同济大学
TONGJI UNIVERSITY

硕士学位论文

(学术学位)

暖通空调系统自动化设计方法研究

姓　　名：陈喆

学　　号：1730215

所在院系：机械与能源工程学院

职业类型：工程

专业领域：供热、供燃气、通风及空调工程

指导教师：许鹏 教授

副指导教师：

二〇二〇年三月

同济大学
TONGJI UNIVERSITY

A dissertation submitted to

Tongji University in conformity with the requirements for

the degree of Master

Research on HVAC System Design Automation Process

Candidate: Chen Zhe

Student Number: 1730215

School/Department: School of Mechanical Engineering

Discipline: Civil Engineering

Major: Heating, Gas Supply, Ventilation and Air-conditioning Engineering

Supervisor: Prof. Peng Xu

March, 2020

暖通空调系统自动化设计方法研究

陈　喆

同济大学

学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名： 

2020 年 3 月 9 日

同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：陈喆

2020 年 3 月 9 日

摘要

近年来，随着人工智能领域的不断发展，各种基于人工智能的算法也得到了一些应用。一般来说，要在某一个领域推广人工智能，需要两个步骤，第一个是大量数据的获取，第二个是基于这些大量的数据来实施人工智能的算法。在暖通空调系统设计领域，我们提出了非监督式学习的暖通空调系统自动化设计流程。该流程包括 BIM 模型检验、自动分区及负荷计算、空调系统类型和各环路（loop）的定义、设备和布管四个模块，其中前序工作我们以及完成了空调系统类型和各环路的定义中对建筑的类型、平面、分区结果的综合判断得到适合该建筑的空调系统类型和各环路（loop）的定义，本文的主要研究内容是将完成自动分区及负荷计算模块中 BIM 模型转化和模型转化和负荷自动计算、以及设备和布管中的初步的空间定位（二维管路的敷设算法）在负荷自动计算过程中，我们介绍了将 gbXML 文件中的几何信息、建筑结构信息等写入到 EnergyPlus 负荷计算文件的方法；在空间定位方面，我们给出了不同类型平面的风口布置算法；在布管连接方面，我们提出了基于最小生成树的房间—走廊主管管路连接算法和基于 A*算法的最小拐点障碍规避路径算法。

关键词：空调系统设计，基于 BIM 的负荷自动计算，管路算法，最小生成树遍历，路径寻优算法

ABSTRACT

In recent years, with the continuous development of the field of artificial intelligence, various artificial intelligence-based algorithms have also been applied. Generally speaking, to promote artificial intelligence in a certain field, two steps are required. The first is to obtain a large amount of data, and the second is to implement artificial intelligence algorithms based on these large amounts of data. In the field of HVAC system design, we propose an unsupervised learning process for the automated design of HVAC systems. The process includes four modules: BIM model inspection, automatic partitioning and load calculation, air conditioning system type and definition of each loop, equipment and pipework. Among them, we have completed the prerequisite work and completed the definition of the air conditioning system type and each loop. The comprehensive judgment of the building type, plane, and partition results in the building obtained the type of air conditioning system and definition of each loop suitable for the building. The main research content of this paper is to convert the BIM model in the automatic partition and load calculation module and Model transformation and automatic load calculation, as well as preliminary spatial positioning in the equipment and pipe laying (two-dimensional pipeline laying algorithm) During the automatic load calculation process, we introduced the writing of geometric information, building structure information, etc. in the gbXML file To EnergyPlus load calculation file method; in terms of spatial positioning, we give different types of air outlet layout algorithms; in terms of pipe connection, we propose a room-corridor main pipe connection algorithm based on the minimum spanning tree and A* Algorithm of the minimum inflection point obstacle avoidance path algorithm.

Key Words: HVAC system design, automatic building load calculation based on BIM, pipeline algorithm, minimum spanning tree traversal, path optimization algorithm

目录

第 1 章 引言

1.1 国内外研究现状

1.1.1 建筑负荷计算

在对建筑进行暖通空调系统的设计之前，需要对建筑的全年负荷特性进行计算，粗略的计算可采用指标估算法，细致的计算可采用软件计算模拟的方法。模拟过程可采用白箱、灰箱或黑箱模型。白箱模型如 EnergyPlus、eQUEST、DeST[1-4]等通过对建筑的围护结构进行详细描述，以及人员、设备、照明等影响建筑得热的因素进行输入，最终计算引擎通过其物理方程迭代计算得到一个建筑的全年逐时负荷。黑箱模型则通过对多个输入变量和对应的输出变量（即建筑的负荷）通过人工神经网络(Artificial Neural Network, ANN), 多元线性回归(Multiple Linear Regression, MLR)，支持向量机(Support Vector Machine, SVM)[5-11]进行训练，得到一个黑箱模型可用于计算新的输入变量对应的负荷输出。

尽管白箱模型的输入参数较多且计算过程往往较长，但由于白箱模型较其他模型有更强的可理解性和可校准性，许多学者不断改进白箱模型使其可更加适用于不同类型建筑的快速模拟，以期实现同灰箱模型和黑线模型的自动化程度。Dogan Timur 等[12, 13]还提出了一种可将基于白线模型的建筑能耗模拟(Building Energy Modeling, BEM)快速应用于大批量建筑能耗模拟的计算，可用于城市规模的建筑能耗模拟。在建筑模拟的白箱模型计算过程中有时候需要人为去给定建筑的内区(Interior Zone)和外区(Perimeter Zone)，这给自动化的白箱模型能耗模拟提出了挑战，Dogan Timur 等提出了可多区能耗模拟的自动热区建立算法，该算法能够基于建筑物的外轮廓通过一种 straight-skeleton subdivision 的算法自动分割出建筑的内区和外区。为了在设计阶段获得更加精确建筑能耗模拟结果，R. Brahme 等[14]使用基于同源性（Homology）的映射方法生成了建筑对应的 HVAC “代理系统”，这一“代理系统”能够较好的融入能耗模拟的过程并提供更好就计算结果，与传统能耗模拟过程中的 HVAC 系统建模相比，减少了大量的输入参数。Bazjanac Vladimir[15]提出了一种基于 IFC 模型的建筑能耗自动化计算框架。工业基础类（Industry Foundation Class, IFC）是由国际互通操作联盟（International Alliance Interoperability, IAI）定义的一种数据格式，用于存储信息及模型的数据，用于各个软件、应用之间交换数据。IFC 定义了很多的类别，每个类别有很多的

属性来描述它，这其实就是类的概念 Class。BIM 是一种面向对象的工程设计方法。IFC 作为一种数据结构，便于各个厂商的产品之间交换数据。该框架适用于建筑的使用阶段，通过从 IFC 格式中提取出建筑能耗模拟所需要的信息，如建筑的几何信息、结构信息、材料信息、机电系统设备信息、运行表数据等，最后需要人为添加 IFC 中缺失的信息以及 EnergyPlus 的模拟参数，最后统一写入至 IDF 文件使用 EnergyPlus 进行模拟得到建筑能耗模拟的结果。同样为了解决从 BIM 到 BEM 的自动化过程，Giannakis G I[16]在完成从 IFC 到 EnergyPlus 的 IDF 文件格式转化后，使用了共同边界相交投影算法(Common Boundary Intersection Projection)，实现了建筑几何信息完整性的检查和校验。梁思雨[17]为了解决 IFC 和 gbXML 导出时存在的构建缺失问题，将两者结合起来，根据预设的 idf 数据模板以及映射关系表来实现根据 IFC 对 gbXML 中可能存在的缺失构建的补充。陈远等[18-20]对比评价了 IFC 和 gbXML 在 Ecotect 模拟软件中的可交互性，指出了不同 BIM 软件之间存在的“信息孤岛”问题。黄多娜[21]对比评测了不同格式与 IES(VE)的可交互性，包括 IFC、gbXML 和 DXF 等。孙红三等[22]开发了 gbXML 与 DeST 能耗模拟软件之间的接口，林佳瑞等[23]则对 IFC 到 gbXML 的自动转换做了研究。王鸿鑫等[24]对目前主流的基于 BIM 的建筑能耗模拟软件做了对比和评测，主要包括 Green Building Studio、OpenStudio、HY-EP、PKPM-Energy、IES(VE)等，这些软件的模拟内核都是基于 EnergyPlus、DOE-2、Apache 等，主要通过内置的格式转换的插件将 BIM 的格式（如 gbXML、Revit、IFC 等）转化为能耗模拟中需要的格式，但这些软件都无法做到自动化的能耗模拟，还需要人工导入 BIM 文件以及手动添加一些缺失的信息。

1.1.2 建筑空调系统辅助设计工具

Ellis M W 等[25]总结了不同类型的集成建筑设计工具(Integrated Building Design Systems, IBDS)的发展，涵盖了建筑热模型能耗模拟、系统选择和模拟、系统设计和选型、计算机辅助设计(Computer-Aided Design, CAD)、建筑声环境计算、建筑规范分析等。作者提出了这一系列的软件的使用潜力巨大，但是却未得到长足的发展，这主要是因为现有的集成建筑设计工具的存在的一系列问题。其中最主要的问题是这一系列软件通常设计的很复杂因此设计师难以使用，有的软件又十分昂贵，计算时间较长因此很难得到广泛的应用。

许多学者将人工智能的算法应用于建筑以及空调系统的设计过程中。Sönmez Nizam Onur[26]对近几年来将算法应用于建筑自动设计的算法做了回顾，但是大多数算法都是基于某些案例的建筑进行自动设计，还没有一个能够将建筑

的设计流程自动化的完整算法，但是这些研究给将来人工智能在建筑设计中的应用提供了良好的研究基础。Lee W L[27]对香港的 50 栋商业建筑共计 186 台冷机进行性能的统计分析，发现冷机的容量选择和部分负荷率对冷机的能效的影响有限，提出了一种对冷冻水系统能效进行评价的方法，该研究还表明更多数量的冷机有助于减少冷冻水系统的能耗，最高可达 9.5%，给冷冻水系统的自动化设计流程提供了一定的指导。Kang Yingzi 等[28]提出了一种新的冷机选型的算法，该算法考虑了冷机参数的不确定性，例如不同可选冷机的名义制冷量，冷机在名义工况和实际工况下的制冷量差异，冷机的生命周期成本(Life Cycle Cost, LCC)全年不满足小时数等，实际案例表明该算法能有效降低冷机选型的名义制冷量大小，且最低能够降低 22.51%。Sun Yuming 等[29]对 HVAC 的选型的不确定性进行了研究，并提出了一种能够用于 HVAC 设备选型的不确定分析(Uncertainty Analysis, UA)和敏感性分析(Sensitivity Analysis, SA)框架，其中不确定性分析替代了传统暖通空调系统选型过程中常用的安全放大系数，避免了因为设计冗余而造成的投资成本增加和运行能耗增加。Huang Pei[30]使用多重标准决策(Multiple-Criterion Decision)和不确定性分析对空调系统的设计做了研究，并提出一种能够在设计阶段对 HVAC 系统的性能进行评价。

1.1.3 管路布置算法

在管路布置算法方面，目前国内外已有基于规则的空调系统自动化设计的相关研究及成果，较为成熟的应用领域主要有船舶工业和航空航天领域等，目前空调系统相关的研究的方向和内容主要是空调水系统的自动设计和管路布局。

Benachir Medjdoub 等[31]通过面向对象的编程方式，通过案例学习的方法实现了，在 CAD 上实现了低层建筑热水系统的选型和布局设计的自动生成，这一自动化过程需要设计者参与一部分设计过程，但也在一定程度上增加了设计的灵活性。此外，研究者还开发了 3D 可视化的设计界面，提高了该设计工具在行业内的可用性。但是该方法具有一定的限制条件[32]，例如，该方法生成的解仅限于几何条件上的定义，不能将一些暖通空调领域的专业知识加以学习以供参考。为了改进该算法，该学者将其案例学习的案例库加以启发式知识的描述，以达到基于实际使用场景进行系统设计生成的目的。Stanescu Magdalena[33]使用演进算法(Evolutionary Algorithm, EA)对 HVAC 系统的几个设计参数进行了优化设计：系统分区的个数、建筑空调系统类型的个数等。

为了解决航空器领域中的管路布置问题，Qiang Liu 等[34-37]提出了一种能使得管路通过生成直角形状的弯头的方法实现自动布置的算法，该算法通过将该问

题描述为 RSMTO (Rectilinear Steiner Minimum Tree with Obstacles)，并通过一些工程学的约束条件加以描述，但是该问题是一个 NP-hard 问题，作者使用粒子群算法(Particle Swarm Optimization)，最终能够在管路布置过程中实现障碍物的规避以及最短路径的近似解。后来该学者还提出了一种基于曼哈顿可视图(Manhattan Visibility Graph, MVG)的直角管路生成算法[38]，相比于原来的 RSMTO 算法，该算法能够有效地减少网格上的节点个数到 n（n 为起终点的个数和障碍物的顶点个数之和），因此大幅较少了搜索的范围。Wang Chengen[39]提出了一种用于航空发动机的管路生成算法，该算法通过投影来判别障碍物，并将其作为规避障碍的启发式算法。Changtao 等[40]使用遗传算法(Genetic Algorithm, GA) 实现了三维空间内基于一系列限制条件（尤其是距离限制条件）的管路布置算法，该算法首先将三维空间转化为三维的网格，然后使用提出的适用于管路生成算法的遗传算法的规则，实现弯头数量的减少和障碍物的规避。Qu Yanfeng[41]提出了一种可连接多个末端支路的管路生成算法，该算法基于三维连接图生成管路连接图，采用蚁群算法对管路进行寻优，实现了障碍物的有效规避和近似最优解的获得，最后在航空发动机的模型上对该算法进行了有效性验证。Sui, Haiteng[42]定义了管路和支路的编码，使用改进遗传算法对船舶的管路进行优化设计。Geisberger Robert[43]对交通网络中的路径规划问题中的转弯个数做了约束，算法中将各个节点的弯头代价存入矩阵中，实现了高效的搜索。

以上学者均在不同领域给出了相应的管路布置算法，但均不能直接应用于暖通空调系统中的系统设计，这是因为在系统设计时需要考虑管路。

Jin-Hyung Park 等[44]研究了室内水系统管路寻优的方法，通过基于几何特征模式识别的算法生成最终管路的候选解，然后综合考虑其他非集合因素如室内障碍、可操作性、材料成本、安装费用、阀门布置等因素综合选出最优解，最后通过轮船引擎室的管路设计作为案例验证了该方法。Medjdoub Benachir[45]使用基于案例推理(Case-Based Reasoning, CBR)的算法对空调系统进行设计。一般地，在基于案例推理的流程中，包含由 Retrieve、Reuse、Revise 和 Retain 组成的“4R”过程以及机器学习的对象数据库。该学者将不同集合类型的房间放到归到不同的案例学习数据库中，最终实现了在建筑平面内风机盘管和风口的初步连接。

同样是面向建筑的路径规划，在除了管路外的其他领域，有学者做出相关的研究。Teo Tee-Ann 等[46]使用 IFC 实现了建筑物室内的路径规划，该路径规划可用于在发生紧急情况时候的楼内人员疏散以及室内的导航路径规划，论文中使用了一种多目的几何网络模型(Multi-Purpose Geometric Network Model, MGNM)，

该模型通过 IFC 文件转换生成，还将该模型与地理信息系统(Geographic Information Systems GIS)相结合。

Aurelien Bres 等[47]研究了 HVAC 水路输配系统的连接关系（直连、二通、三通、阀门等）自动生成及寻优方法，该方法需要设计者预先确定输配节点（风机盘管、散热片等）在建筑内的位置，同样也将生成的 HVAC 系统用于建筑能耗模拟，得到较为接近建筑实际运行能耗的结果。Sandurkar Sunand[48]提出了一种基于镶嵌式花样(Tessellated Format)的搜索空间，使得建筑内管路的生成能够更加的高效。

从建筑的宏观角度来看，建筑布局的设计决定了功能区的分布并限制了 HVAC 系统的分布，因此在从事空调系统自动设计研究的过程中，也需要关注建筑布局自动设计的相关研究，布局和空调系统的融合自动设计将会是未来的发展方向。

类似 CBR 的基本思路，Georg Suter[49]通过选择、聚合、分解三个步骤来从已有的建筑房间布局来生成新的建筑房间布局，在该过程中，布局中各个元素之间的几何或其他约束关系由有向权重网络图来显式表达。

Stefan Runde[50]提出了一种自上而下的建筑空调系统自动化设计方法。该自动化方法由概念性设计和细节设计两个步骤组成以完成自动设计。概念性设计是通过基于平台和制造商提供的独立的功能模块生成程序进行设计。细节设计通过进化技术取代了平台和厂商指定的功能模块。最后作者提出可将这一套方法与房间布局以及空调系统的自动设计集成在一起开发。

1.1.4 障碍规避算法

为了规避障碍物，最常用的算法有 A*算法，许多学者将其应用于多个不同领域。唐崇[51]将 A*启发式搜索算法应用于游戏网格地图的路径搜索，提出了一种加速 A*算法，使用深度优先搜索，降低了 A*算法查找节点的数量，提高了算法的速度。刘梓良[52]提出了一种基于 A*算法的分时路径搜索，为了解决游戏中出现的动态障碍物，使用了路径拼接的方法。陈素琼[53]使用余弦函数对启发式搜索函数进行修改，也减少了算法中查找节点的数量。张永旭[54]提出了一种能够判断多个起止节点的时候的最短路径，同时结合了近路回溯的方法，实现了较高的搜索效率和通用性。邹亮将 A*算法与 B 样条函数结合对月球车的路径规划进行了研究，不仅能够较快地搜索出有效路径，而且能够满足月球车动力车的约束。唐晓东[55]使用改进 A*路径算法实现了无人机的三维路径搜索，马飞[56]使用改进

A*算法实现了地下无人铲机的路径规划，赵晓[57]和刘晨曦[57, 58]使用改进 A*算法对 A*算法对机器人路径进行了规划，实现了环境识别和障碍物的规避。

除 A*算法，另有其他可应用于路径规划的算法。郝新培[59]为了解决电网单线图的自动成图，使用 SP 编码对电站的信息进行描述，同遗传算法对编码进行优化最终使得布线的总面积实现近似的最优。段建民等[60]使用一种经典的强化学习算法——Q-Learning 算法，通过引入环境的势能值，使得移动机器人不断学习实现路径的规划。管亚丽[61]使用 DNA 进化算法实现了多目标物流配送的路径优化过程，克服了传统算法容易陷入局部最优情况的问题。聂廷哲等[62]通过建立 Hopfield 神经网络模型，并通过退能量函数模型的极小值点进行求解得到了燃气输配网络布局优化的解。焦国帅[63]通过一种新的管路路径的编码方式 NSGA-Ⅱ，对三维管路的设计问题进行了多目标优化求解。

对于多个节点的暖通空调输配问题，需要考虑图论中的最小生成树算法，常用的最小生成树算法有普里姆算法(Prim Algorithm)和克鲁斯卡尔算法(Kruskal Algorithm)，已经有学者将其应用于管路的相关问题[64]。仅有最小生成树算法无法解决输配问题中需要解决的其他问题，如减少管路中的弯头数量和平衡管道中的压力。为了解决该问题，往往需要遍历生成图的所有最小生成树[65]。

Gabow Harold[66]提出一种可查找有向图或者无向图中所有生成树的算法，该算法基于广度优先搜索，对于一个有向图，其时间复杂度为$O(V+E+EN)$，其中V为图中顶点的个数，E为图中边的条数，N为生成树的个数。对于一个无向图，其时间复杂度可用$O(V+E+VN)$表示，一般来说，$V<E$，因此该算法对于有向图来说时间复杂度更低。Matsui Tomomi 也提出了一种时间复杂度为$O(V+E+VN)$的生成树枚举算法，该算法较一般的枚举算法有更强的灵活性，当给与一定的限制条件的情况下，该算法可从生成树空间解中更有效地搜索符合条件的生成树。

Kapoor Sanjiv[67]和 Shioura Akiyoshi[68]提出了一种时间复杂度为$O(V+E+N)$的生成树枚举算法，该算法较前两者具有更低的时间复杂度，但是只适用于无向图，Kapoor Sanjiv 的算法的实现思想是通过找到无向图中存在的所有环，然后对环中的边进行遍历得到所有的环进行遍历得到所有生成树，Shioura Akiyoshi 的算法，该算法的结构更为简单，通过一种枚举方法的逆向实现来遍历所有的生成树。

以上的算法仅实现了所有生成树的枚举，没有考虑不同生成树之间代价的不同，Sörensen Kenneth[69]提出的算法能按照生成树的代价的大小顺序进行输出，其时间复杂度为$O(N\cdot Elog(E)+N^2)$。Wright Perrin[70]，Eppstein David[71]和

Yamada Takeo[72]都提出了一个能够枚举所有最小生成树的算法，Wright Perrin 的算法还可以计算得到所有最小生成树的个数，Eppstein David 的算法有三个算法中最理想的时间复杂度，为$O(E + Vlog(V) + K)$，其中K为最小生成树的个数，但其算法的实现过程较为复杂，而且更适用于边的权重相对随机的情况，Eppstein David 和 Yamada Takeo 的算法实现过程相对简单，但算法的时间复杂度相对较高。

尽管有上述的枚举所有生成树和最小生成树的各类算法，而且算法的时间复杂度也较为理想，但是这些算法均无法直接解决管路生成问题中的一些问题，例如，要如何生成图以较少后续的搜索范围，如何减少弯头等。因此需要将最小生成树的算法同路径规划的问题相结合，以解决当前的问题。

在系统设计的后期阶段，可能还需要对空调系统进行动态的模拟，Chen Yan 等[73]基于 Simulink 的组件库搭建了 HVAC 系统的动态模型库，这一系统能够根据系统的特性和风机的曲线实现不同控制策略下新风、排风和回风的风量的实时计算。Wei Xiupeng 等[74]提出了一个多目标优化的 HVAC 系统控制方法，该方法基于数据驱动模型，使用粒子群算法实现了房间相对湿度、干球温度、二氧化碳和能耗的多目标优化。Zeng Yaohui 等[8]使用数据驱动模型对一个多区 HVAC 系统的控制做出了预测和控制，作者使用萤火虫算法对参数进行优化，并对不同参数对节能量的敏感性进行了分析。Kusiak Andrew 等[75]使用非线性自回归模型(Nonlinear Autoregressive Network, NARX)对 HVAC 系统进行模拟和优化，目标函数为系统的能耗和房间的温度，优化方法为粒子群算法，实现了 HVAC 系统的节能控制。Magnier Laurent 等[76]使用 TRNSYS 和神经网络对建筑的设计参数如围护结构、空调系统设定参数进行优化设计。Moosavian Naser[77]提出了一种水系统网络水力计算的方法，可实现较高的计算效率和较强的鲁棒性。Guirardello Reginaldo[78]将机房设备的布置同机房内管路的布置结合。

1.1.5 国内外研究总结

根据以上的文献综述，我们可以看到需要实现空调系统的自动设计流程，目前需要进一步研究主要是负荷的自动化计算、设备和系统的选择、房间设备的布置以及管路的敷设。尽管前人已经在这些分别的领域做了一定的研究，但为了实现空调系统整个流程的设计自动化，还需要将个别方法做的更好，以及将不同步骤之间的有机结合。

1.2 自动设计流程和本文研究内容

1.2.1 自动设计流程

本论文的研究工作是基于我们[79]提出的基于 BIM 的暖通空调自动设计流程，如图 1.1 所示，该流程将暖通空调系统的工作分为以下四个内容：

1. BIM 模型的检验

这一部分工作主要是对现有的 BIM 建模软件导出的 gbXML 格式进行进一步的操作，主要包括对复杂几何的转化和空间完整性的验证。

2. 自动分区及负荷计算

这一部分工作基于第一部分完成检验的 gbXML 文件进一步操作得到对建筑暖通空调系统设计所需要的空间负荷信息，主要包括进一步的模型简化、参数设置、分区参数设置、自动分区算法、idf 模型转化和负荷自动计算。

3. 系统定义及选型

系统定义主要通过对建筑的类型、平面、分区结果的综合判断得到适合该建筑的空调系统类型和各环路（loop）的定义，选型则是基于空调系统类型和第二部分工作得到的建筑负荷对建筑各环路的设备进行选型。

4. 设备和布管

这一部分需要完成对各个房间内的设备布置和各个设备之间的连接关系进行确定，以及最后对系统合理性的验证（如水力校核等）。

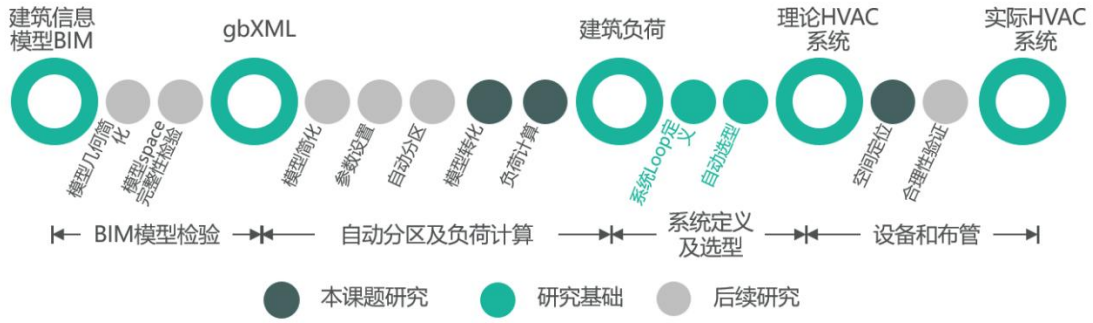


图 1.1 暖通空调系统流程

1.2.2 本文研究工作

杨志伟在文献中[80]已经初步完成了基于负荷的系统定义和选型，本文要做的工作是将该工作继续拓展，进一步完成自动分区部分的基于 gbXML 文件的负荷自动计算和设备和布管中的初步的空间定位（二维管路的敷设算法）。

针对此，我们提出了如图 1.2 所示的研究路线，主要包括

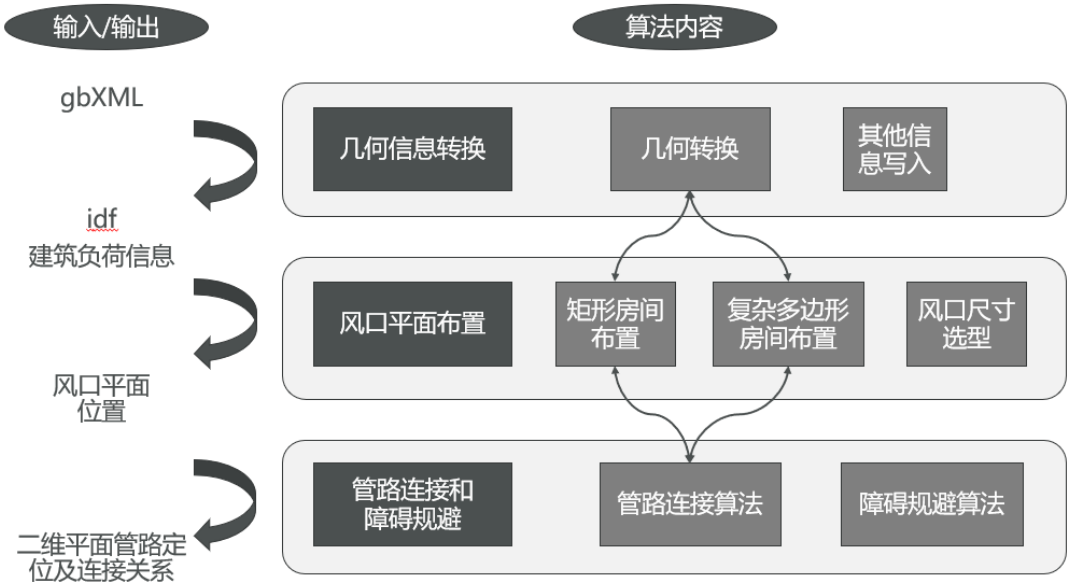

输入/输出
算法内容
gbXML
几何信息转换
几何转换
其他信息写入
idf
建筑负荷信息
风口平面布置
矩形房间布置
复杂多边形房间布置
风口尺寸选型
风口平面位置
管路连接和障碍规避
管路连接算法
障碍规避算法
二维平面管路定位及连接关系

图 1.2 本文研究路线

第 2 章 几何转化和负荷自动化计算

本章主要介绍如何从一个 BIM 模型到建筑负荷的自动计算的过程，主要包括对 BIM 与 gbXML 格式的介绍以及讲 gbXML 文件的几何、建筑结构等信息转写到 idf 文件的过程。

2.1 模型几何转化

2.1.1 BIM 与 gbXML

我们通常说的 BIM 是指建筑信息化（Building Information Modeling）的技术，该技术可以在不同的软件和不同的终端实现不同的交互功能，对于我们空调系统的设计流程而言，需要一种可交互性和通用性较强的交互格式。目前主流的 BIM 开放标准主要有 IFC 和 gbXML 等。但是 IFC 格式目前普遍存在与建筑分析软件可交互性较差的问题，因此我们采用 gbXML 格式来作为建筑空调系统设计原始输入格式，该文件可从 Revit 等 BIM 建模软件直接导出得到。

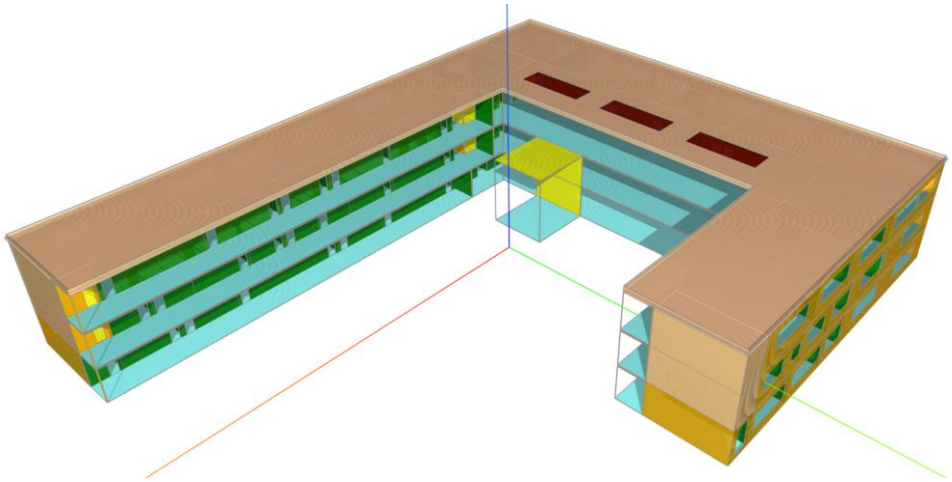


图 2.1gbXML 模型

图 2.2gbXML 文本信息

图 2.1 展示了一个 gbXML 模型的三维图，其文本文档信息如图 2.2 所示，gbXML 是一种基于 XML 格式扩展的文本格式，其基本结构为树状结构，顶层结构如图 2.3 所示。其中 gbXML 是 XML 文件的根元素，对于 XML 中的每个元素，都包含 3 种数据信息，第一个是元素的 attribute，包含了对当前元素每一项属性的描述，一般来说对于一个元素可有多个数学的描述，第二个是元素的子元素，与当前父元素组成被包含关系，第三个是元素的文本描述，一般用于简单元素的赋值型描述。对于根元素 gbXML 来说，其包含 9 个属性标签[81]，其中 id、engine、SurfaceReferenceLocation 这 3 个标签是可选标签，其他均为必选属性标签，这些必选属性标签规定了当前文件的全局基本属性，如温度单位、长度单位、面积单位、体积单位等。有的属性标签的变量类别为字符串，例如 id 属性，有的属性标签的变量类别为枚举类变量，例如 temperatureUnit，可从 F，C，K，R 中进行选择，分别表示华氏温标、摄氏温标、开式温标和勒氏温标。gbXML 有 20 类子元素，其中 Campus 为必选元素，其余均为可选元素，包括对建筑照明、结构、材料、时刻表、分区、HVAC 系统的描述等。

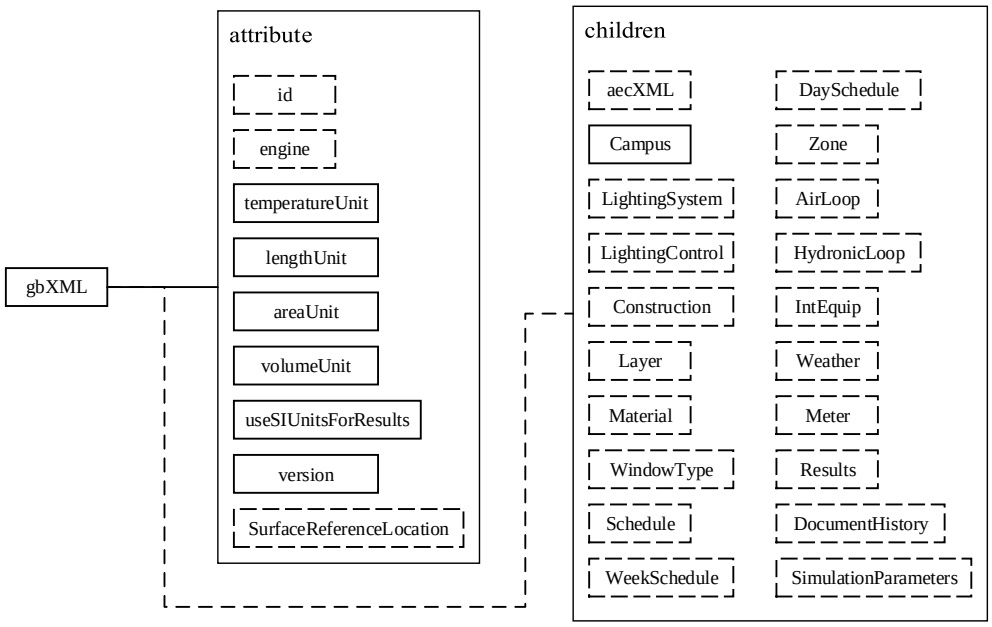


图 2.3gbXML 元素树状结构

2.1.2 gbXML 中的几何描述

在 gbXML 中，对于几何信息的描述主要包括 Surface 和 Space，两个元素分别位于 Campus 的子元素和 Campus 子元素 Building 的子元素。其中对于 Space 元素，一般来说一个 Space 为一个建筑房间或者房间的一部分，其必选属性为 id，此外还可以定义空间对应的类型、楼层信息、IFCGUID（IFC 文件中每一个绘图元素的唯一 id）等。其有两个必选子元素，分别为空间面积和体积的大小，可选子元素包括对空间照明的描述(Lighting, LightingControl)，得热量的描述(InfiltrationFlow, PeopleNumber, PeopleHeatGain, LightPowerPerArea, EquipPowerPerArea, AirChangesPerArea)，几何描述(PlanaryGeometry, ShellGeometry)等。

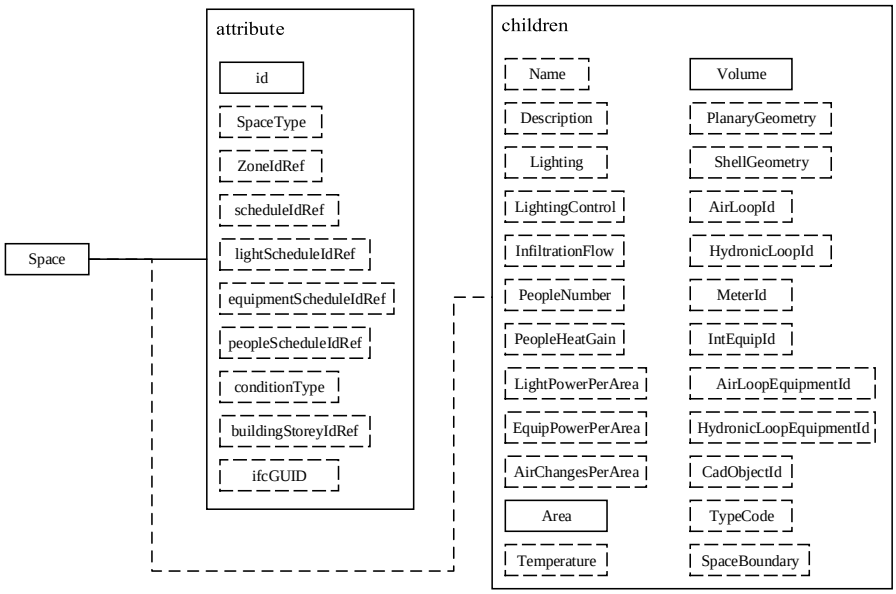


图 2.4Space 元素树状结构

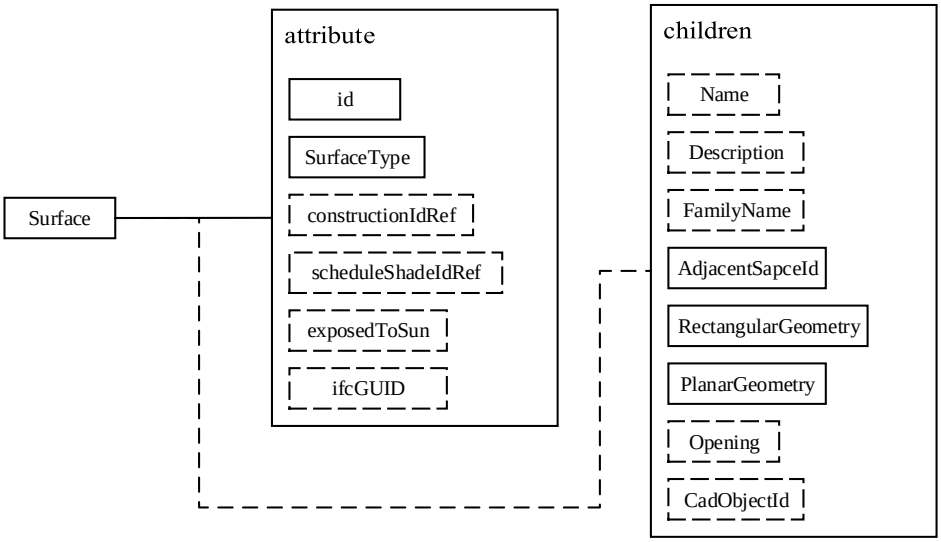


图 2.5Surface 元素树状结构

其中对于 Surface 元素，一般来说为封闭成一个 Space 的其中一个面，为一个建筑房间或者房间的内墙、外墙、楼板等，或者为将一个区域按照一个虚拟的墙(Air)的分隔面，其必选属性为 id 和 SurfaceType，SurfaceType 的可选项如，对于每一个 Surface 应该指定面的类型。此外还可以定义空间对应的类型、楼层信息、IFCGUID（IFC 文件中每一个绘图元素的唯一 id）等。其有两个必选子元素，分别为空间面积和体积的大小。最重要的一个子元素为 AdjacentSpaceId，该元素的 spaceIdRef 可以关联到该 Surface 所属的 1 个或 2 个 Space。其他可选子元素包括对空间照明的描述(Lighting, LightingControl)，得热量的描述(InfiltrationFlow,

PeopleNumber, PeopleHeatGain, LightPowerPerArea, EquipPowerPerArea, AirChangesPerArea)，几何描述(PlanaryGeometry, ShellGeometry)等。

表 2.1Surface 的 SurfaceType 属性可选项

可选项	说明
InteriorWall	内墙，与水平面的夹角大于 60°
ExteriorWall	外墙，与水平面的夹角大于 60°
Roof	屋顶，，与水平面的夹角小于 60°
InteriorFloor	建筑内地板，与水平面的夹角大于 60°
ExposedFloor	建筑外地板，与水平面的夹角大于 60°
Shade	建筑遮阳面，不与任何 Space 相连
UndergroundWall	与土壤相邻的墙，与水平面的夹角大于 60°
UndergroundSlab	与土壤相邻的楼板，与水平面的夹角介于 150°和 180°之间
Ceiling	建筑天花板，与水平面的夹角大于 60°
Air	虚拟（空气）墙，与水平面的夹角介于 0°和 180°之间
UndergroundCeiling	位于地下的天花板，与水平面的夹角大于 60°
RaisedFloor	活动地板，与水平面的夹角介于 150°和 180°之间
SlabOnGrade	建立于土壤上的板式基础，与水平面的夹角介于 150°和 180°之间
FreestandingColumn	独立的柱子，用顶面表示
EmbeddedColumn	同四周墙体相连的柱子

根据以上的信息，我们可以 Space-Surface 的遍历方式遍历到 gbXML 中所有封闭空间中所需要的面。遍历方法如下：

1） 遍历所有 Space 元素，将其 id 和需要的属性或标签信息存入一个用于存储每个 Space 及对应信息的字典当中；

2） 遍历所有 Surface，当其 AdjacentSpaceId 的 spaceIdRef 中存在包含字典中的某个 Space 时，更新该 Space 中的 Surface 信息。

为了将 gbXML 中的信息用于能耗模拟，我们需要提取出几何的坐标信息以供计算引擎进行计算。

2.1.3 idf 文件

为了实现负荷的自动化计算，需要将建筑的信息交给负荷计算引擎，目前主要有 EnergyPlus 和 DOE-2。idf 是 EnergyPlus 读取的文件格式，得益于 EnergyPlus 开源的特性，比较方便对 idf 文件进行读写的操作。一般来说，一个 idf 文件包含了建筑的几何信息、分区及 HVAC 系统的信息等。

```
!- Windows Line endings

VERSION,
    8.6;                    !- Version Identifier

ZONE,
    1F:1 office,            !- Name
    0,                      !- Direction of Relative North
    0,                      !- X Origin
    0,                      !- Y Origin
    0,                      !- Z Origin
    1,                      !- Type
    1,                      !- Multiplier
    autocalculate,          !- Ceiling Height
    511.4422,               !- Volume
    140.1211,               !- Floor Area
    TARP,                   !- Zone Inside Convection Algorithm
    ,                       !- Zone Outside Convection Algorithm
    Yes;                    !- Part of Total Floor Area
```

图 2.6 idf 文本结构

2.1.4 从 gbXML 到 idf 的文件转换

对于一个 Surface，我们需要获取到它的坐标点的信息，这时候就需要进行 gbXML 文件的解析和 idf 文件的写入。

首先是 gbXML 的解析，鉴于这里我们采用 python 自带的 ElementTree 库来对 gbXML 文件进行读取和遍历，为了对 gbXML 文件进行解析，需要设置其对应的名称空间：{http://www.gbxml.org/schema}。有了名称空间之后，我们可以通过对 gbXML 文件进行层级搜索和信息获取。如图 2.7 所示，在获取一个元素之后，可以通过 tag 和 attrib 获取元素的标签信息和属性。

```
import xml.etree.ElementTree as ET #载入ElementTree
tree = ET.parse("test.xml")        #加载gbXML文件
root = tree.getroot()              #获取gbXML根节点
print(root.tag, ":", root.attrib)  # 打印根元素的tag和属性
```

图 2.7 gbXML 读取根节点信息代码

其次是 idf 文件的写入，idf 文件本质上也是一个文本文件，因此可直接通过字符串的编辑功能进行编辑，python 的第三方库 eppy 提供了很好的 idf 文件的读写接口，对于每一个 idf 文件，在代码中需要将其与 idd 文件对应，idd 文件是 EnergyPlus 的输入数据字典，相当于一个模板，每一个 EnergyPlus 版本会对应一个 idd 文件版本，可在 EnergyPlus 的安装目录中获取。在设置完 idd 文件之后，即可通过对 idf 中类标签进行索引的方式对 idf 文件进行修改，如图 2.8 所示，该段代码可通过判断 gbXML 中某一 surface 属性是否为屋顶来给 idf 文件增加一个 Surface Type 为 Roof 的类。

```
if surface.attrib.get('surfaceType') == 'Roof':
    idf_modified.newidfobject('BuildingSurface:Detailed'.upper(),
                              **{'Name' : surface_name,
                                 'Surface_Type' : 'Roof',
                                 'Construction_Name' : 'Project Partition',
                                 'Zone_Name' : zone_name,
                                 'Outside_Boundary_Condition' : 'Outdoors',
                                 'Sun_Exposure' : 'NoSun',
                                 'Wind_Exposure' : 'NoWind',})
```

图 2.8 idf 写入 surface 代码

2.2 其他信息转化

2.2.1 建筑结构描述

在 gbXML 的 Surface 元素中，有一个 constructionIdRef 可以对指定建筑面的结构信息赋值，如图 2.9 所示，gbXML 中的对应的 Construction 元素的子元素中对当前结构的信息进行了描述，包括结构的材料层信息(Layer)，名称，吸收率，粗糙程度，经济性，反射率，吸收率等信息。图 2.10 列出了 gbXML 中 Layer 元素对建筑结构信息的描述，元素的标签 id 表示了 Layer 的名称，子元素 InsideAirFilmResistance 则描述了该结构的总热阻，子元素 MaterialId 及其标签 materialIdRef 定义了当前 Layer。

```
<Construction id="construction-70">
  <LayerId layerIdRef="layer-70" />
  <Name>R13 Wood Frame Wall, Wood Shingle</Name>
  <Description>ASHRAE 90.1 compliant R13 sheathing 16 inch on center wood framed wall</Description>
  <Absorptance unit="Fraction" type="ExtIR">0.7</Absorptance>
  <Roughness value="Smooth" />
  <Cost costType="FirstCost">
    <CostValue currency="USDollars" unit="USDollarsPerSquareMeter">121.847348</CostValue>
  </Cost>
  <Reflectance type="Ground" unit="Percent">0.2</Reflectance>
  <Transmittance type="Solar" unit="Percent">0</Transmittance>
  <Reflectance type="IntSolar" unit="Percent">0.5</Reflectance>
  <Absorptance type="IntTotal" unit="Percent">0.5</Absorptance>
  <Emittance type="IntIR" unit="Percent">0.9</Emittance>
  <Reflectance type="ExtSolar" unit="Percent">0.3</Reflectance>
  <Absorptance type="ExtTotal" unit="Percent">0.7</Absorptance>
  <Emittance type="ExtIR" unit="Percent">0.9</Emittance>
</Construction>
```

图 2.9 gbXML 文件 Construction 元素

```
<Layer id="layer-70">
  <InsideAirFilmResistance unit="SquareMeterKPerW">0.119754995368299</InsideAirFilmResistance>
  <MaterialId materialIdRef="mat-391" />
  <MaterialId materialIdRef="mat-244" />
  <MaterialId materialIdRef="mat-384" />
  <MaterialId materialIdRef="mat-448" />
  <MaterialId materialIdRef="mat-353" />
</Layer>
```

图 2.10 gbXML 文件 Layer 元素

在 idf 的定义中，我们也需要通过 Construction 类对每一层的材料信息进行描述，与 gbXML 文件不同的地方在于，Construction 类中没有对当前结构的热阻

进行描述，图 2.11 列出了 idf 文件中 Construction 类字段中的信息，Construction 主要通过对建筑的每一层材料信息进行描述，其中每一层 Layer 可由 Material 类或 Material:NoMass 类描述，两者的不同在于 Material:NoMass 类的必选字段仅需要对材料的热阻进行描述，而 Material 则需要补充更完整的材料信息。

Field	Units	Obj1	Obj2
Name		Light Exterior Wall	Light Roof/Ceiling
Outside Layer		F08 Metal surface	M11 100mm lightwei
Layer 2		I02 50mm insulation	F05 Ceiling air spaci
Layer 3		F04 Wall air space r	F16 Acoustic tile
Layer 4		G01a 19mm gypsum	
Layer 5			
Layer 6			
Layer 7			
Layer 8			
Layer 9			
Layer 10			

图 2.11 idf 文件 Construction 类字段

图 2.12 列出了 gbXML 文件中 Material 元素中的信息，包括材料名称、热阻、厚度、导热系数、密度、比热等。图 2.13 中列出了 idf 文件中 Material 类字段中的信息，主要包括材料的名称，粗糙程度，厚度，导热系数，密度，比热等必选字段，以及材料吸收率、反射率等可选字段，与 gbXML 中 Material 类字段中的信息一致，只需要进行简单的字段对应和单位换算即可将 gbXML 中的信息转到 idf 中。

```
<Material id="mat-391">
  <Name>Wood Shingle  (WS01)</Name>
  <Description>For Walls</Description>
  <R-value unit="SquareMeterKPerW">0.153215949956501</R-value>
  <Thickness unit="Feet">0.0583</Thickness>
  <Conductivity unit="WPerMeterK">0.1154400034228</Conductivity>
  <Density unit="KgPerCubicM">512.5908326974</Density>
  <SpecificHeat unit="JPerKgK">1256.04</SpecificHeat>
</Material>
```

图 2.12 gbXML 文件 Material 元素

Field	Units	Obj1	Obj2	Obj3
Name		Built-Up Roof	Roof Insulation	Metal Surface
Roughness		Rough	MediumRough	Smooth
Thickness	m	0.2	0.04	0.0008
Conductivity	W/m-K	1.74	0.042	47.6
Density	kg/m3	2500	30	7840
Specific Heat	J/kg-K	837	1380	480
Thermal Absorptance		0.9	0.9	0.9
Solar Absorptance		0.7	0.7	0.7
Visible Absorptance		0.7	0.7	0.7

图 2.13 idf 文件 Material 类字段

图 2.14 列出了 idf 文件中 Material 类字段中的信息主要包括材料的名称，粗糙程度，热阻等必选字段，以及材料吸收率、反射率等可选字段。

Field	Units	Obj1	Obj2	Obj3
Name		Steel Joist Floor with Batt Insulation	Carpet and Pad	SOG Insulation
Roughness		Rough	VeryRough	Rough
Thermal Resistance	m2-K/W	0.001	0.22	1.3
Thermal Absorptance		0.9	0.9	0.9
Solar Absorptance		0.7	0.7	0.7
Visible Absorptance		0.7	0.7	0.7

图 2.14 idf 文件 Material:NoMass 类字段

2.2.2 时刻表信息

在 gbXML 中，可以对每个 Space 都可以进行以下几类时刻表的标签属性描述：

表 2.2 Space 的 Schedule 描述标签

标签	说明
peopleScheduleIdRef	指定当前 Space 人员时刻表的 id，字符串类型
lightScheduleIdRef	指定当前 Space 照明时刻表的 id，字符串类型
equipmentScheduleIdRef	指定当前 Space 设备时刻表的 id，字符串类型
conditionType	指定当前 Space 制冷或制热的需求，可选 Heated，Cooled，HeatedAndCooled，Unconditioned，Vented 等

Field	Units	Obj1	Obj2
Name		2F:62office People	2F:54office People
Zone or ZoneList Name		2F:62office	2F:54office
Number of People Schedule Name		Building_OCC_Sch	Building_OCC_Sch
Number of People Calculation Method		Area/Person	Area/Person
Number of People			
People per Zone Floor Area	person/m2		
Zone Floor Area per Person	m2/person	10	10
Fraction Radiant		0.3	0.3
Sensible Heat Fraction		autocalculate	autocalculate
Activity Level Schedule Name		Activity Schedule	Activity Schedule
Carbon Dioxide Generation Rate	m3/s-W	0.0000000382	0.0000000382
Enable ASHRAE 55 Comfort Warnings		No	No
Mean Radiant Temperature Calculation Type		ZoneAveraged	ZoneAveraged
Surface Name/Angle Factor List Name			
Work Efficiency Schedule Name			
Clothing Insulation Calculation Method			
Clothing Insulation Calculation Method Schedule Name			
Clothing Insulation Schedule Name			
Air Velocity Schedule Name			
Thermal Comfort Model 1 Type			
Thermal Comfort Model 2 Type			
Thermal Comfort Model 3 Type			
Thermal Comfort Model 4 Type			
Thermal Comfort Model 5 Type			

图 2.15 idf 文件 People 类字段

```
<Zone id="aim81737" fanSchedIdRef="FanSch-83" heatSchedIdRef="Heatsched-9" coolSchedIdRef="Coolsched-9">
  <DesignHeatT unit="F">70.0</DesignHeatT>
  <DesignCoolT unit="F">73.9</DesignCoolT>
  <Name>Zone Default</Name>
  <CADObjectId>134083</CADObjectId>
  <Description>Default Zone for OFFICE (ASHRAE 2010)</Description>
  <OAFlowPerArea unit="CubicMPerSecPerSquareM">0.000363154167</OAFlowPerArea>
  <AirChangesPerHour>0.000000</AirChangesPerHour>
</Zone>
```

图 2.16 idf 文件 Material 类字段

表 2.3 Schedule:Compact 的字段内容描述

字段内容	说明
Through	指定当前 Space 人员时刻表的 id，字符串类型
For	指定当前 Space 照明时刻表的 id，字符串类型
Until	指定当前 Space 设备时刻表的 id，字符串类型
conditionType	指定当前 Space 制冷或制热的需求，可选 Heated，Cooled，HeatedAndCooled，Unconditioned，Vented 等

Field	Units	Obj1	Obj2	Obj3
Name		HeatingConditionedTime	CoolingConditionedTime	ConditionedTime
Schedule Type Limits Name		On/Off	On/Off	On/Off
Field 1	varies	Through: 04/01	Through: 04/01	Through: 12/31
Field 2	varies	For: AllDays	For: AllDays	For: Weekend
Field 3	varies	Until: 08:00	Until: 24:00	Until: 08:00
Field 4	varies	0	0	0
Field 5	varies	Until: 23:00	Through: 09/30	Until: 23:00
Field 6	varies	1	For: AllDays	0.2
Field 7	varies	Until: 24:00	Until: 08:00	Until: 24:00
Field 8	varies	0	0	0
Field 9	varies	Through: 09/30	Until: 23:00	For: Holidays
Field 10	varies	For: AllDays	1	Until: 24:00
Field 11	varies	Until: 24:00	Until: 24:00	0
Field 12	varies	0	0	For: AllOtherDays
Field 13	varies	Through: 12/31	Through: 12/31	Until: 08:00
Field 14	varies	For: AllDays	For: AllDays	0
Field 15	varies	Until: 08:00	Until: 24:00	Until: 23:00
Field 16	varies	0	0	1
Field 17	varies	Until: 23:00		Until: 24:00
Field 18	varies	1		0
Field 19	varies	Until: 24:00		
Field 20	varies	0		

图 2.17 idf 文件 Schedule:Compact 类字段

2.2.3 其他信息添加

在 EnergyPlus 中，可以通过对每个 Zone 的设置对应的 ZoneHVAC:IdealLoadsAirSystem 实现建筑负荷的模拟，而不需要对每一个空气环路和水环路进行描述，如图 2.18 所示。该模块将会通过计算将房间的回风与室外空气混合后的状态点处理到维持室内温度和适度所需的状态点所需要的能量（能量转化效率为 100%）。其中 Zone Supply Air Node Name 字段表示理想空调系统每个 Zone 对应的输配节点，只需要和 ZoneHVAC:EquipmentConnections 中的名称设置成相同即可，如图 2.19 所示。

Field	Units	Obj1	Obj2
Name		2F:62office Ideal Loads Air System	2F:54office Ideal Loads Air System
Availability Schedule Name		ConditionedTime	ConditionedTime
Zone Supply Air Node Name		2F:62office Ideal Loads Supply Inlet	2F:54office Ideal Loads Supply Inlet
Zone Exhaust Air Node Name			
Maximum Heating Supply Air Temperature	C	80	80
Minimum Cooling Supply Air Temperature	C	0	0
Maximum Heating Supply Air Humidity Ratio	kgWater/kgDryAi	0.0156	0.0156
Minimum Cooling Supply Air Humidity Ratio	kgWater/kgDryAi	0.0077	0.0077
Heating Limit		NoLimit	NoLimit
Maximum Heating Air Flow Rate	m3/s	autosize	autosize
Maximum Sensible Heating Capacity	W	autosize	autosize
Cooling Limit		NoLimit	NoLimit
Maximum Cooling Air Flow Rate	m3/s	autosize	autosize
Maximum Total Cooling Capacity	W	autosize	autosize
Heating Availability Schedule Name		HeatingConditionedTime	HeatingConditionedTime
Cooling Availability Schedule Name		CoolingConditionedTime	CoolingConditionedTime
Dehumidification Control Type		Humidistat	Humidistat
Cooling Sensible Heat Ratio	dimensionless	0.7	0.7
Humidification Control Type		Humidistat	Humidistat
Design Specification Outdoor Air Object Name		SZ DSOA 2F:62office	SZ DSOA 2F:54office
Outdoor Air Inlet Node Name		2F:62office Ideal Loads Outdoor Air Inl	2F:54office Ideal Loads Outdoor Air Inlet
Demand Controlled Ventilation Type		None	None
Outdoor Air Economizer Type		NoEconomizer	NoEconomizer
Heat Recovery Type		None	None
Sensible Heat Recovery Effectiveness	dimensionless	0.7	0.7
Latent Heat Recovery Effectiveness	dimensionless	0.65	0.65
Design Specification ZoneHVAC Sizing Object Name			

图 2.18 idf 文件 ZoneHVAC:IdealLoadsAirSystem 类字段

Field	Units	Obj1	Obj2
Zone Name		2F:62office	2F:54office
Zone Conditioning Equipment List Name		2F:62office Equipment	2F:54office Equipment
Zone Air Inlet Node or NodeList Name		2F:62office Ideal Loads Supply Inlet	2F:54office Ideal Loads Supply Inlet
Zone Air Exhaust Node or NodeList Name			
Zone Air Node Name		2F:62office Zone Air Node	2F:54office Zone Air Node
Zone Return Air Node Name		2F:62office Return Outlet	2F:54office Return Outlet
Zone Return Air Flow Rate Fraction Schedule Name			
Zone Return Air Flow Rate Basis Node or NodeList Nar			

图 2.19 idf 文件 ZoneHVAC EquipmentConnections 类字段

最后为了得到负荷的计算结果，还需要往 idf 文件的 Output:Variable 字段中添加一下字段，输出每个房间的冷热、潜热、新风等逐时负荷等信息：

Zone Ideal Loads Supply Air Total Heating Energy

Zone Ideal Loads Supply Air Total Cooling Energy

Zone Ideal Loads Zone Latent Heating Rate

Zone Ideal Loads Zone Latent Cooling Rate

Zone Ideal Loads Outdoor Air Total Heating Rate

Zone Ideal Loads Outdoor Air Total Cooling Rate

Zone Ideal Loads Outdoor Air Standard Density Volume Flow Rate

2.3 本章小结

本章首先通过 idf 与 gbXML 的几何对应关系介绍了将 gbXML 的 BIM 模型转换至面向 EnergyPlus 进行负荷计算的 idf 文件；然后将 gbXML 中的其他计算建筑负荷的必要信息如建筑结构信息、时刻表等，最后通过将

ZoneHVAC:IdealLoadsAirSystem 添加至 idf 文件实现最后的负荷计算。本章的内容均通过 python 代码实现从 gbXML 到 idf 文件的转换。

第 3 章 房间风口布置

本章确定各房间内的设备（这里只讨论散流器风口）的布置，包括其个数和位置，需要根据不同情况以及房间内障碍物的情况进行确定。

3.1 矩形区域散流器风口布置算法

首先讨论将 m×n 个散流器风口（方形散流器或者圆形散流器）均匀地布置到 w×h 的矩形区域中的情况，已知条件为矩形区域的长 w 和宽 h，所求的解为长和宽方向分别应该放置的风口数目 m 和 n，如图 3.1 所示，图中的圆圈标注了风口在矩形中的分布：

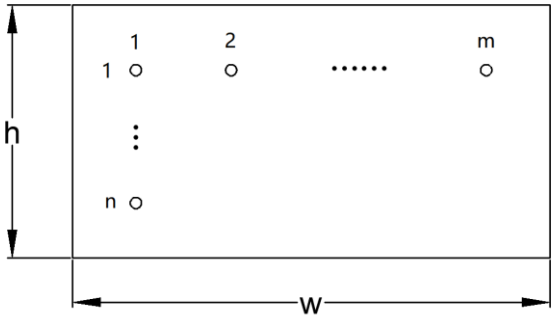


图 3.1 风口布置问题模型

在布置风口时，需要满足气流组织的条件以及均匀分布的约束条件，即将 w×h 的矩形均匀地切割，在算法中设置风口在同一方向中均匀分布，有如下约束条件：

$$dist(p_{i,j}, p_{i,j+1}) = dist(p_{i,j+1}, p_{i,j+2}) \tag{3.1}$$

$$dist(p_{i,j}, p_{i+1,j}) = dist(p_{i+1,j}, p_{i+2,j}) \tag{3.2}$$

为了保证风口的布置能够适应不同长宽条件下，需要灵活设置风口之间的间距以提高算法的鲁棒性，各个风口之间的距离不应太小导致风口数目设置过多，考虑到气流组织要求，风口设置的间距应该过大，因此有如下的约束条件，$dist_{point,max}$ 和 $dist_{point,min}$ 分别表示风口在矩形长和宽方向上所允许的最大间距和最小间距。

$$dist_{point,min} \le dist(p_{i,j}, p_{i+1,j}) \le dist_{point,max} \tag{3.3}$$

$$dist_{point,min} \le dist(p_{i,j}, p_{i,j+1}) \le dist_{point,max} \tag{3.4}$$

本章节选取的是方形散流器或矩形散流器，为了保证气流在不同方向上的均匀性，需要对每个风口服务的矩形区域长宽比进行限制，该矩形区域的长宽比可由风口在矩形长和宽方向上的间距的比值表示：

$$ratio_{rec} = \max(dist(p_{i,j}, p_{i,j+1}), dist(p_{i,j}, p_{i+1,j})) \tag{3.5}$$

该矩形区域的长宽比应该保持小于最大长宽比 $ratio_{max}$ ：

$$ratio_{rec} \le ratio_{max} \tag{3.6}$$

《实用供热空调设计手册》[82]中指出，对于圆形或方形散流器，其服务的区域最大长宽比宜为 1.25，在本算法中我们通过以下情况求出算法鲁棒性最大时推荐的最大长宽比：

假设有一个房间的长和宽分别为 w 和 h 且 w>h，则图 3.2 中的原始矩形长宽比为：

$$ratio_{rec} = \frac{h}{w} \tag{3.7}$$

图 3.2 原始矩形区域

若该矩形无法满足最大长宽比的约束条件，即：

$$\frac{h}{w} > ratio_{max} \tag{3.8}$$

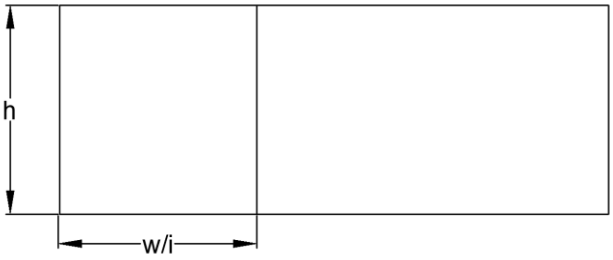


图 3.3 分割后矩形区域

如图 3.3 所示，将该矩形按照长边分割成 i 份，则新的服务的区域的矩形的长宽比为：

$$ratio_{rec}'=\frac{h}{w/i},\ i\geq 2 \tag{3.9}$$

为了保证该分割情况下的矩形满足原始的最大长宽比要求，则有：

$$\frac{h}{w/i}\leq ratio_{max} \tag{3.10}$$

求解以上三个式子，有如下解：

$$(ratio_{max})^2=i,\ i\geq 2 \tag{3.11}$$

当 i=2 时候，满足条件的最小 $ratio_{max}$ 为：

$$ratio_{max}=\sqrt{2} \tag{3.12}$$

因此本算法默认将最大长宽比设置为 $\sqrt{2}$ ，以保证算法最大的鲁棒性，用户也可手动设置为更低的值，但可能返回空结果。

根据以上限制条件，对于一个给定的矩形 w×h 可能存在多解的情况，在算法中我们给出了两种选择最佳布置结果的方法，方法 1 为选取布置点最少的解，如图 3.4 所示，方法 2 为选取服务的区域内长宽比最小的解，如图 3.5 所示。方法 1 得到的结果风口的布置个数为 4×3=12，服务的区域长宽比为 4.5/4=1.125；方法 2 得到的结果风口的布置个数为 5×3=15，服务的区域长宽比为 4/3.6=1.111，

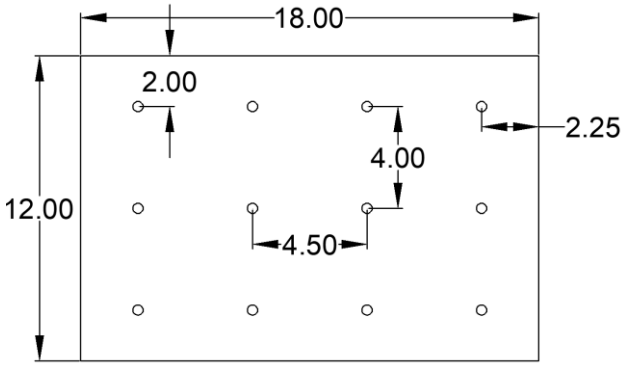


图 3.4 方法 1 风口布置结果

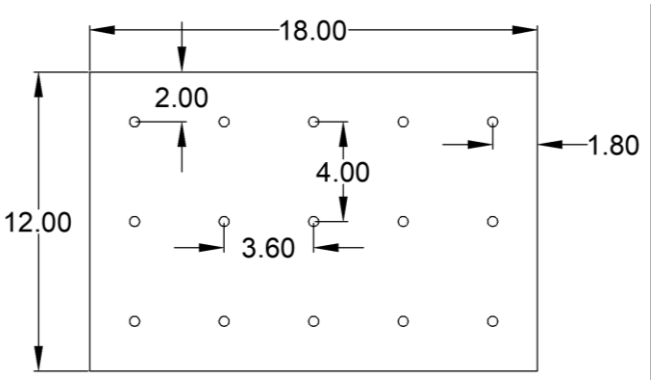


图 3.5 方法 2 风口布置结果

从以上案例我们发现第二种选取服务的区域内长宽比最小的解可能会导致过于追求最小长宽比导致不必要的风口数目增加，因为第一种方法得到的结果长宽比 1.125 与第二种方法得到的长宽比 1.11 并没有显著增加。针对这种情况，我们提出以下结合以上两种方法的优化方法 3：当前矩形存在比当前设置的 $ratio_{max}=\sqrt{2}$ 更优的最大长宽比 $ratio_{max}'$（默认为 1.25）时，选取满足该最大长宽比 $ratio_{max}'$ 的结果中风口数目最少的解，以下为示例：

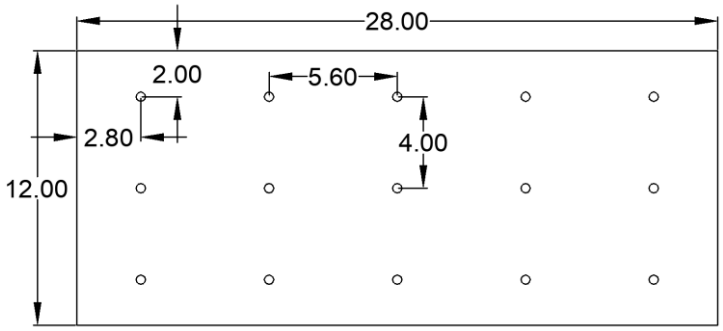


图 3.6 方法 1 风口布置结果

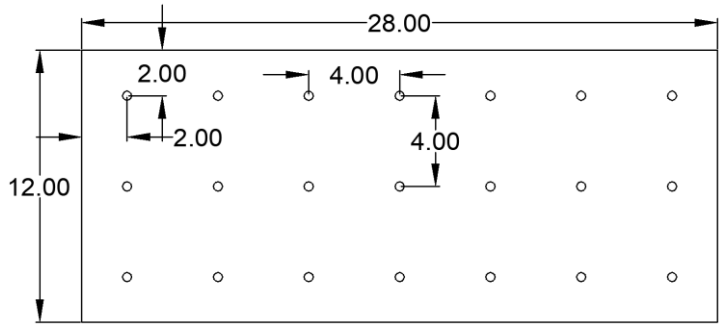


图 3.7 方法 2 风口布置结果

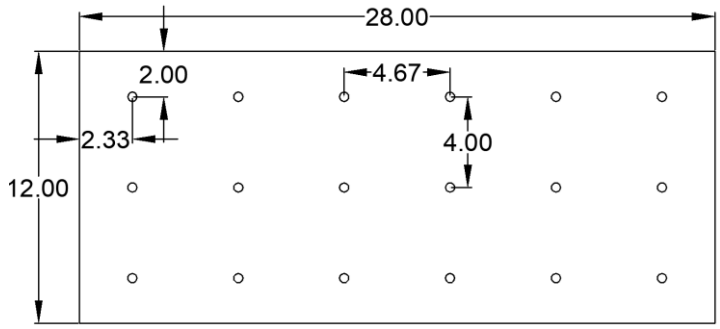


图 3.8 方法 3 风口布置结果

图 3.6 为方法 1 得到的结果，风口数目为 3×5=15，服务区域长宽比为 5.6/4=1，4；图 3.7 为方法 2 得到的结果，风口数目为 3×7=21，服务区域长宽比为 4/4=1；图 3.8 为方法 3 得到的结果，风口数目为 3×6=18，服务区域长宽比为 4.67/4=1.17，可见方法 3 既满足了减少风口数目的要求，也满足了优化长宽比的要求。

将上述约束条件中的 $dist_{point,min}$ 和 $dist_{point,max}$ 分别设置为 4m 和 6m，选用上述方法 3 选取最优分配结果，对不同长和宽尺寸的矩形得到如图 3.9 所示的结果。

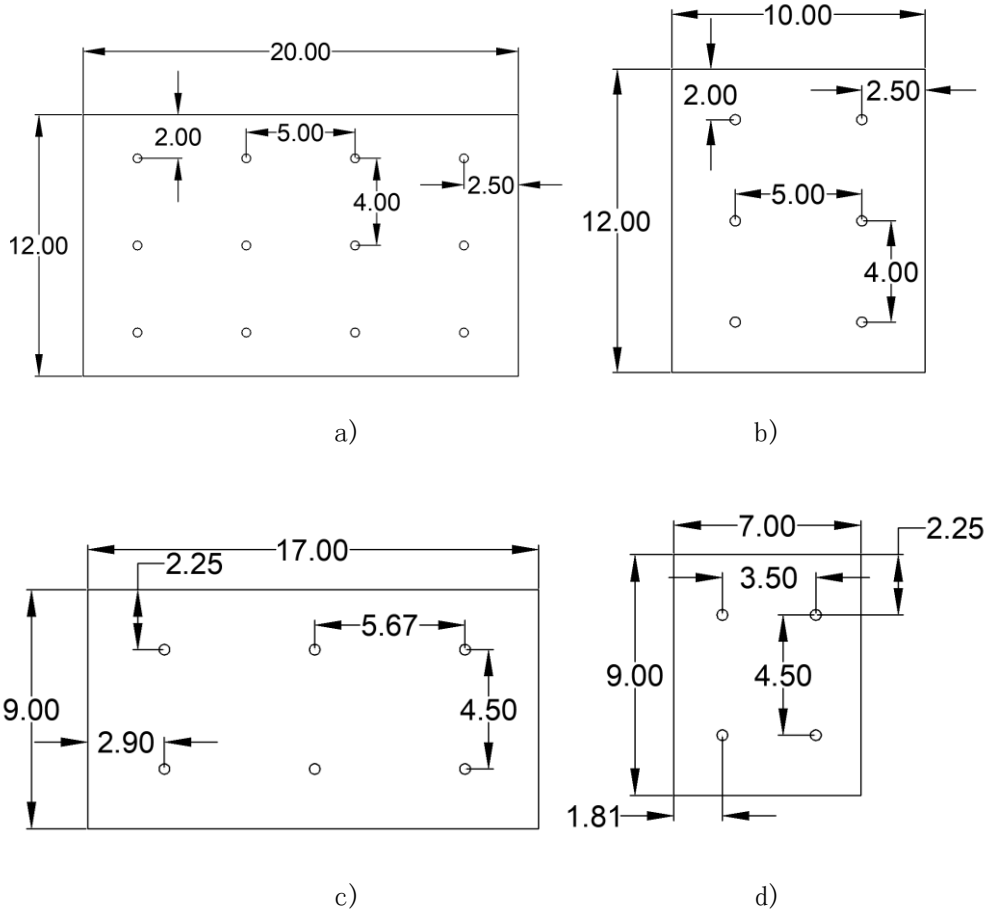


图 3.9 不同长宽矩形区域风口位置生成结果

3.2 多边形区域布置

在不规则区域中布置算法思路是先求出该区域的最小包络矩形，最小包络矩形的求解算法分两种情况求解，分别为凸多边形和凹多边形，此外还考虑了多边形内存在障碍物（如柱子）的情况。

3.2.1 凸多边形

根据最小包络矩形至少经过多边形区域的一条边[83,84]的定理，依次选取多边形区域中的一条边作为包络矩形的边做包络矩形，遍历完所有边之后选取面积最小的包络矩形即为所求多边形的最小包络矩形。

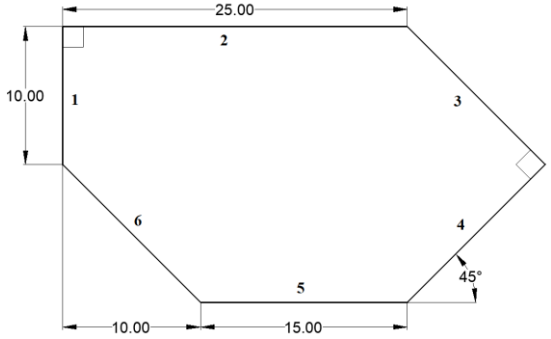


图 3.10 凸多边形

以图 3.10 所示的图形为例，该凸多边形有 6 条边，分别记为 1~6，依次以边 1~6 的顺序，得到图 3.11 和图 3.12 的结果，其中边 1，2，5 有共同的包络矩形，该矩形的长和宽分别为 35 和 20，面积为 700；边 3，4，6 共同的包络矩形，该矩形的长和宽分别为$17.5\sqrt{2}$和$15\sqrt{2}$，面积为 787.5。因此该多边形的最小包络矩形为长为 35 宽为 20 的矩形。

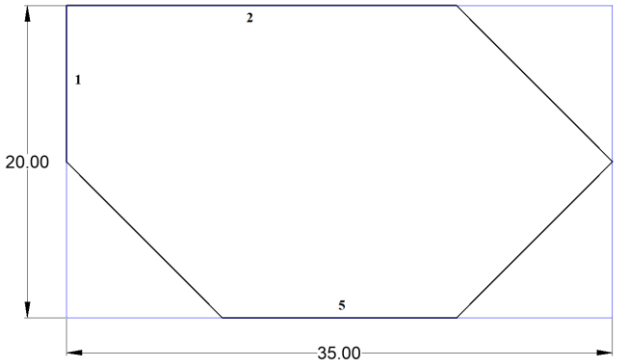


图 3.11 边 1，2，5 的包络矩形

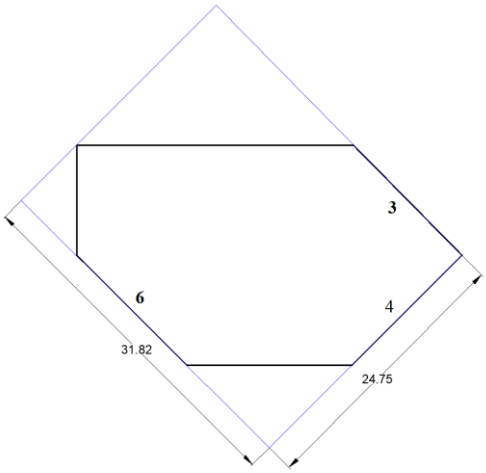


图 3.12 边 3，4，6 的包络矩形

当多边形的包络矩形不为水平方形时，需要通过坐标的旋转和平移变换得到所需要的输入坐标[85]。

1) 点 (x,y) 平移坐标 $(\triangle x,\triangle y)$，得到新坐标 (x',y')：

$$x' = x + \triangle x \tag{3.13}$$

$$y' = y + \triangle y \tag{3.14}$$

用齐次坐标的矩阵形式表示为：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \triangle x \\ 0 & 1 & \triangle y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.15}$$

2) 点 (x,y) 沿坐标原点旋转 θ 得到新坐标 (x',y')

$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta) \tag{3.16}$$

$$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta) \tag{3.17}$$

用齐次坐标的矩阵形式表示为：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.18}$$

3) 点 (x,y) 沿着点 (x_1,y_1) 旋转 θ，则可认为点 (x,y) 先同点 (x_1,y_1) 平移 $(-x_1,-y_1)$，得到 (x',y')：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.19}$$

此时(x_1, y_1)成为了新的坐标原点，使(x', y')沿着新的坐标原点旋转θ，得到(x'', y'')：

$$\begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{3.20}$$

$$= \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

最后再将(x'', y'')平移(x_1, y_1)，得到(x''', y''')：

$$\begin{bmatrix} x''' \\ y''' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} \tag{3.21}$$

$$= \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

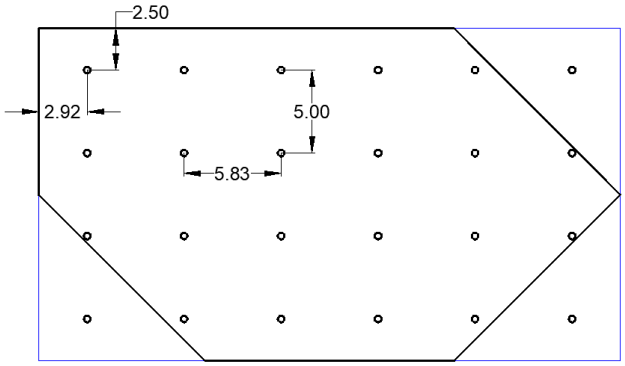


图 3.13 在多边形的最小包络矩形中放置风口的结果

在求出多边形的最小包络矩形后，我们通过上一节描述的算法将风口均匀布置到该矩形中，如图 3.13 所示，可以看到在多边形和矩形的相交区域中风口能够较好的实现均匀布置，下一步通过判断风口布置的位置是否在多边形的内部区域来筛选所需要的布置位置，判断算法如下：

使用射线法[86]判别点在对内部或外部，从上述最小包络矩形放置风口的结果中分别选取一个位于多边形内部和一个位于多边形外部的点作为示例。以该点为起点作任一方形的射线，如果这条射线与原多边形的所有交点为奇数个，则该点位于多边形内部；如果该射线与多边形的所有交点为偶数个，则该点位于多边形外部。如图 3.14 所示，过点 1 做射线，与原多边形的交点个数为 2，因此判定该点位于多边形外部；过点 2 的射线与原多边形的交点个数为 1，因此判定该点位于多边形内部。

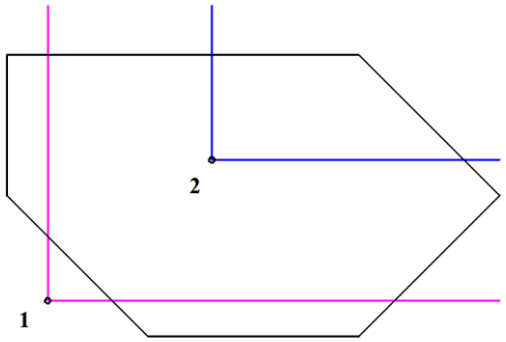


图 3.14 射线法判别点在多边形内部或外部

值得注意的是，若射线经过多边形的顶点时，通过交点个数来判定可能会失效，如图 3.15 所示，过点 1 的射线可能仅与多边形的顶点相交，此时交点个数为 1，若仍使用射线判别法则该点会被错判定为多边形内部的点，因此在算法中应该避免通过多边形顶点的射线来判别。

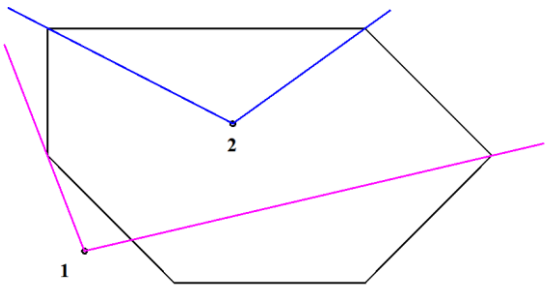


图 3.15 射线与多边形顶点相交的特殊情况

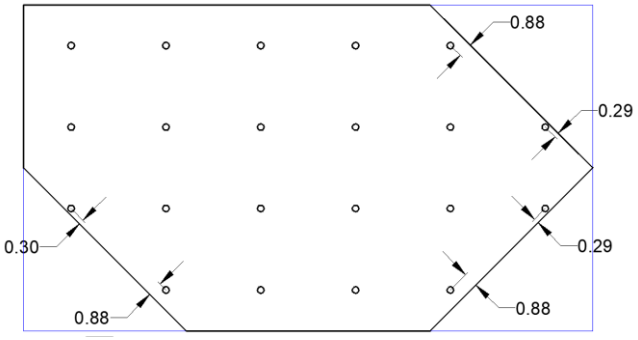


图 3.16 风口位置与多边形位置判定结果

经过射线判别法，得到如图 3.16 所示的结果，可以看到，虽然风口的布置位置都位于多边形区域内，但许多风口距离侧墙的距离都过于近，《实用供热空调设计手册》中建议散流器布置位置的中心距离墙面的距离不宜小于 1m，在本算法中通过判断风口布置位置与多边形轮廓区域的最短距离是否小于设置的最小距离阈值：

$$\min_{k}(dist(l_k, p)) \le dist_{max} \tag{3.22}$$

其中 l_k 为多边形轮廓的线段，将 $dist_{max}$ 设置为 1.5，得到最终风口布置的结果，如图 3.17 所示。

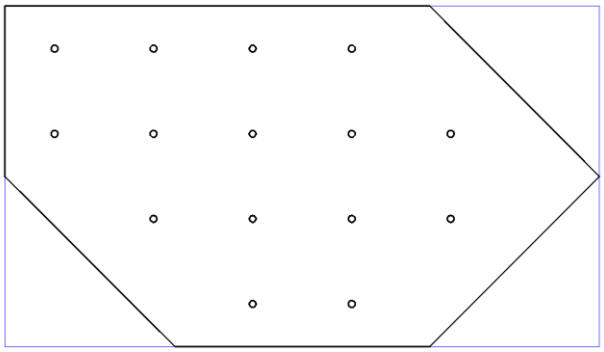


图 3.17 最终风口布置

3.2.2 凹多边形

实际的建筑房间平面往往存在凹多边形的情况，若采用上述凸多边形情况中的算法可能会导致部分区域风口布置不均匀的情况，如图 3.18 所示，在该多边形的右下角区域，风口的布置

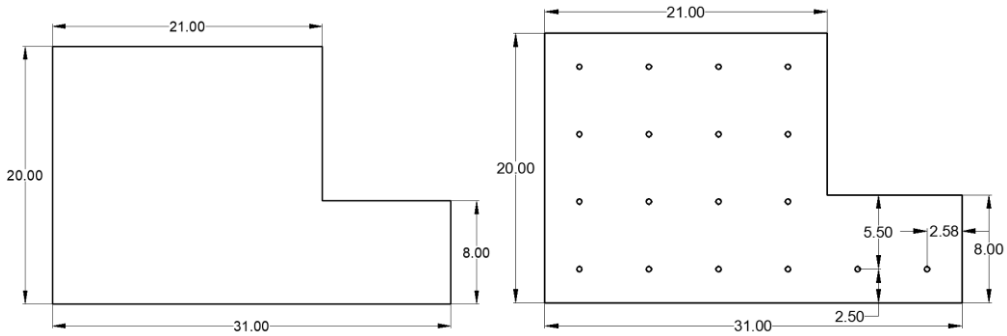


图 3.18 多边形为凹多边形时布置不均匀的情况

为了解决这个问题，我们提出了将凹多边形通过分割转换为凸多边形的算法，其步骤如下：

1) 找到凹多边形中的凹点

2) 根据找到的凹点将凹多边形切割为多个凸多边形

首先找到凹多边形中的凹点，这一步骤可采用几何法或代数法。首先介绍几何法，遍历多边形的顶点，依次过当前顶点和下一个顶点作直线，若其他所有顶点都在该直线的同一侧，则这两个顶点均不是凹点，若遍历至某个点，过该点与前后相邻两个顶点的直线均不能满足上述条件，则该点为凹点。如图所示，过相邻两顶点所作的直线 3 和直线 4 不满足其余顶点不满足，则这两条直线的交点即为所求的凹点。

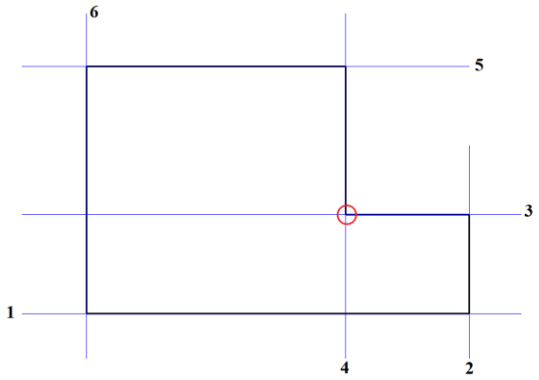


图 3.19 几何法求凹点

几何法适用于凹点数量较少的简单多边形，对于复杂多边形可能存在误判的情况，如图 3.20 所示，直线 3 和直线 4 均不满足判定条件，但两条直线的交点却不是凹点。

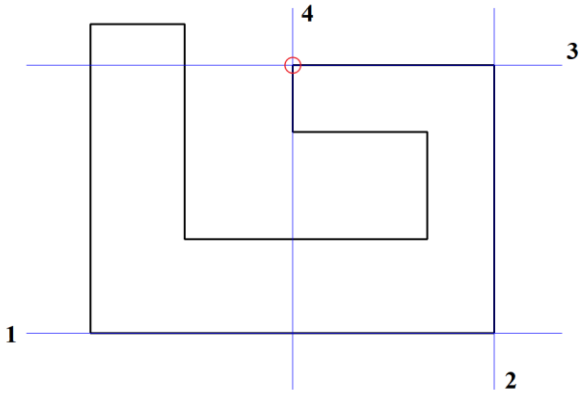


图 3.20 复杂凹多边形用几何法求凹点

第二种方法为代数法，该方法同样需要遍历所有顶点，但不会出现因为多边形过于复杂导致判断失败的情况。其简要流程如下：

对于一个多边形$[(x_1,y_1),(x_2,y_2)...,(x_i,y_i),...(x_n,y_n),(x_1,y_1)]$，各个顶点与其后一个顶点依次连接形成一个封闭几何形状。

对于当前顶点(x_i,y_i),分别计算其与相邻两个顶点(x_{i-1},y_{i-1})和(x_{i+1},y_{i+1})形成的向量：

$$\alpha_i=(x-x_{i-1},y-y_{i-1}) \tag{3.23}$$

$$\alpha_{i+1}=(x_{i+1}-x,y_{i+1}-y) \tag{3.24}$$

并计算两个向量的叉乘：

$$k_i=\alpha_i\times\alpha_{i+1}=(x-x_{i-1},y-y_{i-1})\times(x_{i+1}-x,y_{i+1}-y) \tag{3.25}$$

$$=(x-x_{i-1})(y_{i+1}-y)-(y-y_{i-1})(x_{i+1}-x)$$

遍历所有顶点，得到$[k_1,k_2,...,k_i,...,k_n]$，根据向量叉乘的性质，凹点和非凹点的叉乘结果正负性质不同[Procedural Elements of Computer Graphics]，在建筑平面中的情况，凹点的个数一般小于非凹点的个数，因此我们可以通过k_i的正负性将顶点分为两组，判断数量较少的那组点全为凹点，另一组的点全为凸点。

在完成了多边形凹点的判别之后，我们根据绘制延长线的方法将图 3.19 中的凹多边形转化为多个凸多边形，如图 3.20 所示，可以分别将凹点与相邻的点做直线演唱与多边形相交的方式将该多边形分为两个，在算法中我们默认选取延长长度最短的方式进行切割，即图 3.21-a)所示的结果。

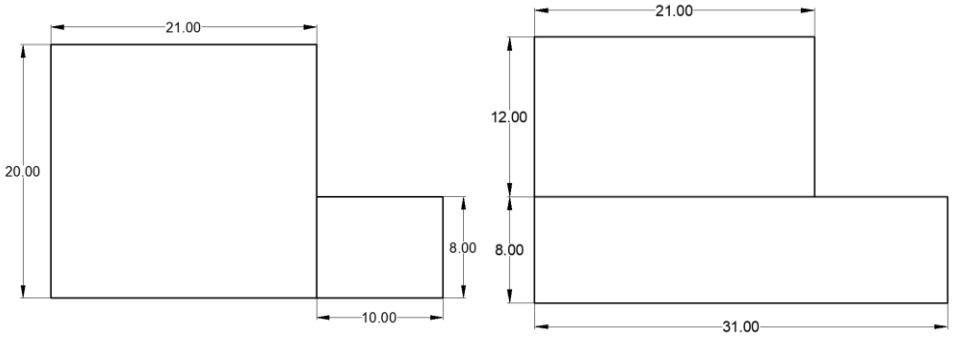


图 3.21 两个相邻点延长线分割结果

完成凹点的识别和分割之后，可将之前用于凸多边形的算法应用至每个分割成的凸多边形，得到如图 3.22 所示的结果，可以看到整个区域内的风口都能够得到比较均匀的分布。

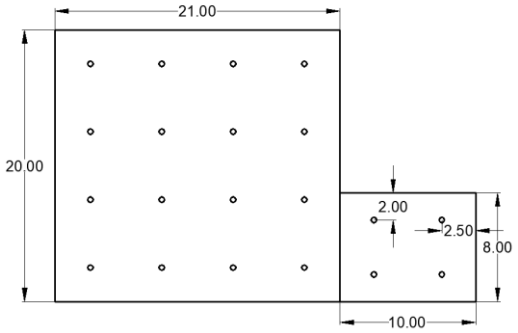


图 3.22 凹多边形分割后的结果

3.2.3 内部障碍物

在散流器的平送方向不应有阻挡物，例如，当空间内存在柱子时，应该调整风口的布置使得风口的平送方向上没有柱子的遮挡。如图 3.23 所示，该区域为凹多边形，经过上述方法完成凹点的识别和区域分割后风口的布置结果符合要求，但是区域内的柱子遮挡了其纵向方向上的两个风口的送风，需要采取一定的措施防止这种情况。

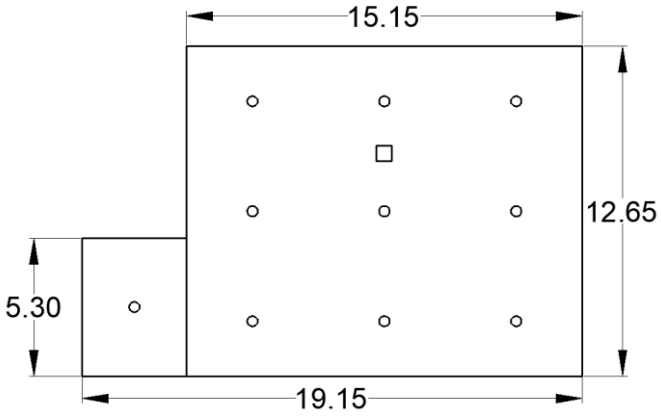


图 3.23 区域内有柱子遮挡的情况

为了解决区域内障碍物的问题，我们提出了障碍物规避的切分算法，算法步骤如下：

1）首先根据已经分割好的凸多边形对该区域进行风口布置，

2）判断区域内的障碍物是否影响到现有的布置，先求出柱子的中心和风口的中心在 x 或 y 方向上的最短距离，若该距离小于设定的距离阈值（默认值为 1m），即需要对该区域在遮挡方向上的切割，对于图 3.23 的情况，需要对该矩形区域进行竖向分割，分割的位置即柱子的中心，对图 3.23 的分割结果如图 3.24 所示。

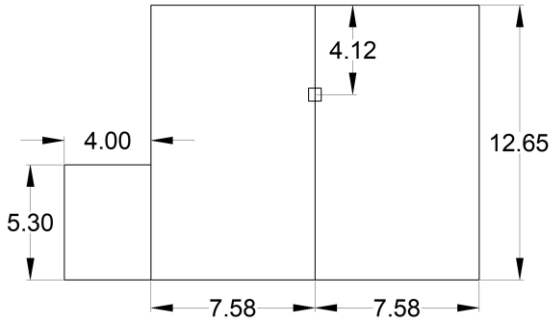


图 3.24 根据障碍物再分割结果

3）对分割后的区域进行凸多边形的布置算法，得到如图 3.25 所示的结果，该结果较好地满足了风口布置的需求。

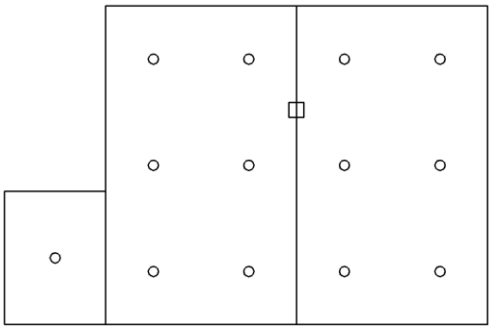


图 3.25 有障碍物时分割后的风口布置结果

3.3 风口选型

在上一节我们完成了风口在房间内位置的分布和放置的数量，现在需要根据室内房间的所需的风量选择合适的风口大小，本节主要介绍方形散流器和矩形散流器的选型。

3.3.1 方形散流器选型

假定在单个区域内选用统一尺寸的散流器，首先根据确定好的房间的风量和散流器的数目确定单个散流器的风量：

$$q_{diffuser}=\frac{q_{zone}}{n_{diffuser}} \tag{3.26}$$

式中，$q_{diffuser}$为房间内单个散流器风量，m3/s。

然后从样本库中检索符合要求的风口，如表 3.1 所示，表中单个方形散流器的风量由下式计算得到：

$$q_{diffuser}=(\frac{w}{1000})^2\times v\times 3600 \tag{3.27}$$

式中，w 为方形散流器的边长，mm；v 为散流器颈部风速，m/s。

表 3.1 标准方形散流器尺寸及对应风量，颈部风速为 v m/s

边长 w（mm）	面积（m2）	风量 q（m3/h）
120	0.014	51.84×v
150	0.023	81.0×v
180	0.032	116.6×v
200	0.040	144.0×v
240	0.058	207.4×v
250	0.063	225.0×v

300	0.090	324.0×v
360	0.130	466.6×v
400	0.160	576.0×v
420	0.176	635.0×v

其中对于不同的功能分区以及室内不同的设计净高度，方形散流器的颈部风速应该满足如表 3.2 所示的要求，例如，对于室内净高为 5m 的办公建筑（公共建筑），方形散流器的设计风速不应大于 7.1 m/s。表 3.3 列出了不同颈部风速下散流器的全压损失，可用于风管的水力校核计算。

表 3.2 散流器颈部最大送风速度（m/s）

建筑类别	室内净高/m			
	3	4	5	6
住宅	4.4	4.6	4.8	5
星级饭店类建筑	5.2	5.4	5.7	5..9
商场、百货类建筑	6.2	6.6	7	7.2
公共建筑	6.5	6.8	7.1	7.5

表 3.3 不同颈部风速下方形散流器的全压损失

颈部风速（m/s)	全压损失（pa）
2.0	14.93
2.5	23.33
3.0	33.59
3.5	45.72
4.0	59.72
4.5	75.58
5.0	93.91
5.5	112.81
6.0	134.37

根据以上条件，矩形散流器选择的约束条件可表示如下：

$$(\frac{w}{1000})^2 \times v \times 3600 \geq \frac{q_{zone}}{n_{diffuser}} \tag{3.28}$$

$$v \leq v_{\max,i,j} \tag{3.29}$$

式中，$v_{\max,i,j}$ 表示不同建筑类别和室内净高下的方形散流器最大颈部风速。

在算法中，我们采用的是从最小尺寸的散流器遍历至最大尺寸的散流器的方法，直到散流器尺寸满足最大风速的约束条件为止。

3.3.2 圆形散流器选型

房间内圆形散流器的单个风口风量计算同方形散流器，从样本库中检索符合要求的风口，如表 3.4 所示，表中单个方形散流器的风量由下式计算得到：

$$q_{diffuser} = \pi \times (\frac{d/2}{1000})^2 \times v \times 3600 \tag{3.30}$$

式中 d 为圆形散流器的名义直径；v 为散流器颈部风速。

表 3.4 标准圆形散流器尺寸及对应风量，颈部风速为 v ㎡/s

名义直径 d（mm）	面积（m2）	风量 q（m3/h）
120	0.011	40.7×v
150	0.018	63.6×v
200	0.031	113.1×v
250	0.049	176.7×v
300	0.071	254.4×v
350	0.096	346.3×v
400	0.126	452.3×v
450	0.159	572.4×v

对于不同的功能分区以及室内不同的设计净高度，圆形散流器的颈部风速要求同表 3.2。表 3.5 列出了不同颈部风速下散流器的全压损失，可用于风管的水力校核计算。

表 3.5 不同颈部风速下圆形散流器的全压损失

颈部风速（m/s)	全压损失（pa）
2	7.28
3	16.37
4	28.27
5	45.45
6	65.44
7	89.09

根据以上条件，方形散流器选择的约束条件可表示如下：

$$\pi \times (\frac{d/2}{1000})^2 \times v \times 3600 \geq \frac{q_{zone}}{n_{diffuser}} \tag{3.31}$$

$$v \leq v_{\max,i,j} \tag{3.32}$$

与矩形散流器的选型方法相同，从最小尺寸的散流器遍历至最大尺寸的散流器的方法，直到所选取的圆形散流器尺寸满足最大风速的约束条件为止。

3.3.3 矩形风口选择

在之前的矩形风口布置生成算法中，已经有较强的鲁棒性，但经过测试仍存在极少数矩形不能够生成满足布置间距要求和最大长宽比要求的情况，例如，一个长为 9，宽为 6 的矩形区域。针对这种情况，我们通过选取矩形风口来满足房间内的送风要求，选择算法如下：

1) 若使用 3.1 节中的布置算法返回空的结果，则说明该矩形的尺寸不满足算法设定的要求，推荐使用矩形风口算法。
2) 仍使用矩形算法对该区域，但将算法中的 $ratio_{max}$ 设置为 2，返回矩形 w ×h 风口的布置结果（m, n）
3) 根据该布置结果求出每个矩形风口负责区域的风量和长宽比：

$$q_{diffuser} = \frac{q_{zone}}{n_{diffuser}} \tag{3.33}$$

$$ratio_{rec} = \max(\frac{w/m}{h/n}, \frac{h/n}{w/m}) \tag{3.34}$$

表 3.6 标准矩形散流器尺寸及对应风量，颈部风速为 v m/s

长（mm）	宽（mm）	长宽比	面积（m2）	风量（m3/s）
150	120	1.25	0.018	64.8×v
180	150	1.20	0.027	97.2×v
200	120	1.67	0.024	86.4×v
200	150	1.33	0.030	108.0×v
200	180	1.11	0.036	129.6×v
240	150	1.60	0.036	129.6×v
240	180	1.33	0.043	154.8×v
240	200	1.20	0.048	172.8×v
300	180	1.67	0.054	194.4×v
300	200	1.50	0.060	216.0×v
300	240	1.25	0.072	259.2×v
360	200	1.80	0.072	259.2×v
360	240	1.50	0.086	309.6×v
360	300	1.20	0.108	388.8×v
400	240	1.67	0.096	345.6×v
400	300	1.33	0.120	432.0×v
400	360	1.11	0.144	518.4×v
420	240	1.75	0.101	363.6×v
420	300	1.40	0.126	453.6×v
420	360	1.17	0.151	543.6×v
420	400	1.05	0.168	604.8×v

4) 从表 3.6 中检索散流器长宽比满足以下条件的矩形散流器并根据散流器面积从小到大排序：

$$ratio_{rec} - 0.2 \le ratio_i \le ratio_{rec} + 0.2 \tag{3.35}$$

5) 按面积从小到大遍历以上排好序的散流器，根据单个风口的风量计算颈部风速，直到颈部风速满足表 3.2 中的要求为止。

3.4 本章小结

本章主要介绍了基于算法的分区设备（主要为风口）的布置方法，实现了风口在不同类型几何区域的均匀布置，为了实现非凸多边形区域的均匀布置，我们使用了多边形凹点计算方法，并利用这些多边形的凹点实现了凹多边形的分割；为了解决房间分区内因障碍物遮挡引起的风口位置布置不佳的问题，我们提出了基于障碍物（梁、柱子等）位置的分区再分割算法，实现了在风口平送方向上对障碍物的规避。最后我们根据相关规范给出了方形散流器和圆形散流器的自动选型算法，该算法以控制送风时的风速为约束条件，选取合适的风口尺寸。

第 4 章 管路路径算法

第 3 章中我们确定了各个房间内风口的布置个数和位置，本章将通过图论的算法生成走廊主管以及将各分区内的各风口之间以及它们同走廊的连接，最后通过修剪将冗余的走廊支路删除得到最后的连接。

4.1 图论基础理论和算法

在本章节后续的内容主将牵涉到许多图论中的经典的基础理论和算法，将在本章节简要介绍。

4.1.1 图的类型

根据图中两个节点 u 和 v 之间的边 edge(u，v)是否等价于 edge (v，u)可将图分为无向图(Undirected Graph)和有向图(Directed Graph or Digraph)[87]，对于无向图，如图 4.1 所示，可认为任意两个节点之间的边不存在方向节点 A 和节点 C 之间的边 edge(A, C)等价于 edge(C, A)；对于有向图，如图 4.2 所示，节点 A 和节点 C 之间是单向连通的，的边 edge(A, C)不等价于 edge(C, A)，edge(A, C)是连通的，而 edge(C, A)是不连通的，有向图也可存在双线连通的情况，例如，图 4.2 中的节点 E 和节点 F。

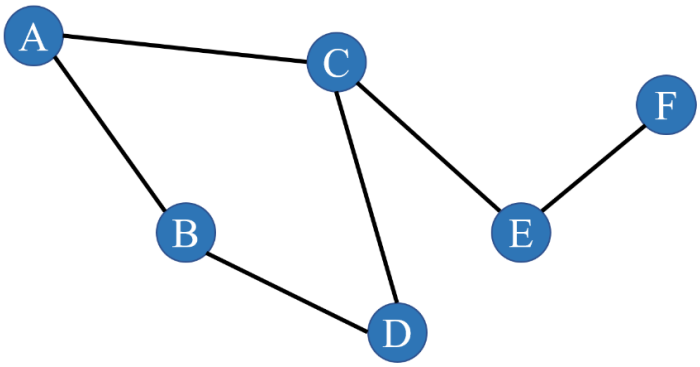


图 4.1 无向图

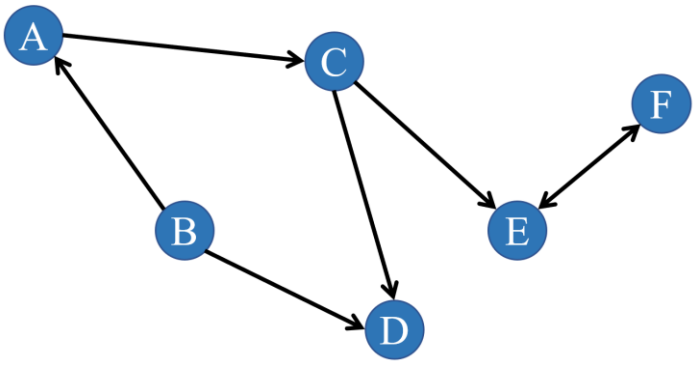


图 4.2 有向图

对于无向图，根据图中所有节点的连通性，可将图分为连通图和非连通图。任意节点都能通过节点之间的路径访问到其他任意节点的图是连通图，例如图 4.1；图 4.3 是非连通图，其中 A，B，C，D 节点和 E，F 节点之间不能互相访问。

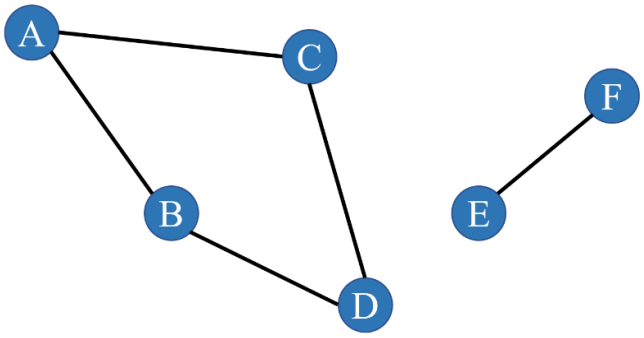


图 4.3 非连通图

根据图中不同的边的权重是否相同可将图分为无权图(Unweighted Graph)和有权图(Weighted Graph)，图 4.1 和图 4.2 都是无权图，可认为连通的两个节点之间的边权重都相等；图 4.4 和图 4.5 都是有权图，对于无向有权图可赋予不同边不同的权重，对于有向有权图不仅可赋予不同边不同的权重，还可赋予两个节点之间不同连通方向的边不同的权重，如图 4.5 所示节点 E 和节点 F 之间的边 edge(E, F)的权重为 4，而 edge(F, E)的权重则为 7。

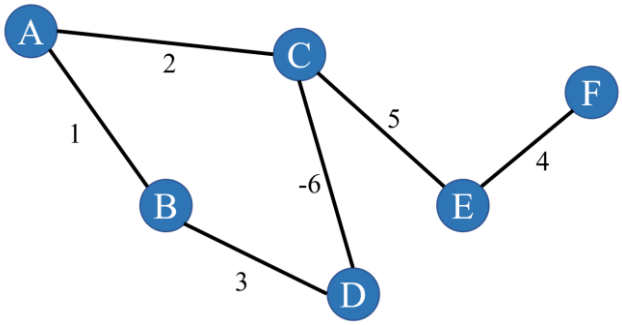


图 4.4 无向有权图

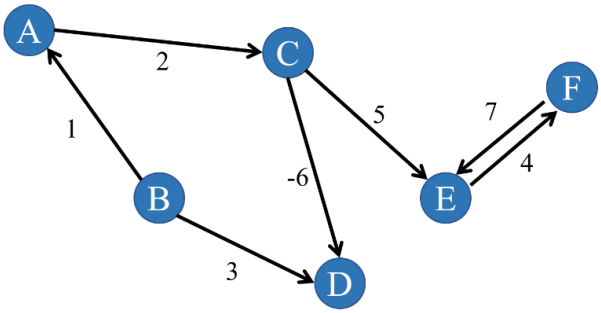


图 4.5 有向有权图

对于有向图，根据是否存在一个节点可经由其他路径访问回自身节点可分为有向无环图(Directed Acylic Graph, DAG) 和有向有环图(Directed Acylic Graph)，图 4.2 是一个有向无环图，图中的所有节点均无法经由其他路径访问回自身节点；图 4.6 是一个有向有环图，其中节点 A，C，D，B 形成一个环(Cycle)，环中的任意一个节点均可经由其他路径访问回自身节点。

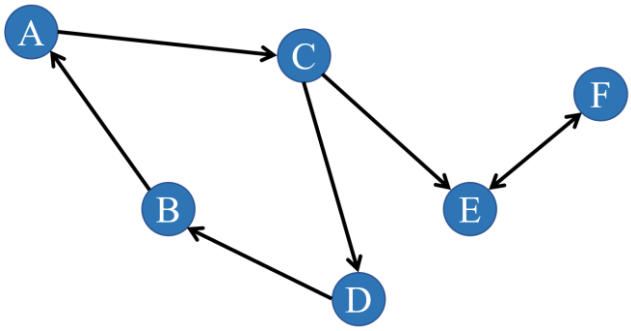


图 4.6 有向有环图

此外还有一些比较特殊的图，例如树(Tree)，树是一种无向图，其定义为任意两个顶点之间存在连通的路径，且该路径是唯一的。如图 4.20 所示，

对于树，存在以下几个等价条件[88]：

1) 图 *G* 中不存在连通的环，那么该图是一棵树；
2) 图 *G* 中不存在连通的环，如果往图 G 中添加一条边，就会形成一个环，那么该图是一棵树；
3) 图 *G* 是一个连通图；如果任意去掉其中一条边，该图变成一个非连通图，那么该图是一棵树；
4) 图 *G* 是一个连通图；其顶点个数为 n，边的个数为 n-1，那么该图是一棵树。

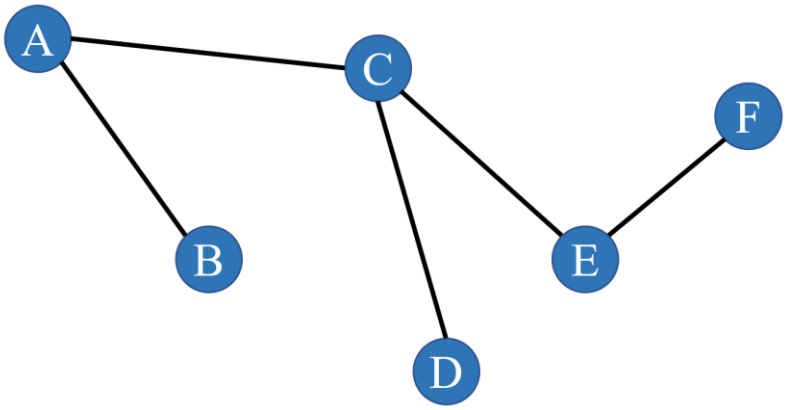


图 4.7 树

对于一棵树，我们还可以指定某个节点作为树的根节点，此时该树叫做有根树(Rooted Tree)，对于一棵有根树，每条边都存在一个指定的方向，即通过父节点指向其子节点，靠近根的节点叫做父节点(Parent Node)，远离根节点的节点叫做子节点(Child Node)，如图 4.8 所示，节点 A 为树的根节点，同时为节点 B 和 C 的父节点，节点 C 又是节点 D 和 E 的父节点。可以根据其距离根经过的节点个数定义该节点的层数，每层包含的节点个数为该层树的宽度，则图 4.8 中的有根树的层数为 4 层，节点 A 位于 1 层，节点 B 和 C 位于 2 层，节点 D 和 E 位于 3 层，节点 F 位于 4 层，树的总高度（或深度）为 4 层。一个节点与其连通的所有下层节点之间的树为称为当前有根树的子树，对于图中的 C 节点，其包含的子树为(C→D, C→E→F)。

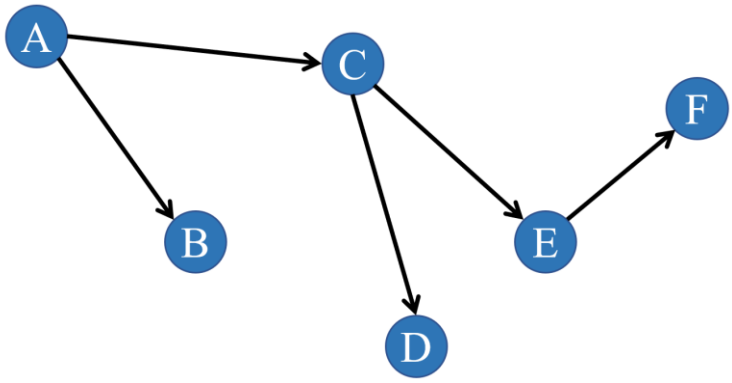


图 4.8 指定根节点的树

另一种比较特殊的树为完全图，其特点为任意两个不同节点之间均有一条边的图，对于一个有 n 个节点的完全图，其边的个数为：

$$edge = \frac{n(n-1)}{2} \tag{4.1}$$

一个节点个数为 4 的完全图如图 4.9 所示，其边的个数为 6。

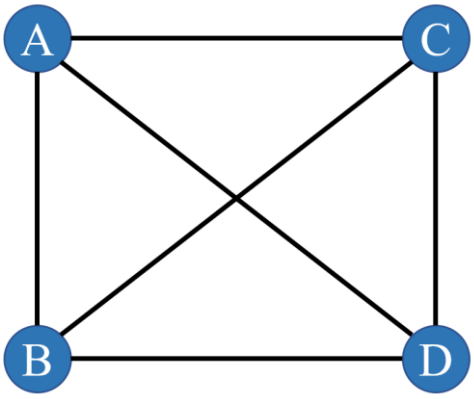


图 4.9 节点个数为 4 的完全图

4.1.2 图的表示方法

1) 连接矩阵法

连接矩阵通过一个 n×n 的矩阵来表示一个有 n 个节点的图，其中每个矩阵元素 $v_{i,j}$ 表示图中边 $edge\left(n_i,n_j\right)$ 之间的连接关系（无权图）或权重（有权图）。对于有向无权图 4.6，可表示如下连接矩阵：

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array}\begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \tag{4.2}$$

其中 1 表示两个节点之间是连通的；0 表示两个节点之间是不连通的；节点与自身的连接矩阵值设为 0。

对于有向无权图 4.5，可表示如下连接矩阵：

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array}\begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} +\infty & +\infty & 2 & +\infty & +\infty & +\infty \\ 1 & +\infty & 3 & \infty & \infty & +\infty \\ +\infty & +\infty & +\infty & -6 & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty \\ +\infty & +\infty & +\infty & +\infty & +\infty & 4 \\ +\infty & +\infty & +\infty & +\infty & +\infty & +\infty \end{bmatrix} \end{array} \tag{4.3}$$

其中矩阵中数值表示两个连通节点之间的边的权重；+∞用来区分连通节点和非连通节点，对于不同的问题可赋予不同的值，当前将非连通节点赋予正无穷大的值可用于求节点之间的最短路径或者最小生成树等。

用连接矩阵来表示一个图具有高效简单的优点，搜索边的权重的算法时间复杂度为$O(1)$，但是其空间复杂度较高，对于节点个数为 n 的图，需要$O(n^2)$的空间复杂度，遍历所有边的时间复杂度为$O(n^2)$。

2) 连接列表法

因为大多数的图都不是完全图，因此其连接矩阵往往是稀疏矩阵，用连接矩阵的方法表示将造成很多空间的浪费，因此可借鉴稀疏矩阵的表示方法来减少空间的占用，对于有向无权图 4.5，可采用如下表示方式：

$$\begin{aligned} &A \to [(C,2)] \\ &B \to [(A,1),(D,3)] \\ &C \to [(D,-6),(E,5)] \\ &D \to [] \\ &E \to [(F,4)] \\ &F \to [(E,7)] \end{aligned} \tag{4.4}$$

式中$n_i \to [(n_{j1},w_1),(n_{j2},w_2),...,(n_{jk},w_k)]$表示节点$n_i$与其他所有 k 个连通节点$(n_{j1},n_{j2},...,n_{jk})$之间的连接关系；$w_k$表示$n_i$与$n_{jk}$之间的连接权重。

可以看到通过连接列表的表示方法，只需要将有效的连接关系表示，因此可节省空间复杂度，但是随着图的边的增加这一优势将逐渐减少，而且该算法搜索边的权重的算法时间复杂度为$O(edge)$，随着图的边的增加，搜索所需要的时间相比矩阵表示法将大幅增加。

3) 边表示法

连接列表法通过描述一个节点与其他节点之间的连接关系来描述一个图，其首要描述对象是图中的节点，边表示法的首要描述对象是图中的边，对于有向无权图 4.5，可采用如下表示方式：

$$[(A,C,2),(B,A,1),(B,D,3),(C,D,-6),(B,E,5),(E,F,4),(F,E,7)] \tag{4.5}$$

这种表示方法描述方法较连接矩阵法和连接列表法简单，但是应用范围较少，因此这种描述方法没有一个较强的结构性，不同边的描述顺序没有固定的要求。使用该方法，随着图的边的增加，搜索所需要的时间相比矩阵表示法将大幅增加，搜索边的权重的算法时间复杂度同连接列表法相同，为$O(edge)$。

4.1.3 广度优先搜索算法

广度优先搜索算法(Breadth First Search, BFS)[89]是一种经典的搜索算法，可通过根节点在树的宽度方向搜索遍历至所有节点，搜索算法如下：

对于如图 4.8 所示的有根树，其搜索过程如下：

首先将根节点 A 加入队列，同时通过 A 节点随机访问至其子节点 B 和 C，将 B 节点和 C 节点加入队列，依次访问 B 节点和 C 节点的子节点，但是 B 节点不存在子节点，因此将 C 节点的子节点加入队列，然后访问 C 节点的子节点 F，搜索结束，整个搜索过程如图 4.10 所示。

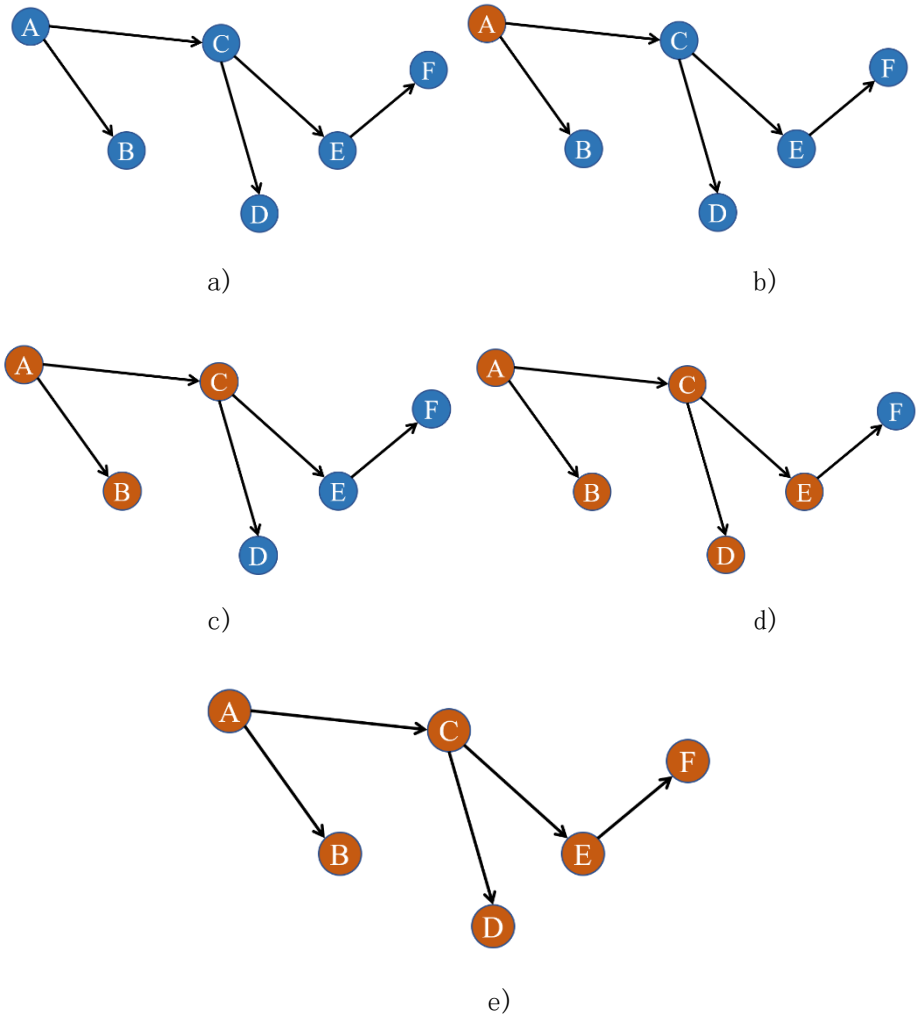


图 4.10 广度优先搜索过程

4.1.4 深度优先搜索算法

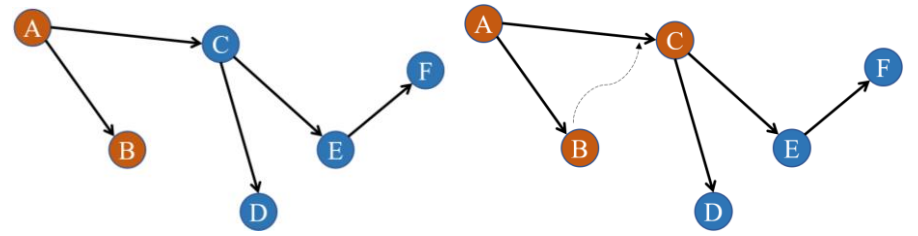
深度优先搜索算法(Depth First Search, DFS)[90]是一种经典的搜索算法，可用于计算图的最小生成树、寻找图中的环、判别一个图是否为二分图、寻找图中的强连接、根据拓扑对图进行排序、寻找图中的桥(Bridge)和关节点(Articulation

Point)、在网络流中寻找强化路径、生成迷宫等相关问题。其采用了回溯的思想，可通过根节点在树的高度方向向下搜索遍历至所有节点，搜索算法如下：

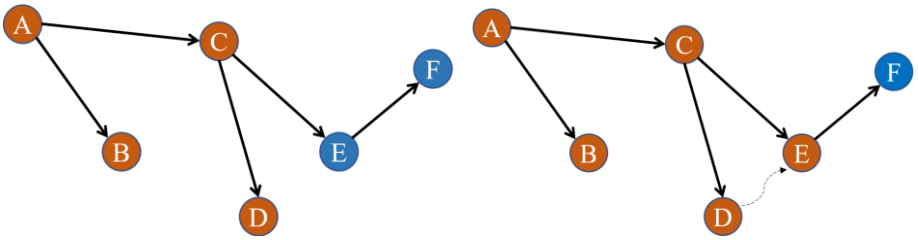
1) 从根节点开始访问树，然后随机访问其相连的子节点；
2) 重复步骤 1)，直到当前节点不存在相应的子节点为止，向其父节点回溯至存在未被访问的其他子节点
3) 重复步骤 1)和 2)，直到所有的节点的父节点都无法回溯为止（即所有的子节点均被访问）停止搜索算法。

对于如图 4.8 所示的有根树，其搜索过程如下：

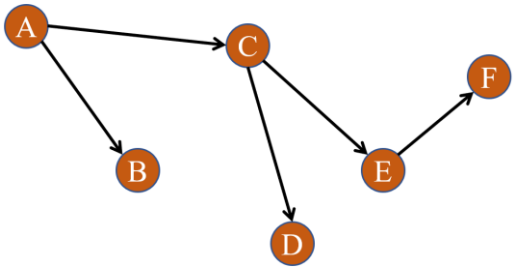
首先通过 A 节点随机访问至其子节点 B，但是 B 节点不存在子节点，因此回溯至其父节点 A，继续从未被访问的节点 C 开始搜索，搜索至 D 节点后回溯至 C 节点的另一个子节点 E，然后搜索至 E 节点的子节点 F，搜索结束，整个搜索过程如图 4.11 所示。



a)　　b)



c)　　d)



e)

图 4.11 深度优先搜索过程

4.1.5 最小生成树算法

最小生成树是图论中的概念，假设有一个图 G，其所有顶点可以表示为 $v(G)$，若 T 是由 G 的顶点和边连接成的图，即 T 图 G 的一个子图，若 T 是一棵树（不含有环的连通图），且满足 $v(T)=v(G)$，则 T 是 G 的一个最小生成树，如图 4.12 所示。每个连通图可存在一个或多个生成树，但是这些生成树都满足一个相同的性质：可经过严格推导[图论和网络流]证明每个连通图都有生成树且生成树的顶点满足以下条件：

$$\varepsilon(T)=v(G)-1 \tag{4.6}$$

式中，$\varepsilon(T)$ 为图 T 的边的个数。

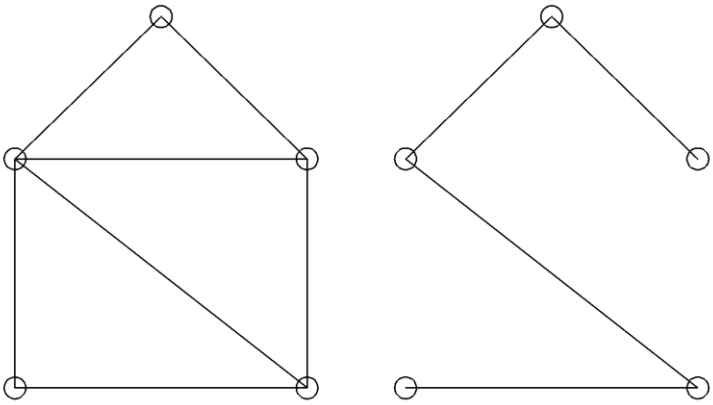


图 4.12 生成树示例

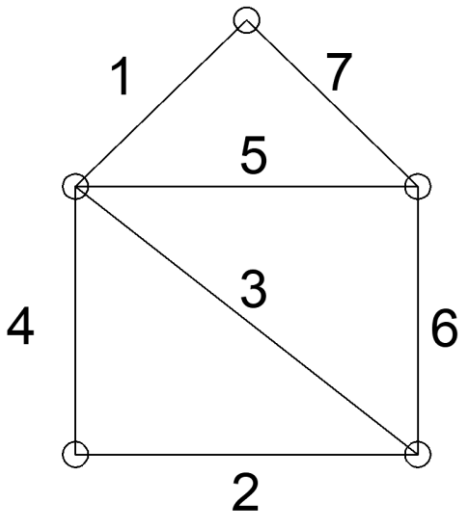


图 4.13 添加权重后的有权图

现在给图上的每条边加上权重，成为有权图，如图 4.13 所示，对于有权图，其不同生成树的边的权重之和可能不同，如图 4.13 所示，图 4.14 a）和图 4.14 b）均有 4 条边，a）的边的权重之和为 22，b 的边的权重之和为 13。对于一个有权连通图，其存在一个生成树，使得其边的权重之和不大于其他所有生成树，这一生成树就叫做有权连通图的最小生成树木。对于一个有权图，因为其边的权重的特点，也可能存在多个生成树的情况。

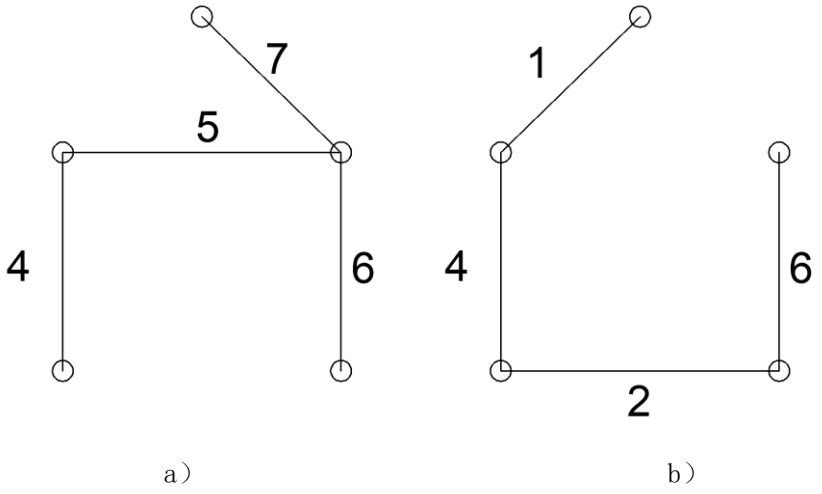


图 4.14 有权图的生成图

为了得到有权连通图的最小生成树，可以使用 kruskal 算法[90]，该算法的复杂度为$O\left(v^2\log_2 v\right)$，其中$v$为有权连通图中顶点的个数，其实现过程如下：

1) 将图 G 中的所有边$\varepsilon(G)$ 按照权重从小到大排序
2) 将排好顺序的边依次添加至图 T 中，若该边与图 T 中已有的边形成环，则不将这条边放入图 T 中，继续遍历下一条边。
3) 直到图 T 中的边数满足$\varepsilon(T)=v(G)-1$的条件，完成最小生成树的生成，退出边的遍历。

以图 4.15 所示的有权连通图为例，该连通图有 5 个顶点，因此其最小生成树有 4 条边，其生成最小生成树的过程如图 4.15 所示，首先将各条边按照权重从小到大排序：1，2，3，4，5，6，7，然后依次将这些边添加到生成树中，当添加至权重为 4 的边时，该边与现有的边长为 2，3 的边形成了环，这条边将不被放入生成树中，直到边长为 5 的边被添加至生成树中，此时该图中没有环且边的个数为 4，该图即为原图的最小生成树。

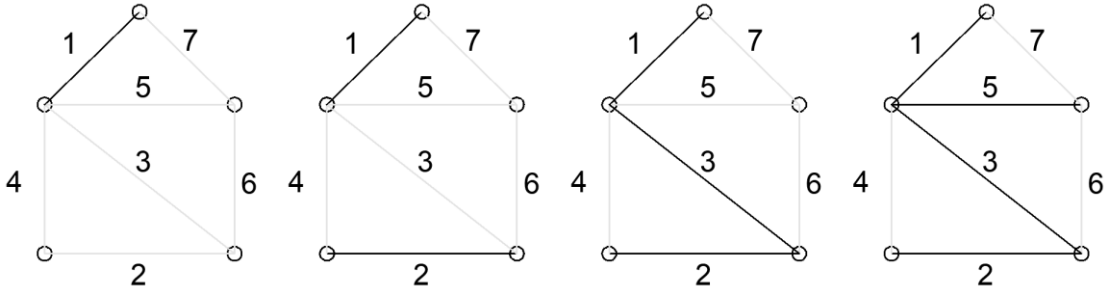


图 4.15 最小生成树生成过程

4.1.6 单源最短路径算法

在布管连接问题的优化过程中，常常需要计算图中从一个节点出发到其余节点的最短路径，这涉及到图论中的单源最短路径算法，常用的算法有 Dijkstra 算法和分支界限法等。Dijkstra 算法是常用的一种贪心算法[90]，其计算时间复杂度为$O(N^2)$，该算法时效性较好，如果采用优先队列对算法进行进一步优化，可使得时间复杂度降低到$O(E \cdot log(V))$，该算法要求图中的边没有负的权值，符合管路连接问题中的情况，因此本文中的单源最短路径算法选用 Dijkstra 算法。

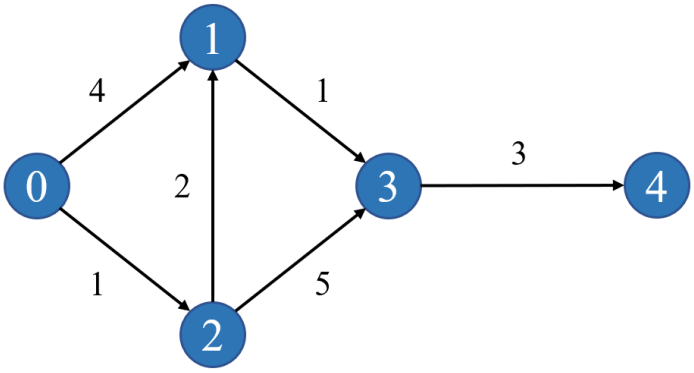


图 4.16 有向图的单源最短路径问题

考虑如下实际问题：如图 4.16 所示的一个有向图，每条边都有对应的方向和权重，现要找到节点 0 和其他所有节点的最短路径，通过 Dijkstra 算法的解法如下：

首先根据有向图生成如下的连接矩阵：

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} \infty & 4 & 1 & \infty & \infty \\ \infty & \infty & 1 & \infty & \infty \\ \infty & 2 & \infty & 5 & \infty \\ \infty & \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \end{array} \tag{4.7}$$

初始化以下最短路径矩阵：

$$\begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix} \end{array} \tag{4.8}$$

从矩阵的第一行开始检索，其中每一次搜索将对当前行的值进行更新，每一列代表从 0 节点走到当前节点的最短距离，因为要求节点 0 同其他节点的单源最短路径，因此将 0 节点所在的列的距离设置为 0。

首先将所有节点加入到开放列表(Open List)当中，并设置一个初始值为空的封闭列表(Closed List)。对第一行的距离进行更新，找到开放列表中存在的节点中在当前行中距离值最小的元素，记为 i，以该距离 $d_{0,i}$ 为基准，通过以下公式更新开放列表中其他节点的最短距离：

$$d_{0,k} = \min(d_{0,k}, d_{0,i} + d_{i,k}) \tag{4.9}$$

当前第一行距离值最小的节点为 0，使用以上公式对节点 0 完成一次距离更新得到以下距离矩阵：

$$\begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 0 & 4 & 1 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix} \end{array} \tag{4.10}$$

当该节点完成了对其他节点的最短距离更新后，将该节点从开放列表中移除，加入封闭列表。现在开放列表中距离最短的节点为节点 2，因此使用节点 2 对第二行距离进行更新，得到以下距离矩阵：

$$\begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix}$$
$$\begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 0 & 4 & 1 & \infty & \infty \\ 0 & 3 & \infty & 6 & \infty \\ 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix} \tag{4.11}$$

将节点 2 从开放列表中移除，加入封闭列表继续对其余节点进行更新，最终得到以下的距离矩阵。在本算法中我们初始化了两个 n×n 的矩阵，将会给计算机的内存造成压力，可通过初始化一个 1×n 的矩阵对改矩阵进行原地更新最短距离。

$$\begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix}$$
$$\begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 0 & 4 & 1 & \infty & \infty \\ 0 & 3 & 1 & 6 & \infty \\ 0 & 3 & 1 & 4 & \infty \\ 0 & 3 & 1 & 4 & 3 \end{bmatrix} \tag{4.12}$$

4. 2 走廊主管生成算法

本节将通过图 4.17 所示的案例来生成最后的管路连接信息，首先我们在建筑的走廊区域生成主管道，然后将其他区域内的设备连接至该虚拟的主管道，最后通过算法生成所需的最短主管道，使其符合设计要求，需要指出的是，该主管道是虚拟的，可以是水管路也可以是风管路，最终需要通过算法确定该主管道需要保留的部分。

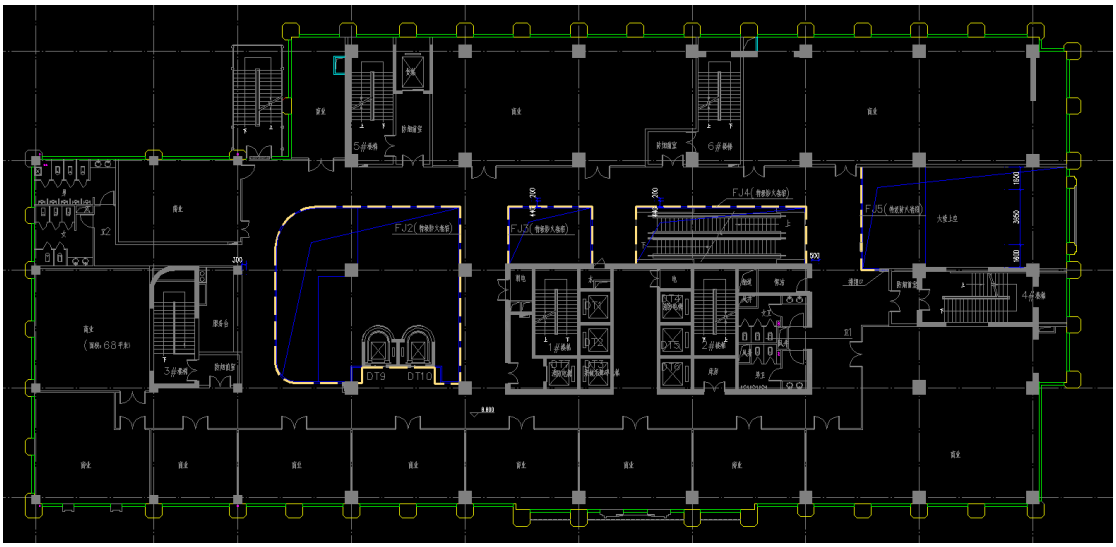


图 4.17 管路生成案例

4.2.1 双线轮廓模型转单线模型

首先通过 idf 文件提取出走廊的几何信息，如图 4.18 所示，为了将几何信息转化为具有拓扑信息的连接关系，需要将具有双线轮廓的几何图形转换为单线的模型。转换的过程首先需要对该几何区域进行分割，在 3.2.2 节中提到过根据多边形凹点进行分割的方法，这里的分割方法与之前的算法类似，同样需要寻找到几何图像的凹点，不同的是对于走廊区域往往存在闭合的情况而形成“复连通区域”[91]。本节所选取的案例存在两个闭合区域，因此也属于“复连通区域”。

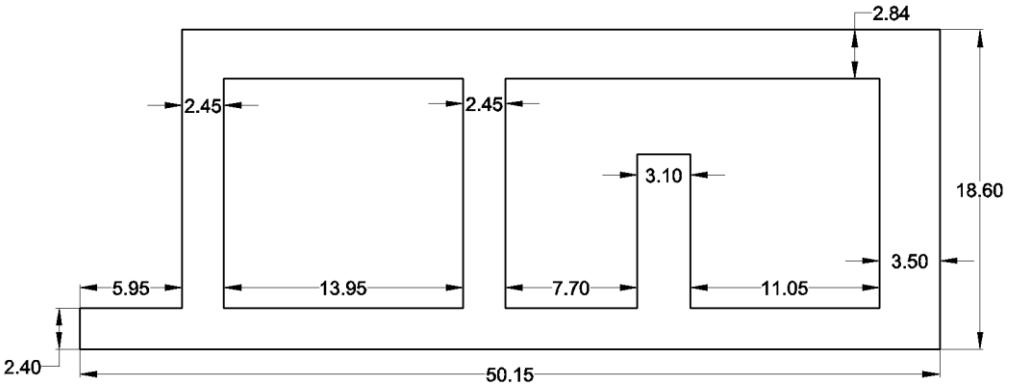


图 4.18 建筑平面走廊几何信息

对于“复连通区域”，需要将图形分为内部和外部分开求解，步骤如下：

1) 首先求图形外部轮廓的凹点，可采用 3.2.2 节中的代数法进行求解，对于上述图形，求得的凹点为 1 点，如图 4.19 所示。
2) 得到“复连通区域”内部的两个多边形，分别求出其“非凹点”，如图 4.19 所示，两个内部多边形的“非凹点”分别为点 2，3，4，5 和点 36，7，8，9，10，11。

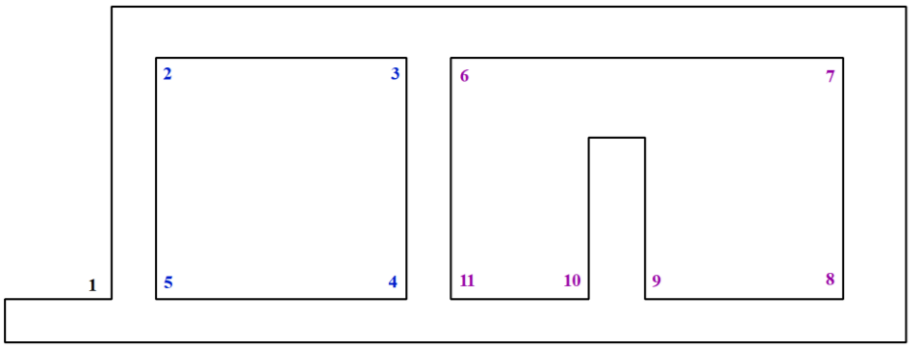


图 4.19 “复连通区域”的凹点

3) 根据以上求出的所有点分割“复连通区域”，根据 3.2.2 所述绘制最短延长线的方法将“复连通区域”分割为多个凸多边形，如图 4.20 所示，最终“复连通区域”被分割为多个矩形。

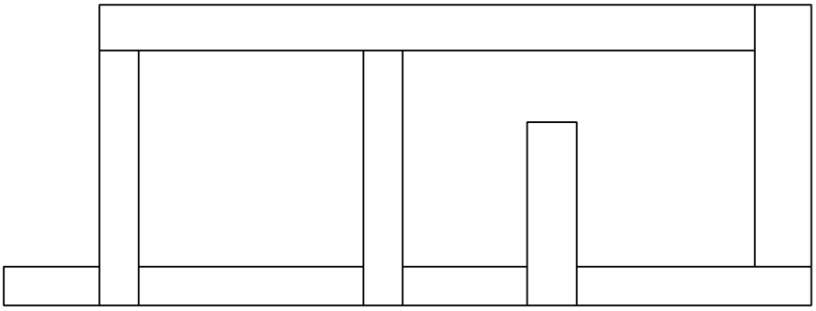


图 4.20 “复连通区域”分割结果

完成“复连通区域”的分割后，首先从这一系列相邻的狭长的矩形中提取出中心线，提取的方法为遍历每个矩形，对每个矩形取其两条短边上的中点，并将其连接，得到如图 4.21 所示的结果。

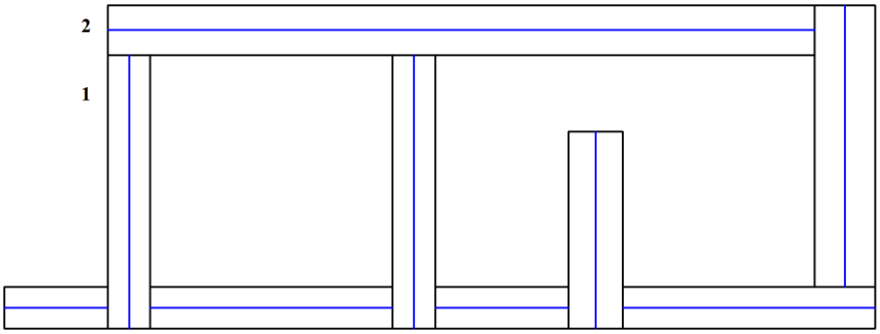


图 4.21 相邻分割矩形中心线生成

目前得到的中心线是多条不相交的线段，需要通过“延伸”和“修剪”的操作将其合成为一条多段线。合成的方法如下：首先判断矩形之间的相邻关系，例如，图 4.21 中的矩形 1 和矩形 2 为相邻关系；然后将“复连通区域”分割步骤中通过凹点延长线而形成完整边的矩形的中心线进行延长并交至另一矩形中心线位置，如图 4.22 所示，对矩形 1 的中心线延长，蓝色虚线为“延伸”部分；最后将另一矩形中被交点分割的短于分割矩形时的延长部分的长度的一边将被“修剪”，图中矩形 2 的中心线的红色虚线为被“修剪”部分。对所有的相邻矩形实施上述步骤，得到如图 4.23 所示的最终结果，该单线图由 6 条线段组成。

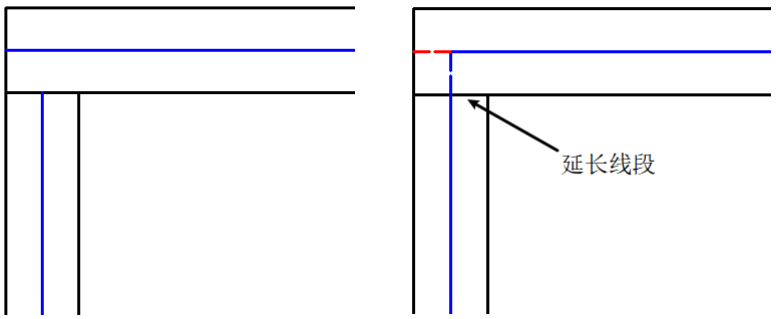


图 4.22 相邻分割矩形中心线“延伸”和“修剪”

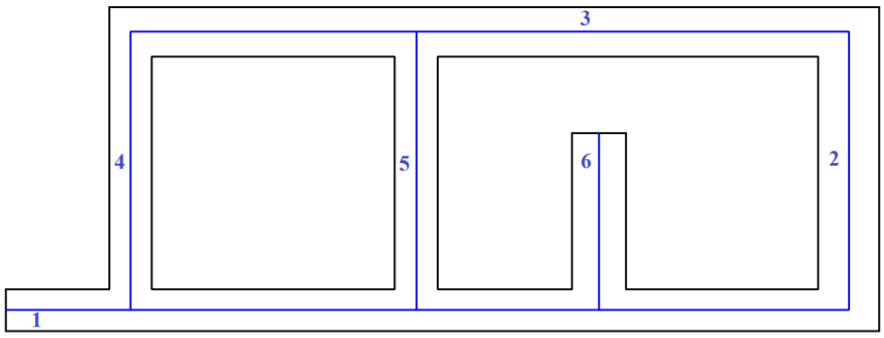


图 4.23 “复连通区域”最后生成的单线图

通过本方法得到的单线图为干管在建筑走廊中可行进的理想路线，实际工程中管道的位置应该能够做到灵活变动以达到不同功能管线之间的交叉和重叠，为此我们提出了“网格化”来解决这一问题。在本节相邻分割矩形中心线生成过程中，我们使用的是矩形短边的中点作为矩形，为了改变该线段的位置，算法中我们通过改变，线段距离中心线的偏移量来生成网格，对于每一个矩形，可按如下公式设置偏移量 Δl：

$$|\Delta l| = k \cdot \Delta x, k = 0,1,2,... \tag{4.13}$$

$$|\Delta l| \le \frac{h}{2} - dist_{min} \tag{4.14}$$

式中，Δx 为设定的单位偏移量，m；h 为矩形短边的高度，m；$dist_{min}$ 为偏移线段距离矩形长边距离的最小值，m。

设定 $\Delta x = 0.4$，$dist_{min}$=0.5，得到如图 4.24 所示的单线图网格。

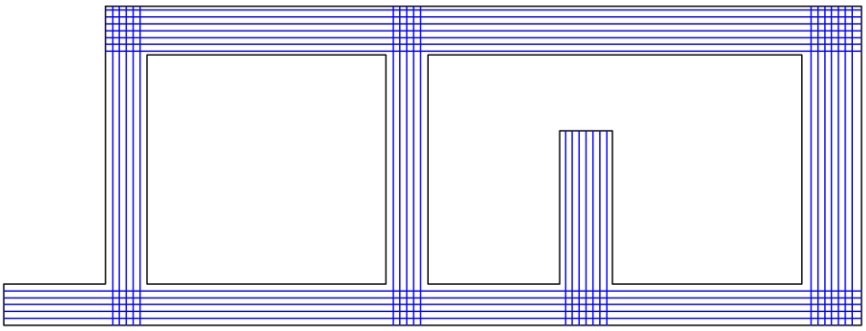


图 4.24 “复连通区域”单线图网格

图 4.24 表示的是对于本案例生成的单线图网格，算法可在每一个网格的交叉点选择是否拐至另一个矩形来实现网格的遍历，图 4.25 表示的是通过单线图网格生成的另一种单线图模型。

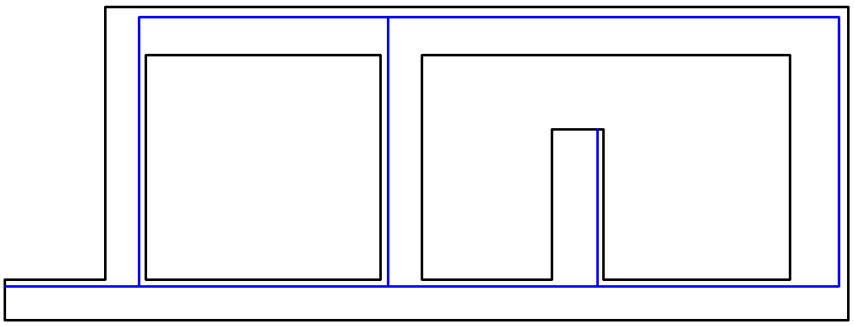


图 4.25 通过网格生成的另一种干管路线图

4.2.2 单线模型转拓扑模型

得到单线模型后计算机只知道每个线段端点的坐标，考虑到走廊几何信息不同可能导致单线模型的各异，需要通过一个通用的算法将任意的单线模型转化为无向图的拓扑，转换的算法如下：

1) 遍历单线模型中的每一条线段，找出其内部所有与其他线段的 n 个交点（不包括该线段的两个端点），并通过这 n 个交点将该线段重新分为 n+1 段，更新到原来的单线模型中。
2) 重新遍历新的单线模型中的所有线段，根据每条线段的两个端点建立图的节点和边，边的权重由两个节点之间的距离确定：

$$w_i = \sqrt{(x_{i,1} - x_{i,2})^2 + (y_{i,1} - y_{i,2})^2} \tag{4.15}$$

式中，w_i 表示第 i 条线段的权重；$(x_{i,1}, y_{i,1})$ 和 $(x_{i,2}, y_{i,2})$ 分别表示第 i 条线段两个端点的坐标。

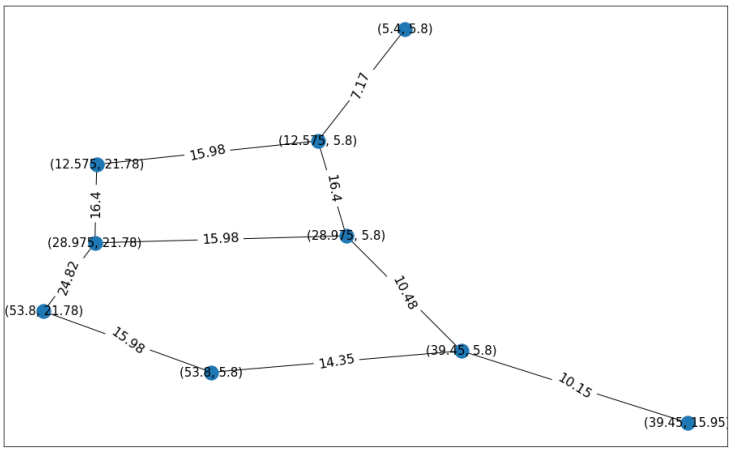


图 4.26 单线模型转化成转化为拓扑图

最终生成的拓扑图如图 4.26 所示，蓝色节点对应原单线模型中的各线段，节点处标注其位置信息，节点与节点之间标注两个节点之间的权重，即距离。将各节点按照其坐标信息绘制成更为直观的拓扑图，如图 4.27 所示。

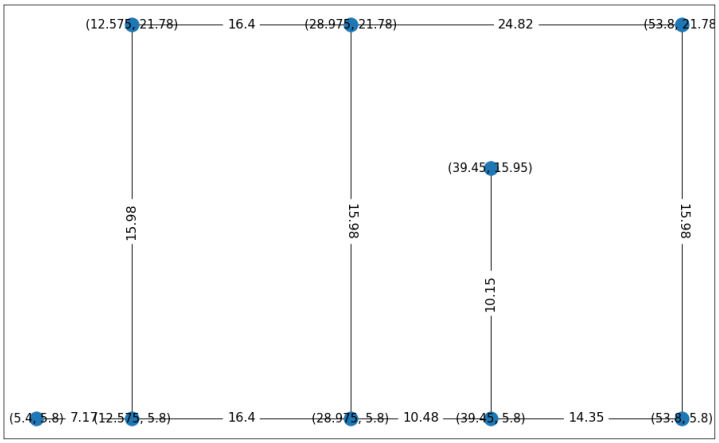


图 4.27 按照位置信息绘制的拓扑模型

为了测试将单线模型转化为拓扑的算法，建立了如图 4.28 所示的更复杂的模型，模型由 5 条线段组成，需要考算法识别线段之间的连接以及相交关系，最终生成的拓扑模型如图 4.29 所示，可以看到该模型正确地反映了原线段中的交点处的连接关系以及 5 个“叶节点”（仅与另外一个节点相连的节点）。

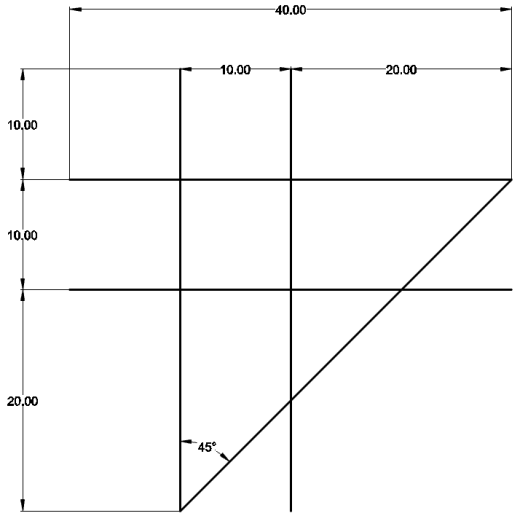


图 4.28 拓扑生成测试模型

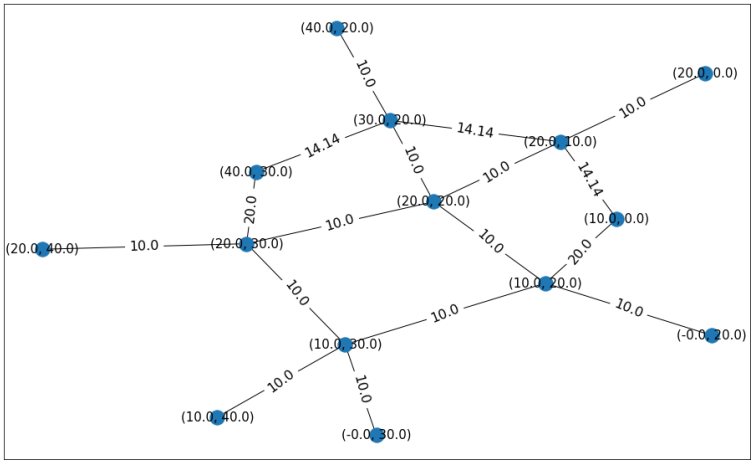


图 4.29 测试模型生成的拓扑结构

4.3 平面最短路径算法

4.3.1 图的生成

对于行和列按照一定距离分布的节点，我们通过以下方法生成图。如图 4.30 所示，该区域中共有 3×3 个节点，节点只会连接其在水平或数值方向上最邻近的点，因此我们不需要将这 9 个节点两两连接形成全连接图，具体生成连接图的算法如下：

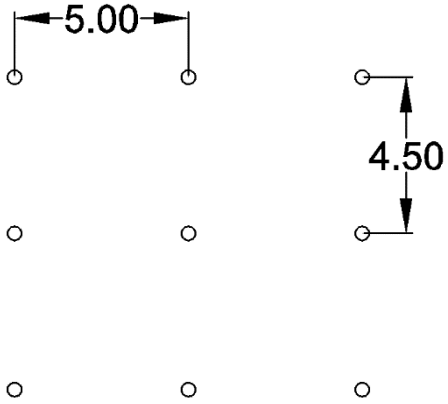


图 4.30 图生成案例

1) 从首行的第一个节点开始遍历每一个节点，

2) 对于每个节点，分别向 x 正方形和 y 负方向寻找最近点，若存在，则将当前节点与寻找到的最近节点相连，

3) 直到遍历到末行最后一个节点，结束遍历。

4) 为每条边添加权重，其值为两个节点之间的曼哈顿距离，若两个节点之间存在障碍物，则需要根据第 5 章的障碍规避算法确定两个节点之间的实际距离。

最后生成的图的结果如图 4.31 所示，该图由 9 个节点和 12 条边组成，其中 6 条边的长度为 5 m，6 条边的长度为 4.5 m。

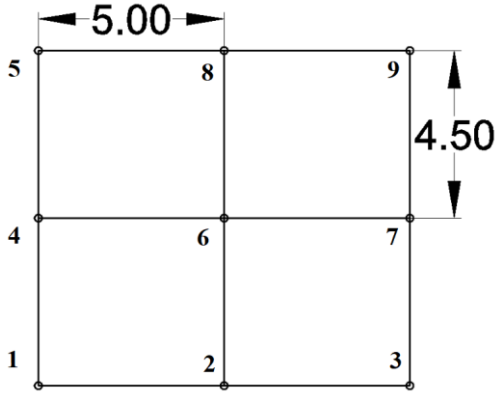


图 4.31 生成图的结果

4.3.2 遍历最小生成树

文献[90]中介绍的最小生成树的方法只能获取当前有权连通图的一个最小生成树的解，针对暖通空调系统中管路的布置时，由于各节点往往是均匀布置，因此各个节点之间的距离往往相等，因此最小生成树的个数往往较多，我们需要一

种算法能够实现有权连通图所有最小生成树的遍历，通过目标函数从这些最小生成树中实现最优解的选取。

遍历最小生成树的最直观的方法就是先通过求出图的一个最小生成树的权值之和，然后通过算法遍历图的所有生成树，最后从所有生成树中筛选出其中权重之和与之前通过 kruskal 算法计算得到的一个最小生成树的权重之和相等的生成树。其中，遍历生成树的方法可采用类似广度优先遍历的算法[参考文献]：

但是，这种方法遍历了连通图的所有生成树，计算了大量的无效生成树，造成了大量计算资源的浪费。为了实现最小生成树遍历这一过程的同时又要保证算法的复杂度不至于过高，我们提出了如下的最小生成树遍历算法：

1) 将图 G 中的所有边 $\varepsilon(G)$ 按照权重从小到大排序并按照权重大小，若若干边的权重相同，将他们放在同一个组内，由此得到一系列由边的组合 $s_1=\{\varepsilon(G)\,|\,lenth(\varepsilon(G))=l_1\}$ ， $s_2=\{\varepsilon(G)\,|\,lenth(\varepsilon(G))=l_2\}$ ， … ， $s_i=\{\varepsilon(G)\,|\,lenth(\varepsilon(G))=l_i\}$ ，…其中 $l_1<l_2<...<l_i<...$。
2) 将排好顺序的边按组添加至图 T 中，若该组边的个数小于图可继续放置的边的个数，继续遍历下一条组边。
3) 若遍历至一组边的个数大于图 T 可继续放置的边的个数，则从这组边中任意选取能够将图 T 的边数要求满足 $\varepsilon(T)=v(G)-1$ 的个数的边，若该组边中共有 n 条边，图 T 距离满足生成树边数要求仍需 m 条边，则一共有以下种可能的组合：

$$C_n^m=\frac{n!}{m!(n-m)!}=C_n^{n-m} \tag{4.16}$$

4) 若某一种边的组合情况满足生成树中规定的要求，即不与图 T 中已有的边形成环，则该组合与图 T 中已有的边形成的图为图 G *的*一个最小生成树。
5) 完成最小生成树的生成，退出边的遍历。

如图 4.32 所示，该连通图有 9 个节点，12 条边，边与边之间的权重用两个连接节点之间的距离表示，通过本算法能够在 $C_6^2=15$ 中组合中生成 9 个最小生成树，如图 4.33 所示。若采用遍历所有生成树的算法，则最后的输出结果有 192 个生成树，搜索的效率仅为 4.69%，若连通图中有 4×4 个节点，24 条边，则有 100352 个生成树，而最小生成树的个数为 64，搜索效率仅为 0.06%。

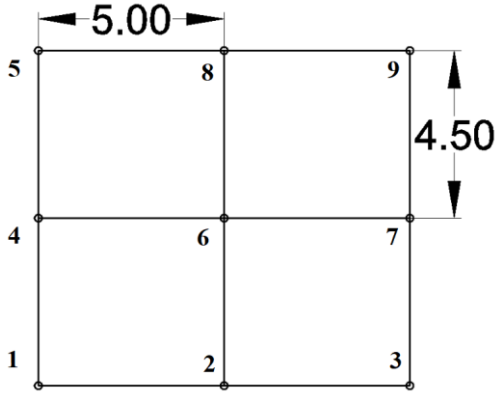


图 4.32 遍历最小生成树案例

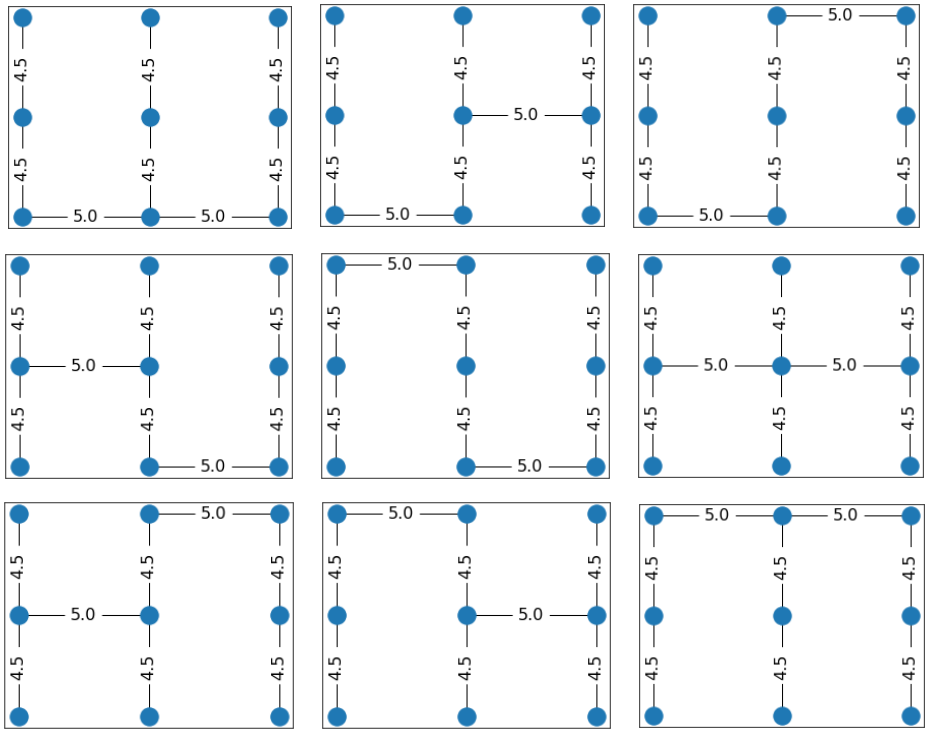


图 4.33 最小生成树遍历结果

4.3.3 最短路径之和

本节将讨论如何在生成树中寻找出以其中一个节点 n_i 为起点的输配最短的输配路径的和，这里的输配路径的和指的是以 n_i 为起点，分别计算 n_i 到其他节点的输配距离（与本身的距离可按照 0 计算），并求和，计算公式如下：

$$l_i = \sum_j path(n_i, n_j) \tag{4.17}$$

式中， $path(n_i, n_j)$ 为生成树中节点 n_i 与节点 n_j 之间路径的长度。

以图 4.33 中的第一个遍历结果为例子，其左下角的 1 号节点到其他节点的距离之和为：

$$4.5+(4.5+4.5)+(4.5+4.5+5)+(4.5+5)+5+$$
$$(4.5+4.5+5+5)+(4.5+5+5)+(5+5)=85.5$$

通过本方法得到的图 4.33 的 9 个最小生成树的遍历结果以 n_i 节点为输配起点的输配距离和依次如表 x 所示，可以看到其中 6 号最小生成树以 6 号节点为输配起点的输配路径之和最短，这也比较符合直观的感受，即将输配中心放置在中心的位置能使得各路支管比较均衡。

表 x 各个最小生成树中以节点 i 为起点的输配路径之和

最小生成树 / 节点	1	2	3	4	5	6	7	8	9
1	85.5	94.5	112.5	121.5	166.5	103.5	121.5	148.5	139.5
2	70.5	79.5	97.5	79.5	97.5	88.5	106.5	106.5	124.5
3	85.5	121.5	166.5	94.5	112.5	103.5	148.5	121.5	139.5
4	108	117	135	90	135	72	90	117	108
5	139.5	148.5	166.5	121.5	112.5	103.5	121.5	94.5	85.5
6	93	75	93	75	93	57	75	75	93
7	108	90	135	117	135	72	117	90	108
8	124.5	106.5	97.5	106.5	97.5	88.5	79.5	79.5	70.5
9	139.5	121.5	112.5	148.5	166.5	103.5	94.5	121.5	85.5

4.3.4 点线最短路径算法

根据以上给出的基于遍历最小生成树的最短路径之和的算法，我们可以解决如下图 4.34 所示的工程问题，一个区域内有若干风口，按照一定位置均匀排布，距离区域一定距离范围内有主风管，现要使得主风管与区域内的每个风口相连，，在保持风管敷设距离较少的前提下并且能够实现较好的管道水力平衡。

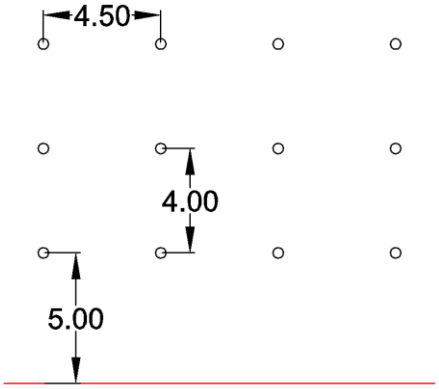


图 4.34 风口和主管连接问题

在 4.3.3 中我们已经讨论过将路径作为权重的有权连通图的生成树遍历方法，对于本问题可采用类似的方法进行求解：

1) 按照与主管垂直的方向遍历区域中的点（对于本案例，即按列遍历），找到主管上离每一列最近的点。
2) 将这些点作为区域中节点的扩展，按照 4.3.1 中的方法生成图并按照 4.3.3 中的方法根据两个点直接路径的距离设置权重，其中，，因此完全由主管上的节点连接成的边的权重设为 0，最终得到如图 4.35 所示的结果。

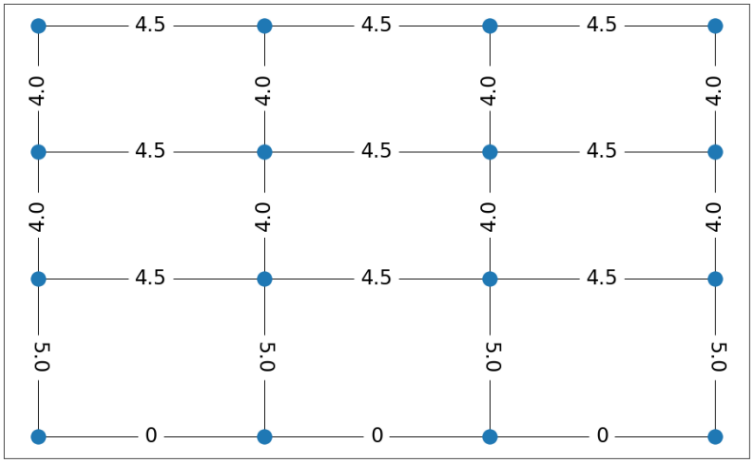


图 4.35 风管路径规划问题有权连通图

得到图 4.35 所示的有权连通图之后，仍旧通过 4.3.2 中的方法遍历所有生成树，由于输配起点可选主管上的任意一点，因此可通过遍历本小节算法步骤 1）中主管上离每一列最近的点来实现最优解的获取，最终我们得到如图 4.36 所示的最短的两种路径方案，两种方案从输配起点开始的最短路径之和均为 162 m，由于本案例对称的特点，输配起点可选骤 1）中主管上离每一列最近的点中间两个点的任意一个。

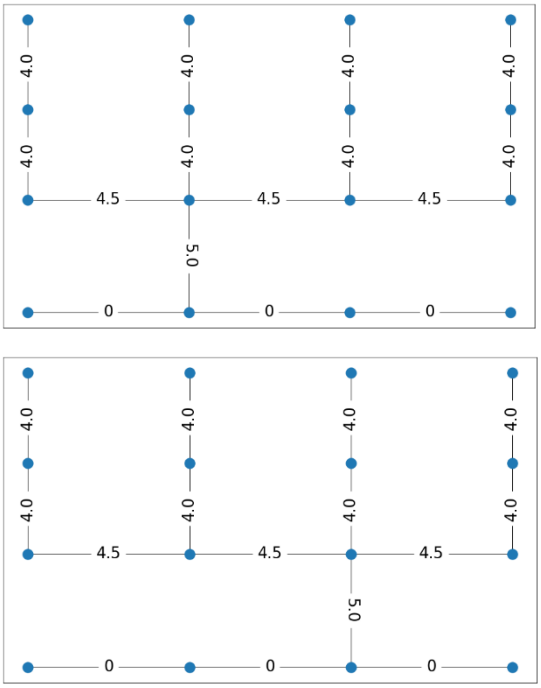


图 4.36 最短的两种路径方案

4.3.5 单点对多点最短敷设路径算法

在 4.3 节中我们介绍了最短路径的算法，该路径为“虚拟路径”，本节将讨论是如何将输配介质通过最短的管路敷设代价从输配起点输配至规划好的各个输配终点。首先考虑以下问题，如图 4.37 所示，黑色圆圈所示的为需要布置风口的位置，即风管需要输配到的位置，即各个输配终点，红色圆圈表示输配的源头，也就是输配的起点，为了表示终点和起点之间的多种连接可能，我们定义了如下的连接方法：

在一个平面内，终点(x_i, y_i)可以以下两种方式走到起点(x_0, y_0)，两种方式的单挑路径长度均可用同一个曼哈顿距离表示：

route1：$(x_i, y_i) \rightarrow (x_i, y_0) \rightarrow (x_0, y_0)$

route2：$(x_i, y_i) \rightarrow (x_0, y_i) \rightarrow (x_0, y_0)$

$$dist_{route1} = dist_{route2} = dist_{manhattan}((x_i, y_i), (x_0, y_0)) = |x_i - x_0| + |y_i - y_0| \quad (4.18)$$

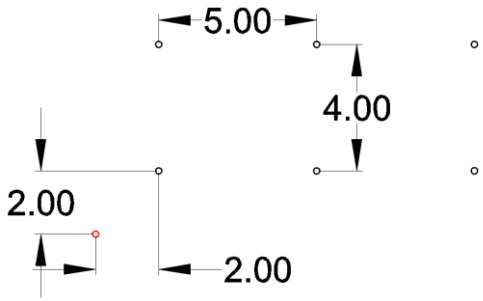


图 4.37 风口布置问题

图 4.38 显示了不同终点通过两种方式到达起点的路径。遍历每个点，我们生成了如图 4.39 所示的“网络”，对于网络上的每一个终点，均有 2 条路线可到达起点，对于 n 个终点，需要有 n 条路径，共有2^n种路径组合可使得每个终点与起点联通。现在要从这2^n种路径组合中求出总路径最小的组合，当某些点经过的路径与其他一条过多条路径有重复段的时候，这些重复的部分将只参与一次计算。

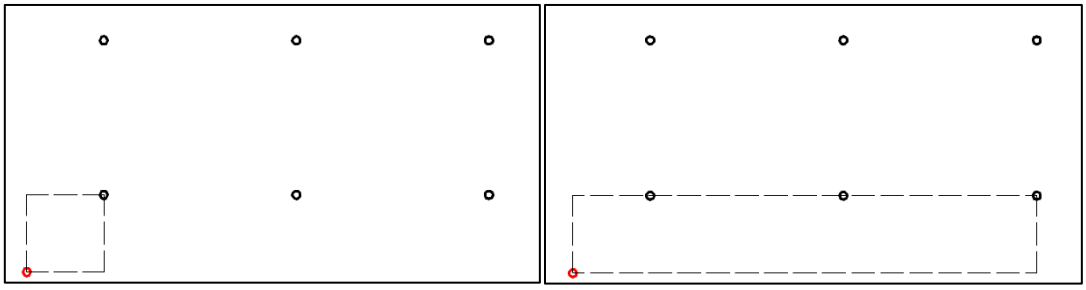


图 4.38 不同终点可走的两种到达起点的路径

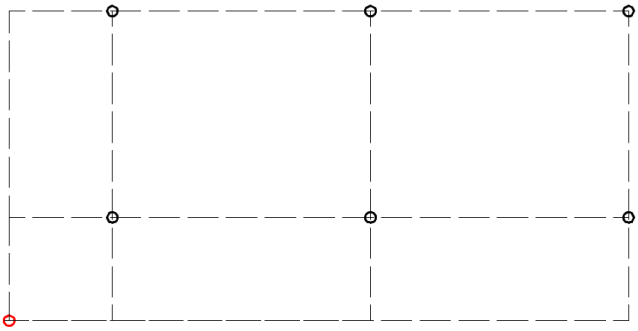


图 4.39 终点和起点的连接网络

共根据以上描述，我们可以使用集合论的方法来计算路径计算方法，我们将这 n 条路径由$A_1, A_2, \cdots, A_n$表示，单条路径的长度为$|A_i|$，任意两条不同路径的重合路径长度可由$|A_i \cap A_j|$表示，任意三条不同路径的重合路径长度可由

$\left|A_i \cap A_j \cap A_k\right|$表示…… 因此 n 条路径的长度可通过以下容斥原理的公式进行计算[92]：

$$\left|A_1 \cup A_2 \cup \cdots \cup A_n\right| = \sum_{1\le i\le n}\left|A_i\right| - \sum_{1\le i<j\le n}\left|A_i \cap A_j\right| + \sum_{1\le i<j<k\le n}\left|A_i \cap A_j \cap A_k\right| \cdots + (-1)^{n-1}\left|A_1 \cap A_2 \cap \cdots \cap A_n\right| \quad (4.19)$$

一般地，当 n=3 时候，有如下公式，其韦恩图(Venn Diagram)如图 4.40 所示。

$$\left|A_1 \cup A_2 \cup A_3\right| = \left|A_1\right| + \left|A_2\right| + \left|A_3\right| - \left|A_1 \cap A_2\right| - \left|A_1 \cap A_3\right| - \left|A_2 \cap A_3\right| + \left|A_1 \cap A_2 \cap A_3\right| \quad (4.20)$$

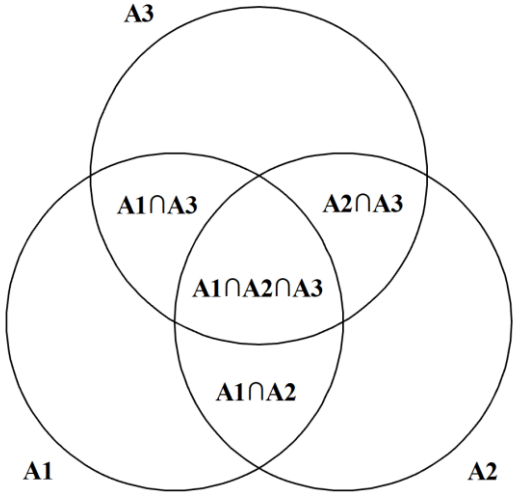


图 4.40 n=3 时的容斥原理

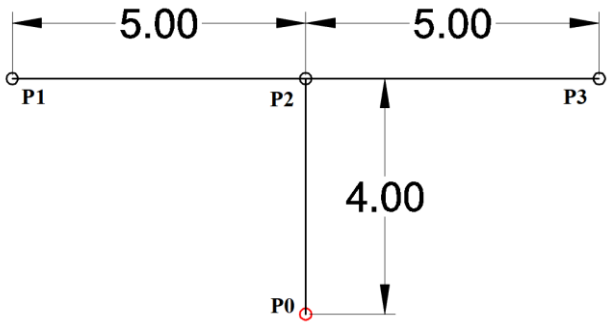


图 4.41 n=3 时的示例

对于如图 4.41 所示的情况，A_1, A_2, A_3分别表示点P_1，P_2，P_3到点P_0的距离，则有：

$$|A_1| = dist_{\text{manhattan}}(P_1, P_0) = 9\text{，}\quad |A_2| = dist_{\text{manhattan}}(P_2, P_0) = 4\text{，}$$

$$|A_3| = dist_{\text{manhattan}}(P_3, P_0) = 9\text{，}\quad |A_1 \cap A_2| = dist_{\text{manhattan}}(P_2, P_0) = 4\text{，}$$

$$|A_1 \cap A_3| = dist_{\text{manhattan}}(P_2, P_0) = 4\text{，}\quad |A_2 \cap A_3| = dist_{\text{manhattan}}(P_2, P_0) = 4\text{，}$$

$$|A_1 \cap A_2 \cap A_3| = dist_{\text{manhattan}}(P_2, P_0) = 4$$

$$|A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| + |A_1 \cap A_2 \cap A_3|$$
$$= 9+4+9-4-4-4+4=14$$

最终的计算结果为 14，符合三条路径的实际长度。遍历图 4.42 中所有2^n种路径组合，得到最短路径如图所示，该路径的长度为 30 m。

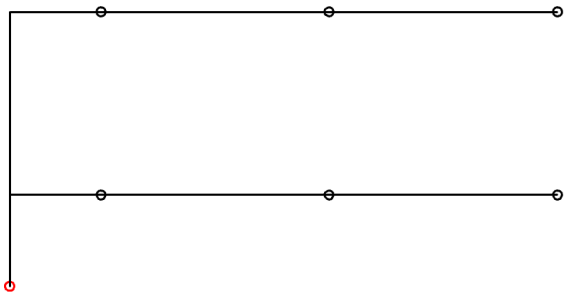
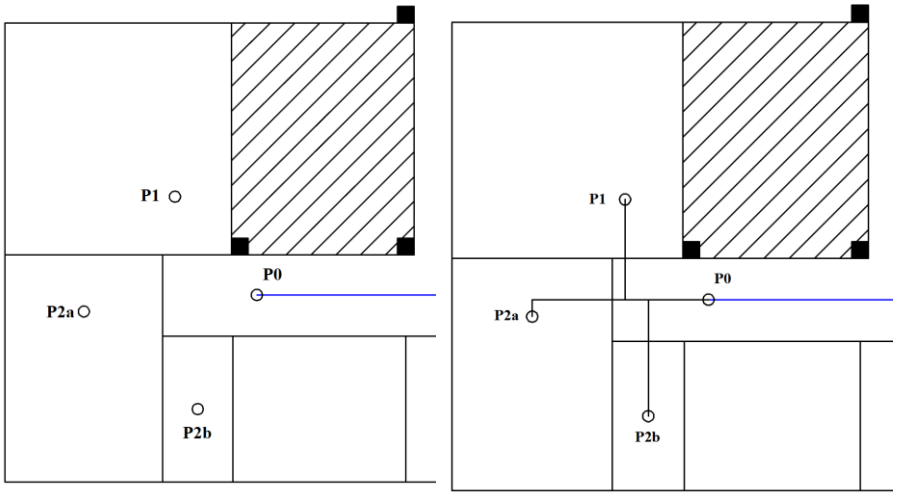


图 4.42 算法生成的结果

本算法适用于走廊末端或拐弯处与其邻近的设备的连接关系，例如，多个位于不同空间的末端设备连接点与走廊主管上的最近的点都是同一个点的情况，如图 4.43 a) 所示，此时需要将位于三个区域内的点 P1，P2a 和 P2b 依次连接至走廊主管与其最近的点 P0，通过上述算法生成的使得连接路径最短的方法如图 4.43 b) 所示。



a) b)

图 4.43 走廊末端与相邻房间设备连接

4.3.6 最远输配点约束

对于一个给定的区域，其长和宽分别 w 和 h，我们可以通过给定管路通过输配起点到输配终点可以行进的最远路程来限制最后的求解：

$$l_i \le a \times (w \times h) \tag{4.21}$$

式中，l_i 为路径方案中单条路径的长度，a 为系数，用于控制生成的图中从输配起点到各个输配节点最长的输配距离。

4.3.7 特殊构建节点识别算法及惩罚

完成了最小生成树的遍历和路径求和，还需要对管路中的特殊构建节点进行判别。对于一个有 $\varepsilon(G)$ 个节点的生成树 G，需要遍历其所有节点以辨别其属于哪类节点，以下分别介绍弯头、三通和四通构建节点的识别方法及惩罚方式：

1) 弯头

对于生成树上的一个节点 n，其坐标为 (x_0, y_0)，遍历其他所有节点，找到与其通过边与其相连的节点，若相连节点个数为 2，坐标分别记为 (x_1, y_1) 和 (x_2, y_2)，若三个点的坐标满足以下条件，则该节点为弯头节点：

$$(x_1 - x_0)(x_2 - x_0) + (y_1 - y_0)(y_2 - y_0) = 0 \tag{4.22}$$

对于弯头，我们通过弯头的个数对总路径长度进行惩罚，惩罚方式如下：

$$\Delta l_{turn} = \frac{n_{turn}}{n_{node}} \times l_{turn} \tag{4.23}$$

式中，Δl_{turn} 为因存在弯头而增加的路径惩罚长度；n_{turn} 为路径中弯头的个数；n_{node} 为路径中节点的个数；l_{turn} 为单位惩罚长度，可自定义。

2) 三通

对于生成树上的一个节点 n，遍历其他所有节点，找到与其通过边与其相连的节点，若相连节点个数为 3，则该节点为三通节点。

对于三通，我们通过三通的个数对总路径长度进行惩罚，惩罚方式如下：

$$\Delta l_{triplet} = \frac{n_{triplet}}{n_{node}} \times l_{triplet}$$

式中，$\Delta l_{triplet}$ 为因存在三通而增加的路径惩罚长度；$n_{triplet}$ 为路径中弯头的个数；$l_{triplet}$ 为单位惩罚长度，可自定义。

3) 四通

对于生成树上的一个节点 n，遍历其他所有节点，找到与其通过边与其相连的节点，若相连节点个数为 3，则该节点为四通节点。

对于四通，我们通过四通的个数对总路径长度进行惩罚，惩罚方式如下：

$$\Delta l_{quadlet} = \frac{n_{quadlet}}{n_{node}} \times l_{quadlet} \tag{4.24}$$

式中，$\Delta l_{quadlet}$ 为因存在三通而增加的路径惩罚长度；$n_{quadlet}$ 为路径中弯头的个数；$l_{quadlet}$ 为单位惩罚长度，可自定义。

完成以上的惩罚项目之后，我们得到最终的结果路径计算公式：

$$L = l_i + \Delta l_{turn} + \Delta l_{triplet} + \Delta l_{quadlet} \tag{4.25}$$

4.3.8 风管布置

采用第 3 章中风口布置的算法对本章节所使用的建筑的平面图进行风口布置的计算，得到如图 4.44 所示的结果，其中房间旁边的数字为其编号，根据凹多边形算法被切割成的房间或区域后面由 a, b, c…表示，例如，房间 9 根据凹多边形的分割算法而被分割为 9a，9b，随后 9b 因为房间内障碍物的规避算法继续被分割成 9b’和 9b’’，阴影的区域为布管道时禁止穿越的区域，黑色矩形标记处为建筑中柱子的位置，蓝色实线标记的为 4.2 节中生成的走廊主管。接下来根据房间空调系统的选择可在房间内布置不同的设备类型，如果选择风机盘管+新风

的方式，那么可根据房间内现有的风口位置布置风机盘管以及对应的新风风管以及与主管的连接方式；如果采用全空气系统的方式，则只需要考虑将房间内的风口通过风管与走廊中的主管相连。

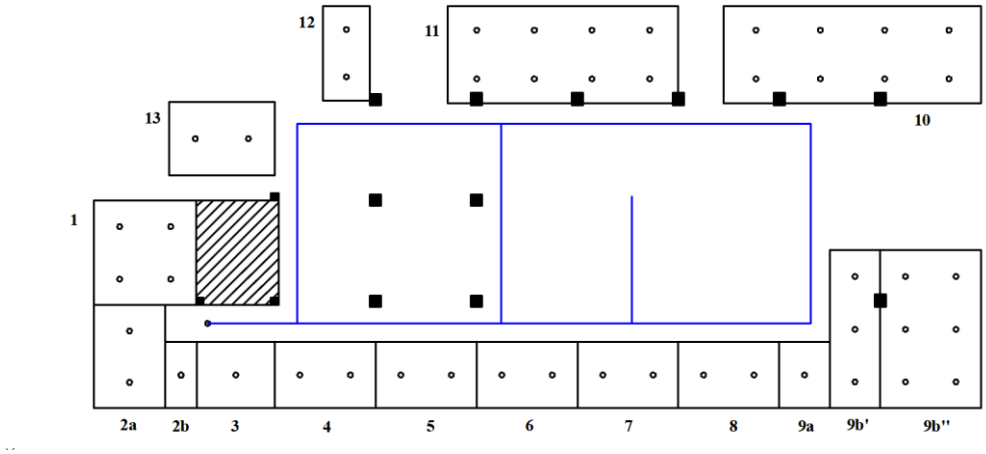


图 4.44 风口布置结果

然后根据本节中的路径算法，最终得到如下的连接结果

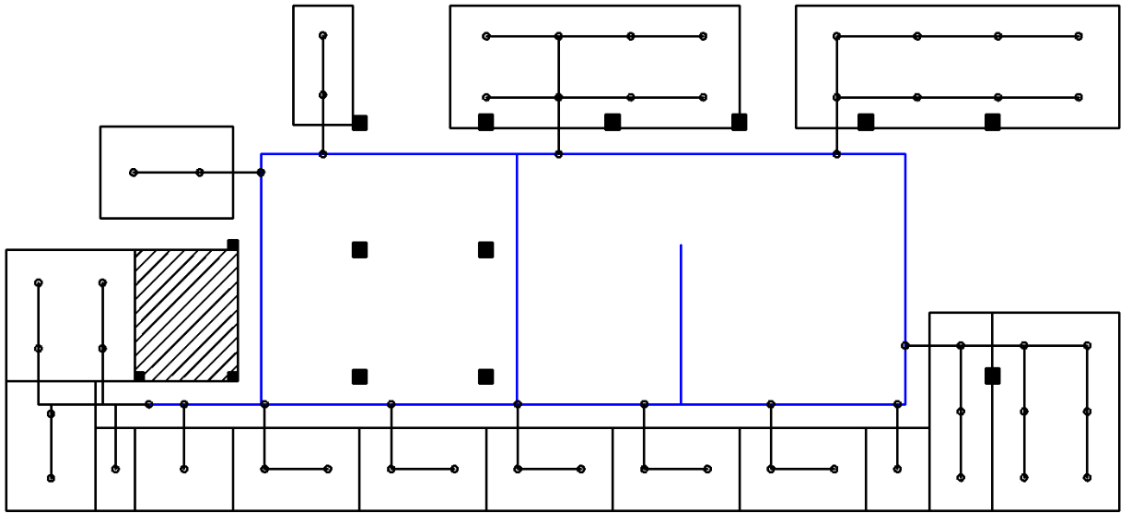


图 4.45 平面内风管连接结果

4.4 主管修剪算法

4.3 节中生成的连接管路（图 4.45）是通过区域内的节点与走廊主管连接形成的，仍为一个闭合的环路，我们需要通过算法将其修剪为最短的生成树。首先我们需要将现有的图同输配起点（机房）节点相连接，如下图所示：

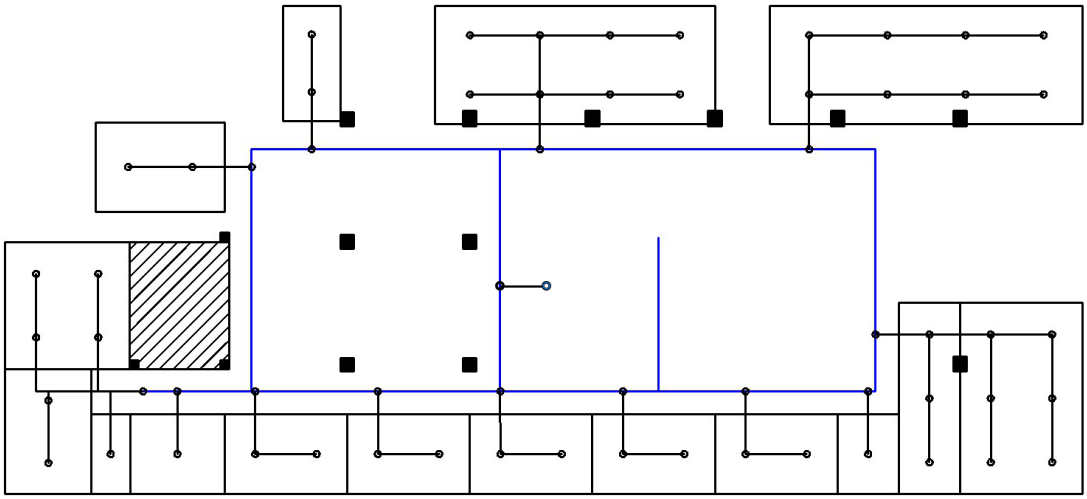


图 4.46 平面图与机房节点连接

如图 4.46 所示，根据我们对树的要求，目前图中存在两条多余的边，因此需要通过判断删除两条多余的边，为此我们采用如下的修剪算法：

1) 将图中所有环包含的边加入待修剪边的列表，环路可通过 4.1.4 中的深度优先搜索算法得到，图 4.46 所示存在两个环路，其所有环路包含的边如图 4.47 所示；
2) 对待修剪边列表中的边进行组合，每次取出环路个数的边，将原存在环路的平面连接图中对应的边进行修剪，并通过 4.1.6 中的单源最短路径算法求得当前从输配起点至所有末端节点的路径之和

步骤 2）中路径之和最短的情况下即为最后的修剪结果，如图 4.48 所示。

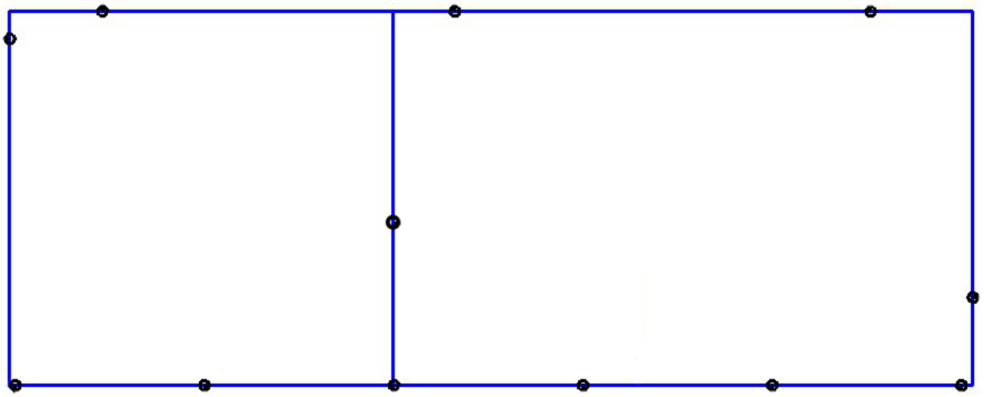


图 4.47 待修剪边列表包含的边

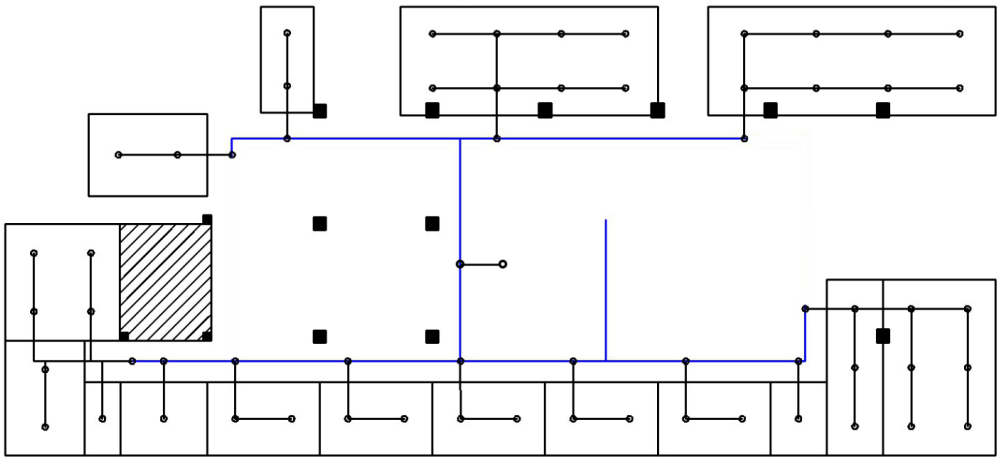


图 4.48 修剪完后的平面连接图

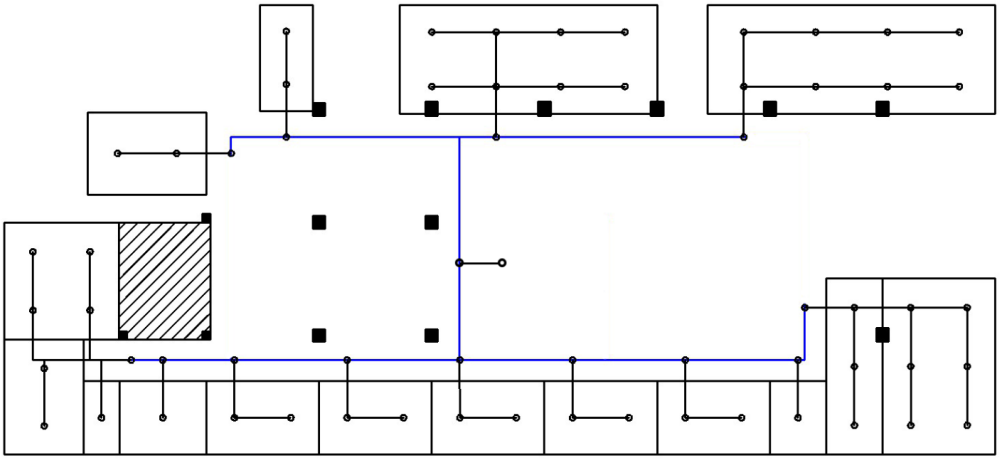


图 4.49 最终修剪结果

经过以上的环路修剪之后，还需要对原走廊中保留下来的孤立的边进行修剪，这些边不参与最后的输配，判断的依据为：边属于走廊，且的其中一个节点不与末端节点相连接，图 4.48 存在一条孤立的边，修剪完最后的结果如图 4.49 所示。

4.5 本章小结

本章主要讲述了平面连接图的生成过程，首先通过走廊的平面图生成走廊的主管，然后通过一些基础的图论算法的平面最短路径算法生成末端与主管的连接，核心的内容是最小生成树的遍历，最后对主管的修剪得到最后的平面连接结果。

第 5 章 管路障碍规避算法

在第 4 章中我们在图的生成过程中提到两个节点之间的距离可能因为存在障碍物而与其坐标的曼哈顿距离不同，在分区设备与主管的连接中，可能出现管线过于靠近障碍物甚至穿过障碍物的情况，为了解决这种情况，我们提出了线惩罚和点惩罚两种障碍惩罚方式，并介绍了如何通过基于 A*路径算法求得两个节点之间最少拐点的连接路径。

5.1 障碍惩罚算法

5.1.1 线惩罚

这种惩罚方式可用于线障碍物的规避以及管线穿墙的惩罚。例如，对于如图 5.1 所示的情况，两个节点可能以 a 路径或者 b 路径相连，这种拐点个数小于等于 1 的路径我们称为为曼哈顿路径，但是 b 路径穿过了阴影处标记的禁止穿越的区域，我们需要通过算法对这种路径进行“惩罚”，算法如下：

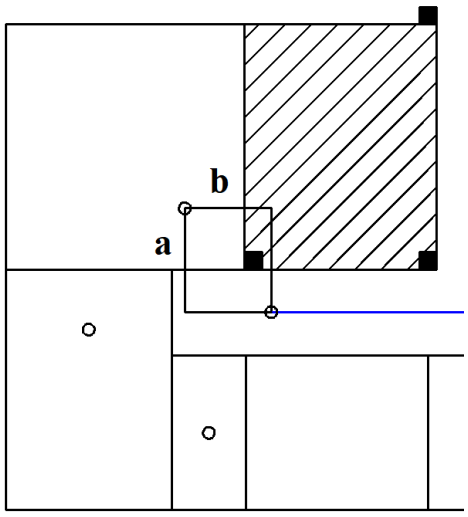


图 5.1 障碍区域规避

对于一个闭合的障碍区域，其外部轮廓可由一系列函数确定：$f_1, f_2, \ldots$，对于一个矩形区域，由四个函数分别表示其轮廓的四条线段。对于一条路径，其由多段线组成，同样可以表达成一系列函数：$g_1, g_2, \ldots$，可通过二维空间中两曲线相交的判断算法判断 $f_1, f_2, \ldots$ 是否与 $g_1, g_2, \ldots$ 相交。若相交，通过以下方式惩罚：

$$\Delta l_{ba,line}=+\infty \tag{5.1}$$

$$\Delta l_{ba,line}=10e5\times n_{node} \tag{5.2}$$

式中，$\Delta l_{ba,line}$ 为路径穿越障碍区域的惩罚，设为正无穷，实际算法中可设为一个较大值，如节点个数的 10e5 倍。

此外，线惩罚还可以用在路线穿墙的惩罚，如图 5.2 所示，该区域内的四条路径都穿过该区域的边界，即建筑的内墙，判断路径经过区域的边界的方法与判断穿越障碍区域的方法相同，此时可通过以下方式给与惩罚：

$$\Delta l_{ba,line}=l_{wall} \tag{5.3}$$

式中，l_{wall} 为设定的穿过墙面的线段的惩罚值。

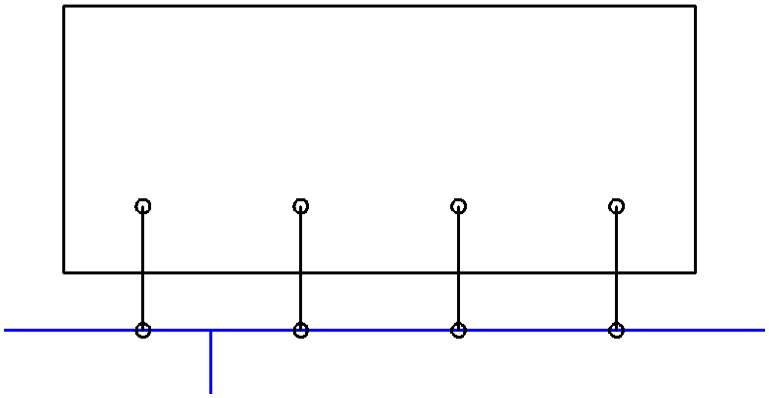


图 5.2 路径穿墙时的情况

5.1.2 点惩罚

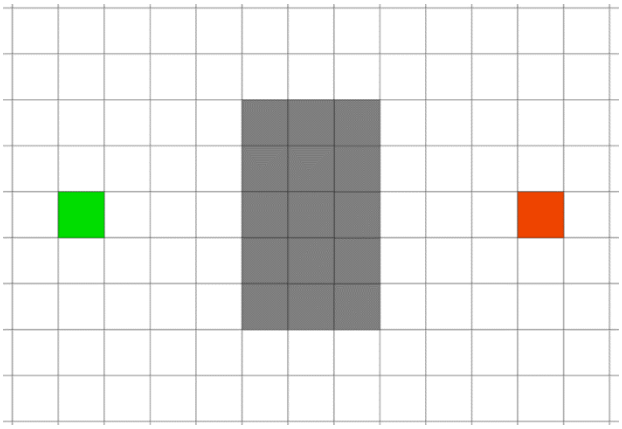
这种惩罚方式可用于点障碍物的规避，例如平面内的立柱，以及当两个节点之间的两种曼哈顿路径中均存在障碍物的情况，当障碍物的中心与平面内的某条路径的最近距离低于设定阈值时，通过以下方式惩罚对该路径给与惩罚：

$$\Delta l_{ba,point}=l_{routeplan} \tag{5.4}$$

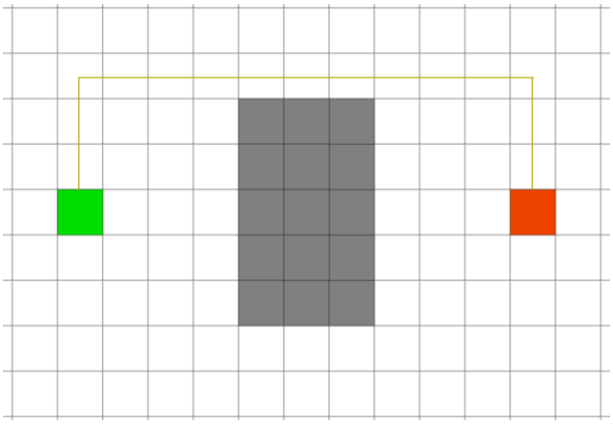
式中，$\Delta l_{ba,point}$ 为路径穿越障碍区域的惩罚，$l_{routeplan}$ 为点惩罚的值，大小需要通过 5.2 中的路径规划算法求得。

5.2 路径规划算法

5.1 中我们给出了点惩罚的方式，本节将讲述如何通过路径规划算法进行计算。现考虑如下问题：现在需要从如下的图中寻找到一条能够从起点到终点的路径，如图 5.3 所示，图中深绿色节点为路径起点，红色节点为路径终点，灰色区域为不可经过的区域，白色区域为可经过的区域，要通过算法实现从绿色节点走到红色节点，每一次只能从一个节点走到其前后左右四个节点，并同时满足经过的路径最短。



a)



b)

图 5.3 路径规划问题

5.2.1 A*路径搜索算法

A*算法(A Star Algorithm)是一个启发式搜索算法，可以通过对可穷举空间内的节点进行启发式搜索，实现最短路径的生成[93]。

A*算法的评价函数如下：

$$F_{i,j} = G_{i,j} + H_{i,j} \tag{5.5}$$

其中$G_{i,j}$表示从开始节点走到当前节点所花费的代价；$H_{i,j}$表示从当前节点走到目标节点所预估的代价，是一个启发式搜索函数，这两个代价应该用同一种计算方法。如图 5.4 所示，每个节点都可以用其坐标(i,j)表示，从绿色起点出发，其坐标为(1, 4)，将其加入开放列表(Open List)中，此后的搜索都将从开放列表中选取节点进行，目前开放列表中仅存在起始节点，因此选其作为搜索节点，有上下左右四种方向可行进，每个节点中，左上角的值为$G_{i,j}$，右上角的值为$H_{i,j}$（这里用曼哈顿距离计算，下面会详细介绍选用曼哈顿距离作为启发式搜索函数的原因），下面的值为$F_{i,j}$，将四个节点加入到开放列表中，在图中用浅蓝色表示。完成对起始节点的检索后，将起始节点加入封闭列表中(Closed List)，在图中用深绿色表示，封闭列表的含义是其包含的节点已经完成搜索，当后续的搜索节点搜索到障碍物节点或者封闭列表中的节点时，不会将其加入开放列表中进行后续的搜索。

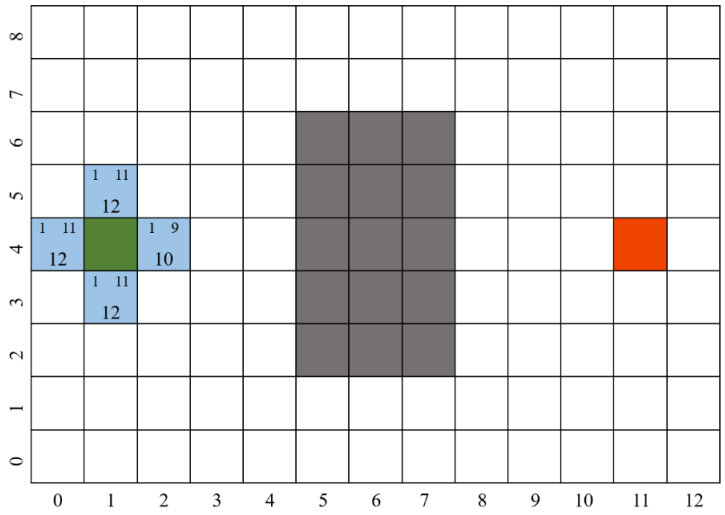


图 5.4 启发式搜索示例

然后从当前封闭列表中取出$F_{i,j}$值最小的节点，重复上述步骤，如图 5.5 所示。此时开放列表中所有节点的$F_{i,j}$值都为 12，选取$H_{i,j}$值最小的点，即坐标为(4, 5)和(4, 3)的节点，可随机选择一个重复进行上述搜索步骤，如节点(4, 5)，执行完该节点后，(4, 3)的节点的$H_{i,j}$值最小，为 8，此时图中$H_{i,j}$值为 8 的节点均完成搜索，因此继续对满足$F_{i,j}$值最小的节点（若$F_{i,j}$相同则选取$H_{i,j}$较小的优先搜索）进行搜索，最终$F_{i,j}$值为 12 的节点均完成搜索，如图 5.13 所示，此时刚好检索完障碍物的一个边界，满足最小节点要求的节点的坐标为(4, 6)和(4, 2)的节点，继续完终点方向进行搜索，最终得到如图 5.7 所示的结果。

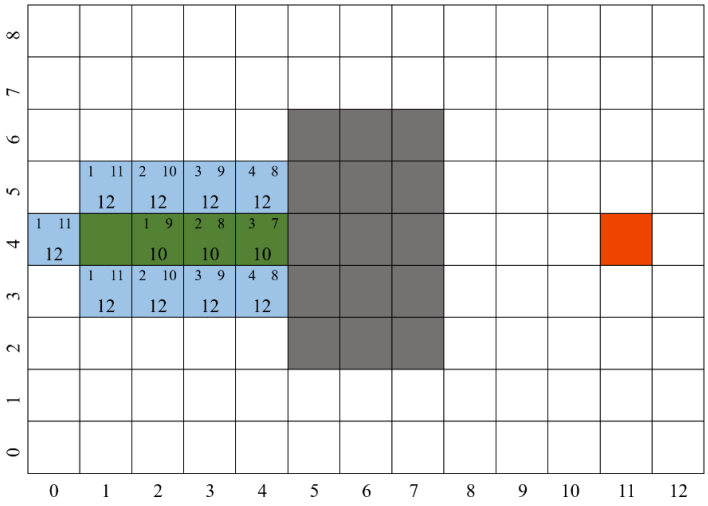


图 5.5 启发式搜索示例

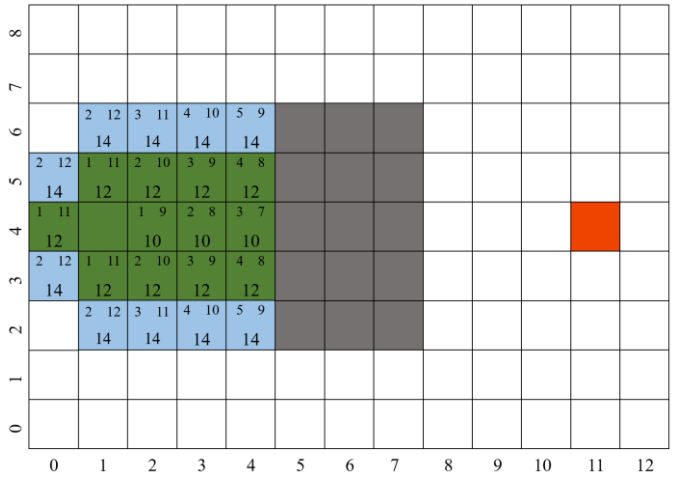


图 5.6 启发式搜索示例

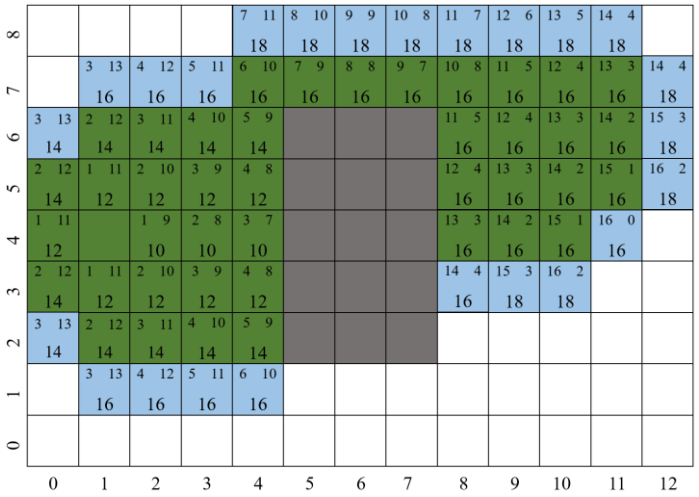


图 5.7 启发式搜索示例

5.2.2 对比其他路径搜索算法

A*算法较其他路径搜索方法具有

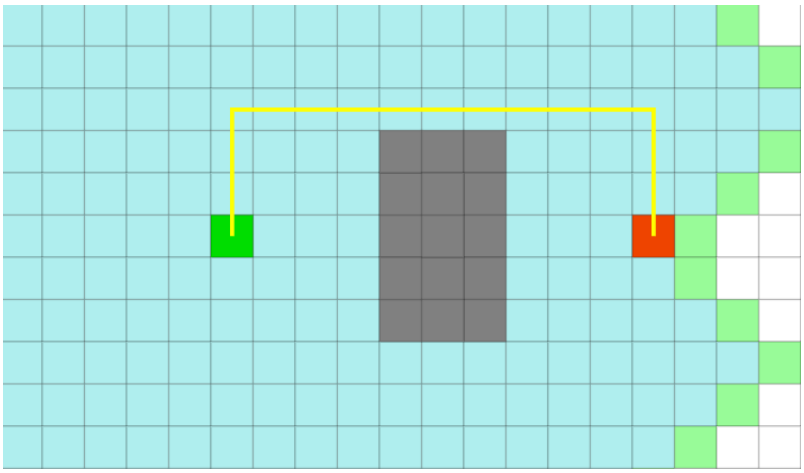


图 5.8 BFS 搜索算法

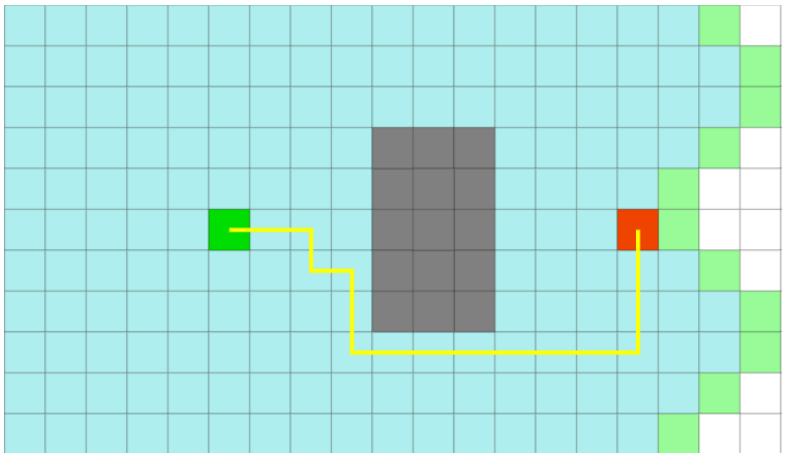


图 5.9 Dijkstra 搜索算法

5.2.3 启发式搜索函数的确定

为了更快地向目标点靠近，需要通过启发式搜索函数来确定哪个点最有希望能够快速地到达目标点，常用的启发式搜索函数有曼哈顿距离、欧几里得距离和 Octile 距离，值得注意的是，在选择不同启发式搜索函数时，需要将 $G_{i,j}$ 更改为相应的距离函数。

1） 曼哈顿距离

曼哈顿距离的计算思路来源于计算从一个街区穿越到另一个街区的最短路径，一般来说对于当前节点只可进行上下左右四个方向上的移动，适用于网格地图，点 (x_1, y_1) 和点 (x_2, y_2) 的曼哈顿距离计算公式如下式所示：

$$H_{manhattan}((x_1, y_1),(x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2| \tag{5.6}$$

2） 欧几里得距离

欧几里得距离的计算方法是二维平面最常用也最为直观的计算方法，通过计算当前节点与目标节点的直线距离来得到启发搜索的函数值，适用于导航网格（在当前节点可沿着上下左右和四个角八个方向进行移动），点 (x_1, y_1) 和点 (x_2, y_2) 的曼哈顿距离计算公式如下式所示：

$$H_{euclidean}((x_1, y_1),(x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{5.7}$$

3） Octile 距离[94, 95]

通过欧几里得距离的方法进行计算是十分直观的，但是每次都需要经过开方运算，对于较大的导航网格计算量会显著增加，因此提出了 Octile 距离的计算方法，该距离计算方法的核心思想是限制了从一个点到另一个点只能沿着水平或竖直方向以及 45° 方向上的行进，点 (x_1, y_1) 和点 (x_2, y_2) 的曼哈顿距离计算公式如下式所示：

$$H_{octile}((x_1, y_1),(x_2, y_2)) = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{5.8}$$

其与曼哈顿距离和欧几里得距离的对比如图 5.10 所示，可以通过数学证明得到，对于二维平面上的任意两个点 (x_1, y_1) 和 (x_2, y_2)，三个距离计算方法的大小关系为：

$$H_{euclidean}((x_1, y_1),(x_2, y_2)) \le H_{octile}((x_1, y_1),(x_2, y_2)) \le H_{manhattan}((x_1, y_1),(x_2, y_2)) \tag{5.9}$$

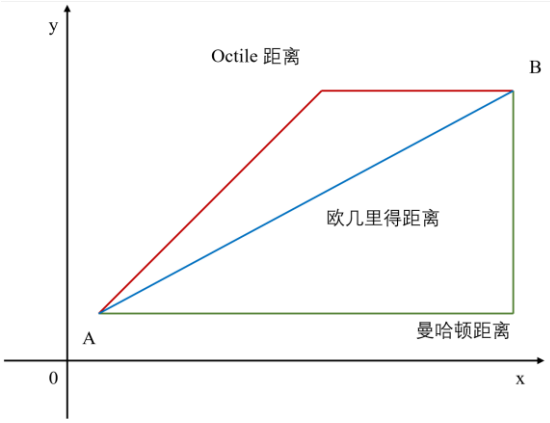


图 5.10 三种启发式搜索函数比较

对于采用不同的应用场景，采用不同的搜索函数会对运算速度产生一定的影响，这里考虑几个布管中可能存在的类似的场景进行模拟，图 5.11~图 5.13 显示了当起点和终点之间的直线连接存在障碍物时的路径（不允许一个点直接与其对角的点直接相连）搜索结果，图中深绿色节点为路径起点，红色节点为路径终点，浅蓝色节点为检索完成后的封闭列表中的节点，浅绿色节点为检索完成后的开放列表中的节点，黄色的折线为使用 A*算法生成的路径，使用曼哈顿搜索的封闭列表中节点个数为 59，欧几里得距离搜索是封闭列表中节点个数为 90，Octile 搜索的封闭列表中节点个数为 91，可以看到采用曼哈顿距离作为启发式搜索函数的检索范围较其他两者小；图 5.14~图 5.16 显示了当起点和终点之间的直线需要通过折线连接（不允许一个点直接与其对角的点直接相连）时的路径搜索结果，类似地，采用曼哈顿距离作为启发式搜索函数的检索范围较其他两者的搜索范围更小。

基于以上两个路径搜索场景，再考虑到管道的走线类似于街区的穿越，因此本文采用曼哈顿距离作为启发式搜索函数。

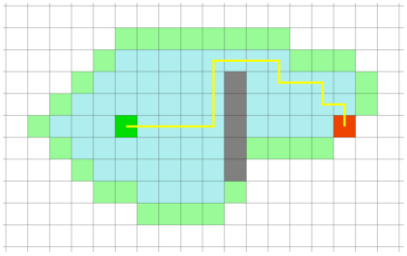


图 5.11 直线连接有障碍物时用曼哈顿距离搜索结果

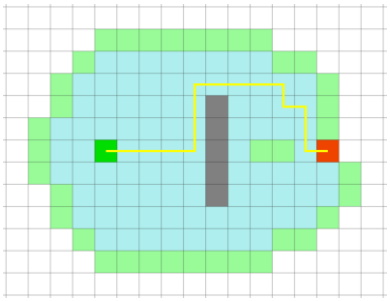


图 5.12 直线连接有障碍物时用欧几里得距离搜索结果

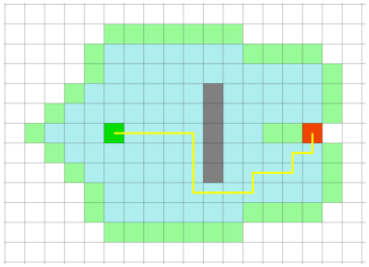


图 5.13 直线连接有障碍物时用 Octile 距离搜索结果

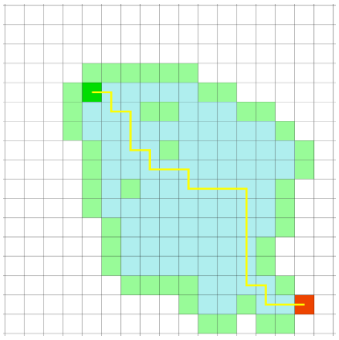


图 5.14 对角连接时用曼哈顿距离搜索结果

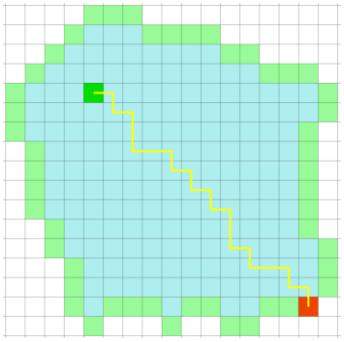


图 5.15 对角连接时用欧几里得距离搜索结果

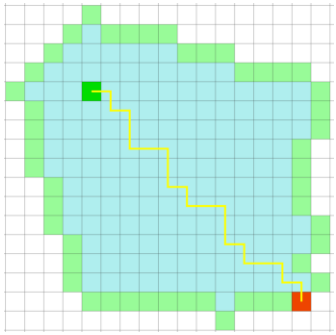


图 5.16 对角连接时用 Octile 距离搜索结果

5.3 路径生成算法

5.3.1 路径生成

通过 A*算法完成启发式路径搜索算法后，可根据最后的搜索结果生成最短的路径，生成方法如下：

从终点开始生成路径，此时终点的 $F_{i,j}=m$，检索其四个方向上（5.2.1 中路径搜索的方向）满足 $F_{i,j}=m-1$ 的节点(i,j)，例如，图 5.7 的终点(11，4)的 $F_{11,4}=16$，其上方和左方的节点(10，4)和(11，5)满足 $F_{i,j}=16-1=15$ 的要求，可随机将任何一个节点添加至路径中。随后依次检索 $F_{i,j}=m-1,m-2,...,1$，直至检索到路径起点为止，其中一个路径的结果如图 5.17 所示。

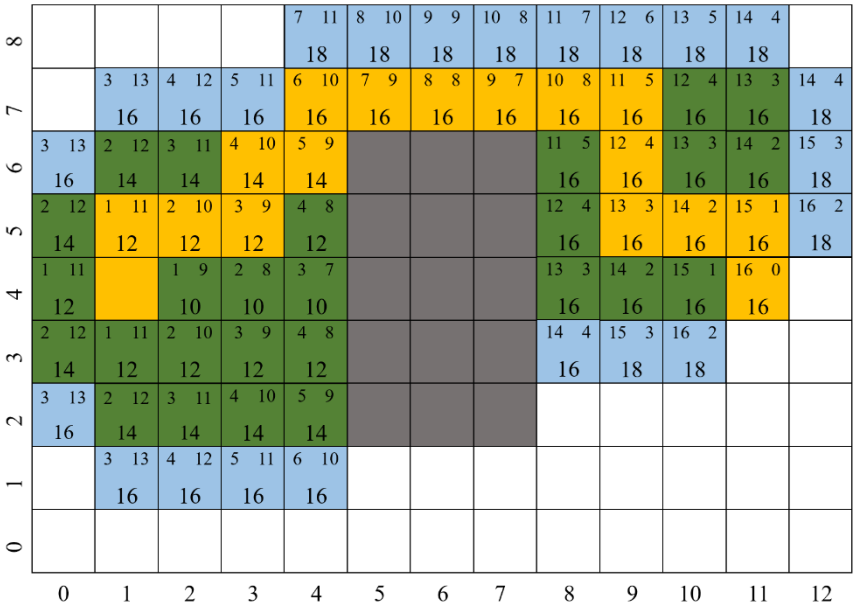


图 5.17 其中一条路径的生成

经过以上的内容我们实现了一个完整的 A*最短路径生成算法，可实现从一个给定的起点经过一定的障碍物达到终点。该算法具有快速高效的优点，较其他算法如 Dijkstra 算法和 BFS（广度优先搜索）搜索算法能用较小的搜索范围实现最短路径的搜索，但是其也存在内存占用量较其他算法高的问题。

A*算法还具有如下性质[96, 97]：对于图，如果从起始节点 S 到终点节点 E 之间存在路径，那么 A*算法一定可以找到最短的路径。

1) 在满足条件的路径中的所有节点，必定满足以下条件：

$$F_{i,j} \le m \tag{5.10}$$

其中 m 为终点的 $F_{i,j}$ 值，即最短路径从起始节点到终点经过的路径长度，因为采用了启发式搜索函数，因此若某个节点的 $F_{i,j}>m$，则通过该节点的路径的长度必定大于 m。

该算法不仅可以实现单个障碍物的规避，还可以实现多个复杂障碍物的规避，如图 5.18 所示。

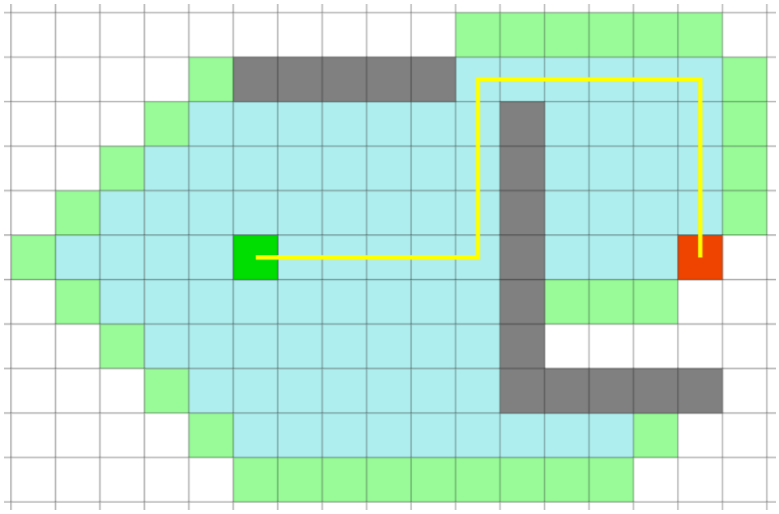


图 5.18 其中一条路径的生成

5.3.2 所有最短路径遍历算法

图 5.17 所示的路径显然不是解决管路连接问题中最佳的结果，同 4.3.2 一样，我们的目标不是找到一条最短的路径，而是要从所有符合最短路径要求的路径中满足最符合设计要求的一条路径，在这里我们定义从起点到终点经过的拐点个数最少的节点即为符合要求的路径。

如图 5.17 所示，根据原始的 A*算法我们只能找到一条从起始节点通往终点的路径，根据该问题的对称性，应该存在对应的一条从下方绕过障碍物到达终点的路径，但是原始算法并没有对(4, 1)之后的节点进行搜索，这是因为 A*算法的退出条件的是搜索到终点节点为止。因此需要修改 A*算法的退出条件，通过观察发现结束搜索时开放列表中仍有许多节点的 $F_{i,j}=m$，根据 5.3 中提到的 A*算法的性质，满足最短路径长度为 m 的路径可能会经过这些节点，因此只需要继续对开放列表中的节点进行遍历直到其中所有节点 $F_{i,j}>m$ 为止即可，对于图 5.17，新的退出条件得到的搜索结果如图 5.19 所示。

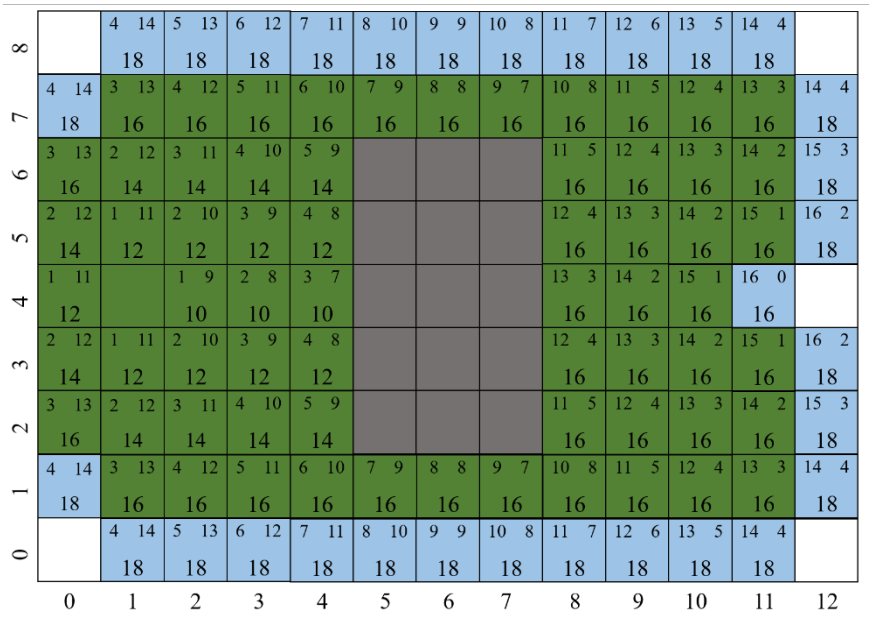


图 5.19 更改 A*算法退出条件后的结果

根据以上的推导过程，有了图 5.19 之后我们可以从图中的封闭列表中获取从终点到起点的所有可能最短路径。同样从终点开始生成路径，此时终点的 $F_{i,j}=m=16$，其四个方向上有 3 个节点(11，5)，(10，4)和(11，3)满足 $F_{i,j}=m-1=15$的要求，与上一次生成路径的方式不同，这一次我们需要遍历所有可能从终点到达起点的路径，因此我们需要考虑这 3 个节点及其连通的所有可能路径，同时，对于(11，5)节点，有(11，6)和(10，5)共 2 个节点满足 $F_{i,j}=m-2=14$ 的要求，对于(10，4)节点，有(10，3)，(9，4)和(10，5)共 3 个节点满足 $F_{i,j}=m-2=14$ 的要求，对于(11，5)节点，有(11，6)和(10，5)两个节点满足 $F_{i,j}=m-2=14$ 的要求……直到一直访问到起始节点为止，为了更直观地表示以上过程，我们将这一遍历过程转化为树的形式，如图 5.20 所示。那么这一问题将转化为树的深度遍历问题，可采用 4.1.4 中介绍的深度优先搜索的算法实现如下的遍历过程：

路径 1：[(11，4)→(11，5)→(11，6)→(10，6)→…→S]

…

路径 j：[(11，4)→(10，4)→(9，4)→…→S]

…

路径 n：[(11，4)→(11，3)→(11，2)→…→S]

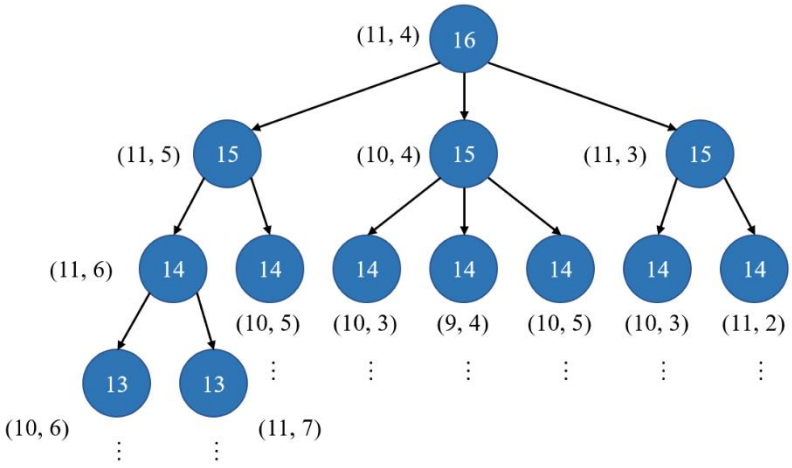


图 5.20 将图中的可能路径转化为树

5.3.3 最少拐点路径——类动态规划优化求解

在实现了所有最短路径的遍历之后，我们还需要考虑其中某一条或者某几条满足我们要求的路径，即从起点到终点经过的拐点个数最少的路径，对于拐点的识别我们可以采用以下方法：

如图 5.20 所示，我们把终点节点当作树的根节点，对于一个节点$(i，j)$，其父节点为(i_{pre}, j_{pre})，子节点为(i_{next}, j_{next})，如果节点(i,j)处形成一个 90° 的拐点，则有：

$$(i_{pre}-i)(i_{next}-i)+(j_{pre}-j)(j_{next}-j)=0 \tag{5.11}$$

例如，节点(11，5)与其父节点(11，4)节点和子节点(10，5)之间形成拐点：

$$(11-11)(10-11)+(4-5)(5-5)=0 \tag{5.12}$$

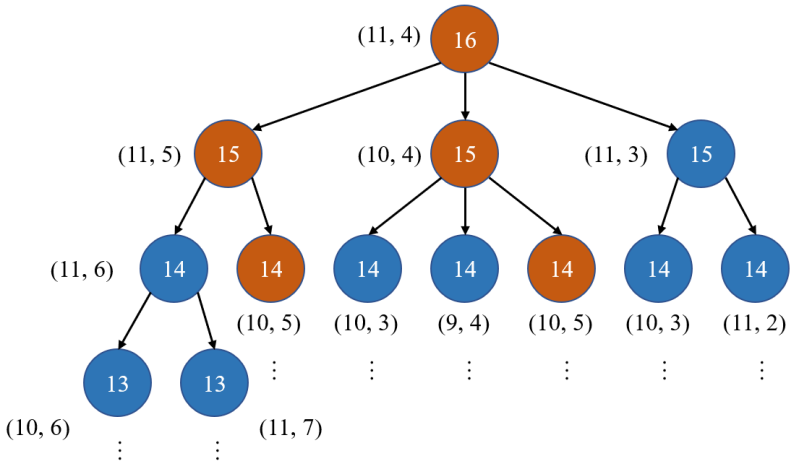
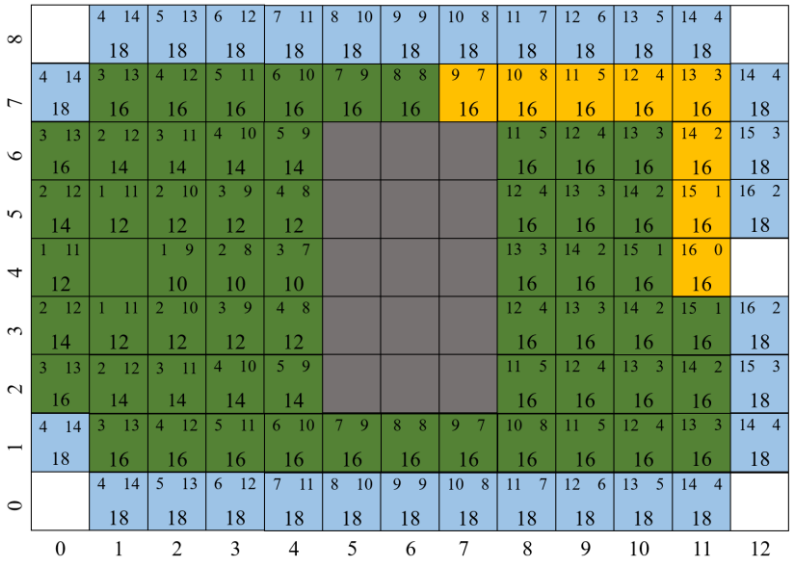
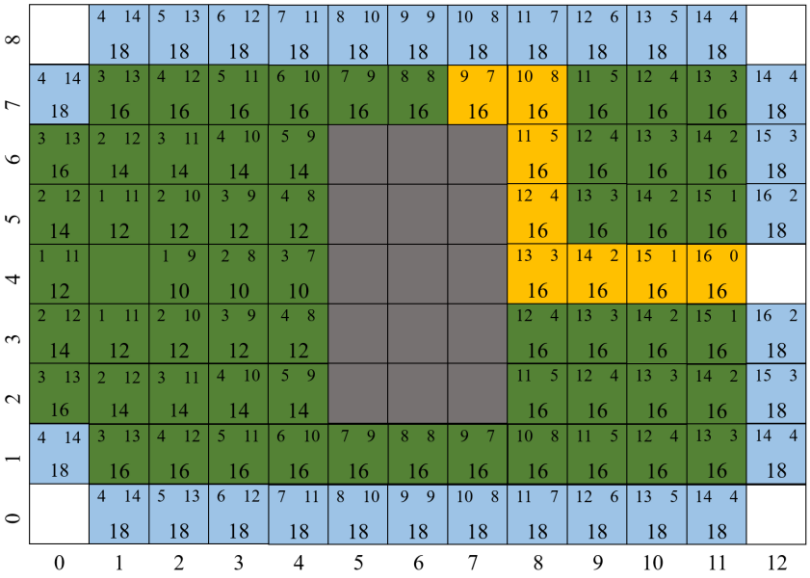


图 5.21 两条路径经过同一节点的情况

对于该问题进行遍历，得到 800 种不同的路径，对于每条路径，还要进行拐点个数的计算，随着网格的节点增加将会大幅增加算法的运算量，因此需要考虑对算法进行优化。通过观察注意到在路径中存在不同的路径经过同一个节点的情况，例如图 5.20 中的路径[(11，4)→(11，5)→(10，5)→…→*S*]和路径[(11，4)→(10，4)→(10，5)→…→*S*]均会经过节点(10，5)，此时从终点(11，4)到达节点(10，5)的路径有 2 条（暂不考虑从该节点到起点的路径可能数量），随着离终点越来越远，到达同一个节点的可能性将越来越多最终这些路径汇集到一个终点，使得到达终点的路径数量很大。例如从终点(11，4)到达节点(7，7)的路径增长到了 20 条，其中两条如图 5.22 所示。其中路径[(11，4)→(11，5)→(11，6)→(11，7)→(10，7)→(9，7)→(8，7)→(7，7)]是拐点最少的路径，如图 5.22 a)所示，拐点个数为 1，路径[(11，4)→(10，4)→(9，4)→(8，4)→(8，5)→(8，6)→(8，7)→(7，7)] 的拐点个数次之，为 2，如图 5.22 a)所示，此外还有 2 条路径的拐点个数为 2，分别为路径[(11，4)→(10，4)→(9，4)→(9，5)→(9，6)→(9，7)→(8，7)→(7，7)]和路径[(11，4)→(10，4)→(10，5)→(10，6)→(10，7)→(9，6)→(8，7)→(7，7)]，其余路径的拐点个数均大于 2。



a)



b)

图 5.22 从终点到达(8，7)节点的两条路径

动态规划(Dynamic Programming, DP)同贪心法需要解决的问题类似，都是从一系列选择中选择一个或多个最优的抉择，在动态规划的过程中，我们首先要确定一个问题，即最优子序列是否包含于最优选择序列当中，如果是的话，那么这个问题可以通过建立动态规划递归方程来解决。本问题不能直接通过动态规划的方法来解决，但是可以借鉴动态规划最优子序列的思想，即在算法中对某一个子问题的最优选择做出记忆以供其他包含该子问题的问题调用避免重复计算。

继续回到本问题求最少拐点的路径，我们可以通过算法求出终点(11，4)到达节点(7，7)的路径的最少拐点个数为 1，对应路径个数为 1，次少拐点个数为 2，对应路径个数为 3，接下来我们先证明经过终点(11，4)经过节点(7，7)到达起始节点中拐点最少的路径一定出现在经过这 4 条子路径：

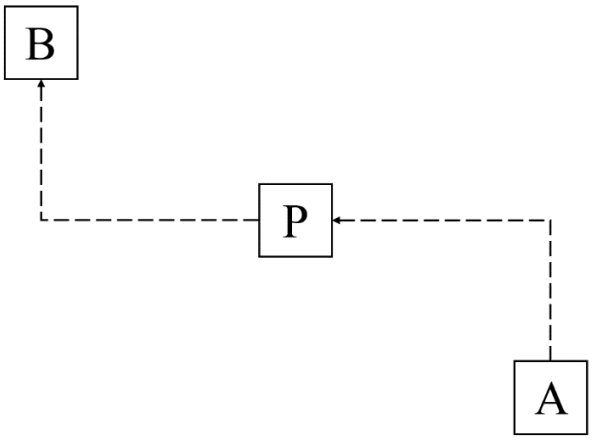


图 5.23 从终点到达(8，7)节点的两条路径

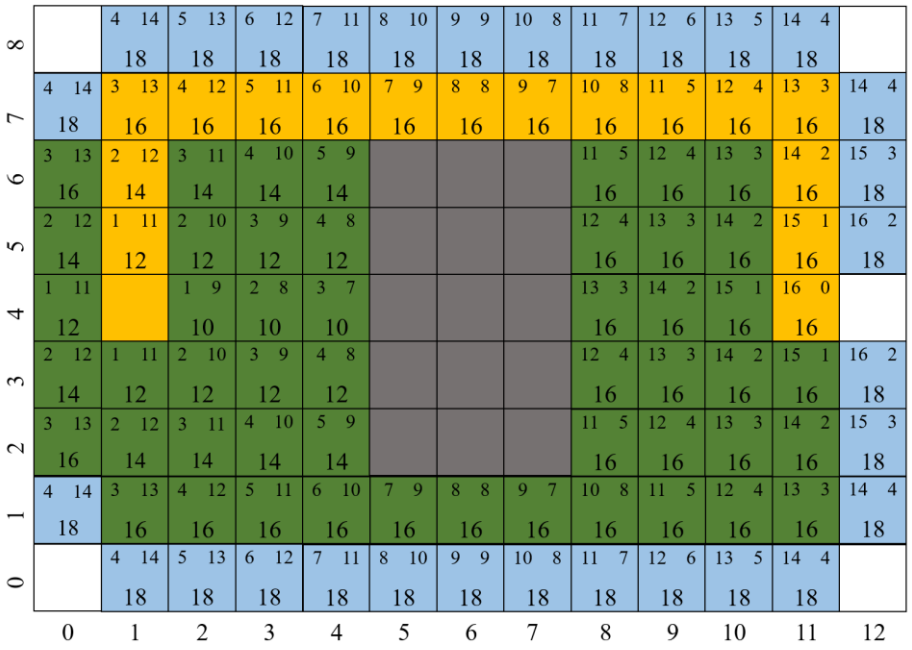
我们将本问题抽象为更为一般的形式，如图 5.23 所示的情形，假设从节点 A 到节点 P 共有 *m* 条路径：$p_{A,P,1}, p_{A,P,2}, ..., p_{A,P,m}$，这 *m* 条路径中最少的拐点个数的 *k* 个，从节点 P 到节点 B 共有 n 条路径：$p_{P,B,1}, p_{P,B,1}, ,..., p_{P,B,n}$，，这 n 条路径中最少的拐点个数的 *l* 个。在经过节点假设 P 的坐标为(*i*，*j*)，路径上其前一个节点的坐标为 (i_{pre}, j_{pre})，后一个节点的坐标为 (i_{next}, j_{next})，如果 $(i_{pre}-i)(i_{next}-i)+(j_{pre}-j)(j_{next}-j)=0$，则在节点 P 处将会增加 1 个拐点，P 处增加的拐点个数为 0，则从节点 A 到节点 B 中的总共 *mn* 条路径中最小拐点的个数为 *k*+*l*；若 $(i_{pre}-i)(i_{next}-i)+(j_{pre}-j)(j_{next}-j)\neq 0$，则 P 处增加的拐点个数为 0。则从节点 A 到节点 B 中的总共 *mn* 条路径中最小拐点的个数为 *k*+*l*+1。

经过上述分析我们得到节点 A 到节点 B 中的总共 *mn* 条路径中最小拐点的个数的最大值为 *k*+*l*+1。如果该拐点最小的路径经过从节点 A 到节点 P 中拐点个数为 *k*+1 的拐点，则该路径可能经过从节点 P 到节点 B 的路径的拐点个数至多为 *l*，即要求通过从节点 P 到节点 B 拐点个数最少的路径；如果从节点 A 到节点 P 的路径的拐点个数为 *k*+2，则该路径可能经过从节点 P 到节点 B 的路径的拐点个数至多为 *l*-1，而之前提到从节点 P 到节点 B 的最少拐点的路径也不符合要求。因此证得从节点 A 经过节点 P 到节点 B 的最少拐点的路径一定经过从节点 A 到节点 P 拐点个数为 *k* 或 *k+1* 的路径。

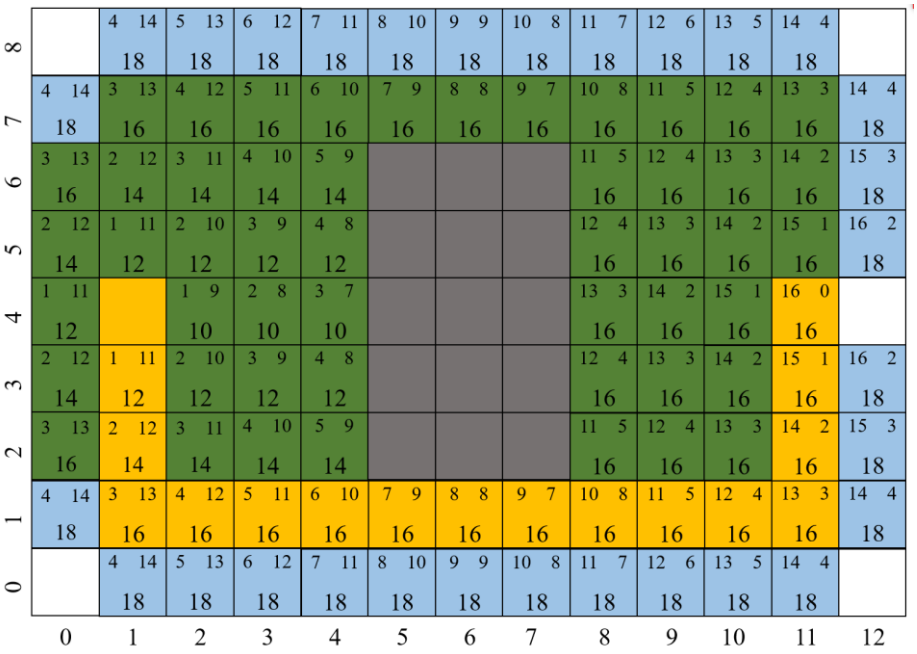
在完成了上述的证明之后，我们可以通过以下算法来实现最少拐点路径的选取：

改用 4.1.3 中提到的广度优先搜索算法，这样我们可以优先对有根树的同一层的内容进行搜索，在同一层中可能会碰到相同的节点，只需要保留根节点到该节点之间拐点个数满足 $\leq k+1$（*k* 为根节点到该节点之间的最小拐点数目），其余部分可进行剪枝(Prune)操作。这样我们就减少了大量重复性的检索工作，尤其是对于那些节点数量较多的图，可大幅减少运算量。

最后，我们得到如图 5.24 最少拐点路径图 5.24 所示的两条拐点数量最少的路径。



a)



b)

图 5.24 最少拐点路径

5.4 本章小结

本章首先通过对障碍物（柱子、墙体、梁等）设置惩罚来确保管线对障碍物的规避，为了解决这一方法对于部分障碍物情况可能无法生成管线的情况，我们提出了一种基于 A*路径搜索算法来使得路径能够规避较为复杂的障碍物信息；为了满足管路在布置的时候需要满足的最少拐点的要求，我们还采用了一种类动态规划的求解方法。

第 6 章 结论与展望

6.1 结论

本文的研究工作初步探讨了空调系统的设计流程在计算机中自动化的过程，主要包括以下四部分内容：

1) 第二章中我们建立了基于 BIM 模型的建筑负荷计算流程，其中使用的 BIM 模型类型为 gbXML 文件，通过将其包含的几何信息、结构信息、时刻表信息等通过 python 脚本的方式写入 idf 文件；
2) 第三章中我们根据平面几何处理算法提出了不同类型房间的风口布置算法，主要思路为根据房间的几何形状和障碍物位置将风口布置在多个非凸多边形当中，最后我们根据相关规范给出了基于风速约束条件的风口尺寸选择算法；
3) 第四章中首先我们通过转换完成的 idf 文件提取出走廊的几何形状，然后将其转换完单线以生成初步的走廊主管图，然后根据第三章选出来的风口位置通过图论的算法将其连接，连接过程中用到的核心算法是最小生成树的遍历；
4) 第五章中我们通过对障碍物的两种惩罚和基于 A*路径搜索算法的障碍物规避算法对第四章图的生成中节点之间距离做了完善，并给出了从一个节点穿越障碍物到另一个节点所需要最少拐点个数的解。

6.2 进一步工作的方向

本文的研究虽然在暖通空调系统的自动设计方面取得了初步的成果，但依然任重道远，尚有许多有待进一步深入进行的研究工作，这里择其要者简要讨论如下：

1) 目前的几何转化算法仍需进一步的校验，尤其对于复杂的 BIM 模型需要对几何进行一定的简化；
2) 该算法尚未考虑到侧送风口的选型，有待后续研究完善
3) 目前针对走廊的主管生成算法，仅适用于可直接分割为若干个长条形矩形的几何，尚未提出一种通用性更强的算法；

4) 目前的最小生成树遍历算法还未能适用于区域内布置节点数目较多的情况；
5) 在最少拐点路径算法中，目前仅考虑了两个节点之间的连接方法，还未将该连接方式与其他支路或主管之间的连接之间产生的影响；
6) 目前生成的管路是“虚拟”的管路，将来将根据实际工程完善每个管段和构件（弯头、三通、阀门、变径等）的定义，并进一步实现基于生成的管理进行自动水力计算的功能；
7) 目前只对单一管路系统做了研究，目前仅对全空气系统做了较为完整的测试，将来将继续完善多管路系统的研究，例如风机盘管+新风的空调形式，需要考虑多个管道之间的干扰问题。

致谢

写下这份致谢时，已经在家里宅过将近两个月，有人说人生是一场华丽的冒险，充满着意外和惊喜，就像我从未料想我的研究生生涯会以这样的方式画上句点。

缘分的奇妙之处就在于哪怕你知道要加入一个大家庭，但你无法预料一个个个体之间会发生什么样的故事。希望以后回顾起这段两年半的生活，还能记起许老师的谆谆教诲，大宝师兄在雨夜载我们吃过的地锅鸡，记得沙师姐煮过的奶茶，记得顾老师和杨经理称号的由来，记得帆帆师兄讲过的段子，记得智博炫酷的车技，记得天寒一同吃过的外卖看过的奇葩说，记得 A434 影院带给我们的惊与喜，记得那场泼水节的狂欢，也还记得黄山落日的秀美……

感谢论文撰写和 PPT 制作过程中给我提出宝贵建议的顾老师和肖桐师妹；感谢在申请学校过程中帮助过我的同学和老师，尤其是给了我很多建议的帆帆师兄和帮我写推荐信的许老师，潘老师和周老师。

最后，要感谢一直支持我深造的父母，是你们一路上给予了我关心和陪伴。

本课题受 2017 年国家重点研发计划项目《多能互补集成优化的分布式能源系统示范》项目 2017YFB0903404 支持。

2020 年 3 月

参考文献

[1] 潘毅群等. 实用建筑能耗模拟手册. 中国建筑工业出版社, 2013

[2] 潘毅群等. 建筑能耗模拟前沿技术与高级应用. 中国建筑工业出版社, 2019

[3] 天津生态城绿色建筑研究院等. 建筑能耗模拟及eQUEST & DeST操作教程. 中国建筑工业出版社, 2014

[4] Turhan Cihan, Kazanasmaz Tugce, Uygun Ilknur Erlalelitepe, et al. Comparative study of a building energy performance software (KEP-IYTE-ESS) and ANN-based building heat load estimation. Energy and Buildings, 2014,85:115-125

[5] Kumar Sachin, Pal Saibal K., Singh Ram Pal. A novel method based on extreme learning machine to predict heating and cooling load through design and structural attributes. Energy and Buildings, 2018,176:275-286

[6] Harish V. S. K. V., Kumar Arun. A review on modeling and simulation of building energy systems. Renewable and Sustainable Energy Reviews, 2016,56:1272-1292

[7] Ekici Betul Bektas, Aksoy U. Teoman. Prediction of building energy consumption by using artificial neural networks. Advances in Engineering Software, 2009,40(5):356-362

[8] Zeng Yaohui, Zhang Zijun, Kusiak Andrew. Predictive modeling and optimization of a multi-zone HVAC system with data mining and firefly algorithms. Energy, 2015,86:393-402

[9] Wang Shengwei, Xu Xinhua. Simplified building model for transient thermal performance estimation using GA-based parameter identification. International Journal of Thermal Sciences, 2006,45(4):419-432

[10] Kalagasidis Angela Sasic, Weitzmann Peter, Nielsen Toke Rammer, et al. The International Building Physics Toolbox in Simulink. Energy and Buildings, 2007,39(6):665-674

[11] Yildiz B., Bilbao J. I., Sproul A. B. A review and analysis of regression and machine learning models on commercial building electricity load forecasting. Renewable and Sustainable Energy Reviews, 2017,73:1104-1122

[12] Dogan Timur, Reinhart Christoph. Shoeboxer: An algorithm for abstracted rapid multi-zone urban building energy model generation and simulation. Energy and Buildings, 2017,140:140-153

[13] Dogan Timur, Reinhart Christoph, Michalatos Panagiotis. Automated multi-zone building energy model generation for schematic design and urban massing studies: IBPSA eSim conference, Ottawa, Canada[Z]. 2014.

[14] R. Brahme, A. Mahdavi, K. P. Lam, et al. COMPLEX BUILDING PERFORMANCE ANALYSIS IN EARLY STAGES OF DESIGN: A solution based on differential modeling, homology-based mapping, and generative design agents. Seventh International IBPSA Conference. 2001.

[15] Bazjanac Vladimir. Implementation of semi-automated energy performance simulation: building geometry: CIB W[Z]. 2009: 78, 595-602.

[16] Giannakis G. I., Lilis G. N., Garcia M. A., et al. A methodology to automatically generate geometry inputs for energy performance simulation from IFC BIM models: 14th International Conference of IBPSA-Building Simulation 2015, BS 2015, Conference Proceedings[Z]. IBPSA, 2015504-511.

[17] 梁思雨. 面向EnergyPlus的BIM数据处理方法研究: [硕士学位论文]. 西安建筑科技大学,

2018

[18] 陈远, 康虹, 范运昌. 基于IFC与gbXML标准的建筑信息模型与绿色建筑分析软件互操作性测试与评估. 图学学报, 2018,39(03):530-537.

[19] 范运昌. 基于IFC的BIM模型与绿色建筑分析软件互操作性研究: [硕士学位论文]. 郑州大学, 2017

[20] 陈远, 范运昌. 基于IFC与gbXML标准的绿色建筑信息模型数据标准及应用研究. 土木建筑工程信息技术, 2018,10(01):9-15.

[21] 黄多娜. 建筑信息模型（BIM）与能量分析程序的互用性研究: [硕士学位论文]. 哈尔滨工业大学, 2013

[22] 孙红三, 吴如宏, 燕达. 建筑能耗模拟软件的BIM数据接口开发与应用. 建筑科学, 2013,29(12):11-15.

[23] 林佳瑞, 张建平. 基于IFC的绿色性能分析数据转换与共享. 清华大学学报(自然科学版), 2016,56(09):997-1002.

[24] 王鸿鑫, 许鹏, 郭明月, 等. 基于BIM的能耗模拟软件功能的测评分析. 建设科技, 2019(16):15-19.

[25] Ellis M. W., Mathews E. H. Needs and trends in building and HVAC system design tools. Building and Environment, 2002,37(5):461-470

[26] Sönmez Nizam Onur. A review of the use of examples for automating architectural design tasks. Computer-Aided Design, 2018,96:13-30

[27] Lee W. L., Lee S. H., Farmani Raziyeh. Developing a simplified model for evaluating chiller-system configurations. Applied Energy, 2007,84(3):290-306

[28] Kang Yingzi, Augenbroe Godfried, Li Qi, et al. Effects of scenario uncertainty on chiller sizing method. Applied Thermal Engineering, 2017,123:187-195

[29] Sun Yuming, Gu Li, Wu C. F. Jeff, et al. Exploring HVAC system sizing under uncertainty. Energy and Buildings, 2014,81:243-252

[30] Huang Pei, Huang Gongsheng, Wang Yu. HVAC system design under peak load prediction uncertainty using multiple-criterion decision making technique. Energy and Buildings, 2015,91:26-36

[31] Medjdoub Benachir, Richens Paul, Barnard Nick. Generation of variational standard plant room solutions. Automation in Construction, 2003,12(2):155-166

[32] Medjdoub Benachir, Chenini Mokhtaria Benzohra. A constraint-based parametric model to support building services design exploration. Architectural Engineering and Design Management, 2015,11(2):123-136

[33] Stanescu Magdalena, Stanislaw Kajl, Louis Lamarche. Evolutionary algorithm with three different permutation options used for preliminary HVAC system design[Z]. 2012.

[34] Liu Qiang, Wang Chengen. A discrete particle swarm optimization algorithm for rectilinear branch pipe routing. Assembly Automation, 2011,31(4):363-368

[35] Liu Qiang, Wang Chengen. A Modified Particle Swarm Optimizer for Pipe Route Design. IEEE, 2008. 157-161

[36] Liu Qiang, Wang Chengen. Multi-terminal pipe routing by Steiner minimal tree and particle swarm optimisation. Enterprise Information Systems, 2012,6(3
):315-327

[37] Liu Lijia, Liu Qiang. Multi-objective routing of multi-terminal rectilinear pipe in 3D space by

MOEA/D and RSMT. IEEE, 2018. 462-467
[38] Liu Qiang. A rectilinear pipe routing algorithm: Manhattan visibility graph. International Journal of Computer Integrated Manufacturing, 2016,29(2):202-211
[39] Wang Chengen, Liu Qiang. Projection and Geodesic-Based Pipe Routing Algorithm. IEEE Transactions on Automation Science and Engineering, 2011,8(3):641-645
[40] Yuan Changtao Wang Xiaotong Sun. A Method Based Genetic Algorithm for Pipe Routing Design. 2015
[41] Qu Yanfeng, Jiang Dan, Yang Qingyan. Branch pipe routing based on 3D connection graph and concurrent ant colony optimization algorithm. Journal of Intelligent Manufacturing, 2018,29(7):1647-1657
[42] Sui Haiteng, Niu Wentie. Branch-pipe-routing approach for ships using improved genetic algorithm. Frontiers of Mechanical Engineering, 2016,11(3):316-323
[43] Geisberger Robert, Vetter Christian. Efficient routing in road networks with turn costs: International Symposium on Experimental Algorithms[Z]. Springer, 2011100-111.
[44] Park Jin-Hyung, Storch Richard L. Pipe-routing algorithm development: case study of a ship engine room design. Expert Systems With Applications, 2002,23(3):299-309
[45] Medjdoub Benachir. Constraint-based adaptation for complex space configuration in building services. 2009
[46] Teo Tee-Ann, Cho Kuan-Hsun, A. Mahdavi. BIM-oriented indoor network model for indoor and outdoor combined route planning. Advanced Engineering Informatics, 2016,30(3):268-282
[47] Aurelien Bres Florian Judex Georg Suter. A Method for Automated Generation of HVAC Distribution Subsystems for Building Performance Simulation. 2017
[48] Sandurkar Sunand, Chen Wei. GAPRUS—genetic algorithms based pipe routing using tessellated objects. Computers in Industry, 1999,38(3):209-223
[49] Suter Georg, Petrushevski Filip, Šipetić Miloš. Operations on network-based space layouts for modeling multiple space views of buildings. Advanced Engineering Informatics, 2014,28(4):395-411
[50] Runde S., Dibowski H., Fay A., et al. Integrated automated design approach for building automation systems. IEEE, 2008. 1488-1495
[51] 唐崇. 基于加速A*算法的游戏网格地图寻径研究: [硕士学位论文]. 江西师范大学, 2015
[52] 刘梓良. 面向游戏地图的寻径算法的研究与实现: [硕士学位论文]. 东南大学, 2016
[53] 陈素琼. 基于改进A~*算法的地图游戏寻径研究: 重庆师范大学, 2016
[54] 张永旭. 基于路径搜索的改进A*算法研究: [硕士学位论文]. 哈尔滨工程大学, 2017
[55] 唐晓东, 吴静. 基于改进A*算法的三维航迹规划技术研究. 电子技术应用, 2015,41(05):163-166.
[56] 马飞, 杨皞岫, 顾青, 等. 基于改进A*算法的地下无人铲运机导航路径规划. 农业机械学报, 2015,46(07):303-309.
[57] 赵晓, 王铮, 黄程侃, 等. 基于改进A*算法的移动机器人路径规划. 机器人, 2018,40(06):903-910.
[58] 刘晨曦. 基于改进A*算法的仿人机器人路径规划研究: [硕士学位论文]. 北京建筑大学, 2018
[59] 郝新培. 单线图自动成图技术研究: 山东大学, 2016
[60] 段建民, 陈强龙. 利用先验知识的Q-Learning路径规划算法研究. 电光与控制, 2019,26(09):29-33.
[61] 管亚丽. 基于DNA进化算法的多目标物流配送路径优化. 商场现代化, 2009(09):153.

[62] 聂廷哲, 段常贵. 基于Hopfield神经网络的输气管网布线优化. 天然气工业, 2005(02):155-157.
[63] 焦国帅, 柳强. 基于NSGA-Ⅱ的三维空间直角管路布局优化. 控制工程, 2018,25(11):2058-2063.
[64] Belaid Emna, Limbourg Sabine, Mostert Martine, et al. Bi-objective Road and Pipe Network Design for Crude Oil Transport in the Sfax Region in Tunisia. Procedia Engineering, 2016,142:108-115
[65] Duan Huan-feng, Yu Guo-ping. Spanning tree-based algorithm for hydraulic simulation of large-scale water supply networks. Water Science and Engineering, 2010,3(1):23-35
[66] Gabow Harold N., Myers Eugene W. Finding All Spanning Trees of Directed and Undirected Graphs. SIAM Journal on Computing, 1978,7(3):280-287
[67] Kapoor Sanjiv, Ramesh Hariharan. Algorithms for enumerating all spanning trees of undirected and weighted graphs. SIAM Journal on Computing, 1995,24(2):247-265
[68] Shioura Akiyoshi, Tamura Akihisa. Efficiently scanning all spanning trees of an undirected graph. Journal of the Operations Research Society of Japan, 1995,38(3):331-344
[69] Sörensen Kenneth, Gerrit K. Janssens. An algorithm to generate all spanning trees of a graph in order of increasing cost. 2015
[70] Wright Perrin. Counting and constructing minimal spanning trees. Bulletin of the Institute of Combinatorics and its Applications, 1997,21:65-76
[71] Eppstein David. Representing all minimum spanning trees with applications to counting and generation. 1995
[72] Yamada Takeo, Kataoka Seiji, Watanabe Kohtaro. Listing all the minimum spanning trees in an undirected graph. International Journal of Computer Mathematics, 2010,87(14):3175-3185
[73] Chen Yan, Treado Stephen. Development of a simulation platform based on dynamic models for HVAC control analysis. Energy and Buildings, 2014,68:376-386
[74] Wei Xiupeng, Kusiak Andrew, Li Mingyang, et al. Multi-objective optimization of the HVAC (heating, ventilation, and air conditioning) system performance. Energy, 2015,83:294-306
[75] Kusiak Andrew, Xu Guanglin. Modeling and optimization of HVAC systems using a dynamic neural network. Energy, 2012,42(1):241-250
[76] Magnier Laurent, Haghighat Fariborz. Multiobjective optimization of building design using TRNSYS simulations, genetic algorithm, and Artificial Neural Network. Building and Environment, 2010,45(3):739-746
[77] Moosavian Naser. Multilinear Method for Hydraulic Analysis of Pipe Networks. JOURNAL OF IRRIGATION AND DRAINAGE ENGINEERING, 2017,143(040170208)
[78] Guirardello Reginaldo, Swaney Ross E. Optimization of process plant layout with pipe routing. Computers & Chemical Engineering, 2005,30(1):99-114
[79] 杨志伟, 许鹏, 陈喆. 基于BIM的暖通空调自动设计. 建设科技, 2018(23):44-48.
[80] 杨志伟. 暖通空调系统结构自动化设计: [硕士学位论文]. 同济大学, 2019
[81] https://www.gbxml.org/schema_doc/6.01/GreenBuildingXML_Ver6.01.html[Z].
[82] 陆耀庆. 实用供热空调设计手册 第2版. 北京: 中国建筑工业出版社, 2008. 1439
[83] 曹新明, 蒋瑞斌. 不规则零件最小包络矩形的求解研究. 科技通报, 2007(01):102-105.
[84] 周敏, 郑国磊, 陈树林. 二维图形最狭长包络矩形的求解原理及方法. 图学学报, 2013,34(04):46-53.
[85] 曾金平, 张忠志. 线性代数. 北京: 北京邮电大学出版社, 2017. 154

[86] 董秀山，刘润涛．判断点与简单多边形位置关系的新算法．计算机工程与应用，2009,45(02):185-186.
[87] 高随祥．图论与网络流理论．高等教育出版社, 2009
[88] https://zh.wikipedia.org/wiki/%E6%A0%91_(%E5%9B%BE%E8%AE%BA).
[89] 张乃孝．算法与数据结构．高等教育出版社, 2006
[90] Cormen Thomas H., Leiserson Charles E., Rivest Ronald, et al. Introduction to algorithms. MIT Press, 2001
[91] 张萍．高等数学．科学出版社, 2004
[92] 姜建国等．组合数学．西安电子科技大学出版社, 2007
[93] 贾庆轩，陈钢，孙汉旭，等．基于A~*算法的空间机械臂避障路径规划．机械工程学报，2010,46(13):109-115.
[94] Kumar Neerendra, Vamossy Zoltan, Szabo-Resch Zsolt Miklos. Heuristic Approaches in Robot Navigation: IEEE International Conference on Intelligent Engineering Systems[Z]. SZAKAL A. 2016219-222.
[95] Hart Peter E., Nilsson Nils J., Raphael Bertram. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 1968,4(2):100-107
[96] Russell Stuart．人工智能——一种现代方法．人民邮电出版社, 2010
[97] Zhang An, Li Chong, Bi Wenhao. Rectangle expansion A* pathfinding for grid maps. CHINESE JOURNAL OF AERONAUTICS, 2016,29(5):1385-1396

附录 A 部分 python 算法代码

```
#散流器分布生成
def segment_ratio(a=12, b=18, ratio=1.41421):
    """
    a: width
    b: height
    ratio: maximum ratio of the segment ratio
    return: the possible distribution result
    """
    seg1=5.99
    seg2=3.99
        a_range=list(range(math.ceil(a/seg1), round(a/seg2)+1))
    b_range=list(range(math.ceil(b/seg1), round(b/seg2)+1))
    if((a-seg1)*(b-seg1)<0)&((a/b>=1)&(a/b>=1<=ratio)):
        a_range.append(1)
    if((a-seg1)*(b-seg1)<0)&((b/a>=1)&(b/a>=1<=ratio)):
        b_range.append(1)
    #方形散流器/圆形散流器,要求区域内长宽比 ratio 以内
    distribution=[]
    minratio =10e5
    for i in a_range:
        for j in b_range:
            ai = a/i
            bj = b/j
            #print(ai, bj)
            if (ai/bj>=1)&(ai/bj<=ratio)|(bj/ai>=1)&(bj/ai<=ratio):
                if max(ai/bj,bj/ai)<minratio:
                    minratio = max(ai/bj,bj/ai)
                    distribution=[i, j]

#散流器布置算法
def draw_diffuser(obj,dist_max = 1.5,ratio = 1,segopt = 'minnum'):
    points = returnpoints(obj)
    vertices = points2vertices(points)
    if len(vertices)!=4:
        vertices.pop(-1)
    points.sort(key = getx)
    x1 = points[0].x
    x2 = points[-1].x
    points.sort(key=gety)
```

```
    y1 = points[0].y
    y2 = points[-1].y
    length = (x2 - x1)
    width = (y2 - y1)
    if segopt == 'minnum':
        seg = segment(length/ratio,width/ratio)
    if segopt == 'minratio':
        seg = segment_ratio(length/ratio,width/ratio)
    deltalen = length/seg[0]
    deltawid = width/seg[1]
    location = []
    for i in range(seg[0]):
        for j in range(seg[1]):
            newx = x1 + deltalen*(i+0.5)
            newy = y1 + deltawid*(j+0.5)
            location.append(APoint(newx,newy))
    poly = Polygon(vertices)
    for loc in location:
        p1 = Point(loc)
        if p1.within(poly)&(poly.exterior.distance(p1)>dist_max):
            acad.model.AddCircle(loc, 0.2*ratio)

#末端有权图生成
def terminallist2grah(terminallist):
    graph = nx.Graph()
    for item in terminallist:
    #       item = terminallist[0]
        node1 = item.bounds[:2]
        graph.add_node(node1,pos=node1)
        tempxplus = [itemx for itemx in terminallist if item.y== itemx.y and
itemx.x>item.x]
        tempxplus.sort(key = lambda it: it.x-item.x)
        for nearlistxplus in tempxplus[0:1]:
            nodex = nearlistxplus.bounds[:2]
            graph.add_node(nodex,pos=nodex)
            graph.add_edge(node1,nodex)

        tempxminus = [itemx for itemx in terminallist if item.y== itemx.y and
itemx.x<item.x]
        tempxminus.sort(key = lambda it: item.x-it.x)
        for nearlistxminus in tempxminus[0:1]:
            nodex = nearlistxminus.bounds[:2]
            graph.add_node(nodex,pos=nodex)
            graph.add_edge(node1,nodex)
```

```
            tempyplus = [itemy for itemy in terminallist if item.x== itemy.x and
itemy.y>item.y]
            tempyplus.sort(key = lambda it: it.y-item.y)
            for nearlistyplus in tempyplus[0:1]:
                  nodey = nearlistyplus.bounds[:2]
                  graph.add_node(nodey,pos=nodey)
                  graph.add_edge(node1,nodey)

            tempyminus = [itemy for itemy in terminallist if item.x== itemy.x and
itemy.y<item.y]
            tempyminus.sort(key = lambda it: item.y-it.y)
            for nearlistyminus in tempyminus[0:1]:
                  nodey = nearlistyminus.bounds[:2]
                  graph.add_node(nodey,pos=nodey)
                  graph.add_edge(node1,nodey)
      return graph

#遍历最小生成树算法 1
def stgenerator(graph):
      nodenum = len(graph.nodes)
      edgenum = len(graph.edges)
      cutlist = itertools.combinations(graph.edges,edgenum-nodenum+1)
      minipathsum = 10e10
      shortgraph = nx.Graph()
      for item in cutlist:
            trygraph = graph.copy()
            trygraph.remove_edges_from(item)
            try:
                  cycles.find_cycle(trygraph)
            except:
                  addweight(trygraph)
#                   edgelabels = nx.get_edge_attributes(trygraph,'weight')
#                   postrygraph=nx.get_node_attributes(trygraph,'pos')
#                   plt.figure(figsize=(16, 10))
#                   nx.draw_networkx_edge_labels(trygraph, postrygraph,
edgelabels,font_size=16)
#
nx.draw_networkx(trygraph,postrygraph,with_labels=False,font_size=15)
                  pathsum = 0
                  for item in npointlist[:-1]:
                        pathsum += nx.dijkstra_path_length(trygraph,(npointlist[-
1].x,npointlist[-1].y),(item.x,item.y))
#                   print(pathsum)
```

```
                if pathsum<minipathsum:
                    minipathsum=pathsum
                    shortgraph=trygraph.copy()

    edgelabels = nx.get_edge_attributes(shortgraph,'weight')
    posshortgraph=nx.get_node_attributes(shortgraph,'pos')
    plt.figure(figsize=(16, 10))
    nx.draw_networkx_edge_labels(shortgraph, posshortgraph,
edgelabels,font_size=16)
    nx.draw_networkx(shortgraph,posshortgraph,with_labels=False,font_size=15)
    return shortgraph


#遍历最小生成树算法 2
def mst_generator(graph,zeroweightpointlist=[]):
    mst = nx.minimum_spanning_tree(graph,algorithm='kruskal')
    left_graph=nx.algorithms.operators.difference(graph,mst)
    left_graph = addweight(left_graph)
    comblist = []
    for edge in mst.edges():
        edge_weight = mst[edge[0]][edge[1]]['weight']
        cut_graph = mst.copy()
        cut_graph.remove_edges_from([edge])
        combedge = [edge]
        for left_edge in left_graph.edges:
            if edge_weight == graph[left_graph[0]][left_graph[1]]['weight']:
                try_graph = cut_graph.copy()
                try_graph.add_edges_from([leftedge])
                try:
    #                 print('circle')
                    cycles.find_cycle(try_graph)
                except:
    #                 print('nocircle')
                    combedge.append(leftedge)
        comblist.append(combedge)
    gra = nx.Graph()
    for node in graph.nodes:
        gra.add_node(node,pos=node)
    msts = itertools.product(*comblist)
    for item in msts:
        gr = gra.copy()
        gr.add_edges_from(item)
        gr = addweightwitho0points(gr,zeroweightpointlist)
        yield gr
```

```
#三通识别
def triplet(Graph):
    i = 0
    for node in Graph.nodes:
        j = 0
        for k in all_neighbors(Graph,node):
            j += 1
        if j == 3:
            i += 1
    return i

#四通识别
def quadlet(Graph):
    i = 0
    for node in Graph.nodes:
        j = 0
        for k in all_neighbors(Graph,node):
            j += 1
        if j == 4:
            i += 1
    return i

#弯头识别
def turn(Graph):
    i = 0
    for node in Graph.nodes:
        j = 0
        neighbor = []
        for k in all_neighbors(Graph,node):
            j += 1
            neighbor.append(k)
        if j == 2:
            dotmultiply = (neighbor[0][0]-node[0])*(neighbor[1][0]-node[0]) +
(neighbor[0][1]-node[1])*(neighbor[1][1]-node[1])
            if dotmultiply == 0:
                i += 1
    return i

#相交关系识别
def find_intersection(obj1,obj2):
    if obj1.intersection(obj2).bounds != ():
        return obj1.intersection(obj2)
    else:
```

```
            return False

#内部相交关系识别
def intersection_within_obj1(obj1,obj2):
    interpoint = obj1.intersection(obj2)
    if obj1.contains(interpoint):
        return interpoint
    else:
        return False

#找到边中存在相交的点
def find_interpoints(edge,edgelist):
    interpoints = []
    for item in edgelist:
        if (edge!= item)&bool((intersection_within_obj1 (edge,item))):
            interpoints.append(intersection_within_obj1 (edge,item))
    return interpoints

#根据新的节点将现有图增加节点
def modify_edge(edge,newnodelist):
    newedgelist = []
    nodes = newnodelist+[edge.boundary[0],edge.boundary[1]]
    nodes.sort(key=lambda point: [point.x,point.y])
    for item in enumerate(nodes[:-1]):
        newedgelist.append(LineString([nodes[item[0]],nodes[item[0]+1]]))
    return newedgelist

#将一系列边转化为图
def edge2graph(edgelist):
    G = nx.Graph()
    newedgelist = []
    for edge in edgelist:
        if find_interpoints(edge,edgelist)==[]:
            newedgelist.append(edge)
        else:
            interpoints = find_interpoints(edge,edgelist)
            newedgelist.extend(modify_edge(edge,interpoints))
    for edge in newedgelist:
        node1 = edge.bounds[:2]
        node2 = edge.bounds[2:]
        G.add_node(node1,pos=node1)
        G.add_node(node2,pos=node2)
        G.add_edge(node1,node2)
    return G
```

```
#找到相交的点所在的边
def npoint_in_edge(npoint,graph):
    for item in graph.edges:
        line = LineString([Point(item[0]),Point(item[1])])
        if npoint.distance(line)==0:
            return item

#判别点是否为图中的节点
def is_node(point,graph):
    a = 0
    for node in graph.nodes:
        if Point(node)==Point(point):
            a += 1
    if a==0:
        return False
    else:
        return True

#为现有的图增加一系列边
def modify_graph_edge(graph,oldedge,newedgelist):

    graph.remove_edge(oldedge[0],oldedge[1])
    for item in newedgelist:
        node1 = item.bounds[:2]
        node2 = item.bounds[2:]
        graph.add_node(node1,pos=node1)
        graph.add_node(node2,pos=node2)
        graph.add_edge(node1,node2)
    return graph

# 为主管的图增加末端节点
def addnpoint2graph(npoints,graph):
    np_edge_list = []
    node_terminal_list = []
    non_node_terminal_list = []
    for item in npoints:
        edge = npoint_in_edge(item,graph)
        npedgelist.append(((item),edge))
    for item in np_edge_list:
        if is_node(item[0],graph):
            node_terminal_list.append(item[1])
        else:
            non_node_terminal_list.append(item)
```

```
    edgeterdict = {}
    for item in non_node_terminal_list:
        if item[1] not in edgeterdict:
            edgeterdict[item[1]] = [item[0]]
        else:
            edgeterdict[item[1]].append(item[0])
    for edge,pointlist in edgeterdict.items():
        newedges = modifyedge(LineString(edge),pointlist)
        modify_graph_edge(graph,edge,newedges)
    return graph,node_terminal_list

#将主管主管与增加末端节点相连
def linkt_erminal(graph,terminals,nearpoints):
    linkedgraph=copy.deepcopy(graph)
    for i in range(len(terminals)):
        node1 = terminals[i].bounds[:2]
        node2 = nearpoints[i].bounds[:2]
        linkedgraph.add_node(node1,pos=node1)
        linkedgraph.add_edge(node1,node2)
    return linkedgraph

#计算两个点的曼哈顿距离
def manhattan_dist(point1,point2):
    point1 = Point(point1)
    point2 = Point(point2)
    return round(abs(point1.x-point2.x)+abs(point1.y-point2.y),2)

#计算两个点的欧几里得距离
def euclid_dist(point1,point2):
    point1 = Point(point1)
    point2 = Point(point2)
    return round(math.sqrt((point1.x-point2.x)**2+(point1.y-point2.y)**2),2)

#为图增加曼哈顿权重
def add_mhweight(graph):
    edgeweight = []
    for edge in graph.edges:
        dist = manhattandist(edge[0],edge[1])
        edgeweight.append((edge[0],edge[1],dist))
    graph.add_weighted_edges_from(edgeweight)
    return graph

#为图增加欧几里得权重
def addweight(graph):
```

```
    edgeweight = []
    for edge in graph.edges:
        dist = euclliddist(edge[0],edge[1])
        edgeweight.append((edge[0],edge[1],dist))
    graph.add_weighted_edges_from(edgeweight)
    return graph

#修剪生成的走廊连接图
def trimnode(corrgraph,linkgraph):
    trimmedgraph = copy.deepcopy(linkgraph)
    for node in list(trimmedgraph.nodes):
        i = len(list(nx.function.neighbors(trimmedgraph,node)))
        if i == 1:
            if node in corrgraph.nodes:
                trimmedgraph.remove_node(node)
    for node in list(trimmedgraph.nodes):
        i = len(list(nx.function.neighbors(trimmedgraph,node)))
        if i == 1:
            if node in corrgraph.nodes:
                trimmedgraph.remove_node(node)
    for node in list(trimmedgraph.nodes):
        i = len(list(nx.function.neighbors(trimmedgraph,node)))
        if i == 1:
            if node in corrgraph.nodes:
                trimmedgraph.remove_node(node)
    return trimmedgraph
```

个人简历、在读期间发表的学术论文与研究成果

个人简历：

陈喆，男，1995 年 6 月生。

2017 年 6 月毕业南京航空航天大学　建筑环境与能源应用专业 获学士学位。

2017 年 9 月入同济大学读硕士研究生。

已发表论文：

[1] Chen, Yongbao, et al. "Quantification of electricity flexibility in demand response: Office building case study." Energy (2019): 116054.

[2] Chen, Zhe, Peng Xu, and Yongbao Chen. "A Peer-to-Peer Electricity System and Its Simulation." IOP Conference Series: Earth and Environmental Science. Vol. 238. No. 1. IOP Publishing, 2019.

[3] 陈喆，许鹏 等.既有公共建筑调适潜力分析及调适价值评价 暖通空调 2019，49（12）

[4] 杨志伟,许鹏,陈喆.基于 BM 的暖通空调自动设计 建设科技,2018(23):44-48.

参加学术会议：

[1] 2018 年 12 月 International Building Performance Simulation Association - Asim 2018 香港，中国

[2] 2019 年 7 月，华人能源与人工环境国际学术会议，成都，中国