大家好, 今天主要讲一下优化方法的一些介绍. 刚入学的时候，很多东西也不会, 走过了很多弯路，今天来总结一下可能在科研中会用到的一些方法。

最开始接触的是合作博弈，也不知道啥是合作博弈，不知道和线性规划啥关系，最开始的时候没什么兴趣，因为之前上过运筹学觉得里面的问题和解法都比较有意思, 而合作博弈这个东西看上去与这些算法都没什么关系. 当时没有足够多的动力和兴趣去研究课题, 问题主要是没有足够的基础知识铺垫, 没有看到背后的优化问题和算法。

这一次借助这个机会, 我就把我走过的一些弯路 做过的一些尝试 总结一下， 希望大家能对自己的工作感兴趣，然后提高学习科研的热情。

因为很难从现有的文章中得到一些基本的优化问题和算法，但这部分又是必需的。所以呢，今天来大致讲一下优化里面都有哪些方法和应用。

内容呢 主要包括五个部分: 刚开始希望通过动态规划进行简单地引入 第二部分是整数和线性规划. 第三部分是一般的优化方法 分为精确算法和近似算法. 第四部分是有关组合优化的内容. 第五部分 是比较杂的部分一般的优化问题.

首先是动态规划 我想以一个简单的问题作为切入点，也就是背包问题。这样说一般是0-1背包问题。背包问题就是将物品装进背包，每个物品有自己的重量w和价值v，背包有一个容量限制W,如何选择物品使得在不超过背包容量的前提下装进背包物品的价值达到最大。 我们可以将这个问题建模为一个整数规划模型。 有哪些解法呢？ 首先这是一个NPC问题，什么是NPC问题我们后面再说。也就是这个问题是不存在一个多项式算法得到最优解的。如果规模小，我们可以用枚举法,小时候肯定做过类似的题. 规模大的话存在伪多项式时间的动态规划算法。

Partition Equal Subset Sum

假设 物品重量w和容量W 是正整数 定义 m[i,w] 装了i件物品总重量不超过w时的最大价值. 那么我们可以得到下面的迭代式 当不装任何物品时,价值为零 事实上这是一个边界条件 当要装的第i个物品重量大于w时,不能装i,显然有 i=i-1 容量可以装下第i个物品时,则需要比较 装和不装 两种方案 取最大

下面这个图表示了计算流程: 第一行表示容量限制 列表示装入物品的件数 先计算第一行,再根据第一行计算第二行 以此类推. 一直计算到 最后一行最后一格就是我们需要的价值.

那么动态规划呢，它是一种算法思想，不是具体的算法。 动态规划可以大大会减少计算量,这一点非常关键，也是动态规划应用非常广泛的原因. 理解动态规划在某些问题中它的计算复杂度为什么不是指数的，这个是灵活运用的关键。

首先是最优子结构的问题，即子问题最优也是可以通过和原问题同样的方法求解出来的。比较简单的例子是斐波拉契数列，比如求斐波拉契数列的第n个数，这样的关系式对于求解任何小于n的数都是适用的。这个是我们能够利用动态规划的关键.

准确提取和定义问题的状态能够帮助我们快速建立起关系式，从而使用动态规划求解。在数列中的状态就为第n个数，在背包中状态就为所装物品的数量以及背包的容量。

除了在背包问题中有比较经典的应用，动态规划另一个经典的应用是最短路问题.

基本的运筹学中,我们知道迪杰斯特拉算法能够求解非负权重的最短路问题. 那么这个算法也可以简单看成一个动态规划的过程.在原有的路径基础上不断地加入新的路径直到到达终点. 之所以能这么做是因为

最短路是有最优子结构的性质.

下面大概讲一下最短路问题 图中是一个有向图(V,A), V节点集合,A边的集合 源节点s 目标节点t 对于边(i,j)的成本 w(i,j) 对每条边有一个整数变量x表示是否经过这条边.

根据上面的定义, 它的整数规划模型表达式就可以写出如下:

虽然模型能这样写出来,但我们知道大规划整数规划比较难求解,通常我们都不会直接去直接解这个整数规划. 然后我们看一下是用动态规划是怎么处理的?

首先需要给出一个结论: 最短路问题是可以证明任何最短路的子路径相对于子问题的 起点，终点的路径也是最短的。 这其实对应到 最优子结构的性质。 如何理解? 就是S 到 C的最短路是最短路的一个子路径 或者 T到B 也是一个子路径

针对这个具体的例子: 找到图中从s到t的最短路径,方法可以是:

依次 考虑从 C 到 T 的最短距离。 考虑从 B 到 C 的最短距离 考虑从 A 到 B 的最短距离 考虑从 S 到 A 的最短距离 最终得到的就是从S到T的最短距离。

写出来的话 就是这么四个阶段, 阶段一C到T的最短,阶段二在阶段一的基础上求最短. 那么我们的最短路就是 min F(S)

可以像前述一样做一个表格,将所有的阶段和状态组合对应的路径值储存在表格中.

所有需要记录的值 共sum(|S|,|A|,|B|,|C|) 同时用u,d表示路径,这样就可以得到最短路径如图中的红线, 当然你可以使用不同的方法进行定义.

这样定义了状态,阶段之后，这个最优阶段序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优阶段序列。这样便可以利用动态规划进行求解。

总结一下动态规划:

先判断该问题是不是能用动态规划进行求解,

1）分析该问题的最优子结构, 有的话就可以使用,

如果没有的话, 在不同问题中动态规划得到的解不是最优解，原因在于他不满足上面提到的最优子结构，因此要注意动态规划求解的条件。

接着是如何定义状态呢？选取好的状态,能够帮助我们快速求解.

2）递归定义最优解的值

2）自底向上计算最优解的值

写动态规划的时候看似递归是从上到下的,但是编程的时候不要用递归,实际上的计算过程是从自下而上从底到顶的.

动态规划：通常与递归相反，其从底部开始解决问题。将所有小问题解决掉，进而解决的整个问题。

3）多阶段 计算得到最优解

在整个过程中，我们把 我们的目标问题转化成了一个个的子问题，在子问题 求 最小值，最后解决了这个问题。

所以求解过程是一个多阶段的过程，这是与静态规划所不同的,静态规划是在可行解中不断改进修正找到最优解. 动态规划每一步处理一个子问题，

多阶段决策过程，每步求解的问题是后面阶段求解问题的子问题，每步决策将依赖于以前步骤的决策结果。

这种多阶段的性质可用于求解组合优化问题。适用条件问题需要满足最优子结构性质。

下面我们讲整数和线性规划

我们回头看最短路问题的整数规划模型，可以看出对应的系数矩阵是 totally unimodular matrix, 也就是全单模的. 这个概念非常重要,network 里的很多基础问题系数矩阵是全单模的. 最短路 是 最小费用网络流问题 的特殊情况. 最大流问题是另一种特殊情况. 还有指派问题 表达式写出来也是全单模的.

该矩阵的特点是:

如果 $A \in R^{m\times n}$是整数矩阵，r =rank(A)

而且A的所有非零r×r子式等于 1 或-1，则称A为幺模矩阵, 如果A是幺模矩阵，而且还有其各阶子式均等于0，1或-1，则称A为全幺模矩阵. 不能从定义来判断 A是否全单模.

显然，全幺模矩阵的所有元素均为0，1或-1。

Totally Unimodular 有三个条件/或者说性质(充分判定)

1. 元素均为 +1 -1 0 由定义得到 各阶子式是0,1,-1
2. 每一列只有不超过两个的非零元素
3. 行可以分为两组，对于每一列这两组数相加相等。

当然这里面会牵扯到非常多的比较难的数学知识，但对我们来说重要的是，如果一个整数规划它的矩阵是 totally unimodular, 那么我们就可以利用线性规划来求解它，这是一个非常重要的性质。 我们自己写一下就能够看出 最短路是满足这个条件的

同时呢, 如果一个线性规划 系数矩阵 A 是TU, 它的可行域的极点就是整数的,因此可行域就是整数的 polyhedron.

正反两方面 整数规划 用 线性规划求解,是线性规划的话 得到整数解

虽然这个性质是可以从 totally unimodular出发得到的，但我们可以从另一个方面来看这里面比较有趣的性质。 回到线性规划的单纯形法解，如果不熟悉就复习一下,我就经常忘.经常复习. 来到最后一步，B基向量 利用克莱姆法则 得到解 x* = (B-1b,0) 矩阵的逆怎么求？B-1 = B*/det(B), 伴随矩阵 代数余子式, 对于 matrix 的逆必然是正负一，b是整数的话 因而解是整数。

好了这部分的整数规划问题可以转化为线性规划进行求解. 那么线性规划大家都很熟悉，最基本的就是单纯形法. 尽管单纯形法本身不是多项式可解（在最坏的情况下）但掌握单纯形法的具体流程还是比较重要的，比如我们上述讲整数规划和线性规划关系的时候就用到了，一些别的算法要在单纯形法基础上进行分析改进，掌握检验数，基变量,非基变量，基本可行解 这些概念对于理解基本的列生成算法，割平面方法也是有帮助的。

这里想提到的就是从另一个角度来看 进基出基过程：

将单纯形法中看成一个迭代过程. 当前解为 x. 是一个多面体的顶点 迭代过程可以看做从多面体的一个顶点到下一个顶点. 设边的方向为λ，我们沿着边走的距离是θ，那么，我们走的就是 x' = x – θλ 那么边的方向λ代表什么呢？ 通常是使得目标函数值减小最大的方向.这其实对应着选择一个非基的向量（进基过程,通过检验数来选取），可以理解为梯度方向。 θ就是步长，步长越大越好（出基过程），最大只能走到可行域的顶点。 https://www.hrwhisper.me/introduction-to-simplex-algorithm/

为什么要讲这个呢，因为一直觉得单纯形法很另类，要做单纯性表,要进行一系列匪夷所思的操作,觉得单纯形法的解法和一般的优化问题不太一样，但实际还是可以看出是一个梯度下降的过程。

下面来看一下有哪些求解MILP的优化方法。

分别有精确算法，启发式算法和近似算法。精确算法一般是 分支算法一大类，最基本的分支方法:

1. 分支定界: 利用定界来减少计算量的方法，
2. 分支割可以理解为不断生成割平面添加约束，是一种分支结合割平面的方法
3. 分支price是分支定界和列生成的结合，

图中只给了分支的过程,对于整数变量一分为二. 定界的过程是(对于求最小值) 如果另一支的线性解得到的目标值大于当前的整数解(即上界) 则将该支进行剪切(prune)

注意分支定界虽然是精确算法 但由于是一个迭代算法，问题规模大的时候存在收敛的问题，因而通常设置gap 提供终止条件。

割平面方法 一般是如图所示 需要找到足够紧的约束(plane) 如图中绿线,加入这些绿线去不断地改进解的质量, 直到得到的最优解是整数解,则得到了原问题的最优解.

那么如何找到这样的割平面呢? Gomory cut: 使用单纯形法求解原问题的松弛问题,得到最后一张单纯形表中问题的表达式为 xi+xj, 其中 xi 为基变量 xj 为非基变量 若此时bi 为整数,则得到的就是整数解. 所以bi不为整数时. 重写上式,将整数部分放在左边, 分式写在右边,得到下式.

右边小于1,并且左边是整数,则必然有右边部分是小于0的,即得到了一个有效的不等式,一定切掉了原可行域的一部分. 添加新的松弛变量xk进入单纯形表继续进行计算. 直到bi为整数.

原始问题的一个表述,抽象出主问题,松弛原来的可行域形成限制主问题. 子问题通常用于产生新的列，难点一般在子问题 pricing problem 如何求解. (seperation probelm 一般是行生成中的子问题)。 行生成的经典问题是 cutting stock. 子问题用于得到切割方案作为列加入到主问题中. 对于不同的问题,则需要具体分析. 根据具体问题分支用于减少不必要的计算量.

branch and price 大家听讲座讲的比较多了,了解基本过程之后可以试试简单的例子，做一下总结。这样遇到问题可能需要这个方法的时候稍微想一下就可以很快用上。

动态规划前面讲了是一种算法思想，掌握它的优化条件，即什么问题适合利用该方法能够得到最优解.

约束编程问题 或是约束满足问题 我的理解是 即不需要找到一个最优解，而是要找到满足约束的解 著名的八皇后问题 图着色问题 这种一般是能够建立起满足条件的约束 得到满足题意的解就可以，因而认为一种精确的算法，当然这部分也可以结合割平面，分支定界，动态规划进行求解。

枚举 不详述.

接着是近似方法，近似方法分为启发式算法和近似算法。

基本所有的启发式都是为了跳出局部最优解。

启发式策略（heuristic）是一类在求解某个具体问题时，在可以接受的时间和空间内能给出其可行解，但又不保证求得最优解（以及可行解与最优解的偏离）的策略的总称。许多启发式算法是相当特殊的，依赖于某个特定问题。当寻求问题的最优解变得不可能或者很难完成时（e.g. NP-Complete 问题），启发式策略就是一个高效的获得可行解的办法。这是一类“problem-specific”的策略

启发式算法是相对“最优算法”而言的，其目标是在某种启发原则的引导下搜寻解（这种解一般是局部最优，但可以很大程上接近最优）；贪心算法的核心——贪心准则就是一种启发原则。

“元启发式算法” 就是：用来构造启发式算法的一些基础方法。元是一种算法框架，

元启发式算法（MetaHeuristic Algorigthm）是启发式算法的改进，它是随机算法与局部搜索算法相结合的产物.

元启发式主要分为 基于单个解的元启发式 和 基于群体（Population-based heuristics）的启发式方法.

近似算法是一大类，大家平常见到的 1+\epsilon O(\epsilon^-1) 这类有近似比的算法表述就是近似算法，它的特点是能够表示出算法得到的解与最优解之间的误差，需要的分析知识较多，证明方法多样非常灵活，要求比较高。在调度问题，设施选址问题中出现的比较多。

基于单个解的元启发式 模拟退火（Simulated Annealing Algorithm） 禁忌搜索（Tabu Search） 变邻域搜索（Variable Neighborhood Search） 自适应大规模领域搜索（Adaptive Large Neighborhood Search）

没有在自己做的研究中用过,不是很熟悉,可以了解一下基本思想.

基于群体（Population-based heuristics）的启发式方法

1、遗传算法（Genetic Algorithm） 2、蚁群算法 3、粒子群算法

利用群体的性质进行解的演化跳出局部解

总结如图所示.

说完了不同的算法区别，我们再来看看组合优化 组合优化中有非常多的经典问题 像是TSP，VRP，FLP， Production Scheduling Problem

这些问题呢都是组合优化问题，组合优化问题有些是可以转化的,按照不同的定义和建模方式可以变成不同的问题,比如之前听到的 VRP的报告,就可以变成一个集合覆盖的问题. 设施选址问题也可以建模成集合覆盖问题.

组合数大家都知道是比指数增长还要快的，组合优化存在大量的离散变量组合的结果，可行解数量是随着规模的变大而迅速增长的。要选出一个最优解就变得极为困难。因而找出一种可以高效率解决组合优化问题的算法是大家一直追求的目标。

说到组合优化不得不提到的概念就是NP-难，首先需要了解 P 和 NP的概念， P类问题：所有可以在多项式时间内求解的判定问题构成P类问题。 判定问题或者说是决策问题：判断是否有一种能够解决某一类问题的算法。 简单来说也就是 是否存在满足某种条件的问题的解，答案是Yes or No.

NP类问题：所有的非确定性多项式时间可解的判定问题构成NP类问题。 非确定性算法：非确定性算法将问题分解成猜测和验证两个阶段。算法的猜测阶段是非确定性的，算法的验证阶段是确定性的，它验证猜测阶段给出解的正确性。

设算法A是解一个判定问题Q的非确定性算法，如果A的验证阶段能在多项式时间内完成，则称A是一个多项式时间非确定性算法。有些计算问题是确定性的，例如加减乘除，只要按照公式推导，按部就班一步步来，就可以得到结果。但是，有些问题是无法按部就班直接地计算出来。比如，找大质数的问题。有没有一个公式能推出下一个质数是多少呢？这种问题的答案，是无法直接计算得到的，只能通过间接的“猜算”来得到结果。这也就是非确定性问题。而这些问题的通常有个算法，它不能直接告诉你答案是什么，但可以告诉你，某个可能的结果 是正确的还是错误的。这个可以告诉你“猜算”的答案正确与否的算法，假如可以在多项式（polynomial）时间内算出来，就叫做多项式非确定性问题。

简单来说就是 NP 对于 验证是对的 来说呢 是多项式容易的。但是算法是不确定的，即求解较难。

比如说 判断给定 一个数N的分解 是否等于两个质数的乘积 K * L 很容易判断是否。但是让你求解 N 是否为质数就是一个非确定性问题。

NPC问题：所有的NP问题都可以在多项式时间内转化/归约为 P \in NP，则P是最难的问题。 称 P \in NPC. 这些问题被称为NP-完全问题(NPC问题)。

NP-hard 至少与 NP 中最难的问题一样难. 是指优化问题 对应的 决策问题是 NPC 完备的。

那么像这样的组合问题有哪些解法呢？分支定界，割平面，行生成，列生成，彼此互相组合衍生出来的更为高效的算法，或者结合启发式算法得到近似解, 需要具体问题具体分析，但需要掌握最基本的原理。

总结: 最后，我们应该掌握如何用这种方式解决问题，这样我们就有了更多的想法，进而尝试不同的方法。

理论上，精确的方法可以保证找到全局解，但在解决一些典型的实际问题时，代价太高。因为在保证找到最优解的前提下，很难对算法进行加速。

对于大多数实际问题，很难找到多项式算法，因为它们大多是NP难问题，那么剩下的选择就是设计一个能够跳出局部最优的算法。

接下来分析一般的优化问题，最初人们认为，优化问题是线性还是非线性，是不同的优化问题的根本区别。后来认为 优化问题是凸还是非凸才是不同的优化问题之间的根本区别。从中看出来线性优化和凸优化是非常重要的.

凸优化问题为什么会像线性优化这样重要呢，因为通常我们认为 凸优化可以求得最优解，所以在应用上非常重要。

Definition: 凸集概念 一个集合是凸集就是对于集合中的两个元素 x,y 连线上的元素也属于这个集合. 凸函数 是一个定义在某个向量空间的凸子集S上的实值函数f 满足下面这个条件. 也就是下凸. 凸优化问题（OPT，convex optimization problem）指定义在凸集中的凸函数最优化的问题。 要求目标函数是凸函数，变量所属集合是凸集合的优化问题。或者目标函数是凸函数，变量的约束函数是凸函数（不等式约束时），或者是仿射函数（等式约束时）。

当 f(x) 和 g_i(x) 均为凸函数，而h_j(x)均为仿射函数时, 上述的优化问题即凸优化问题。

尽管凸优化的条件比较苛刻，但仍然众多领域有十分广泛的应用，比如说在机器学习中。 因为通常我们认为 凸优化因为可以求得最优解，所以在应用上非常重要。

https://www.jianshu.com/p/6a962fb1b4e0

凸优化的标准问题主要有这么几类：

1. Linear Programming(LP)

$$\begin{aligned} \min \quad & c^Tx+d \\ s.t. \quad & G(x) \preceq h \\ & A(x) = b \end{aligned}$$

其中目标函数和不等式约束都是仿射函数，且\preceq表示按元素小于等于。

2. Quadratic Programming(QP) 线性约束的二次规划

\begin{aligned} \min &\quad \frac{1}{2}x^TPx+c^Tx+d \ s.t. &\quad G(x) \preceq h \ &\quad A(x) = b \end{aligned}

其中目标函数为凸二次型，不等式等式约束为仿射函数。

3. Semidefinite Programming(SDP) 半正定规划

\begin{aligned} \min &\quad tr(CX) \ s.t. &\quad tr(A_iX)=b_i, i=1,2,…p \ &\quad X \succeq 0 \end{aligned}

具有线性目标函数，定义在“半正定矩阵构成的锥体”与“仿射空间”的交集上，即光谱面。

其中需要最优化的变量X是一个对称的半正定矩阵，且C, A_1,…,A_p为对阵矩阵。

4. Cone Programming(CP)

那么还有其他的一些问题是凸优化问题或可以通过改变变量而转化为凸优化问题：

Least squares – 最小二乘 SVM – 支持向量机 机器学习里的分类问题 用于最大化间距 QCQP – 二次约束的二次规划 其中目标函数和不等式约束都是凸二次型。 SOCP 二阶锥规划 通常也要求正定 GP 几何规划 多项式乘积的形式 CP 锥规划的概念比较广 理解为在 锥体 空间上的优化问题

\begin{aligned} \min &\quad \frac{1}{2}x^TPx+c^Tx+d \ s.t. &\quad \frac{1}{2}x^TQ_ix + r_ix + s_i \leq 0, i=1,2,…m \ &\quad A(x) = b \end{aligned}

凸优化问题的关系大致如下: 从线性规划 线性约束的二次规划 二阶锥规划 半正定 最后是锥规划

凸优化有哪些性质让它成为了一类重要的优化问题呢?

1. 凸优化问题有一个重要的特性，就是凸优化问题的局部最优解就是全局最优解。
2. 如果目标函数是严格凸的,那么该问题最多只有一个最优点.
3. 很多非凸问题都可以被等价转化为凸优化问题或者被近似为凸优化问题（例如拉格朗日对偶问题）比如凸松弛方法.

4. 凸优化问题的研究较为成熟，有很多方法是现成的,能够被用来求解. 当一个问题被归为一个凸优化问题，基本可以确定该问题是可被求解的。

凸优化（凸最小化）问题通常是用迭代的方法求解

首先是最基本的梯度下降里的 线性搜索 由于凸优化问题具有局部最优解即全局最优解的优良特性，因此求解过程可以简化为：找到一个点列使得目标函数值持续减少，直到触发停止条件或达到一个最小值。

一般求解过程就是：计算一个下降方向, 选取一个步长最小化当前方向的函数值 对当前节点更新,直到梯度小于可以接受的误差.

梯度下降 最主要的步骤呢 就是上面所说的,再求最小值的时候,迭代更新得到不断减小的函数值, 求最大的时候就是梯度上升.

次梯度,一般针对不可导的情况, 注意收敛条件和步长的规则即可.

前面我们提到了单纯形法中旋转操作可以通过一个迭代过程表示出来. 我们也提到了进基出基的过程,对应到梯度和步长的概念. 梯度算法重要的两个概念就是 步长和梯度，有一系列分析的条件去讨论这里面的收敛性. 但如果我们只是拿来用的话,就不用分析这里面的收敛性.

对于梯度算法中的关键两点梯度和步长,根据不同问题的性质衍生出来了各种各样的梯度方法.

随机梯度方法， 批量梯度方法, 在机器学习中进行训练数据用的非常多,主要用于计算快,提高收敛速度. 近端梯度法, 主要用于求解目标函数不可微的最优化问题 牛顿法 是二阶方法,收敛快,计算量大. 共轭梯度 它仅需利用一阶导数信息, 避免了牛顿法需要计算Hesse矩阵并求逆的缺点，有较快的收敛速度.是解大型非线性最优化非常有效的算法. 拟牛顿法 使用正定矩阵 来近似Hessian矩阵的逆 对牛顿法进行了改进,简化了计算.

另一类方法就是 关于内点法或是罚函数法，或是拉格朗日乘子法. 那么说到这些 就不得不提到 KKT条件 大家都非常熟悉 有下面三点 Stationarity 满足驻点条件,x一阶导为零 Primal feasibility Dual feasibility 原始对偶可行性 Complementary slackness 对不等式的 互补松弛性 当没有不等式约束的时候呢,只有h(x), KKT 条件就是拉格朗日条件 乘子就是拉格朗日乘子.

接着讲 KKT 的对偶性质. 加入乘子项之后, 显然 g() 是原问题的一个下界 因为λ>0, fi<0. 我们想最小化这个gap 就最大化g(). 右边实际上就是原问题 对偶性质就是 最小值的最大 < 最大值的最小.

当满足强对偶条件时,对偶问题和原问题的解相等. 至少凸优化问题是满足强对偶性质的.

强对偶的限制条件呢，如斯莱特的条件成立.

其他的一些方法 比如 内点法, 罚函数法, 拉格朗日乘子,增强拉格朗日,admm 交替方向乘子 我的理解都是基于引入乘子,然后根据不同问题的具体情况提出了一些方法. 理解了上面的过程, 这些方法到时候看一下 理解个大概就可以.

非凸优化的方法

通常非凸函数会有较多的驻点, 上面提到的各种方法只能得到一个局部解.

那么解决方法: 找到所有局部最优 指数个不现实,

凸松弛 将非凸 转化为凸优化问题, 以及采用启发式算法，有一定的局限性，一般靠经验调整参数。那么重点在于要能够设计出跳出局部最优解的算法。

凸松弛（一般是整数规划）首先把问题定义域~X的范围从整型松弛到实值范围内，而且构造一个新的目标函数在定义域上是凸的且小于或者等于原目标函数。或者是将函数近似为一个分段线性函数 piecewise linear.

最后是随机优化有哪些方法? 我就大概分一下类,这部分我也不太了解. 随机规划: 蒙特卡洛方法取样 统计中用的多一点. 鲁棒优化 通常的优化中 参数是确定的, 当参数在给定的集合内发生变化时，仍能确保优化方案是可行的. 随机动态规划

随机搜索方法: 模拟退火 随机爬山 之前提到的进化算法 基于种群的算法.

总结:

对于大多数实际问题很难找到多项式算法，因为他们大多是NP难问题,很难保证找到全局解, 或者代价太大，又或者容易陷入局部最优。那么剩下的选择就是设计出能够跳离局部最优的算法。在实际生活中，基本不存在能够利用精确算法求解的问题，大多都是利用近似方法求解的，在这里面精确算法和启发式方法都发挥重要的作用。

总之我这里只是起了抛砖引玉的作用，还需要大家在平常科研学习中多进行总结，进行知识的更新迭代。问题的性质可以促进你想到相应的算法，从而可以有选择的余地。

了解了这些方法之后, 最终能够达到遇到一个问题，如何把它抽象建模出来，建模这个能力也需要大家多进行学习总结。也就是定义了这个问题之后，判断这个问题属于哪一类优化问题，知道要用那种方法去解决，有哪些定理保证了可以这么做，为什么要使用这个方法，即这个方法的优点。这样遇到问题就会有较多的思路，可以多尝试不同的方法。

希望大家能够掌握这些基本的优化方法，之后可以更好地进行深入的科研学习。