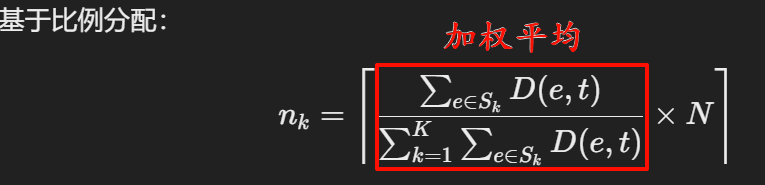
**问题二、三、四建模与解决**

**5 Allocation of campus cleaning vehicles**

区域划分完毕后，需要根据区域各自的特点进行合理的车辆分配，以达到资源利用率的最大化。首先，计算各个区域所含道路的脏度的加权平均数：（这里N=7）



由各个区域所含道路的脏度的加权平均数能够判断各区域的脏度等级，并以此给高脏度区域分配更多数量的清洁车。根据上述公式结合道路初始脏度进行计算，三个区域分配到的清洁车数量分别为3、2、2。

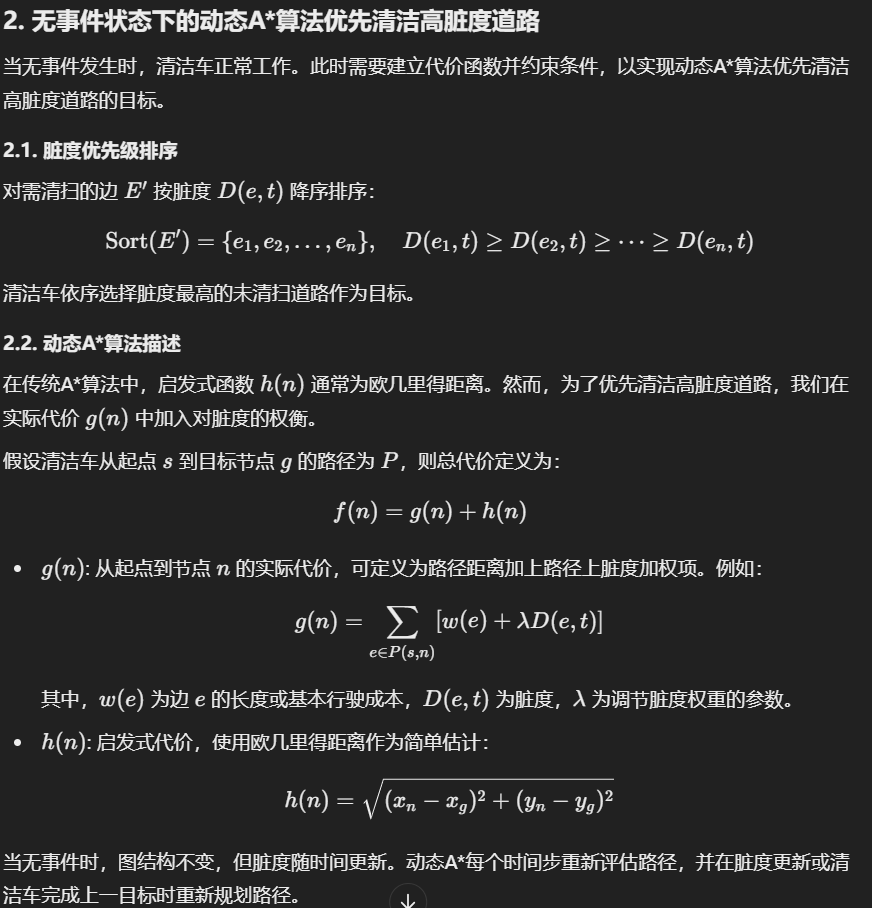
**6 Path Planning and Vehicle Scheduling**

**6.1 Path planning for campus cleaning vehicle**

车辆分配完毕后即可对各自区域的清洁车规划合理的清洁路径。

考虑到清洁车每次打扫某条道路之后，该道路的脏度都会下讲一定比例，且某区域发生突发事件（如开展活动、施工等）时，该区域附近的道路会封锁不允许清洁车进入，因此各条道路脏度的变化具有较大的随机性且清洁路径可能随时改变。为此，我们选用**Dynamic A\* Path Planning and Multi-Mode Vehicle Dispatch Model**，核心为动态A\*算法。该模型在面对实时环境变化，例如天气、活动、道路施工时，可以通过在路径规划过程中动态地更新路径以适应突发事件带来地影响，而不需要每次都重新计算全局路径，从而提高了路径规划地效率和实时性。

当无突发事件发生时，清洁车进入正常清洁模式。此时我们需要建立代价函数并约束条件，以实现动态A\*算法优先清洁高脏度道路的目标。首先，对需清扫的边E'按脏度D(e,t)降序排序，清洁车依序选择脏度最高的未清扫道路作为目标。



在传统A\*算法中，启发式函数h(n)通常为欧几里得距离。然而，为了优先清洁高脏度道路，需要在实际代价g(n）中加入对脏度的权衡。假设清洁车从起点s到目标节点g的路径为P，则总代价定义为：f(n)=g(n)+h(n)。g(n)表示从起点到节点 n 的实际代价，定义为路径距离加上路径上脏度加权项。例如（公式）

其中，w(e) 为边 e 的长度，D(e,t) 为脏度，λ为调节脏度权重的参数。

h(n):表示启发式代价，使用欧几里得距离作为简单估计：（公式）

当无事件发生时，图结构不变，而脏度随时间更新。动态A\*对于每个时间步重新评估路径，但是为了避免清洁车路径过于频繁更换，导致的清洁效率降低以及算力的浪费，我们给清洁车加上路径保护机制，即清洁车只有完成上一目标后才会重新规划路径。

路径规划的伪代码如下：

**Function dynamic\_A\_star(graph, cleaner, target\_node, λ):**

**open\_list, closed\_set, g\_score, f\_score ← priority\_queue(), set(), dict(), dict()**

**For v ∈ graph.V: g\_score[v], f\_score[v] ← infinity, infinity**

**g\_score[start], f\_score[start] ← 0, h(start, target\_node)**

**open\_list.push((f\_score[start], start))**

**While open\_list not empty:**

**\_, current ← open\_list.pop()**

**If current = target\_node: Return reconstruct\_path(came\_from, current)**

**closed\_set.add(current)**

**For neighbor ∈ graph.neighbors(current):**

**If neighbor ∈ closed\_set: Continue**

**tentative\_g ← g\_score[current] + w((current, neighbor)) + λ \* D((current, neighbor), t)**

**If tentative\_g < g\_score[neighbor]:**

**came\_from[neighbor], g\_score[neighbor], f\_score[neighbor] ← current, tentative\_g, tentative\_g + h(neighbor, target\_node)**

**open\_list.push((f\_score[neighbor], neighbor))**

**6.2 Scheduling of campus cleaning vehicles**

而当有突发事件发生时，为确保突发事件发生后产生的高脏度区域能够及时得到清扫，需要实时对突发事件进行检测：

在每个时间步 𝑡，系统根据事件集 𝑂(t) 判断当前校园环境状态：如果发生了活动/施工事件 (A)：当活动或施工事件发生时，部分节点和边临时不可用，表示在事件期间，图结构暂时删除这些节点和边；如果发生了天气事件 (W)：当下雨时，清洁车暂停清扫任务，原地等待，下雨期间不更新路径，不改变清洁车位置，仅脏度随时间按照 𝑎(𝑒)ln(t)增加，这里Δt为雨天持续时间步数。伪代码如下：

**Function check\_events(t):**

**# Input: Time step t**

**# Output: Event status O(t)**

**events ← O(t)**

**If "Event" ∈ events or "Construction" ∈ events:**

**N\_event ← affected\_nodes(events)**

**E\_event ← affected\_edges(events)**

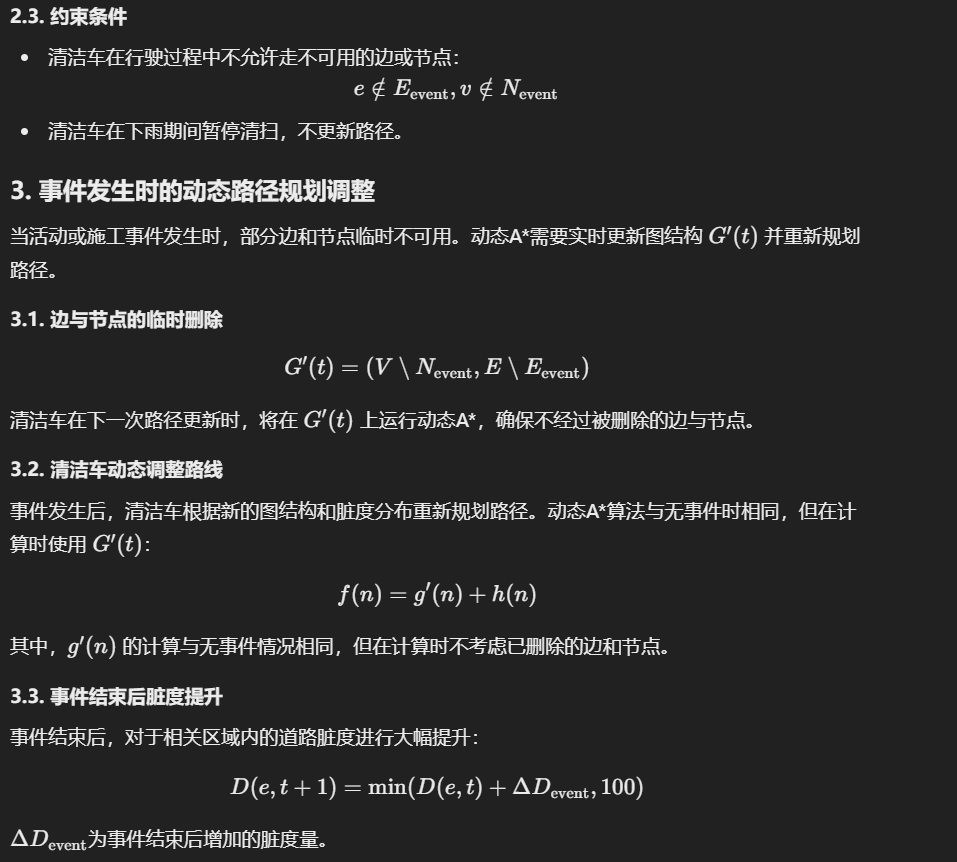
**G' ← G(t) \ (N\_event, E\_event)**

**If "Rain" ∈ events:**

**pause\_cleaning()**

**Return events**

一旦检测到校园内发生了突发事件，就需要对清洁车辆进行动态调度。需要注意的是，模型对活动、施工类突发事件以及天气类突发事件的处理方式不完全相同，下面我们分别讨论：



**活动、施工类：**

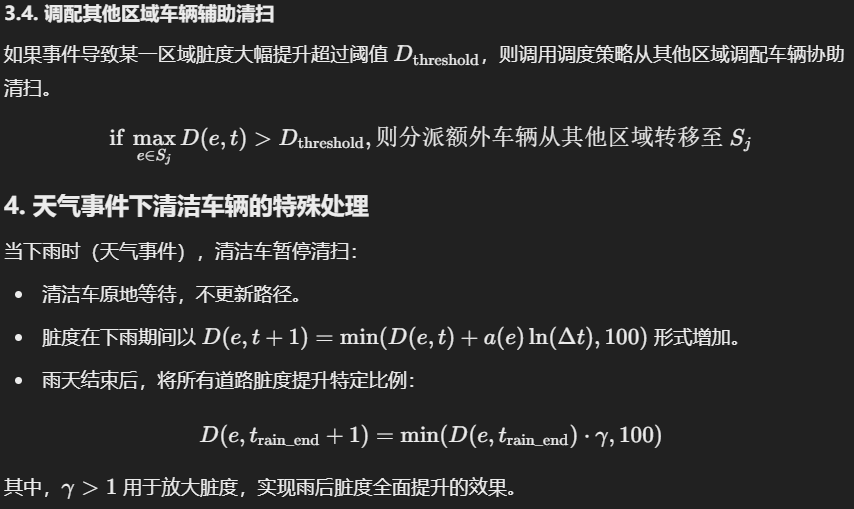
当活动或施工事件发生时，部分边和节点临时不可用。动态A\*需要实时更新图结构 G′(t) 并按照以下步骤重新规划路径：

1.边与节点的临时删除：清洁车在下一次路径更新时，将在 G′(t) 上运行动态A\*算法，确保不经过被删除的边与节点。

2.清洁车动态调整路线：事件发生后，清洁车根据新的图结构和脏度分布重新规划路径。此时动态A\*算法在计算时使用 G′(t)：f(n)=g′(n)+h(n) ，其中g′(n) 的计算与无事件情况相同，但在计算时不考虑已删除的边和节点。

3.事件结束后脏度提升：事件结束后，对于相关区域内的道路脏度进行大幅提升，ΔDevent为事件结束后增加的脏度量。

4.调配其他区域车辆辅助清扫：如果事件导致某一区域脏度大幅提升超过阈值 Dthreshold，则调用调度策略从其他区域调配车辆协助清扫。



**特殊天气类：**

当天气时间发生时，清洁车暂停清扫，并且遵从以下规则：（公式看图）

1.清洁车原地等待，不更新路径。

2.脏度在下雨期间以公式形式增加。

3.雨天结束后，将所有道路脏度提升特定比例：

其中，γ>1用于放大脏度，实现雨后脏度全面提升的效果。伪代码如下：

**Function main\_loop():**

**t ← 0**

**While system\_running:**

**events ← check\_events(t)**

**If "rain" ∈ events: update\_dirtiness\_rain\_mode(G, a, t)**

**Else: G'(t) ← update\_graph\_for\_events(G, N\_event, E\_event) if "event" ∈ events or "construction" ∈ events else G**

**For cleaner ∈ C:**

**If cleaner.path is empty:**

**path ← dynamic\_A\_star(G'(t), cleaner, select\_high\_dirtiness\_edge(G'(t), D), choose\_lambda\_param())**

**cleaner.set\_path(path)**

**Else:**

**cleaner.move\_along\_path()**

**update\_dirtiness(G, cleaner)**

**If event\_ended:**

**restore\_graph(G'(t), G)**

**increase\_dirtiness\_post\_event(G, E\_event, deltaD\_event)**

**If need\_reallocate\_vehicles(G, D\_threshold): reallocate\_vehicles(C, G, D, D\_threshold)**

**t ← t + 1**经历以上过程之后，会有额外的清洁车来到发生突发事件的区域进行清扫，以快速降低该区域的脏度，清扫结束后清洁车回到各自清扫区域进入正常清扫模式，模型恢复到前述没有突发事件发生时的状态。

················································································

补充：

优化效率可以理解为如何在规定的时间内最大限度地完成更多清洁区域的任务，并确保脏度较高的区域能够及时得到清扫。于是，将边权定义为路长和脏度的乘积，边权越高的区域，需要清扫的频率就越高。清洁车每次打扫某条道路之后，该道路的脏度都会下讲一定比例；某区域发生突发事件（如开展活动、施工等）时，该区域附近的道路会封锁不允许清洁车进入。根据上述分析可知，各条道路脏度的变化具有较大的随机性。基于这个特点，选用D\*Lite模型。该模型在面对实时环境变化，例如天气、活动、道路施工时，可以通过在路径规划过程中动态地更新路径以适应突发事件带来地影响，而不需要每次都重新计算全局路径，从而提高了路径规划地效率和实时性。

根据问题地具体背景，采用以下方面来应用D\*lite算法：

1.脏度优先：动态A（D\* Lite）算法的更新使得清洁车能够实时响应脏度变化，并优先选择脏度较高的道路进行清扫，确保脏度较高的区域得到优先处理。

2.实时更新：当检测到天气变化或道路施工时，可以在现有路径基础上对路径进行局部更新，而无需从头开始计算。

3.路径再规划：在清洁车的路径规划过程中，动态A\*算法可以有效处理突发事件，并且使得清洁车能够实时调整路径以避免拥堵或封闭道路。

4.效率优化：D\* Lite通过局部更新路径，避免全局路径的重计算，提高效率，适合用在实时系统中。