UNIVERSIDAD DE GRANADA

E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Metaheurísticas

http://sci2s.ugr.es/graduateCourses/Metaheuristicas
https://decsai.ugr.es

Guión de Prácticas

Práctica 1.a:
Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema de la Mochila Cuadrática

Curso 2023-24

Tercer Curso del Grado en Ingeniería Informática

Práctica 1.a

Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema de la Mochila Cuadrática (QKP)

1. Objetivos

El objetivo de esta práctica es estudiar el funcionamiento de las *Técnicas de Búsqueda Local y de los Algoritmos Greedy* en la resolución del problema del problema de la mochila cuadrático (QKP) descrito en las transparencias del Seminario 2. Para ello, se requerirá que el estudiante adapte los siguientes algoritmos a dicho problema:

- Algoritmo Greedy básico.
- Algoritmo de Búsqueda Local (BL).

El estudiante deberá comparar los resultados obtenidos con las estimaciones existentes para el valor de los óptimos de una serie de casos del problema.

La práctica se evalúa sobre un total de **2 puntos**, distribuidos de la siguiente forma:

- BL (**1.25 puntos**).
- Greedy (**0.75 puntos**).

La fecha límite de entrega será el **Domingo 7 de Abril de 2024** antes de las 23:59 horas. La entrega de la práctica se realizará por internet a través del espacio de la asignatura en PRADO.

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 del código C proporcionado en la web de la asignatura o considerando cualquier código disponible en Internet.

Los métodos desarrollados serán ejecutados 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 los aspectos relacionados con cada algoritmo a desarrollar y las tablas de resultados a obtener.

3. Problema y Casos Considerados

3.1. Introducción al Problema de la Mochila Cuadrática

El problema de la mochila cuadrática (en inglés, *quadratic knapsack problem*, QAP) es uno de los problemas de optimización combinatoria más conocidos. En él se dispone de *n* objetos y una capacidad máxima. Cada objeto presenta un beneficio y, a diferencia del problema sencillo de la mochila, también existe un beneficio por pares de objetiso. El problema consiste en encontrar la selección óptima de objetos de forma que el beneficio total sea máximo cumpliendo la restricción que la suma de los pesos de todos los objetos elegidos sea inferior a una capacidad máxima. La nomenclatura "cuadrática" proviene del hecho de que haya interacciones entre objetos, es decir, que aparte de un vector de beneficios de los objetos, hay una matriz cuadrada de los beneficios combinados. El QKP se puede formular como:

$$\max\left(\sum_{i=1}^{n} p_i \cdot x_i + \sum_{i=1}^{n}\sum_{j=1,\, j\neq i}^{n} p_{ij} \cdot x_i \cdot x_j\right) \text{ sujeto a } x_i \in \{0,1\}, \sum_{i=1}^{n} w_i \cdot x_i \leq W.$$

donde:

- Xi representa si el objeto i es incluído en la mochila.
- Pi representa el beneficio del objeto i (sin considerar el resto).
- Pij representa el beneficio combinado de escoger los objetos Xi y Xj.
- W representa la capacidad máxima de la mochila.

3.2. Casos del QKP Considerados

Se utilizarán 97 **casos** seleccionados de varios de los conjuntos de instancias disponibles en la web https://cedric.cnam.fr/~soutif/QKP/QKP.html. Aunque los ficheros de los casos pueden ser descargados de la web anterior, se han incluido también para facilitar el trabajo del estudiante. El tamaño de los casos seleccionados varía entre 100 y 300.

El formato de los ficheros es el siguiente:

- Una primera línea donde se indica el nombre del fichero.
- El tamaño *n* de elementos.
- Un vector separado por espacios de *n* elementos representando el beneficio individual (Pi).
- *n* líneas que muestran el contenido de la matriz de beneficios por interdependencias entre variables (Pij). Dado que la matriz es simétrica, se muestra únicamente la submatriz superior (es decir, cuando j > i).
- Una línea en blanco.
- Un 0.
- La capacidad de la mochila (W).
- Los coeficientes de los pesos de cada objeto como un vector separado por espacios (Wi).
- Algunas líneas de texto, a ignorar.

Ejemplo:

```
R_10_100_13
10
91 78 22 4 48 85 46 81 3 26
55 23 35 44 5 91 95 26 40
92 11 20 43 71 83 27 65
7 57 33 38 57 63 82
100 87 91 83 44 48
69 57 79 89 21
9 40 22 26
50 6 7
71 52
17

0
145
34 33 12 3 43 26 10 2 48 39

Comments

Density : 100.00 %
Seed : 13
```

4. Algoritmos

4.1. Algoritmo *Greedy*-QKP

El algoritmo *greedy* del QKP es muy competitivo. Se basa en la heurística de asignar en cada paso aquel que maximice el ratio entre el beneficio y el peso. Para ello, primero se asigna aquel objeto que maximice Pi. Luego, en cada paso primero elimina aquellos cuyo peso hiciese exceder la capacidad máxima, y los restantes los clasifica en orden decreciente del valor potencial por unidad de peso y elige aquel con mayor valor. Es decir, aquel que maximice el ratio siguiente:

$$\frac{\left(p_i + \sum_{j=1, i \neq j}^{n} P_{ij}\right)}{w_i}$$

Por tanto, el potencial beneficio de cada objeto se irá actualizando por cada nuevo objeto incorporado.

4.2. Búsqueda Local

El algoritmo de BL tiene las siguientes componentes, varias de las cuales serán comunes a los algoritmos metaheurísticos de las siguientes prácticas:

- *Esquema de representación*: Se seguirá la representación booleana o entera de tamaño n que contendrá en la posición *i* un 0 ó un 1. Un 0 ó falso indica que el objeto i-ésimo no ha sido elegido, y un 1 o verdadero en caso contrario.
- *Función objetivo*:
- *Generación de la solución inicial*: La solución inicial se iniciará todos a cero, y en cada paso se irá escogiendo de entre la lista de objetos que quepan uno aleatoriamente hasta que no quepe ninguno más.
- *Esquema de generación de vecinos*: Se empleará el movimiento de cambio por mutación normal $Mov(W,\sigma)$ que altera el vector W sumándole otro vector Z generado a partir de una distribución normal de media 0 y varianza σ^2.
- *Criterio de aceptación*: Se considera una mejora cuando se aumenta el valor global de la función objetivo.
- ***Exploración del vecindario*:** En cada paso de la exploración se desmarcará un objeto elegido y se escogerá otro no elegido de forma aleatoria. Para explorar todo el vecindario se generará un vector/listado de todos los posibles pares de objetos a intercambiar (considerando únicamente aquellos que generen soluciones válidas), y se barajará aleatoriamente. Luego se irá recorriendo y aplicando dichos cambios para ir generando el vecindario. Cuando una nueva solución generada sea mejor que la actual, el vector de posibles intercambios se volverá a generar. Si se explorasen todos los intercambios válidos y no se encontrase ninguno que mejore la solución actual, el algoritmo parará por haber encontrado un óptimo local.

Algoritmo

Como algoritmo de BL para el QKP consideraremos el esquema del primer mejor, tal y como está descrito en las transparencias del Seminario 2. Sólo se considerarán como movimientos posibles aquellos que no excedan del peso máximo.

Se detendrá la ejecución **cuando se hayan realizado 90000 evaluaciones de la función objetivo, o cuando se haya explorado todo el entorno sin mejorar.** Se realizará una única ejecución sobre cada caso del problema.

4.3. Determinación de la Calidad de un Algoritmo

El modo habitual de determinar la calidad de un algoritmo de resolución aproximada de problemas de optimización es ejecutarlo sobre un conjunto determinado de instancias y comparar los resultados obtenidos con los mejores valores conocidos para dichas instancias.

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 un algoritmo probabilístico 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. Por tanto, resulta necesario efectuar varias ejecuciones con distintas secuencias probabilísticas y calcular el resultado medio y la desviación típica de todas las ejecuciones para representar con mayor fidelidad su comportamiento.

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. En el espacio de PRADO y la web de la asignatura se puede encontrar una implementación en lenguaje C de un generador aleatorio de buena calidad (*random.h*), y en C++ (*random.hpp*).

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. También 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.

Alternativamente, **en el caso de que se considere un número alto de casos del problema, se puede confiar en una única ejecución del algoritmo. Esta condición se cumple en las prácticas que realizaremos con el QKP**. Aun así, **será necesario inicializar la semilla del generador aleatorio para poder repetir el experimento** y obtener los mismos resultados si fuera necesario (en caso contrario, los resultados podrían variar en cada ejecución del mismo algoritmo sobre el mismo caso del problema).

Para facilitar la comparación de algoritmos en las prácticas del QAP se considerarán dos estadísticos distintos denominados *Fitness* y *Tiempo:*

- *Fitness* se calcula como el máximo beneficio obtenido. En problemas con óptimo conocido se suele calcular la media de las diferencias, en porcentaje, del valor obtenido por cada método en cada instancia respecto al óptimo. Sin embargo, en este problema es difícil calcular el óptimo para valores altos como 200 y 300. Por tanto, usaremos directamente el beneficio obtenido.

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

Cuanto mayor es el valor de *Fitness* para un algoritmo, mejor calidad tiene dicho algoritmo. Por otro lado, si dos métodos obtienen soluciones de la misma calidad (tienen valores de *Fitness* similares), uno será mejor que el otro si emplea menos tiempo en media. En la web de la asignatura hay también disponible un código en C (*timer*) para un cálculo adecuado del tiempo de ejecución de los algoritmos metaheurísticos, en C++ se recomienda el uso de la librería estándar *chrono.*

5. Tablas de Resultados a Obtener

Debido al gran número de ficheros (97) en vez de pedir una tabla para cada algoritmo (*Greedy*, BL) con los resultados de la ejecución de dicho algoritmo para cada caso del problema, se pide obtener el promedio para cada tamaño de cadena. Tendrá la misma estructura que la Tabla 5.2.

Tabla 5.2: Resultados para el algoritmo X

Tamaño	**Densidad**	**Fitness**	**Tiempo**
100	25	x	x
100	50	x	x
100	75	x	x
100	100	X	x
200	25	X	x
200	50	X	X
200	75	X	X
200	100	X	X
300	25	X	X
300	50	X	X

Finalmente, se construirá para cada tamaño 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 5.3, para los tamaños 100, 200 y 300. Dicha tabla estará ordenada por Fitness (de mejor a peor).

Tabla 5.3: Resultados globales en el QAP para Tamaño=100

Algoritmo	**Fitness**	**Tiempo**
Greedy	x	x
BL	x	x

A partir de los datos mostrados en estas tablas, el estudiante 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 de calidad de las soluciones y tiempo requerido para producirlas. Por otro lado, se puede analizar también el comportamiento de los algoritmos en algunos de los casos individuales que presenten un comportamiento más destacado.

6. Documentación y Ficheros a Entregar

En general, la **documentación** de ésta y de cualquier otra práctica será un fichero pdf que deberá incluir, al menos, el siguiente contenido:

a) Portada con el número y título de la práctica, el curso académico, el nombre del problema escogido, los algoritmos considerados; el nombre, DNI y dirección e-mail del estudiante, y su grupo y horario de prácticas.

b) Índice del contenido de la documentación con la numeración de las páginas.

c) **Breve** descripción/formulación del problema (**máximo 1 página**). Podrá incluirse el mismo contenido repetido en todas las prácticas presentadas por el estudiante.

d) Breve descripción de la aplicación de los algoritmos empleados al problema (**máximo 4 páginas**): Todas las consideraciones comunes a los distintos algoritmos se describirán en este apartado, que será previo a la descripción de los algoritmos específicos. Incluirá por ejemplo la descripción del esquema de representación de soluciones y la descripción en pseudocódigo (no código) de la función objetivo y los operadores comunes.

e) 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/ 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**.

   Para esta primera práctica, se incluirán al menos las descripciones en pseudocódigo de:
   - Para el algoritmo BL, los métodos de creación de la lista de candidatos y de exploración del entorno, el operador de generación de vecino, la factorización de la BL y la generación de soluciones aleatorias.

f) Descripción en **pseudocódigo** del algoritmo de comparación.

g) Breve explicación del **procedimiento considerado para desarrollar la práctica**: implementación a partir del código proporcionado en prácticas o a partir de cualquier otro, o uso de un framework de metaheurísticas concreto. Inclusión de un pequeño **manual de usuario describiendo el proceso para que el profesor de prácticas pueda replicarlo**.

h) Experimentos y análisis de resultados:
   - Descripción de los casos del problema empleados y 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.

i) 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.

Dentro del código fuente deberá de tener los siguientes métodos/funciones:
- **busquedaLocal:** Que recibiendo de alguna forma el dataset devuelva la mejor solución obtenida por el algoritmo de BL, y su fitness correspondiente.
- **greedy**: Que recibiendo de alguna forma el dataset devuelva la solución obtenida por el algoritmo greedy, y su fitness correspondiente.

No será válido que los métodos se limiten a mostran por pantalla y no devuelvan tanto la solución final obtenida por cada algoritmo como su valor de fitness correspondiente.

El fichero pdf de la documentación y la carpeta software serán comprimidos en un fichero .zip etiquetado con los apellidos y nombre del estudiante (Ej. Pérez Pérez Manuel.zip). Este fichero será entregado por internet a través de la plataforma PRADO.

7. Método de Evaluación

Tanto en esta práctica como en las siguientes, se indicará la puntuación máxima que se puede obtener por cada algoritmo y su análisis. La inclusión de trabajo voluntario (desarrollo de variantes adicionales, experimentación con diferentes parámetros, prueba con otros operadores o versiones adicionales del algoritmo, análisis extendido, etc.) podrá incrementar la nota final por encima de la puntuación máxima definida inicialmente.

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.