分类号 学号 M201572797

学校代码 10487 密级



**硕士学位论文**

**求解单机调度问题的多扰动迭代启发式算法研究**

|  |  |
| --- | --- |
| 学位申请人： | 覃 涛 |
| 学科专业： | 计算机技术 |
| 指导教师： | 吕志鹏 教授 |
| 答辩日期： | 20180522 |

**A Dissertation Submitted in Partial Fulfillment of the Requirements**

**for the Degree of Master of Engineering**

**Iterated Local Search Based on Multi-type Perturbation for Single-Machine Scheduling Problem**

|  |  |
| --- | --- |
| **Candidate :** | **Zhang Junchen** |
| **Major :** | **Computer Technology** |
| **Supervisor :** | **Prof. Lv Zhipeng** |
|  |  |

**Huazhong University of Science & Technology**

**Wuhan 430074, P. R. China**

**May, 2018**

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个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。

保密□，在 年解密后适用本授权书。

本论文属于

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

考虑关键边代价最小化的旅行商问题（简称BTSP）是最基本的路径优化问题，在物流规划、交通规划和网络规划中都有着非常重要的实际应用。已经证明，在计算复杂性理论中，BTSP是NP难问题。所以，对BTSP的求解具有重要的理论研究价值和实际应用价值。

一般地，求解BTSP问题的方法主要有三种：精确算法、下界算法和启发式算法。精确算法能保证得到最优解，但由于问题的计算复杂性，导致在实际应用中很难求解大规模的问题实例；下界算法虽然能给出最优解的大致范围，但往往不能满足实际需求；而启发式算法，来源于对大自然的思考，力求在计算时间和求解质量上达到一种平衡，能在合理的时间内，找到比较优质的解，甚至达到最优解。将所研究的环路问题转化为有序的线性序列问题，并首次将块移动的策略应用到了瓶颈旅行商问题中，提出了求解BTSP的迭代禁忌搜索算法（简称ITS）。然后，利用标准测试算例对算法进行了性能测试，结果表明，ITS算法在求解这一问题上具有较好的计算性能，在绝大多数算例上都能得到最优解，而且相较于文献中的算法，无论从求解速度还是从解的质量上都有较显著的优势。对于难以求解的复杂算例，略微调整评估策略和计算参数，重新计算并得到了最优解，最终在所有的测试算例中，仅仅只有两个算例无法获得最优解。

关键词：NP难问题；旅行商问题；启发式算法；块移动；禁忌搜索

# Abstract

The bottleneck traveling salesman problem (BTSP) is one of the most basic routing problems with significant application in logistics planning, traffic planning and network planning. It has been proved that BTSP is NP-hard in computational complexity theory. Therefore, solving BTSP has both theoretical and practical research values.

In general, there are three kinds of methods for solving BTSP: exact algorithm, lower bound algorithm and heuristic algorithm. The exact algorithms aim at obtaining optimal solution but it is difficult to solve the large-scale practical instance because of the computational complexity of the problem. Although the lower bound algorithm can get the approximate scope of optimal solutions, it always cannot meet the practical demand. Heuristic algorithms, derived from the reflection of nature and strive to achieve the balance of computation speed and solution quality, can obtain excellent or even optimum solutions in a reasonable time. This cycle problem is converted to an orderly linear problem and the block move strategy has been applied to bottleneck traveling salesman problem for the first time. Then an iterated tabu search (ITS) algorithm is proposed for solving BTSP. Afterwards, the performance of this proposed algorithm was tested by using the benchmark instances. The results show that ITS has very outstanding computing performance and can obtain optimal solutions in most cases. Compared to the other published reference algorithms, ITS has very significant advantage in terms of the computation speed or solution quality. In addition, as for the difficult instances, these problems are retackled and product optimal solutions by slightly adjusting the evaluation strategy and setting of the parameters. Finally, there are just two cases for which we cannot get the optimal solution among all eighty six test instances.

**Key Words:** NP-hard problem; traveling salesman problem; heuristic algorithm; block move; tabu search

**目录**

[摘 要 I](#_Toc421198709)

[Abstract II](#_Toc421198710)

[1 绪言 1](#_Toc421198711)

[1.1 课题背景和意义 1](#_Toc421198712)

[1.2 国内外研究概况 2](#_Toc421198713)

[1.3 本文工作及结构安排 5](#_Toc421198714)

[2 考虑关键边最小化的TSP 7](#_Toc421198715)

[2.1 问题概述 7](#_Toc421198716)

[2.1.1 问题定义 7](#_Toc421198717)

[2.1.2 TSP分类 9](#_Toc421198718)

[2.2 BTSP数学模型 10](#_Toc421198719)

[2.3 一般求解思路 11](#_Toc421198720)

[3 迭代禁忌搜索算法 13](#_Toc421198721)

[3.1 算法主体框架 13](#_Toc421198722)

[3.2 构造初始解 14](#_Toc421198723)

[3.2.1 纯随机方式 15](#_Toc421198724)

[3.2.2 纯贪心方式 15](#_Toc421198725)

[3.2.3 “不坏”原则 16](#_Toc421198726)

[3.3 禁忌搜索 17](#_Toc421198727)

[3.3.1 伪代码描述 17](#_Toc421198728)

[3.3.2 邻域动作 18](#_Toc421198729)

[3.3.3 邻域评估 23](#_Toc421198730)

[3.3.4 禁忌方式 25](#_Toc421198731)

[3.3.5 解禁策略 27](#_Toc421198732)

[3.4 扰动机制 27](#_Toc421198733)

[4 实验与结果分析 29](#_Toc421198734)

[4.1 测试方案设计 29](#_Toc421198735)

[4.1.1 测试算例 29](#_Toc421198736)

[4.1.2 实验环境 30](#_Toc421198737)

[4.2 参数设定 30](#_Toc421198738)

[4.2.1 所有参数设置 30](#_Toc421198739)

[4.2.2 块长测试 31](#_Toc421198740)

[4.3 算法性能分析 32](#_Toc421198741)

[5 结论与展望 39](#_Toc421198742)

[致谢 40](#_Toc421198743)

[参考文献 41](#_Toc421198744)

[附录Ⅰ 英文缩写词 44](#_Toc421198745)

[附录Ⅱ 发表论文情况 45](#_Toc421198746)

# 绪言

## 课题背景和意义

组合优化问题是求解最优值问题的一种，主要解决的是离散型变量的问题，其一般形式可以描述为：在满足一定的约束或者限制条件下，求取使得给定的目标函数值最大或者最小的解。组合优化问题主要应用于对资源的管理分配和高效利用问题，期望找到一种较为合理的分配和调度方案以减少开支或者提高效益，其解决的实际问题主要包括运筹问题中的货物分配、生产调度和机器排序，还有一些规划问题中的资本预算、设施选址和投资组合分析，以及设计问题中的通信和交通网络设计、超大型积体电路设计以及自动化生产系统的设计等等[[1](#_ENREF_1)]。这类问题往往由于约束条件复杂，搜索的解空间过于庞大，求解相对困难，依靠人工的方式很难得到令人满意的解决方案。由于组合优化的研究对象是离散型变量，而计算机所能处理的数据类型也是离散型的，现代计算机技术的进步和算法原理的发展极大地促进了组合优化问题的研究工作。

旅行商问题(traveling salesman problem)又译为[旅行推销员问题](http://baike.baidu.com/view/614849.htm)、[货郎担问题](http://baike.baidu.com/view/267558.htm)，简称为[TSP问题](http://baike.baidu.com/view/45957.htm)，是最基本的路径优化问题，也是一类经典的组合优化问题，它在道路交通规划、物流规划和网络规划中都有非常重要的应用。按照字面上的理解：有一个推销员，要到个城市贩卖商品，需要找到一个经过所有个城市的最短路径环，这就是旅行商问题最原始的定义。区别于经典的TSP问题，本论文研究的是考虑关键边代价的TSP问题(bottleneck TSP)，简称BTSP[[2](#_ENREF_2)]，其基本的问题定义与TSP一致，只是求解目标不同，BTSP意图在所有可能的解中找到这样一条环路，使得路径中的最大边的权值最小。BTSP是组合优化中最大最小问题的典型实例[[3](#_ENREF_3)]，虽然约束规则比较简单，但问题的目标是要在所有可能的情况中找到最优的路径，所以原则上需要考虑所有可能的解，确定某一个起点（虽然说环上任意一点都可以作为起点），搜索空间是个点的全排列的集合，这个全排列集合中元素的数量为，所以该问题的计算复杂度达到了*O*(*n*!)。

已经证明，BTSP是典型的NP难问题。NP难度问题广泛地出现在管理科学、计算机科学、通信领域、计算生物学等学科中，大多数组合优化问题都是NP难问题，如单机调度问题、旅行商问题、排课表问题、图着色问题以及蛋白质结构预测问题等等。国内外的科学家们对于BTSP已经进行了很长时间的研究，并提出了大量的求解算法，既有可以得到最优解的精确算法，也有求解问题下界的定界算法，另外还有启发式算法。对于小规模的这类问题，利用精确算法找到其最优解是有可能的，时间代价也可以接受，但是对于较大规模的NP难问题，即使是当前世界上最先进的超级计算机，其计算时间也是一个天文数字！意图通过精确算法来找到问题的最优解显然是不切实际的。

受大自然的启发，人们从大自然的运行规律中找到了许多解决实际问题的方法，对于那些受大自然的运行规律或者面向具体问题的经验、规则启发出来的方法，人们常常称之为启发式算法。一般地，通过精确算法来得到NP难问题的最优解会耗费相当大的时间代价，依靠现在的计算机技术很难完成，所以人们大多是利用某种近似算法来求解这类问题，得到NP难问题的近似解。启发式算法就是其中非常具有代表性的一种，它可以在合理的时间内，得到优度较好的解，甚至有可能得到最优解。

一般地，启发式算法不具有普遍性，对于不同的问题，其解空间结构的复杂性有很大的差异，因此往往人们会根据不同的问题设计相对应的高效启发式算法，本篇论文研究的目的也是期望找到适合解决BTSP问题的高效启发式智能算法。由于BTSP问题在诸如物流规划和工艺生产等领域有广泛的应用，解决这一问题会带来巨大的经济效益，具有很高的现实意义；同时，我们研究的BTSP的数学模型属于算法复杂性理论中非常困难的NP难问题，设计求解该问题的启发式算法对于解决NP难问题具有很高的理论研究价值。

## 国内外研究概况

BTSP最早是由Gilmore和Gomory于1964年提出的，并给出了一些特殊BTSP算例的解决方案，同时他们还提到，经典的哈密顿回路问题也可以归纳到BTSP，只需要将节点间距离的值限制在0或者1，求得的目标函数值为0的解就是哈密顿回路的解[[4](#_ENREF_4)]。它是一类已被证明的NP难问题，关于其计算复杂度的详细讨论可以参考Kabadi和Punnen的著作[[5](#_ENREF_5)]。一般地，关于解决BTSP的算法分为三种，第一种是精确算法，试图通过某种策略快速找到最优解，节省计算时间；第二种是下界算法，即找到问题算例的下界，一般其结果作为判断某些大规模算例结果优劣性的依据；第三种是启发式算法，利用启发式方法来解决BTSP，可以在比较合理的时间内得到目标函数值比较好的解决方案。

1. **精确算法**

1978年，Garfinkel和Gilbert提出了一种基于分支定界(BB)的精确算法来解决非对称BTSP问题[[6](#_ENREF_6)]，并利用该算法对满足其定义的某个密度函数的随机非对称TSP算例进行了测试，算例的规模在10~100个节点之间。由于该精确算法的时间复杂度较高，对于某部分算例仍然无法计算得到最优解，对于已经解决的算例，往往也需要较大的时间代价。1984年，Carpaneto等人继续研究了分支定界算法，他们在分支决策树的每个节点运用启发式搜索来寻找一些不含有超过当前下界的弧的哈密顿圆环，以便减少搜索的节点，降低搜索时间。他们将测试算例的规模提高到了200个节点，并发现这种算法无论是对对称的还是非对称的BTSP都有很好的计算效果[[7](#_ENREF_7)]。另外，Sergeev[[8](#_ENREF_8)]在1995年提出了求取小规模BTSP精确解的动态规划方法，但这种方法依然不适用于节点数量很多的大规模算例。一般来说，精确算法虽然能得到规模不大的问题的最优解，但往往实际生活中，问题规模都很大，精确算法的适用性就不能得到保证，所以关于这方面的研究工作都仅仅只停留在小规模算例上，而且求解时间一般都比较长，在现实生活中的应用受到限制。

1. **下界算法**

对于某些节点数量较多的算例，因为很难有精确算法可以求取到最优解，为了衡量BTSP算法的性能，人们往往将计算结果与下界相比较，把差值作为算法优越性判断的标准；而且，如果某种算法能计算到下界值，也可以证明这个解是最优解。正由于这些原因，多种类型的下界算法应运而生。在本文中，下界算法的原理不是我们讨论的重点，这里只列举几个主要的算法：一种是通过寻找最小的“出边”和“进边”代价来选择所有节点中的最大值来求取BTSP问题的下界，称为*2-Max*定界算法（2MB）[[7](#_ENREF_7)]，其算法复杂度为)，其中表示边的数量；另一种被称为BAPB的算法是将求取BTSP下界的问题转化为求取分配瓶颈问题（bottleneck assignment problem，简称BAP）最优目标函数值的问题[[6](#_ENREF_6)]；另外，双连通子图瓶颈问题（BBSSP）的最优解也可以作为BTSP问题的下界，Punnen提出了瓶颈强连通子图问题下界算法[[3](#_ENREF_3)]，Carpaneto等人提出了利用双向关键路径问题（BBP）求取下界的方法[[7](#_ENREF_7)]。最近LaRusic和Punnen等人对这些下界算法进行了归纳整理和比较，并在BBP下界算法的基础上提出了增强的BBP算法，称为EBBP[[9](#_ENREF_9)]。下界算法虽然不一定得到最优解，但是好的下界完全可以作为评估解的优劣性的依据，而且很多学者研究的启发式算法都利用了下界作为算法中不可或缺的一部分。

1. **启发式算法**

研究表明，启发式算法是非常适用于解决NP难问题的一类算法，它能够在可以接受的时间代价内得到质量较好的解，甚至有可能就是最优解，目前它已经在诸如调度问题、图着色问题、频率分配问题以及packing问题等都有了很好的应用。解决BTSP问题的启发式算法，最早是Garfinkel[[6](#_ENREF_6)]在分支定界算法的基础上实现的旅程构建算法（tour building approach），该算法对大多数随机生成的节点数不超过100的问题都能算到最优解。1979年，Timofeev[[10](#_ENREF_10)]研究了规模相似的非对称BTSP问题，并提出了基于2-连通子图(这个子图包含问题图的所有节点，复杂度为最小最大问题结果的启发式算法，测试结果表明该算法可以求解到所有60个伪随机算例的最优解。2009年，Ramakrishnan等人将一种阈值算法应用到了72种对称TSP，问题的规模也提高到了最多738个城市[[11](#_ENREF_11)]。2010年，Ahmad利用一种基于词典搜索的算法来解决了多个典型的对称的以及非对称BTSP问题，利用独创的字母表（Alphabet Table）结构，测试了节点数量在17~70之间的TSPLIB算例以及随机生成的节点数量在30~50之间的算例，大多数都能在可以接受的时间内得到最优解[[12](#_ENREF_12)]。随后他又将这种算法应用到了非对称性TSP问题，算例规模不超过100个节点，并提出了一个称为“*data-guided*”的数据预处理方案，很大程度上提高了计算质量和计算速度，相比其它已知算法，运行结果也表现了很大的优越性[[13](#_ENREF_13)]。后来，LaRusic等人[[14](#_ENREF_14)]扩展了对称性BTSP的测试结果，问题规模也提高到了最多3162个节点，其利用下界的启发式算法对包括TSPLIB，Johnson-McGeoch随机算例以及VLSI和National TSP算例在内的所有相对简单的算例，都能在合理的时间内得到最优解；对于一些比较难的特殊算例，其算法也能得到大多数的最优解。就在最近，LaRusic和Punnen提出了应用到非对称BTSP的启发式算法[[9](#_ENREF_9)]，它与阈值算法很类似，主体的框架是二分策略，然后在求取的下界和当前解之间不断作可行性判断，直到无法找到最好的解为止，结果显示该算法得到了所有331个算例中270个最优解。

这三种算法都有各自的优点和不足：精确算法可以保证求到最优解，但它们意味着更高的时间代价，对于问题规模稍微大一些的，根本就束手无策，甚至对一些特殊结构的小规模算例也无能为力；下界算法顾名思义就是得到问题最优解的下界，这个结果不仅可以作为评估某个近似算法计算结果优劣性的依据，而且如果一旦达到这个下界，则可以认为是找到了问题的最优解，但问题是，对于一些新的问题，我们无法去衡量该下界算法的好坏，因为无从比较；相对算法性能比较平衡的应该是启发式算法，它虽然不保证能得到最优解，但它可以在一个合理的时间内找到相对较好的解，这对于大规模算例也同样适用。启发式算法往往带有一定的随机性，几次计算的结果可能会有差别，而且对于不同的问题，即使仅仅是目标函数的差别，算法表现的性能也是千差万别。当然，这也正是科学家们研究的重点。

## 本文工作及结构安排

本文从传统TSP问题的背景出发，逐步引出求解目标不同的BTSP的具体定义和数学模型，并首次将块移动策略应用到该问题中，利用迭代禁忌搜索算法求解BTSP。同时我们还通过标准算例，和参考文献中的算法[[9](#_ENREF_9)]对比，分析出我们算法的优越性和不足。另外，本文还对我们的研究工作做了总结，并提出了今后更高层次的研究方向。

第一章详细介绍了本课题研究的目的和意义，并从精确算法、下界算法和启发式算法三个方面介绍了国内外研究发展的现状。

第二章中我们以传统的TSP为基础，描述了不同种类的TSP问题，并给出了BTSP的基本定义，同时，利用数学符号和数学公式，给出了BTSP完整的数学模型。另外，我们还给出了求解TSP问题的一般求解思路。

第三章我们详细描述了在本文中提出的迭代禁忌搜索算法，并从邻域搜索和禁忌策略两个方面阐述了算法的核心部分，尤其是邻域搜索中的局部动作，我们分析了非对称BTSP问题的特殊性，并将哈密顿环问题转化为线性序列问题，从点的插入动作逐步引出块插入的邻域动作。另外，算法中初始解的构造方式和扰动机制我们也给出了说明。

第四章主要是实验结果的分析。首先，通过实验测试，确定了算法中各个参数的实际值，本文以块长为例给出了分析过程。然后，我们测试了所有86个有代表性的标准算例，并与前人的算法作了比较和分析。另外，我们还针对比较困难的算例，调整了小部分策略和参数，得到了更好的计算结果。

第五章主要是对本论文研究工作的总结和评述，分析了我们设计的算法中的优势和不足，并对今后的研究工作做了展望。

# 考虑关键边最小化的TSP

## 问题概述

TSP，即旅行商问题，或称为货郎担问题，是组合优化问题中非常典型的一种，考虑关键边最小化代价的TSP（即BTSP）是旅行商问题的一种分类。已经证明，该问题是NP难的，至今还找不到一个有效的多项式时间复杂度算法来求解这一问题，除非P=NP成立。虽然该问题的求解复杂度较高，但它在实际生产生活中有很大程度的应用，如旅游线路规划问题、物流配送问题、电路板设计钻孔问题以及DNA测序等等。下面给出BTSP的详细定义和常规求解方法。

### 问题定义

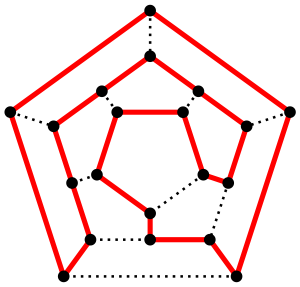


图2.1 正十二面体中的哈密顿路径（红色）

十九世纪，天文学家哈密顿(William Rowan Hamiton)以游戏的形式提出了著名的哈密顿问题：把一个正十二面体的二十个定点看成二十个城市，要求找出一条经过每个城市恰好一次而回到出发点的路线，如图2.1所示。在图论中，哈密顿图是一个无向图，由指定的起点前往指定的终点，途中经过所有其它节点且只有一次，这样包含所有顶点的路径称作哈密顿路径，闭合的哈密顿路径称为哈密顿环（或哈密顿回路）。TSP问题与之稍有不同，哈密顿问题是在研究的问题图中找到可以连通的方案即可，而TSP研究的一般是带权的完全图，有效的哈密顿环是一定存在的，其求解目标是在所有的哈密顿回路中找到总路径长度最小的一种，而本论文中的BTSP则是找到满足条件的路径中最大边的权值最小的哈密顿回路。



图2.2 四个节点TSP示例

图2.2给出的是四个节点的TSP示例，包含A~D这四个顶点，每两点之间都有一条边，它是四个顶点的完全图，边上的数字标注的是构成该边的两点之间的距离值。从图中可以看出，这个示例表示的是对称的无向图，两点之间的距离与点的先后顺序无关，也就是说这个问题是对称型的TSP。我们规定问题以A为起点，要求路径经过其它所有的三个点并最终回到点A，所有可能的方案以及相应的目标函数值如表2.1所示，总路径和表示路径包含的边的距离总和，最大边表示路径中距离最长的边的值。从表中可以看出，对于包含四个节点的TSP问题，其可能的路线一共有种，但由于图是无向图，满足对称性，所以实际一共只有3条目标函数值不一样的路径：比如方案1和方案6，虽然路径的走向不一样，但它们

表2.1 四个节点TSP示例所有路径分析

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 方案 | 总路径和 | 最大边 |
| 1 |  | 69 | 24 |
| 2 |  | 78 | 24 |
| 3 |  | 75 | 22 |
| 4 |  | 78 | 24 |
| 5 |  | 75 | 22 |
| 6 |  | 69 | 24 |

是对称的，函数值没有差别。在这所有可能的路径中，显然方案1和方案6是路径总和最小的方案，而方案3和方案5则是满足最大边的值最小的解。所以，虽然问题的定义一样，但目标函数不一样，TSP和BTSP的最优解是有差别的，适用于TSP问题的算法和数据结构不一样适用于我们研究的BTSP问题，因此，找到适合BTSP的智能启发式算法就很有必要了。

### TSP分类

旅行商问题是一类基本问题，按照约束条件或者目标函数的不同可以分为以下的类型：

* **按目标函数分**
  + - * 最小最大TSP，旨在求解得到路径总和最小（MIN TSP）或者最大(MAX TSP)的哈密顿环路。
* 瓶颈TSP（bottleneck TSP），求解目标是找到这样一条路径，使得路径中的最大边的值最小，或者使得最小边的值最大。
* 平衡TSP[[15](#_ENREF_15)]（balanced TSP），它与瓶颈TSP较为类似，顾名思义，问题是希望路径中所有的边较为平均，目标是最大边与最小边的差值最小。
* 允许多访问的TSP[[16](#_ENREF_16)]（TSP with multiple visits），它要求每个节点最少通过一次（也可以理解为允许对节点的多次访问），使得总的路径和最小，这与“中国邮差问题”的定义是一致的。
* **按距离矩阵分**
* 对称TSP（symmetric TSP），研究的是无向完全图，图中任意两点之间的边的距离与方向无关，满足对称性。
* 非对称TSP（asymmetric TSP），研究的是有向完全图，图中存在着两点之间因为起点终点顺序不同，边的距离也不同的情况，图不是对称的。一般地，非对称TSP也可以转化为对称TSP[[17](#_ENREF_17)]。
* 三角型TSP[[18](#_ENREF_18)]（triangle inequality TSP），图中任意三点之间的距离满足三角不等式关系，这也符合欧式空间几何的基本原理，所以这种问题也被称为几何TSP。

除此之外，还有很多各种不同类型的TSP，这里就不一一列出了，有兴趣的看可以参考Gutin和Punnen的著作[[2](#_ENREF_2)]。在本论文中，我们研究的是考虑关键边代价最小化的TSP，也就是瓶颈TSP中使得最大边的距离最小的一种，测试的算例是非对称的有向图，既包括满足三角不等式的TSP，也包括距离矩阵满足其它条件或者完全随机的类型。

## BTSP数学模型

在图论中，区别于哈密顿回路问题，旅行商问题TSP一般被描述为一个带权完全图，其中表示图中所有节点的集合，也就是旅行商问题中描述的城市，总的数量为；表示图中所有边的集合，所有边的数量记为，即。因为是完全图，所以对于任意，都有。同时，伴随集合的有一个距离矩阵(cost matrix)，即表示边的路程代价，也就是节点到节点的距离，一般满足且。旅行商问题的合法解是条边的集合，该集合中的所有条边必须包含所有的个顶点，也就是“哈密顿环”，设

(2.1)

由于限制条件要求满足经过除顶点外的所有的点有且仅有一次，则哈密顿环可以用如下的数学模型表示：

s.t

(2.2)

如公式(2.2)所示，前两个约束保证了对于图G中的每一个顶点，入度和出度均为1，后一个约束中，表示集合的所有非空子集，表示S中包含的顶点个数，它保证了路线中没有任何子回路的产生，这些加入到路线中的边会形成一条包含所有顶点的闭合环路，也就是哈密顿环H。定义所有的哈密顿环的集合为，一般的旅行商问题是要在中找到使得路径总和最小的哈密顿环H，但在本论文中，我们研究的问题考虑的是关键边的代价，也就是让H中最大边的值最小，所以我们研究的问题可以定义如下：

Minimize

(2.3)

Subject to

这种考虑关键边代价最小化约束的旅行商问题我们一般称为瓶颈旅行商问题（bottleneck travelling salesman problem），简称BTSP[[2](#_ENREF_2)]。BTSP的合法解哈密顿环H是边的集合，但实质上仍然也可以理解成*n*个节点的有序集合，所以可以采用有序集合来表示问题的解。一般研究的BTSP分为两种类型，一种是对称性BTSP，研究的图是无向的，对于其中任意的节点，二者之间的距离；另一种是非对称BTSP，它所研究的图是有向图，存在一些节点，有，不满足对称性。在本论文中，我们所研究的问题是非对称性BTSP。应该来说，非对称的BTSP相对是要复杂一些，从图2.2的示例也可以看出，非对称的BTSP的搜索空间为对称BTSP的两倍！

## 一般求解思路

通过前面的分析我们知道，旅行商问题属于典型的NP困难问题，常规解法的计算复杂度很高，所以旅行商问题大多集中在启发式算法，一般的启发式算法可以分为以下三种[[19](#_ENREF_19)]：

* **途程构建法（Tour Construction Procedures）**

途程构建法一般从一个非法解开始，然后分析距离矩阵信息数据，并应用某种增广策略逐步产生符合要求的合法解，这类算法主要有：

1. 最近邻点法（Nearest Neighbor Procedure）：从起点开始，在剩下的节点中依次选择到路线的终点最近的点，直到所有的点都加入到了路线中为止，这是一种纯贪心原则构造的解。
2. 节省法（Clark and Wright Saving）：从起点向其它个点构造线路并回到起点，在其它除起点之外的任意两点之间构造线路的路程，通过选择节省量最大的连接方式来逐步连接线路，直到加入了所有的点。
3. 插入法（Insertion procedures）：如最近插入法、最省插入法、随意插入法、最远插入法、最大角度插入法等，即生成某段子线路后，将剩下的每个节点依次插入到符合规则的最佳位置，直到路线生成完整。

* **途程改善法（Tour Improvement Procedure）**

途程改善法是先给定一个可行路径，然后进行改善，通过某种动作微调将问题的目标函数值不断优化，一直到不能改善为止。主要有以下几种解法：

1. K-Opt(2/3 Opt)：把尚未加入路径的K条节线暂时取代目前路径中K条节线，并计算其成本（或距离），如果成本降低，则取代之，直到无法改善为止，K通常为2或3。
2. Or-Opt：在相同路径上相邻的需求点，将之和本身或其它路径交换且仍保持路径方向性，并计算其成本（或距离），如果成本降低（距离减少），则取代之，直到无法改善为止。

* **合成启发法（Composite Procedure）**

先由途程建构法产生起始途程，然后再使用途程改善法去寻求最佳解，又称为两段解法(two phase method)，类似于两种方式的结合，即在途程改善法之前就用途程构建法得到比较好的初始解，然后从这个好的解出发去优化。这也是我们求解BTSP问题所准备采取的方法。

# 迭代禁忌搜索算法

在启发式算法中，迭代禁忌搜索是应用比较广泛的一种，虽然它属于比较简单的一类启发式算法，但在求解单机调度问题、护士排班表、无约束二次规划问题以及识别复杂网络中群落结构中都有很大程度的应用，并且得到了非常理想的计算效果[[20-23](#_ENREF_20)]。

禁忌搜索最早是Glover在1986年提出的[[24](#_ENREF_24)]，它是一种全局性邻域搜索算法，模拟人类具有记忆功能的寻优特征，是局部邻域搜索的一种扩展，其基本的框架也是利用局部搜索从当前解的邻域中找到一个改进的解。由于局部搜索的搜索性能很依赖于邻域结构和初始解，容易陷入局部极小而无法保证全局最优性，也可能回到曾经搜索过的解空间，重复不必要的循环搜索，而采用禁忌搜索就可以很大程度上减少这一问题的出现。禁忌搜索的基本内容与局部搜索类似，但是增加了禁忌准则来避免迂回搜索，并通过藐视准则来释放一些被禁忌的优良状态，进而保证多样化的有效探索，以最终实现全局优化。当然，这仅仅是理论层面上的，当用它来求解我们所研究的瓶颈旅行商问题时，初始解构造、邻域结构、禁忌策略和扰动方式等都要进行详细的设计，本章将详细描述应用于求解BTSP的迭代禁忌搜索算法的具体内容。

## 算法主体框架

求解BTSP的迭代禁忌搜索（iterated tabu search）算法主体框架的伪代码如表3.1所示。算法主要内容是重复迭代利用禁忌搜索和扰动机制来获取最好的解决方案：禁忌搜索通过禁忌机制，不重复地搜索以得到局部最优解；扰动机制用来跳出禁忌搜索到达的“怪坑”，通过解的抛出来扩大搜索空间，增强算法的全局最优性。具体来说，算法首先要求输入问题的规模——也就是节点的个数，以及非对称BTSP的带权完全图的距离矩阵。如表3.1所示，第1行表示生成初始解，这个初始解是问题的合法解，也就是一个哈密顿环，当然，这个解的质量并不一定能得到保证。生成初始解的方式我们有三种：纯随机、纯贪心以及“不坏”原则方式生成的，具体选择哪一种采用的是随机方式，三种方法各自的随机比例有所不同，具体内容将在3.2中加以介绍。第4~11行表示迭代禁忌搜索的主要内容，变量表示迭代的次数，当在代数内，如果没有找到更好的解，或者解已经达到最优解OPT，则算法终止，输出最好结果（第4行）；第5行表示应用禁忌搜索算法不断把当前解更新，如果获得了比当前最好结果还要好的解，则迭代次数计数器会重新置0（6~9行）。每次禁忌搜索到达局部最优后，算法会在当前最好解的基础上应用扰动机制，并将结果作为下一次迭代的起始解（第10行）。禁忌搜索的具体内容，主要包括邻域动作和邻域评估策略以及相应的禁忌方式，可以参考3.3节。另外，扰动机制的具体细节将在3.4节中介绍。

表3.1 求解BTSP的迭代禁忌搜索算法

|  |
| --- |
| **Algorithm 1：**Interated Tabu Search For BTSP |
| **Input：** *n*: number of vertexes， C[*n*][*n*]: cost matrix |
| 1： Generate an initial solution //见3.2节 |
| 2： , |
| 3： |
| 4： **while**  **do** |
| 5： //见3.3节 |
| 6： **if**  is best than **then** |
| 7： |
| 8： |
| 9： **end if**  10： //见3.4节 |
| 11： |
| 12： **end while** |
| **Output：** |

## 构造初始解

初始解是算法最初执行的对象，一般有可行解和非可行解两种，由于我们的禁忌搜索算法是基于途程改善法的，所以生成的初始解是可行解。上一节中提到了，初始解的构造我们设计了三种方式：纯随机、纯贪心以及“不坏”原则方式生成，以一定的比例随机选择三者中的一种作为初始解。下面详细介绍这三种方式的执行过程。

### 纯随机方式

纯随机方式就是以完全随机的方式选择某个节点依次加入到路线中。如2.2节中所描述的，在BTSP的数学模型定义中，用个节点的有序集合来表示问题的可行解，用表示图中点的无序集合，则随机方式初始解的构造过程为：置为空集合，首先随机从集合中选取一个节点加入到解中，也就是将加入到集合中，即，并将从集合中剔除出去，即，重复这一过程，直到节点集合*V*中的所有节点都被加入到路径中，变为空集，这样得到的有序序列就是完全随机方式形成的初始解。一共有个节点，这个随机过程一共有次，因为最后一个节点不用选择就可以直接加入到路径的末尾。BTSP的合法解是形成环的边的集合，而得到的集合是点的集合，所以最终的合法路径是由有序序列中相邻节点形成的边组成的环路，路径的起点即是最早加入到中的节点，当然，同时它也是路径的终点。路径经过的点的顺序和节点加入的时间前后是有关的，节点加入到合法解中的时间越早，代表着从起点出发的路径越早经过该节点。这种方式是完全随机的方式构造的，没有考虑边的大小对目标函数值的影响。

### 纯贪心方式

纯贪心的方式就是以当前边的权值大小作为衡量标准，贪心地选择最小边加入到路径中来形成初始解。同随机方式一样，贪心方式的执行过程也是不断从集合中选择某个节点加入到集合中，直到所有节点都在形成的路径上，唯一的区别在于节点选择的策略，所以将选择的节点加入到*S*中并将其从中删除的过程就不再赘述了。首先置为空集合，这里是以随机的方式选择一个节点作为起点，也就是随机从集合*V*中选取一个节点加入到解中，然后以当前路径的终点作为起点，从集合*V*中的剩余元素中贪心选择最优的节点（也就是满足边的成本最小）加入到当前路径的末尾，然后以这个刚加入的节点作为起点重复执行上面的贪心过程，直到所有的节点都被加入到了中。在这种方式下，除了路径的起始点是随机选择的以外，其它所有节点都是根据边代价贪心选择的，这也是我们BTSP求解目标的局部最优选择。当然，这种局部最优性也有可能会对全局最优性产生很大的坏影响。

### “不坏”原则

“不坏”原则是在贪心方式的基础上演化而来的，因为贪心方式是选择边代价最小的节点，所以即使存在某些边的代价相等，但节点的可选性还是很少。最坏的情况下，对于某些所有的边权值都不同的算例，那么贪心方式形成的初始解所有可能情况最多就只有种，这显然会影响算法搜索的解空间广度。为了找到随机性和局部优劣性之间的平衡点，我们设计了这种初始解构造方式，其基本执行过程和贪心方式是一样的，区别只是在于为当前路径寻找下一个后续节点的时候，评估准则不同：贪心方式是选择以当前路径终点作为起点和尚未添加到当前路径的节点作为终点形成的边中代价最小的构造点，而“不坏”原则评估选择加入到路径的新节点的时候，只要新增的边的代价没有超过当前路径的最大边（也就是目标函数值），这些节点都可以作为候选点，随机选择其中的一个添加到当前路径的末尾。由于当前路径包含的节点数越多，余下的节点的数量就越少，可能某个时间之后已经找不到一条不超过当前目标函数值的边，这时候我们就考虑在剩下所有可能的情况中，贪心选择代价最小的那条边的点作为新加入的点，并将这条边的值作为新的目标函数值。

这三种构造初始解的方式各有其优缺点，纯随机的方式随机性最强，能将初始解放在解空间的任意位置，可能与最优解距离非常值近，当然也可能和最优解相距的非常远，这都是不可预测的，就好像“有希望，但是希望很渺茫”，属于小概率事件；纯贪心的方式构造初始解的每一步都考虑到了目标函数值的优劣性，这种方式形成的解虽然原则上一般不能得到最优解，而且越到路径的后面部分，边代价可能更大，但是存在很大的可能，解上面的一部分和最优解是很相似的，依靠后续的搜索策略很有可能找到最好的解决方案；“不坏”原则是在贪心方式的基础上适当增加了节点的选择性，既在一定程度上保证了初始解的质量，也增加了很大的随机性保证其多样性。因此，我们觉得第三种方式是比较好的一种，但同时我们也应该考虑在有的时候既要扩大解的多样性，也要保证解的质量。所以我们构造初始解的方式是三者的随机选择，而且第三种方式选择的概率应该适当大一些，三者具体的概率选择参数将在4.2节中具体说明。

## 禁忌搜索

禁忌搜索是局部搜索的一种扩展，二者之间主要的差别就是前者增加了禁忌准则，简单TS算法的基本思想是：给定一个当前解（初始解）和一种邻域，然后在当前解的邻域中确定若干候选解；若最佳候选解对应的[目标值](http://wiki.mbalib.com/wiki/%E7%9B%AE%E6%A0%87%E5%80%BC)优于当前找到最好解，则忽视其禁忌特性，用其替代当前解和当前最好解，并将相应的对象加入禁忌表；若不存在上述候选解，则选择在候选解中选择非禁忌的最佳状态为新的当前解，而无视它与当前解的优劣，同时将相应的对象加入禁忌表，并修改禁忌表中各对象的禁忌长度；如此重复上述迭代搜索过程，直至满足停止准则。

### 伪代码描述

禁忌搜索是我们整个算法的核心部分，具体的伪代码描述如表3.2所示。其中（第1行）表示在禁忌搜索过程中的全局最优解（3.1节中的是包含扰动过程的算法的全局最优解），（第7行）是一次邻域搜索中的局部最优解。3-6行表示邻域的选择，前文中已经提到了，在我们的算法中存在两种不同的邻域动作，类型Ⅰ和类型Ⅱ，每一次邻域搜索都会从这两种邻域结构中选择一种，选择的方式是随机方式，概率分别为*p*和1-*p*，具体的动作请参考3.3.2节。8-12行表示邻域最优解的选择，对于邻域结构中所有的非禁忌的邻域解，选择对应的评估函数最好的解作为局部最优解，当然，如果禁忌的解达到了解禁条件，也可以用它来更新局部最优解。13-16行表示局部最优解替换全局最优解，当且仅当目标函数值优于全局最优解的时候，如果被替换，则迭代次数计数器重新置0，如果迭代次数*iter*达到最大值*max\_iter*，则一层禁忌搜索执行完毕，返回搜索到的最优解。下面详细介绍各部分的算法思想。

表3.2 主体算法中禁忌搜索部分

|  |
| --- |
| **Algorithm 2：** Tabu Search of the Algorithm 1 |
| **Input：**current solution |
| 1：, |
| 2：**while**  **do** |
| 3： **if then** |
| 4： |
| 5： **else** //选择邻域动作，见3.3.2节 |
| 6： **end if** |
| 7： |
| 8： **for** all Neighborhood solutio*n*  **do** |
| 9： **if**  is better than **then** |
| 10： //邻域评估见3.3.3节 |
| 11： **end if** //禁忌方式和禁忌策略见3.3.4和3.3.5节 |
| 12： **end for** |
| 13： **if** **then** |
| 14： ， |
| 15： **else** |
| 16： **end if** |
| 17：**end while** |
| **Output：** |

### 邻域动作

邻域，指的是解空间中当前解附近的区域，一般动过某个微小的变化可以到达。邻域结构是局部搜索算法的精髓部分，它决定了搜索过程中解更新的方向，影响找到的解的质量，同时也会影响解收敛性，决定搜索耗费的时间。影响邻域解的选择的主要有两个因素：一是邻域动作，二是邻域评估策略。在搜索过程中，当前解在邻域中搜索优度更好的解，算法所能搜索到的地方就是当前解通过邻域动作所能变成的解；而后者决定了当前解的替换和更新方式。



图3.1 对称TSP一般邻域动作

一般地，一个算法的邻域定义为只改变当前解中个成分就能到达的解，那么该算法就称为-opt。现在普遍认为LK(Lin-Kernighan)算法是求解TSP问题最简单最基础的一类算法，它主要是通过几种变化较小的-opt动作来实现的[[25](#_ENREF_25)]，图3.1给出了对称型TSP中的三种*k-opt*动作实例，圆环表示原来的路径，虚线表示动作删除的边，环内的实线均为增加的边，显然对于对称TSP的*-opt*动作会伴随着*k*条边的删除和另外*k*条边的增加。



图3.2 非对称TSP删除两条边动作前后变化

但对于我们研究的非对称TSP，Lin-Kernighan算法的邻域结构就不再适用了，因为边的有向性，当删除边并按照图3.1的方法成环时，会伴随着一段路径的反向，导致解的结构发生了较大的改变。以删除两条边为例，图3.2给出了非对称TSP问题在进行这个动作前后哈密顿环的变化，左边的图表示做动作前的哈密顿环，箭头表示路径中边的方向，其中虚线表示要删除的两条边；右边的环表示删除两条边后为了保证解的有效性而形成的新环，明显可以看到有两条边的方向（最上面两条）发生了变化，而且，如果删除的两条边之间边的数量越多，则反向边也就越多，所以因为图的非对称性，这种动作导致的解的变化就非常大了。因此，对于我们的非对称TSP问题，这种局部动作的幅度偏大，并不适用，而我们的实际测试也正好印证了这一点。

邻域表示当前解的附近区域，所以要求邻域动作要相对小一些。我们研究的问题的合法解是哈密顿环，是边的集合，但我们在实际算法实现的时候，解可以用一个有序序列来表示，其中表示位置处的节点，那么就表示一条路径，要注意的是，由于问题的合法解是哈密顿环，所以起点即是终点，有，这样就可以用一个序列来表示环路。为了将两种情况统一起来，我们定义了两个函数：和，前者表示序列中在之后的节点的位置，后者表示在其之前的节点位置，为了形成环，映射需满足：

(3.1)

(3.2)

由于在序列*S*中，只有相邻点之间才会形成边，所以我们可以通过移动点的位置来完成邻域动作来改变路径上经过的边。一般地，对于线性序列问题，邻域动作有多种，其中点的插入动作是比较有效的，并获得了很好的计算效果[[23](#_ENREF_23)]。由于本文研究的BTSP的合法解是哈密顿环，所以理论上，将某个节点放在序列的首部和末尾是没有任何差别的，所以这里我们规定，一个中的插入动作由两个元素构成，即，它表示将序列中位置第的节点插入到位置为的节点之前，显然，往节点本身和节点位置的插入都是无效的，也就是插入的位置需满足，且，可能的插入位置一共有个。图3.3给出了BTSP问题的点插入邻域动作示例：将位置处的节点插入到位置的节点之前（这与和的大小关系无关，都是插入到前面），其中图(a)是点插入动作的序列变化图，可以看到，动作后形成的新序列中，节点移动到了位置之前，原来两端的两个节点连接在了一起；图(b)是点插入动作的路径变化图，左图是插入前的哈密顿环，将节点插入到节点之前，虚线表示因为这个动作破坏掉的边，很显然，原来节点分别作为起点和终点的两条边都被删除了，与其前面节点的边也被删除了。右图是插入之后的新环，带三角形的箭头表示增加的新边，原位置处两端的节点连接形成一条新的边，同时会在插入的新位置处与两端的节点形成两条新的边。从上面的分析和图示中不难看出，每次点的插入动作虽然对序列中很多点的绝对位置造成很大的影响，但大部分节点的相邻关系没有变化，表现在哈密顿环上也只有三条边被去掉，另外对应增加了三条新边，所以这个动作引起的变化边的总条数是3，属于3-opt的邻域动作。



1. 点插入动作序列变化



1. 点插入动作路径变化

图3.3 点插入邻域动作

因为我们BTSP的求解目标函数是要使得最大边的值最小，我们的插入动作希望能破坏掉当前的最大边，用比其更小的边替换掉当前解的最大边才能对目标函数值做出改进，因此我们选择插入的节点可以选择最大边的两个端点（因为边是有向的，选择起点可能更好）；亦或者选择某个点插入到最大边的两个点之间。后一个动作基本的过程仍然是序列上点的插入，只是选择插入的点不同，这里就不画图演示了。这两种插入都可以选择，邻域结构也因此可以扩展到，虽然增加了一倍的解空间，邻域搜索的时间也相应增加了，但由于邻域结构的增大，更容易覆盖到最优解，经过我们测试发现，计算时间反而减少了，而且更容易得到最优解。



图3.4 块插入示意图

在一般线性序列的问题中，点的插入动作也可以扩展到“块”的插入，在本论文研究的TSP问题里，这个“块”指的就是几个连续的点，也就是哈密顿环上连续相邻的几个节点。块中的几个节点构成的几条边就构成了哈密顿环的子路径，块之间节点的相邻关系越牢固，就表示节点之间的边关系越牢固，说明这段子路径应该作为BTSP解的一个整体，不要轻易破坏掉。图3.4给出了块移动的示意图，假设当前解的最大边为序列中位置与位置（作0处理）处的点形成的有向边，图(a)的类型Ⅰ表示将起点左边的块移动到节点之前，图(b)中的类型Ⅱ表示将节点附近的块移动到最大边的两个端点之间。块的插入动作和点的插入动作很类似，唯一不同的是块是将几个连续的点作为整体来移动，所以一个块的插入动作需要三个元素才能确定，即，这里表示插入的块的位置，既可以用起始位置来表示（如类型Ⅱ），也可以用块的终止位置来表示（如类型Ⅰ）；表示块的大小，也就是这个块中节点的数量；和点插入动作定义一样，表示的是块插入的位置，同样，这里也表示把块插入到位置为的节点之前。要注意的是，当前解的块插入动作形成的邻域中，块的大小是不固定的，仅仅只是规定一个块大小的最大值，我们设为，任何大小不超过的块移动形成的解都是邻域的一部分。和点的插入一样，我们希望邻域解对解的质量是不断更新的，所以插入动作必须要破坏掉当前的最大边。对于类型Ⅰ的块移动，是将最大边左边的块插入到序列的其它位置，所以块的终点位置是固定的，只是块长和插入位置不一样，在这种动作中，一个长为的块，可能的插入位置一共有-1-个；对于类型Ⅱ的块移动，是将序列中的某个块插入到最大边的两个节点之间，以此来破坏掉当前解最大边，所以块插入的位置是固定的，块长和起始位置不一样，一个长为的块，可能的起始位置同样一共有-2-个。因此，块插入的邻域结构的大小如公式(3.3)所示。

(3.3)

从上面的分析不难看出，点插入其实就是特殊的块插入，所以我们选择用块的插入来作为我们禁忌搜索的邻域动作。对于任意大小的块的插入，其中一定会伴随着三条边的删除和三条边的增加，所以这种块插入的邻域动作是3-opt的。

### 邻域评估

根据前文的论述，最适合作为我们非对称BTSP的邻域动作是块的插入动作（对几组算例分别应用上述的邻域结构进行简单的局部搜索，发现块的插入效果是最好的），邻域结构大小约为，其中*n*代表节点的个数，代表块的最大节点数量。如图3.4所示，对于我们研究的BTSP问题，块的插入动作有两种类型，这二者都是以破坏当前解的最大边为目的而进行的局部动作，类型Ⅰ将最大边的起点左边的块移动到序列的其它位置，类型Ⅱ则是将序列中其它位置处的块插入到最大边处的两个节点之间。这两种类型的动作都是3-opt的，即每个动作删除了三条边并引进了三条新的边。每个邻域动作，无论是类型Ⅰ还是类型Ⅱ，去除的三条边中都有一条是当前解的最大边，所以邻域评估的时候我们没有去考虑删除的边的值，仅仅考虑的是增加的三条新边的距离代价。要注意的是，这两种类型的邻域动作，我们评估的方式略微有些不同。

图3.5所示的是按照类型Ⅰ的方式进行的邻域动作后边的变化，上面的图表示当前解，下面的图代表做该邻域动作后的解，为了表示的方便，图中的“”符号和“”符号分别对应上节中的在上图中*next*函数和*up*函数。在图的上部分，编号为和的节点构成了最大边，该图所示的邻域动作是将最大边左边的大小为2的块（也就是编号为和构成的节点集合）移动到编号为的结点之前，这样原来编号为和的节点构成的边和编号为*i*-2和*i*-1的节点构成的边以及破坏掉的当前最大边就是三条被删除的边。生成的新解就是图下面部分显示的内容，其中节点和结点构成新边，节点和节点构成新边，节点*i*-2和节点构成新边，这三条新边也是算法评估邻域动作的依据。在考虑评估策略的时候，我们最先想到的是去选择增加的三条边最大边最小的邻域动作，但问题是，在类型Ⅰ的邻域动作下，当我们评估的块大小是一样的时候（假设为），那么新边就一定会是节点与节点构成的边，如果这条边较大，以至于就等于，那么对于所有块为的类型Ⅰ的邻域动作，其评估函数是一样的，这就导致算法对邻域的优化性不强，计算结果不尽人意。所以，我们对类型Ⅰ的评估函数为：

(3.4)

其中表示一个相应的邻域动作。在搜索过程中，首先选择个每个块长所有的邻域动作中评估函数最小的，对于评估函数相同的情况等概率选择，然后在这个邻域动作中也去选择最小的，但当这个评估函数中有相同的时候，则去选择边最小对应的邻域动作。



图3.5 类型Ⅰ邻域动作边结构变化示意图

图3.5所示的是按照类型Ⅱ的方式进行的邻域动作后边的变化图，这里最大边依然是编号为和的节点构成的边，在这种类型的邻域动作中，是将其它位置处的块插入到两个节点之间来破坏最大边，我们可以用其中来表示。如图的上部分所示，我们将大小为2的块，也就是编号为和两个节点集合，移动到节点之前，这样原来节点与构成的边，节点与构成边以及当前解最大边就是该动作删除掉的三条边。图3.5的下半部分表示做完动作后边的变化，可以看到，编号为和的节点构成新边，节点与节点构成的新边，节点与节点构成新边。虽然对于这三条新边，也会出现不同的邻域动作出现某条相同新边的情况，比如当节点的起点（也就是）相同时，对于不用的，虽然不同，但是增加的边是相同的。同样，新边和新边也会出现相类似的情况。但在我们的算法中，为了增强算法的收敛速度，我们设置的远小于，可能出现相同最大边的邻域动作数量很少，而理论上，考虑三条边的最大值是更符合我们目标函数值的优化方向，所以，针对类型Ⅱ的邻域动作，设计的评估函数为:

(3.5)

在所有的动作中，如果采取该种类型的邻域搜索，则在当前解的邻域中选择评估函数最小的邻域解来作为更新后的解，替代原有的解决方案。值得一提的是，我们在结果测试中发现，类型Ⅱ的评估函数在如此设置时，对于一部分算例起不到效果，而设置为类型Ⅰ样式的时候，对于不能求解的算例起到了很好的效果，具体的我们将在4.2节说明。



图3.6 类型Ⅱ邻域动作边结构变化示意图

### 禁忌方式

在我们的禁忌算法设计中，两种类型的邻域动作都是会以破坏当前解最大边为目的，而如果在后面搜索的过程中，可能由于当前解已经达到了局部最优，解有可能会朝着变坏的方向发展，之前删除的边又被重现添加了进来，然后开始重复做动以前的动作。所以，我们设计的禁忌算法在每次邻域动作结束之后，将删除的这个最大边加入禁忌表outTabuList，这样可以不让最大边重复进入到解的结构中，而对于另外两条删除的边，因为它们是由于“被迫”产生的，而且评估的时候我们并没有考虑到这两条边的代价，所以可以不考虑它们的影响。另外，对于邻域动作做完后，会有三条增加的边，我们将增加的三条新边都加入另一个禁忌表inTabuList，防止刚刚进来的优秀的边马上就被删除掉，同时也可以避免小范围内的重复局部动作，防止重复搜索浪费计算时间。

在进行每一次邻域搜索的时候，一旦邻域动作增加的三条边中有一条是outTabuList中的禁忌边，那么解的目标函数值就会回到以前搜索的值，那么接下来的邻域动作也会围绕着这条禁忌边来进行，这显然在有很大的可能会造成重复搜索，所以我们约束如果一个邻域动作增加的三条边的任意一条是outTabuList禁忌表中的对象，则该动作是被禁忌的。而对于邻域动作删除的边，如果按照增加边相同的判禁方式，那么很容易让邻域动作受到限制，在解禁条件没有达到的时候，失去了算法对某些良好动作的选择性，算法执行时的解空间搜索范围也相对小了一些，影响搜索性能，所以对于删除的边的判禁原则应该稍微松弛一些，只有当三条边都是禁忌表inTabuList中的边，也就是这些边在一段时间之前才刚刚被添加到解中，该动作就是被禁忌的。因为outTabuList禁忌的是出去的边，也就是当前解之外的边，而inTabuList禁忌的是解上的边，对于个结点的带权完全图，边一共为条，在BTSP中解上的边一共有条，解外的边一共有条，所以解外边的数量是远大于解上边的数量的，因此不难想到，outTabuList中边的禁忌长度应该要比inTabuList中的禁忌长度大得多，在本论文中，设置的两个禁忌表长度分别为：

(3.6)

如公式(3.6)所示，无论是inTabuList还是outTabuList，其禁忌长度都由两部分构成，一部分是一个固定值，另一部分是一个不超过的随机数，考虑到两个禁忌表的松弛程度应该有所差异，对于inTabuList和outTabuList，这两个值都不一定相同，具体的参数请参考4.2节的具体内容。

### 解禁策略

禁忌表的存在避免了重复性的搜索动作，节省了搜索时间，但由于问题的复杂性太大，搜索过程中算法的具体行为我们是无法预知的，而人们在利用禁忌搜索算法求解其它NP-hard问题的时候也发现：在禁忌搜索过程中，由于禁忌表的存在，很可能会漏掉一些高质量的解，失去了得到更优解的机会。一般地，解决这一问题的方式是通过藐视准则来实现的，当某个被禁忌的邻域解达到了藐视准则的要求，则可以忽略禁忌表的存在，选择其作为更新的解，这种方式也可以称为解禁策略。在实际的问题中，藐视准则一般设定为：如果禁忌邻域解的目标函数优于当前全局最优解，则动作被解禁，而在本论文研究的BTSP问题中，每评估一个邻域动作我们只考虑了增加的三条边的局部优劣性，不一定能反映出整个解的目标函数的变化，而如果每个动作都去评估整个解的目标函数值，这个时间代价显然是得不偿失的，所以我们的解禁策略应该还是围绕着增加的三条边，而且为了保证禁忌的有效性，解禁的条件应该适当严格一些。在实际算法中，我们规定，如果在所有的邻域动作中，如果当前最优禁忌邻域动作增加的三条边中最大的值比当前最优非禁忌邻域动作增加的三条边中的最小值还要小，则该禁忌邻域动作被解禁，并用其来更新当前解。

## 扰动机制

一般地，当局部搜索算法或者禁忌搜索执行一段时间后，很容易在当前解的邻域中找不到一个更优的解，如果不做其他处理，邻域搜索就陷入了局部最优解，可能这个局部最优解距离最优解还比较远，而依靠原来的搜索过程是无法继续进行下去找到这个最优解的。这时，可以依靠扰动机制来解决这个问题，将某个解通过某种变化到一个新的解或者格局，然后从这个新的解继续应用搜索策略，则有可能到达最优解的位置。在我们提出的迭代禁忌搜索算法中，我们设计了相应的扰动机制，在实验过程中我们发现，不依靠这种扰动机制得到的解已经非常接近问题最优解了，所以我们并没有在这上面花费很大的功夫，简单设计的扰动机制已经有了比较好的效果。

这里首先引入一个概念：年龄矩阵，所谓“年龄”，就是BTSP完全图中，在整个搜索过程中，某条边出现在局部最优解中的次数，我们可以用一个矩阵来表示，即表示边在局部最优解中出现的次数，这个值越大，表示出现的越频繁，则我们的算法就应该尽量保留这条边。在我们设计的扰动动作中，任意选取年龄值不大的边所包含的起点作为移动点，将这个点从插入到某个位置，要求满足移动产生的新边的代价最小，而且去掉的边不能是经常出现的边，也就是年龄值较大的边。这个动作可以多次进行，次数为，具体的设置见表4.2。那么，如何判断一个边是经常出现的边呢？我们是采用函数来判断的，该函数满足映射：

(3.7)

其中，表示所有边中的最大年龄值。当值为1的时候表示该边在局部最优解中是经常出现的，为0时表示不是经常出现的。

# 实验与结果分析

## 测试方案设计

### 测试算例

实验中我们选取了86个比较有代表性的不同类别的非对称BTSP算例，这些算例是前人总结的标准测试算例，已有很多学者利用这些算例展开了对于BTSP问题的研究。这些算例最好的计算结果来源于LaRusic的最新研究成果[[9](#_ENREF_9)]，文中给出了这86个算例的所有下界，而且仅仅只有一个算例没有最优解。按照各个算例的特征，我们将他们划分为以下几类（均为非对称实例）。

1. 不同领域实际应用的仿真算例。这些算例一共有42个，是由Cirasella、Johnson、 McGeoch和Zhang提出来的[[26](#_ENREF_26)]，它们可以分成5组，每一组包括5个100个节点的算例和1个316个节点的算例：

* coin：付费电话硬币收集实例
* crane：随机欧几里得塔式起重机实例
* disk：磁盘驱动器实例
* rtilt：按总和定额的倾斜钻孔机实例
* shop：无等待流车间实例
* stilt：按最大值定额的倾斜钻孔机实例
* super：近似最短常见超弦实例

1. 由Balas生成的5个杜邦化学工厂调度应用模拟算例，这5个算例分别具有84、108、120、160和200个节点的规模[[9](#_ENREF_9)]。
2. 在Reinelt维护的TSP标准库中有27个算例[[27](#_ENREF_27)]。
3. 两个真实世界算例（ftv180和uk66），以及距离值分布在[1，1000]之间，节点数为500个和1000个的随机算例各有5个，这些都由Fischetti创建[[28](#_ENREF_28)]。

这些算例是在多个算例中选取的比较有代表性的几组，我们会用这些算例来测试我们设计的启发式算法的执行效果，并与其它已知的算法进行比较和分析，从计算时间和计算质量两个方面来分析算法的优劣性。

### 实验环境

本毕业设计相关算法的测试是在计算机学院智慧计算与优化实验室内的设备上完成的，实验详细的环境配置如表4.1所示。

表4.1 实验环境配置表

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 配置或参数 |
| 硬件环境 | 处理器 | Intel(R) Core(TM) i3-4130 CPU@ 3.40GHz |
| 内存 | 4GB |
| 硬盘 | 500GB |
| 软件环境 | 操作系统 | Microsoft Windows 8.1 专业版 (64位) |
| 编译器 | mingw32-gcc |
| IDE | CodeBlocks |

## 参数设定

### 所有参数设置

在我们设计的启发式算法中，算法的执行性能受各个参数的影响很大，如邻域解的选择概率、初始解的选择概率以及禁忌长度的大小，都会对算法的性能产生重要的影响。表4.2给出了我们算法中各个类型的参数的实际值，每个名称的参数都有详细的意义说明，另外，在右边“值”一列中，表示BTSP问题中节点的数量。表4.2中所有内容，除第一、二行的参数是为了算法能得到更好的解，保证算法最少的执行时间外，其它参数都是经过了大量的算例测试得到的相应较好或者较平衡的值。三个值表示构造初始解方式选择概率，其中“不坏”原则方式选择概率相对高一些；表示最大块长，它决定了邻域结构大小；和分别表示禁忌长度的固定值部分和随机数部分，前文分析也提到了，outTabuList中的禁忌长度要远大于inTabuList中的禁忌长度。由于参数测试的过程较为繁琐，这里就不一一列举了，仅选取对算法性能影响最大的邻域动作最大块长来作为示例。

表4.2 算法中各参数设置

|  |  |  |
| --- | --- | --- |
| **名称** | **说明** | **值** |
|  | 算法终止的最大迭代次数，也是最大扰动次数 | 1000 |
|  | 禁忌搜索中最大邻域搜索次数 | 1000 |
|  | 完全随机方式构造初始解概率 | 0.25 |
|  | 完全贪心方式构造初始解概率 | 0.25 |
|  | “不坏”原则构造初始解概率 | 0.25 |
| *p* | 邻域动作选择类型Ⅰ的概率 | 0.5 |
|  | 邻域动作的最大块长 |  |
|  | inTabuList表中禁忌长度固定值部分 |  |
|  | outTabuList表中禁忌长度固定值部分 |  |
|  | inTabuList表中禁忌长度随机数部分 | 10 |
|  | outTabuList表中禁忌长度随机数部分 | 20 |
|  | 扰动动作次数 | 10 |

### 块长测试

图4.1给出了算法策略和其它参数相同的情况下，最大块长对算法性能影响的图示，其中，横轴表示最大块长，我们一共测试了1~5、n/10、n/5和n/4共八种不同的值，算例选取的是86个算例中比较容易计算的76个算例（另外10个算例依靠现有策略，无论块长多少，花费很长时间也无法算到最优解，这里就排除了）；左边的纵轴显示的是能计算得到最优解的算例个数；右边显示的是这86个算例的总体计算时间，单位是s。

从图4.1可以看出，随着块长的增大，依靠本论文中的多邻域迭代禁忌搜索算法计算得到最优解个数越多。当块长为1的时候，计算得到的最优解数量明显少于其它块长设定，这说明我们将点插入拓展到块插入的思想是非常有效的。我们还可以看到，当块长最大值不小于n/10的时候，均能稳定得到74个最优解，另外的2个算例虽然能在短时间得到比较好的解，但是直到迭代次数达到最大值，依然无法得到最优解。虽然对于这两个算例，程序消耗完所有的迭代次数才停止，但统计的时间是得到当前最好解的时间，所以并不是很大。对于总的计算时间，理论上说，随着块长的增加，邻域的结构在增大，邻域评估的时间应该增加，但事实上，在块长比较小(1~5)的时候，由于部分算例无法得到最优解，需要耗费的时间很长，这就导致总的计算时间较大，这时候块长的增加一方面由于得到最优解的个数增加，减少了搜索时间，另一方面邻域结构增大，增加了邻域搜索时间，而后者的变化赶不上前者的变化，所以块长较小的时候，总的计算时间在减小；当块长较大的时候（n/10，n/5，n/4），他们计算得到的最优解个数都一样，在很少的迭代次数内，邻域就覆盖到了最优解，但块长过大，邻域结构过于庞大，在搜索过程中花费了很多不必要的时间来评估很大一部分邻域，计算时间开始增加。块长的选择既要考虑到计算结果的质量，同时也要考虑到计算时间，所以根据实验图的分析，我们选择n/10来作为算法中块插入邻域动作中最大块长的参数设定。

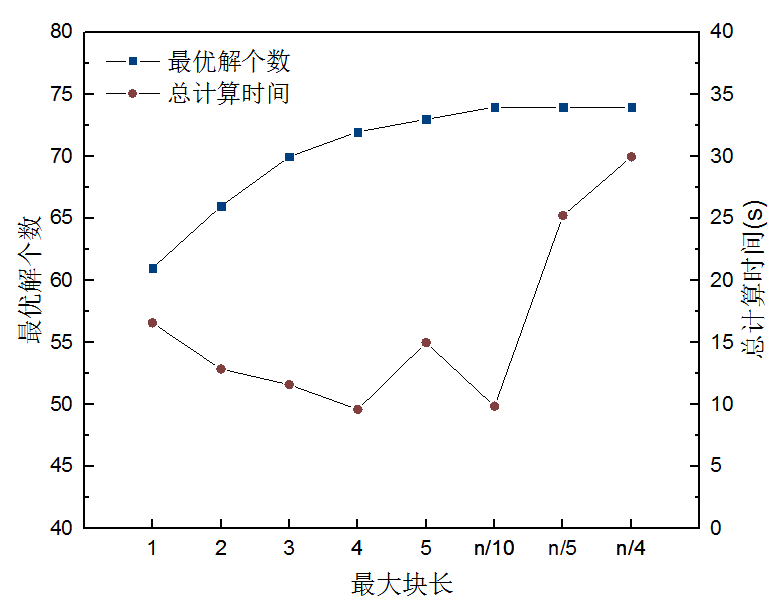


图4.1 块长对算法性能的影响

## 算法性能分析

根据表4.2.1中各参数的设定，将迭代禁忌搜索算法应用到这86个不同类型的benchmark上，其中部分算例已经有精确算法计算到了最优解，还有很大一部分算例虽然仅仅只有算法得到了下界，但是前人的工作中有一些学者曾经计算得到了目标函数值和下界值一样的解，所以可以判定这个解一定是最优解。目前最好的计算结果是2014年LaRusic得到的，这里将论文中算法的求解结果和LaRusic的算法求解结果作对比，统计得到的计算结果如表4.3所示。

表4.3 ITS与JLR的计算性能比较

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem | Size | Opt. sol. | JLR | | | |  | ITS | | | |
| Best gap | Avg. gap | Worst gap | Time(s) |  | Best gap | Avg. gap | Worst gap | Time(s) |
| coin100.0 | 100 | 253 | 0.00 | 0.00 | 0.00 | 0.97 |  | 0.00 | 0.00 | 0.00 | 0.04 |
| coin100.1 | 100 | 219 | 0.00 | 0.00 | 0.00 | 0.42 |  | 0.00 | 0.00 | 0.00 | 0.03 |
| coin100.2 | 100 | 207 | 0.00 | 0.48 | 1.45 | 16.95 |  | 0.00 | 0.00 | 0.00 | 0.28 |
| coin100.3 | 100 | 232 | 0.00 | 0.00 | 0.00 | 0.85 |  | 0.00 | 0.00 | 0.00 | 0.05 |
| coin100.4 | 100 | 214 | 0.00 | 0.00 | 0.00 | 0.11 |  | 0.00 | 0.00 | 0.00 | 0.10 |
| coin316.10 | 316 | 227 | 0.00 | 0.00 | 0.00 | 6.51 |  | 0.00 | 0.00 | 0.00 | 2.53 |
| crane100.0 | 100 | 173390 | 0.00 | 0.00 | 0.00 | 0.12 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| crane100.1 | 100 | 152923 | 17.80 | 17.80 | 17.80 | 25.34 |  | 17.80 | 17.80 | 17.80 | 0.01 |
| crane100.2 | 100 | 214843 | 0.00 | 0.00 | 0.00 | 0.11 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| crane100.3 | 100 | 145622 | 0.00 | 0.00 | 0.00 | 0.65 |  | 0.00 | 0.00 | 0.00 | 0.09 |
| crane100.4 | 100 | 171484 | 0.00 | 0.00 | 0.00 | 0.20 |  | 0.00 | 0.00 | 0.00 | 0.05 |
| crane316.10 | 316 | 120333 | 0.00 | 0.00 | 0.00 | 128.23 |  | 0.00 | 0.00 | 0.00 | 0.27 |
| disk100.0 | 100 | 508034 | 0.00 | 0.00 | 0.00 | 0.12 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| disk100.1 | 100 | 473495 | 0.00 | 0.00 | 0.00 | 0.11 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| disk100.2 | 100 | 382677 | 1.08 | 1.08 | 1.08 | 10.78 |  | 1.08 | 1.08 | 1.08 | 0.42 |
| disk100.3 | 100 | 453657 | 0.00 | 0.00 | 0.00 | 0.13 |  | 0.00 | 0.00 | 0.00 | 0.03 |
| disk100.4 | 100 | 415696 | 0.00 | 0.00 | 0.00 | 0.20 |  | 0.00 | 0.00 | 0.00 | 0.11 |
| disk316.10 | 316 | 309801 | 0.00 | 0.00 | 0.00 | 0.85 |  | 0.00 | 0.00 | 0.00 | 0.36 |
| rtilt100.0 | 100 | 260342 | 9.43 | 12.88 | 18.16 | 47.55 |  | 0.00 | 0.00 | 0.00 | 0.07 |
| rtilt100.1 | 100 | 291040 | 0.00 | 9.65 | 17.33 | 41.22 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| rtilt100.2 | 100 | 227248 | 22.29 | 28.76 | 33.63 | 48.99 |  | 0.00 | 0.00 | 0.00 | 1.61 |
| rtilt100.3 | 100 | 236920 | 22.79 | 29.48 | 44.78 | 54.63 |  | 0.00 | 0.00 | 0.00 | 0.03 |
| rtilt100.4 | 100 | 294367 | 7.99 | 9.19 | 11.96 | 58.21 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| rtilt316.10 | 316 | 152510 | 96.33 | 101.68 | 110.68 | 490.10 |  | 0.00 | 0.00 | 0.00 | 0.65 |
| shop100.0 | 100 | 2232 | 0.00 | 0.06 | 0.58 | 5.57 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| shop100.1 | 100 | 2608 | 0.00 | 0.00 | 0.00 | 0.11 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| shop100.3 | 100 | 2526 | 0.00 | 0.00 | 0.00 | 0.39 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| shop100.4 | 100 | 2792 | 0.00 | 0.00 | 0.00 | 0.10 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| shop316.10 | 316 | 2311 | 0.00 | 1.11 | 2.34 | 92.42 |  | 0.00 | 0.00 | 0.00 | 0.15 |
| stilt100.0 | 100 | 382208 | 7.43 | 11.20 | 20.96 | 45.81 |  | 0.00 | 0.00 | 0.00 | 0.05 |

表4.3(续) ITS与JLR的计算性能比较

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem | Size | Opt. sol. | JLR | | | |  | ITS | | | |
|  |  |  | Best gap | Avg. gap | Worst gap | Time(s) |  | Best gap | Avg. gap | Worst gap | Time(s) |
| stilt100.1 | 100 | 491416 | 0.00 | 0.00 | 0.00 | 2.57 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| stilt100.2 | 100 | 377720 | 5.56 | 11.80 | 19.12 | 48.93 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| stilt100.3 | 100 | 401976 | 2.24 | 7.04 | 19.50 | 42.77 |  | 0.00 | 0.00 | 0.00 | 0.09 |
| stilt100.4 | 100 | 347440 | 24.40 | 29.43 | 38.10 | 52.61 |  | 0.00 | 0.00 | 0.00 | 0.45 |
| stilt316.10 | 316 | 226504 | 83.18 | 91.39 | 99.89 | 352.84 |  | 0.00 | 0.00 | 0.00 | 0.73 |
| super100.0 | 100 | 10 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| super100.1 | 100 | 11 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| super100.2 | 100 | 10 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| super100.3 | 100 | 10 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| super100.4 | 100 | 10 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| super316.10 | 316 | 9 | 0.00 | 0.00 | 0.00 | 0.48 |  | 0.00 | 0.00 | 0.00 | 0.40 |
| ftv180 | 181 | 37 | 0.00 | 0.00 | 0.00 | 25.08 |  | 0.00 | 0.00 | 0.00 | 0.26 |
| uk66 | 66 | 170 | 0.00 | 0.00 | 0.00 | 0.05 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| balas84 | 84 | 18 | 0.00 | 0.00 | 0.00 | 0.07 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| balas108 | 108 | 13 | 0.00 | 0.00 | 0.00 | 0.09 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| balas120 | 120 | 22 | 0.00 | 0.00 | 0.00 | 0.11 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| balas160 | 160 | 13 | 0.00 | 0.00 | 0.00 | 0.17 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| balas200 | 200 | 13 | 0.00 | 0.00 | 0.00 | 0.39 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| ran500.0 | 500 | 24 | 0.00 | 0.00 | 0.00 | 1.23 |  | 41.67 | 41.67 | 41.67 | 129.19 |
| ran500.1 | 500 | 22 | 0.00 | 0.00 | 0.00 | 3.73 |  | 54.55 | 54.55 | 54.55 | 150.29 |
| ran500.2 | 500 | 23 | 0.00 | 0.00 | 0.00 | 2.73 |  | 47.83 | 47.83 | 47.83 | 227.05 |
| ran500.3 | 500 | 23 | 0.00 | 0.00 | 0.00 | 2.43 |  | 47.83 | 47.83 | 47.83 | 146.30 |
| ran500.4 | 500 | 28 | 0.00 | 0.00 | 0.00 | 1.89 |  | 17.86 | 21.43 | 25.00 | 276.30 |
| ran1000.0 | 1000 | 18 | 0.00 | 0.00 | 0.00 | 4.61 |  | 44.44 | 45.83 | 50.00 | 1466.68 |
| ran1000.1 | 1000 | 17 | 0.00 | 0.00 | 0.00 | 7.52 |  | 52.94 | 52.94 | 52.94 | 1456.41 |
| ran1000.2 | 1000 | 17 | 0.00 | 0.00 | 0.00 | 9.46 |  | 52.94 | 54.41 | 58.82 | 2366.25 |
| ran1000.3 | 1000 | 19 | 0.00 | 0.00 | 0.00 | 8.25 |  | 36.84 | 36.84 | 36.84 | 2900.80 |
| ran1000.4 | 1000 | 19 | 0.00 | 0.00 | 0.00 | 7.61 |  | 36.84 | 38.16 | 42.11 | 1690.47 |
| br17 | 17 | 8 | 0.00 | 0.00 | 0.00 | 0.01 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ft53 | 53 | 977 | 0.00 | 0.00 | 0.00 | 0.04 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ft70 | 70 | 1398 | 0.00 | 0.00 | 0.00 | 0.05 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv33 | 34 | 113 | 0.00 | 0.00 | 0.00 | 0.02 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv35 | 36 | 113 | 0.00 | 0.00 | 0.00 | 0.02 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv38 | 39 | 113 | 0.00 | 0.00 | 0.00 | 0.03 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv44 | 45 | 113 | 0.00 | 0.00 | 0.00 | 0.03 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv47 | 48 | 104 | 0.00 | 0.00 | 0.00 | 0.03 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv55 | 56 | 64 | 0.00 | 0.63 | 3.13 | 2.35 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| ftv64 | 65 | 104 | 0.00 | 0.00 | 0.00 | 0.05 |  | 0.00 | 0.00 | 0.00 | 0.00 |

表4.3(续) ITS与JLR的计算性能比较

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Problem | Size | Opt. sol. | JLR | | | |  | ITS | | | |
|  |  |  | Best gap | Avg. gap | Worst gap | Time(s) |  | Best gap | Avg. gap | Worst gap | Time(s) |
| ftv70 | 71 | 104 | 0.00 | 0.00 | 0.00 | 0.06 |  | 0.00 | 0.00 | 0.00 | 0.00 |
| ftv90 | 91 | 48 | 0.00 | 0.00 | 0.00 | 1.21 |  | 0.00 | 0.00 | 0.00 | 0.02 |
| ftv100 | 101 | 53 | 0.00 | 0.00 | 0.00 | 0.66 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| ftv110 | 111 | 39 | 0.00 | 7.95 | 10.26 | 19.03 |  | 0.00 | 0.00 | 0.00 | 0.03 |
| ftv120 | 121 | 39 | 0.00 | 5.38 | 10.26 | 16.85 |  | 0.00 | 0.00 | 0.00 | 0.07 |
| ftv130 | 131 | 39 | 0.00 | 27.18 | 135.90 | 21.42 |  | 0.00 | 0.00 | 0.00 | 0.16 |
| ftv140 | 141 | 41 | 0.00 | 86.10 | 129.27 | 53.34 |  | 0.00 | 0.00 | 0.00 | 0.11 |
| ftv150 | 151 | 37 | 0.00 | 57.30 | 140.54 | 51.01 |  | 0.00 | 0.00 | 0.00 | 0.15 |
| ftv160 | 161 | 37 | 0.00 | 0.00 | 0.00 | 26.14 |  | 0.00 | 0.00 | 0.00 | 0.16 |
| ftv170 | 171 | 37 | 0.00 | 0.00 | 0.00 | 24.24 |  | 0.00 | 0.00 | 0.00 | 0.28 |
| kro124p | 100 | 607 | 0.00 | 0.00 | 0.00 | 0.24 |  | 0.00 | 0.00 | 0.00 | 0.04 |
| kro124p | 100 | 607 | 0.00 | 0.00 | 0.00 | 0.24 |  | 0.00 | 0.00 | 0.00 | 0.04 |
| p43 | 43 | 5008 | 0.00 | 0.00 | 0.00 | 0.02 |  | 0.00 | 0.00 | 0.00 | 0.01 |
| rbg323 | 323 | 12 | 16.67 | 20.00 | 50.00 | 94.78 |  | 0.00 | 0.00 | 0.00 | 1.51 |
| rbg358 | 358 | 14 | 0.00 | 4.29 | 7.14 | 75.06 |  | 0.00 | 0.00 | 0.00 | 0.18 |
| rbg403 | 403 | 20 | 5.00 | 5.50 | 10.00 | 110.60 |  | 0.00 | 0.00 | 0.00 | 0.48 |
| rbg443 | 443 | 20 | 15.00 | 15.00 | 15.00 | 138.43 |  | 0.00 | 0.00 | 0.00 | 0.60 |
| ry48p | 48 | 577 | 0.00 | 0.00 | 0.00 | 5.17 |  | 0.00 | 0.00 | 0.00 | 0.00 |

表4.3中第一列<Problem>表示算例的名称及编号；<Size>表示BTSP的问题规模，也就是节点的总数量；<Opt>表示该问题的最优解；<JLR>和<ITS>分别表示两种算法，前者是由John LaRusic等人实现并以之命名的启发式算法[[9](#_ENREF_9)]，后者是本论文中提出的迭代禁忌搜索算法（Iterated Tabu Search），由于John LaRusic发表的论文中的计算结果都是针对每个算例计算10次得到的，为了比较结果的无差异性，在测试本论文提出的算法的时候，我们同样对每个算例计算10次。在这两个算法的子栏中分别有四项内容，<Avg. gap>表示10次计算的计算结果平均值，<Best gap>和<Worst gap>分别表示最好和最坏的计算结果。其中，的计算公式为，为设计的启发式算法得到的目标函数值，表示该算例的最优解，所以即表示算法计算得到的目标函数值与算例最优解差值百分比(%)。另外，两种算法各自的执行时间均统计的是10次运算的平均值，即<Time>这一栏数据，单位是秒。从表中数据可以看出，不论是ITS算法还是JLR算法，都存在某些算法没有得到最优解的情况，而且两种算法对应的算法类型都比较集中，比如JLR大多集中于rtilit、stilit以及ftv上，ITS没有解决的算法集中于ran系列的算例上。从具体的数据来说，JLR没有得到最优解的算例的值都较大，有些甚至在100以上，而我们的ITS计算结果中，最坏的解的gap值也仅仅在50左右。值得一提的是，算例stilt316.0当时并没有最优解，仅仅只有一个下界，而我们的算法在执行过程中恰好计算得到了目标函数值等于下界值的解决方案，所以这个值必然是最优解。

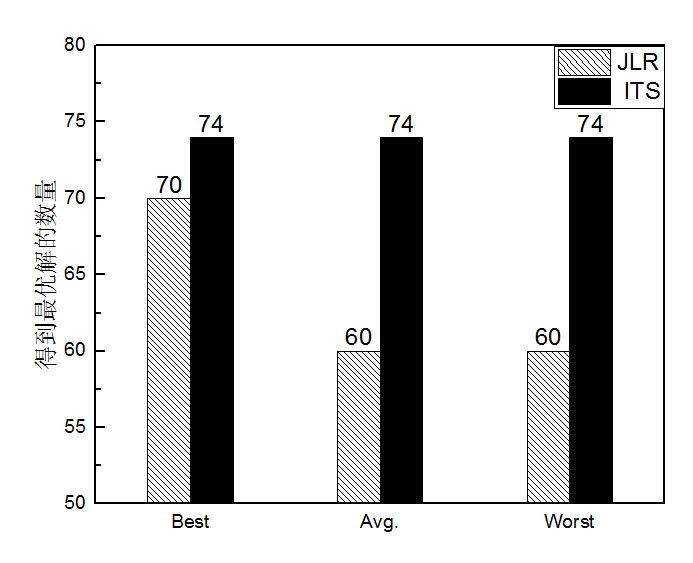


图4.2 ITS与JLR计算结果对比

为了更直观地表现出本论文中的算法ITS在得到的解的质量上与John LaRusic提出的算法JLR的优劣性，我们统计了在这86个算例之中，依靠着这两种算法在最好、最坏以及平均情况下能得到的最优解数量，图4.2所示是两个算法计算结果的柱状图比较。从图中可以看出，JLR算法最好情况下能得到70个问题的最优解，平均情况和最坏情况下均能得到60个算例的最优解；而本论文提出的多邻域迭代禁忌搜索算法的表现性能明显要强一些，无论经过几次计算，都能得到74个算例的最优解，这说明我们算法不仅计算性能较强，能很快得到问题的最优解，而且计算的稳定性也很强，不管生成的初始解如何，不论算法执行随机数产生的规律如何，都能通过迭代禁忌搜索算法得到很好的计算结果。和JLR相比，ITS在最坏情况下都能比前者最好结果多出四个最优解，这一点很好地体现了我们的迭代禁忌搜索算法在求解BTSP问题上卓越的计算性能。

在算法的执行时间上，ITS相较于JLR算法各有优势。根据John LaRusic发表的论文中的描述，他设计的算法的可执行程序是由GNU C编译生成的，测试机器使用的是个人PC，CPU是3.40GHz奔腾4处理器，内存容量2GB，操作系统为Windows XP SP2。因为他所使用的CPU的频率参数和我们本论文测试的CPU频率参数是一样的，所以算法执行时间不需要进行转化。从表4.3可以看出，我们的ITS算法执行时间，除了ran系列的算例，由于我们一直不能得到问题的最优解，所以花费的计算时间特别地长，达到了数千秒，而另外76个算例的计算时间绝大部分都不超过1s；对于JLR的算法执行时间，各个算法的执行时间参差不齐，而且很多算例的执行时间都比较长，在40s以上，最多是490s，但JLR求解我们提出的ITS算法所不能解决的问题却很有优势，花费个位数的时间就能得到ran系列问题的最优解。

表4.4 修改评估函数后ran系列算例计算结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problem | Size | Opt | Best | Avg. | Worst | Time(s) |
| ran500.0 | 500 | 24 | 24 | 24 | 24 | 24.91 |
| ran500.1 | 500 | 22 | 22 | 22 | 22 | 58.63 |
| ran500.2 | 500 | 23 | 23 | 23 | 23 | 18.41 |
| ran500.3 | 500 | 23 | 23 | 23 | 23 | 53.39 |
| ran500.4 | 500 | 28 | 28 | 28 | 28 | 15.42 |
| ran1000.0 | 1000 | 18 | 18 | 18 | 18 | 281.43 |
| ran1000.1 | 1000 | 17 | 17 | 17 | 17 | 271.58 |
| ran1000.2 | 1000 | 17 | 17 | 17 | 17 | 317.60 |
| ran1000.3 | 1000 | 19 | 19 | 19 | 19 | 244.35 |
| ran1000.4 | 1000 | 19 | 19 | 19 | 19 | 213.99 |

在4.1节中我们研究块长对计算结果质量的影响，并发现最大块长设置的越大，算法得到的解质量越高，但时间往往会很长。为了解决ran系列问题，我们将最大块长参数提高到了5/n，并修改了类型Ⅱ邻域动作的评估函数为：

(4.1)

这里各个变量的具体意义请参考3.3.3节中的具体内容。仅仅修改上述的内容，在其它策略和参数不变的情况下，我们重新计算了这10个困难算例，同样每个算例计算10次，得到的ran系列10个算例的计算结果如表4.4所示。可以看出，我们修改了类型Ⅱ邻域动作的评估函数，并提高了块长之后，ITS算法可以得到这10个ran系列算例的最优解，而且稳定性也能保证，最坏情况下也能得到问题的最优解。另外，算法的执行时间也不是很长，虽然结点数量为1000的算例的执行时间依然达到了200多秒，但已经在可以接受的范围之内。但是，当我们调整这两个部分，来重新计算另外76个算例的时候，其计算性能受到了明显的制约，所以这个调整并不适用于所有的情况。值得一提的是，在我们的研究过程中，不论策略如何变化，参数如何设定，crane100.1和disk100.2这个两个算例虽然能很快得到和最优解非常接近的解，但就是没有进一步的改进，尤其是算例disk100.2，我们观察了算法得到的解，发现仅仅有一条边的值比最优解大，但目标函数值在算法之后的执行过程中却一直得不到更新，这也是本论文工作中留有的遗憾。

# 结论与展望

BTSP问题是比较典型的NP-hard问题，并具有很广泛的实际应用，本文主要做了如下几个工作：

1. 将BTSP转化为线性序列问题，并以块移动策略作为核心部分，提出了一种的迭代禁忌搜索算法来求解这一难题；
2. 在详细研究过程中，根据标准算例详细测试了各种算法参数，得到了针对求解这些问题的最佳参数设定。
3. 通过和JLR算法在这些算例上的测试结果作比，分析算法性能优越性。

根据测试的结果，不论是从解的质量还是计算时间上来看，都体现了迭代禁忌搜索算法在求解BTSP上的优越性能。对于绝大多数算例，我们的ITS算法都能得到最优解，而且计算时间非常之短；而对于那些没有得到最优解的算例，我们调整了小部分策略以及参数设定，测试结果显示，改动之后的ITS能在合理的时间内求解这些困难算例，可以得到问题的最优解。遗憾的是，在这86个测试算例中，仍然还有两个算例，依靠现有的算法框架和策略，计算结果一直停滞不前，不论执行时间多长，还是没能得到问题的最优解。

总的来说，本文创造性地提出了禁忌搜索算法来解决BTSP，而且取得了非常好的计算效果，在后面进一步的研究中，我们可以尝试如下几方面的工作：

1. 可以尝试调整或者应用其它的一些新策略，期望在现有算法框架的基础上进行改进，来计算这两个困难算例；
2. 可以去挖掘这两种类型的算例结构上的差异，分析策略和参数不能统一的原因，找到解决这种差异的方案，提升我们算法的适用性；
3. 考虑这种块移动策略对于一些其它类似可以转化的问题是否具有相同的计算效果，如果可行的话，这将对求解很多其它类型的NP难问题提供新的求解思路，甚至有希望找到适用于所有NP难问题的启发式算法。

以上这三个方面，都可以在本论文的基础上开展更深入的研究，得到更高层次的研究成果。

# 参考文献

[1] Wolsey LA, Nemhauser GL. Integer and combinatorial optimization[M]. John Wiley & Sons; 2014.

[2] Gutin G, Punnen AP. The traveling salesman problem and its variations[M]. Springer Science & Business Media; 2002.

[3] Punnen AP, Aneja Y. Minmax combinatorial optimization[J]. European Journal of Operational Research 1995;81:634-43.

[4] Gilmore P, Gomory R. A solvable case of the traveling salesman problem[J]. Proceedings of the National Academy of Sciences of the United States of America 1964;51:178.

[5] Kabadi SN, Punnen AP. The bottleneck TSP. The Traveling Salesman Problem and Its Variations: Springer; 2007. p. 697-735.

[6] Garfinkel RS, Gilbert K. The bottleneck traveling salesman problem: Algorithms and probabilistic analysis[J]. Journal of the ACM (JACM) 1978;25:435-48.

[7] Carpaneto G, Martello S, Toth P. An algorithm for the bottleneck traveling salesman problem[J]. Operations Research 1984;32:380-9.

[8] Sergeev S. Algorithms for the minimax problem of the traveling salesman. I. An approach based on dynamic programming[J]. Avtomatika i Telemekhanika 1995:144-50.

[9] LaRusic J, Punnen AP. The asymmetric bottleneck traveling salesman problem: algorithms, complexity and empirical analysis[J]. Computers & Operations Research 2014;43:20-35.

[10] Timofeev E. Minimax 2-connected subgraphs and the bottleneck traveling salesman problem[J]. Cybernetics and Systems Analysis 1979;15:516-21.

[11] Ramakrishnan R, Sharma P, Punnen AP. An efficient heuristic algorithm for the bottleneck traveling salesman problem[J]. Opsearch 2009;46:275-88.

[12] Ahmed ZH. A lexisearch algorithm for the bottleneck traveling salesman problem[J]. International Journal of Computer Science and Security 2010;3:569-77.

[13] Ahmed ZH. A data-guided lexisearch algorithm for the asymmetric traveling salesman problem[J]. Mathematical Problems in Engineering 2011;2011.

[14] LaRusic J, Punnen AP, Aubanel E. Experimental analysis of heuristics for the bottleneck traveling salesman problem[J]. Journal of heuristics 2012;18:473-503.

[15] Johnson DS, McGeoch LA. The traveling salesman problem: A case study in local optimization[J]. Local search in combinatorial optimization 1997;1:215-310.

[16] Lenstra JK, Kan A. Complexity of vehicle routing and scheduling problems[J]. Networks 1981;11:221-7.

[17] Jonker R, Volgenant T. Transforming asymmetric into symmetric traveling salesman problems[J]. Operations Research Letters 1983;2:161-3.

[18] Bläser M, Manthey B, Sgall J. An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality[J]. Journal of Discrete Algorithms 2006;4:623-32.

[19] Raff S. Routing and scheduling of vehicles and crews: The state of the art[J]. Computers & Operations Research 1983;10:63-211.

[20] Glover F, Lü Z, Hao J-K. Diversification-driven tabu search for unconstrained binary quadratic problems[J]. 4OR 2010;8:239-53.

[21] Lü Z, Hao J-K. Adaptive tabu search for course timetabling[J]. European Journal of Operational Research 2010;200:235-44.

[22] Lü Z, Huang W. Iterated tabu search for identifying community structure in complex networks[J]. Physical Review E 2009;80:026130.

[23] Qin T, Peng B, Benlic U, Cheng T, Wang Y, Lü Z. Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling[J]. Computers & Operations Research 2015;61:81-8.

[24] Glover F. Future paths for integer programming and links to artificial intelligence[J]. Computers & operations research 1986;13:533-49.

[25] Rego C, Gamboa D, Glover F, Osterman C. Traveling salesman problem heuristics: leading methods, implementations and latest advances[J]. European Journal of Operational Research 2011;211:427-41.

[26] Cirasella J, Johnson DS, McGeoch LA, Zhang W. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. Algorithm Engineering and Experimentation: Springer; 2001. p. 32-59.

[27] Reinelt G. TSPLIB, 8 August[EB/OL]. 2008. <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>

[28] Fischetti M, Lodi A, Toth P. Exact methods for the asymmetric traveling salesman problem. The traveling salesman problem and its variations: Springer; 2007. p. 169-205.

# 附录Ⅰ 英文缩写词

|  |  |  |
| --- | --- | --- |
| 英文缩写 | 英文全称 | 具体意义 |
| TSP | traveling salesman problem | 旅行商问题 |
| BTSP | bottleneck traveling salesman problem | 瓶颈旅行商问题 |
| ATSP | asymmetric TSP | 非对称旅行商问题 |
| NP-hard | non-deterministic Polynomial hard problem | 难以在多项式时间内求解的问题 |
| OPT | optimum | 最优解 |
| LB | lower bound | 下界 |
| BB | branch and bound | 分支定界 |
| BAP | bottleneck assignment problem | 求取分配瓶颈问题 |
| BBSSP | bottleneck biconnected spanning subgraph problem | 瓶颈双连通子图问题 |
| BBP | bidirectional bottleneck path problem | 双向关键路径问题 |
| LS | local search | 局部搜索 |
| ITS | tabu search | 迭代禁忌搜索 |
| LK | Lin-Kernighan | 以人名命名的算法 |
| JLR | John LaRusic | 以人名命名的算法 |