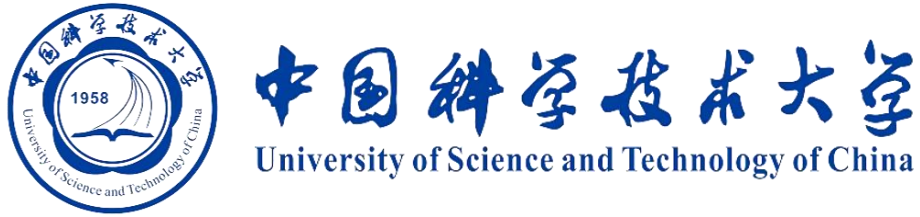


中国科学技术大学
University of Science and Technology of China

实用算法设计

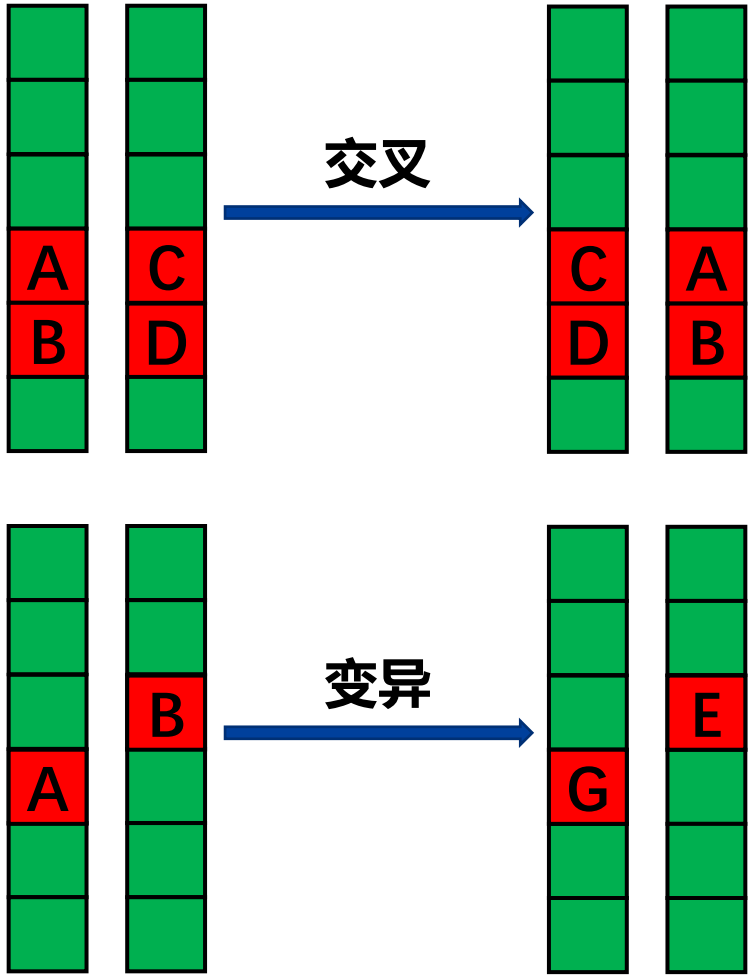
➢ 启发式搜索算法

主讲：娄文启

目录

遗传算法（GA）

- **来源**
  - 遗传算法（Genetic Algorithm，GA）是模拟生物在自然环境中的遗传和进化的过程而形成的自适应全局优化搜索算法。
- **基本原理**
  - 编码
  - 复制、交叉、变异



一个染色体表示一个个体，往往也是一个解决方案

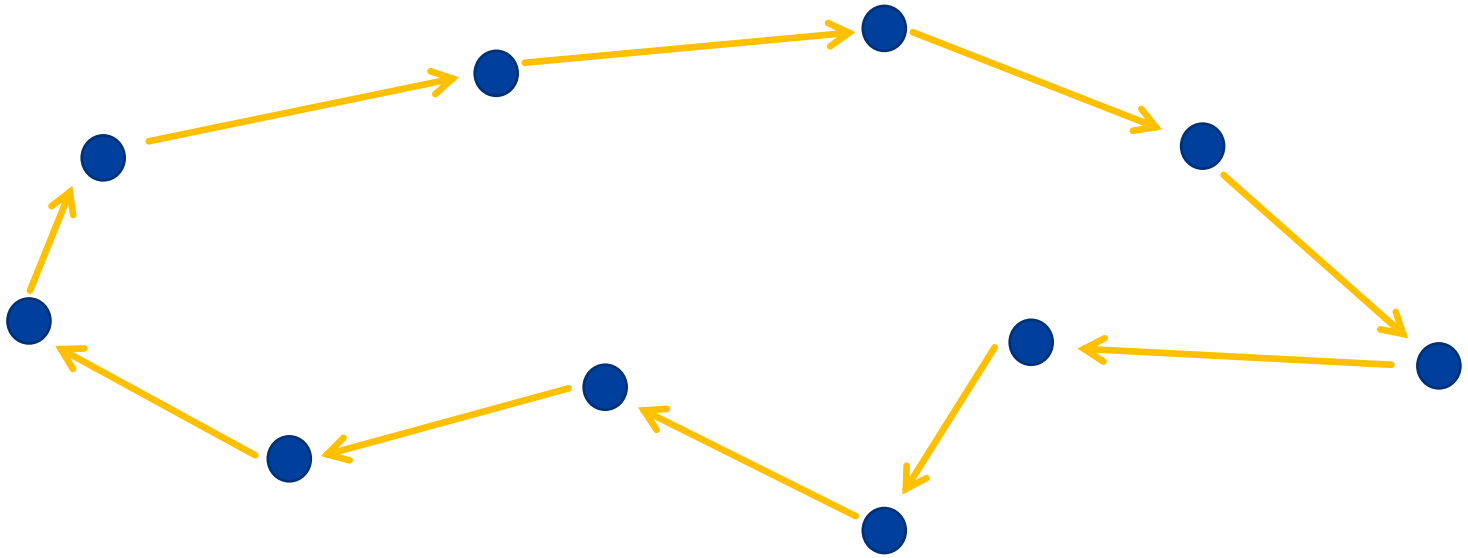
如何将问题的解转换为“染色体”是一个关键问题。

基因?

遗传算法 — 编码

- **编码：将问题的可行解，抽象编码为适用于遗传算法的形式**
  - **TSP问题中，比如10个城市，某个解可以表示为 [3,2,1,4,5,6,7,8,9,10]**
  - **根据不同的问题，进行不同的抽象**

TSP：给定一系列城市和每对城市之间的距离，**求解访问每座城市一次**并回到起始城市的**最短**回路。

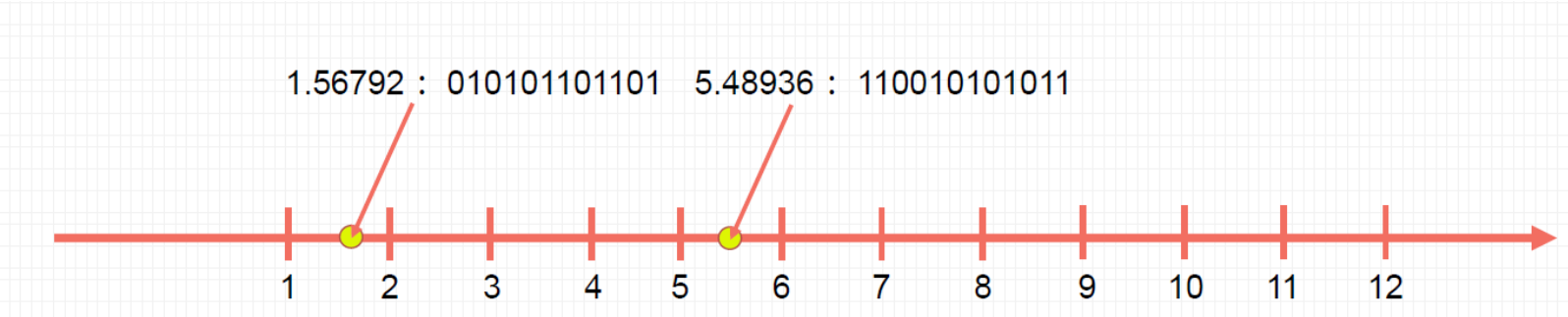


将编码位置作为城市序号

遗传算法 — 编码 （格式）

- **二进制编码：用一个二进制串表示一个十进制数**
  - 是一种映射（一对一），并不一定是十进制对应的二进制表示
  - 一个长度为n的串，可以表示的数值个数为2^n，用来表示[a, b]中的数
  - 类似于线性均匀量化，因此存在精度

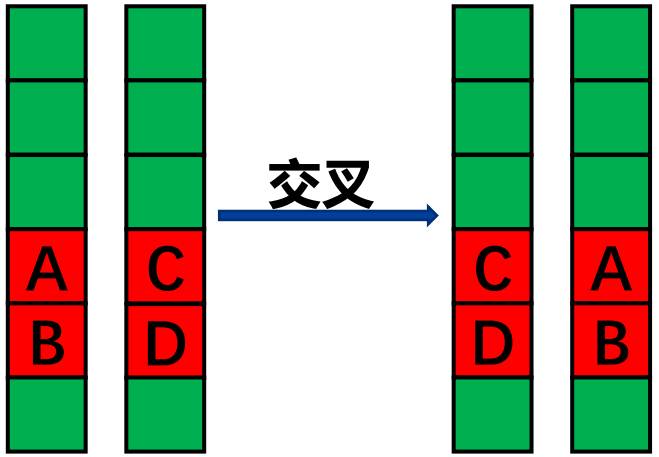
$$X = a + T \times \frac{b-a}{2^n - 1}$$



一般的，区间范围为 [a,b]，区间长度为 L，即 L=b−a，串长为 n，当前串对应十进制为 T，则该串应实值解为：

遗传算法 — 复制交叉

- **复制**
  - 将个体适应度大小映射为概率进行复制：适应度高的个体有更大的概率复制-轮盘赌法
  - 对适应度高的前N/4的个体进行复制，然后替换掉后N/4个体 - 精英法
  - ……
- **交叉**
  - 单点交叉
  - 两点交叉
  - 多点交叉
  - 均匀交叉
  - ……



Algorithm 1 Single-Point Crossover

Require: Parents P_1, P_2 (arrays of length n)

Ensure: Offspring C_1, C_2

1: Choose a random $k \in [1, n-1]$

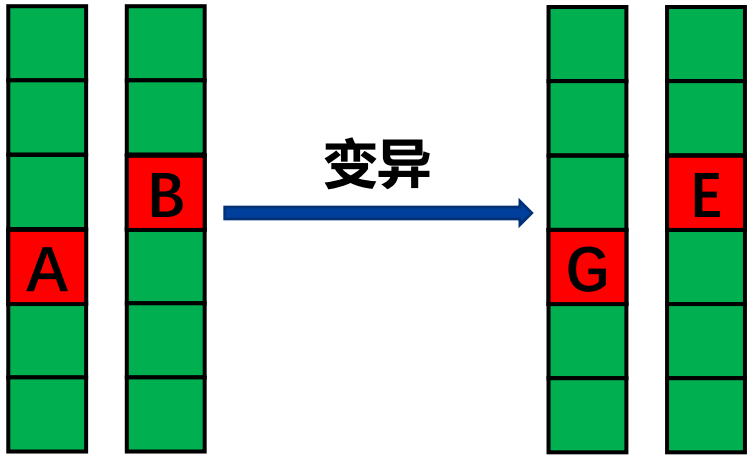
2: $C_1 \leftarrow P_1[1:k] + P_2[k+1:n]$

3: $C_2 \leftarrow P_2[1:k] + P_1[k+1:n]$

4: **return** C_1, C_2

遗传算法 — 变异

- 变异
  - 单点位变异
  - 只对适应度低的后N/4，或者后N/2个体变异
  - 按适应度大小映射变异概率
  - 多个点位变异



Algorithm 2 Mutation

Require: Individual I (length n), mutation rate p

Ensure: Mutated I'

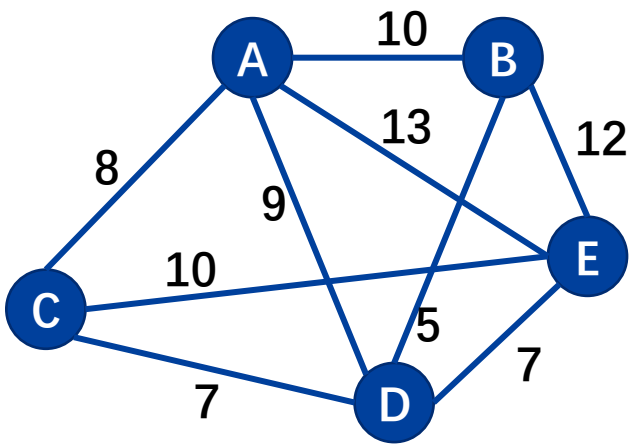
1: $I' \leftarrow I$
2: Generate $r \in [0, 1]$
3: **if** $r < p$ **then**
4: Randomly pick $k \in [1, n]$
5: Mutate $I'[k]$
6: **end if**
7: **return** I'

遗传算法 — 拓展与分析

- 优点
  - 参数少，理论优势
  - 变异机制赋予了群体跳出局部极值的能力
- 缺点
  - 容易陷入局部最优
  - 算法实现较为繁琐
- 策略
  - 适应度函数尺度化
    - 遗传初期缩小尺度差别，减少超级个体竞争力
    - 遗传后期扩大尺度差别，加快收敛
  - 灾变

遗传算法 — 例子

- **旅行商问题**
  - **从A出发，经过每个结点后返回A点，寻找总行程最短的路径**
  - **编码方式：**将路径表示为一串节点序列，例如 [A, D, B, E, C, A]
  - **适应度：**由路径长度决定（越短越好）



1. **初始化种群：**随机生成若干条合法路径作为初始种群

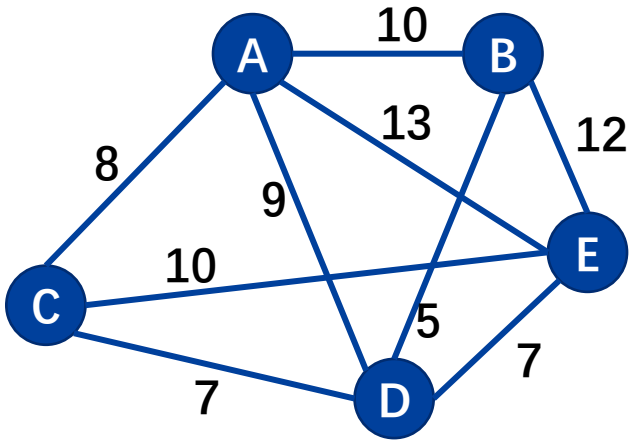
   个体 1: [A, D, B, E, C, A]

   个体 2: [A, B, D, C, E, A]

   个体 3: [A, C, D, B, E, A]
2. **适应度评估**
3. **选择**
4. **交叉变异**
5. **迭代与收敛**

遗传算法 — 例子

- **旅行商问题**
  - **从A出发，经过每个结点后返回A点，寻找总行程最短的路径**
  - **编码方式：**将路径表示为一串节点序列，例如 [A, D, B, E, C, A]
  - **适应度：**由路径长度决定（越短越好）



1. **初始化种群：**随机生成若干条合法路径作为初始种群
2. **适应度评估：**F(x)=1/路径长度

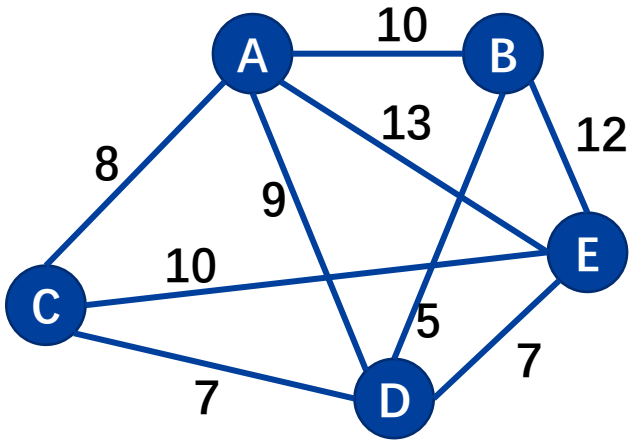
   个体 1: [A, D, B, E, C, A] ------ 适应度：1/44

   个体 2: [A, B, D, C, E, A] ------ 适应度：1/45

   个体 3: [A, C, D, B, E, A] ------ 适应度：1/45
3. **选择**
4. **交叉变异**
5. **迭代与收敛**

遗传算法 — 例子

- **旅行商问题**
  - **从A出发，经过每个结点后返回A点，寻找总行程最短的路径**
  - **编码方式：**将路径表示为一串节点序列，例如 [A, D, B, E, C, A]
  - **适应度：**由路径长度决定（越短越好）



1. **初始化种群**
2. **适应度评估**
3. **选择：选择适应度高的个体**
4. **交叉变异：**
   - **两个个体交叉形成新路径**
   - **随机交换路径中的两个节点**
   - **检查合法性**
5. **迭代与收敛**

[A, D, B, E, C, A]
[A, C, D, B, E, A]
→ [A, C, D, B, E, A]

[A, C, D, B, E, A] → [A, C, E, B, D, A]

粒子群算法（PSO）

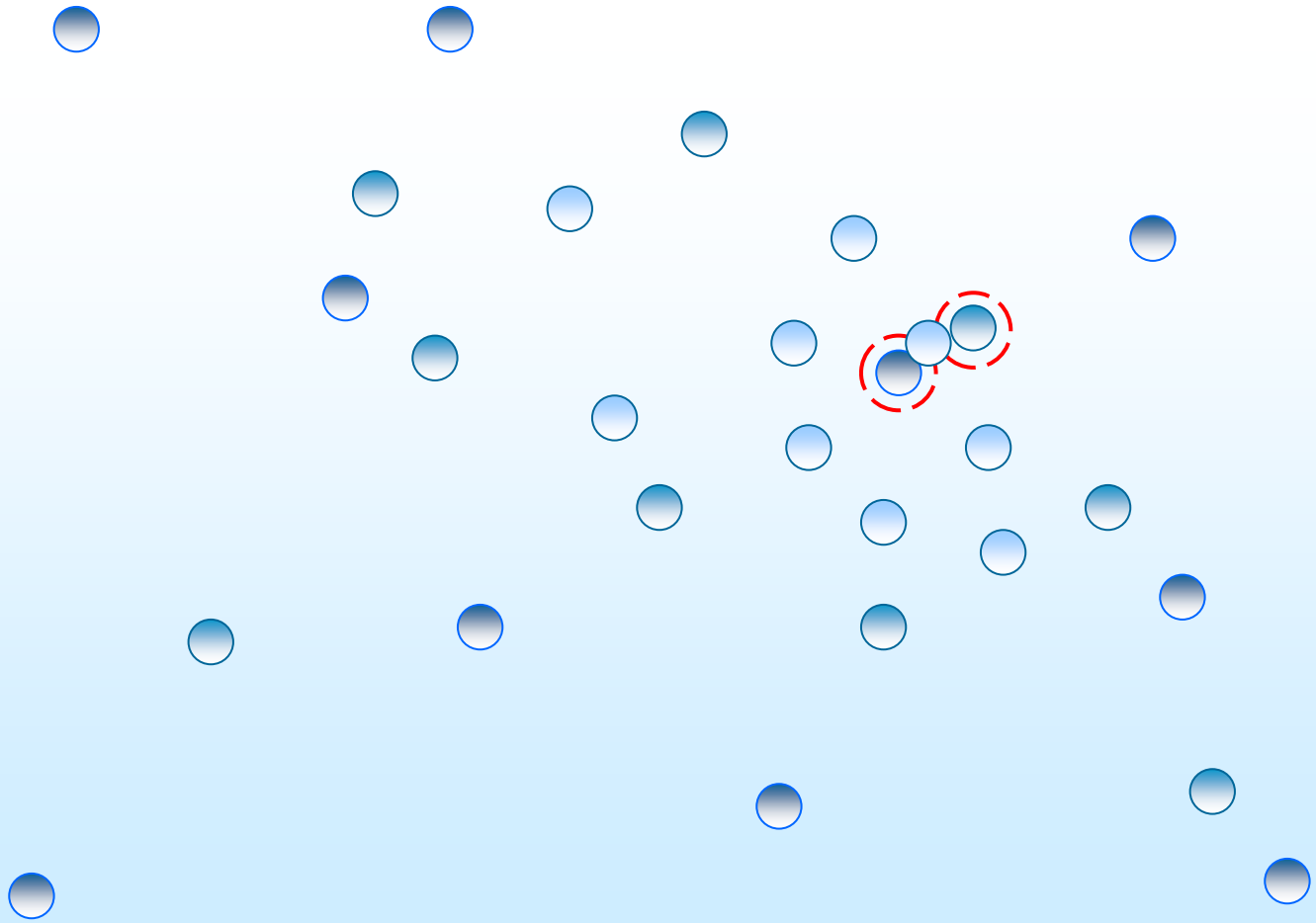
来源

PSO模拟的是鸟群的捕食行为。

设想这样一个场景：一群鸟在随机搜索食物。在这个区域里只有一块食物。所有的鸟都不知道食物在那里。但是他们知道当前的位置离食物还有多远。那么找到食物的最简单有效的就是搜寻目前离食物最近的鸟的周围区域。

鸟群在整个搜寻的过程中，通过相互传递各自的信息，让其他的鸟知道自己的位置，通过这样的协作，来判断自己找到的是不是最优解，同时也将最优解的信息传递给整个鸟群，最终，整个鸟群都能聚集在食物源周围，即找到了最优解。

粒子群算法 — 基本原理

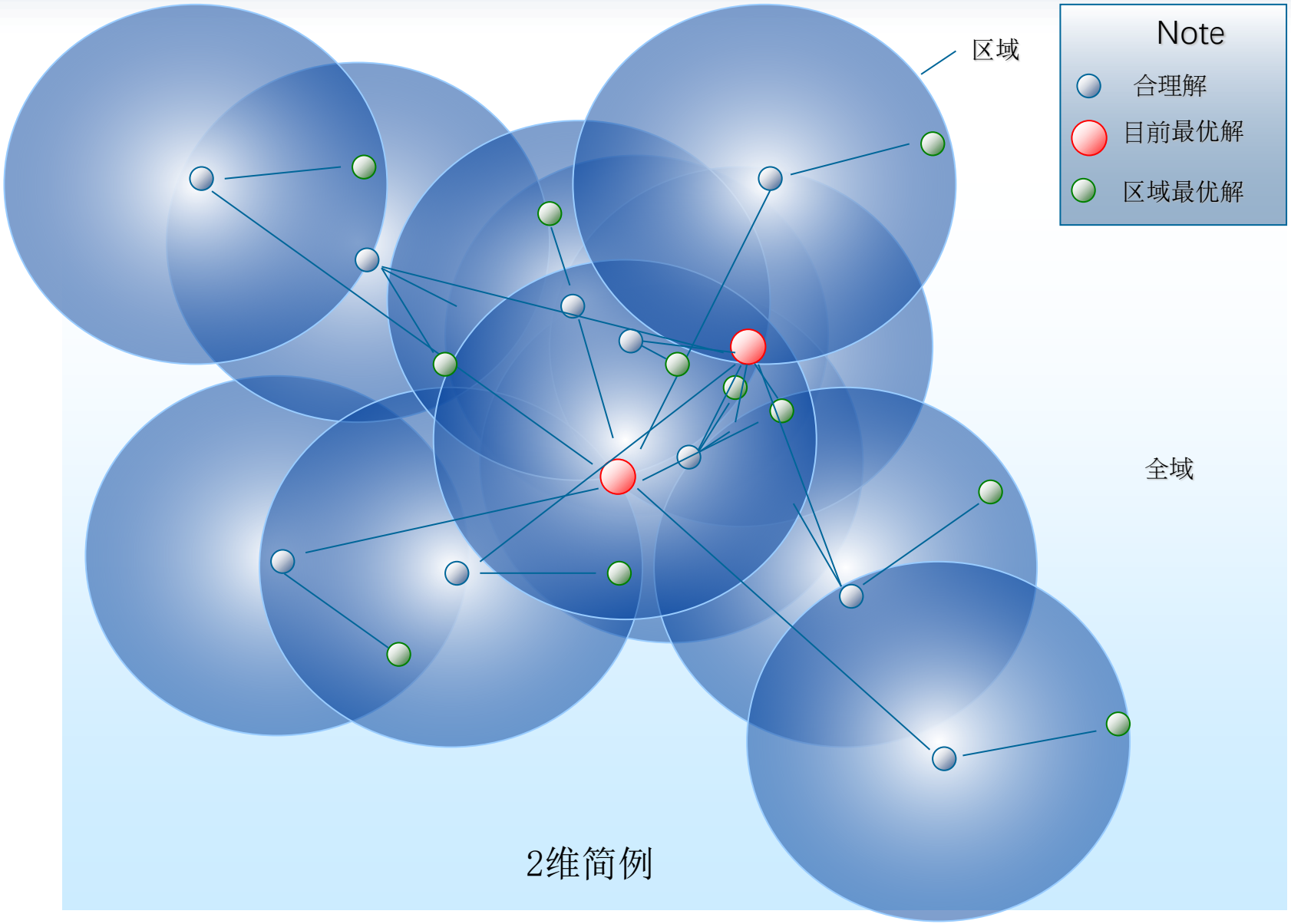


- 每个寻优的问题解都被想像成一只鸟，称为“粒子”。所有粒子都在一个D维空间进行搜索。
- 所有的粒子都由一个适应度函数判断目前的位置好坏。
- 每一个粒子必须赋予记忆功能，能记住所搜寻到的最佳位置。
- 每一个粒子还有一个速度以决定飞行的距离和方向。这个速度根据它本身的飞行经验以及同伴的飞行经验进行动态调整。

粒子群算法 — 基本原理

- D维空间中，有N个粒子
  - 粒子i位置：$\mathbf{x_i} = (\mathrm{xi}_1, \mathrm{xi}_2, \ldots \mathrm{xiD})$
  - 粒子i速度：$\mathbf{v_i} = (\mathrm{vi}_1, \mathrm{vi}_2, \ldots \mathrm{viD})$
  - 粒子i个体经历过的最好位置：$\mathbf{pbest_i} = (\mathrm{pi}_1, \mathrm{pi}_2, \ldots \mathrm{piD})$
  - 种群所经历过的最好位置：$\mathbf{gbest} = (g_1, g_2, \ldots gD)$
- 粒子i在第d维速度更新公式
  - $v_{id}^{k} = wv_{id}^{k-1} + c_1 r_1 (pbest_{id} - x_{id}^{k-1}) + c_2 r_2 (gebst_d - x_{id}^{k-1})$
- 粒子i在第d维位置更新公式
  - $x_{id}^{k} = x_{id}^{k-1} + v_{id}^{k-1}$
- 参数说明
  - x_{id}^{k}: 第k次迭代粒子i飞行速度矢量的第d维分量
  - v_{id}^{k}: 第k次迭代粒子i位置矢量的第d维分量
  - c_1, c_2：加速度常数，调节学习最大步长
  - r_1, r_2 ：两个随机数，取值范围[0，1]，以增加搜索随机性
  - w：惯性权重，非负数，调节对解空间的搜索范围

粒子群算法 — 基本原理



粒子群算法 — 算法流程

- **算法步骤**
  - **初始化粒子群**
  - **个体和全局最佳位置更新**
  - **个体速度和位置更新**
  - **迭代寻优**

Algorithm 3 Particle Swarm Optimization (PSO)

Require: N (number of particles), T (iterations), $f(x)$ (fitness function)
Ensure: Best solution x^*

Initialize particles' positions x_i, velocities v_i, personal best positions p_i, and global best position g
for $t = 1$ to T **do**
  for $i = 1$ to N **do**
    Evaluate $f(x_i)$
    if $f(x_i) < f(p_i)$ **then**
      $p_i \leftarrow x_i$
    end if
    if $f(x_i) < f(g)$ **then**
      $g \leftarrow x_i$
    end if
    Update velocity: $v_i \leftarrow wv_i + c_1 \cdot \text{rand}() \cdot (p_i - x_i) + c_2 \cdot \text{rand}() \cdot (g - x_i)$
    Update position: $x_i \leftarrow x_i + v_i$
  end for
end for
return g

粒子群算法 — 分析与拓展

- **优点**
  - 原理比较简单、实现容易、参数少
- **缺点**
  - 容易早期收敛直局部最优
  - 迭代后期收敛速度慢
- **拓展**
  - ω描述的是粒子的“惯性”，在进化前期ω应该大一些，保证各个粒子独立飞行充分搜索空间，后期应该小一点，多向其他粒子学习
  - c_1, c_2分别向个体极值和全局极值最大飞行步长。前期c_1应该大一些，后期c_2应该大一些，这样就能平衡粒子的全局搜索能力和局部搜索能力

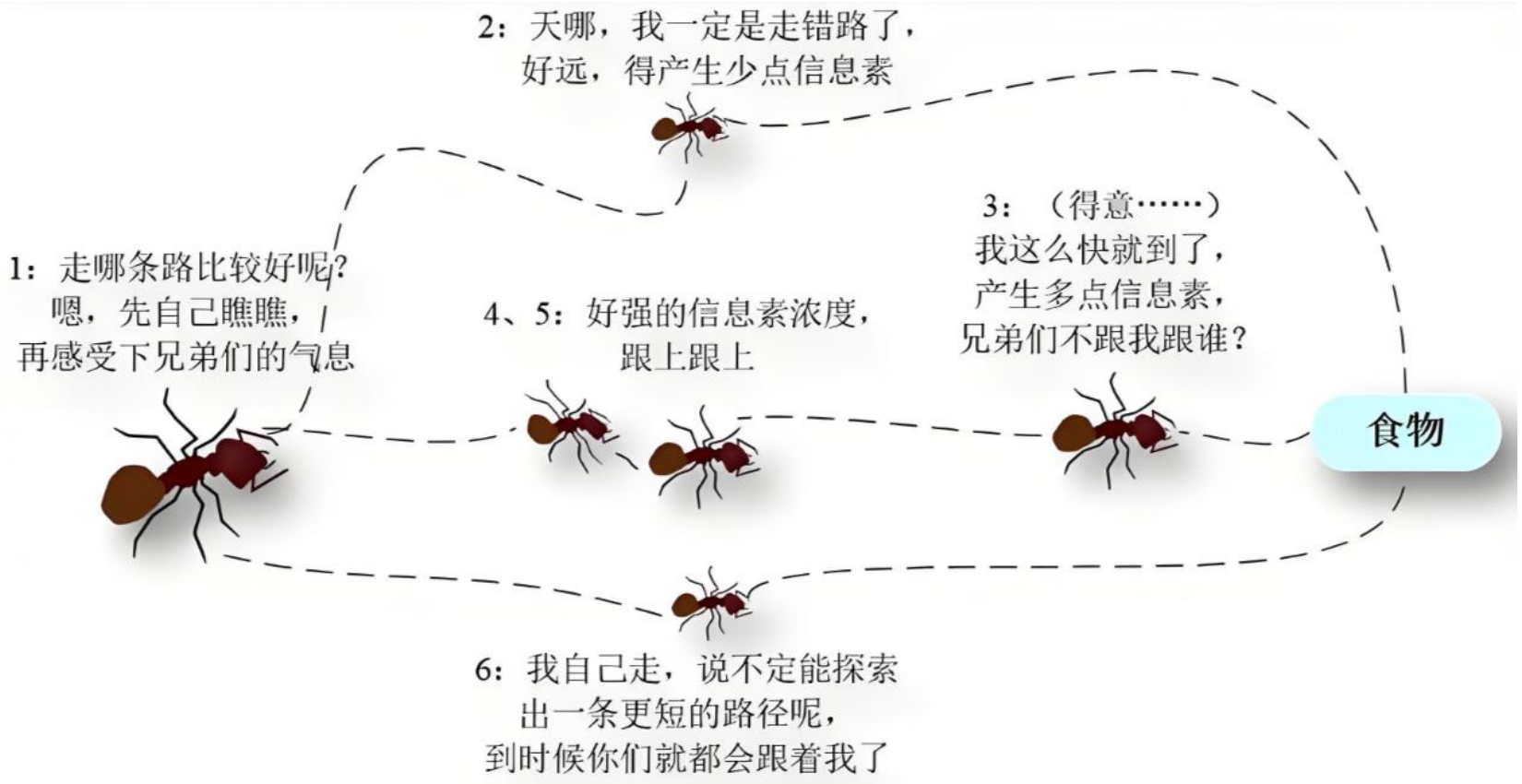
蚁群算法（ACO）

来源

蚁群算法的灵感来源于自然界中蚂蚁寻找食物的行为。

蚂蚁在寻找食物的过程中会释放一种称为信息素的化学物质，这种物质会在蚂蚁走过的路径上留下痕迹，后续的蚂蚁会根据这些信息素的浓度来选择路径，从而形成一条从蚁巢到食物源的最短路径。

蚂蚁觅食的过程是一个正反馈的过程，该路段经过的蚂蚁越多，信息素留下的就越多，浓度越高，更多的蚂蚁都会选择这个路段。



蚁群算法 — 基本原理

概率公式

- 每只蚂蚁根据概率 P 选择路径：$P_{ij}^{k}(t)=\frac{[\tau_{ij}(t)^{\alpha}]\cdot[\eta_{ij}(t)^{\beta}]}{\sum l\in allowed[\tau_{il}(t)^{\alpha}]\cdot[\eta_{il}(t)^{\beta}]}$
- $P_{ij}^{k}(t)$：表示蚂蚁k从节点i移动到节点j的概率
- $\tau_{ij}(t)$：表示节点i到节点j的信息素浓度
- $\eta_{ij}(t)$：表示启发式信息（如节点i到节点j距离的倒数）
- α，β：信息素和启发式信息的重要度参数

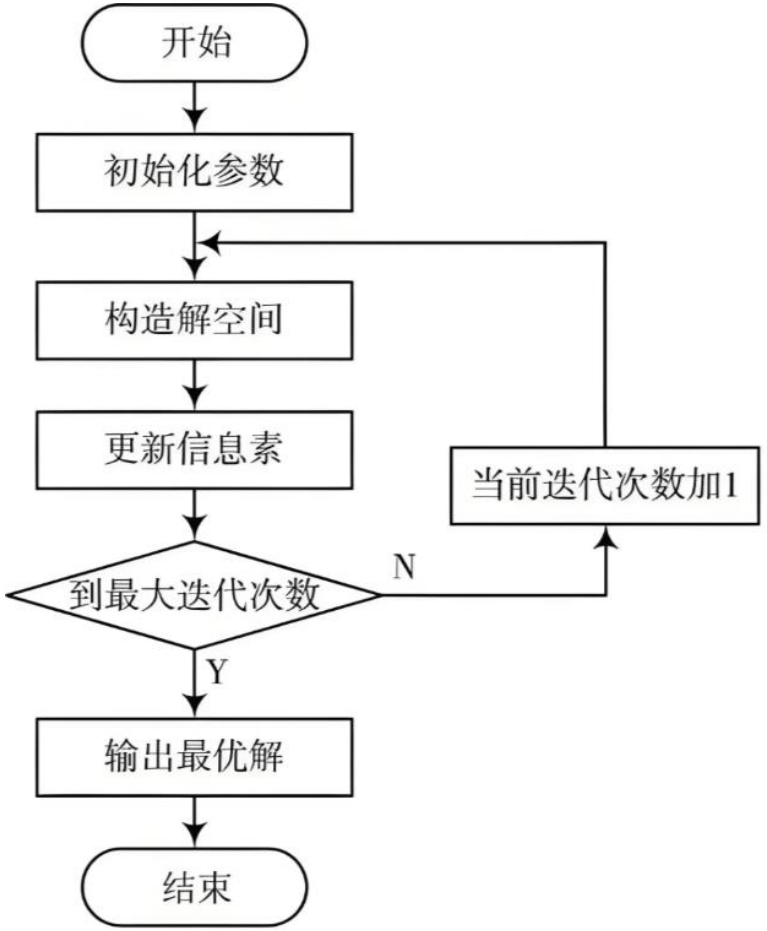
更新信息素浓度

- 每只蚂蚁完成一次路径选择后，根据路径质量更新信息素浓度：

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

- ρ：信息素挥发系数
- $\Delta\tau_{ij}^{k}$：表示蚂蚁k在路径i->j上留下的信息素增量 $1/f(x_k)$

蚁群算法 — 算法流程



Algorithm 7 Ant Colony Optimization (ACO)

Require: N (number of ants), T (iterations), $f(x)$ (fitness function), α (pheromone influence), β (heuristic influence), ρ (pheromone evaporation rate), τ_0 (initial pheromone level)

Ensure: Best solution x^*

Initialize pheromone levels: $\tau_{ij} = \tau_0 \quad \forall (i,j)$
for $t = 1$ to T **do**
  for each ant k **do**
    Initialize ant's solution $x_k = \emptyset$
    for each step s in ant's path **do**
      Calculate probability of moving from node i to node j:

$$P_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

      Ant selects next node j based on P_{ij}
    end for
    Evaluate solution x_k using fitness function $f(x_k)$
    Update best solution if necessary
  end for
  for each ant k **do**
    Calculate pheromone deposit:

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{f(x_k)} & \text{if ant } k \text{ passed through edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

  end for
  Update pheromone levels:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t)$$

end for
return best solution found x^*

模拟退火（SA）

来源

模拟退火算法(SA)来源于固体退火原理，是一种基于概率的算法。

模拟退火算法从某一较高初温出发，伴随温度参数的不断下降,结合概率突跳特性在解空间中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。

一句话就是：如果新状态的解更优则修改答案，否则以一定概率接受新状态。

算法原理

$$P=\begin{cases}1, & \text{if } \Delta E \leq 0, \\ \exp\left(-\frac{\Delta E}{T}\right), & \text{if } \Delta E > 0,\end{cases}$$

模拟退火 — 算法流程

算法流程

- 初始化：初始解 S、初始温度 T、最低温度 T_{min}、降温速率 α。
- 搜索迭代
  - 随机生成新解 S′ 。
  - 若新解更优，接受；否则以概率 $P = exp(-\Delta E/T)$ 接受。
  - 更新当前解 S 和温度 $T = \alpha T$ 。
- 结束条件：当 $T \leq T_{min}$停止，输出最优解

Algorithm 8 Simulated Annealing (SA)

Require: Initial solution S, initial temperature T, cooling rate α, minimum temperature $T_{\min}$
Ensure: Best solution S^*

```
Initialize solution S, evaluate f(S)
while T > T_min do
    Generate new solution S' near S
    Evaluate f(S')
    if f(S') < f(S) then
        S ← S'
    else
        Accept S' with probability P = exp(−ΔE/T)
    end if
    Cool down: T ← αT
end while
return S
```

中国科学技术大学
University of Science and Technology of China

算法	原理	适用问题	优点	缺点	适用问题	收敛速度	计算复杂度	参数敏感性	是否易陷入局部最优
遗传算法 (GA)	自然选择	连续/离散优化	全局搜索能力强 适应性广	收敛速度较慢 参数较多	函数优化 路径规划	慢	高	高	是
粒子群算法 (PSO)	鸟群觅食	连续优化问题	算法简单、易于实现	容易陷入局部最优 初始参数设置敏感	函数优化 数据聚类	快	中	中	是
蚁群算法 (ACO)	蚂蚁觅食	离散优化问题	全局搜索能力强	收敛速度较慢	旅行商问题 物流配送	中	高	高	是
模拟退火算法 (SA)	物理退火	连续优化问题	有效避免局部最优	收敛速度较慢 依赖温度下降策略	组合优化问题	慢	中	高	否

NSGA-II

- **NSGA - II**
  - NSGA-II（Non-dominated Sorting Genetic Algorithm II）是一种多目标优化算法，常用于解决具有多个冲突目标的问题。
  - 它是遗传算法的一种扩展版本，通过引入快速非支配排序和拥挤距离等机制，提高了优化效率和解的多样性。
  - 基本步骤和基础的遗传算法一致，主要增加了非支配排序和拥挤距离的概念来解决多目标优化排序困难的问题。

NSGA-II — 支配

- **帕累托前沿（帕累托最优，Pareto front)**
  - 帕累托前沿由一组解组成，这些解在所有目标上都不可被其他解所支配。
- **支配关系：一个解 A 支配另一个解 B，当且仅当：**
  - A 在所有目标上至少和 B 一样好
  - 并且 A 至少存在一个目标优于 B

NSGA-II — 拥挤距离

- **拥挤距离**
  - 在多目标优化中，拥挤距离是用于衡量一个解在目标空间中与其他解的相对距离的指标。
  - 拥挤距离越大，表示该解在目标空间中越“稀疏”，也就是说，解之间的间隔较大。
  - 反之，拥挤距离较小的解则位于目标空间的拥挤区域，解之间的间隔较小
- **拥挤距离越大**
  - 这里解比较空旷
  - 选择拥挤距离大的解，有利于种群多样性
- **因此，非支配解之间就能通过拥挤距离来比较大小，比较大小的问题就得到了解决**

NSGA-II — 拥挤距离

Algorithm 4 Crowding Distance Calculation

Require: Population $P = \{S_1, S_2, \ldots, S_N\}$, number of objectives M

Ensure: Crowding distances $d(S_1), d(S_2), \ldots, d(S_N)$

1: Initialize $d(S_i) = 0$ for all i
2: **for** each objective m **do**
3: Sort population P by m-th objective
4: $d(S_1) = d(S_N) = \infty$
5: **for** $i = 2$ to $N - 1$ **do**
6: $d(S_i) \leftarrow d(S_i) + \frac{f_m(S_{i+1}) - f_m(S_{i-1})}{f_m^{\max} - f_m^{\min}}$
7: **end for**
8: **end for**
9: **return** $d(S_1), d(S_2), \ldots, d(S_N)$

NSGA-II — 非支配排序

- **非支配排序**

  它通过比较解之间的支配关系来对种群中的解进行排序，目的是将解分成多个等级（或层次），每一层包含互不支配的解，最终生成一个“非支配解集”

最坏复杂度$O(MN^3)$，M是obj个数，N是pop size

Algorithm 5 Non-dominated Sorting

Require: Population P
Ensure: Fronts F

```
Initialize i = 1, F = []
while Pop is not empty do
    S = ∅
    for each p_i ∈ P do
        for each p_j ∈ P do
            if p_j dominate p_i then
                p_i.dominate_count += 1
            end if
        end for
        if p_i.dominate_count == 0 then
            p_i.rank = i
            S.append(p_i)
        end if
    end for
    F.append(S)
    P = P − S
    i = i + 1
end while
return F
```

NSGA-II — 快速非支配排序

- **快速非支配排序**
  - $P_i.dominated\_set$：当前个体P所支配的那些个体
  - $P_i.dominate\_count$：能够支配当前个体的P的个数

最坏复杂度$O(MN^2)$，M是obj个数，N是pop size

Algorithm 6 Fast Non-dominated Sorting

Require: Population Pop
Ensure: Fronts F

Initialize $F_1 = \emptyset$
for each solution $p_i \in Pop$ **do**
  $p_i.dominate\_count = 0$, $p_i.dominated\_set = \emptyset$
  for each solution $p_j \in Pop$, $i \neq j$ **do**
    if p_i dominate p_j **then**
      $p_i.dominated\_set.add(p_j)$
    else if p_j dominate p_i **then**
      $p_i.dominate\_count+ = 1$
    end if
  end for
  if $p_i.dominate\_count = 0$ **then**
    $p_i.rank = 1$
    Add p_i to F_1
  end if
end for
$k = 1$
while F_k is not empty **do**
  Initialize $F_{k+1} = \emptyset$
  for each solution $p_i \in F_k$ **do**
    for each solution $p_j \in p_i.dominated\_set$ **do**
      $p_j.dominate\_count- = 1$
      if $p_j.dominate\_count = 0$ **then**
        $p_i.rank = k + 1$
        Add p_j to F_{k+1}
      end if
    end for
  end for
  $k+ = 1$
end while
return F

中国科学技术大学
University of Science and Technology of China

谢谢！