武漢理工大學

数学建模暑期培训论文

第 5 题

基于 NSGA-II 算法的插板式编码多目标优化生产调度模型

第 10 组

姓名	方向
刘子川	编程
程宇	建模
祁成	写作

2025 年 6 月 5 日

摘要

控制高压油管的压力变化对减小燃油量偏差，提高生产计划效率具有重要意义。本文建立了兼顾全局一体化的多目标优化调度模型，采用插板编码的 **NSGA-II 算法**对模型进行求解，并从帕累托前沿解中求得最短工艺生产时间。

针对炼钢、连铸、热轧工艺过程特点，列举模型的约束条件。对于转炉、连铸机与热轧机而言其各炉次、浇次、轧制单元的生产量**存在上下限**。在炼钢过程中每炉次只可使用同种钢材，在连铸过程中每浇次最多可**生产两种型号**的钢坯。对每个轧制单元而言，需存在一定比例的热棍材与主体材，且在轧制顺序上的相邻钢坯硬度、宽度和厚度需保证**单调增长**且不能大幅度变化。

针对问题二，基于问题一整理归纳特点，建立炼钢、连铸、热轧一体化生产多目标优化模型。首先为使得转炉、连铸机和热轧机的闲置产能最低，本节将其各炉次、浇次、轧制单元的空闲产能分别求和并作为**多目标优化**中的目标函数。之后分别根据各炼钢、连铸、热轧的工艺特点，列举每个生产过程中的约束条件。在设计**插板式编码**作为编码方式后，采用 **NSGA-II** 算法求解该多目标优化的**帕累托前沿**，并在帕累托前沿中选择生产时间最低的解作为最终生产方案。

针对一体化调度问题，基于其归纳的特点及约束，建立炼钢、连铸、热轧一体化生产多目标优化模型。首先为使得转炉、连铸机和热轧机的闲置产能最低，本节将其各炉次、浇次、轧制单元的空闲产能分别求和并作为**多目标优化**中的目标函数。之后分别根据各炼钢、连铸、热轧的工艺特点，列举每个生产过程中的约束条件。在设计**插板式编码**作为编码方式后，采用 **NSGA-II 算法**求解该多目标优化的**帕累托前沿**，并在帕累托前沿中选择生产时间最低的解作为最终生产方案。

文中以某钢铁联合企业为研究对象进行仿真测试。仿真过程中 NSGA-II 算法在 CPU Intel T5800 1.6GHz, 2GB 内存环境下进行多次实验。结果表明设置轧制单元个数为 13 个时，经快速非支配排叙最终得到**非支配解有 11 个**最优解，求得在满足约束条件下完成前十个订单的**最短时间为 36580min**。最后本文调整插板个数并做出灵敏度分析，分析结果表示 NSGA-II 算法具有**较强稳定性**。

本文的优点为：1. 本文多维度考虑生产效率对全局的依赖，并在符合严苛的工艺约束条件下全局求得单元计划排列顺序，利用帕累托前沿解大幅缩短工艺生产时间。2. 针对智能算法求解方面，使用 NSGA-II 算法并巧妙采用插板式编码方式对多目标优化求解，兼顾局部搜索与全局搜索能力。

关键词： NSGA-II 算法　插板式编码　帕累托前沿解　多目标优化调度模型

目录

1 问题重述

1.1 问题背景

随着我国经济发展进入新常态，钢铁行业发展环境发生了深刻变化。2015 年，我国钢铁消费与产量双双进入峰值弧顶区并呈下降态势，钢铁主业从微利经营进入整体亏损，行业发展进入“严冬”期。钢铁行业要落实中央经济工作会议精神，着力推进供给侧结构性改革，扎实完成《国务院关于钢铁行业化解过剩产能实现脱困发展的意见》的各项部署和要求。在化解产能过剩的同时，也将注重产业结构优化和升级，做好规范企业动态管理、智能制造试点示范、钢结构建筑推广应用等重点工作，引领钢铁工业提高综合竞争力。

炼钢、连铸和热轧作为钢铁生产的关键工序，工序之间不仅仅是物流与信息流的交换，还存在能量平衡和时间平衡。炼钢一连铸一热轧一体化生产计划编制是通过多个模块、多个模型互相协作来完成，整个生产计划编制过程按照一定的时间长度周期性进行。在每一个计划编制周期内，具体编制流程如图 1 所示。面向订单组织生产管理思想以及炼钢、连铸和热轧一体化排程使各工序的物流控制以及计划之间具有很好的继承性，可以最大限度地减少物料在工序间的停留时间，缩短制造周期和交货周期，还可以减少温降和能量损失、节约能源、降低生产成本。

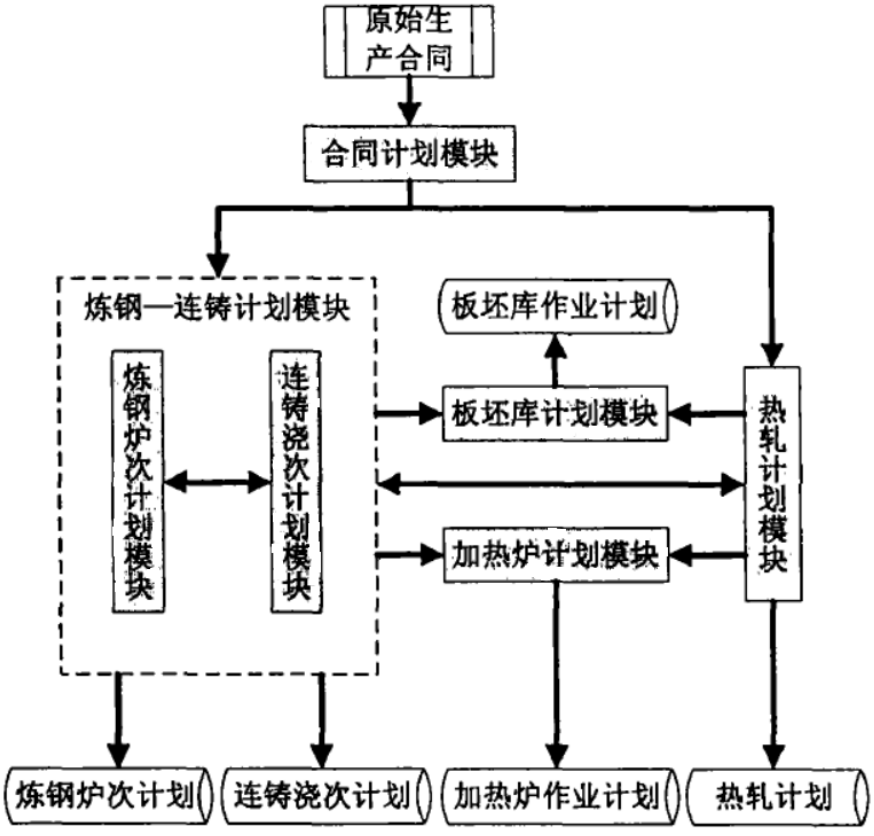


图 **1**　一体化计划编制流程

1.2 问题概述

围绕某钢厂生产数据附件和条件要求，研究炼钢、连铸和热轧一体化生产的调度方案，依次提出以下问题:

问题一：说明炼钢、连铸、热轧一体化生产组织、工艺过程特点，并建立数学模型。

问题二：建立炼钢、连铸、热轧一体化生产计划多目标优化模型，并求解。

2 模型假设

(1) 假设各工序间的成品的物流时间非常短暂，可以忽略不考虑。

(2) 假设每次连铸、热轧的准备时间相同，不会随着工作次数变化而发生变化。

(3) 假设浇铸、热轧每块刚材的所需时间相同，不会因为钢材的型号类型不同而发生变化。

(4) 不考虑制作过程中能量损失，即忽略炼钢、连铸和热轧过程中的材料温度下降。

3 符号说明

符号	说明
c_i, l_i, w_i, t_i, h_i	板坯属性
$c_i', l_i', w_i', t_i', h_i'$	卷钢属性
$Molten_i, Bar_i, Strip_i$	单份钢水、单份板坯、单份卷钢
$Heat_j, Cast_k, Roll_j$	单个炉次、浇次、热轧单元计划
f_1, f_2, f_3	各计划空闲连铸机容量之和
$[n_{alter}]$	变化次数的许用阈值
$[\Delta w]$	板坯宽度的许用变化量
t_{Cast}, t_{Roll}	一次浇铸、轧制操作需要的准备时间
t_{bar}, t_{Strip}	浇铸、轧制一份刚卷需要的时间

注：表中未说明的符号以首次出现处为准

4 问题分析

4.1 问题一的分析

4.1.1 炼钢、连铸、热轧工艺特点分析

钢铁生产是一个高温、高能耗、物流量巨大的加工过程，整个生产流程工艺复杂，工序很多，各工序的工艺约束和生产调度目标也不尽相同[1, 2]。高效实用的钢铁生产调度必须建立在对钢铁生产流程的充分了解基础之上，特别是必须充分考虑到近年来钢铁生产技术的两大飞跃：连铸与热送热装轧制对于生产调度技术的影响[1, 3, 4]。

一个典型的钢铁企业的生产流程如图 2 所示，可分为三大部分：炼铁区域（Iron Making）、主炼钢区域（Primary Steelmaking）和精整加工区域。其中主炼钢包括炼钢、连铸、热轧等工序。

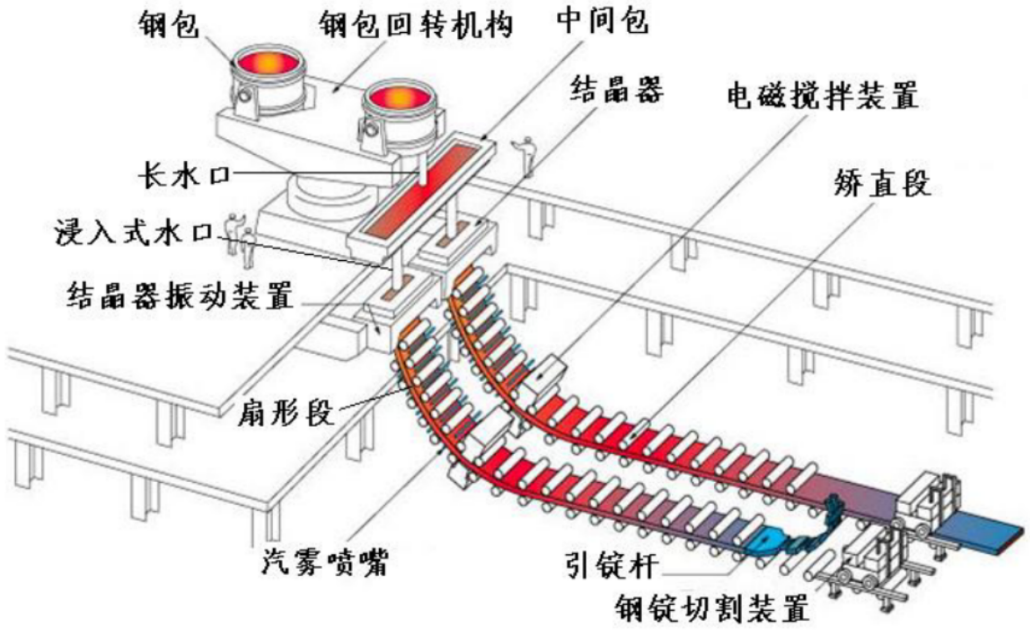


图 **2**　钢铁生产工艺示意图

定义物流传递过程中的主物流等价概念：单份钢水 $Molten_i$、单份板坯 Bar_i、单份卷钢 $Strip_i$，三者是不同表现形式的同一物质；同样可定义工序概念：单个炉次 $Heat_j$、单个浇次 $Cast_k$、单个热轧单元计划 $Roll_j$，三者是同一物质的不同量度。

定义一个单元计划包含的所有板坯的集合为 Bar，单个板坯记为 Bar_i，每个板坯具有属性: 钢种 c_i、长度 l_i、宽度 w_i、厚度 t_i、硬度 h_i。

定义单个卷钢为 $Strip_i$，每个卷钢由板坯经过物理变化得到，视为板坯的变形体，具有与板坯完全相似属性: 钢种 c_i'、长度 l_i'、宽度 w_i'、厚度 t_i'、硬度 h_i'。

不考虑加工过程中物质传递损失，则三种主物流与三种工序的关系为

$$\begin{cases} Heat_j = \sum_{i=1}^{n_{Molten}} Molten_i, \\ Cast_k = \sum_{i=1}^{n_{Bar}} Bar_i, \\ Roll_j = \sum_{i=1}^{n_{Strip}} Strip_i. \end{cases} \tag{1}$$

其中假设多个炉次对应于一个浇次，一个浇次对应于一个热轧单元计划。则有 $Roll_j = Cast_j = \sum_{u=1}^{n_{heat}} Heat_u = \sum_{u=1}^{n_{heat}} Heat_u = \sum_{j=1}^{n_{Heat}} \sum_{i=1}^{n_{Molten}} Molten_{ij}$. 因此，总物流表示为

$$F = \sum_{j=1}^{n_{Roll}} Roll_j = \sum_{j=1}^{n_{Cast}} Cast_j = \sum_{k=1}^{n_{Roll}} \sum_{u=1}^{n_{Heat}} Heat_u = \sum_{h=1}^{n_{Roll}} \sum_{j=1}^{n_{Heat}} \sum_{i=1}^{n_{Molten}} Molten_{ijh}.$$

4.1.2 数学建模需求分析

炼钢、连铸生产动态调度特点：钢铁企业中典型的生产线一般为多座转炉、多台连铸机和一台热轧机顺序生产，如图 3 所示。由于转炉和连铸机都是批量加工的设备，炼钢一连铸生产调度属于一类带有批量约束的并行多机的调度问题[2]。

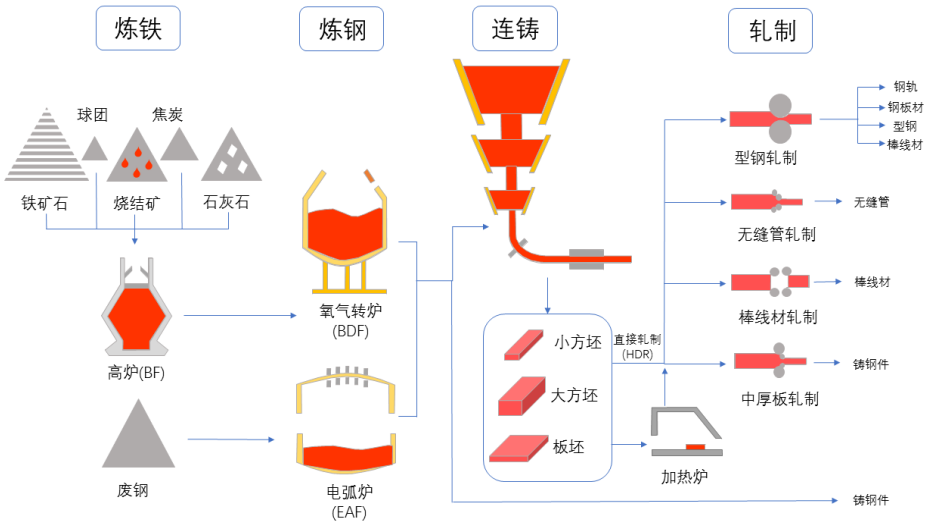


图 3　炼钢、连铸生产流程示意图

文[3] 中提到由于高炉持续不断地向下游工序提供铁水，所以在炼钢一连铸生产中不会出现工件随机到达或到达时间不确定的情况。炼钢一连铸生产动态调度要求根据出现的动态事件，对原有的生产计划进行调整，使得在满足工艺约束的前提下调整后的调度方案在某一指标上达到最优[3]。

本文研究的加工时间可控的炼钢-连铸调度问题具有以下**约束条件**：

(1) 同一生产设备在同一时段只能加工一个炉次；

(2) 相邻炉次之间，后一炉次必须等待前一炉次加工完成后才能开始加工；

(3) 相邻浇次之间，后一阶段的加工开始时刻晚于前一阶段加工完成和设备间运输完成；

(4) 同一炉次内，后一阶段的加工开始时刻晚于前一阶段加工完成和设备间运输完成；

(5) 存在不同程度的扰动，各个炉次在不同生产阶段的加工时间可以在一定范围内调整。

热轧生产计划编制问题特点：精轧机组一般由 6—7 个机架组成连轧，在精轧加工过程中要用到两类轧辊：工作辊和支撑辊，其中工作辊直接和板坯接触，支撑辊对工作辊起到支撑作用[5]。每一个机架上的工作辊和支撑辊由于高温高速轧制，轧辊磨损很大。轧辊磨损到一定程度，**每轧制一定数量的板坯后需要更换轧辊**。为了降低生产成本每个轧制单元轧制的带钢总长度在小于给定的可以轧制的最大带钢轧制长度的前提下要尽可能大[5]。

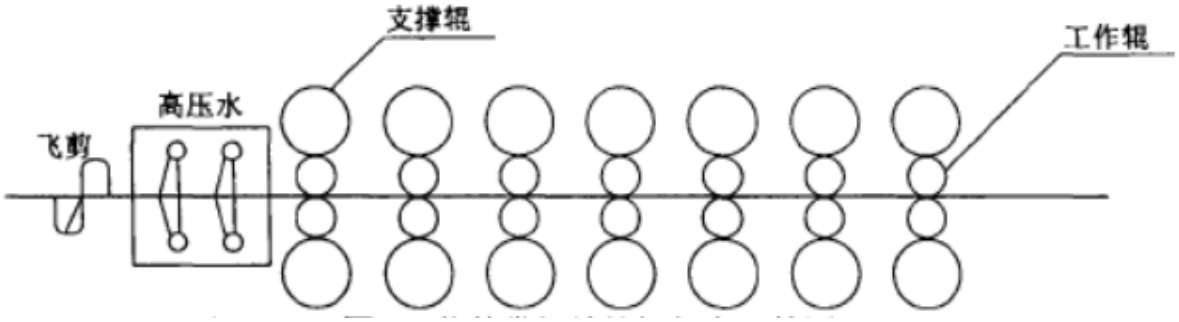


图 **4**　热轧带钢精轧机组布置简图

每个轧制单元由两部分组成：烫辊材和主体材在轧制单元的初始阶段为了使轧辊型达到热平衡和稳定，安排轧制烫辊材。烫辊材在轧制单元中所占比例很小，一般安排硬度较低的带钢。紧跟在烫辊材之后的是主体材，生产合同绝大多数都是在主体材阶段轧制完成。

在编制热轧批量计划时必须遵守如下**工艺约束**：

对于批量计划：烫辊材的宽度递增变化，主体材的宽度递减变化。如图 5 所示，一个完整的的轧制单元的带钢宽度具有“双梯形结构”[6]。

对于烫辊材部分：烫辊材长度有一定限制；烫辊材宽度渐增变化，变化范围有限制轧制的带钢硬度不能太大；硬度变化幅度要小，变化尽量平滑; 厚度变化幅度要小，最好是非减方向变化。

对于主体材部分：主体材总长度有一定限制；主体材宽度非增方向变化，跳跃幅度要小；宽度、厚度、硬度不允许同时跳跃；厚度变化要平稳，不能反复跳跃，同时最好是非减方向变化；硬度变化要平稳，渐近递增或渐近递减均可，只是不能反复跳跃；当三者发生冲突时，优先级为硬度 → 厚度 → 宽度。

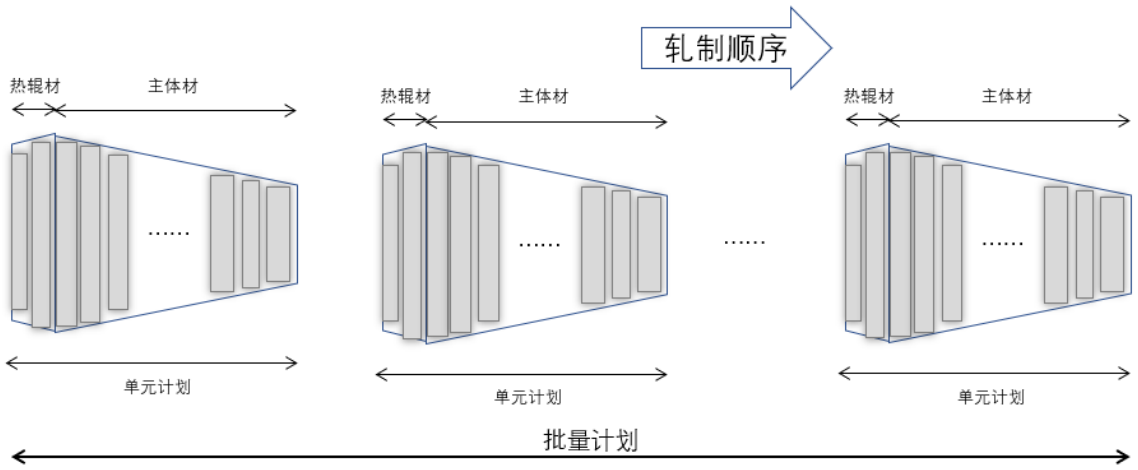


图 **5**　热轧批量计划示意图

根据上述分析, 热轧批量计划编制的目标选取应满足：在保证产品交货期的前提下，保证产品硬度方面的跳跃最小；保证产品厚度方面的跳跃最小；保证产品宽度方面的跳跃最小。

4.2 问题二的分析

4.2.1 一体化调度问题分析

问题二要求建立炼钢、连铸、热轧一体化生产计划数学模型。整理文献资料可知[1, 2, 3]，炼钢、连铸、热轧的工艺特点，一体化调度的主要目标在于优化各设备使用率的同时尽可能降低生产时间，即该一体化调度模型为多目标组合优化模型。

从提高各设备使用率的角度出发，我们将规划各炉次、浇次及轧制单元的生产清单以最大限度的降低设备的闲置产能。由于炼钢、连铸与热轧工艺为顺承关系，本文优先每个规划轧制单元的生产计划，再由此计划逆推各浇次及各炉次的生产清单。其中各轧制单元需存在一定比例的热棍材与主体材，且相邻的轧制板之间的硬度、宽度与厚度的变化不能跳跃式变化。且每个浇次最多可生产两种类型的板材，每个炉次只能生产同种类型的钢水。

当各设备使用率达到最优时，我们认为生产中的非必要能量损耗将降至最低。本文以各设备的闲置产能为多目标优化中的目标函数，通过 $NSGA-II$ 算法求解其对应的帕累托前沿，并在帕累托前沿中选取总生产时间最小的解作为最终方案。

4.2.2 多目标优化需求分析

为降低生产成本，我们将尽可能降低每次生产时的闲置产能。炼钢阶段将优化组炉的空闲炉容量，对每炉次钢水，尽可能降低该炉次的空闲炉容量，即

$$minf_1 = \sum_{v=1}^{n_{heat}}(V_1 - \sum_{u=1}^{n_{Molten}} Molten_u), \tag{2}$$

其中 V_1 表示转炉最大容量，n_{Heat} 表示总炉次，n_{Molten} 表示炉次 v 加工的钢水份数，$Molten_u$ 表示炉次 v 加工的单份钢水的重量。即 f_1 表示每炉次的空闲炉容量之和。同理在连铸阶段时，有

$$minf_2 = \sum_{v=1}^{n_{cast}}(V_2 - \sum_{u=1}^{n_{Bar}} Bar_u), \tag{3}$$

其中 V_2 表示连铸机最大容量，n_{cast} 表示总浇次，n_{Bar} 表示浇次 v 加工的钢坯份数，Bar_u 表示浇次 v 加工的第 u 份钢坯的重量。即 f_2 表示每浇次的空闲连铸机容量之和。同理在热轧阶段时，有

$$minf_3 = \sum_{v=1}^{n_{Roll}}(V_3 - \sum_{u=1}^{n_{Strip}} Strip_u), \tag{4}$$

其中 V_3 表示热轧机最大轧制长度，n_{Roll} 表示总轧制单元数，n_{Strip} 表示浇次 v 加工的钢卷份数，$Strip_u$ 表示轧制单元 v 加工的第 u 份钢卷的厚度。即 f_3 表示每个轧制单元的空闲轧制长度之和。空闲产能优化实意图如图 6 所示

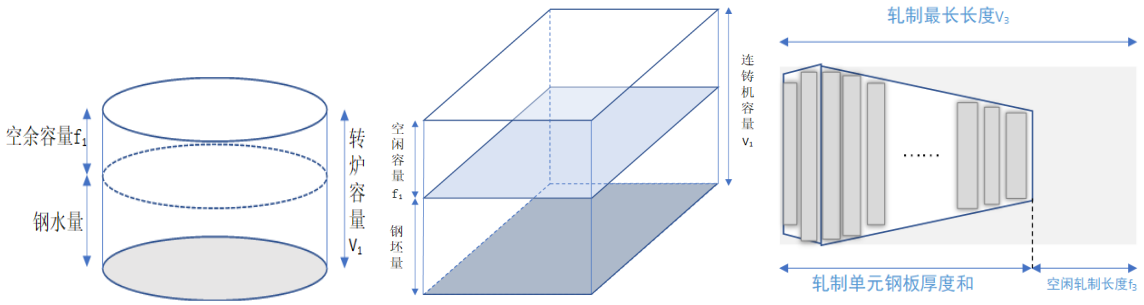


图 **6**　空闲产能示意图

即产能优化目标可表示为

$$min\begin{cases} f_1 = \sum_{v=1}^{n_{heat}}(V_1 - \sum_{u=1}^{n_{Molten}} Molten_u), \\ f_2 = \sum_{v=1}^{n_{cast}}(V_2 - \sum_{u=1}^{n_{Bar}} Bar_u), \\ f_3 = \sum_{v=1}^{n_{Roll}}(V_3 - \sum_{u=1}^{n_{Strip}} Strip_u). \end{cases} \tag{5}$$

为最大限度的减少物料在工序间的停留时间，本文将在产能目标函数达到最优的情况下尽使得热轧时间与浇铸时间达到之和为最小值。由附件 1 可知，共有两台连铸机，即连铸机工作总时间 T_{cast} 可表示为

$$T_{Cast} = \frac{1}{2}\sum_{i=1}^{n_{Cast}}(n_{Bar} \times t_{Bar} + t_{Cast}),$$

其中 t_{Cast} 表示一次浇铸操作需要的准备时间，t_{bar} 表示浇铸每块钢坯需要的时间。同理热轧机的工作总时间 T_{roll} 可表示为

$$T_{Roll} = \sum_{i=1}^{n_{Roll}}(n_{Strip} \times t_{Strip} + t_{Roll}),$$

其中 t_{Roll} 表示一次轧制需要的准备时间，t_{Strip} 表示轧制一份刚卷需要的时间。由于转炉总数未知，且一般炼钢厂转炉数量较多，我们将不把炼钢总时间归入时间目标函数，即时间目标函数 T 可表示为

$$minT = T_{Cast} + T_{Roll}. \tag{6}$$

本文将优先计算使得产能目标函数最优时的帕累托解平面，然后在解平面中选择时间目标函数最优解作为最终生产方案。

5 模型建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型建立

炼钢、连铸、热轧工艺特点模型 根据前述分析，炼钢、连铸、热轧工艺特点可通过物流和工序定义进行建模。物流传递过程中的主物流等价概念为单份钢水 $Molten_i$、单份板坯 Bar_i、单份卷钢 $impegno_i$，三者是同一物质的不同表现形式。工序概念为单个炉

次 $Heat_j$、单个浇次 $Cast_k$、单个热轧单元计划 $Roll_j$。板坯属性包括钢种 c_i、长度 l_i、宽度 w_i、厚度 t_i、硬度 h_i，卷钢属性与之类似。

物流与工序关系为：

$$\begin{cases} Heat_j = \sum_{i=1}^{n_{Molten}} Molten_i, \\ Cast_k = \sum_{i=1}^{n_{Bar}} Bar_i, \\ Roll_j = \sum_{i=1}^{n_{Strip}} Strip_i. \end{cases} \tag{7}$$

总物流表示为：

$$F = \sum_{j=1}^{n_{Roll}} Roll_j = \sum_{j=1}^{n_{Cast}} Cast_j = \sum_{k=1}^{n_{Roll}} \sum_{u=1}^{n_{Heat}} Heat_u = \sum_{h=1}^{n_{Roll}} \sum_{j=1}^{n_{Heat}} \sum_{i=1}^{n_{Molten}} Molten_{ijh}.$$

约束条件 炼钢-连铸调度约束包括：(1) 同一生产设备同一时段只能加工一个炉次；(2) 相邻炉次需等待前一炉次加工完成；(3) 相邻浇次加工开始晚于前一阶段完成及运输；(4) 同一炉次内后一阶段加工开始晚于前一阶段完成及运输；(5) 加工时间可调整。

热轧工艺约束包括：- 烫辊材宽度递增，主体材宽度递减；- 烫辊材长度、硬度、厚度变化受限；- 主体材宽度、厚度、硬度变化平稳，优先级为硬度 → 厚度 → 宽度。

5.1.2 模型求解

求解方法 问题一的模型主要为工艺特点的数学描述，求解通过后续多目标优化模型实现，具体方法在问题二中阐述。

5.2 问题二模型的建立与求解

5.2.1 模型建立

多目标优化模型 多目标优化模型以最小化转炉、连铸机、热轧机的闲置产能为目标，目标函数为：

$$min \begin{cases} f_1 = \sum_{v=1}^{n_{heat}} (V_1 - \sum_{u=1}^{n_{Molten}} Molten_u), \\ f_2 = \sum_{v=1}^{n_{cast}} (V_2 - \sum_{u=1}^{n_{Bar}} Bar_u), \\ f_3 = \sum_{v=1}^{n_{Roll}} (V_3 - \sum_{u=1}^{n_{Strip}} Strip_u). \end{cases} \tag{8}$$

时间目标函数为：

$$minT = T_{Cast} + T_{Roll}, \tag{9}$$

其中：

$$T_{Cast} = \frac{1}{2}\sum_{i=1}^{n_{Cast}}(n_{Bar} \times t_{Bar} + t_{Cast}),$$

$$T_{Roll} = \sum_{i=1}^{n_{Roll}}(n_{Strip} \times t_{Strip} + t_{Roll}).$$

插板式编码设计　插板式编码用于降低决策变量体量。对于轧制单元数量 n，板坯种类总数 m，编码矩阵为：

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n-1} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,n-1} \end{bmatrix},$$

其中 $x_{i,j} \in [0, N_i]$，$x_{i,j} \leqslant x_{i,j+1}$，$N_i$ 为第 i 种板坯的生产总数。编码示意图如图 7 所示。

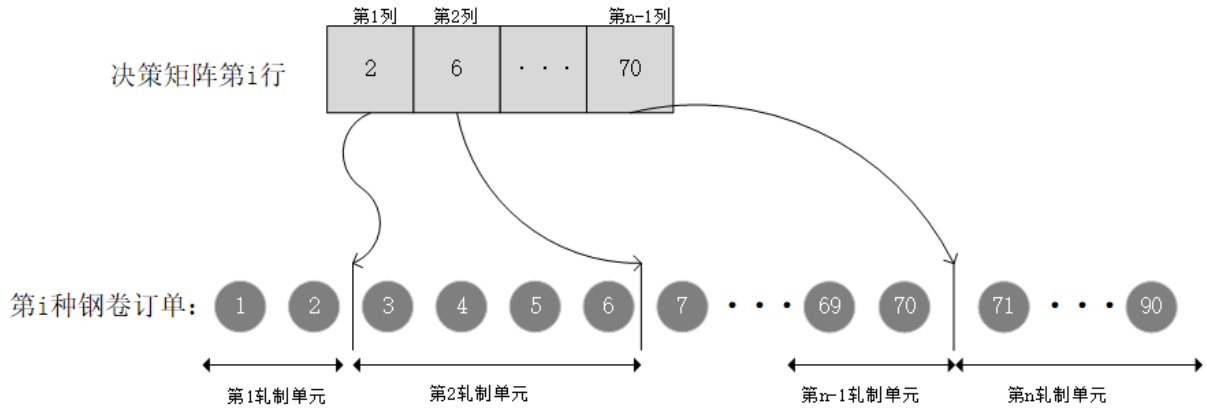


图 **7**　插板式编码示意图

工艺约束条件

组炉约束 组炉约束为：

$$\begin{cases} c_i = c_0, \\ t_i = t_0, \\ w_i = w_0 \geq 140, \\ w_i' \leq w_i \leq w_i' + 100, \\ 95\% \cdot \sum_{u=1}^{n_{Heat}} Heat_u \leq \sum_{i=1}^{n_{Bar}} Bar_i \leq \sum_{u=1}^{n_{Heat}} Heat_u. \end{cases} \tag{10}$$

组浇约束 组浇约束为：

$$\begin{cases} w_{j+1} \leq w_j, \\ |\Delta w_j| \leq [\Delta w], \\ \sum_{j=1}^{n_w - 1} [\frac{|\Delta w_j|}{[|\Delta w_j|]}] \leq [n_{alter}], \end{cases} \tag{11}$$

炉次间钢级关系如表 1 所示。

表 **1** 炉次间的钢级关系

炉次[1]	C_1	C_2	C_3	C_4
C_1	0	1	2	3
C_2	1	0	3	2
C_3	2	3	0	3
C_4	3	2	3	0

[1] C_i 是第 i 个炉次对应的钢级;0-钢级完全相同可直接连浇;1—钢级相近不用隔板, 可直接连浇;2—用隔板进行质量分离连浇;3-不能连浇。

热轧单元约束 热轧单元约束为：

$$\begin{cases} \Delta t_j \cdot \Delta t_{j+1} \geq 0, \\ \Delta h_j \cdot \Delta h_{j+1} \geq 0, \\ \sum_{j=1}^{n} Roll_j \leq B_0, \\ \sum_{j}^{n-1} \Delta w_i \leq W_0, \\ \sum_{j}^{n-1} \Delta t_i \leq T_0, \\ 0 \leq \Delta t_j = t_j - t_{j+1} \leq [\Delta t], \\ 0 \leq \Delta w_j = w_j - w_{j+1} \leq [\Delta w], \\ [\frac{|\Delta w_j|}{[|\Delta w_j|]}] + [\frac{|\Delta t_j|}{[|\Delta t_j|]}] + [\frac{|\Delta h_j|}{[|\Delta h_j|]}] \leq 1. \end{cases} \tag{12}$$

5.2.2 模型求解

NSGA-II 算法　NSGA-II 算法基于遗传算法与快速非支配排序，流程图如图 8 所示。

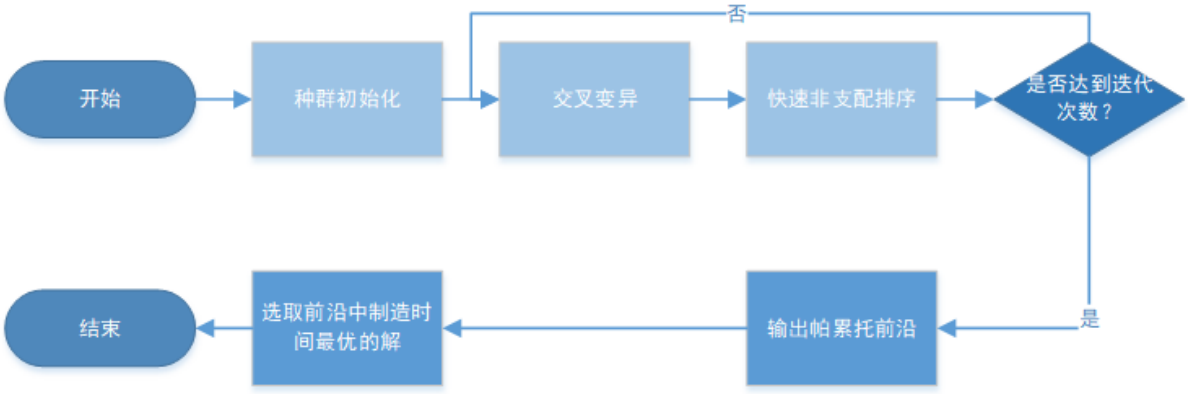


图 **8**　**NSGA-II** 算法流程图

种群初始化　随机生成 p 个满足约束条件的解，编码因子为：

$$x_{q,i,j} = floor(rand(0, N_i + 1)),$$

按行降序排列：

$$[x_{q,i,1}, x_{q,i,2}, \cdots, x_{q,i,n-1}] \leftarrow sort([x_{q,i,1}, x_{q,i,2}, \cdots, x_{q,i,n-1}]).$$

交叉变异操作　中间数交叉：

$$x_{c,i,j} = x_{a,i,j} + floor((x_{b,i,j} - x_{a,i,j}) \times rand(0, 1)).$$

变异操作为：

$$x_{q,i,j}(k) = floor(rand(0, N_i + 1)).$$

快速非支配排序　定义解 X_a 支配 X_b 若：

$$\forall v \in \{1, 2, 3\} : f_v(X_a) \leqslant f_v(X_a) \wedge \exists v \in \{1, 2, 3\} : f_v(X_a) < f_v(X_a).$$

拥挤度计算为：

$$C(X_q) = \sum_{v \in \{1,2,3\}} \frac{f_v(X_{q+1}) - f_v(X_{q-1})}{f_{v,max}\{X\}_k(i) - f_{v,min}\{X\}_k(i)}.$$

循环迭代寻优与选解　迭代 G 次后输出第 1 非支配层作为帕累托前沿，选择锻造时间最小的解。

求解结果与分析　仿真结果表明，设置轧制单元个数为 13 时，得到 11 个非支配解，最短时间为 36580min，插板方式如表 2 所示。

表 **2**　求得最短时间热轧单元计划插板方式

套餐个数 \ 订单号	001	002	003	…	009	010
轧制单元 1	43	11	54	…	12	29
轧制单元 2	52	30	109	…	32	62
轧制单元 3	145	88	147	…	47	100
轧制单元 4	236	101	198	…	61	116
…	…	…	…	…	…	…
轧制单元 9	558	167	508	…	113	350
轧制单元 10	609	188	593	…	113	362
轧制单元 11	626	191	648	…	113	362
轧制单元 12	629	193	656	…	114	336

6 仿真实验与结果分析

6.1 实验数据及参数设置

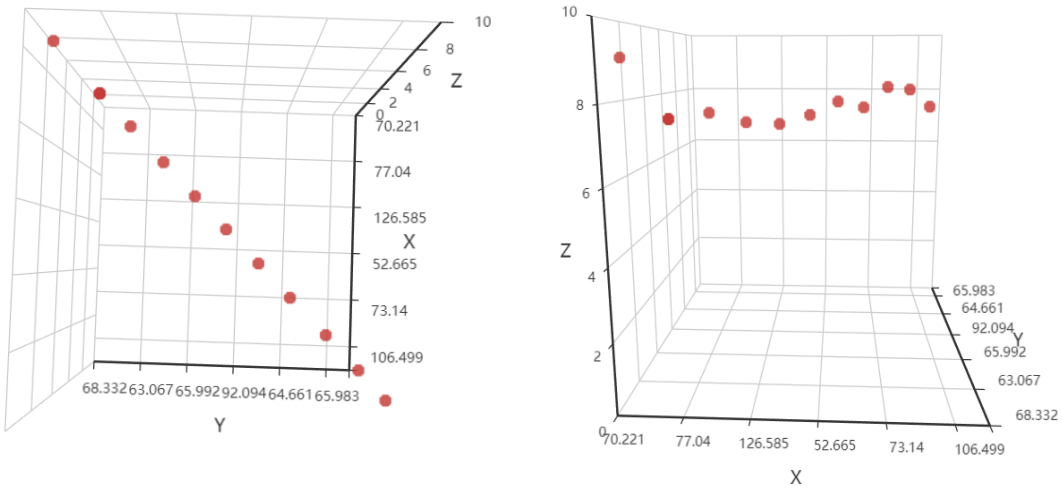
仿真使用 Python 实现，NSGA-II 参数设置如表 3 所示：

表 **3**　实验参数设置表

参数名称	参数值	参数名称	参数值
交叉率	0.5	变异率	0.8
迭代次数	500	种群大小	200
惩罚值	10000	突变领域	10
边界拥挤距离	inf	订单数目	10
连铸设置时间	20min	热轧设置时间	10min
轧制一块板胚时间	2min	浇铸一块板胚时间	5min

6.2 实验结果

算法在 10min 内生成 13 个轧制单元计划、63 个炉次计划和 7 个浇次计划，得到 11 个非支配解，如图 9 所示。



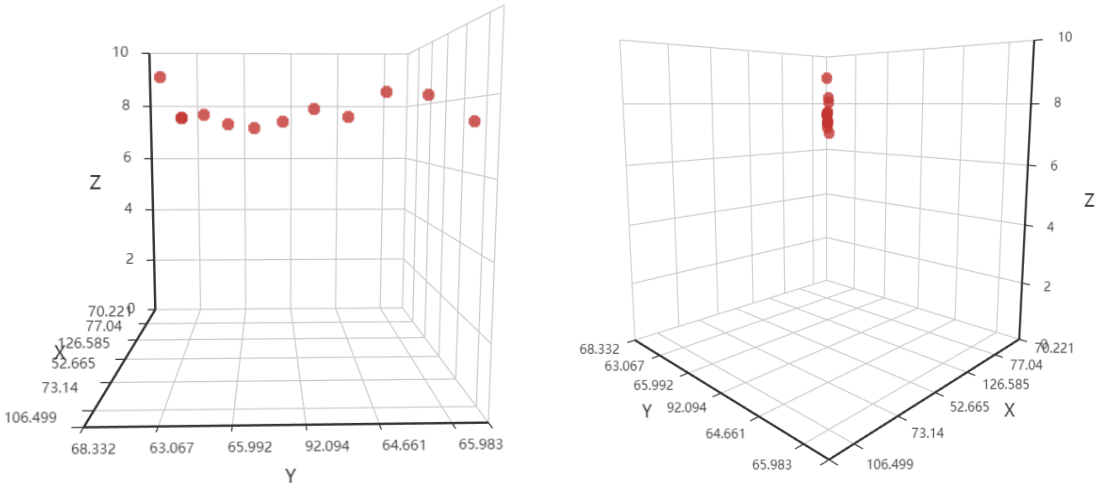


图 **9**　三维 **Pareto** 最优解散点各平面分布（缩放比例 **1**：**100**）

最短时间为 36580min，宽度和厚度变化满足工艺约束，如图 10 所示。

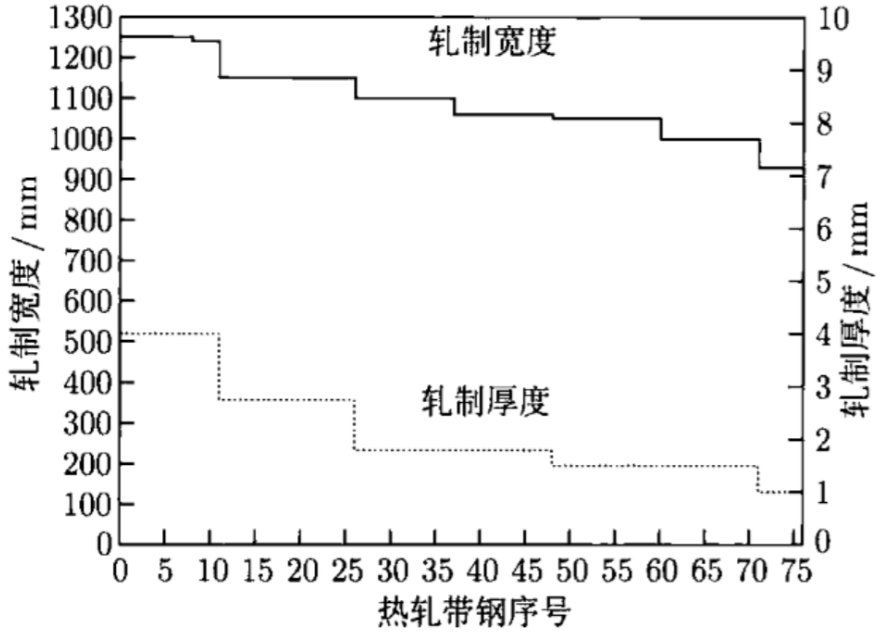


图 **10**　某一轧制单元宽度和厚度变化

7 灵敏度分析

改变插板个数在约束范围内波动 5%，观察非劣解变化。结果如图 11 所示，NSGA-II 算法具有较强稳定性。

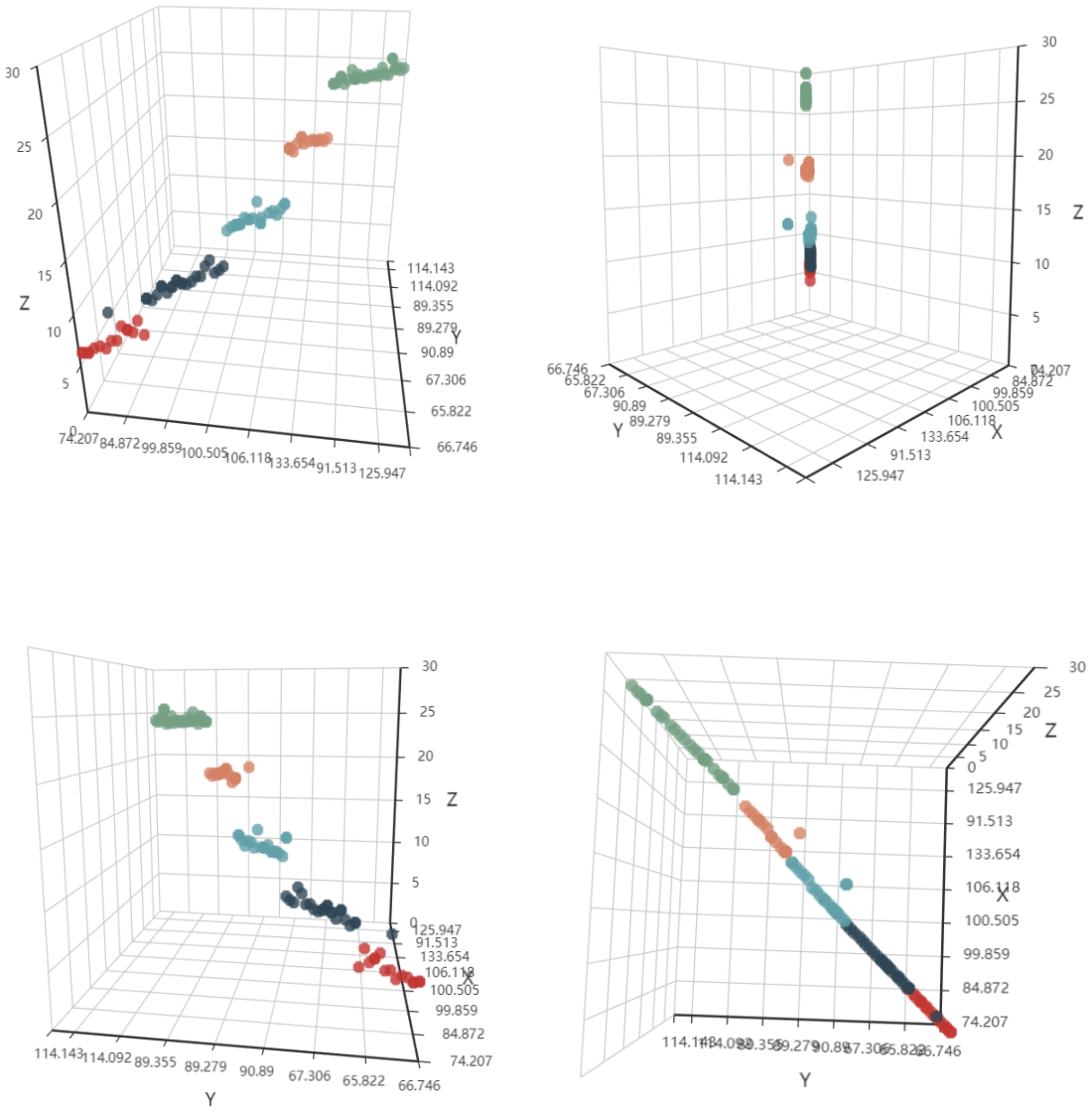


图 **11**　改变插板个数对非劣解的影响（缩放比例 **1**：**100**）

最短生产时间随插板个数变化如图 12 所示，插板数为 12 时停留时间最短。

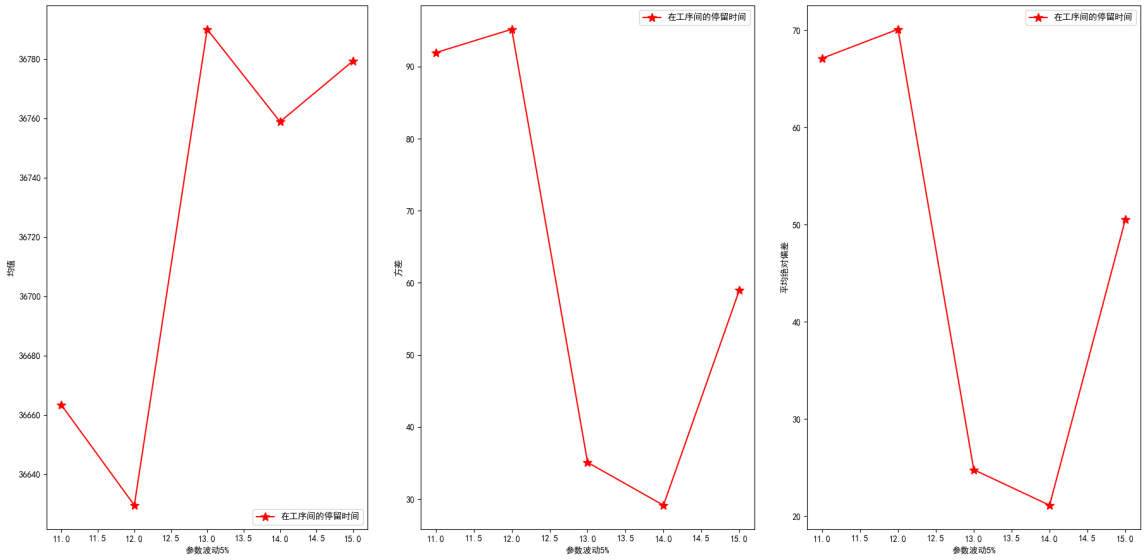


图 **12**　隔板参数变化时灵敏度分析

8 模型评价

8.1 模型的优点

1. 多维度考虑生产效率对全局的依赖，满足严苛工艺约束，优化单元计划排列。
2. 使用 NSGA-II 算法结合插板式编码，兼顾局部和全局搜索能力。
3. 选择帕累托前沿中锻造时间最小的解，缩短制造和交货周期。

8.2 模型的缺点

1. 未考虑温降和能量损失对生产计划的影响。
2. 约束条件严苛，NSGA-II 需较大种群数，影响搜索能力。

参考文献

[1] 李文兵, 毕英杰, 纪扬, 等. 基于生产流程的炼钢-连铸-热轧仿真系统 [J]. 冶金自动化, 2005, 29(3): 37-42.

[2] Funakawa Y, Shiozaki T, Tomita K, et al. Development of high strength hot-rolled sheet steel consisting of ferrite and nanometer-sized carbides[J]. ISIJ international, 2014, 44(11): 1945-1951.

[3] 张春生, 李铁克. 炼钢与热轧调度方案动态协调方法研究 [J]. 冶金自动化, 2016 (5): 19-25.

[4] 朱宝琳, 于海斌. 炼钢—连铸—热轧生产调度模型及算法研究 [J]. 计算机集成制造系统, 2003, 9(1): 33-36.

[5] 李耀华, 胡国奋, 王伟, 等. 炼钢-连铸-热轧一体化生产计划编制方法研究 [J]. 控制工程, 2005, 12(6): 549-552.

[6] 李铁克, 苏志雄. 炼钢连铸生产调度问题的两阶段遗传算法 [J]. 中国管理科学, 2009, 17(5): 68-74.

附录 A NSGA-II 仿真实验代码

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1l1aQ3-BhBtZA9jlKpqy1K5GYrnIznm6r
"""

import pandas as pd
import numpy as np
import random


data = pd.read_excel("/content/drive/My
    Drive/competitions/CMCM/demo5/data.xls")
data.head()

# coding:utf-8
# Importing required modules
import math
import random
import numpy as np
import pandas as pd
from tqdm.notebook import tqdm
```

```
25 import matplotlib.pyplot as plt
26
27 plt.rcParams["font.sans-serif"] = ["SimHei"] # 用来正常显示中文标签
28 plt.rcParams["axes.unicode_minus"] = True # 用来正常显示负号
29
30
31 def missing_data(data):
32     total = data.isnull().sum()
33     percent = data.isnull().sum() / data.isnull().count() * 100
34     tt = pd.concat([total, percent], axis=1, keys=["Total", "Percent"])
35     types = []
36     for col in data.columns:
37         dtype = str(data[col].dtype)
38         types.append(dtype)
39     tt["Types"] = types
40     return np.transpose(tt)
41
42
43 def unique_values(data):
44     total = data.count()
45     tt = pd.DataFrame(total)
46     tt.columns = ["Total"]
47     uniques = []
48     for col in data.columns:
49         unique = data[col].nunique()
50         uniques.append(unique)
51     tt["Uniques"] = uniques
52     return np.transpose(tt)
53
54
55 def most_frequent_values(data):
56     total = data.count()
57     tt = pd.DataFrame(total)
58     tt.columns = ["Total"]
59     items = []
```

```
    vals = []
    for col in data.columns:
        itm = data[col].value_counts().index[0]
        val = data[col].value_counts().values[0]
        items.append(itm)
        vals.append(val)
    tt["Most frequent item"] = items
    tt["Frequence"] = vals
    tt["Percent from total"] = np.round(vals / total * 100, 3)
    return np.transpose(tt)


def plot_count(feature, title, df, size=1, num=20):
    f, ax = plt.subplots(1, 1, figsize=(4 * size, 4))
    total = float(len(df))
    g = sns.countplot(df[feature],
        order=df[feature].value_counts().index[:num], palette="Set3")
    g.set_title("Number and percentage of {}".format(title),
        fontproperties=font)
    if size > 2:
        plt.xticks(rotation=90, size=8)
    for p in ax.patches:
        height = p.get_height()
        ax.text(
            p.get_x() + p.get_width() / 2.0,
            height + 3,
            "{:1.2f}%".format(100 * height / total),
            ha="center",
            fontproperties=font,
        )
    plt.show()


unique_values(data)

```

```
dingdan = pd.read_excel("/content/drive/My Drive/competitions/CMCM/demo5/data_un.xlsx")
del dingdan["Unnamed: 0"]
del dingdan["虚拟板坯号"]
del dingdan["钢卷ID号"]

dingdan.head(5)


class Loader(object):
    def __init__(self, id, num, water_w, board_w, W, D, H, is_Q, is_Tang, gangzhong):
        self.id = id # 订单号
        self.num = num # 板胚个数
        self.water_w = water_w # 钢水重量
        self.board_w = board_w # 钢板重量
        self.W = W # 宽度
        self.D = D # 厚度
        self.H = H # 硬度
        self.is_Tanggun = is_Q != 2 and is_Tang == 1 # 是否为烫辊材
        self.gangzhong = gangzhong # 钢种


loaders = []
# num_loader = len(dingdan)
num_loader = 10
for i in range(num_loader):
    l = Loader(
        dingdan.loc[i]["订单号"],
        dingdan.loc[i]["数目"],
        dingdan.loc[i]["钢水重量"],
        dingdan.loc[i]["板坯重量"],
        dingdan.loc[i]["坯料宽度"],
        dingdan.loc[i]["坯料厚度"],
        dingdan.loc[i]["板坯硬度"],
```

```
        dingdan.loc[i]["表面质量"],
        dingdan.loc[i]["是否可为烫辊材"],
        dingdan.loc[i]["冶炼牌号"],
    )
    loaders.append(l)

# pip install pyecharts


# Function to carry out NSGA-II's fast non dominated sort
# 函数执行NSGA-II的快速非支配排序,将所有的个体都分层
"""
1.np=0 sp=infinite
2.对所有个体进行非支配判断，若p支配q，则将q加入到sp中，
并将q的层级提升一级。
  若q支配p，将p加入sq中，并将p的层级提升一级。
3.对种群当前分层序号k进行初始化，令k=1
4.找出种群中np=0的个体，将其从种群中移除，将其加入到分层
集合fk中，该集合就是层级为0个体的集合。
5.判断fk是否为空，若不为空，将fk中所有的个体sp中对应的个
体层级减去1，且k=k+1,跳到2;
  若为空，则表明得到了所有非支配集合，程序结束
"""
"""基于序列和拥挤距离,这里找到任意两个个体p,q"""
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


def fast_non_dominated_sort(values1, values2, values3):
    S = [[] for i in range(0, len(values1))]
    # 种群中所有个体的sp进行初始化 这里的len(value1)=pop_size
    front = [[]]
    # 分层集合,二维列表中包含第n个层中,有那些个体
    n = [0 for i in range(0, len(values1))]
    rank = [0 for i in range(0, len(values1))]
```

```
161     # 评级
162
163     for p in range(0, len(values1)):
164         S[p] = []
165         n[p] = 0
166         # 寻找第p个个体和其他个体的支配关系
167         # 将第p个个体的sp和np初始化
168         for q in range(0, len(values1)):
169             # step2:p > q 即如果p支配q,则
170             if (
171                 (values1[p] > values1[q] and values2[p] > values2[q] and
                        values3[p] > values3[q])
172                 or (
173                     values1[p] >= values1[q]
174                     and values2[p] > values2[q]
175                     and values3[p] >= values3[q]
176                 )
177                 or (
178                     values1[p] > values1[q]
179                     and values2[p] >= values2[q]
180                     and values3[p] >= values3[q]
181                 )
182                 or (
183                     values1[p] >= values1[q]
184                     and values2[p] >= values2[q]
185                     and values3[p] > values3[q]
186                 )
187             ):
188                 # 支配判定条件:当且仅当,对于任取i属于{1,2},
189                 # 都有fi(p)>fi(q),符合支配.
190                 # 或者当且仅当对于任意i属于{1,2},有fi(p)>=
191                 # fi(q),且至少存在一个j使得fj(p)>fj(q)符合弱支配
192                 if q not in S[p]:
193                     # 同时如果q不属于sp将其添加到sp中
194                     S[p].append(q)
```

```
            # 如果q支配p
            elif (
                (values1[q] > values1[p] and values2[q] > values2[p] and
                   values3[q] > values3[p])
                or (
                    values1[q] >= values1[p]
                    and values2[q] > values2[p]
                    and values3[q] >= values3[p]
                )
                or (
                    values1[q] > values1[p]
                    and values2[q] >= values2[p]
                    and values3[q] >= values3[p]
                )
                or (
                    values1[q] >= values1[p]
                    and values2[q] >= values2[p]
                    and values3[q] > values3[p]
                )
            ):
                # 则将np+1
                n[p] = n[p] + 1
        if n[p] == 0:
            # 找出种群中np=0的个体
            rank[p] = 0
            # 将其从pt中移去
            if p not in front[0]:
                # 如果p不在第0层中
                # 将其追加到第0层中
                front[0].append(p)
    i = 0
    while front[i] != []:
        # 如果分层集合为不为空,
        Q = []
        for p in front[i]:
```

```
            for q in S[p]:
                n[q] = n[q] - 1
                # 则将fk中所有给对应的个体np-1
                if n[q] == 0:
                    # 如果nq==0
                    rank[q] = i + 1
                    if q not in Q:
                        Q.append(q)
        i = i + 1
        # 并且k+1
        front.append(Q)
    del front[len(front) - 1]
    return front
    # 返回将所有个体分层后的结果


# Function to find index of list
# 查找列表指定元素的索引
def index_of(a, list):
    for i in range(0, len(list)):
        if list[i] == a:
            return i
    return -1


# Function to sort by values
# 函数根据指定的值列表排序
"""list1=[1,2,3,4,5,6,7,8,9]
   value=[1,5,6,7]
   sort_list=[1,5,6,7]
"""


def sort_by_values(list1, values):
    sorted_list = []
```

```
    while len(sorted_list) != len(list1):
        # 当结果长度不等于初始长度时，继续循环
        if values.index(min(values)) in list1:
            # 标定值中最小值在目标列表中时
            sorted_list.append(values.index(min(values)))
        #     将标定值的最小值的索引追加到结果列表后面
        values[values.index(min(values))] = math.inf
    #      将标定值的最小值置为无穷小,即删除原来的最小值,移向下一个
    #     infinited
    return sorted_list


# Function to calculate crowding distance
# 计算拥挤距离的函数
"""
1.I[1]=I[l]=inf，I[i]=0 将边界的两个个体拥挤度设为无穷。
2.I=sort(I,m)，基于目标函数m对种群排序
3.I[i]=I[i]+(Im[i+1]-Im[i-1])/(fmax-fmin)
"""


def crowding_distance(values1, values2, values3, front):
    distance = [0 for i in range(0, len(front))]
    # 初始化个体间的拥挤距离
    sorted1 = sort_by_values(front, values1[:])
    sorted2 = sort_by_values(front, values2[:])
    sorted3 = sort_by_values(front, values3[:])

    # 基于目标函数1和目标函数2对已经划分好层级的种群排序
    distance[0] = 4444444444444444
    distance[len(front) - 1] = 4444444444444444
    for k in range(1, len(front) - 1):
        distance[k] = distance[k] + (values1[sorted1[k + 1]] -
            values1[sorted1[k - 1]]) / (
            max(values1) - min(values1)
```

```
        )
    for k in range(1, len(front) - 1):
        distance[k] = distance[k] + (values2[sorted2[k + 1]] -
            values2[sorted2[k - 1]]) / (
            max(values2) - min(values2)
        )
    for k in range(1, len(front) - 1):
        distance[k] = distance[k] + (values3[sorted3[k + 1]] -
            values3[sorted3[k - 1]]) / (
            max(values3) - min(values3)
        )
    return distance
    # 返回拥挤距离


class NSGA(object):
    def __init__(
        self, loaders, partition_num=None, max_num=20, pop_size=10,
            c_rate=0.5, m_rate=0.8
    ):
        self.partition_num = partition_num # 隔板数
        self.max_num = max_num # 迭代次数
        self.loaders = loaders # 订单们
        self.pop_size = pop_size # 种群数目
        self.c_rate = c_rate # 交换率
        self.m_rate = m_rate # 突变率
        pass

    def encode(self): # 初始化编码，问题的解
        gens = []
        if self.partition_num != None:
            for x in range(self.partition_num): # 隔板的每一个基因
                gen = []
                for l in self.loaders:
                    gen.append(random.randint(0, l.num - 1)) #
```

```
                    初始化每一个隔板的位置
330             gens.append(gen)
331         gens = np.sort(gens, axis=0) # 按列排叙。列为单个订单的插板
332         return np.array(gens)
333
334     def creat_pop(self, size):
335         pop = []
336         for i in range(size):
337             pop.append(self.encode()) # 加入种群
338         return pop
339
340     def cross(self, parent1, parent2):
341         """交叉p1,p2的部分基因片段"""
342         # if np.random.rand() > self.c_rate:
343         #     return parent1
344         newGene = np.zeros((parent1.shape[0], parent1.shape[1]),
                dtype=np.int)
345         for i in range(parent1.shape[1]): # 交叉，这里待优化
346             for j in range(parent1.shape[0]):
347                 # if parent2[j][i]>=parent1[j][i]:
348                 #   newGene[j][i] = random.randint(parent1[j][i],
                        parent2[j][i]) # 取两个父代中间的随机数
349                 # else:
350                 #   newGene[j][i] = random.randint(parent2[j][i],
                        parent1[j][i]) # 取两个父代中间的随机数
351
352                 if np.random.rand() > self.c_rate:
353                     newGene[j][i] = parent1[j][i] # 取两个父代中间的随机数
354                 else:
355                     newGene[j][i] = parent2[j][i] # 取两个父代中间的随机数
356         newGene = np.sort(newGene, axis=0) # 按列排叙。列为单个订单的插板
357         return newGene
358
359     def mutate(self, gene):
360         """突变"""
```

```
        if np.random.rand() > self.m_rate:
            return gene
        newGene = gene.copy()
        lingyu = 10 # 领域，在小范围变化
        for i in range(gene.shape[1]): # 交叉，这里待优化
            for j in range(gene.shape[0]):
                if np.random.rand() > 0.8:
                    newGene[j][i] = (
                        newGene[j][i] + random.randint(0, lingyu)
                        if newGene[j][i] + lingyu < self.loaders[i].num
                        else newGene[j][i]
                    )
                elif np.random.rand() < 0.2:
                    newGene[j][i] = (
                        newGene[j][i] - random.randint(0, lingyu)
                        if newGene[j][i] - lingyu > 0
                        else newGene[j][i]
                    )
                # if np.random.rand()>0.8:
                #   newGene[j][i] = random.randint(0,self.loaders[i].num-1)
        return newGene

    def get_fitness(self, pop):
        d = [] # 适应度记录数组
        for i in range(pop.shape[0]):
            gens = pop[i] # 取其中一条基因（编码解，个体）
            f1, f2, f3 = (
                self.get_fun1(gens),
                self.get_fun2(gens),
                self.get_fun3(gens),
            ) # 计算此基因优劣（距离长短）
            d.append((f1, f2, f3))
        return d

    def get_fun1(self, gens):
```

```
        V1_max = 200 # 容积
        f1 = 0 #### ? ? ? ? ?
        for i in range(self.partition_num):
            if i == 0:
                taocan = gens[i]
            elif i == self.partition_num - 1:
                taocan1 = np.array([x.num for x in loaders])
                taocan2 = gens[i]
                taocan = taocan1 - taocan2
            else:
                taocan1 = gens[i]
                taocan2 = gens[i - 1]
                taocan = taocan1 - taocan2
            sum_V1 = 0
            for indx, loader in enumerate(loaders):
                sum_V1 = sum_V1 + loader.water_w * taocan[indx]
            f1 = f1 + V1_max * (int(sum_V1 / V1_max) + 1) - sum_V1
        return -f1

    def get_fun2(self, gens):
        V2_max = 2550 # 容积
        f2 = 0 #### ? ? ? ? ?
        for i in range(self.partition_num):
            if i == 0:
                taocan = gens[i]
            elif i == self.partition_num - 1:
                taocan1 = np.array([x.num for x in loaders])
                taocan2 = gens[i]
                taocan = taocan1 - taocan2
            else:
                taocan1 = gens[i]
                taocan2 = gens[i - 1]
                taocan = taocan1 - taocan2
            sum_V2 = 0
            for indx, loader in enumerate(loaders):
```

```
            sum_V2 = sum_V2 + loader.board_w * taocan[indx]
        f2 = f2 + V2_max * (int(sum_V2 / V2_max) + 1) - sum_V2
    return -f2

def get_fun3(self, gens):
    V3_max = 60000 # 容积
    V3_min = 40000
    C = 500
    f3 = 0 #### ? ? ? ? ?
    for i in range(self.partition_num):
        if i == 0:
            taocan = gens[i]
        elif i == self.partition_num - 1:
            taocan1 = np.array([x.num for x in loaders])
            taocan2 = gens[i]
            taocan = taocan1 - taocan2
        else:
            taocan1 = gens[i]
            taocan2 = gens[i - 1]
            taocan = taocan1 - taocan2
        sum_V3 = 0
        for indx, loader in enumerate(loaders):
            sum_V3 = sum_V3 + loader.D * taocan[indx]
        # print(sum_V3, V3_max-sum_V3)
        f3 = f3 + abs(
            V3_max - sum_V3
        ) # + abs(min(V3_max-sum_V3,0)*C) +
            abs(min(sum_V3-V3_min,0)*C) # 负的罚函数
        # f3 = f3 + V3_max-sum_V3 + abs(min(V3_max-sum_V3,0)*C) # +
            abs(min(sum_V3-V3_min,0)*C) # 负的罚函数
    return -f3

def evolution(self):
    self.pop = self.creat_pop(self.pop_size)
    for num in tqdm(range(self.max_num)):
```

```
            function1_values = [self.get_fun1(x) for x in self.pop]
            function2_values = [self.get_fun2(x) for x in self.pop]
            function3_values = [self.get_fun3(x) for x in self.pop]

            self.F = 0.5 + np.random.rand()

            # 生成两个函数值列表，构成一个种群
            non_dominated_sorted_solution = fast_non_dominated_sort(
                function1_values[:], function2_values[:],
                    function3_values[:]
            )
            # 种群之间进行快速非支配性排序，得到非支配性排序集合
            # print("The best front for Generation number ",num, " is")
            # for valuez in non_dominated_sorted_solution[0]:
            #     print(self.pop[valuez],end=" ")
            # print("\n")
            crowding_distance_values = []
            # 计算非支配集合中每个个体的拥挤度
            for i in range(0, len(non_dominated_sorted_solution)):
                crowding_distance_values.append(
                    crowding_distance(
                        function1_values[:],
                        function2_values[:],
                        function3_values[:],
                        non_dominated_sorted_solution[i][:],
                    )
                )
            # print(crowding_distance_values)
            # 变异
            for i in range(self.pop_size):
                self.pop[i] = self.mutate(self.pop[i])

            solution2 = self.pop
            # 生成了子代
            while len(solution2) != 2 * self.pop_size:
```

```
            a1 = random.randint(0, self.pop_size - 1)
            b1 = random.randint(0, self.pop_size - 1)
            # 选择
            solution2.append(self.cross(self.pop[a1], self.pop[b1]))

        function1_values2 = [self.get_fun1(x) for x in solution2]
        function2_values2 = [self.get_fun2(x) for x in solution2]
        function3_values2 = [self.get_fun3(x) for x in solution2]

        non_dominated_sorted_solution2 = fast_non_dominated_sort(
            function1_values2[:], function2_values2[:],
                function3_values2[:]
        )
        # print('************',non_dominated_sorted_solution2)
        # 将两个目标函数得到的两个种群值value,再进行排序
            得到2*pop_size解
        crowding_distance_values2 = []
        for i in range(0, len(non_dominated_sorted_solution2)):
            crowding_distance_values2.append(
                crowding_distance(
                    function1_values2[:],
                    function2_values2[:],
                    function3_values2[:],
                    non_dominated_sorted_solution2[i][:],
                )
            )
        # 计算子代的个体间的距离值
        new_solution = []
        for i in range(0, len(non_dominated_sorted_solution2)):
            non_dominated_sorted_solution2_1 = [
                index_of(
                    non_dominated_sorted_solution2[i][j],
                        non_dominated_sorted_solution2[i]
                )
                for j in range(0,
```

```
                    len(non_dominated_sorted_solution2[i]))
            ]
            # 排序
            front22 = sort_by_values(
                non_dominated_sorted_solution2_1[:],
                    crowding_distance_values2[i][:]
            )
            front = [
                non_dominated_sorted_solution2[i][front22[j]]
                for j in range(0,
                    len(non_dominated_sorted_solution2[i]))
            ]
            front.reverse()
            for value in front:
                new_solution.append(value)
                if len(new_solution) == self.pop_size:
                    break
            if len(new_solution) == self.pop_size:
                break
        self.pop = [solution2[i] for i in new_solution]

    function1_values = [self.get_fun1(x) for x in self.pop]
    function2_values = [self.get_fun2(x) for x in self.pop]
    function3_values = [self.get_fun3(x) for x in self.pop]
    non_dominated_sorted_solution = fast_non_dominated_sort(
        function1_values[:], function2_values[:], function3_values[:]
    )

    best = []
    for valuez in non_dominated_sorted_solution[0]:
        best.append(self.pop[valuez])
    # 帕解
    function1_values = [-self.get_fun1(x) for x in best]
    function2_values = [-self.get_fun2(x) for x in best]
    function3_values = [-self.get_fun3(x) for x in best]
```

```
562
563         fig = plt.figure()
564         ax1 = plt.axes(projection="3d")
565         ax1.scatter3D(
566             function1_values, function2_values, function3_values,
                cmap="Blues"
567         ) # 绘制散点图
568         plt.show()
569         return function1_values, function2_values, function3_values
570
571
572 partition_num_max = int(
573     sum(dingdan.loc[:num_loader]["坯料厚度"] *
            dingdan.loc[:num_loader]["数目"]) / 40000
574 )
575 partition_num_min = int(
576     sum(dingdan.loc[:num_loader]["坯料厚度"] *
            dingdan.loc[:num_loader]["数目"]) / 60000
577 )
578 partition_num_min, partition_num_max
579
580 nsga = NSGA(loaders, partition_num=12, pop_size=100, max_num=500)
581 x = nsga.encode()
582 nsga.get_fun1(x), nsga.get_fun2(x), nsga.get_fun3(x)
583
584 function1_values, function2_values, function3_values = nsga.evolution()
585
586 # 归一化
587 f1 = [
588     (x - min(function1_values)) / (max(function1_values) -
            min(function1_values))
589     for x in function1_values
590 ]
591 f2 = [
592     (x - min(function2_values)) / (max(function2_values) -
```

```
        min(function2_values))
    for x in function2_values
]
f3 = [
    (x - min(function3_values)) / (max(function3_values) -
        min(function3_values))
    for x in function3_values
]
# 放缩
# f1 = [x/10 for x in function1_values]
# f2 = [x/100 for x in function2_values]
# f3 = [x/1000 for x in function3_values]
# f1 = function1_values
# f2 = function2_values
# f3 = function3_values

import os, random
from pyecharts import options as opts
from pyecharts.charts import Scatter3D

filename = "scatter-3d.html"
data = []
for i in range(len(function1_values)):
    data.append((f1[i], f2[i], f3[i]))
scatter = (
    Scatter3D()
    .add(
        "",
        data,
        xaxis3d_opts=opts.Axis3DOpts(
            name="Fun(1)",
            type_="value",
            # textstyle_opts=opts.TextStyleOpts(color="#fff"),
        ),
        yaxis3d_opts=opts.Axis3DOpts(
```

```
            name="Fun(2)",
            type_="value",
            # textstyle_opts=opts.TextStyleOpts(color="#fff"),
        ),
        zaxis3d_opts=opts.Axis3DOpts(
            name="Fun(3)",
            type_="value",
            # textstyle_opts=opts.TextStyleOpts(color="#fff"),
        ),
        # grid3d_opts=opts.Grid3DOpts(width=100, height=100, depth=100),
    )
    .set_global_opts(
        title_opts=opts.TitleOpts(title="Best Pareto Scatter"),
    )
    .render(filename)
)
os.system(filename)

len(data)

for i in tqdm(range(100)):
    x1 = nsga.encode()
    print(nsga.get_fun3(x1))

function3_values
```