Trabajo práctico 3

Fecha de entrega: viernes 26 de junio, hasta las 19:00 horas.

Sea $G = (V, E)$ un grafo simple. Un conjunto $D \subseteq V$ es un *conjunto dominante* de G si todo vértice de G está en D o bien tiene al menos un vecino que está en D. Por otro lado, un conjunto $I \subseteq V$ es un *conjunto independiente* de G si no existe ningún eje de E entre dos vértices de I. Definimos entonces un *conjunto independiente dominante* de G como un conjunto independiente que a su vez es un conjunto dominante del grafo G.

El problema de *Conjunto Independiente Dominante Mínimo* (CIDM) consiste en hallar un conjunto independiente dominante de G con mínima cardinalidad.

En el presente trabajo práctico se pide:

1. Desarrollar los siguientes puntos:
   a) Relacionar el problema de CIDM con el problema 3 del TP 1 (i.e., similitudes y diferencias).
   b) Un conjunto independiente de $I \subseteq V$ se dice *maximal* si no existe otro conjunto independiente $J \subseteq V$ tal que $I \subset J$, es decir que I está incluido estrictamente en J. Demostrar que todo conjunto independiente maximal es un conjunto dominante.
   c) Describir situaciones de la vida real que puedan modelarse utilizando CIDM.
2. Diseñar e implementar un **algoritmo exacto** para CIDM 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 de los parámetros de la entrada y de las podas y/o estrategias implementadas.
3. Diseñar e implementar una **heurística constructiva golosa** para CIDM 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 CIDM 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 de los parámetros de la entrada.
4. Diseñar e implementar una **heurística de búsqueda local** para CIDM y desarrollar los siguientes puntos:
   a) Explicar detalladamente el algoritmo implementado. Plantear al menos dos vecindades distintas para la búsqueda y al menos dos soluciones iniciales.
   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 y las soluciones iniciales 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 CIDM 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 las vecindades utilizadas en la búsqueda local. 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 de los parámetros de la 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.

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 dos valores enteros, n y m, separados por espacios (con $n > 0$ y $m \geq 0$); los valores n y m indican la cantidad de vértices y aristas de G, respectivamente. A continuación, siguen m líneas, representando las aristas del grafo. Cada una de estas líneas tiene el formato:

```
u v
```

donde `u` y `v` son los extremos de la arista representada (numerados de 1 a n). Se puede suponer que los grafos son simples (i.e., sin bucles ni aristas repetidas).

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

```
k i1 i2 ... ik
```

donde `k` es la cantidad de vértices del conjunto solución y `i1`, ..., `ik` son los vértices del conjunto. En caso de haber más de un conjunto independiente dominante óptimo, el algoritmo exacto puede devolver cualquiera de ellos.

Referencias

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