DOI: 10.11992/tis.201706053

云环境下求解大规模优化问题的协同差分进化算法

谭旭杰[1], 邓长寿[1], 吴志健[2], 彭虎[1], 朱鹊桥[3]

（1. 九江学院 信息科学与技术学院, 江西 九江 332005; 2. 武汉大学 软件工程国家重点实验室, 湖北 武汉 430072; 3. 中国人民解放军 93704 部队）

摘　要: 差分进化是一种求解连续优化问题的高效算法。然而差分进化算法求解大规模优化问题时, 随着问题维数的增加, 算法的性能下降, 且搜索时间呈指数上升。针对此问题, 本文提出了一种新的基于 Spark 的合作协同差分进化算法 (SparkDECC)。SparkDECC 采用分治策略, 首先通过随机分组方法将高维优化问题分解成多个低维子问题, 然后利用 Spark 的弹性分布式数据模型, 对每个子问题并行求解, 最后利用协同机制得到高维问题的完整解。通过在 13 个高维测试函数上进行的对比实验和分析, 实验结果表明算法加速明显且可扩展性好, 验证了 SparkDECC 的有效性和适用性。

关键词: 差分进化; 大规模优化; 协同进化; 弹性分布式数据集; 云计算

中图分类号: TP301　　**文献标志码:** A　　**文章编号:** 1673−4785(2018)02−0243−11

中文引用格式: 谭旭杰, 邓长寿, 吴志健, 等. 云环境下求解大规模优化问题的协同差分进化算法[J]. 智能系统学报, 2018, 13(2): 243–253.

英文引用格式: TAN Xujie, DENG Changshou, WU Zhijian, et al. Cooperative differential evolution in cloud computing for solving large-scale optimization problems[J]. CAAI transactions on intelligent systems, 2018, 13(2): 243–253.

Cooperative differential evolution in cloud computing for solving large-scale optimization problems

TAN Xujie[1], DENG Changshou[1], WU Zhijian[2], PENG Hu[1], ZHU Queqiao[3]

(1. School of Information Science and Technology, Jiujiang University, Jiujiang 332005, China; 2. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China; 3. People's Liberation Army of China 93704)

Abstract: Differential evolution is an efficient algorithm for solving continuous optimization problems. However, its performance deteriorates quickly and the runtime grows exponentially when differential evolution is applied to solve large-scale optimization problems. To overcome this problem, a novel cooperative coevolution differential evolution based on Spark (called SparkDECC) was proposed. The strategy of separate processing is used in SparkDECC. Firstly, the large-scale problem is decomposed into several low-dimensional sub-problems by using the random grouping strategy; then each sub-problem can be tackled in a parallel way by taking advantage of the parallel computation capability of the resilient distributed datasets model in Spark; finally the optimal solution of the entire problem is obtained by using cooperation mechanism. The experimental results on 13 high-dimensional functions show that the new algorithm has good performances of speedup and scalability. The effectiveness and applicability of the proposed algorithm were verified.

Keywords: differential evolution; large-scale optimization; coevolution; resilient distributed dataset; cloud computing

差分进化算法 (differential evolution, DE) 是一种基于实数编码的全局优化算法[1], 因其简单、高效以及具有全局并行性等特点, 近年来已成功应用到工业设计和工程优化等领域。研究人员对 DE 算法进行了改进和创新并取得了一些成果。比如 Brest

收稿日期: 2017−06−13.

基金项目: 国家自然科学基金项目 (61364025, 61763019); 武汉大学软件工程国家重点实验室开放基金项目 (SKLSE2012-09-39); 九江学院科研项目 (2013KJ30, 2014KJYB032); 江西省教育厅科技项目 (GJJ161076, GJJ161072).

通信作者: 邓长寿. E-mail: csdeng@jju.edu.cn.

等[2]构造了控制参数的自适应性方法并提出了自适应 DE 算法 (jDE)。Wang 等[3]提出了复合 DE 算法 (CoDE), 其将精心选择的 3 种变异策略和 3 组控制参数按照随机的方法进行组合。这些研究成果主要集中于低维问题 (30 维), 然而当面向高维问题 (1 000 维) 时, 这些 DE 算法的性能将急剧下降, 而且搜索时间随着维数成指数增长, 求解极为困难, “维灾难”问题依然存在[4]。

为了有效求解高维优化问题, 学者们提出不同的策略, 其中具有代表性的是协同进化 (cooperative coevolution, CC)[5]。CC 采用“分而治之”的思想, 首先将高维复杂问题分解成低维简单的子问题; 其次对每个子问题分别进行求解; 最后通过所有子问题解的协同机制, 得到整个问题的解。Yang 等[6]提出的随机分组策略, 可将两个相关变量以极大的概率放在同一组, 在大规模优化问题中得到较精确的解。研究人员已将 CC 应用到多个领域, 如大规模黑盒优化问题[7]、SCA 问题[8]、FII 算法[9]、CCPSO 算法[10]、DG2 算法[11], 然而它们在求解高维优化问题时, 采用串行方式求解, 因此, 问题求解需要较长的计算时间, 很难在有效时间内提供满意的解。近年来云计算已应用到大规模信息处理领域中, 如在机器学习[12]、蚁群算法[13]、CRFs 算法[14]、差分进化算法[15-16]、图数据分析[17]、分类算法[18] 等获得了成功。因此, 有必要将云计算的分布式处理能力与 CC 的优势相结合, 为大规模优化问题的求解提供新方法。

研究人员已在 Google 开源平台 Hadoop 的 MapReduce 模型之上提出了一些分布式差分进化算法[19-20]。实践中研究人员发现, MapReduce 模型是一个通用的批处理计算模型, 缺乏并行计算数据共享的有效机制, 对迭代运算无法提供有效支持。因此, 基于 MapReduce 模型的差分进化算法需要通过频繁读写文件来交换数据, 降低了其效率[21]。

Spark 云平台是伯克利大学提出的分布式数据处理框架[22], 在许多领域获得了成功应用。Spark 提出了一种全新的数据抽象模型 RDD(弹性分布式数据模型)。RDD 模型能够有效支持迭代计算、关系查询、批处理以及流失数据处理等功能, 使得 Spark 可以应用于多种大规模处理场景。由于 RDD 模型基于内存计算, 避免了 MapReduce 模型频繁读写磁盘数据的弊端, 提高了效率。

本文基于 Spark 云平台提出了合作协同的差分进化算法。SparkDECC 算法采用“分治”策略, 将高维优化问题按随机分组策略分解成低维子问题, 并封装成 RDD; 在 RDD 中, 每个子问题独立并行进化若干代; 利用协同机制将所有子问题合并成完整问题, 评价出最优个体。SparkDECC 算法在 13 个标准函数进行测试, 实验结果表明该算法是有效可行的。

1　差分进化算法

差分进化算法是一种仿生的群体进化算法, 具体操作如下[23]。

1) 种群初始化

DE 算法首先在$[x_{\min}, x_{\max}]$范围内随机初始化 NP 个 D 维向量 $\boldsymbol{x}_i$ 的种群, 其中 NP 为种群大小, D 为种群的维度, $x_{\min}$ 为最小值, $x_{\max}$ 为最大值, $i \in [1, \mathrm{NP}]$。

2) 变异

变异主要是通过种群中个体之间的差向量来改变个体的值。DE 算法根据基准向量的选取和差分向量数目不同, 有多种变异操作。本文的目标向量 $\boldsymbol{x}_i$ 主要通过最常用的变异算子产生, 如式 (1)。

$$\boldsymbol{v}_{i,g} = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}) \tag{1}$$

式中: i, r_1, r_2, $r_3 \in [1, \mathrm{NP}]$且互不相同的 4 个随机整数; $\boldsymbol{v}_{i,g}$ 为目标向量 $\boldsymbol{x}_i$ 在第 g 代时产生的变异向量; 缩放因子 $F \in [0, 1]$。

3) 交叉

变异后, DE 算法通过交叉概率在目标向量 $\boldsymbol{x}_i$ 与变异向量 $\boldsymbol{v}_i$ 进行交叉, 产生新的试验向量 $\boldsymbol{u}_i$。具体操作如式 (2) 所示。

$$u_{i,j,g} = \begin{cases} v_{i,j,g}, & \mathrm{rnd}_1 < \mathrm{CR} \ \text{ or } j == \mathrm{rnd}_2 \\ x_{i,j,g}, & \text{其他} \end{cases} \tag{2}$$

式中: 交叉概率 $\mathrm{CR} \in [0, 1]$, $j \in [1, D]$, 随机数 $\mathrm{rnd}_1 \in [0, 1]$, 随机整数 $\mathrm{rnd}_2 \in [1, D]$。

4) 选择

DE 算法主要通过“贪婪”原则对个体进行选择, 较优个体进入下一代。具体操作如式 (3) 所示。

$$x_{i,g+1} = \begin{cases} u_{i,g}, & f(u_{i,g}) \leqslant f(x_{i,g}) \\ x_{i,g}, & \text{其他} \end{cases} \tag{3}$$

式中 $f(x_{i,g})$ 为个体 $x_{i,g}$ 的适应度值。

DE 算法通过变异、交叉、选择之后, 根据循环代数或求解精度来结束搜索。

2　合作协同的云差分进化算法 SparkDECC

2.1　Spark 云平台

为了更加高效地支持迭代运算, Spark 平台在 MapReduce 云模型的基础上进行了扩展[24]。Spark 平台提供了两个重要的抽象: RDD 和共享变量。RDD 本质是一个容错的、并行的数据结构, 提供了一种只读、分区记录并存放在内存的数据集合。

Broadcast 是一种共享变量, 将数据缓存在每个结点上, 不再需要传递数据, 减少通信开销, 提高通信性能。Spark 平台的 API 为 RDD 提供了两种算子: 转换算子 (Transformations) 和动作算子 (Actions), 其中, 转换算子都是惰性的, 对 RDD 分区的每个数据执行相同的操作, 并返回新的 RDD; 而动作算子将触发 RDD 上的运算, 并将值返回给主控结点。RDD 内部实现机制基于迭代器, 使得数据访问更高效, 避免了中间结果对内存的消耗, 使迭代计算更加高效快速。

DE 是一种基于群体的进化算法, 具有内在并行性的特点。因此, DE 能与 Spark 的并行性充分融合。Spark 在主控结点通过 Parallelize 方法将种群并行初始化, 并采用"键–值"的方式存放在内存中, 即:

$$[\text{key}_i, \text{value}_i], i = 1, 2, \cdots, m \tag{4}$$

式中: m 是子种群的数目; key_i 是整数, 表示第 i 个子种群的编号; value_i 是第 i 个子种群。DE 在 Spark 上的具体实现如图 1 所示。

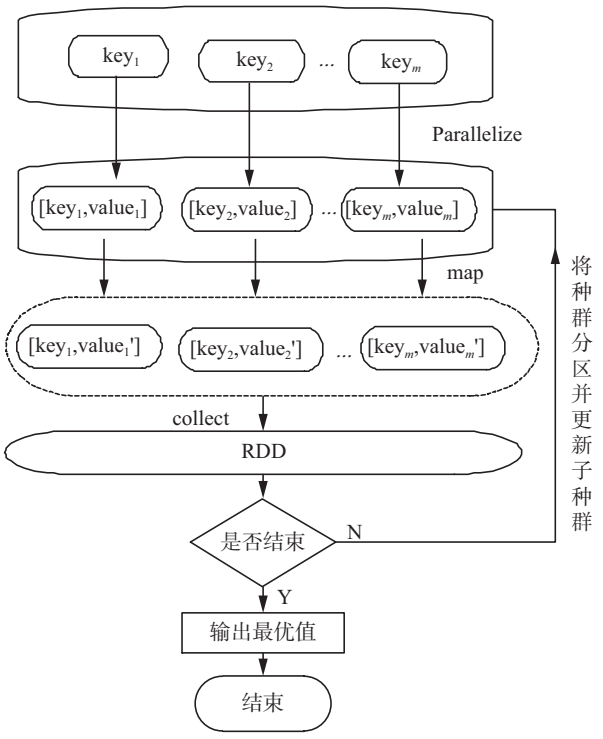

图 1　基于 Spark 的 DE 算法

Fig. 1　DE based on spark

Spark 将内存中的数据按"键–值"对的方式抽象成 RDD, 并根据 key_i 值将子种群分区保存在不同的结点上。利用 RDD 的并行操作算子对各子种群并行进化若干代, 再通过 collect 算子合并生成新的种群。循环结束后通过动作算子 reduce 获取整个种群的最优值。

2.2　SparkDECC 算法

CC 框架处理大规模优化问题优势明显。然而, 随着种群规模的增加, CC 框架所需时间快速增加。为了提高 CC 框架的收敛速度, 将云计算的优势与 CC 框架相结合, 提出了基于 Spark 的合作协同进化算法-SparkDECC 算法。

SparkDECC 算法首先将高维问题按随机分组方法分解成若干个低维子问题, 一个子问题对应一个子种群, 保留每个子问题在完整问题的位置信息; 按 key_i 值将低维子种群分发到 RDD 中相应的分区, 每个分区中的子种群并行执行 DE 算法的变异、交叉、选择等操作, 其中子种群在计算个体适应度值时, 选取上一轮的最优个体合作组成完整的种群并进行局部寻优。低维子种群在对应的分区中进化若干代后, 按照其位置信息合并成新的完整种群, 并通过全局搜索返回最优个体。SparkDECC 算法的流程图如图 2 所示。

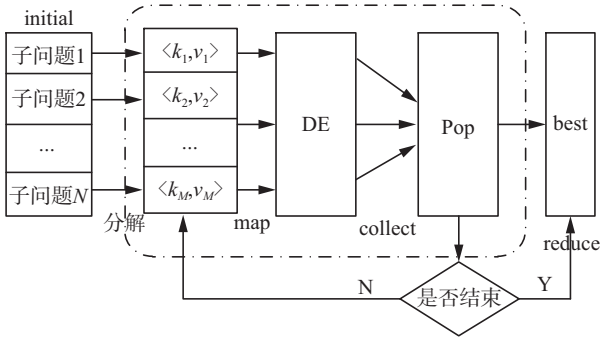

图 2　SparkDECC 算法流程图

Fig. 2　Flowchart of SparkDECC algorithm

SparkDECC 算法的具体步骤如下。

1) 初始化参数: 种群规模 NP、缩放因子 F、交叉概率 CR、问题的维度 D、子问题的维度 DimSub、子种群数 M=D/DimSub、子种群的位置信息 subscript、分区数 Num、进化代数 Gen、子种群合并轮数 Cycles;

2) 初始化种群 Pop, 通过 cache() 将数据存放在内存中;

3) 获取种群最优个体向量 bestInd、最优值 best;

4) 将控制合并轮数的变量 c 赋值为 0;

5) 判断合并轮数 c 是否小于等于 Cycles, 若是, 则循环结束; 否则, 继续;

6) 变量 c 自动加 1;

7) 利用 parallelize 方法将 Pop 按随机分组策略分解成 M 个低维的子种群, 将子种群的位置信息存放 subscript 中, 并按 key 值将子种群分发到相应分区;

8) 通过 broadcast 变量将 Pop 广播至每个结点;

9) 各子种群并行进化 Gen 代, 具体的计算过程为:

① 变异操作, 执行式 (1);

② 交叉操作, 执行式 (2);

③ 选择操作, 执行式 (3), 其中子种群在计算适应度值时, 按照其位置信息替换掉 bestInd 中的部分解。

10) 利用 collect 动作算子将所有低维子种群按协同机制合并成完整种群, 并更新 Pop;

11) 对 Pop 进行全局搜索并返回最优个体 bestInd;

12) 转到 5);

13) 循环结束, 通过 reduce 返回最优值 best。

在上述 SparkDECC 算法中, 包含两个循环, 即 6)、10)。外循环 6) 控制问题的全局优化, 内循环 10) 控制子问题的局部优化。

上述算法的时间复杂度主要集中在 10), 其主要功能是在 Num 个分区中同步并行优化 M 个低维子问题。一个低维子问题的时间复杂度为 $O(M\times NP)$, M 个低维子问题在一个分区里按串行方式进化 Gen 代, 运行 Cycles 轮的时间复杂度为 $O(M\times NP\times Gen\times Cycles)$。在上述算法中, 略去其他步的时间复杂度。因此, SparkDECC 算法在每个分区的时间复杂度为 $O(M\times NP\times Gen\times Cycles)/Num$。

3 数值实验与分析

3.1 测试问题

为了测试 SparkDECC 算法求解大规模优化问题的性能, 本文选取了文献[25]中 13 个测试函数进行实验。其中 f_1~f_8 为单模函数, f_9~f_{13} 为多模函数, f_4 和 f_5 为不可分解的函数, 其他函数都是可分解的。函数的详细说明详见文献[25]。

3.2 实验设置

本实验采用 Spark 云模型, 每个节点的配置如下: 64 bit corei7 CPU, 主频 3.4 GB, 内存 8 GB, Ubuntu13.10 操作系统, 安装使用 Hadoop2.2.0 和 Spark1.2.0, 编程环境为 IntellJ IDEA14.1.2, 使用 Scala 和 Java 语言。

为了验证 SparkDECC 的性能以及影响其性能的各个因素, 实验时 SparkDECC 中问题的维度 D=1 000, 子问题的维度 DimSub=100, 问题规模 NP=100, F=0.5, CR=0.9, 每个子种群独立运行的代数 MaxGen=100, 合并轮数 Cycles=50。为了验证 SparkDECC 算法的可扩展性, 将问题的维度 D 扩展到 5 000, 其他参数保持不变。

3.3 实验结果与分析

表 1 为 SparkDECC 与 OXDE[26]、CoDE、jDE、PSO[27]算法的平均最优值和标准方差的结果对比。5 种算法的参数设置一致, 各算法独立运行 25 次。采用了 Wilcoxon 秩和检验方法对 4 种算法的实验结果进行了统计分析, 显著性水平为 0.05, 其中, "–"表示劣于, "+"表示优于, "≈"表示相当于。

表 1 中, SparkDECC 与其他 3 种 DE 算法进行对比, 结果表明, SparkDECC 在 f_1、f_5、f_6、f_{10}~f_{13} 共 7 个函数能快速收敛到最优结果, 实验数据优于其他 3 种算法; SparkDECC 算法在 f_3、f_8 和 f_9 等 3 个函数的收敛出现了停滞, 实验结果弱于其他算法; 在不可分解函数 f_4 上各算法的运行效果相当; f_2 要劣于 jDE, 优于 OXDE 和 CoDE; 带噪声函数 f_7 的实验结果劣于 CoDE, 优于 jDE, 与 OXDE 相当。

表 1 SparkDECC、OXDE、CoDE、jDE、PSO 算法的平均值、标准差和 Wilcoxon 秩的检验结果

Table 1 Comparison of SparkDE, OXDE, CoDE, jDE and PSO algorithms for solving the results

F	OXDE (mean±std)	CoDE (mean±std)	jDE (mean±std)	PSO (mean±std)	SparkDECC (mean±std)
f_1	$4.76\times10^{+2}\pm1.67\times10^{+2}$ –	$1.50\times10^{-5}\pm1.74\times10^{-5}$ –	$2.46\times10^{-6}\pm1.23\times10^{-5}$ –	$2.30\times10^{+6}\pm2.79\times10^{+4}$ –	$5.85\times10^{-13}\pm1.62\times10^{-13}$
f_2	$5.27\times10^{+1}\pm6.89\times10^{+0}$ –	$8.89\times10^{-1}\pm9.74\times10^{-1}$ –	$1.74\times10^{-10}\pm8.62\times10^{-10}$ +	$7.16\times10^{+4}\pm3.30\times10^{+4}$ –	$6.60\times10^{-7}\pm1.00\times10^{-7}$
f_3	$1.54\times10^{+6}\pm2.33\times10^{+5}$ +	$4.50\times10^{+4}\pm6.12\times10^{+3}$ +	$7.54\times10^{+4}\pm1.48\times10^{+4}$ +	$8.68\times10^{+8}\pm6.51\times10^{+7}$ –	$5.31\times10^{+7}\pm7.20\times10^{+6}$
f_4	$2.59\times10^{+1}\pm1.84\times10^{+0}$ ≈	$2.67\times10^{+1}\pm1.95\times10^{+0}$ ≈	$5.04\times10^{+1}\pm4.58\times10^{+0}$ ≈	$4.01\times10^{+2}\pm7.98\times10^{+0}$ –	$9.76\times10^{+1}\pm2.22\times10^{-1}$
f_5	$2.28\times10^{+4}\pm7.08\times10^{+3}$ –	$2.39\times10^{+3}\pm2.12\times10^{+2}$ ≈	$2.18\times10^{+3}\pm2.21\times10^{+2}$ ≈	$9.23\times10^{+12}\pm1.29\times10^{+12}$ –	$1.62\times10^{+3}\pm1.62\times10^{+2}$
f_6	$7.86\times10^{+3}\pm8.86\times10^{+2}$ –	$5.26\times10^{+1}\pm6.26\times10^{+1}$ –	$1.06\times10^{+4}\pm2.98\times10^{+3}$ –	$2.30\times10^{+6}\pm1.29\times10^{+5}$ –	$1.60\times10^{-1}\pm4.73\times10^{-1}$
f_7	$5.68\times10^{+0}\pm7.25\times10^{-1}$ ≈	$9.06\times10^{-1}\pm1.03\times10^{-1}$ +	$2.91\times10^{+1}\pm1.60\times10^{+1}$ –	$3.48\times10^{+13}\pm3.87\times10^{+12}$ –	$3.62\times10^{+0}\pm1.67\times10^{-1}$
f_8	$-4.17\times10^{+5}\pm7.27\times10^{+2}$ +	$-1.89\times10^{+5}\pm5.14\times10^{+3}$ +	$-4.19\times10^{+5}\pm3.18\times10^{-1}$ +	$-1.34\times10^{+5}\pm4.51\times10^{+3}$ +	$-6.11\times10^{+4}\pm1.18\times10^{+3}$
f_9	$4.94\times10^{+2}\pm5.32\times10^{+1}$ +	$4.59\times10^{+3}\pm2.10\times10^{+2}$ +	$2.98\times10^{+0}\pm4.03\times10^{+0}$ +	$2.33\times10^{+6}\pm1.35\times10^{+5}$ –	$1.10\times10^{+4}\pm3.93\times10^{+1}$
f_{10}	$6.49\times10^{+0}\pm2.79\times10^{-1}$ –	$2.25\times10^{+0}\pm1.82\times10^{-1}$ –	$5.12\times10^{+0}\pm7.06\times10^{-1}$ –	$2.15\times10^{+1}\pm3.77\times10^{-2}$ –	$4.55\times10^{-8}\pm8.16\times10^{-9}$
f_{11}	$5.02\times10^{+0}\pm1.19\times10^{+0}$ –	$7.70\times10^{-3}\pm2.29\times10^{-2}$ –	$8.91\times10^{-1}\pm7.39\times10^{-1}$ –	$5.75\times10^{+2}\pm4.15\times10^{+1}$ –	$3.54\times10^{-14}\pm1.03\times10^{-14}$
f_{12}	$3.01\times10^{+0}\pm7.11\times10^{-11}$ –	$5.75\times10^{-2}\pm4.85\times10^{-2}$ –	$1.32\times10^{+6}\pm2.20\times10^{+6}$ –	$6.99\times10^{+12}\pm7.75\times10^{+11}$ –	$7.46\times10^{-4}\pm2.58\times10^{-3}$
f_{13}	$1.94\times10^{+3}\pm3.40\times10^{+2}$ –	$1.15\times10^{+2}\pm7.53\times10^{+1}$ –	$1.14\times10^{+7}\pm8.17\times10^{+6}$ –	$7.87\times10^{+12}\pm8.78\times10^{+11}$ –	$8.79\times10^{-4}\pm3.04\times10^{-3}$
–/+/≈	8/3/2	7/4/2	7/4/2	12/1/0	

SparkDECC 与 PSO 算法的结果对比，实验结果表明 SparkDECC 算法除函数 f_8 外其他都优于 PSO，说明 SparkDECC 算法总体性能优于 PSO。

函数收敛曲线图主要用来表示算法求解最优值的一种趋势走向，函数曲线下降得越快，收敛性能越好。为了比较 4 种不同 DE 算法在 f_1、f_3、f_5、f_6、f_9、f_{10}、f_{11}、f_{13} 等函数的收敛性能 (受篇幅限制，仅选择这 8 个函数)，选取了各算法独立运行 20 次中的平均收敛数据，收敛取值以 10 为底的对数形式，曲线图如图 3 所示。从图 3 可以看出 SparkDECC 算法的收敛能力较强，在函数 f_1、f_{10}、f_{11} 上的收敛曲线呈直线下降，收敛性能强；函数 f_5、f_6、f_{13} 在进化的前期有很好的收敛速度，但在后期出现了一定程度的局部收敛；函数 f_3, f_9 在整个进化过程中处于局部收敛，收敛性能弱，收敛效果要劣于其他算法。

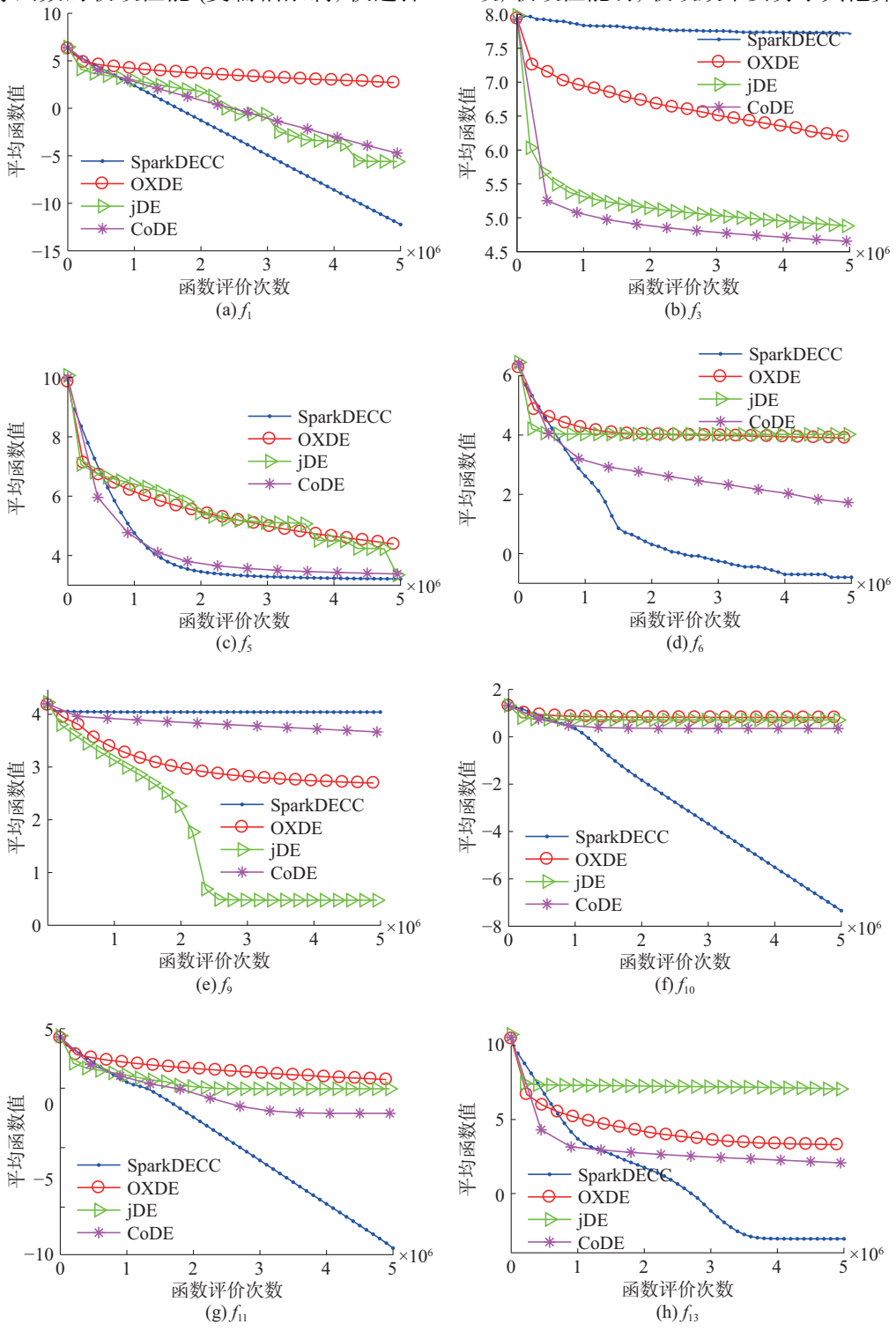

图 3　收敛图

Fig. 3　Convergence graph

综合以上实验数据分析表明，SparkDECC 算法能有效地求解大规模优化问题，提高了 DE 算法的收敛精度和收敛速率。

加速比[28]是衡量算法并行性的有效指标，其定义如式 (5) 所示。

$$S_k(M_n)=\frac{T_k(1)}{T_k(M_n)} \tag{5}$$

式中：$T_k(1)$代表在一个分区上独立运行 k 次的平均时间，$T_k(M_n)$代表在 n 个分区上独立运行 k 次的平均时间。实验时单峰函数选取了 f_1、f_3 和 f_5，多峰函数选取了 f_9、f_{11} 和 f_{13}。针对这 6 个高维优化函数，设定了 3 种不同的评价次数即 5E6、2.5E6 和 5E5，分别独立执行 10 次。图 4 的加速比表明，SparkDECC 算法在测试高维优化函数上，随着分区数的增加，算法执行时间逐渐减少，加速效果越好。当分区数增加到 5 的时候，加速比几乎接近 5 倍，与 2.2 节中对时间复杂度的分析一致。此外，函数的加速曲线与函数的评价次数关系不明显，表明了加速性能的稳定。

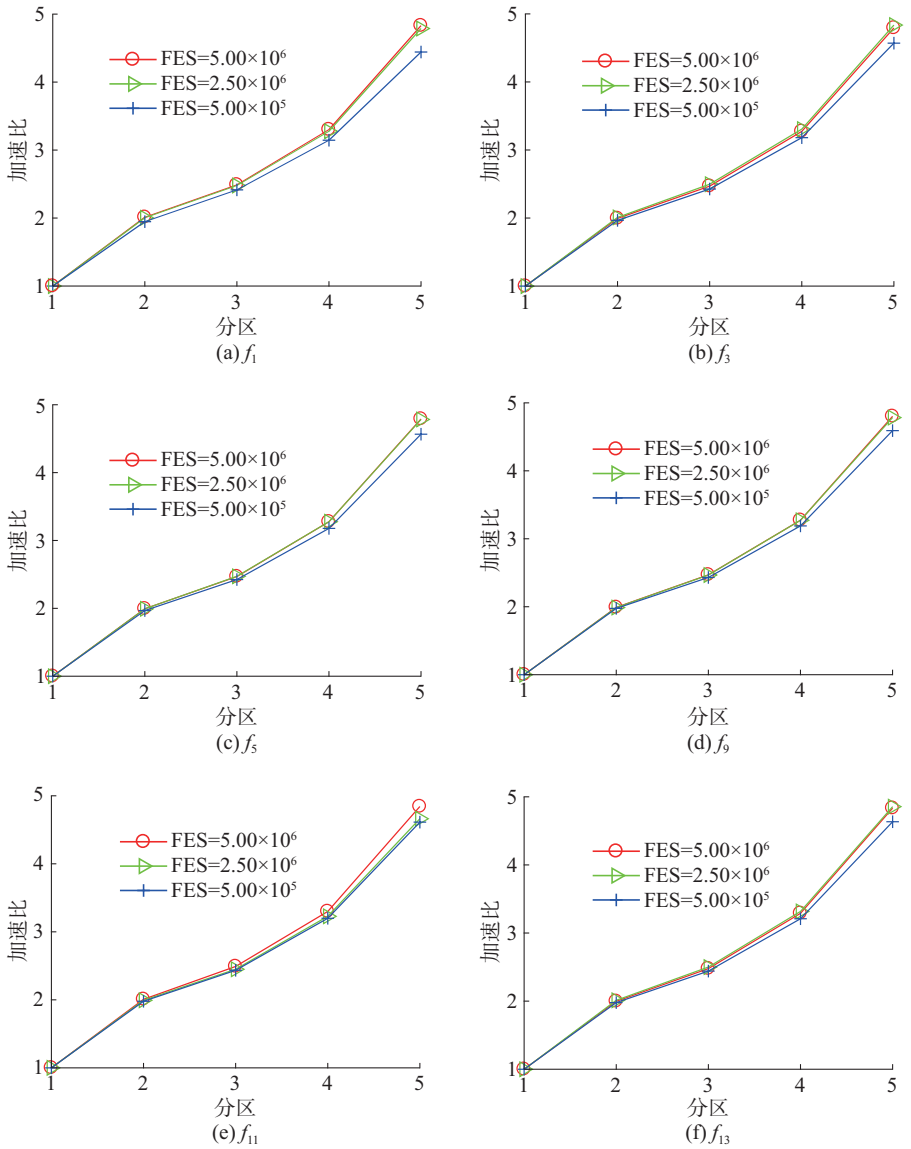

图 4　加速比

Fig. 4　Acceleration ratio

3.4　协同方式对算法性能的影响

SparkDECC 算法对子种群进行局部寻优后，需要合并所有子种群并进行全局搜索。SparkDECC 算法按照原模型进行组合，整个种群保持原来的结构。为了分析子种群合并机制对 SparkDECC 算法性能的影响，本文选取了两种不同的协同方式进行对比实验。这两种协同方式衍生出两种算法，即 SparkDECC-1 算法和 SparkDECC-2 算法，这两种算法在 SparkDECC 算法的基础上，子种群按不同的协同方式进行组合。其中，SparkDECC-1 算法在所有子种群组合之前，首先将所有子种群的个体按其适应度值升序排序，最后将所有子种群按线性方式组

合成完整的种群; SparkDECC-2 算法将所有子种群中的个体按随机方式组合成完整的种群。为了验证 3 种不同的 SparkDECC 算法的性能, 选择了相同的参数设置, 分别在 13 个函数上独立运行 20 次, 实验结果如表 2 所示。

表 2 的实验结果表明, SparkDECC 算法在 f_1、f_2、f_4、f_5、f_{10}、f_{11}、f_{12}、f_{13} 等 8 个函数的实验结果要优于其他两种协同方式。其中, SparkDECC-1 算法在 f_3、f_6、f_8、f_9 等 4 个函数优于 SparkDECC 算法, SparkDECC-2 算法中的所有函数都劣于 SparkDECC 算法。总之, SparkDECC 算法的其他组合方式不能有效地提高算法的收敛精度, 且增加了种群个体的排列组合运算, 算法的运行时间随之增加。因此, SparkDECC 算法选择最初的协同方式合并子种群, 维持原种群的结构, 能有效提高算法的精度和时间。

表 2　SparkDECC 3 种不同协同方式的实验结果

Table 2　Experimental results of SparkDECC on three different cooperative modes

F	SparkDECC-1 (mean±std)	SparkDECC-2 (mean±std)	SparkDECC (mean±std)
f_1	$1.39\times10^{-12}\pm3.11\times10^{-13}$ –	$3.06\times10^{+6}\pm4.79\times10^{+4}$ –	$5.85\times10^{-13}\pm1.62\times10^{-13}$
f_2	$1.11\times10^{-6}\pm1.86\times10^{-7}$ –	$4.73\times10^{+3}\pm3.65\times10^{+1}$ –	$6.60\times10^{-7}\pm1.00\times10^{-7}$
f_3	$4.96\times10^{+7}\pm9.00\times10^{+6}$+	$8.35\times10^{+7}\pm1.62\times10^{+7}$≈	$5.31\times10^{+7}\pm7.20\times10^{+6}$
f_4	$9.77\times10^{+1}\pm2.62\times10^{-1}$≈	$9.95\times10^{+1}\pm1.50\times10^{-1}$≈	$9.76\times10^{+1}\pm2.22\times10^{-1}$
f_5	$1.68\times10^{+3}\pm1.63\times10^{+2}$≈	$1.43\times10^{+10}\pm3.70\times10^{+8}$ –	$1.62\times10^{+3}\pm1.62\times10^{+2}$
f_6	$5.00\times10^{-2}\pm2.24\times10^{-1}$+	$3.05\times10^{+6}\pm5.15\times10^{+4}$ –	$1.60\times10^{-1}\pm4.73\times10^{-1}$
f_7	$2.27\times10^{+0}\pm1.42\times10^{-1}$≈	$2.35\times10^{+5}\pm4.63\times10^{+3}$ –	$3.62\times10^{+0}\pm1.67\times10^{-1}$
f_8	$-7.88\times10^{+4}\pm7.81\times10^{+2}$+	$-1.76\times10^{+4}\pm2.74\times10^{+3}$ –	$-6.11\times10^{+4}\pm1.18\times10^{+3}$
f_9	$1.00\times10^{+4}\pm7.44\times10^{+1}$+	$1.76\times10^{+4}\pm1.28\times10^{+2}$≈	$1.10\times10^{+4}\pm3.93\times10^{+1}$
f_{10}	$7.07\times10^{-8}\pm7.73\times10^{-9}$≈	$2.11\times10^{+1}\pm1.43\times10^{-2}$ –	$4.55\times10^{-8}\pm8.16\times10^{-9}$
f_{11}	$1.72\times10^{-3}\pm4.53\times10^{-3}$ –	$2.78\times10^{+4}\pm2.74\times10^{+2}$ –	$3.54\times10^{-14}\pm1.03\times10^{-14}$
f_{12}	$6.22\times10^{-4}\pm2.78\times10^{-3}$≈	$3.55\times10^{+10}\pm7.77\times10^{+8}$ –	$7.46\times10^{-4}\pm2.58\times10^{-3}$
f_{13}	$5.49\times10^{-4}\pm2.46\times10^{-3}$≈	$6.57\times10^{+10}\pm1.20\times10^{+9}$ –	$8.79\times10^{-4}\pm3.04\times10^{-3}$
–/+/≈	3/4/6	9/0/4	

3.5　子种群数对算法的影响

为了分析子种群数对 SparkDECC 性能的影响, 分别选取了不同数目的子种群进行对比实验, 如 1、2、5、10、20 等 5 种不同的子种群。根据 2.2 节中的子种群数的计算公式, 计算子种群的维度, 如问题维度为 1 000, 则上述 5 种不同子种群的维度分别为 1 000、500、200、1 00、50。算法的其他参数设置保持一致, 分别在 13 个测试函数上独立运行了 20 次, 表 3 和表 4 分别记录了 5 种不同子种群数下的平均最优值和平均运行时间。

表 3 和表 4 的结果分析发现, f_1、f_2、f_6、f_8、f_{10}、f_{12}、f_{13} 共 7 个函数的收敛精度随子种群数的增加而逐渐增强。f_3、f_4、f_9 这 3 个函数解的精度在种群不分解的情况下达到最优, 种群维度的分解并没有提高解的精度。f_7 的解在子种群数为 5 时达到最优, 再增加子种群数并不会提高解的精度。f_5、f_{11} 的解在子种群数为 10 时达到最优。因此, 对于可分解函数, SparkDECC 的种群多样性随子种群数的增加逐渐增强, 提高了解的收敛性。然后, 随子问题数目增多, 子问题之间的交互时间随之增加, 从而增加了函数的收敛时间。表 4 的结果显示, 函数的运行时间与子问题数成正比, 与 2.2 节中的时间复杂度相吻合。当子问题数增加到 20 时, 13 个函数总的平均时间是 1 个子问题的 11.5 倍。因此, SparkDECC 应从收敛时间和收敛精度两方面综合考虑选取合适的子种群数。

3.6　子种群进化代数对算法的影响

SparkDECC 算法在子种群进化若干代后, 需要合并成新的全局最优解。因此, 为了探索子种群局部寻优的代数 Gen 对 SparkDECC 算法的影响, 本文选取了 10、20、50、100、250 等 5 种情况分别进行实验。5 种情况的其他参数一致并设置相同的函数评价次数, 分别独立运行 25 次, 平均运行时间及结果分别如表 5 和表 6 所示。

表 3　SparkDECC 在不同子种群数的实验结果

Table 3　Experimental results of SparkDECC in different number of sub-populations

F/*M*	1 (mean±std)	2 (mean±std)	5 (mean±std)	10 (mean±std)	20 (mean±std)
f_1	$3.67\times10^{+4}\pm4.80\times10^{+3}$	$2.34\times10^{+3}\pm8.35\times10^{+2}$	$2.78\times10^{-3}\pm1.94\times10^{-3}$	$5.69\times10^{-13}\pm1.36\times10^{-13}$	$1.21\times10^{-28}\pm3.14\times10^{-29}$
f_2	$3.18\times10^{+2}\pm2.05\times10^{+1}$	$5.63\times10^{+1}\pm7.90\times10^{+0}$	$3.77\times10^{-2}\pm1.78\times10^{-2}$	$6.89\times10^{-7}\pm8.90\times10^{-8}$	$5.79\times10^{-14}\pm8.83\times10^{-15}$
f_3	$2.27\times10^{+7}\pm2.28\times10^{+6}$	$3.05\times10^{+7}\pm3.89\times10^{+6}$	$4.21\times10^{+7}\pm6.45\times10^{+6}$	$5.30\times10^{+7}\pm1.05\times10^{+7}$	$5.03\times10^{+7}\pm1.09\times10^{+7}$
f_4	$9.43\times10^{+1}\pm1.33\times10^{+0}$	$9.66\times10^{+1}\pm4.75\times10^{-1}$	$9.74\times10^{+1}\pm2.58\times10^{-1}$	$9.75\times10^{+1}\pm3.57\times10^{-1}$	$9.77\times10^{+1}\pm2.94\times10^{-1}$
f_5	$1.62\times10^{+7}\pm2.77\times10^{+6}$	$6.65\times10^{+5}\pm2.33\times10^{+5}$	$3.20\times10^{+3}\pm4.66\times10^{+2}$	$1.64\times10^{+3}\pm1.45\times10^{+2}$	$1.66\times10^{+3}\pm1.28\times10^{+2}$
f_6	$4.18\times10^{+4}\pm4.83\times10^{+3}$	$6.08\times10^{+3}\pm1.34\times10^{+3}$	$8.12\times10^{+1}\pm8.13\times10^{+1}$	$2.00\times10^{-1}\pm4.08\times10^{-1}$	$0.00\times10^{+0}\pm0.00\times10^{+0}$
f_7	$5.24\times10^{+2}\pm3.18\times10^{+2}$	$2.22\times10^{+1}\pm5.63\times10^{+0}$	$2.83\times10^{+0}\pm1.70\times10^{-1}$	$3.63\times10^{+0}\pm1.35\times10^{-1}$	$4.65\times10^{+0}\pm1.53\times10$-01
f_8	$-2.87\times10^{+4}\pm1.54\times10^{+3}$	$-3.43\times10^{+4}\pm1.17\times10^{+3}$	$-4.64\times10^{+4}\pm1.19\times10^{+3}$	$-6.15\times10^{+4}\pm1.12\times10^{+3}$	$-8.27\times10^{+4}\pm1.06\times10^{+3}$
f_9	$3.83\times10^{+3}\pm4.48\times10^{+2}$	$9.20\times10^{+3}\pm3.00\times10^{+2}$	$1.10\times10^{+4}\pm7.07\times10^{+1}$	$1.10\times10^{+4}\pm6.00\times10^{+1}$	$1.04\times10^{+4}\pm5.14\times10^{+1}$
f_{10}	$9.70\times10^{+0}\pm4.05\times10^{-1}$	$4.99\times10^{+0}\pm3.84\times10^{-1}$	$6.37\times10^{-1}\pm4.80\times10^{-1}$	$4.60\times10^{-8}\pm5.22\times10^{-9}$	$1.98\times10^{-13}\pm9.93\times10^{-15}$
f_{11}	$3.34\times10^{+2}\pm4.28\times10^{+1}$	$2.13\times10^{+1}\pm6.73\times10^{+0}$	$1.29\times10^{-2}\pm4.73\times10^{-2}$	$3.95\times10^{-4}\pm1.97\times10^{-3}$	$1.97\times10^{-03}\pm4.66\times10^{-3}$
f_{12}	$2.12\times10^{+6}\pm7.51\times10^{+5}$	$1.11\times10^{+5}\pm5.63\times10^{+4}$	$7.51\times10^{+1}\pm5.03\times10^{+1}$	$4.98\times10^{-4}\pm2.49\times10^{-3}$	$5.50\times10^{-29}\pm2.49\times10^{-29}$
f_{13}	$1.90\times10^{+7}\pm5.26\times10^{+6}$	$8.57\times10^{+5}\pm4.55\times10^{+5}$	$2.73\times10^{+2}\pm1.06\times10^{+2}$	$8.79\times10^{-4}\pm3.04\times10^{-3}$	$9.77\times10^{-28}\pm3.24\times10^{-28}$

表 4　SparkDECC 在不同子种群数的平均运行时间

Table 4　The average running time of SparkDECC in different number of sub-populations　ms

F/*M*	1	2	5	10	20
f_1	$1.70\times10^{+4}$	$1.75\times10^{+4}$	$2.14\times10^{+4}$	$2.67\times10^{+4}$	$3.81\times10^{+4}$
f_2	$1.70\times10^{+4}$	$1.88\times10^{+4}$	$2.46\times10^{+4}$	$3.33\times10^{+4}$	$5.16\times10^{+4}$
f_3	$2.06\times10^{+5}$	$3.94\times10^{+5}$	$9.39\times10^{+5}$	$1.86\times10^{+6}$	$3.76\times10^{+6}$
f_4	$1.64\times10^{+4}$	$1.76\times10^{+4}$	$2.09\times10^{+4}$	$2.52\times10^{+4}$	$3.45\times10^{+4}$
f_5	$1.66\times10^{+4}$	$1.79\times10^{+4}$	$2.23\times10^{+4}$	$2.87\times10^{+4}$	$4.22\times10^{+4}$
f_6	$1.91\times10^{+4}$	$2.28\times10^{+4}$	$2.71\times10^{+4}$	$3.37\times10^{+4}$	$4.98\times10^{+4}$
f_7	$1.68\times10^{+4}$	$1.78\times10^{+4}$	$2.20\times10^{+4}$	$2.84\times10^{+4}$	$4.17\times10^{+4}$
f_8	$4.05\times10^{+4}$	$6.51\times10^{+4}$	$1.34\times10^{+5}$	$2.49\times10^{+5}$	$4.81\times10^{+5}$
f_9	$3.55\times10^{+4}$	$5.66\times10^{+4}$	$1.13\times10^{+5}$	$2.08\times10^{+5}$	$3.94\times10^{+5}$
f_{10}	$3.60\times10^{+4}$	$5.58\times10^{+4}$	$1.02\times10^{+5}$	$1.78\times10^{+5}$	$3.36\times10^{+5}$
f_{11}	$3.85\times10^{+4}$	$5.91\times10^{+4}$	$1.18\times10^{+5}$	$2.20\times10^{+5}$	$4.25\times10^{+5}$
f_{12}	$4.99\times10^{+4}$	$7.91\times10^{+4}$	$1.44\times10^{+5}$	$2.37\times10^{+5}$	$4.41\times10^{+5}$
f_{13}	$6.12\times10^{+4}$	$9.91\times10^{+4}$	$1.63\times10^{+5}$	$2.46\times10^{+5}$	$4.41\times10^{+5}$

表 5　进化代数对 SparkDECC 优化时间的影响

Table 5　Optimization time of SparkDECC with generations　ms

F/Gen	10	20	50	100	250
f_1	$3.69\times10^{+4}$	$3.50\times10^{+4}$	$2.85\times10^{+4}$	$2.72\times10^{+4}$	$2.65\times10^{+4}$
f_2	$4.34\times10^{+4}$	$4.07\times10^{+4}$	$3.48\times10^{+4}$	$3.34\times10^{+4}$	$3.25\times10^{+4}$
f_3	$1.91\times10^{+6}$	$1.89\times10^{+6}$	$1.87\times10^{+6}$	$1.85\times10^{+6}$	$1.86\times10^{+6}$
f_4	$3.44\times10^{+4}$	$2.99\times10^{+4}$	$2.70\times10^{+4}$	$2.56\times10^{+4}$	$2.45\times10^{+4}$
f_5	$3.89\times10^{+4}$	$3.37\times10^{+4}$	$3.03\times10^{+4}$	$2.89\times10^{+4}$	$2.79\times10^{+4}$
f_6	$5.79\times10^{+4}$	$4.89\times10^{+4}$	$3.72\times10^{+4}$	$3.28\times10^{+4}$	$3.25\times10^{+4}$
f_7	$3.91\times10^{+4}$	$3.41\times10^{+4}$	$3.01\times10^{+4}$	$2.86\times10^{+4}$	$2.75\times10^{+4}$
f_8	$2.62\times10^{+5}$	$2.57\times10^{+5}$	$2.52\times10^{+5}$	$2.54\times10^{+5}$	$2.48\times10^{+5}$
f_9	$2.22\times10^{+5}$	$2.15\times10^{+5}$	$2.08\times10^{+5}$	$2.04\times10^{+5}$	$2.02\times10^{+5}$
f_{10}	$2.29\times10^{+5}$	$2.23\times10^{+5}$	$1.84\times10^{+5}$	$1.76\times10^{+5}$	$1.75\times10^{+5}$
f_{11}	$2.42\times10^{+5}$	$2.35\times10^{+5}$	$2.19\times10^{+5}$	$2.17\times10^{+5}$	$2.16\times10^{+5}$
f_{12}	$4.92\times10^{+5}$	$4.76\times10^{+5}$	$3.85\times10^{+5}$	$2.34\times10^{+5}$	$2.29\times10^{+5}$
f_{13}	$5.42\times10^{+5}$	$5.27\times10^{+5}$	$4.06\times10^{+5}$	$2.42\times10^{+5}$	$2.34\times10^{+5}$

表 6 的结果显示，f_1、f_2、f_7、f_8、f_9、f_{10} 等 6 个函数的求解精度随进化代数的增加逐渐提高。f_3 和 f_4 两个函数增加局部寻优的代数并不能提高解的精度，函数易陷入局部最优。f_6、f_{12} 和 f_{13} 等 3 个函数的结果在进化代数为 100 时达到最优。f_5 和 f_{11} 两个函数在进化代数为 50 时达到最优，进化代数的增加并不能提高解的质量。由 2.2 节中的时间复杂度可知，在评价次数相同的情况下，不同进化代数 Gen 的函数优化时间几乎相同。但从表 5 的结果显示，SparkDECC 算法的优化时间随参数 Gen 的增加而逐渐减少。其主要原因是，由 2.2 节中的算法可知，SparkDECC 算法在评价次数相同的情况下，合并轮数 Cycle 的值随进化代数 Gen 变大而逐渐变小，减少了广播变量执行的次数，因此，函数的优化时间会相应减少，但总的优化时间相差无异。

总之，SparkDECC 算法的收敛精度与子问题的

进化代数和合并轮数关系紧密, 增加子问题的进化代数提高了子问题的局部寻优能力。子问题的合并轮数改善了问题多样性。但是, 在评价次数相同的情况下, 子问题的合并轮数随子种群进化代数的增加而减少。因此, 为了在全局搜索和局部寻优之间达到平衡, 并能在有效时间内提高算法的收敛精度, 我们可以选择合适的进化代数, 合理分配计算资源。

表 6　进化代数对 SparkDECC 优化性能的影响

Table 6　Performance affection of SparkDECC with generations

F/Gen	10 (mean±std)	20 (mean±std)	50 (mean±std)	100 (mean±std)	250 (mean±std)
f_1	$1.51\times10^{+6}\pm1.60\times10^{+4}$	$1.14\times10^{+6}\pm1.80\times10^{+4}$	$7.21\times10^{-5}\pm2.67\times10^{-5}$	$5.79\times10^{-13}\pm1.52\times10^{-13}$	$1.73\times10^{-16}\pm4.28\times10^{-17}$
f_2	$3.15\times10^{+3}\pm1.99\times10^{+1}$	$2.68\times10^{+3}\pm2.26\times10^{+1}$	$2.21\times10^{-2}\pm3.11\times10^{-3}$	$6.62\times10^{-7}\pm8.59\times10^{-8}$	$4.63\times10^{-9}\pm7.16\times10^{-10}$
f_3	$3.62\times10^{+7}\pm3.33\times10^{+6}$	$4.15\times10^{+7}\pm7.19\times10^{+6}$	$4.88\times10^{+7}\pm7.61\times10^{+6}$	$5.06\times10^{+7}\pm1.13\times10^{+7}$	$5.60\times10^{+7}\pm1.68\times10^{+7}$
f_4	$9.72\times10^{+1}\pm2.55\times10^{-1}$	$9.72\times10^{+1}\pm3.41\times10^{-1}$	$9.75\times10^{+1}\pm3.13\times10^{-1}$	$9.75\times10^{+1}\pm3.14\times10^{-1}$	$9.77\times10^{+1}\pm3.44\times10^{-1}$
f_5	$5.01\times10^{+9}\pm9.54\times10^{+7}$	$3.39\times10^{+9}\pm9.52\times10^{+7}$	$1.08\times10^{+3}\pm6.90\times10^{+1}$	$1.70\times10^{+3}\pm2.96\times10^{+2}$	$2.47\times10^{+3}\pm2.15\times10^{+2}$
f_6	$1.51\times10^{+6}\pm1.76\times10^{+4}$	$1.13\times10^{+6}\pm2.24\times10^{+4}$	$2.39\times10^{+1}\pm3.29\times10^{+0}$	$2.40\times10^{-1}\pm8.31\times10^{-1}$	$1.52\times10^{+0}\pm2.49\times10^{+0}$
f_7	$1.21\times10^{+5}\pm2.57\times10^{+3}$	$8.28\times10^{+4}\pm2.78\times10^{+3}$	$8.79\times10^{+0}\pm3.26\times10^{-1}$	$3.64\times10^{+0}\pm1.48\times10^{-1}$	$1.67\times10^{+0}\pm8.74\times10^{-2}$
f_8	$-4.64\times10^{+4}\pm1.28\times10^{+3}$	$-5.09\times10^{+4}\pm9.98\times10^{+2}$	$-5.68\times10^{+4}\pm9.84\times10^{+2}$	$-6.08\times10^{+4}\pm1.11\times10^{+3}$	$-6.62\times10^{+4}\pm6.21\times10^{+2}$
f_9	$1.38\times10^{+4}\pm7.15\times10^{+1}$	$1.32\times10^{+4}\pm6.11\times10^{+1}$	$1.21\times10^{+4}\pm5.44\times10^{+1}$	$1.10\times10^{+4}\pm7.38\times10^{+1}$	$9.94\times10^{+3}\pm6.59\times10^{+1}$
f_{10}	$2.01\times10^{+1}\pm1.92\times10^{-2}$	$1.95\times10^{+1}\pm4.83\times10^{-2}$	$5.70\times10^{-4}\pm1.03\times10^{-4}$	$4.63\times10^{-8}\pm7.33\times10^{-9}$	$2.10\times10^{-9}\pm4.34\times10^{-10}$
f_{11}	$1.36\times10^{+4}\pm1.44\times10^{+2}$	$1.01\times10^{+4}\pm1.88\times10^{+2}$	$4.83\times10^{-6}\pm1.55\times10^{-6}$	$3.94\times10^{-4}\pm1.97\times10^{-3}$	$1.08\times10^{-3}\pm3.11\times10^{-3}$
f_{12}	$1.05\times10^{+10}\pm2.76\times10^{+8}$	$6.77\times10^{+9}\pm2.08\times10^{+8}$	$1.35\times10^{+8}\pm2.77\times10^{+7}$	$9.57\times10^{-11}\pm7.66\times10^{-11}$	$1.24\times10^{-3}\pm3.48\times10^{-3}$
f_{13}	$2.09\times10^{+10}\pm3.83\times10^{+8}$	$1.37\times10^{+10}\pm5.74\times10^{+8}$	$7.38\times10^{+6}\pm1.79\times10^{+6}$	$8.79\times10^{-4}\pm3.04\times10^{-3}$	$2.08\times10^{-1}\pm7.75\times10^{-1}$

3.7　SparkDECC 算法的可扩展性分析

为了验证 SparkDECC 算法的可扩展性, 本文将文献[25]中的 13 个测试函数扩展到 5 000 维进行实验。子问题的维度 S=100, F=0.5, CR=0.9, FES=5 000 D, 问题规模 NP=100, 算法独立运行 10 次。OXDE、CoDE 和 jDE 3 种算法在 5 000 维的实验参数与 SparkDECC 的参数一致, 实验结果如表 7。

表 7　SparkDECC、OXDE、CoDE、jDE 算法在 5 000 维的平均值、标准差和 Wilcoxon 秩和检验结果

Table 7　Comparison of SparkDECC, OXDE, CoDE, jDE algorithms for solving the results in 5 000 dimension

F	OXDE (mean±std)	CoDE (mean±std)	jDE (mean±std)	SparkDECC (mean±std)
f_1	$5.69\times10^{+4}\pm4.69\times10^{+3}$ –	$4.34\times10^{+3}\pm1.04\times10^{+3}$ –	$4.49\times10^{+4}\pm1.77\times10^{+4}$ –	$3.56\times10^{-12}\pm3.90\times10^{-13}$
f_2	Inf±NaN –	Inf±NaN –	Inf±NaN –	$3.67\times10^{-6}\pm2.04\times10^{-7}$
f_3	$1.05\times10^{+7}\pm1.33\times10^{+6}$+	$6.88\times10^{+5}\pm9.94\times10^{+4}$+	$1.43\times10^{+6}\pm3.02\times10^{+5}$+	$2.51\times10^{+9}\pm6.30\times10^{+8}$
f_4	$2.83\times10^{+1}\pm1.82\times10^{+0}\approx$	$3.25\times10^{+1}\pm1.99\times10^{+0}\approx$	$5.98\times10^{+1}\pm9.43\times10^{+0}\approx$	$9.93\times10^{+1}\pm9.33\times10^{-2}$
f_5	$5.80\times10^{+6}\pm9.98\times10^{+5}$ –	$1.86\times10^{+5}\pm6.81\times10^{+4}$ –	$2.52\times10^{+8}\pm1.06\times10^{+8}$ –	$9.87\times10^{+3}\pm3.29\times10^{+2}$
f_6	$1.92\times10^{+5}\pm1.52\times10^{+4}$ –	$2.64\times10^{+4}\pm2.86\times10^{+3}$ –	$4.67\times10^{+5}\pm1.40\times10^{+5}$ –	$1.00\times10^{+0}\pm6.67\times10^{-1}$
f_7	$7.10\times10^{+2}\pm1.76\times10^{+2}$ –	$7.11\times10^{+1}\pm1.16\times10^{+1}\approx$	$2.65\times10^{+4}\pm1.12\times10^{+4}$ –	$2.16\times10^{+1}\pm3.59\times10^{-1}$
f_8	$-2.03\times10^{+6}\pm4.91\times10^{+3}$ –	$-1.26\times10^{+6}\pm8.25\times10^{+4}$ –	$-2.09\times10^{+6}\pm4.17\times10^{+2}$ –	$-2.73\times10^{+5}\pm1.48\times10^{+3}$
f_9	$6.68\times10^{+3}\pm3.36\times10^{+2}$+	$2.79\times10^{+3}\pm1.97\times10^{+2}$+	$3.34\times10^{+3}\pm1.09\times10^{+3}$+	$5.67\times10^{+4}\pm1.31\times10^{+2}$
f_{10}	$9.13\times10^{+0}\pm2.84\times10^{-1}$ –	$7.28\times10^{+0}\pm2.39\times10^{-1}$ –	$1.29\times10^{+1}\pm2.06\times10^{-1}$ –	$6.82\times10^{-8}\pm3.46\times10^{-9}$
f_{11}	$4.62\times10^{+2}\pm7.75\times10^{+1}$ –	$3.78\times10^{+1}\pm6.80\times10^{+0}$ –	$5.45\times10^{+2}\pm1.07\times10^{+2}$ –	$5.81\times10^{-14}\pm5.96\times10^{-15}$
f_{12}	$4.55\times10^{+1}\pm1.63\times10^{+3}$ –	$1.90\times10^{+0}\pm3.67\times10^{-1}$ –	$1.16\times10^{+9}\pm4.62\times10^{+8}$ –	$6.22\times10^{-5}\pm1.97\times10^{-4}$
f_{13}	$4.56\times10^{+5}\pm8.23\times10^{+5}$ –	$4.24\times10^{+3}\pm7.11\times10^{+2}$ –	$2.72\times10^{+9}\pm9.69\times10^{+8}$ –	$4.40\times10^{-3}\pm1.39\times10^{-2}$
–/+/≈	10/2/1	9/2/2	10/2/1	

表 7 的结果显示，随着维度的增加，算法的求解性能下降，搜索时间成倍增长。SparkDECC 算法在 f_1、f_2、f_5、f_6、f_7、f_8、f_{10}、f_{11}、f_{12}、f_{13} 等 10 个函数的最优值都优于其他 3 种算法，但在 f_3 和 f_9 两个函数的最优值都劣于其他 3 个算法，4 种算法在函数 f_{10} 上的结果相似。OXDE、CoDE 和 jDE 3 种算法在 13 个测试函数的维度扩展到 5 000 时的运行时间分别是 25.15 天、16.68 天和 16.85 天。SparkDECC 算法的运行时间为 4.71 天，大大提升了算法的收敛时间。实验结果显示，SparkDECC 算法在 5 000 维的整体性能优于其他 3 种算法，具有很好的可扩展性。

4 结束语

本文利用新型的迭代云计算模型，提出新的基于 Spark 的合作协同云差分进化算法 (SparkDECC)。SparkDECC 算法按随机分组策略将高维问题分解成多个同维的低维子问题，将每个子问题与 RDD 模型中的分区一一对应，每个子问题并行执行 DE 算法，子问题独立进化若干代后，更新最优个体，提高种群的多样性。利用 Scala 语言在 Spark 云模型上实现了 SparkDECC，通过 13 个标准测试函数的对比实验，结果表明 SparkDECC 求解精度高，速度快，可扩展性好。此外，选择 6 个测试函数进行加速比实验，实验结果表明加速比与分区数量几乎是线性的，加速效果良好。后续工作将在 SparkDECC 的基础上探索新的分组策略，并不断改进其协同机制，提高算法的收敛效率和求解精度。

参考文献：

[1] STORN R, PRICE K. Differential evolution-a simple and efficient heuristic for global optimization over conti-nuous spaces[J]. Journal of global optimization, 1997, 11(4): 341–359.

[2] BREST J, GREINER S, BOSKOVIC B, et al. Self-adapting cont-rol parameters in differential evolution: a comparati-ve study on numerical benchmark problems[J]. IEEEtransactions on evolutionary computation, 2006, 10(6): 646–657.

[3] WANG Yong, ZHANG Qingfu. Differential evolution with composite trial vector generation strategies and co-ntrol parameters[J]. IEEE transactions on evolutionary computation, 2011, 2(15): 55–66.

[4] FRANS V D B, ENGELBRECHT A P. A cooperative approach to particle swarm optimization[J]. IEEE transactions on evolutionary computation, 2004, 8(3): 225–239.

[5] POTTER M A, De JONG K A. A cooperative coevolutionary approach to function optimization[J]. Lecture notes in computer science, 1994, 866: 249–257.

[6] YANG Z, TANG K, YAO X. Large scale evolutionary optimization using cooperative coevolution[J]. Information sciences, 2008, 178(15): 2985–2999.

[7] MEI Y, OMIDVAR M N, LI X, et al. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization[J]. ACM transactions on mathematical software, 2016, 42(2): 13.

[8] GUAN X, ZHANG X, WEI J, et al. A strategic conflict avoidance approach based on cooperative coevolution-ary with the dynamic grouping strategy[J]. International journal of systems science, 2016, 47(9): 1995–2008.

[9] HU X M, HE F L, CHEN W N, et al. Cooperation coevolution with fast interdependency identification for large scale optimization[J]. Information sciences, 2017, 381: 142–160.

[10] LI X, YAO X. Cooperatively coevolving particle swarms for large scale optimization[J]. IEEE transactions on evolutionary computation, 2012, 16(2): 210–224.

[11] OMIDVAR M N, YANG M, MEI Y, et al. DG2: a faster and more accurate differential grouping for large-scale black-box optimization[J]. IEEE transactions on evolutionary computation, 2017, 21(6): 929–942.

[12] MENG X, BRADLEY J, YUVAZ B, et al. Mllib: machine learning in apache spark[J]. Journal of machine research, 2015, 17(34): 1235–1241.

[13] 王诏远, 王宏杰, 邢焕来, 等. 基于 Spark 的蚁群优化算法[J]. 计算机应用, 2015, 35(10): 2777–2780.
WANG Zhangyuan, WANG Hongjie, XING Huanlai, et al. An implement of ant colony optimization based on spark[J]. Journal of computer applications, 2015, 35(10): 2777–2780.

[14] 朱继召, 贾岩涛, 徐君, 等. SparkCRF: 一种基于 Spark 的并行 CRFs 算法实现[J]. 计算机研究与发展, 2016, 53(8): 1819–1828.
ZHU Jizhao, JIA Yantao, XU Jun, et al. SparkCRF: a parallel implementation of crfs algorithm with spark[J]. Journal of computer research and development, 2016, 53(8): 1819–1828.

[15] DENG C, TAN X, DONG X, et al. A parallel version of differential evolution based on resilient distributed datasets model[C]//Bioinspired Computing-Theories and Applications. Springer, Berlin, Heidelberg, 2015: 84–93.

[16] TEIJEIRO D, PARDO X C, GONZÁLEZ P, et al. Implementing Parallel Differential Evolution on Spark[C]// European Conference on the Applications of Evolutionary Computation. Springer International Publishing, 2016: 75–90.

[17] 王虹旭, 吴斌, 刘旸. 基于 Spark 的并行图数据分析系统[J]. 计算机科学与探索, 2015, 9(9): 1066–1074.
WANG Hongxu, WU Bin, LIU Yang. Parallel graph data

analysis based on spark[J]. Journal of frontiers of computer science and technology, 2015, 9(9): 1066–1074.

[18] 刘志强, 顾荣, 袁春风, 等. 基于 SparkR 的分类算法并行化研究[J]. 计算机科学与探索, 2015, 9(11): 1281–1294.
LIU Zhiqiang, GU Rong, YUAN Chunfeng, et al. Parallelization of classification algorithms based on sparkR[J]. Journal of frontiers of computer science and technology, 2015, 9(11): 1281–1294.

[19] 袁斯昊, 邓长寿, 董小刚, 等. 求解大规模优化问题的云差分进化算法[J]. 计算机应用研究, 2016, 33(10): 2949–2953.
YUAN Sihao, DENG Changshou, DONG Xiaogang, et al. Cloud computing based differential evolution algorithm for large-scale optimization problems[J]. Application research of computers, 2016, 33(10): 2949–2953.

[20] 董小刚, 邓长寿, 袁斯昊, 等. MapReduce 模型下的分布式差分进化算法[J]. 小型微型计算机系统, 2016, 37(12): 2695–2701.
DONG Xiaogang, DENG Changshou, YUAN Sihao, et al. Distributed differential evolution algorithm based on mapreduce model[J]. Journal of Chinese computer systems, 2016, 37(12): 2695–2701.

[21] 谭旭杰, 邓长寿, 董小刚, 等. SparkDE: 一种基于 RDD 云计算模型的并行差分进化算法[J]. 计算机科学, 2016, 43(9): 116–119.
TAN Xujie, DENG Changshou, DONG Xiaogang, et al. SparkDE: a parallel version of differential evolutionbased on resilient distributed datasets model in cloud computing [J]. Computer science, 2016, 43(9): 116–119.

[22] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient di-stributed datasets: a fault-tolerant abstraction for inmemory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2–2.

[23] DAS S, SUGANTHAN P N. Differential evolution: a survey of the state-of-the-art[J]. IEEE transactions on evolutionary computation, 2011, 15(1): 4–31.

[24] 王贤伟, 戴青云, 姜文超, 等. 基于 Mapreduce 的外观设计专利图像检索方法[J]. 小型微型计算机系统, 2012, 33(3): 626–632.
WANG Xianwei, DAI Qingyun, JIANG Wenchao, et al. Retrieval of design patent images based on mapreduce model[J]. Journal of Chinese computer systems, 2012, 33(3): 626–632.

[25] YAO X, LIU Y, LIN G. Evolutionary programming ma-de faster[J]. IEEE transactions on evolutionary computation, 2000, 3(2): 82–102.

[26] WANG Y, CAI Z, ZHANG Q. Enhancing the search ability of differential evolution through orthogonal crossover[J]. Information sciences, 2012, 185(1): 153–177.

[27] 范德斌, 邓长寿, 袁斯昊, 等. 基于 MapReduce 模型的分布式粒子群算法[J]. 山东大学学报: 工学版, 2016, 46(6): 23–30.
FAN Debin, DENG Chanshou, YUAN Sihao, et al. Distributed particle swarm optimization algorithm based on mapreduce[J]. Journal Shandong university: engineering science, 2016, 46(6): 23–30.

[28] TAGAWA K, ISHIMIZU T. Concurrent differential evolution based on MapReduce[J]. International journal of computers, 2010, 4(4): 161–168.

作者简介:

谭旭杰, 男, 1978 年生, 讲师, 主要研究方向为计算智能、云计算。

邓长寿, 男, 1972 年生, 教授, 博士, 主要研究方向为计算智能、云计算、数据挖掘。

吴志健, 男, 1963 年生, 教授, 博士生导师, 主要研究方向为智能计算、并行计算和智能信息处理。主持或参与国家自然科学基金、“863”计划等各类科研项目 20 余项, 发表学术论文 120 余篇。