跳点搜索算法的原理解释及性能分析*

邱 磊，刘辉玲，雷建龙

(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：给出了跳点搜索(Jump Point Search, JPS)算法的原理，分析了邻居裁剪规则，并试着用图来解释该算法而不诉诸于其原始研究论文中提出的基本数学证明．通过3个实验综合分析了JPS的性能优势，实验结果表明：同等地图尺寸下JPS扩展的节点数与障碍物密度成正比，与查看的邻居数成反比；随着地图尺寸的增加，JPS相比于其他典型寻路算法，在时间效率上优势更加显著；地图环境的对称性越高，JPS较之于A*的优势越明显．总之，JPS保持了A*的最优性，可将A*提速一个数量级甚至更多，该算法更适合需要快速寻路的领域．

关键词：寻路；跳点搜索；A*；网格；环境对称性

DOI：10.13568/j.cnki.651094.2016.01.013

中图分类号：TP301.6 **文献标识码**：A **文章编号**：1000-2839(2016)01-0080-08

Principle Explanation and Performance Analysis of Jump Point Search Algorithm

QIU Lei, LIU Huiling, LEI Jianlong

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

Abstract: The principle of JPS algorithm is given and neighbors pruning rules are analyzed, and explained this algorithm without resorting to the underlying mathematical proofs as presented in the original research papers.The performance advantages of JPS are analyzed through three experiments,experiment results show that under the same map size, the number of JPS expanded nodes is proportional to the density of the obstacle and is inversely proportional to the number of neighbors that JPS need to look at; with the increase of map size, JPS is more pronounced than other typical pathfinding algorithms in time efficiency; the higher symmetry of map environment, the more obvious advantages of JPS compared with A*. In conclusion, JPS preserves optimality of A*, which can speed up A* by an order of magnitude and more, it is suitable to use in cases where it is necessary to quickly find a path.

Key words: pathfinding; Jump Point Search; A*; grid;environment symmetry

0 引言

A*算法通过裁剪代价较高的邻居而发现最优路径，这不利于游戏开发和机器人寻路．跳点搜索(JPS)[1]是一种新的寻路算法，是对A*搜索算法的优化．它通过图裁剪来减少搜索过程的对称性，能让搜索在网格上直线长“跳”，而不是普通A*的小步移动．JPS跳过了无用节点，保留了关键的“跳点”．JPS擅长应用在等价网格地图上大的开放区域．在这些开放的区域，JPS能略过或跳过大量使用传统A*算法将被扩展的中间节点．然而，JPS通常要比A*评估更多的障碍物，但大多数情况下，JPS的额外评估要比A*的额外列表操作快得多．在最坏情况下，在搜索时间上JPS与A*也能打成平手；至于大型的开放空间，JPS生成

* 收稿日期：2015-08-26

基金项目：湖北省自然科学基金项目（2012FFB4102）；湖北省教育厅科学技术研究计划指导性项目（B2014202）；湖北省教育厅人文社会科学研究青年项目（15Q263）．

作者简介：邱磊（1982-），男，硕士，讲师，主要研究方向为网格地图寻路．

最优路径要远快于A*. 跳点搜索算法比传统的A*不仅提升了性能而且降低了内存成本，跳点搜索算法将是游戏与机器人领域人工智能的未来，甚至可以应用在全球定位系统、仿真等领域.

本文的创新点如下：1）主要特色是采用图的方式，对JPS算法进行了直观的描述和解释，对算法赋予直观的物理含义，是对理论方法的一种拓展；2）既使用了规则的2D方形网格地图又使用了测试库中的基准地图来表示寻路环境进行仿真实验，使JPS算法的性能分析结论更具说服力；3）对同等地图尺寸与变化地图尺寸下JPS的性能均进行了比较分析. 本文待解决的问题如下：1）拟尝试用图来解释JPS算法而不诉诸于在其研究论文中给出的基本数学证明；2）拟结合仿真实验分别从同等地图尺寸下障碍物密度的变化对JPS扩展节点数和查看邻居数的影响、地图尺寸的变化对JPS与其它典型寻路算法的时间效率趋势的比较影响，以及基准地图环境的对称性特征对JPS算法与A*算法的比较影响程度三个方面综合分析JPS的性能优势. 所做工作的价值和意义如下：1）用图来解释理论算法是一件有意义的工作，对具体应用有更好的指导价值；2）综合全面分析了JPS的性能，弥补了已有文献对JPS性能分析的不足.

1　相关工作

使用2D网格来代表地图是非常普遍的方式，在等价2D网格上有很多与寻找最短路径相关的算法. A*算法是一个常见且简单的广度优先搜索和深度优先搜索的优化，它作为通用的搜索工具仍具有强大的生命力. A*算法有很多扩展或变种[2]，跳点搜索(JPS)算法只是其中之一：1985年Richard E.Korf提出的IDA*(Iterative Deepening A*)算法[3]是A*算法和迭代加深算法的结合，通过调整阀值进行深度优先搜索来找到最优解；1994年Anthony Stentz提出的D*(Dynamic A*)算法[4]研究了在不知道全部地图信息的情况下寻路，可以快速改正A*算法由于信息未知而导致的计算错误；2004年Sven Koenig等提出的LPA*(Lifelong Planning A*)算法[5]研究了在地图代价发生改变的时候，如何重复利用以前的A*计算来产生新的路径；2004年AdiBotea等提出的HPA*(Hierarchical Pathfinding A*)算法[6]是一个在网格地图上降低问题复杂度的分层寻路方法，该技术将地图抽象成相连接的局部簇. 2005年Maxim Likhachev等人提出的AD*(Anytime Dynamic A*)算法[7]研究了如何让搜索算法在复杂的环境和有限的时间内收敛于次优解的问题；2007年Alex Nash等人提出了Theta*算法[8]，该算法在方形网格上寻路不必严格遵循网格的限制，允许任意角度的路径；2010年Daniel Harabor等提出的RSR(Rectangular Symmetry Reduction)算法[9]可以看作是一种预处理算法，它通过将等价网格地图分解为一组中空的矩形区域来识别对称性，从而提高寻找最优路径的速度. 文[1]中提出的JPS算法是一种在方形网格上寻路更加有效率的方法，上述这些A*算法的扩展或变种与JPS解决的问题类似.

2　算法模型

2.1　跳点搜索的原理

JPS算法是基于未修改的A*是相当浪费的，花了大量时间去评估一些无需评估的节点. JPS算法能够让A*寻路更上一个档次，JPS对此是这样改进的：识别并选择性地扩展网格地图上的仅仅某些节点，称之为跳点，两个跳点连接的路径上的中间节点将不被扩展，该方法总能计算出最优解，且能将A*算法的速度提高一个数量级甚至更多[1].

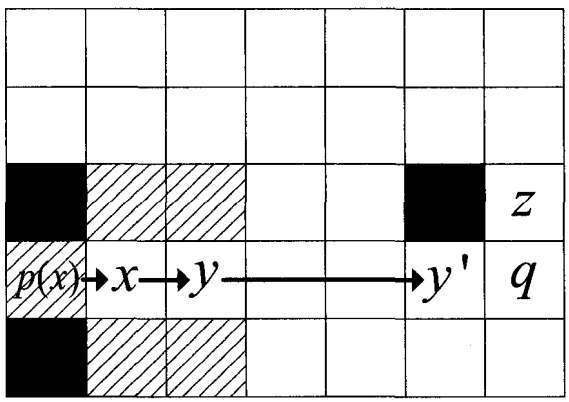
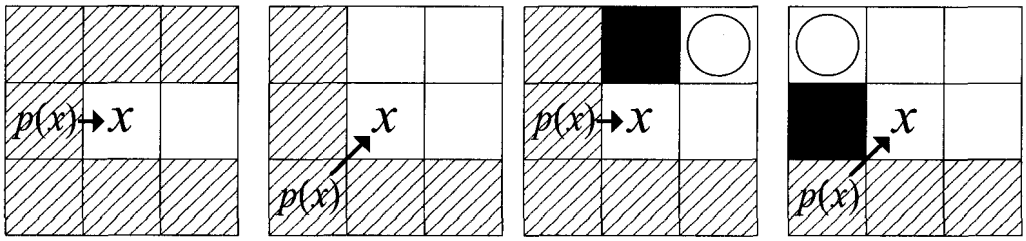


图 1　以直线移动寻找后继

假设搜索到了节点x处，该节点是从父节点$p(x)$直线向右移动而至. 由图1，评估以灰色条纹标记的节点是毫无意义的：因为经由同时穿过$p(x)$和x的路径去往这些节点中的任何一个的成本永远不会比在不访问x的情况下从$p(x)$去往这些节点更低.

在通常情况下是没有障碍物毗邻节点的，如图2a所示，若以直线移动抵达某节点，则只有1个单一的

邻居需要被考虑；若对角线移动后抵达某节点，则有3个节点需要被考虑．假设地形移动成本是一致的，为了找到最短路径，以灰色条纹标记的节点不需要被评估．在搜索中，有部分节点的扩展是无用的，如图2a所示，当前按照箭头方向搜索到了节点x，至此所有带有灰色条纹的节点都是无用的，因为总是存在一条最近路径可以从节点x的父节点$p(x)$出发，不经过节点x，而最终到达带有灰色条纹的节点．但有一些节点比较特殊，如图2b所示，同样，以灰色条纹标记的节点都是无用的，而被黑圈圈住的节点，是不存在一条不经过节点x的最近路径可以从节点x的父节点$p(x)$出发而到达．被黑圈圈住的节点就是特殊的节点，称之为被迫邻居[10−12]，它们是节点x的后继节点，要到达它们就必须经过节点x，否则就不是最近路径．



(a)无障碍物毗邻节点　　(b)有障碍物毗邻节点

图 2　邻居裁剪规则

所以，图1中唯一需要注意的邻居是位于节点x直接右侧的邻居，将其称之为y，该节点可经由x抵达，且成本比从$p(x)$出发但不经过x的路径低．然而，不是停在那儿去检查y的每个毗邻的节点，JPS所做的是继续向右移动，直到抵达了某节点——该节点满足它的至少一个邻居（已访问过的邻居除外）需要被明确地评估．这样的节点被标记为y'，在这个节点的正上方有一个阻塞节点，由于这个阻塞节点，经过y'抵达z是最好的（可对照不触碰y'抵达z的路径），z就是这样的一个邻居．因此当到达了y'时，必须既考虑z又考虑y'直接右侧的节点（标记为q）．因此y'是一个跳点：增加y'作为x的邻居并将其添加到open列表，以供进一步考虑．即当从x沿着直线移动到后继节点y'时，无需评估沿线路径上的任何邻居节点．至于对角线移动，情况是类似的：首先识别哪些毗邻的邻居节点值得考虑，以便开始；然后在所有相关的方向上移动，直到找到x的实际后继节点．总之，JPS实际上是在搜索中寻找跳点的后继节点而转移状态，而不是直接从当前节点进行扩展，为JPS寻找后继可简化为只考虑两种主要的可能性：一、以直线方式抵达某节点；二、进行对角线步后抵达某节点[13]．JPS跳过了部分无用节点，保留关键的“跳点”，令A*在搜索时减少了扩展的节点数，从而达到加速的效果．

2.2　跳点搜索的图解释

A*算法扩展其搜索的方式是：将一个节点的直接邻居添加到下一步要检查的集合．假使能够再向前多看几步，直接跳过某些凭直觉就不值得查看的节点，从而在扩展搜索时，识别出对称路径[1,14]，进而忽略某些节点．所谓“对称路径”是指多条等效的最佳路径的代价相等，唯一的区别在于选择以何种序列沿着对角线方向或水平方向移动．等价网格有助于高度的路径对称性，尽管JPS算法可以被定制为适应其他类型的网格，但它更适合于遍历8路网格地图．

2.2.1　水平/垂直方向移动

本节基于水平/垂直方向上移动来扩展搜索．在一个开放的网格上，节点x向右移动，见图3a所示．给出一些关于该节点x的直接邻居的假设：首先，可以忽略“来自”的节点$p(x)$，因为已经访问过它了，该节点以灰色条纹标记，见图3b．其次，假定当前节点斜后方的节点已通过当前节点x的父亲$p(x)$到达，因为相比于经过当前节点x而到达，该走法在路径长度上更短，见图3c．当前节点x正上方和正下方的节点也可以从父亲$p(x)$更优地到达，倘若经过当前节点x而到达的话，则代价为2而不是$\sqrt{2}$，因此也可以忽略这两个节点，见图3d．斜前方的邻居可以通过当前节点x正上方和正下方的邻居到达，这与从$p(x)$经过当前节点x而到达的路径所花费的成本相同，因此同样可以忽略这些斜前方的节点，见图3e．至此，只剩下当前

节点x直接右侧的邻居要检查，因已经假设当前节点x的所有邻居可通过可供替代路径而到达，因此可以只专注于这个单一的邻居，见图3f. 进而只要前行道路上没有障碍物，即道路是畅通无阻的，就可以直接向右侧跳跃多个节点，并且在不添加节点到open集合的情况下重复节点的检查，见图3g.

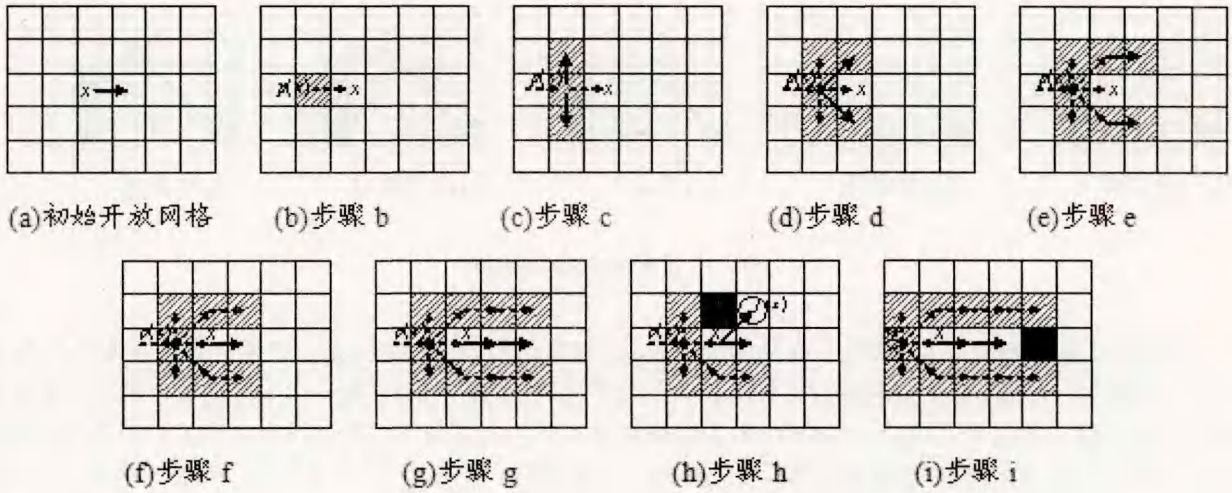


图 3　水平向前移动的步骤图

但是前行道路上没有障碍物并不意味着以上这些假设永远正确且一直不需要停下来做进一步查看. 对于当前节点x斜前方的节点，假定任何等价的路径都将经过当前节点x正上方和正下方的邻居，但是有一种情况，并非如此：即当前节点x正上方或正下方为障碍物从而阻塞了道路. 此时，必须停下来重新检查，不仅必须查看当前节点x右侧的节点，也要被迫查看当前节点x斜上方的节点$f(x)$，由于节点$f(x)$是被迫考虑的而在其它情况下则是已忽略的，在文[1]中，节点$f(x)$称之为“被迫邻居”. 因此，当到达一个含有被迫邻居的节点x时，需停止向右跳跃，并将当前节点x添加到open集合中，以便以后做进一步的检查和扩展，见图3h. 最后一个假设：当向右跳跃时，如果道路被阻塞了，那么可以直接忽略整个跳跃. 已经假设当前节点x正上方和正下方的路径经由其他邻居节点生成，同时由于对角线方向存在一个被迫邻居，因此路径继续生成. 所以只需关心当前节点x的最右侧是什么，若有一个障碍物，则意味着无处可去，见图3i[15].

2.2.2　对角线方向移动

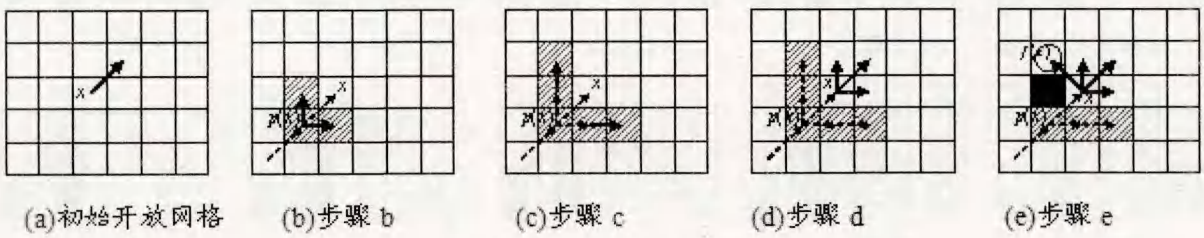


图 4　对角线向前移动的步骤图

当沿着对角线方向移动时，可应用与水平/垂直移动相似的简单假设. 本节基于向右上方移动来扩展搜索，见图4a. 首先假设：当前节点x左侧和正下方的邻居可以直接从当前节点x的父亲$p(x)$通过垂直或水平移动而最优地到达，见图4b. 对于当前节点x左上方和右下方的节点，也可以给出相同的假设，这些节点同样可以通过当前节点x左侧和下方的邻居而到达，而且效率更高，见图4c. 还余下3个邻居需要考虑：其中2个分别位于当前节点x的正上方以及右侧，另1个在当前节点x的对角线方向，见图4d. 与水平移动时讨论的被迫邻居相似，当在当前节点x的左侧（或正下方）存在一个障碍物时，则左上方（或右下方）的邻居无法以其它方式到达，除非经过当前节点x，这些邻居就是关于对角线移动时的被迫邻居，见图4e. 沿

着对角线移动时，当有上述3个邻居需要被考虑时，可以首先在当前节点处沿垂直或水平方向查看一下，如果在这两个方向上都无法寻找到任何值得查看的节点，那么就沿着对角线方向再多移动一步，接着再重复上述过程[15].

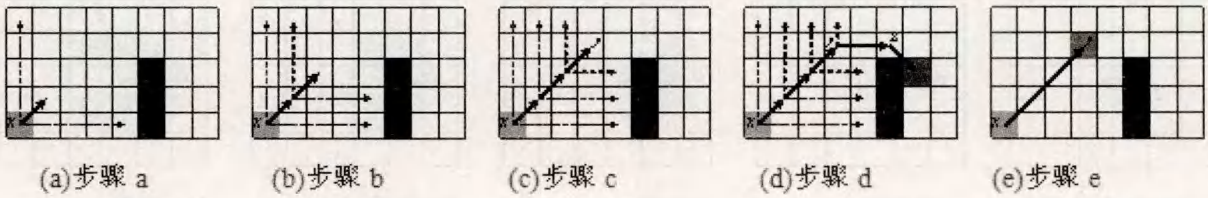


图 5 对角线跳跃的步骤图

图5是对角线跳跃的逐个步骤. 在从当前节点x沿着对角线方向移动之前，需先考虑其在水平与垂直方向上的路径. 首先，在水平和垂直方向上扩展，由于这两个方向上的跳跃均止于障碍物（或地图边界），所以需要沿着对角线方向移动，见图5a. 由于在垂直和水平两个方向上的扩展再次遇到了障碍物（或地图边界），因此仍需要沿着对角线方向移动，见图5b. 以此类推，第三次……，见图5c. 最后，当从节点y沿垂直方向扩展抵达了地图边界时，扩展转而沿水平方向向右跳跃至节点z，该跳跃带出了节点z的一个被迫邻居$f(z)$，见图5d. 因此，遇到这种情况时需要把当前节点y添加到open集合，并继续执行A*的下一次迭代，见图5e.

2.3 算法模型实例

2.2节分析了当朝一个特定的方向行进时跳过节点大多数邻居的方法，也确定了一些何时向前跳跃的规则. 为了使这些规则约束到A*算法上，每当检查open集合中的某一节点时，运用这些“向前跳跃”的步骤并使用该节点的父亲来确定行进的方向，同时尽可能远地向前跳跃. 如果找到了一个感兴趣的节点，将忽略所有中间步骤并将该新节点添加到open集合中[16]. open集合中的每个节点都基于它们父亲的方向而扩展，图6是算法模型实例的路径扩展序列，并在图6j标记出了最终路径.

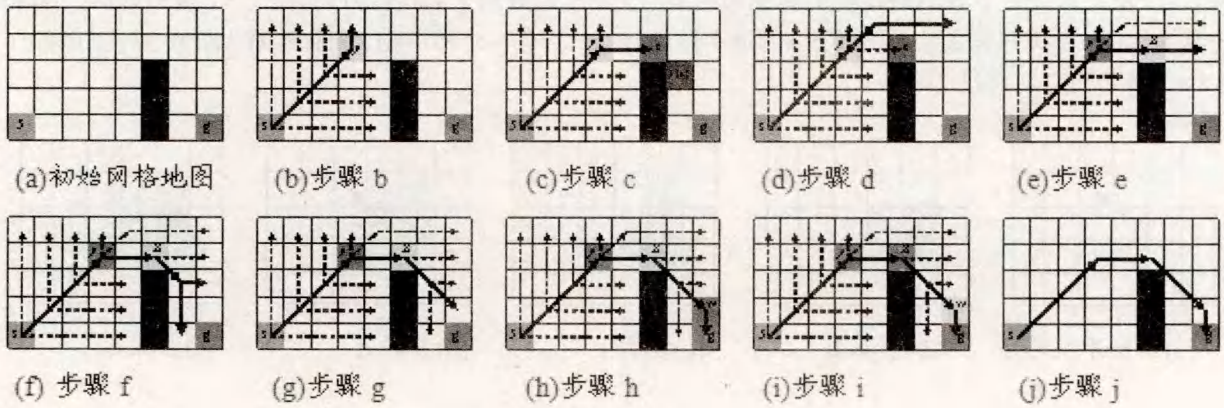


图 6 算法模型实例

本节实例地图与2.2.2节相同，但标记了目标节点g. 图6a为初始网格地图，本路径扩展实例从先前已识别出的节点y（即图5d所识别出的节点，也是open集合中唯一的节点）开始，在垂直方向扩展，结果抵达了地图边界，未找到目标点，见图6b. 沿水平方向向右跳跃，发现了一个含有被迫邻居的节点z（被迫邻居$f(z)$以紫色高亮显示），见图6c，将节点z添加到open集合. 然后，沿着对角方向扩展，结果未找到目标点，因为撞上了地图的边缘，见图6d. 接下来，检查下一个open节点z，由于到达节点z时正在水平移动着，因此扩展继续沿着水平方向向前跳跃，见图6e. 鉴于节点z有一个被迫邻居$f(z)$，不妨沿着$f(z)$方向扩展，

依据对角线跳跃规则，需先沿着对角线方向移动，再在垂直和水平两个方向上查看，见图6f. 再次沿着对角线方向移动，并未找到目标点，见图6g. 此时，继续执行水平与垂直方向上的扩展：水平方向上将导致无路可走，垂直方向上则发现了目标节点g，见图6h，这与找到一个含有被迫邻居的节点一样有趣，因此添加该节点w到open集合. 最后，扩展最后一个open节点w时抵达了目标，见图6i. 跳过该算法的最后一次迭代——添加目标点g本身到open集合，同样识别出了该目标点g——找到了一条最佳路径，见图6j[15,17].

3　实验与分析

实验1：为了测试A*与JPS算法所扩展节点数量的不同，在向队列中放置邻居和从队列中移除f值最小的元素时，两算法使用了完全相同的代码，唯一不同之处在于寻找放置到队列中的邻居的过程. 每次运行时，首先初始化100×100的网格，其上的初始节点和目标节点随机生成. 对于每次迭代，障碍物的数量是没有配额的，所有的节点均有设置为障碍物的可能性，但机率很小. 首先在网格上执行A*算法，并记录扩展的节点与被查看的邻居. 然后，清除网格上所有有用信息（即：启发值，键值以及从初始节点所评估的代价都将被删除）. 接着，再在网格上执行JPS算法，同样记录扩展的节点与查看的邻居. 最后，网格被彻底清除干净，开始新一次的运行. 共执行1000次运行，表1的结果为实验1运行时扩展节点数和查看邻居数的单次平均值.

在表1中，网格上障碍物的密度即是每个节点被设置为障碍物的机率. 对于JPS来说，障碍物的密度越大，则扩展的节点越多，查看的邻居越少；障碍物的密度越小，则扩展的节点越少，查看的邻居越多. 由表1数据可知，JPS扩展节点的数量与查看邻居的数量成反比.

表 1　扩展节点数与查看邻居数的单次平均值

地图尺寸	障碍物密度	平均扩展的节点数		平均查看的邻居数	
		A*	JPS	A*	JPS
100×100	1/100	577	73	4563	6921
	1/75	575	89	4532	6634
	1/50	561	116	4390	6179
	1/25	576	175	4418	5141
	1/20	567	191	4298	4591
	1/15	558	215	4157	4065
	1/10	514	234	3695	3135

实验2：为了说明JPS算法的优越性，对比JPS与其他典型的寻路算法[18]，表2给出了当地图尺寸发生变化的时候，JPS与A*、Dijkstra、A*、BFS、DFS、Theta*的比较结果，可知：随着地图尺寸的增加，路径长度和搜索时间同步增加. 在同一地图尺寸下，各个算法所求得的路径长度相等或一致，但在搜索时间上，JPS明显快于其他算法，且随着地图尺寸的增加，JPS的时间效率优势更加显著.

表 2　JPS与其他典型寻路算法的效率比较

地图尺寸	路径长度						搜索时间/ms					
	A*	Dijkstra	BFS	DFS	Theta*	JPS	A*	Dijkstra	BFS	DFS	Theta*	JPS
10×10	32.49	32.49	32.49	33.31	32.31	32.49	1.00	1.00	1.00	0.00	1.00	0.00
25×25	275.11	275.11	275.11	275.11	274.93	275.11	5.00	5.00	5.00	3.00	7.00	2.00
50×50	1149.05	1149.05	1149.05	1149.88	1148.88	1149.05	30.00	33.00	27.00	25.00	37.00	8.00
100×100	4892.59	4892.59	4892.59	4932.36	4892.19	4892.59	317.00	309.00	290.00	290.00	241.00	33.00
128×128	8054.19	8054.19	8054.19	8105.55	8053.79	8054.19	735.00	731.00	726.00	705.00	453.00	53.00

实验3：为了使实验更有说服力，再在测试库中的基准地图上测试JPS的性能. 测试用的地图来源于NathanSturtevant免费提供的Benchmark sets中Starcraft（星际争霸）地图集，可从http://www.movingai.

(a)高度对称地图 I (b)中度对称地图 R (c)低度对称地图 E

图 7 环境对称性程度不同的3个基准地图

com/benchmarks获得. 测试用的三个地图分别是其中的Isolation.map(I)、RedCanyons.map(R)和Entanglement.map(E), 如图7所示.

A*算法和JPS算法在3个基准地图上被验证或评估, 变化的是环境的对称性. 实验评估了搜索时间、扩展的节点数和路径长度, 为了进行有说服力的算法验证, 算法重复执行1000次, 在搜索时间上给出了单次的平均搜索时间, 实验结果如表3所示.

两算法都能找到从初始节点至目标节点的最优路径, 且JPS保持了A*的最优性. 地图I的环境以高度对称性为特征, 该特征的地图有多条长度相等的对称路径, JPS在搜索时间上要快于A*, 在扩展节点数上也少于A*. 地图R的环境以中度对称性为特征, 对称路径在减少, 在这种情况下, 路径的长度增加了约20%, JPS在搜索时间上仍然快于A*, 且扩展的节点数少于A*, 但返回的路径长度要长于A*返回的路径. 地图E的环境以较低的对称性为特征, 识别这样的环境是非常复杂的, JPS在搜索时间上仍然快于A*, 扩展的节点数同样少于A*.

表 3 对称性程度不同的3个地图上A*与JPS重复执行1000次的性能概要

基准地图	路径长度/节点		平均搜索时间/ms		扩展的节点数		平均搜索时间比值(A*/JPS)/倍	扩展的节点数比值(A*/JPS)/倍
	A*	JPS	A*	JPS	A*	JPS		
低度对称地图E	11,905,340	11,905,340	447	41	138,480	3921	10.90	35.32
中度对称地图R	12,036,022	12,106,316	353	30	115,310	2322	11.77	49.66
高度对称地图I	9,987,250	998,7250	229	17	53,472	963	13.47	55.53

由表3可知, 在所求解到的路径长度相等或一致的情况下, 无论是在平均搜索时间还是在扩展的节点数上, JPS都明显优于A*, JPS可将A*提速一个数量级甚至更多. 同时, 观察表3"平均搜索时间比值"和"扩展的节点数比值"列数据可知, JPS的寻路效率取决于地图环境的对称性, 地图环境的对称程度越高, JPS较之于A*的优势越明显, 那是由于JPS消除了更多的对称路径"贡献"的, 并且JPS较之于A*在扩展的节点数上的优势程度比搜索时间上还要显著.

4 结 论

寻路问题在执行时间上仍有改进的空间, 在A*算法提出近50年后, 一个新的算法JPS能够大幅提高寻优的性能, JPS是一个能显著加速标准A*的算法, 在等价网格上实现了比A*高达一个数量级甚至更多的速度提升. 本文尝试对JPS进行了图解释, 清晰直观地呈现了该算法的原理, 避免了通过难懂的数学证明去理解算法的内涵. 仿真实验对JPS算法的性能进行了综合分析, 实验结果有效表明了JPS算法较之于其它典型寻路算法在搜索时间上的优越性, JPS扩展节点数与查看邻居数之间成反比关系, 地图环境的对称性越高越能体现JPS较之于A*的性能优势. 因此, 从快速寻路角度上看, JPS是各种地图环境下最好的算法, 但通常它发现的路径可能会长于A*算法. 因此, 该算法更适合需要快速寻路的情况.

本文的研究"意犹未尽", 未来的研究可以进一步丰富, 比如: 1) 考虑对加权网格或者多目标点搜索进行研究; 2) 探讨JPS联合Star算法家族各个算法的可能性, 比如以Basic Theta* and Phi*改进JPS, 允许

以任意角度搜索网格地图等.

参考文献：

[1] HaraborD, Grastien A. Online Graph Pruning for Pathfinding on Grid Maps[C]//Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence. San Francisco, California, USA: The AAAI Press, 2011:1114-1119.

[2] Patel A. Amit's A* Pages[EB/OL]. 1997[2015-08-26]. http://theory.stanford.edu/~amitp/GameProgramming/.

[3] KorfR E. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search[J]. ArtificialIntelligence, 1985,(27):97-109.

[4] Stentz A. Optimal and Efficient Path Planning forPartially-Known Environments[C].Proceedings IEEEInternational Conference on Robotics and Automation. SanDiego:[s.n.], 1994:3310-3314.

[5] Koenig S, Likhachey M,Furcy D. Lifelong Planning A*[J]. ArtificialIntelligence Journal, 2004, 155(1-2): 93-146.

[6] BoteaA, M¨ullerM, SchaefferJ. Near OptimalHierarchical Path-Finding[J]Journal of Game Development, 2004, 1(1):7–28

[7] Likhachev M,Ferguson D,Gordon G,*et al.* Anytime Dynamic A*: An Anytime, replanningAlgorithm[C]//Proceedings of the International Conference on Automated Planning and Scheduling(ICAPS). Menlo Park, CA, USA: AAAI, 2005: 262-271.

[8] Nash A, Daniel K, Koenig S, *et al.* Theta*: Any-Angle Path Planning on Grids[C]//Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence. Vancouver,Canada:. AAAI Press, 2007: 1177-1183.

[9] Harabor D, Botea A, Kilby P. Path Symmetries in Uniform-Cost Grid Maps[C].Proceedings of 9th Symposium on Abstraction Reformulation and Approximation (SARA). Barcelona, Spain, 2011.

[10] 邱磊. 利用跳点搜索算法加速A*寻路[J]. 兰州理工大学学报, 2015,41(3): 102-107.

[11] Podhraski T.How to Speed Up A* Pathfindingwith the Jump Point Search Algorithm[EB/OL].(2013-03-12)[2015-08-26]. http://gamedevelopment.tutsplus.com/tutorials/ how-to-speed-up-a-pathfinding-with-the-jump-point-search-algorithm–gamedev-5818.

[12] 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.

[13] Szabó A. Pathfinding: A*/Jump Point Search[EB/OL]. (2014-06-04) [2015-08-26].https://y4pp.wordpress.com/2014/06/04/pathfinding-a-jump-point-search/.

[14] Harabor D. Fast Pathfinding via Symmetry Breaking[EB/OL]. [2015-08-26]. http://aigamedev.com/open/tutorial/symmetry-in-pathfinding/.

[15] Witmer N. Jump Point Search Explained[EB/OL]. (2013-05-05)[2015-08-26]. http://zerowidth.com/2013/05/05/jump-point-search-explained.html.

[16] Tanner B.Jump Point Search Analysis[EB/OL]. 2013[2015-08-26]. http://www.cs.fsu.edu/~cop4601p/project/students/bryan-tanner/JumpPointSearch_tanner.pdf.

[17] Xu X. Pathfinding Visual[EB/OL]. 2013[2015-08-26]. http://qiao.github.io/PathFinding.js/visual/.

[18] Yonaba R. Fast,lightweight and easy-to-use pathfinding library for grid-based games[EB/OL]. (2012-05-26) [2015-08-26].https://github.com/Yonaba/Jumper.

责任编辑： 闫新云