### 项目背景

**⑴** 遗传算法简介

遗传算法(Genetic Algorithm, GA)由于其在解决大空间、非线性、全局寻优等复杂问题时具有传统方法所不具备的独特优越性，已在越来越多的领域得到广泛的应用。近年来，遗传算法在电磁场与天线设计中也得到了较多的应用，成功地实现了阵列天线稀疏，降低最大旁瓣电平，或通过控制阵元位置、激励电流相位与幅度进行方向图零点生成。

遗传算法是建立在遗传学与自然选择基础上的自适应搜索过程，作为解决复杂问题的一种有效手段，遗传算法是目前人工智能和系统优化领域的热点研究课题。遗传算法从一初始化的群体出发，选取性能优良的个体，在这些优良个体之间进行生物化的繁殖，使它们一代代地进化到解空间中越来越好的区域。已经经过严格的数学证明，只要搜索时间足够长，并保留每一代的最优解，最终总能达到全局最优解。与其他优化方法相比，遗传算法以单一字符串的形式描述所研究的问题，只需利用适应度函数值进行优化计算，而不需要函数导数等其他辅助信息，特别适合解决其他科学技术无法解决或难以解决的复杂问题，是继专家系统、人工神经网络之后又一受人青睐的人工智能新学科分支。

遗传算法的主要特点有：

① GA使用所求解问题的参数集合的编码形式，而不是使用问题的参数本身，其操作是在给定字符串上进行的；

② GA只需要一个适应性函数（性能指标），而不是需要导数或其他辅助信息，也不要求目标函数的连续性，因而具有广泛的适应性；

③ GA不是从一点开始，而是从问题的解空间中的多点开始搜索问题的解，因而能够快速全局收敛；

④ GA使用概率规则指导搜索而不是确定性规则，因此能搜索离散有噪声的多峰值复杂空间；

⑤ GA在解空间内进行充分的搜索，但并不是盲目地穷举或瞎碰，评价为选择提供了依据，因此其搜索时耗和效率往往优于其他优化算法。

⑵ 遗传算法的基本步骤

遗传算法的一般流程如图1所示，算法的基本步骤如下：

① 随机产生初始种群，个体数目一定，每个个体表示为染色体的基因编码；

② 计算个体的适应度，并判断是否符合优化准则，若符合，输出最佳个体及其代表的最优解，并结束计算，否则转向第3步；

③ 依据适应度选择再生个体，适应度高的个体被选中的概率高，适应度低的个体可能被淘汰；

④ 按照一定的交叉概率和交叉方法，生成新的个体；

开始

适应度检测

适应度检测

配对

繁殖

基因突变

染色体初始化

结束

N

N

Y

Y

图1. 遗传算法的基本步骤

⑤ 按照一定的变异概率和变异方法，生成新的个体；

⑥ 由交叉和变异产生新一代的种群，返回到第2步。

### 2. 基站天线方向图综合的改进型实数编码遗传算法

遗传算法原理和方法的特点决定了在优化基站天线的方向图指标时，有得天独厚的优势。主要有以下三个方面：

① GA中的优化以及产生染色体基因的过程使用的是概率，它可以极大的减小对一般优化方法对初始值的依赖，同时减少人为的干扰。

② GA使用适应度函数来评估排序染色体基因，适应度函数可以很方便的表达为基站天线需要优化的指标（第一上旁瓣、0-30°内上旁瓣、方向性系数）的函数，方便优化的同时，又不需要其他如导数的信息。

③ GA使用染色体基因交叉变异，交叉保证好的基因能够遗传下去，变异能保证不会陷入局部最优。一般算法在优化基站天线这种多目标、多参数的情形时，容易陷入局部最优。例如10单元天线，优化相位来调节第一上旁瓣，有10个自由度的变量，一般优化算法容易出现一个单元减少相位，上旁瓣变好得很多，这时再去优化其他单元时，如果其他的和它配合都会变差，那么就不会改变这个单元。而遗传算法由于变异的存在就不会陷入这种情况。

现在使用最多的天线仿真软件HFSS中，有5种优化算法，分别是Sequential Nonlinear Programming、Sequential Mixed Integer Nonlinear Programming、Quasi Newton、Pattern Search、Genetic Algorithm，其中仅有GA有Setup如图2所示，由此可见遗传算法的重要性以及可扩展性，其他大部分算法都没有这个优点，以至于不能添加自己在方向图调试优化的经验进去。

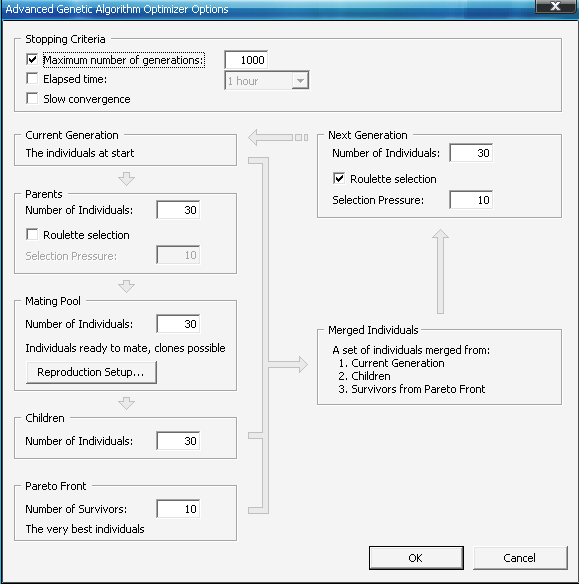


图2. HFSS中遗传算法的优化选项

对于连续型天线变量，二进制编码的求解精度受染色体长度的限制，同时在编程过程中需要不停的编码和解码，计算量巨大。而实数编码遗传算法物理意义明确，对于连续型变量的优化问题，采用实数编码方式具有较好的计算效率。

改进算法在遗传算法模型中采用了向下、向上外推和非一致杂交算子等新的交叉技术，并结合内插、交换等多种方式形成综合交叉技术。由于有效扩大了寻优空间，算法克服了已有的早熟缺陷，同时减少了算法的收敛性对初始种群体的依耐性。

由于在天线方向图综合中需要优化天线各单元的激励以取得最优的方向图，所以在遗传算法模型中使用染色体表示天线各单元的激励值，使用遗传算法对激励值不断优化，最终得到所需的解。

设有M个染色体，每个染色体有N个基因，均采用实数编码，取值范围[x1, x2]。首先确定进行交叉操作的父代，一般可采取轮盘赌或者排序的方法进行选择。综合采用以下4种繁殖方式生成子代：

① 内插：设经过选择的父代为V1、V2，采用随机内插的方法进行交叉，生成的子代为：

(1)

(2)

其中， j表示遗传的代数。这种交叉算法具有很好的收敛性。但由于交叉产生的染色体的基因值都在两个父代染色体对应基因值之间，无法拓展新的搜索空间，所以容易导致算法的早熟。

② 外推：针对前一种交叉算法的早熟问题，同时采取基因值的线性外推算法，设

(3)

(4)

若向上外推的计算值或向下外推的计算值超过基因值的有效取值范围则使用以下的外推法：

(5)

(6)

③ 基因交叉：在染色体中随机的选择一点或多点进行基因交叉，设一个染色体由10个基因组成，如图3所示，随机选择一个交叉位置进行基因操作。

A1

A2

A3

A4

A5

A6

A7

A8

A9

A10

B1

B2

B3

B4

B5

B6

B7

B8

B9

B10

图3. 染色体交叉方案

④ 基因变异：按一定概率在染色体中随机的选择一点或多点进行变异。

A1

A2

A3

A4

A5

A6

A7

A8

A9

A10

A1

A2

A3

A4

A5

A6

B7

B8

B9

B10

图4. 染色体变异方案

算法核心的遗传步骤归纳如下：

① 对于各染色体的适应度进行排序，选择适应度高的前一半染色体进行配对繁殖；

② 将适应度最高的染色体直接复制到下一代，保证最佳染色体在遗传过程中不丢失；

③ 用排序法选择父代染色体用来繁殖；

④ 子代染色体的一半通过父代染色体的基因交叉产生；

⑤ 子代染色体的另一半通过父代染色体基因使用非一致杂交算子内插和外推产生，通过调整两者的相对比例可以有效的控制早熟，改善算法性能。

⑥ 在子代染色体中以较小的概率随机选择基因进行变异操作，在这一操作中必须保证最佳染色体在变异中不被丢失。

### 3. 遗传算法适应度函数讨论

基站天线方向图的综合和优化是一个多目标、多参数的非线性优化问题。由于信号、干扰和信道环境都是时变的，所以一般而言没有对方向图的先验知识可以使用。在这种情况下，一般都采用方向图的性能指标作为优化目的来设计目标函数。

具体到基站天线的方向图指标中，有3个重要的指标需要加入适应度函数中来，分别是第一上旁瓣SLL1，方向性系数DirCoe，0-30°内上旁瓣SLL0\_30。我们设计的目标是通过优化天线单元的幅相，来使得方向图中的旁瓣符合设计要求，同时不能使得方向性系数下降太多。因此旁瓣作为硬性指标，而方向性系数是作为参考指标，而且方向性系数是无单位的，而旁瓣是以dB为单位的。因此每下降1dB的旁瓣和下降1单位的方向性系数是不等价的，因此权重因子的取值将直接影响优化结果的好坏。

正由于适应度函数很简洁、方便，但是它也有一些固有缺陷：过于自由。具体到基站天线的优化中来，就是它有可能将一个10°天线方向图优化成0°天线方向图。因为0°天线方向图的旁瓣一般要优于10°，而且它也不符合我们的设计目标，因此它是不可用的。为了避免这一情况的发生，就需要引入一些辅助方法。首先引入一个下倾角判断函数testilt来判断染色体是否合乎标准，我们的判断标准是染色体的下倾角不能超出原始染色体的下倾角1°的范围，这个值可以视乎情况进行改变。然后我们需要判断经过这些筛选以后的染色体数目是否小于初始种群数，如果是的情况下，还需要对当前代的种群数添加新的染色体。

### 4. 遗传算法的优化公式

天线阵的辐射方向图可以表示为：

(8)

是阵元的复加权系数矢量

(9)

其中是第i个天线的单元方向图。

是选取的一组角度。

是从零点开始阵列的间距列表。

(10)

优化公式的计算为：

幅度：

(11)

相位：

(12)

(13)

单元方向图：

(14)

阵列部分：

(15)

(16)

染色体的类声明如下面代码清单1所示：

代码清单1. 染色体的类声明

class **ChromClass**(object):

*'''*

*classdocs*

*'''*

def **\_\_init\_\_**(*self*, framedata, GeneNum, ElementMatrix, \*args, \*\*kargs):

*'''*

*Constructor, GeneNum is the number of Gene, ELementMatrix*

*is the V matrix to caclulate the pattern, args stored the*

*AmPh or other data, kargs is the Fitness function, which*

*can be modified by invoker.*

*'''*

*self*.GN = GeneNum #基因数、也可以看做单元数

*self*.EM = ElementMatrix #EM就是V矩阵

*self*.FD = framedata #存储了大的字典表

*self*.AmPh = deepcopy(args[0]) #使用深复制备份幅相表

*self*.boundary = args[1] #边界

*self*.FitnessFun = kargs[*'Fitness'*] #适应性函数

*self*.InitGeneFun = kargs[*'InitGene'*]#随机初始化基因的函数

*self*.AT = AntennaArrayTable(*self*.GN)#天线的阵列表

*self*.SN = len(*self*.AT) #移相器自由度

*self*.Gene = *self*.InitGeneFun(*self*.GN,

*self*.SN, *self*.boundary)

*self*.SetAmPh()

*self*.GetPat()

*self*.GetPatParam()

*self*.GetFitness()

相位Ph作为基因，是需要随机产生的，但是必须满足一种规律，即两两一组移向单元，单独移向单元以及不移向单元。这些单元的编码以及随机生成的算法，不应该作为染色体类本身的函数，它破坏了抽象，同时使得染色体类变得不通用。同样，适应度函数以及基因的操作函数中如果有可能变化，也应该能够从外部导入。这里为了适应不同单元数的优化，还必须引入一种静态的天线阵列表来辅助优化算法。

代码清单2. 天线阵列表

**def AntennaArrayTable(N):**

**"""**

**天线阵列表，给出了7-14单元，两两移相的下标。可以随时修改，暂时不考虑支持**

**一带一的天线阵列，因为很容易优化成等相差的情形。**

**"""**

**ArrayTable = {}**

**ArrayTable['6'] = [[0], [1], [4], [5]]**

**ArrayTable['7'] = [[0, 1], [2], [4], [5, 6]]**

**ArrayTable['8a'] = [[0, 1], [2, 3], [6, 7]]**

**ArrayTable['8b'] = [[0], [1, 2], [5, 6], [7]]**

**ArrayTable['9'] = [[0, 1], [2, 3], [6, 7], [8]]**

**ArrayTable['10'] = [[0, 1], [2, 3], [6, 7], [8, 9]]**

**ArrayTable['11'] = [[0, 1], [2, 3], [6, 7], [8, 9], [10]]**

**ArrayTable['12'] = [[0, 1], [2, 3], [6, 7], [8, 9], [10, 11]]**

**ArrayTable['13'] = [[0, 1], [2, 3], [4, 5], [8, 9], [10, 11], [12]]**

**ArrayTable['14'] = [[0, 1], [2, 3], [4, 5], [8, 9], [10, 11], [12, 13]]**

**if (N == 8):**

**return ArrayTable['8a']**

**elif N >= 6 and N <= 14:**

**return ArrayTable['%s' % N]**

**else:**

**return None**