PSO_scheduler 类

1 类概述

SA_scheduler 类，继承 dynamic_scheduler 类。通过粒子群算法对 SFC 进行部署。

2 类属性

名称	描述	类型	默认值
__records	部署记录	{}	{}
pN	粒子数	int	10
max_iter	最大迭代次数	int	100
w_local	局部概率	int	0.3
w_global	全局概率	int	0.4
pbest	个体最优	{}	
gbest	全局最优		
p_fit	个体最佳适应度	[]	
g_fit	全局最优适应度	[]	
particles	例子群	[]	
all_sfc_deploy_solutions	sfc 的所有部署方案	{'sfc1':[{},{},{}],...}	无
solutions_length	每个 sfc 的部署方案个数	{'sfc1':10, 'sfc2':5,...}	无

3 类方法

名称	描述
__init__(pN=10,max_iter=100,w_local=0.3,w_global=0.4)	类内部构造函数
init_particle(network,sfcs,vnf_types,n)	初始化粒子，提前计算一些常用的量
sample_solution(sfc,X=3)	从 sfc 所有部署方案中选取方案构成粒子
clear_network(network,sfcs)	清除网络
deploy_sfc_by_records(sfcs,network,vnf_types,records)	通过记录执行部署操作并计算返回适应度
get_deploy_solution_from_particle(particle)	从粒子中获取部署方案
calculate_fit(sfcs,network,vnf_types,particle)	计算单个粒子适应度

calculate_fits(sfcs,network,vnf_types,neighbours)	计算所有粒子适应度
update(sfcs,network,vnf_types,index)	跟新粒子
deploy_sfcs(network,sfcs,vnf_types,init_record)	主函数

4 粒子群算法流程

4.1 经典粒子群优化算法简介

粒子群优化算法(PSO：Particle swarm optimization)源于对鸟群捕食的行为研究，是通过群体中个体之间的协作和信息共享来寻找最优解。

PSO 算法的优势在于简单容易实现并且没有许多参数的调节。通过设计一种无质量的粒子来模拟鸟群中的鸟，粒子仅具有两个属性：速度和位置，速度代表移动的快慢，位置代表移动的方向。每个粒子在搜索空间中单独的搜寻最优解，并将其记为当前个体极值，并将个体极值与整个粒子群里的其他粒子共享，找到最优的那个个体极值作为整个粒子群的当前全局最优解，粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。

PSO 初始化为一群随机粒子，即空间中的一组随机解，然后通过迭代找到最优解。在每一次的迭代中，粒子通过跟踪个体最优（pbest）、局部最优（gbest）来更新自己。在找到这两个最优值后，粒子通过下面的公式来更新自己的速度和位置。

$$\begin{aligned} v_i &= v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i) \\ x_i &= x_i + v_i \end{aligned} \quad (1)$$

上述是经典 PSO 算法，一般用于解决连续问题寻优。然而 SFCs 部署问题是一个离散问题，因此需要对经典 PSO 算法的步骤做出相应更改以适应我们的问题。

4.2 粒子个体编码

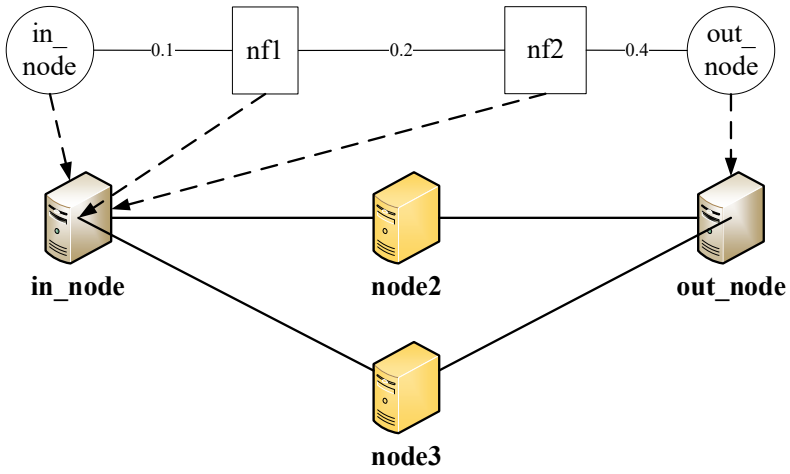
粒子表示将所有 SFCs 部署到网络中的一种部署策略，因此编码如下：

$$x_i = [1,1,2,\dots]$$

其中，数组里的第 i 项表示第 i 条 sfc，值表示第 j 种部署方法。个体编码不仅包括当前状态，还要对历史个体最优 pbest 编码。

4.3 初始化粒子群

对于每一条 sfc，其部署方案的所有选择有两种粒度，首先是选择不同的部署路径，其次是调整统一部署路径上的不同 nf 部署顺序，如下图所示的一条 sfc 的部署方案如下：



路径一　　路径二

{}

{‘sfc1’:{‘node’:{1:in_node,2: in_node }}　　{‘sfc1’:{‘node’:{1:in_node,2: in_node }}

{‘sfc1’:{‘node’:{1:in_node,2: node2 }}　　{‘sfc1’:{‘node’:{1:in_node,2: node3 }}

{‘sfc1’:{‘node’:{1:in_node,2: out_node }}　　{‘sfc1’:{‘node’:{1:in_node,3: out_node }}

{‘sfc1’:{‘node’:{1:node2,2: node2 }}　　{‘sfc1’:{‘node’:{1:node3,2: node3 }}

{‘sfc1’:{‘node’:{1:node2,2: out_node }}　　{‘sfc1’:{‘node’:{1:node3,2: out_node }}

{‘sfc1’:{‘node’:{1:out_node,2: out_node }　　{‘sfc1’:{‘node’:{1:out_node,2: out_node }}

一共有 M 种方案，方案为{}表示当前方案不部署。

若生成 N 个粒子，则每个粒子的每一条 sfc 部署方案从上述方案中选取一个方案，确保所有粒子中出现上述方案按均匀分布

1) 若 M<N，则按照循环顺序从 M 中取出 N 个解，即 1~M~1~M….

2) 若 M>N，由于产生部署方案时将时延小的方案放前面，所以按照的指数下降的方法从 M 中取 N 个解，如下图所示，首先将规定下降次数 X(X=3，表示 2^3=8 分)，即第一次将 1/8*N 分配给后 1/2M~M 解空间，第二次将 1/8 分配给 1/4M~1/2M 解空间，第三次将 1/4 分配给 1/8M~1/4M 解空间，最后将剩余大约 1/2N 分配给 0~1/8M 解空间

4.4 粒子更新

离散化的粒子更新策略，采用以下策略，

1) 速度的更新：由于 $gbest$ 、 $pbest_i$ 、 x_i 为一系列离散的值，其减法采用集合的减法概念，即：

$$\begin{aligned} pbest_i - x_i &= [1,1,1,\dots] \\ &\quad - [2,1,3,\dots] \\ &= [-1,0,-2,\dots] \\ gbest_i - x_i &= [3,2,3,\dots] \\ &\quad - [2,1,3,\dots] \\ &= [1,1,0,\dots] \end{aligned}$$

$pbest_i - x_i$也采用相同的方法，而

$$c_1 * rand() * (pbest_i - x_i) = [c_1 * rand() * \mathrm{SFC1}_1, 0, c_1 * rand() * \mathrm{SFC3}_2, \dots\dots]$$

$$c_2 * rand() * (gbest_i - x_i) = [c_2 * rand() * \mathrm{SFC1}_1, 0, c_2 * rand() * \mathrm{SFC3}_2, \dots\dots]$$

分别对列表中的每一项产生随机数进行学习

2) 位置的更新：

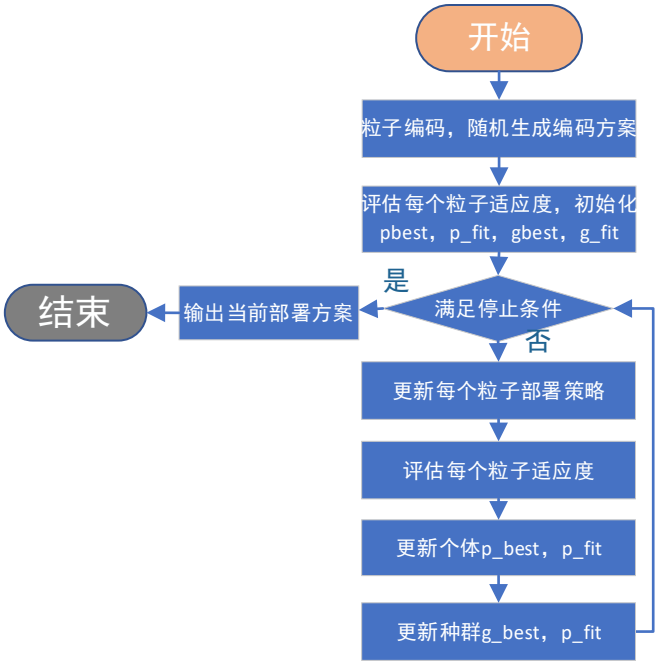
$$x_i = round(x_i + v_i)$$

最终需要对x_i的每一项进行判断，判断其是否在可取部署方案之内

4.5 适应值函数

使用网络部署的总流量为适应值函数，即寻优的目标函数是最大化网络流量。

4.6 算法流程


开始
粒子编码，随机生成编码方案
评估每个粒子适应度，初始化
pbest，p_fit，gbest，g_fit
是
满足停止条件
否
输出当前部署方案
结束
更新每个粒子部署策略
评估每个粒子适应度
更新个体p_best，p_fit
更新种群g_best，p_fit