Técnicas de Soft Computing para Aprendizaje y optimización. Redes Neuronales y Metaheurísticas, programación evolutiva y bioinspirada
2024-2025

Master Ciencia de Datos
Universidad de Granada

Revisión de los Algoritmos Genéticos y problema M2NP

Miguel García López

Índice

Índice de figuras

1. Introducción

En esta revisión se va a proceder a explicar qué son los **algoritmos genéticos**, de dónde vienen y cómo surgen a lo largo del tiempo como una solución bio-inspirada para solucionar problemas de optimización con codificación binaria. Se explicará cómo estos algoritmos surgieron y fueron "evolucionandoçon el tiempo, adaptándose a nuevos problemas y creando variaciones del algoritmo original.

Se recabará información a través del buscador de *Scopus* para realizar un análisis bibliométrico, con el que se analizará la producción científica y académica y se medirá el impacto, la calidad y la evolución de publicaciones, autores, revistas e instituciones en el área específica de los algoritmos genéticos.

Se explicará, además, el funcionamiento básico del algoritmo genético original y el de algunas de sus variantes e hibridaciones. Para ello se explicarán los operadores que lo conforman así como el *workflow* o procesos que sigue el algoritmo y cómo interactúan los operadores entre sí para hacer funcionar al algoritmo de optimización.

Además, después de la revisión de algoritmos genéticos, se explicará el problema de *Multidimensional Two-way Number Partitioning* o **M2NP**. Se realizará una búsqueda bibliográfica sobre el problema, se explicarán una serie de soluciones propuestas para resolver este problema, entre ellas **Greedy**, **Iterated Greedy**, y se aplicarán y explicarán una serie de operadores de representación así como un algoritmo genético adaptado al problema.

2. Contexto

Los algoritmos genéticos están inspirados en la selección natural y se emplean tanto en problemas de optimización con restricciones como en aquellos sin ellas. Esta metaheurística modifica de manera repetida una población de soluciones individuales, seleccionando soluciones "padre"que generarán la siguiente generación de soluciones en la siguiente iteración del algoritmo.

En su forma más básica, un algoritmo genético opera sobre una población de soluciones potenciales a un problema dado. Cada solución potencial, frecuentemente llamada individuo o cromosoma, está representada como una cadena de símbolos, que puede ser binaria, numérica o simbólica. [1].

2.1. Orígenes

Los algoritmos genéticos o **GAs** adquirieron popularidad en la década de 1970, especialmente en 1975 con la publicación del libro de John Holland [2]. Este tipo de me-

taheurísticas se diseñan tomando como inspiración la selección natural. Un conjunto de fenotipos (soluciones) evoluciona a lo largo de generaciones para emular el cruce entre especies, es decir, el cruce de soluciones mediante un intercambio común de cromosomas, lo que da lugar a nuevos individuos con características de ambos padres. A lo largo del tiempo, este tipo de algoritmos fueron incorporando nuevas características y, aunque inicialmente fueron concebidos para resolver problemas discretos, también existen versiones que optimizan problemas continuos [3].

2.2. Definición

Los algoritmos genéticos definen las soluciones como vectores numéricos, en el caso de un problema binario (**GAs** originales) la solución es un vector de $0s$ y $1s$, en el caso de un problema sobre un dominio continuo, la solución es un vector de números reales. Cada individuo $x \in P$ se representa como una cadena de genes:

$$x = (g_1, g_2, ..., g_n)$$

donde $g_i \in \Sigma$ y Σ es el alfabeto de genes que puede ser:

- Binario: $\Sigma = \{0, 1\}$
- Entero: $\Sigma = \mathbb{Z}$
- Real: $\Sigma = \mathbb{R}$

El algoritmo trata de minimizar o maximizar una función objetivo, esta es conocida como la función *fitness*. En esta se representa una métrica de cómo de bien lo está haciendo el algoritmo (o el error) y se trata de optimizar para guiar la evolución de la población.

Se comienza con una población de soluciones totalmente aleatoria partiendo de una distribución (puede ser normal o uniforme). A partir de ahí se evalúan los individuos y se les proporciona un *score* o evaluación a cada uno. Con esa evaluación se ordenan y se eligen dos padres (o varios, dependiendo de la variante) mediante un proceso de selección, como el método de la ruleta.

Esos padres son cruzados entre sí mediante el operador de cruce para obtener uno o más hijos que compartan características de los padres. Dada una probabilidad defindida como p se le aplicará el operador de mutación a los hijos para favorecer la diversidad y exploración.

A partir de ahí, se introducen los hijos en la población, si se pueden introducir, y se vuelve a iterar. La condición de parada puede ser llegar a un valor de *fitness*, condición de tiempo o condición de iteraciones máximas.

Algorithm 1 Algoritmo Genético con Elitismo

Require: Tamaño de población N, tasa de cruce C, tasa de mutación M, porcentaje de elitismo E, condición de terminación
Ensure: La mejor solución encontrada

Inicializar población aleatoria P con N individuos
while no se cumpla la condición de terminación **do**
    Evaluar fitness de cada individuo en P
    Ordenar P de mejor a peor según fitness
    $P_{\text{elite}} \leftarrow$ primeros $E \times N$ individuos de P
    Inicializar $P_{\text{descendencia}} \leftarrow \emptyset$
    for $i \leftarrow 1$ **to** N **do**
        Seleccionar padres p_1, p_2 de P usando selección por ruleta/torneo
        if rand() $< C$ **then**
            $h \leftarrow \text{Cruce}(p_1, p_2)$
        else
            $h \leftarrow$ copia de p_1 o p_2 (aleatorio)
        end if
        if rand() $< M$ **then**
            $h \leftarrow \text{Mutación}(h)$
        end if
        Agregar h a $P_{\text{descendencia}}$
    end for
    Reemplazar los $E \times N$ peores individuos en $P_{\text{descendencia}}$ con P_{elite}
    $P \leftarrow P_{\text{descendencia}}$
end while
return mejor individuo en P

2.3. Operadores principales

Se procede a definir los operadores principales de los algoritmos genéticos, al menos los básicos o más usados, ya que cubrir todos los operadores de todas las variaciones es inabarcable.

2.4. Selección

La selección se puede llevar a cabo por varios métodos. Uno de los más utilizados es **selección por torneo** (eq: 1). En la selección por torneo, los individuos compiten entre sí en grupos pequeños. Cada grupo (torneo) tiene varios competidores. El competidor con mejor *fitness* dentro del grupo tiene más probabilidades de ganar. Los ganadores son seleccionados para participar en el proceso de cruce. El proceso se repite hasta que se completa la *pool* de apareamiento con los ganadores de los torneos [4].

$$P(\text{parent}_i) = \frac{\text{fitness}(\text{parent}_i)}{\sum_{j=1}^{N} \text{fitness}(\text{parent}_j)} \tag{1}$$

2.5. Cruce

El operador de cruce (*crossover*) tiene como propósito principal explorar nuevas regiones del espacio de búsqueda combinando las características favorables de dos soluciones "padres"para generar nuevas soluciones "hijas"potencialmente mejores. Algunas versiones realizan cruce siempre, mientras que otras proponen una probabilidad para que dos soluciones puedan cruzarse, de forma que ocurra más esporádicamente.

One-point crossover (binario): Se elige al azar un punto en los cromosomas de ambos progenitores y se designa como "punto de cruce". Los bits a la derecha de ese punto se intercambian entre los dos cromosomas parentales. El resultado son dos descendientes, cada uno con información genética de ambos progenitores [5]. Dados dos padres (los cromosomas) $P_1 = (p_1^1, p_2^1, \ldots, p_n^1)$ y $P_2 = (p_1^2, p_2^2, \ldots, p_n^2)$, se selecciona un punto de cruce k aleatorio. Los hijos H_1 y H_2 se generan como:

$$H_1 = (p_1^1, p_2^1, \ldots, p_k^1, p_{k+1}^2, \ldots, p_n^2) \tag{2}$$

$$H_2 = (p_1^2, p_2^2, \ldots, p_k^2, p_{k+1}^1, \ldots, p_n^1) \tag{3}$$

Blend crossover (continuo): Dado dos números reales para cada uno de los genes de los padres al hijo se le asignará un número aleatorio entre ese rango de gen para cada

gen que conforme al vector cromosómico [6]. Dados dos padres $P_1 = (p_1^1, p_2^1, \ldots, p_n^1)$ y $P_2 = (p_1^2, p_2^2, \ldots, p_n^2)$, para cada gen i, se calcula un intervalo $[c_{\min}, c_{\max}]$, donde:

$$c_{\min} = \min(p_i^1, p_i^2) - \alpha \cdot d \tag{4}$$

$$c_{\max} = \max(p_i^1, p_i^2) + \alpha \cdot d \tag{5}$$

$$d = |p_i^1 - p_i^2| \tag{6}$$

El valor del gen i en el hijo H se genera aleatoriamente dentro del intervalo $[c_{\min}, c_{\max}]$:

$$H_i = \text{rand}(c_{\min}, c_{\max}) \tag{7}$$

Por supuesto existen muchos más tipos de cruces tanto para versiones en dominio continuo como en dominio discreto y binario, pero los principales operadores y que más suelen utilizarse son los descritos.

2.6. Mutación

El operador de mutación tiene como propósito principal favorecer la exploración del espacio de búsqueda introduciendo perturbaciones aleatorias en las soluciones hijo. Este operador es más delicado que el de cruce y por lo general se propone un valor bajo de probabilidad de mutación, ya que demasiadas perturbaciones pueden afectar a los resultados. Un desajuste entre el ratio de exploración y explotación a favor de la exploración puede dar como resultado soluciones con un *fitness* muy malo.

Mutación binaria: Este operador básico y clásico consiste en cambiar un *bit* arbitrario de un genotipo o solución de un algoritmo genético binario a su estado inverso dada una probabilidad de mutación [7].

$$x_i' = \begin{cases} 1 - x_i & \text{con probabilidad } p_m, \\ x_i & \text{con probabilidad } 1 - p_m. \end{cases} \tag{8}$$

Mutación continua: Este tipo de mutación se utiliza especialmente en problemas en los que se busca una exploración más amplia del espacio de búsqueda, ya que la distribución de *Cauchy* tiene colas pesadas y puede generar valores aleatorios alejados del centro con mayor probabilidad en comparación con una distribución normal. La mutación se define como:

$$m(x_i) = \begin{cases} (2r)^{\frac{1}{\eta+1}} - 1 & \text{si } p \leq 0{,}5, \\ 1 - (2r)^{\frac{1}{\eta+1}} & \text{si } p > 0{,}5, \end{cases} \tag{9}$$

donde:

- r es un número aleatorio en el intervalo $[0, 1]$,
- η es una variable de control que ajusta la distribución,
- p es una probabilidad aleatoria en $[0, 1]$.

Esta función de mutación está diseñada para generar una exploración más amplia del espacio de búsqueda, con la posibilidad de generar valores más alejados del centro de la distribución.

Por supuesto y tal como se mencionó en el apartado de *crossover*, existen multitud de propuestas en cuanto a operadores de mutación.

3. Informe bibliométrico

Como puede verse en la gráfica de la figura 1, los algoritmos genéticos son, pese a su longeva existencia, cada vez más populares. El número de artículos por año crece y crece cada vez más sin ninguna tendencia evidente a bajar, lo que parece indicar que los algoritmos genéticos son más populares que nunca y se sigue innovando en este sector.

Estos datos han sido obtenido mediante el buscador de *Scopus*. Utilizando una herramienta como *OpenAlex* se han obtenido datos complementarios. Por ejemplo, se puede observar que la gran mayoría de las publicaciones son (fig 2) cerradas (se accede mediante pasarelas de pago o acceso institucional). Otros datos interesantes se pueden obtener de este portal, como cuáles son las instituciones más prolíficas (fig 3) en relación a los **GAs** y cuáles son los tipos de publicaciones más comunes (fig 4) (un 88 % son artículos).

A partir de los artículos más citados en Scopus relacionados con Genetic Algorithms (**GAs**), los temas más candentes parecen estar relacionados con:

1. **Optimización y mejora de algoritmos genéticos**
   - Revisión del estado del arte de los **GAs** y sus mejoras a lo largo del tiempo.
   - Optimización de hiperparámetros en modelos de aprendizaje automático, clave para la implementación eficiente de **GAs**.
2. **Metaheurísticas bioinspiradas y nuevas variantes**

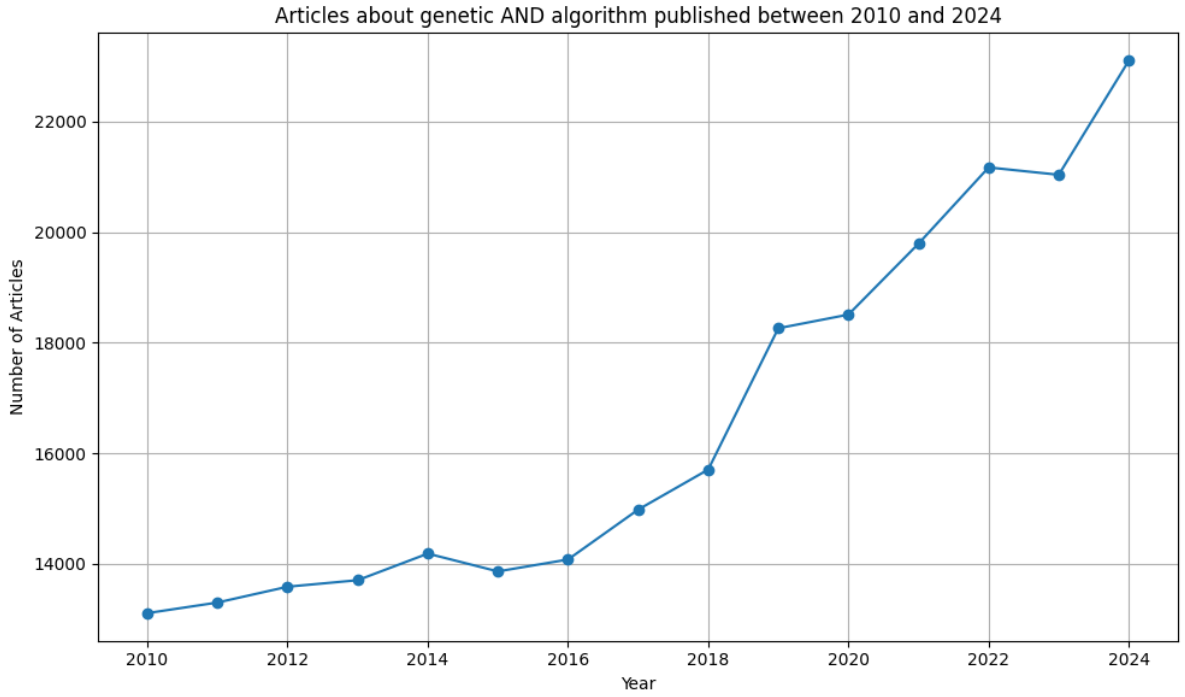

Figura 1: Número de artículos desde $2010-2024$ relacionados con los algoritmos genéticos en *Scopus*.

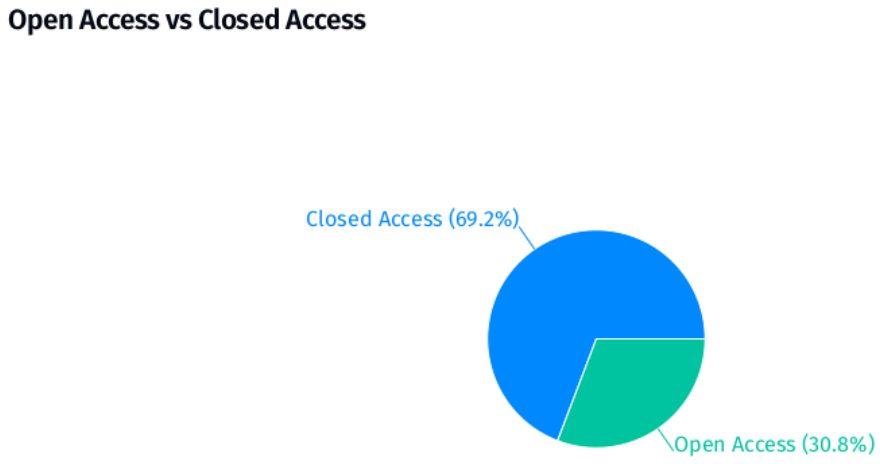

Figura 2: Publicaciones cerradas vs abiertas sobre **GAs**.

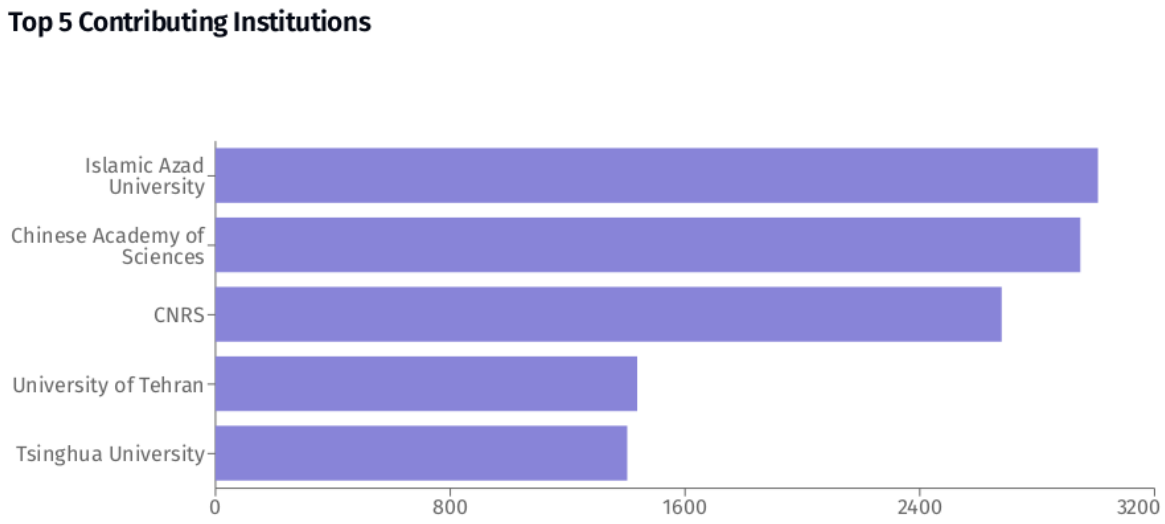

Figura 3: Instituciones que más publican sobre **GAs**.

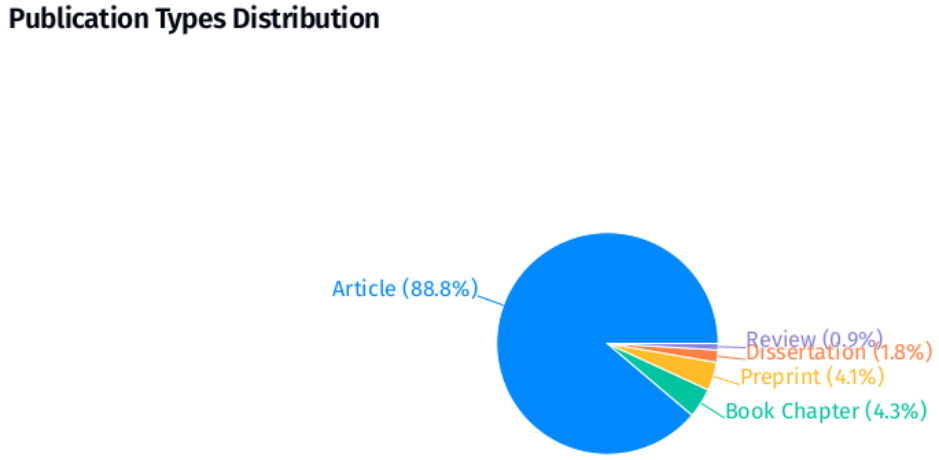

Figura 4: Tipo de publiaciones sobre las **GAs**.

- Desarrollo de nuevos algoritmos de optimización como *Marine Predators Algorithm*, *Equilibrium Optimizer* y *Arithmetic Optimization Algorithm*.
- Técnicas para mejorar la convergencia y evitar estancamientos en óptimos locales.

3. **Aplicaciones en biología y bioinformática**
   - Uso de **GAs** para predecir estructuras de proteínas con aprendizaje profundo.
   - Aplicaciones en la clasificación evolutiva de sistemas *CRISPR-Cas*.
4. **Optimización multiobjetivo y herramientas avanzadas**
   - Desarrollo de herramientas como *Pymoo*, una biblioteca en Python para optimización multiobjetivo con **GAs** y otras metaheurísticas.

4. Hibridaciones

Los algoritmos híbridos entre genéticos y otras técnicas de búsqueda son muy comunes y ampliamente utilizados. Los algoritmos genéticos se combinan con otros métodos de optimización, como búsqueda local o enfriamiento simulado, con el objetivo de mejorar la eficiencia y precisión.

El objetivo detrás de este tipo de algoritmos es utilizar a las **GAs** como método principal de búsqueda y exploración. Tras ir poco a poco convergiendo a una solución, se suelen utilizar métodos de búsqueda local o enfriamiento para mejorar y refinar la fase de explotación del algoritmo.

A continuación se detallan algunos de estos métodos [8] de hibridación y se explicarán las ventajas de cada uno y algunos ejemplos dentro de estos métodos de hibridación.

4.1. Hibridaciones colaborativas

Estas estrategias implican la interacción entre algoritmos independientes que intercambian información durante su ejecución.

4.1.1. Colaborativas *teamwork*

Varios algoritmos se ejecutan en paralelo, compartiendo información periódicamente.

1. Ejemplo 1: Los **GA** distribuidos (**DGAs**) dividen la población en subpoblaciones (islas), cada una procesada por un **GA** independiente. Las migraciones entre islas

introducen diversidad. Por ejemplo, en [9], se utilizan subpoblaciones con distintos parámetros (alta mutación para exploración, baja para explotación).

2. Ejemplo 2: En [10], se combinan **GA**, búsqueda tabú y búsqueda local. Los **GA** generan soluciones en zonas no exploradas usando una memoria adaptativa que registra el historial de búsqueda de otros algoritmos.

4.1.2. Colaborativas *relay*

Los algoritmos se ejecutan secuencialmente en una canalización.

1. Ejemplo 1: Un **GA** global seguido de un **GA** local. El primero explora el espacio de búsqueda, y el segundo refina las mejores soluciones encontradas.
2. Ejemplo 2: Un **GA** se combina con el método Nelder-Mead: el **GA** realiza una exploración global, y Nelder-Mead mejora las soluciones prometedoras. El método de Nelder-Mead es un método numérico utilizado para encontrar el mínimo o el máximo de una función objetivo en un espacio multidimensional. Es un método de búsqueda directa (basado en la comparación de funciones) y suele aplicarse a problemas de optimización no lineal para los que pueden no conocerse las derivadas.

4.2. Hibridaciones integrativas

Un algoritmo actúa como componente de otro, integrándose en su funcionamiento interno.

4.2.1. Integrativas *teamwork*

Un metaheurística se incrusta dentro de otra como componente clave.

1. Ejemplo 1: Algoritmos meméticos [11], donde un **GA** se combina con búsqueda local.
2. Ejemplo 2: Micro-GA (μ**GA**) [12], usado como operador de refinamiento. Un **GA** con población pequeña (ej. 5 individuos) mejora soluciones específicas del **GA** principal, aprovechando su capacidad para seguir crestas en espacios complejos.

4.2.2. Integrativas *relay*

Un **GA** realiza funciones específicas dentro de otra metaheurística dominante.

1. Ejemplo 1: Un **GA** *steady-state* genera soluciones candidatas para un *simulated annealing* (**SA**). El SA decide aceptar o rechazar las soluciones, integrando exploración (**GA**) y explotación (**SA**).
2. Ejemplo 2: En [13], un μ**CHC** (variante de **GA** con alta presión selectiva) actúa como operador de perturbación en una búsqueda local iterada, introduciendo diversidad sin perder calidad.

Las hibridaciones de algoritmos genéticos representan un campo en constante evolución dentro de la computación evolutiva y la optimización metaheurística. El éxito demostrado por estas técnicas híbridas ha llevado a un creciente interés en desarrollar nuevas combinaciones y estrategias que puedan abordar problemas cada vez más complejos. La tendencia actual apunta hacia la integración con técnicas de aprendizaje profundo y otros métodos de inteligencia artificial.

5. Problema Multidimensional Two-way Number Partitioning

El problema **M2NP** es un problema de optimización binario que consiste en partir un conjunto de vectores enteros S en dos grupos disjuntos de forma que se minimice la máxima diferencia entre la suma por coordenadas de los elementos de cada grupo. Específicamente, dado un conjunto de n vectores de dimensión d, $S = \{w_i | w_i = (w_{i1}, w_{i2}, ..., w_{id}), i = 1, ..., d)\}$, el objetivo de este problema consiste en repartir los elementos de S en dos conjuntos S_1, S_2 tal que $S_1 \cap S_2 = \varnothing$ y $S_1 \cup S_2$ y t es mínimo, siendo t:

$$t = max\{|\sum_{i \in S_1} w_{ij} - \sum_{i \in S_2} w_{ij}| : j = 1, ..., d\} \tag{10}$$

5.1. Revisión bibliográfica

Se han estudiado diversas aproximaciones para resolver el problema **M2NP**.

Los primeros trabajos, como el de [14] en 2010, trataron de formular el problema como uno de programación lineal. En este trabajo se demuestra que este problema, una generalización del problema clásico de partición de números, es **NP-difícil** y más complejo que su versión unidimensional. Se proponen restricciones y una formulación matemática para resolverlo con solvers de **ILP** como **CPLEX**, pero los resultados experimentales muestran que el problema sigue siendo difícil de resolver para dimensiones altas. Se sugiere el uso de métodos exactos o heurísticos como futuras líneas de investigación.

En [15], de 2013, se presenta un algoritmo memético para resolver el problema. La propuesta combina un algoritmo genético con una búsqueda local, mejorando la calidad

y el tiempo de solución en comparación con el método basado en programación entera mixta con **CPLEX**. Los resultados experimentales muestran que el enfoque memético supera a **CPLEX** y a los algoritmos genéticos puros ([16]) en términos de eficiencia y precisión.

Más tarde en 2014, el trabajo [17] presenta dos enfoques metaheurísticos, *Variable Neighborhood Search* (**VNS**) y *Electromagnetism-like* (**EM**), para resolver el problema. Ambos métodos utilizan procedimientos de búsqueda local y se comparan con resultados de la literatura, demostrando que superan a otros métodos existentes, con **EM** mostrando un ligero mejor rendimiento general.

Tres años después, en [18] se propone un nuevo método para resolver el problema de **M2NP** combinando **GRASP** (*Greedy Randomized Adaptive Search Procedure*) con una variante de *Path Relinking* llamada *Exterior Path Relinking* (**ePR**). Además, se introduce un nuevo procedimiento de búsqueda local restringida (**RFI**) que mejora la eficiencia del algoritmo. Los experimentos computacionales muestran que esta combinación supera a los métodos existentes, como **VNS** y CPLEX, especialmente en instancias grandes. El enfoque propuesto aprovecha la construcción heurística, la búsqueda local restringida y el esquema de *Path Relinking* para obtener soluciones de alta calidad en tiempos razonables.

Se publican varios artículos en relación al problema, donde el último que parece realizar mejoras sobre en el *estado del arte* es de 2021 en [19], donde se presenta **IMADEB**, un algoritmo memético de evolución diferencial algebraica mejorado para resolver el problema **M2NP**. **IMADEB** supera a su predecesor **MADEB** y a **GRASP+ePR** al utilizar una representación de bits no redundante, un operador de búsqueda local optimizado y una mutación diferencial adaptativa basada en vuelos de *Lévy*, logrando 145 nuevas soluciones óptimas y estableciendo un nuevo estándar en el campo.

5.2. Greedy y Iterated Greey

Básicamente, la búsqueda voraz ordena el vector S inicialmente por su norma en orden decreciente. De esta forma se asegura que el vector/conjunto principal tiene primero aquellos vectores con mayor magnitud, es decir, sus elementos son los más grandes, por lo que a la hora de asignarlos de forma iterativa a los diferentes conjuntos es más cómodo. Se guardan en un grupo u otro intentando mantener un equilibrio entre la diferencia de potenciales sumas de los vectores, de forma que se va equilibrando la adición de vectores de forma voraz.

El algoritmo iterativo voraz o *Iterate Greedy* aplica un número de intercambios (perturbaciones) de forma aleatoria sobre la solución de *greedy* un número de veces igual a las iteraciones permitidas. Se queda con la mejor solución. De esta forma se incorpora una aleatoriedad sobre el algoritmo que hace que este explore de forma aleatoria el espacio de soluciones, evitando mínimos locales.

Algorithm 2 Greedy

procedure HeurísticaVoraz(S, num_swaps)
    Ordenar vectores en S por norma descendente
    Aplicar num_swaps intercambios aleatorios en el orden
    $S1 \leftarrow \emptyset$, $S2 \leftarrow \emptyset$
    $sum1 \leftarrow \vec{0}$, $sum2 \leftarrow \vec{0}$
    for cada vector v_i en orden modificado **do**
        $sum1_{temp} \leftarrow sum1 + v_i$
        $sum2_{temp} \leftarrow sum2 + v_i$
        $diff1 \leftarrow \text{máx}(|sum1_{temp} - sum2|)$
        $diff2 \leftarrow \text{máx}(|sum1 - sum2_{temp}|)$
        if $diff1 \leq diff2$ **then**
            $S1 \leftarrow S1 \cup \{v_i\}$
            $sum1 \leftarrow sum1_{temp}$
        else
            $S2 \leftarrow S2 \cup \{v_i\}$
            $sum2 \leftarrow sum2_{temp}$
        end if
    end for
    return $(S1, S2)$
end procedure

Algorithm 3 Iterated Greedy

procedure HeurísticaIterativa(S, $iteraciones$, num_swaps)
    $mejor\_sol \leftarrow \emptyset$
    $mejor\_valor \leftarrow \infty$
    for $i \leftarrow 1$ **to** $iteraciones$ **do**
        $(S1, S2) \leftarrow$ HeurísticaVoraz(S, num_swaps)
        $valor \leftarrow$ Evaluar($S1, S2$)
        if $valor < mejor\_valor$ **then**
            $mejor\_sol \leftarrow (S1, S2)$
            $mejor\_valor \leftarrow valor$
        end if
    end for
    return $mejor\_sol$
end procedure

5.3. Búsqueda Local

Dado el tipo de problema que se plantea resolver se platea un operador de vecindario de tipo **intercambio** o *swap*. Este tipo de operador trata de intercambiar vectores entre los subconjuntos definidos de S_1 y S_2. Dado este tipo de operador es posible definir una búsqueda local que vaya intercambiando elementos de S_1 a S_2 y viceversa, guardando en el camino solo las soluciones que superen el *score* actual.

5.4. Representación para algoritmos genéticos

El problema sería facilmente representable con un vector de *bits* c donde cada índice i representa la posición de un vector en el conjunto original S y donde $c_i = 0$ significaría que ese vector i está en el subconjunto S_1 y $c_i = 1$ que ese vector i está en el subconjunto S_2:

$$c_i = \begin{cases} 0, & \text{si el vector } i \text{ está en } S_1, \\ 1, & \text{si el vector } i \text{ está en } S_2. \end{cases}$$

Para el operador de cruce puede aplicarse un cruce de tipo "un punto" o *One-point crossover*, donde dados dos padres a, b se selecciona un punto de corte j y se intercambian las secciones de los cromosomas a partir del punto de corte para generar dos hijos 5:

$$\text{Hijo}_1 = [a_1, a_2, \ldots, a_j, b_{j+1}, b_{j+2}, \ldots, b_n],$$

$$\text{Hijo}_2 = [b_1, b_2, \ldots, b_j, a_{j+1}, a_{j+2}, \ldots, a_n],$$

En cuanto al operador de mutación, se proponela mutación *bit-flip*, donde con una probabilidad baja se invierte el valor de un gen en un cromosoma. Introduce pequeñas variaciones para escapar de óptimos locales. La probabilidad baja asegura que no se perturbe demasiado la estructura de soluciones prometedoras:

$$c_i' = \begin{cases} 1 - c_i, & \text{con probabilidad } p_m, \\ c_i, & \text{con probabilidad } 1 - p_m, \end{cases}$$

5.4.1. Inicialización

La literatura propone multitud de posibilidades a la hora de inicializar la población. Si se utilizase un vector con todo ceros como punto de partida, la exploración del espacio de soluciones podría verse limitada, ya que la búsqueda partiría de una configuración homogénea y potencialmente alejada de soluciones óptimas.

En problemas de optimización combinatoria, es común emplear heurísticas constructivas para generar soluciones iniciales con buena diversidad, por ello el uso de *Greedy*

Algorithm 4 Búsqueda Local

procedure BÚSQUEDALOCAL(S, $max\_iteraciones$)
  $(S1, S2) \leftarrow$ HEURÍSTICAVORAZ(S)
  $valor\_actual \leftarrow$ EVALUAR($S1, S2$)
  $iteracion \leftarrow 0$
  $mejorado \leftarrow$ True
  while $iteracion < max\_iteraciones$ **y** $mejorado$ **do**
    $mejorado \leftarrow$ False
    $mejor\_solucion \leftarrow (S1, S2)$
    $mejor\_valor \leftarrow valor\_actual$
    for cada i en $S1$ **do**
      $S1' \leftarrow S1$ sin el elemento i
      $S2' \leftarrow S2$ con el elemento i añadido
      if $S1'$ está vacío **then**
        continuar
      end if
      $valor\_nuevo \leftarrow$ EVALUAR($S1', S2'$)
      if $valor\_nuevo < mejor\_valor$ **then**
        $mejor\_solucion \leftarrow (S1', S2')$
        $mejor\_valor \leftarrow valor\_nuevo$
        $mejorado \leftarrow$ True
      end if
    end for
    for cada j en $S2$ **do**
      $S2' \leftarrow S2$ sin el elemento j
      $S1' \leftarrow S1$ con el elemento j añadido
      if $S2'$ está vacío **then**
        continuar
      end if
      $valor\_nuevo \leftarrow$ EVALUAR($S1', S2'$)
      if $valor\_nuevo < mejor\_valor$ **then**
        $mejor\_solucion \leftarrow (S1', S2')$
        $mejor\_valor \leftarrow valor\_nuevo$
        $mejorado \leftarrow$ True
      end if
    end for
    if $mejorado$ **then**
      $(S1, S2) \leftarrow mejor\_solucion$
      $valor\_actual \leftarrow mejor\_valor$
    end if
    $iteracion \leftarrow iteracion + 1$
  end while
  return $(S1, S2)$
end procedure

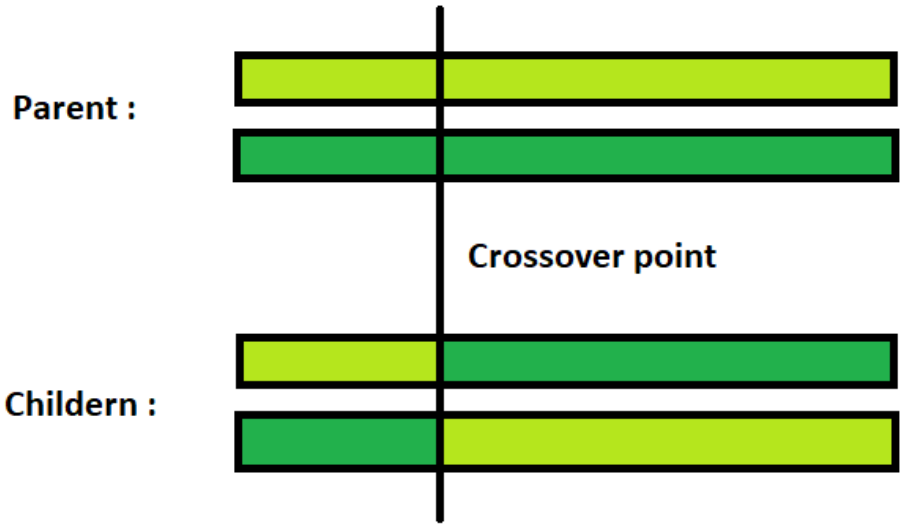

Figura 5: Operador de cruce de un punto para **GAs**.

o *Iterated Greedy* podrían ser buenas heurísticas iniciales para inicializar la solución de partida.

También es posible conformar una solución inicial de forma aleatoria, de forma que el algoritmo se beneficiaría más de un punto de partida con mayor posibilidad de exploración, ya que usar heurísticas como *Greedy* podrían afectar a la balanza de exploración y explotación, sesgando la solución haciendo que la población inicial caiga en una zona del espacio concreta.

Por ello se considera que quizá comenzar con un *Iterated Greedy* con un número de iteraciones pequeñas podría ser interesante, ya que este algoritmo incorpora una componente aleatoria (ligera exploración inicial) y además mejora la solución acercándola a zonas del espacio "prometedoras". Se le asignan un número de iteraciones bajo para no sesgar demasiado y romper el balance con la exploración y explotación.

6. Bibliografía

[1] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN: 0262631857.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd. Ann Arbor, MI: University of Michigan Press, 1975, second edition, 1992.

[3] A. Eiben y J. Smith, *Introduction to Evolutionary Computing* (Natural Computing Series), 2nd. Berlin, Heidelberg: Springer, 2015, pág. 30, S2CID 20912932, ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8.

[4] B. L. Miller, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," en,

[5] Z. C. Dagdia y M. Mirchev, "Chapter 15 - When Evolutionary Computing Meets Astro- and Geoinformatics," en *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, P. Škoda y F. Adam, eds., Elsevier, 2020, págs. 283-306, ISBN: 978-0-12-819154-5. DOI: 10.1016/B978-0-12-819154-5.00026-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128191545000266.

[6] Purdue University College of Engineering, *Lecture 4: Real-Coded Genetic Algorithms*, Lecture notes, Accessed on April 27, 2024. URL: https://engineering.purdue.edu/~sudhoff/ee630/Lecture04.pdf.

[7] S. Mirjalili y S. Mirjalili, "Genetic algorithm," *Evolutionary algorithms and neural networks: Theory and applications*, págs. 43-55, 2019.

[8] C. Garcia-Martinez, F. J. Rodriguez y M. Lozano, "Genetic Algorithms," en *Handbook of Heuristics*, R. Marti, P. M. Pardalos y M. G. C. Resende, eds. Cham: Springer International Publishing, 2018, págs. 431-464, ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_28. URL: https://doi.org/10.1007/978-3-319-07124-4_28.

[9] F. Herrera y M. Lozano, "Gradual distributed real-coded genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, n.º 1, págs. 43-63, 2000. DOI: 10.1109/4235.843494.

[10] E.-G. Talbi y V. Bachelet, "COSEARCH: A parallel cooperative metaheuristic," *J. Math. Model. Algorithms*, vol. 5, págs. 5-22, abr. de 2006. DOI: 10.1007/s10852-005-9029-7.

[11] P. Moscato y C. Cotta, "A Gentle Introduction to Memetic Algorithms," en ene. de 2003, vol. 57, págs. 105-144, ISBN: 1-4020-7263-5. DOI: 10.1007/0-306-48056-5_5.

[12] S. Kazarlis, S. Papadakis, J. Theocharis y V. Petridis, "Microgenetic algorithms as generalized hill-climbing operators for GA optimization," *IEEE Transactions on Evolutionary Computation*, vol. 5, n.º 3, págs. 204-217, 2001. DOI: 10.1109/4235.930311.

[13] M. Lozano y C. García-Martínez, "Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report," *Computers And Operations Research*, vol. 37, págs. 481-497, mar. de 2010. DOI: 10.1016/j.cor.2009.02.010.

[14] J. Kojić, "Integer linear programming model for multidimensional two-way number partitioning problem," *Computers And Mathematics with Applications*, vol. 60, n.º 8, págs. 2302-2308, 2010, ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2010.08.024. URL: https://www.sciencedirect.com/science/article/pii/S0898122110005882.

[15] P. C. Pop y O. Matei, "A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem," *Applied Mathematical Modelling*, vol. 37, n.º 22, págs. 9191-9202, 2013, ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2013.03.075. URL: https://www.sciencedirect.com/science/article/pii/S0307904X13002692.

[16] P. C. Pop y O. Matei, "A Genetic Algorithm Approach for the Multidimensional Two-Way Number Partitioning Problem," en *Learning and Intelligent Optimization*, G. Nicosia y P. Pardalos, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, págs. 81-86, ISBN: 978-3-642-44973-4.

[17] J. Kratica, J. Kojić y A. Savić, "Two metaheuristic approaches for solving multidimensional two-way number partitioning problem," *Computers And Operations Research*, vol. 46, págs. 59-68, 2014, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2014.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0305054814000045.

[18] F. J. Rodriguez, F. Glover, C. García-Martínez, R. Martí y M. Lozano, "GRASP with exterior path-relinking and restricted local search for the multidimensional two-way number partitioning problem," *Computers And Operations Research*, vol. 78, págs. 243-254, 2017, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2016.09.005. URL: https://www.sciencedirect.com/science/article/pii/S0305054816302209.

[19] V. Santucci, M. Baioletti y G. Di Bari, "An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem," *Expert Systems with Applications*, vol. 178, pág. 114 938, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114938. URL: https://www.sciencedirect.com/science/article/pii/S0957417421003791.