动态优化

AGV 运行环境表达

AGV 运行环境的表达是 AGV 路径规划的基础，主要是通过电子地图模型的方式进行表达。电子地图就是在上位机中表达现场运行环境，包括现场中 AGV 的运行路径、工位节点等信息，以便能够在此基础上能够进行算法的调用和数据的分析处理[32]。

拓扑地图法是直观的模拟环境地图，以点来表示环境中的节点，以特定的线段表示地图中的可行路径，路径规划便是沿着环境中的可行路径进行搜索。拓扑地图法不仅可以表示节点与节点，路径与路径之间的拓扑信息，而且通常包含其长度信息，能够直观的观察 AGV 在运行过程中的路径，使拓扑地图同时具有高效性和精确性[33]。

根据任务信息的获得的完整性 AGVS 调度策略可分为离线（静态）调度策略和在线（动态）调度策略。

离线调度策略，指基于已确定的路径或任务信息进行系统任务统筹派发，且一旦派发给对应 AGV 运输车，路线即确定，非特殊情况不做更改[13]。离线调度技术实现简单，操作性强，但是系统的灵活性低，应对突发事情能力差，且系统的总体效率提升有上限。

动态调度策略是随着 AGV 的位置信息和新增任务信息的更新，在离线调度策略的基础上，对 AGV 的执行任务的路线进行实时调整[14]。动态场景下某 AGV 运输车的路径变化有可能会对其他 AGV 造成影响，任务调度策略设计较为复杂，且对计算机硬件性能有较高要求。

1 常用的环境建模方法分析

AGV 的行走空间为二维空间，因此本节只对二维空间常见的环境建模方法进行介绍，如可视图法、栅格地图法和拓扑地图法等。

2.拓扑地图法

1978 年 Mataric 和 Kuipers 首次提出了基于拓扑关系构建实际环境地图的方法，该方法只记录有特殊意义的节点以及节点间的连通关系，即点和边的信息，不考虑障碍物等其他环境情况，拓扑地图如图 2-2 所示。其中，特殊意义的节点指工位点、停靠站、充电桩以及路径交叉点等典型位置，两点间的可通达连线的权值用实际两节点间的路线长度表达。拓扑地图的构建和后续维护的内存成本和技术成本较低，且由于不存储不可通达的路径点的信息，因此对计算机的运算性能要求较低，搜索效率高。

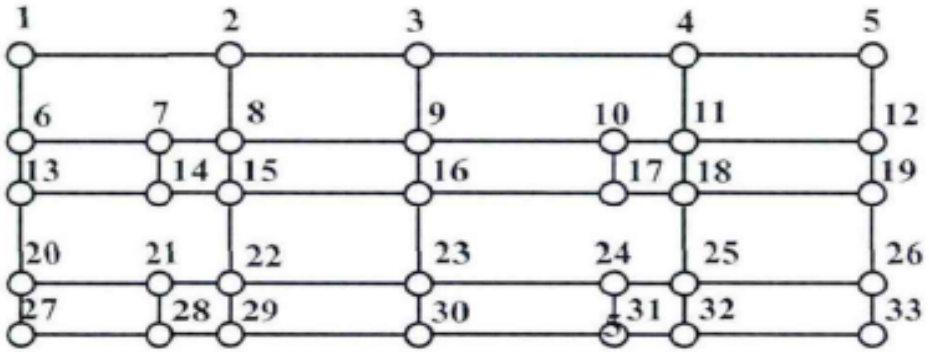


图 2-2 拓扑法环境模型

3.栅格地图法

栅格法最早由 Howden 提出，其思想是用单元格将作业空间中的障碍物等信息表达出来，单元格权值为布尔变量，只有 0 和 1 两种，0 代表自由空间，1 代表障碍物空间，如图 2-3 所示。单元格法类似于将整个实际作业环境放在一个刻度可调的坐标系中，环境表达的准确性与单元格的大小成反比，即单元格刻度越小，越能准确地表达环境信息，因此在选择单元格刻度时需要根据实际环境的大小、误差的承受范围和计算机的运行性能综合考虑。

AGV 系统使用栅格法进行环境建模时，一般栅格的单位值为一台 AGV 宽度的一点几倍，以 AGV 刚好可以在并排的单元格无碰撞行走为宜，既可以满足定位精度的需要又可以节约存储空间、降低运算成本。

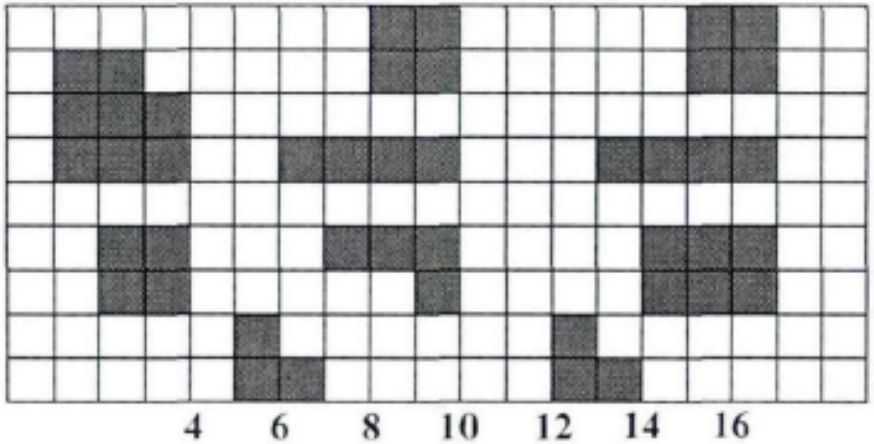


图 2-3 栅格法环境建模

单 AGV 路径规划

单 AGV 路径规划是指在建立好环境电子地图的前提下，对于给定的起点和终点，选择从起点到终点的最优路径。由于从起始节点到终止节点可能有很多的连接方式，每种连接方式的路径长度都各不相同，所以在不考虑车辆之间的干扰、环境的干扰以及线路阻塞的条件下，路径规划问题就转换为在所建立的电子地图的环境下，求起点到终点的最短路径长度[34]。对于搜索最短路径问题，国内外的专家已经提出了很多种求解算法，如 Dijkstra 算法、A*算法等[35, 36]。

1 Dijkstra 算法

Dijkstra 算法核心思想是：以起点为中心逐步向外层扩展，求出到各个节点的最短距离，直到扩展到终点为止[37]。算法是建立在一个带权值的拓扑图 **G = (V,E)** 的基础上，在 AGV 路径规划中，权值一般取为边的长度[38]。算法将节点集合分为**S**（已求最短路径节点）和**U**（未求最短路径节点）两组，在初始状态下，**S**仅有起点一个节点，**U**包含除起点外的所有节点。并且在算法搜索过程中，起点到**S**集合中的节点的最短路径长度小于或等于起点到**U**集合中的最短路径长度。算法从起始节点逐步向外搜索，将求的最短路径的节点由**U**集合转移到**S**集合，直到搜索到终点的最短路径或者**U**集合为空是，算法结束。

其算法的具体步骤如下：

初始条件：起点为*S0*，终点为*U0*；

解决问题：求解*S0*到*U0*的最短路径；

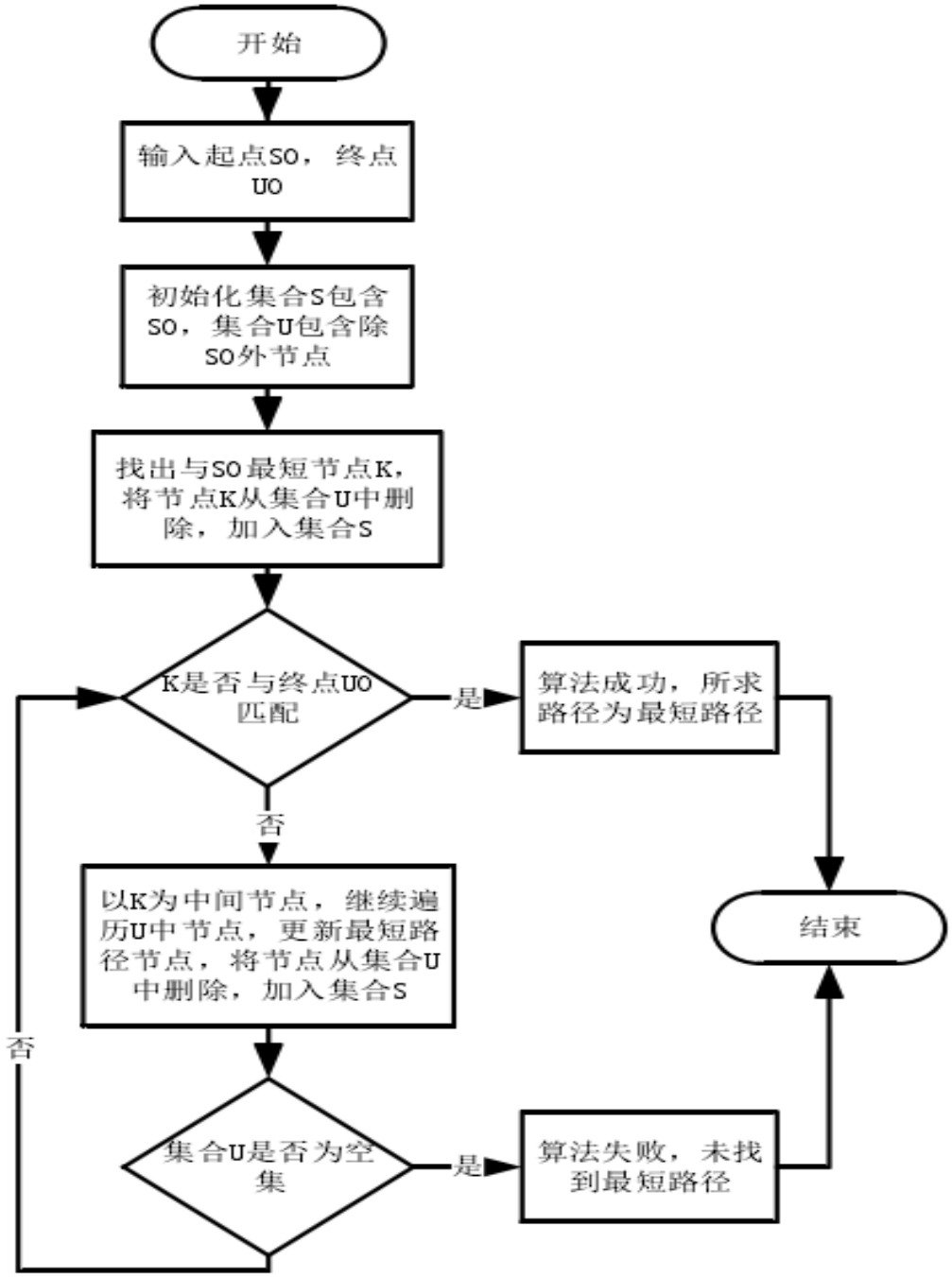
1）初始化集合**S**和**U**，其中**S**包含节点*S0*，**U**包含除*S0*以外节点。初始化权值为边的长度，若两点相连，则权值为连接两点的边的长度，若两点不相连，则两点间的权值为无穷大。

2）遍历各个节点，通过比较找出与起点*S0*最短的节点*k*，将*k*从**U**集合中删除并加入集合**S**中。

3）判断节点*k*是否与终点*U0*相匹配，如节点*k*即为终点*U0*，则算法终止，所求路径即为 Dijkstra 算法下的最短路径；若不匹配，则跳转到下一步。

4）以 k 为中间节点，继续遍历 **U** 中的节点，计算起点 $S0$ 经过 k 节点到 **U** 中节点的距离，与起点 $S0$ 不经过 k 节点的距离进行比较。若经过 k 节点的距离比不经过 k 节点的距离短，则求的最短距离为新的最短距离，该节点为新的中间节点，将此节点加入集合 **S**。

5）判断集合 **U** 是否为空间，若为空集，表示没有找到由起点到终点的最短路径，若不为空集，则跳转到步骤 3 进行继续求解。


开始
输入起点SO，终点UO
初始化集合S包含SO，集合U包含除SO外节点
找出与SO最短节点K，将节点K从集合U中删除，加入集合S
K是否与终点UO匹配
是
算法成功，所求路径为最短路径
否
以K为中间节点，继续遍历U中节点，更新最短路径节点，将节点从集合U中删除，加入集合S
否
集合U是否为空集
是
算法失败，未找到最短路径
结束

2. A*算法

A*算法的特点是它含有一个启发函数。在搜索过程中，并非进行盲目搜索，而是利用启发函数对途中各个节点的代价进行评估，使搜索方向能够尽量的与终点方向一致[39]。由于启发函数的存在，A*算法并不需要对所有的节点进行搜索，从而提高了搜索效率。

A*算法可用一个评价函数进行表示：

$$f(k)=g(k)+h(k)$$

其中：$f(k)$ 为从起点 $S0$ 经过节点 k 到终点 $U0$ 的评价函数，表示起点 $S0$ 到终点 $U0$ 的距离估计值；

$g(k)$ 为从起点 $S0$ 到节点 k 的代价函数，表示起点 $S0$ 到节点 k 的最短距离值；

$h(k)$ 为节点 k 到终点 $U0$ 的启发函数，表示节点 k 到终点 $U0$ 的距离估计值。

由于本项目采用拓扑图法建立电子地图，当前节点到终点的距离能够很好的在电子地图上进行显示。为了避免上述由于启发函数 $h(k)$ 选择不合理的问题，选用直线距离为启发函数，即 $h(k)=\sqrt{(x_{U0}-x_k)^2+(y_{U0}-y_k)^2}$ 。选取此启发函数不仅能够让 $h(k)$ 接近起点到终点的距离，而且能够确保算法的搜索方向朝着目标节点进行搜索。

A*算法的具体搜索步骤如下：

初始条件：起点为$S0$，终点为$U0$；

解决问题：求解$S0$到$U0$的最短路径；

1）初始化集合**S**和**U**，其中**S**包含节点$S0$，**U**包含除$S0$以外所有节点；初始化权值为边的长度，若两点相连，则权值为连接两点边的长度，若两点不相连，则两点间的权值为无穷大；

2）遍历集合**U**，找出集合**U**中f值最小的节点最小节点i，将节点i从**U**集合中删除，加入**S**集合；

3）判断节点i是否为终点$U0$，若是，则算法结束，所求路径即为最优路径，若该点不为终点，则跳转至步骤4；

4）遍历与i节点相邻的节点j，判断节点j是否在**U**中，若节点j在集合**U**中，把节点i当作节点j的上一个点，计算$f(i),g(j),h(j)$的值；若节点j不在集合**U**中，则跳转至步骤5

5) 判断节点 j 的 $h(j)$ 值与之前所求值的大小,若当前值小于之前 $h(j)$ 的值，把节点 i 当作 j 的上一个节点，以新的 $f(i), g(j), h(j)$ 的值替换原来的值。

6）判断 **U** 集合是否为空，若为空，则算法结束，找不到从起点到终点的最短路径，若不为空，跳转到步骤 2。

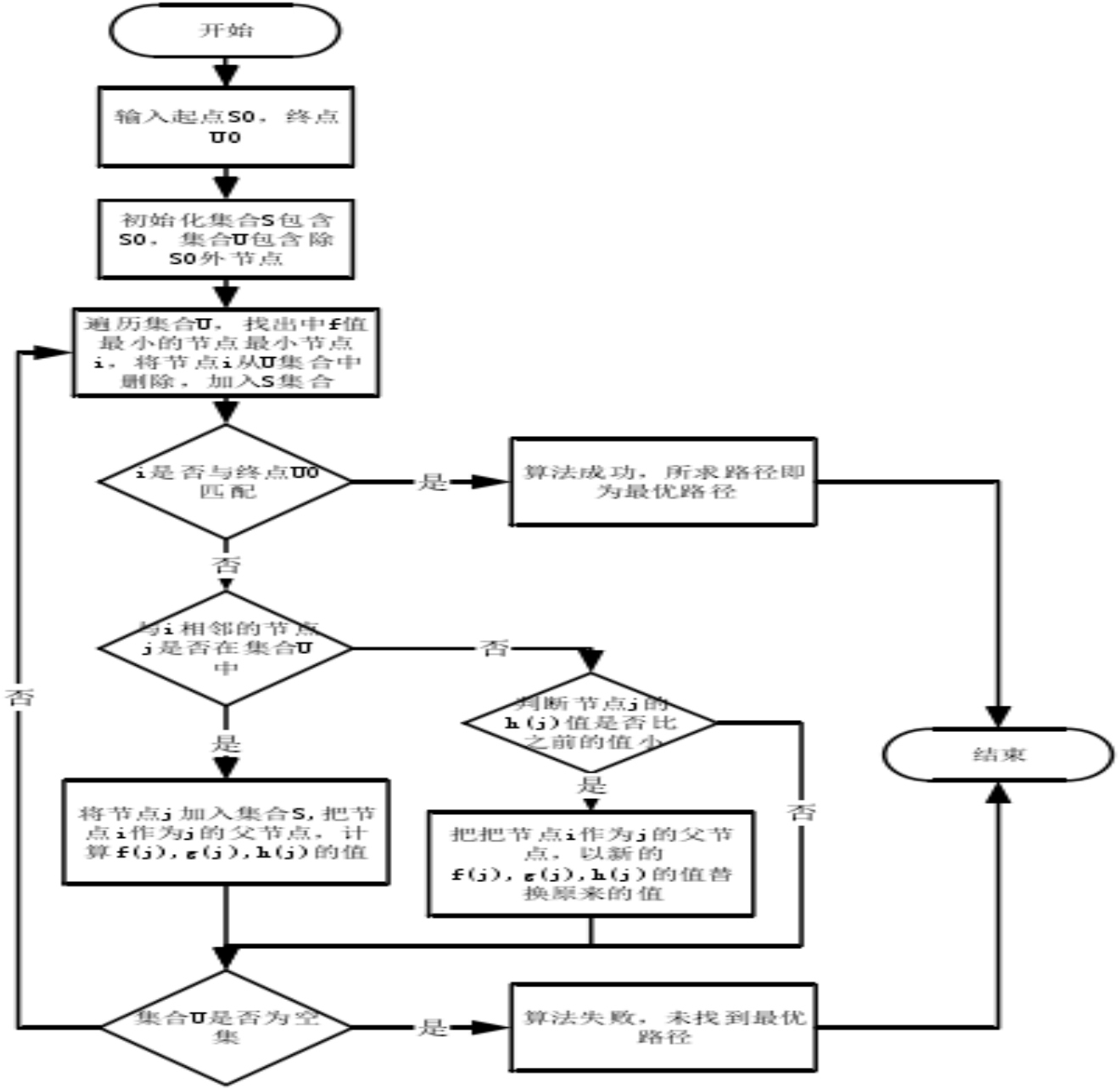


图 2-5 A*算法执行流程图

2.4 改进 Dijkstra 算法

在工厂的实际运输环境中，由于节点加工工位的不同以及工位在地图中所处的位置不同，可能会造成加工工位所对应节点的繁忙程度不同，在 AGV 运输货物过程中可能会造成路径的繁忙程度有较大差异。传统 Dijkstra 算法在进行路径规划时只是根据路径的长度来分配最优路径，而没有考虑到路径的繁忙程度。在采用传统 Dijkstra 算法规划路径时，可能会由于某段路径所规划的车辆过多而造成 AGV 在该段路径调度难度加大甚至阻塞。

改进的 Dijkstra 算法是建立在传统 Dijkstra 算法上的一种有机路径规划算法，加入了热度值和加权值对路径繁忙程度的评估，使规划的路径更合理。其算法流程如下：

（1） 对工厂中 AGV 所需完成的调度任务进行统计，分别调用传统 Dijkstra 算法进行路径规划，将所规划的路径在表格中进行统计；

（2） 将表格中路径分解，对任务中各段路径出现的次数进行统计，并记录在表格中；

（3） 对路径的繁忙程度进行评估，本文用路径热度 k 表示。热度 k 的计算方式如下：设路径热度的集合为 $\mathbf{K}=\{k_1,k_2,k_3,k_i.....\}$，步骤 2 中出现的路径用集合 $\mathbf{L}=\{l_1,l_2,l_3,l_i.....\}$ 表示，路径出现次数用集合 $\mathbf{N}=(n_1,n_2,n_3,n_i.....)$ 表示。则路径 l_i 的热度 k_i 的计算公式如 2-1：

$$k_i=1+\frac{n_i}{n_1+n_2+n_3+n_i....} \tag{2-1}$$

（4） 对于在上述统计中没有出现的路径，其热度 k 均用 1 表示，将热度统计成表格储存到数据库中；

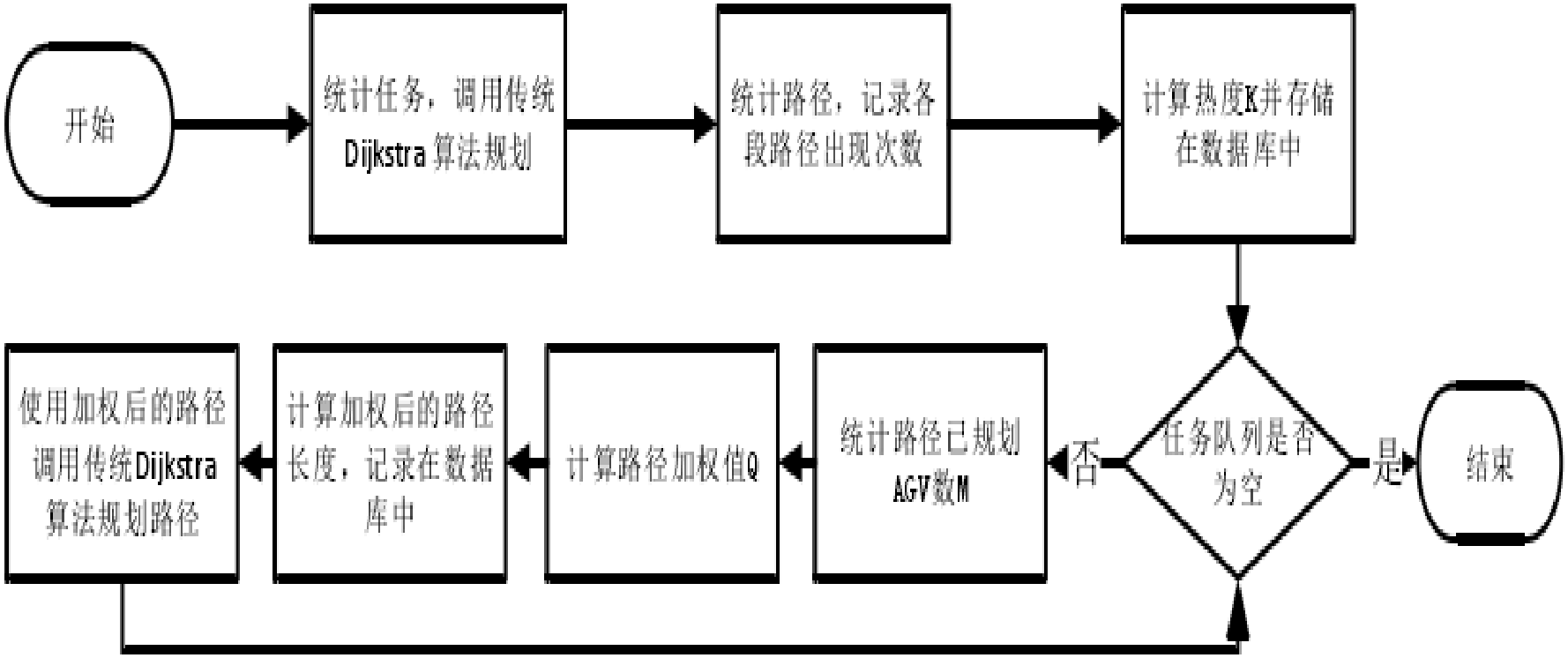
（5） 设集合 $\mathbf{M}=\{m_1,m_2,m_3,m_i\ldots\ldots$ 为系统在运行时所规划的各段路径上所规划的 AGV 数量，若 AGV 通过该段路径则将该段路径对应的 m_i 减 1；

（6） 根据路径的繁忙程度，计算路径长度加权值 $\mathbf{Q}=\{q_1,q_2,q_3,q_i\ldots\ldots\}$，其计算公式如 2-2：

$$q_i=1+\frac{m_i*(k_i-1}{n_i} \tag{2-2}$$

（7） 计算加入路径繁忙程度加权值后的路径长度：$l_i=l_i*q_i$，将加权后的路径长度记录在数据库中；

（8） 采用加权后的路径长度对 AGV 路径进行规划，后续步骤与传统 Dijkstra 算法相同。其算法流程图如图 2-10 所示。


开始
统计任务，调用传统Dijkstra算法规划
统计路径，记录各段路径出现次数
计算热度K并存储在数据库中
任务队列是否为空
否
是
结束
统计路径已规划AGV数M
计算路径加权值Q
计算加权后的路径长度，记录在数据库中
使用加权后的路径调用传统Dijkstra算法规划路径