Grado en Ingeniería Informática

Metaheurísticas

Curso 2019/2020

Universidad de Jaén

Práctica 1

1. Objetivos

El objetivo de esta práctica es estudiar el funcionamiento de Algoritmos Greedy, *Técnicas de Búsqueda Local* y Metaheurísticas basadas en trayectorias en la resolución del problema descrito en las transparencias del Seminario 2. Para ello, se requerirá que el estudiante implemente los algoritmos a dicho problema:

- Algoritmo Greedy básico (2 puntos).
- Algoritmo de búsqueda local del mejor (3 puntos).
- Búsqueda tabú (5 puntos).

La fecha límite de entrega será el **viernes 25 de octubre de 2019 antes de las 23:55 horas**. La entrega se realizará a través de la plataforma virtual.

(*) Entregas fuera de fecha se considerarán fuera de entrega ordinaria.

2. Trabajo a realizar

El estudiante podrá desarrollar los algoritmos de la práctica siguiendo la modalidad que desee: trabajando con cualquiera de los *frameworks* de metaheurísticas estudiados en el Seminario 1, implementándolos a partir de código propio o considerando cualquier código disponible en Internet[1].

Los algoritmos desarrollados serán probados sobre una serie de casos del problema. Se realizará un estudio comparativo de los resultados obtenidos y se analizará el comportamiento de cada algoritmo en base a dichos resultados. **Este análisis influirá decisivamente en la calificación final de la práctica**.

En las secciones siguientes se describen el problema y los casos considerados, los aspectos relacionados con cada algoritmo a desarrollar y las tablas de resultados a obtener.

[1] Cualquier código recuperado o modificado deberá ser indicado con claridad en la documentación y al profesor.

3. Casos de estudio

Dentro del material de las prácticas se encuentras disponibles 2 configuraciones posibles para los aeropuertos de Málaga y Madrid. En concreto, el aeropuerto de Málaga tiene una composición asimétrica en sus datos, mientras que Madrid es simétrico.

El formato de los ficheros puede ser analizado en el documento del Seminario 2. Para cada problema se cuenta con el fichero de datos (.dat) y el fichero con la mejor solución encontrada (.sln). **Es imposible que se puedan encontrar (por vuestros algoritmos) soluciones mejores que la mejor encontrada hasta el momento.**

La forma en que habitualmente se estima la calidad de un algoritmo es la siguiente: se ejecuta sobre un conjunto de ejemplos y se comparan los resultados obtenidos con las mejores soluciones conocidas para dichos problemas.

Además, los algoritmos pueden tener diversos parámetros o pueden emplear diversas estrategias. Para determinar qué valor es el más adecuado para un parámetro o saber la estrategia más efectiva los algoritmos también se comparan entre sí.

La comparación de los algoritmos se lleva a cabo fundamentalmente usando dos criterios, la *calidad* de las soluciones obtenidas y el *tiempo de ejecución* empleado para conseguirlas. Además, es posible que los algoritmos no se comporten de la misma forma si se ejecutan sobre un conjunto de instancias u otro.

Por otro lado, a diferencia de los algoritmos determinísticos, los algoritmos probabilísticos se caracterizan por la toma de decisiones aleatorias a lo largo de su ejecución. Este hecho implica que un mismo algoritmo probabilístico aplicado al mismo caso de un problema pueda comportarse de forma diferente y por tanto proporcionar resultados distintos en cada ejecución. Cuando se analiza el comportamiento de una metaheurística probabilística en un caso de un problema, se desearía que el resultado obtenido no estuviera sesgado por una secuencia aleatoria concreta que pueda influir positiva o negativamente en las decisiones tomadas durante su ejecución.

Dada la influencia de la aleatoriedad en el proceso, es recomendable disponer de un generador de secuencia pseudoaleatoria de buena calidad con el que, dado un valor semilla de inicialización, se obtengan números en una secuencia

lo suficientemente grande (es decir, que no se repitan los números en un margen razonable) como para considerarse aleatoria. Dentro del material de prácticas de la asignatura se puede encontrar una implementación en lenguaje C de un generador aleatorio de buena calidad (*random.h*). En resumen, para el análisis del comportamiento de una metaheurística estocástica es necesario realizar varias ejecuciones con diferentes semillas.

Como norma general, el proceso a seguir consiste en realizar un número de ejecuciones diferentes de cada algoritmo probabilístico considerado para cada caso del problema. Es necesario asegurarse de que se realizan diferentes secuencias aleatorias en dichas ejecuciones. Así, el valor de la semilla que determina la inicialización de cada secuencia deberá ser distinto en cada ejecución y estas semillas deben mantenerse en los distintos algoritmos (es decir, la semilla para la primera ejecución de todos los algoritmos debe ser la misma, la de la segunda también debe ser la misma y distinta de la anterior, etc.). Para mostrar los resultados obtenidos con cada algoritmo en el que se hayan realizado varias ejecuciones, se deben construir tablas que recojan los valores correspondientes a estadísticos como el **mejor** y **peor** resultado para cada caso del problema, así como la **media** y la **desviación típica** de todas las ejecuciones. Alternativamente, se pueden emplear descripciones más representativas como los *boxplots*, que proporcionan información de todas las ejecuciones realizadas mostrando mínimo, máximo, mediana y primer y tercer cuartil de forma gráfica, Finalmente, se construirán unas tablas globales con los resultados agregados que mostrarán la calidad del algoritmo en la resolución del problema desde un punto de vista general.

Este será el procedimiento que aplicaremos en las prácticas. Como norma general, el alumno realizará **5 ejecuciones diferentes** de cada algoritmo probabilístico considerado para cada caso del problema. Se considerará como **semilla inicial el número del DNI del alumno**. En la tabla 2 se incluye un ejemplo sobre los posibles valores de la semilla para el generador aleatorio.

Tabla 2: Ejemplo de posibles semillas a utilizar en el generador aleatorio

Ejecución	**Semilla**
1	12345678
2	23456781
3	34567812
4	45678123
5	56781234

4. Algoritmos

Los algoritmos a implementar en esta práctica se describen en las siguientes secciones

4.1. Algoritmo Greedy

Se debe resolver el problema planteado mediante una solución Greedy. En concreto, vamos a utilizar la idea de asociar unidades de montaje de gran flujo de piezas con localizaciones céntricas en la red y viceversa, es decir, cuántas más piezas mueva una unidad más céntrica debería estar para mejorar el coste.

Se calculan dos vectores, el potencial de flujo y el potencial de distancia, que se componen respectivamente de:

- la sumatoria de flujos de una unidad al resto ($f_i = \sum_{j=1}^{n} f_{ij}$)
- y de distancias desde una localización al resto ($d_k = \sum_{l=1}^{n} d_{kl}$).

Para una unidad de montaje concreta, cuanto mayor sea f_i más importante es la unidad en el intercambio de flujos. Por otro lado, cuanto menor sea d_k más céntrica es una localización concreta. Por tanto, el algoritmo irá seleccionando la unidad *i* libre con mayor f_i y le asignará la localización *k* libre con menor d_k.

4.2. Búsqueda Local del mejor

En esta práctica, se considera la modalidad de algoritmo de búsqueda local: **la búsqueda del mejor**. De forma que se buscarán hasta 10 posibles movimientos y nos quedaremos con el mejor de todos. En caso que tras 100 intentos de posibles movimientos no consigamos mejorar al estado actual el algoritmo finalizará. La inicialización de la solución inicial será completamente aleatoria.

Se empleará el movimiento de intercambio *Int(π,r,s)* que intercambia las localizaciones asignadas a las unidades *r* y *s* en la permutación π. Será obligatorio emplear la **factorización** de la función objetivo en todos los casos. Una vez realizado el movimiento, se actualizan la solución actual, y se comienza a explorar el nuevo entorno.

Finalmente, se detendrá la ejecución **cuando se hayan realizado 50000 evaluaciones de la función objetivo**, es decir, en cuanto se cumpla esta condición o la mencionada anteriormente. Se realizará una única ejecución sobre cada caso del problema.

4.3. Búsqueda tabú

Para realizar este algoritmo se partirá del algoritmo implementado previamente en la búsqueda local del mejor y se tendrán en cuenta las siguientes condiciones:

- En este algoritmo siempre habrá movimiento en cada iteración, por lo que el algoritmo debe llegar a las 50000 iteraciones.
- Se debe implementar un sistema de memorias adaptativas idóneo para resolver el problema del Algoritmo de Búsqueda Tabú.
- La tenencia tabú será de 5 iteraciones.
- En cada iteración/movimiento se deben actualizar las estructuras necesarias para considerar la memoria a largo y corto plazo.
- ~~El criterio de aspiración será válido si el coste obtenido es mejor o igual que el mejor hasta el momento.~~
- Reinicialización cuando se cumpla 100 intentos sin movimientos, se debe aplicar con igual probabilidad (50%):
    - Diversificar: Debe de crear una solución, basándose en la memoria a largo plazo, para buscar zonas no exploradas hasta el momento.
    - Intensificar: Ir a zonas con mayor frecuencia de aparición.

(*) Se debe realizar un estudio del algoritmo para ver:

- ¿qué mecanismo se aplica más número de veces?
- ¿cuál tiene un mejor funcionamiento?
- ¿en qué momento se aplican y por qué funciona mejor?

5. Tablas de resultados

Se diseñará una tabla para cada algoritmo (Greedy, BL y BT) donde se recogerán los resultados de la ejecución de dicho algoritmo en los conjuntos de datos considerados.

Para facilitar la comparación de algoritmos en la práctica se considerarán dos estadísticos distintos denominados Desv y Tiempo:

- *Desv* se calcula como la media de las desviaciones, en porcentaje, del valor obtenido por cada método en cada instancia respecto al mejor valor conocido para ese caso que lo tenemos en los datos entregados.

$$\frac{1}{|casos|} \cdot \sum_{i \in casos} 100 \cdot \frac{valorAlgoritmo_i - mejorValor_i}{mejorValor_i}$$

  De esta forma, no se tiene en cuenta el valor concreto de una solución para una instancia, sino que se determina lo cerca que se está de obtener la mejor solución conocida.

- *Tiempo* se calcula como la media del tiempo de ejecución empleado por el algoritmo para resolver cada caso del problema.

Cuanto menor es el valor de *Desv* para un algoritmo, mejor calidad tiene dicho algoritmo, porque en media obtiene soluciones más cercanas al mejor valor conocido. Por otro lado, si dos métodos obtienen soluciones de la misma calidad (tienen valores de *Desv* similares), uno será mejor que el otro si emplea menos tiempo en media. Dentro del material hay también disponible un código en C (timer) para un cálculo adecuado del tiempo de ejecución de los algoritmos metaheurísticos.

La Tabla resultado tendrá la misma estructura que la Tabla 5.1.

Tabla 5.1: Resultados obtenidos por el algoritmo X en el problema

	Malaga01		Malaga02		Malaga03		…		Madrid04	
	S	Tiempo	*S*	Tiempo	*S*	Tiempo	*S*	Tiempo	*S*	Tiempo
Ejec 1										
Ejec 2										
Ejec 3										
Ejec 4										
Ejec 5										
Media										
Desv. típica										

Finalmente, se construirá una tabla de resultados global que recoja los resultados medios de calidad y tiempo para todos los algoritmos considerados, tal como se muestra en la Tabla 5.2. Para rellenar esta tabla se hará uso de los resultados mostrados en las tablas parciales, siendo m la media y σ la desviación típica.

Tabla 5.2: Resultados globales del problema

	Malaga01		Malaga02		….		Madrid04	
	C	Tiempo	*C*	Tiempo	*C*	Tiempo	*C*	Tiempo
Greedy	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)
BL mejor	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)
BT	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)	m (σ)

A partir de los datos mostrados en estas tablas, el alumno realizará un análisis de los resultados obtenidos, que influirá significativamente en la calificación de la práctica. En dicho análisis se deben comparar los distintos algoritmos en

términos del mejor resultado individual obtenido (capacidad del algoritmo para obtener soluciones de calidad), los resultados medios (robustez del algoritmo) y el tiempo requerido para obtener las soluciones (rapidez del algoritmo). Se comparará el rendimiento de las metaheurísticas entre sí, así como con respecto a la mejor solución.

La hoja Excel *Tablas_MH19.xls*, disponible en el material de prácticas de la asignatura, ayudará en la confección de las distintas tablas.

6. Documentación y normativa

Lee atentamente la normativa de entrega de prácticas:

- Solo se admitirá el formato **ZIP**. No se corregirán guiones en cualquier otro formato al indicado.

- La memoria que acompañará en la entrega será exclusivamente en formato **PDF** y se incluirá una portada con:
  - Identificación de los dos alumnos (Nombre, apellidos y DNI).
  - Identificación del guión.
  - Identificación del grupo de los alumnos.
  - Algoritmos implementados en la práctica.

- Los nombres de los ficheros tendrán el siguiente formato "*Ape11-Ape12-Ape21-Ape22-GuionX.zip*" y dentro la "*Documentacion.pdf*" junto con el código que se detalla más adelante donde:
  - Ape11 es el primer apellido del primer alumno.
  - Ape12 es el segundo apellido del primer alumno.
  - Ape21 es el primer apellido del segundo alumno.
  - Ape22 es el segundo apellido del segundo alumno.
  - X es el número del guión.

- Los ficheros de los algoritmos deberán tener el siguiente formato ~~"*InicialApe11-InicialApe12-InicialApe21-InicialApe22-GuionX-AlgXX.java*"~~

  "AlgXX-ClaseXX-GrupoXX.java"

- Una breve descripción de 1-2 páginas donde se detalle el problema y los algoritmos empleados, así como **todas las restricciones y consideraciones empleadas por los alumnos para la implementación de las metaheurísticas solicitadas en el guión: esquema de representación de soluciones, descripción del código, función objetivo, operadores comunes, etc.**

- Descripción en **pseudocódigo** de la **estructura del método de búsqueda** y de todas aquellas **operaciones relevantes** de cada algoritmo. Este contenido, específico a cada algoritmo se detallará en los correspondientes guiones de prácticas. El pseudocódigo **deberá forzosamente reflejar la implementación y el desarrollo realizados** y no ser una descripción genérica extraída de las transparencias de clase o de cualquier otra fuente. La descripción de cada algoritmo no deberá ocupar más de **2 páginas**.

- Experimentos y análisis de resultados:
    - Descripción de los valores de los parámetros considerados en las ejecuciones de cada algoritmo (**incluyendo las semillas utilizadas**).
    - Resultados obtenidos según el formato especificado.
    - Análisis de resultados. El análisis deberá estar orientado a **justificar** (según el comportamiento de cada algoritmo) **los resultados** obtenidos en lugar de realizar una mera "lectura" de las tablas. Se valorará la inclusión de otros elementos de comparación tales como gráficas de convergencia, boxplots, análisis comparativo de las soluciones obtenidas, representación gráfica de las soluciones, etc.
    - Análisis de tiempos y desviaciones, e influencia del tamaño de la instancia en los resultados obtenidos, etc.
    - Análisis de la solución mejor obtenida con una explicación de qué significa.
    - Por último, es clave obtener un fichero de registro que vaya almacenando la información de la ejecución del algoritmo indicando las soluciones y costes que va obteniendo en las distintas iteraciones siempre que haya un cambio a mejor, o incluso a peor. **Se deberá subir los ficheros "log" al drive e indicar en el informe el directorio para consultarlos.**

- Referencias bibliográficas u otro tipo de material distinto del proporcionado en la asignatura que se haya consultado para realizar la práctica (en caso de haberlo hecho).

Aunque lo esencial es el contenido, también debe cuidarse la presentación y la redacción. **La documentación nunca deberá incluir listado total o parcial del código fuente**.

En lo referente al **desarrollo de la práctica**, se entregará una carpeta llamada **software** que contenga una versión ejecutable de los programas desarrollados, así como los ficheros de datos de los casos del problema y el código fuente implementado o los ficheros de configuración del framework empleado. El código fuente o los ficheros de configuración se organizarán en la estructura de

directorios que sea necesaria y deberán colgar del directorio FUENTES en el raíz. Junto con el código fuente, hay que incluir los ficheros necesarios para construir los ejecutables según el entorno de desarrollo empleado (tales como *.prj, makefile, *.ide, etc.). La versión ejecutable de los programas y los ficheros de datos se incluirán en un subdirectorio del raíz de nombre BIN. En este mismo directorio se adjuntará un pequeño fichero de texto de nombre LEEME que contendrá breves reseñas sobre cada fichero incluido en el directorio. Es importante que los programas realizados puedan leer los valores de los parámetros de los algoritmos desde fichero, es decir, que no tengan que ser recompilados para cambiar éstos ante una nueva ejecución. Por ejemplo, la semilla que inicializa la secuencia pseudoaleatoria debería poder especificarse como un parámetro más.

En caso de que el comportamiento del algoritmo en la versión implementada/desarrollada no coincida con la descripción en pseudocódigo o no incorpore las componentes requeridas, se podría reducir hasta en un 50% la calificación del algoritmo correspondiente.

7. Anexo: Construcción de la búsqueda del mejor

La representación del problema se realiza mediante una permutación

$\pi=[\pi(1), \ldots,\pi(n)]$

Donde las posiciones del vector i=1, 2, …, n representan las unidades de montaje y las valores $\pi(n)$ contenidos en ellas las localizaciones de esas unidades.

Mientras que un operador de vecino de intercambio y su entorno se considera que en una solución π está formado por las soluciones accesibles desde ella a través de un movimiento de intercambio. Por ejemplo, data una solución cualquiera se escogen dos unidades distintas y se intercambia la localización asignada a cada una de ellas (*Int*(π,i,j)), tal que:

$$\pi=[\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n)]$$
$$\pi'=[\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots, \pi(n)]$$

Es importante verificar la restricción del operador done si la solución π es factible siempre genera una solución vecina π' factible.

La aplicación provoca que el tamaño del entorno sea:

$$|E(\pi)| = \frac{n(n-1)}{2}$$

Los ejemplos con los que contamos en nuestro problema no suelen ser demasiado grandes y el cálculo factorizado del coste de una solución se realiza de forma eficiente (O(n)), permitiendo explorar el espacio completo. Sin embargo, discha exploración requería $O(n^3)$ por lo que es recomendable utilizar una estrategia avanzada, considerando una modalidad de lista de candidatos y seleccionando primero los movimientos más prometedores.

Para el algoritmo de búsqueda local del primer mejor se considera que en cuanto se genera una solución vecina que mejora a la actual, se aplica el movimiento y se pasa a la siguiente iteración. De esta forma, se detiene la búsqueda cuando se ha explorado el vecindario completo sin obtener una mejora.

- Factorización: Se considera para calcular el coste de π' a partir de π considerando únicamente los cambios realizados en el movimiento de intercambio

7.1. Factorización del movimiento de intercambio

Sea C(π) el coste de la solución original π. Para generar π' el operador de vecino Int(π,r,s) escoge dos unidades r y s e intercambiar sus localizaciones π(r) y π(s):

$$\pi' = [\pi(1), \ldots, \pi(s), \ldots, \pi(r), \ldots, \pi(n)]$$

De esta forma quedando afectados 2·n sumandos, los relacionados con las dos viejas y las dos nuevas localizaciones de las dos unidades alteradas, considerando un coste del movimiento igual a la diferencia de costes entre las dos soluciones que se puede factorizar como:

nuevas

viejas

$$\Delta C(\pi, r, s) = C(\pi') - C(\pi)$$

$$\sum_{k=1,k\neq r,s}^{n} \left[p_{rk} \cdot \left(d_{\pi(s)\pi(k)} - d_{\pi(r)\pi(k)} \right) + p_{sk} \cdot \left(d_{\pi(r)\pi(k)} - d_{\pi(s)\pi(k)} \right) + p_{kr} \cdot \left(d_{\pi(k)\pi(s)} - d_{\pi(k)\pi(r)} \right) + p_{ks} \cdot \left(d_{\pi(k)\pi(r)} - d_{\pi(k)\pi(s)} \right) \right]$$

En caso de que el valor sea negativo quiere decir que la nueva solución es mejor que la anterior y como estamos en un problema de minimización se aceptaría, en caso contrario se descarta.