分 类 号_____　　学号 D201177601

学校代码 10487　　密级_____

華中科技大學

博 士 学 位 论 文

求解单机调度问题的启发式算法研究

学位申请人：　徐宏云

学 科 专 业：　计算机软件与理论

指 导 教 师：　吕志鹏　教授

答 辩 日 期：　2015 年 1 月 29 日

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Engineering

Research on the Heuristic Algorithms for the Single Machine Scheduling Problem

Ph.D. Candidate : Xu Hongyun

Major : Computer Software and Theory

Supervisor : Prof. Lü Zhipeng

Huazhong University of Science & Technology

Wuhan 430074, P. R. China

February, 2015

独创性声明

本人声明所呈交的学位论文是我个人在导师的指导下进行的研究工作及取得的研究成果。尽我所知，除文中已标明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，在 ____ 年解密后适用本授权书。

　　不保密 □。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

单机调度问题一直是调度领域的研究热点，是生产调度问题的基础及核心问题之一。大多数单机调度问题已经被证明是NP难度的，因此，单机调度问题的研究不论是在当今国际学术界还是在实际生产领域都是一项具有挑战性的重要课题。

求解NP难度的问题一般有三种方法：精确算法、近似算法和启发式算法。精确算法在求解规模较大的问题时，时间复杂度呈指数级增长，计算时间让人无法忍受；近似算法虽然能在较短的时间内找到解，但是，解的质量跟实际需求相差较远；启发式算法在解决大规模复杂问题时，有可能在合理的时间范围内给出一个满意的解。

启发式算法在各个领域已经受到越来越多的关注，逐渐成为求解NP难度问题的有力工具。本文的研究重点是设计求解单机调度问题的高效启发式算法，并通过实验对算法进行了分析和评价。本文的主要贡献包括：

（1）提出了一种新的“块移动”的邻域结构（$N_{Block\ Move}$），该邻域结构扩大了邻域的搜索空间，并且通过限制相关参数的取值范围来提高搜索的有效性。

（2）针对块移动的邻域结构，提出了一种高效的快速增量评估技术，提高了搜索效率。

（3）在块移动邻域结构以及对块移动邻域结构的快速增量评估技术基础上，提出了一种新的迭代局部搜索算法BILS来求解单机调度问题。

（4）提出了一种基于“公共块”概念的交叉算子BOX（Block Order Crossover Operator）。

（5）提出了一种新的基于相似性和质量的优度函数的种群更新策略。

（6）对比了不同的交叉算子和种群更新策略的组合在混合进化算法中处理单机调度问题时的表现，提出了一种高效的求解单机调度问题的混合进化算法LOX⊕B。

用带准备时间的单机加权延迟调度问题的120个公共算例对BILS进行了测试，BILS找到了所有120个算例的最优解。对于算例18和24，BILS找到最优解分别需要8586.6秒（≈2.4小时）和150169.22秒（≈41.7小时），而Tanaka和Araki的精确算法

找到最优解则分别需要两个星期和30天。与其它几种优秀的启发式算法组成的当前最好解比，BILS改进了34个算例的当前最好解，82个算例的解与当前最好解持平。

用带准备时间的单机加权延迟调度问题的120个公共算例对LOX⊕B进行测试，结果与当前文献中最优秀的启发式算法的最好解OBK、ACO_AP、DPSO、DDE、GVNS相比，改进最好解的个数分别为94、84、66、52、44。与本文新提出的BILS算法比，在平均值的优度和找到最好解的平均时间上都有较大的提高，表明了LOX⊕B相对于BILS更加稳定和高效。

用带准备时间不带权的单机延迟调度问题的64个公共算例对LOX⊕B进行测试。LOX⊕B算法用6037.93秒（≈1.7小时）找到算例prob855 的最好解“256”，改进了由Tanaka和Araki的精确算法用超过30天时间找到的当前最好解“258”，并找到了其余63个算例的最优解或当前最好解。其中，对于算例prob751和prob851，LOX⊕B找到最优解或当前最好解的时间分别为33678.64秒（≈9.4小时）和84405.88秒（≈23.4小时），而Tanaka和Araki的精确算法却分别需要34天和超过30天的CPU时间。LOX⊕B算法对这64个算例的计算结果与当前最好的几种启发式算法相比也有很大的改进。

以上研究成果表明，本文提出的迭代局部搜索算法BILS和混合进化算法LOX⊕B都是高效的求解单机调度问题的启发式算法。在今后的研究中，尝试将BILS和LOX⊕B用于求解其它具有NP难度的组合优化问题。

关键词： NP难度　　单机调度问题　　启发式算法　　邻域结构　　迭代局部搜索　　混合进化算法

Abstract

Single machine scheduling problem is a hot topic in the scheduling field and one of the basic and central problems for the production scheduling problems. Some single machine scheduling problems have proved to be NP-Hard, therefore the single machine scheduling problem becomes a challenging task not only in academic but also in practical production.

For NP-Hard problems, there are always three ways to handle them: exact algorithms, approximate algorithms and heuristic algorithms. When the exact algorithms are employed to solve the large scale complex problems the time complexity is growing exponentially that make the computing time unacceptable. The approximate algorithms could find a solution in a relatively short time but the quality of this solution is far from the actual demand. The heuristic algorithms seek for the high quality solutions at a reasonable computational time in solving large scale problems.

The heuristic algorithms attract more and more attention in various fields and become the powerful tools for solving NP-Hard problems. The focus of this dissertation is to design efficient heuristic algorithms for the single machine scheduling problem, and the heuristic algorithms are analyzed and evaluated by the experimental results. The main contributions of this dissertation can be summarized as follows:

(1) A new neighborhood structure Block Move ($N_{Block\ Move}$) is firstly proposed for the single machine scheduling problem. This new neighborhood structure expands the search scope and improves the effectiveness of search by restricting the range of relevant parameters.

(2) An efficient fast incremental evaluation technique for the Block Move neighborhood is presented and the search efficiency is improved.

(3) Based on the Block Move neighborhood structure and fast incremental evaluation technique an iterated local search algorithm BILS is proposed for solving the single machine scheduling problems.

(4) A common block based crossover operator BOX (Block Order Crossover Operator) is proposed for reproduction in the hybrid evolutionary algorithm.

(5) A similarity-and-quality based population updating strategy is presented.

(6) Comparing the performance of the combinations of crossover operators and population updating strategies in the hybrid evolutionary algorithm for solving single machine scheduling problem, an efficient hybrid evolutionary algorithm LOX⊕B is obtained.

The proposed BILS algorithm is tested on the 120 public instances of the single machine total weighted tardiness problem with sequence-dependent setup times. The BILS could find the optimal solutions for all the 120 instances. For instances 18 and 24, BILS finds the optimal solutions only with 8586.6 seconds (≈2.4 hours) and 150169.22 seconds (≈41.7 hours) respectively, while the Tanaka and Araki's exact algorithm needs 2 weeks and 30 days. Comparing with other excellent heuristic algorithms, BILS outperforms the previous best results for 34 instances and matches the best results for 82 instances.

The proposed LOX⊕B algorithm is tested on the 120 public instances of the single machine total weighted tardiness problem with sequence-dependent setup times. Comparing the LOX⊕B algorithm's results with those of OBK, ACO_AP, DPSO, DDE and GVNS, the numbers of better solutions are 94, 84, 66, 52 and 44 respectively. Comparing with the proposed BILS, the quality of average objective value and the average computational time to reach the best results are both better than BILS, demonstrating that LOX⊕B is more stable and effective than the BILS algorithm.

The proposed LOX⊕B algorithm is tested on the 64 public instances of the single machine total tardiness problem with sequence-dependent setup times. The LOX⊕B is able to find the better solution "256" with 6037.93 seconds (≈1.7 hours) CPU time than the solution of Tanaka and Araki's exact algorithm "258" with more than 30 days for instance prob855. For the remaining 63 instances, the LOX⊕B can reach the optimal or the best solutions obtained by Tanaka and Araki. For instances prob751 and prob851, LOX⊕B finds the optimal or best solutions only with 33678.64 seconds (≈9.4 hours) and 84405.88 seconds (≈23.4 hours) respectively, while the CPU time of Tanaka and Araki's exact algorithm to reach the

optimal or best solutions are 34 days and more than 30 days respectively. The LOX⊕B algorithm achieves highly competitive results for these 64 instances compared with other state-of-the-art heuristic algorithms in the literature.

Above results indicate that the iterated local search algorithm BILS and the hybrid evolutionary algorithm LOX⊕B are the efficient and effective heuristic algorithms for solving the single machine scheduling problem. Further research work will be focused on employing the BILS and LOX⊕B algorithms to solve other NP-Hard combinatorial optimization problems.

Key words: NP-Hard Single machine scheduling problem Heuristic algorithm Neighborhood structure Iterated local search Hybrid evolutionary algorithm

目　录

1 引言

1.1 本课题的来源及研究目的

本研究课题得到以下科研项目的资助：

（1）国家自然科学基金青年基金项目“求解大规模约束满足问题的混合进化算法研究”，项目编号：61100144；

（2）国家自然科学基金面上项目“光网络规划流量疏导优化”，项目编号：61370183；

（3）教育部博士点基金（新教师类）项目“面向频率分配问题的混合算法研究”，项目编号：20110142120081；

（4）教育部“新世纪优秀人才支持计划”。

调度问题是对一个可用的生产资源在时间上进行加工任务集分配，以满足一个或多个性能指标，是在生产实践中广泛应用的运筹学问题[1]。单机调度问题是调度问题中的一类，其特点是所有工件只需要在一台机器上进行加工，生产实践中可以将复杂的调度问题分解成单机调度问题来进行求解。单机调度问题的研究是有着实际意义的课题，它的研究成果对生产加工行业起着十分重要的作用。

本文对单机调度问题及其求解方法进行了长期和细致的研究，期望达到以下目标：

（1）在对当前求解单机调度问题的多种启发式算法进行分析研究的基础上，拟提出高效的求解单机调度问题的启发式算法。

（2）在局部搜索算法中，邻域结构的选择是决定其优劣的决定性因素。通过实验对文献中现有邻域结构进行分析，为求解单机调度问题设计新的邻域结构，提高算法寻找最优解的能力和效率。

（3）启发式算法在探索最优解的过程中，邻域解的评估占用了很大一部分计算时间，要提高算法的搜索效率必须加快对邻域解的评估。通过对目前常用的几种评估策略进行分析和研究，提出新的快速评估邻域解的策略。

（4）在新提出的邻域结构和快速评估策略的基础上，提出新的迭代局部搜索算法来求解单机调度问题。

（5）研究者们在对启发式算法研究的过程中发现，将几种启发式算法按照某种方式结合起来能够“取长补短”，提高算法的寻优能力。本文将局部搜索算法混合到进化算法的框架中，构成混合进化算法，采用混合进化算法求解单机调度问题。

（6）在混合进化算法中，子代的生成和种群更新策略决定种群优劣及其多样性。提出新的交叉算子以及种群更新策略，通过实验对比不同的交叉算子和种群更新策略的组合在混合进化算法中处理单机调度问题时的表现，找出最优组合，提高算法寻找最优解的能力和效率。

（7）在用启发式算法求解单机调度问题的过程中，对启发式算法和调度问题有了较深刻的认识，希望能将用在求解单机调度问题中的新算法和新思想用来求解其它类似问题，为今后的研究工作打下基础并指明方向。

1.2 选题的背景、依据及研究意义

现代制造业的发展要求更高效率的生产调度方法。资料显示在生产制造过程中95%的时间消耗在非加工过程中，这表明调度策略的优劣直接影响到企业制造的成本和效益。生产调度问题是CIMS（Computer Integrated Manufacturing Systems）研究领域中的核心，只有提高生产调度的效率，才能提高整个生产制造过程的效率，才能降低资源的消耗，获得更大的效益。

生产调度问题根据加工系统的复杂程度可以分为多种，其中单机调度问题是最基础的调度类型，也是本文研究的对象，同时，它也是最难的组合优化问题之一。在生产调度研究领域，单机调度问题一直是研究的热点，大规模的单机调度问题至今还没有一种有效的方法能在多项式时间内求出其最优解。对于包含N个工件的单机调度问题，希望找到一个加工序列，这个序列在满足各种约束条件的前提下使得目标函数值最优。根据排列组合的知识可知，N个工件的全排列为$N!$，当$N = 50$时，$N! = 3.04 \times 10^{64}$，若要在其全排列中进行搜索，即使是每秒运行10亿次的计算机，也要用9.6×10^{47}年才能完成。大规模的单机调度问题是NP难问题，这个结论已经得到了充分论证[2–4]，尤其当待加工工件数超过50时，分枝定界、动态规划等算法的效

率已经相对较差[5]。

同单机调度问题一样，在科学研究和生产实践的各个领域还存在很多有待解决的NP难优化问题[6]，如护士排班、材料切割、集装箱运输、电路板布线、设施选址、计算机集群系统负载均衡等。这些问题都是各自行业中棘手的问题，如果能为这些问题找到更好的解决方法，就能够为相应的行业节约资源、提高劳动生产率、降低成本、改进产品质量，提升企业在本行业里的核心竞争力。

求解最优化问题时，人们希望能为其找到最优算法，即找到这些问题的最优解，但在求解的过程中发现，使用精确算法在解决小规模问题时，计算时间是可以接受的，但随着问题规模的不断增大，要找到最优解所花费的时间会是一个天文数字，无法满足实际应用的需求。近似算法虽然能够在合理的时间内找到解，但解的优度与实际需求相差较远。在为各种最优化问题寻找最优解的过程中，人们受大自然的启发，从自然规律中找到和总结出许多解决问题的方法，启发式算法（Heuristic Algorithm）应运而生。

启发式算法是一种基于直观或经验构造的算法，在可接受的花费（指计算时间和空间）下给出待解决组合优化问题每一个实例的一个可行解，该可行解与最优解的偏离程度一般不能被预计[7] 。启发式算法在寻找问题最优解的过程中能表现出某种智能行为，因此在解决复杂的最优化问题时得到了广泛的应用，并取得了巨大的成功[8]。启发式算法由于其数学基础和规则不够完善[9,10]，在学术界对启发式算法的评价有着多种不同的观点。然而在实际应用中，优秀的启发式算法能够在合理的时间范围内找到满足实际需求的求解方案，在求解许多大规模复杂的实际应用问题时，启发式算法可以说是唯一的选择[11]。

生产调度问题由Johnson[12]于20世纪50年代提出，至今已经有60多年的历史。问题一经提出就有就许多学者对其进行深入研究，并提出了多种精确方法（Exact Methods）和近似方法（Approximate Methods）对其进行求解。精确方法包括分支定界（Branch and Bound）、动态规划（Dynamic Programming）及拉格朗日松弛法（Lagrangian Relaxation ）等；近似方法包括近似算法和启发式算法，启发式算法包括局部搜索（Local Search）、禁忌搜索（Tabu Search）、模拟退火算法（Simulated Annealing Algorithm）、蚁群算法（Ant Colony Optimization）、粒子群优

化算法（Particle Swarm Optimization）、神经网络（Artificial Neural Networks）等。精确方法从理论上可以求得问题的最优解，但是由于其计算时间过长，因此在实际应用过程中受到很多限制。近似算法的优势在于计算效率高、算法灵活、但其缺点是求出的解比全局最优解差很多。合理的计算时间和所求出解的最优性使得启发式算法成为求解调度问题的一把利器。

近年来，随着生产规模和加工复杂程度的提高，生产调度作为企业管理和控制的核心技术，越来越被重视。调度问题的研究具有两方面的重要意义：（1）学术价值。很多的调度问题已经被证明为NP难的，使用传统的调度算法已经无法获得满意的效果，因此，必须从理论上进行深入的研究，突破当前算法的局限性，寻找新的有效的算法，为求解调度问题提供新的理论依据；（2）实用价值。随着企业间竞争不断加剧，企业只有通过提高生产效率，才能提高自身的竞争力，采用高效的调度策略是提高效率的必要条件。高效的调度策略能对生产过程中的突发事件做出迅速和灵活的响应，可以显著改善企业的生产性能指标，如提高设备使用效率、降低生产成本、减小库存等，从而提高企业的经济效益。调度问题具有很强的实用背景，在生产加工、车辆调度、航空交通管理、通信网络等领域都有着广泛的应用。因此，研究如何提高求解调度问题算法的效率以及寻找最优解的能力是调度领域与最优化领域的重要课题。

1.3 本文的主要工作及结构安排

本文的工作是在吕志鹏教授的具体指导下完成的。主要工作为：

（1）对单机调度问题以及启发式算法进行了深入的研究。

（2）通过对已有邻域结构的研究与分析，本文提出了一种新的块移动的邻域结构$N_{Block\ Move}$。$N_{Block\ Move}$能够扩大搜索空间，同时又能够通过限制相关参数的取值范围来保证搜索空间的有效性。

（3）邻域解的评估策略是局部搜索算法中最关键的技术之一，通过对带准备时间的单机加权延迟调度问题深入分析，结合启发式算法中常用的评估策略提出一种快速增量评估技术对$N_{Block\ Move}$的邻域解进行评估。

（4）通过实验对块移动邻域结构和邻域解的快速增量评估技术进行了系统的对

比分析和讨论。

（5）在$N_{Block\ Move}$邻域结构及其邻域解的快速增量评估技术的基础上提出了高效的迭代局部搜索算法BILS。用BILS算法求解带准备时间的单机加权延迟调度问题，并与目前最优秀的启发式算法和Tanaka和Araki的精确算法进行了对比与分析。

（6）混合进化算法中交叉算子和种群更新策略的选择决定了算法的效率和寻优能力。本文提出了一种新的交叉算子BOX（Block Order Crossover Operator），以及新的基于相似性和质量的优度函数的种群更新策略。通过实验将三种交叉算子和两种种群更新策略组成的6种组合的表现进行对比，找出了最优组合。

（7）提出高效且稳定的混合进化算法LOX⊕B来求解单机调度问题。用LOX⊕B算法求解带准备时间的单机加权（不加权）延迟调度问题，并将实验结果与当前优秀的启发式算法和精确算法进行了分析对比。

（8）使用适应度距离相关性（Fitness Distance Correlation）分析对三种交叉算子和两种种群更新策略组成的6种组合在处理单机调度问题时的表现进行分析讨论。

本学位论文共分为5章，内容具体安排如下:

第1章 引言。介绍了本课题来源、选题、目的和研究意义、研究的背景知识；阐述了本文的主要工作以及结构安排。

第2章 生产调度问题及其求解算法概述。首先阐述了生产调度问题的相关概念，并对单机调度问题进行了形式化描述；对优化问题中涉及到的一些概念进行了定义；详述了几种经典的启发式算法的设计与分析方法，并对单机调度问题的研究现状进行了较详细的介绍。

第3章 迭代局部搜索算法求解单机调度问题的研究。首先介绍了迭代局部搜索算法的基本原理；提出了一种新的称为块移动的邻域结构$N_{Block\ Move}$；提出了对$N_{Block\ Move}$进行评估的快速增量评估技术；对块移动的邻域结构$N_{Block\ Move}$及其快速增量评估技术进行了系统的讨论和分析；提出新的迭代局部搜索算法BILS求解单机调度问题；用文献中常用的公共算例对BILS算法进行测试，并与当前优秀的启发式算法和Tanaka和Araki的精确算法进行对比，结果证明了BILS算法的有效性。

第4章 混合进化算法求解单机调度问题的研究。首先介绍了混合进化算法的基本框架；提出一种新的交叉算子BOX（Block Order Crossover Operator），以及新的

基于相似性和质量的优度函数的种群更新策略；对比分析了三种交叉算子和两种种群更新策略的组合在求解单机调度问题时的表现，提出新的混合进化算法LOX⊕B求解单机调度问题；用经典公共算例对LOX⊕B算法进行测试，并与当前最优秀的启发式算法和精确算法进行对比分析，证明了使用混合进化算法求解单机调度问题的高效性。

第5章 总结与展望。对全文进行了概括，总结了本文的工作情况、取得的成果、创新点以及对后续研究的展望。

2 生产调度问题及其求解算法概述

生产调度涉及生产过程中的多个流程，它是将有限的资源在一定时间内分配给不同任务的决策过程，它需要在满足各种约束条件下使某个或某些性能指标最优[13]。大多数生产调度问题已经被证明是NP难度问题。NP难度的优化问题存在于各个学科领域，并且许多科学研究和生产实践的问题最终都能转换为优化问题。但是，多年的研究结果表明，对于NP难问题，可能根本不存在高效率的精确完备算法。启发式算法的出现成为求解NP难问题的有效途径，它通过对自然规律的模拟以及对各种经验的总结，运用搜索、比较等方式，实现优胜劣汰，进而逐步接近问题的最优解。

2.1 生产调度问题概述

在现今复杂的生产制造环境中，对于众多生产线上的产品，每一样产品都需要经过不同的步骤和不同的机器来加工完成。对于厂商来说，合理的利用现有的资源提高生产效率是非常必要的。因此，生产决策者们就需要设计出一种好的生产调度方式，减少各个步骤或各台机器之间的等待时间，以满足各种不同的生产需求。如何解决生产加工业中的这些需求，是生产调度问题产生的最重要的原因。

2.1.1 生产调度问题

生产调度是在确定生产任务后，制定合理的生产计划，即按照时间的先后顺序，将有限的劳动力和生产所需的资源分配给各项工作任务，使生产任务能按照生产计划准时完成。生产调度问题解决的是如何制定最优的生产计划。

2.1.2 生产调度问题的分类

生产调度问题的分类方法很多[14]，主要有以下几种:

（1）按需求的产生（requirements generation）来划分。需求的产生直接来自客户的要求或间接来自于库存补货决策，由此产生了open shop和closed shop的区别。

在open shop里，所有的生产订单都来自于客户需求而不用考虑库存量；在closed shop里，库存要满足客户的需求，生产任务通常由库存补货决策确定。

（2）按处理的复杂程度（processing complexity）来划分。处理的复杂程度主要是指加工工件所需要的机器数和工序数，一般可分为以下几种：单机调度问题、并行机调度、flow shop和job shop。单机调度问题中所有工件只有一道工序且只需要在一台机器上完成；并行机调度中存在有并行的多台机器，所有工件只有一道工序且只需要在一台机器上完成，然而，每个工件可以在并行的任意一台机器上完成；flow shop中工件在多台机器上加工，每个工件都有着相同的工序；job shop中工件在多台机器上加工，每个工件的工序并不相同。

（3）按调度标准（scheduling criteria）来划分。调度标准可以分为调度成本和调度性能。调度成本主要包括：固定的生产准备费用、库存成本、延期交付成本等；调度性能主要包括：生产资源的利用率、延误工件的百分比、工件的平均或最大延迟等。在实际生产环境中，对调度的评估是基于成本和性能两方面因素的；在学术理论中，往往只考虑其中一个因素。

（4）按需求的本质来划分（nature of the requirement specification）。按需求的本质可分为确定性调度和不确定性调度两种。确定性调度是指调度问题中所有的参数都是已知或确定的；不确定性调度是指调度问题中某些参数会随相关约束条件的变化而不同。

（5）按调度环境（scheduling environment）划分。调度环境分为静态环境和动态环境。在静态环境下，调度问题需要处理的需求是一个包含全部条件的有限集合，不会有额外的需求产生，也不会有任意一个参数发生变化；在动态环境下，调度问题不仅要处理已知的需求，还需要处理可能会产生的额外需求以及某些参数的变化。

2.1.3 生产调度问题的描述

生产调度问题可以描述为：设$m(i = 1, 2, ..., m)$台机器需要处理$n(j = 1, 2, ..., n)$个工件，每个工件由k道工序组成，对每个工件中的每道工序合理分配到某台机器的某个时间段内加工。每个工件j一般有如下参数：加工时间p_{ij}，表示工件j在机器i上的加工时间，如果工件j的加工时间跟机器无关或者只在一台给定的

机器上加工，则下标i省略；预期完工时间d_j，表示工件j承诺的完工时间；权值w_j，表示工件j相对于其它工件的重要性。

所有调度问题主要由三个元素组成：机器环境（machine environment）、约束和特征（constraints and characteristics）、优化标准（optimality criterion）。用Graham 等人[15]提出的三元素标记法可以表示为$\alpha|\beta|\gamma$ 。其中，α 对应机器环境；β 对应约束及特征；γ 对应优化标准。

α域中对应的机器环境主要包括：

（1）单机环境（1）。单机环境是比较简单的一种机器环境，所有工件都在一台机器上进行加工，且同一时间只有一个工件能被加工。

（2）相同并行机环境（Pm）。m台相同的机器并行，工件j可在任意一台机器上进行加工，且加工时间相同。

（3）一致并行机环境（Qm）。有m台不同速度的机器并行，每台机器的速度不会因为待加工工件的不同而改变。

（4）无关并行机环境（Rm）。有m台不同速度的机器并行，每台机器对于不同的工件加工速度不同。

（5）流水车间环境（Fm）。每个工件以相同的工序在m台机器环境下进行加工。

（6）作业车间环境（Jm）。每个工件以不同的工序在m台机器环境下进行加工。

β域中对应的约束条件及特征主要包括：

（1）提交日期（r_j）。表示工件j必须在此时间之后开始加工。

（2）顺序相关的准备时间（s_{ij}）。表示工件j紧跟在工件i之后加工需要的准备时间。

（3）中断（pmtn）。当工件i在机器上进行加工的时候，可以中断对工件i的加工转而加工另一工件。且当工件i重返该机器上加工时，只需继续工件i未完成的工序即可。

（4）优先约束（prec）。某一个工件，只有当另一个或多个工件加工完成后才能开始加工。

（5）机器适用限制（M_j）。表示工件j必须在集合M_j中的某台机器上加工。

γ域中对应的优化标准主要包括：

（1）最大完工时间（C_{max}）。定义为$max(C_j)$，也称为makespan。

（2）加权总完工时间（$\sum w_jC_j$）。

（3）加权总流水时间（$\sum w_jF_j$）。其中$F_j = C_j - r_j$。类似的可以定义总流水时间（$\sum F_j$）、最大流水时间（F_{max}）。

（4）加权总延迟（$\sum w_jT_j$）。其中$T_j = max(0, C_j - d_j)$。类似的可以定义总延迟（$\sum T_j$）、最大延迟（T_{max}）。

（5）加权总误工记数（$\sum w_jU_j$）。其中$U_j = \begin{cases} 0, C_j \leqslant d_j \\ 1, otherwise \end{cases}$。类似的可以定义总误工记数（$\sum U_j$）。

本文主要研究的是单机调度问题（Single Machine Scheduling，SMS），因此用三元素标记法$\alpha|\beta|\gamma$表示单机调度问题时，$\alpha = 1$。

2.2 单机调度问题

在理论上，单机调度问题可以看成是其它调度问题的特殊形式，因此深入研究单机调度问题可以更好的理解复杂调度问题的结构，同时，求解单机调度问题的启发式算法也可以作为求解复杂调度问题的基础。由此可见，单机调度问题是生产调度领域的一类非常重要和基本的问题，虽然它的模型比较简单，但是，在理论和实际生产当中都具有相当重要的地位。

单机调度问题可以简单描述为，N个相互独立的工件需要在一台机器上加工处理，每个工件都有预期完成时间以及加工时间等参数，此外，不同类型的单机调度问题还可以有不同的约束条件。调度目标就是要找到一个最优的加工序列使得系统总成本最小。表2.1列出了部分被证明为NP难的单机调度问题以及证明者和参考文献。

2.2.1 单机调度问题的描述

本文所研究的带准备时间的单机加权延迟调度问题可以描述为：假设N为待处理的工件总数，J 为所有工件的集合，记为：$J = \{1, 2, \ldots, N\}$；每个工件都有加工时间（processing time），记为p_j；预期完成时间（due date）记为d_j；权值（weight）

表 2.1 被证明为NP难的单机调度问题以及证明者和参考文献

单机调度问题	证明此问题为NP难问题的学者及参考文献
$1\|r_j\|L_{max}$	Lenstra等人[16]
$1\|r_j\|\sum C_j$	Lenstra等人[16]
$1\|prec\|\sum C_j$	Lenstra和Rinnooy Kan[17]
$1\|prec;p_j=1\|\sum w_jC_j$	Lenstra和Rinnooy Kan[17]
$1\|chains;r_j;p_j=1\|\sum w_jC_j$	Lenstra和Rinnooy Kan[18]
$1\|r_j;pmtn\|\sum w_jC_j$	Labetoulle等人[19]
$1\|chains;p_j=1\|\sum U_j$	Lenstra和Rinnooy Kan[18]
$1\|\|\sum w_jU_j$	Karp[20]
$1\|\|\sum T_j$	Lawler[21], Du和Leung[22]
$1\|chains;p_j=1\|\sum T_j$	Leung和Young[23]
$1\|\|\sum w_jT_j$	Lawler[21], Lenstra等人[16]

记为w_j；因为工件的准备时间（setup time）是依赖于整个加工序列的排列顺序的（sequence-dependent)，即某个工件j在整个加工序列中如果排在工件i 后面加工，那么工件j排在工件i后的准备时间记为s_{ij}，反之则记为s_{ji}，且s_{ij}可以不等于s_{ji}；每一个工件j 如果作为加工序列的第一个工件，则有一个初始准备时间，记为s_{0j}。求解目标是，找出这N 个工件在机器上进行加工的一个序列，以满足需要达到的优化标准。

假设一个排列$\pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ $(\pi_i \neq \pi_j, i \neq j)$ 表示一个调度方案，即对这N个工件安排的一个加工序列。π_j表示在整个加工序列中排在第j个位置上的工件，那么工件π_j 的完成时间记为C_{π_j}，则：

$$C_{\pi_j} = C_{\pi_{j-1}} + s_{\pi_{j-1}\pi_j} + p_{\pi_j} \quad \text{(式 2.1)}$$

π_j的延迟（Tardiness）记为T_{π_j}：

$$T_{\pi_j} = max(0, C_{\pi_j} - d_{\pi_j}) \quad \text{(式 2.2)}$$

整个调度方案产生的加权总延迟记为T：

$$T = \sum_{j=1}^{N} w_{\pi_j} T_{\pi_j} \quad \text{(式 2.3)}$$

那么，带准备时间的单机加权延迟调度问题的优化目标就是最小化加权总延迟T。

带准备时间的单机加权延迟调度问题用三元素表示法可以简单记为$1|s_{ij}|\sum w_jT_j$，其优化目标为使加权总延迟$T=\sum_{j=1}^{N} w_{\pi_j}T_{\pi_j}$最小。

不带权带准备时间的延迟调度问题$1|s_{ij}|\sum T_j$其实就是$1|s_{ij}|\sum w_jT_j$问题的一种特殊情况，即所有工件的权值为1或者权值都相等。同样，不带准备时间的加权延迟调度问题$1||\Sigma w_jT_j$也可以看成是$1|s_{ij}|\sum w_jT_j$问题当$s_{ij}=0$的情况。所以，只要能找到一个好的解决$1|s_{ij}|\sum w_jT_j$问题的方法，其它不带权或不带准备时间的单机调度问题也可以迎刃而解。

$1||\sum w_jT_j$问题在文献[16,21]中已被证明是NP难问题，当问题规模较大时是不能在多项式时间内能解决的。那么，带准备时间的$1|s_{ij}|\sum w_jT_j$问题比$1||\sum w_jT_j$问题要更为复杂，因此$1|s_{ij}|\sum w_jT_j$问题也是一个NP难问题。值得注意的是，甚至是不带权不带准备时间的$1||\sum T_j$问题都已被证明是一个NP难问题[21,22]，可见，当工件数目较多时，单机调度问题并不是像看上去的那么简单。

2.3 NP难问题与最优解

2.3.1 NP难问题

70年代初期Cook[24]的研究工作奠定了计算复杂性理论的基础，计算复杂性理论主要研究可计算问题求解的难易程度。一般来说，可用多项式时间算法求解的问题可看作容易处理的问题，而把需要超多项式时间才能求解的问题看作难处理的问题。在计算复杂性理论中，讨论问题的复杂性主要考虑判定问题（若一个问题的每个实例只有“是”或“否”两种答案，则称该问题称为判定问题[7]），而在实际应用中大量出现的优化问题可以与判定问题进行等价转化。问题复杂性的形式化定义可用图灵（Turing Machine）计算模型给出[25]。

定义 2.1： P问题。若一个问题存在解它的多项式时间确定型图灵机（Deterministic Turing Machine，DTM）程序，则称该问题属于P类（Polynomial）。

定义 2.2： NP问题。若一个问题存在解它的多项式时间非确定型图灵机（Nondeterministic Turing Machine，NTM）程序，则称该问题属于NP类（Nondeterministic Polynomial）。

Cook在1971年给出并证明了有一类问题具有下述性质:（1）这类问题中任何一个问题至今未找出多项式时间算法;（2）如果这类问题中的一个问题存在有多项式时间算法，那么这类问题都有多项式时间的算法。同时满足上述两个条件的问题就被称为NP完全问题[7,24]。那么，满足NP完全问题的第（2）个条件，但不一定满足第（1）个条件的问题，就被称为NP难问题，记为NP-Hard。从这个定义可以看出来，NP难问题比NP完全问题的范围要更广，有可能比NP完全问题更难于求解。

2.3.2 组合优化问题与最优解

组合优化问题是一类重要的最优化问题，存在于工作生活的各个领域中。各种基础学科及工程领域都存在组合优化问题，例如，调度问题（Scheduling Problem）、图着色问题（Graph Coloring Problem）、旅行商问题（Traveling Salesman Problem）、合取范式可满足性问题（Satisfiability Problem）、0-1背包问题（Knapsack Problem）、装箱问题（Bin Packing Problem）、聚类问题（Clustering Problem）等等。研究认为，组合优化问题是研究某些具有约束条件的问题的最优解，其目标是从组合问题的可行解集中求出最优解。下面是组合优化问题的形式化定义[26]。

定义 2.3： 组合优化问题。令(S, f)是最优化问题的一个实例，若S为离散的有限点集合，则称该问题为组合优化问题。S是可行解的集合，f（称为目标函数或评价函数）是一个映射，定义为

$$f : S \rightarrow R \tag{式 2.4}$$

求最小值的问题称为最小化问题，记为

$$min\ f(i), i \in S \tag{式 2.5}$$

求最大值的问题称为最大化问题,记为

$$max\ f(i), i \in S \tag{式 2.6}$$

组合优化问题的研究目标是从组合问题的可行解集中找出最优解，然而在求解过程中发现，存在两种不同类型的最优解，一种是局部最优解，一种是全局最优解，它们之间存在一定关系但又有很大矛盾，局部最优解往往掩盖了全局最优解的存在，

全局最优解又是众多局部最优解中的一个。优化算法就是要从众多的局部最优解中发现全局最优解。为了准确描述最优解，给出相关定义如下[27]：

定义 2.4： 邻域（Neighborhood）。邻域函数N是一个映射，$N:S\to 2^S$，它将S中的每个元素s映射到S的一个子集$N(s)\subset S$ 上。若$s'\in N(s)$，则称s'是s的一个邻域解。

定义 2.5： 局部最优解（Local Optimum）。对于某个给定的邻域函数N，若解$s\in S$在f上的取值要优于其它所有的邻域解，则称s是局部最优解。对于最小化问题，$\forall s'\in N(s)$有$f(s)\leqslant f(s')$，则s是局部最优解。

定义 2.6： 全局最优解（Global Optimum）。若解$s^*\in S$在f上的取值优于S中任何其它元素在f上的取值，则称s^*是全局最优解。对于最小化问题，$\forall s\in S$有$f(s^*)\leqslant f(s)$，则s^*是全局最优解。

2.4 启发式算法基础

目前，用来求解NP难问题的算法主要有：精确算法、近似算法和启发式算法。求解NP难问题的精确算法的时间复杂度都是指数型的，对于大规模的问题实例，计算时间无法满足实际需求。近似算法找到的解能通过理论证明其优度与全局最优解的优度相差在某个范围之内，但是在实际应用中，近似算法找到的解的优度往往不高，与实际需求相差较远。与精确算法和近似算法不同的是，启发式算法可以用来处理大规模的问题实例，并可能在一个合理的时间范围内给出一个比较满意的解。在过去的二三十年里，由于启发式算法在解决大规模复杂问题时所表现出来的有效性和高效性，使得启发式算法在各个领域已经受到越来越多的关注。启发式算法经过多年的发展，逐步成为求解NP难问题的利器。

Heuristic的本意是研究与发现，这个词来自希腊语文字heuriskein——去发现。启发式算法是受大自然的运行规律或者面向具体问题的经验、规则启发而获得的方法，它能够在可接受的计算花费内找到满足实际需求的解，但不一定保证解的最优性，也无法证明和最优解的近似程度。图2-1（该图来源于文献[27]第1章）给出了一些经典的启发式算法的发展图谱。

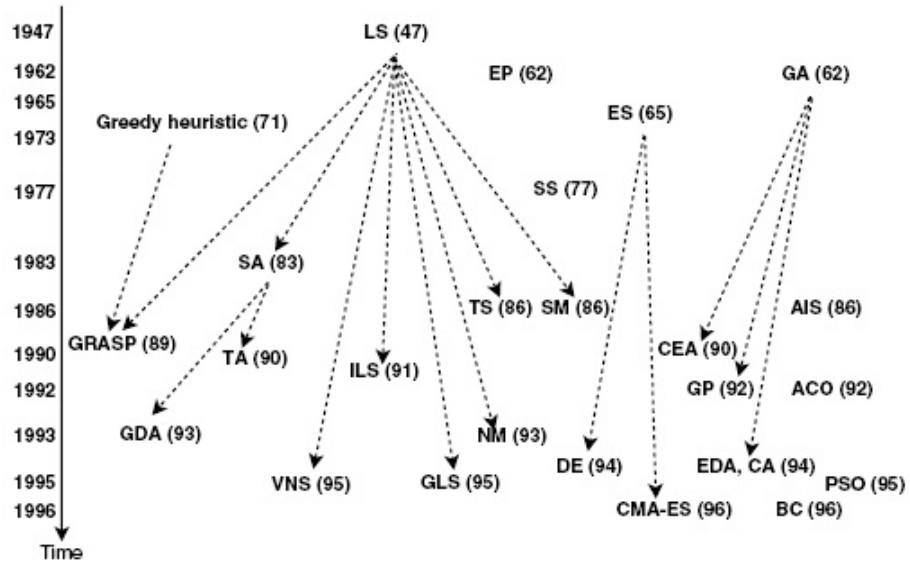

图 2-1 启发式算法图谱（该图来源于文献[27]第1章）

20世纪40年代末期，Polya提出了一些能够解决实际问题的快速有效的启发式算法[28]，并被逐步用于解决各类问题，由此开始，启发式算法进入了快速发展的轨道。单纯形法（Simplex Method）是Dantzig在1947年为求解线性规划问题提出来的，它可以被看做是一个局部搜索算法[29]。Edmonds在1971年首次提出解决组合优化问题的贪心启发式算法（Greedy Heuristic）[30]。Holland通过模拟自然界生物进化的规律提出了遗传算法（Genetic Algorithm，GA ）[31,32]。进入八十年代后，模拟退火算法（Simulated Annealing，SA）[33,34]，禁忌搜索（Tabu Search，TS）[35,36]等启发式算法相继出现。九十年代，迭代局部搜索（Iterated Local Search，ILS）[37], 蚁群算法（Ant Colony Optimization，ACO）[38,39]，变邻域搜索算法（Variable Neighborhood Search，VNS）[40]等如雨后春笋般的相继兴起，掀起了研究启发式算法的高潮，启发式算法已经成为解决组合优化问题的一种重要的手段。

图2-1中列出的其它的启发式优化算法包括：AIS （Artificial Immune Systems）、BC（Bee Colony）、CA（Cultural Algorithms）、CEA （Coevolutionary Algorithms）、CMA-ES （Covariance Matrix Adaptation Evolution Strategy）、DE（Differential Evolution）、EDA （Estimation of Distribution Algorithms）、EP（Evolutionary Programming）、ES（Evolution Strategies）、GDA（Great Deluge）、GLS （Guided Local Search）、GP（Genetic Programming）、GRASP（Greedy Adaptive Search Proce-

dure）、NM（Noisy Method）、PSO（Particle Swarm Optimization）、SM（Smoothing Method）、SS（Scatter Search）和TA（Threshold Accepting）。

本节将介绍几种具有代表性的启发式算法。首先给出局部搜索算法的概念，并介绍三种经典的启发式算法：变邻域搜索算法、禁忌搜索算法、蚁群算法。其中，变邻域搜索算法、禁忌搜索算法是基于单个解的启发式算法，蚁群算法是基于群体解的启发式算法。

2.4.1 局部搜索算法概念

局部搜索算法的基本规则是在邻近的解中迭代，使目标函数逐步优化，直到不能再优化为止。局部搜索每一步都对当前解做一定的改动，使得搜索从当前解到达一个新的邻域解，通过不停的探测逐步扩大到解空间的多个区域。在整个搜索过程中，采用“优胜劣汰”等策略将搜索的目标引向最优解的方向，整个局部搜索过程在达到某个停止标准前不断的运行。局部搜索的思想很符合人的某些思维方式，非常的简单自然，在这种思想的启发下产生了多种目前先进的启发式算法，包括：变邻域搜索算法、禁忌搜索算法、迭代局部搜索算法、单调下降局部搜索算法、模拟退火算法等。

设计局部搜索算法需要解决三个问题：邻域结构的设计；邻域解评估策略；搜索的集中性和多样性的平衡。下面分别对这三个问题进行介绍：

（1）邻域结构的设计。邻域结构是局部搜索算法的基础，邻域结构的设计确定了局部搜索算法能够搜索到的解空间的大小，直接影响算法是否能找到高质量的解。合适的邻域结构是设计高效的局部搜索算法的起点，在实际应用中设计具体问题的邻域结构时，基本思路是让搜索算法能在短时间内找到高质量的解。

（2）邻域解的评估。局部搜索的过程就是不断从当前解s出发找到邻域$N(s)$中的某个邻域解s'。一般要求，s'的优度要比s好，以最小化问题为例来说，即$f(s') < f(s)$。有时不仅只要求s'的优度比s好，还需要s'是邻域$N(s)$中优度最好的解。由此可以看出，局部搜索的过程是一个逐步往优度好的解移动的过程。为了实现上述有偏向性（不断向好解移动）的搜索，在局部搜索的每一步都需要对$N(s)$中的一个或多个邻域解的优度进行评估。在实际计算过程中这种评估占去搜索算法的大部分时间，因此，设计高效的邻域解评估策略是优秀的局部搜索算法的基础。

（3）集中性和多样性的平衡。启发式算法中有两个吸引研究人员不断探索的概念：搜索的集中性（Intensification）和多样性（Diversification）[41]，这两个概念至今也没有严格的定义。通常，集中性表现的是算法能细致深入的搜索解空间中某部分区域的能力，它使得算法能从当前解出发逐步搜寻到优度越来越好的解，反应的是算法搜索的深度；多样性表现的是算法能不断搜索解空间中多个有前途区域的能力，可以避免算法只在局部区域进行搜索，保证搜索能在尽可能多的区域进行，它反应的是搜索的广度。设计启发式算法时需要同时考虑集中性和多样性这两个方面。然而，过度强调集中性会使算法陷入某个局部区域出不来，过度强调多样性会使算法失去搜索方向，无法找到高质量的解。因此，设计高效的启发式算法必须在集中性和多样性之间找到平衡。

2.4.2 变邻域搜索算法（Variable Neighborhood Search）

变邻域搜索算法是一种非常高效的处理组合优化问题的通用启发式算法，由Mladenović[40]首次在20世纪90年代提出。变邻域搜索算法最显著的特征是，在局部搜索过程中可以不断变换邻域结构，这样即避免陷入局部最优，又可以扩展其搜索范围。经过几十年的发展，变邻域搜索算法已被成功的应用到旅行商问题、车辆路径问题、调度问题和图着色等问题中。变邻域搜索算法在处理组合优化问题和实际应用中的一些规则及其改进版本在文献[42–45] 中做了进一步的介绍。

设计变邻域搜索算法时，需要预先确定以下几主要个问题：（1）采用哪几种邻域结构以及一共采用几种邻域结构；（2）邻域结构的使用顺序；（3）变换邻域结构的策略；（4）局部搜索过程；（5）停止标准。只有解决好上述五个问题，变邻域搜索才能更好的发挥其强大的功能。

在变邻域搜索算法中，用$N_k(k=1,...,k_{max})$表示事先选择好的有限的邻域结构的集合，$N_k(x)$表示在第k种邻域结构下x 的邻域解的集合（大多数局部搜索算法中只用到一种邻域结构，即$k=1$）。基本的变邻域搜索算法的步骤描述如下：

Step 1: 选择一个邻域结构的集合$N_k(k=1,...,k_{max})$，生成一个初始解x，令$k=1$，确定算法的停止标准；

Step 2（shaking）：随机选择一个邻域解$x'\in N_k(x)$；

Step 3（local search）：用局部搜索方法从解x'出发找到局部最优解x''；

Step 4（move or not）：如果x''要优于x，则令$x = x''$并且$k = 1$，否则$k = k + 1$（或者如果$k = k_{max}$, 则令$k = 1$）；返回step 2。

从上面的步骤可以看出，变邻域搜索算法主要由两个搜索过程组成：一个是随机过程，即在step 2中在初始解x的邻域中随机找到一个邻域解x'的过程；另一个则是确定过程，即在用局部搜索方法找到x'的局部最优解x''的过程。在基本的变邻域搜索算法的基础上，又产生了很多扩展版本，并已成功的用于解决各种组合优化问题。包括变邻域下降搜索（Variable Neighborhood Descent，VND）和简化变邻域搜索（Reduced Variable Neighborhood Search，RVNS）等。

变邻域搜索算法以及它的扩展都相对比较简单，且很少或几乎不需要设定参数。因此，除了可以提供更好的解之外，变邻域搜索算法实现起来也更加简单，却仍不失其有效性。

2.4.3 禁忌搜索算法（Tabu Search）

禁忌搜索算法最早由Glover[35]和Hansen[36]分别在不同的文献中提出，是对人工智能过程的一种模拟，也是非常著名的局部搜索算法之一。它的主要思想可以简单概括为“不走回头路”，即已经走过的路，哪怕再平坦，前面的路哪怕再崎岖，也轻易不往回走。禁忌搜索算法如今已被广泛应用于求解各个领域的优化问题，并取得了非常好的效果。

在禁忌搜索算法中，为了实现禁忌的思想，主要采用了下面几种机制：

（1）禁忌表（Tabu List）。禁忌表里主要是存放刚刚做过的一些动作和找到的一些解。规定在一定的禁忌期限（Tabu tenure）内，即一定的迭代次数内，这些动作或者解不会被再次选择。当过了这个动作或解的禁忌期限后，该动作或解则被从禁忌表中取出来，这也是禁忌表的一个更新过程。禁忌表的长度（禁忌多少个解）以及禁忌期限的长短（每个解在多少迭代次数内是被禁忌的）都是决定搜索性能的关键参数。

（2）解禁规则（Aspiration criterion）。禁忌表中存放的是最近做过的一些动作和找到的一些解，然而，即使这些动作或解的优度较高，同样还是会被禁忌掉。因此，为了让这些被禁忌的优良解有被选择的机会，则需要解禁规则来完成。

（3）接受非改进邻域解。在禁忌搜索算法中，当搜索达到某个局部最优值的时

候，此时所有邻域解的优度都比当前解的优度要差，但是搜索仍然从那些未被禁忌的解集中挑选一个优度最好的邻域解来替代当前解，并继续迭代过程。

（4）停止标准。禁忌搜索算法中当达到局部最优解时，它通过接受非改进邻域解来跳出局部最优陷阱，到达某个更有前途的区域。常用的停止标准有如下两种：预先设定一个最大迭代次数，当禁忌搜索过程经历的迭代次数超过设定值，则搜索终止；找到当前最好解之后，在预先设定迭代次数内，当前最好解仍然没有被更新，则搜索终止。

禁忌搜索算法的主要步骤如下所示：

Step 1：生成一个初始解s_0，令$s = s_0$，并初始化禁忌表；

Step 2：找出s的邻域中未被禁忌或符合解禁规则中最好的一个解s'，令$s = s'$，更新禁忌表；

Step 3：如未满足停止标准则重复step 2。

禁忌搜索已经被成功的用于求解许多困难的组合优化和全局优化问题。它最显著的特征是能够通过选择非改进邻域解来跳出局部最优解陷阱，并通过使用禁忌表将搜索范围扩大。经过多年的发展，禁忌搜索已经成为解决组合优化问题的有力工具，它经常能在短时间内为复杂问题找到高质量的解。在调度、图着色、最大团等问题上，禁忌搜索算法是当前最好的局部搜索算法之一。

2.4.4 蚁群算法（Ant Colony Optimization）

蚁群算法最早由Dorigo和同事们[38,39]受到蚂蚁群体行动的启发，在处理复杂的组合优化问题时，如旅行商问题和二次分配问题（Quadratic Assignment Problem）时首次提出来的。之后又经过一系列深入的研究，为蚁群算法确定了统一的框架和抽象而规范的算法描述。经过多年的发展，蚁群算法已经被用于求解各种组合优化问题，例如车辆路径问题、图着色问题、调度问题、通信网络中的路由问题（Routing in Communications Networks）等等。

蚁群算法是通过观察蚁群活动得到启发而产生的。蚂蚁是一种群居昆虫，它们的群体协作能力非常强，特别是在觅食的时候，它们总能从自己的巢穴到食物所在地之间找到一条相对较短的路径。蚂蚁群体的这种寻路技能吸引了很多研究人员的注意，那么，蚂蚁是如何找到这条较短的路径的呢？经过大量的研究发现，蚂蚁个

体之间通过一种称为信息素（pheromone）的物质进行信息传递。也就是说，在某一条路上通过的蚂蚁越多，留下的信息素也越多，那么后面的蚂蚁可以感知到信息素的存在及其强度，以此来指导自己的运动方向，即选择信息素强度高的那一条路。

以寻找最短路径为例，对蚁群算法的简单描述如下：

Step 1：一群蚂蚁开始觅食，每只蚂蚁随机选择一条路往食物的方向前进，此时路上没有信息素；

Step 2：蚂蚁们在经过的路上都会留下信息素，回到蚁巢的蚂蚁再次出发，这次它们会按照路上信息素浓度的强弱选择觅食路线，一般是选择信息素强的路径；

Step 3：一条路径上走过的蚂蚁越多，信息素浓度就会增强（reinforcement），相反信息素就会因挥发（evaporation）而减弱；

Step 4：若食物没有搬完，蚂蚁重复steps 2-4，直到食物搬完为止。这时有一条路径上的信息素浓度比其它路径上的信息素浓度都强，那么这条路径就是最短路径。

由以上描述可以看出蚁群算法的重点是蚂蚁选择路径的规则和信息素浓度的变化。蚂蚁选择路径的规则如下：

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{\Gamma_{ij}^{a}(t)\eta_{ij}^{\beta}(t)}{\sum\limits_{s\subset allowed_k}\Gamma_{ij}^{a}(t)\eta_{ij}^{\beta}(t)}, s\subset allowed_k\\ 0, otherwise\end{cases} \tag{式 2.7}$$

${P_{ij}}^{k}(t)$表示t时刻蚂蚁$k(k=1,2,...,m)$由i点向j点移动的概率；$\Gamma_{ij}(t)$表示时刻t在边e_{ij}上残留的信息素，在初始时刻$\Gamma_{ij}(0)=c$（c常取为0）；α为信息素的重要程度；$\eta_{ij}(t)$表示从i点向j点移动的启发信息，$\eta_{ij}(t)=1/d_{ij}$，d_{ij}表示i点与j点之间的距离$(i,j=1,2,...,n)$；β为启发信息的重要程度；$allowed_k$表示蚂蚁k下一步允许选择的点的集合。经过n个时刻，边e_{ij}上的信息素可根据公式（2.8）进行调整。

$$\begin{cases}\Gamma_{ij}(t+n)=(1-\rho)\Gamma_{ij}(t)+\Delta\Gamma_{ij}(t)\\ \Delta\Gamma_{ij}(t)=\sum\limits_{k=1}^{n}\Delta\Gamma_{ij}^{k}(t)\end{cases} \tag{式 2.8}$$

$\Gamma_{ij}(t+n)$表示在$t+n$时刻边e_{ij}上的信息素；ρ表示信息素挥发系数，$\rho \in (0,1)$；$\Delta\Gamma_{ij}(t)$表示在边e_{ij}上的信息素增量；$\Delta{\Gamma_{ij}}^{k}(t)$表示第$k$只蚂蚁留在边$e_{ij}$上的信息素。其中$\Delta{\Gamma_{ij}}^{k}(t)$可按公式（2.9）进行计算。

$$\Delta\Gamma_{ij}^{k}(t)=\begin{cases}\frac{Q}{L_k}, ij \in l_k \\ 0, otherwise\end{cases} \quad \text{(式 2.9)}$$

Q为常数，表示信息强度；L_k为第k只蚂蚁走过的路径长度；l_k为第k只蚂蚁走过的路径。

由以上看出蚁群算法中包含很多的参数，如α 、β、ρ、Q、k等，这些参数的组合设置对蚁群算法的优劣起着至关重要的作用，因此，确定这些参数的取值成为蚁群算法研究的重点。

2.5 单机调度问题的研究方法及现状

单机调度问题成为研究领域的热点已经有几十年了，也有很多优秀的解决单机调度问题的方法被提出来。但是，由于单机调度问题的多样性和复杂性，导致并没有一种普遍适用的方法来求解单机调度问题。常用的解决单机调度问题的方法主要分为以下几类：精确算法、优先分配规则和启发式算法。

一、精确算法

分支定界（Branch and Bound）和动态规划（Dynamic Programming）是两种常用来解决单机调度问题的精确算法。

Abdul-Razaq等人[5]在求解不带准备时间的$1||\sum w_jT_j$ 问题时详细对比了两种动态规划方法和四种分支定界方法。Potts和Van Wassenhove用分支定界法[46]求解了不带准备时间的$1||\sum w_jT_j$ 问题，同时他们还用动态规划法[47]求解了$1||\sum T_j$ 问题。Ragatz[48]、Luo[49]和Bigras等人[50]分别用分支定界法来解决$1|s_{ij}|\sum T_j$ 问题。Rabadi等人[51]提出了一种分支定界法来求解具有相同预期完成时间的$1|s_{ij},d_j=d|\sum(E_j+T_j)$ 问题，同时，Sourd[52]也提出了一种分支定界法来求解具有不相同预期完成时间的$1|s_{ij}|\sum(E_j+T_j)$ 问题。Luo 和Chu[53]则采用分支定界法来解决$1|s_{ij}|T_{max}$ 问题。其中，Bigras等人[50]在用分支定界方法求解不带权$1|s_{ij}|\sum T_j$ 问

题算例的时候，成功处理了工件数达到45的算例，但是，对于最困难的算例，算出最优解需要花费超过7天的时间。

最近，由Tanaka和Araki提出的一种精确算法[54]找到了由Cicirello生成的$1|s_{ij}|\sum w_jT_j$问题的全部120个算例的最优解；对于由Rubin和Ragatz[55]以及Gagné等人[56]生成的$1|s_{ij}|\sum T_j$问题的64个公共算例，精确算法也找到了62个算例的最优解和另外2个算例的当前最好解。该精确算法是作者对之前处理不带准备时间的单机调度问题研究[57,58]的一个延续和扩展，之前提出的算法是基于动态规划逐次升华（Successive Sublimation Dynamic Programming）[59,60]技术的。在文献[57,58]中提出的算法可以找到工件数达到300的$1||\sum w_jT_j$问题算例的最优解。然而，它的算法框架对于$1|s_{ij}|\sum w_jT_j$问题却不那么有效，因为跟$1||\Sigma w_jT_j$问题相比，$1|s_{ij}|\sum w_jT_j$问题多了序列相关的准备时间，导致很难得到紧密下界（tight lower bound）。因此，Tanaka和Araki通过在原有算法框架的基础上加入分支策略（branching scheme），提出了一种有效解决$1|s_{ij}|\sum w_jT_j$问题和$1|s_{ij}|\sum T_j$问题的精确算法。虽然对于某些算例计算时间比较长（超过30天），但是，基本能算出所有算例的最优解。

动态规划或者分枝定界等精确算法只适用于规模不大和问题性质不复杂的优化问题。而实际生产环境具有很多不确定、动态等复杂因素，存在建模不确定性和求解空间太大等问题，造成计算困难，所以很难用精确算法去处理实际调度问题。

二、优先分配规则

优先分配规则属于一种构造算法。对于单机调度问题，它根据一定的分配或调度规则，每一步往序列里只添加一个工件，直到完成整个加工序列。优先分配规则的优点是简单直观、花费的计算代价小、易于实现；缺点是在处理复杂的调度问题时，难以找到全局最优解，或者找到的解质量较差。在求解调度问题时研究者们根据其不同的特点提出了多种优先分配规则，下面介绍其中的几种。

（1）WSPT规则[61]：加权最短处理时间（Weighted Shortest Processing Time）优先规则。对于每个工件，计算w_j/p_j的比率，然后，按照w_j/p_j值的降序对工件进行排列。该规则求解$1||\sum w_jC_j$问题的效果较好。

（2）EDD规则：最早完工（Earliest Due Date）优先规则。按照预期完工时间d_j的升序对工件进行排列。该规则求解$1||L_{max}$问题效果较好。

（3）ATC规则[62]：明显延迟成本（Apparent Tardiness Cost）优先规则。ATC规则是由Vepsalainen和Morton在求解不带准备时间1||$\sum w_jT_j$问题时提出来的，并且他们同时还测试了其它几种优先分配规则的效率。

（4）ATCS规则[63]：带准备时间的明显延迟成本（Apparent Tardiness Cost with Setups）优先规则。ATCS规则是Lee等人在ATC 规则的基础上，将其扩展到求解带准备时间的1|s_{ij}|$\sum w_jT_j$ 问题时所提出的一种新的优先规则。

ATCS规则分为两个阶段。第一阶段是对所需求解问题的所有算例进行统计分析。这个统计分析的结果就是根据算例的具体属性特征得出3个评估参数，包括due date tightness（记为τ ）、due date range（记为R ）和setup time severity factor（记为η ），这个阶段可以被看成是一个预处理阶段。在第二个阶段里，则是利用第一阶段得到的3个参数计算出k_1和k_2的值，然后根据k_1和k_2的值计算出算例中每个工件的优先指数（Priority Index），最后根据优先指数的大小来确定工件的排列顺序。

第一阶段的3个评估参数τ 、R 和η 的值可以分别通过下面的公式计算得出：

$$\tau = 1 - \frac{\overline{d}}{C_{max}} \tag{式 2.10}$$

$$R = \frac{d_{max} - d_{min}}{C_{max}} \tag{式 2.11}$$

$$\eta = \frac{\overline{s}}{\overline{p}} \tag{式 2.12}$$

在上述公式中，$\overline{d}$表示某算例中所有工件预期完成时间的平均值，d_{max}和d_{min}表示所有工件预期完成时间的最大值和最小值；$\overline{s}$和$\overline{p}$分别表示所有工件准备时间的平均值和处理时间的平均值。算例中每个工件的预期完成时间、准备时间和处理时间都会事先给出，因此，上述几项参数值可以很容易计算出来。然而，C_{max} 表示最大完工时间，由于每两个工件之间的准备时间各不相同，对于不同的加工序列，就会有不同的C_{max} 值，因此，要事先计算出C_{max} 的值是非常困难的。那么，可根据下面的公式估算出C_{max} 的值：

$$C_{max} = n(\overline{p} + \beta\overline{s}) \tag{式 2.13}$$

其中n表示算例所包含的工件数；β 是一个系数，它的值会受工件数目和准备时间变化的影响[63]。k_1 和k_2 的值可以依据下面的公式计分别计算出：

$$k_1 = \begin{cases} 4.5 + R, R \leqslant 0.5 \\ 6.0 - 2R, R > 0.5 \end{cases} \quad \text{(式 2.14)}$$

$$k_2 = \frac{\tau}{2\sqrt{\eta}} \quad \text{(式 2.15)}$$

最后，按照ATCS规则的要求，每个工件的优先指数可以根据如下公式计算出：

$$I_j(t,i) = \frac{w_j}{p_j} exp\Big[-\frac{max(d_j - p_j - t, 0)}{k_1 \overline{p}}\Big] exp\Big[-\frac{s_{ij}}{k_2 \overline{s}}\Big] \quad \text{(式 2.16)}$$

其中，t表示当前时间，i表示当前已经安排到加工序列中的工件，j 表示接下来准备安排到加工序列中去的工件。即当加工序列中第i个工件已经安排好后，计算当前所有还未安排的工件的优先指数，然后，优先指数值最大的工件则入选，被安排到工件i后面。

三、启发式算法

Potts和van Wassenhove[64]针对不带准备时间$1||\sum w_j T_j$ 问题对比了几种启发式算法的优劣，从试验结果发现对于该问题成对交换法（Pairwise Interchange）是一个不错的方法。此外，Holsenback等人[65] 在解决不带准备时间$1||\sum w_j T_j$问题时提出了一种改进的启发式算法，并且该算法能处理规模达到50个工件的问题。Feo 等人[66]对带准备成本以及线性延期惩罚的单机调度问题提出了一种贪婪自适应搜索算法（GRASP），同时，Gupta和Smith[67]也提出了一种贪婪自适应搜索算法（GRASP）去解决$1|s_{ij}|\sum T_j$ 问题。关于带权和不带权的延迟调度问题的相关综述可以参考文献[2]中的内容。

Rubin和Ragatz[55]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[68]提出一种模拟退火算法（SA）来解决$1|s_{ij}|\sum T_j$问题。Armentano和Mazzini[69]也提出了一种用遗传算法（GA）解决$1|s_{ij}|\sum T_j$ 问题的方法，而França等人[70]则提出了一种混合进化算法（Memetic）算法解决$1|s_{ij}|\sum T_j$问题。Tan等人[71]在解决$1|s_{ij}|\sum T_j$ 问题的时候，对比了四种方法的优劣，包括分支定界、遗传搜索（genetic search）、随机启动两两交换（random-start pair-wise interchange）

和模拟退火（SA）。Sun等人[72] 在处理1|s_{ij}|$\sum w_j T_j^2$问题的时候提出了一种基于拉格朗日松弛（Lagrangian relaxation-based）的方法，并且将该方法与EDD以及ATCS优先规则、四种不同的基于交换的局部搜索、禁忌搜索（TS）和模拟退火（SA）进行了对比。2002年，Gagné 等人[56]提出了一种蚁群算法（ACO）来求解1|s_{ij}|$\sum T_j$问题，同样，Liao 和Juan[73]也提出了一种蚁群算法求解该问题。2005 年，Gagné 等人[74]又提出了一种混合了禁忌搜索和变邻域搜索的算法（tabu-VNS）求解多目标的调度问题。Ying等人[75]提出了一种迭代贪婪搜索算法（IG）来求解1|s_{ij}|$\sum T_j$问题。

对于既带准备时间又带权值的1|s_{ij}|$\sum w_j T_j$ 问题，学者们也投入了大量的关注。Cicirello 和Smith[76]为1|s_{ij}|$\sum w_j T_j$ 问题生成了120个算例，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工作效率。在文献[73,77]中，蚁群算法（ACO）被应用于求解1|s_{ij}|$\sum w_j T_j$ 问题。Lin和Ying[78]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1|s_{ij}|$\sum w_j T_j$ 问题时的表现。在文献[79]里，Cicirello通过在遗传算法（GA）中引入一种新提出的非包装秩序交叉算子（non-wrapping order crossover）改进了文献[76]中的结果。Valente和Alves[80]提出了一种带可变束和过滤宽度的束搜索算法（Beam Search）。另外，在文献[81]中提出了一种粒子群算法（DPSO）用来求解1|s_{ij}|$\sum w_j T_j$ 问题，并给出了当时该问题算例的当前最优解。近年来Tasgetiren等人[82]提出的离散差分进化算法（Discrete Differential Evolution，DDE）以及Kirlik和Oguz[83]提出的变邻域搜索算法（GVNS）大幅度的改进了1|s_{ij}|$\sum w_j T_j$问题算例的当前最优解。

2.6 本章小节

本章首先阐述了生产调度问题的相关概念，并对单机调度问题进行了形式化描述；对优化问题中涉及到的一些概念进行了定义；详述了几种经典的启发式算法的设计与分析方法，并对单机调度问题的研究现状进行了较详细的介绍。

第一节介绍了生产调度问题的相关概念。

第二节对单机调度问题进行了形式化描述，并对本文研究的两种单机调度问题进行了分析介绍。

第三节用数学的方法对优化问题中涉及到的一些概念进行了定义（包括P、NP、NP完全、NP难、邻域、局部最优解、全局最优解），为后续论述提供相关的理论支持。

第四节首先绍了启发式算法的发展历程，给出了一个发展图谱；然后详细讨论了局部搜索算法的相关概念及其设计准则。接着对变邻域搜索算法、蚁群算法、禁忌搜索算法的理论背景和设计方案进行了详细的描述。

第五节对单机调度问题的研究方法及现状进行了较详细的介绍。

3 迭代局部搜索算法求解单机调度问题

迭代局部搜索算法（Iterated Local Search，ILS）是一种简单、易实现、鲁棒性强且高效的启发式算法[84,85]，是对局部搜索算法的一种改进。它在当局部搜索陷入解空间的局部最优陷阱时，使用扰动算子（Perturbation Operator）对局部最优解进行扰动，使得搜索跳出局部最优陷阱，从而到达某个可能有更好前景的区域。本章提出了一种新的邻域结构以及一种快速增量评估技术，结合适当的扰动机制，设计出高效的迭代局部搜索算法BILS来求解单机调度问题。通过实验证明，BILS算法不论是和当前最好的启发式算法相比，还是和Tanaka和Araki的精确算法相比，都具有较大的优势。

3.1 迭代局部搜索算法的基本理论

局部搜索算法是一种常用的解决组合优化问题的方法，在问题的求解过程中经常对局部搜索算法迭代使用，即完成一次局部搜索后，随机选择一个初始解作为下一次局部搜索的开始，一般来说，随机重启对局部搜索在某些情况下也不失为一种有效的策略。但是，随着问题规模的变大，随机重启的效果就变得越来越差。迭代局部搜索算法在一定程度上改变了上述状况，即一次局部搜索过程结束后，不再随机选择初始解进行下一次局部搜索，而是在当前局部最优解的基础上，对其进行扰动（对局部最优解做适当调整），以扰动后的解为初始解进行下一次局部搜索。这样得出来的结果，要优于随机选择一个初始解再进行局部搜索。迭代局部搜索算法主要由局部搜索过程、扰动、接受标准三部分组成。

迭代局部搜索算法从一个初始解$s \in S$出发，然后不断的在当前解s 的邻域$N(s)$内搜索比s 更好的解s'，即局部最优解；对局部最优解s'进行扰动得到s^*，继续对s^*进行局部搜索得到另一个局部最优解$s^{*\prime}$；用接收标准来判断两个局部最优解s'和$s^{*\prime}$哪一个被接受作为下一轮的初始局部最优解s'，迭代上述过程直到满足停止标准。迭代局部搜索算法的基本框架如算法1所示。

Algorithm 1 迭代局部搜索算法

1: $s \leftarrow$ initial solution
2: $s' \leftarrow$ local search (s)
3: **repeat**
4: $s^* \leftarrow$ perturbation (s')
5: $s^{*\prime} \leftarrow$ local search (s^*)
6: $s' \leftarrow$ acceptance criterion (s', $s^{*\prime}$)
7: **until** stop condition met

3.1.1 初始解的产生和停止标准

一个好的初始解，可以让迭代局部搜索算法在尽可能短的时间内找到高质量的解。可见，初始解的产生，对迭代局部搜索算法是非常重要的。

初始解的产生通常有两种方法：（1）随机产生初始解；（2）用启发式贪心算法来生成初始解。一般来说，用哪种方法产生初始解更好，并没有明确的标准。文献[86]中，作者采用迭代局部搜索算法来求解流水车间调度问题，在局部搜索和扰动机制相同的情形下，对比了随机产生初始解和用NEH算法（NEH算法是流水车间调度问题最好的构造算法之一）[87]产生初始解的优劣。对于两组算例，进行了初始解对比测试，发现，其中一组算例，在短时间内，用NEH算法产生初始解是优于随机产生初始解的，但是，随着时间增加，两种初始解的优劣并不明显；然而对于另一组算例，随机产生初始解在很长一段时间内都要优于用NEH 算法产生初始解。因此，如果算法运行的时间比较长，初始解的生成方法也就不那么重要了[85]。在实践中，应该根据具体的问题规模选择相应的方法来产生初始解。

迭代局部搜索算法以及其它启发式算法的停止标准大体上是相同的，一般包括以下两种：

（1）静态标准。在静态标准中，搜索过程的结束往往取决于一个预先设置的值，超过该预设值，则算法终止。例如，这个预先设置好的值可以是固定的迭代次数或是对CPU的资源限制等。

（2）自适应标准。在自适应标准中，当前最好解或者符合要求的解在连续若干次迭代后没有改进，则算法终止。

3.1.2 局部搜索过程

设计局部搜索过程需要解决三方面的问题：（1）合理的邻域结构设计；（2）高效的邻域解评估策略；（3）搜索的多样性和集中性的平衡。

多种局部搜索算法都可以作为迭代局部搜索算法中的局部搜索过程，例如禁忌搜索、模拟退火等。一般来说，局部搜索效率越高寻找最优解的能力越强，相应的迭代局部搜索算法也就越好。

对于局部搜索算法，构造一个好的邻域结构是它的核心任务之一。局部搜索算法从初始解出发，通过将邻域中的局部最优解与当前最好解的优劣进行比较来确定如何动作。邻域结构设计的好坏是影响局部搜索算法效率的主要因素之一，邻域过大，搜索花费（存储和计算时间）就会增大；邻域过小，搜索容易陷入局部最优。因此，研究人员设计出各种启发式策略来定义邻域结构，期望在规模尽可能小的邻域结构中包含尽可能多的有希望的解。

邻域结构定义好之后，如何选择邻域解就成了关键。一般来说，有以下三种选择方法：

（1）选择改进最大的解。从邻域解的集合中选出相对当前解改进最大的解。此方法的优点是，选出的解相对当前解改进最大；缺点是需要搜索整个邻域空间，当邻域较大或评估邻域解优度的代价较大时，所需要的花费也相对较大。

（2）选择遇到的第一个有改进的解。依次计算邻域中解的优度，选择第一个有改进的解。此方法的优点是不需要搜索当前解的全部邻域；缺点是找到的解不一定是所有邻域解中最好的。

（3）随机选择一个改进的解。从所有有改进的邻域解中随机选择一个解。此方法的优点是增强了多样性；缺点是耗时较长。

局部搜索从初始解开始，不断往有更好解的区域探索，这种特点就像植物有向光性，朝着阳光生长。局部搜索过程中邻域解的选择机制使得它有了这种寻找最好解的偏向性，为了实现上述有偏向性的搜索，在局部搜索的每一步都需要对$N(s)$中的一个或多个解的优度进行评估。在实际计算过程中这种评估占去搜索算法的大部分时间。因此，提高邻域解评估的效率，就能提高局部搜索的效率。目前，评估邻域解的方法主要有以下三种：

（1）直接计算邻域解的目标函数值。这种方法直接计算$N(s)$ 中邻域解s' 的目标函数值$f(s')$，优点在于计算方法简单易实现，缺点是问题规模比较大时计算量也很大。在问题规模较大，且对效率要求较高的情况下，很少使用这一方法。

（2）增量评估。在邻域搜索过程中，如果在当前解s与其邻域解s' 之间存在$\Delta f = f(s') - f(s)$ 的函数关系，那么，在评估邻域解s'时只用计算出Δf 的大小即可，而不用重新计算$f(s')$，这样就节省了大量的计算时间。

（3）近似评估。在求解某些问题时，对邻域解s'进行评价的函数$f(s')$计算量较大，这时考虑使用一个计算量相对较小的函数$f'(s')$来代替$f(s')$ 对邻域解进行评估，这种方法的关键是如何找到一个合适的近似评估函数$f'(s')$。在用局部搜索算法求解工件车间调度问题时，近似评估技术得到了广泛的应用[88,89]。

3.1.3 扰动

局部搜索算法最大的缺点就是容易陷于局部最优，而无法找到更好的解。于是，设计一种有效的机制，使其可以跳出局部最优陷阱是非常好的方法。迭代局部搜索算法采用扰动算子（Perturbation Operator）来对当前局部最优解进行扰动，使之跳出局部最优解陷阱。扰动简言之，就是对当前解按照一定的规则进行改动。那么做多大的改动? 这也是扰动需要解决的问题。

改动大小，就是指扰动的强度（Perturbation Strength）。扰动的强度不能过小，否则很容易再次陷入以往的局部最优陷阱，从而限制了搜索空间的多样性。同样，扰动的强度不能太大，否则就像是又随机生成了一个初始解，破坏了当前解的有效信息，使得找到相对较好解的可能性降低。因此，扰动强度的拿捏，对迭代局部搜索算法的优劣有着很大的影响。

用迭代局部搜索算法求解旅行商问题和流水车间调度问题，合适的扰动强度通常是比较小的，并且和算例的规模无关；然而，在其它一些问题上，例如二次分配问题，比较好的扰动强度却是比较大的，并且，扰动强度的大小因为算例的不同而有所差异[85]。由些可见，扰动强度的确定是跟具体问题相关的。

3.1.4 接受标准

接受标准是用来判断新找到的解是否可以作为当前解继续进行迭代的标准。接

受标准可以被用来很好的平衡搜索的多样性和集中性。

接受标准大体来说有以下几种类型：

（1）只有比当前解好的解才被接受（记为Better）。这种接受标准虽然简单，但是，在很多情况下，这种简单的接受标准也有着很好的效果。以最小化问题为例，该接受标准可以表示为：

$$Better(s', s^{*\prime}) = \begin{cases} s^{*\prime}, f(s^{*\prime}) < f(s') \\ s', otherwise \end{cases} \quad \text{(式 3.1)}$$

（2）随机行走接受（Random Walk，记为RW）。跟第一种不同，不考虑解的目标函数值的优劣，直接接受找到的局部最优解。随机行走接受标准强调解的多样性要多于集中性。

$$RW(s', s^{*\prime}) = s^{*\prime} \quad \text{(式 3.2)}$$

（3）介于第一种方法和第二种方法之间的接受标准，例如，类似模拟退火的接受标准[37,90]。当$s^{*\prime}$优于s'时，接受$s^{*\prime}$；否则，以$exp\{(f(s') - f(s^{*\prime}))/T\}$的概率接受$s^{*\prime}$，其中$T > 0$表示温度，并在运行过程中逐渐减小。可以看出，当$T$值非常小的时候，类似模拟退火的接受标准非常近似于第（1）种接受标准，反之，当T值非常大的时候，则近似于第（2）种接受标准。

3.2 求解单机调度问题的迭代局部搜索算法

随着用启发式算法求解单机调度问题研究的不断发展，研究者们力求在相对较短的时间内找到更好的解。为此，针对单机调度问题提出了各种不同的邻域结构和邻域解的评估策略。本节中，提出了一种新型的邻域结构以及相应的快速评估该邻域结构的技术，结合适当的扰动机制和接受标准，提出了一种高效的迭代局部搜索算法BILS来求解单机调度问题。

3.2.1 搜索空间和目标函数

$1|s_{ij}|\sum w_j T_j$ 问题的优化目标是最小化所有工件的加权延迟之和，即找到一个加工序列π，使它的目标函数值$f(\pi) = \sum_{j=1}^{N} w_{\pi_j} T_{\pi_j}$ 最小。对于工件数

为N的$1|s_{ij}|\sum w_jT_j$ 问题，它的搜索空间是由N个工件组成的所有可能的序列，因此，搜索空间的大小等于工件数N的阶乘，即$N!$ 。

3.2.2 初始解的产生

通常，迭代局部搜索算法的初始解要么随机产生，要么用贪心构造算法来产生。对于$1|s_{ij}|\sum w_jT_j$ 问题，由文献介绍可知，ATCS优先规则[63]是目前最好的构造算法，被广泛的用于各种启发式算法中生成初始解。用$1|s_{ij}|\sum w_jT_j$ 问题的部分算例对BILS算法进行测试，对比了随机生成初始解和用ATCS规则生成初始解的情况。通过测试结果发现，无论是从解的质量还是算法的效率来考虑，BILS算法对这两种生成初始解的方法并不敏感。因此，在BILS算法中采用随机的方法生成初始解。

3.2.3 邻域结构和评估策略的设计

BILS算法中，采用了单调下降局部搜索算法作为局部搜索过程。在每一次搜索过程中，选择邻域解里面最好的那一个解，直到当前的局部最优解不能再被改进为止。

邻域结构定义的好坏直接影响局部搜索算法的寻优能力。本文提出了一种新的块移动（Block Move）的邻域结构$N_{Block\ Move}$，该邻域结构在解决$1|s_{ij}|\sum w_jT_j$ 问题的研究过程中由我们首次提出。为了展示块移动邻域结构$N_{Block\ Move}$的有效性，本文将块移动的邻域结构与之前各种启发式算法在解决$1|s_{ij}|\sum w_jT_j$问题时经常用到的三种邻域结构进行了对比。

一、文献中常用的邻域结构

在文献[83]中介绍了三种邻域结构：插入（insertion）、边插入（edge-insertion）、交换（swap），它们被广泛的应用于解决各种不同的组合优化问题，并且取得了非常好的效果，尤其在处理用序列表示的问题时更为突出。

（1）插入邻域$N_{insertion}$ ，它是一个集合，包括把一个工件插入到当前加工序列的其它位置而产生的所有的解。其动作描述如下：

Step 1：从加工序列π中随机选择一个工件π_i 和一个位置k；

Step 2：将工件π_i 从加工序列π中删除；

Step 3：将工件π_i 插入到加工序列π 的位置k 上。

（2）边插入邻域$N_{edge-insertion}$是把相邻的两个工件（也可叫做边）同时插入到当前加工序列的其它位置而得到的所有解组成的。其动作描述如下：

Step 1：从加工序列π 中随机选择一条边(π_i, π_{i+1})和一个位置k；

Step 2：从加工序列π 中删除边(π_i, π_{i+1})；

Step 3：将边(π_i, π_{i+1}) 插入到加工序列π 的位置k 上。

（3）交换邻域N_{swap} 是交换当前加工序列中的两个工件的位置而产生的所有解的集合。其动作描述如下：

Step 1：从加工序列π 中随机选择两个工件π_i 和π_j；

Step 2：交换π_i 和π_j 在π 中的位置。

可以很容易的看到，插入（insertion）、边插入（edge-insertion）、交换（swap）这三种邻域的大小为$O(N^2)$ 。这些邻域结构在各种启发式算法中都有相当好的表现，但在研究过程中发现这三种邻域结构的搜索空间在某种程度上还是有一定的局限，如果能找到一种搜索空间更大的邻域结构，这样，找到更好解的机率就会大大增加。

二、块移动（Block Move）邻域结构

本文在求解$1|s_{ij}|\sum w_jT_j$问题时，提出了一种新的块移动（Block Move）的邻域结构，记作$N_{Block\ Move}$，它是对当前解做Block Move动作后产生的解的集合。Block Move动作的定义如下：

定义 3.1： 块移动（Block Move）。在当前加工序列$\pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ 中，随机选择从位置i开始的l个连续的工件（块，Block）和目标位置k ，然后，将长度为l 的块移动到当前加工序列π 的位置k 上，其中$\begin{cases} 1 \leqslant l < N \\ i > k \text{ 或 } i < k - l + 1。 \end{cases}$

当$i > k$时，表示将某一个块往前移动；当$i < k - l + 1$时，表示将块往后移动。用三元式表示块移动可以记为$Block\ Move(i, k, l)$ ，i表示被移动的块在当前加工序列中的起始位置；k 表示将块移动到的目标位置；l 表示块的长度。

下面分两种情况介绍块移动的过程：

（1）当$i > k$，即将块往前移动时，是将长度为l的块整个移动到位置k上。移动后，块中的第一个工件的位置变成k，原来位置$[k, \ldots, i-1]$上的一系列工件的新位

置则变成$[k+l,\ldots,i-1+l]$。以工件数为8的加工序列为例，$Block\ Move(5,3,3)$ 的过程如图3-1所示。在$Block\ Move(5,3,3)$中，$i=5$、$k=3$、$l=3$，表示将起始位置为5且长度为3的块向前移动到位置3，即把块{e, f, g}向前移动到位置3，那么，移动后块首工件e的位置变成3。

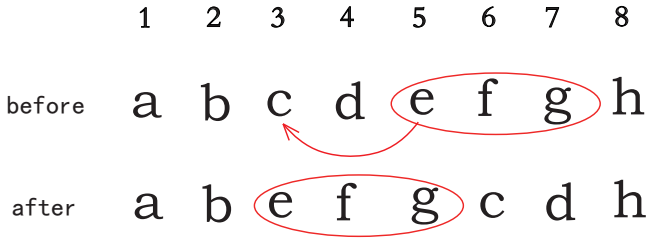

图 3-1 $Block\ Move(5,3,3)$

（2）当$i<k-l+1$时，表示将块往后移动，是将长度为l的整个块向后移动到位置k。移动后，块中最后一个工件的位置变成k，原来位置$[i+l,\ldots,k]$上的工件的新位置则变成$[i,\ldots,k-l]$。同样以包含8个工件的序列为例，$Block\ Move(3,7,3)$的过程如图3-2所示。在$Block\ Move(3,7,3)$中，$i=3$、$k=7$、$l=3$，表示将起始位置为3且长度为3的块向后移动到位置7，即把块{c, d, e}向后移动到位置7，那么，移动后块尾工件e的位置变成7。

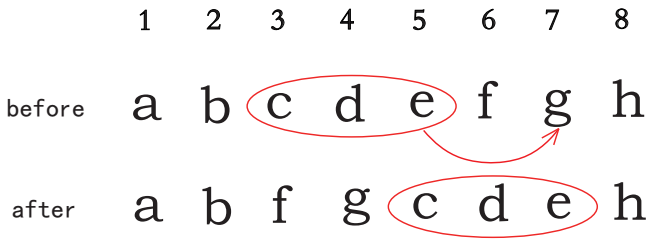

图 3-2 $Block\ Move(3,7,3)$

从图3-2可以看出，把起始位置为3且长度为3的块{c, d, e}往后移动到位置7，等同于把起始位置为6且长度为2的块{f, g}向前移动到位置3。概括的说，把起始位置为i，长度为l的块往后移动到位置k，等同于把起始位置为$i+l$，长度为$k-i-l+1$的块向前移动到位置i。

图3-3中描述了块移动$Block\ Move(i,k,l)$的工作过程。从加工序列$\pi=\{\pi_1,\ldots,\pi_{i-1},\pi_i,\ldots,\pi_{i+l-1},\pi_{i+l},\ldots,\pi_{k-1},\pi_k,\pi_{k+1},\ldots,\pi_N\}$中随机选择一个长度为$l$

的块$\pi_b = \{\pi_i, \ldots, \pi_{i+l-1}\}$，将$\pi_b$插入到$\pi$中的位置$k$，由图3-3可见，原来的3条边（图中打叉的边）被打断，块移动后产生了3条新的边（图中的虚线箭头），由此得到新的加工序列$\pi' = \{\pi_1, \ldots, \pi_{i-1}, \pi_{i+l}, \ldots, \pi_{k-1}, \pi_i, \ldots, \pi_{i+l-1}, \pi_k, \pi_{k+1}, \ldots, \pi_N\}$。由图3-3可见，本文提出的块移动邻域结构与文献中求解TSP问题提出的3-opt的邻域结构有些类似。因此，在今后的研究中，将会尝试把块移动的邻域结构扩展成k-opt。

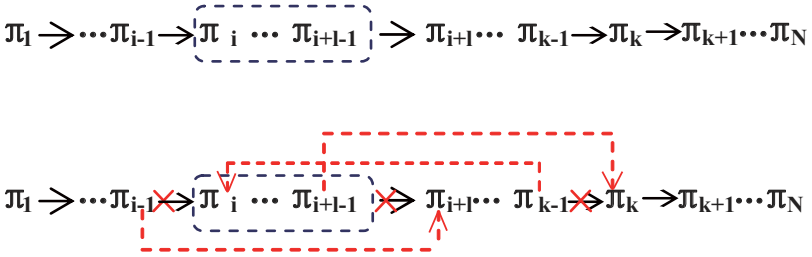

图 3-3 块移动$Block\ Move(i,k,l)$的过程

对比块移动邻域结构$N_{Block\ Move}$和前面介绍的三种邻域结构可以看出，$N_{Block\ Move}$与插入$N_{insertion}$以及边插入$N_{edge-insertion}$邻域结构有一定的联系和差异：

（1）块移动（Block Move）是对l个连续的工件进行移动，插入（insertion）只移动1个工件，边插入（edge-insertion）移动2个连续的工件。也可以说插入和边插入是块移动邻域结构当$l=1$和$l=2$的两种特例情况。

（2）在插入和边插入动作中，有一个把选中的工件和边移出当前加工序列的步骤，而在块移动过程中，没有将选中的块移出当前加工序列的步骤。

通过对块移动邻域结构的介绍可知，块移动邻域的大小为$O(N^3)$，是一种相对比较大的邻域结构。大的邻域结构在搜索过程中有可能找到更好的解，或者在某种程度上可以探索到更多有希望的搜索空间。然而，较大的邻域则意味着需要更多的计算资源。因此，为了在解的质量和计算工作量之间找到一个平衡点，希望通过对块的长度和块移动的距离进行一定的限制，从而即减小了块移动邻域的搜索空间，又在一定程度上保证这个被限制了的搜索空间包含的解是相对较好或有希望的解。

由以上对块移动的介绍可知，将块往后移动可以转化成将某个块往前移动的过程来考虑。因此，在后面的叙述中，以将块往前移动为例进行介绍。在$Block\ Move(i,k,l)$过程中，如何对块长l和块移动的距离$|i-k|$（称为步长）的值

进行限制呢？本文对$1|s_{ij}|\sum w_jT_j$问题的120个公共算例中的一组算例分别进行了如下两组实验，以确定块长和移动步长的取值范围。

第一组实验是为了确定块长l 的值，方案如下：

（1）块的移动步长$|i-k|$ 的取值范围设为$1 \leqslant |i-k| < N$（N 为加工序列的长度），循环运行基于块移动的局部搜索过程，当运行时间大于等于5秒时程序停止。当停止标准没达到并且陷入局部最优时，不进行扰动，而是随机生成一个初始解，继续进行局部搜索。将块的长度l设定9个值进行测试，即$l=\{0.1N, 0.2N, \ldots, 0.9N\}$。对$l$ 的这9个值分别测试20次，计算在相同的运行时间内每个l测试值对应的目标函数值的平均值。

（2）块的移动步长$|i-k|$ 的取值范围设为$1 \leqslant |i-k| < N$，循环运行基于块移动的局部搜索过程，当迭代次数大于等于5000次时程序停止。停止标准没达到并且陷入局部最优时，不进行扰动，而是随机生成一个初始解，继续进行局部搜索。同样将块的长度l 设定9个值进行测试，即$l=\{0.1N, 0.2N, \ldots, 0.9N\}$。对$l$ 的这9个值分别测试1次，记录达到相同迭代次数时所用的CPU时间。

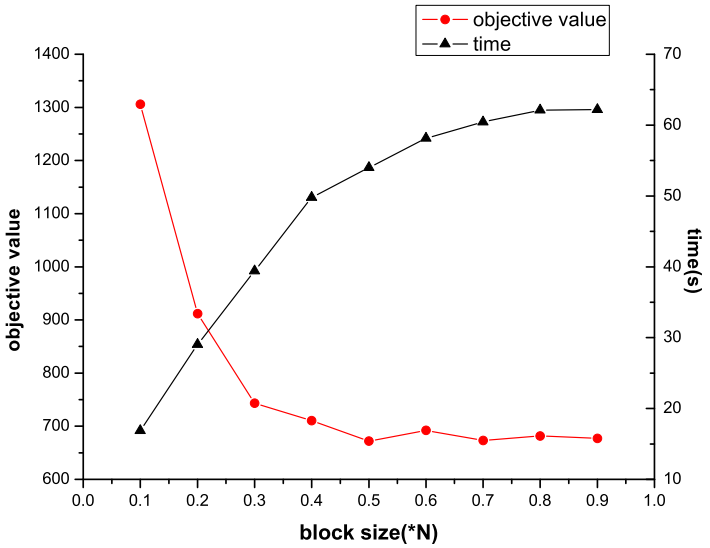

图 3-4 块长取值范围的确定

算例1对于块长进行测试的结果如图3-4所示，图中x 轴表示块长l 的9个测试值；左边的y 轴表示测试20次所得目标函数值的平均值；右边的y 轴表示迭代5000次时所用的CPU时间。从图3-4可以看出，当l 的取值在$0.2N$到$0.4N$ 范围内时，算法的目

标函数值和达到固定迭代次数所需的CPU时间之间达到一个很好的平衡，即目标函数值和所需CPU时间都相对较小。同时，还可以看到，当$l > 0.3N$的时候，目标函数值没有明显下降的趋势，而与此同时，CPU时间却呈现出快速上升的状态。基于这个测试结果，在BILS算法中将块长l的取值限定为小于$\lceil 0.3N \rceil$，即$l \in [1, \lceil 0.3N \rceil]$。需要注意的是，对120个公共算例中的其它算例进行测试，也存在着同样的情况。

第二组实验是在块长l的取值范围已经确定的基础上，寻找移动步长$|i-k|$的取值范围，方案如下：

（1）将块长l的值限定为$l \in [1, \lceil 0.3N \rceil]$，循环运行基于块移动的局部搜索过程，当运行时间大于等于5秒则程序停止。停止标准没达到并且陷入局部最优时，不进行扰动，而是随机生成一个初始解，继续进行局部搜索。对移动步长$|i-k|$设定9个测试值，即$|i-k| = \{0.1N, 0.2N, \ldots, 0.9N\}$，对$|i-k|$的9个值分别测试20次，计算在相同的运行时间内每个$|i-k|$测试值所对应的目标函数值的平均值。

（2）将块长l的值限定为$l \in [1, \lceil 0.3N \rceil]$，循环运行基于块移动的局部搜索过程，当迭代次数大于等于5000次时程序停止。停止标准没达到并且陷入局部最优时，不进行扰动，而是随机生成一个初始解，继续进行局部搜索。对移动步长$|i-k|$设定9个测试值，即$|i-k| = \{0.1N, 0.2N, \ldots, 0.9N\}$，对这9个值分别测试1次，记录达到相同迭代次数时所用的CPU时间。

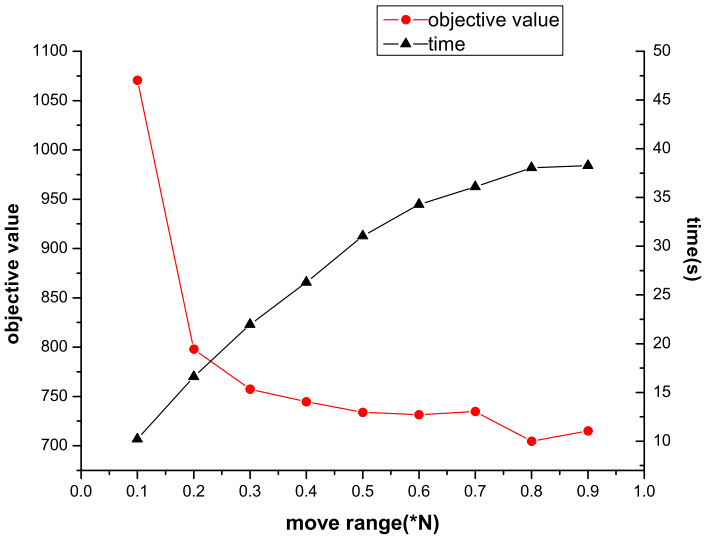

图 3-5 移动步长取值范围的确定

算例1对于移动步长进行测试的实验结果如图3-5所示，x 轴表示的是$|i-k|$的9个测试值；左边的y轴表示对于$|i-k|$ 的9个测试值，运行测试程序20次，所得目标函数值的平均值；右边的y 轴表示达到5000次迭代次数时CPU时间值。通过对实验结果观察可以得出，当$|i-k|$ 的取值在$0.2N$ 到$0.4N$范围内的时候，在目标函数值和计算时间之间达到了一个很好的平衡。因此，将移动步长$|i-k|$ 值限制为小于$\lceil 0.3N \rceil$ ，即$|i-k| \in [1, \lceil 0.3N \rceil]$ 。

通过对块长及移动步长取值范围的确定，限制了$N_{Block\ Move}$ 的搜索空间，这样，就可以节省可观的CPU时间去搜索邻域中更有希望的一些区域。另外，上述实验表明，虽然对$N_{Block\ Move}$ 中块的大小和移动步长进行了限制，但是，块移动搜索的有效性并没有因此减弱，因为，这些被限制掉的动作都是一些比较差的解。

分析迭代局部搜索算法的复杂度是非常困难的，然而，可以分析它一次局部搜索过程的时间复杂度。BILS算法的局部搜索过程中邻域结构采用的是$N_{Block\ Move}$，每次局部搜索的时间复杂度为$O(N^3)$ 。非常明显的是，采用插入、边插入和交换这三种邻域结构的启发式算法，每次局部搜索的时间复杂度为$O(N^2)$ 。虽然BILS算法的时间复杂度比采用插入、边插入和交换这三种邻域结构的启发式算法要高，但是，BILS算法在一次局部搜索过程中可以找到质量更好的解，并且通过以上介绍的对块长和移动步长的限制技术后，可以使BILS算法在效果（Effectiveness）和效率（Efficiency）之间取得很好的平衡。

三、快速增量评估技术

块移动是一种相对较大的邻域结构，在一次局部搜索过程中，需要评估的邻域解就会更多。为了更快的评估邻域中的解，本文提出了一种快速增量评估技术来评估邻域解。快速增量评估技术的主要思想是：加工序列π 在做Block Move动作后，产生的新序列π'中的工件与原加工序列π 中相应工件之间存在时间上的增量Δ ，通过Δ 值来计算移动后被影响的工件π_j所产生的加权延迟增量值$\Delta f(\pi_j)$，最后根据序列中所有被影响工件的加权延迟增量值之和$\Delta f(\pi)$ 就能计算出新序列π' 的目标函数值$f(\pi')$ ，这样做可以大大减少计算量，因此能实现对邻域解的快速评估。

以$Block\ Move(i,k,l)(i>k)$为例来详细介绍本文新提出的快速增量评估技术。$Block\ Move(i,k,l)(i>k)$表示将加工序列π 中长度为l 的块向前移动到位置k

处，块中的第一个工件在加工序列π中的位置为i。图3-6表示对加工序列π进行一次$Block\ Move(i,k,l)$操作之前和之后的变化情况。后续文中用到的一些标记的意义如下所示：

N：工件数；

π：由N个工件组成的加工序列；

π_j：加工序列π中的第j个工件；

C_{π_j}：工件π_j 的完成时间；

X_{π_j}：工件π_j 的开始时间；

w_{π_j}：工件π_j 的权值；

d_{π_j}：工件π_j 的预期完成时间；

p_{π_j}：工件π_j 的加工时间；

$s_{0\pi_j}$：当π_j是加工序列π中的第1个工件时的初始准备时间；

$s_{\pi_i\pi_j}$：π_j 紧跟着工件π_i 之后加工所需要的准备时间。

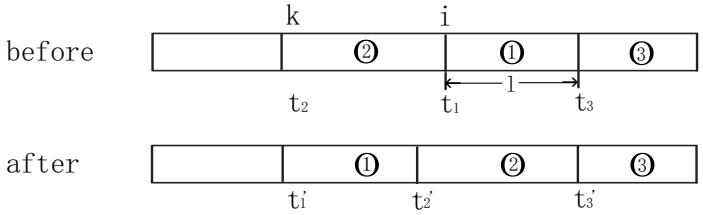

图 3-6 快速增量评估技术示意图

在图3-6中t_1 、t_2 和t_3 分别表示块①、②和③中位于起始位置的工件在执行Block Move动作之前的开始时间。t_1' 、t_2' 和t_3' 分别表示块①、②和③中处于起始位置的工件在执行了Block Move动作之后的开始时间。Δ_1 、Δ_2 和Δ_3表示执行Block Move动作之后块①、②和③中的工件在加工时产生的时间上的增量。Δ_1 、Δ_2 和Δ_3的计算公式如下：

$$\Delta_1 = t_1' - t_1 \tag{式 3.3}$$

$$\Delta_2 = t_2' - t_2 \tag{式 3.4}$$

$$\Delta_3 = t_3' - t_3 \tag{式 3.5}$$

具体计算过程如下：

$$\Delta_1 = \begin{cases} s_{0\pi_i} - X_{\pi_i},\ k = 1 \\ (C_{\pi_{k-1}} + s_{\pi_{k-1}\pi_i}) - X_{\pi_i}, otherwise \end{cases} \tag{式 3.6}$$

$$\Delta_2 = (C_{\pi_{i+l-1}} + \Delta_1 + s_{\pi_{i+l-1}\pi_k}) - X_{\pi_k} \tag{式 3.7}$$

$$\Delta_3 = (C_{\pi_{i-1}} + \Delta_2 + s_{\pi_{i-1}\pi_{i+l}}) - X_{\pi_{i+l}} \tag{式 3.8}$$

下面分步骤详细介绍快速增量评估中$f(\pi')$的计算过程：

（1）如图3-6所示，在对当前加工序列π进行一次$Block\ Move(i,k,l)$操作后，因此受到影响的工件全部包含在块①、②和③中。所以，当工件π_j处在不同的块的时候，其增量δ的取值情况也各不相同：

当工件π_j在块①中的时候，即$j \in [i, i+l-1]$时，$\delta = \Delta_1$；

当工件π_j在块②中的时候，即$j \in [k, i-1]$时，$\delta = \Delta_2$；

当工件π_j在块③中的时候，即$j \in [i+l, N]$时，$\delta = \Delta_3$；

（2）从图3-6可以看出，在对当前加工序列π进行一次$Block\ Move(i,k,l)$操作后，受到影响的工件为$\pi_j, j = \{k, \ldots, N\}$。也就是说，受到影响的工件为插入位置$k$之后的所有工件。那么，对于位置$j$上的工件，$\Delta f(\pi_j)$又分为下面几种情况来考虑：

当工件π_j移动前的完成时间$C_{\pi_j} \leqslant d_{\pi_j}$，且移动后$C_{\pi_j} + \delta \leqslant d_{\pi_j}$时，则说明工件$\pi_j$在块移动前后的加权延迟均为0。由于移动前后的加权延迟都为0，那么增量也为0，即$\Delta f(\pi_j) = 0$。

当工件π_j移动前的完成时间$C_{\pi_j} \leqslant d_{\pi_j}$，而移动后$C_{\pi_j} + \delta > d_{\pi_j}$时，则说明工件$\pi_j$移动前的加权延迟为0，块移动操作后产生了加权延迟。那么，增量为产生的加权延迟，即$\Delta f(\pi_j) = w_{\pi_j}(C_{\pi_j} + \delta - d_{\pi_j})$。

当工件π_j移动前的完成时间$C_{\pi_j} > d_{\pi_j}$，且移动后$C_{\pi_j} + \delta > d_{\pi_j}$时，则说明工件$\pi_j$移动前加权延迟不为0，块移动后加权延迟仍然不为0。那么，增量为移动后的加权延迟减去移动前的加权延迟，即$\Delta f(\pi_j) = w_{\pi_j}\delta$。

当工件π_j移动前的完成时间$C_{\pi_j} > d_{\pi_j}$，而移动后$C_{\pi_j} + \delta \leqslant d_{\pi_j}$时，则说明工件$\pi_j$移动前加权延迟不为0，块移动后加权延迟为0。那么，增量为0减去移动前的加权延迟，即$\Delta f(\pi_j) = -w_{\pi_j}(C_{\pi_j} - d_{\pi_j})$。

（3）求出$j = \{k, \ldots, N\}$上的每一个工件π_j的$\Delta f(\pi_j)$值后，计算出$\Delta f(\pi) = \sum_{j=k}^{N} \Delta f(\pi_j)$。则块移动后产生的新加工序列$\pi'$的目标函数值$f(\pi') = f(\pi) + \Delta f(\pi)$。

Algorithm 2 快速增量评估技术

$f(\pi) \leftarrow$ objective function value of the current job sequence π
$\Delta f(\pi) \leftarrow 0$
for *Block Move* (*i*, *k*, *l*) (*i*>*k*), calculate Δ_1, Δ_2 and Δ_3 as follows:

$$\Delta_1 = \begin{cases} s_{0\pi_i} - X_{\pi_i}, & \text{if } k = 1; \\ (C_{\pi_{k-1}} + s_{\pi_{k-1}\pi_i}) - X_{\pi_i}, & \text{otherwise.} \end{cases}$$

$$\Delta_2 = (C_{\pi_{i+l-1}} + \Delta_1 + s_{\pi_{i+l-1}\pi_k}) - X_{\pi_k}$$

$$\Delta_3 = (C_{\pi_{i-1}} + \Delta_2 + s_{\pi_{i-1}\pi_{i+l}}) - X_{\pi_{i+l}}$$

for $j = k$ to N **do**
  if $(j \in [i, i+l))$ **then**
    $\delta \leftarrow \Delta_1$
  else if $(j \in [k, i))$ **then**
    $\delta \leftarrow \Delta_2$
  else if $(j \in [i+l, N])$ **then**
    $\delta \leftarrow \Delta_3$
  end if
  if $(C_{\pi_j} \leqslant d_{\pi_j})$ **then**
    if $(C_{\pi_j} + \delta \leqslant d_{\pi_j})$ **then**
      continue
    else
      $\Delta f(\pi) \leftarrow \Delta f(\pi) + w_{\pi_j}(C_{\pi_j} + \delta - d_{\pi_j})$
    end if
  else
    if $(C_{\pi_j} + \delta > d_{\pi_j})$ **then**
      $\Delta f(\pi) \leftarrow \Delta f(\pi) + w_{\pi_j}\delta$
    else
      $\Delta f(\pi) \leftarrow \Delta f(\pi) - w_{\pi_j}(C_{\pi_j} - d_{\pi_j})$
    end if
  end if
end for
$f(\pi') \leftarrow f(\pi) + \Delta f(\pi)$

在BILS算法中用来评估块移动的邻域解的快速增量评估技术的伪码见算法2。

确切的说，它计算了执行一次$Block\ Move(i,k,l)$ 操作后所产生的目标函数增量值。执行一次$Block\ Move(i,k,l)$ 操作后，对于块①、②和③会产生3个增量值Δ_1、Δ_2和Δ_3，分别表示执行块移动操作后，块①、②和③中的工件在加工时所产生的时间上的增量。这样，每个受影响工件的加权延迟增量值可以由这3 个Δ值计算得到，从而能更快的计算出块移动操作后产生的加工序列的目标函数值。

3.2.4 扰动

在迭代局部搜索算法的局部搜索过程中，当局部最优解再也不能被改进并且停止标准还没到达时，采用扰动算子来重构当前的局部最优解，然后，再以重构的局部最优解出发，继续进行局部搜索。扰动算子必须解决两个问题：（1）怎么扰动；（2）扰动强度的大小。作为迭代局部搜索算法构架的主要组成部分，扰动算子帮助迭代局部搜索算法找到更多样化的搜索空间，只有这样，才可能找到更好的解。

BILS算法中，采用了一种简单并且有效的扰动策略。具体说来，扰动过程就是随机地对当前加工序列进行块移动操作，即随机选择符合取值范围的i、k、l的值进行$Block\ Move(i,k,l)$操作。那么，扰动强度（Perturbation Strength）如何确定，具体的说，即应该进行多少次块移动操作? 为此，本文对$1|s_{ij}|\sum w_jT_j$ 问题的公共算例中的一些算例进行实验，用来确定BILS算法中扰动强度参数。实验方案设计如下：

实验过程中，块长取值为$l \in [1, \lceil 0.3N \rceil]$，移动步长取值为$|i-k| \in [1, \lceil 0.3N \rceil]$，扰动策略为随机选择符合取值范围的$i$、$k$、$l$的值进行块移动操作，实验停止条件为100秒CPU计算时间。扰动强度参数设置为$\{0.1N, 0.2N, \ldots, 0.9N\}$9个值，用BILS算法对这9个值分别测试20次，计算在9个不同的扰动强度参数下，目标函数值的平均值。

图3-7表示对于算例1，在9种扰动强度参数下，BILS算法分别执行20次后所得目标函数值的平均值。由图可以看出扰动强度值处于$0.2N$ 到$0.6N$ 范围的时候，目标函数值的质量保持在一个相对较好的水平。根据实验结果，将扰动强度的范围限定在$[0.3N, 0.5N]$ 区间内，即在进行扰动操作时，需要进行的Block Move操作的次数为区间$[0.3N, 0.5N]$ 里的一个随机数。

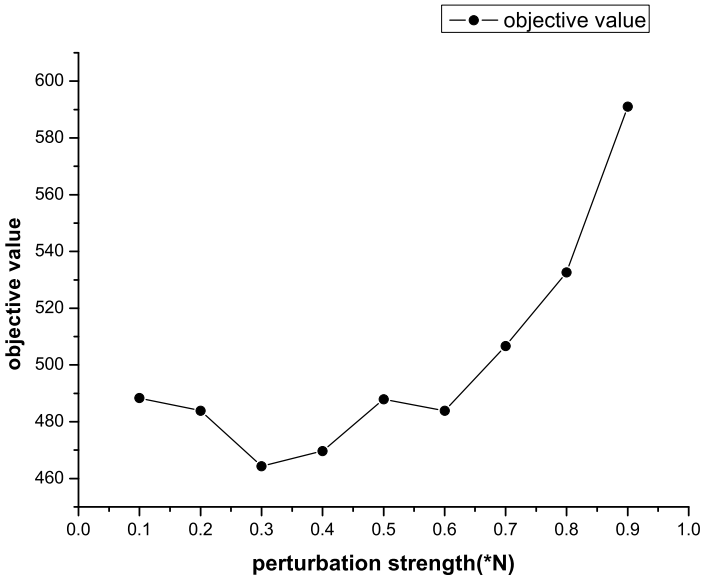

图 3-7 扰动强度的确定

3.2.5 接受标准和停止标准

完成一次局部搜索过程之后，就要将新的局部最优解π'和当前最好解π进行比较。在BILS算法中，判断解的优劣就是直接比较目标函数值的大小，$1|s_{ij}|\sum w_jT_j$是一个最小化问题，目标函数值小的解更优。如果$f(\pi')$小于当前最好解$f(\pi)$，就用π'来取代当前最好解π，即若$f(\pi') < f(\pi)$，则$\pi = \pi'$。

BILS算法的停止标准既可以是事先设定的CPU运行时间或最大迭代次数，也可以是当最优解被找到后则算法终止。

3.3 计算结果及与其它算法的比较

在研究$1|s_{ij}|\sum w_jT_j$问题时，文献中经常使用由Cicirello和Smith[76]根据文献[63]中的规则生成的120个算例对算法进行测试。当前在这组算例上取得最好成绩的是由Tanaka和Araki在文献[54]中介绍的一种精确算法，算出了所有120个算例的最优解。之前Kirlik和Oguz[83]提出的变邻域搜索算法和Tasgetiren等人[82]提出的离散差分进化算法在对这组算例进行测试时也取得了相当好的成绩。本节中把BILS算法的测试结果与文献[54]中找到的最优解以及由几种最优秀的启发式算法得到的最好解进行了对比。为了更好的进行描述，本文将Tanaka和Araki的精确算法找到的最优解

和其它几种启发式算法算出的最好解表示如下：

（1）OPT：表示由Tanaka 和Araki[54]提出的精确算法所得到的最优解。

（2）OBK：是一组算法的最好解的集合，包括Lin 和Ying[78] 提出的模拟退火算法、遗传算法和禁忌算法；Liao 和Juan[73] 提出的蚁群算法；Cicirello[79] 提出的遗传算法。

（3）ACO_AP：Anghinolfi 和Paolucci[77]提出的蚁群算法的最好解。

（4）DPSO：Anghinolfi和Paolucci[81]提出的离散粒子群算法的最好解。

（5）DDE：Tasgetiren等人[82]提出的离散差分进化算法的最好解。

（6）GVNS：Kirlik和Oguz[83]提出的变邻域搜索算法的最好解。

3.3.1 问题所用算例及实验环境

Cicirello 和Smith[76]为研究$1|s_{ij}|\sum w_jT_j$ 问题生成的公共算例由120个算例组成，每个算例包含60个工件。这些算例是依据3个参数生成的：due-date tightness参数τ，取值为$\tau = \{0.3, 0.6, 0.9\}$；due-date range参数R，取值为$R = \{0.25, 0.75\}$；setup time severity参数η，取值为$\eta = \{0.25, 0.75\}$。对于这3个参数值的12种组合的每一种组合，各生成10个算例，总计一共是120个算例。

BILS算法用VC++编程实现，程序运行环境为Core i3 CPU，主频3.1GHz，内存2GB。

3.3.2 实验结果及分析

本实验用$1|s_{ij}|\sum w_jT_j$问题的120个公共算例对BILS算法进行测试，每个算例用随机种子独立运行100次，每次的停止标准设置为100秒的CPU时间。

把BILS算法对$1|s_{ij}|\sum w_jT_j$问题的120个公共算例的计算结果和Tanaka和Araki的精确算法找到的最优解以及几种优秀启发式算法得到的最好解进行了对比。表3.1记录了BILS算法对于120个算例的计算结果。表中第2-5列分别给出了测试算例的性质：工件的个数N、due-date tightness参数τ、due-date range参数R、setup time severity参数η；第6列列出精确算法所得到的最优解OPT；第7列列出的是由五组启发式算法算出的最好解所组成的当前最好解f^*；8-11列中列出了BILS 算法的实验结果，包括BILS算法找到的最好解（f_{best}）、运行100次所得的目标函数值的平均值（f_{aver}）、

找到最好解f_{best}所需要的最短的CPU时间（t_{min}）、找到最好解f_{best}所需要的CPU时间的平均值（t_{aver}）；最后一列#suc表示BILS算法在100次的运行次数中找到最优解OPT的次数。

从表3.1可以看出，BILS算法不论是跟精确算法所得的最优解OPT相比，还是跟当前一些优秀启发式算法所得的当前最好解f^*相比，都具有很强的竞争力。和OPT相比，120个算例，BILS算法可以找到113个算例的最优解，并且余下的7个算例的解也非常接近最优解。这个结果表明了BILS算法的有效性和高效性。

表 3.1: BILS算法求解1$|s_{ij}|\sum w_jT_j$ 问题120个公共算例的计算结果

算例	N	τ	R	η	OPT	f^*	BILS 算法				
							f_{best}	f_{aver}	t_{min}	t_{aver}	#suc
1	60	0.3	0.25	0.25	**453**	471	**453**	480.33	85.71	85.71	1
2	60	0.3	0.25	0.25	**4794**	4878	**4794**	4886.97	25.88	59.30	2
3	60	0.3	0.25	0.25	**1390**	1430	**1390**	1457.54	34.07	63.27	2
4	60	0.3	0.25	0.25	**5866**	5946	**5866**	5978.40	8.69	49.64	25
5	60	0.3	0.25	0.25	**4054**	4084	4074	4215.87	46.61	76.13	–
6	60	0.3	0.25	0.25	**6592**	5788[a]	**6592**	6750.26	22.14	48.39	8
7	60	0.3	0.25	0.25	**3267**	3330	**3267**	3404.18	88.42	88.42	1
8	60	0.3	0.25	0.25	**100**	108	**100**	106.21	71.65	82.73	2
9	60	0.3	0.25	0.25	**5660**	5751	**5660**	5840.84	35.27	65.67	5
10	60	0.3	0.25	0.25	**1740**	1789	**1740**	1793.00	41.61	67.85	3
11	60	0.3	0.25	0.75	**2785**	2998	2830	3125.04	55.58	55.58	–
12	60	0.3	0.25	0.75	**0**	0	**0**	0.00	0.11	0.19	100
13	60	0.3	0.25	0.75	**3904**	4068	3942	4141.87	85.24	89.63	–
14	60	0.3	0.25	0.75	**2075**	2260	2081	2315.06	54.65	54.65	–
15	60	0.3	0.25	0.75	**724**	935	775	891.27	96.99	96.99	–
16	60	0.3	0.25	0.75	**3285**	3381	**3285**	3413.50	33.82	57.64	3
17	60	0.3	0.25	0.75	**0**	0	**0**	13.61	1.05	43.11	62
18	60	0.3	0.25	0.75	**767**	845	**767**	813.47	86.60	86.60	1
19	60	0.3	0.25	0.75	**0**	0	**0**	0.00	0.25	1.28	100
20	60	0.3	0.25	0.75	**1757**	2053	**1757**	1938.90	22.99	57.01	6
21	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.13	0.29	100
22	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.13	0.19	100
23	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.08	0.11	100
24	60	0.3	0.75	0.25	**761**	920	773	1037.24	61.84	61.84	–

25	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.16	0.38	100
26	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.16	0.28	100
27	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.45	4.29	100
28	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.19	0.41	100
29	60	0.3	0.75	0.25	**0**	0	**0**	0.00	0.13	0.24	100
30	60	0.3	0.75	0.25	**0**	0	**0**	28.15	3.18	39.61	72
31	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.11	0.15	100
32	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.17	100
33	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.18	100
34	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.18	100
35	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.11	0.14	100
36	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.15	100
37	60	0.3	0.75	0.75	**0**	46	**0**	293.26	51.03	51.03	1
38	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.17	100
39	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.14	0.20	100
40	60	0.3	0.75	0.75	**0**	0	**0**	0.00	0.13	0.16	100
41	60	0.6	0.25	0.25	**69102**	69102	**69102**	69318.16	11.36	58.43	50
42	60	0.6	0.25	0.25	**57487**	57487	**57487**	57644.63	1.79	46.32	73
43	60	0.6	0.25	0.25	**145310**	145310	**145310**	145930.47	3.29	34.03	9
44	60	0.6	0.25	0.25	**35166**	35289	**35166**	35328.32	3.21	27.83	12
45	60	0.6	0.25	0.25	**58935**	58935	**58935**	59018.58	2.98	43.71	57
46	60	0.6	0.25	0.25	**34764**	34764	**34764**	35034.84	17.05	57.71	13
47	60	0.6	0.25	0.25	**72853**	72853	**72853**	73160.37	6.97	44.39	43
48	60	0.6	0.25	0.25	**64612**	64612	**64612**	64719.63	10.44	52.18	38
49	60	0.6	0.25	0.25	**77449**	77641	**77449**	78176.63	4.96	53.58	10
50	60	0.6	0.25	0.25	**31092**	31292	**31092**	31580.58	31.78	52.42	13
51	60	0.6	0.25	0.75	**49208**	49761	**49208**	50082.49	27.46	63.20	8
52	60	0.6	0.25	0.75	**93045**	93106	**93045**	94653.25	15.16	40.93	7
53	60	0.6	0.25	0.75	**84841**	84841	**84841**	86465.29	13.04	56.70	18
54	60	0.6	0.25	0.75	**118809**	118809	**118809**	120150.82	21.61	52.93	12
55	60	0.6	0.25	0.75	**64315**	65400	**64315**	66055.39	11.42	51.76	9
56	60	0.6	0.25	0.75	**74889**	74940	**74889**	75472.85	12.79	52.49	26
57	60	0.6	0.25	0.75	**63514**	64552	**63514**	65195.36	23.26	31.44	3
58	60	0.6	0.25	0.75	**45322**	45322	**45322**	46286.07	22.71	57.54	5
59	60	0.6	0.25	0.75	**50999**	51649	**50999**	51954.03	24.84	71.24	13
60	60	0.6	0.25	0.75	**60765**	60765	**60765**	62498.75	22.23	55.43	9
61	60	0.6	0.75	0.25	**75916**	75916	**75916**	75998.49	5.63	36.99	86
62	60	0.6	0.75	0.25	**44769**	44769	**44769**	44840.03	5.90	38.81	12

63	60	0.6	0.75	0.25	**75317**	75317	**75317**	75536.88	5.10	48.22	59
64	60	0.6	0.75	0.25	**92572**	92572	**92572**	92609.80	2.65	28.10	70
65	60	0.6	0.75	0.25	**126696**	126696	**126696**	127080.12	8.94	39.55	68
66	60	0.6	0.75	0.25	**59685**	59685	**59685**	59927.30	4.54	56.13	43
67	60	0.6	0.75	0.25	**29390**	29390	**29390**	29415.80	3.01	27.40	40
68	60	0.6	0.75	0.25	**22120**	22120	**22120**	22264.57	4.03	30.46	52
69	60	0.6	0.75	0.25	**71118**	64632[a]	**71118**	71307.72	0.87	37.10	12
70	60	0.6	0.75	0.25	**75102**	75102	**75102**	75427.43	2.36	28.19	23
71	60	0.6	0.75	0.75	**145007**	145007	**145007**	147119.72	8.95	41.80	13
72	60	0.6	0.75	0.75	**43286**	43286	**43286**	45678.28	37.10	63.72	5
73	60	0.6	0.75	0.75	**28785**	28785	**28785**	29054.36	16.91	60.18	16
74	60	0.6	0.75	0.75	**29777**	30136	**29777**	30765.00	10.72	47.75	9
75	60	0.6	0.75	0.75	**21602**	21602	**21602**	22450.00	10.78	61.60	14
76	60	0.6	0.75	0.75	**53555**	53555	**53555**	54529.21	11.06	50.39	17
77	60	0.6	0.75	0.75	**31817**	31817	31937	33374.42	63.91	82.16	–
78	60	0.6	0.75	0.75	**19462**	19462	**19462**	20381.75	10.52	43.91	17
79	60	0.6	0.75	0.75	**114999**	114999	**114999**	116196.84	5.57	50.39	36
80	60	0.6	0.75	0.75	**18157**	18157	**18157**	19274.40	5.68	58.12	8
81	60	0.9	0.25	0.25	**383485**	383485	**383485**	383903.09	0.91	42.74	39
82	60	0.9	0.25	0.25	**409479**	409479	**409479**	409815.64	6.82	49.65	17
83	60	0.9	0.25	0.25	**458752**	458752	**458752**	458922.11	4.45	35.79	18
84	60	0.9	0.25	0.25	**329670**	329670	**329670**	329969.66	11.01	51.82	16
85	60	0.9	0.25	0.25	**554766**	554766	**554766**	555107.20	31.26	31.26	1
86	60	0.9	0.25	0.25	**361417**	361417	**361417**	361826.32	3.15	31.59	76
87	60	0.9	0.25	0.25	**398551**	398551	**398551**	398623.75	0.55	17.44	41
88	60	0.9	0.25	0.25	**433186**	433186	**433186**	433564.36	10.05	44.58	3
89	60	0.9	0.25	0.25	**410092**	410092	**410092**	410187.27	4.04	28.65	19
90	60	0.9	0.25	0.25	**401653**	401653	**401653**	401825.54	3.25	36.64	36
91	60	0.9	0.25	0.75	**339933**	339933	**339933**	340079.31	3.10	35.60	63
92	60	0.9	0.25	0.75	**361152**	361152	**361152**	362030.63	4.35	53.78	47
93	60	0.9	0.25	0.75	**403423**	404548	**403423**	405344.01	13.79	49.58	16
94	60	0.9	0.25	0.75	**332941**	332949	**332941**	333319.28	9.12	36.19	7
95	60	0.9	0.25	0.75	**516926**	517011	**516926**	518751.30	3.10	50.21	12
96	60	0.9	0.25	0.75	**455448**	457631	**455448**	457814.42	54.08	54.08	1
97	60	0.9	0.25	0.75	**407590**	407590	**407590**	408437.75	5.77	52.10	51
98	60	0.9	0.25	0.75	**520582**	520582	**520582**	522055.23	18.15	52.08	30
99	60	0.9	0.25	0.75	**363518**	363977	**363518**	364803.19	37.58	59.03	5
100	60	0.9	0.25	0.75	**431736**	431736	**431736**	433105.79	0.96	35.75	26

101	60	0.9	0.75	0.25	**352990**	352990	**352990**	353033.10	1.19	24.61	90
102	60	0.9	0.75	0.25	**492572**	492572	**492572**	492832.58	2.71	33.88	68
103	60	0.9	0.75	0.25	**378602**	378602	**378602**	378834.01	6.30	30.19	78
104	60	0.9	0.75	0.25	**357963**	357963	**357963**	358174.62	7.93	37.44	24
105	60	0.9	0.75	0.25	**450806**	450806	**450806**	450812.42	2.96	28.52	30
106	60	0.9	0.75	0.25	**454379**	454379	**454379**	454851.60	9.47	44.50	41
107	60	0.9	0.75	0.25	**352766**	352766	**352766**	353002.25	3.54	42.90	52
108	60	0.9	0.75	0.25	**460793**	460793	**460793**	461112.62	5.73	44.97	61
109	60	0.9	0.75	0.25	**413004**	413004	**413004**	413426.10	6.15	31.64	20
110	60	0.9	0.75	0.25	**418769**	418769	**418769**	419030.35	6.88	38.08	39
111	60	0.9	0.75	0.75	**342752**	342752	**342752**	343767.96	9.05	40.85	32
112	60	0.9	0.75	0.75	**367110**	367110	**367110**	369592.02	6.80	42.89	34
113	60	0.9	0.75	0.75	**259649**	259649	**259649**	259909.21	4.68	43.51	33
114	60	0.9	0.75	0.75	**463474**	463474	**463474**	465440.15	10.69	47.23	20
115	60	0.9	0.75	0.75	**456890**	457189	**456890**	458414.67	24.45	60.85	5
116	60	0.9	0.75	0.75	**530601**	527459[a]	**530601**	531410.83	5.23	35.75	45
117	60	0.9	0.75	0.75	**502840**	502840	**502840**	503600.90	12.01	50.47	33
118	60	0.9	0.75	0.75	**349749**	349749	**349749**	352050.87	5.80	53.89	18
119	60	0.9	0.75	0.75	**573046**	573046	**573046**	573759.27	3.95	46.15	47
120	60	0.9	0.75	0.75	**396183**	396183	**396183**	398005.14	2.84	45.80	40

黑体：最优解，[a]：不正确的解

与优秀的启发式算法所组成的当前最好解f^*相比，120个算例，BILS算法对于其中34个算例的解都要优于当前最好解，另外82个算例的解和当前最好解持平，也就是说，只有4个算例的解比当前最好解要差。上述实验结果展示出BILS算法极强的搜索能力。

3.3.3 与精确算法对比

这一节中，将BILS算法与Tanaka和Araki[54]新近提出的精确算法进行对比，该算法算出了所有120个算例的最优解。从表3.1可以看出，和精确算法得到的最优解OPT相比，BILS算法在100次100秒的实验条件下一共有7个算例没有达到OPT，它们分别是算例5、11、13、14、15、24和77。这7个算例里，除了算例77，其余的全包含在前40个算例中。这120个算例中的前40个算例在文献[54]中被认为是最具有挑战性的算例。

为了测试BILS算法的寻优能力，用这7个困难算例对BILS重新测试。为了和精确算法的最优解做一个公正的对比，对这7个算例中的每个算例只测试一次，算法的停止标准设为达到OPT则算法终止。在算法的执行过程中，还设制了如果当前最好解在连续20000次扰动后仍不能改进，则随机生成一个初始解，重新开始执行BILS。表3.2列出了BILS算法对于这7个算例的结果与Tanaka和Araki的精确算法的对比。

表 3.2: BILS算法对于7个困难算例的结果与精确算法对比

算例	Tanaka和Araki的精确算法				BILS算法	
	OPT	时间（秒）			f_{best}	t_{opt}
		512MB	2GB	20GB		
5	**4054**	7259.98	4198.60	3292.41	**4054**	10220.16
11	**2785**	123538.53	51443.47	23359.73	**2785**	1141.89
13	**3904**	19599.13	10677.42	8212.44	**3904**	1182.79
14	**2075**	15645.10	5049.56	5906.87	**2075**	22460.12
15	**724**	4404.47	1632.86	841.15	**724**	950.92
24	**761**	–	–	30 days	**761**	150169.22
77	**31817**	195.42			**31817**	11064.24

注：精确算法的运行环境为Intel Core i7 980X Extreme Edition CPU (3.33 GHz)和24GB内存；BILS算法的运行环境为Intel Core i3 3.1 GHz CPU和2GB内存。

表3.2中，第2列是精确算法得到的最优解；3-5列给出的是精确算法在3种最大内存（即512MB、2GB和20GB）设置下找到最优解的CPU时间；第6列给出的是BILS算法找到的最好解（f_{best}）；最后一列给出的是BILS算法在找到f_{best}的时候所需要的CPU时间（t_{opt}）。从表3.2可以清楚的看到，对这7个算例，BILS算法都能够找到精确算法所得到的最优解。就达到OPT的CPU时间而言，BILS算法跟精确算法相比在其中几个算例上具有一定的优势。例如，对于算例24，精确算法找到最优解的时间在20GB的内存环境下为30天，而BILS算法找到最优解的时间仅仅为150169.22秒（≈41.7小时）。对于算例11和13，BILS算法找到最优解的时间比精确算法短。考虑到精确算法是在大约比我们快两倍的计算机环境下运行的，BILS算法在算例5、14和15的计算时间上和精确算法相比，也是不相上下甚至略有竞争力的。对于算例77，BILS算法为了找到最优解，跟精确算法比起来需要更多的CPU时间。

与此同时，进一步来对比Tanaka和Araki的精确算法和BILS算法的CPU运行时

间。对于算例的3个参数(τ, R, η)一共有12种不同的组合，每种组合有10个算例。以12种组合中每一种组合的10个算例为一组，对比观察精确算法和BILS算法达到OPT的CPU时间的平均值。实验方案同上，即BILS算法对120个公共算例中的每个算例只运行一次，直到找到OPT则算法终止。实验结果如表3.3所示，第2-4列给出的是精确算法在3种内存环境下（512MB、2GB和20GB）的CPU计算时间的平均值；最后一列给出的是BILS算法找到OPT的CPU计算时间的平均值。

表 3.3: BILS算法与精确算法的平均CPU计算时间对比

τ，R，η	Tanaka 和Araki的精确算法			BILS算法
	平均时间（秒）			平均时间（秒）
	512MB	2GB	20GB	
$\tau = 0.3, R = 0.25, \eta = 0.25$(001-010)	3678.31	2270.09	2525.96	4180.60
$\tau = 0.3, R = 0.25, \eta = 0.75$(011-020)	–	–	125636.19	4023.02
$\tau = 0.3, R = 0.75, \eta = 0.25$(021-030)	–	–	259202.23	15025.90
$\tau = 0.3, R = 0.75, \eta = 0.75$(031-040)	121.85	120.49	371.61	485.25
$\tau = 0.6, R = 0.25, \eta = 0.25$(041-050)	138.25			509.14
$\tau = 0.6, R = 0.25, \eta = 0.75$(051-060)	256.24			1991.65
$\tau = 0.6, R = 0.75, \eta = 0.25$(061-070)	79.93			389.35
$\tau = 0.6, R = 0.75, \eta = 0.75$(071-080)	180.90			1888.66
$\tau = 0.9, R = 0.25, \eta = 0.25$(081-090)	129.50			1013.84
$\tau = 0.9, R = 0.25, \eta = 0.75$(091-100)	214.46			552.40
$\tau = 0.9, R = 0.75, \eta = 0.25$(101-110)	118.33			101.44
$\tau = 0.9, R = 0.75, \eta = 0.75$(111-120)	241.05			286.41

从表3.3可以看到，对于(τ, R, η) 的第2和3种组合对应的算例，BILS算法的平均CPU时间要小于精确算法的平均时间。确切地说，对于算例18和24，精确算法分别需要两星期和30天才能找到最优解，然而，BILS算法仅仅需要8586.6秒（≈2.4小时）和150169.22 秒（≈41.7小时）的时间。考虑到精确算法是在两倍于我们的计算机环境下运行的，对于这12个组合中的其中5个组合，BILS算法的计算时间比精确算法的计算时间要少，剩下的7个组合则是精确算法需要的计算时间要更短。从表3.2和3.3可以得出结论，BILS算法在计算效率上和精确算法相比是有可比性的。

3.3.4 与其它启发式算法的比较

表3.4中，总结了BILS算法对于120个算例，每个算例运行100次，每次运行100秒的结果和五组优秀启发式算法的最好解相比，改进、持平及差的解的个数。可以看到，在这种对比情况下，BILS算法具有很强的竞争力。BILS算法的结果跟OBK、ACO_AP、DPSO、DDE和GVNS相比，改进的解的个数分别为94、84、65、52和44，并且，未改进的解的个数最多不超过3个。

表 3.4 BILS算法和相关算法对比结果统计

	OPT	OBK	ACO_AP	DPSO	DDE	GVNS	f^*
改进解的个数	0	94	84	65	52	44	34
持平解的个数	113	23	36	55	68	75	82
差的解的个数	7	3	0	0	0	1	4
所有解的个数	120	120	120	120	120	120	120

表3.4的最后一列给出的是BILS算法和五组启发式算法的最好解（OBK、ACO_AP、DPSO、DDE和GVNS）所组成的当前最好解f^*的对比结果。可以看到，即使与五组解组成的当前最好解相比，BILS算法仍然具有很强的竞争力。对于120个算例，BILS算法和当前最好解f^*比起来，改进了34个算例的当前最好解，只有4个算例未能改进。从表3.4可以推断出，BILS算法在解的质量方面胜过之前的优秀启发式算法。这个结果充分显示出BILS算法在寻找最好解时的有效性。

为了更深入的了解3个参数(τ, R, η)对问题算例的影响，在表3.5中总结了对于这3个参数取值的每一种组合，BILS算法对几种优秀启发式算法的改进率（例如，ΔOBK = (BILS$-$OBK)/OBK$\times$100%）。表中列出了BILS算法对于每一种参数组合中的10个算例及总共120个算例，相对于精确算法的最优解以及优秀启发式算法的最好解的改进率的平均值。第2-7列给出的是BILS算法分别相对于OPT、OBK、ACO_AP、DPSO、DDE和GVNS的平均改进率。

从表3.5可以看出，和启发式算法结果相比，第2行，即当$\tau = 0.3$、$R = 0.25$、$\eta = 0.75$时，BILS算法取得的改进最大。并且，对于不同的组合中，当τ和R的值固定时，η的取值越大，BILS算法取得的改进就越大。总的说来，表3.5中的前4行，

表 3.5 BILS算法相对于其它相关算法的平均改进率

τ，R，η	ΔOPT	ΔOBK	ΔACO_AP	ΔDPSO	ΔDDE	ΔGVNS
$\tau=0.3, R=0.25, \eta=0.25$(001-010)	0.05%	-13.66%	-10.48%	-9.57%	-3.47%	-2.64%
$\tau=0.3, R=0.25, \eta=0.75$(011-020)	0.99%	-46.26%	-30.25%	-27.15%	-19.05%	-6.02%
$\tau=0.3, R=0.75, \eta=0.25$(021-030)	0.16%	-12.73%	-12.62%	-2.59%	-2.52%	-1.60%
$\tau=0.3, R=0.75, \eta=0.75$(031-040)	0.00%	-10.00%	-10.00%	-10.00%	-10.00%	-10.00%
$\tau=0.6, R=0.25, \eta=0.25$(041-050)	0.00%	-1.86%	-1.10%	-0.48%	-0.25%	-0.19%
$\tau=0.6, R=0.25, \eta=0.75$(051-060)	0.00%	-6.79%	-3.64%	-2.05%	-1.06%	-0.76%
$\tau=0.6, R=0.75, \eta=0.25$(061-070)	0.00%	0.81%	-0.02%	0.00%	0.00%	0.00%
$\tau=0.6, R=0.75, \eta=0.75$(071-080)	0.04%	-6.16%	-3.64%	-0.69%	-0.41%	-0.17%
$\tau=0.9, R=0.25, \eta=0.25$(081-090)	0.00%	-0.33%	-0.06%	-0.02%	-0.01%	0.00%
$\tau=0.9, R=0.25, \eta=0.75$(091-110)	0.00%	-1.45%	-0.74%	-0.53%	-0.22%	-0.12%
$\tau=0.9, R=0.75, \eta=0.25$(101-110)	0.00%	-0.19%	-0.09%	-0.03%	0.00%	0.00%
$\tau=0.9, R=0.75, \eta=0.75$(111-120)	0.00%	-0.91%	-0.69%	-0.18%	-0.10%	0.01%
平均	0.10%	-8.29%	-6.11%	-4.44%	-3.09%	-1.79%

BILS算法相对于启发式算法取得的改进较大。从表中的最后一行可以看出，对于这120个算例，BILS算法相对于OBK取得的改进率最大为8.29%，改进最小的是相对于GVNS的1.79%。由表3.5可见，BILS算法相对于OPT只有0.1%的偏差率，要注意的是BILS算法的运行时间是限制在100秒以内的，而Tanaka和Araki的精确算法的运行时间最高达到30天。表3.5的结果表明BILS算法与优秀的启发式算法相比，在解的质量上也有较大的提高。

3.4 分析和讨论

在这一节中，分析和讨论BILS算法最关键的两个部分：块移动的邻域结构和快速增量评估技术。

3.4.1 块移动邻域结构的重要意义

由当前文献所知，块移动（Block Move）的邻域结构是由本研究在解决$1|s_{ij}|\sum w_jT_j$问题时首次提出来的。先前对$1|s_{ij}|\sum w_jT_j$问题的研究中也用到一

些其它的邻域结构，例如插入（insertion）、边插入（edge-insertion）和交换（swap）[83]。为了证明块移动邻域结构的优越性，本文设计了两组实验来对比块移动邻域结构和插入以及边插入邻域结构的表现。第一组实验是在单调下降局部搜索算法中分别采用这三种不同的邻域结构来进行对比；另一组实验则是在BILS算法中的局部搜索过程中分别采用这三种不同的邻域结构来进行对比。

第一组实验方案：在单调下降局部搜索算法中，邻域结构分别使用块移动、插入、边插入，且每次搜索过程中，选择邻域解里最好的解。用120个算例对这三种单调下降局部搜索算法分别进行测试（共3×120组），每组测试进行50次，停止标准为5秒CPU时间，当完成一次局部搜索且运行未结束时，随机生成一个初始解继续运行。

图3-8显示了在单调下降局部搜索算法中分别使用块移动、插入、边插入这三种邻域结构的情况下，每一个算例运行50次后的目标函数值的平均值及其95%的置信区间。图中一共包括12个子图，每个子图由10个算例的结果组成。为了对比这三种不同邻域结构，将块移动、插入和边插入的目标函数值的平均值进行了t检验，差异具有统计学意义。需要注意的是，对于这三种不同的邻域结构，图3-8中的每一个目标函数值的平均值都被函数$\frac{\bar{x}-OPT}{x_{max}-OPT}$进行了归一化，其中$\bar{x}$表示目标函数值$x$的平均值，$x_{max}$ 表示x 所有取值中的最大值。可以看到，几乎对于所有的算例，块移动的目标函数值要明显优于另外两种邻域结构。这个结果高度说明了本文提出的块移动邻域结构相对于插入和边插入的优越性。

第二组实验方案：保持BILS算法的其它部分不变，算法中局部搜索过程中的邻域结构分别采用块移动、插入和边插入。用120个算例分别对三种BILS算法进行测试（共3×120组），每组测试都被执行50次，每次的停止标准为100秒的CPU时间。记录在三种不同的邻域结构条件下，每一个算例运行50次后的目标函数值的平均值。

图3-9显示了在BILS算法中分别采用块移动、插入、边插入这三种不同的邻域结构时，每一个算例运行50次后的目标函数值的平均值及其95%的置信区间。为了对比这三种不同邻域结构的效率，将块移动、插入和边插入的目标函数值的平均值进行了t检验，结果表明差异具有统计学意义。同样，对于这三种不同的邻域结构，

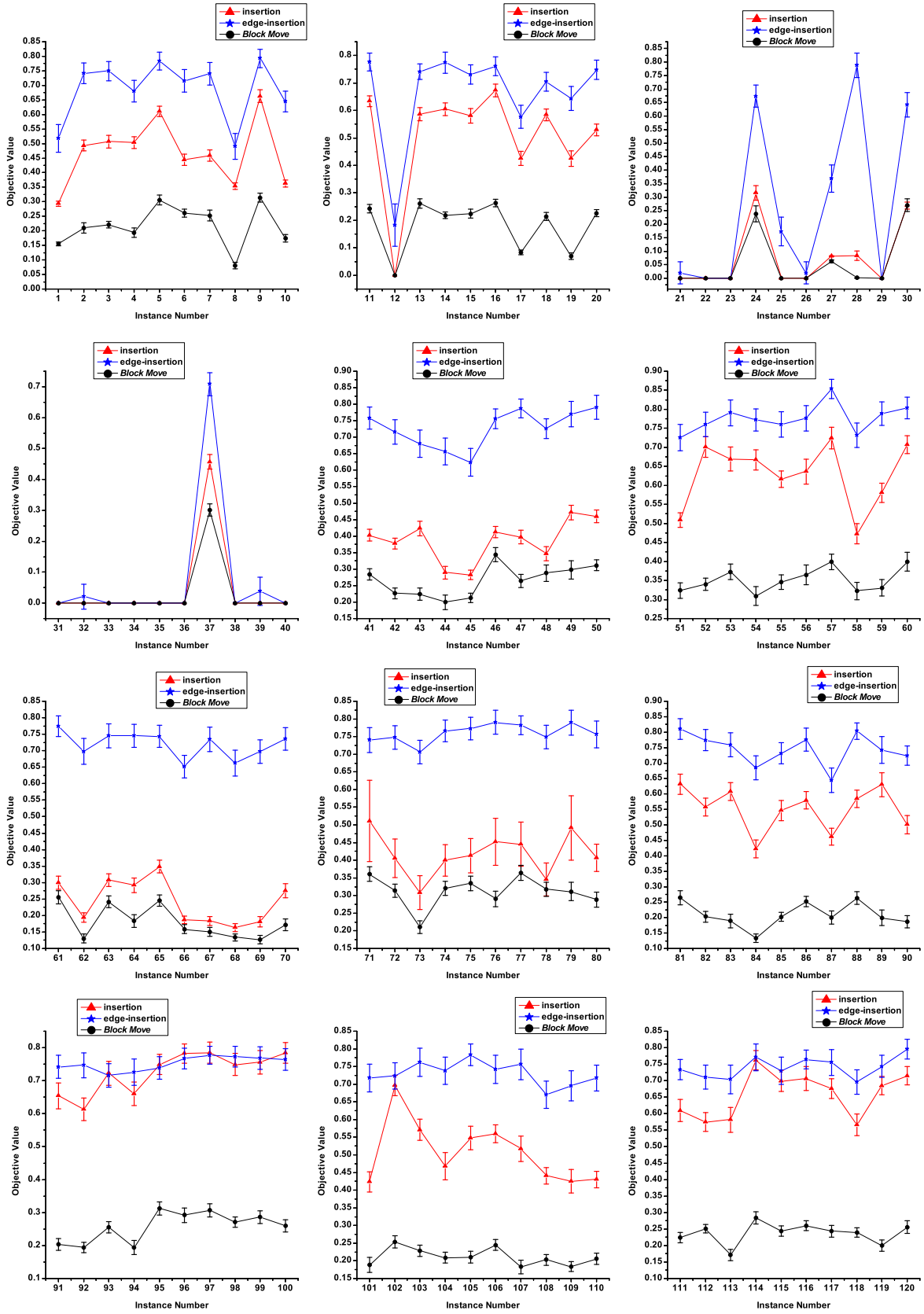

图 3-8 三种邻域结构在单调下降局部搜索算法中的对比（95%的置信区间）

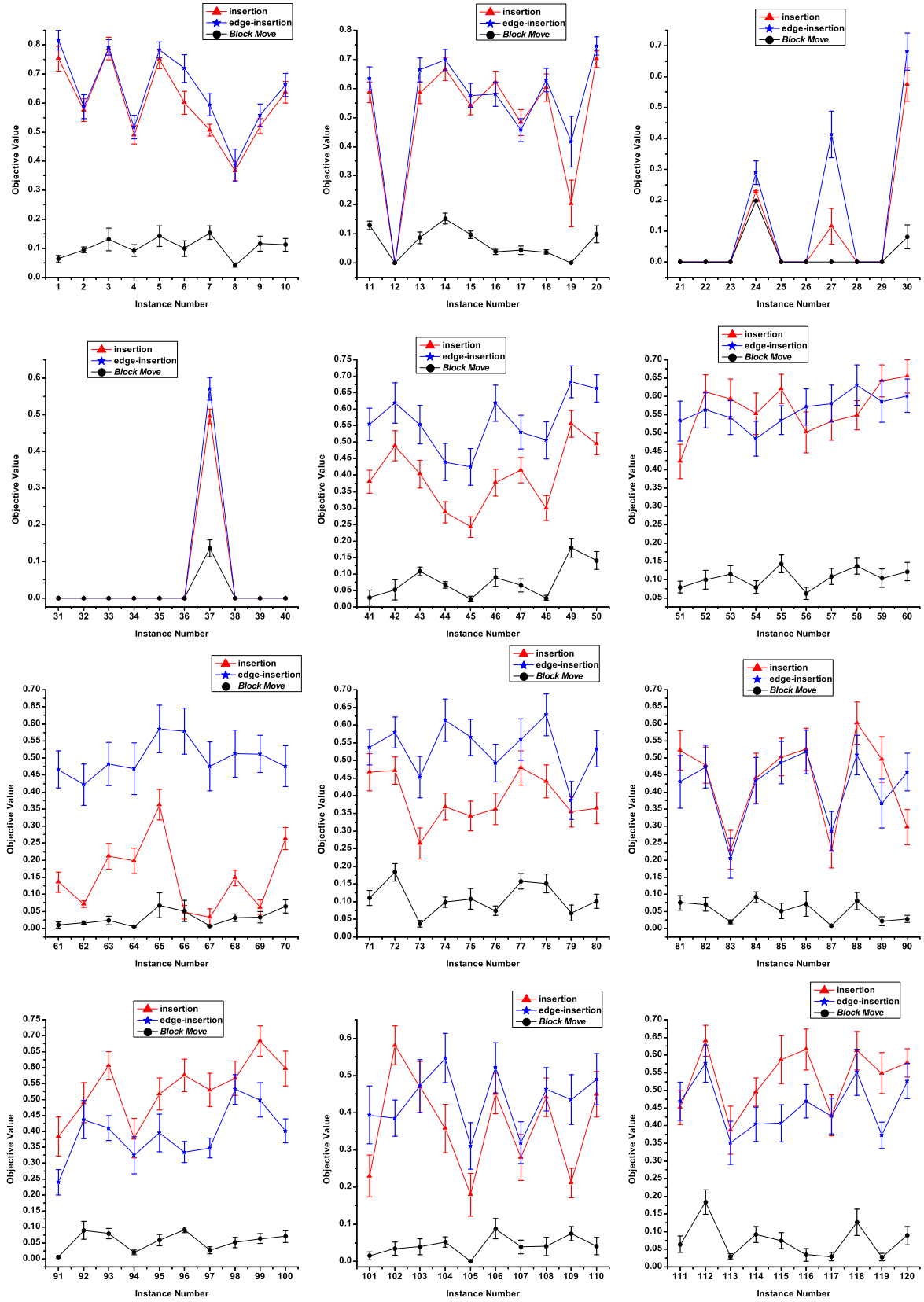

图 3-9 三种邻域结构在BILS算法中的对比（95%的置信区间）

图3-9中的每一个目标函数值的平均值都被函数$\frac{\bar{x}-OPT}{x_{max}-OPT}$进行了归一化，其中$\bar{x}$表示目标函数值$x$的平均值，$x_{max}$ 表示x 所有取值中的最大值。从图3-9可以看出，几乎对于所有的算例，块移动的目标函数值要明显优于另外两种邻域结构。这个结果高度说明了本文提出的块移动邻域结构在BILS算法中的优秀表现。

从图3-8和3-9可以明显的看出，在以上两组关于块移动和插入以及边插入的对比测试中，块移动邻域结构的表现都要明显优于插入和边插入这两种邻域结构。这个结果高度说明了本文新提出的块移动邻域结构的有效性和优越性。同样也表明，对于一个具体问题来说，邻域结构的定义对启发式算法的表现有着非常重要的影响。

3.4.2 快速增量评估技术的效率

Tasgetiren等人[82]设计了一种用于评估插入邻域结构的加速评估方法，这也是他们对1$|s_{ij}|\sum w_jT_j$ 问题的一个创新型的贡献。他们提出的加速评估方法包括两个步骤：（1）将当前加工序列分成两部分，一部分由$N-1$个工件组成，另一部分是剩下的单个待插入的工件；（2）把这单个工件插入到由$N-1$个工件所组成的加工序列的可行位置上。当评估一个插入动作时，首先，计算$N-1$个工件组成的加工序列的目标函数值；然后，分成三种情况来考虑如何计算插入后的目标函数值，这三种情况分别是，把这单个工件插入到$N-1$个工件组成的加工序列的头部、尾部和中间位置。接下来，按照不同的情况，分别计算由于插入动作而产生的目标函数值的偏差值。

为了将本文提出的快速增量评估技术与Tasgetiren等人[82]设计的加速评估方法进行对比。本文将Tasgetiren等人设计的加速评估$N_{insertion}$邻域结构的方法加以扩展用来评估$N_{Block\ Move}$（称为扩展的加速评估方法），它的思想大体上和Tasgetiren等人的加速评估思想相同，差异在于块移动操作后，除了计算被影响工件的偏差值外，还要更新块内部的工件信息。扩展的加速评估方法的过程见图3-10。

本文提出的快速增量评估技术是通过块移动操作后受影响工件的加权延迟增量来计算目标函数值。它的主要思想在于利用图3-6 中的Δ_1、Δ_2、Δ_3这3个Δ值来计算在执行块移动操作后受影响工件的加权延迟增量值。

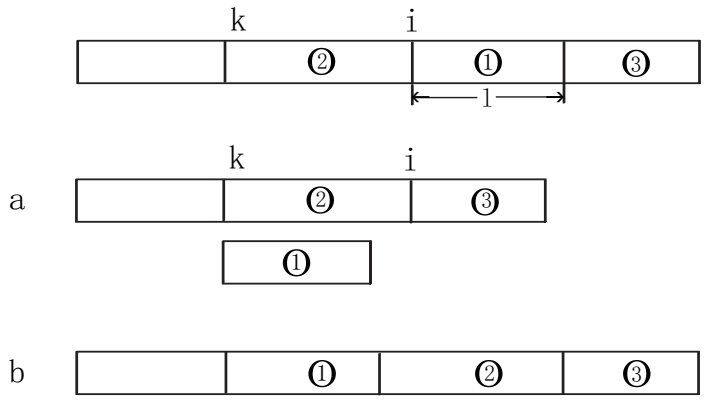

图 3-10 扩展的加速评估方法

为了证明本文提出的快速增量评估技术的高效性，设计了如下实验：保持BILS算法的其它部分不变，在评估邻域解的时候分别采用三种方法，即本文提出的快速增量评估技术、扩展的加速评估方法、直接重新计算目标函数值。由实验结果得知，采用本文新提出的快速增量评估技术时，每秒可以评估2500,000个邻域解；采用扩展的加速评估方法时，每秒钟可以评估1300,000个解；不采用任何加速评估机制，直接计算加工序列的目标函数值，每秒只能评估600,000个解。这个实验结果显示出了本文提出的快速增量评估技术在解决$1|s_{ij}|\sum w_jT_j$ 问题时是非常有价值的。

通过分析可知，本文提出的快速增量估技术之所以有效的主要原因在于：首先，快速增量评估技术并没有把块从当前的加工序列中分离出去，因此，也不用计算由$N-l$个工件组成的加工序列的目标函数值。其次，因为在大多数情况下，$|\Delta_1| \approx |\Delta_2| >> \Delta_3$，受块移动操作影响的工件数比较少，尤其是块③。因此，用来评估这些受影响工件的加权延迟增量值的计算量也很小（见算法2中的14行和20行）。

3.5 本章小节

迭代局部搜索算法是一种高效的求解组合优化问题的方法，本章提出一种新的迭代局部搜索算法BILS来求解$1|s_{ij}|\sum w_jT_j$ 问题。

第一节介绍了迭代局部搜索算法的基本原理，包括初始解的产生、邻域结构的设计、邻域解的评估策略、扰动机制、接受标准、停止标准的设计，以及迭代局部搜索算法的框架。

第二节提出一种新的迭代局部搜索算法BILS求解$1|s_{ij}|\sum w_jT_j$ 问题。分析了当前文献中常用的几种邻域结构，提出了一种新的邻域结构$N_{Block\ Move}$，并通过实验确定了$N_{Block\ Move}$ 中块长和移动步长的取值范围；提出一种评估块移动的邻域解的快速增量评估技术；设计相应的扰动机制，并通过实验确定了扰动强度的取值。

第三节用$1|s_{ij}|\sum w_jT_j$ 问题的120个公共算例对BILS算法进行测试，在100次100秒的测试中找到了113个算例的当前最优解。对没找到最优解的7个算例放宽测试时间，也都找到了最优解。对于算例18和24，BILS找到最优解分别需要8586.6秒（≈2.4小时）和150169.22秒（≈41.7小时），而Tanaka和Araki的精确算法找到最优解则分别需要两个星期和30天。与当前国际上优秀的启发式算法的最好解OBK、ACO_AP、DPSO、DDE和GVNS相比，改进的解的个数分别为94、84、65、52和44，并且在解的质量上也有较大的提高。

第四节着重分析和讨论BILS算法最关键的两个部分：块移动的邻域结构和快速增量评估技术。

4 混合进化算法求解单机调度问题

在对启发式算法的研究过程中，人们发现将几种启发式算法进行有机的组合能获得更强的搜索能力，混合算法的研究已经成为启发式优化算法发展的热点和趋势[91,92]。混合算法集合了所混合的几种算法的优点，克服了各自的一些局陷性，通过融合协作，在求解组合优化问题中取得了非常好的效果[93]。本章将局部搜索算法和进化算法混合起来，用混合进化算法（Hybrid Evolutionary Algorithm，HEA）来求解单机调度问题。

4.1 混合进化算法的基本理论

混合进化算法，顾名思义，就是将进化算法（Evolutionary Algorithm）与其它启发式优化算法相混合，发挥各自的优势，扬长避短，互相协作，以获得更强的求解能力。对于很多问题，进化算法已经可以满足其求解的需要，但是，对于文献[94–97] 中提到的某些问题，直接使用进化算法已经不能找到合适的解了，这无疑为将进化算法和其它启发式算法进行混合提出了需求。将进化算法和其它算法进行混合，简单说来有以下几个主要原因：提高进化算法的性能（例如，收敛的速度）；提高进化算法所得解的质量；激发出参与混合的几种启发式算法的潜力[98]。

混合进化算法通过将进化算法和多种启发式算法进行混合而构成。常用来和进化算法进行混合的一些算法包括：进化算法，也就是说可以将两种不同的进化算法进行混合（例如，遗传规划可以用来提高遗传算法的效率）；神经网络（Neural Network）；模糊逻辑（Fuzzy Logic）；粒子群优化算法（Particle Swarm Optimization）；蚁群算法；细菌觅食优化（Bacterial Foraging Optimization）；其它启发式算法（例如，局部搜索、禁忌搜索、模拟退火、爬山法、动态规划、贪婪随机自适应搜索过程等）[93]。需要说明的是，由进化算法和局部搜索算法混合所组成的混合进化算法又被称为Memetic算法。

进化算法的繁荣发展是从20世纪80年代开始，算法研究者们受达尔文《物种起源》一书中新物种进化思想的启发，设计出的一类算法。进化算法经过三十多年的

发展，主要包括遗传算法（Genetic Algorithms）、进化策略（Evolution Strategies）、进化规划（Evolutionary Programming）和遗传规划（Genetic Programming）这四种典型方法。虽然这些概念的内涵有一定的差别，有着各自不同的侧重点，有不同的生物进化背景，强调了生物进化过程中的不同特性，但本质上都是基于进化思想的，都能产生鲁棒性较强且适应面较广的计算机算法，因此又称它们为进化算法或进化计算。在设计进化算法时所涉及的主要组成部分包括：编码、种群初始化及停止标准、选择策略、繁殖策略、更新策略等。

4.1.1 编码、种群初始化及停止标准

一、编码

算法不能直接处理实际问题中的数据，所以必须将问题中的数据转换成算法能处理的形式，这种转换过程称为编码。在进化算法中，编码方法的选择对于进化过程，尤其是对交叉算子的性能有很大的影响。

编码策略评估规范包括以下三条[99]：

（1）完备性（Completeness）：问题空间中所有候选解都能成为编码集合中的元素。

（2）健全性（Soundness）：编码集合中的元素能对应问题空间的所有候选解。

（3）非冗余性（Nonredundancy）：编码集合中的元素和问题空间的候选解之间是一一对应关系。

这三条规范是独立于问题的普遍准则，因此，对于具体问题应该客观比较和评估该问题领域中的编码方法，由此得到更好的方法和策略。

常用的编码技术：

二进制编码：用0和1构成的二进制位串来表示问题空间的数据。合取范式可满足性问题、最大团问题都是采用二进制编码的典型例子。

排列编码：也叫序列编码，是针对一些特殊问题的特定编码方式，该编码方式将有限集合中的元素进行排列，若有N个元素则有$N!$种排列方法。采用排列编码的典型例子有调度问题和旅行商问题。

根据所处理问题的不同，还有很多种编码技术，例如：格雷码编码、二倍体编码、DNA编码、实数编码、符号编码、混合编码或多参数编码、二维染色体编码或

矩阵编码、树结构编码等。

二、初始种群的产生

局部搜索算法是基于单个初始解的启发式算法，它从一个初始解出发，进行搜索、寻优。而进化算法则是基于群体初始解的启发式算法，它从一组解（初始种群）出发，进行相关操作，迭代寻优。由于种群里的解具有很大的多样性，因此基于群体初始解的启发式算法属于更具开发性（Exploration）的算法，而基于单个解的启发式算法属于更具探索性（Exploitation）的算法。进化算法中保持种群的多样性是非常重要的一个设计原则，如果初始种群不具有较好的多样性，将导致算法过早的收敛。因此，种群初始化对进化算法的效率和有效性有着很大的影响。初始种群的产生主要考虑两方面的问题：种群规模的设定以及种群产生的方式。

种群规模设定的原则。种群中个体数n越大，种群的多样性就越好，算法陷入局部最优解的危险就越小。因此，从考虑种群多样性及不陷入局部最优解出发，n越大越好。但是，n值太大，进化算法的时间复杂度也会呈线性增长，降低收敛速度；另一方面，n值太小，算法搜索的空间容易被限制在一个较小的范围，容易陷入局部最优解。n值的具体大小应该通过对实际问题的难度进行分析来确定，在解的质量和搜索时间之间找到一个折衷点。一般，根据经验在实际应用中，n的取值为几十到几百。

种群产生的方式可采用如下策略[100]:（1）以问题的特点为基础，对其进行分析，设法找出可能出现最优解的区域在整个问题空间中的分布，在此分布区域内产生初始种群。（2）随机产生一些个体，从中选出最好的个体添加到初始种群中，不断执行这个过程，直到个体数达到预先确定的种群规模。

三、停止标准

混合进化算法是一个反复迭代的过程，每次迭代期间都要执行目标函数值的计算、选择、复制、交叉、突变等操作，直到满足停止标准。

混合进化算法常用的停止标准有三种[101]：

（1）预先设定最大迭代次数或CPU计算时间，一旦达到预先设定的最大迭代次数或CPU时间，则算法终止。

（2）设定最小偏差δ。对已知目标函数值的算法，可用最小偏差δ作为终止条件，即：$|f^* - f'| \leqslant \delta$，其中$f'$为已知目标函数值，$f^*$为当前最好解。

（3）观察目标函数值的趋势。初期最好个体和种群的目标函数值都较差，随着选择、复制、交叉、突变等操作，目标函数值逐渐变好，后期目标函数值往好的方向发展的趋势逐渐缓和甚至停止，当发展停止，则终止算法。

4.1.2 选择策略

如何从种群中选择双亲来繁殖后代，是选择算子应该完成的任务。选择算子是进化算法中的重要组成部分之一。选择策略的准则是，种群中越好的个体，被选择作为双亲的机率就越大，这样才会使种群产生出更好的个体。然而，种群中最差的个体也不应该被直接丢弃，它们也要有机会被选中作为双亲，这样有利于维持种群的多样性。比较常用的选择策略有轮盘赌选择、随机遍历抽样、锦标赛选择等。

一、轮盘赌选择（Roulette Wheel Selection）。由于选择策略的准则是保证优秀的个体更容易被选中，那么，在轮盘赌选择中，需要对种群中的每个个体赋一个适应度（Fitness）函数值。适应度函数值的大小代表了个体的优劣，适应度函数值大的个体则较优秀，被选中的机率也大，反之，适应度函数值小的个体则较差，被选中的机率也小。那么，如何给每个个体选择合适的适应度函数值就变得非常重要了。常用的适应度函数的选取有以下两种：

（1）直接将每个个体的目标函数转换成适应度函数。

若目标函数$f(x)$是最大化问题，则适应度函数$fit(x)=f(x)$；若目标函数$f(x)$是最小化问题，则适应度函数$fit(x)=-f(x)$。

（2）将每个个体的目标函数值做适当的处理后，再转换成适应度函数。若目标函数$f(x)$是最大化问题，则适应度函数

$$fit(x)=\begin{cases} f(x)-c_{min}, f(x)>c_{min} \\ 0, otherwise \end{cases} \quad \text{(式 4.1)}$$

式中c_{min}为$f(x)$的最小值估计。若目标函数$f(x)$是最小化问题，则适应度函数

$$fit(x)=\begin{cases} c_{max}-f(x), f(x)<c_{max} \\ 0, otherwise \end{cases} \quad \text{(式 4.2)}$$

式中c_{max}为$f(x)$的最大值估计。

在计算完每个个体的适应度函数值之后，就可以计算出每个个体被选中的概率了。假设p_i是种群中的一个个体，fit_i是种群中个体p_i的适应度函数值，那么，个

体p_i 被选中的概率是$fit_i/(\sum_{j=1}^{n} fit_j)$ 。每一次轮盘选择就是生成一个(0,1)的随机数，由该随机数的位置来决定哪一个个体被选中。根据需要选择的双亲个数决定进行几次轮盘赌选择，每次轮盘赌选择选中一个个体。

如果种群中有非常优秀的个体，那么这些优秀的个体在每一次轮盘赌选择中被选中的概率非常大，将会导致种群过早的收敛，失去种群的多样性。然而，如果种群中每个个体的适应度函数值相当，也会影响优秀的个体被选中的机率。

二、随机遍历抽样（Stochastic Universal Sampling）[102]。随机遍历抽样，前面的步骤和轮盘赌选择一样，都是先给种群中的每个个体赋上适应度函数值。优秀个体的适应度函数值大，被选中的机率就高。随机遍历抽样跟轮盘赌选择区别在于，不论需要选择几个双亲，都只需要进行一次随机遍历抽样。假设需要生成m个双亲，也只用进行一次随机遍历抽样，即生成一个0到1之间的随机数，该随机数所在的位置指示第一个被选中的个体，选择m 个双亲就存在有m 个指针，每个指针之间的距离相等，为$1/m$。

三、锦标赛选择（Tournament Selection）。同样，在锦标赛选择中，还是先给种群中的每个个体赋上相应的适应度函数值。一次锦标赛选择的流程是，首先，从种群中随机选择k 个个体，这个k 为锦标赛的小组规模；然后，从k 个个体中找出适应度函数值最大的个体。这样，一次锦标赛选择就完成了，一次选中一个个体。根据需要选择的双亲个数决定进行几次锦标赛选择。

4.1.3 繁殖策略（交叉算子和变异算子）

进化算法中繁殖策略是产生新个体的手段。繁殖策略的设计须满足可行性、特征继承、完全性和非冗余等标准。繁殖策略包括交叉算子、变异算子，交叉算子产生新的子代，变异算子使得种群保持多样化。

常用的交叉操作有以下几类[100,101,103]：

（1）单点交叉。随机选择一个交叉点，然后对交叉点后面的字符串进行交换。

（2）两点交叉（Two-point Crossover）。随机选择两个交叉点k_1和k_2，$1 < k_1 < k_2 < n$ ，其中n为序列长度，交换双亲中位于(k_1, k_2) 之间的字符串，产生两个后代。

（3）多点交叉（Multi-point Crossover）。随机选择k个交叉点($k > 2$)，则双亲被分为$k+1$段，交换双亲奇数（偶数）段位置上的字符串，产生两个后代。

（4）多点杂乱交叉（Shuffle Crossover）。这种方法是在多点交换的基础上，将原来的序列打乱，以增加各字符相互交换的概率。

（5）均匀交叉。这种方法对双亲个体的每个字符都进行是否交换的选择。

变异是对单个个体进行较小的改动而产生，目的是为了保持群体的多样性。进化算法中变异方法有交换变异、插入变异和逆转变异等。在早期，变异在进化算法中并没有得到太大的重视，变异概率p_m的取值很小，大约0.005左右[101]。实践表明，虽然变异没有交叉在产生个体时重要，但它也是一种有效的方法，特别是在种群质量提高以后，变异的作用就更加明显了。

4.1.4 更新策略

更新策略关注的是如何来实现优胜劣汰的过程，即种群中的哪些个体应该被淘汰，以及新生成的孩子是否应该更新种群中已有的个体。在进化过程中，不断有新的孩子产生，然而，种群的大小又是一个常量，那么必须遵循一定的策略来淘汰一些个体，以保持种群大小的稳定和可持续发展。常用的更新策略包括[27]：

（1）整代更新（Generational Replacement）。在整代更新策略里，需要被更新的是种群里的所有个体，用新生成的孩子将种群中的个体全部替换掉。这种更新策略在Holland提出的经典遗传算法中被采用。

（2）稳态更新（Steady-State Replacement）。在每一次的进化过程中，双亲只生成一个孩子，只用更新种群中的一个个体。比如，用生成的孩子替换掉当前种群中最差的一个个体。

除了上述两种更新策略之外，还有一些其它的策略，例如，只更新当前种群中的λ个个体($1 < \lambda < n$)，n为种群中个体的数量。按照这种策略，更新后的种群往往是由当前种群和生成的孩子中的一些最好解组成。这样，会导致算法收敛过快或过早收敛。出于对种群多样性的考虑，有时候一些较差的个体也应该适当被保留下来。

4.2 求解单机调度问题的混合进化算法

进化算法和局部搜索算法相结合的混合算法，又称为Memetic算法，被认为是一种高效的解决大规模约束满足问题和优化问题的方法[104–109]。在混合进化算法中，只有联合更具多样性的再生搜索和更具集中性的局部搜索，才能在开发和利用搜索空间的过程中达到一个更好的平衡。本节提出一种新的混合进化算法来求解1$|s_{ij}|\sum w_jT_j$和1$|s_{ij}|\sum T_j$ 问题。

在混合进化算法中，局部搜索过程、交叉算子、种群更新策略的选择是非常重要的，它直接影响着混合进化算法寻找全局最优解的速度和性能。本文提出的求解单机调度问题的混合进化算法的总体结构见算法3，它主要由四个部分组成：种群初始化、局部搜索过程、交叉算子和种群更新策略。HEA从一个随机生成的初始种群出发，用局部搜索过程把种群中的每个个体优化达到局部最优（第4-6 行）；然后，用交叉算子来生成新的孩子解x^0（第10 行），并且将x^0用局部搜索过程进行优化；随后，用种群更新策略判定优化后的x^0是否应该替换种群里的解，以及种群里的哪一个解应该被替换（第15行）。在接下来的内容中，HEA的四个组成部分会被详细的介绍。

4.2.1 初始化种群

在本文提出的HEA算法中，首先随机生成一组加工序列构成初始种群，然后使用局部搜索过程对初始种群中的每个个体进行优化。使用Lee 等人提出的ATCS[63]规则来生成初始种群也是一种很好的选择。但是，本文选择随机生成初始种群，因为与用ATCS构造算法生成的初始种群相比，随机生成初始种群的方法具有更好的多样性。

4.2.2 局部搜索过程

在HEA算法中，采用的是单调下降局部搜索来做为局部搜索过程。在每一次搜索过程中，选择当前邻域中最好的那个解直到不能再被改进为止。邻域结构采用的是块移动的邻域结构$N_{Block\ Move}$，即当前解的邻域是由将当前序列中l个连续的工件（称为块）移动到序列的其它位置而产生的，这个动作用三元式$Block\ Move(i,k,l)$来

Algorithm 3 本文提出的求解$1|s_{ij}|\sum w_jT_j$问题的混合进化算法

Input: *J*, s_{ij}
Output: the best job sequence x^* found so far
$P = \{x^1, \ldots, x^p\} \leftarrow$ Population_Initialization()
for $i = \{1, \ldots, p\}$ **do**
  $x^i \leftarrow$ Local_Search(x^i)
end for
$x^* = argmin\{f(x^i)|i = 1, \ldots, p\}$
repeat
  randomly choose two individuals x^j and x^k from P
  $x^0 \leftarrow$ Crossover_Operator(x^j, x^k)
  $x^0 \leftarrow$ Local_Search(x^0)
  if $f(x^0) < f(x^*)$ **then**
    $x^* = x^0$
  end if
  $\{x^1, \ldots, x^p\} \leftarrow$ Population_Updating($x^0, x^1, \ldots, x^p$)
until the stop criterion is met

表示。据我们所知，块移动的邻域结构是本研究在求解$1|s_{ij}|\sum w_jT_j$问题时首次提出的一种新的邻域结构。对于N个工件的加工序列，块移动邻域的大小可达到$O(N^3)$。更多详细的关于新提出的块移动邻域结构，以及对块和步长的取值范围如何进行限制以减少搜索时间但仍保持邻域的高效性的技术可参见本文3.2.3节。同时，本文还提出了一种高效的快速增量评估技术来评估块移动的邻域解，快速增量评估技术的详细描述见本文3.2.3节。

4.2.3 交叉算子

HEA算法中，交叉算子用来生成新的子代个体，交叉算子的设计须满足可行性、特征继承、完全性和非冗余等规则。一般说来，好的交叉算子不仅可以生成和双亲差异比较大的子代，同时还能够让子代继承到双亲足够多的优秀基因。文

献中介绍过的交叉算子有很多种，例如，Linear Order Crossover（LOX）、Position Based Crossover Operator（PBX）、Partially Mapped Crossover Operator（PMX）、Order Crossover（OX）、Cycle Crossover Operator（CX）、Order Based Crossover Operator（OBX）等。通过对这些交叉算子的研究，本文提出一种新的基于公共块（Common Blocks）的交叉算子，块顺序交叉算子（Block Order Crossover Operator，BOX）。下面着重介绍本文新提出的交叉算子BOX以及文献中介绍过的两种有代表性的交叉算子LOX 和PBX。

一、块顺序交叉算子（Block Order Crossover Operator，BOX）

块顺序交叉算子是本文新提出的一种交叉算子，要介绍块顺序交叉算子需要先引入块和公共块的概念。

定义 4.1： 块（Block）。加工序列π 中连续的长度为l $(2 \leqslant l \leqslant N)$ 的工件集合称为块。

定义 4.2： 公共块（Common Blocks）。对于两个加工序列$x^j = \{\pi_{j1}, \pi_{j2}, \ldots, \pi_{jN}\}$和$x^k = \{\pi_{k1}, \pi_{k2}, \ldots, \pi_{kN}\}$来说，有一些块既存在于$x^j$中，也存在于$x^k$中，那么这些块被称为$x^j$和$x^k$的公共块。

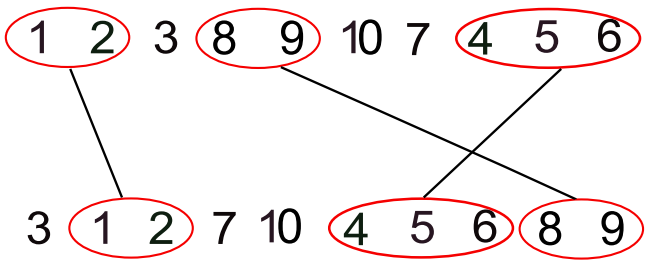

图 4-1 公共块

在图4-1中，以10个工件的加工序列为例来解释公共块的概念。假设有两个已知的加工序列$x^j = \{1, 2, 3, 8, 9, 10, 7, 4, 5, 6\}$和$x^k = \{3, 1, 2, 7, 10, 4, 5, 6, 8, 9\}$。通过比较这两个加工序列包含的相应的块，可以看出，在这两个加工序列中，公共块为{1, 2}、{8, 9}和{4, 5, 6} 。

假设双亲p1和p2是种群中的两个解，孩子c1和c2为双亲p1和p2在交叉算子的作用下生成的两个解。那么，BOX算子的工作步骤描述如下：

Step 1：找到p1和p2所有的公共块，余下的工件存入集合A；

Step 2：把p1和p2的公共块按照它们在p1和p2的位置直接复制到c1和c2的相应位置上；

Step 3：把集合A中的工件按照它们在p2和p1的顺序复制到c1和c2剩下的位置上。

用包含8个工件的加工序列为例来解释BOX算子的交叉过程。假设给定两个双亲p1 = {8, 1, 7, 4, 5, 3, 6, 2} 和p2 = {5, 3, 6, 4, 8, 2, 1, 7} 。首先，找出p1和p2的公共块为{1, 7} 和{5, 3, 6} ，余下的工件为{8, 4, 2} 将其存入集合A，即A = {8, 4, 2} ；然后，把公共块复制到c1和c2中，它们在c1和c2中的位置与在p1和p2 中的位置相同；最后，将集合A 里的工件复制到c1和c2 余下的位置上，集合A中的工件在c1和c2中的顺序和它们在p2和p1的顺序相同。最后得到孩子c1 ={4, 1, 7, 8, 5, 3, 6, 2}和c2= {5, 3, 6, 8, 4, 2, 1, 7} 。这个过程见图4-2。

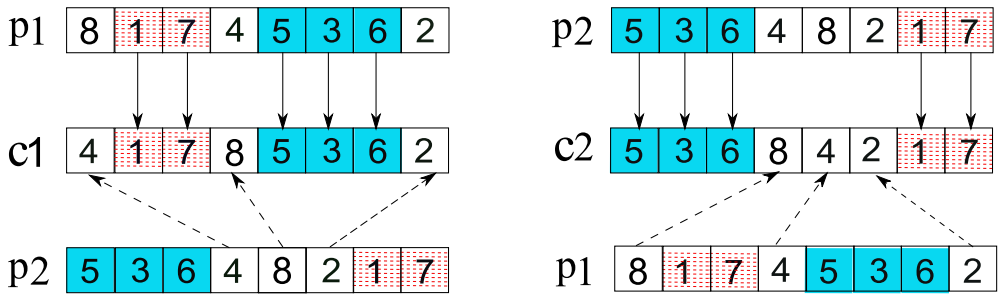

图 4-2 BOX交叉算子

二、线性次序交叉算子（Linear Order Crossover Operator，LOX）

线性次序交叉算子LOX[110]是由Falkenauer和Bouffouix在解决作业车间调度问题的时候提出来的一种交叉算子。LOX算子是对OX的一个改进版本。在LOX中，工件所组成的是一个线形的加工序列，而在OX中，则是一个环形的序列。LOX算子的工作步骤如下：

Step 1：随机在p1和p2上选择两个断点；

Step 2：把p1和p2中两个断点之间的工件按照它们在p1 和p2的位置直接复制到c1和c2的相应位置上；

Step 3：把p1和p2中剩下的工件按照它们在p2和p1的次序依次复制到c1和c2剩下的位置上。

LOX交叉算子的交叉过程如图4-3所示。首先，在p1和p2 上随机选择两个相同的断点，一个断点位于位置3 和4 之间，另一个断点在位置6和7之间；p1和p2中位于两个断点之间的工件分别为{4, 5, 3} 和{4, 8, 2} ，把它们按照当前的位置分别复制到c1和c2中；p1和p2中剩下的工件分别为{8, 1, 7, 6, 2} 和{5, 3, 6, 1, 7} ，把p1和p2 中剩下的这些工件按照它们在p2和p1 中的次序依次放到c1和c2余下的位置上。{6, 8, 2, 4, 5, 3, 1, 7} 和{1, 7, 5, 4, 8, 2, 3, 6} 则是用LOX交叉算子生成的孩子c1 和c2。

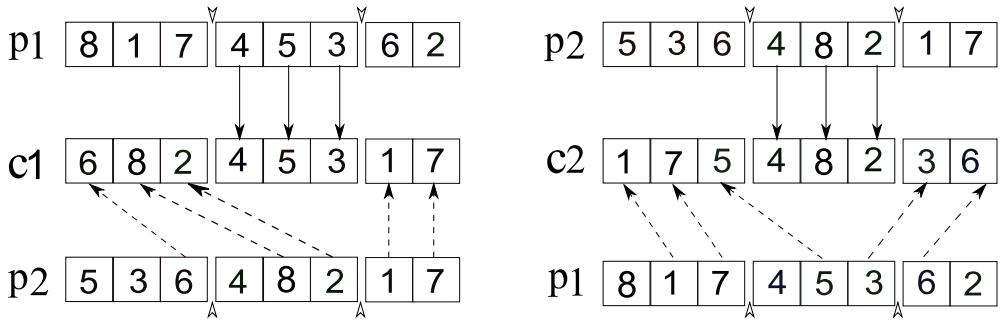

图 4-3 LOX交叉算子

三、基于位置的交叉算子（Position Based Crossover Operator，PBX）

基于位置的交叉算子PBX[111] 最初是由Syswerda提出来的。PBX的工作步骤如下：

Step 1：随机在p1和p2上选择一些位置；

Step 2：把p1和p2中选中位置上的工件按照它们在p1 和p2的位置直接复制到c1和c2的相应位置上；

Step 3：把p1和p2中剩下的工件按照它们在p2和p1 的次序依次复制到c1和c2 剩下的位置上。

图4-4举例说明了PBX算子的交叉过程。在这个例子当中，被选中的位置为1、3、6和7。因此，p1 和p2中被选中的工件分别为{8, 7, 3, 6}和{5, 6, 2, 1}，剩下的工件分别为{1, 4, 5, 2}和{3, 4, 8, 7} ； 把p1和p2 中被选中的工件复制到c1 和c2 相同的位置上，剩下的工件按照它们在p2 和p1 中的次序依次放在c1和c2余下的位置上。这样，生成的孩子分别为c1 = {8, 5, 7, 4, 2, 3, 6, 1} 和c2 = {5, 8, 6, 7, 4, 2, 1, 3} 。

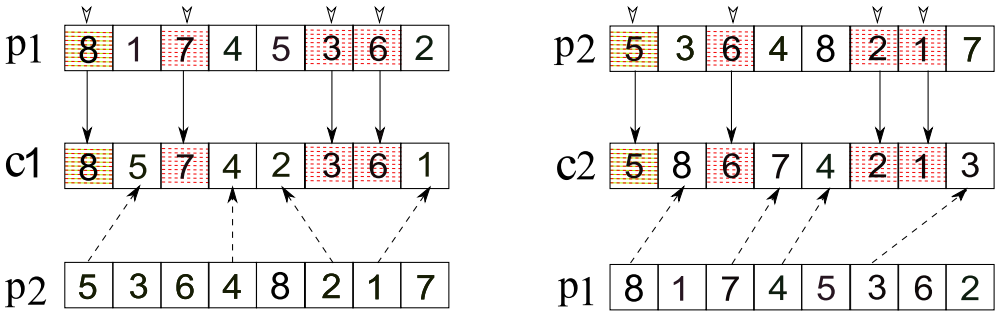

图 4-4 PBX交叉算子

4.2.4 种群更新策略

HEA算法中，新的孩子产生后立即对其使用局部搜索进行优化，选出优秀的孩子作为参与更新种群的备选。然后，就需要判断孩子解是否被添加进种群，以及种群中的哪一个解应该被淘汰。本节介绍两种种群更新策略：一种是在更新种群的时候仅仅以目标函数值来判断；另一种在更新种群的时候根据本文新提出的基于相似性和质量的优度函数（Similarity-and-Quality Based Goodness Function）值来判断。为了在后续工作中描述起来更明确，前一种种群更新策略称为“种群更新策略A”，后一种种群更新策略称为“种群更新策略B”。

一、种群更新策略A

$1|s_{ij}|\sum w_jT_j$ 问题是一个最小化问题，因此目标函数值越大，解的质量越差。在种群更新策略A中，当孩子解的目标函数值小于且不等于种群中最差解的目标函数值时，种群中最差的解则被生成的孩子解替换掉。

二、种群更新策略B

本文新提出的基于相似性和质量的优度函数，即兼顾了目标函数值的优劣又考虑了种群多样性的问题。对于越好的解，其优度值越大，越差的解，优度值则越小。

在种群更新策略B中，如果孩子解的优度值大于种群中优度值最小的个体，就将其替换。基于相似性和质量的优度函数的主要思想是在判断产生的孩子解优劣时，不单只考虑目标函数值的大小，还希望这个孩子解跟种群中的其它解不要太相似，以保证种群的多样性和可持续性。在HEA算法中，是以两个解之间的公共块所包含的工件个数做为相似度的判断标准。为了更清楚的解释两个解之间相似度的概念，

用到了以下定义：

定义 4.3： 两个解之间的相似度。对于两个给定的加工序列$x^i = \{\pi_{i1}, \pi_{i2}, \ldots, \pi_{iN}\}$和$x^j = \{\pi_{j1}, \pi_{j2}, \ldots, \pi_{jN}\}$，它们之间的公共块包含的工件个数称为$x^i$和$x^j$的相似度，用$sim_{ij}$表示。

以图4-2为例，p1和p2的公共块为{1, 7} 和{5, 3, 6}，那么公共块所包含的工件数为5，即p1和p2的相似度等于5。

定义 4.4： 解和种群之间的相似度。已知给定种群$P = \{x^1, x^2, \ldots, x^p\}$和种群中任意两个个体$x^i$和$x^j$ ($i, j = 1, 2, \ldots, p, i \neq j$) 之间的相似度为$sim_{ij}$，那么个体$x^i$和种群$P$的相似度定义为$x^i$ ($i = 1, 2, \ldots, p$) 和种群P 中其它个体相似度值中最大的那一个，记为$S_{i,P}$。

$$S_{i,P} = max\{sim_{ij} | x^j \in P, j \neq i\} \tag{式 4.3}$$

定义 4.5： 解相对于种群的优度值。若种群$P = \{x^1, x^2, \ldots, x^p\}$中任意个体$x^i$ 与种群P的相似度为$S_{i,P}$，那么x^i 相对于种群P 的优度值记为$g(i, P)$，

$$g(i, P) = \beta\widetilde{A}(f(x^i)) + (1 - \beta)\widetilde{A}(S_{i,P}) \tag{式 4.4}$$

其中$f(x^i)$是个体x^i 的目标函数，β 是一个常量参数，$\widetilde{A}(\cdot)$ 表示的是归一化函数：

$$\widetilde{A}(y) = \frac{y_{max} - y}{y_{max} - y_{min} + 1} \tag{式 4.5}$$

其中y_{max} 和y_{min}分别表示种群中所有个体的目标函数$f(x^i)$或相似度$S_{i,P}$的最大值和最小值。

为了确定参数β的取值，将HEA算法中的交叉算子分别采用BOX、LOX 和PBX，更新策略采用种群更新策略B，对β设定9个测试值$\{0.1, \ldots, 0.9\}$。用$1|s_{ij}|\sum w_j T_j$问题公共算例中的一组分别对三种HEA算法在β采用9种不同取值的情况下进行测试，对于β 的每种取值分别测试20次，每次运行100秒。

图4-5描述的是算例7和18对采用三种不同交叉算子的HEA算法测试20次所得目标函数值的平均值。通过观察可知，对于算例7和18，当β 值在0.5到0.7范围内时，

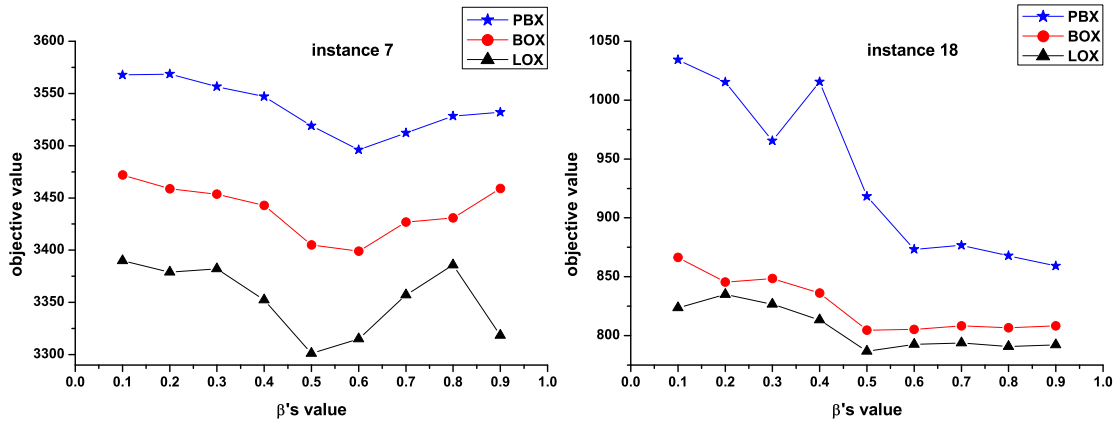

图 4-5 β值的确定

对于三种不同的交叉算子，目标函数值都保持在一个相对稳定且质量较高的状态。需要注意的是，对于其它的算例，也是同样的情况。基于以上的观察，HEA算法中β的取值为区间[0.5, 0.7]里的一个随机数。

需要明确的是，对于种群中的每个个体x^i，它的优度值$g(i, P)$越大表明它的质量越好。本文提出的优度函数同时兼顾了解的目标函数值的优劣以及种群的多样性，即保证了种群中有优质的解，又强调种群的多样性，以避免种群过早的收敛。

HEA算法种群更新策略B的伪码如算法4所示。x^0是被局部搜索优化后的子代解，用下面的规则来决定x^0 是否被添加进种群$P = \{x^1, x^2, \ldots, x^p\}$。首先，把$x^0$临时添加进种群$P$，构成新的种群$P'$（见第3行）。接着计算每个个体$x^i \in P'$ 的优度值（见第4-7 行），并且找到原种群P 中最差的解x^w（见第8 行）。如果孩子解x^0 的优度值要大于原种群P中最差解x^w的优度值，则优化后的孩子解x^0被添加到原种群并替换掉x^w 。

4.3 实验及结果分析

4.3.1 实验方案

首先对比三种交叉算子（BOX、LOX和PBX）和种群更新策略（A和B）所构成的6种组合在HEA算法中处理$1|s_{ij}|\sum w_jT_j$ 问题的120个公共算例时的表现。用$1|s_{ij}|\sum w_jT_j$ 和$1|s_{ij}|\sum T_j$问题的经典公共算例对这6种组合中表现最好的一种组合来进行测试。然后，将实验结果与Tanaka和Araki提出的精确算法的最优解以及其

Algorithm 4 种群更新策略B（基于相似性和质量的种群更新策略）

Input: population $P = \{x^1, \ldots, x^p\}$ and offspring solution x^0
Output: updated population $P = \{x^1, \ldots, x^p\}$
Tentatively add x^0 to population P: $P' = P \cup \{x^0\}$
for $i = \{0, \ldots, p\}$ **do**
    Calculate the similarity between x^i and P' ($S_{i,P'}$) according to Eq. (4.3)
    Calculate the goodness score of x^i ($g(i, P')$) according to Eq. (4.4)
end for
Identify the solution with the worst goodness score in the original population P: $x^w = argmin\{g(i, P')|i = 1, ..., p\}$
if $g(0, P') > g(w, P')$ and $g(0, P') \neq g(i, P'), i = \{1, \ldots, p\}$ **then**
    Replace x^w with $x^0 : P = P \cup \{x^0\} \setminus \{x^w\}$
end if

它一些高水平的启发式算法的最好解进行对比。

HEA算法用VC++编程实现，程序运行环境为Core i3 CPU，主频3.1GHz，内存2GB。

4.3.2 三种交叉算子和两种更新策略组成的6种组合的对比

局部搜索过程、交叉算子、种群更新策略是混合进化算法中最重要的组成部分，本实验将三种交叉算子（BOX、LOX和PBX）和两种种群更新策略（A和B）构成的6种组合（BOX⊕A、BOX⊕B、LOX⊕A、LOX⊕B、PBX⊕A 和PBX⊕B ）进行对比。为了比较这6 种组合在HEA中求解$1|s_{ij}|\sum w_jT_j$问题时的表现，设计了6 组实验，每组实验的HEA程序中的交叉算子和更新策略采用上述6 种组合中的一种（采用6种不同组合的HEA算法分别用BOX⊕A、BOX⊕B、LOX⊕A、LOX⊕B、PBX⊕A、PBX⊕B 来表示）。对于$1|s_{ij}|\sum w_jT_j$ 问题的所有120个算例，6 组实验程序分别以不同的随机种子各自运行50次，每次运行的停止标准为达到100秒的CPU运行时间。

表4.1中，将实验结果和文献[54]中报道的精确算法的结果（OPT）进行了对比统计。表中BOX⊕A、BOX⊕B、LOX⊕A、LOX⊕B、PBX⊕A和PBX⊕B 的最好解和

表 4.1 6种HEA算法的测试结果与最优解的对比统计

	BOX⊕A	BOX⊕B	LOX⊕A	LOX⊕B	PBX⊕A	PBX⊕B
持平解的个数	97	103	111	113	103	100
差的解的个数	23	17	9	7	17	20
所有解的个数	120	120	120	120	120	120

最优解OPT相比，持平解的个数分别为97、103、111、113、103 和100，差的解个数分别为23、17、9、7、17和20。由表4.1可以看出，第5列LOX⊕B与OPT持平的解的个数为113，在6 种组合中是表现最好的。

从表4.1中还可以看出，BOX⊕A 和BOX⊕B与最优解持平的解的个数分别为97和103；LOX⊕A和LOX⊕B与最优解持平的解的个数分别为111和113。就是说，对于BOX 和LOX两种交叉算子来说，当采用种群更新策略B时找到最优解的个数是要优于采用更新策略A的。然而，单从与最优解相比，持平和差的解的个数不足以证明哪一种组合的HEA算法在解决$1|s_{ij}|\sum w_jT_j$ 问题时表现是最优秀的（特别是在结果比较相近的情况下）。

表 4.2 6种HEA算法的最好解相对于OPT的平均差异率

τ，R，η	ΔBOX⊕A	ΔBOX⊕B	ΔLOX⊕A	ΔLOX⊕B	ΔPBX⊕A	ΔPBX⊕B
$\tau=0.3,R=0.25,\eta=0.25$(001-010)	0.45%	0.43%	0.02%	0.02%	1.50%	2.64%
$\tau=0.3,R=0.25,\eta=0.75$(011-020)	2.08%	1.21%	0.39%	0.20%	5.26%	6.02%
$\tau=0.3,R=0.75,\eta=0.25$(021-030)	3.53%	3.53%	3.39%	1.10%	3.53%	3.57%
$\tau=0.3,R=0.75,\eta=0.75$(031-040)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau=0.6,R=0.25,\eta=0.25$(041-050)	0.03%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau=0.6,R=0.25,\eta=0.75$(051-060)	0.26%	0.25%	0.00%	0.00%	0.00%	0.00%
$\tau=0.6,R=0.75,\eta=0.25$(061-070)	0.00%	0.02%	0.00%	0.00%	0.00%	0.00%
$\tau=0.6,R=0.75,\eta=0.75$(071-080)	0.27%	0.12%	0.00%	0.00%	0.00%	0.12%
$\tau=0.9,R=0.25,\eta=0.25$(081-090)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau=0.9,R=0.25,\eta=0.75$(091-110)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau=0.9,R=0.75,\eta=0.25$(101-110)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau=0.9,R=0.75,\eta=0.75$(111-120)	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%
平均	0.55%	0.46%	0.32%	0.11%	0.86 %	1.03%

为了更进一步比较这6种HEA的表现，以及算例的三个参数(τ, R, η) 对结果的影响，表4.2和4.3分别列出了对于参数(τ, R, η) 的12种组合，这6种HEA算法找到的最好解和所有解的平均值相对于最优解（OPT）的差异率（例如，ΔBOX$\oplus$A = (BOX$\oplus$A $-$ OPT)/OPT $\times$ 100%）。在表4.2和4.3中，第2-7列分别给出了6种HEA算法对于(τ, R, η) 的每一种组合中的10个算例的最好解和解的平均值相对于OPT的差异率的平均值以及120个算例总的平均差异率。

表 4.3 6种HEA算法解的平均值相对于OPT的平均差异率

τ，R，η	ΔBOX$\oplus$A	ΔBOX$\oplus$B	ΔLOX$\oplus$A	ΔLOX$\oplus$B	ΔPBX$\oplus$A	ΔPBX$\oplus$B
$\tau = 0.3, R = 0.25, \eta = 0.25$(001-010)	5.02%	2.91%	2.03%	1.14%	6.93%	7.30%
$\tau = 0.3, R = 0.25, \eta = 0.75$(011-020)	6.37%	4.93%	3.09%	2.40%	14.27%	16.64%
$\tau = 0.3, R = 0.75, \eta = 0.25$(021-030)	3.74%	3.64%	3.60%	3.49%	3.69%	3.72%
$\tau = 0.3, R = 0.75, \eta = 0.75$(031-040)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$\tau = 0.6, R = 0.25, \eta = 0.25$(041-050)	0.97%	0.68%	0.43%	0.25%	0.30%	0.25%
$\tau = 0.6, R = 0.25, \eta = 0.75$(051-060)	2.15%	1.62%	1.54%	0.98%	1.05%	0.96%
$\tau = 0.6, R = 0.75, \eta = 0.25$(061-070)	0.75%	0.51%	0.43%	0.28%	0.12%	0.04%
$\tau = 0.6, R = 0.75, \eta = 0.75$(071-080)	4.16%	2.99%	2.70%	1.74%	1.29%	1.11%
$\tau = 0.9, R = 0.25, \eta = 0.25$(081-090)	0.11%	0.08%	0.09%	0.06%	0.03%	0.02%
$\tau = 0.9, R = 0.25, \eta = 0.75$(091-110)	0.44%	0.30%	0.36%	0.29%	0.19%	0.13%
$\tau = 0.9, R = 0.75, \eta = 0.25$(101-110)	0.11%	0.08%	0.11%	0.08%	0.04%	0.02%
$\tau = 0.9, R = 0.75, \eta = 0.75$(111-120)	0.47%	0.33%	0.47%	0.31%	0.20%	0.11%
平均	2.02%	1.51%	1.24%	0.92%	2.34%	2.52%

从表4.2和4.3可以看出，第5列的总平均差异率分别为0.11%和0.92%，比其它五种HEA的总平均差异率要小。在表4.2中，第3列和第5例的总平均差异率分别为0.46% 和0.11%，分别要小于第2列和第4列的总平均差异率。同样，在表4.3中，第3 列和第5例的总平均差异率分别为1.51%和0.92%，分别要小于第2列和第4列的总平均差异率。由表4.1、4.2、4.3 的结果可以推断出，在HEA算法中，对于交叉算子BOX 和LOX，当采用种群更新策略B时的表现要优于采用简单的种群更新策略A。

从表4.1、4.2、4.3 可以得出，不论是从最好解还是平均值的表现来看，LOX$\oplus$B在HEA中的表现要明显优于其它五种混合进化算法。为了验证LOX$\oplus$B算法的结果和

其它五种混合进化算法结果的差异是有统计学意义的，将LOX⊕B和其它5 种HEA算法的解的平均值进行了非参数统计检验，即Wilcoxon符号秩检验，检验结果如表4.4所示。表4.4 中，当$p\text{-}value < 0.05$时，表示LOX⊕B算法和其它五种HEA算法的结果之间存在显著差异。从表4.4可以看出，LOX⊕B 算法明显优于其它其它5种组合的HEA算法。因此，可以推断出，LOX⊕B算法在求解$1|s_{ij}|\sum w_jT_j$问题时与采用其它5种组合的HEA算法相比有非常明显的优势。

表 4.4 LOX⊕B与其它5种HEA算法解的平均值的Wilcoxon符号秩检验结果

	BOX⊕A	BOX⊕B	LOX⊕A	PBX⊕A	PBX⊕B
p-value	$<$ 2.2e-16	1.031e-09	$<$ 2.2e-16	0.0005426	4.5e-06
Diff.?	yes	yes	yes	yes	yes

4.3.3 $1|s_{ij}|\sum w_jT_j$问题的具体计算结果

用$1|s_{ij}|\sum w_jT_j$ 问题的120个公共算例测试LOX⊕B算法，每个算例随机运行100次，每次的运行时间为100秒CPU时间。本节中，用来对比的Tanaka 和Araki的精确算法的最优解（OPT）以及其它五组优秀启发式算法的最好解（OBK、ACO AP、DPSO、DDE 和GVNS）在本文3.3节已经做了介绍，此处就不再赘述；新提出的BILS算法的最好解可参见本文3.3.2 节内容。

需要注意的是，只有Tanaka 和Araki的精确算法[54]和本文提出的BILS算法给出了$1|s_{ij}|\sum w_jT_j$ 问题的120个算例中每个算例找到最好解的具体时间。并且，在上一章中，已经将BILS算法和精确算法的计算效率进行了对比。因此在本节中，对于每个算例只列出BILS算法和LOX⊕B算法找到最好解的计算时间。

本节中的实验旨在评估用LOX⊕B算法处理$1|s_{ij}|\sum w_jT_j$问题时的表现与精确算法的最优解（OPT）、本文提出的BILS算法以及文献中报道的其它五组优秀启发式算法的结果进行对比的情况。表4.5给出了BILS算法和LOX⊕B算法的实验结果，第2列给出的是精确算法的最优解（OPT）；第3-6列给出了BILS 算法的相关实验结果，分别包括BILS 算法找到的最好解（f'_{best}）、程序运行100次的平均值（f'_{aver}）、找到最好解f'_{best}的最短计算时间（t'_{min}）和找到最好解f'_{best}的平均计算时间（t'_{aver}）；第7-10列则是LOX⊕B 算法的实验结果，分别包括找到的最好解（f_{best}）、程序运行100次的平

均值（f_{aver}）、找到最好解f_{best}的最短计算时间（t_{min}）和找到最好解f_{best}的平均计算时间（t_{aver}）；最后3列给出的是f_{best}、f_{aver} 和t_{aver}相对于f'_{best}、f'_{aver}和t'_{aver}的改进率Δf_{best}、Δf_{aver} 和Δt_{aver}（例如，$\Delta f_{best} = (f_{best} - f'_{best})/f'_{best} \times 100\%$）。

表 4.5: LOX⊕B算法求解1$|s_{ij}|\sum w_jT_j$问题的实验结果

算例	OPT	BILS算法				LOX⊕B算法				Δf_{best}	Δf_{aver}	Δt_{aver}
		f'_{best}	f'_{aver}	t'_{min}	t'_{aver}	f_{best}	f_{aver}	t_{min}	t_{aver}			
1	**453**	**453**	480.3	85.7	85.7	**453**	462.2	81.6	81.6	0.0%	-3.8%	-4.8%
2	**4794**	**4794**	4887.0	25.9	59.3	**4794**	4841.5	26.3	69.6	0.0%	-0.9%	17.4%
3	**1390**	**1390**	1457.5	34.1	63.3	**1390**	1401.8	46.3	71.3	0.0%	-3.8%	12.7%
4	**5866**	**5866**	5978.4	8.7	49.6	**5866**	5871.2	7.3	25.7	0.0%	-1.8%	-48.3%
5	**4054**	4074	4215.9	46.6	76.1	**4054**	4096.9	21.2	31.5	-0.5%	-2.8%	-58.6%
6	**6592**	**6592**	6750.3	22.1	48.4	**6592**	6617.2	14.5	50.9	0.0%	-2.0%	5.2%
7	**3267**	**3267**	3404.2	88.4	88.4	**3267**	3319.0	23.9	63.3	0.0%	-2.5%	-28.4%
8	**100**	**100**	106.2	71.7	82.7	**100**	102.1	80.1	92.7	0.0%	-3.9%	12.0%
9	**5660**	**5660**	5840.8	35.3	65.7	**5660**	5699.5	15.6	50.7	0.0%	-2.4%	-22.9%
10	**1740**	**1740**	1793.0	41.6	67.9	**1740**	1756.8	25.5	58.7	0.0%	-2.0%	-13.5%
11	**2785**	2830	3125.0	55.6	55.6	2798	2871.0	47.2	65.1	-1.1%	-8.1%	17.2%
12	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	4.2%
13	**3904**	3942	4141.9	85.2	89.6	**3904**	3986.5	28.1	72.5	-1.0%	-3.8%	-19.1%
14	**2075**	2081	2315.1	54.6	54.6	**2075**	2179.3	62.6	62.6	-0.3%	-5.9%	14.5%
15	**724**	775	891.3	97.0	97.0	**724**	794.8	38.1	38.1	-6.6%	-10.8%	-60.8%
16	**3285**	**3285**	3413.5	33.8	57.6	**3285**	3306.6	24.9	60.4	0.0%	-3.1%	4.8%
17	**0**	**0**	13.6	1.0	43.1	**0**	2.5	4.4	37.4	0.0%	-81.4%	-13.2%
18	**767**	**767**	813.5	86.6	86.6	**767**	789.0	36.5	74.0	0.0%	-3.0%	-14.6%
19	**0**	**0**	0.0	0.2	1.3	**0**	0.0	0.9	5.4	0.0%	0.0%	317.4%
20	**1757**	**1757**	1938.9	23.0	57.0	**1757**	1790.9	21.9	61.6	0.0%	-7.6%	8.0%
21	**0**	**0**	0.0	0.1	0.3	**0**	0.0	0.1	0.4	0.0%	0.0%	32.7%
22	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	25.6%
23	**0**	**0**	0.0	0.1	0.1	**0**	0.0	0.1	0.1	0.0%	0.0%	-3.6%
24	**761**	773	1037.2	61.8	61.8	**761**	1027.8	93.7	71.9	-1.6%	-0.9%	16.3%
25	**0**	**0**	0.0	0.2	0.4	**0**	0.0	0.1	0.6	0.0%	0.0%	64.9%
26	**0**	**0**	0.0	0.2	0.3	**0**	0.0	0.1	0.3	0.0%	0.0%	18.3%
27	**0**	**0**	0.0	0.5	4.3	**0**	0.0	4.1	8.1	0.0%	0.0%	87.9%
28	**0**	**0**	0.0	0.2	0.4	**0**	0.0	0.2	1.7	0.0%	0.0%	312.1%
29	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.4	0.0%	0.0%	69.8%

30	**0**	**0**	28.2	3.2	39.6	**0**	24.9	10.6	35.0	0.0%	-11.5%	-11.6%
31	**0**	**0**	0.0	0.1	0.1	**0**	0.0	0.1	0.1	0.0%	0.0%	-2.1%
32	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	3.0%
33	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	3.5%
34	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	-6.5%
35	**0**	**0**	0.0	0.1	0.1	**0**	0.0	0.1	0.1	0.0%	0.0%	-4.6%
36	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.1	0.0%	0.0%	-4.2%
37	**0**	**0**	293.3	51.0	51.0	**0**	77.4	28.5	65.6	0.0%	-73.6%	28.5%
38	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	-6.0%
39	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	10.0%
40	**0**	**0**	0.0	0.1	0.2	**0**	0.0	0.1	0.2	0.0%	0.0%	-2.6%
41	**69102**	**69102**	69318.2	11.4	58.4	**69102**	69222.7	7.3	16.1	0.0%	-0.1%	-72.5%
42	**57487**	**57487**	57644.6	1.8	46.3	**57487**	57558.0	6.3	31.0	0.0%	-0.2%	-33.2%
43	**145310**	**145310**	145930.5	3.3	34.0	**145310**	145659.6	8.8	23.1	0.0%	-0.2%	-32.2%
44	**35166**	**35166**	35328.3	3.2	27.8	**35166**	35253.8	9.7	29.0	0.0%	-0.2%	4.0%
45	**58935**	**58935**	59018.6	3.0	43.7	**58935**	58974.6	5.9	18.0	0.0%	-0.1%	-58.8%
46	**34764**	**34764**	35034.8	17.1	57.7	**34764**	34896.8	9.6	28.3	0.0%	-0.4%	-51.0%
47	**72853**	**72853**	73160.4	7.0	44.4	**72853**	73070.6	7.2	18.3	0.0%	-0.1%	-58.9%
48	**64612**	**64612**	64719.6	10.4	52.2	**64612**	64699.6	7.9	23.1	0.0%	0.0%	-55.7%
49	**77449**	**77449**	78176.6	5.0	53.6	**77449**	78091.2	9.1	26.5	0.0%	-0.1%	-50.6%
50	**31092**	**31092**	31580.6	31.8	52.4	**31092**	31194.6	9.5	30.0	0.0%	-1.2%	-42.7%
51	**49208**	**49208**	50082.5	27.5	63.2	**49208**	49510.6	10.0	31.3	0.0%	-1.1%	-50.5%
52	**93045**	**93045**	94653.3	15.2	40.9	**93045**	93782.8	11.1	42.6	0.0%	-0.9%	4.0%
53	**84841**	**84841**	86465.3	13.0	56.7	**84841**	86566.7	14.5	31.8	0.0%	0.1%	-43.8%
54	**118809**	**118809**	120150.8	21.6	52.9	**118809**	119639.6	10.6	36.0	0.0%	-0.4%	-32.0%
55	**64315**	**64315**	66055.4	11.4	51.8	**64315**	65106.7	10.8	40.0	0.0%	-1.4%	-22.8%
56	**74889**	**74889**	75472.9	12.8	52.5	**74889**	75060.2	8.5	22.4	0.0%	-0.5%	-57.4%
57	**63514**	**63514**	65195.4	23.3	31.4	**63514**	64494.3	12.0	21.0	0.0%	-1.1%	-33.3%
58	**45322**	**45322**	46286.1	22.7	57.5	**45322**	45781.0	12.7	37.1	0.0%	-1.1%	-35.6%
59	**50999**	**50999**	51954.0	24.8	71.2	**50999**	51348.2	12.9	47.7	0.0%	-1.2%	-33.1%
60	**60765**	**60765**	62498.8	22.2	55.4	**60765**	61397.4	14.2	46.7	0.0%	-1.8%	-15.7%
61	**75916**	**75916**	75998.5	5.6	37.0	**75916**	76094.9	7.4	25.5	0.0%	0.1%	-31.0%
62	**44769**	**44769**	44840.0	5.9	38.8	**44769**	44833.1	6.6	18.3	0.0%	0.0%	-52.8%
63	**75317**	**75317**	75536.9	5.1	48.2	**75317**	75627.0	7.1	11.9	0.0%	0.1%	-75.4%
64	**92572**	**92572**	92609.8	2.7	28.1	**92572**	92650.5	5.7	9.4	0.0%	0.0%	-66.5%
65	**126696**	**126696**	127080.1	8.9	39.6	**126696**	127428.5	6.2	22.0	0.0%	0.3%	-44.4%
66	**59685**	**59685**	59927.3	4.5	56.1	**59685**	60034.8	6.9	20.6	0.0%	0.2%	-63.4%
67	**29390**	**29390**	29415.8	3.0	27.4	**29390**	29397.0	6.4	13.2	0.0%	-0.1%	-51.9%

68	**22120**	**22120**	22264.6	4.0	30.5	**22120**	22143.0	6.7	20.0	0.0%	-0.5%	-34.3%
69	**71118**	**71118**	71307.7	0.9	37.1	**71118**	71164.6	6.2	18.8	0.0%	-0.2%	-49.2%
70	**75102**	**75102**	75427.4	2.4	28.2	**75102**	75433.9	5.8	18.7	0.0%	0.0%	-33.7%
71	**145007**	**145007**	147119.7	9.0	41.8	**145007**	146653.5	15.3	31.3	0.0%	-0.3%	-25.2%
72	**43286**	**43286**	45678.3	37.1	63.7	**43286**	44177.7	11.0	33.7	0.0%	-3.3%	-47.2%
73	**28785**	**28785**	29054.4	16.9	60.2	**28785**	29129.3	9.3	30.5	0.0%	0.3%	-49.3%
74	**29777**	**29777**	30765.0	10.7	47.7	**29777**	30378.2	13.5	44.2	0.0%	-1.3%	-7.4%
75	**21602**	**21602**	22450.0	10.8	61.6	**21602**	22130.9	8.6	25.0	0.0%	-1.4%	-59.4%
76	**53555**	**53555**	54529.2	11.1	50.4	**53555**	53819.2	8.8	27.9	0.0%	-1.3%	-44.7%
77	**31817**	31937	33374.4	63.9	82.2	**31817**	32797.0	12.3	34.9	-0.4%	-1.7%	-57.5%
78	**19462**	**19462**	20381.8	10.5	43.9	**19462**	19871.3	8.3	27.4	0.0%	-2.5%	-37.7%
79	**114999**	**114999**	116196.8	5.6	50.4	**114999**	116054.9	9.7	25.1	0.0%	-0.1%	-50.2%
80	**18157**	**18157**	19274.4	5.7	58.1	**18157**	18660.2	7.5	21.6	0.0%	-3.2%	-62.8%
81	**383485**	**383485**	383903.1	0.9	42.7	**383485**	383662.7	5.7	9.6	0.0%	-0.1%	-77.5%
82	**409479**	**409479**	409815.6	6.8	49.6	**409479**	409922.4	7.2	24.1	0.0%	0.0%	-51.5%
83	**458752**	**458752**	458922.1	4.4	35.8	**458752**	458980.5	6.1	15.8	0.0%	0.0%	-55.8%
84	**329670**	**329670**	329969.7	11.0	51.8	**329670**	329978.1	5.9	13.0	0.0%	0.0%	-74.9%
85	**554766**	**554766**	555107.2	31.3	31.3	**554766**	555118.6	8.8	8.8	0.0%	0.0%	-71.9%
86	**361417**	**361417**	361826.3	3.2	31.6	**361417**	361576.4	5.9	11.9	0.0%	-0.1%	-62.4%
87	**398551**	**398551**	398623.8	0.5	17.4	**398551**	398645.1	5.6	11.0	0.0%	0.0%	-36.8%
88	**433186**	**433186**	433564.4	10.0	44.6	**433186**	433463.8	8.6	17.2	0.0%	0.0%	-61.3%
89	**410092**	**410092**	410187.3	4.0	28.7	**410092**	410411.9	5.8	9.5	0.0%	0.1%	-67.0%
90	**401653**	**401653**	401825.5	3.2	36.6	**401653**	401843.7	6.1	14.3	0.0%	0.0%	-60.9%
91	**339933**	**339933**	340079.3	3.1	35.6	**339933**	340025.4	5.8	16.0	0.0%	0.0%	-55.1%
92	**361152**	**361152**	362030.6	4.4	53.8	**361152**	362196.9	8.2	21.0	0.0%	0.0%	-61.0%
93	**403423**	**403423**	405344.0	13.8	49.6	**403423**	404958.1	7.3	23.7	0.0%	-0.1%	-52.3%
94	**332941**	**332941**	333319.3	9.1	36.2	**332941**	333178.9	7.0	28.0	0.0%	0.0%	-22.6%
95	**516926**	**516926**	518751.3	3.1	50.2	**516926**	518849.9	14.4	49.1	0.0%	0.0%	-2.1%
96	**455448**	**455448**	457814.4	54.1	54.1	**455448**	458241.3	17.5	17.5	0.0%	0.1%	-67.6%
97	**407590**	**407590**	408437.8	5.8	52.1	**407590**	408981.6	7.4	18.9	0.0%	0.1%	-63.8%
98	**520582**	**520582**	522055.2	18.2	52.1	**520582**	522093.9	10.4	20.2	0.0%	0.0%	-61.2%
99	**363518**	**363518**	364803.2	37.6	59.0	**363518**	364709.5	11.3	33.9	0.0%	0.0%	-42.5%
100	**431736**	**431736**	433105.8	1.0	35.8	**431736**	432781.5	8.3	18.1	0.0%	-0.1%	-49.5%
101	**352990**	**352990**	353033.1	1.2	24.6	**352990**	353004.9	5.0	18.0	0.0%	0.0%	-26.9%
102	**492572**	**492572**	492832.6	2.7	33.9	**492572**	492914.3	5.4	13.8	0.0%	0.0%	-59.3%
103	**378602**	**378602**	378834.0	6.3	30.2	**378602**	378934.1	5.6	12.1	0.0%	0.0%	-59.9%
104	**357963**	**357963**	358174.6	7.9	37.4	**357963**	358077.2	6.0	22.7	0.0%	0.0%	-39.3%
105	**450806**	**450806**	450812.4	3.0	28.5	**450806**	450889.9	5.4	8.1	0.0%	0.0%	-71.6%

106	**454379**	**454379**	454851.6	9.5	44.5	**454379**	455091.8	5.9	14.6	0.0%	0.1%	-67.2%
107	**352766**	**352766**	353002.3	3.5	42.9	**352766**	353153.6	5.4	18.7	0.0%	0.0%	-56.4%
108	**460793**	**460793**	461112.6	5.7	45.0	**460793**	461390.3	6.8	13.6	0.0%	0.1%	-69.7%
109	**413004**	**413004**	413426.1	6.1	31.6	**413004**	413643.8	6.2	27.8	0.0%	0.1%	-12.3%
110	**418769**	**418769**	419030.4	6.9	38.1	**418769**	418954.5	5.6	14.8	0.0%	0.0%	-61.1%
111	**342752**	**342752**	343768.0	9.0	40.8	**342752**	343687.6	8.1	17.4	0.0%	0.0%	-57.4%
112	**367110**	**367110**	369592.0	6.8	42.9	**367110**	368861.7	9.4	18.7	0.0%	-0.2%	-56.5%
113	**259649**	**259649**	259909.2	4.7	43.5	**259649**	260246.4	7.3	20.0	0.0%	0.1%	-54.0%
114	**463474**	**463474**	465440.2	10.7	47.2	**463474**	464804.5	8.3	19.5	0.0%	-0.1%	-58.8%
115	**456890**	**456890**	458414.7	24.4	60.8	**456890**	457602.0	12.4	40.2	0.0%	-0.2%	-33.9%
116	**530601**	**530601**	531410.8	5.2	35.8	**530601**	531323.7	7.5	14.3	0.0%	0.0%	-60.0%
117	**502840**	**502840**	503600.9	12.0	50.5	**502840**	503337.0	7.1	19.6	0.0%	-0.1%	-61.2%
118	**349749**	**349749**	352050.9	5.8	53.9	**349749**	352581.9	11.3	15.5	0.0%	0.2%	-71.3%
119	**573046**	**573046**	573759.3	3.9	46.1	**573046**	573457.4	7.9	11.8	0.0%	-0.1%	-74.5%
120	**396183**	**396183**	398005.1	2.8	45.8	**396183**	398556.3	8.6	19.6	0.0%	0.1%	-57.1%
平均										-0.1%	-2.2%	-26.3%

黑体：最优解

从表4.5可以看出，LOX⊕B算法所得的结果不论是与最优解（OPT）还是与BILS算法的结果相比，都有着非常明显的竞争力。与最优解相比，LOX⊕B算法找到了119个算例的最优解，余下的那1个算例的结果也跟最优解非常接近，表明了LOX⊕B算法的高效性；与BILS算法相比，LOX⊕B算法改进了7个算例的最好解，并且没有一个算例的解比BILS算法差。另外，观察最后一行Δf_{best}、Δf_{aver}和Δt_{aver}总的平均值分别为-0.1%、-2.2%和-26.3%。由此可以得出，不论是从最好解还是平均值来看，LOX⊕B 的性能都要优于BILS算法。同时，从找到最好解的平均时间来看，LOX⊕B 也比BILS算法更快。上述结论证实了混合进化算法LOX⊕B 与BILS算法相比，要更稳定且更高效。

另外，将LOX⊕B算法与Tanaka和Araki的精确算法以及优秀的启发式算法进行对比，结果如表4.6所示。表4.6中总结了LOX⊕B与其它相关算法相比，改进、持平和差的解的个数。由表4.6可以看出，LOX⊕B 算法在这种对比规则下仍然非常有优势。将LOX⊕B算法的结果与OBK、ACO_AP、DPSO、DDE、GVNS和BILS相比，对于120个算例，改进的解的个数分别为94、84、66、52、44和7；没有一个算例的

解比上述6组解要差。即使将LOX⊕B算法和Tanaka和Araki的精确算法相比，119个算例的解都与最优解持平。需要注意的是，OBK 这一列中，所有解的个数为117，因为在OBK这组解中，算例6、69和116的解被Tanaka和Araki[54] 证明是不正确的。

表 4.6: LOX⊕B算法对于$1|s_{ij}|\sum w_jT_j$问题的最好解与其它相关算法的对比结果统计

	OPT	OBK	ACO_AP	DPSO	DDE	GVNS	BILS
改进解的个数	0	94	84	66	52	44	7
持平解的个数	119	23	36	54	68	76	113
差的解的个数	1	0	0	0	0	0	0
所有解的个数	120	117	120	120	120	120	120

在文献[82]中，Tasgetiren等人为了将提出的离散差分进化算法（DDE）与ACO_AP和DPSO算法进行一个公平的对比，对120个算例分别运行10 次，每次运行25秒，DDE算法展示出了非常优秀的能力。因此，为了公平的对比LOX⊕B和DDE算法的计算能力，对120个算例也采用上述标准对LOX⊕B算法进行测试。

在处理$1|s_{ij}|\sum w_jT_j$ 问题的120算例时，对每个算例运行10次，每次运行25秒后LOX⊕B算法的最好解和DDE算法最好解的差异（Δbest = LOX⊕B $-$ DDE）见表4.7。从表4.7 可以看出，LOX⊕B 相对于DDE来说，改进、持平和差的解的个数分别为50、62和8。由此可见，在相同的运行时间限制内，从解的质量来看，LOX⊕B算法和DDE算法相比是有较强竞争力的。

4.3.4 $1|s_{ij}|\sum T_j$问题的计算结果

考虑到LOX⊕B 算法在求解$1|s_{ij}|\sum w_jT_j$ 问题时的优秀表现，将LOX⊕B算法不做任何修改用来解决不带权的$1|s_{ij}|\sum w_jT_j$问题，即$1|s_{ij}|\sum T_j$问题。不带权带准备时间的延迟调度问题$1|s_{ij}|\sum T_j$ 其实就是$1|s_{ij}|\sum w_jT_j$ 问题的一种特殊情况，即当$1|s_{ij}|\sum w_jT_j$中所有工件的权值为1或者都相等的情况。

文献中关于$1|s_{ij}|\sum T_j$问题的著名算例由两个部分组成。第一部分包括32个中小规模的算例，它们包含的工件数分别为15、25、35和45[55]。另一部分包括32个大规模的算例，它们包含的工件数则分别为55、65、75和85[56]。对于这64个算例的每

表 4.7 LOX⊕B算法的最好解相对于DDE算法最好解的差异

算例	Δbest	算例	Δbest	算例	Δbest	算例	Δbest	算例	Δbest	算例	Δbest
1	-15	21	0	41	-140	61	0	81	0	101	0
2	-72	22	0	42	-24	62	0	82	-65	102	-176
3	-55	23	0	43	0	63	0	83	0	103	0
4	-80	24	-3	44	-123	64	0	84	264	104	0
5	0	25	0	45	0	65	0	85	-220	105	0
6	-60	26	0	46	0	66	0	86	0	106	0
7	-53	27	0	47	-152	67	0	87	-119	107	0
8	-13	28	0	48	0	68	0	88	58	108	0
9	-117	29	0	49	-169	69	0	89	0	109	404
10	-38	30	0	50	-473	70	0	90	0	110	0
11	-448	31	0	51	-719	71	283	91	-575	111	0
12	0	32	0	52	-1558	72	-129	92	0	112	0
13	-216	33	0	53	1533	73	0	93	-1125	113	-1223
14	-80	34	0	54	-265	74	5	94	-79	114	-2029
15	-138	35	0	55	-1691	75	0	95	-33	115	-385
16	-575	36	0	56	-478	76	0	96	-1959	116	-202
17	-61	37	-51	57	-246	77	-300	97	-1673	117	0
18	-54	38	0	58	200	78	0	98	-2904	118	329
19	0	39	0	59	-974	79	0	99	-924	119	0
20	-338	40	0	60	0	80	0	100	0	120	0
改进解的个数								50			
持平解的个数								62			
差的解的个数								8			
所有解的个数								120			

个算例，将LOX⊕B程序以不同的随机种子运行100次，每次的最长运行时间为100秒CPU时间。

对于$1|s_{ij}|\sum T_j$ 问题的64个公共算例，Tanaka和Araki提出的精确算法找到了其中62个算例的最优解和另外2个算例的当前最好解，本节中，其结果仍然用OPT来表示。将LOX⊕B 算法求解$1|s_{ij}|\sum T_j$ 问题64个算例的结果与其它一些优秀算法的结果进行比较。表4.8给出的是LOX⊕B 算法和其它一些优秀算法求解32个中小规模算例的最好解。用来对比的优秀算法的解包括OPT、RSPI、ACO_GPG、Tabu-VNS、

ACO_LJ、B&B、IG和GVNS，除了精确算法的最优解OPT之外，其它几组最好解见文献[83]。

表 4.8: LOX⊕B算法对于32个中小规模算例的最好解与其它相关算法对比

算例	N	OPT	RSPI	ACO_GPG	Tabu-VNS	ACO_LJ	B&B	IG	GVNS	LOX⊕B
Prob401	15	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**	**90**
Prob402	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob403	15	**3418**	**3418**	**3418**	**3418**	**3418**	**3418**	**3418**	**3418**	**3418**
Prob404	15	**1067**	**1067**	**1067**	**1067**	**1067**	**1067**	**1067**	**1067**	**1067**
Prob405	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob406	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob407	15	**1861**	**1861**	**1861**	**1861**	**1861**	**1861**	**1861**	**1861**	**1861**
Prob408	15	**5660**	**5660**	**5660**	**5660**	**5660**	**5660**	**5660**	**5660**	**5660**
Prob501	25	**261**	266	**261**	**261**	263	**261**	**261**	**261**	**261**
Prob502	25	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob503	25	**3497**	**3497**	**3497**	3503	**3497**	**3497**	**3497**	**3497**	**3497**
Prob504	25	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob505	25	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob506	25	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob507	25	**7225**	**7225**	7268	**7225**	**7225**	**7225**	**7225**	**7225**	**7225**
Prob508	25	**1915**	**1915**	1945	**1915**	**1915**	**1915**	**1915**	**1915**	**1915**
Prob601	35	**12**	36	16	**12**	14	**12**	**12**	**12**	**12**
Prob602	35	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob603	35	**17587**	17792	17685	17605	17654	**17587**	**17587**	**17587**	**17587**
Prob604	35	**19092**	19238	19213	19168	**19092**	**19092**	**19092**	**19092**	**19092**
Prob605	35	**228**	273	247	**228**	240	**228**	**228**	**228**	**228**
Prob606	35	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob607	35	**12969**	13048	13088	**12969**	13010	**12969**	**12969**	**12969**	**12969**
Prob608	35	**4732**	4733	4733	**4732**	**4732**	**4732**	**4732**	**4732**	**4732**
Prob701	45	**97**	118	103	98	103	**97**	103	99	**97**
Prob702	45	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob703	45	**26506**	26745	26663	**26506**	26568	26533	26496[a]	**26506**	**26506**
Prob704	45	**15206**	15415	15495	15213	15409	16577	**15206**	**15206**	**15206**
Prob705	45	**200**	254	222	**200**	219	**200**	**200**	202	**200**
Prob706	45	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob707	45	**23789**	24218	24017	23804	23931	23797	23794	**23789**	**23789**

Prob708	45	**22807**	23158	23351	22873	23028	22829	22829	**22807**	**22807**

黑体：最优解，[a]: 不正确的解

将LOX⊕B算法与其它相关算法对比的结果进行统计，如表4.9所示，列出了LOX⊕B和相关算法相比，改进、持平和差的解的个数。通过观察可以得知，对于这32个中小规模的算例来说，LOX⊕B 算法找到了所有算例的最优解。将LOX⊕B的结果分别和RSPI、ACO_GPG、Tabu-VNS、ACO_LJ、B&B、IG、GVNS 相比，改进解的个数分别为13、14、7、11、4、3、2；并且，没有一个算例的结果比其它算法差。需要注意的是，表4.9 中“IG”这一列，所有解的个数为31，是因为Tanaka和Araki的精确算法证明了IG对于算例prob703的最好解“26496”是错误的[54]。从表4.8 和4.9可以看出，LOX⊕B 算法求解$1|s_{ij}|\sum T_j$问题32个中小规模算例时，在找到最优解方面的表现要胜过之前所有的优秀启发式算法。

表 4.9: LOX⊕B算法对于32个中小规模算例的最好解与其它相关算法的对比结果统计

	OPT	RSPI	ACO_GPG	Tabu-VNS	ACO_LJ	B&B	IG	GVNS
改进解的个数	0	13	14	7	11	4	3	2
持平解的个数	32	19	18	25	21	28	28	30
差的解的个数	0	0	0	0	0	0	0	0
所有解的个数	32	32	32	32	32	32	31	32

表4.10中列出了LOX⊕B算法以及Tanaka和Araki的精确算法和其它优秀的启发式算法对于32个大规模算例的计算结果。ACO_GPG、Tabu-VNS、ACO_LJ、GRASP、IG和GVNS 的最好解可以参见文献[83]。

表 4.10: LOX⊕B算法对于32个大规模算例的最好解与其它相关算法对比

算例	N	OPT	ACO_GPG	Tabu-VNS	ACO_LJ	GRASP	IG	GVNS	LOX⊕B
Prob551	55	**183**	212	185	**183**	242	**183**	194	**183**
Prob552	55	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob553	55	**40498**	40828	40644	40676	40678	40598	40540	**40498**
Prob554	55	**14653**	15091	14711	14684	**14653**	**14653**	**14653**	**14653**

Prob555	55	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob556	55	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob557	55	**35813**	36489	35841	36420	35883	35827	35830	**35813**
Prob558	55	**19871**	20624	19872	19888	**19871**	**19871**	**19871**	**19871**
Prob651	65	**247**	295	268	268	333	268	264	**247**
Prob652	65	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob653	65	**57500**	57779	57602	57584	57880	57584	57515	**57500**
Prob654	65	**34301**	34468	34466	34306	34410	34306	**34301**	**34301**
Prob655	65	**0**	13	2	7	30	2	4	2
Prob656	65	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob657	65	**54895**	56246	55080	55389	55355	55080	**54895**	**54895**
Prob658	65	**27114**	29308	27187	27208	**27114**	**27114**	**27114**	**27114**
Prob751	75	**225**	263	241	241	317	–	241	229
Prob752	75	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob753	75	**77544**	78211	77739	77663	78211	77663	77627	**77544**
Prob754	75	**35200**	35826	35709	35630	35323	35250	35219	**35200**
Prob755	75	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob756	75	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob757	75	**59635**	61513	59763	60108	60217	59763	59716	**59635**
Prob758	75	**38339**	40277	38789	38704	38368	38341	**38339**	**38339**
Prob851	85	*360*	453	384	455	531	390	402	381
Prob852	85	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob853	85	**97497**	98540	97880	98443	98794	97880	97595	**97497**
Prob854	85	**79042**	80693	80122	79553	80338	79631	79271	79086
Prob855	85	*258*	333	283	324	393	283	280	270
Prob856	85	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Prob857	85	**87011**	89654	87244	87504	88089	87244	87075	**87011**
Prob858	85	**74739**	77919	75533	75506	75217	75029	74755	**74739**

黑体：最优解，*斜体*：当前最好解

进一步将LOX⊕B与其它相关算法对32个大规模算例的结果进行统计，结果见表4.11，给出了改进、持平和差的解的个数。把LOX⊕B算法的结果与ACO_GPG、Tabu-VNS、ACO_LJ、GRASP、IG 和GVNS进行对比，改进解的个数分别为22、21、21、19、16和16；并且没有一个算例的解比它们差。因为算法IG对于算例prob751的解在文献[83]中未被给出，故表4.11中“IG”列所有解的个数为31。即使将LOX⊕B算法的结果同OPT相比，也只有5个算例的解没有达到最优解。应该注意的是，对于

这5个算例中的3个算例，Tanaka和Araki的精确算法找到最优解或当前最好解都是在最高超过30天的宽松的时间限制内[54]。从表4.10和4.11 可以得出，LOX⊕B 算法在处理$1|s_{ij}|\sum T_j$ 问题32个大规模算例时，与Tanaka和Araki的精确算法及其它优秀启发式算法相比，还是有竞争力的。

表 4.11: LOX⊕B算法对于32个大规模算例的最好解与其它相关算法的对比结果统计

	OPT	ACO_GPG	Tabu-VNS	ACO_LJ	GRASP	IG	GVNS
改进解的个数	0	22	21	21	19	16	16
持平解的个数	27	10	11	11	13	15	16
差的解的个数	5	0	0	0	0	0	0
所有解的个数	32	32	32	32	32	31	32

从表4.10可以看出，LOX⊕B算法不能达到Tanaka和Araki的精确算法的最优解或当前最好解的5个算例分别是prob655、prob751、prob851、prob854 和prob855。Tanaka和Araki的精确算法，除了算例prob655 和prob854，其它3个算例的最优解需要超过30天才被找到。LOX⊕B算法对这5个困难算例在宽松的运行时间限制内能否找到最优解或当前最好解，甚至改进当前最好解呢? 为此，重新对这5个算例进行实验，每个算例只运行一次，程序的运行时间设置为24小时CPU时间。在具体的实验过程中，设置如果当前最好解在连续2 个小时内未被改进，则随机生成一个新的初始解，重新开始计算。在表4.12中分别列出了LOX⊕B 算法与Tanaka和Araki的精确算法在处理这5个困难算例时所得最好解以及计算时间。

表4.12中，第3列给出的是精确算法找到的最优解或当前最好解（OPT），第4-6列给出的是精确算法在三种最大内存条件（512 MB、2G和20GB）下找到OPT所需的CPU 时间，第7列给出了LOX⊕B 在24小时CPU计算时间限制内找到的最好解（f_{best}），最后一列给出了LOX⊕B 在找到最好解f_{best} 时所需要的CPU时间（t_{total}）。

从表4.12可以清楚的看到，对于算例prob855，LOX⊕B算法找到的最好解为“256”，改进了Tanaka和Araki的精确算法找到的当前最好解“258”。并且值得一提的是，Tanaka和Araki的精确算法的最好解“258”是在20GB的内存环境下用了超过30天的时间才找到的。而LOX⊕B 算法找到最好解“256”

时只用了6037.93秒（≈1.7小时）的CPU时间。同样，对于算例prob751和prob851，LOX⊕B算法找到OPT的时间分别为33678.64秒（≈9.4小时）和84405.88秒（≈23.4小时），而Tanaka和Araki的精确算法找到OPT却分别需要34天和超过30天的CPU时间。以上结果充分体现了LOX⊕B 算法的有效性和高效性。

表 4.12: LOX⊕B算法对于5个困难算例与精确算法的计算效率对比

算例	N	Tanaka和Araki的精确算法				LOX⊕B算法	
		OPT	时间（秒）			f_{best}	t_{total}
			512MB	2GB	20GB		
Prob655	65	**0**	679.37	1032.64	2596.73	**0**	1006.26
Prob751	75	**225**	–	–	34 days	**225**	33678.64
Prob851	85	*360*	–	–	>30 days	*360*	84405.88
Prob854	85	**79042**	2499.42	2451.33	2198.15	**79042**	180.06
Prob855	85	258	–	–	>30 days	*256*	6037.93

黑体：最优解，*斜体*：当前最好解

在文献[75]中，对于$1|s_{ij}|\sum T_j$问题的64个算例，IG算法对于每个算例运行10次，每次的运行时间设为$N^2/300$秒（N为算例中包含的工件数）CPU时间。因此，为了将LOX⊕B算法与IG算法做一个公平的对比，对这64 个算例的每个算例重新运行10次，每次的运行时间设定为$N^2/300$秒CPU时间。

表4.13中总结了LOX⊕B 对于$1|s_{ij}|\sum T_j$ 问题的64个算例的每个算例运行10次，每次的运行时间为$N^2/300$秒CPU时间，找到的最好解相对于IG算法的最好解的差异（Δbest = LOX⊕B $-$ IG ）。需要注意的是，算法IG对于算例prob703的解在Tanaka和Arak[54]的精确算法中被证明是错误的，并且其对于算例prob751的解在文献[83]中未被给出，因此，在表4.13，所有解的个数为62而不是64。从表4.13可以看出，LOX⊕B 相对IG来说，改进、持平和差的解的个数分别为15、40和7。在比IG差的7 个算例中，有4 个算例的工件数都是85。可以观察到，LOX⊕B 跟IG相比还是有竞争力的，尤其是在算例的规模不是太大的时候。

4.4 分析和讨论

在这一节中，首先用适应度距离相关性（Fitness Distance Correlation）分析对三

表 4.13 LOX⊕B算法的最好解和IG算法最好解的差异

算例	N	Δbest	算例	N	Δbest	算例	N	Δbest	算例	N	Δbest
Prob401	15	0	Prob601	35	0	Prob551	55	17	Prob751	75	–
Prob402	15	0	Prob602	35	0	Prob552	55	0	Prob752	75	0
Prob403	15	0	Prob603	35	0	Prob553	55	-42	Prob753	75	-70
Prob404	15	0	Prob604	35	0	Prob554	55	0	Prob754	75	-37
Prob405	15	0	Prob605	35	0	Prob555	55	0	Prob755	75	0
Prob406	15	0	Prob606	35	0	Prob556	55	0	Prob756	75	0
Prob407	15	0	Prob607	35	0	Prob557	55	-14	Prob757	75	-86
Prob408	15	0	Prob608	35	0	Prob558	55	0	Prob758	75	-2
Prob501	25	0	Prob701	45	-5	Prob651	65	-6	Prob851	85	22
Prob502	25	0	Prob702	45	0	Prob652	65	0	Prob852	85	0
Prob503	25	0	Prob703	45	–	Prob653	65	-69	Prob853	85	6
Prob504	25	0	Prob704	45	0	Prob654	65	-5	Prob854	85	-148
Prob505	25	0	Prob705	45	2	Prob655	65	10	Prob855	85	36
Prob506	25	0	Prob706	45	0	Prob656	65	0	Prob856	85	0
Prob507	25	0	Prob707	45	-5	Prob657	65	-185	Prob857	85	357
Prob508	25	0	Prob708	45	-22	Prob658	65	0	Prob858	85	-26
			改进解的个数						15		
			持平解的个数						40		
			差的解的个数						7		
			所有解的个数						62		

种交叉算子（BOX、LOX、PBX）和两种种群更新策略（A和B）构成的6种HEA算法在处理单机调度问题时的表现进行进一步的讨论。接着，探讨在产生初始种群时采用ATCS规则，以及在选择双亲时采用一定的选择策略是否会提高现有混合进化算法LOX⊕B的效率。

4.4.1 适应度距离相关性（Fitness Distance Correlation，FDC）分析

为了更好的了解三种交叉算子和两种种群更新策略所构成的6种组合的性能，对它们各自的表现进行了适应度距离相关性分析。

适应度距离相关性[112]首先被作为遗传算法求解问题难度的度量提出，揭示了局部最优解与全局最优解之间适应度和距离的相互关系。对于给定的n个个体

适应度值集合$F=\{f_1,f_2,\ldots,f_n\}$和这n个个体到最近的全局最优解之间的距离集合$D=\{d_1,d_2,\ldots,d_n\}$，适应度距离相关性系数记为r，

$$r=\frac{C_{FD}}{S_F S_D} \tag{式 4.6}$$

其中，

$$C_{FD}=\frac{1}{n}\sum_{i=1}^{n}(f_i-\overline{f})(d_i-\overline{d}) \tag{式 4.7}$$

S_F、S_D、$\overline{f}$、$\overline{d}$分别表示F和D的标准偏差和平均值。

FDC可以用来评估问题的难易程度以及对比不同的方法处理同一个问题时表现的优劣[112]。为了更加准确直观地描述适应度与距离之间的关系，引入了一种用于分析搜索空间结构的图形，称为适应度距离图（Fitness Distance Plot）。在适应度距离图中，将其横轴设置为局部最优解到与其最近的全局最优解的距离，纵轴设置为局部最优解的适应度，并进行相应的分析，如Boese将其用于TSP问题[113]，Reeves用于flow shop调度问题[114]，Merz等人用于GBP问题[115]等。

在解决$1|s_{ij}|\sum w_jT_j$问题时，对于6种不同的组合，算例5和10的适应度距离图如图4-6所示，算例61和101的适应度距离图如图4-7所示。在图4-6和4-7中，用hamming距离来表示局部最优解到最近的全局最优解的距离，用目标函数值作为适应度值。从图4-6中可以看出，对于算例5和10，所有6种组合随着到全局最优解距离的减少，局部最优解也随之很大程度的减少。这种情形可以在一定程度上说明算例5和10 是比较难求解的。事实上，前40个算例确实被认为是非常有挑战性的算例[54]。

然而，从图4-7中可以发现，对于算例61和101，所有6种组合的局部最优解的分布都比较连续。尤其是，对于算例61和101来说，当采用种群更新策略B时，局部最优解的分布更紧密。也可以说，当采用种群更新策略B时，地貌更平滑，更容易从局部最优解找到全局最优解。

4.4.2 初始种群的产生及选择策略探讨

在4.2.1节中，提到为了保证初始种群的多样性，初始种群中的个体都是随机产生的。对于$1|s_{ij}|\sum w_jT_j$问题，用ATCS 规则生成的解的质量会比随机生成的解要

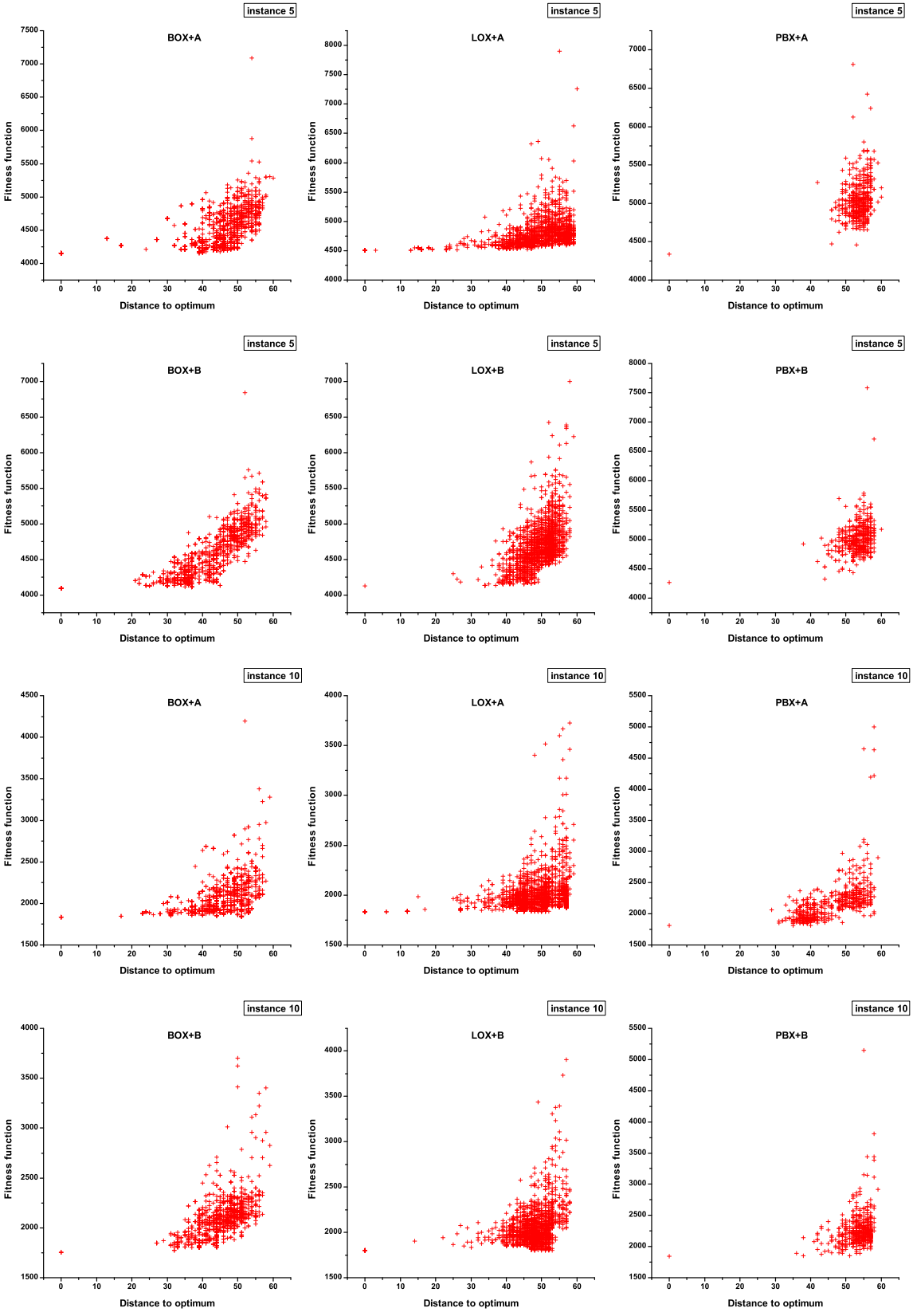

图 4-6 算例5和10的适应度距离图

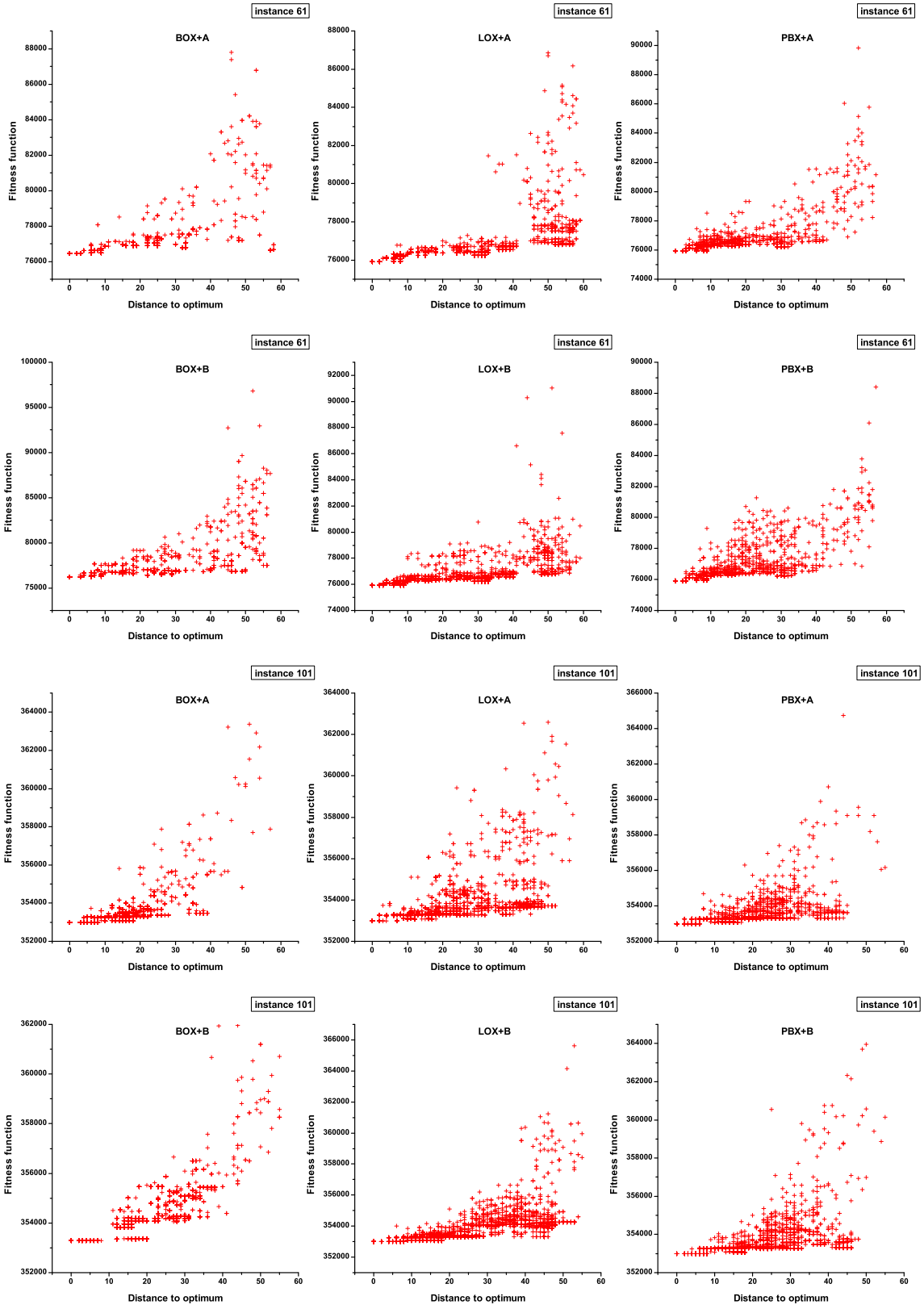

图 4-7 算例61和101的适应度距离图

好。然而，需要注意的是，本文提出的混合进化算法在随机生成初始种群后，种群中的每个个体都会进一步被局部搜索进行优化。为了验证高质量的初始解是否会提高混合进化算法的性能，本文用实验对比采用两种不同初始化种群方法时LOX⊕B算法的表现。

由算法3可见，LOX⊕B算法采用随机的方式选择两个双亲来进行交叉操作。如果采用一定的策略来选择双亲，LOX⊕B算法的表现是否会进一步提高? 为了回答这个问题，在选择双亲的时候考虑使用轮盘赌选择策略，适应度值就采用本文提出的优度函数值，优度值高的个体更可能被选中。在执行交叉操作前，要选择两个双亲，因此需要随机生成两个位于区间(0,1)的随机数。然后，由生成的两个随机数根据轮盘赌选择的规则选择两个双亲。

用$1|s_{ij}|\sum w_jT_j$问题120个算例的前10个算例对LOX⊕B算法及其3种衍生版本进行测试，并将结果进行总结见表4.14所示。表4.14中，LOX⊕B$_A$、LOX⊕B$_R$和LOX⊕B$_{A/R}$分别表示3 种不同版本的LOX⊕B算法，它们都是在LOX⊕B算法上做了一点小的改动。LOX⊕B$_A$表示LOX⊕B在初始化种群的时候，其中一个个体用ATCS算法生成，其它个体还是随机生成；LOX⊕B$_R$表示LOX⊕B 在选择双亲的时候采用了轮盘赌选择策略；LOX⊕B$_{A/R}$表示LOX⊕B在初始化种群中某一个个体的时候采用ATCS规则，并且在选择父母的时候采用了轮盘赌选择策略。LOX⊕B 、LOX⊕B$_A$、LOX⊕B$_R$和LOX⊕B$_{A/R}$对这10 个算例的每个算例分别运行20次，每次的运行时间为100秒。表4.14列出了LOX⊕B 算法的四个不同版本的结果相对于OPT的差异率（例如，ΔLOX⊕B = (LOX⊕B − OPT)/OPT × 100%）。表中best表示最好解相对于OPT的差异率，average表示解的平均值相对于OPT的差异率。

从表4.14可以看出，LOX⊕B$_A$、LOX⊕B$_R$和LOX⊕B$_{A/R}$都能在一定程度上改进LOX⊕B算法解的平均值，并且LOX⊕B$_A$的平均值的表现是最好的。然而，从最好解的角度来看，只有LOX⊕B$_R$的最好解的总平均差异率要比LOX⊕B好。以上结果表明，在选择父母的时候采用轮盘赌选择策略对LOX⊕B是有助益的，并且可以轻微的改进LOX⊕B算法的表现，然而，ATCS规则对LOX⊕B算法并没有太大的影响。之所以ATCS影响不大的原因可能在于，ATCS规则产生的解太集中而不利于保持种群的多样性。

表 4.14 LOX⊕B算法的四个版本相对于OPT的差异率

算例	ΔLOX⊕B		ΔLOX⊕B$_A$		ΔLOX⊕B$_R$		ΔLOX⊕B$_{A/R}$	
	best	average	best	average	best	average	best	average
1	0.00%	4.04%	0.66%	1.75%	0.00%	1.39%	1.32%	2.75%
2	0.00%	0.96%	0.75%	1.30%	0.00%	0.96%	0.50%	1.03%
3	0.36%	0.90%	0.00%	0.65%	0.00%	0.67%	0.00%	1.06%
4	0.00%	0.22%	0.00%	0.00%	0.00%	0.00%	0.00%	0.05%
5	0.49%	1.50%	0.25%	0.94%	0.49%	0.89%	0.25%	0.94%
6	0.00%	0.56%	0.00%	0.18%	0.00%	0.44%	0.00%	0.39%
7	0.00%	1.60%	0.00%	1.20%	0.00%	2.42%	0.00%	1.21%
8	0.00%	1.35%	1.00%	2.50%	0.00%	2.15%	0.00%	1.70%
9	0.00%	0.87%	0.00%	0.59%	0.00%	0.74%	0.00%	0.93%
10	0.00%	0.82%	0.00%	0.88%	0.00%	0.66%	0.00%	0.87%
平均	0.09%	1.28%	0.27%	1.00%	0.05%	1.03%	0.21%	1.09%

4.5 本章小节

本章采用混合进化算法求解单机调度问题，并提出一种高效的混合进化算法LOX⊕B。LOX⊕B 算法混合了局部搜索算法和进化算法，通过交叉和更新不断完善种群，同时通过局部搜索过程对子代进行优化，综合两种算法的优势，从而有可能更迅速地找到全局最优解。

第一节介绍了混合进化算法的基本组成，主要包括编码、初始种群的产生、选择策略、繁殖策略和更新策略等。

第二节提出了一种新的交叉算子BOX（Block Order Crossover Operator），以及一种新的种群更新策略（基于相似性和质量的优度函数的更新策略），并通过实验确定更新策略中参数β的取值。

第三节通过实验对HEA算法进行分析。在对比了几种交叉算子和种群更新策略的组合在HEA算法中处理单机调度问题的表现后，提出高效的混合进化算法LOX⊕B，并使用LOX⊕B算法求解$1|s_{ij}|\sum w_jT_j$ 和$1|s_{ij}|\sum T_j$ 问题。对于$1|s_{ij}|\sum w_jT_j$问题的120个公共算例，LOX⊕B算法找到了119个算例的最优解；与当前优秀的启发式算法的最好解OBK、ACO_AP、DPSO、DDE、GVNS以及本文提出的BILS相比，改进的解的个数分别为94、84、66、52、44、7。对于$1|s_{ij}|\sum T_j$问

题的64个公共算例，LOX⊕B算法找到了全部32个中小规模算例的最优解，找到了32个大规模算例中27 个算例的最优解。对于没找到最优解或当前最好解的5个大规模算例，在放宽CPU计算时间限制到24小时后，改进了算例prob855的当前最好解（LOX⊕B算法对于算例prob855的最好解为256，Tanaka和Araki的精确算法对于算例prob855的当前最好解为258），并找到了另外4个算例的最优解或当前最好解。

第四节首先使用适应度距离相关性对三种交叉算子（BOX、LOX、PBX）和两种种群更新策略（A和B）组成的6种组合在HEA算法中处理单机调度问题时的表现进行分析。然后，探讨在产生初始种群时采用ATCS规则，以及在选择双亲时采用一定的选择策略是否会提高现有LOX⊕B 算法的效率。

5 总结与展望

调度问题已提出近60余年，这期间形成了一定的理论体系，研究人员一直在该领域努力探索，希望能找到更好的解决此类问题的算法。理论上，单机调度可以看成是其它调度问题的特殊形式，深入研究单机调度问题可以更好的理解复杂的调度问题，同时，求解单机调度问题的算法也可以作为求解复杂调度问题的基础。单机调度问题是生产调度领域中一类非常重要和基本的问题，虽然它的模型比较简单，但是，在理论和实际生产当中都具有相当重要的地位。

大部分单机调度问题被证明是NP难的，求解这类问题通常有三种可选方案：精确算法、近似算法、启发式算法。精确算法求解规模较大的问题时的时间复杂度通常是指数型的，无法满足实际应用；近似算法找到的解优度不高，理论上有保障的解的优度离实际需求经常相差较远；启发式算法大多数情形下能够在合理的计算时间内找到质量满足实际需求的解。

近年来，国内外学者对单机调度问题做了广泛而又深入的研究，并提出了多种高效启发式优化算法对其进行求解。启发式算法的主要思想来源于生物世界、物理世界和社会现象，它有可能在较短的时间内求解大规模的问题实例，并得到令人满意的优度。

5.1 全文总结及研究成果

本文对单机调度问题及其求解方法进行了较为深入的研究，提出两种求解单机调度问题的启发式算法，一种是迭代局部搜索算法BILS，一种是混合进化算法LOX⊕B。将本文新提出的两种启发式算法与当前国际文献中介绍的最优秀的精确算法和启发式算法进行了详细对比。实验结果表明，本文提出的BILS算法和LOX⊕B算法，不论是与Tanaka和Araki的精确算法相比还是与最优秀的启发式算法相比都具有较大的优势。论文的主要工作如下：

（1）讨论了生产调度问题的分类和描述方法，着重研究了带序列相关准备时间的单机加权（不加权）延迟调度问题。系统的总结了几种经典的启发式算法的设计

方法，包括：局部搜索算法、变邻域搜索算法、蚁群算法、禁忌搜索算法、迭代局部搜索算法、混合进化算法。

（2）提出了高效的迭代局部搜索算法BILS来求解单机调度问题。迭代局部搜索算法中，邻域结构设计的优劣是决定算法好坏的重要因素之一。本研究对当前文献中求解单机调度问题时用的较多的邻域结构：交换（swap）、插入（insertion）、边插入（edge-insertion）进行了分析比较。提出了一种新的块移动邻域结构$N_{Block\ Move}$，它的主要特点是能够有效的扩大搜索空间。但是扩大搜索空间意味着会花费更多的计算代价，因此$N_{Block\ Move}$中设置了两个重要的参数：块长和移动步长，通过控制这两个参数的取值范围来保证搜索空间的有效性。本文通过用多组实验进行对比的方式，确定了$N_{Block\ Move}$ 中块长和移动步长两个参数的合理取值范围。

（3）迭代局部搜索算法中，邻域解的评估策略是关键技术，它占用了相当大一部分计算时间。本文提出一种新的用来评估块移动邻域结构$N_{Block\ Move}$ 的快速增量评估技术，以提高算法效率。

（4）对BILS算法中最重要的两个部分，块移动的邻域结构和快速增量评估技术进行了进一步的分析和讨论。把块移动的邻域结构与之前文献中介绍的交换、插入和边插入邻域结构进行了对比，实验结果表明了块移动邻域结构在处理1$|s_{ij}|\sum w_jT_j$问题时的有效性。同样，把本文提出的快速增量评估技术与扩展的加速评估方法以及直接计算目标函数值的方法进行了对比讨论，实验结果证明了快速增量评估技术的高效性。

（5）用1$|s_{ij}|\sum w_jT_j$问题的120个公共算例对BILS算法进行测试。将BILS算法在100秒100次的运行条件下的测试结果与当前国际上优秀的启发式算法的最好解OBK、ACO_AP、DPSO、DDE和GVNS相比，改进的解的个数分别为94、84、65、52和44，没有改进的解最多不超过3个；跟Tanaka和Araki的精确算法相比，找到113个算例的最优解，7个算例（算例5、11、13、14、15、24）的当前最优解没有找到。对于这7个困难算例，BILS算法在宽松的运行时间条件下找到了全部7个算例的最优解。并且，对于算例18和24，Tanaka和Araki的精确算法找到最优解分别需要两个星期和30天的CPU时间，而本文提出的BILS算法找到最优解只需要8586.6秒（≈2.4小时）和150169.22秒（≈41.7小时）。

（6）混合算法由两种或几种启发式算法按照一定的规则混合而成，混合后的算法能够“取长补短”，改进单个启发式算法的某些不足之处。混合进化算法中，种群的繁殖策略和更新策略是非常重要的组成部分，本文提出一种新的交叉算子BOX（Block Order Crossover Operator）和一种新的基于相似性和质量的优度函数的种群更新策略。

（7）分析对比了由三种交叉算子和两种种群更新策略所构成的6种组合在混合进化算法中求解$1|s_{ij}|\sum w_jT_j$问题的表现后，提出了一种高效的混合进化算法LOX⊕B来求解单机调度问题。

（8）用$1|s_{ij}|\sum w_jT_j$问题的120个公共算例对LOX⊕B算法进行测试。将LOX⊕B算法在100秒100次的运行条件下的实验结果与文献中介绍的优秀启发式算法的最好解OBK、ACO_AP、DPSO、DDE、GVNS以及本文新提出的BILS算法的最好解相比时，改进的解的个数分别为94、84、66、52、44和7，且没有一个算例的解比上述6组解要差；与Tanaka和Araki的精确算法相比，找到119个算例的最优解。

（9）用$1|s_{ij}|\sum T_j$问题32个中小规模的公共算例对LOX⊕B进行测试。将LOX⊕B算法在100秒100次运行条件下的结果分别与优秀启发式算法和精确算法的最好解RSPI、ACO_GPG、Tabu-VNS、ACO_LJ、B&B、IG、GVNS相比，改进解的个数分别为13、14、7、11、4、3、2，且没有一个算例的结果比其它算法差；跟Tanaka和Araki的精确算法相比，LOX⊕B算法找到了全部32个中小规模算例的最优解。

（10）用$1|s_{ij}|\sum T_j$问题32个大规模的公共算例对LOX⊕B进行测试。将LOX⊕B算法在100秒100次运行条件下的结果与优秀启发式算法的最好解ACO_GPG、Tabu-VNS、ACO_LJ、GRASP、IG、GVNS 对比，改进解的个数分别为22、21、21、19、16、16，且没有一个算例的解比它们差；与Tanaka和Araki的精确算法相比，LOX⊕B算法找到27个算例的最优解或当前最好解。对于未找到OPT的5个算例，将LOX⊕B的停止标准设定为24小时CPU时间重新实验。LOX⊕B算法用6037.93秒（≈1.7小时）找到算例prob855的最好解“256”，改进了由Tanaka和Araki的精确算法用超过30天时间找到的当前最好解“258”；找到了另

外4个算例的最优解或当前最好解，其中算例prob751和prob851，LOX⊕B找到OPT的时间分别为33678.64秒（≈9.4小时）和84405.88秒（≈23.4小时），而Tanaka和Araki的精确算法找到OPT却分别需要34天和超过30天的CPU时间。

5.2 主要创新点

（1）对之前研究单机调度问题的文献中采用的邻域结构进行分析比较，提出了一种基于块移动的邻域结构（$N_{Block\ Move}$），这种邻域结构扩大了邻域的搜索空间，并通过控制块的大小和移动步长两个参数的取值范围，提高搜索的有效性，以节约搜索时间。

（2）针对块移动的邻域结构，提出了一种高效的快速增量评估技术，提高了搜索效率。

（3）在块移动邻域结构，以及对块移动邻域结构快速增量评估技术的基础上，提出一种新的迭代局部搜索算法BILS来求解单机调度问题。

（4）提出一种基于“公共块”概念的交叉算子BOX（Block Order Crossover Operator）。

（5）提出了一种新的基于相似性和质量的优度函数的种群更新策略。

（6）对比了不同的交叉算子和种群更新策略的组合在混合进化算法中处理单机调度问题时表现的优劣，提出了一种高效的求解单机调度问题的混合进化算法LOX⊕B。

5.3 研究展望

本文提出了两种高效的启发式算法：迭代局部搜索算法BILS和混合进化算法LOX⊕B。用这两种算法求解$1|s_{ij}|\sum w_jT_j$和$1|s_{ij}|\sum T_j$问题时取得了很好的成绩。在今后的研究中，主要工作包含以下五个方面：

（1）通过对BILS和LOX⊕B算法中的某些参数或策略进行调整来进一步改进这两种算法的性能。

（2）单机调度问题是一大类问题，其中很多问题都是NP难的。对带其它约束条件和优化标准的单机调度问题进行分析和研究，尝试将本文提出的两种算法用于

求解其它NP难度的单机调度问题。

（3）实验证明，本文提出的邻域结构$N_{Block\ Move}$和针对$N_{Block\ Move}$邻域结构的快速增量评估技术是一组高效的启发式搜索及评估策略，将这组策略融入其它启发式算法的框架，用来求解类似的NP难的组合优化问题。

（4）进一步深入研究混合进化算法，尝试将多种启发式算法进行混合，设计出更高效的求解NP难问题的启发式算法。

（5）目前对启发式算法性能的评价大多是从实验的角度进行的，缺乏严格的理论分析。运用相关理论知识与实验相结合对启发式算法从解的优度、计算时间、鲁棒性等方面进行分析评价。

参考文献

[1] 熊锐, 吴澄. 车间生产调度问题的技术现状与发展趋势. 清华大学学报: 自然科学版, 1998, 38(10):55–60

[2] Sen T, Sulek J M, Dileepan P. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. International Journal of Production Economics, 2003, 83(1):1–12

[3] Feldmann M, Biskup D. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. Computers & Industrial Engineering, 2003, 44(2):307–323

[4] Allahverdi A, Ng C, Cheng T E, et al. A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 2008, 187(3):985–1032

[5] Abdul-Razaq T, Potts C N, Van Wassenhove L N. A survey of algorithms for the single machine total weighted tardiness scheduling problem. Discrete Applied Mathematics, 1990, 26(2):235–253

[6] Michael R G, David S J. Computers and intractability: a guide to the theory of NP-completeness. WH Freeman & Co., San Francisco, 1979

[7] 邢文训, 谢金星. 现代优化计算方法. 北京: 清华大学出版社, 2005

[8] Reeves C R. Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc., 1993

[9] Goldberg D E, Segrest P. Finite Markov chain analysis of genetic algorithms. in: Proceedings of Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application. L. Erlbaum Associates Inc., 1987, 1–8

[10] Ankenbrandt C A. An Extension to the Theory of Convergence and a Proof of the Time Complexity of Genetic Algorithms. in: Proceedings of Foundations of genetic algorithms, 1990, 53–68

[11] 叶涛. 在圆形Packing及团簇结构优化问题上的启发式优化算法研究: [博士学位论文]. 武汉: 华中科技大学, 2012

[12] Johnson S M. Optimal two-and three-stage production schedules with setup times included. Naval research logistics quarterly, 1954, 1(1):61–68

[13] 王万良, 吴启迪. 生产调度智能算法及其应用. 北京: 科学出版社, 2007

[14] Graves S C. A review of production scheduling. Operations research, 1981, 29(4):646–675

[15] Graham R L, Lawler E L, Lenstra J K, et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete Mathematics, 1979, 5:287–326

[16] Lenstra J K, Rinnooy Kan A, Brucker P. Complexity of machine scheduling problems. Annals of discrete mathematics, 1977, 1:343–362

[17] Lenstra J K, Rinnooy Kan A. Complexity of scheduling under precedence constraints. Operations Research, 1978, 26(1):22–35

[18] Lenstra J K, Rinnooy Kan A. Complexity results for scheduling chains on a single machine. European Journal of Operational Research, 1980, 4(4):270–275

[19] Labetoulle J, Lawler E, Lenstra J, et al. Preemptive scheduling of uniform machines subject to release dates. in: Proceedings of Progress in combinatorial optimization. Academic Press, 1984, 245–261

[20] KARP R. Reducibility among combinatorial problems. Complexity of Computer Computations, 1972, pages 85–103

[21] Lawler E L. A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness. Annals of discrete Mathematics, 1977, 1:331–342

[22] Du J, Leung J Y T. Minimizing total tardiness on one machine is NP-hard. Mathematics of operations research, 1990, 15(3):483–495

[23] Leung J Y T, Young G H. Minimizing total tardiness on a single machine with precedence constraints. ORSA Journal on Computing, 1990, 2(4):346–352

[24] Cook S A. The complexity of theorem-proving procedures. in: Proceedings of Proceedings of the third annual ACM symposium on Theory of computing. ACM, 1971, 151–158

[25] Sipser M. Introduction to the Theory of Computation. USA: PWS Publishing Company, 1997

[26] 陈宝林. 最优化理论与算法. 北京: 清华大学出版社, 2005

[27] Talbi E G. Metaheuristics: from design to implementation. John Wiley & Sons, 2009

[28] Pólya G. How to solve it. Princeton. New Jersey: Princeton University, 1945

[29] Dantzig G B. Application of the simplex method to a transportation problem. Activity analysis of production and allocation, 1951, 13:359–373

[30] Edmonds J. Matroids and the greedy algorithm. Mathematical programming, 1971, 1(1):127–136

[31] Holland J H. Outline for a logical theory of adaptive systems. Journal of the ACM (JACM), 1962, 9(3):297–314

[32] Holland J H. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975

[33] Kirkpatrick S, Gelatt C D, Vecchi M P, et al. Optimization by simmulated annealing. science, 1983, 220(4598):671–680

[34] Černỳ V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of optimization theory and applications, 1985, 45(1):41–51

[35] Glover F. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research, 1986, 13(5):533–549

[36] Hansen P. The steepest ascent mildest descent heuristic for combinatorial programming. in: Proceedings of Congress on numerical methods in combinatorial optimization, Capri, Italy, 1986, 70–145

[37] Martin O, Otto S W, Felten E W. Large-step Markov chains for the traveling salesman problem. Oregon Graduate Institute of Science and Technology, Department of Computer Science and Engineering, 1991

[38] Dorigo M. Optimization, learning and natural algorithms: [PhD Dissertation]. Italy: Politecnico di Milano, 1992

[39] Dorigo M, Maniezzo V, Colorni A, et al. Positive feedback as a search strategy. 1991

[40] Mladenovićc N. A variable neighborhood algorithm —— a new metaheuristic for combinatorial optimization. in: Proceedings of Abstracts of Papers Presented at Optimization Days, 1995, 112

[41] Glover F, Laguna M. Tabu search. Springer, 1999

[42] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research, 1997, 24(11):1097–1100

[43] Hansen P, Mladenović N. An introduction to variable neighborhood search. Springer, 1999

[44] Hansen P, Mladenović N. Variable neighborhood search: Principles and applications. European journal of operational research, 2001, 130(3):449–467

[45] Hansen P, Mladenović N, Pérez J A M. Variable neighbourhood search: methods and applications. Annals of Operations Research, 2010, 175(1):367–407

[46] Potts C N, Van Wassenhove L N. A branch and bound algorithm for the total weighted tardiness problem. Operations research, 1985, 33(2):363–377

[47] Potts C N, Van Wassenhove L. Dynamic programming and decomposition approaches for the single machine total tardiness problem. European Journal of Operational Research, 1987, 32(3):405–414

[48] Ragatz G L. A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. in: Proceedings of Proceedings: twenty-fourth annual meeting of the Decision Sciences Institute. John Wiley & Sons, Inc., 1993, 1375–1377

[49] Luo X, Chu F. A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. Applied Mathematics and Computation, 2006, 183(1):575–588

[50] Bigras L P, Gamache M, Savard G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization, 2008, 5(4):685–699

[51] Rabadi G, Mollaghasemi M, Anagnostopoulos G C. A branch-and-bound algorithm for the early/tardy machine scheduling problem with a common due-date and sequence-dependent setup time. Computers & Operations Research, 2004, 31(10):1727–1751

[52] Sourd F. Earliness–tardiness scheduling with setup considerations. Computers & operations research, 2005, 32(7):1849–1865

[53] Luo X, Chu C. A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. European Journal of Operational Research, 2007, 180(1):68–81

[54] Tanaka S, Araki M. An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. Computers & Operations Research, 2013, 40(1):344–352

[55] Rubin P A, Ragatz G L. Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research, 1995, 22(1):85–99

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

[57] Tanaka S, Fujikuma S, Araki M. An exact algorithm for single-machine scheduling without machine idle time. Journal of Scheduling, 2009, 12(6):575–593

[58] Tanaka S, Fujikuma S. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. Journal of Scheduling, 2012, 15(3):347–361

[59] Ibaraki T. Enumerative approaches to combinatorial optimization. Annals of Operations Research, 1988, 10,11

[60] Ibaraki T, Nakamura Y. A dynamic programming method for single machine scheduling. European Journal of Operational Research, 1994, 76(1):72–82

[61] Pinedo M L. Scheduling: theory, algorithms, and systems. Springer, 2012

[62] Vepsalainen A P, Morton T E. Priority rules for job shops with weighted tardiness costs. Management science, 1987, 33(8):1035–1047

[63] Lee Y H, Bhaskaran K, Pinedo M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE transactions, 1997, 29(1):45–52

[64] Potts C, Wassenhove L N. Single machine tardiness sequencing heuristics. IIE transactions, 1991, 23(4):346–354

[65] Holsenback J, Russell R, Markland R, et al. An improved heuristic for the single-machine, weighted-tardiness problem. Omega, 1999, 27(4):485–495

[66] Feo T A, Sarathy K, McGahan J. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Computers & Operations Research, 1996, 23(9):881–895

[67] Gupta S R, Smith J S. Algorithms for single machine total tardiness scheduling with sequence dependent setups. European Journal of Operational Research, 2006, 175(2):722–739

[68] Tan K C, Narasimhan R. Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach. Omega, 1997, 25(6):619–634

[69] Armentano V A, Mazzini R. A genetic algorithm for scheduling on a single machine with set-up times and due dates. Production Planning & Control, 2000, 11(7):713–720

[70] França P M, Mendes A, Moscato P. A memetic algorithm for the total tardiness single machine scheduling problem. European Journal of Operational Research, 2001, 132(1):224–242

[71] Tan K C, Narasimhan R, Rubin P A, et al. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. Omega, 2000, 28(3):313–326

[72] Sun X, Noble J S, Klein C M. Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. IIE transactions, 1999, 31(2):113–124

[73] Liao C J, Juan H C. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research, 2007, 34(7):1899–1909

[74] Gagné C, Gravel M, Price W. Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems. Journal of the Operational Research Society, 2005, 56(6):687–698

[75] Ying K C, Lin S W, Huang C Y. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Systems with Applications, 2009, 36(3):7087–7092

[76] Cicirello V A, Smith S F. Enhancing stochastic search performance by value-biased randomization of heuristics. Journal of Heuristics, 2005, 11(1):5–34

[77] Anghinolfi D, Paolucci M. A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. International Journal of Operations Research, 2008, 5(1):1–17

[78] Lin S W, Ying K C. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. The International Journal of Advanced Manufacturing Technology, 2007, 34(11-12):1183–1190

[79] Cicirello V A. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. in: Proceedings of Proceedings of the 8th annual conference on Genetic and evolutionary computation. ACM, 2006, 1125–1132

[80] Valente J, Alves R A. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. Computers & Operations Research, 2008, 35(7):2388–2405

[81] Anghinolfi D, Paolucci M. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. European Journal of Operational Research, 2009, 193(1):73–85

[82] 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. Computers & Operations Research, 2009, 36(6):1900–1915

[83] Kirlik G, Oguz C. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. Computers & Operations Research, 2012, 39(7):1506–1520

[84] Lourenco H R, Martin O, Stützle T. A beginner's introduction to iterated local search. in: Proceedings of Proceedings of MIC'2001-Meta-heuristics International Conference, 2001, 1–6

[85] Lourenço H R, Martin O C, Stützle T. Iterated local search. Springer, 2003

[86] Stützle T. Applying iterated local search to the permutation flow shop problem. FG Intellektik, TU Darmstadt, Darmstadt, Germany, 1998

[87] Nawaz M, Enscore Jr E E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 1983, 11(1):91–95

[88] Zhang C, Li P, Guan Z, et al. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers & Operations Research, 2007, 34(11):3229–3242

[89] Zhang C Y, Li P, Rao Y, et al. A very fast TS/SA algorithm for the job shop scheduling problem. Computers & Operations Research, 2008, 35(1):282–294

[90] Martin O, Otto S W, Felten E W. Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters, 1992, 11(4):219–224

[91] Wolpert D H, Macready W G. No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on, 1997, 1(1):67–82

[92] Rayward-Smith V J, Consultants U. Modern heuristic search methods. Wiley New York, 1996

[93] Grosan C, Abraham A. Hybrid evolutionary algorithms: methodologies, architectures, and reviews. in: Proceedings of Hybrid evolutionary algorithms, pages 1–17. Springer, 2007

[94] Li F, Morgan R, Williams D. Hybrid genetic approaches to ramping rate constrained dynamic economic dispatch. Electric power systems research, 1997, 43(2):97–103

[95] Lo C C, Chang W H. A multiobjective hybrid genetic algorithm for the capacitated multipoint network design problem. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2000, 30(3):461–470

[96] Somasundaram P, Lakshmiramanan R, Kuppusamy K. Hybrid algorithm based on EP and LP for security constrained economic dispatch problem. Electric Power Systems Research, 2005, 76(1):77–85

[97] Tseng L Y, Liang S C. A hybrid metaheuristic for the quadratic assignment problem. Computational Optimization and Applications, 2006, 34(1):85–113

[98] Sinha A, Goldberg D E. A survey of hybrid genetic and evolutionary algorithms. IlliGAL report, 2003, 2003004

[99] Goldberg D E. Genetic algorithms in search, optimization and machine learning. Addison-Wesley, 1989

[100] 陈国良, 王煦法, 庄镇泉, 等. 遗传算法及其应用. 北京: 人民邮电出版社, 1996

[101] 云庆夏. 进化算法. 北京:冶金工业出版社, 2000

[102] Baker J E. Reducing bias and inefficiency in the selection algorithm. in: Proceedings of Proceedings of the second international conference on genetic algorithms, 1987, 14–21

[103] 张文修, 梁怡. 遗传算法的数学基础. 西安: 西安交通大学出版社, 2000

[104] Nagata Y. Edge Assembly Crossover. A High-power Genetic Algorithm for the Traveling Salesman Problem. in: Proceedings of Proceedings of the Seventh International Conference on Genetic Algorithms, 1997, 450–457

[105] Galinier P, Hao J K. Hybrid evolutionary algorithms for graph coloring. Journal of combinatorial optimization, 1999, 3(4):379–397

[106] Moscato P, Cotta C, Mendes A. Memetic algorithms. in: Proceedings of New optimization techniques in engineering, pages 53–85. Springer, 2004

[107] Lü Z, Hao J K. A memetic algorithm for graph coloring. European Journal of Operational Research, 2010, 203(1):241–250

[108] Xu Y, Qu R. Solving multi-objective multicast routing problems by evolutionary multi-objective simulated annealing algorithms with variable neighbourhoods. Journal of the Operational Research Society, 2011, 62(2):313–325

[109] Benlic U, Hao J K. A multilevel memetic approach for improving graph k-partitions. Evolutionary Computation, IEEE Transactions on, 2011, 15(5):624–642

[110] Falkenauer E, Bouffouix S. A genetic algorithm for job shop. in: Proceedings of IEEE international conference on proceedings of robotics and automation. IEEE, 1991, 824–829

[111] Syswerda G. Schedule Optimization Using Genetic Algorithms. Handbook of Genetic Algorithms, 1991, pages 332–349

[112] Jones T, Forrest S. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. in: Proceedings of Proceedings of the 6th International Conference on Genetic Algorithms. Citeseer, 1995, 184–192

[113] Boese K D. Cost versus distance in the traveling salesman problem. UCLA Computer Science Department, 1995

[114] Reeves C R. Landscapes, operators and heuristic search. Annals of Operations Research, 1999, 86:473–490

[115] Merz P, Freisleben B. Memetic algorithms and the fitness landscape of the graph bi-partitioning problem. in: Proceedings of Parallel Problem Solving from Nature－PPSN V. Springer, 1998, 765–774

致　　谢

首先要感谢我的导师吕志鹏教授。本学位论文从选题、研究、撰写、修改和定稿的全过程都是在吕老师的悉心指导下完成的。在学术上，吕老师渊博的知识和严谨求实的治学态度时刻影响着我，让我受益良多；在工作和生活上，不论我们碰到什么困难和挫折，吕老师都会尽自己最大的努力给予鼓励和帮助。因为吕老师，让我真正体会了“良师益友”这四个字的含义。在此，向吕老师致以最衷心的感谢和最崇高的敬意！

感谢我的丈夫欧阳泉。是他在几乎所有人都不理解不支持我在33岁选择读博这个决定的情况下，义无反顾的站在我这边，给我温暖和力量。在我求学的这几年中，承担了所有本该我承担的家务；在工作中，因为我们是同事的便利，尽量帮我减轻负担；学习上，也经常给我一些好的建议和帮助。套用一句歌词最能表达我的感激之情，那就是“军功章啊，有我的一半，也有你的一半！”。

感谢我的父亲母亲在我求学过程中给予的一贯的理解、支持和帮助。正是他们给予了我前进的动力和战胜困难的勇气，让我一路走到了现在。

感谢金人超教授、许如初副教授和计算机科学理论研究所的各位老师多年来对我的指导和帮助。还要感谢计算机科学与技术学院和研究生院的各位老师和领导的关心和爱护。

感谢实验室的师兄弟们，他们是周淘晴、彭博、吴歆韵、郭琦、王卓、谢龙恩等；感谢我的同班同学江华、刘燕丽、刘永川等。和他们一起讨论问题使我受到不少启发，同时，他们还在不同时期给予了我各种不同的帮助。博士生活因为有了他们的陪伴，也增添了许多乐趣。

感谢所有曾经给予过我关心和帮助的老师、同学、亲朋、同事！在此真诚的祝愿他们身体健康，生活幸福！

最后，衷心感谢各位评委老师的批评和指导！

附录 1　攻读学位期间发表论文目录

[1] Xu Hongyun, Lü Zhipeng, Yin Aihua, Shen Liji, Buscher Udo. A study of hybrid evolutionary algorithms for single machine scheduling problem with sequence-dependent setup times. Computers & Operations Research, 2014, 50: 47-60. (SCI索引)

[2] Xu Hongyun, Lü Zhipeng, Cheng TCE. Iterated Local Search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. Journal of Scheduling, 2014, 17(3): 271-287. (SCI索引)

[3] 钟涛, 萧卫, 徐宏云(通信作者), 刘广, 崔珊珊. 带准备时间的单机调度问题的混合进化算法研究. 计算机应用研究, 2013, 30(11): 3248-3252.

附录 2　攻读博士学位期间参与的科研项目

[1] 国家自然科学基金青年基金项目“求解大规模约束满足问题的混合进化算法研究”， 项目编号：61100144，在研。

[2] 国家自然科学基金面上项目“光网络规划流量疏导优化”， 项目编号：61370183，在研。

[3] 教育部博士点基金（新教师类）项目“面向频率分配问题的混合算法研究”，项目编号：20110142120081，在研。

[4] 教育部“新世纪优秀人才支持计划”，在研。