“华为杯”第十五届中国研究生数学建模竞赛

学　　校	苏州大学
参赛队号	
队员姓名	1.李正阳
	2.王甫涵
	3.林炜

“华为杯”第十五届中国研究生
数学建模竞赛

题 目　　多无人机对组网雷达的协同干扰

摘　　要：

如何分配无人机对组网雷达进行协同干扰是如今电子对抗的重要问题。该问题的主要难点在于无人机复杂的运动轨迹所产生的庞大计算量。遗传算法是一种全局优化的概率算法，由于良好的兼容性在各个领域的都广泛应用，非常适合用于解决复杂问题。我们首先建立了组网雷达与目标虚拟航迹的数学模型，以及对无人机运动轨迹的优化。将其与遗传算法结合，通过遗传算法强大的全局搜索与优化能力得到最优解。

I 绪论

组网雷达是通过多个雷达协同观测和判断来识别最终目标的系统，具有很强的抗干扰能力。通过无人干扰机对雷达电磁波进行处理来对雷达进行距离欺骗是现在常用的形式。组网雷达会通过多部雷达进行“同源检验”来判断目标是否为一个合理目标点，因此需要在每个时刻对多部雷达同时进行干扰。考虑到无人机本身的物理机能，想要通过遍历整个过程来找到问题的最优解需要极其庞大的运算量。我们通过遗传算法（genetic algorithm）来对这个过程进行优化，在较短的时间内得到极优解。

遗传算法[1]是一种受达尔文生物进化论中“自然选择”启发演化而成的计算模型，从属于进化算法（evolutionary algorithm (EA)）。遗传算法按照生物进化的模式分成那么几个过程：初始、选择、遗传、终止。首先构建若干个基因片段，如 Fig.1 所示，默认所有原始族群性质一致。在计算适应度之后，可以分辨基因中的优秀片段（红色）与劣等片段（蓝色）。根据适应度选择合适的基因进行遗传，并且交叉优秀片段。在删除若干个劣质基因后通过变异重新补充基因。

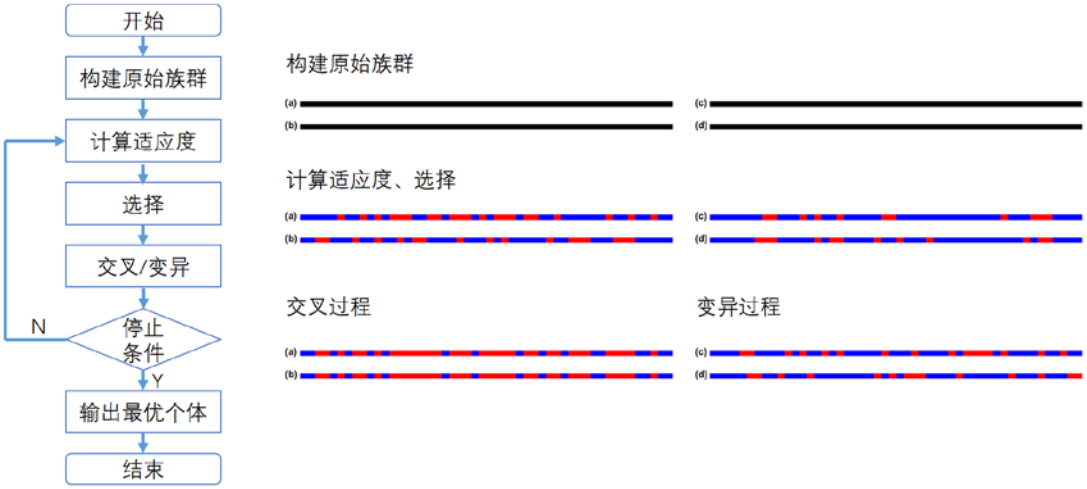

Fig.1 遗传算法示意图

虽然遗传算法源于生物体系，但只要建立合适的模型，它可作用与各个领域，通过强大的运算能力可以解决相当多的问题。通过遗传算法可以极大的优化系统，更加高效的计算出如何用更少的无人机干扰雷达虚构建出虚拟航迹。在 II 中我们根据本课题中组网雷达及无人机建立数学模型。III 和 IV 中分别介绍了两种无人机飞行限制条件结合遗传算法后的具体算法。V 中总结了我们的结论以及改进思路。

II 模型

在本文所考虑的体系中，组网雷达由 5 部雷达组成，且虚拟航迹同时处于 5 部雷达的辐射范围内。融合中心每隔 10s 获得一次雷达数据，并且进行“同源检验”，如若 5 台雷达中有三部雷达通过，则判定为合理的航迹点。现要构建一条虚拟航迹，需要至少 20 个连续的虚拟航迹点，可将此问题构建成数学模型。

在 Fig.1 中，任意时刻都有至少三部雷达被无人机干扰可以通过“同源检验”，产生一个合理的目标轨迹点。在连续的 20 个时刻都完成了上述条件则可构成一条欺骗干扰轨迹。这样便得到这问题最为基础的解决方案：使用 60（3*20）架无人机在特定的时刻飞过特定的干扰点便可达成目标。

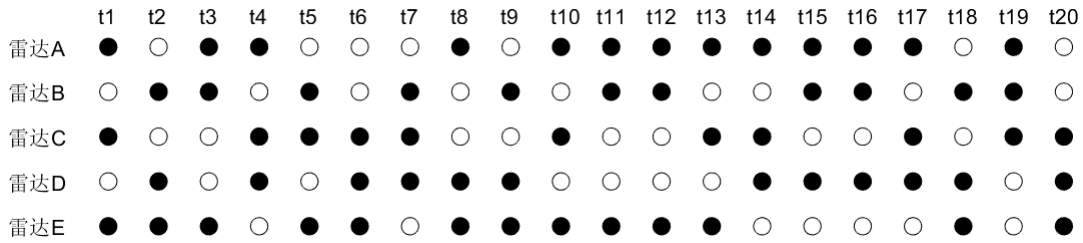

Fig.2 实心圆为被干扰的雷达。

显然，这种基础的方案是及其浪费资源的。为了更加有效的使用无人机，并减少使用无人机的数目，可对问题进行优化。在我们建立的模型中，无人机只要在特定时刻飞过干扰点便可以达成目标，因此只要无人机在预设的轨道上飞过多个干扰点便可以达到减少无人机使用数量的目的。

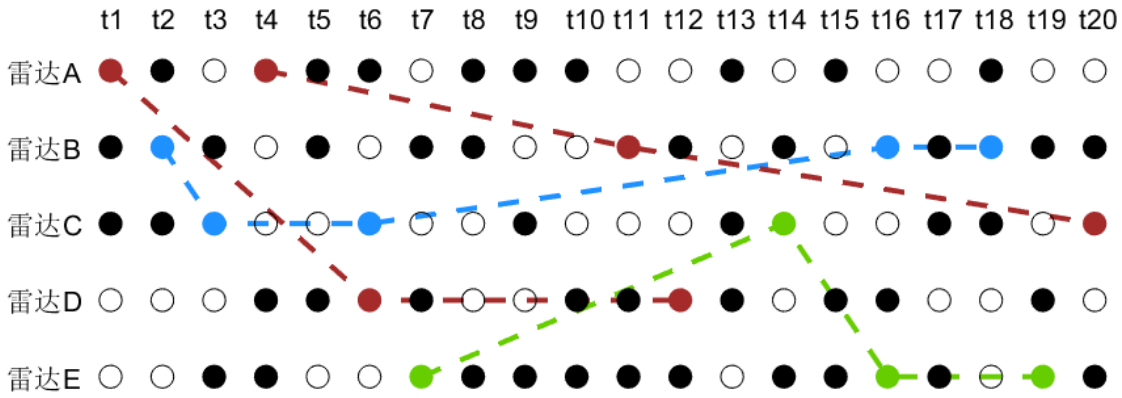

Fig.3 实心圆表示被干扰的雷达，彩色实心圆与虚线表示被单架无人机在不同时刻干扰了的雷达。

如 Fig.3 所示，当一架无人机可飞过多个干扰点的时候可以大量的减少所需无人机的数量。但无人机能飞过干扰点的数量和具体飞过哪些点由无人机自身机能与轨道约束所决定，因此在具体处理方案时还需对模型进行处理才可以运算。

III 匀速直线轨道

现已有欺骗轨道 Γ，以及任意时刻欺骗轨道点的坐标(x_{ti}, y_{ti}, z_{ti})以及雷达坐标(x_j, y_j, z_j)，于是可以确定空间直线 L_{ti}：

$$\frac{X-x_{ti}}{x_{ti}-x_j}=\frac{Y-y_{ti}}{y_{ti}-y_j}=\frac{Z-z_{ti}}{z_{ti}-z_j} \tag{1}$$

规定无人机速度范围为 120Km/h-180Km/h，且做匀速直线运动。可得无人机运动方程：

$$\begin{cases} X_i = V_x * t_i + x_0 \\ Y_i = V_y * t_i + y_0 \\ Z_i = V_z * t_i + z_0 \end{cases} \tag{2}$$

已知无人机在做干扰时必须处于直线 L_{ti} 之上，且无人机自身做匀速直线运动，将方程（1），（2）连列判断是否有解就可以判断无人机能否干扰雷达。

将以上方程作为条件放入遗传算法中进行运算。我们将一种无人机航行的完整方案视为一个生物个体（样本），其中每一架无人机的航线都是一条基因。将穿过干扰点多的基因视为优秀基因，穿过干扰点少的基因视为劣等基因。在问题中我们希望用尽可能少的无人机完成构架虚拟轨迹，因此最开始时每个样本都只具有一条基因（一架无人机）：

Fig.4 初始化模型。举例 4 个样本，括号内表示干扰点的数量。

从若干个样本中抽取数个优质样本，进行遗传过程（交叉与突变）

Fig.5 遗传示意图。

交叉过程可以从其他样本中学习优秀的基因（无人机轨迹）来优化自身，但无法产生新的轨迹；突变过程可以产生新的基因但不一定是优化。因此两种过程

按一定比例进行，并且不断重复遗传算法，可以使系统自身进行优化与进化，直到满足 Fig.2 的条件得到最优解。

IV 自由轨迹

在这个模型中无人机的速度为 120km/h~180km/h，飞行高度控制在 2000m~2500m，最大加速度不超过 10 m/s^2，转弯半径不小于 250m。因此无人机轨迹改为：

$$\begin{cases} X_i = x(t_{i/k}) \\ Y_i = y(t_{i/k}) \\ Z_i = z(t_{i/k}) \end{cases} \quad (3)$$

在公式（3）中，k 表示干扰雷达的时刻，i 表示其他时刻。为了简化运算，将实际的空间曲线轨道轨道表示为折线，在任意两个干扰点之间做直线运动。

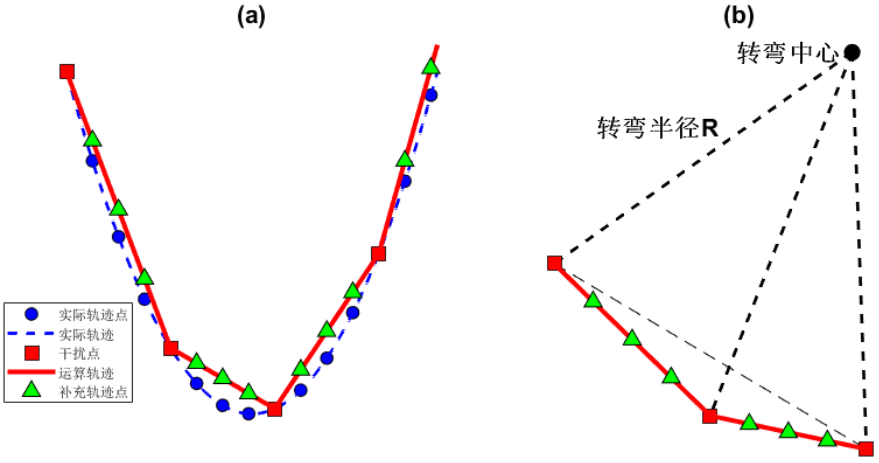

Fig.6 a）轨道粗滤化示意图，蓝色虚线为实际轨道，红色方块为干扰点。计算时无人机在干扰点间走直线轨道。中间时刻位置标记为绿色三角。B）相邻轨道点之间计算转弯半径。

根据以上设定，每次只需要判定三个相邻的干扰点 D_{k-1}, D_k, D_{k+1}：

$$\begin{cases} 120km/h \le V_{k-1,k,k+1} \le 180km/h \\ \frac{|V_k - V_{k-1}|}{|t_k - t_{k-1}|} < 10m/s^2 \\ \frac{|V_k - V_{k+1}|}{|t_k - t_{k+1}|} \le 10m/s^2 \\ R \ge 250 \end{cases} \quad (4)$$

4.1 计算已知轨道

在 III 所得结果中，每一个方案中任意轨道一定符合公式（4）的条件。因此如果两条轨道首尾干扰点符合公式（4）条件，便可相连。按照 Fig.7 的方案对 III 的结论带入遗传算法，当无人机轨迹数量等于 9 的时候结束运算。

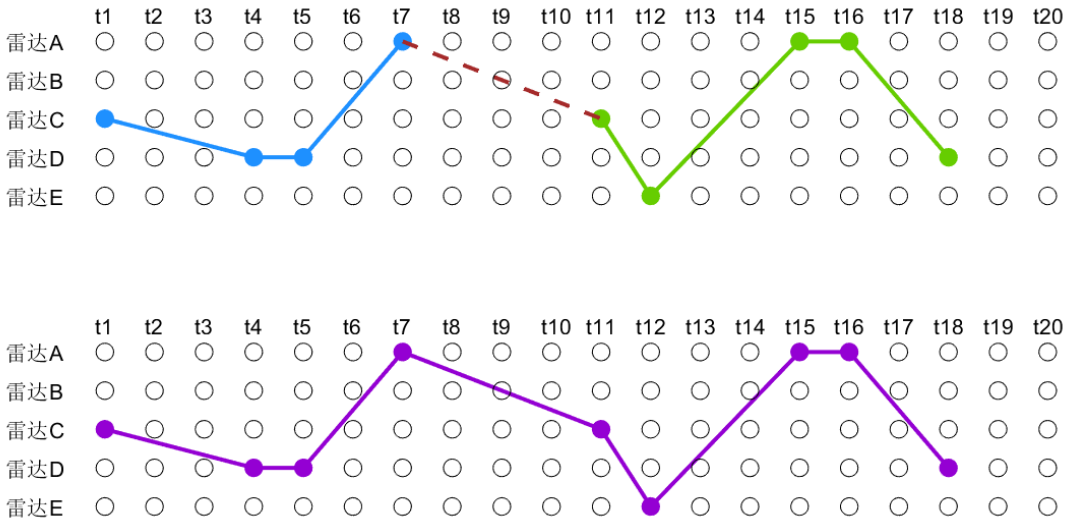

Fig.7 轨迹相连

Ⅴ 结论

遗传算法复杂度较高，在论文截至交稿时尚未完成运算，本章着重描述解决几个技术难点的计算细节：

1） 选择

设计了一个递归程序（附录 1），精简样本，防止样本容量过大。根据样本数量动态地调节适应度。适应度由两个参数（无人机数量与干扰点数量）进行一系列处理得到，让优质个体有更大的概率通过选择。并且考虑单个优秀基因的影响，可让结果收敛的更快。

2） 遗传

对单个个体进行有限制的繁殖（附录 2），并且有权重的选择进行重组或突变。对于优秀个体获得更多的“繁殖”机会。

a) 重组（附录 3）

随机从解集中抽选若干个待重组个体，有概率地挑选比自身优秀的基因进行传递。若重组个体的某基因与被重组个体相同位点无冲突（重组个体的无人机轨迹点与被重组个体无人机其余轨迹无重合）则直接复制。若有冲突但优秀基因所占位点与被重组个体对应位点所对应的无人机轨迹都优秀那么有极大概率进行附带并删除原占位基因。

b) 突变

随机选择该被突变个体可突变部分（所有未完成虚拟轨迹点时刻的集合）再随机选择若干点尝试构建新的轨道，并生成不良组合数集，在之后计算中不再使用，以提高效率。

VI 附录

1）

```
def Pick_better_solutions(all_information):  #kill
    solution_number = len(all_information)
    how_many_to_pick = 1000
    if solution_number <= how_many_to_pick:
        return all_information
    else:
        pick_ratio = (1 - how_many_to_pick / solution_number) * 0.5
        all_radar_fit = []
        all_UAV_number = []
        all_fit_ratio = []
        new_all_information = []
        for single_information in all_information:
            this_radar_fit = 0
            this_UAV_number = single_information[0][0][1]
            all_UAV_number = numpy.append(this_UAV_number, all_UAV_number)
            for a in range(radar_number):
                for b in range(time_list):
                    if single_information[1][1][b][a] >= 0:
                        this_radar_fit = this_radar_fit + 1
            all_radar_fit = numpy.append(this_radar_fit, all_radar_fit)
            try:
                fit_ratio = this_radar_fit / this_UAV_number
            except Exception:
                fit_ratio = 1
            finally:
                all_fit_ratio = numpy.append(fit_ratio, all_fit_ratio)
        max_radar_fit = max(all_radar_fit)
        min_radar_fit = min(all_radar_fit)
        valueable_fit = pick_ratio * max_radar_fit + (
            1 - pick_ratio) * min_radar_fit
        min_fit_ratio = min(all_fit_ratio)
        for i in range(len(all_information)):
            if all_radar_fit[i] >= valueable_fit:
                if all_fit_ratio[i] > min_fit_ratio:
                    new_all_information = numpy.append(all_information[i],
                                                       new_all_information)
        Pick_better_solutions(new_all_information)
```

2）

```
def evolution_of_solution(old_all_information):  #这是一个突变或者重组的过程
    evolution_time = 100
    new_all_sloution = []
```

```
    #old_solution 即为 all_solution 中的某个 all_UAV_information
    #old_solution 无需删除，它自然会因为劣势而被清洗
    new_all_information = copy.deepcopy(old_all_information)
    for i_information in old_all_information:
        #突变或者重组
        j = 0
        while j <= range(evolution_time):
            if random.randint(0, 50) <= 20:
                new_all_information.numpy.append(mutation(i_information))
            else:
                new_all_information = new_all_information + Reorganization(
                    i_information, old_all_information)
            j = j + 1
    delete_Duplicated_Element(new_all_information)
    for i in new_all_information:
        new_all_sloution = numpy.append(new_all_information[i][0],
                                        new_all_sloution)
    delete_Duplicated_Element(new_all_sloution)
    return new_all_sloution

def Reorganization(target_information, all_information):
    def fit_situation(single_information):
        single_fit_situation = numpy.array([])
        for i_UAV in range(single_information[0][1]):
            radar_to_fit = single_information[0][0][i_UAV][1]
            cover_number = 0
            for a in radar_to_fit:
                for b in radar_to_fit[a]:
                    if radar_to_fit[a][b] == True:
                        cover_number = cover_number + 1
            single_fit_situation = numpy.append([i_UAV, cover_number],
                                                single_fit_situation)
        return single_fit_situation
    Reorganization_time = 20
    Reorganization_ID = []
    #dlete_number = []
    #radar_to_fit = target_information[0][0][i][1]
    target_fit_situation = fit_situation(target_information)
    for i in range(Reorganization_time):
        this_ID = random.randint(0, len(all_information))
        if this_ID in Reorganization_ID:
            this_ID = random.randint(0, len(all_information))
    for i in Reorganization_ID:
        be_fitted_situation = fit_situation(all_information[i])
```

```
if min(target_fit_situation[:, 1]) >= max(be_fitted_situation[:, 1]):
    if random.random(0, 2) >= 1:
        #dlete_number = numpy.append(i, dlete_number)
        #del all_information[i]
        all_information[i] = target_information
else:
    #be_fitted_radar_to_fit = all_information[i][0][0][i][1]
    #1、如果 A 第 i 个无人机所对应的轨道在 B 上空，复制
    #2、如果 A 第 i 个无人机所对应的轨道在 B 上有被占，但是那两个占的轨道都没自己好，
    #   all_information 扩增占(记得删除旧轨道)或者没占
    for i_UAV in range(target_information[0][1]):
        target_radar_to_fit = target_information[0][0][i_UAV][1]
        for j_UAV in range(all_information[i][0][1]):
            be_fitted_radar_to_fit = all_information[i][0][0][j_UAV][1]
            similiar_strength = 0
            target_fit_number = 0
            be_fitted_fit_number = 0
            ready_to_cover = 0
            whose_will_cover = []
            for a in range(len(target_radar_to_fit)):
                for b in range(len(target_radar_to_fit[a])):
                    if target_radar_to_fit[a][b] == True:
                        target_fit_number = target_fit_number + 1
                        if target_information[1][1][b][a] == -1:
                            ready_to_cover = ready_to_cover + 1
                        else:
                            if target_information[1][1][b][
                                    a] in whose_will_cover:
                                pass
                            else:
                                whose_will_cover = numpy.append(
                                    target_information[1][1][b][a],
                                    whose_will_cover)
                    if be_fitted_radar_to_fit[a][b] == True:
                        be_fitted_fit_number = be_fitted_fit_number + 1
                    if (target_radar_to_fit[a][b] ==
                            be_fitted_radar_to_fit[a][b]):
                        similiar_strength = similiar_strength + 1
            if ready_to_cover == target_fit_number:
                all_information[i][0][1] = all_information[i][0][1] + 1
                all_information[i][0][0][all_information[i][0][
                    1]] = target_information[0][0][i_UAV]
            else:
                dont_cover = []
```

```
                for i_will_be_covered in whose_will_cover:
                    i_dont_cover = 0
                    for a in all_information[i][0][0][
                            i_will_be_covered][1]:
                        for b in all_information[i][0][0][
                                i_will_be_covered][1][a]:
                            if all_information[i][0][0][
                                    i_will_be_covered][1][a] == True:
                                i_dont_cover = i_dont_cover + 1
                    dont_cover = numpy.append(i_dont_cover, dont_cover)
                if target_fit_number >= max(dont_cover):
                    if random.random > 0.4:
                        whose_will_cover = whose_will_cover[
                            numpy.argsort(whose_will_cover)][::-1]
                        for i_will_be_covered in range(
                                len(whose_will_cover)):
                            del (all_information[i][0][0]
                                 [i_will_be_covered])
                            all_information[i][0][
                                1] = all_information[i][0][1] - 1
                        all_information[i][0][
                            1] = all_information[i][0][1] + 1
                        all_information[i][0][0][all_information[i][0][
                            1]] = target_information[0][0][i_UAV]
                    else:
                        pass
    return all_information
```

VII 引用

[1] Wikipedia, Genetic algorithm, https://en.wikipedia.org/wiki/Genetic_algorithm, 2018/09/18