UNIVERSIDAD DIEGO PORTALES

FACULTAD DE INGENIERÍA Y CIENCIAS

ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Algoritmos Exactos y Metaheurísticas

Tarea 2

Profesor:
Víctor Reyes

Estudiante:
Felipe Méndez

Santiago, Chile 10 de mayo del 2025

Índice

1. Introducción

Este informe aborda el problema de optimizar la secuencia de aterrizajes de aviones en un aeropuerto, con el objetivo de minimizar los costos asociados a desviaciones de los tiempos de aterrizaje preferidos, respetando restricciones de seguridad como los tiempos mínimos de separación de aterrizaje entre aviones.

El problema consiste en programar un conjunto de D aviones, cada uno con un tiempo de aterrizaje más temprano E_k, un tiempo preferente P_k y un tiempo más tardío L_k, junto con penalizaciones por aterrizar antes (C_k) o después (C'_k) del tiempo preferente. Además, se debe cumplir un tiempo mínimo de separación τ_{ij} entre aterrizajes consecutivos para satisfacer requisitos operativos y de seguridad.

Para resolver esta problemática, se diseñaron e implementaron ciertos algoritmos. Entre ellos:

- Un greedy determinista, que construye una secuencia de aterrizaje seleccionando iterativamente el avión con el menor costo, dando siempre el mismo resultado (propiedad del método determinista).
- Un greedy estocástico, que introduce aleatoriedad para explorar soluciones diversas.
- Un GRASP (Greedy Randomized Adaptive Search Procedure), utilizando el algoritmo Hill-Climbing utilizando la estrategia de alguna-mejora. Este algoritmo utiliza las soluciones iniciales de ambos greedy e incorpora reinicios en el caso estocástico para mejorar la exploración del espacio de soluciones.
- Una metaheurística de Tabu Search, que refina las soluciones anteriores mediante múltiples configuraciones de parámetros para obtener resultados de alta calidad.

Si bien existen 4 archivos que representan casos de prueba (`case1.txt`, `case2.txt`, `case3.txt` y `case4.txt`), este informe se centra en las soluciones del caso 1 (`case1.txt`) y además, se asume que existe una única pista de aterrizaje.

2. Importar Casos

Es importante mencionar en cuanto a la estructura, que esta será modular, esto quiere decir, que se separarán los códigos en archivos distintos para obtener mayor rendimiento, mantenimiento y comodidad a la hora de trabajar. Dicho esto, se crea un archivo llamado leer_data.py, el cual se encarga de leer los datos de los archivos proporcionados (case1.txt, case2.txt, etc).

2.1. Pseudocódigo

Función `leer_datos_archivo`

- **1.** Abrir y leer todas las líneas del archivo, limpiándolas:
  - Crear lista vacía `lineas_limpias`
  - Abrir el archivo ubicado en `ruta_del_archivo` en modo lectura
  - Para cada `linea` en el archivo hacer:
    - `linea_procesada` ← quitar espacios al inicio y final de `linea`
    - Si `longitud(linea_procesada) >0`, entonces añadir `linea_procesada` a `lineas_limpias`
  - Cerrar archivo
- **2.** Leer el número total de aviones:
  - D ← convertir a entero(`lineas_limpias[0]`)
- **3.** Inicializar estructuras de datos:
  - `lista_aviones` ← lista vacía
  - `matriz_tiempos_separacion` ← lista vacía
- **4.** Inicializar índice para recorrer líneas:
  - `indice_linea_actual` ← 1
- **5.** Repetir D veces:
  - Leer y dividir `lineas_limpias[indice_linea_actual]` por espacios:
    - $E, P, L, C1, C2$ ← datos de la línea
  - Crear objeto `avion_actual` con los campos:
    - `tiempo_temprano` ← entero(E)
    - `tiempo_preferido` ← entero(P)
    - `tiempo_tarde` ← entero(L)
    - `penalizacion_temprano` ← flotante($C1$)
    - `penalizacion_tarde` ← flotante($C2$)
  - Añadir `avion_actual` a `lista_aviones`
  - Leer `lineas_limpias[indice_linea_actual+1]` y `lineas_limpias[indice_linea_actual+2]`:
    - Dividir ambas por espacios y convertir a enteros
    - Unir ambas listas en `fila_separacion_actual`
    - Añadir `fila_separacion_actual` a `matriz_tiempos_separacion`
  - Actualizar índice: `indice_linea_actual += 3`
- **6.** Retornar:
  - D, `lista_aviones`, `matriz_tiempos_separacion`

3. Diseño e implementación de un greedy determinista

3.1. Pseudocódigo

Función `greedy_determinista`

1. Inicializar:
   - `secuencia_indices_aterrizaje` ← lista vacía
   - `tiempos_programados_aterrizaje` ← array de tamaño D con ceros
   - `costo_total_acumulado` ← 0
2. Repetir D veces:
   a) Establecer:
   - `mejor_indice_avion_actual` ← -1
   - `menor_costo_iteracion` ← ∞
   - `tiempo_aterrizaje_para_mejor_avion` ← -1

   b) Para cada i en 0 hasta $D-1$, si i no está en `secuencia_indices_aterrizaje`:
   1) Calcular el tiempo factible de aterrizaje:
   $$t \leftarrow \texttt{lista\_aviones}[i].\texttt{tiempo\_temprano}$$
   2) Para cada j en `secuencia_indices_aterrizaje`:
   $$t \leftarrow \max(t, \texttt{tiempos\_programados\_aterrizaje}[j] + \texttt{matriz\_tiempos\_separacion}[j][i])$$
   3) Si $t \leq$ `lista_aviones`[i].`tiempo_tarde`:
   a′) Calcular el costo:
   $$c \leftarrow \begin{cases} \texttt{penalizacion\_temprano} \cdot (\texttt{tiempo\_preferido} - t) & \text{si } t < \texttt{tiempo\_preferido} \\ \texttt{penalizacion\_tarde} \cdot (t - \texttt{tiempo\_preferido}) & \text{si } t > \texttt{tiempo\_preferido} \\ 0 & \text{si } t = \texttt{tiempo\_preferido} \end{cases}$$
   b′) Si $c <$ `menor_costo_iteracion`, actualizar:
   - `mejor_indice_avion_actual` ← i
   - `tiempo_aterrizaje_para_mejor_avion` ← t
   - `menor_costo_iteracion` ← c

   c) Si no se encontró un avión factible:
   1) Para cada i no programado:
   - Repetir cálculo de t como antes
   - Limitar $t \leftarrow \min(t, \texttt{tiempo\_tarde})$
   - Calcular costo c como antes
   - Asignar este avión como el mejor y romper el bucle

   d) Actualizar:
   - Añadir `mejor_indice_avion_actual` a la secuencia
   - Asignar t a su tiempo de aterrizaje
   - Incrementar `costo_total_acumulado`
3. Construir `tiempos_aterrizaje_final_ordenados` recorriendo la secuencia y extrayendo los tiempos programados.
4. Retornar:
$$(\texttt{secuencia\_indices\_aterrizaje}, \texttt{costo\_total\_acumulado}, \texttt{tiempos\_aterrizaje\_final\_ordenados})$$

3.2. Funcionamiento

Se implementa la función `greedy_determinista` para secuenciar el aterrizaje de D aviones, con el objetivo de minimizar el costo total basado en penalizaciones por aterrizar antes o después del tiempo preferido de cada avión.

1. **Iteración por Iteración:** El algoritmo construye la secuencia de aterrizaje avión por avión. En cada una de las D iteraciones, selecciona el próximo avión que aterrizará.
2. **Selección del Mejor Próximo Avión:**
   - Para cada avión aún no programado, se calcula su *tiempo factible de aterrizaje más temprano*, definido como el máximo entre:
     a) El tiempo mínimo permitido de aterrizaje (t_{temprano}), y el tiempo de aterrizaje de cada avión ya programado más el tiempo de separación requerido entre ese avión y el avión actual.
   - Si este tiempo factible es menor o igual al tiempo máximo permitido de aterrizaje del avión (t_{tarde}), se calcula el *costo de penalización*, determinado por:
     - Penalización por adelanto: si aterriza antes de su tiempo preferido.
     - Penalización por atraso: si aterriza después de su tiempo preferido.
   - Se elige el avión que minimiza el costo de penalización cumpliendo la restricción del tiempo máximo.
3. **Manejo de Casos Sin Solución Ideal (Fallback):**
   - Si ningún avión puede aterrizar respetando su t_{tarde}, se entra en un modo de *fallback*.
   - Se calcula nuevamente el tiempo factible de aterrizaje y se fuerza a que no sea mayor que t_{tarde}, incluso si se llega demasiado tarde.
   - Se selecciona el primer avión disponible bajo esta lógica forzada.
4. **Actualización y Salida:**
   - El avión seleccionado se añade a la secuencia.
   - Se registra su tiempo de aterrizaje y su costo asociado.
   - Se repite hasta que todos los aviones están programados.
   - La función retorna:
     - la secuencia de aterrizaje (índices de aviones en orden), el costo total acumulado y los tiempos programados de aterrizaje en ese orden.
5. **Determinismo:** Siempre genera la misma salida, ya que no utiliza ninguna técnica aleatoria.

3.3. Resultados

Método	Greedy Determinista
Secuencia	[0, 9, 13, 1, 11, 14, 10, 12, 8, 6, 5, 7, 4, 3, 2]
Costo	51660.0
Tiempos de Aterrizaje	[129, 144, 152, 190, 251, 276, 279, 294, 302, 310, 318, 326, 334, 342, 350]

Cuadro 1: Resultados del algoritmo Greedy Determinista

3.4. Análisis

El algoritmo Greedy Determinista implementado construye una solución de forma secuencial y directa, seleccionando en cada paso el avión que resulta en el menor costo inmediato, considerando las restricciones de tiempo de aterrizaje (temprano, preferido, tardío) y separación. Para el `case1.txt`, se obtuvo un costo total de **51660.0**.

- **Fortalezas:** Es un buen punto de partida para obtener una solución factible rápidamente.

- **Debilidades:** El algoritmo toma decisiones óptimas a nivel local, sin considerar el impacto global de estas decisiones en la secuencia completa. Esto puede llevar a soluciones subóptimas, ya que una elección temprana que parece buena podría restringir severamente las opciones para aviones posteriores, acumulando un costo mayor al final. No posee mecanismos de exploración del espacio de soluciones.
- **Cumplimiento:** Satisface las restricciones del problema, incluyendo el manejo del `fallback` para asegurar que todos los aviones sean programados.

Este algoritmo sirve como una base de comparación importante para las heurísticas que se explorarán posteriormente.

4. Diseño e implementación de un greedy estocástico

4.1. Pseudocódigo

Función `greedy_estocastico`

- **1. Inicialización**
  - Inicializar generador aleatorio con `semilla_aleatoria`
  - `secuencia_indices_aterrizaje` ← lista vacía
  - `tiempos_programados_aterrizaje` ← arreglo de tamaño D con ceros
  - `costo_total_acumulado` ← 0
- **2. Repetir D veces (una por avión a programar):**
  - Inicializar listas vacías para los candidatos actuales:
    - `lista_indices_candidatos_actual`
    - `lista_costos_candidatos_actual`
    - `lista_tiempos_factibles_candidatos_actual`
  - Para cada `indice_candidato` desde 0 hasta $D-1$:
    - Si `indice_candidato` no está en `secuencia_indices_aterrizaje`:
      - `tiempo_factible_aterrizaje` ← `lista_aviones[indice_candidato].tiempo_temprano`
      - Para cada `indice_avion_previo` en `secuencia_indices_aterrizaje`:
      - `tiempo_min_por_separacion` ← `tiempos_programados_aterrizaje[indice_avion_previo]` + `matriz_tiempos_separacion[indice_avion_previo][indice_candidato]`
      - `tiempo_factible_aterrizaje` ← `máximo entre tiempo_factible_aterrizaje y tiempo_min_por_separacion`
      - Si `tiempo_factible_aterrizaje` $\leq$ `lista_aviones[indice_candidato].tiempo_tarde`:
      - `costo_actual` ← 0
      - Si `tiempo_factible_aterrizaje` $<$ `tiempo_preferido` entonces:
      - Sumar penalización temprana correspondiente
      - Si `tiempo_factible_aterrizaje` $>$ `tiempo_preferido` entonces:
      - Sumar penalización tardía correspondiente
      - Añadir a las listas de candidatos: índice, costo y tiempo factible
  - Si no se encontraron candidatos (lista vacía):
    - Para cada `indice_candidato` no programado aún:
      - Calcular `tiempo_factible_aterrizaje` igual que antes
      - Ajustar `tiempo_factible_aterrizaje` a no superar el tiempo tarde
      - Calcular el `costo_actual` como antes
      - Asignar ese candidato como único candidato (listas con un solo elemento)
      - Romper el bucle
  - Seleccionar un candidato aleatoriamente:
    - `indice_aleatorio` ← entero aleatorio entre 0 y longitud de lista de candidatos -1
    - Seleccionar el índice, costo y tiempo correspondiente al índice aleatorio
  - Actualizar resultados:
    - Añadir índice seleccionado a `secuencia_indices_aterrizaje`
    - Guardar su tiempo de aterrizaje en `tiempos_programados_aterrizaje`
    - Sumar su costo al `costo_total_acumulado`

- **3. Preparar la salida final:**
  - `secuencia_aterrizaje_final_ordenada` ← `secuencia_indices_aterrizaje`
  - `tiempos_aterrizaje_final_ordenados` ← lista vacía
  - Para cada `indice_avion` en la secuencia:
    - Añadir `tiempos_programados_aterrizaje[indice_avion]` a la lista
- **4. Retornar:**
  - `secuencia_aterrizaje_final_ordenada`
  - `costo_total_acumulado`
  - `tiempos_aterrizaje_final_ordenados`

4.2. Funcionamiento

La función `greedy_estocastico` es una variante del algoritmo greedy que introduce aleatoriedad en la selección del próximo avión a aterrizar, generando diferentes secuencias de aterrizaje al variar una semilla aleatoria.

- **Inicialización Aleatoria:** Se inicializa el generador de números aleatorios con una semilla para que los resultados puedan ser reproducibles si se usa la misma semilla.
- **Construcción Iterativa de la Secuencia:** Al igual que el determinista, construye la secuencia avión por avión.
- **Creación de Lista de Candidatos:** En cada iteración, en lugar de elegir inmediatamente el avión con el menor costo, primero:
  - Identifica todos los aviones no programados que pueden aterrizar de manera factible (respetando `t_temprano`, `t_tarde` y tiempos de separación).
  - Para cada uno de estos aviones factibles, calcula su tiempo de aterrizaje y el costo de penalización asociado.
  - Estos aviones (junto con sus costos y tiempos) se guardan en listas de "candidatos".
- **Fallback (si es necesario):** Si no se encuentra ningún avión factible (ninguno puede aterrizar antes de su `t_tarde`), se activa un mecanismo de fallback similar al determinista: se fuerza el aterrizaje del primer avión disponible ajustando su tiempo a `t_tarde` si es necesario, y este se convierte en el único candidato.
- **Selección Aleatoria:** De la lista de aviones candidatos, se selecciona uno al azar.
- **Actualización y Repetición:** El avión seleccionado aleatoriamente se añade a la secuencia, se registra su tiempo y costo, y el proceso se repite hasta que todos los aviones están programados.

Al ejecutar esta función múltiples veces con diferentes semillas, se pueden explorar diversas soluciones "buenas" en lugar de una única solución determinista.

4.3. Resultados

Semilla	Secuencia	Costo	Tiempos de Aterrizaje
0	[13, 6, 14, 7, 0, 5, 12, 11, 4, 3, 9, 8, 10, 1, 2]	61470.0	[152, 160, 276, 291, 306, 321, 329, 344, 359, 367, 375, 383, 398, 401, 416]
1	[2, 10, 14, 1, 6, 3, 12, 13, 7, 11, 8, 4, 0, 9, 5]	60440.0	[84, 266, 276, 279, 294, 302, 310, 318, 326, 341, 356, 364, 379, 394, 402]
2	[13, 14, 0, 2, 3, 8, 5, 9, 6, 11, 4, 1, 12, 7, 10]	62510.0	[152, 276, 279, 294, 302, 310, 318, 326, 334, 349, 364, 379, 394, 402, 417]
3	[3, 10, 9, 2, 7, 14, 12, 1, 8, 0, 11, 6, 13, 4, 5]	63920.0	[89, 266, 281, 289, 297, 312, 327, 342, 357, 372, 375, 390, 398, 406, 414]
4	[3, 5, 1, 14, 9, 11, 4, 2, 0, 6, 12, 10, 7, 8, 13]	58470.0	[89, 107, 190, 276, 291, 306, 321, 329, 344, 359, 367, 382, 397, 405, 413]
5	[9, 4, 13, 6, 14, 11, 0, 12, 10, 2, 1, 5, 3, 8, 7]	47460.0	[134, 142, 152, 160, 276, 279, 282, 297, 312, 327, 342, 357, 365, 373, 381]
6	[12, 9, 1, 8, 5, 0, 2, 6, 13, 11, 10, 7, 4, 3, 14]	36960.0	[160, 168, 190, 205, 213, 228, 243, 251, 259, 274, 277, 292, 300, 308, 323]
7	[5, 2, 8, 13, 0, 3, 14, 4, 7, 11, 1, 9, 6, 10, 12]	34330.0	[107, 115, 123, 152, 167, 182, 276, 291, 299, 314, 317, 332, 340, 355, 370]
8	[3, 6, 8, 2, 5, 0, 4, 9, 7, 13, 10, 14, 12, 1, 11]	12100.0	[89, 109, 117, 125, 133, 148, 163, 171, 179, 187, 266, 276, 291, 306, 309]
9	[7, 10, 5, 4, 2, 3, 0, 12, 11, 9, 14, 1, 8, 6, 13]	62290.0	[109, 266, 281, 289, 297, 305, 320, 335, 350, 365, 380, 383, 398, 406, 414]

Cuadro 2: Resultados del algoritmo Greedy Estocástico con distintas semillas

4.4. Análisis

El algoritmo Greedy Estocástico introduce un elemento de aleatoriedad en la selección del próximo avión a aterrizar. En lugar de elegir siempre el avión con el menor costo inmediato (como el determinista), se construye una lista de candidatos factibles y se selecciona uno de ellos al azar. Para el `case1.txt`, se ejecutó 10 veces con diferentes semillas, obteniendo una variedad de costos:

Semilla	Costo
0	61470.0
1	60440.0
2	62510.0
3	63920.0
4	58470.0
5	47460.0
6	36960.0
7	34330.0
8	12100.0
9	62290.0

- **Variabilidad y Exploración:** La aleatoriedad permite explorar diferentes regiones del espacio de soluciones. Como se observa en la tabla, los costos varían significativamente, desde 12100 (semilla 8) hasta 63920.0 (semilla 3).
- **Fortalezas:** La capacidad de generar múltiples soluciones diversas. Al ejecutarlo repetidamente, aumenta la probabilidad de encontrar soluciones de mejor calidad que el enfoque puramente determinista. Se puede observar en el resultado que se obtuvo con la semilla 8 (solución con un costo de 12100.0).
- **Debilidades:** No hay garantía de que una ejecución aleatoria supere a la determinista, por ende requiere múltiples ejecuciones.
- **Comparación:** El mejor resultado del Greedy Estocástico (12100.0) es significativamente superior al del Greedy Determinista (51660.0), justificando la introducción de la aleatoriedad como mecanismo de exploración.

5. Implementación de GRASP + Hill Climbing (Alguna - Mejora)

A continuación, se observa el pseudocódigo de las funciones que posteriormente se utilizarán para aplicar HC determinista y HC estocástico.

Se eligió alguna - mejora por su eficiencia computacional, permitiendo una exploración más rápida del vecindario en cada paso, aunque se reconoce que podría no ser tan exhaustiva como mejor-mejora.

5.1. Pseudocódigo

Función `calcular_costo`

1. Inicializar `costo_total_calculado` $\leftarrow 0$.
2. Inicializar un arreglo `tiempos_aterrizaje_calculados` de tamaño `longitud(lista_aviones)` con todos los elementos en 0. Este arreglo almacenará los tiempos de aterrizaje calculados según el índice original.
3. Para cada índice i desde 0 hasta `longitud(secuencia_aterrizaje) - 1`:
   - *a*) `indice_avion_actual` $\leftarrow$ `secuencia_aterrizaje`$[i]$.
   - *b*) `avion_actual_info` $\leftarrow$ `lista_aviones[indice_avion_actual]`.
   - *c*) `tiempo_minimo_aterrizaje` $\leftarrow$ `avion_actual_info.tiempo_temprano`.
   - *d*) Para cada índice j desde 0 hasta $i - 1$:
     1) `indice_avion_previo` $\leftarrow$ `secuencia_aterrizaje`$[j]$.
     2) `tiempo_requerido_por_separacion` $\leftarrow$ `tiempos_aterrizaje_calculados[indice_avion_previo]` + `matriz_tiempos_separacion[indice_avion_previo][indice_avion_actual]`.
     3) `tiempo_minimo_aterrizaje` $\leftarrow$ máx(`tiempo_minimo_aterrizaje`, `tiempo_requerido_por_separacion`).
   - *e*) `tiempo_aterrizaje_final_avion` $\leftarrow$ `tiempo_minimo_aterrizaje`.
   - *f*) Si `tiempo_aterrizaje_final_avion` $>$ `avion_actual_info.tiempo_tarde`, entonces retornar ∞.
   - *g*) `tiempos_aterrizaje_calculados[indice_avion_actual]` $\leftarrow$ `tiempo_aterrizaje_final_avion`.
   - *h*) Penalizaciones:
     - Si `tiempo_aterrizaje_final_avion` $<$ `avion_actual_info.tiempo_preferido`: `costo_total_calculado += (avion_actual_info.tiempo_preferido - tiempo_aterrizaje_final_avion) * avion_actual_info.penalizacion_temprano`
     - Si `tiempo_aterrizaje_final_avion` $>$ `avion_actual_info.tiempo_preferido`: `costo_total_calculado += (tiempo_aterrizaje_final_avion - avion_actual_info.tiempo_preferido) * avion_actual_info.penalizacion_tarde`
4. Retornar `costo_total_calculado`.

Función `generar_vecino_intercambio`

1. Crear una copia de la `solucion_actual`:
   `solucion_vecina ← copia(solucion_actual)`
2. Generar dos índices distintos aleatoriamente dentro de los límites de la `solucion_vecina`:
   `indice1 ← generar_entero_aleatorio(0, longitud(solucion_vecina) - 1)`
3. Repetir mientras `indice1 == indice2`:
   `indice2 ← generar_entero_aleatorio(0, longitud(solucion_vecina) - 1)`
4. Intercambiar los elementos en los índices `indice1` e `indice2` en `solucion_vecina`:
   `temporal ← solucion_vecina[indice1]`
   `solucion_vecina[indice1] ← solucion_vecina[indice2]`
   `solucion_vecina[indice2] ← temporal`
5. Si `retornar_indices_intercambiados` es VERDADERO:
   `Retornar solucion_vecina, (indice1, indice2)`
6. Sino:
   `Retornar solucion_vecina`

Función `hill_climbing_alguna_mejora`

1. Inicializar `mejor_solucion_actual ← copia(solucion_inicial)`.
2. Calcular el costo de la solución inicial:
   `mejor_costo_actual ← calcular_costo(mejor_solucion_actual, lista_aviones, matriz_tiempos_separacion)`
3. Guardar el costo de la solución inicial:
   `costo_de_entrada ← mejor_costo_actual`
4. Repetir `max_iteraciones_locales` veces:
   a) Generar una solución vecina intercambiando dos elementos de la `mejor_solucion_actual`:
   `solucion_vecina ← generar_vecino_intercambio(mejor_solucion_actual, FALSO)`
   b) Calcular el costo de la solución vecina:
   `costo_vecino ← calcular_costo(solucion_vecina, lista_aviones, matriz_tiempos_separacion)`
   c) Si el costo del vecino es menor que el costo de la mejor solución actual:
   ```
   Si costo_vecino <mejor_costo_actual entonces:
     mejor_solucion_actual ← solucion_vecina
     mejor_costo_actual ← costo_vecino
     Retornar mejor_solucion_actual, mejor_costo_actual
   Fin Si
   ```
5. Si el costo de la mejor solución encontrada es mayor que el costo de la solución inicial:
   ```
   Si mejor_costo_actual >costo_de_entrada entonces:
     Retornar copia(solucion_inicial), costo_de_entrada
   ```
6. Sino:
   `Retornar mejor_solucion_actual, mejor_costo_actual`
7. `Fin Si`

Función `grasp_con_hc`

Parte Determinista

1. Obtener una solución utilizando el algoritmo Greedy Determinista:

   `(secuencia_greedy_det, costo_greedy_det_reportado, _) ← greedy_determinista(D, lista_aviones, matriz_tiempos_separacion)`

2. Verificar el costo de la solución determinista:

   `costo_verificado_det ← calcular_costo(secuencia_greedy_det, lista_aviones, matriz_tiempos_separacion)`

   `costo_greedy_det_real ← costo_verificado_det`

3. Si el costo de la solución determinista es menor que el mejor costo global encontrado:

   ```
   Si costo_greedy_det_real <mejor_costo_global_encontrado entonces:
     mejor_costo_global_encontrado ← costo_greedy_det_real
     mejor_solucion_global_encontrada ← copia(secuencia_greedy_det)
   Fin Si
   ```

4. Aplicar el algoritmo de Hill Climbing a la solución determinista:

   `(solucion_post_hc_det, costo_post_hc_det) ← hill_climbing_alguna_mejora(secuencia_greedy_det, lista_aviones, matriz_tiempos_separacion, max_iteraciones_locales_hc)`

5. Si el costo de la solución post-Hill Climbing determinista es menor que el mejor costo global encontrado:

   ```
   Si costo_post_hc_det <mejor_costo_global_encontrado entonces:
     mejor_costo_global_encontrado ← costo_post_hc_det
     mejor_solucion_global_encontrada ← copia(solucion_post_hc_det)
   Fin Si
   ```

Parte Estocástica

1. Para cada `semilla` desde 0 hasta `num_semillas_estocastico - 1`:

   a) Obtener una solución utilizando el algoritmo Greedy Estocástico con la semilla actual:

   `(secuencia_greedy_estoc, costo_greedy_estoc_reportado, _) ← greedy_estocastico(D, lista_aviones, matriz_tiempos_separacion, semilla)`

   b) Verificar el costo de la solución estocástica:

   `costo_verificado_estoc ← calcular_costo(secuencia_greedy_estoc, lista_aviones, matriz_tiempos_separacion)`

   `costo_greedy_estoc_real ← costo_verificado_estoc`

   c) Si el costo de la solución estocástica es menor que el mejor costo global encontrado:

   ```
   Si costo_greedy_estoc_real <mejor_costo_global_encontrado entonces:
     mejor_costo_global_encontrado ← costo_greedy_estoc_real
     mejor_solucion_global_encontrada ← copia(secuencia_greedy_estoc)
   Fin Si
   ```

   d) Para cada `reinicio` desde 0 hasta `num_reinicios_estocastico_hc - 1`:

   1) Aplicar el algoritmo de Hill Climbing a la solución estocástica actual:

   `(solucion_post_hc_estoc, costo_post_hc_estoc) ← hill_climbing_alguna_mejora(secuencia_greedy_estoc, lista_aviones, matriz_tiempos_separacion, max_iteraciones_locales_hc)`

2) Si el costo de la solución post-Hill Climbing estocástica es menor que el mejor costo global encontrado:

```
Si costo_post_hc_estoc <mejor_costo_global_encontrado entonces:
  mejor_costo_global_encontrado ← costo_post_hc_estoc
  mejor_solucion_global_encontrada ← copia(solucion_post_hc_estoc)
Fin Si
```

5.2. Funcionamiento

Parte Determinista

1. Inicialmente, se construye una solución utilizando un `greedy_determinista`.
2. Se intenta mejorar la solución obtenida en el paso anterior aplicando `hill_climbing`. Este algoritmo explora el vecindario de la solución actual y se mueve a una solución vecina solo si esta mejora el costo. Se aplica la estrategia de alguna - mejora, que implica que la búsqueda se detiene tan pronto como se encuentra un vecino con un costo menor.
3. Se mantiene un registro de la mejor solución encontrada hasta el momento, junto con su costo asociado. Esta variable global se actualiza si la solución obtenida es mejor que la mejor solución global previamente registrada.

Parte Estocástica

1. Se genera una solución inicial utilizando un `greedy_estocastico`. A diferencia del enfoque determinista, este algoritmo incorpora elementos de aleatoriedad en su proceso de construcción, lo que permite generar diferentes soluciones iniciales en cada ejecución (controlado por una semilla diferente en cada iteración).
2. A la solución, se aplica el algoritmo de búsqueda local `hill_climbing_alguna_mejora` un número de veces especificado por el parámetro `num_restarts_estocastico`. Es importante notar que cada aplicación de Hill Climbing comienza desde la misma solución estocástica generada en esta iteración. Sin embargo, la aleatoriedad inherente al mecanismo de generación de vecinos dentro del Hill Climbing (a través de `generar_vecino_intercambio`) puede llevar a la exploración de diferentes caminos en el espacio de soluciones y, potencialmente, a la identificación de diferentes óptimos locales.
3. Después de cada aplicación del HC, la solución resultante se compara con la mejor solución global encontrada. Si la nueva solución obtenida tiene un costo menor, la mejor solución global y su costo se actualizan.

Finalmente, el algoritmo devuelve la mejor secuencia de aterrizaje encontrada junto con el costo asociado.

5.3. Resultados

Método	**Costo Inicial**	**Mejor Costo Hill Climbing**
Greedy Determinista	51660.0	50520.0
GRASP desde Greedy Determinista	50520.0	50520.0
Mejor Secuencia Encontrada	[0, 9, 7, 1, 11, 14, 10, 12, 8, 6, 5, 13, 4, 3, 2]	

Cuadro 3: Resultados para método determinista

Seed	Costo Inicial	Mejor Costo Hill Climbing
0	61470.0	33220.0
1	60440.0	52380.0
2	62510.0	59870.0
3	63920.0	41040.0
4	58470.0	43070.0
5	47460.0	32930.0
6	62130.0	45380.0
7	62760.0	46620.0
8	60710.0	44010.0
9	63090.0	43110.0
Mejor Global	—	**11490.0**
Mejor Secuencia Encontrada	[3, 4, 8, 2, 5, 0, 6, 9, 7, 13, 10, 14, 12, 1, 11]	

Cuadro 4: Resultados para método estocástico

5.4. Análisis

La implementación de GRASP con Hill Climbing Alguna - Mejora busca mejorar las soluciones iniciales generadas tanto por el Greedy Determinista como por el Greedy Estocástico.

- **Parte Determinista:**
  - Greedy Determinista inicial: Costo 51660.0.
  - Post-HC(AM): Costo **50520.0**.
  - Se logró una mejora sobre la solución Greedy Determinista. Esto sugiere que la solución inicial ya estaba en una región relativamente buena o que HC(AM) encontró rápidamente un óptimo local cercano.
- **Parte Estocástica:**
  - Se observa que las soluciones iniciales del Greedy Estocástico varían (como se vio en los resultados de la sección 4.3).
  - El Hill Climbing consistentemente intenta mejorar estas soluciones. Por ejemplo, la semilla 5 del Greedy Estocástico dio un costo de 47460.0, y tras HC(AM) se redujo a 32930.0.
  - La mejor solución global encontrada mediante GRASP + HC(AM) en la parte estocástica tuvo un costo de **11490.0**.
  - El componente de `num_restarts_estocastico` para el HC permite que, para una misma solución inicial estocástica, el HC explore diferentes vecindarios debido a la aleatoriedad en `generar_vecino_intercambio`.
- **Fortalezas:**
  - GRASP aprovecha la aleatoriedad para generar diversos puntos de partida de buena calidad.
  - Hill Climbing con *alguna mejora* es computacionalmente más rápida que *la mejor mejora*, aunque puede no ser tan exhaustiva.
  - La combinación es más robusta que usar solo Greedy o solo HC.
- **Debilidades:**
  - Hill Climbing incluso con Alguna - Mejora puede quedar atrapado en óptimos locales. Por eso, es imporante aplicar los reinicios.
- **Comparación:** La mejor solución de GRASP+HC(AM) (11490.0) es significativamente mejor que la del Greedy Determinista (51660.0) y ligeramente mejor que la mejor del Greedy Estocástico puro (12100.0).

Este enfoque GRASP+HC(AM) proporciona un buen equilibrio entre la diversificación (GRASP) y la intensificación (HC).

6. Implementación de TS

Se optó por implementar Búsqueda Tabú debido a su capacidad de escapar de óptimos locales mediante el uso de su lista tabú y mecanismos de aspiración.

6.1. Pseudocódigo

Función tabu_search

1. **Inicialización:**
   - *a*) mejor_solucion_global ← copia(solucion_inicial).
   - *b*) mejor_costo_global ← calcular_costo(mejor_solucion_global, lista_aviones, matriz_tiempos_separacion).
   - *c*) solucion_actual ← copia(solucion_inicial).
   - *d*) costo_actual ← mejor_costo_global.
   - *e*) lista_tabu ← $\emptyset$ (lista vacía para almacenar movimientos).
   - *f*) iteraciones_sin_mejora_global_actual ← 0.
2. **Bucle Principal de Búsqueda Tabú:**
   - *a*) Para cada iteracion desde 0 hasta max_iteraciones_totales − 1:
     1) mejor_vecino_iteracion ← NULO.
     2) mejor_costo_vecino_iteracion ← ∞.
     3) mejor_movimiento_iteracion ← NULO.
     4) **Explorar el vecindario:**
        - a' Para cada i desde 0 hasta longitud(solucion_actual) − 1:
        - b' (vecino_candidato, movimiento_candidato) ← generar_vecino_intercambio(solucion_actual, VERDADERO).
        - c' costo_vecino_candidato ← calcular_costo(vecino_candidato, lista_aviones, matriz_tiempos_separacion).
        - d' es_movimiento_tabu ← (movimiento_candidato ∈ lista_tabu).
        - e' cumple_criterio_aspiracion ← (costo_vecino_candidato < mejor_costo_global).
        - f' Si ($\neg$es_movimiento_tabu) $\vee$ cumple_criterio_aspiracion entonces:
        - g' Si costo_vecino_candidato < mejor_costo_vecino_iteracion entonces:
        - h' mejor_vecino_iteracion ← vecino_candidato.
        - i' mejor_costo_vecino_iteracion ← costo_vecino_candidato.
        - j' mejor_movimiento_iteracion ← movimiento_candidato.
        - k' Fin Si
        - l' Fin Si
        - m' Fin Para (exploración de vecinos)

5) Si mejor_vecino_iteracion == NULO entonces:

Continuar con la siguiente iteración

6) Fin Si

7) solucion_actual ← copia(mejor_vecino_iteracion).

8) costo_actual ← mejor_costo_vecino_iteracion.

9) Añadir mejor_movimiento_iteracion a lista_tabu.

10) Si longitud(lista_tabu) > duracion_tabu entonces:

Remover el primer elemento de lista_tabu

11) Fin Si

12) Si costo_actual < mejor_costo_global entonces:

mejor_solucion_global ← copia(solucion_actual)

mejor_costo_global ← costo_actual

iteraciones_sin_mejora_global_actual ← 0

13) Sino:

iteraciones_sin_mejora_global_actual ← iteraciones_sin_mejora_global_actual + 1

14) Fin Si

15) Si iteraciones_sin_mejora_global_actual ≥ max_iteraciones_sin_mejora_global entonces:

Romper el bucle principal

16) Fin Si

b) Fin Para (bucle principal)

3. Retornar mejor_solucion_global, mejor_costo_global.

Función tabu_search_main

1. Inicializar mejor_solucion_global_encontrada ← NULO.
2. Inicializar mejor_costo_global_encontrado ← ∞.
3. **Parte 1: Greedy Determinista como punto de partida**

a) Obtener una solución utilizando el algoritmo Greedy Determinista:

(secuencia_greedy_det,costo_greedy_det_reportado,) ←
greedy_determinista(D, lista_aviones,
matriz_tiempos_separacion)

b) Verificar el costo de la solución determinista:

costo_verificado_det ← calcular_costo(secuencia_greedy_det,
lista_aviones, matriz_tiempos_separacion)
costo_greedy_det_real ← costo_verificado_det

c) Si el costo de la solución determinista es menor que el mejor costo global encontrado:

Si costo_greedy_det_real < mejor_costo_global_encontrado entonces:

mejor_costo_global_encontrado ← costo_greedy_det_real
mejor_solucion_global_encontrada ← copia(secuencia_greedy_det)

Fin Si

d) Aplicar el algoritmo de Búsqueda Tabú a la solución determinista:

(solucion_post_ts_det,costo_post_ts_det) ←
tabu_search(secuencia_greedy_det, lista_aviones,
matriz_tiempos_separacion, duracion_tabu,
max_iter_ts, max_iter_sin_mejora_ts)

e) Si el costo de la solución post-Búsqueda Tabú determinista es menor que el mejor costo global encontrado:

Si costo_post_ts_det < mejor_costo_global_encontrado entonces:

mejor_costo_global_encontrado ← costo_post_ts_det
mejor_solucion_global_encontrada ← copia(solucion_post_ts_det)

Fin Si

4. **Parte 2: Greedy Estocástico como múltiples puntos de partida**

a) Para cada semilla desde 0 hasta num_semillas_estocastico − 1:

1) Obtener una solución utilizando el algoritmo Greedy Estocástico con la semilla actual:

(secuencia_greedy_estoc,costo_greedy_estoc_reportado,) ←
greedy_estocastico(D, lista_aviones,
matriz_tiempos_separacion, semilla)

2) Verificar el costo de la solución estocástica:

costo_verificado_estoc ← calcular_costo(secuencia_greedy_estoc,
lista_aviones, matriz_tiempos_separacion)
costo_greedy_estoc_real ← costo_verificado_estoc

3) Si el costo de la solución estocástica es menor que el mejor costo global encontrado:

Si costo_greedy_estoc_real < mejor_costo_global_encontrado entonces:

mejor_costo_global_encontrado ← costo_greedy_estoc_real
mejor_solucion_global_encontrada ← copia(secuencia_greedy_estoc)

Fin Si

4) Para cada reinicio desde 0 hasta num_reinicios_estocastico_ts − 1:

a' Aplicar el algoritmo de Búsqueda Tabú a la solución estocástica actual:

$$\begin{aligned}(\text{solucion\_post\_ts\_estoc}, &\text{costo\_post\_ts\_estoc}) \leftarrow \\ &\text{tabu\_search}(\text{secuencia\_greedy\_estoc}, \\ &\quad \text{lista\_aviones}, \text{matriz\_tiempos\_separacion}, \\ &\quad \text{duracion\_tabu}, \text{max\_iter\_ts}, \\ &\quad \text{max\_iter\_sin\_mejora\_ts})\end{aligned}$$

b' Si el costo de la solución post-Búsqueda Tabú estocástica es menor que el mejor costo global encontrado:

$$\text{Si costo\_post\_ts\_estoc} < \text{mejor\_costo\_global\_encontrado entonces:}$$

$$\begin{aligned}\text{mejor\_costo\_global\_encontrado} &\leftarrow \text{costo\_post\_ts\_estoc} \\ \text{mejor\_solucion\_global\_encontrada} &\leftarrow \text{copia}(\text{solucion\_post\_ts\_estoc})\end{aligned}$$

$$\text{Fin Si}$$

5. Retornar mejor_solucion_global_encontrada, mejor_costo_global_encontrado.

Parte Estocástica (Multi-arranque)

Esta parte se ejecuta un número de veces especificado por el parámetro `num_seeds_estocastico`. Donde en cada iteración:

1. Se genera una solución inicial diferente utilizando un `greedy_estocastico`. A diferencia del determinista, este enfoque incorpora elementos de aleatoriedad, lo que permite generar diversas soluciones de partida al utilizar una semilla diferente en cada iteración.
2. La solución inicial generada estocásticamente se compara con la mejor solución global encontrada hasta el momento. Si su costo es menor, se actualiza la mejor solución global.
3. El algoritmo `tabu_search` se aplica a esta misma solución inicial estocástica un número de veces definido por el parámetro `num_restarts_estocastico`. Aunque cada aplicación del Tabu Search comienza desde la misma solución inicial, la implementación específica de `tabu_search` utilizada en este trabajo incorpora un muestreo aleatorio en la generación de su vecindario en cada iteración (se generan y evalúan D vecinos aleatorios, donde D es el número de aviones). Esta aleatoriedad en la selección del siguiente movimiento, combinada con la lista tabú, permite que cada una de estas `num_restarts_estocastico` ejecuciones de TS pueda seguir trayectorias de búsqueda diferentes, llevando potencialmente a la identificación de distintos óptimos locales y, por ende, a resultados variables para la misma solución de partida.

6.2. Resultados

Configuración	Mejor Costo Final	Mejor Secuencia Final
1	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
2	3550.0	[3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]
3	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
4	3550.0	[3,2,4,7,5,8,6,9,0,13,12,14,1,11,10]
5	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,10,11]

Cuadro 5: Comparación de los Mejores Resultados por Configuración

Detalles Configuración 1

Parámetros

- tenure: 5
- max_iter: 100
- max_iter_sin_mejora: 50
- num_seeds_estocastico: 10
- num_restarts_estocastico: 5

Métrica	**Valor**
Costo Inicial (Greedy Det.)	51660.0
Secuencia Inicial (Greedy Det.)	[0,9,13,1,11,14,10,12,8,6,5,7,4,3,2]
Costo Final (Tabu Search Det.)	3610.0
Secuencia Final (Tabu Search Det.)	[3,2,4,5,7,6,8,9,0,13,12,14,11,1,10]
Iteración Mejor Solución (TS Det.)	27
Nota (TS Det.)	Detenido por 50 iteraciones sin mejora

Cuadro 6: Configuración 1 - Resultados con Greedy Determinista y Tabu Search

Seed	**Costo Greedy Est.**	**Mejor Costo TS**	**Secuencia Mejor TS**
0	61470.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
1	60440.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,10,11]
2	62510.0	3550.0	[3,2,4,5,7,6,8,9,0,13,12,14,1,10,11]
3	63920.0	3550.0	[3,2,4,7,5,6,8,9,0,13,12,14,1,10,11]
4	58470.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,10,11]
5	47460.0	3550.0	[3,2,4,7,6,5,8,9,0,13,12,14,1,11,10]
6	36960.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
7	34330.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
8	12100.0	3610.0	[3,2,4,5,6,7,8,9,0,13,12,14,10,1,11]
9	62290.0	3550.0	[3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]
Mejor Global Costo:		**3550.0**	**Secuencia:** [3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]

Cuadro 7: Configuración 1 - Resultados Tabu Search con Greedy Estocástico (Mejores por Seed)

Detalles Configuración 2

Parámetros

- tenure: 10
- max_iter: 200
- max_iter_sin_mejora: 100
- num_seeds_estocastico: 5
- num_restarts_estocastico: 3

Métrica	**Valor**
Costo Inicial (Greedy Det.)	51660.0
Secuencia Inicial (Greedy Det.)	[0,9,13,1,11,14,10,12,8,6,5,7,4,3,2]
Costo Final (Tabu Search Det.)	3550.0
Secuencia Final (Tabu Search Det.)	[3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]
Iteración Mejor Solución (TS Det.)	64
Nota (TS Det.)	Detenido por 100 iteraciones sin mejora

Cuadro 8: Configuración 2 - Resultados con Greedy Determinista y Tabu Search

Seed	**Costo Greedy Est.**	**Mejor Costo TS**	**Secuencia Mejor TS**
0	61470.0	3610.0	[3,2,4,6,5,7,8,9,0,13,12,14,1,11,10]
1	60440.0	3550.0	[3,2,4,7,6,5,8,9,0,13,12,14,1,11,10]
2	62510.0	3550.0	[3,2,4,7,5,6,8,9,0,13,12,14,1,11,10]
3	63920.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,10,11]
4	58470.0	3550.0	[3,2,4,7,5,8,6,9,0,13,12,14,1,10,11]
Mejor Global Costo:		**3550.0**	**Secuencia:** [3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]

Cuadro 9: Configuración 2 - Resultados Tabu Search con Greedy Estocástico (Mejores por Seed)

Detalles Configuración 3

Parámetros

- tenure: 3
- max_iter: 50
- max_iter_sin_mejora: 25
- num_seeds_estocastico: 10
- num_restarts_estocastico: 5

Métrica	**Valor**
Costo Inicial (Greedy Det.)	51660.0
Secuencia Inicial (Greedy Det.)	[0,9,13,1,11,14,10,12,8,6,5,7,4,3,2]
Costo Final (Tabu Search Det.)	4180.0
Secuencia Final (Tabu Search Det.)	[2,3,5,4,7,6,8,9,0,13,12,10,14,1,11]
Iteración Mejor Solución (TS Det.)	47
Nota (TS Det.)	Max iter (50) alcanzado

Cuadro 10: Configuración 3 - Resultados con Greedy Determinista y Tabu Search

Seed	**Costo Greedy Est.**	**Mejor Costo TS**	**Secuencia Mejor TS**
0	61470.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
1	60440.0	3780.0	[2,3,4,7,5,6,8,0,9,13,12,14,1,11,10]
2	62510.0	3550.0	[3,2,4,7,5,8,6,9,0,13,12,14,1,10,11]
3	63920.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,10,11]
4	58470.0	3550.0	[3,2,4,5,6,7,8,9,0,13,12,14,1,11,10]
5	47460.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,11,10]
6	36960.0	3610.0	[3,2,4,6,7,5,8,9,0,13,12,14,1,10,11]
7	34330.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,10,11]
8	12100.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
9	62290.0	3550.0	[3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]
Mejor Global Costo:		**3550.0**	**Secuencia:** [3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]

Cuadro 11: Configuración 3 - Resultados Tabu Search con Greedy Estocástico (Mejores por Seed)

Detalles Configuración 4

Parámetros

- tenure: 7
- max_iter: 150
- max_iter_sin_mejora: 75
- num_seeds_estocastico: 8
- num_restarts_estocastico: 4

Métrica	Valor
Costo Inicial (Greedy Det.)	51660.0
Secuencia Inicial (Greedy Det.)	[0,9,13,1,11,14,10,12,8,6,5,7,4,3,2]
Costo Final (Tabu Search Det.)	3550.0
Secuencia Final (Tabu Search Det.)	[3,2,4,7,5,8,6,9,0,13,12,14,1,11,10]
Iteración Mejor Solución (TS Det.)	56
Nota (TS Det.)	Detenido por 75 iteraciones sin mejora

Cuadro 12: Configuración 4 - Resultados con Greedy Determinista y Tabu Search

Seed	Costo Greedy Est.	Mejor Costo TS	Secuencia Mejor TS
0	61470.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,10,11]
1	60440.0	3550.0	[3,2,4,7,6,8,5,9,0,13,12,14,1,11,10]
2	62510.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
3	63920.0	3550.0	[3,2,4,5,7,8,6,9,0,13,12,14,1,11,10]
4	58470.0	3550.0	[3,2,4,7,6,8,5,9,0,13,12,14,1,10,11]
5	47460.0	3550.0	[3,2,4,7,5,6,8,9,0,13,12,14,1,10,11]
6	36960.0	3550.0	[3,2,4,5,6,8,7,9,0,13,12,14,1,10,11]
7	34330.0	3550.0	[3,2,4,7,5,8,6,9,0,13,12,14,1,10,11]
Mejor Global Costo:		**3550.0**	**Secuencia:** [3,2,4,7,5,8,6,9,0,13,12,14,1,11,10]

Cuadro 13: Configuración 4 - Resultados Tabu Search con Greedy Estocástico (Mejores por Seed)

Detalles Configuración 5

Parámetros

- tenure: 15
- max_iter: 300
- max_iter_sin_mejora: 150
- num_seeds_estocastico: 3
- num_restarts_estocastico: 2

Métrica	Valor
Costo Inicial (Greedy Det.)	51660.0
Secuencia Inicial (Greedy Det.)	[0,9,13,1,11,14,10,12,8,6,5,7,4,3,2]
Costo Final (Tabu Search Det.)	3550.0
Secuencia Final (Tabu Search Det.)	[3,2,4,5,7,8,6,9,0,13,12,14,1,10,11]
Iteración Mejor Solución (TS Det.)	78
Nota (TS Det.)	Detenido por 150 iteraciones sin mejora

Cuadro 14: Configuración 5 - Resultados con Greedy Determinista y Tabu Search

Seed	Costo Greedy Est.	Mejor Costo TS	Secuencia Mejor TS
0	61470.0	3550.0	[3,2,4,5,7,6,8,9,0,13,12,14,1,10,11]
1	60440.0	3550.0	[3,2,4,7,6,5,8,9,0,13,12,14,1,11,10]
2	62510.0	3550.0	[3,2,4,7,6,5,8,9,0,13,12,14,1,11,10]
Mejor Global Costo:		**3550.0**	**Secuencia:** [3,2,4,5,7,8,6,9,0,13,12,14,1,10,11]

Cuadro 15: Configuración 5 - Resultados Tabu Search con Greedy Estocástico (Mejores por Seed)

6.3. Análisis

La Búsqueda Tabú (TS) implementada se aplicó tanto a la solución inicial del Greedy Determinista como a múltiples soluciones iniciales generadas por el Greedy Estocástico (usando diferentes semillas). Se probaron 5 configuraciones de parámetros para TS (`tenure`, `max_iter`, `max_iter_sin_mejora`, `num_seeds_estocastico`, `num_restarts_estocastico`).

- **Resultados Clave (Mejor de las 5 Configuraciones):**
  - La mejor solución global encontrada por cualquiera de las configuraciones de TS fue de **3550.0**.
  - Esta solución se obtuvo consistentemente a través de varias configuraciones, indicando robustez en la capacidad de TS para converger a soluciones de alta calidad para este problema y este caso.
  - Por ejemplo, en la "Configuración 1":
    - Greedy Determinista + TS: Costo inicial 51660.0 → Costo final TS **3610.0**.
    - Greedy Estocástico + TS (mejor semilla): La semilla 8 (costo inicial 12100.0) mejorada por TS llegó a **3610.0**. Sin embargo, otras semillas estocásticas combinadas con TS también convergieron a costos de **3550.0**.
- **Efecto de los Parámetros de TS:**
  - **Tenure (Duración):** Un tenure demasiado corto puede causar ciclos y uno demasiado largo puede restringir demasiado la búsqueda. Las configuraciones probadas (3, 5, 7, 10, 15) permitieron lograr consistentemente el costo de 3550.0.

- **max_iter y max_iter_sin_mejora:** Estos controlan el esfuerzo computacional. Es importante dar suficientes iteraciones para que el algoritmo explore adecuadamente. En algunas configuraciones, se alcanzó `max_iter_sin_mejora` (ej. Config. 1 y 2 TS Det), lo que indica convergencia o estancamiento. Mientras que en otras (ej. Config. 3 TS Det), se alcanzó `max_iter`, sugiriendo que más iteraciones podrían (o no) haber mejorado.
- **Multi-arranque (`num_seeds_estocastico` y `num_restarts_estocastico_ts`):** Aplicar este algoritmo a múltiples soluciones iniciales estocásticas con reinicios se refiere a la cantidad de veces que se aplica este a *diferentes* soluciones estocásticas iniciales. Esto aumenta significativamente la probabilidad de encontrar el óptimo global o soluciones muy cercanas a él. En este caso, las diversas semillas convergieron a la misma calidad de solución (3550.0).

- **Fortalezas:**
  - TS demostró ser muy eficaz para escapar de óptimos locales donde HC podría haberse quedado atrapado, gracias a la lista tabú y al criterio de aspiración.
  - La capacidad de explorar movimientos que empeoran temporalmente la solución es clave para su éxito.
  - El multi-arranque con soluciones estocásticas mejora aún más su rendimiento global.
- **Debilidades:**
  - Más complejo de implementar y ajustar que los greedy o HC.
  - El rendimiento es sensible a la configuración de sus parámetros.
- **Comparación final de todos los algoritmos:**

Algoritmo	**Costo**
Tabu Search (TS)	3550.0
Greedy Determinista	51660.0
Mejor Greedy Estocástico	12100.0
Mejor GRASP+HC(AM)	11490.0

Cuadro 16: Comparación de costos entre algoritmos

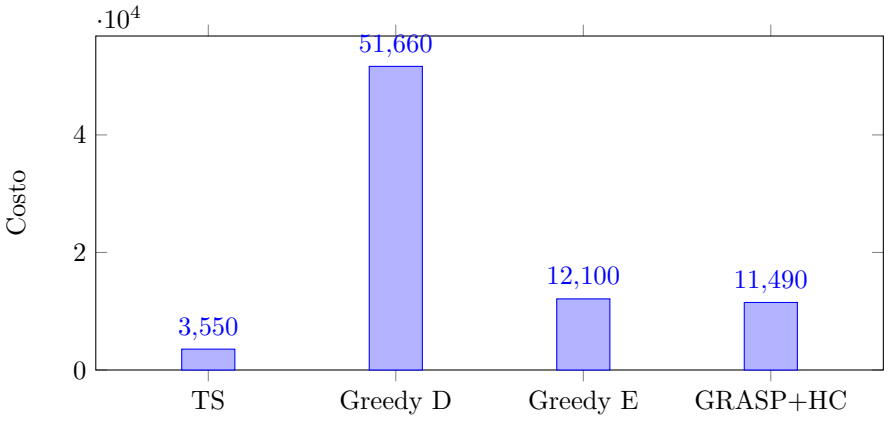

Figura 1: Comparación visual del costo por algoritmo

- Esto recalca la potencia de metaheurísticas más avanzadas como lo es Tabu Search para problemas de optimización combinatoria complejos.

7. Conclusión

En este informe se abordó el problema de la optimización de secuencias de aterrizaje de aviones mediante la implementación de diversas metaheurísticas.

1. **Greedy Determinista:** Proporcionó una solución rápida y factible (costo 51660.0).
2. **Greedy Estocástico:** Al introducir aleatoriedad, se logró una exploración más amplia del espacio de soluciones. La mejor de 10 ejecuciones (costo 12100.0) demostró una mejora sustancial sobre el enfoque determinista, resaltando el valor de la diversificación.
3. **GRASP + Hill Climbing (Alguna - Mejora):** Esta metaheurística combinó la parte constructiva (determinista y estocástica) con una parte de búsqueda local. Este algoritmo mejoró consistentemente las soluciones iniciales, logrando un costo de 50520.0 partiendo de la solución determinista, y un costo óptimo de **11490.0** partiendo de las soluciones estocásticas. Esto evidenció la sinergia entre una buena construcción inicial y la búsqueda local.
4. **Búsqueda Tabú (TS):** Este algoritmo demostró ser la más poderosa. Aplicada tanto a soluciones iniciales deterministas como estocásticas, y probada con 5 configuraciones de parámetros distintas, TS consistentemente encontró soluciones de muy alta calidad, alcanzando un costo óptimo de **3550.0**. La lista tabú y el criterio de aspiración fueron fundamentales para su éxito.

En resumen, para el problema de secuenciación de aterrizajes presentado y el `case1.txt`, Tabu Search se destacó como la técnica más efectiva, logrando la reducción de costos más significativa.