粒子群优化算法浅析及其在经济管理领域的应用

李子岳　　肖昌荣
经 71　　经 71
2017012781　　2017012527

日期：2020 年 6 月 15 日

摘　　要

粒子群算法（PSO）是一种优化问题得到最优解的元启发算法，受到自然界中鸟群活动的启发，有非常简洁的结构，而且在实际中取得了非常广泛的应用。本文介绍了粒子群算法的实现原理，并通过代码模拟的方式解释了 PSO 算法的操作流程，以运营管理中常见的情景为例，得到了优化的结果。文章介绍了 PSO 算法在经济管理领域的应用，并分析了该算法的现状以及未来可能的发展与研究方向。

关键词： 粒子群优化算法，运营管理

1　算法简介

粒子群优化（Particle Swarm Optimization, PSO），又称微粒群算法，是由 Kennedy et al. (1995) 于 1995 年开发的一种演化计算技术。它是一种元启发法，对于要优化的问题只需要做出很少或根本不需要假设、不使用要优化问题的梯度，这意味着 PSO 不需要像经典的优化方法（例如梯度下降和拟牛顿法）所要求的那样使优化问题可微分。所以算法的适用范围更广，可以搜索很大范围的候选解决方案。

粒子群优化算法来源于对鸟群行为社会模型的模拟。其中“群（swarm）”来源于微粒群，“粒子（particle）”则是一个折衷的选择，因为既需要将群体中的成员描述为没有质量、没有体积的，同时也需要描述它的速度和加速状态。算法主要源于两种思想方法，首先是它与人工生命的联系，例如鸟群、鱼群、蚁群等群体理论（Swarming Theory）。其次，它也与进化计算有关，特别是遗传算法和进化编程 (Kennedy et al., 1995)。

PSO 是一个极为简单且十分有效地适用于许多优化问题中的优化算法，它在多维空间函数寻优、动态目标寻优等方面有着收敛速度快、解质量高、鲁棒性好等优点，特别适合于工程应用 (黄磊, 2010)。但是 PSO 作为一种元启发法不能保证找到最佳解决方

案，且存在与算法收敛性相关的问题。这些缺陷并不妨碍粒子群优化算法成为最为经典的智能算法之一，许多学者同样受到自然界启发，开发了诸多类似的智能算法，且 PSO 从提出至今不断有学者对其进行完善，提高其在实际应用中的效果。

2 算法原理与实现

2.1 算法原理

粒子群优化算法受到自然界中鸟群的启发，它考虑的情景是，在一个区域内有大大小小不同的食物源，鸟群的任务是找到最大的食物源。鸟群在整个搜寻的过程中，通过相互传递各自位置的信息，让其他鸟知道食物源的位置，最终整个鸟群都能聚集在最大食物源周围，即找到了全局最优解，问题收敛。

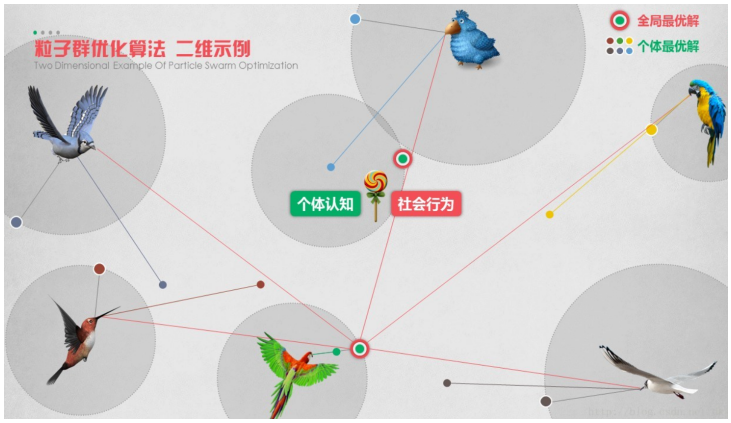

图 1: PSO 算法示例

为了找到食物源，一群鸟在森林里漫无目的的飞着，假设他们能感知到自己距离食物的距离。A 是这群鸟中的一只，它从其他同伴的口中得知，当前鸟 B 距离食物最近，所以它考虑着往鸟 B 的方向飞。但是，鸟 A 突然想到，虽然现在鸟 B 距离食物最近，但自己曾经飞过的位置 Z 是已经飞过的路途中距离食物最近的点。现实情况中，食物源可能与 Z 点距离更近，也可能与鸟 B 与距离更近。因此，一般性原则是，鸟 A 选择朝着鸟 B 和朝着位置 Z 的方向的矢量和的方向飞行，这就是下文算法中提到的速度更新公式。

$$v_{id} = v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})$$

我们把上式中的 c_1 称为社会学习因子，c_2 称为个体学习因子。使用上述速度更新公式，鸟 A 就知道自己应该朝哪个方向飞行，所以就得到下面的位置更新公式。

$$x_{id} = x_{id} + v_{id}$$

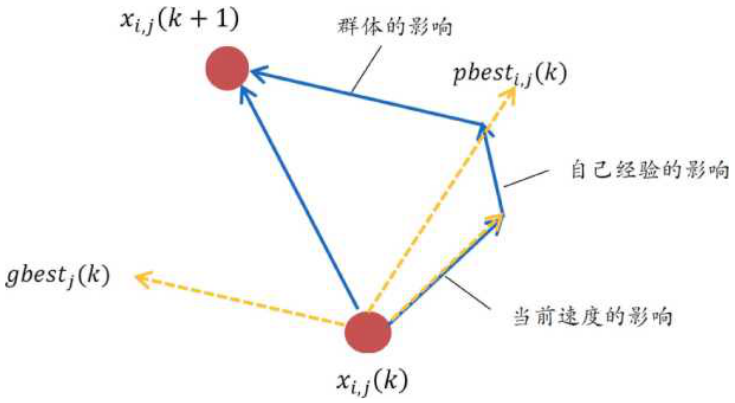

图 2: PSO 更新过程

将上述描述抽象为算法模型，Shi et al. (2001) 在文章中给出的算法如下：

Algorithm 1: Particle Swarm Optimization

Initialize a population of particles with random positions x_{id} and velocities v_{id} on a d dimensions problem space

while *the criterion is not met* **do**

  foreach *particle* **do**

    evaluate the desired optimization fitness function

    compare particle's fitness evaluation with particle's $pbest$

    if *current value is better than* $pbest$ **then**

      $pbest$ value $=$ current value

      $pbest$ location $=$ current location

    end

    compare particle's fitness evaluation with population's overall $gbest$

    if *current value is better than* $gbest$ **then**

      $gbest$ value $=$ current value

      $gbest$ location $=$ current location

      $gbest$ index $=$ current index

    end

    update the velocity and position of the particle

    $v_{id} = v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})$

    $x_{id} = x_{id} + v_{id}$

  end

end

简而言之，算法首先规定变量的取值范围并给出目标函数，初始化粒子群中各个粒子的初始位置、初始速度等参数，接着依据目标函数计算每个粒子的适应度，根据适应度更新粒子自身最佳位置 $pbest$ 和群体最佳位置 $gbest$，如此循环直至满足全局最优条件或到达迭代次数上限。简化的直观流程如下图 (3) 所示：

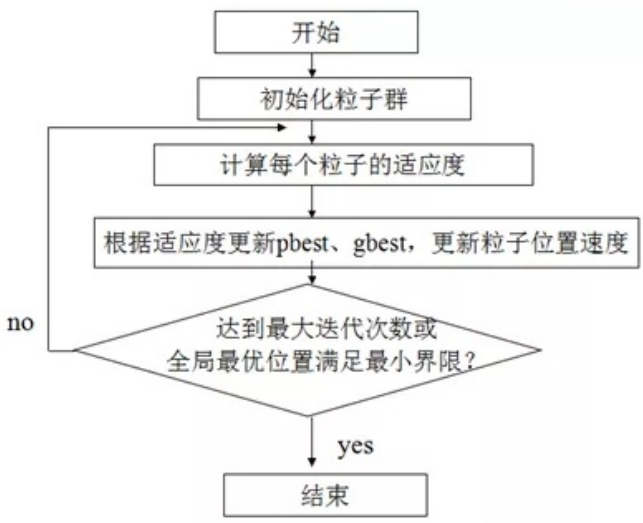

图 3: PSO 算法流程图

可以看出，随着计算的推移，通过探索和利用搜索空间中已知的有利位置，粒子围绕一个或多个最优点聚集或聚合。该算法设计玄妙之处在于它保留了最优全局位置和粒子已知的最优位置两个信息。后续的实验发现，保留这两个信息对于较快收敛速度以及避免过早陷入局部最优解都具有较好的效果。这也奠定了后续粒子群算法改进方向的基础。

2.2 算法实例

在这一部分，我们使用 Python 实现粒子群优化算法在简单问题中的应用，评估算法的表现。

2.2.1 例一

首先我们将所要优化的目标函数设为一个简单的二元函数：

$$\min y = \frac{1}{100}(x_1^2 + x_2^2)$$

其中 x_1 和 x_2 的取值范围为

$$x_1 \in [-300, 300], \quad x_2 \in [-300, 300]$$

将个体学习因子 $c1$ 和社会学习因子 $c2$ 都定为 0.5，设置粒子群大小为 50、迭代上限为 200 次，利用 PSO 算法求解该优化问题，得到最优结果为：

$$x_1^* = -1.1057,\ x_2^* = -0.3589,\ y^* = 0.0135$$

十分接近理论最优值 0。图 (4) 为目标函数值随迭代次数的变化，我们发现算法在大约 75 次迭代就能达到最优，所以无论从运行时间还是优化精度来说，粒子群优化算法的都是较为高效的。

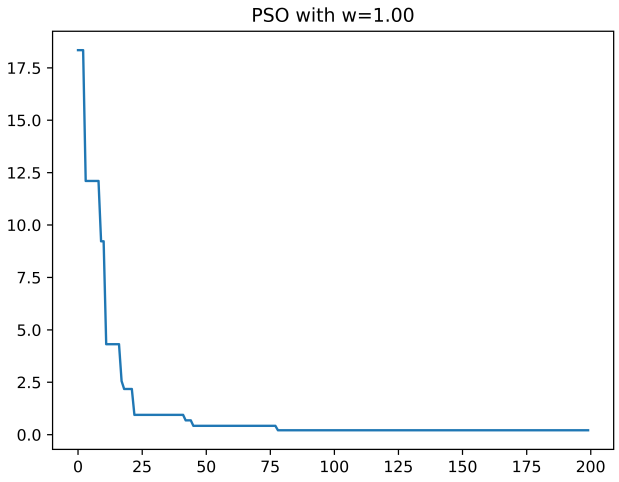

图 4: PSO 算法迭代过程目标函数值

当然，在处理此类较为简单的优化问题时，也许采用传统的凸优化方法会更加方便高效，但 PSO 的优势在于处理更为复杂的、无法计算目标函数的梯度的问题，这个小的例子只是为了展示 PSO 的工作原理。

2.2.2 例二

我们进一步探究算法在实际情景中的应用。假设一生产商的产品有 4 道生产工序，每道工序可以选择 3 种不同的机器，机器所生产产品的成本 c_{ij}、时间 t_{ij} 和质量 q_{ij} 不同，生产商的目标为在 4 道工序中合理选择不同种机器来达到成本、生产时间的最小化

和质量的最大化。问题的优化模型为：

$$
\begin{aligned}
&\min y = w_1 C + w_2 T - w_3 Q \\
s.t.\ &C = \sum_{i=1}^{4}\sum_{j=1}^{3} c_{ij} x_{ij} \\
&T = \sum_{i=1}^{4}\sum_{j=1}^{3} t_{ij} x_{ij} \\
&Q = \sum_{i=1}^{4}\sum_{j=1}^{3} q_{ij} x_{ij} \\
&x_{ij} = 1 \text{ if machine i is selected in task j, otherwise = 0} \\
&\sum_{j=1}^{3} x_{ij} = 1, \text{ for } i = 1,2,3,4
\end{aligned}
$$

生产商通过 AHP 方法确定下三个指标的权重分别为 $w_1 = 0.39$，$w_2 = 0.28$，$w_3 = 0.33$，求生产商最优的选择。

这是一个维数较高（$3\times4 = 12$）的问题，且涉及到整数规划，用常规优化方法进行求解则繁琐且效率不高，现用 PSO 算法尝试求解。图 (5) 为目标函数值随迭代次数的变化，我们发现在本例的情况下，即使不设置整数的要求，最终的最优结果仍能收敛到整数，且在极短的迭代次数内就可以得到最优结果。如果问题的维数更高，目标函数为更复杂的非凸函数，PSO 算法的优势将更加凸显。

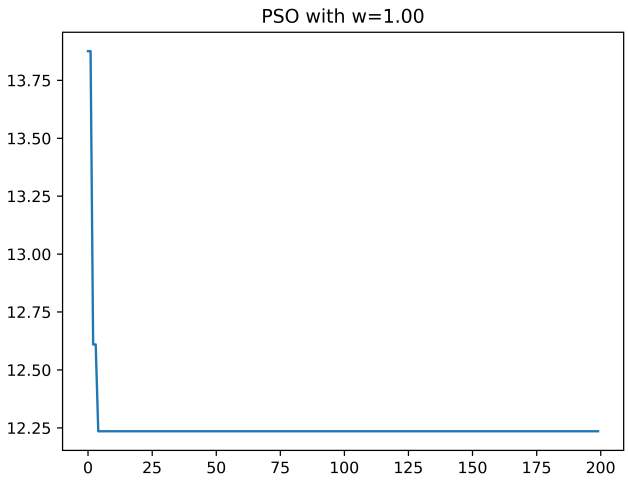

图 5: PSO 算法迭代过程目标函数值

3 算法发展及现状

由于粒子群优化算法容易陷入局部最优值、搜索初期收敛速度快而搜索后期收敛速度变慢等缺点，在其提出后许多学者对其进行了完善与发展。

3.1 添加惯性权重

三位学者在 1998 年的两篇文章 (Shi et al., 1998a) 和 (Shi et al., 1998b) 中引入了惯性权重（Inertia Weight）w，以下为加入了惯性权重的算法更新等式。

$$v_{id} = w * v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

与最初算法中的式子相比，添加惯性权重的变化仅仅是在 v_{id} 前面乘上系数 w，但选取合适的惯性权重能够更好地平衡全局与局部的开发（exploitation）和探索（exploration），于是作者将带惯性权重的 PSO 作为算法标准版本。

3.2 进化优化与粒子群优化结合

Angeline (1998) 将带有自然选择机制思想的进化优化（Evolutionary Optimization）引入粒子群算法中，其核心思想为：当算法更新完所有的粒子后，计算粒子的适应度值并对粒子进行适应度值排序，然后根据排序结果，用粒子群体中最好的一半粒子替换粒子群体中最差的一半粒子，但是保留原来粒子的个体最优位置信息。实验结果表明，自然选择机制的引入增强了粒子的全局寻优能力，提高了解的精度。

3.3 添加压缩因子

由于 PSO 来源于社会模型的启发，在提出时没有考虑算法的数学基础，而Clerc et al. (2002) 在 1999 年的研究中指出，使用压缩因子（Constriction Factor）也许能够保证算法的收敛性。其推导证明过于复杂，不在本文中讨论，简单来说，添加压缩因子即将原始算法的更新等式换为如下所示：

$$v_{id} = K * [v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})]$$

其中 K 是 c_1 与 c_2 的函数：

$$K = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}, \quad \text{where } \phi = c_1 + c_2,\ \phi > 4$$

压缩因子的引入在控制粒子群算法收敛的同时，使得粒子有机会搜索空间中不同的区域，并获得高质量的粒子。实验结果表明，它大大提高了粒子群算法的收敛速度和收敛精度。

3.4 自适应粒子群优化算法

Zhan et al. (2009) 在 2009 年提出了一种自适应的粒子群优化算法（Adaptive Particle Swarm Optimization, APSO），它有着比经典粒子群优化算法更好的搜索效率，且能够以更快的收敛速度在整个搜索空间中执行全局搜索。

APSO 包含两个主要步骤：首先，通过评估种群分布和粒子适应度，执行实时进化状态估计，算法可以在运行时自动控制惯性权重、加速度系数和其他算法参数，从而提高搜索效率和收敛速度。然后，当进化状态被分类为收敛状态时，执行精英学习（Elitist Learning）策略。该策略将作用于全局最佳粒子，以跳出可能的局部最优值。文章结果表明，APSO 在收敛速度、全局最优性、解决方案精度和算法可靠性方面大大提高了 PSO 范式的性能，且不会引入额外的设计或实现复杂性。

4 算法应用与展望

4.1 算法应用

从前文的分析中不难看出，粒子群算法可以广泛运用于优化理论当中。而这方面的研究是当今的前沿热点，尤其是随着计算机的广泛应用，计算性能的加强，优化技术不仅成为迫切需要，而且有了有效的求解工具。在这方面，典型的理论问题包括：组合优化、约束优化、多目标优化、动态系统优化等。实际工业应用有：电力系统、滤波器设计、自动控制、数据聚类、模式识别与图像处理、化工、机械、通信、机器人、经济、生物信息、医学、任务分配、TSP 等等。

在前文中，我们已经展示了粒子群算法在生产与运作管理方面的应用，可以帮助企业选择竞标公司，安排生产流水线等。下面我们再选取一些实际的例子，以展示粒子群算法在经济管理领域的应用。

4.1.1 配送中心的选址

在文章 (王非 等, 2011) 中，作者提到，配送中心选址问题是设施选址问题的重要研究领域之一。配送中心选址决策的优劣，不仅对配送中心本身、供应商及零售商的运营

成本、绩效产生重要影响，且对该地区区域经济的发展也具有突出作用。这个问题也是一个经典的运营管理的例子，也适用于使用 PSO 来寻求最优解。Daskin et al. (2002) 利用库存风险共担策略构建了适用于配送中心选址问题的非线性 0-1 整数规划模型。Shen et al. (2003) 用拉格朗日松弛法求解该模型并获得良好计算效果。该模型假设每个零售商需求都为正态独立分布，所有零售商需求必须满足且必须由一个配送中心供货，配送中心无配送能力限制，模型不仅计算成本的最优值，即最小成本，而且求解配送中心的最优建设区位与配送中心的配送方案。Shen et al. (2003) 首先利用风险共担策略将安全库存设在配送中心，进而采用 EOQ 经济订货批量模型计算配送中心的最优订货量。该模型在实际中取得了良好的效果。

4.1.2 金融风险模型

在金融领域，学者们主要将粒子群算法应用于投资组合优化、股指收益波动率预测、金融风险预警模型构建等方面。如何有效求解基数约束投资组合优化问题，是金融学界的研究热点。例如，基于 Markowitz M-V 投资组合优化问题可以以如下方式建模 (朱沙 等, 2016)。

假设有 N 种资产，r_i 表示第 i 种资产的期望收益 $(i = 1, \cdots, N)$，σ_{ij} 表示资产 i 和资产 j 之间的协方差 $(i, j = 1, \cdots, N)$。x_i 和 λ 分别表示资产 i 的投资比率和投资者风险厌恶系数。基于 Markowitz M-V 模型的投资组合选择问题描述为：

$$\min \lambda[\sum_{i=1}^{N}\sum_{j=1}^{N} x_i \cdot x_j \cdot \sigma_{ij}] - (1-\lambda)[\sum_{i=1}^{N} x_i \cdot r_i]$$

$$s.t. \sum_{i=1}^{N} x_i = 1$$

$$0 \le x_i \le 1, \text{ for } i = 1, \cdots, N$$

由问题的数学形式，非常容易看出，上述问题可以使用粒子群算法得到较好的解决。而事实上，使用 PSO 来解决 Markowitz 基数约束投资组合优化问题非常有效，不仅容易实现，而且优化结果符合实际要求。

4.2 展望

和其他算法一样，PSO 是一种新兴的智能优化算法，在理论和应用方面仍然有很多值得进一步研究的方面。虽然目前对 PSO 稳定性和收敛性的证明已取得了一些初步成果，但自诞生以来其数学基础一直不完备，特别是收敛性一直没有得到彻底解决。因此，仍需要对 PSO 的收敛性等方面进行进一步的理论研究。而由于 PSO 本质上是一种随机

搜索算法，如何提高该算法的可解释性，从而将该算法能与现有的工业技术相结合，并进行推广，还有很长的路要走。

在 PSO 算法研究方面，未来有如下研究方向 (黄磊, 2010)：

1. 应用领域的拓展: 目前该算法主要应用于函数优化及神经网络的训练等领域, 而开拓新的应用领域, 在应用的广度和深度上进行拓展都是很有价值的工作。
2. 算法参数的确定: PSO 中的一些参数如 c_1、c_2、w、微粒个数以及迭代次数等往往依赖于具体问题, 依据应用经验, 经多次测试来确定, 并不具有通用性。
3. 算法的改进研究: 由于实际问题的多样性和复杂性, 尽管已出现了许多改进的算法, 但远不能满足需要，如何根据不同的问题提出相应的改进算法是研究热点。
4. 算法的基础理论研究: PSO 的数学基础相对薄弱, 缺乏深刻且具有普遍意义的理论分析, 不能对 PSO 的工作机理做出合理的数学解释。虽然 PSO 的有效性、收敛性等性能在一些实例中得到验证, 但没能在理论上进行严谨推敲和严格证明。

不可否认的是，PSO 算法已经在实际中有非常广泛的应用。该算法非常简洁，结构简单，需要调节的参数少，需要的专业知识少，实现方式容易，所以在各种自然科学和工程领域中都有很多的应用，这方面的资料也非常丰富。

5 小结

从上述对 PSO 算法的探讨，我们不难发现，一个人工智能算法的成熟，除了需要在实际中得到很好的应用，而且需要有一定的可解释性、有可供选择的参数选择与指导准则，还需要有计算便捷性，理论完整性等。当然，随着人工智能领域在近年来的飞速发展，人工智能、机器学习算法的研究也朝气蓬勃，而这些领域还有很多可以完善、可以发展的部分。由于不同算法所使用工具和分析方法都有很强的相似性，因此，如果一个算法取得突破性进展，将会对整个领域产生非常重大的影响！而人工智能领域发展到现在，上述问题难度较大，也是迫切需要解决的问题。实际中这类问题往往增加很强的假设条件，或者神经网络深度过大，这还有很大的改进空间。

近年来，交叉学科的发展逐渐成为主流。对于信管的同学来说，信息管理，计算机科学，运筹学，数学，统计学等学科都需要深入了解，这能丰富我们的知识，做出有价值、有意义的成果。多领域的交叉发展是大势所趋，而学习粒子群算法等人工智能有关的知识，可以运用在生产与生活当中。特别地，如涉及到生产运作，决策理论，运筹学，优化，机器学习，人工智能，计算机技术等领域时，也可以使用人工智能的算法，探索问题的解，启发新的思路。

参考文献

吴庆洪, 张颖, 马宗民, 2010. 粒子群优化算法及其应用综述[J/OL]. 微计算机信息, 26(30):34-35,10. DOI: 10.3969/j.issn.2095-6835.2010.30.012.

朱沙, 陈臣, 2016. 一种求解基数约束投资组合优化的混合粒子群算法[J]. 统计与决策(10):64-67.

王非, 张佳, 孙浩杰, 等, 2011. 配送中心选址-库存问题的粒子群算法应用[J/OL]. 公路交通科技, 28(12): 152-158. https://doi.org/10.3969/j.issn.1002-0268.2011.12.026.

赵乃刚, 邓景顺, 2015. 粒子群优化算法综述[J]. 科技创新导报, 12(26):216-217.

黄磊, 2010. 粒子群优化算法综述[J]. 机械工程与自动化(05):197-199.

ANGELINE P J, 1998. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences[C]//International Conference on Evolutionary Programming. [S.l.]: Springer: 601-610.

CLERC M, KENNEDY J, 2002. The particle swarm-explosion, stability, and convergence in a multidimensional complex space[J]. IEEE transactions on Evolutionary Computation, 6(1):58-73.

DASKIN M, COULLARD C, SHEN Z J, 2002. An inventory-location model: Formulation, solution algorithm and computational results[J/OL]. Annals of Operations Research, 110(1):83-106. https://doi.org/10.1023/A:1020763400324.

KENNEDY J, EBERHART R, 1995. Particle swarm optimization[C]//Proceedings of ICNN'95-International Conference on Neural Networks: volume 4. [S.l.]: IEEE: 1942-1948.

SHEN Z J M, COULLARD C, DASKIN M S, 2003. A joint location-inventory model[J/OL]. Transportation Science, 37(1):40-55. https://doi.org/10.1287/trsc.37.1.40.12823.

SHI Y, EBERHART R, 1998a. A modified particle swarm optimizer[C]//1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360). [S.l.]: IEEE: 69-73.

SHI Y, EBERHART R C, 1998b. Parameter selection in particle swarm optimization[C]//International conference on evolutionary programming. [S.l.]: Springer: 591-600.

SHI Y, et al., 2001. Particle swarm optimization: developments, applications and resources[C]//Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546): volume 1. [S.l.]: IEEE: 81-86.

ZHAN Z H, ZHANG J, LI Y, et al., 2009. Adaptive particle swarm optimization[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(6):1362-1381.

附录

例一的 Python 代码实现：

```
from sko.PSO import PSO
import matplotlib.pyplot as plt

def demo_func(x):
  x1, x2 = x
  return (x1 ** 2 + x2 ** 2)/100.0

def pso_opt(w=1):
  pso = PSO(func=demo_func, dim=2, pop=50, max_iter=200, lb=[-300,
      -300], ub=[300, 300], w=w, c1=0.5, c2=0.5)
  pso.run()
  print('best_x is ', pso.gbest_x, 'best_y is', pso.gbest_y)
  figure, ax = plt.subplots(1, 1)
  ax.plot(pso.gbest_y_hist)
  ax.set_title("PSO with w=%.2f"%w)
  plt.show()


if __name__ == "__main__":
  pso_opt()
```

例二的 Python 代码实现：

```
from sko.PSO import PSO
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

m_cost = [[2, 1, 3],
      [3, 3, 5],
      [6, 3, 3],
      [9, 8, 6]]
m_time = [[2, 1, 1],
      [6, 3, 2],
      [4, 3, 5],
      [5, 7, 6]]
```

```
m_quality = [[9, 8, 9],
      [5, 7, 6],
      [8, 7, 5],
      [7, 7, 8]]
m_cost = np.mat(m_cost)
m_time = np.mat(m_time)
m_quality = 10 - np.mat(m_quality)
w = [0.39, 0.28, 0.33]

def obj_func(x):
  x_mat = np.mat(x)
  x_mat = np.reshape(x_mat, (4, 3))
  x_mat = pd.DataFrame(x_mat)
  temp = x_mat.apply(lambda y: y/sum(y), axis=1)
  x_mat = np.mat(temp)
  func_value = w[0]*np.multiply(x_mat, m_cost) + w[1]*np.multiply(x_mat
     , m_time) + w[2]*np.multiply(x_mat, m_quality)
  return func_value.sum(axis=0).sum(axis=1)[0, 0]

def run_pso(w=0.8, c1=0.5, c2=0.5):
  pso = PSO(func=obj_func, dim=12, pop=50, max_iter=200, lb=[0]*12, ub
     =[1]*12, w=w, c1=c1, c2=c2)
  pso.run()
  print('best_x␣is␣', pso.gbest_x, 'best_y␣is', pso.gbest_y)
  figure, ax = plt.subplots(1, 1)
  ax.plot(pso.gbest_y_hist)
  ax.set_title("PSO␣with␣w=%.2f"%w)
  plt.show()


if __name__ == "__main__":
  run_pso(w=1, c1=2, c2=2)
```