Aplicaciones de Soft Computing

Implementación de un algoritmo multiobjetivo basado en agregación

Escuela Técnica Superior de Ingeniería Informática

Trabajo realizado por Antonio Cárdenas Luque

Datos de contacto: antcarluq@alum.us.es | Fecha de publicación: 06/09/2020

Índice

Resumen

El trabajo que en esta memoria se describe consiste en la implementación de un algoritmo multiobjetivo basado en agregación. Su eficiencia al evaluar los problemas ZDT3 y CF6 pretende mejorar los resultados obtenidos por el algoritmo NSGAII en un entorno de comparación justo.

Por lo tanto en esta memoria, se explicarán los detalles de la implementación, sin entrar en mostrar código, argumentando las diferentes elecciones de implementación tomadas en el camino.

También, se incluye una pequeña guía para el uso del software y los directorios que componen el proyecto.

Por último, se realizará una extensa comparativa de ambos algoritmos utilizando las herramientas métricas de la asignatura, así como una conclusión en base a los resultados obtenidos.

Implementación

Solución Tecnológica

La solución tecnológica seleccionada para llevar a cabo la implementación del algoritmo ha sido la de utilizar Python ya que es un lenguaje de programación multiparadigma que soporta tanto orientación a objetos como programación funcional.

Además, posee una serie de librerías que han sido útiles y necesarias en el desarrollo del algoritmo y en la visualización de sus resultados. Entre ellas: NumPy, para la gestión de matrices y vectores, Random para la selección de números aleatorios, Math, como herramienta matemática para el cálculo, y Matplotlib, para la representación y visualización de los resultados.

Todo ello en el entorno de desarrollo *PyCharm*, que facilita la programación en entornos virtuales y es uno de los IDE más populares para Python.

Por último, cabe destacar que todo el código se ha versionado y almacenado en el siguiente repositorio de GitHub:

https://github.com/antcarluq/multi-objective-algorithm-ASC

Detalles de Implementación

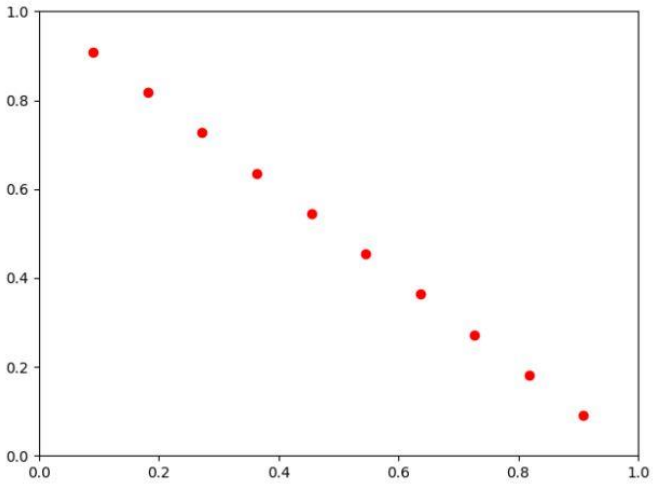
La primera decisión que se tomó respecto a la implementación, debido a la complejidad que albergaba, fue la de subdividir el problema en conjunto en subproblemas (Como si de un algoritmo multiobjetivo basado en agregación se tratase) e ir resolviendo estos problemas pequeños paso a paso.

INICIALIZACIÓN

Con esta filosofía, el primer problema abordado fue el de realizar un método para inicializar una distribución uniforme de n vectores cuyas componentes sumaran la unidad.

La clave para encontrar la solución a este problema fue la de comprender que si se necesitaba distribuir uniformemente n puntos, el espacio que los debía separar tenía que ser *1 / (n + 1).*

Para asegurar que las componentes de este vector sumaran la unidad simplemente se tenía que calcular cada componente *x* con el método anterior y la componente *y* seria en cada caso *yi = 1 – xi.*



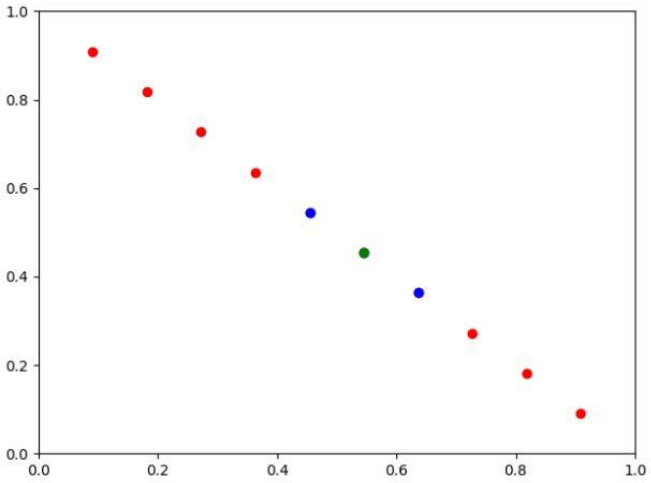
Vectores con una distribución uniforme para n = 10

Conforme se iba realizando el algoritmo se hizo patente la necesidad que había de almacenar información acerca del estado de los subproblemas y las soluciones. Para solventar esto se ideó una clase “subproblema” cuya configuración fue variando en base a las necesidades y cuya estructura finalmente fue la siguiente:

```
Clase Subproblem {
        x -> Componente x
        y -> Componente y
        neighbours -> Subproblemas vecinos
        individual { -> Individuo asociado al subproblema
                gen -> Secuencia de n números que componen el gen
                solution -> Solución asociada al gen
        }
}
```

Configuración de la clase Subproblema

El siguiente paso fue el de idear un método para encontrar los T subproblemas más cercanos a un subproblema (incluido este). Para ello se ideó una función, para: por cada subproblema, obtener en una lista los demás subproblemas y la distancia euclídea al subproblema estudiado. Una vez que se obtenía dicha lista se ordenaba de menor a mayor por distancia y se seleccionaban los T primeros ítems.



Vecinos (azul) para un subproblema (verde) con T = 3

A continuación había que generar una población inicial aleatoria de N individuos con un gen asociado de X dimensión, en función del problema que se estuviese tratando. Además los números aleatorios (cromosomas) que componían el gen de cada individuo debían respetar el espacio de búsqueda asociado a cada problema.

En este paso se tuvo en cuenta la recomendación de incluir una semilla con el objetivo de que esta población inicial fuera la misma en las distintas pruebas que se realizaran del algoritmo y que la valoración de la eficiencia pudiera ser más fidedigna.

Una vez generada la población inicial, esta pasaba a ser evaluada con el objetivo de obtener las primeras soluciones asociadas a los individuos y de inicializar el punto de referencia como el mínimo valor para cada componente de las evaluaciones. Este punto de referencia en cada iteración del algoritmo volvería a ser revisado y actualizado, dado el caso, siguiendo el mismo criterio.

ITERACIÓN

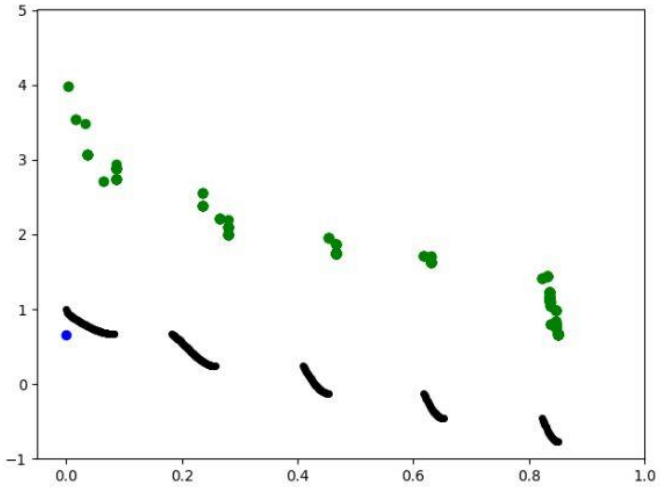
Una vez finalizada la fase de inicialización del algoritmo lo siguientes pasos deberían repetirse iterativamente por cada subproblema durante G generaciones, es decir el numero total de iteraciones serian G x N.

En primer lugar se debía generar un nuevo individuo en base a los vecinos del subproblema actual. Es aquí donde había que idear operadores evolutivos y es aquí en uno de los puntos del algoritmo donde mayor libertad de implementación había.

En consecuencia de ello en esta parte de la implementación se probaron distintos tipos de operadores buscando en ocasiones mayor aleatoriedad en los genes para poder escapar de una búsqueda local, o en otras ocasiones todo lo contrario, buscando que los genes mutaran poco y poder centrarse más en una zona.

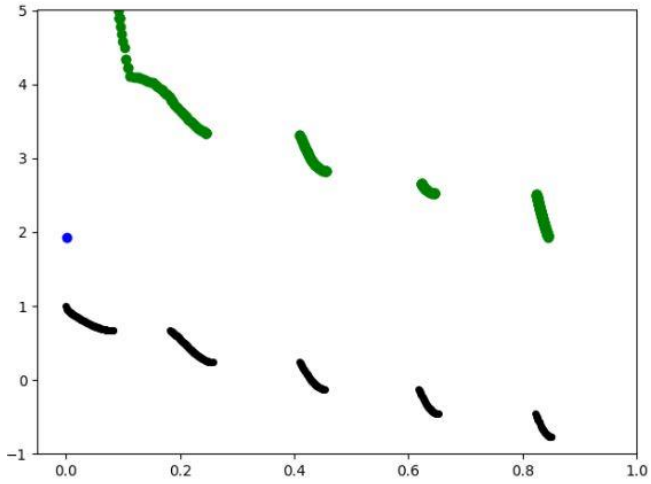
A continuación se explican algunos de los operadores utilizados y sus resultados:

Operador evolutivo 1: Este fue el primer operador que se probó y básicamente consistía en mantener el gen actual con una probabilidad del 50% de que cualquier cromosoma mutara a un número aleatorio (dentro del espacio de búsqueda).



Resultado del operador evolutivo 1 para ZDT3 con G = 20 y N = 500

Operador evolutivo 2: El siguiente operador que se probó generaba un gen a partir de la media de varios vecinos aleatorios del subproblema actual.

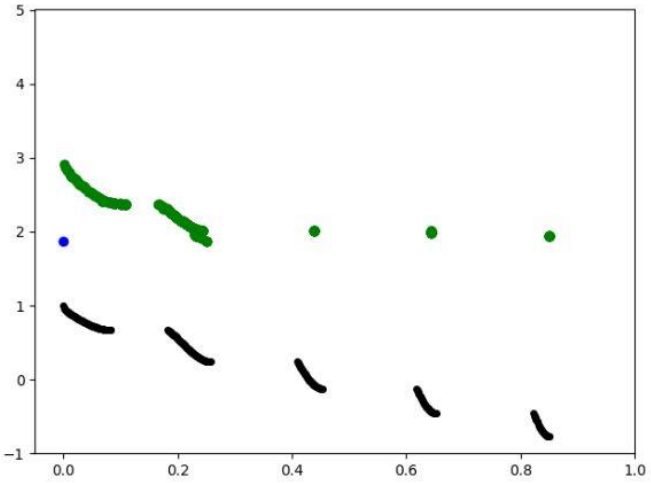


Resultado del operador evolutivo 2 para ZDT3 con G = 20 y N = 500

Operador evolutivo 3: Viendo que los resultados no eran los esperados probé otro tipo de combinación de vectores entre vecinos en base a recomendaciones de la asignatura. Los resultados mejoraban pero las soluciones no terminaban de aproximar bien.

$$v^{(i)}(G+1) = x^{(r1)}(G) + F \cdot (x^{(r2)}(G) - x^{(r3)}(G))$$

Combinación de vectores recomendada siendo F = 0.5

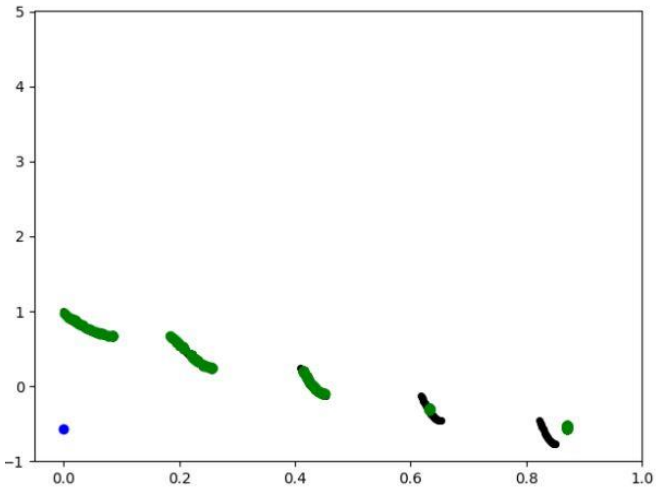


Resultado del operador evolutivo 3 para ZDT3 con G = 20 y N = 500

Operador evolutivo 4: Analizando los resultados de los 3 primeros operadores surgió la idea de que el problema del primer operador quizás era dar demasiado peso a la aleatoriedad y que la idea de que algunos cromosomas se mantuviesen no era mala. En base a ello se ideó el cuarto operador evolutivo, que generaba un gen cuyos

cromosomas tenían un 50% de probabilidades de mutar utilizando la combinación vectorial de vecinos del operador 3.

Como se aprecia en la siguiente imagen los resultados mejoraron mucho y ya solo quedaba intentar perfeccionar y perfilar el operador.

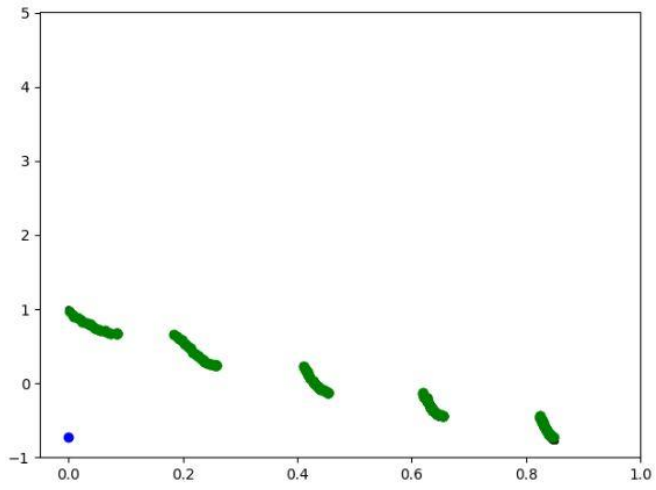


Resultado del operador evolutivo 4 para ZDT3 con G = 20 y N = 500

Operador evolutivo 5: El ultimo operador surgió de la idea de pensar que quizás en las primeras generaciones el operador debía tener unas características distintas de las ultimas.

Por ello en el primer cuarto de las generaciones el operador consistía en una combinación del operador 3 con posibilidad de tener cromosomas aleatorios, buscando que los nuevos genes sean más distintos y se puedan distribuir bien de primera.

Y en el resto de las iteraciones, el operador funciona como el operador 4 para que los genes no muten tanto y la búsqueda sea más local una vez que ya están perfiladas las soluciones.



Resultado del operador evolutivo 5 para ZDT3 con G = 20 y N = 500

Finalmente este ultimo ha sido el operador que mejores resultados ha dado y no se ha conseguido mejorar. Cabe destacar que las capturas mostradas solo son algunos de los resultados que estos han dado y que se han probado estos operadores en distintas ejecuciones con distintas semillas.

Una vez habíamos generado el nuevo individuo debíamos evaluarlo y obtener una solución que mejorase a las actuales. En función del problema que estábamos abordando debíamos utilizar la fórmula pertinente a ZDT3 o a CF6.

$$\text{Minimize} = \begin{cases} f_1(\boldsymbol{x}) = x_1 \\ f_2(\boldsymbol{x}) = g(\boldsymbol{x})\, h(f_1(\boldsymbol{x}), g(\boldsymbol{x})) \\ g(\boldsymbol{x}) = 1 + \frac{9}{29}\sum_{i=2}^{30} x_i \\ h(f_1(\boldsymbol{x}), g(\boldsymbol{x})) = 1 - \sqrt{\frac{f_1(\boldsymbol{x})}{g(\boldsymbol{x})}} - \left(\frac{f_1(\boldsymbol{x})}{g(\boldsymbol{x})}\right)\sin(10\pi f_1(\boldsymbol{x})) \end{cases}$$

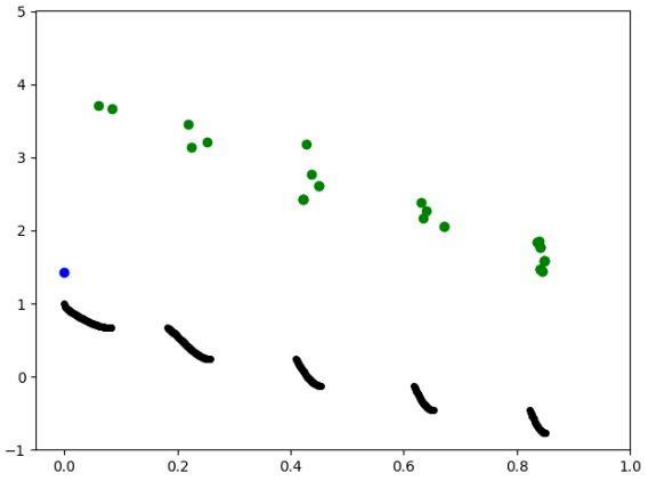
Problema de minimización de función multiobjetivo ZDT3

Cuando ya teníamos la nueva solución debíamos comprobar si esta era mejor que la actual para el subproblema estudiado y también para sus subproblemas vecinos ya que por proximidad también podría ser una buena solución. En este caso, llevándonos por la intuición y la lógica, se realizó un mecanismo de selección basado en la distancia euclídea de las soluciones con el subproblema y el punto de referencia. En ningún caso este mecanismo llegó a funcionar y finalmente utilicé el mecanismo de selección recomendado.

Matemáticamente: para cada $j \in B(i)$, si $g^{te}(y|\lambda^j, z) \le g^{te}(x^j|\lambda^j, z)$ entonces $x^j = y$

En esta parte de la implementación cabe señalar que se cometió el mayor fallo en este proyecto y que llevó bastante tiempo localizar. Una vez el mecanismo de selección comprobaba que estábamos estudiando una mejor solución para el subproblema, únicamente se actualizaba dicha solución y no el gen asociado a esta.

De esta manera, al aplicar los operadores evolutivos, se estaban generando nuevos individuos a partir de genes que no estaban siendo actualizados junto con las mejores soluciones. Por ello, de ninguna manera, estos genes evolucionaban a dar mejores posibles soluciones. El comportamiento que me llevó a descubrir este fallo es que el algoritmo si parecía que aproximaba bien las primeras generaciones, pero pasadas estas, ni en un millón de iteraciones, se llegaban a evaluar mejores soluciones.



Comportamiento resultante del fallo descrito tras 1.000.000 de evaluaciones

Una vez que localicé este fallo el algoritmo ya aproximaba bien y pude seguir probando distintas configuraciones e ir perfilándolo, utilizando los distintos operadores evolutivos que hemos explicado antes.

USO DEL ALGORITMO

Por último, para facilitar el uso del algoritmo se va a explicar la estructura del proyecto y los pasos a seguir para poder ejecutar el software.

Antes de nada, cabe señalar que el algoritmo requiere de algunas librerías, como ya se ha comentado antes. Podrá instalarlas fácilmente con el siguiente comando: *pip install -r requirements.txt.*

En cualquier caso si al ejecutar algún fichero aparece un error porque falte instalar algún módulo en su equipo, añádalo al *requirements.txt* y vuelva a ejecutar el comando.

Una vez solventado esto, si desea probar en casos concretos el algoritmo, únicamente deberá ejecutar el siguiente comando en su terminal: *python3 main.py.* Se establecerá un diálogo en el que tendrá que configurar los parámetros iniciales, y al finalizar tendrá los datos resultantes en un fichero personalizado en el mismo directorio que *main.py*

Por otro lado si quiere ejecutar todas las evaluaciones que se han descrito en esta memoria, deberá ejecutar el archivo *evaluations.py* y en la carpeta *results* se guardarán automáticamente los resultados de forma ordenada.

Una vez ha realizado esto, si desea comparar nuestro algoritmo con el algoritmo NSGAII deberá ejecutar el fichero *metrics.py*. Haciendo uso del software métrico y de la generación propia de ficheros de configuración para cada evaluación, obtendrá en el directorio *results/comparations* los resultados ordenados.

Por último, si quiere obtener un resumen de estos datos, deberá ejecutar *data_processing.py* y se generar un resumen de la evaluación en el directorio *results/comparations/resume.txt*

Resultados

Primeros Pasos

Con el objetivo de facilitar la generación de múltiples evaluaciones del algoritmo desarrollado y la comparación con NGSAII se han desarrollado a parte de la implementación del algoritmo dos scripts.

El primer script evalúa y archiva en directorios ordenados los resultados de las ejecuciones para los tres problemas estudiados: ZDT3, CF64D y CF616D. Mediante los parámetros de entrada podemos indicarle distintas configuraciones del número de población y del número de generaciones iniciales así como cuantas semillas quiere probar.

En el segundo script, se crean archivos .in con los datos de entrada, para, haciendo uso del software para realizar métricas dado en la asignatura, poder comparar el algoritmo que se ha implementado con el algoritmo NSGAII en igualdad de condiciones, es decir con la misma configuración inicial. Una vez se han creado estos archivos, se ejecuta el software métrico en cascada y se guardan los resultados en directorios ordenados.

Métricas

Con el objetivo de poder comparar y mostrar con claridad los resultados de ambos algoritmos se van a exponer dos tipos de pruebas para cada uno de los problemas abordados.

En primer lugar se va a hacer una comparativa visual de los frentes obtenidos en una semilla común, alternando entre distintas configuraciones de población y generaciones, siempre con un número de evaluaciones total igual a 4.000 o 10.000.

Los puntos verdes marcarán el frente obtenido por el algoritmo implementado, los puntos rojos el frente relativo a la ejecución de NSGAII y los puntos negros señalan el frente ideal.

En segundo lugar se realizará un análisis numérico con las tres métricas dadas en la asignatura: El área del hipervolumen, el *spacing* y la cobertura de un frente respecto a otro.

Respecto al área del hipervolumen del frente, una cantidad mayor o menor no marca necesariamente que resultados son mejores. Se puede dar el caso de que el

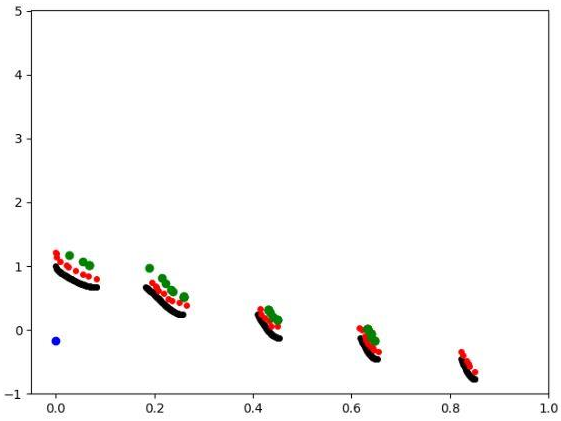
hipervolumen sea pequeño pero los resultados no sean mejores porque las soluciones no estén bien distribuidas.

Justamente para comprobar si las soluciones están distribuidas uniformemente sirve la métrica del *spacing*, un valor mas pequeño en este caso si indica que las soluciones están distribuidas uniformemente.

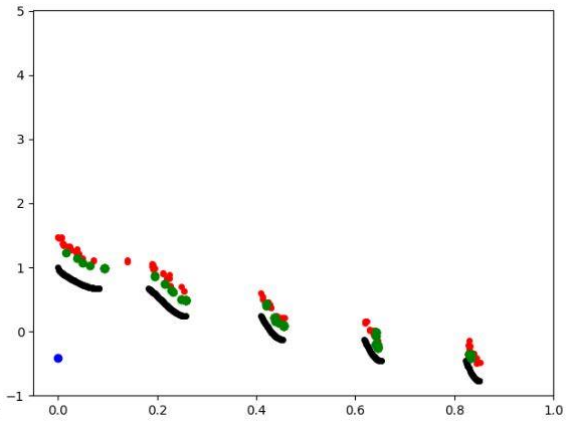
Por ultimo la cobertura nos indica que porcentaje de soluciones de un frente dominan a las de otro frente. Cuantas más soluciones dominen al otro frente esto indicará que los resultados son mejores ya que se ha aproximado más al frente ideal.

Cabe señalar que en este estudio métrico los valores que se van a mostrar en las siguientes tablas son el valor medio de los resultados para 10 semillas distintas con el objetivo de resultados fidedignos y de presentar los datos mas recogidos y que sea más fácil asimilarlos.

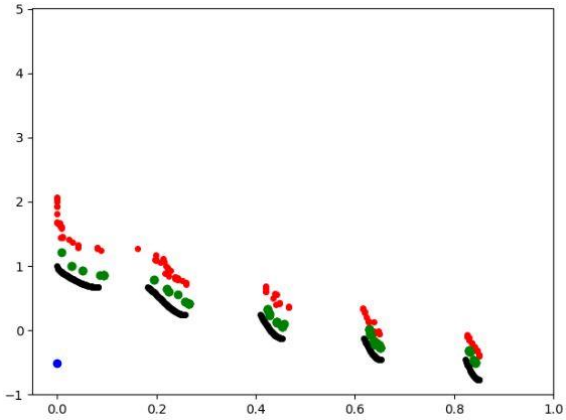
EVALUACIÓN ZDT3



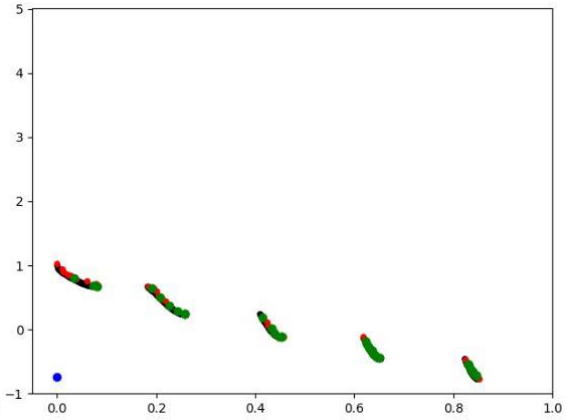
Frentes tras 4.000 ev con P=40 y G=100



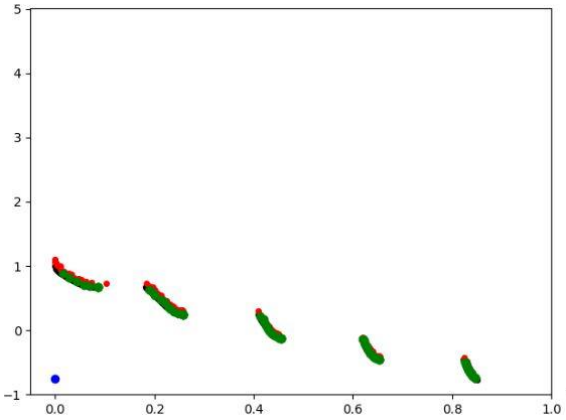
Frentes tras 4.000 ev con P=80 y G=50



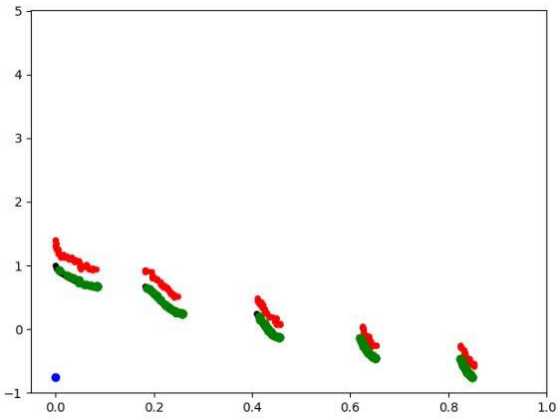
Frentes tras 4.000 ev con P=100 y G=40



Frentes tras 10.000 ev con P=40 y G=250



Frentes tras 10.000 ev con P=100 y G=100



Frentes tras 10.000 ev con P=200 y G=50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**0.883563**	**0.781365**
Spacing	**0.012071**	**0.050490**
Cobertura	**71.28%**	**9.25%**

Resultados tras 4.000 evaluaciones con P = 40 y G = 100

Métricas	*NGSAII*	*ALGORITMO*
Hipervolumen	***1.016713***	***1.165328***
Spacing	***0.009881***	***0.027489***
Cobertura	***3.44%***	***75.24%***

Resultados tras 4.000 evaluaciones con P = 80 y G = 50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**1.135301**	**1.326581**
Spacing	**0.008174**	**0.022676**
Cobertura	**1.44%**	**87.26%**

Resultados tras 4.000 evaluaciones con P = 100 y G = 40

Métricas	NGSAII	ALGORITMO
Hipervolumen	**0.787969**	**0.784455**
Spacing	**0.013168**	**0.030005**
Cobertura	**13.69%**	**14.5%**

Resultados tras 10.000 evaluaciones con P = 40 y G = 250

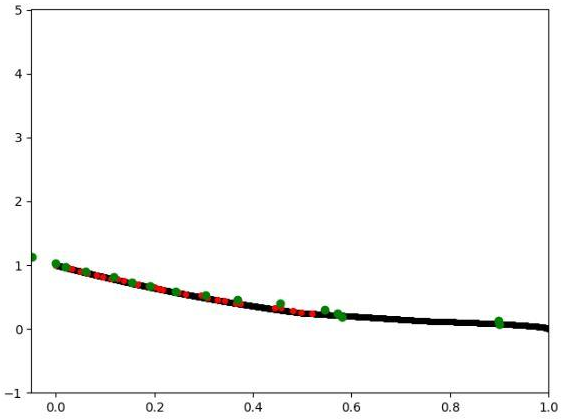
Métricas	NGSAII	ALGORITMO
Hipervolumen	**0.829659**	**0.861048**
Spacing	**0.004888**	**0.013722**
Cobertura	**1.65%**	**70%**

Resultados tras 10.000 evaluaciones con P = 100 y G = 100

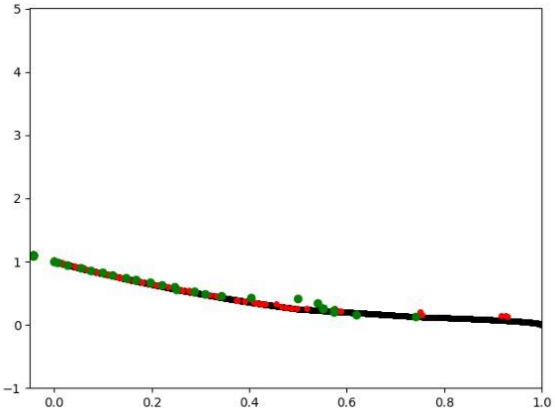
Métricas	NGSAII	ALGORITMO
Hipervolumen	**1.086828**	**1.252945**
Spacing	**0.004105**	**0.008190**
Cobertura	**0%**	**90.72%**

Resultados tras 10.000 evaluaciones con P = 200 y G = 50

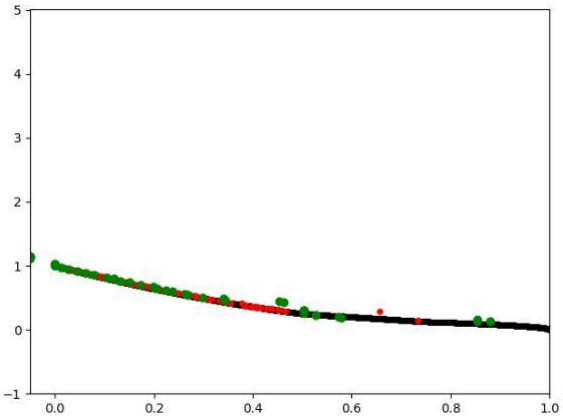
EVALUACIÓN CF6 4D



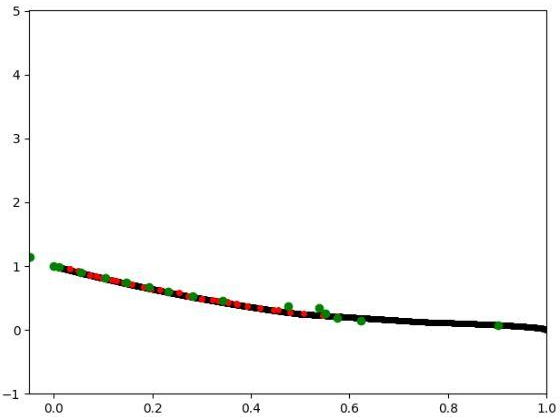
Frentes tras 4.000 ev con P=40 y G=100



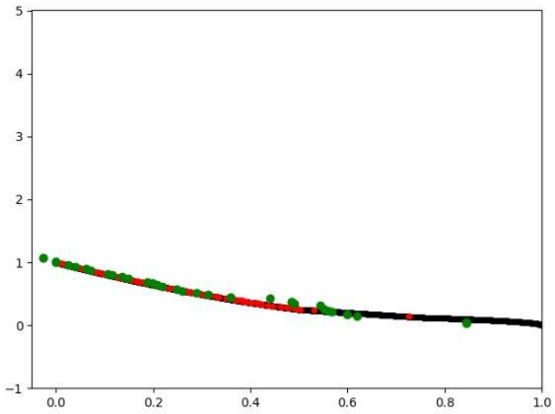
Frentes tras 4.000 ev con P=80 y G=50



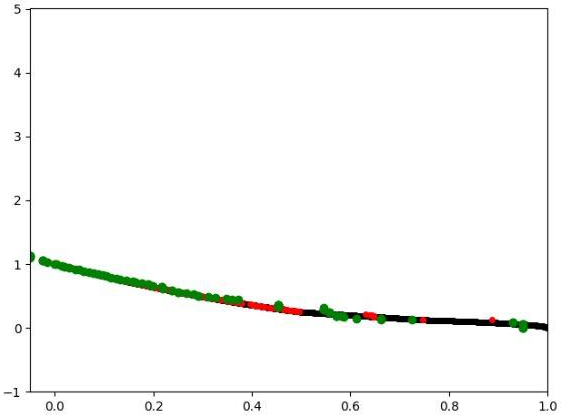
Frentes tras 4.000 ev con P=100 y G=40



Frentes tras 10.000 ev con P=40 y G=250



Frentes tras 10.000 ev con P=100 y G=100



Frentes tras 10.000 ev con P=200 y G=50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**7.486818**	**13.92693**
Spacing	**0.020894**	**0.074001**
Cobertura	**17.80%**	**8.5%**

Resultados tras 4.000 evaluaciones con P = 40 y G = 100

Métricas	NGSAII	ALGORITMO
Hipervolumen	**8.953955**	**16.63162**
Spacing	**0.016910**	**0.037441**
Cobertura	**25.13%**	**7.45%**

Resultados tras 4.000 evaluaciones con P = 80 y G = 50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**7.763240**	**13.98031**
Spacing	**0.018619**	**0.032233**
Cobertura	**27.11%**	**6.28%**

Resultados tras 4.000 evaluaciones con P = 100 y G =40

Métricas	NGSAII	ALGORITMO
Hipervolumen	**7.751714**	**15.05951**
Spacing	**0.027765**	**0.056956**
Cobertura	**13.91%**	**7.19%**

Resultados tras 10.000 evaluaciones con P = 40 y G = 250

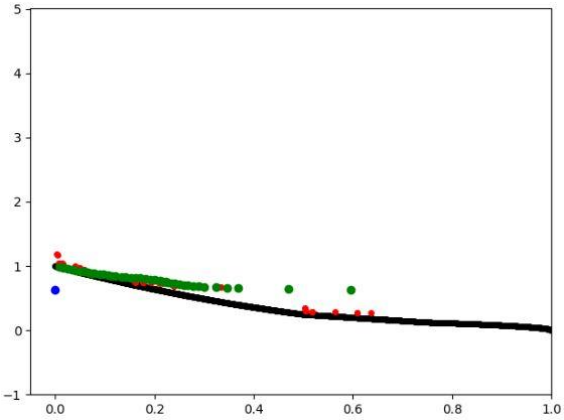
Métricas	NGSAII	ALGORITMO
Hipervolumen	**7.177890**	**14.26566**
Spacing	**0.011769**	**0.028654**
Cobertura	**22.98%**	**6.49%**

Resultados tras 10.000 evaluaciones con P = 100 y G = 100

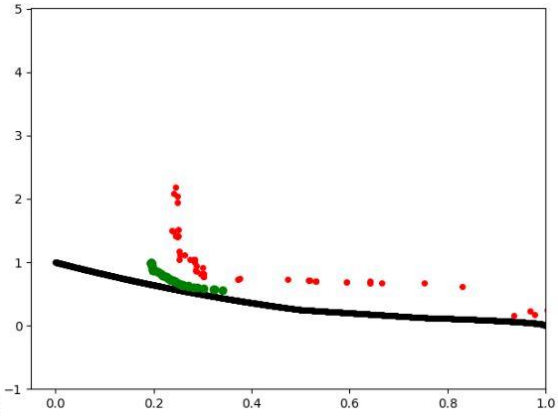
Métricas	NGSAII	ALGORITMO
Hipervolumen	**8.245598**	**16.44423**
Spacing	**0.015118**	**0.016684**
Cobertura	**32.84%**	**6.31%**

Resultados tras 10.000 evaluaciones con P = 200 y G = 50

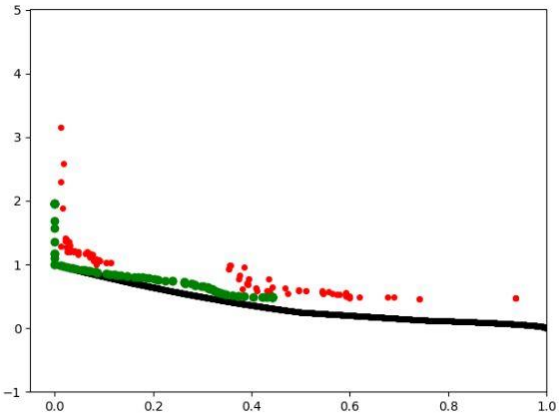
EVALUACIÓN CF6 16D



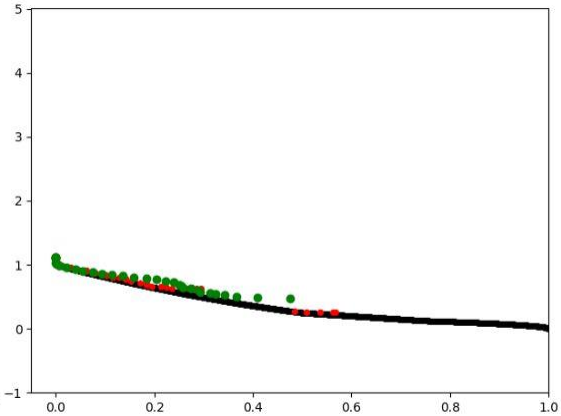
Frentes tras 4000 ev con P=40 y G=100



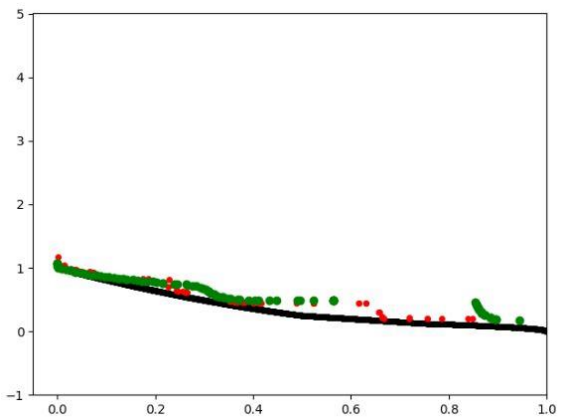
Frentes tras 4000 ev con P=80 y G=50



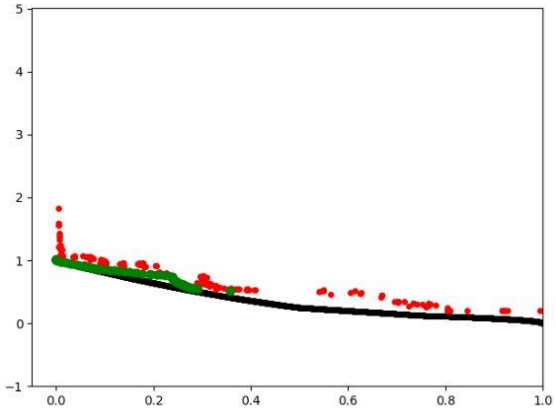
Frentes tras 4000 ev con P=100 y G=40



Frentes tras 10000 ev con P=40 y G=250



Frentes tras 10000 ev con P=100 y G=100



Frentes tras 10000 ev con P=200 y G=50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**1.472201**	**1.436068**
Spacing	**0.039393**	**0.043149**
Cobertura	**24.60%**	**25.42%**

Resultados tras 4.000 evaluaciones con P = 40 y G = 100

Métricas	NGSAII	ALGORITMO
Hipervolumen	**4.508317**	**4.574337**
Spacing	**0.030248**	**0.061541**
Cobertura	**6.98%**	**60%**

Resultados tras 4.000 evaluaciones con P = 80 y G = 50

Métricas	NGSAII	ALGORITMO
Hipervolumen	**6.828795**	**7.602516**
Spacing	**0.026384**	**0.017519**
Cobertura	**3.85%**	**89.35%**

Resultados tras 4.000 evaluaciones con P = 100 y G = 40

Métricas	NGSAII	ALGORITMO
Hipervolumen	**0.956356**	**1.014334**
Spacing	**0.041828**	**0.028827**
Cobertura	**39.39%**	**16.45%**

Resultados tras 10.000 evaluaciones con P = 40 y G = 250

Métricas	NGSAII	ALGORITMO
Hipervolumen	**1.844442**	**2.034692**
Spacing	**0.023398**	**0.020089**
Cobertura	**36.06%**	**39.03%**

Resultados tras 10.000 evaluaciones con P = 100 y G = 100

Métricas	NGSAII	ALGORITMO
Hipervolumen	**7.903303**	**8.084432**
Spacing	**0.011488**	**0.017271**
Cobertura	**10.27%**	**71.79%**

Resultados tras 10.000 evaluaciones con P = 200 y G = 50

Conclusión

Una vez expuesto los resultados de la comparación entre el algoritmo implementado y el algoritmo NSGAII podemos sacar las siguientes conclusiones:

En primer lugar, para el problema ZDT3 podemos ver mejores resultados de nuestro algoritmo sobre todo en aquellos casos en los que disponemos de menos generaciones. El comportamiento de ambos se va igualando cuantas mas generaciones incluyamos y es mejor, para NSGAII, en el caso especifico de que dispongamos de un buen número de generaciones y pocos subproblemas.

En segundo lugar, para el problema CF6 de 4 dimensiones podemos ver que ambos algoritmos aproximan medianamente bien pero fallan un poco en la distribución de las soluciones, en este caso la balanza se decanta por el algoritmo NGSAII.

Por último, es en el problema CF6 de 16 dimensiones donde ambos algoritmos dan peores resultados, en algunos casos funciona mejor nuestro algoritmo y en otros NSGAII a la vista de los resultados.

En conclusión, analizando los resultados de la comparativa entre los dos algoritmos, en el tanteo nuestro algoritmo obtiene, por poco, mejores resultados, pero lo mas justo seria dar un empate técnico, ya que ninguno destaca especialmente sobre el otro y ambos acaban ofreciendo por lo general buenos resultados.

Anexo

Repositorio donde se ubica el código implementado:

https://github.com/antcarluq/multi-objective-algorithm-ASC