

21120491: 高级数据结构与算法分析

2023-2024 学年春夏学期

Lecture 12: 局部搜索

教师: 刘金飞, 助教: 吴一航

日期: 2024 年 5 月 14 日

12.1 基本思想

本讲我们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, 即我们的目标是最大化或最小化某个目标函数——例如 0-1 背包问题, 我们希望最大化背包中的物品权重总和。

在之前的讨论中, 我们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, 我们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我们非常自然地称其为**局部搜索 (local search)**——因为我们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我们的解, 直到我们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 (之后我们会展开讨论)。一个非常简单的观察来源于 Euler-Lagrange 方程的推导, 对于一个泛函的极值, 我们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。我们考虑以下对应:

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，我们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，我们可以找到一个“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 (比如笔者就找不到)、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 (你不能排除有人眼瞎看上笔者的可能)。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：如果我们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)。我们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)，我们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，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)。我们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，这样的个体我们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，即为什么我们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，但我们还是选择它呢？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，有时候我们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，我们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，那么此时我们很自然地就会求助于使用局部搜索这一非常自然的方式。

我们这里来看一个优化问题的例子来印证我们上面的说法。考虑问题 $\min_{\mathbf{x} \in \mathbb{B}^n} f(\mathbf{x})$ ，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)

$$\mathbb{B}^n = \{\mathbf{x} \in \mathbb{R}^n \mid 0 \leq x^{(i)} \leq 1, i = 1, \dots, n\}$$

假如我们对 f 一无所知，那么显然是无法求解的。现在，我们加入一个限制。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）

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

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 ，

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

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	$\min_{\mathbf{x} \in \mathbb{B}^n} f$ 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}_ε	$\bar{\mathbf{x}} \in \mathbb{B}^n, \quad f(\bar{\mathbf{x}}) - f^* \leq \varepsilon$

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： $\mathbf{x}_\alpha = \left(\frac{i_1}{2p}, \frac{i_2}{2p}, \dots, \frac{i_n}{2p}\right)$ ，其中 $\alpha \equiv (i_1, i_2, \dots, i_n) \in \{0, 1, \dots, p\}^n$ ；

2. 在所有的点 x_α 中找到使目标函数达到最小值的点 \bar{x} ;
3. 算法输出 $(\bar{x}, f(\bar{x}))$ 。

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, 我们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, \bar{x} 是均匀网格法的输出, 那么

$$f(\bar{x}) - f^* \leq \frac{L}{2p}$$

证明: 设 x_* 是最小值点, 那么存在 n 维坐标 (i_1, i_2, \dots, i_n) 使得

$$x_1 \equiv x_{(i_1, i_2, \dots, i_n)} \leq x_* \leq x_{(i_1+1, i_2+1, \dots, i_n+1)} \equiv x_2,$$

即 x_* 一定落在某个小立方体之内。注意到对于 $i = 1, \dots, n$, 有 $x_2^{(i)} - x_1^{(i)} = \frac{1}{p}$, 且 $x_1^{(i)} \leq x_*^{(i)} \leq x_2^{(i)}$ 。我们定义 $\hat{x} = (x_1 + x_2)/2$,

$$\tilde{x} = \begin{cases} x_2^{(i)}, & \text{if } x_*^{(i)} \geq \hat{x}^{(i)} \\ x_1^{(i)} & \text{otherwise} \end{cases}$$

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

$$\|\tilde{x} - x_*\|_\infty = \max_{1 \leq i \leq n} |\tilde{x}^{(i)} - x_*^{(i)}| \leq \frac{1}{2p}$$

$$f(\bar{x}) - f(x_*) \leq f(\tilde{x}) - f(x_*) \leq L \|\tilde{x} - x_*\|_\infty \leq \frac{L}{2p}$$

■

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 $\bar{x} \in \mathbb{B}^n$, 使得 $f(\bar{x}) - f^* \leq \varepsilon$ 。我们有如下结论:

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 $p = \left\lfloor \frac{L}{2\varepsilon} \right\rfloor + 1$ 即可保证

$$f(\bar{x}) - f^* \leq \frac{L}{2p} \leq \varepsilon,$$

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 $\mathcal{A}(\mathcal{G})$ 就确定了我们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,就需要我们计算 \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。基于这一思想我们有如下定理:

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

证明: 我们令 $p = \left(\left\lfloor \frac{L}{2\varepsilon} \right\rfloor\right)^n \geq 1$, 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,我们可以从这里下手构造函数。

我们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 $f(\mathbf{x}) = 0$ 。因此,该方法只能求出 $\bar{\mathbf{x}} \in \mathbb{B}^n$, $f(\bar{\mathbf{x}}) = 0$ 。然而,注意到存在 $\hat{\mathbf{x}} \in \mathbb{B}^n$, 使得

$$\hat{\mathbf{x}} + \frac{1}{p}\mathbf{e} \in \mathbb{B}^n, \quad \mathbf{e} \in (1, \dots, 1)^\top \in \mathbb{R}^n$$

且在立方体 $\mathbb{B} = \left\{\mathbf{x} \mid \hat{\mathbf{x}} \leq \mathbf{x} \leq \hat{\mathbf{x}} + \frac{1}{p}\mathbf{e}\right\}$ 内没有任何测试点。我们令 $\mathbf{x}_* = \hat{\mathbf{x}} + \frac{1}{2p}\mathbf{e}$ 。考虑函数

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

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 $\mathbb{B}' = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_*\|_\infty \leq \frac{\varepsilon}{L}\}$ 内不为 0, 由于 $2p \leq L/\varepsilon$, 因此 $\mathbb{B}' \subseteq \mathbb{B}$ 。

而在我们的方法中,对于所有的测试点, $f_p(\mathbf{x})$ 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。■

现在我们可以对 \mathcal{G} 进行总结

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

如果 $\varepsilon \leq O\left(\frac{L}{n}\right)$, 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,如果我们执意要对一个函数求解全局最优(在如上条件下),那么我们的时间复杂度的底线也将是指数级别的。因此我们会求助于局部搜索,以更快速、更精妙的方式找到一个局部最优解。

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，所以我们有

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

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

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

于是我们有

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

■

我们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（就是我们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。

接下来我们开始介绍局部搜索算法。显然我们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，我们的目标是找到一个最小的顶点覆盖，然后我们需要每一步在邻居中找一个更优解，所以我们不妨让邻居就是比当前解少一个顶点的解。这样我们就可以定义局部搜索算法了：我们选择的初始解就是图 G 的所有顶点，即 $S = V$ ，然后我们每次从 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，否则我们就得到了一个局部最优解。

我们称这一算法为“梯度下降法”。相信读者对这一词汇并不陌生，它是我们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。在这里我们相当于借用了其中的思想，我们的梯度下降其实就是任选一个点从现有解中删除。

PPT 第 6 页给出了我们的“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；第三种则是有多种情况，我们可以通过不同的下降选择找到不同的局部最优解。由此我们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。所以接下来我们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, Rosenbluth, Teller 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。然后我们考虑温度 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，我们可以将 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 $E(S') \leq E(S)$ ，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，我们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，设计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 $t \rightarrow \infty$ 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 第七页给出了针对顶点覆盖问题的 Metropolis 算法。对于 PPT 第六页的 case 1，有了 Metropolis 算法，我们就有可能跳出局部最优解，找到全局最优解。然而如果我们看 case 0，就会发现当我们当我们进行到只剩下几个点的时候，这时因为我们是均匀选取下一个状态，那么有非常大的概率我们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，这样我们就会偏离最优解（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，但是即使是不用 Metropolis 算法，我们都能解出最优解，现在反而解不出了，这是很难让人接受的，因此我们还需要进一步优化 Metropolis 算法。

我们发现还有一个关键的因素我们没有考虑：Metropolis 算法的一个重要参数是温度 T 。我们之前也分析了，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，这时候我们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，这样我们就能够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，因为前面的定理告诉我们当 $t \rightarrow \infty$ 时，Metropolis 算法的解会收敛到玻尔兹曼分布，这样我们就能大概率找到很好的解。但是实际中我们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，因此我们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，我们感觉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我们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我们想要的解的情况，还会看到局部最优解是有可证明的近似比的解。我们接下来要看的问题是我们的老熟人：旅行商问题，当然这里我们是建立在上一讲的 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。

这已经是我们第三次遇见旅行商问题了，这次我们将从局部搜索的角度提供一种解决方案。我们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 $w(u, v)$ ，我们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，我们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，我们之前提到过，在输入不算太大的时候我们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），所以我们可以尝试用动态规划来解决旅行商问题。

在动态规划中，我们的关键在于找出最优子结构。我们仍然用一个思维实验来开始我们的讨论，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。假设我们给所有顶点编号 $1, \dots, n$ ，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，那么我们的问题就转换为了找最优的 j 作为最后一跳，即

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

其中我们用 c_{ij} 表示从 i 到 j 的成本。

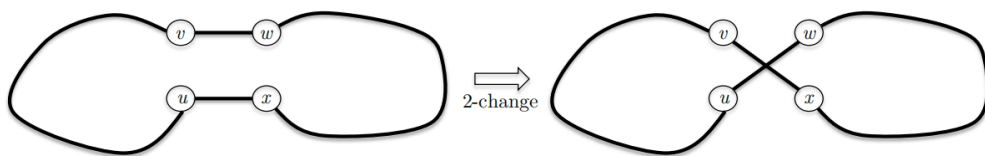
于是我们的问题变成了，怎么找到从 1 出发到任意顶点 $j \neq 1$ 的最短的经过每个顶点恰好一次的路径。如果能解决这一问题，那么我们再代入上面的表达式即可求出旅行商的最短路径。这里我们就要用最短路问题自带的最优子结构来设计动态规划算法了（我们在 Bellman-Ford 和 Floyd-Warshall 算法中已经实践过）：即最短路径的某一段也一定是对应的起点终点的最短路径。因此如果我们要求从 1 到 j 的最短路径，我们可以考虑最后一跳的选取，假设最短路径最后一跳是从 k 到 j ，那么我们可以将问题分解为从 1 到 k 的最短路径加上从 k 到 j 的成本，当然我们并不能直接知道哪个 k 最好，所以还需要遍历所有的 k ，这样就决定了最后一跳。然后我们递归解决倒数第二跳，以此类推。我们不难发现，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。这样我们就可以写出动态规划的递推式：

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

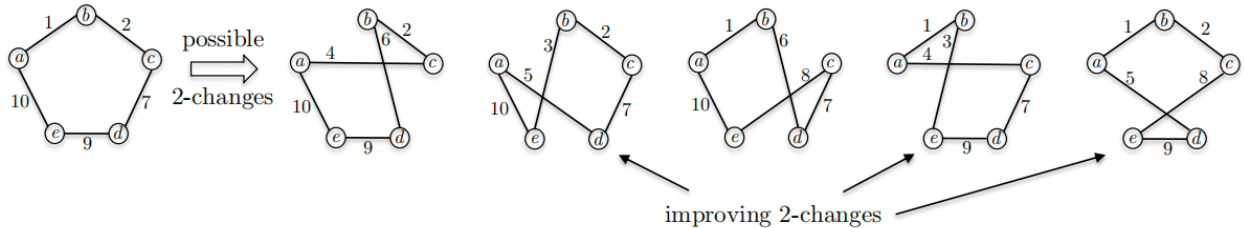
接下来我们就是要写一个迭代的程序填满子问题的表格，根据递推式的特点，我们知道只要按照 S 的大小从小到大依次填表，就能保证解每个问题的时候，需要的子问题都是已经填满了的。这样我们就可以得到最终的最优解。这一算法称为 Held-Karp 算法，复杂度分析也很简单：我们有 $n \cdot 2^n$ 个子问题，每个问题的计算复杂度是 $O(n)$ （求 min 需要遍历所有可能），所以总的复杂度是 $O(n^2 \cdot 2^n)$ （当然不要忘了算法最后一步还需要回归到 1 到 1 的环路，这里需要遍历 $n-1$ 种最后一跳的可能，是线性时间的）。

12.3.2 局部搜索算法

接下来我们回归正题，即设计一个局部搜索算法。在上一讲我们已经介绍了旅行商问题的一个 2-近似算法，我们可以从这个 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？我们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 $n-2$ 。因此我们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：



我们称这样的变换为 2-变换 (2-change)。于是接下来我们就可以在 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。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。

当然这并不是我们讨论的重点, 我们这里希望讨论的是 Hopfield 神经网络的稳定构型 (stable configurations)。为了说明这一问题, 我们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, 我们记顶点 u 的状态为 s_u ;
3. 如果对于边 $e = (u, v)$ 有 $w_e > 0$, 则我们希望 u 和 v 具有相反的状态; 如果 $w_e < 0$, 则我们希望 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. 我们将满足上述条件的边 e 称为 “好的 (good)”, 否则称为 “坏的 (bad)”;
5. 我们称一个点 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

$$\sum_{v: e=(u,v) \in E} w_e s_u s_v \leq 0,$$

反之, 如果 u 不满足这一条件, 我们称 u 是 “不满意的 (unsatisfied)”;

6. 最后，我们称一个构型是“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，但读者如果看到 PPT 第 10 页的例子，应当就会熟悉这些定义。我们的问题是，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，我想我们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，然后我们就可以设计一个局部搜索算法了：我们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，我们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，那么停止的时候我们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（我们称其为 `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，这里我们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。我们先定义这一函数，然后来看这么定义的合理性。我们定义势能函数 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， $\Phi(S) \geq 0$ ，并且最大值也就是所有边权重绝对值之和，即 $\Phi(S) \leq \sum_{e \in E} |w_e|$ 。下面我们来观察一下当我们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' ，那么我们有

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

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 $\Phi(S') \geq \Phi(S) + 1$ 。这就意味着我们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），所以我们的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。

由此我们可以看出我们定义的势能函数的合理性：我们发现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，因此我们的局部搜索算法一定会停止。至于这一势能函数是如何定

义出来的，我们在最后关于纳什均衡的讨论中会给出一个统一的解释，这里我们还需要更进一步理解的是，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：实际上这一结论非常显然：假设 S 不是一个稳定构型，即存在一个不满意的点 u ，那么我们可以翻转 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。 ■

所以我们通过势能函数将 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我们想要找到的解（稳定状态），这一点在将来纳什均衡的讨论中也会再见到：事实上寻找（势）博弈的（纯策略）纳什均衡也就是找势函数的局部最小（大）值。

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，倘若我们仔细思考，便会发现其中有很多深入的问题。我们在前面证明了 **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。

于是我们有个很自然的问题：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：

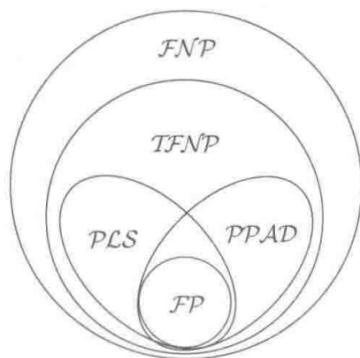
1. 每个解的大小是输入的多项式级别的；
2. 可以在多项式时间内找到一个可行解（不一定是局部最优解）；
3. 可以在多项式时间内计算每个解的代价；
4. 可以在多项式时间内找到每个解的所有邻居。

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，我们知道 PLS 中最困难的问题，也就是所有问题都能归约到它的问题，我们称之为 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](#)。这里我们只介绍一些比较有趣的事实。

为了介绍一些事实，我们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）。我们首先定义 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，这时我们只要输出 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。我们还定义 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 $TFNP$ （例如我们以后会提到的有限博弈的混合策略纳什均衡一定存在）。

除了 PLS ，我们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。我们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？我们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 Yannakakis 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，因此目前我们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 $co-NP = NP$ 。

最后一个结论是最新的，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：

$$CLS = PPAD \cap PLS.$$

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，即我们要最大化

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

12.5.1 局部搜索算法

我们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。我们有一个很简单的观察，对于任意的解 (A, B) ，然后将 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，我们使用 `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

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

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

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

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。我们对所有的 $u \in A$ 都有这样的不等式，将这些不等式求和有

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

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，所以我们有

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

对称的，如果开始我们选取 $u \in B$ ，我们也有

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

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

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

从而我们得证。 ■

这一证明看起来有些巧妙，但我们在下一节会表明，实际上这是博弈论中证明纳什均衡的 POA (price of anarchy, 无秩序代价) 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。我们看到，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，所以我们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。我们这里能给出的是一个类似于 FPTAS 的近似方案，只是我们的近似比是 $2 + \varepsilon$ 。

我们的想法非常简单，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：当我们处于解 $w(A, B)$ 时，我们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，我们有如下结论：

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

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

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。因为我们必须要一个“比较大的改进”才会继续搜索，所以当我们处于算法的结束状态时，我们的解 (A, B) 应当满足，对于任意的 $u \in A$ ，我们有

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

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。因为我们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 $(1 + \frac{1}{x})^x \geq 2, \forall x \geq 1$ ，所以有 $(1 + \frac{\varepsilon}{n})^{\frac{n}{\varepsilon}} \geq 2$ ，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，所以我们的算法在 $O(\frac{n}{\varepsilon} \log W)$ 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），因此我们的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，但我们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我们就提到，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我们的搜索。在 Metropolis 算法和模拟退火中我们特别讨论了后者（特别针对于顶点覆盖问题），现在我们需要仔细讨论前者（特别针对最大割问题）。

很自然地，我们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，否则我们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，否则我们的搜索空间太大，我们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，那么我们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。我们考虑最大割问题，一个非常简单的想法就是，原先我们的邻居关系是把一个点从一个集合换到另一个集合，我们可以考虑把 k 个点从一个集合换到另一个集合，这样我们就有了一个 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 $k' > k$ ）。这样我们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——因为我们的搜索空间将是 $\Theta(n^k)$ 的，在 k 稍微增大一些时这一复杂度就是不可接受的了。

所以我们需要找到一个权衡。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，我们选取一个点进行翻转，我们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），我们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，我们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。然后我们选择一个没有被标记的点进行翻转，我们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），我们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（我们称为 K-L 启发式算法）得到了 $n - 1$ 个邻居，即 $(A_1, B_1), (A_2, B_2), \dots, (A_{n-1}, B_{n-1})$ 。因此 (A, B) 是一个局部最优解当且仅当对于任意的 $1 \leq i \leq n - 1$ ，我们有 $w(A, B) \geq w(A_i, B_i)$ 。

我们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。

12.6 最优反应动力学与纳什均衡

最后我们来讨论一个非常有趣的话题，即最优反应动力学和纳什均衡，它来源于算法博弈论这一年轻而深入的学科。或许说到纳什均衡，每个人都并不陌生——这是博弈论中最有名的一个解概念。但真要说出纳什均衡的具体概念，似乎就出现困难了。

为了定义纳什均衡，我们首先需要定义什么是博弈论，什么是博弈。所谓的博弈论就是研究多个理性参与人互相影响的场景下如何做出最优决策的学科，博弈则是理性参与人在这些场景下的互动。一个博弈通常有三个组成部分组成：

1. 参与人（或称智能体）：无需过多解释，就是参与博弈的理性个体，参与人集合可以记为 $N = \{1, 2, \dots, n\}$ ；
2. 策略：每个参与人在博弈中可以做出的选择，参与人 i 能选择的所有策略集合记为 S_i ；
3. 效用函数：每个参与人在自己和其他人当前的策略选择下的效用，即效用函数是一个映射 $u_i : S_1 \times S_2 \times \dots \times S_n \rightarrow \mathbb{R}$ 。

在博弈论中我们通常用 $-i$ 表达除了参与人 i 之外的所有参与人，即 $-i = \{1, 2, \dots, i-1, i+1, \dots, n\}$ 。我们称一个策略组合 (s_1, s_2, \dots, s_n) 是一个纳什均衡，如果对于任意的 $i \in N$ ，我们有

$$u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i}), \forall s'_i \in S_i.$$

解释一下这句话，实际上就是：纳什均衡就是一个策略组合，其中每个参与人都在自己的策略下是最优的，即如果其他人的策略不变，那么参与人 i 改变自己的策略不可能使得自己的效用更高。因此纳什均衡更通俗而言就是一个没有人愿意偏离的策略组合。

当然上面定义的是效用最大化版本的纳什均衡，有时参与人参与博弈不是带来效用，而是产生一定的成本，此时参与人的目标就是最小化成本。我们设任一参与人 i 的成本函数 $c_i: S_1 \times S_2 \times \cdots \times S_n \rightarrow \mathbb{R}$ ，即成本函数是一个映射，表示参与人 i 在自己和他人当前的策略选择下的成本。我们称一个策略组合 (s_1, s_2, \dots, s_n) 是一个最小化成本纳什均衡，如果对于任意的 $i \in N$ ，我们有

$$c_i(s_i, s_{-i}) \leq c_i(s'_i, s_{-i}), \forall s'_i \in S_i.$$

这个定义和效用最大化的纳什均衡是类似的，只不过这里是最小化成本。

定义了一些抽象的概念后，我们来看一个经典的例子，并且从这个例子出发探讨纳什均衡的求解于局部搜索之间的联系。这一问题称为单元自私路由问题，对网络或交通流量进行了建模。单元自私路由由有向图 $G = (V, E)$ 和 k 个智能体构成，其中每条边 e 有一个取值非负的代价函数 c_e ，它是这条边上通行的智能体数 f_e 的一个函数，随着智能体数增多而不减。每个智能体 i 有一个起点 o_i 和一个终点 d_i ，并且希望以最小的代价从 o_i 到达 d_i 。

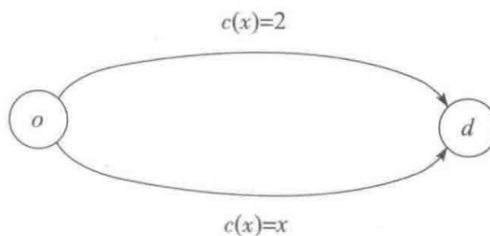
每个智能体做出了他的选择后，我们有一个自然的分流概念：我们的分流由向量 (P_1, P_2, \dots, P_k) 表示，其中 P_i 表示智能体 i 的路径，即 P_i 是一个从 o_i 到 d_i 的路径。对应于前面博弈的定义，这一向量实际上就是一个策略组合，某个智能体在某个策略组合下的效用函数值就是他经过的所有边的代价和。自然的，这里的纳什均衡就是定义在分流向量（因为这就是策略组合）上的，我们称其为均衡分流，它的含义为，没有任何一个智能体可以通过单方面改变路径选择来减少自身路径的代价：

定义 12.10 (均衡分流) 若对每个智能体 i 和 o_i - d_i 路径 \hat{P}_i ，都有

$$\sum_{e \in P_i} c_e(f_e) \leq \sum_{e \in \hat{P}_i \cap P_i} c_e(f_e) + \sum_{e \in \hat{P}_i \setminus P_i} c_e(f_e + 1),$$

则称分流 (P_1, \dots, P_k) 是均衡分流。

下图给出了一个最简单的例子：



这一例子中假设有两个智能体，他们的起点都是 o ，终点都是 d ，上面边的代价是个常数，下面边的代价就等于通过的智能体人数。很明显这一博弈有两个均衡分流：

1. 每条边一个智能体：可以检查，上面的智能体不愿意换到下面，下面的也不愿意换到上面；
2. 两个智能体都在下面。

我们如何评价这两个均衡的好坏呢？我们可以看所有人的代价之和，事实上如果是真实的交通，那这就意味着整体拥塞程度。我们发现，第一种均衡的总代价是 3，第二种则是 4。如果我们继续检查其它所有的策略组合，会发现这是仅有的两个纳什均衡，因此第二种是最差的，并且我们会发现第一种的代价是无论是否均衡的所有策略组合中总代价最小的。我们可以定义一个衡量网络设计的指标 POA (Price Of Anarchy, 无秩序代价)，定义为最坏情况均衡分流的代价比上最优分流的代价。显然这一简单的网络的 POA 等于 $4/3$ 。事实上 POA 是非常直观的，它衡量了一个网络的设计是否合理，如果 POA 过大，那么代表即使是达到纳什均衡这种比较稳定的情况，所有人的代价总和也会很大，因此网络设计可能需要调整。当然，很幸运的是，我们可以证明如下结论：

定理 12.11 在代价函数为仿射函数（即一次函数）的单元自私路由网络中，POA 至多为 $5/2$ 。

证明感兴趣的读者可以参考《算法博弈论二十讲》12.5 节，然后你就会发现，这一定理的证明和之前最大割的证明套路是一模一样的：首先根据均衡分流定义对每个智能体 i 写一个不等式，然后求和，最后经过一些巧妙的变换证明出结果。因此这里的 POA 事实上和近似比也有了些有趣的联系，我们可以将任何一个均衡视为一个可行解，我们希望能找到一个好的均衡逼近最优均衡（当然这里的说法可能不严谨，因为最优分流不一定是均衡分流）。

反过来，我们也可以将最大割问题可以描述为一个博弈，我们可以将问题描述为 $|V|$ 个智能体希望分成两边，使得两边的人之间的那些连线权重最大化。如果每个人都只看到自己而不看全局，则可能会陷入一个不是最优解的纳什均衡：在纳什均衡出现的时候，每个人都觉得自己站在了正确的一边，因为自己如果换到对面会让总权重降低，但这样每个人自私的行为，在我们之前已经证明只是 2-近似的全局最优——很遗憾，个人理性的综合并非集体理性，我相信这不仅是学术上有用的启示，对于现实生活也是一个很好的观察。个体理性无法达到集体理性最著名的例子便是囚徒困境，我想这也是博弈论迷人的地方，它不仅具有学术上的挑战，同时也能告诉我们理性的边界，理性的力量，理性的缺陷。

下面我们讨论如何计算纳什均衡。事实上在最大割问题中，我们定义了一个势函数 $\Phi(S)$ ，当势函数降低到局部最优的时候，表明此时更改单独一个点所在的集合是无法获得更好的点的，也就是说达到了纳什均衡。那么这一势函数的局部最优解为什么会与纳什均衡有关呢？事实上我们可以分析一下，对于两个不同的状态 S 和 S' ，其中差别只在于点 u 从集合 A 换到了集合 B ，并且我们发现 $\Phi(S') - \Phi(S)$ 就等于 u 的满意度的变化，也就是说，势函数的变化值就等于智能体效用的变化值。因此当势函数到达某个局部最优值的时候，智能体不可能通过改变策略让自己效用值更优，因为周围的所有策略势函数值都会变差，因此效用也只能变差。

对于单元自私路由网络，我们也可以定义势函数：

$$\Phi(f) = \sum_{e \in E} \sum_{i=1}^{f_e} c_e(i),$$

其中 f 是一个分流向量，即策略组合。不难验证这一函数的定义符合势函数定义，即满足

$$\Phi(s'_i, s_{-i}) - \Phi(s_i, s_{-i}) = C_i(s'_i, s_{-i}) - C_i(s_i, s_{-i}),$$

其中 C_i 是智能体 i 的成本。我们称有势函数的博弈称为势博弈，很显然，势博弈具有如下性质：

定理 12.12 所有势博弈都存在纳什均衡。

证明很简单，就是因为我们之前已经说了势函数的局部最优就对应于纳什均衡。因此我们求解纳什均衡的过程就可以通过局部搜索实现，这一局部搜索过程实际上就是每次改变一个智能体的策略使得势函数降低，直到降低到一个局部最优值，这一过程我们也称其为最优反应动力学。

有一个自然的问题，即所有的博弈都有相应的势函数吗？答案是否定的，事实上博弈存在势函数当且仅当它同构于一个拥塞博弈，这里也就不再多介绍了，感兴趣的读者可以自行阅读《算法博弈论二十讲》的相关内容，包括下面的 PLS 完全性的证明。

最后，我们想要知道纳什均衡的局部搜索求解的时间复杂度如何，很遗憾，它和前面讨论的 Hopfield 神经网络以及最大割问题一样，都属于 PLS 完全问题，因此目前我们还没有找到一个多项式时间的算法来解决这一问题：

定理 12.13 对拥塞博弈（也就是全体有势函数的博弈）计算纯策略纳什均衡是一个 PLS 完全问题。

事实上前面的均衡中，每个人只能从策略集合中选择一个确定的策略，我们称前面的纳什均衡称为纯策略纳什均衡；但有时候我们会允许选择策略集合的一个概率分布，这样得到的纳什均衡称为混合策略纳什均衡。关于混合策略纳什均衡我们无法展开描述，因为会复杂很多，但我们有一些著名的结论可以展示：

定理 12.14 任意一个有限博弈（参与人有限、每个参与人的策略有限）都有混合策略纳什均衡。

证明需要用到角谷不动点定理等，超出了我们的讨论范围。类似于纳什均衡的 PLS 完全性，一些重要的博弈的混合策略纳什均衡的计算是 PPAD 完全的，例如最经典的双人、三人矩阵博弈，其中双人则是由现在北京大学的邓小铁教授和他当年的学生陈汐共同证明的。事实上，在算法博弈论中还有非常多亟待解决的问题留待后人探索，这里的介绍也仅仅是一个导引，具体的可以阅读《算法博弈论二十讲》或是 Nisan 的算法博弈论专著等。