第3章

群体智能

本章学习目标

- 理解群体智能各个算法的基本概念。
- 熟练掌握各个原始算法的基本流程和数学表达式。
- 了解各个改进算法的基本思想。

本章为读者介绍了经典的群体智能算法的基本概念框架、算法流程及改进算法，具体包括粒子群优化算法、蚁群算法、蜂群算法、烟花算法、头脑风暴优化算法等。

3.1 概念及原理

3.1.1 基本概念

在过去几十年中，全局优化在科学研究和工程应用的众多领域中扮演了重要角色。受到生物集体行为启发的群体智能(Swarm Intelligence，SI)引领了一种新型群体智能算法的发展。基于种群的优化算法已被广泛用于解决实际生产中的多种复杂优化问题。与传统的单点搜索算法(如爬山算法)不同，基于种群的优化算法涉及一组种群通过信息共享来解决问题，从而实现个体间的合作或竞争。

随着研究的深入，各种基于种群的智能算法相继出现。最初的进化计算是最早的基于种群的算法，灵感来源于生物进化。随着优化算法的发展，多种基于群体智能的算法涌现，这些算法由自然启发而来，被称为自然启发式优化算法或群体智能算法。众多启发式优化算法已被提出，包括蚁群优化算法、粒子群优化算法、人工蜂群算法和细菌觅食优化算法等。

3.1.2 基本框架

在群体智能算法中，每个个体代表一个对象，所有个体通过协作在搜索空间中共同寻求更优解。在群体智能算法中，个体通常代表简单对象，例如，粒子群优化算法中的鸟。群体智能系统具有自组织特性，能够在无外界干预的情况下自行完成任务。系统中每个个体都能独立进行空间探索，尽管个体的搜索能力相对有

限。个体之间的通信与协作促使群体智能的形成，使群体能够完成复杂任务，这种智能体间的互动对群体智能的形成至关重要。个体之间的交互可以分为三类：探索、模仿和学习。即群体智能算法通过这三种策略生成新的可行解时，随着群体的迭代收敛到最优解。群智能算法的搜索过程如图 3.1 所示。首先，探索保证个体集群在决策空间搜索的准确性和可行解的多样性，尤其是在没有先验知识的情况下。其次，模仿其他个体是个体的交互过程，进而产生新的可行解。最后，学习的目的是缩短当前解和目标解之间的差距，目标解通常是局部或全局最优解。因此，学习可以确保智能体搜索更好的可行解，也可以确保群智能算法的收敛性。

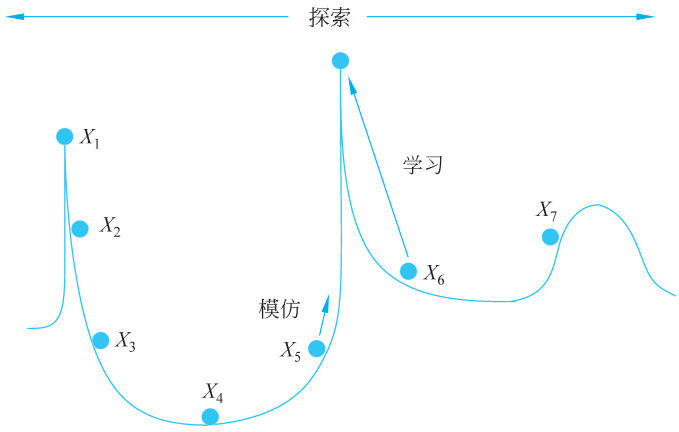


图 3.1　群智能算法的搜索过程

图 3.2 展示了群体智能算法的一般过程。开采操作描述了算法在不同搜索阶段自适应调整参数和结构的能力。探索操作作为底层操作符，描述了算法如何在搜索空间中生成新的可行解。在每次迭代中，种群中的个体通过探索和开发获得搜索空间中的可行解，通过相互协作最终寻找到最优解。

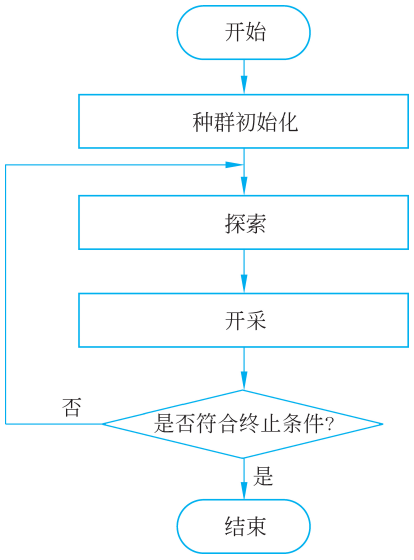


图 3.2　群智能算法的基本流程图

3.2　粒子群优化算法

优化问题在科学技术各领域广泛存在，实际应用中的优化问题通常比较复杂，涉及复杂的目标函数或条件约束。传统优化方法主要基于数学方法求解最优解，如梯度下降法等。传统优化方法已广泛应用于多种优化问题。然而，这些方法具有局限性，包括易陷入局部最优、高计算复杂度，以及不适用于某些类型的目标函数等。相比之下，启发式优化算法是自组织和自适应的，能够快速高效地搜索最优解。1995 年，Kennedy 和 Eberhart 首次引入了粒子群优化(Particle Swarm Optimization，PSO)作为一种新的启发式方法。它从单纯利用数学模型模拟鸟群集体行为发展到将此行为转换为简单高效的优化工具。PSO 算法主要用于处理非线性连续优化问题。随着对 PSO 算法的深入研究，许多改进算法被提出，显著增强了其解决复杂工程和科学优化问题的能力。

3.2.1　基本原理

粒子群优化算法最初设计为使用无实际质量的粒子来模拟鸟群，其算法策略模拟了鸟群的觅食过程。每个粒子具有速度和位置两个属性，速度决定了移动的快慢，而位置指示了移动的方向。每个粒子能够在搜索空间中独立地寻找优化问题的可行解，并将其找到的最优解记作个体极值。接着，该粒子通过群体通信和协作将个体极值传递给所有粒子，从而更新整个粒子群的全局最优解。重复以上过程直至满足终止条件，从而得到最优解。粒子群算法的原理展示在算法 3.1 中。

算法 3.1　PSO 算法。

输入：种群规模，初速度，权重，最大进化代数 T。
输出：最优解 gBest。

```
for each particle i
    初始化速度和位置
    评估适应度并设置 pBest_i = X_i
end for
gBest = min (pBest_i)
while t < T
    for i = 1 to N
        更新粒子的速度和位置
        评估粒子的适应度
        if f(X_i) < f(pBest_i)
            pBest_i = X_i
        if f(pBest_i) < f(gBest)
            gBest = pBest_i
    end for
end while
gBest 即为种群搜索的最优解
```

在粒子群算法中，速度和位置的更新是关键操作，这些操作决定了最优解的质量和收敛速度。粒子更新的原理如图 3.3 所示。

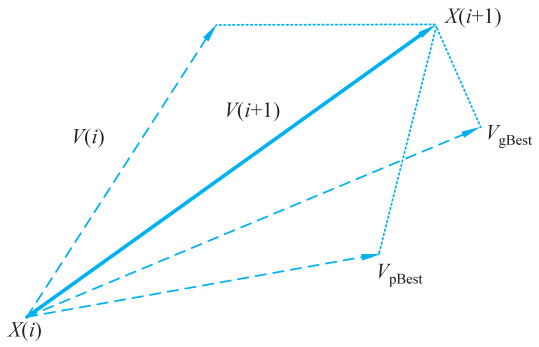


图 3.3 粒子更新的原理

在搜索过程中，粒子同时考虑个体最优解和全局最优解以更新其方向。假设目标搜索空间为 D 维的，种群大小为 N，这里，个体 $\boldsymbol{X}_i$ 表示为一个 D 维向量，具体如下。

$$\boldsymbol{X}_i=(x_{i1},x_{i2},\cdots,x_{iD}),\quad i=1,2,\cdots,N \tag{3.1}$$

同理，个体 $\boldsymbol{X}_i$ 的搜索速度也是一个 D 维的向量，具体如下。

$$\boldsymbol{V}_i=(v_{i1},v_{i2},\cdots,v_{iD}),\quad i=1,2,\cdots,N \tag{3.2}$$

个体 $\boldsymbol{X}_i$ 的历史最优位置，即个体最优解，如下所示。

$$\text{pBest}=(p_{i1},p_{i2},\cdots,p_{iD}),\quad i=1,2,\cdots,N \tag{3.3}$$

整个种群的历史最优位置，即全局最优解，如下所示。

$$\text{gBest}=(g_{i1},g_{i2},\cdots,g_{iD}),\quad i=1,2,\cdots,N \tag{3.4}$$

每个个体根据它的最优解和全局最优解确定其下一搜索位置，其速度更新方式如下。

$$v_i=v_i+c_1\times\text{rand}_1\times(\text{pBest}_i-x_i)+c_2\times\text{rand}_2\times(\text{gBest}_i-x_i) \tag{3.5}$$

其中，c_1 和 c_2 是加速常数。c_1 为粒子为个体学习因子，代表个体最优解的影响；c_2 为社会学习因子，代表全局最优解的影响。rand_1 和 rand_2 是区间[0,1]上的随机数。

搜索更新公式主要包含三部分：第一部分为惯性部分，反映个体的运动习惯，表明个体倾向于保持当前速度；第二部分为自我认知部分，涉及个体对其历史经验的学习，表明个体倾向于接近其历史个体最优解；第三部分为社会认知，涉及个体对种群中所有个体历史经验的学习，表明个体倾向于接近历史全局最优解。在确定新的搜索速度后，随即更新粒子的位置。粒子的下一位置取决于其当前位置和速度，具体公式如下。

$$x_i=x_i+v_i \tag{3.6}$$

3.2.2 算法流程

粒子群优化算法的详细流程如下。

(1) 初始化：随机设定种群中每个个体的初始位置和速度。

(2) 计算适应度值：基于适应度函数计算每个粒子在其当前位置的适应度。

(3) 求个体最优解：比较每个粒子当前位置的适应度与其历史最优适应度，并更新个体最优解。

(4) 求全局最优解：比较每个粒子当前位置的适应度与全局最优适应度，并更新全局

最优解。

(5) 更新粒子位置和速度：使用相应公式更新每个粒子的速度和位置。

(6) 判断是否终止搜索：如果未满足终止条件，返回第(2)步继续搜索；若满足，则算法结束，全局最优解为所求结果。

粒子群优化算法的流程图如图 3.4 所示。

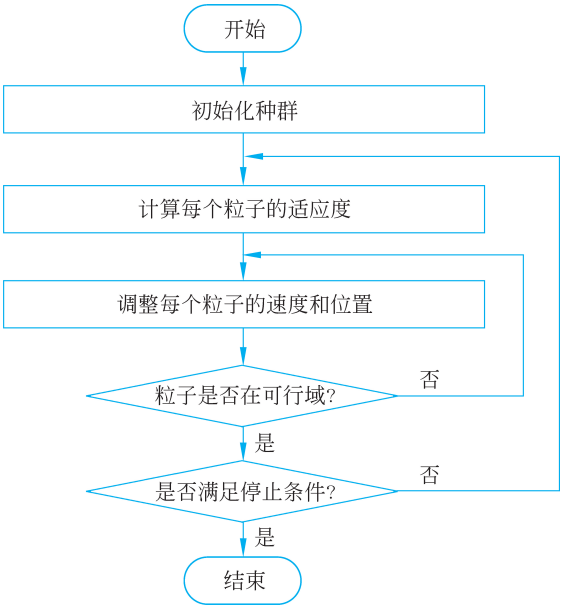


图 3.4　粒子群优化算法的流程图

3.2.3　连续目标粒子群优化算法

1. 局部和全局粒子群优化算法

在标准的粒子群优化(PSO)算法中，粒子根据其速度确定下一个位置，并同时考虑个体最优解与全局最优解来确定下一搜索点。个体速度的更新方式如下。

$$v_i = v_i + c_1 \times \text{rand}_1 \times (\text{pBest}_i - x_i) + c_2 \times \text{rand}_2 \times (\text{gBest}_i - x_i) \tag{3.7}$$

当 $c_1 = 0$ 时，个体速度的更新方式变为

$$v_i = v_i + c_2 \times \text{rand}_2 \times (\text{gBest}_i - x_i) \tag{3.8}$$

此算法形式称为全局粒子群优化算法，此时个体仅保留社会认知能力，缺乏局部搜索能力但具有较强的全局搜索能力。当 $c_2 = 0$ 时，个体速度的更新方式变为

$$v_i = v_i + c_1 \times \text{rand}_1 \times (\text{pBest}_i - x_i) \tag{3.9}$$

上述为局部粒子群优化算法，个体仅依赖自我认知，缺乏全局搜索能力，这限制了粒子的搜索范围并通常导致较低的效率和收敛困难。

2. 惯性权重的粒子群优化算法

探索定义为粒子偏离其既定寻优路径以开辟新路径，旨在寻找更优的可行解，显示算法的全局搜索能力。开发则是沿已定寻优路径继续搜索更优解，表现算法的局部搜索能力。因此，粒子群的探索与开发能力的平衡直接影响种群搜索最优解的效率及其收敛性。1998

年，史玉回教授提出了一种带有惯性权重的改进粒子群优化(PSO-W)算法。

PSO-W 通过为粒子的位置更新引入惯性权重，改进了种群的搜索策略。改进的 PSO-W 中，粒子速度的更新过程如下。

$$v_i = \omega v_i + c_1 \times \text{rand}_1 \times (\text{pBest}_i - x_i) + c_2 \times \text{rand}_2 \times (\text{gBest}_i - x_i) \tag{3.10}$$

其中，ω 是惯性权重，决定了粒子保持其搜索速度的程度。ω 值较大时，种群具有较强的全局搜索能力和较弱的局部搜索能力；相反，ω 值较小时，局部搜索能力强而全局搜索能力弱。当 ω 等于 1 时，算法即为标准 PSO，从而 PSO-W 可视为标准 PSO 的一个扩展。研究表明，当 ω 在 0.8～1.2 时，算法不仅效率更高，而且能有效收敛至最优解；ω 小于 0.8 时，算法搜索效率较低；ω 大于 1.2 时，容易陷入局部最优。

有研究表明，较大的惯性权重增强算法的全局搜索能力，而较小的惯性权重则增强局部搜索能力。因此，为平衡全局和局部搜索能力，引入了随搜索过程变化的惯性权重。ω 的变化随算法迭代次数而定，具体公式见式(3.11)。

$$\omega = \omega_{\max} - \frac{t(\omega_{\max} - \omega_{\min})}{t_{\max}} \tag{3.11}$$

其中，t 表示当前种群的迭代次数，$\omega_{\max}$ 和 $\omega_{\min}$ 分别为惯性权重的最大值和最小值，$t_{\max}$ 是种群的最大迭代次数。

粒子群优化算法中引入了随种群迭代进化而线性变化的惯性权重。算法初期，较大的惯性权重增强全局搜索能力，有利于维持解的多样性；而在后期，较小的惯性权重增强局部搜索能力，有助于收敛至最优解。

3. 自适应权重的粒子群优化算法

为平衡粒子群优化算法的全局与局部搜索能力，同时增强算法的全局搜索效果和收敛性，引入了非线性的动态惯性权重系数。算法中引入了自适应惯性权重，这些权重与种群及个体的目标函数值相关，以便根据种群和个体对最优解的搜索情况自适应地调整惯性权重，优化搜索过程。因此，此算法被称为自适应权重粒子群优化算法。

在自适应权重粒子群优化算法中，ω 的调整公式随当前种群的搜索状况而变，如式(3.12)所示。

$$\omega = \begin{cases} \omega_{\min} - \dfrac{(\omega_{\max} - \omega_{\min}) \times (f - f_{\min})}{(f_{\text{avg}} - f_{\min})}, & f \leqslant f_{\text{avg}} \\ \omega_{\max}, & f > f_{\text{avg}} \end{cases} \tag{3.12}$$

其中，$\omega_{\max}$ 和 $\omega_{\min}$ 分别是最大和最小惯性权重，f 是当前粒子的目标函数值，而 $f_{\min}$ 和 f_{avg} 分别代表种群的最小和平均目标函数值。

3.2.4 离散目标粒子群优化算法

多数优化问题都设定在某个空间内，其中，变量间及其级别之间存在离散和定性的差异。任何离散或连续问题均可采用二进制形式表示，因此，基于二进制函数的优化器具有明显优势。1997 年，Kennedy 和 Eberhart[10] 首次提出了针对 0-1 规划问题的二进制粒子群优化(Binary PSO，BPSO)算法。在 BPSO 中，位置和速度向量完全由 0 和 1 构成，坐标值的变化根据 0 或 1 的概率进行，以实现路径搜索。

BPSO 与标准 PSO 在算法流程上大致相同。首先根据全局最优解和个体最优解确定

粒子的速度，然后基于该速度更新粒子的位置。在 BPSO 中，粒子的速度更新规则与标准 PSO 相同。个体 X_i 的速度更新公式为

$$v_i = v_i + c_1 \times \text{rand}_1 \times (\text{pBest}_i - x_i) + c_2 \times \text{rand}_2 \times (\text{gBest}_i - x_i) \tag{3.13}$$

在 BPSO 中，每个个体的编码仅能为 0 或 1，故在更新位置前需通过 sigmoid 函数将速度转换为 0～1 的浮点数，转换公式为

$$s(v_i) = \frac{1}{1 + \exp(-v_i)} \tag{3.14}$$

更新位置时，生成一个随机数，若此数小于先前计算的概率，则更新粒子的编码为 1，否则为 0。粒子位置的更新规则如下。

$$\begin{cases} x_i = 1, & \text{rand}[0,1] < s(v_{id}) \\ x_i = 0, & \text{其他} \end{cases} \tag{3.15}$$

随着粒子群迭代进行，个体编码位置逐渐收敛至 1 或 0，即获得最优解。

3.3　蚁群算法

3.3.1　概述

蚁群优化(Ant Colony Optimization，ACO)算法(简称蚁群算法)是一种概率型生物启发式搜索算法，由 Marco Dorigo 等[11]在 1991 年提出，具有信息正反馈、协同性、分布式并行计算、鲁棒性和自组织等特点。蚂蚁倾向于选择信息素浓度较高的路径，这增加了路径上的信息素浓度，吸引更多蚂蚁，形成正反馈循环。单个蚂蚁的搜索行为是独立和随机的，信息交流有限，但蚁群通过信息素能快速有序地找到食物源，体现了协同性和分布式并行计算。旧的信息素会逐渐挥发，因此，蚁群能够在遇到障碍物时快速适应并找到新的最优解，显示出算法的鲁棒性。蚁群自主地搜索最短路径并最终收敛至最优解，整个过程无须人工干预，展现了自组织特性。

因上述特性，并且容易与其他算法结合，蚁群算法已受到国内外研究者的广泛关注，并成功应用于多种组合优化问题。例如，旅行商问题(TSP)、非对称旅行商问题(ATSP)、图像检测与分割问题、卫星资源调度问题、作业车间调度问题(JSP)等。

3.3.2　基本原理

蚁群算法的思想源于蚁群根据前代蚂蚁在路径上留下的信息素寻找食物源的最短路径的行为。具体来说，所有路径集合构成优化问题的解集，蚂蚁的行进路径视为可行解，而蚂蚁分泌的信息素浓度则作为寻找最优解的依据。进一步说明，算法迭代包括两个关键步骤："选择路径"与"更新信息素浓度"，分别对应搜索可行解和更新最优解的依据，这两步共同影响算法的收敛速度、随机搜索的效果以及路径选择的质量。下面以旅行商问题为例详细描述蚁群算法的这两个关键步骤。

1. 选择路径

在初始阶段，所有节点间无信息素，初代蚂蚁随机放置于各个节点，随机选择其行进路径。接下来，每只蚂蚁根据轮盘赌法选择其下一条行进路径。

假设信息素浓度控制因子和路径长度启发函数控制因子分别为常数 α 和常数 β，则第 k 只蚂蚁在时刻 t 选择从节点 i 到节点 j 路径的概率 $p_{ij}^{k}(t)$ 表示为

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\times\left[\dfrac{1}{d_{ij}}\right]^{\beta}}{\displaystyle\sum_{s\in \text{allowed}_k}[\tau_{is}(t)]^{\alpha}\times\left[\dfrac{1}{d_{is}}\right]^{\beta}}, & j\in \text{allowed}_k\\ 0, & j\notin \text{allowed}_k\end{cases} \tag{3.16}$$

其中，$\tau_{ij}(t)$ 代表时刻 t 节点 i 到节点 j 路径上的信息素浓度，d_{ij} 代表节点 i 到节点 j 的路径长度，allowed_k 表示第 k 只蚂蚁在时刻 t 之前尚未经过的节点集合。

2. 更新信息素浓度

蚂蚁在路径上释放信息素导致信息素浓度增加；然而，信息素也会随时间逐渐挥发。信息素浓度不仅影响蚂蚁在下一迭代中的路径选择，也间接决定了算法的最终效果。因此，合理更新信息素浓度至关重要。假设每代蚂蚁数量为 m，信息素挥发因子为 $\rho\in(0,1)$，$t+1$ 时刻节点 i 到节点 j 路径上的信息素浓度 $\tau_{ij}(t+1)$ 是 $t+1$ 时刻未挥发的信息素浓度与 t 时刻到 $t+1$ 时刻新增的信息素浓度之和，计算公式为

$$\tau_{ij}(t+1)=\tau_{ij}(t)\times(1-\rho)+\sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{3.17}$$

其中，$\Delta\tau_{ij}^{k}$ 表示第 k 只蚂蚁在节点 i 到节点 j 的路径上所留下的信息素浓度增量。

3.3.3 算法流程

蚁群算法的主要算法流程包括以下 4 步。

(1) 初始化参数包括路径集合、所有常数因子、最大迭代次数及每代蚂蚁数量；初始化时，设定当前时刻为 0，路径信息素浓度为 0。

(2) 选择路径。同一代的蚂蚁同时选择其行进路径。

(3) 持续进行更新，蚁量算法在蚂蚁到达新节点后进行更新，蚁周算法在全部蚂蚁找到食物源后更新。

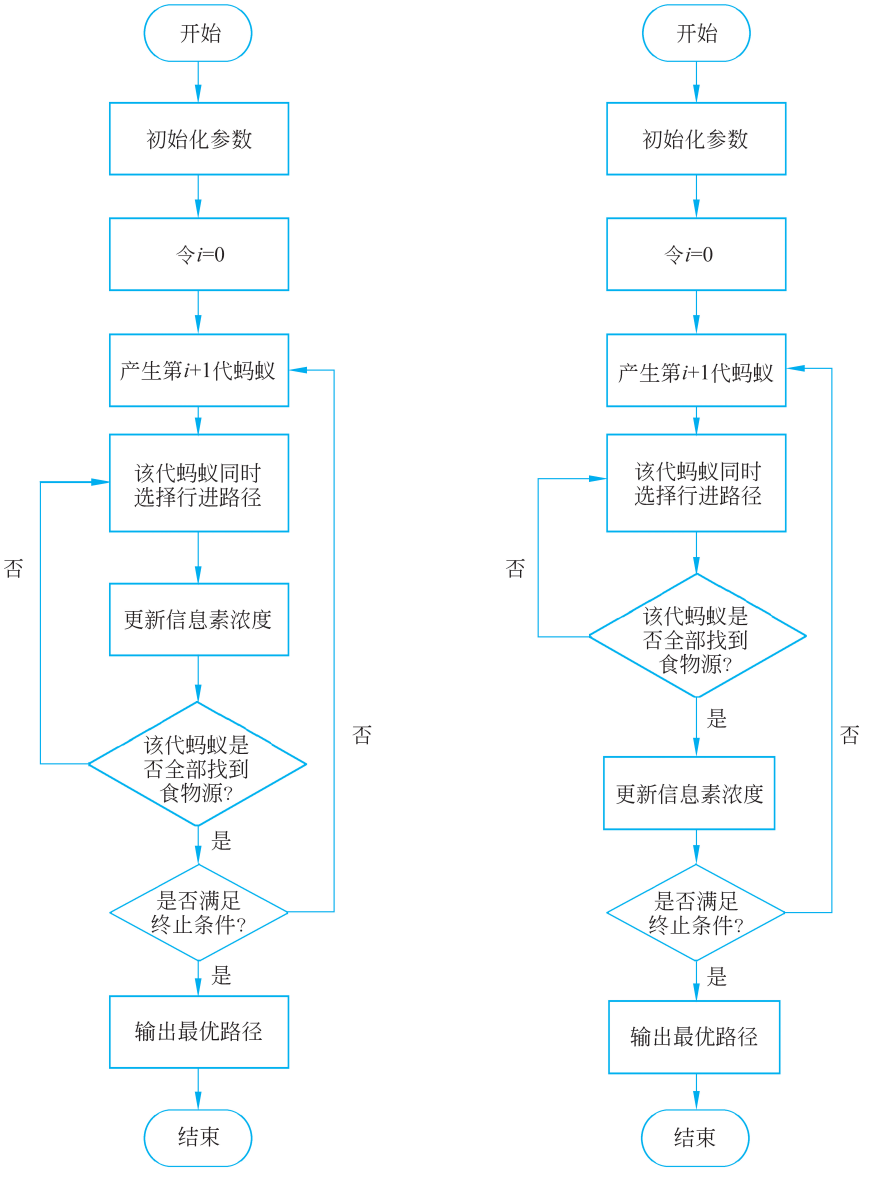
(4) 如果达到最大迭代次数或所有蚂蚁的路径相同，则输出最优路径；否则，生成新一代蚂蚁，并重复执行上述步骤。

根据 ACO 算法模型的差异，存在两种算法流程，如图 3.5 所示。图 3.5(a)为蚁密算法和蚁量算法的流程，图 3.5(b)为蚁周算法的流程。

3.3.4 改进蚁群算法

研究人员为解决蚁群算法的不足，如长运行时间、较差的收敛性、对初始参数的依赖及可能的局部最优问题，已开发多种改进算法，包括精英蚂蚁系统、人工蚁群系统、基于排序的蚂蚁系统、最大-最小蚂蚁系统。这些改进算法通过对蚂蚁的路径选择和信息素浓度更新机制进行改进，显著提高了蚁群算法的性能，并促进了 ACO 的进一步发展。

在精英蚂蚁系统中，每个循环结束后，通过精英策略对已发现的最优路径进行额外的信息素增强，从而增加该路径在后续循环中被选择的概率。在人工蚁群系统中，仅有精英蚂蚁被允许释放信息素。信息素浓度的更新规则如下：全局信息素浓度在每代蚂蚁完成食物源



(a) 蚁密算法和蚁量算法　　(b) 蚁周算法

图 3.5　ACO 算法的流程图

搜索后更新；局部信息素浓度则在每只蚂蚁移动至新节点后更新。此外，采用伪随机比例(Pseudo-Random-Proportional)规则确定蚂蚁行进路径。在基于排序的蚂蚁系统中，每一代的蚂蚁在找到食物源后依据其行进路径总长度排序，路径越短，蚂蚁的排名越高。蚂蚁的排名显著影响信息素的更新：仅排名较高的蚂蚁和精英蚂蚁有权释放信息素；而且，信息素浓度随排名升高而递减。

最大-最小蚂蚁系统(MMAS)基于 ACS 改进，其区别主要在于：限定所有路径的信息素浓度在一定范围内以防算法过早收敛，所有路径上的信息素浓度都被限定在区间$[\tau_{\min},\tau_{\max}]$上；初始信息素浓度为区间$[\tau_{\min},\tau_{\max}]$上的最大浓度值$\tau_{\max}$，以更好地探索新解；精英

蚂蚁既包括全局最优蚂蚁，也包括迭代最优蚂蚁。

3.3.5 混合蚁群算法

1. 与遗传算法结合

蚁群算法与遗传算法的结合策略主要分为两种：基于遗传算法的蚁群算法与基于蚁群算法的遗传算法。基于遗传算法的蚁群算法主要应用 ACO。这种方法通常利用遗传算法来优化由历史经验决定的蚁群算法中的各种控制参数，如信息素浓度控制因子 α、路径长度启发函数控制因子 β、信息素挥发因子 ρ，以及初始信息素浓度值。而基于 ACO 的遗传算法以 GA 为核心，通常在遗传算法的编码、选择、交叉及变异步骤前应用蚁群算法生成 GA 的初始种群。这两种结合策略实现了 ACO 和 GA 算法优势的互补，从而提升了算法的整体性能。具体分析如下：ACO 的信息正反馈机制弥补了 GA 在反馈信息不足、随机性高和迭代过程盲目性方面的劣势；GA 的交叉和变异算子赋予其强大的全局搜索能力，补偿了 ACO 易于陷入局部最优解的缺陷。同时，ACO 的强局部搜索能力也弥补了 GA 在维持局部最优解方面的不足；ACO 的高求解精度弥补了 GA 在精确解求解效率方面的不足；GA 的快速求解能力补偿了 ACO 较长的运行时间和较差的收敛性；利用 GA 优化 ACO 的初始参数和信息素浓度值，弥补了 ACO 性能依赖于经验设定初始参数的不足。

2. 量子蚁群算法

量子蚁群算法(Quantum Ant Colony Algorithm，QACA)结合了量子启发式进化算法(Quantum-inspired Evolutionary Algorithm，QEA)和蚁群算法的优点，由 Liu 等在 2016 年提出。QACA 通过引入量子保真度概念并使用由多个单量子比特位组成的“多量子比特”来表示路径上的信息素浓度，同时采用“量子旋转门”来更新信息素。该方法不仅增强了蚁群算法的鲁棒性和收敛能力，还有效避免了算法停滞。

1) 选择路径

在量子蚁群算法中，除了考虑信息素浓度和路径长度，每只蚂蚁还需考虑系统的量子保真度，即与初始量子态的偏离程度。第 k 只蚂蚁在 t 时刻选择节点 i 到节点 j 路径的概率 $p_{ij}^{k}(t)$ 为

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\times\left[\dfrac{1}{d_{ij}}\right]^{\beta}\times[\varepsilon_{ij}(t)]^{\chi}}{\sum\limits_{s\in \text{allowed}_k}[\tau_{is}(t)]^{\alpha}\times\left[\dfrac{1}{d_{is}}\right]^{\beta}\times[\varepsilon_{is}(t)]^{\chi}}, & j\in \text{allowed}_k\\ 0, & j\notin \text{allowed}_k\end{cases}\tag{3.18}$$

其中，$\varepsilon_{ij}(t)$ 表示 t 时刻节点 i 和节点 j 之间的量子状态保真度，$\varepsilon_{ij}(t)=0$ 表示完全失真；$\varepsilon_{ij}(t)=1$ 表示目前的量子态与初始量子态相比没有任何偏差，一般情况下，$\varepsilon_{ij}(t)\in(0,1)$。$\chi$ 是常数，表示量子保真度控制因子。

2) 更新信息素浓度

路径上信息素浓度值由“多量子比特”表示，并通过“量子旋转门”更新。设信息素浓度存储的量子比特数为 n，第 k 只蚂蚁 t 时刻在路径 (i,j) 上释放的信息素浓度的量子比特为 $\tau_{ij}^{k}(t)=\begin{pmatrix}\cos\phi_{k1}(t)\\ \sin\phi_{k1}(t)\end{pmatrix}|\cdots|\begin{pmatrix}\cos\phi_{ko}(t)\\ \sin\phi_{ko}(t)\end{pmatrix}|\cdots|\begin{pmatrix}\cos\phi_{kn}(t)\\ \sin\phi_{kn}(t)\end{pmatrix}$，则量子比特的第 o 位信息素浓度的更新

公式表示如下。

$$\begin{bmatrix}\cos\phi_{ko}(t+1)\\ \sin\phi_{ko}(t+1)\end{bmatrix}=\begin{bmatrix}\cos\theta_{ko} & -\sin\theta_{ko}\\ \sin\theta_{ko} & \cos\theta_{ko}\end{bmatrix}\begin{bmatrix}\cos\phi_{ko}(t)\\ \sin\phi_{ko}(t)\end{bmatrix} \tag{3.19}$$

其中，$\begin{bmatrix}\cos\phi_{ko}(t)\\ \sin\phi_{ko}(t)\end{bmatrix}$表示在 t 时刻，第 k 只蚂蚁的第 o 位信息素在量子旋转门处理前量子比特的概率幅，$\begin{bmatrix}\cos\phi_{ko}(t+1)\\ \sin\phi_{ko}(t+1)\end{bmatrix}$表示在 $t+1$ 时刻，第 k 只蚂蚁的第 o 位信息素在量子旋转门处理后量子比特的概率幅，$\begin{bmatrix}\cos\theta_{ko} & -\sin\theta_{ko}\\ \sin\theta_{ko} & \cos\theta_{ko}\end{bmatrix}$表示量子旋转门，$\theta_{ko}$ 为旋转角度，其计算公式表示如下。

$$\theta_{ko}=\Delta\theta_{ko}\times d(\cos\phi_{ko}(t),\sin\phi_{ko}(t)),\quad k=1,2,\cdots,N \tag{3.20}$$

其中，$\Delta\theta_{ko}$ 表示与算法循环次数相关的变量，$d(\cos\phi_{ko}(t),\sin\phi_{ko}(t))$为控制旋转方向的函数。当 $d(\cos\phi_{ko}(t),\sin\phi_{ko}(t))<0$，量子旋转门将顺时针旋转，否则将逆时针旋转。

3.4　蜂群算法

3.4.1　概述

2005 年，Karaboga 受到蜂群采蜜行为的启发，提出了用于解决高维连续优化问题的人工蜂群(Artificial Bee Colony，ABC)算法。

为寻找全局最优解，ABC 算法仿真蜜蜂寻找最优蜜源的过程，其中，蜜蜂按各自分工合作，分为蜜源、雇佣蜂和非雇佣蜂三部分，使用雇佣蜂的圆圈舞或摇摆舞以进行种群间的信息交流。在三部分构成元素中，蜜源代表问题的可行解；蜜源的质量越高，其适应度值也越大。雇佣蜂在局部范围内寻找并记录可能的最优蜜源，展示了算法在精确搜索方面的能力。非雇佣蜂包括侦察蜂和跟随蜂。侦察蜂在全局范围内随机寻找食物源，旨在超越局部最优解，展示算法的全局探索能力；跟随蜂则在其跟随的雇佣蜂发现的食物源附近寻找蜜源，从而加速算法的收敛。在这三类构成元素中，每个雇佣蜂与一个蜜源直接对应。非雇佣蜂中的跟随蜂仅可跟随单一雇佣蜂，而每个雇佣蜂同样只有一个跟随蜂；侦察蜂大约构成蜂群总数的 5%～10%。

在人工蜂群算法中，蜜蜂的采蜜行为分为招募蜜蜂到蜜源和抛弃蜜源两种，体现了算法的正反馈和负反馈。雇佣蜂发现的蜜源质量越高，其在蜂巢中执行摇摆舞的时间越长，从而增加被跟随蜂跟随的概率，形成正反馈机制。若雇佣蜂在多次局部搜索后仍未发现更优蜜源，劣质蜜源将被抛弃，体现了人工蜂群算法的负反馈特性。

人工蜂群算法具有正负反馈特性，以及自组织、易收敛、强鲁棒性、低复杂度的优点。该算法已被成功应用于解决包括非线性函数优化、资源调度、路径规划、神经网络训练、图像检测与分割等多种实际问题。

3.4.2　算法流程

初始化阶段、雇佣蜂阶段、跟随蜂阶段和侦察蜂阶段构成了 ABC 算法的 4 个主要流程。

初始化蜜源位置和蜂群后，通过雇佣蜂、跟随蜂和侦察蜂三个阶段的不断进化迭代，更新最优蜜源信息，直至达到最大迭代次数或满足终止条件的最优解。

1. 初始化阶段

此阶段涵盖了蜜源位置和蜂群角色的初始化。蜜源位置初始化是首要步骤。本阶段包括蜜源位置和蜂群角色的初始化。首先进行蜜源位置的初始化。假设 $y_j^{\min}$ 和 $y_j^{\max}$ 分别表示维度 j 的下界和上界。蜜源 i 在第 j 维度的初始位置 y_{ij} 由以下式子随机确定。

$$y_{ij} = y_j^{\min} + \text{rand}[0,1](y_j^{\max} - y_j^{\min}) \tag{3.21}$$

其中，$i \in \{1,2,\cdots,\text{SN}\}$，SN 表示蜜源总数目；$j \in \{1,2,\cdots,J\}$，$J$ 表示所求问题的维度。

接着进行蜂群角色初始化。起初，所有蜜蜂均被视为侦察蜂，并在搜索空间中随机寻找高质量蜜源。依据发现的蜜源质量，对蜜蜂进行排序并分配角色。质量最高的前几个侦察蜂成为雇佣蜂，次高质量的侦察蜂转变为跟随蜂，其余则保持侦察蜂身份。

2. 雇佣蜂阶段

雇佣蜂在第 j 维度的原蜜源 i 周围寻找新蜜源 v_{ij}，如式(3.22)所示。然后，雇佣蜂采用贪婪策略对原蜜源和新蜜源进行选择。当发现适应度更高的新蜜源时，雇佣蜂返回蜂巢并通过摇摆舞与等待中的跟随蜂分享新蜜源信息，如蜜源的位置和质量。蜜源适应度值越高，雇佣蜂的摇摆舞时间越长。

$$v_{ij} = y_{ij} + \mu(y_{ij} - y_{kj}) \tag{3.22}$$

其中，μ 是区间$[-1,1]$上均匀分布的随机数，y_{kj} 表示在第 j 维度随机选择邻居蜜源。

3. 跟随蜂阶段

雇佣蜂的舞蹈结束后，跟随蜂通过轮盘赌法选择一只雇佣蜂并在其蜜源附近搜索食物源。如果跟随蜂找到的蜜源适应度超过了所跟蜂发现的蜜源适应度，则此跟随蜂转变为雇佣蜂。设 ρ_i 表示第 i 个蜜源，M 表示蜜源总数，$F(\rho_i)$表示第 i 个蜜源的适应度，根据轮盘赌法，位于 ρ_i 的蜜源被跟随蜂选中的概率 P_i 为

$$P_i = \frac{F(\rho_i)}{\sum_{k=1}^{M} F(\rho_k)} \tag{3.23}$$

$F(\rho_i)$计算方式如下。

$$F(\rho_i) = \begin{cases} \dfrac{1}{1+f(\rho_i)}, & f(\rho_i) > 0 \\ 1 + \text{abs}(f(\rho_i)), & f(\rho_i) \leqslant 0 \end{cases} \tag{3.24}$$

其中，$f(\rho_i)$是第 i 个蜜源所对应最优化问题的目标函数值。

4. 侦察蜂阶段

在该阶段，侦察蜂在全局范围内随机搜索新蜜源 v_{ij}：

$$v_{ij} = y_j^{\min} + \text{rand}[0,1](y_j^{\max} - y_j^{\min}) \tag{3.25}$$

如果侦察蜂发现的蜜源适应度值超过雇佣蜂发现的蜜源适应度值，则此侦察蜂帮助克服雇佣蜂局部最优解的局限。同时，该侦察蜂将转变为雇佣蜂，而那些放弃劣质蜜源的雇佣蜂则转变为侦察蜂。ABC 算法流程图如图 3.6 所示。

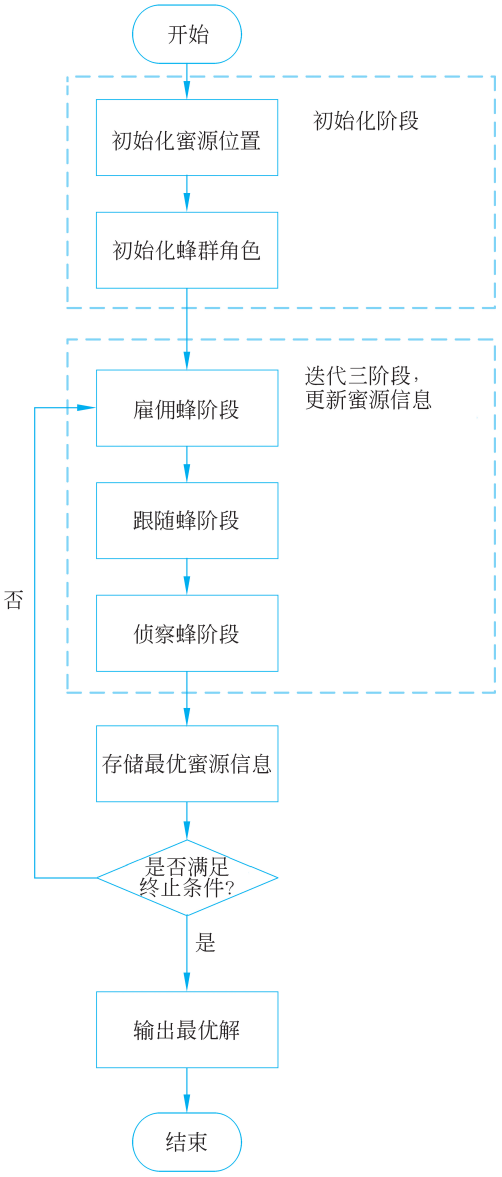


图 3.6　ABC 算法流程图

3.4.3　并行人工蜂群算法

并行人工蜂群(Parallel Artificial Bee Colony,PABC)算法的概念最初是由 Harikrishna Narasimhan 提出的。该算法通过结合一个共享内存和多个处理器上的本地内存(每个服务器配备一块本地内存)来实现算法的并行性。共享内存中的蜜源信息可以被所有蜜蜂并行访问或单独写入;而本地内存中的蜜源信息仅供相应处理器上的蜜蜂访问或单独写入。与顺序人工蜂群算法相比,PABC 算法能在多个处理器上同时运行,克服了不同处理器间蜜蜂的依赖性,从而显著缩短了算法的运行时间并提高了性能。

在初始化阶段,首先随机生成蜜源并存入共享内存,然后将其均匀复制到各个本地内存

中;同时初始化蜂群角色,并将三种类型的蜂群根据角色均匀分配到各个处理器。接着,各处理器将依次执行雇佣蜂阶段、跟随蜂阶段(计算蜜源选中概率、进行轮盘赌选择)、侦察蜂阶段,以及存储本地最优蜜源信息的步骤。

同时在每次迭代过程中将改进后的蜜源(解)保存到本地内存,在每次迭代结束后将本地内存的蜜源(解)复制到共享内存中。最后,满足终止条件时,退出所有处理器内部的循环,从共享内存中输出全局最优解。在算法运行过程中,鉴于不同处理器之间无依赖关系,故并行人工蜂群算法允许各处理器支持蜜蜂的异步行为,从而能够实现并行执行。

假定共享内存中存储的蜜源总数为 SN,并且并行的处理器总数为 Par,则每个处理器中存储的蜜源数(可行解数)为有$\frac{\mathrm{SN}}{\mathrm{Par}}$个。考虑到每个蜜源与一个雇佣蜂直接对应,因此每个处理器上的雇佣蜂数量也为$\frac{\mathrm{SN}}{\mathrm{Par}}$。下面以单个处理器的一个循环过程为例,详细阐述该改进算法的雇佣蜂阶段、跟随蜂阶段和侦察蜂阶段。

1. 雇佣蜂阶段

为了改进本地内存中蜜源的质量,每个处理器上的雇佣蜂从全局共享内存中随机选择一个与本地内存不同的邻居蜜源,然后将旧蜜源与新蜜源之间适应度值更大的蜜源存入本地内存。位于处理器 l 的雇佣蜂在原蜜源 y_{ij}^{l} 附近寻找新蜜源 v_{ij}^{l} 的计算公式如下。

$$v_{ij}^{l}=y_{ij}^{l}+\mu_{ij}^{l}(y_{ij}^{l}-y_{kj}^{l\text{-ALL}}),\quad k\in\{1,2,\cdots,i-1,i+1,\cdots,\mathrm{SN}-1,\mathrm{SN}\} \tag{3.26}$$

其中,$y_{kj}^{l\text{-ALL}}$ 表示位于处理器 l 的雇佣蜂从全局共享内存随机选择的与本地内存不同的邻居蜜源。μ_{ij}^{l} 为区间[-1,1]上均匀分布的随机数。

2. 跟随蜂阶段

为了提高本地处理器中存储的蜜源质量,本地跟随蜂使用轮盘赌法决定后,在选定的雇佣蜂附近搜索蜜源。跟随蜂的整个搜索过程均在本地处理器上执行。若跟随蜂发现的蜜源的适应度值超过雇佣蜂发现的蜜源,便更新本地内存中的蜜源信息。位于 ρ_i 的蜜源被本地处理器 l 上的跟随蜂选中的概率 P_i^l 如下。

$$P_i^l=\frac{F^l(\rho_i)}{\sum_{k=1}^{\mathrm{SN_{Par}}}F^l(\rho_k)} \tag{3.27}$$

其中,$\mathrm{SN_{Par}}$ 表示处理器 l 上的蜜源总数。$F^l(\rho_i)$ 表示处理器 l 上第 i 个蜜源的花蜜量大小。

3. 侦察蜂阶段

在本地处理器中,侦察蜂随机搜索新蜜源,其数量与本地雇佣蜂及跟随蜂放弃的蜜源总数相等。

3.4.4 混合人工蜂群算法

1. 与其他群智能算法结合

人工蜂群算法与其他群智能算法的融合包括人工蜂群-遗传混合算法、人工蜂群-蚁群混合算法、人工蜂群-差分进化算法等。人工蜂群算法与蚁群算法的结合,旨在弥补人工蜂群算法的收敛性不足及蚁群算法在后期可能的停滞现象。Gao 等[12]通过应用差分进化算法搜索新解,以防止人工蜂群算法陷入局部最优。此外,为充分利用布谷鸟算法的优势改进

人工蜂群算法，研究者提出了人工蜂群算法与布谷鸟算法的融合算法。

2. 与机器学习算法结合

人工蜂群算法与机器学习相结合能显著提高机器学习的分类精度并加速人工蜂群算法的收敛速度。主要形式包括人工蜂群算法与神经网络的融合及人工蜂群算法与聚类技术的结合。采用 ABC 算法优化 ANN 的权重和偏置，可以补偿 LM(Levenberg-Marquardt)算法在误差反向传播中的不足。为 ABC 设计 Map/Reduce 编程提高聚类的分类准确度，并减少聚类的执行时间。IABCOCT 是基于聚类的改进 ABC 算法。此算法在雇佣蜂阶段通过速度和位置评估适应度，生成可行解。在跟随蜂阶段，采用基于开发参数的 Grenade Explosion Method 以提高算法的收敛速度并避免陷入局部最优。在侦察蜂阶段，使用 Cauchy Operator 来增强全局搜索能力。实验证明，IABCOCT 显著增强了 ABC 的探索能力、开发能力及计算效率，并能显著降低能耗，因而适用于簇头的优化选择。此外，某些研究在 ABC 算法中引入了基于 Q 函数的强化学习及正交学习策略，有效提高了蜂群之间的信息交换效率。

3. 量子人工蜂群算法

量子人工蜂群(Quantum-inspired Artificial Bee Colony，QABC)算法，基于量子进化人工蜂群算法提出，结合了量子计算和人工蜂群算法的特性，使用“多量子比特”对每个蜜源进行编码，并通过观察测量的概率性推测原始量子态。使用“量子旋转门”更新蜜源搜索，并通过量子干涉引导蜂群朝全局最优解寻找蜜源。QABC 算法降低了算法的空间复杂度，增强了种群多样性，提升了算法的运行及收敛速度，并显著增强了全局搜索能力。下面详细描述量子人工蜂群算法的 4 个阶段。

1) 编码与初始化阶段

蜜源位置通过“多量子比特”进行编码。蜜源 i 的初始位置 y_i 通过以下公式编码。

$$y_i=\begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{iJ} \\ b_{i1} & b_{i2} & \cdots & b_{ij} & \cdots & b_{iJ} \end{pmatrix} \tag{3.28}$$

其中，J 表示问题的维度，第 j 列的二维向量 $\begin{pmatrix} a_{ij} \\ b_{ij} \end{pmatrix}$ 表示蜜源 i 在第 j 维度的初始位置 y_{ij}。对于所有的单量子比特(每一列的二维向量)均满足 $|a_{ij}|^2+|b_{ij}|^2=1, j\in[1,J]$。

此编码方式也可转换为量子角的形式：

$$y_i=\begin{pmatrix} \cos\phi_{i1} & \cos\phi_{i2} & \cdots & \cos\phi_{ij} & \cdots & \cos\phi_{iJ} \\ \sin\phi_{i1} & \sin\phi_{i2} & \cdots & \sin\phi_{ij} & \cdots & \sin\phi_{iJ} \end{pmatrix} \tag{3.29}$$

其中，$\phi_{ij}=\arctan\dfrac{b_{ij}}{a_{ij}}$。该式始终满足 $|\cos\phi_{ij}|^2+|\sin\phi_{ij}|^2=1, j\in[1,J]$。

此外，初始化包括设置量子蜂群、问题维度、侦察蜂数量和算法最大迭代次数。

2) 雇佣蜂阶段

雇佣蜂阶段包括 5 个步骤：观察测量、适应度评价、贪婪选择、量子 ABC 算子和量子干涉。适应度评价和贪婪选择步骤与经典人工蜂群算法相同。本部分将重点介绍观察测量、量子 ABC 算子和量子干涉三个步骤。

通过观察，不可观测的量子系统坍塌成一个可观测的单一量子态(0 态或 1 态)。观察得到的二进制实值解用于推测原始量子态。图 3.7 以 5 个量子比特的观察测量为例，说明

此过程。对于第一位的单比特量子态 $|v\rangle=\begin{bmatrix}0.3215\\0.9469\end{bmatrix}=0.3215\,|0\rangle+0.9469\,|1\rangle$，有 $|0.3215|^2\approx0.1034$ 的概率观察到 0 态，同时有 $|0.9469|^2\approx0.8966$ 的概率观察到 1 态。同理，可概率性得到其他 4 个单量子比特的观察测量结果。

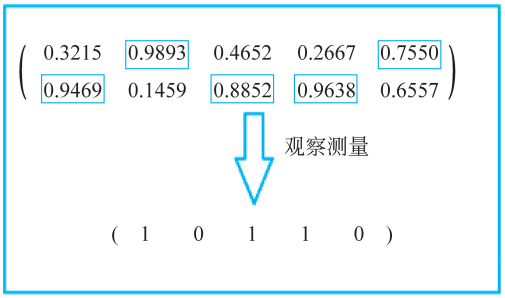


图 3.7 量子比特的一次观察测量

对雇佣蜂在原蜜源 y_{ij} 附近寻找新蜜源 v_{ij} 的公式进行改进，计算如下。

$$v_{ij}\begin{pmatrix}a_{ij}\\b_{ij}\end{pmatrix}=y_{ij}\begin{pmatrix}a_{ij}\\b_{ij}\end{pmatrix}+\mu\left(y_{ij}\begin{pmatrix}a_{ij}\\b_{ij}\end{pmatrix}-y_{kj}\begin{pmatrix}a_{ij}\\b_{ij}\end{pmatrix}\right)\tag{3.30}$$

其中，y_{kj} 表示在第 j 维度原蜜源 y_{ij}^{l} 附近随机选择的邻居蜜源。$\begin{pmatrix}a_{ij}\\b_{ij}\end{pmatrix}$ 表示该时刻对应的量子比特。

此时，原量子比特 $\begin{pmatrix}a_{ij}^{t}\\b_{ij}^{t}\end{pmatrix}$ 通过量子旋转门 $\begin{pmatrix}\cos\phi_{ij}^{t+1} & -\sin\phi_{ij}^{t+1}\\ \sin\phi_{ij}^{t+1} & \cos\phi_{ij}^{t+1}\end{pmatrix}$ 更新为 $\begin{pmatrix}a_{ij}^{t+1}\\b_{ij}^{t+1}\end{pmatrix}$。

$$\begin{pmatrix}a_{ij}^{t+1}\\b_{ij}^{t+1}\end{pmatrix}=\begin{pmatrix}\cos\phi_{ij}^{t+1} & -\sin\phi_{ij}^{t+1}\\ \sin\phi_{ij}^{t+1} & \cos\phi_{ij}^{t+1}\end{pmatrix}\begin{pmatrix}a_{ij}^{t}\\b_{ij}^{t}\end{pmatrix}\tag{3.31}$$

利用全局最优蜜源对原量子比特进行干涉，适当地调整原量子比特。此举提高了蜜蜂选择全局最优蜜源的概率。若原量子比特位对应的全局最优蜜源二进制位为 0，则干涉后 0 态的概率增加；若为 1，则 1 态的概率增加。对原量子比特 $\begin{pmatrix}a_A\\b_A\end{pmatrix}$ 进行干涉后的量子比特 $\begin{pmatrix}a_B\\b_B\end{pmatrix}$ 计算如下。

$$\begin{cases}a_B=\dfrac{\text{gl\_best}(i)+Q\times a_A}{Q+1}\\ b_B=\sqrt{1-a_B^2}\end{cases}\tag{3.32}$$

其中，gl_best(i) 是全局最优解，Q 是常数。

这里，跟随蜂执行与雇佣蜂相同的 5 个步骤，但只在其所跟随的雇佣蜂附近进行局部更新。为跳出局部最优解，在侦察蜂阶段引入再生算子。具体地，如果在达到限定的迭代步数(limit)后，侦察蜂仍未找到更优蜜源，则用随机生成的、满足归一化条件(即 $|a_{ij}|^2+|b_{ij}|^2=1, j\in[1,J]$)的新蜜源替换由该量子比特表示的蜜源。

4. 其他混合子人工蜂群算法

多种优化策略，如混沌系统、万有引力、智能学习等，经常被整合进人工蜂群算法中，旨

在增强蜂群的搜索能力和优化轮盘赌选择机制，从而提高算法的收敛速度，避免陷入局部最优，并增强算法的多样性。Gao[12]在 ABC 算法的初始化阶段采用混沌映射增加种群多样性，并利用基于对立的学习方法加速算法的收敛。研究表明，所提出的 IABC 算法显著增强了蜜蜂的搜索能力，提高了 ABC 算法的整体性能。Tsai 等提出了交互式人工蜂群优化算法。为最大化跟随蜂的探索能力，研究人员改进了 ABC 算法，使跟随蜂不仅限于跟随单一雇佣蜂，在多个雇佣蜂之间通过矢量形式的万有引力进行交互。

实验证明，该算法在增强探索能力的同时，也提高了解的准确度。为增强跟随蜂的采集能力，Xiang 等[13]在跟随蜂阶段同样引入了万有引力策略。Gao 等[14]提出了一种融合智能学习策略和湍流算子的人工蜂群算法 ILTD_ABC。该改进算法通过在雇佣蜂阶段和跟随蜂阶段引入差异化的智能学习策略来加速算法收敛。在雇佣蜂阶段，使用湍流分布算子替代传统人工蜂群算法中的随机数，以自动平衡全局与局部搜索，并定义了一种搜索定向机制以减轻雇佣蜂在寻找蜜源时的振荡现象。在跟随蜂阶段，放弃了收敛速度较慢的轮盘赌法，改为直接采用性能更优的解来更新跟随蜂的位置。实验证明，该算法能有效提升 ABC 的性能，并且适用于图像分割领域。

3.4.5　离散人工蜂群算法

为了解决聚类算法中的特征选择问题，Marinakis 等[15]在离散粒子群算法的启发下，于 2009 年提出了离散人工蜂群(Discrete Artificial Bee Colony，DABC)算法，这标志着人工蜂群算法应用领域的首次扩展至二进制离散空间。DABC 算法与 ABC 算法的主要区别在于蜜源位置的初始化过程。在随机产生蜜源 i 在第 j 维度的初始位置 y_{ij}(实数解)后，DABC 算法通过 sigmoid 函数将蜜源位置转换为二进制形式。

$$y_{ij}^{D}=\begin{cases}1, & \text{rand}[0,1]<\text{sigmoid}(y_{ij})\\ 0, & \text{rand}[0,1]\geqslant\text{sigmoid}(y_{ij})\end{cases} \tag{3.33}$$

其中，y_{ij}^{D}代表转换后在二进制离散空间中蜜源 i 在第 j 维度位置，其中，0 表示特征未被选择，1 表示特征被选择。rand[0,1]表示均匀分布在区间[0,1]的随机数。sigmoid(y_{ij})表示用于转换的 sigmoid 函数，依据连续域蜜源的初始位置计算如下。

$$\text{sigmoid}(y_{ij})=\frac{1}{1+\exp(-y_{ij})} \tag{3.34}$$

3.5　烟花算法

3.5.1　概述

烟花算法(Fireworks Algorithm，FWA)由北京大学谭营教授等在观察烟花爆炸瞬间火花四散的现象后于 2010 年提出，是一种用于解决复杂函数全局优化问题的基于进化的新型非生物群体智能优化算法。烟花算法有效结合了经典群智能算法如蚁群算法和人工蜂群算法的协同合作机制及遗传算法等经典进化算法的竞争机制，其中，烟花和火花的位置被视为问题的可行解。算法通过爆炸算子、高斯变异算子、映射规则及选择算子 4 个步骤迭代进化烟花种群。

爆炸算子是算法的核心步骤，以烟花为圆心，其爆炸幅度为半径，在该圆范围内产生固定数量的火花，体现了 FWA 的独特邻域搜索机制和爆炸特性，同时平衡了全局与局部搜索。高斯变异算子对随机选定的少量烟花执行高斯变异，影响随机维度上的变化，从而帮助种群跳出局部最优并增加多样性。映射规则确保将爆炸算子和高斯变异算子生成的非可行域火花映射回可行域，保证了解的有效性。选择算子是种群进化的关键步骤，利用基于密度的选择机制确定参与下一迭代的烟花，传递种群的优势和多样性；未被选中的烟花或火花将永久消失，突显 FWA 的瞬时性特点。这 4 个步骤共同影响算法的全局搜索能力及收敛性能。

烟花算法具有爆炸性、并行性、多样性、有效性、瞬时性等特点，同时具备开发和探索能力，能够精确地搜索全局最优解。此算法实现简单，易于与其他算法结合，且能有效避免早熟收敛，因此，它被广泛应用于多种复杂优化问题的求解，包括多模态优化问题、滤波器优化、调度问题、垃圾邮件检测、路径规划以及图像配准等。

3.5.2 基本操作

1. 爆炸算子

爆炸算子主要由三部分组成：烟花的爆炸幅度（半径）、烟花的爆炸火花数目以及位移操作。烟花的适应度值决定其爆炸幅度和爆炸火花的数量；位移操作则生成所有爆炸火花的位置。

1）爆炸幅度

假设最大爆炸半径为常数 A，每代烟花种群个数为 N，烟花 x_i 的适应度值为 f_i，其所处烟花 x_i 种群中最优个体的适应度值为 $f_{\min}$。考虑到计算机极小常量为 η，烟花 x_i 的爆炸半径 R_i 可计算为

$$R_i = A \times \frac{f_i - f_{\min} + \eta}{\sum_{i=1}^{N}(f_i - f_{\min}) + \eta} \tag{3.35}$$

2）爆炸火花数目

假设最大爆炸火花数为常数 B，烟花 x_i 种群中最差个体的适应度值为 $f_{\max}$，据此计算烟花 x_i 的爆炸火花数 NUM_i：

$$\mathrm{NUM}_i = B \times \frac{f_{\max} - f_i + \eta}{\sum_{i=1}^{N}(f_{\max} - f_i) + \eta} \tag{3.36}$$

为确保烟花的爆炸火花数目适中，调整烟花实际产生的爆炸火花数目以保持在一定范围内。烟花 x_i 的实际爆炸火花数 NUM'_i 按以下公式计算。

$$\mathrm{NUM}'_i = \begin{cases} \mathrm{round}(mB), & \mathrm{NUM}_i < mB \\ \mathrm{round}(nB), & \mathrm{NUM}_i > nB \\ \mathrm{round}(\mathrm{NUM}_i), & \text{其他情况} \end{cases} \tag{3.37}$$

其中，m 和 n 为两个常数，满足 $m<n<1$；$\mathrm{round}(\cdot)$ 为四舍五入计算得到的整数。

3）位移操作

生成所有爆炸火花。特别地，通过在随机选定的维度 k 上产生新的爆炸火花位置 x_{ik}^{ex} 来模拟烟花的爆炸。这些位置计算如下。

$$x_{ik}^{\mathrm{ex}} = x_i^k + R_i \times \mathrm{rand}[-1,1] \tag{3.38}$$

其中，x_i^k 表示烟花 x_i 在维度 k 的位置，rand[−1,1]为在[−1,1]范围内的均匀随机数。

2. 高斯变异算子

高斯变异算子包括三个主要步骤：首先，随机选择一小部分烟花；其次，从所有可能的维度{0,1,…,k,…,D}中依次使用 $z^k = \mathrm{round}(k \times \mathrm{rand}[0,1])$函数近似均匀地随机选择几个维度；最后，对所有选中的维度进行高斯变异操作。由于高斯变异操作的范围远大于烟花的爆炸范围，此操作有助于增强种群的多样性。

如果烟花 x_i 的 k 维度被选中，则其高斯变异操作执行如下。

$$x_{ik}^{\mathrm{mu}} = x_i^k \times Y \tag{3.39}$$

其中，x_{ik}^{mu}表示烟花 x_i 在维度 k 上新产生的高斯变异火花的位置。Y 满足 $Y \sim N(1,1)$。

3. 映射规则

当爆炸火花 x_{ik}^{ex}或高斯变异火花 x_{ik}^{mu}超出可行域时，使用模运算映射规则将其重新映射到可行域内，以确保解的有效性。

$$x_{ik}^{\mathrm{ALL}} = R_{\mathrm{low},k} + |x_{ik}^{\mathrm{ALL}}| \bmod (R_{\mathrm{high},k} - R_{\mathrm{low},k}) \tag{3.40}$$

其中，x_{ik}^{ALL}表示第 i 个越界火花(包括所有越界的爆炸火花和高斯变异火花)的 k 维映射后的位置，x_{ik}^{ALL}为第 i 个越界火花的 k 维映射前的位置，$R_{\mathrm{high},k}$表示维度 k 的上限，$R_{\mathrm{low},k}$表示维度 k 的下限。

4. 选择算子

为保持种群的优越性和多样性，选择算子在每次迭代中从所有烟花、爆炸火花和高斯变异火花中选出两类个体进入下一迭代：第一类为最优个体；第二类为基于欧氏距离的轮盘赌方法随机选择的 $N-1$ 个非最优个体。其中，与其他烟花或火花的欧氏距离较大者有更高的被选中概率。此外，未被选中进入下一代的烟花或火花将被永久移除。以下是第 S 次迭代过程中类型 2 的 $N-1$ 个非最优个体被选中的示例说明。

假设第 S 代中烟花、爆炸火花及高斯变异火花的总数为 Q，且非最优烟花或火花的集合为 $X_S = \{x_{S_1}, x_{S_2}, \cdots, x_{S_i}, \cdots, x_{S_{(Q-1)}}\}$，那么集合 X_S 中任一非最优烟花或火花 x_{S_i} 被选中的概率计算如下。

$$p_i = \frac{\sum\limits_{S_j=1, S_j \neq S_i}^{Q-1} \| x_{S_i} - x_{S_j} \|}{\sum\limits_{S_i=1}^{Q-1} \sum\limits_{S_i=1, S_j=1}^{Q-1} \| x_{S_i} - x_{S_j} \|} \tag{3.41}$$

其中，$\| x_{S_i} - x_{S_j} \|$表示烟花 / 火花 x_{S_i} 和 x_{S_j} 之间的欧氏距离。$\sum\limits_{S_j=1, S_j \neq S_i}^{Q-1} \| x_{S_i} - x_{S_j} \|$表示烟花 / 火花 x_{S_i} 和集合 X_S 中其余烟花 / 火花的欧氏距离之和。

3.5.3　算法流程

在随机初始化烟花的初始位置后，依次执行 4 个基本操作，并持续迭代以实现烟花种群的进化，直至收敛至全局最优解。烟花算法的整体流程具体如下。

(1) 初始化参数。包括路径集合、所有常数因子、最大迭代次数及每代蚂蚁数量；初始化时，设定当前时刻为 0，路径信息素浓度为 0。

(2) 初始化阶段。设定每代烟花个数 N、最大迭代次数、机器极小常量 η 等参数，并确

定问题的可行域。在此范围内随机初始化 N 个烟花位置，并评估所有烟花的适应度值。

(3) 爆炸阶段。利用爆炸算子产生爆炸火花，并评估所有爆炸火花的适应度。

(4) 变异阶段。选取少量烟花进行高斯变异算子操作以生成高斯变异火花，并评估这些火花的适应度值。

(5) 映射阶段。检查第(2)步和第(3)步中生成的火花是否超出可行域，对超出的火花使用映射规则调整至可行域内，其他保持不变。

(6) 终止条件判断。若达到最大迭代次数、接近全局最优解或达到最大运行时间，则结束循环并输出全局最优解；否则，继续循环本步。

(7) 选择阶段。使用选择算子从本轮迭代的所有烟花和火花中挑选 N 个进入下一轮迭代的烟花，然后回到第(2)步。

烟花算法的算法流程图如图 3.8 所示。

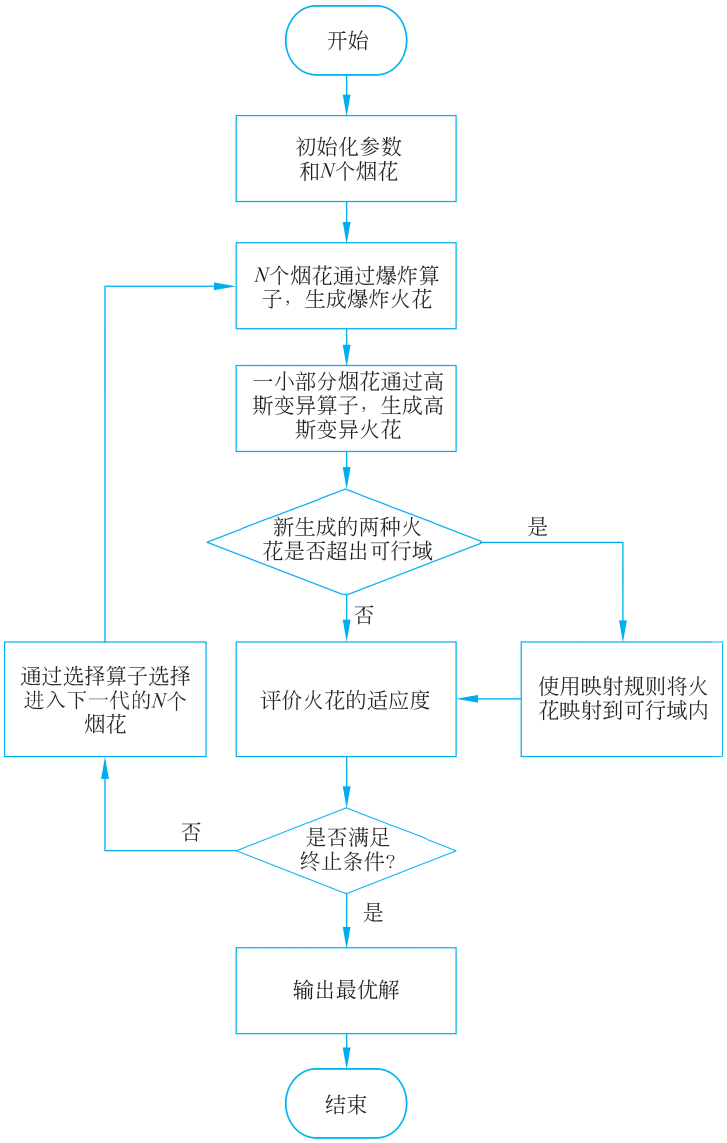


图 3.8 烟花算法的算法流程图