智能优化算法报告素材

题目：基于遗传算法的条件指派问题求解

报告摘要：300字以内

一、问题描述：（请尝试给出数学模型）

指派问题，又称分配问题，可以抽象为m个物品与n个背包的匹配问题，传统的平衡指派问题（Assignment Problem，AP）研究的是单纯的一对一匹配问题。但在实际中，许多指派问题有着复杂的限制条件，难以采用传统的匈牙利法来求解。本报告主要就求解皇家狩猎问题展开。皇家狩猎问题是一个带有条件约束的指派问题，其分配对象是狩猎活动中的人员与马匹，除了满足指派问题的约束条件之外，还要满足以下条件：

- 皇上需要比其他人都更享受这次狩猎。
- 除非马的数量不够，否则所有人都应该骑马。
- 如果一个要员比另外一个要员阶级更高，那么就应该符合以下任意一种情况：
  （a）等级较高的要员骑的马应该比等级较低的要员骑的马更骏伟或两者同等俊伟；
  （b）等级较低的要员没有骑马；
  （c）两个人都没有骑马。
- 如果一匹马比另外一匹马更快，那么就应该符合以下任意一种情况：
  （a）骑快马的人的骑马技术不应该比骑慢马的人的更差；
  （b）没有人骑快马；
  （c）两匹马都没有人骑。

这次的目标是要使所有参加这次狩猎的人的愉悦度最大化。实际上这些约束通常很难同时被满足。所以最后的约束是可以被违反的，不过每次违反都会对目标数值有 100 的降低惩罚。

对相关的变量和参数做如下定义和说明：

- $Court$，人员集合，编号为$1..n$
- $Horse$，马匹集合，编号为$1..m$
- R_i，人员等级，$i \in Court$
- A_i，人员骑术水平，$i \in Court$
- B_j，马匹俊美程度，$j \in Horse$
- S_j，马匹奔跑速度，$j \in Horse$
- emp，皇帝编号，$emp \in Court$

变量：

- x_{ij}，取1表示人员i使用马匹j，否则取0
- $breakrules$，规则破坏数目，为非负整数

指派问题模型如下：

$$\max\ Obj = \sum_{i \in Court} \sum_{j \in Horse} E_{ij} \cdot x_{ij} - 100 \cdot breakrules$$

$$s.t. = \begin{cases} \sum_{j \in Horse} x_{ij} \leq 1 \ \forall i \in Court \\ \sum_{i \in Court} x_{ij} \leq 1 \ \forall j \in Horse \\ x_{ij} = 0, 1 \ \forall i \in Court, j \in Horse \end{cases}$$

相比于指派问题，还增加了以下约束：

$$s.t. = \begin{cases} \sum_{j \in Horse} E_{emp,j} \cdot x_{emp,j} > \sum_{j \in Horse} E_{ij} \cdot x_{ij} \ \forall i \in Court \setminus \{emp\} \\ \sum_{i \in Court} \sum_{j \in Horse} x_{ij} = min(m, n) \\ (R_i - R_j) \cdot (\sum_{k \in Horse} x_{ik} \cdot B_k - \sum_{k \in Horse} x_{jk} \cdot B_k) \geq 0 \ \forall i, j \in Court \\ breakrules = \sum_{i,j \in Horse} \{(S_i - S_j) \cdot (\sum_{k \in Court} x_{ki} \cdot A_k - \sum_{k \in Court} x_{kj} \cdot A_k) > 0 \} \end{cases}$$

二、算法设计与分析

采用遗传算法的思路，构建启发式求解算法，总体流程与普通的遗传算法一致。在具体模块的内部进行定制化的设计。

适应度函数设计

$$Fitness(X) = Obj(X) - 100 * breakrules(X) - 1000 * breakhardrules(X)$$

其中：

$$X = \{x_{ij}\}_{n \times m}$$

$$breakrules(X) = \sum_{i,j \in Horse} \{(S_i - S_j) \cdot (\sum_{k \in Court} x_{ki} \cdot A_k - \sum_{k \in Court} x_{kj} \cdot A_k) > 0 \}$$

$$breakhardrules(X) = \sum_{i,j \in Horse} \{(R_i - R_j) \cdot (\sum_{k \in Horse} x_{ik} \cdot B_k - \sum_{k \in Horse} x_{jk} \cdot B_k) < 0 \}$$

种群生成

生成指派问题的初始可行解

- for all i in $Court \cap Horse$
  - $x_{ij} = 1$;

对初始可行解执行数次变异操作生成种群$Chromosome$，编译过程详见后续

染色体交叉

采用轮盘赌的方法选取两个父代$parentA$，$parentB$，子代为$Offspring$

根据匹配关系，每个子代有以下情况：

$$\forall i \in Court \cup Horse$$
$$\begin{cases} i \in Court, Offspring_i \in Horse \\ i \notin Court, Offspring_i \in Horse \\ i \in Court, Offspring_i \notin Horse \end{cases}$$

其中:

$$i \in Court, parent_i \in Horse \Leftrightarrow x_{ij} = 1$$

> - $Offspring = parentA$
> - $i = Rand() \in Court \cup Horse$
> - **If** i in $Court \cup Horse$ where $parentA_i \neq parentB_i$
>   - **Find** j where $parentA_i = parentB_j$
>   - **Swap** ($Offspring_i$,$Offspring_j$)
> - **End**

染色体变异

针对皇帝约束的变异

> - $i = Rand() \in Court$
> - **If** $i \neq emp$ and $Offspring_i \in Horse$
>   - **Swap** ($Offspring_i$,$Offspring_{emp}$)
> - **End**

针对人员等级和马匹速度约束的变异

> - $i = Rand() \in Court \cup Horse$
> - $j = Rand() \in Court$ where $Offspring_j \in Horse$
> - **If** $i \in Court$ and $parent_i \in Horse$
>   - If $(R_i - R_j) \cdot (\sum_{k \in Horse} x_{ik} \cdot B_k - \sum_{k \in Horse} x_{jk} \cdot B_k) \leq 0$ or $\sum_{k \in Court} x_{ki} \cdot A_k - \sum_{k \in Court} x_{kj} \cdot A_k) \geq 0$
>   - **Swap** ($Offspring_i$,$Offspring_j$)
>   - Endif
> - **Else if** $i \notin Court$ and $parent_i \in Horse$
>   - **Swap** ($Offspring_i$,$Offspring_j$)
> - **Else** $i \in Court$ and $parent_i \notin Horse$
>   - **Swap** ($Offspring_i$,$Offspring_j$)
> - **End**

种群更新

使用生成的子代与种群中最差情况比较，替换最差情况，提取种群中最好情况

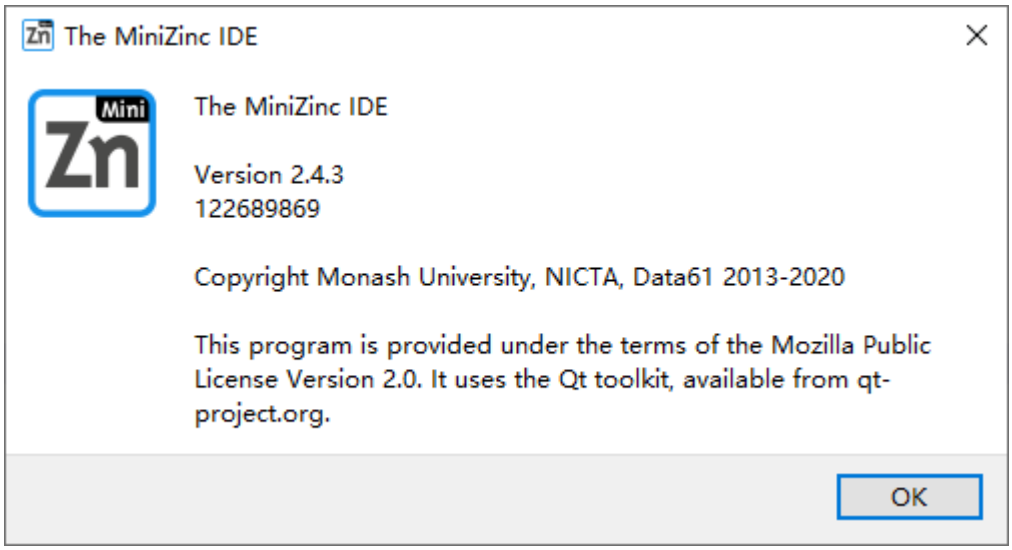
> - $worstone = argmin\{Fitness(X)\ |X \in Chromosome\}$
> - **If** $Fitness(worstone) < Fitness(Offspring)$
>   - **Erase** $worstone$ in $Chromosome$
>   - **Insert** $Offspring$ in $Chromosome$
> - $BestRes = argmax\{Fitness(X)\ |X \in Chromosome\}$
> - **End**

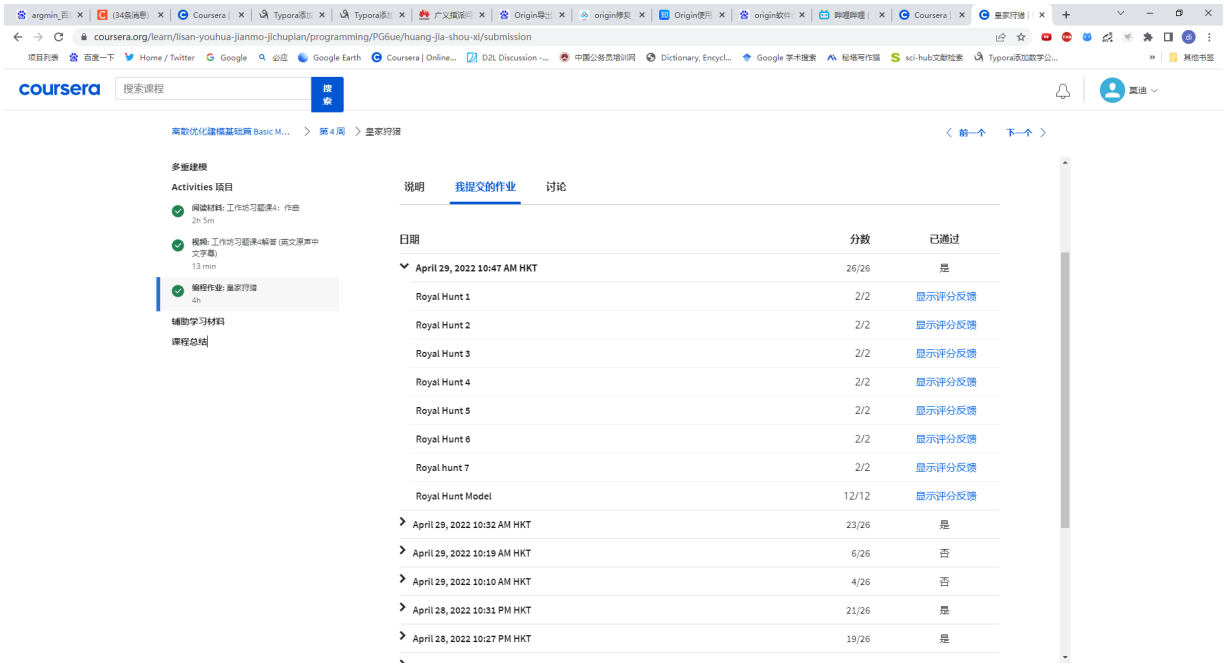
优化点，降低陷入适应度较好的不可行解的机会

> - $worstone = argmin\{Fitness(X)\ |X \in Chromosome\}$
> - **If** $Fitness(worstone) < Fitness(Offspring)$
>
>   and $breakhardrules(worstone) \leq breakhardrules(Offspring)$
>   - **Erase** $worstone$ in $Chromosome$
>   - **Insert** $Offspring$ in $Chromosome$
> - $BestRes = argmax\{Fitness(X)\ |X \in Chromosome\}$
> - **End**

三、结果解释与讨论

使用MiniZinc IDE 2.4.3 对问题进行建模，并在Coursera在线平台验证模型正确性。

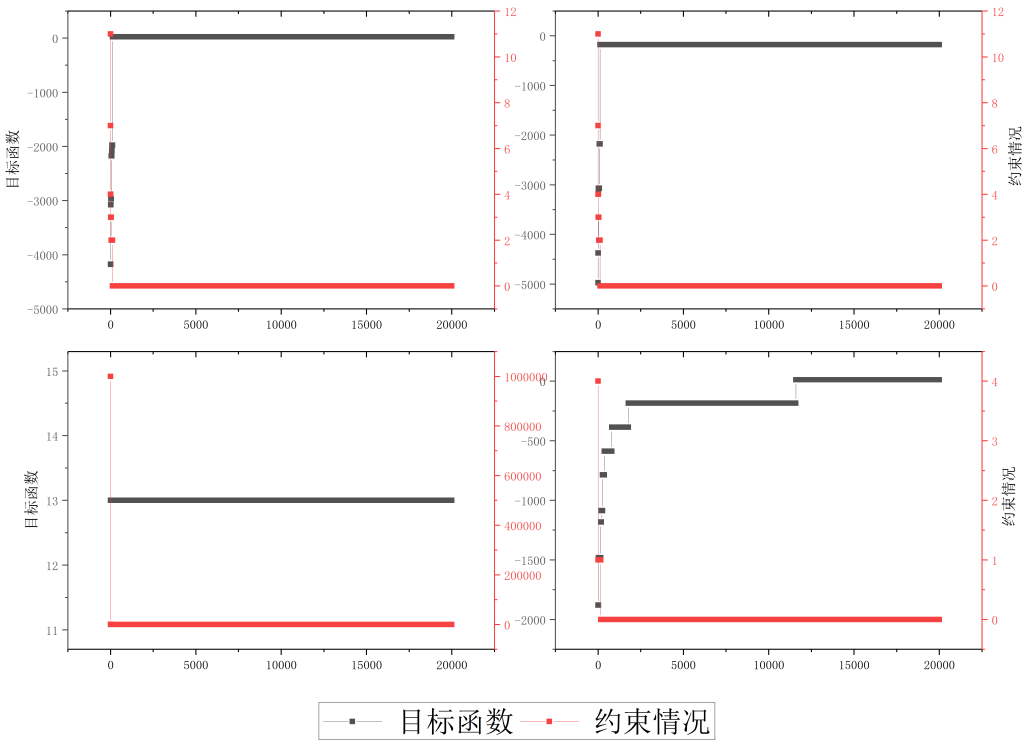




根据Coursera课程平台提供的公开数据获取测试样例，总计10个，样例的规模和在求解器中得到的精确解情况如下表所示：

测试样例	n	m	解	目标函数	运行时间
royalhunt_0.dzn	6	5	[4, 2, 5, 3, 0, 1]	22	713msec
royalhunt_1.dzn	6	5	[4, 2, 5, 3, 0, 1]	-178	722msec
royalhunt_2.dzn	6	4	[4, 1, 2, 3, 0, 0]	13	719msec
royalhunt_3.dzn	4	8	[6, 4, 7, 8]	13	714msec
royalhunt_4.dzn	4	20	[9, 17, 8, 7]	-488	741msec
royalhunt_5.dzn	24	5	UNSATISFIABLE	UNSATISFIABLE	769msec
royalhunt_6.dzn	24	5	[4, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 3, 5, 0, 0]	-580	867msec
royalhunt_7.dzn	24	10	[6, 0, 0, 8, 0, 10, 0, 0, 9, 0, 7, 0, 0, 0, 1, 0, 0, 5, 3, 0, 0, 4, 2, 0]	-666	1h 10m 34s
royalhunt_16.dzn	24	20	-	-	5h未解出
royalhunt_17.dzn	24	20	-	-	5h未解出

以初始种群规模为20，最大迭代次数为20000次，运行各个样例，得到最优解对应函数值和收敛曲线。



附件：优化算法代码

建模代码

```
include "globals.mzn";

int: n; % number of court members
set of int: COURT = 1..n;
int: emperor = 1;
array[COURT] of int: rank;
array[COURT] of int: ability;

int: m; % number of horses
set of int: HORSE = 1..m;
array[HORSE] of int: beauty;
array[HORSE] of int: speed;

array[COURT,HORSE] of int: enjoy;

array[COURT] of var 0..m: horse; % 0->nochoice;
array[HORSE] of var 0..n: court; % 0->nochoice;

constraint forall(i in COURT where horse[i] in HORSE)(
  court[horse[i]] = i
);

constraint forall(i in HORSE where court[i] in COURT)(
  horse[court[i]] = i
```

```
);

constraint horse[emperor] > 0;


constraint forall(i in 2..n where horse[i] in HORSE)(
  enjoy[emperor,horse[emperor]] > enjoy[i,horse[i]]
);

constraint alldifferent_except_0(horse);
constraint alldifferent_except_0(court);

constraint sum(i in COURT)(horse[i]>0) = min(m,n);

constraint forall(i in COURT where horse[i] in HORSE)(enjoy[i,horse[i]] > 0);

constraint forall(i in COURT where horse[i] in HORSE)(
  forall(j in COURT where horse[j] in HORSE /\ rank[i] > rank[j])(
    beauty[horse[i]] >= beauty[horse[j]]
  )
);

constraint forall(i in COURT where horse[i] in HORSE)(
  forall(j in COURT where horse[j] ==0)(
    rank[i] >= rank[j]
  )
);


var int: breakrules;
constraint breakrules=sum(i in HORSE where court[i] in COURT)(
  sum(j in HORSE where speed[i] > speed[j])(
  court[j] == 0 \/ ability[court[i]] < ability[court[j]]
  )
);

var int: obj;
constraint obj=sum(i in COURT where horse[i] in HORSE)
  (enjoy[i,horse[i]]) - 100 * breakrules;

solve maximize obj;

output["horse = ",show(horse),";\nobj = ",show(obj)];
```

遗传算法代码

```
#include<iostream>
#include<algorithm>
#include<fstream>
#include<vector>
```

```cpp
#include<ctime>
#include<unordered_map>
#include<string>

#define cin fin
//#define cout fout
#define INF 1000000
#define Chromosome_Num 100//遗传过程中的群体大小


using namespace std;

const int emperor = 1;

const int breakCost = 20;

int Max_Iter = 30000;//最大遗传迭代次数

string filename = "royalhunt_7";//样例名称

ifstream fin(filename + ".txt", ios::in);
ofstream fout(filename+"out.csv",ios::out);
ofstream fBout("bestRecOut.csv", ios::app);

vector<vector<int>> chromosome;//种群
vector<int> chromo_obj;//对应目标函数值
vector<int> parentA, parentB;//用于杂交的父代
vector<int> Offspring;//子代


//参数
int N, M;
int MTN;
int hbr,temphbr=INF;
vector<int> Rank, Ability, Beauty, Speed;
vector<vector<int>> Enjoy;

bool optBetter = false;

//变量
vector<int> curRes;//初始动态解

int Obj;//目标函数值

int bestObj;//最优目标

vector<int> bestRes;//最优解

void Init();

void Genetic_Construction();

void Genetic_Crossover();
```

```
void Genetic_Mutation(vector<int>& cs);

void Genetic_Update();

void Output(vector<int> cs);

int objCal(vector<int> cs);

void optCurRes();

int main() {

    clock_t startTime, endTime;

    startTime = clock();

    Init();

    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
        << "s" << endl;

    Genetic_Construction();

    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
        << "s" << endl;


    for (int i = 0; i < Max_Iter;i++) {

        //cout <<endl<<"iter "<< i + 1 << endl;

        Genetic_Crossover();//染色体交叉；

        Genetic_Mutation(Offspring);//染色体变异；

        Genetic_Update();//生成下一代群体；

        //cout << "Offspring: ";
        //Output(Offspring);
        //cout << "Obj = " << Obj << ';' << endl;

        //cout << "temp_best: ";
        //Output(bestRes);
        //cout << "bestObj = " << bestObj << ';' << endl;

        fout << i << ',' << bestObj << ',' << hbr << ',' << Obj << ',' << temphbr
<< endl;
    }

    endTime = clock();
    cout << "The run time is: " << (double)(endTime - startTime) / CLOCKS_PER_SEC
        << "s" << endl;
```

```
    cout << endl << "finished!" << endl;

    Output(bestRes);
    cout << "bestObj = " << bestObj << ';' << endl;

    fBout << filename << ',' << bestObj << ',' << temphbr << ','
        << (double)(endTime - startTime) / CLOCKS_PER_SEC << ','
        << Chromosome_Num << ',' << Max_Iter << endl;

    return 0;
}

void Init() {
    cin >> N >> M;
    MTN = (N + 1)*(M + 1);

    int x;
    Rank.push_back(0);
    for (int i = 1; i <= N; i++) {
        cin >> x;
        Rank.push_back(x);
    }
    Ability.push_back(0);
    for (int i = 1; i <= N; i++) {
        cin >> x;
        Ability.push_back(x);
    }

    Beauty.push_back(0);
    for (int i = 1; i <= M; i++) {
        cin >> x;
        Beauty.push_back(x);
    }

    Speed.push_back(0);
    for (int i = 1; i <= M; i++) {
        cin >> x;
        Speed.push_back(x);
    }

    vector<int> temp;
    Enjoy.push_back(temp);
    for (int i = 0; i <= N; i++) {
        temp.clear();
        temp.push_back(0);
        for (int j = 1; j <= M; j++) {
            cin >> x;
            if (x >= 0)temp.push_back(x);
            else temp.push_back(-INF);
        }
        Enjoy.push_back(temp);
    }
```

```
    cout << "finish reading data !" << endl;
}

void Genetic_Construction() {
    bestObj = -INF;
    //初始种群生成
    for (int i = 0; i <= max(M, N); i++) curRes.push_back(i);

    optCurRes();


    for (int i = 0; i < Chromosome_Num; i++) {

        //cout << "Chromosome " << i + 1 << endl;
        chromosome.push_back(curRes);
        chromo_obj.push_back(objCal(curRes));

        //Output(curRes);

        //cout << "Obj = " << chromo_obj[i] << ';' << endl;

        if (chromo_obj[i] > bestObj) {
            bestObj = chromo_obj[i];
            bestRes = chromosome[i];
        }

        Genetic_Mutation(curRes);
    }

    cout << "Constructed !" << endl;
}

void Genetic_Crossover() {
    //cout << "crossover" << endl;
    //用轮盘赌方法从群体中随机选择两个父代

    parentA = chromosome[rand() % (Chromosome_Num)];
    parentB = chromosome[rand() % (Chromosome_Num)];

    //对选取的父代进行杂交得到子代

    Offspring = parentA;
    for (int i = 1; i < parentA.size(); i++) {
        if (i == emperor) continue;
        if (parentA[i] != parentB[i]) {
            if (rand() % 2) {
                for (int j = i + 1; j < parentA.size(); j++) {
                    if (parentA[i] == parentB[j]) {
                        swap(Offspring[i], Offspring[j]);
                        break;
                    }
                }
            }
        }
```

```
    }
}

void Genetic_Mutation(vector<int>& cs) {
    //cout << "mutation" << endl;
    //在一定的概率下，将子代的两个位置互换；

    //cout << cs[emperor] << endl;

    int se, se1, se2, h1, h2;
    se = rand() % N + 1;

    if (se!=emperor && cs[se] <= M){ //&&
        //Enjoy[emperor][cs[emperor]] < Enjoy[se][cs[se]]) {//同时检测皇帝约束
        swap(cs[emperor], cs[se]);
    }

    //cout << "test1" << endl;

    //硬约束

    se1 = rand() % max(M, N) + 1;
    se2 = rand() % N + 1;
    h1 = cs[se1];
    h2 = cs[se2];

    if (se2 == emperor) return;
    if (h2 > M) return;

    //cout <<"se: "<< se1 << ' ' << se2 << endl;
    //cout <<"h: "<< h1 << ' ' << h2 << endl;

    if (se1 <= N && h1 <= M && se2 <= N && h2 <= M) {
        if ((Rank[se1] - Rank[se2]) * (Beauty[h1] - Beauty[h2]) < 0 ||
            (Speed[h1] - Speed[h2]) * (Ability[se1] - Ability[se2]) < 0)
            swap(cs[se1], cs[se2]);
    }
    else if (se1 > N && h1 <= M && se2 <= N && h2 <= M) {//有马无人
        //if(Speed[h1] < Speed[h2])
            swap(cs[se1], cs[se2]);
    }
    else if (se1 <= N && h1 > M && se2 <= N && h2 <= M) {//有人无马
        //if (Rank[se1] > Rank[se2])
            swap(cs[se1], cs[se2]);
    }

    //cout << "se: " << se1 << ' ' << se2 << endl;
    //cout << "h: " << cs[se1] << ' ' << cs[se2] << endl;
}

void Genetic_Update() {


    //更新群体
```

```
    Obj = objCal(Offspring);

    //cout << "update" << endl;

    int MinValue = INF, MinSign;

    for (int i = 0; i < Chromosome_Num; i++) {
        if (chromo_obj[i] < MinValue) {

            MinValue = chromo_obj[i];

            MinSign = i;
        }
    }

    if (MinValue < Obj) {
        chromosome[MinSign] = Offspring;
        if (optBetter && hbr) return;
        if (bestObj < Obj || hbr==0) {
            if(hbr==0) temphbr = hbr;
            bestObj = Obj;
            bestRes = Offspring;
        }
    }
}

void Output(vector<int> cs) {
    for (int i = 1; i <= N;i++) {
        if (i == 1) cout << "horse = [";
        else cout << ", ";
        if (cs[i] <= M)
            cout << cs[i];
        else cout << '0';
        if (i == N) cout << "];" << endl;
    }
}

int objCal(vector<int> cs) {
    int res = 0;

    for (int i = 1; i <= N; i++) {
        if (cs[i] && cs[i] <= M) {
            res += Enjoy[i][cs[i]];
        }
    }



    /*强约束惩罚*/
    /*
    • 皇上需要比其他人都更享受这次狩猎。

    • 除非马的数量不够，否则所有人都应该骑马。
```

```
    • 如果一个要员比另外一个要员阶级更高，那么就应该符合以下任意一种情况：
        （a）等级较高的要员骑的马应该比等级较低的要员骑的马更骏伟或两者同等俊伟；
        （b）等级较低的要员没有骑马；
        （c）两个人都没有骑马。
    */

    int br = 0;
    hbr = 0;

    for (int i = 1; i <= N; i++) {
        if (i == emperor || cs[i] > M) continue;
        if (cs[emperor] > M) hbr++;
        else if (Enjoy[emperor][cs[emperor]] <= Enjoy[i][cs[i]]) hbr++;
    }

    //cout << "test4" << endl;

    for (int i = 1; i <= N; i++) {
        if (cs[i] > M) continue;
        for (int j = 1; j <= N; j++) {
            if (cs[j] > M) {
                if (Rank[i] < Rank[j]) hbr++;
            }
            else {
                if ((Rank[i] - Rank[j]) * (Beauty[cs[i]] - Beauty[cs[j]]) < 0)
hbr++;
            }
        }
    }




    /*弱约束惩罚*/
    /*
    • 如果一匹马比另外一匹马更快，那么就应该符合以下任意一种情况：
        （a）骑快马的人的骑马技术不应该比骑慢马的人的更差；
        （b）没有人骑快马；
        （c）两匹马都没有人骑。
    */

    vector<int> court;
    for (int i = 0; i <= max(M, N); i++) court.push_back(0);
    for (int i = 1; i <= max(M, N); i++) court[cs[i]] = i;



    for (int i = 1; i <= M; i++) {
        if (court[i] > N) continue;
        for (int j = 1; j <= M; j++) {
            if (Speed[i] > Speed[j] && (court[j]>N || Ability[court[i]] <
Ability[court[j]])) {
                br++;
            }
```

```
        }
    }

    res -= br * breakCost;

    res -= hbr * breakCost * 10;

    if (!hbr) optBetter = true;

    //cout << "hbr = " << hbr << endl;

    return res;
}

void optCurRes() {
    int a, b, maxenjoy = -INF,temp;
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= M; j++) {
            if (maxenjoy < Enjoy[i][j]) {
                maxenjoy = Enjoy[i][j];
                a = i;
                b = j;
            }
        }
    }

    swap(curRes[emperor], curRes[a]);
}
```