Algoritmos para la determinación del circuito de menor longitud en un grafo

Algorithms determining the minimum length directed cycle in a graph

Rayco Hernández Delgado

Trabajo de Fin de Grado

Facultad de Matemáticas

Universidad de La Laguna

La Laguna, 12 de julio de 2016

Dr. D. **Antonio Alberto Sedeño Noda**, con N.I.F. 45439024-V profesor Titular de Universidad adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

"*Algoritmos para la determinación del circuito de menor longitud en un grafo.*"

ha sido realizada bajo su dirección por D. **Rayco Hernández Delgado**, con N.I.F. 54053308-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 12 de julio de 2016

Agradecimientos

En primer lugar, quiero mostrar mi agradecimiento a mi tutor, Antonio Alberto Sedeño Noda, por toda la paciencia y el tiempo que me ha prestado durante la realización de este trabajo. A todos los profesores que han contribuido tanto en mi desarrollo académico como personal.

Y finalmente, quiero agradecer a mi familia por su constante apoyo durante estos años.

Resumen

El objetivo de este trabajo ha sido dar a conocer la importancia que está teniendo los grafos en la actualidad. Existen una gran cantidad de aplicaciones prácticas en el contexto del uso social empresarial de las tecnologías actuales, que usan grafos para almacenar información. La necesidad de acceder a dicha información lo mas rápido posible, nos lleva a la búsqueda de los algoritmos para la determinación del circuito de menor longitud en un grafo. Por ejemplo, en este trabajo comentaremos algunos de los algoritmos ya existentes, sus pros y contras, y los implementaremos en C, de estos, intentaremos introducir un nuevo algoritmo que en la práctica sea eficiente comparándolo con los algoritmos ya existentes.

Palabras clave: Grafo dirigido, mínimo longitud en un ciclo, algoritmo, experimentación

Abstract

The aim of this study was to publicize the importance that graphs have today. This is due to the great amount of practical applications that they may have today. For example, many applications use the graphs to store information. They become useful tool to model and to access to the information as quickly as possible. In this context, we consider the minimum directed cycle in a directed graph problem. In this paper, we make an experimental study about of the state-of-the-art algorithms solving this problem. In addition, we try to introduce a new algorithm that in practice becomes more efficient than existing algorithms.
Keywords:

Keywords: *directed graph, minimum length cycles, algorithm, experimentation*

Índice general

1. Motivación y objetivos **1**
1.1. Motivación 1
1.2. Objetivos 2

2. Fundamentos teóricos **3**
2.1. Introducción 3
2.2. Definiciones 3
2.2.1. Grafo 3
2.2.2. Cadena y ciclo 3
2.2.3. Camino y circuito 4
2.2.4. Peso y longitud 4
2.2.5. Predecesores y Sucesores 4
2.2.6. Adyacencia 5
2.2.7. Conectividad y conexidad 5
2.2.8. Árbol 5
2.3. Tipos de Grafos 6
2.3.1. Grafos Simples 6
2.3.2. Grafos Completos 6
2.3.3. Grafos Planares 6
2.3.4. Grafos Dirigidos 7
2.3.5. Grafos Ponderados 7
2.3.6. Grafos Etiquetados 8
2.3.7. Grafos Bipartitos 8
2.4. Representación de Grafos en un ordenador. 9
2.4.1. Listas de Adyacencia 9
2.4.2. Matriz de Adyacencia 9
2.5. Algoritmos 10
2.5.1. Asignación de Variables 10
2.5.2. Objetivos 10
2.5.3. Herramientas esenciales 11
2.5.4. Bellman-End-Ford 11
2.5.5. Dijkstra 12
2.5.6. Floyd y Warshall 13

2.5.7. Johnson 15
2.6. Nuevo Algoritmo (mldc) 17
2.6.1. La clave 17
2.6.2. Ecuaciones de recurrencia para calcular las distancias parciales. . . . 18
2.6.3. Algunas propiedades para mejorar la aplicación de las ecuaciones recurrentes (1) y (2). 21
2.6.4. Un ejemplo. 25

3. Procedimiento experimental **28**
3.1. Descripción del material 28
3.2. Descripción de los experimentos 28
3.3. Resultados obtenidos 30
3.4. Análisis de los resultados 40

Conclusiones **41**

Bibliografía **42**

Índice de Figuras

1.1. Ejemplo del uso de un Grafo en el GPS 1
1.2. Ejemplo del uso de un Grafo para la búsqueda de un viaje más barato . . . 2

2.1. Grafo Simple 6
2.2. Grafo Completo 6
2.3. Grafo Plano 6
2.4. Grafo Dirigido 7
2.5. Grafo Ponderado 7
2.6. Grafo Etiquetado 8
2.7. Grafo Bipartito 8
2.8. Grafo dirigido y no dirigido guardado en Lista 9
2.9. Grafo no dirigido y dirigido guardado en Matriz 9
2.10. Ejemplo de Bellman-Ford 12
2.11. Ejemplo de Dijkstra 13
2.12. Ejemplo de Floyd y Warshall 14
2.13. Ejemplo de Johnson 16
2.14. Ejemplo de un grafo con longitudes en los arcos. 25
2.15. Traza del algoritmo del MLDC en un ejemplo 26

3.1. Gráfica usando el Netgen para n 31
3.2. Gráfica usando el Netgen para $\frac{m}{n}$ 32
3.3. Gráfica usando el Gridgen para n 33
3.4. Gráfica usando el Gridgen para $\frac{m}{n}$ 34
3.5. Gráfica usando el Hpgen para n 35
3.6. Gráfica usando el Hpgen para $\frac{m}{n}$ 36
3.7. Gráfica usando el Netgen para n 38
3.8. Gráfica usando el Netgen para $\frac{m}{n}$ 39

Índice de Tablas

3.1. Tabla del Netgen para n 31
3.2. Tabla del Netgen para $\frac{m}{n}$ 32
3.3. Tabla del Gridgen para n 33
3.4. Tabla del Gridgen para $\frac{m}{n}$ 34
3.5. Tabla del Hpgen para n 35
3.6. Tabla del Hpgen para $\frac{m}{n}$ 36
3.7. Tabla del Netgen para n con arcos negativos 38
3.8. Tabla del Netgen para $\frac{m}{n}$ con arcos negativos 39

Capítulo 1

Motivación y objetivos

1.1. Motivación

Este proyecto lo he querido desarrollar por la importancia que ha adquirido la determinación de los caminos mínimos en un grafo en la actualidad. Algunos ejemplos en la práctica son los siguientes:

- Querer saber el camino mínimo para ir desde un punto A a un punto B en un GPS.

Figura 1.1: Ejemplo del uso de un Grafo en el GPS

- Las rutas de los autobúses para optimizar los costes usando los caminos mínimos para llevar a la mayor cantidad de personas desde donde estén hasta donde quieren ir.

- En las redes sociales como facebook, twitter, etc. los datos son almacenados/estructurados como grafos. Al querer acceder a la información y realizar consultas que den un valor añadido a los datos que se manejan (BIG DATA) necesitaremos herramientas de consulta (algoritmos) que tarden el menor tiempo posible.
- Al buscar un vuelo en internet, la página usa un grafo donde los nodos son lugares (salidas, llegadas o escalas), los trayectos son los arcos entre dichos lugares y dichos arcos tienen diferentes pesos (Tiempo y coste en realizar el trayecto entre otros). Y los usuarios pueden usar filtros para buscar las ofertas que se ajustan a sus deseos, ahí tenemos un claro ejemplo de grafo y búsqueda del camino con el mínimo coste/tiempo para llegar desde un punto A hasta un punto B, y el usuario necesita un buscador eficiente en dicha búsqueda, ahí es donde entran los métodos estudiados en este trabajo, métodos que buscan la mínima longitud en un circuito en el menor tiempo posible.

Figura 1.2: Un ejemplo sería, sale más barato ir desde Tenerife a Málaga y desde Málaga hasta Granada en autobús, que desde Tenerife a Granada con escala en Madrid y esto se puede hacer gracias a una búsqueda de minizar el coste del camino a realizar entre Tenerife y Granada. (transporte multimodal)

1.2. Objetivos

Nos dirigimos a la determinación del circuito de menor longitud en un grafo dirigido (Minimum Length Directed Cycle - MLDC) con n nodos, m arcos y sin ciclos dirigidos de longitud negativa.

Capítulo 2

Fundamentos teóricos

2.1. Introducción

Tradicionalmente, el problema **MLDC** ha sido resuelto a través de la resolución del problema de todos los caminos mínimos entre todos los pares de nodos (All Pairs Shortest Path - **APSP**) utilizando el algoritmo de **Floyd y Warshall (1962)** o el algoritmo de **Johnson (1977)**, este último utilizado para grafos dispersos. Las complejidades teóricas de los algoritmos mencionados son $O(n^3)$ **y** $O(n \cdot m + n^2 \cdot log(n))$, respectivamente. **Pettie (2004)** proporciona un algoritmo mejorado para grafos dispersos que requiere un tiempo $O(n \cdot m + n^2 \cdot log(log(n)))$. Los algoritmos anteriores sólo utilizan métodos combinatorios. **Roditty y Williams (2011)** introduce un algoritmo de tiempo $\tilde{O}(M_n^{2,376})$ (los términos polilogarítmicos se omiten) utilizando un algoritmo de multiplicación de matrices rápido para resolver el **MLDC** en grafos dirigidos con pesos enteros en el intervalo $[-M, M]$. Los algoritmos basados en la multiplicación rápida de matrices tienen limitaciones cuando se utilizan en la práctica porque ocurren fenómenos de desbordamiento de la pila asociados a algunos procedimientos recursivos que estos usan.

2.2. Definiciones

2.2.1. Grafo

- Un Grafo $G := (V, A)$, donde $V := \{v_0, v_1, v_2, ..., v_n\}$ es un conjunto de n elementos denominados **vértices** o nodos y $A := \{a_0, a_1, a_2, ..., a_m\}$ es un conjunto ordenado de pares de nodos de tamaño m que reciben el nombre de **arcos** y si no está ordenado se denominan **aristas**.

2.2.2. Cadena y ciclo

Sea $G = (V, A)$ un grafo,

- **Cadena.-** Una cadena de longitud q es una secuencia de $q-aristas\ C := \{a_1, ..., a_q\}$, tal que una arista a_r de la secuencia ($2 \leq r \leq q$–1) tienen un extremo común con

la arista a_{r-1} ($a_{r-1} \neq a_r$) y otro extremo común con la arista a_{r+1} ($a_{r+1} \neq a_r$). El primer vértice de la primera arista y el último vértice de la última arista se denominan los extremos de la cadena.

- **Cadena Simple.-** Es una cadena que no atraviesa una arista cualquiera más de una vez.
- **Cadena Elemental.-** Es una cadena que no atraviesa ningún vértice más de una vez. O, de otro modo, una cadena en la que sus vértices no tienen grado mayor que 2. Toda cadena elemental es simple.
- **Ciclo.-** Es una cadena en el que sus extremos coinciden.
- **Acíclico** Un grafo se dice acíclico si no contiene ciclos.

2.2.3. Camino y circuito

- **Camino.-** Sea $G = (V, A)$ un grafo, un camino en el grafo G es una sucesión en la que aparecen de forma alternativa elementos de V y de A de la forma $v_0, a_1, v_1, a_2, v_2, ..., v_{k-1}, a_k, v_k$ con $v_i \in V\ \forall i \in \{0, ..., k\}$ y $a_j \in A\ \forall j \in \{1, ..., k\}$. y donde v_{i-1} es **adyacente** con v_i mediante un arco a_i. Denotaremos el camino p_{st} como el camino dirigido desde v_s hasta v_t.
- **Circuito o ciclo dirigido.-** Es un camino en el que sus extremos coinciden.
- **MLDC.-** El problema del MLDC consiste en buscar el ciclo dirigido o circuito de menor longitud dentro del grafo.

2.2.4. Peso y longitud

- **Peso.-** Definimos Peso como el coste de una arista, puede ser el tiempo, la distancia o el coste en dinero de pasar de un vértice al vértice adjunto a él, y lo denotaremos como $w(u, v)$ como el peso de la arista/arco (u, v).
- **Longitud.-** Definimos la longitud de un camino en un grafo como la suma de los pesos de los arcos de ese camino. Denotaremos como $v_{ij}.d$ a la suma de los pesos de las aristas en el camino p_{ij}.

2.2.5. Predecesores y Sucesores

Dado un Grafo G dirigido,

- **Sucesor.-** Denotaremos a $\Gamma_v^+ := \{u \in V/(v, u) \in A\}$ como el conjunto de los **sucesores** del vértice v, $\forall v \in V$.
- **Predecesor.-** Denotaremos a $\Gamma_v^- := \{u \in V/(u, v) \in A\}$ como el conjunto de los **predecesores** del vértice v, $\forall v \in V$.

2.2.6. Adyacencia

- Dado un Grafo G no dirigido, denotaremos a $\Gamma_v := \{u \in V/(v,u) \in A\}$ como nodos adyacentes al nodo v. Por lo tanto, la versión dirigida de un grafo no dirigido se obtiene considerando el mismo conjunto de nodos y por cada arista $(u,v) \in A$ se añade el arco (v,u) en A. Por tanto, $\Gamma_v^+ = \Gamma_v^-$ para la versión dirigida de un grafo no dirigido.

2.2.7. Conectividad y conexidad

- **Conectividad.-** Decimos que dos nodos v_i y v_j están conectados si el grafo contiene al menos una cadena desde el nodo v_i al nodo v_j.
- **Conexo.-** Un grafo es conexo si todo par de nodos están conectados; en otro caso, el grafo no es conexo.
- **Componente conexa.-** Llamaremos componentes conexas de un grafo no conexo a los subgrafos de máxima conexiones de dicho grafo.
- **Conectividad fuerte.-** Un grafo conexo es fuertemente conexo si existe al menos un camino entre cualquiera par de nodos del grafo.
- **Componente fuertemente conexa fuerte.-** Llamaremos componentes fuertemente conexas de un grafo a los subgrafos maximales formado por nodos fuertemente conectados.

2.2.8. Árbol

- **Árbol.-** Un árbol T es un grafo conexo que no contiene ciclos. Asumiremos, las siguientes propiedades elementales de los árboles:
  - Un árbol de n nodos continene exactamente $n-1$ arcos.
  - Un árbol tiene al menos dos nodos hoja (nodos con grado 1).
  - Cada dos nodos de un árbol están conectados por una única cadena o camino.
- **Árbol Generador.-** un árbol T es un árbol generador del grafo G si T es un subgrafo generador de T. Todo árbol generador de un grafo conexo G de n nodos tiene $(n-1)$ aristas.

2.3. Tipos de Grafos

2.3.1. Grafos Simples

- Un grafo es simple si a lo sumo existe una arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos. Un grafo que no es simple se denomina **multigrafo**.

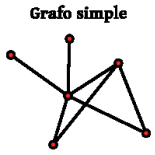


Figura 2.1: Ejemplo de Grafo Simple

2.3.2. Grafos Completos

- Un grafo es completo si existen aristas uniendo todos los pares posibles de vértices. Es decir, todo par de vértices (v_i, v_{i+1}) debe tener una arista a que los une. El conjunto de los grafos completos es denominado usualmente K, siendo K_n el grafo completo de n vértices. Un K_n, es decir, grafo completo no dirigido de n vértices tiene exactamente $\frac{n(n-1)}{2}$ aristas.

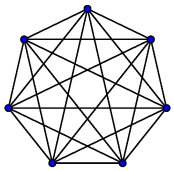


Figura 2.2: Ejemplo de Grafo Completo

2.3.3. Grafos Planares

- Se dice que un grafo es plano si puede dibujarse en el plano de manera que ningún par de sus aristas se corte entre sí. A ese dibujo se le llama representación plana del grafo.

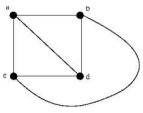


Figura 2.3: Ejemplo de Grafo Plano

2.3.4. Grafos Dirigidos

- Son grafos en los cuales se ha añadido una orientación a las aristas, representada gráficamente por una flecha.

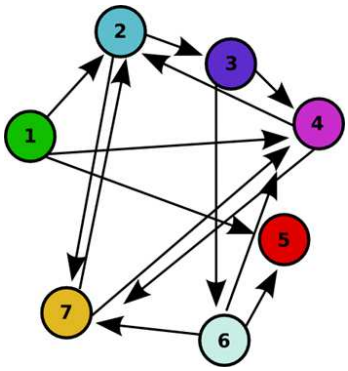


Figura 2.4: Ejemplo de Grafo Dirigido

2.3.5. Grafos Ponderados

- Llamamos grafos ponderados a los grafos en los que se asigna un número a cada una de las aristas. Este número representa un peso para el recorrido a través de la arista. Este peso podrá indicar, por ejemplo, la distancia, el costo monetario o el tiempo invertido, entre otros.

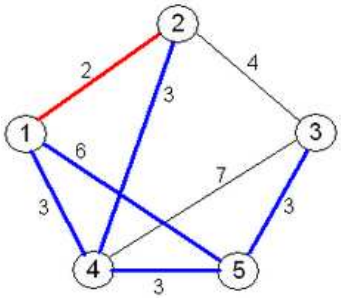


Figura 2.5: Ejemplo de Grafo Ponderado

2.3.6. Grafos Etiquetados

- Grafos en los cuales se ha añadido un etiquetado a los vértices.

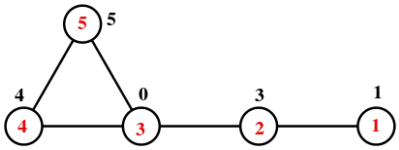


Figura 2.6: Ejemplo de Grafo Etiquetado

2.3.7. Grafos Bipartitos

- Un grafo G es bipartito si puede expresarse como $G = (V_1 \cup V_2, A)$ (es decir, sus vértices son la unión de dos grupos de vértices), bajo las siguientes condiciones:
  - V_1 y V_2 son disjuntos y no vacíos.
  - Cada arista de A une un vértice de V_1 con uno de V_2 .
  - No existen aristas uniendo dos elementos de V_1; análogamente para V_2 .
  - No existe ciclo de cardinalidad par.

  Bajo estas condiciones, el grafo se considera bipartito, y puede describirse informalmente como el grafo que une o relaciona dos conjuntos de elementos diferentes, como aquellos resultantes de los ejercicios y puzzles en los que debe unirse un elemento de la columna A con un elemento de la columna B.

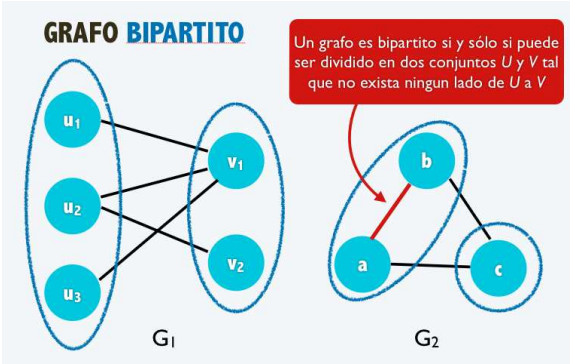


Figura 2.7: Ejemplo de Grafo Bipartito

2.4. Representación de Grafos en un ordenador.

Dado un gráfo $G = (V, A)$.

- Puede ser dirigido o no dirigido.
- Las dos maneras de las más comunes de representar para un grafo son:
  1. Listas de adyacencia.
  2. Matriz de adyacencia.

Se considera que el tamaño de un grafo es determinado por $|V| = n$ y $|A| = m$.

2.4.1. Listas de Adyacencia

- En muchas ocasiones, la representación de un grafo es más eficiente si todas las aristas que salen de un vértice se agrupan juntas. En esta representación se guardan n listas enlazadas (una por cada vértice), con el origen de cada lista dentro de un array. Para cada sucesor de un vértice se guarda la información del vértice destino, del costo de la arista y un puntero al siguiente elemento de la lista.

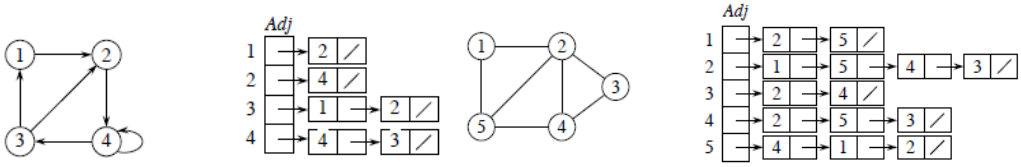


Figura 2.8: Ejemplo: Para un Grafo dirigido y no dirigido.

2.4.2. Matriz de Adyacencia

- Sea $G = (V, A)$ un 1-grafo dirigido (no existe más de una arista entre dos vértices i y j). La **matriz de adyacencia** es (a_{ij}) donde $a_{ij} := \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{En otro caso} \end{cases}$

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	1	1	0	0
4	0	0	1	1

Figura 2.9: Ejemplo: Para un Grafo no dirigido y dirigido.

En nuestro caso, usaremos las **Listas de Adyacencias**, porque ocupan mucho menos memoria en el ordenador, y para grafos muy grandes, se nota la diferencia de espacio y tiene que realizar menos operaciones al tener un tamaño menor.

2.5. Algoritmos

2.5.1. Asignación de Variables

- Sea $G = (V, A)$ un Grafo, computacionalmente, será un vector con dos casillas, la primera $G.V$ será una lista de todos los vértices del Grafo y la segunda $G.A$ serán listas de Adyacencias como las que hemos visto en el apartado anterior.
- p_{ij} será el camino que une los vértices v_i y v_j.
- $w(u, v)$ será el peso del arco que une los vértices u y v.
- $v_{ij}.d$ es la longitud del camino que une v_i y v_j.
- Sea $v_s :=$ el vértice inicial.
- $n :=$ número de vértices o nodos.
- $m :=$ número de aristas o arcos.
- Sea $v_i \in V$ un vértice, computacionalmente, será un vector con dos componentes, $v_i.d := v_{si}.d$ que representa la longitud del camino de v_s a v_i y $v_i.\pi$ que almacena el predecesor de v_i usando ese camino.

2.5.2. Objetivos

- El objetivo de estos algoritmos es que terminen sacando una matriz de tamaño $n \times n$ con elementos v_{ij} que representarán los caminos mínimos entre el nodo v_i y el v_j, teniendo dos componentes, $v_{ij}.d$ y $v_{ij}.\pi$.

$$v_{ij} := \begin{pmatrix} 0|NIL & 1|3 & -3|4 & 2|5 & -4|1 \\ 3|4 & 0|NIL & -4|4 & 1|2 & -1|1 \\ 7|4 & 4|3 & 0|NIL & 5|2 & 3|1 \\ 2|4 & -1|3 & -5|4 & 0|NIL & -2|1 \\ 8|4 & 5|3 & 1|4 & 6|5 & 0|NIL \end{pmatrix}$$

  En este ejemplo, el camino mínimo entre el nodo v_5 y el nodo v_2 es el elemento $v_{5\,2}$. La longitud de ese camino es $v_{5\,2}.d = 5$. Su predecesor es el $v_{5\,2}.\pi = v_3$. El predecesor del v_3 en ese camino es $v_{5\,3}.\pi = v_4$ y el del $v_{5\,4}.\pi = v_5$, por lo tanto, el camino mínimo entre el nodo v_5 y el nodo v_2 es $\{5 \to 4 \to 3 \to 2\}$ con una longitud 5.

- En esta sección veremos que al programar algoritmos de caminos mínimos en grafos ponderados hay que tener en cuenta dos datos importantes:

  1. Saber si hay pesos negativos.
  2. Saber si hay circuitos negativos y como detectarlos, ya que, en proncipio, el problema no tendría solución.

2.5.3. Herramientas esenciales en los algoritmos de caminos mínimos de un origen al resto de nodos

- **Inicialización**(G, v_s)

  1. **for** $i := 1$ **to** n
  2. $\quad v_i.d := \infty$;
  3. $\quad v_i.\pi := NIL$
  4. $v_s.d := 0$

- **Comparación de etiquetas distancia**$(u, v, w(u, v))$

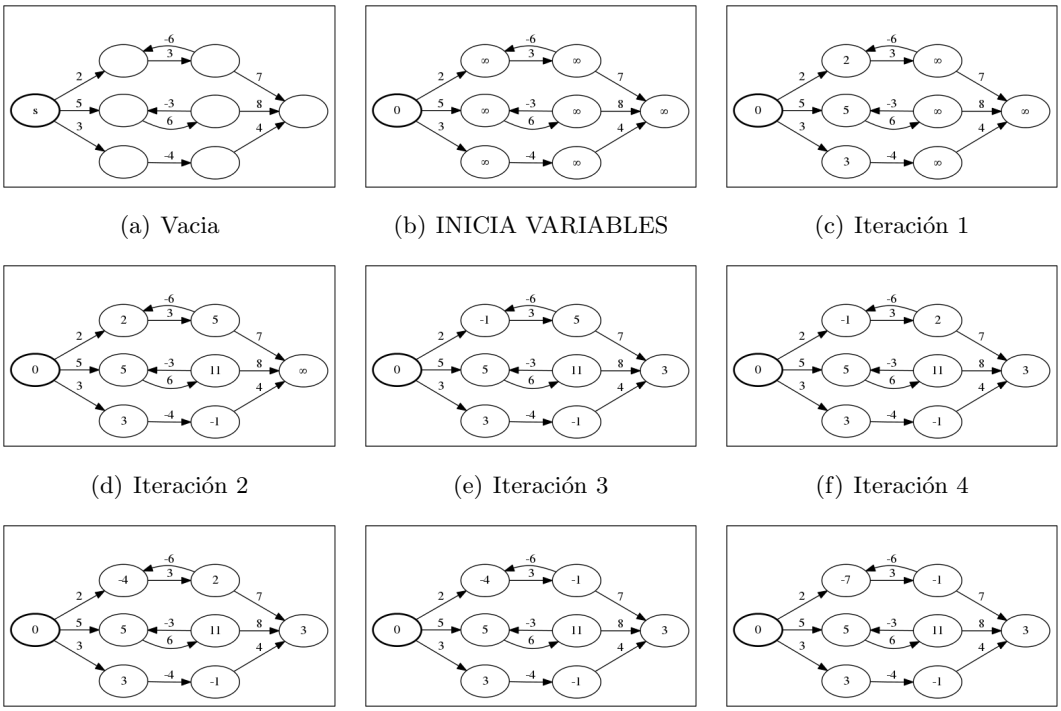
  1. **if** $v.d > u.d + w(u, v)$ **then**
  2. $\quad v.d := u.d + w(u, v)$
  3. $\quad v.\pi := u$

2.5.4. Bellman-End-Ford (1958): Complejidad del Algoritmo = $\boldsymbol{O}(n \cdot m)$

Desarrollado por Richard Bellman, Samuel End y Lester Ford, genera el camino más corto en un grafo dirigido ponderado (en el que el peso de las aristas puede ser negativo). Es capaz de detectar si el grafo contiene un ciclo de longitud negativa, pero en este caso, no encontrará el camino más corto que no repite ningún vértice.

- **BELLMAN-END-FORD**(G, w, s)

  1. **Inicialización**(G, v_s)
  2. **for** cada arco $(u, v) \in G.A$
  3. $\quad$ **for** $j := 1$ **to** m
  4. $\quad\quad$ **Comparación de etiquetas distancia**$(u, v, w(u, v))$
  5. **for** cada arco $(u, v) \in G.A$
  6. $\quad$ **if** $v.d > u.d + w(u, v)$ **then**
  7. $\quad\quad$ **return** FALSE
  8. **return** TRUE



(a) Vacia (b) INICIA VARIABLES (c) Iteración 1

(d) Iteración 2 (e) Iteración 3 (f) Iteración 4

(g) Iteración 5 (h) Iteración 6 (i) Iteración 7

Figura 2.10: **Bellman-Ford:** En este ejemplo, el algoritmo sacaría un **FALSE** ya que $v_2.d > v_3.d + w(v_2, v_3)$ $(-1 > -7 + 3)$.

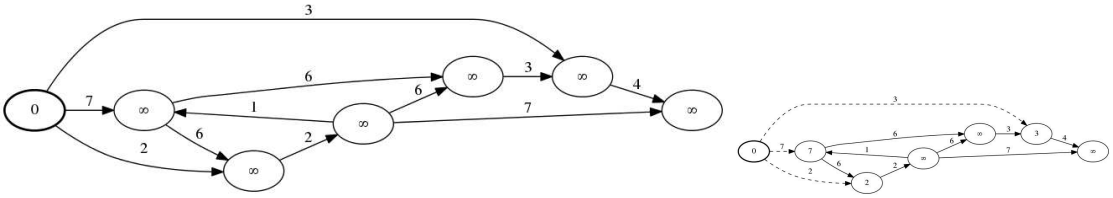
2.5.5. Dijkstra (1959): Complejidad del Algoritmo = $O(\ m + n \cdot log(n)\ )$

Este algoritmo resuelve el problema caminos mínimos en un grafo ponderado dirigido de pesos no negativos. Este algoritmo es mucho más eficiente que el algoritmo de Bellman-End-Ford, pero, solo funciona en el caso de que no existan arcos con pesos negativos.

Por ese motivo, supondremos que $\forall$ arco $(v_i,v_j) \in A$, $w(v_i, v_j) \geq 0$. Este algoritmo mantiene un conjunto S de vértices con las longitudes finales del camino mínimo desde el origen v_s, las cuales ya han sido determinadas. El algoritmo selecciona el vértice $v_i \in \{V - S\}$ de longitud mínima, lo añade a S, y rebaja la longitud de todos los vértices de los cuales v_i es predecesor siempre que su longitud actual sea mayor que la suma de la longitud de v_i más el peso del arco que une v_i con ese vértice.

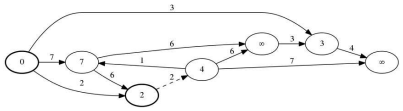
- **DIJKSTRA**(G, w, v_s)

  1. **Inicialización**(G, v_s)
  2. $S := \emptyset$; $\quad Q := G.V$
  3. **while** $Q \neq \emptyset$ **do**
  4. $\quad u := Extrae\ Min(Q)$
  5. $\quad S := S \cup u$
  6. $\quad$ **for** cada vétice $v \in$ *Adyacientes de* $[u]$
  7. $\quad\quad$ **Comparación de etiquetas distancia**$(u, v, w(u, v))$

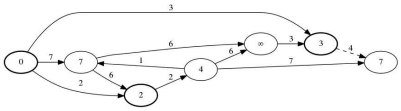


(a) Inicialización

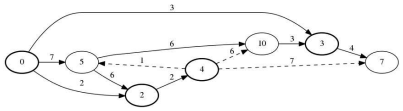
(b) $Q = \{v_2, v_3, v_4, v_5, v_6, v_7\}$



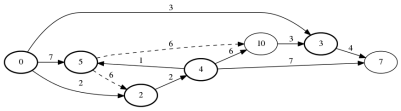
(c) $Q = \{v_2, v_4, v_5, v_6, v_7\}$



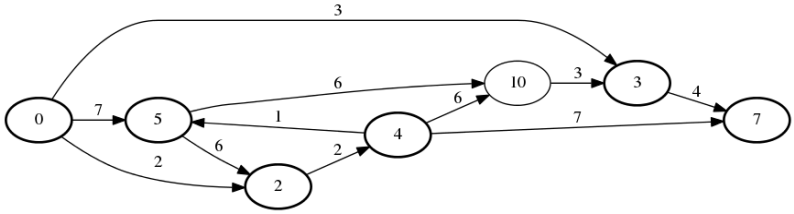
(d) $Q = \{v_2, v_4, v_5, v_7\}$



(e) $Q = \{v_2, v_5, v_7\}$



(f) $Q = \{v_5, v_7\}$



(g) $Q = \{v_5\}$

Figura 2.11: **Dijkstra**

2.5.6. Floyd y Warshall (1962): Complejidad del Algoritmo = $O(\ n^3\ )$

Este algoritmo se plantea de diferente manera, en vez de atacar al problema de buscar el camino de menor longitud entre un vértice inicial (v_s) y los demás, lo que hace es buscar directamente las longitudes de los caminos mínimos entre todos los pares de vértices, construyendo lo que se denomina como matriz de distancias mínimas.

- **FLOYD-WARSHALL**(G)

1. $v^{(0)} := G$
2. **for** $k := 1$ **to** n
3. Sea $v^{(k)} := (v_{ij}^{(k)})$ una nueva matriz $n \times n$.
4. **for** $i := 1$ **to** n
5. **for** $j := 1$ **to** n
6. //Comparamos si los caminos tienen menor longitud pasando por el vértice k.

if $v_{ik}^{(k-1)}.d + v_{kj}^{(k-1)}.d \leq v_{ij}^{(k-1)}.d$

7. $v_{ij}^{(k)}.d := v_{ik}^{(k-1)}.d + v_{kj}^{(k-1)}.d$
8. $v_{ij}^{(k)}.\pi := v_{kj}^{(k-1)}.\pi$
9. **else**
10. $v_{ij}^{(k)}.d := v_{ij}^{(k-1)}.d$
11. $v_{ij}^{(k)}.\pi := v_{ij}^{(k-1)}.\pi$
12. **return** $v^{(n)}$

$$v^{(0)} := \begin{pmatrix} 0\,|NIL & 3\mid 1 & 8\mid 1 & \infty|NIL & -4\mid 1 \\ \infty|NIL & 0\,|NIL & \infty|NIL & 1\mid 2 & 7\mid 2 \\ \infty|NIL & 4\mid 3 & 0\,|NIL & \infty|NIL & \infty|NIL \\ 2\mid 4 & \infty|NIL & -5\mid 4 & 0\,|NIL & \infty|NIL \\ \infty|NIL & \infty|NIL & \infty|NIL & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

$$v^{(1)} := \begin{pmatrix} 0\,|NIL & 3\mid 1 & 8\mid 1 & \infty|NIL & -4\mid 1 \\ \infty|NIL & 0\,|NIL & \infty|NIL & 1\mid 2 & 7\mid 2 \\ \infty|NIL & 4\mid 3 & 0\,|NIL & \infty|NIL & \infty|NIL \\ 2\mid 4 & 5\mid 1 & -5\mid 4 & 0\,|NIL & -2\mid 1 \\ \infty|NIL & \infty|NIL & \infty|NIL & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

$$v^{(2)} := \begin{pmatrix} 0\,|NIL & 3\mid 1 & 8\mid 1 & 4\mid 2 & -4\mid 1 \\ \infty|NIL & 0\,|NIL & \infty|NIL & 1\mid 2 & 7\mid 2 \\ \infty|NIL & 4\mid 3 & 0\,|NIL & 5\mid 2 & 11\mid 2 \\ 2\mid 4 & 5\mid 1 & -5\mid 4 & 0\,|NIL & -2\mid 1 \\ \infty|NIL & \infty|NIL & \infty|NIL & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

$$v^{(3)} := \begin{pmatrix} 0\,|NIL & 3\mid 1 & 8\mid 1 & 4\mid 2 & -4\mid 1 \\ \infty|NIL & 0\,|NIL & \infty|NIL & 1\mid 2 & 7\mid 2 \\ \infty|NIL & 4\mid 3 & 0\,|NIL & 5\mid 2 & 11\mid 2 \\ 2\mid 4 & -1\mid 3 & -5\mid 4 & 0\,|NIL & -2\mid 1 \\ \infty|NIL & \infty|NIL & \infty|NIL & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

$$v^{(4)} := \begin{pmatrix} 0\,|NIL & 3\mid 1 & -1\mid 4 & 4\mid 2 & -4\mid 1 \\ 3\mid 4 & 0\,|NIL & -4\mid 4 & 1\mid 2 & -1\mid 1 \\ 7\mid 4 & 4\mid 3 & 0\,|NIL & 5\mid 2 & 3\mid 1 \\ 2\mid 4 & -1\mid 3 & -5\mid 4 & 0\,|NIL & -2\mid 1 \\ 8\mid 4 & 5\mid 3 & 1\mid 4 & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

$$v^{(5)} := \begin{pmatrix} 0\,|NIL & 1\mid 3 & -3\mid 4 & 2\mid 5 & -4\mid 1 \\ 3\mid 4 & 0\,|NIL & -4\mid 4 & 1\mid 2 & -1\mid 1 \\ 7\mid 4 & 4\mid 3 & 0\,|NIL & 5\mid 2 & 3\mid 1 \\ 2\mid 4 & -1\mid 3 & -5\mid 4 & 0\,|NIL & -2\mid 1 \\ 8\mid 4 & 5\mid 3 & 1\mid 4 & 6\mid 5 & 0\,|NIL \end{pmatrix}$$

Figura 2.12: **Floyd y Warshall**

2.5.7. Johnson (1977): Complejidad del Algoritmo = $O(n \cdot (m + n \cdot log(n)))$

Este algoritmo une las ventajas de los algoritmos **Bellman-End-Ford** y **Dijkstra** para que sea más rápido que el algoritmo **Floyd y Warshall** en grafos poco densos.

1. Se añade un nodo v_q al grafo, conectado a cada uno de los nodos del grafo mediante un arco de peso cero.

2. Se utiliza el algoritmo de BELLMAN-FORD, empezando por el nuevo vértice v_q. Este determinará la longitud del camino mínimo desde v_q hasta v_i para cada vértice v_i. Si se detecta un circuito negativo, el algoritmo termina.

3. Para cada nodo $v_i \in V$ se usa el algoritmo de DIJKSTRA para determinar el camino más corto entre v_i y los otros nodos del grafo, usando el grafo con pesos modificados, los cuales, nos hemos asegurado que son todos positivos, y por tanto, nos aseguramos que el algoritmo de DIJKSTRA encuentra caminos mínimos óptimos.

- **JOHNSON**(G, w)

1. $G' := (G'.V, G'.A)$, donde:
   - $G'.V := G.V \cup \{v_q\}$
   - $G'.A := G.A \cup \{(v_q, v_i) : \forall v_i \in G.V\}$
   - $\forall v_i \in G.V \quad w(v_q, v_i) := 0$
2. **if** BELLMAN-FORD$(G', w, v_q) == FALSE$
3. **Print** ("Este Grafo tiene al menos un ciclo con longitud negativa.")
4. **else**
5. **for** $i = 1$ **to** n
6. crea $h(v_i) := v_{qi}.d$ calculado por el algoritmo de BELLMAN-FORD.
7. **for** cada arco $(v_i, v_j) \in G'.A$
8. $\widehat{w}(v_i, v_j) := w(v_i, v_j) + h(v_i) - h(v_j)$
9. crea $v := (v_{ij})$ una nueva matriz $n \times n$
10. **for** cada vértice $v_i \in G.V$
11. Ejecuta DIJKSTRA$(G, \widehat{w}, v_i)$ para calcular $v_{ij}.d \; \forall v_j \in G.V$
12. **for** cada vértice $v_j \in G.V$
13. $v_{ij}.d := v_{ij}.d - h(v_i) + h(v_j)$
14. **return** v

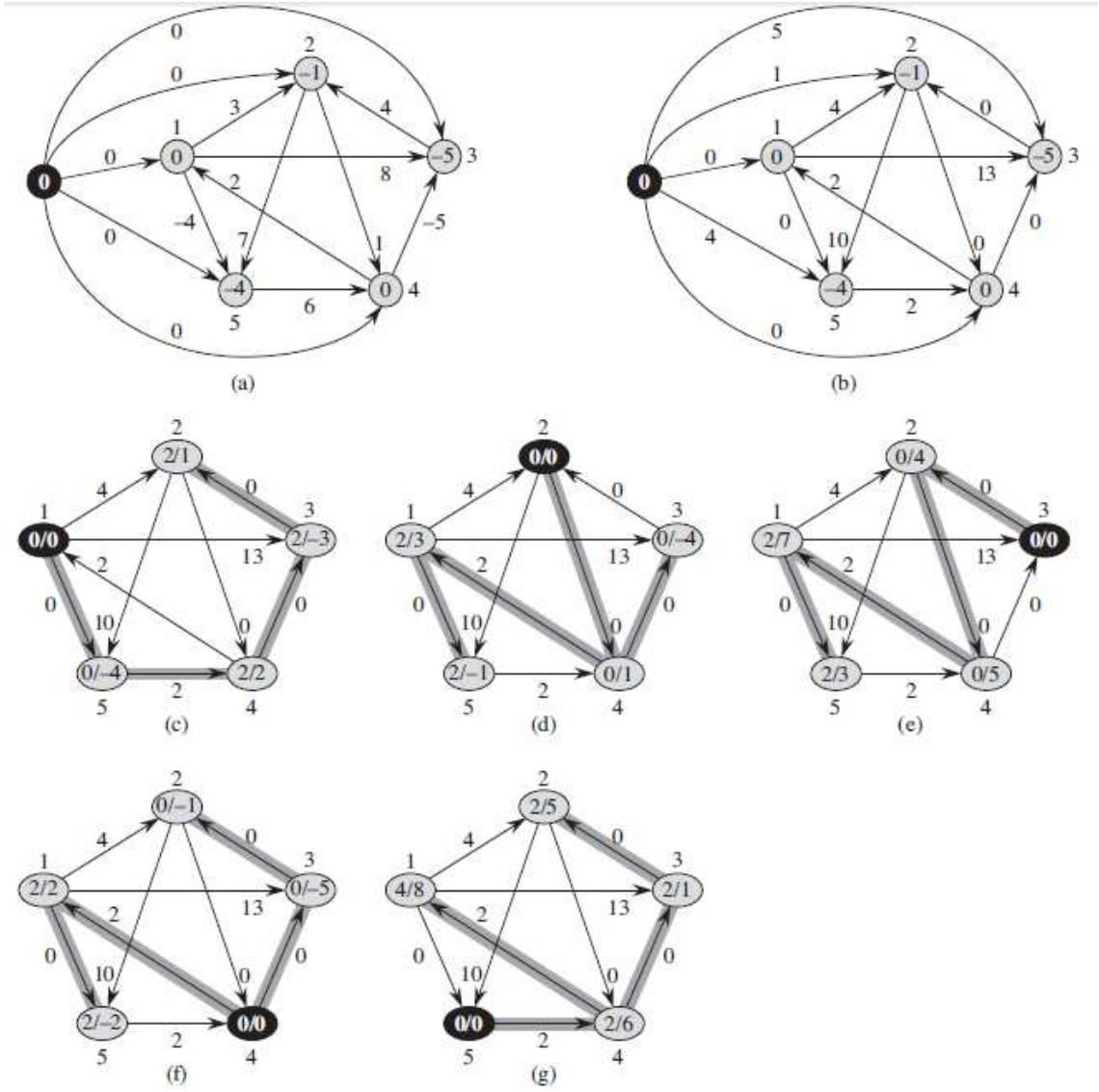


Figura 2.13: **Johnson**

-

2.6. Nuevo Algoritmo (mldc)

En este apartado, explicaremos un nuevo algoritmo para la determinación del circuito de menor longitud en un grafo dirigido fuertemente conexo y arcos de longitud arbitrarias.

2.6.1. La clave

Aquí introduciremos la idea para resolver el problema del MLDC. Definimos $\widehat{v_{ij}}.d$ como la distancia más corta entre el nodo v_i y el nodo v_j, cuyos vértices intermedios pertenecen al conjunto $\{v_1, ..., v_{i-1}\}$ cuando $i > j$, o pertenecen el conjunto $\{v_1, ..., v_{j-1}\}$ cuando $j > i$. En la matriz clásica de los caminos más cortos de dimensión n^2, $v_{ij}.d$ es la longitud del camino mínimo más corto desde el nodo v_i al nodo v_j cuyos vértices pertencen al conjunto $\{v_1, ..., v_n\}$. Posteriormente introduciremos un algoritmo para calcular las distancias $\widehat{v_{ij}}.d$ para todo $1 \leq i, j \leq n$. Obviamnete, $\widehat{v_{ij}}.d \geq v_{ij}.d$ para todos los $1 \leq i, j \leq n$, y, en particular, $\widehat{v_{nj}}.d = v_{nj}.d$ y $\widehat{v_{jn}}.d = v_{jn}.d$ para todo $1 \leq j \leq n$. El algoritmo para la determinación del circuito de menor longitud en un grafo dirigido está basado en las siguientes observaciones:

- **Lema 1:** Solo es necesario calcular las distancias $\widehat{v_{ij}}.d$ para todo $1 \leq i, j \leq n$ para determinar el circuito de menor longitud en un grafo dirigido G.
- **Demostración:** Supongamos que un MLDC existe en $G = (V, A)$ y sea $v_1 = v_s$, v_2, v_3, v_4, ..., $v_p = v_t$ la secuencia con $p \leq n$ nodos que define la secuencia de arcos (v_r, v_{r+1}) con $r \in \{1, ..., p-1\}$ en el MLDC. Por último, el MLDC se completa añadiendo el arco (v_t, v_s). Sea v_i el nodo con el mayor índice en el ciclo, por ejemplo, supongamos que sin pérdida de generalidad, que $v_i = v_s$. Claramente, el camino desde el nodo v_s hasta el v_t debe ser un camino mínimo desde que el MLDC es mínimo. Dado que cualquier nodo intermedio v_j en el camino mínimo desde el nodo v_s al v_t (incluido v_t) satisface $j < s$, la longitud de este camino más corto es precisamente $\widehat{v_{st}}.d$ y la longitud del ciclo dirigido es $\widehat{v_{st}}.d + w(v_t, v_s)$. □

  En otras palabras, una vez que las distancias $\widehat{v_{ij}}.d$ para todo $1 \leq i, j \leq n$ se calculan, el ciclo dirigido de longitud mínima se identifica como $\min_{v_j \in \Gamma_i^-} \{\widehat{v_{ij}}.d + w(v_j, v_i) : i > j\}$ para todo $1 < i \leq n$. Claramente, este proceso requiere un tiempo adicional de ejecución $O(m)$ una vez que la distancia $\widehat{v_{ij}}.d$ son conocidas, siempre que ambos cálculos no se realicen de forma simultánea.
- **Lema 2:** Sea T_i el árbol de raíz en el nodo v_i que contiene a cualquier nodo $v_j \in V$ con $j < i$ tal que $\widehat{v_{ij}}.d < +\infty$, entonces, el MLDC que sale y llega al nodo v_i y contiene exclusivamente los nodos en el conjunto $\{v_1, ..., v_{i-1}\}$ se determina por $\min_{v_j \in \Gamma_i^-} \{\widehat{v_{ij}}.d + w(v_j, v_i) : i > j\}$ en un grafo sin circuitos de longitud negativa.
- **Demostración:** Puesto que $G(V, A)$ no contiene circuitos de longitud negativa, el MLDC que empeza y termina en el nodo v_i es un ciclo dirigido elemental. Por hipótesis, este MLDC debe ir del nodo v_i a algún nodo v_h donde cualquier nodo intermedio

fallará en el conjunto $\{1, ..., i-1\}$ y regresará al nodo v_i usando el arco (v_h, v_i). Claramente, la longitud mínima del camino desde v_i hasta v_h es $\widehat{v_{ih}}.d$. Además, la longitud del circuito es $\widehat{v_{ih}}.d + w(v_h, v_i)$. Por lo tanto, $\min\limits_{v_j \in \Gamma_i^-} \{\widehat{v_{ij}}.d + w(v_j, v_i) : i > j\}$ determina el MLDC empezando y acabando en el nodo v_i con nodos intermedios en el conjunto $\{v_1, ..., v_{i-1}$. □

- **Teorema 1:** El MLDC en $G = (V, A)$ es determinado por $\min\limits_{2 \le i \le n} \{ \min\limits_{v_j \in \Gamma_i^-} \{\widehat{v_{ij}}.d + w(v_i, v_j) : i > j\}\}$ en un grafo sin circuitos de longitud negativa.
- **Demostración:** El resultado se obtiene a partir de los lemas $(1) - (2)$ y el hecho de que la MLDC debe ser el circuito de longitud mínima entre los circuitos determinados por los árboles T_i con ráices en el nodo v_i que contiene cualquier nodo $v_j \in V$ con $j < i$ de tal manera que $\widehat{v_{ij}}.d < +\infty$ junto con los arcos $(v_j,\ v_i) \in A$, para todo $i = 2, ..., n$. El nodo v_1 no es incluido, debido a que el circuito mínimo empezando y terminando en el nodo v_1 siempre contiene al menos un nodo v_i con $i > 1$ y, por lo tanto, este circuito se puede encontrar a partir de algún árbol T_i con $i > 1$. □

El teorema anterior nos informa de que para calcular el MLDC, tenemos que progresar de forma iterativa en el cálculo de MLDC empezando y terminando en el nodo v_i, junto con las distancias del camino más corto $\widehat{v_{ij}}.d$ para todo $1 \le j < i$, en el orden $i = 2, 3, ...n$. Sin embargo, con el fin de calcular correctamente $\widehat{v_{ij}}.d$ con $i > j$, hay que concatenar adecuadamente algunos caminos en los árboles T_k con $k < i$ para derivar el camino más corto desde el nodo v_i al nodo v_j. La razón es que si el MLDC es $\widehat{v_{ij}}.d + w(v_j, v_i)$ entonces, el camino más corto desde el nodo v_i al nodo v_j con los nodos intermedios en el conjunto $\{v_1, ..., v_{i-1}\}$ podría contener cualquier sub-camino de v_u a v_j con $u < j$, donde los nodos intermedios pertenecen al conjunto $\{v_1, ..., v_{j-1}\}$ y el sub-camino de v_i a v_u contiene nodos más pequeños que v_u. En este último caso, necesitamos $\widehat{v_{uj}}.d$ con $u < j$, donde los nodos intermedios están en $\{v_1, ..., v_{j-1}\}$. Por lo tanto, debemos calcular $\widehat{v_{ij}}.d$ cuando $i < j$ considerando los nodos intermedios en el conjunto $\{v_1, ..., v_{j-1}\}$. Las ecuaciones recurrentes y el algoritmo que calcula las distancias $\widehat{v_{ij}}.d$ y el MLDC se dan en la siguiente sección.

2.6.2. Ecuaciones de recurrencia para calcular las distancias parciales.

En esta sección, damos las ecuaciones de recurrencia para determinar $\widehat{v_{ij}}.d$ para todo $1 \le i,\ j \le n$ donde $\widehat{v_{ij}}.d$ es la longitud del camino más corto desde el nodo v_i al nodo v_j cuyos vértices intermedios caen en el conjunto $\{v_1, ..., v_{i-1}\}$, cuando $i > j$ y en el conjunto $\{v_1, ..., v_{j-1}\}$, cuando $i < j$. Con el fin de identificar esos caminos óptimos de los nodos, mantenemos una etiqueta vértice intermedio $\widehat{v_{ij}}.\pi$ asociada a cada camino desde v_i a v_j. $\widehat{v_{ij}}.\pi$ es el nodo v_i cuando el camino desde el nodo v_i al nodo v_j no tiene nodos intermedios, es decir, cuando el v_i es el predecesor del v_j. De lo contrario, $\widehat{v_{ij}}.\pi$ es el nodo con el mayor índice entre todos los nodos intermedios en ese camino (tenga en cuenta que v_i y v_j no son nodos intermedios).

A continuación, sea $\widehat{v_{ij}}^i.d$ y $\widehat{v_{ji}}^i.d$ las distancias $\widehat{v_{ij}}.d$ y $\widehat{v_{ji}}.d$, respectivamente, para un i fijado y para todo $1 \leq j < i$. Hemos añadido un superíndice en la notación para denotar la iteración. Tenga en cuenta que las distancias para $i = 1$ no tienen sentido ya que cualquier j debe ser menor que i y mayores o iguales que 1. En este caso, Sea $\widehat{v_{11}}^1.d = 0$ para facilitar la expresión de las siguientes ecuaciones recurrentes. Con el fin de obtener, las ecuaciones recurrentes, consideramos el caso genérico para un i fijo.

$$\widehat{v_{ij}}^i.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^+ : k \leq j} \{w(v_i, v_k) + \widehat{v_{kj}}^j.d\}, \min_{j<k<i} \{\widehat{v_{ik}}^i.d + \widehat{v_{kj}}^k.d\}\} \quad 1 \leq j \leq i \qquad (1)$$

$$\widehat{v_{ji}}^i.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^- : k \leq j} \{\widehat{v_{jk}}^j.d + w(v_k, v_i)\}, \min_{j<k<i} \{\widehat{v_{jk}}^k.d + \widehat{v_{ki}}^i\}\} \quad 1 \leq j \leq i \qquad (2)$$

- **Teorema 2:** Dado un grafo $G = (V, A)$ sin circuitos de longitud negativa, la longitud del camino más corto $\widehat{v_{ij}}.d$ y $\widehat{v_{ji}}.d$ son iguales a $\widehat{v_{ij}}^i.d$ y $\widehat{v_{ji}}^i.d$ $\forall i > j$ cuando las ecuaciones recurrentes (1)(2) se aplican en el orden $2, ..., i$ y j tomando los valores del $i - 1$ al 1 para un i fijado.

- **Demostración:** Claramente, la ecuación base es $\widehat{v_{11}}^1.d = 0$ y, por lo tanto, $\widehat{v_{11}}.d = \widehat{v_{11}}^1.d$. Del mismo modo, $\widehat{v_{22}}.d = \widehat{v_{22}}^2.d$, $\widehat{v_{21}}^2.d = \text{mín}\{+\infty, w(v_2, v_1) + \widehat{v_{11}}^1.d\}$ donde el arco (v_2, v_1) pertenece a G (siguiendo (1)) y $\widehat{v_{21}}^2.d = \text{mín}\{+\infty, \widehat{v_{11}}^1.d + w(v_1, v_2)\}$ donde el arco (v_1, v_2) pertenece a G (siguiendo (2)). Ahora aplicando el método de inducción, suponemos que $\widehat{v_{kj}}.d$ y $\widehat{v_{jk}}.d$ son igual a $\widehat{v_{kj}}^k.d$ y $\widehat{v_{jk}}^k.d$ para todo $k > j$ y para todos los k desde 1 a $i - 1$ y demostraremos el caso con $k = i$. Siguiendo el teorema, primero se calcula $\widehat{v_{i\ i-1}}^i.d$ como $\widehat{v_{i\ i-1}}^i.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^+ : k \leq i-1} \{w(v_i, vk) + \widehat{v_{k\ i-1}}^{i-1}.d\}\}$ ya que el tercer mín en la ecuación recurrente (1) es vacío. Claramente, esta ecuación recurrente establece que la longitud del camino más corto desde el nodo v_i al nodo v_{i-1} con nodos intermedios en el conjunto $\{v_1, ..., v_{i-1}\}$ es el mínimo de las longitudes $w(v_i, v_k) + \widehat{v_{k\ i-1}}^{i-1}.d$ para todos los arcos $(v_i, v_k) \in G$. La razón es que el camino óptimo desde el nodo v_i al nodo v_{i-1} es un camino que empieza con el arco (v_i, v_k), seguido por un un camino desde el nodo v_k al nodo v_{i-1}. Desde que $k \leq i - 1$, la distancia es $\widehat{v_{k\ i-1}}^{i-1}.d$ y, por lo tanto, $\widehat{v_{i\ i-1}}.d$ es $\widehat{v_{i\ i-1}}^i.d$ el cual fue calculado correctamente por (1). Supongamos ahora $j = i - 2$, la ecuación recurrente (1) se convierte ahora en $\widehat{v_{i\ i-2}}^i.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^+ : k \leq i-2} \{w(v_i, v_k) + \widehat{v_{k\ i-2}}^{i-2}.d\}, \widehat{v_{i\ i-1}}^i.d + \widehat{v_{i-1\ i-2}}^{i-1}.d\}$. Claramente, la longitud del camino mínimo desde el nodo v_i al nodo v_{i-2} con nodos intermedios en el conjunto $\{v_1, ..., v_{i-1}\}$ es un camino donde todos los nodos intermedios son más pequeños que v_{i-2} o un camino donde el nodo v_{i-1} es un nodo intermedio. En el primer caso, el segundo mín de la expresión de arriba determina la longitud del camino más corto de los caminos que no pasan a través del nodo v_{i-1} y la expresión $\widehat{v_{i\ i-1}}^i.d + \widehat{v_{i-1\ i-2}}^{i-1}.d$ considera la posibilidad de un camino con el nodo v_{i-1} como nodo intermedio. Puesto que todas las distancias consideradas en la expresión son óptimas debido a que $\widehat{v_{i\ i-1}}^i.d$ se calculó anteriormente, la distancia $\widehat{v_{i\ i-2}}^i.d$ es calculada correctamente y determina $\widehat{v_{i\ i-2}}.d$. Siguiendo los mismos argumentos, se

demuestra que $\widehat{v_{ij}}^{i}.d$ está calculada correctamente para un i fijo y para todo j en el orden $i-1$ a 1 que corresponde con al camino más corto desde el nodo v_i al nodo v_j con los nodos intermedios en el conjunto $\{v_1, ..., v_{i-1}\}$, es decir, $\widehat{v_{ij}}.d$. Del mismo modo, se puede demostrar aplicando las ecuaciones recurrentes (2) para todo i desde 2 hasta n, y para un i fijado, para todo j en el orden de $i-1$ a 1, las distancias $\widehat{v_{ji}}^{i}.d$ son $\widehat{v_{ji}}.d$. □

- **Corolario 1:** Dado un grafo G sin ciclos dirigidos de longitud negativa y con al menos un ciclo dirigido, el MLDC en G se determina aplicando las ecuaciones recurrentes (1)(2) en el orden $2, ..., n$

- **Demostración:** Por el teorema 1, podemos suponer que la MLDC está determinada por el camino más corto desde el nodo v_i al nodo v_j con los nodos intermedios en el conjunto $\{v_{i-1}, ..., v_1\}$, más el arco (v_j, v_i) siendo $i > j$. Por otra parte, la longitud de la MLDC se identifica por $\widehat{v_{ij}}.d + w(v_i, v_j)$. Por el teorema 2, estamos seguros que $\widehat{v_{ij}}.d$ es $\widehat{v_{ij}}^{i}.d$ y $\widehat{v_{ji}}.d$ es $\widehat{v_{ji}}^{i}.d$ cuando $i > j$ y el MLDC que sale y llega al nodo v_i y que contiene exclusivamente nodos en el conjunto $\{v_1, ..., v_{i-1}\}$ los cuales se pueden determinar como $\min\limits_{1 \le j < i} \{\widehat{v_{ij}}.d + \widehat{v_{ji}}.d\}$. Tenga en cuenta que si los caminos desde v_i hasta v_j y desde v_j a v_i tienen nodos comunes, entonces, esta situación implica la existencia de uno o varios ciclos dirigidos que tienen longitud menor o igual a $\widehat{v_{ij}}.d + \widehat{v_{ji}}.d$ ya que G no tiene ciclos dirigidos de longitud negativos. Por lo tanto, la MLDC se determinará como $\min\limits_{2 \le i \le n} \min\limits_{1 \le j < i} \{\widehat{v_{ij}}^{i}.d + \widehat{v_{ji}}^{i}.d\}$ siguiendo el teorema 1. □

Por el corolario 1, ya sabemos cómo identificar la longitud del MLDC usando las ecuaciones recurrentes (1) y (2). Ahora, necesitamos identificar los nodos en el MLDC. Supongamos que el MLDC es el camino del nodo v_i al v_j y el camino del nodo v_j al nodo v_i, donde v_i es el nodo con índice mayor en el MLDC. Debemos utilizar la matriz intermedia $\widehat{v}.\pi$ para identificar el ciclo dirigido, donde $\widehat{v_{ij}}.\pi$ es el nodo de índice mayor entre los nodos intermedios en el camino desde v_i hasta v_j. Usamos el siguiente procedimiento recursivo llamado PrintPath:

- **Procedure PrintPath (Input i, j)**
  1. **If** $(\widehat{v_{ij}}.\pi == i)$ **Then**
  2. **Print** "$i \rightarrow$ "
  3. **Else**
  4. **PrintPath** (i, $\widehat{v_{ij}}.\pi$);
  5. **PrintPath** ($\widehat{v_{ij}}.\pi$, j);

En este caso, el procedimiento anterior se llama inicialmente como **PrintPath** (i, j) y después por **PrintPath** (j, i). Por último se imprime el nodo j en la pantalla para completar la secuencia de nodos del MLDC. La complejidad general de las dos llamadas al procedimiento PrintPath es $O(i)$. Esto ocurre en el peor de los casos, cuando todos los nodos $\{v_1, ..., v_i\}$ pertenecen a MLDC y el nodo intermedio en cada sub-camino está justo en el medio del sub-camino, en este caso, tenemos la ecuación de recurrencia clásica $T(i) = 2T(\frac{i}{2}) + O(1)$ donde $T(i)$ es el tiempo empleado en

imprimir un camino con i nodos.
Una implementación directa de las ecuaciones recurrentes (1) y (2) implica un algoritmo de complejidad $O(n^3)$ que en el peor de los casos realiza $2nm + \frac{n^3}{3} + n^2 - \frac{4}{3}$ adiciones/comparaciones. Con el fin de mejorar el rendimiento del algoritmo, observamos las siguientes propiedades.

2.6.3. Algunas propiedades para mejorar la aplicación de las ecuaciones recurrentes (1) y (2).

En esta sección, se comentan las propiedades que exhiben los caminos óptimos que pueden ser aprovechadas para mejorar el rendimiento del algoritmo usando las ecuaciones de recurrencia (1) y (2). Estas son las siguientes:

A. Propiedad: Nodos intermedio con mayor índice. Para un i fijo, si la distancia $\widehat{v_{ij}}^i.d$ se hace igual a $d\widehat{v_{ik'}}^i.d + \widehat{v_{k'j'}}^{k'}.d$ entonces, para el resto de nodos $j < j'$, el nodo $v_{j'}$ no es considerado en $\min\limits_{j<k<i}\{\widehat{v_{ik}}^i.d + \widehat{v_{kj}}^k.d\}$ ya que $\widehat{v_{ij'}}^i.d + \widehat{v_{j'j}}^{j'}.d = \widehat{v_{ik'}}^i.d + \widehat{v_{k'j'}}^{k'}.d + \widehat{v_{j'j}}^{j'}.d = \widehat{v_{ik'}}^i.d + \widehat{v_{k'j}}^{k'}.d$. En otras palabras, los nodos $v_{k'}$ y $v_{j'}$ son nodos intermedios en el camino desde v_i a v_j. Con $k' > j'$, la distacia $\widehat{v_{k'j}}^{k'}.d$ es igual a $\widehat{v_{k'j'}}^{k'}.d + \widehat{v_{j'j}}^{j'}.d$. Así, $\widehat{v_{ij}}^i.d$ solo se calcula una vez en el algoritmo desde el nodo intermedio con mayor índice en el camino desde v_i a v_j. Por lo tanto, en una iteración i fija, se utiliza un vector dinámico VR^i (VR en el algoritmo) que inicialmente está vacío. A continuación, para todo $j := i - 1$ hasta el 1 es añadido a VR^i si y solo si $\widehat{v_{ij}}^i.d < \infty$ y $\widehat{v_{ij}}.\pi < j$. Es decir, se añaden los nodos donde se satisfaga que el mayor nodo intermedio en el camino desde el nodo v_i hasta el nodo v_j es menor que v_j. Hay que tener en cuenta, que cuando el camino desde el nodo v_i hasta el nodo v_j es el arco (v_i, v_j), el nodo v_j no se añade al VR^i porque $\widehat{v_{ij}}.\pi = i > j$. En este caso, cuando usamos el conjunto VR^i, es necesario extender el primer mínimo de las ecuaciones recurrentes (1) a cualquier arco (v_i, v_k) con $k < i$ por tanto $k \leq j$. La razón es que un camino desde v_i a v_j que puede comenzar en el arco (v_i, v_k) con $k > j$ donde todos los nodos intermedios en el sub-camino desde v_k a v_j son menores que v_j. En la ecuación recurrente (1) esta posibilidad es completada en el segundo mín de (1), pero desde que ahora el nodo v_k no es añadido a VR^i, consideraremos esta posibilidad en el primer mín de (1). Argumentos similares se sostienen para el camino desde el nodo v_j al nodo v_i, manteniendo un vector dinámico VL^i (VL en el algoritmo) que contiene cualquier nodo $j < i$ donde $\widehat{v_{ji}}\pi < i$. Para hacer esto, aplicando la ecuación recurrente (1) y (2) y separando el mínimo que aparece en estas ecuaciones de la siguiente manera: en primer lugar se calculan las ecuaciones recurrentes (3) y (4) (el primer mín de (1) y (2)) para todo $j < i$.

$$\widehat{v_{ij}}^{i}.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^+ : k \leq i} \{w(v_i, v_k) + \widehat{v_{kj}}^{j}.d\}\} \quad 1 \leq j \leq i \quad (3)$$

$$\widehat{v_{ji}}^{i}.d = \text{mín}\{+\infty, \min_{v_k \in \Gamma_i^- : k \leq j} \{\widehat{v_{jk}}^{j}.d + w(v_k, v_i)\}\} \quad 1 \leq j \leq i \quad (4)$$

Una vez aplicadas (3) y (4), hacemos $VR^i = \emptyset$, $VL^i = \emptyset$ y ahora, sabemos que las etiquetas de distancia $\widehat{v_{i\ i-1}}^{i}.d$ y $\widehat{v_{i-1\ i}}^{i}.d$ son óptimas. A continuación, las ecuaciones recurrentes (5) y (6) se llevan a cabo para todo j en el orden $i-1$ a 1 (en particular, para $j = i-1$ las ecuaciones recurrentes (5) y (6) no son aplicadas desde $VR^i = \emptyset$ y $VL^i = \emptyset$).

$$\widehat{v_{ij}}^{i}.d = \text{mín}\{\widehat{v_{ij}}^{i}.d, \min_{k \in VR^i} \{\widehat{v_{ik}}^{i}.d + \widehat{v_{kj}}^{k}.d\}\} \quad 1 \leq j \leq i \quad (5)$$

$$\widehat{v_{ji}}^{i}.d = \text{mín}\{\widehat{v_{ji}}^{i}.d, \min_{k \in VL^i} \{\widehat{v_{jk}}^{k}.d + \widehat{v_{ki}}^{i}.d\}\} \quad 1 \leq j \leq i \quad (6)$$

Siguiendo con la propiedad **A)**, si $\widehat{v_{ij}}.\pi < j$ y/o $\widehat{v_{ji}}.\pi < j$ donde, el nodo v_j es añadido a VR^i y/o VL^i. Por lo tanto, las ecuaciones de recurrencia (5) y (6) son aplicadas a cualquier nodo v_{j-1} hasta v_1 teniendo en cuenta la posibilidad de que el nodo v_j sea el de mayor índice de los nodos intermedios.

B. Propiedad de las etiquetas. Para un i fijo, una vez que la longitud $\widehat{v_{ij}}^{i}.d$ se calcula, sabemos que la longitud de cualquier nodo v_k en el camino desde v_i hasta v_j debe ser óptima. En este caso, podemos calcular directamente las distancias $\widehat{v_{ik}}^{i}.d$ para todo v_k en el camino desde v_i hasta v_j y considerar que estos nodos tienen una etiqueta correcta. Por eso, podríamos utilizar un procedimiento recursivo como *PrintPathProcedure* para obtener las etiquetas de estos nodos en el camino. En este caso, tendríamos un tiempo de ejecución $O(i)$ en el peor de los casos en esta operación para un nodo $j < i$ y, en general, un esfuerzo computacional $O(i^2)$ en la etapa i. En lugar de hacer esto, se observa que la longitud $\widehat{v_{ik}}^{i}.d$ con k siendo $\widehat{v_{ij}}.\pi$ está correctamente calculada. Nótese que si $\widehat{v_{ij}}.\pi > j$ el nodo $\widehat{v_{ij}}.\pi$ ya ha sido calculado por (3) y (5). Sin embargo, si $\widehat{v_{ij}}.\pi < j$ entonces la distancia $\widehat{v_{ik}}^{i}.d$ esta correctamente calculada por (3) ya que cualquier nodo intermedio en el camino desde v_i hasta v_k tiene un índice inferior a v_k. Por lo tanto, la ecuación de recurrencia (5) no debe ser aplicada para el nodo $\widehat{v_{ij}}.\pi$ cuando sea menor que j. Utilizaremos un vector (*Boolean*) de etiquetas correctas LR^i (LR en el algoritmo) que toma el valor 1 en la posición k siempre que la longitud $\widehat{v_{ik}}^{i}.d$ sea conocida ser óptima para la etapa i. De lo contrario, el valor es 0. Del mismo modo, se utiliza un vector de etiquetas correctas LL^i (LL en el algoritmo) que toma el valor 1 en la posición k siempre que la longitud $\widehat{v_{ki}}^{i}.d$ es conocida ser óptima para la etapa i.

Finalmente, el algoritmo implementando las ecuaciones (3)−(6) y determinando el circuito de longitud mínima se muestra en el pseudo-código MLDC.

- **MLDC**(G)

1. // base de la ecuación recursiva:
   $\widehat{v_{11}}.d := 0$;
   $\widehat{v_{11}}.\pi := -1$; // $\widehat{v_{ij}}.\pi$ será -1 si no existe el arco (v_i, v_j)
   $mldc := +\infty$;
2. **For** $i = 2$ **to** n **do**:
3. $\quad \widehat{v_{ii}}.d := 0$;
   $\quad \widehat{v_{ii}}.\pi := -1$;
   $\quad VR := \emptyset$;
   $\quad VL := \emptyset$;
   $\quad LR[i] := 0$;
   $\quad LL[i] := 0$;
4. $\quad$ **For** $j = i - 1$ **to** 1 **do** // Nodo v_j para la ecuación recursiva (3) y (4)
5. $\quad\quad \widehat{v_{ij}}.d := \infty$;
   $\quad\quad \widehat{v_{ij}}.\pi := -1$;
   $\quad\quad LR[j] := 0$;
6. $\quad\quad$ **For** all $v_k \in \Gamma^+_{v_i}$ con $k < i$ **do**
7. $\quad\quad\quad$ **If** $(w(v_i, v_k) + \widehat{v_{kj}}.d < \widehat{v_{ij}}.d)$ **Then**
8. $\quad\quad\quad\quad \widehat{v_{ij}}.d := w(v_i, v_k) + \widehat{v_{kj}}.d$;
9. $\quad\quad\quad\quad$ **If** $k \neq j$ **Then**
   $\quad\quad\quad\quad\quad \widehat{v_{ij}}.\pi := max\{k, \widehat{v_{kj}}.\pi\}$;
10. $\quad\quad\quad\quad$ **Else**
    $\quad\quad\quad\quad\quad \widehat{v_{ij}}.\pi := i$;
11. $\quad\quad \widehat{v_{ji}}.d = +\infty$;
    $\quad\quad \widehat{v_{ji}}.\pi := -1$;
    $\quad\quad LL[j] := 0$;
12. $\quad\quad$ **For** all $v_k \in \Gamma^-_i$ con $k < i$ **do**
13. $\quad\quad\quad$ **If** $(\widehat{v_{jk}}.d + w(v_k, v_i) < \widehat{v_{ji}}.d)$ **Then**
14. $\quad\quad\quad\quad \widehat{v_{ji}}.d := \widehat{v_{jk}}.d + w(v_k, v_i)$;
    $\quad\quad\quad\quad$ **If** $v_{jk}.\pi \neq j$ **Then**
    $\quad\quad\quad\quad\quad \widehat{v_{ji}}.\pi := max\{k, \widehat{v_{jk}}.\pi\}$
    $\quad\quad\quad\quad$ **else**
    $\quad\quad\quad\quad\quad \widehat{v_{ji}}.\pi := k$
15. $\quad$ **For** $j = i - 1$ **to** 1 **do** // Nodo v_j para la ecuación recursiva (5)
16. $\quad$ **If** $(LR[j] == 0)$ **Then** // Solo donde v_j no está etiquetado correctamente.
17. $\quad\quad$ **For** all $k \in VR$ **do**
18. $\quad\quad\quad$ **If** $\widehat{v_{ik}}.d + \widehat{v_{kj}}.d < \widehat{v_{ij}}.d)$ **Then**
    $\quad\quad\quad\quad \widehat{v_{ij}}.d := \widehat{v_{ik}}.d + \widehat{v_{kj}}.d$;
    $\quad\quad\quad\quad \widehat{v_{ij}}.\pi = k$;
19. $\quad\quad$ **If** $\widehat{v_{ij}}.d < +\infty)$ **and** $(\widehat{v_{ij}}.\pi < j))$ **Then**
    $\quad\quad\quad$ Añade j a VR;
20. $\quad\quad LR[j] := 1$;
    $\quad\quad LR[\widehat{v_{ij}}.\pi] := 1$;
21. $\quad$ **For** $j = i - 1$ **to** 1 **do** // Nodo v_j para la ecuación recursiva (6)

22. **If** $LL[j] == 0$ **Then** // Solo donde v_j no está etiquitado correctamente.
23. **For** all $k \in VL$ **do**
24. **If** $\widehat{v_{jk}}.d + \widehat{v_{ki}}.d < \widehat{v_{ji}}.d$ **Then**
$\widehat{v_{ji}}.d := \widehat{v_{kj}}.d + \widehat{v_{ki}}.d$;
$\widehat{v_{ji}}.\pi := k$;
25. **If** $\widehat{v_{ji}}.d < +\infty$) **and** ($\widehat{v_{ji}}.\pi < j$)) **Then**
Añade j a VL;
26. $LL[j] := 1$;
$LR[\widehat{v_{ji}}.\pi] := 1$;
27. **If** $\widehat{v_{ij}}.d + \widehat{v_{ji}}.d < mldc$) **Then**
$mldc := \widehat{v_{ij}}.d + \widehat{v_{ji}}.d$;
$y := j$;
$x := i$;
28. **Output** la longitud del MLDC es $mldc$ y este es el camino desde v_x hasta v_y y desde v_y hasta v_x identificado por el *PrintPathprocedure* donde quiera que $0 \leq mldc < +\infty$

En el caso de que G no contenga ningún circuito de longitud negativa, el ciclo dirigido de longitud mínima es identificado en la línea (28) del algoritmo consistiendo en el camino desde x hasta y más el camino desde y hasta x identificado por PrintPath(x, y), PrintPath(y, x) y el nodo. En cualquier otro caso, G no tiene un ciclo dirigido o este tiene un circuito de longitud negativa.

Observamos que en el caso de que los arcos tengan longitud no negativa, podemos acelerar el algoritmo solamente aplicando las ecuaciones de recurrencia para las celdas y los arcos con valores inferiores al mldc (lineas 19 y 27).

- **Teorema 2:** El circuito de longitud mínima en un grafo $G = (V, A)$ se calcula mediante el algoritmo propuesto en un tiempo $O(n^3)$.
- **Demostración:** El cálculo de las líneas (3) - (14) requiere un tiempo $O(nm)$. Claramente, el tiempo de ejecución del algoritmo propuesto se determina por la ejecución de los dos bucles anidados triples de las líneas de $\{(2), (15), (17)\}$ y $\{(2), (21), (23)\}$. En otras palabras, el tiempo es

$$\sum_{i=2}^{n} \sum_{j=i-2:LR[j]=0}^{1} \sum_{k \in VR'} O(1) = \sum_{i=2}^{n} \sum_{j=i-2:LL[j]=0}^{1} \sum_{k \in VL'} O(1).$$

En el peor de los casos (con baja probabilidad), $LR[j]$ ($LL[j]$) es 0 cada vez que un nodo v_j con $j < i$ es procesado y el conjunto $VR(VL)$ contiene los nodos desde v_{i-1} hasta v_{j+1}. En este último caso, se obtiene que la suma anterior implica una cota igual a

$$O(2\sum_{i=2}^{n} \sum_{j=i-2:k\in\{i-1,+1\}}^{1} \sum 1) = O(2\sum_{i=2}^{n} \sum_{j=1}^{i-2} j) = O(\frac{n}{3}(n^2 - 3n + 2)) = O(n^3).$$

Por último, la línea (28) para calcular el MLDC requiere un tiempo $O(n)$ para identificar el camino más corto desde v_x hasta v_y y desde v_y hasta v_x utilizando el procedimiento $PrintPath$. Evidentemente, la cota superior del tiempo de ejecución del algoritmo es $O(n^3)$. □

El espacio utilizado por el algoritmo es $O(n^2)$ y se determina por las n^2 entradas $\widehat{v_{ij}}.d$. A fin de reducir el espacio utilizado por el algoritmo, podemos hacer los siguientes cambios. En primer lugar, calcular los componentes fuertemente conexas de $G = (V, A)$ utilizando el algoritmo dado de **Tarjan (1972)** con un tiempo $O(n+m)$. En segundo lugar, ejecutar el algoritmo propuesto en el grafo inducido por los nodos de cada componente fuertemente conexa para cada uno de ellos. En el peor de los casos, es decir, cuando G está conectado firmemente, la complejidad temporal y en el espacio sigue siendo el mismo la misma. De lo contrario, la memoria utilizada es inferior.

2.6.4. Un ejemplo.

En esta sección, se muestran en la **Figura 2.16**, la ejecución del algoritmo MLDC para calcular las matrices de distancia y predecesor del gráfo de la **Figura 2.15**. En la **Figura 2.16**, la secuencia de las matrices $\widehat{v}^i.d$ y $\widehat{v}^i.\pi \forall i$ en el orden 1, 2, 3 , 4, 5, 6. Para un i fijado, la fila y columna negra continen las celdas calculadas por ecuaciones recurrentes $(3) - (6)$. La intención es observar claramente que el algoritmo propuesto es un algoritmo de asignación de etiquetas ya que cada celda de la matriz se calcula solamente una vez. En cada iteración, el conjunto de sucesores (en una fila) y predecesores (en una columna) de los nodos que son inferiores al nodo v_i se muestran. Los valores de color rojo en las celdas negras indican que el valor correspondiente se calculó por las ecuaciones recurrentes $(5) - (6)$. Una vez que las ecuaciones recurrentes $(3) - (6)$ se llevan a cabo para cada valor de i, el algoritmo comprueba la existencia de un mejor MLDC comparando mldc (el valor actual de MLDC) con $\widehat{v_{ij}}.d + \widehat{v_{ji}}.d \; \forall j < i$.

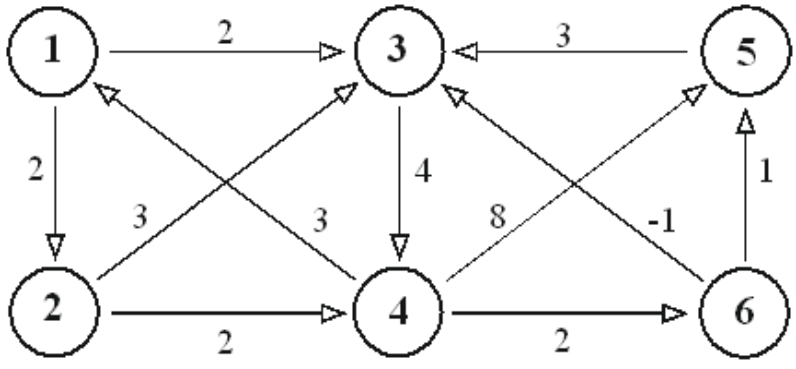


Figura 2.14: Ejemplo de un grafo con longitudes en los arcos.

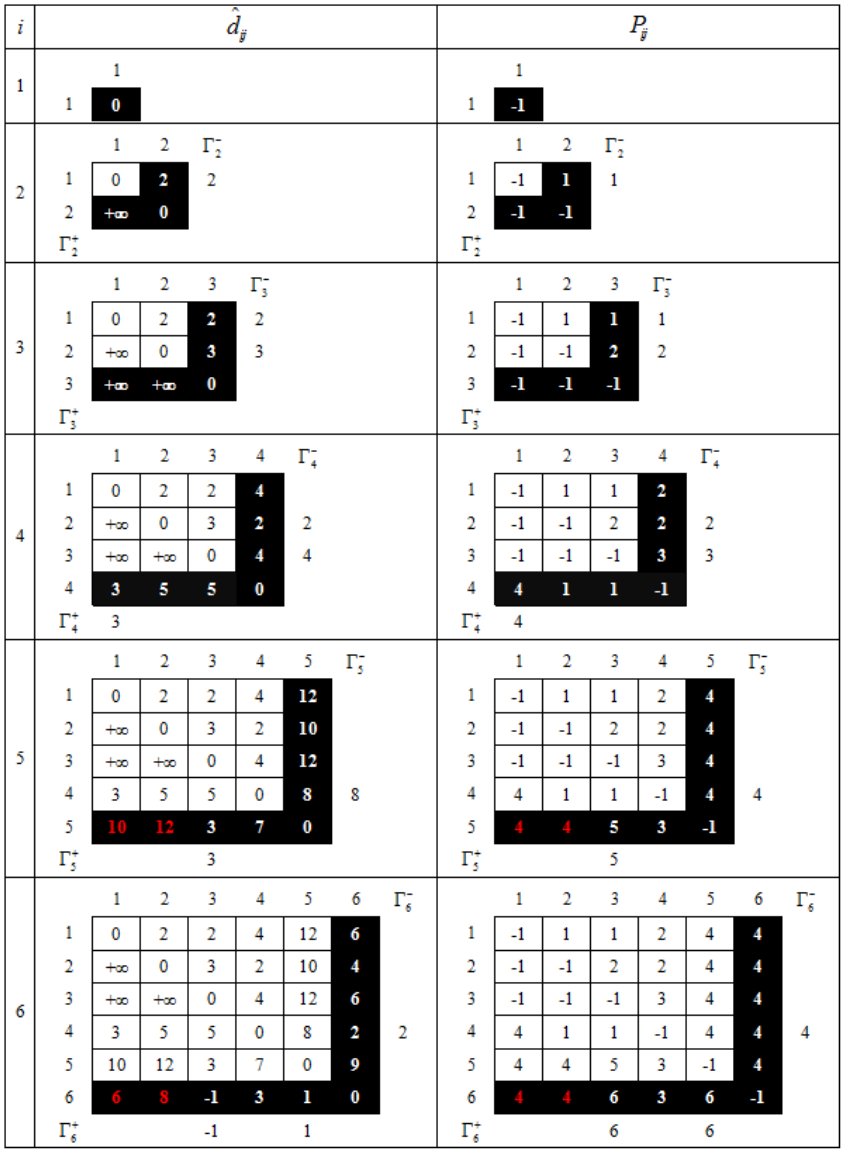


Figura 2.15: Ilustración de la ejecución del algoritmo del MLDC.

La primera iteración corresponde con la ecuación de la base. A continuación, sólo las ecuaciones recurrentes (3) − (4) se llevan a cabo cuando i es 2. Nótese que aquí se comprueba el primer ciclo como $\widehat{v_{21}}.d + \widehat{v_{12}}.d$, pero el valor es infinito significando que no hay circuito que contiene sólo los nodos en el conjunto $\{v_1, v_2\}$. La iteración con i siendo 3 comienza aplicando (3) − (4). Téngase en cuenta que el conjunto de los sucesores del nodo v_3 que son inferiores a v_3 está vacío y, por lo tanto, las ecuaciones recurrentes (3) y (5) no se hacen. Una vez que se realiza la ecuación de recurrencia (4), ahora que $\widehat{v_{23}}.d$ es óptima, pero el nodo v_2 no se añade a la $V.L$ ya que $\widehat{v_{23}}.\pi = 2$. La misma situación se plantea para $\widehat{v_{13}}.d$. Es decir, los conjuntos de

$V.R$ y $V.L$ están vacíos en esta iteración. Ahora, mldc siendo infinito se compara con $\widehat{v_{32}}.d + \widehat{v_{23}}.d$ y $\widehat{v_{31}}.d + \widehat{v_{13}}.d$. Pero, ambas sumas son infinitos y, por lo tanto, el algoritmo todavía no ha encontrado ningún circuito, en este caso conteniendo los nodos en el conjunto $\{v_1, v_2, v_3\}$. Una vez que las ecuaciones de recurrencia (3) y (4) se llevan a cabo en la iteración $i = 4$, sabemos que $\widehat{v_{43}}.d$ y $\widehat{v_{34}}.d$ son óptimas. Los caminos correspondientes son $4 \to 1 \to 3$ y $3 \to 4$ en este caso el nodo v_3 ($\widehat{v_{43}}.\pi = 1 < 3$) se añade a $V.R$ y $V.L$ es vacío ($\widehat{v_{34}}.\pi = 3$). Pero, $LR[1]$, $LR[3]$ y $LL[3]$ toman valor 1. Por lo tanto, la ecuación de recurrencia (5) se realiza para el nodo v_2, pero no la ecuación de recurrencia (6) ya que $V.L$ es vacío. El valor de $\widehat{v_{42}}.d$ no cambia, el nodo v_2 es añadido a $V.R$ ya que $\widehat{v_{42}}.\pi = 1 < 2$. $V.L$ es vacío. Ahora, la ecuación de recurrencia (5) − (6) no se realiza para el nodo v_1, ya que, $L.R[1]$ es 1 y $V.L$ es vacío. Ahora, el actual mldc se compara con $\widehat{v_{43}}.d + \widehat{v_{34}}.d = 9$, $\widehat{v_{42}}.d + \widehat{v_{24}}.d = 7$ y $\widehat{v_{41}}.d + \widehat{v_{14}}.d = 7$. En este caso, mldc toma el valor 7 siendo $y = 2$ y $x = 4$ que se corresponden con el circuito $4 \to 1 \to 2$ y $2 \to 4$. Una vez, las ecuaciones de recurrencia (3) y (4) están hechas para $i = 5$, los valores $\widehat{v_{54}}.d$ y $\widehat{v_{45}}.d$ son óptimos. Los caminos óptimos son $5 \to 3 \to 4$ y $4 \to 5$ con $\widehat{v_{54}}.\pi = 3 < 4$ y $\widehat{v_{45}}.\pi = 4$. Por lo tanto, $V.R = \{4\}$, $V.L$ está vacío y $LR[4]$ y $LL[4]$ toman valor 1. Por lo tanto, el siguiente nodo considerado es v_3. En este caso, la distancia $\widehat{v_{53}}.d$ no cambia, siendo el camino $5 \to 3$ con $\widehat{v_{53}}.\pi = 5$ y $V.R$ no cambia. En este caso, $\widehat{v_{35}}.\pi = 4 > 3$ y $V.L$ sigue vacío. Ahora, el nodo v_2 es considerado. En este caso, la distancia $\widehat{v_{52}}.d$ es igual a $\widehat{v_{54}}.d + \widehat{v_{42}}.d = 12$ (valor de rojo en la celda) siendo el camino $5 \to 3 \to 4 \to 1 \to 2$ con $\widehat{v_{52}}.\pi = 4$. $\widehat{v_{25}}.d$ no es examinada ya que $V.L$ es vacío. En este momento, $LR[2]$ y $LL[2]$ toman el valor 1. Ahora, por lo tanto, la ecuación de recurrencia (5) se lleva a cabo para el nodo v_1 y (6) no se realiza ya que $V.L$ es vacío. El valor $\widehat{v_{51}}.d$ se calcula como $\widehat{v_{54}}.d + \widehat{v_{41}}.d = 10$ (valor de rojo en la celda) siendo el camino $5 \to 3 \to 4 \to 1$ con $\widehat{v_{51}}.\pi = 4$. A continuación, el $mldc = 7$ se compara con $\widehat{v_{54}}.d + \widehat{v_{45}}.d = 15$, $\widehat{v_{53}}.d + \widehat{v_{35}}.d = 15$, $\widehat{v_{52}} + \widehat{v_{25}}.d = 22$ y $\widehat{v_{51}}.d + \widehat{v_{15}}.d = 22$. En este caso, mldc no cambia. Finalmente comienza la última iteración con $i = 6$. Una vez que se aplican (3) y (4), los valores $\widehat{v_{65}}.d = 1$ y $\widehat{v_{56}}.d = 9$ son óptimos. Los caminos $6 \to 5$ y $5 \to 3 \to 4 \to 6$ son óptimos con $\widehat{v_{65}}.\pi = 6$ y $\widehat{v_{56}} = 4$. En este caso, $V.R$ es vacío, $V.L = \{5\}$ y $L.R[5]$, $LL[4]$, $LL[5]$ son 1. Por lo tanto, la ecuaciones de recurrencia (5) se aplican al nodo v_4. Sin embargo, $\widehat{v_{64}}.d = 1$ no cambia, aunque nodo v_4 se añade a $V.R$ ya que $\widehat{v_{64}}.\pi = 3$. También, $LR[3]$ y $LR[4]$ toman el valor 1. A continuación, $V.R = \{4\}$ y $V.L = \{5\}$ y (6) se aplica para el nodo v_3. La distancia $\widehat{v_{36}}.d$ no cambia con $\widehat{v_{36}}.\pi = 4 > 3$. Por lo tanto, (5) y (6) se aplican para el nodo v_2 con $V.R = \{4\}$ y $VL = \{5\}$. La distancia $\widehat{v_{62}}.d$ es igual a $\widehat{v_{64}}.d + \widehat{v_{42}}.d = 8$ siendo el camino $6 \to 3 \to 4 \to 1 \to 2$ con $\widehat{v_{62}}.\pi = 4$ y $\widehat{v_{26}}.d = 4$ ($2 \to 4 \to 6$) con $\widehat{v_{26}}.\pi = 4$ no cambia. En este caso, $V.R = \{4\}$, $V.L = \{5\}$, $LR[2]$, $LL[2]$ son 1. Por lo tanto, las ecuaciones de recurrencia (5) y (6) se realizan para el nodo v_1, pero $\widehat{v_{16}}.d = 6$ no cambia. La longitud $\widehat{v_{61}}.d$ es igual a $\widehat{v_{64}}.d + \widehat{v_{41}}.d = 6$ siendo el camino $6 \to 3 \to 4 \to 1$ con $\widehat{v_{62}}.\pi = 4$. Por último, el mldc siendo 7 se compara con $\widehat{v_{65}}.d + \widehat{v_{56}}.d = 10$, $\widehat{v_{64}}.d + \widehat{v_{46}}.d = 5$, $\widehat{v_{63}}.d + \widehat{v_{36}}.d = 5$, $\widehat{v_{62}}.d + \widehat{v_{26}}.d = 12$ y $\widehat{v_{61}}.d + \widehat{v_{16}}.d = 12$. Claramente, el MLDC se identifica con una longitud de 5 y los nodos $y = 4$, $x = 6$ ($6 \to 3 \to 4 + 4 \to 6$).

Capítulo 3

Procedimiento experimental

Este capítulo incluye secciones que describen los experimentos computacionales realizados con los algoritmos enumerados en el capítulo anterior. Se visualizan mediante gráficas y tablas los resultados obtenidos. Además, incluimos nuestro analísis de estas observaciones.

3.1. Descripción del material

Todos los algoritmos fueron escritos en C y compilados en GCC con la opción -O. Las pruebas se realizaron en un procesador Iter Xeon (R) CPU ES-1620 v2 8GHz x 8 y 64GB de RAM corriendo bajo Ubuntu 14.04 LTS.

3.2. Descripción de los experimentos

Las longitudes de todos los arcos siempre tomaron valores enteros en los grafos sintéticos que hemos usado. Por lo tanto, cada una de las implementaciones de los algoritmos sólo emplea aritmética en punto fijo. Los grafos no están necesariamente conectados. En cada caso, primero se identifican los componentes fuertemente conexas (scc) utilizando el algoritmo de **Tarjan (1972)**. Luego, cada algoritmo en el experimento se ejecuta por separado en cada scc. Cuando se calculan las scc, se vuelve a etiquetar los nodos en el orden en que son visitados en la búsqueda del primero en profundidad dentro del algoritmo de Tarjan. Los tiempos de ejecución no incluyen las operaciones de I/O del grafo, pero incluyen el cálculo de la SCC.

- **Algoritmos Implementados:**
  - **El algoritmo de Foyd-Warsahll (FW):** Hemos incluido el algoritmo de **Floyd (1962) y Warshall (1962)** en el experimento. Para este algoritmo, $v_{ij}^k.d$ es la longitud más corta del camino desde v_i a $v_j \forall i, j, k \in \{1, ..., n-1\}$. En la iteración $k-th$, el algoritmo calcula $v_{ij}^k.d$ para todos los nodos v_i y v_j, incluyendo los casos $i = j$. Uno puede calcular $v_{ij}^{k+1}.d$ utilizando la siguiente relación recursiva (ver **Ahuja et al., 1993, page 148**): $v_{ij}^{k+1}.d =$

mín$\{v_{ij}^k.d, v_{ik}^k.d + v_{kj}^k.d\}$.

Supongamos que se calcula $v_{ij}^{k+1}.d$ para un k y i fijos y dejando j variar. Uno puede acelerar los cálculos en la práctica evitando los cálculos cuando $v_{ij}^k \geq mldc$ cuando las longitudes de los arcos son no negativas. En el algoritmo de FW, las variables respuestas, fue el timpo de CPU en segundos, así como el número de veces que una etiqueta distancia cambia.

- **El algoritmo de Johnson:** Hemos implementado el algoritmo de Dijkstra usando "Heap"binarios. En el caso de que el grafo contiene alg
un arco con longitud negativa, la distancia inicial se calcula utilizando el algoritmo de **Pallotino (1984)**, que es una variante de los algoritmos correctores de etiqueta. Los $n-1$ árboles de caminos mínimos son determinados por la aplicación del algoritmo de Dijkstra como ya fue comentada en el capítulo anterior. Todas las operaciones implementadas para la estructura heap enmplean un tiempo $log(size)(heap))$. Cada cálculo del camino más corto de un nodo v_s, sólo intenta alcanzar los nodos de la misma SCC, y restringido a los nodos cuya etiqueta de distancia sea inferior que mldc (el mejor valor actual para el MLDC encontrado por el algoritmo).
- **El MLDC (el nuevo programa):** Hemos puesto a prueba el algoritmo que hemos creado, para ver si en la práctica se comporta tal y como predice su complejidad teórica basada en el caso peor.

- **Grafos en la experimentación:** Hemos centrado nuestros experimentos en grafos obtenidos por 3 generadores.
Nuestro primer conjunto fue creado usando el generador NETGEN, que fue desarrollado por **Klingman et al. (1974)**. Hemos generado grafos aleatorios con $n \in \{2.000, 4.000, 6.000, 8.000, ..., 20.000\}$ y $m \in \{2n, 4n, ... , 128n\}$. El mayor de estos grafos contiene aproximadamente 2.5 millones de arcos. Esto no se acerca a la cantidad máxima que permite el generador NETGEN, que son 40 millones. Los siguientes parámetros fueron fijados para este problema: sources = 1, sinks = 1, averagedegree = width, mincost = 1 y maxcost = 10.000. Hemos generado diez repeticiones para cada combinación de los parámetros n y m, lo que resulta en 700 $(10 \cdot 7 \cdot 10)$ datos diferentes para cada uno de los 3 algoritmos usados.

Nuestro segundo grupo de casos se crearon usando el generador Gridgen desarrollado por **Lee y Orlin (1.991)**. En este caso n varía de 2.000 a 20.000 (como en el conjunto anterior de los casos), pero se usa el parámetro width$\in$ {2, 4, ... , 128}, donde "width.^{es} la anchura del grafo (tamaño de cada malla).

Como el "*width*" incrementa, la "longitud"del grafo disminuye en consecuencia para que el número de nodos no se vea afectada. Los siguientes parámetros fueron fijados para este problema: sources = 1, sink = 1 averagedegree = width, mincost = 1 y maxcost = 10.000. En este caso, hemos generado once repeticiones para cada combinación de n y $width$, resultando en 2.310 casos $(11 \cdot 10 \cdot 7 \cdot 3)$.

Hemos usado un tercer generador con el fin de crear el tercer grupo de casos. Este generador, el cual lo llamamos HPGEN primero construye un camino de **Hamilton** en el que la longitud del arco se elige de manera uniforme al azar dentro del intervalo [1, 10]. Cada uno de los restantes $m-n+1$ arcos son incidentes a un par seleccionado al azar de nodos. La longitud de cada uno de estos arcos se fija uniformemente al azar en el intervalo [1, 10.000]. Generamos grafos aleatorios con n y m teniendo los mismos valores que en nuestra primera colección de casos de problemas. Como antes, hemos creado diez repeticiones para cada combinación de estos parámetros, lo que resulta en 2.100 casos ($10 \cdot 10 \cdot 7 \cdot 3$).

Todos los arcos de longitud son no negativos en nuestros experimentos. En consecuencia, el algoritmo de Johnson no necesita llevar a cabo la ejecución del algoritmo de Pallotino con el fin de transformar los costes/pesos en valores no negativos.

3.3. Resultados obtenidos

En esta sección crearemos las tablas de los tiempos de CPU obtenidos por los tres generadores para cada algoritmo, colocaremos el mínimo, la media y el máximo de los valores obtenidos. En las gráficas visualizaremos el mínimo, la media, el máximo y cada uno de los tiempos de CPU obtenidos con cada uno de los experimentos realizados. Representaremos en unas gráficas el tiempo de CPU frente al número de vértices (n) y en las otras, el tiempo de CPU frente al número de aristas por cada vértice ($\frac{m}{n}$). Usaremos una gráfica para cada algoritmo, ya que, las diferencias en segundos son muy significativas, y al colocarlas en una sola gráfica, comprobamos que no se visualizaban muy claros los datos, para luego poder hacer el análisis correctamente.

	Floyd y Warshall			**Johson**			**mlcd**		
n	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2000	0,0572	0,0866	0,118	0,0015	0,0136	0,0447	0,1298	0,9746	3,0617
4000	0,3397	0,4258	0,6167	0,0049	0,0306	0,0989	0,5968	5,1304	16,4863
6000	0,7786	1,0003	1,2705	0,0078	0,0515	0,1617	1,5455	12,7501	39,6604
8000	1,4678	1,8324	2,5921	0,0131	0,0708	0,2268	3,1604	23,8248	77,3856
10000	2,1582	2,9083	4,193	0,016	0,0949	0,3107	4,2371	41,3792	130,6427
12000	3,4999	4,3776	5,8421	0,0174	0,1239	0,4382	7,9596	61,1444	191,5249
14000	4,7149	6,1764	7,7088	0,0228	0,1473	0,4852	11,7004	87,7999	268,8481
16000	6,9754	8,6023	11,5287	0,0308	0,1735	0,5929	17,515	120,8406	368,5378
18000	8,6325	10,426	15,9158	0,0293	0,2091	0,6491	26,4204	163,0164	481,3434
20000	9,973	13,057	16,0269	0,0462	0,2406	0,8081	28,9048	212,3462	639,6002

Tabla 3.1: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Netgen *vs* n

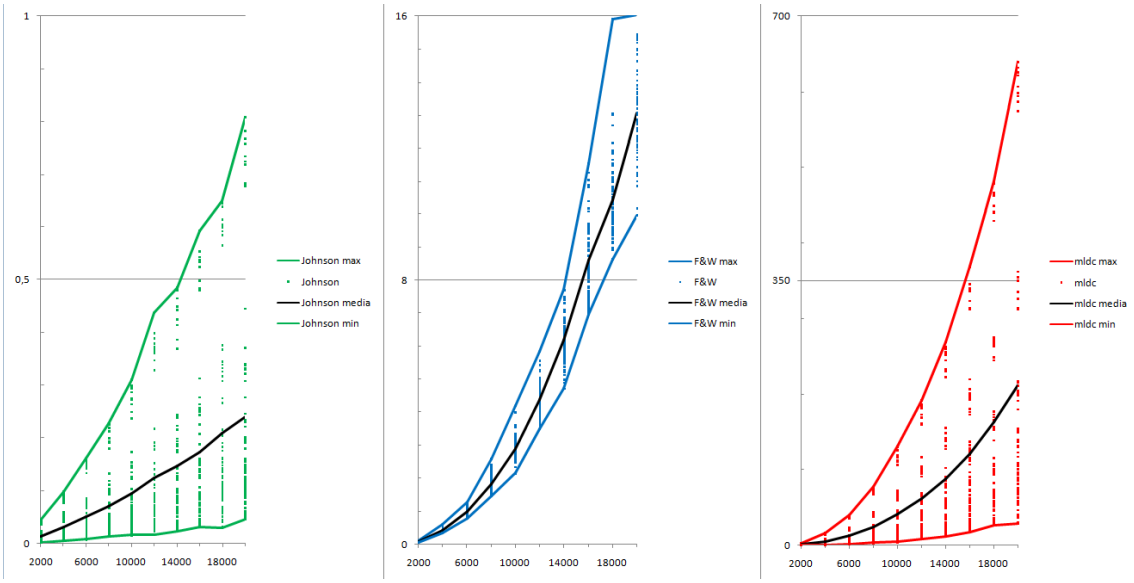


Figura 3.1: Netgen

	Floyd y Warshall			**Johson**			**mlcd**		
$\frac{m}{n}$	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2	0,0687	4,8746	16,0269	0,0015	0,0274	0,0891	0,1298	12,9060	58,9557
4	0,0698	4,9128	15,9158	0,0037	0,0447	0,155	0,2192	20,8926	75,9757
8	0,0753	4,9055	15,2928	0,0044	0,0529	0,1495	0,3435	30,5308	108,7631
16	0,0666	4,7788	15,4477	0,0057	0,0651	0,1917	0,4813	45,5717	154,4012
32	0,0665	4,7439	14,3488	0,01	0,0978	0,2462	0,8696	69,8848	216,5447
64	0,0572	4,6014	14,3403	0,018	0,1667	0,4453	1,4721	114,7677	362,1746
128	0,0636	4,4553	14,2313	0,0367	0,3380	0,8081	2,7136	196,4291	639,6002

Tabla 3.2: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Netgen *vs* $\frac{m}{n}$

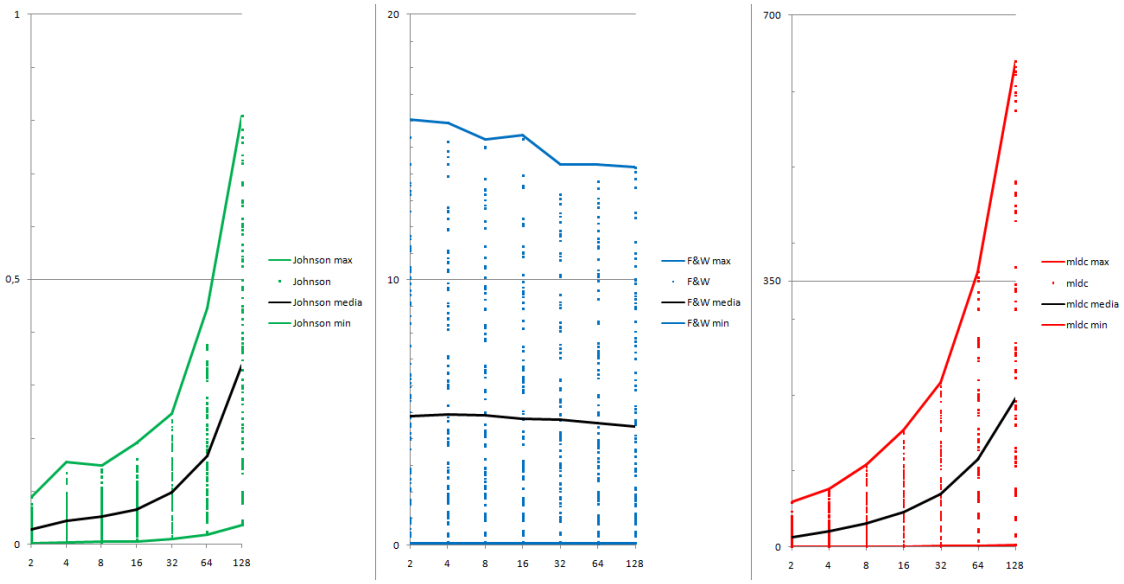


Figura 3.2: Netgen

	Floyd y Warshall			**Johson**			**mlcd**		
n	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2000	0,0477	0,0687	0,1095	0,0023	0,0132	0,0473	0,0803	0,9604	3,6235
4000	0,2382	0,3073	0,439	0,0059	0,0307	0,0969	0,434	5,0698	18,2287
6000	0,5657	0,7417	1,1253	0,0091	0,0512	0,1755	1,052	12,6308	44,6171
8000	1,1094	1,4381	2,0208	0,0152	0,0706	0,2128	1,8079	23,8251	85,2683
10000	1,94	2,3471	3,1535	0,0205	0,0956	0,3096	3,0843	38,7831	134,3623
12000	2,6326	3,3267	4,5695	0,0242	0,1225	0,4498	4,4237	58,2655	199,1566
14000	3,5745	4,5353	6,1768	0,0344	0,1462	0,5036	6,015	81,6997	272,5169
16000	4,8275	5,9755	9,8466	0,027	0,1780	0,625	8,1303	108,9512	354,5528
18000	6,5917	8,2158	11,1049	0,0264	0,2044	0,6688	12,2886	143,9268	490,5352
20000	8,5637	10,1600	15,0379	0,0363	0,2326	0,8081	16,5573	186,7026	659,0011

Tabla 3.3: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Gridgen *vs* n

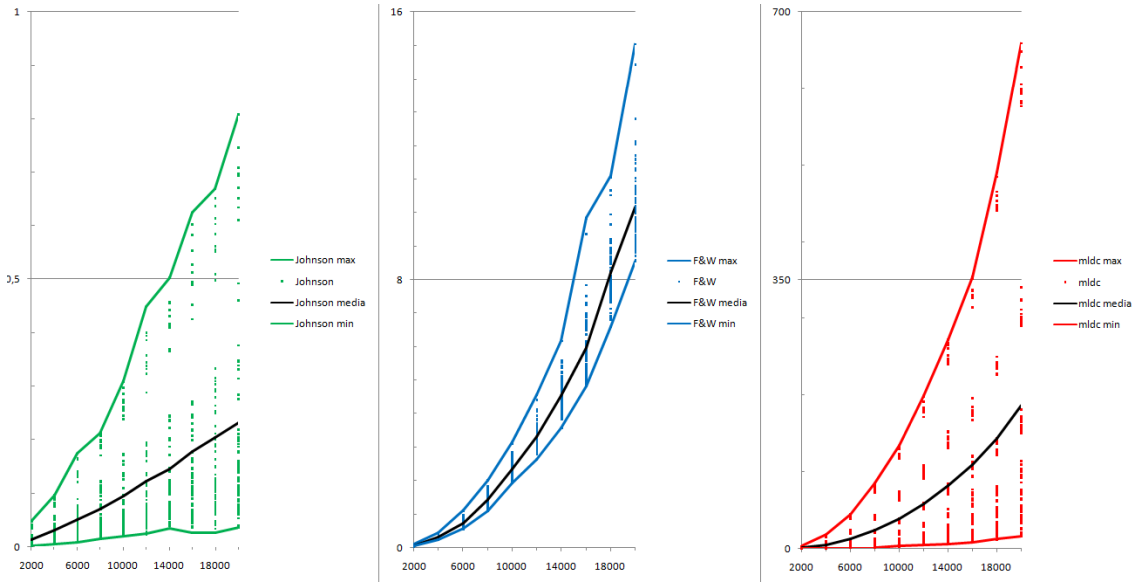


Figura 3.3: Gridgen

	Floyd y Warshall			**Johson**			**mlcd**		
$\frac{m}{n}$	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2	0,0498	3,5544	11,2606	0,0023	0,0394	0,1137	0,0803	7,9243	38,1211
4	0,0519	3,5857	11,3193	0,0032	0,0443	0,1293	0,1551	13,5708	51,4525
8	0,0511	3,7408	12,1262	0,0037	0,0564	0,1568	0,2434	22,4581	77,3023
16	0,057	3,6855	11,625	0,0046	0,0668	0,1925	0,4327	35,8218	114,6427
32	0,0477	3,7837	11,4881	0,0086	0,0968	0,2257	0,7891	61,4973	181,9131
64	0,0513	3,8254	14,4251	0,0165	0,1659	0,4927	1,4575	110,6778	340,9373
128	0,0576	3,8059	15,0379	0,0345	0,3319	0,8081	3,1515	210,6205	659,0011

Tabla 3.4: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Gridgen *vs* $\frac{m}{n}$

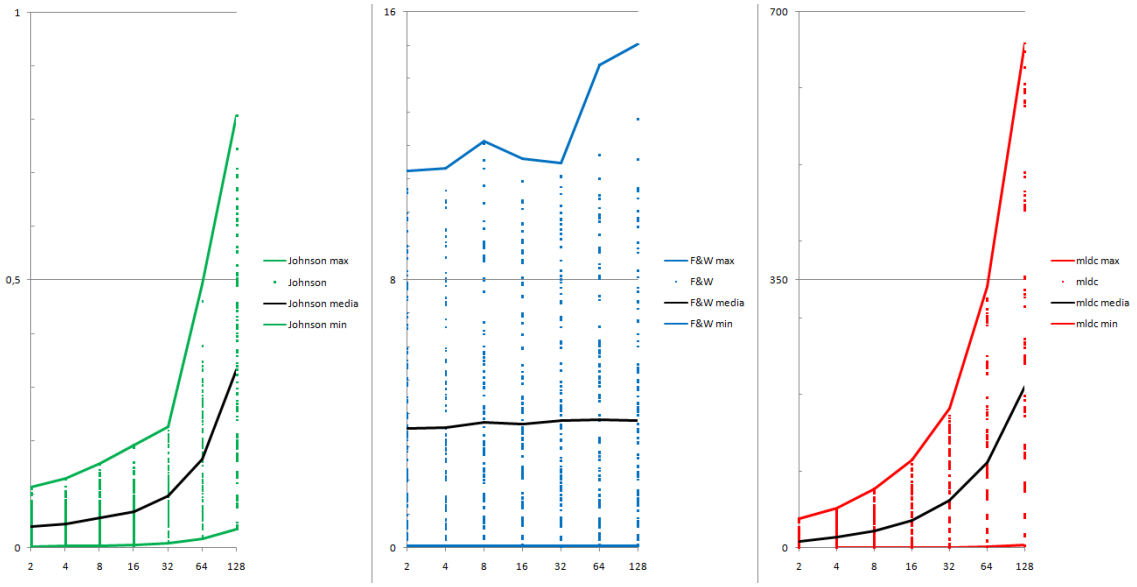


Figura 3.4: Gridgen

	Floyd y Warshall			**Johson**			**mlcd**		
n	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2000	0,0754	0,0877	0,1151	0,0011	0,0113	0,0433	0,1211	1,2466	4,3887
4000	0,3759	0,4318	0,5242	0,0026	0,0254	0,0945	0,5584	6,6027	24,559
6000	0,8471	0,9608	1,1202	0,0035	0,0414	0,188	1,3685	16,5827	60,5804
8000	1,552	1,7893	2,1679	0,0069	0,0614	0,2334	2,5883	32,4271	117,2129
10000	2,6523	2,9276	4,2393	0,0069	0,0804	0,3217	4,1642	54,9197	199,7345
12000	3,983	4,5689	7,6188	0,0104	0,1048	0,423	6,3666	84,1174	301,7198
14000	6,3663	6,8143	7,9615	0,0127	0,1278	0,516	10,8379	118,3799	419,357
16000	9,3057	10,2697	12,4012	0,0173	0,1545	0,6074	15,62	162,1243	551,6112
18000	10,0593	11,0582	14,4562	0,0135	0,1820	0,6866	21,5356	212,3424	724,178
20000	12,7816	14,7130	17,3994	0,0114	0,2091	0,8068	29,6988	269,5772	905,2643

Tabla 3.5: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Hpgen *vs* n

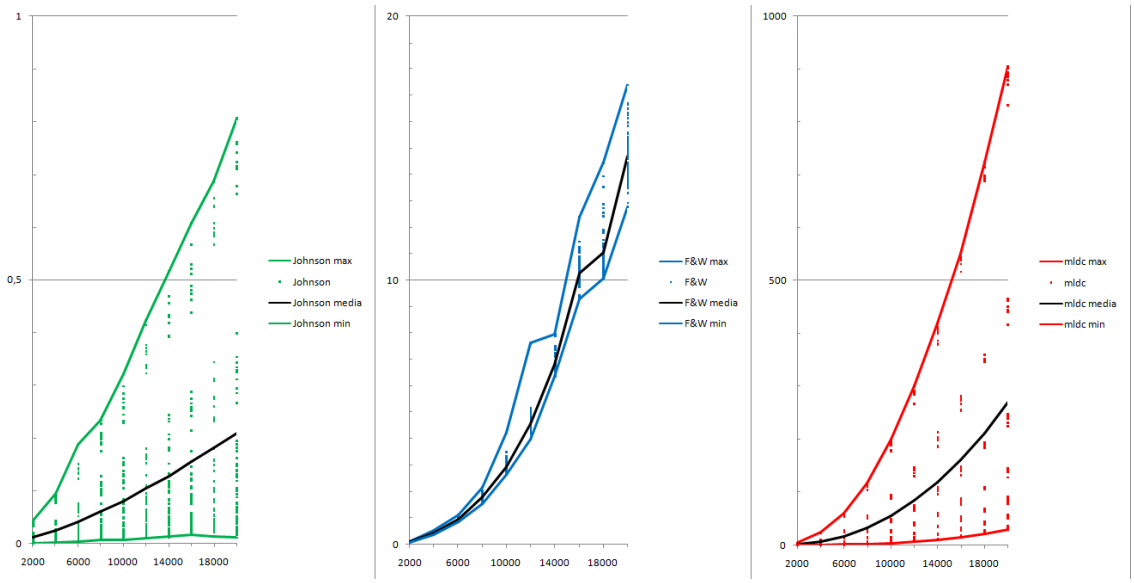


Figura 3.5: Hpgen

	Floyd y Warshall			**Johson**			**mlcd**		
$\frac{m}{n}$	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2	0,0787	5,4203	16,6475	0,0011	0,0146	0,0461	0,1211	10,1763	36,1677
4	0,0756	5,3128	16,1419	0,0016	0,0204	0,0473	0,2127	17,8391	58,3305
8	0,0754	5,2273	15,3703	0,0024	0,0303	0,0761	0,3429	29,0077	91,831
16	0,0774	5,2859	15,8528	0,0042	0,0505	0,1516	0,5889	48,2897	145,9937
32	0,0755	5,3576	16,3333	0,0085	0,0841	0,2329	1,0738	84,2733	248,8539
64	0,0785	5,3994	16,2771	0,0165	0,1589	0,3988	2,0617	160,5179	466,9422
128	0,0822	5,5317	17,3994	0,0337	0,3399	0,8068	4,1568	320,7202	905,2643

Tabla 3.6: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Hpgen *vs* $\frac{m}{n}$

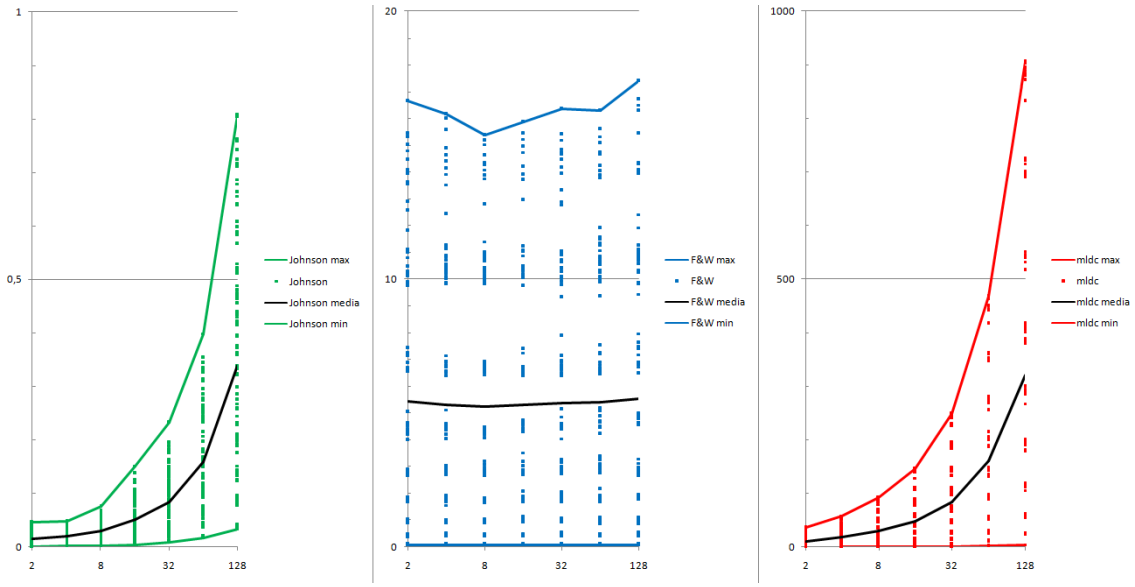


Figura 3.6: Hpgen

Al realizar estos expermientos, nos hemos encontrado con que nuestro algoritmo (**mldc**) no era eficiente, y al intentar dar una explicación de lo ocurrido, nos hemos dado cuenta que la razón era que al usar solo longitudes positivas, los algoritmos se saltaban la mayoría de los pasos, por este motivo, hicimos un nuevo experimento usando el generador NETGEN, y creando grafos aleatorios con $n \in \{1.000, 2.000, 3.000, ... , 8.000\}$ y $m \in \{2, 4n, 8n, 16n, 32n\}$, fijando los siguientes parámetros, sources = 1, sinks = 1, averagedegree = width, mincost = -100 y maxcost = 10.000. De esta forma, tenemos arcos con longitud negativa. Además, una vez realizado la simulación, quitamos los datos de los resultados donde los algoritmos encontraron circuitos con longitud negativa para que no afecte al estudio realizado. Estos son los tiempos de CPU obtinidos en este nuevo experimento:

	Floyd y Warshall			**Johson**			**mlcd**		
n	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
1000	0,785	1,6714	2,1503	0,0008	0,0029	0,0059	0,0251	0,2570	0,3673
2000	9,3151	19,8448	25,6455	0,0012	0,0064	0,0148	0,2755	3,4431	4,6921
3000	31,8609	68,7215	90,1919	0,0037	0,0105	0,0201	0,8753	12,6641	17,6376
4000	76,6159	160,2121	216,9431	0,0043	0,0140	0,0333	1,8658	32,3842	48,2393
5000	153,3234	323,6644	428,1037	0,0086	0,0180	0,0337	3,0855	70,0118	101,9575
6000	271,0406	573,8625	753,3517	0,0074	0,0232	0,0445	6,0481	129,1837	190,4482
7000	436,4959	913,1368	1187,004	0,0111	0,0296	0,0572	8,0129	209,4597	313,6423
8000	670,9178	1382,6667	1787,5424	0,0129	0,0354	0,0798	11,3327	314,8968	482,6501

Tabla 3.7: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Netgen *vs* n

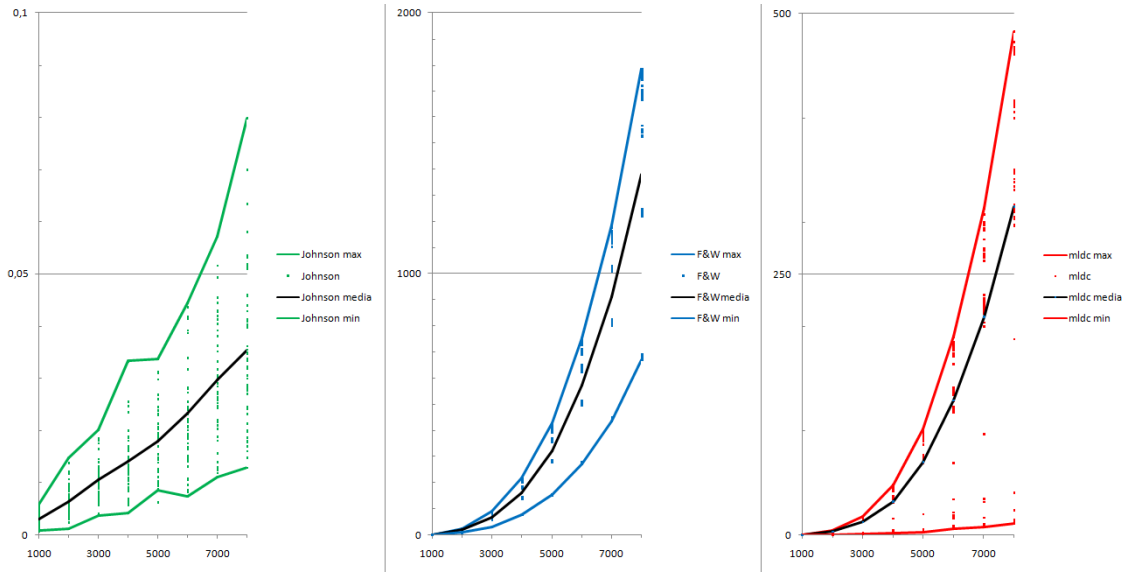


Figura 3.7: Netgen

	Floyd y Warshall			**Johson**			**mlcd**		
$\frac{m}{n}$	Mínimo	Media	Máximo	Mínimo	Media	Máximo	Mínimo	Media	Máximo
2	0,785	212,5246	692,9133	0,0008	0,0095	0,0277	0,0251	11,2911	187,8707
4	1,3489	386,3997	1247,1206	0,0013	0,0144	0,0461	0,2319	98,0911	346,6907
8	1,8431	486,6724	1567,8285	0,0022	0,0162	0,043	0,2962	133,8279	417,1734
16	1,97	530,4176	1721,5352	0,0028	0,0201	0,0532	0,3185	133,2119	482,6501
32	1,9918	562,9554	1787,5424	0,0039	0,0291	0,0798	0,3483	105,7760	350,6946

Tabla 3.8: Mínimos, medias y máximos del tiempo de CPU (en segundos) de cada algoritmo usando el Netgen *vs* $\frac{m}{n}$

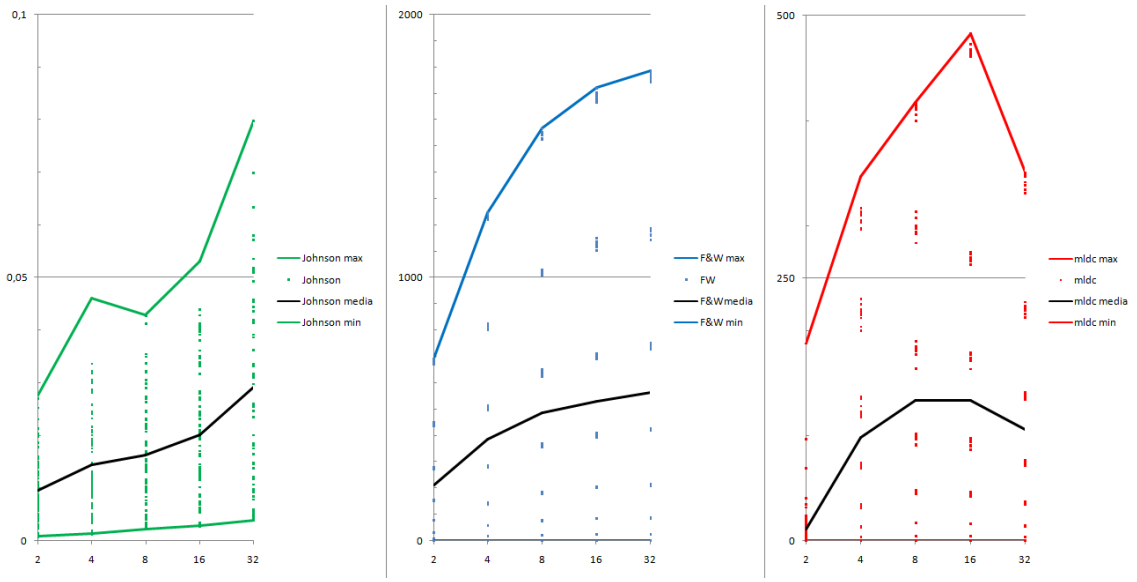


Figura 3.8: Netgen

3.4. Análisis de los resultados

Para grafos sin arcos con longitud negativa, podemos comprobar en las gráficas que el algoritmo de **Johnson** es el algoritmo más rápido en realizar todos los cálculos, aunque, si nos fijamos en las gráficas de $\frac{m}{n}$, podemos ver que el algoritmo de **Floyd-Warsahll** es el único algoritmo que trabaja igual aunque varíen la proporción de aristas por cada vértice, siendo el único que es más o menos constante si salen 128 aristas de cada vértice que si solo salen 2 aristas de cada vértice, esto convierte al algoritmo de **Floyd-Warsahll** en un buen candidato (habría que comprobarlo) si de cada vértice salen muchísimas aristas (al menos más de 128), porque, los demás algoritmos se dispara consideradamente el tiempo de ejecución de CPU, tal y como se muestra en las gráficas. Por último, se puede ver, que nuestro nuevo algoritmo (**mldc**) no es eficiente, ya que, es 800 veces más lento que el **Johnson**, y no tiene ningún comportamiento diferente al **Johnson**, para que digamos que podría ser más eficiente en algún caso especial.

Para grafos cuyos arcos admitan longitudes positivas y negativas, podemos comprobar en las gráficas que el algoritmo de **Johnson** sigue siendo el algoritmo más rápido en resolver el problema del circuito de longitud mínima, pero, en este caso, nuestro nuevo algoritmo (**mldc**) es cuatro veces más eficiente que el algoritmo de **Floyd-Warsahll**, convirtiéndose en un buen candidato como alternativa de éste.

Conclusiones

Una primera conclusión, es que tenemos tres algoritmos muy eficientes para la búsqueda del circuito de menor longitud, que tienen sus pros y sus contras.

El algoritmo de **Johnson** es el más eficaz. Sin embargo resulta más complicado de programar en un lenguaje de programación de alto nivel que los otros dos algoritmos estudiados.

El algoritmo de **Floyd-Warsahll** es eficaz si la cantidad de datos es enorme, siempre que los arcos sólo tengan pesos positivos (como tiempo, distancias, etc.), además es el más fácil de programar.

Nuestro algoritmo, el **mldc** es una buena alternativa al algoritmo de **Floyd-Warsahll**, siempre que los arcos admitan pesos positivos y negativos (coordenadas, costes, benefecios, etc.).

Bibliografía

[1] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[2] Cormen, Thomas H. Cormen, and Charles E. Leiserson. *Introduction to Algorithms*. MIT Press, 2009.

[3] Robert W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[4] Giorgio Gallo and S. Pallotino. Shortest path methods in transportation models. *Publication of: Elsevier Science Publishers BV*, 1984.

[5] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[6] Darwin Klingman, Albert Napier, and Joel Stutz. Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20(5):814–821, 1974.

[7] Liam Roditty and Virginia V. Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 180–189. IEEE, 2011.

[8] Angelo Sifaleras. Minimum cost network flows: Problems, algorithms, and software. *Yugoslav Journal of Operations Research ISSN: 0354-0243 EISSN: 2334-6043*, 23(1), 2013.

[9] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[10] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.