

求解0-1背包问题的改进二进制捕鱼算法

陈建荣

(右江民族医学院 公共卫生与管理学院,广西 百色 533000)

摘要:经典群智能算法在求解0-1背包问题时普遍存在全局搜索能力不强、求解精度不高、收敛速度慢等缺点。针对这一情况,将二进制编码引入捕鱼算法中,提出二进制捕鱼算法。在此基础上,结合算法本身的特点,添加靠近搜索方法,改善渔夫之间的协作效果;借鉴贪心算法和轮盘赌的思想,设计贪心轮盘赌策略,并结合随机比例参数来改善算法初值;同时引入自适应半径系数来解决步长参数设置的问题,进而提出了一种改进二进制捕鱼算法。实验与对比部分对15个0-1背包问题进行求解测试,结果表明,对于常用算例而言,与其它群智能算法相比,改进二进制捕鱼算法能找到全部问题的最优解,且在总体性能上看较优;对于100维及以上的高维背包问题而言,改进算法在求解精度、稳定性、收敛速度、运行耗时等方面均具有明显优势。因此,将改进二进制捕鱼算法应用于求解0-1背包问题是有效的和可行的。

关键词:捕鱼算法;0-1背包问题;贪心算法;群智能;二进制

中图分类号:TP301.6

文献标识码:A

文章编号:1673-629X(2023)05-0187-07

doi:10.3969/j.issn.1673-629X.2023.05.028

An Improved Binary Fishing Algorithm for 0-1 Knapsack Problem

CHEN Jian-rong

(School of Public Health and Management, Youjiang Medical University for Nationalities, Baise 533000, China)

Abstract: In solving 0-1 knapsack problem, the classical swarm intelligence algorithm has some shortcomings, such as weak global search ability, low solution accuracy and slow convergence speed. In view of this situation, the binary coding is introduced into the fishing algorithm, and the binary fishing algorithm is proposed. On this basis, combined with the characteristics of the algorithm itself, the approach search method is added to improve the cooperation effect among fishermen. Drawing on the idea of greedy algorithm and roulette, the greedy roulette strategy is designed, and the initial value of the algorithm is improved by combining the random proportion parameter. At the same time, the adaptive radius coefficient is introduced to solve the problem of step parameter setting, and then an improved binary fishing algorithm is proposed. In the experiment and comparison section, 15 0-1 knapsack problems are solved and tested. The results show that for common examples, compared with other swarm intelligence algorithms, the improved binary fishing algorithm can find the optimal solutions to all problems, and the overall performance is better. For high-dimensional knapsack problems of 100 dimensions and above, the improved algorithm has obvious advantages in solving accuracy, stability, convergence speed and running time. Therefore, it is effective and feasible to apply the modified binary fishing algorithm to solve the 0-1 knapsack problem.

Key words: fishing algorithm; 0-1 knapsack problem; greedy algorithm; swarm intelligence; binary

0 引言

背包问题 (Knapsack Problem, KP)^[1-3] 是经典的 NP 难问题,其目标是寻找在给定背包容量的限制条件下价值量最大的物品选择方案。当涉及资源或任务分配等工作时,可使用背包问题的相关理论去处理和解决,如投资决策等。背包问题有多个变体,最基本的是 0-1 背包问题,其它类型问题能转换成该问题而顺利求解^[3]。

求解 0-1 背包问题的方法包括精确算法和启发式算法这两类^[2-3]。其中,精确算法包括贪心算法、动态规划法、分支定界法、穷举法、回溯法等。这些算法普遍存在计算效率低、计算量随问题维数的增加而呈指数级增长等缺点,因而难以有效处理高维背包问题。另一类算法是启发式算法,其本质上属于近似搜索算法,能在合理时间内找到问题的最优解或者满意解。常见的有遗传算法^[2]、模拟退火算法^[4]、粒子群算

收稿日期:2022-07-26

修回日期:2022-11-29

基金项目:国家自然科学基金面上项目(61976230);2022年度广西高等教育本科教学改革工程项目(2022JGA292);2020年右江民族医学院校级科研课题(yj2020gcky037)

作者简介:陈建荣(1982-),男,副教授(内聘教授),博士,硕导,研究方向为人工智能、神经网络、大数据、机器人等。

法^[5]、蚁群算法^[6]、布谷鸟算法^[7]、蝙蝠算法^[8]等。但这些算法在求解 0-1 背包问题时,仍存在全局搜索能力不强、收敛速度慢等不足。

陈建荣等人^[9]通过观察渔夫在江面上捕鱼的行为习惯而提出了捕鱼算法(Fishing Algorithm)。目前,该算法及其改进或结合算法已被应用于解决各类优化问题,并获得良好的效果^[10-13]。

捕鱼算法是针对连续问题提出的,无法直接应用于求解 0-1 背包问题,所以该文首先对渔夫编码及搜索方法进行重新定义和描述,称之为二进制捕鱼算法(Binary Fishing Algorithm, BFA)。然后,在对其分析的基础上,提出改进二进制捕鱼算法(Improved Binary Fishing Algorithm, IBFA)。最后,使用不同维度的背包问题对算法进行性能测试。

1 捕鱼算法

捕鱼算法对渔夫捕鱼行为的模拟主要包括 7 个假设和 3 类搜索方法^[12]。7 个假设分别是:(1)水中鱼的分布保持不变;(2)渔夫不知道鱼在水里的分布情况;(3)渔夫总是向鱼多的方向前进;(4)渔夫倾向于停留在鱼密度大的位置捕鱼;(5)渔夫希望通过使用网眼更小的渔网将鱼打尽;(6)渔夫会离开没有鱼的位置前往其它地方捕鱼;(7)渔夫之间避免碰撞。3 类搜索方法包括移动搜索、收缩搜索和随机搜索。

算法运行过程中,渔夫在给定的寻优区域(问题的定义域)内撒网捕鱼,并通过渔夫之间的通力合作最终找到水中鱼密度最高的位置(问题的最优解)。

2 问题描述及二进制捕鱼算法

2.1 问题描述

0-1 背包问题可描述为:给定一个最大容量为 C 的背包和 n 个物品,第 k 个物品的价值和体积分别为 p_k 和 w_k 。在不超过背包最大容量的条件下,将物品尽可能地装进背包,使得背包中物品的总价值达到最大。

若假定 x_k 的取值 0 和 1 分别表示第 k 个物品没有装入背包和装入背包两种状态,则该问题用数学模型可表示为:

$$\max f(x) = \sum_{k=1}^n p_k x_k$$

$$\text{s. t. } \sum_{k=1}^n w_k x_k < C$$

其中, $x_k \in \{0, 1\}$ 。

引入罚函数法,可将上述问题转化为:

$$\max f(x) = \sum_{k=1}^n p_k x_k - \vartheta M \quad (1)$$

其中, ϑ 是罚因子, $M = \max(0, \sum_{k=1}^n w_k x_k - C), \max(\cdot, \cdot)$

函数返回两个输入值中的较大值。文中的罚因子统一取 10^3 。

2.2 渔夫编码及相关定义

对于 n 维的 0-1 背包问题,第 i 个渔夫的位置向量表示为 $X_i = (x_{i1}, \dots, x_{ik}, \dots, x_{in})$ 。其中, x_{ik} 是 X_i 的第 k 个分量,取值为 0 或 1。该渔夫对应位置的目标函数值可通过(1)式求得。

定义 1: 设两个位置向量分别为 X_{i_1} 和 X_{i_2} , 则它们之间的距离定义为 $D = \text{sum}(|X_{i_1} - X_{i_2}|)$ 。其中, $|\cdot|$ 表示对向量中的每一个分量取绝对值, $\text{sum}(\cdot)$ 表示将向量的所有分量相加。

例 1: 给定位置 $X_1 = (1, 0, 1, 0)$ 和 $X_2 = (0, 1, 1, 0)$, 那么两位置间的距离 $D = |1 - 0| + |0 - 1| + |1 - 1| + |0 - 0| = 2$ 。

定义 2: 设渔夫的位置向量和撒网半径分别为 X_i 和 L , 其中 $L \leq n$ 。从 X_i 的 n 个分量中随机选取 L 个, 并对这 L 个分量都用其二进制反码替换, 得到撒网点的位位置向量为 X'_i , 则称渔夫 X_i 以半径 L 随机撒网一次。

例 2: 给定渔夫位置 $X_1 = (1, 0, 1, 1, 0)$ 和撒网半径 $L = 2$ 。若随机选取的 L 个位置为 2 和 3, 那么对应的撒网点为 $X'_1 = (1, 1, 0, 1, 0)$ 。若位置为 1 和 4, 则 $X'_1 = (0, 0, 1, 0, 0)$ 。

由定义可知, 当渔夫撒网时, 其所在位置和撒网点之间的距离刚好等于撒网半径。

2.3 搜索方法

设渔夫 i 的当前位置为 $X_i^0 = (x_{i1}^0, \dots, x_{ik}^0, \dots, x_{in}^0)$, 则其以半径 L_i 撒网 m 次所得的撒网点集为:

$$\Psi_i = \{X_i^l = (x_{i1}^l, \dots, x_{ik}^l, \dots, x_{in}^l) \mid l = 1, 2, \dots, m\} \quad (2)$$

其中, 每个 X_i^l 均由渔夫 X_i^0 以半径 L_i 随机撒网一次得到。 $x_{ik}^0, x_{ik}^l \in \{0, 1\}$, 撒网半径 L_i 和撒网次数 m 均为给定的正整数。

下面给出三类搜索方法的具体描述。

(1) 移动搜索。

若 $f(X^*) = \max_{X_i^l \in \Psi_i} f(X_i^l) > f(X_i^0)$, 则渔夫 i 从 X_i^0 移动到 X^* , 并按式(2)重新构造撒网点集。

(2) 收缩搜索。

若 $\max_{X_i^l \in \Psi_i} f(X_i^l) \leq f(X_i^0)$, 则渔夫 i 在当前位置 X_i^0 按式(2)以半径 $L_i' = \lceil \beta L_i \rceil$ 撒网 m 次构造新的撒网点集。($\lceil \cdot \rceil$ 表示向下取整, 且规定其返回的最小值为 1; $\beta \in (0, 1]$)

(3) 随机搜索。

若渔夫 i 不满足前两类搜索的执行条件, 且连续执行收缩搜索的次数达到算法给定的阈值, 则对该渔

夫进行随机初始化,并按式(2)构造撒网点集。

2.4 算法流程

算法设置有公告板,输入参数为:渔夫个体数量 N 、撒网半径 L 、撒网次数 ξ 、收缩系数 β 、收缩搜索阈值 η 和迭代次数 T 。

算法流程:

步骤1:对渔夫进行随机初始化;

步骤2:算法迭代次数达到 T (或找到已知最优解)则停机,并输出最优值和最优解;

步骤3:各渔夫根据当前情况,选择执行相应的搜索方法(移动搜索、收缩搜索和随机搜索);

步骤4:找到更优值则更新公告板;转 Step 2。

3 改进二进制捕鱼算法

3.1 分析与说明

首先,在 BFA 算法中,使用随机函数生成并给渔夫个体位置赋初值,由于受到随机性的影响可能会使得算法收敛速度偏慢。其次,渔夫的撒网半径对算法收敛速度影响较大,而所求解问题的维数不尽相同,若撒网半径设置为固定值,则会出现对于不同问题需要频繁设置不同半径值的尴尬局面。最后,为提高渔夫之间的信息共享,避免陷入局部最优,增加了一种名为靠近搜索的搜索方法。

3.2 贪心轮盘赌

引入贪心算法^[14]和轮盘赌^[15]的思想对算法进行改进。先求出给定物品的价值密度,第 k 个物品的价值密度为 $\zeta_k = p_k/w_k$ 。然后,为能方便地调整物品对应的选中概率的大小,即轮盘赌中该物品对应的选中区间宽度,这里对价值密度求幂,即 ζ_k^θ 。其中,实数 θ 的值依据实际情况设置。最后,根据 ζ_k^θ 的值采用轮盘赌的方式选择装入背包的物品,直至背包装不进下一个物品为止。

3.3 自适应半径

通过下式给出初始半径值:

$$R = \varepsilon n \quad (3)$$

其中, $\varepsilon \in (0,1]$ 为给定的自适应半径系数; n 是所求问题的维数,即0-1背包问题中物品的总数。

3.4 随机比例

为避免因出现早熟而陷入局部最优,算法设置有随机比例参数 Y 用于控制采用随机函数进行初始化渔夫位置的比例,其取值范围是 $[0,1]$ 。当 $Y = 0$ 时,全部采用随机函数进行初始化;当 $Y = 1$ 时,则全部使用贪心轮盘赌的方法对渔夫位置进行初始化。

3.5 靠近搜索

定义3:设渔夫位置和目标位置分别为 X_i 和 X^G ,

且 X_i 与 X^G 之间的距离 $D > R$ 。从 $|X_i - X^G|$ 中随机选出 R 个非零分量,并将 X_i 中位于该非零分量位置的 R 个分量值用其二进制反码替换,则称渔夫 X_i 以步长 R 向位置 X^G 随机靠近一次。

例3:设 $R = 2$ 、 $X_i = (1,0,1,0,1)$ 和 $X^G = (0,0,0,1,1)$ 。由 $|X_i - X^G| = (1,0,1,1,0)$ 可得非零分量的位置为1、3和4。若随机选出的位置为3和4,则渔夫的新位置为 $X'_i = (1,0,0,1,1)$ 。此时,称渔夫 X_i 以步长2向位置 X^G 随机靠近一次。

靠近搜索可描述为:若渔夫 i 连续执行收缩搜索的次数达到算法给定的阈值 η ,且其当前所处的位置并非群体最优,则该渔夫以步长 $\lfloor R \times \text{rand} \rfloor$ 向群体最优位置靠近($\lfloor \cdot \rfloor$ 表示向上取整);渔夫每向群体最优位置随机靠近一次,都重新按式(2)构造撒网点集。

3.6 算法流程

与 BFA 算法相比,IBFA 算法取消了撒网半径 L ,新增自适应半径系数 ε 和随机比例参数 Y ,其它参数保持不变。IBFA 算法使用自适应半径和随机比例对渔夫进行初始化,并在搜索方法中增加了靠近搜索,其余操作和流程均与原算法相同。

4 实验与对比

4.1 软硬件环境与说明

实验电脑是华为 MateBook X Pro 超极本,配置为: Intel Core i7-1165G7 CPU @ 2.8 GHz, 16 GB LPDDR4X 内存;64 位 Windows 11 家庭版操作系统, MATLAB 版本是 2020a。

实验分为常用算例测试和高维算例测试两部分。除另有说明外,各算法均使用固定的参数,并且连续运行一定次数,以降低随机性对结果的影响,同时也能在一定程度上反映不同算法在稳定性方面的差异。

4.2 常用算例测试与结果对比

常用算例测试中的九个经典算例均来自参考文献[3],各算法运行参数见表1。测试时,所有算法都连续运行30次,并记录包括最优值、最差值、平均值、标准差等指标在内的数据以便后续对比。七种不同算法的测试结果见表2。其中,对比算法的运行结果均来自文献[3]。

BFA 与 IBFA 算法测试结果对比:从表2中的数据可以看出,IBFA 算法能找到全部九个背包问题的最优解,且标准差均为零,说明算法适应性强、稳定性好;而 BFA 算法只能找到前三个问题的最优解,这说明对于维数相对较高的问题(KP4-KP9)来说,IBFA 算法的性能有较明显的改善。

IBFA 算法与其它算法测试结果对比:从最差值指标可知,对比算法中的 ACP SO、BBA 和 HBA 算法,在

表 1 各算法运行参数设置

算法名称	参数设置
二进制捕鱼算法(BFA)	渔夫数 $N=20$ 、撒网半径 $L=10$ 、撒网次数 $\xi=15$ 、收缩系数 $\beta=0.8$ 、收缩搜索阈值 $\eta=15$ 、迭代次数 $T=1\ 000$
改进二进制捕鱼算法(IBFA)	贪心轮盘赌系数 $\theta=16$ 、自适应半径 $\varepsilon=0.5$ 、随机比例 $\gamma=0.9$;其余参数值同 BFA
离散粒子群算法(DPSO)	粒子数 50、惯性因子 0.729、学习因子 1.5、迭代次数 500
自适应元胞粒子群算法(ACPSO)	惯性因子 0.8、学习因子 2、被选率 0.5(KP7 中为 0.009);其余参数值同 DPSO
二进制蝙蝠算法(BBA)	蝙蝠数 50、音量 0.25、脉冲发生率 0.5、音量衰减系数 0.9、脉冲发生率系数 0.9、迭代次数 500
混合蝙蝠算法(HBA)	追随率 0.5、反置率 0.2;其余参数值同 BBA
二进制狮群算法(BLSO)	狮子数 50、母狮比例 0.1、最小贪婪度 2、最大贪婪度 12、淘汰率 0.2、迭代次数 500

表 2 七种不同算法的测试结果对比

算例	维数	已知最优解	算法名称	最优值	最差值	平均值	标准差	最小迭代次数	最大迭代次数	平均迭代次数	运行时间/s
KP1	10	295	BFA	295	295	295.00	0	0	8	4.37	0.002
			IBFA	295	295	295.00	0	0	5	4.30	0.002
			DPSO	295	295	295.00	0	0	2	0.10	0.110
			ACPSO	295	295	295.00	—	1	6	1.80	0.190
			BBA	295	294	294.97	0.18	0	14	0.48	2.750
			HBA	295	295	295.00	—	1	1	1.00	0.630
			BLSO	295	295	295.00	0	0	2	0.10	0.130
KP2	20	1 024	BFA	1 024	1 024	1 024.00	0	8	37	13.30	0.007
			IBFA	1 024	1 024	1 024.00	0	0	24	8.50	0.005
			DPSO	1 024	1 024	1 024.00	0	0	3	0.50	0.180
			ACPSO	1 024	1 018	1 021.20	—	1	441	194.19	50.44
			BBA	1 024	1 024	1 024.00	0	0	21	1.67	0.360
			HBA	1 024	1 024	1 024.00	—	1	4	1.43	0.620
			BLSO	1 024	1 024	1 024.00	0	0	2	0.40	0.220
KP3	20	1 042	BFA	1 042	1 042	1 042.00	0	8	69	23.50	0.015
			IBFA	1 042	1 042	1 042.00	0	6	38	11.97	0.007
			DPSO	1 042	1 042	1 042.00	0	0	29	6.10	1.020
			ACPSO	1 042	1 037	1 039.30	—	49	474	235.00	56.450
			BBA	1 042	1 037	1 040.00	2.45	0	32	8.44	31.730
			HBA	1 042	1 042	1 042.00	—	1	142	27.23	3.290
			BLSO	1 042	1 042	1 042.00	0	0	33	7.00	2.000
KP4	50	4 882	BFA	4 834	4 704	4 789.10	925.06	17	998	465.97	0.622
			IBFA	4 882	4 882	4 882.00	0	10	39	12.50	0.009
			DPSO	4 882	4 882	4 882.00	0	3	16	8.63	1.820
			ACPSO	4 882	4 834	4 857.10	—	91	421	133.93	131.900
			BBA	4 882	4 882	4 882.00	0	1	6	2.37	0.600
			HBA	4 882	4 882	4 882.00	—	1	39	6.73	1.510
			BLSO	4 882	4 882	4 882.00	0	1	6	2.70	2.230

续表 2

算例	维数	已知最优解	算法名称	最优值	最差值	平均值	标准差	最小迭代次数	最大迭代次数	平均迭代次数	运行时间/s
KP5	100	15 170	BFA	14 911	14 650	14 772.33	4 631.40	31	961	476.77	0.665
			IBFA	15 170	15 170	15 170.00	0	0	0	0	0
			DPSO	15 170	15 170	15 170.00	0	4	46	19.43	3.770
			ACPSO	15 170	15 170	15 170.00	—	1	1	1.00	0.940
			BBA	15 170	15 170	15 170.00	0	1	2	1.07	0.360
			HBA	15 170	15 170	15 170.00	—	1	2	1.83	0.910
			BLSO	15 170	15 170	15 170.00	0	1	4	2.07	2.070
KP6	100	26 559	BFA	25 951	25 385	25 627.37	16 530.03	39	970	529.13	0.668
			IBFA	26 559	26 559	26 559.00	0	14	65	21.87	0.017
			DPSO	26 559	26 559	26 559.00	0	13	129	48.27	9.580
			ACPSO	26 559	26 525	26 527.00	—	45	45	45.00	367.520
			BBA	26 559	26 559	26 559.00	0	1	98	17.37	3.670
			HBA	26 559	26 534	26 551.00	—	2	103	10.43	35.470
			BLSO	26 559	26 559	26 559.00	0	2	14	5.07	5.100
KP7	100	2 660	BFA	2 452	2 271	2 352.30	2 002.56	31	973	532.97	0.718
			IBFA	2 660	2 660	2 660.00	0	0	0	0	0
			DPSO	2 660	2 660	2 660.00	0	5	28	14.47	3.900
			ACPSO	2 660	2 660	2 660.00	—	20	417	190.38	203.250
			BBA	2 660	2 660	2 660.00	0	1	4	1.77	0.650
			HBA	2 660	2 660	2 660.00	—	1	5	2.47	1.270
			BLSO	2 660	2 660	2 660.00	0	1	6	3.03	3.480
KP8	100	4 143	BFA	3 974	3 856	3 909.60	951.83	65	965	541.23	0.795
			IBFA	4 143	4 143	4 143.00	0	0	82	20.60	0.015
			DPSO	4 143	4 143	4 143.00	0	13	72	36.50	8.330
			ACPSO	4 143	4 143	4 143.00	—	4	162	58.83	37.730
			BBA	4 143	4 141	4 142.93	0.36	2	365	33.86	11.540
			HBA	4 143	4 143	4 143.00	—	3	9	4.57	1.450
			BLSO	4 143	4 143	4 143.00	0	2	7	4.43	5.920
KP9	100	4 987	BFA	4 920	4 852	4 889.43	348.94	35	970	535.10	0.708
			IBFA	4 987	4 987	4 987.00	0	0	286	54.93	0.039
			DPSO	4 987	4 987	4 987.00	0	16	78	36.33	6.670
			ACPSO	4 986	4 986	4 986.00	—	1	1	1.00	0.800
			BBA	4 987	4 987	4 987.00	0	1	60	9.03	1.940
			HBA	4 987	4 986	4 986.50	—	2	33	4.73	31.170
			BLSO	4 987	4 987	4 987.00	0	1	24	3.97	3.440

最差的情况下,未能找到全部九个背包问题(KP1—KP9)的最优解。对于IBFA、DPSPO和BLSO这三个能找到九个问题最优解的算法来说,它们的测试结果数据各有优势。值得注意的是,从100维的五个问题(KP5—KP9)的测试结果来看,除KP6外,IBFA在最小迭代次数指标上是最优的,均优于其它对比算法;特别

是对于KP5和KP7这两个问题,在连续的30次运行中,IBFA算法在每一次初始化时都能找到问题的最优解,其对比指标均优于其它算法,这说明本文的改进方法在获得优秀初始值方面有较好的效果。注意到,IBFA算法的种群数量只有20,而其它对比算法均为50,这表明该算法的全局搜索能力较强,即使在种群数

量较少的情况下,也不容易陷入局部极值点。此外,IBFA 在求解背包问题时的运行耗时非常短,只需要不足 0.04 秒的时间就能找到问题的最优解。

4.3 高维算例测试与结果对比

为进一步验证算法性能,本节进行了高维算例的测试,这些算例使用下面的公式随机产生^[7]:

$$w_i = \text{randint}[1, 10], p_i = w_i + 5, C = 0.5 \times \sum_{i=1}^m w_i$$

其中, w_i 、 p_i 和 C 分别表示算例中的物品体积、价值和

背包容量; $\text{randint}[1, 10]$ 表示随机地从集合 $\{1, 2, \dots, 10\}$ 中抽取一个整数。根据上述公式,随机产生维数为 100、200、400、600、800 和 1 000 的六个高维 0-1 背包问题,每个算例产生后就保持不变。测试时,算法均连续运行 50 次,最大迭代次数均设置为 500 代。BBA 算法参数取表 1 中的对应值;BFA 和 IBFA 算法除 $\eta = 20$ 和 $\theta = 70$ 外,其余参数均与表 1 保持一致。测试结果见表 3。

表 3 高维算例测试结果

算例	维数	算法名称	最优值	最差值	平均值	标准差	最小迭代次数	最大迭代次数	平均迭代次数	运行时间/s
KP01	100	BBA	572	556	564.82	14.84	43	492	311.50	0.258
		BFA	572	557	562.54	14.46	13	496	239.60	0.321
		IBFA	602	602	602.00	0	11	13	12.08	0.352
KP02	200	BBA	1 126	1 105	1 114.86	27.55	18	494	291.26	0.468
		BFA	1 132	1 102	1 113.30	40.62	13	493	221.48	0.363
		IBFA	1 227	1 227	1 227.00	0	14	16	15.20	0.415
KP03	400	BBA	2 255	2 213	2 236.64	100.64	73	490	311.44	0.942
		BFA	2 256	2 211	2 226.40	100.86	12	492	264.14	0.436
		IBFA	2 486	2 486	2 486.00	0	17	19	17.18	0.549
KP04	600	BBA	3 345	3 283	3 308.56	188.74	13	495	319.48	1.437
		BFA	3 322	3 267	3 289.70	152.26	16	471	192.16	0.477
		IBFA	3 727	3 727	3 727.00	0	19	21	19.04	0.693
KP05	800	BBA	4 362	4 300	4 328.08	238.52	13	491	279.28	1.871
		BFA	4 343	4 273	4 306.10	210.09	20	478	228.94	0.521
		IBFA	4 953	4 953	4 953.00	0	20	23	20.38	0.793
KP06	1 000	BBA	5 476	5 393	5 439.44	388.82	13	499	307.54	2.346
		BFA	5 452	5 382	5 409.20	340.98	16	493	184.22	0.578
		IBFA	6 162	6 162	6 162.00	0	21	23	21.26	0.965

从表 3 中的数据可以看到,对于这六个高维背包问题而言,IBFA 算法能找到的解是最优的,且其标准差均为零,说明该算法性能稳定,鲁棒性强,不易陷入局部极值。

从各项对比指标上看,BBA 和 BFA 算法性能基本相当,且这两种算法均因陷入局部极值而未能找到全局最优解。此外,随着问题维数的提高,各对比算法的运行耗时都相应地增加了:BBA 算法耗时增加最快,其次是 IBFA,最后是 BFA。从最小迭代次数、最大迭代次数和平均迭代次数上看,问题维数的变化(提高)对 IBFA 算法收敛速度的影响不大,对于这六个高维问题,该算法最多只需要 23 次迭代就能找到最优解。

图 1 至图 6 展示的是 BBA、BFA 和 IBFA 算法连

续运行 50 次的平均进化曲线。从图中可以看出,IBFA 算法的收敛速度很快,仅需要 20 次左右的迭代就能收敛到最优值,而 BBA 和 BFA 算法经历 500 次迭代后仍未能收敛到稳定值。

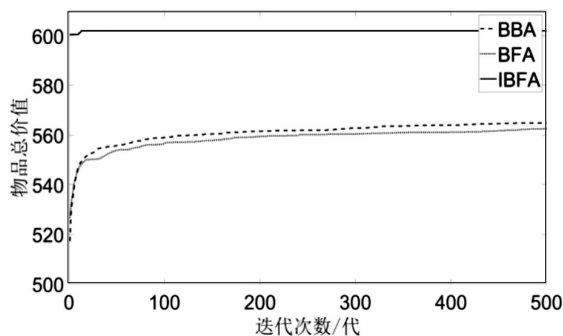


图 1 KP01(100 维)问题平均进化曲线

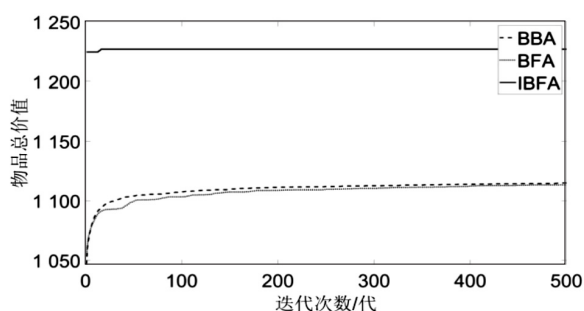


图2 KP02(200维)问题平均进化曲线

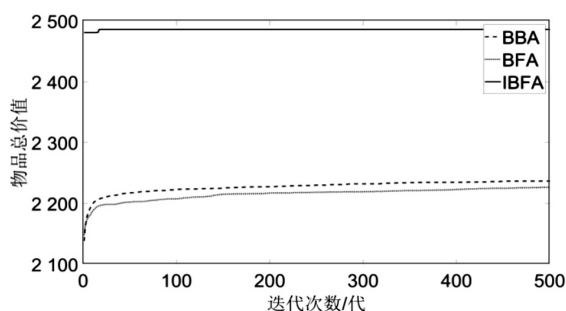


图3 KP03(400维)问题平均进化曲线

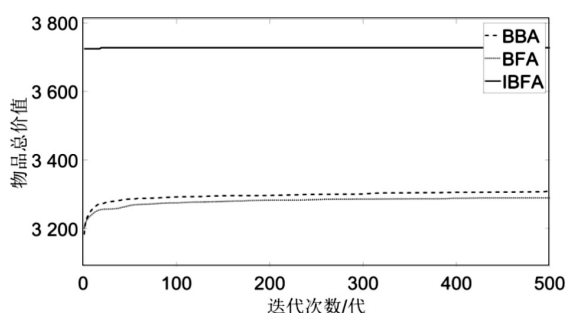


图4 KP04(600维)问题平均进化曲线

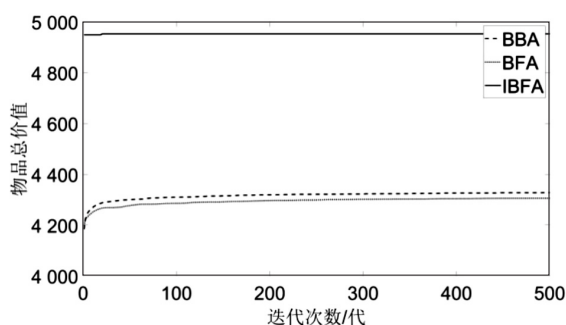


图5 KP05(800维)问题平均进化曲线

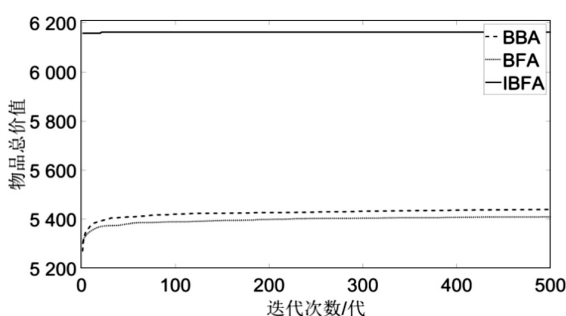


图6 KP06(1000维)问题平均进化曲线

5 结束语

为求解0-1背包问题,首先对捕鱼算法进行修改,即引入二进制编码而提出二进制捕鱼算法。并在此基础上,对其进行优化和改进,提出了改进二进制捕鱼算法。数值实验结果表明,与其它群智能算法相比,改进二进制捕鱼算法具有收敛速度快、全局搜索能力强、求解结果稳定、鲁棒性好等优点。特别是对于高维背包问题,与经典二进制蝙蝠算法相比,改进二进制捕鱼算法的综合性能明显占优。在未来的研究中,拟将该算法应用于车间调度等问题,以进一步测试其性能。

参考文献:

- [1] FAYARD D, PLATEAU G. Resolution of the 0-1 knapsack problem comparison of methods[J]. Mathematical Programming, 1975, 8(1): 272-307.
- [2] 陈 桢, 钟一文, 林 娟. 求解0-1背包问题的混合贪婪遗传算法[J]. 计算机应用, 2021, 41(1): 87-94.
- [3] 刘生建, 杨 艳, 周永权. 求解0-1背包问题的二进制狮群算法[J]. 计算机工程与科学, 2019, 41(11): 2079-2087.
- [4] 李珍萍, 赵雨薇, 张煜炜, 等. 共同配送选址-路径问题及大邻域搜索算法[J]. 系统仿真学报, 2021, 33(10): 2518-2531.
- [5] KENNEDY J, EBERHART R. Particle swarm optimization [C]//Proceedings of the 1995 international conference on neural networks. Piscataway: IEEE, 1995: 1942-1948.
- [6] 刘雨青, 向 军, 曹守启. 基于改进蚁群算法的水下自主航行机器人路径规划[J]. 计算机工程与科学, 2022, 44(3): 536-544.
- [7] FENG Y, WANG G, GAO X. A novel hybrid cuckoo search algorithm with global harmony search for 0-1 knapsack problems[J]. International Journal of Computational Intelligence Systems, 2016, 9(6): 1174-1190.
- [8] MIRJALILI S, MIRJALILI S M, YANG X S. Binary bat algorithm[J]. Neural Computing and Applications, 2013, 25(3-4): 663-681.
- [9] 陈建荣, 王 勇. 采用捕鱼策略的优化方法[J]. 计算机工程与应用, 2009, 45(9): 53-56.
- [10] 陈建荣, 陈建华. 求解TSP问题的离散捕鱼策略优化算法[J]. 计算机科学, 2017, 44(S1): 139-140.
- [11] 徐 敏, 戴 薇. 基于渔夫捕鱼优化算法的配电网重构[J]. 电测与仪表, 2015, 52(13): 43-47.
- [12] 梁晓龙, 李祚泳, 汪嘉杨. 蜜蜂进化遗传与捕鱼策略相结合的优化算法[J]. 数学的实践与认识, 2016, 46(17): 143-148.
- [13] 陈建荣, 陈建华. 求解绝对值方程的捕鱼算法[J]. 计算机时代, 2022(2): 72-75.
- [14] 郭嘉宇, 付晓东, 岳 昆, 等. 偏好匹配满意度最大化的众包任务分配[J]. 计算机工程与科学, 2022, 44(1): 16-26.
- [15] 张国辉, 陆熙熙, 胡一凡, 等. 基于改进帝国竞争算法的柔性作业车间机器故障重调度[J]. 计算机应用, 2021, 41(8): 2242-2248.