ALGORITMO MULTIOBJETIVO BASADO EN AGREGACIÓN

Implementación por

Antonio Vázquez Pérez

Sumario

Introducción

Como forma de evaluación de un bloque de la asignatura *Aplicaciones de Soft Computing* se ha propuesto la realización de este documento, el cual recogerá la implementación de un algoritmo genético que tratará de optimizar múltiples objetivos utilizando una agregación de distintos subproblemas. Dicho algoritmo deberá ser comparado con uno proveído por el mismo profesor de la asignatura, *Francisco Vidal Fernández Fernández,* buscando una mejora en prestaciones con respecto a este.

Además de este algoritmo, se deberá implementar una variante, que tendrá que cumplir con ciertas restricciones impuestas por un nuevo problema.

La filosofía de los algoritmos basados en agregación es descomponer el problema de optimización multi-objetivo en varios subproblemas de un único objetivo. La función objetivo de cada subproblema debe ser una cierta agregación de los objetivos del problema multi-objetivo.

Aquí utilizaremos la formulación de Tchebychef, definida de la siguiente manera:

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

Donde $\lambda = (\lambda_1, \cdots, \lambda_m)^T$ es un vector peso y es $z^* = (z_1^*, \cdots, z_m^*)^T$ un punto de referencia donde cada una de sus componentes está dado por:

$$z_i^* = \min \{ f_i(x) \quad x \in \Omega \}$$

Al término de la ejecución de cada algoritmo nos deberá devolver un conjunto de posibles soluciones adecuadas a nuestro problema lo mas próximas posible al frente de Pareto del espacio de búsqueda.

Enunciado del problema

Consideremos sin pérdida de generalidad un problema multi-objetivo consistente en la minimización de *m* funciones objetivo:

minimize $F(x) = (f_i(x), \cdots, f_m(x))$

Donde el espacio de búsqueda x ∈ Ω está definido mediante los límites inferior y superior de cada de las variables que lo componen:

$$x_{Li} \leqslant x_i \leqslant x_{Ui}, \; i \in [1, p]$$

En el caso del problema de **optimización sin restricciones** 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:

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

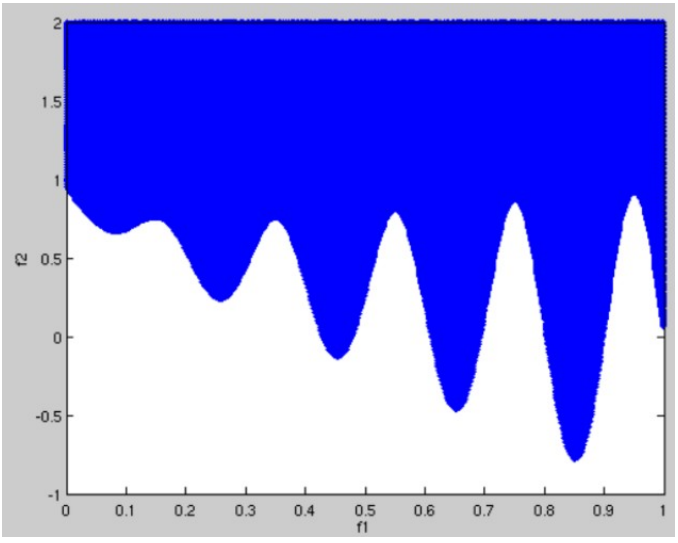
y

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

y el espacio de búsqueda es

$$0 \leq x_i \leq 1$$

Representación del espacio de búsqueda:



Y para el caso de **optimización con restricciones** el enunciado es el siguiente:

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

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

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:

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

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

El espacio de búsqueda es

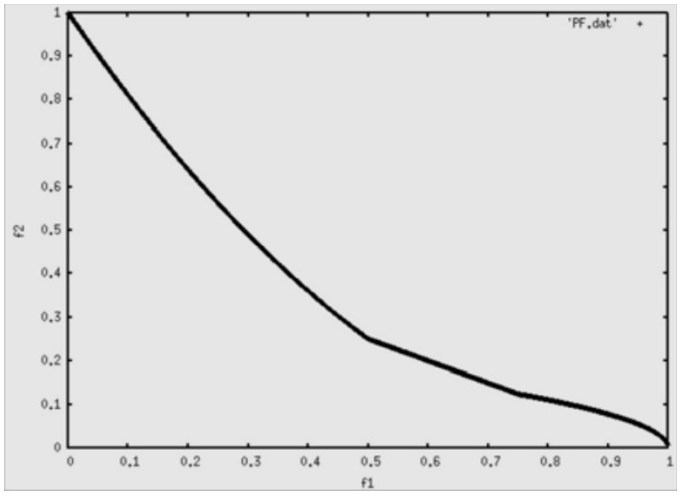
$$0 \le x_1 \le 1$$

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

El frente Pareto óptimo es conocido y viene dado por la siguiente ecuación analítica:

$$f_2 = \begin{cases} (1 - f_1)^2 & \text{si } 0 \le f_1 \le 0.5 \\ 0.5(1 - f_1) & \text{si } 0.5 < f_1 \le 0.75 \\ 0.25\sqrt{(1 - f_1)} & \text{si } 0.75 < f_1 \le 1 \end{cases}$$

cuya representación gráfica es:



Lenguaje y entorno utilizado

Para el desarrollo de este algoritmo he decidido utilizar el lenguaje de programación **C++**. Este lenguaje con su implementación de clases me permitirá crear estructuras mas avanzadas que con C.

En un inicio comencé utilizando el entorno de programación de **visual studio code,** pero, pasadas dos semanas del comienzo del proyecto, tuve que migrar al entorno **eclipse** debido a que la complejidad del código se hizo mayor y a una muy necesaria herramienta de depuración mas avanzada.

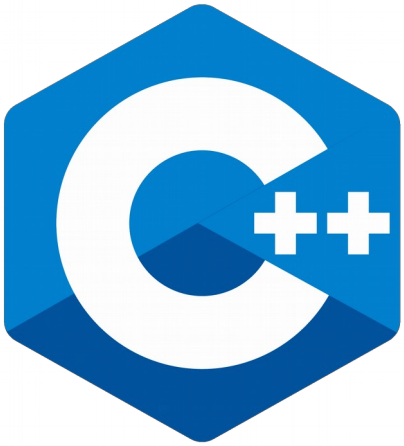
Desde el primer momento creé un repositorio **git** en la página web de GitHub para poder llevar un mejor control de versiones y poder recuperar los archivos en caso de pérdida.

Enlace al repositorio:https://github.com/Ampaex/Algoritmo-evolutivo-agregacion

Me he servido de la herramienta **gnuplot** para representar los resultados del algoritmo en gráficas bidimensionales y de los ficheros **METRICS** entregados por el profesor que me permiten calcular y comparar las métricas de mis ejecuciones.

Métricas utilizadas para comparaciones: Hipervolumen, Espaciado y Cobertura.

Todo ello desarrollado, compilado y ejecutado bajo un sistema operativo **GNU/Linux Debian Buster**.



Algoritmo sin restricciones

Estructuras definidas

Es conveniente que antes de proceder con la explicación se haga una breve descripción de las estructuras y funciones principales que va a tener este algoritmo.

Clase peso: Representa a nuestro peso y almacenará un vector de dos componentes llamado **vector**, definirá sus coordenadas dentro del espacio de objetivos de dos dimensiones.

Clase solucion: Representa a una solución y contiene su posición dentro del espacio con un puntero llamado **vector**. Además, también tiene una variable de tipo entero llamado **dimensiones** que guardará cuantos componentes tiene el vector de la solución.

Clase subproblema: La clase subproblema está compuesta por un **peso** **id**, una **solucion** **x** y un puntero **grupo** que guardará la posición de los vecinos que le corresponde. Extraeremos las dimensiones de este puntero grupo a partir del porcentaje de vecindad y del número de subproblemas.

Función **double** **tchebycheff**() → Nos indica como de cerca nos encontramos de la solución ideal **z** tomando en cuenta una solución y un peso aplicando la función de tchebycheff.

Función **solucion** **busquedaAobjetivo**() → Recibiendo una solución del espacio de búsqueda le aplica las funciones f1 y f2 y nos devuelve la **solucion** correspondiente en el espacio de objetivos.

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

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

donde:

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

y

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

Parámetros de entrada

- **N:** Número de subproblemas, equivale al tamaño de la población que se utiliza en otros algoritmos. **[Número entero > 0]**

- **G:** Número de generaciones, en nuestro caso actúa como condición de parada. El algoritmo terminará su ejecución se hayan realizado tantas iteraciones como número de generaciones. **[Número entero > 0]**

- **Dim:** Número de dimensiones del espacio de búsqueda. **[Número entero > 0]**

- **F:** Parámetro de mutación, define cuánto diferirá el valor mutado del valor original. **[Número real 0-2]**

- **CR:** Parámetro de cruce, a mayor valor, mas componentes de la solución mutada pertenecerán al vector cruzado. **[Número real 0 - 1]**

- **Vecindad:** Tamaño de la vecindad de cada subproblema, porcentaje sobre N. Indica el tamaño del grupo de subproblemas vecinos. **[Porcentaje 0 - 100]**

- **Ls y li**: Definen el máximo y el mínimo valor que puede tomar cada componente del vector solución del espacio de búsqueda. **[Números enteros]**

- **Semilla:** Número que actuará como generador de números aleatorios. **[Número entero]**

Inicialización 1.0

1) **Creación de N peso** $\lambda^1, \lambda^2, \ldots, \lambda^N$, uno para cada subproblema. Cumpliéndose que $\sum_{i=1}^{2} \lambda_i^j = 1$. Distribuidos uniformemente. Para crearlos se ha procedido asignando el valor (0, 1) y luego al primer componente del vector se le va aumentando un *paso* por iteración y el segundo restando un *paso*. Donde: $paso = \frac{1}{N-1}$ Así hasta alcanzar (1, 0).

2) **Creación de los N subproblema** a los que se le pasa por parámetros su **peso** correspondiente, **vecindad**, el grupo de **pesos**, **N**, **dim**, **ls** y **li**. El constructor de **subproblema** requiere todos estos parámetros porque en su creación se encarga de guardar su **peso**, calcular cuál es su grupo de **pesos vecinos** y de crear una nueva **solución** aleatoria entre **ls** y **li** con dimensiones **dim**.

   a) Para calcular el grupo de vecinos de un subproblema he calculado todas las distancias desde mi vector peso al resto de pesos y me he quedado con los $\frac{vecindad * N}{100}$ pesos mas cercanos. Se ha almacenado su posición dentro del array de suproblemas.

3) **Los subproblemas son copiados** a un puntero global con el fin de no perderlos cuando la función de inicialización termine.

4) **Búsqueda de la mejor solución** de toda la inicialización, esta será guardada en una variable global llamada **z** que será el punto de referencia, la mejor solución ideal encontrada del espacio de objetivos. Para ello se recorren todos los **subproblemas**, para cada uno se extrae su **solución**, se le aplica las funciones del espacio de objetivos y se comprueba si cada uno de sus componentes mejora a los que se encuentran ya guardados.

Ejecución 1.0

La función de ejecución se repite tantas veces como (**G**-1) y se recorren todos los subproblemas por cada iteración.

1) **Generación de** un array que contenga **3 enteros** escogidos aleatoriamente.

2) Con las 3 cifras generadas **se accede** al vector que contiene las **posiciones de los vecinos** y se escogen los 3 subproblemas vecinos que se encontraban en las 3 posiciones que indicaban esas cifras. Por ejemplo: Numeros = {0,3,2} | grupo[0], grupo[3],grupo[2] ⟶ Subproblemas 2, 4 y 6

3) Con las soluciones de esos 3 subproblemas (**x**) **se realiza la mutación** de evolución diferencial: $$v^{(i)}(G+1) = x^{(r1)}(G) + F \cdot (x^{(r2)}(G) - x^{(r3)}(G))$$

4) Una vez tenemos un nuevo vector solución mutado **se hace el cruce** entre la solución anterior de nuestro subproblema y la nueva mutada:

$$u_{ij}(G+1) = \begin{cases} v_{ij}(G+1) & \text{if } (rng([0,1] \le p_c) \text{ or } j = \delta \\ x_{ij}(G) & \text{if } (rng([0,1] > p_c) \text{ or } j \ne \delta \end{cases}$$

5) **Actualización de z**, comprueba si la nueva solución mejora algún valor de **z,** si es así lo sustituye.

6) Se **comprueba si la nueva solución mejora la de los vecinos(incluído el mismo)**, es decir:

tchebycheff(nueva_solucion, peso_vecino) < **tchebycheff**(solucion_vecino, peso_vecino) . Si es así, sustituye la solución anterior por la nueva. Peso_vecino = peso de el subproblema que se está evaluando.

Comparativa versión 1.0 con el algoritmo de referencia

A continuación se van a enfrentar las soluciones y estadísticas del algoritmo de referencia contra el trabajo realizado en este proyecto. Se va a tener igualdad de evaluaciones, unos ajustes comunes a los dos, y unos ajustes propios a mi algoritmo ajustados por mi. Los resultados de cada ejecución, a excepción de la primera, serán ajustados a una página con un formato de ficha.

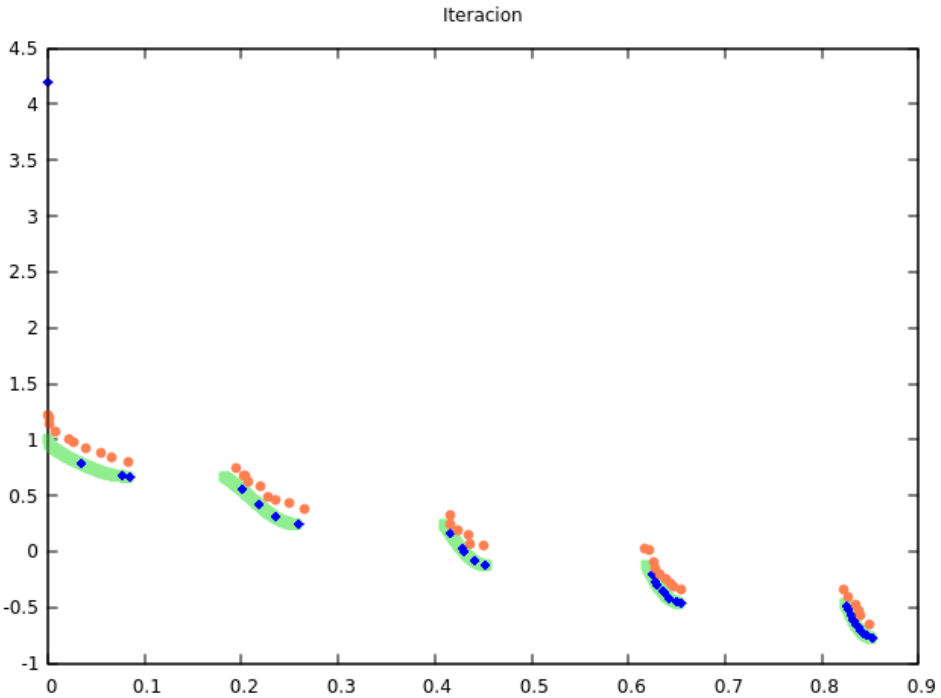
Ejecución 1, comparación con semilla1: **4000 evaluaciones**

Ajustes comunes: 40 individuos | 30 dimensiones | 100 generaciones

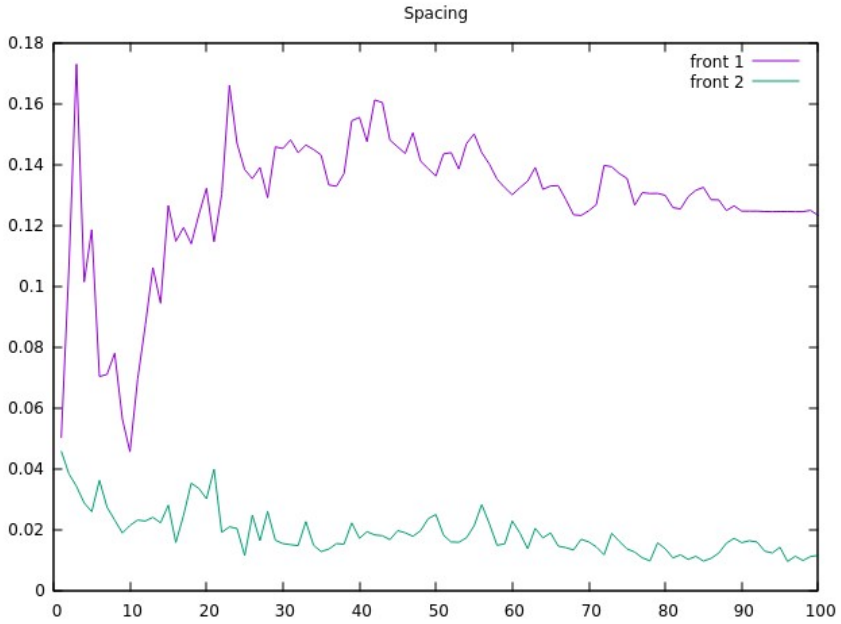
Ajustes propios: F 0.55 | CR 0.6 | Vecindad 40% | semilla 0.3

Última generación de ambos algoritmos

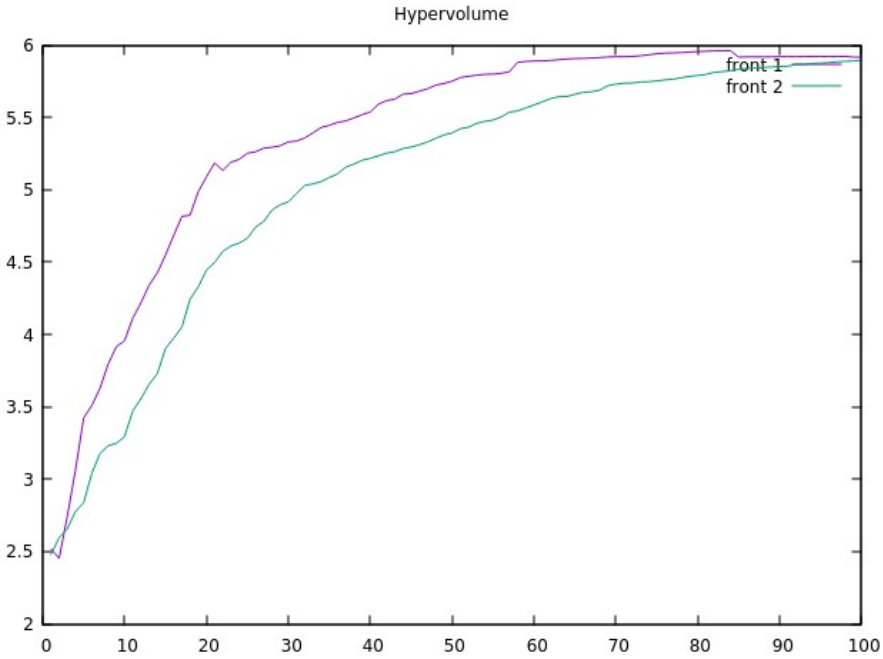
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



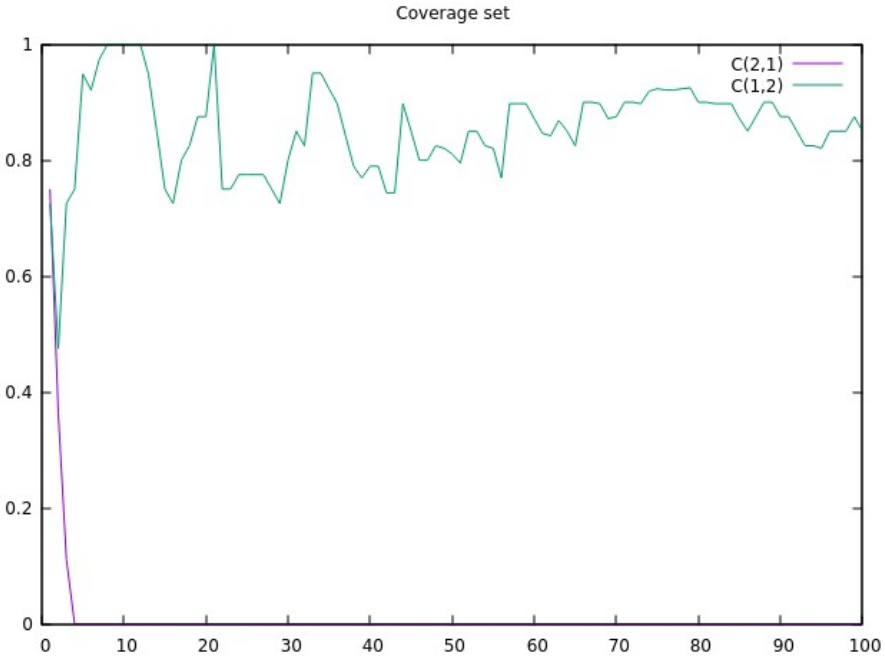
Como se puede apreciar, las soluciones propias, las azules, se han impuesto de forma notable a las soluciones del algoritmo de referencia. También se ha conseguido frente bastante bien alcanzado pero con una densidad de soluciones descompensada, se ven mas concentradas a medida que avanzamos por el eje X. Esto es debido a como se encuentra implementado el algoritmo.



El espaciado en mi algoritmo se presenta un poco peor que el de referencia con una pequeña diferencia de 0.11 aproximadamente. Se puede concluir también que en la iteración 1 los dos generadores de soluciones aleatorias han obtenido un espaciado muy similar. El espaciado en nuestro algoritmo se consigue por medio de la distribución uniforme de pesos, que nos sirve para que cada subproblema valore de una forma diferente cada objetivo produciendo una cierta homogeneidad sobre el frente de Pareto.



El hipervolumen del frente 1, el mio, se encuentra siempre por delante del de referencia llegando ambos a conseguir un valor final muy próximo pero aun así continúa siendo un poco peor el de referencia. Punto de referencia para todas las pruebas: (1, 6)



Cuando se trata de cobertura mi algoritmo se encuentra cubriendo a el otro un 90% desde aproximadamente la iteración 3 debido a que mi frente siempre se ha encontrado por delante. La cobertura del algoritmo de referencia nos revela que prácticamente en ningún momento de la ejecución una solución suya se ha puesto por delante de la mía.

Valores finales	Hipervolumen	Espaciado	Cobertura
Propio	5.9172846267	0.1231715563	C(1,2) 0.85
Referencia	5.8954643081	0.0114129476	C(2,1) 0

Como conclusión de esta primera comparación se puede extraer que el nuevo algoritmo versión 1.0 ya mejora al de referencia en gran medida. Como estamos tratando con algoritmos estocásticos las soluciones que nos arrojan podrían no tener siempre las mismas prestaciones, todo esto depende de la probabilidad, por lo tanto, no podemos compararlos con una sola prueba.

Procederemos a realizar mas pruebas con distintos parámetros y semillas.

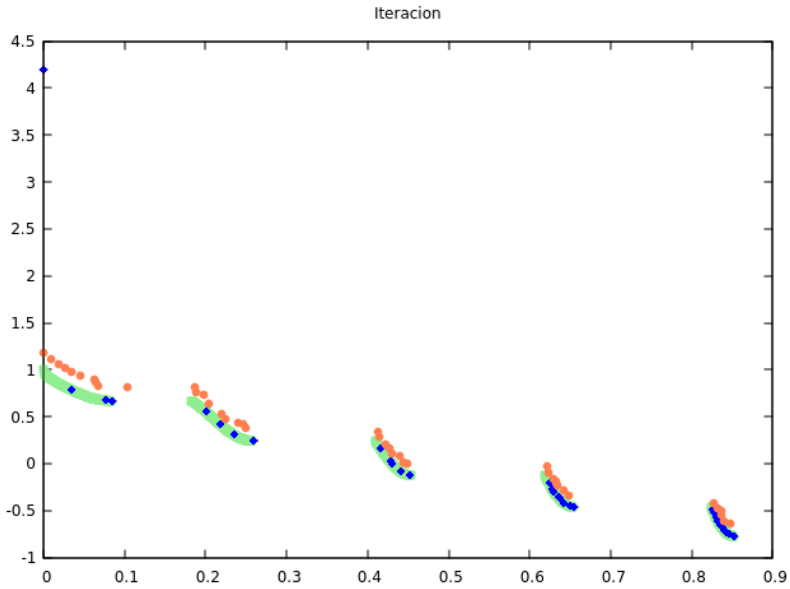
<u>Ficha ejecución 2, comparación con semilla2:</u> 4000 evaluaciones

Ajustes comunes: 40 individuos | 30 dimensiones | 100 generaciones

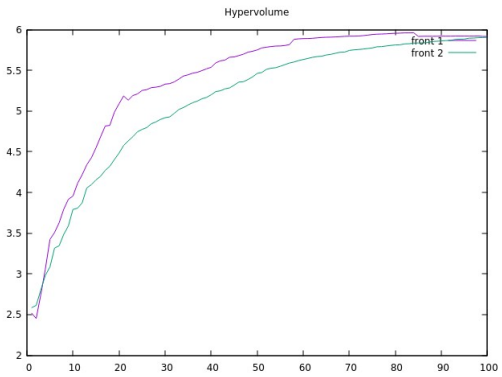
Ajustes propios: F 0.55 | CR 0.6 | Vecindad 40% | **semilla 0.4**

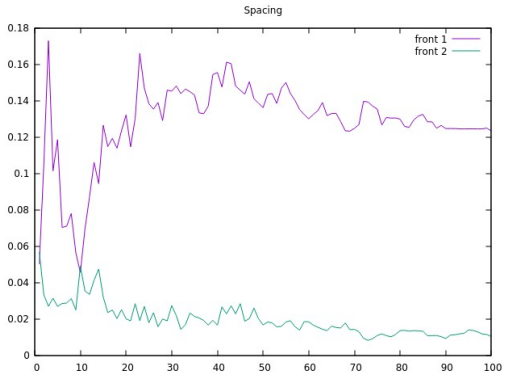
<u>Última generación de ambos algoritmos</u>

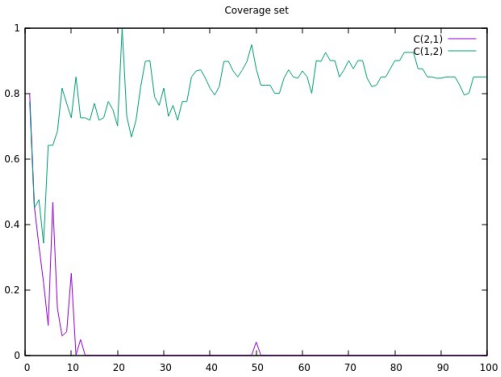
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



Ejecución para comprobar que se aprecian resultados muy similares a los de la primera ejecución con **mismos parámetros** pero **distinta semilla**.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio	5.9172846267	0.1231715563	C(1,2) 0.85
Referencia	5.9065131037	0.0101704618	C(2,1) 0

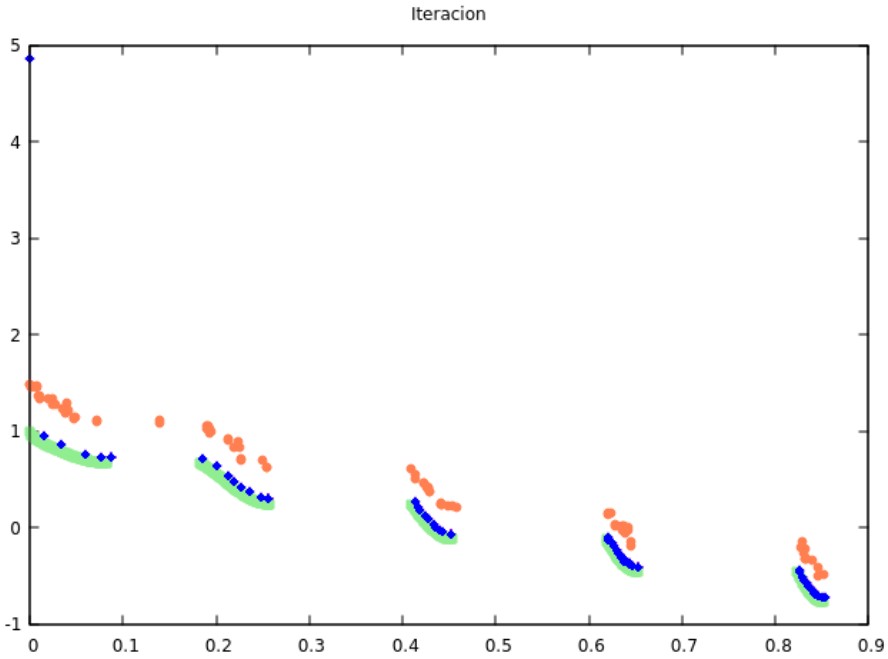
Ficha ejecución 3, comparación con semilla1: 4000 evaluaciones

Ajustes comunes: 80 individuos | 30 dimensiones | **50 generaciones**

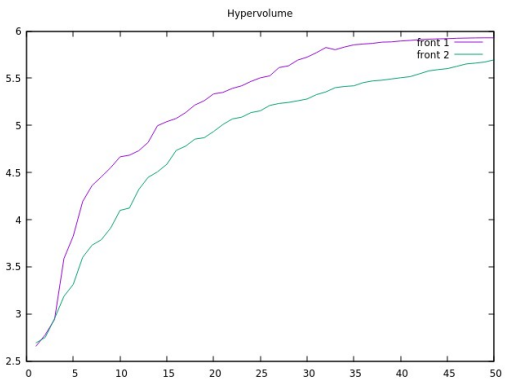
Ajustes propios: F 0.55 | CR 0.6 | Vecindad 40% | semilla 0.4

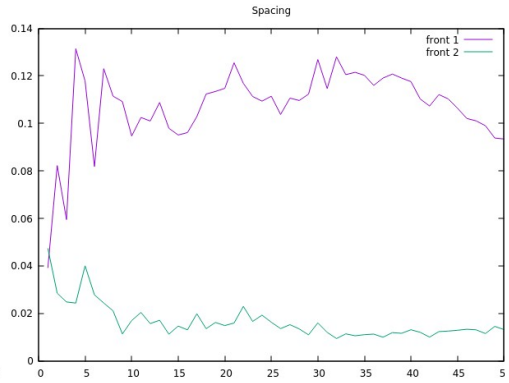
Última generación de ambos algoritmos

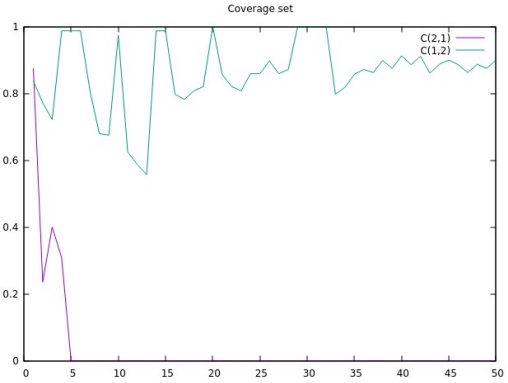
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



El algoritmo también se encuentra por delante con **nuevos parámetros y misma semilla que la anterior**.







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio	5.9294359272	0.0934163721	C(1,2) 0.9
Referencia	5.6955432367	0.0081391350	C(2,1) 0

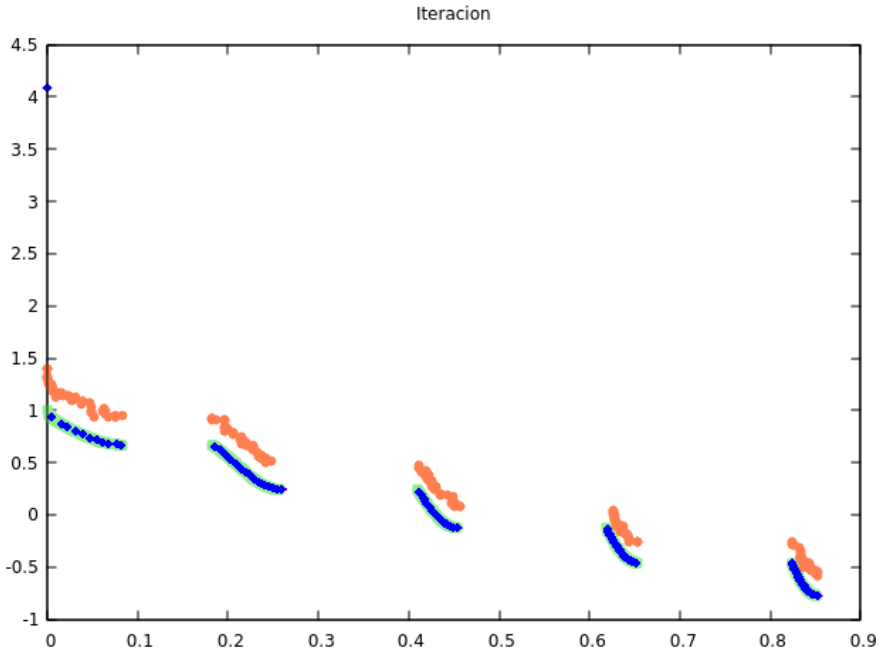
Ficha ejecución 4, comparación con semilla1: 10000 evaluaciones

Ajustes comunes: 200 individuos | 30 dimensiones | **50 generaciones**

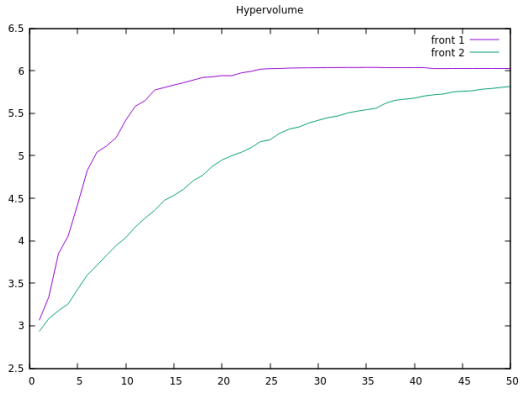
Ajustes propios: F 0.55 | **CR 0.65** | Vecindad 40% | semilla 0.4

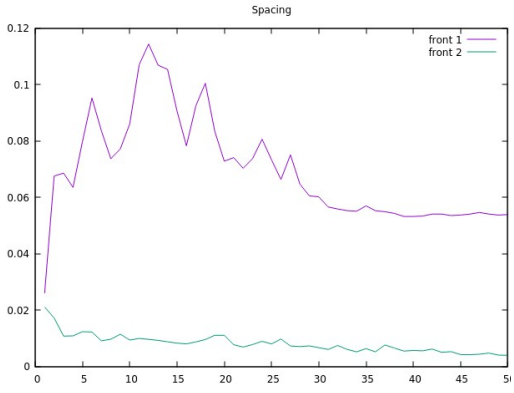
Última generación de ambos algoritmos

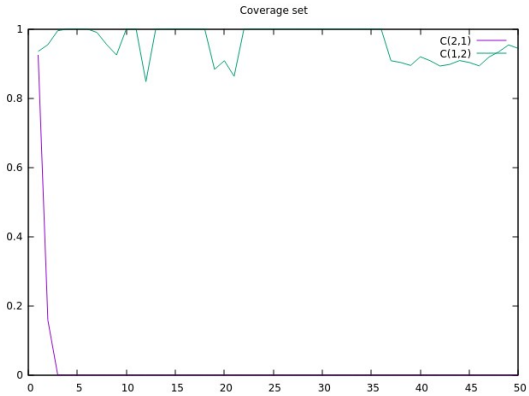
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



Para probar que el algoritmo sigue ganando cuando se aumenta el número de evaluaciones y con muchos individuos. El único **parámetro propio actualizado** para obtener un rendimiento óptimo es: CR0.60 ⟶ CR0.65







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio	6.0266551565	0.0538532794	C(1,2) 0.94358974
Referencia	5.8151371880	0.0038991716	C(2,1) 0

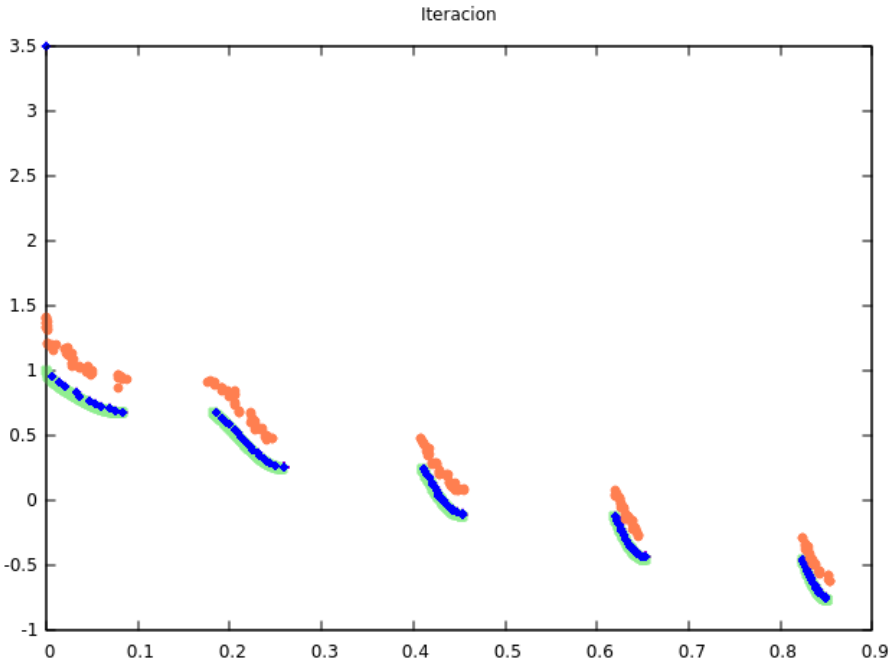
Ficha ejecución 5, comparación con semilla2: 10000 evaluaciones

Ajustes comunes: 200 individuos | 30 dimensiones | 50 generaciones

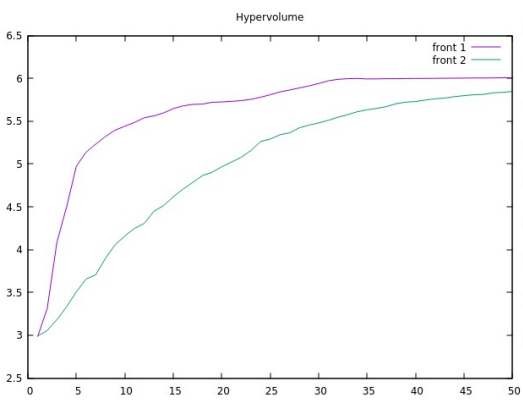
Ajustes propios: F 0.55 | CR 0.65 | Vecindad 40% | **semilla 13**

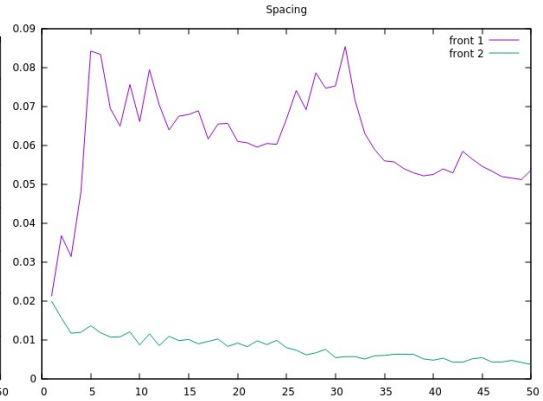
Última generación de ambos algoritmos

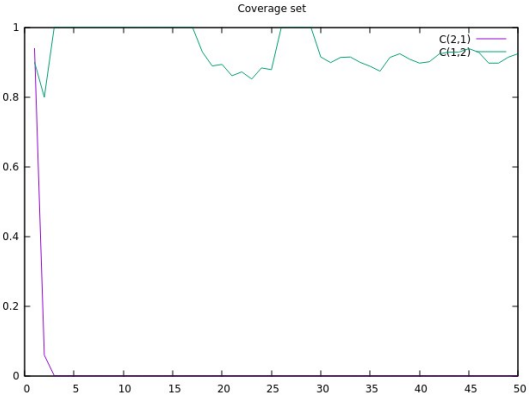
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



Prueba del algoritmo con una semilla muy distinta ahora es 13 en lugar de 0.4.







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio	6.0080451394	0.0536299029	C(1,2) 0.92424242
Referencia	5.8482950310	0.0036157422	C(2,1) 0

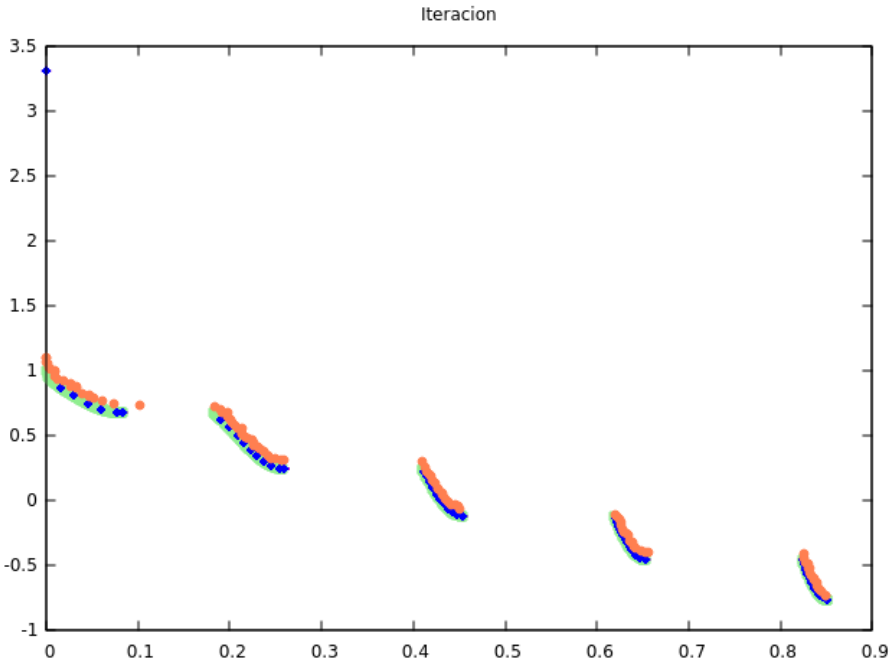
<u>Ficha ejecución 6, comparación con semilla1:</u> 10000 evaluaciones

Ajustes comunes: 100 individuos | 30 dimensiones | **100 generaciones**

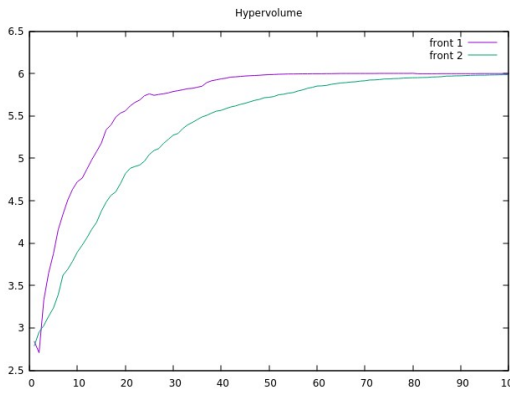
Ajustes propios: F 0.55 | CR 0.65 | Vecindad 40% | semilla 13

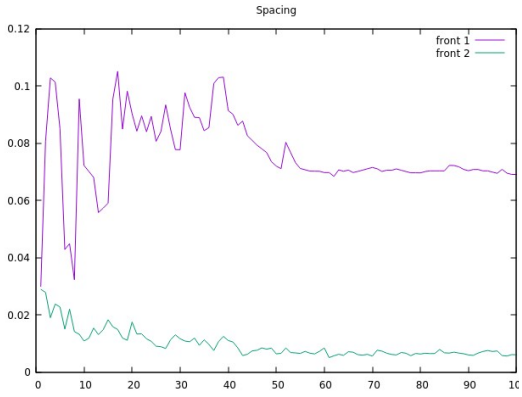
<u>Última generación de ambos algoritmos</u>

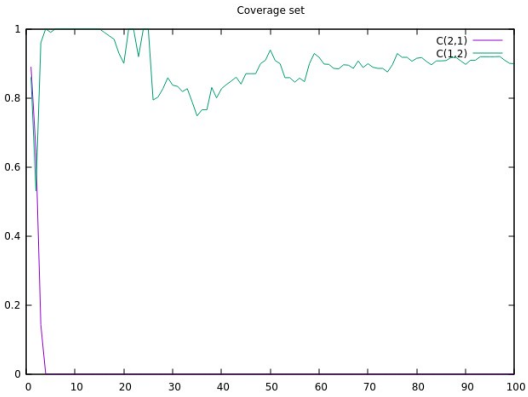
Naranja: Algoritmo de referencia **Azul:** Propio v1.0 **Verde:** Frente ideal



Ejecución que tiene los mismos individuos que generaciones.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio	6.0000545573	0.0690433455	C(1,2) 0.89898989
Referencia	5.9862712163	0.0060045849	C(2,1) 0

Comparación estadística

Tras haber visto todos estos resultados variados podemos concluir que a rasgos generales el nuevo algoritmo consigue unas soluciones de mejor calidad y en menor tiempo que las soluciones del anterior , a cambio tenemos una pérdida de espaciado entre las soluciones. Pero sabiendo esto, es fácil ahora preguntarse: ¿En cuánto mejoran estos resultados a los anteriores?

Pues ahora continuamos con un análisis estadístico de las soluciones

El punto de referencia para el hipervolumen utilizado para todas las pruebas anteriores ha sido el (1, 6), por lo tanto, sus valores son comparables entre sí.

Vamos a agrupar todos los datos anteriores bajo un valor que sea representativo. Como no tenemos valores muy dispares nos puede servir la media:

Media	Propio	Referencia
Hipervolumen	5,96646000563333	5,85787068063333
Espaciado	0,08604766875	0,007207007183333
Cobertura	0,894470341666667	0

TOTAL

+1,8% de hipervolumen

12 veces menos espaciado

89,45% de cobertura

Interpretación final de resultados

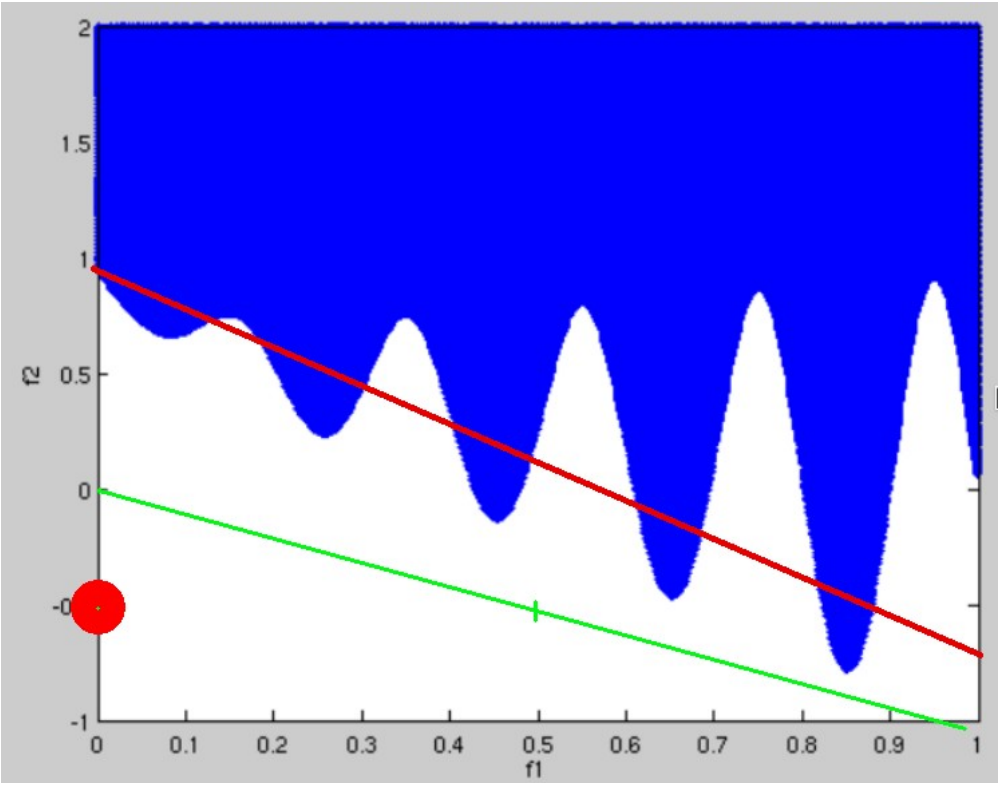
Tras toda la batería de pruebas y cálculo estadístico procedemos a analizar e interpretar nuestros resultados.

Espaciado:

Aquí es donde se encuentra el punto débil de nuestro algoritmo, las soluciones que nos entrega no tienen una distribución uniforme a lo largo del eje 'x', se aprecia una mayor concentración a medida que avanzamos por este.

Se debe a la forma que tiene nuestro algoritmo de elegir las soluciones que van a reemplazar a las anteriores, esto es, la función de tchebycheff y su uso de la proximidad a la solución 'z'. Por la forma del frente esta acaba no siendo equidistante a todas las soluciones y se produce una descompensación.

El punto rojo sería Z en una iteración cualquiera, la línea roja es la 'pendiente' de nuestro frente y la verde es una cuerda(que he prolongado) de la circunferencia que formarían los puntos equidistantes a 'z', mas concretamente una circunferencia de radio 0.5 y centro en z (0, -0.5). Este gráfico no tiene la misma escala en 'x' que en 'y', por eso no se ve claramente el 'concepto' de la línea verde.

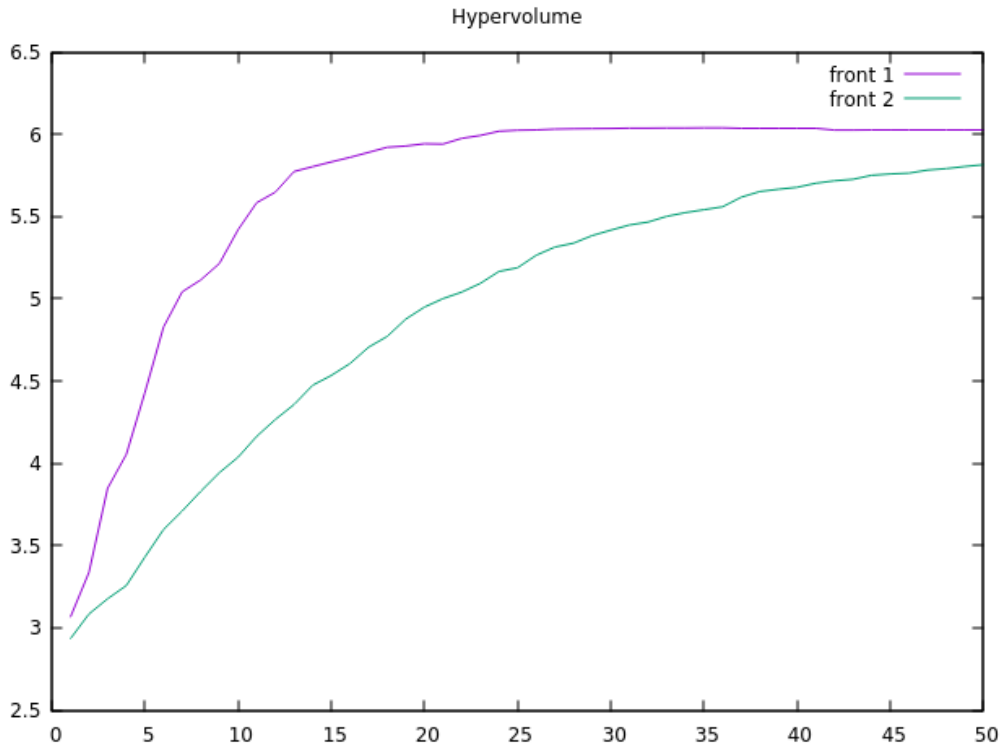


Como se puede ver ambas líneas no son paralelas. Por lo tanto z no está ponderando igual unas soluciones que otras. En el lado derecho se encuentran mas próximas las líneas y debido a esto se aprecia una mayor concentración de soluciones a medida que vamos avanzando en el eje x.

Hipervolumen:

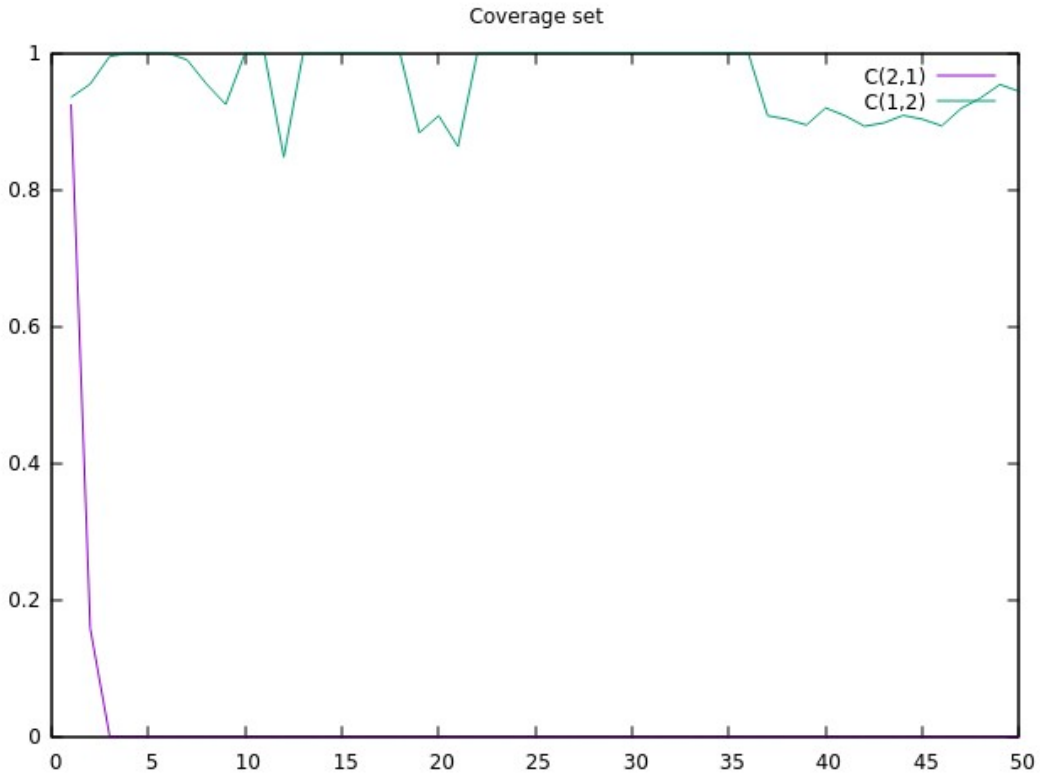
En este apartado el nuevo algoritmo consigue un mayor valor y además aumenta mucho mas rápido que el de referencia. Donde es más acusada esta diferencia es cuando tendemos a mas individuos y menos iteraciones, por ejemplo con 200 individuos y 50 iteraciones. Lo que podemos extraer de aquí es que este algoritmo le saca mucho mas partido a mas individuos que a mas iteraciones.

Como se puede ver en este gráfico de la ficha 4, en la generación 15, mi algoritmo ha alcanzado prácticamente el valor final mientras que el de referencia va aproximadamente por la mitad:



Cobertura:

En cobertura, al comienzo de todas las ejecuciones ambos algoritmos se encuentran aproximadamente en el mismo punto, pero debido a una evolución mas rápida, el frente del algoritmo propio acaba dominando en gran medida durante el resto de la ejecución hasta el final. Esto quiere decir que domina a prácticamente todas las soluciones del adversario. Ejemplo de ello es este gráfico de la ficha 4:



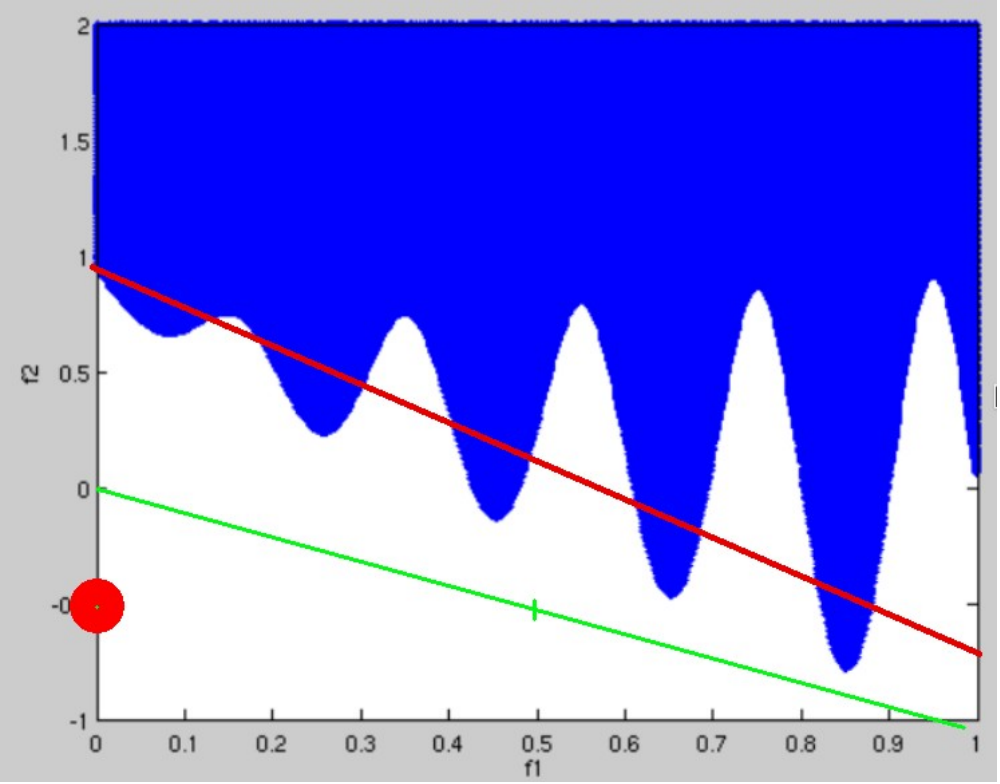
Mejora de la versión 1.0, versión 1.1

Como hemos visto en la interpretación de resultados, el punto débil de nuestro algoritmo es el espaciado, que aunque peor que en el de referencia, las soluciones siguen encontrándose bastante repartidas.

En caso de que no fuera tan crítico el espaciado para nuestra aplicación la versión 1 sería suficiente. Pero si por el contrario nos interesase contar con unas soluciones mas homogéneas, se ha desarrollado esta mejora, que a cambio de una pérdida de parte de las soluciones (derrame a partir de ahora) se obtiene un espaciado mucho mejor, una mayor cobertura y sin una gran pérdida de hipervolumen.

La mejora se basa en un nuevo parámetro que he decidido bautizar como compensación angular que viene a solventar el problema expuesto en la página 20.

Lo que hace este parámetro es desplazar el punto z horizontalmente para así dejar paralelas las dos líneas del gráfico:



La modificación del código es trivial con respecto a la versión anterior, pero es preciso saber por que ocurría este problema y donde hacer la modificación debida.

Este cambio se ha realizado en la función de **tchebycheff**() donde al momento de evaluar la distancia de la solución en cuestión hasta 'z' se le ha sumado el parámetro compensación angular a la componente 'x' de la solución ideal 'z'.

El resultado es que a medida que aumentamos el valor del nuevo parámetro se consigue un giro de la línea verde del gráfico en sentido antihorario mientras que disminuyendo obtenemos un giro en sentido horario.

En nuestro caso particular necesitamos rotar en sentido horario, por lo tanto le hemos dado un valor negativo a el parámetro compensación angular.

Vamos a ponerlo en práctica y a comparar resultados.

Comparativa versión 1.1 con referencia y 1.0

<u>Ejecución 1, comparación con semilla1:</u> 4000 evaluaciones

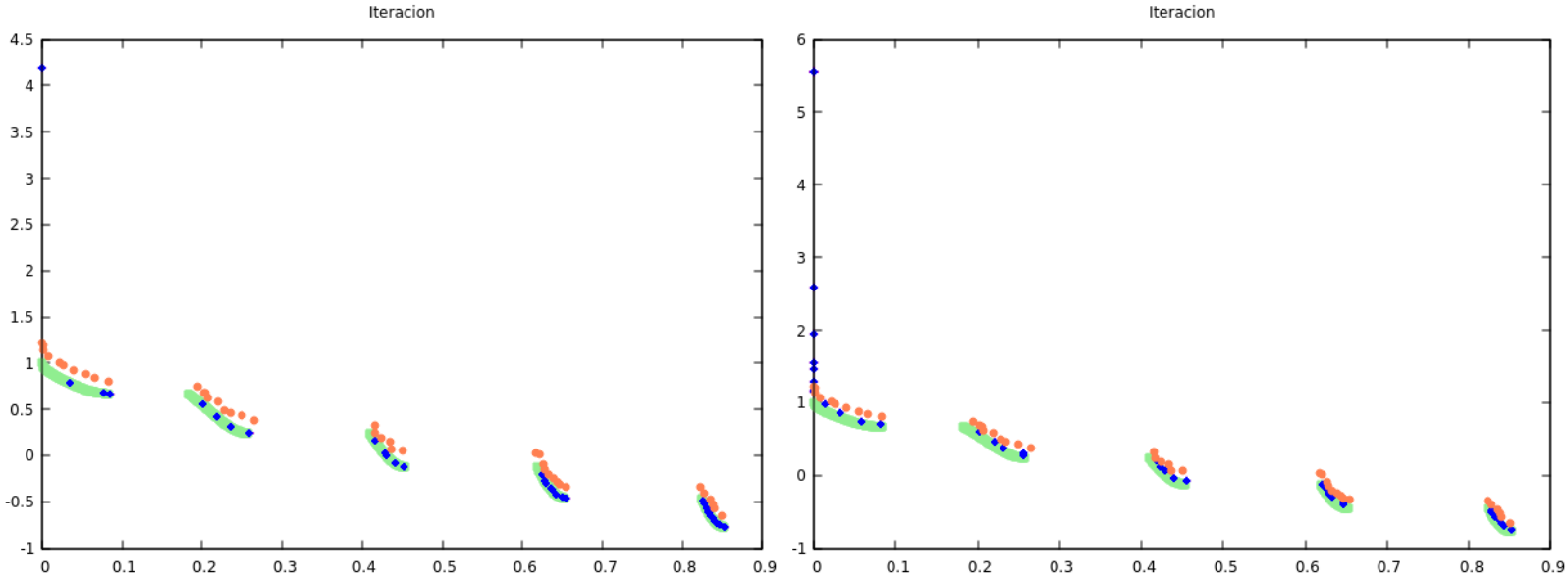
Ajustes comunes: 40 individuos | 30 dimensiones | 100 generaciones

Ajustes propios:F 0.55 | CR 0.6 | Vecindad 40% | semilla 0.3 | comp_angular -0,85

<u>Última generación de ambos algoritmos</u>

Naranja: Algoritmo de referencia **Azul:** Propio| izquierda 1.0 | derecha 1.1

Verde: Frente ideal



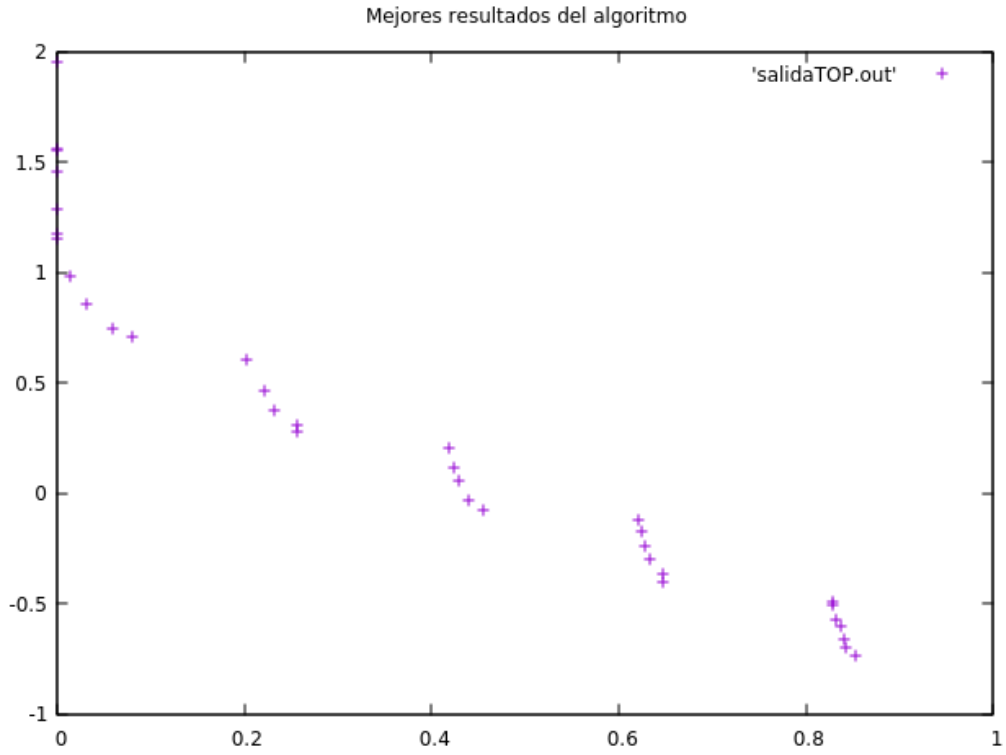
En la versión 1.1 se aprecia un mejor espaciado con respecto a la 1.0 pero con un 'derrame', que son las soluciones que se encuentran sobre el eje de ordenadas.

Quitando las soluciones que han producido derrame nos quedamos con un total de **28** soluciones con respecto a las 40 del algoritmo anterior.

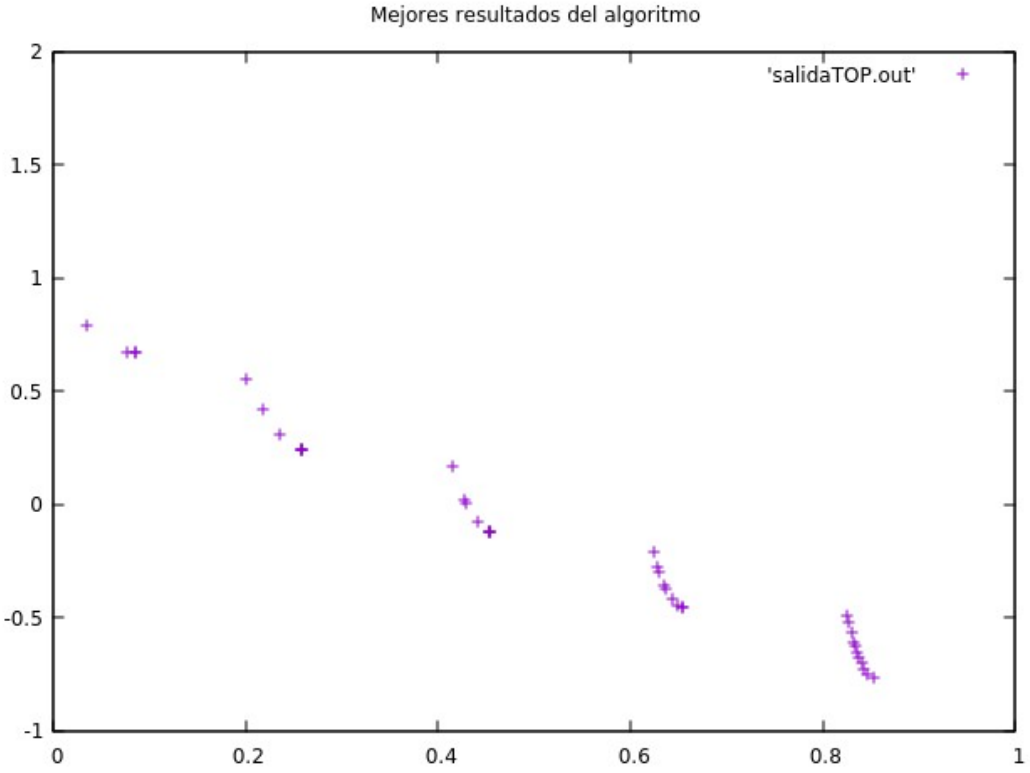
Hay un 43% menos de soluciones válidas, en cambio tenemos un mayor hipervolumen.

Gráficas para observar con mejor detalle la homogeneidad

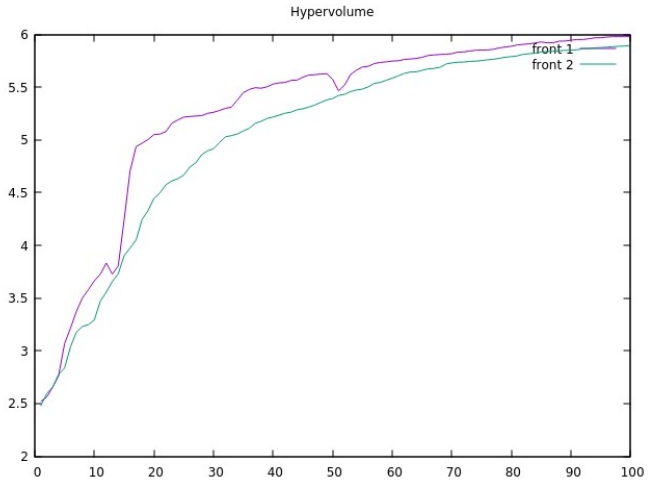
1.1

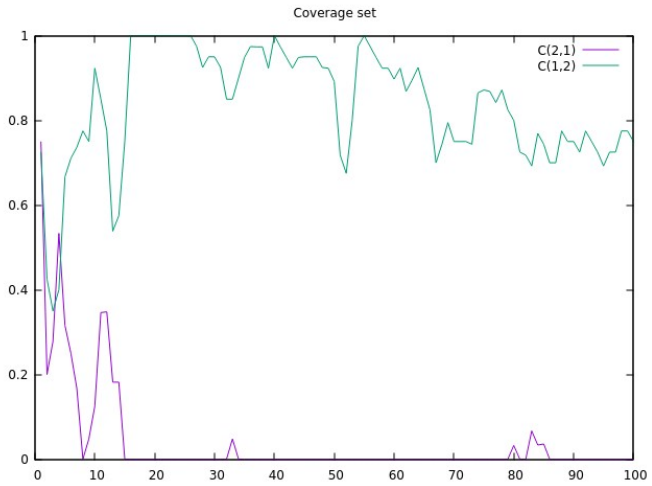


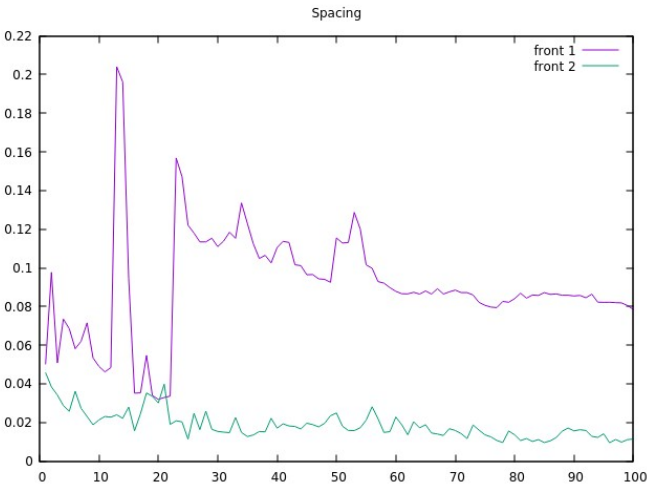
1.0



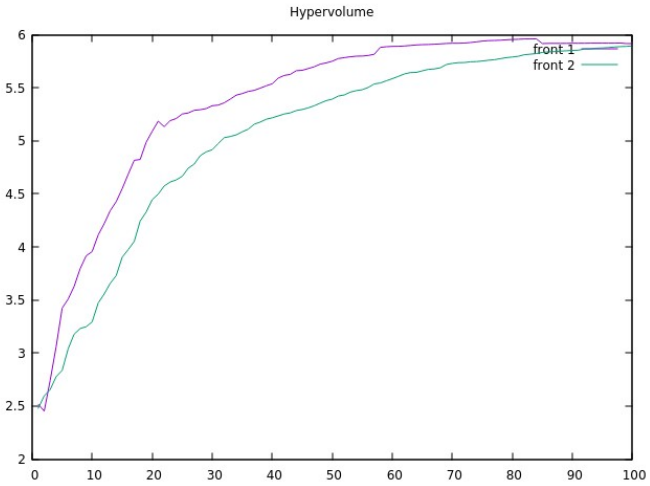
Métricas 1.1

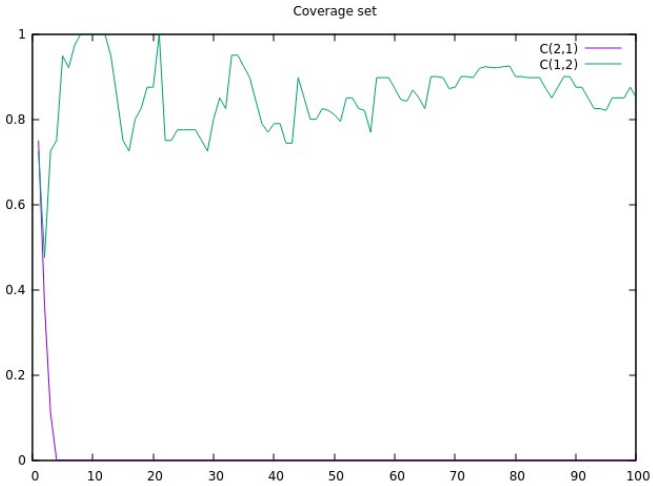


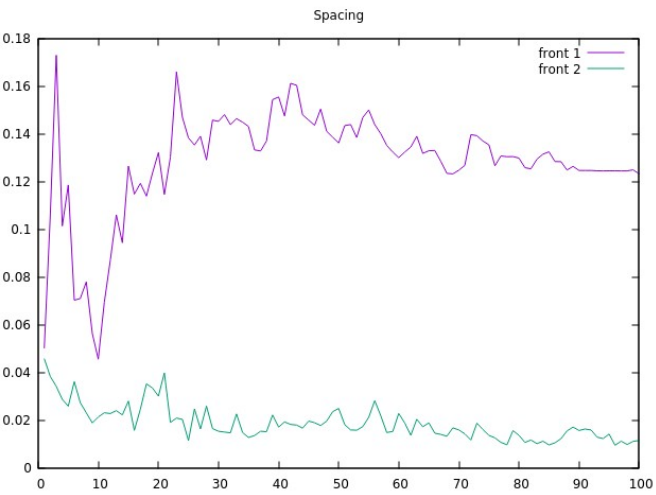




Métricas 1.0







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio 1.1	5.9806526684	0.0783880593	C(1,2) 0.75
Propio 1.0	5.9172846267	0.1231715563	C(1,2) 0.85

<u>Ejecución 2, comparación con semilla1:</u> 10000 evaluaciones

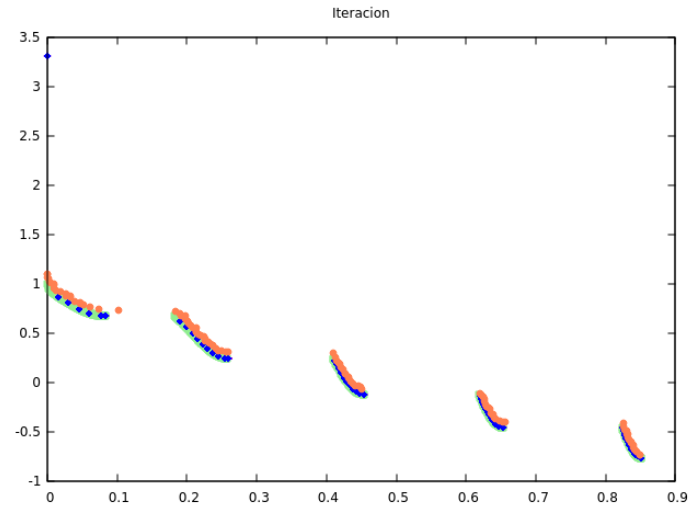
Ajustes comunes: 100 individuos | 30 dimensiones | 100 generaciones

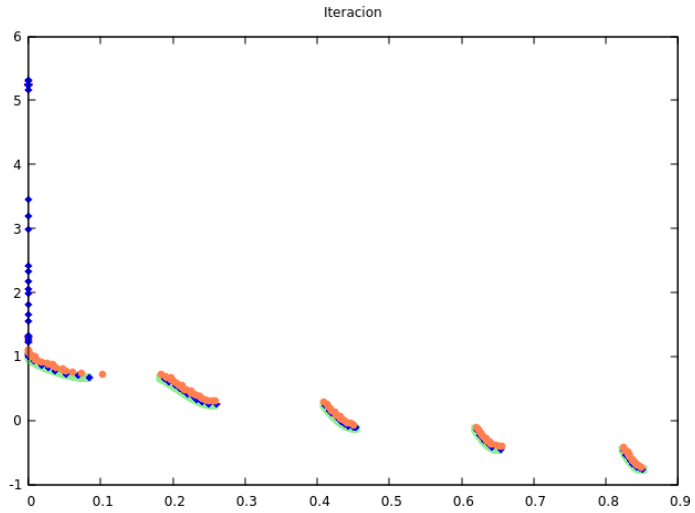
Ajustes propios:F 0.55 | CR 0.6 | Vecindad 40% | semilla 0.3 | comp_angular -1,2

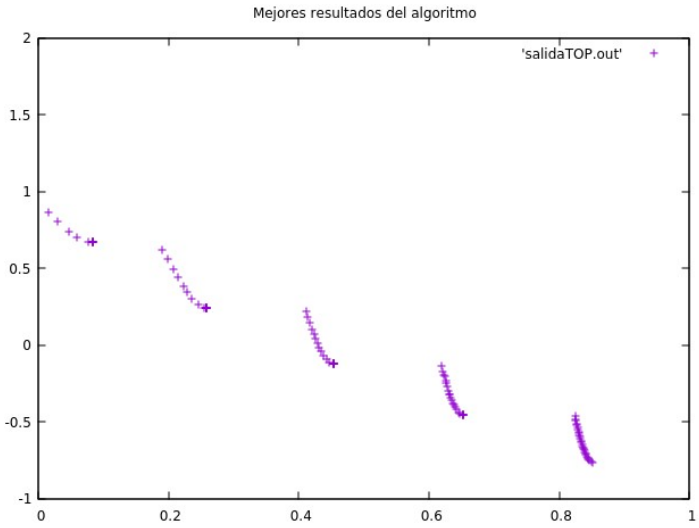
<u>Última generación de ambos algoritmos</u>

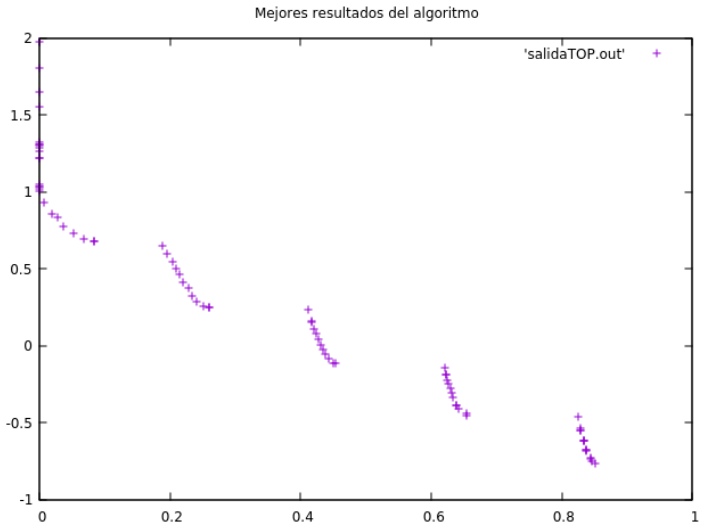
Naranja: Algoritmo de referencia **Azul:** Propio| izquierda 1.0 | derecha 1.1

Verde: Frente ideal



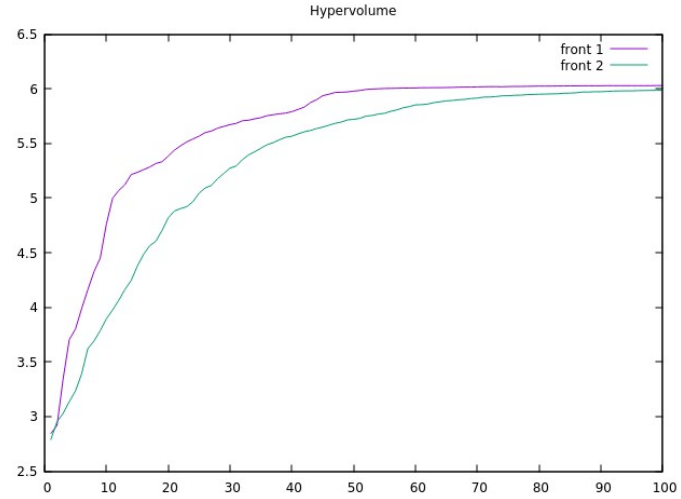


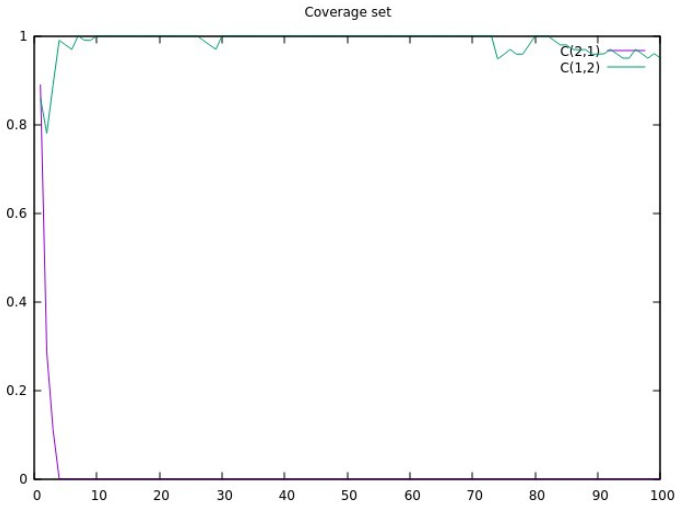


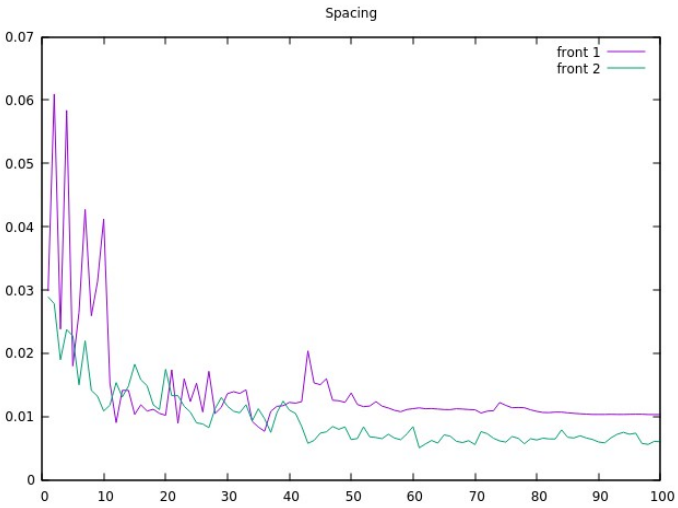


61 soluciones sin derrame

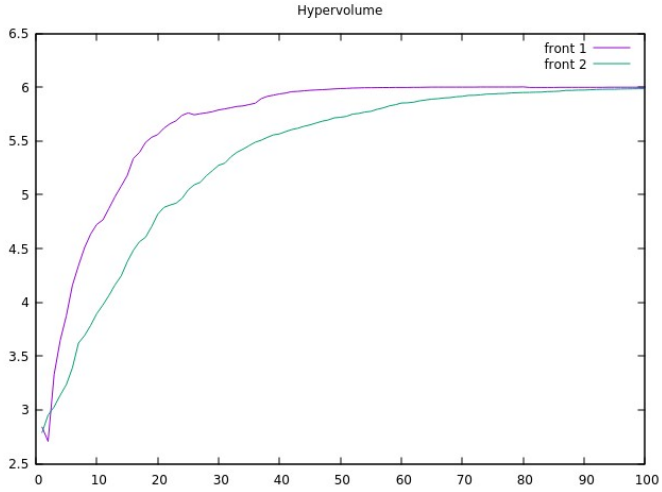
Métricas 1.1

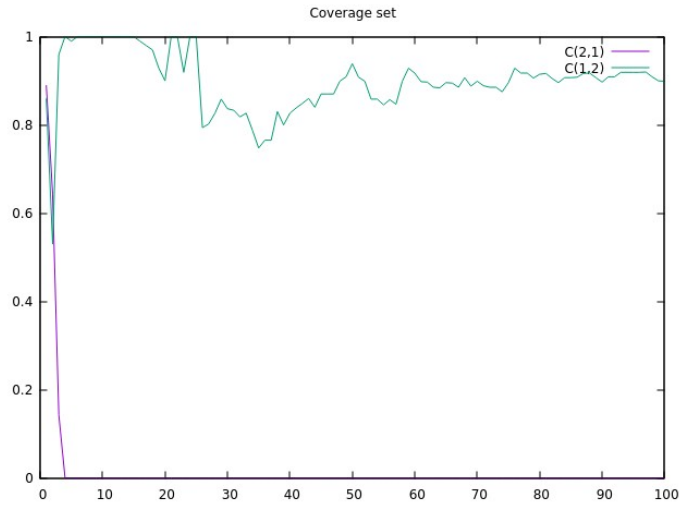


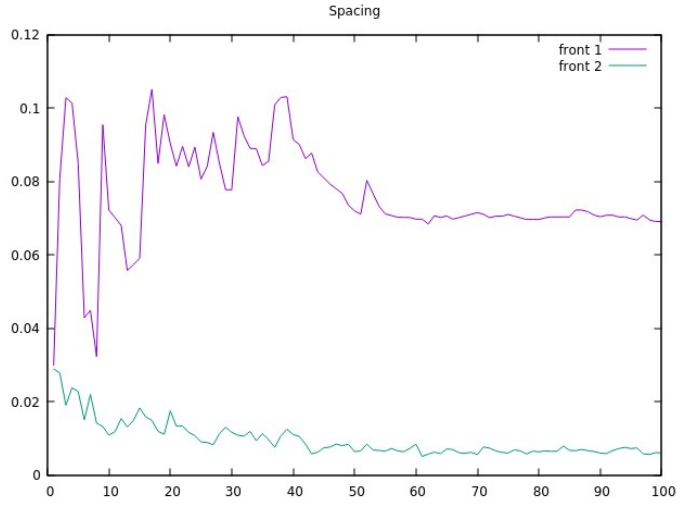




Métricas 1.0







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio 1.1	6.0294248953	0.0102982922	C(1,2) 0.94949494
Propio 1.0	6.0000545573	0.0690433455	C(1,2) 0.89898989

<u>Ejecución 3, comparación con semilla1:</u> 10000 evaluaciones

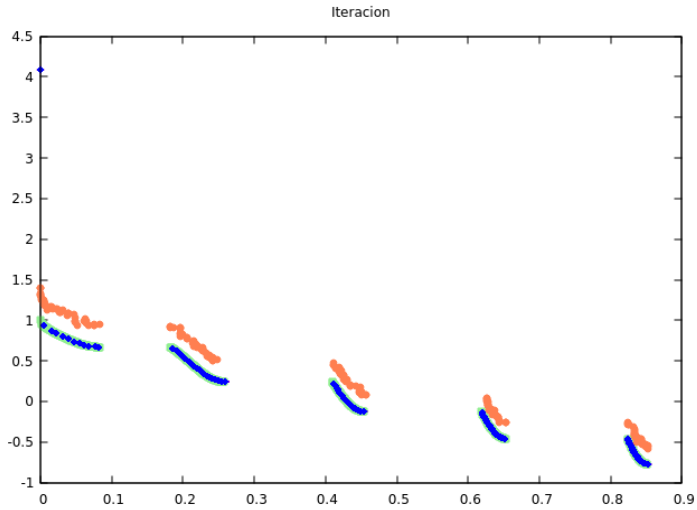
Ajustes comunes: 200 individuos | 30 dimensiones | 50 generaciones

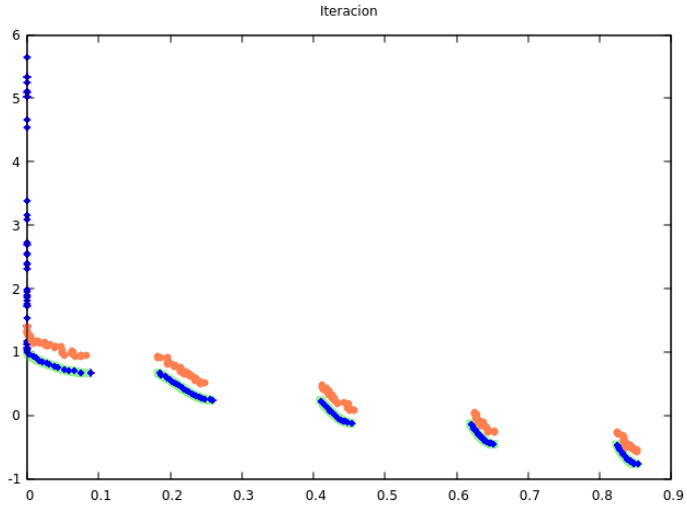
Ajustes propios: F 0.55 | CR 0.65 | Vecindad 40% | semilla 13|comp_angular -1,05

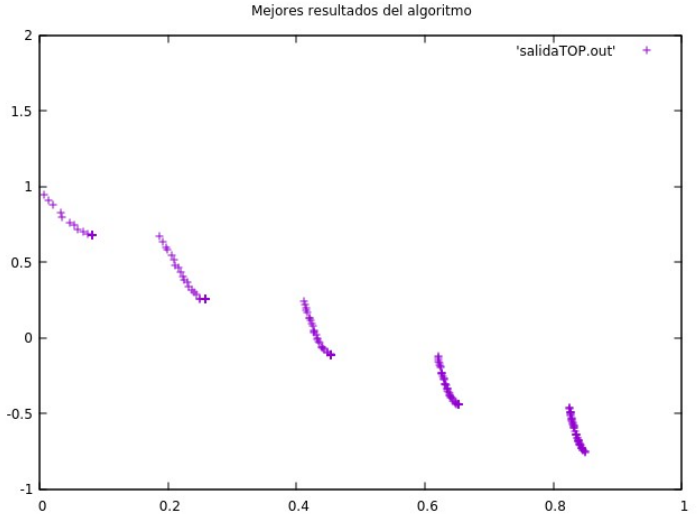
Última generación de ambos algoritmos

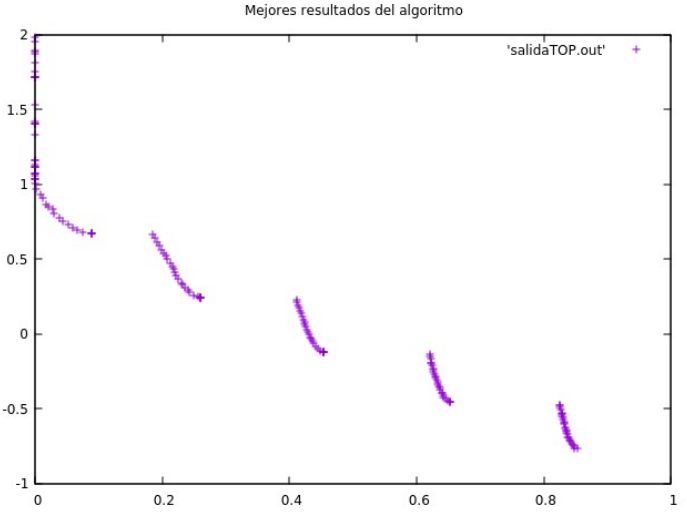
Naranja: Algoritmo de referencia **Azul:** Propio| izquierda 1.0 | derecha 1.1

Verde: Frente ideal



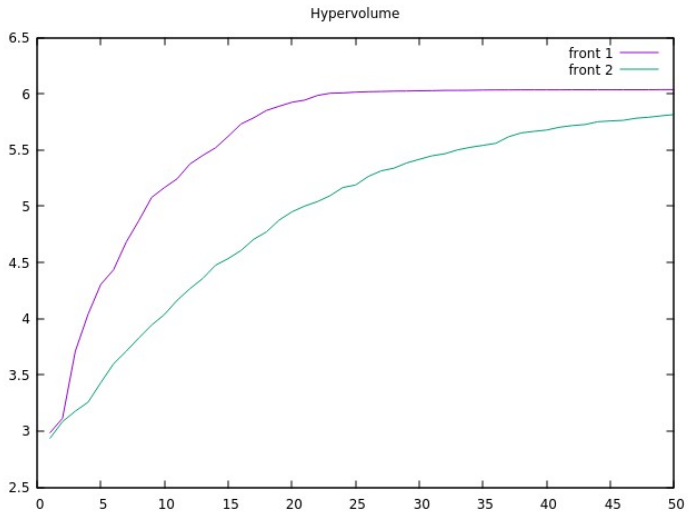


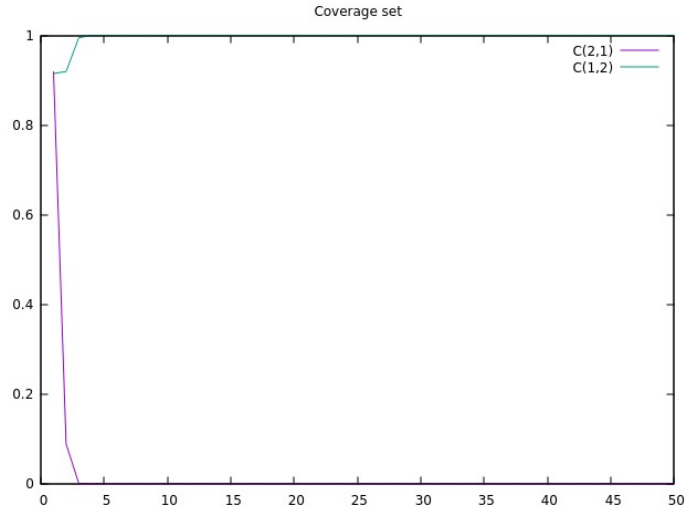


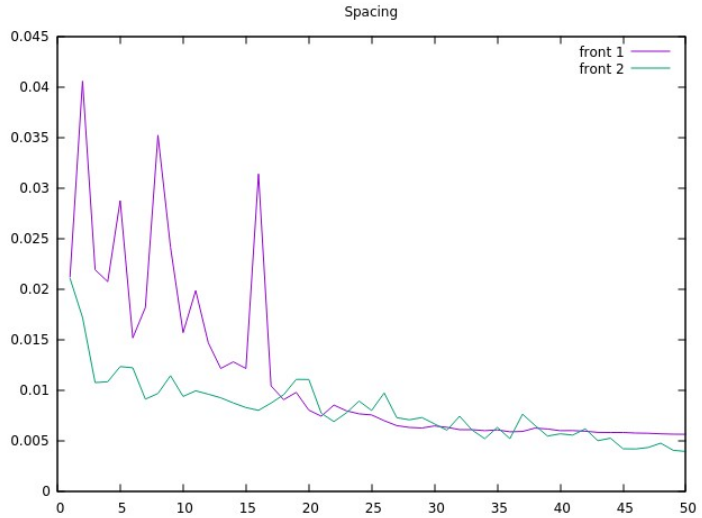


126 soluciones sin derrame

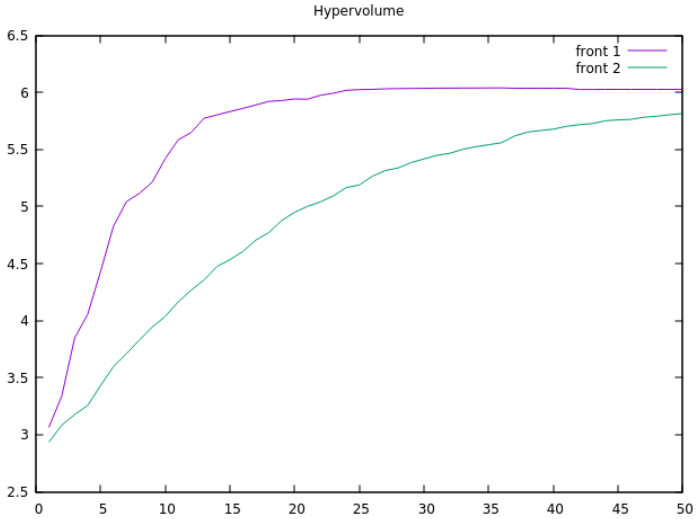
Métricas 1.1

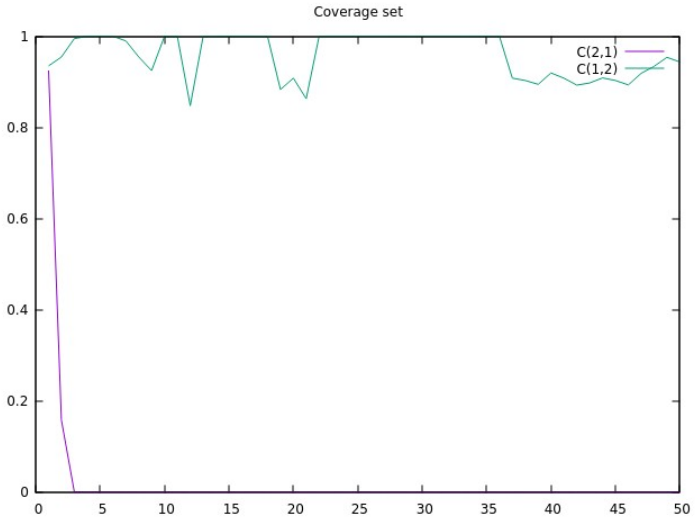


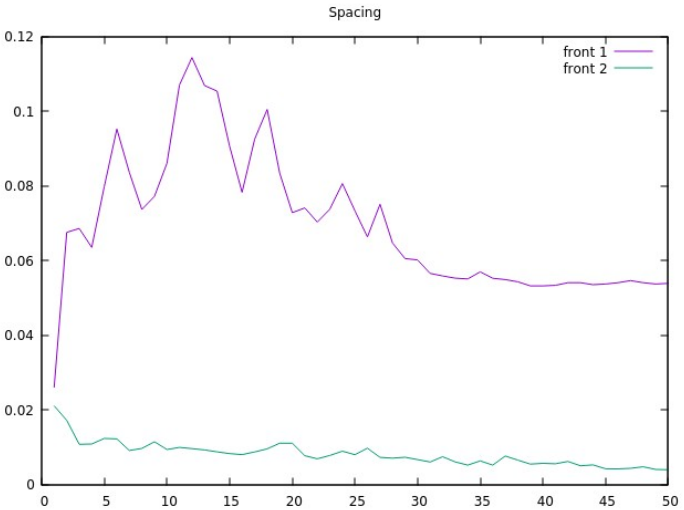




Métricas 1.0







Valores finales	Hipervolumen	Espaciado	Cobertura
Propio 1.1	6.0371824866	0.0056020710	C(1,2) 1
Propio 1.0	6.0266551565	0.0538532794	C(1,2) 0.94358974

Comparación estadística 1.1

Aunque contamos con menos cantidad de pruebas, son suficientes para apreciar el cambio con respecto a la versión 1.0. Procedemos a calcular la media de ambas versiones para poder ver cuanta mejoría hay con respecto a la anterior.

Como la cobertura en todos los casos de C(2, 1) es 0 podemos comparar directamente entre todas las coberturas de C(1, 2)

Media	Propio 1.0	Propio 1.1
Hipervolumen	5,98133144683333	6,0157533501
Espaciado	0,082022727066667	0,031429474166667
Cobertura	0,897526543333333	0,899831646666667

TOTAL

+0,6% de hipervolumen

+2,6 veces mejor espaciado

0,25% mejor cobertura

-45% en cantidad de soluciones resultantes

Interpretación final de resultados

Aunque los resultados estadísticos puedan parecer una pequeña mejoría la realidad es que mejora todas las características del algoritmo.

Lo principal que debemos notar es que esta mejora no es consistente para todo tipo de configuraciones en los parámetros y que su ventaja con respecto al anterior aumenta a medida que aumentamos el número de individuos y en menor medida también influye el aumento de generaciones.

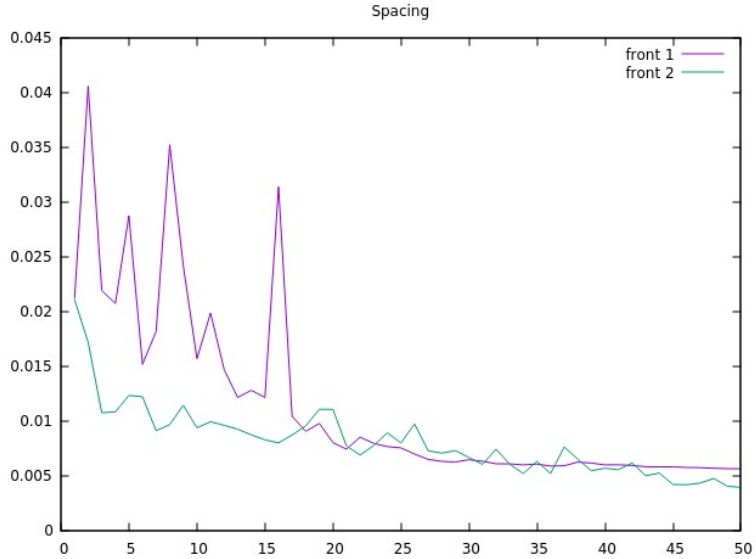
Procederemos al análisis individual de cada métrica:

Hipervolumen:

En lo que respecta al hipervolumen se consigue un aumento debido a encontrarse mejor espaciadas las soluciones y no haber mucha pérdida en cuanto avance del frente con respecto al anterior. Aún con la pérdida de soluciones nos podría interesar mas esta versión debido a que no solo no empeora el hipervolumen sino que lo mejora, pudiéndonos dar una mejor variedad para escoger.

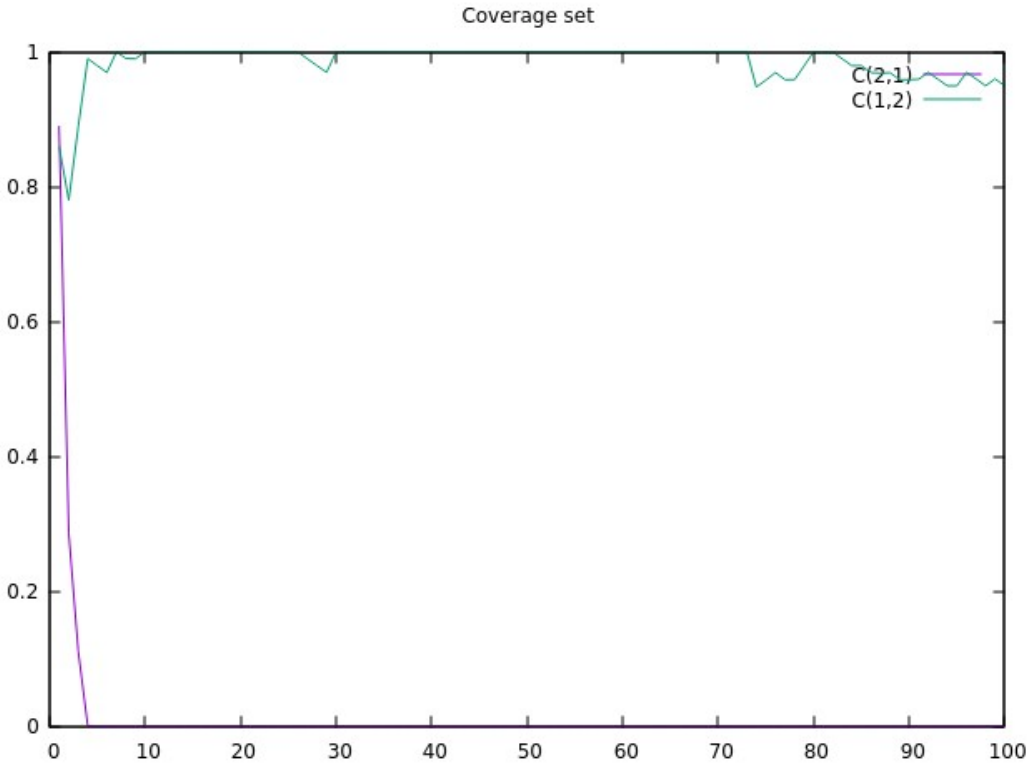
Espaciado:

Este apartado es la razón de ser de esta modificación ya que este era el punto débil de la versión 1.0. En esta nueva versión gracias al espaciado se puede observar el frente mas definido y que lo abarca mejor. El punto muy negativo es que perdemos gran parte de las soluciones pero conseguimos en los casos mas favorables llegar a prácticamente igualar el algoritmo de referencia en cuanto a espaciado se refiere.



Cobertura:

Un mejor espaciado solo nos otorga una mejor visión del frente sino que es capaz de dominar a mas soluciones del algoritmo de referencia, dándonos unos resultados muy buenos como por ejemplo en este caso de la ejecución 2:



Para finalizar, como conclusión, podemos sacar que la tendencia general es la continuar mejorando cuantos mas individuos tiene el algoritmo mejorado y que es una alternativa para nada despreciable y muy a tener en cuenta junto a la versión 1.0

Algoritmo con restricciones 2.0

Nuevo concepto

Se va a proceder en rasgos generales igual que con el anterior problema, minimizando las dos funciones objetivos, pero esta vez se tienen ciertas restricciones, por lo que no nos vale cualquier solución que esté dentro del espacio de objetivos. Estas restricciones nos las dan las funciones de restricción:

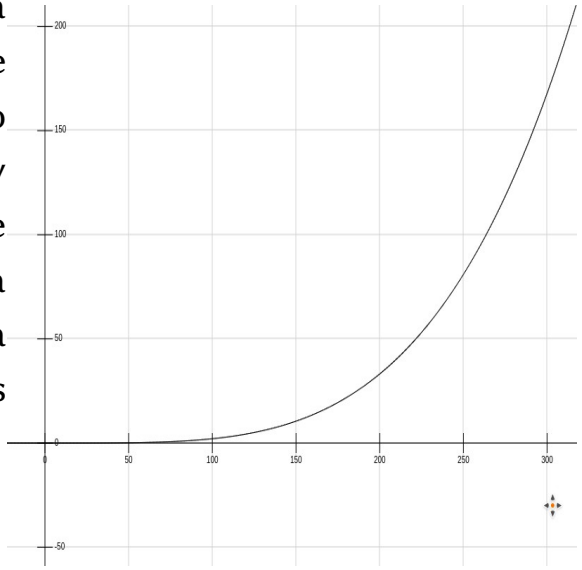
Las restricciones son:

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

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

Una solución completamente válida es la que no viola ninguna de las dos condiciones anteriores.

Para tener en cuenta estas restricciones se ha añadido una penalización a las soluciones que violen las restricciones en mayor o menor medida. Para conseguir un mayor avance de las soluciones y una mejor evolución del algoritmo al principio se ha permitido que las soluciones que no cumplan las restricciones puedan existir y luego ir aumentando la penalización sobre ellas hasta quedarnos con las válidas. La función que he elegido para representar la evolución del incremento de la penalización es la de x^4 , que tiene la siguiente forma⟶



Esta función irá acompañada de un valor de *intensidad* multiplicándole que nos permitirá desplazar la curva ascendente sobre el eje horizontal para cambiar el momento en el que comienza a aplicar las restricciones. X se corresponderá a la iteración actual. **Nuevo parámetro intensidad.**

$$f_{penalización} = (intensidad * x)^4$$

Cambios realizados

Para adaptar el algoritmo desarrollado en el apartado anterior se han realizado una serie de cambios. La ejecución, es decir, páginas *9 y 10* queda exactamente igual, por lo que no será necesario repetir como se ejecuta. Donde sí se han llevado a cabo cambios es en las funciones de **tchebycheff**(), **busquedaAobjetivo**() y en la limitación de valores después de mutar.

Función **solucion** **busquedaAobjetivo**() → Recibiendo una solución del espacio de búsqueda le aplica las funciones f1 y f2 y nos devuelve la **solucion** correspondiente en el espacio de objetivos. Se han cambiado las funciones f1 y f2 por las siguientes:

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

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

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

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

Función **double tchebycheff**() → Nos indica como de cerca nos encontramos de la solución ideal **z** tomando en cuenta una solución y un peso. Ahora además se encargará de penalizar esta valoración en función de si esa solución cumple con las restricciones o no.

Tchebycheff procederá de la siguiente manera:

- Aplica la nueva función **busquedaAobjetivo**() para trasladar la solución dentro del espacio de objetivos y almacenarla en una variable a parte.
- Se ejecutan las funciones de restricción para la solución en el espacio de búsqueda:

Las restricciones son:

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

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

- Se calcula la penalización en esa iteración, que es la suma del resultado de las restricciones en valor absoluto que hayan salido negativas multiplicado por la función penalización anteriormente descrita. Donde R es el vector resultante de las funciones de restricción:

$$\forall R<0 \quad penalización = \sum_{i=0}^{1} |R_i| * (intensidad * iteracion)^4$$

- Finalmente se le suma la penalización a ambos componentes de la solución objetivo y con esta nueva solución se calcula la función de thchebycheff y se devuelve el valor.

<u>Después de *mutar y cruzar*</u> → Se comprueba que la nueva solución no salga ahora de los límites:

El espacio de búsqueda es

$$0 \leq x_1 \leq 1$$

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

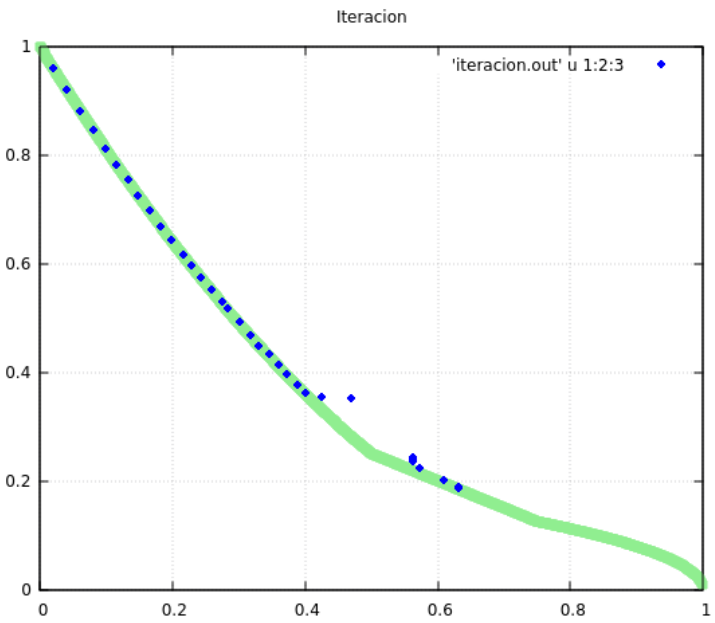
Comparativa versión 2.0 con algoritmo de referencia 4D

Observación sobre el parámetro de compensación angular

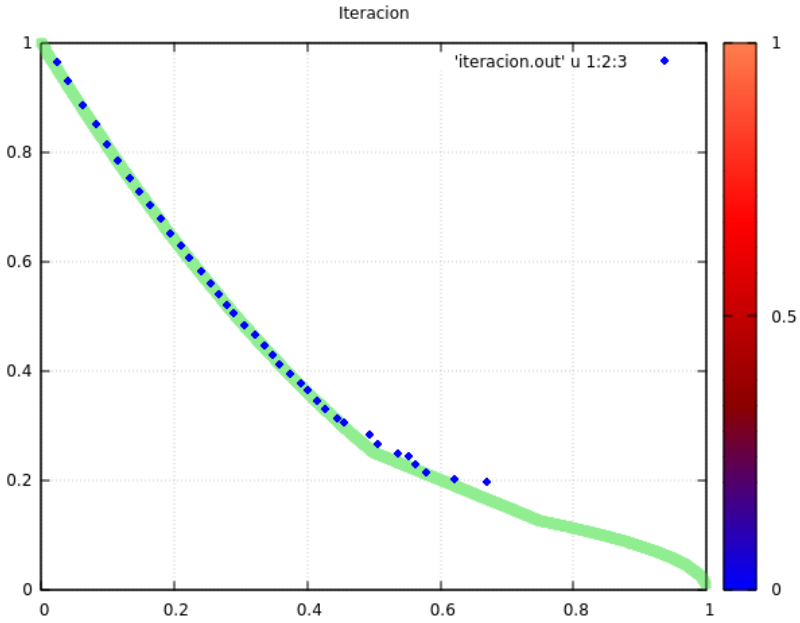
En esta nueva versión el parámetro compensación angular también resulta muy efectivo, de hecho, en este caso no produce derrame y nos permite desplazar las soluciones horizontalmente "forzando" que el algoritmo descubra nuevas soluciones en la zona derecha del frente ideal.

Un ejemplo con y sin el:

Compensación 0



Compensación 0.023



Para todas las pruebas se va a utilizar la compensación angular.

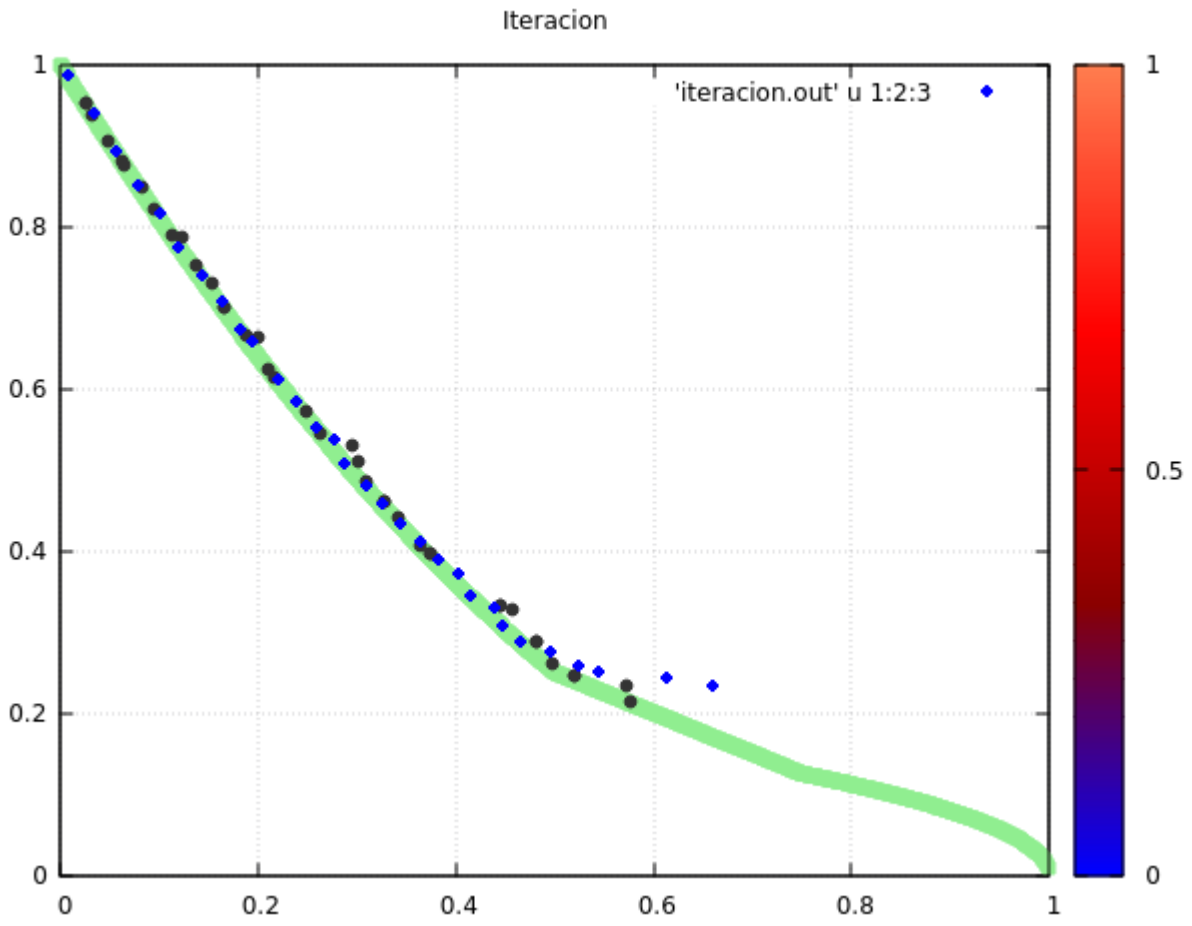
<u>Ejecución 1, comparación con semilla1:</u> 4000 evaluaciones

Ajustes comunes: 40 individuos | 4 dimensiones | 100 generaciones

Ajustes propios:F 0.8 | CR 0.45 | Vecindad 12% | semilla 0.3 |comp_angular 0.009 intensidad 0.012

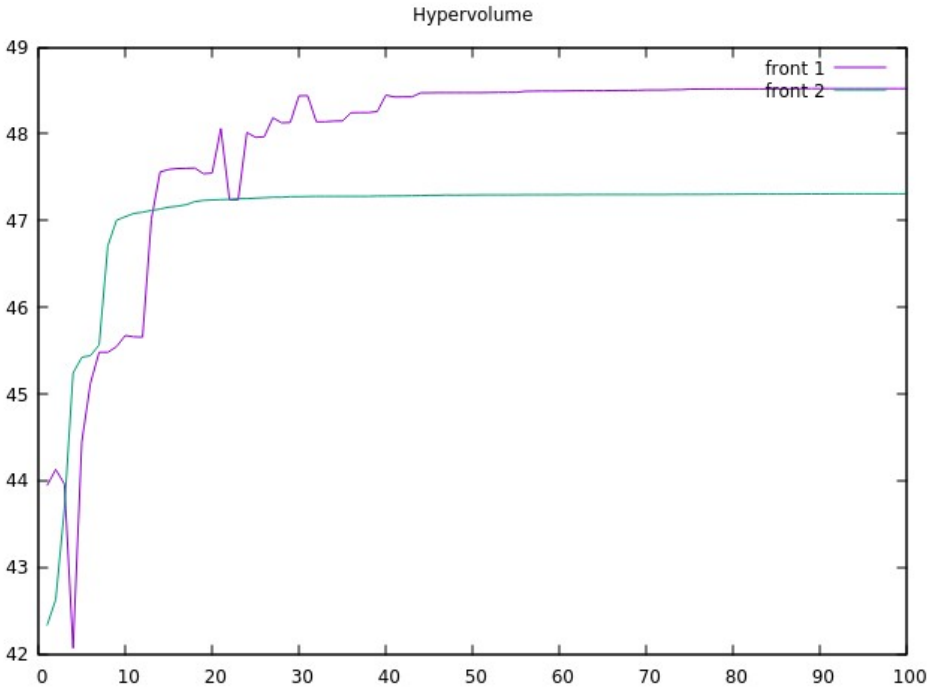
<u>Última generación de ambos algoritmos</u>

Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Se han representado con una degradación de azul a rojo cuanto viola esa solución las restricciones, donde rojo brillante es el máximo y azul es nada. En este caso todas violan 0. Se puede observar además que mi algoritmo aporta una mejor distribución y amplitud de las soluciones.

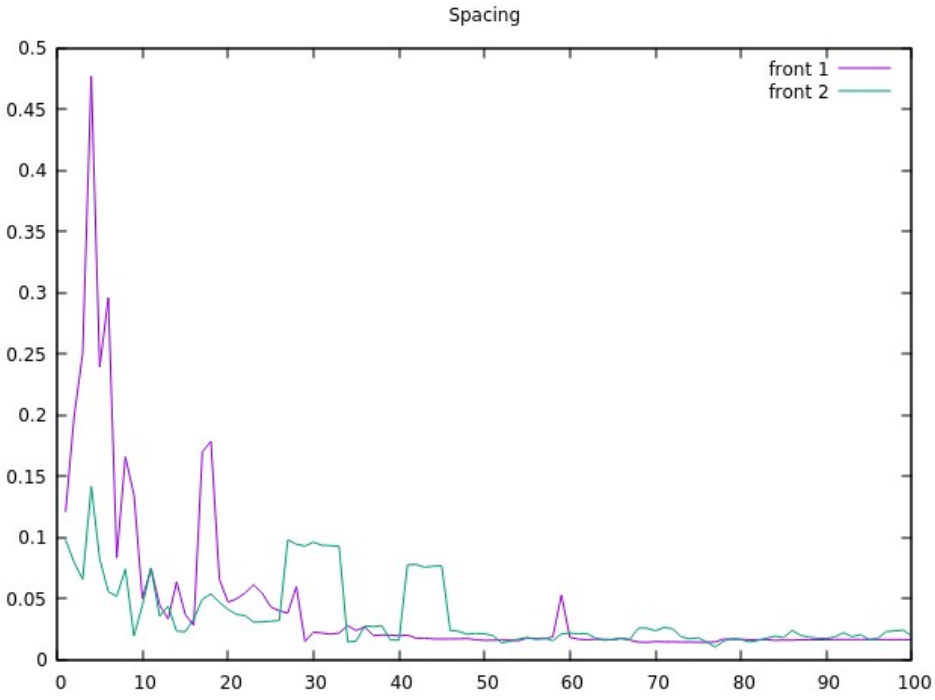
Hipervolumen



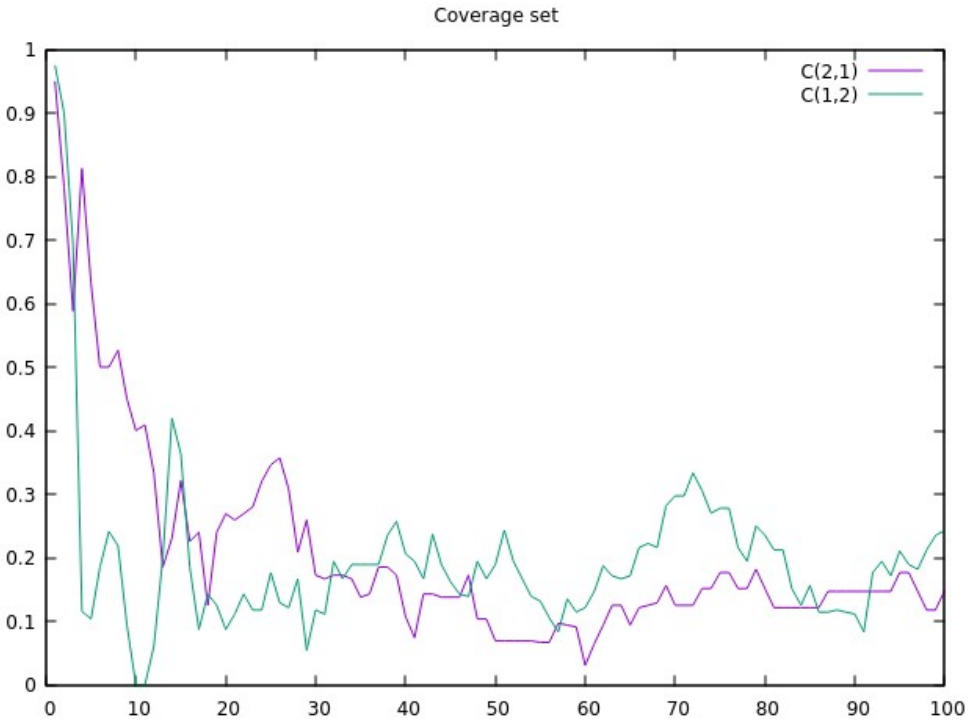
Se aprecia que al comienzo, el hipervolumen de referencia toma la delantera, pero mas tarde, aproximadamente en la iteración 15, comienza a superarle hasta finalizar con un valor significativamente mas alto que el otro. Todas las muestras tendrán el punto de referencia en (7,7)

Espaciado

Ambos algoritmos se encuentran muy próximos durante toda la ejecución obteniendo un valor muy similar al final.



Cobertura



Como se puede observar, en cobertura no podemos decantarnos por un claro vencedor ya que han estado muy reñidos. Pese a ello, mi algoritmo finalmente ha obtenido una cobertura un poco mas alta que el otro,

Valores finales	Hipervolumen	Espaciado	Cobertura
Propio 2.0	48.5232421048	0.0161889959	C(1,2) 0.24242424
Referencia	47.3096127635	0.0190498100	C(2,1) 0.14705882

Vamos a hacer a continuación pruebas con distinta semilla, número de iteraciones, generaciones y evaluaciones para comprobar si sigue venciendo ante otras situaciones y como responde a ellas.

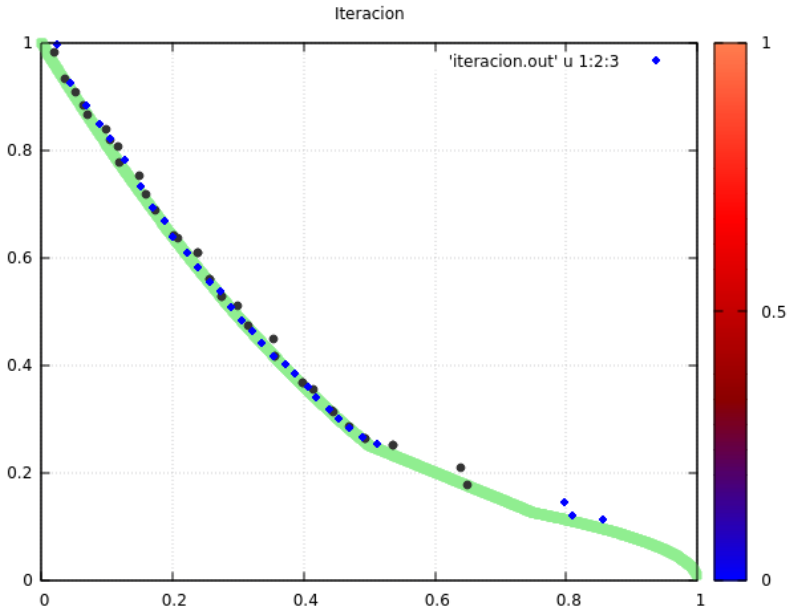
<u>Ejecución 2, comparación con semilla2:</u> 4000 evaluaciones

Ajustes comunes: 40 individuos | 4 dimensiones | 100 generaciones

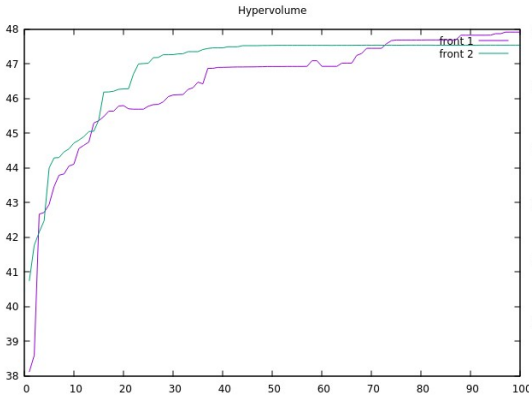
Ajustes propios:F 0.58 | CR 0.6 | Vecindad 18% | **semilla 9**|comp_angular 0.009 intensidad 0.012

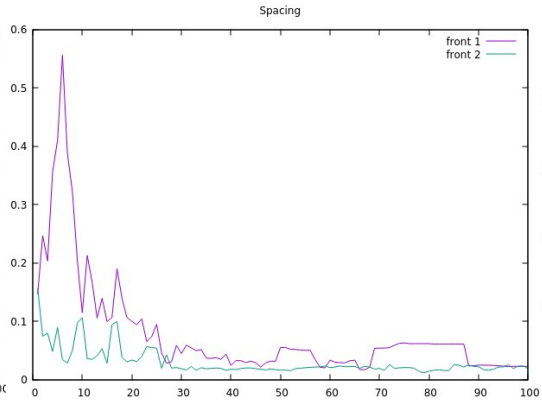
<u>Última generación de ambos algoritmos</u>

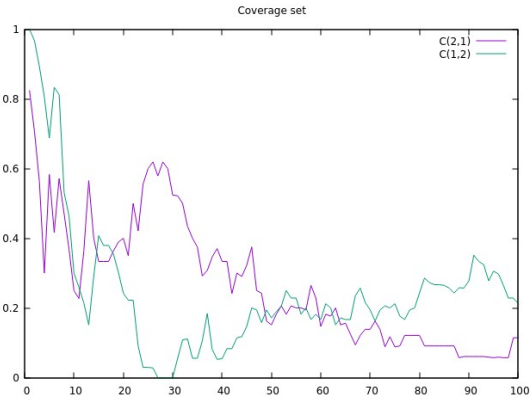
Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Mismos parámetros y nueva semilla para demostrar que sigue ganando incluso con otras condiciones iniciales.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	47.9121141928	0.0225701765	C(1,2) 0.21212121
Referencia	47.5345495227	0.0188317546	C(2,1) 0.11428571

<u>Ejecución 3, comparación con semilla1:</u> 4000 evaluaciones

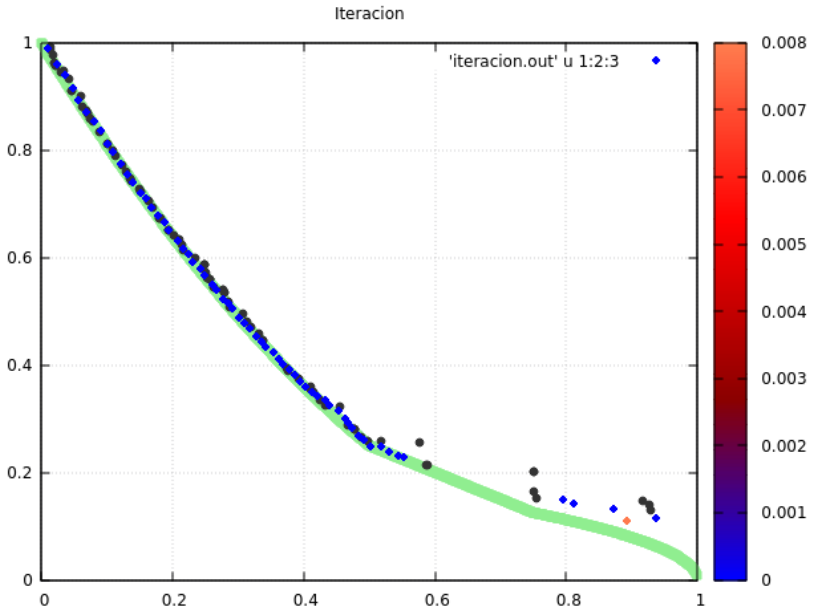
Ajustes comunes: 80 individuos | 4 dimensiones | **50 generaciones**

Ajustes propios:F 0.8 | **CR 0.8** | Vecindad 10% | semilla 0.3| **comp_angular 0.0**

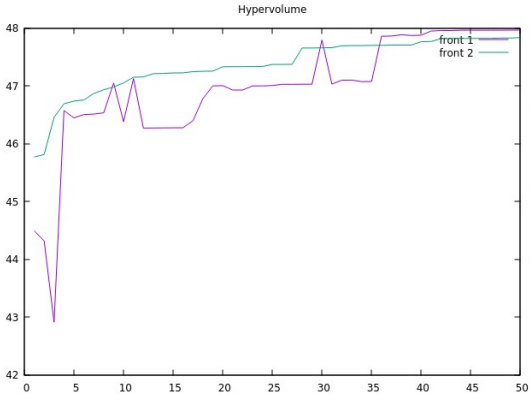
intensidad 0.025

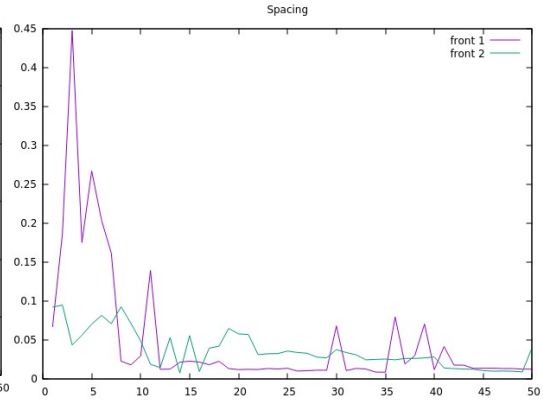
<u>Última generación de ambos algoritmos</u>

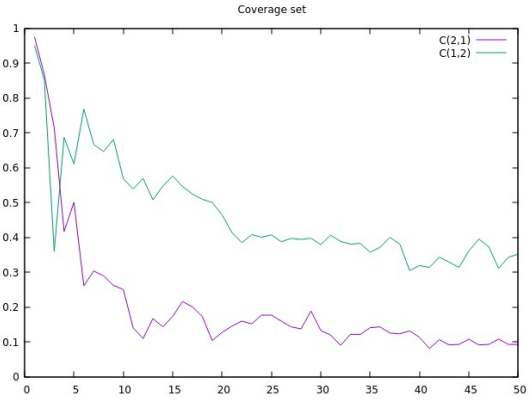
Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Nuevos parámetros, se obtiene una superioridad en todo.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	47.9639840474	0.0119562486	C(1,2) 0.35135135
Referencia	47.8348434089	0.0393064744	C(2,1) 0.09230769

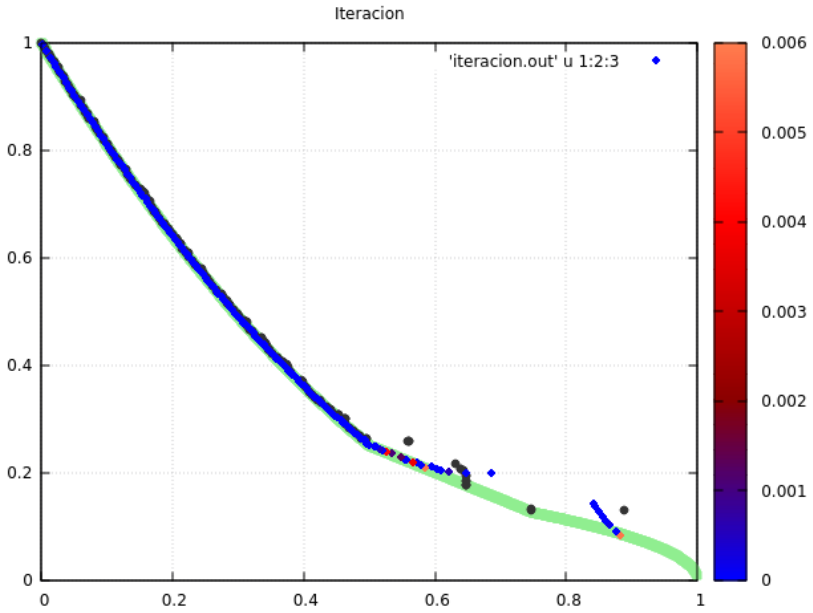
<u>Ejecución 4, comparación con semilla1:</u> 10000 evaluaciones

Ajustes comunes: 200 individuos | 4 dimensiones | 50 generaciones

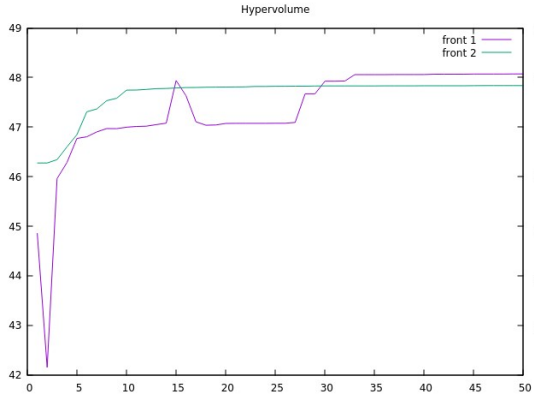
Ajustes propios:F 0.848 | CR 0.8 | **Vecindad 8%** | semilla 0.3| comp_angular 0.0 intensidad 0.02

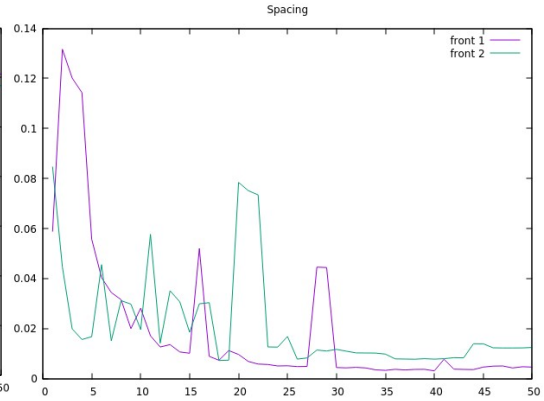
<u>Última generación de ambos algoritmos</u>

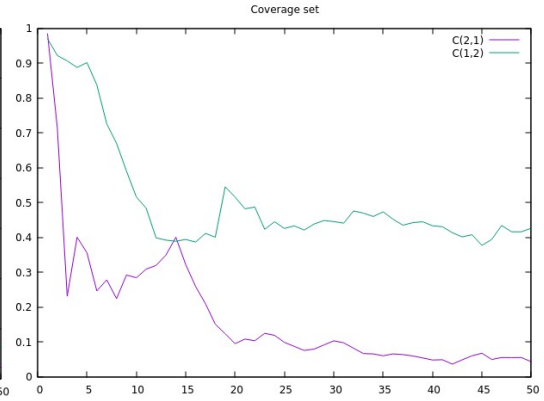
Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Cantidad alta de individuos, se obtiene también una superioridad en todo.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	48.0728759277	0.0045763719	C(1,2) 0.42592592
Referencia	47.8389845791	0.0124165416	C(2,1) 0.04268292

<u>Ejecución 5, comparación con semilla1:</u> 10000 evaluaciones

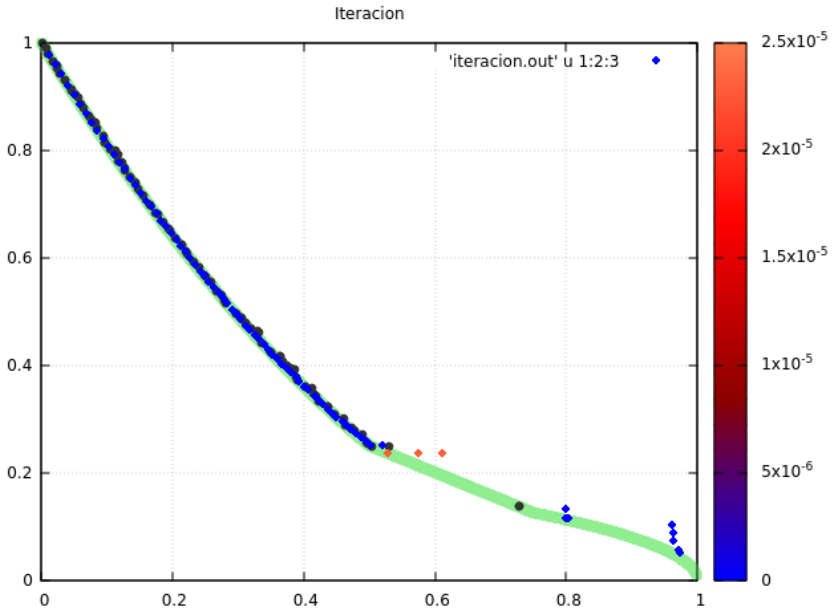
Ajustes comunes: 100 individuos | 4 dimensiones | **100 generaciones**

Ajustes propios:F 0.848 | CR 0.8|Vecindad 8%|semilla 0.3|**comp_angular 0.0025**

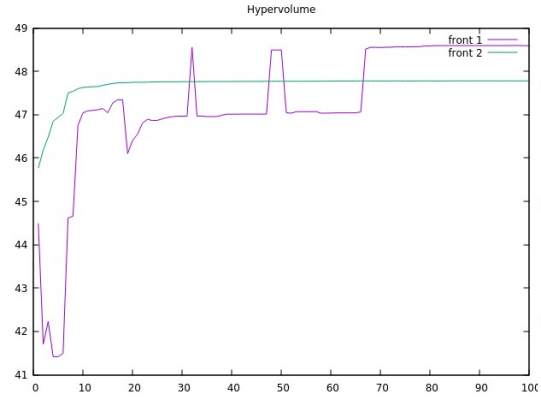
intensidad 0.013

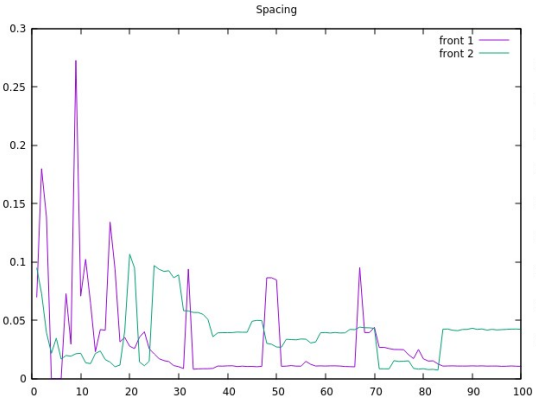
<u>Última generación de ambos algoritmos</u>

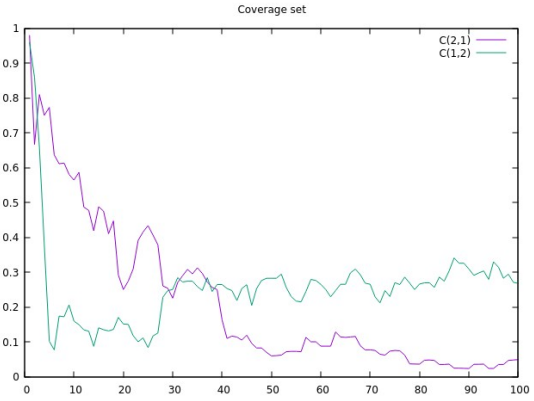
Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Igual cantidad tanto de individuos como de generaciones.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	48.5921874434	0.0104549424	C(1,2) 0.26744186
Referencia	47.7803134891	0.0420950940	C(2,1) 0.04878048

Comparación estadística 2.0 4D

Una vez obtenidos los resultados vamos a proceder a comprobar con datos numéricos cómo de buenos son estos nuevos datos. Se tomará como valor representativo de cada conjunto de datos la media debido a que no tenemos valores muy dispares entre ellos.

Media	Referencia	Propio 2.0
Hipervolumen	47,65966075	48,21288074
Espaciado	0,09709159	0,01314935
Cobertura	0,08902312	0,29985292

TOTAL

+1,01% de hipervolumen

+7,38 veces mejor espaciado

337% mejor cobertura

Interpretación final de resultados

A la vista de los resultados, tanto gráficos como estadísticos se puede llegar a la conclusión de que el nuevo algoritmo implementado es significativamente mejor que el de referencia.

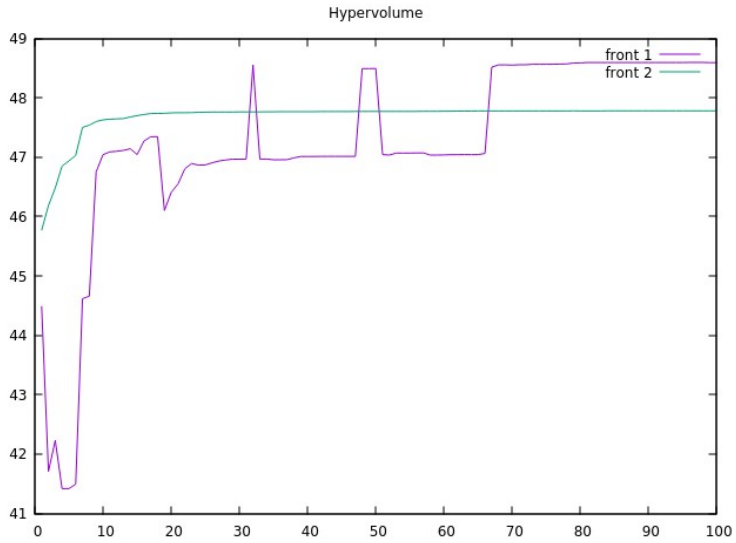
En relación a los parámetros, con respecto a la anterior implementación sin restricciones podemos ver un aumento muy significativo tanto en el parámetro de cruce como en el de mutación. Esto es debido a que al ser un espacio de búsqueda con restricciones se presenta mas accidentado para las soluciones y son muy dadas a permanecer en mínimos locales, impidiendo así el avance del mismo. Para contrarrestar este efecto negativo sobre mi ejecución se ha tenido que aumentar la distancia de los 'saltos' que efectúan las soluciones, es decir, nuevas soluciones más diferenciadas de sus padres. Además, gracias a el nuevo parámetro de compensación angular se ha podido 'incentivar' un poco a las soluciones a que exploren mejor la parte del frente mas difícil, que es la zona de la derecha.

Con respecto a la vecindad, por la misma razón de distinguir más a las nuevas soluciones de los padres y de poder explorar mejor el espacio, se ha reducido en todos los casos a un valor de aproximadamente un 10%, un valor que les permite mejorar localmente sin impedir demasiado que se generen soluciones mejor diferenciadas.

Todas estas medidas, unidas a un aumento gradual de la penalización, permiten una mayor libertad de explorar al comienzo de la ejecución y son la clave para obtener un algoritmo que es capaz de avanzar muy rápido sin quedar atrapado y pudiendo afinar en su cercanía al frente.

A continuación vamos a revisar y analizar mas en profundidad las mejores gráficas de las pruebas.

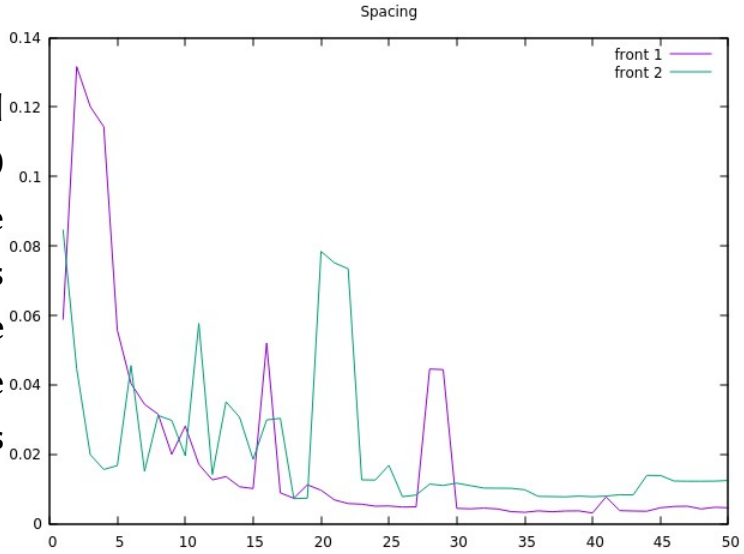
Hipervolumen



El mayor valor de hipervolumen se ha conseguido en esta ejecución con 100 individuos y 100 generaciones.

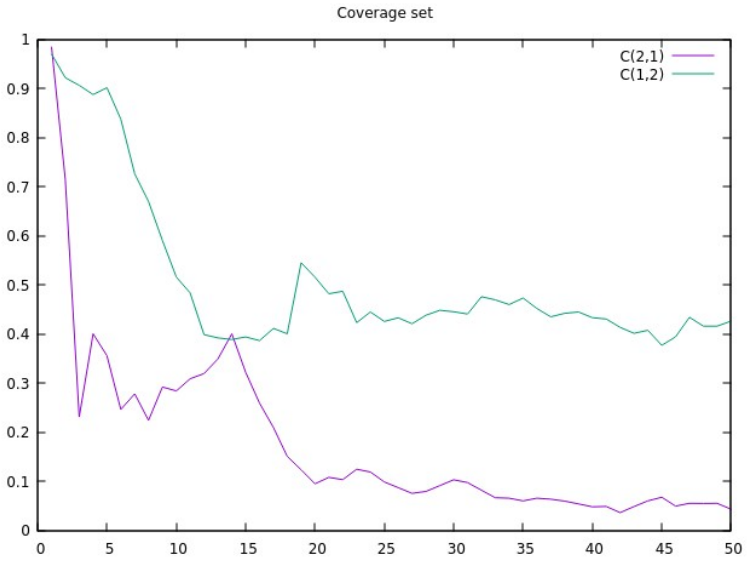
Como tendencia general podemos deducir que nuestro algoritmo es capaz de conseguir un mayor hipervolumen con unos valores equilibrados tanto de individuos como de generaciones.

Espaciado



Esta ejecución es la que se lleva el espaciado mas bajo, 200 individuos y 50 generaciones. Esto ha ocurrido porque al contar con mas soluciones podemos distribuirlas mas uniformemente sobre el frente. A cada solución le 'corresponde' cubrir un espacio mas pequeño del frente.

Cobertura



A diferencia de la versión sin restricciones en este apartado no hemos obtenido una diferencia tan clara, las soluciones de referencia se encontraban mas ajustadas al frente y presentaban una evolución mas rápida en comparación a la versión sin restricciones. Donde se ha obtenido una mejor cobertura es en este caso de 200 individuos y 50 generaciones, que debido a nuestra mejor capacidad y rapidez para ajustarnos al frente hemos obtenido una ventaja considerable.

Comparativa versión 2.0 con algoritmo de referencia 16D

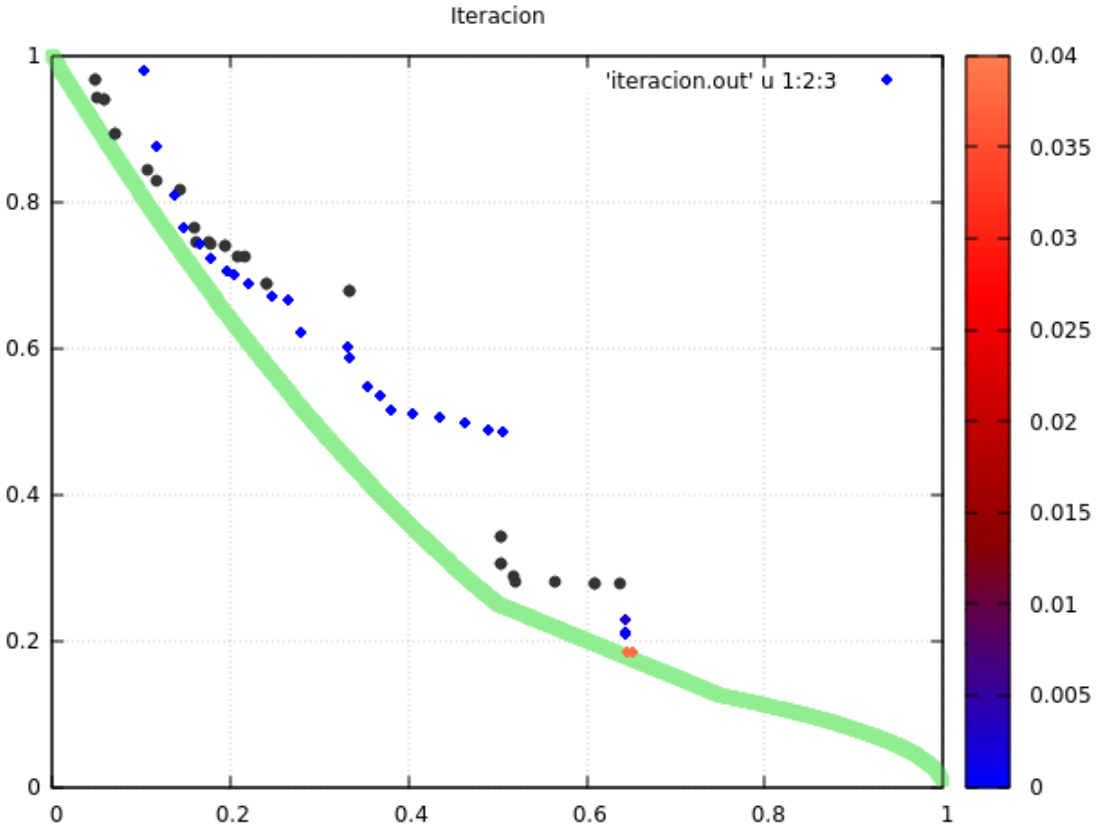
<u>Ejecución 1, comparación con semilla1:</u> 4000 evaluaciones

Ajustes comunes: 40 individuos | 4 dimensiones | 100 generaciones

Ajustes propios:F 0.84|CR 0.5339|Vecindad 8% | semilla 0.3 |comp_angular 0.015 intensidad 0.008

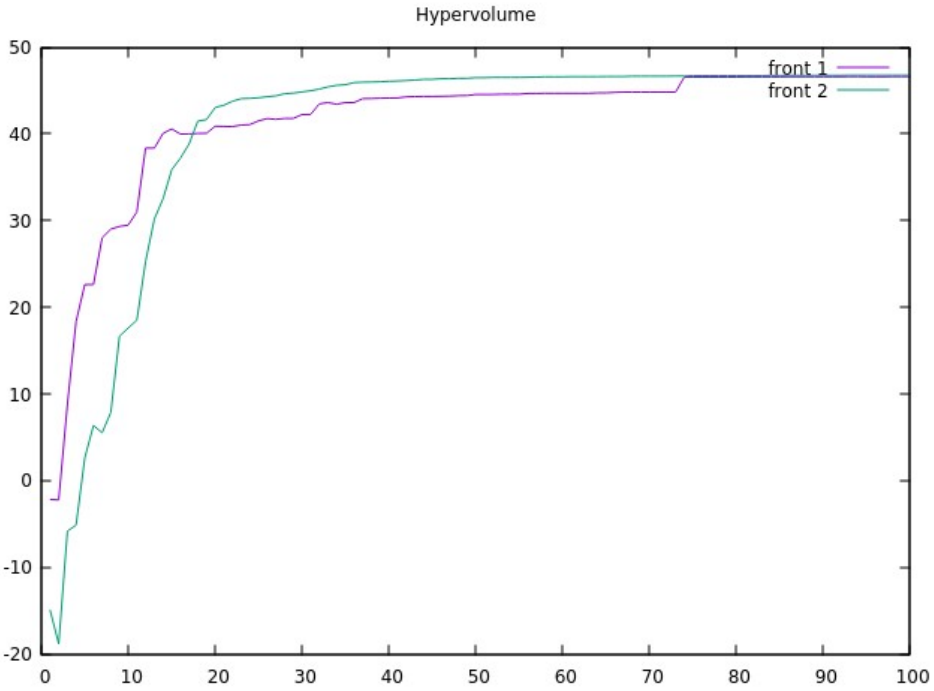
<u>Última generación de ambos algoritmos</u>

Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Para poder superar esta versión de referencia ha sido necesario un ajuste mas preciso de los parámetros. Se ha conseguido un frente bastante amplio aunque no todo lo cerca que debería del frente.

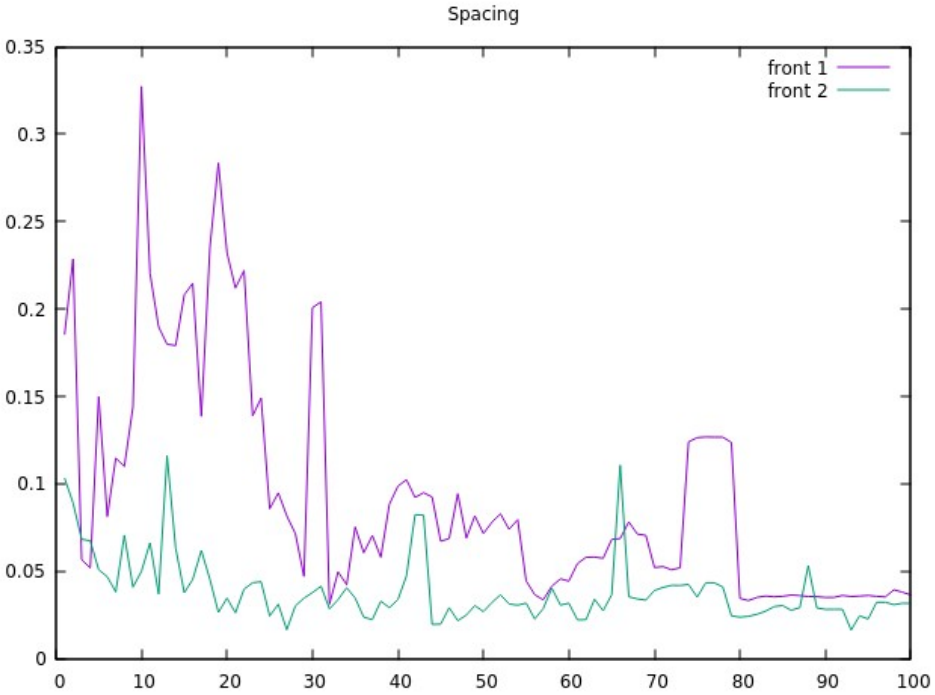
Hipervolumen



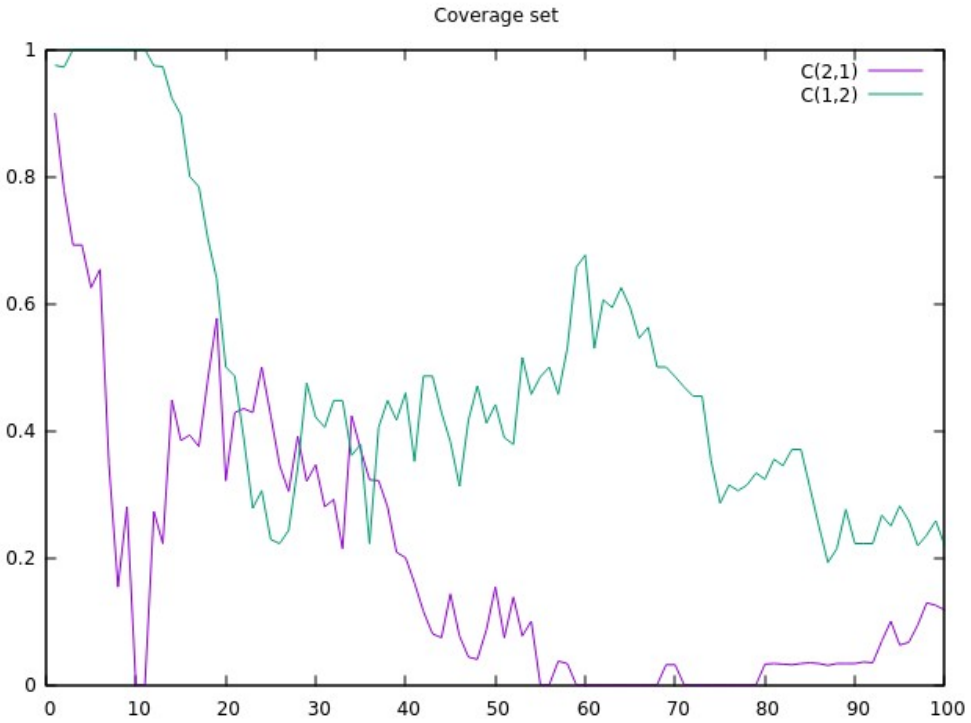
En esta ocasión aunque al principio se encontrara por delante ambos acabaron con un valor muy similar. Todas las muestras tendrán el punto de referencia en (7,7)

Espaciado

Podemos apreciar un espaciado en mi algoritmo bastante irregular mientras avanza, pero que mejora en cuanto encuentra un mínimo local.



Cobertura



En cobertura conseguimos desmarcarnos ligeramente del de referencia.

Valores finales	Hipervolumen	Espaciado	Cobertura
Propio 2.0	46.6095301396	0.0362904111	C(1,2) 0.21875000
Referencia	46.7868749268	0.0315687234	C(2,1) 0.11764705

Vamos a hacer a continuación pruebas con distinta semilla, número de iteraciones, generaciones y evaluaciones para comprobar si sigue venciendo ante otras situaciones y como responde a ellas.

<u>Ejecución 2, comparación con semilla1:</u> 4000 evaluaciones

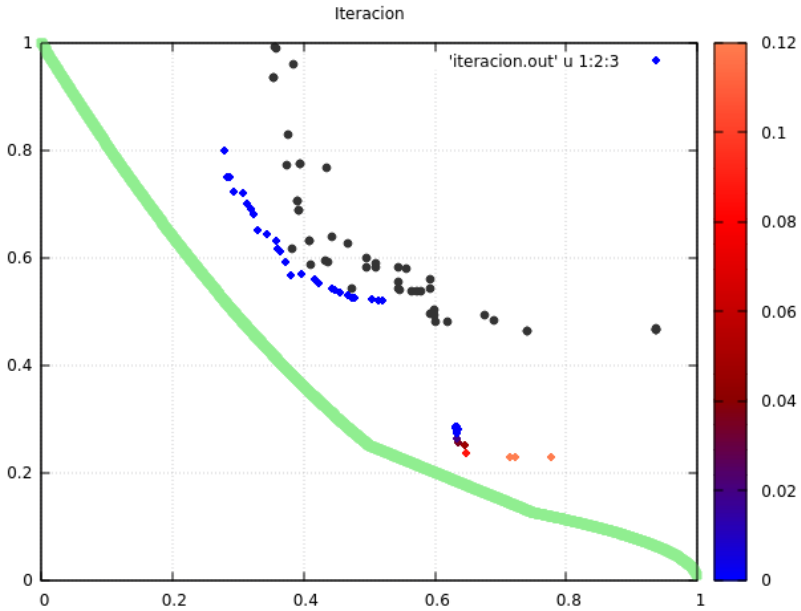
Ajustes comunes: 100 individuos | 4 dimensiones | **40 generaciones**

Ajustes propios:F 0.75|**CR 0.65**|Vecindad 8%|semilla 0.3|**comp_angular 0.0201**

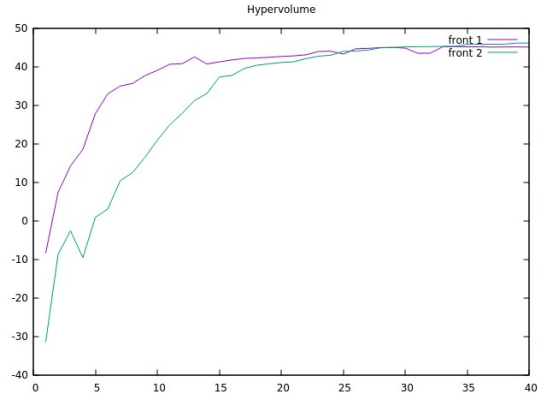
intensidad 0.018

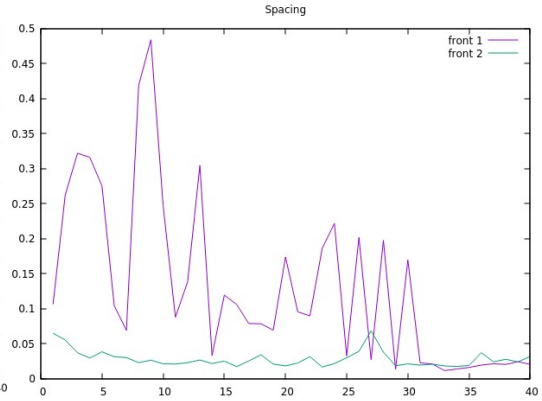
<u>Última generación de ambos algoritmos</u>

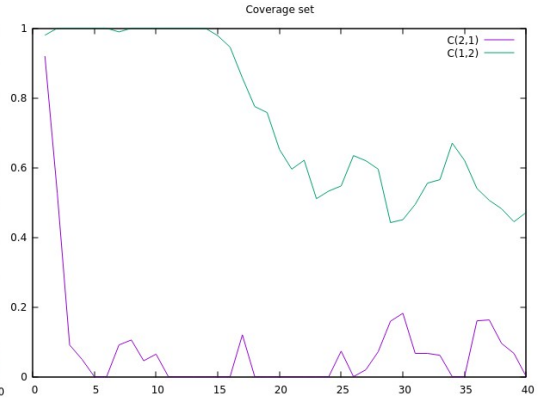
Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal



Podemos apreciar dos hileras de soluciones que cubren a las de referencia.







<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	45.0658850930	0.0203043852	C(1,2) 0.47126436
Referencia	46.1118184570	0.0313949282	C(2,1) 0.00000000

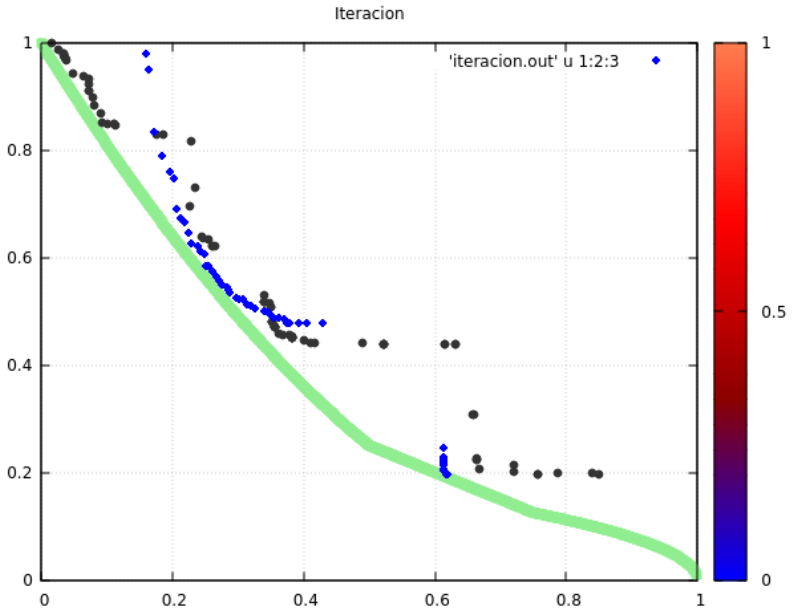
<u>Ejecución 3, comparación con semilla1:</u> 10000 evaluaciones

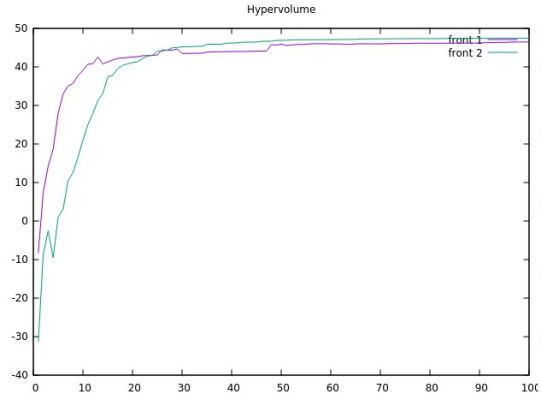
Ajustes comunes: 100 individuos | 4 dimensiones | **100 generaciones**

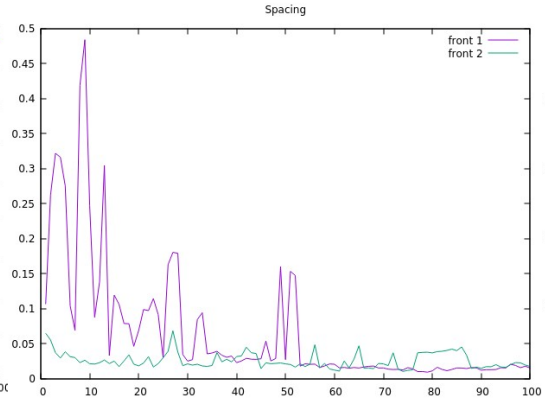
Ajustes propios:F 0.75|CR 0.65|Vecindad 8%|semilla 0.3|comp_angular 0.0201 intensidad 0.0115

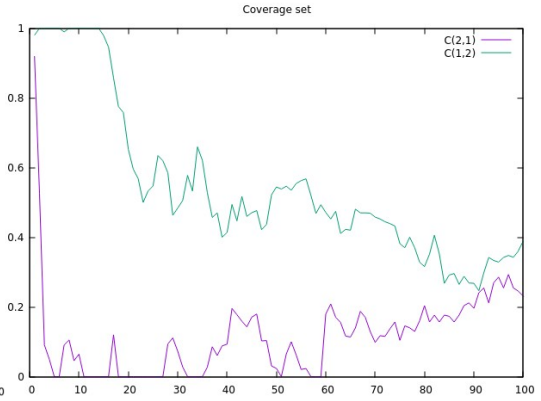
<u>Última generación de ambos algoritmos</u>

Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal









<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	46.3797343079	0.0150017806	C(1,2) 0.38805970
Referencia	47.3298861963	0.0171999264	C(2,1) 0.23076923

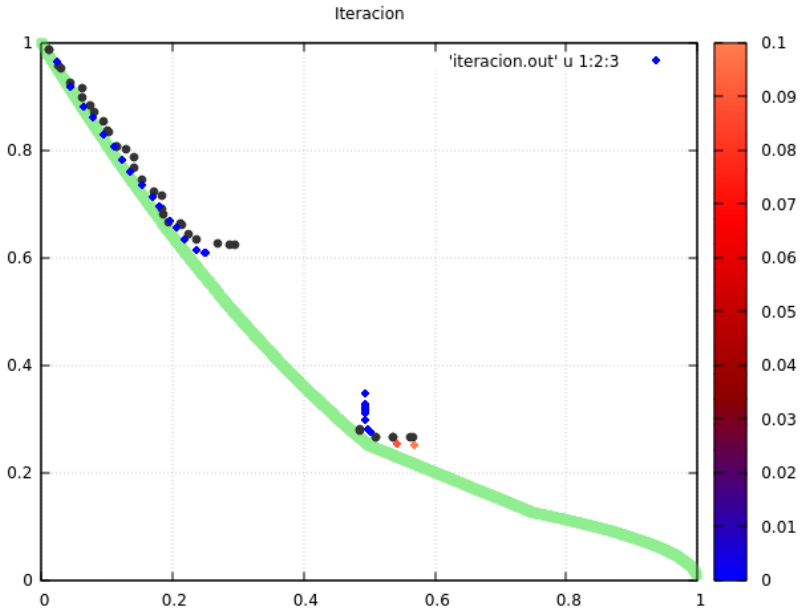
<u>Ejecución 4, comparación con semilla1:</u> 10000 evaluaciones

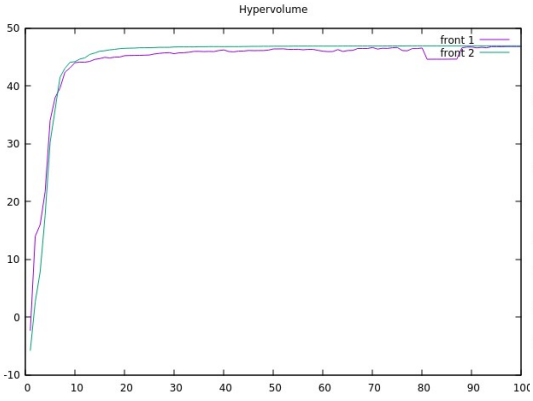
Ajustes comunes: 40 individuos | 4 dimensiones | **250 generaciones**

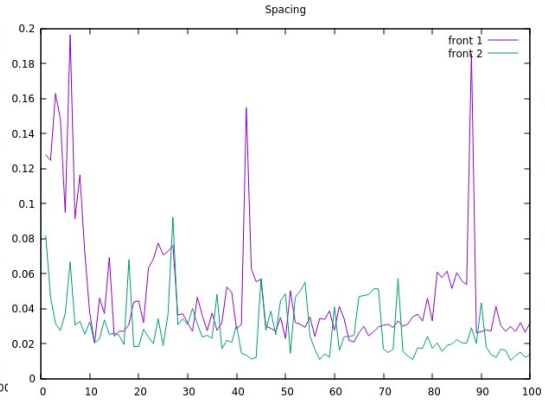
Ajustes propios:F 0.75|**CR 0.479**|Vecindad 8%|semilla 0.3|comp_angular 0.0403 intensidad 0.004

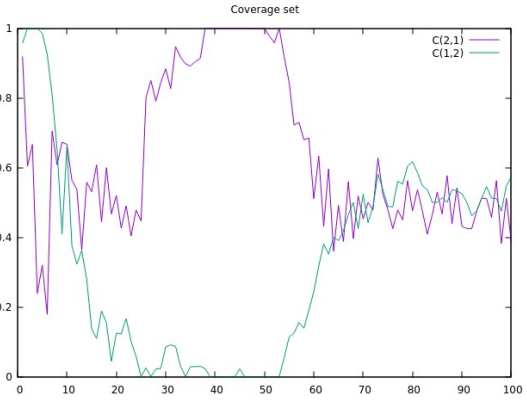
<u>Última generación de ambos algoritmos</u>

Gris oscuro: Algoritmo de referencia **Azul-Rojo:** Propio 2.0 **Verde:** Frente ideal









<u>Valores finales</u>	Hipervolumen	Espaciado	Cobertura
Propio 2.0	46.8020323012	0.0317891175	C(1,2) 0.57500000
Referencia	46.9187741182	0.0136744510	C(2,1) 0.38709677

Comparación estadística 2.0 16D

Vamos a continuar cuantificando la diferencia entre ambos algoritmos.

Media	Referencia	Propio 2.0
Hipervolumen	46,78683842	46,21429546
Espaciado	0,02345951	0,02584642
Cobertura	0,18387826	0,41326852

TOTAL

-1,2% de hipervolumen

-10% de espaciado

224% mejor cobertura

Interpretación final de resultados

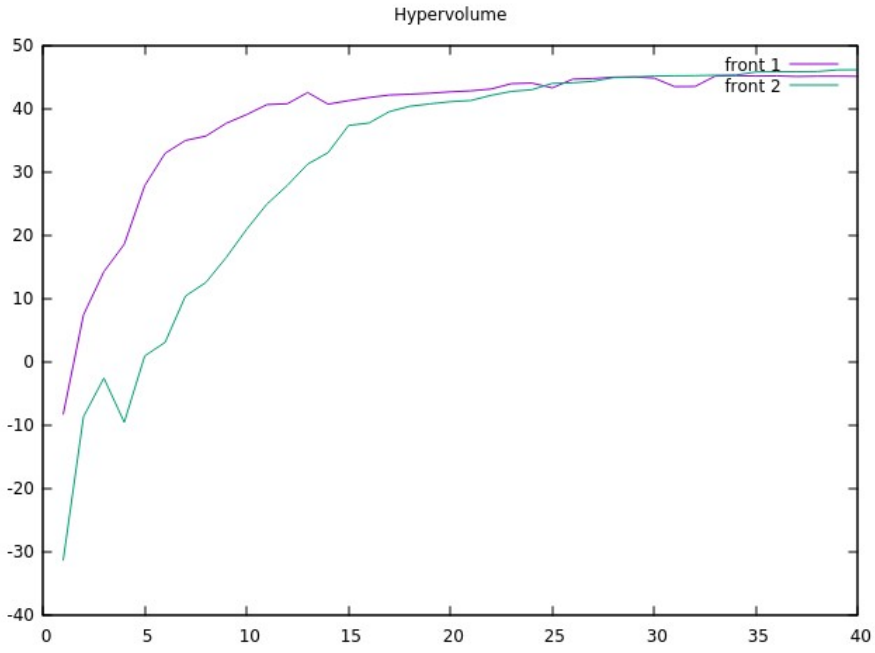
Como hemos podido observar, cuando se trata de 16 dimensiones con restricciones la competición está bastante mas reñida.

A mi algoritmo con tal cantidad de dimensiones con restricciones le resulta fácil quedarse estancado en una zona y es necesario un ajuste muy preciso de los parámetros de entrada para que se consigan los saltos adecuados que le permitan avanzar.

Como se ha visto en la comparación estadística tenemos una pequeña pérdida tanto en hipervolumen como en espaciado pero a cambio tenemos una gran mejora en cobertura, lo que quiere decir que nuestro algoritmo consigue mejorar en gran medida a las soluciones del de referencia aunque con una disposición un poco peor.

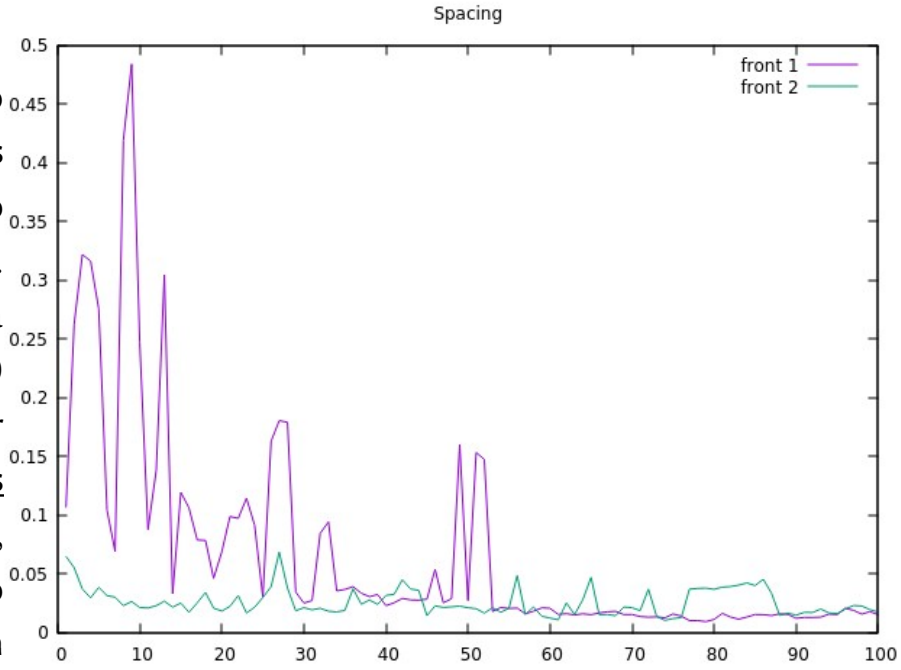
Vamos a proceder a analizar las gráficas mas destacables de cada métrica.

Hipervolumen



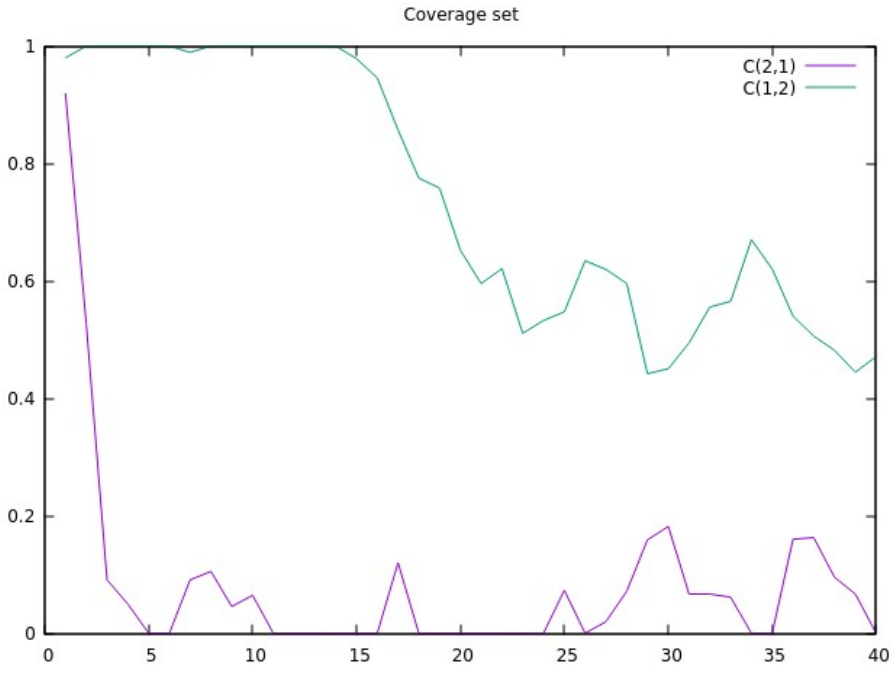
La gráfica que más puede llamar la atención es esta de 100 individuos 40 generaciones, que consigue un mejor valor en menos tiempo, pero que aun así pierde en valor final. Como tendencia general podría destacar que este parámetro tiene un gran crecimiento inicial hasta quedarse estancado en un valor hasta el final de la ejecución en ambos algoritmos.

Espaciado



En esta métrica mi algoritmo no tiene demasiados problemas y depende mucho de cada ejecución su resultado. Como se puede ver en esta gráfica de 100 individuos 100 generaciones el valor permanece inestable mientras el frente continúa avanzando, cuando se queda parado comienza a mejorar en espaciado.

Cobertura



Aquí por lo general mi algoritmo ha mejorado porque ha conseguido colocar mejor sus soluciones por delante de las de referencia.

En esta gráfica de 100 individuos y 40 generaciones ha dominado claramente al otro, mi frente estaba por delante de todas las soluciones del de referencia.

Conclusiones generales

El nuevo algoritmo a grandes rasgos ha tenido un resultado muy positivo, mejorando mucho en todos los aspectos al de referencia. Cuando no tenía restricciones ha podido avanzar sin problemas y su evolución era rápida y precisa. La cosa comienza a complicarse para él cuando se trata con restricciones debido a que depende mucho de la vecindad, y en un espacio donde existen irregularidades es muy importante encontrar soluciones lo suficientemente diferenciadas como para que pueda salir de los mínimos locales. La vecindad en estos casos debía de ser mínima, y esto es algo que lo perjudica mucho al no poder beneficiarse de las soluciones de sus vecinos y no obtener vecinos mas distantes. En caso de haberla dejado alta tiende mucho a quedarse atascado, como ya hemos mencionado, en mínimos locales.

Si tenemos en cuenta los resultados estadísticos podríamos decir que ha sido un éxito, por lo general evoluciona rápido, alcanza hipervolúmenes mas altos, mejores coberturas y un espaciado mejorado.

Opinión personal

A grandes rasgos me ha parecido un trabajo muy interesante, me ha permitido indagar un poco mas en un lenguaje como es c++ y en la implementación de algoritmos. Ponerme directamente a elaborar un algoritmo de este tipo me ha descubierto cosas que no habría sido capaz de ver en otras asignaturas en las que lo ya hecho. Muchas gracias por su tiempo.

Antonio Vázquez Pérez, a 19 de abril de 2020.