**文章编号**：1673-5196(2015)03-0102-06

# 利用跳点搜索算法加速 A* 寻路

邱 磊

（武汉船舶职业技术学院 电气与电子工程学院，湖北 武汉 430050）

**摘要**：介绍广泛应用于游戏寻路中的标准 A* 算法，指出跳点搜索(JPS)算法使 A* 生成并扩展的节点数量很少，而且到达目标的速度很快. 因为跳点搜索能够消除路径间的对称性，通过在直线和对角线方向上修剪节点来识别后继，在搜索时跳过了大量可能会添加到 open 列表和 closed 列表中的中间节点以及其他计算，这使搜索速度有了很大提升. 在5个基准网格地图上测试 A*＋JPS 对 A* 的相对加速比，实验结果表明：跳点搜索可将标准 A* 搜索的速度提高一个数量级甚至更多，并且速度收益的程度取决于基础网格地图的地貌，对于大的开放区域，跳点搜索更加高效. 另外，跳点搜索对 A* 在节点扩展数量上的改进甚至比搜索时间的改进更加显著. 无论从搜索时间还是从节点扩展数量上，A*＋JPS 都明显优于 A*，利用跳点搜索算法可显著加速 A* 寻路.

**关键词**：A* 寻路；跳点搜索；网格；游戏

**中图分类号**：TP301.6 **文献标识码**：A

## Speed-up of A* pathfinding with jump point search algorithm

QIU Lei

(College of Electrical and Electronic Engineering, Wuhan Institute of Shipbuilding Technology, Wuhan 430050, China)

**Abstract**: Standard A* algorithm widely employed in game pathfinding is presented and it is pointed out that jump point search (JPS) algorithm will make the quantity of nodes very small for both A* generation and expansion and also make the speed very high to reach the goal. Because the jump point search can eliminate symmetry of paths and identify their successors by pruning nodes straight and diagonally, therefore a bunch of nodes, which would be added to the open list and closed list, will be jumped over and a lot of calculations can be avoided, so that the searching speed will be improved greatly. By measuring the relative speedup ratio of A*＋JPS to A* on five benchmark grid maps, it is shown by this experimental results that jump point search can improve the search time performance of A* by an order of magnitude or more, and the magnitude of the speed gains will be dependent on the topography of the benchmark grid map, and for large open areas, jump point search will be highly effective. Moreover, jump point search can make the improvement on the total expanded number of nodes even more noticeable than the improvement on the search time. As concerning whether the search time or the number of expanded nodes, A*＋JPS is superior to A* obviously, and the A* pathfinding can be speeded up more noticeably with the jump point search algorithm.

**Key words**: A* pathfinding; jump point search; grid; game

在游戏中寻路无处不在，最著名的寻找最优路径的启发式搜索算法莫过于 A* 算法，但该算法大量时间耗费在对 open 列表的操作上，实现得好的 A* 算法会使用优先队列，甚至 HOT(heap on top)来对操作进行优化[1]，但当 open 列表中的节点过多时，这些操作还是会变得很昂贵. 跳点搜索(JPS)算法是一个高效寻路算法，其思想就是跳过矩形平坦区域的大量对称路径，只寻找所谓的“跳跃点”作为搜索的节点，这样做的好处是修剪了矩形区域内大量的节点，使 open 列表中的节点数相对较少. 利用 JPS 算法能够大大提高 A* 寻路的效率，因为它省去

**收稿日期**：2014-10-11

**基金项目**：湖北省自然科学基金(2012FFB4102)，湖北省教育厅科学技术研究计划指导性项目(B2014202)

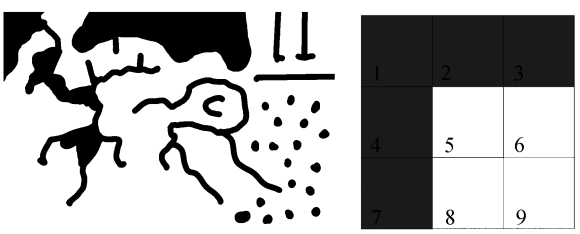
**作者简介**：邱 磊(1982-)，男，黑龙江齐齐哈尔人，硕士，讲师.

了一些无谓的节点判断,当然它需要在寻路之前,开销一些空间和时间进行地图的预处理,不过相对于这点开销,赚回来的效率收益会更多.

本文将简略介绍一下 A*,并研究如何利用一个新算法:跳点搜索,来为 A* 提速,并对比在不同基准网格地图上 A* 与 A* +JPS 在搜索时间和节点扩展数量上的性能差异.

## 1　世界表示方式

当研究游戏中寻路的时候,最先需要考虑的是世界的表示方式.即用寻路算法可以使用的方式来表示游戏世界.可通行区域和障碍物的数据是如何与内存中的程序结构组织的?最简单的表示方式是基于网格的结构,在该结构中,路径节点用网格的形式组织,并可用二维数组来表示.本文便使用这种基于网格结构的表示方式.值得注意的是,它将是八方向网格表示方式,允许在直线和对角线方向运动.




(a) 世界的实际表示方式　　(b) 局部图像放大

**图 1　某游戏场景所使用的世界的实际表示方式**

**Fig. 1　Actual representation of world used in a game development environment**

在图 1 中,图像中的黑色像素代表阻塞单元,图 1a 为某游戏场景所使用的实际世界的表示方式,而图 1b 则是对该实际世界的表示方式进行的局部图像放大.当需求有所不同的时候,该结构可能不再适合.不过,稍稍做些处理(通常离线操作)就可以将寻路的表示方式更改为其他格式.如果不采用基于网格的方法,也可以用包含诸如多边形(障碍物用多边形表示)或导航网格(导航区域用多边形表示)的方法;后者只要用更少的节点就可以表示相同的数据.在地图的表示方式中,可以存储的另一种数据是 cost,代表从一个节点行进到另一个节点所需的代价,可被 AI 用来确定具体路径.例如,倾向去选择道路而非自然地形,因为道路的成本小于地形的成本.

跳点搜索为了适应八方向网格地图的表示方式,进行了专门的设计,只能在规则的网格地图上寻路,普通形式下的跳点搜索不支持加权的地图,地图上的点或边不能带权重,也就是不能有复杂的地形,只支持平坦和障碍两种地形.

## 2　A* 寻路

现在已经有了世界的表示方式,接下来给出 A* 的实现.A* 是一个图搜索算法,对开始节点到目标节点的区域进行启发式搜索[1].在矩形网格中寻路时使用 A*,执行以下步骤:

1) 获取最接近当前位置的节点,将其声明为开始节点并添加到 open 列表.

2) 当 open 列表中有节点,从 open 列表中挑选具有最小 $F$ 值的节点,将它添加到 closed 列表(表示不想再对其进行考虑).

3) 对于每一个不在 closed 列表中的邻居(相邻的单元),将它的父节点设为当前节点.

4) 计算 $G$ 值(从开始节点至该邻居的距离)并将它添加到 open 列表中.

5) 将 $G$ 值加上启发值求得 $F$ 值.

启发式搜索本质上是猜测正在评估的节点能够到达目标的几率,它会导致寻路算法的效率随着所需访问节点的数量的变化而产生巨大波动.为此将使用曼哈顿距离(意味着节点越靠近目标,曼哈顿距离值越小):

```
private function manhattanDistance(start:Node, end:Node):int {
    return Math.abs(end.x - start.x) + Math.abs(end.y - start.y);
}
```

当找到目标节点的时候就终止算法,然后利用节点的父节点变量回溯以构建路径.

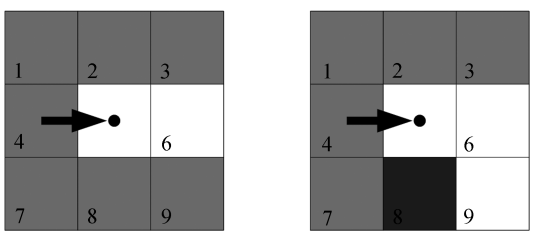
## 3　跳点搜索

本文主要是关于实现跳点搜索的,寻路地图将被表示成网格.与其他算法不同的是,跳点搜索的寻路地图需要一个八方向的网格以便算法直接使用它.跳点搜索能用来消除某类网格聚合的众多中间节点.跳点搜索跳过了大量可能会添加到 open 列表和 closed 列表中的中间节点以及其他计算,而对下一个节点的挑选过程,则需要做更多的处理[2-4].在 A* 里,会从 open 列表中选择具有最小 $F$ 值的节点,但跳点搜索算法(JPS)则不然,不是对相邻节点进行选择,而是调用以下函数[2]:

```
function identifySuccessors(current:Node, start:Node, end:Node):Vector.<Node> {
    var successors:Vector.<Node> = new Vector.<Node>();
```

```
var neighbours:Vector.<Node> = nodeNeighbours(current);
for each (var neighbour:Node in neighbours)
{
    //从当前节点到邻居的方向:
    var dX:int = clamp(neighbour.x-current.x,-1,1);
    var dY:int = clamp(neighbour.y-current.y,-1,1);
    // 试着寻找一个节点来跳跃:
    var jumpPoint:Node = jump(current.x,current.y,dX,dY,start,end);
    //如果找到,将其加入到列表中:
    if (jumpPoint) successors.push(jumpPoint);
}
return successors;
}
```

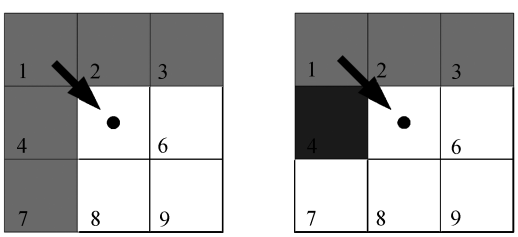
上述函数所实现的是为路径消除累赘的节点,该函数将使用父节点的方向作为主要指导方向.下面是修剪节点的例子(见图 2 和图 3),这些节点将在水平和垂直方向上忽略.




(a) 修剪颜色为灰色的邻居 (b) 阻塞单元要求须做额外的处理

**图 2 一种水平修剪情况的例子**

**Fig. 2 Example of a horizontal pruning situation**




(a) 无阻塞单元 (b) 有阻塞单元的情况

**图 3 对角线修剪情况的例子**

**Fig. 3 Example of diagonal pruning situations**

在程序代码中,节点的修剪以一系列 if 语句对相关情况进行检查而结束.可以看看下面的程序例子,描述了图 3b 的情况[2]:

```
if(directionY==0) {
    if (! _world.isBlocked(current.x+directionX,current.y)) {
        if(_world.isBlocked(current.x,current.y+1)){
            //在新位置创建一个节点
            neighbours.push(
                Node.pooledNode(current.x+directionX,current.y+1));
        }
    }
}
```

选完邻居之后,试着寻找一个跳点:一个可以从当前节点到达、但未必仅以单一道路到达的节点.专业点讲,JPS 旨在消除路径间的对称性——每条路径具有相同的移动序列,但是排列方式不同.如图 4 所示,各条路径均具有相同的长度、相同的起点和终点.

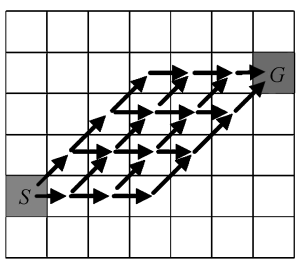



**图 4 路径对称性的例子**

**Fig. 4 Example of path symmetry**

因此,对于大的开放空间,JPS 更能显现出它的优势,能够远胜于其他寻路算法[5-6].下面介绍 jump 方法是如何工作的:

```
function jump(cX:int,cY:int,dX:int,dY:int,start:Node,end:Node):Node {
    //cX,cY 代表当前节点位置;dX,dY 代表方向
    //将要考虑的新节点的位置
    var nextX:int=cX+dX;
    var nextY:int=cY+dY;
    // 如果该节点是阻塞的,那么不能从这里跳跃,返回 null
    if (_world.isBlocked(nextX,nextY)) return null;
    // 如果该节点是目标点,则返回该节点
    if (nextX==end.x && nextY==end.y) return new Node(nextX,nextY);
    // 对角线情况
    if(dX!=0 && dY!=0) {
```

```
        if (/* … 对角线方向的被迫邻居的检查 … */) {
            return Node.pooledNode(nextX, nextY);
        }
    //分别在水平和垂直方向上检查被迫邻居
    //这是对角线方向的一个特例
        if (jump(nextX, nextY, dX, 0, start, end) != null ||
            jump(nextX, nextY, 0, dY, start, end) != null)
        {
        return Node.pooledNode(nextX, nextY);
        }
    } else {
        // 水平线情况
        if (dX != 0) {
            if (/* …水平方向的被迫邻居的检查 … */) {
                return Node.pooledNode(nextX, nextY);
            }
        // 垂直情况
        } else {
            if (/* …垂直方向的被迫邻居的检查 … */) {
        return Node.pooledNode(nextX, nextY);
            }
        }
    }
    // 如果未发现被迫邻居,继续寻找跳点
     return jump(nextX, nextY, dX, dY, start, end);
}
```

由于 if 语句里对于被迫邻居检查的篇幅较大,在这里省略了.这部分主要由一系列检查组成,这些检查与一开始对新节点邻居的选择的做法类似(通过诸多检查来判断单元是否是阻塞的),可以帮助检测在对称性上的各种假设.

如图 5 所示,虚线未能阻止对角线跳跃.共发生了 4 个跳跃:

1) 从起点 $S$ 到 $x$,对角线式的;

2) 从节点 $x$ 到 $y$,水平式的;

3) 从节点 $y$ 到 $z$,对角线式的;

4) 从节点 $z$ 到终点 $G$,垂直式的.

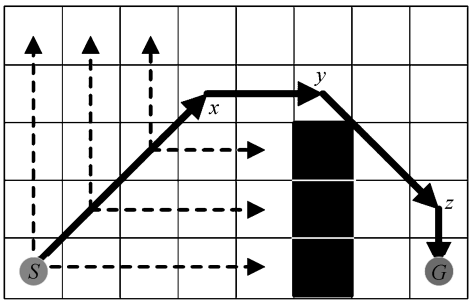



**图 5　jump 函数行为的例子**

**Fig. 5　Example of jump function behavior**

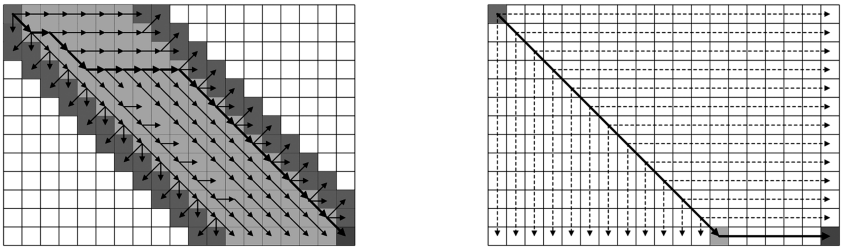
对角线的情况比较特殊,不要只在对角线方向上寻找被迫邻居,也要在水平和垂直方向上寻找.如果在上述方向上都未找到被迫邻居,那么只好将被迫节点归为跳点.

算法执行时,每当未找到一个搜索节点时,就将在指定的方向上递归地调用 jump 函数.在算法运行过程中,由于这个递归被大量调用,实际上已经展开了这个递归调用.JPS 算法基于如下假设:以较少次数的 A* 迭代来访问网格上很多节点的内容,将比通过较多次数的 A* 迭代来维持一个优先级队列更有效.JPS 在搜索时可以跳过很多节点,使搜索速度有了很大提升.以上就是 JPS 的实现,最终的结果是提供给 A* 进行检查的新节点,然后继续执行算法.当目标节点找到了,就可以重构路径并将其返回.

## 4　实验结果

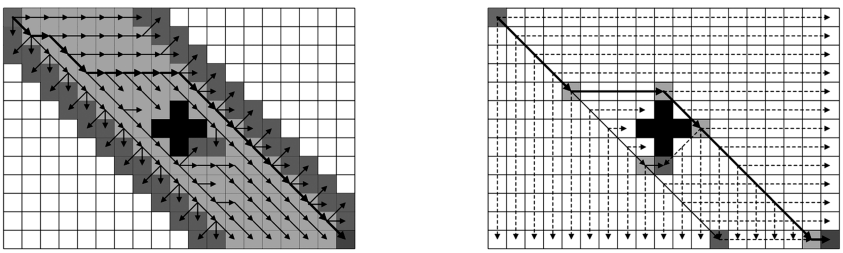
本文的跳点搜索算法的实现是基于 Harabor 等人在 JPS 方面的论文[7-8].测试用的计算机性能为:主频为 2.10GHz 的英特尔酷睿 2 双核处理器,内存大小为 2G,运行的系统为 Windows 7.本实现可以运行 A* 和 A* +JPS,单击并拖动到任意位置可以添加障碍物,拖动开始 $S$ 和目标 $G$ 节点可以移动其自身.

以 5 个网格地图为基准对 JPS 进行测试,如图 6 所示,同时给出 A* +JPS 起作用的截图.在解决相同的问题实例时,为了便于比较,同时展示 A* +JPS 与普通 A* 算法探索“瓷砖”的数量,在每种情况下,为了找到最优路径,标记为浅灰色和深灰色的“瓷砖”必须被探索.其中:第一列中的图展示 A* 算法扩展的全部节点,以便找到至目标点的最优路径(以黑色粗实线标记);第二列中的图为 A* +JPS 扩展的全部节点,注意到 A* +JPS 忽略许多对称的路径分段,不出所料地用较少的花费到达了目标点.



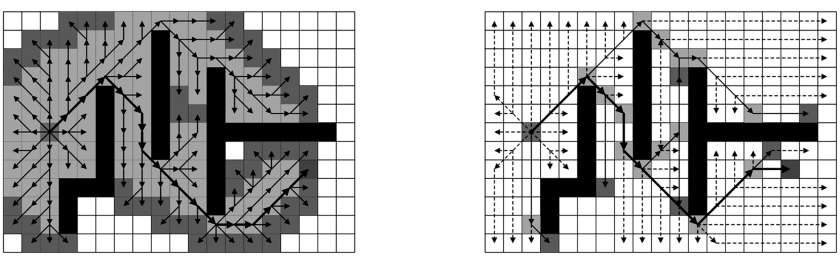
A* A*+JPS

(a) 基准网格地图1



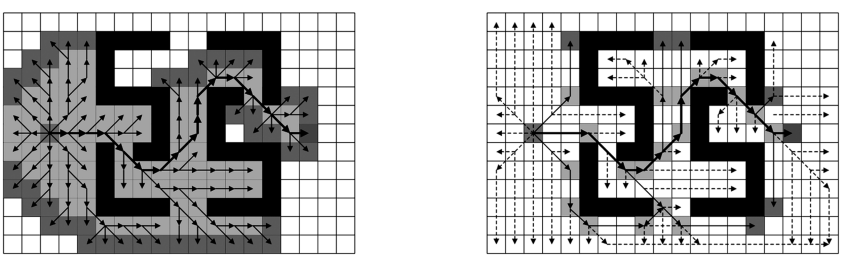
A* A*+JPS

(b) 基准网格地图2



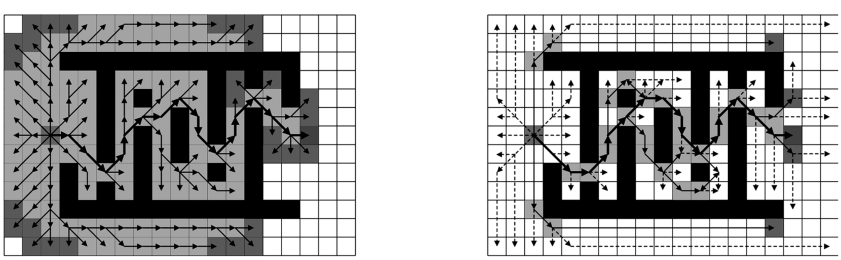
A* A*+JPS

(c) 基准网格地图3



A* A*+JPS

(d) 基准网格地图4



A* A*+JPS

(e) 基准网格地图5

**图 6 A* 和 A* +JPS 的寻路效率比较**

**Fig. 6 Comparison of pathfinding efficiency between A* and A* + JPS**

表 1 给出了 A* 与 A* +JPS 在 5 个基准网格地图上的搜索时间以及两者的相对加速比(即搜索时间的比值),在每个基准网格地图上,通过计算 A* +JPS 对单独 A* 的相对加速比,看出 JPS 可以将 A* 的速度提高 3～20 倍,较低的数字和较高的数字分别代表短寻路问题和长寻路问题的平均性能(也就是说,找到的最优路径越长,越得益于应用了 JPS),JPS 可以将 A* 搜索的速度提高一个数量级甚至更多.强调 A* +JPS 能完成此加速比是很重要的,因为每个基准问题包含有大量的对称路径分段(通常以地图上大的开放区域的形式来显现).在这种情况下,JPS 能够利用对称性并忽略大部分搜索空间,在这一过程中,使 A* 生成并扩展的节点数量很少,而且到达目标的速度很快.

**表 1　A* 与 A* +JPS 搜索时间的对比**

**Tab. 1　Comparison of search time between A* and A* + JPS**

| 基准网格地图 | 搜索时间/s | | 相对加速比/倍 |
|---|---|---|---|
| | A* | A* +JPS | |
| 1 | 9.8 | 0.5 | 19.60 |
| 2 | 8.9 | 0.8 | 11.13 |
| 3 | 11.6 | 2.0 | 5.80 |
| 4 | 9.0 | 2.3 | 3.91 |
| 5 | 12.2 | 3.4 | 3.59 |

另外,发现速度收益的程度将取决于基础网格地图的地貌,若这些地图上以大的开放区域为特征,JPS 是高效的(如图 6a、b);当这些条件不存在时,速度收益也是适度的(如图 6c～e).

A* +JPS 在穿过所有的基准网格地图所需的搜索时间上,显现出了令人信服的提升,在节点扩展数量上也能观察到类似的趋势.表 2 给出了 A* 与 A* +JPS节点扩展数量以及两者的比值,可以看出:JPS 对 A* 在节点扩展数量上的改进甚至比搜索时间的改进更加显著.总之,无论从搜索时间还是从节点扩展数量上,A* +JPS 算法都明显优于 A* 算法.

**表 2　A* 与 A* +JPS 节点扩展数量的对比**

**Tab. 2　Comparison of node expansion quantity between A* and A* + JPS**

| 基准网格地图 | 扩展数量/个 | | 扩展节点数量之比/倍 |
|---|---|---|---|
| | A* | A* +JPS | |
| 1 | 135 | 1 | 135.00 |
| 2 | 130 | 7 | 18.57 |
| 3 | 158 | 21 | 7.52 |
| 4 | 125 | 25 | 5.00 |
| 5 | 141 | 32 | 4.41 |

## 5　结语

寻路在游戏中可谓无所不在.因此当使用诸如 A* 之类的算法时,理解其呈现出的含义是很重要的.本文介绍了一种相对较新的基于网格世界的搜索方法——跳点搜索算法,该算法是一个消除对称性的技术,能够有效地识别和消除路径间的对称性,在搜索时可跳过很多节点,因此大幅度减少了节点扩展的数量,加快了寻路的速度.本文在 5 个基准网格地图上评估了 JPS,给出了 A* 和 A* +JPS 起作用的截图,并从搜索时间和节点扩展数量两个方面对比了 A* 和 A* +JPS.实验结果表明,A* +JPS 可以将 A* 提速一个数量级甚至更多.

**参考文献:**


[1] PATEL A. Variants of A* [EB/OL]. (2013-07-18) [2013-11-01]. http://theory. stanford. edu/～ amitp/GameProgramming/Variations. html.

[2] PODHRASKI T. How to speed up A* pathfinding with the jump point search algorithm [EB/OL]. (2013-03-12) [2013-11-01]. http://gamedev. tutsplus. com/tutorials/implementation/speed-up-a-star-pathfinding-with-the-jump-point-search-algorithm.

[3] 邱　磊. 基于在线图修剪的网格地图寻路 [J]. 贵州大学学报:自然科学版,2014,31(1):84-87.

[4] WITMER N. Jump point search explained [EB/OL]. (2013-05-05) [2013-11-01]. http://zerowidth. com/2013/05/05/jump-point-search-explained. html.

[5] 邱　磊. 基于打破对称性的快速寻路算法 [J]. 宁夏大学学报:自然科学版,2014,35(3):216-220.

[6] XU X. Pathfinding visual [EB/OL]. [2013-11-01]. http://qiao. github. io/PathFinding. js/visual.

[7] HARABOR D, GRASTIEN A. Online graph pruning for pathfinding on grid maps [C]//Proceedings of the 25th National Conference on Artificial Intelligence (AAAI). San Francisco: [s. n.], 2011.

[8] HARABOR D, GRASTIEN A. The JPS pathfinding system [C]//Proceedings of the 5th Symposium on Combinatorial Search (SoCS). Niagara Falls: [s. n.], 2012.