网络出版时间：2018-11-13 09:09:04
网络出版地址：http://kns.cnki.net/kcms/detail/21.1124.TP.20181113.0907.013.html

DOI: 10.13195/j.kzyjc.2018.0583

基于路径优化 D* Lite 算法的移动机器人路径规划

黄 鲁[1†], 周非同[1]

(1. 中国科学技术大学 电子科学与技术系，安徽 合肥 230027)

摘 要: 采用 D* Lite 算法规划出的路径并不平滑, 且预规路径与障碍物均十分接近. 除此之外, 在动态环境下时, 由 D* Lite 算法重规划得到的路径也离障碍物距离很近, 十分容易发生碰撞. 针对此问题, 本文引入懒惰视线算法与距离变换相结合的方法改进 D* Lite 算法. 首先对地图进行距离变换, 并引入距离值的启发式代价, 使得距离障碍物较远的节点优先被选择. 然后在扩展节点时引入视线算法, 增加本地父亲节点和远程父亲节点的概念, 使得路径不局限于八邻域扩展, 从而进化为任意角度路径规划算法. 最后, 在遇到未知障碍物时进行局部距离变换, 结合启发距离值信息进行重规划, 使得重规划得到的路径远离突现的障碍物. 仿真实验表明, 在不同环境下规划所得到的路径均十分平滑与安全.

关键词: D* Lite；路径规划；移动机器人；路径优化；视线算法；距离变换

中图分类号: TP18 **文献标志码:** A

Path Planning of Moving Robot Based on Path Optimization of D* Lite Algorithm

HUANG Lu[1†], *ZHOU Fei-tong*[1]

(1. Department of Electronic Science and Technology，University of Science and Technology of China，Anhui 230027，China)

Abstract: The path planned by D* Lite algorithm was not smooth, and the preplanned path was very close to the known obstaclse. Besides, the replanned path was very close to unknown dynamic obstacles so that the collision can happen very easily. To deal with the problem, the thoughts of Lazy Theta* algorithm and distance transform were combined with D* lite algorithm. Firstly, the map was processed by distance transform algorithm to get the heuristic distance value, which made the nodes that were far away from the obstacles preferred to be selected. Secondly, line of sight algorithm was used while expanding nodes. The concepts of local parents and remote parents were added so that the path was more than eight neighbours. Finally, when unknown obstacles were discovered, local distance transform algorithm was used to speed up the replan process and made the replanned path safer. The experimental results show that the paths planned in different environments are all smooth and safe.

Keywords: D* Lite; path planning; robot; path optimitization;line of sight algorithm; distance transform

0 引 言

关于移动机器人的研究已经取得了许多成果, 移动机器人应用也已经越来越多的渗透到生活的各个方面. 在移动机器人相关领域中, 路径规划是最重要的需求之一.

移动机器人路径规划是在给定环境中从起始点到目标点找到一条在时间、距离或是空间上最优的无碰路径[1]. 一般路径规划的第一步是对环境进行建模, 一种经典的建模方法是占据栅格地图建模[2]. 占据栅格地图以其简明易用性受到了广泛的使用, 也因此衍生出了一系列占据栅格地图为环境的路径规划算法, 例如 A*[3], D*[4] 等. 其中 A* 算法是为了解决静态环境下的寻路问题, 而 D* 算法则实现了不确定环境下的路径规划. Koeing S 和 Likhachev M 在 2004 年提出了 LPA*[5] 算法, 该算法通过复用之前搜索的信息, 实现了在未知环境下遇见障碍物时的快速重规划. LPA* 算法的局限性在于它针对的是定起点定目标点的路径规划问题, 因此，D* Lite[6] 算法被

收稿日期: 2017-xx-xx；**修回日期:** 2017-xx-xx.

责任编委: 编委 1

作者简介: 黄鲁 (1961–), 男, 副教授, 硕士, 从事电路与系统，机器人系统等研究；周非同 (1994–), 男, 硕士, 从事机器人软件设计、路径规划算法的研究.

†通讯作者. E-mail: luhuang@ustc.edu.cn

提出用以实现变起点、定目标点的路径规划.

在栅格环境下进行路径规划所得到的路径由于八邻域的限制, 与实际中的最短路径差距较大, 因此引出了任意角度规划 (any-angle path planning)[7] 的分支, 在静态已知环境下有 Theta*[8]、Lazy Theta*[9] 等方法, 在动态未知环境中下有 Field D*[10] 等. 由 Anthony Stentz 提出的 Field D* 算法采用路径插值的方式平滑路径, 但存在计算量较大, 消耗资源较多的问题. 另一方面，由 Alex Nash 提出的 Theta* 在 A* 算法的基础上引入视线算法[11], 极大的优化了 A* 算法得到的路径, 并进一步提出 Lazy Theta* 算法, 在 Theta* 的基础上减少计算消耗时间, 由此得到计算较快且路径长度较短的路径. 除了上述 A* 系列算法之外, 还有如人工智能类、随机采样类等路径规划算法. 在国内, 智能优化类路径规划算法一直处于研究的前端. 游晓明[12] 等人定义了一种新的动态搜索诱导算子来改进传统的蚁群路径规划算法, 加快了算法的收敛速度, 提高了优化解的质量. 王雷[13] 等人改进了基本的遗传算法, 提出一种生成初始种群的方法和精英策略并设计出自适应变异概率从而提高了算法的求解质量. 然而智能优化算法一般只用于求解路径规划某个过程, 一般需要与其他路径规划方法结合使用[14], 且容易陷入局部最优解.

在国外, 当前的路径规划问题集中于高维环境中的运动规划问题, 例如机械臂的移动问题. 由此引出了基于采样的一系列算法, 例如快速扩展随机树[15](RRT), 概率道路图[16](PRM) 等方法.Sertac Karaman[17] 等人指出,RRT 和 PRM 虽然在概率上完备, 但路径并不是最优, 并由此提出了路径渐进最优的 RRT* 和 PRM* 算法.

然而, 衡量路径规划质量的指标除了路径长度之外, 还有离障碍物的距离[18]. 由上述算法规划出的路径均距离障碍物较近, 在实际环境下容易产生碰撞. 而在动态未知环境下运行的算法例如 D* lite 算法, 由于未知障碍物可能突现, 其规划的路径与静态环境下所规划出的路径相比更为危险.

本文通过对 D* Lite 算法进行分析, 指出 D* Lite 算法中存在路径不平滑和易于误碰障碍物的问题, 对此引入 Lazy Theta* 算法中的视线算法和懒惰更新思想, 并且加入图像处理算法距离变换[19], 使得规划出的路径具有安全、平滑且较短的特点.

1 D* Lite 算法

D* Lite 算法是由 Koeing S 和 Likhachev M 提出的栅格模型下针对动态环境的路径规划算法. D* Lite 算法将 LPA* 算法的思想运用于动态环境下, 使得移动机器人在未知环境下可以快速重规划.

D* Lite 算法采用从目标节点到起始节点的搜索方式, 使得 D* Lite 算法可以运用于起始点改变的情况, 并且引入了 LPA* 算法中 $rhs(s)$ 的定义:

$$rhs(s)=\begin{cases}0 & \text{if } s=s_{goal}\\ \min\limits_{s'\in Succ(s)} c(s,s')+g(s') & \text{otherwise}\end{cases} \tag{1}$$

其中 $Succ(s)$ 表示所有节点 s 的后继节点,$g(s')$ 表示节点 s' 到目标节点的距离. D* Lite 对每个栅格维护 $g(s)$ 与 $rhs(s)$ 两个变量, 当两者相等时, 表示该节点处于一致状态 (Consistent), 否则处于不一致状态 (Inconsistent). 当处于不一致状态时表示该节点的路径信息是不准确的, 只有处于一致状态下的节点是已经扩展完毕、可以通行的.

D* Lite 算法的优势在于, 当环境中出现新的未知障碍物时, 可以快速更新该障碍物周边节点的信息, 并将由此而导致不连续的节点重新压入 open 列表中进行快速的重规划. 但是, 由于是在栅格环境下,D* Lite 算法所规划出的路径是八邻域的, 因此得到路径并不平滑. 而另一方面, 在现实未知环境下, 极有可能在距当前位置较近处出现障碍物, 此时 D* Lite 规划出的路径过于靠近障碍物.

针对上述两个问题, 本文提出一种改进的 D* Lite 算法, 规划出平滑及安全的路径.

2 D* Lite 算法改进

2.1 相关定义

在介绍改进算法之前, 先给出算法中的一些相关定义: 记当前节点为 s, 起始节点为 s_{start}, 目标节点为 s_{goal}.

定义 1 对栅格地图进行距离变换, 记所使用的距离变换范围为 r, 即在障碍物周边 r 之内可得 1 到 r 之间的数值. 设当前节点 s 在范围 r 之内, 记 r_s 为节点 s 得到的距离变换值,$dist(s)$ 为节点 s 的启发距离值, 则：

$$dist(s)=\begin{cases}0 & \text{otherwise}\\ r-r_s+1 & \text{if}(r_s\le r)\end{cases} \tag{2}$$

定义 2 距离变换 DistanceTransform(S) 表示对地图 S 进行距离变换并得到地图中每个节点 s 的启发距离值 $dist(s)$.

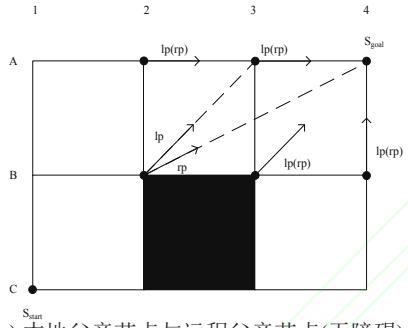
定义 3 距离变换 DistanceTransformLocal(s) 表示对节点 s 周围的 r 范围内的地图进行距离变换，进行距离变换的节点取其较大启发距离作为最终启发距离. 其中 s 为探测到的障碍物节点.

定义 4　当前节点 s 到目标节点的最短路径加启发距离值（下面简称为启发路径）为 $g(s)$, 由 s 后继节点得到 s 到目标节点的启发路径记 $rhs(s)$，则：

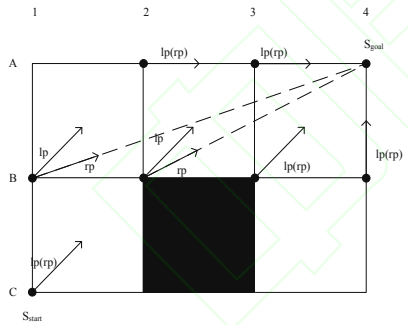
$$rhs(s)=\begin{cases}0 & \text{if}(s=s_{goal})\\ \min\limits_{s'\in Succ(s)}(c(s,s')+g(s') & \text{otherwise}\\ +dist(s'))\end{cases} \tag{3}$$

其中 $c(s,s')$ 表示从节点 s 到 s' 的代价, 定义为两节点间的欧氏距离. 当 $rhs(s)=g(s)$ 时, 表示节点处于一致状态, 是可通行的, 否则表示节点处于不一致状态, 需要进一步更新.

定义 5　定义 $lp(s)$ 为 s 的本地父亲节点,$rp(s)$ 为节点 s 的远程父亲节点. 其中,$lp(s)$ 通过计算后继节点的最小 rhs 得到,$rp(s)$ 则表示通过对节点 s 以及其初始 $rp(s)$ 进行视线算法而得到.$rp(s)$ 有多种更新方式, 将在下文详述。



(a) 本地父亲节点与远程父亲节点(无障碍)



(b) 本地父亲节点与远程父亲节点(有障碍)

图 1　本地父亲节点与远程父亲节点示意图

$lp(s)$ 与 $rp(s)$ 的示意图见图 1. 图 1(a) 在扩展 A4 周围节点时, 由于均在 A4 邻域内, 故 $lp(s)$ 与 $rp(s)$ 均相同. 对于 A3 节点的邻域节点 B2, 其与 A4 节点可视, 故 B2 节点的远程父亲节点即为 A4 节点, 而其本地父亲节点为 A3 节点.

对于节点之间有障碍的情况, 如图 1(b) 中的 C1 节点与 A4 节点, 因其不可视, 将远程父亲节点设置为本地父亲节点.

定义 6　定义 $h(s,s_{start})$ 为节点 s 与起始节点 s_{start} 间的估计启发代价, 设为两节点间的欧氏距离. k_m 为 $h(s_{last},s_{current})$,s_{last} 为上一次检测到地图变化的节点,$s_{current}$ 为当前检测到变化的节点,k_m 则为移动距离的叠加.

定义 7　保存不连续节点的优先队列定义为 U, 在 U 中的节点按其 key 值从小到大排列, 其中 key 是一个二维向量, 定义为:

$$\begin{aligned}key(s)=&[min(g(s),rhs(s))+h(s_{start},s)+k_m;\\ &min(g(s),rhs(s))]\end{aligned} \tag{4}$$

2.2　距离变换与视线算法

2.2.1　距离变换

距离变换是图像处理中的一种常用算法, 其主要思想是通过标识空间点（目标点与背景点）距离的过程, 将二值图转化为灰度图.

在机器人地图中, 一般设置有障碍物区域像素值为 0, 自由空间区域像素值为 255. 将有障碍物区域视为目标点, 自由区域视为背景点. 对于所有背景点, 标识其与最近的目标点的距离 (本文采用欧式距离), 则得到距离变换图. 对于地图中的所有自由空间区域节点, 即得到了该节点离最近的障碍物节点的距离值.

本文使用 chamfer 距离变换[19] 方法完成地图的距离变换操作. 其基本步骤为:

Step 1: 初始化地图的所有自由区域节点距离值为无穷大, 障碍物节点距离值为 0.

Step 2: 从左到右, 从上到下遍历地图的所有节点, 计算节点的左上方四个节点中的最小距离值作为中间节点的距离值.

Step 3: 从底向上、从右至左遍历地图的所有节点, 计算节点的右下方四个节点与 Step 2 所得的最小距离值作为中间节点的距离值.

2.2.2　视线算法

视线算法常被用于游戏领域, 用于判断两点之间是否有障碍物存在. 对于栅格矩阵环境来说, 视线算法和画线算法[20] 十分相似. 画线算法需要决定在栅格矩阵上画直线时需要填充的栅格位置, 而对于视线算法, 只要两点之间画直线填充的栅格中没有障碍物即可. 在本文中, 填充栅格的启发距离值还需要为 0, 表明栅格周围没有障碍物.

设两个栅格的坐标分别为 (x0, y0) 与 (x1, y1),dx 与 dy 为两者横纵坐标之差,sx 与 sy 表示 x 轴与 y 轴前进的步长, 表示当前较短轴与长轴的差距.

假设 dx<dy, 则视线算法的基本流程见图 2, 其中 Bad(x0, y0) 表示 (x0, y0) 处的栅格存在障碍物或是启发距离值不为 0.

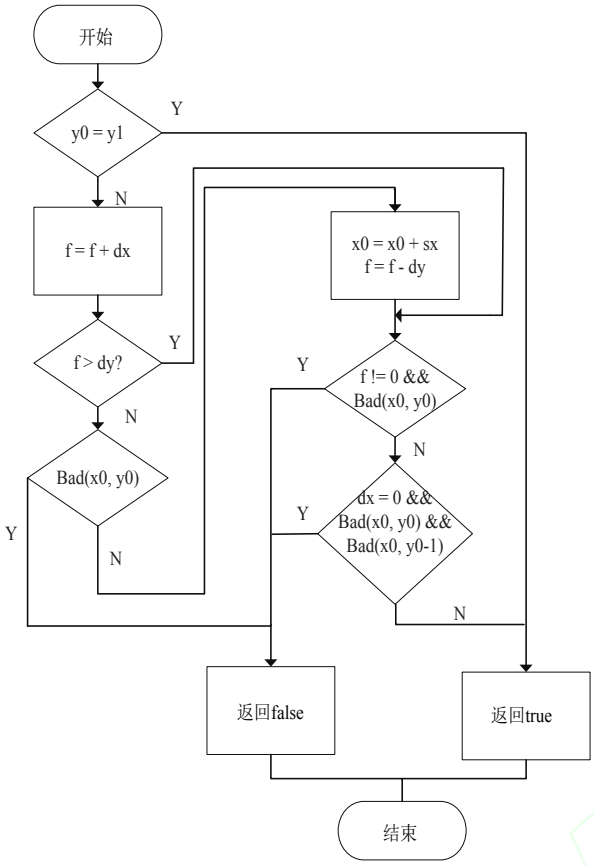


图 2 视线算法流程

2.3 改进 D* Lite 算法流程

改进 D* Lite 算法主流程见 main function 伪代码, 其中所使用到的函数见 helper function 伪代码.

Algorithm 1 main function

function main
  $s_{last} = s_{start}$
  Initialize()
  ComputeShortestPath()
  while $s_{start} \neq s_{goal}$ **do**
    $s_{start} \leftarrow rp(s_{start})$
    Move to s_{start}
    Scan graph for changed edge costs
    if any edge costs changed
    $k_m = k_m + h(s_{last}, s_{start})$
    $s_{last} = s_{start}$
    for all directed edges u, v with changed edge costs **do**
      Update edge cost $c(u, v)$
      DistanceTransformLocal(v)
      UpdateVertex(u)
    end for
    ComputeShortestPath()
  end while
end function

在算法初始化时, 对地图进行距离变换, 并根据定义 1 得到每个节点的启发距离值, 节点 s 距离地图中的障碍物越近, 则其启发距离值 $dist(s)$ 也将越大. 将启发距离值作为计算启发代价 $rhs(s)$ 的一部分, 由于每次都选取 $rhs(s)$ 较小的节点规划路径, 因此在预规划时路径与障碍物会保持一定距离.

Algorithm 2 helper function

function CalculateKey(s)
  return $[min(g(s), rhs(s)) + h(s_{start}, s) + k_m; min(g(s), rhs(s))]$
end function

function Initialize
  $U \leftarrow \phi$
  $k_m \leftarrow 0$
  for $s \in S$ **do**
    $rhs(s) \leftarrow \infty$
    $g(s) \leftarrow \infty$
  end for
  DistanceTransform(S)
  $rhs(s_{goal}) \leftarrow 0$
  $lp(s_{goal}) \leftarrow s_{goal}$
  $rp(s_{goal}) \leftarrow s_{goal}$
  U.insert(s_{goal}, CalculateKey(s_{goal}))
end function

function UpdateVertex(u)
  if $u \neq s_{goal}$ **then**
    $rhs(u) \leftarrow \min_{s' \in Succ(s)} c(u, s') + g(s) + dist(s')$
    $lp(u) \leftarrow s'$ $which\ satisfy\ min_{s' \in Succ(s)} c(u, s') + g(s) + dist(s')$
  end if
  if $rp(u) == NULL$ **then**
    $rp(u) \leftarrow lp(u)$
  end if
  if $u \in$U **then**
    U.Remove(u)
  end if
  if $g(u) \neq rhs(u)$ **then**
    U.insert(u, CalculateKey(u))
  end if
end function

function SetVertex(u)
  if NOT LineOfSight($rp(u), u$) **then**
    $rp(u) \leftarrow lp(u)$
  end if
end function

function ComputeShortestPath
  while U.TopKey() < CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$ **do**
    $k_{old} \leftarrow$ U.TopKey()
    $u =$U.pop()
    SetVertex(u)
    if $k_{old} <$ CalculateKey(u) **then**
      U.insert(u, CalculateKey(u))
    else if $g(u) > rhs(u)$ **then**
      $g(u) \leftarrow rhs(u)$
      for $s \in Pred(u)$ **do**
        UpdateVertex(s)
        if $rp(u)\ AND\ s \in$ U **then**
          if $c(rp(u), s) + rhs(rp(u)) < rhs(s)$ **then**
            $rp(s) \leftarrow rp(u)$

```
                end if
            end if
        end for
    else
        g(u) ← ∞
        for s ∈ Pred(u)∪u do UpdateVertex(s)
        end for
    end if
  end while
end function
```

改进 D* Lite 算法用视线算法进行可视检测, 其作用核心为本地父节点 $lp(s)$ 以及远程父节点 $rp(s)$ 的更新. 其中 $lp(s)$ 在计算每个节点的启发代价 $rhs(s)$ 时更新, 通过最小启发代价可到达的下一节点即为 $lp(s)$.

$$lp(s) = s^{'} \text{ which satisfy } \min_{s^{'} \in Succ(s)} (c(s, s^{'}) + g(s^{'}) + dist(s^{'})) \tag{5}$$

远程父亲节点 $rp(s)$ 是节点 s 的最终可视节点, 由视线算法更新所得. 在引入视线算法时, 采用懒惰更新的方式, 即在将当前节点设置为一致状态时, 假设周围节点与当前节点的父节点均为可视状态, 并计算是否在可视状态下通过当前节点的远程父亲到达此节点会更近, 若是, 则更新此节点的远程父亲节点. 在从优先队列 U 中弹出节点时, 用视线算法判断当前节点与其远程父亲节点是否可视, 若不可视, 则将节点 s 的远程父亲节点置为本地父亲节点.

由于每个节点的初始远程父亲节点均为空, 因而当更新本地父亲节点后, 若远程父亲节点为空, 则将远程父亲节点设置为本地父亲节点.

综上可得, 远程父亲节点的更新方式为:

$$rp(s) = \begin{cases} rp(s^{'}) & \text{if}(rhs(s) > c(rp(s^{'}), s) + rhs(rp(s^{'})) \\ lp(s) & \text{if}(\text{NOT LineOfSight}(rp(s), s)) \\ lp(s) & \text{if}(rp(s) = \text{NULL}) \end{cases} \tag{6}$$

在预规划完成后, 需要从起始节点向目标节点行进, 此时沿着远程父亲节点即可到达目标节点.

若途中遇见未知障碍物, 需要进行重规划. 此时根据定义 3 在障碍物节点周围进行局部距离变换并更新启发距离值, 然后再进行重规划, 路径将会远离未知障碍物.

3　实验结果及分析

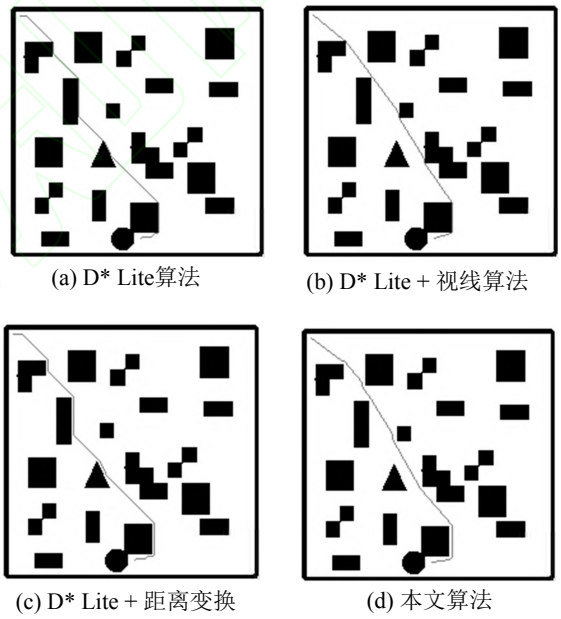
为验证算法的有效性, 以 Linux Ubuntu 16.04 为平台,C++ 语言为编程环境进行仿真. 实验的硬件平台为:Intel Core i5 7500 处理器, 主频 3.4GHz,8GB 内存.

在相同硬件平台的条件下, 对本文算法与原始 D* Lite 算法进行对比实验. 对于预规划, 给出了四种算法规划出的预规划路径及路径长度 (四种算法分别为 D* Lite、D* Lite 结合视线算法、D* Lite 结合距离变换以及本文算法, 以下分别简称为 D* Lite、Los D* Lite、Dist D* Lite 与 DLD* Lite). 对于重规划, 由于视线算法和距离变换的作用在本文算法上可以体现, 因此给出原始 D* Lite 与本文算法在出现未知障碍物情况下的重规划路径.

同时, 为了进一步说明本文算法的作用, 选取智能优化算法遗传算法以及基于随机采样的路径规划算法 RRT 作为比较对象, 在定起点, 定终点, 随机生成障碍物的情况下进行对比实验.

3.1　预规划路径比较

在 200×200 栅格地图上分别使用原始 D* Lite 算法、只引入懒惰视线算法的 D* Lite、只引入距离变换的 D* Lite 以及本文算法进行仿真, 得到如图 3 所示的预规划路径.



(a) D* Lite算法　(b) D* Lite + 视线算法

(c) D* Lite + 距离变换　(d) 本文算法

图 3　路径预规划结果

图 3(a) 为原始 D* Lite 算法规划所得路径, 图 3(b) 为 D* Lite 与懒惰视线算法的结合, 规划出的路径较为平滑, 但在边角处距离障碍物几乎为 0. 图 3(c) 为 D* Lite 算法与距离变换的结合, 规划出的路径离障碍物有 1 到 2 个栅格的距离, 但继承了 D* Lite 算法路径不平滑的缺点. 图 3(d) 为本文算法所规划出的路径, 路径平滑程度较高, 且由表 1 所示, 路径节点个数下降了 77%, 与障碍物也保持有 1 到 2 个栅格的距离.

表 1 为预规划路径的节点个数与总距离统计, 可以看出 DLD* Lite(本文算法) 综合了 Los D* Lite 与 Dist D* Lite 的优点. 与原始 D* Lites 算法相比, 节点

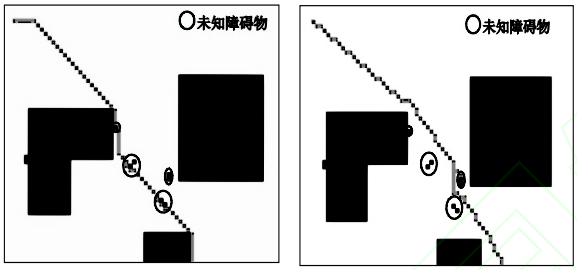
个数下降了 77%, 规划出的路径长度减少了 2%. 虽然路径长度下降不多, 但安全性大大增强. 由于引入了视线算法以及距离变换,DLD* Lite 算法所消耗的时间是最长的, 大约增加了 5%.

表 1 路径预规划相关数据

算法	路径节点数	路径长度	时间 (ms)
D* Lite	192	236.149	556
Los D* Lite	39	225.496	577
Dist D* Lite	196	239.321	563
DLD* Lite(本文算法)	44	231.022	585

3.2 重规划路径比较

在上图环境下, 当路径进行到中途时在路径前方产生未知障碍物. 为了更清晰的看清重规划路径的变化, 只截取在产生的未知障碍物附近得到的路径. 由于预规划路径不同, 因此遇见未知障碍物的位置也不相同, 但不影响路径的重规划结果分析.



(a) D* Lite算法 (b) 本文算法

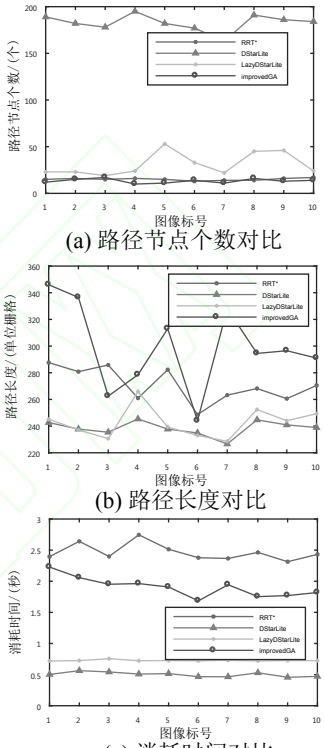
图 4 路径重规划结果

图 4(a) 为原始 D* Lite 的重规划结果, 重规划之后的路径离出现的未知障碍物较为接近. 图 4(b) 为本文算法进行路径重规划的结果, 在发现未知障碍物后, 规划出的路径距离障碍物有 1 到 2 个栅格的距离, 因此会先退出到安全距离后再进行绕过. 由于距离变换, 路径倾向于不存在障碍物的外侧, 而不会进入狭窄的缝隙.

3.3 对比实验

在 200×200 的栅格地图上, 指定起点为 (15, 15), 终点为 (175, 175), 在地图上随机填充障碍物, 生成 10 幅不同的图像, 并且分别用 D* Lite、文献 [22] 所提出的改进遗传算法、RRT* 算法以及本文算法进行路径规划, 记录规划之后的路径节点个数, 路径节点长度以及规划时间进行对比. 其规划结果见图 5.(其中, 实心圆图注为 RRT* 算法, 三角形图注为原始 D* Lite 算法, 菱形图注为本文算法, 空心圆图注为文献 [22] 提出的改进遗传算法).

在路径节点数上,RRT* 算法与文献 [22] 中算法较为接近, 均处于领先地位, 分别约有 15.1 个节点和 13.3 个节点. 本文算法略逊色于 RRT* 算法, 平均约有 31.2 个节点. 而原始的 D* Lite 算法平均约 182.7 个节点, 对于机器人路径规划之后的行动十分不利. 有上述现象的原因是 D* Lite 算法规划出的路径在八邻域上是最优的, 即使本文算法用了视线算法使得一些节点可以互相连接, 在障碍物拐角处的节点仍然会产生截断. 并且, 由于在障碍物节点周围节点其远程父亲节点均是本地父亲节点, 也会导致节点个数的增加.



(a) 路径节点个数对比

(b) 路径长度对比

(c) 消耗时间对比

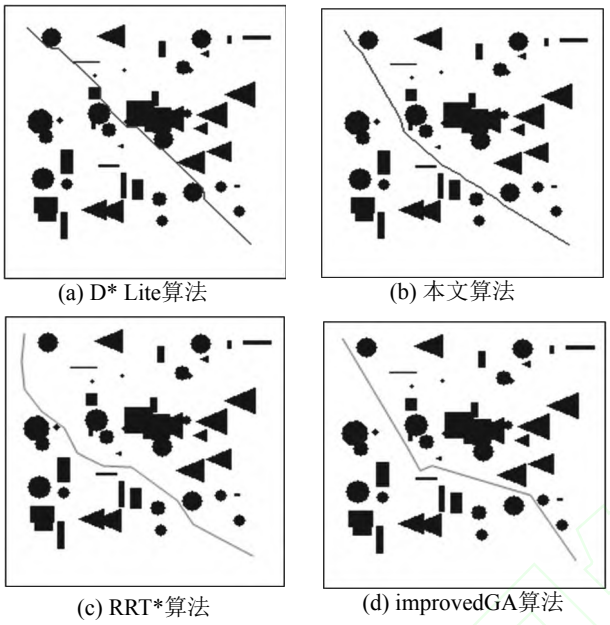
图 5 四种算法数据对比

由图 5(b) 可知, 在路径长度上, 本文算法与 D* Lite 算法处于优势. 因为 D* Lite 算法本身就是栅格分辨率以及八邻域下的最优解, 而本文算法在此基础上进行改进, 会由于平滑性和安全性对长度有一定的牺牲, 但改变并不大, 其得到的路径长度与原始 D* Lite 算法基本相同.

对于另外两种算法,RRT* 算法规划路径对于 RRT 算法作出了改进, 使得其路径会是渐进最优的. 比起 RRT,RRT* 算法的稳定性好了很多, 但由于其路径点的获取本质上仍是随机的, 因此距离本文算法还有差距. 另外, 当处于迷宫环境下等狭窄的通道时,RRT* 算法与 RRT 算法一样会消耗较多时间. 文献 [22] 提出的改进的遗传算法在路径长度上表现最差, 且由于变异算子的差异性, 其规划出的路径长度并不稳定.

由图 5(c) 可知, 本文算法所消耗时间略高于 D* Lite 算法, 分别为平均 727ms 以及平均 502ms. 而

RRT* 算法为了路径的渐进最优性花费了较多的时间, 平均花费 2464ms. 由此可以看出, 基于采样的算法面对二维环境时是有较大的劣势的, 它更适合用于高维空间中的规划. 文献 [22] 提出的算法提出了一种新的变异算子, 虽然加速了优化过程并减少了迭代次数, 但在时间上由于变异操作花费了额外的时间, 其时间仍较长, 平均花费时间为 1900ms.



(a) D* Lite算法　(b) 本文算法　(c) RRT*算法　(d) improvedGA算法

图 6　四种算法的规划结果

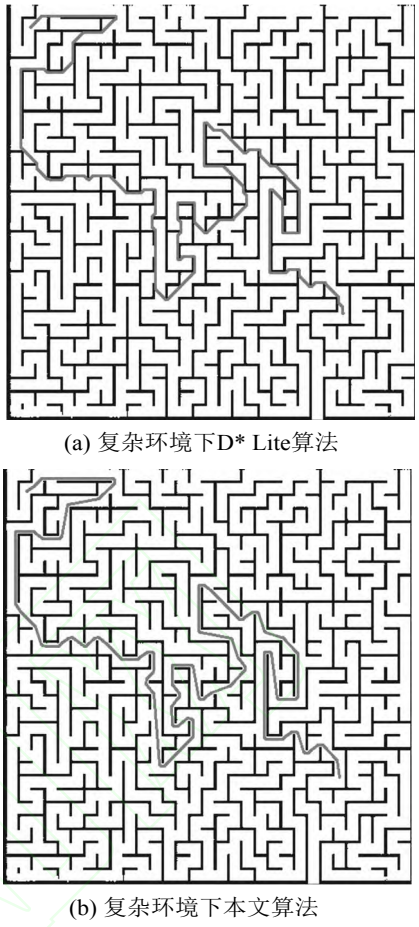
随机挑选一幅规划图像进行对比, 见图 6.

从图 6 可以清晰看出本文算法的优点, 即规划出平滑、安全的路径. 四种算法均找出了一条接近最优的路径, 而本文算法的路径在保持其路径长度较短的情况下, 也距离障碍物有一定距离. 相比之下,RRT* 算法规划出的路径虽然比 RRT 更接近最优解, 但仍存在一定随机性, 会有不必要的转折点. 文献 [22] 中算法规划出的路径虽然距离障碍物有一定距离, 且转折点也不多, 但无法分辨出狭窄的障碍物缝隙从而进入了较不安全的地方. 本文算法避开了上述问题, 但也存在缺点, 即路径节点数较多, 且规划时间比原始 D* Lite 算法有所增加.

3.4　复杂环境下规划路径比较

图 7 为在 500×500 复杂迷宫环境下所规划的路径比较.

图 7(a) 中原始 D* Lite 算法在复杂环境下规划出的路径在沿边时距离障碍物十分接近, 且极不平滑. 图 7(b) 中本文算法规划出的路径离障碍物均有一定距离, 且平滑程度也较高.



(a) 复杂环境下D* Lite算法　(b) 复杂环境下本文算法

图 7　复杂环境下规划比较

4　结语

本文提出了一种对原始 D* Lite 算法作出改进的 DLD* Lite 算法, 通过距离变换使得路径远离障碍物, 通过视线算法平滑路径. 本文将改进后的算法与原始 D* Lite 算法所规划出的路径进行了比较, 验证了算法对于不同环境的适应性. 然而, 在十分狭窄的环境中视线算法带来的时间开销较大, 下一步将继续研究狭窄环境下提升效率的方法.

参考文献 (References)

[1] 赵娟平, 高宪文, 刘金刚, 等. 移动机器人路径规划的参数模糊自适应窗口蚁群优化算法 [J]. 控制与决策, 2011, 26(7): 1096-1100.
(Zhao J P, Gao X W, Liu J G, et al. Parameters self-adaptive fuzzy ant colony optimization algorithm with searching window for path planning of mobile robot[J]. Control and Decision, 2011, 26(7): 1096-1100)

[2] Moravec H P, Elfes A. High resolution maps from angle sonar[C]// IEEE International Conference on Robotics and Automation. St Louis: IEEE, 1985: 116-121.

[3] Yershov D S, LaValle S M. Simplicial Dijkstra and A* algorithms for optimal feedback planning[C]//Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference. San Francisco: IEEE, 2011: 3862-3867.

[4] Dakulović M, Petrović I. Two-way D* algorithm for path planning and replanning[J]. Robotics and autonomous systems, 2011, 59(5): 329-342.

[5] Likhachev M, Koenig S. A generalized framework for Lifelong Planning A* search[C]// Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005). Monterey: AAAI Press, 2005:99-108.

[6] Souissi O, Benatitallah R, Duvivier D, et al. Path planning: A 2013 survey[C]// International Conference on Industrial Engineering and Systems Management. Rabat: IEEE, 2013:1-8.

[7] Uras T, Koenig S. An empirical comparison of any-angle path-planning algorithms[C]//Proceedings of the Eighth Annual Symposium on Combinatorial Search. Ein Gedi, the Dead Sea: AAAI Press, 2015: 206-212.

[8] Daniel K, Nash A, Koenig S, et al. Theta*: Any-angle path planning on grids[J]. Journal of Artificial Intelligence Research, 2010, 39: 533-579.

[9] Nash A, Koenig S, Tovey C A. Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D[C]// Symposium on Combinatorial Search, Socs 2010, Stone Mountain, Atlanta, Georgia, Usa, July. DBLP, 2010: 299-307.

[10] Dave Ferguson. Using interpolation to improve path planning: The Field D* algorithm[J] . Journal of Field Robotics, 2006, 23(2): 79-101

[11] Z.N. Low.Pulse detection algorithm for line-of-sight (LOS) UWB ranging applications[J]. IEEE Antennas & Wireless Propagation Letters, 2005, 4(1): 63-67.

[12] 游晓明, 刘升, 吕金秋. 一种动态搜索策略的蚁群算法及其在机器人路径规划中的应用 [J]. 控制与决策, 2017, 32(3): 552-556.
(You X M, Liu S, Lv J Q. Ant colony algorithm based on dynamic search strategy and its application on path planning of robot[J]. Control and Decision. 2017, 32(3): 552-556.)

[13] 王雷, 李明, 唐敦兵等. 基于改进遗传算法的机器人动态路径规划 [J]. 南京航空航天大学学报, 2016, 48(6): 841-846.
(Wang L, Li M, Tang D B, et al. Dynamic Path Planning for Mobile Robot Based on Improved Genetic Algorithm[J]. Journal of Nanjing University of Aeronautics & Aeronautics, 2016, 48(6): 841-846)

[14] 朱大奇, 颜明重. 移动机器人路径规划技术综述 [J]. 控制与决策, 2010 (7): 961-967.
(Zhu D Q, Yan M C, Survey on technology of mobile robot path planning[J]. Control and Decision. 2010(7): 961-967.)

[15] Bry A, Roy N. Rapidly-exploring Random Belief Trees for motion planning under uncertainty[C]// IEEE International Conference on Robotics and Automation. Shanghai: IEEE, 2011: 723-730.

[16] Yan F, Liu Y S, Xiao J Z. Path planning in complex 3D environments using a probabilistic roadmap method[J]. International Journal of Automation and Computing, 2013, 10(6): 525-533.

[17] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning[J]. The international journal of robotics research, 2011, 30(7): 846-894.

[18] Geraerts, Roland.Creating high-quality paths for motion planning[J].International Journal of Robotics Research, 2007, 26(8):845 - 863.

[19] Bailey D G. An efficient euclidean distance transform[C]//International workshop on combinatorial image analysis. Berlin, Heidelberg: Springer, 2004: 394-408.

[20] Akmal Butt, M. Maragos, P..Optimum design of chamfer distance transforms[J]. IEEE transactions on image processing, 1998, 7(10):1477-1484.

[21] Eades P, Feng Q W, Lin X. Straight-line drawing algorithms for hierarchical graphs and clustered graphs[C]//International Symposium on Graph Drawing. Berlin, Heidelberg: Springer, 1996: 113-128.

[22] Tuncer A, Yildirim M. Dynamic path planning of mobile robots with improved genetic algorithm[J]. Computers & Electrical Engineering, 2012, 38(6): 1564-1572.

(责任编辑：X X)