Master en Sistemas Telemáticos e Informáticos

Asignatura *Optimización de Sistemas de Comunicación*

Curso 2010/11

Práctica 3: Búsquedas locales

Objetivo
El objetivo de la práctica es que el alumno se familiarice con las técnicas más habituales de búsqueda local y sea capaz de mejorar las soluciones generadas utilizando dichas técnicas para los problemas planteados en las Prácticas 1 y 2.

Obligatoriedad
La práctica no es obligatoria.

Prerrequisitos
El alumno debe haber realizado las Prácticas 1 y 2, y en concreto, debe ser capaz de generar soluciones aleatorias para el Problema de la Diversidad MaxMin (MMDP).

Descripción

Búsquedas Locales
La mejora de soluciones mediante búsquedas locales consiste principalmente en partir de una solución factible y explorar el vecindario en busca de soluciones mejores. El vecindario de una solución se construye aplicando *movimientos* sobre la solución de partida. En el caso del MMDP, estos movimientos son *movimientos de intercambio*, donde se intercambia un elemento que está en la solución (seleccionado) por un elemento que no está en la solución (no seleccionado).

Cuando se implementan búsquedas locales, la exploración del vecindario puede seguir diferentes estrategias. Dos de las más utilizadas son *best improvement* y *first improvement:*

- En una búsqueda de tipo *best improvement* partiendo de una solución factible se construye el vecindario y se selecciona la mejor solución del mismo. Este proceso se repite partiendo de esta nueva solución hasta llegar a una solución para la cual no se encuentra en su vecindario ninguna solución que la mejore.

- En una búsqueda de tipo *first improvement* se elige la primera solución que se encuentre en el vecindario que mejore la solución actual. Este proceso se repite hasta llegar a una solución en cuyo vecindario no se encuentre ninguna solución que la mejore.

En el caso de la estrategia *first improvement* es importante el orden de creación de soluciones durante la exploración del vecindario. Si el orden en el que se realizan los intercambios es orden lexicográfico (elementos con índices más pequeños se visitan primero) se explorará más exhaustivamente la zona del espacio de soluciones donde los primeros elementos están seleccionados. Por tanto, es poco probable que el algoritmo explore otras zonas que podrían

contener la solución óptima.

Es por ello que se suele aleatorizar la exploración del vecindario, de manera que se visiten con igual o similar probabilidad todas las zonas del espacio de búsqueda.

Determinar la calidad de un algoritmo

Para determinar la calidad de un algoritmo de resolución aproximada de problemas de optimización, el algoritmo se ejecuta sobre un conjunto determinado de instancias. Posteriormente, se comparan los resultados obtenidos en la ejecución con los mejores valores conocidos para ese conjunto de instancias. Por otro lado, 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.

Para facilitar la comparación de algoritmos se ha desarrollado una herramienta, *mhAnalyzer*, que analiza los resultados obtenidos al ejecutar varios algoritmos sobre un conjunto de instancias. Esta herramienta genera una tabla que resume la calidad de las soluciones obtenidas por cada algoritmo y el tiempo empleado para obtenerlas. En concreto, la herramienta calcula, por cada método, los estadísticos *Dev* y *Time:*

- *Dev* 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 esa instancia. Es decir:

$$\sum_{instancia \in instancias} \frac{mejorValor - valorAlgoritmo}{mejorValor \cdot ١٠٠} / |instancias|$$

  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.

- *Time* se calcula como la media del tiempo de ejecución empleado por cada instancia.

Cuanto menor es el valor de *Dev* para un algoritmo, mejor calidad tiene ese algoritmo, porque en media obtiene soluciones más cercanas al mejor valor conocido. Por ejemplo, si dos métodos obtienen soluciones de la misma calidad (tienen valores de *Dev* similares), uno será mejor que el otro si emplea menos tiempo en media.

Para la realización del análisis, *mhAnalyzer* requiere que existan una serie de ficheros en una carpeta. Deberá existir un fichero (con cualquier nombre) por cada algoritmo que se quiera comparar con el siguiente formato:

```
Descripción del Algoritmo
nombre_fichero_instancia_1 valor_obtenido tiempo  [índices_elementos_solución]
nombre_fichero_instancia_2 valor_obtenido tiempo  [índices_elementos_solución]
nombre_fichero_instancia_3 valor_obtenido tiempo  [índices_elementos_solución]
nombre_fichero_instancia_4 valor_obtenido tiempo  [índices_elementos_solución]
...
```

Deberá existir un fichero llamado `best_values.txt` que contenga los mejores valores conocidos para cada instancia con el siguiente formato:

```
nombre_fichero_instancia_1 mejor_valor_conocido
nombre_fichero_instancia_2 mejor_valor_conocido
nombre_fichero_instancia_3 mejor_valor_conocido
nombre_fichero_instancia_4 mejor_valor_conocido
...
```

Para ejecutar mhAnalyzer se utilizará el siguiente comando:

```
java -jar mhanalyzer.jar carpeta_resultados modo_optimización
```

Donde el último parámetro (*modo_optimización*) será "*min*" si el problema es de minimización y "*max*" si es de maximización.

Se pide

- Definir e implementar una búsqueda local de tipo *best improvement* para el MMDP
- Definir e implementar dos búsquedas locales de tipo *first improvement* para el MMDP:
  - Una donde el orden de exploración sea lexicográfico
  - Otra donde el orden de exploración sea aleatorio
- Programar cuatro algoritmos:
  - A1: constructivo aleatorio,
  - A2: construcción aleatoria y mejora *best improvement*,
  - A3: construcción aleatoria y mejora *first improvement* en orden lexicográfico,
  - A4: construcción aleatoria y mejora *first improvement* en orden aleatorio.
- Ejecutar los cuatro algoritmos con el conjunto de instancias GKD-Ia y generar un fichero por cada algoritmo con el formato requerido por *mhAnalyzer*.
- Ejecutar los cuatro algoritmos con el conjunto de instancias GKD-Ic y generar un fichero por cada algoritmo con el formato requerido por *mhAnalyzer*.
- Analizar los resultados obtenidos para el conjunto de instancias GKD-Ia y determinar cuál es el mejor algoritmo para este conjunto.
- Analizar los resultados obtenidos para el conjunto de instancias GKD-Ic y determinar cuál es el mejor algoritmo para este conjunto.