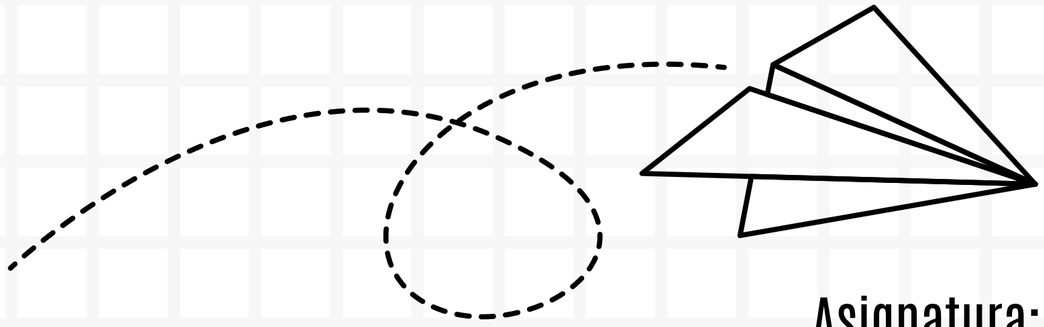

Asignatura: Inteligencia Artificial

# AiRCRAFT LANDiNG SCHELDiNG PROBLEM

Profesor: Nicolás Rojas
Ayudante: Joaquín Gatica
Alumna: Vania Gallardo

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO
DE INFORMÁTICA

# CONTENIDO

# RESTRICCIONES

## 1. Restricción para que cada avión aterrice dentro de su ventana de tiempo.

$$Ei \leq xi \leq Li, \quad \forall i \in \{1, ..., p\}$$

Se creará una función que tenga variable booleana factible = false; esta probará que los tiempos de cada avión estén dentro del rango, si todos estan dentro cambiara la variable a true y la retorna.

## 2. Todos los aviones deben aterrizar.

Al generar la solución inicial mediante greedy deben estar todos aviones dentro de un vector, a la hora de aplicar Tabu Search esta restricción siempre se cumplirá.

## 3. Se debe respetar la separación entre aviones

Al generar la solución inicial se respetará la separación entre aviones desde un comienzo, si al hacer iteraciones la separación se incumple existirá una función con la misma mecánica que la restricción 1.

# PARAMETROS ✈

```
// Abstracción del problema
struct RepresentacionALSP {
    string file_name;
    float **S;
    vector<int> Ei;
    vector<int> Ti;
    vector<int> Li;
    vector<float> gi;
    vector<float> hi;
    unsigned int N;
};

struct SolucionCandidata {
    vector<int> avion;  // Aviones en orden
    vector<int> tiempo;  // Tiempos de aterrizaje correspondientes
};
```

**La asignación de los parámetros de RepresentacionALSP será mediante la lectura de la instancia.**

# LECTURA INSTANCIA

```cpp
int main() {
    // Nombre del archivo de entrada
    string inputFileName = "airland1.txt";

    ifstream inputFile(inputFileName);
    if (!inputFile.is_open()) {
        cerr << "No se pudo abrir el archivo 'airland1.txt'." << endl;
        return 1;
    }

    int N;
    inputFile >> N;

    // Instanciacion de variables para leer
    vector<int> Ei(N);
    vector<int> Ti(N);
    vector<int> Li(N);
    vector<float> gi(N);
    vector<float> hi(N);
    float** S = new float*[N];
```

```cpp
// Se leen las variables desde el archivo
for (int i = 0; i < N; i++) {
    inputFile >> Ei[i];
    inputFile >> Ti[i];
    inputFile >> Li[i];
    inputFile >> gi[i];
    inputFile >> hi[i];
    S[i] = new float[N];
    for (int j = 0; j < N; j++) {
        inputFile >> S[i][j];
    }
}

// Asignacion variables del problema
RepresentacionALSP ALSP;
ALSP.N = N;
ALSP.Ei = Ei;
ALSP.Ti = Ti;
ALSP.Li = Li;
ALSP.gi = gi;
ALSP.hi = hi;
ALSP.S = S;
ALSP.file_name = inputFileName;
```

# LECTURA INSTANCIA ✓

output > airland1.txt

```
1   10
2   129 155 559 10.00 10.00
3   -1 3 15 15 15 15 15 15 15 15
4   195 258 744 10.00 10.00
5   3 -1 15 15 15 15 15 15 15 15
6   89 98 510 30.00 30.00
7   15 15 -1 8 8 8 8 8 8 8
8   96 106 521 30.00 30.00
9   15 15 8 -1 8 8 8 8 8 8
10  110 123 555 30.00 30.00
11  15 15 8 8 -1 8 8 8 8 8
12  120 135 576 30.00 30.00
13  15 15 8 8 8 -1 8 8 8 8
14  124 138 577 30.00 30.00
15  15 15 8 8 8 8 -1 8 8 8
16  126 140 573 30.00 30.00
17  15 15 8 8 8 8 8 -1 8 8
```

```
PS C:\Users\Vannia\Desktop\Proyecto ALSP\output> & .\'ALSP2.exe'
Numero de aviones: 10
Tiempos mas tempranos de aterrizaje: 129 195 89 96 110 120 124 126 135 160
Tiempos ideales de aterrizaje: 155 258 98 106 123 135 138 140 150 180
Tiempos mas tardio de aterrizaje: 559 744 510 521 555 576 577 573 591 657
Penalizacion aterrizar antes: 10 10 30 30 30 30 30 30 30 30
Penalizacion aterrizar despues: 10 10 30 30 30 30 30 30 30 30
Matriz de separacion:
-1 3 15 15 15 15 15 15 15 15
3 -1 15 15 15 15 15 15 15 15
15 15 -1 8 8 8 8 8 8 8
15 15 8 -1 8 8 8 8 8 8
15 15 8 8 -1 8 8 8 8 8
15 15 8 8 8 -1 8 8 8 8
15 15 8 8 8 8 -1 8 8 8
15 15 8 8 8 8 8 -1 8 8
15 15 8 8 8 8 8 8 -1 8
15 15 8 8 8 8 8 8 8 -1
```

# GENERAR SOLUCIÓN INICIAL

```cpp
SolucionCandidata GenerarSolucionInicialGreedy(const RepresentacionALSP& ALSP) {
    unsigned int N = ALSP.N;

    // Inicializar listas de aviones y tiempos
    vector<int> aviones(N);
    vector<int> tiempos(N);

    for (unsigned int i = 0; i < N; i++) {
        aviones[i] = i;
    }

    // Ordenar los aviones por sus tiempos ideales de aterrizaje
    sort(aviones.begin(), aviones.end(), [&](int a, int b) {
        return ALSP.Ti[a] < ALSP.Ti[b];
    });
```

```cpp
    // Asignar tiempos de aterrizaje de manera determinista
    tiempos[0] = ALSP.Ti[aviones[0]];
    for (unsigned int i = 1; i < N; i++) {
        int avion_i = aviones[i];
        int avion_prev = aviones[i - 1];
        int separacion_minima = ALSP.S[avion_i][avion_prev];
        tiempos[i] = max(tiempos[i - 1] + separacion_minima, ALSP.Ti[avion_i]);
    }

    // Crear una solución candidata factible
    SolucionCandidata solucion;
    solucion.avion = aviones;
    solucion.tiempo = tiempos;

    return solucion;
}
```

# GENERAR SOLUCIÓN INICIAL

output > airland1.txt

```
1   10
2   129 155 559 10.00 10.00
3   -1 3 15 15 15 15 15 15 15 15
4   195 258 744 10.00 10.00
5   3 -1 15 15 15 15 15 15 15 15
6   89 98 510 30.00 30.00
7   15 15 -1 8 8 8 8 8 8 8
8   96 106 521 30.00 30.00
9   15 15 8 -1 8 8 8 8 8 8
10  110 123 555 30.00 30.00
11  15 15 8 8 -1 8 8 8 8 8
12  120 135 576 30.00 30.00
13  15 15 8 8 8 -1 8 8 8 8
14  124 138 577 30.00 30.00
15  15 15 8 8 8 8 -1 8 8 8
16  126 140 573 30.00 30.00
17  15 15 8 8 8 8 8 -1 8 8
```

```
3       1       2       4

98      115     258     106

3       4       1        2
98      106     115     258

3       4       5       6       7
98      106     123     135     138

3       4       5       6       7       8
98      106     123     135     143     140

5
3       4       5       6       7       8       9
98      106     123     135     143     151     150

3       4       5       6       7       8       9       1
98      106     123     135     143     151     159     155

3       4       5       6       7       8       9       1       10
98      106     123     135     143     151     159     174     180

3       4       5       6       7       8       9       1       10      2
98      106     123     135     143     151     159     174     189     258
```

```
PS C:\Users\Vannia\Desktop\Proyecto ALSP\output> & .\'ALSP2.exe'
Solucion Inicial:
 Aviones: 3 4 5 6 7 8 9 1 10 2
Tiempos: 98 106 123 135 143 151 159 174 189 258
```

# FUNCIÓN DE EVALUACIÓN

La función objetivo es minimizar los costos de aterrizaje. $$min \sum_{i=1}^{P} (g_i \alpha_i + h_i \beta_i)$$

```
int evaluarCosto(RepresentacionALSP ALSP, vector<SolucionCandidata2> solucion_candidata, int mejor_solucion) {
    int costo_candidato = 0;

    if (solucion_candidata.size() == ALSP.N) {
        for (unsigned int i = 0; i < solucion_candidata.size(); i++)  {
            int avion = solucion_candidata[i].avion[0];  // Acceder al primer elemento del vector

            // Accede a los valores correspondientes en los vectores Ti, gi y hi
            int tiempo_aterrizaje = solucion_candidata[i].tiempo[avion] - ALSP.Ti[avion];

            if (tiempo_aterrizaje < 0) {
                costo_candidato += -tiempo_aterrizaje * ALSP.gi.at(avion);
            } else if (tiempo_aterrizaje > 0) {
                costo_candidato += tiempo_aterrizaje * ALSP.hi.at(avion);
            }
        }

        if (costo_candidato < mejor_solucion) {
            return costo_candidato;
        }
    }
    return -1;
}
```

# TABU SEARCH

1. Solución Inicial. *Generada por el algoritmo greedy mencionado.*
2. Evaluación de Vecindad. *Vecindad se genera con movimiento y se evalúan sus costos.*
3. Función de Evaluación. *Se deben tener en cuenta las restricciones.*
4. Criterio de Vecindad. *Ve modificaciones validas o prohibidas según lista tabú.*
5. Lista Tabú. *Registra los movimientos recientes y prohíbe volver a realizarlos. (iteraciones)*
6. Búsqueda Iterativa. *Puede ser numero de iteraciones, tiempo o solución satisfactoria.*
7. Criterio de Parada. *Puede ser numero de iteraciones, tiempo o solución satisfactoria.*
8. Resultado. *Mejor solución que a encontrado en el proceso.*

# MOVIMIENTO SWAP

1. Diversificación de la búsqueda.
2. Exploración de vecindario.
3. Respeta restricciones.
4. Escape de óptimos locales.
5. Eficiencia en la implementación.

# Conclusiones

- El enfoque propuesto para abordar el problema del ALSP demuestra ser una solución efectiva y eficiente.
- Las restricciones clave del problema se cumplen o serán cumplidas.
- Mediante la lectura de la instancia y la generación de una solución inicial factible, se sientan las bases para la posterior aplicación del algoritmo Tabu Search.

# TRABAJO POR REALIZAR

- Realizar función para restricción que cada avión aterrice dentro de su ventana de tiempo.
- Implementación reloj para medir tiempo de ejecución algoritmo.
- Implementar algoritmo Tabu Search.