Trabajo Práctico Nº 2

Distribución eléctrica

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

Primer Cuatrimestre - 2007

Guillermo Túnez, Matías Albanesi, Mercedes Bianchi

16 de julio de 2007

Nombre	LU	E-mail
Matías Albanesi Cáceres	522/00	mac@sion.com
María Mercedes Bianchi	685/97	mbianchi9@arnet.com.ar
Guillermo Túnez	790/00	guillermotunez@yahoo.com.ar

Extracto: Este trabajo se centra en el análisis de métodos para resolver sistemas de ecuaciones lineales y su aplicación en problemas de flujo en grafos, en particular sobre una red de Morley. Se desarrolló una variante de la eliminación gaussiana con pivoteo parcial que puede trabajar con sistemas indeterminados y se aplicó al problema de la red de Morley. Se estudió su performance y el número de condición de la matriz del sistema para estimar el error numérico cometido. El algoritmo produce un resultado gráfico que permite comprobar fácilmente la correctitud de la solución y compartirla con usuarios sin que necesiten conocer detalles de la implementación.

Palabras Clave: Sistema de ecuaciones lineales, Grafo, Red de Morley, Eliminación gaussiana, Error numérico.

Índice

1. Introducción

1.1. Sistemas de Ecuaciones Lineales

Un sistema lineal de ecuaciones consta de n ecuaciones sobre n incógnitas y puede expresarse en notación matricial como $A \times x = b$. Estas técnicas utilizan una serie finita de operaciones aritméticas para determinar la solución exacta del sistema, sujeta únicamente al error de redondeo.

Una propiedad importante acerca de la existencia o no de la solución del sistema es la existencia de una inversa A^{-1} de la matriz A; si existe el sistema tiene solución y además es única. Equivalentemente el sistema tiene solución si el determinante de la matriz A no es cero. En este caso también se dice que la matriz A es no singular.

La solución del sistema lineal es un vector x tal que $x = A^{-1}b$.

1.2. Método de Gauss para la resolución de sistemas de ecuaciones Lineales

El método de Gauss, conocido también como de triangulación o de eliminación gaussiana, nos permite resolver sistemas de ecuaciones lineales con cualquier número de ecuaciones y de incógnitas.

La idea es muy simple; por ejemplo, para el caso de un sistema de tres ecuaciones con tres incógnitas se trata de obtener un sistema equivalente cuya primera ecuación tenga tres incógnitas, la segunda dos y la tercera una. Se obtiene así un sistema triangular o en cascada.

Al resolver un sistema puede suprimirse, sin que varíe su resolución, cualquier ecuación que pueda obtenerse a partir de otras aplicando las siguientes operaciones:

- Producto o cociente por un número real distinto de cero.
- Suma o diferencia de ecuaciones.

Es importante añadir que el sistema resultante es dependiente de la forma en que apliquemos estos criterios, es decir, las ecuaciones obtenidas no son siempre las mismas, pero si lo hemos aplicado correctamente, el sistema resultante es equivalente al dado.

1.3. Solución de un Sistema

La solución de un sistema es cada conjunto de valores que satisface a todas las ecuaciones. Los tipos de sistemas pueden clasificarse en 3:

- Sistemas Compatibles
- Sistemas Compatibles Indeterminados
- Sistemas Incompatibles

1.3.1. Sistemas Determinados

Se llama sistema compatible determinado al que tiene una sola solución común, es decir, que sus rectas asociadas se cortan en un sólo punto, por lo tanto sus vectores de dirección son linealmente independientes.

1.3.2. Sistemas Compatibles Indeterminados

Se llama sistema compatible indeterminado al que tiene infinitas soluciones, es decir, que sus rectas asociadas son idénticas, por lo tanto sus ecuaciones son equivalentes.

1.3.3. Sistemas Incompatibles

Se llama sistema incompatible al que no tiene ninguna solución, es decir, que sus rectas asociadas son paraleleas, por lo tanto tiene algún vector de dirección linealmente dependiente.

1.4. Norma Matricial

Una norma matricial sobre el conjunto de todas las matrices de $n \times n$ es una función de valor real, $\|*\|$, definida en este conjunto y que satisface para todas las matrices A y B de $n \times n$ y todos los números reales α:

- **(i)** $\|A\| \geq 0$.
- **(ii)** $\|A\| = 0$, si y sólo si A es 0, la matriz con todas las entradas cero.
- **(iii)** $\|\alpha A\| = \mid \alpha \mid \|A\|$.
- **(iv)** $\|A + B\| \leq \|A\| + \|B\|$.
- **(v)** $\|AB\| \leq \|A\|\|B\|$.

1.5. Número de Condición

El número de condición de una matriz no singular A relativo a la norma $\|*\|$ es

$$k(A) = \|A\|.\|A^{-}1\|$$

Una matriz A es bien condicionada si $k(A)$ está cerca a 1 y es mal condicionada si $k(A)$ es significativamente mayor a 1. Dentro de este contexto, la condición de la matriz es una medida de la seguridad relativa de que al tener un vector residual pequeño implique que la solución obtenida sea cercana a la solución exacta.

2. Desarrollo

2.1. Análisis del problema

2.1.1. Modelo matemático

Como paso previo al diseño de una solución en computadora del problema, analicemos el problema y sus implicaciones.

Una *red de Morley* es un sistema interconectado de centrales generadoras de electricidad y nodos concentradores, diseñado para transportar electricidad y abastecer la demanda de todos los usuarios de la red.

Puede modelarse matemáticamente con un grafo, donde los nodos son de dos tipos distintos: concentradores y generadores. Hay ejes entre nodos del mismo tipo o de tipo distinto. Los ejes entre nodos concentradores forman el *backbone* de la red.

Las centrales generadoras producen electricidad, pero también abastecen a sus usuarios, cuyo consumo puede superar (o ser inferior, o igual) a la producción de la central. La diferencia entre la producción y la demanda es el balance, que puede ser positivo, nulo o negativo. Desde el punto de vista matemático, balance es el único dato relevante de una central generadora y nos abstraemos de su producción real y la demanda de sus usuarios directos.

Una demanda que supera la producción implica un balance negativo para una central generadora, y debe cubrirse con energía suministrada por otras centrales que tengan balance positivo.

Hay algunas condiciones adicionales importantes:

- Las centrales generadoras que tienen balance positivo entregan todo su excedente a la red.
- Las centrales generadoras que tienen balance negativo cubren completamente su déficit con energía suministrada por centrales con balance positivo a la red eléctrica.
- Los nodos concentradores transmiten toda la energía que reciben.
- *(Restricción de Morley):* El flujo de energía es idéntico en cada eje del *backbone*.

En resumen, el problema consiste en determinar cuanta energía debe enviarse a través de las interconexiones (ejes del grafo) entre nodos (concentradores o generadores) para cubrir totalmente la demanda.

Notar que estas condiciones implican que no hay exceso de producción de energía globalmente; o sea

$$\sum_{i=1}^{n} b_i \quad = \quad 0$$

siendo b_i el balance del nodo i-ésimo del grafo, y definiendo para los nodos concentradores un balance cero (porque transmiten toda la energía que reciben y no la generan).

2.1.2. Planteo como sistema de ecuaciones lineales

Primero debemos encontrar un planteo apropiado para resolver el problema mediante un sistema de ecuaciones lineales. Nuestras incógnitas son los flujos de energía a asignar a cada eje del grafo.

Las condiciones (i) y (ii) nos proporcionan las ecuaciones:

$$\sum_{\substack{e_j: \\ \text{eje saliente} \\ \text{del nodo i}}} flujo(e_j) - \sum_{\substack{e_k: \\ \text{eje entrante} \\ \text{al nodo i}}} flujo(e_k) \quad = \quad b_i$$

Estas ecuaciones equivalen a:

$$+x_{j_1} - x_{j_2} + x_{j_3} - x_{j_4} + ... \quad = \quad b_i$$

donde x_{j_k} es el flujo de un eje incidente al nodo i-ésimo, y b_i es el balance del nodo i-ésimo. Como puede verse, se coloca un signo positivo si el eje es "saliente" del nodo y signo negativo si el eje es "entrante".

Resolver un sistema formado por esas ecuaciones cumpliría con las condiciones (i), (ii) y también (iii) definiendo para los nodos concentradores un balance cero.

La condición (iv) puede expresarse como:

$$|x_{j_p}| \quad = \quad |x_{j_q}|$$

para cada par de ejes j_p y j_q, $p \neq q$, que formen parte del backbone. Notar que al igualar los valores absolutos de los flujos de los ejes del backbone, no podemos incorporar estas condiciones directamente a un sistema de ecuaciones lineales.

2.1.3. Problemas encontrados

De la sección anterior surgen estos problemas:

- Necesitamos asignar a los ejes una dirección. Saber si son "salientes" o "entrantes".
- Las ecuaciones que cumplen con la condición (iv) del enunciado no pueden incorporarse a un sistema de ecuaciones lineales directamente.

Para resolver el primer punto pensamos inicialmente establecer dos ejes entre cada nodo, uno entrante y otro saliente, y poner a ambos como incógnitas.

Pero eso no es necesario: puede modelarse con sólo un eje orientado entre nodos. Si al resolver el sistema de ecuaciones el valor de flujo para un eje resultara negativo, querría decir que ese eje debería tener el sentido opuesto. Si fuera cero o positivo el eje conservaría su sentido inicial. Entonces el sentido inicial puede ser cualquiera, y se corrige al tener la solución del sistema.

El segundo punto es un obstáculo más importante. Finalmente decidimos asumir que nuestra implementación sólo puede resolver el problema con un backbone donde todos los ejes tienen el mismo sentido. Esto puede definirse sin problemas sólo en el caso de que el backbone sea lineal, por lo tanto también pedimos esta condición en el grafo de entrada.

De no asumirse esas condiciones sobre el backbone, habría que resolver 2^n sistemas, cada uno compuesto por las ecuaciones derivadas de las condiciones (i) a (iii), más la ecuación

$$x_{j_p} = x_{j_q}$$

o

$$x_{j_p} = -x_{j_q}$$

para cada par de ejes del backbone.

2.2. Análisis previo al desarrollo en computadora

2.2.1. Plataforma y compilador

La primer decisión a tomar consiste en plataforma y compilador a utilizar. Como el desarrollo debe cumplir con el requisito de poder compilarse y ejecutarse con el software disponible en los laboratorios del Departamento de Computación, optamos por computadoras con procesadores compatibles con la arquitectura Intel x86, y el compilador g++ de la Free Software Foundation.

2.2.2. Alternativas de representación

Como se comentó en la sección anterior, inicialmente se había planteado establecer dos ejes entre cada par de nodos conectados en el problema de entrada, con sentidos opuestos. Pero una solución equivalente es tener sólo un eje entre dos nodos conectados, y hacerlo dirigido dentro del programa, al leer la instancia del problema. El sentido asignado puede ser cualquiera: una vez resuelto el sistema, si el valor del flujo en el eje queda negativo, se invierten el sentido del eje y el valor del flujo eléctrico.

Planteada esta necesidad, diseñamos una clase *template* Grafo simple, que representa a un grafo dirigido con información arbitraria asignada a los nodos y a los ejes. A los nodos se asignó una estructura de datos que indica el tipo de nodo (Nodo Concentrador o Central Generadora) y el balance si es una central generadora (valor en punto flotante). Y a los ejes se asigna el flujo (este dato se asigna luego de resolverse el sistema de ecuaciones; inicialmente queda en cero y no es utilizado).

También se diseñó una clase Matriz cuyos elementos son **long double**, y soporta las operaciones estándar del álgebra de matrices, e implementa operaciones para resolver sistemas de ecuaciones mediante eliminación gaussiana con pivoteo parcial, como se describirá más adelante.

2.2.3. Planteo del sistema lineal de ecuaciones

Como ya adelantamos en la sección previa, se tomará una instancia del problema y se planteará un sistema lineal de ecuaciones. Las incógnitas serían los flujos de los ejes, por lo tanto el sistema tendrá tantas columnas como ejes existan en el grafo de entrada. Las ecuaciones que afectan a esos ejes son de dos tipos:

- Ecuaciones relativas al balance de cada nodo: son las que cumplen con las condiciones (i), (ii) y (iii) del Enunciado. Las ecuaciónes establecen que la suma de los flujos salientes de un nodo menos la suma de los flujos entrantes al nodo debe corresponder exactamente al balance del nodo; trasladado a la matriz, esto define una fila para cada nodo, donde hay un valor $+1$ si un eje es saliente del nodo, -1 si el eje es entrante

al nodo, y 0 si no es incidente al nodo. El valor colocado en la matriz b de valores es justamente el balance del nodo. Se define el balance de un Nodo Concentrador como 0. Hay una fila por cada nodo.

- Ecuaciones que establecen la restricción de Morley: son las que cumplen la condición (iv) del Enunciado, o sea, dicen que el flujo es idéntico en cada eje del *backbone*. Estas ecuaciones se plantearon como una cadena de igualdades: el flujo del primer eje del backbone es igual al flujo del segundo; el flujo del segundo es igual al del tercero; etc. En la matriz del sistema lineal, se agrega una fila por cada una de estas igualdades, y la forma general es colocar un $+1$ en la columna del eje *i-ésimo* y -1 en la columna del eje siguiente, dejando los demás valores de la fila en 0 y asignando un valor 0 en la matriz b de valores. Esta fila equivale a la ecuación $x_i = x_{i+1}$ o equivalentemente $x_i - x_{i+1} = 0$.

Ya que (en el peor caso) para un grafo de m ejes se tendrían que todos forman parte del backbone, inicialmente reservamos $m - 1$ filas para las ecuaciones de restricciones de Morley, además de n filas para cada ecuación de conservación de balance de los nodos. Esta (incorrecta) decisión de tener una matriz no cuadrada trajo el grave perjuicio de asignar y utilizar mucha más memoria de la necesaria, lo cual afectó al tiempo de cálculo y nos impidió implementar el cálculo del número de condición dentro del programa. Más tarde se modificó la lectura de instancias del problema para utilizar menos memoria y tener una matriz cuadrada. Se comentará sobre este punto en la sección Discusión.

2.2.4. Entrada y salida

Diseñamos un formato muy simple de entrada para el programa: numerar cada nodo y cada eje, declarar si un nodo es Nodo Concentrador o Central Generadora, y en el segundo caso especificar el balance (positivo, cero o negativo) con un valor en punto flotante.

En cambio la salida del programa fue evolucionando a medida que se encontraban inconvenientes. Inicialmente se hizo un volcado de los balances de los ejes, pero esto resulta insuficiente para expresar la orientación de los ejes, y más aún, es de utilidad sólo para depurar el algoritmo y usar instancias pequeñas del problema.

Con esta necesidad en mente se buscó la manera de producir un resultado más útil, y que permita verificar instancias grandes del problema. Luego de hacer distintas consultas, encontramos el software para manipulación de grafos Visone (www.visone.info) que permite importar grafos en una variedad de formatos de entrada, y asignar posición, forma y color a nodos y ejes, permitiendo generar un resultado visual satisfactorio que nos permitió:

- producir la representación de los grafos en este informe

- verificar visualmente la correctitud del algoritmo, incluso con instancias muy grandes.

Evaluamos los formatos de entrada de Visone, y encontramos que el formato Pajek Graph es relativamente sencillo de generar desde nuestra implementación y a la vez lo suficientemente expresivo, ya que permite asignar a los nodos una forma, tamaño, color, texto, y a los ejes un sentido y un texto. Con estos elementos, el algoritmo puede producir una diferencia visual (en el color) entre nodos concentradores y los nodos correspondientes a centrales, y dibujar el sentido correcto y el flujo asignado a todos los ejes del grafo.

Más tarde surgió la necesidad de generar instancias de prueba de gran tamaño para medir el rendimiento de la implementación, y encontrar el tamaño máximo del problema que puede resolverse en 10 minutos (como se pidió en el Enunciado). Visone puede generar grafos aleatoriamente y exportarlo en distintos formatos. Desafortunadamente encontramos inconvenientes al exportar grafos grandes en formato Pajek Graph (sospechamos que se debió a falta de memoria para trabajar) y encontramos otro formato, UCINET DL, fácil de procesar para leer grafos.

A la implementación entonces se agregó un módulo que convierte grafos en formato UCINET DL en instancias de entrada válidas del problema, utilizando el texto escrito en los nodos como indicación del tipo de nodo (Nodo Concentrador o Central Generadora) y el balance del nodo (sólo para centrales).

Resultaba insuficiente para grafos grandes asignarle un tipo a todos los nodos, incluso usando las herramientas de manipulación masiva de nodos de Visone. Por lo tanto se modificó el módulo de importación de grafos UCINET DL para asumir que nodos sin tipo y balance sean centrales generadoras con balance cero.

2.2.5. Cálculo del número de condición

Se optó por realizar el cálculo del número de condición fuera del programa, usando el software Matlab (www.mathworks.com), presente en los laboratorios de la Facultad. Era posible realizarlo dentro del programa, pero teníamos un inconveniente: la matriz sobre la cual aplicaríamos el cálculo no sería cuadrada. Con lo cual no podríamos utilizar ninguna implementacíon de número de condición que dependiera de encontrar la inversa de una matriz. Matlab resuelve el problema, permitiendo el cálculo del número de condición mediante la norma-2 de cualquier matriz, inclusive si no son cuadradas.

Se modificó la implementación para que exporte (opcionalmente) la matriz del sistema de ecuaciones en formato CSV (valores separados por comas) que puede importarse fácilmente en Matlab para calcular el número de condición.

Como se verá en la sección siguiente, no contamos inicialmente con una matriz que represente un sistema determinado, o al menos, no puede garantizarse. Depende del problema de entrada. Para no obtener un número de condición infinito o muy alto con Matlab (señal de que la matriz no es inversible) se realiza la exportación luego de que la implementación asegura que el sistema de ecuaciones es determinado, o sea, que la matriz es inversible.

2.2.6. Algoritmo para resolver el sistema lineal de ecuaciones

Desde el primer momento optamos por implementar el clásico algoritmo de eliminación gaussiana con sustitución hacia atrás, sin embargo hay distintas variantes disponibles.

Una técnica que combina simplicidad y reduce el error numérico es la eliminación gaussiana con pivoteo parcial [ver Burden, cap. 6, algoritmo 6.2], que finalmente fue elegida para nuestra implementación.

Sin embargo, hay una cuestión importante a tener en cuenta: la matriz asociada al sistema lineal no es cuadrada[1]: salvo instancias muy sencillas, tenemos siempre más ecuaciones que incógnitas. Lejos de representar a un sistema incompatible, encontramos que las instancias del problema corresponden a sistemas indeterminados; la eliminaciíon gaussiana tradicional se detendría con error porque para alguna columna, no habría ningún elemento candidato a ser pivote (todos los valores de dicha columna a partir de esa fila en adelante serían 0)[2].

Por lo tanto nos alejamos del algoritmo tradicional aplicando estos cambios:

- Si para una fila no se encuentra un elemento pivote, porque todos los valores en la columna desde esa fila hacia abajo están en cero, el algoritmo 'avanza' a la siguiente columna y vuelve a buscar un pivote. El efecto neto es que se produce una matriz 'cuasi-triangular', que puede llevarse a una matriz triangular insertando filas para definir el valor de la columna para la que no pudo encontrarse el pivote.

- Se separó la triangulación que produce la eliminación gaussiana de la fase final de sustitución hacia atrás, en dos métodos separados. El objetivo es tener la posibilidad de aplicar entre una fase y otra un proceso que defina valores para las columnas que hayan quedado sin un elemento pivote. Entonces, en nuestra implementación tenemos

[1] Debido a que inicialmente para un grafo de n nodos y m ejes reservamos $n + m - 1$ filas y m columnas. Al hacer pruebas con grafos con miles de ejes notamos la ineficiencia y cambiamos la implementación para que la matriz sea cuadrada

[2] Este inconveniente aún sería posible en la nueva implementación que define un sistema con una matriz cuadrada, ver sección Discusión

una eliminación gaussiana sin sustitución hacia atrás en el método *elimGauss* y una sustitución hacia atrás en el método *despejar*.

Se creó un método, al que denominamos *hacerDeterminado*, que agrega filas a la matriz resultante de *elimGauss*, para definir un valor a las columnas sin elemento pivote. El método agrega una fila donde todos los elementos son cero, salvo un valor 1,0 en la columna a definir; en la matriz b de valores se define el valor 0,0. Esto equivale a definir $x_i = 0$ para esa columna; es natural en el marco del problema suponer que si no se tiene información (restricciones) sobre el flujo de un eje, éste sea cero.

2.2.7. Errores de redondeo

Durante la etapa de desarrollo no se encontraron errores de redondeo; eso se debió a que tratamos con instancias muy pequeñas y con balances enteros. Al realizar mediciones con grafos grandes, con balances aleatorios y no enteros, se produjeron numerosos errores de redondeo que se manifestaban en que el sistema de ecuaciones sería incompatible. Por lo tanto se modificaron los métodos relativos a la resolución del sistema de ecuaciones para agregar una tolerancia: en lugar de hacer comparaciones por un valor exactamente igual a 0,0, se pidió que el valor se encuentre a una distancia máxima *EPSILON* de 0,0.

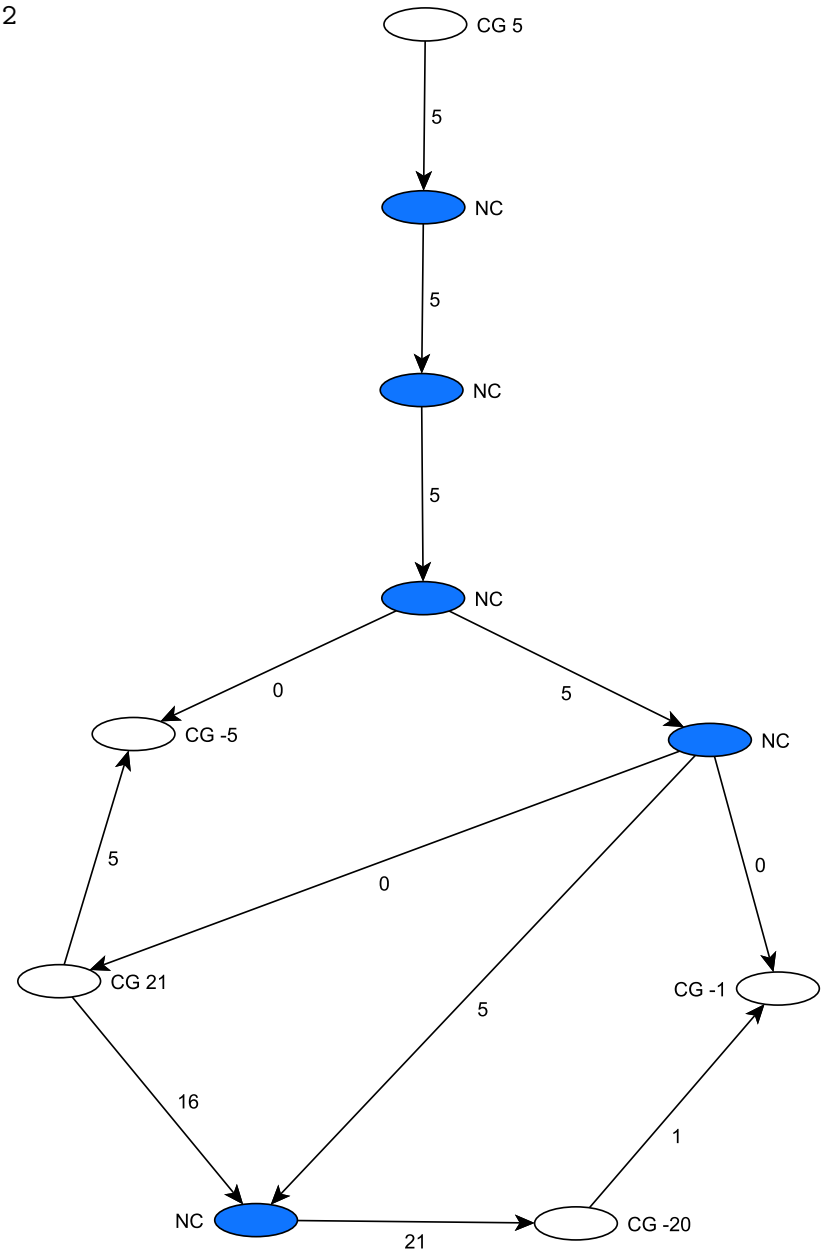
Adicionalmente toda vez que se detecte como resultado de un cálculo un valor cercano a 0, se almacena el valor exacto 0,0.

Esto ayudó a eliminar errores incluso en matrices con un número de error relativamente alto (el máximo que observamos en nuestras pruebas fue del orden de 9000).

3. Resultados

3.1. Grafo de prueba 1

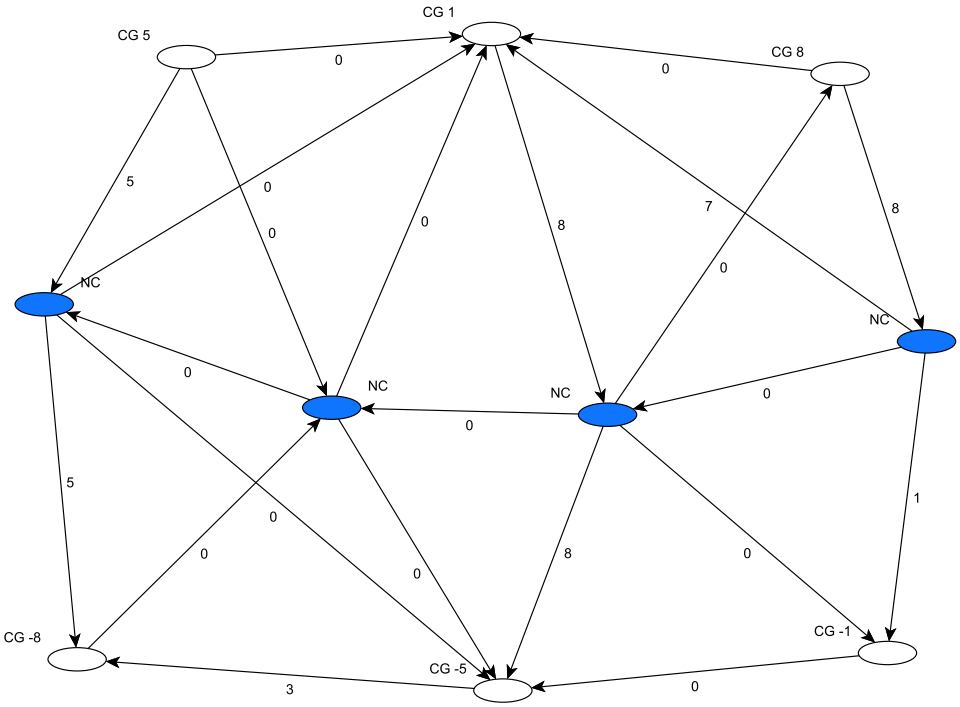
```
Init 10 12

CG 1 +5
NC 2
NC 3
NC 4
CG 5 -5
NC 6
CG 7 -1
CG 8 -20
NC 9
CG 10 +21

EJ 1 2
EJ 2 3
EJ 3 4
EJ 4 5
EJ 4 6
EJ 6 9
EJ 6 7
EJ 7 8
EJ 8 9
EJ 9 10
EJ 5 10
EJ 6 10

End
```



Cantidad de nodos	10
Cantidad de ejes	12

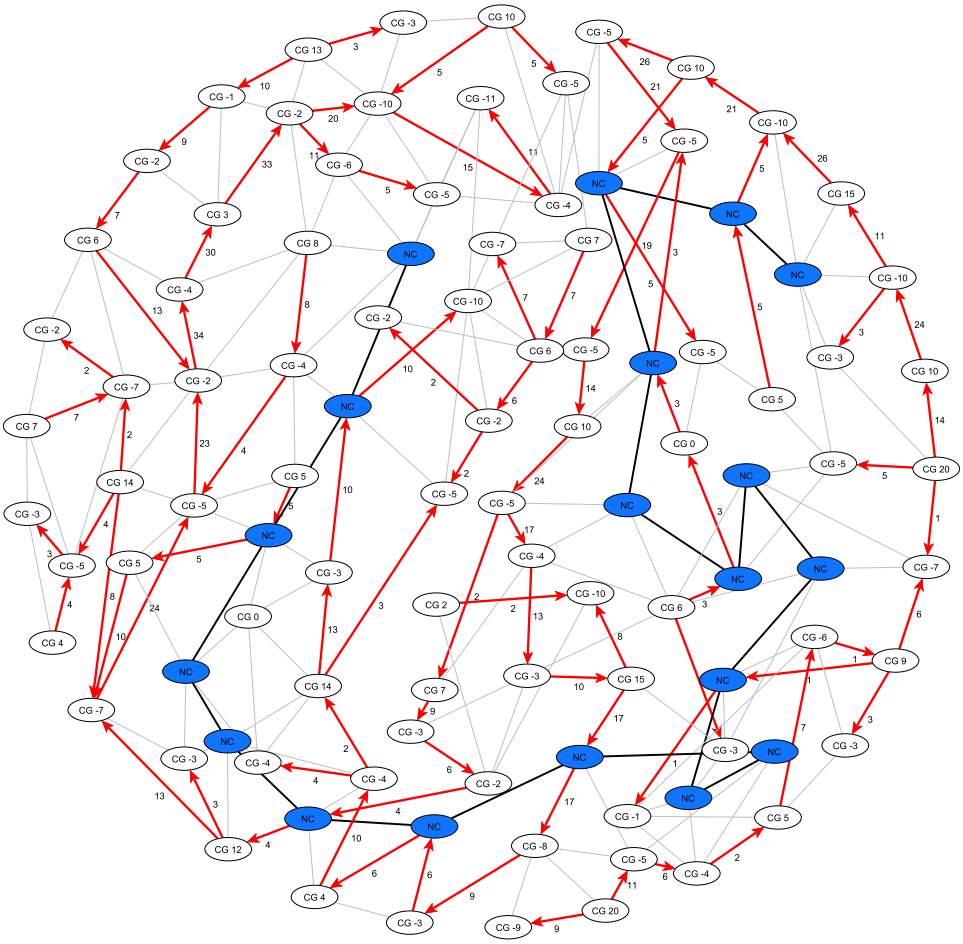
3.2. Grafo de prueba 2

```
Init 10 22          EJ 1 7              EJ 4 8
                    EJ 1 6              EJ 5 6
CG 1 -8             EJ 1 10             EJ 5 8
NC 2                EJ 2 4              EJ 5 7
NC 3                EJ 2 8              EJ 6 8
CG 4 8              EJ 2 3              EJ 6 7
CG 5 5              EJ 2 9              EJ 6 10
NC 6                EJ 3 8              EJ 7 8
NC 7                EJ 3 4              EJ 7 10
CG 8 1              EJ 3 9              EJ 9 10
CG 9 -1             EJ 3 10
CG 10 -5            EJ 3 7              End
```



Cantidad de nodos	10
Cantidad de ejes	22

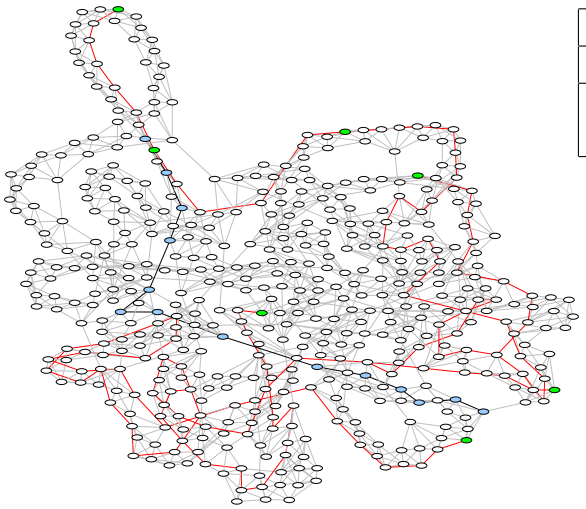
3.3. Grafo de 101 nodos



Tamaño	101 nodos por 224 ejes
Número de condición del sistema	161,20

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	323 filas por 224 columnas	4,23 seg
Método optimizado	224 filas por 224 columnas	0,16 seg

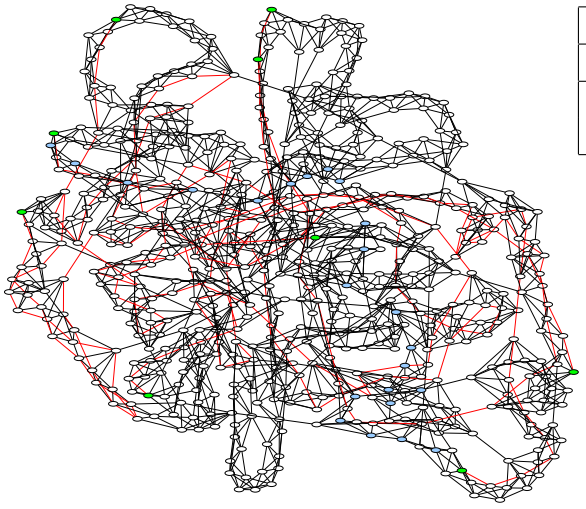
3.4. Grafo de 500 nodos



Cant. de nodos	500
Cant. de ejes	1497
Núm. de condición del sistema	318,94

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	1995 filas por 1497 columnas	24,61 seg
Método optimizado	1497 filas por 1497 columnas	16,4 seg

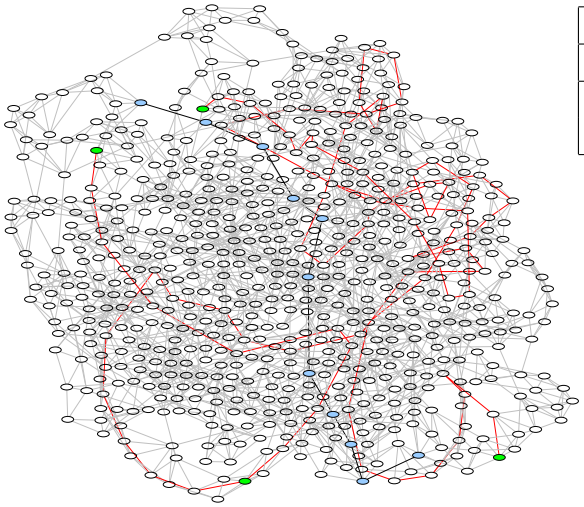
3.5. Grafo de 600 nodos



Cant. de nodos	600
Cant. de ejes	1796
Núm. de condición del sistema	4618,45

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	2394 filas por 1796 columnas	43,27 seg
Método optimizado	1796 filas por 1796 columnas	26,99 seg

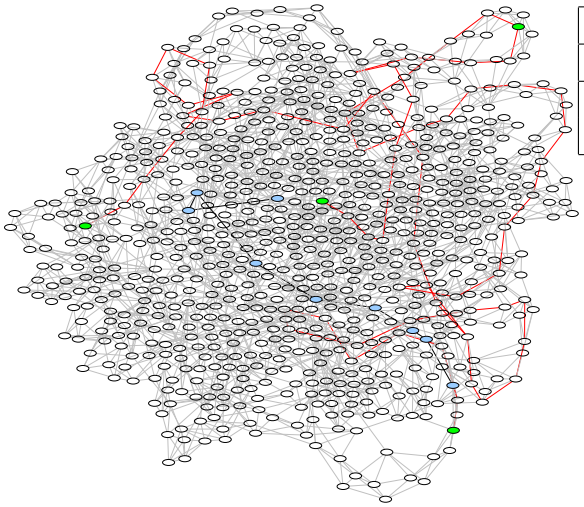
3.6. Grafo de 700 nodos



Cant. de nodos	700
Cant. de ejes	2094
Núm. de condición del sistema	3568,51

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	2792 filas por 2094 columnas	68,01 seg
Método optimizado	2094 filas por 2094 columnas	41,17 seg

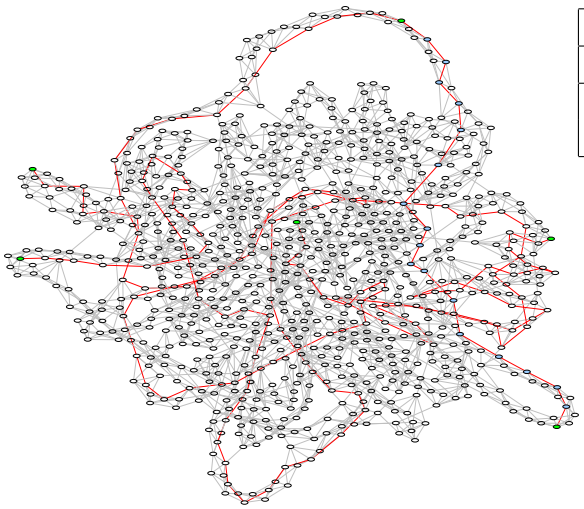
3.7. Grafo de 800 nodos



Cant. de nodos	800
Cant. de ejes	2397
Núm. de condición del sistema	3361,44

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	3195 filas por 2397 columnas	122,09 seg
Método optimizado	2397 filas por 2397 columnas	65,92 seg

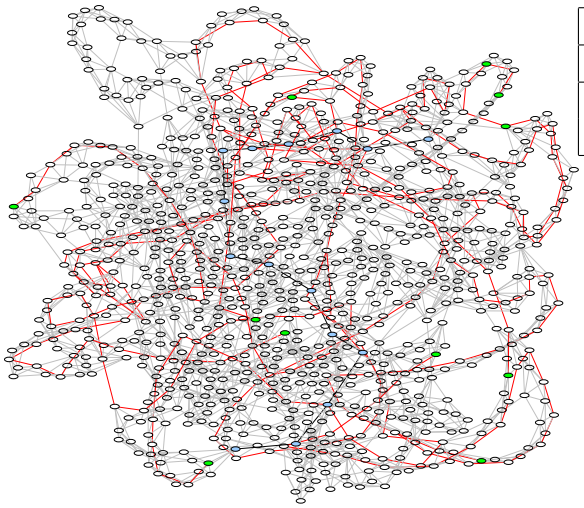
3.8. Grafo de 900 nodos



Cant. de nodos	900
Cant. de ejes	2696
Núm. de condición del sistema	1930,76

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	3594 filas por 2696 columnas	260,61 seg
Método optimizado	2696 filas por 2696 columnas	114,92 seg

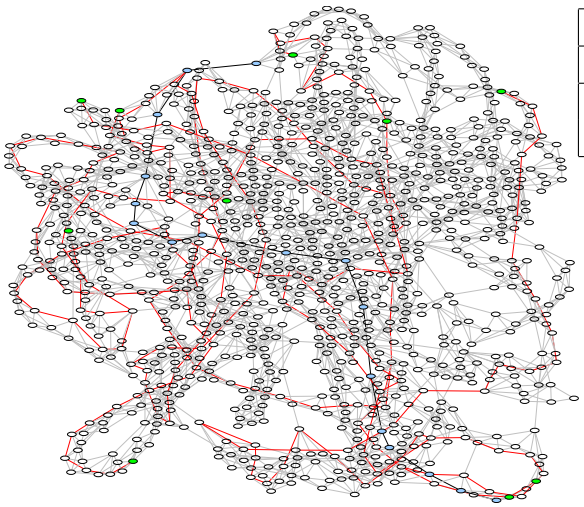
3.9. Grafo de 1000 nodos



Cant. de nodos	1000
Cant. de ejes	2996
Núm. de condición del sistema	5744,37

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	3994 filas por 2996 columnas	389,09 seg
Método optimizado	2996 filas por 2996 columnas	181,44 seg

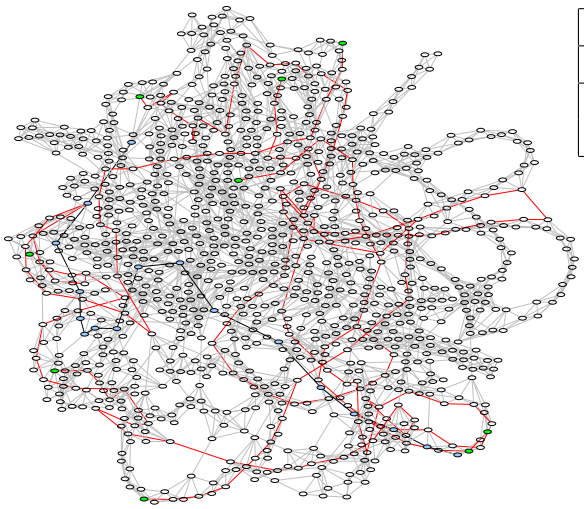
3.10. Grafo de 1100 nodos



Cant. de nodos	1100
Cant. de ejes	3200
Núm. de condición del sistema	4473,66

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	4298 filas por 3200 columnas	563,15 seg
Método optimizado	3200 filas por 3200 columnas	387,88 seg

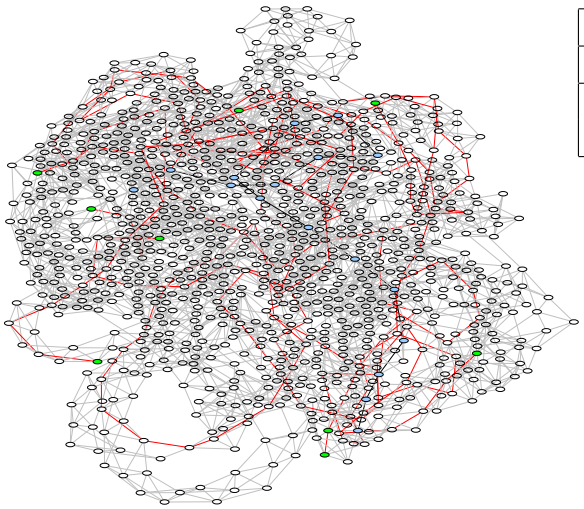
3.11. Grafo de 1120 nodos



Cant. de nodos	1120
Cant. de ejes	3328
Núm. de condición del sistema	5045,62

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	4446 filas por 3328 columnas	573,97 seg
Método optimizado	3328 filas por 3328 columnas	407,11 seg

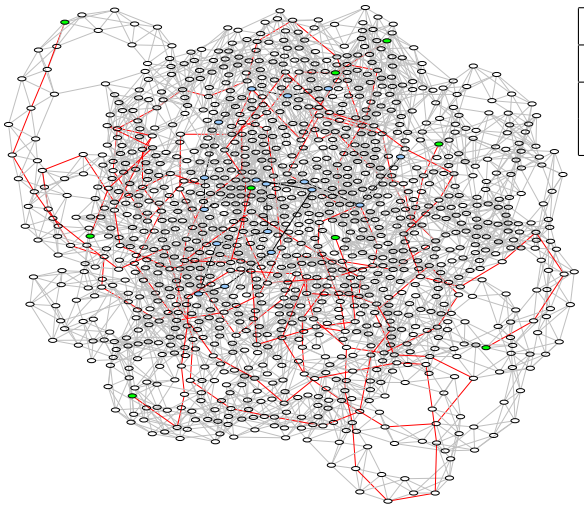
3.12. Grafo de 1130 nodos



Cant. de nodos	1130
Cant. de ejes	3359
Núm. de condición del sistema	4828,01

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	4487 filas por 3359 columnas	632,52 seg
Método optimizado	3359 filas por 3359 columnas	300,14 seg

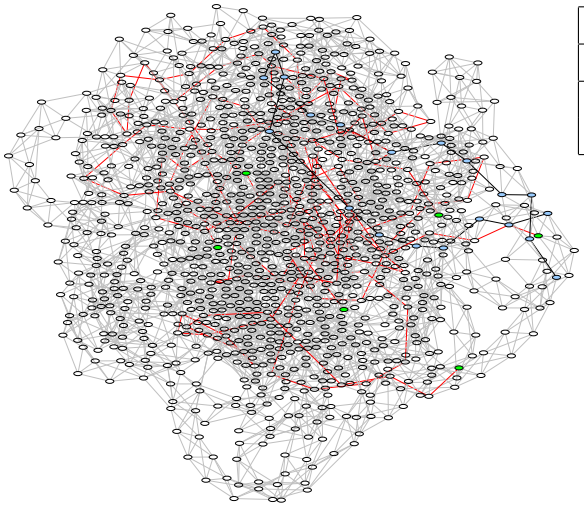
3.13. Grafo de 1150 nodos



Cant. de nodos	1150
Cant. de ejes	3447
Núm. de condición del sistema	4607,99

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	4595 filas por 3447 columnas	701,68 seg
Método optimizado	3447 filas por 3447 columnas	350,78 seg

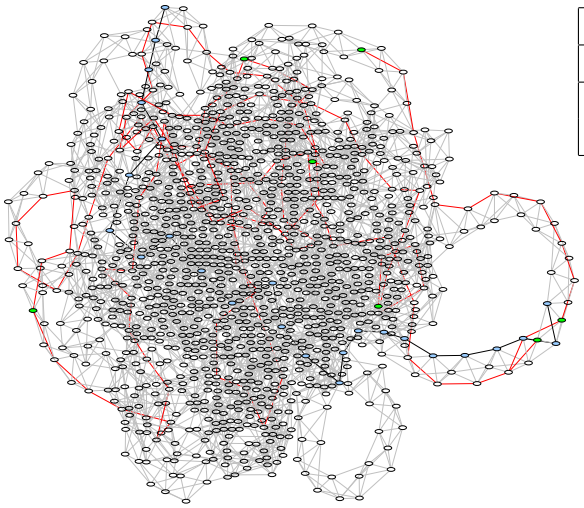
3.14. Grafo de 1200 nodos



Cant. de nodos	1200
Cant. de ejes	3596
Núm. de condición del sistema	3805,32

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	no disponible	no disponible
Método optimizado	3596 filas por 3596 columnas	413,28 seg

3.15. Grafo de 1300 nodos



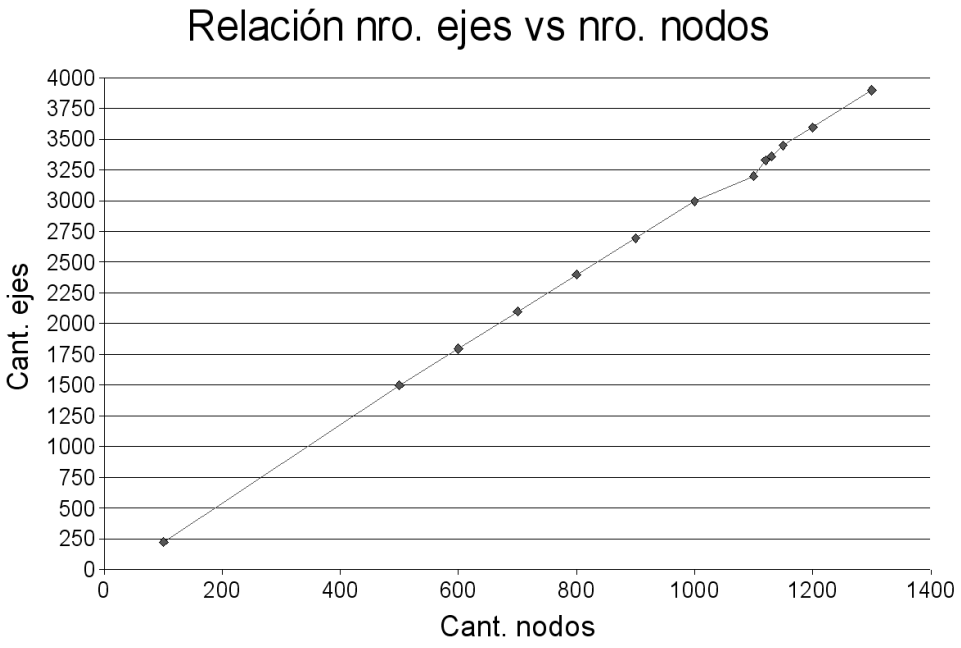
Cant. de nodos	1300
Cant. de ejes	3897
Núm. de condición del sistema	9849,72

	Tamaño de la matriz de ecuaciones	Tiempo de ejecución
Método original	no disponible	no disponible
Método optimizado	3897 filas por 3897 columnas	594,53 seg

3.16. Resumen

#Nodos	#Ejes	$k(A)$	Algoritmo original		Algoritmo optimizado	
			Tamaño	Tiempo(seg)	Tamaño	Tiempo(seg)
101	224	161,20	323×224	4,23	224×224	0,16
500	1497	318,94	1995×1497	24,61	1497×1497	16,4
600	1796	4618,45	2394×1796	43,27	1796×1796	26,99
700	2094	3568,51	2792×2094	68,01	2094×2094	41,17
800	2397	3361,44	3195×2397	122,09	2397×2397	65,92
900	2696	1930,76	3594×2696	260,61	2696×2696	114,92
1000	2996	5744,37	3994×2996	389,09	2996×2996	181,44
1100	3200	4473,66	4298×3200	563,15	3200×3200	387,88
1120	3328	5045,62	4446×3328	573,97	3328×3328	407,11
1130	3359	4828,01	4487×3359	632,52	3359×3359	300,14
1150	3447	3805,32	4595×3447	701,68	3447×3447	350,78
1200	3596	4607,99	no disp	no disp	3596×3596	413,28
1300	3897	4607,99	no disp	no disp	3897×3897	594,53

Gráfico de los tiempos de ejecución en base al tamaño del grafo en nodos



3.17. Comentarios sobre la pruebas

Todas las pruebas se realizaron en un equipo del laboratorio de Linux del Departamento de Computación. Las configuraciones más relevantes del equipo son:

- Pentium 4 2.4Ghz
- 512Mb de memoria RAM
- Debian 4.0

Se midió el uso de memoria durante la ejecución del programa; sabemos que el programa no utilizó memoria *swap* para correr ya que para la instancia mayor no superó los 350Mb de memoria física usada. Lo comprobamos corriendo antes y durante la ejecución el comando free para observar cambios en el uso de memoria virtual y descartamos las corridas que provocaron paginado de otras aplicaciones hacia el swap.

El cálculo del número de condición se realizó en el mismo equipo, mediante Matlab versión 7.2.0.294 (R2006a).

Los grafos de 500 o más nodos se generaron aleatoriamente con Visone, usando la función Network Generation de tipo "Small World" con probabilidad de ejes entre nodos de 0.05 y radio 3. Posteriormente se definió manualmente cuáles nodos serían Nodos Concentradores cuidando que el backbone resultante sea lineal.

Los grafos usados para hacer pruebas se crearon por distintos medios.

Los tres primeros, incluido el grafo de 101 nodos, fueron creados manualmente. En cambio, el resto de los grafos fue creado automáticamente mediante Visone, usando la función Network Generation de tipo "Small World" con probabilidad de ejes entre nodos de 0.05 y radio 3. En este caso el resultado fue un grafo donde cada nodo tiene asignado un *id* numérico, y en el cual debe "dibujarse" el backbone. Este proceso consiste en elegir un conjunto de nodos que sea lineal, y asignar el texto NC en el *id*.

A nuestra implementación se le incorporó una función que toma un grafo creado con Visone y exportado en formato UCINET DL, que utiliza la información en el *id* de cada nodo si es "NC" o "CG" acompañado de un balance, e ignora los casos donde el *id* esté vacío o sea un valor numérico. Ignorar en este caso equivale a asumir que ese nodo es una Central Generadora con balance 0.

3.18. Otros Gráficos

Gráfico de los tiempos de ejecución en base al tamaño del grafo en cantidad de nodos.

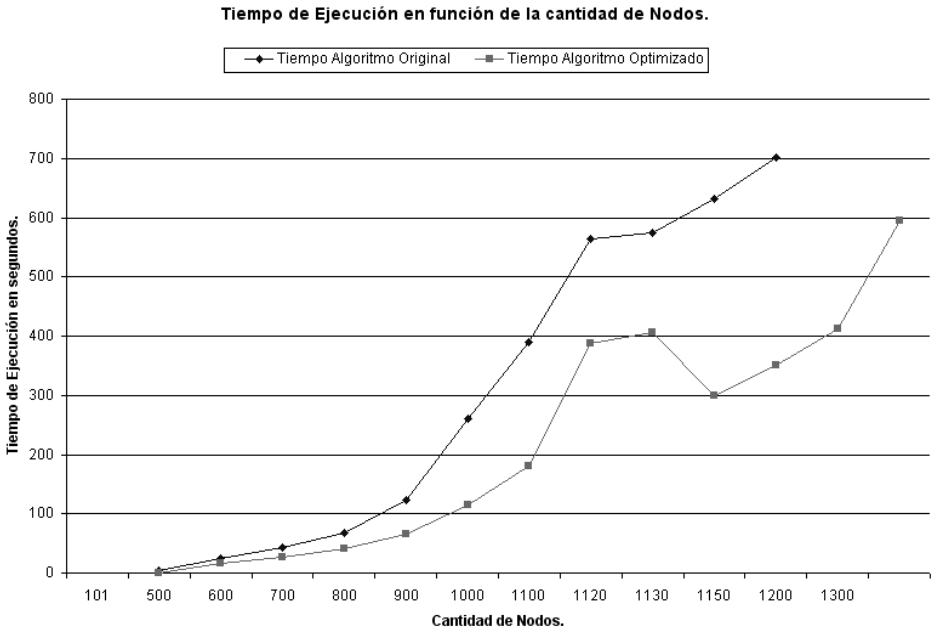


Gráfico de los tiempos de ejecución en base al tamaño del grafo en cantidad de ejes.

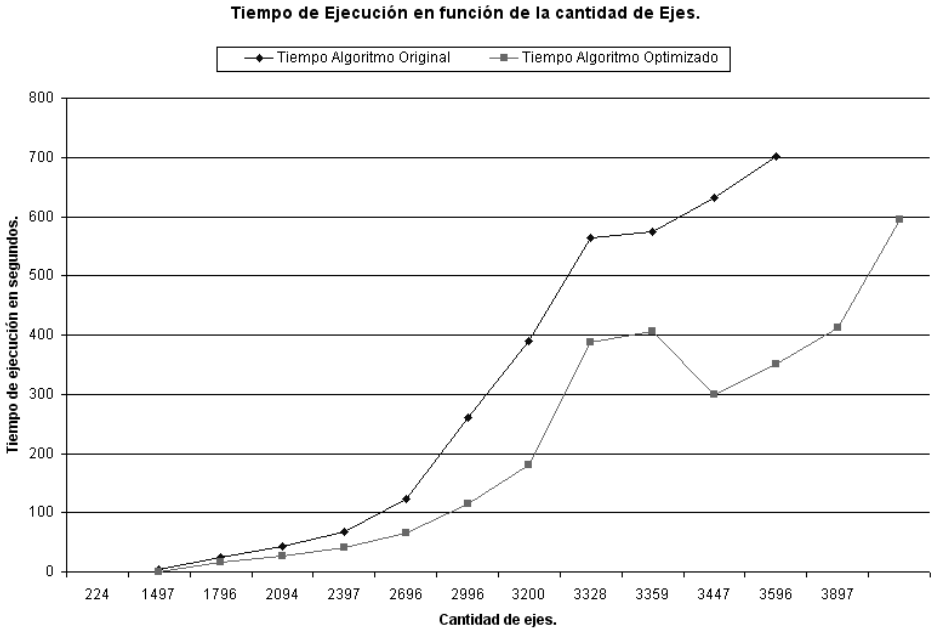
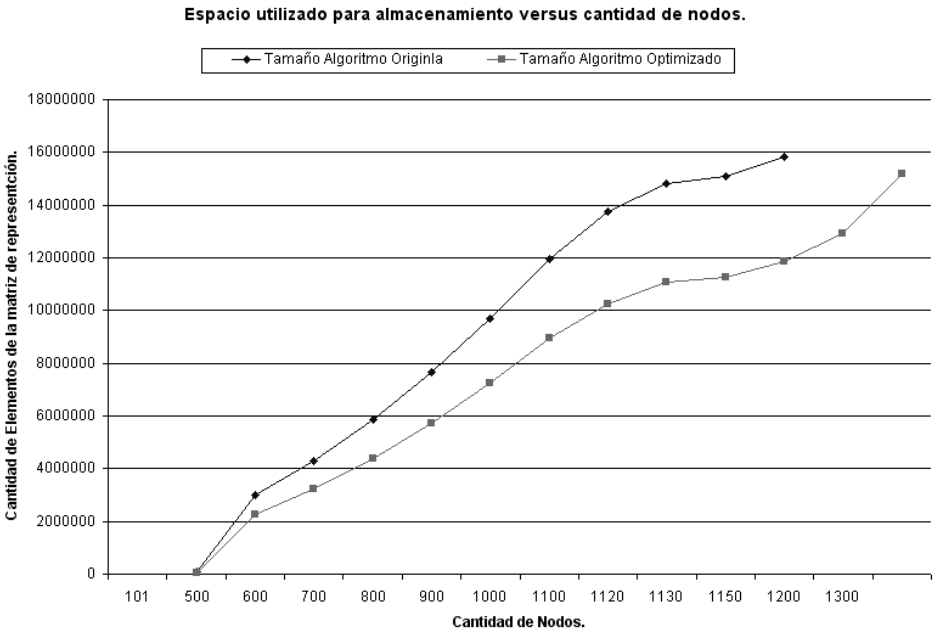


Gráfico del tamaño de la matriz en función de la cantidad de nodos.



4. Discusión

Las pruebas con el "algoritmo original" se hicieron sobre la forma original del programa, que planteaba el sistema lineal de ecuaciones a resolver para un grafo de n nodos y m ejes con una matriz de m columnas y $n+m-1$ filas. El "algoritmo optimizado" también asigna m columnas, pero asigna $max(m, n+p)$ filas, donde p es la cantidad de ejes en el backbone (como el backbone es lineal, se tienen $p+1$ nodos concentradores). Salvo casos particulares de grafos poco densos (Con centrales generadoras dispuestas en forma de estrella alrededor de las centrales concentradoras. Asi como sin conexiones como el caso de árboles, ya que el backbone es lineal, o con muy pocas conexiones entre centrales generadoras. Grafos en los cuales la mayoria de los ejes son ejes pertenecientes al backbone.) donde la cantidad de ejes en el backbone más la cantidad de nodos supere a la cantidad total de ejes, se tiene que $n+p \leq m$ y por lo tanto la matriz del sistema de ecuaciones es cuadrada.

Puede observarse claramente que el tiempo de ejecución del cálculo del flujo en la red es polinomial con grado ≥ 2, lo cual es de esperar ya que conocemos que la eliminación gaussiana es $O(n^3)$ sobre un sistema de n incógnitas y n ecuaciones.

Teniendo el tiempo de ejecución directamente relacionado con el tamaño de la entrada, y siendo las entradas muy similares, nos sorprendió que el número de condición presentara "oscilaciones", aumentando y disminuyendo a medida que se utilizaban instancias más grandes. Sin embargo este número depende de la complejidad del grafo, y el tamaño y forma del backbone puede tener influencia en este valor.

Se hicieron algunas pruebas que no aparecen en este informe, como modificar estos grafos para asignar balances aleatoriamente a las Centrales Generadoras con balance cero, manteniendo la condición que la suma de los balances se mantenga en 0. En estas pruebas no se observó ningún cambio en el número de condición, lo cual era de esperarse ya que sólo cambia la matriz b de valores y no la matriz A de ecuaciones, y hubo un impacto despreciable en el tiempo de ejecución del algoritmo, que podría deberse simplemente a la carga adicional de tener que escribir *strings* más largos al *log* del programa. Ya que no resultaron valiosos estos experimentos no se incluyeron los resultados en este informe.

Sin embargo probar el programa con balances asignados aleatoriamente sacó a la luz un problema subyacente de nuestra implementación de la eliminación gaussiana: en numerosos lugares se comparaba si un valor era exactamente cero (por ejemplo al buscar un elemento pivote para una columna), por ejemplo $v == 0.0$. Pese a que utilizamos **long double** tanto en cálculos intermedios como en el almacenamiento de valores en la matriz, siempre sucede algún error numérico al despejar y al comparar exactamente por cero se estaba agravando. El síntoma del problema se presentaba como una para-

da del programa al encontrar que el sistema de ecuaciones era incompatible, ya que se obtenía una fila en ceros en la matriz A pero un valor muy cercano pero distinto de cero en la matriz de valores b.

Debido a este problema se cambiaron todas las comparaciones exactas por una comparación de cercanía a cero. También se modificáron las sentencias relacionadas con despeje, comparando el valor despejado con cero: valores muy cercanos a cero ahora se cambian por el valor exacto 0.0.

5. Conclusiones

El valor del cual depende muy fuertemente el tiempo de ejecución del algoritmo que resulve el problema planteado para una red elćtrica con las restricciones de Morley es la cantidad de ejes del grafo que representada dicha red. En un principio esto nos tendría que haber resultado bastante razonable, sin tener en cuenta ninguno de los resultados obtenidos, ya que los valores del flujo de cada uno de los ejes eran inicialmente nuestras incógnitas. De esta manera la cantidad de ejes representa una buena medida de la complejidad del problema planteado.
Las instancias de prueba utilizadas son instancias poco densas. En general la relación entre la cantidad de nodos n y ejes m del grafo que representa a la red se corresponde con m aprox $3n$.
Hubiese sido muy interesante poder probar instancias con diferente cantidd de ejes para una misma cantidad de nodos. De esta manera hubiéramos podido notar más claramente la influencia de la variación en la densidad de un grafo con respecto al tiempo de ejecución de cada una de las instancias. Llamamos densidad de un grafo (representación matemática de la red eléctrica) a la cantidad de ejes que contiene dicho grafo, dividido la cantidad máxima posible de ese mismo grafo. Esto sería $m/(n * (n-1)/2)$.
La reducción en el espacio de almacenamiento debido a la nueva representacón del grafo mediante una matriz para poder resolver el problema fue del 25 porciento. Esta reducción en el espacio de almacenamiento trajo aparejada una disminución en el tiempo de ejecución y la posible inclusión de instancias de mayor tamaño para las pruebas realizadas.
El tiempo de ejecución, medido en segundos, en función de la cantidad de nodos o ejes, ya que ambas distribuciones pertenecen a una misma familia, se corresponden con una distribución exponencial.
No existe relación aparente a simple vista entre el tiempo de ejecución y el número de condición de la matriz. Como nosotros sabemos el número de condición nos otorga una medida de confiabilidad sobre la solución encontrada al sistema de ecuaciones planteado para obtener la solucón del problema de flujos sobre las interconexiones de una red eléctrica.

6. Apéndice A

6.1. Enunciado

Laboratorio de Métodos Numéricos - Primer cuatrimestre 2007
Trabajo Práctico Número 2: Distribución eléctrica

Una *red de Morley* de transporte de electricidad es un sistema interconectado de *centrales generadoras* y *nodos concentradores*, unidos por líneas de alta tensión. Por ejemplo, la Figura 1 muestra una red de Morley compuesta por 15 centrales generadoras (círculos blancos) y 4 nodos concentradores (círculos negros). Las líneas que conectan los nodos concentradores entre sí se llaman el *backbone* de la red. Los nodos concentradores no realizan tareas de generación de energía, sino que se limitan a canalizar el flujo eléctrico que proviene de las centrales generadoras. Este modelo debe su nombre a la propuesta realizada por O. Morley en 1907.

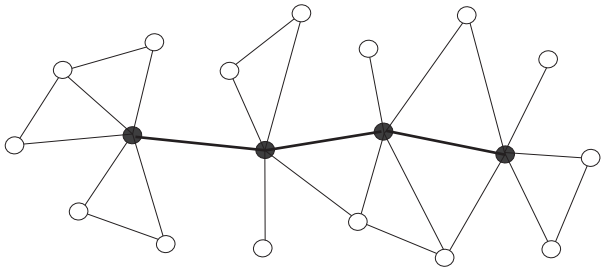


Figura 1: Ejemplo de una red de Morley.

Los usuarios de la red eléctrica se encuentran conectados a la red generadora más cercana, que los abastece de energía. Decimos que una central generadora tiene *balance positivo* si la energía que produce alcanza para abastecer a sus usuarios. La energía sobrante es entregada por la central a la red eléctrica, para abastecer a otras centrales. Por el contrario, decimos que una central generadora tiene *balance negativo* si la energía que genera no es suficiente para cubrir la demanda de sus usuarios. En este caso, la central debe recibir energía desde otras centrales, a través de la red. Notar que las centrales generadoras no sólo están conectadas a los nodos concentradores, sino que puede suceder que haya centrales generadoras con conexiones directas entre sí.

Sea $C = \{1, \ldots, n\}$ el conjunto de centrales generadoras, y llamemos $b_i \in \mathbb{R}$ al balance de la central $i \in C$ (notar que $b_i > 0$ si la central i tiene balance positivo, y $b_i < 0$ en caso contrario). El problema de distribución eléctrica sobre la red consiste en determinar qué flujo de energía eléctrica se debe enviar por cada línea de alta tensión, de manera tal que:

(i) Si $i \in C$ es una central con balance positivo, entonces el flujo neto saliendo de la central i (es decir, la suma de los flujos que salen menos la suma de los flujos que entran a la central) debe ser igual a b_i.

(ii) Si $i \in C$ es una central con balance negativo, entonces el flujo neto entrando a la central i (es decir, la suma de los flujos que entran menos la suma de los flujos que salen de la central) debe ser igual a $-b_i$.

(iii) El flujo total que entra a un nodo concentrador debe ser igual al flujo total que sale de ese nodo.

(iv) Todas las interconexiones del *backbone* de la red deben tener el mismo nivel de flujo.

Esta última condición se conoce con el nombre de *restricción de Morley*. Por último, suponemos que no hay límite al flujo que puede transportar cada línea de alta tensión.

Enunciado

El objetivo del trabajo práctico es implementar un programa que tome como entrada las interconexiones de la red y el balance de cada central generadora, y que determine si existe una distribución de energía eléctrica que cumpla las condiciones (i)-(iv). El programa implementado debe tomar los datos de entrada de un archivo de texto, cuyo formato queda a criterio del grupo.

Se pide plantear un sistema de ecuaciones lineales para este problema. El informe debe justificar la validez del sistema propuesto, explicando por qué dicho sistema resuelve efectivamente el problema de distribución. El informe debe contener también una descripción detallada del método que implementa el programa para la resolución de este sistema de ecuaciones.

Se pide realizar experimentos con redes de prueba para validar el funcionamiento de la implementación. Medir el número de condición de las matrices de los sistemas de ecuaciones resultantes, para tener una idea del error numérico cometido por la resolución. ¿Cuál es el tamaño de la mayor red eléctrica que el programa puede resolver en 10 minutos o menos?

Fecha de entrega: Lunes 30 de Abril

7. Apéndice B

7.1. Funciones Relevantes

```
/**
 * Triangular la matriz *this modificando también b
 * No hace el despeje. Se detiene si la matriz *this representa a un sistema incompatible
 * o indeterminado (devolviendo false).
 * Devuelve true si la matriz representa un sistema determinado con igual cantidad de
 * filas no completamente iguales a 0 que columnas.
 */
bool clase::triangular(clase& b)
{

assert(getFil() == b.getFil());
  bool rv = true;
// Se hará pivoteo parcial. Guardar un vector de subíndices para cada fila
unsigned int nrow[getFil()];
for (uint_t i = 0; i < getFil(); i++)
nrow[i] = i+1; // inicialmente la fila i-ésima es la verdadera fila i
// Triangular la matriz
// Poner en la fila i-ésima la que tiene el elemento en la columna
// i-ésima con valor absoluto más grande

for (uint_t i = 1; i <= getFil(); i++) {
    // i es la fila "lógica" que se está procesando
unsigned int f = nrow[i-1]; // la fila "física" usada para acceder a la matriz
if (i > getCol()) {
// las últimas filas deben estar en cero
if (!tieneFilaEnCero(f) || !b.tieneFilaEnCero(f)) {
rv = false; // indeterminado o incompatible
break;
}
}
else {
// Una de las primeras filas. Buscar el elemento pivote de mayor valor absoluto,
// haciendo intercambio de filas (un intercambio simulado, se intercambian los
// índices a filas en la tabla nrow[])
uint_t filaMejorPivote = i;
uint_t columna = i;
while (columna <= getCol()) {
filaMejorPivote = i;
f = nrow[filaMejorPivote-1];
long double mejorPivote = abs(sub(f, columna));
for (uint_t j = i+1; j <= getFil(); j++) {
// Ver si el elemento sub(nrow[j-1], i) es un mejor pivote
long double candidato = abs(sub(nrow[j-1], columna));
if (candidato > mejorPivote) {
mejorPivote = candidato;
filaMejorPivote = j;
}
}
if (mejorPivote < EPSILON) {
// Problema: tenemos la columna i-ésima en cero a partir de la fila i en adelante
// El sistema es indeterminado.
// Proseguir el algoritmo, tomando como elemento pivote el de la próxima columna
columna++;
}
else
break; // OK, seguir el algoritmo
}
```

```
if (columna > getCol()) {
rv = false; // indeterminado o incompatible
break;
}

// Intercambiar (simuladamente) las filas si corresponde
if (i != filaMejorPivote) {
uint_t tmp = nrow[filaMejorPivote-1];
nrow[filaMejorPivote-1] = nrow[i-1];
nrow[i-1] = tmp;
}

// Para cada fila a partir de i+1 restar un múltiplo conveniente de la fila i
for (uint_t j = i+1; j <= getFil(); j++) {
unsigned int fi = nrow[i-1];
unsigned int fj = nrow[j-1];
if (abs(sub(fj, columna)) >= EPSILON) {
//long double mi = sub(fj, columna) / sub(fi, columna); // multiplicador
long double mi = sub(fj, columna); // multiplicador
long double di = sub(fi, columna); // divisor
sub(fj, columna) = 0.0;
for (uint_t k = columna+1; k <= getCol(); k++) {
           long double thissubfjk = sub(fj, k) - mi * sub(fi, k) / di;
                // E(j) - mi*E(i) -> E(j)
                if (-EPSILON < thissubfjk && thissubfjk < EPSILON)
                 thissubfjk = 0.0;
sub(fj, k) = thissubfjk;
          }
for (uint_t k = 1; k <= b.getCol(); k++) {
              long double bsubfjk = b.sub(fj, k) - mi * b.sub(fi, k) / di;
              // b.E(j) - mi*b.E(i) -> b.E(j)
              if (-EPSILON < bsubfjk && bsubfjk < EPSILON)
               bsubfjk = 0.0;
b.sub(fj, k) = bsubfjk;
          }
}
else {
sub(fj, columna) = 0.0;
}
}
}
}

// Aplicar los intercambios de filas hechos
for (uint_t i = 1; i <= getFil(); i++) {
uint_t f = nrow[i-1]; // fila real en la matriz
// Colocar la fila real en la fila i-ésima, intercambiando si corresponde
if (i != f) {

// Intercambiar las filas i-ésima y f-ésima
for (uint_t k = 1; k <= getCol(); k++) {
long double tmp = sub(i, k);
sub(i, k) = sub(f, k);
sub(f, k) = tmp;
}

// También intercambiar en el valor b
for (uint_t k = 1; k <= b.getCol(); k++) {
long double tmp = b.sub(i, k);
b.sub(i, k) = b.sub(f, k);
b.sub(f, k) = tmp;
}
```

```
// Indicar en la tabla nrow que esas filas fueron intercambiadas
// Buscar cuál elemento en la tabla nrow apuntaba a la fila física i y cambiarlo a f
for (uint_t k = i + 1; k <= getFil(); k++)
if (nrow[k-1] == i) {
nrow[k-1] = f; //la que antes era la fila física i ahora está en la fila física f
break;
}
nrow[i-1] = i;
}
}
return rv;
}
```

```
/**
 * Despejar la matriz
 * Como precondición, la matriz debe corresponder a un sistema triangulado superiormente,
 * de M x N, con las primeras N filas distintas de cero y las restantes (si hay) en cero.
 */

void clase::despejar(clase& b) {

assert(b.getFil() == getFil()); // deben tener la misma cantidad de columnas
assert(clase::esDeterminado(b));

// Modificar la matriz *this y también los valores de la matriz resultado b
for (uint_t i = getFil(); i > 0; i--) {
if (i > getCol()) {
// La fila debe estar toda en cero, tanto para esta matriz como para la matriz b
assert(tieneFilaEnCero(i) && b.tieneFilaEnCero(i));
}
else {

// Los primeros i - 1 elementos deben estar en cero
for (uint_t j = 1; j < i; j++)
assert(sub(i, j) == 0.0); // o usar una tolerancia
assert(sub(i, i) != 0.0); // no puede ser cero!!!

// El valor debe ser 1.0
long double pivote = sub(i, i);
if (pivote != 1.0) {
sub(i, i) = 1.0;
// Multiplicar toda la fila i-ésima de b por 1/pivote
b.multiplicarFilaPorEscalar(i, 1.0/pivote);
}

// Ahora despejar con este valor las filas 1 .. i-1
for (uint_t k = 1; k < i; k++) {
pivote = sub(k, i);

if (abs(pivote) >= EPSILON) {
sub(k, i) = 0.0;

// Restar de la fila k de la matriz b, la fila i multiplicada por el pivote
b.sumarMultiploDeFila(k, -pivote, i, 1);
}
}
}
}
}

void hacerDeterminado(Matriz& A, Matriz& B, AnalisisMatriz& am) {
assert(am.esTriangularSuperiorYDespejable && am.esIndeterminado && A.getFil() >=
A.getCol());
// La matriz A corresponde a un sistema indeterminado.
// Por lo tanto tiene algunas filas en cero, correspondientes a variables
// que no puede despejar.
// Vamos a convertirlo a un sistema determinado, agregando ecuaciones de la forma
// eje(i) = 0
// Recorrer la matriz buscando variables sin un valor definido.
// Sabemos que la matriz tiene am.cantFilasDistintasDeCero filas que no son todas cero.
// Agregar las nuevas ecuaciones en las filas siguientes
unsigned int proxFila = am.cantFilasDistintasDeCero + 1;
unsigned int i = 1;
for (unsigned int j = 1; j <= A.getCol(); j++) {
// Si el valor A(i,j) == 0, tenemos la variable j-ésima indeterminada
```

```
if (abs(A.sub(i, j)) == 0.0) {
// Agregar una ecuación de la pinta x(j-esimo) = 0
A.sub(proxFila, j) = +1.0;
B.multiplicarFilaPorEscalar(proxFila, 0.0); // innecesario. Si fuera distinto
// de cero sería incompatible
// Ubicar a la siguiente fila para las ecuaciones agregadas
proxFila++;
}
else {
// El x(j-esimo) puede despejarse. Saltar a la siguiente fila y variable
i++;
}
}
}
```

```
int calcular(InfoMatriz& im, string salidaMatriz) {
    std::ostream &log = logger::getLogger();
    // Resolver el sistema de ecuaciones
    // Triangular
    bool ok = im.matriz.triangular(im.balances);
    // Ver si el sistema es incompatible
    AnalisisMatriz am = im.matriz.analizarMatriz(im.balances);
    if (!am.esTriangularSuperiorYDespejable) {
     im.matriz.show(log);
     return 255;
    }
    if (am.esIncompatible) {
     im.matriz.show(log);
     im.balances.show(log);
     return 196;
    }
    // Ver si el sistema es indeterminado. En este caso admite múltiples soluciones.
    // Asignar valores para reducirlo a un sistema determinado, y luego despejar
    if (am.esIndeterminado) {
     hacerDeterminado(im.matriz, im.balances, am); // agregar filas a la matriz
     // para que pueda despejarse
     im.matriz.show(log);
  im.balances.show(log);
      im.matriz.triangular(im.balances); // hacer que quede triangular
     am = im.matriz.analizarMatriz(im.balances);
     if (!am.esDeterminado) {
     return 128;}
    }
    // Chequear que el sistema sea determinado
    assert(am.esDeterminado);
// Exportar la matriz para poder calcular su número de condición con Matlab
if(salidaMatriz != "") {
        ofstream salida_matlab(salidaMatriz.c_str());
        if (!salida_matlab.is_open() || !salida_matlab.good()) {
            return 240;}
  // sabemos que la matriz es determinada, entonces las últimas filas están
  // en cero. No es necesario exportarlas.
        im.matriz.show_csv(salida_matlab, 1, im.matriz.getCol(), 1, im.matriz.getCol());
        salida_matlab.close();
    }
im.matriz.despejar(im.balances);
im.matriz.show(log);
im.balances.show(log);
// Volcar el resultado al grafo, asignando dirección y flujo a cada eje
set<unsigned int> ejes = im.grafo.getEjes();
set<unsigned int>::iterator itEjes;
for (itEjes = ejes.begin(); itEjes != ejes.end(); itEjes++) {
// Obtener el eje
Grafo<InfoNodo, InfoEje>::Eje eje = im.grafo.getEje(*itEjes);
// Obtener su flujo
long double flujoEje = im.balances.sub(*itEjes, 1);
if (flujoEje < 0) {
// Invertir el sentido del eje
eje.invertirSentido();
flujoEje = -flujoEje;
}
// Asignar el valor del flujo
InfoEje infoEje;
infoEje.flujo = flujoEje;
eje.setInfoUsuario(infoEje);
}
  return 0;}
```

```
InfoMatriz leerDe(std::istream& in)
{ ...
...
...
 // Agregar las ecuaciones de los nodos en la matriz
set<unsigned int> ejes = rv.grafo.getEjes();
//set<unsigned int>::iterator it;
for (it = ejes.begin(); it != ejes.end(); it++) {
// Para cada nodo debe cumplirse la ecuación
//   SUMA(flujo saliente) - SUMA(flujo entrante) = balance
// Si para un nodo un eje es saliente, se agrega un +1 a la matriz;
// eso significa que el eje está sumandole al balance del nodo.
// En cambio, si el eje es entrante hay que agregarlo como -1 a la matriz
Grafo<InfoNodo, InfoEje>::Eje eje = rv.grafo.getEje(*it);
rv.matriz.sub(eje.getNodoOrigen(), *it) = +1.0; // es saliente del nodo origen
rv.matriz.sub(eje.getNodoDestino(), *it) = -1.0; // es entrante al nodo destino
}
   // Agregar las restricciones de Morley en la matriz
// Las restricciones de Morley dicen: el flujo en cada eje del backbone es igual.
// En nuestra implementación exigimos además que todos los ejes del backbone tengan
// el mismo sentido y el backbone sea lineal, para poder resolver el problema con un
// sistema de ecuaciones planteado en una matriz.
// Planteadas como ecuaciones, las restricciones son:
//     eje(i) == eje(j)
// y para llevar a una matriz:
//     eje(i) - eje(j) = 0
int fila = rv.grafo.getCantNodos(); // colocar a partir de esta fila las
// ecuaciones de la restricción de Morley
if (rv.listaEjesBackbone.size() >= 2) {
list<unsigned int>::iterator itLista = rv.listaEjesBackbone.begin();
unsigned int prevEjeId = *itLista; // como la lista tiene al menos un
// elemento, esto no se pincha
for (itLista++; itLista != rv.listaEjesBackbone.end(); itLista++) {
unsigned int proxEjeId = *itLista;
fila++;
rv.matriz.sub(fila, prevEjeId) = +1.0;
rv.matriz.sub(fila, proxEjeId) = -1.0;
rv.balances.sub(fila, 1) = 0.0; // innecesario (balances se inicializa a 0)
// preparar para la siguiente ecuación
prevEjeId = proxEjeId;
}
}
```

8. Referencias

- Amálisis Numérico de Richard L. Burden y J. Douglas Faires.
- $http://visone.info/$
- $http://en.wikipedia.org/wiki/Small-world_network$