第六届湖南省研究生数学建模竞赛

题 目　　基于迭代邻域搜索算法的交通信号灯全局调控策略研究

摘　　要：

随着城市交通情况日趋复杂，传统固定周期固定顺序的交通信号灯调控策略难以为继。如今，在实现信号灯互联的基础上，现代化城市开始关注信号灯全局优化方案设计这一问题。良好的全局优化方案能大大减少交通拥堵状况，提升居民幸福感以及城市经济发展速度。然而，由于城市中路口众多，交通错综复杂，不同信号灯策略间存在相互影响，如何实现大规模的交通信号灯全局策略调控是亟待解决的问题。本文提出了基于迭代邻域搜索算法的交通信号灯全局调控方法，并引入贪心思想，基于关键路口的局部最优策略生成城市总体的满意调控策略，根据对策略的评价，迭代产生新的局部与全局信号灯调控策略，得到问题的有效解决方案。

针对问题 1：考虑路口的不同实际情况，本文根据进入路口道路数量及类型的不同将其进行分类，其中大多数路口可以根据车辆到达路口的时间直接得出最优调控策略，无需额外优化，极大地降低了问题规模。只有少部分的复杂路口需要对调控策略进行优化，以得到最优策略。具体来说，本文采用了基于贪心思想的邻域搜索算法，以单个调度方案的局部最优来逼近全局调度策略的最优解，对于单个路口而言，引入邻域搜索方法，迭代寻找其邻域中的较优解，最终获得单个路口的最优解。基于本题所给数据，在本文所计算得到的方案中，只有车辆 81 不能按时到达终点，其余车辆经过路口的总等待时间也只有 29 秒，最终求得该方案下车辆总得分为 132193 分。

针对问题 2：针对部分道路突发的障碍情况，本文采用 Dijkstra 算法寻找最短替代路径。对于新生成的车辆行驶路径，按照问题 1 的问题求解模式，首先对新的道路网络进行路口分类，对于少量复杂路口沿用问题 1 中迭代贪心邻域搜

索算法，求得车辆路径调整后的交通信号灯全局优化策略。具体而言，本文用“ebf-ha-b-df-ffef”路段替代了“ebf-ffef”路段，经策略优化，得到的新的方案中,只有车辆81不能按时到达终点,车辆经过路口的总等待时间只有32秒，最终求得该方案下车辆总得分为132190分。

1 问题重述与分析

1.1 问题重述

交通信号灯在城市交通中发挥着巨大的作用。随着技术的发展与进步，现如今许多城市的交通信号灯已经互联，可以实现远程控制每一盏灯的每一种信号的时长，能否实现对交通灯的全局优化调控，以提高道路通行能力，减少通行时间？

问题 1：假设全市路网结构、车辆的行程计划、车辆行驶速度等信息均不变且已知，每个路口的交通信号灯都以各自独立控制的固定周期变换信号，研究交通灯全局优化调控策略，分析算法适用的城市路网规模，并针对附录 2 指定的数据，给出具体的调度方案。

问题 2：若某时刻某条道路上突发道路设施故障，需要立即维修，导致该道路无法继续通行，在问题 1 的基础上，讨论此类时间对交通灯调度方案的影响，设计局部调整调度方案的方法，就附录 4 给出的实例信息和具体条件，给出调整后的调度方案。

1.2 问题 1 分析

针对本题全局交通信号灯的调度策略优化问题，一共包含了 8000 个路口和 63968 条道路的信息，优化规模非常大，进行全局优化的难度需要的算力是无法满足的。因此，首先，针对题目中的车辆信息，对问题进行简化，找出有车辆经过的路口和道路，也即需要进行调度的路口。经过统计，200 辆车总共经过了 2462 个路口和 3566 条道路。即我们需要调控的路口数量有 2462 个。对于需要进行调度的路口来说，根据其经过该路口的道路数量和类型的不同，我们采取的调度策略也应该是不一样的，我们将其分为不同的类型进行讨论。即讨论需要进行交通信号灯调度的路口和道路。值得注意的是，每一辆车行驶轨迹的终点路口，因为不需要穿过终点路口，故不需要对其进行调度。路口分类如下：

类型Ⅰ，只有一条道路经过的路口，这种类型的路口，我们只需要让其对唯一的道路一直亮绿灯就可以满足需要。

类型Ⅱ，有多条道路经过的路口，但每一条道路上只有一辆车经过该路口，对于这种路口，我们需要按照不同道路上到达该路口的时间的先后顺序为其安排绿灯即可。

类型Ⅲ，有两条道路经过的路口且有道路上经过的车辆超过一辆。对于这种路口，我们需要按照车辆到达的先后顺序，延长超过一辆车辆的道路的绿灯时间即可。

类型Ⅳ，有超过三条道路经过的路口，且某条道路上来的车辆超过一辆，并且来的时间不连续，对于这种道路，我们需要采取调度优化算法，来建模求解使它的通行时间最短的交通灯调度方案。

表格 1-1 路口类型分布情况统计表

路口类型	路口数量	拟调度方案
总路口	8000	
车辆经过路口	2462	
类型Ⅰ	1868	该道路一直绿灯
类型Ⅱ	534	按照道路上车辆到达顺序绿灯
类型Ⅲ	53	同类型Ⅱ，增大多车辆道路绿灯时间
类型Ⅳ	7	采用优化算法进行优化调度

经过上述分析，我们只需要对类型Ⅳ的节点进行优化分析，来使总的通行时间最短。

针对类型Ⅳ路口的优化，我们采取一种贪心邻域搜索的算法。即对类型Ⅳ路口节点进行单点领域优化，使其达到最优，即该路口节点的等待时间最短。采用贪心领域搜索算法，需要确定类型Ⅳ路口节点的优化顺序。对此，针对通过多于一个类型Ⅳ路口节点的车辆，只有确定前面一个路口的调度方案之后，才可以得到该车辆在路口的通行时间，进而确定后一个类型Ⅳ路口节点的到达时间。通过检索 200 辆车的路径，来确定类型Ⅳ路口节点的优化顺序，然后进行路口顺序排序搜索和绿灯时间长度领域搜索来使单个的类型Ⅳ路口节点最优。在确定了所有路口节点的调度方案之后，计算目标函数，之后再迭代进行贪心领域搜索来达到一个全局的比较好的调度方案。

1.3 问题 2 分析

针对问题 2，当一条道路上发生道路设施故障，会对经过该路段的车辆造成影响，这时需要根据车辆的当前位置和目的地，来对车辆的行驶路径进行调整。在改变车辆的行驶路径的过程中，由于改变了部分车的路径，可能会改变问题 1 中的部分路口节点类型，从而影响其调度方案。针对类型改变的路口节点，按照问题 1 的分析，当改变的为类型Ⅰ、Ⅱ、Ⅲ的路口节点的时候，对整个车辆的通行时间以及路口交通灯的调度产生的影响较小，而对Ⅳ类型路口节点造成影响时，会对路口的调度方案产生较大影响。因此，可以将其分解为两个方面：

第一个方面，受到道路故障影响的车辆的行程中，包含类型Ⅳ路口。这时，如果故障道路在其经过的类型Ⅳ路口之后，则不会对该类型Ⅳ路口造成影响；如果故障道路在其经过的类型Ⅳ路口之前，则会对该类型Ⅳ路口造成影响。

第二个方面，受到道路故障影响的车辆的行程中，不包含类型Ⅳ的路口。但是可能通过对其行驶路径的改变，使路网中，原本不是类型Ⅳ路口的路口改变为类型Ⅳ路口而产生类型Ⅳ路口的拥堵。

综上分析，对于问题 2 的求解，主要是考虑类型Ⅳ路口的产生以及调度。对于新产生的类型Ⅳ的路口，按照问题 1 中的优化分析方法进行调度。

2 模型假设

针对模型构建，题目以及模型本身给出了如下假设：

假设一：城市路网由路口和道路组成，均有唯一的名称。道路连接路口，路口是道路的起点或终点。每条道路均为单向通行，连接两个不同的路口。但是有可能有方向相反的两条道路连接相同的两个路口（对用现实中双向同行的街道）。

假设二：连接两个路口的同行道路仅有一条，因此可以用路口的编号来代替道路的编号。

假设三：车辆行驶速度相同且不受其他车辆影响，通过一条道路的时间等于该条道路的长度。

假设四：忽略车辆的几何尺寸，不考虑车辆排队时对空间的需求。

假设五：每条道路都对应设置一个交通灯，用于控制该条道路上的车辆能否进入该道路终点处的路口。由于道理与交通灯一一对应，因此使用道路名称指代其交通灯，可称为“某道路的交通灯“。

假设六：某路口的交通灯是指，所有以该路口为终点的道路的交通灯。

假设七：交通灯只有红色、绿色两种。红灯表示停止，即，车辆要停在所在道路上，不能穿过该道路终点处的路口。绿灯表示通行，车辆可以穿过该路口，

并驰往以该路口为起点的其他道路。

假设八：某道路的交通灯为红色时，所有到达该道路终点的车辆排队等待绿灯。

假设九：在任何时刻，每个路口最多只有一个交通灯是绿色。当一条道路的绿灯亮起时，只有这条道路上的车辆可以进入该路口（并驰往以该路口为起点的某一条道路），其他道路上准备进入该路口的车辆都必须排队等候。

假设十：当交通灯变为绿灯后，队列中第一辆车可以立即通过该路口，没有延迟，每辆车通过路口的时间均为 1 秒。即，如果绿灯持续 n 秒，则队列中前 n 辆车恰好能够通过路口，其余车辆只能等待下一个绿灯。

假设十一：已知全部车辆的行程计划，即由一系列道路组成的行驶路径，并且不能改变。

假设十二：每辆车的行程经过任一个路口至多一次。

假设十三：初始状态（即 0 时刻），所有的车辆都在各自路径上第一条道路的终点处，该道路的交通灯若为红灯，则等待绿灯；若为绿灯，则准备移动。如果有多辆车在同一条道路的终点处，则 ID 序号较小的车辆排在前面先行。之后，每辆车按照自己的行程计划行驶，遵守交通灯的指示，到达路径中最后一条道路的终点为止，其行程结束，立即被移出，不再考虑。即，车辆不需要在最后一条道路的终点处排队，也不需要进入下一个路口。

假设十四：每个路口至少是一条道路的终点，另外一条道路的起点。

3 模型构建与求解

3.1 问题 1

根据问题分析，我们将需要调度的路口分为了四种类型，而其中最复杂需要优化的是类型Ⅳ的路口，我们将其称为关键路口，即用关键节点表示。下面对于不同类型的路口分别进行建模分析求解。

3.1.1 不同类型路口建模

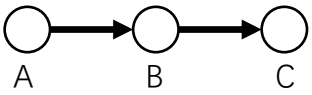

图 3-1 ：类型Ⅰ路口建模示意图

如上图 3-1 所示，B 路口为类型Ⅰ路口，该路口仅有 A→B 道路上有车通行，因此，对于类型 B 路口，我们对它的调度方案就是让其对 A→B 道路上持续出现绿灯，就可以使 B 路口处的通行时间最短，从而使得最终的目标函数最大。

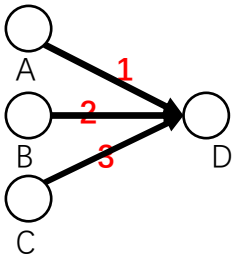

图 3-2: 类型Ⅱ路口建模示意图

如上图 3-2 所示，D 路口为类型Ⅱ路口，该路口在三条道路上分别有 1 辆车辆通行，分别是 A→D（1 号车辆）,B→D（2 号车辆）,C→D（3 号车辆）。假设 1，2，3 号车到达 D 路口的时间分别为（a,b,c）。那么在一个周期内，可以完成

类型Ⅱ路口的调度。假设 a<b<c。那么调度方案为绿灯顺序分别为（A→D,B→D,C→D），绿灯时长为（a+1,b-a,c-b）。对于 a、b、c 可能相等造成的拥堵情况，在后续进行讨论。

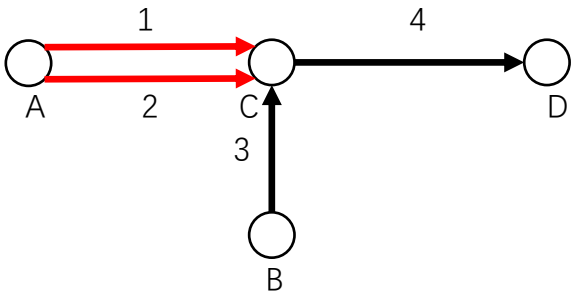

图 3-3. 类型Ⅲ路口示意图

如上图 3-3 所示，C 路口为类型Ⅲ路口，到达 C 路口的道路小于 3，此时按照车辆到达顺序，如果不存在同时到达，则可以调度出所有车辆均不等待的方案。

如果 1，2，3 车辆的到达时间分别记为（a_1, a_2, b）。若$a_1 < a_2$<b,则调度方案为绿灯顺序（A→C，B→C）,绿灯时长（$a_2 + 1$，$b - a_2$）。若a_1 <b<a_2,则调度方案为绿灯顺序（A→C，B→C），绿灯时长为（$a_1 + 1, a_2 - a_1$-1）。

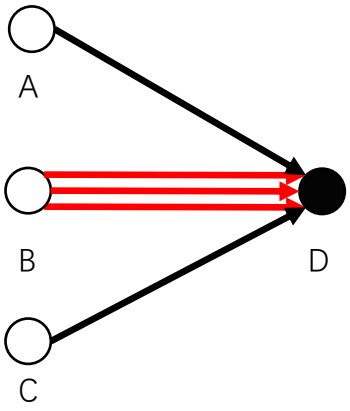

图 3-4. 类型Ⅳ路口示意图

如上图 3-4 所示，D 路口为类型四中需要进行优化调度的关键路口。D 路口是超过 3 条有车通行道路的终点。此时，若同一条道路来车的时间相邻，即它们的到达 D 路口时间间隔内，没有其他道路来车到达 D 路口，则类型Ⅳ路口可以按照类型Ⅱ路口的调度方案来进行调度。当同道路来车时间不相邻时，需要采取优化算法来逐步迭代求出使得所有车辆等待时间最少的调度方案。

3.1.2 调度方案的评价

3.1.2.1 单个路口调度方案的表示

在进行调度方法优化时，需要对现有的调度方案进行评价，设 ID 为$i,(i=0,1,...,I-1)$的路口的前驱路口节点集合为V_i，$V_i=\{v_{i1},v_{i2},...,v_{iN_i}\}$，则路口$i$的调度方案分为两部分，一部分为：一个周期内，指示从各前驱路口行驶而来的车流通行的信号灯变化为绿灯的顺序，可以用前驱路口节点集合V_i的元素的一个排列$[v_{i2},...,v_{iN_i},...v_{i1}]$表示，另一部分为:该元素排列中对应的每一个信号灯绿灯的持续时长排列$[t_{i2},...,t_{iN_i},...t_{i1}]$。

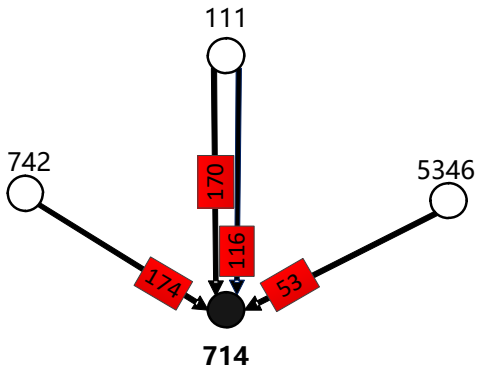

图 3-5：714 路口的车流量情况

如上图所示，ID 编号为 714 的路口的前驱路口节点集合为{742,111,5346}，可依据上述算法构造 ID 编号为 714 的路口的调度方案，包括两个顺序列表，第一个列表可为[742, 111, 5346]，第二个顺序列表可为[1, 7, 1]，表示一个周期内依次为从 742 路口, 111 路口,5346 路口驶来的车流开启绿灯引导通过 714 路口，开启绿灯的持续时间依次为 1s，7s，1s，该路口的调度方案意味着，从初始状态（0 时刻起）, 714 路口指示 742 路口驶来的车流信号灯开启绿灯持续 1s，然后关闭，指示 111 路口驶来的车流信号灯开启绿灯持续 7s，然后关闭，指示 5346 路口驶来的车流信号灯开启绿灯持续 1s，然后关闭，一个周期结束，下一个周期开始。

3.1.2.2 单个路口节点的调度方案的评价

一个好的路口调度方案应使得需要通过该路口的车辆总等待时间最短，如果考虑使得所有路口的车辆总等待时间都最短，那么整个道路的通行时间将可以显著减少，可是在追求某些路口的车辆总等待时间最短时，往往使得另外的一些路口的车辆总等待时间时间显著增加，因此需要从全局角度上来做权衡，为了评价全局调度方案的好坏，因此需要考虑如何评价单个路口的调度方案的好坏，以下给出单个路口调度方案下该路口的车辆总等待时间计算方法。

当给定单个路口的调度方案，即可画出一个周期 T 内，该路口的信号灯持续时间图，如下图所示。

图 3-6：单个路口的信号灯调度时间图

注：该图左图表示一个抽象的单个路口的信号灯调度时间图,右图表示路口 714 的一种信号灯调度时间图。

为了计算路口的车辆总等待时间，需要求出从各前驱路口节点行驶到当前路口的每辆车的到达时刻，从而求出该时刻处于当前信号灯调度窗的位置，推算出每辆车的各自通过当前路口所需要的等待时间，对其求和即可得到当前路口的车辆总等待时间。

3.1.2.3 车辆的到达时间计算过程

首先根据所有车辆的路径信息，挑选出会经过当前路口 i 的车辆集合 $Car^i=\{car^i_{v_{i1},1},car^i_{v_{i1},2},car^i_{v_{i2},1},...,car^i_{v_{iN_i},1}\}$（车辆集合的元素个数大于等于当前路口的前驱节点的数目，因为同一路段可能会驶来多辆车），在假定车辆路径的当前路口 i 的所有在路径中的前驱节点路口的调度方案已经确定的情况下，可以计算出

行驶到当前路口 i 的每辆车的到达时刻集合 $T^i_{arrive}=\{t^i_{v_{i1},arrive1},t^i_{v_{i1},arrive2},...,t^i_{v_{i1},arriven1}\}=\{t^i_{1,arrive},t^i_{2,arrive},...,t^i_{j,arriven},...\}$，每辆车 j 的到达时刻为起点时刻 t_0 加上行驶到当前路口的通行路段时间 t_{pass} 之和，以及花费在路径中当前路口节点之前的所有前驱节点路口上的等待时间 t_{wait} 之和。设 j 车辆路径路口节点 i 之前的 m 个前驱节点顺序列表为 $[v^i_{front01},v^i_{front1},...,v^i_{front\ m}]$，其中 $v^i_{front\ m}\in V_i$，则 T^i_{arrive} 的计算公式如下：

$$t^i_{j,arrive}=t_0+m-1+\sum_{k=2}^{m}t_{pass}(v^i_{frontk-1},v^i_{frontk})+\sum_{k=1}^{m}t_{wait}(v^i_{frontk}),\quad \forall j\in[1,2,...,n]$$

其中 t_0 为 0 时刻，$m-1$ 为过 $m-1$ 个路口所花费的时间（起点路口不计入），$t_{wait}(v^i_{frontk})$ 为 ID 为 v^i_{frontk} 的路口的总等待时间，$t_{pass}(v^i_{frontk-1},v^i_{frontk})$ 为起点为 $v^i_{frontk-1}$，终点为 v^i_{frontk} 的道路的通行时间，$\sum_{k=2}^{m}t_{pass}(v^i_{frontk-1},v^i_{frontk})$ 为路径通行时间之和（第一段道路不计入），$\sum_{k=2}^{m}t_{wait}(v^i_{frontk})$ 为路径上的所有路口总等待时间之和（起点路口不计入）。

3.1.2.4 车辆的等待时间计算过程

在知道当前路口 i 的调度方案和车辆 j 的到达时刻 $t^i_{j,arrive}$ 即可计算车辆 j 的等待时间 $t^i_{j,wait}$，首先判断车辆 j 的到达时刻 $t^i_{j,arrive}$ 处于路口的那一个前驱路口节点对应的绿灯阶段（具体操作为：$t^i_{j,arrive}$ 对周期长度 T 取余数,判断余数处于周期中哪一个信号灯的管控阶段），如果当前路口节点 i 的指示车辆 j 所在道路通行的信号灯处于绿灯阶段且剩余时间大于等于 1s，则车辆 j 不需要等待直接通过，如下图 1 所示:

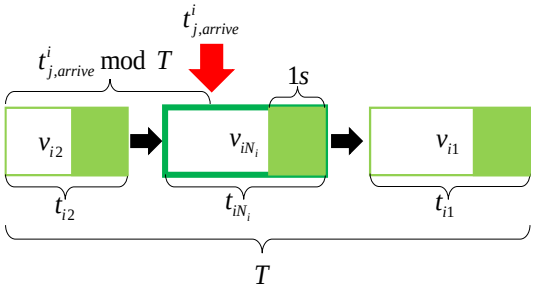

图 3-7：等待时间为 0 的状态图

注：图中红色箭头表示车辆到达时刻，深绿色框矩形代表指示该车辆所在道路绿灯开启时间窗，浅绿色实心矩形代表时间长度为 1s 的时间窗，下图类似。

如果当前路口节点 i 的指示车辆 j 所在道路通行的信号灯在当前周期内还未亮起，则车辆 j 需要等待该信号灯亮起，如下图 2 所示:

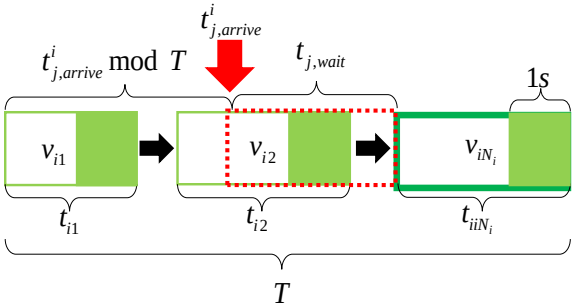

图 3-8：一个周期内等待通过的状态图

如果当前路口节点 i 的指示车辆 j 所在道路通行的信号灯在当前周期内剩余时间不足 1s 或已经亮过，则车辆 j 需要等待到下一个周期内该信号灯亮起，如

下图 3 所示:

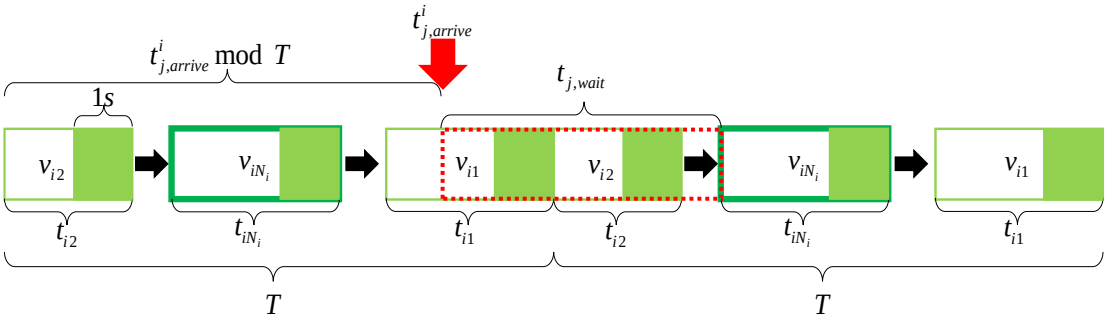

图 3-9：两个周期内等待通过的状态图

3.1.2.5 单个路口节点的总等待时间计算过程

已知每一辆车 j 的等待时间 $t^i_{j,wait}$，可以计算路口节点 i 的总等待时间 t^i_{wait}，具体算法伪代码如下：

Algorithm 1 Single_time：计算路口节点 i 的等待时间 t^i_{wait}

Step1:初始化：

Input 节点 i 的调度方案

$t^i_{wait} = 0$

Step2:计算：

1 计算经过路口节点 i 的车辆路径集合 $Car^i = \{car^i_{v_{i1},1}, car^i_{v_{i1},2}, car^i_{v_{i2},1}, ..., car^i_{v_{iN_i},1}\}$

2 循环迭代

For j in Car^i :

计算车辆 j 到达路口节点 i 的到达时刻 $t^i_{j,arrive}$；

根据车辆 j 的到达时刻 $t^i_{j,arrive}$ 和调度方案计算车辆 j 的等待时间 $t^i_{j,arrive}$；

$t^i_{wait} = t^i_{wait} + t^i_{j,arrive}$

End

Step3:输出结果：

Output t^i_{wait};

以上图所示的 ID 为 714 的路口节点为例子，计算该路口的总等待时间 t^{714}_{wait}，如图 1 所示，有 4 辆车的路径经过 ID 为 714 的路口节点，车辆 ID 分别为：53，116,170,174。其中 53 从前驱节点 5346 所在道路行驶而来，到达时刻为 620s，116 与 170 从前驱节点 111 所在道路行驶而来，到达时刻分别为 0s、173s， 174 从前驱节点 742 所在道路行驶而来，到达时刻为 0s，各车辆的等待时间如下图所示：

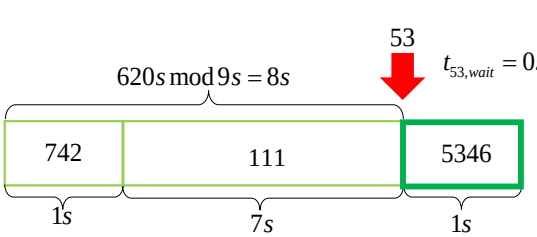

图 3-10：车辆 53 在 714 路口的调度图

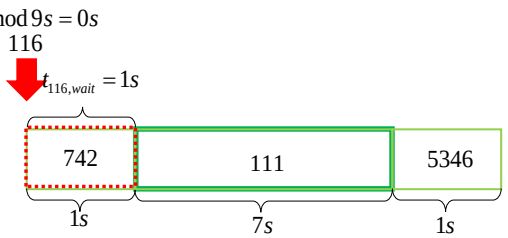

图 3-11：车辆 116 在 714 路口的调度图

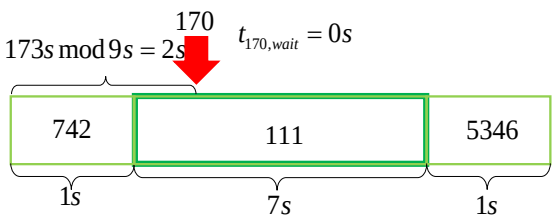

图 3-12：车辆 174 在 714 路口的调度图

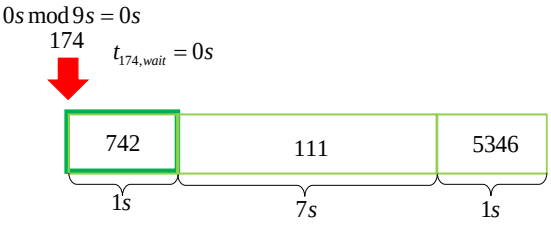

图 3-13：车辆 174 在 714 路口的调度图

由上图可知，ID 为 53，116,170,174 的车辆的等待时间分别为 0s、1s、0s、0s。则路口 714 的总等待时间 $t_{wait}^{714} = 0s + 1s + 0s + 0s = 1s$ 。

3.1.2.6 总调度方案的评价

如果给出所有车辆路径所通过的所有路口节点 $(i \in I)$ 的总调度方案，那么即可推算出所有路口节点的总等待时间 t_{wait}^{i} ，那么每辆车 j 到达终点的时刻 T_j 可由路径上的所有路口总等待时间之和加上路径通行时间之和得出。

$$T_j = m_j - 1 + \sum_{k=2}^{m_j} t_{pass}(k-1,k) + \sum_{k=1}^{m_j-1} t_{wait}^{k}, \quad \forall j \in [1,2,...,200]$$

其中 m_j 是 ID 为 j 的车辆路径的道路总数，k 为 j 的车辆路径的路口 ID，t_{wait}^{k} 为 ID 为 k 的路口的总等待时间，$t_{pass}(k-1,k)$ 为起点为 k-1,终点为 k 的道路的通行时间，$M_j - 1$ 为过 $M_j - 1$ 个路口所花费的时间（最后一个路口不计入），$\sum_{k=2}^{m_j} t_{pass}(k-1,k)$ 为路径通行时间之和（第一段道路不计入），$\sum_{k=1}^{m_j-1} t_{wait}^{k}$ 为路径上的所有路口总等待时间之和（最后一个路口不计入）。

若车辆 j 在 T_j 时刻到达终点，则车辆 j 的得分为

$$M_j = \begin{cases} F + (D - T_j), & T_j \le D \\ 0 \qquad\qquad , & T_j > D \end{cases}$$

其中， F 为车辆按时到达的终点的基本得分，本问题的优化目标为：

$$Max \sum_{j=1}^{200} M_j$$

根据优化目标的结果去评价总调度方案的优劣。

3.1.3 类型Ⅳ路口节点优化调度算法

对于类型Ⅳ路口节点的优化调度，我们采取一种从局部到全局的贪心领域搜索算法，即首先优化调度每一个类型Ⅳ的路口节点，使其达到最优，然后根据调度得到的最优结果去优化调取下一个类型Ⅳ的路口节点。其优化顺序，需要根据 200 辆车的行驶路径来确定，如果 1 辆车先经过 1 个类型Ⅳ路口节点 A，然后经过一个类型Ⅳ路口节点 B，则需要先优化路口 A，再根据 A 的优化结果去优化路口 B。根据数据处理分析，类型Ⅳ共包含 7 个路口，其相关关系示意图如下。其中路口编号 a,b,c,d,e,f,g,h,l,j 分别对应 0,1,2,3,4,5,6,7,8,9。

如图，数字为路口节点编号，黑色实心圆圈代表类型Ⅳ节点，空心圆圈代表非类型Ⅳ路口。实线箭头表示两个路口之间有车辆通过，红线表示有多辆车通过，虚线箭头表示两个路口之间有间隔别的路口。从图中可以看出编号为 226、761 和 45 的类型Ⅳ路口和其他的路口不存在依赖关系，即经过这些路口的车辆的到达路口时间不依赖于其余类型Ⅳ节点的调度方案，因此 226、761、45 号路口可

以直接进行优化调度。而车辆到达 14 号路口之前，先通过了 714 路口，因此 14 号路口车辆的到达路口时间依赖于 714 路口的交通灯调度方案，简记为 14 号路口依赖于 714 路口。同理，1181 号路口的车辆之前虽然相隔了一些其他路口，但其车辆到达时间仍对 714 路口的调度方案存在着依赖关系。而 4114 路口和 4665 路口之间存在着相互依赖关系。因此，在进行优化调度的时候，必须把 4114 路口和 4665 路口看作一个整体进行优化。所以，我们可以得到对类型Ⅳ路口的调度顺序为 226、761、45 可以单独优化调度，然后必须先调度 714 路口，再进行 14 号路口的调度，最后把 4114 和 4665 两个路口作为一个整体进行优化调度来得到所有类型Ⅳ路口的调度方案。

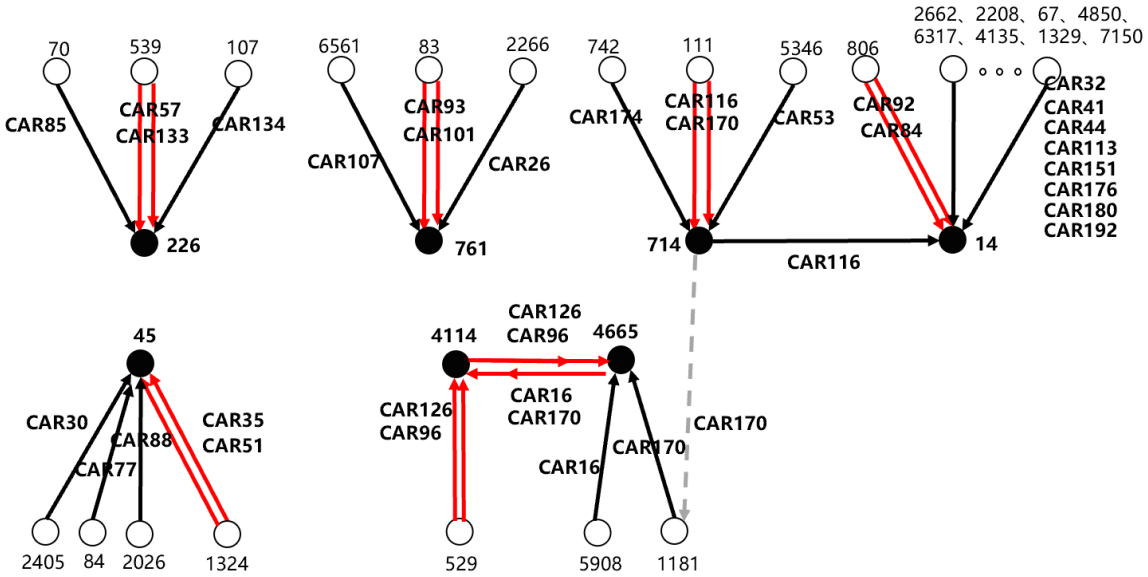

图 3-14. 路口调度关联图

调度算法伪代码：

Step1. 针对类型Ⅳ路口节点 A,假设其有 k 条道路经过路口节点 A，假定其前向路口节点为（$node_1, node_2 \dots node_k$）,用$(node_1, node_2 \dots node_k)$表示道路，则初始化调度方案为顺序$(node_1, node_2 \dots node_k)$，绿灯长度为$(t_1, t_2 \dots t_k)$.将初始解定义为$plan_0$.方案包括两个部分，绿灯道路顺序，和绿灯时间长度。
Step2. 定义长度l的领域为$(t-1, t, t+10)$,顺序的领域为A_k^k全排列中的一种.由此可以得到$plan_0$的领域$\{plan_0'\}$ $\{plan_0'\} = \{[node_x, node_y \dots node_z],[t_x, t_y \dots t_z]\}$
Step3. $TWAIT_0' = single\_time(t_{arrive}, PLAN_0^i)$ 得到$\{plan_0'\}$内的所有解的等待时间，将等待时间最小的解定义为$plan_1$. $$plan_1 = argmin_{\{PLAN_0'\}}\{TWAIT_0^i\}$$
Step4. 若$plan_1. = plan_0$.则循环结束，最优解为$plan_1$。若不等，则令$plan_0 = plan_1$.回到 S2。

补充：假设一个类型Ⅳ路口节点有 3 条道路需要调控，其道路前向路口分别为i, j, k。则用$node_i, node_j, node_k$表示道路编号。用t_i, t_j, t_k表示路口绿灯持续时间。则方案 $plan_0 = ([node_i, node_j, node_k],[t_i, t_j, t_k])$。

领域 $\{PLAN_0'\} = \{[node_x, node_y, node_z],[t_x, t_y, t_z]\}$

其中，$[node_x, node_y, node_z]$是顺序$[node_i, node_j, node_k]$ 的一种全排列

t_x=t_i + [−1: 10]中的某一个整数

t_y=t_j + [−1: 10]中的某一个整数

t_z=t_k + [−1: 10]中的某一个整数

算法流程图如下：

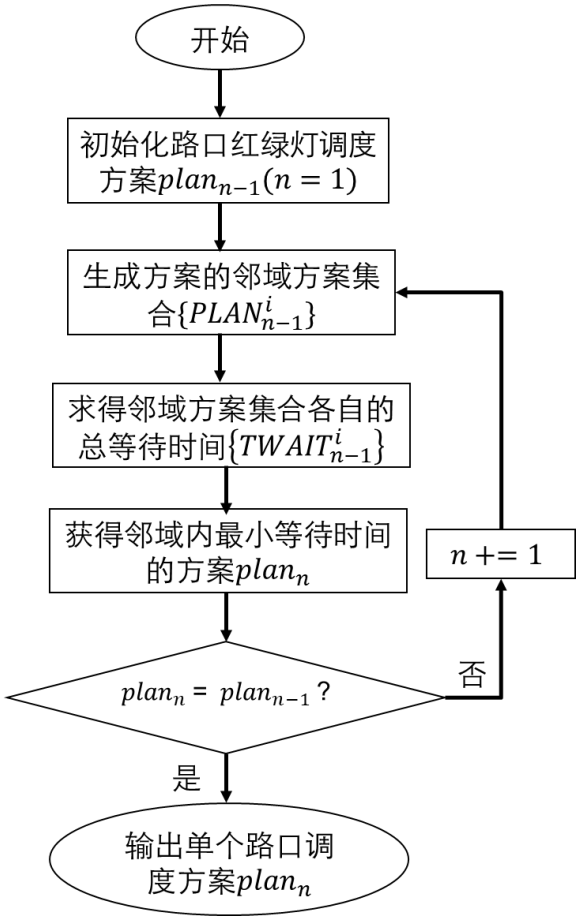

图 3-15 邻域搜索算法流程图

3.1.4 迭代贪心邻域搜索算法

由以上不同类型路口节点的建模分析，我们可以得到模型的总的算法优化思想和优化过程。

首先，基于路口类型的分析，对于Ⅰ、Ⅱ、Ⅲ类型的路口，我们不需要进行搜索更新就可以得到其最优的调度方案，进而可以得到车辆在这类路口的通行时间。

其次，有了其余三类路口的调度方案之后，我们可以得到车辆准确的到达类型Ⅳ路口的时间，由此通过贪心的邻域搜索可以得到单个类型Ⅳ路口的局部最优调度方案，即达到经过类型Ⅳ路口的最短通行时间。

最后，单个类型Ⅳ路口的最优并不一定是全局所有车辆通行时间以及目标函数 M 的最优，因此我们需要算出目标函数值 M 之后，再返回进行新的类型Ⅳ路口的贪心邻域搜索得到新的调度方案之后得到新的 M 值。

进行上述迭代 n 轮，得到 M 值最小的类型Ⅳ路口的调度方案，即为全局最优解。

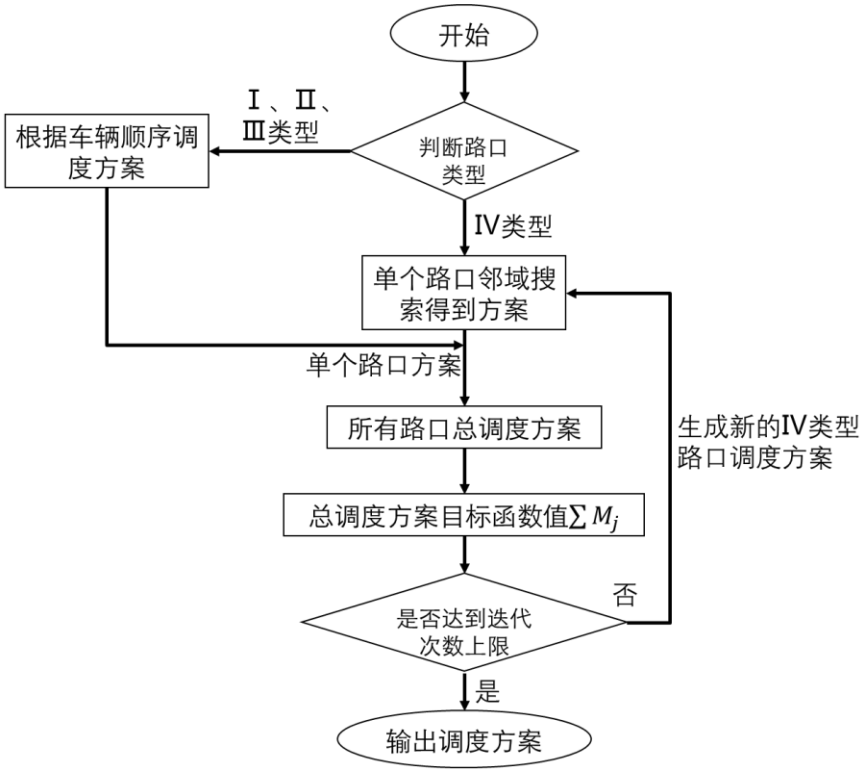

图 3-16. 迭代贪心邻域搜索算法流程图

上图是迭代贪心邻域搜索算法的流程图，流程图和算法思想是一致的，即首先得出需要进行贪心邻域优化的路口Ⅳ，然后优化计算目标函数值，进行多轮迭代，得出使得目标函数值最小的调度方案。

3.1.5 道路拥堵情况处理补充

经过上述分析与建模，在所有路口中，可能产生拥堵的为Ⅱ、Ⅲ、Ⅳ类型路口，针对不同情况的拥堵，采取不同的处理方法，其主要有以下三种拥堵情况。

① .起始道路拥堵，共有 6 个路口为车辆起始通行路口拥堵，分别有两辆车在起始时刻等待通行，经统计，这些车辆在后续路径不会经过类型Ⅳ的路口，所以它们的通行并不会对后续调度产生大的影响。因此，让两辆车中总路程较长的车辆先行，来保障其总通行时间小于时间阈值。

② .在类型Ⅳ路口拥堵，在类型Ⅳ路口的拥堵，无法通过调度来使其等待时间为 0，只能通过优化方法，根据车辆到达路口时间来优化出使其总的等待时间最小的调度方案。

③ .在中间路口非类型Ⅳ路口的拥堵，这种情况下的拥堵是由同一个路口中，不同道路的车辆同时到达该路口所致，概率较低。在初步调度优化中进行忽略，假设其不存在拥堵，在得到调度总方案之后，对可能拥堵的路口进行验证，如果存在拥堵，再对总的调度方案进行微小的调整。

3.1.6 模型求解结果

通过上述局部到全局的贪心邻域搜索算法，我们得到了有关的 2462 个路口节点的调度方案，调到方案部分如下：（其中，路口编号中数字 0,1,2,3,4,5,6,7,8,9 对应路口编号 a,b,c,d,e,f,g,h,i,j。经过统计，在类型Ⅳ的 7 个路口中，仅有两个路口存在车辆等待时间，其节点编号为 714 和 14，这两个路口的道路车辆等待时间，和存在着等待红绿灯的车辆编号和等待时间以及相关路口的调度方案如下：

表格 3-1 路口 714（hbe）调度方案

	道路编号	绿灯时长(s)
路口 714(hbe)调度方案	hec-hbe	1
	bbb-hbe	7
	fdeg-hbe	1

表格 3-2 路口 714 车辆等待时间及相关道路

关键节点	起点路口	车辆编号	等待时间(s)
714	111	116	1
	111	170	0
	742	174	0
	5346	53	0

表格 3-3 路口 14(be)调度方案

	道路编号	绿灯时长(s)
路口 14(be)调度方案	gdbh-be	4
	gh-be	1
	ebdf-be	1
	bdcj-be	2
	cggc-be	2
	ccai-be	2
	iag-be	1
	eifa-be	1
	hbe-be	1

表格 3-4 路口 14 车辆等待时间及相关道路

关键节点	起点路口	车辆编号	等待时间(s)
14	2662	32	0
	2208	41	1
	67	44	3
	806	84	10
	806	92	3
	4850	113	0
	714	116	5
	6317	151	0
	4135	176	4
	1329	180	1
	7150	192	1

表格 3-5 路口 226（ccg）调度方案

路口 226(ccg)调度方案	道路编号	绿灯时长(s)
	fdj-ccg	1
	ha-ccg	4
	bah-ccg	1

表格 3-6 需要等待交通灯的车辆编号及等待时间

车辆编号	路线上的总等待时间(s)
41	1
44	3
84	10
92	3
116	6
176	4
180	1
192	1

综上，经过模型计算，仅有上述 8 辆车在行驶过程中需要等待交通灯，其等待交通灯的时间如上表 3-6.其中车辆编号为 81 的车辆不能在规定的时间 858 秒内到达终点，其余 199 辆车均能顺利到达终点。具体的路口交通灯调度方案见附件。总的目标函数值 M = 132193。

3.1.7 算法适用城市路网规模分析

通过上述模型的建立和求解过程可知，算法可以适用较大的城市路网规模。

首先，对于路口类型的分类方面，我们划分的四类路口涵盖了所有可能出现的路口情况，及时随着路网规模的进一步增大，它的路口类型并不会变多。因此，算法的路口分类覆盖面广且可扩展性良好。

其次，算法对于算力的需求方面，由于合理的对路口类型进行的划分，我们实际需要优化调度的路口规模远远小于总的路网规模。我们采用的从局部到全局

的贪心领域优化算法，相较于全局的优化算法来说，可能结果不一定达到全局调度的最优解，但是会得到一个相对满意的路口交通信号灯的调度方案。并且，我们提出的算法的复杂度相对较低，对算力的需求更小，相较更加适用于真实的大规模城市路网。

最后，对于算法流程来说，规模的增大不会对算法的运算流程造成影响。而真正影响的只是不同类型路口的数量，因此算法流程可以适用于规模大小不等的城市路网。

综上，所提出有关城市交通信号灯的从局部到全局的贪心领域搜索算法涵盖了所有城市路口类型，算法流程稳定，对计算机算力需求较小。因此，既可以适用于小规模的城市路网，也可以适用于大规模的城市路网。

3.2 问题 2

3.2.1 Djistra 算法求解最短路径

首先针对受到故障道路影响的车辆，要更改其路径为距离目的地最短的路径。因此，我们选择了 Djistra 路径规划算法来求解最短路径。Dijstra 算法是运用贪心的策略，从源点即“ebf”开始，不断地通过相联通的点找出到其他点的最短距离基本思想是：设置一个顶点的集合 s，并不断地扩充这个集合，一个顶点属于集合 s 当且仅当从源点到该点的路径已求出。开始时 s 中仅有源点，并且调整非 s 中点的最短路径长度，找当前最短路径点，将其加入到集合 s，直到终点在 s 中。在本题中终点 s 为各自车辆的终点。

值得注意的是，在问题 1 中车辆经过的路段当中以“ebf“为起点的路段，除了故障路段之外，并没有其他路段。因此，在求解最短路径的时候，我们需要对所有的超过 6 万条道路进行搜索来得到最短路径。具体的 Djistra 算法流程见参考文献。

由于题目中未明确给出最短路径，是受影响车辆到故障节点的下一个路口，还是到其行程终点的最短路径。因此，基于尽可能少的改变其原有行程的原则，把故障路段起点到故障路段终点的最短最短路径求出作为其新的路径。

经过算法求解，得出故障路段“ebf-ffef“的最短路径为:

“ebf-ha-b-df-ffef”

将这段路径作为受影响车辆中“ebf-ffef”的替代路径，然后进行优化求解新的路口调度方案。

3.2.2 迭代贪心邻域搜索算法

与问题 1 类似，在得到新的车辆的行驶路径之后，我们根据其受到新路径影响的节点，可以按照问题 1 中的迭代贪心邻域搜索算法来对新的车辆路径的路口的调度方案进行求解。

经过统计，受到故障道路影响的车辆一共有三辆，其车辆编号分别为 35，16，50.根据问题 1 求解得到的路口交通灯调度方案，它们到达故障路段的起点的时间均超过 30 秒，因此这三辆车的行驶路径均会受到影响。受影响车辆经过的类型Ⅳ的节点路口的信息见下表。

表格 3-7 经过故障路口车辆信息

车辆编号	经过类型 IV 节点编号
35	45
16	4665、4114
50	无

经过建模分析，通过对上述三辆车的行程的调整，一共形成了 3 个新的类型Ⅳ的路口，编号分别为（“ha”，“b”，“df”）.对这三个Ⅳ类型路口的交通灯调度方案以及相应道路的车辆的等待时间如下。最终求得的车辆总的等待时间为 32 秒，车辆得分为 132190 分。

表格 3-8 路口 70（ha）调度方案

	道路编号	绿灯时长
路口 70（ha）调度方案 (总等待时间为 0)	hbjd-ha,	357
	hhfb-ha	672

表格 3-9 路口 1（b）调度方案

	道路编号	绿灯时长
路口 1（b）调度方案 (总等待时间为 0)	hifd-b	2
	eigi-b	6
	bahf-b	2
	fhg-b	5
	daei-b	10

表格 3-10 路口 35（df）调度方案

	道路编号	绿灯时长
路口 35（df）调度方案 (总等待时间为 3)	hda-df	1
	ibc-df	1
	fdfi-df	15
	bd-df(等待时间为 3)	1
	ddf-df	2

4 总结与展望

4.1 总结

本文主要从简化问题的规模，使用局部贪心算法来迭代得到全局最优结果的方向出发，在对算力要求相对不高的前提下得到了一个合适的解。

首先，考虑涉及到的路网路口的分类，针对不同的路口的采取不同的调度方法，而不是对所有类型的路口进行统一的全局优化。通过这个方式，我们在很大程度上缩减了问题的规模。

其次，针对需要采取优化的路口类型，考虑其间的相对依赖关系，对依赖等级最低的路口先进行优化，并且针对暂不需要优化的路口，假设其车辆等待时间为 0，这样保证了我们的优化过程不会出现前后矛盾，也更好的适用了我们的贪心邻域搜索算法。

最后，针对问题 2 中，受到影响的车辆，我们采用 Djistra 算法首先求解出了其最短路径，然后运用问题 1 中算法中受到影响的Ⅳ类型路口节点进行了优化

求解，而不是对全部节点的求解，这在很大程度上保证了计算的准确与高效。

4.2 不足与展望

本题的路网交通信号灯的调度问题规模相对较大，问题相对复杂。采用传统的全局优化算法难以计算，本文对路口的分类建模虽然大大的降低了问题的规模和计算量，但在局部优化过程中，对于II和III路口类型车辆等待时间为0的假设可能不是最合适的，未来可以考虑其等待时间为一个可扩展的区间，从而对于路口输入的到达时间是一个时间区间，这可能会更符合实际情况。对于第二问的求解，是按照最短路径进行调整，未考虑到其他道路的拥堵情况，后续需要更具其余道路的拥堵情况来调整车辆的行程，更符合现实也更加高效。

参考文献

[1] 司守奎,孙玺.数学建模算法与应用[M]. 北京:国防工业出版社,2013.
[2] 李大潜,姜启源.UMAP 数学建模案例精选 1[M]. 北京:高等教育出版社,2015.
[3] 张莹，自适应交通信号调度算法的建模、设计与实现[D],昆明：昆明理工大学，2012.
[4] 孙文瑜. 最优化理论与方法[M]. 北京: 科学出版社,1997.
[5] 《运筹学》教材编写组. 运筹学[M]. 北京: 清华大学出版社,2005.
[6] 董霖.MATLB 使用与详解[M].北京:科学出版社.
[7] 李汗龙,缪淑贤,韩婷.Mathematica 基础及其在数学建模中的应用[M]. 北京:国防工业出版社,2013.

附录

```
# 找出主要关键节点
def endId2(pro_node) -> list:
    KEY_NODE = []
    for i in pro_node:
        a = pro_node[i]  # 前驱节点列表
        b = dict(Counter(a))
        for j in b:
            if b[j] > 1 and len(pro_node[i]) > 3:
                KEY_NODE.append(i)
                break
    return KEY_NODE

# 贪心邻域搜索算法
def main_search_plan(node_name):
    from random import shuffle
    # 测试程序
```

```
        import itertools
        NodesPathsTstart = count_arrivetime_ofKEY()
        # print('测试程序:')
        TDict = NodesPathsTstart[node_name]
        NodesList = list(TDict.keys())
        TimeList = [1] * len(TDict)
        # 局部最佳方案
        Twait_opt, TWaitDict, flag = NodesTotalWaitTime(TDict, NodesList,
TimeList)
        NodesList_opt = NodesList.copy()
        TimeList_opt = TimeList.copy()
        TWaitDict_opt = TWaitDict.copy()

        if Twait_opt == 0:
            print('关键节点编号', node_name, '顺序：', NodesList_opt, '长度:',
TimeList_opt, 'Twait:',
                  Twait_opt)
            for iii in NodesList_opt:
                print('节点编码:', N2S(iii))
            return NodesList_opt, TimeList_opt, Twait_opt, TWaitDict_opt, flag

        # 开始搜索
        for times in range(100):  # 局部搜索
            TimeList_opt_last = TimeList_opt.copy()
            NodesList_opt_last = NodesList_opt.copy()
            for num in itertools.permutations(NodesList, len(NodesList)):
                NodesList_new = list(num)
                for i in range(-1, 10):
                    for j in range(len(TimeList_opt)):
                        TimeList_new = TimeList_opt.copy()
                        TimeList_new[j] += i
                        if TimeList_new[j] > 0:
                            Twait_new, TWaitDict, flag =
NodesTotalWaitTime(TDict, NodesList_new, TimeList_new)
                            if Twait_new < Twait_opt:
                                NodesList_opt = NodesList_new.copy()
                                Twait_opt = Twait_new
                                TimeList_opt = TimeList_new.copy()
                                TWaitDict_opt = TWaitDict.copy()
                                if Twait_opt == 0:
                                    print('关键节点编号', node_name, '顺
序：', NodesList_opt, '长度:', TimeList_opt, 'Twait:',
```

```
                                                    Twait_opt)
                                                for iii in NodesList_opt:
                                                    print('节点编码:', N2S(iii))
                                                return NodesList_opt, TimeList_opt,
Twait_opt, TWaitDict_opt, flag
                if NodesList_opt == NodesList_opt_last and TimeList_opt_last ==
TimeList_opt:
                    break

            print('关键节点编号', node_name, '顺序：', NodesList_opt, '长度:',
TimeList_opt, 'Twait:', Twait_opt)
            for iii in NodesList_opt:
                print('节点编码:', N2S(iii))
            return NodesList_opt, TimeList_opt, Twait_opt, TWaitDict_opt, flag


    # 计算单点的等待时间（计算节点的所有边的总等待时间）
    def NodesTotalWaitTime(TDict: dict, NodesList: list, TimeList: list):
        from itertools import accumulate
        TCircle = sum(TimeList)
        TimeList = [0] + list(accumulate(TimeList))
        Twait = 0
        TValuesSorted = sorted([j for i in list(TDict.values()) for j in i])
        TWaitDict = {}
        flaggg = 0
        for I in TDict:
            TWaitDict[I] = [0] * len(TDict[I])
        for I in TDict:
            if len(TDict[I]) == 2:
                indexList = []
                for t in TDict[I]:
                    indexList.append(TValuesSorted.index(t))
                if abs(indexList[0] - indexList[1]) == 1:
                    Twait = 0
                    flaggg = 1  # 关键节点，来自同一路口的多辆车到达时
间相邻
                    print('关键节点，来自同一路口的多辆车到达时间相邻，
调度方案可直接给出，无需优化')
                    return Twait, TWaitDict, flaggg
            for j in range(len(TDict[I])):
                TStart = TDict[I][j]
                for i in range(len(NodesList)):
```

```
                if NodesList[i] == l:
                    flag = i
            Remainder = TStart % TCircle
            for i in range(len(TimeList) - 1):
                if (TimeList[i] <= Remainder <= TimeList[i + 1] - 1):
                    if i < flag:
                        twait = TimeList[flag] - Remainder
                    elif i == flag:
                        twait = 0
                    else:
                        twait = TCircle - Remainder + TimeList[flag]
                    TWaitDict[l][j] = twait
                    break
            Twait += twait
    return Twait, TWaitDict, flaggg
```