Universidad de Buenos Aires
Facultad de
Ciencias Exactas y Naturales
Departamento de Computación

Algorítmos y Estucturas de datos III

Segundo Cuatrimestre de 2011

Trabajo Práctico 3

Cartero Chino en grafo mixto.

Grupo: '1'

Integrante	LU	Correo electrónico
Cammi, Martín	676/02	martincammi@gmail.com
Garbi, Sebastián	179/05	garbyseba@gmail.com
Kretschmayer, Daniel	310/99	daniak@gmail.com

Índice

Ejecución del TP

0.1. Lenguaje utilizado

El lenguaje utilizado para el trabajo práctico ha sido *Java*, compilando con la versión 1.5 de la Virtual Machine.

El trabajo se acompaña con los fuentes de la solución que puede importarse en IDE de Eclipse o ejecutarse desde línea de comandos.

0.2. Como ejecutar el TP

Desde línea de comandos

- Posicionarse en el directorio Algo3Tp3
- Copiar allí el archivo de entrada para el problema i, por ejemplo Ej1.in
- Ejecutar el comando: *java -cp ./bin problema1.Ej1*

Esto generará el archivo Ej1.out con la solución en el mismo directorio Algo3Tp3.

Desde el Eclipse

Primero importaremos el proyecto:

- Seleccionar File ⇒ Import.
- Seleccionar General ⇒ Existing Projects into Workspace ⇒ Next.
- Seleccionar el directorio llamado Algo3Tp3.
- Finish.

Desde la vista de **Package Explorer** bajo el paquete **src** aparecerán tres paquetes más y dentro de cada uno de ellos los siguientes archivos de java:

COLOCAR IMAGEN CORRECTA

Para ejecutar un problema:

- Posicionarse en el directorio Algo3Tp3
- Copiar en el directorio Algo3Tp3 el archivo de entrada para el problema i, por ejemplo Ej1.in
- Con botón derecho Run As ⇒ Java Application. Se ejecutará el problema seleccionado.

Esto generará el archivo Ej1.out con la solución en el mismo directorio Algo3Tp3.

1. Introducción

En 1736 Leonhard Euler publicó un trabajo en el cual resolvía el denominado *problema de los puentes de Königsberg*. La ciudad de Köningsberg, actual Kaliningrado, Rusia, está dividida por el río Pregel en 4 zonas, dos orillas (A y B), la isla de Kneiphof (C) y otras dos partes divididas por el río. (D y F). Las orillas estaban conectadas mediante 7 puentes. El problema consistía partir de una orilla y recorrer todos los puentes, recorriéndo una sola vez cada uno y volviendo al punto de partida.

Este fué el puntapié inicial para un tipo de problema que más adelante de modelaría usando la teoría de grafos. Este problema fue caracterizado por Euler y consistía en encontrar un circuito que pasara por todas las aristas de un grafo una sóla vez volviendo al punto de partida.

Más de doscientos años más tarde en 1962 el matemático chino Kwan Mei-Koo publicó un paper en el que abordaba un problema ligeramente similar, proponía que si un grafo no poseía un circuito Euleriano, podria quizás encontrar el circuito más corto que pasara por todas las aristas aunque repitiera algunas de ellas. Este problema fue luego nombrado como "Problema del cartero chino".

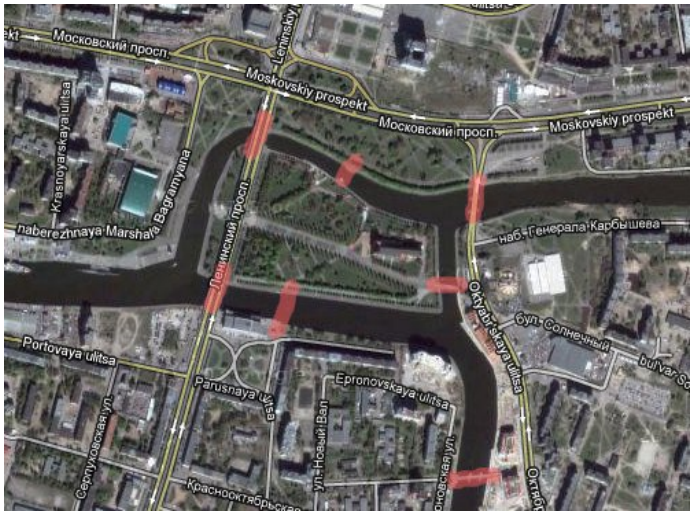


Imagen actual de la ciudad de Kaliningrado, en rojo figuran los puentes del problema original de Königsberg algunos cuales ya no existen hoy en día.

El problema tiene varias variantes, en este trabajo abordaremos la variante *mixta* que consiste en intentar encontrar el mínimo circuito que pase por todas las aristas (no orientadas) y arcos (orientadas) de un grafo donde cada una tiene peso asignado.

2. Situaciones de la vida real

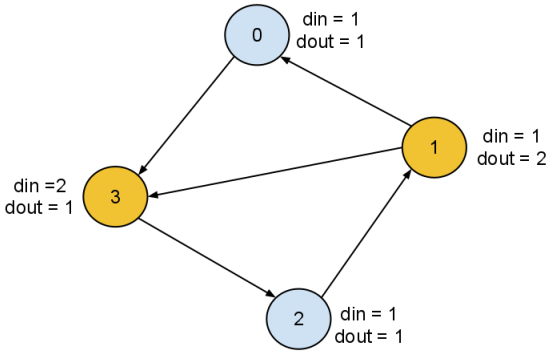
El algorítmo del cartero chino tiene múltiples aplicaciones en la vida cotidiana por ser

- *Camiones de basura*: en donde los camiones deban recorrer las calles de una ciudad y volver al centro de tratamiento de basura. Las calles estarían representadas por aristas y las intersecciones por vértices. Las calles podrían bien tener diferentes sentidos o sentidos únicos, y los pesos podrían corresponder a la cantidad de basura que deban recolectar en ciertas zonas (ej zonas industriales)

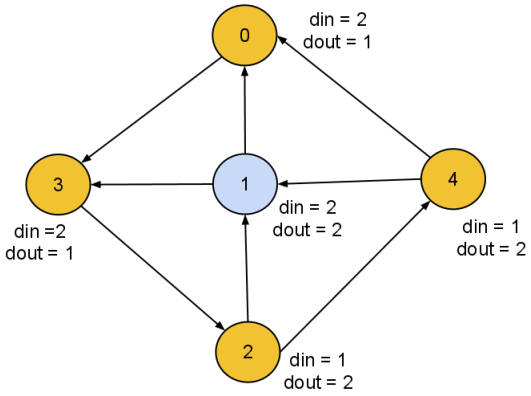
- *Reparto de Volantes:* Cálculo de ruta óptima que pase por todas las calles de una ciudad al menos una vez para poder repartir volantes se publicidad. Las calles estarían representadas por aristas y las intersecciones por vértices. Podría asignarse un peso de 1 a cada arista siendo que no tiene mayor complejidad repartir volantes en una u otra cuadra.

3. Heurística Constructiva

El *algorítmo contructivo* se encargará de tomar un grafo mixto y orientar sus aristas no orientadas y al mismo tiempo establecer un matching inicial.
Un matching consiste en agrupar pares de nodos (v, w) tales que $d_{in}v > 0$ y $d_{out}w > 0$.



En el grafo, los nodos 1 y 3 están desbalanceados con respecto a d_{in} y d_{out}, más aún $d_{in}3 > d_{out}3$ y $d_{out}1 > d_{in}1$ con lo cual pueden formar parte de un matching: [(1,3)]



En el grafo, los nodos 0, 2, 3 y 4 están desbalanceados con respecto a d_{in} y d_{out}, un matching posible para equiparar sus d_{in} y d_{out} sería: [(0,2)(3,4)]

Para ello ideamos diferentes métodos para la orientación de todas las aristas y fuimos experimentando con 5 métodos distintos, todos ellos basándose en los grados de los nodos para la elección de un nodo con el cual empezar a orientar sus aristas. Los métodos elegir:

- 1) Los nodos que tienen igual cantidad de aristas sin orientar que orientadas, y en caso de haber muchos, solo tomará los primeros k nodos de ellos, donde k es un parámetro de entrada de la función.
- 2) Los nodos a partir de un cierto número, que es definido por el parámetro de entrada de la función.

- 3) Los nodos que tengan grado de nodo (sin orientar) mayor o igual al ALFA* nodoGradoMaximo.
- 4) Los nodos que tengan grado de entrada (orientado) mayor o igual al ALFA* nodoGradoMaximo..
- 5) Los nodos que tengan grado de salida (orientado) mayor o igual al ALFA* nodoGradoMaximo.

Se entiende como *nodoGradoMaximo* al nodo de mayor grado, contando aristas no orientadas.

Con cualquiera de estas posibilidades, siempre se intenta hacer un balance entre los grados que quedarán (solo quedarán grados de entrada y salida, ya que será completamente orientado). Realizando pequeñas pruebas terminamos optando por el método número 3) ya que por ejemplo con el método número 1) no podíamos asegurar que siempre existieran tales nodos que cumplieran la condición.

El segundo método se basaba en el parametro para darle la longitud a la lista y no armaba la lista en base al cumplmiento de una condicióón (ej porcentaje de ALFA)

El cuarto y quinto método se buscan los nodos que tengan din o dout $\geq$ a la proporcion ALFA, pero esta proporcion estaba basada en el grado no orientado del más grande y estábamos comparando grados no orientados contra grados orientados que no tenian necesariamente que tener relación.

Eligiendo el método 3), en caso de no encontrar ningún nodo a orientar con estas opciones, encontrá los k primeros nodos (empezando desde 0), sea cual sea el grado de sus nodos. (donde k es el ALFA* nodoGradoMaximo)

Una vez que tenemos todo el grafo orientado, realizamos luego una etapa de matching entre los nodos que quedan con diferentes grado de entrada y de salida, y balanceamos a todos los nodos, para que queden con iguales grados de entrada como de salida, y el grafo quede Euleriano. Este matching lo calculamos con el peso del camino minimo de entre cada uno de los nodos calculado sobre el grafo original utilizando Dantzig, ya que en esta etapa del algorítmo solo nos interesa saber el peso del camino mínimo y no como esta compuesto.

4. Heurística de búsqueda Local

El algorítmo de búsqueda local se encargará, dado uno un grafo orientado completamente, un matching inicial y el grafo original, de buscar en la vecindad un mejor matching. Un matching consiste en agrupar pares de nodos (v, w) tales que $d_{in}v > 0$ y $d_{out}w > 0$.

Lo que haermos para conseguir una solución vecina es intercambiar en un matching dos pares de nodos para obtener así un nuevo matching. Calculamos de esta manera todas las posibles soluciones vecinas intercambiando solo nodos entre dos pares y luego de todas ellas obtenemos la que sea mínima y así poder acercarnos a una solución mejor.

Esta definición de vecindad tiene un tamaño de k^2 donde k es la cantidad de pares en el matching. Esto puede verse ya que para el intercambio de un par con otro existen $k-1$ posibilidades de otros pares. A su vez como la cantidad de pares en el matching es está determinada por la diferencia de grados de entrada y salida de los nodos ésta equivale a

$$\frac{\sum_{i=1}^{n} |din(i) - dout(i)|}{2}$$

$$\frac{\sum_{1 \leq i \leq n / din > dout} din(i) - dout(i) + \sum_{1 \leq i \leq n / dout > din} dout(i) - din(i)}{2} \leq$$

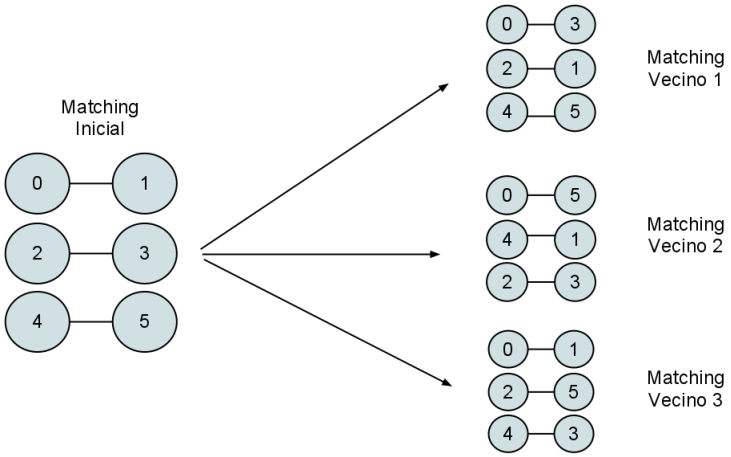
$$\frac{\sum_{1 \leq i \leq n / din > dout} din(i) + \sum_{1 \leq i \leq n / dout > din} dout(i)}{2} \leq$$

$$\frac{m+m}{2} = \frac{2 \cdot m}{2} = m$$

Asi que una vecindad tiene a lo sumo m elementos.

A su vez la cantidad de posibles matchings diferentes que pueden existir es del orden de $k!$ ya en la elección de los pares para un nodo determinado existen $k-1$ posibilidades para su par, una vez establecido para el siguiente nodo habrá $k-2$ posibles pares y así sucesivamente.

Un ejemplo de una vecindad dado un matching inicial sería la siguiente:



Una solución es vecina de otra si y solo si los matchings de ambas soluciones difieren sólo en un intercambio entre dos pares de matchings, así, en el Vecino 1 se han intercambiado los pares (0,1) y (2,3) del matching original, en el Vecino 2 los pares (0,5) y (4,5) y en el Vecino 3 los pares (2,3) y (4,5)

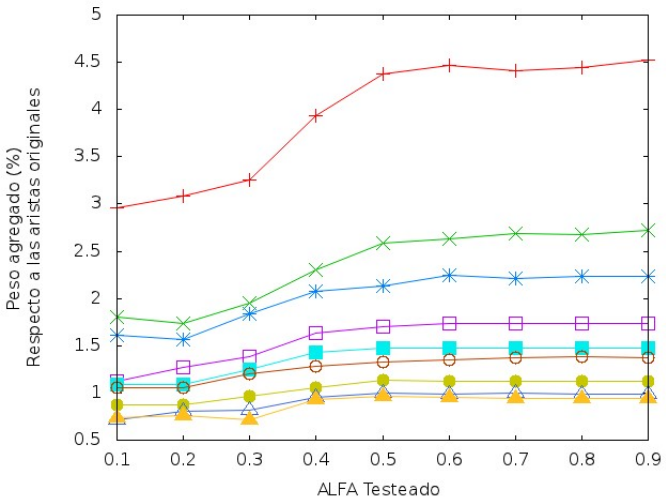
5. Metaheuristica Grasp

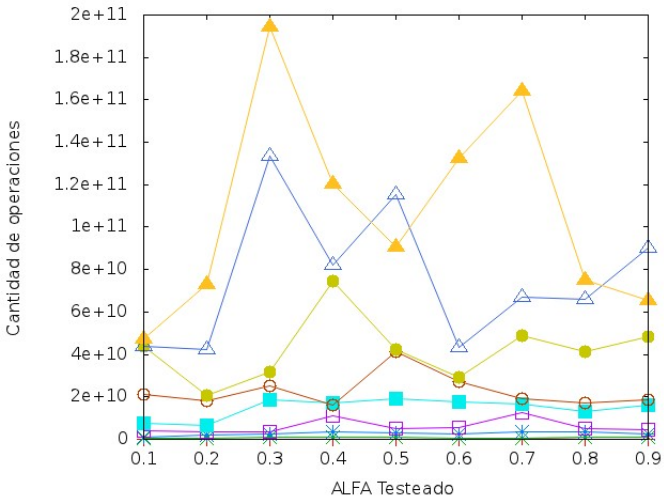
Esta etapa del algorítmo realiza varias iteraciones de los algorítmos de las heurísticas constructiva y de búsqueda local, decidiendo mediante experimentacíon el nivel de aleatoriedad agregado al algorítmo constructivo, el tamaño de la vecindad a visitar y la cantidad de veces que se intentar una solución nueva.

El parametro $CANT\_ITERACIONES\_MAXIMA$ define el máximo de veces que se construirá una solución inicial.
El parametro $CANT\_ITERACIONES\_SIN\_MEJORAR$ se usa como condición de corte, si no se pudo encontrar una solución mejor en tantas iteraciones de $GRASP$ se corta la ejecución y se devuelve la mejor hasta el momento.
El parametro $CANT\_ITERACIONES\_BUSQUEDA\_LOCAL$ define el tamaño maximo que tendra la vecindad ya que este puede ser exponencial en el tamaño del matching.

Como no tenemos el valor real de la solución óptima, medimos la calidad de las soluciones como el porcentaje de peso agregado en la solución con respecto al peso total de las aristas del grafo original.

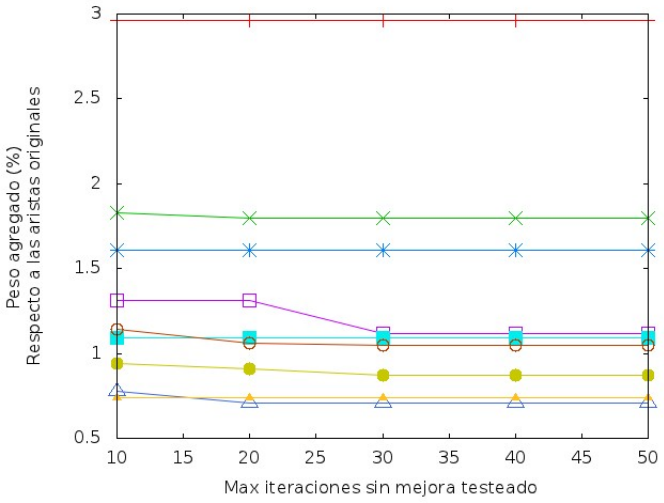
El valor decidido para $ALFA$ fue $0,1$.

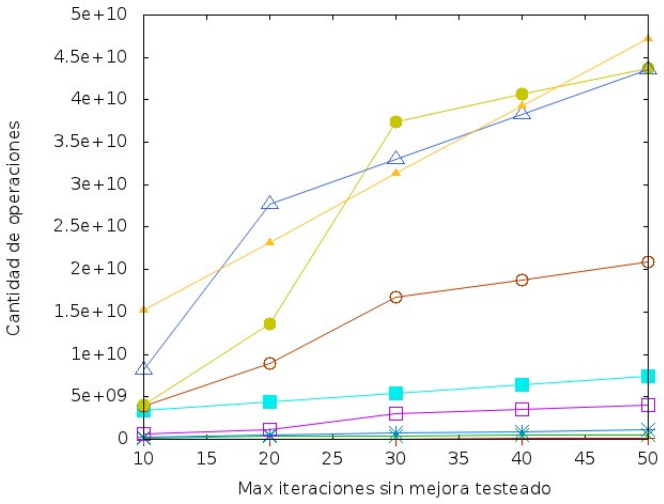




Como se puede observar en las gráficos de las mediciones tomadas para las mismas instancias con distintos valores para $ALFA$, se obtuvieron los mejores resultados para el valor $0,1$ y las complejidades practica para ese valor son bastante razonables.

El valor decidido para $CANT\_ITERACIONES\_SIN\_MEJORAR$ fue 10.





Tomamos esta decisión ya que obsevamos que la complejidad crece mucho y la mejora de la solución no es muy significativa

El valor decidido para $CANT\_ITERACIONES\_MAXIMA$ fue 200, ya que en ninguno de los casos evaluados llegó a cortar por esa cota. El valor decidido para $CANT\_ITERACIONES\_BUSQUEDA\_L$ fue 50.

6. pseudo-algorítmo

El siguiente es un pseudo-código del algorítmo heurístico que realizamos para obtener el circuito Euleriano:

algorítmo GRASP
Mientras no se superen las I iteraciones totales y las J iteraciones sin mejorar de GRASP hacer:

- **Algorítmo Constructivo**
  - 1) Calular Dantzig sobre el grafo.
  - 2) Orientar todas las aristas no orientadas del grafo.
  - 3) Obtener un matching inicial entre los nodos que tengan din $\neq$ dout.

  Fin Algorítmo Constructivo
- **Algorítmo Búsqueda Local**
  Mientras no se superen K iteraciones y no se haya encontrado un matching óptimo hacer
  - 4) Obtener los matchings vecinos y quedarse con el mínimo vecino

  Fin Mientras
  Fin Algorítmo Búsqueda Local
- 5) Calcular los caminos mínimos entre los nodos del matching.
- 6) Calcular el circuito Euleriano

7. Complejidad

Complejidades donde n es la cantidad de nodos, m_1 la cantidad de aristas, m_2 la cantidad de arcos y $m = m_1 + m_2$

- Inicializar grafo (lectura de archivo)
  - Inicializar la matriz de pesos $O(n^2)$
  - Inicializar las listas de adyacencias $O(n + m_1 + m_2)$

  En total: $O(n^2 + m_1 + m_2) \subseteq O(n^2)$
- FuertementeConexo
  - Inicializacion de listas de adyacencias $O(n + m_1 + m_2)$
  - resetVisitados y checkVisitados recorren un array de bools de tamaño n. $O(n)$
  - visitarNodos recorre recursivamente en DFS las adyacencias y marca los nodos que son accesibles desde el nodo 0 $O(m_1 + m_2)$

  En total: $O(n + m_1 + m_2)$
- Calcular Dantzig
  - Inicializacion de la matriz de pesos de caminos minimos $O(n^2)$
  - 3 for anidados de 0 a n con operaciones de tiempo constante $O(n^3)$

  En total: $O(n^3)$
- sumaPesosEjes recorre todas las aristas y arcos de los nodos $O(m_1 + m_2)$
- Clonar el grafo $O(n^2 + m_1 + m_2)$ para copiar las estructuras
- orientarTodasAristas
  - m_1 iteraciones de
    - encontrarNodoAOrientar: Busca cuales nodos cumplen la condicion dada $O(n)$
    - eleccionNodoAOrientar: Elije segun el parametro pasado por cual nodo de la lista anterior va a continuar $O(n)$
    - orientarNodo $O(1)$

  $O(1 + n + n) subset O(n)$

  En total: $O(m_1 * n)$

- encontrarMatchingNodos
  - Inicializacion de arreglos de longitud n $O(n)$
  - para cada exedente de Din arma una pareja con un $Dout$ y lo agrega en una lista con el peso calculado con $Dantzig$ $O(\sum_{i=1}^{n} |din(i) - dout(i)|/2) \subseteq O(m)$

  En total: $O(m + n)$
- encontrarMatchingDeMenorPeso: Dado un matching de tamaño k devuelve el mejor matching que se puede obtener swapeando los destinos de solo dos items del matching original.
  En total: $O(k^2)$
- pesoMatching recorre una lista de tamaño a lo sumo
  En total: m $O(m)$
- agregarCaminosMatcheados
  - Inicializacion de matriz de tamaño $n * n$ $O(n^2)$
  - Para cada eje (i, j) del matching (a lo sumo m ejes)
    - Si no esta calculado el $Dijkstra$ para el nodo $i \rightarrow$ Calcula $Dijkstra$ sobre el grafo original y lo guarda $O(n^2)$
    - Recuperar el camino del $Dijkstra$ calculado para el nodo i hasta el nodo j $O(n)$

  $O(n^2)$ Si no esta calculado $Dijkstra$ (a lo sumo n casos)
  $O(n)$ Si ya esta calculado $Dijkstra$

  En total: $O(m * n + n * n^2)$ como $O(m) \subseteq O(n^2) \Rightarrow O(m * n + n^3) \subseteq O(n^3)$
- Dijkstra
  - Inicializacion de arreglos de int y bool de tamaño n $O(n)$
  - Para cada nodo i (n elementos)
    - Minvertex $O(n)$
    - Para cada adyacente del nodo i actualiza el valor $O(d(i))$

  En total: $O(n + \sum_{i=1}^{n} (d(i) + n)) \subseteq O(n + 2 * m + n * n) \subseteq O(n^2)$
- encontrar circuito euleriano $O(2 * largodelcircuito)$
  Donde largo del circuito puede ser en peor caso $m*n$ cuando todos los nodos tiene matching y todos los caminos agregados tiene longitud n
  En total: $O(m * n)$

Complejidad:

- $O(n^2)$ de inicializacion
- $O(n + m_1 + m_2)$ de fuertementeConexo
- $O(n^3)$ de Dantzig
- $O(m_1 + m_2)$ de sumarPesosEjes
- $MaximoIteracionesGrasp$*
  - $O(n^2 + m_1 + m_2)$ de copiar el grafo
  - $O(m_1 * n)$ de orientarTodasLasAristas
  - $O(m_1 + m_2 + n)$ de encontrarMatchingNodos
  - $MaximoIteracionesBusquedaLocal$*
    - $O(m^2)$ de encontrarMatchingDeMenorPeso
  - $O(m)$ de calcular peso del matching
- $O(n^3)$ de agregarCaminosMatcheados
- $O(m * n)$ de encontrar el CircuitoEuleriano

Para grafos que no son fuertemente conexos la complejidad es $O(n^2 + m)$
Para las instancias interesantes del problema la complejidad esta dada por:
$O(n^2 + n + m + n^3 + m +$
$MaximoIteracionesGrasp * (n^2 + m + m_1 * n + m + n + MaximoIteracionesBusquedaLocal *$
$(m^2) + m) + n^3 + m * n) \subseteq O(n^3 + m^2 + m * n) \subseteq O(n^3 + m^2)$

Definimos el tamaño de entrada como:

$T = log(n)+log(m_1)+log(m_2)+\sum_{i=1}^{m_1+m_2} (log(arista_{i_{nodo1}}) + log(arista_{i_{nodo2}}) + log(arista_{i_{peso}}))$

Como $log(n) + log(m_1) + log(m_2) > 0 \Rightarrow$

$$T > \sum_{i=1}^{m_1+m_2} (log(arista_{i_{nodo1}}) + log(arista_{i_{nodo2}}) + log(arista_{i_{peso}}))$$

Tambien sabemos que para cada eje $log(arista_{i_{nodo1}}) + log(arista_{i_{nodo2}}) + log(arista_{i_{peso}}) > 1 \Rightarrow$

$$T > \sum_{i=1}^{m_1+m_2} 1 = m_1 + m_2 = m$$

Sabemos que el grafo es fuertemente conexo asi que n es del orden de $m \Rightarrow$

$$T > m \geq n$$

Dado la complejidad calculada y las cotas para T vale $O(n^3 + m^2) \subseteq O(T^3 + T^2) \subseteq O(T^3)$

Algorithm 1 Main($grafo$)

```
Grafo GrafoOriginal ← Inicializar con instancia
if es fuertemente Conexo GrafoOriginal then
  PesosCaminosMinimos ← calcularDantzig(GrafoOriginal)
  SumaOriginalPesos ← sumaPesosEjes(GrafoOriginal)
  SumaSolucionMejor ← ∞
  GrafoCopiaMejor ← NULL
  MatchingMejor ← ∅
  while no se alcanzan MaximoIteracionesGrasp ∨ MaximoIteracionesSinMejoraGrasp
  do
    Grafo GC ← copiarGrafo(GrafoOriginal)
    ParametroGRASP ← Random
    orientarTodasAristas(GC,ParametroGRASP)
    matching ← encontrarMatchingNodos(GC,ParametroGRASP)
    while no se alcanza MaximoIteracionesBusquedaLocal ∨ no se encontro un matching
    optimo do
      matching ← encontrarMatchingDeMenorPeso(matching,pesosCaminosMinimos)
    end while
    pesoMatching ← pesoMatching(matchingMejor)
    sumaSolucion ← pesoMatching + SumaOriginalPesos
    if mejoro la solucion then
      SumaSolucionMejor ← sumaSolucion
      GrafoCopiaMejor ← GC
      MatchingMejor ← matching
    end if
  end while
  agregarCaminosMatcheados(GrafoCopiaMejor,MatchingMejor,GrafoOriginal)
  circuito ← encontrarCircuitoEuleriano(GrafoCopiaMejor)
  return circuito
else
  return No hay Tour porque el grafo no era fuertemente conexo
end if
```

8. Tests

9. Gráficos