A * 算法及其在地理信息系统中的应用

A * Algorithm and Its Application to GIS

熊 伟 （解放军后勤工程学院研究生4队 重庆 400016）
张仁平 刘奇韬 王贵新 （解放军后勤工程学院后勤教研室 重庆 400016）

摘要：最短路径问题是地理信息系统的关键问题，A * 算法是解决有附加条件的最短路问题的有效算法。本文在详细分析 A * 算法的基础上，编程实现了 A * 算法。最后对 A * 算法运行结果进行分析。

关键词：A * 算法 地理信息系统 最短路径 算法

1 引言

许多路径探索的优化算法，典型的有 Dijkstra 算法、Bellman – Ford – Moore 算法、A * （也可读作 A 星）算法等等，都能对当前节点进行评估。三种算法比较中，Dijkstra 算法是大家最熟悉的最短路径算法，是目前公认的求解最短路问题的最经典算法，主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。Dijkstra 算法虽然能得出最短路径的最优解，但由于它遍历计算的节点很多，通常计算量比较大，所以效率低，比较适合小范围内的最短路径计算。另外，该算法不能处理权值为负数的最短路问题；Bellman – Ford – Moore 算法相对比较简单，它较好地解决了存在负权的路径计算问题，但是和 Dijkstra 算法一样，其计算量还是比较大；A * 算法在 Dijkstra 算法的基础上，通过增加当前节点有效评估，即增加约束条件，有效解决了计算量大的问题。

2 A * 算法介绍

什么是 A * 算法？如何实现 A * 算法？怎样应用 A * 算法？下面结合笔者在实际工程应用过程中的经验体会，总结如下。

2.1 算法原理

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，算法的创新之处在于选择下一个被探索的结点时引入了已知的路网信息，特别是目标点信息，对当前结点距终点的距离进行评估，作为评价该结点属于最优路节点可能性的评价指标，这样就可以优先搜索可能性较大的结点，从而提高了搜索过程的效率。

从算法原理我们可以看出，A * 算法在 Dijkstra 算法基础之上引入了当前结点的估计函数，则当前结点的估计函数可定义为：

$$f^*(i) = g(i) + h^*(i) \quad ②$$

式中 $g(i)$ 是从起点到当前结点的实际最短距离，可通过 Dijkstra 算法获得，$h^*(i)$ 是从当前结点 i 到终点的最短距离的估计值，可取结点 i 到终点的直线距离。

细心的朋友可能有两个发现：一是②式与①式（已删略）很“相似”，其实②式只是①式的特殊形式，即只对距离进行评估，因为对当前节点的评估方法有许多，如方向、距离或其它指标，甚至各种指标综合应用，但是评估的总原则是评估值与实际值越接近，则评估函数就越好；二是若 $h^*(i) = 0$，即没有利用任何路网信息，这时 A * 算法就变成了普通的 Dijkstra 算法，这说明普通的 Dijkstra 算法可看作 A * 算法的特例，即 A * 算法是 Dijkstra 算法的“改进算法”。这对于我们大多数熟悉 Dijkstra 算法的人来说，实现 A * 算法有了“捷径”可走。

对于 $h^*(i)$ 的具体形式，算法使用者可以依据实际情况选择，正如前面所述，评估的方法有许多，$h^*(i)$ 只需满足一个约束条件：即不能大于结点到终点的实际最短距离，这一条件就是所谓的相容性条件。可以证明，如果估计函数满足相容性条件，则原问题一定存在最优解，即 A * 算法能够求出的“最短路径”是真

正的最短路径；反之，若不满足相容条件，则原问题可能不存在最优解，因为随着探索节点增加，距离可能越来越远，而失去“方向”。所以说，$h*(i)$的取值比较关键，通常可取结点到终点的球面距离或直线距离。

下面给出“A＊算法求解出的‘最短路径’是真正的最短路径”相关证明。

2.2 算法最短路径证明

直接证明“A＊算法求解出的‘最短路径’是真正的最短路径”有一定难度，我们需要转换一下角度，将原问题转换为 A＊算法求解出“最短路径”不会因探索结点次序的不同而改变，即每次搜索到的结点所算出的“最短路径”是相同的，没有比它更短，则该路径是起点到该结点的最短路径。依次类推，当搜索到终点时，得到的“最短路径”就是起点到终点的最短路径，即通过 A＊算法所求出的“最短路径”就是真正的最短路径。

如下图：P_s 表示起点，P_e 表示终点，P_i 和 P_j 表示路网中的两个结点，R_i、R_j 分别表示 P_s 到 P_i 和 P_j 的最短路径，L_i、L_j 分别表示 P_i、P_j 到 P_e 的（球面）距离，A 表示 P_i 到 P_j 的（球面）距离。假设 $R_i < R_j$，$R_j + L_j > R_i + L_i$。很显然，对于 Dijkstra 算法是先搜索到 P_i（因为 $R_i < R_j$），而对于 A＊算法是先搜索 P_i（因为 $R_i + L_i > R_j + L_j$）。无论是 Dijkstra 算法是先搜索到 P_i，还是 A＊算法是先搜索 P_i，P_s 到 P_e 的最短路径仍然是 P_s—P_i—P_e，不会是 P_s—P_j—P_e 或者 P_s—P_j—P_i—P_e

由已经条件 $R_j + L_j > R_i + L_i$ 可知 P_s—P_j—P_e 不是最短路径，证明 P_s—P_j—P_i—P_e 不是最短路径，我们只需证明 $R_j + A > R_i$ 即可，

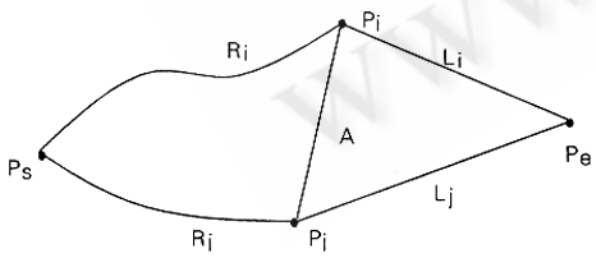


图 1

证明：因为 $R_j + L_j > R_i + L_i$，（已知）

所以 $R_j + L_j - L_i > R_i$，

因为 $A \geq L_j - L_i$，（三角形 $P_i P_j P_e$）

所以 $R_j + A > R_i$

2.3 A＊算法步骤

为了便于说明 A＊算法，引入两个数组 S 和 P，P 表示已经被探索到但尚未进入最短路径的结点集合，相当于 Dijkstra 算法中没有被标识的结点的集合，为已经进入最短路径的结点集合，相当于 Dijkstra 算法中被标识的结点的集合。算法执行完毕之后，S 即为已找到从 v_s 出发的最短路径的结点集合。

结合 Dijkstra 算法和公式②，A＊算法步骤可描述如下：式中的 $f*$ 值是当前结点的估算值（见公式②），是当前结点到起始点的最短距离 g_v 与到目标点的直线距离 $h*_w$ 之和；式中的 g_v 为当前结点 V 到起始点的最短距离；p_w 表示当前结点 W 的父结点。

（1）设置初值。对起始结点 v_s，令 $g_{v_s}=0$；对其余结点 $\forall v \neq V_s$，令 $g_v=\infty$，令 $S=\{v_s\}$，$P=\Phi$，显然 v_s 是最短路径的第一个结点。

（2）将当前结点的临近结点放入 P 中，若 $P=\Phi$，即没有路可走，则算法失败；否则，从 P 中选出具有最小 $f*$ 值的结点 v，即令 $v=\min_{u\in P}\{f*_u\}$。令 $S=S\cup\{v\}$，$P=P-\{v\}$。

判断 v 是否为目标结点。若是目标结点，转（4）；否则，获得 v 的所有邻近结点。继续执行（3）。

（3）对于每一个邻近结点 w，计算 $g_v+D(v,w)$，根据 w 所处的位置，有三种情况：

① 若 $w\in P$，判断是否有 $g_w > g_v + D(v,w)$，若是，则 $g_w = g_v + D(v,w)$，$p_w = v$；

② 若 $w\in S$，判断是否有 $g_w > g_v + D(v,w)$，若是，则 $g_w = g_v + D(v,w)$，$p_w = v$；

③ 若 $w\notin P$ 且 $w\notin S$，令 $g_w = g_v + D(v,w)$，$p_w = v$，$P = P\cup\{w\}$，计算结点 w 的估计函数 $f*_w = g_w + h*_w$，转（2）；

（4）从目标点 V_e 开始，利用回溯的方法找到根据其父节点 p_{v_e}，依次类推，最终找到起始点 v_s，从而得到从起始点到目标点的最短路径，以及最短距离 g_v，算法终止。

为了便于 A＊算法理解和编程实现，下面举个非常具有典型的例子加以说明。

例：在图1中，用 A＊算法求从 U_0 到所有节点的最短距离及路径，其中假定 U_i（$i=0\cdots6$）到终点 U_7 的距离估值 $h*(i)$ 分别为 7、6、5、4、3、2、1。

第一步，将 U_0 的邻近节点{U_1、U_2、U_3}放入数组 P 中，则 P={U_1、U_2、U_3}，找出一个估算值最小的节点 U_2，放入数组 S 中，其估算值为：1+6=7。此时 S={U_0、U_2}，P={U_1、U_3}，pU2=U_0

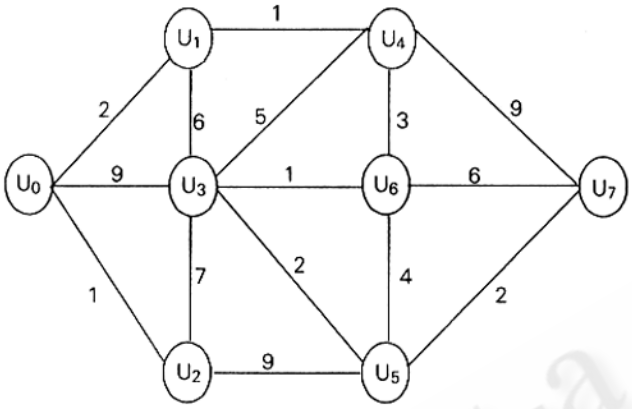


图 2

第二步，U_2 的邻近节点{U_3、U_5}，对于 $U_5 \notin P$ 且 $\notin S$，计算 $g_{U5}=10$，对于已经在 P 里的 U_3，由于 $G_{U3} > g_{U2}+D(U2,U3)$，（因为 $g_{U3}=9$，$g_{U2}=1$，$D(U2,U3)=7$），则 g_{U3} 需要修改为 1+7=8；将{U_3、U_5}放入数组 P 中，则 P={U_1、U_3、U_5}，找出最小估算值的节点 U_1，放入数组 S 中，其估算值为：2+6=8。此时 S={U_0、U_2、U_1}，P={U_3、U_5}，pU1=U_0

第三步，U_1 的邻近节点{U_4、U_3}，对于 U_4 需要计算 g_{U4}，对于 U3，无须改变；将{U_4、U_3}放入数组 P 中，则 P={U_3、U_4、U_5}从中找出最小估值的节点 U_4，其估值为：3+3=6。此时 S={U_0、U_2、U_1、U_4}，P={U_3、U_5}，pU4=U_1

第四步，U_4 的邻近节点{U_1、U_3、U_6、U_7}，对于已存在于 P 中的{U_3}和 S 中的{U_1}，不满足算法描述中的修改条件，而无须修改，将 U_6、U_7 放入数组 P 中，则 P={U_3、U_5、U_6、U_7}，从中找出最小估算值的节点 U_6，放入数组 S 中，其估算值为：6+1=7。此时 S={U_0、U_2、U_1、U_4、U_6}，P={U_3、U_5、U_7}}，pU6=U_4

第五步，U_6 的邻近节点{U_3、U_5、U_7}，对于存在于 P 中的结点{U_3、U_5}，其相应的 g_{U3} 和 g_{U5} 分别修改为 2+1+3+1=7 和 2+1+3+4=10，从 P 中找出最小估算值的节点 U_3，放入数组 S 中，其估算值为：7+4=11。此时 S={U_0、U_2、U_1、U_4、U_6、U_3}，P={U_5、U_7}}，pU3=U_6

第六步，U_3 的邻近节点{U_1、U_2、U_4、U_5、U_6}，对于属于 S 中的结点{U_1、U_2、U_4、U_6}，都不存在 $g_w > gU3+D(U3,w)$（W={1、2、4、6}），即不需要从 S 中清除，然后加入 P 中。对于已经位于 P 中的 U_5，需要将 g_{U5} 的值由 10 改为 9。从 P 中找出最小估算值的节点 U_5，放入数组 S 中，其估算值为：9+2=11。此时 S={U_0、U_2、U_1、U_4、U_6、U_3、U_5}，P={U_7}，pU5=U_3

第七步，U_5 的邻近节点{U_2、U_3、U_6、U_7}，对于属于 S 中的结点{U_2、U_3、U_6}，都不存在 $g_w > gU5+D(U5,w)$（W={2、3、6}），不需要从 S 中清除，然后加入 P 中。由于 P 中只有一个元素 U_7，显然最小估算值的节点 U_1，放入数组 S 中，其估算值为：11+0=11。此时 S={U_0、U_2、U_1、U_4、U_6、U_3、U_7}，P=Φ，pU7=U_5

因为 U_7 是目标点，则用溯的方法找到其父节点 U_5，依次类推，最终找到起始点 U_0，从而得到从起始点到目标点的最短路径{U_0、U_1、U_4、U_6、U_3、U_5、U_7}

2.4 算法的评价

A* 算法和 Dijkstra 算法的比较

在最短路径证明的例子中，Dijkstra 算法是先探索 P_1，但是也必须搜索 P_1，因为 $R_i+L_i<R_i+L_i$，可得 $R_i<R_i+L_i$，所以必须探索 P_i。而 A* 算法则可能不探索 P_i，因此 A* 算法的探索结点个数小于 Dijkstra 算法。

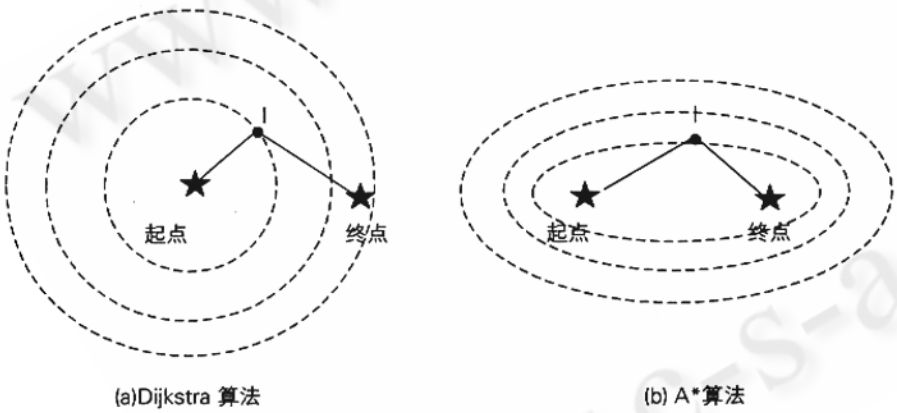
另外，由于 Dijkstra 算法在计算的过程中没有考虑终点的方向和位置，所以在从起点出发搜索过程中，只是对当前点到起点的距离进行评价，因此在不考虑距离时，所有结点被搜索到的概率是相同的。而 A* 算法在选择下一个结点时，对该点距终点的距离作出估计，作为评价该结点处于最短路径上的可能性程度。这样就可以优先搜索可能性较大的结点。其搜索过程可近似为以源点和终点为焦点的一系列同心椭圆。其搜索过程比较示意图如图 3 所示。

3 A* 算法编程实现

在弄清 A* 算法的基本原理和算法的方法步骤后，特别是明白 A* 算法是 Dijkstra 算法的“改进算法”之后，编程实现就显得比较容易。为了进一步说明，我们有必要简要回顾前面内容。我们知道 A* 算法的核心是估价函数 f(n)，它包括 g(n) 和 h(n) 两部分。g(n) 是已经获得的实际值，h(n) 是 n 到目标点的估计值，也是 A* 算法是 Dijkstra 算法之区别的关键所在。

因此，在实际工程应用中，我们仅需要对 Dijkstra 算法进行改进：在确定下一个探索结点时，增加当前结点到目标点距离的判断，即 Dijkstra 算法只考虑起始点到当前结点的距离，而 A＊算法则既要考虑起始点到当前结点的距离，又要考虑当前结点到目标点距离，其它部分基本不需要修改。并且对于大多数 GIS 开发平台，都提供这样的函数，获得前结点到目标点距离。因此 A＊算法的基本没有增加编程工作量，但是效率确大大提高。

另外，有时需要考虑时间最优，费用最优，最安全或者某些道路必须通行或者不能通行等有附加条件的最优路，而不仅仅是距离最短。因此，在公路网格中，需要对道路赋于不同的权重，之后参与计算，这些在 A＊算法中，都能充分考虑，轻松解决。笔者在最近开发的几个地理信息系统中，A＊算法都得到成功应用，解决了部队的实际问题。系统的 GIS 软件开发平台是优秀的国产软件 MapEngine，。系统的前台开发工具是非常优秀的面向对象开发工具 Delphi7.0 C/S 版。



(a)Dijkstra 算法　　(b) A*算法

图 3　两种算法的比较

4　A＊算法运行结果与分析

笔者用在 MapEngine2.5 版和 Delphi7.0 C/S 实现了 A＊算法，并在部队机动最短路优化分析功能中得到应用。确定始发点和目标点有两种方式：一是在地图中选择；二是输入相应的经纬度。在确定部队机动的始发点和目标点后，系统能在地图上寻找到一条最短路，并给出最短路长度等信息。如果最短路中有某条公路不能通行时，注上标记，系统继续寻找去除该条路之后的另一条最短路。

下面，列出部分 A＊算法和 Dijkstra 算法的探索结点个数比较。（算法只考虑公路）

起始点、目标点	Dijkstra 算法	A＊算法	提高效率
重庆、北京	12419	6857	45%
重庆、成都	1505	640	57%
重庆、沈阳	12616	8319	34%
北京、广州	16723	10142	39%
乌鲁木齐、上海	19029	11523	39%
拉萨、兰州	4478	1904	57%
兰州、北京	10149	4179	59%
重庆、兰州	7429	4280	42%
兰州、广州	19395	10790	44%

从表中可以看出，A＊算法比 Dijkstra 算法的探索结点个数要少得多，探索效率提高从 34% 到 57%。如果考虑起点、终点选择上的可能存在微小误差，再考虑不同道路结点个数及其位置不同，A＊算法的效率比 Dijkstra 算法提高 40% 左右。

另外，有时由于电子地图拓扑关系的原因，计算起点与计算终点不能通行，容易出现死循环，可以在获得新的循环起点之后加上判断，看看循环次数是否大于所有满足条件的公路条数，如果大于，则退出循环，给出“不能通行”等相应提示信息。

参考文献

1　龚劬，图论与网络最优化算法，重庆大学出版社，1998.

2　Steve Teixeira & Xavier Pacheco，Delphi5 开发大全，人民邮电出版社，1999 年 8 月.

3　Steve Teixeira & Xavier Pacheco，Delphi5 开发人员指南，机械工业出版社，2001 年 10 月.