题 目 动车组检修方案

摘 要：

随着铁路发展，动车逐渐成为大众首选的出行方式之一，而随着运力的加强与需求的增加，确保动车行驶安全成为了重中之重。为此，需要通过动车组运用所对动车进行检修。动车组运用所是负责对动车进行检修、养护等工作的场所，全国已建成的动车运用所已超过 50 个。动车组的检修根据行驶情况被划分成不同检修等级，不同等级对应不同的工序。

通过安排动车检测调度，有助于最大化合理调度动车运力。并得到动车准确出站时间。有助于乘客合理选择车次进行出行。

本文通过问题中所给的动车组安排规则，对动车组检测方案进行建模。

本文内容主要分为三个部分：

第一部分是已知动车组的检修工序，假设第一辆动车组抵达动车运用所时，所有检修车间都是空闲的，且车间之间的转换时间忽略不计动车运用所某 12 小时内每十五分钟来 1 辆待检修的动车，按照目前的车间设置，计算出维修完所有这些动车组总共需要多长时间。通过贪心算法结合多机调度问题思想，让所有的车能够检修便进行检修。如果检测工序都已经满了则进行等待，直到检测a工序有空位，则进入检修。利用这种方式，最大化利用各个检修工序资源，做到最优化调度。

第二部分是

第三部分是

**关键词**：贪心算法；多机调度问题；Python3；

# 1 问题重述

## 1.1 问题背景

随着铁路发展，动车逐渐成为大众首选的出行方式之一，而随着运力的加强与需求的增加，确保动车行驶安全成为了重中之重。为此，需要通过动车组运用所对动车进行检修。动车组运用所是负责对动车进行检修、养护等工作的场所，全国已建成的动车运用所已超过 50 个。动车组的检修根据行驶情况被划分成不同检修等级，不同等级对应不同的工序。

通过安排动车检测调度，有助于最大化合理调度动车运力。并得到动车准确出站时间。有助于乘客合理选择车次进行出行。

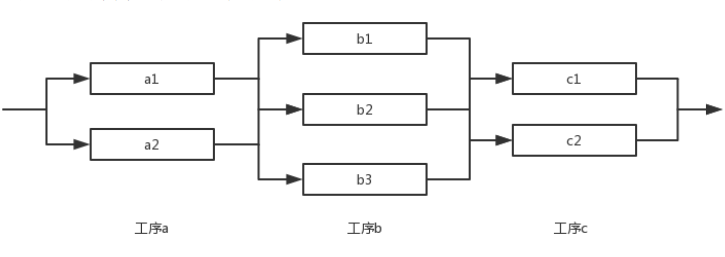


图1.1 检修工序示意图

已知动车检修要经过如图1.1所示的三道检修工序。为了保证在通行安全的情况下使得动车运力最大化，在不同条件下，合理调度检修工序资源。

## 1.2 本文所需完成的任务

（1）在给定车间设置以及各个检修工序数目以及所需要的时间条件下，已知动车在未来12小时内，每隔15分钟有一辆待检测的动车进站，最大化利用各个工序给出部署方案，计算出该检车部署条件下，检修一共花费的时间并结合数据分析。

（2）与问题1不同，由于实际情况下，不同类型的动车组每个工序需要花费的时间是不一样的。请根据附件中附表一到达动车运用所的动车信息，计算维修完这些动车的总时间。

（3）更详细的考虑不同检修等级的情况下，根据列车的行驶时间、历程和检修周期，动车组的检修被划分成不同检修等级 I~V，如表 3 不同的检修等级对应不同的工序组合。 工序 d 与 e 分别设有车间3 和 2 个，相同工序不同车间的耗费时间相同。 表 4 为不同动车类别的每个工序需要耗费的时间。 根据附表二的到所列车信息，计算检修完这些列车需要的总时间。

# 2 问题分析

题目要求对不同条件情景下的任务，通过合理安排动车检测调度，最大化合理调度动车运力。建立模型部署任务方案，计算得到不同条件下的检修时间。并给出分析理由与过程。

## 2.1 问题一分析

该问题主要是假定12小时内每隔15分钟会有一辆动车送来检修，按照题目的检修工序以及各个工序车间数量和检修基本数据（如图表1所示），部署出最优的检修计划，并计算出检修一共需要花费的时间。

通过贪心算法结合多机调度问题，假定动车在工序a中有空余的情况下立刻进入检修。并且在下一阶段工序有空余，且本阶段检修任务已完成的情况下，立刻进入下一段进行检修。

## 2.2 问题二分析

该问题针对于实际情况中，不同类型的动车在各个检修工序所需的时间不同，并且按照附表一所给的数据，会有各个类型的动车在各个时间到站，计算附件一中所有的动车到站检修所需要的时间。

## 2.3 问题三分析

# 3 模型假设

1. 假设第一辆动车组抵达动车运用所时，所有检修车间都是空闲的，  
   且车间之间的转换时间忽略不计。
2. 假设动车组按 a→b→c 顺序进行检修，完成一个检修工序后驶入下一个有空闲位置的车间进行下一个检修工序，若下一个工序所有车间都处于占用状态，则动车组需要在上一个车间中等待。各个检修任务不能同时进行。
3. 假设问题中所给出的数据均能反应真实情况。
4. 假设各个车间性能相同，无任何差异。
5. 每个工序不间断，检修完成之后才可以进行下一道工序。
6. 机器无故障等。

# 4 符号说明

|  |  |
| --- | --- |
| 名称 | 说明 |
| a | a工序中各个车间已经检修动车时间 |
| b | b工序中各个车间已经检修动车时间 |
| c | c工序中各个车间检修动车时间 |
| waiterCar | 等待进入a工序的车的数量 |
| addtime | 12小时之后剩余车辆检修所需的时间 |
|  |  |

# 5 问题一求解

## 5.1 检修方案确定

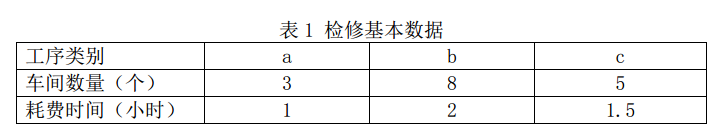


图5.1 检修基本数据表图

参考实际情况下对该任务的解决方式。由于该问题下，各个动车所需要在三种工序下检修所花费的时间一致。根据问题1中的表1数据如图5.1所示，我们对各个工序各个车间中已有车辆检修时间存入数组a,b,c中。并根据题意，我们得知如下规定：

在12个小时之内，每隔15分钟便会有一辆车送来检修，如果a中有空余车间，便可将该车拿来检修，如果没有，则需要等待。当a中车间有检修时间超过1小时的车辆，并且b中有空余车位，即可将该车辆送入b中。对于b同理，当b中车间有检修时间超过2小时的车辆，并且c中有空余车位，即可将该车辆送入c中。当c中车间有检修时间超过1.5小时的车辆，即可将该车辆送出。增加空余车间。

为了最大化利用各个工序以及各个车间，我们结合贪心算法，假设任何车辆只要满足在该工序已检修完成，且下一工序已有空位，便立即送入下一工序进行检修的策略。其中，顾名思义，贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优解，当然希望贪心算法得到的最终结果也是整体最优的。在本问题中，即使贪心算法蹦年得到整体最优解，其最终结果确实最优解的很好近似解。

## 5.2 检修时间计算

根据5.1中的检修方案，将该问题中的检修时间计算分成12小时之内和12小时以外的两个部分。

在12个小时之内，每隔15分钟便会有一辆车送来检修，如果a中有空余车间，便可将该车拿来检修，如果没有，则需要等待。因此，首先计算在12小时之内，每15分钟waiterCar增加1，首先对c中有车车间所有车辆的检修时间增加15分钟，如果c中车辆检修时间大于等于1.5小时，则让该车间的检修时间重置为0，代表该车辆已检修完成顺利出站；接着对b中有车车间所有车辆的检修时间增加15分钟，如果该车检修时间已大于等于2小时，并且c中有空余车位，则可将该车间检修时间重置为0，并在c中找到任意一个空余车间，将该空余车间检修时间设置为1e-10，代表着b中该车位已检修完成，准备进入c车位，在c车位已占的一个检修位置，b中下一辆车不得进入该位置。如果该车检修时间已大于等于2小时，但是c中无空余车位，则继续等待；最后对a工序车间进行分析，如果该车间内有空余位置，且waiterCar的数目大于0，则将等待车辆送入a车间，此时waiterCar的数目将减一。并且对a车间内所有有车车间（包括刚才的进入的等待车辆）检修时间增大15分钟，如果该车辆检修时间大于1小时，并且b中有空余车间，则将该车辆检修时间重置为0，并在b中找到空余车间，将其检修时间设置为1e-10。此举与之间同理。

按照以上思路，得到程序流程图5.2所示。

将以上思路逻辑通过Python3进行编程，具体程序见附录1所示。得到12小时，waiterCar = 12； a工序中各个车间的检修时间为[0.25, 0.75, 0.75]，b工序中各个车间的检修时间为[1.2500000001, 0.7500000001, 0.7500000001, 0.2500000001, 1.7500000001, 1.7500000001, 0, 0]，c工序中各个车间的检修时间为 [0.2500000001, 1.2500000001, 0.7500000001, 0, 0.7500000001]。即代表还有一共25辆动车需要检修完。

在12小时之后，除了等待的waiterCar之外，不再有新的车辆进入排队检修。因此，将剩余的25辆动车按照同样的规则计算，每隔15分钟将各个车间内有车辆的检修时间增加15分钟，只是不再有waiterCar增加1，直到所有的车辆检修完，a，b，c三个数组总和为0。并且waiterCar = 0。经过附录1中的程序计算得到addtime = 4小时+4.25小时 = 8.25小时。因此，最后一共需要花的时间即为20.25小时。

用同样规则计算剩余车辆全部出站需要的时间，并与12小时相加，输出所需总时间。

计算出12小时时，waiterCar的数目，并计算此时a、b、c各工序中车检修的时间。

判断a工序中的动车是否检修够时间，并且b工序中有工车间，若有，该车进去该空车间。若无则继续等待。

判断b工序中的动车是否检修够时间，并且c工序中有工车间，若有，该车进去该空车间。若无则继续等待。

判断c工序中的动车是否检修够时间，若够，则直接离开检修组。

判断a工序中是否有空车间，若有，则waiterCar进去该车间。直到车间为满或waiterCar = 0

12小时内，waiterCar数目每十五分钟+1

图5.2 代码流程图

# 6 问题二求解

## 6.1 模型算法建立

由于问题二动车的到站时间精确到分钟，如果化成以小时为单位的数值型时间会出现数位精度问题，为了避免这个问题，提高计算以及数据的直观感，先把附表一大到站时间化成以分钟为单位，且定义时间00:00为0，每分钟以此递增一单位，具体的到站时间点如下表：

表6.1 动车到站时间表

|  |  |  |
| --- | --- | --- |
| 到站时间 | 动车类别 | 数值型时间 |
| 0:16 | CRH2 | 16 |
| 0:47 | CRH5 | 47 |
| 1:22 | CRH2 | 82 |
| 2:00 | CRH6 | 120 |
| 2:21 | CRH3 | 141 |
| 3:02 | CRH6 | 182 |
| 3:31 | CRH2 | 211 |
| 3:59 | CRH5 | 239 |
| 4:01 | CRH3 | 241 |
| 4:27 | CRH3 | 267 |
| 5:09 | CRH6 | 309 |

此外，对应题目的表二（含有动车类别的检修数据）也转换成以分钟为单位的数值，具体如下表：

表6.2 动车类别的检修耗时数据

|  |  |  |  |
| --- | --- | --- | --- |
| 动车类别 | **工序类别** | | |
| a | b | c |
| CRH2 | 60 | 120 | 90 |
| CRH3 | 48 | 144 | 30 |
| CRH5 | 78 | 150 | 90 |
| CRH6 | 60 | 162 | 18 |

先看工序a的方案规划。四种不同的动车类别对应不同的检修时间，动车的初始到站时间即为动车进入工序a的时间。把每一分钟的时间看成离散变量，工序a可以在三个车间完成，分别记为。在第分钟时，车间的状态记为，当时，意味着在在第分钟时，车间的状态为繁忙；反之值为1时表示闲置状态。假定在分到站的动车，会在分开始检修，以第一辆到站的CRH2类别动车为例，动车在16分到站，则在17分开始检修，经过60分钟的检修后，在76分检修完毕。

与问题一的思路相似，对时间进行遍历，并采用贪心算法对到站的每个动车进行规划，如果在动车到站时，某个车间的状态为闲置时，则送到对应的车间检测。如果无发现闲置车间，推迟改动车的到站时间，推迟幅度为1分钟，直至有闲置车间为止。如果在推迟到站的过程中遇到下一辆本应到站的动车，则让下一辆本应到站的动车依次推移，直至发现闲置车间为止。记第辆到站的动车在时刻的状态为，其中，当值为0时，即在时刻时第辆动车未到站检测，反之则表示已到站检测中或已完成检测。

综合上面的逻辑，当所有动车都完成工序a时，有如下状态条件：

其中方程组第一等式表示在时刻时所用动车都已经到站检修；第二个等式表示完成工序a的3个车间都在闲置状态。合起来则表示动车都完成工序a，求解目标则是找出最小值，即用最短的时间完成工序a。

与问题一有所不同的不止是到站时间无固定规律，而且不同类型的动车耗时不同，因此当在某类型的动车在时刻到站进入车间检修时，有

其中为机车类别名称最后一个数字在工序a中的检修耗时，其数值对应表2.2动车类别的检修耗时数据；表示第辆车在时刻进站检修。

利用贪心算法，可以求得工序a中每辆汽车的到站时间以及检修完时的出车间a的时刻。由于车间之间的转换时间忽略不计，因为可以把工序a的出站时间看成工序b的进站时间，并重复工序a的算法，直接求解出完成工序c后所有动车的出站时间。工序c最晚的出站时间剪去工序a最早的进站时间为题目要求的最短总耗时。

## 6.2 模型算法求解

应用python程序编写贪心算法，以工序a为例，具体的算法流程如下图：

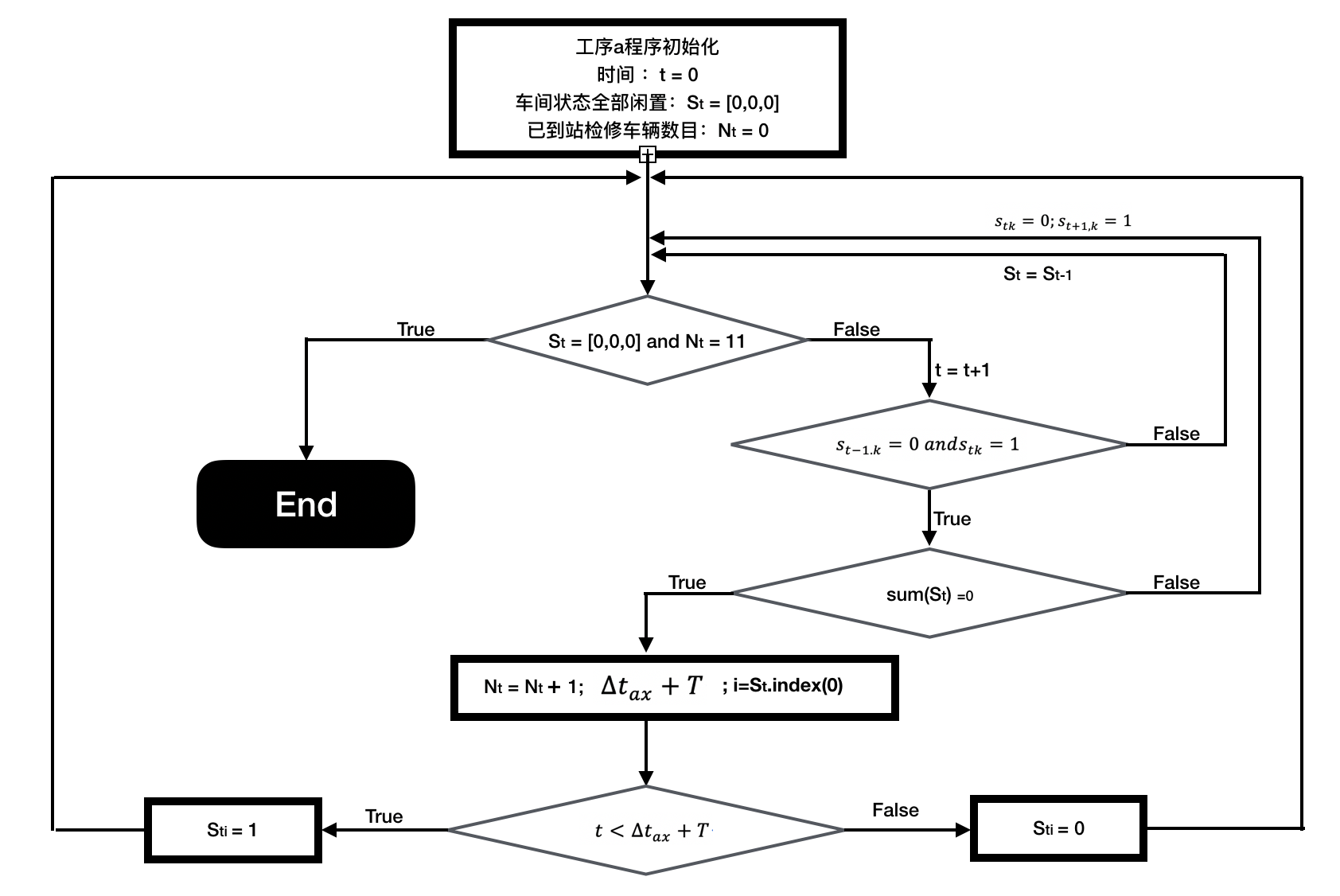


图 6.1 工序a计算流程逻辑图

求解python3代码见附件二。计算各个动车在工序a中的进站时间、出站时间、检修车间编号以及检修耗时，具体如下表所示：

表 6.3 不同类别动车在工序a车间规划方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **动车类别** | **进站时间** | **出站时间** | **检修车间编号** | **检修耗时(分钟)** |
| CRH2 | 16 | 76 | 1 | 60 |
| CRH5 | 47 | 125 | 2 | 78 |
| CRH2 | 82 | 142 | 1 | 60 |
| CRH6 | 120 | 180 | 3 | 60 |
| CRH3 | 141 | 189 | 2 | 48 |
| CRH6 | 182 | 242 | 1 | 60 |
| CRH2 | 211 | 271 | 2 | 60 |
| CRH5 | 239 | 317 | 3 | 78 |
| CRH3 | 242 | 290 | 1 | 48 |
| CRH3 | 271 | 319 | 2 | 48 |
| CRH6 | 309 | 369 | 1 | 60 |

以动车完成工序a后的出站时间为工序b的进账时间，可以类似地计算出工序b和工序c的车间规划方案，具体见表表 2.4、表 2.5:

表 6.4 不同类别动车在工序b车间规划方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **动车类别** | **进站时间** | **出站时间** | **检修车间编号** | **检修耗时(分钟)** |
| CRH2 | 76 | 196 | 1 | 120 |
| CRH5 | 125 | 275 | 2 | 150 |
| CRH2 | 142 | 262 | 3 | 120 |
| CRH6 | 180 | 342 | 4 | 162 |
| CRH3 | 189 | 333 | 5 | 144 |
| CRH6 | 242 | 404 | 1 | 162 |
| CRH2 | 271 | 391 | 3 | 120 |
| CRH3 | 290 | 434 | 2 | 144 |
| CRH5 | 317 | 467 | 6 | 150 |
| CRH3 | 319 | 463 | 7 | 144 |
| CRH6 | 369 | 531 | 4 | 162 |

表 6.5 不同类别动车在工序c车间规划方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **动车类别** | **进站时间** | **出站时间** | **检修车间编号** | **检修耗时(分钟)** |
| CRH2 | 196 | 286 | 1 | 90 |
| CRH2 | 262 | 352 | 2 | 90 |
| CRH5 | 275 | 365 | 3 | 90 |
| CRH3 | 333 | 363 | 1 | 30 |
| CRH6 | 342 | 360 | 4 | 18 |
| CRH2 | 391 | 481 | 1 | 90 |
| CRH6 | 404 | 422 | 2 | 18 |
| CRH3 | 434 | 464 | 2 | 30 |
| CRH3 | 463 | 493 | 3 | 30 |
| CRH5 | 467 | 557 | 2 | 90 |
| CRH6 | 531 | 549 | 1 | 18 |

综上可见，完成工艺c后最晚的出站时刻为557分，而第一辆动车到站水岸为16分，总耗时541分钟，约9.0167小时。

# 7 问题三求解

## 7.1 模型算法建立

与问题二类似，先把到站时间换成为以分钟为单位的数值型时间。检修类型用阿拉伯数值替代。具体如下表：

表7.1 动车到站时间表

|  |  |  |  |
| --- | --- | --- | --- |
| **到站时间** | 动车类别 | 检修类型 | 数值型时间 |
| **0:16** | CRH2 | 4 | 16 |
| **0:47** | CRH5 | 2 | 47 |
| **1:22** | CRH2 | 2 | 82 |
| **2:00** | CRH6 | 1 | 120 |
| **2:21** | CRH3 | 3 | 141 |
| **3:02** | CRH6 | 2 | 182 |
| **3:31** | CRH2 | 5 | 211 |

此外，对应题目的表4（工序基本数据）也转换成以分钟为单位的数值，具体如下表：

表7.2 动车类别的检修耗时数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | **工序类别** | | | | |
| a | b | c | d | e |
| CRH2 | 60 | 120 | 90 | 240 | 420 |
| CRH3 | 48 | 144 | 30 | 288 | 390 |
| CRH5 | 78 | 150 | 90 | 180 | 390 |
| CRH6 | 60 | 162 | 18 | 300 | 420 |

除了工序增多了外，具体的算法模型与第二问模型一致，但特别注意的是在本问中不同时间点到站的不同类别动车有着不同程度的检修工序，如检修等级为1的机车只需要完成a、b两道工序即可。因此在求解完工序a的规划方案后，需要判断机车是否需要进入工序b，对于需要进行工序b的则依照第二问模型继续求解。对于不需要进入工序b的模型，则先单独记录其功效a的出站时间，在进入工序c时判断其是否需要进入工序，对于需要进入工序c的则把上一道工序(工序a)的出站时间作为本工序（工序c）的进站时间，如此类推，直至所有机车检修完成，即程序求解完工序e的规划方案为止。

## 7.2 模型算法求解

以第二问的模型算法适当修改判断条件以及对每道工序的输入输出作调整即可求解出所有的规划方案。对于某一道工序的规划求解算法流程如下图：

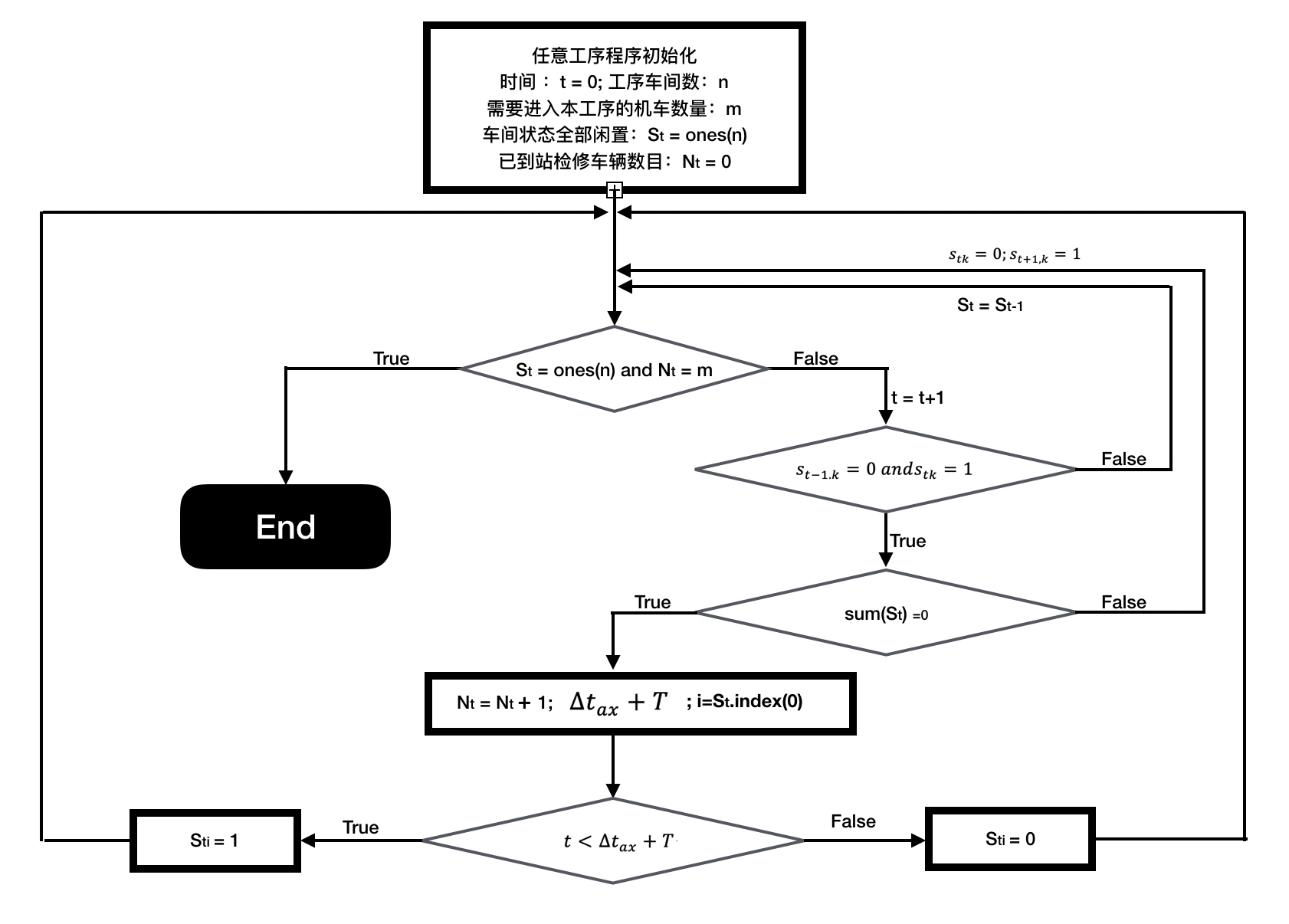


图 7.1 任意工序通用计算流程逻辑图

而对于整体所有工序的求解算法流程图下所示：

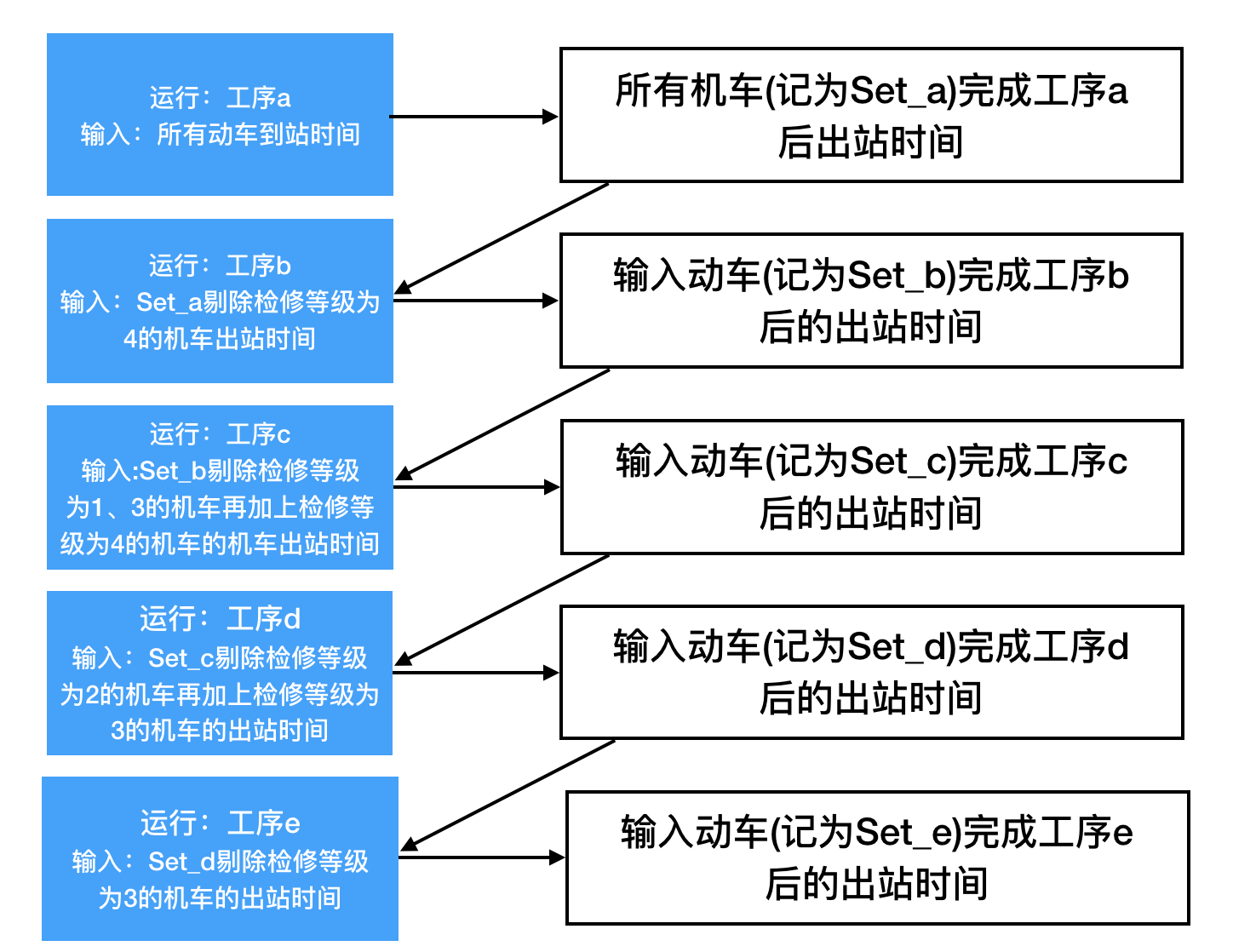


图 7.2 整体所有工序的求解算法流程图

求解python3代码见附件三。对于每道工序的求解规划结果见如下表7.1～7.5:

表 7.1 不同类别动车在工序a车间规划方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | 检修等级 | 进站时间 | 出站时间 | 检修车间编号 | 检修耗时(分钟) |
| CRH2 | 4 | 16 | 76 | 1 | 60 |
| CRH5 | 2 | 47 | 125 | 2 | 78 |
| CRH2 | 2 | 82 | 142 | 1 | 60 |
| CRH6 | 1 | 120 | 180 | 3 | 60 |
| CRH3 | 3 | 141 | 189 | 2 | 48 |
| CRH6 | 2 | 182 | 242 | 1 | 60 |
| CRH2 | 5 | 211 | 271 | 2 | 60 |

表 7.2 不同类别动车在工序b车间规划方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | 检修等级 | 进站时间 | 出站时间 | 检修车间编号 | 检修耗时(分钟) |
| CRH5 | 2 | 125 | 275 | 1 | 150 |
| CRH2 | 2 | 142 | 262 | 2 | 120 |
| CRH6 | 1 | 180 | 342 | 3 | 162 |
| CRH3 | 3 | 189 | 333 | 4 | 144 |
| CRH6 | 2 | 242 | 404 | 5 | 162 |
| CRH2 | 5 | 271 | 391 | 2 | 120 |

表 7.3 不同类别动车在工序c车间规划方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | 检修等级 | 进站时间 | 出站时间 | 检修车间编号 | 检修耗时(分钟) |
| CRH2 | 4 | 76 | 166 | 1 | 41 |
| CRH2 | 2 | 262 | 352 | 1 | 210 |
| CRH5 | 2 | 275 | 365 | 2 | 185 |
| CRH2 | 5 | 391 | 481 | 1 | 292 |
| CRH6 | 2 | 404 | 422 | 2 | 180 |

表 7.4 不同类别动车在工序d车间规划方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | 检修等级 | 进站时间 | 出站时间 | 检修车间编号 | 检修耗时(分钟) |
| CRH2 | 4 | 166 | 406 | 0 | 240 |
| CRH3 | 3 | 333 | 621 | 1 | 288 |
| CRH2 | 5 | 481 | 721 | 0 | 240 |

表 7.5 不同类别动车在工序e车间规划方案

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 动车类别 | 检修等级 | 进站时间 | 出站时间 | 检修车间编号 | 检修耗时(分钟) |
| CRH2 | 4 | 406 | 826 | 1 | 420 |
| CRH2 | 5 | 721 | 1141 | 2 | 420 |

综上可见，完成工艺e后最晚的出站时刻为1141分，而第一辆动车到站水岸为16分，总耗时1125分钟，约18.75小时。

# 8 参考文献

[1]

[2] .

[3]

[4]

# 9 附录

附录1

import numpy  
  
# 思路 利用贪心算法。一共48辆车，每一辆车比之间的车晚15分钟。能进去就进去。进不去就等。  
  
a = [0,0,0]  
b = [0,0,0,0,0,0,0,0]  
c = [0,0,0,0,0]  
  
# 从时间上遍历  
# 分为正在排队的车；a b c中等待的车；已经检修完成的车  
  
waiterCar = 0  
for time in numpy.arange(0,12+1/4,1/4):  
 waiterCar += 1  
 # 如果a数组之中有0；则每次都加入a;首先是等待的车，进入a  
 if 0 in a :  
 # 先从c数组开始。c中的每一个非0数字+1/4。当其等于1.5时，清零即出车。  
 # 再从b数组开始。b中的每一个非0数字+1/4。当其等于2时，并在C中有0位，则进入C。如果没有0位，则等待。  
 for cCar in range(len(c)):  
 if c[cCar] != 0:  
 c[cCar] += 1/4  
 if c[cCar] >= 1.5:  
 c[cCar] = 0  
  
 for bCar in range(len(b)):  
 if b[bCar] != 0:  
 b[bCar] += 1/4  
 if b[bCar] >= 2 and 0 in c:  
 b[bCar] = 0  
 inCcar = c.index(0)  
 c[inCcar] += 1e-10 # 将B中该车加入c里。以1e-10操作表示占据c中该位置  
  
 # 先让不为0的数字都+1/4  
 for aCar in range(len(a)):  
 if a[aCar] != 0:  
 a[aCar] += 1/4  
  
 # 再让等待的车进入a  
 while 0 in a and waiterCar >0 :  
 InCar = a.index(0)  
 a[InCar] += 1/4  
 waiterCar -= 1  
  
 # 最后判断a中的车是否进入B  
  
 for aCar in a:  
 if aCar >= 1 and (0 in b):  
 aCar = 0  
 inBcar = b.index(0)  
 b[inBcar] += 1e-10  
  
 # a 数组中没有0；则让abc之中所有的车的时间加上 1/4  
 else:  
 # 先从c数组开始。c中的每一个非0数字+1/4。当其等于1.5时，清零。  
 # 再从b数组开始。b中的每一个非0数字+1/4。当其等于2时，并在C中有0位，则进入C。如果没用0位，则等待。  
 # 此类是a中无0的，则a中每个数字都+1/4.当其等于1时，并在B中有0位，则进入B。  
  
 for cCar in range(len(c)):  
 if c[cCar] != 0:  
 c[cCar] += 1/4  
 if c[cCar] >= 1.5:  
 c[cCar] = 0  
  
 for bCar in range(len(b)):  
 if b[bCar] != 0:  
 b[bCar] += 1/4  
 if b[bCar] >= 2 and 0 in c:  
 b[bCar] = 0  
 inCcar = c.index(0)  
 c[inCcar] += 1e-10 # 将B中该车加入c里。以1e-10操作表示占据c中该位置  
  
  
 for aCar in range(len(a)):  
 a[aCar] += 1/4  
 if a[aCar] >= 1 and 0 in b:  
 a[aCar] = 0  
 inBcar = b.index(0)  
 b[inBcar] += 1e-10  
  
  
# 此时得到了abc之中所有的车的状态，以及等待的车辆。  
addtime = 0  
print('等待车辆数目 = ',waiterCar)  
print('a工序中各个车间的检修时间，',a)  
print('b工序中各个车间的检修时间，',b)  
print('c工序中各个车间的检修时间，',c)  
  
# 计算剩下车辆需要检测完需要的时间  
# 首先计算车辆不再有排队所经历的时间  
for time in numpy.arange(0,10+1/4,1/4):  
 if waiterCar == 0 :  
 addtime = time  
 print('time0 =', time)  
 break  
  
 else:  
 # 如果a数组之中有0；则每次都加入a;首先是等待的车，进入a  
 if 0 in a:  
 # 先从c数组开始。c中的每一个非0数字+1/4。当其等于1.5时，清零即出车。  
 # 再从b数组开始。b中的每一个非0数字+1/4。当其等于2时，并在C中有0位，则进入C。如果没有0位，则等待。  
 for cCar in range(len(c)):  
 if c[cCar] != 0:  
 c[cCar] += 1 / 4  
 if c[cCar] >= 1.5:  
 c[cCar] = 0  
  
 for bCar in range(len(b)):  
 if b[bCar] != 0:  
 b[bCar] += 1 / 4  
 if b[bCar] >= 2 and 0 in c:  
 b[bCar] = 0  
 inCcar = c.index(0)  
 c[inCcar] += 1e-10 # 将B中该车加入c里。以1e-10操作表示占据c中该位置  
  
 # 先让不为0的数字都+1/4  
 for aCar in range(len(a)):  
 if a[aCar] != 0:  
 a[aCar] += 1 / 4  
  
 # 再让等待的车进入a  
 while 0 in a and waiterCar > 0:  
 InCar = a.index(0)  
 a[InCar] += 1 / 4  
 waiterCar -= 1  
  
 # 最后判断a中的车是否进入B  
  
 for aCar in a:  
 if aCar >= 1 and (0 in b):  
 aCar = 0  
 inBcar = b.index(0)  
 b[inBcar] += 1e-10  
  
 # a 数组中没有0；则让abc之中所有的车的时间加上 1/4  
 else:  
 # 先从c数组开始。c中的每一个非0数字+1/4。当其等于1.5时，清零。  
 # 再从b数组开始。b中的每一个非0数字+1/4。当其等于2时，并在C中有0位，则进入C。如果没用0位，则等待。  
 # 此类是a中无0的，则a中每个数字都+1/4.当其等于1时，并在B中有0位，则进入B。  
  
 for cCar in range(len(c)):  
 if c[cCar] != 0:  
 c[cCar] += 1 / 4  
 if c[cCar] >= 1.5:  
 c[cCar] = 0  
  
 for bCar in range(len(b)):  
 if b[bCar] != 0:  
 b[bCar] += 1 / 4  
 if b[bCar] >= 2 and 0 in c:  
 b[bCar] = 0  
 inCcar = c.index(0)  
 c[inCcar] += 1e-10 # 将B中该车加入c里。以1e-10操作表示占据c中该位置  
  
 for aCar in range(len(a)):  
 a[aCar] += 1 / 4  
 if a[aCar] >= 1 and 0 in b:  
 a[aCar] = 0  
 inBcar = b.index(0)  
 b[inBcar] += 1e-10  
  
# 将a、b、c中所有的车出站所需要的时间  
  
for time in numpy.arange(0,5,1/4):  
  
 if sum(a) <= 0.1 and sum(b) <= 0.1 and sum(c) <= 0.1 :  
 addtime += time  
 print('time1 =', time)  
 break  
  
 else:  
 # 如果a数组之中有0；则每次都加入a;首先是等待的车，进入a  
  
 # 先从c数组开始。c中的每一个非0数字+1/4。当其等于1.5时，清零。  
 # 再从b数组开始。b中的每一个非0数字+1/4。当其等于2时，并在C中有0位，则进入C。如果没用0位，则等待。  
 # 此类是a中无0的，则a中每个数字都+1/4.当其等于1时，并在B中有0位，则进入B。  
  
 for cCar in range(len(c)):  
 if c[cCar] != 0:  
 c[cCar] += 1 / 4  
 if c[cCar] >= 1.5:  
 c[cCar] = 0  
  
 for bCar in range(len(b)):  
 if b[bCar] != 0:  
 b[bCar] += 1 / 4  
 if b[bCar] >= 2 and 0 in c:  
 b[bCar] = 0  
 inCcar = c.index(0)  
 c[inCcar] += 1e-10 # 将B中该车加入c里。以1e-10操作表示占据c中该位置  
  
 for aCar in range(len(a)):  
 if a[aCar] != 0:  
 a[aCar] += 1 / 4  
 if a[aCar] >= 1 and 0 in b:  
 a[aCar] = 0  
 inBcar = b.index(0)  
 b[inBcar] += 1e-10  
  
alltime = addtime+12  
print('等待车辆数目 = ',waiterCar)  
print('a工序中各个车间的检修时间，',a)  
print('b工序中各个车间的检修时间，',b)  
print('c工序中各个车间的检修时间，',c)  
print('总共花费的时间',alltime)

附录2