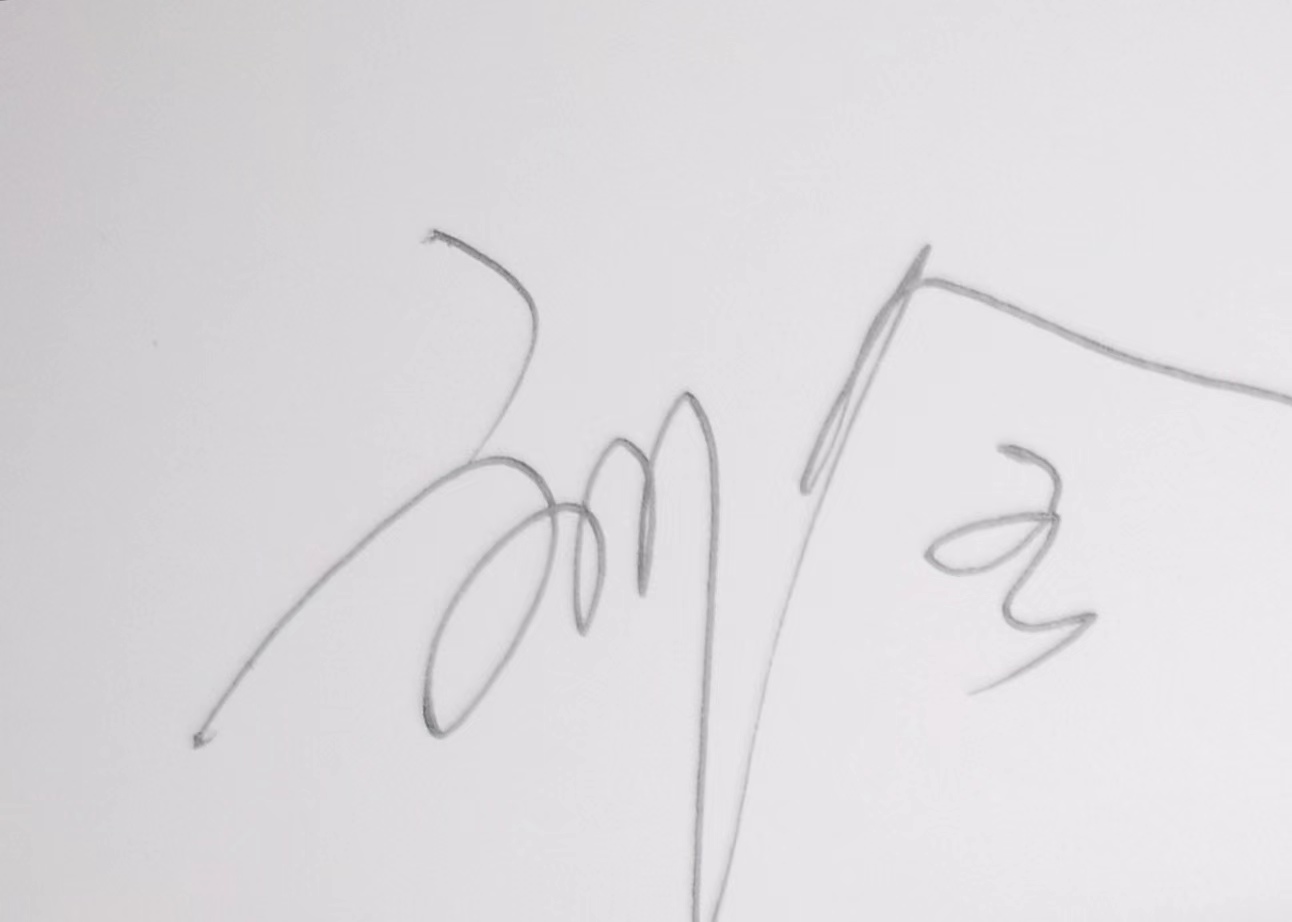
**本人完全了解苏州大学关于收集、保存和使用本科毕业设计（论文）的规定，即：本科毕业设计（论文）的著作权以及文中研究成果的知识产权归属苏州大学。苏州大学有权向国家有关部门或第三方机构送交毕业设计（论文）的复印件和电子文档，允许毕业设计（论文）被查阅和借阅，可以采用影印、缩印或其他复制手段保存和汇编毕业设计（论文），可以将毕业设计（论文）的全部或部分内容编入有关数据库进行检索。**

**涉密设计（论文）□**

**本设计（论文）属 在 年 月解密后适用本规定。**

**非涉密设计（论文）**

**论文作者签名：  日 期： 2022.5.15**

**导师签名  日 期： 2022.5.15**

# 摘要

搜索算法是日常生活中应用最广泛的算法之一，而A\*(A-Star)搜索算法则是搜索算法中最有影响力的启发式算法。A\*搜索算法能够有效的在计算机静态路由网络中解决最短路径问题,是一种被人推崇备至的简单启发式搜索方法。

本课题主要以A\*搜索算法为研究对象，总结A\*搜索算法的实现原理，阐明A\*搜索算法的实现流程，用A\*搜索算法解决现实中常见的问题，对比和其他盲目搜索方式的异同，以及介绍A\*搜索算法在现实中的最新应用。

**关键词**；A-Star算法；状态空间；估价函数；启发式算法

# Abstract

The search algorithm is one of the most widely used algorithms in daily life, and the A\*(A-Star) search algorithm is the most influential heuristic algorithm in the search algorithm. A\* search algorithm is a method that can effectively solve the shortest path problem in computer static routing network, and it is also a simple heuristic search method that is highly praised in artificial intelligence algorithms.

This topic mainly studies the research and implementation of the problem solving method based on the A\* search algorithm, summarizes the implementation principle of the A\* search algorithm, clarifies the implementation process of the A\* search algorithm, and uses the A\* search algorithm to solve common problems in reality. The similarities and differences of other blind search methods, and the latest application of A\* search algorithm in reality.

Keywords: A-Star algorithm; state space; evaluation function; heuristic algorithm

# 目录

[摘要 i](#_Toc103715477)

[Abstract ii](#_Toc103715478)

[第1章 绪论 - 1 -](#_Toc103715479)

[1.1 研究背景与意义 - 1 -](#_Toc103715480)

[1.2 A\*搜索算法的发展 - 1 -](#_Toc103715481)

[1.3 本文的主要内容和各章节安排 - 2 -](#_Toc103715482)

[第2章 A\*搜索算法的实现原理 - 4 -](#_Toc103715483)

[2.1搜索常用名词解释 - 4 -](#_Toc103715484)

[2.1.1 搜索 - 4 -](#_Toc103715485)

[2.1.2 状态空间知识表示法 - 4 -](#_Toc103715486)

[2.1.3 回溯策略 - 4 -](#_Toc103715487)

[2.2 A\*搜索算法概述 - 4 -](#_Toc103715488)

[2.3 A\*搜索算法原理 - 5 -](#_Toc103715489)

[2.3.1 估价函数 - 5 -](#_Toc103715490)

[2.3.2 A\*搜索算法的估价函数优化 - 5 -](#_Toc103715491)

[2.3.3 A\*搜索算法的可采纳性证明 - 6 -](#_Toc103715492)

[2.4 本章小结 - 7 -](#_Toc103715493)

[第3章 A\*搜索算法的实现和问题求解 - 8 -](#_Toc103715494)

[3.1 A\*搜索算法的实现 - 8 -](#_Toc103715495)

[3.2 A\*搜索算法具体应用求解 - 9 -](#_Toc103715496)

[3.2.1 八数码问题求解 - 9 -](#_Toc103715497)

[3.2.2 迷宫寻路问题求解 - 12 -](#_Toc103715498)

[3.3 本章小结 - 14 -](#_Toc103715499)

[第4章 估价函数对A\*搜索算法的影响 - 15 -](#_Toc103715500)

[4.1 不同估价函数下的搜索效率 - 15 -](#_Toc103715501)

[4.1.1常数作为估价函数 - 15 -](#_Toc103715502)

[4.1.2欧式距离作为估价函数 - 16 -](#_Toc103715503)

[4.1.3曼哈顿距离作为估价函数 - 16 -](#_Toc103715504)

[4.1.4三种估价函数随机实验结果 - 17 -](#_Toc103715505)

[4.2 分析估价函数对搜索效率的影响 - 18 -](#_Toc103715506)

[4.3 本章小结 - 18 -](#_Toc103715507)

[第5章 A\*搜索算法和其他搜索的对比 - 19 -](#_Toc103715508)

[5.1 与宽度优先搜索的对比 - 19 -](#_Toc103715509)

[5.2 与深度优先搜索的对比 - 19 -](#_Toc103715510)

[5.3 与Dijkstra算法的对比 - 20 -](#_Toc103715511)

[5.4 本章小结 - 21 -](#_Toc103715512)

[第6章总结和展望 - 22 -](#_Toc103715513)

[6.1 总结 - 22 -](#_Toc103715514)

[6.2 展望 - 22 -](#_Toc103715515)

[参考文献 - 23 -](#_Toc103715516)

[致谢 - 25 -](#_Toc103715517)

# 第1章 绪论

## 1.1 研究背景与意义

随着计算机科学与技术的迅速发展和应用，互联网领域产生的数据量也越来越大，这不光是对硬件的新考验，也是对各种创新的算法的新考验。在数据量爆炸的时代，如果没有一个更优秀，复杂度更低的算法，将很难应对暴涨的数据，最终导致应用瘫痪。因此，许多AI算法应运而生，通过启发式的方式，大大提高了算法的运行效率。和如今的硬件配合，从准确度和速度两个方面都远远超过了朴素的搜索算法。其中，A\*搜索算法则是寻路算法中的兼顾了效率和准确率的算法。

搜索算法通常会在一个由若干个搜索项的集合中，查找某一个具有特别属性的项。通常搜索算法会利用计算机的高性能来有目的的穷举一个问题解空间的部分或所有的可能情况，从而求出问题的解[1]。搜索算法分为两种，一种是盲目搜索，另一种是会用到解决问题相关领域的知识的启发式搜索。盲目搜索不具有问题相关的信息，没有问题解决的相关背景，只是按照预先设定好的程序进行的搜索，主要包括宽度优先搜索（BFS），深度优先搜索（DFS）等等；启发式搜索则是考虑问题相关的特定知识，动态地确定调用操作算子的步骤，优先选择较适合的操作算子，主要包括A及A\*搜索算法[2]。

和普通的盲目搜索相比，A\*搜索算法可以在保证最优性的前提下，更快的找到解。提供高效的寻路效率[3]，从而在多个方面支持民生。其中，最主要的应用场景便是导航等寻路问题。通过A\*搜索算法，快速获得一条可行的最优解路径，增加资源的有效使用，提高工作效率，提高出行智能化水平[4]。

在如今新型冠状病毒肺炎疫情没有好转，仍旧反复的大环境下，A\*搜索算法还可以解决抽象化的资源分配问题，帮助社区和政府更好的调度资源，保障民生，杜绝送菜难等问题。

## 1.2 A\*搜索算法的发展

目前的经常被使用的搜索算法非常多，包括广度优先搜索（BFS），深度优先搜索（DFS），爬山法，回溯法，分支界限算法等等。

A\*搜索算法是一种启发式搜索算法，也影响力最大的启发式搜索算法[5]。在常见的搜索问题实践中，状态空间的大小都是巨大的。例如围棋，象棋等棋类游戏，棋局的总状态数都是非常惊人的，最高甚至达到了*10761*的量级。这对于盲目搜索来说，是无法企及的复杂度。哪怕是用全人类的所有计算机去算，也要至少1亿亿年才能搜索完成。启发式搜索相较于盲目搜索，增加了启发信息和估价函数，可以大大降低状态空间的大小，提高搜索效率，在有限的计算次数内获得更优的解[6]。

1964年Nils Nilsson发明了一种基于启发式的方法来提高Dijkstra算法[7]的速度。这个算法被称为A1算法。1967年，Bertram Raphael设计了A2算法，提高了搜索效率，但是仍然不能保证获得一条最短路径。之后在1968年，PeterE. Hart增加了A2算法的约束条件，并且证明了在这样的条件下，该算法在进行启发式搜索时，一定可以得到一条最短路径。这种增加约束的搜索算法，后来被称为A\*搜索算法。

A\*搜索算法由A搜索算法改进而来，它要求在搜索的每一步中都满足当前状态到最终状态的预估最小步数小于或等于当前状态到最终状态的实际最小步数，以此保证了解的唯一存在。同时，当前状态到最终状态的最小步数的预估和实际越接近，A\*算法就能越快速的找到最终结果[8]。

目前，A\*搜索算法已经运用在多种实践中，尤其常见于游戏中的NPC的移动计算，或线上游戏的BOT的移动计算上[9]，在棋类游戏和导航的路径规划中，也常常能见到A\*搜索算法的影子。

## 1.3 本文的主要内容和各章节安排

本文主要研究基于A\*算法的问题求解方法的研究与实现，主要涉及A\*搜索算法的原理剖析，解决实践和搜索对比等方面。

第1章介绍了A\*搜索算法的研究背景和论文主要内容。

第2章剖析A\*搜索算法的实现原理，解释A\*搜索算法如何在保证找到最优解的情况下，提高搜索效率。

第3章说明A\*搜索算法在具体问题中的求解思路和实现过程，求解问题主要以八数码和迷宫搜索两个问题为例。

第4章主要研究影响A\*算法搜索效率的因素。

第5章比较不同搜索算法的准确率和效率，证明A\*搜索算法的高效率和可采纳性。

第 6 章对本课题进行了总结，并且针对一些目前存在的问题和可以优化的部分做出了展望。

# 第2章 A\*搜索算法的实现原理

## 2.1搜索常用名词解释

### 2.1.1 搜索

搜索是一种在项集合中查找具有指定属性的项的算法。在搜索中通常需要解决是否能够找到一个确定解，是否终止运行，找到的解是否为最优解，搜索过程的世界和空间复杂度等基本问题。搜索的方向有正向搜索，逆向搜索和双向搜索。

搜索算法分为两种，一种是没有问题相关信息的盲目搜索，另一种是会用到解决问题相关领域的知识的启发式搜索。盲目搜索会按既定的程序进行搜索。启发式搜索则会将特定领域的知识纳入考量，动态的选择下一步的搜索方向[10]。

### 2.1.2 状态空间知识表示法

状态空间[11]是利用状态变量和操作符号，表示一个问题或是系统的符号体系，状态空间是一个四元组*（S,O,S0,G）*。

状态空间最终的解，是一条从起始节点到最终节点到路径。状态空间会给出若干次操作，从初始状态按照给出操作不断执行，最终就能变换称为最终状态，也就以最短路径到达终点。

### 2.1.3 回溯策略

使用回溯策略的搜索会从初始节点出发，不断的试探着寻找路径，如果遇到不可解的节点，就回溯到路径中最近的父节点上，判断该节点是否还有其他子节点需要拓展。如果有其他的子节点，则沿该节点进行拓展，直到找到目标解，或者遍历完全部节点。对于找到目标解的情况，可以直接退出搜索，返回路径。

## 2.2 A\*搜索算法概述

A\*搜索算法是一种启发式的AI搜索算法[12]，相较于朴素的搜索算法，搜索效率有了很大的提升。

在朴素的搜索算法中，每次选取搜索节点时，通常只考虑了从起始节点到当前节点距离，并以此为根据判断是否拓展当前的节点和如何选择下一个节点，多次进行这一步骤，持续进行搜索，找到终点[13]。这种搜索方式可以保证按照某些预期进行，同时实现了某些功能上的特性。这种搜索方式也被称之为盲目搜索。目前常用的宽度优先搜索（BFS），深度优先搜索（DFS），分支界限搜索等，都是基于这种盲目的搜索思想。但是随着数据量的增大，搜索空间也以指数级在扩展，导致盲目搜索策略的搜索效率逐渐无法满足要求。为此，探索新的搜索策略变的至关重要。

A\*搜索算法应运而生。除了考虑起始节点到当前节点的距离，还额外考虑了与该问题有关的信息，以此来简化了搜索过程。A\*搜索算法考虑的附加信息是计算的当前节点到目标节点的距离、计算的当前节点到目标节点的距离和当前节点经过的距离，动态选取下一个搜索的节点。需要注意的是，当前节点到目标节点的预估值必须要满足小于当前节点到目标节点的实际值。具体原因详见下文。

## 2.3 A\*搜索算法原理

### 2.3.1 估价函数

在搜索可行的状态空间时，A搜索算法会利用预先设计好的公式来计算每一个节点距离目标节点的期望值。每一次扩展时，这个公式将决定哪一个节点在最短路径上的可能性最大，从而决定从哪个节点开始搜索新节点。因此，估价函数和实际路程之间的差值，决定了算法的实际搜索效率[14]。估价函数的表达式如下：

*f(n)=G(n)+h(n)*

公式中的*f(n)*是指从起点到终点通过节点*n*的最短路径的估计值。*G(n)* 表示当前节点已经行进的实际距离，在搜索中通常是确定的值。*h(n)*代表着节点*n*和终点*G*之间的预估距离，也是整个估价函数的核心所在。根据公式，可以得到*h(n)*的设计影响了整个搜索过程，A\*搜索算法中的启示性也都是集中在了*h(n)*上。而*G(n)*不影响算法的启发性，主要是记录已经走过的距离对搜索状态的影响。

### 2.3.2 A\*搜索算法的估价函数优化

A\*搜索算法本质上也是一种最好优先策略，但是通过一些限制条件，优化了搜索流程。随着信息时代的到来，搜索算法也会面临搜索空间不断膨胀的问题，A\*搜素算法能够兼顾效率和准确性，更快速的找到需要的最短路径。

搜索算法的可采纳性简单说就是指如果存在从起点到终点的路径，该算法一定可以找到一条最短的路径。A\*搜索算法就是通过某些约束条件，保证了算法一定会输出最短路径，满足可采纳性。估价函数的表达式如下：

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

公式中的*f\*(n)*是指从起点到终点通过节点*n*的最短路径的估计值。 *g\*(n)* 表示当前节点已经行进的实际距离，在搜索中通常是确定的值。*h\*(n)*代表着节点*n*和终点*G*之间的预估距离。*f\*(n)*的值在实际问题中不可预知，因此常常直接用*f(n)*的值替代。在进行搜索时，*g\*(n)*的值往往可以确定，*g\*(n)*的值和实际的*g(n)*的值相等或略小，因此可以直接使用*g(n)*的值。A\*搜索算法的额外约束，要求*h\*(n)*的值小于或等于实际的距离，这个约束在A搜索算法中不存在。

在2.3.3章节，会证明优化后的评价函数可以可靠地找到最短路径。

在搜索中，每次取出下一个进行搜索的节点时，会计算每个节点的*F*值，确定“最佳节点”。每次基于“最佳节点”进行搜索，便可以快速的启发式的实现寻址,求解最短路径[15]。

### 2.3.3 A\*搜索算法的可采纳性证明

搜索算法的可采纳性简单说就是指如果存在从起点*S0*到终点*G*的路径，该算法一定可以在有限步的操作中找到一条最短的路径。以下将通过两个辅助定理，证明A\*算法的可采纳性。

定理1：在一张有限图中，只要起点和终点之间存在任意一条可行路径，那么A\*搜索算法一定在有限步中输出找出这条路径。如果起点和终点之间不存在任何路径，A\*搜索算法也会在有限步中结束程序。

证明：由于搜索是在一张有限图上进行的，首先考虑起点和终点之间存在路径的情况。在起点和终点之间存在路径的情况下，A\*算法会扩展每一个节点的全部可拓展节点。因此，起点和终点之间路径的任何一个点都会被加入搜索列表中，最终一定会找到某条可行路径，退出程序。其次当起点和终点之间不存在路径时，A\*算法会访问所有节点，当发现没有可以向外拓展的节点时，退出程序。

定理2:在一张无限图中，只要起点和终点之间存在任意一条可行路径，那么A\*搜索算法一定在有限步中输出找出这条路径。

证明：在一张无限图中，如果起点和终点之间的路径不存在，那么A\*算法拓展的节点一定有任意大的*f\*(n)*值，如果路径存在，那么任意一个节点的*f\*(n)*值都会小于或等于*f\*(S0)*。因此，如果A\*算法不能成功结束，那么一定会产生任意大的*f\*(n)*值，这和路径存在的情况产生冲突。因此，A\*算法在无限图上也能成功结束。

根据前两个定理的证明，无论是有限图还是无限图，只要起点和终点之间的路径存在，那么A\*搜索算法都能找到一条从起点到终点一条可行路径并成功结束。假设A\*算法没有终止在最佳路径上，而是终止在了某个节点*t*上，那么就会有*f(t) = G(t) > f\*(S)*。但是在A\*搜索算法结束前，必然存在最佳路径上的点*n*，使得*f(n) <= f\*(S)<f(t)。*此时，A\*算法会选择点*n*进行拓展，而后拓展点*t。*因此产生矛盾。说明A\*搜索算法就是可采纳的。

## 2.4 本章小结

本章阐述了A\*搜索算法的内容，详细说明了A\*搜索算法的实现原理，并且论证了A\*搜索算法的可采纳性。

# 第3章 A\*搜索算法的实现和问题求解

## 3.1 A\*搜索算法的实现

A\*搜索算法的具体实现流程可以描述为如下几个步骤：

步骤一：对搜索流程进行初始化，把起点*S0*加入*OPEN*表中。

步骤二：如果*OPEN*没有可以取出的节点，证明当前所有可拓展的节点都已经搜索完毕，没有找到路径，直接退出。

步骤三：将*OPEN*表中的估价函数计算值最小的节点*n*取出，加入*CLOSE*表。

步骤四：如果节点*n*是终点，直接返回结果。

步骤五：判断节点*n*是否有可拓展的节点，如果没有可拓展的节点，则重新执行第二步。

步骤六：拓展节点*n*，为拓展的每一个节点计算*f(x)*的值，紧接着做如下操作。

判断在这些扩展的节点中，是否有*OPEN*表或者*CLOSE*表的元素。如果有，则说明这些节点第二次被搜索到，需要判断是否要更新他们的*f(x)*的值，或者是修改他们的父节点。如果当前新计算的*f(x)*的值更优，则代表通过当前节点拓展的路径更优，因此需要更新这些元素。

对于不在*OPEN*表或者是*CLOSE*表中的其余拓展节点，将其父节点设置为节点*n*之后加入*OPEN*表即可。之后根据*OPEN*表中每个节点*f(x)*的值，维持*OPEN*表的单调性，转步骤二。

其主要流程可以用流程图表示。图3-1为A\*搜索算法的实现流程图。

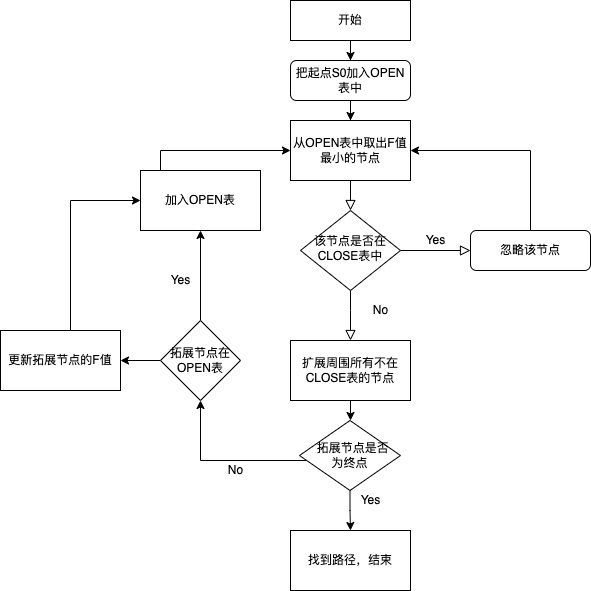


图3-1 A\*搜索算法实现流程图

## 3.2 A\*搜索算法具体应用求解

本段落选取两个实际的算法问题，来介绍A\*搜索算法是如何根据实际问题选取合适的搜索策略，完成搜索流程。主要包括八数码问题和迷宫寻路两个问题。

### 3.2.1 八数码问题求解

3.2.1.1问题描述

八数码问题是指在一个3\*3的矩阵中，随机的填入1-8这八个数字，留下一个空格（可以用*0*代替），作为初始状态；再取一个3\*3的矩阵中，随机的填入1-8这八个数字，留下一个空格（可以用0代替），作为目标状态。接着对初始状态进行操作，每次可以将空格和周边的任意一个数字做交换。不断的操作初始的举证，直到达到目标状态[16]。这样的求解过程就是八数码问题。如图3-2。

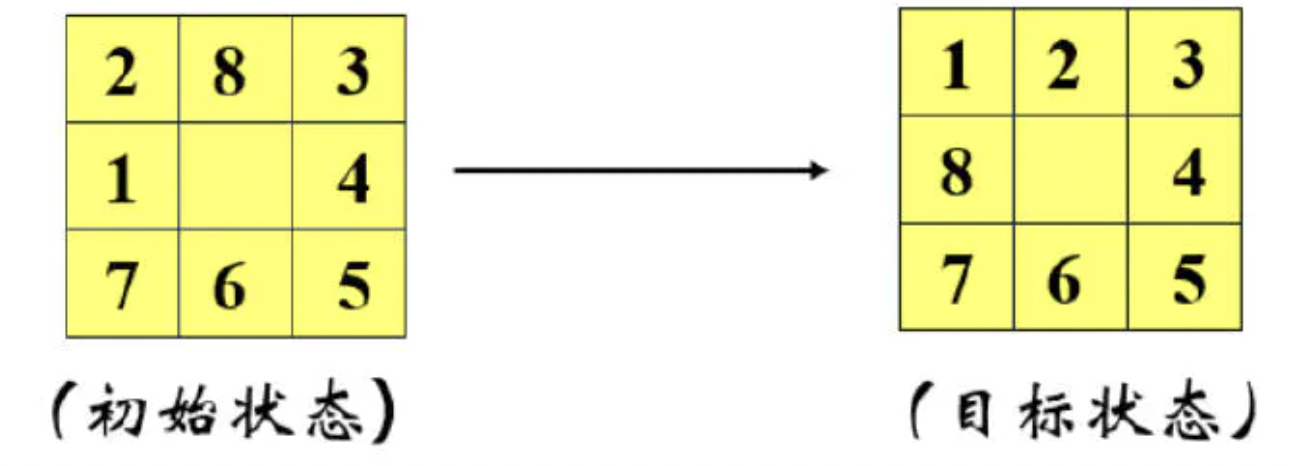


图3-2 八数码问题示意图

3.2.1.2问题分析

首先要解决问题是否可解。

八数码的每个状态可以抽象为一个序列，那么每一次操作都可以转换为序列上的一次交换操作。容易发现，每次交换操作不会改变这个序列逆序对数量的奇偶性。因此，可以简单的通过初试状态和目标状态的逆序对数量的奇偶性来判断是否可解。

其次要设计表示该序列的状态。

由于该问题的状态比较简单，状态空间的大小也比较小，所以可以直接记录序列的顺序作为单个状态。

最后设计针对该问题的估价函数。

对于普通的搜索来说，该问题只需要遍历所有的情况，找到最短的解决路径即可。这个过程由于需要遍历，所以相对还是比较复杂，因此需要设计合理的估价函数。对于该问题，可以以当前状态的数码的坐标差值，作为当前的估价函数，以此来减少搜索的复杂度。估价函数如图3-3

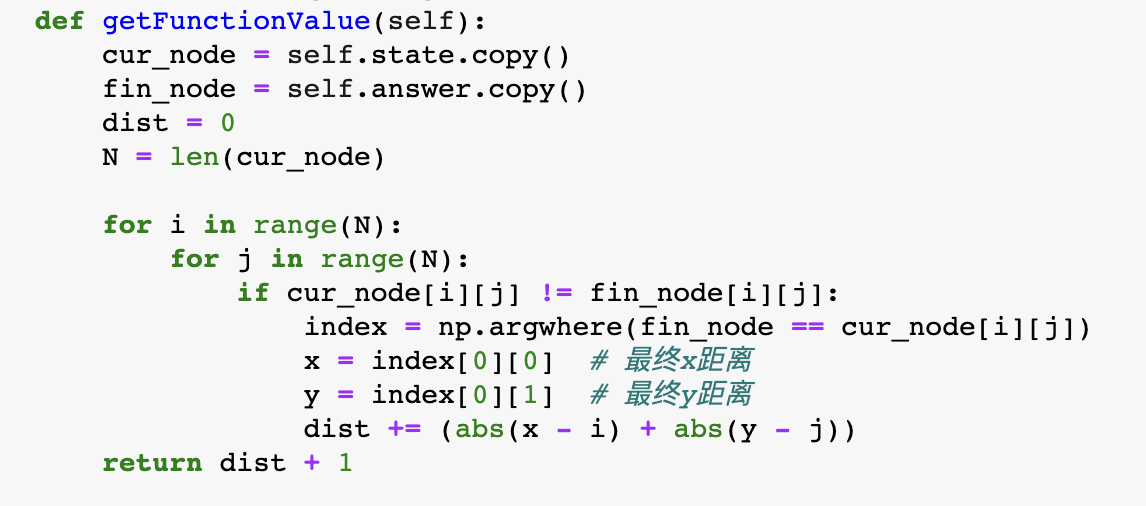


图3-3 八数码问题估价函数

3.2.1.3问题解决

按照上文给出的估价函数，设计代码解决即可。具体代码可以参照附录。图3-4给出了代码关键部分的实现。



图3-4 八数码问题解决函数

### 3.2.2 迷宫寻路问题求解

3.2.2.1问题描述

迷宫寻路问题会给出一个输入迷宫，其中“#”代表迷宫边界和障碍物，“S”代表起点，“E”代表终点，“.”代表迷宫的一般路径。该问题需要你计算从起点到终点的最短距离，并且输出一条从起点到终点的最短路径。迷宫示例见图3-5。

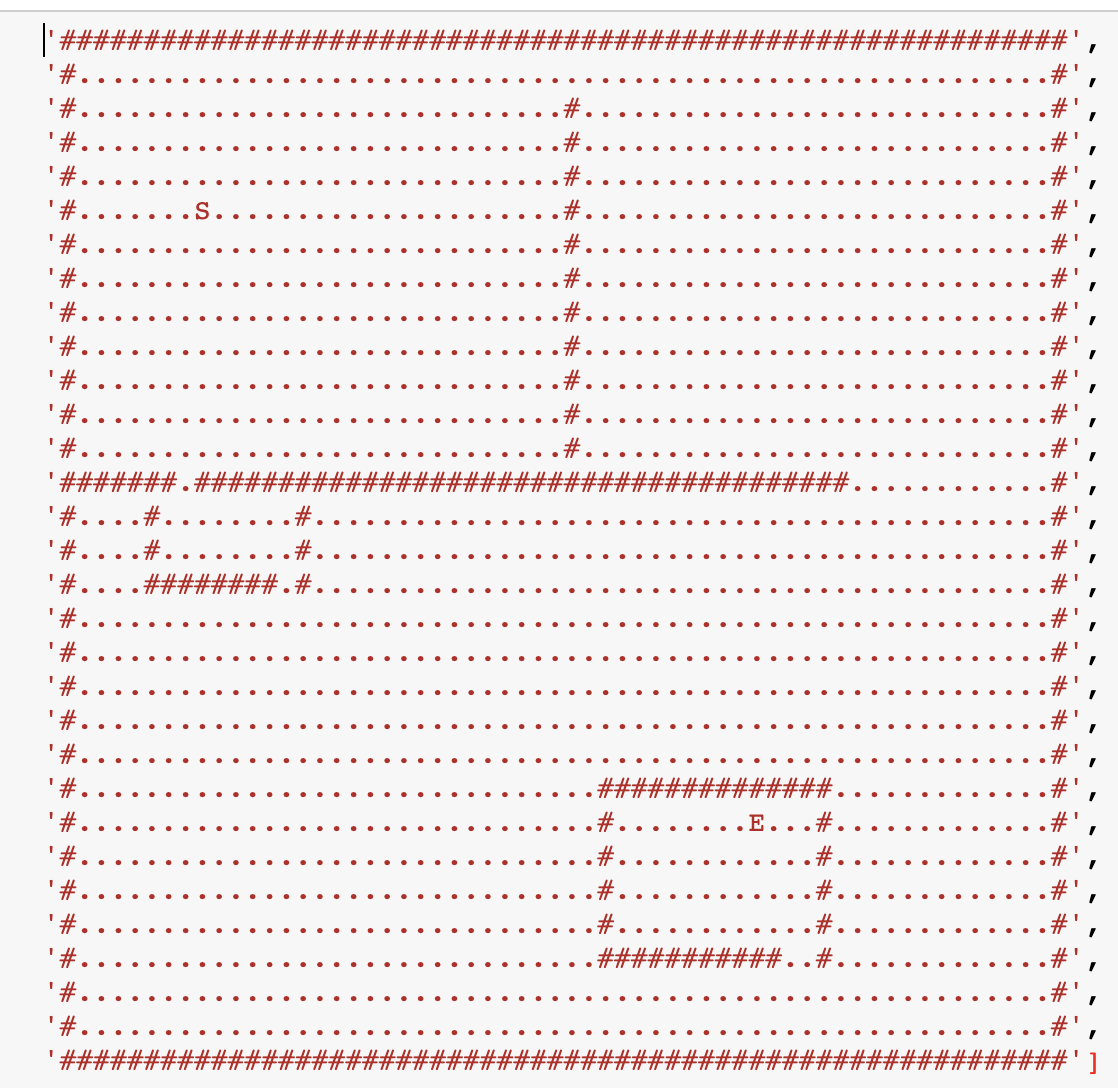


图3-5 迷宫问题迷宫示例图

3.2.2.2问题分析

对于迷宫问题，是否可解十分容易判断，只需要查看最终有没有生成一条到终点的路径即可。因此，在进行搜索时，只需要查看是否已经遍历完整个图像，便可以知道是否有一条从起点到终点的可行路径。

对于寻路问题，需要记录当前寻找到节点的状态，主要包括节点的位置，当前的估价信息和转移的节点。相对来说，单个节点的状态要比八数码问题小很多。

对于该问题的估价函数，要充分考虑到该搜索问题的特点，根据搜索的方式不同，确定合适的估价函数。对于当前问题。估价函数应该与起点到终点的距离有关，我选择使用当前点和终点的坐标之差作为估价函数。这里之所以不选择欧式距离作为估价函数，可以参照第四章的分析。

估价函数可以参照图3-6。

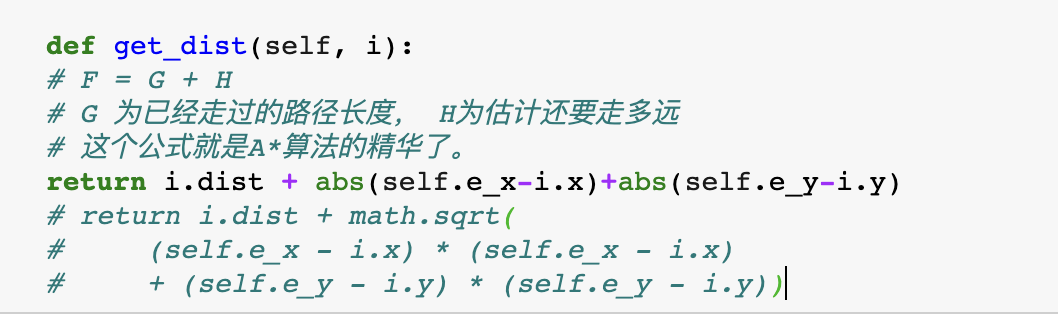


图3-6 迷宫问题估价函数

3.2.2.3问题解决

按照上文给出的估价函数，设计代码解决即可。具体代码可以参照附录。图3-7给出了代码关键部分的实现。

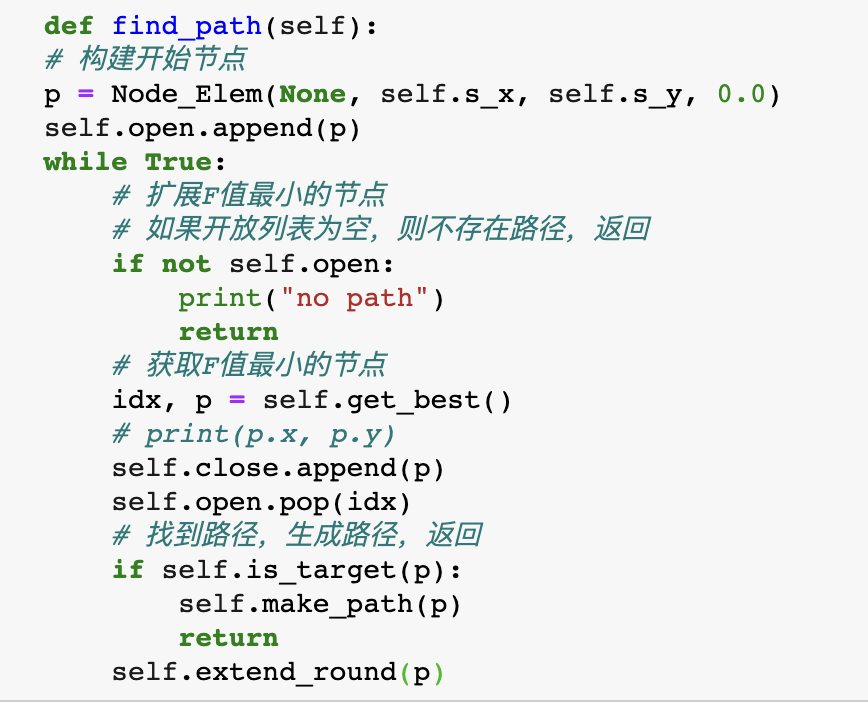


图3-7 迷宫问题解决函数

根据上述函数对图3-5所述的迷宫问题进行求解，可以求得如图3-8代表的最短路径。注意“o”代表已经搜索的节点。

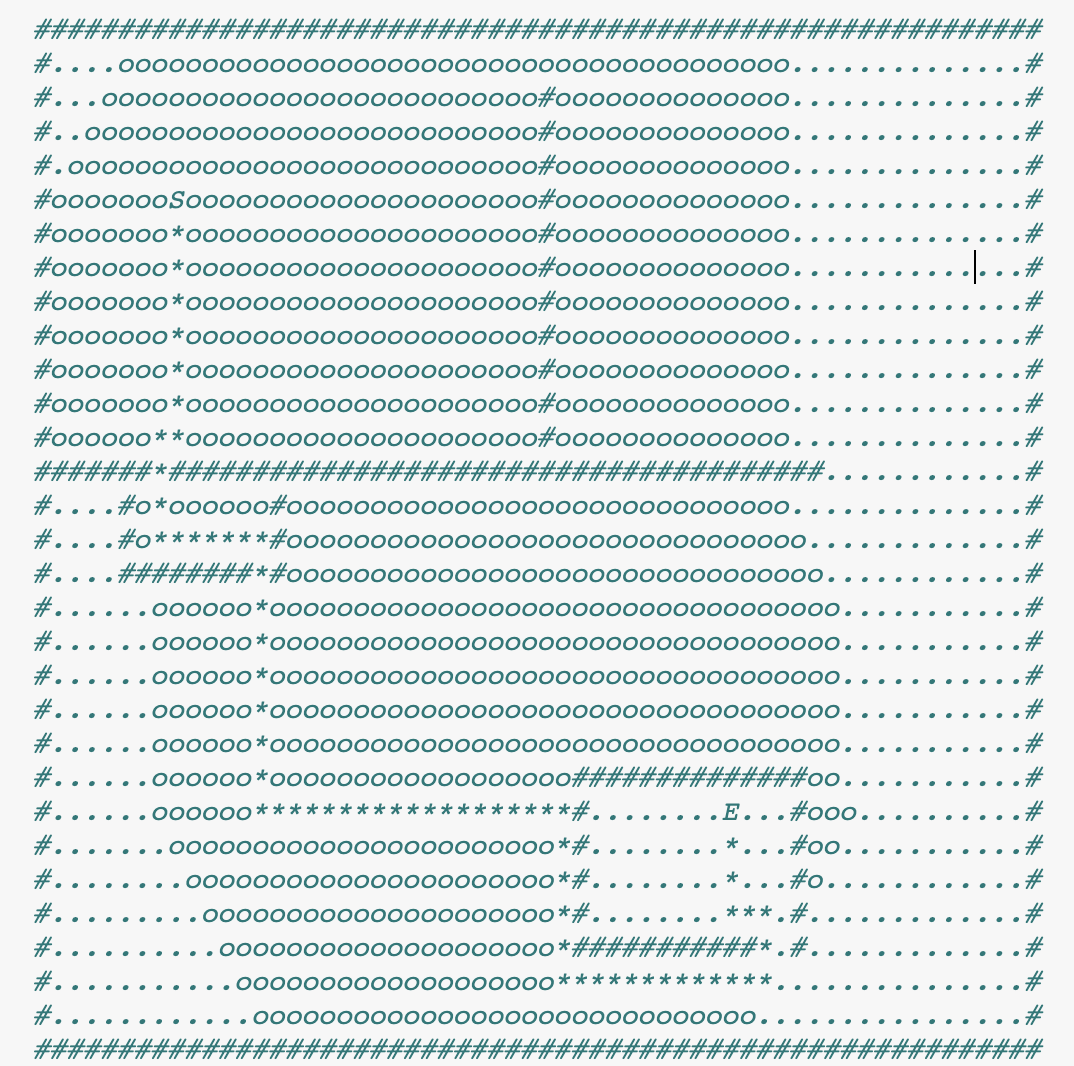


图3-8 迷宫问题最短路径

## 3.3 本章小结

本章针对两个实际问题，对A\*搜索算法如何求解问题进行了探究。成功解决了八数码和迷宫寻路两个问题。

# 第4章 估价函数对A\*搜索算法的影响

## 4.1 不同估价函数下的搜索效率

对于同一个问题，估价函数就是影响A\*搜索算法的关键所在，不同的估价函数往往会带来不同的结果。根据A\*搜索算法的原理，估价函数越接近实际的花费，A\*搜索算法的效率就会越高。但是在实际的使用中，很难确定一个更优的估价函数。本节以迷宫搜索为例，展示不同估价函数对搜索效率的影响。

用于测试的迷宫除了图3-6的迷宫外，还额外添加了一张没有障碍物的图作为对比。如图4-1。

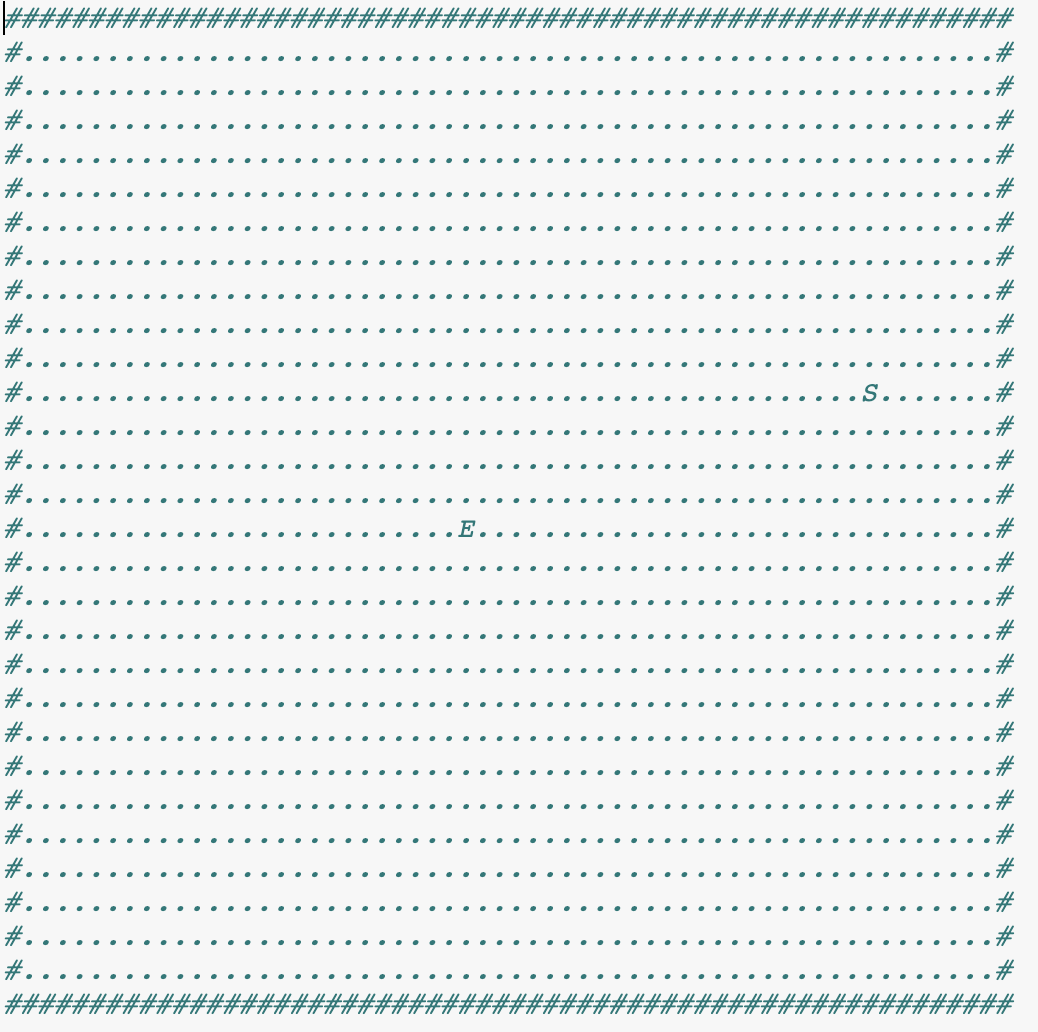


图4-1 迷宫问题无障碍图

### 4.1.1常数作为估价函数

在选择常数估价函数后，实际的A\*搜索算法将和宽度优先搜索类似。通过以常数作为估价函数，可以查看在没有估价函数影响下的搜索效率。

对两张图的测试结果分别如图4-2，图4-3。

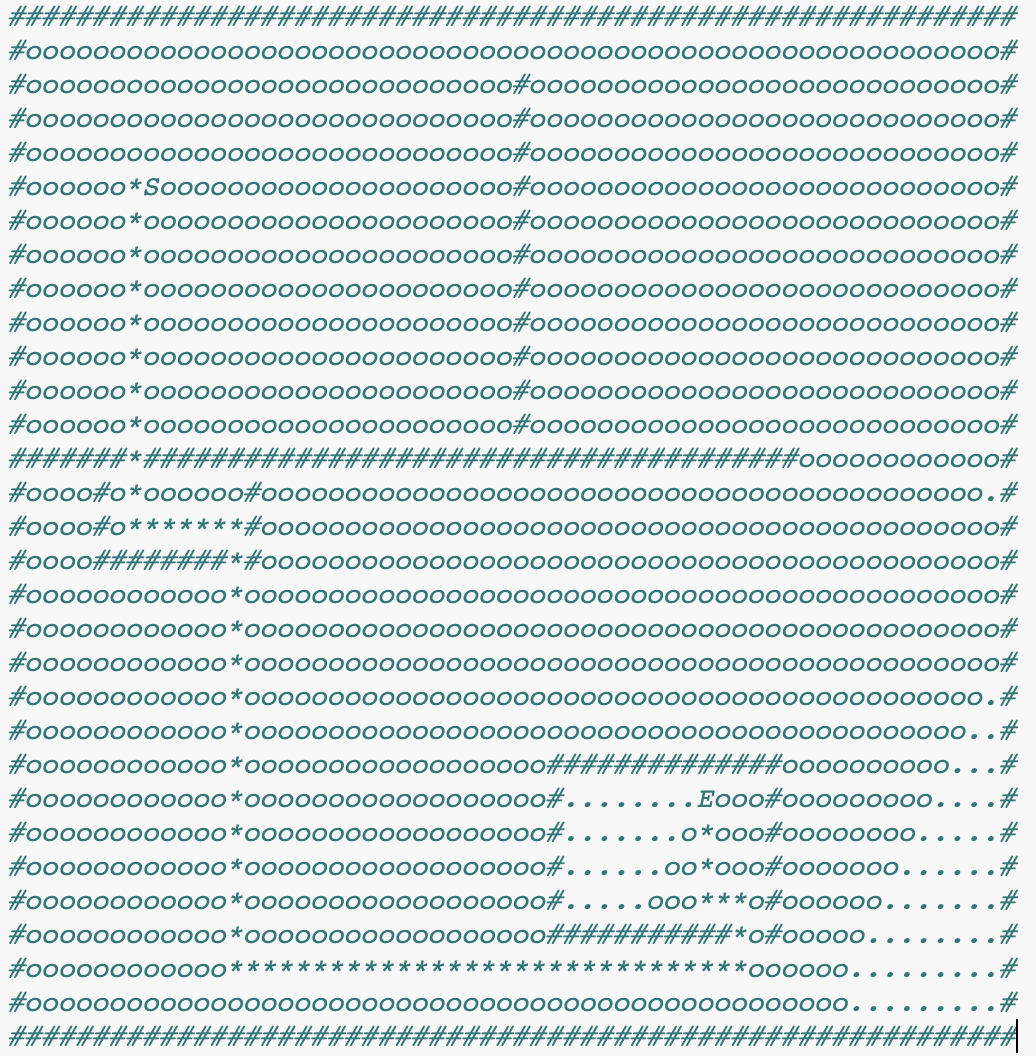
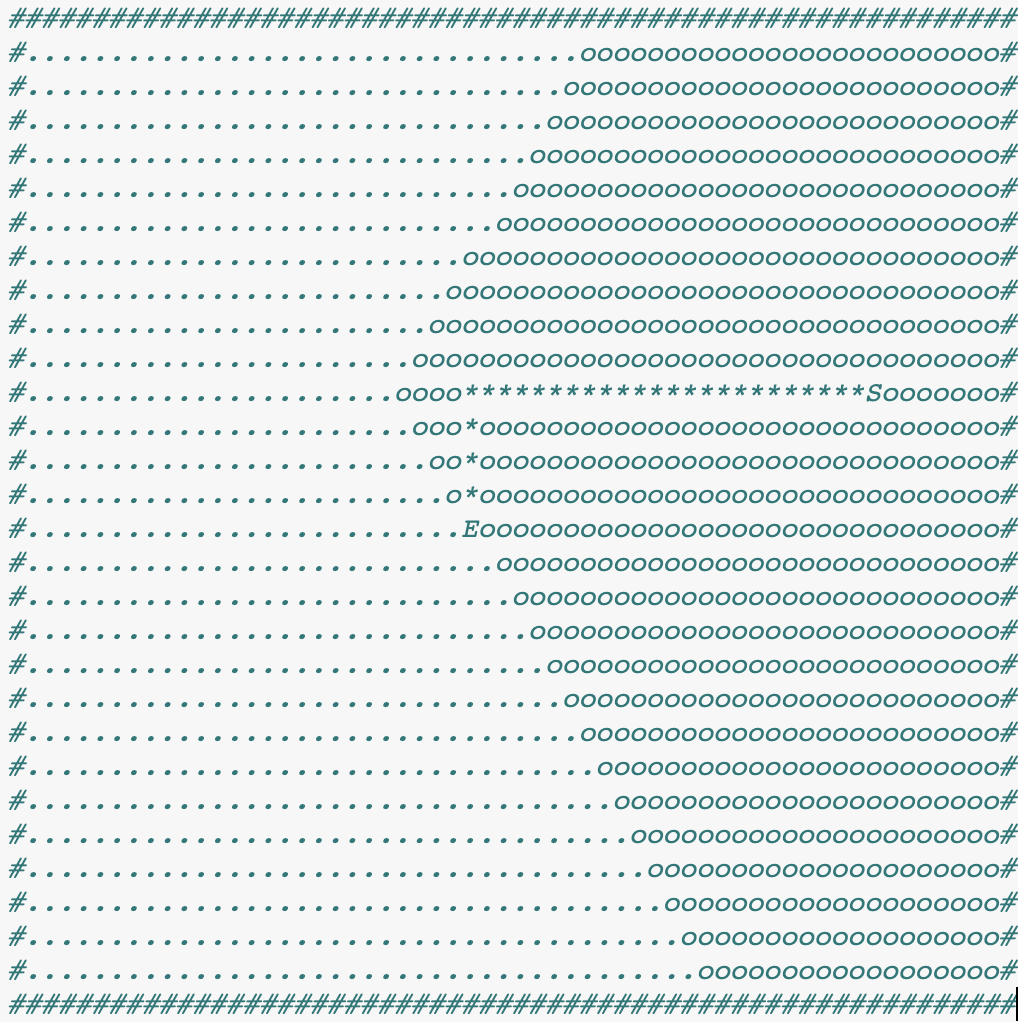
 

图4-2 常数估价有障碍图 图4-3 常数估价无障碍图

可以看出，在没有估价函数加持下的A\*算法基本等同于宽度优先搜索，搜索效率很低。但是要注意，由于已经搜索的点的已有距离也会产生影响，因此此时的搜索不完全等价于宽度优先搜索。

### 4.1.2欧式距离作为估价函数

欧式距离是路径搜索问题的最常用的估价函数，图4-4和通4-5分别展示了该估价函数在不同迷宫的搜索效率。

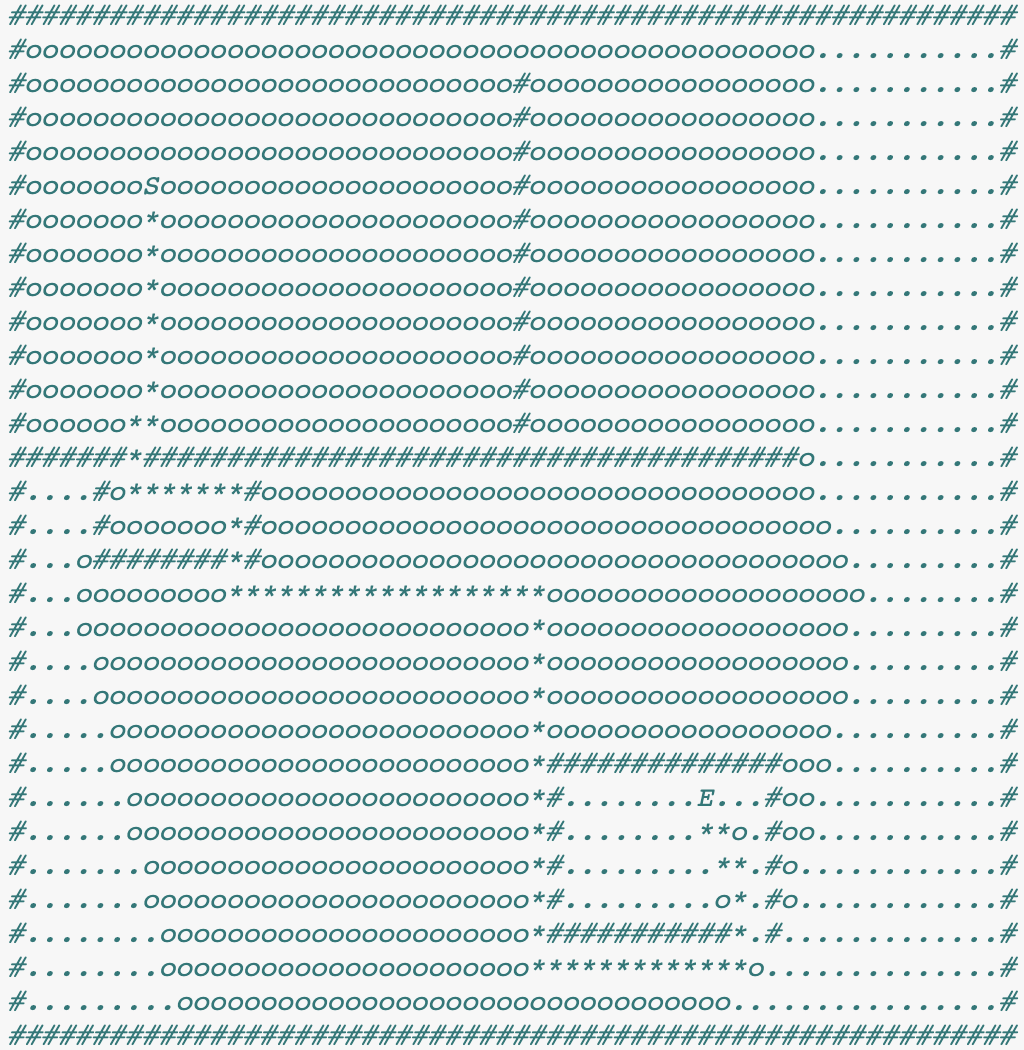
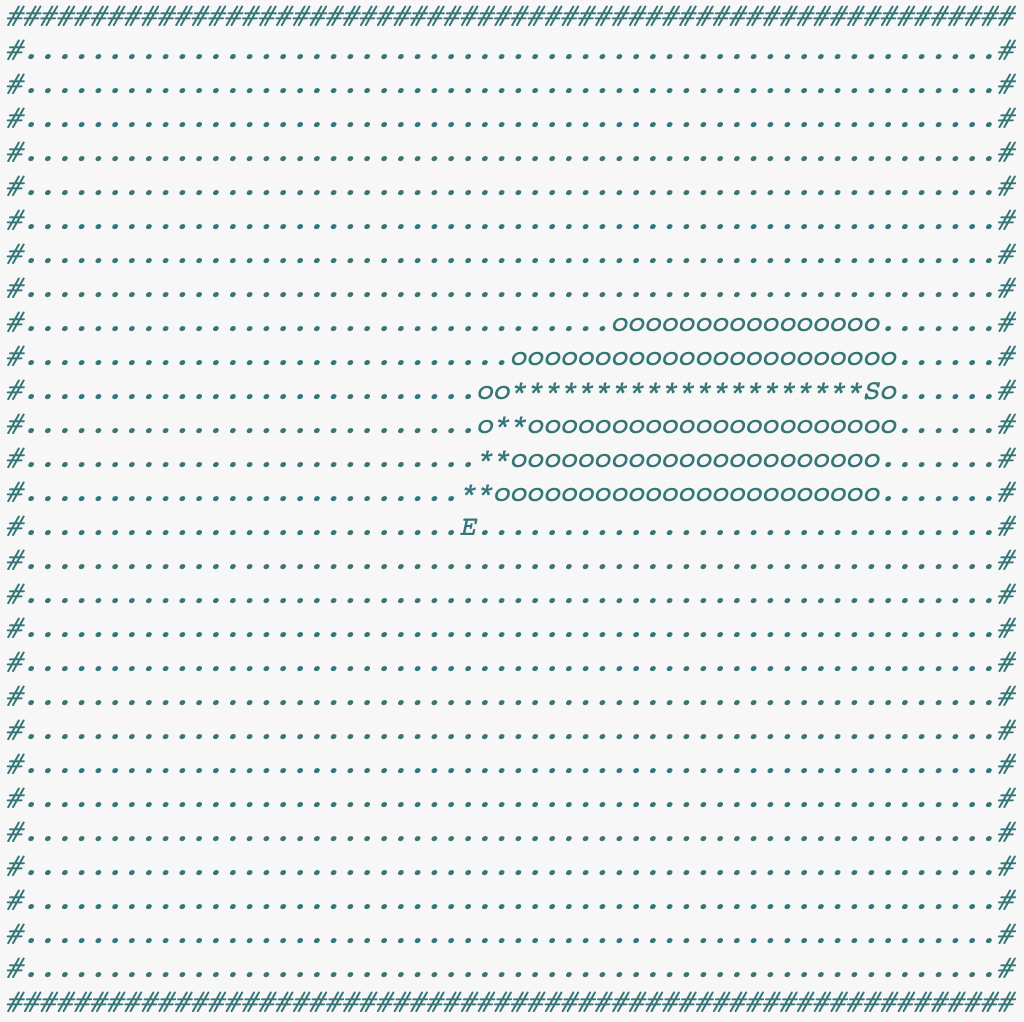
 

图4-4 欧氏距离估价有障碍图 图4-5 欧式距离估价无障碍图

### 4.1.3曼哈顿距离作为估价函数

曼哈顿距离作为估价函数是我在第三章的例子中使用的估价函数，由于禁止斜着走迷宫，所以该估价函数在没有障碍是等同于实际的消耗函数的，根据推测，表现的应该最好。由于有障碍的图已经在图3-9展示过，这里只展示该估价函数在无障碍图的表现。如图4-6。

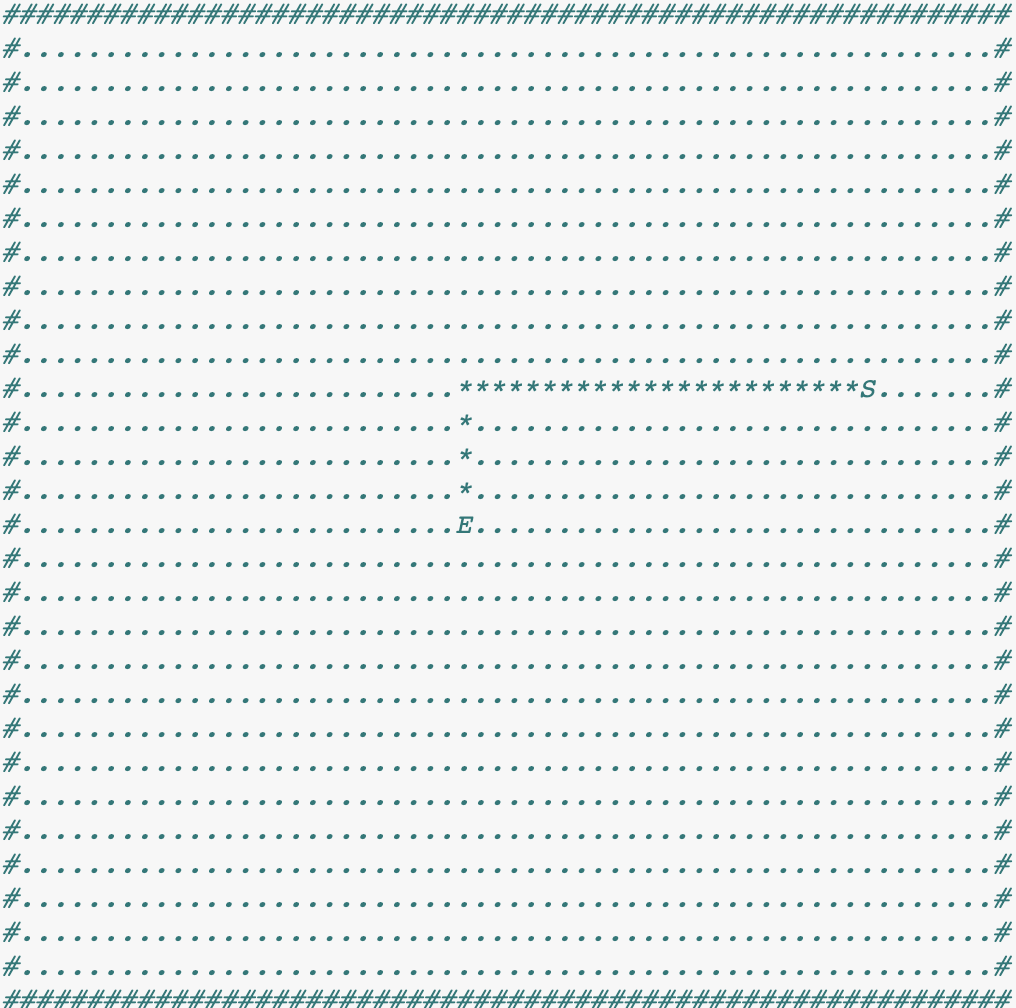


图4-6 曼哈顿估价无障碍图

### 4.1.4三种估价函数随机实验结果

为了更准确的判断，将随机生成100\*100的随机迷宫，对三种估价函数的搜索效率进行比对。实验重复1000次，取实际访问点数量的平均数作为不同估价函数访问效率的依据，详情见表4-1。

表4-1 随机实验效率表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 估价函数 | 无障碍图下1000次随机搜索的平均访问点数 | 10%障碍图下1000次随机搜索的平均访问点数 | 20%障碍图下1000次随机搜索的平均访问点数 | 40%障碍图下1000次随机搜索的平均访问点数 |
| 常数估价函数 | 4309.25 | 4145.55 | 3373.45 | 1196.88 |
| 欧式距离估价函数 | 1561.51 | 1240.89 | 1181.13 | 653.00 |
| 曼哈顿距离估价函数 | 66.68 | 163.74 | 295.06 | 548.61 |

## 4.2 分析估价函数对搜索效率的影响

估价函数对搜索效率的优化是显而易见的，当搜索使用常数估价函数（不使用估价函数时），效率是很低的。但即使是相似的估价函数，对效率的影响也是很大的。

根据表4-1的数据，欧式距离在障碍较少的环境下，实际搜索效率是不如曼哈顿距离估价函数的。原因是欧式距离和实际的路径花费存在偏差，不能贴合实际的路径花费。而曼哈顿距离估价函数可以很好的贴合实际路径花费，所以在搜索效率上，尤其是障碍较少的情况下，可以更快的查找到最优路径。

所以，在满足A\*搜索算法的条件下，估价函数越大（越贴合实际的路径花费），搜索效率就会越高。

但是，由于有障碍的存在，估价函数不可能完全贴合实际路径花费。因此，在障碍数增加的情况下，使用欧式距离的估价函数和曼哈顿距离的估价函数的搜索效率，是在不断缩小差距的。只考虑个例的情况下，甚至表现没那么好的欧式距离估价函数都有不小的可能超过曼哈顿距离估价函数的效率。

## 4.3 本章小结

通过实践分析了估价函数对A\*搜索算法效率的影响，证实了估价函数越贴合实际路径花费，A\*搜索算法的效率越高。在实际使用中，由于障碍的不确定性，无法选择一个完全优秀的估价函数，只能让估价函数尽可能的贴合实际的路径花费，这样才能最大程度的提升搜索效率。

# 第5章 A\*搜索算法和其他搜索的对比

## 5.1 与宽度优先搜索的对比

A\*搜索算法在不考虑*h(n)*时，即只考虑已经走过的距离时，它的表现和宽度优先搜索时高度类似的。实际上，A\*搜索算法就是源于宽度优先搜索，在其基础上扩展了对当前节点到目标节点的预估距离。

对于A\*搜索算法和宽度优先搜索的对比数据，可以参照表5-1。

表5-1 随机实验效率表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 搜索算法 | 无障碍图下1000次随机搜索的平均访问点数 | 10%障碍图下1000次随机搜索的平均访问点数 | 20%障碍图下1000次随机搜索的平均访问点数 | 40%障碍图下1000次随机搜索的平均访问点数 |
| 宽度优先搜索 | 5538.55 | 5179.0 | 3334.45 | 974.76 |
| A\*搜索算法 | 66.68 | 163.74 | 295.06 | 548.61 |

表5-1中A\*搜索的数据来源于第四章的表4-1。从表中可以看出，A\*搜索算法通过对估价函数的使用，大大提高了搜索效率，远远超过了普通的宽度优先搜索的搜索效率。

不过，如果是保证存在路径的复杂迷宫图，A\*搜索算法和宽度优先搜索算法的效率差距并不大。考虑到A\*搜索算法需要计算启发式的估价函数同时内部还需要维护*OPEN*表的优先队列。实际在使用中，如果迷宫很复杂，使用宽度优先搜索在时间上不会比A\*搜索差太多，而且实现简单快捷，仍不失为一个好选择。

## 5.2 与深度优先搜索的对比

显而易见的是，深度优先搜索在保证最优解的情况下，搜索效率是A\*搜索的指数级的倍数。因此深度优先搜索不适合计算最短路径，而是适合通过回溯的方式，找到一条可行的路径。因此，A\*搜索算法和深度优先搜索的对比也主要以发现可行路径的效率进行对比，并不强制要求寻找的路线是最优解。

对于A\*搜索算法和深度优先搜索的对比数据，可以参照表5-2。

表5-2 随机实验效率表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 搜索算法 | 无障碍图下1000次随机搜索的平均访问点数 | 10%障碍图下1000次随机搜索的平均访问点数 | 20%障碍图下1000次随机搜索的平均访问点数 | 40%障碍图下1000次随机搜索的平均访问点数 |
| 深度优先搜索 | 4810.90 | 3638.70 | 2371.25 | 979.56 |
| A\*搜索算法 | 66.68 | 163.74 | 295.06 | 548.61 |

从表中可以看出，A\*搜索算法即使实在寻找可行路径的测试中，运行效率也远远超过了普通的深度优先搜索的搜索效率。

对于深度优先搜索，其搜索效率比起宽度优先搜索有些提升，但是实际的搜索的稳定性比较差，甚至出现了遍历整张图才找到一个可行路径的情况。深度优先搜索是一个不太稳定的搜索方式。考虑到深度优先搜索还不能保证搜索得到的路径是最优解，所以在实际使用中的意义不是很大。

相比之下，A\*搜索算法兼顾了速度和稳定性，性能比起深度优先搜索有了很大的提升，基本可以在最短路问题中替代深度优先搜索。

## 5.3 与Dijkstra算法的对比

Dijkstra算法和A\*搜索算法一样，都是从宽度优先搜索演化而来。Dijkstra算法是由计算机科学家Edsger W. Dijkstra在1956年提出的。Dijkstra算法本质上仍旧是盲目搜索，但是比起宽度优先搜索，Dijkstra额外考虑了移动代价差异对搜索的影响[17]。当在网格图中进行搜索时，如果步长没有区别，Dijkstra算法将退化为A\*搜索算法。

由于Dijkstra算法在普通的网格图中会退化为宽度优先搜索，因此，我们将迷宫问题的规则进行修改。为迷宫的每一个点随机赋一个消耗值，以此来区分不同路径的消耗。

对于A\*搜索算法和Dijkstra算法的对比数据，如表5-3所示。

表5-3 随机实验效率表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 搜索算法 | 无障碍图下1000次随机搜索的平均访问点数 | 10%障碍图下1000次随机搜索的平均访问点数 | 20%障碍图下1000次随机搜索的平均访问点数 | 40%障碍图下1000次随机搜索的平均访问点数 |
| Dijkstra算法 | 4510.70 | 3338.40 | 2271.21 | 979.56 |
| A\*搜索算法 | 1341.61 | 994.71 | 853.12 | 649.61 |

观察实验结果，可以发现Dijkstra的搜索效率似乎不是很高，但是实际上，A\*搜索的效率也比普通的网格图下降了很多。原因是因为加入点的消耗后，估价函数没有普通的网格图那么的贴合了。

可以看到，Dijkstra算法确实比普通的宽度优先搜索优化了很多，但是比起A\*搜索算法，效率上还是有所差距。可以证明，启发式的搜索对搜索效率的影响是极大的。

## 5.4 本章小结

通过实践分析了A\*搜索算法和其他搜索算法的搜索效率。证实了启发式的A\*搜索算法，搜索效率是普通的盲目搜索不能媲美的。

# 第6章总结和展望

## 6.1 总结

本文以理论知识为基础，通过多个角度分析介绍了A\*搜索算法的多个方面，通过实验完成了A\*搜索算法的简单应用和与其他搜索算法的对比。

本文主要完成了以下工作：

（1）分析了A\*搜索算法的发展背景与意义，对A\*搜索算法做了整体上的介绍。

（2）分析A\*搜索算法的实现原理，挖掘A\*搜索算法高效率的原因，并给出证明。

（3）给出A\*算法的调用方式，在具体问题中实践测试A\*算法。

（4）对比分析不同的估价函数对A\*搜索算法效率的影响。

（5）和其他常见的搜索算法进行对比，诊断其效率差异的主要原因。

## 6.2 展望

A\*搜索算法已经是当前最广泛使用的搜索算法之一，在日常生活中也有广泛的使用。作为最有影响力的启发式搜索算法，其本身也演化出许多的变种算法。包括ARA\*，D\*，Field D\*，Block A\*等算法。这些算法都是从A\*演进，特化了某些方向以达到更好的搜索效果。

同时，A\*搜索算法也在工业界被广泛应用，包括但不限于生物学物质的结构[18]，无人机路径规划[19]，锂电池电子平衡[20]，引擎硬件架构优化[21]等应用。

之后的时间我也将继续研究A\*搜索算法，尝试在某一领域更近一步优化其搜索效率，构建一个更有实际意义的解决方案。

# 参考文献

[1] Nosrati M, Karimi R, Hasanvand H A. Investigation of the\*(star) search algorithms: Characteristics, methods and approaches[J]. World Applied Programming, 2012, 2(4): 251-256

[2] 张婷娟. 基于A\*算法的最短路径寻优数学方法研究[J]. 科技通报, 2015, 31(6):4.

[3] 钟志峰, 易明星, 陈智军,等. 基于改进A^\*算法的导购路径规划方法[J]. 计算机工程与应用, 2019, 55(5):6.

[4] Yao J, Lin C, Xie X, et al. Path planning for virtual human motion using improved A\* star algorithm[C]//2010 Seventh international conference on information technology: new generations. IEEE, 2010: 1154-1158.

[5] Pearl J. Heuristics: intelligent search strategies for computer problem solving[M]. Addison-Wesley Longman Publishing Co., Inc., 1984.

[6] Ramalingam g, Reps T. An incremental algorithm for a generalization of the shortest-path problem[J]. Journal of Algorithms, 1996, 21(2): 267-305.

[7] Eklund P W, Kirkby S, Pollitt S. A dynamic multi-source Dijkstra's algorithm for vehicle routing[C]//1996 Australian New Zealand Conference on Intelligent Information Systems. Proceedings. ANZIIS 96. IEEE, 1996: 329-333.

[8] Arai H, Maung C, Schweitzer H. Optimal column subset selection by A-Star search[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2015, 29(1).

[9] Kusuma M, Machbub C. Humanoid robot path planning and rerouting using A-Star search algorithm[C]//2019 IEEE International Conference on Signals and Systems (ICSigSys). IEEE, 2019: 110-115.

[10] 张胜. 一种基于状态空间的启发式搜索算法及其实现[J]. 现代电子技术, 2008, 31(16): 79-80.

[11] 丛明煜, 王丽萍. 现代启发式算法理论研究[J]. 高技术通讯, 2003, 13(5): 105-110.

[12] Niu L, Zhuo g. An improved real 3D A\* algorithm for difficult path finding situation[J]. Proceeding of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2008, 37.

[13] 朱学智. 基于遗传算法的最短路径问题研究[D]. 中国科学技术大学, 2015.

[14] Seo W J, Ok S H, Ahn J H, et al. An efficient hardware architecture of the A-star algorithm for the shortest path search engine[C]//2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE, 2009: 1499-1502.

[15] 邓新国, 叶似锦, 陈家瑞, 等. 结合改进 A\* 算法与拆线重布的有序逃逸布线[J]. 电子与信息学报, 2021, 43(6): 1609-1616.

[16] 王妍, 许崇芳, 雷玉霞. 八数码问题的 JAVA 设计与实现[J]. 计算机与信息技术, 2006 (8): 32-34.

[17] 刘云翔, 杜杰, 张晴. 基于路径优化的A\*算法与Dijkstra算法的性能比较[J]. 现代电子技术, 2017, 40(13):4.

[18] Stocco D M. A StAR search: implications in controlling steroidogenesis[J]. Biology of reproduction, 1997, 56(2): 328-336.

[19] Tseng F H, Liang T T, Lee C H, et al. A star search algorithm for civil UAV path planning with 3g communication[C]//2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IEEE, 2014: 942-945.

[20] Dong g, Yang F, Tsui K L, et al. Active balancing of lithium-ion batteries using graph theory and A-Star search algorithm[J]. IEEE Transactions on Industrial Informatics, 2020, 17(4): 2587-2599.

[21] Seo W J, Ok S H, Ahn J H, et al. An efficient hardware architecture of the A-star algorithm for the shortest path search engine[C]//2009 Fifth International Joint Conference on INC, IMS and IDC. IEEE, 2009: 1499-1502.

# 致谢

在毕设的完成过程中，有很多人给予了我支持。

我首先要感谢我的指导老师刘全教授。在疫情下，很多指导不能当面展开，但仍旧通过邮件和微信等方式指导毕设的开展，帮助我完善论文。在此由衷的感谢刘老师的教导。

其次要感谢朱晓旭老师。他是我大学竞赛生涯的教练。在他的指导下，我才能打扎实算法和代码功底。在生活和训练中，他也就给予了我许多的帮助，是我不可多得的良师。

我也要感谢四年授予我知识的所有老师，是你们的奉献和教导造就了现在的我。

最后，我要感谢我的家人，感谢他们对于我大学四年学业的支持，感谢对于我写作论文期间的支持。