第 2 章　正文

2.1　分层安全区间路径规划（HSIPP）

我们首先实现了 SIPP 算法[17]的拓展，并将其称之为分层安全区间路径规划（HSIPP）。尽管都是将 SIPP 算法拓展到连续时间的路径规划上，通过不同于 MAPF-LNS2[19]中 SIPPS 算法的实现方法，HSIPP 算法能够保证安全区间的数量与 SIPP 中的理论是同阶的。进一步地，我们能够给出 HSIPP 算法的时间复杂度的理论保证。据我们所知，HSIPP 是在连续时间的条件下，第一个给出了理论时间复杂度保证的，能够被应用在一般图、一般冲突关系和软约束的算法。

2.1.1　HSIPP 算法的相关定义

软约束路径规划　一般的无碰撞路径规划问题要求对于已经存在的路径 $P_1, P_2, ..., P_k$，找到一条路径 P 满足 $\forall 1 \leqslant i \leqslant k, cost(P, P_i) = 0$。但是这种路径并不总是存在的（例如起点在一开始就被阻挡的情况）。因此，我们希望找到一条路径在最小化 $\sum_{i=1}^{k} cost(P, P_i)$ 的基础上尽可能短。事实上，出于对时间效率的考虑，在 HSIPP 中，我们寻找的路径实际上最小化的是代价之和的一个下界 $c(P) \leqslant \sum_{i=1}^{k} cost(P, P_i)$。但是，我们要求 $c(P) = 0$ 当且仅当 $\sum_{i=1}^{k} cost(P, P_i) = 0$，这就保证了如果我们找到了一条 $c = 0$ 的路径 P，那么这一路径也一定是无碰撞路径。

安全区间　在 SIPP[17]中，一个节点的安全区间被定义为这一节点上所有的极长的的与已存在路径路径无冲突的时间区间。在 HSIPP 中，我们使用和 SIPPS[19]类似的方法，将 SIPP 安全区间的补集也定义为安全区间。因此，对于一个节点 x，它的所有安全区间可以表示为 $\{[l_{x,1}, r_{x,1}), [l_{x,2}, r_{x,2}), ..., [l_{x_{S_x}}, +\infty)\}$，其中 $r_{x,i} = l_{x,i+1}$，记作 $I_{x,1}, I_{x,2}, ...$。在每一段安全区间内部，要么所有的时间都与某一个已经存在的路径（不一定是同一个）相冲突，此时我们称这一个安全区间是有代价区间；要么所有的时间都不与任何已存在的路径冲突，称之为无代价区间。尽管在同一个安全区间可能会与多个不同的路径发生冲突，但是我们假定在安全区间内部停留不会增加额外的代价（即 $c(P)$），这一点与 SIPPS 相似。

2.1.2 HSIPP 结点与拓展

HSIPP 的结点 x 可以用六元组 $x.cost, x.v, x.k, x.arrival, x.t_l, x.t_r$ 表示，其中 $x.cost$ 表示当前到达结点 x 的路径 P 的代价下界 $c(P) = x.cost$。当前的结点对应图上节点 $x.v$ 的第 $x.k$ 个安全区间，其中最早的满足代价为 $x.cost$ 的到达结点 x 的时间是 $x.arrival$，当且节点对应的安全区间是 $[x.t_l, x.t_r)$。注意到在一个安全区间内停留是不会增加代价的，因此实际上在 $[x.arrival, x.t_r)$ 时间 AGV 都可以停留在 $x.v$ 上（注：在无歧义条件下，HSIPP 的结点和图上的节点分别简称为结点和节点）。

HSIPP 的结点拓展有两种方式：一种是在当前节点等待到同一节点的下一个安全区间，若下一个安全区间是有代价区间，那么 $cost$ 增加 1，这对应于算法2.1中的第一部分；另一种则是通过一条边 e 到达新的节点 y，我们将详细阐述如何通过合理的实现来确保这一部分的运行效率，算法2.1中由第二部分实现。

假设我们从图上的节点 u 通过边 e 前往节点 v，当前的 HSIPP 结点为 x。首先考虑无代价的情况：如果存在一个 e 上无任何冲突的区间 $[l_e, r_e)$ 和 v 上的第 k 个（无代价）安全区间 $[l_{v,k}, r_{v,k})$，使得存在 t 满足：1、$t \in [x.arrival, x.t_r)$；2、$[t, t + L_e) \in [l_e, r_e)$；3、$t + L_e \in [l_{v,k}, r_{v,k})$，那么我们就可以添加新的节点 y 满足 $y.cost = x.cost, y.v = v, y.k = k, y.arrival = t + L_e, y.t_l = l_{v,k}, y.t_r = r_{v.k}$。实际上，对于给定的 x, u, v, e，满足条件的 t 一定构成一个区间，我们只需计算出左端点并更新即可。对于有代价的情况，无论是在边上还是点上的冲突，我们一律使用如下的拓展方式：对于点 v 上的第 k 个（有无代价均可）安全区间 $[l_{v,k}, r_{v,k})$，如果 x 能够通过 e 在时间 $l_{v,k}$ 到达，则添加新的结点 y 满足 $y.cost = x.cost + 1, y.v = v, y.k = k, y.arrival = l_{v,k}, y.t_l = l_{v,k}, y.t_r = r_{v.k}$，否则新的结点 y 的到达时间 $y.arrival = x.arrival + L_e$。

在实现上面的算法时，如果我们按照从左到右的顺序来处理区间，那么可以做到每个安全区间对应的结点被拓展的次数关于总安全区间次数是均摊 $O(1)$ 的。同时，对于每一个安全区间，我们只需要保留其所有结点中 $x.arrival$ 最小的 x 即可（这一 $x.arrival$ 称为该安全区间的最早到达时间）。更仔细的分析将会在定理2.1中给出。

2.1.3 HSIPP 的整体算法

代码2.2给出了 HSIPP 的主体伪代码。HSIPP 首先需要对所有图上的点和边找到无冲突的极大区间，从而生成节点上的安全区间以及边上的无冲突的极大区间。

HSIPP 的运行是按照 $c(P)$ 进行分层的，因此总是会找到 $c(P)$ 最小的路径。从 $c(P) = 0$ 开始，HSIPP 首先找到起点并生成初始的无代价队列 Q。对每一轮 $c(P)$，

算法 2.1 单层 SIPP($Q, Q_c, goal, \mathcal{I}$)

输入： 无代价优先队列 Q，有代价集合 Q_c，终点节点 $goal$，所有安全区间 $\mathcal{I}$
输出： 路径 P，有代价集合 Q_c

```
P ← ∅
while Q ≠ ∅ do
    x ← Q.pop()
    if x.v = goal then
        P ← 生成从起点到终点，代价为 x.cost 的路径
        break
    end if
    if I_{x.v,x.k+1} 是无代价安全区间 then
        y ← Node(x.cost, x.v, x.k + 1, l_{x.v,x.k+1}, l_{x.v,x.k+1}, r_{x.v,x.k+1})
        Q.push(y)
    else
        y ← Node(x.cost + 1, x.v, x.k + 1, l_{x.v,x.k+1}, l_{x.v,x.k+1}, r_{x.v,x.k+1})
        Q_c.append(y)
    end if// 第一部分
    for e 为从 x.v 出发的边 do
        y.v = Y_e
        for I_{y.v,k} 为所有 y.v 上的和 [x.arrival + L_e, x.t_r + L_e) 相交的安全区间 do
            t ← max{x.arrival + L_e, l_{y.v,k}}
            Q_c.append(Node(x.cost + 1, y.v, k, t, l_{y.v,k}, r_{y.v,k}))
            t_0 ← min{t ∈ [x.arrival + L_e, x.t_r + L_e) ∩ [l_{y.v,k}, r_{y.v,k}) | [t, t + L_e)对于边e是无代价的}
            if t_0 存在 then
                Q.push(Node(x.cost, y.v, k, t_0, l_{y.v,k}, r_{y.v,k}))
            end if
        end for//第二部分
    end for
end while
return P, Q_c
```

HSIPP 维护一个优先队列 Q 表示所有代价为 $c(P)$ 的路径的终点结点，以及一个集合 Q_c 表示所有在当前轮次生成的代价为 $c(P)+1$ 的路径的终点结点。然后我们就可以使用上文所说的拓展的方法来得到所有 $c(P)$ 代价下能够被拓展到的结点，以及可能得新一轮 $c(P)+1$ 的初始结点并将它们存储在 Q_c 中。在这一轮中，Q_c 并不需要做任何排序操作，这样可以避免大量的计算开销。一旦我们找到了终点所对应的结点，这一路径一定是当前 $c(P)$ 下长度最短的路径，可以提前返回结果。

2.1.4 HSIPP 分析

首先，我们容易发现如果存在无代价路径，那么 HSIPP 的拓展方式一定能够将其找到，因此 HSIPP 对于无代价路径的存在性的判定是正确的。这也说明 HSIPP 满足我们在上文所说的 $c(P)=0$ 当且仅当 $\sum_{i=1}^{k} cost(P, P_i) = 0$。

同时，如果假设所有的点和边上的平均安全区间数量为 $\rho(I)$，最大安全区间

算法 2.2 HSIPP

输入： 起点 S，终点 T
初始化所有安全区间 $\mathcal{I}$
$cost \leftarrow 0$
$start.cost \leftarrow 0, start.v \leftarrow S, start.k \leftarrow 0, start.t_l \leftarrow 0, start.t_r \leftarrow r_{S,0}$
$Q_0 = \{start\}, Q_1 = \{\}$
最终路径 $P \leftarrow \varnothing$
while $P = \varnothing$ **do**
将集合 Q_{cost} 转化为优先队列
$P, Q_{cost+1} \leftarrow$ 单层 SIPP$(Q_{cost}, Q_{cost+1}, T, \mathcal{I})$
$cost \leftarrow cost + 1$
end while
return P

数量为 $m(I)$，那么我们有如下定理来保证时间复杂度：

定理 2.1： 在 HSIPP 算法的每一轮中，所有安全区间的最优到达时间最多被更新 $O(\rho(I)|E|)$ 次；所有拓展出的结点最多为 $O(\rho(I)m(I)|E|)$ 个。

证明： 对于每一个安全区间 $I_{u,k}$ 对应的结点 x, 其通过边 e 出发最多会将到达的点 v 上的所有安全区间全部尝试更新一次最优到达时间, 因此总的拓展结点数量最多为 $O(\rho(I)m(I)|E|)$。实际上, 如果我们将 u, v, e 上的安全区间从左到右排序后, 尝试更新的次数和遍历区间的次数将相等,因此我们的运行时间也将是 $O(\rho(I)m(I)|E|)$。接下来我们考察所有安全区间被更新的情况：假设安全区间 $I_{u,k}$ 通过边 e 尝试更新 $I_{v,l}$ 的最优到达时间。首先，每个安全区间 $I_{v,l}$ 的最优到达时间一旦被更新为了安全区间的起始时间 $l_{v,l}$，则任何尝试更新都将被拒绝；因此我们只需考察所有非起始时间的尝试更新，而这只能由 $I_{u,k}$ 的到达时间或者 e 中的无冲突极大区间的左端点贡献，因此总的尝试更新不超过 $O(\rho(I)|E|)$（即安全区间总数）次。 ∎

根据定理2.1，如果我们使用二叉堆来作为优先队列，则每一轮的运行时间复杂度为 $O(\rho(I)|E|(\log|V| + m(I)))$，因此假设最终找到的路径的代价下界为 $c(P)$，总的复杂度就是 $O(c(P)\rho(I)|E|(\log|V| + m(I)))$。在实际情况中，$c(P)$ 一般都是 0 或 1，$\rho(I)$ 一般在 2 3 之间，$m(I)$ 的贡献可以忽略不计。如果与一般的基于（时间，节点）的 A*（即 PP 算法[7]）相比较，得到的单次路径规划的平均运行时间如表格2.1所示。更进一步，HSIPP 的优势在于对于连续时间更好的精度，因此实际上更容易找到代价更小或者更短的路径。