山东大学

硕士学位论文

用遗传算法求解TSP问题

姓名：任昊南

申请学位级别：硕士

专业：计算机技术

指导教师：邱洪泽;王良

20080405

摘 要

巡回旅行商问题（TSP）是一个组合优化方面的问题，已经成为并将继续成为测试组合优化新算法的标准问题。从理论上讲，使用穷举法不但可以求解TSP问题，而且还可以求出该问题的最优解。但是对现有的计算机来说，使用常规的穷举法在如此庞大的搜索空间中寻求最优解，几乎是不可能的。所以，各种求解TSP问题的优化算法应运而生了，本文所用到的遗传算法也在其中。

遗传算法是一种高效智能搜索方法，并行遗传算法是遗传算法研究中的一个重要方向。并行遗传算法能够提供各种大型计算问题的解决方案。Java语言提供了对并发的语言级支持，这个特性是Java的伟大创新之一，同时为并行遗传算法的设计提供了最佳的技术支持。在收集国内外相关资料，阅读了相关文献的基础上，本文系统地阐述了遗传算法的构成原理。介绍了作者借助Java语言实现的一种应用“轮盘赌”选择操作，顺序交叉操作以及启发式变异操作求解TSP问题的遗传算法。在了解和掌握并行遗传算法的基本概念和工作原理后，针对TSP问题，提出了两种基于并行遗传算法的求解方法。

第一章介绍了课题的研究背景、研究的可行性和意义，并行遗传算法的基本理论以及所面对的问题。此外还介绍了TSP问题的研究现状，并对论文内容进行了概括性综述。

第二章介绍了遗传算法及并行遗传算法模型

第三章介绍了遗传算法求解TSP问题的基本理论，提出了一种求解TSP问题的串行遗传算法模型，结合实例分析了该算法的创新之处。

第四章介绍了两种求解TSP问题的并行遗传算法模型，详细地论述了采用这两种并行模型求解TSP问题的过程。分别分析了这两种遗传算法的创新之处，并通过实例对它们之间的性能进行了比较。

第五章总结了整个研究工作，并对研究的方向进行了展望。

关键词：遗传算法，并行遗传算法，组合优化问题，巡回旅行商问题，“轮盘赌”选择，顺序交叉

ABSTRACT

TSP (Traveling Salesman Problem) is a problem of combination optimization with simple definition but difficult to be solved, which attracts many researchers in various fields including mathematics, physics, biology and artificial intelligence (AI). It has become and will continue to become a standard problem to test a new algorithm of combination optimization. Theoretically speaking, the enumeration not only can solve TSP, but also can get the best answer. But to all computers nowadays, it's hardly to obtain its best answer in such huge researching space by using common enumeration. Therefore, all kinds of algorithms to solve TSP emerged because of demand. Among of them, Genetic Algorithm (GA) is one of advanced technologies.

Genetic Algorithm (GA) are powerful search techniques. Parallel genetic algorithm (PGA) is one of the main research fields of genetic algorithms. It can efficiently satisfy the multiple needs of large-scale computing on a network of workstation clusters. Java language provides a parallel level with the language support, this characteristic is one of Java's great innovation, and for the design of parallel genetic algorithm provides the best technical support. In the collection of relevant information at home and abroad, read the related literature on the basis of this paper, systematic exposition of the principles of a genetic algorithm. In using the Java language, the author introduces a TSP genetic algorithm which uses a "roulette wheel" selection, Ordered Crossover operation and heuristic mutation. When the author understands the basic concept of parallel genetic algorithms, the author introduces two parallel genetic algorithm to solve the TSP problem.

Chapter one describes the research projects in the background, the feasibility and significance of parallel genetic algorithm, as well as the basic theory of the problems faced by. The author also introduces the background of TSP and significance of research, and the content of the thesis.

Chapter two describes the genetic algorithms and parallel genetic algorithm model

Chapter three describes the basic theory of solving TSP with genetic algorithm, and introduces a genetic algorithm model. With examples, the author analyses the innovation of the algorithm.

Chapter four describes two parallel genetic algorithm models to solve TSP, and discusses the process for TSP in detail. The author analyses the innovation of each parallel genetic algorithm, and compares their performance with examples.

Chapter five sums up the entire research work and study in the direction of the future.

Keywords: Genetic Algorithm, Parallel Genetic Algorithm, combination optimum problem, Traveling Salesman Problem, "roulette wheel" selection, Ordered Crossover

原创性声明和关于论文使用授权的说明

原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名：任昊南 日 期：2008年5月24日

关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或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；本人授权山东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、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名：任昊南 导师签名：[illegible] 日 期：2008年5月24日

第一章 绪 论

1.1 背景与意义

遗传算法的概念最早是由Bagley J.D在1967年提出的，而开始遗传算法的理论和方法的系统性研究是在1975年，这一开创性工作是由Michigan大学的John Holland[1]所实行的。遗传算法简称GA(Genetic Algorithm)，在本质上是一种求解问题的高效并行全局搜索方法。遗传算法的两大主要特点是群体搜索策略和群体中个体之间的信息相互交换，它实际上是模拟由个体组成的群体的整体学习过程，其中每个个体表示给定问题搜索空间中的一个解。遗传算法从任意一个初始化的群体出发，通过随机选择、交叉和变异等遗传操作，使群体一代一代地进化到搜索空间中越来越好的区域，直至抵达最优解。遗传算法适合于处理传统搜索方法难以解决的高度复杂的非线性问题，在模式识别、神经网络[2]、图像处理、机器学习[3]、工业优化控制、自适应控制、生物科学、社会科学等方面都得到广泛应用。

遗传算法来源于自然进化，与自然进化有着密不可分的关系，因此继承了自然进化过程所固有的并行性。伴随着遗传算法应用的深入开展，并行遗传算法（parallel genetic algorithm,PGA）及其实现的研究也变得十分重要。下面是一个遗传算法的形式化描述：

```
begin
    initialization
        产生一个初始群体
        计算第一代群体中各个个体的适应度
    repeat
        选择父代
        交叉操作
        子代变异
        计算子代的适应度
        子代取代父代，形成新的一代个体
    until some stopping criterion applies
```

end

这里所指的某种结束准则一般是指个体的适应度达到给定的阀值，或者个体的适应度的变化率为零。当我们考察上述遗传算法时，发现该算法只有在最外层结构（初始化部分）上是串行的，而在内部层次上存在着许多潜在的并发性。例如，遗传算法中适应度的计算最费时间，可以借助遗传算法的内在并行性，相互独立地计算群体内的每个个体适应度，提高算法的运行效率。

计算机硬件与软件技术的发展也为并行遗传算法的研究提供了强大的支持。硬件方面，多核已经成为不可阻挡的历史潮流。多核是在一块芯片上嵌入更多的处理单元，相对于复杂昂贵的并行系统，多核是提升系统并行性与性能的简单、经济的途径。软件方面，Java语言提供了对并发的语言级支持，这个特性正是Java的伟大创新之一，尤其是Java5的发布更是Java并发开发的质的飞跃[5]。本文正是借助Java语言的这一特性，在装备了Intel Core2 双核处理器的计算机环境中对求解TSP问题的并行遗传算法开展的研究。

“并行”在当今高性能计算技术中具有极其重要的地位，“遗传算法”在工程计算问题中得到广泛应用，这些都预示着并行遗传算法必将成为解决当今世界计算科学与工程问题的主要手段。

1.2 并行遗传算法

1.2.1 并行

并行，是指有一个以上的事件在同一时刻或同一时间段内发生。有人把并行分为几类：数据并行、分布式并行与人的并行。世界上客观事物的发展过程很多是并行的，彼此相互独立，又相互有一定的联系和制约。

1.2.2 并行遗传算法

遗传算法的并行性表现在两个方面，一个方面是遗传算法是内在并行的（inherent parallelism），即遗传算法本身非常适合大规模并行，最简单的方式是让几百上千台计算机各自进行独立种群的演化计算，运算结束才通信比较，选取最佳个体。该并行处理方式对并行系统结构没有较多限制和要求，遗传算法适合在目前所有的并行机或分布式系统上进行并行处理，而且对并行效率没有太

大影响。另一个方面是遗传算法的内含并行性（implicit parallelism），由于遗传算法采用种群的方式组织搜索，因而可以同时搜索解空间内的多个区域，并相互交流信息。本文正是以遗传算法的内含并行性作为主要的研究方向，并开展了大量研究工作。

虽然遗传算法无论是从直观上还是理论上来看都具备并行性，遗传算法的并行化好像是水到渠成、自然而然的，但是实际上标准遗传算法在并行化的过程中会遇到通信量过大的问题。因此，必须对标准遗传算法进行改造，尽量减少大量通信从而获得高效率。但是这样做可能会使遗传算法求解问题的质量有所下降，这就是遗传算法在并行化中遇到的效率与效果之间的矛盾。任何对标准遗传算法的改进都必须以尽可能少地影响其进化效果为前提。 遗传算法的并行化将在第二章详细讨论。

1.3 TSP的研究现状

巡回旅行商问题（Traveling Salesman Problem, TSP）是组合优化问题中的经典问题。组合优化问题的目标是从组合问题的可行解中求出最优解。在现实世界里存在大量组合优化问题，其中许多问题（如：TSP问题、图着色问题、分配问题、调度问题、布线问题及路由选择问题等）至今没有找到有效地多项式算法，这些问题已被证明是NP完全问题。优化问题有三个基本要素：变量、约束和目标函数。在求解过程中选定的基本参数称为变量，对变量取值的限制称为约束，表示可行方案衡量标准的函数称为目标函数。组合优化问题就是在给定的约束条件下，求目标函数最优值的问题。本文以TSP问题作为研究对象，正是由于TSP问题具有求解难度的代表性，激发了人们对优化技术的研究和对TSP问题本身的挑战，不仅TSP本身成为人们研究的热点，而且由于TSP问题的代表性，许多新的算法，理论和思想在被提出后也常常使用TSP作为测试其自身性能的标准。因此，TSP成为多种搜索、优化算法的间接比较标准，已经成为衡量优化技术成功与否的主要标准之一。

随着对优化技术的深入研究，以及计算机处理速度和内存容量的快速增长，在求解TSP问题过程中，人们取得了一个又一个的纪录。

1980年Crowder和Padberg求解了318个城市的问题。

1987年Padberg和Rinaldi将这个城市数增加到了2392个。

1992年美国Rice大学的CRPC研究小组用50台工作站使用了基于“cutting planes”算法解决了3038个城市的问题，被《发现杂志》评为当年的前50条科学新闻。

1994年，Applegate, Bixby, Chvatal等人使用若干台SPARC工作站组成的机群用了3-4 年的CPU时间解决了7397个城市的TSP问题。

1998年，CRPC研究小组使用三台Digital AlphaServer 4100s（12个处理器）组成的集群和32台Pentium-II个人计算机解决了美国13,509个城市组成的TSP问题。

2003年2月，Hisao Tamaki使用了路径融合同Lin-Kernighan启发（LKH）的变种相结合的方法发现了TSPLIB中pla33810的一个次优解。

2004 年2月，Keld Helsgaun发现了pla85900问题的一个次优解。此外，他又于2003年12月发现了7,516,353,779个节点的世界TSP问题的一条比较好的解。

1.4 研究内容与创新

1、研究内容

随着计算机技术的发展和普及，TSP问题在很多方面得到迅速发展和推广。本文将遗传算法和并行遗传算法用于TSP问题的求解，主要研究工作包括：

(1)首先在收集国内外相关资料，阅读了相关文献的基础上，理解了遗传算法的思想及理论；其次进一步对并行遗传算法的主要思想和理论进行了系统的学习，对基于并行遗传算法的算法和模型进行较为深入的研究；最后针对一些基础性的模型和算法进行较为认真的学习。

(2)通过深入分析遗传算法和TSP问题，提出借助Java语言实现的一种应用“轮盘赌”选择操作，顺序交叉操作、启发式变异操作以及位置重排变异操作求解TSP问题的遗传算法。对TSP问题实例进行测试的结果表明：该算法很容易收敛到问题的最优解。

(3)在了解和掌握并行遗传算法的基本概念和工作原理后，针对TSP问题，提出了两种并行遗传算法求解方法，并进行实现和试验。通过TSP问题实例进行测试，结果表明：两种算法都具有良好的性能。

(4) 借助Java语言，在装备了Intel Core2 T7300 2.0GHz双核处理器的计算机环境中对求解TSP问题的并行遗传算法进行研究。分别使用含有20座城市和48座城市的TSP实例进行测试，并对实验结果进行分析。

2、研究创新

（1）变异操作采用启发式变异操作与位置重排变异操作相结合的方式。使用该方式可以以较大概率生成优于父个体的子个体。

（2）本文采用没有重串的稳态繁殖方式。该方式改善了遗传算法的行为，增大了个体在种群中的分布区域，但增加了计算时间。

（3）通过顺序交叉及变异操作生成的新个体，如果其适应度大于父个体，则子个体取代父个体，该子个体将参与到生成下一代新个体的遗传操作中。取代了父个体的子个体，适应度高于父个体，在轮盘赌选择法中，其被选中进行交叉操作的概率将高于被替代的父个体。该遗传策略将加快收敛速度，提高运行效率。

（4）Java 多线程技术，使算法能够并行计算个体的适应度。提高了程序的运行效率，加快了初始化以及交叉操作后个体适应度的计算速度。

（5）并行算法一中种群中的个体被随机分配到各子群中，每个子群独立的进行选择、交叉和变异操作，新个体将替换 slaveList 表中适应度低于它的父个体生成新子群，所有新子群组成新一代的种群。再将新种群中的个体随机分配给各子群，开始新一轮的遗传操作。各子群在进行遗传操作时是并发运行的。

（6）并行算法一将 slaveList 表中的个体顺序打乱，再分给各子群，实现了个体在子群之间的传递策略，使优质个体能够在子群中传播，有助于加快收敛速度。

（7）并行算法二将每个子群的优秀个体存放在LinkedList类的对象bestList中。该最优个体与相邻子群的个体进行比较，如果优于相邻子群的个体，则取代其位置，加入到该子群中进行遗传操作。该策略实现了最优个体在子群之间的迁移。

1.5 本文的结构

第一章介绍了课题的研究背景、研究的可行性和意义，并行遗传算法的基本

理论以及所面对的问题。此外还介绍了TSP问题的研究现状，并对论文内容进行了概括性综述。

第二章介绍了遗传算法及并行遗传算法模型

第三章介绍了遗传算法求解TSP问题的基本理论，提出了一种求解TSP问题的串行遗传算法模型，结合实例分析了该算法的创新之处。

第四章介绍了两种求解TSP问题的并行遗传算法模型，详细地论述了采用这两种并行模型求解TSP问题的过程。分别分析了这两种遗传算法的创新之处，并通过实例对它们之间的性能进行了比较。

第五章总结了整个研究工作，并对研究的方向进行了展望。

第二章　遗传算法与并行遗传算法

2.1 GA 遗传算法

遗传算法主要是通过遗传操作对群体中具有某种结构形式的个体施加结构重组处理，从而不断地搜索出群体中个体间的结构相似性，形成并优化积木块以逐渐逼近最优解。

2.1.1 编码

遗传算法不能直接处理问题空间的参数，必须把它们转换成遗传空间中的由基因按一定结构组成的染色体或个体，这一转换过程就叫做编码。编码是从表现型到基因型的映射。

评价一种编码策略常采用以下三个规范:

(1)完备性(completeness):问题空间中的所有解都能被构造出来。

(2)健全性(soundness):GA中的染色体能对应所有问题空间中的候选解。

(3)非冗余性(nonredundancy):染色体和候选解一一对应。

上述的三个评价规范是独立于问题空间的普遍准则。因此，对于某个具体的应用领域而言，应该客观地比较和评价该问题空间中所用的编码方法，由此得到更好的方法或策略。以下对几种编码方法进行分析。

1、一维染色体编码

一维染色体编码是指从表现型映射到基因型后，其相应的基因呈一维排列构成染色体。一维染色体编码中最常用的符号集是二进制符号集{0，1}。基于此符号集的个体是一个二进制符号串。二进制符号串是目前遗传算法中常用的编码方法。它具有以下的优点:

(1)编码、解码操作简单。

(2)交叉、变异等遗传操作便于实现。

(3)便于利用模式定理进行理论分析。

二进制编码有其优点，但也存在着连续函数离散化时的映射误差，特别是它不能直接反映出所求问题的特定知识。目前已有许多新方法对二进制编码进行改进，其中包括实数表示、格雷码（Grey Code）表示和表表示等。

2、多参数映射编码

在组合优化问题求解中常常会碰到多参数优化问题，对于此类问题的遗传算法常采用多参数编码。其基本思想是把每个参数先进行二进制编码得到子串，再把这些子串连成一个完整的染色体。一般来讲，多参数映射编码中的每一个子串对应各自的编码参数。多参数映射编码包括：可变染色体长度编码、二维染色体编码和树结构编码。

2.1.2 群体设定

遗传操作是对众多个体同时进行的，众多个体组成了群体。在遗传算法处理流程中，继编码设计后的任务是初始群体的设定，并以此为起点一代代进化，直到按某种进化停止准则终止进化过程，由此得到最后一代。群体设定的关键问题是——群体规模，即群体中包含个体的数目如何确定。其中有两个需考虑的因素：

(1)初始群体的设定；

(2)进化过程中各代的规模如何维持。

群体规模作为遗传算法的控制参数之一，它和交叉率、变异率等参数一样，对于遗传算法效能的发挥有一定的影响。

1、初始群体设定

遗传算法中初始群体中的个体是随机产生的。一般来讲，初始群体的设定可采用如下的策略：

(1)根据问题固有知识，设法掌握最优解所占空间在整个问题空间的分布范围，然后在此分布范围内设定初始群体。

(2)先随机生成一定数目的个体，然后从中挑出最好的个体加到初始群体中。这种过程不断迭代，直到初始群体中个体数达到了预先确定的规模。

2、群体多样性

群体规模的确定受遗传操作中选择操作的影响很大。群体规模越大，群体中个体的多样性越高，算法陷入局部解的危险就越小。所以，从考虑群体多样性出发，群体规模应较大。但是群体规模太大会带来若干弊病：一是群体越大，其适应度计算次数增加，计算量也增加，从而影响算法效率；二是群体中个体生存下来的概率大多采用和适应度成比例的方法，当群体中个体非常多时，少量适应度很高的个体会被选择而生存下来，但大多数个体却被淘汰，这会影响配对库的形

成，从而影响交叉操作。另一方面，群体规模太小，会使遗传算法的搜索空间中分布范围有限，因而搜索有可能停止在未成熟阶段，引起未成熟收敛现象。显然，要避免未成熟收敛现象，必须保持群体的多样性，即群体规模不能太小。

2.1.3 适应度函数

遗传算法在进化搜索中基本上不用外部信息，仅以适应度函数为依据。适应度函数是由目标函数变换而成，对目标函数值域的某种映射变换称为适应度的尺度变换（fitness scaling）。遗传算法的适应度函数不受连续可微的约束且定义域可以为任意集合。对适应度函数的唯一要求是，针对输入可快速计算出能加以比较的非负结果[17]。这一特点使得遗传算法应用范围很广，在具体应用中，适应度函数的设计要结合求解问题本身的要求而定，适应度函数的计算是选择操作的依据，适应度函数的设计直接影响到遗传算法的性能。

2.1.4 遗传操作

遗传操作是模拟生物基因遗传的操作。在遗传算法中，通过编码组成初始群体后，遗传操作的任务就是对群体的个体按照它们对环境适应的程度施加一定的操作，从而实现优胜劣汰的进化过程。从优化搜索的角度而言，遗传操作可使问题的解，一代又一代地进化，并逼近最优解。遗传操作包括以下三个基本遗传算子:选择、交叉、变异。这三个遗传算子有如下特点:

(1)三个遗传算子的操作都是在随机扰动情况下进行的。即遗传操作是随机化操作，因此，群体中个体向最优解迁移的规则是随机的。

(2)遗传操作的效果和上述三个遗传算子所取的操作概率，编码方法，群体大小，初始群体以及适应度函数的设定密切相关。

(3)三个基本遗传算子的操作方法或操作策略随具体求解问题的不同而异。具体地讲，是和个体的编码方式直接有关。

1、选择算子

从群体中选择优胜的个体，淘汰劣质个体的操作叫选择。选择是用来确定重组或交叉个体，以及被选个体将产生多少个子代个体。选择操作是建立在群体中个体的适应度计算基础上的。目前常用的选择算子有以下几种:

(1)按比例的适应度计算

该方法是目前遗传算法中最基本也是最常用的方法，它也叫轮盘赌选择（roulette wheel selection）。在该方法中，个体的选择概率和其适应度成正比。设群体大小为n，其中个体i的适应度为f_i，则i被选择的概率p_i为

$p_i = f_i \Big/ \sum_{i=1}^{n} f_i$ 显然，概率p_i反映了个体i的适应度在整个群体的个体适应度总和中所占的比例。个体适应度越大，其被选择的概率就越高，反之亦然。

(2)基于排序的适应度计算

在基于排序的适应度计算中，种群按目标值进行排序，适应度取决于个体在种群中的序位，而不是实际的目标值。排序方法克服了比例适应度计算的尺度问题，以及在选择压力太小的情况下，选择导致搜索带迅速变窄而产生的过早收敛。排序方法比比例方法表现出更好的鲁棒性。

(3)期望值方法

在轮盘赌选择方法中，当个体数不太多时，依据产生的随机数有可能会出现不正确地反映个体适应度的选择，即存在统计误差。也就是说，适应度高的个体也有可能被淘汰，为克服这种误差，期望值方法用了如下思想。

a、计算群体中每个个体在下一代生存的期望数目：

$$M = f_i \Big/ \bar{f} = \frac{f_i}{\sum_{i=1}^{n} f_i \Big/ n}$$

b、若某个体被选中并要参与配对和交叉，则它在下一代中的生存的期望数减去0.5;若不参与配对和交叉，则该个体的生存期望数减去1。

c、在b的两种情况中，若一个个体的期望值小于零时，则该个体不参与选择。

(4)锦标赛选择方法

在锦标赛选择法中，随机地从种群中选择一定数目（Tour）个体(Tour称为竞赛规模)，其中适应度最高的个体保存到下一代。这一过程反复执行，直到保存到下一代的个体数达到预先设定的数目为止。

2、交叉算子

在自然界生物进化过程中起核心作用的是生物遗传基因的重组。同样，遗传算法中起核心作用的是遗传操作的交叉算子。所谓交叉是指把两个父代个体的部

分结构加以替换重组而生成新个体的操作。通过交叉，遗传算法的搜索能力得以飞跃提高。基因重组和交叉是遗传算法获取新优良个体的最重要的手段。

(1)单点交叉。

单点交叉中，交叉点k的范围为[1, Nvar -1]，Nvar为个体变量数目，在该点位分界相互交换变量，并生成两个新个体。下面是单点交叉的例子。

个体A　　1101|101 ——→ 1101011　　新个体A'

个体B　　0010|011 ——→ 0010101　　新个体B'

交叉点是随机设定的，当染色体长为n时，可能有n-1个交叉点设置，所以，单点交叉可以实现n-1个不同的交叉结果。

(2)多点交叉

对于多点交叉，m个交叉位置K_i可无重复随机地选择，在交叉点之间的变量间续地相互交换，产生两个新的后代，但在第一个变量与第一个交叉点之间的一段不作交换。多点交叉的破坏性可以促进解空间的搜索，而不是促进过早地收敛。因此搜索更加健壮。

(3)均匀交叉

均匀交叉将每个点都作为潜在的交叉点，随机的产生与个体等长的0 — 1掩码，掩码中的片断表明了哪个父个体向子个体提供变量值。均匀交叉的操作过程表示如下:当掩码中的位为0时，新个体A'继承旧个体A中对应的基因，当掩码中的位为1时，新个体A'继承旧个体B中对应的基因，由此生成一个完整的新个体A'。同样，可生成新个体B'。一个均匀交叉的例子表示如下:

旧个体A　　101001

旧个体B　　110101

屏蔽字　　010101

新个体A'　　111101

新个体B'　　100001

(4)其他交叉方法

针对各个问题的不同，可以提出各种各样的交叉方法。例如二维交叉，树结构交叉以及TSP问题中的部分匹配交叉(PMX)，顺序交叉(OX)和循环交叉(CX)等

（PMX，OX，CX将在第三章详细介绍）。

3、变异算子

变异算子的基本内容是对群体中个体串的某些基因值作变动。子个体变量已很小的概率或步长产生转变，变量转变的概率或步长与维数(变量的个数)成反比，与种群大小无关。例如对于二进制码串而言，变异操作就是把某些基因值取反，即1变为0，0变为1。一般来说，变异算子的基本操作是:在群体中所有个体的码串范围内随机确定基因，再以事先设定的变异概率 P_m 来对这些基因值进行变异。遗传算法导入变异的目的有两个:一是使遗传算法具有局部的随机搜索能力。当遗传算法通过交叉算子已接近最优解领域时，利用变异算子的这种局部随机搜索能力可以加速向最优解收敛。显然，此种情况下的变异率应取较小值。二是使遗传算法可维持群体多样性，以防止出现未成熟收敛现象。此时，收敛概率应取较大值。变异算子与选择交叉算子结合在一起，保证了遗传算法的有效性。在变异操作中，变异概率不能取得太大，如果变异概率大于0.5，遗传算法就退化为随机搜索，而遗传算法的一些重要的数学特性和搜索能力也就不复存在了。

2.2PGA 并行遗传算法

2.2.1 全局型——主从式模型（master-slave model）

并行系统分为一个主处理器和若干个从处理器。主处理器监控整个染色体种群，并基于全局统计执行选择操作；各个从处理器接受来自主处理器的个体进行重组交叉和变异，产生新一代个体，并计算适应度，再把计算结果传给主处理器。主从式方法在适应度计算很费时且远远超过通信时间的情况下才有效，否则通信时间超过计算时间，反而会降低速度。该方法要求有同步机制，这就会导致主进程忙而子进程闲或主进程闲而子进程忙，引起负载不平衡，效率不高。如果盲目地增加从进程数量，会导致通信量开销急剧上升。

全局并行易于实现，如果时间主要用在适应度计算上，这是一种非常有效的并行化方法，它还保留了简单遗传算法的搜索行为，因而可以直接应用简单遗传算法的理论成果。

2.2.2 独立型——粗粒度模型（coarse-grained model）

粗粒度模型是对经典遗传算法结构的扩展。在自然界中，物种的群体是由一些个体组成，将种群分成若干个子群并分配给各自对应的处理器，每个处理器不仅独立计算适应度，而且独立进行选择，交叉和变异操作，还要定期相互传送适应度最好的个体，从而加快满足终止条件的要求。这种粗粒度的并行遗传算法被称作迁移式或孤岛模型，基于粗粒度模型的遗传算法也称为分布式遗传算法（DGA）。在粗粒度模型的研究中，要解决的重要问题是参数选择，包括:迁移拓扑、迁移率、迁移周期等。

2.2.3 分散型——细粒度模型（fine-grained model）

细粒度模型也称领域模型，为群体中每一个个体分配一个处理器，每个处理器进行适应度的计算，而选择、交叉和变异操作仅在相邻的处理器之间进行，这样就能获得最大可能的并发性。细粒度模型要解决的主要问题是领域结构和选择策略。

第三章 遗传算法求解TSP问题的实现

3.1 遗传算法与组合优化

组合优化是遗传算法最基本的也是最重要的研究和应用领域之一。所谓组合优化是指在离散的、有限的数学结构上，寻找一个满足给定约束条件并使其目标函数值达到最大或最小的解。一般来说，组合优化问题通常带有大量的局部极值点，往往是不可微的、不连续的、多维的、有约束条件的、高度非线性的NP完全问题，因此，精确地求解组合优化问题的全局最优解一般是不可能的。遗传算法作为一种新型的、模拟生物进化过程的随机搜索、优化方法，近十几年来在组合优化领域得到了相当广泛的研究和应用，并已在解决诸多典型组合优化问题中显示了良好的性能和效果。

3.1.1 基于遗传算法的组合优化方法

对采用遗传算法解决组合优化问题算法的一般性描述如下：

(1)确定群体规模 n(整数)，使用随机方法或其它方法产生 n 个可能解 $x_i(k)$ ($1 \leq i \leq n$)组成初始解群；

(2)对于每一个个体 $x_i(k)$ (变量 k 为世代数，初始时 k=1)，计算其适应度 $f(x_i(k))$；

(3)对于每一个体 $x_i(k)$，计算其生存概率 $P_i(k)$：$P_i(k)=f(x_i)\Big/\sum_{i=1}^{n} f(x_i)$。然后设一个随机选择器，依据 $P_i(k)$ 以一定的随机方法产生配种个体 $x_i(k)$。

(4)产生下一代种群。选取两个配种个体 $X_1(k)$、$X_2(k)$，并依据一定的组合规则(如交叉、变异、逆转等)将 $X_1(k)$、$X_2(k)$ 结合成两个新一代的个体 $X_1(k+1)$、$X_2(k+1)$，直至新一代 n 个个体形成完毕。

(5)重复(2)至(4)步，直至满足程序终结条件(如时间上的限制或解的质量达到满意的范围等)。

上述算法在合适的条件下，其适应度函数值会逐代增加，并收敛到某一最大值。这个算法涉及的主要问题有编码方案、适应度函数设计、约束条件处理、选择机制、遗传操作，遗传算法参数确定等。在组合优化问题的实际应用中，合理地处理以上问题，构造合适的遗传算法框架是遗传优化的关键所在。

3.1.2 TSP巡回旅行商问题

TSP问题描述十分简单，简言之就是寻找一条最短的遍历n个城市的路径，或者说搜索整数子集X={1，2，…，n}(X的元素表示对n个城市的编号)的一个排列π (X={v_1，v_2，…，v_n})，使$T_d = d(v_1, v_n) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$取最小值，式中的d($v_i$，$v_{i+1}$)表示城市$v_i$到城市$v_{i+1}$的距离。

1、编码和适应度函数

根据TSP问题的要求，任意一个城市必须而且只能访问一次。因此在编码时，要求一个个体（即一条遍历路径）的染色体编码中不允许有重复的基因码[16]。

在求解 TSP 问题的各种遗传算法中，多采用以遍历城市的次序排列进行编码的方法，如码串 82631457 表示自城市 8 开始，依次经城市 2，6，3，1，4，5，7，最后返回城市 8 的遍历路径。显然，这是一种针对 TSP 问题的最自然的编码方式，这一编码方案的主要缺陷在于造成了交叉操作的困难。

另一种较为常用的编码方案是采用“边”的组合方式进行编码。例如码串 24536871 的第 1 个码 2 表示城市 1 到城市 2 的路径在 TSP 圈中，第 2 个码 4 表示城市 2 到城市 4 的路径在 TSP 圈中。这一编码方式有着与前面的“节点”遍历次序编码方式相类似的缺陷。

适应度函数常取路径长度T_d的倒数，即$f = 1/T_d$。若结合TSP的约束条件（每个城市经过且只经过一次），则适应度函数可表示为：$f = 1/(T_d + \alpha \times N_t)$，其中$N_t$是对TSP路径不合法的度量(如取$N_t$为未遍历的城市的个数)，$\alpha$为惩罚系数，常取城市间最长距离的两倍多一点(如$2.05 \times d_{max}$)。

2、交叉策略

基于 TSP 问题的顺序编码，若采取简单的一点交叉或多点交叉策略，必然

产生未能完全遍历所有城市的非法路径。解决这一问题的一种处理方法是对交叉、变异等遗传操作作适当地修正，使其自动满足 TSP 的约束条件。针对 TSP 问题的交叉操作包括三种：部分匹配交叉(PMX)、顺序交叉(OX)、循环交叉(CX)。

(1)PMX 部分匹配交叉

PMX 操作是由 Goldberg 和 Lingle 于 1985 年提出的，在 PMX 操作中先随机产生两个位串交叉点，定义这两点之间的区域为一匹配交叉区域，并使用位置交换操作来交换两个父串的匹配区域。考虑下面一个实例，如两父串及匹配区域为：

A = 9 8 5 | 4 6 7 | 1 3 2 0

B = 8 6 3 | 2 0 1 | 9 5 4 7

首先交换 A 和 B 的两个匹配区域，得到：

A’ = 9 8 5 | 2 0 1 | 1 3 2 0

B’ = 8 6 3 | 4 6 7 | 9 5 4 7

对于 A’、B’ 两子串中匹配区域以外出现的遍历重复，依据匹配区域内的位置映射关系，逐一进行交换。对于 A’ 有 2 到 4，0 到 6，1 到 7 的位置符号映射，对于 A’ 的匹配区域以外的 2，0，1 分别以 4，6，7 替换，则得：

A” = 9 8 5 | 2 0 1 | 7 3 4 6

同理可得：

B” = 8 0 3 | 4 6 7 | 9 5 2 1

这样，每个子串的次序由其父串部分地确定。

(2) OX 顺序交叉

1985 年 Davis 等人提出了基于路径表示的顺序交叉（OX）操作，OX 操作能够保留排列，并融合不同排列的有序结构单元。此方法开始也是选择一个匹配区域：

A = 9 8 5 | 4 6 7 | 1 3 2 0

B = 8 6 3 | 2 0 1 | 9 5 4 7

首先，两个交叉点之间的中间段保持不变，在其区域外的相应位置标记 X，得到：

A’ = X X X | 4 6 7 | X X X X

B’ = X X X | 2 0 1 | X X X X

其次，记录父个体 B 从第二个交叉点开始城市码的排列顺序，当到达表尾时，返回表头继续记录城市码，直至到达第二个交叉点结束，这样便获得了父个体 B 从第二个交叉点开始的城市码排列顺序为 9-5-4-7-8-6-3-2-0-1。对于父个体 A 而言，已有城市码 4，6，7 将它们从父个体 B 的城市码排列顺序中去掉，得到排列顺序 9-5-8-3-2-0-1，再将这个排列顺序复制给父个体 A，复制的起点也是从第二个交叉点开始，以此决定子个体 1 对应位置的未知码 x，这样新个体 A”为：

A” = 2 0 1 4 6 7 9 5 8 3

同样，可以产生子个体 B”为：

B” = 4 6 7 2 0 1 3 9 8 5

(3)CX 循环交叉

1987 年，Oliver 等人提出循环交叉 CX 方法，与 PMX 方法和 OX 方法不同，循环交叉的执行是以父串的特征作为参考，使每个城市在约束条件下进行重组。设两个父串为：

C = 9 8 2 1 7 4 5 0 6 3

D = 1 2 3 4 5 6 7 8 9 0

不同于选择交叉位置，我们从左边开始选择一个城市：

C’ = 9 — — — — — — — — —

D’ = 1 — — — — — — — — —

再从另一父串中的相应位置，寻找下一个城市：

C’ = 9 — — 1 — — — — — —

D’ = 1 — — — — — — — 9 —

再轮流选择下去，最后可得到：

C” = 9 2 3 1 5 4 7 8 6 1 0

D” = 1 8 2 4 7 6 5 1 0 9 3

(4)类似于 OX 的交叉

首先在两个父串中随机选择一个交配区域，如两父串及交配区域选定为：

A = 1 2 | 3 4 5 6 | 7 8 9

B = 9 8 | 7 6 5 4 | 3 2 1

然后将 B 的交配区域加到 A 的前面（或后面），A 的交配区域加到 B 的前面（或

后面）得到：

A’ ＝7 6 5 4 | 1 2 3 4 5 6 7 8 9

B’ ＝3 4 5 6 | 9 8 7 6 5 4 3 2 1

最后在 A’中自交配区域后依次删除与交配区域相同的城市码，得到最终的两子串为：

A” ＝7 6 5 4 1 2 3 8 9

B” ＝3 4 5 6 9 8 7 2 1

与其它方法相比，这种方法在两父串相同的情况下仍能产生一定程度的变异效果，这对维持群体多样化特性有一定的作用。

3、变异技术

目前已有多种变异算子，如 2-opt 变异算子、倒位变异算子、启发式变异算子等，其中启发式变异算子相对其他变异算子更有效。

启发式变异算子的操作过程如下：

设:P1 ＝ 1 2 3 4 5 6 7 8 9

随机选择三个点，例如：2、6、8，任意交换位置可以得到 5 个不同个体：

A1 ＝ 1 2 3 4 5 8 7 6 9

A2 ＝ 1 6 3 4 5 2 7 8 9

A3 ＝ 1 8 3 4 5 6 7 2 9

A4 ＝ 1 6 3 4 5 8 7 2 9

A5 ＝ 1 8 3 4 5 2 7 6 9

从中选择最好的作为新的个体。

3.2 遗传算法求解 TSP 问题的核心代码

在掌握了遗传算法和 TSP 问题的基础知识后，作者用 JAVA 语言实现了一种应用“轮盘赌”选择操作，顺序交叉操作、启发式变异操作和位置重排变异操作求解 TSP 问题的遗传算法。通过分别使用包含 20 座城市和 48 座城市的 TSP 实例测试该算法，实验结果证明该算法能够求得最优路径。

3.2.1 编码与适应度函数

采用以遍历城市的次序排列进行编码的方法，并将 n 座城市的数字表示形式转化为字母表示形式，这样可以借助 Java 语言的强大字符串处理函数来降低算法设计难度。即数字 0，1，2，3，4，5，6，7，8，9，10，11，12，13，14，15，16，17，18，19……25，26…分别对应字母序列 A，B，C，D，E，F，G，H，I，J，K，L，M，N，O，P，Q，R，S，T，……Z，a…。单个个体的遍历路径以字符串形式存储，种群中的全部个体的遍历路径存储在字符串数组中。

1、初始化种群

随机生成popsize（种群规模）个个体并将其存入字符串数组中。代码如下：

```
private void inialChroms() {                    //初始化popsize个个体
    for (int i = 0; i < popsize; i++) {
        ichrom[i] = inialChrom();               //初始化一条路径即单个个体
        }
    }
```

根据 TSP 问题的约束条件:任意一个城市必须而且只能访问一次。使用 Math 类的 random()函数随机生成等待访问的城市。如果该城市尚未访问，则将其加入到遍历路径中，否则重新生成一个待访问城市，直至全部城市都被访问到，并且每座城市仅被访问一次。代码如下：

```
private String inialChrom() {                   //初始化染色体
    String str = "";                            //str表示整条路径
    int m;                                      //m是城市的数字表示形式
    String c = "";                              //c是城市的字母表示形式
    int i = 0;
    while (i < lcity) {
        m =((int)(Math.random()*100)% lcity);//随机生成数字，代表相应城市
    switch(m){                     //将城市的数字表示生成对应的字母表示
            case 0:    c = "A";break;

            ......................

            case 19:   c = "T";break;

            ......................

            default:   c = "";break;
        }
```

```
        if(!findCity(str, c)){    //判断新生成的城市是否已经存在于路径之中
            str += c;           //新生成的城市不在路径中，将其放入到路径中
            i++;}
    }
    return str;
}
```

2、适应度计算

TSP 问题是寻找一条遍历 n 个城市的最短路径 T_d，$T_d=d(v_1,v_n)+\sum_{i=1}^{n-1}d(v_i,v_{i+1})$。适应度函数取路径长度 T_d 的倒数，即 $f=1/T_d$，核心代码如下：

```
/*求d(v1,vn)*/
double fldistance = Math.sqrt(Math.pow((icity[first][0] - city[last][0]),
2) + Math.pow((icity[first][1] - icity[last][1]), 2));
/*求其它城市之间的路径长度*/
for(int index=1;index<lcity;index++){  //求整条路径的长度
    i = temp;
    j = toInteger(str.charAt(index));//将第二个城市的字母转换成相应数字代码
    temp = j;
    distance += Math.sqrt(Math.pow((icity[i][0] - icity[j][0]), 2) +
Math.pow((icity[i][1] - icity[j][1]), 2));//计算城市之间距离并累加
}
fitness = 1/(distance+fldistance);        //求出该染色体的适应度
```

3.2.2 遗传操作设计

本算法遗传操作采用“轮盘赌”选择操作，顺序交叉操作、启发式变异操作以及位置重排变异操作。

1、选择操作核心代码

首先，计算当前世代种群中各个体的适应度，并将其存放到实数数组中。代码如下：

```
for (int i = 0; i < popsize; i++) {
    evals[i] = calculatefitnessvalue(ichrom[i]);//求各个体的适应值
    sumfitness = sumfitness + evals[i];    //所有染色体适应值总和
}
```

其次，计算当前世代种群的累积概率，代码如下：

```
for (int i = 0; i < popsize; i++) {            //求累积概率
    p[i] = evals[i] / sumfitness;           //求出各染色体选择概率
    if (i == 0){q[i] = p[i];                 //求出累计概率
    }else {q[i] = q[i - 1] + p[i];}
}
```

最后，采用轮盘赌方式进行选择。选出的个体进行交叉变异操作。

```
double r = Math.random();//采用轮盘赌选择法方式，选择1个个体，并返回该个体代码
if (r <= q[0]) {return 0;
} else {
    for (int j = 1; j < popsize; j++) {//从剩下的个体中选择
        if (r < q[j]) {return j;}       //选出后停止循环,返回选中个体的编号
    }
}
```

2、顺序交叉操作核心代码

交叉操作采用顺序交叉（OX）。如果生成的随机数小于交叉概率，则进行交叉操作，随机生成两个交叉点。

```
int pos1, pos2;                                   //两个交叉点的位置
pos1=(int)(Math.round(Math.random()*1000))%lcity;//生成的第一个交叉位置
if (pos1 == 0) {pos1 = 1;}
pos2 = pos1;
while(pos2 == pos1){                    //确保两个交叉位置不相同
    pos2=(int)(Math.round(Math.random()*1000))%lcity;//第二个交叉位置
    if (pos2 == 0) {pos2 = 1;}
}
int temp = pos1;
if(pos1 > pos2) {      //让pos1成为第一个交叉位置，即pos1小于pos2
    pos1 = pos2;
    pos2 = temp;
}
```

根据两个交叉点位置分割两个父个体，f1temp1、f1temp2、f1temp3存放父个体1分割后的三段字符串路径，f2temp1、f2temp2、f2temp3存放父个体2分割后的三段字符串路径。

```
f1temp1 = ichrom[father1].substring(0, pos1+1);        //生成父个体1第一段
f1temp2 = ichrom[father1].substring(pos1+1, pos2+1);//生成父个体1第二段
f1temp3 = ichrom[father1].substring(pos2+1);       //生成父个体1第三段
```

```
f2temp1 = ichrom[father2].substring(0, pos1+1);      //生成父个体2第一段
f2temp2 = ichrom[father2].substring(pos1+1, pos2+1);//生成父个体2第二段
f2temp3 = ichrom[father2].substring(pos2+1);      //生成父个体2第三段
```

按照OX交叉操作规则生成子个体child1与child2。

```
child1=OX(f1temp1,f1temp2,f1temp3,f2temp2);//按照ox交叉规则生成子个体1
child2=OX(f2temp1,f2temp2,f2temp3,f1temp2);//按照ox交叉规则生成子个体2
```

OX函数返回新生成的子个体，findSameCity()函数用于查询并从字符串中删除重复的城市，得到新的不含f2temp2中城市的路径。代码如下：

```
private String OX(String f1temp1, String f1temp2, String f1temp3, String
f2temp2) {
    String str="";                          //存放新生成的子个体的路径
    String temp=f1temp3 + f1temp1 + f1temp2;  //按照OX交叉规则，从第二个交
叉点开始城市码的排列顺序，到达表尾时，返回表头继续记录城市码，直至第二个交叉点结束
    int length1 = f1temp3.length();         //求出父个体1的第三段的长度
    temp = findSameCity(temp, f2temp2);//查询并剔除重复的城市，得到新的路径
    String temp1 = "";
    String temp2 = "";
    temp1=temp.substring(0,length1);    //得到子个体第三段的路径代码表示
    temp2=temp.substring(length1);      //得到子个体第一段的路径代码表示
    str = temp2 + f2temp2 + temp1;              //得到子个体
    return str;
}
```

分别计算两个子个体与其父个体的适应度。如果子个体的适应度高于相应父个体的适应度，则用该子个体替换父个体加入种群执行遗传操作，否则舍弃新生成的子个体。

```
double cfit1=calculatefitnessvalue(child1);  //计算子个体的适应度
double cfit2=calculatefitnessvalue(child2);
double ffit1=calculatefitnessvalue(ichrom[father1]);//计算父个体的适应度
double ffit2=calculatefitnessvalue(ichrom[father2]);
if (cfit1>ffit1) {//若子代的适应度高则替换父代，否则舍弃生成的子代。
    ichrom[father1] = child1;
    if(cfit2>ffit2){ichrom[father2] = child2;
    }else{
        if(cfit2>ffit2){ichrom[father2] = child2;}
    }
}
```

3、启发式变异操作核心代码

如果生成的随机数小于变异概率，则对个体进行启发式变异操作。随机选择三个变异点pos1、pos2和pos3，并对其位置排序，任意交换三个城市位置可以得到5个不同个体，从中选择适应度最好的个体作为新的个体。

```
int pos1, pos2, pos3;
pos1=(int)(Math.round(Math.random()*1000))%lcity;//生成第一个变异位置
pos2 = pos1;
while(pos2 == pos1){                //确保两个变异位置不相同
  pos2=(int)(Math.round(Math.random()*1000))%lcity;//生成第二个变异位置
}
pos3 = pos2;
while(pos3 == pos2 || pos3 == pos1){//确保三个变异位置不相同
  pos3=(int)(Math.round(Math.random()*1000))%lcity;// 生成第三个变异位置
}
```

排序后分别求得三个变异位置的城市数字代码，函数lmutation()与rmutation()返回与变异城市相邻的城市代码，函数mDistance()返回相邻三座城市之间的距离。最后计算变异发生前后三个变异位置城市与相邻城市的距离之和。

```
int city1=toInteger(str.charAt(pos1));     //求得三个变异位置的城市数字代码
int city2=toInteger(str.charAt(pos2));
int city3=toInteger(str.charAt(pos3));
int city1L=lmutation(str, pos1);//求得与三个变异位置相邻的左右城市的数字代码
int city1R=rmutation(str, pos1);
int city2L=lmutation(str, pos2);
int city2R=rmutation(str, pos2);
int city3L=lmutation(str, pos3);
int city3R=rmutation(str, pos3);
double sum1,sum2,sum3,sum4,sum5,sum6;//变换位置前后三个变异位置城市与相邻城市的距离之和
sum1=mDistance(city1L,city1,city1R)+mDistance(city2L,city2,city2R)+
mDistance(city3L, city3, city3R);
sum2 = mDistance(city1L, city2, city1R) + mDistance(city2L, city3, city2R)
+ mDistance(city3L, city1, city3R);
sum3 = mDistance(city1L, city3, city1R) + mDistance(city2L, city1, city2R)
+ mDistance(city3L, city2, city3R);
sum4 = mDistance(city1L, city3, city1R) + mDistance(city2L, city2, city2R)
+ mDistance(city3L, city1, city3R);
sum5 = mDistance(city1L, city1, city1R) + mDistance(city2L, city3, city2R)
```

```
+ mDistance(city3L, city2, city3R);
sum6 = mDistance(city1L, city2, city1R) + mDistance(city2L, city1, city2R)
+ mDistance(city3L, city3, city3R);
```

函数min()用于比较变异前后变异点位置附近的路径长度，并返回路径长度最小的变异个体代码，根据该返回值生成变异后的个体，并将其存入字符串数组中。

```
int result = min(sum1, sum2, sum3, sum4, sum5, sum6);
switch(result){   //根据result对个体进行变异
  case 1:
    str = sTemp;
    break;
  case 2:
    str = sTemp.substring(0, pos1) + sTemp.charAt(pos2) +
          sTemp.substring(pos1 + 1, pos2) + sTemp.charAt(pos3) +
          sTemp.substring(pos2+1, pos3) + sTemp.charAt(pos1) +
          sTemp.substring(pos3+1);
    break;
  case 3:
    str = sTemp.substring(0, pos1) + sTemp.charAt(pos3) +
          sTemp.substring(pos1 + 1, pos2) + sTemp.charAt(pos1) +
          sTemp.substring(pos2+1, pos3) + sTemp.charAt(pos2) +
          sTemp.substring(pos3+1);
    break;

              ……………………

  case 6:
    str = sTemp.substring(0, pos1) + sTemp.charAt(pos2) +
          sTemp.substring(pos1 + 1, pos2) + sTemp.charAt(pos1) +
          sTemp.substring(pos2+1, pos3) + sTemp.charAt(pos3) +
          sTemp.substring(pos3+1);
    break;
  default:
    break;
}
```

4、位置重排变异操作核心代码

如果生成的随机数小于变异概率，则对新生成的个体进行位置重排变异操作。随机选择两个变异点pos1、pos2，将路径分为三段，对这三段路径中城市的位置分别进行随机排列，最后将这三段路径组合成新的路径。

3.2.3 创新点与实验结果分析

1、本算法创新点

（1）用字符串数据类型表示个体的遍历路径，借助 Java 语言强大的字符串处理函数，降低了算法的实现难度。

（2）顺序交叉操作的交叉点与变异操作的变异点，均为随机产生，确保了个体的多样性，扩展了搜索空间。

（3）变异操作采用启发式变异操作与位置重排变异操作相结合的方式，使用该方式可以以较大概率生成优于父个体的子个体。

（4）当种群中个体适应度彼此非常接近时，个体进入配对集的机会相当，交配后得到的新个体也不会有多大变化。搜索过程不能有效地进行，选择机制有可能倾向于纯粹的随机选择，从而进化过程陷于停顿状态，难以找到最优解。本文采用没有重串的稳态繁殖方式解决该问题，即在形成新的种群时，使其中的个体均不重复。当将某个个体加入到新一代种群之前，先检查该个体适应度与种群中现有的个体适应度是否相同，如果相同就舍弃。该方式改善了遗传算法的行为，增大了个体在种群中的分布区域，但增加了计算时间。

（5）通过顺序交叉及变异操作生成的新个体，如果其适应度大于父个体，则子个体取代父个体，该子个体将参与到生成下一个新个体的遗传操作中。取代了父个体的子个体，适应度高于父个体，在轮盘赌选择中，其被选中进行交叉操作的概率将高于被替代的父个体。该遗传策略将加快收敛速度，提高运行效率。

2、实验结果分析

程序中坐标存放在一个二维数组中，第一维代表城市代号。第二维是 x，y 坐标，0 代表 x 坐标，1 代表 y 坐标。设定交叉率为 0.45，变异率为 0.001，种群大小为 300。两个实例的城市坐标见附录。

用该遗传算法求解含有 20 座城市TSP问题，得到的最优解与使用启发式搜索A^*算法得到的最优解相同[6]。种群大小为 200，最大运行世代数 500，最优解出现的世代数：第 230 代，共交叉 34741 次,共变异 43 次。算法求解过程中，曾出现的最优路径、路径长度值及出现的世代数见下表：

表 1 最优路径、路径长度值及世代数

世代	路径长度	路径
0	57.0662344	IJAQPGSLOFMBCNKDEHTR
1	54.8794117	MRTQBFCJGEAILODPKNHS
1	54.5231063	CIDLRFMAKNPESHOJQBTG
1	51.9279889	RFQLBANGTEMHPDSJKCI
1	47.5812465	DHILCRKNGOJEQTMSPFBA
3	45.9592676	GHRBCLIQFDEJSTMOANKP
10	45.0886288	IOSMTFKANDLHCPEBJRGQ
10	44.8148231	SPFQLANGERDKBHITMJCO
11	43.657987	HGLBCREMTPJOFQANKSDI
19	42.9599986	JRHSGEQFMTODIKNCABLP
22	42.7120798	HLJBCREMTPGOFQANKSDI
22	42.6444973	JBLHPERGMTCKDFQNASIO
22	41.2582111	KADIQMTBLHCOJFGSEPRN
37	36.0850614	DKNAHBISRPEGJCOLQFMT
56	32.9033738	NAHBISRPEGOJCLQFMTDK
82	31.9116314	TFQBOJSGRDCANKLHIPEM
93	31.9004231	KNLDOSRPEGMTQFBIHJCA
96	30.4196505	LCBIQFTMEGRSJPHODKNA
100	30.0538605	CLBIQFTMEGRSJPHODKNA
104	29.9459686	NALCBGRPEMTFQIHOSJDK
111	29.8726651	GMTQFBIHDCANKLJOSRPE
111	28.9139911	FTMSGORPEJHDKNALCBIQ
114	28.4370284	KBIQFTMPEGRSJOHDCLNA
122	28.2949	NKDJOSRPEGMTFQBIHLCA
133	28.2267851	LBIQFTMPEGRSJONKAHDC
141	28.1135203	NAKOSGRPEMTFQBIJHCLD
145	27.2275518	LBIQFTMPEGRSJOHCANKD
150	25.0995871	BIQFTMEPRSGJOHDKNACL
148	25.0995871	LBIQFTMEPRSGJOHDKNAC
190	24.9192612	IQFTMEPGRSOJHDKNACLB
195	24.7236679	IQFTMEPRGSJOHDKNACLB
230	24.5222344	TMEPRGSOJHDKNACLBIQF

根据上表绘制路径长度与世代数关系图

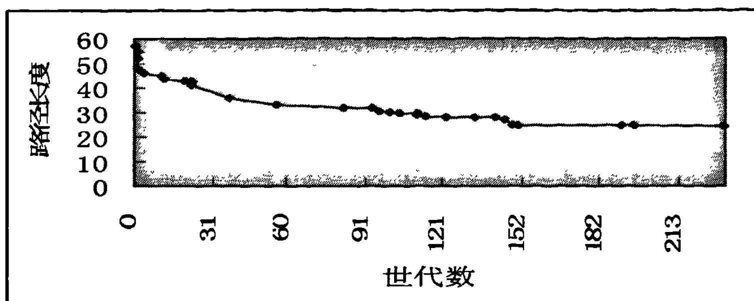

图 1 路径长度与世代数关系图

用该遗传算法求解含有 48 座城市 TSP 问题，种群大小为 300，最大运行世代数 6000。得到的最佳路径见表 2。表 2 也列出了使用文献[8]中的遗传算法思想求解 48 座城市 TSP 问题得到的最佳结果。从表 2 的对比结果可知，本文算法在相同条件下找到了更好的路径，因此本算法的遗传策略拥有较高的求解效率。

表 2 求解 48 座城市 TSP 问题优化结果对比

城市数	48	路径长度
文献[8]的优化结果	34326.49132536447	DisJXpEvmfUuMYNWKLTgnOtjFkSaQqdbGRrelIHAPVChocBZ
本文算法的优化结果	33894.45901304105	NWKnOLTgtjFkSaQqdbGRrelIHAPVChocBZDisJXpEvmfUuMY

第四章 并行遗传算法求解TSP问题的实现

4.1 并行遗传算法求解TSP问题模型

在了解和掌握并行遗传算法的基本概念与工作原理后，借助JAVA多线程技术，提出了两种基于并行遗传算法求解TSP问题的方法。两种求解方法所采用的编码方法，适应度计算方法，选择操作，交叉操作以及变异操作都与第三章中的串行遗传算法相同，具有该算法的全部优点。下面将重点介绍如何实现两种算法的并行机制。

4.2 并行遗传算法一

4.2.1 核心代码

MANAGER 线程负责初始化并启动 SLAVE 与 COUNTER 线程，每个个体被实现为一个 SLAVE 线程，COUNTER 线程根据子群数目产生 MASTER 线程。各 MASTER 线程并行进行选择操作，被选择的个体进行交叉和变异操作，生成新一代子个体。COUNTER 线程将新种群中个体的顺序随机重新排列，再分配给 MASTER 线程进行新一轮操作，直至满足终止条件停止运行。

1、MANAGER线程

MANAGER 线程负责初始化，该线程初始并启动 SLAVE 与 COUNTER 线程。每个个体被实现为一个 SLAVE 类的对象，初始化后存放在 slaveList 链表中，每个个体并行计算适应度。代码如下：

```
for(int i=0;i<SSize;i++){            //生成初代种群，SSize为种群中个体数
    Slave s = new Slave(this);       //生成一个个体
    slaveList.add(s);              //放到slaveList链表中
    Thread ts = new Thread(s);       //启动该个体线程完成适应度计算
    ts.start();
}
Counter c = new Counter(this);       //启动Counter线程,
Thread tc = new Thread(c);
tc.start();
```

当世代数大于最高世代数时，MANAGER线程向COUNTER线程发出终止信号(endflag

= true)，终止算法执行。

```
while(generation < maxgeneration){ //当世代数小于最高世代数时等待
   //wait}
endflag = true;                    //发出终止命令
```

2、COUNTER线程

线程COUNTER负责负载平衡，它根据子群数目产生MASTER线程，为MASTER线程分配相应的编号，并动态地分配个体给MASTER线程，分配的个体存放在该MASTER线程的masterList链表中。当该MASTER线程获得足够的个体就开始进行遗传操作，个体只能和子群体中的其他个体进行配对，搜索空间是局域性的。为了达到即缩短计算时间又增大搜索空间的目的，最好的办法就是将新种群的个体重新随机分配给子群，函数arrayNumber()生成随机顺序的整数序列，用于排列个体在链表中的顺序。这样适应度高的个体可以在不同的子群之间迁移，加快了收敛速度。

```
b = arrayNumber();         //生成0~~299的随机顺序的整数序列
for(i=0;i<mg.MSize;i++){//共有MSize个子种群，为子种群分配相应数目的个体
  Master ma = new Master(this, i); //生成一个Master对象，
  while(!flagc){        //为每个子种群分配相应数目的个体
    if(mg.slaveList.size()%mg.MasterSize==0&&mg.slaveList.size()!=0){
      for(j=0;j<mg.MasterSize;j++){
        ma.masterList.add(mg.slaveList.get(b[j + i * mg.MasterSize]));
      }
    flagc = true;}
  }
  Thread t = new Thread(ma);
  t.start();
  flagc = false;}
}
```

3、MASTER线程

MASTER线程获得足够的个体就开始进行遗传操作。MASTER线程执行选择操作，两个父个体经过交叉变异操作生成新个体，该遗传过程循环进行直至产生新一代子群。各MASTER线程并发执行，提高了算法运行效率。

```
while(masterNum < c.mg.MasterSize){         //生成新一代子群
  father1 = select();                  //用轮盘赌方式得到参与交叉的个体
  father2 = select();
  masterList.get(father1).cross(father2, father1, this); //进行交叉操作
```

```
masterList.get(father1).mutation1(this, father1); //进行启发式变异操作
masterList.get(father2).mutation1(this, father2);
masterList.get(father1).mutation2(this, father1); //位置重排变异操作
masterList.get(father2).mutation2(this, father2);
masterNum += 2;}
```

4、SLAVE 线程

每个 SLAVE 线程并行计算适应度，个体仅与同一 MASTER 线程中的其他个体（位于同一个 masterList 链表中）进行交叉操作。当个体收到另一个交叉个体在 LinkedList 链表中的编号及其所属的 MASTER 线程编号时，开始进行交叉操作和变异操作。

4.2.2 创新点与实验结果分析

1、本算法创新点

（1）种群中的个体存放在 Java 容器类 LinkedList 的对象 slaveList 中，分配到子群的个体存放在 LinkedList 类的对象 masterList 中。运用 Java 语言容器类的特性，降低了遗传算法的实现难度。

（2）Java 多线程技术，使算法能够并行计算个体的适应度，提高了程序的运行效率，加快了初始化以及交叉操作时个体适应度的计算速度。

（3）种群中的个体被平均分到各子群中，每个子群并行的进行选择、交叉和变异操作，新个体将替换 slaveList 表中适应度低于它的父个体。新生成的子群体组成新一代的群体，将 slaveList 表中的个体顺序打乱，然后分给各子群，开始新一轮的遗传操作。

（4）将 slaveList 表中的个体顺序打乱，再分给各子群，实现了个体在子群之间的传递策略，使优质个体能够在子群中传播，有助于加快收敛速度。

2、实验结果分析

用该遗传算法求解 TSP 问题，初始条件：20 座城市坐标与第三章串行遗传算法的坐标值相同，种群大小 300，每个子种群含 100 个个体，子种群数 3。最大运行世代数 500，最优解出现的世代数：第 182 代，共交叉 33741 次，共变异 23 次。算法求解过程中，曾出现的最优路径、路径长度值及出现的世代数见下表：

表 3 最优路径、路径长度值及世代数

代数	路径长度	路径
0	60.96945724	NJDRQFAPILSEGBKTOCHM
0	51.60404001	BRIEMHJDSGNCLPTAFQOK
0	48.76789567	LOJRKNSFQHDBCEGMAITP
0	47.3906703	TSOPALBIQFCDNKEGHJRM
1	44.77613938	NAOBILSJGMTREPCHFQDK
2	40.79699128	NABIODCPETSRGMFQLKHJ
7	40.64618378	DHNKALBIQFCPSROGETMJ
7	38.30681529	HNKALBIQFCSRPOGETMJD
8	37.71994573	HNKALBIQFCSROGPETMJD
12	34.93711195	GPESRDKANMTFQIHCBLJO
13	34.76876691	RSCLANKDHOGPETMIFQBJ
16	34.7386355	CLANKDHOGPETMFQBSRJI
23	28.8071133	KDHSOGPEMTFQBRJICLAN
40	28.38407491	DHRSOGPEMTFQBJICLANK
58	28.24827611	DSOJGRPETMFQBICLANKH
62	27.83636995	DSOJGRPEMTFQBICLANKH
67	27.79737348	DOJGRSPEMTFQBICLANKH
68	27.4619839	HSOJGRPEMTFQBICLANKD
76	26.92411683	HSOJGRPEMTFQBILCANKD
77	26.39352814	HOSJGRPEMTFQBILCANKD
102	26.30914362	HJOGSRPEMTFQBICLANKD
105	26.16295758	GSRPEMTFQBICLANKDHJO
108	26.02748976	HOJSGRPEMTFQBICLANKD
117	25.1495833	HOJSGRPEMTFQBILCANKD
123	25.14165662	GRPEMTFQIBLCANKDHOJS
125	24.75255647	GRPEMTFQBILCANKDHJOS
176	24.7236679	GRPEMTFQIBLCANKDHJOS
182	24.52223435	MTFQIBLCANKDHJOSGRPE

根据上表绘制路径长度与世代数关系图

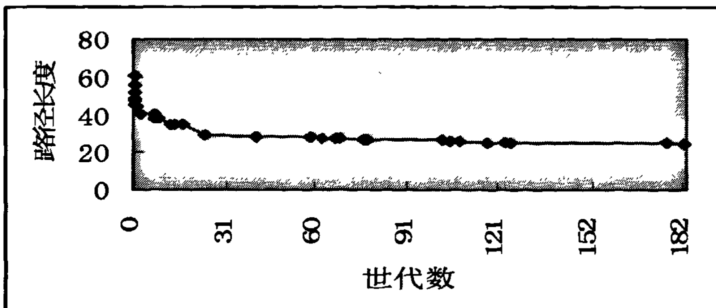

图2路径长度与世代数关系图

与第三章中的串行遗传算法相比，该算法并行计算个体适应度，提高了运行效率。三个子种群并行进行遗传操作，为提高搜索空间范围，避免其只局限于子种群内，使用函数arrayNumber()将新个体随机配给MASTER线程。该算法每生成新一代种群就需要重新分配子群，降低了运行效率，不能充分发挥并行遗传算法的性能，需要进一步改进。

4.3 并行遗传算法二

4.3.1 核心代码

该算法在 4.2 算法的基础上，对 COUNTER 与 MASTER 线程代码进行了改进，线程 COUNTER 根据子群数目产生 MASTER 线程后，每个 MASTER 线程并行执行遗传操作，运行过程中不再给子群重新分配个体。各个子群将新产生的最佳个体保存到 bestSlaveList 链表的对应位置，最佳个体在各个子群之间进行传播，实现了最优个体在子群之间的迁移，提高了遗传算法的收敛性及其搜索能力。

1、COUNTER 线程

线程 COUNTER 改进后的核心代码如下：

```
b = arrayNumber();          //生成随机顺序的整数序列
for(i=0;i<mg.MSize;i++){ //共有MSize个子种群，为每个子种群分配相应数目的个体
    Master ma = new Master(this, i);    //生成一个Master对象，
    while(!flagc){          //为每个子种群分配相应数目的个体
    if(mg.slaveList.size()%mg.MasterSize==0&&mg.slaveList.size()!=0){
     for(j=0;j<mg.MasterSize;j++){
        ma.masterList.add(mg.slaveList.get(b[j+i*mg.MasterSize]));}
    flagc = true; }
}
mg.bestSlaveList.add(i, ma.masterList.get(0));//初始化该子种群的最佳个体
Thread t = new Thread(ma);
t.start();
flagc = false;
}
```

2、MASTER 线程

线程MASTER改进后的核心代码如下：

```
while(masterNum < c.mg.MasterSize){         //生成新一代子群
  father1 = select();                    //用轮盘赌方式得到参与交叉的个体
```

```
  father2 = select();
  masterList.get(father1).cross(father2, father1, this); //进行交叉操作
  masterList.get(father1).mutation1(this, father1); //进行启发式变异操作
  masterList.get(father2).mutation1(this, father2);
  masterList.get(father1).mutation2(this, father1); //位置重排变异操作
  masterList.get(father2).mutation2(this, father2);
  masterNum += 2;}
masterNum = 0;
maGeneration++;
```

子群将新产生的最佳个体保存到bestSlaveList链表的对应位置。

```
    c.mg.bestSlaveList.get(mCode).fitness = bd.fitness;
    c.mg.bestSlaveList.get(mCode).route = bd.route;
    c.mg.bestSlaveList.get(mCode).generations = bd.generations;
```

最佳个体在各个子群之间进行传播，如果该最佳个体的适应度大于其他子群中对应个体的适应度，则该最佳个体取代该个体参与遗传操作。

```
for(int i=0;i<c.mg.MSize;i++){
  if(mCode != i){
    if(c.mg.bestSlaveList.get(i).fitness>
              this.masterList.get(i).fitness){
      masterList.remove(i);
      masterList.add(i,c.mg.bestSlaveList.get(i));}
    }
}
```

4.3.2 创新点与实验结果分析

1、本算法创新点

（1）种群分为三个子群后，三个子群并行进行遗传操作。新个体将替换masterList表中适应度低于它的父个体，运行过程中不再给子群重新分配个体，三个子群在整个遗传操作过程中都是并行运行的。

（2）每个子群的优秀个体存放在LinkedList类的对象bestList中，最优个体与相邻子群的个体进行比较，如果优于相邻子群的个体，则取代其位置，加入到该子群的遗传操作中，该策略实现了最优个体在子群之间的迁移。

2、实验结果分析

用该遗传算法求解TSP问题，初始条件：20座城市坐标与第三章串行遗传算法的坐标值相同，种群大小300，每个子种群含100个个体，子种群数3。最大运行世代数500，最优解出现的世代数：第162代，交叉次数33656次，变异次数282

次。算法求解过程中，曾出现的最优路径、路径长度值及出现的世代数见下表：

表 4 最优路径、路径长度值及世代数

世代	路径长度	路径
0	64.613431	NJDRQFAPILSEGBKTOCHM
1	58.051266	BRIEMHJDSGNCLPTAFQOK
1	55.027136	LOJRKNSFQHDBCEGMAITP
1	42.695238	TSOPALBIQFCDNKEGHJRM
3	42.112978	TSOALBCDNKGJFQHIERPM
3	42.112945	NAOBILSJGMTREPCHFQDK
8	40.836023	NABIODCPETSRGMFQLKHJ
9	37.612989	DHNKALBIQFCPSROGETMJ
10	36.497918	HNKALBIQFCSRPOGETMJD
12	35.84243	HNKALBIQFCSROGPETMJD
18	35.543656	GPESRDKANMTFQIHCBLJO
20	33.670253	RSCLANKDHOGPETMIFQBJ
31	31.565478	CLANKDHOGPETMFQBSRJI
41	31.405907	KDHSOGPEMTFQBRJICLAN
43	29.608176	DHRSOGPEMTFQBJICLANK
48	29.381715	DSOJGRPETMFQBICLANKH
49	27.666759	DSOJGRPEMTFQBICLANKH
55	26.868718	DOJGRSPEMTFQBICLANKH
58	26.718657	HSOJGRPEMTFQBICLANKD
59	26.3782	HSOJGRPEMTFQBILCANKD
60	26.171565	HOSJGRPEMTFQBILCANKD
62	25.251792	GSRPEMTFQBICLANKDHJO
63	25.251792	HJOGSRPEMTFQBICLANKD
70	25.180032	HOJSGRPEMTFQBICLANKD
75	24.839575	HOJSGRPEMTFQBILCANKD
76	24.723668	GRPEMTFQIBLCANKDHOJS
86	24.638141	GRPEMTFQBILCANKDHJOS
97	24.522234	GRPEMTFQIBLCANKDHJOS
138	24.522234	MTFQIBLCANKDHJOSGRPE
162	24.522234	QIBLCANKDHJOSGRPEMTF

根据上表绘制路径长度与世代数关系图

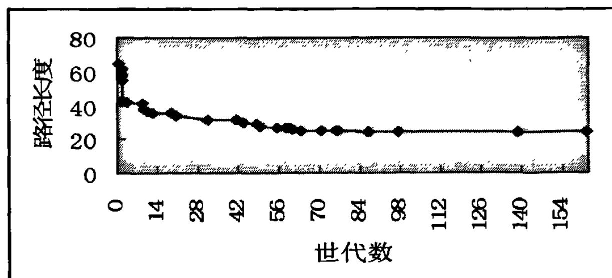

图3路径长度与世代数关系图

改进后的算法，具有完全的可扩展性和灵活性。并行遗传算法的迁移策略具有对‘精英’染色体的保护作用，提高了遗传算法的收敛性及其搜索能力。

4.4 与其它并行遗传算法比较

为了便于比较和分析，本文选择了城市数目分别为20和48的两个TSP问题进行了实验，种群规模为300，子群数目为3，最大遗传世代数6000，城市坐标见附录。本文的两个并行遗传算法优化后的最佳路径长度如表5所示。表5还列出了使用文献[12]中的遗传算法思想与文献[13]中的遗传算法思想求解这两个TSP问题得到的最佳结果。从表5中的对比结果可知，对于20座城市的TSP问题，本文的两个算法与其他算法都找到了相同的最佳路径。对于48座城市的TSP问题，本文的两个算法在相同条件下找到了比表5中其它算法更好的路径。比较本文中的两个算法，算法二的结果好于算法一的结果，说明在算法一基础上的改进，使算法二的收敛性和搜索能力得到了提高。

表5 两个TSP问题的优化结果

城市数	20	48
文献[12]的优化结果	24.522234352060764	34411.19110135375
文献[13]的优化结果	24.522234352060764	34195.055288027055
本文并行算法一的优化结果	24.522234352060764	33966.140475810505
本文并行算法二的优化结果	24.522234352060764	33164.66189281454

实验结果表明，本文提出的算法提高了全局和局部搜索能力，并避免了未成熟收敛缺陷的出现。

第五章 结束语

5.1 研究总结

本文在深入分析和研究了遗传算法基本理论与方法的基础上，提出了求解TSP问题的遗传算法，以及用并行遗传算法求解TSP问题的两种方法。总结整个研究工作，以下是我们得出的主要结论与成果:

(1)针对TSP问题的特征，设计了交叉、变异算子，设计出的遗传算法不仅能保持父代优良模式，而且能以很大概率生成好的模式。

(2)借助JAVA语言的多线程特点，提出两种求解TSP问题的并行遗传算法。

(3)遗传算法是求解TSP问题的有效算法，但传统的遗传算法在求解TSP问题时也存在一些不足之处，如在远离最优点的地方停滞不前和过早收敛等。本文采用没有重串的稳态繁殖方式解决该问题，改善了遗传算法的行为，增大了个体在种群中的分布区域，但增加了计算时间。

(4)在使用遗传算法求解具体问题时，算法的设计应该与问题特点相结合，才能达到更好的求解效果。例如本文中TPS问题的解路径不能包含重复的城市，那么编码时就一定要考虑到这一点。

研究成果表明:我们找到了适合于求解TSP问题的遗传操作方法和遗传策略，并在模拟实验中得到了检验和证明。但还存在许多问题有待于进一步的深入研究，比如寻找更好的交叉和变异算子，以及与之相适应的遗传策略。

5.2 研究展望

在本文研究基础上，还可以进一步开拓以下几个方向的研究工作:

(1)将遗传算法运用于大规模的旅行商问题。

(2)对于算法的理论进行进一步研究。

(3)将许多实际应用问题(如印制电路板的钻孔路线方案、连锁店的货物配送路线等)建模为TSP问题，并应用并行遗传算法来解决。

(4)获得更好的性能是并行遗传算法研究中的重要课题。在保证求解质量的同时降低通信开销、针对具体问题寻找优化的并行计算策略和群体划分策略，也是并行遗传算法中需要进一步深入研究的重要课题。

遗传算法是新发展起来的一门学科，各种理论、方法尚未成熟，需要进一步地发展和完善，但它已经为解决许多复杂问题提供了希望。尽管在遗传算法的研究和应用过程中会出现许多难题，同时也会产生许多不同的算法设计观点，但是，目前遗传算法的各种应用实践已经展现出了其优异的性能和巨大的发展潜力，它的发展前景激励着各类专业技术人员把遗传算法的理论和方法运用于自己的专业领域中。随着研究工作的进一步深入和发展，遗传算法必将在智能计算领域中起到关键的作用。

附 录

附表一：20 座城市 TSP 坐标

城市	X	Y	城市	X	Y	城市	X	Y	城市	X	Y
A	5.294	1.558	F	4.771	6.041	K	4.399	1.194	P	1.072	3.454
B	4.286	3.622	G	1.524	2.871	L	4.66	2.949	Q	5.855	6.203
C	4.719	2.774	H	3.447	2.111	M	1.232	6.44	R	0.194	1.862
D	4.185	2.23	I	3.718	3.665	N	5.036	0.244	S	1.762	2.693
E	0.915	3.821	J	2.649	2.556	0	2.71	3.14	T	2.682	6.097

附表二：48 座城市 TSP 坐标

城市	X	Y	城市	X	Y	城市	X	Y	城市	X	Y
1	6734	1453	13	4706	2674	25	4307	2322	37	7762	4595
2	2233	10	14	4612	2035	26	675	1006	38	7392	2244
3	5530	1424	15	6347	2683	27	7555	4819	39	3484	2829
4	401	841	16	6107	669	28	7541	3981	40	6271	2135
5	3082	1644	17	7611	5184	29	3177	756	41	4985	140
6	7608	4458	18	7462	3590	30	7352	4506	42	1916	1569
7	7573	3716	19	7732	4723	31	7545	2801	43	7280	4899
8	7265	1268	20	5900	3561	32	3245	3305	44	7509	3239
9	6898	1885	21	4483	3369	33	6426	3173	45	10	2676
10	1112	2049	22	6101	1110	34	4608	1198	46	6807	2993
11	5468	2606	23	5199	2182	35	23	2216	47	5185	3258
12	5989	2873	24	1633	2809	36	7248	3779	48	3023	1942

参考文献

[1] Holland J H. Adaptation in Nature and Artificial Systems[M]. Michigan: University of Michigan press, 1975.

[2] Hopfield J J. Neural networks and physical systems with emergent collective computational abilities[J]. Proceedings of National Academy of Scientists, 1982 ,79:2 554-2 558.

[3] Goldberg D E. Genetic Algorithms in Search Optimization and Machine Learning[M]. Addison-Wesley, 1989.

[4] Oliver L M, et. al. A study of Permutation Crossover Operators On the Traveling Salesman Problem[C]. In: Proc. of 2nd Int. Conf. On Genetic Algorithm, Lawrence Erlbaum Associates, 1987.

[5] Brian Goetz. Java Concurrency In Practice[M]. Addison-Wesley, 2006: 4-5.

[6] 王小平，曹立明. 遗传算法理论应与软件实现[M]. 西安:西安交通大学出版社, 2002:129.

[7] 侯建花,杨长青. 一种求解 TSP 问题的并行遗传算法[J]. 计算机仿真,2005, 22(2): 83-84.

[8] 佘一娇. 用简单遗传算法求解TSP问题的参数组合研究[j]. 华中师范大学学报(自然科学版),2002, 36(1): 25-26.

[9] 陈国良,等. 遗传算法及其应用[M]. 北京:人民邮电出版社,1996.

[10] 周明,孙树栋. 遗传算法原理及应用[M]. 北京:国防工业出版社,1997.

[11] 李晓梅,莫则尧. 面向结构的并行算法-设计与分析[M]. 长沙:国防科技大学出版社,1996.

[12] D Whitley etal. Scheduling problems and traveling salesman : the genetic edge recombination operator[C]. proc. of 3rd Int. conf on genetic Algorithms, 1989:133-140.

[13] D B Fogel. Applying evolutionary programming to selected traveling salesman problems[J]. Cybemtics and systems, 1993, (24):27-36.

[14] Davis L D, Handbook of Genetic Algorithms[M], Van Nostrand reinhold,

1991.

[15] Michalewicz Z. Genetic Alorithms+Data Structures=Evolution Programs[M]. 3rd ed berlin: Springer verlag 1996.

[16] Srinivas M, Patnaik L M. Adaptive probabilities of crossover and mutation in genetic algonithms[J]. IEEE Trabsactions on Systems, Man and Cybernetics, 1994,24(4):656-666.

[17] Fogel DB. App;ying evolutionary programming to selected traveling sales man problems[J].Cybernetics And System, 2001, vol.24:27-36.

致 谢

首先要感谢邱洪泽教授。如果没有您对我的指导与支持，我将很难完成本文问题的研究。还要感谢您不厌其烦地回答了我大量的问题，邱教授每次平易近人和极为认真的回复，都让我在研究问题的过程中受益匪浅。

感谢张立新老师，您给予我的关怀与建议，为我的学习研究提供了极大的帮助。在您的帮助与鼓励下，我的研究工作才能顺利进行。

感谢王良高级工程师，没有您的帮助，我的研究工作无法顺利完成。

还要感谢山东省第一劳教所的领导与技术科的各位同事，是你们的帮助，让我有更多的时间能够投入到研究工作中。

更要感谢山东大学对我的教育培养，使我学业有成，理论水平得到提高，为做好本职工作打下了坚实基础。