运动规划作业 2: A* 与 JPS

杨逸初

March 26, 2022

1 算法流程

Algorithm 1: A* 算法

```
初始化 openSet;
初始化起点的 gscore 和 fscore;
将起点加入 openSet 中;
while openSet 不为空 do
    取出 openset 中 fscore 最小的节点 currentNode;
    if currentNode 是终点 then
        从 currentNode 回溯路径至起点;
        return 路径;
    else
        扩展 currentNode 周围的邻点放入 neighborset;
        for neighborNode in neighborset do
            if neighborNode 在 openSet 中 then
                计算从 currentNode 到 neighborNode 的 gscore;
                if 从 currentNode 到 neighborNode 的 cost 比原来更少 then
                    将 currentNode 设置为 neighborNode 的父节点;
                    更新 neighborNode 的 gscore 和 fscore;
            else if neighborNode 不在 closeSet 也不在 openSet 中 (新节点) then
                将 currentNode 设置为 neighborNode 的父节点;
                计算 neighborNode 的 fscore;
                将 neighborNode 加入 openSet 中;
            else
                跳过;
            end
        end
    end
end
```

2 运行结果

为了保证结果的一致性，这里将随机引擎的 seed 固定为 1234, 起点均为 (0,0,1), 选取四个不同距离的终点对 A* 算法的运行结果做分析, 此处启发式函数为欧式距离:

坐标	时间 (ms)	访问节点数	距离 (m)
(-1.0,-1.0,1.0)	0.257	539	1.414
(2.0,4.0,1.0)	7.843	15689	4.828
(5.0,-4.0,2.0)	19.305	38719	6.892
(-4.3,-4.8,0.0)	22.857	57264	9.712

3 对比不同启发式函数 (Manhattan, Euclidean, Diagonal Heuristic) 对 A* 运行效率的影响

	Manhattan		**Euclidian**		**Diagonal（3d）**	
坐标	时间 (ms)	访问节点数	时间 (ms)	访问节点数	时间 (ms)	访问节点数
(-1.0,-1.0,1.0)	**0.200**	**372**	0.257	539	0.332	499
(2.0,4.0,1.0)	**6.424**	**11821**	7.843	15689	9.072	14875
(5.0,-4.0,2.0)	**16.341**	**31181**	19.305	38719	20.228	37079
(-4.3,-4.8,0.0)	**19.881**	**53993**	22.857	57264	21.523	57016

运行时间仅做参考，每次运行都会不同. 此处使用的对角距离是 3d 对角距离：$dist = (\sqrt{3} - \sqrt{2}) * dmin + (\sqrt{2} - 1) * dmid + dmax$. 此处 Manhattan Heuristic 效率最高.

4 对比是否加入 Tie Breaker 对 A* 运行效率的影响

坐标	**Manhattan (tie breaker)** 访问节点数	**Manhattan** 访问节点数	**Euclidian (tie breaker)** 访问节点数	**Euclidian** 访问节点数	**Diagonal (tie breaker)** 访问节点数	**Diagonal** 访问节点数
(-1.0,-1.0,1.0)	**372**	372	539	539	499	608
(2.0,4.0,1.0)	**11792**	11821	15666	15689	14855	17347
(5.0,-4.0,2.0)	**31120**	31181	38652	38719	37018	40998
(-4.3,-4.8,0.0)	**53963**	53993	57252	57264	57005	57606

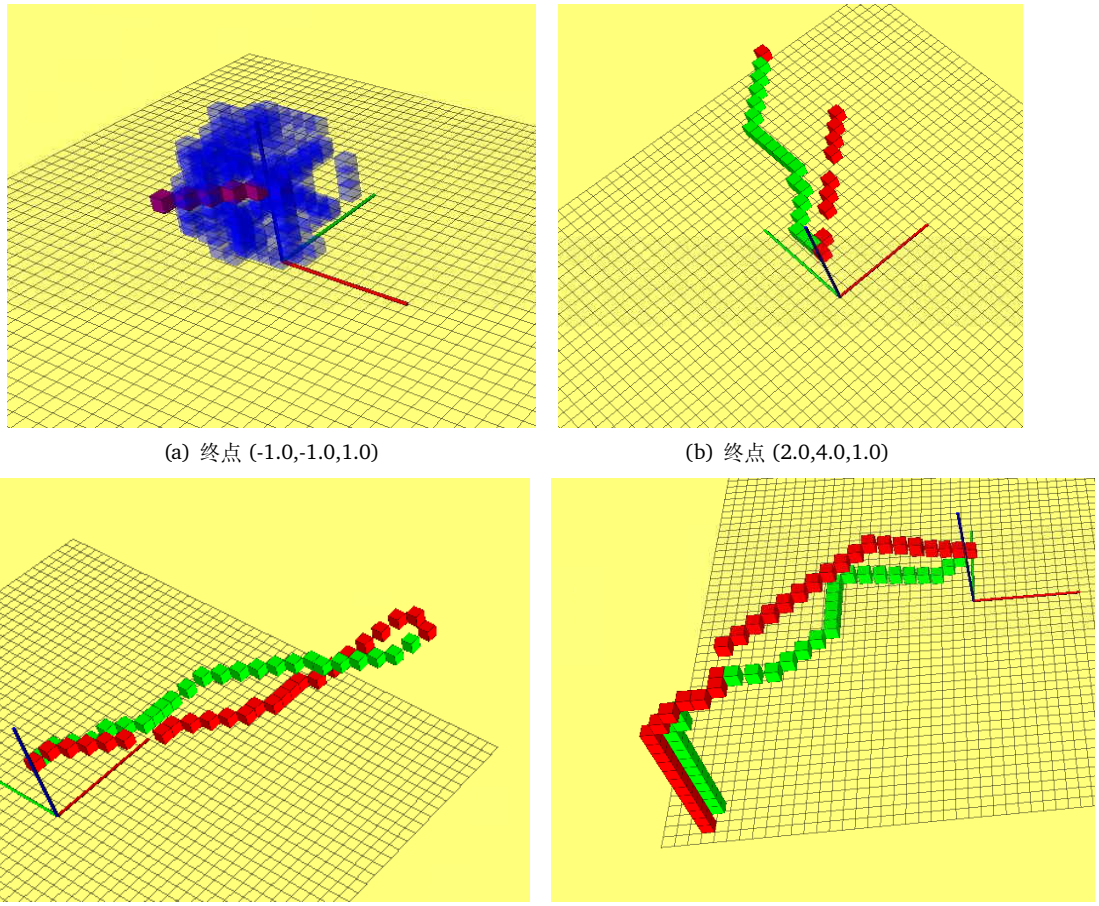
使用的 tie breaker 为当前点到终点连接向量与起点终点连接向量之间的叉乘的绝对值之和，即 $tie\_breaker = abs(m2 * n3 - m3 * n2) + abs(m1 * n3 - m3 * n1) + abs(m1 * n2 - m2 * n1)$. 表示路径更偏向起终点连线附近，但实际路径长度经实验后并无变化，加入 tie breaker 后算法性能略有提升，算法访问节点数略有下降. 下降在 diagonal heuristic 更加明显一些.

5 A* 和 JPS 算法效率的分析

	Manhattan		**Euclidian**		**Diagonal**		**路径长度 (m)**		**时间 (ms)**	
坐标	**A***	**JPS**	**A***	**JPS**	**A***	**JPS**	**A***	**JPS**	**A***	**JPS**
(-1.0,-1.0,1.0)	372	**124**	539	179	608	173	1.414	1.414	0.24	**0.18**
(2.0,4.0,1.0)	11821	12481	15689	12749	17347	**6089**	4.828	4.828	**5.39**	13.11
(5.0,-4.0,2.0)	31181	**13360**	38719	20671	40998	19571	**6.891**	7.038	**16.8**	35.7
(-4.3,-4.8,3.0)	53993	**21003**	57264	22221	57606	21875	**9.712**	9.868	**25.8**	37.3

可以看到 JPS 算法在大多数情况下通过跳点大大减少了需要探索的邻点，提高了搜索效率. 同时 JPS 得到的路径可能会劣于 A* 得到的最短路径. 本次实验中由于搜索的路径较短，实际上即使 JPS 算法减少了访问节点的数量，但由于花费了大量时间查询节点，因此 A* 算法所花费的时间要少于 JPS，但在更为复杂的环境和更远的终点情况下，A* 所需要访问的节点数将大大增加，消耗时间也会大大增加，而 JPS 则可以以较少的访问换取更快的计算时间. JPS 某些空旷的环境下效率可能不及 A* 算法，虽然 JPS 放入 openSet 中的点较少，但是要耗费大量时间来查询节点之间的关系. 在较复杂的环境中, JPS 算法比 A* 效率更高.(JPS 局限于 gridMap)

6 附录 1



(a) 终点 (-1.0,-1.0,1.0)

(b) 终点 (2.0,4.0,1.0)

(c) 终点 (5.0,-4.0,2.0)

(d) 终点 (-4.3,-4.8,0.0)

Figure 1: A* 与 JPS 生成路径对比，绿色为 A*，红色为 JPS，可以看到 JPS 生成的路径有明显的跳跃