# 摘 要

0-1背包问题是一个著名的优化问题，该问题已被证明是一个NP难题。因此，无法找到一个有效的算法使得人们可以在非多项式时间内得到问题的最优解。

蚁群算法是一种被广泛应用与求解优化问题中的算法。该算法通过模拟蚁群觅食时释放信息素的自然现象，建立了一种具有正反馈作用的寻路机制。通过该算法能够有效快速的求解各种优化问题的局部最优解。

本文通过分析蚁群算法的基本原理，针对0-1背包问题作出合理的改进，基于KPACA算法对0-1背包的求解问题作出了讨论，给出该算法的编程实现与核心代码。并在两例仿真实例中均取得了较好的运算结果。

关键字：0-1背包问题 优化问题 蚁群算法 KPACA

# Abstract

0-1 knapsack problem is a well-known optimization problem, this problem has been proved to be a NP problem. Therefore, people can not to find an effective algorithm in polynomial time to obtain the optimal solution of problem.

Ant colony algorithm is a kind of widely application and algorithm of solving optimization problems. The algorithm through the simulation of the pheromone released when ant colony foraging natural phenomena, with positive feedback effect is established a pathfinding mechanism. By the algorithm can efficiently solve a variety of local optimal solution of optimization problem.

In this paper, by analyzing the basic principle of ant colony algorithm, make reasonable improvement for 0-1 knapsack problem, based on KPACA algorithm to solve 0-1 knapsack problems made discussion, programming and the core code of the algorithm is presented. And in the two cases of simulation experiments are good experimental results were obtained.

Key word: 0-1 knapsack problem, optimization problem,

Ant colony algorithm, KPACA

目 录

[摘 要 I](#_Toc420919740)

[Abstract II](#_Toc420919741)

[1 绪论 2](#_Toc420919742)

[2 0-1背包问题的概述 2](#_Toc420919743)

[2.1 0-1背包问题的数学描述 2](#_Toc420919744)

[2.2 0-1背包问题的传统求解方法 2](#_Toc420919745)

[2.2.1 递归法 2](#_Toc420919746)

[2.2.2 动态规划法 2](#_Toc420919747)

[2.2.3 分支定界法 2](#_Toc420919748)

[3 蚁群算法的基本原理 2](#_Toc420919749)

[3.1 蚁群算法的简介 2](#_Toc420919750)

[3.2 基本蚁群算法的数学模型 2](#_Toc420919751)

[4 蚁群算法求解0-1背包问题 2](#_Toc420919752)

[4.1 KPACA蚁群算法的数学描述 2](#_Toc420919753)

[4.2 KPACA蚁群算法的编程实现 2](#_Toc420919754)

[4.3 仿真实例 2](#_Toc420919755)

[4.3.1 实例一 2](#_Toc420919756)

[4.3.2 实例二 2](#_Toc420919757)

[5 结论 2](#_Toc420919758)

[6 存在的不足 2](#_Toc420919759)

[7 致谢 2](#_Toc420919760)

[8 参考文献 2](#_Toc420919761)

[附录 参考外文及翻译 2](#_Toc420919762)

# 1 绪论

蚁群算法是一种基于模拟蚂蚁觅食行为的仿生学优化算法。该算法常常被用来解决路径优化等相关问题（如TSP问题、0-1背包问题）。本文将讨论蚁群算法在0-1背包问题中的应用，并给出相关的算法与结论。

# 2 0-1背包问题的概述

## 2.1 0-1背包问题的数学描述

背包问题(Knapsack problem)是由Merkel和Hellman于1978年提出的一种组合优化问题。该问题表述为：给定一组物品，每种物品的重量与价格已知。在不超过给定的总重量的前提下，如何选取物品的组合，使他们的总价值达到最大。

这里给出其数学描述：

给定种物品，、分别为第件物品的重量与价值。现有一个总容量为的背包。要求一个维0-1向量，使得满足的条件下，使达到最大。

即求解规划问题



## 2.2 0-1背包问题的传统求解方法

0-1背包问题已被证明为是一种NP完全问题。即无法找到一个有效的算法，能在多项式的时间复杂度内求出该问题的精确解。该问题有很多常用的优化算法，这里列举以下几种：

### 2.2.1 递归法

递归法是解决0-1背包问题的最常用的方法之一。该算法的描述如下：

设表示将前件物品放入容量为的背包中所能得到的最大价值。则有

|  |  |  |
| --- | --- | --- |
|  |  | (2-1) |

对上式的含义可以做出这样的解释：考虑第件物品放与不放的情形，若第件物品不被放入背包之中，则显然有。若第件物品放入背包中，则需腾出的空间来存放前件物品，故此时有，考虑到上述两种情况，便可得到式2-1。

递归法求解0-1背包问题具有易于编程，容易理解，能取得全局最优解等优点。但当问题的规模较大时，递归算法往往会引起栈溢出等问题，需要对进行合理的优化以降低算法的空间复杂度。

尽管通过对算法的优化能降低其空间复杂度，但时间复杂度无法降低。从本质上讲，递归算法也属于穷举算法的一种，并不适用于求解大规模的背包问题。

### 2.2.2 动态规划法

动态规划是求解决策过程最优化的一种常用数学方法。该方法将多阶段的求解过程转化为一系列单阶段的问题，并通过各阶段之间的关系，依次求解。

相对于常见的静态规划模型，动态规划法具有可以找到全局最优解的优点，并且在计算中可以利用实际知识和经验来提高模型的求解效率。但该方法求解规划问题，难以找到一个统一的模型，只能具体问题具体分析。并且通过数值方式求解时，有可能存在维数灾。

### 2.2.3 分支定界法

分支定界法是一种求解整数规划问题的常用算法。该算法的主要步骤如下：

1. 放宽或取消原规划问题的限制条件，如所求变量必须取整数等条件。若此时求得的解为可行解，则该解为问题的最优解。否则该解为此问题的最优解上界。

2. 将被放宽条件的问题分解成若干个子问题，子问题解的集合的并集为原问题的可行解。对每个子问题进行求解。这些子问题的解中最优者若是原问题的可行解，则该解为原问题的最优解。否则，该解为原问题的一个新的最优解上界。另外，子问题的解中若存在原问题的可行解，则将这些解中的最优解作为原问题最优解的下界。

3. 若最优解的目标函数值已经低于最优解下界的子问题，可以肯定其中已不包含原问题的最优解，可以不作考虑。其余的子问题保留。

4. 在剩余的子问题中，选取具有最优的最优解的子问题。重复步骤1、2。指导求出原问题的最优解。

分支定界法的优点是可以求得问题的最优解，并且相对普通的穷举算法具有较快的求解速度，主要缺点是需要消耗大量的内存空间存储各分支的限界等数据。

对于0-1背包问题，还有许多其他的算法如贪婪算法、图论法等等。限于篇幅，本文不对这些方法一一进行讨论。

由于背包问题的NP性质，求得大规模的背包问题的精确解几乎是无法实现的。因此寻找较优解的算法啊具有深刻的研究意义。本文将基于蚁群算法，讨论其求解背包问题的改进算法——KPACA算法。通过该算法的原理进行深入分析，并编程实现。最后给出两例仿真实例验证该算法的有效性。

# 3 蚁群算法的基本原理

## 3.1 蚁群算法的简介

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。

自然界中蚂蚁在寻找食物时，总能聚集到从蚁穴到食物之间最短的路径之上。这是因为蚂蚁在寻找路径的同时释放了一种信息素。蚂蚁经历的路径越长，被释放的信息素浓度将越低。如此，较短的路径上将聚集浓度较高的信息素。而其它蚂蚁在选择路径时，信息素浓度较高的路径具有较大的概率被选择。在这样一种正反馈的机制下，蚂蚁将趋于集中到最短的路径上。这就是蚁群算法的核心思想。

蚁群算法广泛应用于各类优化问题的求解。对于0-1背包问题，蚁群算法也具有较好的求解性能。本文将做出基于蚁群算法求解0-1背包问题的相关讨论。

## 3.2 基本蚁群算法的数学模型

Marco Dorigo在他的博士论文中提出了三种基本蚁群算法的模型，分别为And-Cycle模型、Ant-Quantity模型以及Ant-Density模型。这三者的区别在于模型中蚂蚁释放信息素的时机不同。

这里给出Ant-Cycle蚁群算法的数学模型（以求解TSP问题为例）：

设蚂蚁需搜寻的路径节点数为，蚂蚁数量为，时刻路径上的信息素含量为，路径的长度为，第只蚂蚁已经过的节点集合为，第只蚂蚁下一步可以选择的节点集合为，第只蚂蚁在时刻从节点转移到节点的概率为，则有

|  |  |  |
| --- | --- | --- |
|  |  | (3-1) |

上式中，、分别为信息启发因子、期望启发因子。信息启发因子表明了信息素的浓度对蚂蚁选择路径时的重要性，期望启发因子代表能见度（即路径长度）对蚂蚁选择路径的重要性。

蚂蚁在所有城市都被遍历一次之后，将更新所经过路径上的信息素，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

|  |  |  |
| --- | --- | --- |
|  |  | (3-2) |

所有蚂蚁遍历城市一次后，信息素总量将更新为

|  |  |  |
| --- | --- | --- |
|  |  | (3-3) |

公式2-3中，被称为信息素挥发系数，其值越大，信息素的挥发效果将越明显。

通过多次迭代上述搜索过程，由于信息素逐渐积累，蚁群将逐渐收敛至该TSP问题的某个局部最优解上。

# 4 蚁群算法求解0-1背包问题

## 4.1 KPACA蚁群算法的数学描述

前文所述的基本蚁群算法是基于求解TSP问题的。对于0-1背包问题，需要对该模型做适当的修正。这里采用KPACA算法进行求解。

下面给出KPACA算法的数学模型：

将0-1背包问题的可行解以0、1的形式存在于数组中，其中表示蚂蚁对物品的操作状态，该值取0表明未装入背包，取1表明已装入背包。

其中第件物品被蚂蚁装入背包的概率为

|  |  |  |
| --- | --- | --- |
|  |  | (4-1) |

上式中为时刻物品所携带的信息素量，为物品的启发函数，此处启发函数为物品的单位重量价值，即。为蚂蚁尚未搜索过的物品集合。

在所有蚂蚁搜索完成之后，对物品携带的信息素进行更新操作：

|  |  |  |
| --- | --- | --- |
|  |  | (4-2) |

上式中为信息素挥发系数。若物品被该代蚁群中总价值最大的蚂蚁选中，则，否则。其中为信息素强度，为该代蚁群中价值最大蚂蚁背包中物品的总价值。

## 4.2 KPACA蚁群算法的编程实现

4.2.1 KPACA蚁群算法步骤

上述的KPACA蚁群算法的实现步骤如下：

1. 初始化参数，令蚂蚁数量为m，最大迭代次数为MAX\_LOOP\_COUNT，当前代数i = 0，各物品的信息素浓度pheromone[i] = 1，进入步骤2。

2. 令蚂蚁索引号k = 0，重置该蚂蚁的禁忌表，进入步骤3。

3. 蚂蚁按照公式4-1的概率以轮盘赌规则选取一件物品，若背包尚有空间存放该物品，则将该物品放入背包，并更新禁忌表，重复执行步骤3。否则进入步骤4。

4. 蚂蚁索引号k = k + 1，若k < m，进入步骤3，否则进入步骤5。

5. 按照公式4-2更新物品携带的信息素。

6. 迭代次数i = i + 1,若i < MAX\_LOOP\_COUNT，则进入步骤2，否则进入步骤7。

7. 算法结束，得到价值最大的物品组合。

4.2.2 KPACA蚁群算法核心代码

/\*\*

\* 求解 0-1背包 问题

\* \*/

public boolean[] solveBagProblem(BagProblem bagProblem) {

// 记录信息素数组

double[] pheromoneArray = new double[bagProblem.n];

for (int i = 0; i < pheromoneArray.length; i++) {

pheromoneArray[i] = 1;

}

// 记录最优解

double maxValue = 0;

// 记录最优路径

boolean[] maxSolution = new boolean[bagProblem.n];

// 迭代

for (int i = 0; i < MAX\_LOOP\_COUNT; i++) {

// 记录本次迭代的最优解

double currentValue = 0;

double currentWeigth = 0;

// 记录本次迭代的最优路径

boolean[] solution = new boolean[bagProblem.n];

// 对所有蚂蚁进行操作

for (int j = 0; j < m; j++) {

// 计算轮盘赌权重

double[] weightArray = new double[bagProblem.n];

for (int k = 0; k < bagProblem.n; k++) {

weightArray[k] = Math.pow(pheromoneArray[k], alpha) \* Math.pow(bagProblem.vArray[k] / bagProblem.wArray[k], beta);

}

boolean[] banArray = new boolean[bagProblem.n];

boolean[] currentSolution = new boolean[bagProblem.n];

double wSum = 0;

double vSum = 0;

boolean flag = true;

while (flag) {

// 轮盘赌选择下一件物品

int index = Utils.roulette(weightArray, banArray);

wSum += bagProblem.wArray[index];

vSum += bagProblem.vArray[index];

banArray[index] = true;

currentSolution[index] = true;

// 无法被选择的物品加入禁忌表

flag = false;

for (int k = 0; k < bagProblem.n; k++) {

if (!banArray[k]) {

if (bagProblem.wArray[k] + wSum > bagProblem.capacity) {

banArray[k] = true;

}

else {

// 若尚有能选择的物品，则继续循环

flag = true;

}

}

}

}

if (vSum > currentValue) {

currentValue = vSum;

currentWeigth = wSum;

System.arraycopy(currentSolution, 0, solution, 0, bagProblem.n);

}

}

// 打印出本次循环的最优解

System.out.println(i + "\t" + currentValue + "\t" + currentWeigth);

if (maxValue < currentValue) {

maxValue = currentValue;

System.arraycopy(solution, 0, maxSolution, 0, bagProblem.n);

}

// 更新信息素

for (int j = 0; j < bagProblem.n; j++) {

pheromoneArray[j] \*= rho;

if (solution[j]) {

pheromoneArray[j] += Q \* bagProblem.vArray[j] / currentValue;

}

}

}

return maxSolution;

}

## 4.3 仿真实例

本文中仿真实例分为两部分构成：第一部分数据来源于文献[1]，第二部分数据由计算机随机生成。

### 4.3.1 实例一

下表为取自文献[1]中的背包问题数据。

表4-1 引自文献的0-1背包问题物品数据表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | w | v | v/w | 序号 | w | v | v/w |
| 1 | 54 | 71 | 1.314815 | 51 | 49 | 37 | 0.755102 |
| 2 | 29 | 58 | 2 | 52 | 48 | 16 | 0.333333 |
| 3 | 78 | 31 | 0.397436 | 53 | 63 | 26 | 0.412698 |
| 4 | 77 | 2 | 0.025974 | 54 | 16 | 55 | 3.4375 |
| 5 | 71 | 82 | 1.15493 | 55 | 66 | 94 | 1.424242 |
| 6 | 42 | 5 | 0.119048 | 56 | 40 | 51 | 1.275 |
| 7 | 80 | 87 | 1.0875 | 57 | 79 | 11 | 0.139241 |
| 8 | 97 | 38 | 0.391753 | 58 | 76 | 46 | 0.605263 |
| 9 | 6 | 88 | 14.66667 | 59 | 84 | 60 | 0.714286 |
| 10 | 37 | 95 | 2.567568 | 60 | 22 | 2 | 0.090909 |
| 11 | 77 | 53 | 0.688312 | 61 | 11 | 8 | 0.727273 |
| 12 | 60 | 6 | 0.1 | 62 | 13 | 34 | 2.615385 |
| 13 | 30 | 47 | 1.566667 | 63 | 54 | 1 | 0.018519 |
| 14 | 65 | 63 | 0.969231 | 64 | 55 | 66 | 1.2 |
| 15 | 28 | 27 | 0.964286 | 65 | 9 | 83 | 9.222222 |
| 16 | 83 | 83 | 1 | 66 | 68 | 20 | 0.294118 |
| 17 | 99 | 59 | 0.59596 | 67 | 36 | 46 | 1.277778 |
| 18 | 23 | 92 | 4 | 68 | 71 | 15 | 0.211268 |
| 19 | 99 | 70 | 0.707071 | 69 | 54 | 93 | 1.722222 |
| 20 | 54 | 25 | 0.462963 | 70 | 76 | 9 | 0.118421 |
| 21 | 100 | 11 | 0.11 | 71 | 47 | 41 | 0.87234 |
| 22 | 2 | 68 | 34 | 72 | 21 | 50 | 2.380952 |
| 23 | 11 | 58 | 5.272727 | 73 | 10 | 33 | 3.3 |
| 24 | 80 | 11 | 0.1375 | 74 | 17 | 59 | 3.470588 |
| 25 | 5 | 29 | 5.8 | 75 | 10 | 93 | 9.3 |
| 26 | 39 | 30 | 0.769231 | 76 | 28 | 45 | 1.607143 |
| 27 | 95 | 31 | 0.326316 | 77 | 76 | 88 | 1.157895 |
| 28 | 41 | 98 | 2.390244 | 78 | 68 | 28 | 0.411765 |
| 29 | 17 | 28 | 1.647059 | 79 | 9 | 26 | 2.888889 |
| 30 | 65 | 17 | 0.261538 | 80 | 33 | 4 | 0.121212 |
| 31 | 42 | 42 | 1 | 81 | 30 | 80 | 2.666667 |
| 32 | 33 | 72 | 2.181818 | 82 | 49 | 24 | 0.489796 |
| 33 | 21 | 64 | 3.047619 | 83 | 35 | 26 | 0.742857 |
| 34 | 59 | 19 | 0.322034 | 84 | 49 | 5 | 0.102041 |
| 35 | 46 | 9 | 0.195652 | 85 | 87 | 21 | 0.241379 |
| 36 | 27 | 91 | 3.37037 | 86 | 76 | 59 | 0.776316 |
| 37 | 38 | 79 | 2.078947 | 87 | 34 | 93 | 2.735294 |
| 38 | 92 | 29 | 0.315217 | 88 | 9 | 55 | 6.111111 |
| 39 | 63 | 64 | 1.015873 | 89 | 42 | 64 | 1.52381 |
| 40 | 10 | 43 | 4.3 | 90 | 12 | 97 | 8.083333 |
| 41 | 70 | 57 | 0.814286 | 91 | 63 | 93 | 1.47619 |
| 42 | 84 | 92 | 1.095238 | 92 | 15 | 35 | 2.333333 |
| 43 | 55 | 3 | 0.054545 | 93 | 22 | 48 | 2.181818 |
| 44 | 44 | 92 | 2.090909 | 94 | 14 | 100 | 7.142857 |
| 45 | 51 | 68 | 1.333333 | 95 | 35 | 3 | 0.085714 |
| 46 | 47 | 52 | 1.106383 | 96 | 93 | 55 | 0.591398 |
| 47 | 41 | 36 | 0.878049 | 97 | 41 | 54 | 1.317073 |
| 48 | 6 | 27 | 4.5 | 98 | 83 | 85 | 1.024096 |
| 49 | 98 | 25 | 0.255102 | 99 | 73 | 68 | 0.931507 |
| 50 | 40 | 7 | 0.175 | 100 | 34 | 100 | 2.941176 |

问题中背包容量。

本实例运行参数设置为与文献[1]中运行参数相同，其中信息素强度、信息启发因子、期望启发因子、蚂蚁数量、信息素挥发系数。

将上表中数据带入程序中进行计算，得到下表所示结果：

表4-2 实例一计算结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 代数 | 最大价值 | 总重量 | 代数 | 最大价值 | 总重量 |
| 1 | 4710 | 3846 | 11 | 4703 | 3838 |
| 2 | 4694 | 3845 | 12 | 4719 | 3845 |
| 3 | 4692 | 3841 | 13 | 4717 | 3851 |
| 4 | 4700 | 3848 | 14 | 4717 | 3839 |
| 5 | 4711 | 3849 | 15 | 4718 | 3832 |
| 6 | 4701 | 3849 | 16 | 4721 | 3850 |
| 7 | 4705 | 3850 | 17 | 4721 | 3850 |
| 8 | 4701 | 3850 | 18 | 4721 | 3850 |
| 9 | 4710 | 3837 | 19 | 4721 | 3850 |
| 10 | 4703 | 3847 | 20 | 4721 | 3850 |

从上表中可以看到，KPACA算法在蚁群第16次遍历城市后，收敛到局部最优解4721。该解的结果与文献[1]中采用KPACA算法得到的结果一致。说明本实例中算法取得了预期的效果。

### 4.3.2 实例二

下表为计算机随机生成的背包问题数据。其中物品重量、均为内均匀分布的随机数。背包容量取

表4-3 随机生成的0-1背包问题物品数据表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | w | v | v/w | 序号 | w | v | v/w |
| 1 | 78.4906 | 68.11731 | 0.86784 | 51 | 87.48654 | 23.10787 | 0.264131 |
| 2 | 35.01846 | 49.12721 | 1.402895 | 52 | 82.16165 | 24.13343 | 0.293731 |
| 3 | 44.88174 | 0.526983 | 0.011742 | 53 | 11.14719 | 29.37842 | 2.6355 |
| 4 | 86.55969 | 9.89589 | 0.114324 | 54 | 47.00005 | 52.78799 | 1.123147 |
| 5 | 42.54246 | 3.616222 | 0.085003 | 55 | 16.88381 | 17.30549 | 1.024975 |
| 6 | 11.94509 | 47.96534 | 4.015486 | 56 | 39.01964 | 0.057787 | 0.001481 |
| 7 | 41.76371 | 41.26958 | 0.988168 | 57 | 31.21494 | 74.38054 | 2.382851 |
| 8 | 22.19377 | 90.66521 | 4.085165 | 58 | 82.27611 | 97.265 | 1.182178 |
| 9 | 94.40884 | 5.772815 | 0.061147 | 59 | 6.079258 | 89.63936 | 14.74511 |
| 10 | 16.61729 | 62.14494 | 3.739775 | 60 | 84.17545 | 20.53034 | 0.243899 |
| 11 | 10.54493 | 9.32333 | 0.884153 | 61 | 83.64234 | 32.33833 | 0.386626 |
| 12 | 76.77023 | 30.43157 | 0.396398 | 62 | 26.62238 | 70.0178 | 2.630035 |
| 13 | 86.16236 | 25.18586 | 0.292307 | 63 | 2.331177 | 2.764544 | 1.1859 |
| 14 | 65.20129 | 47.84775 | 0.733847 | 64 | 93.5641 | 42.81126 | 0.457561 |
| 15 | 35.91597 | 11.84618 | 0.329831 | 65 | 31.72167 | 76.12988 | 2.399933 |
| 16 | 13.84682 | 88.50394 | 6.391642 | 66 | 67.52524 | 67.29706 | 0.996621 |
| 17 | 28.66951 | 7.519434 | 0.26228 | 67 | 20.87682 | 92.62836 | 4.4369 |
| 18 | 7.318853 | 79.64528 | 10.88221 | 68 | 51.26495 | 88.11816 | 1.718877 |
| 19 | 45.5971 | 39.49834 | 0.866247 | 69 | 19.83111 | 81.98996 | 4.134411 |
| 20 | 14.87484 | 69.82486 | 4.694158 | 70 | 22.34736 | 65.08172 | 2.912277 |
| 21 | 80.69368 | 53.08099 | 0.657809 | 71 | 87.13907 | 49.24281 | 0.565106 |
| 22 | 1.516583 | 76.1916 | 50.23901 | 72 | 8.20155 | 76.84358 | 9.369396 |
| 23 | 81.76856 | 28.19619 | 0.344829 | 73 | 27.75389 | 89.89981 | 3.239179 |
| 24 | 53.45075 | 5.622424 | 0.105189 | 74 | 77.15763 | 0.759366 | 0.009842 |
| 25 | 86.34416 | 18.34882 | 0.212508 | 75 | 14.97437 | 47.41442 | 3.166372 |
| 26 | 86.05384 | 49.9946 | 0.580969 | 76 | 76.98046 | 73.98344 | 0.961068 |
| 27 | 41.35466 | 16.13605 | 0.390187 | 77 | 37.79639 | 73.50452 | 1.94475 |
| 28 | 63.51929 | 14.58591 | 0.22963 | 78 | 16.58276 | 27.71314 | 1.671202 |
| 29 | 8.13669 | 22.33968 | 2.745549 | 79 | 80.7275 | 2.821834 | 0.034955 |
| 30 | 48.52487 | 50.40617 | 1.03877 | 80 | 78.8846 | 26.4175 | 0.334888 |
| 31 | 44.92723 | 71.47827 | 1.590979 | 81 | 86.97106 | 7.448634 | 0.085645 |
| 32 | 17.54352 | 26.20569 | 1.493754 | 82 | 73.19627 | 36.9557 | 0.504885 |
| 33 | 27.93118 | 45.70707 | 1.636418 | 83 | 77.85896 | 53.83207 | 0.691405 |
| 34 | 22.83079 | 93.97201 | 4.11602 | 84 | 84.03678 | 98.6821 | 1.174273 |
| 35 | 18.99401 | 91.08236 | 4.79532 | 85 | 1.509589 | 45.02063 | 29.8231 |
| 36 | 55.65766 | 72.91814 | 1.310119 | 86 | 49.96046 | 64.95104 | 1.300049 |
| 37 | 13.0123 | 35.46086 | 2.72518 | 87 | 61.91039 | 94.44586 | 1.525525 |
| 38 | 67.01706 | 1.281736 | 0.019126 | 88 | 10.74901 | 38.87606 | 3.616711 |
| 39 | 1.437834 | 95.68379 | 66.54718 | 89 | 35.97203 | 43.99677 | 1.223083 |
| 40 | 31.81813 | 82.69603 | 2.599022 | 90 | 94.23696 | 31.76454 | 0.337071 |
| 41 | 69.57851 | 53.19893 | 0.764589 | 91 | 20.57379 | 8.762739 | 0.425918 |
| 42 | 0.492039 | 84.45982 | 171.6525 | 92 | 75.07318 | 31.34017 | 0.417462 |
| 43 | 4.871903 | 71.9031 | 14.75873 | 93 | 64.21752 | 3.575154 | 0.055673 |
| 44 | 67.96852 | 28.72517 | 0.422625 | 94 | 26.88153 | 78.68211 | 2.926995 |
| 45 | 30.90512 | 29.74132 | 0.962343 | 95 | 28.63013 | 82.35659 | 2.87657 |
| 46 | 4.142321 | 54.43799 | 13.1419 | 96 | 6.568881 | 17.75108 | 2.702299 |
| 47 | 97.67706 | 36.47779 | 0.373453 | 97 | 94.50219 | 0.290374 | 0.003073 |
| 48 | 32.23413 | 17.8078 | 0.552452 | 98 | 54.9387 | 99.43896 | 1.809998 |
| 49 | 52.4471 | 14.05767 | 0.268035 | 99 | 72.52297 | 90.95221 | 1.254116 |
| 50 | 11.86034 | 42.14981 | 3.553844 | 100 | 71.30381 | 93.27006 | 1.308066 |

本实例运行参数设置为与实例一中运行参数保持一致，其中信息素强度、信息启发因子、期望启发因子、蚂蚁数量、信息素挥发系数。

将上表中数据带入程序中进行计算，得到下表所示结果：

表4-4 实例二计算结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 代数 | 最大价值 | 总重量 | 代数 | 最大价值 | 总重量 |
| 1 | 4636.155 | 3670.702 | 16 | 4648.058 | 3665.373 |
| 2 | 4637.286 | 3646.718 | 17 | 4648.058 | 3665.373 |
| 3 | 4636.906 | 3640.493 | 18 | 4648.058 | 3665.373 |
| 4 | 4641.373 | 3661.313 | 19 | 4649.05 | 3668.886 |
| 5 | 4641.373 | 3661.313 | 20 | 4649.05 | 3668.886 |
| 6 | 4648.058 | 3665.373 | 21 | 4649.05 | 3668.886 |
| 7 | 4648.058 | 3665.373 | 22 | 4649.05 | 3668.886 |
| 8 | 4648.058 | 3665.373 | 23 | 4649.05 | 3668.886 |
| 9 | 4638.602 | 3670.291 | 24 | 4649.05 | 3668.886 |
| 10 | 4643.731 | 3658.126 | 25 | 4649.05 | 3668.886 |
| 11 | 4638.602 | 3670.291 | 26 | 4649.05 | 3668.886 |
| 12 | 4643.052 | 3650.644 | 27 | 4649.05 | 3668.886 |
| 13 | 4647.055 | 3670.075 | 28 | 4649.05 | 3668.886 |
| 14 | 4648.058 | 3665.373 | 29 | 4649.05 | 3668.886 |
| 15 | 4647.055 | 3670.075 | 30 | 4649.05 | 3668.886 |

从上表中可以看到，KPACA算法在蚁群第19次遍历城市后，收敛到局部最优解4649.05。达到预期的效果。

# 5 结论

本文通过对0-1背包问题进行了初步的讨论，并在深入分析了蚁群算法的基本原理后，通过编程实现KPACA算法。随后本文通过两个仿真实例来验证算法的有效性。

实例一中引用文献[1]中的部分数据。并将结果与文献[1]中的结果进行比较。两者结果完全一致。实例二中采用了随机生成的数据构造0-1背包问题，并且同样得到了较好的结果。

通过本文的讨论，可以认为蚁群算法对于求解0-1背包这类优化问题十分有效。能够在较快的时间内得到问题的一个较优解。

# 6 存在的不足

由于本人学术水平有限，本文的撰写难免存在不足之处，这里总结一下几点不足：

1. 求解0-1背包问题的蚁群算法KPACA虽然具有许多传统算法不具备的优点，但其缺点也很明显：极易陷于问题的局部最优解中。以实例一为例，采用KPACA算法反复多次运行，得到的结果始终都是4721，然而该解并非问题的最优解（文献[1]中采用KPQACA算法得到的解为4776）。说明KPACA算法仍有较大的改进空间。

2. KPACA算法的参数选取并没有合理的分析过程。由于蚁群算法参数空间的庞大以及复杂的关联性，导致参数的选取没有一个合适的标准。应当选取多组参数为基础进行计算，并对比计算结果， 得到更为明确的结论。

# 7 致谢

本人的本科毕业设计论文一直是在杜大刚老师的悉心指导下进行的。他在整个毕业设计过程中，不断对我得到的结论进行总结，并提出新的问题，使我接触到了许多理论和实际上未成发现的问题，不断完善了我的毕业论文，在此表示诚挚的感谢和由衷的敬意。

# 8 参考文献

[1] 赵专政.0-1背包问题的递归算法.益阳师专学报,2002,18(6):50~53

[2] 张治洪，刘玉贵.0-1背包问题的动态状态树的回溯算法.天津理工学院学报,1996,12(4):18~22

[3] 李鸣山，郑海红.0-1背包问题多重分支-限界算法.武汉测绘科技大学学报，1995，50(1):83~88

[4] 温文波，杜维.蚁群算法概述.石油化工自动业,2002,1:19~22

[5] 刘金江.用蚁群算法解决0/1背包问题.中华现代教育.24~25

[6] 王会颖,贾瑞玉等.一种求解0-1背包问题的快速蚁群算法.计算机技术与发展,2007，1(17):104~107

[7] 王潮，时向勇等.基于群体智能的0-1背包问题求解研究进展.微型电脑应用，2007，23(6):1~5

# 附录 参考外文及翻译