首页 博客 学院 下载 论坛 图文课 问答 商城 活动 专题 招聘 ITeye GitChat APP VIP会员 续费8

原创

D*(Dynamic A*) 算法详细解析

目录

最初是Anthony Stentz发表在ICRA上，《Optimal and Efficient Path Planning for Partially-Known Environments》

记号

1）G：终点
2）X：表示当前节点，用$b(X)=Y$表示Y是X的下一个节点，称为后向指针（backpointer），通过后向指针，很容易获得
3）$c(X,Y)$：从X到Y的cost，对于不存在边的两个节点，$c(X,Y)$可定义为无穷大。
4）同A^*一样，D^*也用$OPEN$表和$CLOSED$表来保存待访问和已经访问的节点。对每个节点X，用$t(X)$来表示X的状态
$OPEN$表示当前处于$OPEN$ list，$CLOSED$表示已从$OPEN$ list剔除。
5）同A^*一样，D^*也有一个对目标的估计函数，$h(G,X)$，不同的是，D^*还多了一个key函数，对于在$OPEN$ list中的每
equal to the minimum of $h(G,X)$ before modification and all valuses assumed by $h(G,X)$ since X was placed on the OP
扑结构的变化而变化，key始终是h中最小的那一个。key函数将$OPEN$中的节点X分为两类，一类记为$RAISE$，如果$k(G,$
$LOWER$，如果$k(G,X)=h(G,X)$。也就是说，$k(G,X)\le h(G,X)$，小于的情况如前方出现障碍物导致某些边
于障碍物移动导致某些边cost减小。
6）k_{min}和k_{old}，前者表示当前$OPEN$中key的最小值，后者表示上一次的最小值。
7）如果一系列节点$X_1, X_2, ..., X_N$满足$b(X_{i+1})=X_i$，则称为一个序列（sequence）。一个序列表示了一条从X_N
的是一条从起点到终点的路径中，但是在动态路径规划中，起点是不断变化的，严格来讲应该是找寻一条从当前节点到
当前节点更为方便，故这里X_1反而是最靠近终点的节点。称一个序列为单调（monotonic）的如果满足$t(X_i)=CLC$
$t(X_i)=OPEN\ and\ k(G,X_i)<h(G,X_{i+1})$。杂乱无序的序列是没有意义的，单调的序列其实就是我们要找的一
8）对X_i, X_j，如果$i<j$则称X_i是X_j的祖先（ancestor），反之则称为后代（descendant）。
9）如果记号涉及两个节点且其中一个为终点，略去终点，如$f(X)=f(G,X)$

算法描述

D^*主要包括两个部分，$PROCESS-STATE$和$MODIFY-COST$，前者用来计算终点到当前节点的最优cost，
1）将所有节点的tag设置为NEW，$h(G)$设为0，将G放置在$OPEN$中。
2）重复调用$PROCESS-STATE$直到当前节点X从$OPEN$中移除，即$t(X)=CLOSED$，说明此时已经找
3）根据2）中获得的路径，从节点X往后移动，直到达到终点或者检测到cost发送变化。
4）当cost发生变化时，调用第二个函数$MODIFY-COST$，并将因为障碍物而cost受到影响的节点重新放入OP
点。通过调用$PROCESS-STATE$直到$k_{min}\ge h(Y)$，此时cost的变化已经传播到Y，因此可以找到一条新的从
论文中的伪代码如下：

Function: PROCESS-STATE ()

```
X = MIN-STATE( )
if X = NULL then return -1
k_old = GET-KMIN( ); DELETE(X)
if k_old < h(X) then
  for each neighbor Y of X:
    if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
      b(X) = Y; h(X) = h(Y) + c(Y, X)
if k_old = h(X) then
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
      (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
else
  for each neighbor Y of X:
    if t(Y) = NEW or
      (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
      b(Y) = X; INSERT(Y, h(X) + c(X, Y))
    else
      if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
        INSERT(X, h(X))
      else
        if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
          t(Y) = CLOSED and h(Y) > k_old then
          INSERT(Y, h(Y))
return GET-KMIN( )
```


这里$MIN-STATE$和$GET-MIN$的区别是前者返回的是具有最小k值的节点，后者返回的是k_{min}。而$INSERT(X$
情况：
a）$t(X)=NEW, k(X)=h_{new}$
b）$t(X)=OPEN, k(X)=min(k(X),h_{new})$
c）$t(X)=CLOSED, k(X)=min(h(X),h_{new}), h(X)=h_{new}\ and\ t(X)=OPEN$
最后一个条件就是针对已规划路径cost发生变化的状态。

Function: MODIFY-COST (X, Y, cval)

```
c(X, Y) = cval
if t(X) = CLOSED then INSERT(X, h(X))
return GET-KMIN( )
```

梳理一下上述流程
1）在静态条件下，利用Dijkstra或A*算法找到一条最优路径，同时利用后向指针确定每个节点的下一节点。
2）机器人从起点出发，沿路径移动，考虑最简单的情况，假设机器人的传感器范围为1，也就是说，只要下一个节点没有障碍
3）假设下一节点出现障碍物，怎么进行$MODIFY-COST$呢？参考伪代码，当某个节点出现障碍物了，可以认为h_{new}
的值，而h已经变成无穷了，并且，X被重新放入$OPEN$中。
4）放入$OPEN$当然是要重新规划，这里就涉及到一个关键问题了

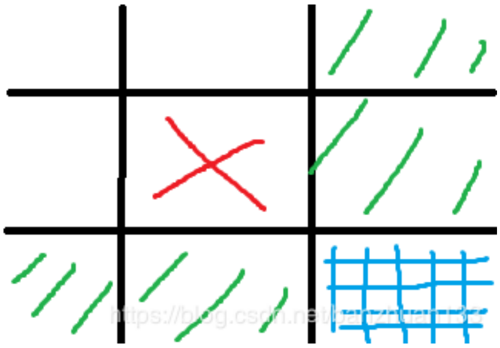
已经有h函数了，为什么还要定义key函数？

其实这里的h函数应该和A^*中的f函数是等价的，虽然只有到终点的开销，但是因为D^*是终点开始的，起点就是当前节点X，
$g(X)+h(X)=g(X)$。在静态情况下，直接用$h(X)$作为排序，是可以找到一条全局最优路径的（实际就是Dijkstra）。在

了不可达，那么相应的$h(X)=\infty$，将X状态添加进$OPEN$ list，那么X节点就是排在$OPEN$ list的最后。也就是说，此时态的节点开始，一直到扩张全图都没有不可达节点之后，才会访问X，这显然是不合理的。我们的目的就是要减少搜索空间，表示这里曾经有一条捷径，那么我优先在这附近搜索，进一步的，当障碍物消失，也能第一时间恢复到最短路径。

那么接下来就是最关键的步骤，怎么通过局部调整避开障碍物找到一条合理的路径（实际是贪心的最优）。

5）此时$PROCESS-STATE$进入的是$L4-L7$，注意，这里我们最简单的想法应该是找到除了Y之外最短的路径，也$C(Y,X)$，但是作者这里加了一个条件，$h(Y)\leq k_{old}$，初衷很好理解，我可以绕障碍物，但是不能绕太远，还是要往终点估计采用的是曼哈顿距离，邻域考虑最简单的八邻域，那么如下图所示，红色是当前节点，蓝色是初始规划的下一节点，此时节点用绿色表示，考虑极端情况，所有绿色格子也被障碍物占据，此时算法会陷入死循环（去掉该条件则不会）。这里暂时不们可以默认在已知环境中突然出现大量的障碍物概率很小）。



6）在5）中我们找到一个当前最优的下一节点，注意，因为这里是贪心策略，所以并没有考虑到下一节点是否是可行的（可能的）。这样做的好处是尽可能地利用了之前已经探索的区域，减少了搜索空间，坏处是可能会绕远路。
7）上述过程的终止条件，$k_{min}\geq h(Y)$，也就是说，当前所有处于$OPEN$状态的节点，都不可能再降低cost了，那么自然成了一次路径的修正。
8）这里考虑的是cost增加的情况，实际cost减少的情况也是类似的。

参考

Optimal and Efficient Path Planning for Partially-Known Environments
https://blog.csdn.net/lqzdreamer/article/details/85055569#_437