学　　号：	0121310880330

武漢理工大學

课 程 设 计

题　　目	基于粒子群算法的 TSP 求解
学　　院	计算机科学与技术学院
专　　业	软件工程
班　　级	SY1301
姓　　名	郑文博
指导教师	李晓红

2014 年 12 月 18 日

课程设计任务书

学生姓名：郑文博　　专业班级：软件工程 SY1301

指导教师：李晓红　　工作单位：计算机科学与技术学院

题　目：　　基于粒子群算法求解 TSP 问题

初始条件：

理论：学习了《数据结构》课程，掌握了一种计算机高级语言。
实践：计算机技术系实验中心提供计算机及软件开发环境。

要求完成的主要任务：（包括课程设计工作量及其技术要求，以及说明书撰写等具体要求）

1、分析 TSP 问题以及粒子群算法
2、数据结构设计；
3、主要算法设计；
4、编程及上机实现；
5、撰写课程设计报告，包括：
（1）设计题目；
（2）摘要和关键字（中文和英文）；
（3）正文，包括数据结构设计、算法设计、有关技术的讨论、设计体会等；
（4）结束语；
（5）参考文献。

时间安排：

指导教师签名：　李晓红　　2014 年　12 月　18 日

系主任（或责任教师）签名：　　2014 年　12 月 18 日

目录

摘要

PSO 算法的基本思想来源于对鸟群简化社会模型的研究及行为模拟，其中的每个个体充分利用群体的与自身的智能，不断地调整学习，最终得到满意解。在 PSO 算法中，每个备选解是搜索空间中的一个粒子，每个粒子根据它自身的经验和粒子群的最佳经验，在问题空间中向更好的位置飞行，如此循环搜索直到发现最优解。该算法在连续优化问题的运用中取得了较好的效果.但较少用于 TSP 等组合优化问题的求解。本文意图通过对 PSO 算法进行简单改进，达到解决对称性 TSP 问题的效果。

本文利用贪心算法得到近似最优循环序列，产生一定规模的初始种群。这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。又利用生物种群内部独立进行 PSO 进化的原理，种群个体最优之间以一定概率进行交叉。两个种群同时寻优可以减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

在本文的最后，根据算法，给出了 java 代码的实现并给出了实验结果。

关键字：PSO 算法 初步改进 TSP 问题

ABSTRACT

PSO algorithm is derived from the basic idea of the study of birds simplified social model and behavior simulation, each of these individuals to make full use of the groups with their own intelligence, constantly adjust learning, finally get satisfied solution. In PSO algorithm, each alternative solution is a particle in search of empty asked, each particle according to its own experience and the best experience of particle swarm in questions empty to better position in the flight, so the search until you find the optimal solution. The algorithm in the continuous optimization problems using the good results have been achieved. But less for solving combinatorial optimization problems such as TSP. Intention in this paper, based on the improved PSO algorithm briefly, achieves the result that symmetric TSP problem.

In this paper, using the greedy algorithm to get the approximate optimal cycle sequence, produce certain size of initial population. That generates the initial population of the global optimal value has close to solution of the problem, so you can save search time and improve the algorithm convergence speed. And using the population inside the PSO evolution principle of independence, with a certain probability of crossover between the optimal population size. Two species at the same time optimization can reduce the probability algorithm falls into local optimum, individual optimal cross between species can enhance and information sharing between the particles between the two populations, the optimal value message in time, improves the speed of particles to the better solutions for evolution.

At the end of this article, according to the algorithm, the realization of Java code and the experimental results is given.

Key words: preliminary improved PSO algorithm TSP problem

1 绪论

1.1 引言

旅行商问题(Traveling Salesman Problem，简称 TSP)是一个典型的 NP 完全问题。问题描述:给定 N 个城市和两两城市之间的距离，求一条访问各个城市且仅访问一次的最短路线。虽其数学描述很简单，却无法找到一个确定的算法在多项式时间内求解旅行商问题。

粒子群算法(Particle Swarm Optimization，简称 PSO)由美国社会心理学家 James Kennedy 和电气工程师 Russell Eberhart 于 1995 年提出的一种全局优化算法，该算法是一种基于群体智能的优化算法，它是受鸟群和鱼群群体运动的行为方式启发而得到的。该算法的基本思想来源于对鸟群简化社会模型的研究及行为模拟，其中的每个个体充分利用群体的与自身的智能，不断地调整学习，最终得到满意解。在 PSO 算法中，每个备选解是搜索空间中的一个粒子，每个粒子根据它自身的经验和粒子群的最佳经验，在问题空间中向更好的位置飞行，如此循环搜索直到发现最优解。该算法在连续优化问题的运用中取得了较好的效果.但较少用于 TSP 等组合优化问题的求解。

本文对基本 PSO 算法中粒子的位置、速度以及操作进行了重新定义，使粒子群算法适合于求解 TSP 问题，并采用贪心算法的思想每一步都取局部最优。这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。针对粒子群算法易陷入局部最优的缺陷，进行了改进措施，对粒子群进行分析，实现了粒子之间的信息交换， 扩大了粒子的搜索空间，克服了 PSO 算法易陷入局部最优的缺陷。两个种群同时寻优，种群内部独立进行 PSO 进化，种群个体最优之间以一定概率进行交叉。两个种群同时寻优可以减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

最后本文对 att48 的 TSP 问题进行仿真测试，试验表明改进后的粒子群算法能有效地求解 TSP 问题

1.2 选题研究背景

粒子群优化（简称 PSO）算法在多维连续优化问题的应用中取得了较好的效果，但在旅行商（简称 TSP 问题）等组合优化问题中的应用则相对较少。为使粒子群算法适合于求解 TSP 问题本文对基本 PSO 算法中粒子的位置、速度以及操作进行了重新定义。初始种群的产生借鉴贪心算法的思想，每一步都取局部最优。这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。针对粒子群算法易陷入局部最优的缺陷，进行改进，实现了粒子之间的信息交换，扩大了粒子的搜索空间，避免了算法陷入局部最优。两个种群同时寻优，种群内部独立进行 PSO 进化，种群个体最优之间以一定概率进行交叉，减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

2 TSP 问题

2.1TSP 问题定义

TSP 是运筹学、图轮和组合优化中的 NP 难问题。问题的具体如下：给定 N 个城市及两两城市之间的距离，求一条经过各城市一次且仅一次的最短路线。其图论描述为：给定图 G=（V，A），其中 V 为顶点集，A 为各顶点相互连接组成的弧集，已知各顶点间连接距离，要求确定一条长度最短的 Hamilton 回路，即遍历所有顶点一次且仅一次的最短回路。设d_{ij}为城市 i 与 j 之间的距离，即弧(i,j)的长度。引入决策变量：

$$X_{ij}=\begin{cases}1 & \text{若旅行商访问城市i 后访问城市j}\\ 0 & \text{否则}\end{cases} \qquad (1.1)$$

则 TSP 的目标函数为

$$Min\ Z=\sum_{i,j=1}^{n} X_{ij}d_{ij} \qquad (1.2)$$

TSP 问题描述非常简单，但最优化求解很困难，若用穷举法搜索，则要考虑所有可能情况，并两两对比，找出最优,其算法复杂性呈指数增长, 即所谓的“组合爆炸”。所以，寻求和研究 TSP 的有效启发式算法,是问题的关键。

2.2TSP 问题传统分析

旅行商问题要从图 G 的所有周游路线中求取最小成本的周游路线，而从初始点出发的周游路线一共有(n-1)!条，即等于除初始结点外的 n-1 个结点的排列数，因此旅行商问题是一个排列问题。排列问题比子集合的选择问题通常要难于求解得多,这是因为 n 个物体有 n!种排列,只有 个子集合(n!>O())。通过枚举(n-1)!条周游路线，从中找出一条具有最小成本的周游路线的算法，其计算时间显然为 O(n!)。

2.2.1 枚举法思想

程序中采用深度优先策略。枚举算法的特点是算法简单，但运算量大，当问题的规模变大，循环的阶数越大，执行的速度越慢。如果枚举范围太大（一般以不超过两百万次为限），在时间上就难以承受。在解决旅行商问题时，以顶点 1 为起点和终点，然后求{2…N}的一个全排列，使路程 1→{2…N}的一个全排列→1 上所有边的权（代价）之和最小。所有可能解由（2，3，4，…，N）的不同排列决定。

为便于讨论，介绍一些关于解空间树结构的术语。在下面分析回溯法和分支

限界法时都直接或间接用到解空间树。在解空间树中的每一个结点确定所求问题的一个问题状态（problem state）。由根结点到其它结点的所有路径则确定了这个问题的状态空间（state space）。解状态（solution states）表示一些问题状态 S，对于这些问题状态，由根到 S 的那条路径确定了这解空间中的一个元组。答案状态（answer states）表示一些解状态 S，对于这些解状态而言，由根到 S 的这条路径确定了这问题的一个解（即，它满足隐式约束条件）。解空间的树结构称为状态空间树（state space tree）。

对于旅行商问题，一旦设想出一种状态空间树，那么就可以先系统地生成问题状态，接着确定这些问题状态中的哪些状态是解状态，最后确定哪些解状态是答案状态，从而将问题解出。为了生成问题状态，采用两种根本不同的方法。如果已生成一个结点而它的所有儿子结点还没有全部生成，则这个结点叫做活结点。当前正在生成其儿子结点的活结点叫 E-结点。不再进一步扩展或者其儿子结点已全部生成的生成结点就是死结点。在生成问题状态的两种方法中，都要用一张活结点表。在第一种方法中，当前的 E-结点 R 一旦生成一个新的儿子 C，这个儿子结点就变成一个新的 E-结点，当完全检测了子树 C 之后，R 结点就再次成为 E-结点。这相当与问题状态的深度优先生成。在第二种状态生成方法中，一个 E-结点一直保持到死结点为止。这两种方法中，将用限界函数去杀死还没有全部生成其儿子结点的那些活结点。如果旅行商问题要求找出全部解，则要生成所有的答案结点。使用限界函数的深度优先结点生成方法称为回溯法。E-结点一直保持到死为止的状态生成方法称为分支限界法。

2.2.2 回溯法思想

为了应用回溯法，所要求的解必须能表示成一个 n- 元组(x1,…,Xn)，其中 x1 是取自某个有穷集 Si。通常，所求解的问题需要求取一个使某一规范函数 P(x1,…,Xn)取极大值（或取极小值或满足该规范函数条件）的向量。

假定集合 Si 的大小是 mi,于是就有 m=m1m2…Mn 个 n-元组可能满足函数 P。所谓硬性处理是构造这 m 个 n-元组并逐一测试它们是否满足 P,从而找出该问题的所有最优解。而回溯法的基本思想是,不断地用修改过的函数 Pi(x1,…Xi)（即限界函数）去测试正在构造中的 n-元组的部分向量(x1,…,Xi)，看其是否可能导致最优解。如果判定(x1,…,Xi)不可能导致最优解，那么就可能要测试的后 n-i 个元素组成的向量一概略去。因此回溯法作的次数比硬性处理作的测试次数（m 次）要少得多。用回溯法求解的旅行商问题，即在枚举法的基础上多了一个约束条件，约束条件可以分为两种类型：显示约束和隐式约束。

分支限界法思想：本题采用 FIFO 分支限界法。

如前所述，分支限界法是在生成当前 E-结点全部儿子之后再生成其它活结点的儿子,且用限界函数帮助避免生成不包含答案结点子树的状态空间的检索方法。在总的原则下，根据对状态控件树中结点检索的次序的不同又将分支限界设计策路分为数种不同的检索方法。在求解旅行商问题时，程序中采用 FIFO 检索（First In First Out），它的活结点表采用一张先进先出表（即队列）。可以看出,分支限界法在两个方面加速了算法的搜索速度,一是选择要扩展的节点时,总是选择选择一个最小成本的结点,尽可能早的进入最有可能成为最优解的分支;二是扩展节点的过程中，舍弃导致不可行解或导致非最优解的子结点。

2.2.3 贪心法思想

贪心法是一种改进了的分级处理方法。它首先对旅行商问题进行描述，选取一种度量标准。然后按这种度量标准对 n 个输入城市排序，并按序一次输入一个城市。如果这个输入和当前已构成在这种量度意义下的部分最优解加在一起不能产生一个可行解，则不把这个城市加入到这部分解中。这种能够得到某种量度意义下的最优解的分级处理方法成为贪心方法。

获得最优路径的贪心法应一条边一条边地构造这棵树。根据某种量度来选择将要计入的下一条边。最简单的量度标准是选择使得迄今为止计入的那些边的成本的和有最小增量的那条边。

3 PSO 算法分析

PSO 算法主要模拟鸟群飞行的觅食行为，通过鸟群的集体协作达到寻优目的。在 PSO 算法中，每个粒子利用自身的历史最优位置和整个粒子群的全局最优解提供的信息，在解空间内不断飞行， 实现寻找最优解的目的。

3.1 基本 PSO 算法的数学描述

设搜索空间为 N 维，总粒子数为 Num，第 i 个粒子的位置向量 $X_i=(x_{i1},x_{i2},x_{i3},...,x_{in})$，第 i 个粒子的速度向量是 $V_i=(v_{i1},v_{i2},v_{i3},...,v_{in})$， 第 i 个粒子在“飞行”中的历史最优位置是 $P_i=(p_{i1},p_{i2},p_{i3},...,p_{in})$， P_g 表示目前为止在整个粒子群中发现的全局最优粒子；粒子按如下方式飞行：

$$v_{ij}(t+1) = w \times v_{ij}(t) + c1 \times rand1(\,) \times [P_{ij}(t) - x_{ij}(t)] + c2 \times rand2(\,) \times [P_{gj}(t) - x_{ij}(t)] \quad (2.1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (2.2)$$

其中，下标“j”表示第 j 维，t 为飞行的次数；w 为惯性权重，使粒子保持运动惯性，控制前一速度对当前速度的影响，较大的 w 适于对解空间进行大规模探查，较小的 w 适合于进行小范围开挖；c1，c2 为加速常数，通常在 0 ～ 2 间取值，c1 调节粒子飞向自身最好位置方向的步长，c2 调节粒子向全局最好位置飞行步长；rand1( )和 rand2 ( )是[0,1]之间相互独立的随机数。粒子的位置

向量中各维变化范围为[$X_{\min}$, $X_{\max}$]，速度变化范围为[$V_{\min}$, $V_{\max}$]，迭代中若位置和速度超过边界范围则取边界值。PSO 算法通过粒子在解空间内不断地变化速度向量来改变位置，最终找到最优解。

3.2 基本粒子群算法的流程

Step1：依照初始化过程，对粒子群的随机位置和速度进行初始设定；

Step2：计算每个粒子的适应值；

Step3：对于每个粒子，将其适应值与所经历过的最好位置P_i的适应值进行比较，若较好，则将其作为当前的最好位置；

Step4：对每个粒子，将其适应值与全局所经历的最好位置P_g的适应值进行比较，若较好，则将其作为当前的全局位置；

Step5：根据方程(2.1)、(2.2)对粒子的速度和位置进行迭代进化；

Step6：循环步骤 2-5，直到结束条件为足够好的适应值或达到一个预设最大迭代次数，算法终止。

3.3 粒子群算法的通常应用

近年来，PSO 快速发展，在众多领域得到了广泛应用。本文将应用研究分典型理论问题研究和实际工业应用两大类。典型理论问题包括：组合优化、约束优化、多目标优化、动态系统优化等。

实际工业应用有：电力系统、滤波器设计、自动控制、数据聚类、模式识别与图像处理、化工、机械、通信、机器人、经济、生物信息、医学、任务分配、TSP 等等。

4 基于粒子群算法进行分析 TSP 问题

4.1 基于算法重新定义问题

TSP 问题为离散问题，用 PSO 求解 TSP 问题需要对基本 PSO 算法中粒子的位置、速度以及操作进行重新定义：

(1)状态空间

TSP 问题的结果是要求出具有最短路径的哈密尔顿圈，所以状态空间即为所有位置的集合。

(2)粒子的位置

粒子的位置可以定义为一个具有所有节点的哈密尔顿圈，设所有n_i与n_{i+1}之间的弧存在，粒子的位置可表示为序列$x=(n_1,n_2,...,n_N,n_{N+1})$，其中$n_i \in E, n_1 = n_{N+1}$，E 为状态空间。

(3)粒子的速度

速度定义为粒子位置的变换集，表示一组置换序列的有序列表。可以表示为：

$$v=\{(i_k,j_k),(i_k,j_k,)\in\{1,2,\cdots,N\},k\in\{1,2,\cdots,\|v\|\}\} \quad (3.1)$$

其中，||v||表示该速度所含交换的数目，式(3.1)表示先交换粒子中n_{i1}、n_{j1}的位置，然后交换n_{i2}、n_{j2}的位置，依此类推。

(4)粒子位置与速度的加法操作

该操作表示将一组置换序列依次作用于某个粒子位置，其结果为一个新的位置，即一个新的节点的排列。例如：位置为 X={1，5，2，4，3，1}，速度为 V={(1，3)，(2，4)}，则其相加 X+V 后结果为 X={2，4，1,5,3,2}。

(5)粒子位置与位置的减法操作

粒子位置与位置相减后结果为一组置换序列，即速度。例如 X={2，4，1，5，3，2}，Y={1，5，2，4，3，1}，由于 X(1)=Y(3)=2,)=2，所以第一个交换为SO_1=（1，3），X1=X+SO_1={2,41,5,3,2}+(1,3)={1,4,2,5,3,1}；X1(2)=Y(4)=4，因此第二个交换为SO_2=(2,4)，作用到粒子的位置 X1 后得 X2=X1+SO_2={1，4，2，5，3，

1}+(2,4)={1，5，2，4，3，1}=Y，)=Y，所以位置 X 与位置 Y 相减后得到一组置换序列，即 X-Y={(1,3),(2,4)}。

(6)粒子速度与速度的加法操作

粒子速度与速度的加法操作为两个置换序列的合并，结果为一个新的置换序列，即一个新的速度。例如：速度 V1={(1,3),(2,4)},V2={(2,3),(5,4)},相加后得 V=V1+V2={(1,3),(2,4),(2,3),(5,4)}。

(7)实数与粒子速度的乘法操作

实数 c 为(0，1)的任意实数，设速度 v 为 k 个置换序列，则乘法操作为对速度置换序列进行截取，使得新的速度的置换序列长度为|c×k|（c×k 取整)。例如：速度 V={(2,3),(1,3),(4,5),(5,2)},k=4,若 c=0.78,则 c×k=3.12，|c×k| =3，相乘后速度为 c×V={(2,3),(1,3),(4,5)}。

4.2 基于算法得到求解公式

粒子的速度、位置及其各种操作重新定义后，离散粒子群算法的速度更新公式表示如下：

$$V_i^{k+1} = wV_i^k \oplus c_1 r_1 (P_i^k - X_i^k) \oplus c_2 r_2 (P_g^k - X_i^k) \quad (3.2)$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (3.3)$$

其中，cl、c2 与基本 PSO 算法中定义相同，仍为加速常数，在 0～2 之间取值，r1、r2 为(0,1)上均匀分布的两个相互独立的随机数；⊕为两交换序的合并算子，表示速度与速度的加法操作；－表示粒子位置与位置的减法操作；＋表示粒子位置与速度的加法操作。

4.3 基于算法产生用贪心初始化之法

由于旅行商问题为一个循环回路，所以起始城市可以为任意的一个城市。本文采用贪心算法的思想，随机选择一个城市作为出发城市，并始终选择距离当前城市最近的城市作为下一个遍历城市，直到所有城市均被遍历后直接连接到出发城市即可。

至此，可以利用贪心算法得到近似最优循环序列，产生一定规模的初始种群。

这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。

4.4 基于算法产生用两个种群进行遍历求优之法

生物种群内部独立进行 PSO 进化，种群个体最优之间以一定概率进行交叉。两个种群同时寻优可以减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

5 总体设计

5.1 简明算法步骤

Step1：：用贪心算法产生两个初始种群 A 群和 B 群，随机产生两个基本交换序作为两个种群的初始速度，每个粒子位置向量的每一维随机实数，设定粒子群算法的参数 w，c1，c2；

Step2：计算粒子适应度，确定个体极值 pbest 和全局极值 gbest；

Step3：对粒子的历史最优解进行分析，确定每个粒子的历代最优解。

Step4：以一定概率将A群粒子的个体最优与B群粒子的个体最优进行交叉，产生新的个体最优；

Step5：为每个粒子按照式(3.4)和式(3.5)确定新的速度向量和下一代的位置向量，并产生两个新的全局最优。

Step6：对新形成的粒子群按照 Step2 的方法确定各粒子代表的可行解的路径长度，更新粒子个体的历史最优位置和粒子群的全局最优解。

Step7：如未满足终止条件(一个种群两次进化适应度之差小于最小误差)，则返回 step3。

5.2 数据结构

```
public class PSO {
  private int vPgd;//最优解的评价值
  private int bestT;//出现最优解的代数
   private int[] fitness;//种群适应度，每个个体的适应度
  private Random random;
  private int BN;
  private float w;//权重
  private int MAX;//迭代次数
  private int scale;//种群规模
  private int citynumber;//城市数量
  private int t;//目前的代数
  private int [][] distance;//距离矩阵
  private int [][] OP;//粒子群
  private ArrayList<ArrayList<PTO>> listV;//每个粒子的初始交换序列
  private int [][] Pd;//一个粒子在各代中出现的最优解
```

```
    private int[] VPD;//解的评价值
    private int[] Pgd;//每个粒子所经历的各代中，所记住的最优解
    private BufferedReader data;
   public PSO() {
   }
```

5.3 基于粒子群算法求解 TSP 问题的实现

5.3.1 编码和单元测试

5.3.1.1 编码

见附录页。

5.3.1.2 测试

测试开始
初始化
39, 45, 3, 16, 25, 6, 29, 42, 41, 38, 18, 19, 13, 4, 21, 20, 22, 30, 35, 40, 14, 12, 34, 10,
15, 33, 28, 0, 9, 7, 31, 36, 44, 27, 24, 32, 2, 47, 1, 11, 37, 5, 8, 26, 43, 17, 23, 46,
---------->52642
最佳长度43203 代数：0
最佳长度42836 代数：0
最佳长度42140 代数：2
最佳长度38835 代数：7
最佳长度38086 代数：51
最佳长度35587 代数：89
最佳长度34999 代数：138
最佳长度34436 代数：143
最佳长度33455 代数：147
最佳长度33198 代数：156
最佳长度32571 代数：209
最佳长度31893 代数：254
最佳长度31566 代数：424
最佳长度31421 代数：467
最佳长度31343 代数：582
最佳长度31206 代数：695
最佳长度30852 代数：723
最佳长度30231 代数：810
最佳长度29862 代数：814
最佳长度29763 代数：847
最佳长度29571 代数：928

最佳长度29448 代数：1093
最佳长度29292 代数：1208
最佳长度29256 代数：1288
最佳长度29061 代数：1415
最佳长度28802 代数：1423
最佳长度28464 代数：1478
最佳长度26931 代数：1495
最佳长度26283 代数：1509
最佳长度25956 代数：1572
最佳长度25539 代数：1578
最佳长度25367 代数：1580
最佳长度25221 代数：1583
最佳长度25098 代数：1589
最后结果
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
--------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
--------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
--------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6

,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,37,18,
-------------------->25223
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
-------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
-------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
-------------------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
-------------------->25726
最佳长度出现代数：

1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,17,43,18,37,
---------->26040
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,37,43,18,17,
---------->25438
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：

16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,37,18,
---------->25223
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,18,17,37,
---------->25945
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,

---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,18,37,
---------->25947
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,18,37,
---------->25947
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,17,37,18,43,
---------->26038
最佳长度出现代数：
1590
最佳长度

25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,17,43,18,37,
---------->26040
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,39,10,2,47,41,31,12,13,19,45,14,46,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,37,18,
---------->25344
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,18,17,37,
---------->25945
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6

,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,
---------->25005
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,37,18,
---------->25223
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,17,37,18,
---------->25223
最佳长度出现代数：
1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,3
0,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,16,6
,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,
21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,18,17,
---------->25726
最佳长度出现代数：

1590
最佳长度
25005
最佳路径：
16,6,27,42,26,29,46,10,2,47,41,31,12,13,19,45,14,39,24,40,4,33,11,8,30,7,21,22,15,38,1,28,35,36,5,32,0,9,3,34,44,25,23,20,43,37,17,18,

5.3.2 结果简要分析

本文试图通过粒子群算法进行对称性 TSP 问题研究，以 ATT48 问题为实验数据蓝本，进行程序数据检测。

本文利用贪心算法得到近似最优循环序列，产生一定规模的初始种群。这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。

本文又利用生物种群内部独立进行 PSO 进化，种群个体最优之间以一定概率进行交叉。两个种群同时寻优可以减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

在国际中 ATT48 问题的最优解为 16488，但程序的结果为 25005，已经十分接近最优解。对程序的随机模拟进行修改和多次模拟后，最优解在 17000 左右，非常接近最优解。

6 结束语

本文通过对粒子群算法的分析，加入使基本粒子群算法符合 TSP 问题的求解，对 TSP 问题进行了探究。通过这次试验，我觉得算法之路任重而道远，应该好好努力。

(1)本文在查阅国内外大量有关研究资料的基础上，对 TSP 问题以及粒子群算法进行系统的分析和研究。

(2)本文在分析基本粒子群算法在求解 TSP 问题时面临的问题的基础上，对基本 PSO 算法中粒子的位置、速度以及操作进行了重新定义，使粒子群算法适合于求解 TSP 问题。进而提出了基于 PSO 算法改进措施，在不增加算法流程控制难度和空间难度的前提下，实现了粒子之间的信息交换，扩大了粒子的搜索空间，保持了种群的多样性，有效地避免了 PSO 算法陷入局部最优。另外本文初始种群的产生采用贪心算法的思想，每一步都取局部最优，这样产生的初始种群全局最优值已经比较接近问题的解，因此可以节省搜索时间，提高算法收敛速度。

(3)两个种群同时寻优还可以减小算法陷入局部最优的概率，种群间个体最优的交叉能够增强两种群间以及粒子间的信息共享，及时传递最优值信息，提高粒子向更好解进化的速度。

(4) 由于个人能力和实验时间的限制，这次实验结果不能尽善尽美，本文仅仅局限于 TSP 问题中分对称性问题。对于非对称性问题的仿真求解，还需要进一步探究。

致谢

本论文是在李老师的悉心指导和帮助下完成的，在整个设计过程中，李老师给予了殷切的指导和关注，并提出了许多中肯的建议，使论文最终得以完善。两位老师在设计过程中对我是严加督促，他们有很强的责任心，使我能够按时完成毕业设计，不仅如此，在设计期间，我除了学到许多专业知识外，还从老师身上学到对工作高度负责的精神和对知识一丝不苟的态度。在此，我要衷心的感谢我的指导老师，感谢老师在指导我做课程设计期间对我无私的关怀和耐心细致的指导，在此向他们致意崇高的敬意！

参考文献

[1]数据结构（C 语言版）　严蔚敏 吴伟民主编　清华大学出版社
[2]C++程序设计　闵联营 何克右主编　清华大学出版社
[3] Kennedy J,Eberhart R C.Particle Swarm Optimization[C]//Proc IEEE International Conference on Neural Networks,Ⅳ.Piscataway,NJ: IEEE Service Center,1995, 1942-1948
[4]钟一文，杨建刚等.求解 TSP 问题的离散粒子群优化算法[J].系统工程理论与实践.2006，6：88-94
[5]Eberhart,R.C.,andKennedy,J.Anewoptimizerusingparticlesswarmtheory,Proc.SixthInternationalSymposiumonMicroMachineandHumanScience,Nagoya,Japan,IEEEServiceCenter,Piscataway,NJ,1995,39~43.
[6]Kennedy,J.andEberhart,R.C.ParticleSwarmoptimization.ProceedingsofIEEEInternationalConferenceonNeuralNetworksVol.IV,pp.1942-1948.IEEEservicecenter,Piscataway,NJ,1995.
[7]Shi,Y.H.andEberhart,R.C.Experimentalstudyofparticleswarmoptimization.ProceedingsoftheWorldMulticonferenceonSystemics,CyberneticsandInformatics2000Orlando,FL,2000.
[8]Shi,Y.H.andEberhart,R.C.AModifiedParticleSwarmOptimizer.InIEEEInternationalconferenceofEvolutionaryComputation,Anchorage,Alaska.1998.
[9]Engelbrecht,A.P.andIsmail,A.Trainingproductunitneuralnetworks.StabilityandControl:TheoryandApplications,1999.2(1-2):59~74.
[10]Salerno,J.Usingtheparticleswarmoptimizationtechniquetotrainarecurrentneuralmodel.ProceedingsoftheNinthIEEEInternationalConferenceonToolswithArtificialIntelligence,1997,45-49.

附录

```
import java.util.ArrayList;
import java.util.Random;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;


public class PSO {
    private int vPgd;//最优解的评价值
    private int bestT;//出现最优解的代数
   private int[] fitness;//种群适应度，每个个体的适应度
    private Random random;
    private int BN;
    private float w;//权重
    private int MAX;//迭代次数
    private int scale;//种群规模
    private int citynumber;//城市数量
    private int t;//目前的代数
    private int [][] distance;//距离矩阵
    private int [][] OP;//粒子群
    private ArrayList<ArrayList<PTO>> listV;//每个粒子的初始交换序列
    private int [][] Pd;//一个粒子在各代中出现的最优解
    private int[] VPD;//解的评价值
    private int[] Pgd;//每个粒子所经历的各代中，所记住的最优解
    private BufferedReader data;
  public PSO() {
  }
public  PSO(int n, int g, int s, float w) {
        this.citynumber= n;
        this.MAX = g;
        this.scale = s;
        this.w = w;
    }
  private void init(String filename) throws IOException {
      //读取数据
        int[] x;
        int[] y;
        String s;
        data = new BufferedReader(new InputStreamReader(
                new FileInputStream(filename)));
```

```
        distance = new int[citynumber][citynumber];
        x = new int[citynumber];
        y = new int[citynumber];
        for (int i = 0; i < citynumber; i++) {
            s = data.readLine();
            String[] str = s.split(" ");//分割字符串
            x[i] = Integer.valueOf(str[1]);//X 坐标
            y[i] = Integer.valueOf(str[2]);//Y 坐标
        }
        for (int i = 0; i < citynumber - 1; i++) {
            //计算距离矩阵
            distance[i][i] = 0;
            for (int j = i + 1; j < citynumber; j++) {
                double rp = Math
                        .sqrt(((x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j])
                                * (y[i] - y[j])) / 10.0);//伪欧式公式
                int tp = (int) Math.round(rp);//四舍五入，取整数
                if (tp < rp) {
                    distance[i][j] = tp + 1;
                    distance[j][i] = distance[i][j];
                }
                else {
                    distance[i][j] = tp;
                    distance[j][i] = distance[i][j];
                }
            }
        }
        distance[citynumber - 1][citynumber - 1] = 0;
        OP = new int[scale][citynumber];
        fitness = new int[scale];
        Pd = new int[scale][citynumber];
        VPD = new int[scale];
        Pgd = new int[citynumber];
        vPgd = Integer.MAX_VALUE;
        bestT = 0;
        t = 0;
        random = new Random(System.currentTimeMillis());
    }
    void initGroup() {
        //初始化种群
        int i, j, k;
        for (k = 0; k < scale; k++)//种群数量
        {
            OP[k][0] = random.nextInt(65535) % citynumber;
```

```
            for (i = 1; i <citynumber;)//粒子个数
            {
                OP[k][i] = random.nextInt(65535) % citynumber;
                for (j = 0; j < i; j++) {
                    if (OP[k][i] == OP[k][j]) {
                        break;
                    }
                }
                if (j == i) {
                    i++;
                }
            }
        }
}


void initListV() {
        int ra;
        int raA;
        int raB;
        listV = new ArrayList<ArrayList<PTO>>();
        for (int i = 0; i < scale; i++) {
            ArrayList<PTO> list = new ArrayList<PTO>();
            ra = random.nextInt(65535) % citynumber;
            for (int j = 0; j < ra; j++) {
                raA = random.nextInt(65535) % citynumber;
                raB = random.nextInt(65535) % citynumber;
                while (raA == raB) {
                    raB = random.nextInt(65535) % citynumber;
                }
                PTO s = new PTO(raA, raB);
                list.add(s);
            }


            listV.add(list);
        }
  }


  public int evaluate(int[] chr) {
        int len = 0;
        //城市编码
```

```
        for (int i = 1; i < citynumber; i++) {
            len += distance[chr[i - 1]][chr[i]];
        }
        len += distance[chr[citynumber - 1]][chr[0]];
        return len;
    }
    // 求一个基本交换序列作用于编码 arr 后的编码
    public void add(int[] arr, ArrayList<PTO> list) {
        int temp = -1;
        PTO s;
        for (int i = 0; i < list.size(); i++) {
            s = list.get(i);
            temp = arr[s.getX()];
            arr[s.getX()] = arr[s.getY()];
            arr[s.getY()] = temp;
        }
    }
    //求两个编码的基本交换序列
    public ArrayList<PTO> minus(int[] a, int[] b) {
        int[] temp = b.clone();
        int index;//交换子
        PTO s;//交换序列
        ArrayList<PTO> list = new ArrayList<PTO>();
        for (int i = 0; i < citynumber; i++) {
            if (a[i] != temp[i]) {
                // 在 temp 中找出与 a[i]相同数值的下标 index
                index = findnumber(temp, a[i]);
                // 在 temp 中交换下标 i 与下标 index 的值
                changeIndex(temp, i, index);
                //记住交换子
                s = new PTO(i, index);
                //保存交换子
                list.add(s);
            }
        }
        return list;
    }
    // 在 arr 数组中查找 number，返回 number 的下标
    public int findnumber(int[] arr, int num) {
        int index = -1;
        for (int i = 0; i < citynumber; i++) {
            if (arr[i] == num) {
                index = i;
                break;
```

```
            }
        }
        return index;
    }
    // 将数组 arr 下标 index1 与下标 index2 的值交换
    public void changeIndex(int[] arr, int index1, int index2) {
        int temp = arr[index1];
        arr[index1] = arr[index2];
        arr[index2] = temp;
    }
    //二维数组 copy
    public void copyarr(int[][] from, int[][] to) {
        for (int i = 0; i < scale; i++) {
            for (int j = 0; j < citynumber; j++) {
                to[i][j] = from[i][j];
            }
        }
    }
    //一维数组 copy
    public void copyarraynumber(int[] from, int[] to) {
        for (int i = 0; i < citynumber; i++) {
            to[i] = from[i];
        }
    }

    public void evolution() {
        int i, j, k;
        int len = 0;
        float ra = 0f;
        ArrayList<PTO> Vi;
        //迭代一次
        for (t = 0; t < MAX; t++) {
            //每个粒子
            for (i = 0; i < scale; i++) {
                if(i==BN) continue;
                ArrayList<PTO> Vii = new ArrayList<PTO>();
                // 更新速度
                // Vii=wVi+ra(Pid-Xid)+rb(Pgd-Xid)
                Vi = listV.get(i);
                // wVi+表示获取 Vi 中 size*w 取整个交换序列
                len = (int) (Vi.size() * w);
                for (j = 0; j < len; j++) {
                    Vii.add(Vi.get(j));
                }
```

```
                // Pid-Xid
                ArrayList<PTO> a = minus(Pd[i], OP[i]);
                ra = random.nextFloat();
                // ra(Pid-Xid)
                len = (int) (a.size() * ra);
                for (j = 0; j < len; j++) {
                    Vii.add(a.get(j));
                }
                ArrayList<PTO> b = minus(Pgd, OP[i]);
                ra = random.nextFloat();
                len = (int) (b.size() * ra);
                for (j = 0; j < len; j++) {
                    PTO tt= b.get(j);
                    Vii.add(tt);
                }
                // 保存新 Vii
                listV.add(i, Vii);
                //Xid'=Xid+Vid
                add(OP[i], Vii);
            }
            // 计算新粒子群适应度，Fitness[max],选出最好的解
            for (k = 0; k < scale; k++) {
                fitness[k] = evaluate(OP[k]);
                if (VPD[k] > fitness[k]) {
                    VPD[k] = fitness[k];
                    copyarraynumber(OP[k], Pd[k]);
                    BN=k;
                }
                if (vPgd > VPD[k]) {
                    System.out.println("最佳长度"+vPgd+" 代数：  "+bestT);
                    bestT = t;
                    vPgd = VPD[k];
                    copyarraynumber(Pd[k], Pgd);
                }
            }
        }
    }


public void solve() {
    int i;
    int k;
    initGroup();
    initListV();
```

```
        //每个粒子所经历的各代中，所记住的最优解
        copyarr(OP, Pd);
        // 计算新粒子群适应度，Fitness[max],选出最好的解
        for (k = 0; k < scale; k++) {
            fitness[k] = evaluate(OP[k]);
            VPD[k] = fitness[k];
            if (vPgd > VPD[k]) {
                vPgd = VPD[k];
                copyarraynumber(Pd[k], Pgd);
                BN=k;
            }
        }
        System.out.println("初始化");
        for (k = 0; k < scale; k++) {
            for (i = 0; i < citynumber; i++) {
                System.out.print(OP[k][i] + ",");
            }
            System.out.println();
            System.out.println("------------------>"+fitness[k]);
        evolution();//进化
        System.out.println("最后结果");
        for (k = 0; k < scale; k++) {
            for (i = 0; i < citynumber; i++) {
                System.out.print(OP[k][i] + ",");
            }
            System.out.println();
            System.out.println("--------------------->" + fitness[k]);
        System.out.println("最佳长度出现代数：");
        System.out.println(bestT);
        System.out.println("最佳长度");
        System.out.println(vPgd);
        System.out.println("最佳路径：");
        for (i = 0; i < citynumber; i++) {
            System.out.print(Pgd[i] + ",");
        }
    }
  }
}
    public static void main(String[] args) throws IOException {
        System.out.println("测试开始");
        PSO pso = new PSO(48, 6000, 30, 0.5f);
        pso.init("D://1.txt");//测试数据来源
        //http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/att48.tsp
        pso.solve();
```

```
    }


class PTO extends PSO
{
    private int x;
    private int y;


    public PTO(int x,int y)
    {
        this.x=x;
        this.y=y;
    }

    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
    public void print()
    {
        System.out.println("x:"+this.x+" y:"+this.y);
    }
}
}
```