UNIVERSIDAD DIEGO PORTALES

FACULTAD DE INGENIERÍA Y CIENCIAS

ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES

Algoritmos y Metaheurísticas

Tarea 02: Colonias espaciales 25XX

Profesor: Victor Reyes

Alumnos: Felipe Condore y Fernando Burón

Santiago, Chile Mayo 2023

Índice

1. Introducción

En el año 2535, la humanidad ha logrado establecer colonias espaciales gracias a la creación de atmósferas artificiales. Estas colonias requieren la recolección de distintos tipos de minerales para construir más infraestructura y garantizar su supervivencia. Como jefe general de las colonias IO, Europa, Deimos y Titán, se utilizan vehículos autónomos no tripulados (UAVs) de distintos tamaños para recolectar estos recursos. El desafío es organizar de manera eficiente los aterrizajes y descargas de minerales, asignando a cada UAV un tiempo de aterrizaje determinado.

2. Metodología

2.1. Algoritmo Greedy

El algoritmo Greedy, también conocido como algoritmo voraz, toma la decisión que parece ser la mejor en cada momento con la esperanza de que estas decisiones locales lleven a una solución global óptima. En este caso, se implementarán dos versiones de este algoritmo: determinista y estocástico.

- **Determinista**

  El algoritmo Greedy determinista tomará la decisión óptima en cada paso sin tener en cuenta ninguna aleatoriedad. En el contexto de este problema, podría implicar asignar a los UAVs los tiempos de aterrizaje de manera que se minimice el costo total en cada paso.

- **Estocástico**

  A diferencia del Greedy determinista, el algoritmo Greedy estocástico introduce un elemento de aleatoriedad en la toma de decisiones. Esto significa que no siempre se toma la decisión óptima en cada paso, lo que puede permitir explorar un espacio de soluciones más amplio. Se realizarán cinco ejecuciones de este algoritmo para cada caso de prueba, controladas por una variable seed.

2.2. Algoritmo Hill-Climbing

El algoritmo Hill-Climbing es un algoritmo de búsqueda local que comienza con una solución arbitraria a un problema y luego intenta encontrar una solución mejor mediante cambios incrementales. Se implementarán dos variantes de este algoritmo: alguna-mejora y mejor-mejora.

- **Alguna-Mejora**

  En la variante alguna-mejora, el algoritmo realiza un cambio en la solución actual tan pronto como encuentra una solución vecina que es mejor que la actual. Esto puede permitir que el algoritmo salga de los óptimos locales.

- **Mejor-Mejora**

  En la variante mejor-mejora, el algoritmo examina todas las soluciones vecinas y elige la mejor de todas ellas. Aunque este enfoque puede ser más lento, puede conducir a soluciones de mejor calidad.

2.3. Algoritmo Tabu Search

El algoritmo Tabu Search es un método de búsqueda metaheurística que se utiliza para encontrar la solución óptima a problemas de optimización combinatoria. Este algoritmo utiliza una lista tabú para evitar visitar soluciones previamente visitadas y así escapar de los óptimos locales. En este caso, se utilizarán las soluciones generadas por los algoritmos Greedy como punto de partida.

Cada uno de estos algoritmos se implementará y se probará en los casos de prueba proporcionados. Los resultados se analizarán y se discutirán en detalle en la sección de resultados y discusión del informe.

3. Implementación del algoritmo

Para comenzar, es necesario leer la entrada desde los archivos de texto proporcionados, los cuales siguen el formato descrito a continuación:

La primera línea, denotada por N, indica la cantidad de UAVs presentes. Después de esta línea, se inicia un bucle de tamaño N, que contiene una línea con los tiempos mínimo, preferible y máximo de aterrizaje, seguido de una lista que representa la separación mínima entre cada UAV.

La función implementada para esta tarea se muestra a continuación (en pseudocódigo):

```
Función leer_archivo_entrada(nombre_archivo):
    Abrir nombre_archivo en modo lectura como f:
        Leer la primera línea de f y asignarla a n_uavs_line
        Convertir n_uavs_line a entero y asignarlo a n_uavs

        Inicializar uav_data como lista vacía
        Inicializar separation_times como lista vacía
        Para i en el rango de 0 a n_uavs:
            Leer la siguiente línea de f y asignarla a uav_line
            Convertir uav_line a una lista de enteros y asignarla a min_time, pref_time, max_time
            Añadir (min_time, pref_time, max_time, i) a uav_data

            Inicializar separation_row como lista vacía
            Mientras la longitud de separation_row sea menor que n_uavs:
                Leer la siguiente línea de f y asignarla a separation_line
                Convertir separation_line a una lista de enteros y añadirlos a separation_row
            Añadir separation_row a separation_times

    Devolver n_uavs, uav_data, separation_times
```

3.1. Greedy Determinístico

Este código implementa un algoritmo determinista y codicioso (greedy) para programar UAVs basado en tiempos de separación y preferencias de tiempo.

1. El código toma tres argumentos: 'n_uavs' (el número de UAVs), 'uav_data' (una lista de tuplas, donde cada tupla contiene el tiempo mínimo, el tiempo preferido, el tiempo máximo y el ID para cada UAV), y 'separation_times' (una matriz que contiene los tiempos de separación entre cada par de UAVs).

2. Inicializa 'costo' a 0 y 'schedule' a una lista vacía. 'costo' llevará la cuenta del costo total, que es la suma de las diferencias absolutas entre los tiempos preferidos y los tiempos de inicio reales para cada UAV. 'schedule' almacenará la programación final de los UAVs.

3. Ordena 'uav_data' en orden ascendente de tiempo preferido.

4. Luego, para cada UAV (en el orden arreglado), hace lo siguiente:

   - Si 'schedule' está vacío (es decir, si es el primer UAV), simplemente agrega el tiempo preferido y el ID del UAV a 'schedule'.
   - Si 'schedule' no está vacío, calcula el tiempo de inicio para el UAV actual como el máximo entre su tiempo mínimo y la suma del tiempo de inicio del UAV anterior y el tiempo de separación entre el UAV anterior y el actual. Si este tiempo de inicio calculado es menor o igual al tiempo máximo del UAV actual, agrega la diferencia absoluta entre el tiempo preferido y el tiempo de inicio al 'costo', y agrega el tiempo de inicio y el ID del UAV a 'schedule'.

5. Finalmente, devuelve 'schedule' y 'costo'.

Pseudocódigo:

```
procedure DeterministicGreedy(n_uavs, uav_data, separation_times)
    costo ← 0
    schedule ← []
    tmp_data ← uav_data[:]
    tmp_data.sort(key = lambdax : x[0])
    for i in range(n_uavs) do
        min_time, pref_time, max_time, id ← tmp_data[i]
        if not schedule then
            schedule.append((pref_time, id))
            continue
        end if
        start_time ← max(min_time, schedule[i − 1][0] + separation_times[schedule[i − 1][1]][id])
        if start_time ≤ max_time then
            costo ← costo + abs(pref_time − start_time)
            schedule.append((start_time, id))
        end if
    end for
    return schedule, costo
end procedure
```

A continuación, se muestra un diagrama que ayuda a comprender mejor la solución:

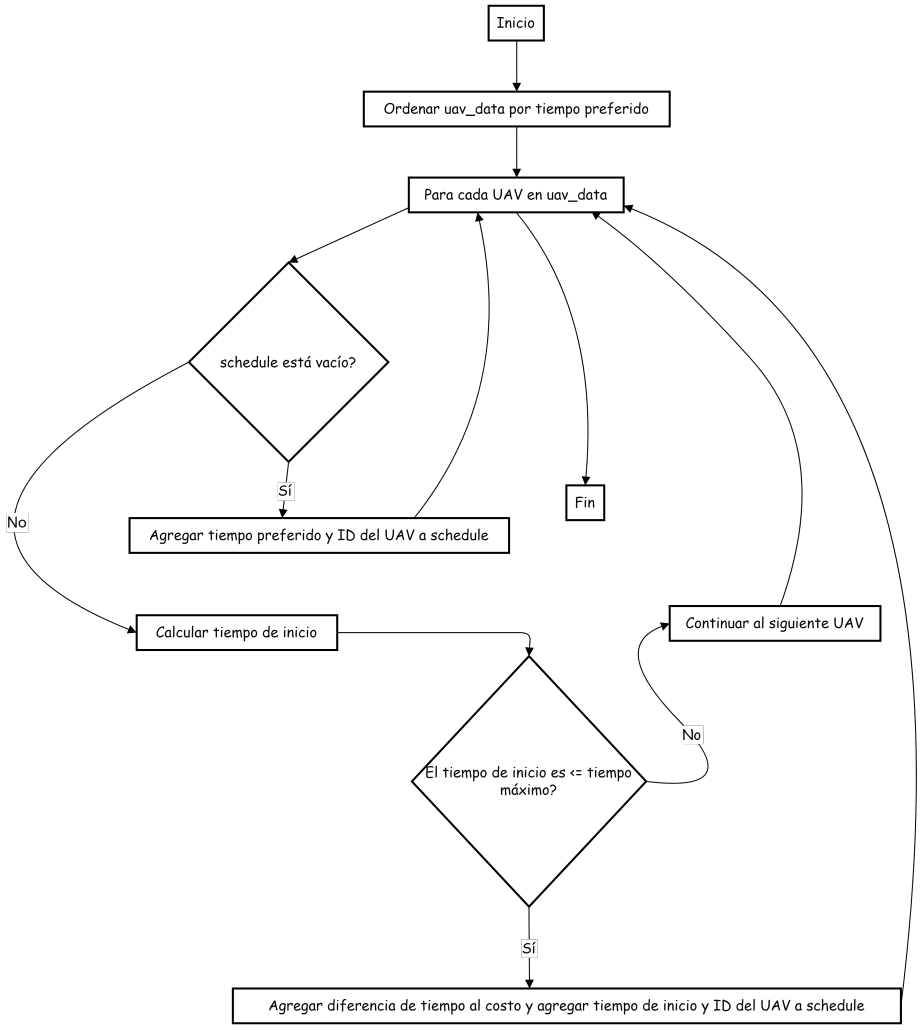

Figura 1: Diagrama de Flujo Greedy Determinista

3.2. Greedy Estocástico

La implementación del Greedy Estocástico toma en cuenta la base del determinista, pero utiliza probabilidades y decisiones aleatorias que permiten generar soluciones (factibles e infactibles).

1. El código toma cuatro argumentos: 'n_uavs' (el número de UAVs), uav_data (una lista de tuplas, donde cada tupla contiene el tiempo mínimo, el tiempo preferido, el tiempo máximo y el ID para cada UAV), separation_times (una matriz que contiene los tiempos de separación entre cada par de UAVs), y seed (una semilla opcional para la generación de números aleatorios).
2. Si se proporciona una semilla, se utiliza para inicializar el generador de números aleatorios.
3. Inicializa costo a 0 y schedule a una lista vacía. costo llevará la cuenta del costo total, que es la suma de las diferencias absolutas entre los tiempos preferidos y los tiempos de inicio reales para cada UAV. schedule almacenará la programación final de los UAVs.
4. Ordena uav_data en orden ascendente de tiempo preferido.
5. Luego, mientras i sea menor que 'n_uavs', hace lo siguiente:
   a) Si schedule está vacío (es decir, si es el primer UAV), simplemente agrega el tiempo preferido y el ID del UAV a schedule.
   b) Si schedule no está vacío, calcula el tiempo de inicio para el UAV actual como el máximo entre su tiempo mínimo y la suma del tiempo de inicio del UAV anterior y el tiempo de separación entre el UAV anterior y el actual.
   c) Si este tiempo de inicio calculado es menor o igual al tiempo máximo del UAV actual, calcula una lista de probabilidades para cada tiempo posible entre el tiempo de inicio y el tiempo máximo, donde la probabilidad de un tiempo dado es inversamente proporcional a la diferencia absoluta entre el tiempo preferido y ese tiempo. Luego, selecciona un tiempo de esta distribución de probabilidad, agrega la diferencia absoluta entre el tiempo preferido y el tiempo seleccionado al costo, y agrega el tiempo seleccionado y el ID del UAV a schedule.
   d) Si el tiempo de inicio calculado es mayor que el tiempo máximo del UAV actual, agrega el tiempo máximo y el ID del UAV a schedule, y agrega el doble de la diferencia absoluta entre el tiempo preferido y el tiempo máximo al costo, porque no cumple una de las restricciones y se genera una **penalización**.
6. Finalmente, devuelve schedule y costo.

Pseudocódigo:

```
procedure StochasticGreedy(n_uavs, uav_data, separation_times, seed)
    if seed ≠ None then
        random.seed(seed)
    end if
    costo ← 0
    schedule ← []
    tmp_data ← uav_data[:]
    tmp_data.sort(key = lambdax : x[1])
    i ← 0
    while i < n_uavs do
        min_time, pref_time, max_time, id ← tmp_data[i]
        if schedule = [] then
            schedule.append((pref_time, id))
            i ← i + 1
            continue
        end if
        start_time ← max(min_time, schedule[−1][0] + separation_times[schedule[−1][1]][id])
        if start_time ≤ max_time then
            weights ← []
            for j ← start_time to max_time do
                weights.append(1/(abs(pref_time − j) + 1e − 10))
```

end for
$total\_weight \leftarrow sum(weights)$
$probabilities \leftarrow [weight/total\_weight$ **for** $weight$ **in** $weights]$
$chosen\_time \leftarrow random.choices(range(start\_time, max\_time + 1), probabilities)[0]$
$costo \leftarrow costo + abs(pref\_time - chosen\_time)$
$schedule.append((chosen\_time, id))$
$i \leftarrow i + 1$
else
$schedule.append((max\_time, id))$
$costo \leftarrow costo + 2 * abs(pref\_time - max\_time)$
$i \leftarrow i + 1$
end if
end while
return $schedule, costo$
end procedure

A continuación, se muestra un diagrama que ayuda a comprender mejor la solución:

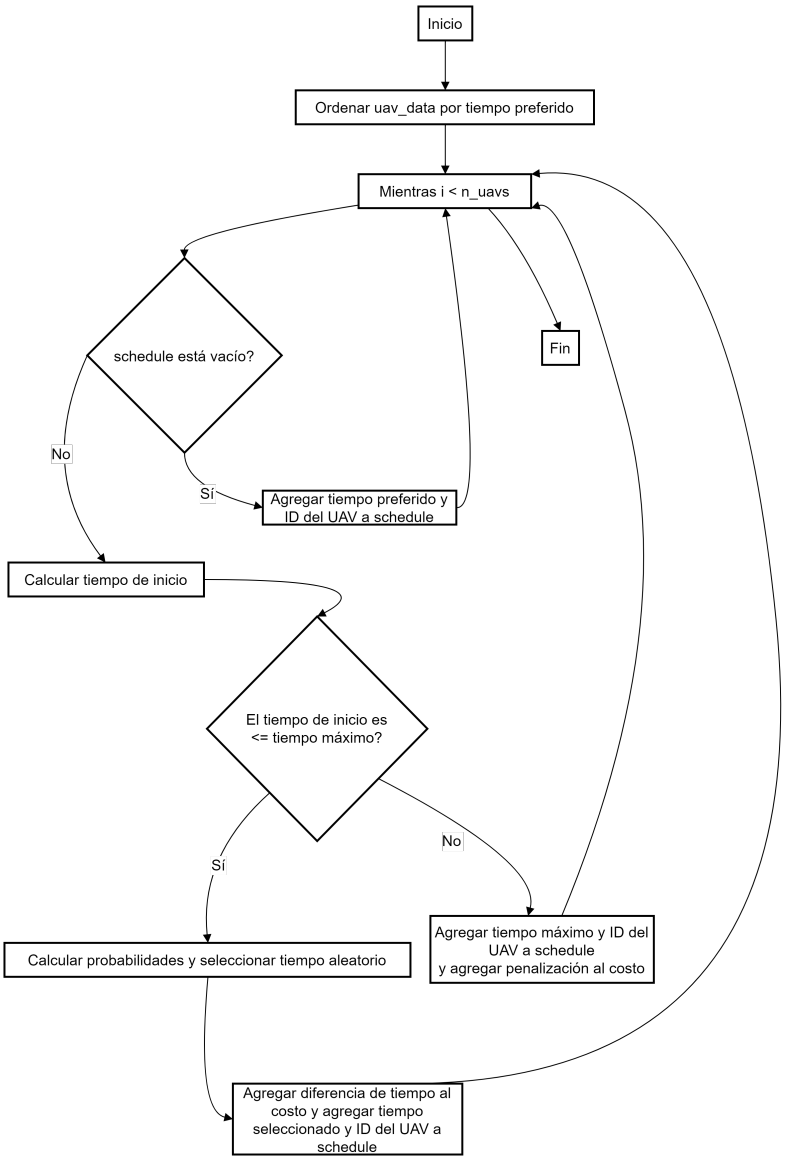

Figura 2: Diagrama de Flujo Greedy Estocástico

3.3. Hill Climbing Primera Mejora

Este código utiliza de entrada la programación de los aterrizajes de las UAVs de los algoritmos anteriores y hace lo siguiente:

1. El código toma cinco argumentos: n_uavs (el número de UAVs), uav_data (una lista de tuplas, donde cada tupla contiene el tiempo mínimo, el tiempo preferido, el tiempo máximo y el ID para cada UAV), $separation\_times$ (una matriz que contiene los tiempos de separación entre cada par de UAVs), $initial\_schedule$ (la programación inicial de los UAVs) y $initial\_cost$ (el costo inicial de la programación).
2. Inicializa $current\_schedule$ a $initial\_schedule$ y $current\_cost$ a $initial\_cost$.
3. Luego, entra en un bucle infinito, donde hace lo siguiente:
   a) Para cada par de UAVs en $current\_schedule$, intercambia sus posiciones y recalcula la programación y el costo.
   b) Si la nueva programación es válida y su costo es menor que $current\_cost$, actualiza $current\_schedule$ y $current\_cost$ a la nueva programación y su costo, y rompe el bucle interno.
   c) Si la nueva programación no es mejor, intercambia las posiciones de los UAVs de nuevo para revertir el cambio.
4. Si después de revisar todos los pares de UAVs no se encontró una mejor programación, devuelve $current\_schedule$ y $current\_cost$.

Pseudocódigo:

```
1: procedure FirstChoiceHillClimbing(n_uavs, uav_data, separation_times, initial_schedule, initial_cost)
2:     current_schedule ← initial_schedule
3:     current_cost ← initial_cost
4:     while True do
5:         for i ← 0 to n_uavs − 1 do
6:             for j ← i + 1 to n_uavs do
7:                 swap_positions(current_schedule, i, j)
8:                 new_schedule ← recalculate_schedule(uav_data, separation_times, current_schedule)
9:                 if new_schedule ≠ None then
10:                    new_cost ← calculate_cost(uav_data, new_schedule)
11:                    if new_cost < current_cost then
12:                        current_schedule ← new_schedule
13:                        current_cost ← new_cost
14:                        break
15:                    end if
16:                end if
17:                swap_positions(current_schedule, i, j)
18:            end for
19:            if new_cost ≥ current_cost then
20:                continue
21:            else
22:                break
23:            end if
24:        end for
25:        if new_cost ≥ current_cost then
26:            return current_schedule, current_cost
27:        end if
28:    end while
29: end procedure
```

Para entender mejor el procedimiento del código, la siguiente figura es un diagrama de flujo que permite captar la idea.

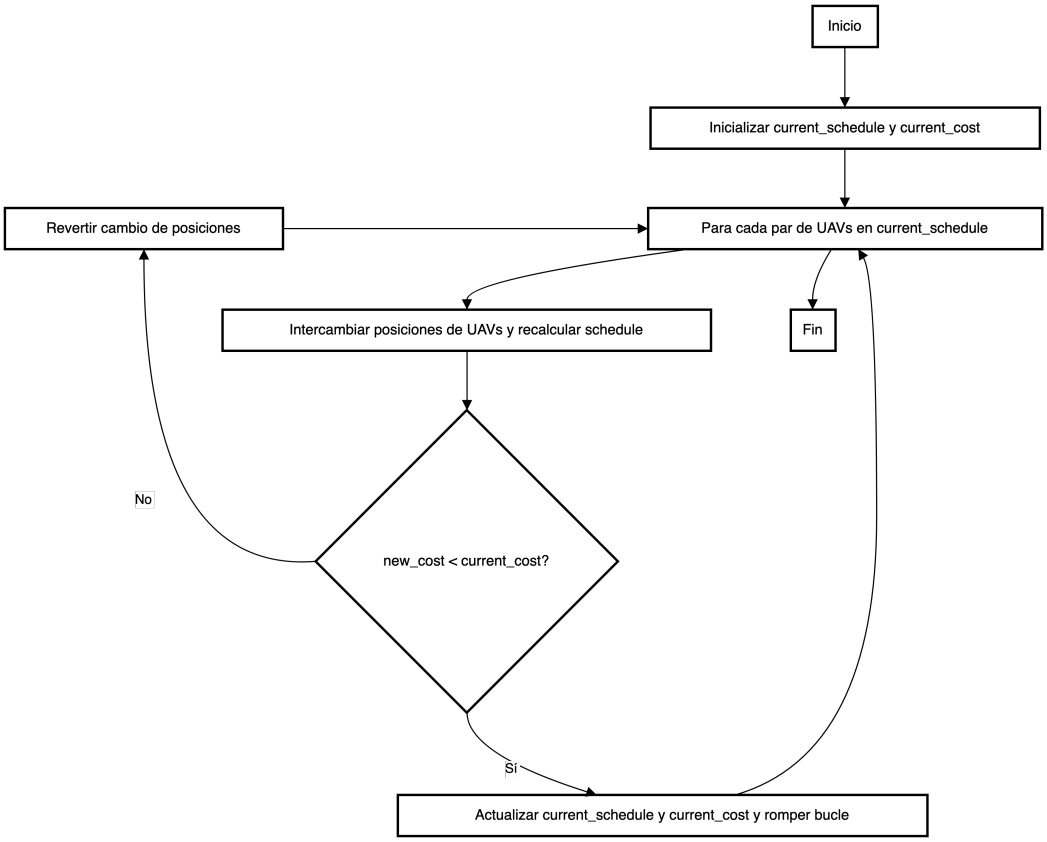

Figura 3: Diagrama de Flujo Hill Climbing Alguna Mejora

3.4. Hill Climbing Mejor Mejora

A diferencia del algoritmo anterior, el Mejor-Mejora no se detiene al encontrar un mejor costo, sino que sigue indagando en el vecindario generado hasta encontrar el mínimo local.

1. El código toma cinco argumentos: $n_$uavs (el número de UAVs), $uav_$data (una lista de tuplas, donde cada tupla contiene el tiempo mínimo, el tiempo preferido, el tiempo máximo y el ID para cada UAV), $separation_$times (una matriz que contiene los tiempos de separación entre cada par de UAVs), $initial_$schedule (la programación inicial de los UAVs) y $initial_$cost (el costo inicial de la programación).
2. Inicializa $current_$schedule a $initial_$schedule y $current_$cost a $initial_$cost.
3. Luego, entra en un bucle infinito, donde hace lo siguiente:
   a) Para cada par de UAVs en $current_$schedule, intercambia sus posiciones y recalcula la programación y el costo.
   b) Si la nueva programación es válida y su costo es menor que $current_$cost, actualiza $current_$schedule y $current_$cost a la nueva programación y su costo, y rompe el bucle interno.
   c) Si la nueva programación no es mejor, intercambia las posiciones de los UAVs de nuevo para revertir el cambio.
4. Si después de revisar todos los pares de UAVs no se encontró una mejor programación, devuelve $current_$schedule y $current_$cost.

El Pseudocódigo del algoritmo es el siguiente:

1: **procedure** BestChoiceHillClimbing($n\_uavs, uav\_data, separation\_times, initial\_schedule, initial\_cost$)
2: $current\_schedule \leftarrow initial\_schedule$
3: $current\_cost \leftarrow initial\_cost$

```
    while True do
        best_schedule ← current_schedule
        best_cost ← current_cost
        for i ← 0 to n_uavs − 1 do
            for j ← i + 1 to n_uavs do
                swap_positions(current_schedule, i, j)
                new_schedule ← recalculate_schedule(uav_data, separation_times, current_schedule)
                if new_schedule ≠ None then
                    new_cost ← calculate_cost(uav_data, new_schedule)
                    if new_cost < best_cost then
                        best_schedule ← new_schedule
                        best_cost ← new_cost
                    end if
                end if
                swap_positions(current_schedule, i, j)
            end for
        end for
        if best_cost < current_cost then
            current_schedule ← best_schedule
            current_cost ← best_cost
        else
            return current_schedule, current_cost
        end if
    end while
end procedure
```

El diagrama de flujo del algoritmo es el siguiente:

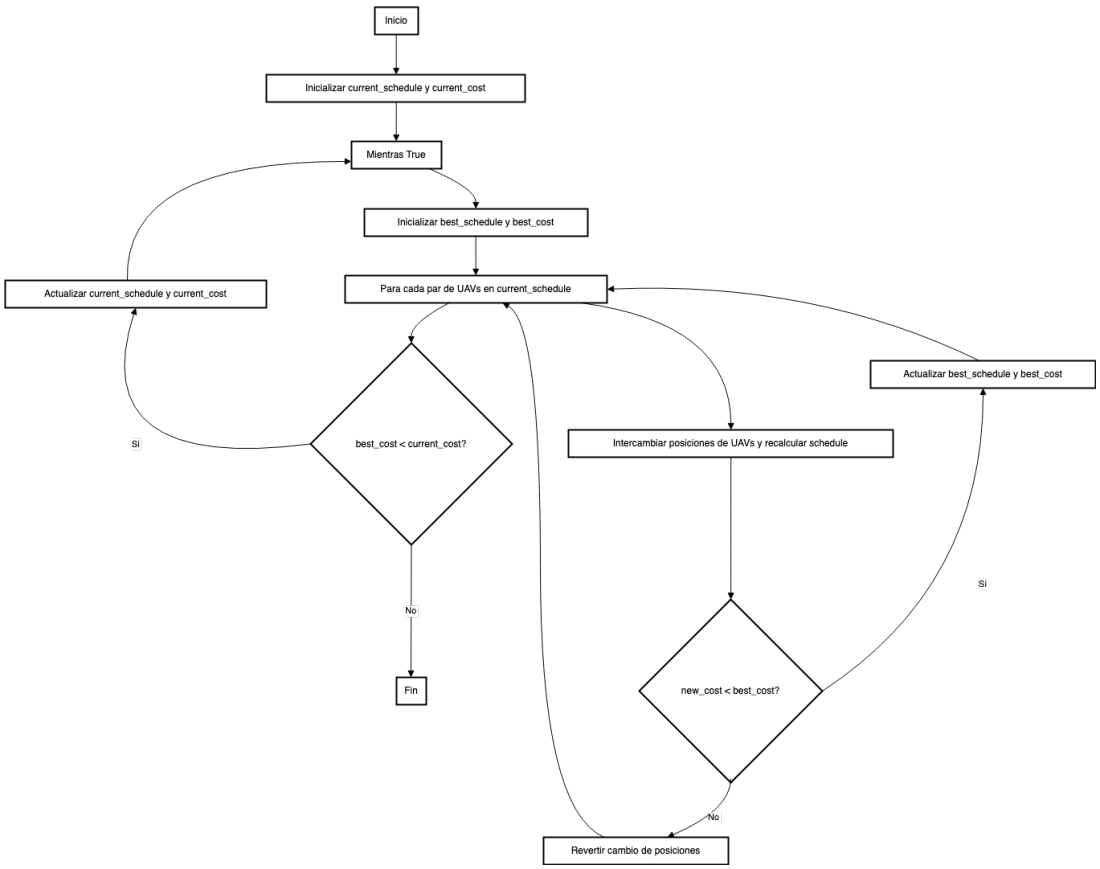

Figura 4: Diagrama de Flujo Hill Climbing Mejor Mejora

3.5. Tabu Search

El Tabú Search es una metaheurística que guía una búsqueda local en la solución de problemas de optimización combinatoria. Utiliza una estructura de memoria, llamada Tabu List, para evitar que la búsqueda vuelva a soluciones previamente visitadas, lo que ayuda a escapar de los óptimos locales.

1. El código toma siete argumentos: *n_uavs* (el número de UAVs), *uav_data* (una lista de tuplas, donde cada tupla contiene el tiempo mínimo, el tiempo preferido, el tiempo máximo y el ID para cada UAV), *separation_times* (una matriz que contiene los tiempos de separación entre cada par de UAVs), *initial_schedule* (la programación inicial de los UAVs), *initial_cost* (el costo inicial de la programación), *max_iterations* (el número máximo de iteraciones para el algoritmo) y *tabu_size* (el tamaño de la lista tabú).

2. Inicializa *current_schedule* a *initial_schedule*, *current_cost* a *initial_cost*, *best_schedule* a *current_schedule*, *best_cost* a *current_cost* y *tabu_list* a una lista vacía.

3. Luego, durante *max_iterations* iteraciones, hace lo siguiente:

   a) Inicializa *best_candidate* a None y *best_candidate_cost* a infinito.

   b) Para cada par de UAVs en *current_schedule*, intercambia sus posiciones y recalcula la programación y el costo.

   c) Si la nueva programación es válida y no está en *tabu_list* (o su costo es menor que *best_cost*), y su costo es menor que *best_candidate_cost*, actualiza *best_candidate* y *best_candidate_cost* a la nueva programación y su costo.

   d) Después de revisar todos los pares de UAVs, si *best_candidate* no es None, actualiza *current_schedule* y *current_cost* a *best_candidate* y *best_candidate_cost*, y agrega *best_candidate* a *tabu_list*. Si *tabu_list* excede *tabu_size*, elimina el elemento más antiguo de *tabu_list*.

   e) Si *current_cost* es menor que *best_cost*, actualiza *best_schedule* y *best_cost* a *current_schedule* y *current_cost*.

4. Finalmente, devuelve *best_schedule* y *best_cost*.

Pseudocódigo del algoritmo:

```
procedure TabuSearch(n_uavs, uav_data, separation_times, initial_schedule, initial_cost, max_iterations, tabu_size)
    current_schedule ← initial_schedule
    current_cost ← initial_cost
    best_schedule ← current_schedule
    best_cost ← current_cost
    tabu_list ← []
    for _ ← 0 to max_iterations − 1 do
        best_candidate ← None
        best_candidate_cost ← float('inf')
        for i ← 0 to n_uavs − 1 do
            for j ← i + 1 to n_uavs do
                swap_positions(current_schedule, i, j)
                new_schedule ← recalculate_schedule(uav_data, separation_times, current_schedule)
                if new_schedule ≠ None and (new_schedule not in tabu_list or calculate_cost(uav_data, new_schedule) <
best_cost) then
                    new_cost ← calculate_cost(uav_data, new_schedule)
                    if new_cost < best_candidate_cost then
                        best_candidate ← new_schedule
                        best_candidate_cost ← new_cost
                    end if
                end if
                swap_positions(current_schedule, i, j)
            end for
        end for
        if best_candidate ≠ None then
            current_schedule ← best_candidate
            current_cost ← best_candidate_cost
```

```
            tabu_list.append(best_candidate)
            if len(tabu_list) > tabu_size then
                tabu_list.pop(0)
            end if
            if current_cost < best_cost then
                best_schedule ← current_schedule
                best_cost ← current_cost
            end if
        end if
    end for
    return best_schedule, best_cost
end procedure
```

Diagrama de Flujos para Tabu Search:

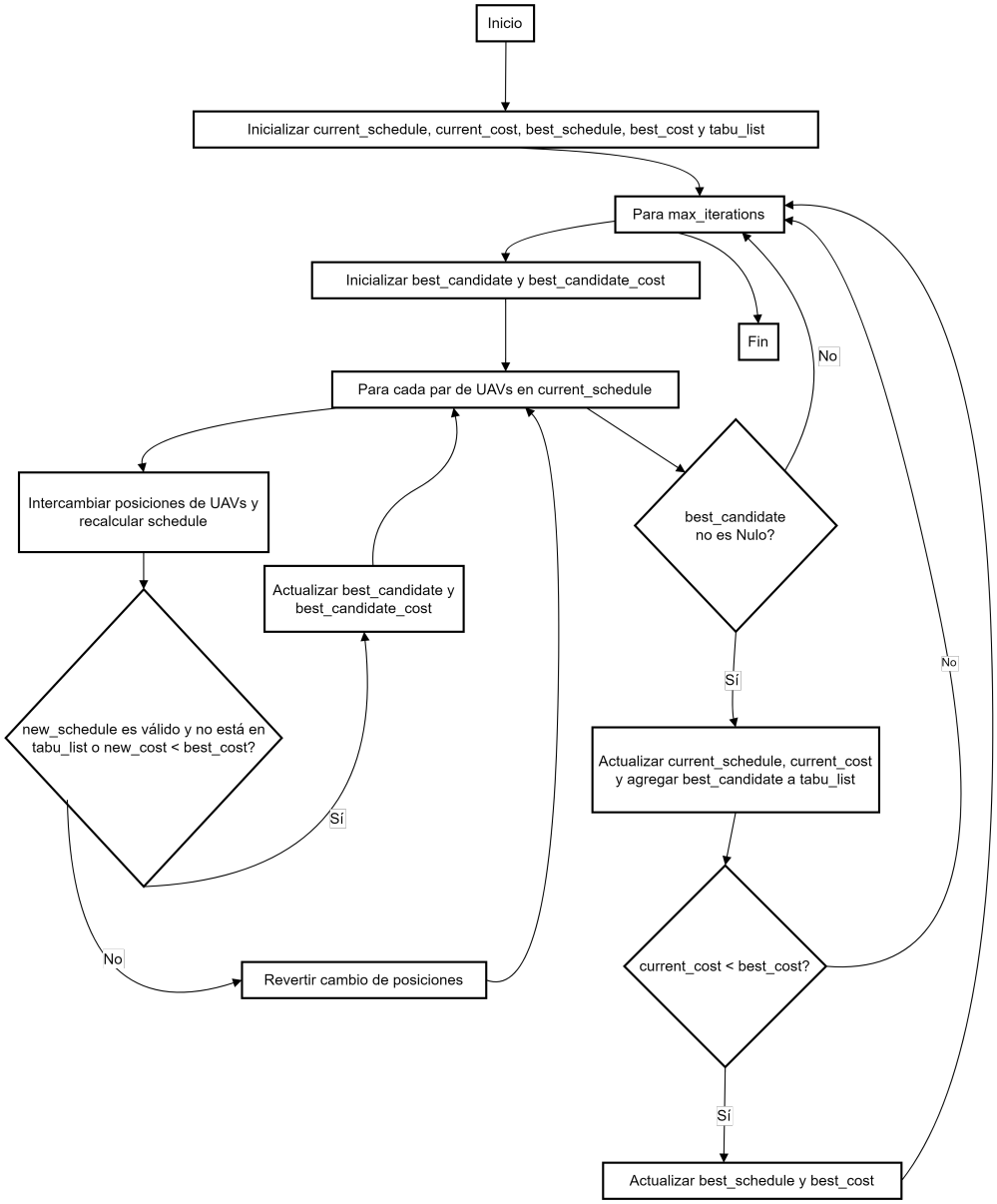

Figura 5: Diagrama de Flujo Tabu Search

4. Resultados

Algoritmo utilizado	**Costo**	**Factible**	**Tiempo (ms)**
Archivo Titan			
Deterministic Greedy Schedule	314	TRUE	0.027895
Stochastic Greedy Schedule (Seed 42)	4773	FALSE	1.259327
Stochastic Greedy Schedule (Seed 45)	3118	TRUE	1.70064
Stochastic Greedy Schedule (Seed 47)	4253	TRUE	1.18351
Stochastic Greedy Schedule (Seed 48)	5415	FALSE	1.029015
Stochastic Greedy Schedule (Seed 51)	5144	FALSE	1.257896
Hill Climbing first choice (Deterministic Greedy)	245	TRUE	8.163929
Hill Climbing first choice (Stochastic Greedy, Seed 42)	960	TRUE	8.401871
Hill Climbing first choice (Stochastic Greedy, Seed 45)	370	TRUE	10.22744
Hill Climbing first choice (Stochastic Greedy, Seed 47)	423	TRUE	10.68068
Hill Climbing first choice (Stochastic Greedy, Seed 48)	702	TRUE	11.80363
Hill Climbing first choice (Stochastic Greedy, Seed 51)	361	TRUE	9.842396
Hill Climbing steepest ascent (Deterministic Greedy)	245	TRUE	4.099607
Hill Climbing steepest ascent (Stochastic Greedy, Seed 42)	245	TRUE	7.175922
Hill Climbing steepest ascent (Stochastic Greedy, Seed 45)	245	TRUE	7.214546
Hill Climbing steepest ascent (Stochastic Greedy, Seed 47)	245	TRUE	6.852865
Hill Climbing steepest ascent (Stochastic Greedy, Seed 48)	245	TRUE	7.256269
Hill Climbing steepest ascent (Stochastic Greedy, Seed 51)	245	TRUE	7.270336
Tabu Search (Deterministic Greedy)	245	TRUE	1797.777
Tabu Search (Stochastic Greedy, Seed 42)	245	TRUE	1827.961
Tabu Search (Stochastic Greedy, Seed 45)	245	TRUE	1799.741
Tabu Search (Stochastic Greedy, Seed 47)	245	TRUE	1800.273
Tabu Search (Stochastic Greedy, Seed 48)	245	TRUE	1805.483
Tabu Search (Stochastic Greedy, Seed 51)	245	TRUE	1828.832
Archivo Europa			
Deterministic Greedy Schedule	8027	TRUE	0.056267
Stochastic Greedy Schedule (Seed 42)	8027	TRUE	0.214815
Stochastic Greedy Schedule (Seed 45)	8027	TRUE	0.23818
Stochastic Greedy Schedule (Seed 47)	8027	TRUE	0.195742
Stochastic Greedy Schedule (Seed 48)	8027	TRUE	0.276804
Stochastic Greedy Schedule (Seed 51)	8027	TRUE	0.19908
Hill Climbing first choice (Deterministic Greedy)	8027	TRUE	4.346371
Hill Climbing first choice (Stochastic Greedy, Seed 42)	8027	TRUE	4.132509
Hill Climbing first choice (Stochastic Greedy, Seed 45)	8027	TRUE	4.030704
Hill Climbing first choice (Stochastic Greedy, Seed 47)	8027	TRUE	4.053831
Hill Climbing first choice (Stochastic Greedy, Seed 48)	8027	TRUE	4.357815
Hill Climbing first choice (Stochastic Greedy, Seed 51)	8027	TRUE	4.277945
Hill Climbing steepest ascent (Deterministic Greedy)	8027	TRUE	4.721642
Hill Climbing steepest ascent (Stochastic Greedy, Seed 42)	8027	TRUE	4.2696
Hill Climbing steepest ascent (Stochastic Greedy, Seed 45)	8027	TRUE	4.425049
Hill Climbing steepest ascent (Stochastic Greedy, Seed 47)	8027	TRUE	4.496813
Hill Climbing steepest ascent (Stochastic Greedy, Seed 48)	8027	TRUE	4.302263
Hill Climbing steepest ascent (Stochastic Greedy, Seed 51)	8027	TRUE	3.965616
Tabu Search (Deterministic Greedy)	8027	TRUE	4073.291
Tabu Search (Stochastic Greedy, Seed 42)	8027	TRUE	3955.737
Tabu Search (Stochastic Greedy, Seed 45)	8027	TRUE	3928.908
Tabu Search (Stochastic Greedy, Seed 47)	8027	TRUE	4106.291
Tabu Search (Stochastic Greedy, Seed 48)	8027	TRUE	4105.822
Tabu Search (Stochastic Greedy, Seed 51)	8027	TRUE	4194.719

Algoritmo utilizado	**Costo**	**Factible**	**Tiempo (ms)**
Archivo Deimos			
Deterministic Greedy Schedule	14993	TRUE	0.164032
Stochastic Greedy Schedule (Seed 42)	159804	FALSE	11.47747
Stochastic Greedy Schedule (Seed 45)	162840	FALSE	12.04801
Stochastic Greedy Schedule (Seed 47)	173189	FALSE	11.73425
Stochastic Greedy Schedule (Seed 48)	166689	FALSE	10.04386
Stochastic Greedy Schedule (Seed 51)	175832	FALSE	11.23238
Hill Climbing first choice (Deterministic Greedy)	9261	TRUE	5787.539
Hill Climbing first choice (Stochastic Greedy, Seed 42)	10123	TRUE	5755.202
Hill Climbing first choice (Stochastic Greedy, Seed 45)	10123	TRUE	5743.514
Hill Climbing first choice (Stochastic Greedy, Seed 47)	10123	TRUE	5778.286
Hill Climbing first choice (Stochastic Greedy, Seed 48)	10123	TRUE	5791.058
Hill Climbing first choice (Stochastic Greedy, Seed 51)	10123	TRUE	5821.645
Hill Climbing steepest ascent (Deterministic Greedy)	8572	TRUE	4537.238
Hill Climbing steepest ascent (Stochastic Greedy, Seed 42)	9556	TRUE	4751.2
Hill Climbing steepest ascent (Stochastic Greedy, Seed 45)	9556	TRUE	4759.331
Hill Climbing steepest ascent (Stochastic Greedy, Seed 47)	9556	TRUE	4733.681
Hill Climbing steepest ascent (Stochastic Greedy, Seed 48)	9556	TRUE	4749.956
Hill Climbing steepest ascent (Stochastic Greedy, Seed 51)	9556	TRUE	5042.418
Tabu Search (Deterministic Greedy)	8567	TRUE	19742.01
Tabu Search (Stochastic Greedy, Seed 42)	9443	TRUE	19100.75
Tabu Search (Stochastic Greedy, Seed 45)	9443	TRUE	19067.91
Tabu Search (Stochastic Greedy, Seed 47)	9443	TRUE	19099.35
Tabu Search (Stochastic Greedy, Seed 48)	9443	TRUE	19064.3
Tabu Search (Stochastic Greedy, Seed 51)	9443	TRUE	19088.87

Cuadro 1: Resultados

En la tabla anterior se pueden observar los resultados de los distintos archivos o casos propuestos para el problema. Se destacan el algoritmo utilizado, el costo obtenido, si la solución resultante es válida considerando los tiempos de espera junto a las ventanas de tiempo y el tiempo de ejecución que le tomó al algoritmo poder resolverlo.

5. Análisis

En el contexto del problema de programación de aterrizajes de Vehículos Autónomos No Tripulados (UAVs) en colonias espaciales, se han implementado y evaluado varios algoritmos y metaheurísticas. Cada uno de estos algoritmos tiene características inherentes que influyen en su tiempo de ejecución y en la calidad de las soluciones que proporcionan. A continuación, se presenta un análisis detallado de los tiempos de ejecución de cada algoritmo, considerando su naturaleza y cómo se justifican estos tiempos en el contexto del problema.

- **Greedy Determinístico y Estocástico**: Estos algoritmos son conocidos por su eficiencia y rapidez, ya que buscan soluciones aceptables de manera rápida, pero no necesariamente óptimas. En el problema de programación de aterrizajes de UAVs, estos algoritmos pueden proporcionar soluciones rápidas al asignar tiempos de aterrizaje basándose en criterios simples, como el tiempo preferido de aterrizaje de cada UAV. Sin embargo, estas soluciones pueden no ser óptimas, especialmente en situaciones donde las restricciones de tiempo y separación son complejas. Los tiempos de ejecución registrados para estos algoritmos son muy bajos, lo que refleja su naturaleza eficiente.
- **Hill Climbing (Primera Mejora y Mejor Mejora)**: Estos algoritmos son más sofisticados que los algoritmos Greedy y buscan mejorar la solución actual moviéndose a un estado vecino que tenga un mejor resultado. En el contexto del problema de programación de aterrizajes de UAVs, estos algoritmos pueden explorar más a fondo el espacio de soluciones al considerar diferentes secuencias de aterrizaje y buscar aquellas que minimicen el costo total. Sin embargo, esta exploración más exhaustiva también conlleva tiempos de ejecución más largos. La versión de Mejor Mejora, que busca la mejor opción en todo el vecindario antes de hacer un movimiento, tiende a tener tiempos de ejecución más largos que la versión de Primera Mejora, que se mueve a la primera opción que mejora la solución actual.
- **Tabu Search**: Este es el algoritmo más sofisticado y también el que tiene los tiempos de ejecución más largos. Tabu Search es una metaheurística que permite la exploración de la región de soluciones más allá de los óptimos locales mediante el uso de una lista tabú que prohíbe o penaliza el movimiento a soluciones previamente visitadas. En el problema de programación de aterrizajes de UAVs, este algoritmo puede explorar a fondo el espacio de soluciones y encontrar soluciones de alta calidad que otros algoritmos pueden pasar por alto. Sin embargo, esta exploración exhaustiva también conlleva tiempos de ejecución más largos.

En resumen, los tiempos de ejecución de los algoritmos reflejan su complejidad y la profundidad de su exploración del espacio de soluciones. Los algoritmos Greedy son los más rápidos pero pueden no encontrar la solución óptima, mientras que Hill Climbing y Tabu Search realizan una exploración más exhaustiva del espacio de soluciones a costa de tiempos de ejecución más largos. Sin embargo, es importante tener en cuenta que la calidad de las soluciones es un factor crucial en este problema, ya que el objetivo es minimizar el costo total de los aterrizajes de los UAVs. Por lo tanto, un tiempo de ejecución más largo puede ser justificable si el algoritmo es capaz de encontrar una solución de mayor calidad. Para ello a continuación se presenta con mayo detalle respecto a la calidad de la solución encontrada.

- **Greedy Determinístico y Estocástico**: Para el archivo Europa, ambos algoritmos Greedy proporcionaron la misma solución con un costo de 8027. Para Deimos y Titan, los resultados variaron, con el Greedy Determinístico proporcionando soluciones factibles con costos de 14993 y 314 respectivamente, mientras que el Greedy Estocástico proporcionó soluciones con costos que variaban dependiendo de la semilla utilizada.
- **Hill Climbing (Primera Mejora y Mejor Mejora)**: Ambas variantes de Hill Climbing proporcionaron la misma solución con un costo de 8027 para el archivo Europa. Para Deimos y Titan, los costos de las soluciones variaron dependiendo de la variante de Hill Climbing y la solución inicial utilizada, pero en general, estas variantes proporcionaron soluciones de mayor calidad que los algoritmos Greedy.
- **Tabu Search**: Al igual que los otros algoritmos, Tabu Search proporcionó una solución con un costo de 8027 para el archivo Europa. Para Deimos y Titan, Tabu Search proporcionó soluciones de alta calidad con costos que variaban dependiendo de la solución inicial utilizada.

En general, estos resultados sugieren que la naturaleza del problema en el archivo Europa puede ser tal que la solución óptima es fácilmente alcanzable por todos los algoritmos. Sin embargo, para los archivos Deimos y Titan, se observa una mejora en la calidad de las soluciones al pasar de los algoritmos Greedy a Hill Climbing y luego a

Tabu Search, lo que refleja la capacidad de estos algoritmos más sofisticados para explorar más a fondo el espacio de soluciones y encontrar soluciones de mayor calidad.

Además, se observa que los algoritmos que se basan en soluciones previas de Greedy Estocástico tienden a proporcionar soluciones de mayor calidad. Esto puede ser debido a que el Greedy Estocástico puede explorar diferentes partes del espacio de soluciones en cada ejecución debido a su elemento de aleatoriedad, lo que puede conducir a soluciones iniciales de mayor calidad que luego pueden ser mejoradas por algoritmos como Hill Climbing y Tabu Search.

En resumen, aunque los tiempos de ejecución son un aspecto importante a considerar al evaluar los algoritmos, la calidad de las soluciones que proporcionan es igualmente crucial. En este caso, todos los algoritmos fueron capaces de encontrar la misma solución óptima para el archivo Europa, pero se observaron diferencias en la calidad de las soluciones para los archivos Deimos y Titan. Estas diferencias reflejan la capacidad de los algoritmos más sofisticados para explorar más a fondo el espacio de soluciones y encontrar soluciones de mayor calidad, especialmente en problemas más complejos.

Otro aspecto importante a considerar es la manera en que se explora el espacio de búsqueda, puesto que los algoritmos estocásticos pueden partir de soluciones no válidas ni óptimas, es importante lograr a través de los algoritmos de hill climbing y Tabu Search que puedan iterar y explorar de tal manera de siempre adquirir soluciones factibles. Basándonos en los resultados obtenidos a partir de soluciones estocásticas, se puede observar como a medida que los algoritmos se vuelven más sofisticado, poseen una mayor capacidad de exploración y posibilidad de evitar mínimos locales.

Esto último se puede observar al comparar las diferencias entre el *hill climbing* de primer mejora y mejor mejora, como se observa en la figura 6. Aunque para ambas respuestas ya se encuentran dentro del dominio de soluciones válidas, la versión de mejor mejora aún así no es capaz de obtener un mejor rendimiento que el algoritmo de Tabu Search, probablemente debido a que tienen un mayor desafío encontrando alguna localidad válida en un comienzo en vez de seguir iterando para optimizar el costo. Esto puede deberse al uso de su propia Tabu List, que le permite descartar operaciones de movimiento repetidas explorando así soluciones peores, a diferencia de los hill climbing que siguen buscando óptimos con la única condición de ser soluciones válidas.

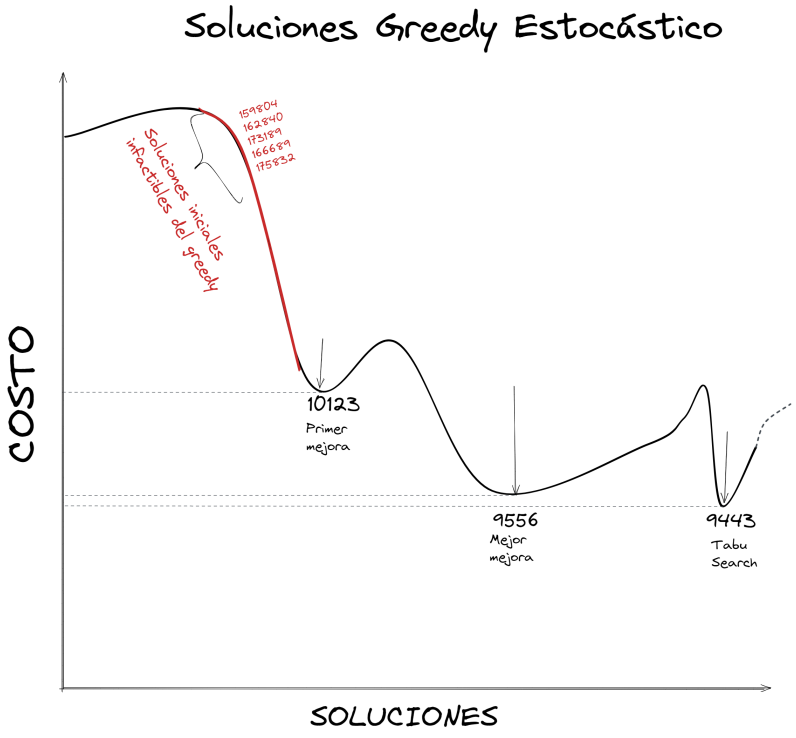

Figura 6: Gráfico explicativo de resultados Deimos

6. Conclusión

La actividad abarcaba desde la forma de interpretar el problema, los diseños y estudio de distintos tipos de soluciones, al igual que la manera de interpretar el caso propuesto a las operaciones propias de cada tipo de algoritmo utilizado. La aplicación y diseño del movimiento basado en la reasignación o rescheduling, no solo nos permite explorar de mejor manera el espacio de búsqueda, ya que a través de las implementaciones realizadas y los resultados obtenidos es posible determinar una solución al problema planteado de manera genérica para cada algoritmo, sino también comprender e interpretar el impacto que puede tener ciertos conjuntos de soluciones. Con esto último se considera también el comportamiento que puedan tener las diferentes técnicas para moverse a través del espacio solución y no solo desde el punto de vista del costo en el tiempo o de la calidad de la solución.