地下冲洗点位自动布置算法开发文档

1 背景介绍

本算法的目标是：根据一定的规则，为地下车库自动生成一套冲洗点位方案，为用户的后续加工提供基础。

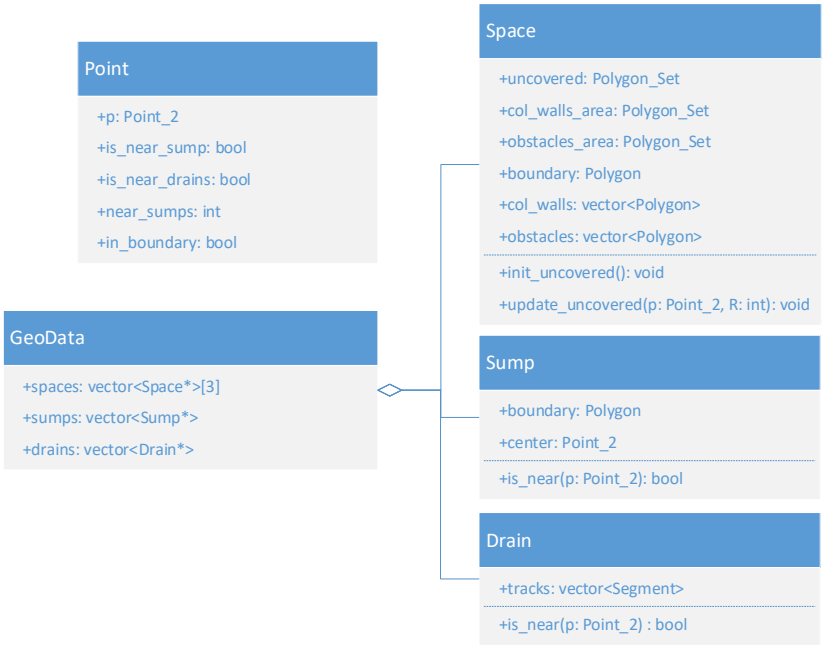
地下车库由三类空间（Space）构成：建筑空间（隔油池、水泵房、垃圾房等）、停车空间和其他空间。每个空间实例用封闭的多边形表示，但是可能会有“洞”的情况。建筑空间是必须布置的空间，停车空间是可以布置的空间，其他空间根据参数决定属于可以布置的空间还是不可布置的空间。必须布置的空间中，冲洗点根据参数决定是否能保护可以布置的空间和不可布置的空间；可以布置的空间中，冲洗点无法保护必须布置的空间，但可根据参数决定是否能够保护不可布置的空间。

每个空间实例当中存在四类元素：墙（ShearWall）、柱（Column）、障碍（Obstacle）和排水设施（DrainageFacility）。其中墙、柱、障碍和排水设施中的集水坑与地漏都用封闭多边形来表示，排水设施中的排水沟用多段线表示。一个墙、柱和障碍实例不会跨越两个或多个空间。所有冲洗点只能布置在墙和柱的边缘，且要避开所有障碍。在停车空间当中，冲洗点要优先布置在集水坑或地漏附近，其次则是排水沟附近，再次是孤立墙和孤立柱的边缘，最后是空间轮廓上。同一个集水坑或地漏附近视情况可布置零个或一个冲洗点。除停车空间以外的其他空间则不考虑排水设施。

在上述要求和约束的背景下，设计算法自动完成布置冲洗点位，使得在给定的冲洗半径之下没有盲区（除非盲区不可避免），且布置的冲洗点位尽量要少。注意冲洗点提供保护时可以穿透所有的墙、柱、障碍和排水设施，不需要考虑遮挡效应。

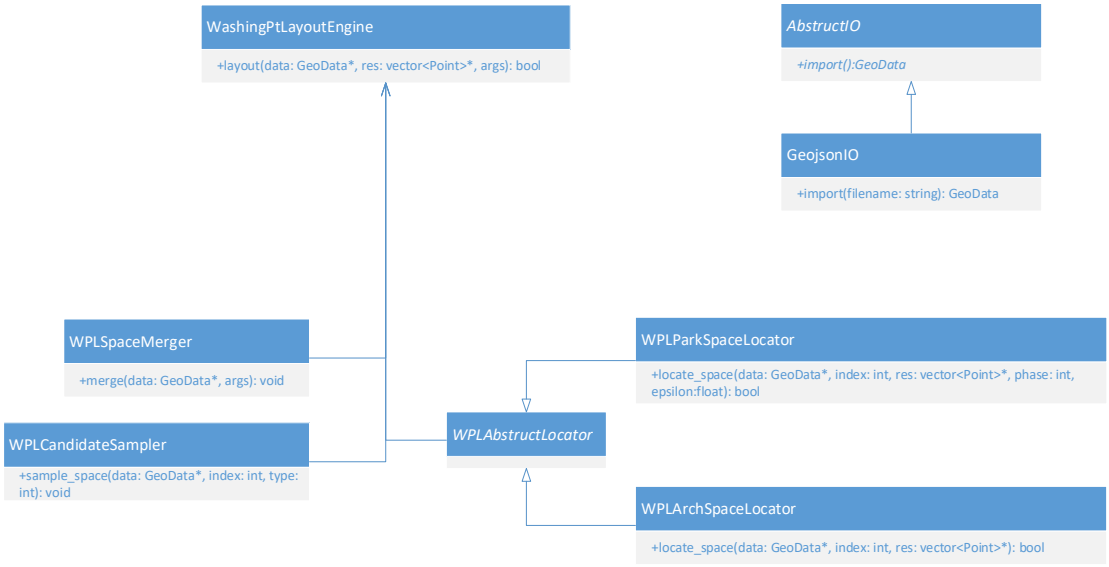
2 数据结构与算法接口

该算法采用 GeoData 类来存储重要的空间数据。相关类图见下。



GeoData 维护了建筑空间列表、停车空间列表和其他空间列表，也存储了整个空间中所有的集水坑和排水沟，区分这两者的关键是看第一个点是否与最后一个点相同。不同类型的空间共享同一个类 Space，它存储了该空间的边界轮廓，以及所有在该空间以内的墙、柱、障碍，并在算法执行过程中用方法 init_uncovered 和 update_uncovered 去维护一个未被保护的区域集合 uncovered。集水坑 Sump 和排水沟 Drain 各用一个类来表示，里面主要存放各自的关键信息，以及判断一个点是否靠近它们的方法。候选点类 Point 除了保存坐标以外，还预留了一些关键属性，包括是否靠近集水坑与排水沟，靠近集水坑的数目，以及是否位于空间轮廓上。在可布置空间中，所有候选点将根据这些属性按照优先顺序逐步选取和布置。

为了将原始数据封装成 GeoData 以供算法处理，本项目设计了 IO 基类 BaseIO，在开发过程中，采用该基类的一个继承类 GeojsonIO，该类负责从 geojson 文件中读取数据，然后封装成 GeoData 返回。后续的应用可以通过再继承 BaseIO 类，实现以 dwg 文件为数据源的封装操作。



为了使算法的具体实现与数据结构分离，本项目设计了多个算法类，其中 WashingPtLayoutEngine 是充当了枢纽的角色，是实际应用中的顶层类，它的 layout 方法接收 GeoData 的指针 data、布置位点向量（初始为空）的指针 res 和一些参数的指针 args，返回一个布尔值，表示空间是否按要求被完全保护，同时布置位点向量 res 被填充。在 layout 方法中，需要调用一些重要的算法子模块，这些子模块也用类的方式各自封装起来，其中包括 WPLSpaceMerger（负责归并所有可布置空间和所有不可布置空间，方便后续算法统一处理）、WPLCandidateSampler（负责对必须布置空间或可布置空间采样过滤候选点）、WPLAbstractLocator（负责选取候选点进行布置）。其中 WPLAbstractLocator 是抽象基类，考虑到产品的具体需求，继承出两个类：WPLArchLocator 和 WPLParkLocator，它们分别负责对一个必须布置空间布点和对一个可布置空间布点。

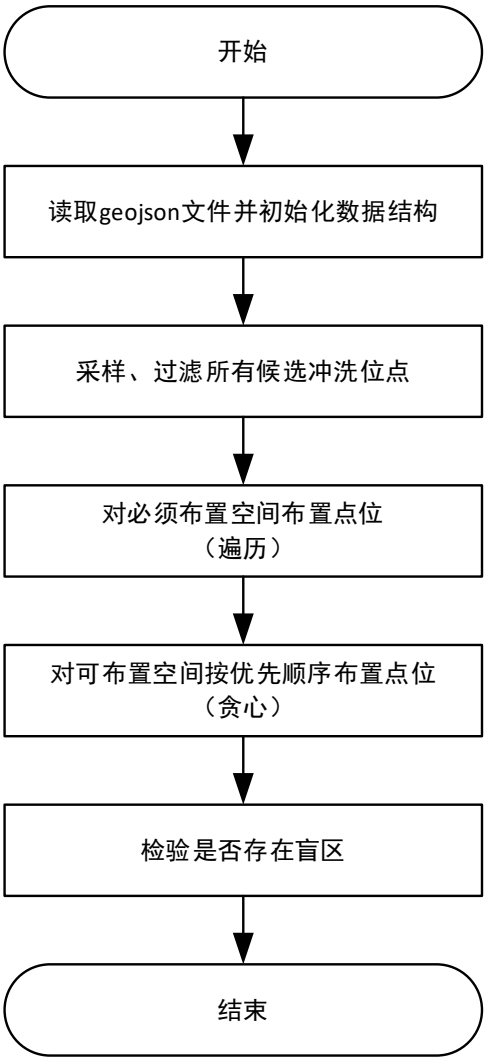
类图中的方法只列出了主干方法，一些辅助性的方法没有列出。

3 算法思路

3.1 核心思想

1. 变连续为离散，通过采样得到有限个候选位点，这样可以大大简化选择布置位点的过程；
2. 按照空间类型和优先级梯次布点，并与贪心算法相结合，尽量减少各个冲洗点保护区域的重叠；
3. 利用平面图形的集合操作来引导布点和检验效果，显著降低了编程和调试的难度。

3.2 流程图



3.3 读取文件并初始化数据结构

算法的第一步就是读取 geojson 文件，然后对数据结构进行初始化。总共需要遍历两次 geojson 文件。这一步通过调用 GeojsonIO 类的方法来实现。

第一遍遍历文件时，将所有的空间按照空间名称添加进对应的向量里。对于所有的排水设施，通过比较第一个点与最后一个点是否相等，来判断是按照集水坑还是排水沟来处理，然后添加到对应列表里。第二遍遍历文件时，对于每一个墙、柱和障碍，找到其所在的空间，更新对应的空间数据。

随后 WashingPtLayoutEngine 会调用 WPLSpaceMerger 类的方法进行空间的归并。归并的主要操作有：归并墙柱向量、归并障碍物向量、归并和更新未保护区域集合等。所有停车空间归并到一个大的空间里，所有其他空间也同理操作，建筑空间不归并。将所有障碍的区域相加，形成全局障碍的区域，区域的加法用平面多边形的并集实现。更新后的未保护区域，等于空间轮廓围成的区域，减去全局障碍的区域，再减去该空间中所有墙和柱的区域，区域的减法用平面多边形的差集实现。

3.4 采样与过滤候选位点

经过初始化之后，GeoData 中的空间结构向量应该会包含一个可布置空间、一个不可布置空间（可选）和一系列必须布置空间。在 WashingPtLayoutEngine 当中，对每个空间（不可布置空间除外）调用 WPLCandidateSampler 类的方法进行采样和过滤。

采样：对于每个可布置空间或必须布置空间，取到所有隶属于该空间的墙和柱，在墙和柱的每一条线段上采样得到若干点。采样的策略待定，不过初步的想法是根据线段的长度来选择如何采样：长度太小的，不采样；长度稍大但不超过 1m 的，只取中点；长度大于 1m 的，平均每隔约 0.5m 采一个点。线段的端点也纳入候选，但要注意避免重复采样。

过滤：对于采样得到的每个点，检验它是否落在全局障碍区域中，如果是则抛弃不要。将过滤后的所有候选点构造成结构后添加进对应空间的候选点向量中。候选点是否来自于孤立的墙和柱是一个关键属性，需要更新。对于可布置空间的候选点，还需要做一步额外的操作，也就是遍历全局所有的集水坑和排水沟，计算该候选点是否位于它们附近。如果是，则更新候选点对应的属性。

之后所有布点的步骤，都只会从这一步选出来的位点当中进行选择。过低的采样频率会导致无法对所有区域提供保护，但过高的采样频率会加重计算负担，也没有必要。所以，使用合理的采样频率非常关键。

3.5 对必须布置空间布置点位（含近似圆处理）

考虑到每个空间都是一个封闭的多边形，而且必然存在一个冲洗点可以保护整个空间，所以问题可以简化为：找到一个候选点，使得它与多边形各顶点的距离均小于保护半径。

在算法当中，这一步是采用遍历空间中所有候选点的方法，找到候选点中与多边形各顶点的最大距离最小的那一个，布置该空间的冲洗点。如果这个点仍然无法保护到整个空间，就与假设矛盾，程序会报错。这一部分的具体代码拟在 WPLArchLocator 类中实现。下面是这一部分的算法伪代码，每个必须布置空间都按此处理。

1 Locating in a must-locate space.

Input:

List of the space boundary vertices, V;
List of candidate points, C;
Protection radius, R;
Uncovered regions, U;

Output:

The chosen candidate point, c^*;
function MUSTLOCATESPACE(V, C, R, U)
    $c^* \leftarrow \min_{c \in C}$ MAXDIST(c, V);
    $d \leftarrow$ MAXDIST(c^*, V)
    assert $D < R$;
    update uncovered regions U using c^*;
    return c^*;
end function

function MAXDIST(c, V)
    return $\max_{v \in V} ||c - v||_2$;
end function

每当布置一个冲洗位点时，都会同步更新当前未保护到的区域。注意除了要更新位点所在空间的未保护区域，还需要更新可布置空间和不可布置空间的未保护区域（可选）。后文所述的更新未保护区域根据参数的设置同理操作。

上述算法只考虑了必须布置空间内的候选点保护它所在空间的情况。然而，当不可布置空间需要被保护、而且只能通过必须布置空间中的点来保护时，上述算法就不起作用了。为了让必须布置空间的候选点既能保护所在空间又能保护到所有的不可布置空间，暂时拟定下面的算法：

假设有 M 个必须布置空间，且已对每个必须布置空间采得候选点集合 S_k(k=1…M)，另有归并成一个大空间后的不可布置空间 U。

1. 对候选点集合 S_k(k=1…M)进行筛选，留下那些可以完全覆盖第 k 个必须布置空间的点，构成新集合 S_k^*(k=1…M)；**（先保证能保护自己的空间）**

2. 拉通遍历 S_k^*(k=1…M)的每一个候选点 c，形成圆形保护区域与不可布置空间做交集 Circle(c,R)∩U，找到交集面积最大的那个点（记为 c*，且假设 c*∈S_i^*），选取并布置 c*，更新不可布置空间未保护区域，并且删除它所在的集合 S_i^*；**（用保护面积大小做贪心）**

3. 重复第 2 步（每次迭代，参与遍历的候选点集合都会少一个），直到删除完所有的候选点集合；**（布置完所有的必须布置空间）**

4. 检查不可布置空间 U 是否被完全保护，返回结果。

该算法需要计算平面图形交集的总面积，而 CGAL 库对具有弧形的平面图形没有计算面积的 API，所以在本项目中，**所有的圆形都用多边形来近似**，使得布尔运算后的结果没有弧形，而 CGAL 库对这种集合是有办法间接计算总面积的。目前采取的近似精度是，保证近似之后的多边形，每条边长为 500 毫米。

3.6 对可布置空间布置点位

在可布置空间中，点位需要按照集水坑附近、排水沟附近、孤立墙柱位点和空间轮廓位点的优先顺序来选取与布置。这四处区域都采用贪心算法来处理，代码框架基本相同，但是在初始化和某些策略上，互相略有不同。

统一的代码框架：不管是集水坑附近、排水沟附近还是其他位置，代码的输入都包含待布置的候选点列表 Cin，已经布置的点列表 Cout。如果初始的 Cout 列表为空，则需要按照某种首点生成策略，从 Cin 中选一个点布置，并更新 Cout 和未保护区域。随后每次迭代中，首先按照某种终止循环策略决定是否跳出循环。若不跳出循环，则从 Cin 里选择出与 Cout 中各点的最短距离最大的点并将其移出 Cin，然后按照某种候选点过滤策略决定要不要布置该点。如果要布置，那就更新 Cout 列表和未保护区域，否则就直接抛弃该点。跳出循环之后，检验此时的未保护区域是否为空，并返回该布尔值。

四个阶段的区别：具体体现在代码框架中的 Cin 初始值、Cout 初始值、首点生成策略、候选点过滤策略与终止循环策略上面。详见下表。

	集水坑附近	排水沟附近	孤立墙柱位点	空间轮廓位点
Cin 初始值	所有位于集水坑附近的候选点	所有不在任何集水坑附近，且位于排水沟附近的候选点	所有不在任何集水坑附近，且位于孤立墙柱边沿的未选的候选点	所有不在任何集水坑附近的未选的候选点
Cout 初始值	空	布置完集水坑附近后的结果	布置完排水沟附近后的结果	布置完孤立墙柱位点后的结果
首点生成策略	位于最多数量集水坑附近的候选点	随机选取候选点	随机选取候选点	随机选取候选点
候选点过滤策略	该点的保护区域与当前未保护区域交集不为空，且候选点最大最短距离不小于某阈值	该点的保护区域与当前未保护区域交集不为空，且候选点最大最短距离不小于某阈值	该点的保护区域与当前未保护区域交集不为空，且候选点最大最短距离不小于某阈值	该点的保护区域与当前未保护区域交集不为空
终止循环策略	未保护区域为空，或候选点最大平均距离小于某阈值，或 Cin 为空	未保护区域为空，或候选点最大平均距离小于某阈值，或 Cin 为空	未保护区域为空，或候选点最大平均距离小于某阈值，或 Cin 为空	未保护区域为空，或 Cin 为空

阶段 0：对于集水坑附近，只用考虑位于集水坑附近的候选点；而且为了让布置位点尽量少，所以让第一个点就能涵盖尽可能多的集水坑；为了保证一个集水坑附近最多一个点，也为了保证程序不拘泥于集水坑附近，所以设置了一个阈值，要求待布置的候选点与 Cout 中各点的平均距离必须大于该阈值，否则舍弃该候选点并跳出循环（当阈值大于集水坑的附

近直径时，就可严格保证一个集水坑附近最多只布置一个位点）；如果跳出循环之后未保护区仍不空，则尝试去排水沟附近寻找新的布置位点。

阶段 1：对于排水沟附近，Cin 列表中的候选点除了必须在排水沟附近之外，还不应该在集水坑附近；上一步输出的 Cout 继续保留以供贪心计算；虽然没有对同一个排水沟附近的点位数有明确要求，但是为了防止程序拘泥于排水沟附近，仍然设置一个阈值（该阈值可以与上一步中的阈值不相等），候选点过滤策略和终止循环策略与集水坑附近相同；如果跳出循环之后未保护区仍不空，则尝试去其他区域寻找新的布置位点。

阶段 2：对于孤立墙柱位点，Cin 初始时包含上一步 Cin 剩下的，再加上其他区域中位于孤立墙柱上的所有候选点（也就是说，排水沟区域里剩下的候选点还有机会被布置上）；上一步输出的 Cout 继续保留以供贪心计算；仍然设置一个阈值（该阈值可以与上两步中的阈值不相等），候选点过滤策略和终止循环策略与集水坑和排水沟附近相同；如果跳出循环之后未保护区仍不空，则尝试去空间轮廓位点寻找新的布置位点。

阶段 3：对于空间轮廓位点，Cin 初始时包含上一步 Cin 剩下的，再加上所有位于空间轮廓上的点（这一步中，排水沟区域和孤立墙柱上的候选点都有可能再次被选到）；上一步输出的 Cout 继续保留以供贪心计算；和前三个步骤都不同的是，该步骤是最后一个可以布点的部分，所以不再有阈值的限制，从而让程序尽可能地消灭所有未保护区域，直到所有候选点用光为止。

考虑到阶段 0 到阶段 2 除了首点生成策略以外完全一致，而且阶段 3 可以看成前三个阶段中阈值等于零的特殊情况，所以这四个阶段可以用一段统一的代码来处理，开发时具体将在 WPLParkLocator 类中实现，最后由 WashingPtLayoutEngine 用不同参数调用四次，即可完成在可布置空间的布点操作。每个阶段的算法伪代码见下。

2 Locating in a could-locate space, each phase.

Input:

List of candidate points, C_{in};
List of chosen points, C_{out};
Uncovered regions (with spaces which can be protected), U;
Distance threshold, ϵ;
(For phase 0 to 2, $\epsilon >$ candidate sample rate; for phase 3, $\epsilon \leftarrow 0$)

Output:

C_{in}, C_{out}, U;
Whether all the regions are covered, *succ*;

```
function GreedyKernel(C_in, C_out, U, ε)
    if C_out is empty then
        generate c* according to First-Point-Generating-Policy;
        d ← +inf;
        assert U ∩ Circle(c*, R) ≠ ∅
        remove c* from C_in;
        add c* into C_out;
        update uncovered regions U using c*;
    end if
    while (U is not empty) ∧ (d ≥ ε) ∧ (C_in is not empty) do

        /* find the next candidate point */
        c* ← arg max_{c_i ∈ C_in} MinDist(c_i, C_out);
        d ← MinDist(c*, C_out);
        remove c* from C_in;

        /* check whether to choose the point */
        if (U ∩ Circle(c, R) ≠ ∅) ∧ (d ≥ ε) then
            add c* into C_out;
            update uncovered regions U using c*;
        end if

    end while
    succ ← U is empty;
    return C_in, C_out, U, succ;
end function

function MinDist(c, C_out)
    return min_{c_j ∈ C_out} ||c − c_j||_2;
end function
```

4.潜在问题

建筑空间（即必须布置空间）的布点必须满足的条件：一个建筑空间中必然存在一个合法的候选点，它可以覆盖到整个空间。如果建筑空间太大，或者保护半径太小，都会影响该条件的成立。

候选点的采样率会对布点结果产生重要的影响，如果最后的布点策略无法覆盖整个空间，则有可能是采样率过低导致的。

在集水坑附近布点时，一个点布置好，其他跟它位于同一个集水坑附近的所有候选点都不能再布置。另外，阈值的限制也可能将两个位于不同集水坑附近、但彼此相距较近的两个点排除掉其中一个。这些情况都有可能会导致最后保护区域无法覆盖整个空间。

对于排水沟附近及以后的三个阶段，如果 Cout 列表为空，则需要随机生成第一个布置点，这会给结果带来不确定性。

5.开发配置

编程语言：C++

编程环境：Visul Studio 2019

所需软件包：CGAL5.2（用 vcpkg 安装到本地，生成 NuGet 包）

撰写人：郑友怡团队 殷俊麟

撰写时间：2021 年 3 月 25 日星期四