# 一、问题描述

程序针对带有时间窗的VRP问题，即VRPTW问题，目标是找到一组广义系统成本最小的路线，包括运输成本、车辆等待成本和车辆固定使用成本。给定一个由(N,M)表示的物理运输网络，其中N是节点集合，M是有向弧的集合。我们使用TT(i,j,t)来表示在时刻t出发在弧(i,j)∈M上的行驶时间。假定行驶时间可以根据交通条件计算得到。

考虑的网络中有两种不同类型的节点，子集P表示的客户节点，由配送中心o表示的客户节点，即N={o}⋃P。每个客户p∈P有相应的需求的体积和重量以及服务时间窗[,]，其中是最早服务时间，是最晚服务时间。P中的所有客户节点只能被一辆车访问一次。

配送中心o是所有车辆出发的起点和终点，同时车辆也可以回到配送中心重新装卸货。也就是说，在送完一批包裹之后，车辆可以返回配送中心装载下一批需要配送的货物。每个车辆也有相应的服务时段[,]，其中是最早离开配送中心的时间，是最晚返回配送中心的时间。

在Python程序中，一共有number（N）=number（P）+2（起讫点）

基于物理网络，将其扩展为具有空间维度、时间维度和累积服务状态维度的基于时间索引和状态索引网络。用表示高维网络，点有物理网中的节点扩展，弧表示从顶点到顶点的有向时空状态路径。其中，t表示规划时间维度中的均匀离散时间间隔(例如，1分钟)，这个维度自然地嵌入了时间窗约束。w表示车辆的“累积服务状态”，用于记录服务了的客户并提供相应的状态和体积信息，以满足车辆容量约束。返回配送中心o后，车辆的累计服务状态被重置。

（该图中没有体现出装卸弧和等待弧）

# 二、模型构建

包括三种类型的弧：网络中的弧可以分为以下三类：运输弧、等待弧和装卸弧。运输弧表示基于给定的旅行时间，=，一辆车从节点I移动到节点J。等待弧表示从时间t到t + 1的节点i上的等待活动，即车辆位置和承载状态在一个时间间隔内保持不变。装卸弧 表示车辆在配送中心装载和在乘客点配送过程，只有这个弧会有状态的变化。装卸弧没有费用，车辆出动则有固定使用成本，运输有运输成本，等待有等待成本，所以考虑了三类成本。

时间窗 、车辆能力（体积、重量）能力约束 已经被提前嵌入到高维网络中。因此，对每辆车而言，我们只需要确保所有选择的弧都能够遵循流量平衡约束，构成从起点到终点的可行路径。

起点的流平衡约束：

|  |  |
| --- | --- |
|  | (2) |
| 终点的流平衡约束： | (3) |
| 中间点的流平衡约束： | (4) |

约束(2)和(3)确保每个车辆在研究时段开始时（即时刻）以初始状态离开原点，并在研究时段结束时（即时刻）到达目的地。约束(4)保证其他中间节点上的流量平衡。

|  |  |
| --- | --- |
| 需求约束： | (5) |

约束（5）确保每一个客户正好被服务一次。

|  |  |
| --- | --- |
|  | (6) |

约束（6）定义了车辆对于每个弧的选择是个0-1变量。

# 三、模型重构

1、对偶化

可以分解子问题为：

2、增广拉格朗日松弛

ADMM内部的块坐标下降法可将其分解子问题为：

基于决策变量的二元性（只有0，1两个取值），子问题中的二次项可以被分离、化简和重组为线性化的目标函数：

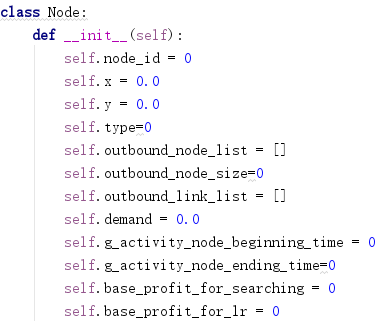
|  |
| --- |
|  |
|  |

# 四、算法流程

## 输入数据

### 1、node数据

Node的字段包括：



（1）车辆起点node数据 (node\_type=1)，Node\_id=0, node.g\_activity\_node\_beginning\_time=0

（2）客户node数据（node\_type=2）"input\_node.xlsx"，包括客户id，物理坐标（X，Y坐标），每个点的capacity需求，最早服务时间和最晚服务时间，服务时长。

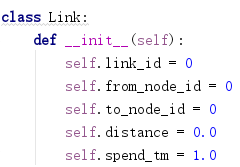
设置客户node的node.base\_profit\_for\_searching = base\_profit, node.base\_profit\_for\_lr = base\_profit。node.base\_profit\_for\_searching是指（本文程序中乘以2，因为roh值是给的，所以可以乘以2），node.base\_profit\_for\_lr是指。

（3）车辆终点数据 (node\_type=1)，即原点，但是程序中所罗门数据因为不能返回起点，所以需要将其终点重新设置。终点的基本数据为node\_id=26, node.g\_activity\_node\_ending\_time = g\_number\_of\_time\_intervals.

此时，共有node数量27个，客户数量为25个。

### 2、link数据（物理线路数据）'input\_link.csv'

link字段包括：



Link数据内容包括：link\_id, 起点node的id，终点node的id，link的长度（可由x.y坐标计算），link上车辆旅行时间。

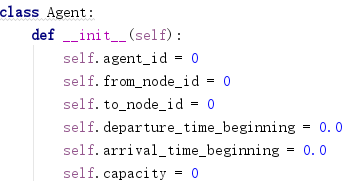
### 3、根据link数据，将点和线关联起来，确定每个点的outbound link，outbound node，写入class node中。

g\_node\_list[link.from\_node\_id].outbound\_node\_list.append(link.to\_node\_id)

g\_node\_list[link.from\_node\_id].outbound\_link\_list.append(link)

### 4、agent数据（车辆数据）

agent字段包括：



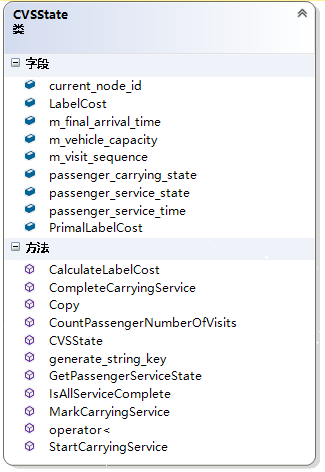
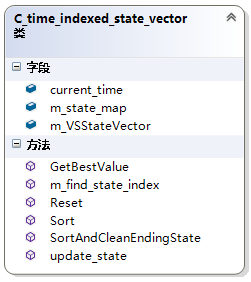
Agent数据内容包括agent\_id, 车辆起点node\_id，车辆终点node\_id，车辆能力，车辆开始服务时间，车辆结束时间。

## 算法过程



### DP算法（Mahmoudi M（2016））

程序利用前向动态规划算法求解车辆最短路径。动态规划的原理是：每个阶段的最优状态可以从之前某个阶段的某个或某些状态直接得到而不管之前这个状态是如何得到的。

在DP算法中，需要用到CVSState和C\_time\_indexed\_state\_vector两个类，类视图如图所示。

CVSState用来更新、记录车辆的位置、状态、费用等信息。

C\_time\_indexed\_state\_vector用来对每个时间t上不同状态费用进行排序，在动态规划算法中，随着t的推移，每一个时间t存在的状态会越来越多，到达中间时（状态最多的部分），可能会发生数据爆炸使得无法求解。

因此在动态规划算法中我们，引入了参数BestKSize。通过对每一个时间t的状态的labelcost进行排序，只保留前BestKSize个状态，可以有效地控制问题规模。

我们采用前向动态规划来计算一辆车最短路径的最小成本。步骤如下：

For vehicle v

For time t

for state w //当前时间t前BestKSize个状态（即当状态超过BestKSize数量之后，这一步按照cost只保留前BestKSize个状态进行下一步计算，以保证下一步的状态数量不会过多，之后再状态超过BestKSize数量之后，再按照cost只保留前BestKSize个状态进行下一步计算，每一步到下一步的可分支数保证为BestKSize个）

for outbound of current w’s node i

calculate labelCost of (i,t,w):

if(the state does not exist)

create it

else if(the label cost of the temp state < label cost of the existing state)

update the cost of the state

在 *for outbound of current w’s node i*中，分以下几类情况：

（1）对于可能发生状态变化的：

①判断是否超出时间窗结束时间；

② feasible state transitions 发生状态变化:



a. next\_time 小于pickup时间窗的开始时间；

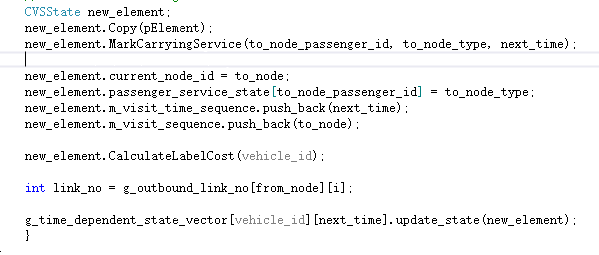
b. next\_time 小于 delivery 时间窗的开始时间；

c. next\_time 整好处于pickup 或delivery时间窗内。

（2）到达最终状态

（3）普通physical link

//根据以上各种情形分类，对CVSState进行更新



CVSState 更新

最终DP返回车辆到达最终状态所需要的最小费用。

return g\_ending\_state\_vector[vehicle\_id].GetBestValue(DualCostFlag, vehicle\_id);

其中，CalculateLabelCost根据目标函数来确定，见void CalculateLabelCost(int vehicle\_id)

本程序中，LabelCost包括三部分：

① vehicle transportation cost：费用为1；

② vehicle waiting cost：费用为0.5；

③ passenger waiting cost: 费用为0.3。

各部分的费用可以根据需要自行设定。

参数DualCostFlag说明：

当计算LR下界，调用DP算法时，参数DualCostFlag=1，path\_cost\_by\_vehicle\_v的返回值为LabelCost，即模型中的。

当计算LR上界，调用DP算法时，参数DualCostFlag=0，path\_cost\_by\_vehicle\_v的返回值为primalLabelCost，即模型中的cy。

输出数据：profit、path（path\_no\_eq、path\_time\_sq）、gap（LB、UB）