苏州大学软件学院“慧工云杯”

第二届工业软件实践创新大赛设计文档

|  |  |
| --- | --- |
| 团队名称： | 即将获得一万奖金 |
| 所在学院： | 计算机科学与技术学院 |
| 所在专业： | 计算机科学与技术 |
| 队 长： | 纪东旭 |
| 联系电话： | 15841778985 |
| 队 员： | 彭晖、袁天博 |

**目录**

[基于蚁群算法的工业排程问题方案设计 2](#_Toc173146243)

[一、工业排程问题和蚁群算法 2](#_Toc173146244)

[二、问题描述 3](#_Toc173146245)

[三、蚁群算法原理 3](#_Toc173146246)

[3.1 信息素（Pheromone） 3](#_Toc173146247)

[3.2 启发式信息（Heuristic Information） 4](#_Toc173146248)

[3.3 蚂蚁的移动规则 4](#_Toc173146249)

[3.4 蚁群算法工作流程 4](#_Toc173146250)

[3.5 优化后蚁群算法工作流程 5](#_Toc173146251)

[四、数据结构 5](#_Toc173146252)

[4.1 字典（Dict）（程序中部分变量） 5](#_Toc173146253)

[4.2 列表（List）（程序中部分变量） 6](#_Toc173146254)

[4.3 嵌套数据结构（程序中部分变量） 6](#_Toc173146255)

[五、数据的解析与处理 6](#_Toc173146256)

[六、设计架构 10](#_Toc173146257)

[6.1 数据准备模块 10](#_Toc173146258)

[6.2 蚁群算法模块 11](#_Toc173146259)

[6.3 可视化算法模块 11](#_Toc173146260)

[七、函数详细说明 11](#_Toc173146261)

[7.1 数据准备模块函数说明 11](#_Toc173146262)

[7.2 蚁群算法模块函数说明 16](#_Toc173146263)

[7.3 可视化模块函数说明 19](#_Toc173146264)

[八、参数设置 19](#_Toc173146265)

[九、总结和展望 20](#_Toc173146266)

[9.1 工作成果 20](#_Toc173146267)

[9.2 优化空间 20](#_Toc173146268)

# 基于蚁群算法的工业排程问题方案设计

## 工业排程问题和蚁群算法

工业排程问题是生产过程中关键的计划和调度任务，涉及到任务分配、资源调度、时间安排等方面。其目的是在最短的时间内完成生产任务，最大化资源利用率，降低成本，提高生产效率。

蚁群算法是一种启发式优化算法，灵感来源于蚁群在寻找食物过程中的行为。蚁群算法模拟了蚁群在寻找食物时的行为，通过信息素和启发式信息的相互作用，使蚂蚁群体能够找到较优的路径，适用于解决组合优化问题。 在工业排程问题中，蚁群算法可以应用于优化生产任务的调度顺序和资源分配，以实现生产过程的最优化。通过模拟蚂蚁在任务和资源之间的搜索过程，蚁群算法可以帮助优化生产计划，提高生产效率和资源利用率。

因此，将蚁群算法应用于工业排程问题可以帮助解决生产过程中的调度难题，优化生产计划，提高生产效率，降低成本，实现生产过程的智能化和优化化。

## 问题描述

在装备制造行业，特别是在制造业和工程项目管理中，要考虑原材料、设备和人力资源的限制，根据工作中心的能力，工人的技能和项目需求，合理分配任务到相应的工作中心和工人，以减少生产周期、提高生产效率，并确保项目按时交付。此次赛题主要的解决难点为两个方面：一是如何将各种限制以及特殊情况数据化，即将其转换为可参与运算的数据；二是如何合理优化蚁群算法来更好地适应实际情况。

针对以上两个难点，我们提出几项简化方案：

1.机器视为24h全天候工作且不用维修

2.人力资源将平均工作效率视为指定工作效率

## 蚁群算法原理

蚁群算法是一种启发式优化算法，灵感来源于蚂蚁在寻找食物时的行为。蚁群算法模拟了蚂蚁群体在寻找食物过程中的行为，通过信息素和启发式信息的相互作用，实现对优化问题的求解。以下是蚁群算法的基本原理和工作方式：

### 3.1 信息素（Pheromone）

- 信息素是蚂蚁在路径上释放的化学物质，用于标记路径的好坏程度。

- 蚂蚁会根据路径上的信息素浓度选择下一步移动的方向，信息素浓度较高的路径更有可能被选择。

- 信息素的释放和更新受到蚂蚁经过路径的顺序和路径质量的影响。

### 3.2 启发式信息（Heuristic Information）

- 启发式信息是指蚂蚁根据问题本身的启发式信息来做出决策，帮助蚂蚁选择更优的路径。

- 启发式信息可以是距离、时间、资源消耗等问题特定的信息，用于指导蚂蚁的移动和选择。

### 3.3 蚂蚁的移动规则

- 蚂蚁在搜索过程中遵循一定的移动规则，包括路径选择、信息素更新等。

- 蚂蚁根据信息素和启发式信息选择下一步要移动的位置，一般会考虑信息素浓度和启发式信息的权重。

- 蚂蚁在移动后会释放信息素，信息素浓度与路径质量成正比，优质路径释放的信息素浓度更高。

### 3.4 蚁群算法工作流程

1. 初始化设置参数，包括蚂蚁数量、信息素浓度、启发式因子、最大迭代次数等。并初始化蚂蚁在解空间中的随机位置。

2. 每只蚂蚁从当前位置出发，根据信息素浓度和启发式因子的概率分布轮盘赌选择下一个节点。

3. 重复上一个过程，直到蚂蚁构建完成一条路径或解决方案。

4. 所有蚂蚁完成路径构建后，根据它们构建的路径的质量（例如，路径长度或成本）来更新信息素浓度。

5. 重复步骤2、3、4，直到达到最大迭代次数或找到满意的解

### 3.5 优化后蚁群算法工作流程

1. 初始化设置参数，除蚂蚁数量、信息素浓度、启发式因子、最大迭代次数等参数之外，还要根据前置工序等因素来初始化搜索禁忌表。

2. 随机轮盘赌一个节点，设置消耗资源

3. 根据变化后的资源来设置搜索禁忌表

4. 重复步骤2和3直到搜索禁忌表包含所有节点，表示当前蚁群无法再扩张

5. 记录以上选取节点的开始时间

6. 不断增加当前时间，用当前时间减去每个工序的开始时间，若结果大于等于该工序完成所需的时间，则该工序被完成，将资源放回。

7. 每当有工序完成时，重新设置禁忌搜索表，重复步骤2-6

8. 直到所有的工序都完成后，更新信息素。

9. 记录此时的最终时间，并与最短时间比较，重置资源列表和当前时间，并开始下一次循环。

10. 重复步骤2-9，直到达到最大迭代次数或找到满意的解

## 数据结构

在代码中，以下数据结构被使用（列举出一部分示例）

### 4.1 字典（Dict）（程序中部分变量）

front\_or\_back：存储每个工序是正排还是倒排。键为每个工序的名字，值为该工序是正排还是倒排，正排True，倒排False

Results：存储最终正排工序的执行字典，键为每个工序的名字，值为该工序开始时间和结束时间组成的列表

Locked\_work：存储锁排程数据，以工单编号和物料编号为键，开始时间和结束时间组成的列表作为值。

Results\_reverse：存储最终倒排的执行字典，结构与results相同

Min\_results：用时最少的正排工序的执行字典

Min\_results\_reverse：用时最少的倒排工序的执行字典

### 4.2 列表（List）（程序中部分变量）

Point\_table：工单、物料、工序结点表。存储着所有工单物料工序。

resources：在解析工艺路线数据时，用于收集当前工序的所有资源。 resources\_count：在解析锁排程数据时，用于收集当前工序的资源需求数量。

resources\_usetime：资源的开始时间及结束时间。包含每个资源的开始时间和结束时间列表

Finish\_work：已完成工序列表。记录工序是否已完成，用True和False表示。

### 4.3 嵌套数据结构（程序中部分变量）

Results，Results\_reverse中使用了嵌套字典结构，以工单编号和物料编号为键，工序的开始时间和结束时间作为值。该数据结构能够很好地表示一个工序的开始时间和结束时间，并在最后转换成answer.json

## 数据的解析与处理

从题目给定的表格中进行解析数据，得到需要的数据并处理成后续需要的数据格式，存储为output.json。我们处理之后的数据格式为一个嵌套字典结构，以工单编号和物料编号等为键，包含多个工序及其相关信息的字典作为值，资源日历，锁排程，设备换型的数据信息同样采用这样的嵌套字典结构，如下所示：

|  |
| --- |
| {  "全部工单": {  "MO03": {  "工单编号": "MO03",  "物料编码": "FG03",  "数量": 1,  "前置工单": "/",  "排程策略": "正排",  "计划开始时间": "2023/04/12",  "计划完工时间": "2023/04/23"  }  },  "工艺路线": {  "MO03": {  "FG03": {  "工序列表": [  "GY302",  "GY303",  "GY304"  ],  "前置工序": [  "/",  "/",  "/"  ],  "资源需求": [  [  "资源B1",  "资源C2"  ],  [],  []  ],  "准备工时": [  10,  10,  10  ],  "作业工时": [  10,  10,  5  ],  "后置工时": [  10,  20,  20  ],  "是否可跨班组": [  0,  0,  0  ]  }  }  },  "资源日历": {  "R004": {  "资源编号": "R004",  "资源名称": "资源A",  "资源分类": "设备",  "资源数量": 1.0,  "开始日期": "2023/04/12",  "结束日期": "2023/04/23",  "开始时间": "00:00",  "结束时间": "23:59",  "优先级": 1.0  }  },  "锁排程": {  "MO05": {  "FG05": {  "工序": "GY302",  "资源池": [  [  "资源A",  "资源B",  "资源C"  ]  ],  "资源ID": [  [  "R004",  "R005",  "R006"  ]  ],  "资源需求": [  [  1,  1,  2  ]  ],  "开始时间": [  [  "2023/04/12 00:30",  "2023/04/12 00:30",  "2023/04/12 00:30"  ]  ],  "结束时间": [  [  "2023/04/12 01:10",  "2023/04/12 01:10",  "2023/04/12 01:10"  ]  ]  }  }  },  "设备换型": {  "工序列表": [  "FG04 GY302",  "FG05 GY302",  "FG06 GY302"  ],  "换型矩阵": [  [  "0",  "5",  "10"  ],  [  "20",  "0",  "20"  ],  [  "5",  "10",  "0"  ]  ]  },  "排程策略": []  } |

**对于工艺路线数据处理的解读**：

一个工单编号对应一个物料编号，一个物料编号对应多个工序，分别为工序1，工序2等等，那么我需要将一个物料编号对应的多个工序都放在一个数组中，其键为工序列表。

接着一个工序对应一个前置工序，那么我需要将所有的前置工序同样放在一个数组中，这样我可以根据下标来访问一个工序对应的前置工序。例如：工序1没有对应的前置工序；工序2对应的前置工序是工序1，工序3对应的前置工序是工序2，那么对应的工序列表为：[工序1，工序2，工序3]，而前置工序为：[/，工序1，工序2]

一个工序对应多个资源需求，且每个资源需求有对应的需要的资源个数，那么需要将一个工序对应的多个资源需求及所需资源个数放在一个数组中。例如：若工序1对应的资源有A，B，C，且需要1个资源A，2个资源B，3个资源C；工序2和工序3对应的资源需求为空，那么对应的资源需求数组为：[[A1,B2,C3],[],[]]。

接着一个工序有其对应的准备工时，作业工时，后置工时，那么该工序的总工时为准备工时+作业工时+后置工时。我需要将每个工序对应的总工时计算出来，并放在数组中。例如：若工序1的总工时计算为30，工序2的总工时计算为40，工序3的总工时计算为50，那么对应的工时数组为[30,40,50]

最后，每个工序对应一个是否可跨班组占用资源，对应数据为1或0，我需要将每个工序对应的是否可跨班组放在一个数组中。例如：工序1是否可跨班组对应为1，工序2是否可跨班组对应为0，工序3是否可跨班组对应为1,。那么对应的是否可跨班组数组为：[1,0,1]。

## 设计架构

对于该算法实现的系统架构，我们将从三个模块进行介绍，分别是数据准备模块，蚁群算法模块和可视化模块。

### 6.1 数据准备模块

数据准备模块相关函数主要用于数据处理和数据读取，正排倒排情况下进行初始化资源、时间、信息素矩阵以及路径搜索。它们涵盖了从数据准备到路径选择的全过程。资源管理函数负责初始化资源表、计算资源需求，以及生成可用资源列表。时间管理函数用于初始化时间和处理时间计算。信息素管理函数包括更新信息素矩阵。确保蚁群算法顺利执行并找到最优路径解决方案。

### 6.2 蚁群算法模块

蚁群算法模块中的蚂蚁类和SSP类功能齐备，通过模拟蚂蚁行为和信息素更新来优化工序执行顺序，以最小化完成时间为目标。蚂蚁类负责模拟蚂蚁在路径选择、移动和计算距离等行为，帮助蚁群寻找最优路径。SSP类则实现了蚁群算法在解决SSP问题中的关键功能，包括路径搜索、资源管理、信息素更新和禁忌搜索表更新等操作。通过迭代优化和信息素更新，蚁群算法能够寻找到最佳的工序执行顺序，提高生产效率和资源利用率。这些功能共同组成了一个完整的蚁群算法模块，能够有效解决工序调度问题，优化生产过程并提高效率。

### 6.3 可视化算法模块

通过matplotlib库实现了甘特图和折线图的绘制，用于直观展示工序调度问题的解决过程和结果。甘特图展示了不同资源类型的任务执行情况，而折线图则反映了蚁群算法在不同迭代次数下的最佳完成时间情况。绘制甘特图的函数`draw\_gantt\_chart(json\_file\_path)`读取JSON文件，解析数据并填充任务列表，为每个工序创建子坐标，并根据资源类型分配颜色绘制甘特图，调整日期格式和设置刻度，并展示甘特图。在蚁群算法中记录每次迭代的最佳完成时间和迭代次数，使用matplotlib库绘制折线图，横坐标为迭代次数，纵坐标为最佳完成时间，设置标题和坐标轴标签，并显示绘制的折线图。这些可视化工具帮助更直观地理解和分析算法执行过程和结果，为工序调度问题提供可视化展示和分析。

## 函数详细说明

### 7.1 数据准备模块函数说明

#### 7.1.1 数据读取模块函数说明

ExcelToJsonConverter类：实现将已有的excel表格中的数据解析为我们需要的数据的格式，并保存为json文件。

read\_data(file\_path)：读取json数据的函数。

#### 7.1.2 数据初始化模块函数说明

**clean\_change\_equipment\_array(data)：**

初始化换型矩阵。首先，如果前置工序矩阵为空，将所有工序默认设置为True（即前置工序完成）。然后，遍历全部工单数据，检验含有前置工单的工单，若前置工单未完成，则将对应工序的前置工序设置为False（未完成）。最后，遍历工艺路线数据，检验前置工序，若前置工序未完成，则将当前工序的前置工序设置为False（未完成）。

**change\_point\_table(data)：**

初始化工单/物料/工序结点表。首先，如果工序节点表为空，遍历工艺路线数据，构建工序节点表。然后对于每个工序，提取工序名称并添加到工序节点表中，同时初始化工序完成状态和工序选择状态为False。最后，如果准备工时、作业工时和后置工时信息为空，从工艺路线数据中提取并存储到对应的列表中。

**change\_front\_back(data)：**

根据全部工单中的排程策略（正排或倒排），将工单分别留在point\_table中或取出放入point\_table\_post中。首先，从全部工单中读取工单的信息，包括工单编号、物料编码和排程策略。然后，如果工单的排程策略为正排，将该工单留在point\_table中。将工单的物料编码与point\_table中的对应工单匹配，若匹配则将其标记为正排工单。最后，如果工单的排程策略为倒排，将该工单取出并放入point\_table\_post中。将工单的物料编码与point\_table中的对应工单匹配，若匹配则将其标记为倒排工单。

**change\_post\_process(data)：**

初始化正排情况下一个名为 post\_process 的列表，用于表示每个工序或工单的前置条件是否满足。这个列表对于正排调度策略来说非常重要，它可以确保每个工序或工单只有在所有前置条件都满足的情况下才能被执行。首先，如果 post\_process 列表为空，则创建一个新的列表，其长度与 point\_table 列表相同。point\_table 列表存储了所有工序和工单的编号信息，所以 post\_process 列表中每个元素对应一个工序或工单。每个元素的初始值为 True，表示所有工序和工单的前置条件默认都满足。然后，判断前置工序，遍历 data["工艺路线"] 中的每个工序，检查每个工序的 前置工序 列表。对于每个前置工序，检查它是否在 point\_table 中，并判断它的完成状态（finish\_work）。如果前置工序未完成，则将该工序在 post\_process\_gongxu 列表中标记为 False，表示该工序的前置条件不满足。如果前置工序已完成，则将该工序在 post\_process\_gongxu 列表中标记为 True，表示该工序的前置条件满足。然后，遍历 data["全部工单"] 中的每个工单，检查每个工单的 前置工单 字段。对于每个前置工单，检查它是否在 point\_table 中，并判断它的完成状态（finish\_work）。如果前置工单未完成，则将该工单在 post\_process\_gongdan 列表中标记为 False，表示该工单的前置条件不满足。如果前置工单已完成，则将该工单在 post\_process\_gongdan 列表中标记为 True，表示该工单的前置条件满足。最后，遍历 post\_process 列表，将 post\_process\_gongdan 和 post\_process\_gongxu 列表中对应元素进行逻辑与运算，将结果更新到 post\_process 列表中。

**change\_pre\_process(data)：**

用于初始化倒排情况下所需的前置工序矩阵。首先，初始化前置工序矩阵，如果 pre\_process 列表为空，将其初始化为一个与 point\_table 长度相同的列表，初始值为 True。point\_table 列表存储了工序和工单的编号信息。然乎，判断前置工序情况，遍历 data["工艺路线"] 中的每个工序，提取工序的信息和前置工序列表。检查每个工序的前置工序，如果前置工序未完成，则将该工序的前置工序在 pre\_process\_gongxu 列表中标记为 False，表示前置条件不满足。遍历 data["全部工单"] 中的每个工单，检查每个工单的前置工单字段。如果前置工单未完成，则将该工单在 pre\_process\_gongdan 列表中标记为 False，表示前置条件不满足。最后，更新前置工序矩阵，将 pre\_process\_gongdan 和 pre\_process\_gongxu 列表中的对应元素进行逻辑与运算，将结果更新到 pre\_process 列表中。只有当工序和工单的前置条件都满足时，pre\_process 列表中对应元素才会为 True。

**change\_time\_needs(data)：**

旨在根据数据集中的工单信息和工艺路线，计算每个工序所需的总工时，并将结果存储在time\_needs列表中。首先，遍历全部工单数据，提取工单编号、物料编码和数量信息。然后匹配工序信息，根据工单编号和物料编码，在工艺路线中找到对应的工序列表。再计算每个工序的总工时，包括准备工时、作业工时和后置工时的累加。最后，存储结果：将每个工序的总工时存储在time\_needs列表中，以便后续算法使用。

**changeD(data)：**

根据输入的数据初始化距离矩阵D。首先，遍历工艺路线数据，提取每个工序的工时信息。然后，对于每个工序：复制时间需求数据，确保每个工序的时间需求独立。将对角线元素设为零，表示同一工序间的距离为零。最后，将处理后的工序间距离存储在距离矩阵D中。

**clean\_time(data)：**

正排情况下，初始化时间。遍历全部工单数据，提取每个工单的计划开始时间。将计划开始时间转换为特定的时间格式，便于比较和更新。如果当前的开始时间大于提取的时间或开始时间为0，则更新开始时间为提取的时间。

**clean\_time\_reverse(data)：**

用于根据工单的排程策略和计划完工时间，初始化倒排情况下的结束时间 end\_time。首先，初始化结束时间 end\_time，遍历 data["全部工单"] 中的每个工单，提取工单的排程策略和计划完工时间。如果工单的排程策略为倒排，则将 temp 标记为 True。然后，对于排程策略为倒排的工单，提取其计划完工时间并转换为数字格式，以方便比较大小。将计划完工时间转换为格式为 HHMM 的数字，例如 2359 表示晚上 11:59。将时间转换为数字格式后，与当前的 end\_time 比较，取较大值作为新的 end\_time。最后，输出信息，输出提示信息，表示开始时间初始化完成。返回计算得到的最终结束时间 end\_time。

**clean\_resources(data)：**

初始化资源列表及资源/时间列表。如果当前资源数量列表为空，遍历资源日历数据中的每个资源信息。提取资源名称、资源数量，并初始化资源数量列表和当前资源数量列表。解析资源的起始日期和结束日期，并计算资源可用时间段。将资源可用时间段添加到资源可用时间列表中。

**create\_resources\_need(data)：**

生成资源需求矩阵。如果资源需求矩阵为空，遍历工艺路线中的每个工序，提取资源需求信息。将每个工序的资源需求信息添加到资源需求矩阵中。

**clean\_locked\_work(data)：**

该函数的主要目的是根据给定的数据集初始化锁排程信息。对于每个锁排程信息，首先，提取锁排程名称和相关信息。染后，解析工序、开始时间和结束时间。最后，将每个锁排程的信息存储在locked\_work字典中，以锁排程名称作为键，工序、开始时间和结束时间作为值。

#### 7.1.3 资源管理函数说明

**create\_resources\_useful(current\_time, current\_resources\_count, resources, resources\_usetime)：**

生成当前可用资源列表。如果资源可用表为空，将资源可用状态初始化为True。遍历每个资源，根据资源的可用时间段和当前资源数量判断资源是否可用。如果资源在可用时间段内且资源数量大于0，则资源可用状态为True；否则为False。

**change\_taboo\_table(resources\_need,resources\_useful,current\_resources\_count)：**

根据资源需求、资源可用性和当前资源数量，结合前置条件、换型矩阵、锁排程等因素，调整禁忌搜索表 taboo\_table确保遵守资源约束和前置工序完成情况。首先，遍历所有工序，判断工序是否已选择，且不在禁忌搜索表中，若满足条件则加入禁忌搜索表。同时，检查工序的前置工序是否完成，若未完成，则将工序加入禁忌搜索表，验证当前资源是否满足工序所需资源，若不足则加入禁忌搜索表。然后，根据换型矩阵和锁排程等条件，判断是否需要加入禁忌表。最后，考虑正倒排情况，根据工序的正排情况，判断是否需要加入禁忌表。

**change\_taboo\_table\_reverse(resources\_need,resources\_useful,current\_resources\_count) ：**

根据资源需求、资源可用性和当前资源数量，结合前置条件、换型矩阵、锁排程等因素，调整倒排情况下的禁忌搜索表 taboo\_table\_reverse，以确保在倒排工艺调度和作业安排过程中遵守相关约束条件。首先，初始化变量，初始化 taboo\_table\_reverse 为空列表，用于存储禁忌的工序或工单。然后，判断禁忌条件，遍历 point\_table 中的每个工序或工单，依次进行禁忌条件判断。判断该工序是否被选择（chooced\_work[i]），如果是且尚未在倒排禁忌表中，则将其加入倒排禁忌表。检查该工序的后置工序是否完成（pre\_process[i]），若不满足则加入倒排禁忌表。检查该工序所需的资源是否在当前资源可用性范围内，若不足则加入倒排禁忌表。同时，处理换型矩阵和锁排程：根据换型矩阵和锁排程等条件，判断是否需要加入倒排禁忌表。最后，考虑正倒排情况：根据工序的倒排情况，判断是否需要加入倒排禁忌表。输出提示信息，表示倒排搜索禁忌表修改完成。

**set\_current\_resources\_count(data, city\_index)：**

根据工艺路线中的信息，遍历每个工序，找到与城市索引对应的工序。根据工序中的资源需求，更新当前资源数量列表。根据状态（state）决定是增加还是减少资源数量。

#### 7.1.4 时间和信息素管理函数说明

**change\_current\_time(current\_city, next\_city)：**

将输入的时间戳解析为年、月、日、时、分等时间单位。根据状态（state）来判断是进行时间的加法运算还是减法运算。根据时间间隔（time2）来更新时间，返回更新后的时间结果。

**subtime(time1, time2)：**

用于实现两个时间的相减操作，并返回相差的分钟数。

### 7.2 蚁群算法模块函数说明

#### 7.2.1 蚂蚁类模块函数说明

Ant 类通过选择下一个城市、计算路径距离、移动操作等方法，模拟了蚂蚁在蚁群算法中的行为，帮助蚁群寻找最优路径并优化工序执行顺序以最小化完成时间。

**\_\_init\_\_(self, ID) 方法：**

初始化函数，用于创建一个蚂蚁对象。接受一个ID参数来标识每只蚂蚁的唯一性。调用\_\_clean\_data()方法进行数据初始化。

**\_\_clean\_data(self) 方法：**

目的是在初始化蚂蚁对象时，对不同的数据进行清理和准备工作，以确保蚂蚁算法能够顺利执行。 首先，初始化当前时间current\_time，确保算法开始时具有准确的时间参考。初始化资源表，包括资源数量和可用时间段等信息。创建资源需求矩阵，记录每个工序对资源的需求。

同时，初始化工单/物料/工序结点表，记录工序的相关信息，change\_front\_back(data)、change\_post\_process(data)、change\_pre\_process(data) 函数，分别初始化正排和倒排情况下的前置工单表和前置工序表。然后，初始化时间需求和距离矩阵：调用 change\_time\_needs(data) 和 changeD(data) 函数，初始化时间需求和距离矩阵。并处理锁定工作，调用 clean\_locked\_work(data) 函数，处理锁定工作信息。并初始化资源列表和换型矩阵和可用资源列表和换型矩阵。最后，初始化禁忌搜索表和信息素矩阵，并初始化蚂蚁对象的属性：设置当前蚂蚁的路径、总距离、移动次数和当前停留的城市等属性。

**choice\_next\_city(self，taboo\_table) 方法：**

选择下一个城市的函数，根据信息素浓度和距离计算概率选择下一个城市。首先，计算每个未访问城市的选择概率，考虑信息素浓度和距离的影响。然后，根据概率进行轮盘赌选择，选择下一个城市作为移动目标。若概率总和大于0，则根据轮盘赌选择机制选择下一个城市；否则随机选择一个未访问城市作为下一个城市。最后，若所有城市都在禁忌搜索表中，则随机选择下一个城市。

**cal\_total\_distance(self) 方法：**

根据蚂蚁当前的路径计算总距离。遍历蚂蚁路径中的每个城市，计算当前城市到下一个城市的距离并累加到临时距离变量中。考虑路径是一个闭合回路，需要将最后一个城市到起点城市的距禂加到总距离中，确保计算包含完整路径的总距离。

**move(self, next\_city) 方法：**

移动操作函数，根据选择的下一个城市移动蚂蚁。更新蚂蚁的路径、总距离、当前城市等信息。首先，将下一个城市添加到蚂蚁的路径中。然后，根据当前城市和下一个城市之间的距离，更新蚂蚁的总距离。最后，将当前城市更新为下一个城市并增加移动次数计数。

#### 7.2.2 SSP类模块函数说明

实现了蚁群算法在解决SSP问题中的关键功能，包括蚂蚁路径选择、信息素更新、资源管理、工序调度等操作，以达到优化工序执行顺序和最小化完成时间的目的。通过迭代优化和信息素更新，蚁群算法能够寻找到最优的工序执行顺序，以提高工厂生产效率和资源利用率。

**\_\_init\_\_(self)方法：**

初始化函数，用于创建一个SSP对象。

**new(self)方法：**

初始化函数，用于创建新的蚁群算法实例。首先，创建包含指定数量蚂蚁对象的蚁群列表。然后，初始化最优解蚂蚁对象，设置初始最大距离作为初始最优解的总距离。再次，初始化信息素矩阵中每个元素的信息素浓度为1.0。最后，保存初始的信息素矩阵副本。

**search\_path\_front(self, epoch)方法：**

实现了正排情况下蚁群算法中的路径搜索过程。首先，初始化迭代过程，设定迭代次数为 epoch，每轮迭代中重新初始化一系列变量和数据结构。然后，迭代搜索路径，在每轮迭代中，重置工序完成状态并初始化当前时间、资源可用性、资源需求等信息。同时，根据当前资源和时间设置禁忌搜索表。遍历蚂蚁群中的每只蚂蚁，选择下一个工序并更新资源、时间等信息。根据选择的工序更新资源、时间、禁忌表等信息，直至满足条件或搜索完所有工序。更新信息素矩阵和禁忌搜索表。最后，更新信息素和结果，

根据最新的路径和完成时间更新信息素矩阵。获取最晚完成时间，并将其作为当前时间。将最佳完成时间和结果记录下来，并输出迭代次数和最佳完成时间。如果所有正排工序都已完成或达到迭代次数上限，结束当前轮迭代。将最佳完成时间和迭代次数记录在结果列表中。输出每轮迭代的迭代次数和最佳完成时间。

**search\_path\_reverse(self, epoch) 方法：**

用于执行倒排搜索路径的蚂蚁算法，并更新信息素矩阵。首先，初始化迭代过程，设定迭代次数为 epoch，每轮迭代中重新初始化一系列变量和数据结构，如果倒排工作数为0，直接结束迭代。然后，迭代搜索路径，在每轮迭代中，重置工序完成状态并初始化当前时间、资源可用性、资源需求等信息。同时，根据当前资源和时间设置禁忌搜索表。遍历蚂蚁群中的每只蚂蚁，选择下一个工序并更新资源、时间等信息。根据选择的工序更新资源、时间、禁忌表等信息，直至满足条件或搜索完所有工序。更新信息素矩阵和禁忌搜索表。最后，更新信息素和结果，根据最新的路径和完成时间更新信息素矩阵。获取最晚完成时间，并将其作为当前时间。将最佳完成时间和结果记录下来，并输出迭代次数和最佳完成时间。如果所有正排工序都已完成或达到迭代次数上限，结束当前轮迭代。将最佳完成时间和迭代次数记录在结果列表中。输出每轮迭代的迭代次数和最佳完成时间。

**\_\_update\_pheromone\_graph(self, old\_pheromone\_graph)方法：**

用于更新信息素矩阵。遍历每只蚂蚁在其路径上的相邻城市，根据路径总距离和信息素更新频率计算信息素的增加量。更新所有城市之间的信息素，包括旧信息素的蒸发和新迭代信息素的添加。

### 7.3 可视化模块函数说明

通过matplotlib库实现了甘特图和折线图的绘制，用于直观展示工序调度问题的解决过程和结果。甘特图展示了不同资源类型的任务执行情况，而折线图则反映了蚁群算法在不同迭代次数下的最佳完成时间情况。通过可视化，可以更直观地理解和分析算法的执行过程和结果。

绘制甘特图的函数 draw\_gantt\_chart(json\_file\_path)：首先读取JSON文件，解析数据并填充任务列表。然后，为每个工序创建子坐标，并根据不同资源类型分配颜色绘制甘特图，调整横坐标日期格式和设置刻度。再创建资源类型到y坐标的映射，绘制各个资源类型的任务甘特图。最后，展示甘特图。

绘制折线图的部分：在蚁群算法中记录每次迭代的最佳完成时间和迭代次数。然后，使用matplotlib库绘制折线图，横坐标为迭代次数，纵坐标为最佳完成时间。最后，设置图表标题和坐标轴标签，并显示绘制的折线图。

## 参数设置

迭代轮次n = 10

蚂蚁个数m = 30

信息素指数ALPHA = 1.5

可见度指数BETA = 2.0

挥发系数RHO = 0.9

## 总结和展望

通过持续的研究和优化，蚁群算法在工序调度和生产优化领域具有广阔的应用前景，可为实际生产过程带来更高效、更智能的解决方案。

### 9.1 工作成果

1. 完成了蚁群算法在SSP问题中的关键功能实现，包括路径选择、资源管理、信息素更新等操作。
2. 通过迭代优化和信息素更新，成功寻找到最佳的工序执行顺序，提高了生产效率和资源利用率。
3. 实现了多排程方向，支持正排和倒排两种方式。
4. 实现了订单锁单和换型矩阵功能。
5. 生成了JSON格式的最终工序执行计划，展示了最优路径和资源利用情况。
6. 绘制了折线图展示迭代次数与最佳完成时间的关系，以及甘特图展示工序执行情况和资源利用情况。

### 9.2 优化空间

1. 进一步优化算法参数设置，如信息素浓度指数、可见度指数和挥发系数，以提高算法的收敛速度和优化效果。
2. 考虑引入更复杂的资源约束和工时安排，以更全面地模拟实际生产环境。
3. 结合更多的实际生产数据和需求，不断优化算法，提高算法的健壮性和实用性。