第27卷 第13期 计算机工程与设计 2006年7月
Vol. 27 No. 13 Computer Engineering and Design July 2006

游戏开发中智能路径搜索算法的研究

何国辉， 陈家琪
(上海理工大学 计算机工程学院，上海 200093)

摘 要：路径搜索是许多游戏特别是即时战略游戏的核心组成部分，首先介绍了游戏中路径搜索的相关概念。路径搜索的算法有很多，不同的搜索算法有其不同的搜索策略、时间效率、空间消耗与应用场合。分析对比了多种路径搜索算法的运行数据之后，详细讨论了A*算法。由于游戏中的路径搜索有其自身的特点，针对游戏中路径搜索的具体要求从搜索效率、路径的真实平滑性和动态变化状态空间的适应性等方面对A*算法进行了优化和改进。

关键词：路径搜索； 游戏开发； A*算法； 人工智能； 动态环境
中图法分类号：TP18 **文献标识码**：A **文章编号**：1000-7024 (2006) 13-2334-04

Research on algorithm of intelligent path finding in game development

HE Guo-hui, CHEN Jia-qi
(College of Computer Engineering, Shanghai University of Science and Technology, Shanghai 200093, China)

Abstract: The pathfinding technique is one of the core techniques in many games, especially in real-time strategy games (RTS). Firstly, some concepts of pathfinding in game development are introduced. There are many pathfinding algorithms, which have different search strategies, time efficiency and space efficiency. So they have different application situations. Then, some pathfinding algorithms are analyzed and contrasted. The A* algorithm is discussed particularly. For the special requirements of pathfinding in game, the A* algorithm is optimized and improved in the efficiency of search, the smooth ability of the path and the adaptability of dynamic state space.

Key words: path finding; game development; A* algorithm; artificial intelligence; dynamic environment

0 引 言

游戏开发涉及许多方面的内容，其中路径搜索是许多游戏中必不可少的部分。在计算机及相关学科的发展过程中也产生了许多路径搜索算法。然而游戏中的路径搜索有其自身的特点，它要求搜索时间短，这是很重要的，如果指挥100个作战单位前往目的地，每个作战单位光搜索路径就耗费掉1秒钟将是无法忍受的；还有路径的真实平滑性，即使路径长度一样，可是一条平滑的路线显然更真实；由于游戏中各元素单位可能会动态地影响状态空间，所以在游戏中路径搜索必须要求能够适应状态空间的动态改变性。文章探讨的就是在游戏中如何更快更好地找到一条从起点到终点的路径。

1 主要路径搜索算法对比分析

对于一个实际的问题，我们可以把它进行一定的抽象。通俗地说，状态(state)是对问题在某一时刻进展情况的数学描述，状态转移(state-transition)就是问题从一种状态转移到另一种(或几种)状态的操作。搜索的过程实际是在遍历一个隐式图，它的结点是所有的状态，有向边对应于状态转移。一个可行解就是一条从起始结点出发到目标结点的路径。这个图称为状态空间(state space)，这样的搜索就是状态空间搜索，游戏中的路径搜索就是状态空间的搜索。可分为盲目搜索 (uninformed search)和启发式搜索(heuristically search)。

盲目搜索就是按照一定的规则下去，不考虑结点带给的任何信息。盲目搜索主要包括纯随机搜索 (random generation and random walk)、广度优先搜索(BFS)、深度优先搜索(DFS)及由它们产生的一些其它衍生算法[1]；启发式搜索就是在状态空间搜索时对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提到了效率。启发式搜索也有很多种算法，比如局部择优搜索法、最好优先搜索法等。这些算法都使用了启发函数，但在具体选取最佳搜索节点时的策略不同。A*算法在人工智能中是一种典型的启发式搜索算法，它属于最好优先搜索法的一种[2,3]。

各种搜索算法都可以找到起始结点至目的结点的路径(如果有一条可行路径的话)，但性能却相差很大。通过图1所示的搜索算法演示程序(背景是一个2 000*2 000像素的地图，用一个200*200的数组控制每个结点是否可以通过)。得到的数据如表1所示(所有数据均为当时环境下的实测数据)。由表1可以看出，标准A*算法搜索的结点数最少，可是在时间性能

收稿日期：2005-05-18。
基金项目：上海市教育委员会科研基金项目 (04EB12)。
作者简介：何国辉 (1981—)，男，江西余干人，硕士研究生，研究方向为人工智能、游戏开发； 陈家琪 (1957—)，男，教授。

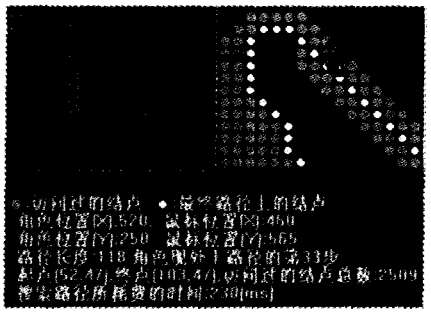

图 1 搜索算法演示结果

上却不尽人意。如果能减少每个结点的访问时间，那么A*算法将是性能很好的路径搜索方法。另外针对游戏本身的特点还需要对标准A*算法做一些优化与改进。

表 1 各搜索算法运行数据

搜索算法	起点	终点	搜索结点数	路径长度	消耗时间/ms
广度优先	(52,47)	(103,47)	21 979	118	79
	(81,27)	(51,44)	9 818	66	31
深度优先	(52,47)	(103,47)	13 827	4 576	47
	(81,27)	(51,44)	24 827	8 069	62
标准 A*	(52,47)	(103,47)	2 509	118	230
	(81,27)	(51,44)	908	66	31

2 A*算法的优化与改进

2.1 A*算法简介

因为启发式搜索是在状态空间的搜索时对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到找到目标结点或是搜索失败。所以在启发式搜索中，对位置的估价是十分重要的。采用了不同的估价可以有不同的效果。估价是用估价函数表示的，如：$f^*(n)= g^*(n)+h^*(n)$，其中 $f^*(n)$ 是节点n的估价函数，$g^*(n)$是指状态空间中从初始节点到节点n的实际代价，$h^*(n)$是从节点n到目标节点最佳路径的代价。在这里主要是 $h^*(n)$体现了搜索的启发信息，因为 $g(n)$是已知的。由于在一些问题求解时，希望能够求解出状态空间搜索的最短路径，如果一个估价函数可以找出最短的路径，称之为可采纳性。A*算法是一个可采纳算法。A*算法的估价函数可表示为：$f(n)=g(n)+h(n)$，这里，$f(n)$是估价函数，$g(n)$是起始结点到结点 n 目前所知的最短路径值，$h(n)$是结点n到目标的估价启发值。由于 $f^*(n)$是无法预先知道的，所以用估价函数 $f(n)$做近似。$g^*(n)$用 $g(n)$做近似，但保证 $g(n)>=g^*(n)$，$h^*(n)$ 用 $h(n)$做近似，但保证 $h(n)<=h^*(n)$。应用这样的估价函数是可以找到最短路径的（如果有可达路径的话），也就是可采纳的。A*算法要用到两个表，OPEN 表和 CLOSED 表，分别存放未扩展的结点和已扩展的结点。A*算法的搜索过程可以表述如下：

(1)把起点 S 放入 OPEN 表，记该结点的 f=h，令 CLOSED 为空表。

(2)重复下列过程，直至找到目标节点为止，若OPEN为空表，则宣告失败。

(3)选取OPEN表中未设置过的具有最小f值的节点为最佳节点 BESTNODE，并把它放入 CLOSED 表。

(4)若 BESTNODE 为目标节点，则求得问题的解。

(5)若 BESTNODE 不是目标节点，则扩展之，产生后继节点 SUCCSSOR。

(6)对每个SUCCSSOR进行下列过程：①建立从SUCCSSOR返回 BESTNODE 的指针；②计算 g (SUC)=g (BES)+g (BES, SUC)；③如果SUCCSSOR∈OPEN，则称该节点为OLD；④比较新旧路径代价。如果 g(SUC)<g(OLD)，则重新确定 OLD 的父辈节点为 BESTNODE，修改 g(OLD)=g(SUC)，并修正 f(OLD)值；⑤若 OLD 节点的代价较低或是一样，则停止扩展该节点；⑥若 SUCCSSOR 不在 OPEN 表中，则看其是否在 CLOSED 表中；⑦若 SUCCSSOR 在 CLOSED 表中，则转向③；⑧若 SUCCSSOR 既不在 OPEN 表中，又不在 CLOSED 表中，则把它放入 OPEN 表中，然后转向(7)。

(7)计算 f 值。

(8)GO LOOP，转向(2)。

2.2 通过改变数据结构减少搜索时间

标准 A*算法用于游戏中有几处非常影响速度的地方，程序在第(5)步每次扩展结点时都要进行繁琐的内存分配，这是项耗费时间的工作，其次在第(6-③)和第(6-⑥)步需要检查是否在 OPEN 和 CLOSED 两个表中，这点也是很慢的，另外要用一个数组来保存每个结点的历史最短距离（初值为无穷大），在第(6-②)及(6-④)步中会修改g，为了进行多次搜索必须对保存动态规划数据(历史最短距离)的数组进行还原，重置无穷大（赋值一个足够大的整数保证大于每个 g(n) 就行），每次寻路前或是寻路后都要先还原这个数组，若数组比较大将是无法接受的，用的时间可能比整个搜索时间还长。

为了消除繁琐的内存分配我们可以把那个保存每个结点的历史最短距离的数组改为如下结构体的一个等大小数组

```
typedef struct NodeType
{    int x, y;      float f;
     int g;  float h;
     NodeType *father;
     NodeType *next;
}NodeType;
NodeType Nodes[WIDTH][HEIGHT];
```

father 指向它的父节点，Next 指向下一结点，WIDTH 与 HEIGHT 指的是与地图上面每个节点一一对应的节点数组的维数。由于不可能出现一个节点在 OPEN/CLOSED 两个表同时出现的情况，因此把一个结点(x,y)插入到OPEN或CLOSED表时也就是把节点地址&Nodes[x][y]插入到这个两个表的合适位置，Nodes[x][y]的Next指向表的下一个节点，如果没有节点则该点的 Next 为 NULL。这样每次搜寻的时候就不必内存分配了，插入节点 (x, y) 时只要改动 Nodes [x] [y] 中的数据和 OPEN/CLOSED 表中的指针。

为了消除在第(6-③)和第(6-⑥)步的两次对OPEN表和CLOSED表的查询，可以在上面的数据结构中再加入两个 BOOL 型的变量 BOOL InOpen;BOOL InClosed，初值都设定为 false，当把结点(x,y)加入到 OPEN 表和 CLOSED 表中就把相应的变量赋值为 true；这样只要访问这两个变量的值就可以知道一个结点是否在 Open 表或是 Closed 表中了，而不必去查询这两个表。

通常情况下，被访问的结点和整个结点数相比起来只是很少的一部分。为了进行多次搜索对保存动态规划数据（历史最短距离）的数组进行还原，重置无穷大，是非常浪费时间的，其实在搜索的过程中记录下哪些结点被访问了，只有这些点的g、f、h值发生了改变，因此还原的时候只要对这些结点进行还原就可以了（只要对 g 重赋无穷大），另外一点就是这些结点的 InOpen、InClosed 变量也发生过改变，在还原的时候把这些变量也一并进行还原赋值于 false。

因为第 (3) 步要求每次取出f值最小的结点，因此需要对 OPEN 表进行排序，这也很耗时，可以让插入结点的时候插入到 OPEN 表的适当位置，使前面的结点的f值都小于插入结点的f值，而后继结点的f值则大于它，这样可以保持 OPEN 表的有序性，不必每次都排序，如果要求更好的性能可以使用 Hash 表或二叉树。在第(6-④)步中修改了f值，要求对 OPEN 表重新排序。可以使用先把这个结点脱离整个 OPEN 表，然后再把它插入到合适位置的方法来保持整个表的有序性，而不必对整个表全部进行重新排序。估价函数对整个搜索时间也会产生重大影响，用当前结点和目标点的 x、y 坐标值之差的绝对值之和乘以 10 作为估价函数，即

$$h(n)=(|nx-endx|+|ny-endy|)*10$$

显然比用距离作为估价函数，也就是

$$h(n)=\sqrt{(nx-endx)^2+(ny-endy)^2}$$

能更快地接近目标，并且访问结点通常更少，但前者却不能保证找到的是最短路径，也可根据情况可以给 h(n)加一个权值，让它在前期求快，后期求精。在实际情况中也可能会有些地方根本不可到达，如果一直搜索下去会浪费大量的时间，所以必须设定一个最长搜索时间或是最多搜索的结点数，当超过上限还没到达就认为是不可到达的目标，进而中止搜索；也可以在预处理地图时先标明哪些地方是根本不可到达的，这样也可以提高搜索速度。

表 2 列出了经过改进后的算法运行情况，表中数据均包括对 Nodes 数组的相关数据的重置时间，但不是对整个数组的初始化，而只是对搜索过而发生过改变的结点的相关变量进行重置。表 2 中演示算法使用的估价函数都为

$$h(n)=(|nx-endx|+|ny-endy|)*100$$

表 2　经过优化前后 A*搜索算法运行情况对比

搜索算法	起点	终点	搜索结点数	路径长度	消耗时间(ms)
标准 A*	(52,47)	(103,47)	2 509	118	230
	(81,27)	(51,44)	908	66	31
消除动态内存分配 A*	(52,47)	(103,47)	2 509	118	156
	(81,27)	(51,44)	908	66	16
消除动态内存分配以及表查找的 A*	(52,47)	(103,47)	2 509	118	15
	(81,27)	(51,44)	908	66	0(<1)

从表2可以看出这些优化对搜索性能的提升是十分明显的。

2.3　更真实的路径

要使得 A*算法搜索出来的路径更加真实化，第 1 步也是最基本的一步就是要消除“Z”字效应。这种效应是由于 A*算法搜索了一个格子的周围 8 个格子后才处理下一个格子产生的。这在早期的那种游戏单元只是简单地从一个格子跳到另外一个格子的游戏里是不错的，但是在今天的大多数需要平滑移动的游戏中则是不可接受的。一个简单的减少转弯数目的方法就是每次转弯的时候都增加一个代价值。这样对于距离相同的路径，转弯少的就会被选出来。

对不同的地形可以设定不同的损耗，比如沼泽、小山、楼梯等可以设定比平地更高的移动耗费。公路就应该比自然地形移动耗费更低。解决这个问题只要在计算任何地形的 h (n) 值的时候增加地形损耗就可以了。由于 A*算法已经按照寻找最低耗费的路径来设计，所以很容易处理这种情况。但是在地形耗费不同的场合，耗费最低的路径也许会包含很长的移动距离，就像沿着公路绕过沼泽而不是直接穿过它。

如果大量的寻路者都要通过一个狭小的关口（因为从这里可能是最近的），它就会变得更拥挤，就会产生相互阻塞的情况。对于这些有大量单元频繁经过的格子应该施以不利影响，让后面的寻路者可以选择更远一点但更不拥挤的路径。避免总是仅仅因为路径短(但可能更拥挤)而持续把队伍和寻路者送到某一特定路径。

2.4　层次化的路径搜索

当地图很大或是要跨越地图进行路径寻找时就要用到层次化的路径搜索。在真实世界中，比如要找一个村，如果路径太远，人们可能不知道如何直接到达，但可以先到目标所在的县，再找路到目标所在的乡，最终再找到目的地。在游戏中也是一样，也可以在地图上做一些中转站，使得到达某个区域前先经过它最近的一个中转结点，再从这个中转结点搜索到达这个区域内的目标结点的路径，这样就可以使搜索的目标不至于太远，可以很快地提高搜索速度。但也不宜过多中转结点，这样会增加管理这些结点的成本而且还可能会使最终得到的路径显得很不真实。

2.5　适应动态的状态空间

在游戏中经常会遇到一种情况就是已经搜索好了一条路径中的某些结点产生了变化，变得不可经过了。比如在一个单位要经过的路径中突然建造了一座房子或是另外一个单位正好移动到或是停在它下一步要经过的位置。这样就出现阻塞了，此时就要求搜索算法能适应这种动态变化的状态空间。通常会让它从这一点重新再找一条到终点的路径、等待一段时间或是用其它的改进搜索算法[4-6]。可以用下面的简单方法来解决这个问题并能得到很好平均的搜索效率。如图 2 所示，本来搜索好了一条从 S 经过 a、b、c、d 到达目标 E 的路径，可是当走到结点 a 时发现前面的结点 b 已经发生了改变，变得不可经过了，从 b 出发找到这条路径中的第 1 个可行结点 d，此时不必从 a 点出发重新找一条到 E 点的路径，只需要找一条从 a 到 d 的路径，把这条路径通过链表链接到原来的路径中，退换掉原来路径中的从 a 到 d 部分。这样可以提高效率，因为很多已经求得的路径可以重用，并且搜索相距不远的两个结点中间的路径时效率很高，扩展结点通常很少。这种方法不知道提前回避障碍物，所以真实性方面不够。移动前可以让它检查一定步数而不是只检查前面一步，这样可以提高真实性，但同时也将提高时间消耗。

3　结束语

通过对A*算法的优化改进后，A*算法可以很好地胜任游

图 2 路径阻塞

戏中的路径搜索，但在减少搜索时间的同时也增加了空间的消耗；在路径的真实性方面，游戏提出了越来越高的要求，也产生了很多方法，游戏中角色行走的路径必将越来越人性化，越来越真实。

参考文献:

[1] 严蔚敏,吴伟民.数据结构(C 语言版)[M].北京:清华大学出版社, 1997.167-169.

[2] 庄越挺,吴飞.人工智能游戏编程真言[M].北京:清华大学出版社, 2005. 90-133.

[3] 蔡自兴,徐光祐.人工智能及其应用(研究生用书)[M].北京: 清华大学出版社, 2004. 25-31.

[4] Roth U,Walker M, Hilmann A,et al.Dynamic path planning with spiking neural networks[C].*Spain: IWANN*, 1997. 1355- 1363.

[5] Stentz A. Optimal and efficient path planning for partially-known environments[C]. *San Diego: Proceedings of the IEEE International conference on Robotics and Automation*, 1994. 3310-3317.

[6] 袁曾任,高明.在动态环境中移动机器人导航和避碰的一种新方法[J].机器人, 2000,22(2): 81-88.

(上接第 2323 页)

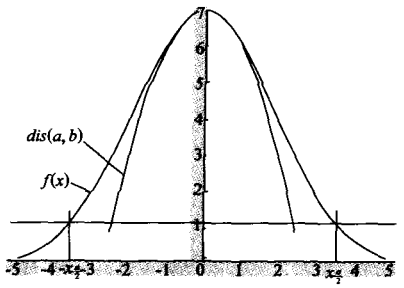

图 3 $f(x)$和$dis(a,b)$的曲线

针对灰度图像 Lena(128*128*8)进行仿真实验，置信度设置分别为 0.96、0.92，表 2 列出了两种编码方案的编码时间、压缩比和 PSNR 经验计算出的数据。实验效果如图 4 所示。

实验结果分析，每一个子块与父块的匹配门限都是根据当前子块的方差计算出来的，$dis(a,b)$的自适应门限 tol 与子块的方差σ^2成正比，当子块的方差大时，则门限高，允许恢复的子块有较大的失真；当子块的方差小时，门限低，将限制恢复的子块有较大的失真程度。另外，子块的方差在一定程度上反映了它的复杂程度，方差大，子块的复杂程度高，且其中的高频信号多；而方差小，子块的复杂度低，其中的低频信号多。许多研究表明，由于人类的视觉系统对不同频率信号的分辨能力是不相同的，对高频信号的分辨能力弱，对低频信号的分辨能力高，所以对复杂的子块(方差大，高频信号多)设置较高的门限，使得该子块的尺寸可以划分得大些，以保证有较高的压缩比；而对于平坦的子块(方差小，低频信号多)可以设置较低的门限，使得恢复图像有着较好的视觉效果。

表 2 自适应门限的四叉树编码方法

置信度	0.92	0.96
分级块的个数(最大子块 64*64,最小子块 2*2)	子块总个数:420 2*2 子块:272 4*4 子块:144 8*8 子块:4	子块总个数:986 2*2 子块:321 4*4 子块:603 8*8 子块:62
编码时间	49S	130S
压缩比	9.91:1	4.62:1
PSNR	26.9	33.6

(a)原始图像 (b)置信度为 0.92 还原图像 (c)置信度为 0.96 还原图像

图 4 Lena 图像经过自适应门限编码后的 10 次解码的图像

3 结束语

本文提出分形图像压缩中的四叉树自适应门限方法，给出结论具体的推导过程，每一个子块与父块的匹配门限都是根据当前子块的方差计算出来，解决了分形图像压缩中匹配门限的取值来源于经验判断的问题，是对Fisher的固定门限四叉树方法的改进和提高，提高了分形图像压缩编码的实用性。

参考文献:

[1] Jacquin A E. A novel fractal block-coding technique for digital Image [C]. *Proceedings of ICASSP IEEE International conference on ASSP*, 1990. 872-891.

[2] Jacquin A E.Fractal image coding: A review[C]. *Proceeding of the IEEE*, 1993. 1451-1465.

[3] Fisher Y. Fractal image compression: Theory and application [M]. *New York: Springer-Verlag*, 1995. 49-51.

[4] Fisher Y.Fractal image compression[J].*Fractals*,1994,(3): 30-52.

[5] 何传将, 李高平. 分形图像编码的改进算法 [J]. 计算机仿真, 2004,(8): 62-67.

[6] 徐谨, 蔡秀云. 基于图像特征的分形图像压缩方法[J].工程图学学报, 2004,(3): 82-85.

[7] 陈守吉,张立明.分形与图像压缩[M].上海:上海科技教育出版社, 1998.

[8] 郭京蕾, 基于分形理论和演化算法的灰度图像压缩[D]. 武汉: 武汉理工大学, 2002.