Modelos Bioinspirados y Hurísticas de Búsqueda

Práctica 1

Daniel Pérez Rodríguez

Índice

Introducción

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

- Greedy
- Búsqueda aleatoria
- Búsqueda local
- Enfriamiento simulado
- Búsqueda tabú

El código desarrallodo deberá optimizar el uso de una red de bicicletas y seguidamente los resultados obtenidos por cada algoritmo deberán ser comparados con la solución greedy.

Algoritmos

Para la ejecución de los algoritmos se han usado las siguiente **semillas: 13961564, 190879351, 98576233, 45672790, 190879351**. Siendo semilla 1, 2, 3, 4 y 5 respectivamente. Por otro lado, es necesario explicar el funcionamiento de la función de evaluación, el generador de soluciones iniciales y el operador de movimiento.

Función de evaluación

Para realizar la función de evaluación primero deberemos transformar la matriz delta en una lista de movimientos. Cada valor de esta lista es una tupla formada por la estación y el número de bicicletas desplazadas de la estación. A la hora de evaluar un movimiento, se asignará el número de bicis resultante a la estación. Y en caso de que el número de bicis a tomar o el numero de slots no sea suficiente, se buscarán bicis o slots en las estaciones más cercanas. De forma progresiva, es decir, si la estación más cercana no puede suplir nuestra demanda. Entonces se asignará/tomará el número de bicis o slots disponibles y buscaremos en la siguiente estación más cercana.

Generador de soluciones iniciales

A la hora de generar soluciones iniciales, lo que haremos será generar valores entre 2 y 10 para cada una de las estaciones. Una vez hecho esto, debemos ajustar el array de forma proporcional hasta el número de slots máximos, 220. Pero en lugar de hacer eso, lo que haremos será ajustarlo a 200. De esta forma, nos reservamos cierta cantidad de slots para hacer que la solución generada sea compatible con el estado inicial. Ya que si alguna de las estaciones tiene más bicis que slots asignados en el estado inicial no será una solución válida, es por eso, que se usarán estos 20 slots reservados para repartir entre las estaciones que no cumplan esta condicion. Una vez asignados los slots necesarios, se asignara de 1 en 1 desde la estación 0 hasta la estación ‘n’. Hasta que el total sea 220.

Operador de movimiento

Será necesario crear soluciones vecinas en la mayoría de algoritmos y para ello usaremos la función llamada operador de movimiento. Esta función hará uso de un mapa de vecinos. Este mapa esta

formado por todas las posibles combinaciones de estaciones sin repetir (esta lista es creada al principio del programa, solo se calcula una vez), y la forma de utilizarla será la siguiente. Se lanzará un número aleatorio entre el valor 1 y el máximo de combinaciones posibles, obteniendo así una combinación de estaciones. Cuando deseemos obtener una cantidad de vecinos determinada, iremos recorriendo el mapa de forma secuencial hasta que eventualmente lleguemos hasta la primera combinación de estaciones explorada. Momento en el que dejaremos de explorar vecinos. Por último, quiero aclarar que un vecino es aceptado cuando es compatible con el estado inicial.

Greedy

El primer algoritmo implementado es el greedy. Este algoritmo es el más sencillo y rápido, nos servirá para tener una solución de refencia. Consiste en generar una solución a partir del estado inicial, para ello deberemos aumentar el número de cada estación de forma proporcional hasta que la suma total de slots en cada una de las estaciones sea 220 (el máximo permitido).

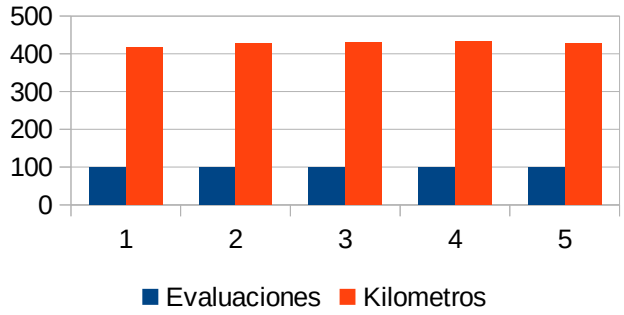
Semilla	Evaluaciones	Kilometros
0	1	574,551

Tras evaluar la solución generada nos encontramos con que su coste es de 574,55 km. Con el resto de algoritmo trataremos de encontrar un coste menor al aquí mostrado.

Búsqueda aleatoria

En la búsqueda aleatoria, se elige aleatoriamente una muestra del entorno que conforma todo el espacio de búsqueda. Y tras un número finito de iteraciones, el algoritmo devuelve la mejor solución encontrada.

Semilla	Evaluaciones	Kilometros
1	100	415,702
2	100	426,419
3	100	429,483
4	100	432,032
5	100	426,419



Búsqueda local

El algoritmo búsqueda local, se asocia al uso de estructuras de entorno. De esta forma, su

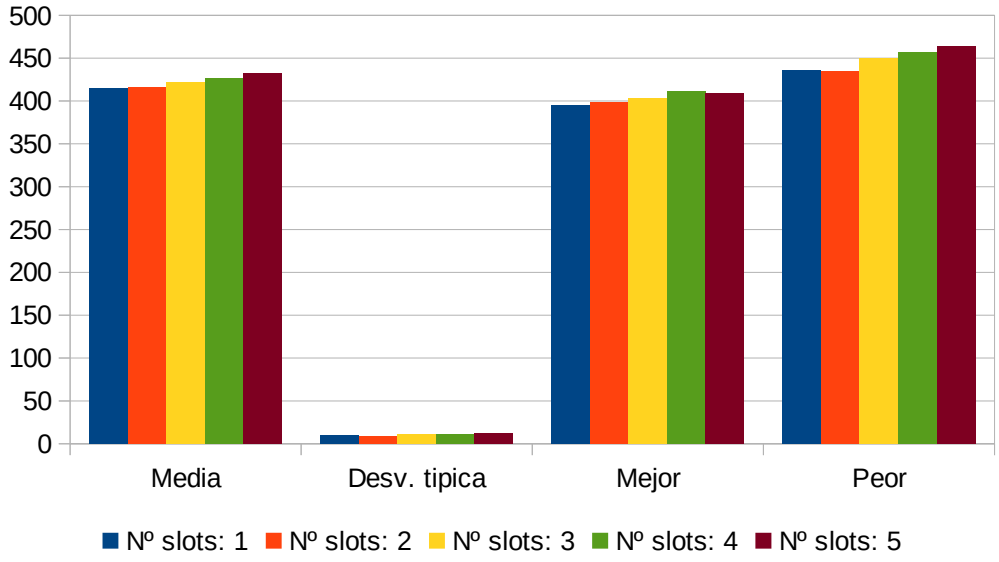
funcionamiento consistirá en que a partir de una solución, se seleccionará una nueva solución del entorno para continuar su búsqueda. La estratagia para seleccionar al siguiente vecino será usar primero el mejor, para así realizar un menor número de evaluaciones. Además, será establecerá un número máximo de nuevos vecinos encontrados.

Estudio de número de slots a mover

Para generar nuevos vecinos, es necesario utilizar un operador de generación de nuevas soluciones. Este operador consistirá en seleccionar dos posiciones (estaciones) del vector solución y mover cierto número de slots de una estación a otra. Es por eso que realizaremos un pequeño estudio probando diferentes tamaños.

Se han establecido 50 semillas diferentes, y se han ejecutado todas las semillas con un número de slots determinado:

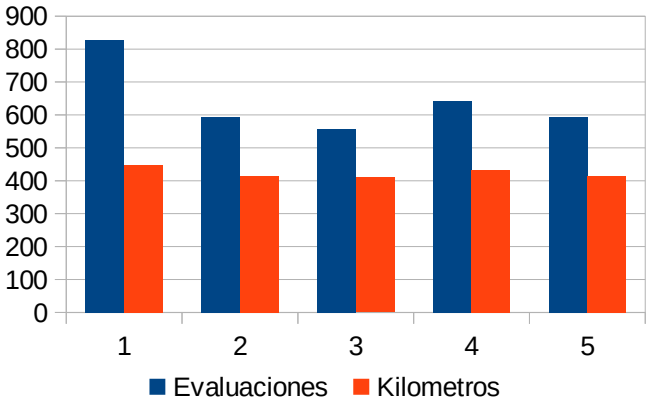
	Nº slots: 1	Nº slots: 2	Nº slots: 3	Nº slots: 4	Nº slots: 5
Media	414,270	416,0275	422,009	425,935	432,244
Desv. tipica	9,255	8,336	11,016	10,748	12,083
Mejor	395,526	398,683	403,524	411,187	409,106
Peor	435,442	434,821	449,587	457,324	464,294



Con los resultados, podemos observar que al mover 1 o 2 slots obtenemos resultados muy parecidos. Sin embargo, mover 1 slots ha obtenido ligeramente mejores resultados. Y es por eso que será el número de slots a utilizar.

Resultados búsqueda local

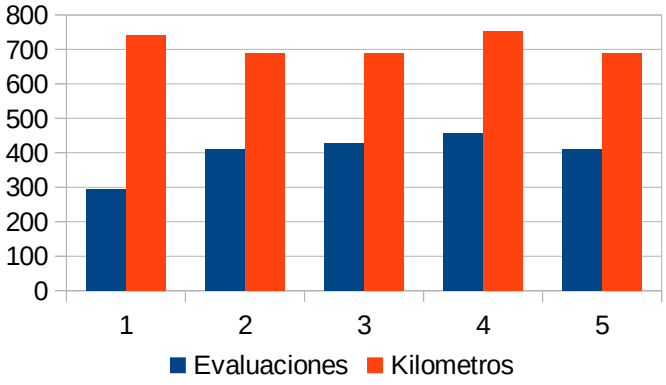
Semilla	Evaluaciones	Kilometros
1	827	448,684
2	595	413,908
3	557	410,153
4	643	431,943
5	595	413,908



Enfriamiento simulado

El algoritmo de enfriamiento simulado se caracteriza por permitir movimientos hacia soluciones peores para escapar de óptimos locales. Para lograr esto, tendremos una función de probabilidad que hará disminuir la probabilidad de estos moviemientos hacia soluciones peores conforme avanza la búsqueda. Diversifica al principio e intensifica al final.

Semilla	Evaluaciones	Kilometros
1	295	741,822
2	411	687,715
3	428	687,715
4	458	751,826
5	411	687,715



Búsqueda tabú

La búsqueda tabú es un procedimiento de búsqueda por entornos y se caracteriza por dos rasgos principales:

- Permitir moviemientos con peores resultados (como en enfriamiento simulado)
- Emplear mecanismos de reinicialización para mejorar la capacidad del algoritmo de explorar y explotar

Para llevar acabo estos dos apartados, se hará uso de una lista tabú a corto plazo y de una matriz de frecuencias.

Lista tabú de corto plazo

Se ha creado la clase Tabu, para desempeñar esta lista. La clase se comporta como una cola, de forma que cuando la lista esta llena y se desea introducir otro elemento en la misma. Se expulsa el elemento más antiguo y se añade el nuevo. Cada elemento de la lista esta formado por una tupla de 4 valores: ($estacion_1$, $estacion_2$, $valor\_estacion_1$, $valor\_estacion_2$) , que identifican las estaciones implicadas y los valores tomados al realizar un movimiento. Además, la clase cuenta con un diccionario para realizar las búsquedas dentro de la misma de forma más rápida. La estructura del diccionario es la siguiente, cada clave del mismo es la combinación de estaciones de la tupla comentada anteriormente. Y por cada clave, hay una lista de tuplas que indica el valor que tuvieron dichas estaciones en un movimiento determinado.

Matriz de frecuencias

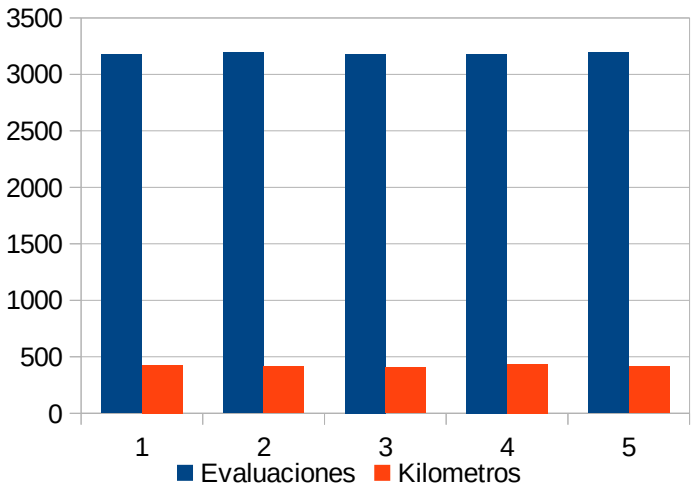
En la matriz de frecuencias se almacenará el número de veces que un par estación/n_slots ha sido explorado. Para ello se usará la siguiente matriz:

Matriz	Estación_0	Estación_1	Estación_2	Estación_3	...	Estación_n
Slots 0	1	1	1	1	1	1
Slots 1	1	1	1	1	1	1
Slots 2	1	1	1	1	1	1
Slots 3	1	1	1	1	1	1
...	1	1	1	1	1	1
Slots m	1	1	1	1	1	1

Sea creará una matriz inicializada a 1 donde en cada celda se indicará el número de veces que cierta cantidad de slots se ha utilizado en un estación. Será necesario establecer un máximo de slots a usar (en la práctica sea utilizado m=35, porque en las pruebas realizadas no se ha llegado a ver ningún valor mayor a este).

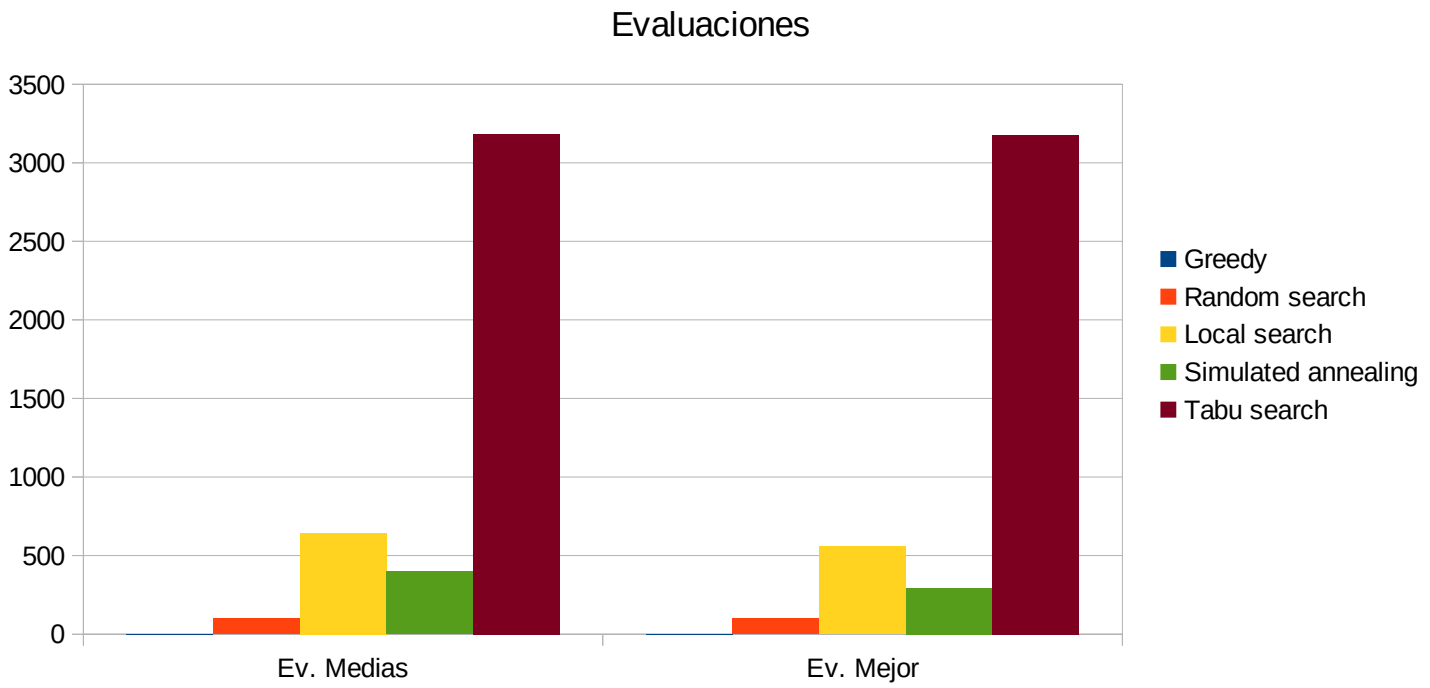
Resultados

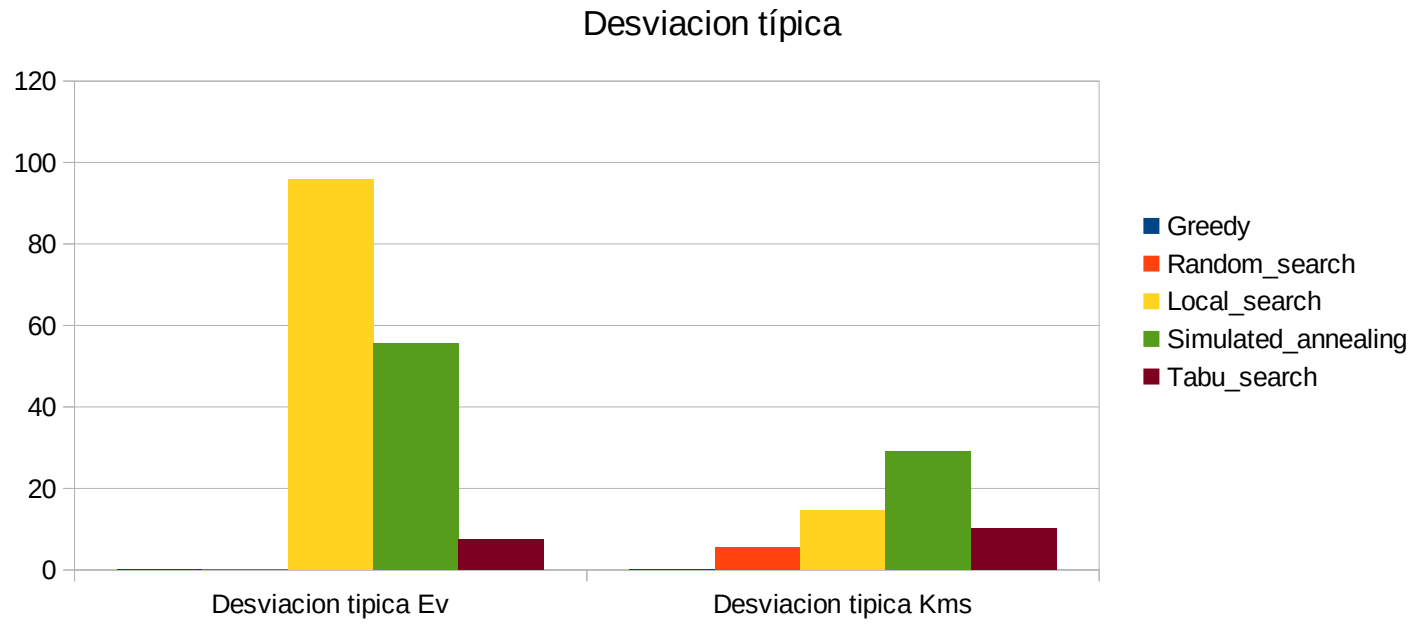
Semilla	Evaluaciones	Kilometros
1	3176	424,741
2	3193	414,304
3	3178	404,450
4	3180	434,096
5	3193	414,304

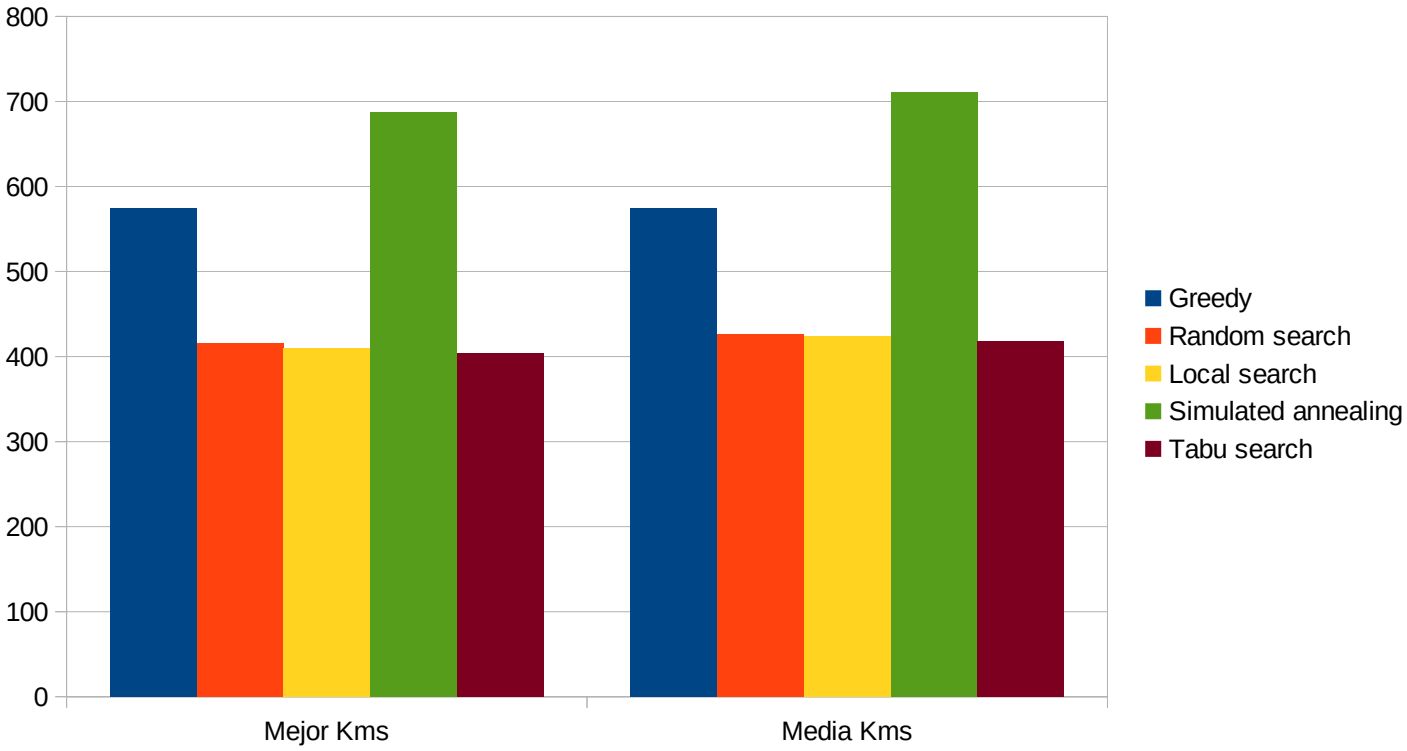


Comparación de algoritmos

Algoritmo	Ev. Medias	Ev. Mejor	Desv. tipica Ev	Mejor Kms	Media Kms	Desv. tipica Kms
Greedy	1	1	0	574,551	574,551	0
Random search	100	100	0	415,702	426,011	5,566
Local search	643,4	557	95,769	410,153	423,719	14,611
Simulated annealing	400,6	295	55,521	687,715	711,359	29,129
Tabu search	3184,0	3176	7,456	404,450	418,379	10,146







Como podemos ver, el algoritmo de búsqueda aleatoria es el que evaluaciones realiza. Esto es así porque hemos establecido un número máximo de nuevas soluciones. Por otro lado, la búsqueda tabú supera ampliamiente al resto, debido al número de reinicializaciones llevadas a cabo para poder diversificar todo lo posible. Aunque la ésta haya conseguido obtener la mejor solución, la búsqueda local y la aleatoria, han conseguido resultados muy parecidos. Por lo que no compensa el número de evaluaciones tan grande. De haberse establecido un tamaño mayor de lista tabú a corto plazo quizás se podría haber conseguido un mejor resultado. Por último, destacar el mal desempeño del enfriamiento simulado. Que ha conseguido resultados mucho peores que el resto de algoritmos, inclusive, que el algoritmo greedy.

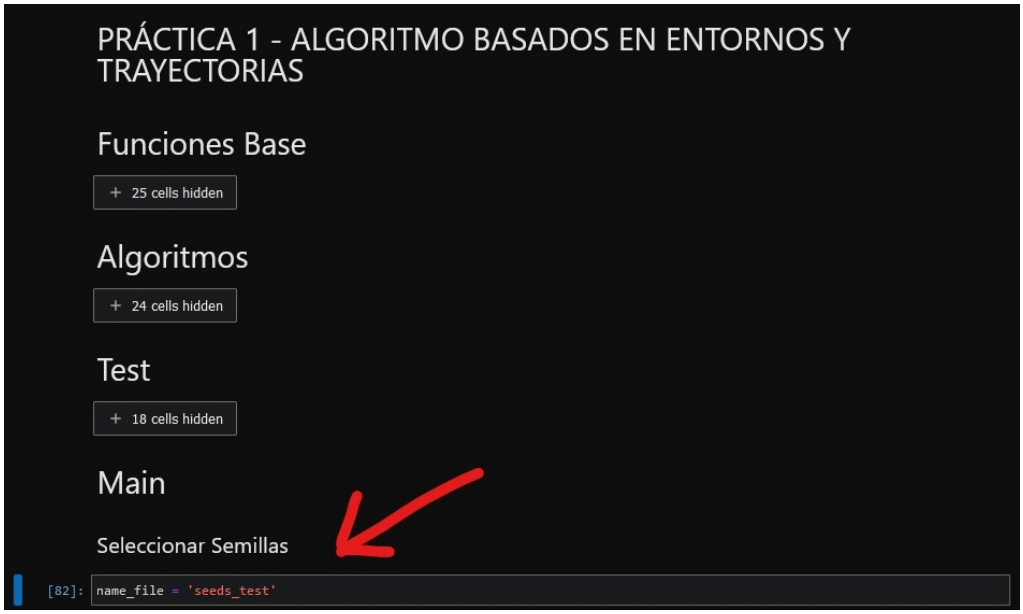
Como ejecutar la práctica

Para ejecutar la practica abriremos el archivo Practica1.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:



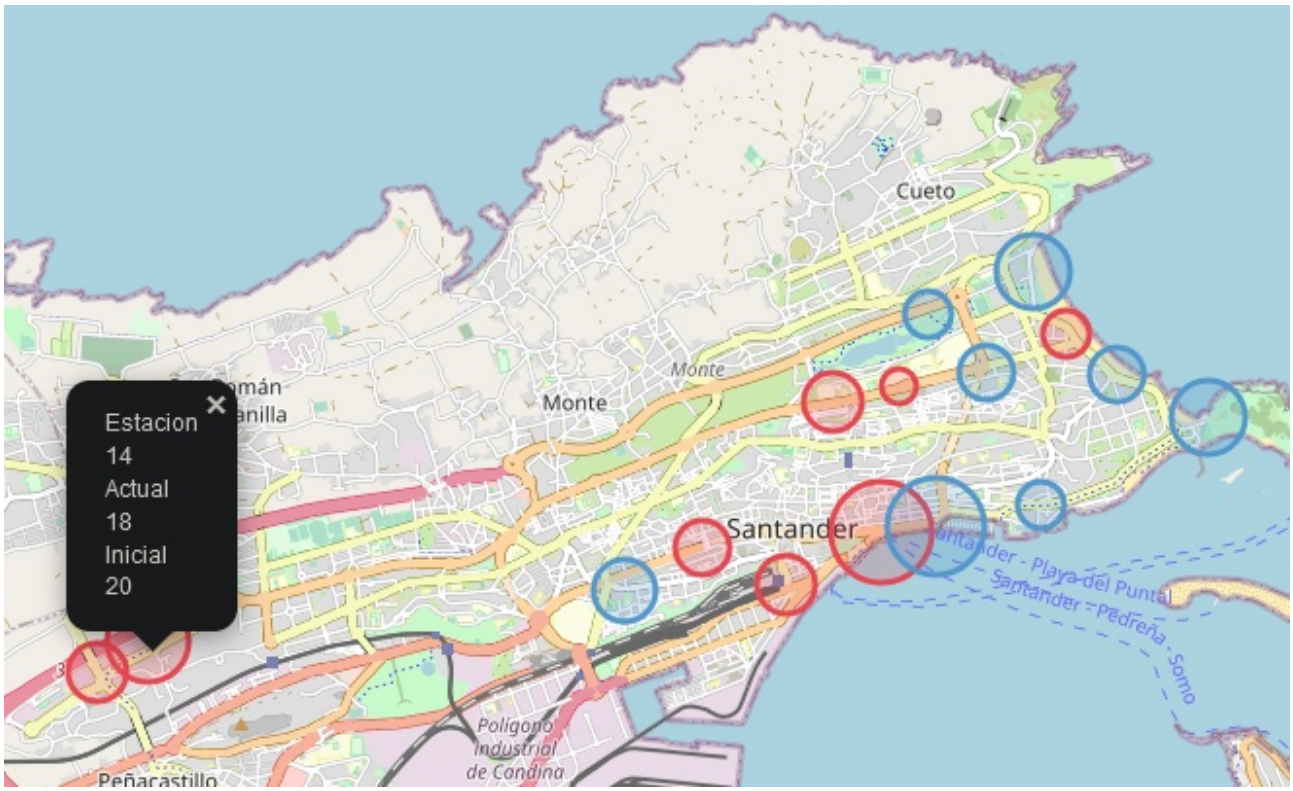
Una vez finalizado, se guardaran los resultados en la carpeta ‘resultados’ con el mismo nombre que hemos introducido. Además, se crearán todas las tablas presentadas en este documento en la carpeta ‘estudio’ (menos el estudio de slots).

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
```



Nota importante

Para la búsqueda de las semillas se hicieron pruebas en 1000 semillas diferentes (las pruebas se encuentran en la carpeta seeds y resultados). Sin embargo, en el archivo 'seeds_1000.csv' en la carpeta de resultados. Se encuentran mejores valores que los expuestos en esta memoria. Y es que cuando intentas obtener los mismos valores ejecutando otra vez la semilla no obtengo los mismos resultados. **Durante la ejecución de estas 1000 semillas el ordenador se suspendio**, provocando a mi entender un problema con las mismas. Ya que, **si se vuelven a ejecutar las semillas utilizadas en esta memoria, se obtendrán los mismos resultados expuestos.** Pero en el archivo comentado, las mismas tienen mejores resultados.