考虑出行时间窗的定制公交线路车辆调度方法：http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CJFQ&dbname=CJFDLAST2018&filename=ZGGL201805018&uid=WEEvREcwSlJHSldRa1FhdXNXaEd1OFFqN1JLQ0syd1lZUDlTSjhMdURDTT0=$9A4hF\_YAuvQ5obgVAqNKPCYcEjKensW4IQMovwHtwkF4VYPoHbKxJw!!&v=MDk5NDRSTE9mWk9kckZpRGtVYnZMUHlyTVlyRzRIOW5NcW85RWJJUjhlWDFMdXhZUzdEaDFUM3FUcldNMUZyQ1U=

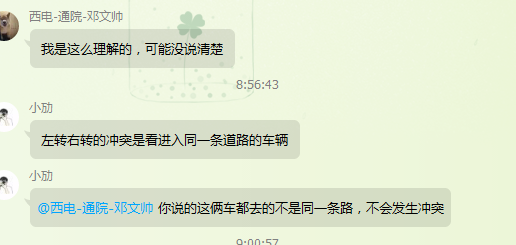
可考虑的思路：

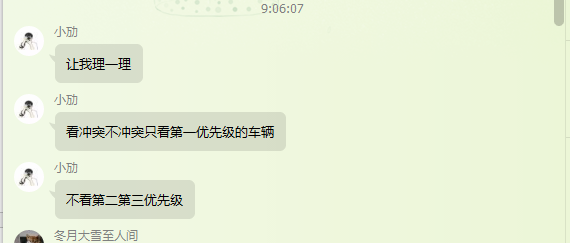
1.分为离线阶段和在线阶段：离线阶段:根据车间路线图提取节点坐标和节点间关系, 建立以节点距离为权值的无向图模型。用该模型和K-最短路径算法, 生成任意两个节点间的K条候补路径集, 并以路径表的形式存储起来, 构成路径库。

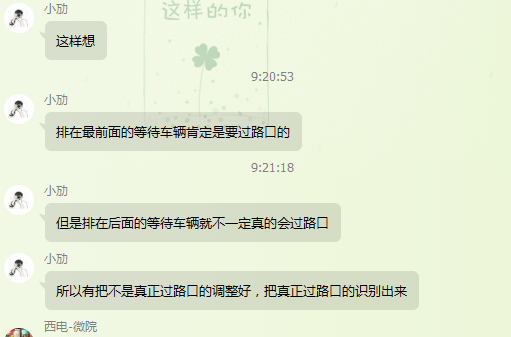
参考：基于多种群遗传算法的多AGV调度，http://kns.cnki.net/KXReader/Detail?TIMESTAMP=636881523229291250&DBCODE=CJFD&TABLEName=CJFDLAST2018&FileName=DZKK201811013&RESULT=1&SIGN=TkR6179KAnsWKhz03ehPqVLqGGM%3d&UID=WEEvREcwSlJHSldRa1FhdXNXaEd1OFFqN1JLQ0syd1lZUDlTSjhMdURDTT0=$9A4hF\_YAuvQ5obgVAqNKPCYcEjKensW4IQMovwHtwkF4VYPoHbKxJw!!&filetitle=%E5%9F%BA%E4%BA%8E%E5%A4%9A%E7%A7%8D%E7%BE%A4%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95%E7%9A%84%E5%A4%9AAGV%E8%B0%83%E5%BA%A6\_%E5%AD%9F%E5%86%B2

规则解释：

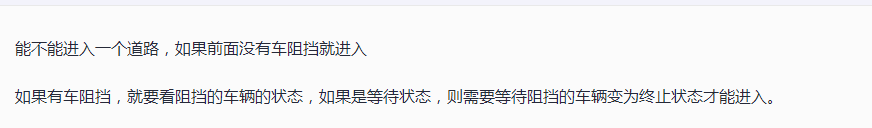
左转右转的冲突是看进入同一条道路的车辆







https://bbs.huaweicloud.com/forum/thread-15007-1-1.html



1. 给每辆车规划N条不同最短路径的思路：

weight = weight\_initial()

adj\_list = {}

paths = {}

for i in range(N):

path = shortest\_path(adj\_list, car\_df)

paths.append(path)

# 修改邻接表中边的权重，增大在path中使用过的边的权重

adj\_list = adl\_update(adj\_list, path)

return paths

函数adj\_list = adl\_update(adj\_list, path)中权重更新的设计思路：

For edge in all:

# 得到该条边在所规划的路径中出现的次数

num = use\_time(edge, paths)

Weight += lambda \* num^2 #lambda为权重因子，初始时暂取0.02

1. 权重初始化函数的设计：weight = weight\_initial()

# 最短路径是使所有边的权重之和最小

weight = lambda1 \* (s/v) + lambda2 \* (1/n) + lambda3 \* X….

其中lambda表示权重因子，(s/v)表示该条边长度除以最大通行速度，即表示该条边的通行时间; n表示该条边对应道路的车道数，即车道数越大，(1/n)值越小，表示通行耗时越短; X表示其他要考虑的因素。

对于权重的修改是为了规划出不同的路径，也可以考虑直接对地图进行剪枝操作，然后再进行路径规划，剪枝后的规划也可以得到不同路径，并且规划速度更短，但是可能会出现无法得到可行路径的问题。

3．根据出发时刻分布，以及车的数量对路径进行规划：首先得到车出发的计划时刻分布，对这些不同发车时刻的车按照一定比例分割为N份，分别对这N份车进行路径规划。在对每份车进行路径规划时，每得到一辆车的路径，就更新更新一次邻接表中的权重，思路类似1中邻接表的更新adj\_list = adl\_update(adj\_list, path)，这样做的目的是争取让地图中的每条路都被用到，从而使地图道路的利用率最高。

4. 道路当前时刻负荷率的设定：

miu = (car\_in\_road \* 1) / road\_all\_space

road\_all\_space：表示所有道路的每条车道的长度之和，即可以用来表征当前地图的负载能力。计算公式是：

for road in roads:

for channel in road:

road\_all\_space += channel.getLength()

通过控制负荷率小于一个值，来防止锁死情况的发生，即当负荷率高于某个值时，暂停发车。

5 . Hamilton问题：<https://github.com/michzio/Python---Public-Transport-Shortest-Path-Dijsktra-and-Hamilton-Cycle-Search/tree/master/python>

<https://github.com/zxlim123/Hamiltonian-Path/blob/master/HamiltonianPath.py>

50个城市路径问题：

<https://github.com/TheMalaka/Ensemble-Cheapest-Insertion-Heuristic-to-Solve-Hamilton-Path-Problem>

Java 基因遗传算法求取hamilton路径：

<https://github.com/mohamedalamicom/HamiltonianCircuit/blob/master/js/genetic-algorithm.js>

java:

https://github.com/rayronvictor/DisjointHamiltonianCycles

C++实现：

<https://github.com/samarth-p/Euler-and-Hamiltonian-Path/blob/master/DMS%20program.cpp>

https://github.com/Guber/Hamiltonian

复杂度最低的C++实现：Christofides algorithm in plane. 平面克里斯托费尔德斯算法

<https://github.com/VladX/Euclidean-Hamiltonian-Path>

重要的python实现：https://github.com/kriths/square-sum-problem

https://github.com/anmolg1024/AdvancedTSP/blob/master/AdvancedTSP.py

其他博客：

<https://blog.csdn.net/baimafujinji/article/details/49687061>

<https://blog.csdn.net/zhangyifei521/article/details/53283028>

https://blog.csdn.net/zhb67473517/article/details/80364476

首先考虑Hamilton路径是否存在的问题

<https://max.book118.com/html/2017/0706/120636894.shtm>

matlab实现，以及各种方法性能对比：DFS，BFS

https://blog.csdn.net/u010244645/article/details/72799963

6.思路：先根据每条道路的通行能力，即之前的权重函数，对通行能力较差的节点剔除，然后求剩余节点间的Hamilton路径（大圆环），在考虑将剔除的节点接回至Hamilton路径中（再构成一个小圆环）

TSP问题：<https://github.com/dmishin/tsp-solver>

hp:

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4963334/>

<https://arxiv.org/pdf/1405.6347.pdf>

实现代码：<https://figshare.com/articles/Implementation_Necessary_Code/1057722>

jave：https://www.sanfoundry.com/java-program-hamiltonian-cycle-algorithm/

论文：

<https://www.hindawi.com/journals/jopti/2018/9328103/#B22>

可用的算法：

<https://en.wikipedia.org/wiki/Hamiltonian_path_problem#Algorithms>

c++实现和算法讲解：

<https://www.hackerearth.com/practice/algorithms/graphs/hamiltonian-path/tutorial/>

有用的实现：

<http://blog.mayec.eu/2011/05/python-hamiltonian-cycles-in-graph.html>

# 参考论文：Efficient solution for finding Hamilton cycles in undirected graphs

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4963334/

很厉害的论文：武汉大学A Polynomial time Algorithm for Hamilton Cycle and Its Proof

https://www.researchgate.net/post/My\_Algorithm\_and\_exe\_download\_for\_Hamilton\_cyclepath\_Can\_anyone\_help\_with\_English\_expression\_of\_the\_paper

超赞的算法实现和讲解：

<https://towardsdatascience.com/slitherin-solving-the-classic-game-of-snake-with-ai-part-1-domain-specific-solvers-d1f5a5ccd635>

<https://github.com/gsurma/slitherin>

超牛B的数学软件系统，可以计算图论相关问题：

<http://www.sagemath.org/>

api介绍：

<http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/generic_graph.html>

今日最佳：<https://github.com/SiewYean/HamiltonianPathProblem-Non-exactMethod-/blob/master/grasp.py>

待查：<https://github.com/LeslieK/Algorithms-Python/blob/master/Hamiltonian.py>

20190318：

HP—>hamiltonian path HC—>hamiltonian cycle

当前程序存在的问题：

1） 权重函数的设置问题（车道数、最大速度的因子设置？查看道路车道数和最大速度的分布情况，根据分布情况决定参数设置？）

2）剪枝参数的设置问题（根据道路的速度和车道数如何设置合理的剪枝参数）

3） 在选择HP中的路径时，没有考虑最短路径的问题

4）时间规划问题还没有考虑（如何规划出发时间？将跑得慢的车延后出发？）

5）将都在HP中的节点打包成一个大节点后，邻接关系的继承问题？（这一点没有完全理清楚，有待细想，特别是大节点如何继承两个同时属于HP的节点间的邻接关系）

6） HC的寻找问题？能否利用当前的HP搜索程序，在一定时间中通过检查首尾是否相连来寻找最优的HC？（HC属于NP难问题，暴力搜索不能解决，复杂度为指数级）

7） HP的搜索没有考虑最短路径问题，即得到的HP通行能力不一定最强，所以需要添加通行能力的指标（权重），来找一条最优的HP。

8）HP搜索次数的设置，以及HP包含最少节点数的设置需要尝试不同参数

9）HP包含节点个数与能否降低锁死情况发生的讨论，同时要在HP长度与最终车运行时间之间进行权衡

一些想法

1.为什么提供如此详细的交规，如何将路径规划和交规结合起来，从而提高调度效率防止锁死？？

2.多进程的想法：python多线程对计算速度提升不大，故考虑多进程的实现，可以在为每辆车求解路径时使用多进程，将要求的小车分成N份，启动N个进程同时求解。。

多进程使用的参考网 ：<https://www.cnblogs.com/kangoroo/p/7628092.html>

3.是否存在空间换时间的思路？？？（比如预先搜索和存储HP,HC？）

4.为了减少运行时间，必须要求HC

5.考虑数据量的问题，针对不同的数据量能否使用不同的算法？

6.出入HC/HP时是否会发生锁死的情况？？：

理想情况：所有节点均在HC/HP上，肯定不会锁死

可以这样想：尽可能多的节点在HP/HC上，锁死的概率就会降低？？

7.时间规划的想法：

1）根据道路当前负载能力（近似表示为：能排开多少辆车，也就是长度乘以车道数，然后求和所有在HP上的路），按出发时刻对车辆排序后进行分批

2）考虑车起始点和HC/HP之间的关系分析如何安排出发时刻

3）批与批之间考虑延时？

8.必须考虑某种规划方法失败时如何重新规划的问题，即必须要提高程序的鲁棒性：

当前程序的规划流程：

origin\_adl：原始的邻接表

cutted\_adl: 剪枝后重建的邻接表

get all cars paths:

for car in all\_cars:

try:

# 基于Hamiltonian path的路径规划

path = get\_path\_with\_hp(car, cutted\_adl)

except:

# replan

# 当基于HP的路径规划失败或出错时，使用原始邻接表和规划算法重规划路径

path\_origin = get\_path\_use\_dijkstra(car, origin\_adl)

# 从首位搜索其中是否包含存在于HP中的节点，然后将规划出来的路径重组，连接至HP中

path = connect\_path\_to\_pah(path\_origin)

#将得到的路径存储

all\_paths.append(path)

return all\_paths

下一步合理的结构：

路径规划的算法使用优先级: get\_path\_with\_hc > path = get\_path\_with\_hp > replan

get all cars paths:

for car in all\_cars:

try:

# 基于Hamiltonian cycle的路径规划

path = get\_path\_with\_hc(car, cutted\_adl)

except:

# 基于HC的路径规划失败，尝试基于HP的路径规划

try:

# 基于Hamiltonian path的路径规划

path = get\_path\_with\_hp(car, cutted\_adl)

except:

# replan

# 当基于HP的路径规划失败或出错时，使用原始邻接表和规划算法重规划路径

path\_origin = get\_path\_use\_dijkstra(car, origin\_adl)

# 从首位搜索其中是否包含存在于HP中的节点，然后将规划出来的路径重组，连接至HP中

path = connect\_path\_to\_pah(path\_origin)

#将得到的路径存储

all\_paths.append(path)

return all\_paths

9.HP/HC的选择标准：

1） 尽量使直行的道路多，转弯的道路少

2）根据车辆出发地和目的地的数量选择，让尽可能多的车处于HP,HC上

10. 接入HP/HC时路径的选择：

1）根据要走的节点多少选择

2）根据要走的边权重之和的大小决定

11. 时间规划与路径规划耦合进行：每次路径规划后实时更新权重信息，根据路径规划的信息进行时间规划，然后根据时间规划的车辆发车时间分布，再进行路径规划。

相当于下一辆车的时间和路径规划要考虑上一辆车的情况。

对road.txt car.txt cross.txt的数据分布进行分析，挖掘有用的信息

12.分析当前耗时的原因：

1）由于添加了延时导致？

2）由于同时发车太多堵车导致？

12.初始权重函数

T = s/v

La1 = 1

La2 = 0.5

W = la1 \* (s/v) + la2 \* (- n/n\_max \* t\_mean )

此时w分布于[0.1, t\_MAX]

设置W下界为0.1

13.更新权重函数

根据道路使用来更新，统计node🡪next\_node，边的使用频次n

n[edge] += 1

W += n[edge] \*

每T时刻重置一次权重至初始状态

14.HC的最优选择问题，除了考虑长度问题，考虑直行路口多少的问题。

For pre\_node,node, nextnode in zip(path)

If is\_direction\_cross(pre\_node,node, nextnode):

Count += 1

Return count

15.时间规划问题：