TRABAJO BLOQUE III (COMPETICIÓN)

Resumen:

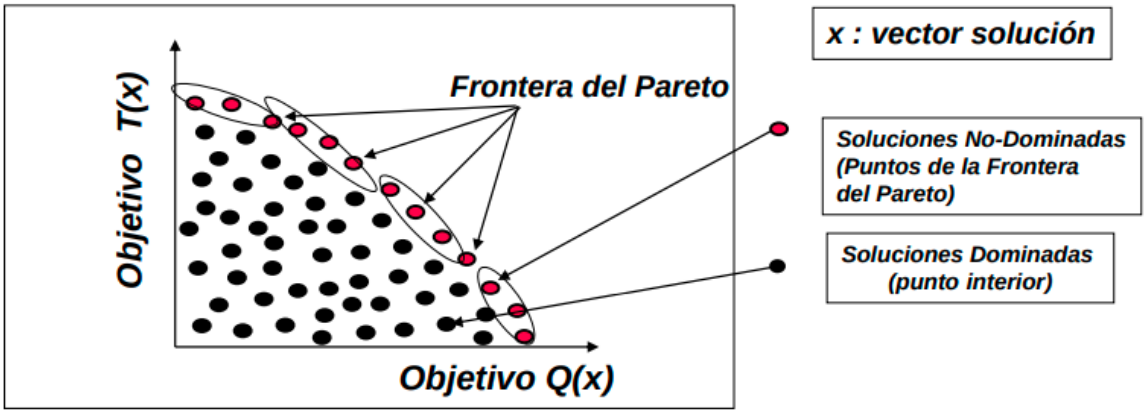
En este bloque de la asignatura tratamos técnicas de optimización a través de **algoritmos de Computación Evolutiva**, los cuales se engloban dentro del **Soft Computing,** rama de la Inteligencia Artificial (se especializa en poder resolver problemas que parten de funciones lineales o no lineales a través de métodos no convencionales).

En este trabajo trataremos **técnicas de optimización para problemas con varios objetivos: MO** (que pueden ser contradictorios, ejemplo: mayor rendimiento de un procesador vs procesador más barato) con y sin restricciones.

Para resolver estos problemas, **utilizaremos algoritmos multi-objetivos basados en agregación**, es decir, se agregan los objetivos (diferentes funciones) en una única función, dando lugar a una función adecuada para el algoritmo (nueva función mono-objetivo)

Los diferentes tipos de algoritmos dentro de la **Computación Evolutiva**, son buenas técnicas para resolver este tipo de problemas, debido a que son algoritmos poblacionales que permiten obtener múltiples soluciones en una única ejecución, pueden explorar distintas zonas del espacio de búsqueda simultáneamente y son más flexibles a la hora de encontrar soluciones óptimas (frentes de Pareto) cuando tienen una forma irregular (ej: frente cóncavo o desconectado)

Para entender mejor este concepto de **Pareto**, podemos apreciar en la imagen de abajo diferentes puntos que representan el espacio de soluciones de nuestro problema, cuando tratamos de maximizar estos objetivos (dos en este caso, T(x) y Q(x)), el frente de **pareto óptimo se encuentra en soluciones que no sean dominadas por ninguna otra.**



(un mismo problema puede tener diferentes soluciones pareto óptimas)

DESCRIPCIÓN DEL ALGORITMO USADO

Teniendo una función con múltiples objetivos (variables) a minimizar y un espacio de búsqueda representado por esas variables con sus universos de discurso, construiremos un algoritmo basado en agregación sin restricciones y otro con restricciones donde descomponemos el problema multi-objetivo en varios subproblemas mono-objetivo (los subproblemas representarán los diferentes individuos de nuestra población).

Usaremos un Algoritmo Evolutivo, rama de la **Computación Evolutiva** que utilizan transformaciones para combinar a los individuos a través de operaciones como selección, recombinación, competición e incluyendo, al igual que la genética en la naturaleza, una cierta aleatoriedad.
Típicamente estos algoritmos encuentran buenas soluciones para una gran diversidad de problemas, aunque no suele ser la más óptima (dependiendo del problema en sí)

Para nuestro algoritmo, **usaremos los operadores evolutivos de mutación y cruce del algoritmo de Evolución Diferencial,** donde, para cada individuo de la población, se realiza una operación de reproducción entre él mismo y una combinación de tres vectores (individuos) aleatorios dentro de un conjunto formado por los T-vecinos más cercanos:

- VECTOR MUTANTE = `VECTOR_1 + F * (VECTOR_2 - VECTOR_3)`
- VECTOR NUEVO : Probabilidad ½ para cada elemento entre el VECTOR MUTANTE y el ANTIGUO VECTOR

*Para asegurarme de que al menos se produce un cruce entre el vector mutante y el antiguo vector, forzamos siempre que un elemento aleatorio del vector mutante forme parte del nuevo vector, de forma aleatoria.

A cada individuo le corresponde un vector peso que identifica el subproblema a optimizar y que permanecerá hasta finalizar el algoritmo. Este vector peso está formado por dos componentes donde la suma de ambas hace la unidad, de esta forma, la población se distribuye uniformemente por el espacio vectorial bi-dimensional:

$$\boldsymbol{\lambda}^* = (\lambda_1^*, \cdots, \lambda_m^*)^T$$

A través de esto, podemos conformar **una lista para cada individuo que represente los T individuos más cercanos** cuyos objetivos tienen un parecido más cercano a otros que estén a mayor distancia.

Se necesita un **punto de referencia** con el fin de explorar el espacio de búsqueda para comparar las diferentes funciones objetivo (diferentes individuos), en nuestro caso, escogeremos los valores más óptimos obtenidos para minimizar cada objetivo del problema.

$$\mathbf{z}^* = (z_1^*, \cdots, z_m^*)^T \qquad z_i^* = \min\{f_i(\mathbf{x}) \qquad \mathbf{x} \in \Omega\}$$

Para medir (comparar) las diferentes soluciones obtenidas para las diferentes funciones objetivo, usaremos la formulación de Tchebycheff:

$$g^{te}(\mathbf{x}|\boldsymbol{\lambda},\mathbf{z}^*) = \max_{1 \le i \le m} \{\lambda_i | f_i(\mathbf{x}) - z_i^* |\}$$

- Para la primera iteración, **creamos un conjunto de N individuos inicializados de forma aleatoria y lo evaluamos mediante la función de evaluación** (en nuestro caso **ZDTE** y **CF6**), posteriormente inicializamos el punto de referencia.

- En las iteraciones posteriores: (hasta G: nº de generaciones)
    - Por cada individuo: (N individuos)

        1) Aplicamos operadores de evolución para el individuo dado.
        2) Evaluamos las prestaciones del individuo.
        3) Actualizamos el punto de referencia en caso de encontrar un valor más óptimo.
        4) Comprobamos para cada vecino del individuo, los valores obtenidos de este para su subproblema, en caso de que el vecino obtenga mejores resultados, sustituir los valores nuevos por los originales del vecino.

    ***Coste computacional global: N x G iteraciones.

Aplicaremos este algoritmo como he mencionado anteriormente, para resolver los problemas (funciones de evaluación) **ZDTE (30 dimensiones) y CF6 (4 y 16 dimensiones)** con un costo computacional de **10000 y 4000 iteraciones** y **comparar los resultados con el algoritmo NSGA II.**
(Dimensiones = Nº objetivos de nuestra función)

- **ZDTE**

Las funciones a minimizar son:

$$f_1(\boldsymbol{x}) = x_1$$

$$f_2(\boldsymbol{x}) = g(\boldsymbol{x}) \cdot h(f_1(\boldsymbol{x}), g(\boldsymbol{x}))$$

donde:

$$h(f_1(\boldsymbol{x}), g(\boldsymbol{x})) = 1 - \sqrt{f_1(\boldsymbol{x})/g(\boldsymbol{x})} - (f_1(\boldsymbol{x})/g(\boldsymbol{x}))\sin(10\pi f_1(\boldsymbol{x}))$$

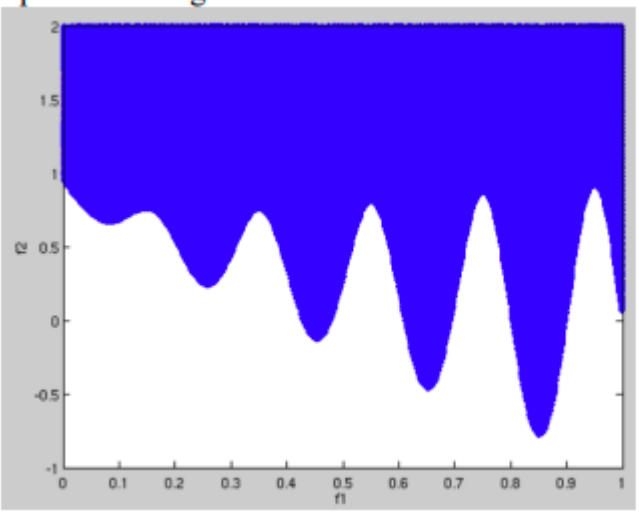
y

$$g(\boldsymbol{x}) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$

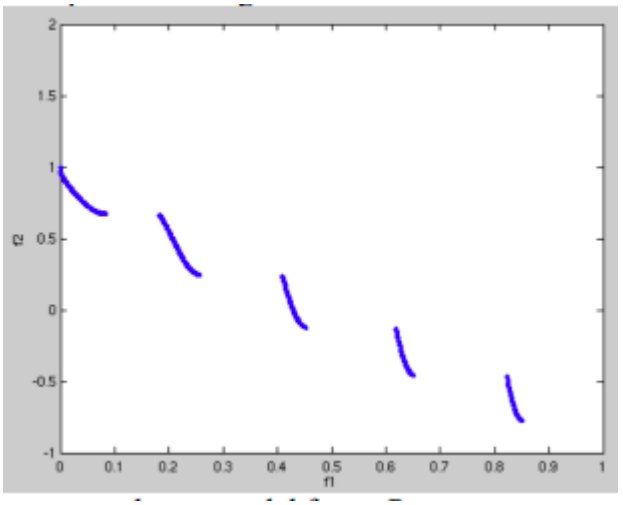
y el espacio de búsqueda es

$$0 \le x_i \le 1$$

La representación gráfica de dichas funciones es:



Representación del frente de Pareto Óptimo:



- **CF6**

Las funciones a minimizar son:

$$f_1(\boldsymbol{x}) = x_1 + \sum_{j \in J_1} y_j^2$$

$$f_2(\boldsymbol{x}) = (1 - x_1)^2 + \sum_{j \in J_2} y_j^2$$

donde:

$$J_1 = \{ j \mid j \text{ es impar y } 2 \le j \le n \}$$

$$J_2 = \{ j \mid j \text{ es par y } 2 \le j \le n \}$$

y

$$y_j = \begin{cases} x_j - 0.8 x_1 \cos(6\pi x_1 + \frac{j\pi}{n}) & \text{si } j \in J_1 \\ x_j - 0.8 x_1 \sin(6\pi x_1 + \frac{j\pi}{n}) & \text{si } j \in J_2 \end{cases}$$

Las restricciones son:

$$x_2 - 0.8 x_1 \sin(6\pi x_1 + \frac{2\pi}{n}) - \text{sgn}(0.5(1 - x_1) - (1 - x_1)^2) \sqrt{|0.5(1 - x_1) - (1 - x_1)^2|} \ge 0$$

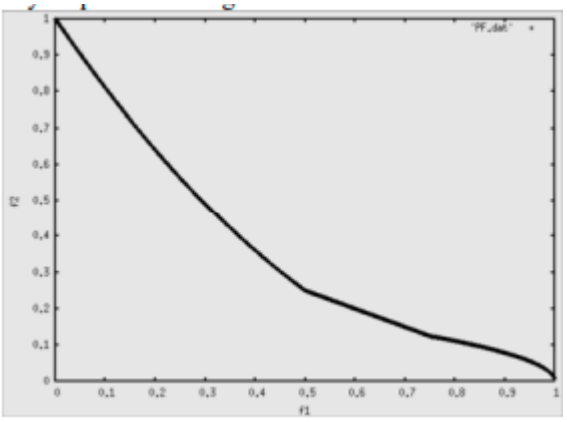
$$x_4 - 0.8 x_1 \sin(6\pi x_1 + \frac{4\pi}{n}) - \text{sgn}(0.25\sqrt{(1 - x_1)} - 0.5(1 - x_1)) \sqrt{|0.25\sqrt{1 - x_1} - 0.5(1 - x_1)|} \ge 0$$

El espacio de búsqueda es

$$0 \le x_1 \le 1$$

$$-2 \le x_i \le 2 \quad i > 1$$

Representación del frente de Pareto Óptimo:



Documentos entregados:

1) **ZDT3:**
    a) **4000 ITERACIONES**
    b) **10000 ITERACIONES**

2) **CF6_4D**
    a) **4000 ITERACIONES**
    b) **10000 ITERACIONES**

3) **CF6_16D**
    a) **4000 ITERACIONES**
    b) **10000 ITERACIONES**

4) **REPOSITORIO DE ALGORITMO IMPLEMENTADO CON ZDT3**

5) **REPOSITORIO DE ALGORITMO IMPLEMENTADO CON CF6**

Seguimiento de la documentación (seguir en este orden):

- **DOCUMENTO Nº 1:**
    - **ZDT3, para 4000 iteraciones con 10 ejecuciones con diferente número de población y generaciones.**
    - **ZDT3, para 10000 iteraciones con 4 ejecuciones con diferente número de población y generaciones.**

- **DOCUMENTO Nº 2: CF6, para 4000 iteraciones con 4 ejecuciones con diferente número de población y generaciones.**

- **DOCUMENTO Nº 3: ZDT3, para 4000 iteraciones con 10 ejecuciones con diferente número de población y generaciones.**

Se representarán los resultados de NSGA II frente al pareto óptimo, fichero dado en PF.dat (frente de color verde) y posteriormente, nuestro algoritmo (color azul) vs NSGA II (color morado)

Representación en azul → NSGA II pierde
Representación en morado → NSGA II gana
Representación en negro → Por decidir

Después de comparar la última generación de cada algoritmo, se procede a usar métricas para todas las generaciones (hipervolumen, spacing y coverage set) (las pruebas resaltadas en verde serán usadas para generar las métricas)

ZDT3 (30 DIMENSIONES)

- **Para 4000 iteraciones: individuos - generaciones**
    1) 80-50 (NSGA pierde)
    2) 48-82
    3) 100-40
    4) 40-100
    5) 124-32
    6) 32-125 (NSGA gana)
    7) 160-25
    8) 24-165 (por decidir)
    9) 200-20
    10) 20-200

- **Para 10000 iteraciones: individuos - generaciones**
    1) 100-100
    2) 200-80
    3) 80-200
    4) 248-40
    5) 40-250

Conclusiones:

Podemos apreciar con las diferentes pruebas realizadas que aproximadamente en **un 66% de los casos, el algoritmo realizado supera al NSGA II con el problema ZDT3**, en 10 de los 15 casos probados.

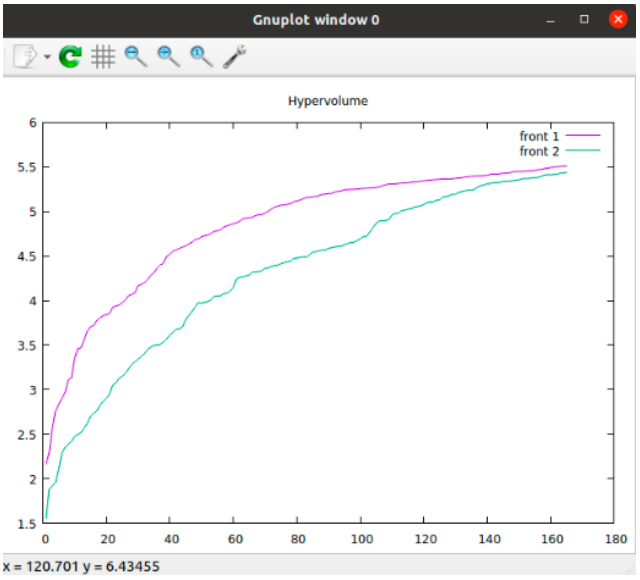
Se podrían seguir haciendo muchas pruebas, pero parece que **el algoritmo NSGA II mejora a mejor ritmo respecto a las generaciones y con menor nº de individuos que el algoritmo creado** (en las pruebas nº 6 y 10 para 4000 evaluaciones, NSGA II mejora respecto al algoritmo creado), **sin embargo**, **nuestro algoritmo funciona mucho mejor cuando el número de individuos aumenta**.

Para el caso 8 (archivo ZDT3_4000)con 4000 iteraciones y los casos 3 y 5 con 10000 iteraciones(archivo CF6_10000), ambos algoritmos compiten por quien refleja mejores resultados, ya que ambas nubes de puntos son similares.

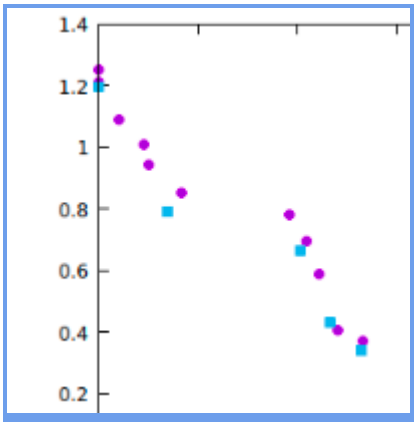
Debido a que las soluciones encontradas por NSGA II están repartidas con mayor uniformidad, en mi opinión es una solución más óptima.

Para el caso 8, archivo ZDT3_4000: 24 individuos para 165 generaciones:

- Podemos observar en primer lugar que el algoritmo NSGA II consigue explorar más a fondo el espacio de soluciones que nuestro algoritmo, es decir, se aproxima mejor al frente de pareto (más uniforme), obteniendo así un mayor hipervolumen, aunque como hemos mencionado anteriormente.



- Hablando de la distribución de las diferentes soluciones (individuos) obtenidas, también podemos observar que el algoritmo NSGA II está mayormente distribuido equitativamente por el espacio de soluciones que el algoritmo creado, sobretodo por las dos primeras componentes de lo que sería el frente ideal del problema ZDT3:



Podemos apreciar como efectivamente nuestro algoritmo creado (representado en azul) se distribuye de una forma menos igualitaria (mayor variación de distancias entre los diferentes individuos), esto queda reflejado en spacing donde para el algoritmo NSGA II obtiene un spacing menor a lo largo de todas las soluciones.

- En términos de decidir cuáles de los dos frentes obtenidos ofrece soluciones más óptimas, tenemos el Coverage Set, que calcula de un conjunto sobre otro, cuáles de sus soluciones son dominadas por las soluciones del otro conjunto, obteniendo así un recubrimiento de 1 sobre 2, y otro de 2 sobre 1

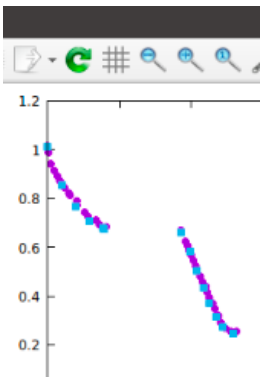
  Si observamos el plot del Coverage Set, **observamos en verde el porcentaje de dominancia de las soluciones de NSGA II respecto a nuestro algoritmo**, obteniéndose en la mayor parte de las generaciones (eje abscisas) ofrecidas por nuestro algoritmo (línea verde) un 1 en el valor de coordenadas, es decir, **en la mayor parte del total de generaciones, los individuos de cada generación en nuestro algoritmo (caso 8) están completamente dominados por los individuos de NSGA II,** la parte del eje abscisas de la **última generación (165)** indica aproximadamente que un 40% de los individuos de esa generación son dominados por el algoritmo NSGA II, y, aproximadamente un 60% no lo son, mientras que para el algoritmo NSGA II (línea morada), sólo en un pequeño porcentaje y en las primeras generaciones es cuando algunos individuos eran dominados por los de el algoritmo creado.

Podemos afirmar haciendo uso de las métricas, que, para este ejemplo, NSGA II supera a nuestro algoritmo creado.

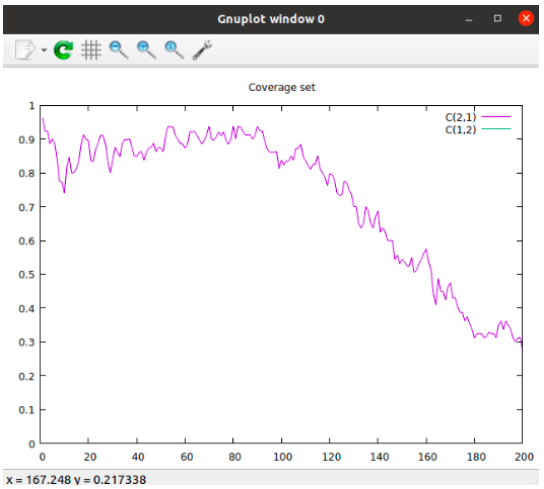
Para los casos 3 y 5 archivo ZDT3_10000 : (los casos donde las nubes de puntos son más próximas)

- **Caso 3:**

    - Hipervolumen y spacing muy similares, aunque se aprecia que, por muy poco, nuestro algoritmo cubre más a fondo el espacio de soluciones (mayor hipervolumen).
      Al igual que en el caso anterior en nuestro algoritmo, para las dos primeras componentes del frente de Pareto, la varianza entre los individuos es mayor, y, por lo tanto, NSGA II obtiene un menor spacing.



    - A diferencia del caso mostrado anteriormente, para este ejemplo, las soluciones de NSGA II respecto a nuestro algoritmo son dominadas por la mayor parte de individuos (reflejado en el Coverage Set, de un 70% en las primeras generaciones hasta un 30% en las últimas), mientras que las soluciones nuestro algoritmo respecto a NSGA II no son dominadas por ninguna solución, en ninguna generación.



    Podemos afirmar que en este caso, nuestro algoritmo funciona mejor que NSGA II.

- **Caso 5:**

    - Hipervolumen y Spacing muy cercanos, creo que es más acertado mirar directamente el Coverage Set, donde observamos que muy pocas soluciones por parte de NSGA II dominan a nuestro algoritmo (línea verde) a lo largo de las 250 generaciones.
    - Por otro lado, se observa cómo el algoritmo NSGA II va mejorando a lo largo de las generaciones reduciendo el porcentaje de individuos dominados por soluciones de nuestro algoritmo, hasta llegar a un 20% aproximadamente.
    -

CF6 (4 DIMENSIONES)

- **Para 4000 iteraciones: individuos - generaciones**
    1) 100-40
    2) 125-32
    3) 200-20
- **Para 10000 iteraciones: individuos - generaciones**
    4) 100-100
    5) 200-80
    6) 248-40

CF6 (16 DIMENSIONES)

- **Para 4000 iteraciones: individuos - generaciones**
    1) 100-40
    2) 125-32
    3) 200-20
- **Para 10000 iteraciones: individuos - generaciones**
    4) 100-100
    5) 200-80
    6) 248-40

Conclusiones:

Lo primero que cabe recordar, es que estamos hablando de algoritmos estocásticos donde en cada ejecución el algoritmo puede dar una solución diferente, ya sea buena o mala.

Para ZDT3 y CF6-4 dimensiones, la mayoría de las ejecuciones presentan resultados semejantes, sin embargo, para el problema CF6 con 16 dimensiones, si hay mayor variedad en cuanto a las soluciones dadas (para concretar, el algoritmo NSGA II presenta una mayor estabilidad a la hora ejecutar diferentes veces el algoritmo y, además de eso, presenta una mayor firmeza o distribución a la hora de buscar soluciones el frente a lo largo del espacio de búsqueda, cosa que nuestro algoritmo creado no consigue hacer)

Para 4 dimensiones, ambos algoritmos en la mayoría de sus ejecuciones tienen resultados similares, como he mencionado anteriormente.

Ambos algoritmos tienen en común que no son capaces de encontrar soluciones óptimas en el tramo de 0.6-0.7 a 1.0 por parte del objetivo de la abscisas (hablando del frente ideal), o encuentran muy pocas.

Sin embargo, la mayor diferencia entre nuestro algoritmo realizado y el NSGA II, es que mientras que el nuestro condensa todos los individuos entre 0.0 y 0.5 (hablando en referencia al eje de las abscisas), el algoritmo NSGA II si es capaz de encontrar soluciones más a lo largo del frente de Pareto (aunque no sean las más óptimas).

Esto es una clara ventaja del algoritmo NSGA II, debido a que tiene una mayor diversidad de soluciones que pueden llegar a cubrir de mejor forma el frente, también significa que con más generaciones podría encontrar este frente con mayor facilidad.

He intentado diferenciar para cada una de las pruebas realizadas, cuál de los dos algoritmos tiene mejor rendimiento:

4 DIMENSIONES:

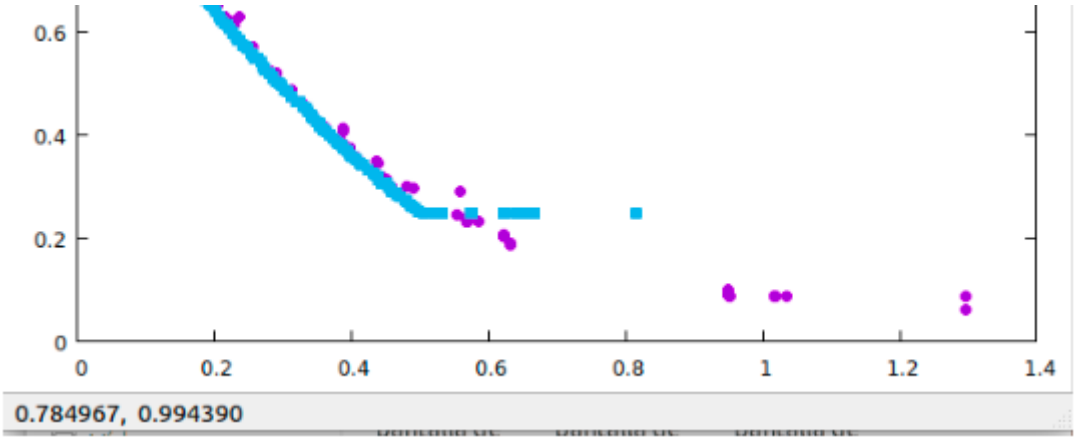
En términos generales para CF6 en 4 dimensiones, nuestro algoritmo funciona algo mejor, es capaz de encontrar algunos individuos que pertenezcan a esa parte “oscura” donde ambos algoritmos no son capaces de llegar (entre 0.6 y 1.0 en la parte de abscisas, cerca del frente ideal), probablemente debido a las restricciones del algoritmo.

En términos de hipervolumen ambos algoritmos tienen uno muy cercano (dependiendo de la ejecución)

A diferencia que en el problema ZDT3, para 4000 iteraciones, nuestro algoritmo suele tener una menor variación entre los diferentes individuos (menor Spacing)

A la hora de hablar de qué algoritmo encuentra soluciones más óptimas (Coverage Set), podemos apreciar que en los casos 1, 2 y 3, que para la gran mayoría de generaciones, una media de un 50% de los diferentes individuos de NSGA II son dominados por los de nuestro algoritmo hasta la última generación, que consiguen mismo porcentaje de dominancia, en torno a un 25%.

Esto en principio nos podría indicar que nuestro algoritmo encuentra mejores soluciones, sin embargo, he reflejado que para el caso 3, NSGA funciona de forma más óptima, debido a que ha conseguido encontrar soluciones en puntos del frente donde nuestro algoritmo no ha sido capaz de llegar y, esto, como mencioné anteriormente, podría converger en las próximas generaciones a encontrar más puntos (soluciones) en ese dominio.



16 DIMENSIONES:

Para el problema CF6 con 16 dimensiones, observamos más fácilmente las diferencias (ventajas y desventajas) de los dos algoritmos que estamos evaluando.

Mientras que para el algoritmo NSGA II existe una gran diversidad de individuos que exploran más o menos uniformemente el espacio de búsqueda, nuestro algoritmo pierde diversidad y se centra en explorar el espacio de soluciones que encuentra más óptimo, perdiendo individuos en otras zonas del frente donde explorar.

Por otro lado, nuestro algoritmo tiene soluciones que no son dominadas por ninguna solución, se centra más en explotar las soluciones buenas.

Dependiendo del tipo de problema o del tipo de solución que necesites, ambos algoritmos podrían ser válidos aunque yo me quedaría con el NSGA II, que no pierde tanta diversidad y con más generaciones podría encontrar un mejor frente con más seguridad.

FIN DEL TRABAJO