Computer Science and Application 计算机科学与应用, 2020, 10(11), 2034-2048
Published Online November 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.1011215

Hans汉斯

基于约束一致策略的人工蜂群算法

吴钰晗，梁晓丹

天津工业大学计算机科学与技术学院，天津
Email: 1831125476@tiangong.edu.cn, lxdtjpu@163.com

收稿日期：2020年11月3日；录用日期：2020年11月18日；发布日期：2020年11月25日

摘　要

在过去的研究中，进化算法逐渐被广泛应用于求解复杂优化问题。然而，利用进化算法解决约束优化问题时，常常不能得到很好的结果，因为它们不能直接减少约束问题的约束违反程度。为了能够更好地得到目标函数的最优解，且减少约束违反对最优解的影响，本文将人工蜂群算法(ABC)的全局优化优势和约束一致策略(CC)的稳定计算特性集成到一种新的混合启发式算法中——基于约束一致策略的人工蜂群算法(ABCCC)。在进化搜索过程中，约束一致策略对于快速减少约束违反是相当有效的。通过一组测试函数，以及基于约束一致策略的粒子群算法(PSOCC)和基于约束一致策略的差分进化算法(DECC)两种方法进行比较，证明ABCCC具有一定的处理约束优化问题的能力。实验结果表明，该算法在优化质量和收敛速度方面都具有良好的性能。

关键词

约束优化，人工蜂群算法，约束一致，进化算法

Artificial Bee Colony Algorithm Based on Constrained Consistent Strategy

Yuhan Wu, Xiaodan Liang

School of Computer Science and Technology, Tiangong University, Tianjin
Email: 1831125476@tiangong.edu.cn, lxdtjpu@163.com

Received: Nov. 3rd, 2020; accepted: Nov. 18th, 2020; published: Nov. 25th, 2020

Abstract

Over the last few decades, evolutionary algorithms have been widely used to solve complex optimization problems. However, when using evolutionary algorithms to solve constraint optimization problems, best results are often not obtained, because they cannot directly reduce the degree of constraint violation. In order to obtain the better optimal solution of the objective function and reduce the impact of constraint violation on the optimal solution, this paper integrates the global optimization advantages of the artificial bee colony algorithm (ABC) and the stable computing cha-

文章引用: 吴钰晗, 梁晓丹. 基于约束一致策略的人工蜂群算法[J]. 计算机科学与应用, 2020, 10(11): 2034-2048.
DOI: 10.12677/csa.2020.1011215

racteristics of the constraint consensus strategy (CC) into a new hybrid heuristic algorithm—the constraint consensus strategy based artificial bee colony algorithm (ABCCC). During the evolutionary search, the constraint consensus strategy is quite effective for rapidly reducing constraint violations. Through a set of test functions and a comparison between PSOCC and DECC, it is proved that ABCCC has certain ability to deal with constraint optimization problems. Experimental results show that the algorithm has good performance in optimizing quality and convergence speed.

Keywords

Constraint Optimization Problem, Artificial Bee Colony (ABC), Constraint Consensus (CC), Evolutionary Algorithms

1. 引言

约束优化问题是在给定约束条件下对目标函数进行优化的问题[1]，在实际应用中经常出现。具有不等式、等式、上界和下界约束的一般约束优化问题定义为：

$$\begin{aligned} &\min \ f(x) \\ &s.t. \ g_j(x) \le 0, \ j = 1,2,\cdots,q \\ &\quad h_j(x) = 0, \ j = q+1,\cdots,m \\ &\quad l_i \le x_i \le u_i, i = 1,2,\cdots,n \end{aligned} \tag{1}$$

其中 $x = (x_1, x_2, \cdots, x_n)$ 为 n 维变量，$f(x)$为目标函数，$g(x)$为不等式约束，$h(x)$为等式约束，j 为不等式或等式约束的个数，l_i 和 u_i 分别是 x_i 的下界和上界。上下界定义了约束优化问题的搜索空间，不等式和等式定义了可行域。可行域或边界上的点称为可行点，否则即为不可行点。

由于约束优化问题的复杂性，传统的进化算法难以求解。同时，进化算法大多是基于无约束优化问题的搜索技术，在应用进化算法求解约束优化问题时，其最优解可能不精确。因此，有必要结合合适的约束处理技术来处理约束优化问题。

本文提出了一种基于约束一致策略的人工蜂群算法(ABCCC)。人工蜂群算法(Artificial bee colony, ABC)是由 Karaboga 提出的一种最新的启发式算法，其灵感来自于蜜蜂的觅食行为，用于求解数值优化问题[2]。相对于差分进化算法(DE)和粒子群算法(PSO)，ABC 算法有两个明显的优势：1) ABC 在局部优化和全局优化方面都很好。2) ABC 灵活、稳健、使用简单，它可以有效地应用于多模态、多变量问题的优化。为了驱动个体向可行域移动，在 ABC 算法中引入了约束一致策略。CC 策略具有以下优点[3]：1) CC 策略易于实现。2) CC 计算量小，计算稳定性好。为了有效地结合 ABC 算法和 CC 策略，文中重新设计了对不可行个体的处理。即部分不可行个体采用 CC 策略进行进化，其余不可行个体和可行个体采用 ABC 算法进行再生。为了验证 ABCCC 的特性，采用 CEC2017 基准函数对其性能进行测试。实验表明，在大多数情况下，ABCCC 收敛速度较快，收敛精度较好，并且能较好地考虑种群多样性问题。ABCCC 与 DECC 和 PSOCC 相比也更具竞争力。然后利用 ABCCC 算法与其他算法(CALSHADE [4]、UDE [5]、CABC [6]和 GABC [7])进行了比较。实验结果表明，ABCCC 算法的性能优于其他算法。

本文组织结构如下，引言后，第 2 节详细阐述了约束一致策略，第 3 节介绍了人工蜂群算法，新算

法的详细信息见第 4 节，第 5 节给出了实验结果和分析。最后，第 6 节对全文做了总结。

2. 约束一致策略

约束一致策略(Constraint Consensus Strategy, CC)是一种同步分量平均梯度投影算法[8] [9] [10]，主要的思想是通过采取某些行动，在当前违反的约束之间产生一个在行动的方向和距离上“一致意见”的结果，根据这个结果改进当前的不可行点以实现可行性。

约束一致策略算法的第一步就是找到在当前点 x 处违反的每个约束的可行性向量，对于每个约束违反，可行性向量近似从当前的不可行解到最近的可行解的移动。它对于线性约束是精确的，但对于非线性约束只是一个估计。衡量不可行点到可行区域的距离是点与可行区域之间的最小欧几里得距离，称为可行性距离[11]。可行性向量的计算公式如下：

$$fv = vd\nabla c(x) \Big/ \left\|\nabla c(x)\right\|^2 \tag{2}$$

其中，$\nabla c(x)$是约束的梯度，$\|\nabla c(x)\|$是它的长度；v 是约束违反量，当满足约束条件时 $v = 0$；d 是一个方向参数，如果有必要增加 $c(x)$满足约束条件则 $d + 1$，如果有必要减少 $c(x)$来满足约束条件则 $d - 1$。可行性向量的长度表示为$\|fv\|$，也即可行性距离。

Table 1. Pseudocode of constraint consensus strategy
表 1. 约束一致策略

Inputs:	a set of constraints c
	an initial point x
	a feasibility distance tolerance α
	a movement tolerance β
1	NINF = 0, for all j: $n_j = 0$, $s_j = 0$
2	For every constraint c_i
3	If c_i is violated
4	Calculate feasibility vector fv and the feasibility distance$\|fv\|$
5	If $\|fv\| > \alpha$
6	NINF = NINF + 1
7	For every variable x_j in c_i
8	$n_j \leftarrow n_j + 1$, $s_j \leftarrow s_j + fv_{ij}$
9	End for
10	End if
11	End if
12	End for
13	If NINF = 0, then exist successfully
14	For every variable x_j
15	$t_j = s_j / n_j$
16	End for
17	If $\|t\| \le \beta$ then exit unsuccessfully
18	$x \leftarrow x + t$
19	If necessary, reset x to respect any violated variable bounds
20	Go to step1

第二步是对所有违反约束的可行性向量进行组合，得到实际用于更新的一致向量。这个过程是以组件方式完成的：只有在 $c(x)$中包含特定变量的违反约束才能对该维度中的移动进行“投票”。通过对各可行性向量的相关分量进行平均得到各维度的运动，产生的一致向量指定了移动的方向和距离，应用一致向量来更新当前点。这两步不断重复直至满足终止条件。约束一致策略的算法步骤如表 1 所示。NINF 是当前违反约束的数量，s_j 表示变量 x_j 的 fv_{ij} 对所有违反约束条件的可行性向量的和，n_j 代表以变量 x_j 为组件的违反约束的数量，fv_{ij} 表示第 i 个约束的可行性向量中变量 x_j 的分量，t 为一致向量。如果这个向量太短，那么迭代将停止，结果是失败的。当每个约束的约束违反量为零或可行性距离小于α，即 NINF 为零时，算法停止[8]。

约束一致策略的方法是简单的，缺点在于，单个可行性向量组合的方法可能会被特殊情况所阻碍，特别是一组向某个方向拉动的可行性向量与另一组向相反方向拉动的可行性向量相平衡时。但是，当约束一致策略与进化算法一起使用时，这样的个体不会产生任何问题。

3. 人工蜂群算法

人工蜂群算法(Artificial Bee Colony, ABC)，是由 Karaboga 发明的一种基于群体的随机优化方法，它模拟了蜜蜂的智能觅食行为，蜜蜂根据各自的分工进行不同的活动，并实现蜂群信息的共享和交流，从而找到问题的最优解。

Table 2. Pseudocode of ABC algorithm
表 2. ABC 的伪代码

```
Initialize the food sources and evaluate the nectar amount (fitness) of food sources
Send the employed bees to the current food source
Iteration = 0
Do while (the termination conditions are not met)
    /*Employed Bees' Phase*/
    For (each employed bee)
        Generate a candidate solution in its neighborhood following Equation (4)
        Evaluate the candidate solution and apply greedy selection
    End for
    Calculate the probability P for each food source according to Equation (5)
    /*Onlooker Bees' Phase*/
    For (each onlooker bee)
        Send onlooker bees to food sources by the roulette selection depending on P
        Generate a candidate solution in its neighborhood following Equation (4)
        Evaluate the candidate solution and apply greedy selection
    End for
    /*Scout Bees' Phase*/
    If (any employed bee becomes scout bee)
        Send the scout bee to a randomly produced food source
    End if
    Memorize the best solution achieved so far
    Iteration = Iteration + 1
End while
Output the best solution achieved
```

在一个自然的蜂群中，通常有三种觅食的蜜蜂，分别是雇佣蜂、旁观蜂和侦查蜂。蜂群的一半是雇佣蜂，另一半是旁观蜂。雇佣蜂负责在自己的蜜源周围搜寻，同时向旁观蜂传递高质量的蜜源信息。旁观蜂根据收到的信息，倾向于选择更好的蜜源进行开采。若某蜜源在规定迭代次数内未更新，则雇佣蜂抛弃该位置转化为侦查蜂，去随机寻找新的蜜源[1] [12]。表 2 列出了人工蜂群算法的伪代码。

人工蜂群算法中，蜜源的位置表示优化问题的解，每个蜜源的花蜜量表示相应解的适应度值。首先，对蜜源的位置进行初始化，雇佣蜂(或旁观蜂)的数量等于蜜源的数量：

$$x_{i,j} = x_j^{\min} + rand(0,1)\left(x_j^{\max} - x_j^{\min}\right) \tag{3}$$

其中，$i = 1, 2, \ldots, SN$，$j = 1, 2, \ldots D$，SN 是蜜源的数量。D 为决策变量的维度，表示待优化参数的个数。$xmax\ j$ 和 $xmin\ j$ 分别是蜜蜂在第 j 维坐标位置的上界和下界。另外，在此阶段，存储每个蜜蜂试验次数的计数器被设置为 0。

初始化阶段结束后，ABC 进入雇佣蜂、旁观蜂和侦查蜂阶段的循环，直到满足终止条件。在雇佣蜂阶段，每只雇佣蜂对应一个确定的蜜源并在迭代中对邻近的蜜源进行搜索，根据式(4)产生相邻的蜜源：

$$v_{i,j} = x_{i,j} + \phi\left(x_{i,j} - x_{k,j}\right) \tag{4}$$

k 是不同于 i 的随机蜜源，j 是随机选择的维度。ϕ是在[−1, 1]范围内的一个均匀分布的随机数，决定了扰动程度。通过改变 x 上的一个维度来确定新的蜜源 v，如果这个操作产生的这个维度的值超过了预定边界，就把其设置为边界值。

然后评估新蜜源的适应度值，当新蜜源 v 的适应度优于 x 时，采用贪婪选择法用新蜜源代替原来的蜜源，否则保留 x。如果蜜源发生了变化，试验计数器将重置为零；否则，其值将增加 1。所有的雇佣蜂完成该步骤的运算后，飞回信息交流区共享蜜源[13]。

在旁观蜂阶段，旁观蜂接收到雇佣蜂共享的蜜源信息，然后他们会根据蜜源的花蜜量的概率，各自选择一种蜜源进行开发。概率的计算公式如(5)所示：

$$P_i = \frac{fitness_i}{\sum_{j=1}^{SN} fitness_j} \tag{5}$$

人工蜂群算法中，解的适应度评估根据公式(6)计算，f_i 表示解的函数值：

$$fit_i = \begin{cases} 1/(1+f_i) & f_i \geq 0 \\ 1+abs(f_i) & otherwise \end{cases} \tag{6}$$

旁观蜂采用轮盘赌的方法选择蜜源后，每只旁观蜂会像雇佣蜂一样，按照公式(4)在其附近继续寻找新的蜜源。

在侦查蜂阶段，引入一个叫做“$limit$”的控制参数来决定是否抛弃某一蜜源。如果该蜜源没有在“$limit$”预定的周期内得到改善，那么该蜜源就会被抛弃，并且对应该蜜源的雇佣蜂转化为侦查蜂。接着使用(3)在搜索空间中随机产生一个新的蜜源，就像初始化阶段一样。

$$x_i^{t+1} = \begin{cases} x_j^{\min} + rand(0,1)\left(x_j^{\max} - x_j^{\min}\right) & \text{trial}_i \geq \text{limit} \\ x_i^t & \text{trial}_i < \text{limit} \end{cases} \tag{7}$$

4. 基于约束一致策略的人工蜂群算法

该章节中，将详细介绍基于约束一致策略的人工蜂群算法的内容。图 1 是该算法的流程图。根据流

程图可以看出，种群在处理的过程中，根据个体是否可行分成了两部分，一部分是少量的不可行个体，另一部分则是可行个体和剩余不可行个体的集合。然后分别采用约束一致策略和人工蜂群算法处理两部分种群个体。

在该算法中，首先随机初始化一个种群 *P*，然后评估种群当前的适应度值和约束违反程度，根据约束违反程度可以找到当前种群状态下的可行个体集合和不可行个体集合。假设初始化后的种群中存在的不可行个体为 *SP*，在 *SP* 中选取部分不可行个体，记为 *SSP*(即只选取部分不可行个体，而不是全部的不可行个体 *SP*)，基于章节 2 介绍的约束一致策略，对 *SSP* 个不可行个体进行处理，产生原部分不可行个体的新位置。随后，剩余的不可行个体与可行个体合并，将经典的人工蜂群算法应用于该个体的集合。算法之所以只选取部分不可行个体进行操作的目的是可以节省计算时间，不可行个体越多，约束一致策略运行的时间越长，另外一个目的则是能够保持种群的多样性。

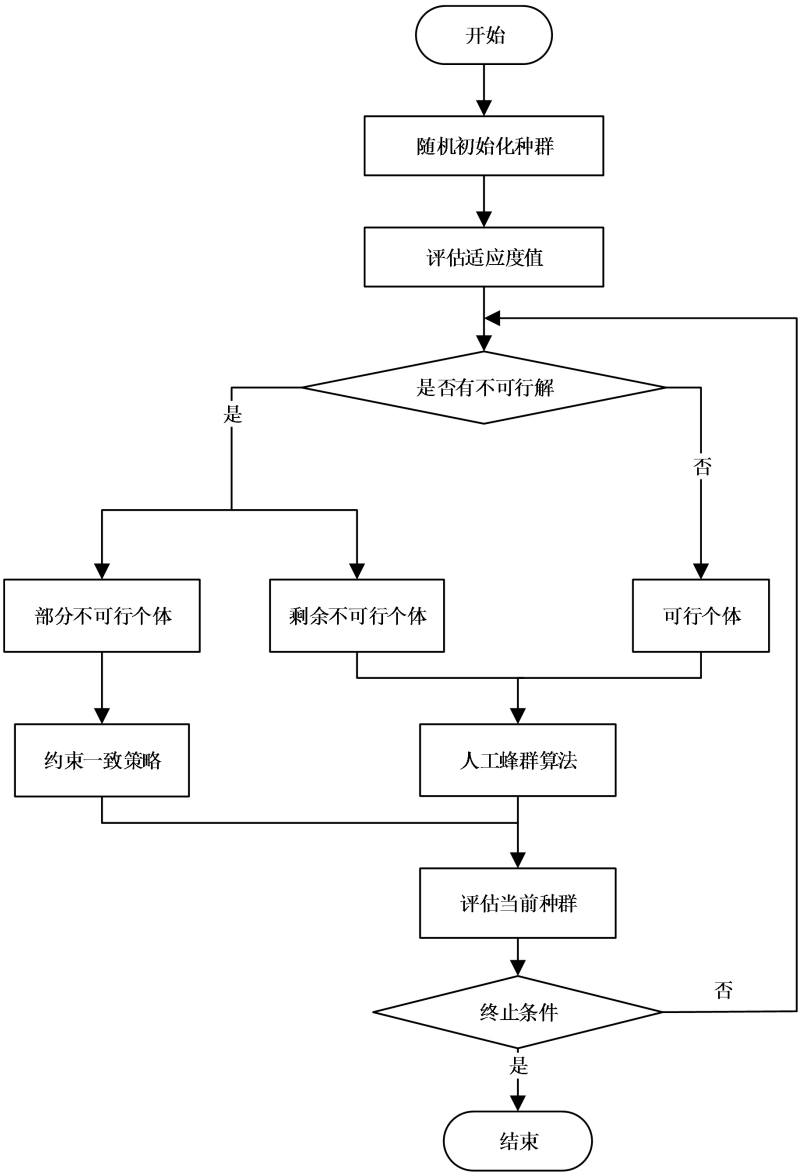


Figure 1. Flow chart of the ABCCC algorithm
图 1. 基于约束一致策略的人工蜂群算法流程

初始种群 P 采用如下公式生成：

$$x_{i,d} = L_d + rand(0,1)(U_d - L_d) \tag{8}$$

其中，U_d 和 L_d 是决策变量的上下界，d 为变量的维度。$rand(0, 1)$表示$(0, 1)$范围内的随机数。$x_{i,d}$即代表初始的种群个体。

接下来，计算初始化种群 P 内全部个体的适应度值和约束违反量，如果所有约束的违反量均小于等于 0，表明当前个体没有违反约束，是可行的个体，这样的个体暂时不采取任何措施，等待后续与剩余不可行个体合并直接采用 ABC 进化。如果至少有一个约束产生违反，即约束违反量大于 0，那么这个个体就是不可行个体，所有不可行个体的集合为 SP。

当不可行个体的集合 SP 不为空时，在 SP 内随机的选取 SPP 个不可行个体，利用前面章节介绍的约束一致策略，改变当前 SPP 个不可行个体的位置。个体的改变依照公式(9)进行。即对每个不可行个体，计算出针对它的一致向量 t，将当前的位置与向量的坐标相加得到个体的新位置，新个体记为 $x'_{i,d}$。剩余的不可行个体和原先找到的可行个体形成的合集利用人工蜂群算法进行演化。若 SP 的集合为空时，说明当前迭代下没有不可行个体，此时，所有的可行个体直接利用人工蜂群算法产生后代，寻找最优解。

$$x'_{i,d} = x_{i,d} + x_{cc,d} \tag{9}$$

需要注意的是，在人工蜂群算法原来的贪婪选择操作下，做了一点改动，将可行性规则引入对个体进行选拔，即个体的选择需要同时考虑目标函数值和约束违反程度。当个体均可行时，目标函数值越小(极小化问题)，个体的解最优；当可行个体与不可行个体同时存在，可行个体的解总是优于不可行个体的解；当同为不可行个体时，选取约束违反量总和最小的解予以保留。

该算法中，等式约束的处理将转换为公式(10)的不等式，ε是一个容差值，实验中取值 10^{-4}。

$$\left|h_j(x)\right| - \varepsilon \le 0 \tag{10}$$

算法的伪代码如表 3 所示：

Table 3. Pseudocode of ABCCC
表 3. ABCCC 伪代码

1	Initialize a population P
2	Evaluate the fitness values and constraint violations
3	If there are infeasible solutions (SP)
4	Select SPP of the infeasible solutions and generate offspring using CC strategy
5	The merged group evolves utilizing ABC algorithm
6	Sort the entire population based on the feasibility rules
7	End if
8	Go to step 2

5. 实验结果与分析

该章节主要列举和分析了 ABCCC 算法的实验结果，首先在指定的测试函数集进行测试，找到了测试集的优秀结果，验证了该算法能够用来解决约束优化问题。接下来，还将 ABCCC 与 DECC 和 PSOCC 算法进行了对比(把 ABC 替换为差分进化算法(DE)和粒子群算法(PSO))，实验结果表明，ABCCC 优于其余两个算法。同时该章节还分析了算法内部参数对实验结果的影响。最后将 ABCCC 算法与其他较优秀

的算法进行了比较，实验表明了 ABCCC 在解决约束优化问题上具有一定的竞争力。

5.1. 测试函数集与参数设置

通过求解 CEC2017 约束优化测试函数集中的部分函数，对算法进行了测试[14]。函数集中的问题具有不同的数学特征，比如有的目标函数或约束是线性或非线性的，有的约束是等式或不等式，有的目标函数是单峰或多峰。并且有的函数是可行空间非常小，这更增加了解决问题的难度。

算法的参数设置包括如下内容：种群 P 的大小为 100，选择 SPP 个不可行个体的依据是 $SPP = 0.5SP$ (在 5.3 章节讨论为什么将比例设置为 0.5)，总迭代次数设为 100，每个问题独立运行的次数为 10 次。对比算法 DECC 和 PSOCC 的参数设置为：DE 的变异因子 F 设为 0.5，交叉因子 CR 设为 0.4；PSO 的学习因子 C_1 和 C_2 相等，取值均为 1.4，惯性权值 w 设为 0.8。约束一致策略的参数设置如下：可行距离的容差值α设为 10^{-6}，移动容差值β为 10^{-4}。

5.2. 实验结果

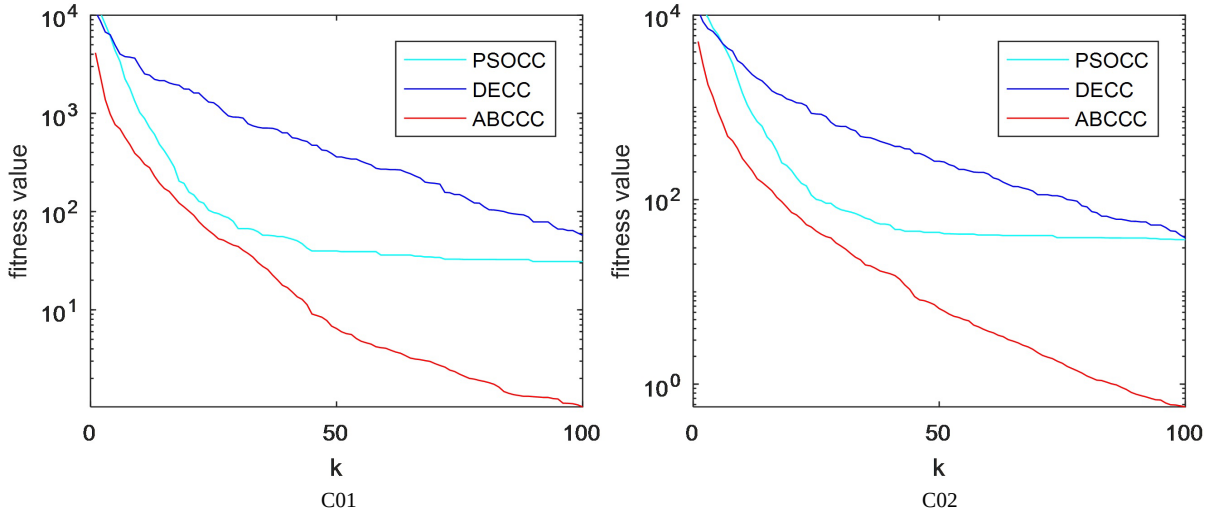
在本节，首先列出了 ABCCC 在 CEC2017 函数集下的实验结果，同时还对比了 DECC 和 PSOCC 在相同状况下的实验结果，表 4 列出了实验过程中得到的最优解、平均值和标准差。最优解表明算法在解决该问题时找到的目标函数的最小值，平均值反映了 10 次运行的平均结果，标准差则能表明算法的鲁棒性，标准差越小，表明算法每次运行得到的结果都是近似的，说明算法是稳定的。实验结果表明 ABCCC 的算法性能总体上要优于 DECC 和 PSOCC，收敛迅速且能找到最优解。

从表 4 可以看出，ABCCC 算法得到的最优解和平均解在大多数函数(C01、C02、C03、C04、C05、C06、C07 和 C09)的表现都优于 DECC 和 PSOCC。而对于函数 C08 和 C10，DECC 得到的结果则是最小的，优于 PSOCC 和 ABCCC。其中有一个函数 C09，该函数在 ABCCC 和 PSOCC 下取得的最优解相同。

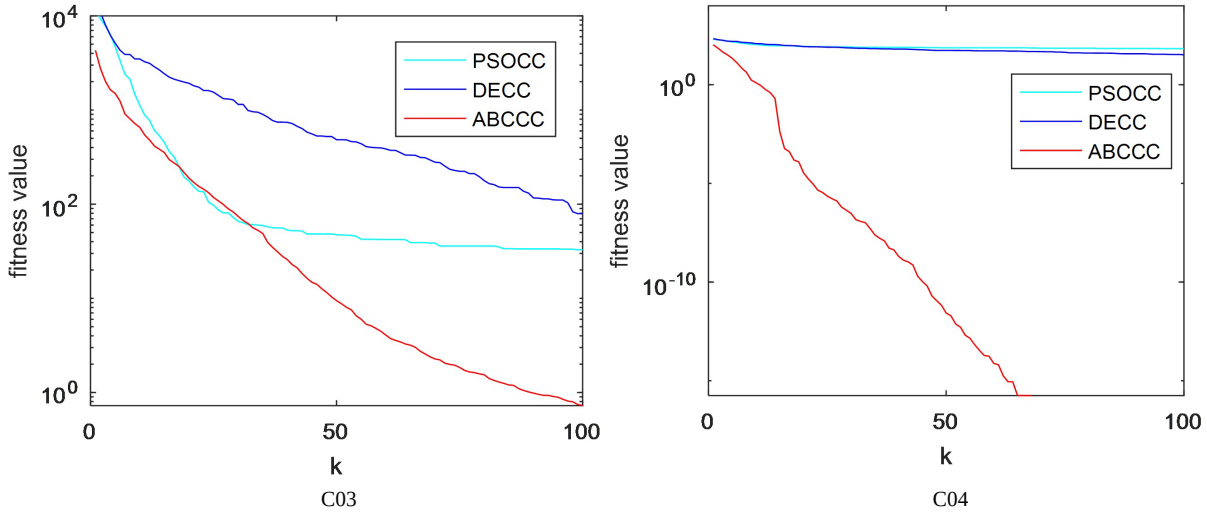
Table 4. Comparison results of ABCCC, DECC and PSOCC
表 4. ABCCC 与 DECC 和 PSOCC 的对比结果

	Algorithm	best	mean	std		best	mean	std
	ABCCC	**4.3422e−01**	**1.0276e+00**	**7.7749e−01**		**0.0000e+00**	**1.9903e−01**	**6.2940e−01**
C01	DECC	2.0552e+01	5.7294e+01	2.6456e+01	C06	1.1025e+02	1.5862e+02	5.6512e+01
	PSOCC	1.9394e+01	3.0953e+01	7.5116e+00		6.9656e+01	1.3079e+02	5.3494e+01
	ABCCC	**1.1917e−01**	**5.6705e−01**	**3.6207e−01**		**−5.2474e+02**	**−5.2474e+02**	**6.7115e−06**
C02	DECC	2.2081e+01	3.8531e+01	1.2447e+01	C07	−3.1552e+02	−3.0261e+02	8.6554e+00
	PSOCC	2.8930e+01	3.6837e+01	4.6696e+00		−2.3716e+02	−2.0193e+02	2.3230e+01
	ABCCC	**2.6169e−01**	**7.2534e−01**	**4.5649e−01**		1.2499e+00	2.6492e+00	1.0700e+00
C03	DECC	1.9347e+01	7.9392e+01	2.6244e+01	C08	**7.6596e−01**	**1.4086e+00**	**4.8360e−01**
	PSOCC	2.5698e+01	3.2883e+01	3.7946e+00		1.2276e+00	2.2839e+00	5.8413e−01
	ABCCC	**0.0000e+00**	**0.0000e+00**	**0.0000e+00**		**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**
C04	DECC	1.7852e+01	3.2618e+01	7.5740e+00	C09	1.8540e−01	3.2352e−01	1.2396e−01
	PSOCC	4.4996e+01	6.7187e+01	1.4799e+01		**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**
	ABCCC	**3.8795e−04**	**5.4388e−03**	**9.4572e−03**		1.0175e+01	1.5753e+01	5.1411e+00
C05	DECC	7.2160e+00	8.1310e+00	6.5960e−01	C10	**4.5172e−01**	**1.1297e+00**	6.1977e−01
	PSOCC	2.0620e+01	4.6958e+01	3.0318e+01		1.3172e+00	2.3567e+00	**4.3120e−01**

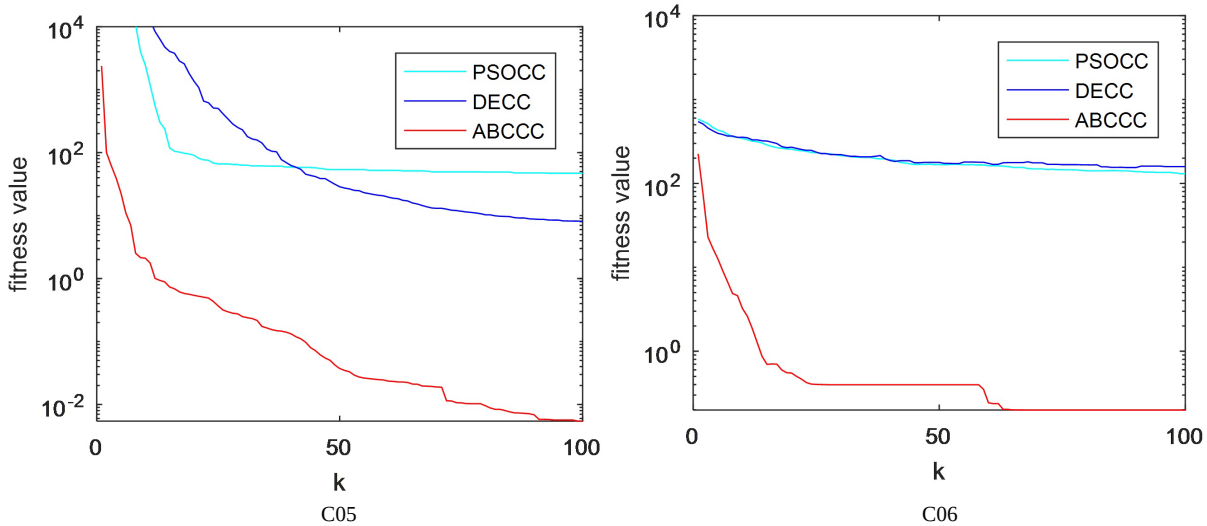
从图 2 所示的收敛曲线图可以看出，ABCCC 比其他算法收敛速度更快，收敛结果较优，得到了更理想的全局最优解。然而，函数 C08 和 C10 则是 DECC 得到的解更优。



C01 C02



C03 C04



C05 C06

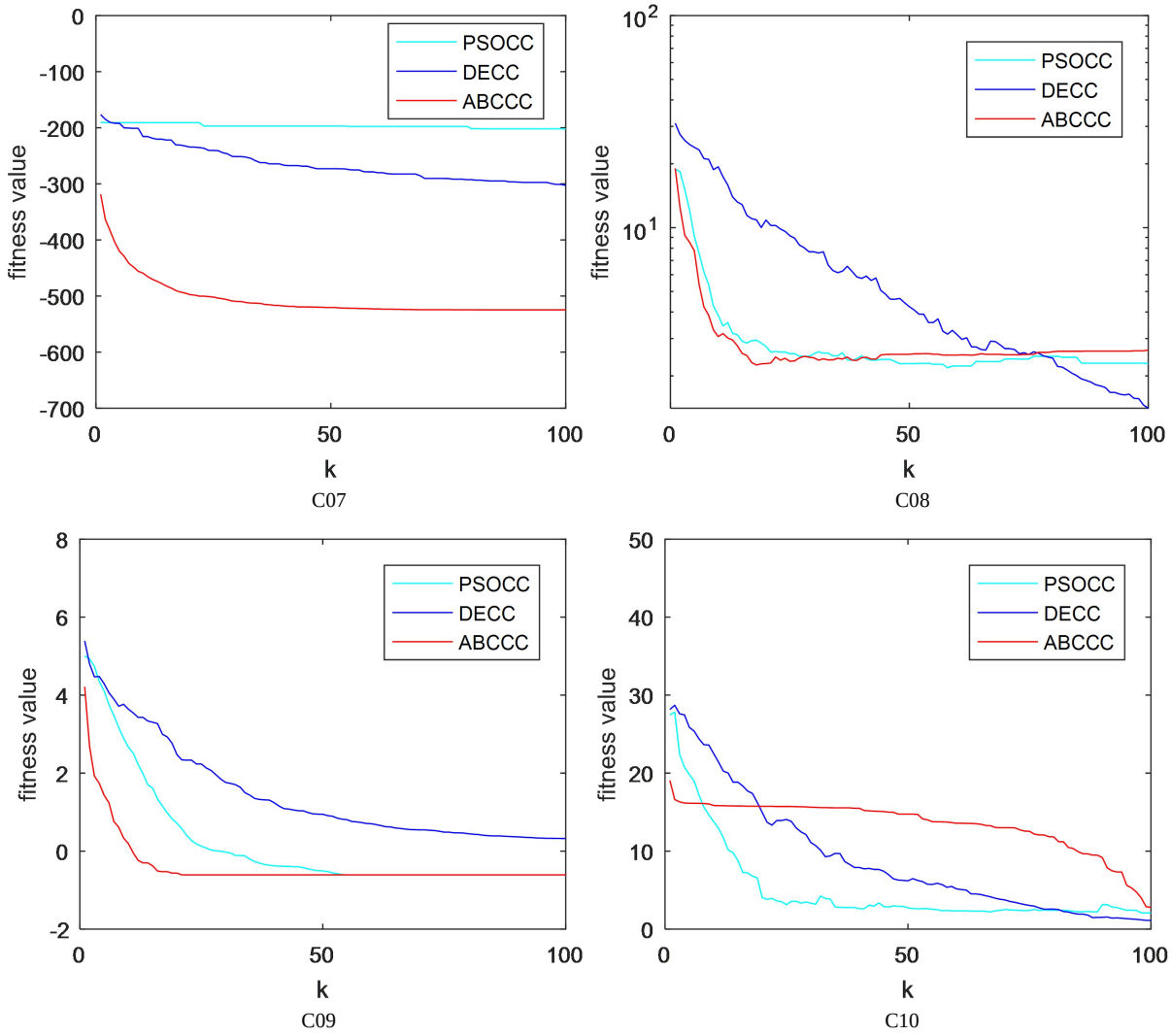
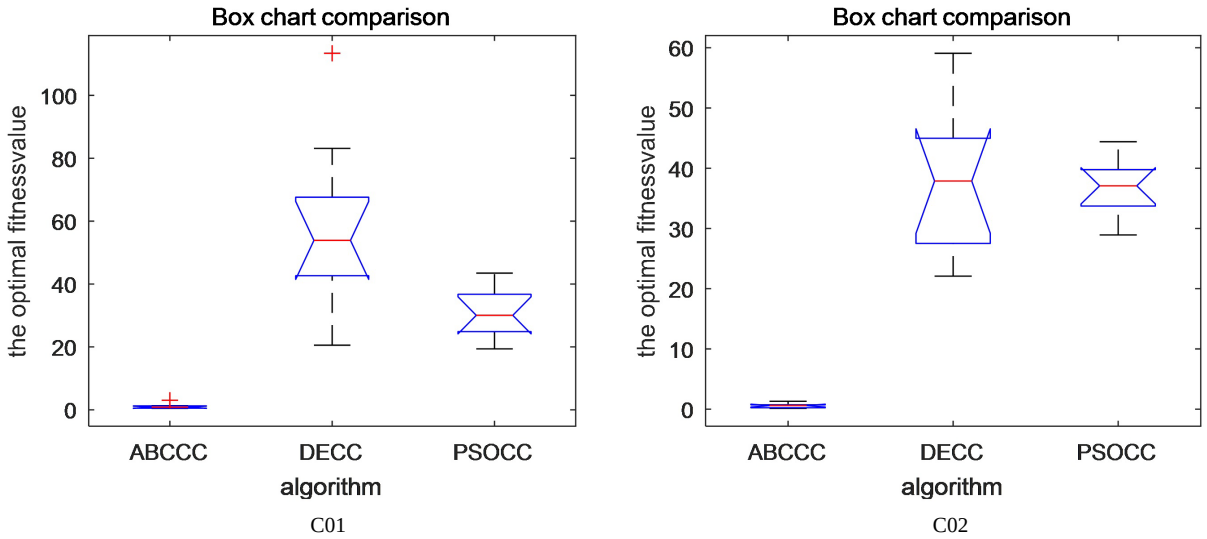
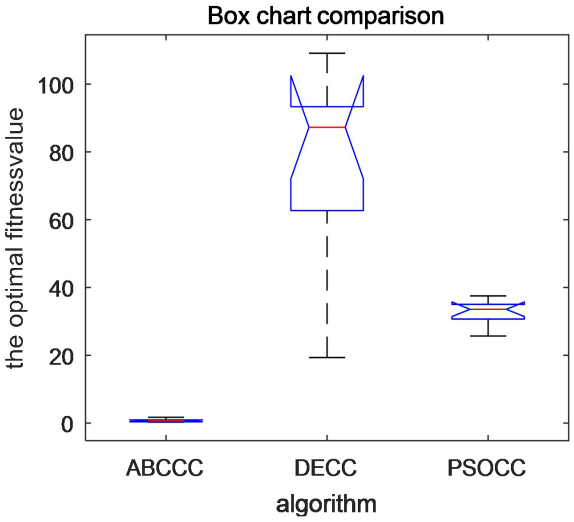
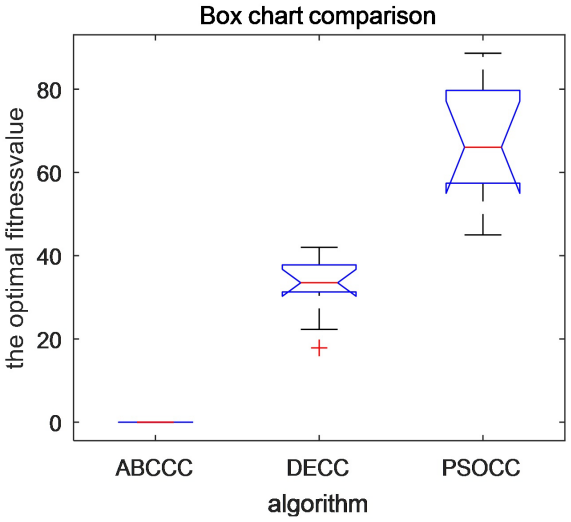


Figure 2. Convergence curves of each test functions
图 2. 各算法的收敛曲线对比图

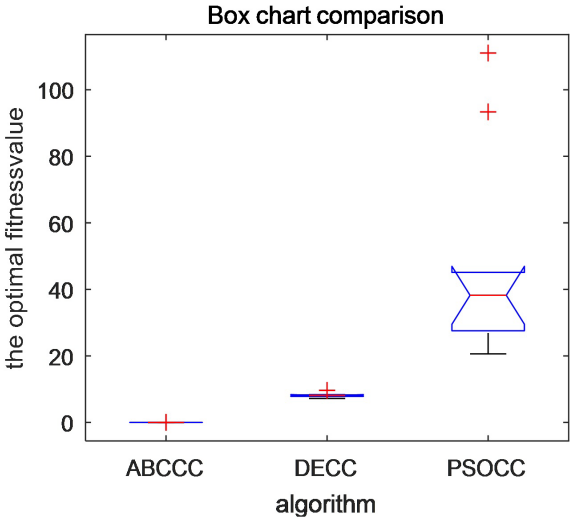




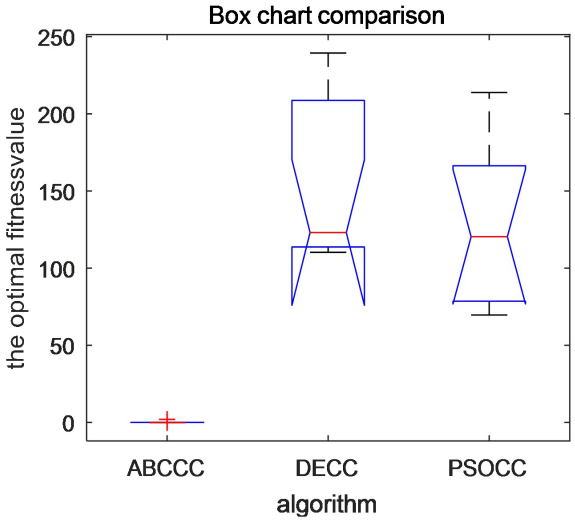
C03



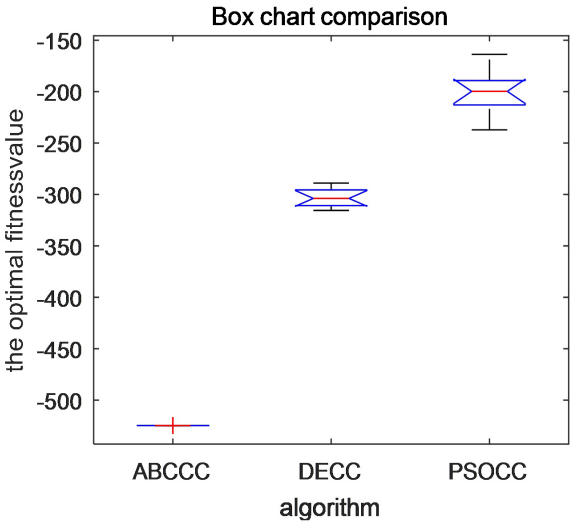
C 04



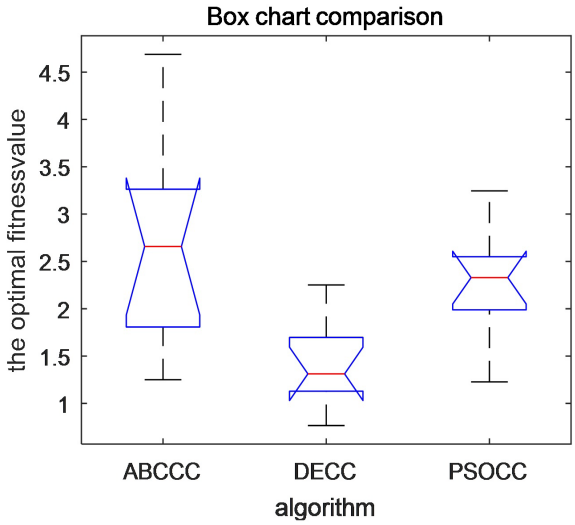
C05



C 06



C07



C 08

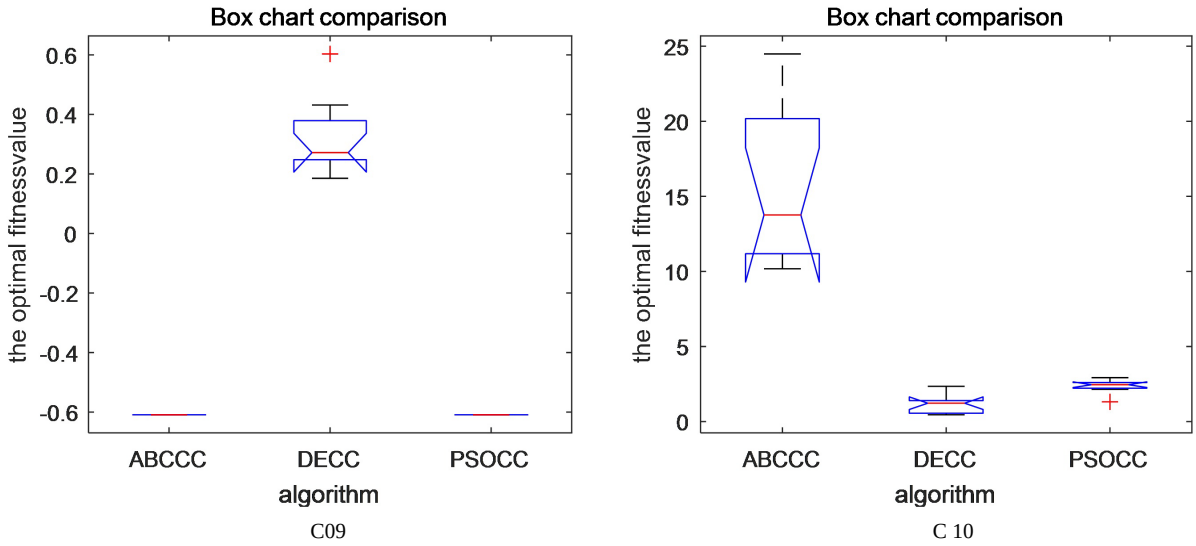


Figure 3. Box chart comparison results of each test functions
图 3. 算法的盒图对比结果

图 3 给出了算法的盒图。对于每个盒图，中心标记表示中间值，方框的底部和顶部边缘分别表示第 25 百分位数和第 75 百分位数。胡须延伸到不被认为是异常值的最极端数据点，并使用“+”符号分别绘制异常值。从图 3 可以看出，ABCCC 在大部分测试函数(C01、C02、C03、C04、C05、C06、C07、C09)均是性能较优越的。

5.3. *SPP* 值对 ABCCC 的影响

在 5.1 章节中，*SPP* 参数的选取为 0.5，即选取了不可行个体的一半，在该章节中，我们将测试 *SPP* 值对算法性能的影响。

处理不可行个体时，5.1 节中介绍的是在所有的不可行个体中只选取部分个体，因此本节按照不同比例范围选值进行测试，其中 *SPP* 设置为 0.1*SP*、0.3*SP*、0.5*SP*、0.7*SP* 和 0.9*SP* (*SP* 即为前面章节介绍的全部不可行个体)。*SPP* 值越高，预计搜索周期越长，具体的实验结果如表 5 所示。从搜索周期的角度看，*SPP* 的值越小，搜索的时间越短，所以当 *SPP* = 0.1*SP* 时算法会更好，因为它的运行时间相对最短。但是从解的质量角度看，当 *SPP* = 0.5*SP* 时，大多数函数的实验结果比其他情况较好，尤其是平均适应度值的结果。因此 ABCCC 算法中 *SPP* 值的选取为 0.5 时更合适，能够解决大部分的测试函数问题。

Table 5. The influence of *SSP* on results
表 5. *SPP* 值的选取对算法结果的影响

	SPP−value	best	aver	std		best	aver	std
C01	*SPP* = 0.1	3.6224e−01	2.1745e+00	3.7883e+00	C02	**9.6252e−02**	7.5394e−01	5.1756e−01
	SPP = 0.3	6.6545e−01	1.0406e+00	**3.4463e−01**		1.2425e−01	1.9530e+00	2.8007e+00
	SPP = 0.5	4.3422e−01	**1.0276e+00**	7.7749e−01		1.1917e−01	**5.6705e−01**	**3.6207e−01**
	SPP = 0.7	8.0072e−01	1.4671e+00	7.0296e−01		2.6261e−01	9.3691e−01	1.2533e+00
	SPP = 0.9	**2.7358e−01**	1.2008e+00	6.8819e−01		2.5149e−01	9.4132e−01	1.2486e+00
C03	*SPP* = 0.1	3.3249e−01	1.2147e+00	5.6815e−01	C04	**0.0000e+00**	**0.0000e+00**	**0.0000e+00**
	SPP = 0.3	5.2694e−01	1.3587e+00	7.7649e−01		**0.0000e+00**	**0.0000e+00**	**0.0000e+00**

Continued

	SPP = 0.5	2.6169e−01	**7.2534e−01**	**4.5649e−01**		**0.0000e+00**	**0.0000e+00**	**0.0000e+00**
C03	*SPP* = 0.7	**1.7941e−01**	2.2968e+00	3.1205e+00	C04	**0.0000e+00**	**0.0000e+00**	**0.0000e+00**
	SPP = 0.9	3.1033e−01	3.1911e+00	3.8990e+00		**0.0000e+00**	**0.0000e+00**	**0.0000e+00**
	SPP = 0.1	3.1579e−04	**3.0156e−03**	**3.4689e−03**		**0.0000e+00**	2.0000e−01	6.3244e−01
	SPP = 0.3	1.3199e−03	1.2395e−02	1.7242e−02		**0.0000e+00**	7.9602e−01	1.0277e+00
C05	*SPP* = 0.5	3.8795e−04	5.4388e−03	9.4572e−03	C06	**0.0000e+00**	**1.9903e−01**	**6.2940e−01**
	SPP = 0.7	**1.4399e−04**	5.3629e−03	6.7180e−03		**0.0000e+00**	1.9921e+00	1.3279e+00
	SPP = 0.9	6.0703e−04	1.2685e−02	2.9036e−02		3.9851e+00	1.1887e+02	1.4430e+02
	SPP = 0.1	**−5.2474e+02**	−5.2474e+02	**2.1047e−07**		5.1001e+00	8.7588e+00	2.7110e+00
	SPP = 0.3	**−5.2474e+02**	−5.2474e+02	2.6055e−07		1.1888e+00	2.8243e+00	**9.0287e−01**
C07	*SPP* = 0.5	**−5.2474e+02**	−5.2474e+02	6.7115e−06	C08	1.2499e+00	**2.6492e+00**	1.0700e+00
	SPP = 0.7	**−5.2474e+02**	**−5.2453e+02**	6.7970e−01		2.5761e+00	4.3356e+00	1.0971e+00
	SPP = 0.9	**−5.2474e+02**	−5.2474e+02	9.0928e−06		**6.5659e−01**	4.0359e+00	1.9499e+00
	SPP = 0.1	**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**		1.4054e+01	2.0605e+01	5.9842e+00
	SPP = 0.3	**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**		8.9854e+00	1.6984e+01	5.7525e+00
C09	*SPP* = 0.5	**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**	C10	1.0175e+01	**1.5753e+01**	5.1411e+00
	SPP = 0.7	**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**		**8.3204e+00**	1.7478e+01	5.2565e+00
	SPP = 0.9	**−6.0917e−01**	**−6.0917e−01**	**0.0000e+00**		8.9586e+00	1.6326e+01	**4.7826e+00**

图 4 为测试函数 C02 的收敛曲线图，从图中可以明显地看出，*SPP* = 0.5*SP* 时收敛速度更快，得到的最优解更好。

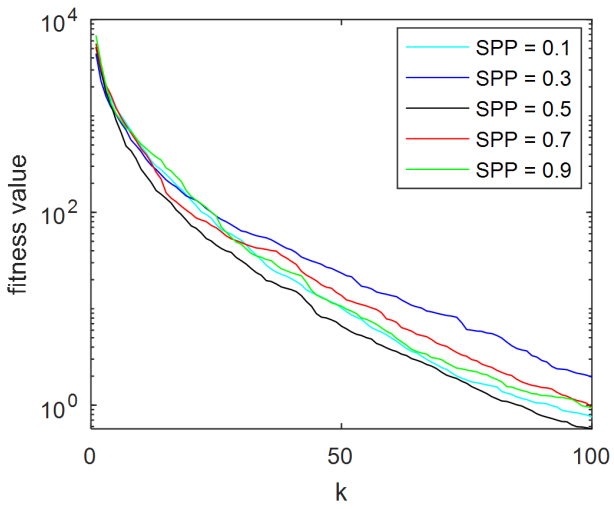


Figure 4. Convergence curve of ABCCC under different *SPP* values (C02)
图 4. 不同 *SPP* 值下 ABCCC 的收敛曲线图(函数 C02)

5.4. ABCCC 与其他算法的比较结果

在该章节，ABCCC 算法与其他的算法进行了比较，具体的实验结果展示在下面的表 6 中。ABCCC

先与 CALSHADE 和 UDE 进行了实验对比，结果表明 ABCCC 能取得大多数测试函数的最优解。但对于函数 C01 和 C02，CALSHADE 算法取得了函数的最优解；对于函数 C08 和 C10，算法 UDE 的表现更好。但从整体来说，实验结果表明 ABCCC 算法的性能优于其他算法。

Table 6. Comparison results of ABCCC、CLSHADE and UDE
表 6. ABCCC 与 CLSHADE 和 UDE 算法的比较结果

	ABCCC		CALSHADE		UDE	
	best	mean	best	mean	best	mean
C01	4.3422e−01	1.0276e+00	**1.6437e−02**	**1.7105e−01**	3.1160e+03	5.2019e+03
C02	1.1917e−01	5.6705e−01	**2.4550e−03**	**1.3606e−01**	2.4394e+03	4.3835e+03
C03	**2.6169e−01**	**7.2534e−01**	1.7797e+04	2.2553e+05	2.0650e+03	4.1730e+03
C04	**0.0000e+00**	**0.0000e+00**	1.8905e+01	3.4916e+01	1.0086e+02	1.3747e+02
C05	**3.8795e−04**	**5.4388e−03**	2.9752e+00	4.0586e+00	1.3716e+04	3.6528e+04
C06	**0.0000e+00**	**1.9903e−01**	1.6082e+02	3.7367e+02	2.1652e+02	3.0200e+02
C07	**−5.2474e+02**	**−5.2474e+02**	−2.1798e+02	−5.9272e+00	−4.0431e+02	−3.5367e+02
C08	1.2499e+00	2.6492e+00	1.5861e−02	8.6148e−02	**−9.0367e+01**	**−9.0367e+01**
C09	**−6.0917e−01**	**−6.0917e−01**	−4.9730e−03	1.8271e−01	**−6.0917e−01**	**−6.0917e−01**
C10	1.0175e+01	1.5753e+01	8.7000e−05	4.6556e−03	**−5.9754e+01**	**−5.9754e+01**

人工蜂群算法在后来的研究中被许多研究者提出了一些变体，比如 CABC 和 GABC，因此该章节也将 ABC 算法替换为 CABC 和 GABC，将这些算法进行了比较，结果如表 7 所示。除了测试函数 C01，C08 和 C10 外，ABCCC 在其他测试函数中均能够找到最优解。而函数 C01 和 C08 的最优解由 GABCCC 找到，对于 C10，则是 CABCCC 获得了最小值。究其原因，GABCCC 利用全局最优值在搜索周期中对个体进行变异，对结果的影响较大。虽然 CABCCC 基于随机选择的某个维度对每个个体进行变异操作，但也降低了收敛速度，导致搜索周期变长。

Table 7. Comparison results of ABCCC, CABCCC and GABCCC
表 7. ABCCC 与 CABCCC 和 GABCCC 算法的比较结果

	ABCCC		CABCCC		GABCCC	
	best	mean	best	mean	best	mean
C01	4.3422e−01	1.0276e+00	4.0138e+02	1.1968e+03	**2.7349e−01**	**8.8519e−01**
C02	**1.1917e−01**	**5.6705e−01**	1.9751e+02	8.9848e+02	1.7921e−01	1.7533e+00
C03	**2.6169e−01**	**7.2534e−01**	3.8785e+02	7.8629e+02	2.8725e−01	1.2546e+00
C04	**0.0000e+00**	**0.0000e+00**	6.3515e+01	7.9414e+01	0.0000e+00	1.9362e−14
C05	**3.8795e−04**	**5.4388e−03**	7.5505e+02	9.1375e+03	4.9405e−04	3.8794e−02
C06	**0.0000e+00**	**1.9903e−01**	1.0994e+02	1.7150e+02	0.0000e+00	4.5917e+00
C07	**−5.2474e+02**	**−5.2474e+02**	−4.2280e+02	−3.8719e+02	−5.2446e+02	−5.2152e+02
C08	1.2499e+00	2.6492e+00	5.3728e+00	1.2295e+01	**1.1618e+00**	**2.3740e+00**
C09	**−6.0917e−01**	**−6.0917e−01**	−6.0917e−01	1.8960e−01	**−6.0917e−01**	**−6.0917e−01**
C10	1.0175e+01	1.5753e+01	**7.7372e+00**	**1.5545e+01**	9.8156e+00	1.6622e+01

6. 结论

在过去的几十年里，许多进化算法被引入来解决约束优化问题。然而，在进化优化中缺乏有效的约束处理机制。在本研究中，基于约束一致策略的有效性，提出了一种新的人工蜂群算法。为了使计算时间最小化并保持种群内良好的多样性，对每代不可行个体采用基于约束一致方法的更新策略，剩余个体(包括可行个体和不可行个体)采用人工蜂群算法进行演化。

该算法在一组约束问题上进行了测试。实验还比较了 ABCCC 与 DECC 和 PSOCC 的性能。结果表明，在大多数情况下，ABCCC 算法比其他两种算法都能获得更好的结果和更快的收敛速度。该算法还与 CLSHADE、UDE 等算法进行了比较，实验结果表明，ABCCC 算法性能良好，能得到最优解。此外，与采用 CC 策略的 ABC 的变体相比，我们的 ABCCC 算法的结果优于其他算法。

在未来的工作中，我们还将分析不同参数值对该算法性能的影响，并尝试进一步改进所提出的算法。

参考文献

[1] Antoniou, A. and Lu, W.S. (2010) Practical Optimization: Algorithms and Engineering Applications. In: *Practical Optimization*: *Algorithms and Engineering Applications*, Springer Publishing Company, Incorporated.

[2] Karaboga, D. and Basturk, B. (2008) On the Performance of Artificial Bee Colony (ABC) Algorithm. *Applied Soft Computing*, **8**, 687-697. https://doi.org/10.1016/j.asoc.2007.05.007

[3] Smith, L., Chinneck, J. and Aitken, V. (2013) Improved Constraint Consensus Methods for Seeking Feasibility in Nonlinear Programs. *Computational Optimization & Applications*, **54**, 555-578. https://doi.org/10.1007/s10589-012-9473-z

[4] Zamuda, A. (2017) Adaptive Constraint Handling and Success History Differential Evolution for CEC 2017 Constrained Real-Parameter Optimization. 2017 *IEEE Congress on Evolutionary Computation* (*CEC*), San Sebastian, 5-8 June 2017, 2443-2450. https://doi.org/10.1109/CEC.2017.7969601

[5] Trivedi, A., Sanyal, K., Verma, P., *et al.* (2017) A Unified Differential Evolution Algorithm for Constrained Optimization Problems. 2017 *IEEE Congress on Evolutionary Computation* (*CEC*), San Sebastian, 5-8 June 2017, 1231-1238. https://doi.org/10.1109/CEC.2017.7969446

[6] Gao, W., Liu, S. and Huang, L. (2013) A Novel Artificial Bee Colony Algorithm Based on Modified Search Equation and Orthogonal Learning. *IEEE Transactions on Cybernetics*, **43**, 1011-1024. https://doi.org/10.1109/TSMCB.2012.2222373

[7] Zhu, G. and Kwong, S. (2010) Gbest-Guided Artificial Bee Colony Algorithm for Numerical Function Optimization. *Applied Mathematics & Computation*, **217**, 3166-3173. https://doi.org/10.1016/j.amc.2010.08.049

[8] Smith, L., Chinneck, J. and Aitken, V. (2013) Constraint Consensus Concentration for Identifying Disjoint Feasible Regions in Nonlinear Programmes. *Optimization Methods and Software*, **28**, 339-363. https://doi.org/10.1080/10556788.2011.647818

[9] Hamza, N.M., Essam, D.L. and Sarker, R.A. (2015) Constraint Consensus Mutation Based Differential Evolution for Constrained Optimization. *IEEE Transactions on Evolutionary Computation*, **20**, 447-459. https://doi.org/10.1109/TEVC.2015.2477402

[10] Hamza, N.M., Elsayed, S.M., Essam, D.L., *et al*. (2011) Differential Evolution Combined with Constraint Consensus for Constrained Optimization. *IEEE Congress on Evolutionary Computation*, **30**, 865-872. https://doi.org/10.1109/CEC.2011.5949709

[11] Ibrahim, W. and Chinneck, J.W. (2008) Improving Solver Success in Reaching Feasibility for Sets of Nonlinear Constraints. *Computers & Operations Research*, **35**, 1394-1411. https://doi.org/10.1016/j.cor.2006.08.002

[12] El-Abd, M. (2010) A Cooperative Approach to the Artificial Bee Colony Algorithm. *Proceedings of the IEEE Congress on Evolutionary Computation CEC* 2010, Barcelona, Spain, 18-23 July 2010, 1-5. https://doi.org/10.1109/CEC.2010.5586007

[13] Ma, L., Hu, K., Zhu, Y., *et al.* (2014) Cooperative Artificial Bee Colony Algorithm for Multi-Objective RFID Network Planning. *Journal of Network and Computer Applications*, **42**, 143-162. https://doi.org/10.1016/j.jnca.2014.02.012

[14] Wu, G.H., Mallipeddi, R. andSuganthan, P.N. (2010) Problem Definitions and Evaluation Criteria for the CEC 2017 Competition on Constrained Real-Parameter Optimization Technology.