控制与决策
Control and Decision

基于协同过滤的连续黑箱优化问题元启发算法选择

张永韡， 汪镭

引用本文:
张永, 汪镭. 基于协同过滤的连续黑箱优化问题元启发算法选择[J]. 控制与决策, 2020, 35(6): 1297−1306.

在线阅读 View online: https://doi.org/10.13195/j.kzyjc.2018.0801

文章编号: 1001-0920(2020)06-1297-10 DOI: 10.13195/j.kzyjc.2018.0801

基于协同过滤的连续黑箱优化问题元启发算法选择

张永韡[1†], 汪 镭[2]

(1. 江苏科技大学 电子与信息学院，江苏 镇江 212003；2. 同济大学 电子与信息工程学院，上海 200092)

摘 要: 算法选择(AS)问题旨在为给定问题在算法集合中选择最佳算法. 随着优化算法的不断提出,算法选择问题是优化领域亟待解决的问题. 提出基于聚类的元启发算法五星评价体系,将算法性能指标映射至整数评价以减小评价空间. 通过测试24种常见优化算法与4种最新CEC大赛优胜算法在219种、3 000多个标准测试问题上的性能,得到评价矩阵. 将评价矩阵作为训练数据,使用协同过滤(CF)算法建立算法评价的预测模型. 使用该模型预测算法集内的所有算法在新问题上的评价,结果显示所提出方法预测精度较高,超过90 %的预测最佳算法为最终可行算法. 敏感性分析显示,该方法在先验信息有限的情况下仍可以保持较高的预测精度.

关键词: 算法选择；连续优化；黑箱优化；协同过滤；元启发；推荐系统

中图分类号: TP399 **文献标志码:** A

Metaheuristic algorithm selection system for continuous black-box optimization problems based on collaborative filtering

ZHANG Yong-wei[1†], *WANG Lei*[2]

(1. College of Electronics and Information，Jiangsu University of Science and Technology，Zhenjiang 212003，China；
2. College of Electronics and Information Engineering，Tongji University，Shanghai 200092，China)

Abstract: Selecting the best algorithm out of an algorithm set for a given problem is referred to as the algorithm selection (AS) problems. The importance of AS problems increases with the emerge of many optimization algorithms. Therefore, a five-star ranking system based on clustering is proposed, which maps the algorithm performance criteria to integers and reduces the ranking space. An algorithm set is prepared, including 24 commonly used optimization algorithms and four algorithms that win the CEC competition in 2016 and 2017. By testing the performance of the selected algorithms on 219 benchmark problems, a ranking matrix is obtained. The ranking matrix is used as the data source of the collaborative filtering (CF) algorithm to obtain a prediction model of algorithm ranking. For a new problem instance, the model predicts the ranking of all the algorithms in the algorithm set. The results show that the prediction accuracy is high, and over 90% of predicted best algorithms are capable of solving the problem instance. Sensitivity analysis shows that the proposed method can still maintain high prediction accuracy with limited prior information.

Keywords: algorithm selection; continuous optimization; black-box optimization; collaborative filtering; metaheuristics; recommendation system

0 引 言

设计或寻找更加快速、高效和稳定的算法是最优化研究领域不懈的追求. 在元启发算法领域,从以最速下降、置信域算法为代表的确定性算法,到以模拟退火为代表的随机搜索算法,再到以遗传算法、粒子群算法为代表的群体智能算法,最后到当前协方差自适应算子及协同进化思想在算法中的广泛应用,每次算法机理的深刻变化都引发了连续优化算法性能的飞跃.

在此过程中,以粒子群、差分进化为代表的群体智能进化算法得到算法研究领域广泛的关注,使得各类相似的新算法或改进算法层出不穷. 每年有10余种全新算法及无数改进算法被提出. Claus Aranha等在GitHub上维护一个进化算法的列表https://github.com/fcampelo/EC-Bestiary#evolutionary-computation-bestiary,目前已收集了超过200种进化算法. 另一方

收稿日期: 2018-06-12；**修回日期:** 2019-02-15.

基金项目: 国家自然科学基金项目(71371142, 71771176)；江苏政府海外留学奖学金项目(JS-2015-200)；镇江市软科学基金项目(2225031701).

责任编委: 巩敦卫.

†通讯作者. E-mail: ywzhang@just.edu.cn.

面,随着NFL(no free lunch)定理的提出[1],算法研究者越来越清晰地认识到设计通用高性能算法是不可行的,根据问题量身定制求解算法才是真正的出路. 然而,定制化设计算法同样需要大量算法领域及问题领域的相关知识,很多时候算法使用者并不完全具备这样的知识. 在这种背景下,算法的种类繁多反而成了算法使用者的最大障碍. 因此,如何向算法的使用者准确地推荐算法已成为算法研究领域迫切需要解决的实际问题[2].

算法选择(algorithm selection, AS)旨在为使用者在给定算法集合中推荐最佳算法. 从用户角度而言,AS不要求使用者具备算法内部机理的知识. ASP的形式化表达是由Rice提出的[3]:对于问题实例空间$\mathcal{I}$和算法空间$\mathcal{A}$,存在性能的映射$\mathcal{A}\times\mathcal{I}\to\boldsymbol{R}$. AS通过预测算法$A\in\mathcal{A}$对新问题$I\in\mathcal{I}$的性能$P(A,I)\in\boldsymbol{R}$,将算法集合中具有最佳预测性能的算法作为推荐的算法.

当前,尝试解决ASP的主流方法均基于探索地形分析(exploratory landscape analysis, ELA)[4]方法. ELA涵盖一系列用以获取最优化问题特性的技术. 基于ELA的算法选择方案通过建立问题特征与算法性能之间的映射,使用递归模型预测算法在给定问题上的性能[5]. 鉴于问题特征与问题类型的强相关性,目前基于ELA方法的AS只应用在特定问题领域,如旅行商问题[6]、数据挖掘[7]、聚类[8]及分类[9]等. 在应用ELA于连续黑箱优化问题(BCOP)的研究中,文献[10-12]使用机器学习算法对问题特征到算法性能的映射进行建模,但上述研究并没有考虑主流的群体智能进化算法,算法的涵盖面不足.

通过研究ASP的定义可以发现,对于BCOP的算法选择问题,如果将优化算法作为待推荐项目,待优化问题作为用户,则可将给定BCOP推荐算法的过程看作是向用户推荐项目的过程,因此本研究将算法选择问题看作推荐系统设计问题. 推荐系统算法按照使用的数据类型可以粗略地划分为3大类[13]:1)基于项目内容(例如电影的类别,内容关键字等)或用户的属性(如年龄,性别等)的推荐系统;2)基于项目评价的推荐系统;3)同时使用上述两种方法的综合系统. 对于BCOP,尽管问题(用户)的属性可通过ELA方法取得,但是目前并没有针对算法特征的量化分析方法,从而使基于内容的算法推荐受到了限制. 从另一个角度来看,如果将算法在每个问题上的评价看作算法的特征,则基于项目评价的推荐系统也适合解决ASP.

在基于评价的推荐系统算法中,协同过滤(collaborative filtering, CF)[13]是当前主流的高效推荐算法. CF基于其他用户的偏好向当前用户推荐项目. 在算法选择问题中,CF与其他方法的区别在于CF提取算法和问题的隐特征(latent features)[14],而基于ELA的方法[15]则依赖于问题的显式特征.

当前,基于CF的方法主要应用于面向网络服务推荐的场景. 将CF方法应用于ASP的相关研究并不多. 文献[15-16]首次将CF的概念应用于ASP中,但其涉及的求解算法为相关领域的专有算法. 文献[17]基于CF方法建立了面向命题可满足性,约束满足和机器学习3个领域的算法推荐系统,但该系统没有包含BCOP以及相应的求解算法. 文献[18]讨论了CF评价矩阵的采样问题,使得只获取部分评价的情况下仍可作出可靠的算法选择. 上述基于CF解决ASP的文献[15-18]中,均没有涉及BCOP及元启发算法.

综上,目前在算法选择领域,尚缺乏针对元启发算法选择的研究. 此外,使用CF解决BCOP的算法选择也处于研究的空白. 因此,针对元启发研究领域各类算法不断推陈出新的局面,本研究旨在填补以上研究领域的空白,研究解决基于CF的BCOP元启发算法选择问题. 本研究在大量算法-实例测试结果基础上,通过建立简化算法评级系统压缩评价空间,得出算法-实例评价矩阵. 在此基础上,使用CF建立评价矩阵的降维表达,并对新问题的算法评价作出预测.

1 基于算法评级的协同过滤

协同过滤算法基于表1所示的评价矩阵中的可用项对尚未评价的项目(标为“–”)进行预测. 当评价矩阵补充完整后,对每个用户,排名前N的预测项目可以作为向该用户推荐的项目.

表1 典型协同过滤评价矩阵

	用户1	用户2	用户3	用户4
项目1	5	4	3	2
项目2	–	2	4	–
项目3	3	2	3	3
项目4	1	–	5	1

CF方法通过在特征空间建立用户和项目的k维特征向量生成预测模型,学习特征向量的过程可以考虑为如下优化问题:

$$\min_{(\Theta,X)}\frac{1}{2}\sum_{(i,j):M_{i,j}=1}(\theta_j x_i^{\mathrm{T}}-R_{i,j}^{\mathrm{act}})^2+\frac{\lambda}{2}(\mathrm{tr}(\Theta\Theta^{\mathrm{T}})+\mathrm{tr}(XX^{\mathrm{T}})).\tag{1}$$

其中:$X=[x_1,\dots,x_m]^{\mathrm{T}}$为关于项目的$m\times k$特征矩阵;$\Theta=[\theta_1,\dots,\theta_n]^{\mathrm{T}}$为用户的$n\times k$特征矩阵;$R_{i,j}^{\mathrm{act}}$

为用户j对项目i的实际评价,R^{act}为$m \times n$矩阵,m为项目数量,n为用户数量. M为$m \times n$矩阵,用以标记是否存在相应的评价,1为是,0为否. 而最后的两部分用以防止X与Θ过拟合,λ为正则化参数. 式(1)是凸优化问题,可由任何现代凸优化算法快速求解. 学习完成后,用户j对项目i的预测评价可由下式得出:

$$R_{i,j}^{\mathrm{pdt}} = \theta_j x_i^{\mathrm{T}} + \mu_i, \tag{2}$$

其中μ_i是所有用户对项目i的评价均值. 在这里,项目代表算法,而用户代表问题实例. 因此,对给定问题推荐算法的过程将被看作将最佳项目推荐给指定用户的过程.

1.1　算法评价

对于给定问题,算法运行效果可由多种性能指标表达,为了在多个算法之间进行合理的比较,并将比较的结果应用于CF中,需要制定算法的评价系统.

1.1.1　绝对性能指标

绝对性能指标指不依赖于其他算法的性能指标. 常见的绝对性能指标如到达规定误差ϵ时所用的函数计算次数(function evaluation, FE),或者如计算资源耗尽(通常以允许的最大函数计算次数$\mathrm{FE}^{\max}$表达)时残余的误差.

本研究以$\mathrm{FE}_{i,j}^{\mathrm{mean}}$代表当误差小于$\epsilon$时,算法$i$对实例$j$所用的函数计算次数的均值,以$\mathrm{Error}_{i,j}^{\mathrm{mean}}$代表当函数计算次数达到$\mathrm{FE}^{\max}$时,算法$i$对实例$j$残余误差的均值.

据此,当$\mathrm{FE}_{i,j}^{\mathrm{mean}} < \mathrm{FE}^{\max}$时认为算法$i$对实例$j$是可行的,因为该算法在计算资源耗尽前找到了与理论极值误差小于ϵ的解. 当$\mathrm{Error}_{i,j}^{\mathrm{mean}} > \epsilon$时认为该算法不可行,因此误差高于$\epsilon$意味着计算资源已耗尽,算法在未找到满足精度的解之前便被迫中止了.

1.1.2　算法评级

对协同过滤推荐系统而言,基于相对评级的系统比基于绝对评价的系统具有更高的准确度[19]. 考虑到算法选择的最终目的为推荐最佳算法,而该算法的具体性能并不是使用者最关注的,相对的评级系统更为适用. 同时,由于相对评级系统的数值构成简单,可以降低学习系统的学习代价

更进一步的,在给定的算法集合中,对特定问题,往往只有部分算法有效. 对于用户而言,无效算法的评价并无意义,对无效算法进行精确评级会浪费取值空间,降低推荐系统的性能. 为解决该问题,本研究使用五星评级系统对算法进行评级,压缩评级的参数空间以降低推荐系统学习代价. 同时,为了评级结果与简单排序具有数值一致性(即数值小的代表性能好),五星评价的对应数值按照下式换算成排序数:

$$\text{排序数} = 5 - \text{算法星级} + 1. \tag{3}$$

具体评级流程如下.

step 1: 根据算法是否解决了给定问题,将算法分为可行/不可行两组.

step 1.1: 如果仅有一种可行算法,则将可行的最佳算法评为五星级算法(排序数1);

step 1.2: 如果没有可行算法,转step 3.

step 2: 将剩余的可行算法按照$\mathrm{FE}_{i,j}^{\mathrm{mean}}$聚类为两组,具有较低平均$\mathrm{FE}_{i,j}^{\mathrm{mean}}$组的所有算法评为4星(排序数2),其余算法为3星(排序数3).

step 3: 将不可行算法按照$\mathrm{Error}_{i,j}^{\mathrm{mean}}$聚类为两组,具有较低平均$\mathrm{Error}_{i,j}^{\mathrm{mean}}$的评为2星(排序数4),其余算法为1星(排序数5).

根据以上评级准则,3星及以上算法(排序数$\leqslant$3的)为可行算法,需要使用者重点关注. 2星和1星算法(排序数4和5)为不推荐的算法. 本研究不设置0星的评级准则,因为会与未评级项产生冲突.

最后,考虑到算法推荐系统的输出应为算法的评级(整数),而由式(2)得到的预测值为浮点数,需要将预测值映射为评级. 对于简单排序方法,使用预测值的排序数作为实际的算法评级(四舍五入法可能导致多个相同的排序数). 对于五星评级法,将预测值四舍五入取整后,令大于5的评级为5,小于1的评级为1.

1.1.3　先验率

对于黑箱问题而言,问题内部的任何信息均不可用. 在没有进行适当的测试前,无从判断给定的问题是否在训练集中出现过. 因此,对于黑箱优化问题,任何给出的测试问题均认为是新问题. 当可用算法集$\mathcal{A}$给定时,对于任意黑箱优化问题I_{new},由于不存在该问题的先验信息,推荐系统无法给出有效的推荐. 这种情况被称为“冷启动”问题. 取先验率为

$$r_{\mathrm{priori}} = \frac{\mathrm{card}(\mathcal{A}_r)}{\mathrm{card}(\mathcal{A})}. \tag{4}$$

其中:$\mathcal{A}_r$为已测试算法集合,$\mathrm{card}(\cdot)$为集合元素数量. 由式(4)可知,冷启动情况的先验率为0. 在实际中,一般使用全体用户对项目的平均评价作为新用户对该项目的默认评价. 然而,为了得到更精确的评价预测,有必要先行测试部分算法在该问题上的性能,并将排序后的算法排名作为推荐系统的输入. 显然,先行测试的算法占算法集的比例越大,预测的准确性越高,所需的计算资源也越大. 若算法集合中的所有

算法都在给定问题上进行了测试,则最佳算法可以完全确定,此时先验率为1.

显然先验率为0和1的情况对算法选择并无实际意义. 算法选择应至少在算法集中所有算法测试完成前对最佳算法进行预测,这表明需要在(0,1)范围内,以尽可能低的先验率获得尽可能高的预测准确度.

1.1.4 低先验率时的算法排序

当先验率不同时,对不同数量的算法进行测试,算法的排序可能不同. 对于简单排序法,若仅有一个算法进行测试,则将该算法评级为1是显然不合理的. 这里对简单排序引入一些改动以适应低先验率时的情况:

将已测试的算法按性能指标分为可行(na个)和不可行(nb个)两组. 对于可行算法,按$\mathrm{FE}_{i,j}^{\mathrm{mean}}$从小到大,从1到$na$,指定算法评级. 对于不可行算法,按照$\mathrm{Error}_{i,j}^{\mathrm{mean}}$从小到大,从$n-nb+1$到$n$指定算法评级,其中$n=na+nb=\mathrm{card}(\mathcal{A}_r)$为已测试算法数量. 这样,简单排序可以避免低先验率时不可行算法排序过高的问题.

对于五星评级方法,仍可按照1.1.2节的方法进行算法评级.

1.2 算法选择流程

算法选择的流程如图1所示. 具体步骤如下.

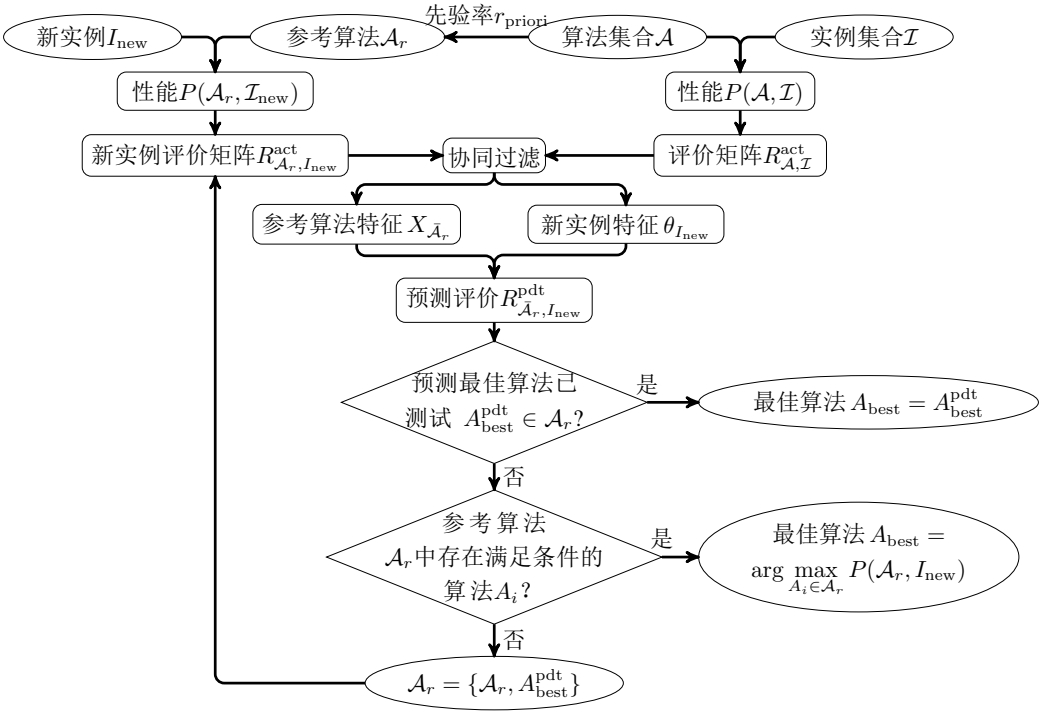

图1 基于协同过滤的算法选择流程

step 1: 测试算法集合中所有算法在问题集合上的性能,记录$P(\mathcal{A},\mathcal{I})$. 准备不同评级体系下的算法评级矩阵$R^{\mathrm{act}}_{\mathcal{A},\mathcal{I}}$,以算法评级矩阵作为算法选择的知识库. 此步骤可以先行完成,不以待推荐的问题而改变,且一次完成可应用于任意新问题,故不必计入推荐系统的计算开销内.

step 2: 对于任意新问题I_{new},按照先验率r_{priori}在算法库$\mathcal{A}$中随机选择一定数量的算法作为参考算法组成集合$\mathcal{A}_r$,并测试参考算法$\mathcal{A}_r$. 将I_{new}加入问题集合$\mathcal{I}\leftarrow\{\mathcal{I},I_{\mathrm{new}}\}$.

step 3: 按性能指标在指定的评级系统下进行评级得到参考算法$\mathcal{A}_r$在新问题I_{new}上的实际评级$R^{\mathrm{act}}_{\mathcal{A}_r,I_{\mathrm{new}}}$.

step 4: 将所得评级加入算法评级矩阵中,以此矩阵作为CF训练集,学习得到参考算法的特征矩阵$X_{\bar{\mathcal{A}}_r}$和问题实例特征矩阵$\theta_{I_{\mathrm{new}}}$. 使用式(2)得出未测试算法$\bar{\mathcal{A}}_r$在新问题$I_{\mathrm{new}}$上的评级预测数值$R^{\mathrm{pdt}}_{\bar{\mathcal{A}}_r,I_{\mathrm{new}}}$.

step 5: 由于$R^{\mathrm{pdt}}_{\bar{\mathcal{A}}_r,I_{\mathrm{new}}}$为小数,需按照对应评级系统的取整方法得到整数表达的预测评级. 对于五星评级法,由于已测试算法的评级是准确的,应使用已测试算法的评级代替该算法的预测评级. 举例来说,如果5个算法的现有评级为$[2,4,0,0,1]$,而根据式(2)得到的预测评级为$[2.8,3.7,4.2,0.8,2.3]$(评级为0代表尚未测试该算法),则预测评级应为$[2,4,4,1,1]$,而不是直接取整得到的$[3,4,4,1,2]$. 而对于简单排序法,由于低先验率时的评级并不准确,应对所有算法

使用预测评级.

step 6: 根据算法集中每个算法的最终预测评级,可以确定当前预测的最佳算法 $A_{\text{best}}^{\text{pdt}}$. 对于五星评价体系,选择预测评价最高的算法为最佳算法,对于简单排序体系,选择预测评价最小的算法为最佳算法.

step 7: 若预测最佳算法 $A_{\text{best}}^{\text{pdt}}$ 已测试($A_{\text{best}}^{\text{pdt}} \in \mathcal{A}_r$),则无必要继续算法选择,可将预测的最佳算法作为最终的选择;若预测最佳算法尚未测试,则转step 8.

step 8: 如果已测试算法 $\mathcal{A}_r$ 中存在可满足需求的算法(例如解的质量达到要求),则可终止算法选择,以测试算法中性能最佳者为推荐算法;如果已测试算法不满足需求,则转step 9.

step 9: 测试预测的最佳算法 $A_{\text{best}}^{\text{pdt}}$ 在问题 I_{new} 上的性能,并将测试结果与之前所测试的所有算法一同进行评级. 预测的最佳算法与之前已经测试的算法构成新的参考算法集合 $\mathcal{A}_r \leftarrow \{\mathcal{A}_r, A_{\text{best}}^{\text{pdt}}\}$,并转到step 3.

step 3 ~ step 9可以一直反复直到所测试的算法中性能最佳者可以满足需求或所有集合内的算法都被测试过.

2　协同过滤算法性能指标和参数优化

2.1　协同过滤性能指标

准确率和召回率是衡量推荐系统精度的常用指标,而由准确率和召回率共同决定的 F_1 分数代表了推荐系统在准确率和召回率之间的平衡. 准确率与召回率在不同的应用场景具有不同的解释. 一般的,准确率可以定义为所给出推荐中“好”项目的比例,而召回率可以定义为实际的“好”项目中被推荐的比例. 在这里,将“好”项目定义为可行算法,则可得到如下CF算法性能指标:

$$p_{N|M} = \frac{\text{card}(\{R_{i,j}^{\text{act}} \leqslant M | \{R_{i,j}^{\text{pdt}} \leqslant N | R_i^{\text{pdt}}\}\})}{\text{card}(\{R_{i,j}^{\text{pdt}} \leqslant N | R_i^{\text{pdt}}\})}, \quad (5)$$

$$r_{N|M} = \frac{\text{card}(\{R_{i,j}^{\text{pdt}} \leqslant M | \{R_{i,j}^{\text{act}} \leqslant N | R_i^{\text{act}}\}\})}{\text{card}(\{R_{i,j}^{\text{act}} \leqslant N | R_i^{\text{act}}\})}, \quad (6)$$

$$F_{1N|M} = \frac{2p_{N|M}r_{N|M}}{p_{N|M} + r_{N|M}}. \quad (7)$$

其中: $p_{N|M}$ 代表在预测排名前 N 的算法中,实际排名前 M 的算法比例; $r_{N|M}$ 代表在实际排名前 N 的算法中,被预测为排名前 M 算法的比例. 对于不同的问题实例,由于算法的性能各不相同, M 与 N 的值可能会不同.

更进一步,可将 M 取为可行算法的最大排序数,对于简单排序法, M 的数值视具体可行算法数量而变化;对于五星评价算法, M 的数值为3. 根据本定义,若可行算法的数量为 G,则可以进一步定义如表2所示的6种CF性能指标.

表 2　协同过滤算法选择性能指标

符号	含义
$p_{B\|G}$	预测最佳算法中实际可行算法的比例
$r_{B\|G}$	实际最佳算法中被预测为可行算法的比例
$F_{1B\|G}$	最佳-可行 F_1 分数
$p_{G\|G}$	预测可行算法中实际可行算法的比例
$r_{G\|G}$	实际可行算法中被预测为可行算法的比例
$F_{1G\|G}$	可行-可行 F_1 分数

2.2　协同过滤参数优化

在训练CF系统特征矩阵 Θ 和 X 时,式(1)代表了需要最小化的代价函数,该代价函数的特性很大程度取决于两个参数:特征数量 k 和正则化参数 λ. 其中特征数量 k 决定了CF系统可表达的多样性,低 k 值可能导致高偏置,高 k 值可能导致过拟合. λ 控制特征矩阵的取值大小,小 λ 可能导致过拟合,大 λ 可能导致高偏置. 训练得出的特征矩阵的性能通常由下式计算:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(R_i^{\text{act}} - R_i^{\text{pdt}})^2}. \quad (8)$$

其中: R_i^{act} 为算法 i 的实际评价,而 R_i^{pdt} 为预测评价, n 为可用的评价数量. 为了确定对于给定数据集的最佳参数 k 和 λ,通常采用 K-fold交叉验证[20]方法:将数据集随机划分为 K 组数据子集,每个子集的数据量相同,且数据互不重复. 每次取1组作为验证集,而剩余的 $K-1$ 组数据集作为训练集. 分别进行 K 次不同验证集的训练,取平均RMSE为本次交叉验证的性能指标.

在使用 K-fold交叉验证方法优化机器学习模型时, K 的取值是模型偏差和方差之间的权衡. 对于包含 m 个数据的数据集, K 的取值最小为2,最大为 n. 在不同的极端情况下,例如当 $K=2$ 时,训练集与测试集的数据完全不重复,此时训练得到模型的偏差较大而方差较小. 当 $K=n$ 时,每次训练都使用 $n-1$ 个数据,并使用剩下的一个数据作为验证. 这时每次训练的数据集几乎是一样的,导致模型的偏差较小,但是训练集之间的大相关性又导致测试误差具有较大的方差. 此外, K 的取值决定了需要训练模型的数量,因此,当数据集很大时,选取较高的 K 值将增加大量的运算量. 在实践中,通常取 $K=5$ 或者 $K=10$[21],以最佳地权衡模型偏差与误差.

本研究取 $K=10$,经过交叉验证后得到,对于简单排序系统,最佳参数为 $k=28, \lambda=0.0825$;对于五星系统,最佳参数为 $k=14, \lambda=20.48$. 实验证

实,使用五星评价系统可以在较少的CF特征下达到训练精度. 由于CF系统训练时间与特征数量成正比,较少的特征数量可节省大量训练时间.

3 优化算法性能测试

本研究使用安装了64位Windows 10企业版的酷睿i7 3.4 GHz 8核心处理器,8 G内存PC作为硬件平台. 使用Matlab R2017a作为软件开发环境. 为了便于扩展,所有算法(包括协同滤波,K-fold交叉验证,各类优化算法等)和测试函数均为自主实现.

3.1 测试函数

为了涵盖尽可能多的问题种类,本研究收集整理了文献中出现的219种标准测试函数[22-24],因篇幅有限,不再列出所涉及函数名称. 所选函数包含类型丰富,据作者所知,是目前最为全面的基础标准测试函数集. 其中包含连续/非连续问题,规则/不规则问题,可分/不可分问题,以及单模/多模问题. 测试问题的每一个可行维度都被看作问题的具体实例,不同的参数配置也看作不同的实例. 对于不可扩展的问题,仅测试其可行维度;对于可扩展问题,测试 $[1, 2, \ldots, 10, 12, 14, \ldots, 30, 40, \ldots, 100]$ 共27个维度. 考虑到高昂的计算代价,高于100维的实例不进行测试. 参与测试的219个问题共形成3 286个问题实例.

3.2 测试算法

共选择28种算法参与测试. 为了包含尽可能多的算法种类及特性,所选算法涵盖自1949年以来的代表性算法,其中包含基于梯度的和非梯度的,基于种群的和基于个体的,决定性的和随机性的. 算法列表如表3所示,部分算法年代较远,不再给出原始参考文献. 其中SGD、Newton、SR1、HJ、NM、BFGS、SW和TR为具有代表性的确定性局部搜索算法[25];SA、GA、PSO、CMA-ES和DE为具有代表性的经典元启发算法[26];mPSO、IWO、HS、SFLA、ICA、FA、ABC、BBO、ACOR、CS和TLBO为具有代表性的新兴元启发算法;UMOEAsII、LSHADE、jSO和EBO为2016～2017年CEC进化计算大会的最新优胜算法.

需要指出的是,尽管上述算法集合涵盖范围较广,但无法包含所有现存的算法. 从ASP的框架来看,在给定算法集合中预测性能最佳的算法是ASP的核心,在给定集合外是否存在性能更佳的算法并不是ASP尝试解决的问题. 因此,尽管无法涵盖所有现存算法,但作为验证本研究中的算法选择框架所使用的算法集合,上述28种算法是满足要求的.

表3 测试算法

缩写	年份	算法名
SGD	–	最速梯度[25]
Newton	–	牛顿法[25]
SR1	1949	对称秩1法[25]
HJ	1961	Hooke-Jeeves模式搜索[25]
NM	1965	Nelder-Mead单纯形法[25]
BFGS	1970	Broyden-Fletcher-Goldfarb-Shanno法[25]
SW	1981	Solis-Wet法[25]
TR	1982	置信域[25]
SA	1983	模拟退火[25]
GA	1986	遗传算法[26]
PSO	1995	粒子群算法[26]
CMA-ES	1996	协方差矩阵自适应进化策略[27]
DE	1997	差分进化[26]
mPSO	1998	惯性粒子群[28]
IWO	2006	野草入侵[29]
HS	2006	和声搜索[26]
SFLA	2006	混合蛙跳[26]
ICA	2007	帝国主义竞争[30]
FA	2007	萤火虫[26]
ABC	2007	人工蜂群[26]
BBO	2008	生物地理学优化[26]
ACOR	2008	连续蚁群算法[31]
CS	2009	布谷鸟搜索[32]
TLBO	2011	教学-学习优化[26]
UMOEAsII	2016	联合多算子进化算法II[33]
LSHADE	2017	具有欧几里德邻域的正弦微分协方差矩阵自适应[34]
jSO	2017	单目标实参优化[35]
EBO	2017	具有撤退阶段的协方差矩阵自适应蝴蝶优化算子[26]

3.3 测试结果

将28个算法在3 286个实例上进行测试,共对92 008个算法-实例组合进行了实验. 每组实验独立运行30次,共得到超过270万条测试结果. 对每次测试,设置$\mathrm{FE}^{\max} = D \times 10^4$,$D$为测试问题的维度,$\epsilon = 10^{-8}$. 当算法误差小于$\epsilon$或者FE达到$\mathrm{FE}^{\max}$时终止算法并记录优化结果,认为小于$\epsilon$的误差为0. 对测试结果进行统计分析并使用两种评价系统对算法排序后,可得到如图2和图3所示的评价矩阵,其中的列为所有算法特定实例的性能评级,每一行是特定算法对所有实例的性能评级. 算法缩写右侧括号内的数字是平均评级,算法按照平均评级数排列. 可以看出,在不同的评级下,会得到不同的平均评级排名.

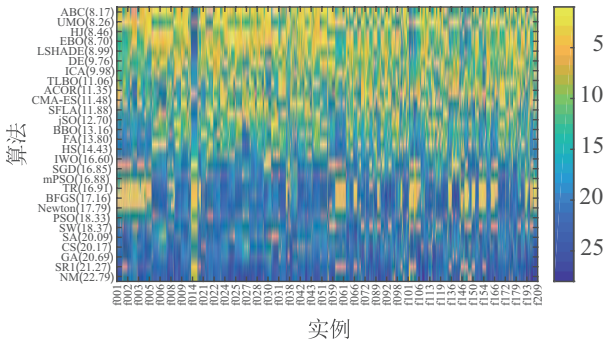

图2 简单排序评价矩阵图

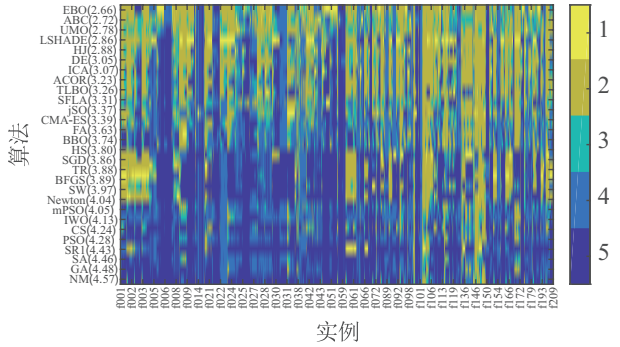

图 3　五星排序评价矩阵图

注1　为了表达形式的统一,对五星评级系统也使用了排序号表达,即序号越小的算法排名越高.

对给定问题所有测试算法的排序(评价矩阵的列)可以看作是问题的特征,类似的,也可以将某种算法在整个问题集上的评价作为算法的特征或“算法谱”. 推荐系统的任务是仅根据评价矩阵提供的信息,挖掘埋藏在算法谱之下的信息. 而推荐系统的性能,则很大程度地由评价矩阵的表达方法决定. 接下来将详细分析比较不同评价系统的算法选择性能差异.

4　数值实验结果分析

为了验证所提出算法选择方案的有效性,在先验率为0.2的情况下测试推荐系统的预测准确性. 如此,对任意新问题,将随机选择6个算法的评级作为先验信息,而CF系统将基于这些先验信息推断剩余22个算法的评级. 本实验使用共轭梯度法训练算法特征矩阵和实例特征矩阵,最大迭代为2 000代,实际中大多数训练(包括K-fold交叉验证训练和下文的预测实验)在1 000代之前完成. 因式(1)是凸优化问题,可以确定训练得到的特征为最佳特征.

对于优化了参数的预测实验,平均训练的迭代次数约为200次. 平均每次迭代的训练时间约为0.1 s,故大多数预测实验的运行时间为20 s左右. 测试时间由测试集的数量决定,平均测试速度为300实例/s. 总体而言,系统可在30 s内完成模型训练与测试.

4.1　基于协同过滤的算法选择有效性验证

将数据集按实例随机划分为两组,两组数据互不重复. 其中一组用以训练CF系统,所用数据占80%,剩余20%数据用以验证推荐系统的准确性. 由于共有3 286个实例,每次预测实验的训练集包含2 629个实例,而测试集包含657个实例. 取657个实例上预测准确性的均值作为本次预测的性能指标,分别进行20组预测实验,取20次实验的均值作为最终的准确性指标. 根据预测评级,统计相应性能指标如表4所示. 为了对比的需要,加入统计的还有五星评价系统下的其他两种推荐算法,其中随机推荐方法作为基线,随机指定所有算法的评级,统计推荐方法指按照训练集中算法的平均评级作为预测值. 对于相同的训练集,统计推荐法对任何新问题的预测都是一样的.

表 4　4种算法推荐系统的性能评价

评级方法	预测方法	$p_{B\|G}$	$r_{B\|G}$	$F_{1B\|G}$	$p_{G\|G}$	$r_{G\|G}$	$F_{1G\|G}$
五星	CF	0.93	0.83	0.88	0.73	0.82	0.78
简单	CF	0.72	0.60	0.66	0.72	0.72	0.72
五星	随机	0.35	0.60	0.44	0.37	0.63	0.47
五星	统计	0.62	0.44	0.51	0.46	0.61	0.53

在表4中,准确率和召回率均是在所有预测实例上的均值. 从统计结果可以看出,相对于简单排序法,五星评级法具有高很多的准确率和召回率. 具体的,$p_{B|G}$说明五星评级系统预测的最佳算法有93%的概率是可行的算法,$r_{B|G}$说明五星评级系统实际最佳算法有83%的概率被预测为可行算法. 所有可行算法的预测准确率低于最佳算法的预测准确率,这是因为有时可行算法的数目较多,要完整地预测出所有可行算法的名单显然比仅仅预测一个最佳算法要更加困难. 在数值方面,五星系统与简单排序系统的$p_{G|G}$值相近. $r_{G|G}$告诉我们对于实际可行算法,五星评级系统给出82%的准确预测,而简单排序系统则只有72%的召回率.

最后,在F_1分数反映出的系统综合性能上,五星评级系统除了远高于对照方法的分数,也显示了相对于简单排序的确定优势.

4.2　各问题族上的预测准确性

对于不同的问题实例或问题族(即具有相同表达式,不同参数的多个问题实例组成的集合),预测的准确率可能会极为不同. 此外,当问题族内的许多实例存在于训练集中时,预测该族的新问题准确率会很高,这相当于变相地为新问题提供了大量先验信息. 为了验证本系统的泛化能力,特选择比较有代表性的10个问题族. 在预测前,删除训练集内该问题族的所有实例,以消除先验信息的干扰. 每个问题族的准确率取所有实例的平均值,同样运行20次预测,并再取性能指标的均值,统计结果如表5所示.

由表5可看出,对于Sphere, Schwefel 1.2, Ackley, Schwefel 2.22和Step五个问题族内的所有实例,本系统预测的最佳算法均为可行算法,而其他问题族也保持了较高的准确率. 其中Schwefel 2.22和Step两个问题族在准确率和召回率上均保持了较高的水平.

对于Sphere等问题族较低的最佳算法召回率,原因是Sphere问题族在整个训练集中相对简单,几乎

表5 部分基础问题族的预测准确性

问题族	$p_{B\|G}$	$r_{B\|G}$	$F_{1B\|G}$	$p_{G\|G}$	$r_{G\|G}$	$F_{1G\|G}$
Sphere	1.00	0.66	0.79	0.94	0.85	0.89
Schwefel 1.2	1.00	0.66	0.79	0.92	0.85	0.88
Rosenbrock	0.97	0.67	0.79	0.57	0.73	0.64
Rastrigin	0.99	0.85	0.91	0.65	0.90	0.75
Griewank	0.93	0.65	0.77	0.81	0.81	0.81
Ackley	1.00	0.85	0.92	0.83	0.85	0.84
Shubert	0.85	0.90	0.94	0.59	0.64	0.63
Schwefel 2.22	1.00	0.96	0.98	0.83	0.93	0.87
Step	1.00	0.95	0.97	0.87	0.84	0.85
Schaffer	0.83	0.84	0.83	0.55	0.76	0.64
均值	0.96	0.80	0.87	0.76	0.82	0.78

所有算法均可行,使得这些问题实例的区分度不够高. 此外,这几类问题的最佳算法往往并不是平均评级较好的算法,使推荐系统出现了误判.

4.3 最坏情况分析

为了更进一步解释预测的细节,这里选用最佳算法和可行算法对预测 F_1 分数均值最低的Rosenbrock函数族进行分析. Rosenbrock问题为可扩展问题,实验中选取 $[2,\dots,10,12,14,\dots,30,40,\dots,100]$ 共26个维度的相应实例进行预测(1维Rosenbrock函数不存在). 在20次实验中任选一次,并考察预测精度最低的一个实例,其具体预测结果与实际结果的对比如表6所示.

表6 40维Rosenbrock问题的算法推荐数据

算法	预测评级	实际评级	可用评级	预测值	算法	预测评级	实际评级	可用评级	预测值
SGD	4	4	—	4.08	HS	5	4	5	3.91
Newton	4	4	—	3.98	BBO	4	4	—	3.84
SR1	4	4	4	4.21	FA	4	4	—	3.60
BFGS	4	2	—	3.90	ACOR	4	4	—	3.50
TR	4	1	—	4.00	SFLA	3	4	—	3.42
SW	4	4	—	4.20	TLBO	3	4	—	3.32
NM	5	4	—	4.64	DE	3	4	—	3.34
HJ	3	2	—	3.28	ABC	3	4	—	3.12
SA	4	4	4	4.50	CMA-ES	4	4	—	3.73
GA	5	4	5	4.49	CS	4	5	—	4.30
PSO	5	4	5	4.16	UMOEAsII	3	2	—	2.78
mPSO	4	4	—	4.02	LSHADE	3	3	—	2.87
IWO	4	4	—	4.11	jSO	3	4	—	3.14
ICA	3	4	—	3.29	EBO	1	2	1	2.75

表6显示,共有6种算法解决了40维Rosenbrock问题(即实际评级 $\leqslant$ 3的算法). Rosenbrock问题的解空间具有狭长的山谷,尽管该问题是单模问题,但是局部搜索能力不够强的算法无法在给定的计算资源下完成搜索. 如结果显示,一类可行算法是以BFGS为代表的经典局部搜索算法,另一类则是以UMOEAsII为代表的增强了局部搜索能力的进化算法. 在先验率 $r_{\text{priori}}=0.2$ 时,首先随机选择并测试了6种算法的性能. 对6种算法评级后,发现1种可行算法(EBO),根据此信息,推荐系统得出了所有算法的评级预测值. 根据1.1.2节的方法得到预测评级,并以测试得到的可用评级替换相应的预测评级.

对比预测评级与实际评级,可发现预测的最佳算法实际评级为2,出现预测偏差的原因是EBO可用评级为1,为保证总体准确性,系统使用可用评级1代替了原预测评级2. 本例中预测最佳算法的实际评级为2,因此有 $p_{B|G}=1$. 由于Rosenbrock问题相对较简单,预测的最佳算法有很大概率是可行的,这解释了高 $p_{B|G}$ 的原因. 另一方面,本例实际最佳算法为TR,而对TR的预测评级为4,为不可行. 由图3可知,TR在整个训练集上的平均评级为3.88,表明多数情况下TR为不可行算法. 因此,除非存在充分的先验信息,否则推荐系统对TR的预测评级往往过低. 由于本例的实际最佳算法为平均评级较低的算法(即反直觉的),导致了较低的 $r_{B|G}$. 对于这类存在反直觉效应的问题,往往具有低召回率. 在本例的6个实际可行算法中,除了已测试的1个,剩余5个中只有HJ、UMOEAsII和LSHADE这3个算法的评级类型是正确的. TR及BFGS的类型错误导致了较低的 $r_{G|G}$.

总体看来,在对22个未测试算法的评级预测中,40维Rosenbrock问题的最坏情况仍然正确分类了15个算法,占68%,表明了本算法推荐系统具有较高的可靠性.

4.4 关于先验率的敏感性分析

本节讨论不同先验率对于算法推荐系统的性能影响. 选择合适的先验率,使得对给定问题测试的算法数量分别为0~27. 当先验率为0时,意味着没有事先测试任何算法,推荐系统将以平均评级最高的算法作为推荐算法. 当先验率为1时,意味着所有算法已经测试完毕,此时无需再进行预测. 每种先验率数值下分别进行20次算法预测实验,所得结果如图4所示.

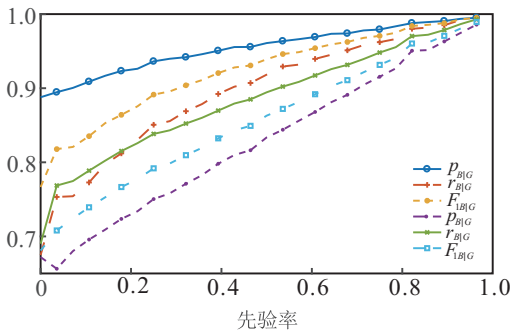

图4 先验率变化

观察图4可发现,推荐系统的各项精度总体随先验率的提升而提升,当测试算法达到27个时,各项数值达到1. 当先验率为0时,$p_{B|G}$仍保持接近90%的水平. 这是因为EBO算法解决了训练集中近90%的实例,当没有先验信息可用时,推荐系统选择平均评级最高的EBO算法为推荐算法,将在近90%的情况下是可行的. 然而,应该关注的是EBO没有解决的剩余10%问题,这才是算法推荐系统性能的关键. 对于这些实例,如果贸然预测最佳算法为EBO,则会带来错误. 当取得关于新实例的更多信息后,各项准确性逐步提高. 当先验率达到0.2时,$p_{B|G}$达到0.93,其他各项准确率指标也与表4中的数据一致.

另一方面,推荐系统在先验率连续的变化下各项准确率也以稳定的速率变化,说明投入的计算资源与获得的预测准确率总是成正比的. 而且,即使在较低的先验率水平下,系统仍然可以较为可靠地运行. 值得注意的是,在先验率由0到0.1变化的过程中,多项精度指标均有较大的跳变(包括$r_{B|G}$, $F_{1B|G}$和$r_{G|G}$),表明即使非常有限的信息也可以使本系统将给定的实例区分出来,继而给出较为准确的算法推荐,充分表明了本系统的可靠性.

5 结　论

算法选择问题的求解在算法理论与应用领域均具有较高的重要性. 本研究测试了28种元启发优化算法在219种测试函数,共计3 286个问题实例上的性能. 在测试数据的基础上,提出算法性能的五星评级体系以降低计算复杂度. 在算法-问题实例的五星评级矩阵上,应用协同过滤算法预测各个算法在新问题上的评级,以评级最佳的算法为推荐算法.

由实验结果可以得出以下结论: 1) 协同过滤算法在连续黑箱优化问题的算法推荐中具有很高的准确性. 总体而言,推荐的最佳算法有93%的概率是实际可行的算法,且实际最佳算法有83%的概率被预测为可行算法. 对比随机推荐和统计推荐方法,准确率分别提升了166%和50%. 2) 通过引入五星评级系统,降低了算法推荐系统的学习复杂度,使性能提升. 对比基于简单排序的协同过滤推荐算法,准确率提升了29%. 3) 关于先验率的敏感性分析显示本方法在关于问题的先验信息较少的情况下仍可作出较可靠的算法推荐. 具体的,在先验率达到0.2时,本研究所提出的方法即可达到93%的准确率.

然而,本方法仍存在一些缺点. 首先,协同过滤算法在每次向新问题推荐算法时都需要进行完整的学习,导致推荐系统的运行速度较慢. 其次,本方法需要至少一个算法完整运行一次才能进行较为有效的算法推荐. 最后,算法集合与测试问题集合的覆盖性可能仍有不足. 为了解决上述缺点,未来的研究可包括如下方向: 1) 研究基于增量式学习的协同过滤算法,提高对新问题推荐算法的实时性. 2) 通过结合ELA方法,在尚未完整运行一次优化算法的情况下作出首次推荐,可在较小的计算代价下快速确定实例的特征并提高首次推荐的准确率. 3) 在获得至少一次优化算法运行结果后,可考虑结合基于ELA方法的结果进行信息融合,进一步提高预测准确性. 4) 结合贝叶斯理论,建立新问题的算法偏好模型,提高迭代预测的精度. 5) 扩展问题集和算法集,提高知识库的覆盖面.

参考文献(References)

[1] Wolpert D H, Macready W G. No free lunch theorems for optimization[J]. IEEE Trans on Evolutionary Computation, 1997, 1(1): 67-82.

[2] 曾子林, 张宏军, 张睿, 等. 基于元学习思想的算法选择问题综述[J]. 控制与决策, 2014, 29(6): 961-968.
(Zeng Z L, Zhang H J, Zhang R, et al. Summary of algorithm selection problem based on meta-learning[J]. Control and Decision, 2014, 29(6): 961-968.)

[3] Rice J R. The algorithm selection problem[J]. Advances in Computers, 1976, 15(C): 65-118.

[4] Mersmann O, Bischl B, Trautmann H, et al. Exploratory landscape analysis[C]. The 13th Annual Conference on Genetic and Evolutionary Computation. Dublin: ACM, 2011: 829-836.

[5] Muñoz M A, Kirley M, Halgamuge S K. Exploratory landscape analysis of continuous space optimization problems using information content[J]. IEEE Transactions on Evolutionary Computation, 2015, 19(1): 74-87.

[6] Kanda J, Carvalho A de, Hruschka E, et al. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features[J]. Neurocomputing, 2016, 205: 393-406.

[7] Parmezan A R S, Lee H D, Wu F C. Metalearning for choosing feature selection algorithms in data mining: proposal of a new framework[J]. Expert Systems with Applications, 2017, 75: 1-24.

[8] Ferrari D G, De Castro L N. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods[J]. Information Sciences, 2015, 301: 181-194.

[9] Brazdil P B, Soares C. A comparison of ranking methods for classification algorithm selection[C]. European Conference on Machine Learning. Barcelona: Springer, 2000: 63-75.

[10] Kerschke P, Trautmann H. Automated algorithm selection

on continuous black-box problems by combining exploratory landscape analysis and machine learning[J]. Evolutionary Computation, 2019, 27(1): 99-127.

[11] Muñoz M A, Kirley M, Halgamuge S K. A meta-learning prediction model of algorithm performance for continuous optimization problems[C]. International Conference on Parallel Problem Solving from Nature. Taormina: Springer, 2012: 226-235.

[12] Bischl B, Mersmann O, Trautmann H, et al. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning[C]. The 14th International Conference on Genetic and Evolutionary Computation. Philadelphia: ACM, 2012: 313-320.

[13] Ekstrand M D, Riedl J T, Konstan J A, et al. Collaborative filtering recommender systems[J]. Foundations and Trends in Human-Computer Interaction, 2011, 4(2): 81-173.

[14] Misir M, Sebag M. Algorithm selection as a collaborative filtering problem[R]. https://hal.inria.fr/hal-00922840, 2013.

[15] Stern D, Herbrich R, Graepel T. Matchbox: large scale online bayesian recommendations[C]. The 18th International Conference on World Wide Web. Madrid: ACM, 2009: 111-120.

[16] Stern D, Samulowitz H, Pulina L, et al. Collaborative expert portfolio management[J]. Artificial Intelligence, 2010, 116(3): 179-184.

[17] Misir M, Sebag M. Alors: An algorithm recommender system[J]. Artificial Intelligence, 2017, 244: 291-314.

[18] Misir M. Data sampling through collaborative filtering for algorithm selection[C]. IEEE Congress on Evolutionary Computation. Donostia: IEEE, 2017: 2494-2501.

[19] Wang S, Liu T Y, Research M, et al. Ranking-oriented collaborative filtering: A listwise approach[J]. ACM Transactions on Information Systems, 2016, 35(2): 343–352.

[20] Kuhn M, Johnson K. Applied predictive modeling[M]. New York: Springer, 2013: 69-71.

[21] Molinaro A. Prediction error estimation: A comparison of resampling methods[J]. Bioinformatics, 2005, 21(15): 3301-3307.

[22] Adorio E, Diliman U. MVF multivariate test functions library in C for unconstrained global optimization[R]. Diliman: University of the Philippines, 2005: 1-56.

[23] Molga M, Smutnicki C. Test functions for optimization needs[EB/OL]. (2005-04-03)[2019-02-13]. https://www.robertmarks.org / Classes /ENGR5358 /Papers /functions.pdf.

[24] Jamil M, Yang X S. A literature survey of benchmark functions for global optimization problems[J]. International Journal of Mathematical Modelling and Numerical Optimisation, 2013, 4(2): 150-194.

[25] Nocedal J, Wright S J. Numerical optimization[M]. New York: Springer, 2006: 46-54.

[26] Simon D. Evolutionary optimization algorithms[M]. Hoboken: John Wiley & Sons, Inc, 2013: 35-62.

[27] Hansen N, Ostermeier A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation[C]. IEEE International Conference on Evolutionary Computation. Nagoya: IEEE, 1996: 312-317.

[28] Shi Y, Eberhart R. A modified particle swarm optimizer[C]. IEEE International Conference on Evolutionary Computation. Alaska: IEEE, 1998: 69-73.

[29] Mehrabian A R, Lucas C. A novel numerical optimization algorithm inspired from weed colonization[J]. Ecological Informatics, 2006, 1(4): 355-366.

[30] Atashpaz-Gargari E, Lucas C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition[C]. IEEE Congress on Evolutionary Computation. Singapore: IEEE, 2007: 4661-4667.

[31] Socha K, Dorigo M. Ant colony optimization for continuous domains[J]. European Journal of Operational Research, 2008, 185(3): 1155-1173.

[32] Yang X S, Deb S. Cuckoo search via Lévy flights[C]. World Congress on Nature & Biologically Inspired Computing. Coimbatore: IEEE, 2009: 210-214.

[33] Elsayed S, Hamza N, Sarker R. Testing united multi-operator evolutionary algorithms-II on single objective optimization problems[C]. IEEE Congress on Evolutionary Computation. Vancouver: IEEE, 2016: 2966-2973.

[34] Awad N H, Ali M Z, Suganthan P N. Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems[C]. IEEE Congress on Evolutionary Computation. Donostia: IEEE, 2017: 372-379.

[35] Brest J, Mauec M S, Bokovi B. Single objective real-parameter optimization: Algorithm JSO[C]. IEEE Congress on Evolutionary Computation. Donostia: IEEE, 2017: 1311-1318.

[36] Kumar A, Misra R K, Singh D. Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase[C]. IEEE Congress on Evolutionary Computation. Donostia: IEEE, 2017: 1835-1842.

作者简介

张永韡(1983–), 男, 讲师, 博士, 从事元启发算法理论与应用等研究, E-mail: ywzhang@just.edu.cn;

汪镭(1990–), 男, 教授, 博士生导师, 从事智能优化、人工智能、进化计算等研究, E-mail: wanglei_tj@126.com.

(责任编辑：孙艺红)