Sección de Matemáticas
Universidad de La Laguna

Elena Martín Raya

Optimización combinatoria mediante computación evolutiva

Combinatorial optimisation through evolutionary computation

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Julio de 2024

Dirigido por
Eduardo Manuel Segredo González

Eduardo Manuel Segredo González
Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Me gustaría agradecer a todas esas personas que de una forma u otra han ayudado para que haya llegado hasta aquí. En especial, a mi familia por apoyarme a lo largo de estos últimos años. A mis amigos, por escucharme y animarme siempre sin tan siquiera entender de qué les hablaba, estaré siempre agradecida. A mis compañeros de la carrera, ya amigos, por apoyarnos y hacer mucho más amena, divertida y mejor esta etapa. Y, sobre todo, a mi tutor Eduardo, por ayudarme a lo largo de todo este trabajo. Su paciencia, conocimientos y predisposición por conseguir el mejor resultado han hecho posible esto.

Elena Martín Raya
La Laguna, 5 de julio de 2024

Resumen · Abstract

Resumen

El problema "Quantum Communications Constellations" es un desafío clave para establecer canales de comunicación confiables entre la superficie de Nuevo Marte, donde se desplegarán los rovers, y las naves nodrizas en órbita. Para este problema de optimización combinatoria multi-objetivo, planteado por la Agencia Espacial Europea (ESA), se ha llevado a cabo, en primer lugar, una ponderación de las funciones objetivo para, luego, implementar y aplicar tres métodos de resolución mono-objetivo diferentes: Búsqueda Aleatoria (BA), Búsqueda Local (BL) y Algoritmo Memético (AM). Además, los resultados han sido comparados estadísticamente para identificar cuál de los algoritmos proporciona mejores resultados.

El objetivo principal de este trabajo ha sido, por lo tanto, estudiar si el AM es capaz de superar, en términos de rendimiento, a la BA y la BL. La métrica seleccionada para comparar las aproximaciones al Frente de Pareto proporcionadas por los tres algoritmos ha sido el hipervolúmen. El análisis de este trabajo concluye que el rendimiento del AM, en términos de las soluciones proporcionadas, es significativamente superior al de los otros dos algoritmos, corroborando la hipótesis de partida. Este estudio contribuye a la optimización en comunicaciones cuánticas, proporcionando una comparación exhaustiva de metodologías y destacando el rendimiento de la computación evolutiva.

Palabras clave: *Optimización Combinatoria – Computación Evolutiva – Algoritmo Memético – Búsqueda Local – Búsqueda Aleatoria*

Abstract

The "Quantum Communications Constellations" problem is a key challenge for establishing reliable communication channels between the surface of New Mars, where rovers will be deployed, and the motherships in orbit. For this multi-objective combinatorial optimization problem, posed by the European Space Agency (ESA), an initial weighting of the objective functions was carried out. Then, three different single-objective resolution methods were implemented and applied: Random Search (RS), Local Search (LS), and Memetic Algorithm (MA). Additionally, the results were statistically compared to identify which of the algorithms provides better results.
The main objective of this work was, therefore, to study whether the MA can outperform the RS and LS in terms of performance. The metric selected to compare the approximations to the Pareto Front provided by the three algorithms was the hypervolume. The analysis of this work concludes that the performance of the MA, in terms of the solutions provided, is significantly superior to that of the other two algorithms, corroborating the initial hypothesis. This study contributes to optimization in quantum communications, providing a comprehensive comparison of methodologies and highlighting the performance of evolutionary computation.

Keywords: *Combinatorial Optimization – Evolutionary Computation – Memetic Algorithm – Local Search – Random Search*

Contenido

1

Introducción

En este trabajo aborda un problema de optimización combinatoria multiobjetivo titulado *Quantum Communications Constellations* planteado como desafío por la Agencia Espacial Europea (ESA, su acrónimo en inglés). Dicho problema trata de establecer comunicaciones entre los *rovers* que se encuentran en la superficie de Nuevo Marte y las naves nodrizas en órbita. Se propone utilizar redes de comunicaciones cuánticas en órbita, aprovechando la comunicación láser y constelaciones de satélites cuánticos. Sin embargo, la eficiencia de estos enlaces depende de la distancia entre los satélites y conlleva importantes costos operativos. Se busca optimizar el diseño de las constelaciones para maximizar la eficiencia de las comunicaciones sin aumentar excesivamente los costos operativos. Se necesita ayuda para encontrar configuraciones orbitales óptimas y libres de colisiones. La computación evolutiva es una herramienta útil para resolver este problema de optimización. Este tipo de computación se basa en la evolución biológica para proporcionar técnicas que resuelvan el problema encontrando soluciones eficientes.

1.1. Optimización combinatoria

Un problema de **optimización combinatoria** [1] es un tipo de problema de optimización que se caracteriza por tener un espacio de soluciones discreto. En estos problemas, el modelo y la salida del problema son conocidos y la tarea consiste en determinar las entradas necesarias para obtener los resultados deseados.

Figura 1.1: Esquema de un problema de optimización

Además, en función de cuántos objetivos tenga el problema podemos diferenciar dos tipos:

- **Problemas de optimización combinatoria mono-objetivo** [2]: se tratan de problemas que tienen un único objetivo. Se definen de la siguiente manera:

  min or max: $f(x)$ sujeto a $g_i(x) \leq 0$, $i = \{1, \ldots, m\}$ y $h_j(x) = 0$, $j = \{1, \ldots, p\}$, x $\in \Omega$.

  Se trata de minimizar o maxizar $f(x)$. Donde $x = (x_1, \ldots, x_n)$ es la variable de decisión, un vector n-dimensional en el espacio Ω. Las funciones $g_i(x)$ y $h_j(x)$ representan las restricciones que deben cumplirse al optimizar $f(x)$, y Ω contiene todas las posibles soluciones x que satisfacen $f(x)$ y sus restricciones.

- **Problemas de optimización combinatoria multi-objetivo** [2]: se tratan de problemas que tienen dos o más objetivos. Se definen de la siguiente manera:

  min or max: $F(x) = (f_1(x), \ldots, f_k(x))$ sujeto a $g_i(x) \leq 0$, $i = \{1, \ldots, m\}$ y $h_j(x) = 0$, $j = \{1, ..., p\}$, x $\in \Omega$.

  En este tipo de problemas se minimizan o maximizan las componentes de F(x). Donde $x = (x_1, \ldots, x_n)$ es la variable de decisión, un vector n-dimensional del espacio Ω. Las funciones $g_i(x)$ y $h_j(x)$ representan las restricciones que deben cumplirse al optimizar $F(x)$ y Ω contiene todas las posible soluciones x que satisfacen $F(x)$ y sus restricciones.

1.2. Computación evolutiva

La computación evolutiva es un área de investigación en Ciencias de la Computación que se inspira en el proceso natural de evolución. Su idea principal es que, al igual que la evolución ha permitido que las especies se adapten a sus entornos, también puede aplicarse para resolver problemas a través de ensayo y error. Los científicos adoptan este enfoque porque ven en la evolución natural un modelo exitoso de adaptación y supervivencia en diversos entornos, utilizando esta idea para crear métodos efectivos de resolución de problemas.

1.2.1. Breve historia

La Computación Evolutiva [1] se originó en la década de 1930 con Sewell Wright, quien la influyó al explicar la evolución como un proceso de cambio genético continuo mediante prueba y error para adaptarse al entorno. Walter Cannon también observó la evolución como aprendizaje a través de pruebas aleatorias y errores. Aunque fue Hans Bremermann, a finales de los años 50, quien relacionó la evolución con la optimización matemática, siendo pionero en la Computación Evolutiva.

Se exploraron diferentes enfoques en el campo de la Computación Evolutiva. Estos abordaron cómo utilizar procesos evolutivos para resolver problemas de optimización y aplicar estas ideas en la Inteligencia Artificial. Además, se intentó crear sistemas adaptativos sólidos e independientes, basados en la retroalimentación de su interacción con un entorno específico. La investigación se centró en mejorar la implementación de estos enfoques, la gestión de la población de individuos y la selección de padres para la descendencia. Como resultado, se identificaron tres Algoritmos Evolutivos principales: Algoritmos Genéticos, Estrategias Evolutivas y Programación Evolutiva.

En la actualidad, la Computación Evolutiva continúa expandiéndose hacia nuevos campos de aplicación, como la ingeniería, la economía y la biología computacional, donde se utilizan para resolver problemas complejos de optimización y diseño. Sin embargo, la optimización sigue siendo su principal campo de aplicación.

1.2.2. Algoritmos evolutivos

Como ya se ha mencionado existen diferentes algoritmos evolutivos. Todos ellos comparten una idea común: dada una población de individuos dentro de un entorno con recursos limitados, la competencia por esos recursos causa una selección natural provocando así un aumento de la aptitud de la población. Por ello, los algoritmos evolutivos siguen el siguiente esquema [1]:

1. **Inicialización:** es el proceso inicial en algoritmos evolutivos donde se establece la población inicial mediante la generación aleatoria de individuos.

2. **Representación (definición de individuos):** se refiere a cómo se asignan los fenotipos (objetos que son soluciones en el problema original) a genotipos o cromosomas. Estos genotipos son la codificación de los objetos y son utilizados para la evolución computacional.

3. **Función de evaluación:** también llamada función de aptitud. Es una función que asigna una medida de calidad a los genotipos, representando los requisitos a los que la población debe adaptarse. Esta función guía la selección y mejora de las soluciones.

4. **Población:** compuesta por posibles soluciones, forma la unidad de evolución. Esta unidad fundamental evoluciona a lo largo del algoritmo, determinando la diversidad y calidad de las soluciones. Está conformada por múltiples genotipos y puede tener un tamaño fijo o incluir una estructura espacial adicional. Los operadores de selección y la diversidad de la población son aspectos cruciales en la búsqueda evolutiva, ya que garantizan la competencia y la exploración efectiva del espacio de soluciones.

5. **Mecanismo de selección de Padres:** es el proceso que elige a los mejores individuos para ser padres en la generación actual, impulsando así mejoras en la calidad de las soluciones. Es un proceso probabilístico que favorece a los individuos de alta calidad mientras que también permite una pequeña oportunidad para los de baja calidad evitando que la búsqueda se vuelva demasiado elitista.

6. **Operadores de variación (recombinación y mutación):** son herramientas utilizadas para crear nuevos individuos a partir de los existentes, lo que permite explorar y modificar el espacio de soluciones. Estos operadores son fundamentales para la evolución de las poblaciones y se dividen en mutación y recombinación.
   - **Mutación:** es un operador unario de variación que se aplica a un genotipo individual para introducir cambios aleatorios y no direccionados, con el fin de explorar nuevas soluciones dentro del espacio de búsqueda. La mutación es crucial para mantener la diversidad genética y evitar la convergencia prematura hacia óptimos locales.
   - **Recombinación:** es un operador binario de variación que combina información de dos genotipos parentales para crear uno o dos nuevos genotipos descendientes. Este proceso emula el cruce genético en la naturaleza y facilita la combinación de características deseables de los padres, contribu-

yendo así a la mejora de la calidad de las soluciones en el proceso evolutivo.

7. **Mecanismo de selección de supervivencia:** determina qué individuos serán conservados en la población de la siguiente generación después de la creación de la descendencia. Este proceso se basa en la calidad de los individuos evaluada mediante sus valores de aptitud. Los individuos de mayor calidad tienen mayores probabilidades de ser seleccionados para la siguiente generación, asegurando así la mejora continua de la población y la evolución hacia soluciones óptimas.

8. **Criterio de parada:** es el criterio que determina cuándo finaliza el proceso de evolución. Puede basarse en alcanzar un nivel óptimo conocido de aptitud, límites de tiempo de CPU, número máximo de evaluaciones de aptitud, falta de mejora en la aptitud, o la caída de la diversidad poblacional bajo un umbral.

En resumen, los algoritmos evolutivos son fundamentales para resolver problemas complejos al simular procesos evolutivos naturales. A través de la inicialización aleatoria, selección de padres y supervivientes, operadores de variación y un criterio de parada definido, exploran eficazmente el espacio de soluciones en busca de óptimos. La elección adecuada de la condición de parada es crucial para garantizar la eficiencia y efectividad de estos algoritmos en la búsqueda de soluciones óptimas en un tiempo razonable.

1.3. Agencia Espacial Europea (ESA) y Genetic and Evolutionary Computation Conference (GECCO)

El problema a tratar proviene de una competición llamada *SpOC: Space Optimisation Competition* [8], organizada por la Agencia Espacial Europea (ESA). En esta competición se proponen varios retos espaciales con distintos niveles de dificultad en diversas disciplinas científicas y de ingeniería, incluyendo el análisis de misiones. Cada uno de estos desafíos está inspirado en el trabajo real de los ingenieros y científicos de la ESA.

Además de esta competición, también se lleva a cabo una conferencia llamada GECCO (*The Genetic and Evolutionary Computation Conference*)[9]. Esta conferencia, que se celebra anualmente desde 1999, presenta los últimos resultados de alta calidad en computación genética y evolutiva. En la edición de 2023, se incluyeron desafíos planteados por la ESA, entre los cuales se encuentra el problema que se va a desarrollar. Este año también cuenta con desafíos planteados por la ESA, aunque son diferentes a los de ediciones anteriores.

Como se trata de una competición, existe una plataforma donde los participantes pueden subir las soluciones que hayan desarrollado. Una vez que se sube

una solución, el sistema la evalúa y asigna una puntuación. Esta puntuación se utiliza para crear un ranking de los participantes en la competición.

1.4. Objetivos

El objetivo principal de este trabajo es estudiar la capacidad de algoritmos evolutivos simples para abordar problemas de optimización combinatoria, así como comparar su eficacia frente a algoritmos más sencillos.

Se emplearán tres métodos diferentes para resolver el problema. En primer lugar, se aplicará una búsqueda aleatoria de soluciones, la cual servirá como punto de referencia. En segundo lugar, se implementará un algoritmo de búsqueda local esperando obtener mejores resultados en comparación con el método aleatorio. Finalmente, se ejecutará un algoritmo memético, combinando el algoritmo evolutivo y la búsqueda local. Confiando encontrar soluciones más óptimas con este último en comparación con los dos métodos anteriores.

Dado que el problema *Quantum Communications Constellations* involucra dos objetivos, los métodos se diseñarán para resolver la suma ponderada de estos, donde los pesos asignados a cada objetivo variarán para ver como influye cada objetivo al intentar optimizar el problema.

Este enfoque permitirá evaluar si un algoritmos evolutivo puede superar el rendimiento de métodos más sencillos, como la búsqueda aleatoria o la búsqueda local. La hipótesis de partida de este trabajo establece una respuesta afirmativa a la pregunta anterior.

1.5. Tecnologías utilizadas

El lenguaje de programación elegido para la implementación de los diferentes algoritmos ha sido Python, debido al conocimiento previo adquirido en diversas asignaturas durante el Grado. Además, el código de evaluación del problema proporcionado por la ESA está escrito en dicho lenguaje [3].

Figura 1.2: Logo Python

Para la programación en Python, se ha utilizado Google Colab, debido a las facilidades que ofrece para la programación [4].

Figura 1.3: Logo GoogleColab

Finalmente, parte de la evaluación experimental se ejecutó en un entorno local. Para ello, se utilizó Anaconda, ya que se requería una biblioteca que dependía de conda [5]. La ejecución en local se llevó a cabo mediante Visual Studio Code [6].

(a) Logo Anaconda

(b) Logo Visual Studio Code

2

Estado del Arte

En este capítulo, se estudiarán las soluciones ya planteadas para nuestro problema *Quantum Constellation Communications*. Estas soluciones fueron presentadas a la ESA en la conferencia GECCO 2023. Entre todas ellas destaca el uso de la librería ***fcmaes*** por ser la número 1, en el ranking de puntuación, y por las diferentes maneras que plantea para resolver el problema: Fcmaes MO-optimization y Optimización mono-objetivo del Frente de Pareto.

2.1. Librería *fcmaes*

Fcmaes se trata de una biblioteca de optimización sin gradientes de Python 3. Esta biblioteca complementa *scipy.optimize* proporcionando métodos de optimización adicionales, implementaciones más rápidas y un mecanismo de reintento paralelo coordinado.

En la resolución de nuestro problema se hicieron diferentes cambios al código de evaluación proporcionado para así conseguir una solución más óptima en menor tiempo. Se realizaron cambios significativos en el código aplicando diferentes algoritmos. A continuación, se explicarán los métodos utilizados siguiendo el propio tutorial de fcmaes [7].

Antes de aplicar los algoritmos se procedió a acelerar el proceso utilizando *numba* e *igraph*. En el código original la función de aptitud utiliza la biblioteca de gráficos *networkx* implementada en Python. Sin embargo, esta no debería usarse para determinar la ruta más corta promedio dentro de una función de aptitud crítica en el tiempo llamada millones de veces ya que puede resultar muy costosa. En cambio, se debe aplicar para este propósito una alternativa más rápida como es *igraph*, para ello es necesario una adaptación del código, adaptando tanto la construcción del gráfico como el cálculo del camino más corto.

Aún más significativo es factorizar otras funciones críticas en el tiempo que realizan operaciones en matrices *numba* para funciones independientes fuera de los objetos. Luego, estos se pueden anotar con *"@njit()"*(No Python JIT)

usando *numba*. Como el código existente se basa en numpy, las nuevas funciones adaptadas son muy similares a su versión original.

Estos pequeños cambios logran alrededor de una velocidad 30 veces mayor sin necesidad de recodificar la función aptitud original.

2.1.1. Fcmaes MO-optimization

El algoritmo *fcmaes MO-optimization* es la combinación del conocido algoritmo NSGA-II junto con un cruce binario simulado y mutación polinómica. Este algoritmo se caracteriza por:

- Evaluar funciones en paralelo.
- Mejorar el Frente de Pareto existente.
- Comunicarse con la interfaz mediante las funciones *.ask* y *.tell*

Este algoritmo es similar al NSGA-II clásico, con algunos ajustes en cuanto a la paralelización. El código utiliza una nueva característica de *MO-optimization* de *fcmaes* usando *set_guess*. Esto permite definir una población inicial como una suposición o un Frente de Pareto que deseas mejorar. El código también funciona con "guess = None" aunque emplea más tiempo en ejecutarse. Además, utilizando la interfaz *ask* podemos verificar el hipervolumen en cada iteración, mientras que con la interfaz *tell* podemos registrar mejoras.

En el diagrama de la izquierda de la Figura 2.1 podemos observar el comportamiento de este algoritmo a lo largo del tiempo con respecto a otros tipos de algoritmos que se explicarán a continuación.

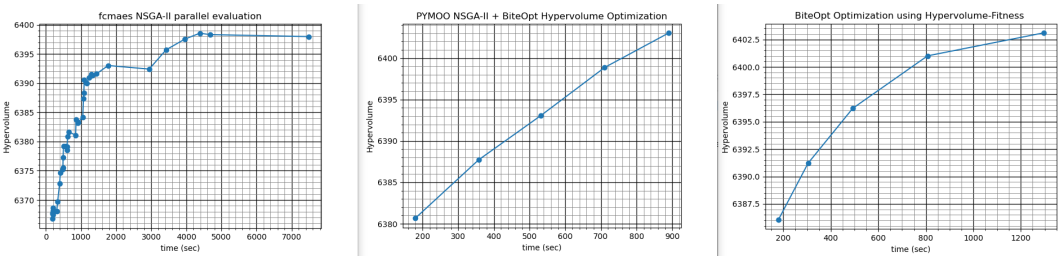


Figura 2.1: Gráfico MO-optimization

2.1.2. Optimización mono-objetivo del Frente de Pareto

Otra manera de solucionar el problema puede ser afrontar una estrategia de descomposición que se ejecuta en relación con las diversas soluciones en el Frente de Pareto. Cada solución se considera como una suposición, y se lleva a cabo una optimización separada para cada una de ellas. Estas optimizaciones se realizan de forma simultánea, dividiendo el proceso en una secuencia de iteraciones. Se elige el hipervolumen de toda la frontera como único objetivo, reemplazando la

solución que mejoramos por las soluciones actuales verificadas por el algoritmo. Para este propósito, se utiliza el algoritmo ***BiteOpt*** como optimizador de objetivo único. Cualquier mejora del Frente de Pareto se refleja inmediatamente al reemplazar la solución correspondiente en el Frente de Pareto compartido.

En el gráfico de la Figura 2.2 se puede ver como el rendimiento de este algoritmo es comparable con el rendimiento Pymoo ya que se puede observar como ambos métodos tienen una barrera con la puntuación -6450. Hay que tener en cuenta que con este algoritmo el Frente de Pareto es mayor, con una frontera de 100 soluciones se obtendrá una puntuación ligeramente más baja.

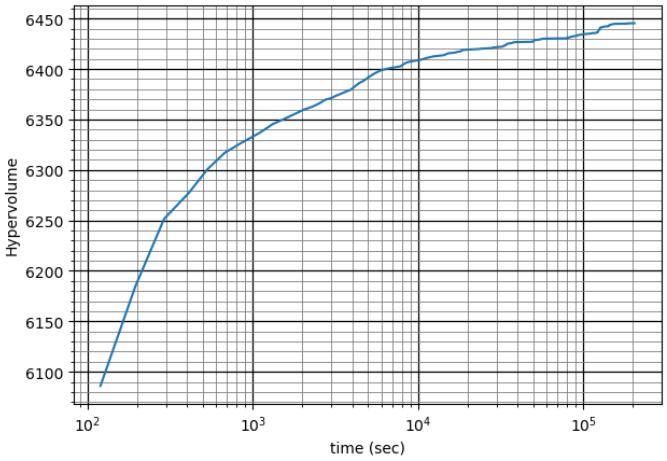


Figura 2.2: Gráfico BiteOpt

Basándonos en esta idea, se abordará la resolución del problema *Quantum Communications Constellations* como un problema mono-objetivo. La función a optimizar será la suma ponderada de los dos objetivos del problema original. Este enfoque permitirá aplicar varios algoritmos de optimización para evaluar su rendimiento y determinar cuál de ellos proporciona los mejores resultados.

2.2. Librería Pymoo

Pymoo se trata de una librería de optimización muy popular. Esta realiza la paralelización mediante la función objetivo proporcionada, aunque su sobrecarga es tan alta que en vez de acelerar la computación la hace más lenta. Para la resolución del problema se podría iniciar pymoo junto con NSGA-II en paralelo de manera manual y posteriormente unir los resultados aunque no superaría la puntuación ya obtenida. Este algoritmo se utilizó por tres motivos: para demostrar que se pueden implementar algoritmos de otras librerías fácilmente si

ajustan adecuadamente, que la sobrecarga del algoritmo es menos relevante para funciones de aptitud costosas y que una mala escalabilidad para la paralelización no importa si implementas tu propio mecanismo de paralelización.

A partir de los resultados obtenidos con pymoo se podría construir el Frente de Pareto. Aunque como se quiere precisar más el hipervolumen se construye el Frente de Pareto a partir de las siguientes ideas:

- Mejorar el Frente de Pareto de forma iterativa, en cada iteración se mejora el frente existente realizando muchas ejecuciones de optimización en paralelo.
- El nuevo Frente de Pareto se genera mediante la unión del frente existente junto con todas las soluciones de todas las ejecuciones.
- Las soluciones se recopilan dentro de la función de aptitud independientemente de la población real mantenida por el optimizador de **pymoo**.
- Las soluciones se mantienen para todos los optimizadores paralelos localmente en sus hilos para minimizar la comunicación entre procesos.
- Solo se almacena el Frente de Pareto para reducir el número de soluciones mantenidas.
- Sólo si el hipervolumen mejora, el Frente de Pareto se comparte globalmente a través de un diccionario de multiprocesamiento gestionado.

Después de cada iteración, se recopilan los Frentes de Paretos para cada optimización y se unen con el resultado anterior. Esto lleva a que el Frente de Pareto resultante puede crecer rápidamente a más de 2000 soluciones. Antes de la siguiente iteración, se necesita reducir su tamaño sin afectar su hipervolumen.

Esto se puede solucionar utilizando fcmaes [7]. Fcmaes gestiona la paralelización y proporciona un algoritmo de optimización adecuado llamado ***BiteOpt***.

BiteOpt es difícil de superar en la aplicación de este problema, ya que realiza múltiples reinicios que pueden ejecutarse en paralelo. Se puede ver su efectividad en la gráfica central de la Figura 2.1.

El rendimiento de la optimización del hipervolumen basada en pymoo se muestra en el gráfico de la Figura 2.3, en ella podemos ver que habría sido suficiente lograr una puntuación competitiva, por encima de -6400, si se ejecutara durante la noche, incluso si comienza con una suposición vacía. Hay que tener en cuenta que se aplicó “AdaptiveEpsilonConstraintHandling” para mejorar el manejo de restricciones.

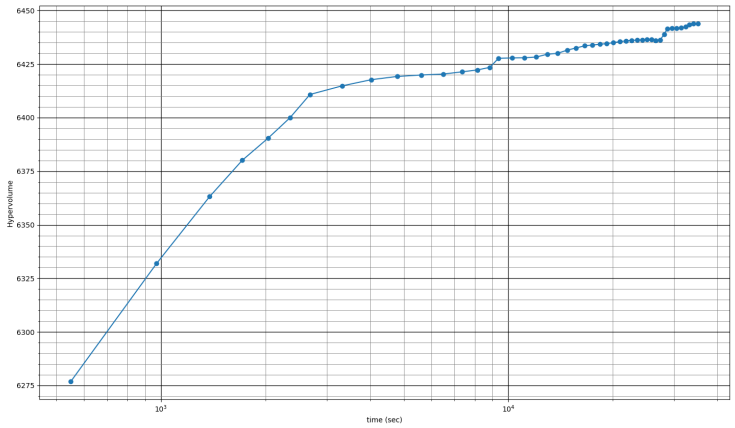


Figura 2.3: Gráfico Pymoo

3

Descripción formal del problema

En este capítulo se va a definir de manera formal el problema *Quantum Communications Constellations* obtenido por parte de la ESA [8]. Para la resolución del mismo, es fundamental comprender el enunciado del problema, el formato de los datos, cómo está definido el vector de decisión, cuándo son satisfechas las restricciones y qué objetivos se deben optimizar.

3.1. Enunciado del problema

El enunciado del problema *Quantum Communications Constellations* es el siguiente:

Después de un largo y arriesgado viaje, las naves nodrizas finalmente alcanzan Nuevo Marte. Al iniciar el esfuerzo de establecimiento de colonias en el planeta, surge un desafío crucial: establecer canales de comunicación confiables entre la superficie, donde los rovers serán desplegados, y las naves nodrizas en órbita. Para abordar este desafío, equipos de ingenieros han estado trabajando incansablemente en el diseño y despliegue de redes de comunicaciones cuánticas en el espacio.

Estas constelaciones de satélites cuánticos, basadas en comunicación láser de línea de visión, ofrecen una infraestructura segura para respaldar las operaciones de los rovers. Sin embargo, la eficiencia de la comunicación cuántica varía con la distancia entre los satélites. Además, el despliegue de esta tecnología avanzada conlleva costos operativos significativos. Los ingenieros se enfrentan a un dilema: ¿maximizar la eficiencia de la comunicación a costa de recursos operativos elevados, o minimizar el número de satélites y arriesgar la fiabilidad de la red? Para resolver esta cuestión, recurren a la optimización del diseño de las constelaciones.

Tras intensas deliberaciones, los ingenieros acuerdan que dos constelaciones del patrón Walker-Delta proporcionarán soporte adecuado para los próximos 10 años de operaciones planificadas de los rovers. Sin embargo, todavía necesitan

ayuda para encontrar configuraciones orbitales óptimas y libres de colisiones que permitan a cada constelación transmitir las señales de manera confiable entre las naves nodrizas y la superficie. Se deben determinar detalles como el número de planos por constelación, la cantidad de satélites cuánticos por plano, y sus características orbitales específicas.

Afortunadamente, los ingenieros tienen cierta flexibilidad para elegir las ubicaciones iniciales de los rovers, que llevarán a cabo las operaciones en la superficie del planeta. De un total de 100 ubicaciones posibles, deben identificar 4 puntos de inicio no coincidentes que mantengan una recepción estable de las transmisiones de las naves nodrizas durante una década.

A pesar de que el despliegue de la comunicación confiable en órbita es un paso crucial en la misión de Nuevo Marte, los ingenieros confían en que la comunidad encontrará la mejor manera de utilizar los recursos limitados para llevar este proyecto a buen término.

3.2. Datos

La competición proporciona una base de datos con cien posibles localizaciones de los rovers con el formato que se puede observar en la Tabla 3.1.

Tabla 3.1: Formato del archivo proporcionado con los datos de las posibles localizaciones de los rovers

Latitud(rad.)	**Longitud(rad.)**
3.290891754147420301e-01	3.455813102533720649e+00
9.210033238890864560e-01	1.222549040618014837e+00
.....	.....

La selección de 4 posiciones candidatas para el despliegue del rover se realiza codificando en el vector de decisión la línea correspondiente $r_i \in [0, 99]$ como se indica a continuación en la Sección 3.3.

3.3. Codificación de una solución

En esta sección vamos a definir el vector de decisión x, también llamado *cromosoma*[1], el cual se divide de la siguiente manera:

[1] El código para evaluar las posibles soluciones al problema es proporcionado por la ESA. Se puede acceder a él a través de la siguiente URL.

- Parámetros orbitales de la primera constelación de Walker: [a_1, e_1, i_1, w_1, eta_1]. Donde:
  - $a_1 \in [1.06, 1.8]$ es de tipo *float* y representa el semieje mayor en kilómetros.
  - $e_1 \in [0, 0.02]$ es de tipo *float* y representa la excentricidad.
  - $i_1 \in [0, \pi]$ es de tipo *float* y representa la inclinación en radianes.
  - $w_1 \in [0, 2\pi]$ es de tipo *float* y representa el perigeo en radianes.
  - $eta_1 \in [1, 1000]$ es de tipo *float* se define como el indicador de calidad común η de todos los satélites de la primera constelación Walker.
- Parámetros orbitales de la segunda constelación de Walker: [a_2, e_2, i_2, w_2, eta_2]. Donde:
  - $a_2 \in [2, 3.5]$ es de tipo *float* y representa el semieje mayor normalizado en kilómetros.
  - $e_2 \in [0, 0.02]$ es de tipo *float* y representa la excentricidad.
  - $i_2 \in [0, \pi]$ es de tipo *float* y representa la inclinación en radianes.
  - $w_2 \in [0, 2\pi]$ es de tipo *float* y representa el perigeo en radianes.
  - $eta_2 \in [1, 1000]$ es de tipo *float* se define como el indicador de calidad común η de todos los satélites de la segunda constelación Walker.
- Parametrización de la primera constelación de Walker: [S_1, P_1, F_1]. Donde:
  - $S_1 \in [4, 10]$ es de tipo *integer* y corresponde al número de satélites por plano.
  - $P_1 \in [2, 10]$ es de tipo *integer* y corresponde al número de planos.
  - $F_1 \in [0, 9]$ es de tipo *integer* y define el ajuste de fase de la constelación. Es decir, si $F_1 = N$ la fase de la constelación se repite cada N planos.
- Parametrización de la segunda constelación de Walker: [S_2, P_2, F_2]. Donde:
  - $S_2 \in [4, 10]$ es de tipo *integer* y corresponde al número de satélites por plano.
  - $P_2 \in [2, 10]$ es de tipo *integer* y corresponde al número de planos.
  - $F_2 \in [0, 9]$ es de tipo *integer* y define el ajuste de fase de la constelación. Es decir, si $F_2 = N$ la fase de la constelación se repite cada N planos.
- Posición de los rovers en la superficie: [r_1, r_2, r_3, r_4]. Donde:
  - $r_1 \in [0, 99]$ es de tipo *integer* y corresponde al primer rover.
  - $r_2 \in [0, 99]$ es de tipo *integer* y corresponde al segundo rover.
  - $r_3 \in [0, 99]$ es de tipo *integer* y corresponde al tercer rover.
  - $r_4 \in [0, 99]$ es de tipo *integer* y corresponde al cuarto rover.

Por lo tanto, tenemos que el vector de decisión x se define como:

$$x = [a_1, e_1, i_1, w_1, eta_1]+[a_2, e_2, i_2, w_2, eta_2]+[S_1, P_1, F_1]+[S_2, P_2, F_2]+[r_1, r_2, r_3, r_4]$$

3.4. Restricciones

Además, tenemos dos restricciones a tener en cuenta durante la resolución del problema:

1. **Cobertura de superficie:** la distancia mínima entre dos posiciones de rovers seleccionadas debe ser al menos *3000 km* para garantizar una buena cobertura de la superficie planetaria. Implementado como una restricción de desigualdad: se calcula como $rover\_constraint = udp.fitness(x)[3]$. Esta restricción es satisfecha cuando dicho valor es **negativo**.
2. **Prevención de colisiones:** dos satélites sólo pueden considerarse parte de la ruta de comunicación si existe una distancia de al menos *50 km* entre ellos; esto se aplica a las comunicaciones por satélite tanto entre constelaciones como dentro de ellas. Implementado como una restricción de desigualdad: se calcula como $collision\_constraint = udp.fitness(x)[4]$. Esta restricción es satisfecha cuando dicho valor es **negativo**.

3.5. Objetivos

Se trata de un problema multi-objetivo, ya que tiene dos objetivos. Los objetivos intentarán:

1. **Minimizar el costo de comunicación promedio más corto** entre las 7 naves nodrizas y los 4 rovers (a través de los satélites de la constelación Walker).
   Para que sea factible, una ruta de comunicación debe comenzar con una nave nodriza (nodo de origen) y terminar con un rover (nodo de destino), y puede pasar a través de N - 2 satélites cuánticos cercanos (nodos intermedios) que tienen línea de visión. Además, el ángulo cenital θ_z del enlace de comunicaciones que llega al rover debe satisfacer $\theta_z \leq \pi/3$. Entonces, el costo de las comunicaciones se puede expresar como la suma de $N-1$ enlaces:

$$QC = \sum_{i=1}^{N-1}(-\log(\eta_{i+1}) + 2\log(L_i)) + \frac{1}{\sin(\pi/2 - \theta_z)}$$

   Donde:
   - η_i es el indicador de calidad del i-ésimo satélite en la cadena. Será numéricamente igual a *eta1* o *eta2* según la constelación de Walker a la que pertenece el satélite.
   - L_i es la distancia en kilómetros entre el satélite i y el satélite $i+1$ de la cadena.
   - θ_z es el ángulo cenital del enlace de comunicaciones para el enlace final satélite-rover.

Denotamos por QC^* el valor mínimo (en todos los caminos factibles). El primer objetivo se trata del promedio de la ruta más corta entre cualquier par de nave nodriza - rover durante un período de diez años.

$$J_1 = \bar{QC}^*$$

Dicho valor se calcula como: $J_1 = udp.fitness(x)[0]$.

2. **Minimizar el coste total de fabricación, lanzamiento y operación de ambas constelaciones.** Dicho coste se define en función del número de satélites puestos en órbita así como de su medida de calidad:

$$J_2 = \eta_1 S_1 P_1 + \eta_2 S_2 P_2$$

Dicho valor se calcula como: $J_2 = udp.fitness(x)[1]$.

Se puede observar como estos dos objetivos están en conflicto entre sí ya que un buen camino de comunicación más corto (J_1) requiere muchos satélites y una gran *eta*. Al contrario que sucede para J_2 ya que esto haría mayor el coste total de fabricación, lanzamiento y operación de ambas constelaciones.

Como estamos ante un problema multi-objetivo, como ya se ha comentado, a lo largo de este trabajo se intentará encontrar una solución tratando al problema como un problema mono-objetivo. Se utilizará como función $f(x)$ a optimizar una suma ponderada entre los dos objetivos a minimizar, que denominaremos como función de fitness, trabajando con diferentes pesos w para así ver como varía dicho valor mínimo ponderado.

$$f(x) = J_1 \cdot w + J_2 \cdot (1 - w), \quad \text{donde} \quad w \in [0, 1]$$

Estamos interesados en las mejores compensaciones posibles entre esos dos objetivos que también satisfagan las restricciones antes mencionadas en la Sección 3.4. La calidad de la aproximación del Frente de Pareto será evaluada por el indicador de hipervolumen con un punto de referencia de [1.2, 1.4] en el espacio objetivo, que luego será multiplicado por 10000. Por lo tanto, sólo las soluciones con costos J_1 menor que 1.2 y J_2 menor que 1.4, serán las que contribuyan al hipervolumen. Además, J_1 y J_2 se encontrarán normalizados para que tengan una magnitud similar.

4

Técnicas Algorítmicas

A lo largo de este capítulo se explorarán tres algoritmos diferentes: una Búsqueda Aleatoria, un algoritmo de Búsqueda Local y un Algoritmo Memético. Cada uno de ellos ofrece maneras diferentes de abordar el problema entre manos. Se detallarán sus definiciones e implementaciones, proporcionando así un análisis exhaustivo de su aplicación en este contexto específico.

4.1. Algoritmo de búsqueda aleatoria

Una **Búsqueda Aleatoria (BA)** [10] consiste en generar un gran número de soluciones de manera aleatoria, dentro del conjunto de soluciones posibles, para un problema de optimización y elegir la mejor de ellas. Este método de búsqueda converge a la solución óptima, aunque el proceso para encontrar dicha solución puede ser lento. La BA se puede expresar como el pseudocódigo mostrado en el Algoritmo 1.

Algorithm 1 Algoritmo de Búsqueda Aleatoria

```
Definir f(x) (la función objetivo)
Definir el espacio de búsqueda
Establecer N (el número máximo de iteraciones)

MejorSolución ← None
MejorValor ← None
for i = 1 to N do
    x_i ← GenerarSoluciónAleatoria()
    Evaluar f(x_i)
    if f(x_i) es mejor que MejorValor then
        MejorSolución ← x_i
        MejorValor ← f(x_i)
    end if
end for

return MejorSolución y MejorValor
```

Mediante este algoritmo se puede dar el caso en el que obtengamos soluciones no factibles ya que aunque pertenecen al conjunto de soluciones puede que no cumplan las restricciones a nuestro problema.

En un primer lugar, se implementó el siguiente código con el que se define una función que genera de manera aleatoria un cromosoma que pertenece a nuestro conjunto de soluciones, es decir, los valores de las variables están definidas en los intervalos predeterminados por el enunciado del problema *Quantum Communications Constellations*:

```
import random
import math
def v_aleatorio():
  #Parametros orbitales de la primera constelacion de Walker
  a1 = random.uniform(1.06,1.8)
  e1 = random.uniform(0,0.02)
  i1 = random.uniform(0, math.pi)
  w1 = random.uniform(0, 2*math.pi)
  eta1 = random.uniform(1, 1000)
  #Parametros orbitales de la segunda constelacion de Walker
  a2 = random.uniform(2,3.5)
  e2 = random.uniform(0,0.01)
  i2 = random.uniform(0, math.pi)
  w2 = random.uniform(0, 2*math.pi)
  eta2 = random.uniform(1, 1000)
  #Parametrizacion de la primera constelacion de Walker
  S1 = random.randint(4,10)
  P1 = random.randint(2,10)
  F1 = random.randint(0,9)
  #Parametrizacion de la segunda constelacion de Walker
  S2 = random.randint(4,10)
  P2 = random.randint(2,10)
  F2 = random.randint(0,9)
  #Posicion de los rovers en la superficie
  r1 = random.randint(0,99)
  r2 = random.randint(0,99)
  r3 = random.randint(0,99)
  r4 = random.randint(0,99)
  #cromosoma aleatorio
  x = [a1, e1, i1, w1, eta1] + [a2, e2, i2, w2, eta2] + [S1, P1, F1] +
      [S2, P2, F2] + [r1, r2, r3, r4]
  return x
```

Una vez se tiene la función **v_aleatorio()** que define un cromosoma aleatorio dentro de los parámetros para cada variable se procede a hacer el código en el que se implementa la búsqueda aleatoria, pero previamente hay que definir dos funciones adicionales que facilitarán la implementación de la búsqueda aleatoria.

La función **deg_insatisf()** se implementa debido a que el problema *Quantum Communications Constellations* presenta dos restricciones: la cobertura de superficies y la prevención de colisiones. Ambas son satisfechas cuando al hacer la evaluación del cromosoma x a través de la invocación a la función *udp.fitness(x)* se tiene que *udp.fitness(x)*[3] y *udp.fitness(x)*[4] son valores negativos. El grado de insatisfacción varía en función de:

- Se cumplen ambas restricciones: se tiene que el grado de insatisfacción es 0 ya que se satisfacen ambas restricciones.
- Se cumple la primera restricción pero no la segunda: el valor de la segunda restricción se multiplica por un número suficientemente grande para diferenciarla de cuando no se cumple la primera restricción.
- Se cumple la segunda restricción pero no la primera: el valor de la primera restricción se multiplica por un número menor que el número por el que se multiplica la segunda restricción para diferenciarlos.
- No se cumple ninguna de las restricciones: cada valor se multiplica por los números utilizados anteriormente y se suman entre ellos.

Durante la proceso, buscamos soluciones factibles que cumplan ambas restricciones por lo que deben tener su grado de insatisfacción igual a cero. Al la función deg_insatisf() se le pasa como parámetro la evaluación del vector x del que se quiere saber si cumple o no las restricciones. El código implementado para esta función es el siguiente:

```
def deg_insatisf(evaluacion):
  if evaluacion[2] < 0 and evaluacion[3] < 0:
    deg_insatisf = 0
  elif evaluacion[2] < 0 and evaluacion[3] > 0:
    deg_insatisf = evaluacion[3] * 10**10
  elif evaluacion[2] > 0 and evaluacion[3] < 0:
    deg_insatisf = evaluacion[2]*10**6
  else:
    deg_insatisf = evaluacion[2]*10**6 + evaluacion[3]*10**10
  return deg_insatisf
```

La función **fun_obj()** se implementa ya que nuestro problema *Quantum Communications Constellations* se trata de un problema multi-objetivo. En concreto, tiene dos objetivos J_1 y J_2, por lo que nuestra función a minimizar será la suma ponderada de J_1 y J_2. Para ello, la función recibe como parámetros la evaluación del cromosoma x, al que se le quiere calcular la función, y el peso que se le dará a J_1. Su código viene dado por:

```
def fun_obj(evaluacion, weight):
  fun_obj = evaluacion[0]*weight + evaluacion[1]*(1-weight)
  return fun_obj
```

Con estas funciones implementadas se puede realizar la búsqueda aleatoria. Este algoritmo genera una primera solución aleatoria que almacena y compara con una nueva solución en cada iteración. Si esta nueva solución es mejor que la ya almacenada, se guarda esta como la mejor solución. Una nueva solución será mejor que la ya almacenada si su grado de insatisfacción es menor que el de la solución almacenada o en caso de que fueran iguales si el valor de la función objetivo ponderada es menor que el valor de la función objetivo ponderada de la solución ya almacenada. El código implementado para este algoritmo es el siguiente:

```
def random_search(num_iter, weight):
  current_sol = v_aleatorio()
  current_sol_eval = udp.fitness(current_sol)
  actual_iter = 0
  while actual_iter < num_iter:
    new_sol = v_aleatorio()
    new_sol_eval = udp.fitness(new_sol)
    if deg_insatisf(new_sol_eval) < deg_insatisf(current_sol_eval):
      current_sol = new_sol
      current_sol_eval = new_sol_eval
    elif deg_insatisf(new_sol_eval) == deg_insatisf(current_sol_eval):
      if fun_obj(new_sol_eval, weight) < fun_obj(current_sol_eval,
          weight):
        current_sol = new_sol
        current_sol_eval = new_sol_eval
    actual_iter = actual_iter + 1
  return [current_sol, current_sol_eval]
```

4.2. Algoritmo de búsqueda local

Un **Algoritmo de Búsqueda Local (LS)** [11] es una técnica iterativa de optimización que mejora una solución candidata mediante pequeños cambios en su estructura, evaluando los vecinos de la solución actual y aceptando mejoras hasta alcanzar un óptimo local. La LS se ajusta al pseudocódigo mostrado en el Algoritmo 2.

Si el criterio de parada no es elegido de manera adecuada se puede dar el caso en el que las soluciones obtenidas puedan ser no factibles si no cumplen las restricciones del problema.

Algorithm 2 Algoritmo de Búsqueda Local

```
1: SolucionActual ← GenerarSolucionInicial(Problema)
2: while ¬CriterioDeParada() do
3:     Vecinos ← GenerarVecinos(SolucionActual)
4:     MejorVecino ← SeleccionarMejorVecino(Vecinos)
5:     if MejorVecino mejora la solucion actual then
6:         SolucionActual ← MejorVecino
7:     end if
8:     ActualizarCriterioParada()
9: end while
10:
11: return SolucionActual
```

Para realizar el código del algoritmo de Búsqueda Local se utilizaron las funciones v_aleatorio(), deg_insatisf() y fun_obj() ya explicadas anteriormente en la Sección 4.1. Además de estas funciones, se realizaron dos nuevas funciones adicionales necesarias. Dichas funciones se explican a continuación.

La función **perturbar()** es la que se encarga de generar vecinos. A esta función se le pasa un vector a perturbar, elige de manera aleatoria la posición a la que se le va a realizar dicha perturbación y genera un nuevo parámetro para esa posición dentro de los valores permitidos, teniendo en cuenta si debe ser un número entero o decimal. Una vez realizado esto, devuelve el nuevo vector. Su código es el siguiente:

```
1 def perturbar(x):
2   rangos = [(1.06,1.8), (0,0.02), (0,math.pi), (0,2*math.pi),(1,1000),
        (2,3.5), (0,0.01), (0,math.pi), (0,2*math.pi), (1,1000), (4,10),
        (2,10), (0,9), (4,10), (2,10), (0,9),(0,99), (0,99), (0,99),
        (0,99)]
3   posicion = random.randint(0,19)
4   x_per = x.copy()
5   if posicion in [0,1,2,3,4,5,6,7,8,9]:
6     x_per[posicion] = random.uniform(rangos[posicion][0], rangos[
          posicion][1])
7   else:
8     x_per[posicion] = random.randint(rangos[posicion][0], rangos[
          posicion][1])
9   return x_per
```

La función **mejor()** se encarga de comparar dos evaluaciones. A esta función se le pasan dos vectores evaluados y un peso. Primero, compara el grado de insatisfacción de ambos vectores. Si el grado de insatisfacción del primer vector es menor, devuelve la evaluación de dicho vector. Si el grado de insatisfacción del segundo vector es menor, devuelve la evaluación de este. Si ambos tienen el mismo grado de insatisfacción, compara los valores de la función objetivo

ponderada y devuelve la evaluación del vector con el valor menor. En cualquier caso, devuelve la evaluación del vector que se considera mejor según los criterios especificados anteriormente. El código de esta función es el siguiente:

```
def mejor(eval_x,eval_y,weight):
  if deg_insatisf(eval_x) < deg_insatisf(eval_y):
    mejor = eval_x
    return mejor
  elif deg_insatisf(eval_x) > deg_insatisf(eval_y):
    mejor = eval_y
    return mejor
  else:
    if fun_obj(eval_x, weight) < fun_obj(eval_y, weight):
      mejor = eval_x
      return mejor
    else:
      mejor = eval_y
      return mejor
  return mejor
```

Una vez implementadas todas las funciones auxiliares se procedió a realizar el código para la Búsqueda Local. La función **busq_local()** implementa una Búsqueda Local que tiene por parámetros el peso y el número límite de iteraciones sin mejoras. Primero, genera una solución aleatoria inicial y evalúa su aptitud. Luego, entra en un bucle que continúa mientras no se alcance el número máximo de iteraciones sin mejoras. En cada iteración, genera un nuevo vecino perturbando la solución actual y evalúa su aptitud. Si el nuevo vecino es más próximo a óptimo según la función **mejor**, actualiza la solución actual y reinicia el contador de mejoras sin éxito. Si no, incrementa el contador de mejoras sin éxito. Finalmente, cuando llega al número máximo permitido de iteraciones sin mejoras, devuelve la mejor solución encontrada y su evaluación. A continuación, se muestra su código.

```
def busq_local(weight, num_mejoras):
  current_sol = v_aleatorio()
  current_sol_eval = udp.fitness(current_sol)
  no_mejoras = 0
  while no_mejoras < num_mejoras:
    new_neighbor = perturbar(current_sol)
    new_neighbor_eval = udp.fitness(new_neighbor)
    if mejor(current_sol_eval,new_neighbor_eval,weight) ==
        new_neighbor_eval:
      current_sol = new_neighbor
      current_sol_eval = new_neighbor_eval
      no_mejoras = 0
    else:
      no_mejoras += 1
  return [current_sol, current_sol_eval]
```

4.3. Algoritmo memético

Un **Algoritmo Memético (AM)** [12] es una técnica de optimización que combina un algoritmo evolutivo y un procedimiento de mejora de soluciones como, por ejemplo, una búsqueda local. Comienza con un conjunto de soluciones de alta calidad o aleatorias, mejora estas soluciones mediante la BL, y usa procedimientos de recombinación para crear nuevas soluciones. Los AMs incluyen mecanismos de inicialización, criterios de terminación, cooperación, mejora, competencia y reinicio para mantener la diversidad y evitar el estancamiento en la búsqueda de soluciones óptimas. El MA se expresa con el pseudocódigo mostrado en el Algoritmo 3.

Algorithm 3 Algoritmo Memético

1: **Función AlgoritmoMemético** (P, par)
2: pop $\leftarrow$ Inicializar(par, P);
3: **repeat**
4: newpop1 $\leftarrow$ Combinar(pop, par, P);
5: newpop2 $\leftarrow$ Mejorar(newpop1, par, P);
6: pop $\leftarrow$ Competir(pop, newpop2);
7: **until** CriterioTerminación(par);
8:
9: **return** **ObtenerNthMejor**(pop, 1);

Donde las funciones **Inicializar()**, **Combinar()** , **Mejorar()**, **Competir()** y **Reiniciar()** se pueden explicar como:

- **Inicializar:** se genera una población de manera aleatoria que se mejora mediante la búsqueda local.

- **Combinar:** mediante la recombinación de las soluciones ya existentes se crea una nueva población.
- **Mejorar:** se le aplica a la nueva población un algoritmo de búsqueda local.
- **Competir:** evalúa y selecciona las mejores soluciones de entre la población actual y las nuevas generadas. Determina qué soluciones deben mantenerse para la siguiente iteración.

En el código del Algoritmo Memético se utilizaron las funciones v_aleatorio(), deg_insatisf() y fun_obj() ya explicadas anteriormente en la Sección 4.1. Además de estas funciones, se realizó una adaptación de la Búsqueda Local descrita en la Sección 4.2 para aplicarla en el MA. Por último, se implementaron nuevas funciones para hacer más sencilla la implementación.

Se utilizaron las mismas funciones auxiliares para la Búsqueda Local. La modificación se hizo directamente sobre la función busq_local() de la manera siguiente:

```
def busq_local(weight,x, x_eval, num_mejoras):
  no_mejoras = 0
  while no_mejoras < num_mejoras:
    new_neighbor = perturbar(x)
    new_neighbor_eval = udp.fitness(new_neighbor)
    if mejor(x_eval,new_neighbor_eval,weight) == new_neighbor_eval:
      x = new_neighbor
      x_eval = new_neighbor_eval
      no_mejoras = 0
    else:
      no_mejoras += 1
  return [x, x_eval]
```

Ahora a la búsqueda local, además del peso y el número de iteraciones sin mejoras, se le pasa el cromosoma a mejorar y su evaluación. De esta manera, su aplicación en el Algoritmo Memético es directa ya que devuelve el propio vector ya mejorado y la evaluación del mismo.

A continuación, se explicarán las funciones auxiliares para el Algoritmo Memético. Para inicializar el algoritmo, se necesita una población inicial de individuos. Esto se realiza mediante la función **padres()**, la cual recibe como parámetro el número de individuos N deseado para la población. La función generará N vectores aleatorios que conformarán nuestra población inicial. Estos vectores serán devueltos como resultado. El código de esta función es el siguiente:

```
def padres(num_ind):
  pob_padre = []
  for i in range(0,num_ind):
    x = v_aleatorio()
    pob_padre.append(x)
  return pob_padre
```

La función **torneo_binario()** selecciona un individuo de una población utilizando el método de torneo binario. Recibe tres parámetros: "poblacion" (en nuestro caso, la población padre), "poblacion_eval" (evaluaciones de los individuos) y "weight" (peso utilizado para evaluar). Esta función se utiliza para seleccionar los padres que se van a utilizar para generar la población hija. Primero, selecciona dos individuos al azar de la población y obtiene sus evaluaciones correspondientes. Luego, compara estas evaluaciones utilizando la función **mejor**, que determina cuál de los dos individuos es superior según el peso dado. Finalmente, retorna el individuo con la mejor evaluación, asegurando así que el mejor evaluado tiene una mayor probabilidad de ser seleccionado. Su código tiene la siguiente forma:

```
def torneo_binario(poblacion, poblacion_eval, weight):
  num1 = random.randint(1,len(poblacion))
  num2 = random.randint(1,len(poblacion))
  indv1 = poblacion[num1-1]
  indv1_eval = poblacion_eval[num1-1]
  indv2 = poblacion[num2-1]
  indv2_eval = poblacion_eval[num2-1]
  if mejor(indv1_eval, indv2_eval, weight) == indv1_eval:
    return indv1
  else:
    return indv2
```

La función **one_point_cross()** se utiliza para generar descendencia (offspring) mediante un proceso de recombinación. Recibe tres parámetros: parent1 y parent2, que son los padres a cruzar, y pc, que es la probabilidad de cruce. Primero, se genera un número aleatorio (num_aleat) entre 0 y 1. Si la probabilidad de cruce es mayor o igual a num_aleat, se selecciona un punto de cruce aleatorio punto_cruce entre 0 y 19. Luego, se crean dos descendientes intercambiando segmentos de los padres en el punto de cruce: offspring1 toma la parte inicial de parent1 hasta punto_cruce y el resto de parent2, mientras que offspring2 toma la parte inicial de parent2 hasta punto_cruce y el resto de parent1. Si pc es menor que num_aleat, los descendientes son copias exactas de los padres. Finalmente, se retorna una lista con los dos descendientes. Podemos ver su implementación

a continuación:

```
def one_point_cross(parent1, parent2, pc):
  num_aleat = random.uniform(0,1)
  if pc >= num_aleat:
    punto_cruce = random.randint(0,19)
    offspring1 = parent1[:punto_cruce] + parent2[punto_cruce:]
    offspring2 = parent2[:punto_cruce] + parent1[punto_cruce:]
  else:
    offspring1 = parent1.copy()
    offspring2 = parent2.copy()
  return [offspring1, offspring2]
```

La función **seleccion_superv()** selecciona una cantidad específica de individuos (num_ind) de dos poblaciones combinadas, basándose en su evaluación y criterios de insatisfacción. Recibe seis parámetros: poblacion1 y poblacion2 (dos listas de individuos, en nuestro caso serán la población padre y la población hija), poblacion1_eval y poblacion2_eval (evaluaciones correspondientes de cada individuo en las poblaciones padre e hija), num_ind (número de individuos a seleccionar) y weight (peso utilizado en la función de evaluación).
Primero, se combinan ambas poblaciones y sus evaluaciones en listas únicas (pob y pob_eval). Luego, se calculan dos métricas para cada individuo: el grado de insatisfacción (deg_insatisf()) y el valor de la función objetivo (fun_obj()), almacenándolos en listas separadas (pob_deg_insat y pob_fun_obj). Se crea una nueva lista de tuplas (poblacion), donde cada tupla contiene un individuo, su evaluación, su grado de insatisfacción y su valor de función objetivo.
Esta lista se ordena primero por el grado de insatisfacción y luego por el valor de la función objetivo. Los primeros num_ind individuos de esta lista ordenada se seleccionan como supervivientes. Finalmente, la función retorna dos listas: una con los individuos seleccionados y otra con sus evaluaciones correspondientes. Su código es el que sigue:

```
def seleccion_superv(poblacion1, poblacion2, poblacion1_eval,
    poblacion2_eval, num_ind, weight):
  pob = poblacion1 + poblacion2
  pob_eval = poblacion1_eval + poblacion2_eval
  pob_deg_insat = []
  pob_fun_obj = []
  for i in range(0,len(pob_eval)):
    pob_deg_insat.append(deg_insatisf(pob_eval[i]))
    pob_fun_obj.append(fun_obj(pob_eval[i], weight))
  poblacion = []
  for i in range(0,len(pob_eval)):
    poblacion.append((pob[i], pob_eval[i], pob_deg_insat[i],
        pob_fun_obj[i]))
  poblacion_ordenada = sorted(poblacion, key=lambda x: (x[2], x[3]))
  supervivientes = [individuo[0] for individuo in poblacion_ordenada[:
      num_ind]]
  supervivientes_eval = [individuo[1] for individuo in
      poblacion_ordenada[:num_ind]]
  return [supervivientes, supervivientes_eval]
```

Con las funciones auxiliares ya implementadas y explicadas, se procede a describir la implementación del MA:

1. **Inicialización:**
   - Se crea una población inicial de padres (pob_padre) mediante la función **padres()** con num_ind individuos.
   - Se evalúan estos padres utilizando la función de aptitud **udp.fitness()** y se almacena en pob_padre_eval.
   - Cada padre es mejorado localmente mediante la función **busq_local()**, actualizando la población y sus evaluaciones.
2. **Ciclo de generaciones:**
   - Para cada generación (hasta num_gen), se genera una nueva población hija:
     - Se seleccionan pares de padres mediante el método de torneo binario (**torneo_binario()**).
     - Se cruzan los padres utilizando el cruce de un punto (**one_point_cross()**) con probabilidad pc.
     - Se evalúan los hijos generados (**udp.fitness()**).
     - Cada hijo es mejorado localmente mediante **busq_local()**.
   - Se combinan las poblaciones de padres e hijos, y se seleccionan los mejores num_ind individuos para la siguiente generación utilizando la función **seleccion_superv()**.
   - La mejor solución de la generación actual se guarda en evolucion.
   - Se incrementa el contador de generaciones.

3. **Retorno:**
   - Al finalizar todas las generaciones, se retorna la mejor solución encontrada, su evaluación y la evolución completa de soluciones.

   El código del algoritmo memético viene dado por:

```
def alg_memetico(num_ind, num_gen, weight, pc, num_mejoras):
  pob_padre = padres(num_ind)
  pob_padre_eval = []
  generacion = 0
  evolucion = []
  for i in range(0,len(pob_padre)):
    pob_padre_eval.append(udp.fitness(pob_padre[i]))
  for i in range(0,len(pob_padre)):
    x = pob_padre[i]
    x_eval = pob_padre_eval[i]
    padre_mejor = busq_local(weight, x, x_eval, num_mejoras)
    pob_padre[i] = padre_mejor[0]
    pob_padre_eval[i] = padre_mejor[1]
  while generacion < num_gen:
    pob_hija = []
    for i in range(0, int(num_ind/2)):
      parent1 = torneo_binario(pob_padre, pob_padre_eval, weight)
      parent2 = torneo_binario(pob_padre, pob_padre_eval, weight)
      hijos = one_point_cross(parent1, parent2, prob_cruce)
      pob_hija = pob_hija + hijos
    pob_hija_eval = []
    for i in range(0,len(pob_hija)):
      pob_hija_eval.append(udp.fitness(pob_hija[i]))
    for i in range(0,len(pob_hija)):
      x = pob_hija[i]
      x_eval = pob_hija_eval[i]
      hijo_mejor = busq_local(weight, x, x_eval, num_mejoras)
      pob_hija[i] = hijo_mejor[0]
      pob_hija_eval[i] = hijo_mejor[1]
    superv = seleccion_superv(pob_padre, pob_hija, pob_padre_eval,
        pob_hija_eval, num_ind, weight)
    pob_padre = superv[0]
    pob_padre_eval = superv[1]
    current_sol = pob_padre[0]
    current_sol_eval = pob_padre_eval[0]
    evolucion.append((current_sol, current_sol_eval, deg_insatisf(
        current_sol_eval), fun_obj(current_sol_eval, weight)))
    generacion +=1
  return [current_sol, current_sol_eval, evolucion]
```

5

Resultados y comparación de algoritmos

A lo largo de este trabajo se han explorado tres enfoques diferentes para abordar el problema *Quantum Communications Constellations*. Estos enfoques se han aplicado con las mismas condiciones, en la medida de lo posible, para que la comparación de los algoritmos sea lo más acertada posible. Recordemos que el problema a resolver es multi-objetivo, pero los algoritmos se han implementado de manera que se resuelve una suma ponderada de los objetivos (función de fitness), convirtiéndolo así en un problema mono-objetivo. Por lo tanto, se asigna un peso w al primer objetivo y $1 - w$ al segundo objetivo. Los 10 pesos elegidos se han tomado en el intervalo [0,1] tal y como se observa en la Tabla 5.1. Además, para cada uno de estos pesos se han realizado 10 repeticiones. En este capítulo, se mostrarán los resultados obtenidos para cada algoritmo. Además, se intentará encontrar el método más eficaz para la búsqueda de la solución óptima comparando los tres algoritmos.

5.1. Búsqueda aleatoria

Para la búsqueda de soluciones aleatorias se ejecutó el código fuente descrito en la Sección 4.1 con diferentes pesos y varias repeticiones para cada peso. Los parámetros utilizados para la ejecución de este algoritmo se pueden ver en la Tabla 5.1.

Tabla 5.1: Parámetros de la búsqueda aleatoria

Parámetro	**Variable**	**Valores**
Número de iteraciones	num_iter	100
Pesos	weight	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
Repeticiones	repeticiones	10

Al terminar las ejecuciones, se obtuvieron soluciones factibles en todos los casos, es decir, que cumplen las restricciones al problema. Los valores de

la función de fitness utilizada (suma ponderada de ambas funciones objetivo), obtenidos para los diferentes pesos y repeticiones, se pueden observar en los diagramas de cajas de la Figura 5.1. Cabe destacar que, a medida que aumenta el peso dado al objetivo 1, también aumenta el valor de la función de fitness. Esto significa que el objetivo 1 contribuye en mayor medida a la función de fitness.

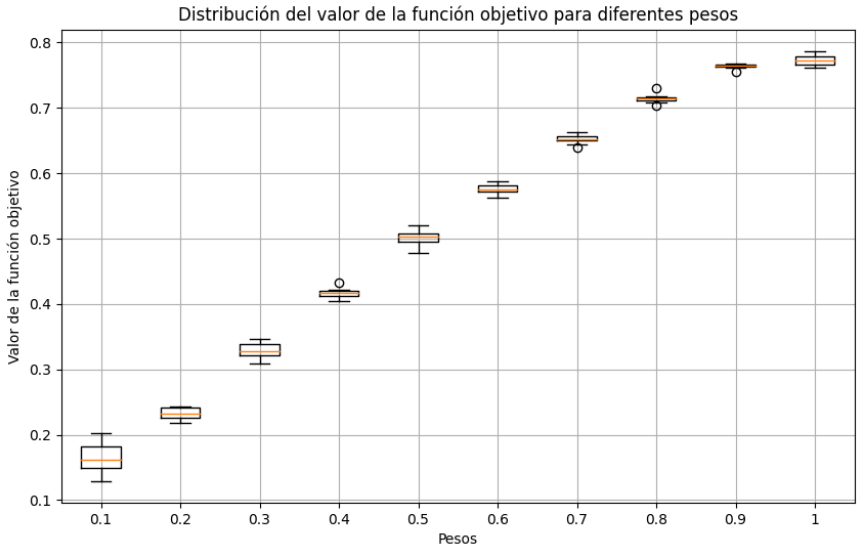


Figura 5.1: Búsqueda aleatoria - Valores de la función de fitness para diferentes pesos y repeticiones

Después de aproximadamente seis horas de ejecución con un procesador Intel Core i3-6100U CPU 2.30GHz, se consiguieron cien soluciones al problema utilizando la búsqueda aleatoria, debido a que la función de evaluación *udp.fitness()* proporcionada por la ESA es muy costosa desde el punto de vista de su cómputo.

Este conjunto con cien soluciones se subió a través de la plataforma proporcionada por la ESA para probar el sistema de envío. La plataforma calcula el hipervolumen de una aproximación a un Frente de Pareto, tomando como referencia el punto [1.2, 1.4], tal y como se comentó con anterioridad, y multiplicándolo por 10000. Además, el valor resultante se invierte, por lo que, cuanto menor sea el valor de hipervolumen, mejor será la aproximación al Frente de Pareto. En el caso que nos ocupa, se obtuvo un valor de hipervolumen igual a -5606.734.

Para cada una de las 10 repeticiones realizadas, se obtuvo una aproximación al Frente de Pareto combinando las soluciones obtenidas para cada uno de los pesos. El cálculo de los hipervolúmenes para las diez aproximaciones al Frente de Pareto también se realizó mediante la plataforma de la ESA. Los valores

obtenidos se pueden observar en la Tabla 5.2 y su distribución, a través del diagrama de cajas de la Figura 5.2.

Tabla 5.2: Búsqueda aleatoria - Valores de hipervolumen obtenidos en cada repetición

Repetición	**Hipervolumen**
1	-5450.584
2	-5030.761
3	-5161.933
4	-5239.993
5	-5391.349
6	-5308.355
7	-5354.867
8	-5219.399
9	-5262.798
10	-5345.752

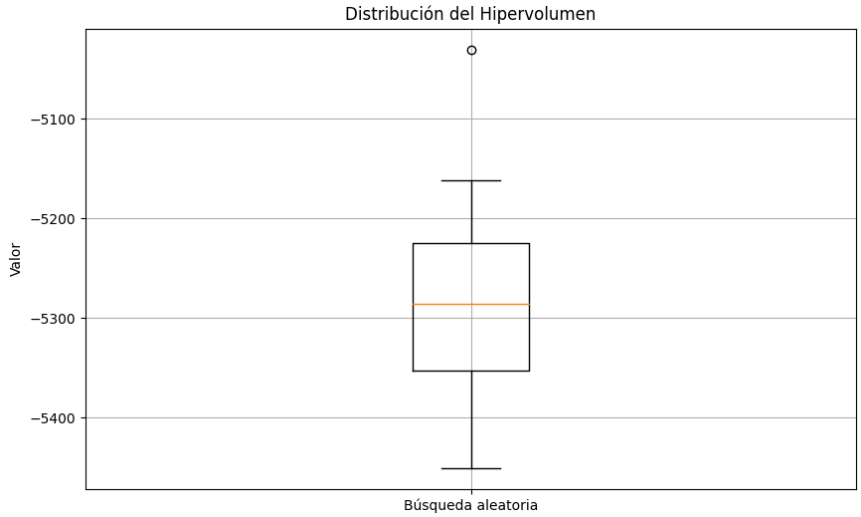


Figura 5.2: Búsqueda aleatoria – Distribución de los valores de hipervolumen

Al mismo tiempo, la Tabla 5.3 muestra varios estadísticos descriptivos del hipervolumen obtenido en las diez repeticiones. Dicha tabla proporciona una visión general del comportamiento de la búsqueda aleatoria en términos del hipervolumen. En ella, se puede observar como la mejor de las aproximaciones al Frente de Pareto tiene un hipervolumen igual a -5450.584, correspondiendo a la primera repetición.

Tabla 5.3: Búsqueda aleatoria - Estadísticos del hipervolumen

Estadístico	Valor
Mínimo	-5450.584
Máximo	-5030.761
Media	-5276.5791
Mediana	-5285.5765

La mejor aproximación al Frente de Pareto (la correspondiente a la primera repetición, tal y como se ha indicado anteriormente), se puede observar en la Figura 5.3. Se tiene que los extremos corresponden a las soluciones obtenidas con peso 0.2 (extremo superior-izquierdo) y con peso 1 (extremo inferior-derecho).

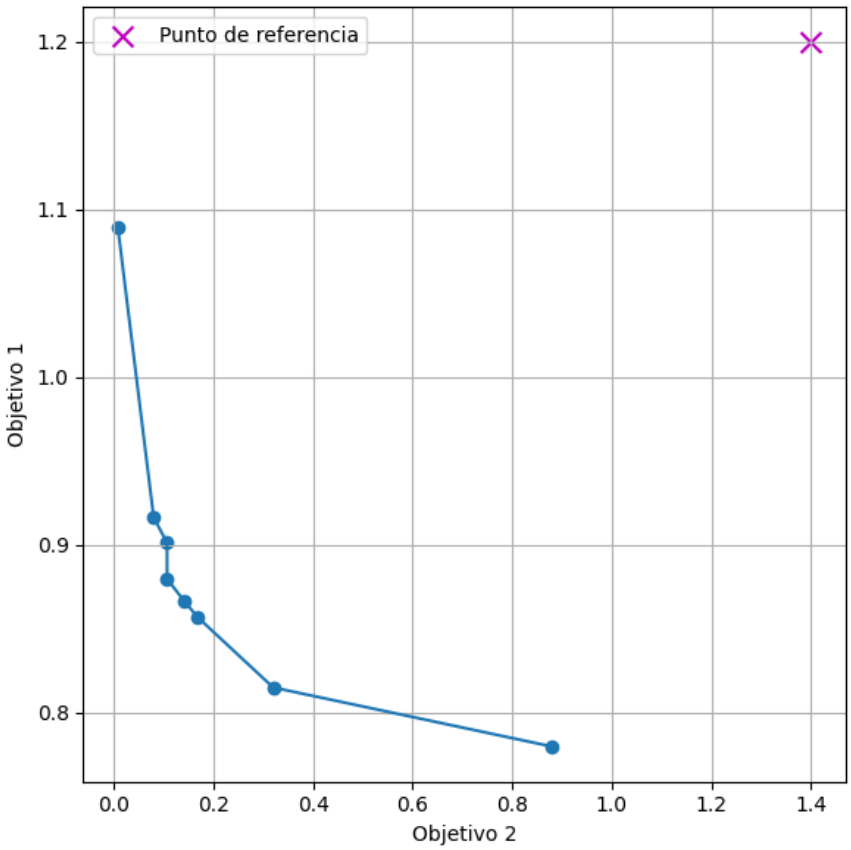


Figura 5.3: Búsqueda aleatoria - Mejor aproximación al Frente de Pareto

A partir de la función *udp.pretty()*, a la cual se le pasa como parámetro un cromosoma x, y que se encuentra definida en el código de evaluación proporcionado por la ESA, se ha obtenido la información con ambos extremos de la mejor

aproximación del Frente de Pareto. Como salida se ha obtenido lo siguiente:

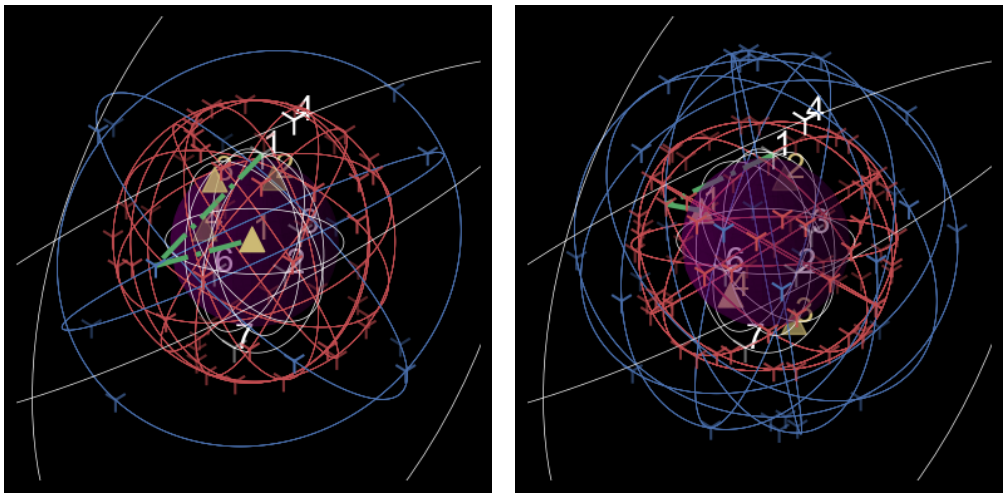
```
FITNESS EVALUATION: Extremo superior-izquierdo
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 45, W2: 12): 57
OBJECTIVE 1 - Average communications cost: 1.0890663873012036
OBJECTIVE 2 - Cost of infrastructure: 0.008169117729345528
CONSTRAINT - Minimum distance between rovers (OK): 5638.003011420617 km
CONSTRAINT - Minimum distance between sats (OK): 1077.4580730777343 km
------------------------------------------------------------------------

(1.0890663873012036, 0.008169117729345528, -2638.0030114206174,
    -1027.4580730777343)
```

```
FITNESS EVALUATION: Extremo inferior-derecho
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 70, W2: 28): 98
OBJECTIVE 1 - Average communications cost: 0.7796196083374967
OBJECTIVE 2 - Cost of infrastructure: 0.8784473346894205
CONSTRAINT - Minimum distance between rovers (OK): 6105.529888386955 km
CONSTRAINT - Minimum distance between sats (OK): 965.8617553313253 km
------------------------------------------------------------------------

(0.7796196083374967, 0.8784473346894205, -3105.5298883869546,
    -915.8617553313253)
```

Se observa que el extremo superior-izquierdo, con una menor cantidad de satélites (57 satélites), presenta un mayor coste medio de comunicaciones ($J_1 = 1.0891$) en comparación con el extremo inferior-derecho, que tiene una mayor cantidad de satélites (98 satélites) y un coste medio de comunicaciones más bajo ($J_1 = 0.7796$). Sin embargo, esta última solución, al contar con más satélites, resulta en un coste de infraestructuras considerablemente mayor ($J_2 = 0.87844$ frente a $J_2 = 0.0082$).

Además, a partir de la función *udp.plot()*, también proporcionada en el código de evaluación de la ESA, se han dibujado las órbitas correspondientes a las soluciones extremas de la mejor aproximación del Frente de Pareto. En azul se representa la constelación Walker 1, en rojo se representa la constelación Walker 2, en blanco se representan las naves nodrizas y en amarillo se representan los rovers. En la Figura 5.4(a) se pueden observar las órbitas de la solución correspondiente al extremo superior-izquierdo, mientras que en la Figura 5.4(b), se observan las órbitas de la solución correspondiente al extremo inferior-derecho.

Además, en verde, se representa el camino entre una nave nodriza y un rover, en estos casos entre la nave nodriza 1 y el rover 1.



(a) Mothership 1 (node 57) communicates with rover 1 (node 64) at epoch 1 via: (57, 45, 64)

(b) Mothership 1 (node 98) communicates with rover 1 (node 105) at epoch 1 via: (98, 64, 105)

Figura 5.4: Búsqueda aleatoria - Representación de las órbitas correspondientes a los puntos extremos de la mejor aproximación al Frente de Pareto

5.2. Búsqueda local

De un modo similar que para la búsqueda aleatoria, considerando la búsqueda local se ejecutó el código fuente descrito en la Sección 4.2. Los parámetros utilizados para la ejecución de este algoritmo se pueden ver en la Tabla 5.4.

Tabla 5.4: Parámetros de la búsqueda local

Parámetro	**Variable**	**Valores**
Número sin mejoras	num_mejoras	30
Pesos	weight	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
Repeticiones	repeticiones	10

En este caso, se consideró que el número de no mejoras fuera igual a 30, es decir, para que la búsqueda local finalizara debía suceder que la solución actual no mejorase durante 30 iteraciones del algoritmo. Se consiguieron soluciones factibles en todas las repeticiones, es decir, con grado de insatisfacción igual a cero.

Además, los valores de la función de fitness (suma ponderada de ambos objetivos) obtenidos para los diferentes pesos y repeticiones se pueden observar en el diagrama de cajas de la Figura 5.5. Cabe destacar de nuevo que, a medida que aumenta el peso dado al objetivo 1, también aumenta el valor de la función de fitness. Esto significa que el objetivo 1 contribuye en mayor medida a dicha función ponderada.

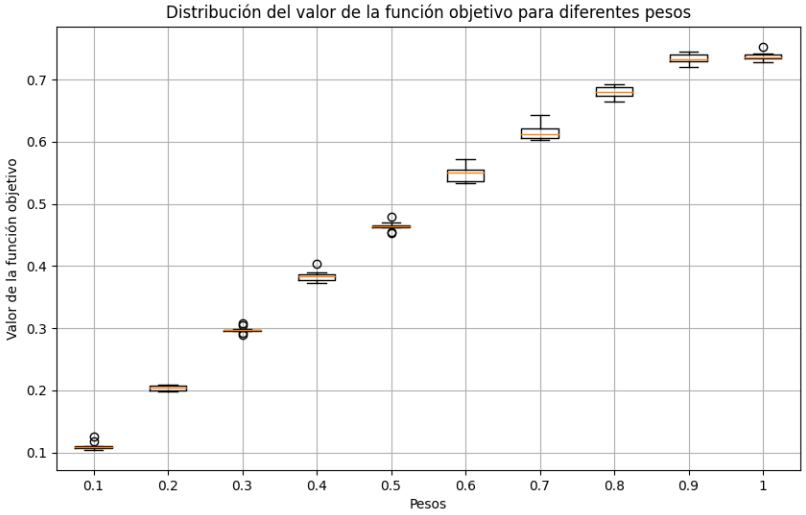


Figura 5.5: Búsqueda local - Valores de la función de fitness para diferentes pesos y repeticiones

Después de aproximadamente once horas con un procesador Intel Core i3-6100U CPU 2.30GHz se consiguieron cien soluciones al problema utilizando la búsqueda local. Se observa nuevamente que la función de evaluación *udp.fitness()* proporcionada por la ESA es muy costosa desde el punto de vista de su cómputo.

Este conjunto con cien soluciones se subió a través de la plataforma proporcionada por la ESA. En el caso que nos ocupa, se obtuvo un valor de hipervolumen igual a -6,035.142. Se observa entonces que la búsqueda local proporciona mejores aproximaciones al Frente de Pareto en comparación con la búsqueda aleatoria, tal y como se esperaba.

Para cada una de las 10 repeticiones realizadas, se obtuvo una aproximación al Frente de Pareto combinando las soluciones obtenidas para cada uno de los pesos. El cálculo de los hipervolúmenes para las diez aproximaciones al Frente de Pareto también se realizó mediante la plataforma de la ESA. Los valores obtenidos se pueden observar en la Tabla 5.5 y su distribución, a través del diagrama de cajas de la Figura 5.6.

Tabla 5.5: Búsqueda local - Valores de hipervolumen obtenidos en cada repetición

Frente de Pareto	**Hipervolumen**
Repetición 1	-5671.083
Repetición 2	-5737.657
Repetición 3	-5703.154
Repetición 4	-5694.847
Repetición 5	-5611.071
Repetición 6	-5752.945
Repetición 7	-5802.821
Repetición 8	-5851.535
Repetición 9	-5841.918
Repetición 10	-5890.853

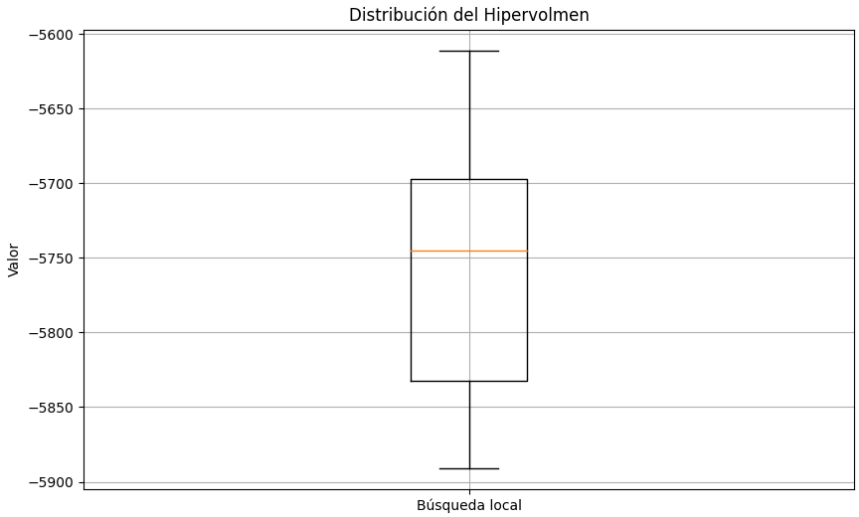


Figura 5.6: Búsqueda local - Distribución de los valores de hipervolumen

Conjuntamente, la Tabla 5.6 muestra varios estadísticos descriptivos del hipervolumen obtenido en las diez repeticiones. Dicha tabla proporciona una visión general del comportamiento del hipervolumen en la búsqueda local. En ella, se puede observar como la mejor de las aproximaciones al Frente de Pareto tiene un hipervolumen igual a -5890.853, correspondiendo a la décima repetición.

Tabla 5.6: Búsqueda local - Estadísticos Hipervolumen

Estadístico	Valor
Mínimo	-5890.853
Máximo	-5611.071
Media	-5755.7884
Mediana	-5745.3010

La mejor aproximación al Frente de Pareto (la correspondiente a la décima repetición, tal y como se ha indicado anteriormente), se puede observar en la Figura 5.7. Se tiene que los extremos corresponden a las soluciones obtenidas con peso 1 (extremo inferior-derecho) y con peso 0.1 (extremo superior-izquierdo).

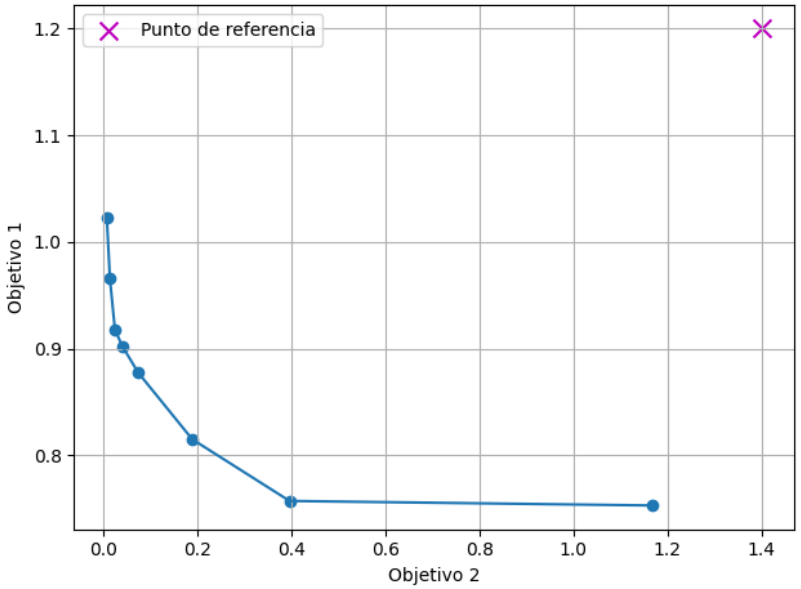


Figura 5.7: Búsqueda local - Mejor aproximación al Frente de Pareto

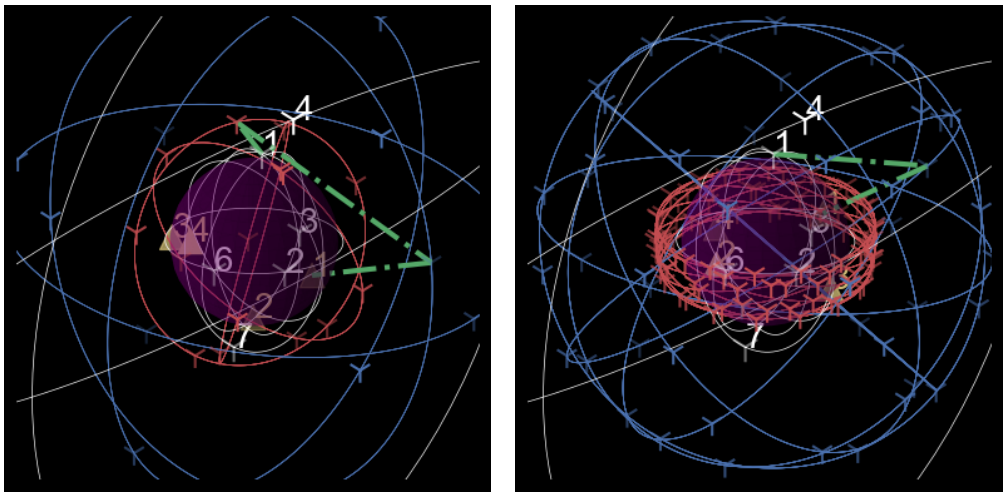
A partir de la función *udp.pretty()*, a la cual se le pasa como parámetro un cromosoma x, y que se encuentra definida en el código de evaluación proporcionado por la ESA, se ha obtenido la información con ambos extremos de la mejor aproximación del Frente de Pareto. Como salida se ha obtenido lo siguiente

```
FITNESS EVALUATION: Extremo superior-izquierdo
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 15, W2: 12): 27
OBJECTIVE 1 - Average communications cost: 1.022215073357422
OBJECTIVE 2 - Cost of infrastructure: 0.007684196058843211
CONSTRAINT - Minimum distance between rovers (OK): 3134.96776080528 km
CONSTRAINT - Minimum distance between sats (OK): 298.1598998172953 km
------------------------------------------------------------------------

(1.022215073357422, 0.007684196058843211, -134.9677608052798,
    -248.1598998172953)
```

```
FITNESS EVALUATION: Extremo inferior-derecho
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 90, W2: 49): 139
OBJECTIVE 1 - Average communications cost: 0.7528287425476556
OBJECTIVE 2 - Cost of infrastructure: 1.1687146311274823
CONSTRAINT - Minimum distance between rovers (OK): 9154.185525993144 km
CONSTRAINT - Minimum distance between sats (OK): 249.58759033781655 km
------------------------------------------------------------------------

(0.7528287425476556, 1.1687146311274823, -6154.1855259931435,
    -199.58759033781655)
```

Se observa que el extremo superior-izquierdo, con una menor cantidad de satélites (27 satélites), presenta un mayor coste medio de comunicaciones ($J_1 = 1.0222$) en comparación con el extremo inferior-derecho, que tiene una mayor cantidad de satélites (139 satélites) y un coste medio de comunicaciones más bajo ($J_1 = 0.7528$). Sin embargo, esta última solución, al contar con más satélites, resulta en un coste de infraestructuras considerablemente mayor ($J_2 = 1.1687$ frente a $J_2 = 0.0078$).

Además, a partir de la función *udp.plot()*, también proporcionada en el código de evaluación de la ESA, se han dibujado las órbitas correspondientes a las soluciones extremas de la mejor aproximación del Frente de Pareto. En azul se representa la constelación Walker 1, en rojo se representa la constelación Walker 2, en blanco se representan las naves nodrizas y en amarillo se representan los rovers. En la Figura 5.8(a) se pueden observar las órbitas de la solución correspondiente al extremo superior-izquierdo, mientras que en la Figura 5.8(b), se observan las órbitas de la solución correspondiente al extremo inferior-derecho. Además, en verde, se representa el camino entre una nave nodriza y un rover, en estos casos entre la nave nodriza 1 y el rover 1.



(a) Mothership 1 (node 27) communicates with rover 1 (node 34) at epoch 1 via: (27, 0, 20, 34)

(b) Mothership 1 (node 139) communicates with rover 1 (node 146) at epoch 1 via: (139, 92, 146)

Figura 5.8: Búsqueda local - Representación de las órbitas correspondientes a los puntos extremos de la mejor aproximación al Frente de Pareto

5.3. Algoritmo memético

Al igual que para la búsqueda aleatoria y la búsqueda local, para la búsqueda de soluciones con el algoritmo memético se ejecutó el código fuente descrito en la Sección 4.3. Los parámetros utilizados para la ejecución de este algoritmo se pueden ver en la Tabla 5.7.

Tabla 5.7: Parámetros del algoritmo memético

Parámetro	Variable	Valores
Número de generaciones	num_gen	70
Probabilidad de cruce	pc	0.7
Número sin mejoras	num_mejoras	5
Número de individuos	num_ind	10
Pesos	weight	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
Repeticiones	repeticiones	10

Al terminar las ejecuciones, se obtuvieron soluciones factibles en todos los casos, es decir, que cumplen las restricciones al problema. Los valores de la función de fitness utilizada (suma ponderada de ambas funciones objetivo), obtenidos para los diferentes pesos y repeticiones, se pueden observar en los diagramas de cajas de la Figura 5.9. Cabe destacar una vez más que, a medida

que aumenta el peso dado al objetivo 1, también aumenta el valor de la función de fitness. Esto significa que el objetivo 1 contribuye en mayor medida a la función de fitness.

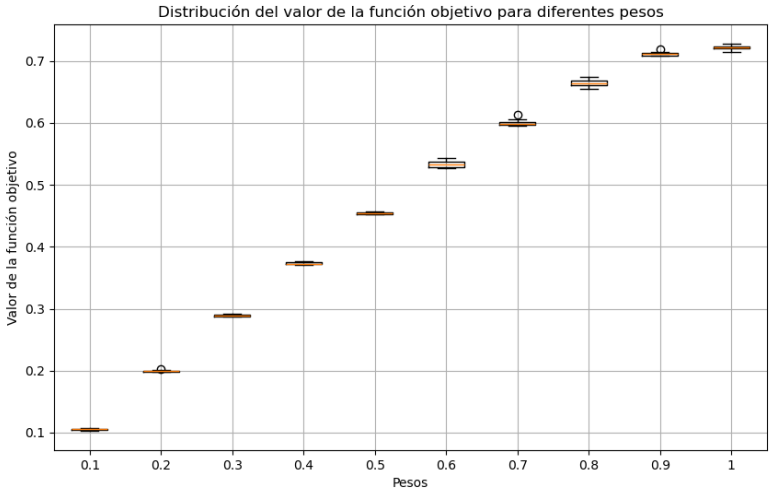


Figura 5.9: Algoritmo memético - Valores de la función de fitness para diferentes pesos y repeticiones

Después de aproximadamente 12 días ejecución con un procesador Intel Core i3-6100U CPU 2.30GHz, se consiguieron cien soluciones al problema utilizando el algoritmo memético. Se refleja de nuevo que la función de evaluación *udp.fitness()* proporcionada por la ESA es muy costosa desde el punto de vista de su cómputo.

Este conjunto con cien soluciones se subió a través de la plataforma proporcionada por la ESA. En este caso concreto, se obtuvo un valor de hipervolumen igual a -6,096.122.

Finalmente, tras haber ejecutado los tres algoritmos, la mejor aproximación al Frente de Pareto obtenida fue la correspondiente al algoritmo memético. En la imagen de la Figura 5.10 se puede observar como queda el ranking final de la ESA para este problema, siendo la puntuación presentada en este trabajo la correspondiente a “emarray”.

Leaderboard

	USERNAME	SCORE	SUBMISSION DATE
	fcmaes	-6,466.645	Sep 12, 2023, 8:29 PM
	ML Actonauts	-6,444.772	Jun 30, 2023, 9:01 PM
	Team HRI	-6,382.697	Jun 30, 2023, 12:03 PM
4.	theEs	-6,315.333	Apr 20, 2023, 9:54 PM
5.	CGG	-6,270.739	Jun 27, 2023, 2:10 AM
6.	TOPDEI	-6,232.813	Apr 23, 2023, 1:27 PM
7.	Marvin	-6,149.84	Jun 18, 2023, 7:47 PM
8.	emarray	-6,096.122	Jun 19, 2024, 8:23 PM
9.	Boolean_Autocrats	-5,372.039	Jun 25, 2023, 2:46 PM

Figura 5.10: Ranking de puntuaciones final - ESA

En la Figura 5.11 se puede ver la evolución del algoritmo memético en la repetición 1 con peso 0.9 a lo largo de las generaciones. Siendo esta repetición una de las que mejor resultado obtiene para ambos objetivos teniendo por valores objetivo $J_1 = 0.7567691475093402$ y $J_2 = 0.27143440824071136$.

Figura 5.11: Algoritmo memético -Evolución durante las generaciones

Para cada una de las 10 repeticiones realizadas, se obtuvo una aproximación al Frente de Pareto combinando las soluciones obtenidas para cada uno de los pesos. El cálculo de los hipervolúmenes para las diez aproximaciones al Frente de Pareto también se realizó mediante la plataforma de la ESA. Los valores obtenidos se pueden observar en la Tabla 5.8 y su distribución, a través del diagrama de cajas de la Figura 5.12.

Tabla 5.8: Algoritmo memético - Valores de hipervolumen obtenidos en cada repetición

Frente de Pareto	**Hipervolumen**
Repetición 1	-5989.31
Repetición 2	-6025.32
Repetición 3	-5898.501
Repetición 4	-6018.937
Repetición 5	-5994.621
Repetición 6	-5976.549
Repetición 7	-6035.779
Repetición 8	-5982.532
Repetición 9	-5950.307
Repetición 10	-5915.979

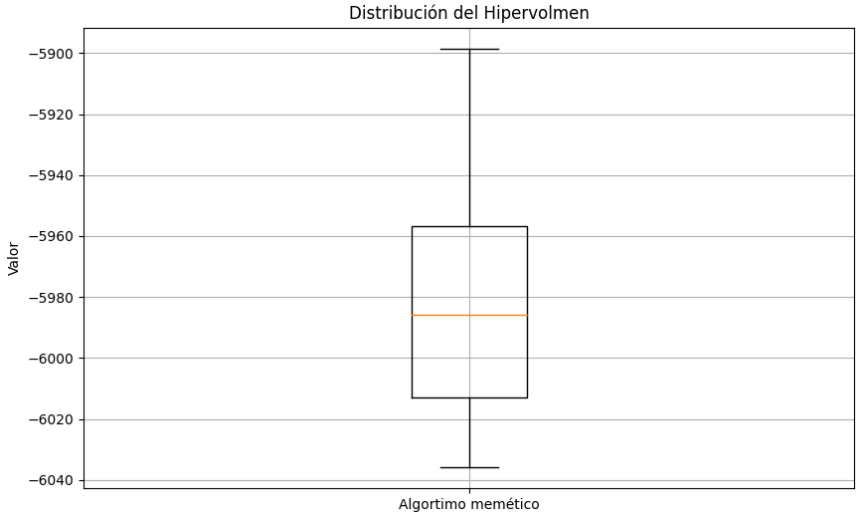


Figura 5.12: Algoritmo memético - Distribución de los valores de hipervolumen

A su vez, la Tabla 5.9 muestra varios estadísticos descriptivos del hipervolumen obtenido en las diez repeticiones. Dicha tabla proporciona una visión general del comportamiento del algoritmo memético. En ella, se puede observar

como la mejor de las aproximaciones al Frente de Pareto tiene un hipervolumen igual a -6035.779, correspondiendo a la séptima repetición.

Tabla 5.9: Algortimo memético - Estadísticos del hipervolumen

Estadístico	Valor
Mínimo	-6035.779
Máximo	-5898.501
Media	-5978.7835
Mediana	-5985.921

La mejor aproximación al Frente de Pareto (la correspondiente a la séptima repetición, tal y como se ha indicado anteriormente), se puede observar en la Figura 5.13. Se tiene que los extremos corresponden a las soluciones obtenidas con peso 1 (extremo inferior-derecho) y con peso 0.1 (extremo superior-izquierdo). Se puede ver como hay una solución cuyo objetivo 2 sobrepasa el punto de referencia. Esto se tiene debido a que el algoritmo memético con peso 1, aunque minimiza mejor el primer objetivo, como no tiene en cuenta el segundo objetivo pasa a tomar valores mayores llegando a pasar el punto de referencia del Frente de Pareto en todas las repeticiones. Cabe mencionar que dicho extremo, no se considerará en el cálculo del hipervolumen, a pesar de ser una solución factible del problema.

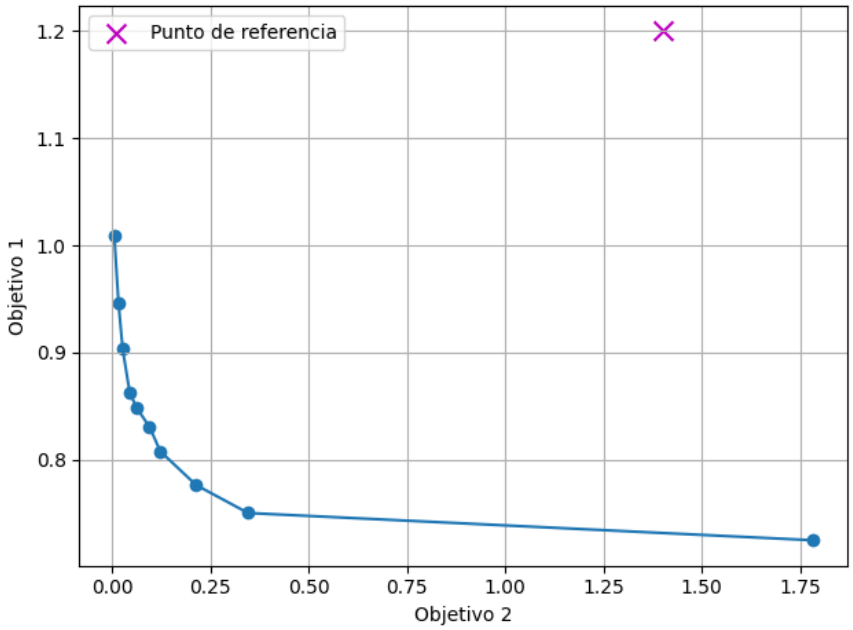


Figura 5.13: Algoritmo memético - Mejor aproximación al Frente de Pareto

A partir de la función *udp.pretty()*, a la cual se le pasa como parámetro un cromosoma x, y que se encuentra definida en el código de evaluación proporcionado por la ESA, se ha obtenido la información con ambos extremos de la mejor aproximación del Frente de Pareto. Como salida se ha obtenido lo siguiente:

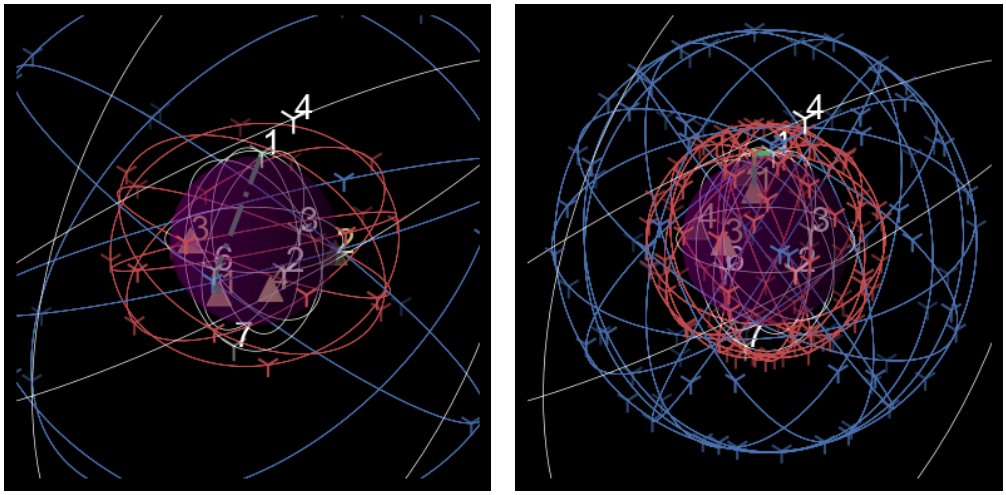
```
FITNESS EVALUATION: Extremo superior-izquierdo
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 20, W2: 20): 40
OBJECTIVE 1 - Average communications cost: 1.0085905325027336
OBJECTIVE 2 - Cost of infrastructure: 0.005199830132629857
CONSTRAINT - Minimum distance between rovers (OK): 3930.37341665266 km
CONSTRAINT - Minimum distance between sats (OK): 963.2870043496 km
------------------------------------------------------------------------

(1.0085905325027336, 0.005199830132629857, -930.3734166526601,
    -913.2870043496)
```

```
FITNESS EVALUATION: Extremo inferior-derecho
------------------------------------------------------------------------

RESULTS:
Total number of satellites (W1: 100, W2: 80): 180
OBJECTIVE 1 - Average communications cost: 0.7248920729952227
OBJECTIVE 2 - Cost of infrastructure: 1.7811928896181042
CONSTRAINT - Minimum distance between rovers (OK): 3287.7869701220375
    km
CONSTRAINT - Minimum distance between sats (OK): 128.16646098092443 km
------------------------------------------------------------------------

(0.7248920729952227, 1.7811928896181042, -287.7869701220375,
    -78.16646098092443)
```

Se observa que el extremo superior-izquierdo, con una menor cantidad de satélites (40 satélites), presenta un mayor coste medio de comunicaciones ($J_1 = 1.0086$) en comparación con el extremo inferior-derecho, que tiene una mayor cantidad de satélites (180 satélites) y un coste medio de comunicaciones más bajo ($J_1 = 0.7249$). Sin embargo, esta última solución, al contar con más satélites, resulta en un coste de infraestructuras considerablemente mayor ($J_2 = 1.7812$ frente a $J_2 = 0.0052$).

Además, a partir de la función *udp.plot()*, también proporcionada en el código de evaluación de la ESA, se han dibujado las órbitas correspondientes a las soluciones extremas de la mejor aproximación del Frente de Pareto. En azul se representa la constelación Walker 1, en rojo se representa la constelación Walker 2, en blanco se representan las naves nodrizas y en amarillo se representan

los rovers. En la Figura 5.14(a) se pueden observar las órbitas de la solución correspondiente al extremo superior-izquierdo, mientras que en la Figura 5.14(b), se observan las órbitas de la solución correspondiente al extremo inferior-derecho. Además, en verde, se representa el camino entre una nave nodriza y un rover, en estos casos entre la nave nodriza 1 y el rover 1.



(a) Mothership 1 (node 40) communicates with rover 1 (node 47) at epoch 1 via: (40, 31, 47)

(b) Mothership 1 (node 180) communicates with rover 1 (node 187) at epoch 1 via: (180, 38, 187)

Figura 5.14: Algoritmo memético - Representación de las órbitas correspondientes a los puntos extremos de la mejor aproximación al Frente de Pareto

5.4. Análisis comparativo de los métodos

A lo largo de esta sección se hará una comparación de los tres métodos de optimización implementados: Búsqueda Aleatoria (BA), Búsqueda Local (BS) y Algoritmo Memético (AM). La hipótesis inicial era que el Algoritmo Memético es mejor que los otros métodos.

Para la comparación de los algoritmos se utilizará el hipervolumen. En el diagrama de cajas que se muestra en la Figura 5.15 se puede observar la distribución del hipervolumen de las diez aproximaciones al Frente de Pareto obtenidas en las Secciones 5.1, 5.2 y 5.3 para cada algoritmo. A primera vista se puede ver como el AM tiene menor hipervolumen que la BA y la BL.

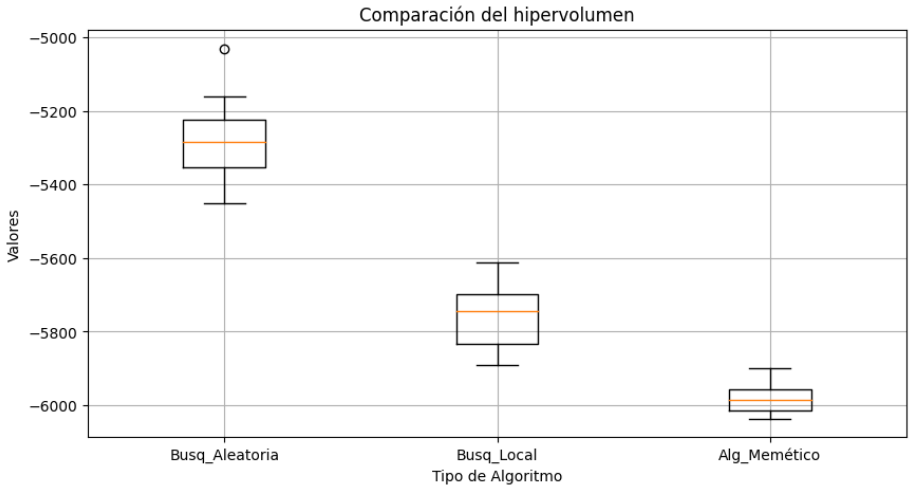


Figura 5.15: Boxplot comparativo de la distribución del hipervolumen

La Tabla 5.10 presenta estadísticas clave obtenidas de los diferentes algoritmos, incluyendo mínimo, máximo, media y mediana del hipervolumen. Estos resultados proporcionan una visión comparativa del comportamiento de cada algoritmo en términos del hipervolumen. A primera vista, se puede observar como el algoritmo memético consigue mejores resultados que los otros métodos.

Tabla 5.10: Comparación de los estadísticos del hipervolumen

Estadístico	**BA**	**BL**	**AM**
Mínimo	-5450.584	-5890.853	**-6035.779**
Máximo	-5030.761	-5611.071	**-5898.501**
Media	-5276.5791	-5755.7884	**-5978.7835**
Mediana	-5285.5765	-5745.3010	**-5985.921**

Para determinar si la diferencia entre los algoritmos es estadísticamente significativa, se aplica un test estadístico comparando los hipervolúmenes calculados previamente para cada algoritmo. La comparación se realiza dos a dos.

Primero, se utiliza el test de normalidad Shapiro-Wilk con un nivel de significación $\alpha = 0.05$. Los resultados de este test para los diferentes algoritmos se muestran en la Tabla 5.11. En todos los casos, se encontró que el valor p-valor $\geq \alpha$, lo que indica que las muestras siguen una distribución normal.

Tabla 5.11: Resultados Test Shapiro-Wilk

Algoritmo	**Estadístico**	**p-valor**
Búsqueda Aleatoria	0.9699	0.8908
Búsqueda Local	0.9701	0.8921
Algoritmo Memético	0.9353	0.5029

Dado que las muestras de los tres algoritmos siguen una distribución normal, el siguiente paso es determinar si las distribuciones tienen varianzas iguales. Esto es necesario para decidir si se debe aplicar un test ANOVA (en caso de varianzas iguales) o un test de Welch (en caso de varianzas diferentes). Para evaluar la igualdad de varianzas, se emplea el test de Bartlett.

Finalmente, con un nivel de significación $\alpha = 0.05$, se obtuvieron los resultados que se presentan en la Tabla 5.12. A partir de estos resultados, se concluye que el algoritmo memético es significativamente mejor que la búsqueda aleatoria y la búsqueda local, puesto que p-valor $< \alpha$, confirmando nuestra hipótesis inicial. Además, se encontró que la BL es significativamente mejor que la BA, se tiene también que p-valor $< \alpha$.

Estos resultados refuerzan la superioridad del algoritmo memético sobre la búsqueda aleatoria y la búsqueda local en la resolución del problema de optimización estudiado, tal como se observa en el diagrama de cajas de la Figura 5.15. El AM consigue mejores resultados porque combina la exploración global de algoritmos evolutivos con la explotación local de los algoritmos de búsquedas locales, permitiendo una búsqueda más eficaz y precisa en el espacio de soluciones. Por otro lado, la BL obtiene resultados mejores que la BA porque explora sistemáticamente el entorno cercano a una solución actual, lo que aumenta las probabilidades de encontrar soluciones óptimas o cercanas a óptimas de manera más eficiente.

Tabla 5.12: Resultados estadísticos de la comparación entre métodos

Comparación	**Test**	**Estadístico**	**p-valor**	**Interpretación**
Búsqueda Aleatoria vs Búsqueda Local	ANOVA	100.464	8.617e-09	Diferencia Significativa
Búsqueda Aleatoria vs Algoritmo Memético	Welch	17.067	1.675e-09	Diferencia Significativa
Búsqueda Local vs Algoritmo Memético	ANOVA	49.404	1.470e-06	Diferencia Significativa

Conclusiones

En este trabajo se abordó la resolución del problema *Quantum Communications Constellations*, perteneciente a una competición de la ESA, mediante diferentes técnicas algorítmicas. Inicialmente, se implementó una búsqueda aleatoria que sirvió como punto de referencia básica, aunque se identificó rápidamente como el método menos eficaz debido a su naturaleza aleatoria.

Posteriormente, se desarrolló y aplicó un algoritmo de búsqueda local, que mostró mejoras significativas en comparación con la búsqueda aleatoria. Esto se debe a su capacidad para explorar sistemáticamente el entorno cercano a una solución actual, incrementando así las probabilidades de hallar soluciones óptimas o cercanas a óptimas de manera más eficiente.

Finalmente, se implementó un algoritmo memético, que combina técnicas de algoritmos evolutivos con búsqueda local. A pesar de requerir un tiempo considerable para su ejecución, este algoritmo logró obtener mejores resultados en términos de rendimiento que los métodos anteriores. Para validar estadísticamente su superioridad, se realizaron pruebas comparativas entre los diferentes algoritmos, confirmando que el algoritmo memético ofrecía soluciones significativamente mejores. Esto se debe a la combinación de la exploración global de los algoritmos evolutivos y la explotación local de los algoritmos de búsqueda local, lo que permite una búsqueda más eficaz y precisa en el espacio de soluciones.

En conclusión, aunque no se logró una posición destacada en la clasificación de la competición, se demostró que tanto los algoritmos simples como los meméticos sencillos pueden proporcionar soluciones competitivas al problema. Los resultados indican que el algoritmo memético ofrece soluciones de mayor calidad en comparación con las otras técnicas evaluadas para el problema de las constelaciones de comunicaciones cuánticas.

Para futuras investigaciones, se propone incorporar criterios de optimización y restricciones adicionales. El algoritmo memético utilizado es simple, tanto en los operadores de variación empleados, como en el procedimiento de mejora utilizado. Por ello, explorar operadores más complejos y sofisticados que conduzcan a mejores soluciones representa una línea de investigación de interés.

Además, se podrían implementar algoritmos multi-objetivos, los cuales permiten optimizar simultáneamente varios criterios de desempeño. Estos algoritmos son especialmente útiles en problemas complejos como el de las constelaciones de comunicaciones cuánticas, donde es necesario balancear múltiples objetivos.

Bibliografía

[1] A.E. Eiben, *Introduction to Evolutionary Computing*, Second edition, Springer Nature, Berlin, Heidelberg, 2015. ISBN: 9783662448748.

[2] Carlos Coello Coello, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed., Springer Nature, Netherlands, 2007. ISBN: 0387367977.

[3] Python Software Foundation. *Python: A programming language for scientific computing*. Python Software Foundation, Delaware, USA. https://www.python.org/.

[4] Google Research. *Google Colaboratory: A platform for machine learning and data science*. Google, Mountain View, CA, USA. https://colab.research.google.com/.

[5] Anaconda, Inc. *Anaconda: The World's Most Popular Data Science Platform*. Anaconda, Inc., USA. https://www.anaconda.com/.

[6] Microsoft Corporation. *Visual Studio Code: Code Editing. Redefined.*. Microsoft Corporation, USA. https://code.visualstudio.com/.

[7] Dietmar Wojtas, *fast-cma-es: A Python implementation of CMA-ES for evolutionary optimization*,2024. https://github.com/dietmarwo/fast-cma-es/blob/master/tutorials/ESAChallenge.adoc.

[8] European Space Agency. SPOC-2 Quantum Communications Constellations. ESA Optimization Benchmark Challenge. https://optimize.esa.int/challenge/spoc-2-quantum-communications-constellations/About.

[9] GECCO 2024, Association for Computing Machinery, 2024. https://gecco-2024.sigevo.org/HomePage

[10] Vélez, M.C., Castro, C.A., y Maya, J. Algoritmo de búsqueda aleatoria para la programación de la producción en un taller de fabricación. *REVISTA Universidad EAFIT*, Vol. 39, No. 131, pp. 76-86, 2003.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[12] Ferrante Neri, Carlos Cotta, Pablo Moscato, *Handbook of Memetic Algorithms*, 1st ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, ISBN 3-642-23247-7.

Combinatorial optimisation through evolutionary computation

Elena Martín Raya
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101333519@ull.edu.es

Abstract

The study addresses the challenge of establishing reliable communications between the surface of New Mars and orbiting spacecraft, known as "Quantum Communications Constellations." It uses three methods: Random Search (RS), Local Search (LS), and Memetic Algorithm (MA) to tackle the combinatorial optimization problem. The results show that the Memetic Algorithm significantly outperforms the others according to hypervolume measurements, validating its superiority. This study makes a significant contribution to quantum communications by highlighting the effectiveness of the memetic approach in optimizing complex systems.

1. Introduction

This paper addresses the optimization problem of "Quantum Communications Constellations," proposed by the European Space Agency (ESA). The objective is to optimize the design of quantum satellite constellations to enhance the efficiency of communications between rovers on New Mars and orbiting mother ships, while minimizing operational costs and avoiding orbital collisions. Evolutionary computation is employed to find optimal and collision-free orbital configurations, drawing inspiration from biological evolution principles to solve the optimization problem.

2. Problem and Algorithms

Quantum Communications Constellations problem is a multi-objective combinatorial optimization problem. The two objectives of the problem are to minimize the average shortest communication cost (J_1) and to minimize the total cost of manufacturing, launching, and operating both constellations (J_2). To solve this problem, a weighted function was used: $f(x) = J_1 \cdot w + J_2 \cdot (1 - w)$ where $w \in [0, 1]$. This transforms the problem into a single-objective minimization.
Algorithms used to solve the problem:

- **Random Search (RS):** Involves generating a large number of solutions randomly within the possible solution set for an optimization problem and selecting the best one.
- **Local Search (LS):** An iterative optimization technique that improves a candidate solution by making small changes to its structure, evaluating neighboring solutions, and accepting improvements until a local optimum is reached.
- **Memetic Algorithm (AM):** An optimization technique that combines Local Search with population-based methods, such as Evolutionary Algorithms. It starts with a set of high-quality or random solutions, improves these solutions using Local Search, and uses recombination procedures to create new solutions.

3. Comparison of results

To compare the algorithms, the hypervolume is used. An algorithm with a lower hypervolume is better than those with a higher hypervolume. In the box plot of Figure 1, it can be observed that the AM presents a lower hypervolume compared to RS and LS.

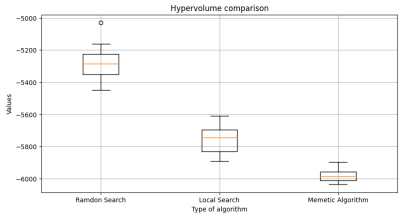


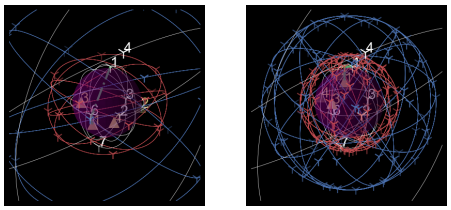
Figure 1: *Comparative boxplot of hypervolume distribution*

Furthermore, Table 1 shows the descriptive statistics: minimum, maximum, mean, and median for all evaluated algorithms. The results indicate that the MA not only has the lowest hypervolume but is also significantly better according to the results of a statistical test applied, which underscores the superiority of the MA method in our study.

Table 1: *Statistics regarding hypervolume values*

Statistical	RS	LS	MA
Min	-5450.584	-5890.853	-6035.779
Max	-5030.761	-5611.071	-5898.501
Mean	-5276.5791	-5755.7884	-5978.7835
Median	-5285.5765	-5745.3010	-5985.921

In Figure 2, the representation of the orbits corresponding to the extreme points of the best approximation to the Pareto Front is shown. The upper left extreme has a smaller number of satellites (40 satellites), while the lower right extreme has a larger number of satellites (180 satellites).



(a) Upper left extreme (b) Lower right extreme

Figure 2: *Representation of Orbits Corresponding to the Extreme Points of the Best Approximation to the Pareto Front*

References

[1] European Space Agency. SPOC-2 Quantum Communications Constellations. ESA Optimization Benchmark Challenge. https://optimize.esa.int/challenge/spoc-2-quantum-communications-constellations/About.

[2] A.E. Eiben, *Introduction to Evolutionary Computing*, Second edition, Springer Nature, Berlin, Heidelberg, 2015. ISBN: 9783662448748.

[3] Ferrante Neri, Carlos Cotta, Pablo Moscato, *Handbook of Memetic Algorithms*, 1st ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, ISBN 3-642-23247-7.