Introduction to Artificial Intelligence
Class Notes

Chenghua Liu
liuch18@mails.tsinghua.edu.cn
Department of Computer Science
Tsinghua University

目录

1 Heuristic Search

启发式搜索是利用问题拥有的启发信息来引导搜索，从而减少搜索的范围，降低问题的复杂度。启发信息较弱时，搜索算法在找到一条路径之前将扩展过多的节点，求得解路径的耗散值较大；启发信息强的时候，会大大降低搜素的工作量，但不能保证找到最小耗散值的解路径。因此实际应用中我们希望在找到最优路径的同时尽可能降低搜索的工作量。

1.1 A search algorithm

A 算法实际中典型的启发式搜索算法。其基本思想是：定义一个评价函数 $f(n)$ ，对当前的搜索状态进行评估，找到一个最好的节点来拓展。我们定义评价函数：

$$f(n) = g(n) + h(n) \tag{1}$$

其中 n 是被评估的节点，定义：

g*(n): 表示从初始节点 s 到节点 n 的最短路径的耗散值

h*(n): 表示从节点 n 到目标节点 g 的最短路径的耗散值，也称为启发函数

f*(n)=g*(n)+h*(n): 表示从初始节点 s 经过节点 n 到目标节点 g 的最短路径的耗散值

$f(n)$、$g(n)$、$h(n)$ 分别表示对上面三个函数的估计值。A 算法每次按照 $f(n)$ 值的大小对 OPEN 表中元素进行排序，大的放在前面，小的放在后面，每次拓展节点时总选择当前 f 值最小的点优先拓展。

当 h(n) ≡ 0 时，则 A 算法演变为动态规划算法。而当在 A 算法的评价函数中，使用的启发函数 $h(n)$ 是处在 $h^*(n)$ 的下界范围，即满足 $h(n) \leq h^*(n)$ 时，我们把这个算法称为 A* 算法。

1.2 A* search algorithm

A* 算法实际上是分支界限和动态规划原理及使用下界范围的 h 相结合的算法。当问题有解时，A* 一定能找到一条到达目标节点的最佳路径。A* 算法有一些好的性质，我们不加证明的在下面给出。

Theorem 1.2.1. *对有限图，如果从初始节点 s 到目标节点 t 有路径存在，则算法 A 一定成功结束。*

Lemma 1.2.1. *对无限图，若有从初始节点 s 到目标点 t 的一条路径，则 A^* 不结束时，在 OPEN 中即使最小的一个 f 值也将增到任意大，或有 $f(n)>f^*(s)$。*

Lemma 1.2.2. *A^* 结束前，OPEN 表中必存在 $f(n) \leq f^*(s)$ 的节点 (n 是在最佳路径上的节点)。*

Theorem 1.2.2. *对无限图，若从初始节点 s 到目标节点 t 有路径存在，A^* 也一定成功结束。*

证明. 假定 A* 不结束，由引理 1.2.1 有 f(n)>f*(s)，或 OPEN 表中最小的一个 f 值也变成无界，这与引理 1.2.2 的结论矛盾，所以 A* 只能成功结束。推论:OPEN 表上任一具有 f(n)<f*(s) 的节点 n，最终都将被 A* 选作为扩展的节点。 □

Theorem 1.2.3. *若存在初始节点 s 到目标节点 t 的路径，则 A^* 必能找到最佳解结束。*

证明. (1) 由定理 (1.2.1)(1.2.2) 知 A* 一定会找到一个目标节点结束。(2) 设找到一个目标节点 t 结束，而 $s\psi t$ 不是一条最佳路径，即 $f(t) = g(t) > f^*(s)$ 而根据引理 1.2.2 知结束前 OPEN 表上有节点 n ，且处在最佳路径上，并有 f(n) ⩽ f*(s)，所以

$$f(n) \leqslant f^*(s) < f(t)$$

这时算法 A^* 应选 n 作为当前节点扩展，不可能选 t, 从而也不会去测试目标节点 t, 即这与假定 A^* 选 t 结束矛盾，所以 A^* 只能结束在最佳路径上。

推论: A^* 选作扩展的任一节点 n, 有 $f(n) \leqslant f^*(s)$。证明：令 n 是由 A^* 选作扩展的任一节点，因此 n 不会是目标节点，且搜索没有结束, 由引理 1.2.2 而知在 $OPEN$ 中有满足 $f(n') \leq f^*(s)$ 的节点 n'。若 $n = n'$, 则 $f(n) \leqslant f^*(s)$, 否则选 n 扩展，必有 $f(n) \leq f(n')$, 所以 $f(n) \leqslant f^*(s)$ 成立。 □

对于 A* 算法，如果选作扩展的节点 n，其评价函数值 $f(n) = f^*(n)$，则不去扩展多余的节点就可找到解。$f(n)$ 越接近于 $f^*(n)$，扩展的节点数就会越少，即启发函数中应用的启发信息 (问题知识) 愈多，扩展的节点数就越少。使用数学归纳法对节点的深度进行归纳，我们有以下定理。

Theorem 1.2.4. *有两个 A^* 算法 $A1$ 和 $A2$，若 $A2$ 比 $A1$ 有更多的启发信息，即对所有非目标节点均有 $h2(n)>h1(n)$，则在具有一条从 s 到 t 的隐含图上，搜索结束时，由 $A2$ 所扩展的每一个节点，也必定由 $A1$ 所扩展，即 $A1$ 扩展的节点至少和 $A2$ 一样多。*

因 A 算法对 m_1 类节点可能要重新放回到 OPEN 表中，因此可能会导致多次重复扩展同一个节点，导致搜索效率下降。我们可以通过对 h 加以限制，使得第一次扩展一个节点时，就找到了从 s 到该节点的最短路径。一个启发函数 h，如果对所有节点 n_i 和 n_j，其中 n_j 是 n_i 的子节点，满足

$$\begin{cases} h(n_i) - h(n_j) \leq c(n_i, n_j) \\ h(t) = 0 \end{cases} \tag{2}$$

则称 h 是单调的。若 $h(n)$ 是单调的，则 A* 扩展了节点 n 之后，就已经找到了到达节点 n 的最佳路径。即: 当 A* 选 n 扩展时，有 $g(n) = g^*(n)$。利用单调条件，从目标及目标的父亲节点向上归纳我们可以得知满足单调条件的 h 一定满足 A* 条件。在单调的情形下，我们引入变量 f_m：到目前为止已扩展节点的最大 f 值，用 f_m 代替 $f^*(s)$ 对算法进行改进。

A* 算法是 N.J.Nilsson70 年代初的研究成果，是从函数的观点来讨论搜索问题，并在理论上取得若干结果。但 A* 算法不能完全克服“指数爆炸”的困难。70 年代末 J.Pear1 从概率观点

研究了启发式估计的精度同 A* 算法平均复杂性的关系。Pear1 假设如下的概率搜索空间: 一个一致的 m-枝树 G，在深度 d 处有一个唯一的目标节点 G_d，其位置事先并不知道。设估计量 $h(n)$ 是在 $[0, h^*(n)]$ 区间中的随机应量，由分布函数 $F_{h(n)}(x) = P[h(n) \leq x]$ 来描述；$E(Z)$ 表示用 A* 算法求到目标 G_d 时所展开的节点平均个数，并称之为 A* 的平均复杂性。若 $h(n)$ 满足

$$P\left[\frac{h^*(n) - h(n)}{h^*(n)} > \varepsilon\right] > \frac{1}{m}, \varepsilon > 0 \tag{3}$$

则有 $E(Z) \sim O\left(e^{cd}\right), \quad c > 0$。

80 年代初，张拔、张钠提出把启发式搜索看成某种随机取样的过程，从而将统计推断引入启发式搜索。把各种统计推断方法，如序贯概率比检验法（SPRT），均值固定宽度信度区间渐近序贯法（ASM）等，同启发式搜索算法相结合，得到一种称为 SA (统计启发式）的搜索算法。该算法在一定假设条件下，能以概率为一找到目标，且其平均复杂性为 $O(d \ln d)$ 或 $O\left(d \ln^2 d\right)$。

2 Adversarial Search

在多 Agent 环境中（竞争环境），每个 Agent 的目标之间是有冲突的，所以就引出了对抗搜索（通常称为博弈）。人工智能中的博弈通常指博弈论专家们称为有完整信息的，确定性的，轮流行动的，两个游戏者的零和游戏（如象棋）。注意到在博弈问题中，搜索树实在太大，A* 搜索效率很低。所以在博弈搜索的算法中，可以通过剪枝在搜索树中忽略那些不影响最后决定的部分，通过启发式的评估函数不进行完全搜索的情况下估计某状态的真实效用值，从而大大提高算法的速度。

2.1 Minimax algorithm

极小极大搜索方法是博弈树搜索的基本方法，现在博弈树搜索中最常用的 - 剪枝搜索方法，就是从这一方法发展而来的。

首先假定，有一个评价函数可以对所有的棋局进行评估。当评价函数值大于 0 时，表示棋局对我方有利，对对方不利。当评价函数小于 0 时，表示棋局对我方不利，对对方有利。而评价函数值越大，表示对我方越有利。当评价函数值等于正无穷大时，表示我方必胜。评价函数值越小，表示对我方越不利。当评价函数值等于负无穷大时，表示对方必胜。假设双方都是对弈高手，在只看一步棋的情况下，我方一定走评价函数值最大的一步棋，而对方一定走评价函数值最小的一步棋。会下棋的读者都知道，在只看一步的情况下最好的棋，从全局来说不一定就好，还可能很不好。因此为了走出好棋，必须多看几步，从多种可能状态中选择一步好棋。

极小极大搜索策略是考虑双方对弈若干步之后，从可能的走步中选一步相对好棋的着法来走，即在有限的搜索深度范围内进行求解。为此要定义一个静态估计函数 f，以便对棋局的势态 (节点) 作出优劣估值，这个函数可根据势态优劣特征来定义 (主要用于对端节点的“价值”进行度量)。一般规定有利于 MAX 的势态，$f(p)$ 取正值，有利于 MIN 的势态，$f(p)$ 取负值，势

均力敌的势态，$f(p)$ 取 0 值。若 $f(p)=+\infty$，则表示 MAX 赢，若 $f(p)=-\infty$，则表示 MIN 赢。下面的讨论规定: 顶节点深度 $d=0$，MAX 代表程序方，MIN 代表对手方，MAX 先走。

2.2 Alpha-beta pruning

MINIMAX 过程是把搜索树的生成和格局估值这两个过程分开来进行，即先生成全部搜索树，然后再进行端节点静态估值和倒推值计算，这显然会导致低效率。把生成和倒推估值结合起来进行，再根据一定的条件判定，有可能尽早修剪掉一些无用的分枝，同样可获得类似的效果，这就是 $\alpha-\beta$ 过程的基本思想。

为了使生成和估值过程紧密结合，采用有界深度优先策略进行搜索，这样当生成达到规定深度的节点时，就立即计算其静态估值函数，而一旦某个非端节点有条件确定其倒推值时就立即计算赋值。我们称极大值层的这个下界值为 α，称极小值层的这个上界值为 β。

(1) α 剪枝: 若任一**极小值层节点**的 β 值小于或等于它任一先辈极大值居节点的 α 值，即 α (先辈层)$\geq\beta$ (后继层)，则可中止该极小值层中这个 MIN 节点以下的搜索过程。这个 MIN 节点最终的倒推值就确定为这个 β 值。

(2) β 剪枝: 若任一**极大值层节点**的 α 值大于或等于它任一先辈极小值层节点的 β 值，即 α (后继层) $\geq\beta$ (先辈层)，则可以中止该极大值层中这个 MAX 节点以下的搜索过程。这个 MAX 节点的最终倒推值就确定为这个 α 值。

应该注意的是，博弈树搜索的目标就是找到当前棋局的一步走法，所以 $\alpha-\beta$ 剪枝搜索的结果是得到了一步最佳走步，而不是象一般的图搜索或者与或图搜索那样，得到的是从初始节点到目标节点 (集) 的一条路径或者解图。

下面分析一下剪枝的效率问题。若以最理想的情况进行搜索，即对 MIN 节点先扩展最低估值的节点 (若从左向右顺序进行，则设节点估计值从左向右递增排序)，MAX 先扩展最高估值的节点 (设估计值从左向右递减排序)，则当搜索树深度为 D，分枝因数为 B 时，若不使用 $\alpha-\beta$ 剪枝技术，搜索树的端节点数 $N_D=B^D$; 若使用 $\alpha-\beta$ 剪枝技术. 可以证明理想条件下生成的端节点数最少，有

$$\mathrm{N_D}=2\ \mathrm{B^{D/2}}-1(\text{D为偶数}),\quad \mathrm{N_D}=\mathrm{B^{(D+1)}}/2+\mathrm{B^{(D-1)/2}}-1(\text{D为奇数}) \tag{4}$$

比较后得出最佳 $\alpha-\beta$ 搜索技术所生成深度为 D 处的端节点数约等于不用 $\alpha-\beta$ 搜索技术所生成深度为 D/2 处的端节点数。这就是说，在一般条件下使用 $\alpha-\beta$ 搜索技术，在同样的资源限制下，可以向前考虑更多的走步数，这样选取当前的最好优先走步，将带来更大的取胜优势。

最后要说的是，$\alpha-\beta$ 搜索严重依赖于局面评估的准确性，而局面评估需要大量的专家知识和人工整理，这也是该方法在象棋非常有效但围棋上行不通的原因。

2.3 Monte Carlo tree search

蒙特卡罗树搜索是解决马尔科夫决策问题的有效方法之一，它是以蒙特卡罗方法的思想为基础的一种搜索方法。在蒙特卡罗树搜索中，可以将马尔科夫决策过程中可能出现的状态转移过程用状态树建立并表示出来。不同于普通搜索树的建立，蒙特卡罗树搜索通过从初始状态开始重复给出随机模拟事件，并逐步扩展搜索树中的每个节点，而每一次模拟事件都是“状态-行为-回报”的三元序列。采用蒙特卡洛树搜索的一个重要原因是，它可以保证我们在从始至终的每一次随机模拟中随时得到行为的评价。因此，如果在模拟过程中某个状态被再次访问到，我们便可以利用之前的“行为-回报”信息来为接下来的选择做参考，借此便可以加速评价的收敛速度。

更具体来讲，蒙特卡罗树搜索共包含四个基本步骤，分别为选择 (selection)、扩展 (Expansion)、模拟 (Simulation) 和回溯 (Back propagation)，见图 1。

1. 选择: 从根节点出发，在搜索树上自上而下迭代式执行一个子节点选择策略，直至找到当前最为紧迫的可扩展节点为止，一个节点是可扩展的，当且仅当其所对应的状态是非停止状态，且拥有未被访问过的子状态;
2. 扩展: 根据当前可执行的行动，向选定的节点上添加一个 (或多个) 子节点以扩展搜索树;
3. 模拟: 根据默认策略在扩展出来的一个 (或多个) 子节点上执行蒙特卡罗棋局模拟，并确定节点的估计值;
4. 回溯: 根据模拟结果向上依次更新祖先节点的估计值，并更新其状态。

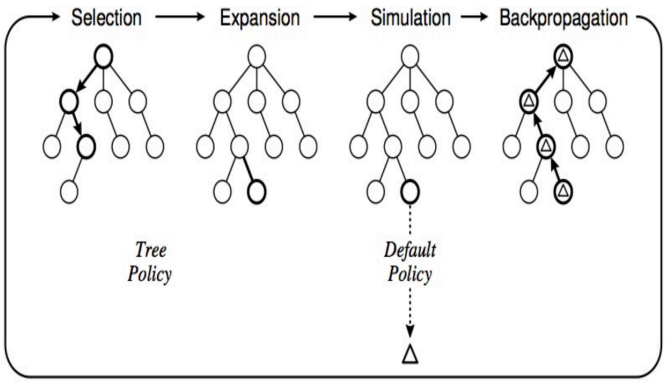

图 1: Monte Carlo tree search

2.3.1 Upper Confidence Bound algorithm

多臂老虎机拥有 k 个手臂，拉动每个手臂所能产生的回报互不相关，而赌博者的目的就是为了获得尽可能多的收益，在多臂老虎机模型中也是如此，我们希望找到一个合理的策略，可以使得拉动手臂的人获得最大的收益。

多臂老虎机问题可以抽象为如下的数学模型：一个有 k 个臂的老虎机被一系列的随机收益值所表示 $\{X_{1t}, X_{2t}, \ldots, X_{it}, \ldots, X_{kt}\}$，其中 $1 \leq i \leq k, t \geq 1$ 。这里的 i 表示第 i 个臂的编号，拉动第 i 个臂，所得到的回报序列为 $\{X_{i1}, X_{i2}, X_{i3}, \ldots\}$，且不同的手臂的收益分布之间没有相互关系。解决多臂老虎机问题实际上就是选择一个策略 A 去决定下一次将被拉动的手臂的编号，这个决策既取决于过去每个臂已经被拉动的次数，也受限于这些被拉动的次数中每个手臂上取得的收益值。假设 $T_j(n)$ 表示当总的拉动次数为 n 时，第 j 号手臂被拉动的次数，那么当进行 n 次拉动后根据当前策略 A 所产生的损失可以表示为:

$$\rho = n\mu^* - \sum_{j=1}^{k} E\left[T_j(n)\right] \mu_j \tag{5}$$

其中 μ^* 表示收益最好的手臂的收益均值，μ_j 为第 j 号手臂被拉动后所产生的收益均值。在 Lai 和 Robbins 的 1985 年的关于该问题的经典文章中讲到，对于某一特定的回报分布，拉动策略满足:

$$E\left[T_j(n)\right] \leq \frac{\ln(n)}{D\left(P_j \| P^*\right) + o(1)} \tag{6}$$

其中，当 $n \to \infty$ 时，有 o(1) $\to 0$, 并且:

$$D\left(P_j \| P^*\right) = \int P_j \ln \frac{P_j}{P^*} \tag{7}$$

为在任一次尝试中，手臂 j 的回报分布概率密度 P_j 和当前回报最高的手臂的概率密度 P^* 之间的 KL 测度。因而在尝试的次数（亦即随机模拟的次数）足够多时，最优的手臂被拉动的次数和其它手臂被拉动的次数之间有较大的差距，具体而言，次优的手臂 j 被拉动的次数满足:

$$E\left[T_j(n)\right] \leq \frac{\ln(n)}{D\left(P_j \| P^*\right)} \tag{8}$$

我们还可以把损失表示为:

$$\rho = \max_i E\left[\sum_{t=1}^{n} X_{it}\right] - E\left[\sum_{i=1}^{k} \sum_{t=1}^{T_i(n)} X_{it}\right] \tag{9}$$

从这个公式我们可以看出，造成损失的原因是我们选取的策略不一定会每次都选择收益最佳的臂。对于这样的一大类收益分布问题，已经证明没有任何策略能让损失的增长速度低于 O $O(\ln(n))$ 。对于这种收益分布，如果一个策略的损失增长速度不超过 $C\ln(n)$ ，其中 C 为一个常数，我们就认为它在探索和利用之间找到了一个平衡方法。

UCB 算法即信心上界算法，是一类解决多臂老虎机问题的算法的总称，其中，我们将在本节介绍是 UCB1 算法是该类算法的代表，可以用如下伪码表示:

```
function UCB1
        for each 手臂 :
                访问该手臂并记录收益
        end for
        while 尚未达到访问次数限制 do:
```

```
6                 计算每个手臂的 UCB1 信心上界I（如下所述）
7                 访问信心上界最大的手臂
8           end while
```

UCB1 算法的核心在于解决继续探索和利用当前状态之间的平衡问题。它为所有臂记录了平均收益 $\bar{X}_{i,T_i(t-1)}$ ，并选择具有最大置信上界的臂:

$$I_t = \text{argmax}_{1\leq i\leq k}\left\{\bar{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)}\right\} \tag{10}$$

这里的偏移序列 $c_{t,s}$ 定义为:

$$c_{t,s} = \sqrt{\frac{2\ln(t)}{s}} \tag{11}$$

在 $X_{i,t}$ 独立同分布的条件下，偏移序列 $c_{t,s}$ 满足如下的概率收敛不等式:

$$\begin{aligned} P\left(\bar{X}_{is} \geq \mu_i + c_{t,s}\right) &\leq t^{-4} \\ P\left(\bar{X}_{is} \leq \mu_i - c_{t,s}\right) &\leq t^{-4} \end{aligned} \tag{12}$$

在搜索的过程中，UCB1 被用来确定内部节点将要选择的下一个尝试的行为，我们可以用更简单具体的形式表示 UCB1 的信心上界索引:

$$I_j = \bar{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}} \tag{13}$$

其中，$\bar{X}_j$ 是手臂 j 所获得的回报的均值，n 是到当前这一时刻为止所访问的总次数，$T_j(n)$ 是手臂 j 到目前为止总共所访问的次数。在 UCB1 算法中，当手臂数目 $k > 1$ 时，如果 k 个手臂的回报在 [0,1] 区间上分别服从分布 $P_1, P_2, \ldots, P_k$, 那么当总共拉动 n 次手臂之后，该算法的损失不超过以下上限:

$$\left[8 \cdot \sum_{i:\mu_i<\mu^*} \frac{\ln(n)}{\Delta_i}\right] + \left(1 + \frac{\pi^2}{3}\right) \cdot \left(\sum_{j=1}^{k} \Delta_j\right) \tag{14}$$

其中 $\Delta_j = \mu^* - \mu_j, \mu_i (1 \leq i \leq k)$ 分别为分布 P_i 的数学期望，μ^* 为 $\mu_i (1 \leq i \leq k)$ 中的最大值。此外，次优手臂 j 的访问次数的数学期望满足以下上限:

$$E\left[T_j(n)\right] \leq \frac{8}{\Delta_j^2 \ln(n)} \tag{15}$$

观察 UCB1 算法中上限信心索引的计算公式 $I_j = \bar{X}_j + \sqrt{\frac{2\ln(n)}{T_j(n)}}$, 我们可以更加直观的理解其两部分的含义，其中前一部分 $\bar{X}_j$ 就是对到目前为止已经搜集到的知识的价值，而后一部分则可以看作是尚未充分探索过的节点需要继续探索的必要性。

2.3.2 Upper Confidence Bound Apply to Tree

虽然利用构造蒙特卡罗树搜索可以解决围棋博弈问题，但是由于蒙特卡罗树搜索在没有知识的指导时树的扩展层数较少，不利于最优解的获取，因此我们将 UCB1 算法加入蒙特卡罗树搜索的构建过程中，形成了我们即将要介绍的信心上限树算法。

UCT 算法是将 UCB1 算法思想用于蒙特卡罗树搜索的特定算法，其与只利用蒙特卡罗方法构建搜索树的方法的主要区别如下:

1. 可落子点的选择不是随机的，而是根据 UCB1 的信心上界值进行选择，如果可落子没有被访问，则其信心上限值为正无穷大，如果可落子点已经被访问过，则其信心上限索引值可以根据 UCB1 算法给出的值来确定。在实际应用中，我们采用 UCB1 给出的信心上限值，当我们需要在众多可下点中选取一个时，我们选择信心上限值最大的那一个。

2. 当模拟结束后确定最终选择结果时，我们不再仅仅根据胜率做出判断，而是兼顾信心上限所给出的估计值。在实际应用中，选择方法也是多种多样的，一些策略中选择估计值最大的子节点为落子点，另外由于选择可落子点进行访问的过程已经兼顾了极小极大算法的思想，所以在另外一些策略中也经常直接选择那个被访问了最多次的可落子点，等等。

3 Advanced Search

3.1 Local search

基本思想: 在搜索过程中，始终向着离目标最接近的方向搜索。该方法存在以下两个问题

(1) 局部最优问题

每次并不一定选择邻域内最优的点，而是依据一定的概率，从邻域内选择一个点，指标函数优的点，被选中的概率比较大，而指标函数差的点，被选中的概率比较小。

(2) 步长问题

一种可行的方法是将固定步长的搜索方法变为动态步长，开始时选择比较大的步长，随着搜索的进行，逐步减小步长。这样既解决了固定步长所带来的问题，又在一定程度上解决了小步长搜索耗时的问题。

3.2 Simulated annealing

模拟退火算法是局部搜索算法的一种扩展，该算法的思想最早由 Metropolis 在 1953 年提出，Kirkpatrick 等人在 1983 年成功地将模拟退火算法用于求解组合优化问题。为了更好的了解该算法，我们先来看固体退火的过程。

- 溶解过程

  随着温度的不断上升，粒子逐渐脱离开其平衡位置，变得越来越自由，直到达到固体的溶解温度，粒子排列从原来的有序状态变为完全的无序状态。

- 退火过程

  随着温度的下降，粒子的热运动逐渐减弱，粒子逐渐停留在不同的状态，其排列也从无序向有序方向发展，直至到温度很低时，粒子重新以一定的结构排列。

粒子不同的排列结构，对应着不同的能量水平。如果退火过程是缓慢进行的，也就是说，温度的下降如果非常缓慢的话，使得在每个温度下，粒子的排列都达到一种平衡态，则当温度趋于0(绝对温度) 时，系统的能量将趋于最小值。如果以粒子的排列或者相应的能量来表达固体所处的状态，在温度 T 下，固体所处的状态具有一定的随机性。一方面，物理系统倾向于能量较低的状态，另一方面，热运动又妨碍了系统准确落入低能状态。

组合优化问题与固体退火过程的类比关系见下表。

固体退火过程	组合优化问题
物理系统中的一个状态	组合优化问题的解
状态的能量	解的指标函数
能量最低状态	最优解
温度	控制参数

3.2.1 Principle of the Metropolis

设一个定义在有限集 S 上的组合优化问题，$i \in S$ 是该问题的一个解, 从解 i 到新解 j 的转移概率，按照 Metropolis 准则确定，即:

$$P_t(i \Rightarrow j) = \begin{cases} 1 & \text{如果} f(j) < f(i) \\ e^{-\frac{E(j)-E(i)}{Kt}} & \text{其他} \end{cases}$$

其中 $E(i)$、$E(j)$ 分别表示在状态 i、j 下的能量，T 是温度，$K > 0$ 是波尔兹曼常数。

在给定的温度 T 下，当进行足够多次的状态转换后，系统将达到热平衡。此时系统处于某个状态 i 的概率由波尔兹曼（Boltzmann）分布给出:

$$P_i(T) = \frac{e^{-\frac{E(i)}{KT}}}{Z_T}$$

其中 $Z_T = \sum_{j \in S} e^{-\frac{E(j)}{KT}}$ 为归一化因子，S 是所有可能状态的集合。下面，我们从四个角度考察一下上式随温度 T 的变化情况

1. 在给定的温度 T 下，设有 i、j 两个状态，$E(i) < E(j)$:

$$\begin{aligned} P_i(T) - P_j(T) &= \frac{e^{-\frac{E(i)}{KT}}}{Z_T} - \frac{e^{-\frac{E(j)}{KT}}}{Z_T} \\ &= \frac{1}{Z_T} e^{-\frac{E(i)}{KT}} \left(1 - \frac{e^{-\frac{E(j)}{KT}}}{e^{-\frac{E(i)}{KT}}}\right) \\ &= \frac{1}{Z_T} e^{-\frac{E(i)}{KT}} \left(1 - e^{-\frac{E(j)-E(i)}{KT}}\right) \end{aligned}$$

即在任何温度 T 下，系统处于能量低的状态的概率大于处于能量高的状态的概率。

2. 当温度趋于无穷时：

$$\lim_{T\to\infty}\left(P_i(T)\right)=\lim_{T\to\infty}\left(\frac{e^{-\frac{E(i)}{KT}}}{\sum_{j\in S}e^{-\frac{E(j)}{KT}}}\right)=\frac{1}{|S|}$$

其中 $|S|$ 表示系统所有可能的状态数。即当温度很高时，系统处于各个状态的概率基本相等，接近于平均值，与所处状态的能量几乎无关。

3. 当温度趋于零时：

设 S_m 表示系统最小能量状态的集合，E_m 是系统的最小能量。分子、分母同乘以 $e^{\frac{Em}{KT}}$

$$\begin{aligned}
\lim_{T\to 0}\left(P_i(T)\right)&=\lim_{T\to 0}\left(\frac{e^{-\frac{E(i)}{KT}}}{\sum_{j\in S}e^{-\frac{E(j)}{KT}}}\right)\\
&=\lim_{T\to 0}\left(\frac{e^{-\frac{E(i)-E_m}{KT}}}{\sum_{j\in S}e^{-\frac{E(j)-E_m}{KT}}}\right)\\
&=\lim_{T\to 0}\left(\frac{e^{-\frac{E(i)-E_m}{KT}}}{\sum_{j\in S_m}e^{-\frac{E(j)-E_m}{KT}}+\sum_{j\notin S_m}e^{-\frac{E(j)-E_m}{KT}}}\right)\\
&=\lim_{T\to 0}\left(\frac{e^{-\frac{E(i)-E_m}{KT}}}{\sum_{j\in S_m}e^{-\frac{E(j)-E_m}{KT}}}\right)\\
&=\begin{cases}\frac{1}{|S_m|} & if\ i\in S_m\\ 0 & if\ i\notin S_m\end{cases}
\end{aligned}$$

即当温度趋近于 0 时，系统以等概率趋近于几个能量最小的状态之一，而系统处于其他状态的概率为 0。以概率 1 达到能量最小的状态。

4. 当温度上升或下降时：

$$\begin{aligned}
\frac{\partial P_i(T)}{\partial T}&=\frac{\partial}{\partial T}\left(\frac{e^{-\frac{E(i)}{KT}}}{Z_T}\right)\\
&=\frac{E(i)}{KT^2}\frac{e^{-\frac{E(i)}{KT}}}{Z_T}-\frac{1}{Z_T^2}e^{-\frac{E(i)}{KT}}\frac{\partial Z_T}{\partial T}\\
&=\frac{E(i)}{KT^2}P_i(T)-\frac{P_i(T)}{Z_T}\frac{\partial Z_T}{\partial T}\\
&=\frac{E(i)}{KT^2}P_i(T)-\frac{P_i(T)}{Z_T}\frac{1}{KT^2}\sum_{j\in S}E(j)e^{-\frac{E(j)}{KT}}\\
&=\frac{P_i(T)}{KT^2}\left(E(i)-\sum_{j\in S}E(j)\frac{e^{-\frac{E(j)}{KT}}}{Z_T}\right)\\
&=\frac{P_i(T)}{KT^2}\left(E(i)-\sum_{j\in S}E(j)P_j(T)\right)\\
&=\frac{P_i(T)}{KT^2}\left(E(i)-\overline{E_T}\right)
\end{aligned}$$

所以我们有

$$\frac{\partial P_i(T)}{\partial T} = \begin{cases} > 0 & if \quad E(i) > \overline{E_T} \\ < 0 & if \quad E(i) < \overline{E_T} \end{cases}$$

系统落入低能量状态的概率随着温度的下降单调上升，而系统落入高能量状态的概率随着温度的下降单调下降。

在高温下，系统基本处于无序的状态，基本以等概率落入各个状态。在给定的温度下，系统落入低能量状态的概率大于系统落入高能量状态的概率，这样在同一温度下，如果系统交换的足够充分，则系统会趋向于落入较低能量的状态。随着温度的缓慢下降，系统落入低能量状态的概率逐步增加，而落入高能量状态的概率逐步减少，使得系统各状态能量的期望值随温度的下降单调下降，而只有那些能量小于期望值的状态，其概率才随温度下降增加，其他状态均随温度下降而下降。因此，随着能量期望值的逐步下降，能量低于期望值的状态逐步减少，当温度趋于 0 时，只剩下那些具有最小能量的状态，系统处于其他状态的概率趋近于 0。因此最终系统将以概率 1 处于具有最小能量的一个状态。容易理解的，达到最小能量状态有三个条件：

- 初始温度必须足够高
- 在每个温度下，状态的交换必须足够充分
- 温度 T 的下降必须足够缓慢

3.2.2 Parameter Setting

大量的实验表明，解的质量与算法的运行时间是成正比关系的，很难做到两全其美。下面我们给出模拟退火算法中一些参数或者准则的确定方法, 试图在求解时间与解的质量之间作一个折中的选择。这些参数或者准则包括:

- 初始温度 $= t_0$
- 温度 t 的衰减函数，即温度的下降方法;
- 算法的终止准则，用终止温度 t_f 或者终止条件给出;
- 每个温度 t 下的马尔可夫链长度 L_k

具体的我们有

(1) 起始温度的选取

起始温度应保证平稳分布中每一个状态的概率基本相等，也就是接受概率 P_0 近似等于 1。在 Metropolis 准则下，即要求 $e^{-\frac{\Delta f(i,j)}{t_0}} \approx 1$。如果我们给定一个比较大的接受概率 P_0, 比如 $P_0 = 0.9$ 或者 0.8, 就可以计算出 t_0 值:

$$t_0 = \frac{\Delta f(i,j)}{\ln\left(P_0^{-1}\right)}$$

其中，$\Delta f(i,j)$ 可以有以下估计方式:

$$\begin{aligned}\Delta f(i,j) =& \max_{i\in S}(f(i)) - \min_{i\in S}(f(i)) \\ \Delta f(i,j) =& \frac{\sum_{i,j\in S}|f(i)-f(j)|}{|S|^2} \\ \Delta f(i,j) =& \frac{\sum_{i=0}^{|S|-1}|f\left(S'(i)\right)-f\left(S'(i+1)\right)|}{|S'|}\end{aligned}$$

其中 S' 是由 S 随机产生的一个有序集，$S'(i)$ 表示 S' 的第 i 个元素。

(2) 温度下降的方法

最简单的，我们有等值下降和等比例下降。等值下降每次下降一个固定的值；等比例下降方法通过设置一个衰减系数（一般可以选取 0.8 0.95），使得温度每次以相同的比率下降:

$$t_{k+1} = \alpha t_k \quad 0 < \alpha < 1$$

在一定的条件下，与模拟退火算法相伴的时齐马尔可夫链存在平稳分布。如果温度每次下降的幅度比较小的话，则相邻温度下的平稳分布应该变化不大，也就是说，对于任意一个状态 $i \in S$ ，相邻温度下的平稳分布应满足:

$$\frac{1}{1+\delta} < \frac{P_{t_k}(i)}{P_{t_{k+1}}(i)} < 1+\delta$$

其中 δ 是一个较小的正数，被称作距离参数。使得上式成立的一个充分条件是对于任意一个状态 $i \in S$ ，有:

$$\frac{e^{-\frac{f(i)-f_m}{t_k}}}{e^{-\frac{f(i)-f_m}{t_{k+1}}}} < 1+\delta, k = 0,1,\ldots$$

其中 f_m 表示最优解的指标函数值。由式（7.40）有:

$$e^{-\frac{f(i)-f_m}{t_k}} < (1+\delta)e^{-\frac{f(i)-f_m}{t_{k+1}}}, k = 0,1,\ldots$$

两边取对数，整理可得:

$$t_{k+1} > \frac{t_k}{1+\frac{t_k\ln(1+\delta)}{f(i)-f_m}}, k = 0,1,\ldots$$

用 $3\sigma_{t_k}$ 代替 $f(i)-f_m$ 可得温度的衰减函数:

$$t_{k+1} = \frac{t_k}{1+\frac{t_k\ln(1+\delta)}{3\sigma_{t_k}}}, k = 0,1,\ldots$$

其中 σ_{t_k} 为温度 $\mathrm{t_k}$ 下所产生的状态的指标函数值的标准差，实际计算时，可通过样本值计算近似得到。

(3) 每一温度下的停止准则

最容易想到的，固定长度方法。即在每一个温度下，都使用相同的 L_k。L_k 的选取与具体的问题相关，一般与邻域的大小直接关联，通常选择为问题规模 n 的一个多项式函数。

由前面我们对退火过程的分析知道，在比较高的温度时，系统处于每一个状态的概率基本相同，而且每一个状态的接受概率也接近于 1。因此在高温时，即便比较小的迭代数，也可以基本达到平稳状态。而随着温度的下降，被拒绝的状态数随之增加，因此在低温下迭代数应增加，以免由于迭代数太少，而过早的陷入局部最优状态。因此一个直观的想法就是随着温度的下降适当的增加迭代次数。

一种方法就是，规定一个接受次数 R，在某一温度下，只有被接受的状态数达到 R 时，在该温度下的迭代才停止，转入下一个温度。由于随着温度的下降，状态被接受的概率随之下降，因此这样的一种准则是满足随着温度的下降适当的增加迭代次数的。但由于在温度比较低时，接受概率很低，为了防止出现过多的迭代次数，一般设置一个迭代次数的上限，当迭代次数达到上限时，即便不满足接受次数 R，也停止这一温度的迭代过程。

(4) 算法的终止准则

从理论上讲，当温度趋近于 0 时，模拟退火算法才结束。因此，可以设定一个正常数 e，当 $t_k < e$ 时，算法结束。此方法称为零度法。

还容易想到循环控制法。给定一个指定的温度下降次数 K，当温度的迭代次数达到 K 次时，则算法停止。但这要求给定一个合适的 K。如果 K 值选择不合适，对于小规模问题将导致增加算法无谓的运行时间，而对于大规模问题，则可能难于得到高质量的解。

为了改进循环控制法中的问题，我们有无变化控制法。如果在相邻的 n 个温度中，得到的解的指标函数值无任何变化，则说明算法已经收敛。

3.3 Genetic Algorithm

遗传算法是根据自然界的“物竞天择、适者生存”现象而提出的一种随机搜索算法，70 年代由美国的密执根大学的 Holland 教授首先提出。该算法将优化问题看作是自然界中生物的进化过程，通过模拟大自然中生物进化过程中的遗传规律，来达到寻优的目的。

在遗传算法中，首先对优化问题的解进行编码，编码后的一个解称为一个染色体，组成染色体的元素称为基因。一个群体由若干个染色体组成，染色体的个数称为群体的规模。与自然界中的生存环境相对应，在遗传算法中用适应函数表示环境，它是已编码的解的函数，是一个解适应环境程度的评价。适应函数的构造一般与优化问题的指标函数相关，简单的情况下，直接用指标函数或者指标函数经过简单的变换后作为适应函数使用。一般情况下，适应函数值大表示所对应的染色体适应环境的能力强。适应函数起着自然界中环境的作用，当适应函数确定后，自然选择规律将以适应函数值的大小来决定一个染色体是否继续生存下去的概率。生存下来的染色体成为种群，它们中的部分或者全部以一定的概率进行交叉，繁衍得到下一代的群体。交叉是一个生殖过程，发生在两个染色体之间，作为双亲的两个染色体，交换部分基因之后，生殖出两个新的染色体，即问题的新解。交叉或者称作杂交，是遗传算法区别于其他优化算法的

最主要的特征。在进化的过程中，染色体的某些基因可能会发生变异，即表示染色体的编码发生了某些变化。一个群体的进化需要染色体的多样性，而变异对保持群体的多样性具有一定的作用。下给出了遗传算法与生物进化之间的对应关系。

生物进化中的概念	遗传算法中的作用
环境	适应函数
适应性	适应值函数
适者生存	适应函数值最大的解被保留的概率最大
个体	问题的一个解
染色体	解的编码
基因	编码的元素
群体	被选定的一组解(以编码形式表示)
种群	根据适应函数选择的一组解(
交叉	以一定的方式由双亲产生后代的过程
变异	编码的某些分量发生变化的过程

3.3.1 Operation

选择、交叉和变异是遗传算法的三个主要操作。下面分别说明。

(1) Selection

依据适应值的大小，选择操作从规模为 N 的群体中随机的选择若干染色体构成种群，种群的规模可以与原来的群体的规模一致，也可以不一致，我们这里假设二者的规模是一致的，即从群体中选择 N 个染色体构成种群。由于是依据适应值的大小进行随机选择的，因此虽然种群的规模与群体的规模是一致的，但二者并不完全一样，因为适应值大的染色体可能会被多次从群体选出，而适应值小的染色体有可能会失去被选中的机会。

- 轮盘赌方法

  设群体的规模为 N, $F(x_i)\,(i=1,\ldots,N)$ 是其中 N 个染色体的适应值。则第 i 个染色体被选中的概率由下式给出:

$$p(x_i)=\frac{F(x_i)}{\sum_{j=1}^{N}F(x_j)}$$

  有一个共有 N 个格子的转盘, 每一个 x_i 在转盘上占有一格, 而格子的大小与 $p(x_i)$ 成正比关系。在选择一个染色体时，先转动轮盘，待转盘停下后，指针指向的格子所对应的 x_i 就是被选中的染色体。

- 确定性方法

  对于规模为 N 的群体，一个选择概率为 $p(x_i)$ 的染色体 x_{i} 被选择次数的期望值

$$e(x_i)=p(x_i)N$$

对于群体中的每一个 x_i, 首先选择 $\lfloor e(x_i) \rfloor$ 次。这样共得到 $\sum_{i=1}^{N} \lfloor e(x_i) \rfloor$ 个染色体。然后按照 $e(x_i) - \lfloor e(x_i) \rfloor$ 从大到小对染色体排序，依次取出 $N - \sum_{i=1}^{N} \lfloor e(x_i) \rfloor$ 个染色体，这样就得到了 N 个染色体，以这 N 个染色体组成种群。

(2) Crossover

交叉发生在两个染色体之间，由两个被称之为双亲的父代染色体，经交叉以后，产生两个具有双亲的部分基因的新的染色体。当染色体采用二进制形式编码时，交叉过程是以这样一种形式进行的:

设 a 、b 是两个交叉的染色体:

$$a : a_1 a_2 \dots a_i a_{i+1} \dots a_n$$

$$b : b_1 b_2 \dots b_i b_{i+1} \dots b_n$$

其中 a_i 、$b_i \in \{0, 1\}$。随机的产生一个交叉位设为 i, 则 a, b 两个染色体从 $i+1$ 以后的基因进行交换，得到两个新的染色体。

(3) Mutation

变异发生在染色体的某一个基因上，当以二进制编码时，变异的基因由 0 变成 1，或者由 1 变成 0。如对于染色体 x=11001，如果变异位发生在第三位，则变异后的染色体变成了 y=11101

3.3.2 Algorithm

下面给出算法的伪代码。

(1) 给定群体规模 N，交叉概率 p_c 和变异概率 p_m，$t = 0$;

(2) 随机生成 N 个染色体作为初始群体;

(3) 对于群体中的每一个染色体 x_i 分别计算其适应值 $F(x_i)$;

(4) 如果算法满足停止准则，则转（10）;

(5) 对群体中的每一个染色体 x_i 计算概率;

(6) 依据计算得到的概率值，从群体中随机的选取 N 个染色体，得到种群;

(7) 依据交叉概率 p_c 从种群中选择染色体进行交叉，其子代进入新的群体，种群中未进行交叉的染色体，直接复制到新群体中;

(8) 依据变异概率 p_m 从新群体中选择染色体进行变异，用变异后的染色体代替新群体中的原染色体;

(9) 用新群体代替旧群体，$t = t + 1$，转（3）;

(10) 进化过程中适应值最大的染色体，经解码后作为最优解输出；

(11) 结束。

我们有如下收敛性定理来保证算法的的正确性。如果在代的进化过程中，遗传算法每次保留到目前为止的最好解，并且算法以交叉和变异为其随机化操作，则对于一个全局最优化问题，当进化代数趋于无穷时，遗传算法找到最优解的概率为 1。

作为一个随机搜索算法，遗传算法适用于数值求解具有多参数、多变量、多目标等复杂的最优化问题。该算法对待求解问题的指标函数没有什么特殊的要求，比如不要求诸如连续性、导数存在、单峰值假设等。甚至于不需要显式的写出指标函数。在经过编码以后，遗传算法几乎不需要任何与问题有关的知识，唯一需要的信息是适应值的计算。也不需要使用者对问题有很深入的了解和求解技巧，通过选择、交叉和变异等简单的操作求解复杂的问题，是一个比较通用的优化算法。最后值得一提的是，遗传算法具有天然的并行性，适用于并行求解。

3.3.3 Implementation

(1) 编码

常用的是手段是二进制编码，但该方法有些情况会出现问题。例如对于 n 城市的旅行商问题，至少需要用 $n \times n$ 位二进制向量表示一个可能的旅行路线。一个 $n \times n$ 位二进制向量，所有可能的编码个数为 2^{n^2} ，而一个对称的 n 城市旅行商问题的可能解个数为 $n!/2$，只占编码个数非常小的比例。以 n 10 为例，编码个数为可能解个数的 7.0×10^{23} 倍。可能解在整个状态空间中，是非常稀疏的，交叉和变异所产生的是大量的非可能解。

(2) 评价

- 当前最好法

  该方法在每一代进化过程中，记录得到的最好解，通过最好解的变化，了解算法的变化趋势。不同的算法之间，也可以通过该最好解的变化情况进行横向比较。

- 在线比较法

  该方法用当前代中染色体的平均指标函数值来刻划算法的变化坎势。计算方法如下:

  $$v_{online} = \frac{1}{T}\sum_{t=1}^{T} f(t)$$

  其中 T 为当前代中染色体的个数。

- 离线比较法

  该方法与在线比较法有些相似，但是用进化过程中每代最好解的指标函数值的平均值，来评价算法的进化过程。计算方法如下:

  $$v_{offline} = \frac{1}{T}\sum_{t=1}^{T} f^*(t)$$

  其中 T 是到目前为止的进化代数，$f^*(t)$ 是第 t 代中，染色体的最好指标函数值。

以上方法都可以监控算法的进化趋势，掌握遗传算法的进化情况，从而决定算法是否停止。

(3) 适应函数

一般情况下，我们可以直接选取问题的指标函数作为适应函数。如求函数 $f(x)$ 的最大值，就可以直接采用 $f(x)$ 为适应函数。但在有些情况下，函数 $f(x)$ 在最大值附近的变化可能会非常小，以至于他们的适应值非常接近，很难区分出哪个染色体占优。在这种情况下，希望定义新的适应函数，要求该适应函数与问题的指标函数具有相同的变化趋势，但变化的速度更快。

- 非线性加速适应函数

$$f'(x)=\begin{cases}\frac{1}{f_{\max}-f(x)}, & \text{如果} f(x)<f_{\max}\\ M, & \text{其他}\end{cases}$$

其中 $f(x)$ 是问题的指标函数，$f_{\max}$ 是当前得到的最优指标函数值，M 是一个充分大的数。

- 线性加速适应函数

$$f'(x)=\alpha f(x)+\beta$$

其中 α 、β 由下述方程确定:

$$\begin{cases}\alpha\frac{\sum_{i=1}^{m}f(x_i)}{m}+\beta=\frac{\sum_{i=1}^{m}f(x_i)}{m}\\ \alpha\max_{1\le i\le m}\{f(x_i)\}+\beta=M\frac{\sum_{i=1}^{m}f(x_i)}{m}\end{cases}$$

其中 $x_i(i=1,2,\ldots,m)$ 为当前代中的染色体。两个方程式中的第一个方程表示变换前后的平均值不变，第二个方程表示将当前的最优值放大为平均值的 M 倍。

(4) 交叉规则

对于二进制编码:

- 双亲双子法，就是前面已经介绍过的最常用的交叉方法。
- 多交叉位法，顾名思义，多交叉位法就是不止产生一个交叉位，而是产生多个交叉位进行交叉。在交叉时采用交叉区间交替进行的方法。
- 双亲单子法，该方法是两个染色体交叉后，只产生一个子染色体。一般是从一般的交叉法得到的两个子染色体中，随机的选择一个，或者选择适应值大的那一个子染色体。

在旅行商问题，一个很自然的想法就是采用 1 到 n 的排列来表示一个可能解。即所谓的整数编码。在这种情况下，如果采取交换两个父染色体的部分基因的方法进行交叉的话，所产生的子染色体不一定刚好是 1 到 n 的一个排列，从而产生无效解。对于整数编码:

- 常规交叉法

  随机选取一个交叉位，子代 1 交叉位之前的基因选自父代 1 交叉位之间的基因，交叉位之后的基因，从父代 2 中按顺序选取那些没有出现过的基因。

- 基于次序的交叉法

  对于两个选定的父代染色体父代 1 和父代 2 , 首先随机的选定一组位置。设父代 1 中与所选位置相对应的数字从左到右依次为 $x_1, x_2, \ldots, x_k$, 然后从父代 2 中也找到这 k 个数字, 并从父代 2 中把它们去除, 这样在父代 2 中就有了 k 个空位置。用 $x_1, x_2, \ldots x_k$ 依次填入到父代 2 的空位置中, 就得到了交叉后的一个子染色体子代 1 。采用同样的方法处理父代 1 ，可以得到另一个子染色体子代 2 。

- 基于部分映射的交叉法

  对于两个选定的父代染色体父代 1 和父代 2，随机产生两个位置，两个父代在这两个位置之间的基因产生对应对，然后用这种对应对分别去替换两个父代的基因（双向对应的），从而产生两个子代。

(5) 变异规则

- 二进制编码，01 替换。
- 整数编码

  基于位置的变异, 随机的产生两个变异位，然后将第二个变异位上的基因移动到第一变异位之前；基于次序的变异，随机的产生两个变异位，然后将第二个变异位上的基因移动到第一变异位之前；打乱变异，随机选取染色体上的一段，然后打乱在该段内的基因次序。

(6) 终止条件

通常，我们有以下终止条件

- 计算耗费的资源限制（例如计算时间、计算占用的内存等）
- 进化次数限制
- 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
- 以上的组合

(7) 性能评价

如何对遗传算法的性能进行评价是一件很困难的事情，为此需要确定性能度量和一些具有代表性的函数。一般可以用达到最优点时性能函数值的平均计算次数和计算所需要的时间来进行度量。一些学者给出了如下的测试函数集，这些函数或者是多峰值的，或者是不连续的，或者是具有一定的噪声的，对于求解他们的最小值具有一定的难度。

$$f_{11}(x) = \frac{\pi}{n}\left\{k_1 \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - k_2)^2\left[1 + k_1 \sin^2(\pi y_{i+1})\right] + (y_n - k_2)^2\right\} \quad -10 \le x_i \le 10$$

4 Statistical Machine Learning

4.1 Naïve Bayes

朴素贝叶斯法是一种常见有效的分类方法，基于特征条件独立假设学习输入/输出的联合概率分布，以此为模型，对于给定的输入 x，利用贝叶斯定理求出后验概率最大的输出 y。

设输入空间 $X \subseteq R^n$ 为 n 维向量的集合，输出空间为类标记集合 $Y = \{c_1, c_2, \ldots, c_k\}$。$X$ 是定义在输入空间上的随机变量，ΔY 是定义在输出空间上的随机变量。$P(X,Y)$ 是 X 和 Y 的联合概率分布，$T = \{(x_1, y_1), (x_2, y_2) \ldots (x_N, y_N)\}$ 是由 $P(X,Y)$ 独立同分布产生的训练集。

贝叶斯分类器:

$$\begin{aligned} y &= \arg\max_{c_k} \frac{P(X = x \mid Y = c_k) P(Y = c_k)}{\sum_k P(X = x \mid Y = c_k) P(Y = c_k)} \\ &= arg\max_{c_k} P(X = x \mid Y = c_k) P(Y = c_k) \end{aligned}$$

$P(X = x \mid Y = c_k) = P\left(X^{(1)} = x^{(1)}, \ldots, X^{(n)} = x^{(n)} \mid Y = c_k\right)$ 具有指数量级的参数，实际上不可计算。为此我们引入独立性假设，得到朴素贝叶斯分类器

$$y = \arg\max_{c_k} P(Y = c_k) \prod_{j=1}^{n} P\left(X^{(j)} = x^{(j)} \mid Y = c_k\right)$$

我们有

$$\begin{aligned} &P(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k) + \lambda}{N + K\lambda}, k = 1, 2, \ldots, K \\ &\text{where } I(y_i = c_k) = \begin{cases} 1, & \text{if } y_i = c_k \\ 0, & \text{if } y_i \neq c_k \end{cases} \end{aligned}$$

$$\begin{aligned} &P\left(X^{(j)} = a_{jl} \mid Y = c_k\right) = \frac{\sum_{i=1}^{N} I\left(x_i^{(j)} = a_{jl}, y_i = c_k\right) + \lambda}{\sum_{i=1}^{N} I(y_i = c_k) + S_j \lambda} \\ &\text{where } j = 1, 2, \ldots, n; l = 1, 2, \ldots, S_j; k = 1, 2, \ldots, K \end{aligned}$$

$\left\{a_{j1}, a_{j2}, \ldots, a_{jS_j}\right\}$ 为第 j 个特征 $x^{(j)}$ 可能取值的集合。常取 $\lambda = 1$，称为拉普拉斯平滑。S_j 是第 j 个特征的取值数

4.2 Support Vector Machines

支持向量机是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器。

4.2.1 Principle

假设给定一个特征空间上的训练数据集

$$T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

其中, $x_i \in R^n$, $\quad y_i \in \{+1, -1\}, i = 1, 2, \ldots N$, $\quad x_i$ 为第 i 个特征向量, y_i 为类标记，当它等于 +1 时为正例; 为-1 时为负例。再假设训练数据集是线性可分的。对于给定的数据集 T 和超平面 $w \cdot x + b = 0$, 超平面关于样本点 (x_i, y_i) 的几何间隔为

$$\gamma_i = y_i \left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right)$$

超平面关于所有样本点的几何间隔的最小值为

$$\gamma = \min_{i=1,2\ldots,N} \gamma_i$$

实际上这个距离就是我们所谓的支持向量到超平面的距离。根据以上定义，SVM 模型的求解最大分割超平面问题可以表示为以下约束最优化问题

$$\begin{aligned} &\max_{w,b} \gamma \\ s.t. \quad &y_i \left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|} \right) \geq \gamma, i = 1, 2, \ldots, N \end{aligned}$$

将约束条件两边同时除以 γ, 得到

$$y_i \left(\frac{w}{\|w\|\gamma} \cdot x_i + \frac{b}{\|w\|\gamma} \right) \geq 1$$

因为 $\|w\|, \gamma$ 都是标量，所以为了表达式简洁起见，令 $w = \frac{w}{\|w\|\gamma}$ $b = \frac{b}{\|w\|\gamma}$ 得到

$$y_i \left(w \cdot x_i + b \right) \geq 1, i = 1, 2, \ldots, N$$

又因为最大化 γ, 等价于最大化 $\frac{1}{\|w\|}$, 也就等价于最小化 $\frac{1}{2}\|w\|^2$, 因此 SVM 模型的求解最大分割超平面问题又可以表示为以下约束最优化问题

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ s.t. \quad &y_i \left(w \cdot x_i + b \right) \geq 1, i = 1, 2, \ldots, N \end{aligned}$$

这是一个含有不等式约束的凸二次规划问题，可以对其使用拉格朗日乘子法得到其对偶问题。首先，我们将有约束的原始目标函数转换为无约束的新构造的拉格朗日目标函数

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left(y_i \left(w \cdot x_i + b \right) - 1 \right)$$

其中 α_i 为拉格朗日乘子，且 $\alpha_i \geq 0$ 。现在我们令

$$\theta(w) = \max_{\alpha_i \geq 0} L(w, b, \alpha)$$

当样本点不满足约束条件时，即在可行解区域外: $y_i \left(w \cdot x_i + b \right) < 1$ 此时，将 α_i 设置为无穷大，则 $\theta(w)$ 也为无穷大。当满本点满足约束条件时，即在可行解区域内: $y_i \left(w \cdot x_i + b \right) \geq 1$ 此时, $\theta(w)$ 为原函数本身。于是，将两种情况合并起来就可以得到我们新的目标函数

$$\theta(w) = \begin{cases} \frac{1}{2}\|w\|^2, x \in \text{ 可行区域} \\ +\infty, x \in \text{ 不可行区域} \end{cases}$$

于是原约束问题就等价于

$$\min_{w,b}\theta(w)=\min_{w,b}\max_{\alpha_i\geq 0}L(w,b,\alpha)=p^*$$

看一下我们的新目标函数，先求最大值，再求最小值。这样的话，我们首先就要面对带有需要求解的参数 w 和 b 的方程, 而 α_i 又是不等式约束，这个求解过程不好做。所以，我们需要使用拉格朗日函数对偶性, 将最小和最大的位置交换一下，这样就变成了:

$$\max_{\alpha_i\geq 0}\min_{w,b}L(w,b,\alpha)=d^*$$

这是凸优化问题，要有 $p^*=d^*$, 只需满足 KKT 条件，即要求

$$\begin{cases}\nabla_{w,b}L(w,b,\alpha)=0\\ \alpha_i\geq 0\\ y_i\left(w_i\cdot x_i+b\right)-1\geq 0\\ \alpha_i\left(y_i\left(w_i\cdot x_i+b\right)-1\right)=0\end{cases}$$

令 $L(w,b,\alpha)$ 对 w 和 b 的偏导为 0, 可得 $w=\sum_{i=1}^{N}\alpha_i y_i x_i$，$\sum_{i=1}^{N}\alpha_i y_i=0$ 将以上两个等式带入拉格朗日目标函数，消去 w 和 b，得

$$\begin{aligned}L(w,b,\alpha)&=\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(x_i\cdot x_j\right)-\sum_{i=1}^{N}\alpha_i y_i\left(\left(\sum_{j=1}^{N}\alpha_j y_j x_j\right)\cdot x_i+b\right)+\sum_{i=1}^{N}\alpha_i\\&=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(x_i\cdot x_j\right)+\sum_{i=1}^{N}\alpha_i\end{aligned}$$

即

$$\min_{w,b}L(w,b,\alpha)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(x_i\cdot x_j\right)+\sum_{i=1}^{N}\alpha_i$$

求 $\min_{w,b}L(w,b,\alpha)$ 对 α 的极大，即是对偶问题

$$\begin{aligned}&\max_{\alpha}-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(x_i\cdot x_j\right)+\sum_{i=1}^{N}\alpha_i\\&s.t.\sum_{i=1}^{N}\alpha_i y_i=0\\&\alpha_i\geq 0, i=1,2,\ldots,N\end{aligned}$$

将求解极大转换为求解极小

$$\begin{aligned}&\min_{\alpha}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j\left(x_i\cdot x_j\right)-\sum_{i=1}^{N}\alpha_i\\&s.t.\sum_{i=1}^{N}\alpha_i y_i=0\\&\alpha_i\geq 0, i=1,2,\ldots,N\end{aligned}$$

现在我们的优化问题变成了如上的形式。对于这个问题，我们有更高效的优化算法，即序列最小优化（SMO）算法（Platt，1998）。

4.2.2 Slack Variables

上面都是基于训练集数据线性可分的假设下进行的，但是实际情况下几乎不存在完全线性可分的数据，为了解决这个问题，引入了" 软间隔" 的概念，即允许某些点不满足约束

$$y_j\left(w \cdot x_j + b\right) \geq 1$$

采用 hinge 损失，将原优化问题改写为

$$\begin{aligned} &\min_{w,b,\xi_i} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ &s.t. y_i\left(w \cdot x_i + b\right) \geq 1 - \xi_i \\ &\xi_i \geq 0, i = 1, 2, \ldots, N \end{aligned}$$

其中 ξ_i 为" 松弛变量",

$$\xi_i = \max\left(0, 1 - y_i\left(w \cdot x_i + b\right)\right)$$

即一个 hinge 损失函数。每一个样本都有一个对应的松他变量，表征该样本不满足约束的程度。$C > 0$ 称为惩罚参数, C 值越大，对分类的惩罚越大。跟线性可分求解的思路一致，同样这里先用拉格朗日乘子法得到拉格朗日函数，再求其对偶问题。

4.2.3 Kernal Method

我们定义核函数如下，设 $\mathcal{X}$ 是输入空间（即 $x_i \in \mathcal{X}$，　$\mathcal{X}$ 是 R^n 的子集或离散集合) , 又设 $\mathcal{H}$ 为特征空间 ($\mathcal{H}$ 是希尔伯特空间)，如果存在一个从 $\mathcal{X}$ 到 $\mathcal{H}$ 的映射

$$\phi(x) : \mathcal{X} \to \mathcal{H}$$

使得对所有 $x, z \in \mathcal{X}$, 函数 $K(x, z)$ 满足条件

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

则称 K 为核函数。其中 $\phi(x)$ 为映射函数, $\langle \cdot, \cdot \rangle$ 为内积。即核函数输入两个向量，它返回的值跟两个向量分别作 ϕ 映射然后点积的结果相同。常用的核函数有:

名称	表达式
线性核	$\kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j$
多项式核	$\kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j\right)^d$
高斯核	$\kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$
拉普拉斯核	$\kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{\sigma}\right)$
Sigmoid 核	$\kappa\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \tanh\left(\beta\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + \theta\right)$

不知道 ϕ 的情况下，如何判断某个 K 是不是核函数? 答案是 K 是核函数当且仅当对任意数据 $D = x_1, x_2, \ldots, x_m$, 核矩阵 (kernal matrix, gram matrix) 总是半正定的:

$$\text{Kernal Matrix } = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \ldots & K(x_1, x_m) \\ K(x_2, x_1) & K(x_2, x_2) & \ldots & K(x_2, x_m) \\ \vdots & \vdots & & \vdots \\ K(x_m, x_1) & K(x_m, x_2) & \ldots & K(x_m, x_m) \end{bmatrix}$$

总结: 核方法是一类把低维空间的非线性可分问题，转化为高维空间的线性可分问题的方法。核函数输入两个向量，它返回的值跟两个向量分别作 ϕ 映射然后点积的结果相同。核技巧是一种利用核函数直接计算 $\langle \phi(x), \phi(z) \rangle$, 以避开分别计算 $\phi(x)$ 和 $\phi(z)$, 从而加速核方法计算的技巧。核方法不仅仅适用于 SVM，还适用于其他数据为非线性可分的问题和算法。SVM 的表现形式包含了映射的点积，所以可以用核技巧加速核方法的计算。

4.3 Decision Tree

决策树模型是一种描述对实例进行分类的树形结构，由节点和有向边组成。节点有两种类型: 内部节点和叶节点。内部节点表示一个特征或者属性，叶节点表示一个类。

决策树学习就是从训练集中归纳出一组分类规则，得到一个与训练集矛盾较小的决策树。对于给定的训练集，可以构造出多个决策树，一般以损失函数最小化作为优化目标。从所有决策树中选取最优决策树是一个 NPC 问题，所以一般采用启发式方法，得到一个近似解。决策树学习主要包括以下三部分:

(1) 特征选择

一个问题中可能有不同的特征，不同的特征具有不同的分类能力，特征选择就是如何选取出那些分类能力强的特征。决策树中一般按照信息增益选择特征，所谓的信息增益就是某个特征 A 对数据集 D 进行分类的不确定性减少的程度。

随机变量 X 的熵:

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i$$

其中 $p_i = P(X = x_i)$, 也记作 $H(p)$. 当概率由数据集 D 估计得到时，记作 $H(D)$

条件熵:

$$H(Y \mid X) = \sum_{i=1}^{n} p_i H(Y \mid X = x_i)$$

表示已知 X 时 Y 的不确定性

特征 A 对数据集 D 的信息增益定义为:

$$g(D, A) = H(D) - H(D \mid A)$$

表示特征 A 对数据集 D 的分类的不确定性减少的程度，信息增益大的特征具有更强的分类能力。下面我们举例说明。

设训练集 D，K 个类 C_k ，特征 A 有 n 个不同的取值 $\{a_i, \ldots, a_n\}$, A 的不同取值将 D 划分为 n 个子集 $D_1 \ldots D_n$，D_i 中属于类 C_k 的样本的集合为 D $_{ik}$。信息增益计算如下:

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

$$H(D \mid A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H\left(D_i\right) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

$$g(D, A) = H(D) - H(D \mid A)$$

(2) 决策树生成

一个基本的决策树生成算法是 ID3 算法。输入: 训练集 D，特征集 A，阈值 $\epsilon > 0$, 输出: 决策树 T。ID3 算法伪代码如下:

(1) 若 D 中所有实例属于同一类 C_k ，则 T 为单节点树，将 C_k 作为该节点的类标记，返回 T

(2) 若 A 为空，则 T 为单节点树，将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T

(3) 否则计算 A 中各特征对 D 的信息增益，选择信息最大的特征 A_g

(4) 如果 A_g 的信息增益小于阈值 ε ，则置 T 为单节点树，将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T

(5) 否则对 A_g 的每一可能值 a_i, 依 $A_g = a_i$ 将 D 分割为若干子集 D_i ，作为 D 的子节点

(6) 对于 D 的每个子节点 D_i，如果 D_i 为空，则将 D 中实例最大的类作为标记，构建子节点

(7) 否则以 D_i 为训练集，以 $A - \{A_g\}$ 为特征集，递归地调用步 1 步 6，得到子树 T_i，返回 T_i

ID3 存在信息增益倾向于选择分枝比较多的属性的问题。C4.5 算法是对 ID3 算法的改进。C4.5 根据信息增益比来选择特征，具体的，

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)}$$

$$H_A(D) = -\sum_{k=1}^{n} \frac{|D_k|}{|D|} \log_2 \frac{|D_k|}{|D|}$$

其中 A 为属性，A 的不同取值将 D 划分为 n 个子集 $D_1 \ldots D_n$。同时 C4.5 增加了对连续值属性的处理，对于连续值属性 A，找到一个属性值 a_0 ，将 $\leq a_0$ 的划分到左子树，$> a_0$ 的划分到右子树。

但信息增益比存在倾向于选择分割不均匀的特征的问题，为此我们可以先选择 n 个信息增益大的特征，再从这 n 个特征中选择信息增益比最大的特征。

(3) 决策树剪枝

为了防止出现过拟合，对生成的决策树进行简化的过程称为剪枝。也就是从已经生成的树上裁掉一些子树或者叶节点，将其父节点作为新的页节点，用其实例数最大的类别作为标记。这种先生成树再剪枝的方法称为后剪枝。

当数据量大时，将数据划分为训练集、验证集和测试集。用训练集训练得到决策树，从下向上逐步剪枝；在验证集上测试性能，直到性能下降为止；最后在测试集上的性能作为系统的性能。

当数据量小时，直接利用训练集进行剪枝。

树 T 的叶节点个数为 $|T|$，t 是树 T 的叶节点, 该节点有 N_t 个样本，其中 k 类的样本点有 N_{tk} 个 $(k=1,\ldots,K)$，$H_t(T)$ 为叶节点 t 上的经验熵，$a \geq 0$ 为参数。定义损失函数

$$C_a(T) = \sum_{t=1}^{|T|} N_t H_t(T) + a|T|$$

其中经验熵为:

$$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

记:

$$C(T) = \sum_{i=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

有

$$C_a(T) = C(T) + a|T|$$

$C(T)$ 表示模型对训练数据的预测误差,$|T|$ 表示模型的复杂程度。剪枝，就是当 a 确定时，选择损失函数最小的模型。

下面给出剪枝算法的步骤。第一步计算每个节点的经验熵。第二步递归地从树的叶节点向上回缩，如果回缩后的损失函数小于等于回缩前，则剪枝，将父节点变为新的叶节点。第三步，回到第二步直到不能继续为止，得到损失函数最小的子树 T_a。

随机森林是由多个决策树组成的分类器，并且其输出的类别是由个别树输出的类别的众数而定，这里不再详细说明。