UNIVERSIDAD DE GRANADA
E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN

Departamento de Ciencias de la
Computación e Inteligencia Artificial

Metaheurísticas

http://sci2s.ugr.es/docencia/metah
https://decsai.ugr.es

Guión de Prácticas

Práctica 1.a:
Búsquedas por Trayectorias para el Problema de la Asignación Cuadrática

Curso 2015-2016

Tercer Curso del Grado en Ingeniería Informática

Práctica 1.a

Búsquedas por Trayectorias para el Problema de la Asignación Cuadrática

1. Objetivos

El objetivo de esta práctica es estudiar el funcionamiento de los siguientes algoritmos: Búsqueda Local (BL), Enfriamiento Simulado (ES) y Búsqueda Tabú (BT). Para ello, se requerirá que el alumno adapte estos métodos para resolver el problema de la asignación cuadrática (QAP) descrito en las transparencias del Seminario 2.a y que compare los resultados obtenidos con el valor de los óptimos o de las estimaciones existentes para de una serie de casos del problema seleccionados de la QAPLIB. Para efectuar la comparativa de resultados, *se deberá también implementar un algoritmo greedy para resolver el QAP*, que será considerado en todas las prácticas posteriores.

La práctica se evalúa sobre un total de **2,5 puntos**, distribuidos de la siguiente forma: BL (0,75 puntos), ES (0,75 puntos) y BT (1 punto). Además, se podrá implementar opcionalmente una variante extendida de la BT que puede suponer **1 punto adicional**.

La fecha límite de entrega será el **Lunes 4 de Abril de 2016** antes de las 23:59 horas. La entrega de la práctica se realizará por internet a través del acceso identificado de la web del departamento de CCIA (https://decsai.ugr.es).

2. Trabajo a Realizar

El alumno 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 el problema y los casos considerados, los aspectos concretos de cada algoritmo y las tablas de resultados a obtener.

3. Problema y Casos Considerados

3.1. Introducción al Problema de la Asignación Cuadrática

El problema de asignación cuadrática (en inglés, *quadratic assignment problem*, QAP) es uno de los problemas de optimización combinatoria más conocidos. En él se dispone de n unidades y n localizaciones en las que situarlas, por lo que el problema consiste en encontrar la asignación óptima de cada unidad a una localización. La nomenclatura "cuadrático" proviene de la función objetivo que mide la bondad de una asignación, la cual considera el producto de dos términos, la distancia entre cada par de localizaciones y el flujo que circula entre cada par de unidades. El QAP se puede formular como:

$$\min_{\pi \in \Pi_N} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \right)$$

donde:

- π es una solución al problema que consiste en una permutación que representa la asignación de la unidad i a la localización $\pi(i)$
- f_{ij} es el flujo que circula entre la unidad i y la j.
- d_{kl} es la distancia existente entre la localización k y la l.

3.2. Casos del QAP Considerados

Se utilizarán **20 casos** seleccionados de varios de los conjuntos de instancias disponibles en la QAPLIB (http://www.seas.upenn.edu/qaplib/inst.html). El tamaño de los casos seleccionados oscila entre 20 y 256.

El fichero *Tablas_QAP_2015-16.xls*, disponible en la web de la asignatura, incluye las características (nombre, tamaño y coste de la mejor solución conocida) de las instancias seleccionadas. Aunque los ficheros de los casos pueden ser descargados de la QAPLIB, se han incluido también en la web de la asignatura para facilitar el trabajo del alumno.

El formato de los ficheros es el siguiente:

- Una primera línea donde se indica el tamaño del problema n (número de unidades/localizaciones).
- Una línea en blanco.
- n líneas que muestran el contenido de la matriz de flujo entre cada par de unidades.
- Una línea en blanco.
- n líneas que muestran el contenido de la matriz de distancias entre cada par de localizaciones.

Ejemplo (Tai60a.dat):

```
  60
  0 79 71  7 70 35 96 47 30 32 23 35 39 56 80 28 78 62 81 80 13 85 26
5 62  3 26 46 91 49 27 73 21 58  3 21 27 57 83 30 60 94 44 97 94 66  5
36 21 99 83 28  2 26 60 34 76 64 87 53
 79  0 98 76 40 70 43 41 79 88 14 49 72 89 52 55 12  7 31 31 98 45 59
22 87  6 36 26 85  6 33 35 48 59 88 25 58 76 98 28 46 79 25 18 48 95
78 77 90 23  9 85 55 54 66 11 57 69 81 99
…

  0 21 95 82 56 41  6 25 10  4 63  6 44 40 75 79  0 89 35  9  1 85 84
12  0 26 91 11 35 82 26 69 56 86 45 91 59 18 76 39 18 57 36 61 36 21
71 11 29 82 82  6 71  8 77 74 30 89 76 76
 21  0 40 93 56  1 50  4 36 27 85  2  1 15 11 35 11 20 21 61 80 58 21
76 72 44 85 94 90 51  3 48 29 90 66 41 15 83 96 74 45 65 40 54 83 14
71 77 36 53 37 26 87 76 91 13 29 11 77 32
…
```

3.3. Algoritmo de Comparación: *Greedy*-QAP

El algoritmo *greedy* del QAP se basa en la heurística de asociar unidades de gran flujo con localizaciones céntricas en la red y viceversa. Para ello, 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 ($\hat{f}_i = \sum_{j=1}^{n} f_{ij}$) y de distancias desde una localización al resto ($\hat{d}_k = \sum_{l=1}^{n} d_{kl}$). Para una unidad concreta, cuanto mayor sea $\hat{f}_i$ más importante es la unidad en el intercambio de flujos. Por otro lado, cuanto menor sea $\hat{d}_k$ más céntrica es una localización concreta. Por tanto, el algoritmo irá seleccionando la unidad *i* libre con mayor $\hat{f}_i$ y le asignará la localización *k* libre con menor $\hat{d}_k$.

3.4. 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 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. 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 la web de la asignatura se puede encontrar una implementación en lenguaje C de un generador aleatorio de buena calidad (*random.h*).

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 QAP**. 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 *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. Es decir, para un problema de minimización como el QAP:

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

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. 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.

La hoja Excel *Tablas_QAP_2015-16.xls*, disponible en la web de la asignatura, permite generar de forma automática los estadísticos comentados para las tablas de resultados de la práctica.

4. Componentes de los Algoritmos

Los algoritmos de esta práctica tienen en común las siguientes componentes:

- *Esquema de representación*: Se seguirá la representación en forma de permutación π de tamaño n que representa una asignación de unidades (los índices del vector) a localizaciones (los contenidos del mismo).
- *Función objetivo*: $\min_{\pi \in \Pi_N} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \right)$
- *Generación de la solución inicial*: La solución inicial se generará de forma aleatoria en todos los casos.
- *Esquema de generación de vecinos*: Se empleará el movimiento de intercambio $Int(\pi,r,s)$ 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.
- *Criterio de parada*: Se detendrá la ejecución del algoritmo bien cuando no se encuentre mejora en todo el entorno (BL) o bien cuando se hayan evaluado 25000 soluciones distintas (en cualquier caso).

A continuación veremos las particularidades de cada algoritmo.

4.1. Búsqueda Local

Algoritmo

Como algoritmo de BL para el QAP consideraremos el esquema del primer mejor, tal y como está descrito en las transparencias del Seminario 2.a. Sólo se considerarán como movimientos posibles los asociados a las unidades activadas en la máscara ***don't look bits***, ***DLB***. Se explorará el entorno definido por las unidades activas según la máscara *DLB* en orden ascendente según el número de unidad.

Una vez realizado el movimiento, se actualizan la solución actual y la máscara *DLB*, y se comienza a explorar el nuevo entorno desde la primera unidad activa. Se detiene la ejecución cuando se hayan explorado todos los movimientos posibles del entorno sin encontrar mejora.

4.2. Enfriamiento Simulado

Algoritmo

Se ha de emplear un algoritmo ES con las siguientes componentes:

- *Esquema de enfriamiento*: Se empleará el esquema de Cauchy modificado:

$$T_{k+1} = \frac{T_k}{1 + \beta \cdot T_k} \qquad ; \qquad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

  donde M es el número de enfriamientos (iteraciones) a realizar, T_0 es la temperatura inicial y T_f la temperatura final que tendrá un valor cercano a cero.

- *Operador de Vecino y exploración del entorno para L(T)*: En cada iteración del bucle interno *L(T)*, se aplicará un único movimiento *Int*(π,*r*,*s*) para generar una única solución vecina que será comparada con la solución actual. Se escogerán aleatoriamente las unidades *r* y *s* a las cuales se les cambiará la localización. **No se considerará el uso de la máscara *DLB***.

- *Condición de enfriamiento L(T)*: Se enfriará la temperatura, finalizando la iteración actual, bien cuando se haya generado un número máximo de vecinos *máx_vecinos* (independientemente de si han sido o no aceptados) o bien cuando se haya aceptado un número máximo de los vecinos generados *máx_éxitos*.

- *Condición de parada*: El algoritmo finalizará bien cuando haya alcanzado el número máximo de evaluaciones prefijado o bien cuando el número de éxitos en el enfriamiento actual sea igual a 0.

Valores de los parámetros y ejecuciones

La temperatura inicial se calculará en función de la siguiente fórmula:

$$T_0 = \frac{\mu \cdot C(S_0)}{-\ln(\phi)}$$

donde T_0 es la temperatura inicial, $C(S_0)$ es el coste de la solución inicial y $\phi \in [0,1]$ es la probabilidad de aceptar una solución un μ por 1 peor que la inicial. En las ejecuciones se considerará $\phi=\mu=0,3$. La temperatura final T_f se fijará a 10^{-3} (*¡comprobando siempre que sea menor que la inicial!*).

Los parámetros que definen el bucle interno *L(T)* tomarán valor *máx_vecinos*=10·*n* (tamaño del caso del problema) y *máx_éxitos=0.1*máx_vecinos*. El número máximo de evaluaciones será 25000. Por lo tanto, el número de iteraciones (enfriamientos) M del algoritmo ES será igual a 25000/*máx_vecinos* [1].

4.3. Búsqueda Tabú

Algoritmo versión básica

Se trabajará con la versión del algoritmo BT estudiada en clase, la cual utiliza una lista de movimientos tabú:

- *Lista tabú*: Almacena las localizaciones intercambiadas y las nuevas posiciones que ocupan (*i*, *j*, Pos(*i*), Pos(*j*)). A partir de ese momento y mientras ese registro se mantenga como tabú-activo, serán movimientos tabú aquellos que den lugar a que el elemento 'i' ocupe la posición 'Pos(i)' y/o el elemento 'j' ocupe la posición 'Pos(j)'. Tener en cuenta que (*i*, *j*, Pos(*i*), Pos(*j*)) ⇔ (j, i, Pos(j), Pos(i)).

- *Operador de Vecino y exploración del entorno para L(T)*: En cada iteración se generarán 30 soluciones vecinas aplicando un único movimiento *Int*(π,*r*,*s*) para obtener cada una de ellas. Se escogerán aleatoriamente las unidades *r* y *s* para las cuales se intercambiarán las localizaciones previamente asignadas. Se seleccionará el mejor vecino de acuerdo a los criterios tabú.

- *Criterio de aspiración*: Tener mejor coste (menor) que la mejor solución obtenida hasta el momento.

Valores de los parámetros y ejecuciones de la versión básica

El número máximo de evaluaciones será 25000. El tamaño de la lista tabú *n*/2.

[1] **NOTA 1**: Es posible que se realicen menos de 25000 evaluaciones durante la ejecución debido a la condición de enfriamiento cuando se alcanzan *máx_éxitos*.
NOTA 2: Puede que con *máx_vecinos*=10·*n* se generen demasiados vecinos para cada enfriamiento. El alumno puede probar también con *máx_vecinos*=5·*n* o directamente *n*.

Algoritmo versión extendida

La versión extendida consiste en incorporar una memoria de largo plazo y un procedimiento de reinicialización a la versión básica anterior. Para ello, se aplicará una oscilación estratégica basada en tres estrategias distintas, escogidas aleatoriamente según una probabilidad: construcción de una solución inicial aleatoria (diversificación), nuevo arranque de la búsqueda desde la mejor solución encontrada hasta el momento (intensificación) y uso de la memoria a largo plazo para generar una nueva solución diferente a las visitadas (diversificación controlada).

El uso de la memoria a largo plazo implica mantener una estadística en forma de matriz simétrica *frec* de tamaño $n \times n$ (basta con mantener la diagonal superior) que almacena el número de veces que se ha producido cada asignación unidad-localización en las soluciones aceptadas durante la búsqueda. Para reinicializar con diversidad, se van realizando una a una las asignaciones que se dieron con menor frecuencia en el pasado (asignación greedy). Un posible algoritmo de reinicialización se presenta en las transparencias del Seminario 2.a. **La memoria de largo plazo no se actualiza tras cada reinicialización**.

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 tanto por ciento según una decisión aleatoria uniforme.

Valores de los parámetros y ejecuciones de la versión extendida

El número máximo de evaluaciones se mantendrá 25000. Se realizará una reinicialización cuando transcurran **10 o 5 iteraciones** (para casos de tamaño menor o igual a 50 o mayor que 50, respectivamente) sin mejorar la mejor solución obtenida hasta el momento en la ejecución del algoritmo (mejor global). La probabilidad de reinicializar desde una solución inicial aleatoria es 0,25, la de partir de la mejor solución obtenida es 0,25 y la de usar la memoria a largo plazo parar generar la solución más diversa es 0,5.

El tamaño inicial de cada lista tabú será $n/2$. Este valor cambiará después de las reinicializaciones, aumentando o disminuyendo en un 50%.

5. Tablas de Resultados a Obtener

Se diseñará una tabla para cada algoritmo (*Greedy*, BL, ES, BT básica y BT extendida) donde se recojan los resultados de la ejecución de dicho algoritmo al conjunto de casos del problema. Tendrá la misma estructura que la Tabla 5.1.

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

Algoritmo X					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20a	x	X	Sko49	x	x
Chr20c	x	x	Sko81	x	x
Chr22b	x	x	Sko90	x	x
Chr25a	x	x	Sko100f	x	x
Esc32a	x	x	Tai64c	x	x
Esc64a	x	x	Tai80a	x	x
Esc128	x	x	Tai100a	x	x
Kra32	x	x	Tai150b	x	x
Lipa90a	x	x	Tai256c	x	x
Sko42	x	x	Tho150	x	x

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. Aunque en la tabla que sirve de ejemplo se han incluido todos los algoritmos considerados en esta práctica, naturalmente sólo se incluirán los que se hayan desarrollado.

Tabla 5.2: Resultados globales en el QAP

Algoritmo	**Desv**	**Tiempo**
Greedy	x	x
BL	x	x
ES	x	x
BT básica	x	x
BT extendida	x	x

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 de calidad de las soluciones y tiempo requerido para obtenerlas. 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 esta y de cualquier otra práctica será un fichero pdf que deberá incluir, al menos, el siguiente contenido:

1. 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 alumno, y su grupo y horario de prácticas.
2. Índice del contenido de la documentación con la numeración de las páginas.

3. **Breve** descripción/formulación del problema (**máximo 1 página**). Podrá incluirse el mismo contenido repetido en todas las prácticas presentadas por el alumno.
4. Breve descripción de la aplicación de los algoritmos empleados al problema (**máximo 2 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 de la función objetivo y los operadores comunes (en este caso, el de **generación de vecino** y su **factorización**), etc.
5. 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, además de la descripción del esquema de búsqueda, se deberá incluir al menos:
   - Para el algoritmo BL, descripción en pseudocódigo del método de creación de la lista de candidatos (máscara *DLB*) y de exploración del entorno.
   - Para el algoritmo ES, descripción del cálculo de la temperatura inicial y del esquema de enfriamiento.
   - Para el algoritmo BT básico, descripción en pseudocódigo del manejo de la lista tabú.
   - Para el algoritmo BT extendido, en caso de haberlo realizado, descripción en pseudocódigo del manejo de los mecanismos de reinicialización.
6. Descripción en **pseudocódigo** del algoritmo de comparación.
7. 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**.
8. 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.

9. 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.

El fichero pdf de la documentación y la carpeta software serán comprimidos en un fichero .zip etiquetado con los apellidos y nombre del alumno (Ej. Pérez Pérez Manuel.zip). Este fichero será entregado por internet a través del acceso identificado de la web del departamento de CCIA (https://decsai.ugr.es).

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, como la BT extendida de esta práctica, experimentación con diferentes parámetros, prueba con otros operadores o versiones adicionales del algoritmo, análisis extendido, etc.) podrá incrementar la nota final.

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.