Modelos Bioinspirados y
Heurísticas de Búsqueda
4 Curso Grado Ingeniería en
Informática
Área de Ciencias de la Computación e
Inteligencia Artificial
Departamento de
Tecnologías de la Información

PRÁCTICA 1 (Versión 2022, 3.0)

Algoritmos basados en Entornos y Trayectorias

Objetivos

El objetivo de esta práctica es estudiar el funcionamiento de los *Algoritmos de Búsqueda Aleatoria, Local, Enfriamiento Simulado y Búsqueda Tabú*. Para ello, se requerirá que el alumno implemente estos algoritmos, para resolver un problema de *optimización de uso de una red de bicicletas*. El comportamiento de los algoritmos de OCH implementados deberá compararse con un *Algoritmo Greedy*.

Problema de la optimización de la red de bicicletas en una ciudad

Partimos de una red de estaciones de bicicletas con unas capacidades dadas donde se ha extraído el comportamiento medio a lo largo de un día capturando el movimiento de bicicletas en cada estación por horas desde las 8:00 am hasta las 7:am del día siguiente, es decir , el número de bicicletas que cada hora en un día se han tomado o dejado en cada estación. Conocemos las coordenadas x, y de cada estación. Se pide :

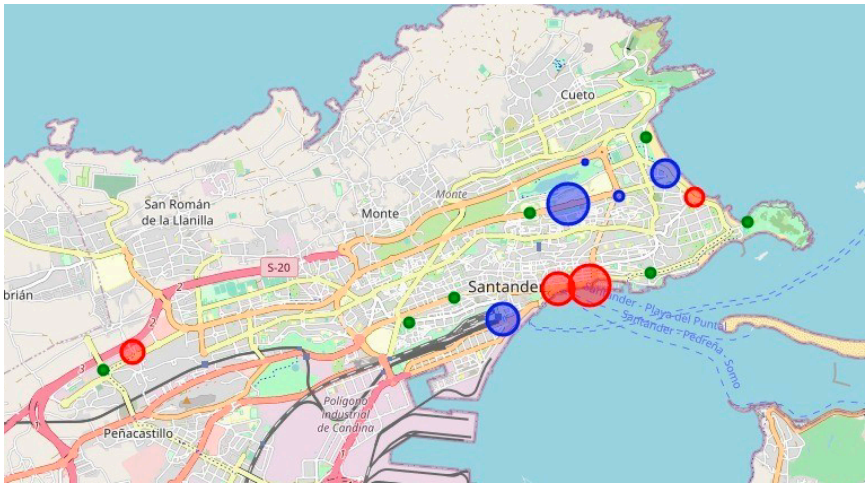
Optimizar **el número de kilómetros que recorren las personas cuando no pueden coger una bicicleta o dejarla porque la estación está vacía /llena**. Condicionantes:

- Corresponde a grabaciones de uso de abril de 2019 en Santander, con 16 estaciones.
- El número total de plazas máximas disponibles es de 220 y el mínimo de 205 .
- El número de bicicletas es desconocido, pero no supera el número de plazas.
- Asumimos que siempre va a haber una plaza o bici disponible en alguna estación.
- Los usuarios pueden consultar el estado de la capacidad en una app y dirigirse directamente a la estación más próxima con capacidad/bicis (dependiendo si queremos coger o soltar).
- La capacidad total de la solución total tiene que ser igual o inferior a 220, pero no tenemos ningún beneficio en la configuración básica del problema por tener menos plazas.
- Los kilómetros extra andando cuestan el triple que en bicicleta.
- **Avanzado: Genera una función de simulación que muestre la evolución de la misma sobre el mapa con al menos los trayectos extra y las situaciones de llenado o vaciado de las estaciones de forma dinámica (animación en un tiempo determinado)**

- **Avanzado: Guarda las situaciones especiales en un fichero csv: llenado, vaciado, desplazamiento por bici/plaza y el número de movimiento (o día/hora dependiendo de la codificación de los datos)**

- **Avanzado: Modifica la función de evaluación para que las estaciones que se estén próximas a llenarse/vaciarse cuesten dinero a los usuarios que quieran dejar/coger bicis y así fomentar la elección de otras con más disponibilidad. Razona una forma de implementarlo. ¿Mejora el número de kilómetros finales?**

- **Avanzado: pinta la situación final de la optimización respecto de la solución inicial**
- **Se ha incluido un fichero posiciones.csv con las latitudes y longitudes de las estaciones en el mismo enlace de los datos originales.**



Datos fijos de configuración del problema

Los datos se encuentran en un fichero zip en la dirección:

https://www.dropbox.com/s/b0ih9c7oflaj82e/bicicletas.zip?dl=0

Matriz Delta movimientos 16*estaciones
El primer registro muestra los Id de las estaciones, del 0 al 15.
El segundo registro es un movimiento con el estado inicial de la red al comienzo del periodo de estudio.
Del segundo en adelante se tienen los movimientos cada 5 minutos para cada estación en la forma:
(- 5, +5 ……., 0). Cojo 5 bicis de la primera, dejo 5 en la segunda , ….. no hago nada en la 16.
(-10, +8,……...-1)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5	7	13	6	8	13	8	9	6	10	10	18	8	13	15	14
0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Dado que hay muchos movimientos = 0, se podría convertir en una lista de movimientos (id, movimientos) , aunque se pierde la posibilidad de realizar los movimientos en distinto orden dentro del mismo delta, por lo que los resultados pueden ser algo peores.

Update: Los datos que tenéis has sido reducidos para conseguir tiempos de computación acorde a la capacidad de los ordenadores que tenéis.

Solución- Los movimientos debéis multiplicarlos por 2, podéis hacerlo directamente sobre los datos , o al cargarlos en el programa

Notar que tratamos con un ejemplo real, por lo que se podrán tomar decisiones de tiempo y procesamiento debidamente justificados. A medida que vayamos viendo tiempos de ejecución en cada uno de los casos se irán comentando en las clases de práctica y publicando en los foros de Moodle y actualizaciones de la práctica.

Distancias:
Para elegir la estación mas cercana hay dos matrices:

Cercanas_indices: Matriz con las estaciones más cercanas a cada índice

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	4	6	10	8	1	12	15	11	5	2	9	3	14	13
1	12	6	10	5	7	0	15	8	11	4	2	9	3	14	13
2	9	11	15	3	5	12	8	6	1	7	0	10	4	14	13
3	9	2	11	5	15	12	1	6	8	14	7	10	13	0	4
4	0	7	8	6	10	15	1	12	11	5	2	9	3	14	13
5	12	1	11	6	15	2	9	8	10	7	3	0	4	14	13
6	1	12	7	10	0	8	5	15	11	4	2	9	3	14	13
7	0	10	6	1	4	8	12	15	5	11	2	9	3	14	13
8	15	6	0	11	7	12	4	1	10	5	2	9	3	14	13
9	3	2	11	5	15	12	1	6	8	7	10	0	4	14	13
10	7	6	1	0	12	4	8	5	15	11	2	9	3	14	13
11	15	2	5	12	8	9	6	1	3	7	0	10	4	14	13
12	5	1	6	15	11	7	10	8	0	2	9	4	3	14	13
13	14	3	9	2	5	11	12	15	1	6	8	10	7	0	4
14	13	3	9	2	5	11	12	15	1	6	8	10	7	0	4
15	11	8	12	6	2	5	1	7	0	9	10	4	3	14	13

Cercanas_kms: distancia de cada una.

0	1	2	3	4	5
0.0	0.4612275771434886	0.716556975483184	0.9313621873428618	0.9477275970723462	1.0635186361556144
0.0	0.5530185954345544	0.5939211595070305	0.819978299475489	0.9210339622316458	1.0045346448337067
0.0	0.661118187182893	0.7719908030770463	1.1490439477242063	1.1609208865928937	1.3478521492750146
0.0	0.6315818426361309	1.1609208865928937	1.883933147494817	2.007667350665301	2.2781766815791635
0.0	0.716556975483184	1.1686787287120894	1.3444067143388139	1.6148276615883383	1.6188773495262376
0.0	0.4897223724780759	0.9210339622316458	1.0037473059397939	1.1220529184477213	1.1643931713197897
0.0	0.5939211595070305	0.6328831596461355	0.6335764884966517	0.7987412528665032	0.9313621873428618
0.0	0.4612275771434886	0.5073157312018843	0.6335764884966517	1.0045346448337067	1.1686787287120894

De esta forma no hay que buscar la más cercana en cada iteración ya que esta es una información que puede ser pre calculada una sola vez

Base

El siguiente código es total o parcialmente reutilizable. Códifica con la suficiente generalización para su uso en los algoritmos de esta y siguientes prácticas con la denominación , parámetros y retorno que estimes oportuno. Ten en cuenta el uso de semillas preestablecidas para el generador de números aleatorios.

Las soluciones deben ser siempre válidas dentro de los mínimos y máximos de capacidad

En las pruebas iniciales de los algoritmos prueba con un número menor de movimientos. Estimaremos en clase si es necesario tomar menos días para la simulación si el procesamiento resulta excesivo para algún algoritmo en particular.

a) Función de evaluación .
b) Generación de la solución inicial.
c) Operador de movimiento . El operador de movimiento debe elegir dos posiciones aleatorias del vector de capacidades y mover un número de espacios de aparcamiento de una a otra.

En el primer mejor vecino, debe asegurarse de que, en caso de que no se encuentre un mejor vecino, se explore todo el entorno de esta solución. Para otros algoritmos, las dos posiciones se generarán aleatoriamente, por lo que la función base puede tomar dos índices como parámetros para que sea reutilizable.

Algoritmo de comparación: *Greedy*

Para efectuar la comparativa de resultados entre los distintos algoritmos de búsqueda, se debe implementar como algoritmo básico, un *Greedy*, es decir, siguiendo la heurística de asignar las capacidades proporcionalmente al vector inicial de estado prestando atención a que la cantidad total de plazas sea la indicada.

Búsqueda Aleatoria

El Algoritmo de Búsqueda Aleatoria (BA) consistirá en generar aleatoriamente una solución en cada iteración debiéndose ejecutar 100 iteraciones con cada semilla devolviendo la mejor de las iteraciones.

La búsqueda aleatoria completa debe ejecutarse 5 veces, cada vez con una semilla distinta (por tanto, se deben anotar las 5 semillas que se utilizarán sistemáticamente), para el generador aleatorio.

Búsquedas Locales

Se implementará siguiendo el esquema de ***el primer mejor*** vecino, según el Tema 1 de teoría.

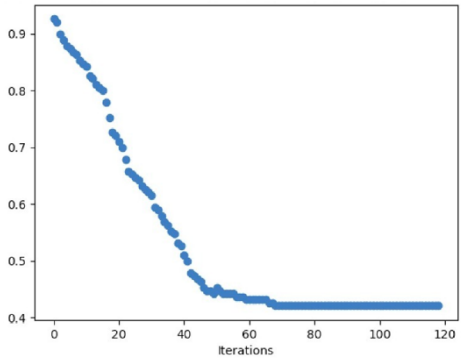
Se partirá de una solución inicial aleatoria. Los algoritmos de búsqueda local tienen su propia condición de parada, pero adicionalmente, en prevención de tiempos excesivos en algún caso, se añadirá una condición de parada alternativa (OR) basada en el número de evaluaciones que esté realizando la búsqueda, es decir, el número de veces se llame al cálculo de la función de coste. Este valor para la Búsqueda Local será de 3000 llamadas a la función de coste.

La búsqueda con cada algoritmo se debe ejecutar 5 veces con semillas distintas, como en el caso de la Aleatoria.

A) No se va a realizar la implementación del mejor vecino por motivos de rendimiento. Estima en base a la ejecución de la función de evaluación cuantos vecinos habría que generar con una capacidad máxima de la estación de 40 y cuanto tardaría en base a la función de evaluación actual.

B) Haz un estudio de los distintos parámetros de la búsqueda local para hacer que esta converja de manera adecuada en un tiempo razonable.
Para ello deberá mostrar varios gráficos de mejor solución (total de kilómetros recorridos) en cada iteración para distintas situaciones y otro de tiempos. Justificar la elección final con una tabla con distintos valores marcando en negrita los valores elegidos.

I. Granularidad del operador de movimiento (cuanto cambia una solución respecto a un vecino)
II. Número de vecinos que considera adecuado.
III. ¿Que medidas/simplificaciones podemos tomar para alcanzar un tiempo razonable de ejecución? (en torno a 20 minutos por ejecución)



Enfriamiento Simulado

Se ha de implementar el algoritmo de Enfriamiento Simulado (ES) con las siguientes componentes:

- *Esquema de enfriamiento*: Se implementará el esquema de *Cauchy*, *Tk* =

$$T0/(1 + k)$$

- *Condición de enfriamiento L*(*T*): Se enfriará la temperatura, y finalizará a la iteración actual, cuando se haya generado un número máximo de vecinos (independientemente de si han sido o no aceptados). **Determinar este parámetro con la experimentación de al menos tres valores en el entorno de 20 vecinos.**

- *Condición de parada*: El algoritmo finalizará cuando se alcance un número máximo de iteraciones (enfriamientos). **Determinar este parámetro con la experimentación de al menos tres valores en el entorno de 80 iteraciones**.

Se calculará la temperatura inicial en función de la siguiente fórmula:

$$T_0 = \frac{\mu}{-\log(\phi)} \cdot C(S_i)$$

donde *T*0 es la temperatura inicial, *C*(*Si*) es el costo de la solución inicial y Φ[0,1] es la probabilidad de aceptar una solución un μ por 1 peor que la inicial. En las ejecuciones se considerar Φ,μ, entre 0.1 y 0.3 mediante una pequeña experimentación que determine el que el número de soluciones iniciales no aceptadas sea de un 20% para T0, tomando C(s) como el coste de la solución Greedy.

Se debe repetir también 5 veces con distintas semillas, partiendo de una solución inicial aleatoria.

Para las experimentaciones previas se puede tomar un conjunto de datos reducido siempre que los resultados sean extrapolables

Búsqueda Tabú

Se implementará la versión de BT utilizando una lista de movimientos tabú y tres estrategias de reinicialización.

Sus principales características son:

- Estrategia de selección de vecino: Consistirá en examinar 40 vecinos para coger el mejor de acuerdo a los criterios tabú.
- Codificación Matriz Frecuencia largo plazo~~tabú~~: codificación de valores en cada estación. Cuantas veces toma un valor una estación. Para que el greedy tenga sentido el número de columnas no debe ser elevado, se recomienda utilizar rangos de valores en vez de valores individuales por ejemplo [0-5, 6-10, ……..45-50]
- Lista Tabú: se admitirá cualquier propuesta con suficiente capacidad de restricción de los movimientos. Una opción válida para la codificación tabú puede ser tener en cuenta sólo que los movimientos realizados no sean los mismos.
  Si partiendo del vector [5, 7, 8, 10] , cambiamos posición 2 y 3 con un movimiento de 2 quedarán --> [5, 9, 6, 19] , la codificación tabú por movimientos --> [2, 3] cualquier vecino que no incluya cambios de uno de esos indices o cumpla el criterio de aspiración sería un vecino válido.
- Selección de estrategias de reinicialización: La probabilidad de escoger la reinicialización construyendo una solución inicial aleatoria es 0,25, la de usar la memoria a largo plazo al generar una nueva solución greedy es 0,5, y la de utilizar la reinicialización desde la mejor solución obtenida es 0,25.
- La solución greedy debe generar soluciones con mayor probabilidad para los valores que menos veces se han producido en cada estación.
  Para cada estación se deberá calcular las inversas de los valores acumulados y luego normalizar para tener valores entre 0-1 correspondiente a su probabilidad. Se tirará un dado y se elige el primer valor que supera dicho valor añadiéndose a la solución greedy

  Ejemplo:

  Si tenemos una estación Ej con tres posibles valores valores(v1, v2 ,v3) con los siguientes valores de frecuencia acumulada :
  matrizFrecuencias(…….
  2, 3, 5
  ……..).

  1. Calcular inversas (…1/2, 1/3 , 1/5…)
  2. Calcular suma total 1,03333…
  3. Normalizar dividiendo por la suma de las inversas : 0.48, 0.32, 0.19

  *Atención: usar una resolución alta de decimales o tener en cuenta la posibilidad de que el algoritmo pueda salir sin encontrar el último valor.

Una vez hecho esto todas las posiciones de la matriz *matrizProbabilidades* tienen su probabilidad directa y se pasa a construir un greedy probabilístico a las proporciones de esta matriz.

```
soluciónGreedy = {}
Para cada estación en estaciones
    Número = random(0,1)
    suma=0
    Para i = valor en valores
        Suma+= matrizProbabilidades (estación, i)
        If número < suma
            soluciónGreedy.add(valor)
             break()

return soluciónGreedy
```

Estimar el número máximo de iteraciones en total. Se realizarán 4 reinicializaciones, es decir, una cada Numtotal-iteraciones/4. El tamaño inicial de cada lista tabú será n=4, estos valores cambiarán después de las reinicializaciones según se ha comentado.

Además de saltar a una solución concreta según las reinicializaciones comentadas, también se alterará un parámetro del algoritmo de búsqueda para provocar un cambio de comportamiento del algoritmo más efectivo. Este consistirá en variar el tamaño de la lista tabú, incrementándola o reduciéndola en un 50% según una decisión aleatoria uniforme, empezando la lista desde vacío en cada reinicialización.

Dado el carácter probabilístico del algoritmo, deberá ejecutarse 5 veces (con semillas distintas para el generador aleatorio, para cada caso del problema (*dataset*).

Metodología de Comparación

El alumno tendrá que contabilizar el número de evaluaciones (llamadas realizadas a la función de coste) producidas por los distintos algoritmos, que será empleado como una medida adicional de comparación de su calidad.

A partir de la experimentación efectuada, se construirá una tabla global de resultados con la estructura mostrada en la Tabla mostrada. ~~El dato entre paréntesis debajo del nombre de la instancia en la Tabla semuestra el coste de la solución óptima (o la mejor encontrada que se conoce a priori) de la instancia, valor que debe ser considerado en el análisis de resultados~~. La columna etiquetada con *Mej* indica el valor de la mejor solución encontrada, la columna σ refiere a la desviación típica y la columna etiquetada con *Ev y Ev. Mejor* indica respectivamente el número medio y mínimo de evaluaciones realizadas por el algoritmo en las cinco ejecuciones (salvo en el caso del algoritmo *Greedy* que sólo se ejecuta una vez).

Algoritmo	**Ev.Medias**	**Ev. Mejor**	**σ EV**	**Mejor Kms**	**Media Kms**	**σ.Kms**
Greedy						
Aleatoria						
BL Primer						
Enf. Simulado						
Tabú						

A partir de los datos mostrados en estas tablas, el alumno realizará un **análisis de los resultados obtenidos**, **que influirá de forma decisiva en la calificación de la práctica**. En dicho análisis, se deben comparar, para cada instancia, las distintas variantes de los algoritmos en términos de: número de evaluaciones, mejor resultado individual obtenido y mejor resultado medio (robustez del algoritmo).

Las prácticas se realizarán individualmente.

Fecha y Método de Entrega;

El día 4 del Abril durante la sesión de prácticas. Debe entregar 1 fichero comprimido ZIP, que contenga:

- Documento DOC (MS Word) ó PDF con las tablas de resultados y análisis.
- Ficheros de código fuente completo ejecutable utilizado.
- Scripts, si los ha utilizado.
- Se realizará una defensa de la práctica explicando cada una de las decisiones tomadas en el código. La evaluación tendrá en cuenta:
  Nivel Básico (4):
  - Corrección del algoritmo
  - Utilización de la terminología apropiada (la utilizada en el material de clase)
  Nivel intermedio(5-6):
  - Análisis e interpretación sobre la capacidad de exploración/explotación de cada algoritmo relacionado con los resultados obtenidos.
  - Gráficas y ejemplos concretos comparados.
  Nivel Alto (7-8)
  - Mejoras en el rendimiento de la aplicación
  - Resultados obtenidos razonablemente cercanos al optimo
  - Análisis de la estadística de los resultados. Algoritmos más estables. Desviación típica.
  Nivel Avanzado(9-10):
  - Función de simulación que muestre la evolución .
  - Guardado en Fichero
  - Función de fitness mejorada
  - Solución final dibujada

Permanezca atento a posibles nuevas versiones mejoradas de este documento.