遗传算法的特点及其应用

上海复旦大学附属中学　张宁

目录

【关键词】
【摘要】
【正文】
§1 遗传算法的基本概念
§2 简单的遗传算法
1. 选择
2. 交换
3. 变异
§3 简单的遗传算法运算示例
1. 计算机公司的经营策略优化问题
2. 函数优化问题
§4 遗传算法应用举例
1. 子集和问题
2. TSP（旅行商）问题
§5 结束语
【附录】
1. 子集和问题源程序
2. TSP（旅行商）问题源程序
【参考文献】

【关键词】

遗传算法　遗传　变异　染色体　基因　群体

【摘要】

遗传算法是基于达尔文进化论，在计算机上模拟生命进化机制而发展起来的一门新学科。它根据适者生存，优胜劣汰等自然进化规则来进行搜索计算和问题求解。

文章的第一部分介绍了遗传算法的基本概念。第二部分介绍了遗传算法的原理以及三种运算：选择、交换、变异。第三部分着重介绍三种运算的具体实现，以及简单实例，主要体现遗传算法的实现过程。第四部分介绍了两个具体问题，都是属于 NP-完全问题，如何用遗传算法来解决，以及实现时的一些基本问题。

文章在介绍遗传算法的原理以及各种运算的同时，还分析了一些应用中出现的基本问题，对于我们的解题实践有一定的指导意义。

【正文】

遗传算法作为一门新兴学科，在信息学竞赛中还未普及，但由于遗传算法对许多用传统数学难以解决或明显失效的复杂问题，特别是优化问题，提供了一个行之有效的新途径，且能够较好地解决信息学竞赛中的 NP 难题，因此值得我们进行深入的讨论。

要掌握遗传算法的应用技巧，就要了解它的各方面的特点。首先，让我们来了解一下什么是遗传算法。

§1 遗传算法的基本概念

遗传算法（Genetic Algorithms，简称 GA）是人工智能的重要新分支，是

基于达尔文进化论，在计算机上模拟生命进化机制而发展起来的一门新学科。它根据适者生存，优胜劣汰等自然进化规则来进行搜索计算和问题求解。

对许多用传统数学难以解决或明显失效的复杂问题，特别是优化问题，GA 提供了一个行之有效的新途径，也为人工智能的研究带来了新的生机。

GA 由美国 J. H. Holland 博士 1975 年提出，当时并没有引起学术界的关注，因而发展比较缓慢。从 80 年代中期开始，随着人工智能的发展和计算机技术的进步，遗传算法逐步成熟，应用日渐增多，不仅应用于人工智能领域（如机器学习和神经网络），也开始在工业系统，如控制、机械、土木、电力工程中得到成功应用，显示出了诱人的前景。与此同时，GA 也得到了国际学术界的普遍肯定。

从 1985 年至今国际上已举行了五届遗传算法和进化计算会议，第一本《进化计算》杂志 1993 年在 MIT 创刊，1994 年 IEEE 神经网络汇刊出版了进化规划理论几应用专集，同年 IEEE 将神经网络，模糊系统，进化计算三个国际会议合并为’94IEEE 全球计算智能大会(WCCI)，会上发表进化计算方面的论文 255 篇，引起了国际学术界的广泛关注。

目前，GA 已在组合优化问题求解、自适应控制、程序自动生成、机器学习、神经网络训练、人工生命研究、经济组合等领域取得了令人著目的应用成果，GA 也成为当前人工智能及其应用的热门课题。

§2 简单的遗传算法

遗传算法（Genetic Algorithms，以下简称 GA）是基于自然选择，在计算机上模拟生物进化机制的寻优搜索算法。

在自然界的演化过程中，生物体通过遗传(传种接代，后代与夫辈非常相像)、变异（后代与夫辈又不完全相像）来适应外界环境，一代又一代地优胜劣汰，发展进化。

GA 则模拟了上述进化现象。它把搜索空间（欲求解问题的解空间）映射为遗传空间，即把每一个可能的解编码为一个向量（二进制或十进制数字串），称为一个染色体（chromosome，或个体），向量的每一个元素称为基因(genes)。所有染色体组成群体（population，或集团）。并按预定的目标函数（或某种评价指标，如商业经营中的利润、工程项目中的最小费用、最短路径等）对每个染色提进行评价，根据其结果给出一个适应度的值。

算法开始时先随机地产生一些染色体（欲求解问题的侯选解），计算其适应度，根据适应度对诸染色体进行选择、交换、变异等遗传操作，剔除适应度低（性能不佳）的染色体，留下适应度高（性能优良）的染色体，从而得到新的群体。

由于新群体的成员是上一代群体的优秀者，继承了上一代的优良性态，因而在总体上明显优于上一代。GA 就这样反复迭代，向着更优解的方向进化，直至满足某种预定的优化指标。上述 GA 的工作过程可用图 1简要描述。

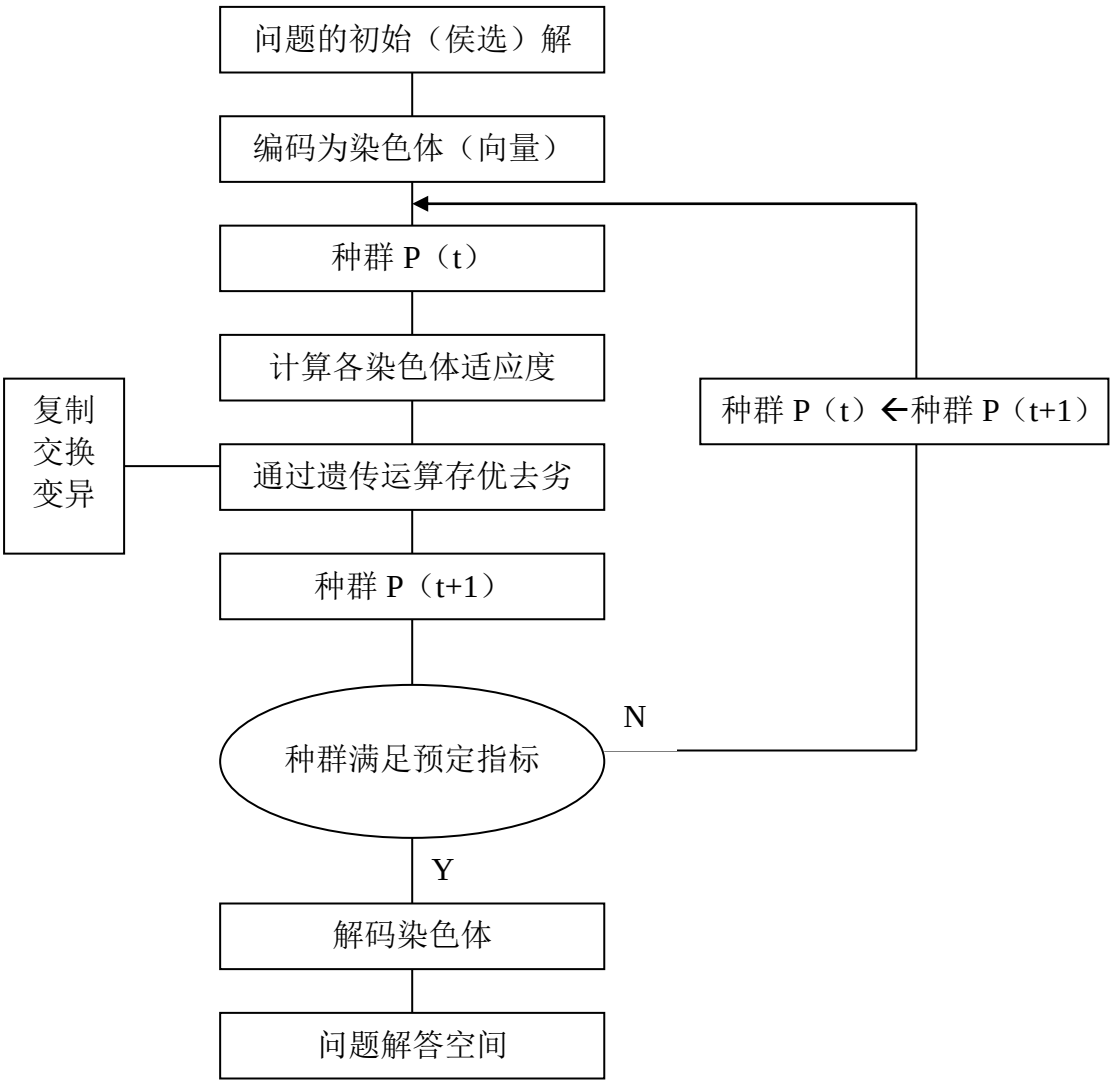


图 1 遗传算法工作原理示意图

简单遗传算法的三个基本运算是选择、交换、变异，下面详细介绍。

1. 选择

选择运算又称为繁殖、再生，或复制运算，用于模拟生物界去劣存优的自然选择现象。它从旧种群中选择出适应性强的某些染色体，放入匹配集（缓冲区），为染色体交换和变异运算产生新种群做准备。适应度越高的染色体被选择的可能性越大，其遗传基因在下一代群体中的分布就越广，其子孙在下一代出现的数量就越多。有多种选择方法，使用比较普遍的一种是适应度比例法，简述如下：

适应度比例法又称为轮转法，它把种群中所有染色体适应度的总和看作一个轮子的圆周，而每个染色体按其适应度在总和中所占的比例占据轮子的一个扇区。每次染色体的选择可看作轮子的一次随机转动，它转到哪个扇区停下来，那个扇区对应的染色体就被选中。其实就是将适应度值视为其权值，权值大的被选中的概率也大。尽管这种选择方法是随机的，但它与各染色体适应度成比例。某一染色体被选中的概率（选择概率）为

$$P_c = f(x_c) / \sum f(x_i)$$

式中 x_i 为种群中第 i 个染色体对应的数字串，$f(x_i)$ 是第 i 个染色体的适应度值，$\sum f(x_i)$ 是种群中所有染色体的适应度值之和。

用适应度比例法进行选择时，首先计算每个染色体的适应度，然后按比例于各染色体适应度的概率进入交换（匹配）集的染色体，其具体步骤如下：

（1）计算每个染色体的适应度值 $f(x_i)$；

（2）累加所有染色体的适应度值，得最终累加值 SUM=$\sum f(x_i)$，记录对应于每个染色体的中间累加值 $g(x_i)$；

（3）产生一个随机数 N，0<N<SUM；

（4）选择其对应的中间累加值满足 $g(x_{i-1})$<N<= $g(x_i)$ 的染色体进入交换集。

（5）重复（3），（4），直到交换集中包含足够多的染色体数字串为止。

重复上述过程，直到交换集中包含足够多的染色体为止。显然，此法要求染色体的适应度应为正值。请看下例：

[例 1]：某种群包含 10 个染色体，按适应度比例法选择进入交换集的过程示于表 1及表 2。

表 1

染色体编号	1	2	3	4	5	6	7	8	9	10
适应度	8	2	3	16	6	12	11	7	3	7
选择概率	0.11	0.03	0.04	0.21	0.08	0.16	0.15	0.09	0.04	0.09
适应度累加值	8	10	13	29	35	47	58	65	68	75

表 2

随机数	23	49	70	14	28	37	57
所选染色体号	4	7	10	4	4	6	7

由表 1,表 2可以看到，3 号染色体的选择概率最高，被选中的次数最多，其他选择概率较低的染色体被选中的次数就少。当然，用适应度比例法进行选择时，性能最坏的染色体也可能被选择，但概率极小，当种群长度较大时，这种情况可以忽略不计。

2. 交换

复制操作虽然能够从旧种群中选择出优秀者，但不能创造新的染色体，因此，遗传算法的开创者提出了交换操作。它模拟生物进化过程中的繁殖现象，优良品种，即：在匹配集中任选两个染色体（称为双亲的染色体）；随机选择一点或多点交换点位置 J（0<J<L，L 是染色体数字串的长度）；交换双亲染色体交换点右边的部分，即可得到两个新的（下一代）染色体数字串。也就是说，交换操作能够创造新的染色体（子孙染色体），从而允许测试在搜索空间中的新点。交换也体现了自然界中信息交换的思想。请看下例：

[例 2]：从匹配集中取出的一对染色体为：

染色体 A　　0111|1010

染色体 B　　0101|0010

随机产生的一点交换位置是 4，交换染色体 A，B 中第 4 位右边的部分——1010 和 0010，则得两个下一代（子孙）染色体数字串：

染色体 A’　　01111010

染色体 B’　　01010010

3. 变异

变异运算用来模拟生物在自然界的遗传环境中由于各种偶然因素引起的基因突变，它以很小概率随机地改变遗传基因（表示染色体的符号串的某一位）的值。在染色体以二进制编码的系统中，它随机地将染色体的某一个基因由 1 变成 0，或由 0 变成 1。若只有选择和交换，而没有变异操作，则无法在初始基因组合以外的空间进行搜索，使进化过程在早期就陷入局部解而终止进化过程，从而使解的质量受到很大限制。通过变异操作，可确保群体中遗传基因类型的多样性，以使搜索能在尽可能大的空间中进行，避免丢失在搜索中有用的遗传信息而陷入局部解，获得质量较高的优化解答。

§3 简单的遗传算法运算示例

让我们从下面两个非常简单的例子来具体了解一下简单遗传算法的各种操作。虽然下面的例子都有更好处理方式，但为了深入了解遗传算法的各种操作，还是从简单的例子入手：

[例 3]：计算机公司的经营策略优化问题

一个计算机公司追求的目标是最高的利润，为达此目标，必须选择适当的经营策略。一种可能的策略是对以下四个问题作出决策：

- 每台 PC 计算机的价格定为 5000 元（低价）还是 8000 元（高价）；
- 与 PC 机配套的免费软件是 Windows2000 还是 Linux；
- 是否提供网络技术服务；
- 提供保修期为半年或是一年；

下面用遗传算法来解决这个决策优化问题。

（1） 把问题的可能解表示为染色体数字串。因为有四个决策变量，而每个变量只有两种选择，其取值可用 0 或 1 来表示，于是，可用四位的二进制数表示一种可能的经营策略，解的搜索空间为 2^4=16，即共有 16 种经营策略可供选择，表 3表示出了其中计算机公司经理已知的 4 种经营策略（初始解答）。表中数字串的第一位取 0 表示高价，1 表示低价；第二位取 0 表示 Windows2000，1 表示配套 Linux；第三位取 0 表示不支持网络技术服务，1 表示支持网络技术服务；第四位取 0 表示保修期为一年，1 表示表示保修期为半年。

表 3 问题的初始解答

序号	价格	配套软件	网络服务	保修期	染色体数字串
1	高	Linux	支持	一年	0110
2	高	Windows2000	支持	半年	0011
3	低	Linux	不支持	一年	1100
4	高	Linux	不支持	半年	0101

（2） 求各染色体的适应度。在这个问题中，我们不妨设染色体的适应度就是染色体的二进制数字串的数值，对应经营策略的利润。

表 4 第 0 代种群的适应度

序号 i	染色体串 x_i	适应度 $f(x_i)$
1	0110	6
2	0011	3
3	1100	12
4	0101	5
适应度总和		26
最坏适应度		3
最好适应度		12
平均适应度		6.5

（3） 选择进入交换集的染色体

由表 4 可知，所有染色体串适应度的总和是 26，串 1100 的适应度是 12，占适应度总和的 6/13，也就是串 1100 被选中的机会接近两次；串 0110，0011，0101 被选中的概率分别是 3/13，3/26，5/26。按前述适应度比例法，选择进入交换集的染色体串及其适应度情况如表 5所示。

表 5 第 0 代种群选择后的适应度

序号 i	染色体串 x_i	适应度 $f(x_i)$
1	0110	6
2	1100	12
3	1100	12
4	0101	5
适应度总和		35
最坏适应度		5
最好适应度		12
平均适应度		8.75

由表 5可知，选择操作的作用是改进了种群的平均适应度，使其由原来的 6.5 提高到了 8.75，最坏适应度由原来的 3 改进为 5，性能最差的染色体已从种群中删除。但是，选择不能创造新的染色体，须进行交换操作。

（4） 交换操作

设采用单点交换，随机产生的交换点是 2，从交换集中任取一对染色体 1100 和 0101，互换它们的第 3，4 位，得子孙染色体串 1101 和 0100。由此得到下一代种群。如表 6所示。

表 6 第 1 代种群的适应度

序号 i	染色体串 x_i	适应度 $f(x_i)$
1	0110	6
2	1101	13

3	1100	12
4	0100	4
适应度总和		35
最坏适应度		4
最好适应度		13
平均适应度		8.75

（5） 估新一代的种群的适应度

由表 6 可知，在新一代（第 1 代）种群中，最优染色体适应度由原来 12 提高到了 13，其对应的二进制串是 1101，表示一种优化的经营策略。平均适应度由原来的 6.5 提高到 8.75，整个种群的适应度从总体上提高了，被优化了。

遗传算法重复地执行每一代的运算操作，产生新的一代，直到满足某些终止标准或条件。在本例中，假定已知染色体数字串 1111 代表的是已知的最好利润 15%，则遗传算法停止运行。

[例 4]：函数优化问题

设有函数 $f(x)=2x^2-5x+4$（$x\in Z$，$x\in[0，63]$），求其在区间内的最大值。下面用遗传算法求解这一问题。

（1） 确定适当的编码，把问题的可能解表示为染色体数字串。因为有一个决策变量 x，其取值范围为[0，63]，$2^6=64$，使用 6 位无符号二进制数组成染色体数字串，即可表达变量 x，以及问题的解答方案。

（2） 选择初始种群。通过随机的方法产生由 4 个染色体数字串组成的初始种群，见表 7。

（3） 计算适应度值及选择概率。此问题中染色体的适应度取函数自身 $f(x)=2x^2-5x+4$。将每个染色体的 f(x)计算出来，作为适应度值。在此基础上计算选择概率 $P_s=f_i/\sum f$。

表 7 函数优化过程

编号	染色体	X 值	适应度（目标函数）$f(x)=2x^2-5x+4$	选择概率 $f_i/\sum f$	实选染色体编号
1	011010	26	1226	0.15	1
2	100100	36	2416	0.29	1
3	010101	21	781	0.09	0
4	101110	46	4006	0.48	2
和			8429	1	4
平均			2107.25	0.25	1
最大			4006	0.49	2

（4） 选择进入交换集的染色体。按适应度比例法，选择进入交换集的染色体串，如表 8。可见，染色提 1 和 2 都被选择了 1 次；染色体 4 被选择了 2 次；染色体 3 没有被选择。所选的 4 个染色体被送到交换集，准备参加交换。

（5） 交换染色体。首先对进入交换集的染色体进行随机配对，此例中是染色体 1 和 3 配对，染色体 2 和 4 配对。接着随机确定交换位置，结果是第 1 对

染色体的交换位置是 1，第 2 对染色体的交换位置是 4。经过交换操作后得到的新种群如表 8所示。

（6） 变异。若此例中变异概率取 0.01。由于种群中 4 个染色体总共只有 24 位代码，变异的期望值为 24*0.01=0.24 位，可以忽略不计。

表 8 函数优化过程

复制后交换集种群	交换配对	交换位置	新染色体	X 值	适应度（目标函数）$f(x)=2x^2-5x+4$
011010	3	3	001110	14	326
100100	4	4	100110	38	2702
101110	1	3	111010	58	6442
101110	2	4	101100	44	3656
和					13126
平均					3281.5
最大					6442

从表 8可以看出，虽然仅进行了一代遗传操作，但种群适应度的平均值及最大值却比初始群有了很大提高，平均值由 2107.25 变到 3281.5，最大值由 4006 变到 6442。这说明随着遗传运算的进行，种群正向着优化的方向发展。

§4 遗传算法应用举例

1.[例 5]：GA 在子集和问题上的应用

子集和问题 SUBSET_SUM：给定正整数集合 S 和一个整数 t，判定是否存在 S 的一个子集 $S'\subseteq S$ 使得 S’中整数的和为 t。

例如，若 S={1，4，16，64，256，1040，1041，1093，1284，1344}且 t=3754，则子集 S’={1，16，64，256，1040，1093，1284}是一个解。

设子集和问题的一个实例为<S，t>。其中，S 是一个正整数的集合{x_1，x_2，…，x_n}，t 是一个正整数。子集和问题要求判定是否存在 S 的一个子集 S’，使得 $\sum_{x_i\in S'} x_i = t$。我们已知道该问题是一个 NP-完全问题。在实际应用中，我们常遇到的是最优化子集和问题。在这种情况下，我们要找出 S 的一个子集 S’，使得其和不超过 t，但又尽可能接近于 t。例如，我们有一辆载重车，其载重量不能超过 t 公斤。有 n 个不同的箱子要用载重车来装运，其中第 i 个箱子重 x_i 公斤。我们希望在不超过载重限制的前提下将载重车尽可能地装满。这个问题实质上就是一个最优化形式的子集和问题。

下面用遗传算法来解决：

若集合 S 中元素的个数为 n，每个元素只有两种可能：属于 S’或不属于 S’。因此我们可以用 n 为二进制数来表示每个染色体。每一位，若为 0 则说明这个元素不属于 S’，若为 1 则说明这个元素属于 S’。

确定了问题的描述方式，我们只需随机取出染色体组成初始群体。接着，便可以按前所述，进行选择、交换以及变异运算。

我们将染色体所表示的子集的元素和与所给 t 的差异记为适应度。

即令染色体 x 的每一位为 x_i，所表示元素的值为 S_i 则

$$f(x)=\sum_{x_i\neq 0}S_i-t$$

但是经过实践后发现由于适应度相对差异较小，使得适应度非常接近，难以区分优秀的染色体，使得遗传进化变得非常缓慢，且 f(x)可能为负值，因此还需对适应度函数做一下变换，才可以适合本题的要求。

令|f(k)|为当前群体中所有染色体适应度的最大值

f’(x)=|f(k)-f(x)|

所以适应度为 f’(x)。

选择时可以用前面所介绍的适应度比例法，但采用随机方式，可能会出现随机错误，使得优秀的染色体没有子孙。因此，在这里我们对选择方法作一下改进，采用确定性选择法，使得选择不依靠随机的好坏，先计算群体中每个串的生存概率 $P_s=f_i/\sum f_j$，1<j<n，然后计算期望复制数 $e_i=P_s*n$，式中：n 为群体中染色体的数目。根据 e_i 值的整数部分给每个染色体串分配一个复制数，而按 e_i 的小数部分对群体中的染色体串排序，最后按排列的大小顺序选择要保留到下一代的染色体串（子孙）。

接着可以进行交换运算，交换运算与前述相同，不过若进行单点交换有可能使得两个染色体在交换时产生的差异过大，使得遗传变得不稳定，优秀的染色体不能遗传到下一代。因此可以采用多点交换，不过本题中只需进行两点交换即可，其实两点交换与单点交换是类似的。

设有两个父辈染色体 A 和 B：

A：10100100101110

B：01001110110001

设两个交换点选择如下：

A：10100|10010|1110

B：01001|11011|0001

则两点交换运算就是交换染色体 A 和染色体 B，两个交换点之间的部分，则交换结果如下：

A’：10100110111110

B’：01001100100001

交换运算完成后，可进行变异运算，变异运算时，只需注意变异概率的取值，至于具体算法如前面所述。

在本题中的一些数值不妨取值如下：

种群长度（染色体个数）：20

选择概率：0.9

变异概率：0.1

结束条件：当前最优解在 100 代遗传后仍未改变，或已取到最优解

2.[例 6]：GA 在 TSP（旅行商）问题求解中的应用

设存在 N 个城市， D_{ij} 表示城 i 与城 j 之间的距离， $D_{ij}=D_{ji}$，现在要求一条遍历所有 N 个城市，且不走重复路的最短路径（最短哈密尔顿圈）。

这是一个典型的排列顺序问题，属于 NP-完全问题。传统解法诸如：限界剪枝法，或是最小支撑树算法，对此都并不太奏效（或是运算速度上的缺陷，或是解的质量上的缺陷）。下面我们试着用遗传算法来解决这道题目。

我们先采用十进制编码，每个染色体由按一定顺序排列的 N 个城市的序号组成，表示一条可能的旅行路径，其中每个基因表示一个城市。染色体的长度为 N，解空间为 N!。适应度为一条旅行路径对应的距离，路径越短的染色体适应度越高。例如，取 N=10，城市代号为 1，2，3，4，5，6，7，8，9，10。

则种群中的染色体：2 8 4 10 5 1 7 3 6 9；

表示一条旅行路径：2→8→4→10→5→1→7→3→6→9→2；

其总路径长 $\sum D_{ij} = D_{28} + D_{84} + D_{410} + D_{105} + D_{51} + D_{17} + D_{73} + D_{36} + D_{69} + D_{92}$

我们可以采用非负变换，把最小化优化目标函数变换为以最大值为目标的适应度函数，可以如下定义：

$$f(x) = c_{\max} - \sum D_{ij}$$

其中 c_{max} 为可以取为进化过程中路径长度的最大值，或者为了保证 f(x)为正而预先设定为一个与种群无关的常数。

下面介绍具体算法：

首先我们由城市数目 N，可以确定染色体长度为 N，染色体的基因代码由城市编码组成。但是，我们仍须确定种群长度、变异概率等参数。这些参数都由人为确定，与算法无关，可暂时不考虑，讨论完算法后，可再行确定具体数值。

随机组合成染色体，每个染色体须表示一条哈密尔顿回路，生成初始种群。然后分别计算每一个染色体的适应度，按适应度比例法选择染色体到交换集。不过，我们也可以考虑一下改进后的选择方法。

适应度比例法实现简单，缺点是选择过程中最好的染色体可能在下一代产生不了子孙，可能发生随机错误。因此我们可以采用择优选择法，即对于群体中最优秀（适应度最高）的染色体不进行交换和变异运算，而是直接把它们复制到下一代。以免交换和变异运算破坏种群中的优秀解答。这种方法可加快局部搜索速度，但又可能因群体中优秀染色体的急剧增加而导致搜索陷入局部最优解。同时，我们也可以采用确定性选择法，使得选择不依靠随机的好坏，先计算群体中每个串的生存概率 $P_s = f_i / \sum f_j$ ，1<j<n，然后计算期望复制数 $e_i=P_s*n$，式中：n 为群体中染色体的树木。根据 e_i 值的整数部分给每个染色体串分配一个复制数，而按 e_i 的小数部分对群体中的染色体串排序，最后按排列的大小顺序选择要保留到下一代的染色体串（子孙）。

完成选择运算后，可以进行交换运算。但是，此处的交换运算不同于前，须改进原有的交换运算，具体做法如下：

因为两个染色体，若进行简单的交换运算，可能会使得染色体所表示路径中会重复经过同一城市，即同一染色体中的两个基因有着相同的城市编号。这就造成了染色体不再表示一条哈密尔顿回路，因此须改进交换运算。

我们可以采用部分匹配交换运算（PMX）。它先用随机均匀分布方法在欲交换两父染色体串中各产生两个交换点，把这两点之间的区域定义为匹配区域，再对两个匹配区域中的基因通过对应匹配置换。例如，两父染色体串为：

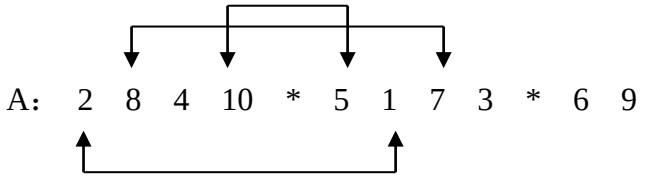
A：　2　8　4　10　*　5　1　7　3　*　6　9

B：　5　6　7　1　*　10　2　8　3　*　9　4

符号“*”表示交换点。

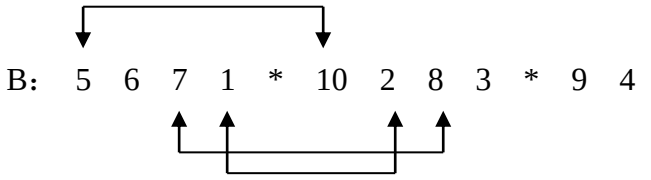
对染色体串 A，其匹配区域中有四个元素 5，1，7，3，应与染色体串 B 匹配

区域中的四个元素 10，2，8，3，逐一匹配，即通过在染色体串 A 内进行元素间的置换，使得匹配区域内这四个元素换为 10，2，8，3。为此，染色体串 A 中的元素作如下置换：



A’： 1 7 4 5 * 10 2 8 3 * 6 9

对染色体串 B，其匹配区域中有四个元素 10，2，8，3，应与染色体串 A 匹配区域中的四个元素 5，1，7，3，逐一匹配，即通过在染色体串 B 内进行元素间的置换，使得匹配区域内这四个元素换为 5，1，7，3。为此，染色体串 B 中的元素作如下置换：



B’： 10 6 8 2 * 5 1 7 3 * 9 4

当然，在本题中最简单的做法是不进行交换运算，而直接进行变异运算，因为在变异运算中本就隐含了交换的含义。

对于群体中某些染色体进行变异运算，由于基因采用的是城市编码，变异操作与二进制编码时不同，从群体中随机抽取一个染色体，随机抽取两个基因，将两者交换，即颠倒城市次序，用以完成变异运算。

当然我们还需确定一下终止条件，因为目前使用严格的数学方法判定遗传算法的终止（收敛）条件还比较困难。因此，在实现时采用的主要还是启发式方法，我们可以以连续几次迭代后得到的解群中最优解是否变化来作为终止时的判定条件。

算法的主要部分已经讨论完了，但是还有一点值得提出的，由于遗传算法是一种不断优化的搜索算法，因此给出的初始群体的优劣程度会影响到最终解答的优劣程度及其出解速度。因此，我们可以用贪心算法构造初始群，或当 TSP 问题具有三角不等式性质时，我们可以考虑先使用速度较快的最小支撑树算法的解答来构造初始群。

我们再考虑用最小支撑树算法来构造初始群是有利有弊，由于有了高质量的初始群使得遗传算法能在较短的时间内收敛，且得到的解有质量上的保证，稳定性也高。但是由于初始群的优秀，对于后代的影响很大，容易造成过早收敛于局部最优解，因此在使用时，需按情况而定，在解的要求、时间上的要求综合考虑下作出决定。这些，是我在解决 TSP 问题时，总结的一些经验，希望能对大家在解决问题的过程中有一定的指导意义。

题目中的一些数值不妨取值如下：

种群长度（染色体个数）：20

变异概率：0.1

通过两道例题，我们在遗传算法原有简单定义上，加以扩充，介绍了若干高级遗传算法在具体实例中的应用，旨在打开思想的局限性，不仅仅在原有简单定义下做文章，而是充分发挥想象，对于不同问题，采取不同对策，在遗传算法的框架下，安排使用合理的算法，改进原有算法，或与原有算法相结合。这样，才能充分发挥遗传算法的长处解决问题。

§5 结束语

遗传算法的原理是简单的，但是如何熟练运用遗传算法却并不是一个简单的问题，理论要结合实际，对于不同问题，遗传算法要稍加变化，就如同画龙点睛一般，切忌不可生搬硬套。我认为遗传算法应当与现有优化算法结合，可产生比单独使用遗传算法或现有优化算法更好，更实用的算法。

本文的主要目的还是让大家对遗传算法能够有一个初步的了解，这样对大家进行深入的实践是有帮助的。希望大家通过本文的指导，能够将遗传算法熟练应用在各个方面。

【附录】

1. 例 5 子集和问题源程序：

```
program subset_sum_GA;
const inp='subset.in';{输入文件}
      out='subset.out';{输出文件}
      maxn=1000;{集合最大元素个数}
      pl=20;{群体中染色体个数}
      pc=10;{交换次数}
      pv=0.1;{变异概率}
      ec=100;{结束条件}
type ch=array[1..maxn] of 0..1;
var sset:array[1..maxn] of integer;{集合}
    pop,pop1:array[1..pl] of ch;{群体}
    result:ch;{结果}
    n:integer;{集合元素个数}
    t{子集和},sum{集合元素和},diff{当前最优解的差距}:longint;
  procedure input;{输入}
  var fin:text;
      i:integer;
  begin
    assign(fin,inp);
    reset(fin);
    readln(fin,n,t);
    for i:=1 to n do
      readln(fin,sset[i]);
```

```
  close(fin);
end;
procedure init;{初始化}
var i,j,k:integer;
begin
  randomize;
  sum:=0;
  for i:=1 to n do
    inc(sum,sset[i]);
  if sum<=t then
  begin
    for i:=1 to n do
      result[i]:=1;
    exit;
  end;
  k:=trunc(sum/t)+1;
  for i:=1 to pl do
  begin
    for j:=1 to n do
      if random(k)=0 then
        pop[i,j]:=1
      else
        pop[i,j]:=0;
  end;
  diff:=t;
end;
procedure solve;{遗传算法}
var f1:array[1..pl] of longint;{适应度}
    dc,pr:integer;
    work:boolean;
  procedure select;{选择运算}
  var f:array[1..pl] of longint;{适应度}
      g:array[1..pl] of integer;{复制数}
      fs,fm:longint;
      i,j,k:integer;
  begin
    fm:=0;
    for i:=1 to pl do
    begin{计算适应度}
      f[i]:=0;
      for j:=1 to n do
        inc(f[i],sset[j]*pop[i,j]);
      f[i]:=f[i]-t;
      if (f[i]<0) and (abs(f[i])<diff) then
      begin
        diff:=abs(f[i]);
        dc:=0;
        result:=pop[i];
      end;
      if f[i]=0 then
      begin
```

```
        work:=false;
        exit;
      end;
      if abs(f[i])<abs(fm) then
        fs:=f[i];
    end;
    inc(dc);
    for i:=1 to pl do{适应度作变换}
      if f[i]<0 then
        f[i]:=fm+f[i]
      else
        f[i]:=fm-f[i];
    fs:=0;
    for i:=1 to pl do{计算适应度和}
      inc(fs,abs(f[i]));
    k:=0;
    for i:=1 to pl do
    begin{分配复制数}
      g[i]:=trunc(abs(f[i])/fs*pl);
      inc(k,g[i]);
    end;
    while k<pl do
    begin
      inc(g[random(pl)+1]);
      inc(k);
    end;
    k:=0;
    for i:=1 to pl do{复制到交换集}
      for j:=1 to g[i] do
      begin
        inc(k);
        pop1[k]:=pop[i];
        f1[k]:=f[i];
      end;
    pop:=pop1;
  end;
  procedure cross;{交换运算}
  var i,j,k,u,v,p,q:integer;
  begin
    for i:=1 to pc do
    begin
      u:=random(pl)+1;{染色体 A}
      v:=random(pl)+1;{染色体 B}
      p:=random(n)+1;{交换点 1}
      q:=random(n)+1;{交换点 2}
      if p>q then
      begin
        k:=p;
        p:=q;
        q:=k;
      end;
```

```
      k:=0;
      for j:=p to q do
        inc(k,pop[u,j]-pop[v,j]);{计算交换部分差距}
      if not (((f1[u]<0) and (f1[v]>0) and (k>0)) or
             ((f1[u]>0) and (f1[v]<0) and (k<0))) then{若差异变大则不交换}
        continue;
      dec(f1[u],k);
      inc(f1[v],k);
      for j:=p to q do
      begin{交换}
        k:=pop[u,j];
        pop[u,j]:=pop[v,j];
        pop[v,j]:=k;
      end;
    end;
  end;
  procedure variety;{变异运算}
  var i,j,k:integer;
  begin
    for i:=1 to pr do
    begin
      j:=random(pl)+1;{染色体}
      k:=random(n)+1;{基因}
      if ((f1[j]<0) and (pop[j,k]=0)) or {变异}
        ((f1[j]>0) and (pop[j,k]=1)) then
      begin
        inc(f1[j],(1-2*pop[j,k])*sset[k]);
        pop[j,k]:=1-pop[j,k];
      end;
    end;
  end;
begin
  if sum<=t then
    exit;
  work:=true;
  pr:=round(n*pl*pv);{计算变异次数}
  dc:=0;
  while work do
  begin
    select;{选择}
    if dc>ec then
      exit;{是否满足结束条件}
    if work then
    begin
      cross;{交换}
      variety;{变异}
    end;
  end;
end;
procedure output;{输出}
var fout:text;
```

```
      i,k:integer;
      s:longint;
  begin
    assign(fout,out);
    rewrite(fout);
    s:=0;
    k:=0;
    for i:=1 to maxn do
      if result[i]=1 then
      begin
        inc(s,sset[i]);
        inc(k);
      end;
    writeln(fout,s);
    writeln(fout,k);
    for i:=1 to maxn do
      if result[i]=1 then
        writeln(fout,i);
    close(fout);
  end;
begin{主程序}
  input;
  init;
  solve;
  output;
end.
```

说明：

在子集和问题中，随机构造的数据基本上都能达到最优解，偶尔也会收敛到准最优解。总体情况非常良好。

2. 例 6 TSP 问题源程序

```
program TSP_GA;
const inp='tsp.in';{输入文件}
      out='tsp.out'; {输出文件}
      maxn=100;{城市最大值}
      pl=100;{染色体个数}
      pv=0.1;{遗传概率}
      ec=100;{结束条件}
      no=10000;{无通路}
type ch=array[1..maxn] of integer;
var d:array[1..maxn,1..maxn] of integer;{邻接矩阵}
    pop,pop1:array[1..pl] of ch;{染色体}
    result:ch;{结果}
    n:integer;{城市个数}
    min:longint;{当前最短路径长度}
  procedure input;{输入}
  var i,j:integer;
```

```
    fin:text;
begin
  assign(fin,inp);
  reset(fin);
  readln(fin,n);
  for i:=1 to n do
    for j:=1 to n do
    begin
      read(fin,d[i,j]);
      if d[i,j]=-1 then
        d[i,j]:=no;
    end;
  close(fin);
end;
procedure init;{初始化}
var temp:ch;
    i:integer;
  procedure greedy;{贪心}
  var use:array[1..maxn] of boolean;
      i,j,k,min,r:integer;
  begin
    r:=random(n)+1;
    fillchar(use,sizeof(use),true);
    temp[1]:=r;
    use[r]:=false;
    for i:=2 to n do
    begin
      min:=no+1;
      k:=0;
      for j:=1 to n do
        if use[j] then
          if d[r,j]<min then
          begin
            min:=d[r,j];
            k:=j;
          end;
      temp[i]:=k;
      use[k]:=false;
      r:=k;
    end;
  end;
begin{构造初始群}
  randomize;
  for i:=1 to pl do
  begin
    greedy;
    pop[i]:=temp;
  end;
  min:=no;
end;
procedure solve;{遗传算法}
```

```
var f1:array[1..pl] of longint;
    dc,pr:integer;
    work:boolean;
  function suit(a:ch):longint;{适应度}
  var p:longint;
      i:integer;
  begin
    p:=0;
    for i:=1 to n-1 do
      inc(p,d[a[i],a[i+1]]);
    inc(p,d[a[n],a[1]]);
    suit:=p;
  end;
  procedure select;{选择}
  var f:array[1..pl] of longint;
      g:array[1..pl] of integer;
      fs,fm:longint;
      i,j,k:integer;
  begin
    fm:=0;
    for i:=1 to pl do
    begin
      f[i]:=suit(pop[i]);
      if f[i]<min then
      begin
        min:=f[i];
        result:=pop[i];
        dc:=0;
      end;
      if f[i]>fm then
        fm:=f[i];
    end;
    inc(dc);
    fs:=0;
    for i:=1 to pl do
      inc(fs,fm+1-f[i]);
    for i:=1 to pl do
      g[i]:=trunc((fm+1-f[i])/fs*pl);
    k:=0;
    for i:=1 to pl do
      for j:=1 to g[i] do
      begin
        inc(k);
        pop1[k]:=pop[i];
        f1[k]:=f[i];
      end;
    for i:=k+1 to pl do
    begin
      pop1[i]:=result;
      f1[i]:=min;
    end;
```

```
      pop:=pop1;
    end;
    procedure variety;{变异}
    var i,j,k,l,p:integer;
        t:longint;
        temp:ch;
    begin
      for i:=1 to pr do
      begin
        j:=random(pl)+1;
        k:=random(n)+1;
        l:=random(n)+1;
        temp:=pop[j];
        p:=temp[k];
        temp[k]:=temp[l];
        temp[l]:=p;
        t:=suit(temp);
        if t<f1[j] then
        begin
          pop[j]:=temp;
          f1[j]:=t;
        end;
      end;
    end;
  begin
    work:=true;
    pr:=round(n*pl*pv);
    dc:=0;
    while true do
    begin
      select;
      if dc>ec then
        exit;
      if work then
        variety;
    end;
  end;
  procedure output;{输出}
  var i:integer;
      fout:text;
  begin
    assign(fout,out);
    rewrite(fout);
    for i:=1 to n do
      writeln(fout,result[i]);
    close(fout);
  end;
begin{主程序}
  input;
  init;
  solve;
```

```
    output;
end.
```

说明：

由于 TSP 问题本身就是一个 NP 问题。对于随机构造的数据，无从知道最优解，而且由于随机性，最优解与局部最优解的差距不会非常明显，虽然差距不明显，但是可能路线截然不同。因此，可能会收敛到局部最优解。而人为构造的数据，最优解有较明显的区别时，一般可以收敛到最优解。

【参考文献】

1．邵军力　张　景　魏长华　《人工智能基础》　电子工业出版社
2．傅清祥　王晓东　《算法与数据结构》　电子工业出版社
3．吴文虎　王建德　《实用算法的分析与程序设计》　电子工业出版社