**学 士 学 位 论 文**

摘 要

旅行商问题(Travelling Salesman Problem, TSP)是数学领域中著名问题之一。问题的描述为：在一个阶带权完全图中，求一条哈密顿(Hamiltonian)回路，使的该回路经过的所有边权和最小。

TSP问题在通讯、交通、军事等诸多领域均有着广泛的应用价值，然而它却是一个NP完全问题，人们无法给出一个有效的算法以在多项式时间内对其求解。在规模稍大的TSP模型下，获取最优解几乎是不可能做到的。因此，寻找一种能快速获取较优解的算法具有重要的科研意义。

蚁群算法(Ant Colony Optimization, ACO),是由Marco Dorigo于1991年在他的博士论文[1]中提出的一种仿生算法。该算法通过模拟蚂蚁觅食过程中利用信息素的释放相互合作，选择路径的行为，在求解TSP问题中取得了十分卓越的成果。

在基本蚁群算法中，针对给定的TSP问题，需要对蚁群设置包括：信息素强度、信息素挥发系数、信息启发式因子、期望启发式因子、蚁群规模等参数。参数的选取与算法的全局收敛性和收敛速度有着密切的关联。但由于蚁群算法参数空间的庞大性和各参数之间的关联性，如何确定最优组合参数使蚁群算法求解性能最佳一直是一个极其复杂的优化问题，目前上没有完善的理论依据，大多情况下都是根据经验而定[2]。

蚁群算法参数的选取，可以看成一个多目标组合优化问题。遗传算法(Genetic Algorithm, GA)对于这类问题的求解卓有成效。遗传算法模拟自然界中生物进化的过程，以生物个体的适应度作为标准，对生物群体分别作用于选择、交叉、变异算子，得到下一代种群。随着遗传代数的增加，群体中的个体逐渐接近最优解。

本文的工作是利用遗传算法，尝试对基本蚁群算法的参数最优组合进行估算。通过大量仿真计算，可以得到如下结果 // …

关键字： TSP问题、蚁群算法、遗传算法、多目标组合优化、参数估计

Abstract

目 录

1. 绪论

1.1.

1. 蚁群算法的基本原理与编程实现

2.1. 蚁群算法的基本原理

2.1.1 TSP问题概述

TSP问题（Travelling Salesman Problem，又称作旅行商问题、货郎担问题），是数学领域中的一个著名问题。该问题的描述为：一个旅行商欲不重复无遗漏的路过N个城市并最终回到起点，怎样设计路线使得其行程达到最短。即：在一个节点数为N的无向完全图中，如何构造一条最短的哈密顿回路。

对于个城市的TSP问题，所有可能的路径选择有种。当较大时，这将是一个天文数字！TSP问题已被证明是一个NP问题，也即是说我们无法找到一个有效的算法，能够在多项式的时间复杂度以内找到问题的精确解。因此，对于大规模的TSP问题，人们往往采用近似算法以得到问题的较优解。

2.1.2. 蚁群算法简介

蚁群算法（Ant Colony Optimization, ACO）是一种模拟进化算法，由Marco Dorigo于1992年在他的博士论文中提出。该算法模拟自然界中蚂蚁觅食寻找路线时表现出的协作行为，广泛应用于各类优化问题的求解。而该算法最常见的应用场景便是对TSP问题进行求解，本文对蚁群算法的若干讨论也将基于TSP问题的求解之上。

2.1.3 蚁群算法的数学模型

蚂蚁通过信息素的释放，与其他的个体之间产生协作。距离食物较近的路线具有较浓的信息素，而信息素浓度较高的路线又具有较大的概率被其它蚂蚁所选择，在这样一种正反馈作用下，蚁群将逐渐聚集到最优的路径上。这就是蚁群算法的基本思想。

这里给出基本蚁群算法的Ant-Cycle数学模型：

设为TSP问题中的城市数量，为蚁群中的蚂蚁数量，为

时刻路径上的信息素含量，为路径的长度，为所有节点的集合，为第只蚂蚁已经经过的节点集合，为第只蚂蚁在时刻从节点转移到节点的概率



上式中，为信息启发因子，表示在蚂蚁选择路径时，信息素的重要性，该值越大，表明蚂蚁越重视其它蚂蚁释放的信息素提供的信息。为期望启发因子，表示在蚂蚁选择路径时，路程长短的重要性，该值越大，蚂蚁将趋于选择较短路径前进。

在Ant-Cycle模型中，蚂蚁将在遍历所有城市一遍之后更新信息素，记为第只蚂蚁遍历所有城市后，在路径上所释放的信息素量



则在所有蚂蚁遍历完城市一遍后，所有路径上的信息素总量将更新为



上式中，为信息素挥发系数。

2.2 蚁群算法的编程实现

2.2.1 蚁群算法的算法描述

Ant-Cycle蚁群算法的实现步骤（以求解TSP问题为例）：

1.初始化参数，令循环次数loopCount = 0,设置最大循环次数MAX\_LOOP\_COUNT，初始化信息素矩阵pheromoneMatrix,其中pheromoneMatrix[i][j] = const，const为常量，初始化禁忌表tabus，tabus[i][j]表明第i只蚂蚁是否进过节点j，进入步骤2；

2.若loopCount > MAX\_LOOP\_COUNT,进入步骤8，否则进入步骤3；

3.令蚂蚁索引号为i；

4.蚂蚁按照概率转移公式移动到下一个城市，更新禁忌表tabus[i]，若尚有城市未被遍历，重复步骤4，否则进入步骤5；

5.蚂蚁索引号i = i + 1,若i < m，进入步骤4，否则进入步骤6；

6.更新所有路径上的信息素，进入步骤7；

7.循环次数loopCount = loopCount + 1，进入步骤2；

8.算法结束。

2.2.2 蚁群算法的核心代码

*/\*\*  
 \* 求解给定的 TSP 问题  
 \*  
 \** ***@param tsp*** *给定的tsp问题  
 \** ***@return*** *求得的解  
 \* \*/***public** TSPSolution solveTSP(**final** TSP tsp) {  
 TSPSolution solution = **new** TSPSolution();  
 *// 初始化信息素矩阵* **double**[][] pheromoneMatrix = **new double**[tsp.**n**][tsp.**n**];  
 **for** (**int** i = 0; i < tsp.**n**; i++) {  
 **for** (**int** j = 0; j < tsp.**n**; j++) {  
 pheromoneMatrix[i][j] = **q** / tsp.**averageDistance** \* tsp.**n**;  
 }  
 }  
 *// 开始迭代求解* **for** (**int** loopCount = 0; loopCount < ***MAX\_LOOP\_COUNT***; loopCount++) {  
 *// 信息素增量矩阵* **double**[][] pheromoneDeltaMatrix = **new double**[tsp.**n**][tsp.**n**];  
 *// 记录蚂蚁路径* **int**[][] paths = **new int**[**m**][tsp.**n**];  
 *// 记录蚂蚁禁忌表* **boolean**[][] tabus = **new boolean**[**m**][tsp.**n**];  
 *// 初始化路径记录与禁忌表* **for** (**int** i = 0; i < **m**; i++) {  
 **for** (**int** j = 0; j < tsp.**n**; j++) {  
 paths[i][j] = -1;  
 tabus[i][j] = **false**;  
 }  
 }  
 *// 记录所有蚂蚁的最短行程* **double** currentMinDistance = Double.***MAX\_VALUE***;  
 **int**[] currentMinPath = **new int**[tsp.**n**];  
 *// 对所有蚂蚁进行遍历操作* **for** (**int** i = 0; i < **m**; i++) {  
 *// 随机初始化蚂蚁位置* **int** position = 0;  
 *// 更新路径* paths[i][0] = position;  
 *// 更新禁忌表* tabus[i][position] = **true**;  
 *// 遍历城市* **for** (**int** j = 1; j < tsp.**n**; j++) {  
 *// 当前所在位置* **int** currentPosition = paths[i][j - 1];  
 *// 计算权重* **double**[] weightArray = **new double**[tsp.**n**];  
 **for** (**int** k = 0; k < weightArray.**length**; k++) {  
 **if** (!tabus[i][k]) {  
 weightArray[k] = Math.*pow*(pheromoneMatrix[currentPosition][k], **alpha**) /  
 Math.*pow*(tsp.**distanceMatrix**[currentPosition][k], **beta**);  
 }  
 **else** {  
 weightArray[k] = 0;  
 }  
 }  
 *// 选取下一个城市* **int** nextPosition = Utils.*roulette*(weightArray, tabus[i]);  
 *// 更新路径* paths[i][j] = nextPosition;  
 *// 更新禁忌表* tabus[i][nextPosition] = **true**;  
 }  
 *// 计算总路程* **double** distance = tsp.calcDistance(paths[i]);  
 *// 释放信息素* **double** pheromoneDelta = **q** / distance;  
 **for** (**int** j = 0; j < tsp.**n** - 1; j++) {  
 pheromoneDeltaMatrix[paths[i][j]][paths[i][j + 1]] += pheromoneDelta;  
 }  
 pheromoneDeltaMatrix[paths[i][tsp.**n** - 1]][paths[i][0]] += pheromoneDelta;  
 *// 若该蚂蚁找到较短的路径，则更新路程信息* **if** (distance < currentMinDistance) {  
 currentMinDistance = distance;  
 currentMinPath = paths[i];  
 }  
 }  
 *// 更新信息素* **for** (**int** i = 0; i < tsp.**n**; i++) {  
 **for** (**int** j = 0; j < tsp.**n**; j++) {  
 *// 信息素挥发* pheromoneMatrix[i][j] \*= 1 - **lambda**;  
 *// 信息素增加* pheromoneMatrix[i][j] += pheromoneDeltaMatrix[i][j];  
 }  
 }  
 *// 记录本次迭代路程信息* solution.addPath(currentMinDistance, currentMinPath);  
 }  
 **return** solution;  
}

1. 遗传算法的基本原理与编程实现

3.1 遗传算法的基本原理

3.1.1 遗传算法简介

遗传算法（Genetic Algorithm）是一种模拟自然界中种群生物进化的优化算法，常被应用与多目标参数优化问题。遗传算法在问题的潜在解集中选取部分个体并视作一个种群（Population），将个体的差异性特征（即问题的参数）进行适当的编码作为基因（Gene），通过定义合适的适应度函数，对种群中的个体进行自然选择。然后对于选择算子作用后的种群进行个体的交叉与变异，最终得到下一代种群。经过一定代数的进化，种群中的个体的平均适应度将明显优于初代种群。

3.1.2 遗传算法的数学模型

由编码方式的不同，遗传算法通常被分为二进制编码遗传算法与浮点数编码遗传算法。在本文中，将采用二进制编码的遗传算法进行相关讨论。

遗传算法每一次进化过程可以抽象为三个作用于种群的算子：

1.选择算子：

选择运算是遗传算法中最为关键的一个步骤，其中，对于适应度函数的定义与算法的全局收敛性与收敛速度紧密相关。适应度函数是一个个体到非负实数之间的映射关系，体现了个体对于环境的适应程度，适应度越高的个体被选择到下一代的概率也就越大（通常采用轮盘赌方式进行选择）。

2.交叉算子：

对于被选择出的个体，进行两两随机配对，配对后的个体之间进行染色体交叉互换，得到新的个体。

3.变异算子：

为了防止算法陷于局部最优解中，经过交叉算子作用后的个体将以一定的概率产生基因突变，突变后得到的种群将作为新一代的种群。

迭代上述进化过程，种群的基因将趋于最优。

这里给出二进制编码遗传算法的数学模型：

设所求问题为非负多元函数在可行域上的最大值，其中为所求参数。

二进制编码遗传算法中，选取个可行的解作为初始种群集合，将他们的决策变量编码为长度为的二进制数。对种群分别作用以选择算子，交叉算子，变异算子，得到下一代种群



选择算子：根据种群中个体的适应度，对种群进行次选择，每次各个体被选择到下一代的概率。经过选择后，得到

交叉算子：在种群中进行随机两两配对，对每对个体进行基因重组，即



其中，交叉位置为随机选取。得到

变异算子：对交叉算子作用后的，其中各个体以一定概率按位产生突变。得到。

3.2 遗传算法的编程实现

3.2.1 遗传算法的算法描述

1. 基于遗传算法对蚁群算法进行参数优化

4.1 蚁群算法存在的缺陷

在蚁群算法中，需要包括蚂蚁数量、信息素强度、信息素挥发系数、信息启发因子、期望启发因子等众多初始参数。这些参数的选取对于蚁群算法的全局收敛性和求解效率都有着显著的影响。如何确定这些参数的最佳组合是一个极其复杂的优化问题。通常而言，蚁群算法的参数选取往往凭借经验以及反复试凑得到。这显然是蚁群算法的一个重要的缺陷。

本文基于遗传算法，对蚁群算法的参数进行优化。试图为蚁群算法的参数提供一个普适的、科学的选取方法。

4.2 蚁群的适应度函数

4.2.1 蚁群算法的性能评价指标

要采用遗传算法对蚁群算法进行参数优化，合理定义蚁群的适应度函数是整个算法的核心。

评价一个蚁群求解TSP问题性能的好坏，我们给出以下三个指标：

1.最佳性能指标

定义基本蚁群算法的相对误差作为最佳性能指标



其中为蚁群多次运算后得到的最优解，表示所求TSP问题的理论最优解（若理论最优解未知，可用当前已知最优解替代）。该指标代表了蚁群算法对问题的优化性能，其值越小，表明该蚁群的优化性能越好。

2.时间性能指标

定义基本蚁群算法的时间性能指标如下：



其中，为本次计算首次搜索到最优解时的迭代次数，为给定的最大迭代次数，为迭代一次所需的平均时间。时间性能指标用来衡量蚁群算法的收敛速度，其值越小，表明算法的收敛速度越快。

3.鲁棒性能指标

定义基本蚁群算法的鲁棒性能指标如下：



其中为蚁群多次运算后得到的解的平均值，表示所求TSP问题的理论最优解（若理论最优解未知，可用当前已知最优解替代）。该指标代表了蚁群算法对算法随机性的依赖程度。

综上所述，我们可以定义蚁群算法的综合性能指标如下：



其中、和分别为最佳性能指标、时间性能指标和鲁棒性能指标的权重，满足。综合性能指标值越小，表明蚁群算法的性能越好。

4.2.2 蚁群算法的适应度函数

定义蚁群的适应度函数



其中，令算法平均一次迭代时间（蚁群规模与城市数量之比），权重参数。

4.3 蚁群基因的二进制编码与解

4.3.1 格雷编码的引入

遗传算法的部搜索能力不强，导致这个问题的主要原因是新一代的种群是有上一代种群经过交叉、变异后得到的。即便上一代种群个体已经接近了最优解，经过变异算子作用后，参数值可能会产生较大的变化，导致种群无法继续向最优解方向收敛。

格雷编码是广泛应用于通信、模拟-数字信号转换等领域的一种编码方式。格雷码具有这样一个特性：任意两个相邻整数对应的格雷码之间汉明距离为1。正是由于这个特性的存在，使得格雷码十分适用于遗传算法。编码过后的基因经过突变后，其对应的原始数据只会产生细微的变化。这样可以大大提高遗传算法的局部搜索能力。

对于给定的自然二进制编码，其与对应的格雷码之间由以下公式进行转换。

自然二进制码至格雷码：



格雷码至自然二进制码：



4.3.2 编码与解码

本文对蚁群算法的参数采用以下二进制编码、解码方式：

1.编码

设为待编码实数，区间为的取值范围，为指定的编码长度。则将编码为，其中，函数为格雷编码函数。

2.解码

设为待解码数，区间为的取值范围，为指定的编码长度。则将编码为，其中，函数为格雷解码函数。

4.4 基于遗传算法的参数优化结果

1. 结论