CS2045M: 高级数据结构与算法分析 **2025-2026 学年秋冬学期**

Lecture 12: 局部搜索

编写人：吴一航 yhwu_is@zju.edu.cn

12.1 基本思想

本讲将讨论一个与过往风格不太一致，但在解决最优化问题时非常通用的方法，我们称之为局部搜索。在前面的讨论中，我们经常讨论最优化问题，即目标是最大化或最小化某个目标函数——例如 0-1 背包问题，其中目标是最大化背包中的物品权重总和。

之前的讨论通常是考虑一些算法设计策略来解决这一问题，但现在请退回到一无所知的状态，将问题也退回到最原始的状态。假如我们在一座山上，想要走到这座山海拔最低的位置，那么你会怎么办呢? 很显然，最自然的想法就是看看我站的这个点附近能不能再往下走，直到我无法走到更低的位置为止。

当然，很显然的问题是如果这座山是一个山顶一个山谷这样连绵起伏的，我们很有可能走到的只是一个局部的山谷，也就是局部的最低点，并非整座山脉的最低点。所以这样的搜索最优解的方式非常自然地称为**局部搜索（local search）**——因为找最优解的方式是看看当前位置的附近位置能不能进一步优化解，直到附近没有最优解时才会停止，并且最终搜索到的结果是一个局部最优解。PPT 第 2 页给出了一个图示，展示了类似的思想。

12.1.1 自然科学中的局部搜索

相信局部搜索的想法会很容易让我们联想到物理学中的势能等概念。的确如此，在物理学中的局部搜索是非常常见的。我们知道，大自然是 “懒惰” 的，它总是更加倾向于采纳那些能量更低的状态（最低势能），或者受到影响更小的情形（最小作用量）。因此，某个（经典物理）问题的解往往是某个局部最小值的情形。基于此，局部搜索的算法的物理背景往往可以十分清晰。比方说，模拟退火就是一个非常简单的例子（之后会展开讨论）。一个非常简单的观察来源于 Euler-Lagrange 方程的推导，即一个泛函的极值通过一个偏微分方程来描述。而这个泛函的极值本身的求法，由直接的搜索也能得出。这样的思路在计算化学中相当常见，比如说非常经典的密度泛函理论（DFT），其中可以选择许多组不同的 “基” 作为计算的前提，例如 6-31G** 等，这是一种非常常用的加速计算的方法，它比从头计算（ab initio）方法要快上许多。

在生物学中，所谓的物竞天择就是某种更广义的局部搜索。基于 Darwin 的进化论和较为晚近的基因理论，人们发展出了进化计算（evolutionary computation）来解决一些问题。考虑以下对应：

生物学	计算机
环境	问题
个体	可行解
适应性	开销

这是进化计算的基本隐喻。接下来，考虑初始化一个种群（population），然后通过其中的一次选择（亲代选择，parent selection）选出亲代，然后通过基因型的重组（recombination）和变异（mutation）来产生后代（offspring），再通过子代选择（survivor selection）使得后代重新构成种群。通过这样的一代一代的循环，可以找到一个“适应环境”的种群结果，这就是遗传算法的整体思路。

接下来的问题是：如何做选择，以及如何做变异。在这个过程中，有很多比较复杂的构造。实际上，第一性的问题是，如何表达某个个体的基因型。这将其分成了许多种类，例如所谓的遗传算法（genetic algorithm）、进化策略（evolution strategy）、进化规划（evolutionary programming）和遗传规划（genetic programming）等等，这些区分是纯粹历史性的，实际上也不是非常重要，基本上，用字符串、树、向量、有限状态机等等，都是可以的。

从生物学的角度看，亲代选择（或称配偶选择）是根据个体的性质判断某个个体能否找到配偶、以及能否繁衍后代。典型的亲代选择都是概率性的，尤其是在能否找到配偶方面。而子代选择则也被称为替换策略或环境选择，它反映的是某个个体能够存活到发生繁衍的时候，它不应该在出生之后立刻死亡，等等。当然，这样的事情也是概率性的。

在进化计算的设计中，这样的选择往往是重要的。一个非常经典的笑话就来源于评估函数设计的不当：如果要让一只电子狼去在尽可能短的时间内避开障碍，追上一只仿生人梦到的电子羊，那么一只最厉害的电子狼可能是直接撞向障碍物的一只。它花费了最短的时间，抵达了狼生的终点。哪怕它没有获得抓羊的福利，减小的时间开销也足以让其成为狼群中的佼佼者，这是一个非常错误的结果，但也反映了某种有意思的现实情境。

这样的算法最常用的地方在于软件工程中的模糊测试（fuzzing），即通过一代代的测试样本变异来找出可能发生漏洞的地方，这往往也就是启发式的，比方说输入是一个结构化的东西（XML，等等），然后通过某些变异规则使得它能够大概率成为合法的测试样本，然后供分词器等完成测试，进而发现代码中可能存在的问题。

举个例子，我们希望通过进化计算来解决 0-1 背包问题。这时，候选解的表示方式非常自然，可以表示为一个二进制字符串（这就是可行解的基因型，表现型就是最后的物品选择）。有两种方法来解读一个字符串，第一个是，如果一个字符是 1，就表示它包含这个物体；反之则不包含这个物体。第二个是，如果它是 1，就表示它如果不会超过容量限制，则包含这个物体。很显然，第二个比第一个的利用率更高，这是因为其中没有 “即死” 的个体，即那些本身重量超过背包容量的个体，这样的个体在子代选择的时候就让他直接丢弃。但不同的 “基因型” 可能会对应相同的 “表现型”，例如两个基因型在前几个物品上选择一致，并且此时以及塞满了，那么后面的 01 串就不会影响最终物品的选择。然后，通过现有的最优解判断亲代选择的几率，即更接近较优解的更容易留下后代。子代通过某个给定的继承概率从

亲代中继承某个对应值。当然，这是一个 “无性生殖” 的例子，也就是说，没有配偶选择的过程。实际上，有研究表明，有性生殖更能提升种群的基因多样性，进而有利于进一步的搜索。

12.1.2 全局优化的复杂度界

进行到这里，我们已经看到了局部搜索作为一个优化方法的合理性——它是一种来源于自然界的合理性。但是有的读者可能还有另一个疑惑，即为什么明知道局部搜索很可能只能带来局部最优解，在很差的情况下可能与真正的最优解相去甚远，但还是选择它呢？也许当你要找一座山脉最低点的时候你可以爬上最近的山峰先看看，然后决定往哪里走最好。但有时候山脉可能绵延数千里，你爬上了一个山峰也无法断定哪里是全局最优。当然这只是一个比喻，在解决一般优化问题时，有时候可行解的个数可能太多（一般是指数级别），因此穷举或者回溯找到全局最优解的方式非常耗时，而很多优化问题又是 NP 困难的，暂时也无法给出高效算法，并且近似算法可能需要精巧的设计和精妙的证明，或者有些问题本身就无法获得很好的近似比，那么此时很自然地就会求助于使用局部搜索这一非常自然的方式。

这里来看一个优化问题的例子来印证上面的说法。考虑问题 $\min\limits_{\boldsymbol{x}\in\mathbb{B}^n} f(\boldsymbol{x})$，其中可行解的集合 $\mathbb{B}^n$ 是 $\mathbb{R}^n$ 中的一个 n 维盒子 n-dimensional box）

$$\mathbb{B}^n = \{\boldsymbol{x} \in \mathbb{R}^n \mid 0 \leqslant \boldsymbol{x}^{(i)} \leqslant 1, i = 1, \ldots, n\}$$

假如我们对 f 一无所知，那么显然是无法求解的。现在加入一个限制。首先设使用 ℓ_∞ 对 $\mathbb{R}^n$ 中的距离进行度量（即切比雪夫距离）

$$\|\boldsymbol{x}\|_\infty = \max_{1\leqslant i\leqslant n} |\boldsymbol{x}^{(i)}|$$

并假设目标函数 $f : \mathbb{R}^n \mapsto \mathbb{R}$ 在 $\mathbb{B}^n$ 上满足利普希茨连续（Lipschitz continuous）条件，即对于某个常数 L，

$$|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leqslant L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_\infty, \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{B}^n,$$

其中，L 称为（关于 $\|\cdot\|_\infty$ 的）Lipschitz 常数。那么问题类 $\mathcal{P} = (\Sigma, \mathcal{O}, \mathcal{T}_\varepsilon)$ 描述如下，

模型Σ	$\min\limits_{x\in\mathbb{B}^n} f$ 其中f 在$\mathbb{B}^n$ 上是ℓ_∞Lipschitz 连续的
Oracle $\mathcal{O}$	与优化目标相关的唯一局部信息是给定点能返回函数值
停止准则$\mathcal{T}_\varepsilon$	$\overline{\boldsymbol{x}} \in \mathbb{B}^n, \quad f(\boldsymbol{x}) - f^* \leqslant \varepsilon$

有一个很朴素的想法，以二维形式为例，$\mathbb{B}^2$ 是一个边长为 1 的正方形；将其边长均分为 p 份，将正方形划分成网格，得到了 $(p+1)^2$ 个交点（如果是 n 维，则为 $(p+1)^n$），接下来逐个计算这些点的函数值，选择其中最小的一个值作为 f 的最小值。这种方法被称作均匀网格法（uniform grid method），算法如下：

1. 生成 $(p+1)^n$ 个点：$\boldsymbol{x}_\alpha = \left(\dfrac{i_1}{2p}, \dfrac{i_2}{2p}, \ldots, \dfrac{i_n}{2p}\right)$，其中 $\alpha \equiv (i_1, i_2, \ldots, i_n) \in \{0, 1, \ldots, p\}^n$；
2. 在所有的点 x_α 中找到使目标函数达到最小值的点 $\overline{\boldsymbol{x}}$；

3. 算法输出 $(\overline{\boldsymbol{x}}, f(\overline{\boldsymbol{x}}))$。

这是一种累计信息对测试点顺序没有任何影响的零阶优化算法。直觉上，当 p 足够大时，一定能找到一个足够逼近 f 最小值的点：

定理 12.1 *设 f^* 是 f 的全局最优值，$\overline{\boldsymbol{x}}$ 是均匀网格法的输出，那么*

$$f(\overline{\boldsymbol{x}}) - f^* \leqslant \frac{L}{2p}.$$

证明: 设 $\boldsymbol{x}_*$ 是最小值点，那么存在 n 维坐标 $(i_1, i_2, \ldots, i_n)$ 使得

$$\boldsymbol{x}_1 \equiv \boldsymbol{x}_{(i_1,i_2,\ldots,i_n)} \leqslant \boldsymbol{x}_* \leqslant \boldsymbol{x}_{(i_1+1,i_2+1,\ldots,i_n+1)} \equiv \boldsymbol{x}_2,$$

即 $\boldsymbol{x}_*$ 一定落在某个小立方体之内。注意到对于 $i = 1, \ldots, n$ ，有 $\boldsymbol{x}_2^{(i)} - \boldsymbol{x}_1^{(i)} = \dfrac{1}{p}$，且 $\boldsymbol{x}_1^{(i)} \leqslant \boldsymbol{x}_*^{(i)} \leqslant \boldsymbol{x}_2^{(i)}$。定义 $\hat{\boldsymbol{x}} = (\boldsymbol{x}_1 + \boldsymbol{x}_2)/2$，

$$\tilde{\boldsymbol{x}} = \begin{cases} \boldsymbol{x}_2^{(i)}, & \text{if } \boldsymbol{x}_*^{(i)} \geqslant \hat{\boldsymbol{x}}^{(i)} \\ \boldsymbol{x}_1^{(i)} & \text{otherwise} \end{cases}$$

显然有 $\left|\tilde{\boldsymbol{x}}^{(i)} - \boldsymbol{x}_*^{(i)}\right| \leqslant \dfrac{1}{2p}$, $i = 1, \ldots, n$，即 $\boldsymbol{x}_*$ 落在的小正方体的 2^n 个顶点中，一定有一个顶点，其到 $\boldsymbol{x}_*$ 的 ℓ_∞ 距离不大于 $\dfrac{1}{2p}$)；因此，

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}_*\|_\infty = \max_{1 \leqslant i \leqslant n} \left|\tilde{\boldsymbol{x}}^{(i)} - \boldsymbol{x}_*^{(i)}\right| \leqslant \frac{1}{2p}$$

$$f(\overline{\boldsymbol{x}}) - f(\boldsymbol{x}_*) \leqslant f(\tilde{\boldsymbol{x}}) - f(\boldsymbol{x}_*) \leqslant L \|\tilde{\boldsymbol{x}} - \boldsymbol{x}_*\|_\infty \leqslant \frac{L}{2p}$$

■

通常而言，算法需要 “找到问题类 $\mathcal{P}$ 有精度 $\varepsilon > 0$ 的近似解，现将其定义为：寻找一个 $\overline{\boldsymbol{x}} \in \mathbb{B}^n$，使得 $f(\overline{\boldsymbol{x}}) - f^* \leqslant \varepsilon$。有如下结论：

定理 12.2 *对于问题类 $\mathcal{P}$ 的方法 $\mathcal{G}$，其分析复杂度最多为*

$$\mathcal{A}(\mathcal{G}) = \left(\left\lfloor \frac{L}{2\varepsilon} \right\rfloor + 2\right)^n$$

因为只需要在上面的算法中取 $p = \left\lfloor \dfrac{L}{2\varepsilon} \right\rfloor + 1$ 即可保证

$$f(\overline{\boldsymbol{x}}) - f^* \leqslant \frac{L}{2p} \leqslant \varepsilon,$$

而且我们需要在 $(p+1)^n$ 个点调用 Oracle（即获取当前位置的函数值）。所以 $\mathcal{A}(\mathcal{G})$ 就确定了问题类的复杂度的上界。可以看出，除了精度 ε 外，均匀网格法的分析复杂度与划分点数 p 与维数 n 有关。

但目前还存在一些问题。我们没有证明方法 $\mathcal{G}(p)$ 的这个上界能否取到，也许事实上其性能会更好；另外，还可能存在比 $\mathcal{G}(p)$ 更好的算法。要回答这个问题，就需要计算 $\mathcal{P}$ 的复杂度下界，该下界满足

- 基于黑箱概念；
- 对于所有合理的迭代算法都有效；
- 采用对抗 Oracle（Resisting Oracle）思想。

一个 "对抗 Oracle" 思想是，为每个具体方法创造一个尽可能最差的问题。也就是说，对于每一个 x_k，返回一个尽可能差的 $\mathcal{O}(x_k)$。但是该返回值必须与之前的答案相符，也与问题类 $\mathcal{P}$ 的描述相符。基于这一思想有如下定理：

定理 12.3 对于 $\varepsilon < \dfrac{1}{2}L$，$\mathcal{P}$ 的分析复杂度下界是 $\left(\left\lfloor \dfrac{L}{2\varepsilon} \right\rfloor\right)^n$。

证明: 令 $p = \left(\left\lfloor \dfrac{L}{2\varepsilon} \right\rfloor\right)^n \geqslant 1$，假设存在一个方法，对于 $\mathcal{P}$ 中的任意问题，都只需要 $N < p^n$ 次对 Oracle 的调用；根据鸽巢原理，一定有一个小立方体中没有测试点，故可以从这里下手构造函数。

将该算法应用于下面的对抗策略：Oracle 在任意测试点处都返回 $f(\boldsymbol{x}) = 0$。因此，该方法只能求出 $\overline{\boldsymbol{x}} \in \mathbb{B}^n$，$f(\overline{\boldsymbol{x}}) = 0$。然而，注意到存在 $\hat{\boldsymbol{x}} \in \mathbb{B}^n$，使得

$$\hat{\boldsymbol{x}} + \frac{1}{p}\boldsymbol{e} \in \mathbb{B}^n, \qquad \boldsymbol{e} \in (1, \ldots, 1)^{\mathrm{T}} \in \mathbb{R}^n$$

且在立方体 $\mathbb{B} = \left\{ \boldsymbol{x} \mid \hat{\boldsymbol{x}} \leqslant \boldsymbol{x} \leqslant \hat{\boldsymbol{x}} + \dfrac{1}{p}\boldsymbol{e} \right\}$ 内没有任何测试点。令 $\boldsymbol{x}_* = \hat{\boldsymbol{x}} + \dfrac{1}{2p}\boldsymbol{e}$。考虑函数

$$f_p(\boldsymbol{x}) = \min\{0, L\|\boldsymbol{x} - \boldsymbol{x}_*\|_\infty - \varepsilon\}$$

易知 f_p 满足 ℓ_∞-Lipschitz 连续，其 Lipschitz 常数为 L，全局最优为 $-\varepsilon$。然而，f_p 仅在立方体 $\mathbb{B}' = \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}_*\|_\infty \leqslant \dfrac{\varepsilon}{L}\}$ 内不为 0，由于 $2p \leqslant L/\varepsilon$，因此 $\mathbb{B}' \subseteq \mathbb{B}$。

而在我们的方法中，对于所有的测试点，$f_p(\boldsymbol{x})$ 都会返回 0。由于我们方法的精度为 ε，可以得出结论：如果 Oracle 的调用次数小于 p^n，那么结果的精度不可能小于 ε。定理得证。 ■

现在可以对 $\mathcal{G}$ 进行总结

$$\mathcal{G}: \left(\left\lfloor \frac{L}{2\varepsilon} \right\rfloor + 2\right)^n, \qquad \text{下界} \left(\left\lfloor \frac{L}{2\varepsilon} \right\rfloor\right)^n$$

如果 $\varepsilon \leqslant O\left(\dfrac{L}{n}\right)$，那么上界和下界是一致的；因此对于问题类 $\mathcal{P}$，方法 $\mathcal{G}(p)$ 是最优的。因此如果执意要对一个函数求解全局最优（在如上条件下），那么时间复杂度的底线也将是指数级别的。因此会求助于局部搜索，以更快速、更精妙的方式找到一个局部最优解。

12.2 第一个例子：顶点覆盖问题

12.2.1 局部搜索的范式

在讨论了局部搜索的大致思想后，下面开始正式的讨论。在正式讨论开始之前，先将前面的大致思想形式化。首先分别讨论局部搜索中的 “局部” 和 “搜索” 两个关键词：

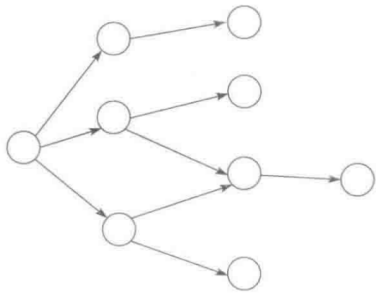
1. 为了定义 “局部”，实际上就是定义 “邻居（neighborhood）” 这一概念。假设 S 是一个可行解，所谓 S 的邻居就是一个集合

   $$N(S) = \{S' \mid S' \text{ 是 } S \text{ 经过一个小的更改之后得到的解}\},$$

   这里 “小的更改” 需要算法设计者自己根据问题的特点定义，例如背包问题可以设置为进行一个物品的替换，旅行商问题可能是交换两个顶点的途径顺序等等；于是所谓的解 S 的局部也就是 S 的邻居集合 $N(S)$；

2. 所谓的搜索就是从任意一个可行解开始，不断在当前所处的解的局部（也就是邻居）中选择一个最优解，直到无法在局部找到更优解为止，这样就能得到一个局部最优解。

可以对局部搜索做一个可视化的理解，展示为下图中的有向图上的一个行走。图中每条有向边表示一个更优的局部移动，并且显然这些局部移动间不会有环，所以下图是一个有向无环图。图中没有出边的节点（吸收点）表示局部最优。局部搜索算法事实上就是在图中不断地沿着出边进行行走，直到进入一个吸收点为止。



12.2.2 顶点覆盖问题

在讲完了抽象的范式之后，马上来看一个具体的基础的例子进行应用。考虑顶点覆盖问题：给定一个无向图 $G = (V, E)$，找到一个最小的顶点集合 $S \subseteq V$，使得对于每一条边 $(u, v) \in E$，至少有一个顶点在 S 中。

这个问题是一个 NP 困难问题，这是在 NP 问题一讲中给出的结果，也就是说暂时没有找到一个多项式时间的算法来解决这个问题。在给出局部搜索解前先简单回顾一下近似算法。这里考虑一个简单的

贪心算法（记为算法 A）：任意一条边 (u,v)，然后将 u 和 v 同时加入到 C 中，然后把 u 和 v 所在的所有边全部移除，下一轮继续在剩下的边中随机选择一条重复上述操作，直到所有的边都被删除为止。

定理 12.4 *上述顶点覆盖问题的贪心算法 A 是一个 2-近似算法。*

证明: 设 S 为算法 A 运行过程中选出的所有边的集合。为了覆盖 S 中的边，任意一个顶点覆盖（特别是最优覆盖 C^*）都必须至少包含 S 中每条边的一个端点。又因为每次把边选进 S 后就会把所有与选入的边的端点相关联的所有边，所以 S 中不存在两条边具有共同的端点，所以有

$$|C^*| \geqslant |S|.$$

设算法 A 选出的顶点覆盖为 C，因为一条边的两个端点都被选入到 C 中，并且没有两条边共同的端点，所以

$$|C| = 2|S|.$$

于是有

$$|C| = 2|S| \leqslant 2|C^*|,$$

■

再来回顾一下上述证明过程。事实上与上一讲中见到的问题类似，我们需要在不知道最优顶点覆盖的规模到底是多少的情况下证明算法 A 所返回顶点覆盖的规模至多为最优顶点覆盖规模的 2 倍。为此，我们巧妙地利用了最优顶点覆盖规模的一个下界（就是贪心算法选出的边数），从而使证明过程不牵涉最优顶点覆盖的实际规模。

接下来开始介绍局部搜索算法。显然首先需要定义这一问题中的邻居是什么。在顶点覆盖问题中，目标是找到一个最小的顶点覆盖，然后需要每一步在邻居中找一个更优解，所以不妨让邻居就是比当前解少一个顶点的解。这样就可以定义局部搜索算法了：选择的初始解就是图 G 的所有顶点，即 $S = V$，然后每次从 S 中随机选择一个顶点 v，然后将 v 从 S 中移除，检查新的解 $S - \{v\}$ 是否是一个顶点覆盖，如果是则继续后续删除操作，否则就得到了一个局部最优解。

称这一算法为 “梯度下降法”。相信读者对这一词汇并不陌生，它是求解一个函数的最小值时方法，常用于机器学习、优化理论中。所谓的梯度下降就是因为梯度反映的是函数下降最快的方向，因此沿着梯度方向寻找局部更优的解是合理的。在这里借用了其中的思想，这里的梯度下降其实就是任选一个点从现有解中删除。

PPT 第 6 页给出了 “梯度下降法” 合适与不合适的例子。对于第一种情况而言，实际上就对应于局部最优与全局最优完全等价的情况；第二种则表明局部最优（选择外面一圈的所有点）相比于全局最优（选择中心点）可以任意差；第三种则是有很多种情况，可以通过不同的下降选择找到不同的局部最优解。由此也看出局部搜索与近似算法的一个很重要的区别，即局部搜索得到的解可能是没有任何近似比保证的——局部搜索只是一种启发式的算法。所以接下来希望对这一简单的方法做一些改进：我们希望有时候我们能跳出一些很差的局部最优解，换条路继续搜索。

12.2.3 Metropolis 算法与模拟退火

这种改进的方法由 Metropolis，Rosenbluth 和 Teller 于 1953 年提出，被称为 Metropolis 算法。他们希望利用统计力学中的原理模拟物理系统的行为。在统计物理中有一个假设，当一个系统的能量为 E 时，它出现的概率为 $e^{-E/kT}$，其中 $T>0$ 是温度，k 是玻尔兹曼常数。这个假设被称为 Boltzmann 分布。显然的，当 T 固定时，能量越低的状态出现的概率越大，因此一个物理系统也有更大的概率处于能量低的状态。然后考虑温度 T 的影响，当 T 很大时，根据指数函数的特点，不同能量对应的概率其实差别可能不是很大；但 T 较低的时候，不同的能量对应的概率差别就会很大。

基于玻尔兹曼分布，可以将 Metropolis 算法描述如下：

1. 初始化：随机选择一个可行解 S，设 S 的能量为 $E(S)$，并确定一个温度 T；
2. 不断进行如下步骤：
   (a) 随机选择 S 的一个邻居 S'（可以按均匀分布随机选择）；
   (b) 如果 $E(S')\leqslant E(S)$，则接受 S' 作为新的解；
   (c) 如果 $E(S')>E(S)$，则以概率 $e^{-(E(S')-E(S))/kT}$ 接受 S' 作为新的解；如果接受了则更新解，继续下一轮迭代，否则保持原解 S，继续迭代。

直观地说，Metropolis 算法就是在当前解 S 的邻居中随机选择一个解 S'，如果 S' 的能量更低则接受，否则以一定概率接受一个更差的解——这就使得我们有可能跳出一个局部最优解。当然有一个注意的问题是，上面没有写算法的停止点，实际上进行到一个满意的结果中断算法即可。Metropolis 等人证明了他们的算法具有如下性质（证明不属于这门课的范畴，涉及一些简单的随机过程）：

定理 12.5 *设 $Z=\sum_S e^{-E(S)/kT}$。对于任一状态 S，记 $f_S(t)$ 为在 t 轮迭代中选到了状态 S 的比例，则当 $t\to\infty$ 时，$f_S(t)$ 将以概率 1 收敛于 $e^{-E(S)/kT}/Z$。*

这一结论表明，Metropolis 算法在足够长的时间后，在每个状态上停留的时间比例将和玻尔兹曼分布近似，也就是说这一算法成功模拟了玻尔兹曼分布，有更大的可能停留在低能量状态，即有更大的概率获得很好的解。

PPT 第 7 页给出了针对顶点覆盖问题的 Metropolis 算法。对于 PPT 第 6 页的 case 1，有了 Metropolis 算法，就有可能跳出局部最优解，找到全局最优解。然而如果看 case 0，就会发现当进行到没剩下几个点的时候，这时因为是均匀选取下一个状态，那么有非常大的概率会抽到一个比当前解差的解，然后又有一定概率接受这个解，这样就会偏离最优解（一个点都不选），甚至陷入一种在几个解之间来回跳但就是找不到最优解的境地，但是不用 Metropolis 算法都能解出最优解，现在反而解不出了，这是很难让人接受的，因此还需要进一步优化 Metropolis 算法。

事实上还有一个关键的因素没有考虑：Metropolis 算法的一个重要参数是温度 T。之前也分析了，当温度很高的时候，不同能量对应的概率差别不大，因此高温非常适合于跳出局部最优解；而当温度很低的

时候，不同能量对应的概率差别很大，这时候就有更大的概率进入好的解。因此我们希望在开始的时候温度很高，然后逐渐降低温度，这样就能在开始的时候跳到一个比较好的局部，然后在降温后降到能到的最低点。这就是模拟退火（Simulated Annealing）的思想，它的思想来源于模拟降温结晶过程。有一个有趣的问题，就是为什么不一开始就选择低温，因为前面的定理表明当 $t \to \infty$ 时，Metropolis 算法的解会收敛到玻尔兹曼分布，这样就能大概率找到很好的解。但在实际中发现温度较低的时候这一过程会非常漫长，所以是不切实际的，因此采取先用高温偏向于好的局部，然后降温找到最优解这样的过程。当然模拟退火也不能保证一定能找到最优解，毕竟一切选择都是概率性的。

12.3 旅行商问题

在顶点覆盖中，直观感觉是局部搜索就是一种启发式的、没有效果保证的算法，那么接下来的几个例子会给出一些有保证的局部搜索算法：我们会看到局部最优解就是想要的解的情况，还会看到局部最优解是有可证明的近似比的解。接下来要看的问题是老熟人：旅行商问题，当然这里是建立在上一讲的 2-近似算法的基础上做进一步的局部搜索改进，所以自然也是一个近似比有保证的算法。

这已经是第三次遇见旅行商问题了，这次将从局部搜索的角度提供一种解决方案。首先来回顾一下旅行商问题的定义：给定一个无向完全图 $G = (V, E)$，每条边 $(u, v) \in E$ 有一个权重 $w(u, v)$，希望找到一个最短的回路，使得每个顶点恰好被访问一次。这个问题是一个 NP 困难问题，暂时没有找到一个多项式时间的算法来解决这个问题。

12.3.1 一个准确的动态规划算法

在正式讨论局部搜索算法前，我希望在这里先补全一个之前遗漏的关于旅行商问题的讨论。事实上对于这类 NP 困难问题，之前提到过，在输入不算太大的时候也可以接受指数时间的精确算法。回顾 0-1 背包问题和最小化工时调度问题，我们都提到了动态规划算法，这是一个可以支持指数级别精确算法的比较精巧算法设计策略（回溯过于暴力了），所以也可以尝试用动态规划来解决旅行商问题。

动态规划的关键在于找出最优子结构。仍然用一个思维实验来开始讨论，假如上帝给了你一个解，那么上帝自己一定是一个一个点慢慢选出来的，但你只能从最后选的那个点观察一些选点的原则。假设给所有顶点编号 $1, \ldots, n$，然后我们的目标是找一条最短的经过每个点恰好一次从 1 回到 1（从任意点到自己都可以，但其实是等价的，所以用 1 举例子说明）的哈密顿回路。那么这个路径的最后一跳应当是从某个点 $j \neq 1$ 到 1，那么问题就转换为了找最优的 j 作为最后一跳，即

$$\text{最优路线的成本} = \min_{j=2,\ldots,n} \text{访问每个顶点的无环 } 1 \to j \text{ 路径的最低成本} + c_{j1},$$

其中用 c_{ij} 表示从 i 到 j 的成本。

于是问题变成了怎么找到从 1 出发到任意顶点 $j \neq 1$ 的最短的经过每个顶点恰好一次的路径。如果能解决这一问题，那么再代入上面的表达式即可求出旅行商的最短路径。这里就要用最短路问题自带的最

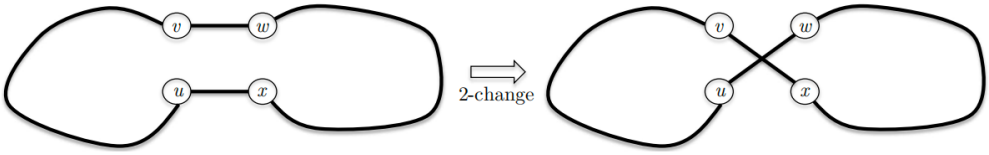
优子结构来设计动态规划算法了（在 Bellman-Ford 和 Floyd-Warshall 算法中已经实践过）：即最短路径的某一段也一定是对应的起点终点的最短路径。因此如果要求从 1 到 j 的最短路径，可以考虑最后一跳的选取，假设最短路径最后一跳是从 k 到 j，那么可以将问题分解为从 1 到 k 的最短路径加上从 k 到 j 的成本，当然我们并不能直接知道哪个 k 最好，所以还需要遍历所有的 k，这样就决定了最后一跳。然后递归解决倒数第二跳，以此类推。不难发现，这样的动态规划算法的子问题可以表达为 $C_{S,j}$，表示从 1 到 $j \in S$ 经过 S 中所有点恰好一次的最短路径。这样就可以写出动态规划的递推式：

$$C_{S,j} = \min_{k \in S, k \neq 1, j} C_{S-\{j\},k} + c_{kj}.$$

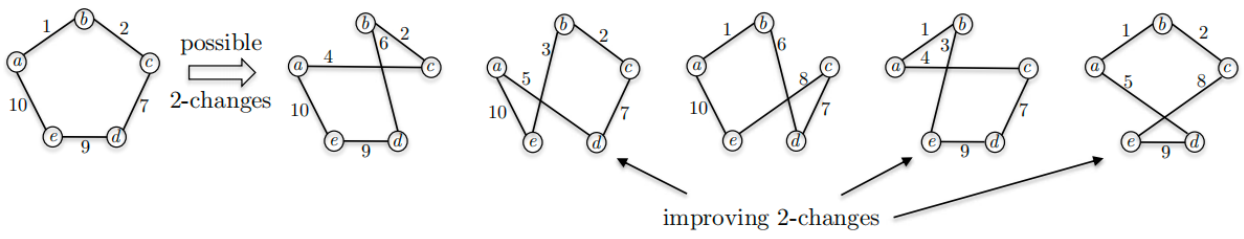
接下来就是要写一个迭代的程序填满子问题的表格，根据递推式的特点，只要按照 S 的大小从小到大依次填表，就能保证解每个问题的时候，需要的子问题都是已经填满了的。这样就可以得到最终的最优解。这一算法称为 Held-Karp 算法，复杂度分析也很简单：有 $n \cdot 2^n$ 个子问题，每个问题的计算复杂度是 $O(n)$（求 min 需要遍历所有可能），所以总的复杂度是 $O(n^2 \cdot 2^n)$（当然不要忘了算法最后一步还需要回归到 1 到 1 的环路，这里需要遍历 $n-1$ 种最后一跳的可能，是线性时间的）。

12.3.2 局部搜索算法

接下来回归正题，即设计一个局部搜索算法。在上一讲已经介绍了旅行商问题的一个 2-近似算法，可以从这个 2-近似算法开始，然后用局部搜索来改进得到的解。为了设计局部搜索算法，首先需要定义邻居关系，对于一条路线来说，怎样的邻居关系是自然的呢？首先有一个基本的观察，对于两条长度为 n 的不同的哈密顿回路，显然它们之间能共享的最大边数为 $n-2$。因此可以自然地定义两条哈密顿回路 S 和 S' 之间的邻居关系为：S 和 S' 之间的边有两条不一样，如下图所示：



称这样的变换为 2-变换（2-change）。于是接下来就可以在 2-近似的解的基础上做 2-变换进一步改进解（即检查改变后的受到影响的边的长度是否减小）。当然要注意，上图中二变换不能做成 v 和 u 相连，w 和 x 相连，这样变换之后就不是哈密顿回路了。下图给出了一个真实的例子，读者可以参照理解：



12.4 Hopfield 神经网络的稳定构型

Hopfield 神经网络是一种全连接的反馈神经网络，于 1982 年由 J.Hopfield 教授提出（他也因此获得 2024 年诺贝尔物理学奖）。关于 Hopfield 神经网络的介绍并非本门课程要求的内容，感兴趣的读者利用互联网自行搜索即可。与优化问题相关的一点是，Hopfield 神经网络有一个能量函数，如果能将能量函数与一个优化问题绑定，那么 Hopfield 神经网络就可以用来解决这个优化问题，例如旅行商问题。

当然这并不是讨论的重点，这里希望讨论的是 Hopfield 神经网络的稳定构型（stable configurations）。为了研究这一问题，需要首先进行一些定义：

1. Hopfield 神经网络可以抽象为一个无向图 $G = (V, E)$，其中 V 是神经元的集合，E 是神经元之间的连接关系，并且每条边 e 都有一个权重 w_e，这可能是正数或负数；
2. Hopfield 神经网络的一个构型（configuration）是指对网络中每个神经元（即图的顶点）的状态的一个赋值，赋值可能为 1 或 -1，记顶点 u 的状态为 s_u；
3. 如果对于边 $e = (u, v)$ 有 $w_e > 0$，则希望 u 和 v 具有相反的状态；如果 $w_e < 0$，则希望 u 和 v 具有相同的状态；综合而言，我们希望
$$w_e s_u s_v < 0.$$
4. 将满足上述条件的边 e 称为 “好的（good）”，否则称为 “坏的（bad）”；
5. 称一个点 u 是 “满意的（satisfied）”，当且仅当 u 所在的边中，好边的权重绝对值之和大于等于坏边的权重绝对值之和，即
$$\sum_{v:e=(u,v)\in E} w_e s_u s_v \leqslant 0,$$
反之，如果 u 不满足这一条件，称 u 是 “不满意的（unsatisfied）”；
6. 最后，称一个构型是 “稳定的（stable）”，当且仅当所有的点都是满意的。

这些定义或许太多太杂，但如果读者看到 PPT 第 10 页的例子，应当就会熟悉这些定义。现在的问题是，给定一个 Hopfield 神经网络，是否存在一个稳定构型？如果存在，如何找到这个稳定构型?

当然有的读者可能感兴趣这一问题的研究背景究竟是什么，当然这超出了我们的讨论范围，但简而言之就是 Hopfield 神经网络这样的反馈神经网络在接受输入后会不断自我迭代，最终稳定于一个稳定构型。希望了解细节的读者可以自行查阅相关的人工神经网络的资料。

12.4.1 局部搜索算法

如果要设计一个局部搜索算法，有一个非常简单直接的方式。在这一问题中，我们自然地就会定义一个构型的邻居就是将其中一个点的状态取反得到的新构型，然后就可以设计一个局部搜索算法了：从一

个随机初始构型开始，然后检查每个点是否满意，如果有不满意的点，就翻转这个点的状态（那么这个点自然就变得满意了），然后继续检查，直到所有的点都满意为止。如果这个算法会停止，那么停止的时候得到的就是一个稳定构型：因为所有点都满意了。PPT 第 11 页给出了这一局部搜索算法的伪代码（称其为 `State_flipping` 算法），以及一个例子。

那么自然的问题就是，这个算法一定会停止吗? 这一问题的回答并不是很显然，这里需要引入一个非常常用的工具来解答这一问题，即势能函数。首先定义这一函数，然后来看这么定义的合理性。定义势能函数 Φ，它输入一个构型 S，返回这个构型下所有好边的权重绝对值的和，即

$$\Phi(S) = \sum_{e\text{是好的}} |w_e|.$$

显然的一点是，对于任意的构型，$\Phi(S) \geqslant 0$，并且最大值也就是所有边权重绝对值之和，即 $\Phi(S) \leqslant \sum_{e\in E} |w_e|$。下面来观察翻转一个不满意的点后势能函数的变化。设当前状态为 S，有一个不满意点 u，翻转 u 的状态后得到 S'，那么有

$$\Phi(S') - \Phi(S) = \sum_{e=(u,v)\in E,\ e\text{是坏的}} |w_e| - \sum_{e=(u,v)\in E,\ e\text{是好的}} |w_e|.$$

这是因为翻转后原先与 u 相连的好边都变成了坏边，坏边都变成了好边，其余边没有变化。又因为 u 是不满意的，因此与 u 相连的坏边比好边权重绝对值之和大，所以上式大于 0，即 $\Phi(S') > \Phi(S)$。又因为势能函数只能取整数值，故 $\Phi(S') \geqslant \Phi(S) + 1$。这就意味着每次翻转一个不满意的点，势能函数就会增加至少 1。因为势能函数的取值范围是有限的（0 到所有边权重绝对值之和），所以局部搜索算法一定会停止：

定理 12.6 *`State_flipping` 算法最多翻转 $\sum_{e\in E} |w_e|$ 次后会停止。*

由此可以看出定义的势能函数的合理性：这一势能函数随着不满意顶点的翻转，值一定增大，并且势能函数取值有限，因此局部搜索算法一定会停止。此外，不难证明势能函数的局部最大值其实就对应于一个稳定构型：

定理 12.7 *设 S 是一个构型，如果 $\Phi(S)$ 是局部最大值，则 S 是一个稳定构型。*

证明: 假设 S 不是一个稳定构型，即存在一个不满意的点 u，那么可以翻转 u 的状态，得到 S 的一个邻居 S'，并且有 $\Phi(S') > \Phi(S)$，这与 $\Phi(S)$ 是局部最大值矛盾。 ■

所以通过势能函数将 Hopfield 神经网络的稳定构型的寻找转化为了利用 `State_flipping` 这一局部搜索算法找到势能函数的局部最优解，并且局部最优解在这一问题中其实就是想要找到的解（稳定状态）。事实上所谓的稳定构型就是一种纳什均衡，而前面定义的势函数就是算法博弈论中势函数求解纳什均衡的方法，事实上这就解释了为什么会突然冒出一个神奇的势能函数可以证明上述结论，事实上这背后有更一般的方法支撑，感兴趣的读者可以了解相关的概念，这里不展开叙述。

12.4.2 PLS 完全性

到目前为止的讨论似乎都忽略了一件事情：那就是局部搜索的时间复杂度，倘若仔细思考，便会发现其中有很多深入的问题。前面证明了 `State_flipping` 算法最多经过 $\sum_{e\in E}|w_e|$ 次翻转后会停止——相信到今天你应当能一眼看出这是伪多项式时间复杂度，考虑到每条边的长度输入的二进制编码长度，这一时间复杂度实际上是指数的。当然在边权都比较小，例如是点的个数的多项式级别的时候，这一时间复杂度还能退回多项式级别——这些分析和 0-1 背包的动态规划算法完全一致。

于是有个很自然的问题：寻找 Hopfield 神经网络的稳定构型是否存在多项式级别的局部搜索算法呢？答案是不知道——这与 PLS 这一复杂度类有关。事实上类似于 NP，PLS 也是一个复杂度类，只不过 PLS 是对寻找优化问题局部最优解的困难性进行建模。1988 年，Johnson, Papadimitriou 和 Yannakakis 在他们的文章 *How easy is local search?* 引入了复杂度类 PLS。一个局部搜索问题在 PLS 中，如果：

1. 每个解的大小是输入的多项式级别的；
2. 可以在多项式时间内找到一个可行解（不一定是局部最优解）；
3. 可以在多项式时间内计算每个解的代价；
4. 可以在多项式时间内找到每个解的所有邻居。

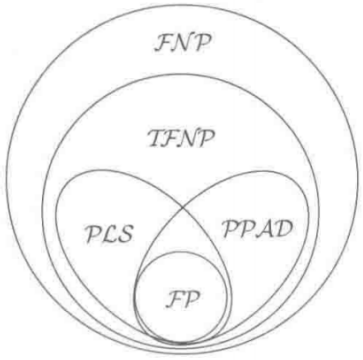
这些要求表明，一个问题在 PLS 中，则可以在多项式时间内判断一个解是不是局部最优解。利用 NP 一讲中学到的知识，可知 PLS 中最困难的问题，也就是所有问题都能归约到它的问题，称之为 PLS-完全问题。在上面的论文中，他们也给出了第一个 PLS-完全问题，即 circuit problem，这里不再深入讲述。关于 PLS 的形式语言定义、归约等内容，读者可以点击这个维基百科链接。这里只介绍一些比较有趣的事实。

为了介绍一些事实，还需要定义一些复杂类（当然我们可能是损失一些严谨性，目的仅仅是介绍一些有趣的事实）。首先定义 FP（functional P），我们知道 P 问题是判定问题，只需要回答是和否，FP 则是在判断是的时候还要给出证据。例如判断两数相加是不是偶数是一个 P 问题，这时只要输出 0/1 即可；但如果是偶数时还要求输出两个数的和，这就是一个 FP 问题。进一步定义 FNP（functional NP），这是 FP 的 NP 版本，即在判断是的时候还要给出证据。具有至少一个证据的 FNP 问题子集称为 TFNP。

除了 PLS，还有一个与搜索相关的复杂类 PPAD（Polynomial Parity Arguments on Directed graphs）。这一名称非常怪异，是多项式时间的有向图上的奇偶校验的含义，当然也直接表明了这一复杂类的来源。之前的讨论已经看到，PLS 的搜索可以视为一个有向无环图对吸收点的搜索，PPAD 则是在一条有向路径图或一个圆环图上的搜索。具体的定义比较繁杂，但关联就是，纯策略纳什均衡的搜索属于 PLS 完全问题，而一些混合策略纳什均衡的搜索属于 PPAD 完全问题。

这些复杂类之间是什么关联呢？可以用一张图来形象地说明：

具体相关的结论与讨论如下（事实上图中所有的包含关系都应当是暂时未被确认的）：



1. 最关键的直观：PLS 处于 P 和 NP 的难度之间，这也符合我们对局部搜索问题的直观；

2. Schäffer 和 Y annakakis 在 1991 年证明了 Hopfield 神经网络的稳定构型的局部搜索是 PLS 完全的，因此目前还没有找到一个多项式时间的算法来解决这一问题；

3. 如果所有 TFNP 问题都是 FNP 完全的，那么 co-NP = NP；

4. TFNP 是没有完全问题的，这是因为它是一个语义定义的复杂类，而别的复杂类都是语法定义的；

5. 如果所有 PLS 问题都是 FNP 完全的，那么 co-NP = NP.

最后一个结论是很新的，发表在 STOC 2021 上，John Fearnley, Paul W. Goldberg, Alexandros Hollender, Rahul Savani 证明了连续局部搜索（例如凸优化中的梯度下降等问题）对应的复杂类 CLS（continuous local search）是 PPAD 和 PLS 的交集，非常有趣：

$$\mathsf{CLS} = \mathsf{PPAD} \cap \mathsf{PLS}.$$

总而言之，上述提到的复杂类之间的关系仍待解决，特别是算法博弈论这一学科的兴起使得对 PLS 和 PPAD 的研究变得更加重要，这也是一个非常有趣的研究方向。

12.5 最大割问题

最大割问题也是一个经典的 NP 困难问题，在这一问题中，我们希望将一个边权全为正的无向图 $G = (V, E)$ 的顶点集合 V 分成两个集合 A 和 B（这时的解记为 (A, B)），使得割边的权重和最大，其中割边的含义就是一条边的两个端点分别在 A 和 B 中，即要最大化

$$w(A, B) = \sum_{(u,v)\in E, u\in A, v\in B} w_{uv}.$$

12.5.1 局部搜索算法

下面希望给出一个局部搜索解法——这是非常自然的，因为它和 Hopfield 神经网络问题之间有一个很直接的关联。有一个很简单的观察，对于任意的解 (A, B)，将 A 中的点赋状态 -1，B 中的点赋状态 1，因为所有边的权重都是正数，所以最大化割边权重和对应于 Hopfield 神经网络问题其实就是最大化好边的总权重和 $\Phi(S)$，这就和 Hopfield 神经网络问题一样了。在 Hopfield 神经网络问题中，使用 `State_flipping` 算法来找到局部最大值，在那时局部最大值对应的构型就是一个稳定构型，现在则对应一个最大割的局部最优解。我们想要知道这个解的质量如何，事实上令人惊喜的是，这个局部搜索算法给出的局部最优解最差也不会低于最优解的一半：

定理 12.8 *设 (A, B) 是如上局部搜索算法得出的一个局部最优解，(A^*, B^*) 是最优解，则*

$$\frac{w(A,B)}{w(A^*,B^*)} \geqslant \frac{1}{2}.$$

证明: 记图 G 中所有边的权重之和为 $W = \sum_{(u,v)\in E} w_{uv}$。因为 (A, B) 是一个局部最优解，即把 u 放到 B 中会使得割边权重和降低，即

$$\sum_{v\in A} w_{uv} \leqslant \sum_{v\in B} w_{uv},$$

这是因为把 u 从 A 换到 B 后，割边总权重只是增加了不等式左边权重，减小了不等式右边权重。对所有的 $u \in A$ 都有这样的不等式，将这些不等式求和有

$$\sum_{u\in A}\sum_{v\in A} w_{uv} \leqslant \sum_{u\in A}\sum_{v\in B} w_{uv},$$

很显然，等式左边就是 2 倍的 A 中所有边的权重之和，因为每条边都被计算了两次，等式右边就是所有割边的权重之和，所以有

$$2\sum_{(u,v)\in A} w_{uv} = \sum_{u\in A}\sum_{v\in A} w_{uv} \leqslant \sum_{u\in A}\sum_{v\in B} w_{uv} = w(A,B),$$

对称地，如果开始选取 $u \in B$，也有

$$2\sum_{(u,v)\in B} w_{uv} \leqslant w(A,B),$$

又知道 $W = \sum_{(u,v)\in A} w_{uv} + \sum_{(u,v)\in B} w_{uv} + w(A,B)$，并且最大割的权重和不可能超过全体边的权重和 W，所以有

$$w(A^*,B^*) \leqslant W = \sum_{(u,v)\in A} w_{uv} + \sum_{(u,v)\in B} w_{uv} + w(A,B) \leqslant 2w(A,B),$$

从而得证。 ■

这一证明看起来有些巧妙，但事实上这里的套路仍然来源于算法博弈论中更一般的框架，实际上就是证明纳什均衡的 POA（price of anarchy，无秩序代价，即纳什均衡下的代价（局部最优解）与最优代

价（全局最优解）的比值）的标准证明方法：第一步利用局部最优的特点得到任意一个元素具有的表达式（之后就是纳什均衡），第二步将它们求和，第三步则是利用一些巧妙的观察或技术性的操作得到最终的结论。不难看出，第一步和第二步是非常标准的，第三步则是不同问题需要不同巧妙的想法。

上面的结论表明，通过局部搜索找到的局部最优解和最优解之间是可能有可以证明的近似比的。但是注意，这并不一定是 2-近似算法。注意这里的局部搜索算法和 Hopfield 神经网络一样，都是通过 `State_flipping` 寻找好边权重和 $\Phi(S)$ 的局部最优点，而我们已经提及，这一问题是 PLS 完全的，所以目前还没有找到一个多项式时间的算法来解决这一问题。PPT 第 17 页给出了最大割问题真正的多项式时间的近似算法的研究历史，但背后需要的基础知识目前我们并未涉及，所以这里不再展开。这里能给出的是一个类似于 FPTAS 的近似方案，只是近似比是 $2+\varepsilon$。

这里的想法非常简单，事实上在控制迭代次数引起的复杂度时非常常用（这里就是求 $\Phi(S)$ 的局部最优值需要的迭代次数可能太多）。我们要求算法在找不到一个能对解有 “比较大的提升” 的时候就停止，即使当时的解不是局部最优解：当处于解 $w(A,B)$ 时，要求下一个解的权重至少要增大 $\frac{\varepsilon}{n}w(A,B)$，其中 n 是图 G 的顶点数。对于这一算法，有如下结论：

定理 12.9 *设 (A,B) 是上述算法给出的解，(A^*,B^*) 是最优解，则*

$$\frac{w(A,B)}{w(A^*,B^*)} \geqslant \frac{1}{2+\varepsilon}.$$

并且这一算法会在 $O(\frac{n}{\varepsilon}\log W)$ 次状态翻转后停止，其中 W 是所有边的权重之和。

证明: 近似比的证明事实上非常简单，只需要对前一个近似比为 2 的证明做一些修改即可。因为必须要一个 “比较大的改进” 才会继续搜索，所以当处于算法的结束状态时，解 (A,B) 应当满足，对于任意的 $u\in A$ 有

$$\sum_{v\in A} w_{uv} \leqslant \sum_{v\in B} w_{uv} + \frac{\varepsilon}{n}w(A,B),$$

这是因为这时将 u 从 A 换到 B 后，增加的割边权重和不会超过 $\frac{\varepsilon}{n}w(A,B)$。然后接下来所有的证明都基于此改进继续推进即可证明。

接下来证明时间复杂度的结论。因为每次状态翻转都会使得割边权重和增加 $1+\frac{\varepsilon}{n}$ 倍，又因为 $(1+\frac{1}{x})^x \geqslant 2, \forall x\geqslant 1$，所以有 $(1+\frac{\varepsilon}{n})^{\frac{n}{\varepsilon}} \geqslant 2$，即目标函数 Φ 变为原先的 2 倍需要的状态翻转次数至多为 $\frac{n}{\varepsilon}$ 次，所以算法在 $O(\frac{n}{\varepsilon}\log W)$ 次状态翻转后能翻转 W 倍（这也是从目标函数最低值 1 翻转到目标函数最大值 W 所需的最大倍数），因此状态翻转次数的上界得证。 ■

12.5.2 更优的邻居关系

事实上进行到现在我们已经见到很多局部搜索的例子了，但还没有对某个问题的邻居关系的选择有很深入的探讨，都是选择了最直观的邻居关系。事实上，在最开始引入局部搜索的时候就提到，局部搜索

有两个关键点，第一是选取好的邻居关系，第二是每一步找到一个好的邻居继续搜索。在 Metropolis 算法和模拟退火中特别讨论了后者（特别针对于顶点覆盖问题），现在需要仔细讨论前者（特别针对最大割问题）。

很自然地，选取邻居关系时应当注意以下两点：

1. 每个解的邻居数目不应该太少，否则很容易在短短几步之内就陷入一个不佳的局部最优解；
2. 每个解的邻居数目不应该太多，否则搜索空间太大，很难在短时间内找到一个好的邻居——考虑极端情况，如果每个解的邻居数目是所有可能的解，那么就退化成了穷举搜索。

这些原则说起来容易，但真的找到一个更好的邻居关系并非易事。考虑最大割问题，一个非常简单的想法就是，原先的邻居关系是把一个点从一个集合换到另一个集合，现在可以考虑把 k 个点从一个集合换到另一个集合，这样就有了一个 k-flip 的邻居关系。一个显然的关系是，(A, B) 和 (A', B') 是 k-flip 邻居，那么必然也是 k'-flip 邻居，其中 $k' > k$。因此如果 (A, B) 是 k'-flip 邻居关系下的局部最优解，那么它也是 k-flip 邻居关系下的局部最优解（其中 $k' > k$）。这样就可以通过不断增大 k 来在每次翻转时找到一个更好的邻居关系，最后解的近似比可能更低。

然而，当 k 增大时，每一步翻转找到一个更好的邻居的时间复杂度也会增大——因为搜索空间将是 $\Theta(n^k)$ 的，在 k 稍微增大一些时这一复杂度就是不可接受的了。

所以需要找到一个权衡。Kernighan 和 Lin（1970）提出了一种定义邻居关系的方式，它通过如下算法得到 (A, B) 的邻居集合：

1. 第一步，选取一个点进行翻转，要求选取的这个点是让翻转后结果的割边权重和最大的那个点（即使有可能最大的权重也会比现有权重低），标记这个被翻转的点，令翻转后的解为 (A_1, B_1)；
2. 之后第 $k (k > 1)$ 步时，有前一步得到的结果 (A_{k-1}, B_{k-1})，以及有 $k-1$ 个已经被标记的点。然后选择一个没有被标记的点进行翻转，要求选取的这个点是让翻转后结果的割边权重和最大的那个点（即使有可能最大的权重也会比现有权重低），标记这个被翻转的点，令翻转后的解为 (A_k, B_k)；
3. n 步之后所有点都被标记了，也就是说所有的点都恰好被翻转了一次，因此实际上就有 $(A_n, B_n) = (B, A)$，这本质上与 (A, B) 并无区别。因此上述过程（称为 K-L 启发式算法）得到了 $n-1$ 个邻居，即 $(A_1, B_1), (A_2, B_2), \ldots, (A_{n-1}, B_{n-1})$。因此 (A, B) 是一个局部最优解当且仅当对于任意的 $1 \leqslant i \leqslant n-1$，有 $w(A, B) \geqslant w(A_i, B_i)$。

简单分析即可发现上述寻找邻居的过程是一个 $O(n^2)$ 的过程，因此相比于简单的进行 k 比较大的 k-flip 的邻居关系，K-L 启发式算法的时间复杂度是可以接受的，并且实践表明，局部搜索使用 K-L 启发式算法给出的邻居的效果很好，尽管这一事实暂时还没有理论推导来说明。总而言之，K-L 启发式算法给出的邻居比较好地权衡了算法效率和结果准确性。

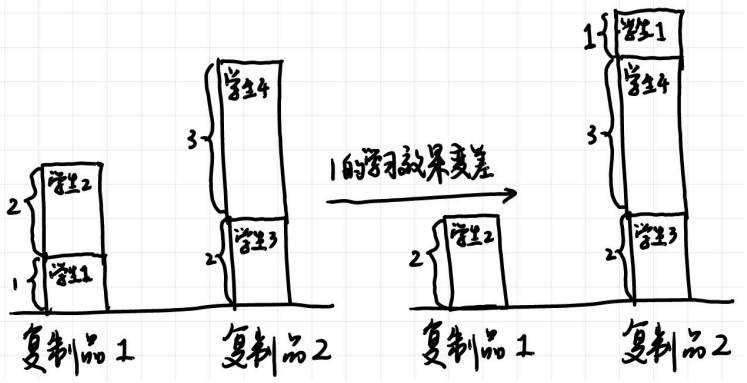
问题：如果助教可以复制

期末考试的脚步逐渐逼近，有 k 个正在经受 ADS 折磨的同学（编号为 $1,\ldots,k$）希望能找 ADS 助教补习功课。每个同学 i 都自带一个正整数 w_i，代表他/她学习 ADS 的困难程度，并且这个数字是公开的：每个人不仅知道自己学习的困难程度，也知道别人学习的困难程度。

正在加急完成毕业论文的助教见此情形觉得压力太大，于是复制了 m 个一模一样的自己来辅导同学。助教把这 m 个复制品放在教室后就离开了，留下同学们自由选择 m 个复制品中的一个辅导自己。当然复制品也是一条生命，因此一个复制品的培训效果会随着它辅导的同学的困难程度之和增加而减小，因此每个同学都希望自己选择的辅导助教的困难程度之和尽可能小。于是我们可以定义一个 “均衡状态”：在这一状态下，每个同学都选择了一个复制品，且每个同学都不愿意换一个复制品来辅导自己，因为这样一定会使得自己选择的新复制品的困难程度之和大于等于原先的困难程度之和，学习效果不会提升。

注意我们有一个重要的假设。即在选择复制品期间每个同学**不能互相讨论**，也就是每个人做出的决定都是独立完成的，每个同学只是根据当前状态和自身需求来行动，不能和别人商量着一起做什么。

举个简单的例子，假如助教复制了两个一模一样的自己，有四个同学 $1,2,3,4$，他们学习 ADS 的困难程度分别为 $1,2,2,3$，那么有一种均衡状态是：同学 1 和 2 选择了第一个复制品，同学 3 和 4 选择了第二个复制品，此时第一个复制品的困难程度之和为 $1+2=3$，第二个复制品的困难程之和为 $2+3=5$。这一选择为什么是均衡状态呢？因为每个同学都不愿意换一个复制品来辅导自己，比如对于同学 1，现在选择的复制品的困难程度之和为 3，如果他换成选择 2，那么变成了 1、3、4 都选择了第二个复制品，第二个复制品的困难程度之和变为 $1+2+3=6>3$，所以同学 1 完全没必要从 1 换到 2，因为会使得自己的学习效果变差（如下图所示）；又例如对于同学 3，现在所在的第二个复制品的困难程度之和为 5，如果他换成第一个复制品，那么变成了 1、2、3 都选择了第一个复制品，则第一个复制品的困难程度之和变为 $1+2+2=5$，所以同学 3 也完全没必要换一个复制品，因为无法改变自己的学习效果。对于同学 2 和 4 也是类似的，所以不再赘述。



1. 自然地，可以用一个局部搜索算法来寻找均衡状态：每个同学首先都随机选择辅导自己的复制品，然后在局部搜索算法的每一步，其中的**一个**同学会表达自己的不满，因为它自己为了提升学习效果提出要换成**更换之后困难程度之和更小的复制品**辅导自己。然后不断会有同学表达不满，直到没有同学表达不满时，算法停止。

(a) 请写出这一局部搜索算法中的邻居关系是什么；

(b) 证明：如果算法会停止，那么算法停止的时候，得到的状态一定是一个均衡状态；

(c) 这一算法对于任意情况都会停止吗？如果是，请给出证明，如果不是，请给出一个反例。

2. 现在放宽假设，假如每个同学在做决策的时候是可以互相交流的，也就是他们可以一起商定最后的选择。这些同学内心的良知告诉他们，他们应该让最后的局势尽可能对每个人公平，也就是希望最小化**困难程度之和最大的复制品的困难程度（简称困难瓶颈）**。还是前面的例子，商量后的结果可以是第 1、4 个同学选择第一个复制品，第 2、3 个同学选择第二个复制品，这样第一个复制品的困难程度之和为 $1+3=4$，第二个复制品的困难程度之和为 $2+2=4$，此时的困难瓶颈为 4（并且显然是最优的了，因为平摊了全部困难程度），而之前的选择导致的困难瓶颈为 5，不如这一分配。

(a) 最小化困难瓶颈的选择结果一定是一个均衡状态吗？如果是，请给出证明，如果不是，请给出一个反例；

(b) 若限制 $m=2$，最小化困难瓶颈的选择结果一定是一个均衡状态吗？如果是，请给出证明，如果不是，请给出一个反例；

(c) 证明：任意一个均衡状态的困难瓶颈一定不大于最小困难瓶颈的两倍。

提示：

1. 1(c) 和 2(b) 可以利用上课所学 Hopfield 神经网络的定义势函数的方法解决；
2. 2(c) 可以参考近似算法的讲义中的最小化工时调度问题，实际上整个问题都只是从那个问题改编过来的，所以如果你仔细阅读了过往的讲义，这题的大部分小问应当是不困难的。