第 39 卷 第 4 期
Vol.39 No.4

计 算 机 工 程
Computer Engineering

2013 年 4 月
April 2013

·人工智能及识别技术· 文章编号：1000—3428(2013)04—0248—06 文献标识码：A 中图分类号：TP301.6

求解多维背包问题的 MapReduce 蚁群优化算法

王会颖[1,2,3]，倪志伟[1,2]，吴 昊[1,2]

(1. 合肥工业大学管理学院，合肥 230009；2. 教育部过程优化与智能决策重点实验室，合肥 230009；
3. 安徽财贸职业学院电子信息系，合肥 230601)

摘 要：应用 MapReduce 编程模式实现蚁群优化算法的并行化计算，提出基于 MapReduce 的改进背包问题蚁群算法。通过改进概率计算时机、轮盘赌、交叉、变异等技术，降低蚁群算法的计算复杂度。在云计算环境中应用该算法分布式并行地求解大规模多维背包问题，仿真实验结果表明，该算法能改善蚁群算法搜索时间长的缺陷，增强对大规模问题的处理能力。

关键词：云计算；MapReduce 编程模式；蚁群优化算法；多维背包问题；遗传算法；群体智能

MapReduce-based Ant Colony Optimization Algorithm for Multi-dimensional Knapsack Problem

WANG Hui-ying[1,2,3], NI Zhi-wei[1,2], WU Hao[1,2]

(1. School of Management, Hefei University of Technology, Hefei 230009, China;
2. Key Laboratory of Process Optimization and Intelligent Decision-making, Ministry of Education, Hefei 230009, China;
3. Department of Electronics Information, Anhui Finance & Trade Vocational College, Hefei 230601, China)

【Abstract】This paper uses MapReduce parallel programming mode to make the Ant Colony Optimization(ACO) algorithm parallel and bring forward the MapReduce-based improved ACO for Multi-dimensional Knapsack Problem(MKP). A variety of techniques, such as change the probability calculation of the timing, roulette, crossover and mutation, are applied for improving the drawback of the ACO and complexity of the algorithm is greatly reduced. It is applied to distributed parallel as to solve the large-scale MKP in cloud computing. Simulation experimental results show that the algorithm can improve the defects of long search time for ant colony algorithm and the processing power for large-scale problems.

【Key words】cloud computing; MapReduce programming mode; Ant Colony Optimization(ACO) algorithm; Multi-dimensional Knapsack Problem(MKP); Genetic Algorithm(GA); swarm intelligence

DOI: 10.3969/j.issn.1000-3428.2013.04.057

1 概述

多维背包问题(Multi-dimensional Knapsack Problem, MKP)[1]是运筹学中一个典型的优化难题，是 NP 难问题。对该问题的研究无论在理论上，还是在实践中都具有重要意义。实际问题中的资源分配、投资决策、装载问题、网络资源分配等均可建模为多维背包问题。

求解 MKP 的算法主要包括精确求解和启发式求解 2 类算法[2]。目前，求解 MKP 的方法主要是近年兴起的启发式智能优化算法，如遗传算法[1]、分散搜索算法[3]、蚁群算法[4]等。这些算法的研究主要是在集中式串行的环境中。而在云计算环境下，应用群体智能算法分布式并行求解 MKP 的研究很少。

云计算是并行计算、分布式计算、网格计算的发展，或者说是这些计算科学概念的商业实现。Google 发展了 Google File System[5]、MapReduce[6]等云计算技术，解决了海量数据的存储及快速处理等问题。云计算和智能算法联系紧密，许多群体智能算法因具有隐含并行性，可以在云计算系统中实现分布式并行计算。蚁群优化(Ant Colony Optimization, ACO)算法[7-8]是新近发展的一种仿生类群体

基金项目：国家自然科学基金资助项目(71271071)；国家“863”计划基金资助项目(2011AA040501)；国家社会科学基金资助项目(10CGL024)；安徽省教育厅自然科学基金资助项目(KJ2011A006, KJ2013B010)；合肥学院科研发展基金资助重点项目(12KY03ZD)

作者简介：王会颖(1969－)，女，讲师、博士研究生，主研方向：智能软件，群体智能；倪志伟，教授、博士、博士生导师；吴 昊，博士研究生

收稿日期：2012-03-27 **修回日期：**2012-06-02 **E-mail：**wangh_ying@163.com

智能优化算法，受启发于科学家对蚂蚁觅食模型的研究，具有高的隐含并行性。算法成功地应用于求解许多组合优化问题，如 TSP、二次分配、图着色、集成电路设计等。虽然蚁群算法在求解组合优化问题中表现突出，但也存在缺陷，如搜索时间较长，易于过早地收敛于非最优解；当问题的规模达到一定程度时，在集中式串行环境下求解效率较低。

2　MapReduce 编程模式

Google 构造 MapReduce 编程规范来简化分布式系统编程。编程人员只需将精力放在应用程序本身，而关于集群处理问题，包括可靠性和可扩展性，则交由平台来处理[9]。MapReduce 运行模型中包含 m 个 Map 和 r 个 Reduce。每个 Map 处理一部分不同原始数据，各个 Map 间相互独立，可实现充分的并行化。Reduce 对每个 Map 产生的中间结果进行合并，每个 Reduce 所处理的中间结果互不交叉，可在并行环境下执行，所有 Reduce 产生的结果经过简单的链接就形成了完整的结果集。

3　背包问题的蚁群算法及复杂度分析

MKP 的数学模型可描述为式(1):

$$\max \sum_{j=1}^{n} p_j x_j ; x_j \in \{0,1\}$$
$$\text{s.t. } \sum_{j=1}^{n} r_{ij} x_j \leqslant b_i,\ i=1,2,\cdots,m;\ j=1,2,\cdots,n \qquad (1)$$

其中，n 为物品个数；m>1 为资源个数；P_j>0 为物品 j 的价值；r_{ij}>0 为物品 j 耗用资源 i 的数量；b_i>0 为资源 i 的总量，且 $r_{ij}<b_i$, $r_{i1}+r_{i2}+\cdots+r_{in}>b_i$；$x_j$ 为物品 j 的变量，x_j=1 表示物品 j 被选中，x_j=0 表示物品 j 未被选中。MKP 的本质是在满足某些资源约束的条件下，从候选对象集中找出能够使总的利益函数值最大的一个对象子集。

现有文献中求解 MKP 的 ACO 算法主要过程大体相同，其描述如算法 1 所示。其中，N_{iter} 为最大迭代次数；N_{ant} 为蚂蚁数[10]。

算法 1 ACO 求解 MKP 的算法

```
1  初始化;
2   for(t=1 to N_iter){
3    for(k=1 to N_ant){
4      while(候选集 allowed_k 非空){
5        计算物品 j ∈ allowed_k 的概率 P^k_j;
6        根据概率选择某物品 j;
7        If(物品 j 满足问题约束){
8            将物品 j 加入部分解 S^k; }
9        从候选集 allowed_k 中移去物品 j;
}}//end while, end for k
10   记录蚂蚁求得的最好解;
11   更新信息素; }//end for t
```

这些算法所不同的主要是启发函数、概率计算、选点方式、信息素存放位置及更新方式。文献[4]最早给出了求解子集问题的蚁群算法，并应用于求解 MKP。它将信息素存放在物品上。启发函数采用静态和动态 2 种。静态启发函数 η_j 对所有物品在开始时被设置为固定值。

动态启发函数 $\eta_j(S_k(t))$被定义为蚂蚁 k 在 t 时刻所构造的部分解 $S_k(t)$的函数，$j \in allowed_k$，$allowed_k$ 为 t 时刻蚂蚁 k 没搜索过的物品集合，即蚂蚁构造解时每一步都要重新计算启发函数，计算公式如式(2)所示。概率计算公式如式(3)所示。按此设置，算法 1 的第 5 步计算概率时间复杂度为 $O(nm)$，第 6 步为 $O(n)$，第 7 步~第 8 步为 $O(m)$，第 9 步、第 10 步为 $O(1)$，第 11 步为 $O(n)$。因此，在文献[4]中，每个蚂蚁构造一个解的时间复杂度为 $O(n^2m)$；算法总时间复杂度为 $O(n^2mN_{\text{iter}}N_{\text{ant}})$，空间复杂度为 $O(nm)$，算法的主要计算量集中在第 5 步，若能减小其时间复杂度，则算法的计算量将显著减小。

$$\begin{aligned}
&\delta_{ij}(k,t) = r_{ij} \big/ (b_i - \textstyle\sum_{l \in S_k(t)} r_{il}) \\
&\bar{\delta}_j(k,t) = (\textstyle\sum_{i=1}^{m} \delta_{ij}(k,t)) / m \\
&\eta_j(S_k(t)) = P_j / \bar{\delta}_j(k,t)
\end{aligned} \qquad (2)$$

$$P_{i_p}^k(t) = \begin{cases} \dfrac{[\tau_{i_p}(t)]^{\alpha}[\eta_{i_p}(S_k(t))]^{\beta}}{\sum\limits_{j \in allowed_k(t)} [\tau_j(t)]^{\alpha}[\eta_j(S_k(t))]^{\beta}} & i_p \in allowed_k(t) \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

4　改进背包问题蚁群算法

4.1　算法设计及概念

基于 MapReduce 的改进背包问题蚁群算法(MapReduce-based Improved Ant Colony Optimization for Multidimensional Knapsack Problem, MIAM)应用云计算的 MapReduce 编程模式将蚁群算法并行化，使之适应云计算的分布式并行环境，并对蚁群算法进行改进。MIAM 的设计包含 Map 函数和 Reduce 函数，应用 Map 函数并行化蚁群算法中最耗时的部分——蚂蚁独立求解过程；应用 Reduce 函数处理信息素的更新过程，但信息素的改变对下一代的影响不能体现；应用云计算的管道能力，使 Reduce 函数的输出信息作为下一代 Map 函数的输入，进行下一代的循环求解。相关概念有：

(1)一代：为云计算的一个 job，包括 Map、Reduce 函数。

(2)代数：利用云计算的管道能力，本代 job 的输出作为下一代 job 的输入，使多个 job 串行起来，串行 job 的次数为代数。

(3)蚂蚁家族：云计算环境中运行任务，框架本身需要消耗一定的时间，若 Map 中蚂蚁数量不足，则问题的求解时间较短，额外开销较大。

因此，在一个 Map 中使用多个蚂蚁去求解，则一个 Map 中的多个蚂蚁构成一个蚂蚁家族。蚁群算法中相关

元素的定义如下：启发函数 $\eta_j(j=1,2,\cdots,n)$定义为伪效用函数，即物品 j 的价值 P_j、耗用资源 r_{ij} 和资源总量 b_i 的函数，表达式见式(4)；概率 $P_i(t)$的计算公式见式(5)；信息素存放在物品上，更新公式见式(6)，$\rho(0<\rho<1)$表示信息素挥发程度，对解中为 1 的物品进行信息素的修改，增加$\Delta\tau_j=Q\times P_j/lmb$，其余衰减，$Q$ 为参数，lmb 为利益函数值。

$$\delta_{ij}=r_{ij}/b_i;\ \overline{\delta}_j=(\sum_{i=1}^{m}\delta_{ij})/m;\ \eta_j=P_j/\overline{\delta} \tag{4}$$

$$P_i(t)=\begin{cases}\dfrac{[\tau_i]^{\alpha}[\eta_i]^{\beta}}{\sum_{j\in allowed_k(t)}([\tau_j]^{\alpha}[\eta_j]^{\beta})} & i\in allowed_k(t)\\ 0 & \text{otherwise}\end{cases} \tag{5}$$

$$\tau_j=(1-\rho)\tau_j+\Delta\tau_j \tag{6}$$

4.2 MIAM 的键值对

MIAM 算法包括 Map、Reduce 函数，其参数<*key, value*>的类型及意义如下：*in_key*: integer；蚂蚁家族索引。*in_value*: text；包含最优值、最优解。*key*1: integer；固定整数，如 1。*value*1: text；包含各蚂蚁家族求得的最优值和解。*key*2: integer；蚂蚁家族的索引。*value*2: text；包含当前最优值、最优解。

4.3 MIAM 的 Map 函数

应用 Map 函数并行化蚁群算法中最耗时部分——蚂蚁独立求解过程。其功能是蚂蚁家族中的各成员各自独立生成一个解，对本家族求得的最优解使用局部搜索算法进行优化，输出最优值和解，描述如函数 1 所示。在 MIAM 中的 Map 函数中对蚁群算法进行优化。其详述和改进如下。

引理 MIAM 中启发函数 $\eta_j(j=1,2,\cdots,n)$的计算放在算法 1 的如下位置，其值不变。位置 1：第 1 步；位置 2：第 2 步和第 3 步之间；位置 3：第 5 步。

证明：在 MIAM 中，η_j 的计算如式(4)所示，η_j 仅由物品 j 的价值 P_j、耗用的资源 r_{ij} 和资源总量 b_i 定义，且 P_j、r_{ij}、b_i 在算法的整个求解过程中保持不变，所以 η_j 的值不变。在位置 1 或 2 或 3 计算 η_j，η_j 值不变。

函数 1 MIAM 算法的 Map 函数

```
Function Map(in_key, in_value){
1   初始化；按(4)式计算启发函数 ηj;
2   计算伪效用函数 uj；按 uj 排序生成数组 a;
3   If(是第一代求解){
4     所有物品的信息素 τ(j)＝1;
5   }else{从信息素文件中读取 τ(j)；}
6   根据式(5)计算概率 Pi(t);
7   While(蚂蚁家族中的成员未求解完){
8     While(候选集 allowedk 非空){
9       按物品选择方式选择某物品 j;
10      If(j 满足问题约束){
11        将物品 j 加入解 s；}
12      从候选集中移去 j；}//end while
13    记录家族最优解 bs 和值 bv；}//end while
14  用局部搜索算法优化本家族的 bs 和 bv;
15 Key1=1; value1=bv+bs; 输出 key1,value1；}
```

定理 MIAM 中概率 $P_i(t)$的计算放在位置 2、位置 3，物品间依概率的偏序关系保持不变。

证明：设 S 为物品全集；对 $\forall\ w1\in S,\ \forall\ w2\in S$，$w1\neq w2$，设其信息素分别为 τ_{w1}、τ_{w2}。在算法 1 第 t 次迭代中，只有到第 11 步，信息素才发生变化，即从第 2 步到第 10 步，各物品上的信息素不变。对位置 2，在式(5)中，$allowed_k(t)=S$；记 $sum1=\sum_{j\in S}([\tau_j]^{\alpha}[\eta_j]^{\beta})$，则 $P_{w1}(t)=(\tau_{w1}{}^{\alpha}\eta_{w1}{}^{\beta})/sum1$，$P_{w2}(t)=([\tau_{w2}]^{\alpha}[\eta_{w2}]^{\beta})/sum1$；若 $P_{w1}(t)\geqslant P_{w2}(t)$，则 $\tau_{w1}{}^{\alpha}\eta_{w1}{}^{\beta}\geqslant\tau_{w2}{}^{\alpha}\eta_{w2}{}^{\beta}$。对位置 3，在式(5)中，记 $J=allowed_k(t)$表示在第 t 次迭代中蚂蚁 k 没有搜索过的物品集合，记 $sum2=\sum_{j\in J}([\tau_j]^{\alpha}[\eta_j]^{\beta})$，对上述物品 $w1$ 和 $w2$，记其概率分别为 $P_{w1}{}'(t)$和 $P_{w2}{}'(t)$，则 $P_{w1}{}'(t)=(\tau_{w1}{}^{\alpha}\eta_{w1}{}^{\beta})/sum2$，$P_{w2}{}'(t)=(\tau_{w2}{}^{\alpha}\eta_{w2}{}^{\beta})/sum2$；由位置 2 的 $\tau_{w1}{}^{\alpha}\eta_{w1}{}^{\beta}\geqslant\tau_{w2}{}^{\alpha}\eta_{w2}{}^{\beta}$，得 $P_{w1}{}'(t)\geqslant P_{w2}{}'(t)$。即在位置 2 计算出的物品间的概率偏序关系，若放到位置 3 计算，其偏序关系保持不变。同理可证：在位置 3 计算出的物品间的概率偏序关系，若放到位置 2 计算，其偏序关系保持不变。定理得证。

改变启发函数 η_j 和概率 $P_i(t)$的计算时机：根据引理，MIAM 把 η_j 的计算从算法 1 的第 5 步移到第 1 步中完成，η_j 计算的时间复杂度为 $O(nm)$。计算概率 $P_i(t)$的目的是依据概率确定物品间的偏序关系，然后按一定规则选择物品，若概率的计算放在不同的位置，能保持物品间的偏序关系，则对算法求解的结果没有影响。根据定理 1，MIAM 把概率 $P_i(t)$的计算从算法 1 的第 5 步移到第 2 步和第 3 步之间，计算概率 $P_i(t)$的时间复杂度为 $O(n)$。

改变蚂蚁选择物品的方式，蚂蚁选择物品采用 2 种方式。

方式 1：从 $allowed_k(t)$中选择概率最大者；

方式 2：从 $allowed_k(t)$中依概率按轮盘赌方式选择。

设置参数 $R(0\leqslant R\leqslant 1)$，表示使用轮盘赌方式的比率，生成随机数 $r(0\leqslant r<1)$，若 $r\leqslant R$，则采用方式 2，否则方式 1，且 R 是动态的，在运行前期 R 偏小，后期增大到 1。选择物品的时间复杂度为 $O(n)$。这样，增加了搜索的随机性，有利于跳出局部最优解，向全局最优解转化。

融合遗传算法：在局部搜索(Local Search, LS)算法中融合遗传算法(Genetic Algorithm, GA)。局部搜索算法为在 Map 函数中将上一代的一个解和本代蚂蚁家族构造出的最优解进行下面 4 个过程的操作来改进解：

(1)一致交叉

随机生成交叉模板，产生一个与父辈个体基因串等长的二进制串，该串中 0 表示不交换，1 表示交换。根据该模板对 2 个父辈基因串进行交叉，得到 2 个新基因串，即为后代新个体。例如，父辈个体 A：10110111001，父辈个体 B：00101100100，交叉模板：10011011100，则新个

体 *A*'为：00101100101，新个体 *B*'为：101101110 00。

(2)变异

在后代新个体中随机选取 2 位或 3 位进行变异，由原来的 0 变为 1，或 1 变为 0。

(3)解修复

利用贪心算法根据伪效用函数 u_j 对新个体进行修复，修复算法描述见算法 2。

算法 2 LS 算法的解修复算法

输入个体 s；计算背包中物品所耗用的资源；
If(个体 s 不满足约束条件){
  For(i=0;i<n;i++){//去物品
    If(a[i]对应的物品在背包中){
    取出该物品；If(物品满足约束条件)退出 for 循环; }}}
For(i=n-1;i>0;i- -){//加物品
  If(加入 a[i]对应的物品满足约束条件){
    向背包中加入该物品； }else{退出 for 循环； }}
返回新解；

为了扩大解搜索空间，这里函数 u_j 改变为采用文献[11]设计的伪效用比，定义见式(7)和式(8)。为减少计算，增加一个辅助数组 *a* 存储物品标号的一个排序，a_i 表示按伪效用函数 u_j 升序排序时第 *i* 位物品的标号。函数 u_j 计算和排序在 Map 函数第 2 步完成。

$$r_i = \sum_{j=i}^{n} r_{ij} - b_i \Big/ \sum_{j=i}^{n} r_{ij};\ i = 1,2,\cdots,m \tag{7}$$

$$u_j = P_j \Big/ \sum_{j=i}^{m} (r_i r_{ij});\ j = 1,2,\cdots,n \tag{8}$$

(4)循环

若有更好的解，则新解代替原解；继续执行局部搜索直到一定的次数 N_{ls}。算法 LS 的时间复杂度为 $O(mnN_{ls})$，因为：一致交叉为 $O(n)$；变异为 $O(1)$；解修复为 $O(mn)$。

4.4 MIAM 的 Reduce 函数

MIAM 中应用 Reduce 函数处理信息素的更新过程。其主要功能是：从 Map 输出的 *value*1 中分解出问题的解和值，记录全局最优值和解，对不同的解和全局最优解，根据式(6)和最大最小信息素的值更新信息素，写出更新后的信息素文件，输出不同的解和值及全局最优解和值。在 Reduce 函数中对蚁群算法的信息素处理方式进行改进，其描述为：

(1)设置信息素的最大值 *max_t* 和最小值 *min_t*；信息素更新时，若某物品上的信息素大于 *max_t*，则设置为 *max_t*；若信息素小于 *min_t*，则设置为 *min_t*。

(2)保留求解以来得到的最优解 *s_op*；信息素的更新不仅对各蚂蚁家族本代求得局部最优进行更新，而且根据 *s_op* 进行更新。其描述如函数 2 所示。

函数 2 MIAM 算法的 Reduce 函数

Function Reduce(key1, list(value1)){
  v_max=0; HashSet set; 从信息素文件中读出信息素τ
  while(list(value1) is not null){
    从 value1 中取出解 s 和值 v；
    If(v>v_max){ v_max=v; s_op=s; }
    If(set.add(v)==true){//不同的解
      Key2=v; value2= s;输出<key2,value2>;
      按式(6)，s,max_t, min_t 更新信息素}}//end if, while
  按式(6)，s_op, max_t,min_t 更新信息素；
  把更新后的信息素写入信息素文件；
  Key2=v_max; value2= s_op; 输出<key2,value2>;}

4.5 MIAM 的任务管道

因为一代求解不能完成任务，所以在 MIAM 中引入任务管道，实现任务串行策略。设计为：将多对 Map、Reduce 任务串行起来，形成 M1→R1→M2→R2→…→Mn→Rn 的执行序列。将 MIAM 中 Reduce 的输出作为下一轮 Map 的输入，继续下一个并行计算任务。

4.6 MIAM 的算法框架及复杂度分析

MIAM 算法框架如图 1 所示。

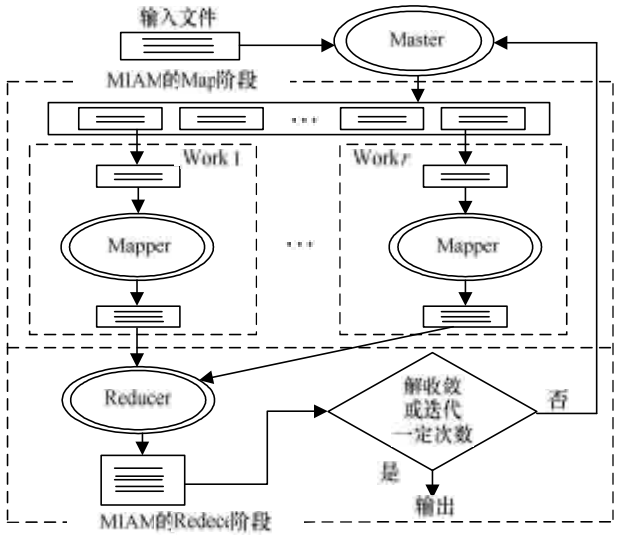

图 1 算法 MIAM 框架

综合算法框架、Map 函数、Reduce 函数内容分析 MIAM 的复杂度。其空间复杂度为 $O(nm)$。和文献[4]相同，设置 MIAM 任务迭代的次数(代数)为 N_{iter}，蚂蚁总数为 N_{ant}，Map 数量为 N_{Map}，蚂蚁家族成员数为 N_{member}，且 $N_{\text{ant}}=N_{\text{Map}}\times N_{\text{member}}$。在 Map 函数(函数 1)中，第 1 步~第 2 步为 $O(\max(n,m)n)$，因为：启发函数、伪效用函数 u_j 计算为 $O(nm)$，u_j 为排序 $O(n^2)$；第 3 步~第 5 步为 $O(n)$；第 6 步为 $O(n)$；第 8 步~第 12 步，一个蚂蚁求解为 $O(\max(n,m)n)$；第 7 步~第 13 步为解 $O(\max(n,m)nN_{\text{member}})$；第 14 步为 $O(mnN_{ls})$，且设置 N_{ls} 不大于 N_{member}；因此，一个 Map 为 $O(\max(n,m)nN_{\text{member}})$，$N_{\text{Map}}$ 个 Map 为 $O(\max(n,m)nN_{ant})$。Reduce 函数(函数 2)为 $O(nN_{\text{Map}})$。所以 MIAM 的时间复杂度为 $O(\max(n,m)n\ N_{\text{ant}}N_{\text{iter}})$。而文献[4]算法的时间复杂度为 $O(n^2mN_{\text{ant}}N_{\text{iter}})$，MIAM 比文献[4]的算法复杂度大幅降低，说明 MIAM 算法能大大减少计算量，有效地改进蚁群算法搜索时间长的缺陷。

4.7 基于 MapReduce 的背包问题蚁群算法

基于 MapReduce 的背包问题蚁群算法(Map Reduce-based Ant Colony Optimization for Multi-dimensional Knapsack Problem, MAM)框架，Map、Reduce 的原理及过程和 MIAM 基本相同。不同之处在于蚁群算法的具体实现，采用算法 1 实现蚁群算法求解多维背包问题，这里不再赘述。

5 仿真实验及分析

5.1 实验环境及参数

实验硬件环境为：Intel(R) Core (TM)2 Dou CPU，2.00 GHz；2 GB 内存；100 Mb/s 带宽；2 台计算机。实验运行在 hadoop 环境中，实验软件选用 Ubuntu、hadoop、eclipse 等。实验数据选用通用的 ORLIB[12]中的 MKP 基准测试数据，测试实例选取大规模难解的实例；实验参数包括：蚁群算法中的 α、β、ρ、Q、max_t、min_t；MIAM 中蚁群家族的成员数(N_{member})、Map 数(N_{Map})、机群中计算机数($N_{computer}$)；局部搜索迭代次数 N_{ls} 等。

5.2 算法的求解性能

实验选取 5.100、5.250、5.500 实例组中不同问题紧度的实例，问题紧度分别为 α=0.25,0.5,0.75，应用算法 MIAM 和 MAM 对这些实例分别求解，比较其求解性能和效率。实验参数为：$\alpha=\beta=Q=1$, $\rho=0.05$, $max\_t=1$, $min\_t=0.0001$；$N_{member}=n/2$, $N_{Map}=10$, $N_{computer}=2$；$N_{ls}=n/4$。实验结果如表 1、表 2 所示。表 1 为连续运行 10 次的结果，其中，次数为 10 次运算中得到 ORLIB 提供的最优值的次数，代数为算法首次得到其最优值时迭代的次数。

算法 MIAM 在求解 5.250-12 实例时产生新解，得到的最优值 108 501 好于 ORLIB 的最优值 108 489。最优值为 108 501 时，其占用的各资源总数如表 2 所示。

表 1 MIAM、MAM 算法求解结果比较

实例	ORLIB 最优值	MIAM 算法					MAM 算法				
		最优值	次数	平均值	代数	每代时间/s	最优值	次数	平均值	代数	每代时间/s
5.100-00	24 381	24 381	6	24 373.6	24	24.03	24 381	5	34 363.6	20	23.95
5.100-10	42 757	42 757	8	42 746.6	19	23.68	24 757	7	42 741.4	19	23.75
5.100-22	59 802	59 802	6	59 786.0	32	23.44	59 802	5	59 782.0	27	22.55
5.250-05	60 056	60 056	4	60 039.3	90	23.43	60 035	0	59 968.9	60	26.84
5.250-12	108 489	108 501	1	108 465.0	70	23.33	108 472	0	108 455.1	46	27.12
5.250-22	149 316	149 316	3	149 301.0	97	23.13	149 266	0	149 223.6	65	28.40
5.500-02	121 109	121 093	0	120 988.0	342	26.54	120 973	0	120 922.4	78	48.94
5.500-19	219 693	219 684	0	219 635.0	395	27.04	219 624	0	219 562.8	95	56.15
5.500-27	306 430	306 422	0	306 376.0	368	26.73	306 414	0	306 362.3	89	51.27

表 2 MIAM 算法的最优解所占用的各资源总量

实例	最优值	最优值时占用的资源量
5.100-00	24 381	(11 822, 13 714, 11 376, 12 931, 13 412)
5.100-10	42 757	(23 950, 24 243, 26 084, 25 544, 24 277)
5.100-22	59 802	(316 182, 39 686, 38 209, 36 775, 36 999)
5.250-05	60 056	(31 389, 29 710, 31 616, 29 138, 30 613)
5.250-12	108 501	(60 276, 58 326, 62 264, 67 183, 59 519)
5.250-22	149 316	(93 850, 99 427, 94 871, 92 851, 91 768)

从表 1、表 2 看出，算法 MIAM 得到 5.100、5.250 组实例 ORLIB 提供的最优值，说明该算法对较大规模问题求解能得到好的结果。虽然对 5.500 组更大规模的实例没得到已知最优值，但相差较小。比较算法 MIAM 和 MAM 的结果，MIAM 算法的平均值和得到最优值的次数均优于 MAM。表 1 和表 2 说明改进算法对大规模问题的求解效果较好，具有较好的求解性能。

5.3 算法的时间开销和求解效率

在云计算环境中，任务运行时间，除问题求解所需的时间外，框架也要消耗一定的时间。云计算框架耗用的时间包括：任务的部署，Map 中间结果的排序、归类，Reduce 结果的归并，结果的输出等时间。问题求解所需时间包括：Map 和 Reduce 时间等。为此，研究云计算环境中算法运行时各阶段的时间开销，以更好地求解问题。

选用更大规模的 30.500-00 实例比较算法 MIAM 和 MAM 运行时各部分的时间开销。实验参数设置为：N_{member} 为 n，Map 个数分别取 10、20、30，其余参数同上。结果如表 3、图 2 所示。

表 3 MIAM、MAM 算法各阶段运行时间比较 s

比较项	$T1$	$T2$	$T3$	$T4$	$T5$	总计
I-10	6.336	9.884	9.101	0.213	7.892	33.426
M-10	6.025	222.099	9.035	0.180	8.032	245.371
I-20	7.963	13.846	8.706	0.179	7.574	38.268
M-20	8.213	437.918	8.864	0.207	7.731	462.933
I-30	7.429	14.819	8.758	0.218	6.820	38.044
M-30	7.162	595.170	8.631	0.206	7.214	618.383

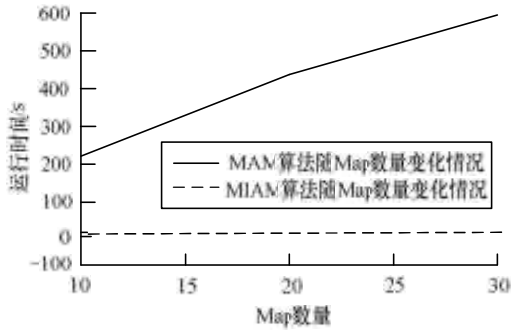

图 2　Map 阶段运行时间比较

表 3 中结果为连续运行 20 代，每代各阶段的平均值。*T*1 表示任务开始到 Map 开始运行所需的时间；*T*2 是 Map 阶段的时间；*T*3 是从 Map 结束到 Reduce 开始的时间，包括中间结果的排序、归类等操作的时间；*T*4 为 Reduce 阶段的时间；*T*5 为 Reduce 结束到下一任务开始的时间，包括 Reduce 结果的归并、输出等时间。I 表示改进算法 MIAM、M 表示算法 MAM；10、20、30 分别代表 Map 的数量。如 I-10 表示算法 MIAM 在 Map 数量为 10 的情况下各阶段的运行时间。*Total* 为总时间。图 2 为表 3 中的 *T*2(Map)阶段运行时间的比较。

从表 3 可以看出，*T*1、*T*3、*T*5 这 3 个部分框架耗用的时间基本稳定，为云计算环境中运行任务必要的时间。*T*2、*T*4 为问题求解的有效时间。

从表 3、图 2 可以看出，对算法 MIAM，因该算法的求解效率高，Map 阶段时间短，框架占用的时间长，Map 数量的变化对任务执行时间影响不大。对算法 MAM，Map 时间长，问题求解困难，Map 数量的变化对任务执行时间影响很大。因此，在实际运行任务时，可根据具体情况和用户的需求更好地求解问题。

从表 1 可以看出，对 5.100 组的实例，算法 MIAM 和 MAM 平均每代的执行时间相差不大。这是因为在这组实例中问题本身求解的时间很短，框架消耗的时间比例很大，体现不出问题求解时间的差别。但对 5.500 实例组算法 MIAM 和 MAM 相差约一倍，初步体现了改进算法的求解效率。表 3 中对大规模实例，2 个算法的求解时间相差很大，充分体现改进算法的求解效率，有效实证了算法复杂度分析所描述的 MIAM 复杂度的降低，说明改进算法能改善蚁群算法搜索时间长的缺陷。

6　结束语

本文应用 MapReduce 的编程模式将蚁群算法并行化，定义了 Map 函数和 Reduce 函数，并应用于求解多维背包问题，提出基于 MapReduce 的改进多维背包问题蚁群算法，使之对问题的求解可运行在分布式并行的云计算环境中。且对蚁群算法进行优化，采用改变启发函数和概率的计算时机，物品选择方式上加入轮盘赌，设置最大最小信息素，应用交叉和变异算子进行局部优化等技术改善蚁群算法的缺陷，算法复杂度从 $O(n^2mN_{ant}N_{iter})$降低到 $O(\max(n,m)nN_{ant}N_{iter})$。实验运行在 Hadoop 云计算环境中，选用 ORLIB 中标准的大规模 MKP 测试实例，实验分析了算法的有效性、时间开销及求解效率，改进后的算法 MIAM 取得了较好的结果，并得到实例 5.250-12 的新解。研究算法相关参数的变化对问题求解的影响是下一步要进行的工作。

参考文献

[1] Chu P, Beasley J. A Genetic Algorithm for the Multidimensional Knapsack Problem[J]. Journal of Heuristics, 1998, 4(1): 63-86.

[2] Freville A. The Multidimensional 0-1 Knapsack Problem: An Overview[J]. European Journal of Operational Research, 2004, 155(1): 1-21.

[3] Hanafi S, Wilbaut C. Scatter Search for the 0-1 Multidimensional Knapsack Problem[J]. Journal of Mathematical Modeling and Algorithms, 2008, 7(2): 143-159.

[4] Leguizamon C, Michalewicz Z. A New Version of Ant System for Subset Problems[C]//Proc. of Congress on Evolutionary Computation. Piscataway, USA: [s. n.], 1999: 1459-1464.

[5] Ghemawat S, Gobioff H, Leung S T. The Google File System[C]//Proc. of the 19th ACM Symposium on Operating Systems Principles. New York, USA: ACM Press, 2003: 29-43.

[6] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]//Proc. of the 6th Symposium on Operating System Design and Implementation. Berkeley, USA: USENIX Association, 2004: 137-150.

[7] Colorm A, Dorigo M, Manieaao V. Distributed Optimization by Ant Colonies[C]//Proc. of the 1st European Conference on Artificial Life. Paris, France: Elsevier Publishing, 1991: 134-142.

[8] Origo M. Optimization Learning and Nature Algorithms[D]. Milano, Italy: Department of Electronics, Politecnico di Milano, 1992.

[9] 陈　康，郑纬民. 云计算：系统实例与研究现状[J]. 软件学报, 2009, 20(5): 1337-1348.

[10] 喻学才，张田文. 多维背包问题的一个蚁群优化算法[J]. 计算机学报, 2008, 31(5): 810-819.

[11] Puchinger J, Raidl G R, Pferschy U. The Core Concept for the Multidimensional Knapsack Problem[M]. Berlin, Germany: Springer, 2006.

编辑　顾逸斐