计算智能

一、 粒子群算法

粒子群优化（**Particle Swarm Optimization, PSO**），又称粒子群算法、微粒群算法，是由 J. Kennedy 和 R. C. Eberhart 等于1995年开发的一种演化计算技术，来源于对一个简化社会模型的模拟。其中“群（swarm）”来源于微粒群符合 M. M. Millonas 在开发应用于人工生命（artificial life）的模型时所提出的群体智能的5个基本原则。“粒子（particle）”是一个折衷的选择，因为既需要将群体中的成员描述为没有质量、没有体积的，同时也需要描述它的速度和加速状态。

PSO 算法最初是为了图形化地模拟鸟群优美而不可预测的运动。而通过对动物社会行为的观察，发现在群体中对信息的社会共享提供一个演化的优势，并以此作为开发算法的基础。通过加入近邻的速度匹配、并考虑了多维搜索和根据距离的加速，形成了 PSO 的最初版本。之后引入了惯性权重*w*来更好的控制开发（exploitation）和探索（exploration），形成了标准版本。为了提高粒群算法的性能和实用性，中山大学、（英国）格拉斯哥大学等又开发了自适应（Adaptive PSO）版本[2]和离散（discrete）版本

PSO 算法属于一种万能启发式算法，能够在没有得知太多问题信息的情况下，有效的搜索具有庞大解空间的问题并找到候选解，但同时不保证其找到的最佳解为真实的最佳解。

算法原理

PSO 算法是基于群体的，根据对环境的适应度将群体中的个体移动到好的区域。然而它不对个体使用演化算子，而是将每个个体看作是 D 维搜索空间中的一个没有体积的微粒（点），在搜索空间中以一定的速度飞行，这个速度根据它本身的飞行经验和同伴的飞行经验来动态调整。第 i 个微粒表示为 Xi =(xi1, xi2, ..., xiD) ，它经历过的最好位置（有最好的适应值）记为 Pi = (pi1, pi2, ..., piD)，也称为 pbest。在群体所有微粒经历过的最好位置的索引号用符号 g 表示，即 Pg，也称为 gbest 。微粒 i 的速度用 Vi = (vi1, vi2, ..., viD) 表示。对每一代，它的第 d+1 维（$1 \le d+1 \le D$）根据如下方程进行变化：

```
        vid+1 = w•vid+c1•rand()•(pid-xid)+c2•Rand()•(pgd-xid)
(1a)
        xid+1 = xid+vid+1                                   (1b)
```

其中w为惯性权重（inertia weight），c1和c2为加速常数（acceleration constants），rand() 和 Rand() 为两个在[0,1]范围里变化的随机值。

此外，微粒的速度 Vi 被一个最大速度 Vmax 所限制。如果当前对微粒的加速导致它的在某维的速度 vid 超过该维的最大速度 vmax,d，则该维的速度被限制为该维最大速度 vmax,d。

对公式（1a），第一部分为微粒先前行为的惯性，第二部分为“认知（cognition）”部分，表示微粒本身的思考；第三部分为“社会（social）”部分，表示微粒间的信息共享与相互合作。

“认知”部分可以由 Thorndike 的效应法则（law of effect）所解释，即一个得到加强的随机行为在将来更有可能出现。这里的行为即“认知”，并假设获得正确的知识是得到加强的，这样的一个模型假定微粒被激励着去减小误差。

“社会”部分可以由 Bandura 的替代强化（vicarious reinforcement）所解释。根据该理论的预期，当观察者观察到一个模型在加强某一行为时，将增加它实行该行为的几率。即微粒本身的认知将被其它微粒所模仿。

PSO 算法使用如下心理学假设：在寻求一致的认知过程中，个体往往记住自身的信念，并同时考虑同事们的信念。当其察觉同事的信念较好的时候，将进行适应性地调整。

标准 PSO 的算法流程如下：

1. 初始化一群微粒（群体规模为 m ），包括随机的位置和速度；
2. 评价每个微粒的适应度；
3. 对每个微粒，将它的适应值和它经历过的最好位置 pbest 的作比较，如果较好，则将其作为当前的最好位置pbest；
4. 对每个微粒，将它的适应值和全局所经历最好位置 gbest 的作比较，如果较好，则重新设置gbest的索引号；
5. 根据方程（1）变化微粒的速度和位置；
6. 如未达到结束条件（通常为足够好的适应值或达到一个预设最大代数 Gmax ），回到（2）。

算法参数

PSO 参数包括：群体规模 m ，惯性权重 w ，加速常数 c1 和 c2 ，最大速度 Vmax，最大代数 Gmax。

Vmax 决定在当前位置与最好位置之间的区域的分辨率（或精度）。如果 Vmax 太高，微粒可能会飞过好解，如果 Vmax 太小，微粒不能进行足够的探索，导致陷入局部优值。该限制有三个目的：防止计算溢出；实现人工学习和态度转变；决定问题空间搜索的粒度。

惯性权重w使微粒保持运动的惯性，使其有扩展搜索空间的趋势，有能力探索新的区域。

加速常数 c1 和 c2 代表将每个微粒推向 pbest 和 gbest 位置的统计加速项的权重。低的值允许微粒在被拉回来之前可以在目标区域外徘徊，而高的值导致微粒突然的冲向或者越过目标区域。

如果没有后两部分，即 c1 = c2 = 0，微粒将一直以当前的速度飞行，直到到达边界。由于它只能搜索有限的区域，将很难找到好的解。

如果没有第一部分，即 w = 0，则速度只取决于微粒当前的位置和它们历史最好位置 pbest 和 gbest ，速度本身没有记忆性。假设一个微粒位于全局最好位置，它将保持静止。而其它微粒则飞向它本身最好位置 pbest 和全局最好位置 gbest 的加权中心。在这种条件下，微粒群将统计的收缩到当前的全局最好位置，更象一个局部算法。

在加上第一部分后，微粒有扩展搜索空间的趋势，即第一部分有全局搜索的能力。这也使得w的作用为针对不同的搜索问题，调整算法全局和局部搜索能力的平衡。

如果没有第二部分，即 c1 = 0，则微粒没有认知能力，也就是“只有社会（social-only）”的模型。在微粒的相互作用下，有能力到达新的搜索空间。它的收敛速度比标准版本更快，但是对复杂问题，比标准版本更容易陷入局部优值点。

如果没有第三部分，即 c2 = 0，则微粒之间没有社会信息共享，也就是“只有认知（cognition-only）”的模型。因为个体间没有交互，一个规模为m的群体等价于m个单个微粒的运行。因而得到解的几率非常小。

收敛性

收敛性的数学证明帮助了 PSO 的发展和应用[4], 但此类分析具有很大的局限性.[5] 为 PSO 加入正交学习后，算法的全局收敛、收敛精度及鲁棒可靠性都得到了提高.[6]

二、 蚁群算法

蚁群算法（**A**nt **C**olony **O**ptimization，**ACO**），又称蚂蚁算法，是一种用来在图中寻找优化路径的机率型算法。它由Marco Dorigo于1992年在他的博士论文“Ant system: optimization by a colony of cooperating agents”中提出，其灵感来源于蚂蚁在寻找食物过程中发现路径的行为。蚁群算法是一种模拟进化算法，初步的研究表明该算法具有许多优良的性质。针对PID控制器参数优化设计问题，将蚁群算法设计的结果与遗传算法设计的结果进行了比较，数值仿真结果表明，蚁群算法具有一种新的模拟进化优化方法的有效性和应用价值。

- https://ws.wiki.fallingwaterdesignbuild.com/baike-蚁群算法#历史)

常见的扩展

下面是一些最常用的变异蚁群算法：

精英蚂蚁系统

全局最优解决方案在每个迭代以及其他所有的蚂蚁的沉积信息素。

最大最小蚂蚁系统（MMAS）

添加的最大和最小的信息素量[τmax，τmin]，只有全局最佳或迭代最好的巡逻沉积的信息素。所有的边缘都被初始化为τmax并且当接近停滞时重新初始化为τmax。

蚁群系统

蚁群系统已被提出。

基于排序的蚂蚁系统（ASrank）

所有解决方案都根据其长度排名。然后为每个解决方案衡量信息素的沉积量，最短路径相比较长路径的解沉积了更多的信息素。

连续正交蚁群（COAC）

COAC的信息素沉积机制能使蚂蚁协作而有效地寻解。利用正交设计方法，在可行域的蚂蚁可以使用增大的全局搜索能力和精度，快速、高效地探索他们选择的区域。 正交设计方法和自适应半径调整方法也可推广到其他优化算法中，在解决实际问题施展更大的威力。

收敛

一些版本的算法可以被证明是收敛的（即它能够在有限时间找到全局最优解）。第一个蚁群算法收敛的证据制作于2000年，创建了基于图像的蚁群算法，继而是ACS和MMAS的算法。像大多数元启发式方法一样，估计理论收敛速度是很难的。在2004年，Zlochin和他的同事们表明，COA-type算法在分布算法的叉熵和估计方面能被随机梯度下降法吸收。他们建议这些元启发式方法作为一个“研究性模式”。

应用

蚁群优化算法已应用于许多组合优化问题，包括蛋白质折叠或路由车辆的二次分配问题，很多派生的方法已经应用于实变量动力学问题，随机问题，多目标并行的实现。它也被用旅行推销员问题的拟最优解。在图表动态变化的情况下解决相似问题时，他们相比模拟退火算法和遗传算法方法有优势；蚁群算法可以连续运行并适应实时变化。这在网络路由和城市交通系统中是有利的。

第一蚁群优化算法被称为“蚂蚁系统”，它旨在解决推销员问题，其目标是要找到一系列城市的最短遍历路线。总体算法相对简单，它基于一组蚂蚁，每只完成一次城市间的遍历。在每个阶段，蚂蚁根据一些规则选择从一个城市移动到另一个：它必须访问每个城市一次;一个越远的城市被选中的机会越少（能见度更低）;在两个城市边际的一边形成的信息素越浓烈，这边被选择的概率越大;如果路程短的话，已经完成旅程的蚂蚁会在所有走过的路径上沉积更多信息素，每次迭代后，信息素轨迹挥发。

三、 模拟退火算法

模拟退火（英语：Simulated annealing，缩写作SA）是一种通用概率算法，常用来在一定时间内寻找在一个很大搜寻空间中的近似最優解。模拟退火在1983年为S. Kirkpatrick, C. D. Gelatt和M. P. Vecchi所发明，V. Černý也在1985年独立发明此算法。

简介

模拟退火来自冶金学的专有名词退火。退火是将材料加热后再经特定速率冷却，目的是增大晶粒的体积，并且减少晶格中的缺陷。材料中的原子原来会停留在使内能有局部最小值的位置，加热使能量变大，原子会离开原来位置，而随机在其他位置中移动。退火冷却时速度较慢，使得原子有较多可能可以找到内能比原先更低的位置。

模拟退火的原理也和金属退火的原理近似：我们将热力学的理论套用到统计学上，将搜寻空间内每一点想像成空气内的分子；分子的能量，就是它本身的动能；而搜寻空间内的每一点，也像空气分子一样带有“能量”，以表示该点对命题的合适程度。算法先以搜寻空间内一个任意点作起始：每一步先选择一个“邻居”，然后再计算从现有位置到达“邻居”的概率。

可以证明，模拟退火算法所得解依概率收敛到全局最优解。

演算步骤

初始化

由一个产生函数从当前解产生一个位于解空间的新解，并定义一个足够大的数值作为初始温度。

迭代过程

迭代过程是模拟退火算法的核心步骤，分为新解的产生和接受新解两部分：

1. 由一个产生函数从当前解产生一个位于解空间的新解；为便于后续的计算和接受，减少算法耗时，通常选择由当前新解经过简单地变换即可产生新解的方法，如对构成新解的全部或部分元素进行置换、互换等，注意到产生新解的变换方法决定了当前新解的邻域结构，因而对冷却进度表的选取有一定的影响。
2. 计算与新解所对应的目标函数差。因为目标函数差仅由变换部分产生，所以目标函数差的计算最好按增量计算。事实表明，对大多数应用而言，这是计算目标函数差的最快方法。
3. 判断新解是否被接受，判断的依据是一个接受准则，最常用的接受准则是Metropolis准则：若$\Delta t'<0$则接受S'作为新的当前解S，否则以概率$\exp(-\Delta t'/T)$接受S'作为新的当前解S。
4. 当新解被确定接受时，用新解代替当前解，这只需将当前解中对应于产生新解时的变换部分予以实现，同时修正目标函数值即可。此时，当前解实现了一次迭代。可在此基础上开始下一轮试验。而当新解被判定为舍弃时，则在原当前解的基础上继续下一轮试验。

模拟退火算法与初始值无关，算法求得的解与初始解状态S（是算法迭代的起点）无关；模拟退火算法具有渐近收敛性，已在理论上被证明是一种以概率1收敛于全局最优解的全局优化算法；模拟退火算法具有并行性。

停止准则

迭代过程的一般停止准则：温度T降低至某阈值时，或连续若干次迭代均未接受新解时，停止迭代，接受当前寻找的最优解为最终解。

退火方案

在某个温度状态T下，当一定数量的迭代操作完成后，降低温度T，在新的温度状态下执行下一个批次的迭代操作。

虚拟码（伪代码）

寻找能量 E(s)} 最低的状态 s

```
s := s0; e := E(s)                                  // 設定目前狀態為s0，其能
量E (s0)
k := 0                                              // 評估次數k
while k < kmax and e > emin                         // 若還有時間（評估次數k還不
到kmax）且結果還不夠好（能量e不夠低）則:
    sn := neighbour(s)                                  // 隨機選取一鄰近狀態
sn
    en := E(sn)                                         // sn的能量為E (sn)
    if random() < P(e, en, temp(k/kmax))                // 決定是否移至鄰近狀
態sn
        s := sn; e := en                                        // 移至鄰近狀
態sn
    k := k + 1                                          // 評估完成，次數k加
一
return s                                            // 回傳狀態s
```

四、 遗传算法

遗传算法（英语：Genetic Algorithm，**GA**）是计算数学中用于解决最优化的搜索算法，是进化算法的一种。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等等。

遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）可抽象表示为染色体，使种群向更好的解进化。传统上，解用二进制表示（即0和1的串），但也可以用其他表示方法。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中评价整个种群的适应度，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群。

遗传算法的机理

在遗传算法里，优化问题的解被称为个体，它表示为一个变量序列，叫做染色体或者基因串。染色体一般被表达为简单的字符串或数字符串，不过也有其他的依赖于特殊问题的表示方法适用，这一过程称为编码。首先，算法随机生成一定数量的个体，有时候操作者也可以干预这个随机产生过程，以提高初始种群的质量。在每一代中，都会评价每一个体，并通过计算适应度函数得到适应度数值。按照适应度排序种群个体，适应度高的在前面。这里的“高”是相对于初始的种群的低适应度而言。

下一步是产生下一代个体并组成种群。这个过程是通过选择和繁殖完成，其中繁殖包括交配（crossover，在算法研究领域中我们称之为交叉操作）和突变（mutation）。选择则是根据新个体的适应度进行，但同时不意味着完全以适应度高低为导向，因为单纯选择适应度高的个体将可能导致算法快速收敛到局部最优解而非全局最优解，我们称之为早熟。作为折中，遗传算法依据原则：适应度越高，被选择的机会越高，而适应度低的，被选择的机会就低。初始的数据可以通过这样的选择过程组成一个相对优化的群体。之后，被选择的个体进入交配过程。一般的遗传算法都有一个交配概率（又称为交叉概率），范围一般是0.6~1，这个交配概率反映两个被选中的个体进行交配的概率。例如，交配概率为0.8，则80%的“夫妻”会生育后代。每两个个体通过交配产生两个新个体，代替原来的“老”个体，而不交配的个体则保持不变。交配父母的染色体相互交换，从而产生两个新的染色体，第一个个体前半段是父亲的染色体，后半段是母亲的，第二个个体则正好相反。不过这里的半段并不是真正的一半，这个位置叫做交配点，也是随机产生的，可以是染色体的任意位置。再下一步是突变，通过突变产生新的“子”个体。一般遗传算法都有一个固定的突变常数（又称为变异概率），通常是0.1或者更小，这代表变异发生的概率。根据这个概率，新个体的染色体随机的突变，通常就是改变染色体的一个字节（0变到1，或者1变到0）。

经过这一系列的过程（选择、交配和突变），产生的新一代个体不同于初始的一代，并一代一代向增加整体适应度的方向发展，因为总是更常选择最好的个体产生下一代，而适应度低的个体逐渐被淘汰掉。这样的过程不断的重复：评价每个个体，计算适应度，两两交配，然后突变，产生第三代。周而复始，直到终止条件满足为止。一般终止条件有以下几种：

- 进化次数限制；
- 计算耗费的资源限制（例如计算时间、计算占用的内存等）；
- 一个个体已经满足最优值的条件，即最优值已经找到；
- 适应度已经达到饱和，继续进化不会产生适应度更好的个体；
- 人为干预；
- 以及以上两种或更多种的组合。

算法

- 选择初始生命种群
- 循环
    - 评价种群中的个体适应度
    - 以比例原则（分数高的挑中几率也较高）选择产生下一个种群（轮盘法（roulette wheel selection）、竞争法（tournament selection）及等级轮盘法（Rank Based Wheel Selection））。不仅仅挑分数最高的的原因是这么做可能收敛到局部的最佳点，而非整体的。
    - 改变该种群（交叉和变异）
- 直到停止循环的条件满足.

GA参数

- 种群规模（P,population size）：即种群中染色体个体的数目。
- 字符串长度（l, string length）：个体中染色体的长度。
- 交配概率（pc, probability of performing crossover）：控制着交配算子的使用频率。交配操作可以加快收敛，使解达到最有希望的最佳解区域，因此一般取较大的交配概率，但交配概率太高也可能导致过早收敛，则称为早熟。
- 突变概率（pm, probability of mutation）：控制着突变算子的使用频率。
- 中止条件（termination criteria）

特点

遗传算法在解决优化问题过程中有如下特点：

- 遗传算法在适应度函数选择不当的情况下有可能收敛于局部最优，而不能达到全局最优。
- 初始种群的数量很重要，如果初始种群数量过多，算法会占用大量系统资源；如果初始种群数量过少，算法很可能忽略掉最优解。
- 对于每个解，一般根据实际情况进行编码，这样有利于编写变异函数和适应度函数(Fitness Function)。
- 在编码过的遗传算法中，每次变异的编码长度也影响到遗传算法的效率。如果变异代码长度过短，变异的多样性会受到限制；如果变异代码过长，变异的效率会非常低下，选择适当的变异长度是提高效率的关键。
- 变异率也是一个重要的参数。
- 对于动态数据，用遗传算法求最优解比较困难，因为染色体种群很可能过早地收敛，而对以后变化了的数据不再产生变化。对于这个问题，研究者提出了一些方法增加基因的多样性，从而防止过早的收敛。其中一种是所谓触发式超级变异，就是当染色体群体的质量下降（彼此的区别减少）时增加变异概率；另一种叫随机外来染色体，是偶尔加入一些全新的随机生成的染色体个体，从而增加染色体多样性。
- 选择过程很重要，但交叉和变异的重要性存在争议。一种观点认为交叉比变异更重要，因为变异仅仅是保证不丢失某些可能的解；而另一种观点则认为交叉过程的作用只不过是在种群中推广变异过程所造成的更新，对于初期的种群来说，交叉几乎等效于一个非常大的变异率，而这么大的变异很可能影响进化过程。
- 遗传算法很快就能找到良好的解，即使是在很复杂的解空间中。
- 遗传算法并不一定总是最好的优化策略，优化问题要具体情况具体分析。所以在使用遗传算法的同时，也可以尝试其他算法，互相补充，甚至根本不用遗传算法。
- 遗传算法不能解决那些“大海捞针”的问题，所谓“大海捞针”问题就是没有一个确切的适应度函数表征个体好坏的问题，使得算法的进化失去导向。
- 对于任何一个具体的优化问题，调节遗传算法的参数可能会有利于更好更快收敛，这些参数包括个体数目、交叉率和变异率。例如太大的变异率会导致丢失最优解，而过小的变异率会导致算法过早的收敛于局部最优点。对于这些参数的选择，现在还没有实用的上下限。

- 适应度函数对于算法的速度和效果也很重要。

变量

最简单的遗传算法将染色体表示为一个数位串，数值变量也可以表示成整数，或者实数（浮点数）。算法中的杂交和突变都是在字节串上进行的，所以所谓的整数或者实数表示也一定要转化为数位形式。例如一个变量的形式是实数，其范围是0～1，而要求的精度是0.001，那么可以用10个数位表示：0000000000表示0，1111111111表示1。那么0110001110就代表0.398。

在遗传算法里，精英选择是一种非常成功的产生新个体的策略，它是把最好的若干个个体作为精英直接带入下一代个体中，而不经过任何改变。

通过并行计算实现遗传算法一般有两种，一种是所谓粗糙并行遗传算法，即一个计算单元包含一个种群；而另一种是所谓精细并行遗传算法，每一个计算单元处理一个染色体个体。

遗传算法有时候还引入其他变量，例如在实时优化问题中，可以在适应度函数中引入时间相关性和干扰。

适用的问题

遗传算法擅长解决的问题是全局最优化问题，例如，解决时间表安排问题就是它的一个特长，很多安排时间表的软件都使用遗传算法，遗传算法还经常被用于解决实际工程问题。

跟传统的爬山算法相比，遗传算法能够跳出局部最优而找到全局最优点。而且遗传算法允许使用非常复杂的适应度函数（或者叫做目标函数），并对变量的变化范围可以加以限制。而如果是传统的爬山算法，对变量范围进行限制意味着复杂的多的解决过程，这方面的介绍可以参看受限优化问题和非受限优化问题。

发展历史

遗传算法由密歇根大学的约翰·霍兰德和他的同事于二十世纪六十年代在对细胞自动机（英文：cellular automata）进行研究时率先提出。在二十世纪八十年代中期之前，对于遗传算法的研究还仅仅限于理论方面，直到在匹兹堡召开了第一届世界遗传算法大会。随着计算机计算能力的发展和实际应用需求的增多，遗传算法逐渐进入实际应用阶段。1989年，纽约时报作者约翰·马科夫写了一篇文章描述第一个商业用途的遗传算法--进化者（英文：Evolver）。之后，越来越多种类的遗传算法出现并被用于许多领域中，财富杂志500强企业中大多数都用它进行时间表安排、数据分析、未来趋势预测、预算、以及解决很多其他组合优化问题。

应用领域

日本新干线N700系列车“气动双翼”的独特空气动力造型车鼻；是遗传算法运算结果

- 计算机自动设计 （CAD, Computer-Automated Design）
- 工业工程与运作管理
- 物流系统设计
- 生产调度
- 制造系统控制
- 系统优化设计
- 汽车设计，包括材料选择、多目标汽车组件设计、减轻重量等。
- 机电系统设计。
- 分布计算机网络的拓扑结构。
- 电路设计，此类用途的遗传算法叫做进化电路。
- 电子游戏设计，例如计算平衡解决方案。
- 机器智能设计和机器人学习。
- 模糊控制系统的训练。
- 移动通讯优化结构。
- 时间表安排，例如为一个大学安排不冲突的课程时间表。
- 旅行推销员问题。
- 神经网络的训练，也叫做神经进化。

相关技术

遗传程序是John Koza与遗传算法相关的一个技术，在遗传程序中，并不是参数优化，而是计算机程序优化。遗传程序一般采用树型结构表示计算机程序用于进化，而不是遗传算法中的列表或者数组。一般来说，遗传程序比遗传算法慢，但同时也可以解决一些遗传算法解决不了的问题。

交互式遗传算法是利用人工评价进行操作的遗传算法，一般用于适应度函数无法得到的情况，例如，对于图像、音乐、艺术的设计和“优化”，或者对运动员的训练等。

模拟退火是解决全局优化问题的另一个可能选择。它是通过一个解在搜索空间的随机变动寻找最优点的方法：如果某一阶段的随机变动增加适应度，则总是被接受，而降低适应度的随机变动根据一定的概率被有选择的接受。这个概率由当时的退火温度和适应度恶化的程度决定，而退火温度按一定速度降低。从模拟退火算法看，最优化问题的解是通过寻找最小能量点找到的，而不是寻找最佳适应点找到的。模拟退火也可以用于标准遗传算法里，只要把突变率随时间逐渐降低就可以了

四、 禁忌搜索

禁忌搜索是属于模拟人类智能的一种优化算法，它模仿了人类的记忆功能，在求解问题的过程中，采用了禁忌技术，对已经搜索过的局部最优解进行标记，并且在迭代中尽量避免重复相同的搜索（但不是完全隔绝），从而获得更广的搜索区间，有利于寻找到全局最优解。

模拟退火算法基本思想：在一定温度下，搜索从一个状态随机地变化到另一个状态；随着温度的不断下降直到最低温度，搜索过程以概率1停留在最优解。

五、 人工鱼群算法

人工鱼群算法是根据鱼类的活动特点提出的一种基于动物行为的自治寻优模式。
在一片水域中，鱼存在的数目最多的地方就是本水域中富含营养物质最多的地方，依据这一特点来模仿鱼群的觅食，聚群，追尾等行为，从而实现全局最优，这就是鱼群算法的基本思想。

六、 基本免疫算法

基本免疫算法基于生物免疫系统基本机制，模仿了人体的免疫系统。
基本免疫算法从体细胞理论和网络理论得到启发，实现了类似于生物免疫系统的抗原识别、细胞分化、记忆和自我调节的功能。
如果将免疫算法与求解优化问题的一般搜索方法相比较，那么抗原、抗体、抗原和抗体之间的亲和性分别对应于优化问题的目标函数、优化解、解与目标函数的匹配程度。

机器视觉

计算机视觉(Computer Vision)是采用图像处理、模式识别、人工智能技术相结合的手段，着重于一幅或多幅图像的计算机分析。具体来说，计算机视觉为机器视觉提供图像和景物分析的理论及算法基础，机器视觉为计算机视觉的实现提供传感器模型、系统构造和实现手段。

机器视觉(Machine Vision) 偏重于计算机视觉技术工程化，能够自动获取和分析特定的图像，以控制相应的行为。机器视觉系统就是一个能自动获取一幅或多幅目标物体图像，对所获取图像的各种特征量进行处理、分析和测量，并对测量做出定性分析和定量解释，从而得到有关目标物体的某种认识并作出相应决策的系统。

translation、rigid (Euclidean)、similarity、affine、projective

- 欧拉旋转定理:任何两个独立的正交坐标系都可以通过一系列(不超过两次)相对于坐标轴的旋转联系起来，但其中连续的两次旋转不能绕同一轴线。
- 最终得到的坐标系方向取决于旋转的顺序，旋转的叠加是不可交换的。
- 旋转顺序分为两种:欧拉式和卡尔丹式，分别以欧拉和卡尔丹(Cardano)的名字命名。
- 欧拉式是绕一个特定的轴重复旋转，但不是连续的:XYX、XZX、YXY、YZY、ZXZ或ZYZ。
- 卡尔丹式的特点 是绕3个不同轴旋转:XYZ、XZY、YZX、YXZ、ZXY或ZYX。
- 一般来说，所有这些序列均被统称为欧拉角，共有12种形式可供选择。
- 另一种广泛使用的旋转角顺序是横滚-俯仰-偏航角。
- 横滚、俯仰和偏航(也称为侧倾、姿态和航向)是指分别绕x、y、zPitch Axis轴的旋转。这个xyz角序列，即
  专业上的卡尔丹角，也被称为泰特-布莱恩角(Tait-Bryan)导航角。
- 横滚-俯仰-偏航序列允许每个角度值有任意正负号,不会产生多解的情况。但它也有一个奇异点，即当θp=±π/2时,不过这个点刚好在大多数车辆可能的姿态范围以外。

网络爬虫

1.列举您使用过的**python**网络爬虫所用到的网络数据包（最熟悉的在前）：

requests、urllib、urllib2、httplib2

2.列举您使用过的**python**网络爬虫所用到的解析数据包（最熟悉的在前）：

BeautifulSoup、pyquery、Xpath、lxml

3.列举您使用过的**python**中的编码方式（最熟悉的在前）：

UTF-8，ASCII，gbk

4.写出在网络爬虫爬取数据的过程中，遇到的反爬虫问题的解决方案

通过headers反爬虫：解决策略，伪造headers

基于用户行为反爬虫：动态变化去爬取数据，模拟普通用户的行为

基于动态页面的反爬虫：跟踪服务器发送的ajax请求，模拟ajax请求

尽量减少请求次数，能抓列表页就不抓详情页

可以考虑多线程，以及分布式

5.列出比较熟悉的爬虫框架？

常见爬虫框架列表

JAVA	PYTHON	PHP	C#	C/C++
Apache Nutch2	scrapy	phpspider	DotnetSpider	open-source-search-engine
webmagic	Crawley	Beanbun	NWebCrawler	Cobweb
Heritrix	Portia	PHPCrawl	SmartSpider	upton
WebCollector	PySpider	php selenium	Abot	wombat
crawler4j	grab		xNet	Spidr
Spiderman	cola		AngleSharp	Larbin
SeimiCrawler	python selenium		HtmlAgilityPack	
jsoup			CSQuery	
java selenium				
htmlunit				

6.scrapy由几个主要的部分组成**?**分别代表什么意思**?**

a.Engine：引擎负责控制系统所有组件之间的数据流，并在发生某些操作时触发事件。

b.Scheduler：调度程序接收来自引擎的请求，并将它们排入队列，并在之后，当Engine需要的时候，将requests发送给engine。

c.Downloader：下载器负责提取网页并将它们馈送到引擎，然后引擎将其发送给spider。

d.Spiders：蜘蛛是Scrapy用户编写的自定义类，用于解析响应并从中提取item项目（也称为抓取的项目）或追加的其他请求。

e.Item Pipeline：Item Pipeline负责处理被蜘蛛提取的item， 典型的任务包括清理，验证和持久性 （如将项目存储在数据库中）。

f.Downloader middlewares：下载器中间件是位于引擎和下载器之间的特定的钩子，当它们从引擎传递到下载器时处理请求，以及从下载器传递到引擎的响应。使用下载中间件可以达成如下的目的：

在将请求发送到下载器之前处理请求（即在Scrapy将请求发送到网站之前）。在传递给蜘蛛之前改变接收到的响应;
发送新的请求，而不是将接收到的响应传递给蜘蛛;向蜘蛛传递响应而不需要获取网页;

7.scrapy项目的搭建过程**?**

先安装scrapy

```
pip install scrapy
```

安装完成后就可以使用scrapy命令来创建项目了，如下：

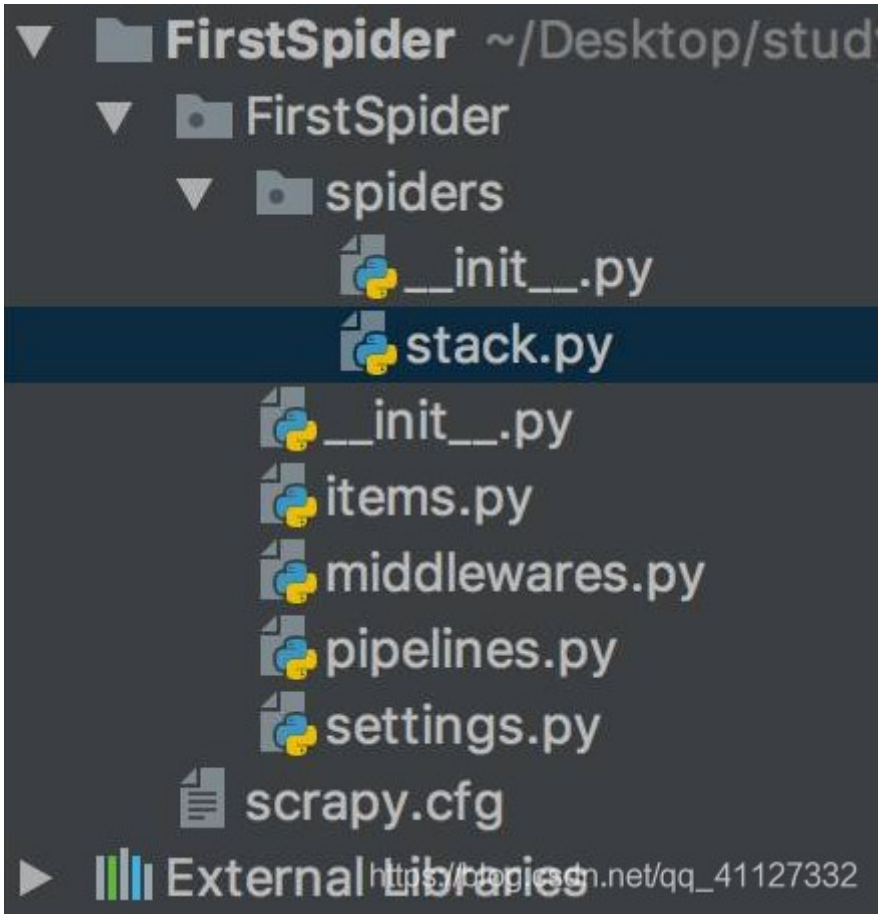
```
scrapy startproject FirstSpider
```

上面的命令只是生成了一个scrapy项目，之后还需要创建爬虫才能爬取，创建爬虫的命令如下：

```
scrapy genspider stack http://stackoverflow.com/
```

使用scrapy genspider来创建一个爬虫，并且指定名称为stack，起始爬取路径为http://stackoverflow.com/

创建完成后的项目目录结构如下：



可以看到在项目目录下会有一个与项目名同名的FirstSpider包，里面是我们Scrapy项目的各个模块。

8.xpath是什么**?**常用的路径及节点表示法**?**

XPath即为XML路径语言，它是一种用来确定XML（标准通用标记语言的子集）文档中某部分位置的语言。

XPath基于XML的树状结构，有不同类型的节点，包括元素节点，属性节点和文本节点，提供在数据结构树中找寻节点的能力。

路径表达式语法：

1.路径 = 相对路径 | 绝对路径

2.XPath路径表达式 = 步进表达式 | 相对路径 "/"步进表达式。

3.步进表达式=轴 节点测试 谓词

9. BS4是什么，有什么作用**?**常见的函数有哪些**?**

网络爬虫的最终目的就是过滤选取网络信息，最重要的部分可以说是解析器。解析器的优劣决定了爬虫的速度和效率。

解析器库bs4 库将复杂的html文档转化为一个复杂的树形结构，每个节点都是Python对象，所有对象可以分为以下四个类型：Tag , NavigableString , BeautifulSoup , Comment

Tag： 和html中的Tag基本没有区别，可以简单上手使用
NavigableString： 被包裹在tag内的字符串
BeautifulSoup： 表示一个文档的全部内容，大部分的时候可以吧他看做一个tag对象，支持遍历文档树和搜索文档树方法。
Comment：这是一个特殊的NavigableSting对象，在出现在html文档中时，会以特殊的格式输出，比如注释类型。

10.urllib的常见子模块是哪些?

urllib.request模块：帮助在复杂的世界中打开URL（主要是HTTP）的函数和类——基本和摘要身份验证、重定向、cookie等等。

urllib.error子模块相对来说比较简单，官方定义为由urllib.request导致的异常，其中URLError是最基本的异常。

urllib.parse可以拆分URL，也可以拼接URL。

11. 如何给 **request**加入一个代理，如何修改 **user-agent**的值**?**

代理（Streaming Requests）
如果你需要使用一个代理，你可以在任何请求方法中使用proxies参数配置单个请求：

```
import requests









proxies = {


  "http": "10.10.1.10:3128",


  "https": "10.10.1.10:1080",
```

```
}


requests.get("http://example.org", proxies=proxies)
```

你也可以通过环境变量HTTP_PROXY和HTTPS_PROXY来配置代理:

```
$ export HTTP_PROXY="10.10.1.10:3128"


$ export HTTPS_PROXY="10.10.1.10:1080"


$ python


>>> import requests


>>> requests.get("http://example.org")
```

如何修改网页的user-agent值？

12. selenium是一个什么样的爬虫框架**?**如何使用**?**

自动化测试工具。它支持各种浏览器，包括 Chrome，Safari，Firefox 等主流界面式浏览器，如果你在这些浏览器里面安装一个 Selenium 的插件，那么便可以方便地实现Web界面的测试。换句话说叫 Selenium 支持这些浏览器驱动。话说回来，PhantomJS不也是一个浏览器吗，那么 Selenium 支持不？答案是肯定的，这样二者便可以实现无缝对接了。
模拟提交 测试用例 页面交互 填充表单 元素拖拽 页面切换 弹窗处理 Cookies处理 元素选取 页面等待
Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。
隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行

https://www.cnblogs.com/BigFishFly/p/6380024.html

13.如何给**selenium**爬虫添加**user-agent, proxy?**

默认情况下，是没有自动设置User-Agent的，默认的User-Agent显示为PhantomJS；
selenium可调用firefox chrome phantomjs等各种浏览器。设置PhantomJS的user-agent，是要设置“phantomjs.page.settings.userAgent”这个desired_capability。

https://blog.csdn.net/zaixiahanli/article/details/64479463

14.selenium爬虫如何跟踪**ajax**请求**?**

想办法分析出接收ajax返回数据的变量，然后用

```
data = driver.execute_script("return Json") //假设Json是那个变量。
```

驱动要用phantomjs，data就是你要的json数据。

15. 是否了解线程的同步和异步？

多线程并发时，多个线程同时请求同一个资源，必然导致此资源的数据不安全，A线程修改了B线
程的处理的数据，而B线程又修改了A线程处理的数理。显然这是由于全局资源造成的，有时为了解
决此问题，优先考虑使用局部变量，退而求其次使用同步代码块，出于这样的安全考虑就必须牺牲
系统处理性能，加在多线程并发时资源挣夺最激烈的地方，这就实现了线程的同步机制。
同步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为同步机制存在，A线程请求
不到，怎么办，A线程只能等待下去
异步：A线程要请求某个资源，但是此资源正在被B线程使用中，因为没有同步机制存在，A线程
仍然请求的到，A线程无需等待
显然，同步最最安全，最保险的。而异步不安全，容易导致死锁，这样一个线程死掉就会导致整个
进程崩溃，但没有同步机制的存在，性能会有所提升。

16.你是否了解谷歌的无头浏览器？

无头浏览器即headless browser，是一种没有界面的浏览器。既然是浏览器那么浏览器该有的东西它都应该有，只是看不到界面而已。

Python中selenium模块中的PhantomJS即为无界面浏览器（无头浏览器）:是基于QtWebkit的无头浏览器。

17.分布式爬虫的设计原理是什么，解决了什么问题**?**

设计原理：引入了master-slaver机制，用master来控制，监控slaver爬取状态，进度引入心跳机制。

解决问题：(1)ip (2)带宽 (3）cpu (4）io

18.gzip压缩是什么**?python**爬取网页时，如何处理**?**

在我们用python3爬取一些网站时，获取网页url后进行解析，在采用decode('utf-8')解码时有时候会出现utf-8无法解码的问题，比如结果会提示：

```
Unicode Decode Error: 'utf8' codec can't decode byte 0xb2 in
position 0: invalid start byte
```

是因为有些网站进行了gzip压缩，最典型的就是sina。

HTTP协议上的GZIP编码是一种用来改进WEB应用程序性能的技术。大流量的WEB站点常常使用GZIP压缩技术来让用户感受更快的速度。这一般是指WWW服务器中安装的一个功能，当有人来访问这个服务器中的网站时，服务器中的这个功能就将网页内容压缩后传输到来访的电脑浏览器中显示出来.一般对纯文本内容可压缩到原大小的40%.这样传输就快了，效果就是你点击网址后会很快的显示出来.当然这也会增加服务器的负载*. *一般服务器中都安装有这个功能模块的。

19.scrapy的去重原理

1.Scrapy本身自带有一个中间件;

2.scrapy源码中可以找到一个dupefilters.py去重器;

3.需要将dont_filter设置为False开启去重，默认是True，没有开启去重;

4 .对于每一个url的请求，调度器都会根据请求得相关信息加密得到一个指纹信息，并且将指纹信息和set()集合中的指纹信息进 行 比对，如果set()集合中已经存在这个数据，就不在将这个Request放入队列中;

5.如果set()集合中没有存在这个加密后的数据，就将这个Request对象放入队列中，等待被调度。

20.代理问题**:** 为什么会用到代理**?**代理怎么使用**?**代理失效了怎么处理**?**

为什么会用到代理**?**

1.安全避免同一个代理IP访问同一个网页，对于长时间访问同一个网页的IP，极大可能性IP会被封掉。

2.方便解决IP代理问题技术含量高，找代理处理方便省事。

3.成本低自己去维护服务器成本过高，不低于长久持续发展。

代理怎么使用**?**代理失效了怎么处理**?**

https://baijiahao.baidu.com/s?id=1618299340903906830&wfr=spider&for=pc

事先用检测代码检测可用的代理，每隔一段时间更换一次代理，如果出现302等状态码，则立即更换下一个可用的IP。

21. http协议**: http**协议请求头，请求体，响应头，响应体语法**?**

HTTP请求：Hyper Text Transfer Protocol的缩写，即超文本传输协议。

HTTP请求报文由3部分组成（请求行+请求头+请求体）。HTTP的响应报文也由3部分组成（响应行+响应头+响应体）

网络爬虫（英语：web crawler），也叫网络蜘蛛（spider），是一种用来自动浏览万维网的网络机器人。其目的一般为编纂网络索引。

网络搜索引擎等站点通过爬虫软件更新自身的网站内容或其对其他网站的索引。网络爬虫可以将自己所访问的页面保存下来，以便搜索引擎事后生成索引供用户搜索。

爬虫访问网站的过程会消耗目标系统资源。不少网络系统并不默许爬虫工作。因此在访问大量页面时，爬虫需要考虑到规划、负载，还需要讲“礼貌”。 不愿意被爬虫访问、被爬虫主人知晓的公开站点可以使用robots.txt文件之类的方法避免访问。这个文件可以要求机器人只对网站的一部分进行索引，或完全不作处理。

互联网上的页面极多，即使是最大的爬虫系统也无法做出完整的索引。因此在公元2000年之前的万维网出现初期，搜索引擎经常找不到多少相关结果。现在的搜索引擎在这方面已经进步很多，能够即刻给出高质量结果。

爬虫还可以验证超链接和HTML代码，用于网络抓取（参见数据驱动编程）。

命名[编辑]

网络爬虫也可称作网络蜘蛛[1]、蚂蚁、自动索引程序（automatic indexer）[2]，或（在FOAF软件中）称为网络疾走（web scutter）。[3]

概述[编辑]

网络爬虫始于一张被称作种子的统一资源地址（URL）列表。当网络爬虫访问这些统一资源定位器时，它们会甄别出页面上所有的超链接，并将它们写入一张“待访列表”，即所谓爬行疆域。此疆域上的URL将会被按照一套策略循环来访问。如果爬虫在执行的过程中复制归档和保存网站上的信息，这些文件通常储存，使他们可以较容易的被查看。阅读和浏览他们存储的网站上并即时更新的信息，这些被存储的网页又被称为“快照”。越大容量的网页意味着网络爬虫只能在给予的时间内下载越少部分的网页，所以要优先考虑其下载。高变化率意味着网页可能已经被更新或者被取代。一些服务器端软件生成的URL（统一资源定位符）也使得网络爬虫很难避免检索到重复内容。

但是互联网的资源卷帙浩繁，这也意味着网络爬虫只能在一定时间内下载有限数量的网页，因此它需要衡量优先级的下载方式。有时候网页出现、更新和消失的速度很快，也就是说网络爬虫下载的网页在几秒后就已经被修改或甚至删除了。这些都是网络爬虫设计师们所面临的两个问题。

再者，服务器端软件所生成的统一资源地址数量庞大，以致网络爬虫难免也会采集到重复的内容。根据超文本传输协议，无尽组合的参数所返回的页面中，只有很少一部分确实传回正确的内容。例如：数张快照陈列室的网站，可能通过几个参数，让用户选择相关快照：其一是通过四种方法对快照排序，其二是关于快照分辨率的的三种选择，其三是两种文件格式，另加一个用户可否提供内容的选择，这样对于同样的结果会有48种（432）不同的统一资源地址与其关联。这种数学组合替网络爬虫造成了麻烦，因为它们必须越过这些无关脚本变化的组合，寻找不重复的内容。

爬虫策略[编辑]

爬虫的实现由以下策略组成：[4]

- 指定页面下载的选择策略
- 检测页面是否改变的重新访问策略
- 定义如何避免网站过度访问的约定性策略
- 如何部署分布式网络爬虫的并行策略

选择策略[编辑]

链接跟随限制[编辑]

爬虫可能只想搜索HTML页面而避免其他MIME 类型。为了只请求HTML资源，爬虫在抓取整个以GET方式请求的资源之前，通过创建HTTP的HEAD请求来决定网络资源的MIME类型。为了避免发出过多的请求，爬虫会检查URL和只请求那些以某些字符（如.html, .htm, .asp, .aspx, .php, .jsp, .jspx 或 / ）作为后缀的URL。这个策略可能会跳过很多HTML网络资源。

为了避免掉入从网站下载无限量的URL的爬虫陷阱，有些爬虫还能避免请求一些带有“?”的资源（动态生成）。不过假若网站重写URL以简化URL的目的，这个策略就变得不可靠了。

URL规范化[编辑]

主条目：URL规范化

爬虫通常使用某些URL规范化的方式以避免资源的重复爬取。URL规范化，指的是以某种一致的方式修改和标准化URL的过程。这个过程有各种各样的处理规则，包括统一转换为小写、移除“.”和“..”片段，以及在非空路径里插入斜杆。

路径上移爬取[编辑]

有些爬虫希望从指定的网站中尽可能地爬取资源。而路径上移爬虫就是为了能爬取每个URL里提示出的每个路径。[5] 例如，给定一个Http的种子URL: http://llama.org/hamster/monkey/page.html ，要爬取 /hamster/monkey/ ， /hamster/ 和 / 。Cothey发现路径能非常有效地爬取独立的资源，或以某种规律无法在站内链接爬取到的资源。

主题爬取[编辑]

主条目：主题爬虫

对于爬虫来说，一个页面的重要性也可以说是，给定查询条件一个页面相似性能起到的作用。网络爬虫要下载相似的网页被称为主题爬虫或局部爬虫。这个主题爬虫或局部爬虫的概念第一次被Filippo Menczer[6][7] 和 Soumen Chakrabarti 等人提出的。[8]

重新访问策略[编辑]

网站的属性之一就是经常动态变化，而爬取网站的一小部分往往需要花费几个星期或者几个月。等到网站爬虫完成它的爬取，很多事件也已经发生了，包括增加、更新和删除。 在搜索引擎的角度，因为没有检测这些变化，会导致存储了过期资源的代价。最常用的估价函数是新鲜度和过时性。 新鲜度：这是一个衡量抓取内容是不是准确的二元值。在时间t内，仓库中页面p的新鲜度是这样定义的：

过时性:这是一个衡量本地已抓取的内容过时程度的指标。在时间t时，仓库中页面p的时效性的定义如下：

平衡礼貌策略[编辑]

爬虫相比于人，可以有更快的检索速度和更深的层次，所以，他们可能使一个站点瘫痪。不需要说一个单独的爬虫一秒钟要执行多条请求，下载大的文件。一个服务器也会很难响应多线程爬虫的请求。 就像Koster所注意的那样，爬虫的使用对很多任务作都是很有用的，但是对一般的社区，也需要付出代价。使用爬虫的代价包括：[9]

- 网络资源：在很长一段时间，爬虫使用相当的带宽高度并行地工作。
- 服务器超载：尤其是对给定服务器的访问过高时。
- 质量糟糕的爬虫，可能导致服务器或者路由器瘫痪，或者会尝试下载自己无法处理的页面。
- 个人爬虫，如果过多的人使用，可能导致网络或者服务器阻塞。

对这些问题的局部解决方法是漫游器排除协议（Robots exclusion protocol），也被称为robots.txt议定书[10]，这份协议是让管理员指明网络服务器的不应该爬取的约定。这个标准没有包括重新访问一台服务器的间隔的建议，虽然设置访问间隔是避免服务器超载的最有效办法。最近的商业搜索引擎，如Google，Ask Jeeves，MSN和Yahoo可以在robots.txt中使用一个额外的 “Crawl-delay”参数来指明请求之间的延迟。

并行策略[编辑]

主条目：Distributed web crawling

一个并行爬虫是并行运行多个进程的爬虫。它的目标是最大化下载的速度，同时尽量减少并行的开销和下载重复的页面。为了避免下载一个页面两次，爬虫系统需要策略来处理爬虫运行时新发现的URL，因为同一个URL地址，可能被不同的爬虫进程抓到。