建模与仿真报告

A*算法在最短路径优化设计中的应用

学生姓名：侯爽

组　　员：李璐君、侯爽

学　　号：140543111

指导教师：周隽

学　　院：中欧航空工程师学院

二〇一八年四月

摘　要

最短路径设计问题是地理信息系统、航空领域航路规划问题中的关键问题，目前针对这一问题许多专家学者已经提出了许多路径搜索的优化算法，典型的有 Dijkstra 算法、最佳优先算法（Best-First Search， BFS）、A*算法等等，都能对当前结点进行评估。A*算法是建立在典型的 Dijkstra 算法基础之上的启发式搜索算法，它结合了 Dijkstra 算法和最佳优先搜索算法的双重优点，变得更灵活、搜索效率更高。

利用 A*算法，我们进行了两部分的最优路径搜索与设计，分别是网格式地图中的最优路径搜索，和二维平面中的最优航路设计。

在网格式地图的路最优路径搜索中，标准启发式函数采用的是曼哈顿距离和切比雪夫距离。两者对应的展开方式分别为：曼哈顿展开方式是向上下左右四个方向相邻的结点展开，切比雪夫展开方式是在曼哈顿展开方式的基础上，又增加了两条对角线上的结点展开。A*算法的 Java 实现仿真结果显示，A 算法确实能搜索出一条距离最短的最优路径。

在二维平面最优航路设计中，启发式函数选为当前结点到目标结点之间的欧氏距离，将障碍物模型化为多边形、圆、椭圆、一维线段等几何图形，利用引射线法、跨立实验等方法，重点研究了搜索路径过程中的避障问题。Java 实现的仿真结果表面，A*算法搜索得到的路径确实是一条距离最短的航路。

通过观察展开结点的数量以及展开趋势可以发现，A*算法也有一定的局限性。由此，我们进一步提出了两种基于 A*算法的航路设计优化方案，分别是深度优先搜索优化算法（Depth First Search，DFS）和 Chebychev 优化算法。

在同样的起点终点、同样的障碍物分布的地图中，分别用这两种优化算法以及最初的 A*算法进行测试。通过对比三个算法的结果可以看出，DFS 算法找到的最优路径和 A*算法的路径有时不完全一样，而 Chebychev 优化算法找到的最优路径与 A*算法的路径是几乎相同。三种算法中，DFS 算法得到的最优路径的距离要比 A*算法和 Chebychev 算法得到的路径要远一些。但另一方面，DFS 算法比 A*算法展开的结点要少很多，Chebychev 算法展开的结点更是远远少于 DFS 算法。更直观的体现是在时间方面， DFS 优化算法搜索用时比 A*算法减少了近 17 倍，而 Chebychev 优化算法搜索用时比 A*算法用时减少了近 85 倍，比 DFS 优化算法用时减少了近 5 倍。由此可以得出结论，大部分情况下，Chebychev 优化算法相比较其他两者是更优的，并且是搜索效率最高的。

在真实的终端区航路设计中，如果遇到多架航空器同时进近的情况，需要我们在

最短的时间内规划出每架航空器的最优路径，在这种情况下，路径距离的长短与规划搜索时间都是限制因素，在这种情况下，A*算法、DFS 算法和 Chebychev 优化算法都适用，并且，基于 A*算法的 Chebychev 优化算法更优。

关键词：A*算法，最短路径设计，二维航路设计，A*算法优化

Abstract

The shortest path problem is the geographic information system design, air route planning problem, the key issues in the field at present in order to solve this problem many experts and scholars have put forward many path Search optimization algorithm, A typical have Dijkstra algorithm, best-first algorithm (Best - First Search, BFS), A * algorithm, etc., can be to evaluate the current node.

Three kinds of algorithms, Dijkstra algorithm is the most familiar with the basic algorithm of solving the shortest path problem, the main feature is the starting point as the center to the outer layers of extension, until the extension to the target destination location. Although Dijkstra algorithm can get the optimal solution of the shortest path, but as it traverses a lot of computing nodes, usually relatively large amount of calculation, so the efficiency is lower, is suitable for small scale of the shortest path computation. In addition, Dijkstra algorithm cannot handle the shortest path problem with negative weight value.

Best-First Search (BFS) runs in a similar process, using a heuristic function (heuristic function) to quickly guide the target node. Unlike the closest node to the initial node, it chooses the closest node to the target, and it can evaluate the cost of any node to the target point. BFS is not guaranteed to find a shortest path, but BFS runs faster than Dijkstra.

Combined with the advantages of Dijkstra algorithm and the optimal priority algorithm, A* algorithm with more flexibility and higher search efficiency is generated. A * algorithm is based on the typical Dijkstra algorithm based intelligence heuristic search algorithm, the innovation of the A * algorithm is to select the node is next to search the introduction of the known road network information, especially the target node information, to evaluate the distance to the current node to the end, the nodes belong to the optimal path as the evaluation of the possibility of indicators, so that you can first search probability of the node, thus improve the efficiency of the search process. A little bit different, a heuristic method like BFS often gives an approximate solution without a guarantee of the optimal solution. However, while A* algorithm is based on A heuristic method that does not guarantee optimal solutions, A* can guarantee to find A shortest path.

Using the A* algorithm, we conducted the optimal path search and design of the two parts, respectively the optimal path search in the network format map and the optimal route design in the two-dimensional plane.

Firstly, the optimal path search is conducted in the network format map. For the heuristic function selection, we use two standard heuristic functions: Manhattan distance and chebyshev distance. The Manhattan distance is defined as the absolute value of the difference between the current node and the horizontal ordinate of the target node, multiplied by the cost coefficient; The chebyshev distance is defined as the larger of the absolute value of the difference between the current node and the horizontal ordinate of the target node, multiplied by the cost coefficient. The corresponding way: Manhattan way is in the direction of the up and down or so four neighboring nodes, chebyshev way is the way in Manhattan, on the basis of adding the nodes on the two diagonals.

Through the Java implementation of A* algorithm, we have obtained the search results of the optimal path in the two search modes. And on this basis, we further tested the map with more grid Numbers. Empirical evidence shows that A algorithm can actually search the optimal path, and the optimal path obtained by the search is indeed the shortest path. And, in comparison, chebyshev's search results are shorter than those obtained in Manhattan.

In addition, in order to improve the visual effect, make the path is more intuitive, we will be carried out in the nodes and stay on for not optimal and is not a node respectively in different colors, can be found through observation, those who stay on but has not been launched by not optimal nodes nodes is always surrounded by those who fought, was carried out in the boundary of the node.

Secondly, the route design in two - dimensional plane is carried out. Route design and design is the biggest difference between grid map path, each of grid map grid corresponds to a node, the grid is generated, each node corresponding to the location information is then determined. In the two-dimensional plane, there is no fixed node, so we define a search Angle and a search step d, and we start out five nodes each time, until we stop at the target node. In addition, the heuristic function h(n) used in the two-dimensional plane route design is the Euclidean distance between the current node and the target node.

In order to simplify the model, we will obstacle model into a polygon, circle, ellipse, a one-dimensional line geometry, we studied the search process is the most important obstacle avoidance problem, namely the search point and the position relationship between geometrical obstacles, and the search path intersecting geometric obstacles of sex. In this part we make full use of the various properties of the geometry, combining ray method, fast exclusion experiments, straddles the experimental method, solves the route is very important part of design: the search path of obstacle avoidance problem.

Through the implementation of Java, we have obtained the optimal paths of different

obstacles, and the results obtained by A* algorithm are indeed the shortest route. And, similarly, we mark each node in a different color, in addition to the obstacles along the way in other colors, as well as the final path search results. Can be seen from the results on the way, those who stay on for not optimal nodes without being expanded is always carried out in the boundary of the nodes, the discovery at the same time should also be card in the grid map we've come to the conclusion that this rule.

On the other hand, by observing the number of nodes and expanding the trend, it can be found that A* algorithm has certain limitations. For example, sometimes it will expand some unnecessary nodes, increase the computation and iteration times, resulting in too long search time. Therefore, we further proposed two route design optimization schemes based on A* algorithm, which are Depth First Search, DFS and Chebychev optimization algorithm respectively.

The basic idea of the depth-first search optimization algorithm is to go deep into every possible branching path, which is to give priority to searching for those nodes that are already deep. This will give priority to the node near the target point when several nodes have the same f(n) value.

Chebychev optimization algorithm the basic idea is to first path roughing, find A can from beginning to end A better path as A directed path, then near the instructions path using the A * algorithm to search the optimal path. Both of these scenarios can avoid unneeded nodes in the search process.

In the same map, we used these two optimization algorithms and the original A* algorithm to find the ultimate optimal route. Can be seen by comparing the results of the three algorithms, depth first search algorithm find the optimal path and the path of the A * algorithm is not exactly the same, sometimes and Chebychev optimization algorithm to find the optimal path and the path of the A * algorithm is almost the same. In the three algorithms, the distance of the optimal path obtained by the depth-first algorithm is far more than that of A* algorithm and Chebychev algorithm.

But on the other hand, depth-first search algorithm has fewer nodes than A* algorithm, and the node of Chebychev algorithm is far less than the depth first search algorithm. Is more intuitive representation in terms of time, A * algorithm to search 57600 milliseconds, DFS optimization algorithm search 3445 milliseconds, and Chebychev optimization algorithm search just 676 milliseconds, than A * algorithm reduces the nearly 85 times, than the DFS optimization algorithm is reduced nearly 5 times. It can be found that in most cases, Chebychev optimization algorithm is better than the other two, and it is the most efficient.

In the terminal area real route design, if there are any more spacecraft into nearly at the same time, we need to in the shortest possible time planning out the optimal path of each aircraft, in this case, the length of the path distance and planning the search time is the limiting factor, in this case, the A * algorithm, depth first search algorithm and Chebychev optimization algorithm, and Chebychev optimization algorithm based on A * algorithm is better.

In this paper, the optimization and improvement of A* algorithm in the two-dimensional planar route design is carried out, which is mainly aimed at the expansion point. The improved algorithm in the test result is ideal, but the current research is still in the two-dimensional plane in the air route design, in reality are need to consider more yes optimal route design problem in three-dimensional space. Therefore, the future design of the optimal navigation path in 3d space will be further deepened, which makes the A* algorithm and the improved optimization plan more in line with the actual requirements.

Key words： A* algorithm, Shortest path design, Two-dimensional route design, A* algorithm optimization.

目　录

摘　要 I
Abstract III
目　录 VII
第 1 章　绪论 1
1.1　研究背景与意义 1
1.2　常用搜索算法回顾 2
1.2.1　Dijkstra 算法 2
1.2.2　最佳优先算法 4
1.3　A*算法的引入 5
第 2 章　A*算法在网格图中的应用 8
2.1　启发式函数的选取 8
2.1.1　曼哈顿距离 8
2.1.2　切比雪夫距离 9
2.2　A*算法的设计与实现 10
2.3　仿真结果与分析 10
第 3 章　A*算法在二维航路设计中的应用 15
3.1　模型假设 15
3.2　启发式函数的选取 15
3.3　点与障碍物的位置判定 15
3.3.1　点与多边形障碍物的位置关系 15
3.2.2　点与特殊形状障碍物的位置关系 18
3.4　搜索路径与障碍物的相交性判定 18
3.4.1　搜索路径与一维线段障碍物的相交性 18
3.4.2　搜索路径与特殊形状障碍物的相交性 20
3.5　A*算法的设计与实现 20
3.5　仿真结果与分析 21
第 4 章　航路设计中 A*算法的优化 22
4.1　深度优先搜索优化 22
4.2　Chebychev 优化 22
4.3　两种优化算法的对比 23
4.3.1　最优路径的对比 23
4.3.2　搜索时间的对比 25
第 5 章　总结与展望 27
5.1　研究工作总结 27
5.2　未来研究期望 28
致　谢 29
参考文献 30

第 1 章　绪论

1.1　研究背景与意义

随着民用航空事业的蓬勃发展、军事飞行训练任务的逐步加重和航空器的大型化，以及航空的日益增多，空中交通流量急剧增加。但我国的空域结构和航路设计等却未出现太大变化，至使现有的空域已趋于饱和状态，空中交通网络拥挤现象不断发生，因此航班延误现象屡见不鲜，这不但增加了空管人员工作的难度，也大大增加了航空公司的运行成本，并且造成航空安全隐患。

交通需求量高的区域通常拥有两个及以上的机场，这些机场的进离场运行相互影响。密集的机场布局、复杂的空域结构和繁忙的交通流量使得空中交通愈发拥挤，空域结构规划不合理导致的空中交通拥挤问题日益严重。因此，航路的设计成为了一个重要问题。

我国航空运输占世界的比重由 1996 年的 2.4%上升到 2016 年的 6.7%，但是因空域不足和流量分布不均带来的飞行冲突和延误也日趋增多，这也加剧了拥挤现象的发生。空中交通网络的拥堵的主要原因在于由机场、终端区以及航路交叉点的容量限制引发的“瓶颈”效应;就中国的空中交通系统而言,容量限制情况最严重的部分往往是终端区。为了合理的使用空域资源，加快空中交通流量，保证飞行安全，终端区的航路规划非常重要。

终端(进近)管制区(TERMINAL/APPROACH CONTROL AREA)设在一个或者几个主要机场附近的空中交通服务航路汇合处的管制区。终端区空域是整个空中交通网络中非常复杂的一个空域，这个空域是到达航班脱离航路进行进近着陆的过渡区，也是起飞航班起飞后加入航路的必经之地。不管是进场航班还是离场航班，它们在终端区都需要不断得到改变高度、速度、航向等飞行状态，特别是加上多机场终端区的航线结构错综复杂，为了保证航班的安全间隔，使航班有条不紊的进、离场，对多机场终端区的空域结构的认识很有必要。多机场终端区的进离场航线错综复杂、互相交织，很多进场和离场航线还共用一个走廊口，如图 1-1 所示。因此，合理的规划进离场航线、优化进离场航班次序，才能充分利用多机场终端区的容量，提高多机场终端区中航班运行效率。

航路设计用到的是最优路径设计的各种算法,目前已经有几种相对比较成熟的算法可以用来设计较优的航路。本文，我们将研究一种最受欢迎、应用率最高、也是最灵活的最优路径设计算法--A*算法，将它先在网格图中进行初步实现，然后进一步应用于二维最优航路设计中去。

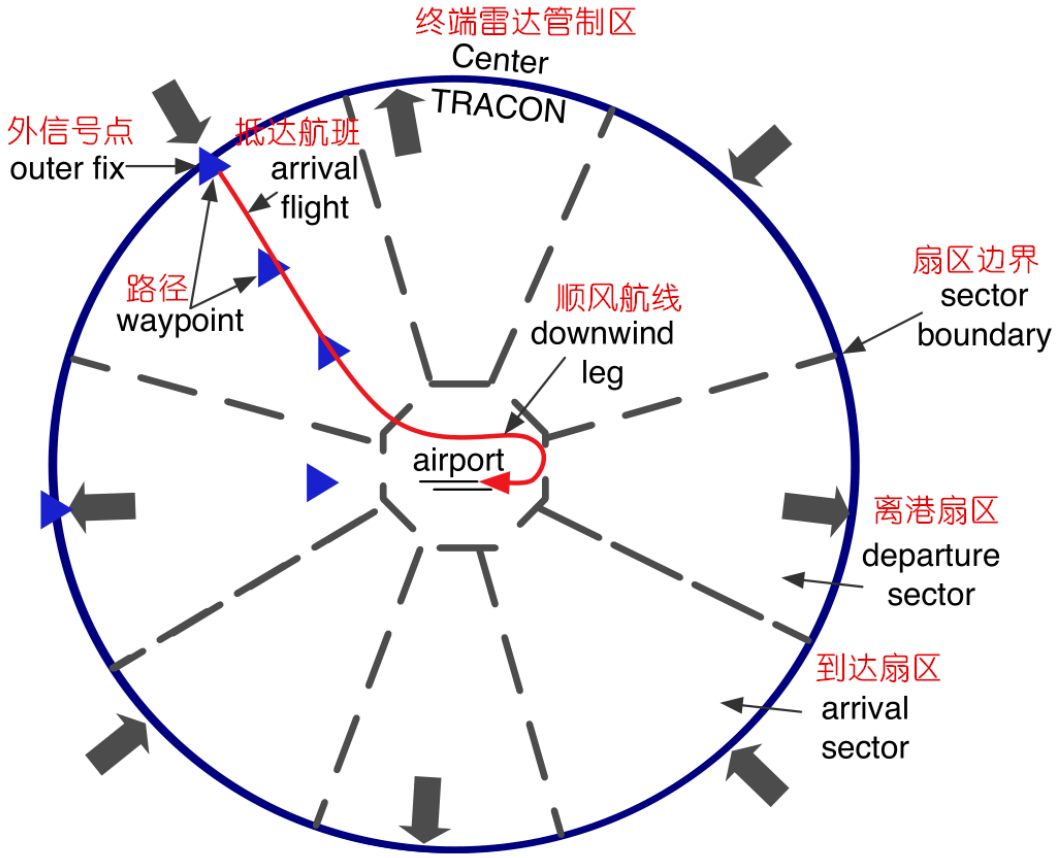

图 1-1 机场终端区模型

1.2 常用搜索算法回顾

1.2.1 Dijkstra 算法

Dijkstra 算法从物体所在的初始点开始，访问图中的结点。它迭代检查待检查结点集中的结点，并把和该结点最靠近的尚未检查的结点加入待检查结点集。该结点集从初始结点向外扩展，直到到达目标结点。

只要所有的边都有一个非负的代价值，Dijkstra 算法保证能找到一条从初始点到目标点的最短路径。在下图 1-2 中，粉红色的结点是初始结点，蓝色的是目标点，而类似菱形的有色区域，则是 Dijkstra 算法扫描过的区域。颜色最淡的区域是那些离初始点最远的，因而形成探测过程（exploration）的边界（frontier）。

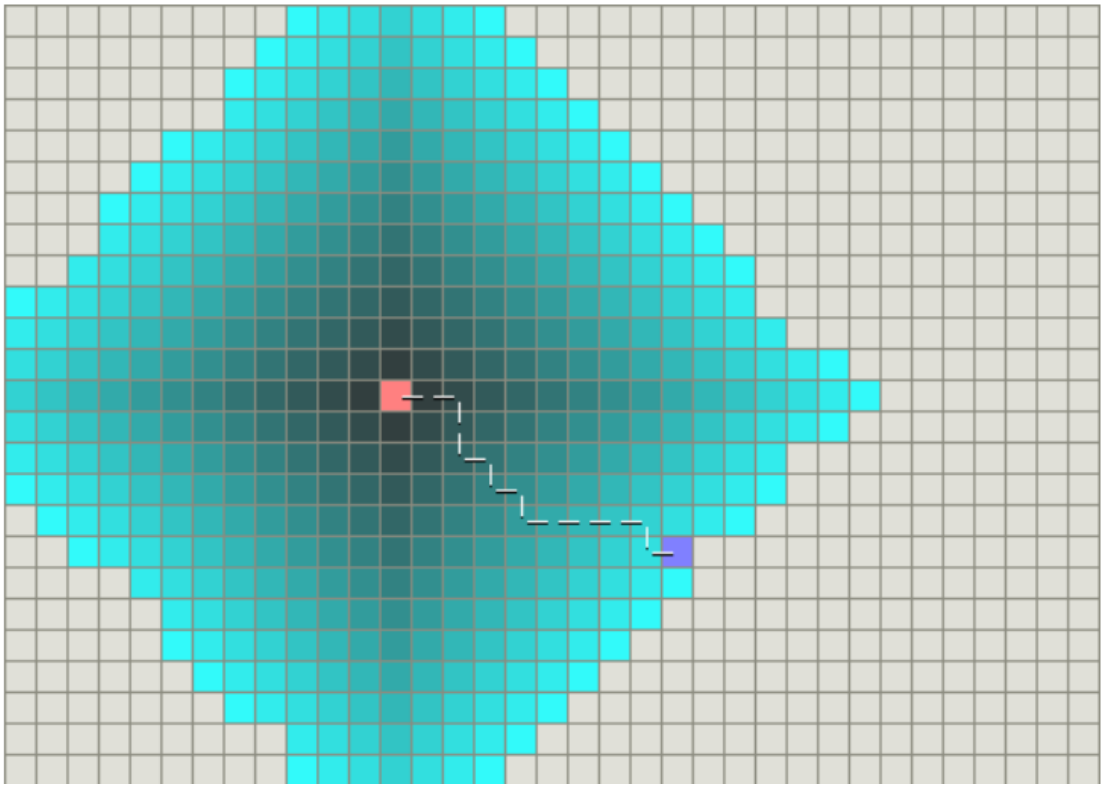

图 1-2　Dijkstra 算法在无障碍时搜索路径

上述这个例子仅仅是最简单的情况——地图中没有障碍物，最短路径是直线的。现在考虑当地图上存在凹型障碍物时的情况。

Dijkstra 算法运行得较慢，但确实能保证找到一条最短路径，如图 1-3 所示。

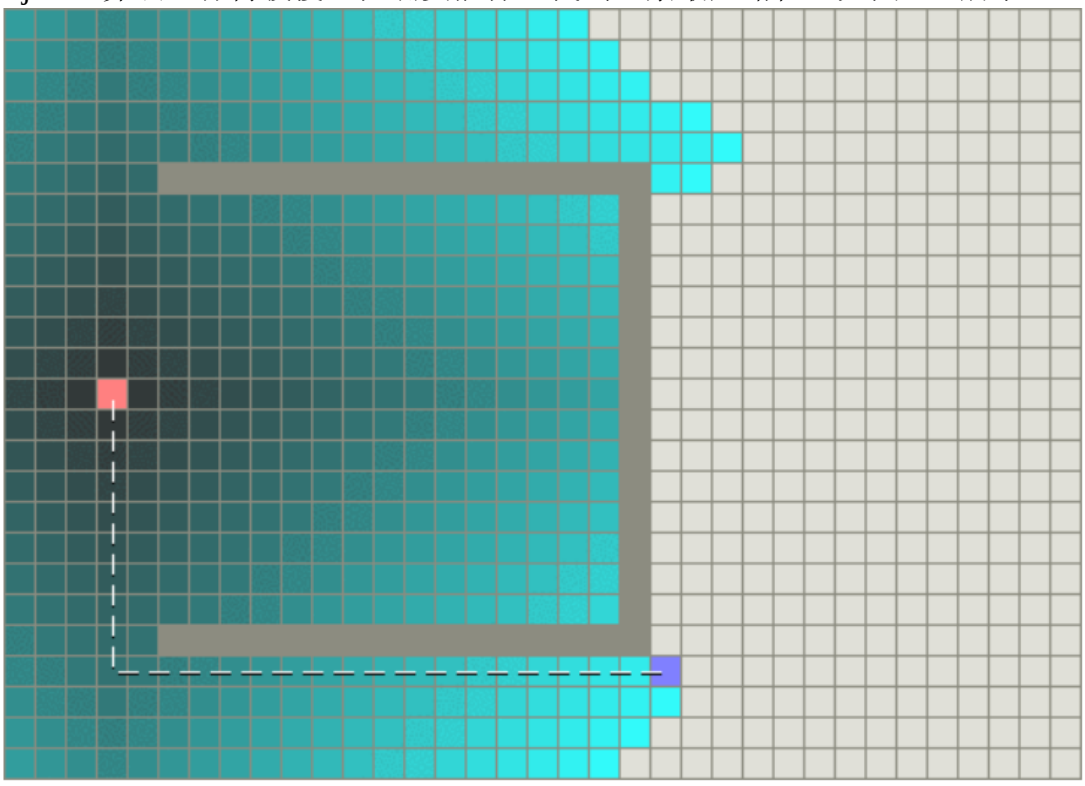

图 1-3　Dijkstra 算法遇凹型障碍物时搜索路径

1.2.2 最佳优先算法

最佳优先搜索（Best-First Search, BFS）算法按照类似的流程运行，不同的是它能够评估（称为启发式的）任意结点到目标点的代价。与选择离初始结点最近的结点不同的是，它选择离目标最近的结点。BFS 不能保证找到一条最短路径。然而，它比 Dijkstra 算法快的多，因为它用了一个启发式函数（heuristic function）快速地导向目标结点。例如，如果目标位于出发点的南方，BFS 将趋向于导向南方的路径。在图 1-4 中，越黄的结点代表越高的启发式值（移动到目标的代价高），而越黑的结点代表越低的启发式值（移动到目标的代价低）。这表明了与 Dijkstra 算法相比，BFS 运行得更快。

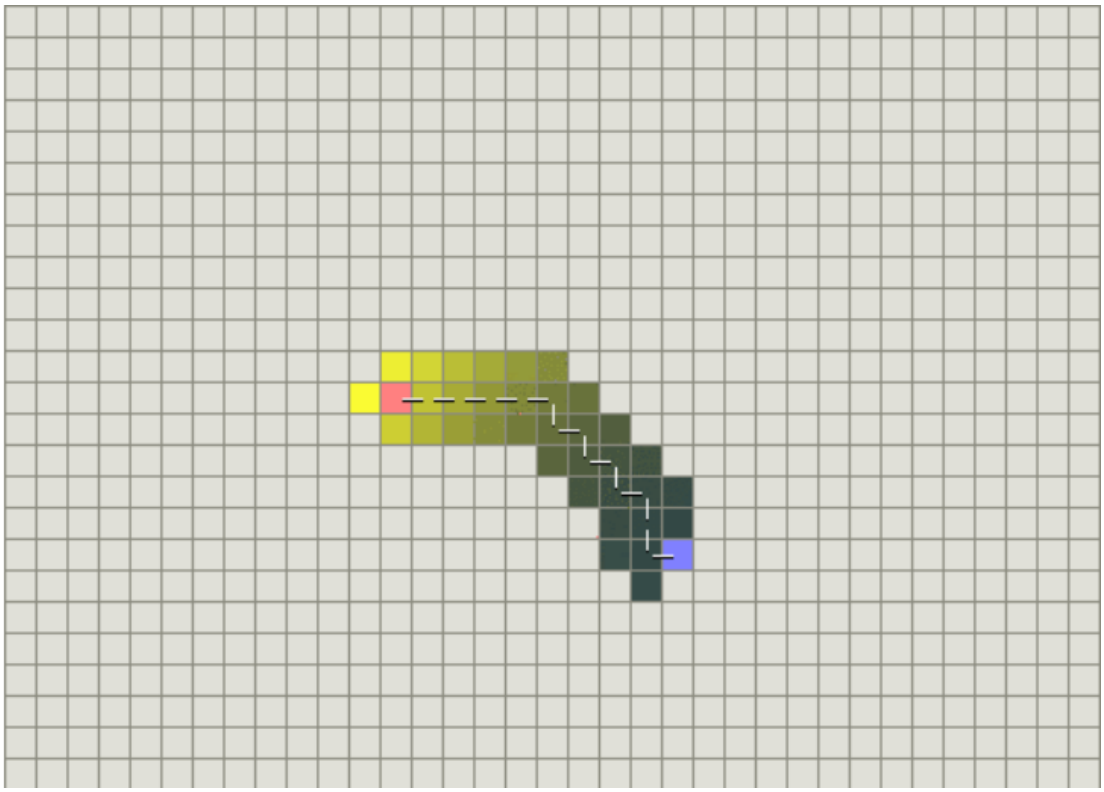

图 1-4 最佳优先算法在无障碍时搜索路径

同样，上述这个例子仅仅是最简单的情况——地图中没有障碍物，最短路径是直线的。现在同样考虑当地图上存在凹型障碍物时的情况。

BFS 运行得较快，如图 1-5 所示，但是它找到的路径明显不是一条好的路径。主要原因在于 BFS 是基于贪心策略的，它试图向目标移动尽管这不是正确的路径。由于它仅仅考虑到达目标的代价，而忽略了当前已花费的代价，于是尽管路径变得很长，它仍然继续走下去。

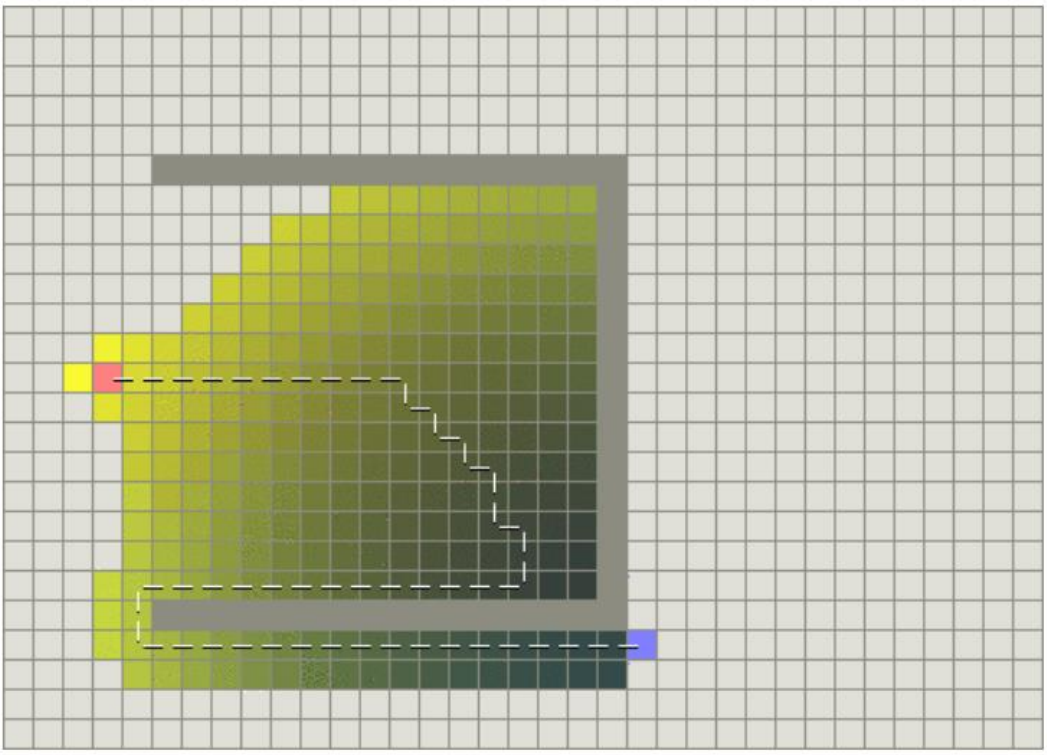

图 1-5　BFS 算法遇凹型障碍物时搜索路径

1.3　A*算法的引入

结合 Dijkstra 算法和最佳优先算法的优点，于是生成了更加灵活的 A*算法。A*算法就是把启发式方法（heuristic approaches）如 BFS，和常规方法如 Dijsktra 算法结合在一起。有点不同的是，类似 BFS 的启发式方法经常给出一个近似解而不是保证最佳解。然而，尽管 A*基于无法保证最佳解的启发式方法，A*却能保证找到一条最短路径。

和其它的图搜索算法一样，A*潜在地搜索图中一个很大的区域。和 Dijkstra 一样，A*能用于搜索最短路径。和 BFS 一样，A*能用启发式函数引导它自己。在简单的情况中，它和 BFS 一样快，如图 1-6 所示。

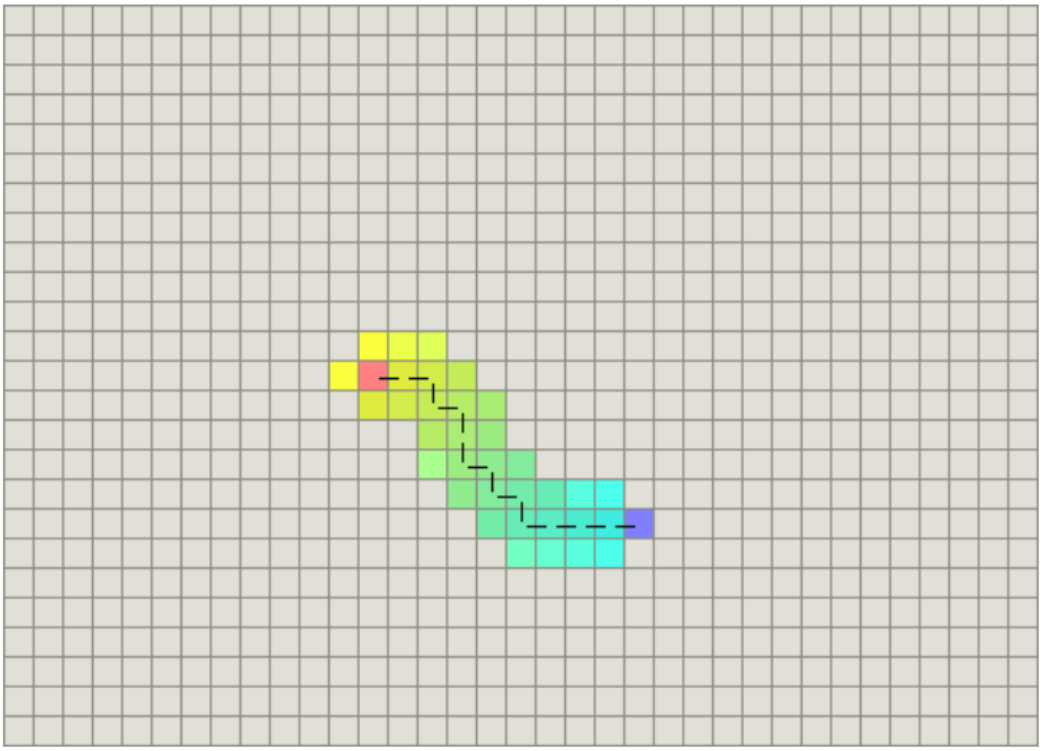

图 1-6　A*算法在无障碍时搜索路径

在遇到凹型障碍物的例子中，A*找到一条和 Dijkstra 算法一样好的路径，如图 1-7 所示。

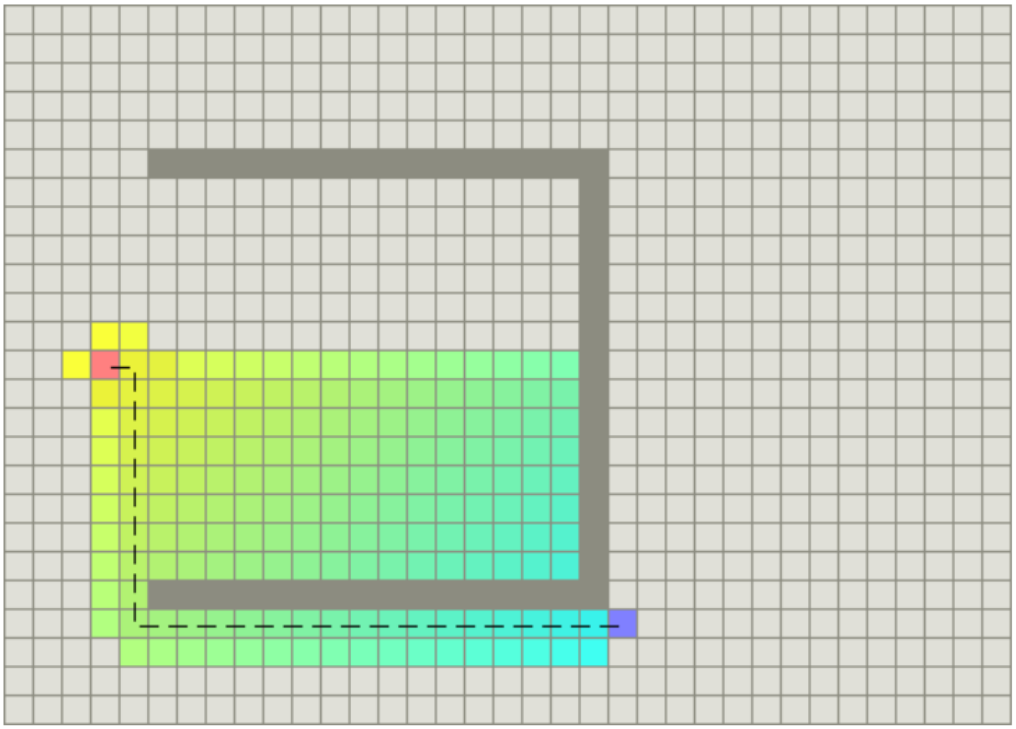

图 1-7　A*算法遇凹型障碍物时搜索路径

成功的秘决在于，它把 Dijkstra 算法（靠近初始点的结点）和 BFS 算法（靠近目

标点的结点）的信息块结合起来。在讨论 A*的标准术语中，g(n)表示从初始结点到任意结点 n 的代价，h(n)表示从结点 n 到目标点的启发式评估代价（heuristic estimated cost）。在上图中，yellow(h)表示远离目标的结点而 teal(g)表示远离初始点的结点。当从初始点向目标点移动时，A*权衡这两者。每次进行主循环时，它检查 f(n)最小的结点 n，其中 f(n) = g(n) + h(n)。

第 2 章　A*算法在网格图中的应用

2.1　启发式函数的选取

启发式函数 h(n)告诉 A*从任意结点 n 到目标结点的最小代价评估值。选择一个好的启发式函数是非常重要的。

启发式函数可以控制 A*的行为：

1）一种极端情况，如果 h(n)是 0，则只有 g(n)起作用，此时 A*演变成 Dijkstra 算法，这保证能找到最短路径。

2）如果 h(n)几乎都比从 n 点移动到目标的实际代价小（或者相等），则 A*保证能找到一条最短路径。h(n)越小，A*扩展的结点越多，运行就得就越慢。

3）如果 h(n)精确地等于从 n 点移动到目标的代价，则 A*将会仅仅寻找并沿着最佳路径而不扩展别的任何结点，从而运行得非常快。尽管这不可能在所有情况下发生，但是仍可以在一些特殊情况下让 h(n)精确地等于实际值。

4）如果 h(n)有时比从 n 点移动到目标的实际代价高，则 A*不能保证找到一条最短路径，但它运行得更快。

5）另一种极端情况，如果 h(n)比 g(n)大很多，则只有 h(n)起作用，A*演变成 BFS 算法。

所以这就得到一个很有趣的情况，那就是我们可以决定我们想要从 A*中获得什么。理想情况下，我们想最快地得到最短路径。如果我们的目标太低，我们仍会得到最短路径，不过速度变慢了；如果我们的目标太高，那我们就放弃了最短路径，但 A*运行得更快。

在很多现实情况中，A*的这个特性非常有用。例如，在机场的终端区，由于起降的航空器太多，在紧急情况下，可能更希望得到一条时间更短的路径，而不是一条距离最短的路径。在这种情况下，可以通过修改 g(n)和 h(n)的权重，来实现真实情况下的要求。

在网格地图中，每个网格都对应一个结点，当网格地图给定时，每个结点的位置信息和 h(n)就已经确定了，因此，在网格地图中，我们采用两种距离以及它们对应的展开方式，来搜索最优路径。分别是曼哈顿距离（Manhattan distance）和切比雪夫距离（Chebychev distance）。

2.1.1　曼哈顿距离

标准的启发式函数是曼哈顿距离（Manhattan distance）。如图 2-1 所示，曼哈顿

距离等于当前结点到目标结点的横纵坐标的差值的绝对值，再乘以代价系数，如公式2-1。考虑代价函数并找到从一个位置移动到邻近位置的最小代价D，这里我们取D=1。因此，网格图中的启发式函数应该是曼哈顿距离的D倍：

$$h(n)=1*(|n.x-goal.x|+|n.y-goal.y|) \tag{2-1}$$

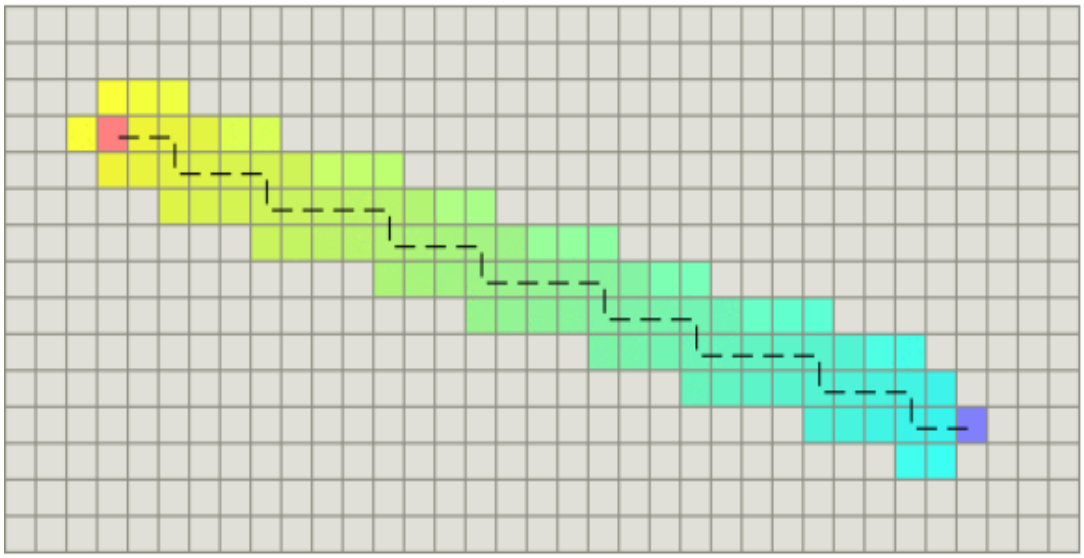

图2-1　曼哈顿距离

根据曼哈顿距离的定义可以知道，曼哈顿方法的展开方式是从当前结点开始，向上下左右四个方向分别展开，比较待展开结点的f(n)的大小，选取f(n)较小的结点优先展开，如果f(n)相同，则选取h(n)较小者优先展开，直至到达目标结点结束。

2.1.2　切比雪夫距离

切比雪夫距离（Chebychev distance）的展开方式是在曼哈顿方法的基础上又扩展了左右两条对角线上的展开，如图2-2。切比雪夫距离定义为当前结点到目标结点的横纵坐标的差值的绝对值中的较大者，再乘以代价系数。上下左右两个相邻网格之间连线的直线距离定义为 10，对角线上相邻的两个网格之间连线的距离是两网格间直线距离的根号2倍，约为14。

假设直线和对角线的代价都是D，则切比雪夫距离为：

$$h(n)=D*max(|n.x-goal.x|,|n.y-goal.y|) \tag{2-2}$$

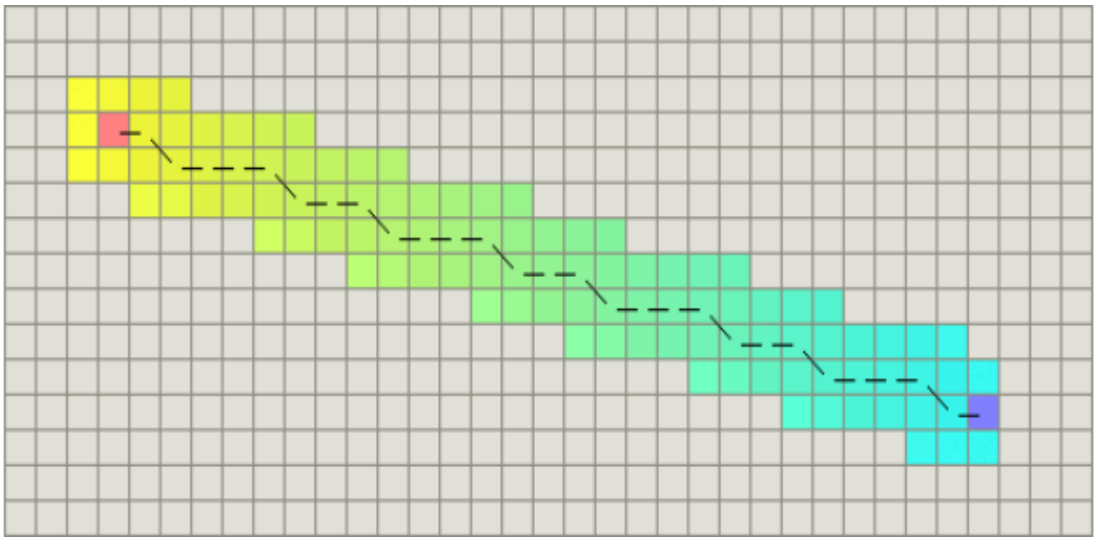

图 2-2　切比雪夫距离

2.2　A*算法的设计与实现

在网格式地图的路径搜索中，网格图在生成时就已经确定了每一个网格所对应的位置坐标信息以及距离信息，路径搜索的过程只需要在这些已知的确定点上展开搜索即可。

我们定义上下左右方向上相邻的两个网格之间的距离为 10，对角线方向上相邻的两个网格之间距离为 14。对于曼哈顿搜索方式而言，每次搜索都会基于当前结点向上下左右四个方向分别展开，四个方向所需要付出的代价均为 10；对于切比雪夫展开方式，每次搜索除了向上下左右四个方向展开之外，还会向两条对角线上相邻的四个结点展开，对角线上的结点每展开一次所需要付出的代价为 14。

按照既定的展开方式一直搜索下去，每次展开都会比较所有分支路径的 f(n)的值的大小，选取 f(n)最小者进行下一次的展开，如果最小的 f(n)有不止一个结点满足，则比较这些结点的 h(n)大小，选取 h(n)较小者进行下一次的展开，直至搜索达到目标结点时停止。

2.3　仿真结果与分析

对于一个给定的 11*11 的网格图，起点在左下角，目标点在右上角，用曼哈顿和切比雪夫两种展开方式得到的结果分别如图 2-3、2-4 所示。

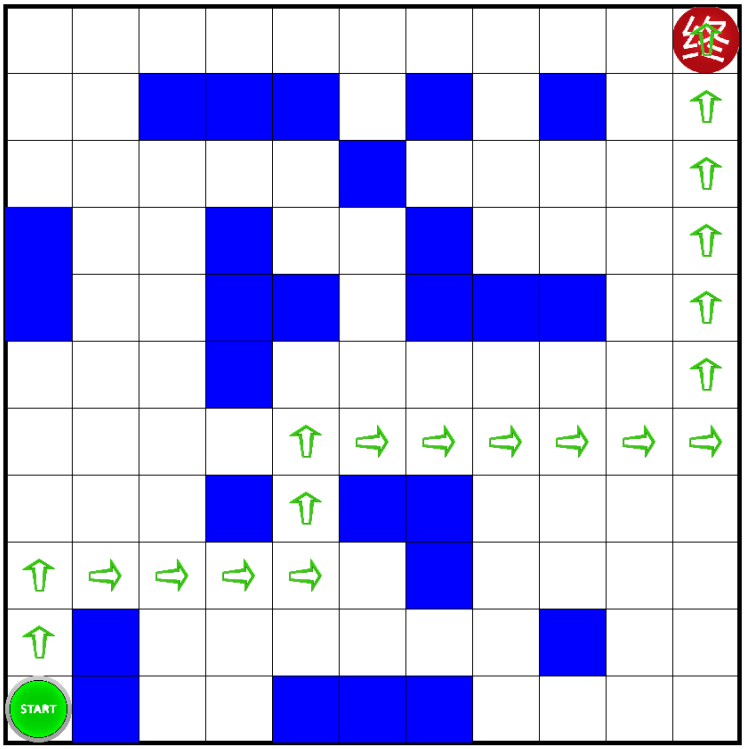

图 2-3　曼哈顿展开结果

曼哈顿展开结果中，蓝色网格表示障碍物，绿色箭头表示最终搜索得到的最优路径。经过验证，该条路径确实是起点与终点间的最短路径。

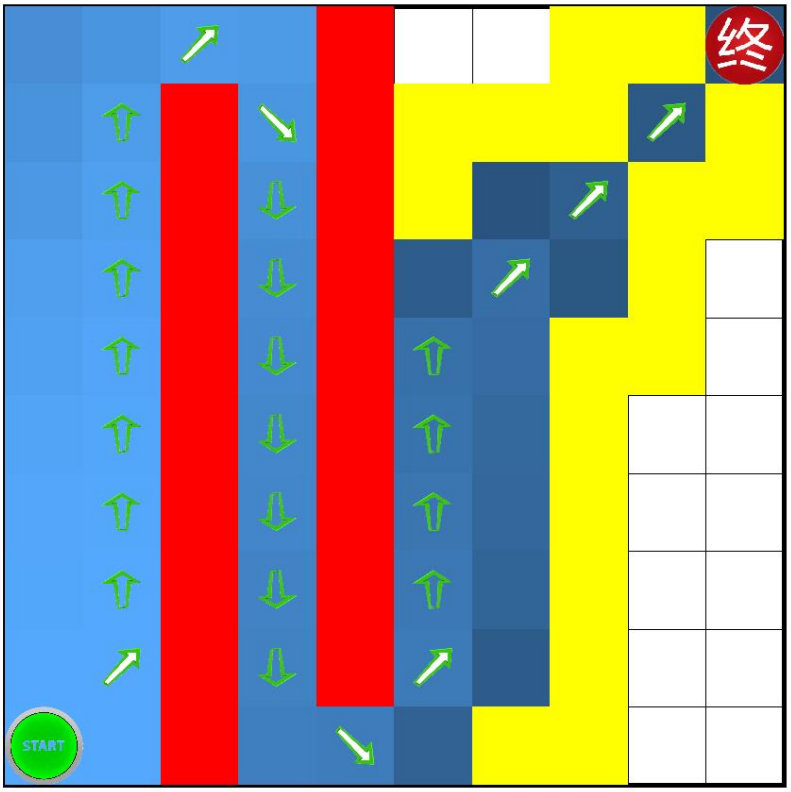

图 2-3　切比雪夫展开结果

切比雪夫展开结果中，红色网格表示障碍物，绿色箭头表示最终搜索得到的最优路径。并且，所有蓝色的网格都是在搜索过程被展开过的结点，而黄色的网格则是可以被展开却没有被展开的结点，这是因为，在比较各个待展开结点的 f(n)的值时，总是优先展开 f(n)较小者，因此，在还没有展开这些黄色结点时，就已经到达了目标结点，所以就不再展开这些结点了。从图中可以看出，黄色网格总是包围着蓝色网格，是蓝色网格的边界。也就是说，待展开结点中没有被展开的结点总是已经被展开结点的边界。

相比于曼哈顿展开方式，切比雪夫展开方式的路径更短，搜索得更快。因此，下面针对切比雪夫距离进行进一步的验证。

分别采用 20*20、30*30、150*150 的网格，起点仍在左下角，目标点在右上角，红色网格表示障碍物，蓝色网格表示在搜索过程被展开过的结点，黄色网格表示可以被展开却没有被展开的结点。得到的最优路径如下图 2-4、2-5、2-6 所示。经过验证，A*算法搜索得到的路径确实是一条距离最短的最优路径，且黄色网格总是蓝色网格的边界。

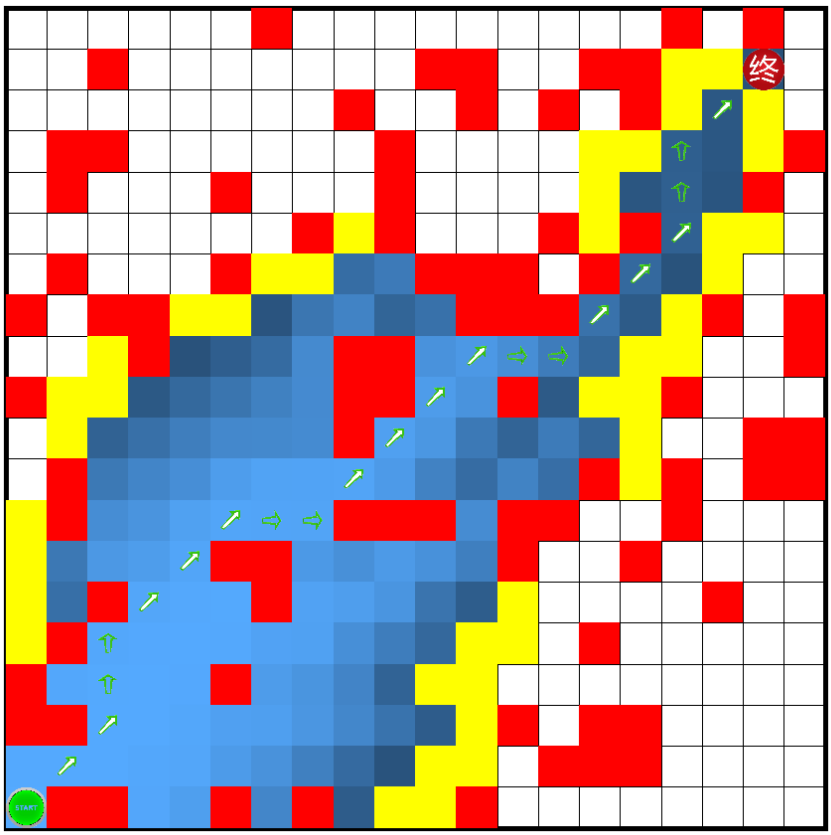

图 2-4　切比雪夫展开（20*20 网格）

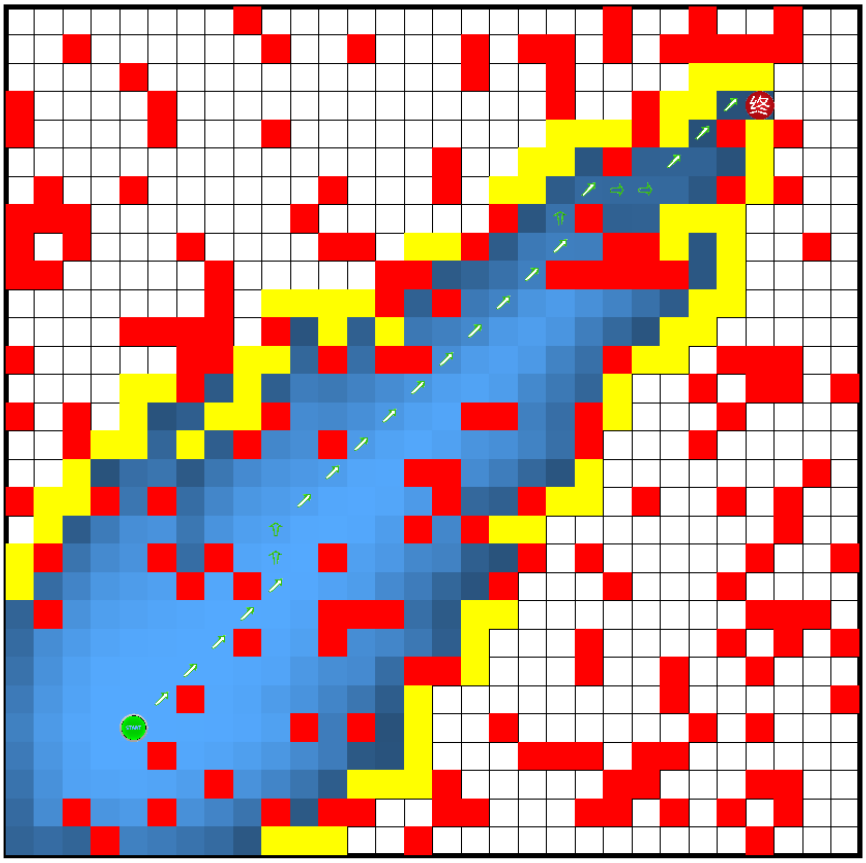

图 2-5　切比雪夫展开（30*30 网格）

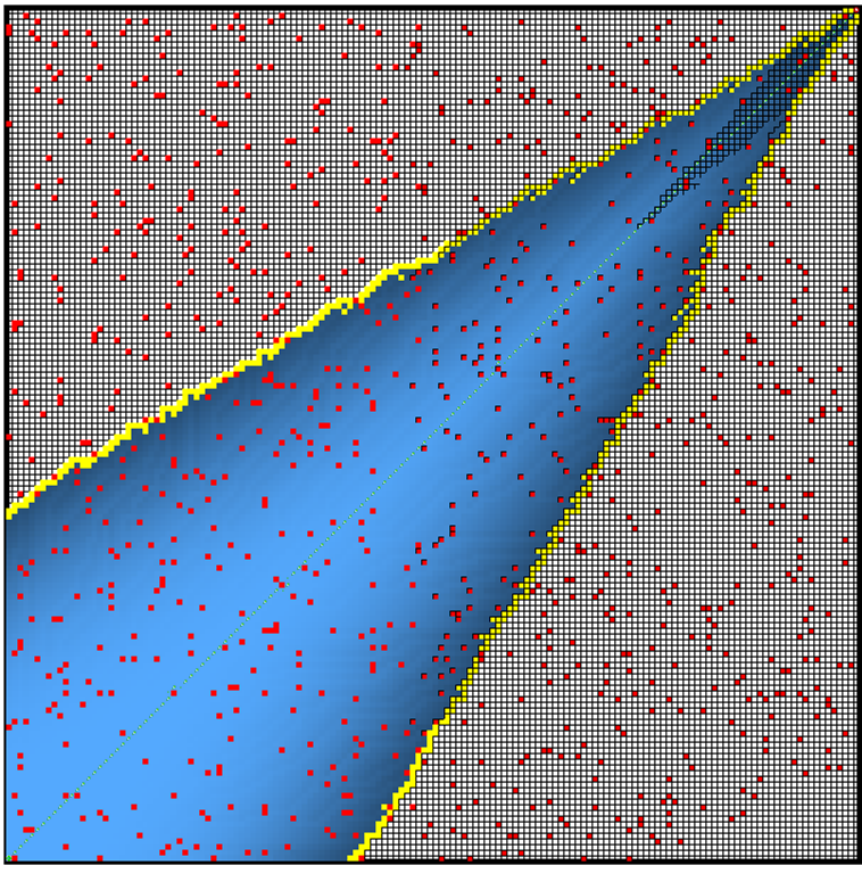

图 2-6　切比雪夫展开（150*150 网格）

在进行 100*100 的网格图路径设计时，出现了如图 2-7 所示的情况：即目标点被障碍物堵住了。所有没有被障碍物堵住的结点都呈现蓝绿色，也就是说，A*算法搜索了所有可以搜索到的结点之后，仍然找不到一条可以到达目标点的路径。

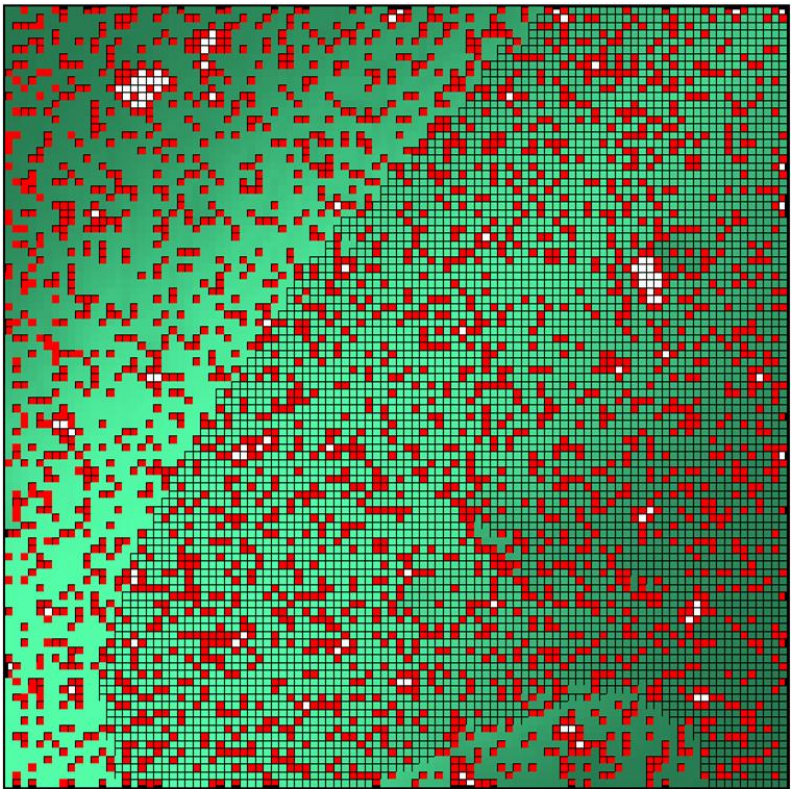

图 2-7　切比雪夫展开（100*100 网格）

第 3 章　A*算法在二维航路设计中的应用

3.1　模型假设

对于终端区的飞机进场离场建立物理模型如下：

- 机场终端区整合为二维平面；
- 飞机结点的偏航角不超过 30 度；
- 飞机在搜索步长内飞行轨迹几乎为直线，不考虑终端雷达的探测误差；
- 终端区附近复杂的障碍物简化拟合为椭圆形、圆形、线段及多边形；
- 达到机场判定方式简化为飞机与机场中心距离小于某一门限值（忽略飞机跑道方向因素）。

3.2　启发式函数的选取

航路设计与网格地图路径设计的最大区别是，网格地图中的每一个网格就对应一个结点，网格在生成时，每个结点对应的位置信息和 h(n)的值就随之确定了。而二维平面中，没有既定的结点，因此，点与点之间的距离最好的表示方式就是欧氏距离。因此，二维平面航路设计中采用的启发式函数 h(n)为当前结点到目标结点之间的欧氏距离。

$$h(n)=\sqrt{(n.x-goal.x)^2+(n.y-goal.y)^2} \tag{3-1}$$

3.3　点与障碍物的位置判定

根据模型假设，障碍物可以简化为多边形、圆、椭圆、一维线段。下面的位置判定方法主要是针对这四种形状的障碍物做研究。

3.3.1　点与多边形障碍物的位置关系

多边形障碍物是最为复杂的一种障碍物。本文采用的是“引射线法（ray casting）”来判定搜索点与多边形障碍物的位置关系。引射线法的基本原理即为：从该点向外一条射线，计算它与多边形边界的交点个数，如果交点个数为奇数，那么搜索点在多边形内部，否则搜索点在多边形外部。

下面对“引射线法”的基本原理做详细解释。

首先，对于平面内任意闭合曲线，我们都可以直观地认为，曲线把平面分割成了内、外两部分，其中“内”就是我们所谓的多边形区域。

基于这一认识，对于平面内任意一条直线，我们可以得出下面这些结论：

- 直线穿越多边形边界时，有且只有两种情况：进入多边形或穿出多边形。

• 在不考虑非欧空间的情况下，直线不可能从内部再次进入多边形，或从外部再次穿出多边形，即连续两次穿越边界的情况必然成对。

• 直线可以无限延伸，而闭合曲线包围的区域是有限的，因此最后一次穿越多边形边界，一定是穿出多边形，到达外部。 示意图如图 3-1 所示。

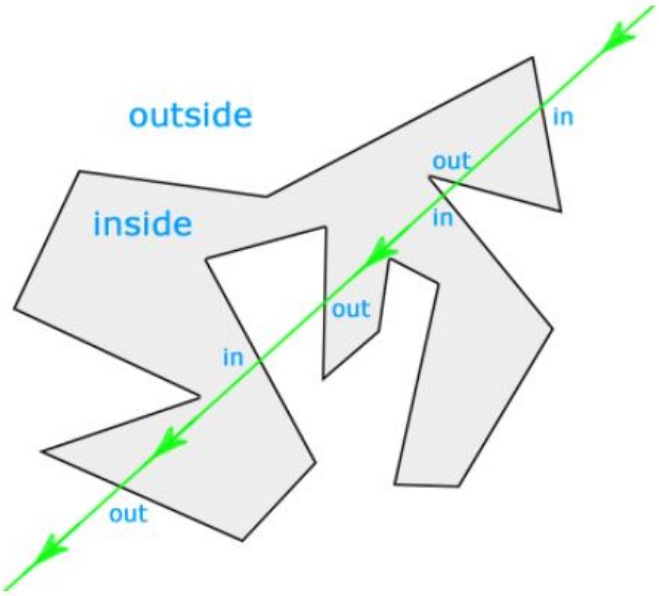

图 3-1　直线与多边形交点情况

把上面这些结论综合起来，我们可以归纳出：

当射线穿越多边形边界的次数为偶数时，所有第偶数次（包括最后一次）穿越都是穿出，因此所有第奇数次（包括第一次）穿越都为穿入，由此可推断点在多边形外部。如图 3-2 所示。

当射线穿越多边形边界的次数为奇数时，所有第奇数次（包括第一次和最后一次）穿越都是穿出，由此可推断点在多边形内部。如图 3-3 所示。

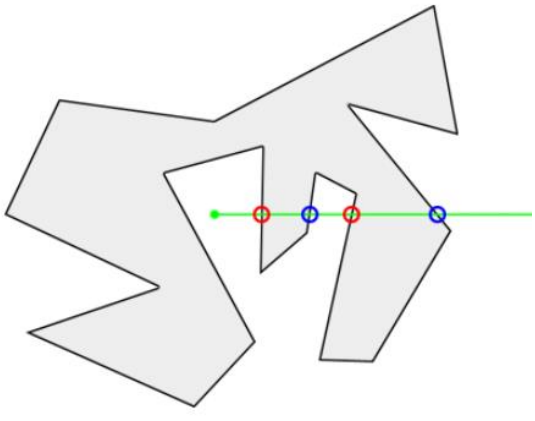

图 3-2　点在多边形外部

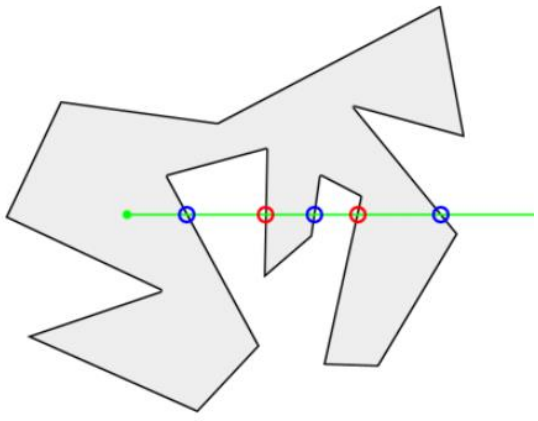

图 3-3　点在多边形内部

在搜索过程中，可能会遇到以下几种特殊情况：

1）点在多边形的边上

一个点在线段所在的直线上的条件是，该点与线段的两个端点做叉乘的乘积为

零。因此，利用搜索点和多边形各个顶点两两做叉乘，如果结果为零，则说明搜索点在该条线段所在的直线上，否则点不在线段边上。进一步，要想证明点在该条线段上，只需要让该点位于线段所在的坐标系中围成的长方形中即可。如图 3-4 所示。

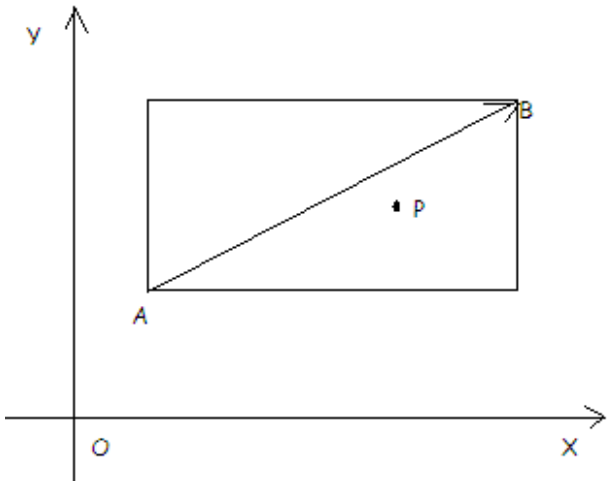

图 3-4　判断点在线段上

2）点和多边形顶点重合

判断搜索点是否和障碍物顶点重合，只需要比较点的坐标即可。

3）射线经过多边形顶点

首先，射线穿越一条线段的前提条件是线段的两个端点分别在射线两侧。因此，我们规定被射线穿越的点都算作位于射线的其中一侧即可。

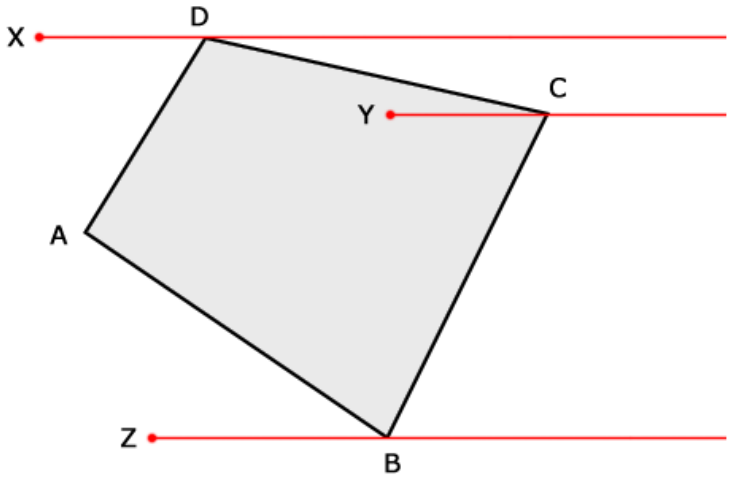

图 3-5　射线经过顶点

如图 3-5 所示，假如我们规定射线经过的点都属于射线以上的一侧，显然点 D 和发生顶点穿越的点 C 都位于射线 Y 的同一侧，所以射线 Y 其实并没有穿越 CD 这条边。而点 C 和点 B 则分别位于射线 Y 的两侧，所以射线 Y 和 BC 发生了穿越，由此我们可以断定点 Y 在多边形内。同理，射线 X 分别与 AD 和 CD 都发生了穿越，因此点 X 在多边形外，而射线 Z 没有和多边形发生穿越，点 Z 位于多边形外。

4）射线经过多边形的一条边

解决了第三种特殊情况，这一点就不难解决了。根据上面的假设，射线连续经过的两个顶点显然都位于射线以上的一侧，因此射线经过一条边的情况就看作没有发生穿越即可。由于第三点的解决方案实际上已经覆盖到这种特例，因此不需要再做特别的处理。

3.2.2 点与特殊形状障碍物的位置关系

1）搜索点与圆形障碍物的位置关系

对于圆形障碍物完全对成的形状，只需要判断搜索点到圆心的欧式直线距离即可，若欧式直线距离小于等于圆的半径，那么搜索点就在障碍物内部，否则就在障碍物外部。

2）搜索点与椭圆型障碍物的位置关系

对于椭圆型障碍物，我们可以将它简化近似看作为以长轴为长边，短轴为短边的一个矩形，从而就可以用上一小节中提出的点与多边形障碍物的位置判定方法来判定搜索点是否在障碍物内部。

3）搜索点与一维障碍物的位置关系

对于一维障碍物，我们将它简化为一条没有粗细厚度的线段，线段以两个端点来定义，从而可以利用上一小节中提出的点在多边形一条边上的判定方法来判断搜索点是否位于一维障碍物上。

3.4 搜索路径与障碍物的相交性判定

根据模型假设，这里障碍物仍然简化为一维线段、圆、椭圆、多边形。下面主要是针对这四种障碍物做研究。

3.4.1 搜索路径与一维线段障碍物的相交性

判断搜索路径与一维线段的相交与否，即为判断两条线段的相交性。两条线段有且仅有一个公共点，且这个点不是任何一条线段的端点时，称这两条线段是严格相交的。也就是说线段不严格相交时可以将端点作为交点，但本文中，若交点为线段的一个端点时，我们可以用上一节中判定点与多边形位置关系的方法，将该点视为与多边形相交。因此，本节不讨论不严格相交，只讨论严格相交的情况。

在判断两条线段是否相交时，我们采用快速排斥实验和跨立实验这两种方法。快速排斥实验能很快的排除掉线段不相交的情况，但并不能成为线段相交的充要条件，在快速排斥实验之后再加上跨立实验就能完全的判断两线段是否相交。

首先介绍快速排斥实验。

设以线段 P1、P2 为对角线的矩形为 R，设以线段 Q1、Q2 为对角线的矩形为 T，若 R 与 T 不相交，则两线段不可能相交。利用快速排斥实验可以很快的排除掉那些完

全没有可能相交的线段。

快速排斥实验的原理如下：

假设 P1 = (x1, y1)，P2 = (x2, y2)，Q1 = (x3, y3)，Q2 = (x4, y4)，设矩形 R 的 x 坐标的最小边界为 minRX = min(x1, x2)，以此类推，将矩形表示为如下形式：

$$R=(minRX,minRY,maxRX,maxRY) \tag{3-2}$$

若两矩形相交，则相交的部分构成了一个新的矩形 F，如图 3-6 所示，我们可以知道 F 的四个顶点分别为

$$minFX=max(minRX,minTX) \tag{3-3}$$

$$minFY=max(minRY,minTY) \tag{3-4}$$

$$maxFX=min(maxRX,maxTX) \tag{3-5}$$

$$maxFY=min(maxRY,maxTX) \tag{3-6}$$

得到 F 的各个顶点值之后，只要判断矩形 F 是否成立就知道 R 与 T 是否相交，若 $minFX>maxFX$ 或 $minFY>maxFY$，则 F 无法构成，R 与 T 不相交，否则 R 与 T 相交。

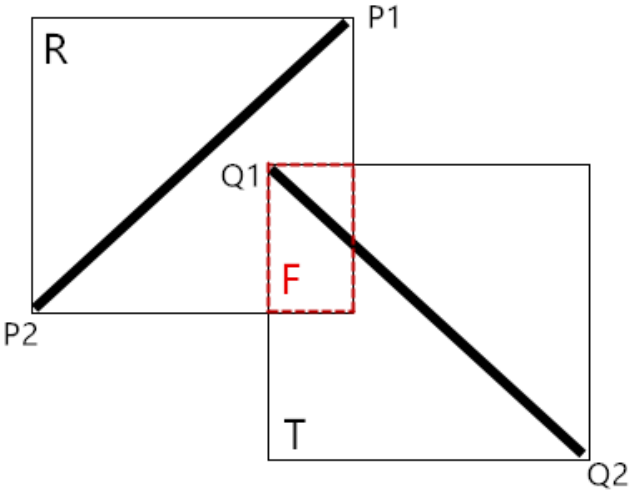

图 3-6　快速排斥实验

下面介绍跨立实验。

如果一条线段 P1P2 跨过了线段 Q1Q2,那么 P1、P2 就分布在线段 Q1Q2 的两边，那么有

$$[(P1-Q1)\times(Q2-Q1)]\cdot[(Q2-Q1)\times(P2-Q1)]>0 \tag{3-7}$$

其中，“×”是叉乘，而“·”是点乘。

式中，（P1-Q1）×（Q2-Q1）能得出线段 Q1P1 与线段 Q1Q2 叉乘的结果的向量的方向，（Q2-Q1）×（P2-Q1）同样也会得出线段 Q1Q2 和线段 Q1P2 叉乘的结果的向量的方向，只有这两个结果向量的点乘的结果大于 0，即这两个结果向量的方向一致时，P1、P2 才会分布在线段 Q1Q2 的两边。如图 3-7 所示。

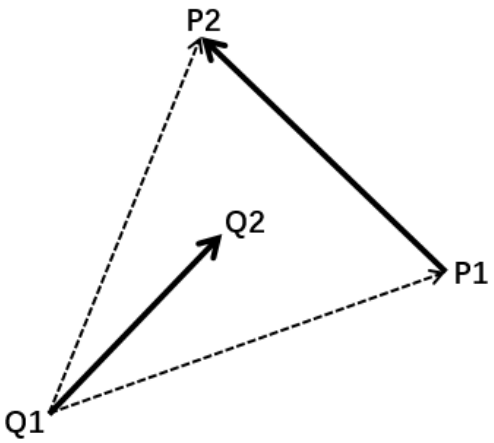

图 3-7　跨立实验

同理 Q1Q2 分布在 P1P2 两边的条件为

$$[(Q1-P1)\times(P2-P1)]\cdot[(P2-P1)\times(Q2-P1)]>0 \tag{3-8}$$

同时满足以上两个条件时，线段 Q1Q2 与线段 P1P2 才会相交。

3.4.2　搜索路径与特殊形状障碍物的相交性

1）搜索路径与圆的相交性

首先，把搜索路径看作一条有限线段，求出圆心到该线段的最短距离（垂线段距离或圆心到某一端点的距离），通过比较最短距离与圆的半径的大小关系，就可以知道搜索路径与圆是否相交。若最短距离大于圆的半径，则搜索路径与圆不相交，否则搜索路径会穿过圆形障碍物。

2）搜索路径与椭圆的相交性

同样，把搜索路径看作一条有限线段，再把椭圆形障碍物近似为一个以椭圆长半轴为长、短半轴为宽的一个矩形。 从而，通过判断搜索路径与矩形各个边的相交与否，即可以近似知道搜索路径是否穿过椭圆。

3）搜索路径与多边形的相交性

仍然把搜索路径看作一条有限线段，通过判断搜索路径与多边形各个边的相交与否，即可以知道搜索路径是否穿过椭圆。

3.5　A*算法的设计与实现

航路设计与网格式地图中路径设计的主要区别在于，网格式地图在生成时就已经确定了每一个网格所对应的位置坐标信息以及距离信息，路径搜索的过程只需要在这些已知的确定点上展开搜索即可。而航路设计中的已知信息只有起点和终点的位置坐标，以及平面中既定的障碍物的位置信息，并没有其他任何直接拿来用的搜索结点。因此，需要采用一种新的结点搜索方式。

我们定义一个搜索角度θ和一个搜索步长 d，基于初始的模型假设，飞机偏航角

不超过 30 度，在规定每次搜索展开 5 个结点的前提下，搜索角度θ选取不能超过 15 度。搜索步长则根据起点和终的相对距离而定，步长太大会导致航路设计拐角过多，不适合实际飞行中飞机的转弯运动情况，而步长太小则会导致搜索时间过长，加大迭代次数和计算量，给航路设计系统增加负担。

根据已给的搜索角度θ和搜索步长 d，每一次搜索都会临时生成 5 个新结点，新节点中包含了自身的位置坐标、航向角等信息，直至搜索达到目标结点停止。

3.5 仿真结果与分析

在二维平面地图中，利用 Java 实现的路径搜索结果示例如下图 3-8、3-9、3-10、3-11 所示。其中，起点设在左下角，目标点设在右上角，两者都为黄色的圆，蓝色的图案表示障碍物，红色的连线即为搜索出的最优航路。

除此之外，蓝色的结点为被展开搜索过的结点，绿色的结点为待展开却没有被展开的结点。和网格地图中的结果类似，待展开却没有被展开的结点总是被展开搜索的结点的边界，即绿色结点总是蓝色结点的边界。

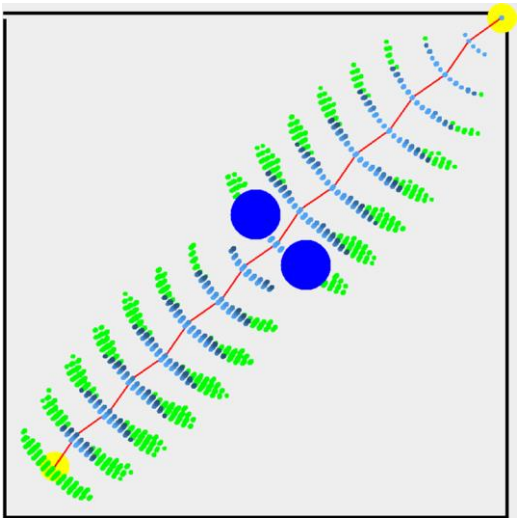

图 3-8　航路设计结果示例

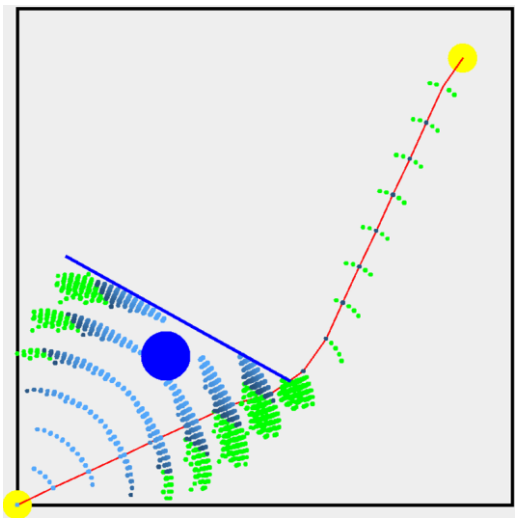

图 3-9　航路设计结果示例

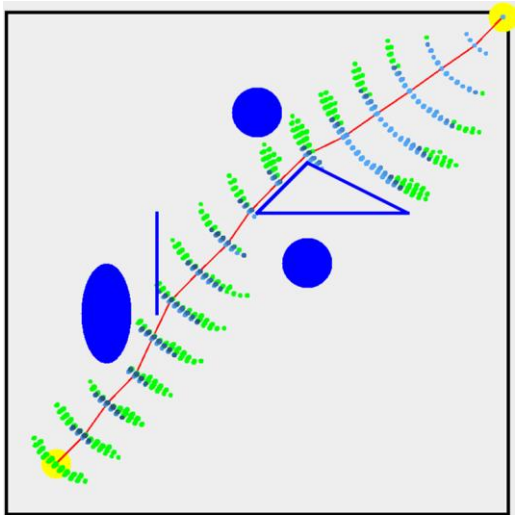

图 3-10　航路设计结果示例

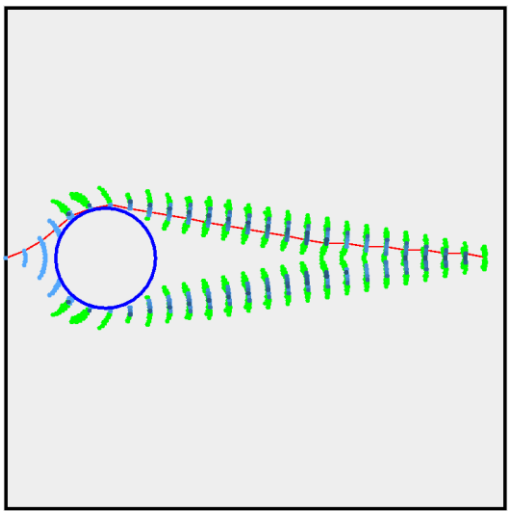

图 3-11　航路设计结果示例

第 4 章　航路设计中 A*算法的优化

由上一节中最后的图 3-10、3-11 中可以看出，3-10 的示例中，三角形障碍物右上方有一些绿色和蓝色的结点在搜索的过程中被展开了，3-11 示例中，由于障碍物是对称的，因此，下方展开搜索的结点和上方的是完全一样的。但是从最后的结果路径来看，这些点都是没有必要被展开的，展开的点越多，程序运行的越慢，路径搜索耗费时间越久。因此本章提出两种 A*算法的优化方案，分别是深度优先搜索优化，和 Chebychev 优化。

4.1　深度优先搜索优化

深度优先搜索优化算法（Depth First Search，DFS）的基本思想就是对每一个可能的分支路径深入到不能再深入为止，即优先搜索那些展开较深的结点。这样可以在几个结点有相同的 f(n)值时，优先展开靠近目标点的结点，从而避免展开一些不必要展开的结点。

在同样的起终点、障碍物的情况下，深度优先搜索优化算法（DFS）得到的结果与 A*算法的结果对比如下图 4-1、4-2 所示。

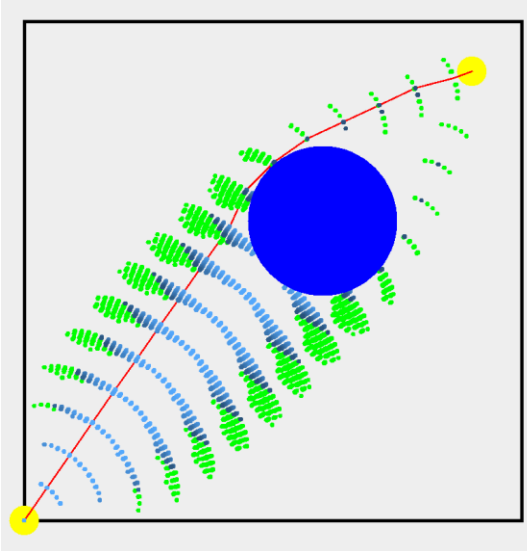

图 4-1　A*算法搜索结果

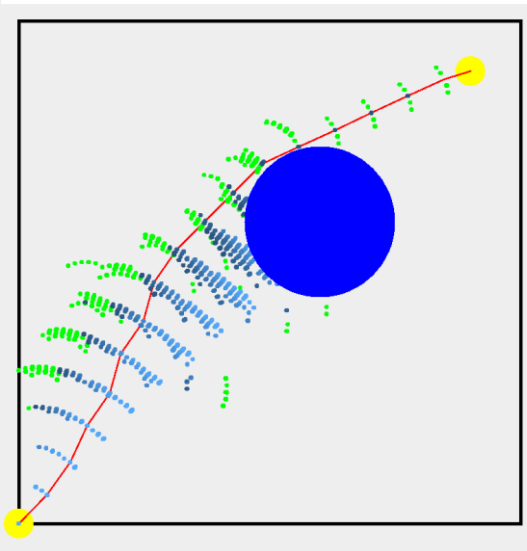

图 4-2　DFS 优化搜索结果

两图对比可知，DFS 优化算法搜索到的最优路径和 A*算法的结果路径相差不大，但是 DFS 算法展开的结点远远少于 A*优化展开的结点。这一点主要体现在搜索时间上，A*算法耗时 57600 毫秒，而 DFS 优化算法仅耗时 3445 毫秒，这充分体现出 DFS 算法在搜索时长方面的优势。

4.2　Chebychev 优化

Chebychev 优化算法的基本思想就是先利用网格式地图中的 Chebychev 展开方式

进行路径粗选，找出一条可以从起点达到终点的较优路径作为指示路径，然后在指示路径附近利用 A*算法搜索一条最优路径。这个方案同样可以避免展开一些不必要展开的结点。

在同样的起点终点、障碍物的情况下，Chebychev 优化算法得到的结果与 A*算法的结果对比如下图 4-3、4-4 所示。其中，Chebychev 优化搜索结果图中，偏折较大的那条路径是利用 Chebychev 算法在网格图中的思想找到的一条较优的指示路径，较平滑的那条是最终的最优路径。

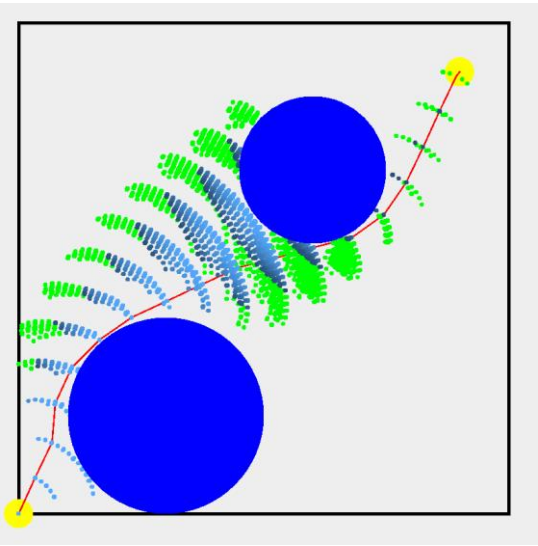

图 4-3　A*算法搜索结果

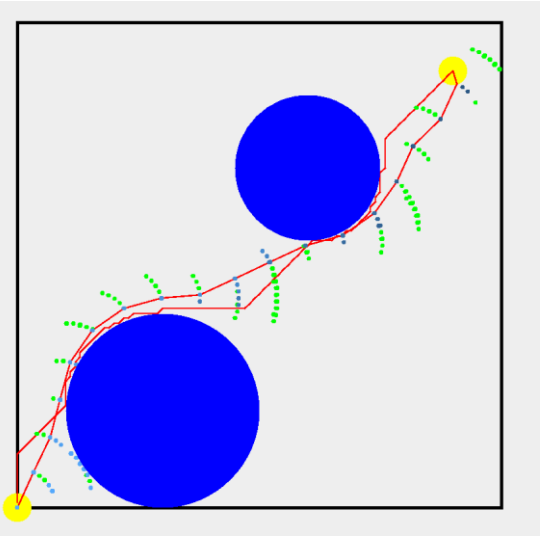

图 4-4　Chebychev 优化搜索结果

两图对比可知，Chebychev 优化算法搜索到的最优路径和 A*算法的结果路径也相差不大，但是 Chebychev 算法展开的结点相比于 DFS 优化算法，更加远远少于 A*优化展开的结点。

4.3　两种优化算法的对比

上面提出的两种优化算法都有各自的优点，考虑到在航路设计中，路径距离和搜索时间是的两个非常主要的因素，下面将两种优化算法与原有的*算法放在同样的地图情况下，分别从最优路径和搜索时间两个方面进行算法的比较与分析。

4.3.1　最优路径的对比

在同样的起点终点、同样的障碍物的情况下，A*算法、DFS 优化算法、Chebychev 优化算法搜索得到的结果分别如下图 4-6、4-7、4-8 所示。

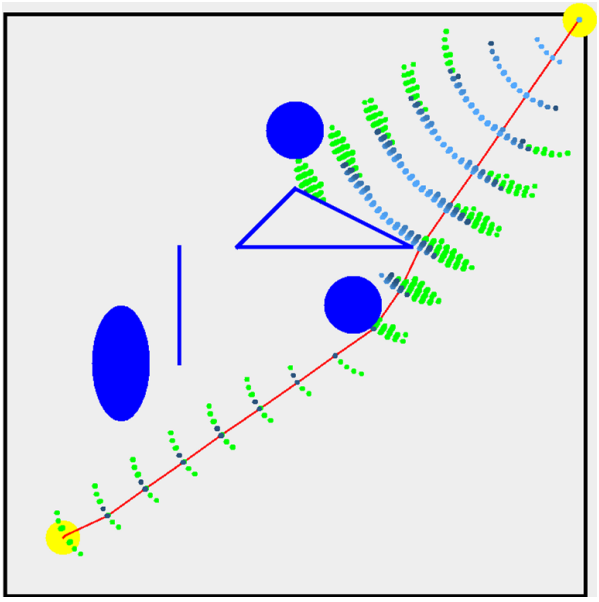

图 4-5　A*算法结果

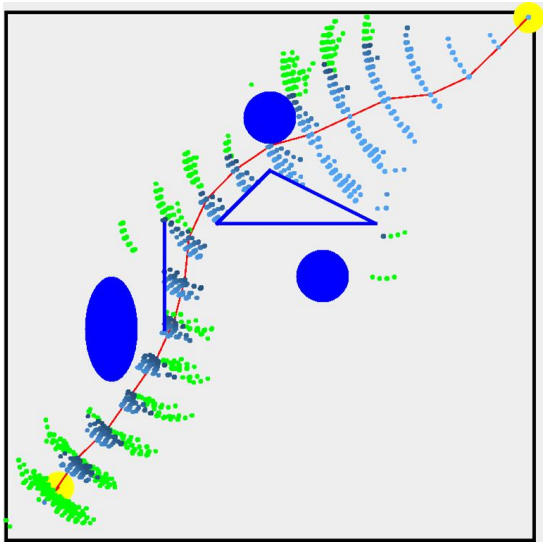

图 4-6　DFS 优化算法搜索结果

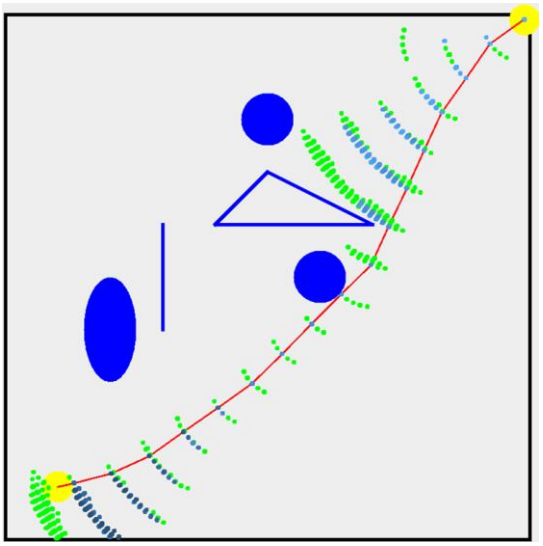

图 4-7　Chebychev 优化算法搜索结果

通过对比三个算法的结果可以看出，深度优先搜索算法（DFS）找到的最优路径和 A*算法的路径并不一样，DFS 算法选择了穿过障碍物去绕行。显然 DFS 优化算法找到的这条路径不是距离最短的，但是它保证找到的是一条可行的路径。除此之外，DFS 算法与 A*算法展开的结点大致相差不多，这种情况下，可以根据两者的搜索时间来做进一步的分析。

然而 Chebychev 算法找到的最优路径与 A*算法的路径几乎相同，且是距离最优的，并且 Chebychev 算法展开的结点远远少于 A*算法展开的结点。因此，这种情况下，Chebychev 优化算法相比较其他两者是最优的。

4.3.2 搜索时间的对比

同样地，我们为三种算法设置相同的起点终点、同样的障碍物，A*算法、DFS 优化算法、Chebychev 优化算法搜索得到的路径结果和耗时对比分别如下图 4-8、4-9、4-10 所示。

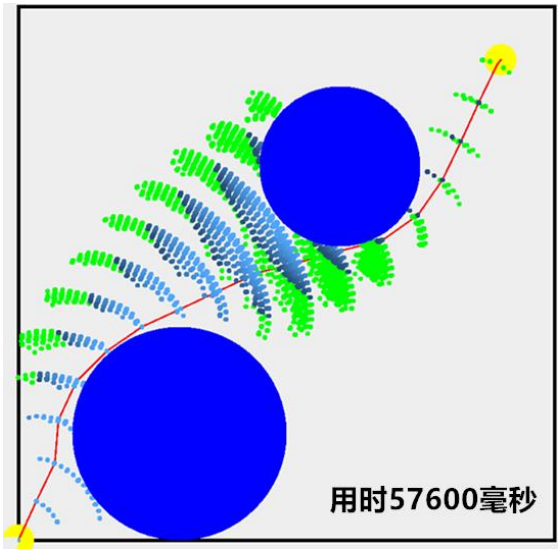

图 4-5　A*算法结果

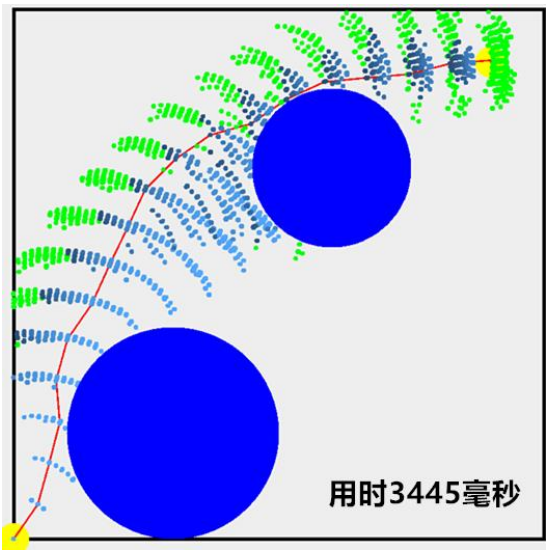

图 4-6　DFS 优化算法搜索结果

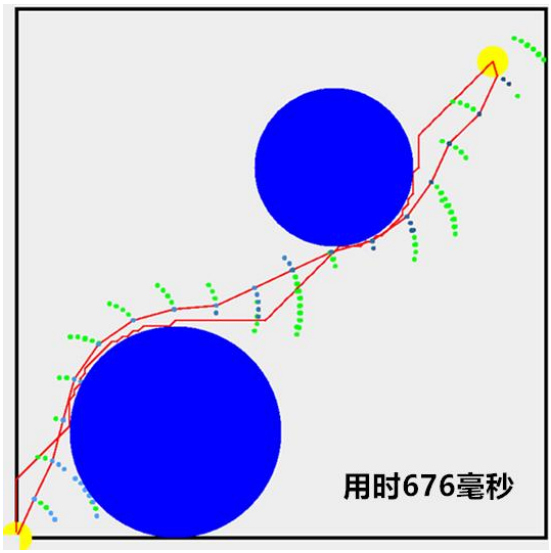

图 4-7　Chebychev 优化算法搜索结果

通过对比三个算法的结果可以看出，深度优先搜索算法（DFS）找到的最优路径和 A*算法的路径并不一样，但是此种情况下，DFS 算法比 A*算法展开的结点要少很多，A*算法搜索用时 57600 毫秒，DFS 优化算法搜索用时 3445 毫秒，DFS 优化算法搜索用时减少了近 17 倍。

而 Chebychev 算法找到的最优路径与 A*算法的路径几乎相同，并且 Chebychev 算法展开的结点远远少于 A*算法展开的结点，Chebychev 优化算法搜索用时只需 676

毫秒，比 A*算法用时减少了近 85 倍，比 DFS 优化算法用时减少了近 5 倍。

在真实的终端区航路设计中，如果遇到多架航空器同时进近的情况，需要我们在最短的时间内规划出每架航空器的最优路径，在这种情况下，规划搜索时间是限制因素，因此，在这种情况下，DFS 优化算法和 Chebychev 优化算法都适用，具体选用哪一种算法，要根据规划出的航路的安全性与可行性来决定。不过，大部分情况下，Chebychev 算法相比较其他两者是更优的，并且是搜索效率最高的。

第 5 章　总结与展望

5.1　研究工作总结

本文从最基本的路径搜索算法入手，引出了 A*算法的来源，对 A*算法在最优路径设计中的应用进行了相关的研究，主要研究两种情况，网格式地图中的路径搜索、和二维平面中的航机设计，并在此基础上进一步提出了两种改进算法。具体的研究概括为：

1. 从常用的路径搜索算法入手，引出了 A*算法的来源：把启发式方法（heuristic approaches）如最佳优先搜索算法（DFS），和常规方法如 Dijsktra 算法结合在一起。结合了二者的优点，既可以找到一条距离最短的路径，也可以使搜索方向始终朝着目标结点前进。

2. 首先研究了 A*算法在网格式地图中的路径搜索问题，通过两种不同的启发式函数的选取：曼哈顿距离（Manhattan distance）和切比雪夫距离（Chebychev distance），以及两者对应的不同搜索方式，来研究最短路径的搜索。通过对比两种搜索方式下的最优结果路径以及在搜索过程中展开的结点，我们可以知道，A*算法确实能找到一条距离最有的路径。

3. 其次，进一步地研究了真实情况下的二维平面中的航路设计问题，采用搜索步长和搜索角度代替网格图中的各个网格结点，将障碍物模型化为多边形、圆、椭圆、一维线段等几何图形，充分利用多边形的各种性质，结合引射线法、跨立实验等多种方法，重点研究了搜索路径过程中的避障问题。搜索得到的最优结果路径表明，A*算法确实能有效地找到一条规避障碍物的最优路径。

4. 考虑到A*算法有时会展开一些不必要展开的结点，加大了计算量和迭代次数，使得搜索时间太长，因此，提出了两种基于 A*算法的航路设计中的优化方案，分别是：深度优先搜索优化算法（Depth First Search，DFS）、和 Chebychev 优化算法。深度优先搜索优化算法的基本思想就是对每一个可能的分支路径深入到不能再深入为止，即优先搜索那些展开较深的结点。这样可以在几个结点有相同的代价值时，优先展开靠近目标点的结点，从而避免展开一些不必要展开的结点。Chebychev 优化算法的基本思想就是先利用网格式地图中的 Chebychev 展开方式进行路径粗选，找出一条可以从起点达到终点的较优路径作为指示路径，然后在指示路径附近利用 A*算法搜索一条最优路径。这个方案同样可以避免展开一些不必要展开的结点。

5. 在同样的起点终点、同样的障碍物分布的地图中，分别用这两种优化算法以及最初的 A*算法进行实验，来寻找最终的最优航路。通过对比三个算法的结果可以

看出，深度优先搜索算法找到的最优路径和 A*算法的路径有时不完全一样，而 Chebychev 优化算法找到的最优路径与 A*算法的路径是几乎相同。三种算法中，深度优先算法得到的最优路径的距离要比 A*算法和 Chebychev 算法得到的路径要远一些。但是另一方面，深度优先搜索算法比 A*算法展开的结点要少很多，Chebychev 算法展开的结点更是远远少于深度优先搜索算法。更直观的体现是在时间方面，DFS 优化算法搜索用时比 A*算法减少了近 17 倍，而 Chebychev 优化算法搜索用时比 A*算法用时减少了近 85 倍，比 DFS 优化算法用时减少了近 5 倍。由此可以得出结论，大部分情况下，Chebychev 优化算法相比较其他两者是更优的，并且是搜索效率最高的。

5.2 未来研究期望

本文对二维平面航路设计中的 A*算法进行了优化与改进，主要针对展开点过多，搜索用时过长提出了新的展开思路。改进后的算法在测试中的结果也较为理想，但是目前研究的仍然是在二维平面中的航路设计，现实中更多情况下需要考虑是的三维空间中的最优航路设计问题。因此，未来关于三维空间中的最优航路设计必将进一步深入，使得 A*算法以及改进后的优化方案更加符合实际要求。

致　　谢

经过对已有工作的总结和整理分析后，完成了这篇论文。在建模仿真设计和论文的写作过程中，我得到了老师和同学们的很多帮助和支持。

首先感谢周隽老师对我们的严谨教学与悉心指导，在周老师的耐心指导下，这门课使我们提高了分析问题和解决问题的能力。

还要感谢我的组员李璐君同学在工作中给予的帮助与合作，以及其他同学在日常学习中提供的帮助，大家一起探讨问题同时也丰富了我的知识。

参考文献

[1] 张锦明,洪刚,文锐,王学涛.Dijkstra 最短路径算法优化策略[J].测绘科学,2009,34(05):105-106+99.

[2] 段莉琼,朱建军,王庆社,马玲.改进的最短路径搜索 A*算法的高效实现[J].海洋测绘,2004(05):20-22.

[3] 熊伟,张仁平,刘奇韬,王贵新.A*算法及其在地理信息系统中的应用[J].计算机系统应用,2007(04):14-17.

[4] 史辉,曹闻,朱述龙,朱宝山.A~*算法的改进及其在路径规划中的应用[J].测绘与空间地理信息,2009,32(06):208-211.

[5] 张慧. 终端区空域结构优化理论与方法[D].南京航空航天大学,2009.

[6] A*算法改进算法及其应用_张仁平.pdf

[7] 李伟,王仲生.A 算法在终端区飞机排序中的应用[J].科学技术与工程,2007(11):2594-2598

[8] 占伟伟,王伟,陈能成,王超.一种利用改进 A*算法的无人机航迹规划[J].武汉大学学报(信息科学版),2015,40(03):315-320.

[9] Jung-Ying Wang. Implementation and Comparison the Dynamic Pathfinding Algorithm and Two Modified A~* Pathfinding Algorithms in a Car Racing Game[A]. IEEE.Proceedings of 2011 4th IEEE International Conference on Computer Science and Information Technology(ICCSIT 2011) VOL06[C].IEEE:,2011:5.

[10] Jihye Hong,Kisung Park,Yongkoo Han,Mostofa Kamal Rasel,Dawanga Vonvou,Young-Koo Lee. Disk-based shortest path discovery using distance index over large dynamic graphs[J]. Information Sciences,2017,382-383.