摘要

基于提前惩罚和延误惩罚的单机调度问题（简称 SPSP-LEQT）是经典的组合优化问题，在实时生产理念、机场飞机起降和供应链管理中都有着非常重要的应用。已经证明，在计算复杂性理论中，SMS-LEQT 是 NP 难问题，对 SMSP-LEQT 问题的求解具有重要的理论研究价值和实际应用价值。

SPSP-LEQT 问题的求解目标是使得所有工件线性的提前惩罚和二次的延误惩罚之和最小，文中提出了一种基于多扰动机制的迭代局部搜索算法（ILS-MP）来求解 SMSP-LEQT 问题。为了搜索更广泛的邻域空间，文中设计了两种不同的邻域动作，包括工件的插入和交换动作，并通过增量评估策略和设置动作距离阈值来减少邻域评估时间，提高搜索效率。文中创新性提出了多扰动机制来增强搜索的疏散性，具体地，ILS-MP 多扰动机制结合了三种不同的扰动方式，分别是基于禁忌的扰动、基于构造的扰动以及随机扰动，通过调整三种扰动方式的选择概率来达到平衡扰动幅度的目的，即保留了解的优良结构，也有助于搜索到新的搜索区间。文中用 ILS-MP 算法来求解众多广泛使用的标准测试算例，并和已有的比较前沿的求解算法进行比较，无论是计算结果还是求解时间上，ILS-MP 都表现出了优势。对于规模不大的算例，ILS-MP 可以计算得到绝大多数算例的最优解。另外一方面，文中还分析了 ILS-MP 在不同类型的算例上的求解性能。

关键词：NP 难问题；单机调度问题；迭代局部搜索；多扰动机制；禁忌搜索

Abstract

The single machine scheduling problem with linear earliness and quadratic tardiness (SMSP-LEQT) is one of scheduling problem with significant application in just-in-time production philosophy, aircraft landing and supply chain management. It has been proved that SMSP-LEQT is NP-hard in computational complexity theory. Therefore, solving SMSP-LEQT has both theoretical and practical research values.

We propose an iterated local search based on a multi-type perturbation (ILS-MP) approach for SMSP-LEQT whose object is to minimize the sum of linear earliness and quadratic tardiness penalties. ILS-MP employs two different neighborhood relations that consist of insert move and swap move to explore wider searching space. In order to shorten the searching time, we introduce fast evaluation strategies and distance thresholds to these two neighborhood moves respectively. This paper innovatively proposes a multi-perturbation mechanism to enhance the diversity of search. Specifically, the multi-type perturbation mechanism in ILS-MP probabilistically combines three types of perturbation strategies, namely tabu-based perturbation, construction-based perturbation, and random perturbation. The technique both maintain the fantastic structure of solution and move the procedure to a new search space. Despite its simplicity, experimental results on a wide set of commonly used benchmark instances show that ILS-MP performs favourably in comparison with the current best approaches in the literature. For small-scale cases, ILS-MP can obtain the optimal solution for most cases. On the other hand, the paper also analyzes the performance of ILS-MP on different types of cases.

Key Words: NP-hard problem; single machine scheduling problem; iterated local search; multi-type perturbation mechanism; tabu search

目录

1 绪论

1.1 选题背景及意义

从科技的角度来看，未来二、三十年人类社会将演变成一个智能社会，其深度和广度我们还想象不到[1]。近些年来，得益于大数据技术的发展，人工智能逐步兴起，并开始影响人们的生活，智能家居、无人驾驶汽车、个性化推荐、人脸识别等等，都是人工智能给人类带来的科技福利。在大数据和人工智能时代的大背景下，信息科学、机器学习和运筹学分别扮演者不同的角色。数据的采集、挖掘、存储和管理属于信息科学的范畴；拿到数据后，根据数据做分析得出某些规律，这属于统计学习、机器学习甚至深度学习的范畴；而根据数据总结出的规律，决策人需要做决定的时候，既需要考虑方案的合理性，也需要考虑约束条件的限制，需要考虑成本代价或者最终受益，希望在复杂的规律中找到最优化的决策，这就是运筹学要解决的问题。可以预见的是，随着人们对更深层次更全局性优化决策的需求，运筹学必然会扮演着更重要的角色。

在有限数量可行解的集合中找出最优解的一类问题称为组合优化问题，它是运筹学的一个重要分支。所研究的问题涉及信息技术、经济管理、工业工程、交通运输、通讯网络等诸多领域。组合优化问题求解的是离散型变量的优化问题，其一般形式可以描述为：在满足一定的约束或者限制条件下，求取使得给定的目标函数值最大或者最小的解。

单机调度问题（single machine scheduling problem）是组合优化问题的一种[1]，具有较高的理论研究价值，而且实际应用中有很多相关问题亟待解决，因此吸引了全世界研究人员的广泛关注。单机调度问题主要研究的是如何在一台给定的机器上安排一组加工任务，并希望该调度方案使得绩效度量达到最优。这类问题在实际生产生活中有广泛的应用，如适时生产理念（just-in-time，简称 JIT）和供应链管理[2]。适时生产理念提倡物品按需按量按时生产，无论是提前或者延后生产都不建议，都会对生产效益产生不好的影响，太早工厂库存压力大，太晚影响订

[1] 华为任正非 2016 年在全国科技创新大会上的汇报开篇发言

单进度，因此理想的生产计划是所有的工作都在规定的时刻完成。单机调度问题在供应链管理中也有应用，将供应商比做生产机器，每道作业或者工序就是客户的配送任务，在实际中，强调供应商和客户之间的协调，要求尽量在指定的时间完成客户的配送，太早客户库存容量限制不能接受，太晚影响供应商信誉，也可能会使客户的其它安排滞后，所以安排合理的配送方案使得供应需求按时满足在供应链管理中就显得尤为重要。另外，在 CPU 调度、车间作业调度和机场飞机起降调度等问题中，单机调度问题都有广泛的应用。

根据约束条件或者求解目标的不同，单机调度问题有很多变种。一般来说，衡量单机调度问题效益的主要关注点在于作业加工的提前时间惩罚（Earliness）和延迟惩罚（Tardiness）。顾名思义，提前时间惩罚指的是由于作业实际完工时间早于作业要求的完工时间受到的惩罚，延迟惩罚指的是作业实际完工时间晚于作业要求的完工时间受到的惩罚。在不同目标的问题中，这两种惩罚的权重可能不同，文中主要研究的是线性的提前时间惩罚和二次的延迟惩罚问题。由于作业间不存在前后序关系，因此所有工序任意一种排列组合方式都是问题的解。因此如果需要求解最优方案，最差需要遍历所有可能的解，考虑n件作业的单机调度问题，其搜索空间为n个点的全排列的集合，其计算复杂度达到了 $O(n!)$。

已经证明，单机调度问题属于典型的 NP 难问题[3]。大多数组合优化问题都是 NP 难问题，如单机调度问题、旅行商问题、排课表问题以及图着色问题等等[4-7]。在计算复杂性理论中，这类问题的计算复杂度很高，目前很难在多项式时间内求解。一般来说，求解组合优化问题有精确算法、近似算法和启发式算法。精确算法致力于找到问题的最优解，但由于问题的复杂性和计算机技术的制约，在有限的时间内很难利用精确算法求解，但这并不妨碍有毅力的研究人员致力于寻找精确的求解算法求解某些 NP 难问题。近似算法旨在通过数学推导，有限的时间内求取和最优解相差一定范围内的解，但在实际生活中往往和全局最优解相差很大，但并不能满足实际需求。启发式算法是比较常用的求解组合优化的方法，它来自于科学家们对大自然观察，受启发而创造，也因此而得名。通过总结大自然的运行规律或者面向具体问题的经验、规则来求解问题，通过循环迭代不断更新当前获得的解决方案，以便或者更好的解，并有机会得到全局最优解。

一般来说，启发式算法分为很多种，包括局部搜索、禁忌搜索、遗传算法和种群算法等等，其复杂程度和计算效率都不同，但对于不同的问题，不同的算法表现性能均不同，因此往往需要根据具体问题去设计算法，本文研究的目的也是为了找到求解带线性提前惩罚和二次延迟惩罚的单机调度问题的启发式优化算法。该问题在车间生产调度、供应链管理等领域中有广泛的应用，因此解决这一问题可以带来巨大的经济效益；而且单机调度问题属于 NP 难问题，求解这一问题具有重要的理论研究价值。

1.2 国内外研究历史及现状

单机调度问题是比较经典的组合优化问题[8]，业已吸引了国内外众多学者的研究。生产调度问题的研究历史较为悠久，最早是由 Johnson[9]于 20 世纪 50 年代提出，距今 60 多年的历史。根据实际生产生活中的需要，后来诸多学者提出了几种不同求解目标的单机调度问题，并总结出了科学描述生产调度问题的方法。1967 年 Conway 提出了调度问题的四参数表示法[10]，后来 Graham、Lawler[11]等人（1979）提出了更好的三参数表示法（$\alpha|\beta|\gamma$），其中α用来描述机器的数量、类型等机器环境信息，β用于描述工件的加工约束信息，γ用于描述目标函数。

单机调度问题是已被证明的 NP 难问题[3]，由于其复杂性和多样性，很难找到一种通用的求解算法。一般地，解决单机调度问题的算法主要分为两类：第一种是精确算法，利用分支定界或者动态规划来求解；第二种是启发式算法，通过解的迭代更新，不断改进解。

- 精确算法

一般来说，求解单机调度问题的精确算法主要是分支定界（Branch and Bound）和动态规划（Dynamic Programming）两种。1990 年 Abdul-Razaq 提出了两种动态规划方法和四中分支定界算法来求解考虑带权延误惩罚的单机调度问题[12]。Potts 和 Van Wassenhove[13]提出了求解延误惩罚权值相同的单机调度问题的动态规划算法，亦提出了延误惩罚权值不同时的分支定界算法。Sourd 提出了一种分支定界法来求解具有相同预期完成时间的单机调度问题[14]，并同时考虑了提前时间惩罚和延误惩罚。对于考虑二次延误惩罚的单机调度问题，Valente 基于工

序完成时间的松弛提出了一种求解单机调度问题下界的方法，并采用合适的下界，结合插入探测，提出了求解该问题的分支定界算法[15]。Schaller[16]针对该类问题提出了几种分支定界算法，这些算法通过计算紧凑的下界和测试两种优势条件来组合完成。最近，由 Tanaka 和 Araki[17]提出的一种精确算法找到了由 Cicirello[18]生成的考虑带加工准备时间和加权延误惩罚的单机调度问题的全部 120 算例的最优解；并用相同的算法求解延误惩罚不加权的单机调度问题，由 62/64 个算例[19]得到了最优解。

动态规划或者分支定界等精确算法求解时一般可以获得较好的优度，但往往计算时间较长，对于较大规模的算例，需要几天甚至更久的时间，因此在实际中很难得到应用。

- 启发式算法

启发式算法主要包括局部搜索、禁忌搜索、遗传算法和混合算法等，在很多类型的单机调度问题上都有良好的计算效果。Potts 和 Van Wassenhove[20]提出了多种启发式算法来求解不带准备时间的单机调度问题，并发现通过成对交换工件加工位置是不错的策略。Holsenback 提出了一种求解加权延误惩罚的单机调度问题，处理的问题规模可以达到 50 个工件数量[21]。Feo[22]提出了一种贪婪自适应搜索算法（GRASP）来求解带准备成本和线性延误惩罚的单机调度问题。Valente 和 Gonçalves[23]提出了基于随机关键字的几种遗传算法，这些遗传算法大致相同，只是在生成初始解和利用局部搜索收敛时有差异。Valente and Schaller 针对考虑机器准备时间和不考虑准备时间的单机调度问题，均提出了几种求解的启发式算法[24]。Rubin 和 Ragatz[25]提出了一种遗传算法（GA）用来求解最小化总延迟的单机调度问题。Tan 和 Narasimhan[26]提出一种模拟退火算法（SA）来解决带准备时间的单机调度问题，Armentano 和 Mazzini[27]同样则提出用遗传算法（GA）解决该问题，Franca 等人[28]则提出了一种混合进化算法（Memetic）算法解决考虑准备时间的单机调度问题。

对于既考虑准备时间也考虑延误惩罚权重的单机调度问题，也有众多的学者参与研究。Cicirello 和 Smith[18]生成了 120 个带准备时间带延误惩罚权重的单机调度问题算例，每个算例包括 60 个工件；并且他们还分析了一些随机抽样方法

与模拟退火算法搭配在一起的性能。Lin 和 Ying[29]对比了遗传算法（GA）、禁忌搜索算法（TS）以及模拟退火算法（SA）在求解该类单机调度问题时的表现。Valente[30]提出了几种基于束搜索的启发式算法求解带二次延误惩罚的单机调度问题，这些启发式算法既包括利用优先级和总惩罚值来进行评估的束搜索算法，也有带可变束和过滤宽度的变体。近年来 Tasgetiren 等人[31]提出的离散差分进化算法（Discrete Differential Evolution， DDE）以及 Kirlik 和 Oguz[32]提出的变邻域搜索算法（GVNS）大幅度的改进了该类单机调度问题算例的解。2014 年，Xu[33]提出了基于“块移动”迭代局部搜索算法和混合进化算法来求解，在很多该类单机调度问题的标准算例上得到了当前最好解。

1.3 本文主要工作及结构安排

本文主要研究的是单机调度问题，考虑了线性的提前惩罚和二次的延误惩罚。求解单机调度问题的算法有很多，本文设计了一种基于多种邻域动作的带多种扰动机制的局部搜索算法来求解该问题，为了加快搜索速度，针对不同的邻域动作设计了快速评估策略。另外，本文应用设计的迭代局部搜索算法在大量的标准算例上进行测试，和其它已有的求解方法做对比分析，验证算法的优越性能。

本文第一章节主要介绍论文的选题背景和意义，并分别从精确算法和启发式算法两个方面介绍了国内外对各类单机调度问题的研究历史和现状。

第二章中主要介绍的是单机调度问题和启发式算法的概念。由于单机调度问题有不同类别，本节给出了单机调度问题的基本概率和主要分类方式，并采用形式化的语言描述了考虑线性提前惩罚和二次延误惩罚单机调度问题（SMSP-LEQT）的数学模型。另外介绍了几种常见的启发式算法，并简单描述了本人使用到的局部搜索和禁忌搜索的概念。

第三章主要描述求解 SMSP-LEQT 问题的启发式算法。首先介绍了迭代局部搜索的一般性框架，包括主要流程、邻域动作和评估以及扰动机制的概念，然后结合具体的 SMSP-LEQT，提出了使用多种邻域动作、带多种扰动机制的迭代局部搜索算法，并详细阐述了算法的实现细节。另外，本章节还提到了针对不同邻域动作设计的快速评估策略。

第四章主要是实验结果分析。由于启发式算法中涉及多种参数，参数配置信息需要经过大量的实验测试，本文针对邻域动作选择、距离阈值和扰动类型选择概率，详细介绍实验分析过程和结果。为了验证本文设计的迭代局部搜索算法求解 SMSP-LEQT 问题的性能，本文在标准测试集上利用迭代局部搜索算法进行求解，根据实验求解结果和现已有的算法进行对比，从计算结果和运行时间两个方面衡量算法的优劣性。

第五章首先对本论文研究工作做总结，并阐述了本文设计的求解 SMSP-LEQT 算法的创新点。

2 单机调度问题及启发式算法概述

本文主要研究的是考虑线性提前惩罚和二次延误惩罚的单机调度问题，一般求解该问题的算法分为精确算法和启发式算法。由于精确算法实现较为复杂，而且需要较大的时间代价，不适合应用在实际中，而启发式算法往往能在比较合理的时间内得到很优质的解（但不能证明是全局最优解）。

2.1 单机调度问题

单机调度是生产生活中比较常见的组合优化问题，本章主要给出该问题的详细描述以及按照约束条件和求解目标的问题分类。

2.1.1 问题概述

生产调度问题主要是研究如何在一台或者多台机器上加工生产线上的工件。一般来说，在各个工件都有预期的完工时间，甚至工件之间还有可能会有先后序关系，生产调度就是在满足各个约束限制条件下，制定出生产计划，规定每个工件的加工顺序或者时间，以期望实际生产尽可能在安排的时间完成。

单机调度问题，顾名思义，指的是所有的工件都在一台机器上进行加工，这在某些贵重仪器加工上比较常见。另外，单核 CPU 上作业的调度，机场跑道上飞机起飞或者降落的调度也可以抽象成单机调度问题。求解单机调度问题，就是希望在有限的资源下，通过制定合理的工件加工计划，平衡所有工件的完工时间，最小化违约惩罚，最大化经济效益。

2.1.2 单机调度问题分类

单机调度问题根据约束条件和求解目标的不同有多种不同类型，不同类型的单机调度问题需要设计不同的求解算法，很难找到普遍性的解决单机调度问题的算法。

- 根据机器类型划分

- 抢占式：工件在加工的时候允许被抢占，加工到一部分的时候，可以让机器转去完成其它工件的加工，等另外的工件完成后反过来完成剩下的部分，这在 CPU 调度中较为常见；
- 非抢占式：即加工时不允许抢占，只有当当前工件完成了加工，机器才能开始下一个工件的加工；

- 根据是否考虑准备时间来划分：
  - 考虑准备时间：当某个工件加工完成后，紧接着加工的工件需要等待一段时间后才允许在机器上开始加工；
  - 不考虑准备时间：加工完成一个工件后可以马上进行下一个工件的加工；
- 根据机器是否允许有空闲时间：
  - 允许空闲时间：为了避免提前惩罚，允许机器空闲一段时间之后再开始加工；
  - 不允许空闲时间：机器不允许空闲，工件加工时间时紧密相邻的；
- 根据目标函数划分：
  - 不带权延误惩罚：当每个工件的实际完工时间比工期晚时，有延误惩罚，调度需满足所有工件的延误惩罚之和最小；
  - 考虑带权延误惩罚：每个工件的延误惩罚的权值不相同，调度需考虑使得所有工件的带权惩罚值之和最小；
  - 考虑提前惩罚和延误惩罚：不仅考虑工件的延误惩罚，当工件的加工时间比工期早，也有提前惩罚值，调度需满足提前惩罚和延误惩罚的和最小；
  - 考虑提前惩罚和二次延误惩罚：该种单机调度问题既考虑了提前惩罚也考虑了延误惩罚，而且实际生活中，一般延误惩罚比提前惩罚更严重，因此延误惩罚是二次的，这也是本文所研究的问题。

2.2 SMSP-LEQT 问题模型

根据约束条件和目标函数的不同，单机调度问题可划分成多种类型，本论文

主要研究的是带线性提前惩罚和二次延误惩罚的单机调度问题（single-machine scheduling problem with linear earliness and quadratic tardiness penalties，以下简称 SMSP-LEQT 问题），下面使用自然语言和形式化语言相结合的方式给出该问题的数学模型。

给定包含n个工件的集合$J = \{J_1, J_2, \dots, J_n\}$，每个待加工的工件都有加工所需的时间（用$P_i$表示）以及期望的加工完成时间（用$C_i$表示）。所谓单机调度，即只给定一台加工机器 M，这在价格昂贵的精密仪器加工机床、单核 CPU 或机场等场景比较常见。这n个工件需要分别在机器 M 上加工完成，加工的时候不允许抢占，即对于任意两个不同的工件$J_i,\ J_j (1 \le i \le n\ \ and\ \ i \ne j)$，当工件$j_i$在加工的时候，$J_j$必须等待$J_i$加工完成后才能在机器 M 上加工。这些工件是相互独立的，即任意两个工件之间不存在前后序关系，二者间的加工前后关系是可选的，不存在任何限制。机器 M 在 0 时刻就可以认为准备就绪，工序加工没有空闲时间，即当某一个工件加工完之后，可以认为在上一个工件的加工结束时刻就允许开始下一个工件的加工。从上述描述的分析来看，单机调度问题的一个合法解可以用这n个工件的加工序列来表示，所以单机调度问题一般也被称为排列问题。用$x_{it} \in \{0,1\}(1 \le i \le n, 1 \le t \le n)$表示工件$J_i$是否在第$t$个加工，$x_{it} = 1$表示$J_i$是在机器 M 上加工的第$t$个工件，反之则不是。根据问题描述，存在如下约束：

$$\begin{cases} \sum_{t=1}^{n} x_{it} = 1, 1 \le i \le n \\ \sum_{i=1}^{n} x_{it} = 1, 1 \le t \le n \end{cases} \tag{2.1}$$

公式(2.1)中，第一个约束表示每个工件都会被加工，而且只被加工一次；第二个约束表示每个工件不允许抢占，每个时刻只允许一个工件在机器上被加工。

每个工件加工需要一定的时间，而且只有一台机器 M 可以使用，加工能力有限，因此不一定每个工件都能在指定安排的时刻完成加工。由于 SMSP-LEQT 问题中**不考虑工件的准备时间**，而且机器 M **不允许抢占**，也**不允许有空闲时间**，因此一个合法的调度计划可以直接用一个工件序列来表示，如图 2-1 所示，因此单机调度问题一般也被称为序列问题。

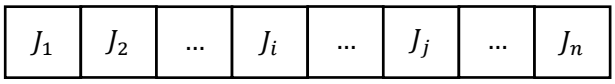


图 2-1 单机调度问题合法解示意图

用D_i表示工件J_i的实际完工时间，存在如下惩罚值定义：

- 提前惩罚(earliness)E_i：如果工件的实际加工时间早于工件的期望完工时间，则会有提前惩罚（如难以保存或者积货影响库存）$E_i = \max\{0, C_i - D_i\}$，用 0 截取表示当加工完成时间晚于期望时间时不会有提前惩罚；
- 延误惩罚(tardiness)T_i：如果工件的实际加工时间晚于工件的期望加工时间，则会有延误惩罚(如影响交货时间或者飞机延误等等造成的损失) $T_i = \max\{0, D_i - C_i\}$,用 0 截取表示当加工完成时间早于期望时间时不会有延误惩罚；

在一般单机调度问题中，提前惩罚和延误惩罚都是一次项，即对于这两种惩罚来说，认为的不良后果是相同的，在带权延误惩罚值的单机调度问题中考虑的仅仅是不同的工件带来的延误惩罚会不同。而在 SMSP-LEQT 问题中，提前惩罚是一次的，延误惩罚是二次的，即认为延误交货时间带来的后果往往更严重，这在如机场调度等场景中十分常见，飞机延误不仅会影响其它飞机调度，还严重影响乘客的满意度。根据上面的描述，SMSP-LEQT 问题的求解目标定义如下：

$$\text{minimize} \sum_{i=1}^{n}(E_i + T_i^2) \tag{2.2}$$

2.3 启发式算法概述

在实际生产生活中，往往会遇到一些大规模的复杂优化问题，如机场航班调度、停机位分配问题、排课表问题、车间作业调度、圆形 packing 问题和负载均问题等等[4, 5, 34-37]，这些问题都是 NP 难问题。目前求解 NP 难问题的主要算法有精确算法、近似算法和启发式算法三种，应用最有效最广泛的当属启发式算法。

启发式算法（Heuristic）指的是人们受大自然的启发，通过总结经验和规则来求解问题的算法。实际中的组合优化问题往往求解难度很大，而且在大多数场景下，要求能快速决策，给出解决方案。一般地，精确算法很难在比较短的时间内找到较好的解，而启发式算法通过对问题的分析以及结合实际中的方案规则，能在合理的时间内得到较好的解，但启发式算法的求解性能缺乏理论支持，往往是通过实际求解结果来验证其性能。

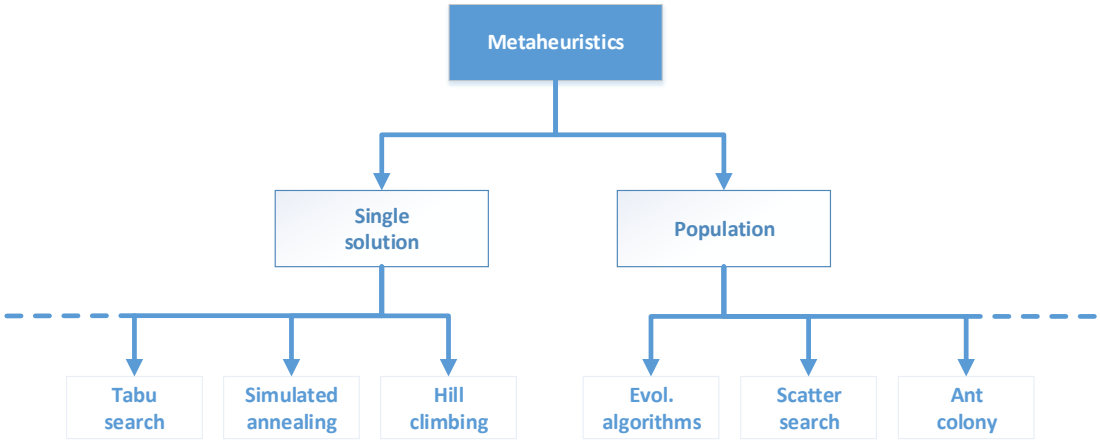


图 2-2 启发式算法族谱

图 2-2 给出了元启发式算法族谱，启发式算法一般分为个体优化算法和种群优化算法。个体优化算法包括局部搜索（又称为爬山法）、禁忌搜索、模拟退火等等，种群优化算法包括遗传算法、蚁群算法、散射搜索算法等等。

图 2-3 给出了启发式算法的设计空间示意图。对启发式算法来说，好的算法设计既要考虑到解的集中性，也要考虑到解的疏散性。所谓集中性，指的是解能够快速迭代更新优化，更快找到或者收敛到局部最优解；所谓疏散性，指的是搜索更广阔的解空间，更有利于增加搜索到全局最优解或者更好解结构的可能性。随机搜索算法中，疏散性的体现最强，搜索的空间会很大，但解的质量变化也很大，不会收敛；局部搜索中集中性的体现最强，随着搜索的进行，解会不断下降优化。一般来说个体优化算法更着重于体现解的集中性，但并非完全不考虑解的疏散性，一般会通过扰动或者禁忌表来增强解的疏散性。种群优化算法也并非完全指考虑解的疏散性，它通过解的交叉来扩展解空间，但依然会通过一些简单快

速的局部搜索算法来使种群中的个体收敛到更优的解。

图 2-3 启发式算法的设计空间

本文求解单机调度问题主要运用了局部搜索和禁忌搜索两种算法，这里简单地介绍下局部搜索和禁忌搜索的概念。

2.3.1 局部搜索

局部搜索，顾名思义，就是通过有效的搜索策略，找到局部最优解。局部搜索的主要过程是从一个初始解出发，通过在解空间中搜索当前解的邻域，找到邻域中更好或者最好解，并用找到的新解替换当前解，重复上述过程直到解不再更新为止。

局部搜索算法设计主要集中在三个方面：邻域结构设计，评估函数设计以及达到局部最优后如何进一步拓展解空间。所谓邻域结构，指的是从当前解出发，通过简单的邻域动作对当前解进行幅度较小的改动，得到邻域解空间。图 2-4 给出了局部搜索示意图，最左表示当前解，右边表示邻域候选解，局部搜索就是不断从当前解出发构造邻域得到候选解集，并从中选择某个解作为新的当前解，重复这一过程就是局部搜索的主要流程。一般来说，邻域解是通过在当前解的基础上，对解的构成做比较小的改动（称为邻域动作），得到的具备新结构的解。当然，做邻域动作需要保证解的合法性不受影响。邻域空间越小，从邻域中找到最优解的速度越快，但搜索的范围小，不一定能覆盖到好的解，很容易陷入局部最优；邻域空间越大，从邻域中找到最优解的速度越慢，但搜索的解数量多，更容易持续更新当前解。因此，好的邻域结构设计也要考虑集中性和疏散性的平衡，既需要保证搜索的效率，也要尽可能搜索到更多的优质解，避免很快陷入局部最优。评估函数指的在确定邻域结构之后，如何从邻域中选择新的解替换当前解，

这依赖于评估函数的选择。评估函数需要从领域解结构中挑选出优质的解，一般来说评估函数直接反应了目标函数的值或者目标函数的变化量，旨在不断得到更优的解。评估函数的确立也要注意搜索的效率，由于局部搜索中往往需要大量重复使用评估函数来评估，高效的评估函数可以极大的提高搜索效率。

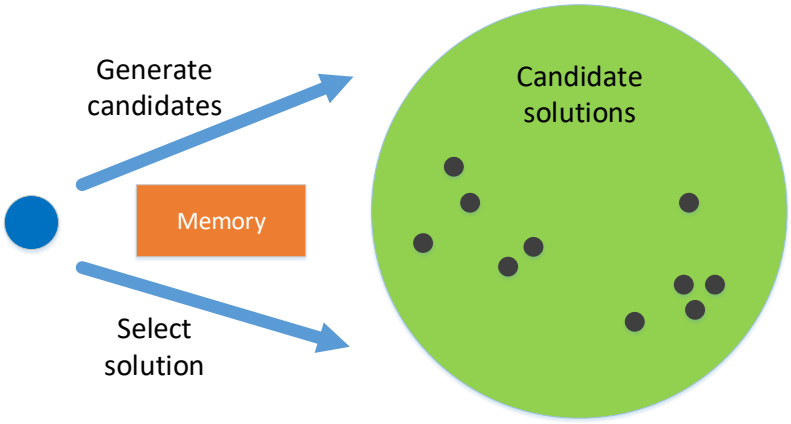


图 2-4 局部搜索示意图

局部搜索一个重要的特征就是算法在执行过程中会逐步陷入局部最优，即当前解就是邻域中的最优解时，算法将无法搜索更好的解，因此需要设计某种机制，如对解进行扰动（较大程度改变解的结构）扩展新的搜索空间。扰动机制的作用是希望跳出局部最优，因此扰动的幅度不能太小，避免又回到局部最优解；扰动的幅度也不宜过大，幅度过大的扰动类似于重新生成初始解，之前搜索得到的解毫无作用，不易于保持优良的解结构，会因为无效率的搜索大幅度降低算法的计算速度和性能，也不利于进一步改进解。

2.3.2 禁忌搜索

禁忌搜索（Tabu Search 或 Taboo Search，简称 TS）的思想最早由 Glover(1986)提出[38]，它是对局部领域搜索的一种扩展。禁忌搜索的主要思想是通过特殊的存储结构记录搜索过的解空间。当然，一般来说很难直接将搜索过的解完全保存下来，通常是通过记录一些邻域动作，来判断当前的邻域动作是否被禁忌。

禁忌搜索的大致流程和局部搜索很相似，也是从一个初始解出发，通过搜寻

邻域中的解来不断更新当前解。但与之不同的是，禁忌搜索通过禁忌表来记录搜索区域避免迂回搜索。在禁忌搜索的过程中，会将搜索过的解或者动作记录在禁忌表中，禁忌表中的元素也并非一直被禁忌，每个元素的禁忌时间称为禁忌长度，当迭代一段时间之后允许禁忌表中的元素被解禁，避免漏掉结构性良好的解。由于禁忌表的存在，禁忌搜索在每次扩展邻域的时候会将邻域解集合划分为受禁忌的候选集和不受禁忌的候选集，确立邻域最优解会从不受禁忌的候选集中的元素中挑选，但也存在一些例外情况，称为特赦准则。所谓特赦准则，指的是有一些邻域解虽然被禁忌，但为了获得更好的解，也必须选择，所以一般认为如果当前解的目标函数值优于找到的最好解的目标函数值，即便该解被禁忌，也允许选择该动作来替换当前解。

从禁忌搜索的流程上来看，禁忌表的设计显得尤为重要。一方面，禁忌表要对搜索过的解空间进行封闭，因此禁忌表既不能漏掉太多搜索过的空间，不然难以避免迂回搜索；另一方面亦不能将未搜索过的解标记为禁忌状态，否则可能会因为错误的标记漏掉结构较好的解。禁忌表不但需要保存禁忌元素，还要判断邻域解是否处于禁忌状态，因此保证禁忌判断的效率也显得尤为关键。由于组合优化问题的结构一般比较复杂，即便是本文研究的单机调度问题，其解也是工件的序列，但如果禁忌表中直接保存单机调度问题的解，那可能搜索的绝大部分时间开销都是在判断当前工件序列是否和解中的一致，显然算法的性能大打折扣，因此通常来说会将一些邻域动作加入到禁忌表，既能通过判断邻域动作是否在禁忌表中来快速判断当前解是否被禁忌，也能有效地避免迂回搜素。

3 求解 SMSP-LEQT 的多邻域迭代局部搜索算法

求解组合优化问题的启发式算法有很多，包括局部搜索、禁忌搜索、遗传算法、蚁群算法[38-41]等等。对于 SMSP-LEQT 问题，本文设计了一种带多种扰动机制的迭代局部搜索算法来求解。局部搜索是比较简单而且较容易实现的算法，但对于组合优化问题，并不是计算的算法越复杂，效果越好，简单的算法，如果能对问题理解地更透彻，往往也有较好的效果，用简单的算法来求解复杂问题，也是算法优势的体现。当然，由于局部搜索的局限性，当单机调度问题的规模较大的时候，很容易陷入局部最优，因此本文设计了一种多扰动机制来增强解的疏散性。下面首先介绍迭代局部搜索算法的具体流程，然后再结合实际研究的 SMSP-LEQT 问题来详细描述针对该问题的求解算法。

3.1 迭代局部搜索算法

迭代局部搜索算法是一种常见的解决组合优化问题的算法，也是比较容易实现的算法，它在解决二次分配问题[42]都有很好的应用。本节主要介绍迭代局部搜索算法的详细过程和在求解 SMSP-LEQT 问题时的具体应用。

3.1.1 基本思想

局部搜索的过程在 2.3.1 中已经阐述，主要是通过邻域动作从当前解出发获得多个邻域解，从中挑选出一个解替换当前解，达到更新解的目的。但如果当前解就是邻域中的最好解时，局部搜索就会陷入局部最优，无法再进行搜索，所以常迭代使用局部搜索算法。因此，很多时候会通过重启局部搜索来扩大搜索范围。但随着问题规模的增大，重启的效果变得越来越差，因为重启是通过重新构造初始解再应用局部搜索。但由于随机构造的初始解和全局最优解往往相去甚远，完全从初始解出发，很快就会收敛，很难具备达到全局最优解的条件。一般迭代局部搜索不通过重启来进行迭代，而是在局部最优解的基础上进行扰动，也就是对局部最优解进行微小的调整，再从扰动后的解出发，重新进行局部搜索。因为扰

动是从局部最优解为基础，所以在一定程度上保留了之前搜索到的优良的解结构，而且通过扰动，可以在不同的邻域解空间进行局部搜索，搜索的疏散性显著增强，有助于找到优度更高的解。

3.1.2 迭代局部搜索流程

算法 1 迭代局部搜索主体框架

Algorithm 1 General framework of local search

1: s ← generate initial solution
2: Repeat:
3: **Repeat**
4: choose best or better solution s^* from N(s)
5: s ← s^*
6: **Until** s^* is not better than s
7: $s \leftarrow Perturbation(s)$
8: Until stop condition is meet

迭代局部搜索算法的主体框架如算法 1 所示。迭代局部搜索算法会首先构造一个合法的初始解（第一行），如何生成初始解要根据具体的问题来决定。初始解的构造约精细，越能很快收敛到局部最优解，而且初始解的结构越好，越容易收敛到更好的解，但一般来说，初始解的构造不需要太复杂，好的局部搜索算法不能太依靠初始解的构造，无论初始解的构造结果如何，通过迭代进行局部搜索均能得到较好的求解方案。通过邻域动作，从当前解出发得到其邻域解集合$\mathrm{N(s)}$，根据设计的评估函数和接受准则，从中选择一个邻域解替换当前解（第四行）。如果得到的新解比当前解要优，则用新的解替换当前解（第 5 行）；如果当前解释邻域中的最优解，则表示陷入了局部最优，局部搜索停止（第 6 行）。以上过程就是局部搜索的主要过程，图 3-1 给出了几步局部搜索的示意图，σ_0表示初始解，通过在邻域$N(\sigma)$中搜索最好解替换当前解，解经历了从σ_0到σ_4的更新优

化，而σ_4的邻域中不存在比σ_4更优的解，即σ_4为局部最优解，局部搜索因陷入局部最优而停止。陷入局部最优之后就需要通过扰动来进行新一轮的迭代搜索（第7行），通过扰动从局部最优解到达新的搜索起点，继续进行局部搜索，迭代局部搜索就是不断完成扰动和局部搜索的过程。

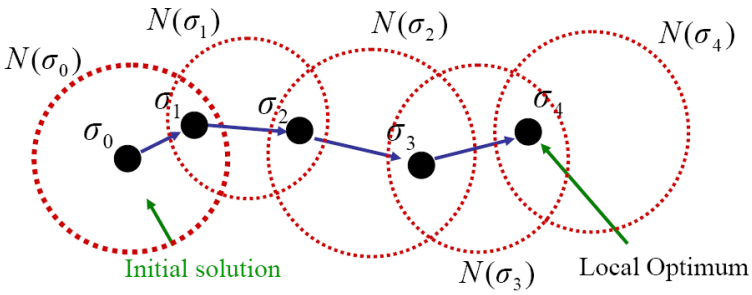


图 3-1 局部搜索过程

迭代局部搜索中局部搜索是迭代反复执行的，需要设定程序的停机条件。一般来说，迭代局部搜索的停机条件可以使静态的也可以是自适应的。静态的指设定固定的求解时间，但这种方式很少使用，因为对于不同规模的算例来说，设置相同的计算时间显然是不太合理的，所以多采用自适应的形式。所谓自适应，指的是根据当前解的更新情况来决定是否继续搜索。当进行几次迭代局部搜索的时候，解一般会呈现比较明显的优化趋势，显然在这个时间点是应该继续搜索的。当迭代一段时间之后，可能解的优化速度放缓，甚至很长的一段时间解都没有更新，这时候就要考虑是否停止搜索了。一般地，会设置某一个迭代代数阈值，当迭代次数超过这个阈值，搜索到的解一直没有更新，算法就会自动停止。

3.1.3 邻域评估和接受准则

设计邻域动作得到邻域动作之后，算法需要从邻域解集合中选取合适的解替换掉当前解，这依赖于评估函数的选择。一般来说，评估邻域动作或者评估邻域解主要有三种方式：

- **评估完整目标函数**

评估完整目标函数是通过直接计算并比较解的目标函数值来决定的。这种方

法一般比较容易实现，但实现时一般要遍历所有的决策变量，如单机调度问题中需要计算所有工件的提前惩罚和延误惩罚，这显然是非常消耗计算资源，效率极低的做法，问题规模越大，该方法的弊端越明显，所以一般情况下不采纳。但某些问题可能约束条件太多，或者邻域动作导致的变化太大，才会采用直接计算的方式。

- **增量评估**

增量评估是一种比较常用的评估方法，指的是不直接计算解的目标函数值，而且通过计算邻域动作引起的目标函数值变化量来达到评估邻域解的目的。一般邻域动作都是对解做比较微小的改动，所以增量评估只需要计算特别少的部分，计算量很小，在同样的时间内可以进行多次迭代，解更容易更新，大大提高了算法求解性能和效率，因此增量评估被应用的场景比较广泛，在设计评估函数的时候也一般会优先考虑能否使用增量评估的方式。

- **近似评估**

在某些问题场景下，尤其是复杂的实际或者工业问题中，涉及的约束较多，因为邻域动作带来的解结构的变化量很大，很难通过增量评估来达到快速评估的目的，但计算完整的目标函数值同样会消耗很大的时间代价，因此可以考虑采用近似评估的方法。即不直接计算目标函数值，而是通过一些策略对目标函数做一些改进，比如有些问题中会通过评估下界（或者上界）来达到评估的目的，通过近似函数替代目标函数，也能继续采用增量评估的策略，进一步提高搜索速度。这种方法在近似函数选择适当的时候会很有效果，既能大大提高搜索速度，也能达到优化解的目的，但往往近似函数的选择比较难以权衡和证明。

通过评估函数可以评估邻域中解的优度，但如何从邻域中选择合适的解来替换当前解也是需要考虑的。一般来说，邻域解的接受准则有两种：

- **最优改进原则（Best ImprovedPrinciple）**

最优解原则就是要求评估邻域中的所有解，并根据评估函数选出评估值最优的解。这种接收准则必须检验邻域内所有解的评估函数值，可以选出最好的解来替代当前解；但当邻域空间增大的时候，评估的代价会很大，每次都需要遍历所有的邻域解。

- **第一次改进原则（First Improved Principle）**

第一次改进原则指的是选择第一个目标函数值或者评估函数值优于当前解的邻域解，即搜索到更优的邻域解就替换当前解。这种方式一般在搜素的前期会迭代的很快，因为当解的优度不是很高的时候，邻域解中的更优解较多，迭代速度较快；但因为不是选择的最优解，有可能会漏掉邻域中更好的解。

这两种邻域解的接收准则各有其优缺点，一般来说，当邻域评估比较快速的时候，按照最优解原则可以快速找到邻域中最好的解，当邻域评估比较困难，一次邻域动作的评估比较复杂时，可以选择第一次改进原则，达到快速迭代的目的。

3.1.4 扰动机制

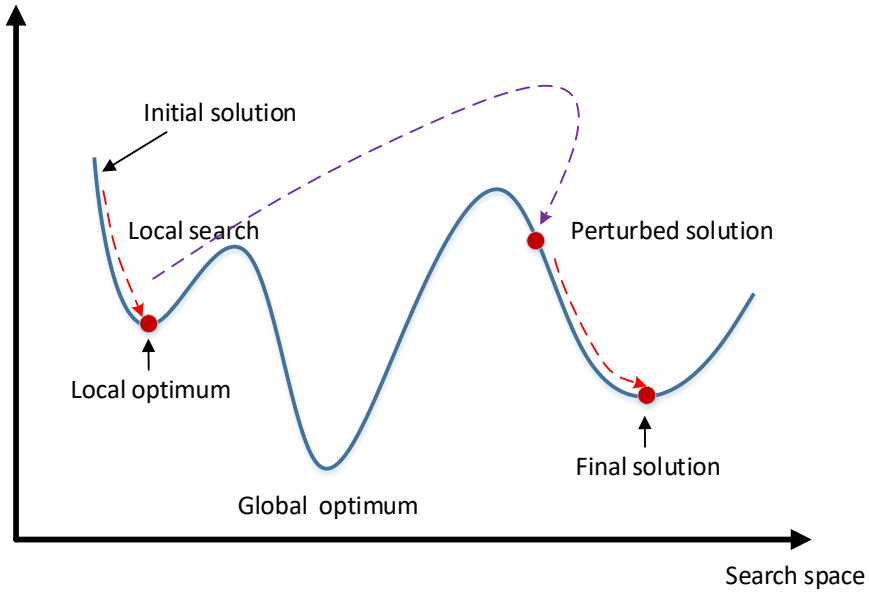


图 3-2 扰动效果示意图

局部搜索一个很大的弊端就是容易陷入局部最优，即当邻域中的最好解就是当前解本身的时候，局部搜索就停止了，因此需要重新设置局部搜索的起点。一般地，采取多次重启的机制，可以很方便地迭代进行局部搜索，但往往这种方式效率很低，之前搜索的局部最优解没有保存其优良的解结构，几次局部搜索很少能得到更优的解。扰动是一种比较简单易实现的增强解的疏散性的方式，扰动指的是在局部搜索陷入局部最优之后，在局部最优解的基础上对解进行一定的改动。

图 3-2 给出了扰动后迭代局部搜索效果搜索示意图，其中纵轴表示解的目标函数值，越小表示解越优。当局部搜索（爬山法）从差解出发，通过搜索邻域最优不断更新解，一定步数之后，当前解即为局部最优解，局部搜索无法继续进行。此时通过扰动可以将局部搜索的起点移动到解空间的其它位置，经过局部搜索之后有可能得到更好的局部最优解，甚至有概率能到达全局最优解所在的解空间。扰动既要保持解的优良结构，也要和当前解有较大区分，这样搜索空间才会扩展。扰动的设计理念来自于“物以类聚”的思想，一般认为目标函数值较优的解应该具有相似的结构，扰动就是利用这种相似结构对解进行拓展，为了在迭代局部搜索过程中搜索到更广泛的解空间。相较于邻域动作，扰动的动作幅度要稍大一些，否则扰动之后只是重复之前的搜索过程，无任何意义；扰动的幅度也不能过大，否则局部最优解丧失了其优良结构，扰动之后的搜索过程和重新构造初始解无异，也就失去了扰动的意义。

在不同的问题中，扰动的方式均不同，这依赖于不同问题的解结构。但一个很简单的思想是，根据求解算法中的邻域动作，进行较多次操作，可以认为是一个比较好的方案。在排列问题中，元素的顺序就是问题的解，可以采取打乱某一部分的顺序来进行扰动，这部分的数量的多少也就决定了扰动的幅度；在图着色问题中，顶点着的颜色是问题的解，对于一个局部最优解，随机改变一定数量顶点的颜色，可以作为一种扰动手段；在顶点覆盖问题中，选择的顶点集是问题的解，可以从中删除一部分结点，并按照约束添加必要的新结点，就可以达到扰动的效果。在这些问题中，这些扰动动作很类似与邻域动作，但一般涉及到多个元素，比邻域动作的幅度大很多。而且因为不是完全的重新构造，排列问题中很大一部分元素的顺序、图着色问题中大部分顶点的颜色和顶点覆盖问题中大部分的顶点集都是和扰动前的局部最优解一致的，因此解的部分结构被保存下来，部分结构发生了变化，既考虑了算法的集中性，也考虑了疏散性。

3.2 求解 SMSP-LEQT 的启发式算法设计

本论文研究的是带线性提前惩罚和二次延误惩罚的单机调度问题（SMSP-LEQT），并采用迭代局部搜索的方法来求解。迭代局部搜索算法是一种比较简单

易实现的启发式算法，在很多组合优化问题上表现了极好的求解性能。为了增强迭代局部搜索算法在 SMSP-LEQT 的表现，文中设计了多种邻域动作，增大搜索空间，并设计了复杂的扰动机制，达到了很好的疏散性，下面就详细描述求解 SMSP-LEQT 的启发式优化算法。

3.2.1 总体求解框架

依据迭代局部搜索的主体框架，本文设计的带多动扰动机制的迭代局部搜索算法（iterated local search based on multi-type，简称 ILS-MP），通过局部搜索找到局部最优解，通过多扰动机制去探寻新的可能的搜索空间，二者交替使用来求解 SMSP-LEQT 问题，求解框架如算法 2 所示。具体地，算法首先会构造初始解（第一行，见 3.2.2 节），并应用梯度下降算法搜索局部最优解（第 5 行，见 3.2.3 节），在梯度下降算法的每一次迭代过程内，均会随机选择一种邻域结构来搜索局部最优解，可选的邻域动作包括工件插入和交换这两种邻域结构，并按照最优改进原则选取邻域中的最优解作为接收准则。如果当前解的邻域中不存在比当前解更好的解，则表示梯度下降算法（亦称为局部搜索算法）陷入局部最优。当出现这种情况时，ILS-MP 算法会启动多扰动机制（第 12~18 行，见 3.2.5 节），扰动机制中包括基于禁忌的扰动动作、基于构造的禁忌动作和随机的扰动动作三种，进行扰动的时候，算法会从这三种扰动动作中选择一种。这三种扰动动作选择的概率是不一样的，会优先选择基于禁忌的扰动动作，然后选择基于构造的扰动动作，在极小的概率下选择完全随机的扰动动作。在执行扰动的时候，不仅会给出当前搜索到的局部最优解，也会根据给出的扰动动作次数 L 来控制扰动的幅度。扰动之后的解即为新的局部搜索算法的起点，并迭代进行下一轮的下降搜索，多次迭代之后找到的最优解会被保存下来，即S_{best}就是迭代局部搜索找到最终结果（第 20 行）。迭代局部搜索会不断进行迭代，以希望搜索更大的搜索空间，但当很长一段迭代时间内（超出迭代步长$stop_iter$）搜索的结果没有改进，就表示算法无法继续持续改进。第 7~8 行表示解更新之后，迭代步长计数器会置零，第 10 行如果解未更新时，迭代步长计数器会自增 1，当未改进的迭代步数大于阈值，索索过程即停止。

算法 2 求解 SMSP-LEQT 的迭代局部搜索算法框架

Algorithm 2 ILS for SMSP-LEQT

1: Generate an initial solution S_0

2: $S_{best} \leftarrow S_0$

3: $iter\_no\_improv \leftarrow 0$

4: **while** $iter\_no\_improv < stop\_iter$ **do**

5: $S \leftarrow LocalSearch(S')$

6: **if** S is better than S_{best} **then**

7: $S_{best} \leftarrow S$

8: $iter\_no\_improv \leftarrow 0$

9: **else**

10: $iter\_no\_improv \leftarrow iter\_no\_improv + 1$

11: **end if**

12: **if** $(rand(0, 0.1, \dots, 1) < P)$ **then** /*P ∈ [0,0.1, … ,1] is a coeff.*/

13: $S' = TabuBasedPerturbation(S, L_1)$

14: **else if** $(rand(0, 0.1, \dots, 1) < (1 - P) \cdot Q)$ /*P ∈ [0,0.1, … ,1] is a coeff.*/

15: $S' = ConstructionBasePerturbation(S, L_2)$

16: **else**

17: $S' = RandomPerturbation(S, L_2)$

18: **end if**

19: **end while**

20: return S_{best}

从以上的算法描述可以看出，ILS-MP 主要过程是构造初始解、进行局部搜索、扰动后继续进行局部搜索。下文将详细介绍 ILS-MP 求解算法的各个部分内容，主要包括如何构造初始解、邻域动作和评估策略以及多扰动机制的详细内容。

3.2.2 构造初始解

一般来说，对于迭代局部搜索算法，初始解的构造不需要太复杂，应该尽可能依靠局部搜索的性能去对解进行改进，这也是评价局部搜索算法优劣的一个方面。初始解的构造也不能过于随机，因为对于大规模的组合优化问题，局部搜索的解空间有限，完全随机的初始解所在的解空间可能和全局最优解相距甚远，最终得到的解的质量可能不高。

对于给定的包含n个相互独立工件集合$S=\{J_1,J_2,\dots,J_n\}$，本文用一种贪心的启发式方法来构造初始解，也就是得到一个包含这n个工件的有序集S'。具体地，S'首先会被置为空集$\emptyset$，并从工件集合S中随机选择一道工件J_i加入到S'中，此时$S=S-\{J_i\}$，随机选择的工序J_i会根据评估函数（见 3.2.4 节）来决定最佳的插入位置，因为S'是有序工件集，工件的顺序决定了问题的解。S'中添加了新的工件元素，即$S'=S'\cup\{J_i\}$。因此，构造初始解时，会在集合S和S'中不断进行工件的添加和删除，直到集合S中的所有工件被完全加入到S'中，初始解的构造算法停止，得到了局部搜索的起始解。

在设计迭代局部搜索算法的初始解时，曾经设计过采用比较复杂的 NEH 启发式算法[43]来构造初始解，但得到的结果和应用上文描述得到的简单的构造算法的得到的最优解并没有相差多少，这也表明本文设计的带多扰动机制的局部搜索算法性能较强，当陷入局部最优之后，会利用多种不同类型的扰动策略来扩展搜索区域，并取得了比较显著的成效，也也印证了本文设计 ILS-MP 算法的优势。

3.2.3 邻域动作和局部搜索

迭代局部搜索算法中比较核心的部分就是邻域结构，局部搜索更新解主要是通过搜索邻域最优解获得。邻域结构的大小既决定了搜索的广度，也决定了搜索

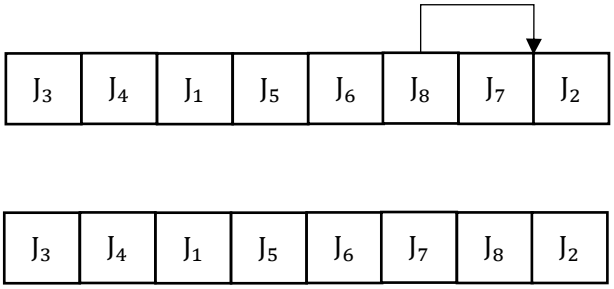


图 3-3 向前插入动作示例

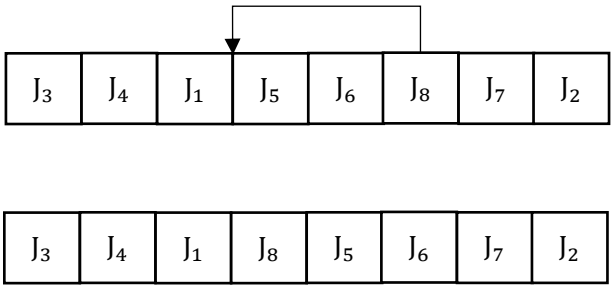


图 3-4 向后插入动作示例

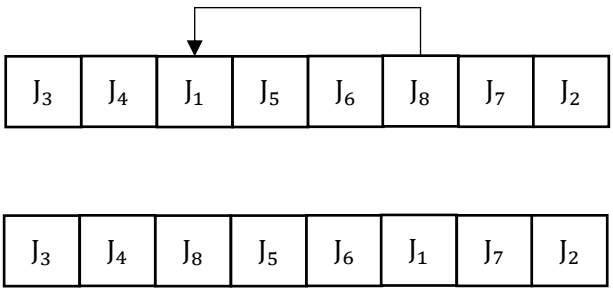


图 3-5 交换动作示例

的深度。进行局部搜索的时候，需要从邻域中找到更优的解，因此邻域空间不能太小，否则难以覆盖到改进解，局部搜索很快陷入局部最优；邻域空间也不能过大，否则邻域进行邻域评估的时候，效率会很低，每次搜索邻域最优解的时间很长，影响搜索的效率。根据 SMSP-LEQT 的问题特点和该问题合法解的构成，本

文设计了两种不同的邻域动作来构造邻域空间进行搜索，即插入动作和交换动作，局部搜索过程中每次迭代都会从二者间选择一种方式作为邻域动作。

具体地，**插入动作**指的是将工件序列中的某道工件插入到其它工件之前或之后。对于给定的n道工件序列$S=\{J_1,J_2,\dots,J_n\}$，将$J_i(1\le i\le n)$插入到工序插入到工序J_k之前或之后，即对于每一道工件，均有$n-1$个可能的插入位置（除去J_i本身的位置），故其邻域空间内大约有$n(n-1)$个可能的邻域解，当然，其中可能包含少量重复的位置。方式作为邻域动作。

图 3-3 和图 3-4 给出了插入动作的两个示例，分别演示了插入到工件之前和之后两种邻域动作方式，图 3-3 表示将工序J_8插入到工件J_5之前，图 3-4 表示将工序J_8插入到J_7之后。**交换动作**指的是将工件序列中某两个工件交换顺序，图 3-5 给出了工件J_8和工件J_1交换的邻域动作，任意两个工件之间的交换均是允许的，因此一共有$n(n-1)/2$种可能的交换动作。

显然，对于n个工件规模的 SMSP-LEQT 问题，插入邻域动作和交换邻域动作的邻域空间大小均为$\mathrm{O(n^2)}$，所以邻域评估的代价较大。为了解决邻域搜索的效率问题，本文针对插入动作和交换动作的距离做了限制处理。对于插入动作，其插入距离$d^i_{insert}=|p_i-p'_i|$，即对于工件J_i的插入动作的插入距离用工件J_i的原位置p_i和新位置p'_i差的绝对值来表示，也就是插入位置和原位置之间的工件数量。对于交换动作，其交换距离$d^{i,j}_{swap}=|p_i-p_j|$，即对于工件J_i和工件J_j的交换动作，其交换距离就是两个交换工件位置差的绝对值，也就是工件J_i和工件J_j之间间隔的工件数量。为了减少评估的时间，我们对这两种动作的距离都做了限制，任意一个工件的插入动作的距离不允许超过$threshold(insert)$，任意两个工件间的交换动作的距离不允许超过$threshold(swap)$，距离超过限制阈值的邻域动作均是不参与邻域评估的。考虑移动的距离，是为了减少邻域评估的时间，尽可能短的时间内找到邻域中的最优解，而移动的距离太远，邻域动作导致的解的变化太大，不适合作为邻域解考虑。而事实也是，一般算法执行中得到的比较优质的解，工件的加工结束时间不会和其工期相差过远，以插入动作为例，如果移动的距离过长，距离工期极远，该工件的惩罚值会特别大，严重影响解的质量。当解本身特别差，工件的结束时间普遍和工期相差较远，那么长距离的移动也可以通过几次

短距离插入来玩成，同样可以通过邻域动作得到更好的解。通过限制邻域动作移动距离的方式，完全可以在不影响解的质量的情况下大幅度节省计算时间。

利用工件的插入和交换动作实现的邻域结构，ILS-MP 算法通过简单最优化接受原则实现了简单的局部搜索。在每次局部搜索迭代的时候，会按照一定的概率λ从插入动作和交换动作中来随机选择某一种邻域结构，并在众多邻域解中选择目标函数值最优的邻域解来替换当前解，即按照最优性原则作为邻域替代解的接受准则。局部搜索会不断迭代进行上述过程，直到获得局部最优解不再更新为止，这也代表当前邻域中并不存在比当前解更好的解，局部搜索无法继续执行。

3.2.4 邻域快速评估策略

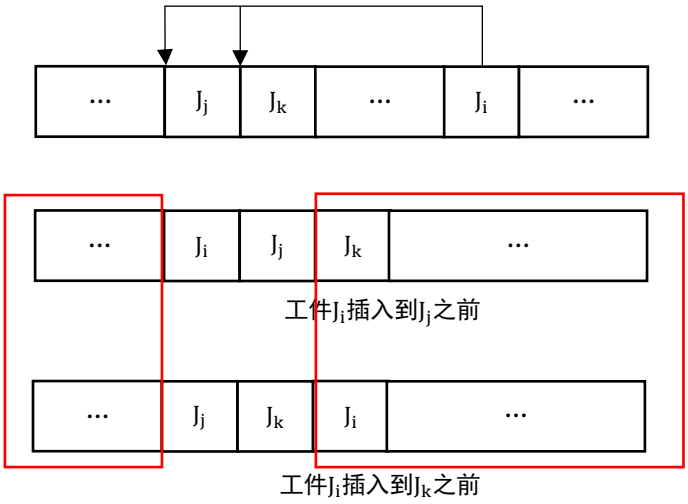


图 3-6 相邻插入动作（朝前）对比

3.2.3 节中提到了两种邻域动作——工件的插入和交换，无论是哪种邻域动作，其可能的邻域大小均是$O(n^2)$，因此必须使用快速评估策略，否则邻域评估的代价太大，邻域搜索时间过长，迭代的次数有限，很难在较短的时间内得到好的解。在 3.2.3 节已经通过设置插入动作和交换的动作的距离阈值来限制邻域结构的大小，在本节中设计了一种针对插入动作和交换动作的邻域快速评估策略，

进一步提高邻域搜索的效率。

一般的评估策略主要有三种：全量评估、增量评估和近似评估。全量评估指的是直接计算解的目标函数值，在 SMSP-LEQT 问题中需要对每个邻域解计算n道工件的提前惩罚和延误惩罚，这样计算的代价显然是巨大的，不适合应用在本问题中。而近似评估由于评估值是通过计算目标函数值近似值的方式，SMSP-LEQT 问题的解结构比较简单，不需要采用近似评估的方式，而且采用近似评估很容易造成评估值不准确，影响搜寻的邻域解的质量。针对插入动作和邻域动作导致的解结构变化，本文设计了增量评估的方式来进行快速搜索。

如图 3-6 所示的例子，在当前解的基础上，对比将工件J_i插入到工件J_j和J_k之前的两种邻域动作，其中J_j和J_k在当前解的序列中是相邻的，并且J_k在J_j之后，J_j在J_k之后，在向前插入的邻域结构中，这两种邻域动作在评估的时候是紧接着被评估的。对比这两个邻域动作形成的新解，红色框中的工件其完成时间是一致的。对于J_j之前的工件，邻域动作不会影响其加工，完成时间不变，因此这两个解中J_j及其之前的工件的惩罚值是相同的；对于J_i之后的工件，因为J_i和J_j之间所有的工件完成之后的总时间没有变化，所以做完邻域动作后，J_i之后的工件的完成时间不变，两个解中这部分的惩罚值也是相同的；而对于J_k和J_i之间的工件（包括J_k），虽然邻域动作会导致这部分工件的惩罚值有变化，但比较这两个新解，因为J_k均是在J_i和J_j完成之后开始的，因此这两个新解中J_k和J_i之间的工件的完工时间是一样的，惩罚值也相同。如果要计算J_i插入到J_k之前形成的解的目标函数值(用f'表示)，只需要通过J_i插入到J_j之前形成的解的目标函数值（用f表示）并结合二解的差值得到，而从前面的分析可知，要计算差值，仅仅需要计算J_i和J_j这两个工件的目标函数值变化，即

$$f' = f + (P_i' - P_i) + (P_j' - P_j)$$

其中P_i和P_j分别表示工件J_i插入工件J_j之前对应的解中工件J_i和J_j的惩罚值，P_i'和P_j'表示J_i插入工件J_k之前对应解中两个工件的惩罚值。计算该值是非常省时的，评估一个邻域解的时间复杂度仅为$O(1)$，且这种增量评估的方式非常适合迭代，因此可以很高效地计算工件J_i插入到所有可能位置的目标函数值。同样地，往后插入的邻域动作也可以采用这种方式增强评估，这里不再详细举例了。

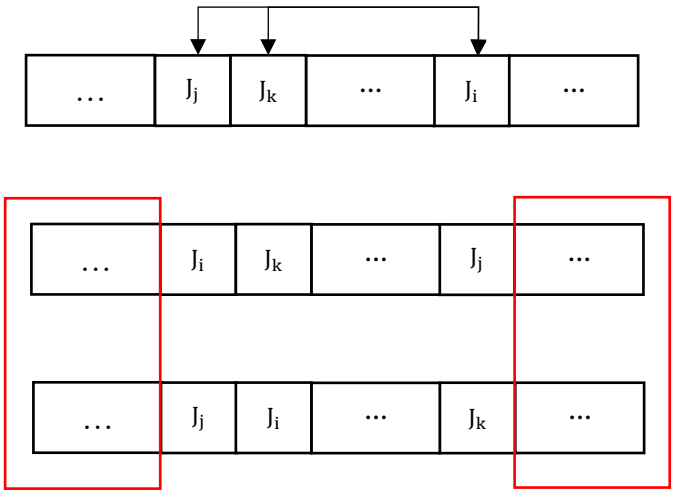


图 3-7 相邻交换动作对比

图 3-7 给出了相邻交换动作对比，同样采取增量评估的策略，但相较于插入动作，需要重新计算的部分更多。图 3-7 中上一个解是将工件J_i和J_j交换位置，下一个解中是将工件J_i和J_k交换位置，J_j和J_k在原解中是相邻的。对于原解中J_j之前的工件，交换两工件的顺序完全不会影响，所以这部分工件的惩罚值不会发生变化，不需要重新计算；对于J_i之后的工件，由于前面所有工件的完成时间不会发生变化，因此这部分工件的惩罚值也不会变化，也不需要重新计算；但J_i和J_j之间的工件（包括二者）在新的两个解中的工件的完工时间明显不同，都需要重新计算。但这里并不需要计算着两个新解中这部分工件的惩罚值，可以直接通过计算新解和原解的目标函数值变化，如图 3-8 所示是交换动作解的变化，目标函数值变化更清晰明了，有

$$f' = f + \sum_{l=j}^{i} (P_l' - P_l)$$

其中f'表示执行交换动作之后新解的目标函数值，P_l和P_l'分别表示交换动作前后J_i和J_j之间各个工件的惩罚值。可以看出，相较于插入动作，交换动作需要重新计算的工件数量和交换工件的距离有关，时间复杂度为$O(n)$，当工件数量规模增大的时候，交换较远距离的工件，需要重新计算的工件数量会很多，会消耗较长

的时间，因此限制交换距离在交换动作邻域中显得更加重要。

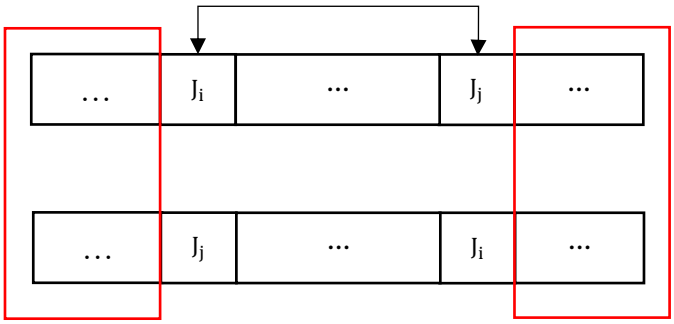


图 3-8 交换动作解的变化

3.2.5 多扰动机制

在 3.2.1 节中提到，区别于传统的迭代局部搜索算法，本文提出了一种带多扰动机制的迭代局部搜索算法（ILS-MP），采用了三种不同的扰动策略，分别是基于禁忌的扰动、基于构造的扰动以及完全随机性的扰动。

基于禁忌的扰动指的是通过迭代固定代数的禁忌搜索来进行扰动。禁忌搜索和局部搜索的过程是类似的，同样是搜索邻域找到邻域最优解替换当前解，但区别在于禁忌搜索会通过禁忌表来记录已经执行过的邻域动作，由于进行扰动的时候，ILS-MP 已经陷入了局部最优，禁忌搜索并不会找到更优的局部解，所以禁忌搜索会用当前非禁忌的最好邻域解来替代，即便此解比当前解要差。由于禁忌表的存在，搜索过程会增加一定的疏散性，也正因如此，以这种方式来对解进行扰动，可以起到积极的效果。禁忌搜索可以接受非禁忌的差解，可以在陷入局部最优之后“绕过山坡”，到达新的搜索路径。基于禁忌的扰动可以根据禁忌表的筛选规则，尽量选择不被禁忌的解来迭代，达到扩展邻域搜索空间的目的。同样地，和之前描述的局部搜索过程类似，基于禁忌的扰动中使用的局部动作也是采用的插入动作和交换动作，在每次迭代过程都会按概率选择某一种邻域动作，故而在进行禁忌搜索扰动的时候，需要两张表来作为禁忌表，分别记录插入和交换的邻域动作。具体地，在每一次迭代的时候，如果某个工件从当前位置i移动到新位置j（可能是插入动作也可能是交换动作），则该工件又回到位置i的动作显然是

不允许的，这会使解回到以前搜索过的结构，浪费搜索时间。从图 3-6 中可以看到，插入动作会导致J_i和J_j中间一系列工件的位置发生变化，但因为中间的位置变化是“被迫”产生的，因此只会讲“主动”工件J_i的位置变化加入到禁忌表中。而对于图 3-8 中的交换邻域动作，仅仅J_i和J_j这两个工件的位置发生了变化，这两个工件的原位置就会被加入禁忌表，短时间内不允许回到原位置。为了使 ILS-MP 更加灵活，本文针对插入动作和交换动作设计了两种不同的禁忌长度，这是因为考虑到禁忌长度应该和不同动作的距离限制有关，因为该值直接决定了邻域空间的大小，禁忌长度是和邻域空间大小强相关的。具体设置的禁忌长度如下：

$$\gamma_{insert} = \alpha_1 \cdot threshold(insert) + \alpha_2 \cdot random(threshold(insert))$$

$$\gamma_{swap} = \alpha_1 \cdot threshold(swap) + \alpha_2 \cdot random(threshold(swap))$$

无论是对于插入动作还是交换动作，禁忌长度都由两部分组成，这也是一般构建禁忌表的做法。一部分是局部搜索中各个动作的距离限制，对于某一个算例来说，它是固定值；另一部分是随机值，大小不超过限制距离的大小，这两部分是线性相关的，通过α_1和α_2两个线性系数构建了每一个禁忌表中每一项不同的禁忌长度。当然，基于禁忌的扰动中还存在一种藐视准则，即如果邻域解虽然被禁忌，但其目标函数值优于当前找到的最优解，该解会被解禁，毕竟局部搜索的目的就在于找到更优质的解。

基于构造的扰动通过几次移动位置的操作构造出新的扰动解。具体地，首先从当前解J_c对应的工件序列中随机抽取一个工件J_r从原先的序列中删除，为了构造出新的合法解，需要给删除的工件J_r找到新的位置插入。相较于局部搜索中插入到目标函数值最佳的位置，构造扰动采取的是随机自适应的方式。在选择插入位置时，会根据插入到各个位置的目标函数值对插入的可能位置排序，如果插入之后形成的新解的惩罚值越小，则选择该位置插入的概率越大；反之惩罚值越高，选择该位置插入的概率越小。这种构造方式，即考虑了贪心选择目标函数值最优的解，也因为随机性的存在，允许接受差的位置插入，提高了疏散性。为了达到贪心自适应扰动的目的，对于每个可能的插入位置设计了概率选择函数。对于插入后某目标函数值排名第 k 的位置，其概率选择函数为

$$VF(j) = \left(\frac{1}{k}\right)\sum_{k=1}\frac{1}{k}$$

其中 j 表示插入的位置序号，k 表示插入位置后目标函数值的排名，显然 k 越大，选择的概率越低。依据这种构造方式，不断从当前解中抽取工件插入到新的位置，构造出扰动解。

第三种扰动是随机的扰动，旨在进一步增加解的疏散性。随机扰动即通过随机选择某工件执行插入动作或者交换动作，插入或者交换的位置也是完全随机的，因此这种扰动方式随机性最强，但也有可能会破坏掉局部最优解的结构。

三种扰动方式都能探寻到新的解空间，在很大程度上提高搜索空间的广泛性，但相较而言，基于禁忌的扰动提升的疏散性是最小的，但能最大程度保留解的优良结构。虽然由于禁忌表的存在可以接受差解到达新的搜索区域，但和原解的距离不会太远，因为禁忌搜索接受的仍然是最优的非禁忌解。随机性的扰动可以最大程度扩展新的搜索区域，但很容易丧失解的结构，类似于重启搜索，意义不大，基于构造的扰动则相对比较平衡。在 3.2.1 节中已经提到了，ILS-MP 会从这三种扰动方式中随机选择一种，首先会以概率 P 选择疏散性最差的基于禁忌的扰动，另外以$(1-P)\cdot Q$的概率选择基于构造的扰动，最后以$(1-P)\cdot(1-Q)$的概率选择随机的扰动。因为每种扰动都需要执行一定的迭代次数 L，为了增强算法的灵活性，选择了不同方式来对当前局部最优解J_c进行扰动时的迭代步长均不同，基于禁忌的扰动、基于构造的扰动和随机性扰动对应的扰动步长分别为L_1、L_2和L_3，具体参数的设置将在后文详细说明。

4 实验结果比较和分析

在前面的章节中，针对 SMSP-LEQT 问题，详细描述了求解该问题的带多种扰动机制的迭代局部搜索算法。因为算法中涉及多个参数，在本章节中，将通过标准算例测试得到最佳的参数配置，并和其它研究人员的计算结果作对比，从惩罚值和时间效率两个方面比较本文提出的 ILS-MP 算法的计算性能。

4.1 实验方案设计

为了更好地衡量算法的性能，需要在标准算例上对算法进行验证，本节将主要描述测试算例的规模和类型，以及实验平台的各类配置参数。

4.1.1 测试算例

SMSP-LEQT 测试算例来自于 Valente[44]等人的构造，问题规模包括 10、15、20、25、30、40、50、70 和 100 个工件这几类。算例是采用随机的方式生成，每个工件的加工时长p_i有两种不同的范围，第一种是从 45 和 55 中随机选择一个整数作为工件的加工时长，第二种是从 1 到 100 中随机选择，前者的变化性低，称为 L 类，后者的变化性高，称为 H 类。对于每道工件的工期，同样是服从均匀分布的，对于工件J_i，其工期d_i是$P(1-T-R/2)$到$P(1-T+R/2)$的平均值，其中P表示所有工件的加工时间的总和，T 指的是延误系数，R 指的是工期的分布范围。$P(1-T)$表示所有工期的期望，因为所有工件的加工时间总和 P 是固定值，显然 T 越大，工期的期望越小，延误时间就越大。R 的大小决定了工期的分布范围，R 越小，工件的工期相对越集中。对于不同规模的算例，T 有 0.0、0.2、0.4、0.6、0.8 和 1.0 共 6 种取值，分布范围 R 共有 0.2、0.4、0.6 和 0.8 共 4 种不同的取值。对于每一种工件数量规模、每一类随机工件加工时长、每一种工期延误系数 T 和分布范围 R 均随机生成 50 个算例，因此对于每一种工件数量规模和一类加工时长（L 类或者 H 类）的算例共有$50 \times 6 \times 4 = 1200$个算例，总计$9 \times 1200 = 10800$个算例。这些算例可以在网站 http://fep.up.pt/docentes/jvalente/bench-

marks.html 找到，同时提到一些国内外学者求解 SMSP-LEQT 问题的最好研究成果，包括 MA_IN，RBS 和 EQTP 三种前沿算法以及某些小规模算例的最优解。

4.1.2 实验环境

本文算法的实现均是在个人 PC 电脑上完成，具体的硬件配置和软件环境如表 4-1 所示。至于 4.1.1 节中提到的 EQTP、RBS、GA、GA_IN、MA 和 MA_IN 算法均是运行在 Pentium IV-2.8 GHz 的个人 PC 上，因此相较而言，本文所使用的机器配置更差。

表 4-1 硬件配置和软件环境

类型	名称	配置或参数
硬件环境	处理器	Intel(R) Celeron(R) G530-2.40 GHz
	内存	2GB
	硬盘	500GB
软件环境	操作系统	Microsoft Windows 7（64 位）
	编译器	mingw32-gcc
	IDE	CodeBlocks

4.2 参数设定

3.2 节描述了求解 SMSP-LEQT 问题的带多种扰动机制的迭代局部搜索算法，由于很难找到普遍性的启发式算法，因此在实际问题中，启发式算法会涉及到多种参数。为了得到最佳的参数设置，需要对不同的参数做了对比测试。本节主要描述迭代局部搜索算法在求解 SMSP-LEQT 问题时的参数设定，以及某些重要参数的比较分析过程。在比较的时候，不光会比较结果的目标函数值，求解时间也是重要的衡量标准，因为对于实际问题来说，计算结果和计算效率都是实际生活中需要考虑的重要

因素。

4.2.1 所有参数设置

表 4-2 参数设置

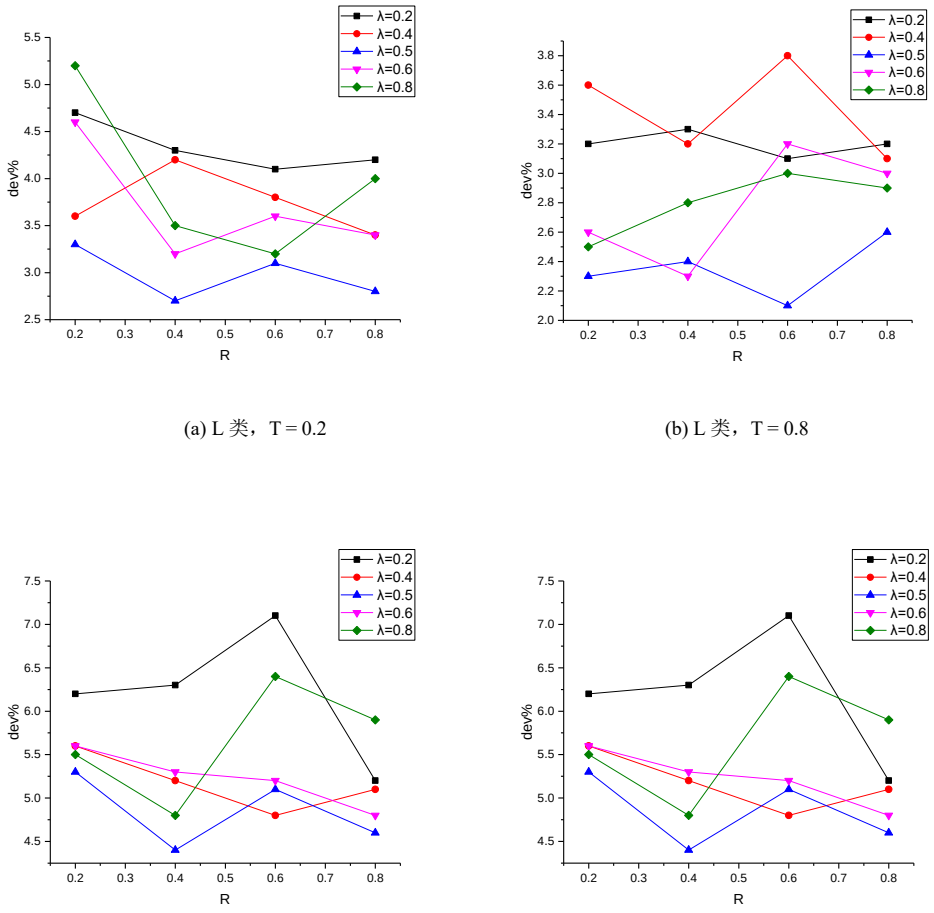
名称	说明	值
$stop\_iter$	算法终止的最大迭代次数，也就是停机条件	50
$threshold(insert)$	插入动作的距离阈值	$n/2$
$threshold(swap)$	交换动作的距离阈值	$n/3$
λ	两种邻域动作中选择插入动作的概率	0.5
L_1	基于禁忌的扰动的移动次数	0.25
L_2	基于构造的扰动的移动次数	0.5
L_3	随机扰动的移动次数	$n/10$
α_1	禁忌扰动中禁忌长度固定值部分系数	0.5
α_2	禁忌扰动中禁忌长度随机值部分系数	0.9
P	选择禁忌扰动的概率	0.7
Q	选择构造扰动而不是随机扰动的概率	0.5

表 4-2 给出了所有参数具体的值设置，主要有六个方面：最大迭代次数、邻域动作距离阈值、邻域动作选择概率、扰动的移动次数、禁忌长度系数和扰动类型选择的概率。各个方面可能有多个参数，具体的意义和值可以参考表中的说明。在对这几类参数设置进行参数调试的时候，因为参数数量众多，用参数集的方式来分析效率太低，因此根据参数重要性逐步对参数进行测试。在测试中发现，邻域动作选择概率、邻域动作阈值和扰动选择概率对解的影响最大，下面给出这三类参数的对比分析过程。

4.2.2 邻域动作选择概率测试

在 3.2.3 节中提到本文求解 SMSP-LEQT 问题用到的迭代局部邻域搜索算法

中设计了两种邻域动作，包括插入动作和交换动作，在每一次迭代的时候，会随机选择一种邻域动作来进行邻域搜索，邻域动作的选择是随机的，因此需要对邻域动作的选择概率做测试。



(a) L 类，T = 0.2　　(b) L 类，T = 0.8

(c) H 类，T = 0.2　　(d) H 类，T = 0.8

图 4-1 邻域动作选择概率测试

在对邻域动作选择概率做测试时，本文选取的是规模为 20 个工件数量，L 和 H 两种类型、T 为 0.2 或 0.8，R 为 0.2、0.4、0.6 和 0.8 的算例集，因为该类工序集存在最优解，可以根据计算结果和最优解的差距来评估参数。测定邻域动作选择概率时，更专注于解的质量，因此邻域动作阈值不做限制，即会评估邻域结构中的所有解。因为扰动参数还没有测定，所以在局部搜索陷入局部最优后通过简单的多重启动来重新搜索，旨在屏蔽其它因素的影响，专注于测定邻域动作

选择概率对解的影响。另外，为了消除偶然性的影响，每种类型（T 和 R 值一定）的 50 个算例都会计算 10 次，并选取最好的计算结果，利用 50 个算例的平均值来和最优解作比较。具体地，会计算不同参数值求解结果和最优解的偏移比 $dev = (H - O)/O \times 100$，其中 H 表示是通过迭代局部搜索获得的结果，而 O 表示是已经被证明的算例的最优目标函数值。图 4-1 给出了λ为 0.2、0.4、0.5、0.6 和 0.8 这五种取值时的测试结果，可以看出，虽然在某些算例上$\lambda = 0.5$计算结果稍逊于其它值，但总体上对于绝大多数类型的算例来说，$\lambda = 0.5$的表现性能要更优，这表明插入动作和交换动作具有同等的选择概率时效果最好，可能是因为这种方式的邻域结构疏散性更强，更容易找到更优的解。

4.2.3 邻域动作距离阈值测试

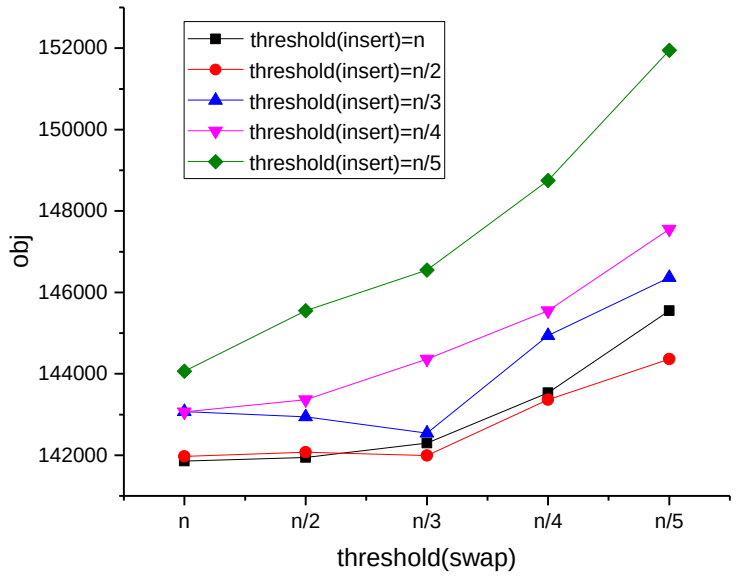


图 4-2 邻域动作阈值对计算结果的影响（L 类，n=100，T=0.2， R=0.4）

在 3.2.3 节中提到两种邻域动作形成的邻域结构大小均为$O(n^2)$，即便是按照加速评估策略，评估一个插入动作的时间复杂度是$O(1)$，评估一个交换动作的时间复杂度是$O(n)$，因此邻域动作评估的时间复杂度不低，尤其是当工件数量增多的时候，局部搜索迭代的次数就非常有限了。设置距离动作阈值，限制插入动作

和交换动作移动的距离，可以在不影响计算结果的情况下，节省计算时间。为了测试最佳的邻域动作距离阈值，本文选取的是工件数量为 100、T=0.2、R=0.4 的 L 型算例进行测试，threshold(insert)和threshold(swap)的取值均可以取到集合$\{n, n/2, n/3, n/4, n/5\}$中的任意一个。同样地，为了消除偶然性影响，每种类型的每个算例会计算 10 次，最后选取最好的实验结果。图 4-2 给出了邻域动作阈值对解的影响，其中横坐标表示threshold(swap)的取值，纵坐标表示 50 个算例目标函数值的平均值，不同的线条表示threshold(insert)的不同取值。可以看出，当threshold(insert)和threshold(swap)值越大的时候，目标函数值越小，这是因为距离阈值越大，邻域空间越大，搜索的结果自然越好。可以发现，threshold(swap) $\geq n/3$，threshold(insert) $\geq n/3$时，解的平均目标函数值已经较低了。

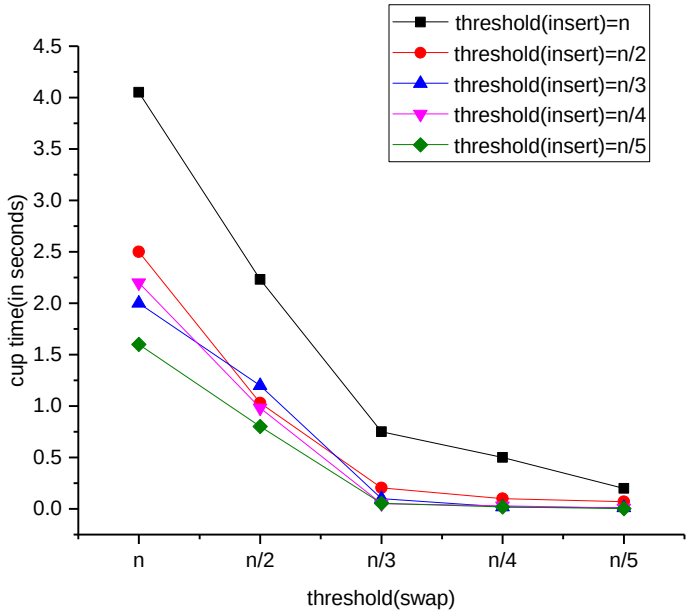


图 4-3 邻域动作距离阈值对计算时间的影响（L 类，n=100，T=0.2， R=0.4）

考虑邻域动作阈值的时候，不仅要考虑解的优度，还要考虑算法的求解时间。图 4-3 给出了邻域动作阈值对计算时间的影响，使用的算例和测试结果优度的算例一样。从图中可以看出，距离阈值对计算时间的影响非常明显，当两种邻域动作的阈值在增大的时候，计算时间也在增加，尤其是对于交换邻域动作，由于

评估的代价更大，其距离阈值大小对计算时间的影响更加明显。由于我们选取的是 20 个工件数量的算例，如果算例规模增大，距离阈值的影响应该会更大。结合计算时间和结果优度的双重考虑，threshold(swap)的值设置为$n/3$，threshold($insert$)的值设置为$n/2$的值最为合适。从邻域动作的评估策略来看，交换动作的评估时间是线性时间的，而插入动作的评估时间是常量级的，因此threshold(swap)的值略小，可以提高邻域评估的效率。

4.2.4 扰动类型选择概率测试

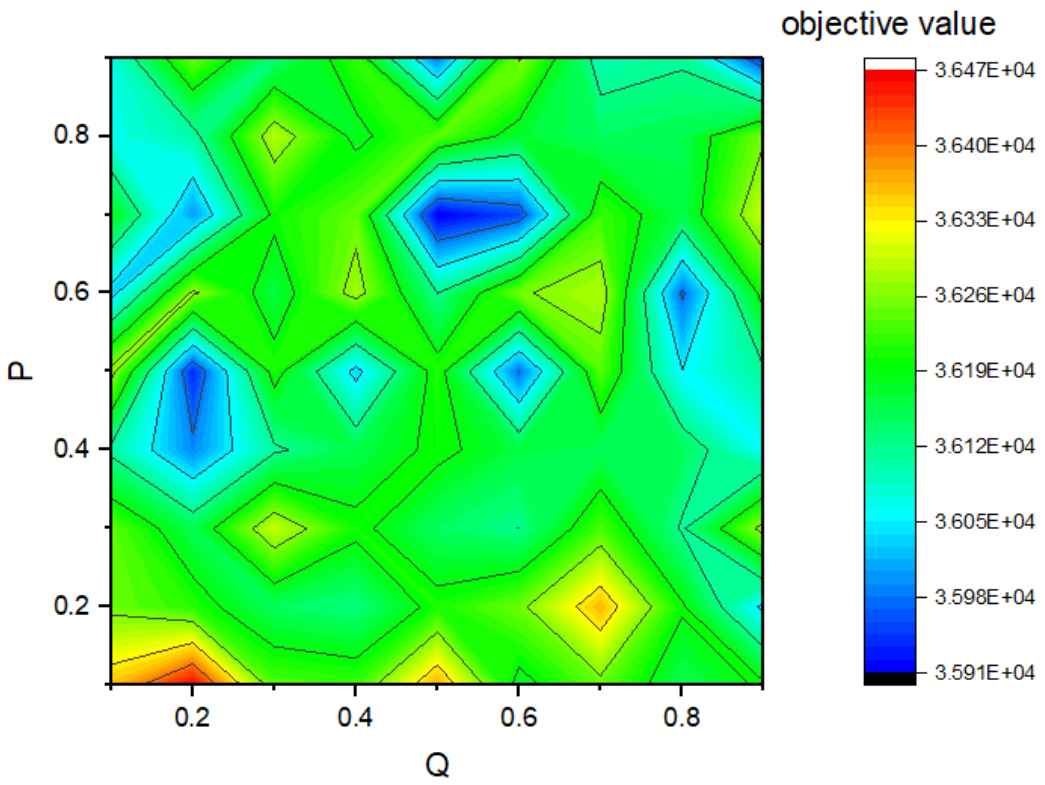


图 4-4 扰动概率选择测试

区别于传统的迭代局部搜索算法，本文提出的 ILS-MP 算法除了设计了两种不同的邻域结构以外，还设计了多扰动机制：基于禁忌的扰动、基于构造的扰动以及随机的扰动，在局部搜索陷入局部最优之后，会随机选择一种扰动类型移动解到新的搜索空间进行迭代搜索。具体地，会以概率P选择基于禁忌的扰动，以

概率$(1-P)\times Q$选择基于构造的扰动，另以概率$(1-P)\times(1-Q)$选择随机扰动。这三种扰动方式对解的优良结构保存和解的疏散性的改变不一样，禁忌扰动最能保持搜索到的解结构，而随机扰动能最大程度增强算法的疏散性，不同的概率对最终扰动的效果有一定的影响。为了测定扰动的概率（即 P 和 Q 的值），本文选取工件数量为 50，T 的值为 0.2，R 值为 0.4 的 50 个算例进行测试，每个算例计算 10 次并选取最好的计算结果。图 4-4 给出了扰动类型选择概率对目标函数值的影响，用于比较的目标函数值是 P、Q 不同取值 50 个算例目标函数值的平均值，P 和 Q 的取值均为 0.1、0.2、0.3…直到 0.9，共 81 种不同的取值。很明显可以看出，当 P 的取值为 0.7，Q 的取值为 0.5 时，计算结果最好，而且 P 和 Q 的取值在这二者附近时的求解效果也不差。从以上结果可以看出，基于禁忌的搜索由于保存了解的优良结构，而且由于接受了差解，具备一定的疏散性，因此选择概率高一点，计算结果会稍好。但同时也需要以一定概率进行构造扰动或者随机扰动，进一步增强搜索的疏散性，计算效果是最好的。

4.3 实验结果对比

SMSP-LEQT 问题曾吸引了国内外众多学者的研究，本节主要是对比 ILS-MP 算法求解 SMSP-LEQT 以及使用其它已有的前沿算法求解的结果，包括束搜索（RBS）和四种基于种群优化的启发式算法（两种遗传算法 GA 和 GA_IN 以及两种混合进化算法 MA 和 MAIN），四种种群优化算法的主要区别是在初始解的构造和局部搜索的使用上。在已有的算法中，就求解质量来说，求解 SMSP-LEQT 问题性能表现最好的当属 MA_IN 算法。

4.3.1 相较 RBS 算法改进幅度

RBS[30]、GA、GA_IN[23]、MA 以及 MA_IN[24]算法是目前已有的求解 SMSP-LEQT 问题的算法，其中 RBS 提出的时间最早，求解结果稍逊；GA 和 GA_IN 属于遗传算法，结果求解结果较好；MA 和 MA_IN 是由 Valente 提出的混合进化算法，是目前最优秀的求解 SMSP-LEQT 问题的算法。本节主要比较 GA 等四种算法以及本文提出的 ILS-MP 算法相对于 RBS 的求解效果。

表 4-3 各类启发式算法与 RBS 算法结果对比

n	Heur.	Low var %imp			High var %imp		
		Best	*Avg.*	*Worst*	*Best*	*Avg.*	*Worst*
10	GA	0.02	0.18	1.58	0.36	0.31	2.42
	GA_IN	0.02	0.01	0.10	0.35	0.17	1.48
	MA	0.02	0.01	0.09	0.37	0.35	0.28
	MA_IN	0.02	0.02	0.02	0.37	0.37	0.35
	ILS-MP	0.02	0.02	0.02	0.37	0.37	**0.37**
20	GA	0.11	0.26	1.74	0.36	0.96	3.95
	GA_IN	0.1	0.05	0.03	0.51	0.17	1.33
	MA	0.11	0.07	0.13	0.66	0.5	0.02
	MA_IN	0.11	0.11	0.08	0.66	0.6	0.45
	ILS-MP	0.11	0.11	**0.11**	0.66	**0.65**	**0.6**
30	GA	0.22	0.20	1.42	0.35	0.96	4.07
	GA_IN	0.23	0.18	0.11	0.71	0.14	0.67
	MA	0.25	0.21	0.03	0.98	0.76	0.23
	MA_IN	0.25	0.24	0.22	0.98	0.89	0.71
	ILS-MP	0.25	**0.25**	**0.24**	**0.99**	**0.96**	**0.88**
40	GA	0.43	0.03	1.49	0.59	0.70	3.40
	GA_IN	0.45	0.4	0.33	1.08	0.56	0.18
	MA	0.48	0.44	0.37	1.45	1.23	0.81
	MA_IN	0.48	0.46	0.44	1.45	1.36	1.17
	ILS-MP	0.48	**0.48**	**0.46**	**1.48**	**1.44**	**1.34**
50	GA	0.42	0.05	1.63	0.68	0.41	2.48
	GA_IN	0.44	0.4	0.33	1.27	0.8	0.17
	MA	0.47	0.45	0.39	1.67	1.45	1.06
	MA_IN	0.47	0.46	0.44	1.69	1.59	1.42
	ILS-MP	0.47	**0.47**	0.44	**1.71**	**1.66**	**1.5**
75	GA	0.63	0.14	1.15	1	0.05	2.01
	GA_IN	0.71	0.67	0.61	1.76	1.4	0.94
	MA	0.74	0.72	0.67	2.14	1.94	1.66
	MA_IN	0.74	0.73	**0.72**	2.18	2.09	1.96
	ILS-MP	**0.75**	0.73	0.64	**2.22**	**2.16**	1.96

n	Heur.	Low var %imp			High var %imp		
		Best	*Avg.*	*Worst*	*Best*	*Avg.*	*Worst*
100	GA	–	–	–	–	–	–
	GA_IN	0.76	0.73	0.68	2.31	2.01	1.62
	MA	0.8	0.78	0.76	2.73	2.57	2.53
	MA_IN	0.8	**0.79**	**0.77**	2.78	2.69	**2.58**
	ILS-MP	**0.81**	0.78	0.68	**2.83**	**2.77**	2.56

给出了各类启发式算法和 RBS 结果对比，其中也包括本文提出的 ILS-MP 算法。为了减少算法的偶然性因素，对于 L 和 H 两种类型、不同规模、不同 T 和 R 值的算例均计算 10 次，并比较最好、最差以及平均的计算结果。为了比较和 RBS 算法的区别，计算并比较各类算法相较于 RBS 的改进幅度：$\%imp = (OFV_{RBS} - OFV)/OFV_{RBS} \times 100$。其中$OFV_{RBS}$表示通过 RBS 算法获得的目标函数值，$OFV$表示通过其它算法获得的目标函数值。要注意的是，表中加粗的表示该算法在该项上优于其它算法（不包括相等的情况）。从表中不难看出，本文提出的 ILS-MP 算法在所有算例的计算上要优于 RBS 算法，而且相较于表现较好的 MA 和 MA_IN 算法，ILS-MP 在很多算例上能持平或者具有微弱的优势，尤其是算例规模中等（30~50）的算例，IL-MP 的改进幅度最大，而对于最大规模（100 个工件）的算例，IL-MP 的改进幅度也和 MA_IN 不相上下。

图 4-5 给出了 ILS-MP 算法和 MA_IN 算法相较于 RBS 算法的改进幅度对比，对于工件数量为 100 的大型算例，ILS-MP 相较于 RBS 算法在 L 类型算例的改进的平均幅度约为 0.78%，在 H 类型算例的平均改进幅度约为 2.77%；而 MA_IN 在 L 类型算例上的平均改进幅度为 0.79%，在 H 类型算例上的平均改进幅度为 2.69%。可以看出，无论对于哪种类型的算例，ILS-MP 和 MA_IN 算法在求解 SMSP-LEQT 问题时相较于 RBS 的改进幅度相差不大，在 L 型算例问题上，两种算法相对于 RBS 均只有微弱优势，但对于 H 型算例，这两种算法相较于 RBS 均有约 3%的改进幅度。这也反映出 L 型算例的求解相对 H 型算例来说要更加容易，

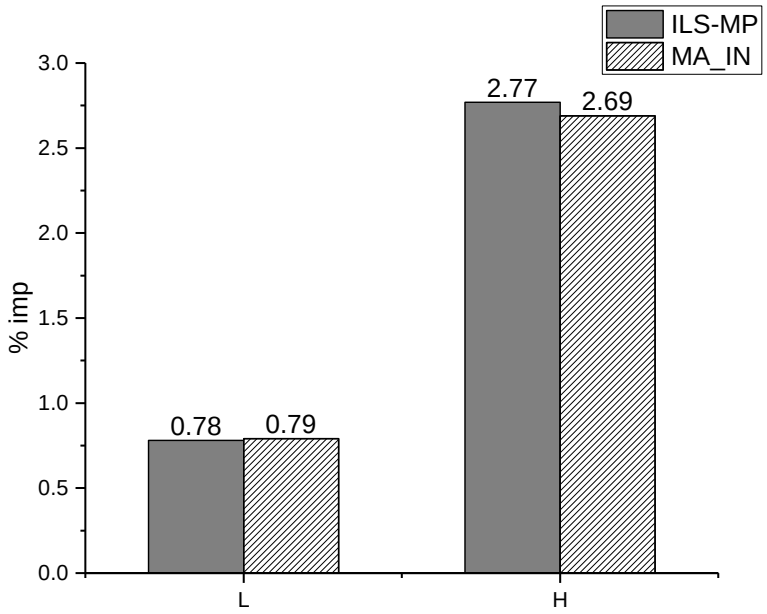


图 4-5 ILS-MP 和 MA_IN 算法相较于 RBS 的改进幅度（*n*=100）

4.3.2 ILS-MP 与 MA_IN 算法对比

表 4-4 ILS-MP 与 MA_IN 算法对比——改进幅度

n	Low var			High var		
	Best	*Avg.*	*Worst*	*Best*	*Avg.*	*Worst*
10	0.000	0.000	0.003	0.000	0.003	0.019
15	0.000	0.004	0.020	0.000	0.023	0.095
20	0.000	0.008	0.031	0.002	0.047	0.144
25	0.000	0.010	0.030	0.004	0.081	0.188
30	0.000	0.009	0.023	0.009	0.071	0.172
40	0.001	0.011	0.009	0.016	0.088	0.180
50	0.002	0.005	-0.035	0.018	0.075	0.082
75	0.005	-0.002	-0.091	0.045	0.079	-0.029
100	0.007	-0.007	-0.119	0.068	0.092	-0.053

在前人的研究成果中，Valente 提出的 MA_IN 算法[24]是性能最优的算法，为了分析 ILS-MP 的算法求解性能，本节将比较 ILS-MP 和 MA_IN 算法在不同规

表 4-5 ILS-MP 与 MA_IN 算法对比——改进算例占比

Criteria	*n*	Low var			High var		
		<	=	>	<	=	>
Best	10	0.0	100.0	0.0	0.0	100.0	0.0
	15	0.0	100.0	0.0	0.0	100.0	0.0
	20	0.0	100.0	0.0	1.0	99.0	0.0
	25	0.2	99.8	0.0	2.3	97.4	0.3
	30	0.4	99.6	0.0	4.3	95.5	0.3
	40	1.6	98.4	0.0	11.0	88.0	1.0
	50	4.5	95.5	0.0	18.0	79.5	2.5
	75	12.1	87.9	0.0	30.2	67.7	2.2
	100	16.5	83.5	0.0	37.1	61.3	1.7
Avg	10	0.5	99.5	0.0	1.3	98.7	0.1
	15	5.9	94.1	0.0	11.9	86.3	1.8
	20	14.1	85.8	0.1	24.1	72.1	3.8
	25	18.8	81.0	0.2	34.1	62.4	3.5
	30	21.8	77.8	0.3	40.0	57.0	3.0
	40	24.8	74.5	0.7	44.0	51.1	4.9
	50	25.3	72.6	2.1	46.0	47.8	6.3
	75	24.3	68.9	6.8	44.5	44.7	10.8
	100	24.1	67.6	8.3	47.9	39.5	12.6
Worst	10	0.8	99.2	0.0	1.8	98.1	0.1
	15	6.8	93.1	0.1	12.8	85.1	1.8
	20	14.8	85.0	0.3	23.6	72.3	4.1
	25	20.0	79.8	0.3	34.0	62.5	3.5
	30	23.9	75.7	0.4	40.3	56.2	3.6
	40	26.6	72.5	0.9	44.1	49.4	6.5
	50	27.4	70.3	2.3	47.0	45.7	7.3
	75	26.4	66.1	7.5	46.5	40.8	12.8
	100	26.2	63.3	10.5	48.3	36.0	15.7

模不同类型的算例上的计算结果，对比分析两种算法各自的优劣性。表 4-4 给出了 ILS-MP 相较于 MA_IN 算法的改进幅度，分别给出了 L 和 H 两种类型、不同工件数量的算例、10 次计算的最优（Best）、最差（Worst）和平均（Avg.）

目标函数值改进幅度。从表 4-4 中不难看出，在绝大多数情况下，ILS-MP 相较于 MA_IN 算法的改进幅度为正值，即本论文提出的 ILS-MP 的计算结果要比 MA_IN 算法有改进，但当工件数量$n \geq 75$时，MA_IN 算法在最坏和平均情况下的计算结果要比 ILS-MP 算法略占优势（H 类算例的平均情况依然是 ILS-MP 更优）。在最好情况下的计算结果，无论多大规模的算例，ILS-MP 均能获得比较好的计算结果，这说明 ILS-MP 在多次计算的情况下容易获得更好的计算效果，但求解的稳定性相对于 MA_IN 算法较差，这是因为 MA_IN 算法利用了种群算法来求解，其疏散性更强，而 ILS-MP 主要利用的是局部搜索算法，集中性更强。

表 4-5 同样给出的是 ILS-MP 与 MA_IN 算法的计算结果对比，但比较的是 ILS-MP 相对于 MA_IN 改进算例的占比，同样是依据表 4-4 计算时采用的 L 和 H 两种类型、10~100 工件数量的算例，并分别比较 10 次计算中统计平均结果、最好结果和最差结果时的改进算例比例。

表 4-5 中（<）列表示 ILS-MP 的计算结果优于 MA_IN 算法的算例占比，即 ILS-MP 算法求解的目标函数惩罚值要小于 MA_IN 算法；（=）列表示两种算法计算结果相同的算例占比；（>）列则表示该部分的算例 MA_IN 算法的计算结果要比 ILS-MP 算法更好。从表中数据不难看出，对于 L 型的算例，绝大多数算例上 ILS-MP 和 MA_IN 的算法表现是相同的，相等算例的比例最低也有 63.3%，ILS-MP 更优的算例比例也不超过 30%。但对于 H 型的算例，差距就比较明显，ILS-MP 更优的算例比例最大接近 50%。对于小规模算例，两种算法都能得到比较优的结果，尤其是对于工件数量为 10 和 15 的算例，相等结果的比例均接近 100%；对于中等规模的算例（工件数量为 20 至 40），ILS-MP 表现出微弱的优势；对于大规模的算例，即工件数量不少于 50 的算例，ILS-MP 的优势就比较明显，改进的算例比例约为 1/3！，这依赖于我们设计的评估策略，对于大规模算例，也可以快速评估迭代多次得到很优的结果。

4.3.3 与最优解对比

在网站 http://fep.up.pt/docentes/jvalente/benchmarks.html 提供的算例中，小规模的算例（10、15 和 20 个工件数量）存在最优解，为了比较各个算法求解结果

和最优解的差距，本文统计了六种算法（已有的 RBS、GA、GA_IN、MA、MA_IN 算法和本文提出的 ILS-MP 算法）的计算结果和最优解的比较，如表 4-6 所示。要注意的是，在统计计算结果时，每个算例会计算 10 次，并选取最好的计算结果来作为算法的求解结果。表中 *dev* 表示和所有该类型该规模下算例目标函数值的平均值和最优解的差距，即$\%dev = (H - O)/O \times 100$，H 表示上面提到的算法求解的结果，O 表示最优解。另外，opt 表示目标函数值和最优解相等的算例数量所占的比例。显然 ILS-MP 在小规模算例上的计算效果和精确算法很相似，绝大多数算例都能得到最优解，工件数量为 10 和 15 的算例均能找到最优解，其它规模和类型的算例也能找到绝大多数的最优解，而且解的值和最优解的相差幅度不超过 0.02%。显然无论是得到最优解的算例数量还是未达到最优解的相差幅度，ILS-MP 都优于其它算法。

表 4-6 各类算法计算结果和最优解对比

Heur.	T	n = 10		n = 15		n = 20	
		%dev.	%opt	%dev.	%opt	%dev.	%opt
L	RBS	0.02	97.00	0.03	83.17	0.13	73.25
	GA	0.20	81.09	0.30	50.83	0.38	29.98
	GA_IN	0.03	79.59	0.05	56.53	0.06	44.23
	MA	0.01	99.56	0.02	97.31	0.04	91.75
	MA_IN	0.00	99.89	0.00	98.41	0.01	95.53
	ILS-MP	**0.00**	**100.00**	**0.00**	**100.00**	**0.00**	**99.50**
H	RBS	0.46	88.83	0.89	75.83	0.81	56.83
	GA	0.69	68.87	1.26	33.03	1.66	12.58
	GA_IN	0.55	60.18	0.89	28.88	0.86	17.48
	MA	0.02	98.98	0.09	93.8	0.16	84.37
	MA_IN	0.00	99.73	0.03	96.03	0.06	89.33
	ILS-MP	**0.00**	**99.8**	**0.01**	**98.20**	**0.01**	**94.3**

4.3.4 不同类型算例求解效果比较

本文求解的 ILS-MP 算法是随机生成的算例，L 和 H 表示工件加工时间的分布区间大小，T 和 R 分别表示工件工期的期望和取值空间分布，本节主要对于 L

和 H 类型、不同 T 和不同 R 的取值时，算法求解 ILS-MP 问题的效果。这里选取的是工件数量为 20 的算例，T 有 0.0、0.2、0.4、0.6、0.8 和 1.0 这 6 中不同的取值，R 有 0.2、0.4、0.6 和 0.8 这 4 种不同的取值，每种类型的算例计算 10 次，并比较最优的计算结果。这表 4-7 给出了各算法在求解工件数量为 20 的算例时和最优解的相差幅度，并按照延误因子 T 和工期分布范围 R 的不同取值给出相应的值。我们发现，当$T < 0.6$ 时，我们不难发现各种算法求解该类问题时，目标函数值和最优解的差值幅度相对较高，我们分析是因为当 T 较小时，大多数工件的工期都比较晚，大多数工件都在工期之前完成了加工。而对于$T \geq 0.6$的算例，这些启发式算法都能获得比较好的结果，都能计算得到最优解或者近似最优解。从表中可以看出，不论是哪种算法，T=0.2 和 T=0.4 时，目标函数值相对最优解的幅度最大。对于本文提出的 ILS-MP 算法，L 型算例的相差幅度不超过 0.02%，H 型算例的相差幅度不超过 0.155%。$T \geq 0.6$的算例，ILS-MP 得到的和最优解十分接近，但对于$T < 0.6$的算例（尤其是 H 型），目标函数值和最优解存在微弱的差距，而且 R=0.4 和 0.6 时，相较于 R 的其它取值，计算结果要更差。综合上面的分析不难看出，本文提出的 ILS-MP 算法在求解 L 型算例时效果要更好，而且 T 越大效果越好。当工期范围相对比较合适时（即 R 接近于 0.5），计算要相对复杂一些。

表 4-7 各种类型算例的求解结果对比（n=20）

Heur.	T	Low var				High var			
		R=0.2	*R=0.4*	*R=0.6*	*R=0.8*	*R=0.2*	*R=0.4*	*R=0.6*	*R=0.8*
RBS	0.0	0.000	0.000	0.003	0.007	0.040	0.115	0.279	0.147
	0.2	1.857	0.208	0.227	0.266	3.974	4.516	2.308	1.433
	0.4	0.012	0.031	0.077	0.505	0.682	0.335	0.718	3.790
	0.6	0.003	0.002	0.000	0.000	0.673	0.133	0.057	0.045
	0.8	0.000	0.000	0.000	0.000	0.043	0.059	0.001	0.008
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000
GA	0.0	0.014	0.022	0.031	0.039	0.418	0.986	1.623	1.620
	0.2	0.073	0.256	0.397	0.458	1.013	5.955	7.442	6.676
	0.4	0.043	0.182	0.741	5.253	0.243	0.424	1.469	9.291
	0.6	0.031	0.106	0.236	0.737	0.148	0.256	0.451	0.853
	0.8	0.023	0.054	0.086	0.127	0.101	0.144	0.206	0.229
	1.0	0.013	0.025	0.037	0.058	0.050	0.082	0.086	0.107

Heur.	T	Low var				High var			
		R=0.2	*R=0.4*	*R=0.6*	*R=0.8*	*R=0.2*	*R=0.4*	*R=0.6*	*R=0.8*
GA_IN	0.0	0.010	0.009	0.019	0.022	0.191	0.317	0.515	0.529
	0.2	0.038	0.200	0.236	0.341	0.749	3.215	3.761	3.581
	0.4	0.011	0.020	0.057	0.460	0.240	0.390	1.058	4.114
	0.6	0.009	0.007	0.004	0.004	0.161	0.228	0.320	0.641
	0.8	0.004	0.003	0.002	0.001	0.094	0.175	0.134	0.135
	1.0	0.001	0.001	0.000	0.001	0.012	0.029	0.030	0.034
MA	0.0	0.001	0.004	0.004	0.008	0.025	0.075	0.129	0.114
	0.2	0.142	0.125	0.192	0.101	0.338	0.993	0.992	0.578
	0.4	0.029	0.045	0.013	0.325	0.108	0.086	0.114	0.267
	0.6	0.001	0.000	0.000	0.000	0.031	0.023	0.002	0.000
	0.8	0.000	0.000	0.000	0.000	0.002	0.001	0.000	0.000
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MA_IN	0.0	0.000	0.000	0.001	0.000	0.015	0.027	0.026	0.030
	0.2	0.009	0.058	0.045	0.049	0.098	0.343	0.453	0.255
	0.4	0.000	0.001	0.003	0.038	0.026	0.015	0.036	0.145
	0.6	0.000	0.000	0.000	0.000	0.008	0.002	0.001	0.000
	0.8	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.001
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
ILS-MP	0.0	0.000	0.000	0.000	0.000	0.000	0.001	0.007	0.004
	0.2	0.000	0.000	0.001	0.002	0.006	0.155	0.078	0.070
	0.4	0.000	0.000	0.000	0.001	0.000	0.001	0.001	0.004
	0.6	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.000
	0.8	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

4.3.5 计算时间分析

除了比较求解得到的目标函数值大小，计算效率也是衡量算法质量的一个重要指标。

表 4-8 给出了各种算法在求解不同类型不同规模的算例上的求解时间。为了减少偶然性，对于每个算例会计算 10 次，然后统计搜索到最好结果的最早时间，并比较 10 次计算时间的平均值。因为使用的机器不同，根据不同的机器配置，换算成本论文使用的机器处理时间来进行同等条件下的比较。不难发现，本文提出的 ILS-MP 算法在计算效率上相较于其它算法具有明显的优势，当工件数量

$n \leq 20$时，每个算例的平均计算时间少于 0.001s，即便是对于规模较大的 100 工件数量的算例，计算时间也不超过 1s。另外，我们发现，L 型算例的求解时间要普遍短于 H 型算例，这也说明当工件加工时间分布区间较大的时候，求解的难度要更大一些

表 4-8 算法计算时间

Var.	Heur.	n=10	n=20	n=30	n=40	n=50	n=75	n=100
L	RBS	0.001	0.004	0.009	0.019	0.033	0.100	0.227
	GA	0.012	0.174	0.221	0.492	0.932	3.028	-
	GA_IN	0.003	0.015	0.042	0.088	0.158	0.480	1.070
	MA	0.003	0.011	0.037	0.089	0.172	0.612	1.617
	MA_IN	0.004	0.011	0.033	0.079	0.153	0.545	1.445
	ILS-MP	0.000	0.001	0.002	0.006	0.013	0.068	0.205
H	RBS	0.001	0.004	0.009	0.020	0.035	0.109	0.250
	GA	0.013	0.008	0.242	0.549	1.051	3.476	-
	GA_IN	0.004	0.020	0.052	0.106	0.192	0.593	1.353
	MA	0.004	0.013	0.041	0.102	0.203	0.767	2.108
	MA_IN	0.004	0.011	0.036	0.089	0.172	0.649	1.790
	ILS-MP	0.000	0.001	0.004	0.021	0.052	0.263	0.843

5 研究工作总结

5.1 工作总结

本文研究的基于提前和延误惩罚的单机调度问题属于经典的组合优化问题，组合优化问题一般都属于 NP 难问题，这类问题往往求解复杂度高，很难在多项式时间内求解。由于单机调度问题的复杂性，在学术上该问题具有极大的理论研究价值，而且实际生活中该问题也很普遍，广泛存在于机器加工、飞机起降调度等场景中，具有较高的工业价值。正因如此，单机调度问题吸引了国内外众多优化算法研究者的关注。

区别于一般的单机调度问题，本文研究的是基于提前惩罚和延误惩罚的单机调度问题，目标函数更加复杂，问题也很难得到简化，复杂程度更高。在本文中，我们采取的是启发式算法来求解，因为启发式算法已经在很多组合优化问题（包括排课表问题、图着色问题、P-Center 问题和 TSP 问题等等）中都得到了广泛的应用，并验证了其求解的效果。具体地，本文采取的是比较简单易实现的迭代局部搜索 ILS 来求解。为了探索更广阔的邻域，提出了两种不同的邻域动作——工件的插入和交换，每次邻域搜索都会随机选择一种方式来生成邻域候选集。同时，针对这两种邻域设计了快速评估策略，旨在尽可能快速地从邻域候选集中挑选出最优的邻域解，为了进一步加快搜索效率，在不影响解质量的基础上对邻域动作的距离阈值做了限制，避免搜索时间的浪费。另外，由于局部搜索容易陷入局部最优，本文创新性提出了三种扰动机制，包括基于禁忌的扰动、基于构造的扰动以及随机的扰动，这三种方式对扰动的幅度影响不同，我们通过设置三种扰动方式的选择概率来控制扰动的幅度，既不能太大丢失局部最优解的结构，也不能太小到达不了新的解空间。

由于启发式算法不具备普适性，算法设计中存在多种参数，包括邻域动作选择概率、扰动方式选择概率等，为了得到最佳的参数设置，我们对各类参数分别做了测试，在文中描述了三个最重要的参数——邻域动作选择概率、邻域动作距离阈值和扰动类型选择概率的测试过程，并选取了最佳的参数值。为了评估 ILS-

MP 算法在求解 SMSP-LEQT 问题上的性能，在标准算例集上进行了计算测试，并同当前已有的四种前沿算法（RBS、GA、GA_IN、MA 和 MA_IN）进行了详细的数据对比分析，包括不同类型不同规模的目标函数值对比以及求解时间对比，分析表明本文提出的 ILS-MP 算法在求解 SMSP-LEQT 问题上具有卓越的性能表现。

5.2 研究创新点

好的启发式算法的设计一定是集中性和疏散性的结合，这个概念贯彻在了启发式算法的各个环节。以本文为求解 SMSP-LEQT 问题提出的 ILS-MP 算法为例，在邻域动作设计的时候，考虑了两种邻域动作，这是由于邻域搜索本身是需要下降搜索收敛的，集中性很强。但我们考虑了两种不同的邻域结构，这使得在每次邻域搜索的时候选择性会更多，提升了搜索的疏散性，而邻域动作的选择概率自然就决定了邻域搜索的集中性和疏散性。为了进一步提升搜索效率，我们对邻域动作的距离阈值还做了限制，距离阈值越大，邻域搜索的范围越小，越容易收敛，集中性越强，因此调整合适的距离阈值大小也是集中性和疏散性平衡的体现。在实现扰动机制的时候，由于扰动本身动作是有助于提升解的疏散性的，扰动的幅度越大，疏散性越强，反之疏散性越小。为了调节更合适的扰动幅度，本文创新性的提出了多扰动策略，设计了三种不同的扰动方式，其中基于禁忌的扰动方式对解的扰动幅度最小，随机的扰动对解的扰动幅度最大，平衡好三种扰动方式的选择概率也可以平衡搜索的集中性和疏散性。可以看到，在 ILS-MP 设计的各个环节，我们始终都遵循着集中性和疏散性平衡的原则，这也是 ILS-MP 在求解 SMSP-LEQT 问题时具备良好效果的原因。

参考文献

[1] Schaller J. Single machine scheduling with early and quadratic tardy penalties ☆ [J]. Computers & Industrial Engineering, 2004, 46(3): 511-32.

[2] Qin T, Peng B, Benlic U, et al. Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling [J]. Computers & Operations Research, 2015, 61(C): 81-8.

[3] Lenstra J K, Kan A H G R, Brucker P. Complexity of Machine Scheduling Problems [J]. Annals of Discrete Mathematics, 1977, 1(4): 343-62.

[4] Lüaba Z. Adaptive Tabu Search for course timetabling [J]. European Journal of Operational Research, 2010, 200(1): 235-44.

[5] Peng B, Lü Z, Cheng T C E. A tabu search/path relinking algorithm to solve the job shop scheduling problem [J]. Computers & Operations Research, 2014, 53(53): 154-64.

[6] Kearns M, Suri S, Montfort N. An experimental study of the coloring problem on human subject networks [J]. Science, 2006, 313(5788): 824-7.

[7] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem [J]. IEEE Trans on Ec, 1997, 1(1): 53-66.

[8] 熊锐, 吴澄. 车间生产调度问题的技术现状与发展趋势 [J]. 清华大学学报(自然科学版), 1998, 10): 55-60.

[9] Johnson S M. Optimal two- and three-stage production schedules with setup times included [J]. Naval Research Logistics Quarterly, 1954, 1(1): 61–8.

[10] Conway R W, Maxwell W L, Miller L W. Theory of Scheduling [J]. Addison-Wesley Publishing Co, Reading, Mass-London-Don Mills, Ont, 1967, x,294.

[11] Graham R L, Lawler E L, Lenstra J K, et al. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey [J]. Annals of Discrete Mathematics, 1979, 5(1): 287-326.

[12] Abdul-Razaq T S, Potts C N, Wassenhove L N V. A survey of algorithms for the

single machine total weighted tardiness scheduling problem [J]. Discrete Applied Mathematics, 1990, 26(2): 235-53.

[13]Potts C N, Wassenhove L N V. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem [J]. Operations Research, 1985, 33(2): 363-77.

[14]Sourd F. Earliness–tardiness scheduling with setup considerations [J]. Computers & Operations Research, 2005, 32(7): 1849-65.

[15]Valente J M S, Schaller J E. AN EXACT APPROACH FOR THE SINGLE MACHINE SCHEDULING PROBLEM WITH LINEAR EARLY AND QUADRATIC TARDY PENALTIES [J]. Asia-Pacific Journal of Operational Research, 2008, 25(02): 169-86.

[16]Schaller J. A comparison of lower bounds for the single-machine early/tardy problem [J]. Computers & Operations Research, 2007, 34(8): 2279-92.

[17]Tanaka S, Araki M. An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times [J]. Computers & Operations Research, 2013, 40(1): 344-52.

[18]Cicirello V A, Smith S F. Enhancing Stochastic Search Performance by Value-Biased Randomization of Heuristics [J]. Journal of Heuristics, 2005, 11(1): 5-34.

[19]Gagné C, Price W L, Gravel M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times [J]. Journal of the Operational Research Society, 2002, 53(8): 895-906.

[20]Potts C N, Wassenhove L N V. Single Machine Tardiness Sequencing Heuristics [J]. A I I E Transactions, 1991, 23(4): 346-54.

[21]Holsenback J E, Russell R M, Markland R E, et al. An improved heuristic for the single-machine, weighted-tardiness problem [J]. Omega, 1999, 27(4): 485-95.

[22]Feo T A, Sarathy K, Mcgahan J. A grasp for single machine scheduling with sequence dependent setup costs and linear delay penalties [J]. Computers & Operations Research, 1996, 23(9): 881-95.

[23]Valente J M S, Gon, Alves J, et al. A genetic algorithm approach for the single

machine scheduling problem with linear earliness and quadratic tardiness penalties [J]. Computers & Operations Research, 2009, 36(10): 2707-15.

[24] Valente J M S, Schaller J E. Improved heuristics for the single machine scheduling problem with linear early and quadratic tardy penalties [J]. European Journal of Industrial Engineering, 2010, 4(1): 99-129.

[25] Rubin P A, Ragatz G L. Scheduling in a sequence dependent setup environment with genetic search [M]. Elsevier Science Ltd., 1995.

[26] Tan K C, Narasimhan R. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach [J]. Omega, 1997, 25(6): 619-34.

[27] Armentano V, Renatamazzini. A genetic algorithm for scheduling on a single machine with set-up times and due dates [J]. Production Planning & Control, 2000, 11(7): 713-20.

[28] França P M, Mendes A, Moscato P. A memetic algorithm for the total tardiness single machine scheduling problem [J]. European Journal of Operational Research, 2001, 132(1): 224-42.

[29] Lin S W, Ying K C. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics [J]. International Journal of Advanced Manufacturing Technology, 2007, 34(11-12): 1183-90.

[30] Valente J M S. BEAM SEARCH HEURISTICS FOR THE SINGLE MACHINE SCHEDULING PROBLEM WITH LINEAR EARLINESS AND QUADRATIC TARDINESS COSTS [J]. Asia-Pacific Journal of Operational Research, 2009, 26(03): 319-39.

[31] Tasgetiren M F, Pan Q K, Liang Y C. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times [J]. Computers & Operations Research, 2009, 36(6): 1900-15.

[32] Kirlik G, Oguz C. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine [M]. Elsevier

Science Ltd., 2012.

[33]Xu H, Lü Z, Yin A, et al. A study of hybrid evolutionary algorithms for single machine scheduling problem with sequence-dependent setup times [J]. Computers & Operations Research, 2014, 50(10): 47-60.

[34]Xu J, Bailey G. The airport gate assignment problem: mathematical model and a tabu search algorithm [M]. 2001.

[35]Stojković G, Soumis F, Desrosiers J, et al. An optimization model for a real-time flight scheduling problem [J]. Transportation Research Part A Policy & Practice, 2002, 36(9): 779-88.

[36]黄文奇, 叶涛. 求解等圆 Packing 问题的拟物型全局优化算法 [J]. 中国科学: 信息科学, 2011, 6): 686-93.

[37]Wang Z, Lü Z, Ye T. Local search algorithms for large-scale load balancing problems in cloud computing [J]. Scientia Sinica, 2015, 45(5): 587.

[38]Glover F. Tabu Search, Part II [J]. Orsa J Computing, 1990, 2(1):

[39]Crama Y, Kolen A W J, Pesch E J. Local search in combinatorial optimization; proceedings of the Artificial Neural Networks: An Introduction to ANN Theory and Practice, F, 1995 [C].

[40]Goldberg D E. Genetic Algorithm in Search Optimization and Machine Learning [J]. Addison Wesley, 1989, xiii(7): 2104–16.

[41]Colorni A. Distributed Optimization by Ant Colonies; proceedings of the European Conference on Artificial Life, F, 1991 [C].

[42]Stützle T. Iterated local search for the quadratic assignment problem [J]. European Journal of Operational Research, 2006, 174(3): 1519-39.

[43]Kalczynski P J, Kamburowski J. An improved NEH heuristic to minimize makespan in permutation flow shops [J]. Computers & Operations Research, 2008, 35(9): 3001-8.

[44]Valente J M S. Heuristics for the single machine scheduling problem with early and quadratic tardy penalties [J]. European Journal of Industrial Engineering, 2007, 1(4):

431-48.