

# A\*算法在矢量地图最优路径搜索中的应用

刘浩, 鲍远律

(中国科学技术大学自动化系, 安徽 合肥 230027)

**摘要:** A\*算法是启发式搜索算法的一种, 广泛应用于图搜索和路径规划。本文简要介绍了A\*算法, 并通过对A\*算法可接纳性和一致性的分析, 说明了A\*算法是一类适合矢量地图最优路径搜索的较好算法。

**关键词:** 最优路径, A\*算法, 矢量地图, GIS

## A\* Algorithm for the Shortest Path on Vector Maps

LIU Hao, BAO Yuanlu

(Department of Automation, University of Science and Technology of China, Hefei Anhui 230027)

**【Abstract】** A\* algorithm is one of heuristic search algorithms, which commonly used in graph search and path plan. This thesis introduces A\* algorithm briefly. Through the analysis of admissibility and consistency of A\* algorithm, the author illustrates that A\* algorithm is a kind of fairly good algorithm suitable for vector maps.

**【Key words】** optimal path, A\* algorithm, vector map, GIS

### 1 问题的由来

最优路径问题是一个很古老的问题, 其原型是运筹学中的最短路径问题, 目前在互联网的寻址计算、智能交通系统, 地理信息系统等领域有着广泛的应用。现代意义上的最优路径已不再仅仅是指地理意义上的距离最短, 它还可以是指时间最少、费用最省、线路容量最大等等。但无论采用何种标准, 最优路径问题都可归结为在带权网络中寻找最短路径的研究, 即图论中的最短路径问题。

随着信息科学的发展, 地理信息系统(GIS)广泛应用于人们生产和生活的各个领域, 在各类的GIS应用中, 交通地理信息系统(GIS-T)又是其中的一个热点研究领域。当代世界各发达国家和部分发展中国家都在积极发展智能交通系统(ITS), 以缓解由于经济发展导致的出行需求增长和由于交通智能化不够导致的出行需求不能得到较好满足之间的矛盾。在ITS所包含的6个高级交通系统<sup>[1]</sup>中, 有4个与GIS-T支持下的交通网络分析直接相关。交通网络分析的一个重要组成部分是路径分析, 而路径分析的核心为最优路径算法。因此, 对最优路径问题的研究不但具有重要的理论价值, 而且具有重要的应用价值。

矢量地图是大多数GIS应用(如定位、导航、监控等等)的基础, 是对实际交通网络拓扑的一个近似反映。从而, 我们可将将在交通网络分析中的最优路径问题的研究转化为在矢量地图中求解最优路径算法的研究。Dijkstra算法是目前GIS应用领域用于

求解最短路径问题的首选算法, 同时也是经典算法, 其优势在于求解单个顶点到其余所有顶点间的最优路径, 但用以求解单对顶点间的最优路径问题时必然产生冗余, 故不是一个应用到实际路径寻优的较好算法。基于启发式搜索的A\*算法, 因其在路径搜索过程中对问题域信息的充分利用, 使得搜索的目标性更加明确, 是求解单对顶点间最优路径的一类较好算法。

### 2 最优路径的研究现状<sup>[1-5]</sup>

最优路径问题的本质是路径搜索。从搜索算法的实时性的角度可以分为两类: 静态最优路径和动态最优路径。前者在路径寻优的过程中路径的权值为定值, 后者路径的权值则可以根据交通状况实时变化, 以适应动态寻优。虽然后者更能反应实际的情况, 但是对交通环境的智能化要求也较高, 即要求能够实时接收交通信息以更新路径的权值。在国外, 动态最优路径算法研究是一个热点, 尤其是在一些发达国家。我国目前在交通网络最优路径方面的研究主要还是集中在静态最优路径算法上, 其中一个重要原因在于我国的交通智能化水平不高, 限于人口、环境、经济发展等种种因素, 我国的ITS总体上仍然还处在一个初级的发展阶段。

从问题求解目标的不同, 可将搜索算法分为求

---

基金项目: 国家自然科学基金资助项目(60272040)

作者简介: 刘浩(1980-), 男, 硕士研究生, 主研方向: 智能交通网络系统, 最优路径; 鲍远律教授。

E-mail: haohanliu@gmail.com

解单源点多汇点的最优路径和所有顶点对之间的最优路径。前者的典型代表如 Dijkstra 算法，后者的典型代表如 Floyd 算法。

从搜索算法的是否具有智能性的角度，可将其分为两大类，即盲目搜索和启发搜索。前者如 BFS 算法、DFS 算法、IDS 算法等等，其中也包括了 Dijkstra 算法(为简便起见，下称 D 算法)。后者主要是一些人工智能领域的研究成果，包括遗传算法<sup>[4]</sup>、A\*算法<sup>[5]</sup>等等。普遍认为启发搜索算法比盲目搜索算法的效率要高，因为启发搜索引入了问题域的相关信息，从而使得搜索更具针对性，这一点在稠密图搜索中表现得尤其明显。

### 3 启发式搜索简介

#### 3.1 衡量搜索算法的执行效率

一般说来，可以用以下四个指标<sup>[2]</sup>来衡量一个搜索算法执行性能的好坏：1)完备性，即如果存在一条最优路径，算法是否保证能找到这条路径？2)最优性，即找到的路径是不是最优的？3)时间复杂度，即一个搜索算法需要花费的时间代价，一般用术语搜索代价来表示。4)空间复杂度，即需要多少存储空间来执行这个搜索算法，一般用术语路径代价(指总路径长度)来表示。由于现代存储一般能满足要求，故典型的参数是搜索代价，即时间复杂度。

#### 3.2 启发式搜索<sup>[2-3]</sup>

启发式搜索(又称 best-first 搜索)来源于 AI 历史早期对 Agent 搜索“计划”模型的简化。Best-first search 包括 Greedy search、A\* search 和 Memory-bounded heuristic search 等几类。其中 Greedy search 总是扩展离目标节点最近的结点，它不是一个完备的搜索算法(因为有可能沿着一条无限的路径不断搜索下去而永远不会尝试其他的可能性)，同时它也不是一个最优的算法(类似深度优先算法)。A\*算法根据启发式函数值的大小来重排 OPEN 表，在某些场合下，通过加适当的条件，A\*算法可以既是完备的又是最优的。Memory-bounded heuristic search，顾名思义，适合那些存储空间受限的场合，主要包括 IDA\*、RBFS 以及 MA\*算法。在以时间交换空间的搜索算法中，他们的搜索代价大于 A\*算法的搜索代价。

启发式搜索的基本思想有以下三条：1)假定有一个启发式函数  $\tilde{f}$ ，可以帮助确定下一个要扩展的最优节点。采用这样的一个约定，即  $\tilde{f}$  值小表示找到好的节点。2)下一个要扩展的节点是  $\tilde{f}$  值最小的结点(这里假设扩展一个节点产生它的所有后继)。3)当下一个要扩展的节点是目标节点时过程终止。

### 4 A\*算法的简介<sup>[3]</sup>

#### 4.1 算法的提出

Nilsson 曾开发一个通用的图搜索算法。根据插入到 OPEN 表(一种表示已搜索到但尚未扩展的节点集合的数据结构)中的新产生节点重排方式的不同，这个图搜索算法可以演化成宽度优先算法(FIFO)、深度优先算法(LIFO)和 best-first 算法(节点启发式方式)。其中根据启发式函数  $\tilde{f}$  值的大小重排 OPEN 中节点的 best-first 算法称之为 A\*算法。

#### 4.2 算法步骤

OPEN：算法已搜索但尚未扩展的结点集合。

CLOSED：算法已扩展的节点集合。

算法如下：

- 1)生成一个只包含开始节点  $n_0$  的搜索图  $G$ ，把  $n_0$  放在一个叫 OPEN 的列表上。
- 2)生成一个列表 CLOSED，它的初始值为空。
- 3)如果 OPEN 为空，则失败退出。
- 4)选择 OPEN 上的第一个节点，把它从 OPEN 上移入 CLOSED 中，称该节点为  $n$ 。
- 5)如果  $n$  是目标节点，顺着  $G$  中，从  $n$  到  $n_0$  的指针找到一条路径，获得解决方案，成功退出(该指针定义了一个搜索树，在第 7 步中建立)。
- 6)扩展节点  $n$ ，生成其后继节点集  $M$ ，在  $G$  中， $n$  的祖先不能在  $M$  中，在  $G$  中安置  $M$  的成员，使它们成为  $n$  的后继。
- 7)从  $M$  的每一个不在  $G$  中的成员建立一个指向  $n$  的指针(例如，即不在 OPEN 中，也不在 CLOSED 中)。把  $M$  的这些成员加到 OPEN 中。对  $M$  的每一个已在 OPEN 中或 CLOSED 中的成员  $m$ ，如果到目前为止找到的到达  $m$  的最好路径通过  $n$ ，就把它的指针指向  $n$ (本句是就节点  $n$  与  $m$  的关系来说的)。已在 CLOSED 中的  $M$  的每一个成员，重定向它在  $G$  中的每一个后继，以使它们顺着到目前为止发现的最好路径指向它们的祖先(本句是就  $m$  与  $G$  中除  $n$  以外的其他节点的关系来说的)。
- 8)按递增  $\tilde{f}$  值，重排 OPEN(相同最小  $\tilde{f}$  值可根据搜索树中的最深节点来解决)。
- 9)返回第 3 步。

#### 4.3 算法分析

设  $\tilde{f}(n)$  是搜索模型的一个实数值函数，将其表示为  $\tilde{f}(n) = \tilde{g}(n) + \tilde{h}(n)$ ，其中

$\tilde{f}(n)$  - 起始节点到目标节点间通过节点  $n$  的最优路径代价的实际值

$\tilde{g}(n)$  - 起始节点到节点  $n$  的最优路径代价的实际值

$\tilde{h}(n)$  - 节点  $n$  到目标节点的最优路径代价的实际值

$\tilde{f}(n)$  - 起始节点到目标节点的最优路径的估计值，是

$f(n)$  的某个估计  
 $\tilde{g}(n)$ -深度因子, 启发式搜索算法发现的从起始节点到节点  $n$  的最优路径代价, 是  $g(n)$  的某个估计  
 $\tilde{h}(n)$ -启发因子, 是  $h(n)$  的某个估计  
 两个重要条件:

**可接纳性条件:** 如果对  $\forall n, \tilde{h}(n) \leq h(n)$ , 则称  $\tilde{h}$  是可接纳的。可接纳性条件是对启发函数的一个约束, 满足这种约束条件的  $\tilde{h}$  再对图施加一定的条件可以保证算法的完备性。

**一致性条件(单调性条件):** 考虑节点对  $(n_i, n_j)$ ,  $n_j$  是  $n_i$  的一个后继。如果搜索图中所有的这种节点对满足一下条件:

$\tilde{h}(n_i) - \tilde{h}(n_j) \leq c(n_i, n_j)$ , 其中  $c(n_i, n_j)$  是从  $n_i$  到  $n_j$  的代价。

上式也写作  $\tilde{h}(n_i) \leq \tilde{h}(n_j) + c(n_i, n_j)$  和  $\tilde{h}(n_j) \geq \tilde{h}(n_i) - c(n_i, n_j)$

我们就称  $\tilde{h}$  服从一致性条件。一致性条件说明了沿图中任何路径到达目标节点的剩余代价估计值的减少不会大于该路径弧的代价, 即在考虑一个弧的已知代价后, 启发函数是局部一致的。一致性条件可表示成下图的三角不等式:

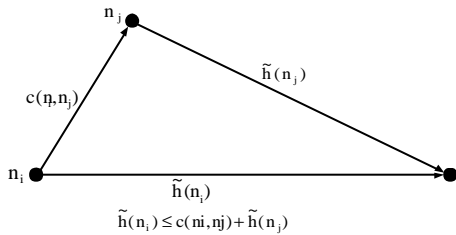


图1 一致性条件

一致性条件同时也表明当搜索远离开始节点时, 节点的最优路径估计值是单调非递减的。简要证明如下: 若满足一致性条件, 由  $\tilde{h}(n_j) \geq \tilde{h}(n_i) - c(n_i, n_j)$  可得  $\tilde{h}(n_j) + \tilde{g}(n_j) \geq \tilde{h}(n_i) + \tilde{g}(n_j) - c(n_i, n_j)$ , 即  $\tilde{f}(n_j) \geq \tilde{f}(n_i)$ 。因此, 一致性条件(对  $\tilde{h}$ ) 也称之为单调性条件(对  $\tilde{f}$ )。

Nilsson<sup>[3]</sup> 等人通过研究发现, 对搜索图和  $\tilde{h}$  施加一些条件, 可以保证应用到搜索图的 A\* 算法总能找到最小代价路径。其中, 搜索图的条件是:

- 1) 搜索图中每个节点的后继是有限的(如果有的话)。
- 2) 搜索图中所有弧的代价都大于某个正数。

$\tilde{h}$  的条件是:

对搜索图中所有节点,  $\tilde{h}$  是可接纳的。并提出以下两条定理(详细证明见文献[4]):

定理 1: 如果搜索图和  $\tilde{h}$  具有上述稳定条件, 而且从开始节点  $n_0$  到目标节点有一条有限代价路径, 那么算法 A\* 保证终止于到达目标的一条最小代价路径。

定理 2: 如果  $\tilde{f}$  上的一致性条件被满足, 那么当 A\* 扩展节点  $n$  时, 它已经找到了到达节点  $n$  的一条最优路径。

其中, 定理说明了满足上述条件的 A\* 算法既是完备的, 又是最优的。一致性条件(定理 2) 则保证了对于任何重复节点的搜索总是在第一次扩展时就找到了到达该点的最优路径。从而使得算法的第七步变得简单, 节点不再需要重定向。

## 5 A\* 算法在矢量地图中的应用

### 5.1 矢量地图简介

矢量地图是电子地图的一种, 是多数 GIS 应用(如定位、导航、监控等)的基础。相比栅格地图, 矢量地图具有数据量小、可随意缩放且不会失真、方便进行全局或者局部校正以及便于更新等优点。

矢量地图可看作由点、线、面这三种几何对象以及它们的属性数据构成的集合。这里引入地图矢量库和地图数据库两个概念。矢量库是一组图形描述数据, 保存了地图的几何数据; 数据库是一组描述数据, 保存地图各种几何对象的属性数据, 二者之间可通过对象的序号建立起索引关系。如下图。

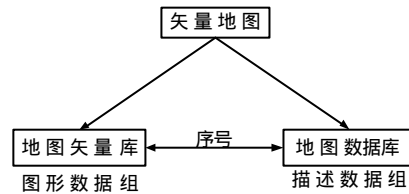


图2 矢量地图的组成

最优路径算法主要涉及到地图矢量库的相关结构, 一幅简单的矢量地图如下图所示。

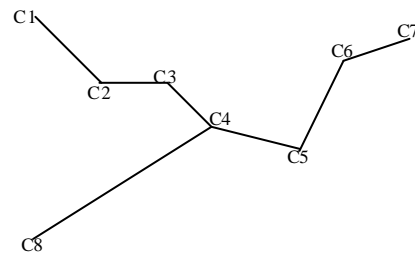


图3 简单矢量地图示意图

对矢量图中有关结构定义如下:

定义 1 矢量边。是一些点的坐标的集合, 表征着一条连续的折线。如  $c1c2c3c4$

定义 2 大节点、小节点、节点。大节点是矢量边的端点, 如  $c1$ 、 $c4$ 、 $c7$ 、 $c8$ ; 小节点是指矢量边除端点外的内部点, 如  $c2$ 、 $c3$ 、 $c5$ 、 $c6$ ; 大小节点统称

为节点。

定义 3 弧。一条弧就是一条矢量边，是若干个节点的集合。

定义 4 路。路是若干条弧的集合。如 c8c4c5c6c7。

显然，与算法关联最大的结构正是节点与弧。在一幅矢量图里，所有的节点与弧都分别被赋予一个唯一的 ID 号。通过这个 ID 号，使得由节点索引弧和由弧索引节点变得简单。与算法相关的矢量地图的数据结构定义如下：(这些数据结构对于正确显示地图拓扑是重要的)

节点：

```
struct Point{           // 常规点结构
    int x;
    int y;
};
typedef struct NetPoint{ // 节点结构,生成地图拓扑
    int Id;              // 节点的 ID 号
    struct Point Pt;    // 节点的几何坐标,包括 x,y
    int ArcNum;         // 包含该节点的弧的条数
    int *pArcHead;     // 包含了该节点的弧的序号
}NetPoint;             // 组成的链表的头指针
```

弧：

```
typedef struct Link{
    int Id;              // 弧的 ID 号
    double Len;         // 弧长
    int NPointNum;     // 弧包含的节点的数目
    int *pNPointHead;  // 构成该弧的序号组成
}Link;                 // 的链表的头指针
```

### 5.2 A\*算法涉及的数据结构

从图 2 我们可以看到，对于小节点，我们不需要计算它的 f 值，因为此时只存在唯一一条路径。也就是说，A\*搜索应集中在大节点上进行。为此定义了另一种节点结构，该结构存储了搜索过程中节点的启发信息以及最优路径上的父节点。定义两种数据点的数据结构的好处在于：1)将矢量图的显示与最优路径算法分离，矢量图自身的数据结构不需变化，原有在该结构上的其他应用也不需作变化。从而便于 A\*算法的移植，可以花较少代价应用到具有其他数据结构的矢量图中。2)分散了计算量，有利于提高算法的执行效率。

```
typedef struct Node { //大节点结构,用于实现算法
    int Id;           // 节点的 ID 号
    CRect Pt;        // 为使节点清楚显示,并捕捉
                    //用户的输入,并非必需
```

```
int ParentID; //存储最优路径上父节点的 ID
                //初始化为 0 表示不存在父子关系
double g_value; // 节点的 g 值,初始化为 0
double h_value; // 节点的 h 值,初始化为 0
double f_value; // 节点的 f 值,初始化为 0
```

}Node;

OPEN 表和 CLOSED 表的实现：

OPEN 表这里采用排序链表实现，从而节省了存储空间。

```
typedef struct NodeID{ //用于表示 OPEN 表
```

```
int Id;
struct NodeID *pNext;
```

}NodeID;

CLOSED 表这里采用动态数组实现，最大程度地节省了存储空间

### 5.3 应用于矢量地图的 A\*算法完备性、最优性分析

在矢量地图中，我们采用两点间的欧式距离来表示节点的启发值。节点间的估计距离必然小于等于实际距离，从而 A\*算法是可接纳的。而矢量图显然也满足前述对图的要求。根据定理 1，A\*算法是最优的，也必然是完备的。

由于是静态权值，算法显然也满足一致性条件，从而定理 2 也成立。故前面提出的算法第 7 步得以简化，节点不再需要重定向指针。

## 6 实验及结果分析

如图 4 所示显示了 2 条最优路径(用粗线表示的始点 1 到终点 1 的最优路径和始点 2 到终点 2 的最优路径)。在各类求解最优路径的算法中，标号算法是一类较常用的算法，而 D 算法又是各类标号算法中效率较高的一种。D 算法的时间复杂度取决于寻找标记节点的方式。设搜索图的节点数为 n，边数为 m，弧长非负且其绝对值小于常数 C。其中效率较高的几类分别是基于 two-level R-heaps 和 Fibonacci heaps 混合优先级队列实现的 D 算法，其时间复杂度为  $O(m+n\sqrt{\log C})^{[6]}$ 。基于 Bucket 优先级队列实现的 D 算法，其时间复杂度为  $O(m+nC)^{[6]}$ 。基于 Double Buckets 优先级队列实现的 D 算法，其时间复杂度为  $O(m+n(\Delta+C/\Delta))$ ，其中  $\Delta = \Theta(\sqrt{C})^{[6]}$ 。相比之下 A\*算法的时间复杂度和空间复杂度仅为  $O(n')$ (其中  $n'$  是 A\*算法扩展的节点数)，具有很明显的优势。A\*算法的缺点在于它本质上是上一种特殊的 BFS 算法，具有 BFS 算法的一般缺点，尤其是当应用到求解具有海量节点的网络中最优路径时，其指数级扩展节点的缺点表现的更为明显。但是应用于城市 GIS-T 系统的 A\*算法已能胜任快速性、准确性等实践要求。

在 VC++ 6.0 环境下实验 A\* 算法，其效果图如下：

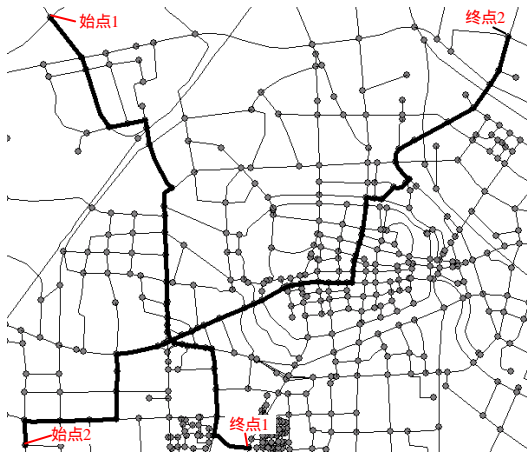


图4 合肥城区应用 A\*算法求解最优路径效果图

#### 参考文献

- [1]陆锋.城市数字交通工程研究.数字地球研究,崔伟宏主编.北京:中国环境科学出版社,1999.
- [2]Stuart J Russell, Peter Norving. Artificial Intelligence: a modern approach(影印版).北京:清华大学出版社,2006.
- [3]Nils J. Nilsson.人工智能.北京:机械工业出版社,2000.
- [4]孙世博,冯勇,郑剑飞.车辆导航系统最优径规划研究.计算机应用,2006,25(9):p44~p46.
- [5]赵伟华,章复嘉,梁红兵.车辆导航系统最优路径规划的研究与实现.杭州电子工业学院学报,2003,23(1):p16~p19.
- [6]Cherkassky B V, Goldberg A V, and Radzik T. Shortest paths algorithms: Theory and Experimental Evaluation [R]. Technical Report 9321480, Computer Science Department, Stanford University. 1993.