

# Datos del estudiante

Nombre y apellidos	Miguel Chiara Aguilar-Amat
Fecha de entrega	05/01/2026

## Actividad 2. Planificación multiobjetivo de rutas de drones en entornos urbanos

### Indice

Resumen.....	2
Repositorio.....	2
Branch-and-Bound.....	2
código y explicación.....	2
Algoritmo geométrico.....	4
código y explicación.....	4
Metaheurística aleatorizada.....	5
código y explicación.....	5
Graficas.....	8

# Resumen

En esta actividad vamos a resolver un problema usando 3 algoritmos distintos: Recursividad directa, Estrategia divide y conquista de Hirschberg y Programación dinámica tabular, para poder ejecutar el código de esta actividad será necesario tener python instalado con una versión Python 3.11.9 o superior

python[<https://www.python.org/downloads/>]

# Repositorio

Todo lo visto en esta actividad se encuentra subido en el repositorio:

[https://github.com/MiguelChiara/Disenio\\_Avanzado\\_de\\_Algoritmos\\_ACT2](https://github.com/MiguelChiara/Disenio_Avanzado_de_Algoritmos_ACT2)

# Branch-and-Bound

código y explicación

```
from itertools import permutations

def domina(a, b):
    return (
        a["distancia"] <= b["distancia"] and
        a["riesgo"] <= b["riesgo"] and
        a["recargas"] <= b["recargas"] and
        (a["distancia"], a["riesgo"], a["recargas"]) !=
        (b["distancia"], b["riesgo"], b["recargas"])
    )

def es_dominada(sol, frente):
    return any(domina(f, sol) for f in frente)

# Algoritmo de Branch and Bound para encontrar la ruta optima
def branch_and_bound(instancia):
    # Extraer datos de la instancia
    nodos = instancia["nodes"]
    hub = instancia["hub"]
    destinos = [n["id"] for n in nodos if n["id"] != hub]
```

```

mejor_frente = []

# Generar todas las permutaciones posibles de destinos
for perm in permutations(destinos):
    ruta = [hub] + list(perm) + [hub]
    evaluacion = instancia["evaluar_ruta"](ruta)
    # Si la ruta no es valida, continuar
    if evaluacion is None:
        continue

    sol = {
        "ruta": ruta,
        **evaluacion
    }

    if not es_dominada(sol, mejor_frente):
        mejor_frente = [
            f for f in mejor_frente if not domina(sol, f)
        ]
        mejor_frente.append(sol)

return mejor_frente

def resolver(instancia):
    return branch_and_bound(instancia)

```

El algoritmo busca la ruta mas optima, esta ruta es una permutación de N destinos mas el hub, para N destino existen  $N!$  Rutas posibles por eso mismo he optado por crear un archivo json mas liviano para probar esta casuistica.

$N=10! \rightarrow 3628800$

$N=15! \rightarrow 1307674368000$

$N=20! \rightarrow 2432902008176640000$

$N=25! \rightarrow 15511210043330985984000000$

Como se puede ver es inviable por el tamaño de los numero con los que estamos trabajando, Cada nodo explorado evalua la ruta y valcula distancia, riesgo, recargas, evitando zonas no-fly

# Algoritmo geométrico

## código y explicación

```
# implementacion de un algoritmo para la heuristica geo
def greedy(instancia):
    # Extraer datos de la instancia
    hub = instancia["hub"]
    nodos = instancia["nodes"]
    evaluar = instancia["evaluar_ruta"]

    soluciones = []

    # Probar diferentes valores de alpha para combinar distancia y riesgo
    for alpha in [0.2, 0.5, 0.8]:
        no_visitados = {n["id"] for n in nodos if n["id"] != hub}
        ruta = [hub]
        # Construir la ruta
        while no_visitados:
            mejor = None
            mejor_coste = float("inf")
            # Evaluar cada candidato no visitado
            for candidato in no_visitados:
                eval_parcial = evaluar(ruta + [candidato])
                if eval_parcial is None:
                    continue

                coste = (
                    alpha * eval_parcial["distancia"] +
                    (1 - alpha) * eval_parcial["riesgo"]
                )

                if coste < mejor_coste:
                    mejor = candidato
                    mejor_coste = coste
            # Si no hay candidato valido, terminar la construccion
            if mejor is None:
                break
            # Anadir el mejor candidato a la ruta
            ruta.append(mejor)
            no_visitados.remove(mejor)
```

```

ruta.append(hub)
evaluacion = evaluar(ruta)
# Si la ruta es valida, guardarla
if evaluacion:
    soluciones.append({
        "ruta": ruta,
        **evaluacion
    })

return soluciones

def resolver(instancia):
    return greedy(instancia)

```

El algoritmo geométrico construye una ruta de forma incremental, buscando el destino no visitado, para minimizar el coste. Aun que no asegura la forma optima global, es mas optimo que el algoritmo de Branch-and-Bound

## Metaheurística aleatorizada

código y explicación

```

import random

# Declaracion de constantes
POP_SIZE = 4 # Tamaño de la poblacion
GENERATIONS = 5 # Numero de generaciones
MUT_RATE = 0.2 #probabilidad de mutar una ruta
MAX_INTENTOS = 1000 * POP_SIZE #maximo numero de intentos para crear la
poblacion inicial

# Algoritmo NSGA-II
def nsga2(instancia):
    #Preparacion de datos
    hub = instancia["hub"]
    nodos = [n["id"] for n in instancia["nodes"] if n["id"] != hub]
    evaluar = instancia["evaluar_ruta"]
    # Un individuo es una ruta completa devuelve un objeto tal que [hub, n1, n2,

```

```

n3, ..., hub]

def crear_individuo():
    perm = nodos[:]
    random.shuffle(perm)
    return [hub] + perm + [hub]

# Evalua un individuo y devuelve su evaluacion junto con la ruta
def evaluar_ind(ruta):
    ev = evaluar(ruta)
    if ev is None:
        return None
    return {"ruta": ruta, **ev}

poblacion = []
intentos = 0

# Crear poblacion inicial
while len(poblacion) < POP_SIZE and intentos < MAX_INTENTOS:
    r = crear_individuo()
    ev = evaluar_ind(r)
    if ev:
        poblacion.append(ev)
    intentos += 1

# Evolucion de la poblacion
for _ in range(GENERATIONS):
    hijos = []
    if poblacion != []:
        # Crear hijos mediante mutacion
        while len(hijos) < POP_SIZE:
            p = random.choice(poblacion)["ruta"][:]
            if random.random() < MUT_RATE:
                i, j = random.sample(range(1, len(p)-1), 2)
                p[i], p[j] = p[j], p[i]

            ev = evaluar_ind(p)
            if ev:
                hijos.append(ev)

        poblacion.extend(hijos)
        poblacion = seleccionar_pareto(poblacion, POP_SIZE)

```

```

    return poblacion

def domina(a, b):
    return (
        a["distancia"] <= b["distancia"] and
        a["riesgo"] <= b["riesgo"] and
        a["recargas"] <= b["recargas"] and
        (a["distancia"], a["riesgo"], a["recargas"]) !=
        (b["distancia"], b["riesgo"], b["recargas"])
    )

def seleccionar_pareto(poblacion, k):
    frente = []
    for p in poblacion:
        if not any(domina(o, p) for o in poblacion):
            frente.append(p)

    if len(frente) >= k:
        return frente[:k]

    resto = [p for p in poblacion if p not in frente]
    return frente + resto[:k - len(frente)]

def resolver(instancia):
    return nsga2(instancia)

```

Este algoritmo permite obtener un conjunto de soluciones, cada individuo de la población representa una ruta válida que comienza y termina en el hub,

El problema que presenta este algoritmo al igual que los otros es que no existe una solución que mejore simultáneamente todos los objetivos, (distancia, riesgo, número de recargas), cada punto representa un compromiso distinto entre los objetivos.

# Graficas

Las graficas han sido generadas en codigo ascii y python nativo para poder visualizar los resultados en tiempo real

```
PS C:\UNIPRO\2025-2026\Algoritmos avanzados\ACT2\workspace> python .\main.py .\instances\intanciaN25.json

RESULTADOS
-----
No se puede calcular Exact_BB tiene demasiados nodos | 0.0000s | 0.0 KB | 0 soluciones | 0.0 hipervolumen | 0.0000 diversidad
Greedy          | 0.1170s | 10.8 KB | 3 soluciones | 3.42285344334821 hipervolumen | 0.0000 diversidad
NSGA2           | 0.3706s | 9.6 KB | 4 soluciones | 0.33215045090710055 hipervolumen | 1.8864 diversidad

TIEMPO VS N (log-scale conceptual)
No se puede calcular Exact_BB tiene demasiados nodos
N=25 tiempo=0.000000s
Greedy
N=25 tiempo=0.116959s
NSGA2
N=25 tiempo=0.370619s
```