UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

COMPARACION COMPUTACIONAL DE ALGORITMOS PARA EL PROBLEMA DE ASIGNACION "CUELLO DE BOTELLA"

T E S I S

Que para obtener el Título de

A C T U A R I O

P r e s e n t a

ANA MARGARITA PEREZ MIRANDA

México, D. F. Septiembre 1989

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

PREFACIO

Vivimos en un mundo que día a día se desenvuelve con mayor velocidad. Las comunicaciones, cada día más rápidas y eficientes, exigen soluciones más precisas y oportunas a los problemas cotidianos y, al mismo tiempo, estos problemas involucran más variables que necesitan ser consideradas en la toma de decisiones. Tal es el caso de las matrices de Insumo-Producto, Planeación de Mercadotecnia, Asignación Optima de Recursos en Macroproyectos Industriales, Distribución de Recursos en Problemas de Interés Social, y muchos otros similares.

El área de las matemáticas que se encarga de atacar estos problemas es la "Investigación de Operaciones"; dentro de ésta encontramos el denominado "Problema de Asignación" y una variante de éste que es el conocido como "Problema de Asignación Cuello de Botella", los cuales se definen y explican a detalle en el presente trabajo.

En este trabajo se presentan algunos algoritmos para resolver el Problema de Asignación Cuello de Botella, comparándolos computacionalmente entre sí. Durante el desarrollo del trabajo observamos que podían aplicarse algunas mejoras a los procedimientos conocidos, las cuales presentamos en el texto.

1. INTRODUCCION

1.1 Historia y Significado de la Investigación de Operaciones.

Lo que hoy día conocemos como Investigación de Operaciones tuvo su desarrollo durante la Segunda Guerra Mundial, aunque sus orígenes se remontan muchos años atrás.

En el período comprendido entre la Primera y la Segunda Guerra Mundial Inglaterra decidió formar pequeños grupos interdisciplinarios de científicos que empezaron a desarrollar métodos prácticos y técnicas de detección de equipos militares.

Se formaron muchos grupos, principalmente en Estados Unidos, Canadá, Francia e Inglaterra y fue entonces cuando se le empezó a denominar a este trabajo por diversos nombres de los cuales podemos citar: análisis operacional, investigación de operaciones, análisis de sistemas, evaluación de operaciones, investigación de sistemas y ciencia de la administración. El nombre de Investigación de Operaciones fue y es el más ampliamente usado, y por consiguiente el que se utiliza en el presente trabajo.

Al finalizar la guerra, se crearon a nivel mundial muchas fundaciones y organizaciones con gente que hubiera o estuviera trabajando en Investigación de Operaciones, lo que ayudó a todas estas pequeñas comunidades para consolidar los avances que hasta entonces se tenían.

Al finalizar la década de 1940 comenzó una nueva revolución cuando aparecieron en el mercado las computadoras electrónicas.

Muchas organizaciones desarrollaron trabajos tales como la simulación en computadoras, el análisis de costo-beneficio y sistemas de análisis; también hicieron muchos avances en áreas tales como la teoría de juego, la teoría de búsqueda, la teoría del valor, y otras técnicas matemáticas. En esta época la Investigación de Operaciones se extendió hacia el área industrial, diversificándose en un universo de ramas en las cuales se puede aplicar.

En las décadas recientes, la Investigación de Operaciones ha continuado su expansión, siendo utilizada en las áreas de gobierno no militares tales como transportes, educación, servicios sociales, sanidad, etc., así como en la industria privada, con lo que se estima un impacto significativo en el futuro.

En la actualidad se cuenta con miles de teorías desarrolladas y aplicaciones a diversos campos, así como una gran cantidad de libros y revistas sobre las distintas áreas de la Investigación de Operaciones a nivel mundial. En muchos programas de educación se incluye ya esta área

como conocimiento básico para los estudiantes, para que las teorías y modelos que hasta ahora han sido desarrollados tengan continuidad y se puedan aplicar a la realidad.

Bajo el enfoque actual, la Investigación de Operaciones se puede definir como: "La aplicación del método científico, por equipos interdisciplinarios a problemas que comprenden el control de sistemas organizados hombre-máquina, para dar soluciones que sirvan mejor a los propósitos de la organización como un todo".[1]

En México la Investigación de Operaciones se utiliza dentro del sector de servicios públicos, entre otros en Petróleos Mexicanos, la Comisión Federal de Electricidad, Fertilizantes Mexicanos, el Banco de México, la Secretaría de Obras Públicas, la Secretaría de Comunicaciones y Transportes, la Secretaría de Recursos Hidráulicos, la Secretaría de la Presidencia, el Instituto Mexicano del Seguro Social, el Departamento del Distrito Federal, la Compañía Nacional de Subsistencias Populares (CONASUPO), etc., dentro de los cuales se han desarrollados distintos modelos matemáticos con aplicaciones diversas.[2]

Las raíces del desarrollo de la Investigación de Operaciones ya han sido cimentadas, su crecimiento depende del aprovechamiento que se les pueda dar. Si la Investigación de Operaciones se sigue extendiendo como hasta ahora, su futuro ofrece grandes promesas hacia su importante desarrollo y práctica.

1.2 Método de la Investigación de Operaciones.

La Investigación de Operaciones construye y resuelve modelos matemáticos del sistema bajo estudio. El modelo se representa por una función objetivo o función de utilidad sujeta a ciertas restricciones. La función de utilidad tiene la siguiente estructura:

$$U = f (X_i , Y_i)$$

donde:

- **U** es la utilidad o valor de la ejecución del sistema
- f es la relación entre **U**, X_i y Y_i
- X_i son las variables no controlables
- Y_i son las variables controlables.

Las restricciones se representan por medio de ecuaciones o inecuaciones (desigualdades) y expresan los límites de las variables. Tomando todo esto se contruye el modelo del sistema objeto de estudio.

Una vez que se ha construido el modelo, puede usarse un algoritmo para encontrar exacta o aproximadamente los valores de las variables no controlables, que cumplan con las restricciones y que producen la mejor ejecución del sistema, es decir, se deriva una solución óptima, que es aquella que minimiza o maximiza (según convenga) la función objetivo.

Ahora bien, por una parte, se desea que el modelo sea lo más cercano a la realidad, es decir, que no la distorsione y por otra, se quieren modelos sencillos para que sean ejecutables. El modelador debe resolver este compromiso mediante modelos satisfactorios desde ambos puntos de vista.

Una vez construido el modelo, la solución se obtiene mediante un análisis matemático o por medio de la experimentación (simulación). En la actualidad, el uso de la computadora digital reduce el tiempo utilizado por los algoritmos para hallar la solución óptima. Cuando el tiempo de computadora es demasiado alto, se puede optar por un método heurístico para la obtención de la solución o también se puede recurrir a una solución analógica.

Debido a que un modelo es una representación del problema real, la solución óptima del modelo no necesariamente es la mejor solución del problema real, pero se puede decir que es una "buena" aproximación.

Existen muchas y diferentes formas de agrupación de los problemas prototipo en la Investigación de Operaciones, algunos libros los presentan con ejemplos clásicos, otros se basan en el modelo matemático a aplicar, pero, en general, los problemas prototipo se pueden clasificar en los siguientes modelos:

1. Programación Lineal.
2. Programación Entera.

3. Teoría de Redes.
4. Programación No Lineal.
5. Procesos Estocásticos.
6. Teoría de Colas.
7. Teoría del Valor.
8. Análisis de Decisión.
9. Teoría de Juegos.
10. Inventarios.
11. Simulación.
12. Programación Dinámica.
13. Búsqueda.

En el presente trabajo el problema del que haremos referencia se analiza en sus detalles en el siguiente capítulo.

1.3 El Problema de Asignación.

Dentro de las diversas áreas de la Investigación de Operaciones el problema a discutir en el presente trabajo se puede ubicar dentro de la programación lineal y entera.

Primero plantearemos el llamado Problema de Asignación con el siguiente ejemplo: Supongamos que tenemos un taller con n obreros que deben hacer n trabajos. Supongamos también que todos los obreros son capaces de hacer todos los trabajos, pero sus aptitudes son generalmente distintas, es decir, dos obreros no tardan necesariamente el mismo tiempo en terminar un trabajo; asimismo, los trabajos son, en principio, de distintos grados de dificultad, es decir, dos trabajos hechos por el miso obrero tienen una duración generalmente diferente.

El problema que se plantea es asignar uno y un sólo trabajo a cada obrero, de manera tal que se realicen todos los trabajos a la mayor brevedad posible.

El ejemplo planteado es un problema clásico de asignación. Para obtener una visión clara de éste definamos:

Definición I. Problema de Apareamiento. [3]

Sean dos conjuntos finitos, no vacíos **R** y **L**, tales que $|R| \leq |L|$. Sea **H** un subconjunto de **R**x**L**. Los pares (r,l) que pertenecen a H se les llama *compatibles* , mientras que los que no pertenecen se les llama *incompatibles.* Un *apareamiento* **M** es una colección de pares compatibles (r_1,l_1), (r_2,l_2), ... , (r_p,l_p), tales que los elementos $r_1, r_2, \ldots, r_p$ son todos distintos, así como los elementos $l_1, l_2, \ldots, l_p$. En otras palabras, un apareamiento es un subconjunto **M** de **H** que define una correspondencia uno-a-uno entre un subconjunto de **R** y un subconjunto de **L**.

El *Problema de Apareamiento* consiste en determinar un apareamiento de cardinalidad máxima.

El problema de apareamiento puede ser representado en forma matricial en términos de *matriz de compatibilidad* para la relación **H**, cuyas entradas, en la matriz **A**, **h**(r,l) sean 1 ó 0, dependiendo si (r,l) es compatible o no. Bajo este concepto, un apareamiento **M** corresponde a un subconjunto de unos de la matriz, tales que no existen dos en la misma línea (una línea es un renglón o una columna de **A**). Ver figura 1.3.1, en la cual las entradas de la matriz **A** que están encerradas en un círculo representan un apareamiento **M** de cardinalidad 7.

Definición II. Asignación. [3]

Una *asignación* de **R** a **L**, compatible con **H**, es un apareamiento **M** con $|M| = |R|$. Esto es, una función uno-a-uno de todos los elementos de **R** hacia **L**, constituída por pares en **H**.

Matriz A

	1	2	3	4	5	6	7	8	9
1	(1)	1	0	1	1	1	0	1	1
2	1	1	0	0	(1)	1	1	0	0
3	1	1	(1)	0	1	1	0	0	1
4	0	0	1	1	0	1	0	1	0
5	1	0	0	(1)	1	0	0	1	1
6	0	1	1	1	1	(1)	1	1	1
7	0	0	0	1	1	1	0	0	(1)
8	1	0	0	1	1	1	1	(1)	0
9	1	1	1	0	1	0	0	1	1

Figura 1.3.1

El primer problema que se presenta es el de la existencia de una asignación. Es fácil si se resuelve el problema de apareamiento. Si $|M| = |R|$, entonces el apareamiento es una *asignación* . Si $|M| < |R|$, entonces la asignación no existe.

Definición III. Problema de Asignación. [3]

Supongamos que al menos hay una asignación **M** relativa a la relación **H**. Sea **q** una función real valuada en **H**, y asociemos con cada asignación **M** el número:

$$q(M) = \sum_{(r,l) \in M} q(r,l)$$

Aquí $q(r,l)$ puede ser interpretado como el *"costo"* del par (r,l), y $q(M)$ como el *"costo"* de **M**.

El *Problema de la Asignación* consiste en minimizar $q(M)$ sobre todas las asignaciones **M**.

Se han dado muchos algoritmos para la resolución del Problema de Asignación[2][3]. En el presente trabajo no se citará ninguno, ya que el objetivo de éste no compete con alguno de ellos. En el siguiente capítulo se examinará una variante del Problema de Asignación, analizándose algunos algoritmos y comparándolos entre sí.

2. EL PROBLEMA DE ASIGNACION CUELLO DE BOTELLA.

En este capítulo se plantea el Problema de Asignación Cuello de Botella y se presentan varios métodos exactos para su resolución.

En la sección 2.1 se da la definición general del Problema de Asignación Cuello de Botella. Enseguida se presenta el Algoritmo de Gross en la sección 2.2.; posteriormente el Algoritmo de Garfinkel se trata en la sección 2.3, seguido de el Algoritmo de Derigs - Zimmermann en la sección 2.4 y por último el Algoritmo de Romero en la sección 2.5.

2.1 Planteamiento General.

El *Problema de Asignación "Cuello de Botella" (Bottleneck)*, conocido también como el *"Problema Min-Max"* fue mencionado por primera vez por Fulkerson, Glieksberg y Gross[4]. El Cuello de Botella de una asignación M se define como:

$$f(M) = \max_{(r,l)\in M} q(r,l)$$

En lo siguiente supondremos que al menos una asignación M existe y que si $(r,l) \notin H \Rightarrow q(r,l) = \infty$.

El *Problema de Asignación Cuello de Botella* consiste en minimizar $f(M)$.

También, visto desde el punto de vista gráfico, este problema consiste en encontrar un apareamiento **M** de máxima cardinalidad en el cual el arco de peso máximo es mínimo, esto se sigue de la filosofía de que 'una cadena es sólo tan fuerte como lo es su eslabón más débil'.

Existe un número finito de niveles $k(r,l)$ para diferentes pares $(r,l) \in H$. Sean $\partial_1, \ldots, \partial_s$ dichos niveles en orden decreciente. Para $p = 1, \ldots, s$ sea H^p el conjunto de todos los pares $(r,l) \in H$ tales que $k(r,l) \geq \partial_p$. Obviamente, **M** es una asignación relativa a H^p si y sólo si $f(M) \geq \partial_p$. Entonces, sólo se tiene que determinar el primer índice **p** tal que existe una asignación relativa a H^p, y cualquier asignación correspondiente resolverá el problema. La prueba puede hacerse sistemáticamente de varias maneras. Se puede empezar de $p = 1$ hasta $p = s$ ó o se puede dar un salto para acercarse al índice deseado **p**.

El Problema de Asignación Cuello de Botella puede plantearse de la siguiente forma:

$$\text{minimizar } z = \max_{x_{ij} = 1} C_{ij}$$

Sujeto a:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{para todo } i = 1,2,\ldots,m$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad \text{para todo } j = 1,2,...,n$$

$$x_{ij} \in \{0,1\}$$

donde $C = \{C_{ij}\}$ es una matriz $m \times n$ de reales ($m \leq n$).

Hay que hacer notar que si se tiene una matriz cuadrada el planteamiento general queda como sigue:

$$\text{minimizar } z = \underset{x_{ij} = 1}{\text{máximo}} \; C_{ij}$$

Sujeto a:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{para todo } i = 1,2,...,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{para todo } j = 1,2,...,n$$

$$x_{ij} \in \{0,1\}$$

2.2 Algoritmo de Gross.[5]

2.2.1 Planteamiento.

Este algoritmo es propuesto para **m=n**, es decir, para una matriz cuadrada, y es el siguiente:

Paso 1. Tome cualquiera de las **n!** posibles asignaciones.
(por ejemplo, tomar x_{ii} , i = 1,2,...,n).

Paso 2. Encontrar $V = \max \{C_{ij} | x_{ij} = 1\}$.

Paso 3. Escoger cualquier elemento [nombrarlo (i^*,j^*)] del conjunto: $B = \{(i,j) \mid C_{ij} = V , x_{ij} = 1\}$.
Encontrar un ciclo que empiece y termine en (i^*,j^*) como sigue:
a) De cada entrada asignada ir a una entrada no asignada, en esa columna con costo menor a V,
b) De cada entrada no asignada ir a la entrada asignada en ese renglón. Si tal ciclo no existe, entonces la asignación es óptima.

Paso 4. Invertir las asignaciones en el primer ciclo encontrado, en otras palabras, para elementos del ciclo sea $x'_{ij} = 1 - x_{ij}$ la nueva asignación. Ir al Paso 2.

Es fácil ver que el ciclo siempre existe para una asignación no óptima. Una posible opción de entradas no asignadas en el Paso 3a sería el conjunto de entradas que constituyen la solución óptima, pero que no están en la presente solución. Un simple argumento inductivo muestra que tal opción produce un mejor ciclo, empezando en cualquiera de los elementos de **B**, ya que se va a una entrada cuyo C_{ij} es menor que **V**.

El problema de degeneración que ocurre en el algoritmo de transporte[6] en el 'paso-clave' ('stepping-stone') no ocurre aquí. El algoritmo termina en no más de n^2 iteraciones, ya que ninguna entrada es repetida como el origen del ciclo.

Debe notarse que, generalmente, habrá un gran número de soluciones óptimas alternativas, pero este algoritmo no da un método para identificarlas. En la práctica lo que se hace es plantear una función objetivo secundaria para así tomar una decisión final.

2.2.2 Ejemplo de Aplicación.

Para efectos prácticos, en todos los algoritmos analizados en el presente trabajo realizaremos el ejemplo con la misma matriz. Supongamos que se tiene la siguiente matriz de costos **C** con **m=n=5**:

	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Paso 1 Tomamos la asignación diagonal, es decir tomamos $x_{ii}=1$ para i=1,2,3,4,5.

Paso 2 Encontramos que V = máx {15,7,20,4,2} = 20

Paso 3 B = {(3,3) | C_{33} = 20 , X_{33} = 1}

Buscamos el ciclo que empiece y termine en (3,3), como se muestra:

	1	2	3	4	5
1	**15**	12	21	16	3
2	6	**7**	5	25	8
3	23	18	**20**	14	3
4	3	6	22	4	15
5	18	3	8	15	**2**

Paso 4 Invertimos las asignaciones quedando como sigue:

	1	2	3	4	5
1	**15**	12	21	16	3
2	6	7	**5**	25	8
3	23	18	20	14	**3**
4	3	6	22	**4**	15
5	18	**3**	8	15	2

Paso 2 V = máx {15,3,5,4,3} = 15

Paso 3 B = {(1,1) | C_{11} = 15 , X_{11} = 1}

Buscamos el ciclo que empiece y termine en (1,1), como se

muestra:

	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Paso 4 Invertimos las asignaciones quedando como sigue:

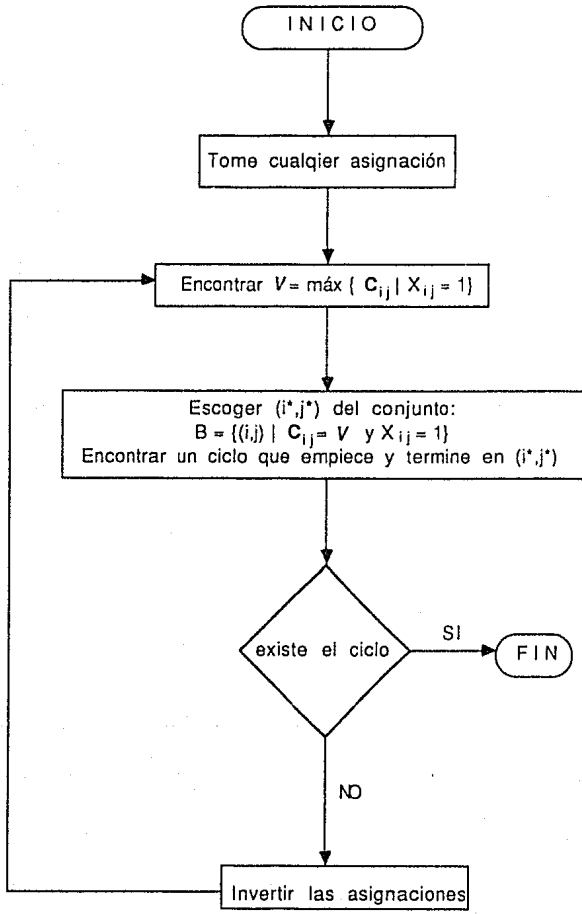
	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Paso 2 $V = \max\{5,12,8,4,3\} = 12$

Paso 3 $B = \{(1,2) \mid C_{12} = 12 , X_{12} = 1\}$

Buscamos el ciclo que empiece y termine en (1,2), pero tal ciclo no existe, por lo tanto la asignación es óptima $z = 12$.

2.2.3 Diagrama de Flujo.


INICIO
Tome cualqier asignación
Encontrar V = máx { C i j | X i j = 1}
Escoger (i*,j*) del conjunto:
B = {(i,j) | C i j = V y X i j = 1}
Encontrar un ciclo que empiece y termine en (i*,j*)
existe el ciclo
SI
FIN
NO
Invertir las asignaciones

2.3. Algoritmo de Garfinkel.[7]

2.3.1. Planteamiento.

J. Edmonds y D.R. Fulkerson[8] dan la etiqueta "threshold" (umbral) a una clase de algoritmos para el problema de asignación cuello de botella, en los cuales se toma un valor inicial V y se incrementa progresivamente probando para cada valor si existe una solución cuyo resultado sea V. Basándose en esto, Garfinkel[7] da un algoritmo para matrices cuadradas el cual termina en a lo más n^2 iteraciones, ya que ninguna celda es repetida como el origen del ciclo buscado en el caso de Gross.

Garfinkel propone 'limpiar' inicialmente la matriz de costos y sobre las celdas que queden (las cuales llamaremos celdas admisibles), plantea una red de flujo y aplica el algoritmo de Ford-Fulkerson de etiquetas para obtener el flujo máximo.

El algoritmo es el siguiente:

Paso 1. Sea : $V = V_0 = \max \{t_1,...,t_n, w_1,..., w_n\}$

con:

$$t_i = \min_{j = 1,..., n} C_{ij} \qquad i = 1,..., n$$

$$w_j = \min_{i=1,\ldots,n} C_{ij} \qquad j = 1,\ldots,n$$

Paso 2. Sean todas las celdas (i,j) con $C_{ij} \leq V$ admisibles. Formar una red R (Ver Figura 2.3.1.1) con un origen ***s*** y un destino ***t*** imaginarios, un nodo r_i correspondiente a cada renglón i y un nodo c_j para cad columna j. Trazar arcos con capacidad 1 de ***s*** a r_i y de c_j a ***t*** (i,j = 1,2,...,n) y con capacidad infinita de r_i a c_j correspondientes a celdas (i,j) admisibles.

Aplicar el algoritmo de etiquetas de Ford-Fulkerson[6] para obtener el flujo máximo de los orígenes (renglones) a los destinos (columnas).

Si el valor del flujo máximo es n entonces termina. La asignación corresponde a las unidades de flujo de los renglones a las columnas. El costo óptimo es V.

Si el flujo es menor a n ir al paso 3.

Paso 3. Sea $V = \min \{ C_{ij} ; i$ está etiquetada y j no lo está $\}$.

Deje las etiquetas de la iteración anterior intactas y vaya al paso 2.

El algoritmo de etiquetas de Ford-Fulkerson[6] para problemas de flujo máximo es el siguiente:

Denotaremos al flujo de **x** a **y** como **f(x,y)**, y a la capacidad de **x** a **y** como **c(x,y)**.

RED R

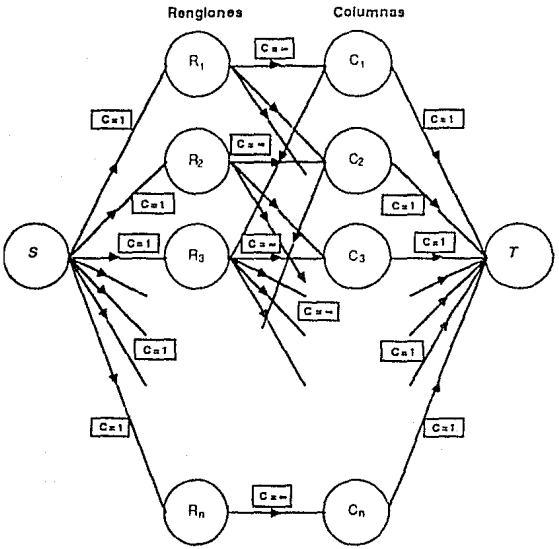


Figura 2.3.1.1

El algoritmo debe empezar con flujo cero.

Durante la rutina A un nodo puede estar en cualquiera de los siguientes estados:

1. No Etiquetado
2. Etiquetado y no Registrado
3. Etiquetado y Registrado

Rutina A *(Proceso de Etiquetación)*

Rutina A: El origen s recibe la etiqueta $(-, \partial(s) = \infty)$, y está en el estado 3; todos los demás nodos están en el estado 1.

En general, tomar cualquier nodo x en el estado 2 ó 3, supongamos que está etiquetado $(z^{\pm}, \partial(x))$, para todos los nodos y que no están etiquetados (estado 1) y tales que $f(x,y) < c(x,y)$ asignar la etiqueta $(x^{+}, \partial(y))$ donde:

$$\partial(y) = \min \{ \partial(x) , c(x,y) - f(x,y) \}$$

Tales nodos y están ahora en el estado 3.

Para todos los nodos y que estén ahora en el estado 1 y tales que $f(y,x) > 0$ asignar la etiqueta $(x^{-}, \partial(y))$ donde:

$$\partial(y) = \min \{ \partial(x) , f(y,x) \}$$

Tales nodos y están en el estado 3 y x en el estado 2.

Repetir el paso general hasta que el destino t está en el estado 3 o hasta que no puedan ser asignadas más etiquetas y el destino t está en el estado 1.

En el primer caso vaya a la rutina B.

En el segundo termina.

Rutina B. *(Cambio de Flujo)*

Rutina B: El destino t ha sido etiquetado $(y^{\pm}, \partial(t))$. Por lo que tenemos los siguientes casos:

Etiqueta	Reemplazar $f(y,t)$ por:
$(y^{+}, \partial(t))$	$f(y,t) + \partial(t)$
$(y^{-}, \partial(t))$	$f(y,t) - \partial(t)$

Ir al nodo **y**, si **y** está etiquetado $(x^{\pm}, \partial(y))$, examinar los siguientes casos:

Etiqueta	Reemplazar $f(y,t)$ por:
$(x^{+}, \partial(y))$	$f(x,y) + \partial(t)$
$(x^{-}, \partial(y))$	$f(x,y) - \partial(t)$

Ir al nodo **x** y repetir el análisis de etiquetación del nodo hasta alcanzar el origen ***s***.

Descartar las antiguas etiquetas y vaya a la rutina A.

No es difícil mostrar que el algoritmo de Garfinkel resuelve el problema de asignación cuello de botella. Claramente V_0 da un límite menor en el valor óptimo de **z**, entonces el uso de V_0 como V en el paso 2 es válido. Si un flujo de **n** unidades no es posible en el paso 2, entonces el paso 3 da la celda de costo mínimo admisible que permitirá próximas etiquetaciones. Note que el paso 3 siempre da un incremento en V, ya que un elemento $C_{i'j'} \leq V$ con **i'** etiquetado y **j'** no etiquetado permitirá a **j'** ser etiquetado desde **i'**. De este modo el máximo número de posibles ocurrencias en el paso 3 es el número de entradas distintas en la matriz de costos.

Este algoritmo es muy similar al algoritmo estándar para el problema clásico de asignación. Sin embargo, en este método las celdas admisibles siguen siéndolo y generalmente se utilizan mas números que en el algoritmo clásico.

2.3.2 Ejemplo de Aplicación.

Supongamos que se tiene la siguiente matriz de costos:

	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Paso 1. Encontrar : $V = V_0 = \text{máx}\ \{t_1,...,t_n, w_1,..., w_n\}$

con:

$$t_i = \min_{j = 1,..., n} C_{ij} \qquad i = 1,..., n$$

$t_1 = \min\ \{15,12,21,16,3\} = 3$

$t_2 = \min\ \{6,7,5,25,8\} = 5$

$t_3 = \min\ \{3,6,22,4,15\} = 3$

$t_4 = \min\ \{3,6,22,4,15\} = 3$

$t_5 = \min\ \{18,3,8,15,2\} = 2$

$$w_j = \min_{i = 1,..., n} C_{ij} \qquad j = 1,..., n$$

$w_1 = \min\ \{15,6,23,3,18\} = 3$

w_2 = min {12,7,18,6,3} = 3

w_3 = min {21,5,20,22,8} = 5

w_4 = min {16,25,14,4,15} = 4

w_5 = min {3,8,3,15,2} = 2

$\Rightarrow V = V_0$ = máx [máx {3,5,3,3,2} , máx {3,3,5,4,2}]

$\therefore V = V_0$ = máx {5,5} = 5

Parso 2. Las celdas (i,j) con $C_{ij} \leq 5 = V$ son admisibles y son las siguientes:

	1	2	3	4	5
1	-	-	-	-	3
2	-	-	5	-	-
3	-	-	-	-	3
4	3	-	-	4	-
5	-	3	-	-	2

Tomando éstas formamos la red R como se muestra en la figura 2.3.2.1 y aplicamos el algoritmo de etiquetas de Ford-Fulkerson. El valor del flujo máximo es 4 quedando R_3 y R_1 etiquetados y C_1, C_2, C_3, C_4, no etiquetadas, por lo que vamos al paso 3.

Paso 3. V = min { C_{ij} ; i está etiquetada y j no lo está } = min {C_{31}, C_{32}, C_{33}, C_{34}, C_{11}, C_{12}, C_{13}, C_{14},} = min {23, 18, 20, 14, 15, 12, 21, 16} = 12. Dejamos las etiquetas iguales y vamos al paso 2.

RED R

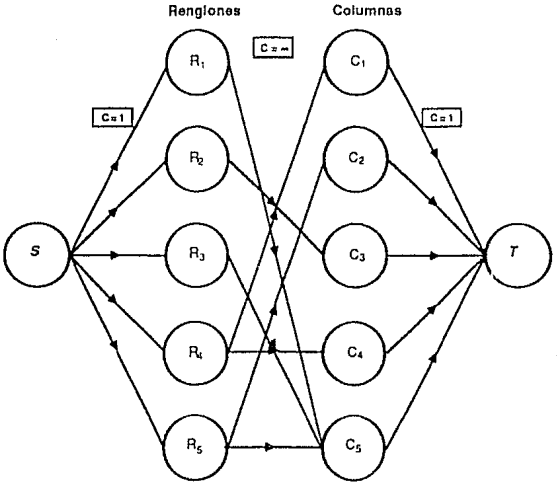


Figura 2.3.2.1

Paso 2. Las celdas (i,j) con $C_{ij} \leq 12 = V$ son admisibles y son las siguientes:

	1	2	3	4	5
1	-	12	-	-	3
2	6	7	5	-	8
3	-	-	-	-	3
4	3	6	-	4	-
5	-	3	8	-	2

Tomando éstas transformamos la red **R** y aplicamos el algoritmo de etiquetas de Ford-Fulkerson. El valor del flujo máximo es 5 = **n**, entonces el algoritmo termina. La

asignación óptima correspondiente se muestra en la figura 2.3.2.2. Y el costo óptimo es z = 12 = V.

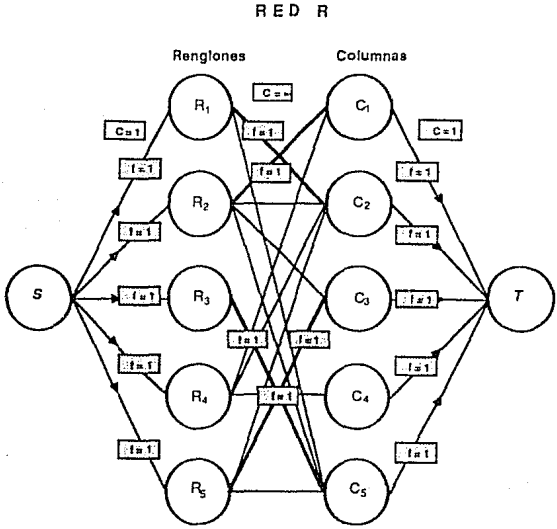
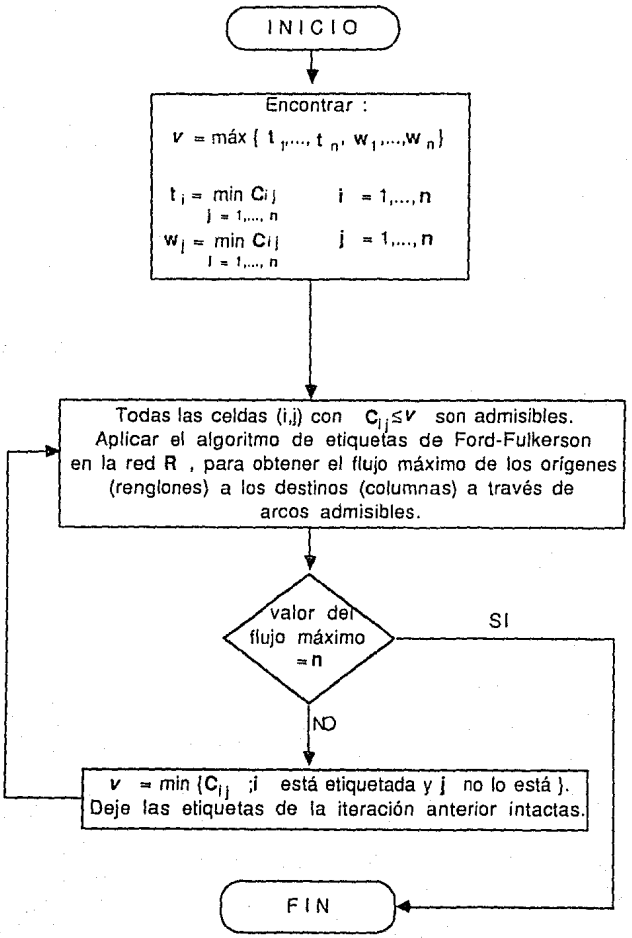


Figura 2.3.2.2

2.3.3. Diagrama de Flujo


INICIO
Encontrar :
v = máx { t 1,..., t n, w 1,...,w n}
t i = min Ci j
j = 1,..., n
i = 1,..., n
w j = min Ci j
I = 1,..., n
j = 1,..., n
Todas las celdas (i,j) con C i j ≤ v son admisibles.
Aplicar el algoritmo de etiquetas de Ford-Fulkerson
en la red R , para obtener el flujo máximo de los orígenes
(renglones) a los destinos (columnas) a través de
arcos admisibles.
valor del
flujo máximo
= n
SI
NO
v = min {C i j ;i está etiquetada y j no lo está }.
Deje las etiquetas de la iteración anterior intactas.
FIN

2.4 Agortimo de Derigs - Zimmermann.[9]

2.4.1 Planteamiento

Este algoritmo empieza con una asignación parcial heurística y se buscan 'cortos caminos de aumento' (shortest augmenting paths) con una modificación del algoritmo de Dijkstra para rutas más cortas en redes[10]. Puede ser aplicado a problemas de asignación con matrices no cuadradas ($m \leq n$).

El algoritmo es el siguiente:

Denotaremos al problema k como P_k, éste es resuelto para $k = 1,2,...,m$.

Sea S(k) el conjunto de todas las asignaciones parciales $\beta : K \rightarrow m$ para $K = \{ 1,2,...,k \}$, entonces el problema P_k se define como:

$$z_k = \min_{\beta \in S(k)} \max_{i \in K} C_{i\,\beta(i)}$$

La solución de P_1 es obvia. Empezando con una solución β para P_k con $k \in \{ 1,2,...,m-1 \}$ la solución ∂ de P_{k+1} se construye por medio de 'caminos de aumento'. Para simplificar la notación supongamos, sin pérdida de generalidad, que β es la diagonal en K, esto es

$$ß(i) = i \qquad \text{para todo } i \in K$$

Esto puede darse permutando renglones y columnas en la matriz de costos C. Sea C_k la submatriz de C que consta de los primeros k renglones de C. Entonces C_{k+1} se particiona así:

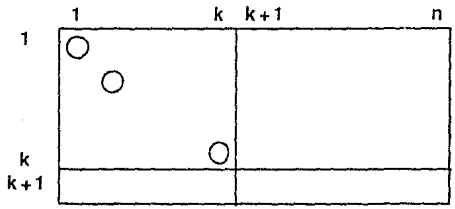


Figura 2.4.1.1

Los elementos de la diagonal en C_{k+1} corresponden a la solución ß. A la secuencia $\Delta ß$ de entradas mutuamente distintas (i,j) en C_{k+1} se le llama un 'camino de aumento' de longitud **2r+1** si tiene la siguiente forma:

$$(k+1, j_1),\ (j_1, j_1),\ (j_1, j_2), (j_2, j_2), \ldots, (j_r, j_r), (j_r, j_r + 1)$$

con : $1 \le j_i \le k$ para todo $i = 1,2,\ldots,r$ y $j_r + 1 > j_r$

En particular, las entradas (k+1,j) con $j > k$ son 'caminos de aumento' de longitud 1.

La asignación parcial $ß \in S(k)$ puede ser transformada por medio de $\Delta ß$ a $\partial\ S(k+1)$ $ß + \Delta ß = \partial$ haciendo:

$$\partial\ (i) = i \qquad \forall\ i \in K\backslash\{j_1, j_2, \ldots, j_r\}$$

$$\partial\ (j+1) = j_i + 1 \qquad \forall\ i = 1,2,\ldots,r$$

$$\partial\ (k+1) = j_1$$

La asignación parcial $\beta + \Delta\beta = \partial\ S(k+1)$ es una solución de P_{k+1} con valor de la función objetivo:

$$w\ (\Delta\beta) = \max_{1 \le i \le k+1} C_{i\partial(i)}$$

Sea D_k el conjunto de todos los 'caminos de aumento' con respecto a una solución óptima β de P_k.

Teorema[10]. Sea $\beta \in S(k)$ una solución óptima de P_k. Si

$$w\ (\Delta\beta) = \min_{\Delta\beta \in S(k)} w\ (\Delta\beta)$$

existe para $\Delta\beta \in D_k$ entonces $\beta + \Delta\beta$ es una solución óptima para P_{k+1}.

Con este teorema el método propuesto encuentra una solución óptima en m pasos. Aún más, con esto se prueba que puede ser aplicado a problemas de asignación con matrices no cuadradas tales que $m \le n$.

El problema que ahora se presenta es encontrar el camino más corto. En el algoritmo los autores proponen que la solución se encuentra con una modificación del algoritmo de Dijkstra[10].

Derigs - Zimmermann hacen notar que el tiempo de corrida del algoritmo puede decrecer con la ayuda de un método heurístico para obtener una asignación inicial parcial. Este consta de dos partes, la primera es encontrar un valor menor z para el valor óptimo de z , se determina por:

$$z = \max \{\max_{i \in m} s_i , \max_{j \in n} t_j\}$$

con:

$$s_i = \min_{j \in n} C_{ij} \qquad \forall\, i \in m$$

$$t_j = \min_{i \in m} C_{ij} \qquad \forall\, j \in n$$

En la segunda parte se toma una asignación parcial $\beta : K \rightarrow n$ en la cual:

$$C_{i,\beta(i)} \leq z \qquad \forall\, i \in K \qquad (1)$$

El número de problemas P_k que faltan por resolver es $n - |K|$. Sin embargo, es importante determinar una asignación parcial inicial con la mayor cardinalidad posible $|K|$.

Un método simple para determinar dicha asignación parcial inicial requerida a continuación se describe:

Encontrar : $\quad a(i) = |\{j \in n ; C_{ij} \leq z\}| \quad \forall\, i \in m \qquad (2)$

Después de permutar renglones se puede suponer que:

$$i < j \Rightarrow a(i) < a(j) \quad \forall\, i \in m, j \in n$$

Entonces el siguiente algoritmo escoge una asignación parcial ß que cumple con (1):

Nota: los renglones de la matriz C deben estar en orden no descendiente de acuerdo a (2).

Paso 1 . $J = n$, $\mu = \emptyset$, $i = 1$

Paso 2. Si existe algún elemento $j \in J$ con $C_{ij} \leq z$ definir :

$J = J \backslash \{j\}$, $\mu = \mu \cup \{i\}$, $ß(i) = j$

En el caso contrario ir al paso 3.

Paso 3. Si $i = n$, $K = |\mu|$, alto.

De lo contrario $i = i+1$ e ir al paso 2.

Si consideramos los renglones en una secuencia con un número no decreciente de elementos menores o iguales a z , entonces la probabilidad de que encontremos una asignación parcial inicial con número suficientemente grande K crece. Derigs - Zimmermann dicen que utilizando el algoritmo anteriormente descrito el dominio K contiene renglones en un 90 a 95 % del total, esto es:

$$0.9 \leq K \div n \leq 0.95$$

Así el tiempo total de corrida es menor.

2.4.2 Ejemplo de Aplicación.

Supongamos que se tiene la siguiente matriz de costos:

	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Asignación Heurística.

Parte I.

Paso 1. Encontrar un valor menor z para el valor óptimo de z. $n = m = \{1,2,3,4,5\}$.

$$z = \max \{\max_{i \in m} s_i , \max_{j \in n} t_j\}$$

con: $s_i = \min_{j \in n} C_{ij} \quad \forall i \in m$

$s_1 = \min \{15,12,21,16,3\} = 3$

$s_2 = \min \{6,7,5,25,8\} = 5$

$s_3 = \min \{3,6,22,4,15\} = 3$

$s_4 = \min \{3,6,22,4,15\} = 3$

$s_5 = \min \{18,3,8,15,2\} = 2$

$t_j = \min_{i \in m} C_{ij} \qquad \forall\, j \in n$

$t_1 = \min \{15,6,23,3,18\} = 3$

$t_2 = \min \{12,7,18,6,3\} = 3$

$t_3 = \min \{21,5,20,22,8\} = 5$

$t_4 = \min \{16,25,14,4,15\} = 4$

$t_5 = \min \{3,8,3,15,2\} = 2$

$\Rightarrow z = \text{máx} \{\text{máx} \{3,5,3,3,2\} , \text{máx} \{3,3,5,4,2\}\}$

$\therefore z = \text{máx} \{5,5\} = 5$

Parte 2.

Paso 1. Encontrar : $a(i) = | \{j \in n ; C_{ij} \leq z \} | \quad \forall\, i \in m$

$a(1) = | \{j_5\} | = 1$

Ya que en el renglón 1 de la matriz C el elemento C_{15} cumple con $C_{1j} \leq z = 5$

$a(2) = | \{j_3\} | = 1$

$a(3) = | \{j_5\} | = 1$

$a(4) = | \{j_1, j_4\} | = 2$

$a(5) = | \{j_2, j_5\} | = 2$

Paso 2. En este caso específico no es necesario permutar renglones, ya que se cumple que $i < j \Rightarrow a(i) < a(j)\ \forall\, i \in m, j \in n$, $\therefore$ la matriz queda igual.

Asignación Parcial Heurística.

Paso 1. $J = m = \{1,2,3,4,5\}$, $\mu = \emptyset$, $i = 1$

Paso 2. $5 \in J$ y $C_{15} = 3 \leq z = 5$, $\Rightarrow J = \{1,2,3,4\}$, $\mu = \mu \cup \{5\} = \{5\}$, $\beta(1)=5$

Paso 3. $i \neq 5$, $\therefore i = 2$

Paso 2. $3 \in J$ y $C_{23} = 5 \leq 5$, $\Rightarrow J = \{1,2,4\}$, $\mu = \{5,3\}$, $\beta(2)=3$

Paso 3. $i \neq 5$, $\therefore i = 3$

Paso 2. $5 \notin J$

Paso 3. $i \neq 5$, $\therefore i = 4$

Paso 2. $1 \in J$ y $C_{41} = 3 \leq 5$, $\Rightarrow J = \{2,4\}$, $\mu = \{5,3,1\}$, $\beta(4)=1$

Paso 3. $i \neq 5$, $\therefore i = 5$

Paso 2. $2 \in J$ y $C_{22} = 3 \leq 5$, $\Rightarrow J = \{4\}$, $\mu = \{5,3,1,2\}$, $\beta(5)=2$

Paso 3. $i = 5$, $K = |\mu| = 4$, $\therefore$ Fin de la primera parte.

El valor de **z** en esta asignación es 5. Como $K = 4$, procederemos a la búsqueda del 'camino de aumento'.

Permuntando renglones y columnas obtenemos la asignación en la diagonal como sigue:

	5	3	1	2	4
1	**3**	21	15	12	16
2	8	**5**	6	7	25
4	15	22	**3**	6	4
5	2	8	18	**3**	15
3	3	20	23	18	14

En la cual tenemos definido el problema P_4. En [9] se sugiere hacer una modificación al algoritmo de Dijkstra para encontrar mas rápido el camino de aumento, pero este caso se analizará más adelante, ahora se presentará con el algoritmo tal y como está planteado por los autores.

	5	3	1	2	4
1	3	21	15	12	16
2	8	5	6	7	25
4	15	22	3	6	4
5	2	8	18	3	15
3	3	20	23	18	14

Este 'camino de aumento' es de la forma:

(3,5),(1,5),(1,2),(5,2),(5,3),(2,3),(2,1),(4,1),(4,4)

pero reenumerando tanto renglones como columnas tenemos:

(5,1),(1,1),(1,4),(4,4),(4,2),(2,2),(2,3),(3,3),(3,5)

el cual es un 'camino de aumento de longitud 9 (r=4). La asignación parcial a través del camino de aumento es una solución para P_5 con valor de la función objetivo $z = 12$, la cual constituye una del conjunto de posibles asignaciones D_4 , para este caso en especial, se eligió la óptima.

Es fácil ver que existen muchos y diversos caminos de aumento en el ejemplo planteado anteriormente, pero, eligiendo las celdas adecuadas, se puede encontrar rápidamente el óptimo. Obteniendo así la asignación óptima como se muestra a continuación en la figura 2.4.2.1.

	5	3	1	2	4
1	3	21	15	12	16
2	8	5	6	7	25
4	15	22	3	6	4
5	2	8	18	3	15
3	3	20	23	18	14

Figura 2.4.2.1

2.4.3 Modificación del Algoritmo de Dijkstra[10].

El algoritmo de Dijkstra para la ruta más corta entre el origen y el destino con distancias no negativas es el que sigue:

Inicialización. Sea $L(x_i)$ la etiqueta del vértice x_i, y sea $\Gamma^+(x_i)$ el conjunto de nodos a los cuales se llega desde x_i.

En lo subsiguiente llamaremos s al origen y t al destino.

Tomar: $L(s) = 0$ y permanente

$L(x_i) = \infty \quad \forall\, x_i \neq s$ y temporal

Sea $p = s$.

Paso 1. Actualización de etiquetas.

$\forall\, x_i \in \Gamma^+(p)$ con etiqueta temporal actualizar etiquetas como sigue:

$$L(x_i) = \min \{L(x_i), L(p) + C(p,x_i)\} \qquad (1)$$

Paso 2. Etiquetación Permanente.

Sea x_i^* tal que $L(x_i^*) = \min \{L(x_i) \mid L(x_i)$ es temporal$\}$

hacer $L(x_i^*)$ permanente y $p = x_i^*$.

Paso 3. Si $p = t$, $L(p)$ es la longitud de la ruta. Fin.

Si $p \neq t$, ir al Paso 1.

Es evidente que este algoritmo funciona para el problema clásico de asignación (min-sum), pero para plantearlo con el 'cuello de botella' la modificación que tenemos que hacer está en el Paso 1, quedando éste como sigue:

Paso 1. Actualización de etiquetas.

Para todo $x_i \in \Gamma^+(p)$ con etiqueta temporal actualizar etiquetas como sigue:

$$L(x_i) = \min [L(x_i), \max \{L(p), C(p,x_i)\}] \qquad (2)$$

Para poder aplicar el algoritmo de Dijkstra, una vez encontrada la solución al problema P_k, se plantea la red que se muestra en la figura 2.4.3.1, en la cual se tiene un origen s y un destino t imaginarios, se toman los nodos x_i para $i = 1,2,\ldots,k$, ya que para $k+1$ es un 'camino de aumento' de longitud 1 y sólo se toma un arco directo que conduce del origen s al destino t.

RED PARA APLICAR DIJKSTRA

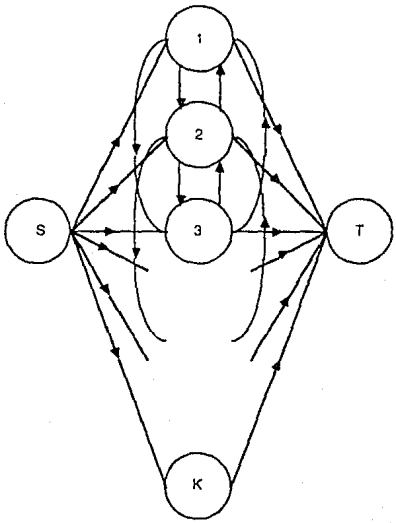


Figura 2.4.3.1

2.4.4 Ejemplo de Aplicación Aplicando Dijkstra.

Retomando el ejemplo planteado en la parte 2.3.2, una vez aplicado el método heurístico encontramos una solución para P_4, obteniendo así la siguiente matriz, donde la asignación está en la diagonal:

	5	3	1	2	4
1	3	21	15	12	16
2	8	5	6	7	25
4	15	22	3	6	4
5	2	8	18	3	15
3	3	20	23	18	14

Para tener una visión clara del algoritmo de Dijkstra planteamos la red sugerida para este caso en la figura 2.4.4.1.

Inicialización.

xi	L(xi)	Estado
s	0	Permanente
1	∞	Temporal
2	∞	Temporal
3	∞	Temporal
4	∞	Temporal
t	∞	Temporal

Paso 1. Actualización de etiquetas.

$\Gamma^+(s) = \{1,2,3,4,t\}$, tomando la fórmula propuesta en 2.3 (2)

$$L(x_i) = \min [L(x_i), \text{máx}\{L(p), C(p,x_i)\}]$$

Obtenemos el siguiente cuadro de etiquetas:

xi	L(xi)	Estado
s	0	Permanente
1	3	Temporal
2	20	Temporal
3	23	Temporal
4	18	Temporal
t	14	Temporal

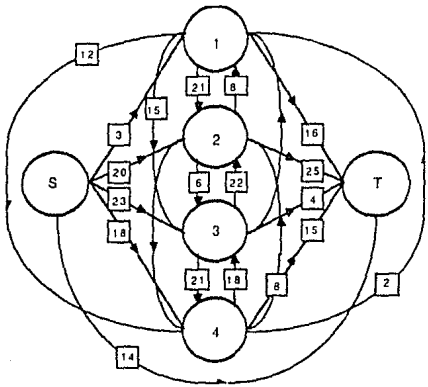


Figura 2.4.4.1

Paso 2. Etiquetación permanente.

Utilizando 2.3. (1) obtenemos como x_i^* el nodo 1, que viene representando la entrada C_{35}. $p = 1$.

Paso 3. $p \neq t$, $\therefore$ vamos al Paso 1

Paso 1. Actualización de etiquetas.

$\Gamma^+(1) = \{2,3,4,t\}$, aplicando la fórmula 2.3. (2) obtenemos:

xi	L(xi)	Estado
s	0	Permanente
1	3	Permanente
2	20	Temporal
3	15	Temporal
4	12	Temporal
t	16	Temporal

Paso 2. Etiquetación permanente.

Obtenemos como x_i^* el nodo 4, que viene representando la entrada C_{12}. $p = 4$.

Paso 3. $p \neq t$, ∴ vamos al Paso 1

Paso 1. Actualización de etiquetas.

$\Gamma^+(4) = \{2,3,t\}$, aplicando la fórmula 2.3. (2) obtenemos:

xi	L(xi)	Estado
s	0	Permanente
1	3	Permanente
2	12	Temporal
3	12	Temporal
4	12	Permanente
t	15	Temporal

Paso 2. Etiquetación permanente.

Obtenemos como x_i^* el nodo 2, que viene representando la entrada C_{53}. $p = 2$.

Paso 3. $p \neq t$, ∴ vamos al Paso 1

Paso 1. Actualización de etiquetas.

$\Gamma^+(2) = \{3,t\}$, aplicando la fórmula 2.3. (2) obtenemos:

xi	L(xi)	Estado
s	0	Permanente
1	3	Permanente
2	12	Permanente
3	12	Temporal
4	12	Permanente
t	15	Temporal

Paso 2. Etiquetación permanente.

Obtenemos como x_i^* el nodo 3, que viene representando la

entrada C_{21}. $p = 3$.

Paso 3. $p \neq t$, ∴ vamos al Paso 1

Paso 1. Actualización de etiquetas.

$\Gamma^+(3) = \{t\}$, aplicando la fórmula 2.3. (2) obtenemos:

xi	L(xi)	Estado
s	0	Permanente
1	3	Permanente
2	12	Permanente
3	12	Permanente
4	12	Permanente
t	12	Temporal

Paso 2. Etiquetación permanente.

Obtenemos como x_i^* el nodo t, que viene representando la entrada C_{44}. $p = t$.

Paso 3. $p = t$, ∴ FIN.

Visto el algoritmo de Dijkstra en forma gráfica obtenemos la red como se muestra en la figura 2.4.4.2, en donde las lineas gruesas dan el 'camino de aumento' óptimo que obtuvimos en el algoritmo de Derigs - Zimmermann. Representado matricialmente obtenemos la asignación óptima:

	5	3	1	2	4
1	3	21	15	**12**	16
2	8	5	**6**	7	25
4	15	22	3	6	**4**
5	2	**8**	18	3	15
3	**3**	20	23	18	14

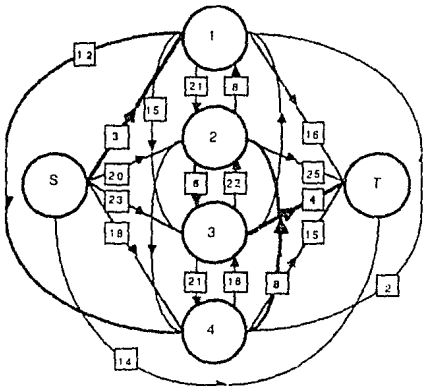
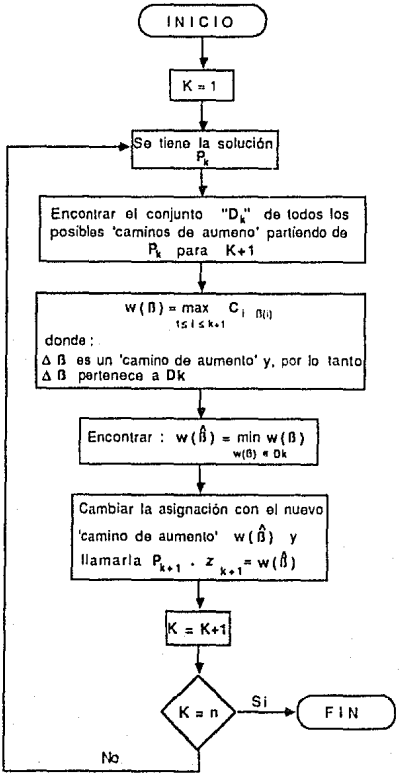


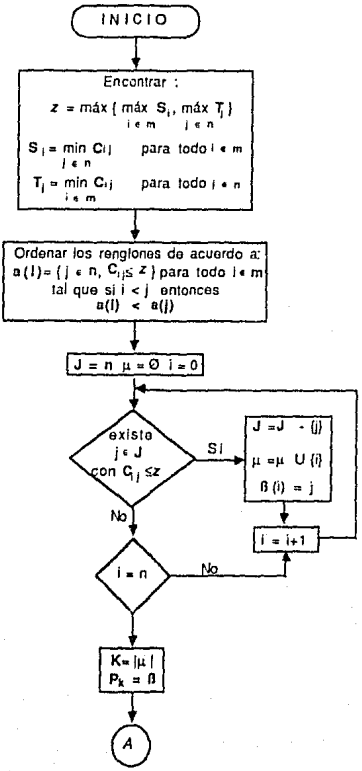
Figura 2.4.4.2

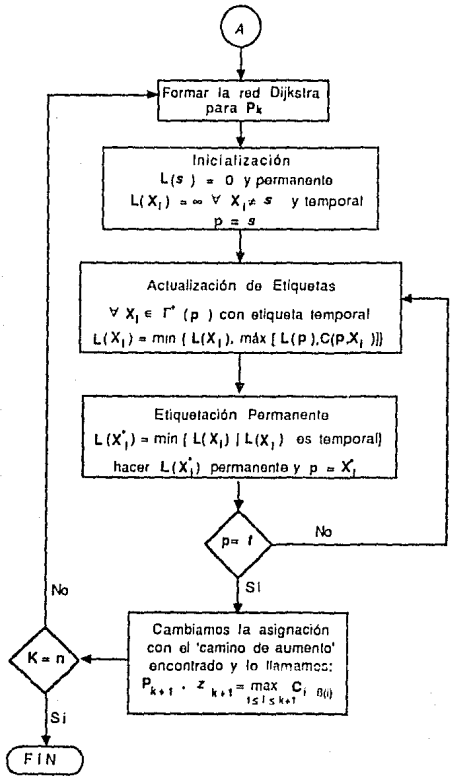
2.4.5. Diagrama de Flujo

2.4.5.1 Problema General



2.4.5.2 Problema con Asignación Heurística y la modificación Dijkstra




A
Formar la red Dijkstra para Pk
Inicialización
L(s) = 0 y permanente
L(Xi) = ∞ ∀ Xi ≠ s y temporal
p = s
Actualización de Etiquetas
∀ Xi ∈ Γ+ (p) con etiqueta temporal
L(Xi) = min { L(Xi), máx [L(p), C(p,Xi)]}
Etiquetación Permanente
L(X*i) = min { L(Xi) | L(Xi) es temporal}
hacer L(X*i) permanente y p = X*i
p= t
No
Si
No
Cambiamos la asignación con el 'camino de aumento' encontrado y lo llamamos:
Pk+1 , zk+1 = max Ci β(i)
1≤i≤k+1
K = n
Si
FIN

2.5 Algoritmo de Romero.[11]

2.5.1. Planteamiento.

Este algoritmo está basado en la ordenación de una matriz $C_{m\times n}$ $(m \leq n)$ mediante permutaciones de renglones y/o columnas, de tal forma que al terminar, la asignación óptima se encuentra en la diagonal principal.

El algoritmo es el siguiente:

Se toma como asignación inicial la diagonal principal y sea $P^* = \max \{C_{ij} \mid C_{ij}$ está en la diagonal principal$\}$ con coordenadas (i^*,j^*).

A lo largo del algoritmo y dependiendo de (i^*,j^*) nos iremos refiriendo a diferentes submatrices de C, las cuales llamaremos zonas. Una zona Z puede constar de uno o más renglones; para ambos casos definiremos:

I_z = conjunto de renglones de Z

J_z = conjunto de columnas de Z

W_Z = $\min \{ C_{ij} \mid C_{ij} \in$ zona Z $\}$ con coordenadas (i_z,j_z)

En el caso en que $|I_z| > 1$ definiremos:

$\bar{Z}$ = $\{ i \mid i \in I_z$ y $\min \{ C_{ij} \} \geq P^* \}$

$$\bar{Z}' = \{ i \mid i \in I_z \text{ y } \min \{ C_{ij} \} < P^* \}$$

y para efectos de notación denotaremos:

$$|\bar{Z}| = k \qquad \text{y} \qquad |\bar{Z}'| = k'$$

Es decir, en la zona Z encontraremos renglones en los que todos los elementos son mayores o iguales que P^* y/o renglones en los que al menos un elemento es menor que P^*.

Cuando tengamos una zona Z en la que en que $|I_z| > 1$, $k_z \neq 0$ y $k_z' \neq 0$, se procederá a ordenar Z de tal forma que:

En los primeros k renglones de Z tengamos todos los elementos de $\bar{Z}$ y a partir de k+1, los elementos de $\bar{Z}'$ (1)

La figura 2.5.1.1 ilustra este caso, en donde se muestra primero la zona Z como se puede encontrar, y después ésta ordenada.

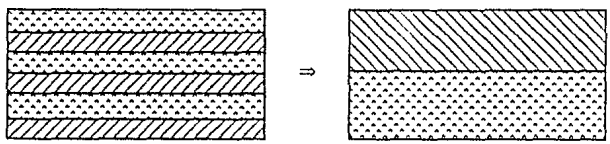


Figura 2.5.1

Llamaremos *pivoteo de orden* λ a la permutación de $\lambda+1$ columnas de **C**. Se dice de orden λ porque, además, en este pivoteo siempre se permuta

también j^*. En un pivoteo de orden 1 sólo se intercambian las columnas j_z y j^*. Supongamos que tenemos las columnas e y f para hacer un pivoteo de orden 2, de tal forma que $e < j^* < f$, entonces permutamos las columnas de tal forma que obtengamos $f < e < j^*$. Si el pivoteo es de orden 4, con $e < j^* < f < g < h$ entonces permutamos las columnas para obtener $f < e < g < h < j^*$.

Los pasos a seguir, tomando en cuenta lo anterior son:

Paso 1. Tomar la asignación inicial en la diagonal principal.

Paso 2. Encontrar $P^* = \text{máx}\{C_{ij} \mid C_{ij}$ está en la diagonal principal$\}$ con coordenadas (i^*, j^*).

Paso 3. zona $A = \{C_{ij} \mid i = i^*, j = m+1, \ldots, n\}$.

Si $W_A < P^*$ hacer un pivoteo de orden 1 con j_A e ir al Paso 2.

Paso 4. Ordenar los elementos de la columna j^*, sin alterar la diagonal principal (es decir, permutando renglones y columnas simultáneamente), de tal forma que:

$$C_{ij^*} \geq P^* \qquad \forall\ i = 1, \ldots, i^* - 1$$

$$C_{ij^*} < P^* \qquad \forall\ i = i^* + 1, \ldots, m$$

Paso 5. zona $B = \{C_{ij} \mid i = i^*, j = j^* + 1, \ldots, m\}$.

Si $W_B < P^*$ hacer un pivoteo de orden 1 con j_B e ir al Paso 2.

Paso 6. zona $C = \{C_{ij} \mid i = 1, \ldots, i^* - 1, j = j^* + 1, \ldots, n\}$.

Si $W_C \geq P^*$ ir al Paso 10, de lo contrario ordenar la zona C sin alterar la diagonal principal de acuerdo a (1).

Paso 7. zona D = $\{C_{ij} \mid i = i^*, j = k,\ldots,j^*-1\}$.

Si $W_D < P^*$ hacer un pivoteo de orden 2 (j_d, j_c, j^*) e ir al Paso 2, en caso contrario $\lambda = 1$.

Paso 8. Permutar simultáneamente renglones i^* por **k** y columnas j^* por **k**.

Si $(i^*, j^*) = (1,1)$ ir al Paso 10.

Paso 9. zona E = $\{C_{ij} \mid i = 1,\ldots,i^*-1, j = j^*+1,\ldots,k-1\}$.

Si $W_E \geq P^*$ ir al Paso 10, de lo contrario ordenar la zona E sin alterar la diagonal principal de acuerdo a (1) y definir la correspondiente zona D como se hizo en el Paso 7, si :

$W_D \geq P^* \Rightarrow \lambda = \lambda + 1$ e ir al Paso 8

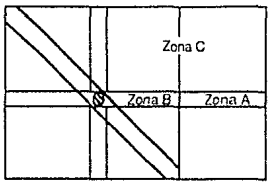
$W_D < P^* \Rightarrow$ hacer un pivoteo de orden λ e ir al Paso 2.

Paso 10. Alto. La asignación óptima está en la diagonal principal de la matriz.

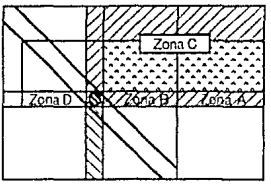
El paso clave está en la forma de definir las zonas y hacer los pivoteos. La forma en que se van definiendo las zonas se puede apreciar con mayor claridad en la figura 2.5.1.2 a), b), c).

La forma en que pueden hacerse los pivoteos se ilustran en la figura 2.5.1.3 a) y b), en donde se muestra un pivoteo de orden 2 y un pivoteo de

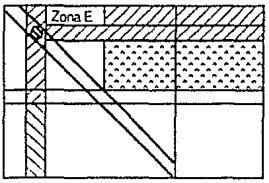
orden 3 respectivamente.



a)



b)



c)

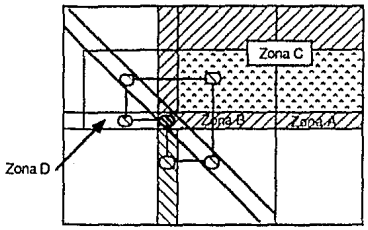
donde:

En cada renglón todos los elementos son mayores o iguales que P^* (en la figura ⊘)

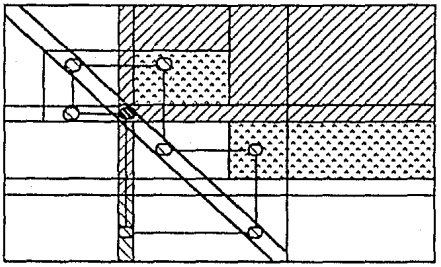
Elementos menores que P^* (en la figura ⊘)

En cada renglón ∃ al menos un elemento menor que P^* (en la figura ⊘)

Figura 2.5.1.2



a) Pivoteo de orden 2



b) Pivoteo de Orden 3

Figura 2.5.1.3

2.5.2 Ejemplo de Aplicación.

Supongamos que se tiene la siguiente matriz de costos **C** con m=n=5:

	1	2	3	4	5
1	15	12	21	16	3
2	6	7	5	25	8
3	23	18	20	14	3
4	3	6	22	4	15
5	18	3	8	15	2

Paso 1. La asignación inicial que está en la diagonal principal corresponde a $C_{ij} = \{15, 7, 20, 4, 2\}$ $i = 1,...,5$.

Paso 2. $P^* = \max\{15,7,20,4,2\} = 20$ con coordenadas **(3,3)**.

Paso 3. Como en este ejemplo de aplicación **m=n=5**, entonces zona A = ø.

Paso 4. Ordenamos los elementos de la columna **3** sin alterar la diagonal principal obteniendo:

	1	4	3	2	5
1	15	16	21	12	3
4	3	4	22	6	15
3	23	14	20	18	3
2	6	25	5	7	8
5	18	15	8	3	2

de tal forma que:

$C_{i3} \geq 20$ $\qquad$ $i = 1,2$

$C_{i3} < 20$ $\qquad$ $i = 4,5$

Paso 5. zona $B = \{C_{ij} \mid i = 3, j = 4,5\} = \{18,3\}$.

$W_B = \min\{ C_{ij} \mid C_{ij} \in \text{zona B} \} = \min\{18,3\} = 3$ con coordenadas **(3,5)**.

$3 < 20 \Rightarrow$ hacemos un pivoteo de orden 1 permutando la columna **5** por la columna **3** quedando:

	1	4	5	2	3
1	15	16	3	12	21
4	3	4	15	6	22
3	23	14	3	18	20
2	6	25	8	7	5
5	18	15	2	3	8

Paso 2. $P^* = \max\{15,4,3,7,8\} = 15$ con coordenadas (1,1).

Paso 3. Como $m=n=5$, entonces zona $A = \emptyset$.

Paso 4. Ordenamos los elementos de la columna 1 sin alterar la diagonal principal obteniendo:

	5	3	1	2	4
3	3	20	23	18	14
5	2	8	18	3	15
1	3	21	15	12	16
2	8	5	6	7	25
4	15	22	3	6	4

de tal forma que:

$C_{i3} \geq 15 \qquad i = 1,2$

$C_{i3} < 15 \qquad i = 4,5$

Paso 5. zona $B = \{C_{ij} \mid i = 3,\ j = 4,5\} = \{12,16\}$.

$W_B = 12$ con coordenadas (3,4).

$12 < 15 \Rightarrow$ hacemos un pivoteo de orden 1 permutando la columna 3 por la columna 4 quedando:

	5	3	2	1	4
3	3	20	18	23	14
5	2	8	3	18	15
1	3	21	12	15	16
2	8	5	7	6	25
4	15	22	6	3	4

Paso 2. $P^* = \max\{3,8,12,6,4\} = 12$ con coordenadas (3,3).

Paso 3. Como m=n=5, entonces zona A = ∅.

Paso 4. Ordenamos los elementos de la columna 3 sin alterar la diagonal principal obteniendo:

	5	2	3	1	4
3	3	18	20	23	14
1	3	12	21	15	16
5	2	3	8	18	15
2	8	7	5	6	25
4	15	6	22	3	4

de tal forma que:

$C_{i2} \geq 12 \qquad i = 1$

$C_{i2} < 15 \qquad i = 3,4,5$

Paso 5. zona $B = \{C_{ij} \mid i = 2, j = 3,4,5\} = \{21,15,16\}$.

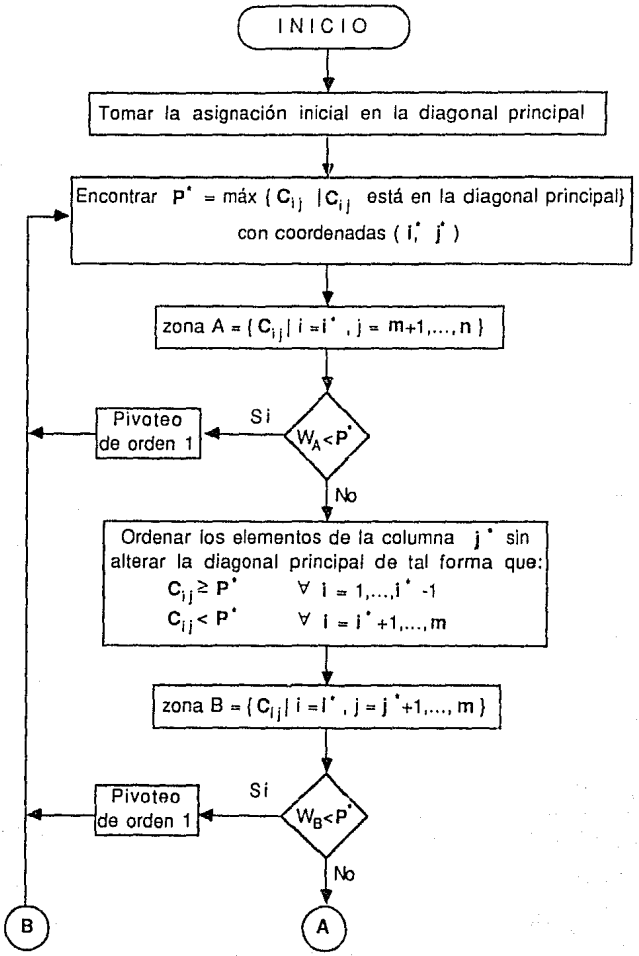
$W_B = 15$ con coordenadas (2,4), y $15 > 12$.

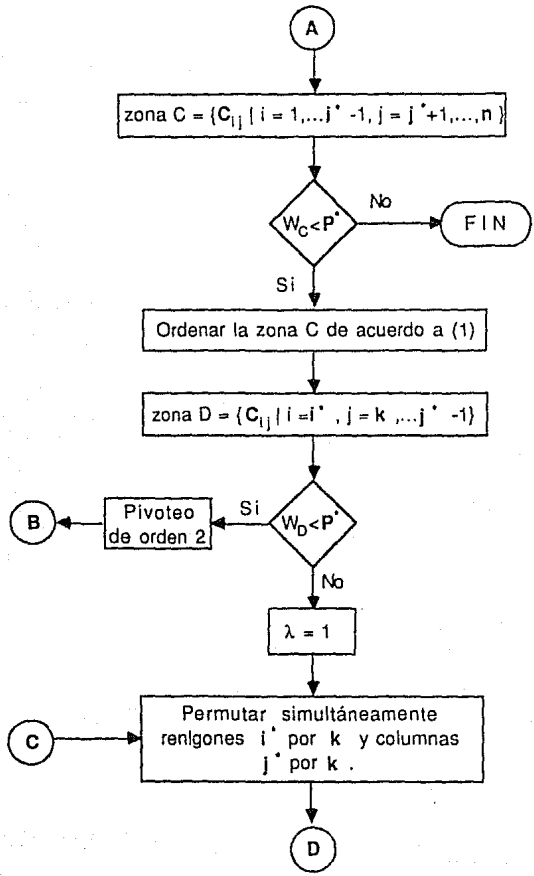
Paso 6. zona $C = \{C_{ij} \mid i = 1, j = 3,4,5\} = \{20,23,14\}$.

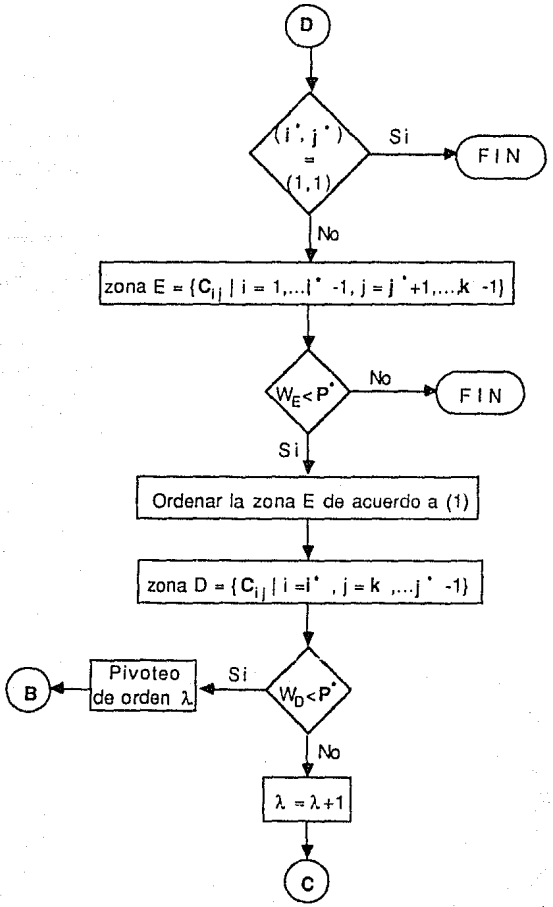
$W_C = 14$ con coordenadas (1,5), y $14 > 12$.

Paso 10. Alto. La asignación óptima está en la diagonal principal de la matriz.

2.5.3 Diagrama de Flujo.


INICIO
Tomar la asignación inicial en la diagonal principal
Encontrar P* = máx { Cij | Cij está en la diagonal principal}
con coordenadas (i*, j*)
zona A = { Cij | i = i* , j = m+1,..., n }
Si
Pivoteo de orden 1
WA<P*
No
Ordenar los elementos de la columna j* sin alterar la diagonal principal de tal forma que:
Cij ≥ P* ∀ i = 1,...,i* -1
Cij < P* ∀ i = i* +1,...,m
zona B = { Cij | i = i* , j = j* +1,..., m }
Si
Pivoteo de orden 1
WB<P*
No
B
A


A
zona C = {C_ij | i = 1,...j* -1, j = j*+1,...,n }
W_C<P*
No
FIN
Si
Ordenar la zona C de acuerdo a (1)
zona D = {C_ij | i =i* , j = k ,...j* -1}
W_D<P*
Si
Pivoteo de orden 2
B
No
λ = 1
C
Permutar simultáneamente renlgones i* por k y columnas j* por k .
D


D
(i*, j*) = (1,1)
Si
FIN
No
zona E = {C_ij | i = 1,...i* -1, j = j*+1,...k -1}
W_E < P*
No
FIN
Si
Ordenar la zona E de acuerdo a (1)
zona D = {C_ij | i = i* , j = k ,...j* -1}
B
Pivoteo de orden λ
Si
W_D < P*
No
λ = λ+1
C

3. EXPERIENCIA COMPUTACIONAL.

En este capítulo se presenta el trabajo experimental. En la sección 3.1 se dan los lineamientos generales y las especificaciones que se siguieron para realizar el trabajo experimental; en la sección 3.2 se lista el código utilizado para cada algoritmo. En la sección 3.3 se presentan los resultados obtenidos, también se incluyen las adaptaciones a los algoritmos en aquellos casos en que éstas podrían mejorar la eficiencia, y por último se hace un análisis.

3.1 Lineamientos Generales.

Para cada algoritmo aplicado se utiliza una tabla en la cual alimentamos los siguientes datos:

n número de renglones y columnas

$\bar{t}$ Promedio de tiempos de corrida

$$\bar{t} = \frac{1}{50} \sum_{i=1}^{50} t_i$$

donde: t_i = tiempo de la i-ésima corrida con **m** y **n** fijos.

$t_{máx}$ $t_{máx} = \max \{t_i\} \; \forall \; i = 1,\ldots,50$

t_{min} $t_{min} = \min \{t_i\} \; \forall \; i = 1,\ldots,50$

Heu	Promedio de $\|\mu\|$ a través del procedimiento heurístico presentado en la sección 2.4.1.
σ	Desviación estándar de los tiempos de corrida

$$\sigma^2 = \sum_{j=1}^{50} (t_j - \bar{t})^2 \; / \; (50-1)$$

Para los algoritmos en donde es posible aplicar el procedimiento a matrices no cuadradas ($m \leq n$) se presentan los mismos datos que en el caso anterior cambiando la nomenclatura y añadiendo:

n	número de renglones
m	número de columnas

Para completar la presentación, se incluyen gráficas que nos indican las medias de los tiempos de corrida obtenidos en una misma máquina y bajo las mismas condiciones para todos los algoritmos analizados. Todos los algoritmos fueron codificados en lenguaje A.P.L. (A Programming Language). La máquina utilizada para este trabajo fué una IBM modelo 3278-2. Los datos se generaron aleatoriamente con una aproximación de dos cifras decimales, siendo el rango entre 0 y mxn, de tal forma que $0 \leq C_{ij} \leq mxn$.

Para registrar los tiempos de corrida se utiliza el tiempo de procesamiento de la máquina, con una aproximación de milésimas de segundos y comenzando a partir del momento en el cual el algoritmo

empieza a correr, es decir, en todos los casos se excluyó el tiempo consumido en la generación de la matriz de coeficientes C_{ij}. Para medir los tiempos promedio cada algoritmo se corrió 50 veces con distintas matrices aleatorias del mismo tamaño.

3.2 Codificación de los Algoritmos

3.2.1 Programas Utilizados en Todos los Algoritmos

Nombre : DATOS
Descripción : Función que genera una matriz aleatoria Z de NxM que contiene datos con dos decimales
Input : N, M
Output : Z
Variables : I

```
[1]     I←1
[2]     Z←(N,M)⍴0
[3]     Z[I;]←(M?N×M×100)+100∩→⍞LC SI N≥I←I+1
```

Nombre : MSEG
Descripción : Función que convierte un vector V de (minutos, segundos, milésimas de segundos) en un número real con segundos.milésimas de segundos
Input : V
Output : Z
Variables : I

```
[1]     Z←(V[3]÷1000)+Z←V[2]+60×V[1]
```

Nombre : SEG
Descripción : Función inversa de MSEG. Convierte Segundos.Milésimas de segundos a (minutos,segundos,milésimas de segundos)
Input : T - Número real con Segundos.Milésimas de Segundos
Output : Z - Vectos con las tres componentes deseadas
Variables : -

```
[1]     Z←3⍴0
[2]     Z[1 2]←⌊60 60⊤T
[3]     Z[3]←1000×(60 60⊤T)[2]-Z[2]
```

Nombre : SI
Descripción : Función condicional. Ejecuta A si B es verdadera
Input : A, B
Output : Z
Variables : -

[1] Z ← B/A

3.2.2 Gross

Programa Pincipal :	GROSS	
Funciones :	DATOS	*
	MSEG	*
	CICLO	
	SEG	*
	SI	*
Sub-Programas :	BUSCA	
Variables Globales :	A	Matriz de Costos
	AS	Asignación
	ASO	Asignación Nueva
	END	Bandera 1 No Existe el ciclo 0 Existe el ciclo
	JR	Conjunto del número de Renglones
	MAX	Valor máximo en la asignación AS
	REN, COL	Coordenadas de Max, donde inicia y termina el ciclo
	RENMAX	Renglón donde se va el ciclo
	COLANT	Columna donde se va el ciclo
	N	Número de renglones y columnas de la matriz A
	TIEMPO	Tiempo de corrida
Variables Locales :	DATOS	
	I	Contador
	MSEG	
	-	
	CICLO	
	J	Conjunto de columnas

SEG

-

BUSCA

I	Contador
JC	Conjunto de renglones
J	Conjunto de columnas

SI

-

* Estos Programas no se presentan en esta sección, ya que fueron descritos en la sección 3.2.1

```
Nombre :      GROSS
Descripción :  PROGRAMA PRINCIAPAL
Variables :    END, JR, REN, COL, MAX, ASO, RENMAS, COLANT, AS, A, N,TIEMPO

[1]      A ← N DATOS N
[2]      END ← 0
[3]      TIEMPO ← MSEG ⌸TS [5 6 7]
[4]      AS ← A × A ^ . = O A
[5]      CICLO:
[6]        JR ← ιN
[7]        REN ← (0≠MAX ← AS[;COL ← (∇+/[1]AS) [1]]) ι1
[8]        → TIEMP SI (ρ+/A) < (A[;COL] < MAX ← +/MAX) ι1
[9]        ASO ← AS
[10]       L1:
[11]         Φ 'BUSCA ∩ → TIEMP SI END = 1 ∩ → FIN' SI (ρJR) < (A[JR;COL] < MAX) ι1
[12]         RENMAX ← JR[(A[JR;COL] < MAX) ι1]
[13]         Φ 'CICLO (RENMAX,COL,REN,COLANT) ∩ → FIN' SI A[REN;COLANT ← (AS[RENMAX]
             ≠ 0) ι1] < MAX
[14]         JR ← (JR-RENMAX) ≠ 0)/JR
[15]       → L1
[16]       FIN
[17]       AS ← ASO
[18]       → CICLO
[19]     TIEMP:
[20]       TIEMPO ← SEG TIEMPO ← (MSEG ⌸TS [5 6 7]) - TIEMPO
```

```
Nombre :      BUSCA
Descripción :  Programa que busca un ciclo con más de 4 componentes en la matriz A
Input :        -
Output :       END - Bandera, ASO - Asignación con el ciclo encontrado
Variables :    I, JC, J
```

```
[1]     Φ 'END ← 1 ∩ → 0' SI 0 = ρJC ← +/[1](O(ρJC) ριN) × JC ← AS[;(A[REN;] < MAX) / ιN] ≠
        0
[2]     ASO[RENMAX;COL] ← A[RENMAX;COL]
[3]     JR ← (((ιN)-RENMAX)≠ 0) /[1] ιN
[4]     L1:
[5]       I ← 1
[6]       Φ 'END ← 1 ∩ → 0' SI (ρJR) < (A[JR;COLANT] < MAX) ι1
[7]       RENMAX ← JR[(A[JR;COLANT] < MAX) ι1
[8]       Φ 'RENMAX ← JC[I] ∩ CICLO (RENMAX,COLANT,REN,(0 ≠ AS[RENMAX;]) ι1) ∩ → 0' SI
          (A[JC[I];COLANT] < MAX) ∧ JC[I] ∈ JR ∩ → ⎕LC SI (ρJC) ≥ I ← I+1
[9]       JR ← ((JR-RENMAX) ≠ 0) /[1]JR
[10]      ASO[RENMAX;COLANT] ← A[RENMAX;COLANT]
[11]      COLANT ← (AS[RENMAX;] ≠ 0) ι1
[12]    → L1 SI (RENMAX = REN) ∧ (ρJR) ≥ (A[JR;COLANT] < MAX) ι1
[13]    ASO ← A × ASO ≠ AS
[14]    ASO[;J] ← AS[;J ← (0 ≠ J) /J ← (ιN) × 0 = +/[1]ASO]
[15]    Φ 'END ← 1 ∩ → 0' SI 1 = ∨/ 0 = +/ASO
[16]    Φ 'ASO ← AS ∩ → L1' SI ((ρ+/[1]ASO) ≥ (0 = +/[1]ASO) ι1) ∨ ((ρ+/ASO) ≥ (0 = +/ASO)
        ι1)
```

Nombre : CICLO
Descripción : Función que cambia en la asignación un ciclo de 4 componentes
Input : V - Vector de 4 componentes (renglón1,columna1,renglón 2,columna2)
Output : ASO - Nueva asignación
Variables : J

```
[1]     ASO[V[1];V[2]] ← A[V[1];V[2]]
[2]     ASO[V[3];V[4]] ← A[V[3];V[4]]
[3]     ASO ← A × ASO ≠ AS
[4]     AOS[;J] ← AS[;J ← (0 ≠ J) /J ← (ιN) × 0 = +/[1]ASO]
```

3.2.3 Garfinkel

Programa Pincipal : GARF

Funciones : DATOS *
MSEG *
FORDE

	SEG	*
	SI	*
Variables Globales :	A	Matriz de Costos
	V	Cota para definir los arcos que son admisibles
	T	Matriz de 2 x N 1 - Etiqueta de Ri a Cj 2 - Etiqueta de Cj a el destino
	N	Número de renglones y columnas de la matriz A
	TIEMPO	Tiempo de corrida
Variables Locales :	DATOS	
	I	Contador
	MSEG	
	-	
	FORDF	
	MAT	Matriz con entradas admisibles
	ETIQUETA	Matriz de 2 x 2 x N 1 - Renglones, Columnas 2 - Flujo, Etiqueta 3 - Nodos
	R	Conjunto de a lo más N elementos
	T	Conjunto Temporal
	FIN	Contador
	TEMP	Temporal
	SEG	
	-	
	SI	
	-	

* Estos Programas no se presentan en esta sección, ya que fueron descritos en la sección 3.2.1

Nombre : **GAR**
Descripción : PROGRAMA PRINCIAPAL
Variables : A,N,TIEMPO,V,T

```
[1]     A ← N DATOS N
[2]     TIEMPO ← MSEG ⎕TS [5 6 7]
[3]     V ← ⌈/ ( ⌊/A ),/[1]A
[4]     INICIO:
[5]       T ← FORDF A ≤ V
[6]       Φ 'V ← ⌊/,A[(T[1;] ≠ 0)/ ι N; (T[2;] = 0)/ ι N] ∩ → INICIO' SI ~∧/∧/ T = 0
[7]     TIEMPO:
[8]       TIEMPO ← SEG TIEMPO ← (MSEG ⎕TS [5 6 7]) - TIEMPO
```

Nombre : FORDF
Descripción : Programa que realiza el algoritmo de etiquetas de Ford-Fulkerson para flujo máximo
Input : MAT - Matriz con entradas admisibles
Output : Z - Matriz de 2 × N con etiquetas
Variables : ETIQUETA,R,T,FIN,TEMP

```
[1]     FIN ← N ρ0
[2]     ETIQUETA ← (2 2,N) ρ0
[3]     ETIQUETA [1;2;] ← N+1
[4]     RUTAA:
[5]       ETIQUETA [1;2;R ← ((ETIQUETA [1;2;] = N+1) ∧ ETIQUETA [1;1;] = 0) / ι N ] ← N+1
[6]       Φ '→ FIN' SI 0 = ρR
[7]       R ← R[1]
[8]       ETIQUETA [2;2; ( MAT[R;] = 1)/ ι N] ← R
[9]       R ← (ETIQUETA[2;2] = R) ι 1
[10]      Φ 'FIN[R] ← 1 ∩ → RUTAB' SI FIN [R] = 0
[11]      ETIQUETA [1;2;T] ← T ← (MAT[;R] = 1)/ ι N
[12]    → RUTAA
[13]    RUTAB:
[14]      ETIQUETA [2;1;R] ← 1
[15]      R ← ETIQUETA [2;2;R]
[16]      ETIQUETA [1;1;R] ← 1
[17]    → RUTAA
[18]    FIN:
[19]      Z ← (2,N) ρ0
[20]      → 0 SI N = +/FIN
[21]      Z[1;R] ← R ← ( ETIQUETA[1;2;] ≠ N+1)/ ι N
[22]      R ← (ETIQUETA [1;1;] = 0)/ ι N
[23]      Z[2;TEMP] ← TEMP ← ( ETIQUETA [2;2;] = R[1] )/ ι N ∩ R ← 1↓R ∩ → ⎕LC SI 0 ≠ ρR
```

3.2.4 Derigs - Zimmermann

Programa Pincipal : ZIMMER

Funciones :
- DATOS .
- MSEG .
- HEURISTICA
- AUMPATH
- SEG .
- SI .

Sub-Programas :
- BUSCAZL
- PATH
- PATH1
- DIJKSTRA

Variables Globales :

A	Matriz de Costos
S	Vector de renglones
T	Vector de columnas
KL	Contador de número de renglones que están en la asignación heurística
ASMAXOP	Valor del máximo en la asignación encontrada
ASMAX	Valor del máximo en la asignación actual
K	Contador del número de renglones que están en la asignación
ASO	Vector de columnas que contienen en la diagonal la asignación
ZL	Valor de Z en la asignación heurística
N	Número de renglones en la matriz A
M	Número de columnas en la matriz A
TIEMPO	Tiempo de corrida

Variables Locales :

DATOS
- I Contador

MSEG
-

HEURISTICA
- TEMP Temporal
- JC Conjunto de columnas que están asignadas
- I Contador
- COL Conjunto de columnas que las entradas de la matriz son menores que ZL

AUMPATH
- JR Conjunto de renglones
- ETIQUETA Matriz de 2 x (KL+1)
  1 - Etiqueta de Xi; 2 - Estado
- J Contador
- R Renglón Inicial
- VAL Valor asignado en las columnas: KL+1,..., M
- REN,COL Coordenadas de VAL
- P Nodo Inicial
- TM Conjunto de renglones

SEG
-

PATH
- JC Conjunto de columnas

PATH1
-

DIJKSTRA
-

* Estos Programas no se presentan en esta sección, ya que fueron descritos en la sección 3.2.1

Nombre : **ZIMMER**
Descripción : PROGRAMA PRINCIAPAL
Variables : KL,ASMAXOP,ASMAX,K,BANDERA,ASO,ZL,S,T,A,N,M,TIEMPO

[1] A ← N DATOS N

```
[2]     TIEMPO ← MSEG ⍞TS [5 6 7]
[3]     HEURISTICA
[4]     HEU ← KL∩→ TIEMP SI KL = N
[5]     INICIO:
[6]       K ← KL+1
[7]       ASMAX ← ⌈/ 1 1 ○ A[S;T[⍳K]]
[8]       AUMPATH
[9]       Φ 'T ← ASO' SI ASMAXOP ≤ ASMAX
[10]      KL ← KL+1
[12]    →INICIO SI KL < N
[13]    TIEMP:
[14]      TIEMPO ← SEG TIEMPO ← (MSEG ⍞TS [5 6 7]) - TIEMPO
```

Nombre : **HEURISTICA**
Descripción : Programa que encuentra la asignación parcial inicial utilizando el método heurístico en una matriz dada
Inputs: A - Matriz de constantes
Outputs: S - Vector de renglones ordenados; T - Vector de columnas ordenadas; ZL - Valor óptimo de la asignación
Variables: TEMP,JC,I,COL

```
[1]     T ← M⍴0
[2]     ZL ← ⌈/ (⌊/A ),/[1]A
[3]     Φ 'BUSCAZL' SI N < M
[4]     S ← Δ +/A ≤ ZL
[5]     JC ← ⍳M
[6]     KL ← 0
[7]     I ← 1
[8]     LC:
[9]       TEMP ← ((COL ← (COL ≠ 0)/COL ← (⍳M)× A[S[I];] ≤ ZL) ∈ JC) ⍳1
[10]      Φ 'I ← I+1 ∩→ FINAL SI I > N ∩→ LC' SI TEMP > ⍴COL
[11]      KL ← KL+1
[12]      T[I] ← COL[TEMP]
[13]      JC ← (0 ≠ JC × JC ≠ COL[TEMP])/JC
[14]    →LC SI N ≥ I ← I+1
[15]    FINAL:
[16]      S ← ((T[⍳N] ≠ 0)/S),(T[⍳N] = 0)/S
[17]      T ← (T ≠ 0)/T ← ((T≠0)/T),(T = 0)/T,JC
[18]      T[(¯1+N)+⍳(1+M-N)] ← T[(¯1+N)+ Δ A[S[N];T[(¯1+M)+⍳(1+M-N)]]]
```

Nombre : **BUSCAZL**
Descripción : Programa que encuentra el valor de Z para la asignación heruística en matrices no cuadradas
Inputs: -
Outputs: ZL = Z
Variables: COLL

```
[1]     COLL←⍳M
[2]     LL:
[3]     Φ' →0' SI (N>⍴COLL←(ZL≠⌊/[1]A[;COLL])/COLL) ∨ (⌈/(⌊/[1]A[;COLL])<⌈/⌊/A
[4]   ZL←⌈/(⌊/A),⌊/[1]A[;COLL] ∩ → LL
```

Nombre : **AUMPATH**
Descripción : Programa que encuentra el 'camino de aumento' para K+1 de una matriz A con su asignación correspontiende hasta K utilizando el método de Dijkstra modificado
Inputs: A - Matriz de constantes; S - Vector de Renglones; T - Vector de columnas
Outputs: ASO - Vector de columnas ordenadas para la nueva asignación; ASMAXOP - valor máximo en la nueva asignación
Variables: JR, ETIQUETA, J, R, VAL, COL, REN, P, TM

```
[1]   ETIQUETA←(2,(K+1))⍴0∩ETIQUETA[1;1+⍳K]←VAL←1+M×N∩ETIQUETA[;1]←0 1
[2]   P←1
[3]   R←KL+1
[4]   BANDERA←0
[5]   PATH∩→FIN SI BANDERA=1
[6]   P←(0≠P)/P←(⍳(¯1↑⍴ETIQUETA))×ETIQUETA[2;]=0
[7]   ETIQUETA[2;P←P[(∇ ETIQUETA[1;P])[⍋P]]]←1
[8]   R←¯1+P
[9]   INICIO:
[10]    Φ'PATH' SI ~∧/ETIQUETA[2;⍳KL+1]=1∩→FIN SI BANDERA=1
[11]    Φ'PATH1' SI ETIQUETA[2;KL+2]=0
[12]    P←(0≠P)/P←(⍳(¯1↑⍴ETIQUETA))×ETIQUETA[2;]=0
[13]    ETIQUETA[2;P←P[(∇ ETIQUETA[1;P])[⍋P]]]←1
[14]    R←¯1+P
[15]  →INICIO SI ETIQUETA[2;KL+2]=0
[16]  ASO←T
[17]  →FIN SI VAL=1+M×N
[18]  ASO[(KL+1),(ASO=COL)/⍳M]←COL,ASO[KL+1]
[19]  JR←⍳N
[20]  J←KL
[21]  TM←0⍴0
[22]  CICLO:
[23]    TM←TM,REN
[24]    JR←(JR-REN)≠0)/JR
[25]    REN←(REN∈JR)/REN←(0≠REN)/REN←(⍳(KL+1))×ETIQUETA[2;1+⍳(KL+1)]=1
[26]    →FIN SI 0=⍴REN
[27]    REN←REN[(⍋A[S[REN];ASO[J]])[1]]
[28]    J←REN
[29]  →CICLO SI J≠KL+1
[30]  TM←TM,REN
[31]  ASO[TM]←ASO[KL+1;¯1↓TM]
[32]  ASMAXOP←⌈/1 1 0 A[S;ASO]
```

```
[33]        → 0
[34]        FIN: ASMAXOP ← VAL

Nombre :    PATH
Descripción : Función que evalúa el método de Dijkstra
Inputs:     -
Outputs:    ETIQUETA - Evaluado
Variables:  JC

[1]         JC ← 1+JC ← ( T[JC] ≠ T[R] )/JC ← (JC ≠ 0)/JC ← (ι KL) × ETIQUETA [2;1+ ι KL] = 0
[2]         Φ 'BANDERA ← 1 ∩ → 0' SI 0 = ρJC
[3]         J ← 1
[4]         ETIQUETA [1;JC[J]] ← DIJKSTRA ( ETIQUETA [1;JC[J]], ETIQUETA [1;P], A[S[R];T[`
            1+JC[J]]]) ∩ → ÉLC SI ( ρ JC) ≥ J ← J+1

Nombre :    DIJKSTRA
Descripción : Función que evalúa las etiquetas según el procedimiento Dijkstra
Inputs:     V - vector que contiene: V[1] - Etiqueta de Xj; V[2] - Etiqueta de P; V[3] - Costo de P
            a Xj
Outputs:    Z - Nuevo valor de la etiqueta de Xi
Variables:  -

[1]         Z ← V[2] ∩ Φ 'Z ← V[3]' SI V[2] < V[3]
[2]         Φ 'Z ← V[1]' SI V[1] < Z

Nombre :    PATH1
Descripción : Función que evalúa el método de Dijkstra desde KL hasta M tomando el valor mínimo
Inputs:     -
Outputs:    VAL - Valor mínimo; COL - Columna donde se encuentra VAL; REN - Rengón donde se
            encuentra VAL
Variables:  -

[1]         VAL ← ⌊ / A[S[R];T[KL+ ι (M-KL)]]
[2]         COL ← (COL ≠ 0)/COL ← (KL+ ι (M-KL)) × ,A[S[R];T[KL+ ι (M-KL)]] = VAL
[3]         REN←R
[4]         ETIQUETA [1; KL+2] ← DIJKSTRA ( ETIQUETA [1;KL+2]; ETIQUETA [1;P]; VAL )
```

3.2.5 Romero.

Programa Pincipal : DAVID

Funciones :

DATOS	·
MSEG	·
HEURISTICA	·
BUSCAMAX	
P1	
CAMBIAP1	
P2	
BUSCAD	
CAMBIACOL	
SEG	·
SI	·

Sub-Programas : BUSCAZL ·

Variables Globales :

A	Matriz de Costos
S	Vector de renglones
T	Vector de columnas
KL	Contador de número de renglones que están en la asignación heurística
MAX	Valor del máximo en la asignación encontrada
REN	Coordenadas de Max
COL1	Columna en la que la entrada en A es menor que MAX
WB	Mínimo en la zona B
WC	Mínimo en la zona C
MIN	Mínimo en una determinada zona heurística
N	Número de renglones en la matriz A
M	Número de columnas en la matriz A
TIEMPO	Tiempo de corrida

Variables Locales :

DATOS
- I Contador

MSEG
-

HEURISTICA
- TEMP Temporal
- JC Conjunto de columnas que están asignadas
- I Contador
- COL Conjunto de columnas que las entradas de la matriz son menores que ZL

BUSCAMAX
-

P1
- R Conjunto de columnas que las entradas de A son menores que MAX
- R1 Conjunto de columnas que las entradas de A son ≥ que MAX

CAMBIAP1
-

P2
- B Zona de la matriz a ordenar
- R Conjunto de renglones que las entradas de B son > que MAX
- C Conjunto de renglones que las entradas de B son ≤ que MAX
- F ρ C

BUSCAD
- JC Conjunto de columnas
- R Renglón donde se encuentra el nuevo elemento

CAMBIACOL
- TEMP Temporal

SEG
-

* Estos Programas no se presentan en esta sección, ya que fueron descritos en la sección 3.2.1 y 3.2.4

```
Nombre :      DAVID
Descripción :  PROGRAMA PRINCIAPAL
Variables :    KL,ASMAXOP,ASMAX,K,BANDERA,ASO,ZL,S,T,A  N,M,TIEMPO

[1]      A ← N DATOS N
[2]      TIEMPO ← MSEG ⍕TS [5 6 7]
[3]      S ← ⍳N
[4]      T ← ⍳M
[5]      HEURISTICA
[6]      HEU ← KL ∩ → TIEMP SI KL = N
[7]      INICIO1:
[8]         BUSCAMAX
[9]         P1
[10]        Φ 'COL1 ← COL1[(⍳A[S[REN];T[COL1]] = ⌊/A [S[REN];T[COL1]] ) ⍳1 ∩ CAMBIACOL
            (REN,COL1) ∩ → INICIO1' SI ( ⌊/A [S[REN];T[COL1 ← REN + ⍳ (M-REN)]] ) < MAX
[11]        WB ← ⌊/⌊/ A[S[⍳(⍕1+REN)];T[REN + ⍳ (M-REN)]]
[12]        → TIEMP SI WB ≥ MAX
[13]        WC:
[14]           WC ← 1+(⍳ (⍕1+REN)) P2 REN + ⍳ (M-REN)
[15]           Φ 'BUSCAD ∩ → INICIO1' SI  (MIN ← ⌊/⌊/A[S[REN];T[COL1←(⍕1+WC) ↓ ⍳⍕1+REN]] )
               < MAX
[16]           CAMBIAP1 (REN,WC)
[17]           → TIEMP SI ∧/ (,A[S[⍳ WC];T[WC+ ⍳ (M-REN)]] ) ≥ MAX
[18]           BUSCAMAX
[19]        → WC
[20]     TIEMP:
[21]        TIEMPO ← SEG TIEMPO ← (MSEG ⍕TS [5 6 7]) - TIEMPO


Nombre :      P 1
Descripción :  Programa que ordena la matriz A en base a la subrutina P1
Inputs:        -
Outputs:       Matriz ordenada
Variables:     R,R1

[1]      R ← (0 ≠ R) / R ← (⍳ N) × A[S;T[REN]] < MAX
[2]      R1 ← (0 ≠ R1) / R1 ← (⍳ N) × A[S;T[REN]] ≥ MAX
[3]      Φ 'CAMBIAP1 (1,REN) ∩ REN ← 1 ∩ → FIN' SI 1 = ∧/R1 = REN
[4]      CASOA:
[5]         R ← (0 ≠ R) / R ← R × R ≤ REN
[6]         R1 ← (0 ≠ R1) / R1 ← R1 × R1 > REN
[7]         Φ '→ CASOB' SI (0 = ρR) ∨ 0 = ρR1
[8]         CAMBIAP1 (R[1],R1[1]) ∩ R ← 1 ↓ R ∩ R1 ← 1 ↓ R1 ∩ → ⍕LC SI (0 ≠ ρR) ∧ 0 ≠ ρR1
[9]         CASOB:
[10]           → FIN SI (0 = ρR) ∧ 0 = ρR1
[11]           → CASOC SI 0 = ρR1
[12]           Φ 'CAMBIAP1 (R1[ρR1],REN) ∩ REN ← REN + (R1[ρR1] - REN) ∩ → FIN' SI 1 = ∧/R1
```

```
          = REN + ι (ρR1)
[13]      Φ 'CAMBIAP1 (R1[1],(1+REN))' SI R1[1] ≠ 1 ∩ CAMBIAP1 (REN,1+REN) ∩ R1 ← 1
          ↓ R1 ∩ REN ← 1 + REN ∩ → ¯LC SI 0 ≠ ρR1
[14]      → FIN
[15]      CASOC:
[16]      Φ 'CAMBIAP1 (R[1],REN) ∩ REN ← REN - (REN - R[1]) ∩ → FIN' SI ∧/R = Φ REN - ι ρR
[17]      Φ 'CAMBIAP1 (R1[1],(¯1+REN))' SI R[1] ≠ ¯1+REN ∩ Φ 'CAMBIAP1 (REN,(¯1+REN))' SI R[1] ≠ REN ∩ R ← 1 ↓ R ∩ REN ← ¯1 + REN ∩ → ¯LC SI 0 ≠ ρR
[18]  FIN: → 0
```

Nombre : **CAMBIAP1**
Descripción : Permuta dos renglones y dos columnas sin alterar la diagonal
Inputs: Row - vector que contiene los dos renglones y las dos columnas a permutar
Outputs: Nuevo S y T
Variables: -

```
[1]   S[ROW[1],ROW[2]] ← S[ROW[2],ROW[1]]
[2]   T[ROW[1],ROW[2]] ← T[ROW[2],ROW[1]]
```

Nombre : **P 2**
Descripción : Ordena la zona B
Inputs: Zona B, determinada por los vectores V (renglones) y V1(columnas)
Outputs: Zona B ordenada y el elemento mínimo WC
Variables: B,R,C,F

```
[1]   INICIO:
[2]      B ← A[S[V];T[V1]
[3]      F ← ρC ← (C ≠ 0)/C ← (ι 1↑ ρB) × MAX ≤ ⌊/ B
[4]      R ← (R ≠ 0)/R ← (ι 1↑ ρB) × MAX > ⌊/ B
[5]      Φ 'Z ← 0 ∩ → 0' SI 0 = ρC
[6]      Φ 'Z ← C[ρC ] ∩ → 0' SI ∧/C[ρC ] < R[1]
[7]      L1:
[8]         CAMBIAP1 (R[1],C[ρC ]) ∩ R ← 1 ↓ R ∩ C ← ¯1 ↓ C
[9]         → INICIO SI 0 = (∨/A[S[V];T[V1]] < MAX ) [ι F]
[10]     → L1 SI (0 ≠ ρR) ∧ 0 ≠ ρC
[11]  → INICIO
```

Nombre : **BUSCAD**
Descripción : Programa que altera la diagonal con el mínimo de una zona
Inputs: -
Outputs: Nuevo S, T, y nueva asignación
Variables: JC,R

```
[1]   JC ← 0 ρ0
```

```
[2]     COL1 ← COL1[(ι MIN = A[S[REN];T[COL1]]) ι 1]
[3]     R ← REN
[4]     JC ← JC,COL1
[5]     IN:
[6]       JC ← JC,COL1 ← R + (A[S[COL1];T[R + (ι M - R)]] < MAX) ι 1
[7]       ⍝ 'JC ← ¯1 ↓ JC' SI COL1 = 1 + M
[8]       → FIN SI COL1 ≥ N
[9]       → FIN SI A[S[COL1];T[REN]] < MAX
[10]      R ← COL1
[11]    → IN
[12]    FIN:
[13]    JC ← JC[ΔJC]
[14]    CAMBIACOL JC
```

```
Nombre :      CAMBIACOL
Descripción :  Programa que altera la diagonal
Inputs:       CL - Vector de renglones y columnas nuevas
Outputs:      Nuevo T
Variables:    TEMP
```

```
[1]     TEMP ← T[REN]
[2]     T[REN] ← T[CL[1]]
[3]     T[CL[ι¯1 + ρCL]] ← T[CL[1 + ι¯1 + ρCL]
[4]     T[CL[ ρCL] ← TEMP
```

3.3 Resultados y Comentarios

3.3.1 Comentarios y Tablas de Resultados

De acuerdo con el procedimiento propuesto por Gross, se toma como punto de partida para la primera asignación la diagonal de la matriz. A partir de ésta se busca el ciclo más corto, es decir, se toma el primer elemento en la columna de (i^*,j^*) que sea menor que V, se va a la asignación y se busca cerrar el ciclo. En el caso de que esto no suceda, se busca un ciclo más largo, tratando de cerrarlo en la primera oportunidad que se presente. Una descripción gráfica de lo anterior lo constituye la figura 3.3.1.1.

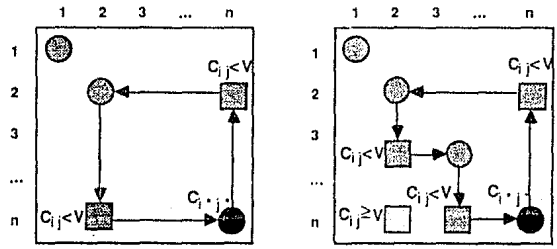


Figura 3.3.1.1

Este algoritmo no presentó ningún problema de codificación y se vió que para matrices chicas ($n \leq 50$) es bastante rápido. Los resultados obtenidos

se presentan en la Tabla I.

Tabla I

	t			t máx			t min			Heu	σ		
n	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
5	0	0	76	0	0	180	0	0	17	-	0	0	38
10	0	0	369	0	4	707	0	0	116	-	0	0	638
20	0	1	474	0	2	396	0	0	613	-	0	0	454
30	0	3	93	0	5	547	0	1	777	-	0	0	692
40	0	7	128	0	16	873	0	3	464	-	0	2	439
50	0	16	758	0	34	954	0	7	444	-	0	9	145
60	0	28	681	0	55	3	0	16	14	-	0	9	543
70	0	47	678	1	2	260	0	27	727	-	0	8	556
80	1	8	217	2	3	767	0	42	480	-	0	14	848

M Minutos S Segundos MMS Milésimas de Segundos

El único inconveniente que se presentó en este algoritmo fué que por falta de capacidad de memoria en la máquina sólo fué posible correrlo hasta con matrices de 80x80.

El algoritmo de Garfinkel se corrió únicamente con matrices cuadradas, ya que el autor así lo propone, aunque también es posible correrlo con matrices no cuadradas, haciendo una modificación a la red **R** planteada en la sección 2.3.1. Los resultados obtenidos se presentan en la Tabla II. Este algoritmo fué posible correrlo hasta con matrices de 120x120, ya que para matrices más grandes la memoria de la computadora se satura.

Como señalamos en la sección 2.4.1., en el algoritmo de Derigs - Zimmermann, se busca primero una asignación parcial inicial para el problema P_k por el método heurístico presentado por los autores. Durante

la aplicación de éste método, para matrices no cuadradas se vió que en el momento de calcular el valor de z no se debe de tomar únicamente el valor máximo de t_j :

$$t_j^* = \min_{I \in m} C_{ij} \qquad \text{para todo } j \in n \qquad (1)$$

ya que podría darse el caso de que t_j^* esté en una columna fuera de la asignación, y si existiera un elemento $C_{ij} < t_j^*$ en alguna columna i dentro o fuera de la asignación éste es el que se debe de tomar como t_j^* (Ver figura 3.3.1.2.). En general, para encontrar el verdadero t_j^* se debe tomar en cuenta las combinaciones de n elementos t_j tomados de m en m, ya que para efectos de asignación sólo se toman m elementos, siendo la fórmula:

$$t_j^* = \min [\ \text{máx}\ C_m^n\ t_j]$$

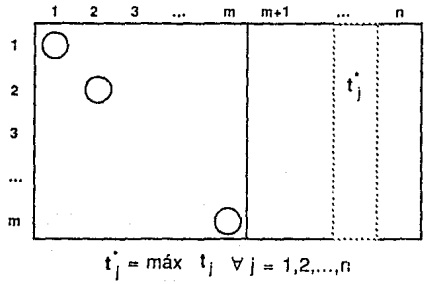


Figura 3.3.1.2

Para mejorar el tiempo de respuesta en la computadora, modificamos el

programa para que ordenaran los elementos t_j en forma creciente y se tomaran únicamente los primeros **m** términos, obteniendo con éstos el valor inicial de z.

Para conocer $|\mu| = K$ del método heurístico se tomó un contador **Heu**, y partiendo de esta solución inicial P_k se aplica la modificación del Algoritmo de Dijkstra planteado en la sección 2.4.3 para $k+1,\ldots,m$. Los resultados obtenidos se presentan en la Tabla III.

Como el Algoritmo de D. Romero es primal, se le aplicó el método heurístico par inicializar la matriz, a diferencia de que al encontrar la solución inicial P_k se sigue tomando la diagonal como solución, ya que este algoritmo supone una asignación inicial. Los resultados obtenidos se presentan en la Tabla IV.

Tabla II

	t			t máx			t min			Heu	σ		
n	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
10	0	1	256	0	2	568	0	0	226	-	0	7	766
20	0	5	15	0	25	252	0	1	458	-	0	33	767
30	0	11	988	0	43	247	0	3	526	-	0	41	215
40	0	22	589	0	55	873	0	8	489	-	0	38	632
50	0	37	831	1	29	856	0	9	587	-	0	28	988
60	0	46	996	2	18	325	0	12	357	-	0	46	480
70	1	0	453	2	35	486	0	30	456	-	0	18	984
80	1	30	570	2	42	596	0	54	896	-	0	50	602
90	3	41	973	6	11	782	0	59	365	-	0	57	970
100	3	53	306	11	55	465	1	3	54	-	1	1	724
110	5	50	849	15	19	159	1	43	631	-	1	41	483
120	6	46	690	20	25	798	3	12	187	-	2	51	382

M Minutos **S** Segundos **MMS** Milésimas de Segundos

Tabla III

n	`t			t máx			t min			Heu	σ		
m	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
10													
10	0	0	487	0	5	146	0	0	53	9	0	0	743
20													
20	0	3	202	0	28	570	0	0	103	18	0	4	563
30	0	0	747	0	3	710	0	0	143	19	0	0	710
40	0	0	790	0	2	700	0	0	173	20	0	0	601
50	0	0	875	0	2	894	0	0	164	20	0	0	577
60	0	1	104	0	2	163	0	0	360	20	0	0	389
70	0	1	494	0	3	0	0	0	343	20	0	0	614
80	0	2	944	0	13	303	0	0	687	20	0	2	877
90	0	6	684	0	28	517	0	0	960	20	0	5	402
100	0	27	514	1	15	830	0	1	146	20	0	23	608
120	0	28	325	0	52	244	0	0	852	20	0	12	943
30													
30	0	14	925	0	57	167	0	0	417	27	0	11	952
40	0	6	663	0	45	803	0	0	184	29	0	9	476
50	0	3	471	0	41	80	0	0	250	29	0	6	438
60	0	2	710	0	9	310	0	0	373	30	0	2	727
70	0	3	781	0	28	760	0	0	400	30	0	4	893
80	0	3	431	0	8	14	0	0	623	30	0	2	648
90	0	9	705	0	20	200	0	0	773	30	0	4	226
100	0	10	262	0	29	157	0	2	280	30	0	5	600
40													
40	0	27	941	1	2	640	0	0	240	37	0	11	397
50	0	20	222	0	43	900	0	0	243	38	0	8	579
60	0	14	314	0	24	220	0	0	263	39	0	4	338
70	0	10	736	1	0	743	0	0	396	40	0	10	394
80	0	5	944	0	32	440	0	0	450	40	0	5	634
90	0	6	559	0	29	334	0	0	753	40	0	5	712
100	0	7	496	0	22	396	0	0	954	40	0	3	413
120	0	15	697	0	47	419	0	1	426	40	0	8	487
50													
50	0	36	854	2	54	87	0	10	273	46	0	28	607
60	0	14	49	1	28	746	0	0	326	47	0	16	808
70	0	7	412	0	47	236	0	0	417	48	0	11	795
80	0	6	391	0	36	123	0	0	586	49	0	9	623
90	0	6	822	0	41	254	0	0	766	49	0	9	148
100	0	5	234	0	25	493	0	0	804	49	0	5	829
110	0	7	254	0	24	260	0	0	920	49	0	4	877
120	0	15	506	1	4	387	0	1	243	49	0	11	783
130	0	10	978	1	4	667	0	3	310	50	0	11	890
140	0	11	903	0	53	190	0	1	957	49	0	8	843
150	0	11	809	0	42	750	0	3	223	49	0	7	826
160	0	11	407	0	42	215	0	2	428	50	0	6	214
60													
60	0	43	577	2	21	476	0	7	800	55	0	28	938

Tabla III (Continuación)

n / m	t			t máx			t min			Heu	σ		
	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
60													
70	0	29	879	2	35	413	0	0	703	57	0	26	723
80	0	11	767	0	54	160	0	0	647	58	0	11	657
90	0	8	406	0	36	494	0	0	497	58	0	8	365
100	0	6	511	0	16	350	0	0	663	59	0	4	55
110	0	7	273	0	24	453	0	0	720	59	0	5	252
120	0	8	645	0	39	330	0	1	113	59	0	6	625
130	0	10	339	0	27	370	0	1	233	59	0	4	908
140	0	13	224	0	30	520	0	1	586	58	0	6	692
150	0	17	824	1	24	170	0	3	133	59	0	13	197
160	0	21	281	0	50	890	0	6	3	59	0	11	627
70													
70	0	41	210	1	41	884	0	12	360	64	0	16	339
80	0	25	406	1	35	640	0	0	660	67	0	19	771
90	0	14	385	1	9	603	0	0	614	68	0	13	156
100	0	6	765	0	25	550	0	0	576	68	0	6	176
110	0	4	947	0	17	506	0	0	573	68	0	4	754
120	0	5	603	0	10	617	0	0	757	69	0	3	234
130	0	6	800	0	16	733	0	0	770	69	0	4	470
140	0	8	402	0	19	514	0	1	223	69	0	4	684
150	0	9	924	0	24	383	0	0	863	69	0	5	152
160	0	10	66	0	24	400	0	1	856	69	0	4	839
80													
80	0	48	258	1	43	587	0	16	266	74	0	19	97
90	1	5	399	3	5	813	0	7	947	76	0	46	122
100	0	19	559	1	11	920	0	0	866	78	0	17	662
110	0	17	395	1	9	684	0	1	230	78	0	15	380
120	0	13	906	0	54	253	0	1	310	78	0	12	158
130	0	18	840	1	29	3	0	1	807	78	0	16	855
140	0	17	984	1	3	77	0	1	560	79	0	14	421
150	0	15	390	1	47	287	0	2	367	79	0	15	448
160	0	22	787	1	14	797	0	2	347	79	0	14	957
90													
90	1	45	375	4	15	803	0	30	757	83	0	55	32
100	1	0	232	5	13	703	0	3	300	86	0	48	98
110	0	31	699	1	49	884	0	0	934	87	0	27	700
120	0	17	381	1	37	320	0	0	940	88	0	21	429
130	0	11	284	1	42	840	0	1	150	89	0	11	810
140	0	7	835	0	41	327	0	1	270	89	0	10	768
150	0	6	861	0	26	570	0	1	290	90	0	5	705
160	0	8	767	0	24	690	0	1	146	90	0	3	516
100													
100	2	23	934	6	4	753	0	35	917	93	1	10	532
110	1	16	379	4	28	50	0	4	137	94	1	0	458
120	0	22	480	1	21	500	0	2	173	97	0	14	892
130	0	17	641	0	53	314	0	1	670	98	0	13	868

Tabla III (Continuación)

n	't			t máx			t min			Heu	σ		
m	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
100													
140	0	15	420	1	2	580	0	1	153	99	0	14	339
110													
110	3	58	638	12	25	275	1	42	265	101	1	50	235
120													
120	4	32	523	23	45	757	1	21	856	111	1	36	289

M Minutos S Segundos MMS Milésimas de Segundos

Tabla IV

n	't			t máx			t min			Heu	σ		
m	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
10													
10	0	1	657	0	9	377	0	0	80	9	0	2	579
20													
20	0	5	354	0	49	103	0	0	140	18	0	9	280
30	0	2	975	0	43	163	0	0	144	19	0	8	536
40	0	1	472	0	13	693	0	0	173	20	0	2	553
50	0	1	13	0	6	797	0	0	196	20	0	1	522
60	0	3	567	0	34	207	0	0	420	20	0	5	943
70	0	2	736	0	9	740	0	0	524	20	0	2	729
80	0	3	827	0	13	430	0	0	763	20	0	3	145
90	0	4	714	0	28	40	0	0	736	20	0	4	571
100	0	14	325	1	31	990	0	2	670	20	0	5	724
30													
30	0	5	959	0	25	536	0	0	473	27	0	5	226
40	0	3	100	0	11	306	0	0	190	29	0	3	297
50	0	2	335	0	10	596	0	0	210	29	0	2	843
60	0	1	556	0	6	426	0	0	290	30	0	1	908
70	0	2	144	0	7	657	0	0	384	30	0	2	308
80	0	3	960	0	26	750	0	0	530	30	0	4	472
90	0	6	768	0	44	327	0	0	774	30	0	8	320
100	0	6	570	0	33	160	0	0	787	30	0	5	886
40													
40	0	10	108	0	32	517	0	0	457	37	0	7	36
50	0	6	727	0	26	174	0	0	307	38	0	6	130
60	0	5	539	0	25	770	0	0	346	39	0	8	972
70	0	3	509	0	14	460	0	0	283	40	0	5	119
80	0	5	918	0	16	557	0	0	466	40	0	3	503
90	0	7	708	0	29	923	0	0	597	40	0	4	642
100	0	5	151	0	10	500	0	0	486	40	0	2	868
110	0	8	817	0	17	614	0	1	663	40	0	3	843
120	0	17	531	1	17	506	0	2	161	40	0	13	492

Tabla IV (Continuación)

n	t			t máx			t min			Heu	σ		
m	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
50													
50	0	15	417	1	1	80	0	1	580	46	0	10	696
60	0	11	845	0	59	420	0	0	327	47	0	10	86
70	0	4	45	0	20	834	0	0	343	49	0	5	489
80	0	3	733	0	10	717	0	0	360	49	0	3	268
90	0	3	197	0	11	386	0	0	823	50	0	2	333
100	0	4	741	0	20	767	0	1	210	50	0	3	778
110	0	6	77	0	32	0	0	1	777	50	0	4	746
120	0	5	687	0	13	877	0	1	340	50	0	2	917
130	0	9	902	1	18	950	0	2	853	50	0	11	610
140	0	10	527	0	42	556	0	2	213	50	0	7	954
150	0	7	988	0	22	557	0	1	420	50	0	4	24
160	0	18	599	0	51	543	0	3	660	50	0	16	456
60													
60	0	21	734	0	26	777	0	2	850	55	0	6	400
70	0	18	461	1	0	247	0	0	527	57	0	15	414
80	0	15	259	1	8	450	0	0	600	58	0	14	908
90	0	9	171	0	34	580	0	0	636	59	0	8	317
100	0	8	774	0	27	770	0	0	693	59	0	7	0
110	0	6	679	0	26	444	0	0	793	60	0	5	35
120	0	10	958	0	42	664	0	0	863	59	0	9	337
130	0	10	492	0	25	920	0	1	390	60	0	6	14
140	0	13	331	0	40	64	0	1	133	60	0	8	478
150	0	16	855	1	10	244	0	1	673	60	0	12	360
160	0	22	530	0	53	470	0	2	277	60	0	9	987
70													
70	0	32	19	1	38	503	0	3	317	65	0	21	776
80	0	17	449	0	49	540	0	0	497	66	0	12	971
90	0	11	890	0	55	973	0	0	554	68	0	12	84
100	0	8	377	0	36	720	0	0	603	69	0	8	72
110	0	6	532	0	19	220	0	0	580	69	0	4	619
120	0	4	847	0	18	533	0	0	753	70	0	4	959
130	0	6	962	0	17	554	0	0	640	69	0	4	506
140	0	8	412	0	34	534	0	1	180	70	0	6	32
150	0	9	77	0	24	393	0	0	800	70	0	5	826
160	0	12	202	0	34	337	0	1	77	70	0	7	302
80													
80	1	4	663	3	51	720	0	15	770	74	0	33	879
90	0	47	670	5	25	14	0	0	770	76	0	56	974
100	0	21	633	1	44	990	0	0	873	78	0	19	892
110	0	12	739	0	40	33	0	0	870	79	0	11	557
120	0	10	897	0	52	47	0	1	330	79	0	10	699
130	0	14	290	1	17	131	0	1	167	79	0	15	679
140	0	11	677	0	41	447	0	1	316	80	0	8	201
150	0	19	195	1	6	777	0	2	156	80	0	14	144
160	0	20	475	1	4	327	0	2	310	80	0	13	409

Tabla IV (Continuación)

n	`t			t máx			t min			Heu	σ		
m	M	S	MMS	M	S	MMS	M	S	MMS		M	S	MMS
90													
90	1	58	322	6	45	447	0	19	340	83	1	37	472
100	0	37	883	2	5	250	0	3	656	86	0	34	188
110	0	37	872	3	2	270	0	1	943	86	0	36	962
120	0	32	975	2	20	156	0	0	780	87	0	34	560
130	0	30	330	1	25	875	0	0	954	87	0	32	766
140	0	29	356	0	54	278	0	0	877	87	0	31	42
150	0	28	388	0	55	548	0	0	963	88	0	30	462
160	0	26	977	0	45	951	0	0	649	88	0	29	241
100													
100	2	5	915	5	56	147	0	12	124	93	0	34	413
110	0	30	491	3	15	90	0	6	590	89	0	31	929
120	0	30	468	3	20	394	0	14	807	89	0	30	690
130	0	29	958	2	55	465	0	12	587	90	0	29	752
140	0	29	280	1	49	863	0	10	177	91	0	29	185
110													
110	3	6	408	6	5	250	0	3	656	102	1	34	188
120	0	40	320	5	5	194	0	1	187	104	0	55	736
120													
120	3	32	882	8	40	783	0	37	287	111	0	40	157

M Minutos S Segundos MMS Milésimas de Segundos

3.3.2 Gráficas de Resultados

Las gráficas de resultados se presentan a continuación. En todas se muestran los tiempos promedio de corrida (`t) en segundos de los algoritmos presentados en este trabajo y que se muestran en las tablas de resultados.

Gráfica I.

Como el Algoritmo de Gross sólo fué posible correrlo hasta con matrices

de 80x80, se presenta la comparación de los cuatro algoritmos con matrices cuadradas tomando como máximo este rango. Como se puede ver, el agoritmo de Gross presenta una tendencia exponencial conforme aumenta **m**, aunque para **m** ≤ 50 el tiempo promedio es el mínimo. Garfinkel sigue una tendencia lineal para **m** ≤ 80, llegando a ser el tiempo promedio intermedio para **m** ≤ 40 pero se eleva mucho a medida que **m** crece. En Zimmermann se observa que para matrices con **m** ≤ 50 el tiempo promedio es el máximo, disminuyéndose para **m** grande. Romero sigue un comportiamiento estándar y baja para **m** = 50, 60, 70 elevándose después.

Gráfica II.

Aquí se presentan los tiempos promedio para matrices cuadradas para los algoritmos que fue posible correr con **m** ≥ 80, en ella se observa que el algoritmo de Garfinkel resulta ser el máximo siempre; Zimmerman baja para **m** = 80, 90, elevándose después en forma casi exponencial. Romero baja notablemente, y se mantiene para **m** = 90, 100, 110, 120.

Las siguientes gráficas se tomaron de los algoritmos que se pueden aplicar para matrices no cuadradas (Derigs - Zimmermann y Romero), presentando para cada uno de ellos el tiempo promedio en segundos de las tablas anteriormente presentadas.

Gráfica III y IV

Tiempos promedio para todas las corridas de los algoritmos tomando el recorrido de **m** y **n** de 10 en 10 hasta llegar los máximos **m** y **n** registrados (90x160). Como podemos ver, los "picos" de la gráfica

representan el tiempo promedio (que es el máximo) para matrices cuadradas, disminuyéndo notablemente, pero elevándose un poco cuando **n** es aproximadamente igual a 2 veces **m** para casi todos los casos.

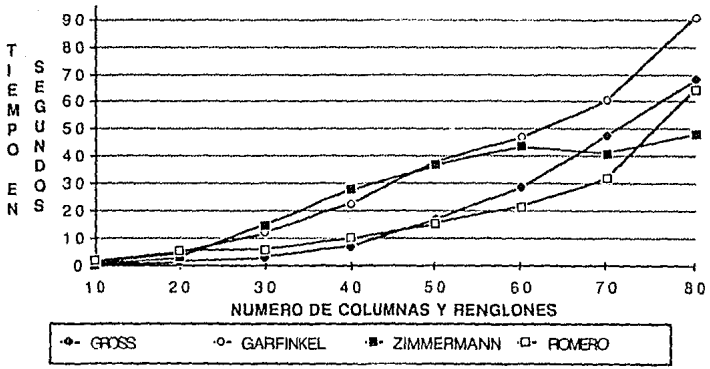
Gráfica V y VI.

Tiempos promedio tomando el recorrido de **m** y **n** de 20 en 20 hasta llegar a 120x120.

Gráfica VII y VIII.

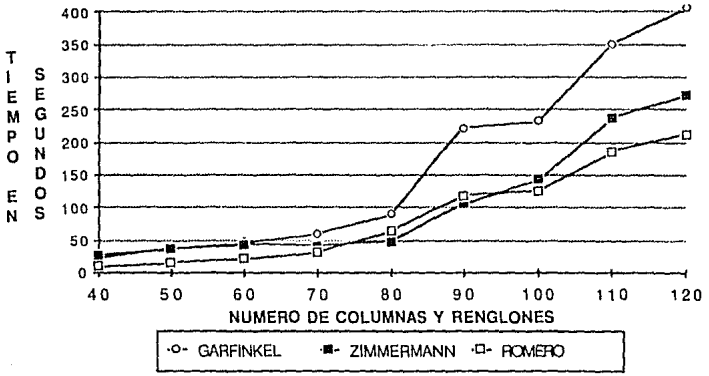
Tiempos promedio de las gráficas anteriores acortando el máximo a 60 y 70 segundos respectivamente.

TIEMPOS PROMEDIO DE CORRIDA PARA MATRICES CUADRADAS



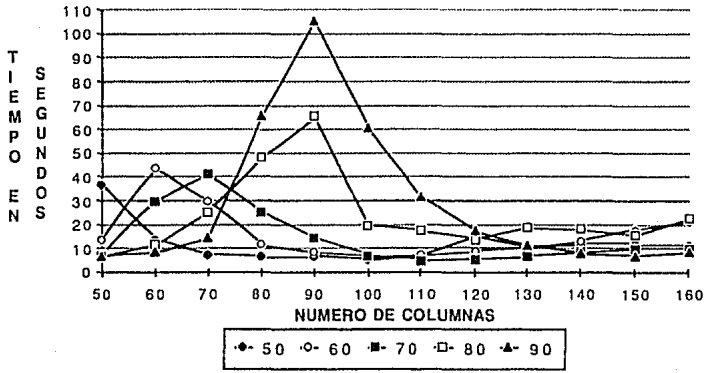
Gráfica I

TIEMPOS PROMEDIO DE CORRIDA PARA MATRICES CUADRADAS



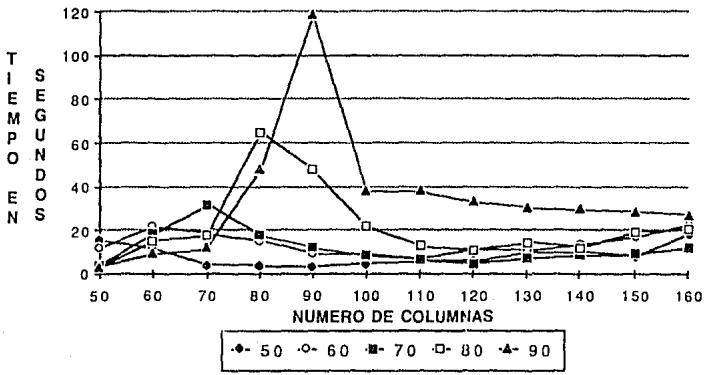
Gráfica II

ALGORITMO DE DERIGS - ZIMMERMANN



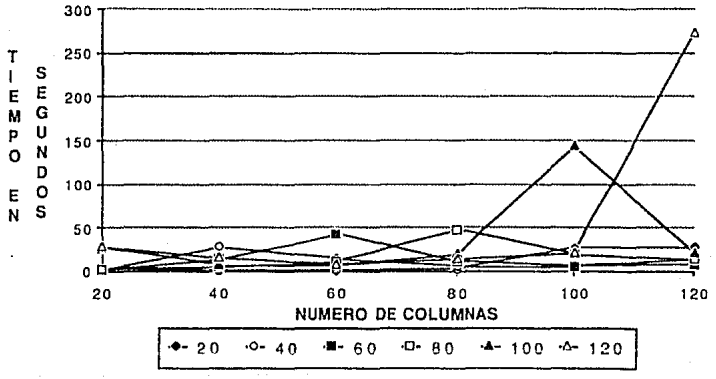
Gráfica III

ALGORITMO DE ROMERO



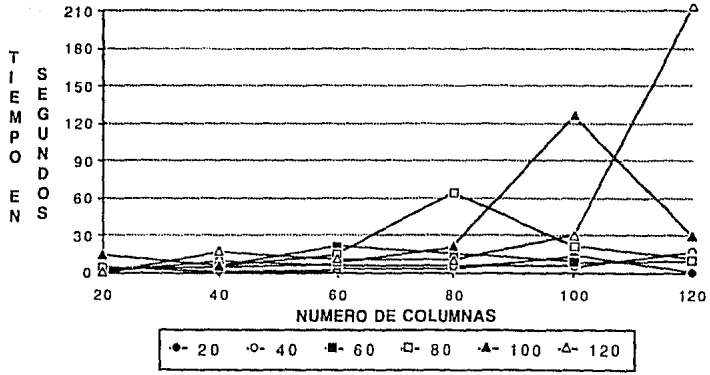
Gráfica IV

ALGORITMO DE DERIGS - ZIMMERMANN



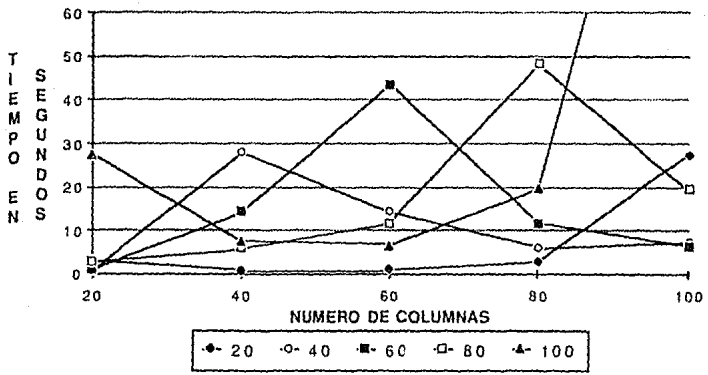
Gráfica V

ALGORITMO DE ROMERO

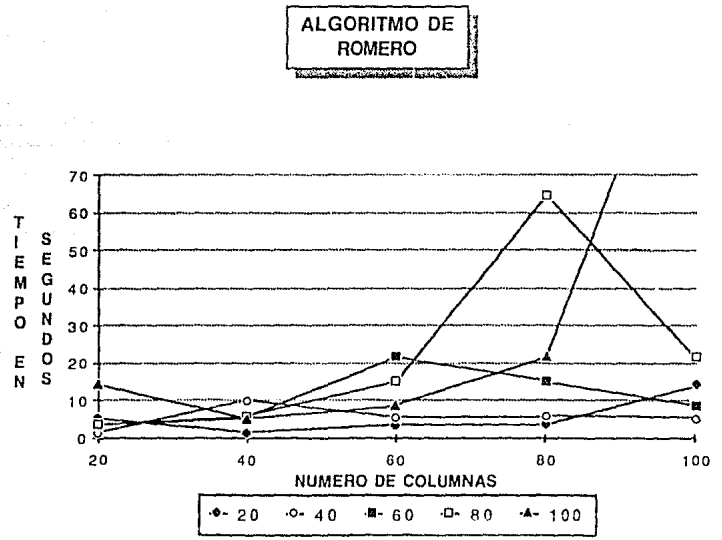


Gráfica VI

ALGORITMO DE DERIGS - ZIMMERMANN



Gráfica VII



Gráfica VIII

4. CONCLUSIONES

El Problema de Asignación Cuello de Botella es un problema de optimización combinatoria, el cual tiene aplicación a diversas situaciones. En el presente trabajo tras exponerlo y analizarlo se ha hecho una comparación computacional de cuatro algoritmos para su resolución.

Los algoritmos se analizaron con matrices cuadradas y no cuadradas, encontrando que su comportamiento resultó ser distinto cuando se aplican a matrices de diferente tamaño.

De los algoritmos codificados, se encontró que para matrices cuadradas, los que presentaron menores tiempos de corrida fueron: para $m \leq 50$: Gross, para $m = 60,70$: Romero, para $m = 80,90$: Derigs-Zimmermann, y para $m \geq 100$: Romero.

Nuestros experimentos nos permitieron observar una aparente contradicción, es decir, encontramos que, para m fija, al aumentar el valor de n los tiempos de corrida no necesariamente aumentaban. Al contrario, éstos disminuyen hasta que $n \approx 2m$, empezando a aumentar ligeramente a partir de este valor.

Para matrices no cuadradas los resultados obtenidos nos señalan que el

algoritmo más rápido es el de Romero, a excepción de $m=80,90$, donde los menores tiempos de corrida se obtuvieron por el método de Derigs-Zimmermann.

En cuanto a la codificación, podemos decir que los cuatro algoritmos bajo estudio presentaron aproximadamente la misma dificultad, siendo el algoritmo de Garfinkel el menos fácil de codificar.

Finalmente, creemos que no es redundante insistir en la importancia de la Investigación de Operaciones para el análisis de sistemas complejos, especialmente en un mundo tan cambiante como el que vivimos y donde la optimización de resultados, reportados en forma acertada y oportuna, es determinante no sólo a nivel empresarial, sino también nacional.

Este fue el motivo que nos impulsó a trabajar en este tema. El contribuir en forma por demás modesta al avance de la metodología de la Investigación de Operaciones es nuestra satisfacción, ya que estamos convencidos de que el desarrollo de nuestro país depende de la unión de esfuerzos individuales de todos los que deseamos una vida mejor para los que nos rodean.

5. REFERENCIAS.

[1] Ackoff, Rusell L., *Science in the Systems Age: Beyond I.E., O.R., and M.S.,* Operations Research 21 (1973).

[2] Prawda, J., *Métodos y Modelos de Investigación de Operaciones,* Vol. I Modelos Determinísticos, México (1976).

[3] Rockafellar, R.T., *Network Flows and Monotropic Optimization,* A Wiley-Interscience Publication, New York.

[4] Fulkerson, D. R., Glieksberg, I. and Gross, O., *A Production Line Assignment Problem,* RAND Res, Mem, RM-1402, (1953).

[5] Gross, O., *The Bottleneck Assignment Problem,* Reporte P-1630, RAND Corp., (1959).

[6] Ford, L.R. Jr., and Fulkerson, D.R., *Flows in Networks,* Princeton University, Press, Princeton, N.J., (1962).

[7] Garfinkel, R. S., *An Improved Algorithm for the Bottleneck Assignment Problem,* Operations Research 19, (1971).

[8] Edmonds, J., and Fulkerson, D.R., *Bottleneck Extrema,* RM-5375-PR, The Rand Corporation, Santa Monica, Calif., (1968).

[9] U. Derigs and U. Zimmermann, K., *An Augmenting Path Method for Solving Linear Bottleneck Assignment Problems,* Computing 19, (1978).

[10] Dijkstra, E.W., *A Note on Two Problems in Connection with Graphs,* Numerische Mathematik 1, (1959).

[11] Romero, D., *Matrices with Prominent Staircases. an Extension to the Bottleneck Matching Problem,* En preparación.