Modelos Bioinspirados y Heurísticas de Búsqueda

Práctica 2

Daniel Pérez Rodríguez

Índice

Introducción

El objetivo de la práctica es estudiar y desarrollar los siguientes algoritmos:

- Algoritmo genético básico
- Algoritmo genético CHC
- Algoritmo genético multimodal
- Algoritmo de búsqueda VNS

El código desarrollado deberá optimizar el uso de una red de bicicletas y seguidamente los resultados obtenidos por cada algoritmo deberán ser comparados con la solución obtenida por la búsqueda local implementada en la práctica 1.

Algoritmos

Para el estudio de los distintos parámetros que compone cada algoritmo, se han utilizado 5 semillas. Siendo las siguiente: **56428042**, **27886648**, **173386297**, **78746995**, **95206404**. Siendo semilla 1, 2, 3, 4 y 5 respectivamente (los resultados se encuentran en ‘./study’)

Algoritmo genético básico

Un algoritmo genético básico consiste en que a partir de una población de individuos inicial. Se seleccionan a los mejores individuos de la población para que los descendientes hereden sus características y así obtengan mejores resultados. El algoritmo esta compuesto por múltiples parámetros que pasaremos a comentar a continuación: **tipo, cruce, mutación, reemplazo, fitness y población inicial .**

Tipo

Emplearemos un modelo de **algoritmo genético estacionario**. En el mismo, se escogen a varios padres de la población y se cruzan para crear nuevos descendientes. Una vez cruzados, los hacemos mutar. Este modelo, nos permite generar una presión selectiva alta consiguiendo una convergencia más rápida que el modelo generacional.

Cruce

Para cruzar a los mejores individuos de la población realizaremos una **selección por torneo**. De forma que realizaremos una preselección de candidatos de forma aleatoria, siendo esta cantidad igual al 30% de la población (k=population*0.3). Y a partir de la misma, seleccionaremos a los 4 mejores individuos para que se reproduzcan (L=4). Formaremos 2 parejas, y se combinarán mezclando dos partes de cada uno de los cromosoma de los padres. Ejemplo para cromosomas de 6 genes:

Padre A: [1,1,1,1,1,1] – Padre B: [2,2,2,2,2,2] | Punto de corte [1]

A partir del punto de corte dividimos

Hijo A: [1,1,2,2,2,2] – Hijo B: [2,2,1,1,1,1]

Mutación

Con los nuevos cromosomas creados a partir de los padres, debemos mutar sus genes. Tenemos que mutar entre un 5% y 20% de los genes totales. Para ello, cada vez que debamos seleccionar que genes mutar. Elegiremos de forma aleatoria una cantidad entre el 5% y el 20% del total. Después, una vez más, de forma aleatoria seleccionaremos los genes a mutar. Para variar el número de slots de cada gen, se ha empleado una campana de gauss, con media sobre el valor del gen y una desviación típica igual a 1. De esta forma tendremos una configuración semejante al operador de vecindad en búsqueda local de la práctica 1.

Reemplazo

Al igual que para seleccionar que individuos cruzar, para elegir que cromosomas reemplazar por los nuevos hijos utilizaremos **selección por torneo**. De forma que realizaremos una preselección de candidatos de forma aleatoria, siendo esta cantidad igual al 30% de la población (k=population*0.3). Y a partir de la misma, seleccionaremos a los 4 peores individuos para intercambiarlos por nuestros 4 hijos.

Fitness

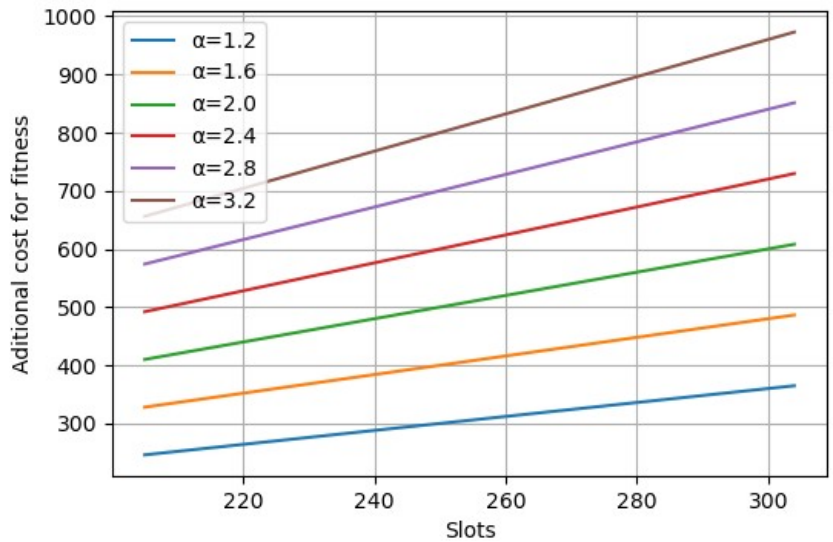
La función de fitness ha sido actualizada con respecto a la práctica 1. En la misma, nuestra función devolvía el número de kilómetros por las personas a partir de una solución. Ahora, además se le sumará un coste en función del número de slots que presente la solución:

$$Fitness = km + \alpha * slots$$

De esta manera, conseguiremos evitar que el número de slots aumente de forma desproporcionada. Ya que, si no aplicamos un castigo. Contra más slots tengamos, menos movimientos de bicicletas tendremos que realizar. Debido a que la oferta de bicis es muy superior a la demanda.

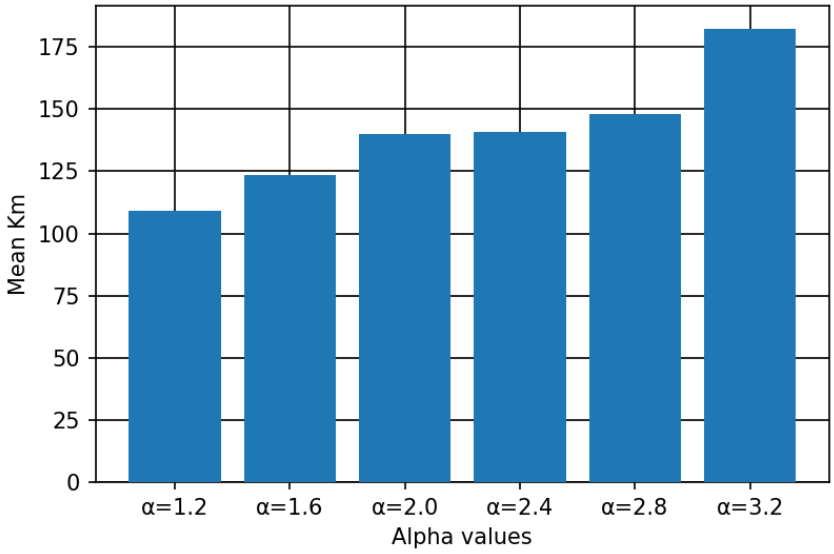
Los valores de alpha estudiados son los siguiente: [1.2, 1.6, 2,0, 2.4, 2.8, 3.2]. Este es el comportamiento que tienen a medida que aumentan los slots.

Coste adicional para fitness en función de alpha

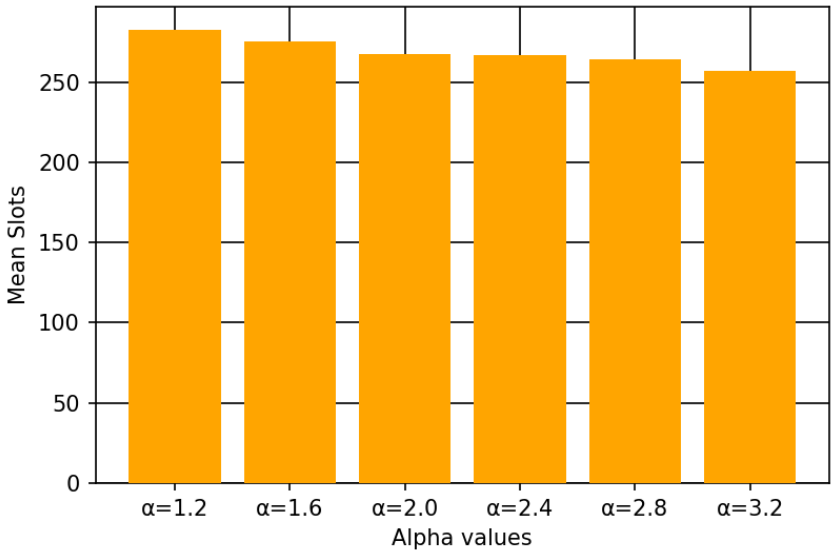


Para saber que *alpha* es más adecuado para nuestro problema, hemos realizado pruebas con sus distintos valores y en todas las semillas de test. Una vez encontrado los mejores valores, se han vuelto a evaluar las mejores soluciones encontradas en cada semilla. Pero esta vez, con una función de evaluación que solo evalúa los kilómetros recorridos (como en la práctica 1). Sean evaluado 1000 generaciones por semilla.

Kilómetros



Slots



Como podemos observar, los mejores resultados los obtenemos cuando *alpha* es más pequeña. Esto es porque el número de slots empleados es muy grande, y se puede observar como a medida que aumentando el *alpha*. El número de slots va decreciendo y el número de kilómetros va creciendo. El problema de emplear un *alpha* tan grande como *3.2*, o superior. Es que como vimos en la gráfica de *coste adicional para fitness*, a medida que aumenta *alpha*. El parámetro de kilómetros recorridos empleados en la fórmula del

fitness, será cada vez más pequeño. Solo teniendo relevancia el número de slots de la solución.

Para solucionar este problema se ha modificado la fórmula de fitness previamente presentada por la siguiente:

$$\alpha = (slots_{solución} - slots_{min})^{\gamma} / slots_{max}$$

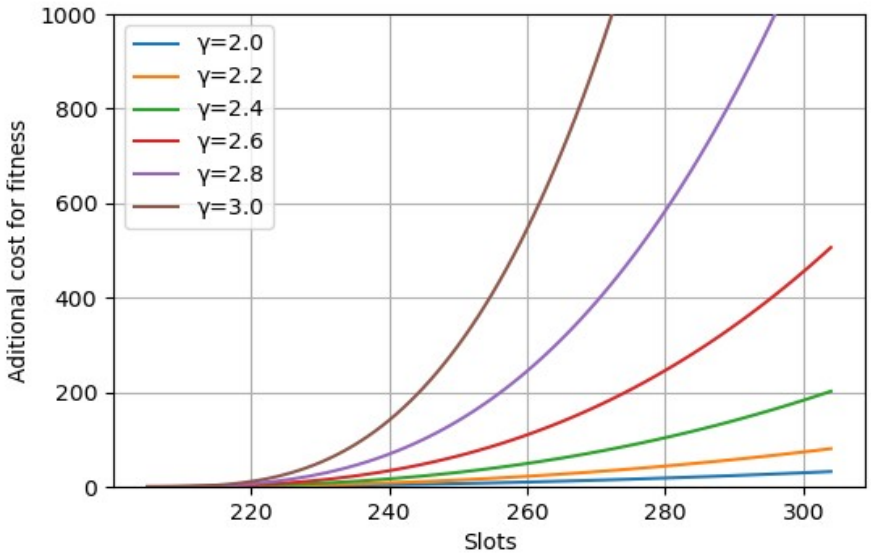
$$Fitness = km + \alpha * slots_{solución}$$

Siendo : $slots_{min} = 205$ y $slots_{max} = 305$

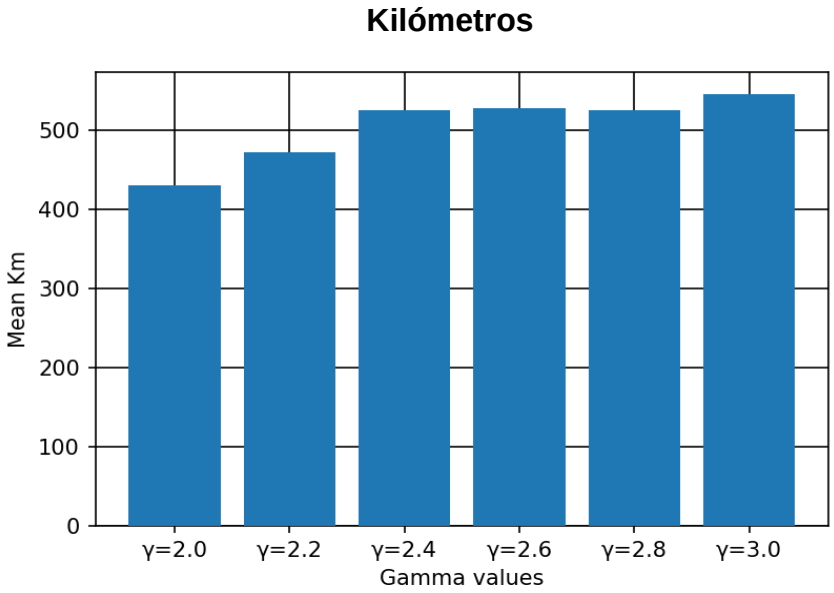
Ahora el parámetro esta ajustado a una función exponencial, de forma que en función de *gamma*. A medida que nos alejemos del mínimo número de slots, más penalización tendremos. El mejor valor de slots posibles es 205, lo ideal es acercanos lo máximo posible a este valor. Por contra partida, se ha establecido un número de slots máximos a utilizar, siendo este 305, 100 slots más del mínimo permitido.

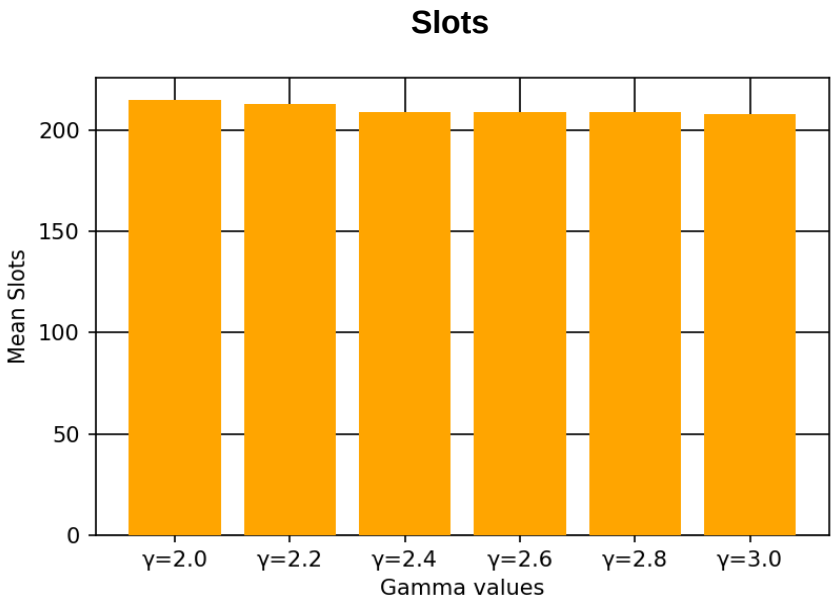
El parámetro que nos permitirá castigar más o menos, es ***gamma***. Por lo tanto, se ha realizado un pequeño estudio para ver como se comporta a medida que el número de slots aumenta y como es que se comporta el algoritmo tras usar este nuevo método. Los valores de *gamma* estudiados son los siguientes: [2.0, 2.2, 2.4, 2.6, 2.8, 3.0]. Este es su comportamiento:

Coste adicional para fitness con distintos valores de *gamma*



Para saber que *gamma* es más adecuado para nuestro problema, hemos vuelto a realizar pruebas con distintos valores y en todas las semillas de test. Una vez encontrado los mejores valores, se han vuelto a evaluar las mejores soluciones encontradas en cada semilla.





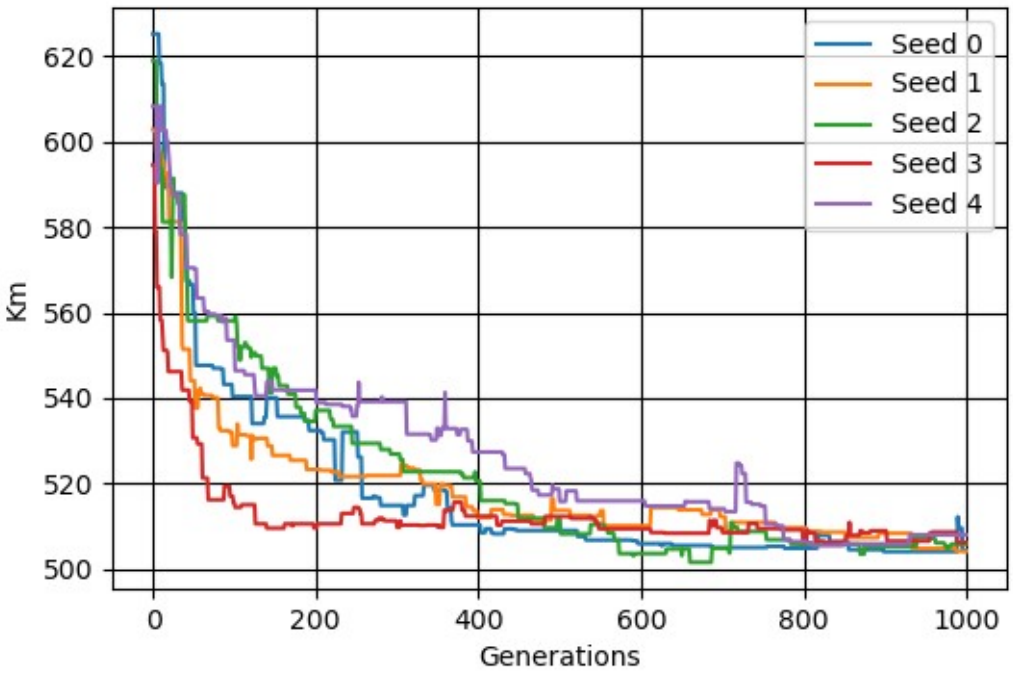
Como podemos apreciar, en esta ocasión. Siempre conseguimos una solución con una cantidad de slots muy cercana al mínimo permitido. Este es a costa de obtener peores resultados de fitness. También podemos observar como cuando *gamma* es igual a 2, es cuando se obtiene la mejor media de resultados. Mientras tanto, el resto de valores, castigan demasiado a cada uno de los individuos. Consiguiendo peores resultados. Por último, cabe aclarar que no es han utilizado valores *gamma* menores a 2 porque representarían un coste extra despreciable.

Por lo tanto, el modelo para evaluar a cada individuo será el último presentado, con ***gamma* igual a 2**. Este método de evaluación será el mismo en todos los algoritmos siguientes. Para que al final de la práctica, podamos comparar los resultados de cada algoritmo de la misma manera.

Población

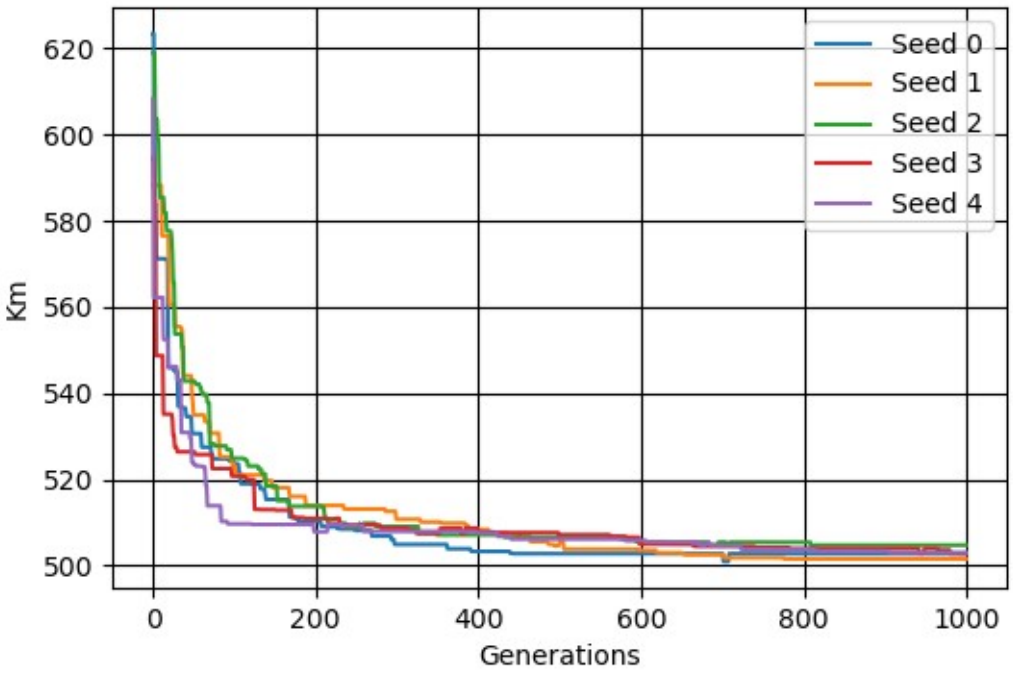
La población inicial que debe presentar el algoritmo debe estar entre 15 y 30 individuos. Además, debemos establecer un número máximo de generaciones. Para ello, se han estimado 3 tamaños de población diferentes y se ha comprobado como ha sido su comportamiento en distintas semillas durante 1000 generaciones. Los tamaños establecidos han sido: [16,24,30]

Población con tamaño 16



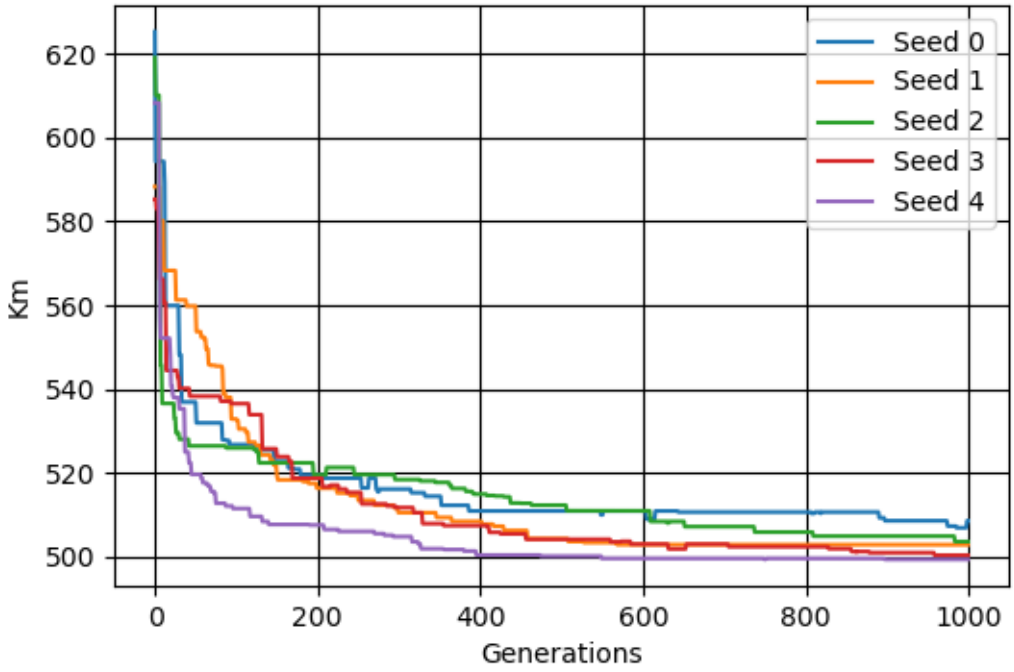
El mínimo encontrado por todas las semillas está entre 530-550 km. Como podemos observar, cada semilla ha conseguido acercarse al mínimo local a una velocidad diferente. Siendo la semilla 3 la rápida y la 4 la más lenta. Finalmente sobre la generación 800, todas las semillas encuentran el mínimo. Por lo que realizar más evaluaciones es innecesario.

Población con tamaño 24



Como se puede comprobar, con un tamaño de 24 individuos. Todas las semillas son capaces de encontrar el mínimo local a un ritmo similar. No encontrado cambios significativos a partir de la generación número 600.

Población con tamaño 30



A pesar de que la semilla 4 ha conseguido obtener una solución sobre la generación número 400, la mayor cantidad de diversidad genética presente en la población. Ha hecho que el resto de semillas tarden más en encontrar el mínimo local. Tardando muchas generaciones en conseguir mejorar su mejor solución. Probablemente, todas las semillas se acercarían al mismo resultado si realizáramos un mayor número de generaciones.

Con los distintos tamaños medidos, **una población de 24 individuos parece ser el tamaño óptimo** de la misma. Ya que es la que se acerca el mínimo local de forma más rápida y en menos tiempo.

Algoritmo CHC

El algoritmo CHC se caracteriza por tener un buen equilibrio entre diversidad y convergencia. El objetivo es combinar una selección elitista que preserva los mejores individuos que han aparecido hasta el momento con un operador de cruce que produce hijos muy diferentes a sus padres. El algoritmo cuenta con las siguientes características: **selección elitista, cruce BLX-α, prevención de incesto y multiarranque.**

Selección elitista

Durante el desarrollo del CHC, tendremos un conjunto de cromosomas considerados élites. Contaremos con **5 individuos de élite**. Una vez evaluada a una generación se comprobará si algunos de los nuevos individuos puede entrar en la élite.

Cruce BLX-α

El cruce BLX-α, permite mantener el equilibrio entre la exploración y la explotación que otorga este algoritmo. El mismo consiste en seleccionar de forma aleatoria la mitad de los genes que tengan ambos padres, intercambiarlos y finalmente mutarlos. Ejemplo con un cromosomas de 6 genes:

Padre A: [1,1,1,1,1,1] – Padre B: [4,5,6,7,8,9] | *Selección de genes [6/2=3]*

De forma aleatoria se escogen 3 genes: [2,1,5]

Hijo A: [1,5,6,1,1,9] – Hijo B: [4,1,1,7,8,1]

Una vez intercambiados mutamos los mismos genes.

Para mutar el número de slots de cada gen seleccionado, se ha empleado una campana de gauss, con media sobre el valor del gen y una desviación típica igual a 1.

Todo este proceso nos permitirá mantener un buen grado de exploración.

Prevención de incesto

A la hora de cruzar a los distintos individuos de la población. Se formaran N/2 parejas con elementos de la población. Y solo cruzaremos a las parejas que sean diferentes genéticamente. Para lograr esto utilizaremos la distancia de Hamming, Para saber si dos genes son iguales deberemos comprobar cuantos slots de diferencian hay entre uno y otro, teniendo que establecer un rango en el cual la diferencia absoluta entre ambos es menor al radio(rango). Solo los padres que sean considerados como diferentes, podrán tener descendencia.

A su vez, establecemos un umbral L/4 (L es longitud del cromosoma). Si durante una generación no se produce ni un solo cruce, el umbral de cruce se le resta 1. Cuando el umbral llegue a 0. Se producirá un nuevo arranque.

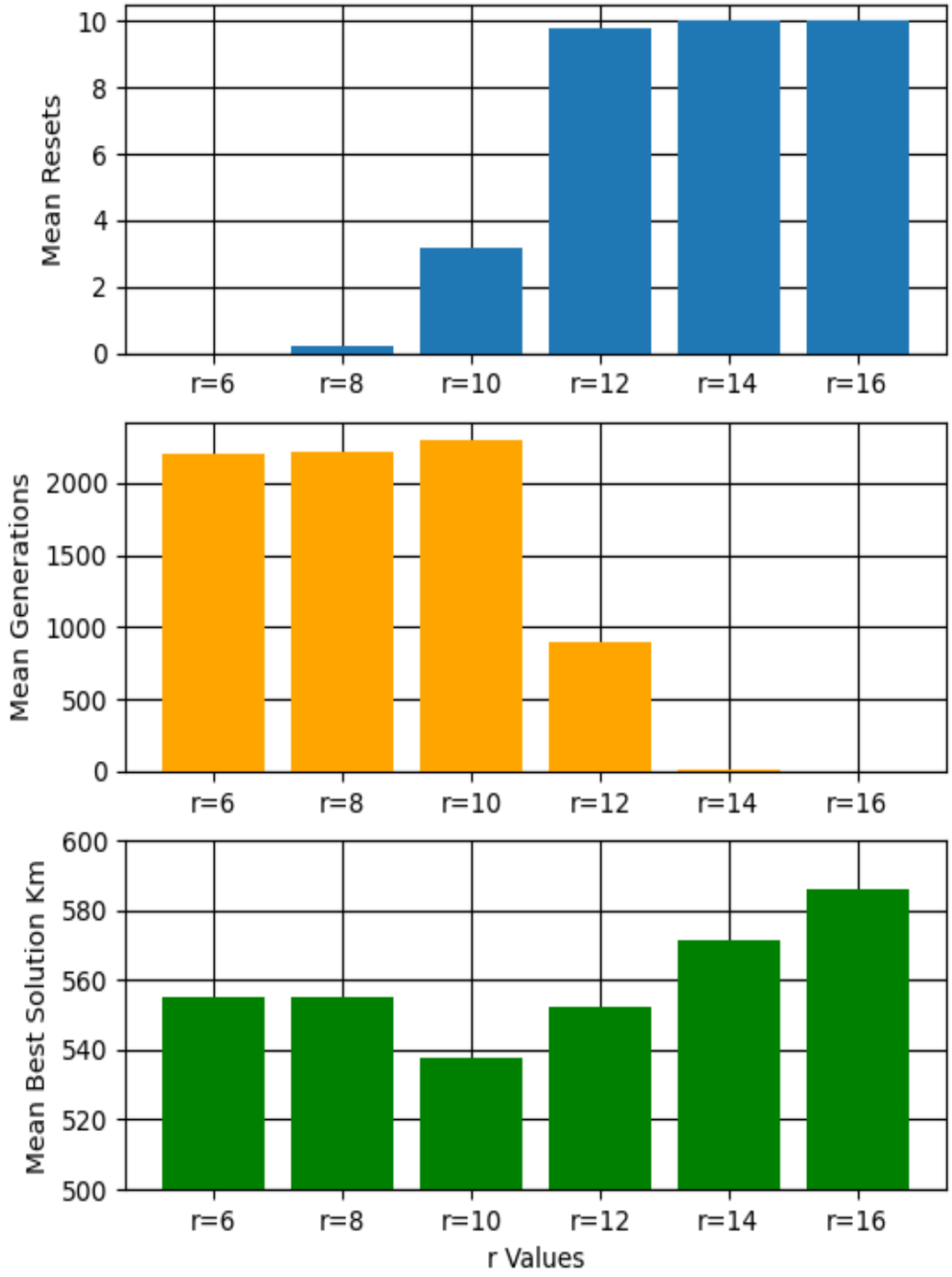
Multiarranque

Cuando el umbral previamente comentado llega a 0, la población se reinicializa. Creando una nueva población de individuos de forma aleatoria pero manteniendo a los individuos de élite encontrados.

Estudio sobre el tamaño del radio y los multiarranques

Debemos encontrar cual es el tamaño óptimo del radio a la hora de realizar el cruce BLX-α y comprobar cuantos rearranques se producen con dicho radio. Para realizar la prueba se ha dejado un tiempo máximo de 5 minutos por semilla. Se han probado los siguientes tamaños de radio: [6,8,10,12,14,16]

CHC Estudio de distintos tamaños de radio

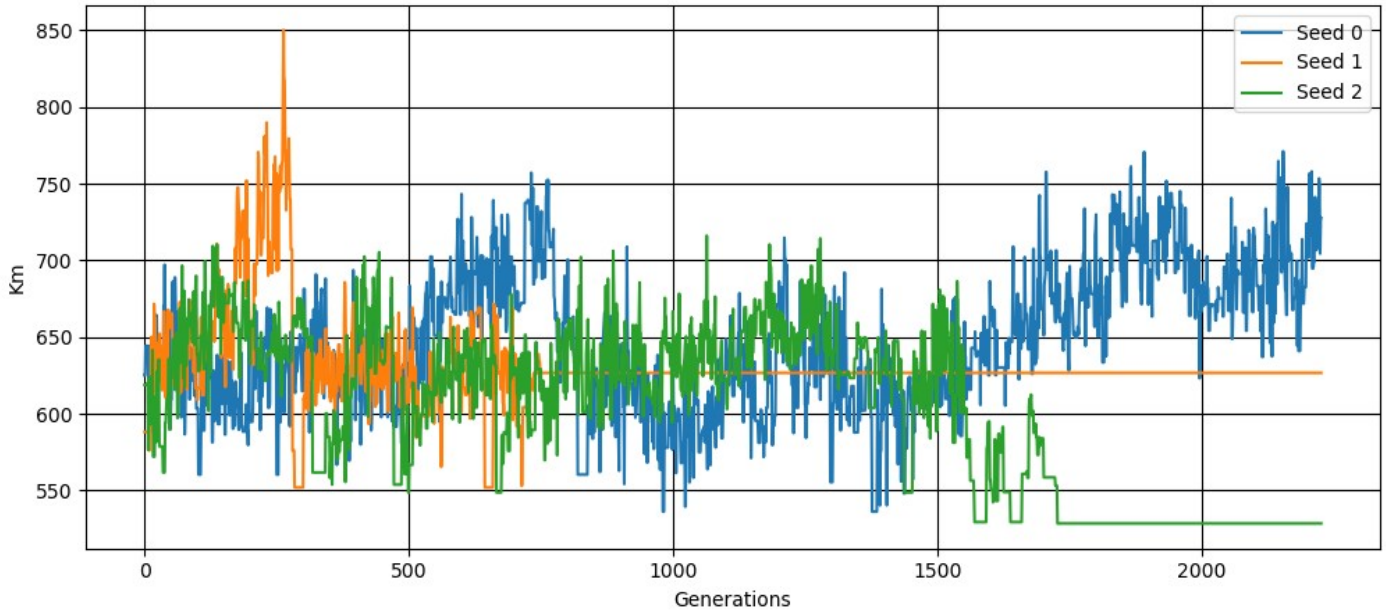


Como podemos observar, durante los tamaños de radio 6 y 8. El algoritmo no es capaz de converger en ninguna solución. Ya que casi nunca es capaz de resetear la población, gastando todo el tiempo permitido. Cuando tenemos un radio igual a 10, es cuando hemos conseguido obtener mejores valores de media. Pero una vez más, usando el tiempo máximo permitido en el experimento y no pudiendo resetear el número de veces establecido. Por otro lado, los radios 14 y 16 han sido los que han obtenido peores

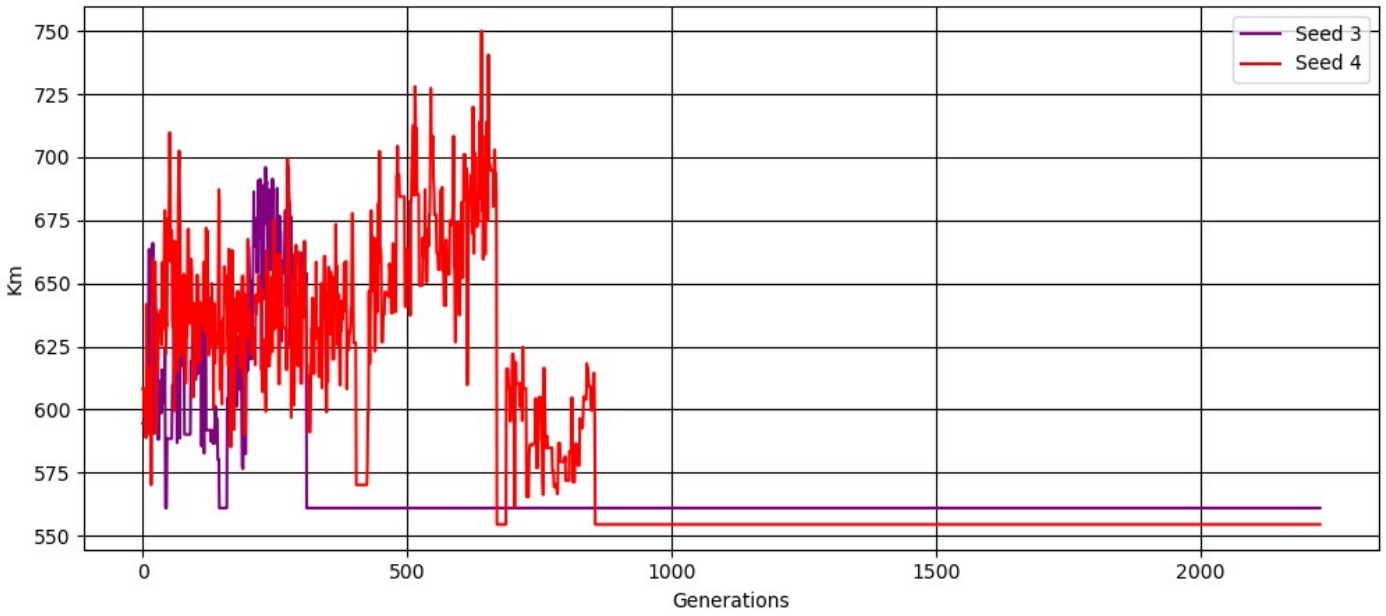
resultados. Consiguen realizar el número de reseteos máximo, pero no llegan a explorar porque su población es muy similar entre ella. Casi no se han llegado a reproducir.

El valor de **radio más adecuado es 12**, consigue completar casi siempre el número de reseteos, explorar un buen número de generaciones como podemos observar en la segunda gráfica. Y aunque obtiene ligeramente peores resultados que los obtenidos por las evaluaciones con radio 10. Lo hace en un tiempo mucho más razonable. Comprobemos como se comporta:

Comportamiento del CHC con r = 12, semillas = [0,1,2]



Comportamiento del CHC con r = 12, semillas = [3,4]



Para que las graficas pudieran ser representadas en la misma gráfica, se ha repetido el último valor encontrado para cada una de las semillas(que no significa que sea el mejor). Además se han separado las semillas para que se pueda ver con más claridad.

Las semillas 1, 3 y 4, No han conseguido obtener un buen balance entre exploración y explotación, ya que van reiniciando la población de forma prematura. Llegando a encontrar su mínimo local muy rápidamente. Por otro lado, las semillas 0 y 2 han sido las que más generaciones han realizado. Sin embargo, la semilla 0 ha seguido explorando al final sin ser capaz de encontrar su mínimo y sin realizar el número de reseteos máximo. Nuestra mejor semilla ha sido la número 2, que tras diversificar, consigue converger en el mejor resultado encontrado por todas las semilla.

A pesar no de comportarse igual en todas las semillas, **r igual a 12 es la que consigue el mejor balance de todos los tamaños de radio.**

Algoritmo genético multimodal

El tipo de algoritmo multimodal empleado es el modelo secuencial. Que consiste en la creación secuencial de distintos algoritmos genéticos básicos de forma dependiente. Es decir, el fitness obtenido por cada individuo en cada uno de los nichos se verá condicionado por el mejor resultado encontrado en nichos anteriores. Castigando a los cromosomas que sean muy similares a los mejores obtenidos en nichos previos.

Para saber como de cerca se encuentran dos individuos, calcularemos la distancia de Hamming.

Distancia de Hamming

Para determinar la distancia existente entre dos cromosomas, deberemos contar cuantos genes difieren entre si. Para saber si dos genes son iguales deberemos comprobar cuantos slots de diferencian hay entre uno y otro, teniendo que establecer un rango en el cual la diferencia entre ambos es menor al radio(rango). Ejemplo sobre 2 cromosomas de 6 genes:

Cromosoma A: [1,3,3,2,3,2] - Cromosoma B: [2,4,7,6,6,5] - Radio = 2

Diferencia absoluta: [1,1,4,4,3,3]

Con un radio de tamaño 2, solo los genes 0 y 1 tienen un valor menor o igual que el radio. Por lo tanto, podemos decir que ambos cromosomas están a una distancia de 4 unidades porque hay 4 genes que son diferentes.

Castigo por cercanía a un nicho

Necesitamos una forma de castigar a las soluciones si están muy cerca de nichos previos, y para ellos emplearemos la fórmula empleada en algoritmos multimodales secuenciales:

$$G(x, s_n - 1) = (d(x, s_n - 1)/r_{share})^{\beta}$$

Siempre y cuando se cumpla: $d(x, s_n - 1) < r_{share}$

Sino,la función será igual a 1

- x: Nuevo individuo encontrado
- s_{n-1}: Solución nicho encontrada en ejecuciones previas
- β(beta): Parámetro que indica el grado de castigo en función de la proximidad entre la solución y el nicho. Si beta=1, entonces la penalización será proporcional a los cerca que este. Sin embargo, si beta=2, tendremos una penalización exponencial. De forma que cuanto más se este acercando el nicho, más será penalizado.
- r_{share} : radio que emplearemos para determinar la distancia mínima a la que se debe encontrar un individuo de un nicho para no ser penalizado.

El problema de esta fórmula, es que esta pensada para maximizar, es decir, que castiga menos cuanto más se acerque a nichos previos. Por lo tanto, calcularemos el valor inverso:

$$G(x, s_n - 1) = 1 - (d(x, s_n - 1)/r_{SHARE})^{\beta}$$

Siempre y cuando se cumpla: $d(x, s_n - 1) < r_{SHARE}$

Sino,la función será igual a 0

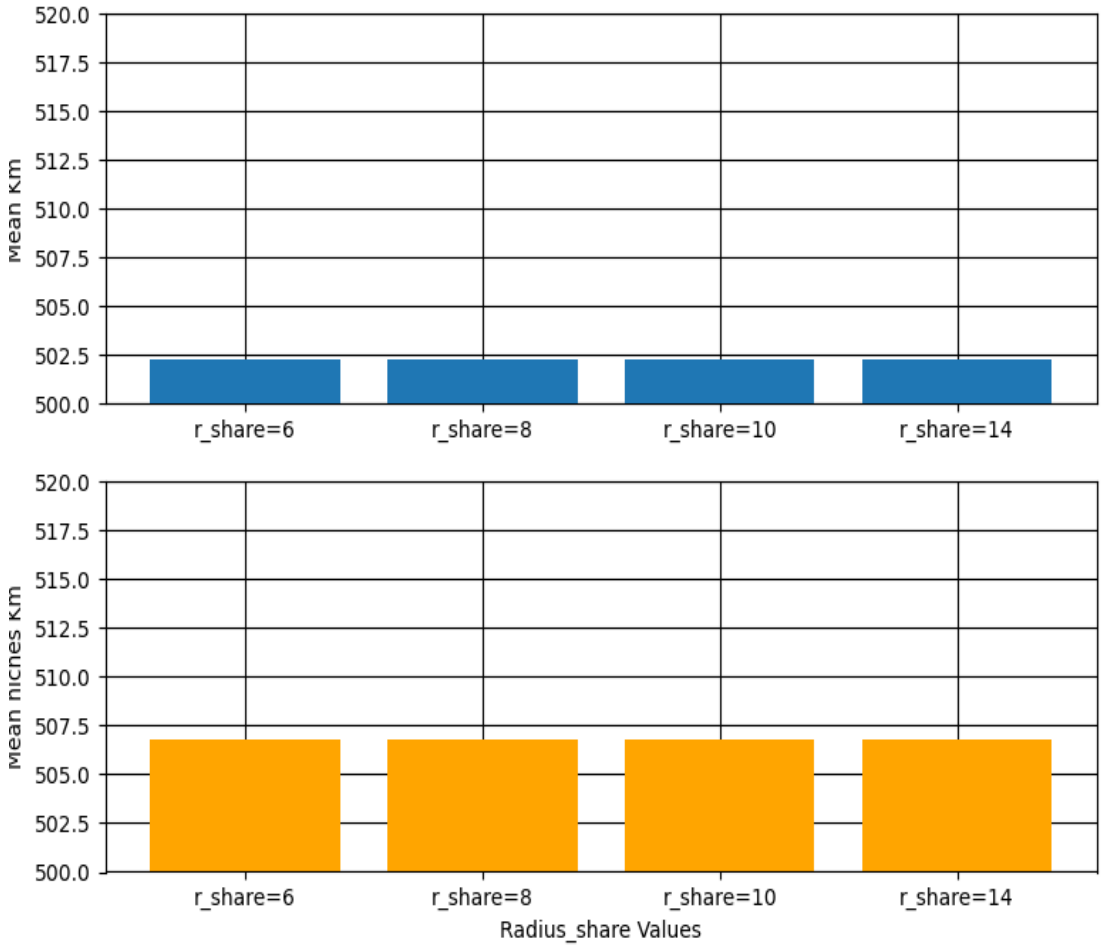
Finalmente, debemos sumarle al fitness obtenido el castigo:

$$Fitness_{final} = Fitness + \sum_{i=0}^{n_{nichos}} (Fitness * G(x, s_i))$$

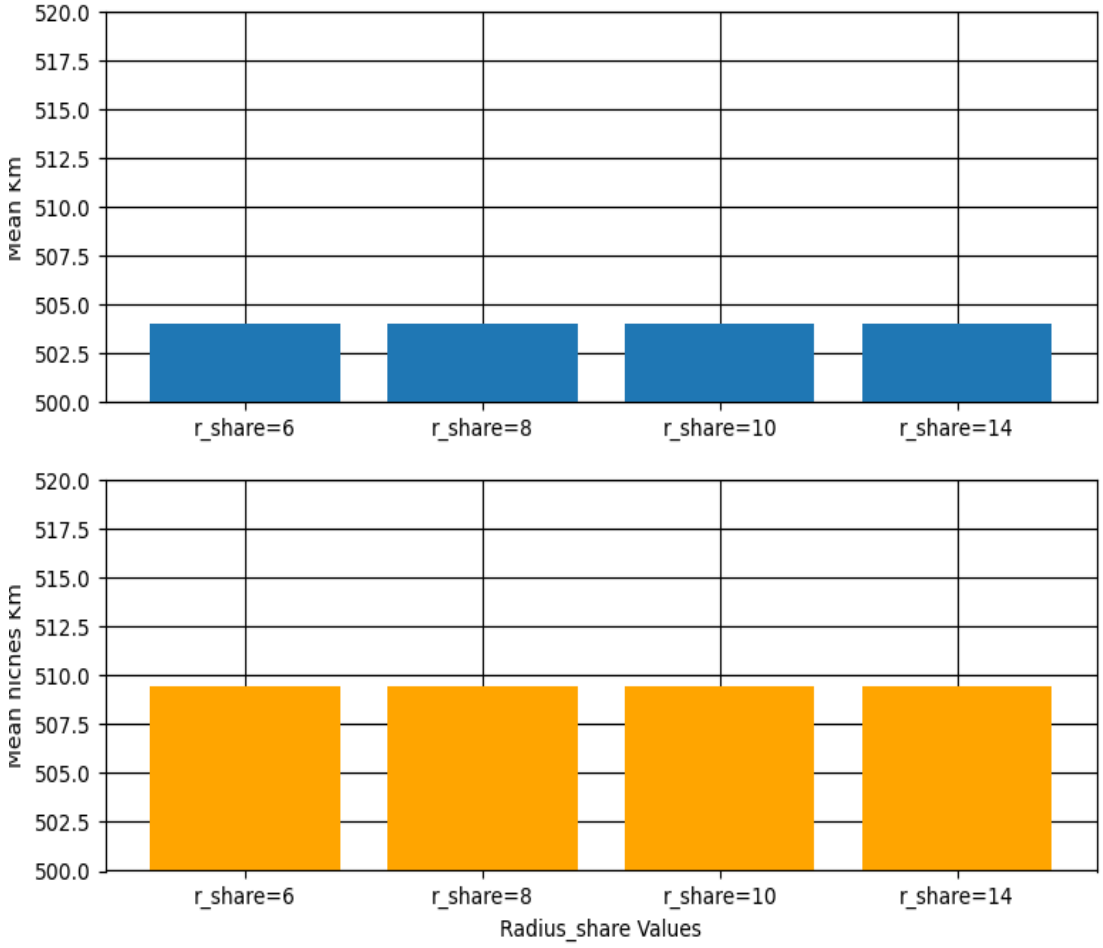
Estudio del radio y radio$_{SHARE}$

El algoritmo debe de encontrar 5 nichos. Y para eso debemos ajustar los valores del radio, radio$_{SHARE}$ y el exponente beta. Se ha establecido un número máximo de **500 generaciones por búsqueda local** (con el propósito de poder reducir un poco el tiempo de ejecución. 5 minutos aprox.). Además, **beta tendrá valor 1**. Penalizando de forma lineal a medida que nos acerquemos más al nicho. El valor del radio, nos dirá como de similares son dos genes. Mientras que el radio$_{SHARE}$ como de cerca estamos a un nicho. Los valores testeados han sido los siguientes: radio = [8,10] – radio$_{SHARE}$=[6,8,10,14]

Media de Km cuando r = 8

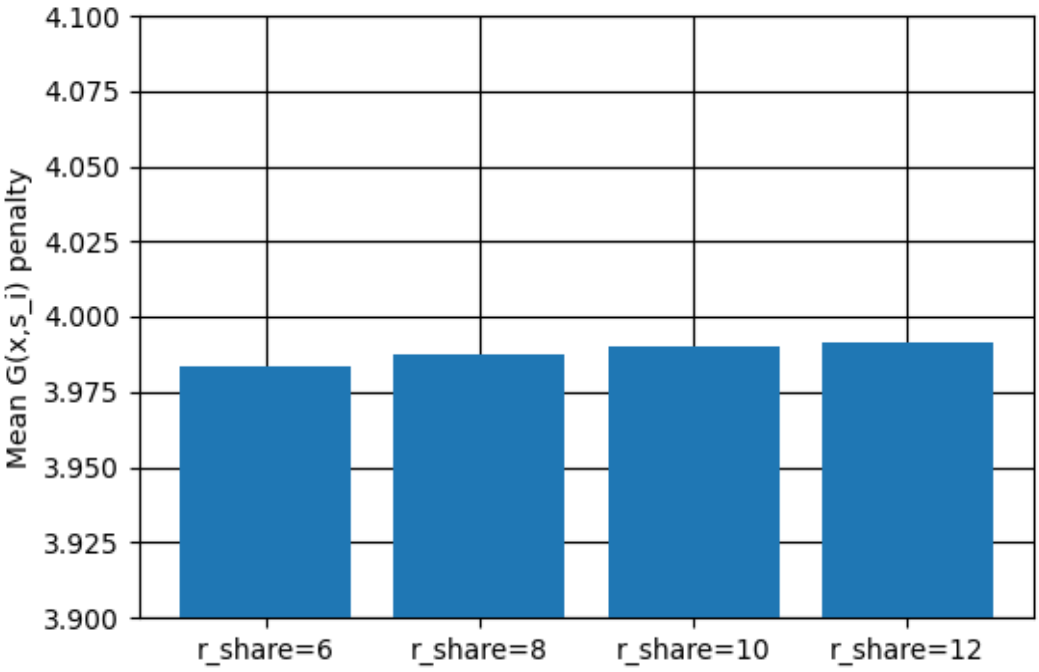


Media de Km cuando r = 10



Como podemos observar, cuando el radio es igual a 8 es cuando obtenemos mejores resultados. Comprobemos como son sus soluciones. Para ello, vamos a calcular cuanto penalización de la función **G**(x,s_i) consigue la primera solución nicho comparada con las demás. Una vez calculada el coste en cada semilla, se realizará el promedio

Media de penalización en función del radio$_{SHARE}$ cuando radio = 8



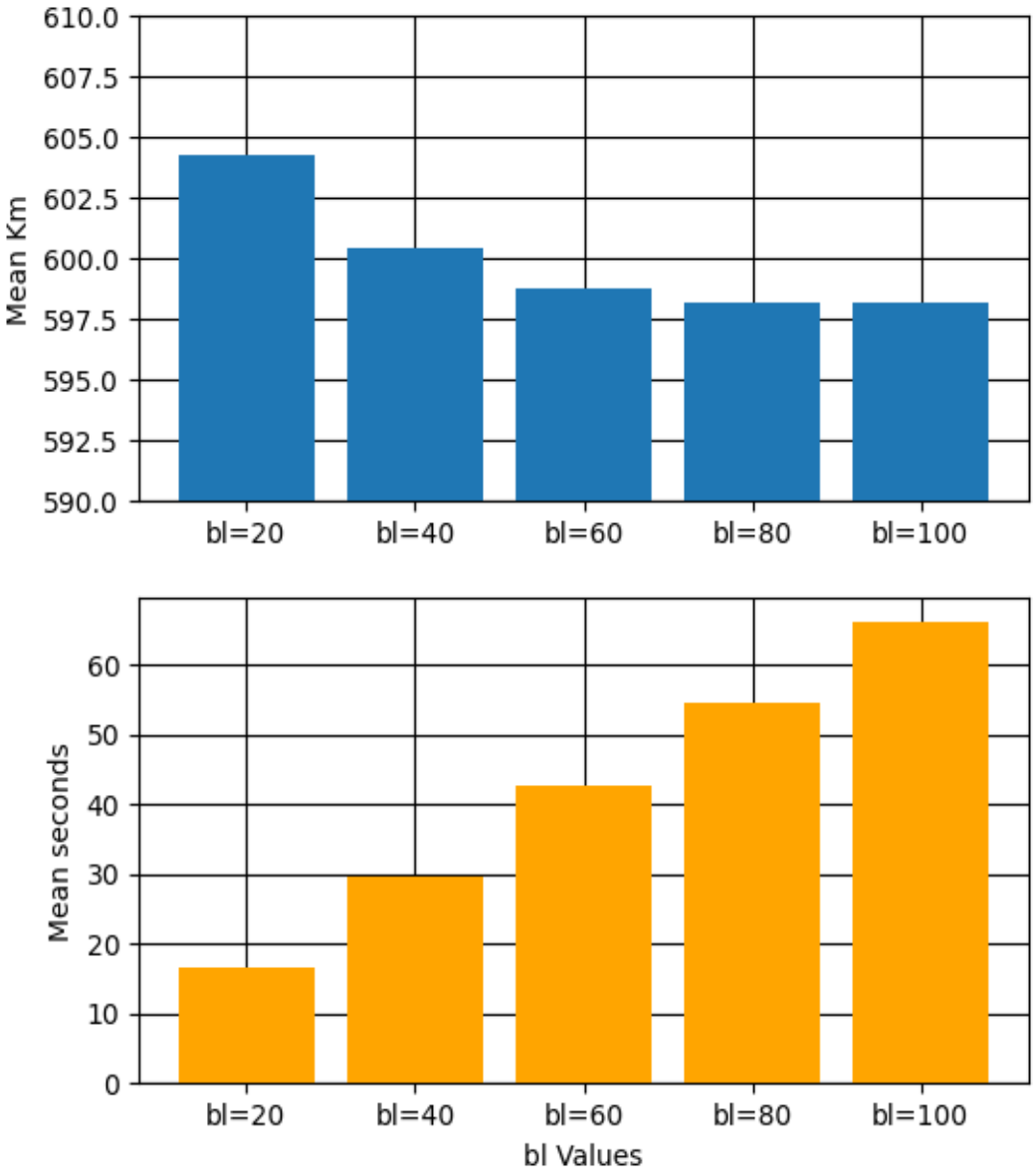
Tras comprobar los resultados, cuando el radio$_{SHARE}$ es igual a 6. Consigue obtener ligeramente menor penalización, significando que se aleja más de los nichos encontrados. Pero no representa una diferencia significativa con respecto a los otros valores. Prácticamente todos reciben la misma cantidad de castigo en promedio. Por lo tanto, no he sido capaz de encontrar un valor adecuado o significativamente mejor para radio$_{SHARE}$. En su defecto, **radio$_{SHARE}$ será igual a 6**.

Algoritmo VNS

El algoritmo VNS consiste en realizar búsquedas locales en tamaños de vecindarios definidos. El tamaño estará definido por la variable s, que podrá presentar los siguientes valores: [4,8,12,16]. Con en este valor podemos crear un nuevo vecino de la solución actual, formando s/2 parejas y moviendo n_{slots} entre las mismas (la cantidad a mover será igual 1, la misma que la que se utilizó en la búsqueda local de la práctica 1). Una vez generado el vecino. Se hará una búsqueda local sobre el mismo y se comprobará si el resultado obtenido es mejor que la solución actual. En caso de serlo, el valor de s volverá a ser el más pequeño (s=4). En caso contrario, su valor será igual al siguiente valor de la lista. Si siendo s=16 no se encuentra un vecino mejor a la solución actual. El valor de s volverá a valer 4 y se volverá a intentar.

La condición de parada de algoritmo viene dada por el número de búsquedas locales máximas a realizar, es por ello que se ha realizado el siguiente experimento:

Medias de Km y tiempo con distintos tamaños de búsquedas locales



Se puede observar como la media de mejores resultados obtenidos se estabiliza cuando llegamos a realizar un total de 60 búsquedas locales. A partir de las mismas, no consigue haber mejora y el tiempo empleado es mayor. Por lo tanto, **el valor de búsquedas locales máximas utilizado será 60.**

Comparación de resultados

Para realizar las pruebas de comparación entre los distintos algoritmos se han utilizado dos nuevas semillas, están se encuentran en el archivo ‘seeds.csv’.

Algoritmo	Ev. Medias	Ev. Mejor	Desv. tipica Ev	Mejor Kms	Media Kms	Desv. tipica Kms
Local_search	1136,5	953	183,5	578,780	580,477	1,697
VNS	6929	5286	1643	558,640	573,453	14,813
Basic_genetic	14424	14424	0	500,334	503,383	3,048
Multimodal	60120	60120	0	500,374	501,587	1,212
CHC	48756	47088	1668	544,957	546,208	1,251

Como podemos observar, todos los algoritmos han conseguido mejorar el resultado de la búsqueda local. Siendo el genético básico y el genético multimodal los que mejores resultados han obtenido. Sin embargo, vista la similitud en ambos resultados, no existe una correlación con el número de evaluaciones. Siendo mucho más rápido el genético básico. Lo mismo sucede con el algoritmo CHC, no se llega a acercar al genético a pesar de su gran número de evaluaciones.

Como conclusión final, podemos observar que el algoritmo genético multimodal ha sido el que mejores resultados de media ha conseguido obtener con la menor desviación típica.

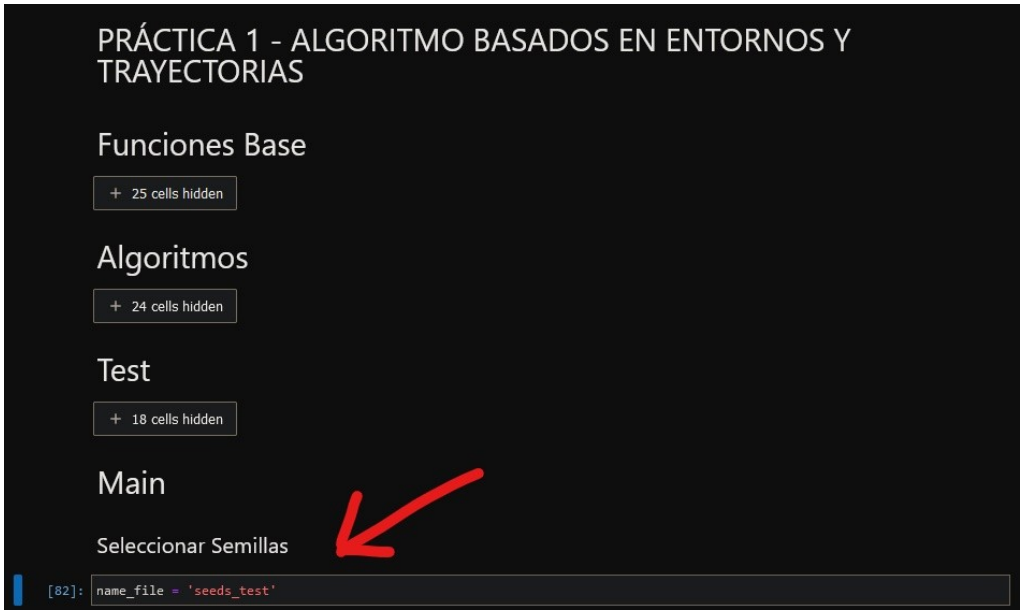
Como ejecutar la práctica

Para ejecutar la practica abriremos el archivo Practica2.ipynb con jupyterLab o Jupyter Notebook.

Una vez hecho eso, el código esta estructura de forma que a partir de un archivo de semillas en un archivo .csv, se ejecutará la práctica. Este archivo se deberá guardar en la carpeta ‘seeds’. Un archivo seed tiene la siguiente estructura:

Orden	Semilla
0	876643
1	486432
…	...
n	435468

Escribiremos el nombre del archivo .csv sin la extensión en la siguiente variable:



Una vez finalizado, se guardaran los resultados en la carpeta ‘resultados’ con el mismo nombre que hemos introducido. Además, se crearán todas las tablas presentadas en este documento en la carpeta ‘estudio’ (menos el estudio de slots).

Por último, se representará la mejor solución por pantalla y se guardará en la carpeta ‘map’. Se mostrará una circunferencia por cada estación, su tamaño indica la cantidad de slots que presenta. Y el color si ha aumentado o disminuido con respecto a la solucion greedy. Colores posibles:

- Color rojo: El número de slots ha disminuido
- Color verde: El número de slots se ha mantenido igual
- Color azul: El número de slots ha aumentado

Para su correcto funcionamiento se deberá instalar la librería folium:

```
pip install folium
```