中兴捧月迪杰斯特拉派

算法思路

作者：邢卓林

学校：西安电子科技大学

日期：2020 年 5 月 7 日

图表目录

目录

第一章 主程序

1.1 主程序概述

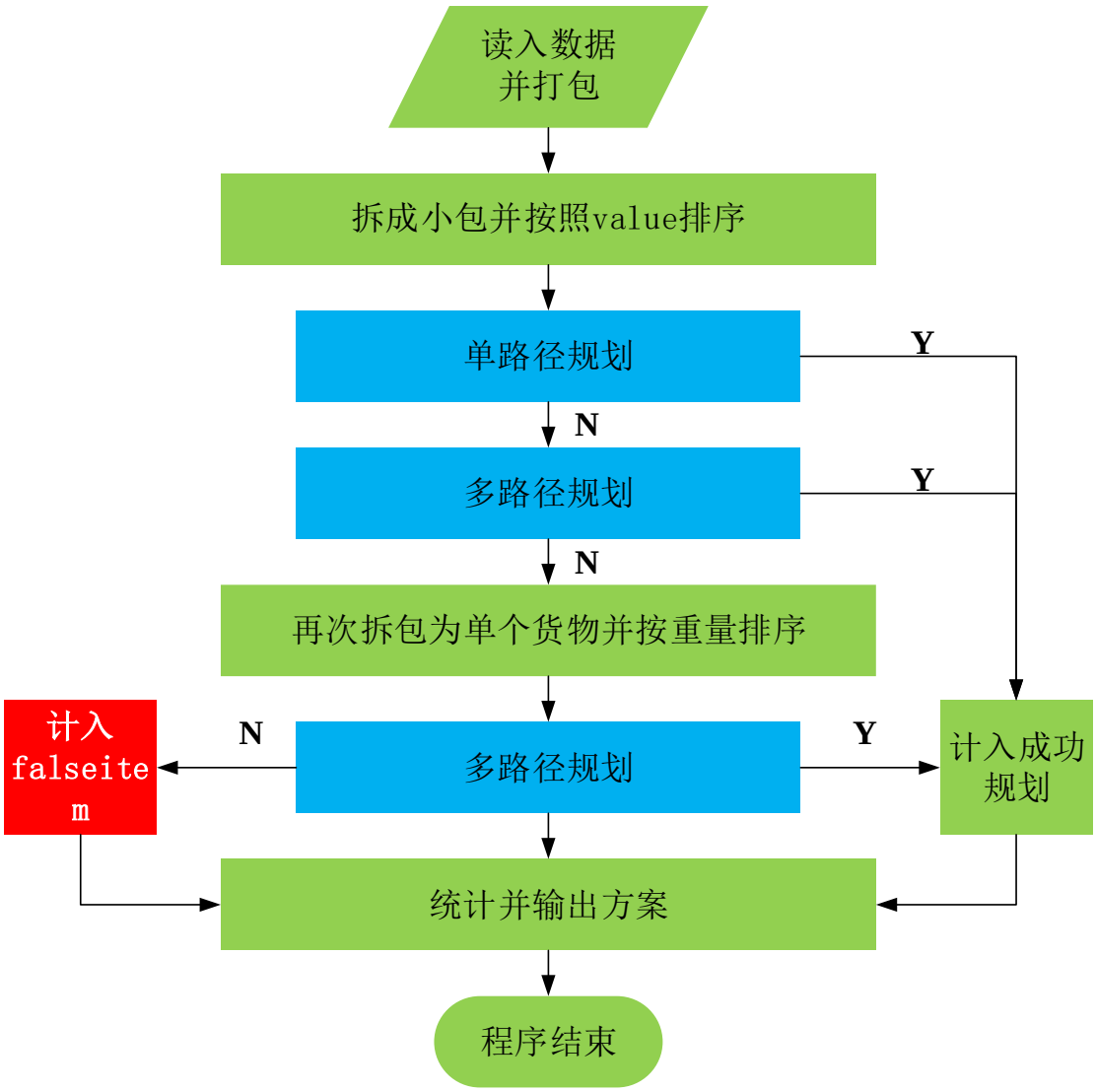

图 1 主程序流程图

主程序算法流程说明：

Step1：读入数据，取货物终点起点中标号小的作为起点，标号大的作为终点。将原宿点相同，并且必经节点相同的不同货物合并打包为 package（数据结构如图 2 所示）。

Step2：将每个 package 中的单个货物合并为重量不大于最大载重的货物包 Items，计算每个 Items 的 value 值，计算公式为：

$$Items.value = Items.size + Items.weight/MaxLoad$$

其中 size 为 Items 中包含的货物个数，Weight 为 Items 中货物的总重量，MaxLoad 为单个列车的最大载重。

Step3：将打包好的 Items 按照 Value 值从大到小排列。

Step4：以 Value 值大小为准，按从大到小的顺序依次进行单路径规划，全部规划一次后，将失败的货物包重新进行多路径规划。

Step5：将规划失败的货物包拆为单个货物，并按照货物重量大小排序，从大到小依次进行多路径规划。失败货物计入 falseitem。

Step6：统计失败货物数量和重量，输出所有货物规划结果，退出程序。

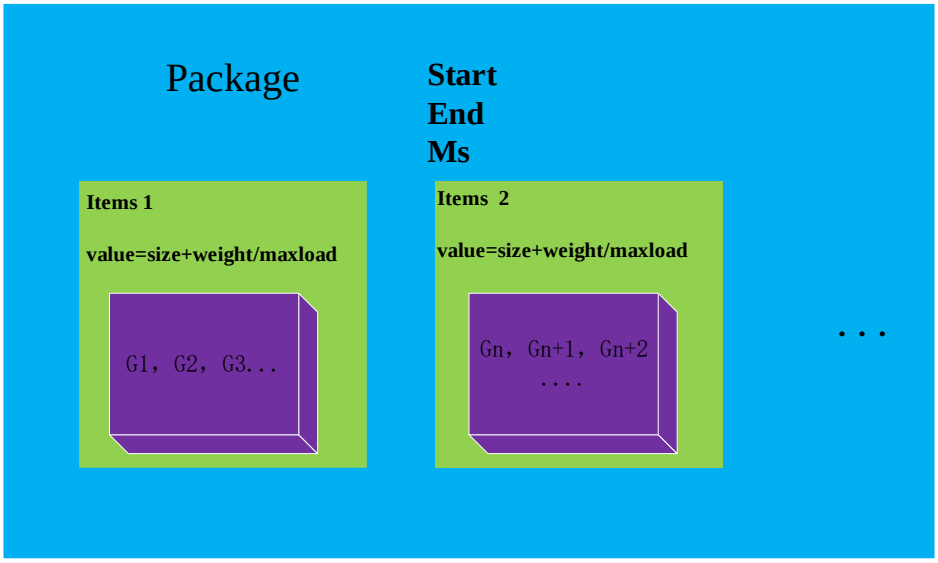

图 2 货物打包结构

第二章 路径搜索算法

2.1 改进迪杰斯特拉

为了使迪杰斯特拉算法能搜索到必经节点，对经典的迪杰斯特拉算法进行了改进。流程图如图 3 所示：

算法流程：

Step1：将终点加入必经节点列表，置为最后一个。

Step2：Source 为起点，将第一个终点设置为第一个必经节点。

Step3：使用 D 算法搜索路径，并记录。

Step4：判断是否到达终点，如过没有，更新起点为上一个终点，终点为下一个必经节点。将图重置，标记已有路径，转步骤 3。如果到达终点则输出路径。

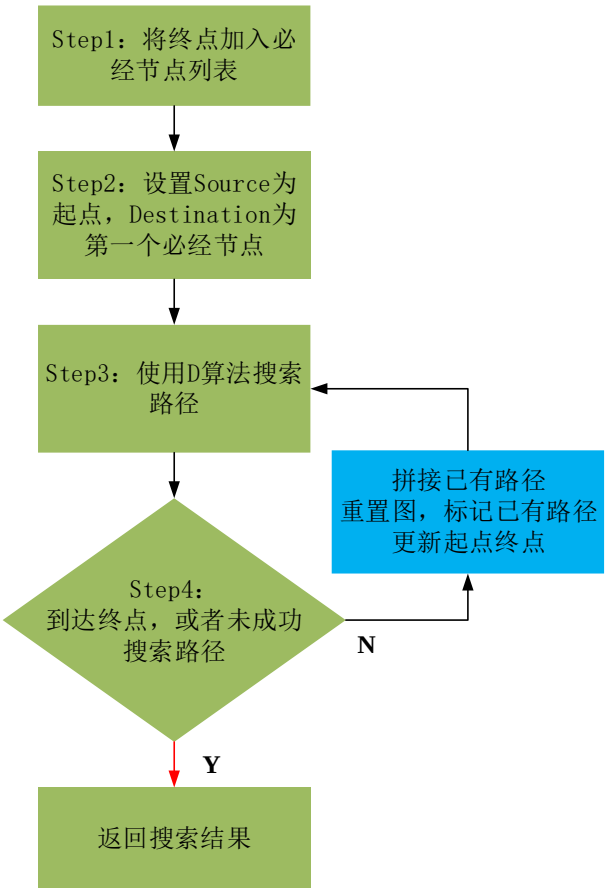

图 3 改进迪杰斯特拉算法流程

2.2 改进 DFS 搜索多个路径

算法流程（如图 4）概述：

Step1：起点节点入栈。

Step2：判断栈空，路径数量和路径长度是否超过预设值，超过或栈空则退出否则转 step3。

Step3：节点出栈，若节点已访问，取消访问标记，从路径中删除节点，转步骤 2，若节点未访问，标记节点将节点加入路径，转 Step4。

Step4：判断节点是否为终点，是则记录从开始节点到终点的路径，取消节点标记，从当前路径中删除节点，返回 Step3。否则转 Step5。

Step5：将该节点入栈，遍历当前节点邻居，把未访问节点入栈，转 Step2。

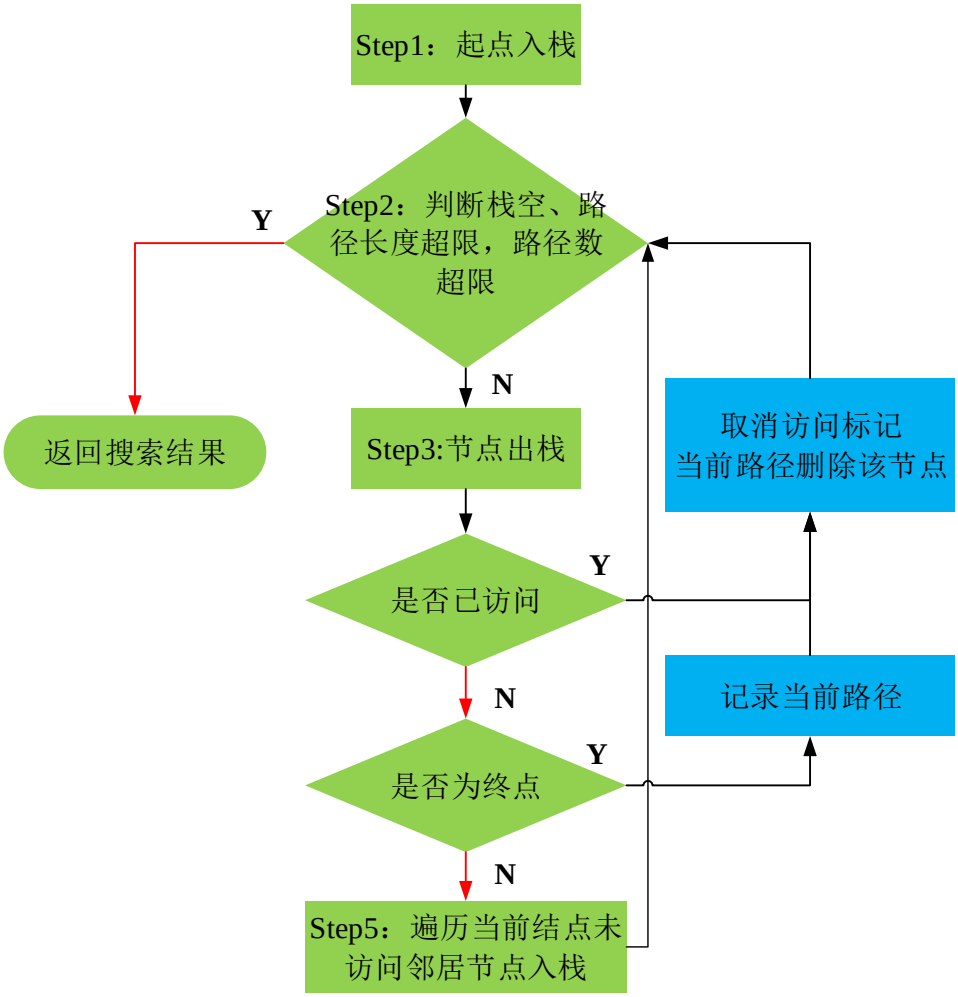

图 4 多路径 DFS 搜索流程图

第三章 资源规划

3.1 多路径规划与单路径规划

规划流程示意图如图 5,概述如下：

- 单路径规划使用迪杰斯特拉算法和改进的迪杰斯特拉最短路径算法计算路径，每次计算结果只有一条路径。
- 多路径规划使用 DFS 多路径算法计算路径，每次计算路径不超过 15 条，路径长度不超过 15。
- 在进行资源规划的时候，使用单路径规划只进行一次规划，而使用多路径规划则需要多次尝试直到路径全部尝试过或者规划成功。
- 资源规划按照独占车道、不变道拼车、变道拼车的优先级依次尝试。

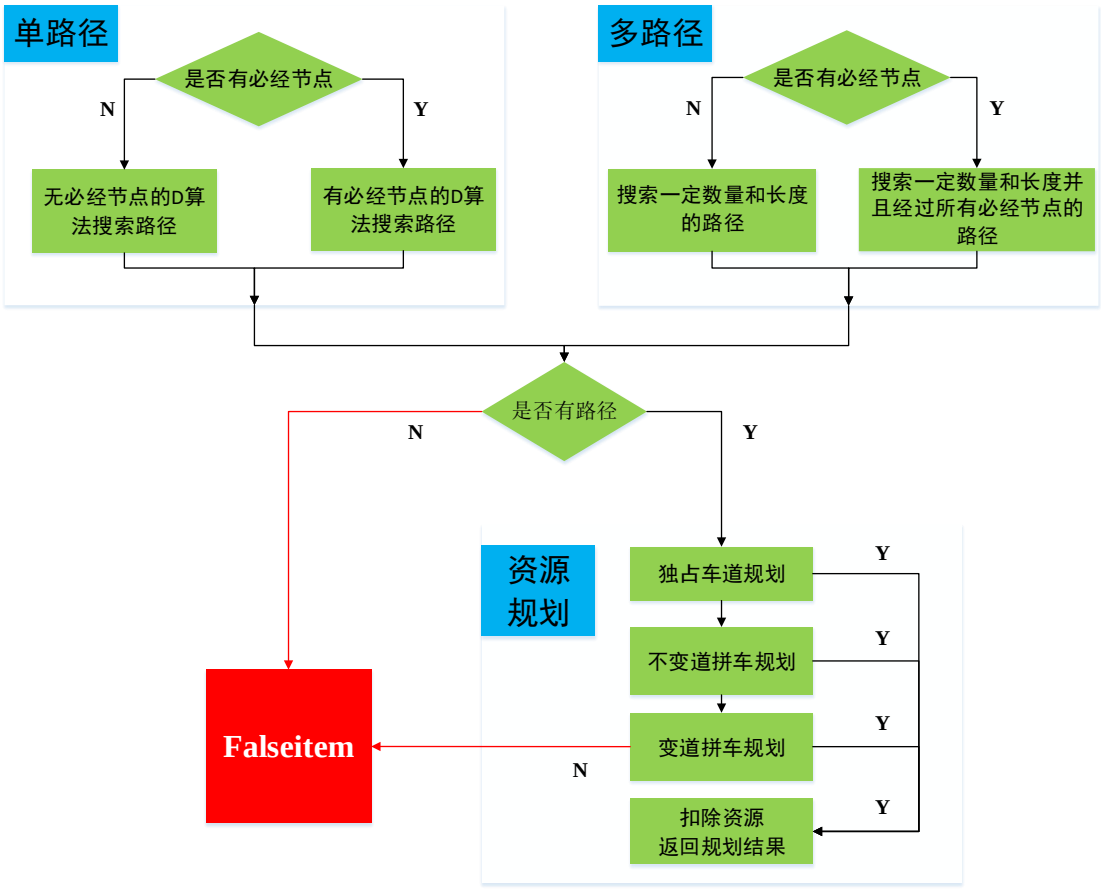

图 5 资源规划流程图

3.2 拼车策略

3.2.1 需要维护的的信息简述

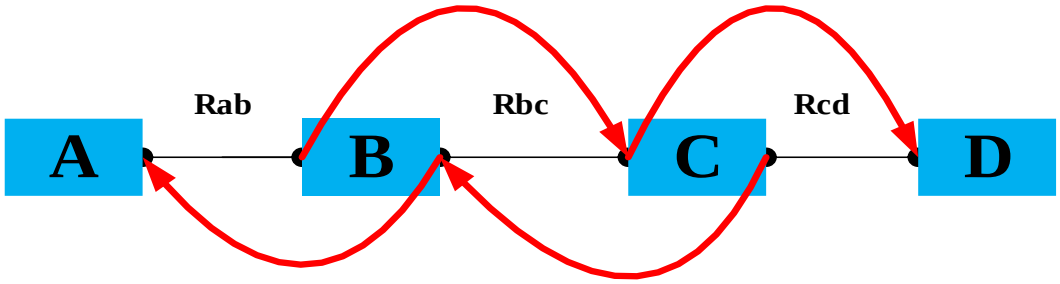

图 6 Next 信息记录示意

注:图中蓝色方块结合英文大写字母代表某个站点，带黑色圆点的直线代表轨道上的某个车道，Rab 代表轨道号。红色箭头代表记录的信息。例如：Rbc 的 B 端记录的下一跳信息为{当前站点（B），车道号码（B 到 C 的车道号），下一车道号码（A 到 B 的黑线），下一站点（A）}，通过 A，B 可定位下一轨道。

拼车前每条轨道需要为维护的数据结构如下：

a) 捡货工位图：用于判断某个工位是否有拣货员，使用轨道号、站点名、车道号定位。

b) 起始点位图：用于判断某个车道是否是起点或者终点，使用轨道号、站点名、车道号定位。

c) 装载余量图：用于记录当前车道剩余装载量，使用轨道号、站点名、车道号定位。

d) 车辆占用图：用于记录当前车道是否已被占用，使用轨道号、站点名、车道号定位。

e) 下一跳信息：用于记录当前车道在当前车站的下一跳，使用轨道号、站点名、车道号、下一车道号定位下一站点列表。记录方式为双向记录。

其中 next 的信息记录示意图如图 5 所示，另外每个站点的剩余拣货员也需要进行维更新。

3.2.2 拼车问题分析

首先假设拼车时货物全程在同一车道，那么中间站点可能遇到的情况以及需要中间站点货员如图 7 所示。途中所示均为分叉的情况，反向则为汇聚的的情况，由于货物双向运输，所以中间站点需要进行分叉汇聚的双向检查与资源分配。而起点终点只需要进行单向检查与

分配起始点可能出现情况参照中间站点的 case6。

- 资源校检策略，假设路径为 A-B-C，使用轨道为 Rab，Rbc：

Step1：对于中间站点 B 检查当前货物所在轨道 Rab 的 B 端点是否有下一跳，没有则转 Step2，有则检查每一个下一跳对应的工位是否有拣货员，记录所需拣货员。

Step2：检查当前路径的下一轨道 Rbc 的 B 端点是否有下一跳，检查每一个下一跳对应的工位是否有拣货员，记录所需拣货员。

Step3：检查 B 站点的两个工位，是否有拣货员，是否为起止点，是否包含在 step1：step2 的下一跳中，记录所需拣货员。

Step4：判断 A 站点拣货员是否足够，若不够则退出，够则检查下一个中间点。

- 资源分配策略同校检策略。

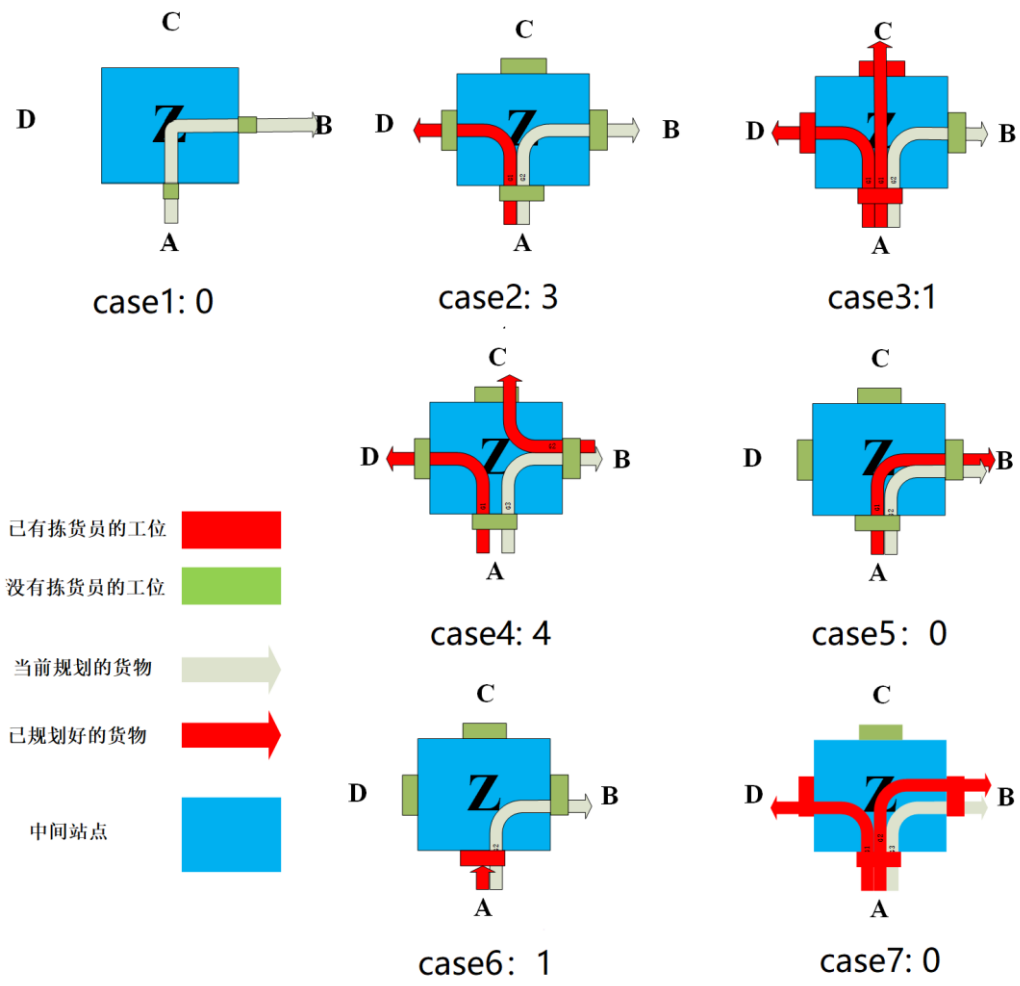

图 7 中间站点拼车

3.3.3 拼车算法

- 不换车道拼车算法思路：

Step1：根据当前货物重量，更新链路上的可用车道，并求出交集。

Step2：判断是否有可用车道，有则取下一个可用车道，否则转 step4。

Step3：参考上一节的几种情况，对链路资源进行检查，如果资源满足则转 step4。否则转 step:2.

Step4：拼车成功则扣除资源，返回结果，否则规划失败返回”null”。

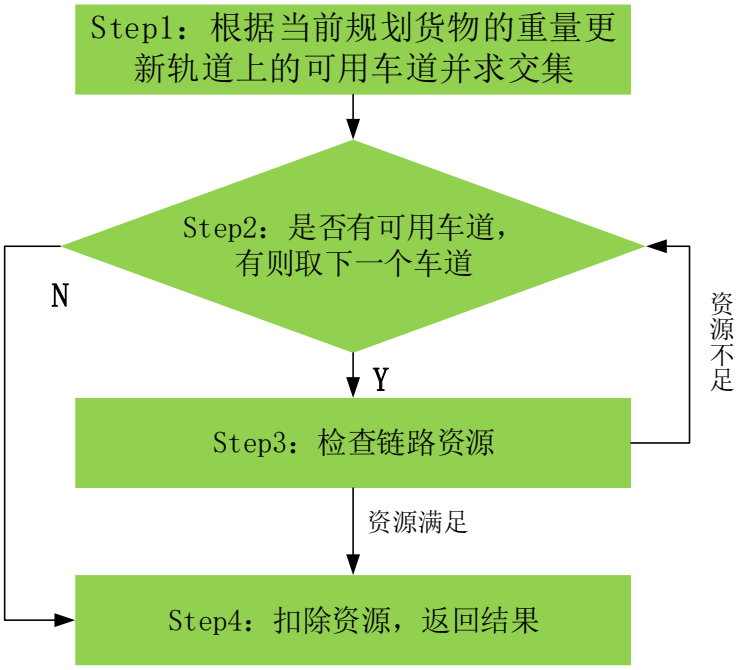

图 8 不换车道拼车流程

- 换车道拼车思路

随机选取可用车道，尝试一定次数，资源校检算法除需要遍历 next 的所有下一条车道外其余与不换车道算法类似。由于后期拣货员资源不足，所以该算法规划成功的数量极少，需要改进。

第四章 小结

4.1 未解决问题以及可用优化思路

1、必经节点较多的物品规划失败。

分析：路径搜索方面还是很盲目，有待改进，部分有必经节点和无未经节点的货物应该可以打包规划。

2、部分重量为 100 无未经节点的货物规划失败。

分析：可能由于前期部分小重量货物优先独占车道 导致无法分配，货物分配优先级还可以优化。

3、程序运行时间较长，大约 0.2S-20S 不等

分析：部分数据结构使用数组代替，路径搜索算法较为盲目，尤其是多个路径搜索，虽然对路径数和路径长度加了限制但是还是较为耗时。可以用部分数组代替当前的数据结构，采用时间复杂度低的路径搜索算法。

4.2 未尝试优化思路

- 在拼车时选择占用资源最少的车道，比如选择剩余装载量最小的车道，选择使用拣货员最少的车道。
- 在货物规划完成后，对车道进行二次调整，合并车道提高效率。

4.3 结束语

我是来自西安电子科大学的研一学生。热爱运动，之前的大部分课余时间都去参加了足球比赛，所以在竞赛方面参与很少，非常感谢中兴的这次比赛，能然我有一次参与竞赛的机会，在半个多月的编码过程中我也学到了很多。同时感谢各位赛事工作人员的辛勤付出，希望各位身体健康，希望中兴发展越来越好！