HW2 第二次作业解答

更新：2025 年 11 月 4 日

项目复现见 `README.md`，使用 `sh main.sh` 或 `.\main.bat` 命令。

Exercise 1

结合教材例 3.1 的基因图谱案例，现提供 96 条模拟染色体序列数据（每条序列长度为 12，详见附件）。请基于这些数据，估计遗传距离和基因位点顺序。

a 估计遗传距离。
b 计算基因位点顺序全局最优解（$[2,4,12,6,3,7,1,11,9,8,5,10]$）的负对数似然。（注意：若将其顺序完全颠倒，表示的依然是同一个解）
c 使用 15 个随机初始值进行**随机下降**（邻域大小取 20）求解基因位点顺序并可视化算法收敛过程。
d 使用**禁忌搜索**求解基因位点顺序，比较不同**禁忌期限**的影响并可视化。
e 使用**模拟退火**求解基因位点顺序，找出一组合适的**温度控制参数**并可视化。
f 使用**遗传算法**，交叉算子选择**边缘重组交叉**求解基因位点顺序并可视化。

解答 (a) 针对目前的 12 个基因点位，标记 $l=1,2,\cdots,p$，其中 $p=12$。设 $\boldsymbol{\theta}=(\theta_1,\cdots,\theta_p)$ 为目标参数，其中 $\theta_j=l$ 意味着真实的第 j 个点位为 loc_l 基因。针对某一个解 $\boldsymbol{\theta}$，设它的相邻 θ_j,θ_{j+1} 的基因发生断裂的概率为 $d(\theta_j,\theta_{j+1})$ 称为遗传距离。

我们现在有 n 组样本 $\{\boldsymbol{x}_i : i=1,\cdots,n\}$，其中 $n=96$。每一个样本形如

$$\boldsymbol{x}_i=(x_{i,l})_{l=1,\cdots,p}\in\{0,1\}^{1\times p},\quad x_{i,l}\in\{0,1\}$$

那么针对某一解 $\boldsymbol{\theta}$（即 $l=1,\cdots,p$ 的某一排列）有 $d(\theta_j,\theta_{j+1})$ 的估计为

$$\hat{d}(\theta_j,\theta_{j+1})=\frac{1}{n}\sum_{i=1}^{n}|x_{i,\theta_j}-x_{i,\theta_{j+1}}|,\quad j=1,2,\cdots,p-1$$

例如，若按照 $\boldsymbol{\theta}=[1,2,\cdots,12]$ 排序，得到的估计为

$$\hat{\boldsymbol{d}}=(0.28,0.24,0.21,0.39,0.35,0.06,0.33,0.05,0.18,0.20,0.28)^T\in\mathbb{R}^{p-1}$$

解答 (b) 若给定相邻 θ_j, θ_{j+1} 的基因发生断裂的概率为 $d(\theta_j, \theta_{j+1})$，那么

$$\sum_{i=1}^{n} |x_{i,\theta_j} - x_{i,\theta_{j+1}}| \sim B(n, d(\theta_j, \theta_{j+1}))$$

于是，似然函数为

$$L(\boldsymbol{\theta}; \boldsymbol{x}) = \prod_{j=1}^{p-1} \prod_{i=1}^{n} d(\theta_j, \theta_{j+1})^{|x_{i,\theta_j} - x_{i,\theta_{j+1}}|} (1 - d(\theta_j, \theta_{j+1}))^{1 - |x_{i,\theta_j} - x_{i,\theta_{j+1}}|}$$

在估计值 $\hat{d}(\theta_j, \theta_{j+1})$ 条件下的对数似然为

$$l(\boldsymbol{\theta}|\hat{\boldsymbol{d}}) = \sum_{j=1}^{p-1} n\hat{d}(\theta_j, \theta_{j+1}) \log \hat{d}(\theta_j, \theta_{j+1}) + \sum_{j=1}^{p-1} (n - n\hat{d}(\theta_j, \theta_{j+1})) \log(1 - \hat{d}(\theta_j, \theta_{j+1}))$$

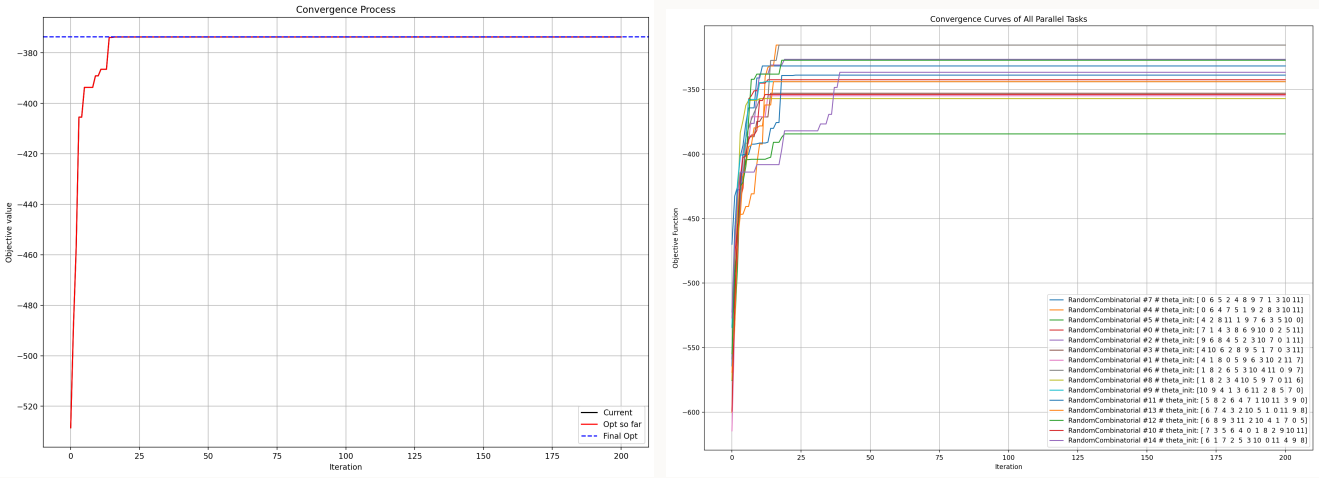
这是因为之前的估计值 $n\hat{d}(\theta_j, \theta_{j+1}) = \sum_{i=1}^{n} |x_{i,\theta_j} - x_{i,\theta_{j+1}}|$。为了提供计算效率，改为矩阵运算

$$l(\boldsymbol{\theta}|\hat{\boldsymbol{d}}) = n \cdot [\hat{\boldsymbol{d}}^T \log \hat{\boldsymbol{d}} + (\mathbf{1} - \hat{\boldsymbol{d}})^T \log(\mathbf{1} - \hat{\boldsymbol{d}})] \in \mathbb{R}$$

其中 $\log \hat{\boldsymbol{d}} = (\log \hat{d}(\theta_j, \theta_{j+1}))_{j=1,\cdots,p-1}^T \in \mathbb{R}^{p-1}$ 及 $\mathbf{1} = (1, \cdots, 1)^T \in \mathbb{R}^{p-1}$ 均为向量。

代入 $\boldsymbol{\theta}^* = (2, 4, 12, 6, 3, 7, 1, 11, 9, 8, 5, 10)$ 的解，其负对数似然为 298.3949。

解答 (c) 设置邻域大小为 20，不妨设每次生成邻域时交换 1 次（两个位点）。从 15 个初始点开始，随机组合以生成邻域，每次直接更新到邻域最优处，并行优化（进程数等于 CPU 核数）。设置迭代次数为 200 次，收敛过程见图 1 所示。



(a) Single Initial Points　　**(b)** Multiple Initial Points

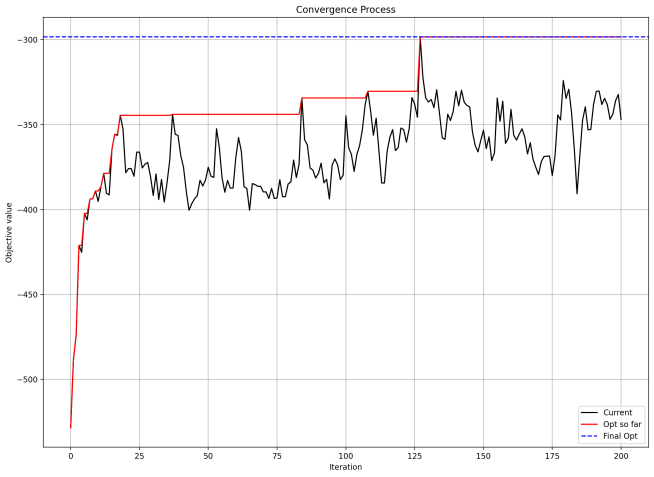
图 1: Random Combin Optim

此方法得到的最优解为 $\boldsymbol{\theta}^* = (2, 4, 12, 6, 3, 7, 1, 11, 9, 8, 5, 10)$，对数似然为 -298.3949。程序日志写入文件 `log/q1_c.log` 中。

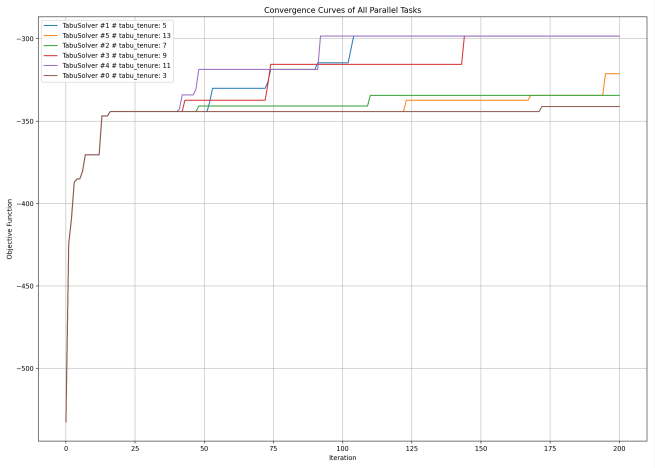
经过测试，单起点优化，很难收敛到最优解，例如本题运行的一个单起点优化的解为 $\boldsymbol{\theta}^* = (9, 11, 1, 7, 6, 3, 10, 5, 8, 12, 4, 2)$，对数似然为 -373.6973。

注 特别注意，本项目代码编写，解的形式均是从 0 开始的记号，故最终结果应是 $\boldsymbol{\theta}_{\text{real}} = \boldsymbol{\theta}_{\text{code}} + 1$。

解答 (d) 使用禁忌搜索求解基因位点顺序，与题 (c) 设置相同的参数，设置不同的禁忌期限，分别为 $3, 5, 7, 9, 11$，得到的结果如图 2 所示。



(a) tabu $\tau = 5$



(b) Multiple tabu $\tau = (3, 5, 7, 9, 11)$

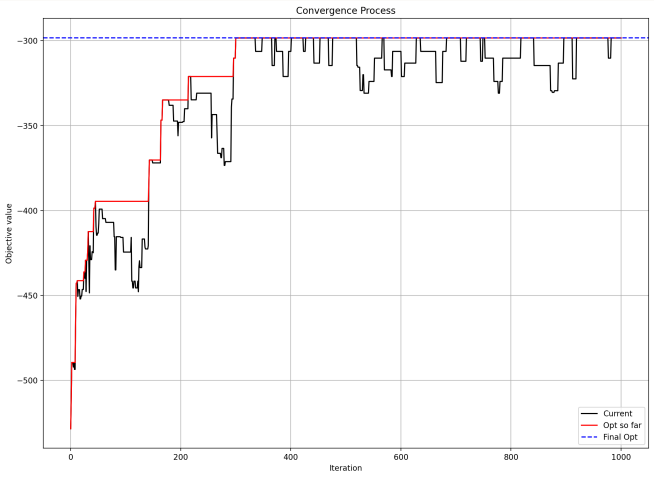
图 2: Tabu Combin Optim

由此同样可解出最优解为 $\boldsymbol{\theta}^* = (10, 5, 8, 9, 11, 1, 7, 3, 6, 12, 4, 2)$（恰好相反，是同一个解），对数似然为 -298.3949。程序日志写入文件 `log/q1_d.log` 中。由图可知，当禁忌期限取 $\tau = 5, 9, 11$ 时，收敛到最优解。

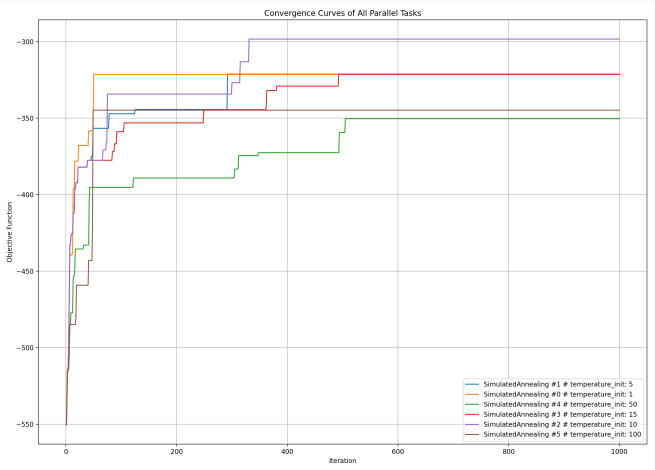
解答 (e) 使用模拟退火，是否更新遵循概率为

$$\min\left\{1,\ \exp\left(\frac{f(\boldsymbol{\theta}^*) - f(\boldsymbol{\theta}^{(t)})}{\tau_j}\right)\right\}$$

其中 $\tau_j = \alpha(\tau_{j-1})$，且持续 $m_j = \beta(m_{j-1})$ 次。而 $\boldsymbol{\theta}^*$ 为 $\mathcal{N}(\boldsymbol{\theta}^{(t)})$ 中最优的解。



(a) Simulated Annealing $\tau_0 = 10$



(b) Simulated Annealing $\tau_0 = (1, 5, 10, 15, 50, 100)$

图 3: Simulated Annealing Combin Optim

这里为了凸显模拟退火的优势，我们选则较小的邻域大小（本项目选择为 2），初始温度选择为 $\tau_0 = 10$，初始持续次数为 $m_0 = 100$。冷却函数设置为 $\alpha(\tau) = 0.9\tau$，而次数递增函数为 $\beta(m) = m + 40$。总迭代次数 1000 次，观察收敛过程如图 3 所示。

同样，得到的最优解为 $\boldsymbol{\theta}^* = (10, 5, 8, 9, 11, 1, 7, 3, 6, 12, 4, 2)$（恰好相反，是同一个解），对数似然为 -298.3949。程序日志写入文件 `log/q1_e.log` 中。

注 这里邻域大小取为 $\#\{\mathcal{N}(\boldsymbol{\theta}^{(t)})\} = 2$，同样能够收敛到最优解。这提高了计算效率，因为取邻域相当于“枚举”，减少邻域大小，则需要计算的可能种类减少，真正的优化了过程。

解答 (f) 设置参数：迭代次数 500 次，族群数量 40 个，采用顺序编码，交叉互换概率为 0.8，变异率为 0.01，更新替代率为 0.3，更新策略采用替代双亲，交叉互换点位为 2，以概率 0.1 保留本族群最优，抽取双亲采用锦标赛方法，分 8 组。结果如图 4 所示。

同样，得到的最优解为 $\boldsymbol{\theta}^* = (10, 5, 8, 9, 11, 1, 7, 3, 6, 12, 4, 2)$（恰好相反，是同一个解），对数似然为 -298.3949。程序日志写入文件 `log/q1_f.log` 中。

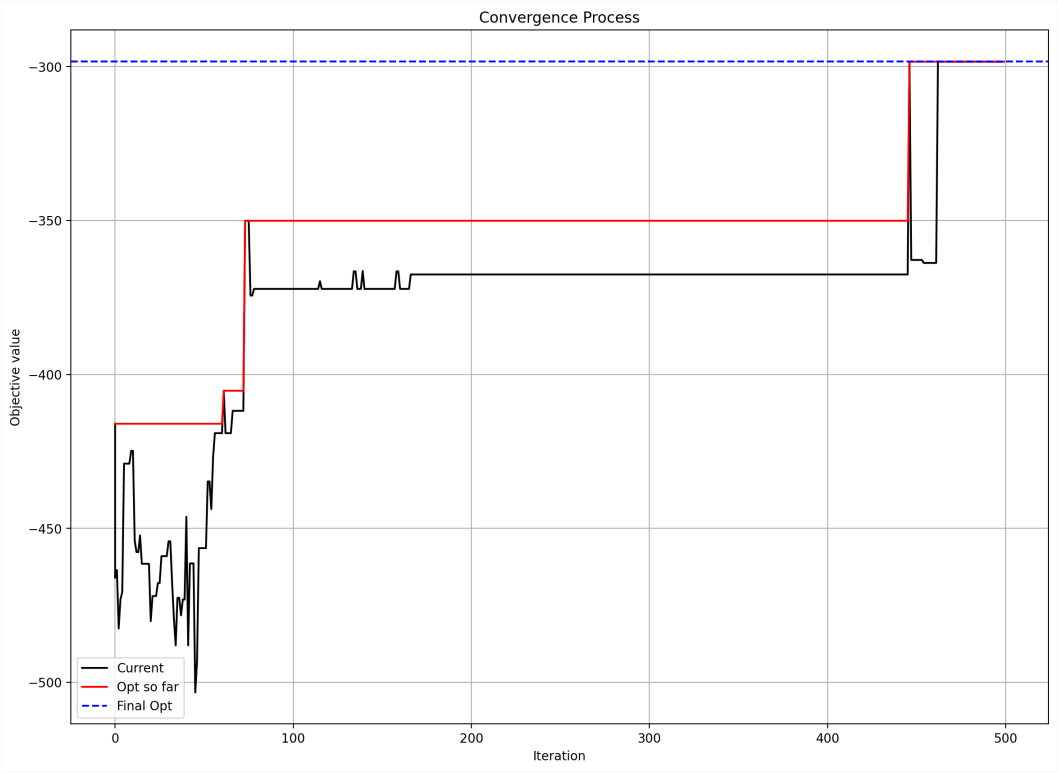


图 4: Genetic Algorithm Method

由图可知，有几处较为平坦的值，可以间接说明遗传算法可能是在寻找某些范式。在测试过程中，经常出现陷入局部最优，或是多样性太小，导致一直无法跳出局部最优，可以尝试多起点，或改变族群进化方式（只需修改代码中 `GeneticAlgorithm._evolution()` 方法，其他常用操作均已实现）。

注 本项目实现的 `ParallelSolver` 类，可以传入任何优化器，并行进行优化。

Exercise 2

附录一给出了一个“蛋白质构象搜索”的案例。这里将问题规模进一步扩展，附件的 `Python` 代码用于在 3D HP 格子模型中求解一个 64-mer 序列的折叠方式，该序列目前已知的最优能量值为 -59。

a 尝试修改参数、运行代码，找到较优的解。

b 本案例中使用的建模方式为相对转向编码。是否可以直接使用绝对方向编码进行建模?（例如 2D 场景中为 $\pm x, \pm y$ 四种选择，3D 场景中为 $\pm x, \pm y, \pm z$ 六种选择）。

c 附件给出的代码实现是否存在不足之处可以进一步优化?

解答 (a) 对于经典遗传算法，因为时间有限，只尝试了部分参数，最终选择：种群大小为 100，迭代次数为 3000。得到结果如图 5 所示，求解的最优函数值为 -29。

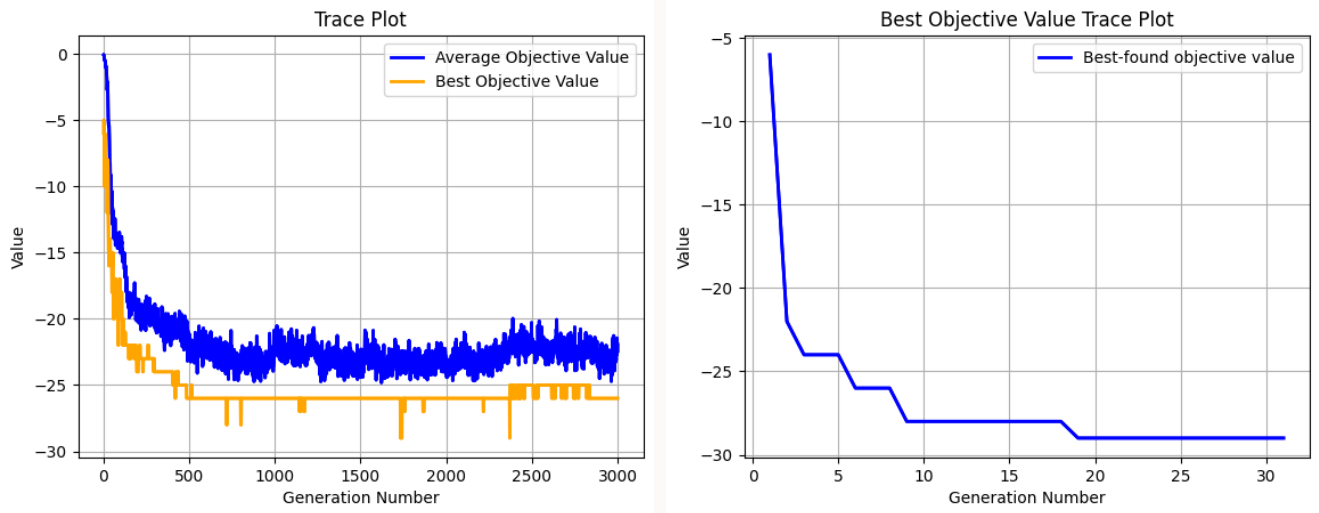


图 5: Simple Genetic Algorithm

对于多种群精英保留遗传算法，选择参数：种群个数为 10，种群大小为 5000，迭代次数为 3000，迁移频率为 30。得到结果如图 6 所示。求解的最优函数值为 -48。

图 6: Multi-Subpopulation Evolution Genetic Algorithm

解答 (b) 可以使用绝对方向编码。但效率不如相对方向编码：

1. 绝对方向会比相对方向多一个选择，即“向后”，复杂度更高；
2. 正是因为绝对方向中会选择“向后”，使得出现重叠，本应该避免重叠的情况，浪费计算资源。

不过对应的，绝对方向编码可省略“解码”部分。

解答 (c) `get_absolute_direction` 函数和 `obj_function` 函数处：

(1) 根据前一个绝对方向和相对方向，计算新的绝对方向：可以使用字典映射的方式，减少 `if-else` 语句。因为 `if-else` 仍然是逐步向下进行的，浪费了许多时间进行比较。

(2) 计算目标函数时，使用两层 `for` 循环寻找 `H-H` 键个数，复杂度为 $O(n^2)$，其中 n 为肽链序列长度。

可以采用“用空间换时间”的思想：维护字典 $D : \mathcal{C} \to \mathcal{I}$，其中 $\mathcal{C}$ 为 H 的坐标集合，$\mathcal{I}$ 为某个或某些 H 在原序列的索引集合（可行性由整数格点有限，且元组可哈希保证：由于序列长度有限，且位于整点处，可将坐标转为元组）。

1. 设肽链序列为 $T = \{H, P\}^n$。存储其中 H 的索引为 $I_H \in \mathbb{N}^p$，其中 p 为 T 中 H 的个数。
2. 对每一个 H，通过 $D^{-1}(i) = (x, y, z) \in \mathcal{C}$ 获取其坐标，其中 $i \in \mathcal{I}$ 为其在原序列的索引。检查其 6 个方向，距离为 1 的点 $\mathcal{N}(H) = \{H_1, \cdots, H_6\}$。
3. 通过 $D(\mathcal{N}(H)) = \{i_1, \cdots, i_6\} \in \mathcal{I}$ 得到其邻点在原序列的位置。若 $|i - i_j| > 1$, $j = 1, \cdots, 6$，则加入目标函数。
4. 遍历 I_H 即可。因为每个符合条件的 `H-H` 键都会被考虑两次，故最后目标函数除以 2。

伪代码如下

```
[INPUT]
  T (List[int]): The sequence of 'HP'
  C (Dict[int, np.ndarray]): Mapping index to coordinates
  D (Dict[Tuple[int], int]): Mapping coordinates to index
  M (np.ndarray): Movements Matrix, defaults to [[x, y, z]], shape = (6, 3)

[OUTPUT]
  f (int): Objective Function Value

[Algorithm]
  1. Index list of 'H'
    I_H <- []
    for i = 1 to len(T):
      if T[i] == 'H':
        I_H.append(i)
  2. Neighborhood function
    def neighborhood(idx: int) -> np.ndarray:
```

```
      curr <- C[idx]  # current coordinates
      return curr + M  # broadcast rule
  3. Check whether adjacent or not
    def isAdjacent(idx: int, cor: tuple) -> bool:
      if abs(idx - D[cor]) > 1:
        return False
      return True
  4. Main: calculate the objective function
    f <- 0
    for idx in I_H:
      neighbors <- neighborhood(idx)
      for cor in neighbors:
        if not isAdjacent(idx, cor):
          f <- f - 1
    return f / 2
```

类似地，在索引所有 H，然后生成字典 D 时，复杂度为 $O(n)$。后续遍历 p 个 H，找出其 6 个邻点，然后比较每个邻点的索引和 H 的索引，总复杂度为 $O(6p)$。注意到字典映射复杂度平均意义下为 $O(1)$。故最后的总时间复杂度为 $O(n+6p) \sim O(n)$。

Exercise 3

模因算法（Memetic Algorithm, MA）是一种结合了全局搜索和局部优化的进化算法。其核心思想是模仿文化演进的过程，即好的想法（“模因”）会在人群中传播和改进。

在算法实现上，它通常融合了遗传算法和局部搜索。具体来说：在**全局探索层面**，像遗传算法一样，通过选择、交叉和变异等操作在整个解空间中进行广泛搜索，以发现有潜力的区域；在**局部利用层面**，对通过全局探索产生的个体（即候选解），应用一个局部搜索方法（如爬山法、模拟退火等），在其邻域内进行深度挖掘和优化，以提高解的精度。**两者交替进行**。

简单来说，模因算法就是“遗传算法 + 局部搜索”，旨在利用前者进行全局漫游，利用后者进行局部爬山，从而实现更高效的寻优。

虽然模因算法融合了全局探索与局部利用的优点，但其中一个关键挑战在于如何有效运用局部搜索。如果对每个个体都进行强度过大的局部搜索，会消耗巨大的**计算资源**，并可能导致种群**过早收敛**（即多样性丧失，算法陷入局部最优），从而使算法性能退化。

请你设计一种策略，用于决定“**何时**”以及“**对哪些个体**”应用局部搜索。请具体阐述你的策略。

解答 设第 t 时刻，族群中的解集为 $\boldsymbol{\theta}^{(t)} = \{\theta^{(t)}\}$，记 f 为适应度函数或为目标函数。记 $\boldsymbol{\Theta}^{(t)} = \{\boldsymbol{\theta}^{(\tau)} : \tau = 1, 2, \cdots, t\}$ 为截止当前的所有族群。不妨设，我们关注的是最大化问题。

“何时”应用局部搜索

(1) 当**最优适应度**有显著提升时，即

$$\max_{\theta\in\boldsymbol{\theta}^{(t)}} f(\theta) > \Gamma(\boldsymbol{\Theta}^{(t-1)})$$

时，开始局部搜索。其中 $\Gamma(\boldsymbol{\Theta}^{(t-1)})$ 是此时的一个阈值函数。例如可以取

$$\Gamma(\boldsymbol{\Theta}^{(t-1)}) = \frac{1}{t-1}\sum_{\tau=1}^{t-1}\max_{\theta\in\boldsymbol{\theta}^{(\tau)}} f(\theta)$$

即当前解相对于之前的平均值而言显著提升。类似地，取分位数点可能更好。不过这过于依赖 f 函数值，缺少了对“解”本身的考虑。

(2) 当**族群与之前**显著不同时，即

$$D(\boldsymbol{\theta}^t, \boldsymbol{\Theta}^{(t-1)}) > \Gamma(\boldsymbol{\Theta}^{(t-1)})$$

简单起见，可取 $\boldsymbol{\Theta}^{(t-1)}$ 为上一个族群 $\boldsymbol{\theta}^{t-1}$，阈值也可以当作一个固定的超参数 Γ。其中 $D(\cdot)$ 定义了 2 个族群间的距离（差异）。例如，采用简单的范数

$$\frac{1}{N}\sum_{\theta^{(t)}\in\boldsymbol{\theta}^{(t)}}\sum_{\theta^{(t-1)}\in\boldsymbol{\theta}^{(t-1)}}\frac{1}{\dim\boldsymbol{\theta}}\left(\sum_{j=1}^{\dim\boldsymbol{\theta}}|\theta_j^{(t)}-\theta_j^{(t-1)}|^p\right)^{\frac{1}{p}} > \Gamma$$

其中 N 为族群大小。对于普通的优化问题，欧式距离 $p=2$ 较为常见；对于 0-1 编码问题，可以使用 $p=1$ 绝对值估计；对于顺序编码的组合优化，则需要定义特殊的“距离”，例如使用机器翻译里常用的 BLEU 分数。

(3) 当**族群自身**多样性足够时，即

$$D(\boldsymbol{\theta}^t) = \binom{N}{2}^{-1}\sum_{\hat{\theta}^{(t)}\neq\tilde{\theta}^{(t)}} d(\hat{\theta}^{(t)},\tilde{\theta}^{(t)}) > \Gamma,\quad \hat{\theta}^{(t)},\tilde{\theta}^{(t)}\in\boldsymbol{\theta}^t$$

或者考察函数值的方差

$$\frac{1}{N}\sum_{\theta^{(t)}}\left[f(\theta^{(t)})-\bar{f}(\theta^{(t)})\right]^2 > \Gamma,\quad \theta^{(t)}\in\boldsymbol{\theta}^t$$

这里的距离函数 $d(\cdot)$ 也类似地可取 L_p, $p=1,2$ 范数，或是 BLEU 分数。相比之前需要比较不同时刻 t 的族群，这里只需考虑当前族群的多样性，计算复杂度大幅降低。

最后，也可以简单地将上面 3 个准则加权组合或者使用投票，混合决定是否开始局部搜索。为了防止一直进行全局搜索，可以手动设置时期 T，要求每迭代 T 次至少局部搜索一次。

“对哪些个体”应用局部搜索

(1) 最简单地，对**最优个体**进行局部搜索。

$$S = \{\theta^{(t)} : f(\theta^{(t)}) \in \{f_{(N)}, f_{(N-1)}, \cdots, f_{(N-m+1)}\}\}$$

其中 $f_{(i)}$ 是次序量，满足 $f_{(1)} \leq \cdots \leq f_{(N)}$。收敛速度可能会更快，但可能会陷入局部最优，减少族群多样性。

(2) 类似遗传算法双亲解的选取，可以采用**锦标赛规则**，选取分组最优的解进行局部搜索。

$$S = \left\{ \arg\max_{\theta^{(t)} \in \boldsymbol{\theta}_g^{(t)}} \ f(\theta^{(t)}) \right\}$$

其中 $\boldsymbol{\theta}_{g_1}^{(t)} \cap \boldsymbol{\theta}_{g_2}^{(t)} = \phi,\ g_1 \neq g_2$ 且 $\cup_g \boldsymbol{\theta}_g^{(t)} = \boldsymbol{\theta}^{(t)}$。

(3) 为不损失多样性，可以对**最“特殊”个体**进行局部搜索。

$$S = \{\theta^{(t)} : s(\theta^{(t)}) \in \{s_{(N)}, s_{(N-1)}, \cdots, s_{(N-m+1)}\}\}, \quad s(\theta^{(t)}) = \frac{1}{N-1} \sum_{\tilde{\theta^t} \neq \theta^t} d(\theta^{(t)}, \tilde{\theta^{(t)}})$$

即前 m 个距离其他点最远的个体，他们可以理解为全局搜索时扩展出的点。

最后，也可以简单地将上面 3 个准则加权组合或者使用投票，混合使用来选择。