U 型生产线辅助系统设计

摘要

U 型生产线的平衡研究一直是精益生产关注的重点之一，对于精益生产的效率提升有着极大作用，平衡效果差会导致工作效率低，在制品积压，中间存贮空间需求大等问题，而目前，在该方面尚未有可普及的进行 U 型生产线辅助开发的平衡计算软件。

本文使用两种算法进行 U 型生产线线的平衡进行研究：其一，使用标准的蚁群算法，采用随机规则和贪心规则作为内核对 U 型线进行平衡计算，在测试的 52 个实例组合当中，获得 18 个最优解，占比 34.62%，比历史最优解更优的解 7 个，占比 13.46%，算法表现出一定的竞争性，但是竞争性不是很强，由于内核使用贪心规则，很多时候和最优解的相差只有 1 个单位；其二，本文提出了一种积木模型算法，使用 ln 函数作为内核计算权重，当权重超过 1 时使用沿 ln(x)=1 直线的对称线的镜像结果，权重低于 0 时采用随机权重进行计算，在所测试的 102 个问题组合当中，求得最优解的数量为 72 个，占比为 70.59%，其中，取得比文献最优解更优的解得个数为 27 个，占比 26.47%，求解效果很好，具有很强的竞争性。

本文最后设计了一个轻量化的 UI 界面，将基本的文件操作与问题求解结合起来，集成为一个轻量化软件，可以在该软件当中直接进行 U 型生产线的平衡研究和辅助设计。

关键词： U 型生产线，一个流，蚁群算法，积木模型，软件设计

DESIGN OF AUXILIARY SYSTEM FOR U-TYPE PRODUCTION LINE

ABSTRACT

U-type line balance study has always been the focus of the lean production, one of the lean production efficiency has a great effect, balance will lead to low efficiency, poor quality products in the backlog, intermediate storage space needs big problems, and at present, there is no universal in the aspects of u-shaped production line auxiliary development balance calculation software.

This article uses two kinds of algorithms for U to study the production lines of the balance: first, use the standard ant colony algorithm, using random and greedy rules as the kernel of u-shaped line balance calculation, the test of 52 examples of combination, 18 for the optimal solution, accounted for 34.62%, better than the history of the optimal solution of the seven, accounting for more than 13.46, the algorithm showed certain competitive, competitive is not very strong, but because the kernel USES greedy rules, most of the time and the optimal solution is only one unit; Secondly, this paper proposes a lego model algorithm, ln function as the kernel is used to calculate the weight, when more than 1, use the ln (x) = 1 line of mirror symmetry line as a result, the weight is lower than 0 calculated, using random weighting of the tested 102 problems combination, the number of optimal solution is obtained for 72, 70.59%, among them, obtain the optimal solution more than literature optimal solution number 27, accounted for 26.47%, solving the effect is very good, has a strong competitive.

Finally, the paper designed a lightweight UI interface, combined basic file operations and problem solving, integration as a lightweight software, can be in the

midst of the software directly to U line balance study and aided design.

Key words: U - type production line, A flow ，Ant colony algorithm, Block model , Software design

目 录

第一章 绪论

1.1 选题背景

该问题来源于对东风汽车泵业公司的实际考察，该公司的模具铣削生产线是U型生产线，是该公司的工艺改善项目的重要成果，具有减小人力资源浪费，减少库存，节约空间，灵活性高等特点。

随着机械生产技术的发展，市场竞争的日益激烈，利润缩小，提高生产效率必要性日益显著，传统的直线型流水线生产线虽然在效率上提升不少，但是在应对小批量精细化生产时表现出灵活性差等缺点，容易形成物料的积压，而同时，现代自动化智能化生产技术的应用与提高，让一个工位可以同时完成多个工序，多个面的加工工作，结果是该工位的单工位加工时间延长，而此时如果仍按照传统的流水线方式在此配置工人很容易造成工人的有效工作时间的浪费，而传统直线型生产线若是将一个人同时分配给多个工位，每个人的工作时间工作强度差异将会很大，闲忙不均，在薪资分配和生产效率上面都是很大的隐患，不利于公司利润的进一步提高，人力资源、生产资源的有效合理配置。在现代化的产业竞争中会造成公司的竞争力下降。而 U 型生产线能够灵活解决各种精益化生产当中面临的问题，能够弹性的进行生产，有效的减少库存和积压，尤其是小批量产品的生产当中具有独到的优势。

1.2 选题意义

U型生产线能够有效解决各种实际生产中遇到的生产线产能弹性的问题，有效合理配置机器产能和人力资源，提高工业生产效率和生产能力，加强工业企业的技术和生产竞争力。

U型生产线的设计一向都是一项复杂的灵活性很高的工作，所安排的生产方案并不总是最优的，生产一线并不是长期都需要进行调度的，往往是一个新产品

的生产是才会进行新的调度，而新产品的生产准备期遇到的更多的是许多生产工艺和技术的问题，而不是调度问题，因此工业企业常常生产工艺和技术人员比较多，而车间调度人员比较少，且常有调度人员没有其他可交流同事或调度人员由技术工程师兼任的情况。没有专职的生产调度人员或生产调度人员示例薄弱的情况下，往往不能产生很好的调度效果，而外聘专业的调度专家进行调度设计则过于压缩企业的利润空间。

目前针对 U 性生产线的调度问题已经有许多方案，这些方案虽不能完全保证全局最优，但在其适应的范围内的解基本是最优，将这些及相关的解决防范集成到一个软件里面，可以有效的解决重复造轮子的问题，各公司的调度人员也方便迅速快捷的进行生产的调度，甚至是对于生产线人员和资源分配的实时最优调度。

1.3 问题的现状及相应解决办法

U 型生产线拥有以上种种优势，但是在实际的企业当中，研究课题式的进行 U 型生产线的探索却是不切实际的，而且周期过长，不利于迅速有效的提高企业的生产效率，优化企业的生产调度，产生一系列冗长的过渡期问题，导致生产效率不但没有得到优化，反而受到了干扰。

U 型生产（装配）线平衡是精益生产（Lean Production，LP）和准时制生产（Just-in-time，JIT）的重要内容，属于 NP 难问题，目前主要的解决方案有动态规划方法、各种启发式/元启发式算法，智能算法等。不同的算法在寻找最优解适应不同规模的问题。

当前在 U 型生产线的研究和应用方面，大多数的文献都集中于算法求解，而与生产直接结合例子较少，针对这一现象，故本文将集成部分的算法为一个软件，供生产一线的工程师使用。

第二章 基本理论及工具概述

2.1 基本理论概述

U 型生产线是相对于传统的直线型生产线而言的，

U 型生产（装配）线平衡属于 NP 难问题，通过建立数学模型，用精确算法一般只能够求解小规模问题，不适用于规模较大的问题，因此，较大规模的问题的求解通常使用近似求解，而在近似求解方法中，诸如禁忌搜索、模拟退火、遗传算法、蚁群算法、粒子群算法等人工智能或群智能算法又受到了广泛的关注。

本文主要使用蚁群算法和自己提出的积木模型算法，下面将做简要介绍。

蚁群算法：生物学家在长期观察蚂蚁后发现，蚂蚁的个体只能低下，但是相互之间的协作却很紧密，依靠群体的的协作发挥出了远胜于个体智能的能力，第一个蚁群算法 Ant System（AS）由 Colorni 等人提出于 1991-1992，并用于解决已知的旅行商问题（TSP）。由于由 Dorigo 等人在 1996 开发的算法无法与大规模 TSP 实例的最新算法竞争，因此刺激了该领域的进一步研究。

积木模型算法：积木模型算法以搭积木的思想，将每一个任务抽象为一块等底但不等高的积木进行搭建，并最后试图将这些积木搭建得竟可能一样高，或者高过某一个高度，或者不高于某一个高度，以此来实现求解，具体视实际问题情况而定。

2.2 Python 语言介绍

Python 是一种开源语言，受到市场的广泛关注。它结合了易用性和在多个平台上运行的功能，因为它是针对每个主要操作系统实现的。吉多 •范 •罗赞(Guido van Rossum)创造了这门语言，此后，Python 多年来不断变化，成为目前可用的最强大的编程语言之一。

Python 是一种很好的原型语言。在短短几分钟内，您就可以开发出可以在其他语言中使用几个小时的原型。它还体现了所有面向对象的概念，作为其核心引擎的一部分。因此，在 Python 中创建面向对象的编程应用程序比使用其他语言(如 Java 或 C++)要容易得多。

Python 是一个开源项目。因此，它是真正自由的。在其许可协议中没有涉及版权或版权。你可以改变它，修改它，赠送它，出售它，甚至免费分发它供商业使用。它的版权只保护作者不受法律问题的影响，如果有人因为使用 Python 而导致错误，或者其他人试图声称拥有该语言的所有权，可能会出现法律问题。

Python 是一种解释的、高级的编程语言、纯面向对象的、强大的 Web 服务器端脚本语言。与所有脚本语言一样，Python 代码类似于 pseudo 代码。它的语法规则和优雅的设计使它即使在多程序员开发团队中也是可读的。该语言没有提供丰富的语法，这是非常有用的。这背后的想法是让您考虑应用程序的业务规则，而不是花时间去弄清楚应该使用什么命令。引用 Guido van Rossum 的话说: “丰富的语法更像是一种负担，而不是一种帮助。”这也是真的，Python 是交互式的、可移植的、易于学习的、易于使用的，以及一种严肃的语言。此外，它还提供了动态语义和快速原型功能。Python 在很大程度上被称为连接现有组件的胶水语言。它在其他语言(C/ C++、Java 等等)的应用程序中是可嵌入的，而且还可以向 Python 中添加新模块，扩展其核心词汇表。Python 是一种非常稳定的语言，因为它在过去的数十年里一直在市场上，也因如此。

它的解释器和所有标准库都有它们的源代码和二进制文件。为每个人分配资源是一个很好的开发策略，因为它使来自世界各地的开发人员一起工作。任何人都可以向官方开发团队提交建议和补丁，由 Python 的 creator-Guido van Rossum 领导。Guido 是第二版脚本语言 ABC 语言的合著者，该语言主要用于 80 年代少数人的教学目的。Python 直接来自于 ABC。

2.3 库介绍

NumPy：NumPy 是 python 的高性能科学计算和数据分析的基础包，NumPy 的主要对象是通数据类型的多维数组，具有强大的数组、矩阵创建和运算功能，

提供大量的能对整个数组直接进行运算的数学运算公式。NumPy 是开源库，有许多共同开发者。

SciPy：SciPy 是用于科学计算和技术计算的开放源码 Python 库。SciPy 包含了优化、线性代数、集成、插值、特殊函数、FFT、信号和图像处理、ODE 求解器等在科学和工程中常见的任务模块。SciPy 构建在 NumPy 数组对象上，是 NumPy 栈的一部分，其中包括 Matplotlib，pandas 和 SymPy 等工具，以及一套扩展的科学计算库。这个 NumPy 堆栈与其他应用程序（如 MATLAB，GNU Octave 和 Scilab）具有相似的用户。 NumPy 堆栈有时也被称为 SciPy 堆栈。SciPy 也是这些工具的用户和开发者的会议系列：SciPy（在美国），EuroSciPy（在欧洲）和 SciPy.in（在印度）。Enthought 起源于美国的 SciPy 会议，并继续赞助许多国际会议以及主办 SciPy 网站。SciPy 库目前是在 BSD 许可下分发的，其开发由一个开放的开发社区赞助和支持。Numfocus 也支持它，这是一个支持可复制和可访问的科学的社区。

Pandas：pandas 是为 Python 编程语言编写的用于数据操作和分析的软件库。它提供了用于处理数字表格和时间序列的数据结构和操作。 它是根据三个条款的 BSD 许可证发布的免费软件。这个名字来源于“面板数据”一词，这是一个包含时间序列和横截面数据的计量经济学术语。

Matplotlib：matplotlib 是 Python 编程语言及其数值数学扩展 NumPy 的绘图库。 它提供了一个面向对象的 API，用于使用像 Tkinter，wxPython，Qt 或 GTK +这样的通用 GUI 工具包嵌入到应用程序中。还有一个基于状态机（如 OpenGL）的程序“pylab”接口，其设计与 MATLAB 非常相似。 SciPy 使用 matplotlib 库。matplotlib 最初是由 John D. Hunter 编写的，有一个活跃的开发社区，并且是以 BSD 格式的许可证发布的。

MoviePy：MoviePy 是一个用于视频编辑的 Python 模块，可用于基本操作（如剪切，连接，标题插入），视频合成（也称为非线性编辑），视频处理或创建高级效果。 它可以读写最常见的视频格式，包括 GIF。

PyQt：Qt 是一个跨平台的应用程序框架和小部件工具包，用于创建经典的嵌入式图形用户界面，以及在各种软硬件平台上运行的应用程序，在底层代码库中

很少或根本没有变化，而仍然是具有本地功能和速度的本地应用程序。Qt 目前正在由 Qt 公司、一家上市公司和 Qt 项目共同开发，包括个人开发人员和正在开发 Qt 的公司。Qt 可以使用专有和开源 GPL 2.0、GPL 3.0 和 LGPL 3.0 许可。而 PyQt 是跨平台 GUI 工具包 Qt 的 Python 绑定，实现为 Python 插件。

Pyinstaller：PyInstaller 是一个在 Windows，Linux，Mac OS X，FreeBSD，Solaris 和 AIX 下将 Python 程序冻结（打包）为独立可执行文件的程序。与类似的工具相比，它的主要优点是 PyInstaller 可以与 Python 2.7 和 3.3-3.6 协同工作，由于透明压缩而构建了更小的可执行文件，它完全是多平台的，使用 OS 支持来加载动态库，从而确保完全兼容。PyInstaller 的主要目标是与即开即用的第三方软件包兼容。这意味着，使用 PyInstaller，所有使外部软件包工作所需的技巧已经集成到 PyInstaller 本身，因此不需要用户干预。您永远不需要在维基中寻找技巧，并将自定义修改应用到您的文件或安装脚本。作为一个例子，像 PyQt，Django 或 matplotlib 库是完全支持，而不必手动处理插件或外部数据文件。

2.4 Windows 和 Ubuntu 系统简介

Windows：Microsoft Windows,是美国微软公司研发的一套操作系统，它问世于 1985 年，起初仅仅是 Microsoft-DOS 模拟环境，后续的系统版本由于微软不断的更新升级，不但易用，也慢慢的成为家家户户人们最喜爱的操作系统。Windows 采用了图形化模式 GUI，比起从前的 DOS 需要键入指令使用的方式更为人性化。随着电脑硬件和软件的不断升级，微软的 Windows 也在不断升级，从架构的 16 位、32 位再到 64 位， 系统版本从最初的 Windows 1.0 到大家熟知的 Windows 95、Windows 98、Windows ME、Windows 2000、Windows 2003、Windows XP、Windows Vista、Windows 7、Windows 8、Windows 8.1、Windows 10 和 Windows Server 服务器企业级操作系统，不断持续更新，微软一直在致力于 Windows 操作系统的开发和完善。是日常使用最多的电脑系统。

Ubuntu：是一个开源的操作系统的电脑。它是基于 Debian 架构的 Linux 发行版。它通常在个人计算机上运行，并且在网络服务器上也很流行，通常运行 Ubuntu 服务器的变体，具有企业级特性。Ubuntu 运行在最流行的架构上，包括

英特尔、AMD 和基于 arm 的机器。Ubuntu 也适用于平板电脑和智能手机，还有 Ubuntu Touch 版。Ubuntu 由 Canonical Ltd 发布，后者提供商业支持。[它是以自由软件为基础，以南部非洲的 ubuntu 哲学(字面意思是“人的”)命名，Canonical 公司建议将其大致翻译为“人类对他人”或“我就是我自己，因为我们都是这样的人”。Ubuntu 是在托管环境中运行的最流行的操作系统，称为“云”[5]，因为它是最流行的服务器 Linux 发行版。Ubuntu 的开发由总部位于英国的 Canonical 有限公司领导，该公司由南非企业家马克·沙特尔沃斯创立。Canonical 通过销售与 Ubuntu 相关的技术支持和其他服务来产生收入。Ubuntu 项目公开致力于开源软件开发的原则;人们被鼓励使用自由软件，研究它如何工作，改进它，并分享它。

第三章 算法设计与实现

3.1 算法的基本要素

算法的出发点和立足点在于解决实际工程问题，因此算法的输入输出量以实际工程效果为准，基本的输入量应该有工作站的（平均）工作时间，工人的（平均）距离行走时间，工作站之间的占地面积，工作站之间的距离等等。

根据应用情况不同有可以细分为 3 中情况，在接下来的 3 个晓杰里面进行描述。

3.1.1 设备安装阶段

该阶段的基本已知条件只有工作站（机床）的占地面积、工序的优先度和工人的平均行走时间，而各个工序的时间需要进行测试才能确定，因此，他的实际输入和输出量如表 3.1 所示。

表 3.1 设备安装阶段输入输出量

输入量	隐形输入量	输出量	干扰量
工作站（机床）面积	工人基本活动空间	总工作节拍（及波动范围）	旷勤、缺班
工序优先矩阵		工人和工作站的对应关系	工人个人因素的正态分布影响
工人平均各类时间（如取料、夹持、安装及行走等）		工作站安装图	
工作站工作时间		路线图	
		工人工作强度	
		平衡程度	

3.1.2 设备安装完成后

在设备安装完成后，由于工作站（机床）调整的成本等原因，一般不会进行工作站的大规模调整，只会进行小范围的调整，很多量的统计都已经比较准确，而且，在算法的设计当中不再涉及到工作站的安装位置设计等工作，而只是集中注意力于其他输出量的求解，一来可以节约大量的工作站安装设计时间，二来可以沿用之前的项目输入文档进行求解，，而且，此时不再需要进行工人基本活动空间的判断，可以大量的节约时间并降低算法复杂性。因此，设备安装安成后的输入和输出量如表 3.2 所示。

表 3.2 设备安装完成后输入输出量

输入量	沿用输入量	输出量	干扰量
工作站布局	工作站布局	总工作节拍（及波动范围）	旷勤、缺班
工序优先矩阵	工序优先矩阵	工人和工作站的对应关系	工人个人因素的正态分布影响
工人各类时间		工作站安装图	
工作站工作时间		路线图	
		工人工作强度	
		平衡程度	

3.1.3 安装完成后的更换产品

如上一项所述的成本原因，生产线一般定定型后除非大的项目外不会进行重新安装布置，灵活的小批量产品生产线尤其如此，当然，若要重新布置安装就是第一种情况，此时由于产品更换但是工作站及工作站位置没有更换，替换了工序优先矩阵及工人各类时间，同样具有一定的问题简化性并有效节约求解时间，其输入输出量如表 3.3 所示。

表 3.3 安装完成后的更换产品阶段的输入输出量

输入量	沿用输入量	输出量	干扰量
工作站布局	工作站布局	总工作节拍（及波动范围）	旷勤、缺班
工序优先矩阵		工人和工作站的对应关系	工人个人因素的正态分布影响
工人各类时间		工作站安装图	
工作站工作时间		路线图	
		工人工作强度	
		平衡程度	

关于工人各类时间，工人的各类时间包括工人取件、安装的时间，进行工件翻转安装的时间，中间行走的时间等，在实际进行建模的时候，有可能会将除行走时间外的时间都计算到工作站的工作时间里面，但是这样计算出来的分配结果工人的工作强度会偏高，也有可能将这部分时间连同行走时间一起计入工作时间，这样对于结果的计算会更加精确，对工人工作强度的评估也更加准确，对于调薪有较大帮助，只是统计上面会稍微复杂一点。

3.2 基本的结果评价方式

3.2.1 符号说明

基本符号说明如表 3.4 所示：

表 3.4 基本结果评价方式符号说明

1	$t_{machine}$	机床工作时间；
2	t_{worker}	工人工作时间；
3	$P_{machine}$	自动化导向型生产线平衡率；
4	P_{worker}	装配导向型生产线平衡率；
5	ST_{worker}	工人工作强度；

3.2.2 基本评价方式

关于结果，即上一节所言之输出量，应当包含基本的评价方式，在本文中，工人和机床的对应关系，机床和任务之间的分配关系直接由核心算法给出。

工作节拍视单个机床或单个工人的最大时间而定，具体因生产线的性质不同而不同，该不同指是否只有工人在进行该项任务的时候，该任务才在实际生产当中；若只有当工人对任务进行操作的时候任务才在生产，则属于装配导向型生产线，装配导向型生产线的的节拍为单个工人的最大工作时间；若工人只负责上下料和一些搬运，则属于自动化导向型生产线，自动化导向型生产线的节拍为单个机床的最大工作时间。

工作节拍的波动范围是指，工作节拍可能的最大值及可能的最小值的范围，关于这一部分将主要在本文第四章作为生产线效益评价的重要部分进行讨论。

生产线的平衡程度，本文采用平衡率进行表示，即：

$$P_{machine} = \frac{\sum t_{machine}}{\max(t_{machine}) \cdot n_{machine}} \tag{3.1}$$

和：

$$P_{worker} = \frac{\sum t_{worker}}{\max(t_{worker}) \cdot n_{worker}} \tag{3.2}$$

式(3.1)表示自动化导向型的工作平衡率，式(3.2)表示装配导向型的生产线平衡率。

工人工作的强度的基本计算虽然都是由工人的实际工作时间除以工作节拍，但由于工作节拍的计算方法分为两种，因此工人的工作强度的计算方法也分两种，如式(3.3)所示：

$$ST_{worker} = \begin{cases} \frac{t_{worker}}{\max(t_{worker})}, & \text{装配导向型} \\ \frac{t_{worker}}{\max(t_{machine})}, & \text{自动化导向型} \end{cases} \tag{3.3}$$

这里对自动化导向型中的生产线工作节拍为$\max(t_{machine})$，即最大机器时间做必要的说明，在自动化导向型的生产线上每一个工人负责一到多台机台，由于自动化导向型生产线的定义，工作主要由机器完成并且机器是可以同时进行工作的，工人的工作时间不大于所负责机台当中工作时间最长的一台机台，不然就会

造成生产线阻塞，在制品积压，而工人的工作时间为在其所负责的每一台机台上操作时间的总和，因此，在自动化导向型的生产线当中，生产线的节拍为机床最大工作时间。

3.3 蚁群算法的基本假设与说明

3.3.1 基本假设

根据生产线及装配线的实际生产特点，其工序受基准和前向工序的影响，因此为方便计算及贴近实际，在此做出假设如表 3.5 所示。

表 3.5 蚁群算法基本假设

编号	假设
1	工序有严格的优先顺序；
2	生产线上的任意工作站可执行任意工序；
3	工人路径可以交叉；
4	每一个（组）工人的各类时间不能高于其所负责的工作站工作总时间，考虑到没有库存的要求等情况，必要时可设工人的各类时间为 0，如自动化程度极高的情况。

3.3.2 符号说明

蚁群算法所使用的基本符号参考表 3.6。

表 3.6 蚁群算法符号说明

编号	符号	描述
1	i	任务编号；
2	j	机器编号；
3	k	蚂蚁编号；
4	m	蚂蚁总数；

5	n	遍历次数；
6	x	机床数量；
7	y	工人数量；
8	f_i	任务i 的前向任务集；
9	b_i	任务i 的后向任务集；
10	$w_i^f(k)$	任务i 的前向权重，定义为任务的任务时间；
11	$w_i^b(k)$	任务i 的后向权重，定义为所有其他任务的任务时间；
12	η_i	启发值，贪婪值；
13	T_{ij}	轨迹积累，表示将任务i 分配给机床j 的需求；
14	t_i	第i 个任务的任务时间；
15	$allowed_k$	可以分配给当前机床的任务集；
16	$p_{ij}^k(t)$	对于蚂蚁k ，任务i 分配到j 的概率；
17	E	已分配前向工序；
18	t_{allow_max}	进行任务分配时，允许的单个任务最大工作时间，或单个工人最大的机床工作时间；
19	P_{allow_mean}	进行任务分配时，根据输入条件，每个机床平均工作时间或每个工人平均机器时间；
20	ג	进行任务分配时，允许最大时间和平均时间之间的倍率；
21	t_{allow_min}	进行任务分配时的允许单个机床最少任务时间或者，单个工人，最少的机床工作时间；

3.4 蚁群算法基本模型建立

此部分的核心算法蚁群算法主体设计采用 Ihsan Sabuncuoglu 教授等人提出的蚁群算法，后面会根据实际的任务需要进行改进。

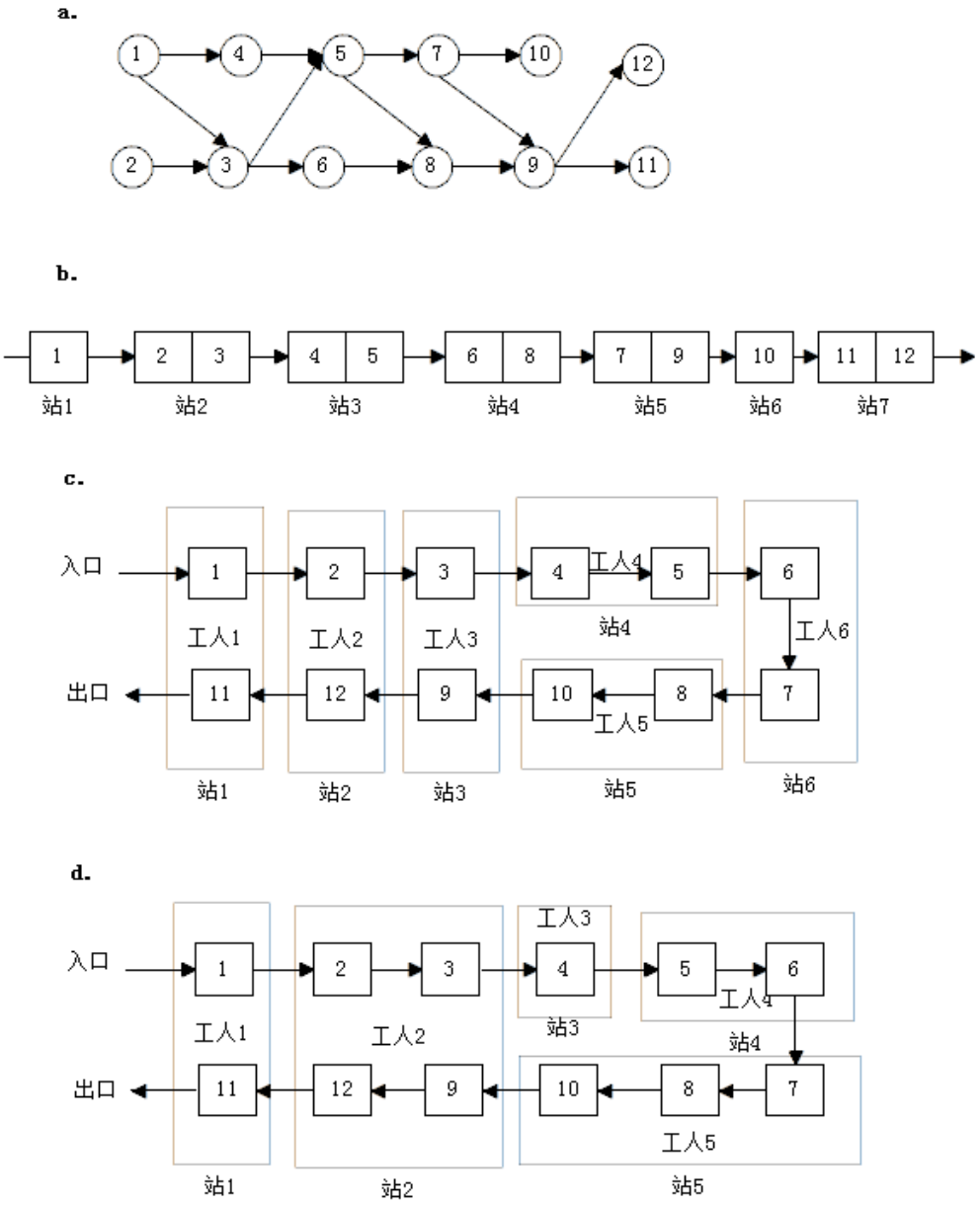


图 3. 1 示例:(a)优先级图示例（School and Klein,1999）；(b)a 的直线配置；（c）a 的 U 型配置；(d)a 的可能配置

在 Ihsan Sabuncuoglu 教授等人提出的算法当中，U 型线的布局和设置如下图 c 所示，由于其是装配线，主要工作都是依靠工人的手动操作来完成的，此种布局及其分配模式和节拍才能被接受，而实际上的生产可能是零件产线而不是装配线，主要的工作可能是由机器来完成，也就是说零件生产线和装配线的时间在这

方面有着质的区别，并且工作时可能不是恰好这么巧的是一个人两台机器，也可能是三台或四台甚至是更多，也可能是一台，如下图 d 所示，而且，在实际的车间生产中，一部机床在生产流程中可能并不止完成一道工序，而是能够完成几道相近或类似的工序，而自动化的发展，使得工人几乎只需要完成工件的装卸和机床的启动工作，因此，本文采用的模型将不只是任务和工作站的二元分配关系，而是三元的“任务—>机床—>工人”分配关系，如图 2 所示。先将任务均衡的分配给各个机床，然后在将各个机床均衡的分配给工人，以达到生产线平衡的目的。

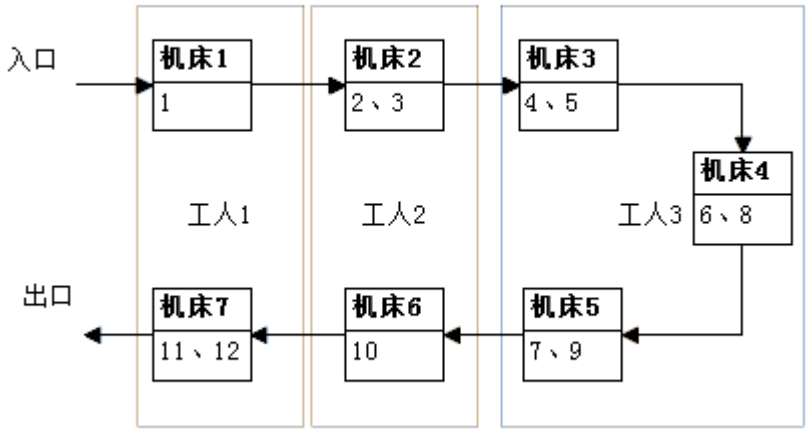


图 3.2 “任务—>机床—>工人”分配示例

介于生产上的特殊性，工序都有其优先关系，采用优先矩阵表示其优先关系，矩阵的第一行表示前向工序，第二行表示后向工序，即$\begin{bmatrix} 0_1, & 0_2, & \ldots, & 0_i, & \ldots \\ 1_1, & 1_2, & \ldots, & 1_i, & \ldots \end{bmatrix}$，因此，在进行当前站任务分配的时候，由入口向出口方向配置时，任何工序必须其前向工序已经分配其才能够分配，反之则为其后向任务已分配其才可以分配。使用E表示已经分配的任务，允许集的获得公式为式(3.4)。

$$allowed_k = \begin{cases} \{i|f_i \in \emptyset \cup f_i \in E\}, \text{直线前向分配模式} \\ \{i|b_i \in \emptyset \cup b_i \in E\}, \text{直线后向分配模式} \\ \{i|f_i \in \emptyset \cup f_i \in E \cup b_i \in \emptyset \cup b_i \in E\}, \text{U 型分配模式} \end{cases} \tag{3.4}$$

在生产优先关系的基础上，再添加每一个任务的权重，每一个任务的权重分为两个部分，前向权重$W^f(k)$和后向权重$W^b(k)$，分别定义为任务时间和所有其

他任务时间。然后是η_i，表示启发值，贪婪值，在蚁群算法的基本模型当中，η_i的定义如式 3(3.5)所示。

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{3.5}$$

其中η_{ij}为启发函数，d_{ij}表示两个城市之间的距离，距离越大则启发值越小，因此，在该问题中，将启发值与权重进行挂钩，权重越大，则启发值越大，进行严格的优先顺序筛选与分配，定义为式(3.6)。

$$\eta_i = \begin{cases} \frac{w_i^f(k)}{\max(w_z^f(k))}, & i \in allowed_k \\ 0, & \text{其他情况} \end{cases} \tag{3.6}$$

至此，我们可以定义对于蚂蚁k 将任务i 分配给j 站的一般概率公式(3.7)。

$$p_{ij}^k(t) = \begin{cases} \frac{[T_{ij}(t)]^\alpha [\eta_i]^\beta}{\sum_{z \in allowed_k} [T_{zj}(t)]^\alpha [\eta_z]^\beta}, & i \in allowed_k \\ 0, & \text{其他情况} \end{cases} \tag{3.7}$$

这里要注意，虽然前面已经将η_i分出一种为 0 的情况了，但是这里分子不能为 0，还是将这种情况提出来进行计算，以免在计算中出现意外的值。

接下来，便是关于T_{ij}的初始化和更新方式，常规而言在任意t 时刻，每条路径上的信息素更新规则如式(3.8)和式(3.9)所示。

$$T_{ij}(t) = (1-\rho)T_{ij}(t-1) + \Delta T_{ij}^k(t) \tag{3.8}$$

$$\Delta T_{ij}(t) = \sum_{k=1}^{m} \Delta T_{ij}^k(t) \tag{3.9}$$

这里ρ是信息挥发系数，$0 \le \rho \le 1$，$\Delta T_{ij}(t)$每次有蚂蚁选择该路径时在路径上新添的信息素浓度，在各个版本中基本都与路径长度有关，比如：

在 Ant Cycle 模型（以下简称 AC 模型）中为式(3.10)：

$$\Delta T_{ij}^k(t) = \begin{cases} \frac{Q}{L_k}, & \text{如果蚂蚁} k \text{ 在本次遍历中经过} (i,j) \\ 0, & \text{其他} \end{cases} \tag{3.10}$$

其中 Q 表示信息素强度，为初始需要设置的参数，在其他分支模型中表示相

同含义，L_k为蚂蚁k 在本次遍历当中经过的路程（含(i,j)）。

在 Ant-Quantity 模型（以下简称 AQ 模型）中为式(3.11)：

$$\Delta T_{ij}^{k}(t)=\begin{cases}\frac{Q}{d_{ij}}, & \text{如果蚂蚁}k\ \text{在本次遍历中经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{3.11}$$

这里所有参数都已经叙述过，就不在赘述。

在 Ant-Density（以下简称 AD 模型）中为式(3.12)：

$$\Delta T_{ij}^{k}(t)=\begin{cases}Q, & \text{如果蚂蚁}k\ \text{在本次遍历中经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{3.12}$$

该模型的信息素受距离，或者说权重影响更小，收敛速度会更慢。

因此，在我们的实际问题当中，不存在任务到机站的距离因素，但存在任务的完成时间因素，该因素，直接与权重挂钩，也即各基本蚁群当中的距离的倒数，因此，在本文中，基本的信息素累积量将定义为仿 AQ 模型模式，如下式(3.13)：

$$\Delta T_{ij}^{k}(t)=\begin{cases}Q\cdot\frac{w_i^f(k)}{\max(w_z^f(k))}, & \text{如果蚂蚁}k\ \text{在本次遍历中经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{3.13}$$

另有仿 AC 模式如下式(3.14)：

$$\Delta T_{ij}^{k}(t)=\begin{cases}\frac{Q}{L_k}, & \text{如果蚂蚁}k\ \text{在本次遍历中经过}(i,j)\\ 0, & \text{其他}\end{cases} \tag{3.14}$$

式(3.14)中L_k为式(3.15)给出：

$$L_k=\frac{\sum w^f(k)}{\sum w_i^f(k)},\ \text{表示已走路径在总路径的百分比} \tag{3.15}$$

然后，由于我们的简化，并不能像实际生活当中那样完整的体会到同时的概念，都转换成了迭代次数的问题，但是，失去了这个时间概念以后，每一次迭代时候是谁先到达下一个任务的呢？由于每次迭代每只蚂蚁都是直接将任务i分配给机台j，失去了时间上的由于距离长短的到达次序先后的的概念，也就是，到底谁先到达，谁先在路径上留下信息素，为了完成这个模拟，我们做了如下的定义：由于在本实际问题中，将任务分配给机站没有距离一说，但是考虑到和基本

蚁群算法对比时，启发值η_i^k将距离d_{ij}与任务的前向权重$w_i^f(k)$进行了挂钩，因此，除初始化时所有蚂蚁可以理解为同时出发外，其余任何时候，都由已经走过前向权重最大路径$\sum w_i^f(k)$的蚂蚁优先更新信息素，以体现路径短能给其他蚂蚁参考的实际意义。基本流程如图 3.3 所示。

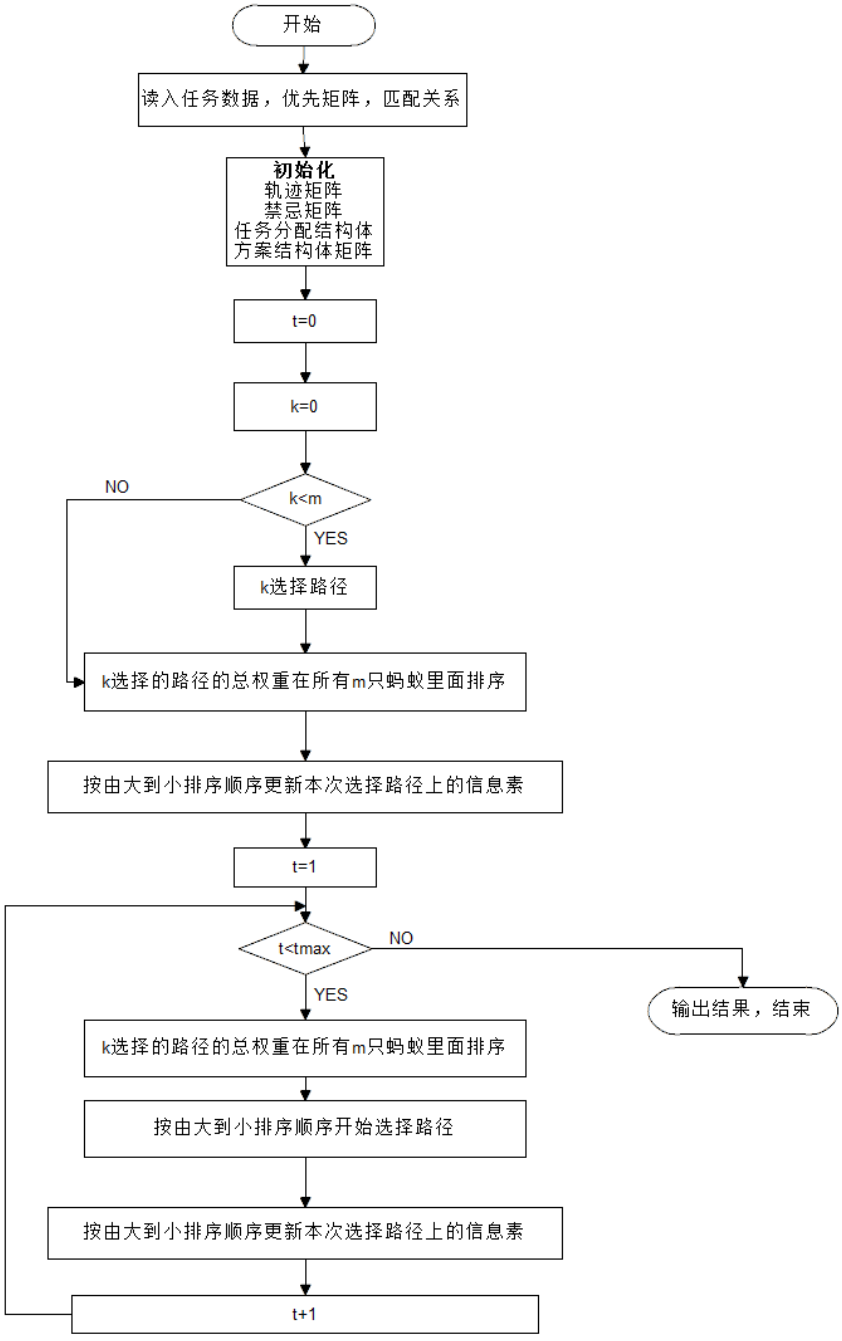


图 3.3 本问题蚁群算法流程

在基本的结果评价方式一节本文有提到，生产线的平衡程度，采用平衡率进行计算，，该平衡率即是本文重点优化的目标之一，为了把平衡率的优化更好的贯穿于整个求解的过程，在给定基本参数蚂蚁数量m，机床数量x，工人数量y和任务集的时候就应该进行基本的限制了，本文采用的限制方法是，在第一阶段，将任务集分配给机床集的时候，限制单个机床的任务时间最大值为平均值的诸如1.1 倍，而，只要某机床的任务时间没有超过最大任务的时间就可以对其进行任务分配，以求将平衡率一开始就控制在一定的范围以内，概括为平均值、允许的最大值和最小值的计算公式分别为(式(3.16)、式(3.17)和式(3.1 所示)：

$$t_{allow\_mean} = \frac{\sum t_i}{n} \tag{3.16}$$

$$t_{allow\_max} = \lambda P_{allow\_mean} \tag{3.17}$$

$$t_{allow\_min} = \begin{cases} \max(t_i)， & t_{allow\_mean} > \max(t_i) \\ \min(t_i)， & t_{allow\_mean} < \max(t_i) \end{cases} \tag{3.18}$$

需要特别说明的是，此处的t_{allow_min}并不一定是实际最小值下限，只是在进行任务分配的时候进行一定的判断，减小分配不均的概率，t_{allow_max}同理，λ为倍率关系，可以自行设定。

3.5 蚁群算法的变种

3.5.1 随机扰动版本

随机扰动版本的产生主要是针对在生产线当中，距离被定为了任务权重的倒数，即任务时间的，而任务时间在数据集当中是已经确定了的，因此，上一节中蚁群算法在该问题中的表现将会永远都是优先分配权重最大的任务，若某一个机器编满了则分配到下一个机器，这样的效果不是最优的，因此，在此基础上添加随机模型，随机模型的基本思想是，在去顶任务权重和确定轨迹值的情况下，每一只蚂蚁仍然具有一定的可能性去选择权重和轨迹都比较小的任务给当前机床，增加随机性扰动带来最优解的可能性。

即，在随机扰动模型当中，基本的选择概率公式如式(3.14)：

$$p_{ij}^{k}(t)=\begin{cases} rand(1)\dfrac{[T_{ij}(t)]^{\alpha}[\eta_i]^{\beta}}{\sum_{z\,\epsilon\,allowed_k}[T_{zj}(t)]^{\alpha}[\eta_z]^{\beta}}, & i\,\epsilon\,allowed_k \\ 0, & \text{其他情况} \end{cases} \tag{3.19}$$

3.6 蚁群算法的结果

本文使用 Scholl(1993)的数据集[16]，其范围为 25 -297 个任务的 168 个平衡问题实例。

在前两次结果相同的时候，不再进行第三次求解，在前两次结果不同的时候，至少进行十次求解，求解结果即对比在表 3.7 中。

表 3. 7 蚁群算法求解结果

问题	m	总时间	理想节拍	文献最优节拍	平均节拍	均方差	最优节拍
Buxey	7	324	46	47	48	0	48
Buxey	8		41	41	42	0	42
Buxey	10		32	34	35	0	35
Buxey	13		25	27	28	0	28
Gunther	6	483	81	84	84	1.154701	83
Gunther	8		60	63	62.85714	0.690066	62
Gunther	12		40	44	45.125	0.64087	44
Gunther	13		37	42	43	0	43
Kilbridge	3	552	184	184	184	0	184
Kilbridge	5		110	111	111	0	111
Kilbridge	7		79	79	81	0	81
Kilbridge	10		55	56	58	0	58
Lutz1	8	14140	1768	1860	1854.667	4.844241	1846
Lutz1	10		1414	1526	1478.857	11.53875	1460
Lutz1	11		1285	1400	1400	0	1400
Lutz2	9	485	54	54	55	0	55
Lutz2	10		49	49	49	0	49
Lutz2	13		37	38	38.33333	0.57735	38

Lutz2	15		32	33	34	0	34
Lutz2	17		29	29	30	0	30
Lutz2	20		24	25	26	0	26
Lutz2	26		19	19	20	0	20
Sawyer	7	324	46	47	48	0	48
Sawyer	9		36	37	38	0	38
Sawyer	12		27	28	29	0	29
Tonge	7		501	502	507.5	2.12132	506
Tonge	9		390	391	397	0	397
Tonge	10		351	352	362	0.816497	361
Arcus1	3	75707	25236	25236	25249.33	3.05505	25246
Arcus1	4		18927	18927	18965.33	8.504901	18957
Arcus1	6		12618	12620	12689.33	5.507571	12683
Arcus1	10		7571	7580	7679	23.51595	7663
Arcus1	20		3785	3882	4046	0	4046
Hahn	3	14026	4675	4787	4682.6	2.701851	4679
Hahn	4		3507	3677	3527.6	10.31019	3516
Hahn	8		1753	1907	1872.25	46.38516	1805
Hahn	10		1403	1775	1775	0	1775
Warnecke	3	1548	516	516	516.25	0.5	516
Warnecke	4		387	387	388	0	388
Warnecke	10		155	155	160.1667	0.752773	159
Warnecke	20		77	79	86	0	86
Warnecke	21		74	76	84	0	84
Lutz3	3	1644	548	548	548	0	548
Lutz3	4		411	411	412	0	412
Lutz3	10		164	165	167	0.57735	166
Lutz3	15		110	110	117	4.358899	114
Mukherje	3	4208	1403	1403	1403.25	0.5	1403

Mukherje	6	701	704	704	0	704
Mukherje	10	421	424	432.3333	1.154701	431
Mukherje	18	234	239	250	1	249
Mukherje	21	200	208	218	0	218
Barthold	6	939	939	944.5	0.707107	944

在 52 个实例组合中，求解到最优解 18 个，占比 34.62%，比历史最优解更优的解 7 个，算法表现出一定的竞争性，但是竞争性不是很强，由于内核使用贪心规则，很多时候和最优解的相差只有 1 个单位。

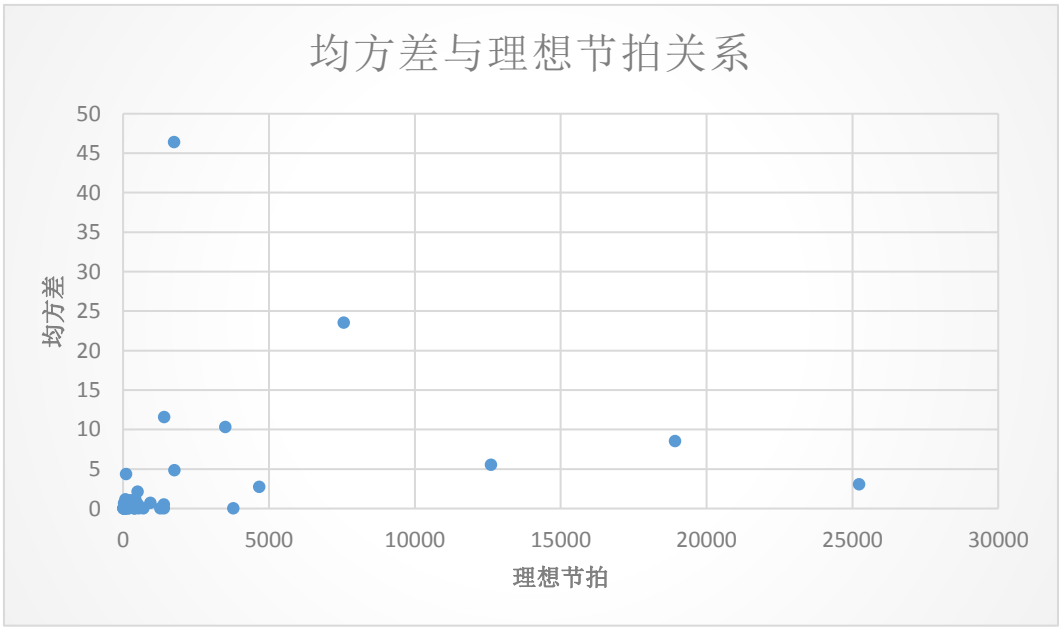
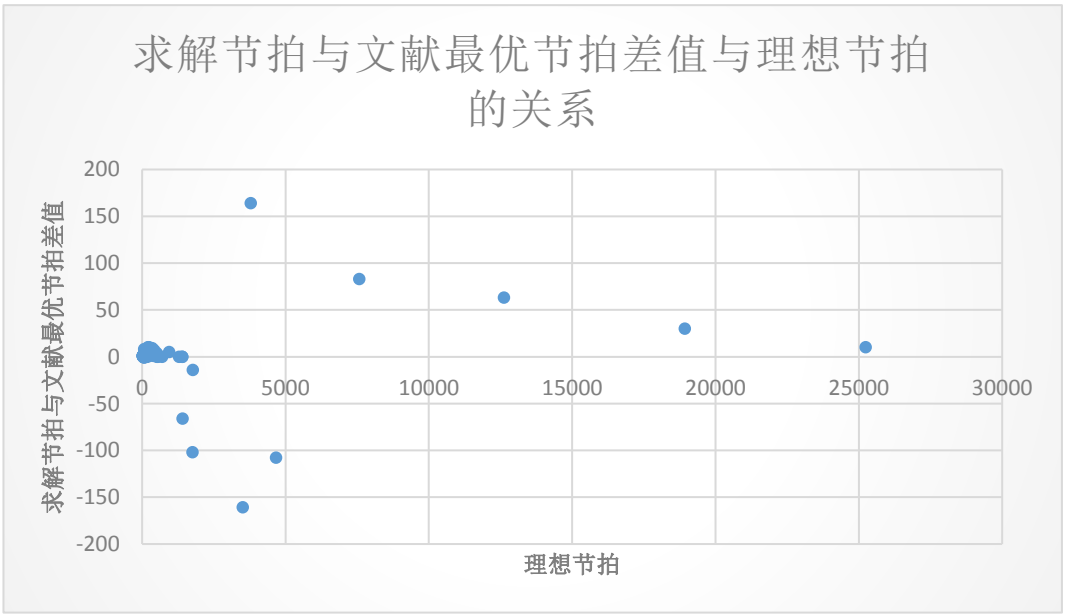


图 3.4 蚁群算法求解结果均方差与理想节拍关系



图

3.5 差值与文献最优节拍关系

3.7 算法的不足

蚁群算法当前还存在着收敛慢，寻优能力稍差的问题，在和最优解相差一个单位的情况表现出明显的该特点。

3.8 积木模型的基本假设与说明

3.8.1 基本假设

积木的底面积相等、底面形状相同，区别只在于高的不同产生的体积不同。其余与蚁群算法的基本假设一致。

3.8.2 符号说明

积木模型基本符号说明如表 3.8 所示。

表 3.8 积木模型符号说明

编号	符号	描述
1	i	小积木块编号，即进行任务分配时候的任务编号，进行机器分配时候的机器编号；
2	j	积木槽编号，进行任务分配时候的机器编号或工人编号，进行机器分配时候的工人编号；
3	t_{mean}	理论上最优的各个积木槽的高度，即平均高度；
4	t_i	小积木块的高度，即，进行任务分配时候的任务时间，进行机器分配时候的机器时间；
5	$allowed_k$	当前允许分配的任务集，简称允许集；
6	$\max\_t_j$	当前积木槽允许的高度上界；
7	P_{ij}	将小积木块i分配给积木槽j的基本概率；

3.9 积木模型算法的基本建模

积木模型的思想来源于儿童玩的乐高积木，当所拥有的积木的为底面积都相等，底面形状都相同的立方体积木的时候，积木所搭出来的模型的高度只受到各个积木的长度的影响，需要搭建整体高度相似的积木的时候，儿童会仔细挑选竟可能能够达到预期高度的小积木块，而不是类似蚁群算法在求解这类问题的时候所采取的倾向于贪婪算法的取权重最大的任务，而是，在选取任何一块积木的时候，有可能是随机的，但是，当该积木可选的时候，会优先判断该积木是否符合预期要搭的积木的高度，若越是接近于预期的高度，则选择该积木块的概率就越大，而若该积木块若搭上去之后明显不符合预期高度或略高于预期高度，则会进行相关调整，以期达到最好的预期高度范围。

而我们进行任务的分配，其目的为使每一个机器所接受的任务总时间尽可能的接近均值，即，将每一个积木槽所搭积木的高度尽可能的接近理论均值高度。

以下为具体定义，首先，取当前所有被允许使用的积木进行备用，即取当前任务的允许选取集$allowed_k$，为保证全文叙述一致，以下统称为允许集，其意义与蚁群算法当中一致，取得允许集后，进行第一块小积木的选取，该小块的选取可以为随机也可以为权重，但此处使用权重的方法与蚁群算法当中有所差异，将任务i分配给基本 i选取概率定义为式(3.20)：

$$P_{ij}=\begin{cases} rand(1), & P_{ij}\le 0 \quad \text{随机选择} \\ 1+\ln\left|\frac{t_{mean}-t_j-t_i}{t_{mean}}\right|, & \text{根据理论均值进行选择} \end{cases} \tag{3.20}$$

其中，t_{mean}表示理论上每一个积木槽里积木的最好高度，即每一个机器分得任务的理论时间均值，t_j表示当前积木槽j 已经分得任务的总时间，t_i表示小积木块的高度，即当前任务的时间。

进行小积木块分配的时候，首先按照理基本概率公式选取，当首次迭代完成后，更新可进行分配的上界$\max_t_j$，此时的根据理论均值选取同样是越接近与理论均值选取概率越大，但是当大于理论均值的时候，越接近于上界选取概率越小，定义为式(3.21)：

$$P_{ij}=\begin{cases}1+\ln\frac{t_j+t_i}{t_{mean}}, & (t_j+t_i)\le t_{mean}\\ 1-\frac{t_j+t_i-t_{mean}}{\max\_t_j-t_{mean}}, & t_{mean}\le (t_j+t_i)\le \max\_t_j\end{cases} \tag{3.21}$$

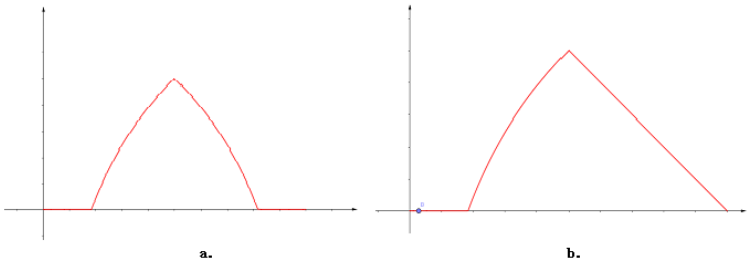


图 3. 6 积木模型选取概率示意，(a),原生选择概率，(b)有上界后的选择概率

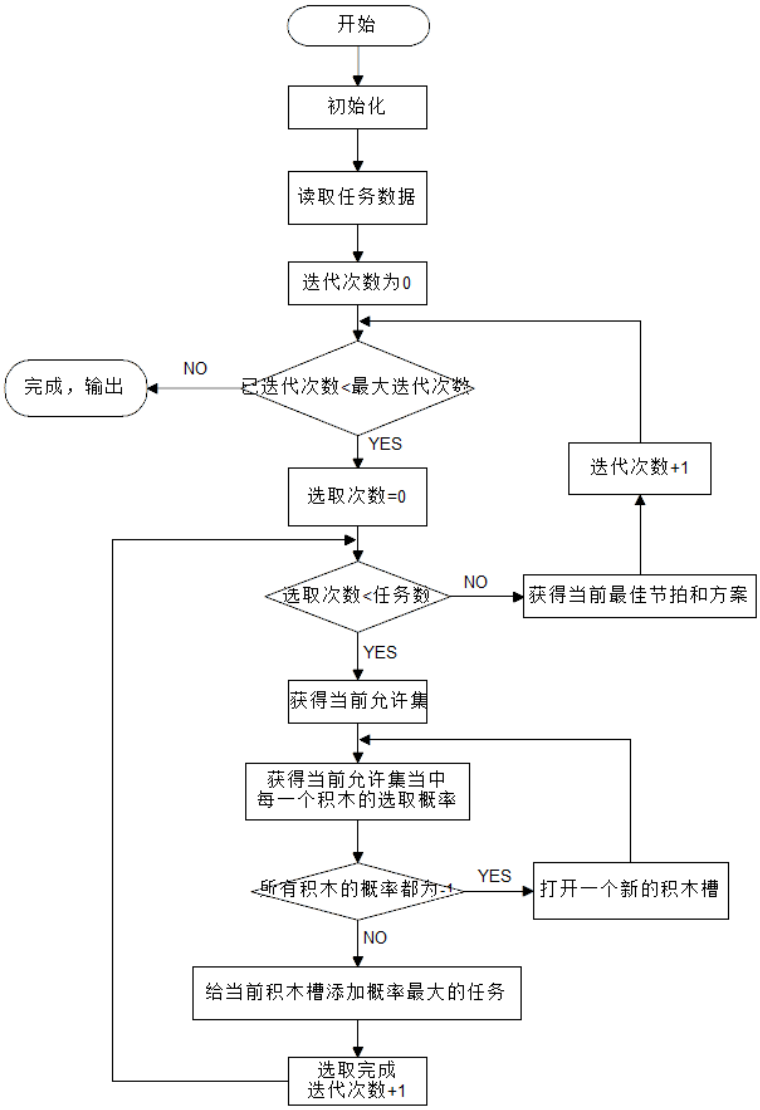


图 3. 7 积木模型流程图

选取概率示意如图 3.6 所示。当所求解在迭代当中已经比较固定的时候，使用随机捆绑的方法，将两个或更多允许集当中的小积木当做是一个来进行选取，以期获得更好的分配结果。积木模型整体流程如图 3.7 所示。

3.10 积木模型的结果

原始数据来源与蚁群算法数据来源相同，求解结果及对比字表 3.9 中。

在使用积木模型求解的 14 个 U 型问题的 102 种组合中，求得最优解的数量为 72 个，占比为 70.59%，其中，取得比文献最优解更优的解得个数为 27 个，占比 26.47%，求解效果很好。

但是，求解当中存在一定的短板，对于中等规模问题（任务数大于 30）求解时易出现所得解在最优解及上方一点点波动的情况，对于大规模问题（任务数大于 70）且总时间较大，分到单个机器的时间也较大（时间超过 1500）的时候，求解性能下降很快，可参考均方差与理想节拍之间关系。

表 3.9 积木模型求解结果表

问题	m	总时间	理想节拍	文献最优节拍	平均节拍	均方差	最优节拍
Buxey	7	324	47	47	47	0	47
Buxey	8		41	41	42	0	42
Buxey	9		36	37	37	0	37
Buxey	10		32	34	33	0	33
Buxey	11		29	32	31	0	31
Buxey	12		27	28	28	0	28
Buxey	13		25	27	25.90909	0.301511	25
Buxey	14		23	25	25	0	25
Gunther	6	483	81	84	81	0	81
Gunther	7		69	72	70	0	70
Gunther	8		60	63	61	0	61
Gunther	9		54	54	54.83333	0.389249	54
Gunther	10		48	50	49.33333	0.5	49
Gunther	11		44	48	45	0	45
Gunther	12		40	44	41.57143	0.534522	41

Gunther	13		37	42	40	0	40
Kilbridge	3	552	184	184	184	0	184
Kilbridge	4		138	138	138	0	138
Kilbridge	5		110	111	111	0	111
Kilbridge	6		92	92	93	0	93
Kilbridge	7		79	79	79	0	79
Kilbridge	8		69	69	70	0	70
Kilbridge	9		61	62	62	0	62
Kilbridge	10		55	56	56	0	56
Kilbridge	11		50	55	55	0	55
Lutz1	8	14140	1768	1860	1854	0	1854
Lutz1	9		1571	1638	1632	7.483315	1624
Lutz1	10		1414	1526	1518	0	1518
Lutz1	11		1285	1400	1400	0	1400
Lutz1	12		1178	1400	1400	0	1400
Lutz2	9	485	54	54	54	0	54
Lutz2	10		49	49	49	0	49
Lutz2	11		44	45	45	0	45
Lutz2	12		40	41	41	0	41
Lutz2	13		37	38	38	0	38
Lutz2	14		35	35	35	0	35
Sawyer	7	324	47	47	47	0	47
Sawyer	8		41	41	41	0	41
Tonge	3	3510	1170	1170	1170.25	0.5	1170
Tonge	4		878	878	878	0	878
Tonge	5		702	702	703	0	703
Tonge	6		585	585	586	0	586
Tonge	7		501	502	503	0	503
Tonge	8		439	439	440	0	440
Tonge	9		390	391	392	0	392
Tonge	10		351	352	352.25	0.5	352
Tonge	11		319	320	321	0	321
Tonge	12		293	294	294	0	294
Arcus1	3	75707	25236	25236	25240	0	25240
Arcus1	4		18927	18927	18958.33	16.74316	18939
Arcus1	10		7571	7580	7668	0	7668
Arcus1	20		3785	3882	3918.333	2.309401	3917
Arcus1	21		3605	3691	3710.667	0.57735	3710
Hahn	3	14026	4675	4787	4703	0	4703
Hahn	4		3507	3677	3547	0	3547
Hahn	7		2004	2336	2056	0	2056
Hahn	8		1753	1907	1792	0	1792
Hahn	9		1558	1827	1775	0	1775

Hahn	10		1403	1775	1775	0	1775
Warnecke	3	1548	516	516	517	0	517
Warnecke	4		387	387	387	0	387
Warnecke	5		310	310	310	0	310
Warnecke	6		258	258	259	0	259
Warnecke	7		221	222	222	0	222
Warnecke	10		155	155	156	0	156
Warnecke	20		77	79	80	0	80
Warnecke	21		74	76	76	0	76
Lutz3	3	1644	548	548	548	0	548
Lutz3	4		411	411	411	0	411
Lutz3	5		329	329	329	0	329
Lutz3	10		164	165	165	0	165
Lutz3	15		110	110	111	0	111
Lutz3	16		103	105	104	0	104
Mukherje	3	4208	1403	1403	1403	0	1403
Mukherje	4		1052	1052	1053	0	1053
Mukherje	6		701	704	702.3333	0.57735	702
Mukherje	8		526	532	528	0	528
Mukherje	10		421	424	422.1111	0.600925	421
Mukherje	11		383	391	385	0	385
Mukherje	12		351	358	354.25	0.5	354
Mukherje	18		234	239	237	0	237
Mukherje	19		221	226	224	0	224
Mukherje	20		210	220	212	0	212
Mukherje	21		200	208	203	0	203
Barthold	3	5634	1878	1878	1878	0	1878
Barthold	6		939	939	940	0	940
Barthold	8		704	705	705	0	705
Barthold	10		563	564	564	0	564
Barthold	15		376	383	383	0	383
Barthol2	27	4234	157	157	158	0	158
Barthol2	28		151	152	152	0	152
Barthol2	29		146	146	147	0	147
Barthol2	30		141	142	142	0	142
Barthol2	39		109	109	109	0	109
Barthol2	40		106	106	106.75	0.5	106
Barthol2	46		92	93	93	0	93
Scholl	25	69655	2786	2787	2797	2.54951	2795
Scholl	30		2322	2322	2329.25	0.5	2329
Scholl	35		1990	1991	1997	0	1997
Scholl	40		1741	1742	1749.667	0.57735	1749
Scholl	50		1393	1394	1401.667	0.57735	1401

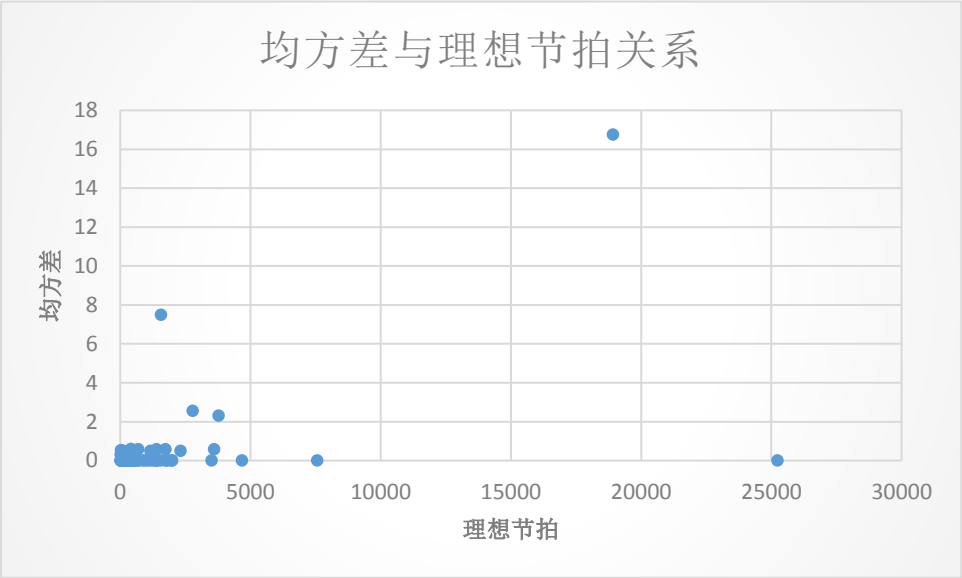


图 3.8 均方差与理想节拍关系

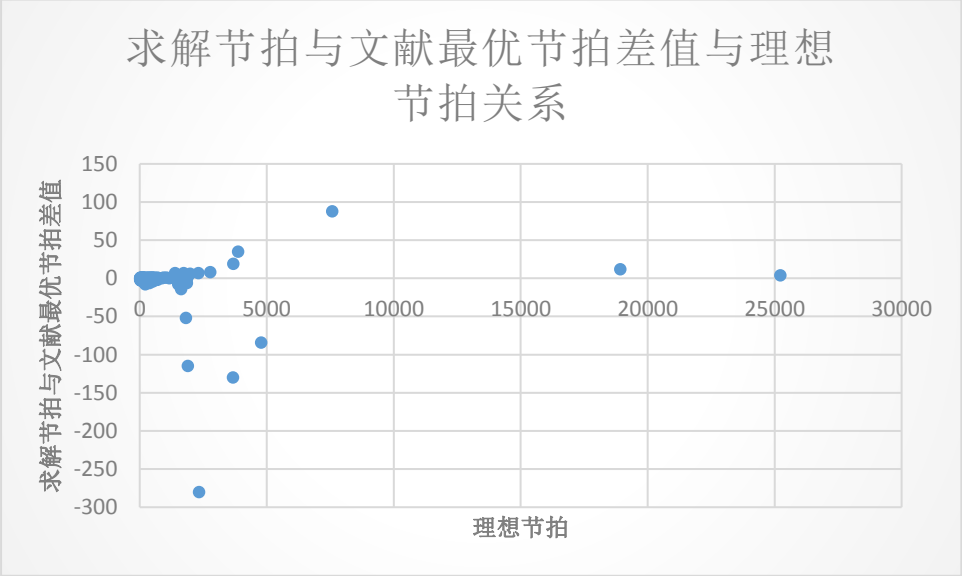


图 3.9 求解节拍与文献最优节拍差值与理想节拍关系

3.11 将机床分配给工人

在自动化导向型生产线上，将机器分配给工人和将任务分配给机器原理类似，只是约束条件产生了一定变化。而装配导向型生产线，在进行任务分配的时候即直接把任务分配到工人，不需要再进行该步骤。

3.11.1 约束兼假设条件

表 3.10 将机床分配给工人的约束和假设

序号	类型	描述
1	假设	工人在任意机床上的操作时间小该机床上所有任务的时间，即机床时间为任务的自动时间加上手动时间；
2	假设	假设人员行走时间在统计到任务手动时间里；
3	假设	工人在相邻两机器间行走时间相等；
4	假设	机床大小及摆放位置为默认；
5	约束	人员走动路线不可以交叉，即当任意工人已经被分配到某一个机床集的时候，他能够被继续分配的允许集只能是当前机器集的相邻机器或 U 型线的对向机器；

3.11.2 符号说明

表 3.11 将机器分配到工人符号说明

序号	符号	描述
1	w	工人编号；
2	j	机器编号；
3	$allowed_w$	机器允许集；

3.12.3 处理方式

机床大小和摆放位置为默认，即机床是严格按序号顺序摆放的，然后在居中处折叠返回，允许集为当前工人已经取得机器的相邻机器或当该工人尚未分配机器时所有已经分配机器的相邻集。然后，依据工序时间和工人工作时间按积木模型算法进行机器分配。

3.12.4 数据及实验

该部分数据来源于参考文献[3]，我们将其中每一个工序视为一个机器然后进行分配，实际上，由于该数据来源中每一个工序已经占据一个工位或者一个机器，将其视为模型当中的一个机器是完全合乎实际情况的。

将数据整理成我们要求的数据表格统计如表 3.12 所示。

表 3.12 参考文献[3]数据整理

机器序号	机器对应任务名称	工人时间	作业时间	机器序号	机器对应任务名称	工人时间	作业时间
1	油封 1 压入	14	20	25	调整工序	24	24
2	阀组件清洗	16	51	26	耐压 1	12	52
3	油封 2 压入	8	14	27	耐压 2	6	6
4	轴承 1 压入	7	13	28	性能试验 1	18	65
5	轴承 2 压入	21	27	29	性能试验 2	18	56
6	油封 3 压入	22	27	30	排油	17	54
7	轴承 3 压入	16	23	31	后装配 2IBJ 打紧	10	10
8	组件清洗	10	47	32	后装配 2 锁止垫片锁紧	10	10
9	齿条涂脂	16	16	33	后装配 2 装卡箍 1	20	20
10	齿条清洗插入	32	61	34	后装配 2 锁紧卡箍 1	8	8
11	压支撑套油封	15	22	35	后装配 2 装卡箍 2	15	15
12	装支撑套组件	27	27	36	装配 3 装螺钉、防尘罩	12	12
13	前装配 1 组件装夹具	15	15	37	后装配 3 装小型螺母	36	36
14	前装配 1 齿条堵头拧紧	8	8	38	后装配 3 装 OBJ	28	28
15	前装配 1 齿轮涂脂	6	6	39	后装配 3 装支架	9	9

16	前装配 1 装转阀组件	12	12	40	后装配 3 花键、螺纹检验	50	50
17	前装配 2 装油管 1	22	22	41	刻字	19	24
18	前装配 2 装调整体	11	11	42	夹具放行	7	7
19	前装配 2 装调整螺塞	13	13	43	终检擦拭、点检	26	26
20	前装配 3	49	49	44	终检装胶套	18	18
21	后装配 1 装油口堵头卸齿条堵头	28	28	45	终检贴标签	8	8
22	后装配 1 装锁止垫片、IBJ	30	30	46	总成下线	4	4
23	气试	30	76	47	返程	5	5
24	磨合试验	16	67				

在此实际问题当中，可以将工人时间看作是在不超过工站最大时间或者不超过分配到每个工人的任务时间的情况进行计算，即可以完成优化任务，有可以有效节省计算资源。

在参考文献[3]中，改善前工人总的手工时间为 824，9 个工人的情况下是瓶颈节拍为 161，改善后手工时间为 767，瓶颈节拍为 96。本文使用积木模型采用改善前的数据所求得的瓶颈节拍为 98，仅次于改善后的数据。

求解所得的$P_{worker} = \frac{\sum t_{worker}}{\max(t_{worker})\cdot n_{worker}} = \frac{824}{98*9} \times 100\% = 93.42\%$，显著优于改善前的 56.87%，也明显优于源文献改善后的 88.77%。当然，由于并没有拿到源文献数据的准确优先关系，计算结果可能稍有出入。

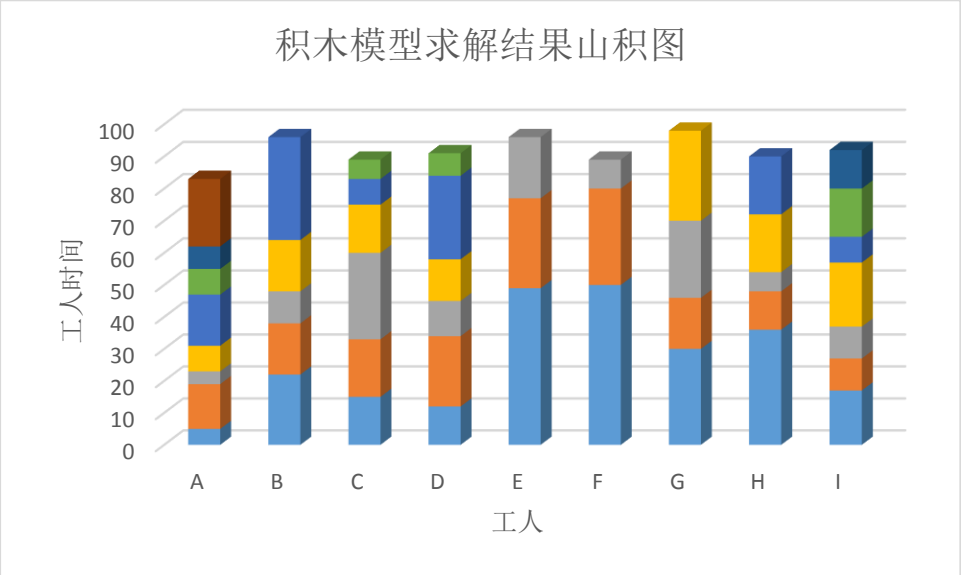


图 3. 10 参考文献 3 数据求解山积图

3.12 添加机器和任务属性的情况

目前的 U 型线研究主要集中于装配导向型 U 型装配线，而对自动化导向型研究较少，在自动化导向型生产线上，装配导向型对工站的适应性较强，基本任意工序都可以在工作台上装配完成，而自动化导向型对机器的依赖较强，只能由特定的机器和工装夹具来完成特定的工序。

本节主要针对的是算法当中生产线上的任意工作站可执行任意工序的假设的，在实际的生产当中，每一个机器只适合特定的工序，比如，车床只适合车削加工工序，铣床只适合表面铣削，电镀池只适合做特定的电镀处理等，这些条件会在实际当中影响到允许集的选取，从而影响到全局最优节拍的计算，也是实际生产当中比较重要的情况。

在机器和任务有其特定属性的情况下，允许集的选取添加约束条件为只有任务的属性为机器属性的子集的工序才可以分配给相应的机器，但是工人不受此约束。

由于此部分没有找到可以使用的公开数据，因此只能将该部分作为一个测试功能提供，而不能直接提供可信赖的计算结果参考。

第四章 效能评估

4.1 评估要素

由于生产线的实际应用性质，在生产线上会有很多临时性的，突发性和随机性的因素会影响工作质量或者工作时间，因此，需要对这些因素进行一定的预演和评价，以确保其处在一个稳定可控的范围内，实现生产线可控的最低效应和对各种随机因素的兼容性，容错性。

这些因素包含但不仅限于以下：

(1) 平衡率及平衡率损失；

(2) 工人的生产效率关于平均效率呈正态分布；

(3) 工人可能会缺班（即请假和旷勤，以及其他不满编的情况）；

(4) 工件出现的随机缺陷；

(5) 工作站（机床等）出现的随机故障；

(6) 工人的工作强度，如工作强度过强会导致离职率高，新手更多，熟练度差等状况，不利于生产线统一节拍。

4.2 成本绩效评估

成本绩效评估本质上是生产线平衡率和机器（工人）的工作强度的评估，可分为两个部分：总体成本评估和个人绩效评估。

4.2.1 总体成本评估

总体成本评估本质上上是平衡率及其损失的评估，在给定的问题当中，可以通过获得的最优解进行计算得出。公式为(3.1)和(3.2)。示例可参照将机床分配给

工人一节。

4.2.2 个人绩效评估

此处的个人不是单纯的指工人，也指机器，只是只有在当任务是分配到工人的时候需要进行该项评估，以提供给相应部门作为薪资报酬等的参考。

本小节狭义的绩效即指工人的工作强度，计算方式参考公式(3.3)，以 Bowman 问题为例，当给定 3 个机器（工人）的时候，输出结果为：

表 4.1 输出示例

机器	机器时间	任务数	任务序号	任务时间
1	26	3	8，1，6	3，11，12
2	26	2	2，3	17，9
3	23	3	7，4，5	10，5，8

在该种情况下，如机器 1 和 2 的工作强度即为 100%，机器 3 的工作强度为 $\frac{23}{26} = 88.46\%$，由于工人的任务分配和机器的任务分配同理，因此计算也同理。

4.3 弹性及可靠性评估

弹性，可靠性评估是同一个性质，当其弹性越好的时候，其可靠性也就越好。

该评价主要针对工人的工作时间在均值附近呈正态分布及工人缺勤等的影响。当工人在完成某一个任务时候的时间关于平均时间呈正态分布时，整个产线的节拍及效率都会受到极大的影响，本小节旨在通过给任务添加时间添加正态分布特性来观察周期及工作强度的变化范围。

首先构造符合正态分布的随机数，可以在 excel 当中构造，构造函数为 NORMINV(RAND(),mean,standard_dav)，mean 即为我们统计的任务平均时间，standard_dav 为标准方差，此处取 standard_dav=0.1*mean，然后统计总时间和节拍的变化。

采用 lutz3 问题 10 机器（工人）时候的数据，求得节拍 166，进行可靠性计算得到总时间$\pm 3.61\%$的变化幅度范围，单个机器的时间变化范围为 118.3498~190.4232，幅度为$\begin{matrix}+15.41\% \\ -20.03\%\end{matrix}$，以第一台机器为例强度变化范围为 87.3824%~97.5456% ，差距还是挺大，但这是用人上的问题，而不属于算法的问题，暂时还未有能够保证在受如此大的随机影响的情况下不重新分配就能够解决该问题的算法。

当工人临时缺勤或者机器临时故障是任务的分配，只需预先先进行相应机器数或工人数的情况的分配求解，即可对工人缺勤，机器故障对整个生产线的影响有准备有预案，而不用临时寻求分配方法。

4.4 结果对比分析

从蚁群算法和积木模型算法的结果来看，蚁群求得最优解的几率更低，而且局部寻优能力较差，积木模型寻优能力较强，但是对于较大的数据敏感度不够。

第五章 GUI 系统的设计及封装

5.1 流程及功能设计

软件的流程和功能设计直观重要，直接涉及到怎么用，是否好用。

本软件的基本流程为打开生产线问题文件，显示，设置参数，接收参数然后求解，显示结果。核心功能只有两个，求解生产线问题和显示生产线问题，设计如下：

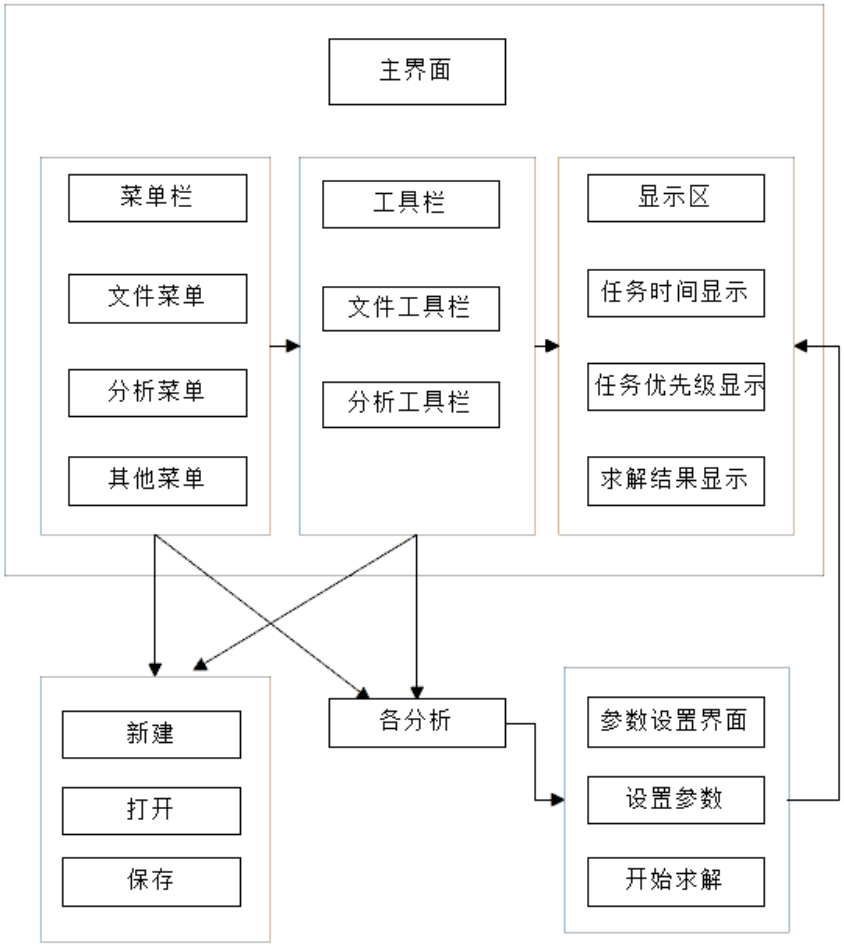


图 5.1 界面逻辑

5.2 基本界面设计

基本界面设计氛围两大部分，主界面设计和参数界面设计。

主界面设计涵盖菜单栏，工具栏和显示区，菜单栏涵盖文件操作，编辑操作，分析操作及帮助等。工具栏主要涵盖常用的菜单里面的工具，如打开、分析之类。

主界面上方是菜单栏，依次为各种菜单；下是工具栏，主要放置了文件操作和模型求解的各按钮；主页面分为三个表格：表格 1 为任务的时间，显示的是个任务及其所对应的时间，该时间一般指工序时间，特殊计算情况下应将其理解为工序的手动操作时间，表格 2 为优先级关系，第一列为前向工序，第二列为其对应的后向工序，表格 3 为求解结果输出表格，一共有 6 列，第 1 列为机器的序号，第 2 列为该机器所分的的任务时间，第三列为该机器分配到的任务数，第 4 列为该机器分配到的各任务的序号，第 5 列为各任务所对应的任务时间，第 6 列为当前机器的工作强度，时间越接近生产线节拍，则强度越大，该项主要是给与一些与劳动报酬相关的提议。

图 5. 2 主界面设计

图 5.3 蚁群参数设计界面

图 5.4 积木模型参数设计

5.3 工具安全性设计

软件将开放源码及使用权限，不涉及账号问题，只需要确认内部数据访问安全即可，而内部数据都在类内部进行计算和修改，不存在外泄活溢出的风险。

5.4 数据的输入输出设计

任何软件都有其标准的数据格式和能够接受的数据格式，才能保证其在后续的运行当中不会出错，同时接受的数据类型除了有特殊性，不会和其他不同类型的软件产生不必要的可能造成可能的共性外，还需要和同行业的软件尽可能的兼容。本软件由于致力于 U 型线的数据求解，目前主流的数据格式为“.IN2”，因此，本软件也使用”.IN2”格式作为标准输入格式，以兼容新出的数据集。当前的输出仅限于输出到界面表格当中，后续会考虑使用或创建标准格式进行输出，并与同行进行兼容。

“.IN2”文件的格式为文本形式，如图 5.5 所示，第 1 行为任务数，从第二行到第（任务数+1）行为任务 1 到任务结束的每一个任务所对应的时间，从第（任务数+2）行，一直到某一行为”-1,-1”或”-1,-1\n”结束为优先矩阵。其后可以是各种说明或者备注。

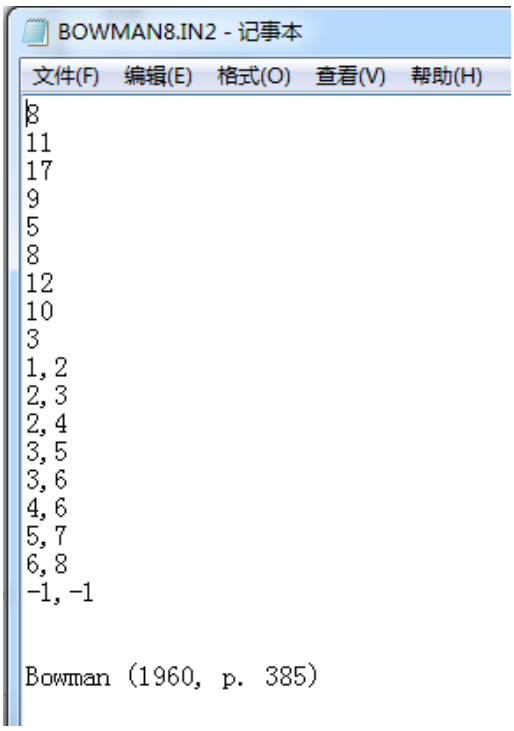


图 5.5 输入数据格式示例

输出文件由于其作用的不同和显示位置的不同分为 3 个小块，第一是界面的输出，如图 5.6 所示，显示在表格里面，每一行表示一个机器（或工人），依次

是对应的总机器时间，分配到的任务数，任务序号，及任务对应的时间，还有该机器（工人）的工作强度。

输出的文件采用“.OUT2”文件后缀，与输入的“.IN2”呼应，易于理解，内部格式如图 5.7 所示，为 python 语言的 dict（字典）格式，第一行和最后一行为 dict 的构造大括号，然后中间每一行是一个机器对应的分配数据，“number”和杭寿第一个数字表示的是机器序号，“total_mask_nums”表示该机器分配到的任务数，“total_times”表示该机器分配到的总任务时间，“the_task_number”表示分配到的所有任务的序号，“the_task_time”表示分配到的每一个任务的任务时间。

图片输出是山积图形式，格式为“.png”，如图 5.8 所示，主要是用于直观的显示每一个机器所分配到的任务的时间，能够明显的看出每一个机器的分配到的时间的差异情况。

求解结果

	机器序号	机器时间	任务数	任务序号	任务时间	强度
1	1	26.0	3	[8, 1, 6]	[3.0, 11.0, 1...	100.0%
2	2	24.0	3	[7, 3, 4]	[10.0, 9.0, 5...	92.307692...
3	3	25.0	2	[2, 5]	[17.0, 8.0]	96.153846...
4						
5						
6						
7						
8						
9						

图 5.6 界面结果输出

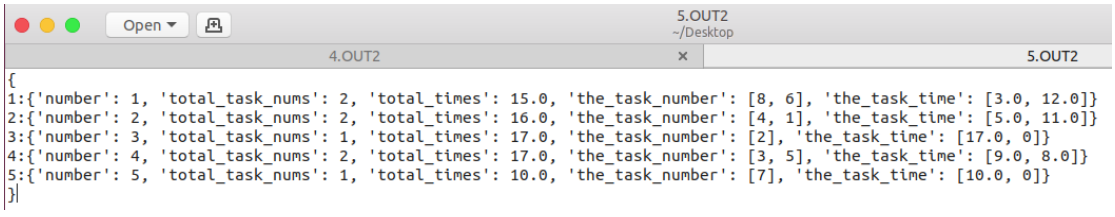


图 5.7 结果输出文件

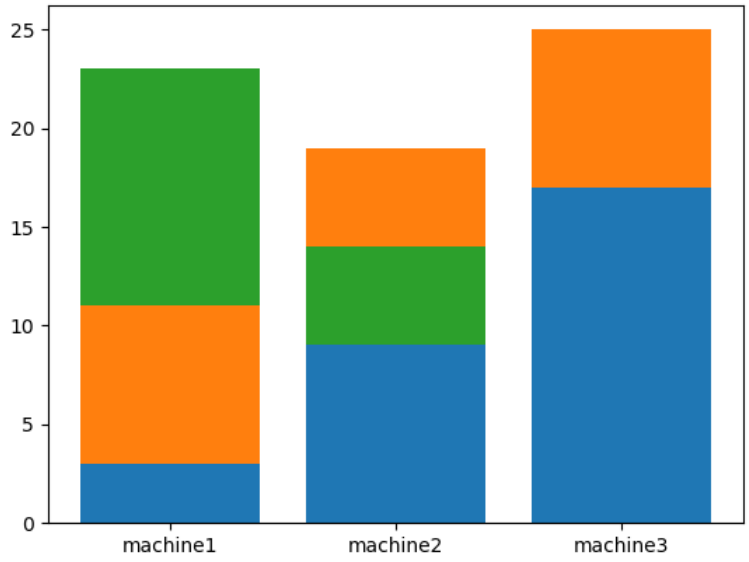


图 5.8 结果山积图形式

第六章 总结与展望

6.1 算法方面

本文使用了基本群算法用于解决第二类 U 型线问题，但是由于蚁群算法的参数设置等是诸多组合当中选取最优的方式来进行的，导致在同一组未经调整的参数的情况下，程序运行的结果不是很理想。

同时，本文在算法方面提出了使用积木模型求解 U 型线问题的思路，积木模型具有运算比较快，运算结果比较稳定，没有大量的参数依赖等优点，缺点是在数据集比较大的时候敏感度度会降低，求解效果会下降。

另外，本文提出了任务先分配给机器，再分配给工人的二级分步走策略，而每一级的算法原理都完全一样，用于解决在实际过程当中任务是先分配给机器再分配给工人的情况。

基于两个算法的各自特性，未来可尝试进行算法融合测试。

6.2 UI 方面

本文为整个求解过程设计了 UI 界面，UI 界面的使用更适合模型求解在各生产厂家进行推广，不需要了解多少算法的东西就能够进行运算，求解出尽可能最优的分配方案，并且提供了便利的快捷键和默认参数进行使用。

提供了分开的三个表格试图，分别为任务时间，任务优先级矩阵，结果视图。结果视图提供了完整的结果显示，能够迅速清楚的知道每一个机器的和任务的分配情况。

6.3 展望

在未来，该辅助设计工具还可以集成更多优秀的求解算法和优化当前已有的算法，在界面上面，可以添加可编辑的拖动式任务按钮，可以自由自在的画布上修改任务属性和时间，同时直观的看到各任务之间的优先级关系，对于数据的临时变更更有用，在输出方面也可以优化，进行山积图的优化，建立可拖动按钮式的结果分配界面等，增加用户界面的友好程度。

致谢

转眼间，整个论文的书写周期就已结束，在这个过程当中，留下了很多不同的记忆，感受到了很多人的关怀，也接收了很多人的帮助。

感谢指导老师朱德馨的悉心指导，不离不弃，能够让论文顺利完成，软件顺利运行。

同时在此衷心感谢邢凡凤同学无私的帮我的软件设计了几套 logo 方案，虽然最后没有采用，但是为我的使用 logo 和挑选各个快捷键的图标提供了有利的参考意见。

感谢大学四年带过我的导师，和和我一起做项目的同学，感谢你们帮助我完成了自我成长。

最后，感谢学校提供的友好宽松的科研氛围。

参考文献

[1] Ihsan Sabuncuoglu,Erdal Erel,Arda Alp.Ant colony optimization for the single model U-type assembly line balancing problem，Int. J. Production Economics ，2009，120：287－300；

[2] 段海滨.蚁群算法及其应用[M].北京：科学出版社，2005；

[3] 刘宗宁,孙先普.“一个流”U 型生产线的平衡分析和改善[J].研究与开发，2013，2：90-93；

[4] 童艺川,吴峰.U 型装配线的启发式平衡方法[J].南京理工大学学报,2000,24(5)：394-397；

[5] 郑巧仙,何国良等.第 2 类U型装配线平衡问题的双阶段蚁群算法[J].计算机科学,2017，44（6）：206-211,225；

[6] 查靓,徐学军等.多类约束下 U 型装配线平衡建模研究[J].工业工程与管理,2011,16（1）：59-69；

[7] 宋华明,韩玉启.基于 GA-SA 的混合U 型装配线平衡[J].运筹与管理,2002,11（6）：70-76；

[8] 曹阳华,孔繁森.以人为主的U 型装配线生产效率仿真[J].北京工业大学学报,2016，42（1）：42-50；

[9] 查靓,徐学军等.运用改进蚁群算法求解直线型和 U 型装配线平衡问题[J].工业工程，2010,13（6）：76-81；

[10]刘凯.U型装配线平衡与仿真研究[D].广东：广东工业大学，2012；

[11]林斌,不确定条件下U型装配线平衡模型研究[D].湖北：武汉科技大学，2015；

[12]邓福平.基于蚁群算法的装配线平衡问题研究[D].湖北：华中科技大学，2011；

[13]寇少威.具有小距离属性的NP难图问题何新华算法研究[D].四川:电子科技大学,2017；

[14]万建建.U型装配线人员步行路径规划[J].工程管理，2015，06：108-109；

[15] 曹阳华,孔繁森.基于行为研究的U 型装配线仿真[J].计算机集成制造系统，2013，19（11）：2765-2772；

[16] Scholl A,Klein R. Benchmark Data Sets by Scholl (1993)[DB/OL].[2018-3-18].https://assembly-line-balancing.de/salbp/benchmark-data-sets-1993/

附录

蚁群算法源码

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
import tkinter as tk
import sys
from copy import deepcopy


'''
版本说明：
该版本是随机扰动版本，且允许集包含前向允许和后向允许。
'''


'''
特别说明一下：
    由于 python 是从零开始计数的，而实际问题是由 1 开始计数的，这当中各项关系对应的时候，
转换难免会有一些麻烦，因此，我使用构造 0 元素的方法来进行计数和计算，即当元素为 0 时，其
赋值都为 0 或为空，以保证后续的序号都能对接上。可以将其称为伪任务，伪机器，伪工人，伪
位置等等。
'''
```

```
#定义任务类
class Task:
    '''
    任务矩阵，包含基本的任务数矩阵，任务时间矩阵，任务优先级矩阵
    '''
    def __init__(self, task_nums, task_time, precedence):
        self.task_nums=task_nums
        self.task_time=task_time
        self.precedence=precedence

    def out(self):
        '''输出 task'''
        print('task_nums=',self.task_nums)
        print('task_time=', self.task_time)
        print('precedence=', self.precedence)

    def weightForward(self,i):
        '''定义前向权重，即任务的任务时间'''
        return self.task_time[i]

# 通过输入进行定义的参数列表
# alpha：蚁群算法的参数 alpha
# beta ：蚁群算法的参数 beta
# thero：蚁群算法的参数 thero
# Q      ：信息素强度
# max_iter:
#           迭代次数，即需要完成的遍历次数
class parameter:
```

```
    def __init__(self,alpha,beta,thero,Q,max_iter,machine_nums):
        self.alpha=alpha
        self.beta=beta
        self.thero=thero
        self.Q=Q
        self.iter=max_iter
        self.machine_nums=machine_nums


#蚂蚁类
class Ant:
    def __init__(self,task_nums,total_times_max,machine_nums):
        self.task_nums=task_nums
        self.machine_nums=machine_nums
        self.total_times_max=total_times_max
        self.position=np.array([0])
        self.tabu=np.zeros(task_nums+1)-1
        self.machine={'number':0,
                      'total_task_nums':0,
                      'total_times':0,
                      'the_task_number':[],
                      'the_task_time':[]}
        self.plan={}

    '"重设所有非定义时初始化的值，以方便进行新的迭代而不会影响"'
    def re_init(self,meter):
        self.total_times_max=meter
        self.position=np.array([0])
        self.tabu=np.zeros(self.task_nums+1)-1
        self.machine={'number':0,
```

```
                    'total_task_nums':0,
                    'total_times':0,
                    'the_task_number':[],
                    'the_task_time':[]}
        self.plan={}

    '''设置当前位置'''
    def position_update(self,position):
        self.position=position
        return self.position

    '''设置当前禁忌表'''
    def set_tabu(self):
        #TODO:设置禁忌表方法
        j=0
        while j<self.task_nums+1:
            if self.tabu[j]==-1:
                self.tabu[j]=self.position
                break
            j=j+1
        #print('tabu=',self.tabu)

    '''或得当前总权重'''
    def get_sum_weight(self,task):
        sum_weight=0
        for position in self.tabu:
            if position==-1:
                break
            else:
```

```
                sum_weight=sum_weight+task.task_time[position]

        return sum_weight

    '''设置当前机器和方案'''
    def machine_update(self,task):
        #先检查零值的情况
        if self.position==0:
            self.plan[str(self.machine['number'])]=deepcopy(self.machine)
            self.machine['number']=self.machine['number']+1
            #print('open a new machine',self.machine['number'])

            #print("machine=",self.machine)
            #print("plan=",self.plan)
            return self.machine,self.plan

        #如果某机器的时间超过最大允许的 0.9，则判断添加新任务后会不会超时，超时则添加到下一机器，不超则添加到本机器
        #如果某机器时间未达到最大允许的 0.9，则任务添加到本机器
        #如果机器是最后一个机器，则不能打开一个新机器
        if self.machine['number']==self.machine_nums:
            self.machine['total_task_nums']+=1
            self.machine['total_times']+=task.task_time[self.position]
            #self.machine['the_task_number']=
            self.machine['the_task_number'].append(self.position)
            #self.machine['the_task_time']=
            self.machine['the_task_time'].append(task.task_time[self.position])
            '''
        elif self.machine['total_times']>=(self.total_times_max*0.9):
```

```
            if
(self.machine['total_times']+task.task_time[self.position])>self.total_times_max:
                self.plan[str(self.machine['number'])]=deepcopy(self.machine)
                self.machine['number']=self.machine['number']+1
                print('open a new machine',self.machine['number'])
                self.machine['total_task_nums']=1
                self.machine['total_times']=task.task_time[self.position]
                self.machine['the_task_number']=[self.position]
                self.machine['the_task_time']=[task.task_time[self.position]]
            else:
                self.machine['total_task_nums']+=1
                self.machine['total_times']+=task.task_time[self.position]
                #self.machine['the_task_number']=
                self.machine['the_task_number'].append(self.position)
                #self.machine['the_task_time']=
                self.machine['the_task_time'].append(task.task_time[self.position])
            '''
        elif self.position==0:
            self.plan[str(self.machine['number'])]=deepcopy(self.machine)
            self.machine['number']=self.machine['number']+1
            #print('open a new machine',self.machine['number'])
        elif
(self.machine['total_times']+task.task_time[self.position])>self.total_times_max:
            self.plan[str(self.machine['number'])]=deepcopy(self.machine)
            self.machine['number']=self.machine['number']+1
            #print('open a new machine',self.machine['number'])
            self.machine['total_task_nums']=1
            self.machine['total_times']=task.task_time[self.position]
            self.machine['the_task_number']=[self.position]
```

```
            self.machine['the_task_time']=[task.task_time[self.position]]
        else:
            self.machine['total_task_nums']+=1
            self.machine['total_times']+=task.task_time[self.position]
            #self.machine['the_task_number']=
            self.machine['the_task_number'].append(self.position)
            #self.machine['the_task_time']=
            self.machine['the_task_time'].append(task.task_time[self.position])

        #如果所有任务已经分配完成，则更新最后一个机器的数据到方案里面
        if (self.position)==self.task_nums:
            self.plan[str(self.machine['number'])]=deepcopy(self.machine)
        self.plan[str(self.machine['number'])]=deepcopy(self.machine)

        #print("machine=",self.machine)
        #print("plan=",self.plan)

        return self.machine,self.plan

    '''计算当前路径索要添加的信息素并返回'''
    def delta_phero(self,task,para):
        if self.position==0:
            delta_T=1
        else:
            delta_T=para.Q*task.task_time[self.position]/np.max(task.task_time)

        return    delta_T

    '''获得选取概率，然后选择相应路径'''
```

```
def chioce(self,task,map_a,allowed_k,para):
    #先取得允许集的最大权重
    weight=[]
    for i in allowed_k:
        weight.append(task.weightForward(i))

    #print('weight=',weight)
    max_weight=max(weight)

    #获得地图信息素浓度并求解 ng_i
    T=[]
    ng_i=[]
    for i in np.arange(np.size(allowed_k)):
        t=map_a[allowed_k[i],self.machine['number']] #轨迹值
        T.append(t)
        tt=task.weightForward(allowed_k[i])/max_weight #前向权重除以最大
权重
        ng_i.append(tt)

    #求和获得一般概率公式的分母
    T=np.array(T)
    ng_i=np.array(ng_i)
    den=sum(T**para.alpha*ng_i**para.beta)

    #求解一般概率矩阵,就是再这儿添加的随机扰动矩阵

P_ijk=rd.rand((np.size(allowed_k)))*((ng_i**para.alpha)*(T**para.beta))/den
    #求解所要获得的选择的路径
    #方法是取得最大概率的位置，然后该位置所对应的允许集的位置的坐
```

标就是所选的坐标

```
        position=allowed_k[np.where(P_ijk==np.max(P_ijk))]
        #print('np.max(P_ijk)=',np.max(P_ijk))
        #print('np.where(np.max(P_ijk))=',np.where(np.max(P_ijk)))
        #print('P_ijk=',P_ijk)

        return position,P_ijk

    '''获取当前允许分配的任务集'''
    def get_allowed_k(self,task):
        #挨个所有任务是否在禁忌表里,若不在则放到 d 里面
        d=[]
        for t in range(task.task_nums+1):
            if (t in self.tabu):
                continue
            elif (t in d):
                continue
            else:
                d.append(t)
        #print('d=',d)


        #可分配任务分为两种，没有前序任务的任务和所有前序任务都分配完成的任务
        #查找所有未分配的后序任务的前序任务是否已经分配完成
        #若是，则将该后序任务写入 allowed_k
        #否则，继续下移次循环
        #print('task.precedence[1]=',task.precedence[1])
        #print('list(task.precedence[1])=',list(task.precedence[1]))
```

```
        allow_f=[]
        for t in d:
            e=np.where(task.precedence[1]==t)
            #记得改回去
            #if t in list(task.precedence[1]):
                #e.append(list(task.precedence[1]).index(t))
            #print('t=',t)
            #print('e=',e)

            if len(e[0])==0:
                allow_f.append(t)
            else:
                ite=0
                while ite<len(e[0]):
                    #print('tt=',tt)
                    #print('len(e)=',len(e[0]))

#print('task.precedence[0,e[0][ite]]=',task.precedence[0,e[0][ite]])
                    if (task.precedence[0,e[0][ite]] in self.tabu):
                        if (t in allow_f):
                            pass
                        else:
                            allow_f.append(t)
                        #print('if:allow=',allow)

                    else:
                        #if t==1:
                            #allow.append(t)
                        if (t in allow_f):
```

```
                        allow_f.remove(t) #删掉 t
                    #print('else:allow=',allow)
                #print('allow=',allow)
                ite+=1
    allow_b=[]
    for t in d:
        e=np.where(task.precedence[0]==t)

        if len(e[0])==0:
            if (t in allow_b):
                continue
            else:
                allow_b.append(t)
        else:
            ite=0
            while ite<len(e[0]):
                #print('tt=',tt)
                #print('len(e)=',len(e[0]))

#print('task.precedence[0,e[0][ite]]=',task.precedence[0,e[0][ite]])
                if (task.precedence[1,e[0][ite]] in self.tabu):
                    if (t in allow_b):
                        pass
                    else:
                        allow_b.append(t)
                    #print('if:allow=',allow)

                else:
                    #if t==1:
```

```
                        #allow.append(t)
                    if (t in allow_b):
                        allow_b.remove(t) #删掉 t
                    #print('else:allow=',allow)
                #print('allow=',allow)
                ite+=1
        allow=allow_f
        for t in allow_b:
            if t in allow:
                pass
            else:
                allow.append(t)

        allowed_k=np.array(allow)
        #print('allowed_k=',allow)
        return allowed_k

    '''输出测试'''
    def out(self):
        print('self.position=',self.position)
        print('self.tabu=',self.tabu)

def inital_ant(num,task_nums,total_times_max,machine_nums):
    '''初始化输入数量的蚂蚁'''

    char=['ant_','=Ant('+str(task_nums)+','+str(total_times_max)+','+str(machine_nu
ms)+')']
    ant_list=[]
    ants_list=[]
```

```
    for i in range(num):
        ant_list.append(char[0]+str(i)+char[1])

    #print(ant_list)

    i=0
    while i<num:
        exec(ant_list[i])
        ants_list.append(eval(char[0]+str(i)))
        i=i+1
    #print(ants_list)
    return ants_list

#获取任务数据
def get_task():
    #使用对话框选择文件夹或文件
    root=tk.Tk()
    root.withdraw()
    #fpath=tk.filedialog.askopenfilename()
    string=tk.filedialog.askopenfilename()

    message='Error:filename wrong\n filename\'s type not a string'
    print(string)
    if type(string)!=str:
        return message
    df=open(string).readlines()
    task_num=int(df[0])             #获得任务数
    #print(task_num)
```

```
    #获得任务时间
    task_time=np.zeros(task_num+1)
    i=1
    while i<task_num+1:
        task_time[i]=int(df[i])
        i+=1


    #获得优先级
    n=df.index('-1,-1\n')
    precedence = np.zeros([2, n-task_num-1])
    while i<n:
        precedence_port=df[i]                    #precedence 的中间量
        j=precedence_port.index(',')          #j,k 在这里的作用是提出优先阵中的分
隔符‘，’和‘\’
        k=precedence_port.index('\n')
        precedence[0,i-task_num-1]=int(precedence_port[:j])
        precedence[1,i-task_num-1]=int(precedence_port[j+1:k])
        #print(precedence)
        i+=1

    #赋值并返回
    task=Task(task_num,task_time,precedence)
    #task.out()
    return task

'''初始化地图'''
def initmap(task_nums,machine_nums):
    map_a=np.ones((task_nums+1,machine_nums+1))
```

```
    return map_a

'''基本蚁群办法解决生产线问题'''
def ANT_STD_ULINE(task,ants,map_a,para):
    #整个函数就是一堆嵌套循环，直到所有条件满足，输出最佳方案

    #定义全局最佳方案和最佳节拍
    global_min_meter=sys.maxsize
    global_best_plan={}

    #最外层，判断是否完成遍历条件
    for ite in np.arange(para.iter):
        print(ite,'次迭代开始')

        #先进行零值的初始赋值，然后才进行实际运算
        delta_phero=np.zeros((task.task_nums+1,para.machine_nums+1))
        delta_map=np.zeros((task.task_nums+1,para.machine_nums+1))
        for k in ants:
            k.position_update(0)                                    #将当前位置设
为 0
            k.set_tabu()                                             #更新禁忌表
            machine,plan=k.machine_update(task)                     #更新机器和方
案
            #print(machine)
            delta_map=np.zeros((task.task_nums+1,para.machine_nums+1))
            delta_map[machine['number'],0]=k.delta_phero(task,para)
            delta_phero+=delta_map[machine['number'],0]
                                                                    #累加所有
蚂蚁将要添加的信息素
```

```
        map_a=(1-para.thero)*map_a+delta_phero                    #更新全局
信息素

        #开始进行实际的路径选择
        t=0
        while t<task.task_nums:
            #print(t,'次选择开始')
            delta_phero=[]
            weight=[]
            for k in ants:
                weight_k=k.get_sum_weight(task)                    #获取当前蚂
蚁已走路径的总权重
                weight.append(weight_k)                           #将每只蚂蚁
已走路径的总权重存起来

            if sum(weight)==0:
                #如果已经走过的路径的总权重为零，则依次选择路径，
                #然后统一更新信息素
                for k in ants:
                    allowed_k=k.get_allowed_k(task)                #获得允许
集
                    position,P_ijk=k.chioce(task,map_a,allowed_k,para)  #获得选
择位置
                    #print('position=',position)
                    k.position_update(position)                    #更新当前位
置
                    k.set_tabu()                                  #更新禁忌表
                    machine,plan=k.machine_update(task)            #将当前任
务分配给机器并且更新方案
```

```
delta_map=np.zeros((task.task_nums+1,para.machine_nums+1))
                                                    #每只蚂蚁的
时候都清零

delta_map[position,machine['number']]=k.delta_phero(task,para)
                                                    #获得
每只蚂蚁所要添加的信息素
                    delta_phero.append(delta_map)          #获得所有
蚂蚁所路过的位置所需要添加的信息素的列表
                map_a=(1-para.thero)*map_a+sum(delta_phero)          #更新
全局信息素
            else:
                #所有蚂蚁已走过的路径的权重的和不为零
                #则已走路径权重大的蚂蚁有限进行路径选择并更新信息
                #此举意在使路径短的蚂蚁能够影响到路径长的蚂蚁的选择
                map_a=(1-para.thero)*map_a
#先蒸发后读取
                ii=0
                while ii<len(weight):
                    if max(weight)!=-1:
                        p=weight.index(max(weight))                    #
找到最大权重第一次出现的地方
                        k=ants[p]
#该位置对应的蚂蚁
                        weight[p]=-1
#将最大值改为最小值

                        #进行常规的选择操作
```

```
                allowed_k=k.get_allowed_k(task)
                #print('allowed_k=',allowed_k)                #获得允许集
                position,P_ijk=k.chioce(task,map_a,allowed_k,para)  #获得选择位置
                #print('position=',position)
                k.position_update(position)                  #更新当前位置
                k.set_tabu()                                 #更新禁忌表
                machine,plan=k.machine_update(task)
                #print('machine=',machine)                   #将当前任务分配给机器并且更新方案
                #print('plan=',plan)

    delta_map=np.zeros((task.task_nums+1,para.machine_nums+1))      #每只蚂蚁的时候都清零

    delta_map[position,machine['number']]=k.delta_phero(task,para)  #获得每只蚂蚁所要添加的信息素
                map_a=map_a+delta_map                        #进行单个蚂蚁的信息素的更新,这样先进性选择的蚂蚁的信息素就可以影响到后选择的蚂蚁的选择
            #else:
                #基本不会有希望用到这个 else，但还是保险起见，让他跳出循环
                #break

            ii+=1
        #print(t,'次选择结束')
        t+=1
```

```
#每只蚂蚁都遍历完成后，提取出最佳方案
#最佳方案的标准是平衡率最大，即节拍最小
plans=[]
meter=[]                                    #表示节拍
for k in ants:
    plans.append(deepcopy(k.plan))
    #print('plan=',plan)

    #取得单个 plan 的节拍
    meter_port=0
    for key in k.plan.keys():
        #print('k.key=',k,key)
        #print("k.plan[key]['total_times']",k.plan[key]['total_times'])

        if k.plan[key]['total_times']>meter_port:
            meter_port=k.plan[key]['total_times']
            #print('meter_port=',meter_port)
        else:
            pass

    meter.append(meter_port)

#print('plan=',plan)
print('meter=',meter)
#获得此次遍历的最佳方案
min_meter=min(meter)
index=meter.index(min_meter)
#print('index=',index)
```

```
            #print('plans[index]=',plans[index])
            part_best_plan=deepcopy(plans[index])
            #print('part_best_plan=',part_best_plan)


            #获得当前全局最佳方案
            if global_min_meter>min_meter:
                global_min_meter=min_meter
                global_best_plan=deepcopy(part_best_plan)
            else:
                pass


            #现完成单次遍历完成后，还需进行多次遍历的计算
            #因此，在此处对每只蚂蚁再进行重初始化
            #重初始化不会影响到蚂蚁最开始对蚂蚁进行初始化定义时的值
            for k in ants:
                k.re_init(global_min_meter-1)

            print(ite,'次迭代结束')


        #所有遍历次数都结束后，输出最佳方案，及其节拍
        return global_best_plan,global_min_meter

def check_mean_and_max(task,machine_nums):
    pass
```

```
#########################################################################
##      以下是主程序
#########################################################################
if __name__ == '__main__':

    #task=get_task('F:\毕业论文\数据\SALBP-data-sets\precedence
graphs\ARC83.IN2')
    task=get_task()
    #task.out()
    ant_num=10
    machine_nums=int(input('please input the machine numbers:'))
    #para=parameter(alpha,beta,thero,Q,max_iter,machine_nums)
    para=parameter(0.6,0.6,0.8,0.5,200,machine_nums)

    #print(type(machine_nums))
    total_times_max=np.sum(task.task_time)/machine_nums*1.05
    #check_mean_and_max();
    #print('sum of task.task_time=',np.sum(task.task_time))
    #print('total_times_max=',total_times_max)
    ants=inital_ant(ant_num,task.task_nums,total_times_max,machine_nums)
    #print(type(ants[0]))
    #ants[0].out()
```

```
map_a=initmap(task.task_nums,machine_nums)
plan,meter=ANT_STD_ULINE(task,ants,map_a,para)
#print('plan=',plan)
#print('meter=',meter)
s=0
s_t=0
for key in plan:
    s+=plan[key]['total_times']
    print(key,plan[key]['total_times'])
    print(plan[key]['the_task_number'])
    s_t+=plan[key]['total_task_nums']
print('s=',s)
print('s_t=',s_t)
print('meter=',meter)

#print('total_times_max=',total_times_max)
```

积木模型源码

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
import tkinter as tk
import sys
from copy import deepcopy


'''
版本说明：积木模型基本版本
```

```
'''



'''
max_iter:最大迭代次数；
machine_nums：机器数量；
top_boder：上界条件；
'''
class parameter:
    def __init__(self,max_iter,machine_nums,time_mean,rand):
        self.iter=max_iter
        self.machine_nums=machine_nums
        self.top_boder=0
        self.time_mean=time_mean
        self.rand=rand
    def set_top_boder(self,top_boder):
        '''设置上界'''
        self.top_boder=top_boder


class Task:
    '''
    任务矩阵，包含基本的任务数矩阵，任务时间矩阵，任务优先级矩阵
    '''
    def __init__(self, task_nums, task_time, precedence):
        self.task_nums=task_nums
        self.task_time=task_time
        self.precedence=precedence


    def out(self):
```

```
        '''输出 task'''
        print('task_nums=',self.task_nums)
        print('task_time=', self.task_time)
        print('precedence=', self.precedence)
    def get_time(self,i):
        return self.task_time[i]
    def get_timesum(self):
        return np.sum(self.task_time)


class Wood:
    def __init__(self,task_nums,machine_nums,top_boder):
        self.task_nums=task_nums
        self.machine_nums=machine_nums
        self.top_boder=top_boder
        self.wood=np.array([0])
        self.tabu=np.zeros(task_nums+1)-1
        self.space={'number':0,
                    'total_task_nums':0,
                    'total_times':0,
                    'the_task_number':[],
                    'the_task_time':[]}
        self.plan={}

    '''设置禁忌表'''
    def set_tabu(self):
        #TODO:设置禁忌表方法
        j=0
        while j<self.task_nums+1:
            if self.tabu[j]==-1:
```

```
                self.tabu[j]=self.wood
                break
            j=j+1
        #print('tabu=',self.tabu)

'''设置当前小积木块'''
def wood_update(self,wood):
    self.wood=wood
    return self.wood

'''常规选择方式,不涉及到随机捆绑'''
def chioce(self,allowed_k,task,para):

    #取得允许集对应的时间
    time=[]
    for i in allowed_k:
        time.append(task.get_time(i))
    #print('time=',time)

    P=[]
    if self.top_boder==0:
        #当上界为初始上界的时候，执行概率计算
        for i in range(len(time)):

P.append(1+np.log((self.space['total_times']+time[i])/para.time_mean))
            if P[i]<0 and (para.rand):
                #print('run if if')
                P[i]=rd.rand(1)
                P[i]=[0]
```

```
                print(P[i])
            else:
                #print('run if else')
                P[i]=[0]
                print(P[i])
        else:
            #当上界为确定上界的时候，超过均值的部分采用线性下降
            for i in range(len(time)):
                time_total=self.space['total_times']+time[i]
                if time_total>para.time_mean:
                    #大于上界就把概率改为-1
                    #print('run else if',para.time_mean,self.top_boder)
                    if time_total>self.top_boder:
                        P.append(-1)
                    else:
                        print('time_total',time_total)
                        P.append((time_total-para.time_mean)/(self.top_boder-
para.time_mean))
                else:
                    #print('run else else')

    P.append(1+np.log((self.space['total_times']+time[i])/para.time_mean))
                    if P[i]<0 and (para.rand):
                        #此处有问题，待修改
                        P[i]=rd.rand(1)
                    else:
                        P[i]=0

        #print('P',P)
```

```
        P_ij=np.array(P)
        #print('1,P_ij',P_ij)

        #如果全部都超过上界的话就只能分配给下一个槽了，
        #但是此处不做槽的更新，只能将该信息传出去，并重新计算概率

        new_space=0
        if np.max(P_ij)==-1:
            print('re caculate')
            new_space=1
            P=[]
            for i in range(len(time)):
                time_total=time[i]
                if time_total>para.time_mean:
                    #大于上界就把概率改为-1
                    if time_total>para.top_boder:
                        P.append([-1])
                    else:
                        P.append((time_total-para.time_mean)/(self.top_boder-
para.time_mean))
                else:
                    P.append(1+np.log(time[i]/para.time_mean))
                    if P[i]<0 and (~para.rand):
                        P[i]=[0]
                    else:
                        P[i]=rd.rand(1)
        P_ij=np.array(P)
        #print('P_ij',P_ij)
        #print('max_position',np.where(P_ij==np.max(P_ij)))
```

```
        m=np.where(P_ij==np.max(P_ij))
        #print('m',m,'\nm[0]',m[0])
        if len(m)>1:
            wood=allowed_k[m[0][0]]
        elif len(m[0])>1:
            wood=allowed_k[m[0][0]]
        else:
            wood=allowed_k[m[0]]



        return wood,P_ij,new_space

    '''更新积木槽'''
    def space_update(self,task,new_space):
        #先检查零值的情况
        if self.wood==0:
            self.plan[str(self.space['number'])]=deepcopy(self.space)
            self.space['number']=self.space['number']+1
            #print('open a new space',self.space['number'])


            #print("space=",self.space)
            #print("plan=",self.plan)
            return self.space,self.plan
        if self.space['number']==self.machine_nums:
            self.space['total_task_nums']+=1
            self.space['total_times']+=task.task_time[self.wood]
            self.space['the_task_number'].append([self.wood])
            self.space['the_task_time'].append([task.task_time[self.wood]])
        elif new_space==1:
```

```
            self.plan[str(self.space['number'])]=deepcopy(self.space)
            self.space['number']=self.space['number']+1
            print('open a new space',self.space['number'])
            self.space['total_task_nums']=1
            self.space['total_times']=task.task_time[self.wood]
            self.space['the_task_number']=[self.wood]
            self.space['the_task_time']=[task.task_time[self.wood]]
        else:
            self.space['total_task_nums']+=1
            self.space['total_times']+=task.task_time[self.wood]
            self.space['the_task_number'].append([self.wood])
            self.space['the_task_time'].append([task.task_time[self.wood]])

        self.plan[str(self.space['number'])]=deepcopy(self.space)
        return self.space,self.plan

    '''获取当前允许分配的任务集'''
    def get_allowed_k(self,task):
        #挨个所有任务是否在禁忌表里,若不在则放到 d 里面
        d=[]
        for t in range(task.task_nums+1):
            if (t in self.tabu):
                continue
            elif (t in d):
                continue
            else:
                d.append(t)
        #print('d=',d)
```

```
        #可分配任务分为两种，没有前序任务的任务和所有前序任务都分配完成的任务
        #查找所有未分配的后序任务的前序任务是否已经分配完成
        #若是，则将该后序任务写入 allowed_k
        #否则，继续下移次循环
        #print('task.precedence[1]=',task.precedence[1])
        #print('list(task.precedence[1])=',list(task.precedence[1]))
        allow_f=[]
        for t in d:
            e=np.where(task.precedence[1]==t)
            #记得改回去
            #if t in list(task.precedence[1]):
                #e.append(list(task.precedence[1]).index(t))
            #print('t=',t)
            #print('e=',e)

            if len(e[0])==0:
                allow_f.append(t)
            else:
                ite=0
                while ite<len(e[0]):
                    #print('tt=',tt)
                    #print('len(e)=',len(e[0]))

    #print('task.precedence[0,e[0][ite]]=',task.precedence[0,e[0][ite]])
                    if (task.precedence[0,e[0][ite]] in self.tabu):
                        if (t in allow_f):
                            pass
```

```
                    else:
                        allow_f.append(t)
                    #print('if:allow=',allow)


                else:
                    #if t==1:
                        #allow.append(t)
                    if (t in allow_f):
                        allow_f.remove(t) #删掉 t
                    #print('else:allow=',allow)
                #print('allow=',allow)
                ite+=1
    allow_b=[]
    for t in d:
        e=np.where(task.precedence[0]==t)

        if len(e[0])==0:
            if (t in allow_b):
                continue
            else:
                allow_b.append(t)
        else:
            ite=0
            while ite<len(e[0]):
                #print('tt=',tt)
                #print('len(e)=',len(e[0]))

#print('task.precedence[0,e[0][ite]]=',task.precedence[0,e[0][ite]])
                if (task.precedence[1,e[0][ite]] in self.tabu):
```

```
                    if (t in allow_b):
                        pass
                    else:
                        allow_b.append(t)
                    #print('if:allow=',allow)


                else:
                    #if t==1:
                        #allow.append(t)
                    if (t in allow_b):
                        allow_b.remove(t) #删掉 t
                    #print('else:allow=',allow)
                #print('allow=',allow)
                ite+=1
        allow=allow_f
        for t in allow_b:
            if t in allow:
                pass
            else:
                allow.append(t)

        allowed_k=np.array(allow)
        #print('allowed_k=',allow)
        return allowed_k

    '''重设所有非定义时初始化的值，以方便进行新的迭代而不会影响'''
    def re_init(self,meter):
        self.top_boder=meter
        self.position=np.array([0])
```

```
        self.tabu=np.zeros(self.task_nums+1)-1
        self.space={'number':0,
                            'total_task_nums':0,
                            'total_times':0,
                            'the_task_number':[],
                            'the_task_time':[]}
        self.plan={}

def get_task():
    #使用对话框选择文件夹或文件
    root=tk.Tk()
    root.withdraw()
    #fpath=tk.filedialog.askopenfilename()
    string=tk.filedialog.askopenfilename()




    message='Error:filename wrong\n filename\'s type not a string'
    print(string)
    if type(string)!=str:
        return message
    df=open(string).readlines()
    task_num=int(df[0])              #获得任务数
    #print(task_num)

    #获得任务时间
    task_time=np.zeros(task_num+1)
```

```
    i=1
    while i<task_num+1:
        task_time[i]=int(df[i])
        i+=1



    #获得优先级
    n=df.index('-1,-1\n')
    precedence = np.zeros([2, n-task_num-1])
    while i<n:
        precedence_port=df[i]                    #precedence 的中间量
        j=precedence_port.index(',')          #j,k 在这里的作用是提出优先阵中的分
隔符‘，’和‘\’
        k=precedence_port.index('\n')
        precedence[0,i-task_num-1]=int(precedence_port[:j])
        precedence[1,i-task_num-1]=int(precedence_port[j+1:k])
        #print(precedence)
        i+=1



    #赋值并返回
    task=Task(task_num,task_time,precedence)
    #task.out()
    return task

'''初始化地图'''
def initmap(task_nums,machine_nums):
    map_a=np.ones((task_nums+1,machine_nums+1))
    return map_a
```

```
def STD_WOOD(task,para,wood):
    #定义全局最佳方案和最佳节拍
    global_min_meter=sys.maxsize
    global_best_plan={}

    #最外层，判断是否完成遍历条件
    for ite in np.arange(para.iter):
        print(ite,'次迭代开始')
        i=0
        part_meter=0
        while i<=task.task_nums:
            if i==0:
                wood.wood_update(0)                                    #将当前位
置设为 0
                wood.set_tabu()                                        #更新
禁忌表
                machine,plan=wood.space_update(task,1)
                i+=1
                continue
            #print(i,'次选择开始')
            allowed_k=wood.get_allowed_k(task)
            #print('allowed_k=',allowed_k)
            wood_i,P_ij,new_space=wood.chioce(allowed_k,task,para)
            #print('wood_i',wood_i)
            #print('P_ij=',P_ij)
            #print('new_space=',new_space)
            wood.wood_update(wood_i)
```

```
            wood.set_tabu()
            space,plan=wood.space_update(task,new_space)
            #print('plan',plan)
            #print('space',space)
            #print(i,'次选择结束')
            i+=1
        #更新上界和清空槽和禁忌表
        for key in plan.keys():
            if plan[key]['total_times']>part_meter:
                part_meter=plan[key]['total_times']

        if global_min_meter>part_meter:
            global_min_meter=part_meter
            global_best_plan=deepcopy(plan)

        wood.re_init(global_min_meter-1)
        print(ite,'次迭代结束')


    #所有遍历次数都结束后，输出最佳方案，及其节拍
    return global_best_plan,global_min_meter



if __name__ == '__main__':
    task=get_task()
    print('task_time',task.task_time)
    machine_nums=int(input('please input the machine numbers:'))
    time_mean=task.get_timesum()/machine_nums
```

```
    #para=parameter(max_ite,machine_nums,time_mean,rand)
    para=parameter(100,machine_nums,time_mean,1)
    top_boder=1.02*time_mean
    wood=Wood(task.task_nums,machine_nums,top_boder)
    plan,meter=STD_WOOD(task,para,wood)
    print('plan=',plan)
    print('meter=',meter)
```

界面及整合源码：

主调用程式：test_main.py

```
from test_myself import *
import sys


if __name__ == '__main__':
     app = QApplication(sys.argv)
     Window=MainWindow()
     Window.show()
     sys.exit(app.exec_())
```

任务类：Task.py

```
import numpy as np
#纯粹的就是任务的类
class Task:
    '''
    任务矩阵，包含基本的任务数矩阵，任务时间矩阵，任务优先级矩阵
    '''
    def __init__(self, task_nums, task_time, precedence):
```

```
        self.task_nums=task_nums
        self.task_time=task_time
        self.precedence=precedence

    def out(self):
        '''输出 task'''
        print('task_nums=',self.task_nums)
        print('task_time=', self.task_time)
        print('precedence=', self.precedence)
    def get_time(self,i):
        return self.task_time[i]
    def get_timesum(self):
        return np.sum(self.task_time)
```

界面程序：test_myself.py

```
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *

import sys
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
import tkinter as tk
import sys
from copy import deepcopy

from Task import *
```

```
from both_para import *
from Wood import *
from ACO_3fb import *
#import ACO_3fb

#蚁群参数设置界面
class Ants_Dialog(QDialog):
    def __init__(self,*argv,**kwargs):
        super(Ants_Dialog,self).__init__(*argv,**kwargs)
        self.setWindowTitle('设置蚁群参数')
        self.setWindowIcon(QIcon('../Icon/蚂蚁.png'))
        self.resize(500,200)

        QBtn=QDialogButtonBox.Ok |QDialogButtonBox.Cancel
        self.buttonBox=QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)

        #所有的标签和框
        self.ants_label=QLabel("蚂蚁数量")
        self.ants_edit=QLineEdit()
        self.machines_label=QLabel("机器数量")
        self.machines_edit=QLineEdit()
        self.top_label=QLabel("上限倍数")
        self.top_edit=QLineEdit()
        self.iter_label=QLabel("迭代次数")
        self.iter_edit=QLineEdit()

        self.alpha_label=QLabel("alpha")
```

```
self.alpha_edit=QLineEdit()
self.beta_label=QLabel("beta")
self.beta_edit=QLineEdit()
self.thero_label=QLabel("thero")
self.thero_edit=QLineEdit()
self.rand_radio=QRadioButton("使用随机规则")
self.Q_label=QLabel("信息素强度 Q")
self.Q_edit=QLineEdit()

self.mapping_radio=QRadioButton("对应关系")
self.mount_radio=QRadioButton("山积图")

#参数设置组
self.groupBox_1 = QGroupBox("参数设置")
layout_1=QVBoxLayout()
layout_1.addWidget(self.ants_label)
layout_1.addWidget(self.ants_edit)
layout_1.addWidget(self.machines_label)
layout_1.addWidget(self.machines_edit)
layout_1.addWidget(self.top_label)
layout_1.addWidget(self.top_edit)
layout_1.addWidget(self.iter_label)
layout_1.addWidget(self.iter_edit)
self.groupBox_1.setLayout(layout_1)

self.groupBox_2 = QGroupBox("算法参数")
layout_2=QVBoxLayout()
layout_2.addWidget(self.alpha_label)
layout_2.addWidget(self.alpha_edit)
```

```
        layout_2.addWidget(self.beta_label)
        layout_2.addWidget(self.beta_edit)
        layout_2.addWidget(self.thero_label)
        layout_2.addWidget(self.thero_edit)
        layout_2.addWidget(self.Q_label)
        layout_2.addWidget(self.Q_edit)
        layout_2.addWidget(self.rand_radio)
        self.groupBox_2.setLayout(layout_2)

        self.groupBox_3 = QGroupBox("输出选项")
        layout_3=QVBoxLayout()
        layout_3.addWidget(self.mapping_radio)
        layout_3.addWidget(self.mount_radio)
        self.groupBox_3.setLayout(layout_3)

        #竖直切割布局，放三个参数框
        self.layout_H=QHBoxLayout()
        self.layout_H.addWidget(self.groupBox_1)
        self.layout_H.addWidget(self.groupBox_2)
        self.layout_H.addWidget(self.groupBox_3)


        self.layout=QVBoxLayout()
        self.layout.addLayout(self.layout_H)
        self.layout.addWidget(self.buttonBox)
        self.setLayout(self.layout)


class Wood_Dialog(QDialog):
```

```
def __init__(self,*argv,**kwargs):
    super(Wood_Dialog,self).__init__(*argv,**kwargs)
    self.setWindowTitle('设置积木参数')
    self.setWindowIcon(QIcon('../Icon/积木(8).png'))
    self.resize(300,200)

    QBtn=QDialogButtonBox.Ok |QDialogButtonBox.Cancel
    self.buttonBox=QDialogButtonBox(QBtn)
    self.buttonBox.accepted.connect(self.accept)
    self.buttonBox.rejected.connect(self.reject)


    self.machines_label=QLabel("机器数量")
    self.machines_edit=QLineEdit()
    self.top_label=QLabel("上限倍数")
    self.top_edit=QLineEdit()
    self.iter_label=QLabel("迭代次数")
    self.iter_edit=QLineEdit()
    self.rand_radio=QRadioButton("使用随机规则")
    self.mapping_radio=QRadioButton("对应关系")
    self.mount_radio=QRadioButton("山积图")

    self.groupBox_1 = QGroupBox("参数设置")
    layout_1=QVBoxLayout()
    layout_1.addWidget(self.machines_label)
    layout_1.addWidget(self.machines_edit)
    layout_1.addWidget(self.top_label)
    layout_1.addWidget(self.top_edit)
    layout_1.addWidget(self.iter_label)
```

```
        layout_1.addWidget(self.iter_edit)
        self.groupBox_1.setLayout(layout_1)

        self.groupBox_2 = QGroupBox("算法参数")
        layout_2=QVBoxLayout()
        layout_2.addWidget(self.rand_radio)
        self.groupBox_2.setLayout(layout_2)

        self.groupBox_3 = QGroupBox("输出选项")
        layout_3=QVBoxLayout()
        layout_3.addWidget(self.mapping_radio)
        layout_3.addWidget(self.mount_radio)
        self.groupBox_3.setLayout(layout_3)

        #竖直切割布局，放三个参数框
        self.layout_H=QHBoxLayout()
        self.layout_H.addWidget(self.groupBox_1)
        self.layout_H.addWidget(self.groupBox_2)
        self.layout_H.addWidget(self.groupBox_3)



        self.layout=QVBoxLayout()
        self.layout.addLayout(self.layout_H)
        self.layout.addWidget(self.buttonBox)
        self.setLayout(self.layout)

class MainWindow(QMainWindow):
    def __init__(self,*args,**kargs):
        super(MainWindow,self).__init__(*args,**kargs)
```

```
#标题
self.setWindowTitle('U-Line')
#LOGO
self.setWindowIcon(QIcon('../ICON/mainIcon.png'))
self.resize(1100, 600)

#初始化任务类为空
self.task=None

#工具栏：文件拦
File_toolbar=QToolBar('File_toolbar')
self.addToolBar(File_toolbar)
New_action=QAction(QIcon('../ICON/新建.png'),"新建(N)",self)
Open_action=QAction(QIcon('../ICON/打开.png'),"打开(Open)",self)
Save_action=QAction(QIcon('../ICON/保存.png'),"保存(S)",self)
Saveas_action=QAction(QIcon('../ICON/保存(1).png'),"另存为(S)",self)
Out_action=QAction(QIcon('../ICON/保存(2).png'),"输出(Out)",self)
New_action.triggered.connect(self.New_clicked)
Open_action.triggered.connect(self.Open_clicked)
#Open_action.triggered.connect(self.show_input)
Save_action.triggered.connect(self.Save_clicked)
Saveas_action.triggered.connect(self.Saveas_clicked)
Out_action.triggered.connect(self.Out_clicked)
File_toolbar.addAction(New_action)
File_toolbar.addAction(Open_action)
File_toolbar.addAction(Save_action)
File_toolbar.addAction(Saveas_action)
File_toolbar.addAction(Out_action)
```

```
        #工具栏:U 线工具栏
        U_toolbar=QToolBar('U_toolbar')
        self.addToolBar(U_toolbar)
        U_AS_action=QAction(QIcon('../ICON/u(1).png'),"装配导向型(U-AS)",self)
        U_AU_action=QAction(QIcon('../ICON/u(2).png'),"自动化导向型(U-AU)",self)
        U_UP_action=QAction(QIcon('../ICON/u(3).png'),"无优先关系(U-UP)",self)
        U_CC_action=QAction(QIcon('../ICON/u(4).png'),"复杂约束条件(U-CC)",self)
        L_AS_action=QAction(QIcon('../ICON/L(4).png'),"直线型生产线(L-AS)",self)
        L_AU_action=QAction(QIcon('../ICON/L(5).png'),"直线型生产线(L-AU)",self)
        U_AS_action.triggered.connect(self.U_AS_clicked)
        U_AU_action.triggered.connect(self.U_AU_clicked)
        U_UP_action.triggered.connect(self.U_UP_clicked)
        U_CC_action.triggered.connect(self.U_CC_clicked)
        L_AS_action.triggered.connect(self.L_AS_clicked)
        L_AU_action.triggered.connect(self.L_AU_clicked)
        U_toolbar.addAction(U_AS_action)
        U_toolbar.addAction(U_AU_action)
        U_toolbar.addAction(U_UP_action)
        U_toolbar.addAction(U_CC_action)
        U_toolbar.addAction(L_AS_action)
        U_toolbar.addAction(L_AU_action)

        #编辑拦
```

```
        Select_action=QAction(QIcon(),"选择(S)",self)
        Copy_action=QAction(QIcon(),"复制(CP)",self)
        Cut_action=QAction(QIcon(),"剪切(CUT)",self)
        Paste_action=QAction(QIcon(),"粘贴(P)",self)
        Select_action.triggered.connect(self.Select_clicked)
        Copy_action.triggered.connect(self.Copy_clicked)
        Cut_action.triggered.connect(self.Cut_clicked)
        Paste_action.triggered.connect(self.Paste_clicked)

        #设置栏
        Normal_action=QAction(QIcon(),"常规(N)",self)
        Language_action=QAction(QIcon(),"语言(LA)",self)
        Tool_action=QAction(QIcon(),"工具(T)",self)
        #Normal_action.triggered.connect(self.Select_clicked)
        #Language_action.triggered.connect(self.Copy_clicked)
        #Tool_action.triggered.connect(self.Cut_clicked)

        #试图栏：暂时放弃

        #窗口栏：暂时放弃

        #帮助栏
        PDF_action=QAction(QIcon('../ICON/PDF.png'),"软件说明(PDF)",self)
        Contanct_action=QAction(QIcon('../ICON/电话.png'),"联系我们
(COT)",self)
        PDF_action.triggered.connect(self.PDF_clicked)
        Contanct_action.triggered.connect(self.Contanct_clicked)
```

```
##############################
#菜单栏
##############################
#文件
menu=self.menuBar()
menu.setNativeMenuBar(False)
File_menu=menu.addMenu("&文件(F)")
File_menu.addAction(New_action)
File_menu.addAction(Open_action)
File_menu.addAction(Save_action)
File_menu.addAction(Saveas_action)
File_menu.addAction(Out_action)
#编辑
Edit_menu=menu.addMenu("&编辑(E)")
Edit_menu.addAction(Select_action)
Edit_menu.addAction(Copy_action)
Edit_menu.addAction(Cut_action)
Edit_menu.addAction(Paste_action)
#设置栏
Setting_menu=menu.addMenu("&设置(S)")
Setting_menu.addAction(Normal_action)
Setting_menu.addAction(Language_action)
Setting_menu.addAction(Tool_action)

View_menu=menu.addMenu("&视图(V)")
#View_menu.addAction(U_AS_action)
#View_menu.addAction(U_AS_action)
#View_menu.addAction(U_AS_action)
```

```
#View_menu.addAction(U_AS_action)
#View_menu.addAction(U_AS_action)
#View_menu.addAction(U_AS_action)
#View_menu.addAction(U_AS_action)

#分析
Analysis_menu=menu.addMenu("&分析(A)")
Analysis_menu.addAction(U_AS_action)
Analysis_menu.addAction(U_AU_action)
Analysis_menu.addAction(U_UP_action)
Analysis_menu.addAction(U_CC_action)
Analysis_menu.addAction(L_AS_action)
Analysis_menu.addAction(L_AU_action)
#窗口
Window_menu=menu.addMenu("&窗口(W)")
#Window_menu.addAction(U_AS_action)

#帮助
Help_menu=menu.addMenu("&帮助(H)")
Help_menu.addAction(PDF_action)
Help_menu.addAction(Contanct_action)

#表格 1:任务和时间的对应关系
self.time_label=QLabel("任务和时间的对应关系:")
time_header=["任务序号","任务时间"]
self.time_table=QTableWidget()
self.time_table.setRowCount(500)                #行数
self.time_table.setColumnCount(2)               #列数
self.time_table.setHorizontalHeaderLabels(time_header)
```

```
        #表格 2:优先级对应关系
        self.predence_label=QLabel("优先级约束关系")
        predence_header=["前向工序","后向工序"]
        self.predence_table=QTableWidget()
        self.predence_table.setRowCount(500)                    #行数
        self.predence_table.setColumnCount(2)                   #列数
        self.predence_table.setHorizontalHeaderLabels(predence_header)

        #表格 3:输出结果
        self.result_label=QLabel("求解结果")
        result_header=["机器序号","机器时间","任务数","任务序号","任务时间","
强度"]
        self.result_table=QTableWidget()
        self.result_table.setRowCount(100)                      #行数
        self.result_table.setColumnCount(6)                     #列数
        self.result_table.setHorizontalHeaderLabels(result_header)



        #放第一个表格和标签
        layout_1=QVBoxLayout()
        layout_1.addWidget(self.time_label)
        layout_1.addWidget(self.time_table)

        #放第二个表格和标签
        layout_2=QVBoxLayout()
        layout_2.addWidget(self.predence_label)
        layout_2.addWidget(self.predence_table)
```

```
        #放第二个表格和标签
        layout_3=QVBoxLayout()
        layout_3.addWidget(self.result_label)
        layout_3.addWidget(self.result_table)

        #把一和二先放在一起
        layout_half=QHBoxLayout()
        layout_half.addLayout(layout_1)
        layout_half.addLayout(layout_2)

        #放所有表格和标签
        layout_H=QGridLayout()
        #layout_H.setSpacing(10)
        layout_H.addLayout(layout_half,0,0)
        #layout_H.addLayout(layout_2,0,1)
        layout_H.addLayout(layout_3,0,2)
        widget=QWidget()

        widget.setLayout(layout_H)
        self.setCentralWidget(widget)


    def show_input(self):
        print('run show_input')
        #显示数值
        #显示时间
        for i in range(1,self.task.task_nums+1):
            self.time_table.setItem(i-1,0,QTableWidgetItem(str(i)))
```

```
                self.time_table.setItem(i-
1,1,QTableWidgetItem(str(self.task.task_time[i])))
            print(len(self.task.precedence[0]))
            for i in range(len(self.task.precedence[0])):

    self.predence_table.setItem(i,0,QTableWidgetItem(str(int(self.task.precedence[0,i
])))))

    self.predence_table.setItem(i,1,QTableWidgetItem(str(int(self.task.precedence[1,i
])))))
    def show_reslt(self):
        i=1
        while i<=self.para.machine_nums:
            if i==len(self.plan):
                break
            machine=self.plan[str(i)]
            self.result_table.setItem(i-
1,0,QTableWidgetItem(str(machine['number'])))
            self.result_table.setItem(i-
1,1,QTableWidgetItem(str(machine['total_times'])))
            self.result_table.setItem(i-
1,2,QTableWidgetItem(str(machine['total_task_nums'])))
            self.result_table.setItem(i-
1,3,QTableWidgetItem(str(machine['the_task_number'])))
            self.result_table.setItem(i-
1,4,QTableWidgetItem(str(machine['the_task_time'])))
            self.result_table.setItem(i-
1,5,QTableWidgetItem(str(machine['total_times']/self.meter*100)+'%'))
            i+=1
```

```
    def New_clicked(self):
        #新文件
        pass
    def  Open_clicked(self):
        #新文件
        fileName1,filetype=QFileDialog.getOpenFileName(self,
            "选取文件",
            "C:/",
            "IN2 (*.IN2);;Text Files (*.txt)")     #设置文件扩展名过滤,注意用双
分号间隔

        #print(fileName1,filetype)

        df=open(fileName1).readlines()
        task_num=int(df[0])              #获得任务数
        #print(task_num)

        #获得任务时间
        task_time=np.zeros(task_num+1)
        i=1
        while i<task_num+1:
            task_time[i]=int(df[i])
            i+=1


        #获得优先级
```

```
        n=df.index('-1,-1\n')
        precedence = np.zeros([2, n-task_num-1])
        while i<n:
            precedence_port=df[i]                    #precedence 的中间量
            j=precedence_port.index(',')          #j,k 在这里的作用是提出优先阵中
的分隔符‘，’和‘\’
            k=precedence_port.index('\n')
            precedence[0,i-task_num-1]=int(precedence_port[:j])
            precedence[1,i-task_num-1]=int(precedence_port[j+1:k])
            #print(precedence)
            i+=1


        #赋值并返回
        self.task=Task(task_num,task_time,precedence)
        self.show_input()

    def Save_clicked(self):
        #新文件
        pass
    def Saveas_clicked(self):
        #新文件
        pass
    def Out_clicked(self):
        #新文件
        pass

    def Select_clicked(self):
        #编辑
```

```
        pass
    def  Copy_clicked(self):
        #编辑
        pass
    def  Cut_clicked(self):
        #编辑
        pass
    def  Paste_clicked(self):
        #编辑
        pass
    def U_AS_clicked(self):
        #打开蚁群算法参数设置窗口
        self.U_AS_window=Ants_Dialog(self)
        self.U_AS_window.buttonBox.accepted.connect(self.getUASPara)
        #para.
        self.U_AS_window.exec_()
        pass
    def U_AU_clicked(self):
        #打开积木算法参数设置窗口
        self.U_AU_window=Wood_Dialog(self)
        self.U_AU_window.buttonBox.accepted.connect(self.getUAUPara)
        self.U_AU_window.exec_()
        pass
    def U_UP_clicked(self):
        #打开蚁群算法参数设置窗口
        pass
    def U_CC_clicked(self):
        #打开蚁群算法参数设置窗口
        pass
```

```
def L_AS_clicked(self):
    #打开蚁群算法参数设置窗口
    pass
def L_AU_clicked(self):
    #打开蚁群算法参数设置窗口
    pass
def PDF_clicked(self):
    #显示文档
    pass
def Contanct_clicked(self):
    #显示电话
    pass
def getUASPara(self):
    #获得参数
    ants=int(self.U_AS_window.ants_edit.text())
    machines=int(self.U_AS_window.machines_edit.text())
    top=float(self.U_AS_window.top_edit.text())
    ite=int(self.U_AS_window.iter_edit.text())
    alpha=float(self.U_AS_window.alpha_edit.text())
    beta=float(self.U_AS_window.beta_edit.text())
    thero=float(self.U_AS_window.thero_edit.text())
    Q=float(self.U_AS_window.Q_edit.text())
    rand=1#self.U_AS_window.rand_radio.text()

    time_mean=self.task.get_timesum()/machines

    self.para=ANTS_parameter(alpha,beta,thero,Q,ite,ants,machines)
    map_a=initmap(task.task_nums,machine_nums)
    self.plan,self.meter=ANT_STD_ULINE(task,ants,map_a,para)
```

```
        #print('Mainwindow get text is',ants)


    def getUAUPara(self):
        machines=int(self.U_AU_window.machines_edit.text())
        top=float(self.U_AU_window.top_edit.text())
        ite=int(self.U_AU_window.iter_edit.text())
        #rand=self.U_AU_window.rand_radio.isCheck()
        rand=1

        time_total=self.task.get_timesum()

        time_mean=time_total/machines

        self.para=WOOD_parameter(ite,machines,time_mean,rand)
        self.para.set_top_boder(time_mean*top)

        wood=Wood(self.task.task_nums,machines,self.para.top_boder)
        self.plan,self.meter=STD_WOOD(self.task,self.para,wood)
        self.show_reslt()
```