

群智能算法研究报告

冯梁铭, 21721143

(浙江大学, 计算机科学与技术学院, 人工智能研究所)

摘要 群智能是一种仿生自然界动物昆虫觅食筑巢行为的新兴演化计算技术。目前主要的群智能优化算法有蚁群算法、微粒群算法和人工鱼群算法。本次课程报告主要调研了狼群算法和经典的粒子群算法,介绍了算法的产生、发展和优点,并着力阐述这两种算法的基本原理,同时概述它们的应用现状,最后提出了算法将来有待研究的内容。

关键词 群智能算法 狼群算法 粒子群算法

中图分类号 TG007 **文献标识码** X

1 引言

与各种各样的自适应随机搜索算法相比,演化计算技术创造了被称为“种群”的潜在解,并通过种群间个体的相互协作与竞争来实现对优化问题最优解的搜索。这类方法一般能够比传统优化方法更快地发现复杂优化问题的最优解。群智能算法(Swarm Intelligence Algorithm,SIA)是人工智能的一个重要分支,起源于对人工生命的研宄,它作为一种新兴的演化计算方法已越来越受到国内外研宄者的关注。

文章的剩余部分将会如下组织:第二章将会介绍群智能算法的基本概念。第三章将会介绍狼群算法。第四章将会引入经典的粒子群算法。第五章将会编程实现狼群算法和粒子群算法,并选取了若干个测试函数对算法进行测试。最后一章会对两种算法的特点进行总结概括。

2 群智能算法介绍

群智能的概念最早由Beni,Hackwood 和Wang 在分子自动机系统中提出[1]。分子自动机中的主体在一维或二维网格空间中与相邻个体相互作用,从而实现自组织。群智能中的群,可被定义为“一组相互之间可以进行直接或间接通信(通过改变局部环境)的主体”。群的个体组织包括在结构上很简单的鸟群、蚁群、鱼群、蜂群等,而它们的集体行为却可能变得相当复杂。群智能则是指“无智能或简单智能的主体通过任何形式的聚集协作而表现出智能行为的特性”。群智能在没有集中且不提供全局模型的前提下,为寻找复杂分布式问题解决方案提供了基础。

目前,群智能算法研究领域主要存在有以下三种算法:蚁群算法(Ant Colony Optimization,ACO)、微粒群算法(Particle Swarm Optimization,PSO)和人工鱼群算法(Artificial Fishswarm Algorithm,AFA)。蚁群算法是对蚂蚁群落食物采集过程的模拟,已成功应用于许多离散优化问题。微粒群算法最初是模拟鸟群觅食的过程,但后来发现它是一种很好的优化工具。人工鱼群算法则是根据“水域中鱼生存数目最多的地方一般就是本水域中富含营养物质最多的地方”这一特点来模仿鱼群觅食行为而实现寻优的。

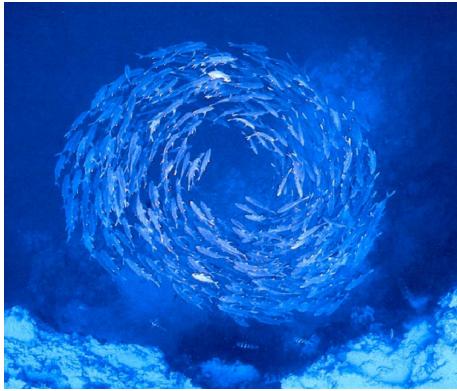


Fig 1: 海洋中鱼群



Fig 2: 呈V形的鸟群

自然界中有许多智能的种群，如图1和图2。群智能算法是一种概率搜索算法，与大多数基于梯度应用优化算法及传统的演化算法相比，其具有如下的几个特点[1]，这些特点可以通过观察自然界中的种群得到：

1. 采用完全分布式控制来实现个体与个体和个体与环境的交互作用，具有良好的自组织性
2. 个体之间的交流方式是非直接的，各个体通过对环境的感知（感觉能力）来进行合作，确保了系统具有更好的可扩展性和安全性
3. 没有集中控制的约束，系统具有更好的鲁棒性，不会因为个别个体的故障而影响整个问题的求解
4. 由于系统中单个个体的能力十分简单，只需要最小智能，这样每个个体的执行时间较短，实现起来比较方便，具有简单性

基于群智能的几个特点，我们可以概括出群体智能算法应该具有5个基本原则：

1. 接近性原则：群体应能够实现简单的时空计算
2. 优质性原则：群体能够响应环境要素
3. 变化相应原则：群体不应把自己的活动限制在一狭小范围
4. 稳定性原则：群体不应每次随环境改变自己的模式
5. 适应性原则：群体的模式应在计算代价值得时候改变

群智能算法的基本思想是模拟自然界生物的群体行为来构造随机优化算法。它将搜索和优化过程模拟成个体的进化或觅食过程，用搜索空间中的点模拟自然界中的个体，将求解问题的目标函数度量成个体对环境的适应能力；将个体的优胜劣汰过程或觅食过程类比为搜索和优化过程中用好的可行解取代较差可行解的迭代过程。因此，群智能算法是一种具有“生成+检验”特征的迭代搜索算法。群智能算法的原理将在狼群算法和粒子群算法中进行详细的说明。

3 狼群算法

灰狼优化算法(Grey Wolf Optimizer, GWO)是一种模拟灰狼捕食行为的群体智能算法，该算法最先由澳大利亚学者Mirjalili于2014年提出[2]，根据灰狼的社会等级将包围、追捕、攻击等捕食任务分配给不同等级的灰狼群来完成捕食行为，从而实现全局优化的过程。GWO 算法具有操作简单、调节参数少、编程易实现等特点。在函数优化方面，与其他群智能优化算法相比有明显的优越性。但同时也存在着易陷入局部最优、求解精度不高、收敛速度慢等缺点。

3.1 灰狼群介绍

灰狼是现存犬科动物中体型最大的物种，是食物链顶端的顶级捕食者。通常是5~10只组成一群，在这一小型群体中，有一只领头的雄狼，所有的雄狼常被依次分在甲、乙、丙各等级，雌狼亦是如此。狼群的等级划分可以参见图3。

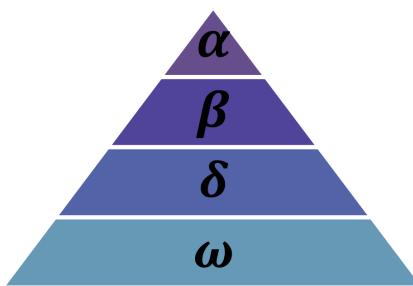


Fig 3: 狼群等级结构

狼群的首领由一条公狼和一条母狼组成，称为 α 。 α 主要负责决定打猎、睡觉的地方、醒来的时间等等。只有 α 狼才拥有狼群中的交配权。有趣的是， α 并不一定是最强大的成员，但一定是最好的管理者。这表明在一个组织中，组织和纪律比力量更加重要。

灰狼等级制度中的第二个等级是 β 。 β 是头狼 α 的下属，帮助 α 进行决策。 β 可能是头狼 α 的最佳候选者，以防止头狼意外死亡或衰老。 β 应该服从阿尔法，但也要指挥其他低等级的狼。

排名最靠后的灰狼是 ω ，它们扮演替罪羊的角色。 ω 狼总是屈服于其他所有的狼。他们是最后被允许进食的狼。虽然看起 ω 不是很重要，但是在狼群中， ω 可以调节种群的内部关系。

如果狼不是 α, β, ω 之中的一种，它则被称为下属（ δ ）。狼 δ 需要服从于 α 和 β ，但它们可以命令 ω 。 ω 一般由经验丰富的年长者担任，它们可能是曾经的 α 或 β 。它们可以进行狩猎，侦查，以及照顾受伤的灰狼个体。

除了灰狼的社会等级，灰狼群的另一个重要特征是群体捕猎。相关的研究把狼群的捕猎行为分为三类(图4)：

- 跟踪, 追逐, 接近猎物
- 追逐, 包围, 骚扰猎物, 直到猎物停止移动
- 向猎物发起攻击



Fig 4: 狼群的狩猎行为: A:跟踪猎物; B-D: 追逐包围猎物; E: 发起攻击

3.2 数学模型和算法

在GWO 算法中，领导能力最强的灰狼被记为 α ，主要负责捕猎(寻优)过程中的决策部分及管理狼群。剩下的灰狼个体按社会等级被依次记为 β, δ 和 ω 。 α 狼是整个灰狼群在捕猎过程中的领导者，是最有智慧和能力最强的个体(即其适应度最佳、离最优值最接近的狼)； β 狼和 δ 狼是适应度次佳的两个个体，捕猎中它们会协助 α 狼对灰狼群的进行管理及捕猎过程中的决策问题，同时也是 α 狼的候选者；剩余的狼群被定义为 ω ，其主要职责是平衡灰狼种群的内务关系及协助 α, β, δ 对猎物进行攻击。在整个捕猎过程中，首先由 α 狼带领狼群搜索、跟踪、接近猎物，当距离猎物的范围足够小时， β, δ 狼在 α 的指挥下对猎物进行围攻，并召唤周围的 ω 狼对猎物进行攻击，当猎物移动时，狼群形成包围猎物的圈也随之移动，直至捕获猎物。

3.2.1 围猎encircling

围猎行为是狼群向猎物靠拢的过程，该过程用数学模型表达为：

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \quad (1)$$

$$\vec{X}(t+1) = \vec{X}(t) - \vec{A} - \vec{D} \quad (2)$$

其中， t 代表当前的迭代次数， \vec{A} 和 \vec{C} 是系数向量， \vec{X}_p 是当前估计的猎物位置向量， \vec{X} 是灰狼的位置向量。

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \quad (3)$$

$$\vec{C} = 2 \cdot \vec{r}_2 \quad (4)$$

其中， \vec{a} 取值在0和2之间，且在迭代过程中逐渐变小， \vec{r}_1 和 \vec{r}_2 是取值在[0, 1]之间的随机向量。

3.2.2 捕猎hunton

灰狼有能力识别猎物的位置并包围它们。狩猎通常由 α 引导。 β 和 δ 也可能偶尔参与狩猎。然而，在一个抽象的搜索空间中，我们不知道最优（猎物）的位置。为了对灰狼的捕食行为进行数学模拟，

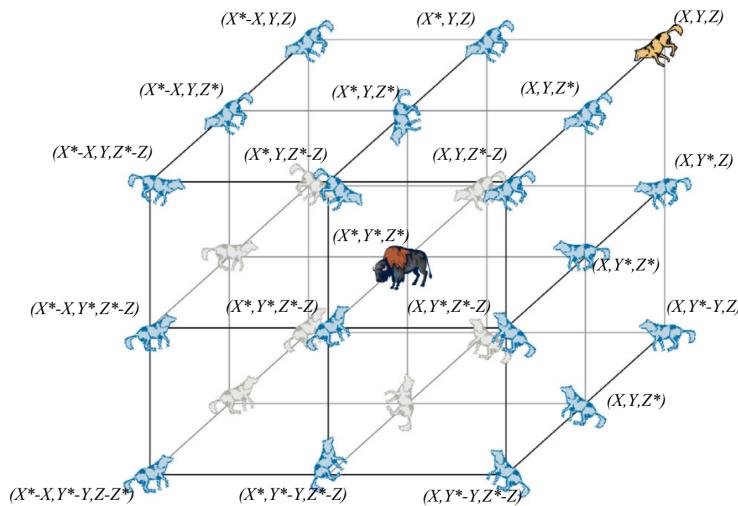


Fig 5: 3D空间中灰狼的下一个可能位置

我们假设 α （最佳候选解）, β 和 δ 对猎物的潜在位置有更好的了解。因此，我们把前三个可能最优解记录下来，其他个体根据这三个解得位置进行位置移动。据此提出了以下公式：

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|, \quad \vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|, \quad \vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}| \quad (5)$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha, \quad \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta, \quad \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta \quad (6)$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad (7)$$

图6是在2D空间中，某个狼个体根据 α, β, δ 位置决定自己移动行为的示意图。换而言之， α, β, δ 估计了猎物的可能位置，其他狼个体综合这些信息进行捕猎。

3.2.3 攻击attacking

当猎物停止移动或者进入狼群攻击范围时，狼群会发动攻击。在数学模型上，狼群的攻击行为通过降低 \vec{a} 的值来实现。注意到随着 \vec{a} 值的降低， $|\vec{A}|$ 的值也在下降。换而言之，在算法迭代过程中， $|\vec{A}|$ 在区间 $[-2a, +2a]$ 之间变化。当 $|\vec{A}|$ 的值小于1时，说明狼将会向猎物方向移动（图7a），而 $|\vec{A}| > 1$ 则会使狼远离猎物。

在狼群的移动过程中，GWO算法还引入了“狼群的替换机制”。也就是说 α, β, δ 不是固定的，而是动态变化的，取代标准则是适应度。

3.2.4 搜寻searching

灰狼群通常会根据 α, β, δ 的位置来更新自己的位置。为了避免陷入局部最优解，GWO算法引入 \vec{A} 来调整攻击（大致确定最优解并向其靠近）和搜寻（寻找最优解）两种行为。如图7b所示，当 $|\vec{A}| > 1$ 时，灰狼个体将会放弃当前猎物（局部最优解），而去选择另一个更优的猎物。除此之外，FWO算法还引入了另外一个变量 \vec{C} 来平衡这两种行为。 \vec{C} 是一个随机变量，其元素的值在[0,2]之间变化， \vec{C} 相当于灰狼个体对当前猎物的重视程度（ $C > 1$: 重视; $C < 1$: 不重视;）。与 \vec{A} 不同，在迭代过程中， \vec{C} 一直是随机变化的而非线性降低的，这是为了避免算法陷入局部最优解。

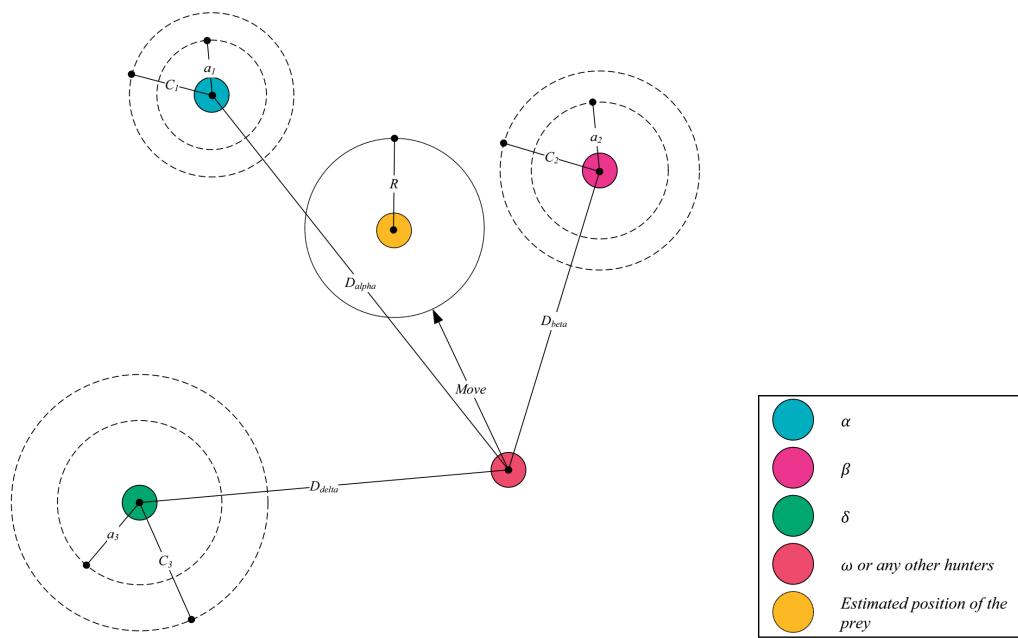


Fig 6: 狼群hunting行为示意图

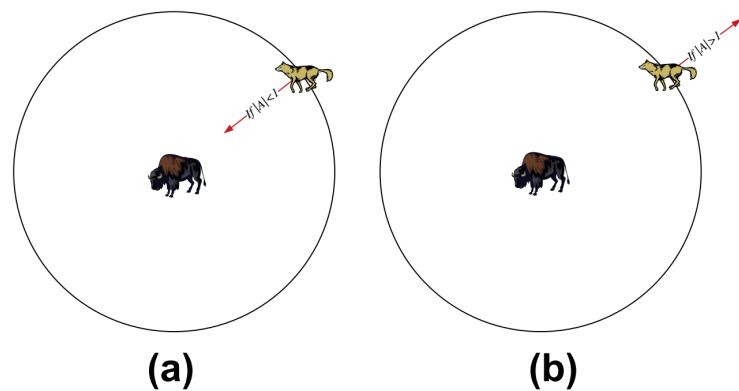


Fig 7: 攻击猎物(a)和搜寻猎物(b)

总而言之，GWO算法的搜索过程最初初始化一定数量的灰狼个体，并计算选取 α, β, δ 。其他灰狼个体根据 α, β, δ 的位置进行搜寻和攻击。相应的，参数 \bar{A} 从2逐渐降低至0用来强调搜索。当 $\bar{A} > 1$ 时，灰狼个体会从猎物处散开，而当 $\bar{A} < 1$ 时，灰狼则会向猎物移动最后发动攻击。最后，GWO算法将会在达到终止标准时停止搜索过程。算法的具体描述如下：

Algorithm 1 GWO

```

Initialize the grey wolf population  $X_i (i = 1, 2, 3, \dots, n)$ 
Initialize a, A and C
Calculate the fitness of each search agent
 $X_\alpha$ : the best search agent
 $X_\beta$ : the second best search agent
 $X_\delta$ : the third best search agent
while  $t <$  Max number of iterations do
    for each search agent do
        Update the position of current search agent
    end for
    Update a, A and C
    Calculate the fitness of all search agents
    Update  $X_\alpha$ ,  $X_\beta$  and  $X_\delta$ 
     $t = t + 1$ 
end while
return  $X_\alpha$ 

```

4 粒子群算法

粒子群优化算法是在1995年由Eberhart博士和Kennedy博士一起提出的，它源于对鸟群捕食行为的研究。它的基本核心是利用群体中的个体对信息的共享从而使得整个群体的运动在问题求解空间中产生从无序到有序的演化过程，从而获得问题的最优解。我们可以利用一个有关PSO的经典描述来对PSO算法进行一个直观的描述。设想这么一个场景：一群鸟进行觅食，而远处有一片玉米地，所有的鸟都不知道玉米地到底在哪里，但是它们知道自己当前的位置距离玉米地有多远。那么找到玉米地的最佳策略，也是最简单有效的策略就是搜寻目前距离玉米地最近的鸟群的周围区域。PSO就是从这种群体觅食的行为中得到了启示，从而构建的一种优化模型。

在PSO中，每个优化问题的解都是搜索空间中的一只鸟，称之为“粒子”，而问题的最优解就对应为鸟群要寻找的“玉米地”。所有的粒子都具有一个位置向量（粒子在解空间的位置）和速度向量（决定下次飞行的方向和速度），并可以根据目标函数来计算当前的所在位置的适应值（fitness value），可以将其理解为距离“玉米地”的距离。在每次的迭代中，种群中的粒子除了根据自身的“经验”（历史位置）进行学习以外，还可以根据种群中最优粒子的“经验”来学习，从而确定下一次迭代时需要如何调整和改变飞行的方向和速度。就这样逐步迭代，最终整个种群的粒子就会逐步趋于最优解。

4.1 粒子群算法基本框架

在PSO算法中，有两个非常重要的参数：粒子的位置 $X_i = (x_{i1}, x_{i2}, \dots, x_{in})$ 和粒子的速度 $V_i = (v_{i1}, v_{i2}, \dots, v_{in})$ 。最初粒子群优化算法的迭代算子形式如下：

$$V_{i+1} = V_i + c_1 r_1 (Pbest_i - X_i) + c_2 r_2 (Gbest_i - X_i) \quad (8)$$

$$X_{i+1} = X_i + V_i \quad (9)$$

其中在公式8中， $Pbest_i$ 和 $Gbest_i$ 分别代表粒子ii的历史最佳位置向量和种群历史最佳位置向量。根据公式??可以看出，种群中的粒子通过不断地向自身和种群的历史信息进行学习，从而可以找出问题的最优解。

但是，在后续的研究中表明，上述原始的公式中存在一个问题：公式8中 V_i 的更新太具有随机性，从而使得整个PSO算法的全局优化能力很强，但是局部搜索能力较差。而实际上，我们需要在算法迭代初期PSO有着较强的全局优化能力，而在算法的后期，整个种群应该具有更强的局部搜索能力。所以根据上述的弊端，通过引入惯性权重修改了公式8，从而提出了PSO的惯性权重模型：

$$V_{i+1} = wV_i + c_1r_1(Pbest_i - X_i) + c_2r_2(Gbest_i - X_i) \quad (10)$$

其中参数w称为是PSO的惯性权重（inertia weight），它的取值介于[0,1]区间，一般应用中均采取自适应的取值方法，即一开始令w=0.9，使得PSO全局优化能力较强，随着迭代的深入，参数w进行递减，从而使得PSO具有较强的局部优化能力，当迭代结束时，w=0.1。参数 c_1 和 c_2 称为是学习因子（learn factor），一般设置为1.4961或2；而 r_1 和 r_2 为介于[0,1]之间的随机概率值。

整个粒子群优化算法的算法框架如下：

Algorithm 2 PSO

```

Initialize the particle's population  $X_i (i = 1, 2, 3, \dots, m)$  and velocity  $V_i (i = 1, 2, 3, \dots, m)$ 
Calculate the fitness of each agent
while  $t < \text{Max number of iterations}$  do
    for each search agent do
        Update the position of current agent
        Update the velocity of current agent
    end for
    Update  $Pbest_i$  and  $Gbest_i$ 
     $t = t + 1$ 
end while
return  $Gbest$ 

```

4.2 粒子群算法中参数说明

惯性权重w，顾名思义w实际反映了粒子过去的运动状态对当前行为的影响，就像是我们物理中提到的惯性。如果 $w << 1$ ，从前的运动状态很少能影响当前的行为，粒子的速度会很快的改变；相反，w较大，虽然会有很大的搜索空间，但是粒子很难改变其运动方向，很难向较优位置收敛，由于算法速度的因素，在实际运用中很少这样设置。也就是说，较高的w设置促进全局搜索，较低的w设置促进快速的局部搜索。

群体规模m。一般而言，群体规模越大，相互协同搜索的粒子就越多，信息就越充分，能更好的发挥PSO的搜索能力。虽然群体的规模增大，能使PSO算法的全局优化能力增强，但成本是搜索计算的时间增长，而且收敛到全局最优点的速度会明显减慢。如果规模过小，又会陷入局部最优。

学习因子 c_1 和 C_2 。学习因子是控制粒子自我历史经验和群体最优个体学习的因子，控制着粒子向自己历史最优位置和全局最优位置移动的步伐。它也起到了平衡局部搜索和全局搜索的能力。与惯性权重不同的是，学习因子越大，局部搜索的能力越强，也越有利于算法的收敛。

表 1: unimodal functions

function	dim	range	f_{min}
$f_1(x) = \sum_{i=1}^n x_i^2$	30	[-100,100]	0
$f_2(x) = \sum_{i=1}^n x_i + \prod_{i=1}^n x_i $	30	[-10,10]	0
$f_3(x) = \sum_{i=1}^n (\sum_{j=1}^i x_j)^2$	30	[-100,100]	0
$f_4(x) = \max_i\{ x_i , 1 \leq i \leq n\}$	30	[-100,100]	0
$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	[-30,30]	0

表 2: multimodal functions

function	dim	range	f_{min}
$f_6(x) = \sum_{i=1}^n -x_i \sin(\sqrt{ x_i })$	30	[-500,500]	-418.9829×5
$f_7(x) = \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i) + 10]$	30	[-5.12,5.12]	0
$f_8(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{2} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e$	30	[-32,32]	0
$f_9(x) = \frac{1}{4000} \sum_{i=1}^n nx_1^2 - \prod_{i=1}^n \cos\left(\frac{x_1}{\sqrt{i}}\right) + 1$	30	[-600,600]	0
$f_{10}(x) = \frac{\pi}{n} \{10 \sin(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + 10 \sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^n u(x_i, 10, 100, 4)$	30	[-50,50]	0

5 实验

通过编程实验来检测GWO和PSO两类算法在求解函数最小值问题上的能力。相关的参数设置为：population为50，最大迭代次数为500。对于PSO算法，额外的参数设置为：最大速度 V_{max} 设置为6， w_{max} 为0.9， w_{min} 为0.2，学习因子 c_1 和 c_2 设置为2。

5.1 测试函数

测试函数分为两大类：单峰函数（见表5.1）和多峰函数（见表5.1）。GWO算法和PSO算法在每个benchmark函数上运行30次，统计平均结果和方差。

5.2 实验结果

实验结果汇总在表3当中。对于单峰函数而言，GWO算法能够提供非常优秀的结果，在测试用例上要优于PSO算法。这说明GWO算法有不错的搜素能力。至于多峰函数，相较于单峰函数，优化算法在测试函数上更加容易出现陷入局部最优解的问题。实验结果表明，GWO算法也具有不错的求解能力，在部分的测试用例上，GWO的求解能力要优于PSO。这里的实验主要是针对于无约束条件下的优化问题，而实际工程中的求解往往有多个约束条件。由于GWO算法的更新机制并不直接与适应度值相关，而是与三个群体最优解位置有关，所以在解决约束条件下的优化问题中可以引入惩罚函数，对不满足约束条件的个体施加一定的惩罚。

如前所述，GWO算法通过调整 \vec{A} 的值来避免陷入局部最优解的问题，测试表明GWO的全局搜索能力可以满足大部分应用要求。但是从算法上来看，GWO算法每个搜索节点的位置更新只取决于具有最优适应度的三个个体，这就会导致狼群的个体之间具有很强的同质性。这种同质性很容易导致搜

表 3: GWO与PSO对比

function	min	GWO		PSO	
		ave	std	ave	std
F1	0	6.59e-28	6.34e-05	1.36e-04	2.02e-04
F2	0	7.18e-17	0.029	4.21e-02	0.045
F3	0	3.29e-06	79.149	70.125	22.119
F4	0	5.61e-07	1.315	1.086	0.317
F5	0	26.81258	69.904	96.718	60.116
F6	-2094.91	-6123.1	-4087.44	-4841.3	1152.814
F7	0	0.310	47.356	46.704	11.629
F8	0	1.06e-13	0.078	0.276	0.509
F9	0	0.004	0.007	0.009	0.008
F10	0	0.053	0.020	0.007	0.026

索求解过程陷入局部最优的状况。与之相反，PSO 算法再设计上，粒子的位置更新有三个参考标准：群体当前最优位置，个体历史最优位置以及粒子当前的搜索惯性。这样一来，PSO 具有不错的全局搜索能力，能够有效的避免局部最优解。

6 分析与讨论

6.1 狼群算法的改进

正如前文所提及，GWO 算法具有操作简单、调节参数少、编程易实现等特点。在函数优化方面，与其他群智能优化算法相比有明显的优越性。但同时也存在着易陷入局部最优、求解精度不高、收敛速度慢等缺点。有许多学者有针对性的提出了一系列的优化方法：[3]采用计算分配值的方法提出一种自适应搜索的灰狼求解算法从而加快算法的收敛速度[4]将混沌序列方法引入初始化种群个体，给出了一种寻优性和鲁棒性更好的改进GWO 算法[5]引入了佳点集理来初始化狼群，并用非固定多段映射罚函数法处理约束条件，利用改进GWO 算法求解约束优化问题[6]引入了小生境概念，提出了小生境灰狼优化算法(Niche Grey Wolf Optimization, NGWO)，该算法利用基本GWO 算法计算每个灰狼的适应度值。当灰狼距离小于生存半径时，对适应度较差的个体施加惩罚函数，以提高全局搜索能力。类似的思路还有淘汰掉适应度较差的种群个体，在引入相同数量随机初始化的个体来增强全局搜索能力。

6.2 智能算法展望

群体中相互合作的个体是分布式的(Distributed)，这样更能够适应当前网络环境下的工作状态；没有中心的控制与数据，这样的系统更具有鲁棒性(Robust)，不会由于某一个或者某几个个体的故障而影响整个问题的求解。可以不通过个体之间直接通信而是通过非直接通信(Stimergy)进行合作，这样的系统具有更好的可扩充性(Scalability)。由于系统中个体的增加而增加的系统的通信开销在这里十分小。系统中每个个体的能力十分简单，这样每个个体的执行时间比较短，并且实现也比较简单，具有简单性(Simplicity)。研究表明，群智能算法作为一类新型进化算法，能够有效解决大多数优化

问题,并且,其潜在的并行性和分布式特点为处理大量的以数据库形式存在的数据提供了技术保证。因为具有这些优点,可以预言群集智能的研究代表了以后计算机研究发展的一个重要方向。然而,相对于其它比较成熟的优化算法来说,群智能算法的研究还处于起步阶段,还存在很多问题有待深入研究,归纳起来有以下几点[7]:

- 理论基础的研究:群智能算法的数学理论基础还比较薄弱,缺乏具有普遍意义的理论分析。算法中各种参数的设置通常都是按照经验法确定,没有确切的理论依据,对应用环境和具体问题的依赖性比较大。
- 比较性和融合性研究:群智能算法与其它比较成熟的优化算法之间的比较性研究不足,与其它各种智能方法和先进技术的融合性研究还不够充分。
- 各算法适用范围的研究:“无免费午餐”定理表明不存在适用于任何问题的优化算法,因此,深入研究各群智能算法的适用范围并进一步拓展各算法的应用领域将是一个十分必要的课题。

参考文献

- [1] HACKWOOD S,BENI G.Self-organization of sensors for Swarm Intelligence[C].IN:IEEE International conference on Robotics and Automation.Piscataway,NJ:IEEE Press,1992:819-829. media. SDM. SIAM.
- [2] Mirjalili, S., Mirjalili, S. and Lewis, A. (2014) Grey Wolf Optimizer. *Advances in Engineering Software*, 69, 46-61
- [3] 罗佳, 唐斌. 新型灰狼优化算法在函数优化中的应用[J]. 兰州理工大学学报, 2016, 6(3): 97-101
- [4] 魏政磊, 赵辉, 韩邦杰, 等. 基于自适应GWO 的多UCAV 协同攻击目标决策[J]. 计算机工程与应用, 2016, 25(18): 97-101
- [5] 龙文, 赵东泉, 等. 求解约束优化问题的改进灰狼优化算法[J]. 计算机应用, 2015, 35(9): 2590-2595
- [6] 白媛, 陈京荣, 展之婵. 改进灰狼优化算法的研究与分析[J]. 计算机科学与应用, 2017, 7(6): 562-571
- [7] E BONABEAU , M DORIGO , G THERAULAZ. Swarm Intelligence:From Natural to Artificial Systems [M] .New York: Oxford University Press,1999.

A Code of GWO

```
import random
import numpy
import math
from solution import solution
import time

def GWO(objf,lb,ub,dim,SearchAgents_no,Max_iter):
    #objf: benchmark function
    #lb,ub: lower bound, upper bound
    #dim: dim of X
    #SearchAgents: number of wolves

    # initialize alpha, beta, and delta_pos
    Alpha_pos=numpy.zeros(dim)
    Alpha_score=float("inf")

    Beta_pos=numpy.zeros(dim)
    Beta_score=float("inf")

    Delta_pos=numpy.zeros(dim)
    Delta_score=float("inf")

    #Initialize the positions of search agents
    Positions=numpy.random.uniform(0,1,(SearchAgents_no,dim)) *(ub-lb)+lb

    Convergence_curve=numpy.zeros(Max_iter)
    s=solution()

    # Loop counter
    print("GWO is optimizing \\""+objf.__name__+"\\")"

    timerStart=time.time()
    s.startTime=time.strftime("%Y-%m-%d-%H-%M-%S")
    # Main loop
    for l in range(0,Max_iter):
        for i in range(0,SearchAgents_no):

            # Return back the search agents that go beyond the boundaries of the search
            # space
            Positions[i,:]=numpy.clip(Positions[i,:], lb, ub)

            # Calculate objective function for each search agent
            fitness=objf(Positions[i,:])

            # Update Alpha, Beta, and Delta
            if fitness<Alpha_score :
                Alpha_score=fitness; # Update alpha
```

```
Alpha_pos=Positions [i,:]

if (fitness>Alpha_score and fitness<Beta_score ):
    Beta_score=fitness # Update beta
    Beta_pos=Positions [i,:]

if (fitness>Alpha_score and fitness>Beta_score and fitness<Delta_score):
    Delta_score=fitness # Update delta
    Delta_pos=Positions [i,:]

a=2-1*((2)/Max_iter); # a decreases linearly from 2 to 0

# Update the Position of search agents including omegas
for i in range(0,SearchAgents_no):
    for j in range (0,dim):

        r1=random.random() # r1 is a random number in [0,1]
        r2=random.random() # r2 is a random number in [0,1]

        A1=2*a*r1-a
        C1=2*r2

        D_alpha=abs(C1*Alpha_pos[j]-Positions[i,j]) #calculte the distance to alpha
        X1=Alpha_pos[j]-A1*D_alpha                         #run towards alpha

        r1=random.random()
        r2=random.random()

        A2=2*a*r1-a
        C2=2*r2

        D_beta=abs(C2*Beta_pos[j]-Positions[i,j])
        X2=Beta_pos[j]-A2*D_beta

        r1=random.random()
        r2=random.random()

        A3=2*a*r1-a
        C3=2*r2

        D_delta=abs(C3*Delta_pos[j]-Positions[i,j])
        X3=Delta_pos[j]-A3*D_delta

        Positions [i,j]=(X1+X2+X3)/3

Convergence_curve [l]=Alpha_score;
```

```
if (l%1==0):
    print(['At iteration '+ str(l)+ ' the best fitness is '+ str(Alpha_score)]);

timerEnd=time.time()
s.endTime=time.strftime("%Y-%m-%d-%H-%M-%S")
s.executionTime=timerEnd-timerStart
s.convergence=Convergence_curve
s.optimizer="GWO"
s.objfname=objf.__name__

return s
```

B Code of PSO

```
import random
import numpy
import math
from colorama import Fore, Back, Style
from solution import solution
import time

def PSO(objf ,lb ,ub ,dim ,PopSize ,iters):

    # PSO parameters
    #dim=30
    #iters=200
    Vmax=6
    #PopSize=50      #population size
    wMax=0.9
    wMin=0.2
    c1=2
    c2=2
    #lb=-10
    #ub=10
    s=solution()
    # Initializations
    vel=numpy.zeros((PopSize ,dim))
    pBestScore=numpy.zeros(PopSize)
    pBestScore.fill(float("inf"))
    pBest=numpy.zeros((PopSize ,dim))
    gBest=numpy.zeros(dim)

    gBestScore=float("inf")
    pos=numpy.random.uniform(0,1,(PopSize ,dim)) *(ub-lb)+lb
    convergence_curve=numpy.zeros(iters)

    print("PSO is optimizing \\""+objf.__name__+"\\")
```

```
timerStart=time.time()
s.startTime=time.strftime("%Y-%m-%d-%H-%M-%S")
for l in range(0,iters):
    for i in range(0,PopSize):
        #pos[i,:]=checkBounds(pos[i,:],lb,ub)
        pos[i,:]=numpy.clip(pos[i,:], lb, ub)
        #Calculate objective function for each particle
        fitness=objf(pos[i,:])

        if(pBestScore[i]>fitness):
            pBestScore[i]=fitness
            pBest[i,:]=pos[i,:]

        if(gBestScore>fitness):
            gBestScore=fitness
            gBest=pos[i,:]
    #Update the W of PSO
    w=wMax-l*((wMax-wMin)/iters);
    for i in range(0,PopSize):
        for j in range (0,dim):
            r1=random.random()
            r2=random.random()
            vel[i,j]=w*vel[i,j]+c1*r1*(pBest[i,j]-pos[i,j])+c2*r2*(gBest[j]-pos[i,j])
            if(vel[i,j]>Vmax):
                vel[i,j]=Vmax
            if(vel[i,j]<-Vmax):
                vel[i,j]=-Vmax
            pos[i,j]=pos[i,j]+vel[i,j]
    convergence_curve[l]=gBestScore
    if (l%1==0):
        print(['At iteration '+ str(l+1)+ ' the best fitness is '+ str(gBestScore)])
        ;
timerEnd=time.time()
s.endTime=time.strftime("%Y-%m-%d-%H-%M-%S")
s.executionTime=timerEnd-timerStart
s.convergence=convergence_curve
s.optimizer="PSO"
s.objfname=objf.__name__

return s
```