

混合多约束处理技术的并行约束差分进化算法

魏文红*

(东莞理工学院 计算机学院, 广东 东莞 523808)

(* 通信作者电子邮箱 weihw@dgut.edu.cn)

摘要:针对约束差分进化算法中单一约束处理技术无法适合所有优化问题的情况,提出了一种混合多种约束处理技术的并行约束差分进化算法。该算法将种群分成多个子种群,各子种群采用不同的约束处理技术并行地独立进化,在适应值评价时进行种群间的通信交流。通过混合4种约束处理技术,使得算法对于所有测试函数都能成功地寻找到最优解,而且运算时间是串行算法的1/4。实验结果表明:与相应的串行算法及采用单一约束处理技术的算法比较,所提算法具有更高的求解精度、更少的计算时间和更快的收敛速度。

关键词:约束处理技术;差分进化;约束优化;并行;收敛性

中图分类号: TP301.6 **文献标志码:** A

Parallel constrained differential evolution algorithm merging with multi-constraint handling techniques

WEI Wenhong*

(School of Computer, Dongguan University of Technology, Dongguan Guangdong 523808, China)

Abstract: Aiming at the problem that constrained differential evolution with single constraint handling technique is not suitable for all constrained optimization problems, a parallel constrained differential evolution algorithm using multi-constraint handling techniques was proposed. The algorithm divided an initial population into several sub-populations, and then the sub-populations evolved with different constraint handling techniques in parallel, they communicated with each other at fitness evaluation. By using four constraint handling techniques, the algorithm can find the best known optimization solution and compared with serial algorithm, the computation time is 1/4 while solving all benchmark functions. The experimental results show that the proposed algorithm is able to decrease computation time, and improve solution accuracy and convergence speed in the majority of test cases compared with corresponding serial algorithm and those algorithms which only use one constraint handling technique.

Key words: constraint handling technique; Differential Evolution (DE); constrained optimization; parallel; convergence

0 引言

许多的科学和工程问题都是约束优化问题(Constrained Optimization Problem, COP),存在着各种各样的约束条件^[1]。由于约束条件的出现,进化算法(Evolution Algorithm, EA)在求解约束优化问题时,会导致可行解区域减少并使得搜索解的过程变得更为复杂。通过加入约束处理技术后,进化算法可以成功地求解约束优化问题。

差分进化(Differential Evolution, DE)算法^[2]是一种结构简单、功能强大、且鲁棒性强的全局优化算法,该算法已经成功地用于许多不同的应用领域。约束差分进化(Constrained DE, CDE)算法就是在DE算法中加入约束处理技术,以解决约束优化问题,如Storn^[3]提出的约束适应性差分进化算法。在CDE算法中,根据“没有免费的午餐”理论^[4],没有哪种单一的约束处理技术能够对所有的约束优化问题都具有良好的优化效果。这是因为不同的约束处理技术都存在着各自的优缺点,即某一类约束处理技术可能只适合于求解某些类型的约束优化问题,这使得混合多种约束处理技术求解约束优化

问题成为了可能。然而在CDE算法中同时使用多种约束处理技术势必会增加计算时间,因此如何在优化求解精度和计算时间方面取得一个良好的平衡将是一个亟待解决的问题。

随着并行计算技术的发展,并行DE算法也得到了快速的发展。一般情况下,并行DE算法有三个典型的计算模型^[5]:目标函数评价级的主从模型、种群级的迁移模型和个体级的细粒度模型。Lampinen^[6]提出了基于主从模型的并行DE算法,他把种群分成多个子种群,子种群独立进行进化,然后通过目标函数值进行评价;Zaharie等^[7]提出了基于迁移模型的并行DE算法,该算法也是将种群划分成多个子种群,然后构建一个虚拟拓扑,再按照该拓扑结构进行子种群间的迁移通信;Salomon^[8]提出了基于细粒度模型的并行DE算法,该算法把每个种群的个体分配到每个处理器,在处理器之间进行迭代通信。由于主从模型构建比较方便,直接通过目标函数值进行评价,所以目前所提出的并行DE算法大多数是基于该模型的,如多策略多参数的并行DE算法^[9]、多种群并行的自适应DE算法^[10]、改进的动态多种群并行DE算法^[11]、多种群多策略的并行DE算法^[12]。然而所有这些并行

收稿日期:2015-04-17;修回日期:2015-06-11。 基金项目:国家自然科学基金资助项目(61103037, 61300198);广东省自然科学基金资助项目(S2013010011858);广东省高校科技创新项目(2013KJJCX0178)。

作者简介:魏文红(1977-),男,江西南昌人,副教授,博士,CCF会员,主要研究方向:网络与并行分布计算、智能优化处理。

DE算法都是处理无约束优化问题的,对于约束优化问题的并行DE算法却少见报道。

在这种情况下,本文采用多种约束处理技术,结合并行计算技术提出了一种混合多约束处理技术的并行CDE(Multi-Constraint Handling Techniques Parallel CDE, MCHT-PCDE)算法。该算法既可以针对不同的约束优化问题混合采用多种约束处理技术以求解到最优解,又满足计算时间少的要求。

1 相关研究

1.1 约束优化问题

在约束优化问题中,不但包括等式约束条件和不等式约束条件,还包括变量 \mathbf{x} 的上界和下界^[3],具体如下:

$$\min f(\mathbf{x}) \quad (1)$$

$$\text{s. t. } g_j(\mathbf{x}) \leq 0; j = 1, 2, \dots, q \quad (2)$$

$$h_j(\mathbf{x}) = 0; j = q + 1, q + 2, \dots, m \quad (3)$$

$$\text{low}_i \leq x_i \leq \text{upper}_i; i = 1, 2, \dots, n \quad (4)$$

其中: $\mathbf{x} = (x_1, x_2, \dots, x_n)$ 是一个 n 维决策向量, $f(\mathbf{x})$ 为目标函数, $g_j(\mathbf{x}) \leq 0$ 和 $h_j(\mathbf{x}) = 0$ 分别表示 q 个不等式约束和 $m - q$ 个等式约束;函数 f, g_j 和 h_j 为线性或非线性的实数函数; upper_i 和 low_i 分别为向量 \mathbf{x}_i 的上界和下界。另外,假定可行解空间中满足所有约束的点集合用 U 表示,搜索空间中满足上界和下界约束的点集合用 S 表示,其中 $S \subset U$ 。

一般地,在约束进化算法中,等式约束经常需要转化成不等式约束,具体如下:

$$|h_j(\mathbf{x})| - \delta \leq 0 \quad (5)$$

其中: $j \in \{q + 1, q + 2, \dots, m\}$; δ 为等式约束违反的容忍因子,一般取正整数。解 \mathbf{x} 到第 j 个约束的距离可以表示为:

$$G_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\}, & 1 \leq j \leq q \\ \max\{0, |h_j(\mathbf{x})| - \delta\}, & q + 1 \leq j \leq m \end{cases} \quad (6)$$

那么解 \mathbf{x} 到可行解区域的边界距离,即约束违反程度可以表示为:

$$v(\mathbf{x}) = \sum_{j=1}^m w_j (G_j(\mathbf{x})) / \sum_{j=1}^m w_j \quad (7)$$

其中: $w_j = 1/G_{\max_j}$ 为一个权值参数, G_{\max_j} 为 $G_j(\mathbf{x})$ 的最大值。

1.2 差分进化算法

在DE算法中,一般包括 NP 个种群,个体向量的维度为 n 。一般地,个体表示为: $\mathbf{x}_{i,t} = (x_{i,1,t}, x_{i,2,t}, \dots, x_{i,n,t})$ 。其中: $i = 1, 2, \dots, NP$, NP 为种群大小; t 为当前种群的代数。DE算法包括3个主要的操作^[2]。

1) 变异。

在变异操作中,对于每个种群产生目标向量 $\mathbf{v}_{i,t}$,变异策略主要如下:

$$\text{rand}/1: \mathbf{v}_{i,t} = \mathbf{x}_{r1,t} + F \cdot (\mathbf{x}_{r2,t} - \mathbf{x}_{r3,t}) \quad (8)$$

$$\text{best}/1: \mathbf{v}_{i,t} = \mathbf{x}_{\text{best},t} + F \cdot (\mathbf{x}_{r1,t} - \mathbf{x}_{r2,t}) \quad (9)$$

current-to-best/1:

$$\mathbf{v}_{i,t} = \mathbf{x}_{i,t} + F \cdot (\mathbf{x}_{\text{best},t} - \mathbf{x}_{i,t}) + F \cdot (\mathbf{x}_{r1,t} - \mathbf{x}_{r2,t}) \quad (10)$$

$$\text{best}/2: \mathbf{v}_{i,t} = \mathbf{x}_{\text{best},t} + F \cdot (\mathbf{x}_{r1,t} - \mathbf{x}_{r2,t}) + F \cdot (\mathbf{x}_{r3,t} - \mathbf{x}_{r4,t}) \quad (11)$$

$$\text{rand}/2: \mathbf{v}_{i,t} = \mathbf{x}_{r1,t} + F \cdot (\mathbf{x}_{r2,t} - \mathbf{x}_{r3,t}) + F \cdot (\mathbf{x}_{r4,t} - \mathbf{x}_{r5,t}) \quad (12)$$

其中: r_1, r_2, r_3, r_4 和 r_5 都是随机从集合 $\{1, 2, \dots, NP\} \setminus \{i\}$ 中选取的; $\mathbf{x}_{\text{best},t}$ 为种群在第 t 代最好的个体;缩放因子 F 为实

数, $F \in [0, 1]$ 。

2) 交叉。

交叉操作主要产生实验向量 $\mathbf{u}_{i,t}$,具体如下:

$$\mathbf{u}_{i,t} = \begin{cases} \mathbf{v}_{i,t}, & \text{rand}_j(0, 1) \leq CR \text{ or } j = j_{\text{rand}} \\ \mathbf{x}_{i,t}, & \text{其他} \end{cases} \quad (13)$$

其中: $i = 1, 2, \dots, NP$; $j = 1, 2, \dots, n$; j_{rand} 是属于从1到 n 之间的一个随机整数; $\text{rand}_j(0, 1)$ 为对于每个 j 产生 $[0, 1]$ 均匀分布的随机数。使用参数 j_{rand} 是为了保证实验向量 $\mathbf{u}_{i,t}$ 不同于目标向量 $\mathbf{x}_{i,t}$ 。交叉概率因子 CR 的取值通常在 $0 \sim 1$ 。

3) 选择。

在选择操作中,目标向量 $\mathbf{x}_{i,t}$ 和实验向量 $\mathbf{u}_{i,t}$ 根据它们的适应值进行比较,选择适应值更优的进入下一代种群:

$$\mathbf{x}_{i,t+1} = \begin{cases} \mathbf{u}_{i,t}, & f(\mathbf{u}_{i,t}) \leq f(\mathbf{x}_{i,t}) \\ \mathbf{x}_{i,t}, & \text{其他} \end{cases} \quad (14)$$

1.3 约束处理技术

1) 可行解优先(Superiority of Feasible Solutions, SF)^[13]。

在约束处理技术SF中,如果两个解,一个是可行的,一个是不可行的,则选择可行解进入迭代;如果两个解都是可行的,则比较它们的目标函数值;如果两个解都是不可行的,则比较它们的约束违反程度。通过约束违反程度的比较,可以慢慢地从不可行解区域向可行解区域进化。种群采用SF约束处理技术进化时,个体的生存选择依据此原则进行。

2) 自适应惩罚函数(Self-adaptive Penalty, SP)^[14]。

自适应惩罚函数的方法就是在不可行个体中针对两种情况增加惩罚值就可以区分出哪些是好的不可行个体。这两种情况是,如果种群中只有少量的可行个体,那么约束违反程度高的不可行个体应增加更高的惩罚值;相反,如果种群中存在大量的可行个体,那么具有更好目标函数值的不可行个体就增加更低的惩罚值。即增加的惩罚值由种群中可行个体的数量决定。种群采用SP约束处理技术进化时,所有个体的生存选择都由增加惩罚值之后的适应值决定。

3) ε 约束(ε -Constraint, EC)^[15]。

ε 约束处理技术是通过 ε 参数来控制约束操作,它在等式约束优化问题中,能够搜索到高质量的解。通过 $\varepsilon(k)$ 来判断某个解是否为可行解,即当个体的约束违反程度低于 $\varepsilon(k)$ 时,则认为它是可行解;否则就认为它是不可行解。之后的选择方式就类似于SF约束处理技术。

4) 随机排序(Stochastic Ranking, SR)^[16]。

约束处理技术SR可以通过随机选择的方式以达到目标函数值和约束违反程度的平衡,该方法采用概率 p_m 来决定是采用个体的目标函数值排序还是采用个体的约束违反程度排序。在个体的生存选择时,开始阶段设置 $p_m = 0.0475$,然后线性下降,到最后 $p_m = 0.0025$ 。最终按照排序的位置选择进入下一代的个体。

此4种约束处理技术也存在各自的优缺点,具体如下:

1) 可行解优先的约束处理技术同时考虑了目标函数和约束违反程度,对可行解和不可行解区别对待,提高了算法的收敛速度。但该方法的缺点是忽略了不可行解的作用,特别是当种群中的绝大部分个体均为可行解时,不可行解将很难进入种群,从而忽略了种群的多样性,容易陷入局部最优。

2) 虽然自适应惩罚函数的约束处理技术操作简单,但它也存在一些缺点,主要是惩罚值的自适应设置比较复杂。虽

然惩罚值可以根据种群中可行解的数量动态地进行设置,但在具体的操作过程中,却必须要通过多次实验来不断地进行调整,以观察出它们之间的相应关系。

3) ε 约束的约束处理技术其实是在 SF 约束处理技术基础之上发展而来,它基本上保持了 SF 约束处理技术的优点,同时又考虑了不可行解的作用,但对于 ε 的自适应调整也是一个比较复杂的操作,会带来一定的计算时间复杂度。

4) 随机排序的约束处理技术以最简单的方式达到目标函数值和约束违反程度的平衡,但是该方法由于随机性过强,往往会造成算法的收敛速度过慢且容易陷入局部最优。

可行解优先的约束处理技术对于单模的约束优化问题的求解比较有优势,自适应惩罚函数和 ε 约束的约束处理技术比较擅长多模和非线性约束优化问题的求解,随机排序的约束处理技术对于求解线性的约束优化问题有一定的优势。

综上所述,由于这些约束处理技术优缺点的存在,混合多种约束处理技术来解决约束优化问题能够使算法的性能更加稳定。

2 多约束处理技术的并行 CDE 算法

2.1 算法思想

MCMT-PCDE 的算法思想是同时采用 SF、SP、EC 和 SR 4 种约束处理技术加入到 CDE 算法中,并使其并行化。该算法把种群划分为 4 个子种群,每个子种群分别采用 SF、SP、EC 和 SR 约束处理技术并行地选择最优的个体进入子代。由于每个子种群间约束处理技术的不同,因此会出现这样一种情况,即:在某个子种群迭代过程中评价很差的个体,却是别的子种群中评价很好的个体。此时,子种群间并行进化的同时,相互通信是有必要的。通过子种群间的通信,规约收集每个子种群的子代个体,并分别广播给其他子种群。然后每个子种群分别采用各自的约束处理技术进行选择,生成新的子种群。MCMT-PCDE 算法一方面通过多约束处理技术保证了种群进化的多样性,另一方面又采用并行计算技术有效地减少了计算时间。

2.2 算法步骤

根据算法思想,MCMT-PCDE 算法的具体步骤如下:

Step1 分别在 4 个处理器上随机生成 4 个大小为 NP 的 n 维种群 $POP_i (i = 1, 2, 3, 4)$,并为每个子种群分配一种约束处理技术(SF、SP、EC 和 SR),设置相应的参数,设置当前代数 $k = 0$ 。

Step2 在每个处理器上对每个子种群 POP_i 中的个体进行评价,求出相应的目标函数值和约束违反程度。

Step3 在每个处理器上根据违反约束情况更新子种群中相应的约束处理参数。

Step4 在每个处理器上子种群 POP_i 通过变异和交叉操作产生子代种群 $OFFS_i$ 。

Step5 在每个处理器上对每个子代种群 $OFFS_i$ 进行评价,求出相应的目标函数值和约束违反程度,并保存该结果。

Step6 在某个处理器上对每个子种群 POP_i 进行规约通信,收集所有子代种群 $OFFS_i$,再把它们广播给其他所有处理器。

Step7 位于每个处理器上的子种群 POP_i 合并通信传递过来的所有子代种群 $OFFS_i$ 。

Step8 在每个处理器上分别通过各自的约束处理技术从合并的种群中选择最优的个体组成新的子种群 POP_i 。

Step9 如果中止条件满足,则选择输出最优的种群个体 POP ;否则 $k = k + 1$,跳转到 Step 3。

MCMT-PCDE 算法流程如图 1 所示。



图 1 MCMT-PCDE 算法流程

2.3 时间复杂度分析

因为原始 DE 的时间复杂为 $O(G \times NP \times n)$, 其中: G 表示迭代的代数, NP 表示种群数, n 表示个体的维度。由于约束处理技术都是针对种群个体的操作, 这些操作的时间复杂度为 $O(NP \times n)$, 所以加上约束处理技术的 CDE 算法并没有增加算法的时间复杂度。另外由于 MCMT-PCDE 属于并行算法, 设种群间的通信时间为 T , 所以在算法 Step 6 中 4 个子种群的广播通信时间复杂度为 $O(4 \times 3 \times T) = O(T)$ 。当算法在多核处理器上运行时, $O(T)$ 可以忽略不计。综上所述, MCMT-PCDE 算法的时间复杂仍然为 $O(G \times NP \times n)$ 。与单一约束处理技术的 CDE 算法相比, 虽然它们在数量级上是相同的, 但是由于 G 未必相同(即混合多种约束处理技术的 CDE 算法不用进化到最大代数就能找到最优解, 而单一约束处理技术的 CDE 可能必须进化到最大代数才能找到最优解), 所以会出现 MCMT-

PCDE 算法执行时间可能比某些单一约束处理技术的 CDE 算法执行时间更少的情况。与串行的混合 4 种约束处理技术的 CDE 算法相比,由于 MCHT-PCDE 算法采用 4 个处理器同时执行,而串行的混合 4 种约束处理技术的 CDE 算法只在一个处理器上运行,如果除去通信开销的时间,那么 MCHT-PCDE 算法在理想状态下的运行效率是能够高近 4 倍,而实际上应该是 3 倍多。

3 实验与分析

本文对 MCHT-PCDE 算法和当前几个比较著名的 CDE 算法(如 rank iMDDE^[17]、ISAMODE-CMA^[18]、SAMODE^[19]、ATMES^[20]、SMES^[21])进行了实验对比测试,测试函数为 CEC'2006 中的 13 个 Benchmark 函数^[22]。实验环境为 64 位的 Windows 7 系统,其中 CPU 为 Intel Core TM 2.8 GHz,4 核处理器,16 GB 内存。编程环境为 Matlab R2013b 和并行计算工具箱,该工具箱提供了并行 Matlab 编程的所有函数和语句。另外,也对 MCHT-PCDE 算法和分别由 4 种约束处理技术对应构成的串行 CDE 算法(如 SF-CDE、SP-CDE、EC-CDE 和 SR-CDE)及混合 4 种约束处理技术的串行 CDE 算法(MCHT-CDE)在相同的实验环境下进行了实验比较。

设置种群大小 $NP = 50$,交叉概率 $CR = 0.7$,缩放因子 $F = 0.9$ 。所有的算法对于每一个测试函数都独立运行 30 次,设置最大函数评价次数 $Max_FEs = 2.4 \times 10^5$ 。

1) MCHT-PCDE 与其他著名 CDE 算法的比较。

MCHT-PCDE 和 rank iMDDE、ISAMODE-CMA、SAMODE、ATMES、SMES 等算法的实验测试结果显示在表 1 中,由于评价 CDE 算法的优劣主要取决于测试结果的 Mean 值,所以表 1 只给出测试结果的平均值及标准差。从表 1 中的结果来看,MCHT-PCDE 求得了所有函数的最优解,而且比较明显地优于其他所有 CDE 算法。

2) MCHT-PCDE 与相应串行算法的比较。

MCHT-PCDE 和 SF-CDE、SP-CDE、EC-CDE、SR-CDE 及 MCHT-CDE 等串行算法的实验比较结果如表 2 所示。

从表 2 可看出,MCHT-PCDE 和 MCHT-CDE 在求解精度上明显优于 SF-CDE、SP-CDE、EC-CDE、SR-CDE,因为 MCHT-PCDE 和 MCHT-CDE 都混合了 4 种约束处理技术,比使用单一约束处理技术的 SF-CDE、SP-CDE、EC-CDE 和 SR-CDE 有明显的优势。但是 MCHT-CDE 在计算时间上却明显比 SF-CDE、SP-CDE、EC-CDE 和 SR-CDE 高很多,而 MCHT-PCDE 由于采用并行计算技术,克服了 MCHT-CDE 计算时间长的缺点,却又保持了多约束处理技术求解精度高的优点。另外可以注意到,MCHT-PCDE 在某些测试函数上的计算时间比 SF-CDE、SP-CDE、EC-CDE、SR-CDE 还要低(如 g_{01} 、 g_{05} 、 g_{07} 、 g_{12} 和 g_{13}),这主要是因为 MCHT-PCDE 采用多约束处理技术后,不用进化到最大代数就能找到最优解,即收敛性更好。本文没有分析 MCHT-PCDE 相对于 MCHT-CDE 的加速比,这是因为首先算法的计算量大,而通信时间占算法本身的运行时间很少,可以忽略不计。其次 Matlab 运行在 4 核处理器上,通信

表 1 MCHT-PCDE 与其他著名 CDE 算法实验结果比较

函数	适应值	最优值	MCHT-PCDE	rank-iMDDE	ISAMODE-CMA	SAMODE	ATMES	SMES
	平均值	-15	-15	-15	-15	-15	-15	-15
g_{01}	标准差	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.60E-14	0.00E+00
g_{02}	平均值	-0.8036191	-0.803619085	-0.80202119	-0.79244	-0.7987352	-0.790148	-0.785238
	标准差	0.00E+00	2.61E-02	1.11E-02	2.80E-02	8.80E-03	1.30E-02	1.67E-02
g_{03}	平均值	-1.0005	-1.0005001	-1.0005001	-1.0005	-1.0005	-1	-1
	标准差	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	5.90E-05	2.09E-05
g_{04}	平均值	-30665.5387	-30665.53867	-30665.539	-30665.539	-30665.539	30665.539	-30665.539
	标准差	0.00E+00	5.04E-12	7.31E-12	0.00E+00	0.00E+00	0.00E+00	0.00E+00
g_{05}	平均值	5126.4967	5126.496714	5126.496714	5126.496714	5126.497	5127.648	5174.492
	标准差	0.00E+00	9.70E-13	7.31E-12	0.00E+00	0.00E+00	1.80E+00	5.01E+01
g_{06}	平均值	-6961.8139	-6961.813876	-6961.81388	-6961.813875	-6961.814	-6961.814	-6961.814
	标准差	0.00E+00	3.71E-12	1.37E-11	0.00E+00	0.00E+00	4.60E-12	1.85E+00
g_{07}	平均值	24.3062	24.30620907	24.3062097	24.3062	24.3096	24.316	24.475
	标准差	0.00E+00	6.61E-12	7.74E-07	1.32E-03	1.59E-03	1.10E-02	1.32E-01
g_{08}	平均值	-0.09582504	-0.095825041	-0.09582504	-0.095825	-0.095825	-0.095825	-0.095825
	标准差	0.00E+00	4.91E-18	8.37E-17	0.00E+00	0.00E+00	2.80E-17	0.00E+00
g_{09}	平均值	680.630057	680.6300574	680.6300574	680.63	680.63	680.639	680.639
	标准差	0.00E+00	3.69E-13	2.29E-13	1.23E-06	1.16E-05	1.00E-02	1.55E-02
g_{10}	平均值	7049.2480	7049.248021	7049.248021	7049.24802	7059.81345	7250.437	7253.047
	标准差	0.00E+00	3.08E-12	1.50E-05	5.42E-06	7.86E+00	1.20E+02	1.36E+02
g_{11}	平均值	0.7499	0.7499	0.7499	0.7499	0.7499	0.75	0.75
	标准差	0.00E+00	1.13E-16	1.79E-15	1.31E-08	0.00E+00	3.40E-04	1.52E-04
g_{12}	平均值	-1	-1	-1	-1	-1	-1	-1
	标准差	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E-03	0.00E+00
g_{13}	平均值	0.053941514	0.053941514	0.053941514	0.053942	0.053942	0.053959	0.166385
	标准差	0.00E+00	1.53E-17	3.04E-07	2.21E-06	1.75E-08	1.30E-05	1.77E-01

注:粗体表示最优值。

在 CPU 内部进行,数据传递速度之快以至于通信基本上可以忽略,所以 MCHT-PCDE 算法的加速比和效率都比较高。

SR-CDE 在测试函数 g_{01} 和 g_{02} 上的收敛图。从图 2 可看出, MCHT-PCDE 的收敛性都明显优于 SF-CDE、SP-CDE、EC-CDE 和 SR-CDE。

表 2 MCHT-PCDE 与相应串行算法实验结果比较

函数	适应值	MCHT-PCDE	SF-CDE	SP-CDE	EC-CDE	SR-CDE	MCHT-CDE
g_{01}	平均值	-15(1)	-12.899 973 62(6)	-14.142 139 52(4)	-14.180 986 15(3)	-13.299 365 06(5)	-15(1)
	标准差	0.00E+00	1.95E+00	1.41E+00	1.37E+00	1.29E+00	0.00E+00
	计算时间/s	35.048 104 87	35.725 984 73	37.083 458 92	40.630 920 48	38.585 779 14	134.543 863 6
g_{02}	平均值	-0.803 619 085(1)	-0.328 952 458(5)	-0.337 076 244(4)	-0.344 228 256(3)	-0.178 196 068(6)	-0.803 619 085(1)
	标准差	2.61E-02	3.84E-02	3.64E-02	1.78E-02	4.69E-02	2.61E-02
	计算时间/s	52.644 883 07	45.732 921 13	43.796 499 12	51.287 477 13	49.813 991 04	191.371 381 1
g_{03}	平均值	-1.000 500 1(1)	-0.002 316 175(5)	-0.002 383 91(4)	-0.013 432 848(3)	-7.758 7E-05(6)	-1.000 500 1(1)
	标准差	0.00E+00	1.18E-02	1.04E-02	3.35E-04	3.74E-02	0.00E+00
	计算时间/s	38.357 410 66	35.799 246 97	35.292 528 95	36.726 868 48	36.593 747 98	147.008 171
g_{04}	平均值	-30 665.538 67(1)	-30 051.967 21(5)	-30 345.554 17(4)	-30 023.112 44(6)	-30 609.772 6(3)	-30 665.538 67(1)
	标准差	5.04E-12	8.80E+02	7.76E+02	4.28E+01	8.26E+02	1.19E-11
	计算时间/s	38.569 741 27	32.701 670 83	34.534 339 39	33.230 766 53	34.940 972 96	139.305 795
g_{05}	平均值	5 126.496 714(1)	5 355.952 294(6)	5 153.768 265(4)	5 126.545 956(3)	5 251.682 697(5)	5 126.496 714(1)
	标准差	9.70E-13	9.39E+02	1.24E+03	2.01E+03	8.70E+02	9.70E-13
	计算时间/s	35.459 364 72	36.218 433 27	38.311 888 08	36.435 289 61	36.534 089 63	135.298 763 6
g_{06}	平均值	-6 961.813 876(1)	-6 961.813 876(1)	-6 953.873 731(4)	-6 610.470 974(5)	-5 481.342 003(6)	-6 961.813 876(1)
	标准差	3.71E-12	4.63E-12	5.29E+00	2.06E+03	1.10E+03	4.63E-12
	计算时间/s	36.517 615 97	33.192 648 84	36.227 908 27	34.163 501 73	35.565 288 28	140.755 997 7
g_{07}	平均值	24.306 209 07(1)	24.318 061 2(3)	24.330 829 18(5)	24.320 489 77(4)	25.651 756 23(6)	24.306 209 07(1)
	标准差	6.61E-12	3.91E-03	2.20E-02	3.09E-01	5.41E-03	3.36E-11
	计算时间/s	33.728 587 22	33.877 786 75	36.789 987 46	34.012 648 46	36.556 119 14	126.688 272 1
g_{08}	平均值	-0.095 825 041(1)	-0.095 825 041(1)	-3.993 06E-06(6)	-0.095 825 041(1)	-0.095 825 041(1)	-0.095 825 041(1)
	标准差	4.91E-18	2.88E-17	1.57E-02	3.05E-17	3.10E-17	3.41E-17
	计算时间/s	31.706 042 48	27.789 836 53	35.367 365 46	27.960 499 94	27.857 400 95	118.516 219 5
g_{09}	平均值	680.630 057 4(1)	680.630 057 4(1)	680.630 057 4(1)	680.630 057 4(1)	680.638 587 9(6)	680.630 057 4(1)
	标准差	3.69E-13	7.83E-10	5.86E-10	3.15E-03	2.06E-09	4.49E-13
	计算时间/s	37.195 315 14	31.601 711 48	35.361 458 39	32.747 298 79	34.631 139 97	140.559 100 9
g_{10}	平均值	7 049.248 021(1)	7 050.906 119(3)	7 412.028 908(5)	7 051.516 645(4)	7 709.014 218(6)	7 049.248 021(1)
	标准差	3.08E-12	8.65E-01	1.02E+02	1.55E+02	7.45E-01	6.74E-12
	计算时间/s	37.102 654 55	31.470 354 92	37.823 296 41	32.797 330 33	35.219 155 41	141.914 693 3
g_{11}	平均值	0.749 9(1)	0.980 024 338(4)	0.992 692 387(5)	1(6)	0.953 800 077(3)	0.749 9(1)
	标准差	1.13E-16	6.42E-02	4.00E-02	9.47E-02	0.00E+00	1.13E-16
	计算时间/s	37.070 085 48	34.064 123 38	36.649 857 55	33.212 641 52	33.654 445 46	140.087 019 4
g_{12}	平均值	-1(1)	-1(1)	-1(1)	-1(1)	-1(1)	-1(1)
	标准差	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
	计算时间/s	37.155 510 9	47.739 395 84	48.307 889 64	44.484 471 69	45.203 730 61	138.058 280 9
g_{13}	平均值	0.053 941 514(1)	0.673 425 07(6)	0.659 248 544(5)	0.526 365 231(4)	0.488 713 944(3)	0.053 941 514(1)
	标准差	1.53E-17	3.53E-01	3.60E-01	3.92E-01	3.59E-01	1.69E-17
	计算时间/s	34.006 304 45	35.279 562 16	37.287 937 93	33.630 085 71	34.425 978 01	128.456 655 2

注:粗体表示最优值,括号中的数字为算法的排名。

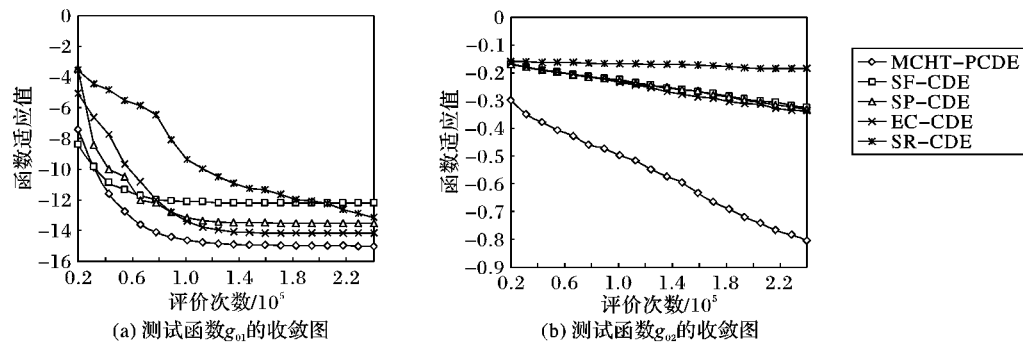


图 2 收敛性比较

4 结语

在进化算法中,差分进化算法结构简单、鲁棒性强。通过在差分进化算法中加入约束处理技术,可以用来解决约束优化问题。然而每种约束处理技术都存在各自的优缺点,因此同时混合使用几种约束处理技术可以相互弥补各自的优缺点。另外,由于每种约束处理技术在差分进化算法中具有较强的独立性,这使得将其并行化成为可能。因此本文结合并行化的思想,混合四种约束处理技术提出了 MCHT-PCDE 算法,该算法把种群划分成四个子种群,分别采用四个不同的约束处理技术进行独立进化,并在适应值评价的时候进行规约通信。最后针对 CEC 2006 中的 13 个 Benchmark 函数进行了实验测试,实验结果表明:与其他 CDE 算法相比, MCHT-PCDE 算法既提高了求解精度,又减少了计算时间,而且收敛性也更好。

参考文献:

- [1] RUNARSSON T P, YAO X. Search biases in constrained evolutionary optimization [J]. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2005, 35(2): 233 – 243.
- [2] STORN R, PRICE K. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces [J]. *Journal of Global Optimisation*, 1997, 11(4): 341 – 359.
- [3] STORN R. System design by constraint adaptation and differential evolution [J]. *IEEE Transactions on Evolutionary Computation*, 1999, 3(1): 22 – 34.
- [4] WOLPERT D H, MACREADY W G. No free lunch theorems for optimization [J]. *IEEE Transactions on Evolution Computing*, 1997, 1(1): 67 – 82.
- [5] TOMASSINI M. Parallel and distributed evolutionary algorithms: a review in evolutionary algorithms [J]. *Engineering and Computer Science*, 1999: 113 – 133.
- [6] LAMPINEN J. Differential evolution-new naturally parallel approach for engineering design optimization [J]. *Developments in Computational Mechanics with High Performance Computing*, 1999: 217 – 228.
- [7] ZAHARIE D, PETCU D. Parallel implementation of multi-population differential evolution [EB/OL]. [2015-04-10]. http://www.ieat.ro/wp-content/uploads/2012/09/technical_reports/DZ-PD.pdf.
- [8] SALOMON M. Parallelisation de l'évolution différentielle pour le recalage rigide d'images médicales volumiques [J]. *Technique et Science Informatiques*, 2001, 20(5): 605 – 627.
- [9] QU F, HU Y, YANG Y, *et al.* Differential evolution algorithm with different strategies and control parameters [J]. *Journal of Computer Applications*, 2011, 31(11): 3097 – 3100. (曲福恒, 胡雅婷, 杨勇, 等. 多策略多参数并行差分进化算法[J]. 计算机应用, 2011, 31(11): 3097 – 3100.)
- [10] GE Y, JIN W, GAO L, *et al.* An adaptive differential evolution algorithm based on a multi-population parallel [J]. *Journal of Northeastern University: Natural Science*, 2011, 32(4): 481 – 484. (葛延峰, 金文静, 高立群, 等. 多种群并行的自适应差分进化算法[J]. 东北大学学报: 自然科学版, 2011, 32(4): 481 – 484.)
- [11] LONG W. Dynamic multi-species parallel differential evolution algorithm [J]. *Application Research of Computers*, 2012, 29(7): 2429 – 2431. (龙文. 一种改进的动态多种群并行差分进化算法[J]. 计算机应用研究, 2012, 29(7): 2429 – 2431.)
- [12] CHEN Y, LIN Y, HU X. Parallel differential evolution with multi-population and multi-strategy [J]. *Journal of Frontiers of Computer Science and Technology*, 2014, 8(12): 1502 – 1510. (陈颖, 林盈, 胡晓敏. 多种群多策略的并行差分进化算法[J]. 计算机科学与探索, 2014, 8(12): 1502 – 1510.)
- [13] DEB K. An efficient constraint handling method for genetic algorithms [J]. *Computer Methods in Applied Mechanics and Engineering*, 2000, 186(2/3/4): 311 – 338.
- [14] TESSEMA B, YEN G G. A self-adaptive penalty function based algorithm for constrained optimization [C]// *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. Washington, DC: IEEE Computer Society, 2006: 246 – 253.
- [15] TAKAHAMA T, SAKAI S. Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites [C]// *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. Washington, DC: IEEE Computer Society, 2006: 1 – 8.
- [16] RUNARSSON T P, YAO X. Stochastic ranking for constrained evolutionary optimization [J]. *IEEE Transactions on Evolutionary Computation*, 2000, 4(3): 284 – 294.
- [17] GONG W, CAI Z, LIANG D. Engineering optimization by means of an improved constrained differential evolution [J]. *Computer Methods in Applied Mechanics and Engineering*, 2014, 268: 884 – 904.
- [18] HANSEN N, MULLER S D, KOUMOUTSAKOS P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES) [J]. *Evolutionary Computation*, 2003, 11(1): 1 – 18.
- [19] ELSAYED S M, SARKER R A, ESSAM D L. Multi-operator based evolutionary algorithms for solving constrained optimization problems [J]. *Computers and Operations Research*, 2011, 38(12): 1877 – 1896.
- [20] WANG Y, CAI Z, ZHOU Y, *et al.* An adaptive tradeoff model for constrained evolutionary optimization [J]. *IEEE Transactions on Evolutionary Computation*, 2008, 12(1): 80 – 92.
- [21] MEZURA-MONTES E, COELLO C. A simple multimembered evolution strategy to solve constrained optimization problems [J]. *IEEE Transactions on Evolutionary Computation*, 2005, 9(1): 1 – 17.
- [22] LIANG J J, RUNARSSON T P, MEZURA-MONTES E, *et al.* Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real parameter optimization [R]. Singapore: Nanyang Technological University, 2006.