2023 年美国数模赛电子科技大学模拟赛

承诺书

我们仔细阅读了数学建模竞赛的竞赛规则.

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮 件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料(包括网上查到的资料)，必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们的题目编号是（填写：A 或者 B）： A 题

我们报名队伍号是：L157

参赛队员姓名学号:

1. **王捷扬+2022120905014**
2. **段清宇+2022010906005**
3. **周靖雯+2022100902036**

指导教师或指导教师组负责人（有的话填写）:

是否愿意参加 2023年美国赛（是，否）： 是

日期：2022 年 11 月 28 日

Team Control Number

L157

Summary

Electromagnetic spectrum scheduling is one of the important technologies in unmanned systems, and now the main requirement about it is that each task can distinguish signals and receive them correctly, but for a multitasking system, it is difficult to avoid signal conflicts or signal interference among various tasks, so in this thesis we mainly discuss how to solve the problem by building mathematical models under the analysis of the intermittent characteristics of the electromagnetic spectrum in unmanned systems In this thesis, we discuss how to solve the problems of senders such as signal conflict and encounter, and the problems of receivers such as incomplete and long signal reception, so as to improve the efficiency of its use. In this thesis, after analyzing the operation of the electromagnetic spectrum in UAS and the signal reception analysis by combining the two, we agreed through our research and discussion that the model established in this paper can well solve a series of problems such as signal conflict, signal interference, and inability to receive beneficial signals in multi-task signal transmission and reception in UAS, so our group would like to Therefore, our group would like to present our insights through this paper. At the same time, we believe that the EM spectrum scheduling of unmanned systems can be better used under this approach, thus improving the efficiency of taskors and better applying this technology in practice. In the analysis of this problem, we adopt a step-by-step approach. In the first problem we build a model based on dichotomous search and linear programming and depth-first search, so that we can send as many periodic tasks as possible as a sender; in the second task we build a model based on a dynamic programming model with principle state compression, so that we can detect as many signals as possible as a searcher, and at the same time we In the third problem, we used the emergency task as a contingency to explore whether the different requirements of both the sender and the searcher are satisfied when the proportion of tasks is adjusted to the minimum within a given time, and through the simulation and analysis of the scenario, and with the help of multi-objective optimization, dichotomous search and greedy strategy, we again constructed a model and analyzed it objectively so as to The problem is solved perfectly. In the process of our research, we found that our algorithm is very extensive, and the models constructed above can be well applied to the EM spectrum management system, not only limited to the unmanned system, and we believe that the solutions brought by our models can be better applied to more places.

The title of solution

Contents

Introduce

Reviewing the whole research process, we mainly analyzed the operation mode of the electromagnetic spectrum in the UAS and the signal reception analysis, while referring to the two papers[1]~[2] recommended in the title, and under the comprehensive consideration of many aspects, we tried to optimize the electromagnetic spectrum management of the UAS with the help of establishing several mathematical models, and all the models and algorithms we explored in the paper are to try to All the models and algorithms in this paper are designed to solve the problems of signal conflicts and encounters, as well as incomplete signal reception and long reception time, so in this thesis, we focus on solving the problems of both the sender and the receiver, so as to improve the efficiency of the use of electromagnetic spectrum in unmanned systems.

1. For the sender, how can it send as many periodic tasks as possible in a given time period and make the moment of the first interference as late as possible? We use binary search and linear programming and depth-first search to build a mathematical model, and keep adjusting in the model to approach the critical value, so as to solve the problem and maximize the task sending.
2. For the receiver, how to be able to successfully detect as many signals as possible in a given time period, and at the same time, the first moment of the detection time should be as early as possible? For this problem, we build a reconnaissance policy model to optimize the detection of the received signal system by state compression of the dynamic programming model.
3. Finally for sudden urgent tasks need to be temporarily added or removed by the sender and receiver, the proportion of tasks that need to be adjusted in a given time is minimal, while the moment of interference on the sender side should be as good as possible, and the moment of new tasks on the receiver side should be successfully detected as early as possible, how should we model for both separately at this time? We solve this problem by building a model with the help of multi-objective optimization and another model with the help of binary search and greedy strategy.

In this study, we focus on the main goal of optimizing the system, and we continue to improve our algorithms and simplify our models to solve problems and improve the efficiency of the use of the electromagnetic spectrum in multiple fields by building models to understand the electromagnetic spectrum management system in depth.

The first problem

For the first problem, the goal is to maximize the time of the first conflict.

Set $task_i$'s period be 'T_i, phase be ϕ_i, and duration be $D_i + \tau_0$

Let's consider the case of 2 tasks, when the two tasks collide while $task_1$ in its cycle a and $task_2$ in its cycle b, it meets the following conditions:

$$
\begin{gathered}
|(aT_1 + \phi_1) - (bT_2 + \phi_2)| \leqslant \frac{D_1 + D_2 + 2\tau_0}{2} - \tau_0 = \frac{D_1 + D_2}{2} \\
let\ \ \Delta\phi = \phi_1 - \phi_2,\ Thus: \\
\begin{cases}
a \leq U(b) = \dfrac{T_2 b - \Delta\phi}{T_1} + \dfrac{D_x + D_y}{2} \\
a \geq L(b) = \dfrac{T_2 b - \Delta\phi}{T_1} - \dfrac{D_x + D_y}{2}
\end{cases}
\end{gathered}
$$

If there exits integers a and b meet the above formulas, its show that when $task_1$ in its cycle a and $task_2$ in its cycle b, they will cause conflicts.

And the time of the first conflict can be described as $FC(\phi_1, T_1, \phi_2, T_2) = T_2 b + \phi_2$

So far , we get the following model:

$$
\begin{gathered}
maxmize\ min(FC(\phi_i, T_i, \phi_j, T_j)) \\
i \neq j \\
i, j \in [1, n] \cap Z
\end{gathered}
$$

Assume that ϕ_1 and ϕ_2 have been determined , the minimum positive integers a and b can be worked out by following formulas:

set:

$$
\begin{gathered}
k = \frac{T2}{T1}, k_1 = k - \lfloor k \rfloor \\
f_0(b) = \lceil U(0) \rceil - U(0) - k_1 b \\
g_0(b) = \lceil U(0) \rceil - L(0) - k_1 b \\
i \in Z \\
f_i(b) = F_0(b) + i \\
g_i(b) = G_0(b) + i
\end{gathered}
$$

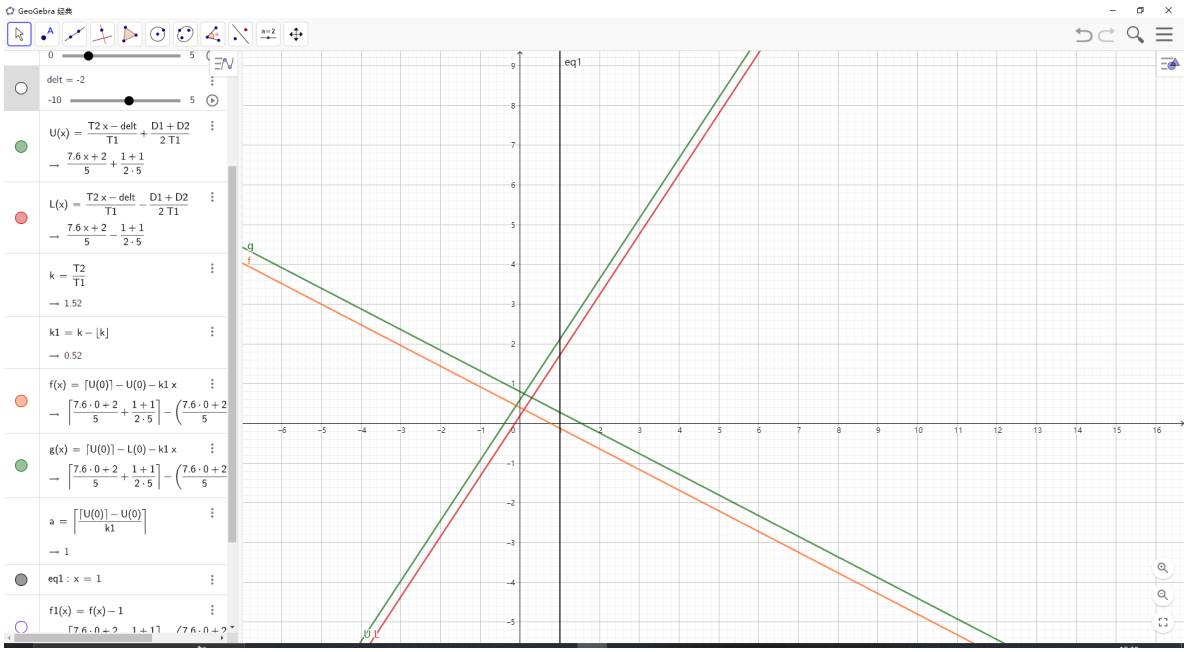
We can find that:

$$
\begin{gathered}
when\ F_i(b) \leq 0\ and\ G_i(b) \geq 0 \\
\exists a \in [\lceil L(b) \rceil, \lfloor U(b) \rfloor] \cap Z\ meets \begin{cases}
a \leq U(b) = \dfrac{T_2 b - \Delta\phi}{T_1} + \dfrac{D_x + D_y}{2} \\
a \geq L(b) = \dfrac{T_2 b - \Delta\phi}{T_1} - \dfrac{D_x + D_y}{2}
\end{cases}
\end{gathered}
$$

So when it comes to finding the minimum integers a and b , the first thing we need to do is find b fitting $f_i(b) \leq 0\ and\ g_i(b) \geq 0$

set $F_i = \lceil \frac{\lceil U(0) \rceil - U(0) + i}{k_1} \rceil$, $G_i = \lfloor \frac{\lceil U(0) \rceil - L(0) + i}{k_1} \rfloor$

if $F_i \leq G_i$, there exists $\exists b \in [F_i, G_i] \cap Z$ meets $f_i(b) \leq 0 \; and \; g_i(b) \geq 0$



As shown in figure, point (1,2) is the minimum positive integer solution of the inequality, it's under the U(b) and above the L(b).

Based on the above principles, we can write the following program in order to find the time of the first conflict of two tasks,and this algorithm is one order of magnitude faster than brute force search:

```
//Time complexity is O(n), compared with O(n^2) of brute force
double first_conflict(Wave a,Wave b,int boundary=100){
    if(b<a)swap(a,b);
    if(a.D<0||b.D<0)return inf;
    double delt=a.phi-b.phi,width=(a.D+b.D)/a.T,k=b.T/a.T;
    double U0=-delt/a.T+width/2,L0=-delt/a.T-width/2;
    double k1=k-floor(k),U0_=ceil(U0),L0_=floor(L0);
    if(k1<eps&&U0_-L0_<=1)return inf;
    if(k1<eps&&U0_-L0_>1+eps){
        double re=b.phi-floor(b.phi/b.T)*b.T;
        return re;
    }
    for(int i=-boundary;i<=boundary;i++){
        double F=(U0_-U0+i)/k1,G=(U0_-L0+i)/k1;
        int b_min=(int)ceil(F),b_max=(int)floor(G);
        if(b_max*b.T+b.phi<0)continue;
        for(int b_=b_min;b_<=b_max;b_++){
            double Ub=U0+k*b_;
            int a_=floor(Ub);
            if(b_*b.T+b.phi<0||a_*a.T+a.phi<0)continue;
            return b_*b.T+b.phi;
        }
    }
}
```

After solved the problem of finding a, b , followings will talk about the strategy of setting ϕ to maximize FC

Following are two different strategies:

1.Maximize the FC when the tasks are required to be all performed.

Considering the binary searching, at first we set $l = 0$, $r = k \times T_{max}$ as the boundary of the binary searching.

```
double l=0,r=maxL*128,ans=0;
while(r-l>minL*eps){
    double mid=(r+l)/2;
    cerr<<"checking "<<mid<<"\n";
    if(Dfs(n,mid,n)){
        l=ans=mid;
        cerr<<mid<<" is achievabel"<<endl;
        for(int i=1;i<=n;i++){
            while(wave[i].phi>wave[i].T)wave[i].phi-=wave[i].T;
            phi[wave[i].id]=wave[i].phi;
        }
    }else{
        r=mid;
        cerr<<mid<<" is unachievabel"<<endl;
    }
}
```

Then, at each time set $mid = \frac{l+r}{2}$ and check whether the global FC_{max} can exceed mid

Therefore use the *Depth First Search* to check whether FC_{max} can reach mid:

```
bool Dfs(int id,double lim,int goal){
    if(goal==0)return true;
    if(goal>id)return false;
    if(id==n){
        wave[id].phi=0;
        wave[id].place=true;
        if(Dfs(id-1,lim,goal-1))return true;
        else{
            wave[id].place=false;
            return Dfs(id-1,lim,goal);
        }
    }
    wave[id].phi=wave[id+1].phi;
    wave[id].place=true;
    if(wave[id].D<0)
        return Dfs(id-1,lim,goal-1);
    double step=wave[id].T/steps;
    for(int i=0;i<steps;i++,wave[id].phi+=step){
        double farthest=inf;
        for(int j=id+1;j<=n;j++){
```

```
            if(!wave[j].place)continue;
            double tmp=first_conflict(wave[id],wave[j]);
            farthest=min(farthest,tmp);
        }
        if(farthest<lim)continue;
        if(Dfs(id-1,lim,goal-1))return true;
    }
    wave[id].place=false;
    return Dfs(id-1,lim,goal);
}
```

following is a test data describing a multi-tasks of 10, each row presenting the D_i and T_i

```
10 0.06623888363332854
0.5654156823480465 8.299971234788403
0.34809079800211873 6.727747744287007
0.3216343054048834 7.004151867048586
0.24141098417300982 4.167094048341133
0.006367100772135271 4.807209417901058
0.1802396932813039 6.9680332455544995
0.04384768436222811 1.9675609671005667
0.020277296636061357 1.0164766671218217
0.6733342288255395 9.609498362844334
0.5539618496415146 7.833455372654835
```

following is the running result of the program, solving a multi-tasks of 10

```
The maximum time of the first interference is 57.66
phi(1) is 5.98
phi(2) is 6.65
phi(3) is 1.99
phi(4) is 3.18
phi(5) is 0.87
phi(6) is 4.01
phi(7) is 0.41
phi(8) is 0.64
phi(9) is 0.00
phi(10) is 2.84
```

As it shows, under the optimal strategy the first interference occurs at time **57.66(a average of 12 cycles)**, and output the ϕ of each task.

2.Maximize the numbers of tasks being performed when a determined FC_{max} is given.

The binary searching and DFS works as well, the only thing changed is that the threshold to be *Check* is no longer the FC_{max} but the N_{max} which means the number of tasks can be performed.

```
To prevent first interference from occuring before 80.00 , the maximum
amount of tasks is 8
phi(1) is 4.48
phi(2) is 1.45
task 3 can't be performed under current conditions.
phi(4) is 2.94
phi(5) is 4.33
phi(6) is 3.15
phi(7) is 0.63
phi(8) is 0.60
phi(9) is 0.00
task 10 can't be performed under current conditions.
```

As it shows, determining the FC_{max} to reach 80 , the maximum number of tasks to be performed is 8, while $task_3$ and $task_{10}$ can't be performed.

The second problem

set $E1$ to be the set of signals sent by scout， $E2$ to be the set of signals sent by sender，$\phi 1$ to be set of scout's central lines of signals，$\phi 2$ to be set of sender's central lines of signals，similarly，$D1$ and $D2$ refer the width of signals，$T1$ and $T2$ refer period，$Fr1$ and $Fr2$ refer frequencies.

According to *Optimisation of periodic search strategies for electronic support*，the signals sent by scout on different frequencies are of same period，It can be known from mathematical derivation that the conflict between signals x and y occurs when $\exists a,b \in \mathbb{N}$, meets

$$|\phi_x + aT_x - \phi_y - bT_y| \leq \frac{D_x + D_y}{2} - \tau_0$$

so we build the following functions:

$$f(x,y,a,b) = \begin{cases} 1 & |\phi_x + aT_x - \phi_y - bT_y| \leq \dfrac{D_x + D_y}{2} - \tau_0 \\ 0 & |\phi_x + aT_x - \phi_y - bT_y| \geq \dfrac{D_x + D_y}{2} - \tau_0 \end{cases}$$

$$W(x,y) = \begin{cases} 1 & Fr_x = Fr_y \\ 0 & Fr_x \neq Fr_y \end{cases}$$

So far we get the model as following:

$$\begin{aligned} &\max \sum_{i=1}\sum_{j=1} W(E1_i, E2_j) \sum_{a=0}\sum_{b=0} f(E1_i, E2_j, a, b) \\ &st. \sum_{i=1}\sum_{j=i+1}\sum_{a=0}\sum_{b=0} f(E1_i, E1_j, a, b) = 0 \end{aligned}$$

Because the signals sent by scout on different frequencies are of same period，so we only need to consider the first period when considering the restrictions

$$st.\sum_{i=1}\sum_{j=i+1} f(E1_i,E1_j,0,0)=0$$

However, in addition to maximizing the number of detectable signals, we also need to make the first conflict as early as possible,consider a coefficient $\frac{k\times len-FC(x,y)}{k\times len}$， in which len is the total length of time given， $FC(x,y)$ is the first time x,y have conflicted， k is a parameter, finally we get the following model:

$$\max\sum_{i=1}\sum_{j=1} W(E1_i,E2_j)\frac{k\times len-FC(E1_i,E2_j)}{k\times len}\sum_{a=0}\sum_{b=0} f(E1_i,E2_j,a,b)$$
$$st.\sum_{i=1}\sum_{j=i+1} f(E1_i,E1_j,0,0)=0$$

In the process of solving the model, we adopted a dynamic programming method based on state compression. In the progress of calculating $\sum_{a=0}\sum_{b=0} f(E1_i,E2_j,a,b)$， We used the method improved from the algorithm that calculate the $FC(x,y)$ in the first problem, and the actual effect is relatively ideal.

set $g(x,y)$ to be the maximum contribute can be made at the step of x and the situation of used frequency is y， in which y is a string of binary numbers， The i-th bit under y binary indicates the usage of the i-th frequency band. The transfer equation is:

$$g(x,y)=\max\{g(i,j)+\sum_{j=1} W(V,E2_j)\frac{k\times len-FC(V,E2_j)}{k\times len}\sum_{a=0}\sum_{b=0} f(V,E2_j,a,b)\}(i<x)$$

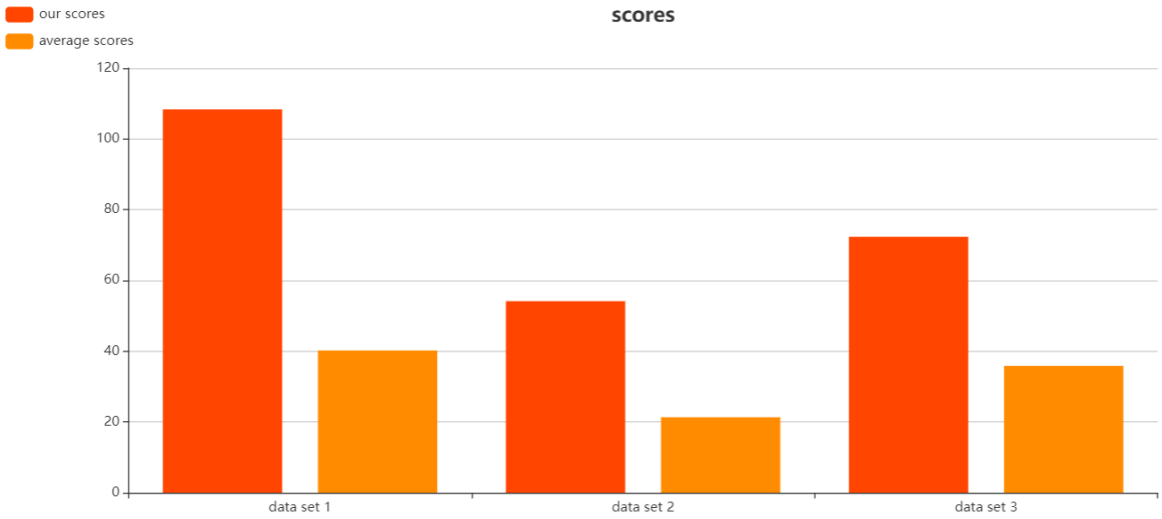
In which V is a signal of which the end position is x, frequency band is $y\ xor\ j$, period is T and pulse width is $x-i$

Following is the code of dynamic programming.

```
for(int pv=1;pv<=steps;pv++){
    for(int j=0;j<(1<<cnt);j++){
        for(int i=1;i<pv;i++){
            double d=(double)(pv-i)/steps*T;
            double phi_now=(double)(pv+i)/(2*steps)*T;
            if(f[pv][j].val<=f[i][j].val){
                f[pv][j].val=f[i][j].val;
                f[pv][j].du=d;
                f[pv][j].phi=phi_now;
                f[pv][j].lstp=i;
                f[pv][j].lsts=j;
            }
            if(d<ml||d>md)continue;
            for(int k=1;k<=cnt;k++){
                double re=0;
                if(!(j&(1<<(k-1))))continue;
                for(auto z:p[k])
                    re+=conflict(phi_now,t*mt,z,d-ml);
                int state=j^(1<<(k-1));
                if(f[pv][j].val<=f[i][state].val+re+reward){
                    f[pv][j].val=f[i][state].val+re+reward;
```

```
                    f[pv][j].du=d;
                    f[pv][j].phi=phi_now;
                    f[pv][j].lstp=i;
                    f[pv][j].lsts=state;
                }
            }
        }
    }
}
```

We generated several groups of data and calculated the average score of the random decision scheme and the score of our algorithm scheme under the same group of data. It can be seen that we should be twice as good or even more.



The third problem (sender part)

When it comes to a urgent task needed to be added into the task list, we are trying to minimize the $Proportion$ of tasks being modified while maximizing the time of the $First\ Conflict$. In ideal, a valuation function $V(proption, FC_{max})$ should be build to optimize the strategy.

However, $Proportion_{min}$ and FC_{max} is conjugate, it's hard to optimize the two values in same time, thus consider a phased optimization.

First, build a function $FC_{max} = MF(p)$, in which $MF(p)$ stands for the highest FC value can be get while the $Proportion$ do not exceed p

Then, build another function $proportoin_{max} = MP(FC_{max})$, in which $MP(FC_{max})$ stands for the highest $Proportion$ can be reached while FC no less than FC_{max}

Thus, our model is to maximize $MF(p)$ and then minimize $MP(MF(p))$, in which we can set p a value such as 0.3

Binary search and *greedy strategy* are chosen to be the algorithms for the optimization.

Whatever the strategy chosen, the $MF(p)$ will not exceed the FC_{max} calculated out in *the first problem*, so choose it to be the upper bound of the binary search.

When modifying the tasks, the task which conflicts most with other tasks while also conflicting with the urgent task should be adjusted in priority, that's how greedy strategy works:

```
int id=-1,maxc=0;
for(int i=1;i<=n;i++){
    int conflicts=0;
    double fc=first_conflict(wave_[i],ugt);
    if(fc<lim)conflicts=n;
    for(int j=1;j<=n;j++){
        if(i==j)continue;
        fc=first_conflict(wave_[i],wave_[j]);
        if(fc>=lim)continue;
        conflicts++;
    }
    if(conflicts>maxc){
        maxc=conflicts;
        id=i;
    }
}
```

Above is the greedy strategy of selecting the first task to be adjusted.

Following is test data of 5 periodic tasks and 1 urgent task

```
5 0.1256
7.1183 82.4657
2.8997 59.9758
0.0974 5.5539
1.1685 25.7628
0.0615 0.4391

0.7590 17.2351
```

following is the program's output:

```
The maximum time of the first interference is 412.33
phi(1) is 0.00
phi(2) is 13.19
phi(3) is 2.83
phi(4) is 25.05
phi(5) is 0.02

Now considering insert a urgent task.
With changed proportion of 0.20 the maximum first interference is 328.20
phi(1) is 80.82
phi(2) is 13.19
phi(3) is 2.83
phi(4) is 25.05
phi(5) is 0.02
urgent phi is 0.00
```

It shows that the best strategy of inserting the urgent task is to move $task_1$'s ϕ from 0 to 80.82, and set urgent task to 0, which can make the FC up to 328.20 while only adjust 1 task.

However, amount of experiments tell a fact that in most of time there needs no change to achieve the original FC_{max} after inserting the urgent task, just as the following example:

```
The maximum time of the first interference is 512.84
phi(1) is 10.76
phi(2) is 0.00
phi(3) is 16.59
phi(4) is 0.00
phi(5) is 25.20
phi(6) is 9.20
phi(7) is 38.85
phi(8) is 36.11
phi(9) is 38.85
phi(10) is 30.87

Now considering insert a urgent task.
With changed proportion of 0.00 the maximum first interference is 512.81
phi(1) is 10.76
phi(2) is 0.00
phi(3) is 16.59
phi(4) is 0.00
phi(5) is 25.20
phi(6) is 9.20
phi(7) is 38.85
phi(8) is 36.11
phi(9) is 38.85
phi(10) is 30.87
urgent phi is 62.94
```

The third problem (scout part)

We set V to represent the strategy of scout and U to represent the whole strategy space. We have two standards to value V: the proportion of tasks to be adjusted and the first moment of new task is successfully detected . Taking the two factors into consideration, we defined the function W

$$W(V) = k_1 \times (1 - \frac{num}{size}) + k_2 \times (1 - \frac{FC}{len})$$

num in the formula represents the number of tasks to be adjusted after we adopt the strategy V. $size$ represents the number of tasks before change. FC represents the moment of new task is successfully detected. len represents the given time. Besides, k_1 and k_2 are parameter.

According to the function, we can establish the following model

$$\max W(V)$$
$$st. V \in U$$

Actually, U is very huge. To improve the efficiency of our model, we need to further reduce the strategy space.

First, let's consider a urgent task. Now, we want to detect the urgent task. To achieve our goal, we can find that it's useless to barely delete a original task sent by the scout. Generally, we need to add signal for the scout instead of deleting. It's very reasonable.

Second, what we need to do is barely detect the new task. We don't need any extra goals. So, the duration of new signal that we are going to send is exactly τ_0.

Based on the above two constraints, we get U_1 as a subset of U, and establish the following model

$$\max W(V)$$
$$st. V \in U_1$$

To show the superiority of our model, we generate some data sets. Then we calculate the average scores of stochastic decision model and our scores. It can be seen that our strategy gets higher scores of % 20~% 30.

Conclusion

With the help of several optimization algorithms and strategy models, we have solved to a large extent the sender's problems such as signal conflict and encounter, and receiver's problems such as incomplete signal reception and long time, as well as the problems in transmitting and receiving for unmanned system's electromagnetic spectrum management system. In addition, due to the wide range and usability of our algorithm, our optimized model for the EM spectrum management system of unmanned systems can be applied to the EM spectrum management in many fields, such as maritime access systems. We are confident that our solution, although it still has some shortcomings, is able to solve the existing problems and, in general, our team is very sure of our proposed approach.

References

[1] I. Vaughan L. Clarkson. Optimisation of periodic search strategies for electronic support. *IEEE Transactions on Aerospace and Electronic Systems*, 47(3), 2011: 1770-1784.

[2] I. Vaughan L. Clarkson, The arithmetic of receiver scheduling for electronic support. *In proceedings of the Aerospace Conference, vol. 5, Mar. 2003, 2049-2064*.

Appendixes

```
//problem 1, strategy 1
#include<bits/stdc++.h>
using namespace std;
const int N=114514;
const double eps=1e-3,inf=1145141919810;
int n,tot;
int steps=50;
struct Wave{
    double D,T,phi;
    int id;
    bool place;
    bool operator < (const Wave &a)const{return T<a.T;}
}wave[N];
double first_conflict(Wave a,Wave b,int boundary=100){
    if(b<a)swap(a,b);
    if(a.D<0||b.D<0)return inf;
    double delt=a.phi-b.phi,width=(a.D+b.D)/a.T,k=b.T/a.T;
    double U0=-delt/a.T+width/2,L0=-delt/a.T-width/2;
    double k1=k-floor(k),U0_=ceil(U0),L0_=floor(L0);
    if(k1<eps&&U0_-L0_<=1)return inf;
    if(k1<eps&&U0_-L0_>1+eps){
        double re=b.phi-floor(b.phi/b.T)*b.T;
        return re;
    }
    for(int i=-boundary;i<=boundary;i++){
        double F=(U0_-U0+i)/k1,G=(U0_-L0+i)/k1;
        int b_min=(int)ceil(F),b_max=(int)floor(G);
        if(b_max*b.T+b.phi<0)continue;
        for(int b_=b_min;b_<=b_max;b_++){
            double Ub=U0+k*b_;
            int a_=floor(Ub);
            if(b_*b.T+b.phi<0||a_*a.T+a.phi<0)continue;
            return b_*b.T+b.phi;
        }
    }
}
double maxL=-inf,minL=inf,ml;
bool Dfs(int id,double lim,int goal){
    if(goal==0)return true;
    if(goal>id)return false;
```

```
    if(id==n){
        wave[id].phi=0;
        wave[id].place=true;
        if(Dfs(id-1,lim,goal-1))return true;
        else{
            wave[id].place=false;
            return Dfs(id-1,lim,goal);
        }
    }
    wave[id].phi=wave[id+1].phi;
    wave[id].place=true;
    if(wave[id].D<0)
        return Dfs(id-1,lim,goal-1);
    double step=wave[id].T/steps;
    for(int i=0;i<steps;i++,wave[id].phi+=step){
        double farthest=inf;
        for(int j=id+1;j<=n;j++){
            if(!wave[j].place)continue;
            double tmp=first_conflict(wave[id],wave[j]);
            farthest=min(farthest,tmp);
        }
        if(farthest<lim)continue;
        if(Dfs(id-1,lim,goal-1))return true;
    }
    wave[id].place=false;
    return Dfs(id-1,lim,goal);
}
double phi[N];
signed main(){
    //freopen("data/task1/2.txt","r",stdin);
    //freopen("result/task1/1.out","w",stdout);
    scanf("%d%lf",&n,&ml);
    double dsum=0;
    for(int i=1;i<=n;i++){
        scanf("%lf%lf",&wave[i].D,&wave[i].T);
        maxL=max(maxL,wave[i].T);
        wave[i].D-=ml;
        if(wave[i].D>0)minL=min(minL,wave[i].T);
        wave[i].id=i;
        dsum+=wave[i].D;
    }
    sort(wave+1,wave+n+1);
    double l=0,r=maxL*128,ans=0;
    while(r-l>minL*eps){
        double mid=(r+l)/2;
        cerr<<"checking "<<mid<<"\n";
        if(Dfs(n,mid,n)){
            l=ans=mid;
            cerr<<mid<<" is achievabel"<<endl;
            for(int i=1;i<=n;i++){
                while(wave[i].phi>wave[i].T)wave[i].phi-=wave[i].T;
                phi[wave[i].id]=wave[i].phi;
```

```
            }
        }else{
            r=mid;
            cerr<<mid<<" is unachievabel"<<endl;
        }
    }
    if(ans<eps){
        printf("There is no way to prevent the first interference from
occuring in the first cycle.");
        return 0;
    }
    printf("The maximum time of the first interference is %.2lf\n",ans);
    for(int i=1;i<=n;i++){
        printf("phi(%d) is %.2lf\n",i,phi[i]);
    }
    return 0;
}
```

```
//problem 1, strategy 2, extract
int l=0,r=n,ans=0;
double lim=80;
while(l<=r){
    int mid=(r+l)/2;
    cerr<<"checking "<<mid<<"\n";
    if(Dfs(n,lim,mid)){
        ans=mid;
        l=mid+1;
        cerr<<mid<<" is achievabel"<<endl;
        for(int i=1;i<=n;i++){
            while(wave[i].phi>wave[i].T)wave[i].phi-=wave[i].T;
            phi[wave[i].id]=wave[i].phi;
            apr[wave[i].id]=wave[i].place;
        }
    }else{
        r=mid-1;
        cerr<<mid<<" is unachievabel"<<endl;
    }
}
```

```
//problem 2, extract
......
double count_conflict(spec A,spec b,int boundary=100){
    if(b.T<A.T) swap(A,b);
    if(A.d<0||b.d<0) return 0;
    double delt=A.phi-b.phi,width=(A.d+b.d)/A.T,k=b.T/A.T;
    double U0=-delt/A.T+width/2,L0=-delt/A.T-width/2;
    double k1=k-floor(k),U0_=ceil(U0),L0_=floor(L0);
    if(k1<eps&&U0_-L0_<=1)return 0;
    double re=0;
    if(k1<eps&&U0_-L0_>1+eps){
        re=floor((len-b.phi)/b.T);
```

```
        return re;
    }
    double first=0;
    for(int i=-boundary;i<=boundary;i++){
        double F=(U0_-U0+i)/k1,G=(U0_-L0+i)/k1;
        int b_min=(int)ceil(F),b_max=(int)floor(G);
        if(b_max*b.T+b.phi<0)continue;
        if(b_min*b.T+b.phi>len)break;
        for(int b_=b_min;b_<=b_max;b_++){
            double Ub=U0+k*b_;
            int a_=floor(Ub);
            if(b_*b.T+b.phi<0||a_*A.T+A.phi<0)continue;
            if(b_*b.T+b.phi>len||a_*A.T+A.phi>len)break;
            re+=1;
            if(first<eps)first=b_*b.T+b.phi;
        }
    }
    if(re>eps)re+=reward;
    return re*(2*len-first)/(2*len);
}
double conflict(double pi,double t,int id,double d){
    spec x;
    x.phi=pi;x.T=t;x.d=d;
    double re=count_conflict(x,a[id]);
    return re;
}
signed main(){
......
    for(double t=0.1;t<=1;t+=0.1){
        double T=t*mt;
        for(int pv=1;pv<=steps;pv++){
            for(int j=0;j<(1<<cnt);j++){
                for(int i=1;i<pv;i++){
                    double d=(double)(pv-i)/steps*T;
                    double phi_now=(double)(pv+i)/(2*steps)*T;
                    if(f[pv][j].val<=f[i][j].val){
                        f[pv][j].val=f[i][j].val;
                        f[pv][j].du=d;
                        f[pv][j].phi=phi_now;
                        f[pv][j].lstp=i;
                        f[pv][j].lsts=j;
                    }
                    if(d<ml||d>md)continue;
                    for(int k=1;k<=cnt;k++){
                        double re=0;
                        if(!(j&(1<<(k-1))))continue;
                        for(auto z:p[k])
                            re+=conflict(phi_now,t*mt,z,d-ml);
                        int state=j^(1<<(k-1));
                        if(f[pv][j].val<=f[i][state].val+re+reward){
                            f[pv][j].val=f[i][state].val+re+reward;
                            f[pv][j].du=d;
```

```
                            f[pv][j].phi=phi_now;
                            f[pv][j].lstp=i;
                            f[pv][j].lsts=state;
                        }
                    }
                }
            }
        }
......
}
```

```
//problem 3, sender part, extract
bool Check(Wave ugt,double lim,double propmax,int boundary=100){
    for(int i=1;i<=n;i++)wave_[i]=wave[i],change[i]=false;
    int csum=0;
    for(int turn=1;turn<=boundary;turn++){
        int id=-1,maxc=0;
        for(int i=1;i<=n;i++){
            int conflicts=0;
            double fc=first_conflict(wave_[i],ugt);
            if(fc<lim)conflicts=n;
            for(int j=1;j<=n;j++){
                if(i==j)continue;
                fc=first_conflict(wave_[i],wave_[j]);
                if(fc>=lim)continue;
                conflicts++;
            }
            if(conflicts>maxc){
                maxc=conflicts;
                id=i;
            }
        }
        if(id==-1){
            prop=(double)csum/n;
            return prop<=propmax+eps;
        }
        if(!change[id]){
            csum++;
            change[id]=true;
        }
        if((double)csum/n>propmax)return false;
        Wave pwave=wave_[id];
        for(int i=0;i<steps;i++){
            pwave.phi=pwave.T*i/steps;
            int conflicts=0;
            double fc=first_conflict(pwave,ugt);
            if(fc<lim)conflicts=n;
            for(int j=1;j<=n;j++){
                if(id==j)continue;
                fc=first_conflict(pwave,wave_[j]);
                if(fc>=lim)continue;
```

```
                conflicts++;
            }
            if(conflicts<maxc){
                maxc=conflicts;
                wave_[id]=pwave;
            }
        }
    }
    return false;
}
```

```
//problem 3, scout part, extract
scanf("%d",&m);
for(int i=1;i<=m;i++){
    spec x;
    scanf("%lf%lf%lf%lf",&x.phi,&x.d,&x.T,&x.hz);
    flag=0;
    if(x.d<ml){
        printf("It's impossible to detect the signal\n");
        continue;
    }
    spec y;
    y.d=ml;y.T=rT;y.hz=x.hz;
    y.phi=x.phi-x.d/2+ml/2;
    double l=y.phi,r=x.phi+x.d/2-ml/2,mw=0,mphi=-1;
    for(auto v:ans[rid]){
        if(abs(v.hz-x.hz)>eps) continue;
        double dd;
        if(x.T<rT) dd=v.d;
        else dd=x.d;
        v.d-=ml/2;x.d-=ml/2;
        double fc=first_conflict(v,x);
        if(fc>eps) mw=K*(1-(fc-dd/2)/len)+K2;
    }
    for(double now_phi=l;now_phi<=r;now_phi+=0.01*(r-l)){
        double w=0,re=0;
        w+=(1-(now_phi-ml/2)/len)*K;
        y.phi=now_phi;
        if(y.phi>=rT){
            int kl=y.phi/rT;
            y.phi=y.phi-kl*rT;
        }
        for(auto v:ans[rid])
            if(count_conflict(v,y)) re=re+1;
        w+=K2*(1-re/ans[rid].size());
        if(mw<w) mw=w,mphi=now_phi;
    }
    if(mphi==-1) printf("Value:%lf\n No Change Required\n",mw);
    else{
        y.phi=mphi;
        for(int i=0;i<ans[rid].size();i++)
```

```
            if(count_conflict(ans[rid][i],y))
                ans[rid][i].d=abs(y.phi-ml/2-ans[rid][i].phi);
        ans[rid].push_back(y);
        printf("We make changes like these:\n");
        printf("Value:%lf\n",mw);
        for(auto v:ans[rid])
            printf("phi=%lf duration=%lf frequency=%lf
T=%lf\n",v.phi,v.d,v.hz,rT);
    }
}
```