

多维背包问题的一个蚁群优化算法

喻学才 张田文

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 蚁群优化(ACO)是一种通用的启发式方法,已被用来求解很多离散优化问题.近年来,已提出几个 ACO 算法求解多维背包问题(MKP).这些算法虽然能获得较好的解但也耗用太多的 CPU 时间.为了降低用 ACO 求解 MKP 的复杂性,文章基于一种已提出但未实现过的 MKP 的信息素表示定义了新的选择概率的规则和相应的基于背包项的一种序的启发式信息,从而提出了一种计算复杂性较低、求解性能较好的改进型蚁群算法.实验结果表明,无论串行执行还是虚拟并行执行,在计算相同任务时,新算法耗用时间少且解的价值更高.不仅如此,在实验中,文新的算法获得了 ORLIB 中测试算例 5.250-22 的两个“新”解.

关键词 蚁群优化;信息素模型;启发式信息;组合优化;多维背包问题

中图法分类号 TP316

An Improved Ant Algorithm for Multidimensional Knapsack Problem

YU Xue-Cai ZHANG Tian-Wen

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract Ant Colony Optimization (ACO) is a metaheuristic. And it has been applied to many hard discrete optimization problems. Recently, some researchers have proposed several different ACO algorithms to solve the multidimensional knapsack problem (MKP), which is an NP-hard combinatorial optimization problem. Although these algorithms could obtain good solutions of MKP, they consume too more CPU time. For the sake of decreasing the time complexity of the existing ACO algorithms, a new improved ACO algorithm is proposed for MKP in this paper. First, a pheromone representation, which was defined by Blum C. in his paper "the hyper-cube framework for ant colony optimization" as an example, models binary decision variables assigned to all items and their values. Based on this pheromone model, a new rule choosing the objective item is defined. In this way, the time complexity of the proposed ACO algorithm can be decreased. However, incorporating the heuristic information into the solution construction of the artificial ants is difficult. A new heuristic information of MKP based on some order of all items is described. Experimental results show that in the case of the same computing task, the new one uses less CPU time and obtains better solutions compared with other ACO algorithms whether the serial or parallel implementation. Also, the new algorithm has obtained two new solutions of test 5.250-22.

Keywords ant colony optimization; pheromone model; heuristic information; combinatorial optimization; multidimensional knapsack problem

1 引言

多维背包问题(MKP)^[1]是一类 NP 难的组合优化问题. 给定待选项的集合 $I = \{1, 2, \dots, n\}$, MKP 可以叙述如下: 从集合 I 中选出一组满足所有问题约束的项, 使其价值之和最大. MKP 的数学公式如下:

$$\text{Max} \sum_{j=0}^{n-1} p_j x_j \quad (1)$$

满足于

$$\begin{aligned} \sum_{j=0}^{n-1} r_{ij} x_j &\leq b_i, i=0, 1, \dots, m-1; \\ x_j &\in \{0, 1\}, j=0, 1, \dots, n-1. \end{aligned}$$

式中: n 为项数, $n > 1$; p_j 为项 j 的价值, $p_j > 0$; x_j 为对应项 j 的变量, 项 j 被选中对应 $x_j = 1$, 否则, $x_j = 0$; $r_{ij} > 0$ 为项 j 耗用资源 i 的量; b_i 为资源 i 的总量; m 为约束数, $m > 1$.

一个定义完整的 MKP 应该满足 $r_{ij} < b_i$ 并且

$$\sum_{j=1}^n r_{ij} > b_i.$$

近年来, 蚂蚁、蜜蜂等群居昆虫的集体行为引起了人们的广泛关注. 每个昆虫的能力虽然十分有限, 但昆虫群体的能力却远远超过所有个体能力的总和. 比如, 蚂蚁群可以快速建立起巢穴与食物源之间的最短路径. 令人惊奇的是, 每只蚂蚁并不直接比较每条路径, 而仅仅是遵守信息素迹释放/跟随规则就能找到最佳路径. 蚂蚁群的这种能力很自然地引起了计算机科学家的兴趣. 人工智能发展中的最大困难是“组合爆炸”问题, 其实质是组合优化问题. 如果能定义某种计算能力有限的“人工昆虫”, 使其组成的群体具有解决某类组合优化难题的能力, 则整个人工智能甚至整个计算技术都会有一个里程碑式的发展. Dorigo 等人沿着这个方向迈出了第一步: 他们根据蚁群的觅食行为定义了求解旅行商问题(TSP)的蚁群(AS)^[2]算法, 并取得了令人鼓舞的结果.

蚁群优化(ACO)^[3]是由 AS 算法演变而成的一种难解问题的通用启发式解法. 现已广泛用于求解包括 TSP 在内的各种组合优化问题. 应用 ACO 求解 MKP 是一项颇具挑战性的研究工作. 近年来, 国内外的研究者对此作了一些尝试性的应用研究. Leguizamón 等人分析了应用 ACO 求解 TSP 和 MKP 之间的区别, 并将 AS 算法应用于 MKP^[4]. Fidanova 提出增强当前未被蚂蚁访问过的项的迹

浓度, 以增加这些项被蚂蚁选中的概率^[5]. Parra-Hernandez 等人将 ACS^[6]的迹全局更新策略更改为满足一定条件的新解发现时才进行, 并引入迹的局部存储, 由此实现了 ACS 求解 MKP 的并行计算^[7]. Alaya 等人将信息素迹放在被选择项的所有序对组合上, 在此基础上提出了求解 MKP 的 Ant-knapsack 算法^[8]. 国内与此相关的文献[9-11]大多是将国外文献中的算法用于求解背包问题, 对 ACO 算法求解背包问题的性能并没有多少改进. 仔细分析文献中的求解 MKP 的 ACO 算法会发现它们的复杂性偏高. 然而, 应用 ACO 算法求解 MKP 的算法复杂性还可以进一步降低.

本文在文献[12]背包问题表示的基础上提出了一种新的构造解的规则, 并为背包问题设计了一种适合该规则的新的启发式信息. 改进后的 ACO 求解 MKP 的空间复杂性为 $O(nm)$, 构造每个解的时间复杂性为 $O(nm)$. 实验表明, 无论在串行执行还是在虚拟并行执行, 在计算目标函数次数相同的条件下, 本文的 ACO 算法不仅耗用时间少, 而且求得最优解的能力也提高了. 在后面的叙述中, 称本文实现的 ACO 算法为 NAntKp.

本文第 2 节简要描述现有文献中求解 MKP 的 ACO 算法的关键内容并分析了它们的复杂性; 第 3 节详细定义 NAntKp 算法; 第 4 节应用 NAntKp 算法求解 MKP 的标准测试实例, 讨论算法性能并与其它 ACO 算法比较; 第 5 节简单总结全文并指出了将来的研究方向.

2 已有算法简介及复杂性分析

MKP 的一个解是所有项的集合的子集. 现有文献中描述的 MKP 的各种 ACO 算法采用了不同的信息素释放方法和不同的启发式信息. 不过, 这些算法的主要过程大致相同. 假设给定的最大迭代步数为 N_{iter} , 允许搜索的解的数量为 N_{sol} , 蚂蚁数为 N_{ant} ($N_{\text{sol}} = N_{\text{iter}} N_{\text{ant}}$), 算法 1 描述了这个过程.

算法 1. AntMkp.

1. 初始化信息素值;
2. while 迭代步数不大于 N_{iter} do
3. for 每只蚂蚁 k do
4. while 可行候选集 allowed_k 非空 do
5. 计算选择项 $j \in \text{allowed}_k$ 的概率 P_j^k ;
6. 根据随机比例规则选择某项 j ;
7. if 项 j 满足问题约束 then
8. 将项 j 加入部分解 \tilde{S}^k ;

9. end if
10. 从候选集 $allowed_k$ 中移去项 j ;
11. end while
12. end for
13. 记录最好蚂蚁;
14. 更新信息素值;
15. end while

不同的算法采用不同的信息素模型,因而有不同的启发式信息和选择概率的计算方法即行 5 和 6 的具体实现不同,进而有不同的复杂性和求解性能. 下面简要描述几个算法选用的信息素模型,选择概率及启发式信息的计算方法,并在算法 1 的框架下分析它们的复杂性.

2.1 KpAs 算法

Leguizamón 等人^[4]最早应用 AS 算法求解 MKP. 本文称他们描述的算法为 KpAs 算法. KpAs 将信息素释放在项上. 蚂蚁从一个空解开始,一步一步地根据项的信息素信息和启发式信息在所有可选项中概率地选择一个项并加入到当前解中. 记一个蚂蚁 k 在迭代步 t 的部分解 $\tilde{S}_k(t) = \{i_0, i_1, \dots, i_j\}$, 则选择下一个组件 i_p ($p \in \{j+1, j+2, \dots, n-1\}$) 的概率选择公式如下:

$$P_{i_p}^k(t) = \begin{cases} \frac{[\tau_{i_p}(t)]^\alpha [\eta_{i_p}(\tilde{S}_k(t))]^\beta}{\sum_{j \in allowed_k(t)} [\tau_j(t)]^\alpha [\eta_j(\tilde{S}_k(t))]^\beta}, & i_p \in allowed_k(t) \\ 0, & \text{其它} \end{cases}$$

其中, $allowed_k(t) \subseteq I - \tilde{S}_k(t)$ 是余下的待选可行项的集合; $\eta_j(\tilde{S}_k(t))$ 是依赖于部分解的动态启发式信息; α 和 β 是控制信息素信息和启发式信息重要性的参数.

启发式信息 $\eta_j(\tilde{S}_k(t))$ 是部分解 $\tilde{S}_k(t)$ 的函数. 这就是说,在构造解的每一步,对每个 $j \in allowed_k(t)$, 都要重新计算 $\eta_j(\tilde{S}_k(t))$. 记

$$\delta_{ij}(k, t) = \frac{r_{ij}}{c_i - \sum_{l \in \tilde{S}_k(t)} r_{il}}, \quad \bar{\delta}_j(k, t) = \frac{\sum_{i=1}^m \delta_{ij}(k, t)}{m}$$

分别表示项 j 对部分解 $\tilde{S}_k(t)$ 而言关于约束 i 的紧度(tightness)和关于所有约束的平均紧度,那么,启发式信息的计算方法如下

$$\eta_j(\tilde{S}_k(t)) = \frac{p_j}{\bar{\delta}_j(k, t)}.$$

结合算法 1 分析 KpAs. 在行 4~11, 当蚂蚁 k 在构造一个解的每一步时,如果有项加入到部分解

$\tilde{S}_k(t)$ 中,则需要对候选集 $allowed_k$ 中的所有项计算启发式信息 η_j . 为了降低时间复杂性,需要存储部分分解已耗用的每种资源量. 因此空间复杂性为 $O(m)$, 时间复杂性为 $O(nm)$. 行 5 和 6 的计算概率和随机比例选择的时间复杂性为 $O(n)$. 行 7~9 检查可行性的时间复杂性为 $O(m)$. 所以,构造一个解的时间复杂性为 $O(n^2m)$, 空间复杂性为 $O(m)$. 由于更新信息素值的时间复杂性为 $O(n)$, 存储问题数据的空间复杂性为 $O(nm)$, 于是整个算法的空间复杂性为 $O(nm)$, 时间复杂性为 $O(n^2mN_{sol})$.

2.2 AntKnapsack 算法

AntKnapsack 算法^[8]与许多别的蚁群优化算法不同的地方是它在 MKP 的一个解包含的所有项的序对上释放信息素. 当加入某个项到当前的部分解时,选择项的概率依赖于连接部分解中的项和候选项的所有序对上的信息素值. 记 $\tau_{i_p}(\tilde{S}_k) = \sum_{j \in \tilde{S}_k} \tau(i_p, j)$ 表示候选项为 i 时的信息素值,则项的选择概率定义为

$$P_{i_p}^k = \frac{[\tau_{i_p}(\tilde{S}_k)]^\alpha [\eta_{i_p}(\tilde{S}_k)]^\beta}{\sum_{j \in allowed_k} [\tau_j(\tilde{S}_k)]^\alpha [\eta_j(\tilde{S}_k)]^\beta},$$

其中, $\eta_j(\tilde{S}_k) = \frac{p_j}{\bar{\delta}_j(k)m}$ 是启发式信息, $\bar{\delta}_j(k)m$ 表示相对于部分解 \tilde{S}_k 而言项关于 j 所有约束的紧度的和; α, β 的意义同前文.

同样地,结合算法 1 分析 AntKnapsack 可得空间复杂性为 $O(\max\{nm, n^2\})$, 时间复杂性为 $O(n^2mN_{sol})$. 不过,在计算解的数量相同的条件下, AntKnapsack 构造一个解耗用的时间比 KpAs 多; 整个求解过程耗用的时间也会比 KpAs 多(另外的原因是信息素更新的时间复杂性和频率比 KpAs 高).

2.3 KpArAco 算法

KpArAco 算法^[5]释放信息素于解 $\tilde{S}_k = \{i_0, i_1, \dots, i_j\}$ 包含的项的顺序序对 $(i_0, i_1), \dots, (i_{j-1}, i_j)$ 上. 它的主要想法是在信息素更新时增加在一次迭代中未被包含在任何蚂蚁的解中的序对上的信息素值,使之大于包含在最坏解中的序对上的信息素值,但小于包含在最优解中序对上的信息素值. 从而增加未被使用过的序对在下次迭代中被使用的概率. 项的选择概率依赖于最近加入部分解的项 i_j 与候选集中的项 i_p 组成的序对上的信息素值 $\tau_{i_j i_p}$,

$$P_{i_p}^k(t) = \frac{\tau_{i_j i_p} \eta_{i_p}}{\sum_{i_b \in allowed_k} \tau_{i_j i_b} \eta_{i_b}}, \quad i_p \in allowed_k.$$

其中, $\eta_j = \frac{p_j^{d_1}}{(\sum_{i=1}^m r_{ij})^{d_2}}$ 是启发式信息, 表示项 j 的价值与耗用所有资源的比, d_1, d_2 是调节这个比值的参数.

KpArAco 算法的空间复杂性为 $O(\max\{nm, n^2\})$, 构造一个解的时间复杂性为 $O(\max\{nm, n^2\})$, 整个算法的时间复杂性为 $O(\max\{nm, n^2\} \cdot N_{\text{sol}})$.

2.4 KpAcs 算法

KpAcs 算法^[7]在一个解包含的项上释放信息素. 它对 ACS 算法^[6]的主要改进在于: (1) 每只蚂蚁存储自己的信息素值, 信息素值的局部更新不影响别的蚂蚁的行为; (2) 最好蚂蚁的信息素值的全局更新要么在搜索到一个更好的解后进行, 要么依赖于与时间相关的更新机制, 而不是每个迭代步之后都更新; (3) 信息素值在全局更新之后, 最好蚂蚁将其传给别的蚂蚁. 经过这些改进, KpAcs 的求解性能增强了, 且适宜并行实现. 项的选择概率如下:

$$P_j^k = \frac{\tau_j \eta_j^\beta}{\sum_{i \in \text{allowed}_k} \tau_i \eta_i^\beta}.$$

记 $\mu_j = \frac{p_j}{\sum_{i=1}^m \lambda_i r_{ij}}$, 其中 λ_i 是一组拉格朗日乘子

(Lagrange multiplier). 启发式信息定义为

$$\eta_j = \frac{\mu_j}{\max\{\mu_j : j = 0, 1, \dots, n-1\}} + 1.$$

分析 KpAcs 算法可知空间复杂性为 $O(nm)$; 构造一个解的时间复杂性为 $O(\max\{nm, n^2\})$, 整个算法的时间复杂性为 $O(\max\{nm, n^2\} N_{\text{sol}})$.

3 NAntKp 算法

从前文可知, 算法 1 中行 5 和 6 的时间复杂性至少为 $O(n)$. 这是这些算法复杂性偏高的根源. 如果能将行 5 和 6 的时间复杂性降至 $O(1)$, 则整个算法的计算量可以显著地降低. 事实上, 将信息素释放在问题的变量和它所有可能的取值的组合上, 就能达到此目的. 这就是文献[12]中针对背包类问题所举的一个例子. 但这种只使用信息素引导的搜索只能有效地求解很小规模的问题, 因此必须将具体问题实例的启发式信息集成到项的选择概率中. 然而, 集成启发式信息是一件困难的事情, 这正是我们解决的关键问题.

3.1 信息素模型

设问题(1)的解集为 $\{x | \langle x_0, \dots, x_j, \dots, x_{n-1} \rangle : x_j \in \{0, 1\}\}$, 则搜索空间的大小 $|\{x\}| = 2^n$. 定义问题(1)的信息素模型 $\mathcal{T} = \{\tau_{rj} | r \in \{0, 1\}, j = 0, 1, \dots, n-1\}$, τ_{0j} 对应 $x_j = 0$, τ_{1j} 对应 $x_j = 1$. 也就是说, 人工蚁对每个待选项有两种处理: ‘选’或‘不选’. 当作出‘选’时, 人工蚁就增加 τ_{1j} , 使得再构造候选解时, 作出‘选’这种处理的概率比‘不选’大; 反之亦然.

我们将在 HCF-ACO^[12] 框架下参照 MMAS^[13] 算法的规则定义新算法. 所以, 将信息素值均初始化为 1, 即 $\tau_{rj} = 1$.

3.2 项的选择概率

在迭代步 t , 蚂蚁 k 从 $j = 0$ 到 $j = n-1$ 逐项按式(2)计算变量 $x_j = 1$ 的概率, 即项 j 被‘选’的概率.

$$p_{1j}^k(t) = \frac{\tau_{1j}^\alpha(t) \eta_j^\beta}{\tau_{0j}^\alpha(t) + \tau_{1j}^\alpha(t) \eta_j^\beta} \quad (2)$$

式中: α 为信息素信息的重要性因子, β 为下面将要定义的序启发式信息 η_j 的重要性因子.

因为蚂蚁 k 对项 j 要么‘选’, 要么‘不选’, 故变量 $x_j = 0$ 的概率 $p_{0j}^k(t) = 1 - p_{1j}^k(t)$.

3.2.1 序启发式信息的定义

在式(1)的 m 个约束两边各乘一个非负常数 μ_i , 整理后得其替代约束 (surrogate constraint) 如下式:

$$\sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} \mu_i r_{ij} \right) x_j \leq \sum_{i=0}^{m-1} \mu_i b_i \quad (3)$$

替代乘子 (surrogate multipliers) 向量 $\langle \mu_0, \mu_1, \dots, \mu_{m-1} \rangle$ 的计算原则是: 使替代松弛 (surrogate relaxation) 后的一维 0-1 背包问题的最优解能最大程度地逼近原始的 MKP 的最优解. 计算实践表明^[1], MKP 线性松弛 (LP relaxation) 的对偶问题 (duality problem) 的变量值就是一组非常好的 $\mu_0, \mu_1, \dots, \mu_{m-1}$ 的值. 线性规划松弛后, MKP 演变为下面的线性规划:

$$\text{Max} \sum_{j=0}^{n-1} p_j x_j \quad (4)$$

满足于

$$\sum_{j=0}^{n-1} r_{ij} x_j \leq b_i, \quad i = 0, 1, \dots, m-1, \\ x_j \in [0, 1], \quad j = 0, 1, \dots, n-1.$$

注意式(4)允许 x_j 可取 $[0, 1]$ 区间内的任何值, 而式(1)仅允许 x_j 取 0, 1 两个值. 求式(4)的对偶线性规划即得 $\mu_0, \mu_1, \dots, \mu_{m-1}$ 的值. 从而将形如式(1)的

MKP 问题转化为如下形式的 0-1 背包问题

$$\text{Max} \sum_{j=0}^{n-1} p_j x_j \quad (5)$$

满足于

$$\sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} \mu_i r_{ij} \right) x_j \leq \sum_{i=0}^{m-1} \mu_i b_i, \\ x_j \in \{0, 1\}, j = 0, 1, \dots, n-1.$$

记 $\tilde{\eta}_j = p_j / \sum_{i=0}^{m-1} \mu_i r_{ij}$, 且 $\tilde{\eta}_0 \geq \tilde{\eta}_2 \geq \dots \geq \tilde{\eta}_{n-1}$, 则

序启发信息 η_j 就是 $\tilde{\eta}_j$ 的规范值.

3.2.2 序启发式信息的规范处理

在式(2)中 η_j 和 β 共同决定启发式信息的重要程度. 由于 η_j 的定义要求所有项按 η_j 从大到小排好序, 如果 η_j 的值太大, 则 $x_j=1$ 的概率几乎总是 1, $x_j=0$ 的概率几乎总是 0, 这就使 NAntKp 算法退化为贪心爬山算法, 求解性能显著降低; 反之, 如果 η_j 的值太小, 则 $p_{ij}^k(t)$ 的值主要由信息素值决定, 与 η_j 的值几乎无关, 这就使 η_j 不能引导蚁群搜索, NAntKp 算法退化为纯粹的随机算法, 求解性能也会显著下降. 因此, 必须控制 η_j 的值在某个适当的范围内. 另外, 当 $\eta_j > 1$, $x_j=1$ 的概率比 $x_j=0$ 的概率大; 当 $\eta_j < 1$, $x_j=0$ 的概率比 $x_j=1$ 的概率大. 如果所有项的 $\eta_j > 1$, 则 NAntKp 向贪心爬山算法退化; 或者, 如果所有项的 $\eta_j < 1$, 则 NAntKp 向纯粹随机算法退化, 这两种情况都是我们不希望发生的. 故必须对所有项的 η_j 值进行控制, 使很有希望被选择的一部分项的 $\eta_j \geq 1$, 而其余项的 $\eta_j < 1$.

因此需要选择某项物品 \mathfrak{U} , 使

$$\begin{aligned} \mu_{\mathfrak{U}} &= 1, \\ \eta_j &> 1, \text{ for } j = 0, 1, \dots, \mathfrak{U}-1, \\ \eta_j &< 1, \text{ for } j = \mathfrak{U}+1, \dots, n-1 \end{aligned}$$

成立. 选择物品项 \mathfrak{U} 的一个合理的根据是 $\lambda =$

$$\sum_{i=0}^{m-1} \mu_i b_i / \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} \mu_i r_{ij} \text{ 也就是可用资源的加权和与所有项占用资源的加权和的比. } \lambda \text{ 值部分地反映了可选择项的比例. 按照贪心的原则, 我们希望项 } 0, 1, \dots, \mathfrak{U} \text{ 被选择. 故设 } \mathfrak{U} = n \cdot \lambda. \text{ 综合上述分析, 可以设}$$

$$\eta_j = \begin{cases} \frac{\tilde{\eta}_j}{\tilde{\eta}_{\mathfrak{U}}} \cdot b + a, & j = 0, 1, \dots, \mathfrak{U} \\ \frac{\tilde{\eta}_j}{\tilde{\eta}_{\mathfrak{U}}} \cdot d + c, & j = \mathfrak{U}+1, \dots, n-1 \end{cases} \quad (6)$$

其中, a, b, c, d 是常数.

适当选择 a, b, c, d 的值就可以控制启发式信息 η_j 的值. 由于 $\eta_{\mathfrak{U}}=1$, 所以可固定 $a=1, c+d=1$. 因此, 只需选择 b 和 c 的值就可以得到适当序启发式信息 η_j 的值.

3.3 信息素模型的更新

在每一迭代步 t , 蚁群在构造了候选解之后开始更新信息素模型的值. 首先, 信息素值按式(7)减少, 以忘记以前对待选项作出的不理想处理.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) \quad (7)$$

式中, ρ 是信息素的挥发率. 然后, 一些满足预定义规则的蚂蚁构造的候选解(本文使用价值最大的解)被用于信息素模型的增强. 为此, 蚂蚁计算它的解对应的目标函数值并与别的蚂蚁进行比较. 满足更新规则的蚂蚁被允许按式(8)增加候选解 s^u 对应的信息素值,

$$\begin{cases} \tau_{0j}(t+1) = \tau_{0j}(t) + Q, & s^u(j) = 0 \\ \tau_{1j}(t+1) = \tau_{1j}(t) + Q, & s^u(j) = 1 \end{cases} \quad (8)$$

3.4 局部搜索算法

计算实践证明, 使用简单的局部搜索算法来改善蚁群构造的候选解是相当有效的方法. NAntKp 使每只蚂蚁利用一个简单的局部贪心搜索^[1]算法(LGS)来改善其解的质量, 试图增大解的价值. 记蚂蚁 k 的解为 s^k . 蚂蚁 k 从 $j=0$ 到 $j=n-1$ 逐项检查. 如果 $x_j=1$, LGS 跳过; 如果 $x_j=0$, LGS 先设 $x_j=1$, 然后检查 s^k 是否满足问题的所有约束, 如果满足, 则继续检查下一项; 否则, 仍置 $x_j=0$, 然后继续检查下一项.

3.5 NAntKp 的描述及复杂性分析

设蚂蚁数为 M , 蚂蚁 k 的解为 s^k , 在每个迭代步蚂蚁发现的最优解为 s^{ib} , 最优的解为 s^b , 最大的迭代步数 N_{iter} . 假设已经设定了 NAntKp 算法的最优参数.

算法 2. NAntKp.

//初始化

1. 根据式(6)求项的启发式信息;
2. 按项的启发式信息从大到小排序;
3. for $j=0, 1, \dots, n-1$ do
4. for $k=0, 1, \dots, M-1$ do
5. $s^k(j) \leftarrow 0$;
6. end for
7. $s^b(j) \leftarrow 0$;
8. $\tau_{ij} \leftarrow 1, i=0, 1$;
9. end for
10. for $t=0, 1, \dots, N_{\text{iter}}$ do //迭代

```

11. for  $k=0,1,\dots,M-1$  do //构造候选解
12.   for  $j=0,1,\dots,n-1$  do
13.     蚂蚁  $k$  根据式(2)计算  $p_{ij}^k(t)$ ;
14.     产生一个在  $[0,1]$  区间均匀分布的随机数  $r_0$ ;
15.     if  $p_{ij}^k(t) > r_0$  then
16.       for  $l=0,1,\dots,m-1$  do
17.         if  $\sum_{h=1}^j r_{lh}x_h \leq b_l$  then
18.            $s^k(j) \leftarrow 1$ ;
19.         end if
20.       end for
21.     end if
22.   end for
23. end for
24. for  $k=0,1,\dots,M-1$  do //应用 LGS
25.   for  $j=0,1,\dots,n-1$  do
26.     for  $x_j=0$  且  $l=0,1,\dots,m-1$  do
27.       if  $\sum_{h=1}^n r_{lh}x_h + r_{lj} \leq b_l$  then
28.          $x^k(j) \leftarrow 1$ ;
29.       end if
30.     end for
31.   end for
32. end for
33. 计算  $Obj(s^{sb})$ ;
34. if  $Obj(s^{ib}) > Obj(s^b)$  then
35.    $s^b \leftarrow s^{ib}$ ;
36. end if
37. for  $j=0,1,\dots,n-1$  do //更新信息素值
38.    $\tau_{ij}, i=0,1$  按式(7),(8)更新;
39. end for
40. if 搜索收敛 then //重启算法
41.   将信息素值设置为 1;
42. end if
43. end for

```

在 NAntKp 中,行 1~9 初始化的时间复杂性是 $\max(mn \log n, nM)$;在行 12~22,蚂蚁 k 构造一个候选解,如果 $\sum_{h=1}^j r_{lh}x_h$ 采用增量计算即将以前计算的和缓存,则构造一个解的时间复杂性为 $O(nm)$;在行 24~32,如果 $\sum_{h=1}^n r_{lh}x_h + r_{lj}$ 也采用增量计算,应用 LGS 的时间复杂性为 $O(nmM)$;在行 33~39,找出具有最大目标值的蚂蚁并更新最优解,时间复杂性为 $O(n)$,更新信息素模型的时间复杂性为 $O(n)$;行 40~42 检查搜索是否停止^[12],时间复杂性为 $O(n)$.由于一般有 $<N_{\text{sol}}, M < N_{\text{sol}}$,所以,NAntKp

算法的时间复杂性为 $O(nmN_{\text{sol}})$.信息素模型的空间复杂性为 $O(n)$,存储问题实例的空间复杂性为 $O(nm)$.在串行执行时,蚁群共享存储空间,算法的空间复杂性为 $O(nm)$;而在虚拟并行执行时,每只蚂蚁需要有单独的存储空间,故算法的空间复杂性为 $O(nmM)$.

4 实验研究

所有试验程序使用 Visual C++6.0 生成,并在 Founder PC N243S9 微机上运行,该机使用的操作系统为 Microsoft Windows 2000.所有实验采用的计算实例来自 ORLIB^①.

4.1 实例选择

ORLIB 库有两组 MKP 实例.一组是小规模容易求解的,能被 NAntKp 算法很容易地求解.它们对判断新算法的性能无实际意义,所以不选择它们.另一组是大规模难解的,按 $m-n$ 组合(n 设置为 100,250,500, m 设置为 5,10,30),每组产生 30 问题实例.其中,前 10 个问题的紧度^[1] $\alpha=0.25$,中间 10 个为 0.5,最后 10 个为 0.75.根据现有文献中通常的做法,选择 5~100,5~250,10~100 三组中的 30 个.每组均匀地选择 10 个,其中 $\alpha=0.25$ 的选 4 个,0.5 的选 3 个,0.75 的选 3 个.这样,既可以有效地测试我们的新算法,又不致做过多的实验.

4.2 NAntKp 的参数

在 NAntKp 的定义中,有几个参数需要优化.这些参数分别是蚂蚁数 M ,信息素信息重要性因子 α ,序启发式信息重要性因子 β ,信息素的挥发率 ρ ;控制序启发信息值的参数 b, c .由于没有数学模型可用,所以我们采用大多数文献中使用的实验方法来确定这些参数的最优值.实验表明, $M=5, \alpha=1, \beta=1, \rho=0.01, Q=\rho, b=19, c=0.05, d=1-c$ 是一组较好的参数值.

4.3 串行执行

4.3.1 实验设置

为了确定 NAntKp 算法的求解性能和效率,我们实现了 KpAs, KpArAco, AntKnapsack(见表 1).这些算法的参数按相应文献的说明进行设置. NAntKp 的参数设置为 4.2 节中的实验结果.因为所有的算法都是随机算法,所以实验中每个实例运行每个算法 10 次,每次计算的目标函数次数为 100000 次.

① <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapsack/info.html>

表 1 各算法求最优解的能力比较

算例	最优解	KpAs			KpArAco			AntKnapsack			NAntKp		
		次数	平均解	方差	次数	平均解	方差	次数	平均解	方差	次数	平均解	方差
5.100-00	24381	10	24381.0	0	0	22248.0	197.4	4	24352.6	35.7	10	24381.0	0
5.100-02	23551	1	23539.3	4.11	0	21212.4	180.7	2	23536.0	12.8	4	23543.2	6.7
5.100-04	23991	4	23968.4	20.8	0	21905.9	233.7	1	23950.1	24.0	4	23976.0	12.9
5.100-06	25591	9	25584.0	22.1	0	23075.3	131.3	3	25532.9	44.0	10	25591.0	0
5.100-11	42545	2	42495.8	31.2	0	39980.4	217.9	1	42468.8	27.3	10	42545.0	0
5.100-13	45090	0	45044.1	23.1	0	42392.9	185.2	0	45063.3	22.6	5	45080.2	10.3
5.100-15	42927	10	42927.0	0	0	40228.2	137.6	10	42927.0	0	10	42927.0	0
5.100-22	59802	9	59800.1	6.0	0	58476.2	176.2	3	59771.3	22.2	6	59789.6	18.1
5.100-24	61091	1	61058.6	11.4	0	59542.8	153.4	5	61077.2	21.5	10	61091.0	0
5.100-26	61538	10	61538.0	0	0	60052.2	199.7	4	61512.5	29.4	7	61532.3	9.2
5.250-00	59312	0	59109.0	29.5	0	50843.5	350.8	0	58939.3	91.7	7	59291.3	33.3
5.250-02	62130	0	61903.8	48.9	0	53804.9	640.5	0	61750.5	139.6	8	62117.0	27.4
5.250-04	58951	0	58742.4	25.7	0	51244.0	332.6	0	58604.0	76.5	10	58951.0	0
5.250-06	60414	0	60247.6	46.3	0	52308.0	223.6	0	60066.8	112.1	8	60400.5	28.5
5.250-11	109841	0	109598.8	50.7	0	100431.3	509.6	0	109693.7	47.0	10	109841.0	0
5.250-13	109383	0	109050.3	33.5	0	100126.5	469.0	0	109135.0	90.9	10	109383.0	0
5.250-15	110256	0	109908.4	40.4	0	100917.4	265.1	0	110046.6	98.9	1	110225.7	15.6
5.250-22	149316	0	149114.4	38.6	0	144104.2	371.7	0	149213.5	44.3	1	149289.6	17.4
5.250-24	150353	0	150125.3	31.1	0	145005.9	381.6	0	150301.8	38.0	0	150341.6	1.3
5.250-26	148607	0	148421.9	41.8	0	143030.2	223.2	0	148498.2	46.4	4	148589.4	18.0
10.100-00	23064	0	23014.8	37.7	0	20946.9	138.7	0	23001.5	54.2	1	23051.4	4.4
10.100-02	22131	9	22126.0	15.8	0	20048.8	190.7	3	22046.2	68.4	10	22131.0	0
10.100-04	22751	0	22573.5	37.8	0	20522.6	139.1	0	22575.8	49.7	1	22633.6	44.4
10.100-06	21875	1	21829.9	19.3	0	19614.1	192.2	0	21675.3	120.7	1	21808.2	24.3
10.100-11	42344	0	42205.8	34.1	0	36369.9	209.2	0	42222.7	35.7	10	42344.0	0
10.100-13	45624	0	45492.6	53.2	0	42528.6	213.7	0	45448.4	48.9	0	45557.4	45.7
10.100-15	42995	6	42947.5	65.1	0	40109.2	226.5	1	42872.9	56.6	10	42995.0	0
10.100-22	58391	9	58387.6	10.8	0	56812.3	162.9	1	58335.5	36.8	2	58359.7	17.3
10.100-24	60803	10	60803.0	0	0	58860.3	225.9	8	60801.8	2.5	10	60803.0	0
10.100-26	56377	0	56341.0	8.7	0	54824.4	156.9	5	56340.2	48.4	10	56377.0	0

4.3.2 结 果

实验串行执行时,记录各算法在 10 次运算中求得已知最优解的次数,求得的最优解的平均值及方差,耗用的时间的平均值.

- 各表中表目的意义如下:
- 算例表示实验采用的计算实例的名称;
- 最优解表示发布在网页上的已知最优解;
- 次数表示在 10 次运算中求得已知最优解的次数;
- 平均解表示算法在 10 次运算中求得的最优解的平均值;
- 方差表示算法在 10 次运算中求得的最优解的方差;
- 时间表示算法 10 次耗用的时间的平均值.

(1)新‘解’

NAntKp 求得了实例 5.250-22 的两个新解.这两个新解的目标函数值分别为 149332,149334,均比 ORLIB 库中给出的最大值 149316 大.这说明在这个实例上,我们的新算法比所有已有的方法求解能力强.

(2)求解性能

表 1 说明 NAntKp 在大部分被测试算例上求最优解的能力显著地超过其它 3 个算法. NAntKp 能求得已知最优解的实例数最多,为 28 个,其中的 13 个在每次运行中都能求得已知最优解;KpAs 和 AntKnapsack 其次,但仅为 14 个,其中分别只有 4 个和 1 个实例能求得 10 次最优解;KpArAco 不能求得任何实例的最优解. NAntKp 的求解性能最稳定,大多数方差小于 20,最多仅达 45.7;KpAs 次之,多数方差大于 20,最长达 65.1;AntKnapsack 居中,多数方差在 40 以上,最大者为 139.7;KpArAco 最不稳定,方差最小的高达 131.3,最高的竟达 640.5. NAntKp 能求解 5~250 组几乎所有实例的已知最优解,但其它算法对此无能为力,这说明新算法能求解大规模的问题.

另外,表 1 NAntKp 栏表明,求解实例规模为 10~100,紧度 α 为 0.25 和实例规模为 5~250,紧度 α 为 0.75 时性能严重下降,而在其他情况下算法保持非常好的性能表现.这说明问题规模越大,紧度值越大即可行解空间越大或者约束数过多,紧度值

过小即可行解空间太小都可能会造成性能下降. 出现这种情况的原因是:在可行解空间过大的情况下,算法可能需要更长的搜索时间即需要计算更多解的目标函数的次数;在可行解空间过小的情况下,算法需要花费更多时间寻找不同的可行解.

虽然新算法在大多数实例上表现优越,然而在实例 5.100-22, 5.100-26, 10.100-22 上 NAntKp 的性能虽然比 KpArAco、AntKnapsack 好但比 KpAs 稍差. 这说明在问题规模较小(3 个实例 $n=100$),紧度大($\alpha=0.75$),即在问题比较容易时, NAntKp 略逊于 KpAs.

(3) 求解效率

由于各算法在同一组内的测试算例上耗用的时间基本相同,故将测试算例分为三组: m, n 相同的算例在同一组. 表 2 统计的是每组算例各次运行所耗用时间的平均值的平均(已经圆整到整数).

表 2 说明在同样的计算要求下 NAntKp 耗用的时间明显少于其它算法. AntKnapsack 耗用最多时间,因为它在计算待选项的概率时(算法 1 中的行 5)需要计算信息素信息和启发式信息,而且这两个操作的时间复杂性分别为 $O(n), O(nm)$. KpAs 次之,因为它在计算待选项的概率时只需计算启发式信息,而且信息素值更新的次数少(因为蚂蚁数多),操作相对简单. KpArAco 耗用时间比 KpAs 少数倍,因为它计算待选项的概率时不需计算信息素信息和启发式信息,计算量少,时间复杂性为 $O(n)$. NAntKp 耗用的时间又比 KpArAco 少数倍,因为它只需对某个项的二值变量进行操作,时间复杂性为 $O(1)$. 但 NAntKp 在检查可行性及构造一个解的步数方面与其它算法具有相同的复杂性;其次,由于蚂蚁数最少, NAntKp 更新信息素值的频率比 KpArAco 高很多,且操作稍微更耗时些;另外,每个解要用局部搜索进行改进. 这些就是两者耗时又相对比较接近的原因.

表 2 各算法耗用的平均时间 (单位:s)

算例组	KpAs	KpArAco	AntKnapsack	NAntKp
5.100	272	30	884	12
5.250	1724	190	7762	38
10.100	357	30	940	12

(4) 静态启发搜索与动态启发搜索

从前文可知,KpAs 和 AntKnapsack 在运行中要动态地计算启发式信息的值,而 KpArAco 和 NAntKp 只使用静态启发式信息. 那么,这两种方法有什么不同呢?从表 1 可知,KpAs 和 AntKnapsack

的搜索能力比 KpArAco 强,但比 NAntKp 低很多. 如它们求得最优解的实例数分别为 14,14,0,28. KpAs 和 AntKnapsack 耗用的时间随着 m 的增加明显,如从实例组 5~100 到 10~100,耗用的时间分别从 272s,884s 增加到 357s 和 940s. 相反,KpArAco 和 NAntKp 耗用的时间随变化很小,如从实例组 5~100 到 10~100,耗用的时间变化不超过 1s.

4.4 虚拟并行执行

为了与文献[7]中的算法比较,我们实现了该文中描述的算法. 并将 NAntKp 实现为适宜并行计算. 也就是说,和 KpAcs 一样,每只蚂蚁(slavery)都有自己的 CPU、内存(存储问题实例和信息素信息等),蚂蚁都相同且可相互通信(传递信息素值),系统中有一个“主机”(master)以决定哪只蚂蚁最好.

4.4.1 实验设置

KpAcs 的参数设置与文献[7]相同,即蚂蚁数(CPU)为 30. NAntKp 的蚂蚁数为 5,其他参数同 4.3 节. 两个算法的代码都在 Founder PC N243S9 机上虚拟执行. 对每个实例运行 10 次,每次允许计算的目标函数次数都为 100000.

4.4.2 结果

(1) 求最优解的性能

NAntKp 的最优解的结果与表 1 的 NAntKp 列相同. 比较表 3 和表 1 可知,NAntKp 求最优解的表现比 KpAcs 出色. NAntKp 能求到 28 个实例的最优解,其中 13 个每次运行都能求得;而 KpAcs 只能求得 18 个实例,其中只有 6 个每次运行都能求得最优解. NAntKp 的表现更稳定,例如它的方差最大为 45.7,而 KpAcs 最大为 136.7. NAntKp 能求解的实例规模更大, KpAcs 不能求得实例组 5~250 的任何已知最优解.

(2) 求解效率

对比表 3 的时间列可知,KpAcs 耗用的时间是 NAntKp 的 3 倍. 需要注意的是,NAntKp 占用的计算资源少(5 只蚂蚁). 这说明 NAntKp 的求解效率高很多.

表 3 并行求解的比较

算例	最优解	KpAcs				NAntKp 时间/s
		次数	平均解	方差	时间/s	
5.100-00	24381	10	24381.0	0	10.2	2.7
5.100-02	23551	3	23540.7	7.3	10.1	2.7
5.100-04	23991	2	23953.0	22.2	10.3	3.0
5.100-06	25591	7	25570.0	33.8	10.2	2.7
5.100-11	42545	0	42467.2	5.2	10.1	2.8
5.100-13	45090	0	45070.8	0.4	10.1	2.8
5.100-15	42927	10	42927.0	0	10.2	2.8

(续 表)

算例	最优解	KpAcs				NAntKp 时间/s
		次数	平均解	方差	时间/s	
5.100-22	59802	3	59774.3	19.2	10.4	2.8
5.100-24	61091	9	61089.8	3.8	10.3	2.8
5.100-26	61538	10	61538.0	0	10.3	2.9
5.250-00	59312	0	58634.9	100.0	70.7	16.4
5.250-02	62130	0	61473.0	43.0	69.3	13.5
5.250-04	58951	0	57208.6	136.7	68.6	13.6
5.250-06	60414	0	60009.4	41.6	68.1	13.9
5.250-11	109841	0	109271.6	54.5	69	16.3
5.250-13	109383	0	108634.1	62.6	71	17.8
5.250-15	110256	0	109245.0	118.7	69.3	18.4
5.250-22	149316	0	149106.5	47.8	69.9	17.1
5.250-24	150353	0	149997.4	31.2	70.2	16.2
5.250-26	148607	0	148355.6	40.7	71.2	15.3
10.100-00	23064	3	23058.2	4.5	10.1	2.9
10.100-02	22131	9	22126.0	15.8	10.3	2.9
10.100-04	22751	2	22686.8	48.1	10.4	2.9
10.100-06	21875	3	21853.2	20.2	10.5	3.0
10.100-11	42344	10	42344.0	0	10.5	3.1
10.100-13	45624	1	45556.6	52.3	10.5	3.3
10.100-15	42995	10	42995.0	0	10.5	3.2
10.100-22	58391	1	58357.7	11.7	10.2	3.2
10.100-24	60803	10	60803.0	0	10.1	3.1
10.100-26	56377	9	56374.6	7.6	10.3	3.2

5 结 论

本文在文献[12]定义的信息素模型(未见利用此模型的文献)的基础上,提出了一种新的选择概率计算规则.在这种新规则中,我们定义了一种全新的序启发式信息.使用这种规则的 NAntKp 算法,构造一个解的时间复杂性为 $O(n \cdot m)$,整个算法的复杂性比现有文献中利用 ACO 求解 MKP 的所有算法都要低,而求最优解的能力更强.令人鼓舞的是,本文中的新算法在较短的时间内求出了 ORLIB 中的算例 5.250-22 的两个新解.这两个新解的目标函数值明显比现在已知的最优解的值大.

NAntKp 算法的良好性能促使我们对其做进一步的研究.研究的可能方向之一是将该算法用于求解别的子集问题;之二是对该算法进行建模,分析信息素模型的不平衡性与性能的相关性.

致 谢 感谢审稿专家对本文提出的具有很高价值的审稿意见,感谢 Leguizamón 的论文,感谢哈尔滨工业大学人工智能研究室的王海晶等人在实验程序设计及论文写作过程中对作者的帮助,感谢谢海丹对本文的标点及语法进行的校对!

参 考 文 献

[1] Chu P, Beasley J. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics, 1998, 4(1): 63-86

[2] Dorigo M, Maniezzo V, Colnari A. The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics, Part B, 1996, 26(1): 29-42

[3] Dorigo M, Di Caro G. The ant colony optimization metaheuristic//New Ideas in Optimization. London, UK: McGraw-Hill, 1999: 11-32

[4] Leguizamón G, Michalewicz Z. A new version of ant system for subset problems//Proceedings of the Congress on Evolutionary Computation. Piscataway, NJ, 1999: 1459-1464

[5] Fidanova S. Evolutionary algorithm for multiple knapsack problem//Proceedings of PPSN VII Workshops. Granada, Spain, 2002: 831-840

[6] Dorigo M, Gambadella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 53-67

[7] Parra-Hernandez R, Dimopoulos N. On the performance of the ant colony system for solving the multidimensional knapsack problem//Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM, 2003, Piscataway, NJ: IEEE Press, 2003: 338-341

[8] Alaya I, Solnon C, Ghédira K. Ant algorithm for the multidimensional knapsack problem//Proceedings of the Dans International Conference on Bioinspired Optimization Methods and Their Applications. Ljubljana, Slovenia, 2004: 63-72

[9] Yu Yong-Xin, Zhang Xin-Rong. Optimization algorithm for multiple-choice knapsack problem based on ant colony system. Computer Engineering, 2003, 29(20): 75-76 (in Chinese)

(于永新, 张兴荣. 基于蚁群系统的多选择背包问题优化算法. 计算机工程, 2003, 29(20): 75-76)

[10] Liu Hua-Ying, Lin Yu-E, Liu Jin-Yue. Solution to the 0/1 knapsack problem based on ant algorithm. Journal of Daqing Petroleum Institute, 2005, 29(3): 59-62 (in Chinese)

(刘华, 林玉娥, 刘金月. 基于蚁群算法求解 0/1 背包问题. 大庆石油学院学报, 2005, 29(3): 59-62)

[11] Luo Xiao-Hu, Zhao Lei. Ant colony algorithm for 0/1 knapsack problem. Journal of Suzhou Institute of Silk Textile Technology, 2004, 24(1): 41-44 (in Chinese)

(罗小虎, 赵雷. 一个解决 0/1 背包问题的蚁群算法. 苏州大学学报(工科版), 2004, 24(1): 41-44)

[12] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man and Cybernetics, Part B, 2004, 34(2): 1161-1172

[13] Stutzle T, Hoos H H. Max-min ant system. Future Generation Computer Systems, 2000, 16(8): 889-914



YU Xue-Cai, born in 1975, Ph. D. candidate. His research interests include the basic theory and the application of ant colony optimization.

ZHANG Tian-Wen, born in 1940, professor, Ph. D. supervisor. His research interests include active artificial intelligence, active vision, video code and retrieve, virtual reality, artificial life.

Background

There still exists many other complex calculations which computers can not solve in reasonable CPU time limits. Examples are large-scale instances of the travel salesman problem, the multidimensional knapsack problem, the job shop problem, and so on. Thus various heuristic and metaheuristic approaches have been suggested for solving discrete optimization problems. In general, metaheuristic performs over heuristic. Ant colony optimization is a metaheuristic. The authors mainly bend themselves to theoretical foundation and application of ant colony optimization. The results will aid to deepen understanding of ant colony optimization algorithm, to improve the performance of ant colony optimization and to extend the area of applying ant colony optimization approach. Ant colony optimization has been extensively applied to many

combinatorial optimization problems. Many experimental results show that ant colony optimization is powerful. However, all ant colony optimization algorithms for solving the multidimensional knapsack problem perform not very good. This work designed a new rule selecting a knapsack item and defined a new heuristic information which can be easily incorporated into the new solution constructing rule. The new ant colony optimization algorithm provided in this paper greatly enhances the capability of ant colony optimization solving the multidimensional knapsack problem. The approach described in this paper can aid to develop ant colony optimization algorithms for other subset problems such as the cardinality tree problem and the multi-demand multidimensional knapsack problem.