Trabajo práctico 3

Fecha de entrega: viernes 21 de noviembre, hasta las 20:00 horas.

Dado un grafo simple $G = (V, E)$ y un entero k, se define una k-partición de G como una partición de V en k conjuntos de vértices $V_1, \ldots, V_k$. Las aristas *intrapartición* de una k-partición, son aquellas aristas de G cuyos extremos se encuentran en un mismo conjunto de la partición, es decir, una arista $uv \in E$ es *intrapartición* si existe un $i \in \{1, \ldots, k\}$, tal que $u, v \in V_i$. Dada una función de peso $\omega : E \to \mathbb{R}_+$, definida sobre las aristas de G, el peso de una k-partición es la suma de los pesos de las aristas intrapartición.

Siendo $G = (V, E)$ un grafo simple, dado un entero k y una función de peso, $\omega : E \to \mathbb{R}_+$, el problema de la *k-Partición de Mínimo Peso* (k-PMP) consiste en hallar una k-partición de G con peso mínimo, según ω (los conjuntos de la partición resultado no son necesariamente no vacíos).

En el presente trabajo práctico se pide:

1. Desarrollar los siguientes puntos:
   a) Relacionar el problema de k-PMP con el problema 3 del TP 1.
   b) Relacionar el problema de k-PMP con el problema de coloreo de los vértices de un grafo.
   c) Describir situaciones de la vida real que puedan modelarse utilizando k-PMP.
2. Diseñar e implementar un **algoritmo exacto** para k-PMP y desarrollar los siguientes puntos:
   a) Explicar detalladamente el algoritmo implementado. Elaborar podas y estrategias que permitan mejorar los tiempos de resolución.
   b) Calcular el orden de complejidad temporal de peor caso del algoritmo.
   c) Realizar una experimentación que permita observar los tiempos de ejecución del algoritmo en función del tamaño de entrada y de las podas y/o estrategias implementadas.
3. Diseñar e implementar una **heurística constructiva golosa** para k-PMP y desarrollar los siguientes puntos:
   a) Explicar detalladamente el algoritmo implementado.
   b) Calcular el orden de complejidad temporal de peor caso del algoritmo.
   c) Describir instancias de k-PMP para las cuales la heurística no proporciona una solución óptima. Indicar qué tan mala puede ser la solución obtenida respecto de la solución óptima.
   d) Realizar una experimentación que permita observar la performance del algoritmo en términos de tiempo de ejecución en función del tamaño de entrada.
4. Diseñar e implementar una **heurística de búsqueda local** para k-PMP y desarrollar los siguientes puntos:
   a) Explicar detalladamente el algoritmo implementado. Plantear al menos dos vecindades distintas para la búsqueda.
   b) Calcular el orden de complejidad temporal de peor caso de una iteración del algoritmo de búsqueda local (para las vecindades planteadas). Si es posible, dar una cota superior para la cantidad de iteraciones de la heurística.
   c) Realizar una experimentación que permita observar la performance del algoritmo comparando los tiempos de ejecución y la calidad de las soluciones obtenidas, en función de las vecindades utilizadas y elegir, si es posible, la configuración que mejores resultados provea para el grupo de instancias utilizado.
5. Utilizando las heurísticas implementadas en los puntos anteriores, diseñar e implementar un algoritmo para k-PMP que use la **metaheurística GRASP** [1] y desarrollar los siguientes puntos:
   a) Explicar detalladamente el algoritmo implementado. Plantear distintos criterios de parada y de selección de la lista de candidatos (RCL) de la heurística golosa aleatorizada.

b) Realizar una experimentación que permita observar los tiempos de ejecución y la calidad de las soluciones obtenidas. Se debe experimentar variando los valores de los parámetros de la metaheurística (lista de candidatos, criterios de parada, etc.) y elegir, si es posible, la configuración que mejores resultados provea para el grupo de instancias utilizado.

6. Una vez elegidos los mejores valores de configuración para cada heurística implementada (si fue posible), realizar una **experimentación sobre un conjunto nuevo de instancias** para observar la performance de los métodos comparando nuevamente la calidad de las soluciones obtenidas y los tiempos de ejecución en función del tamaño de entrada. Para los casos que sea posible, comparar también los resultados del algoritmo exacto implementado. Presentar todos los resultados obtenidos mediante gráficos adecuados y discutir al respecto de los mismos.

Condiciones de entrega y términos de aprobación

Este trabajo práctico consta de varias partes y para aprobar el trabajo se requiere aprobar todas las partes del mismo. La nota final del trabajo será un promedio ponderado de las notas finales de las partes y el trabajo práctico se aprobará con una nota de 5 (*cinco*) o superior. De ser necesario (o si el grupo lo desea) el trabajo podrá reentregarse una vez corregido por los docentes y en ese caso la reentrega deberá estar acompañada por un *informe de modificaciones*. Este informe deberá detallar brevemente las diferencias entre las dos entregas, especificando los cambios, agregados y/o partes eliminadas del trabajo. Cualquier cambio que no se encuentre en dicho informe podrá no ser tenido en cuenta en la corrección de la reentrega. En caso de reentregar, la nota final del trabajo será el 20 % del puntaje otorgado en la primera corrección más el 80 % del puntaje obtenido al recuperar.

Respecto de las implementaciones, se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia. Además, debe poder compilarse y ejecutarse correctamente en las máquinas de los laboratorios del Departamento de Computación. La cátedra recomienda el uso de C++ o Java, y se sugiere consultar con los docentes la elección de otros lenguajes para la implementación.

Deberá entregarse un informe impreso que desarrolle los puntos mencionados. Por otro lado, deberá entregarse el mismo informe en formato digital acompañado de los códigos fuentes desarrollados e instrucciones de compilación, de ser necesarias. Estos archivos deberán enviarse a la dirección `algo3.dc@gmail.com` con el asunto *"TP 3: Apellido_1, ..., Apellido_n"*, donde n es la cantidad de integrantes del grupo y *Apellido_i* es el apellido del i-ésimo integrante.

La entrada y salida de los programas **deberá hacerse por medio de la entrada y salida estándar del sistema**. No se deben considerar los tiempos de lectura/escritura al medir los tiempos de ejecución de los programas implementados.

Formato de entrada: La entrada comienza con una línea con 3 valores enteros, n, m y k, separados por espacios (siendo n y k valores positivos y m no negativo). Los valores n y m indican la cantidad de vértices y aristas de G, respectivamente, y k es la cantidad de conjuntos de la partición a encontrar. A continuación, siguen m líneas, representando las aristas del grafo. Cada una de estas líneas tiene el formato:

```
u v w
```

donde `u` y `v` son los extremos de la arista representada (numerados de 1 a n) y `w` es el peso de la función ω correspondiente a la arista (notar que `w` puede ser fraccionario). Se puede suponer que los grafos son simples (i.e., no tienen bucles ni ejes repetidos).

Formato de salida: La salida debe contener una línea con el siguiente formato:

```
i1 i2 ... in
```

donde `i1`, ..., `in` son los índices de los conjuntos asignados a cada uno de los n vértices del grafo. Es decir, `ij` es el índice del conjunto de la partición al cual pertenece el vértice j (notar que estos valores deben estar entre 1 y k). En caso de haber más de una partición óptima, el algoritmo puede devolver cualquiera de ellas.

Referencias

[1] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, pp 109–134, 1995.