DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 3

Métodos directos vs Iterativos: PageRank

24 de noviembre de 2022 Métodos Numéricos

12

Integrante	LU	Correo electrónico
Damburiarena, Gabriel	889/19	gabriel.damburiarena@gmail.com
Guastella, Mariano	888/19	marianoguastella@gmail.com
Silva, Ignacio Tomas	410/19	ignaciotomas.silva622@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción

Entre tantos métodos para resolver sistemas de ecuaciones lineales, nos vamos a detener en dos y los vamos a comparar entre si. Primero necesitamos un problema que se pueda modelar con sistemas de ecuaciones lineales para poder resolverlo. Bueno, tomaremos el problema del PageRank. ¿Qué es el PageRank? Pues como de su nombre se deduce, es un ranking de páginas, en este caso páginas web, cuyo fin es ordenar por importancia a las mismas para un usuario. Para calcular este ranking hay que resolver un sistema de ecuaciones lineales donde la cantidad de ecuaciones e incógnitas del sistema es igual al número de páginas consideradas. Entraremos en detalle más adelante en la sección 1.1.

Ahora que tenemos el problema, los métodos que vamos a utilizar para resolverlo se categorizan en dos clases: métodos iterativos y métodos directos, ambos para la resolución de sistemas de ecuaciones lineales, claro. Implementaremos los métodos de Jacobi, Gauss Seidel, ambos pertenecientes a la primera clase y Eliminación Gaussiana, que pertenece a la segunda clase. La diferencia entre las clases radica en que el primero itera y siempre, al finalizar una iteración, uno puede terminar el proceso y quedarse con una solución aproximada mientras que el segundo requiere que se termine de ejecutar para encontrar una solución. La desventaja de los métodos iterativos pasa por 2 lugares. Uno es que es difícil establecer un criterio certero para cuando convergen, ya que uno podría tomar un vector inicial que lo haga imposible. Y la otra gran desventaja es que no esta garantizada su convergencia para cualquier tipo de matrices. En nuestro caso, las matrices con las que vamos a trabajar son estrictamente diagonal dominantes, por lo que sabemos que ambos métodos convergen.

Importante notar que como vamos a tener matrices ralas, ya que la mayoría de las páginas no se van a apuntar entre ellas, basaremos fuertemente las implementaciones de las matrices ralas en la biblioteca Eigen de C++.

Pasemos a definir tanto el problema como los métodos para resolverlo y la implementación de los mismos.

1.1. Definiciones

PageRank: Como adelantamos previamente, vamos a crear un ranking para determinar el orden de las páginas según su importancia. Contaremos la cantidad y la calidad de los links de cada página que apunten a otra página determinada para asignar un puntaje. Se busca que el puntaje del ranking sea mayor en las páginas importantes.

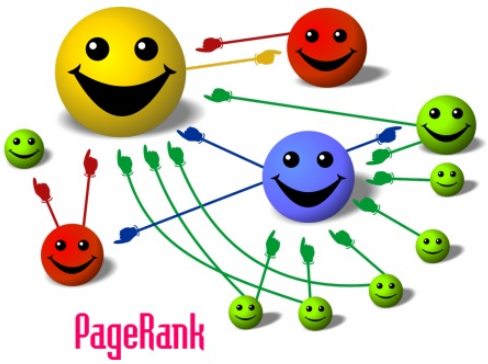

Figura 1: Esquema del PageRank

Representaremos las conexiones de n páginas web con la matriz de conectividad $\mathbf{W}$ definiendo $w_{ij} = 1$ si la página j tiene un link a la página i y $w_{ij} = 0$ en caso contrario. Además $w_{ii} = 0$ ya que no tiene sentido tener en cuenta los autolinks. Un dato no menor es que la matriz $\mathbf{W}$ puede resultar extremadamente rala y muy grande de acuerdo al tamaño del conjunto de páginas que hay en la web.

Para cada página $j = 1 \ldots n$, definimos su grado como:

$$c_{ij} = \sum_{i=1}^{n} w_{ij} \tag{1}$$

Es decir, el la cantidad de links salientes de j.

Como dijimos antes, vamos a asignarle calidad a los links, por lo tanto una página es importante cuando es apuntada por muchas otras páginas. Por ende, no todos los links pesan igual: los links de páginas más importantes valen más. Pocos links de páginas importantes pueden valer más que muchos links de páginas poco importantes. Y los links de páginas con muchos links valen poco. Por lo tanto, una forma de calcular la importancia o puntaje x_i de la página i es:

$$x_i = \sum_{j=1}^{n} \frac{x_j}{c_j} w_{ij} \tag{2}$$

También, se puede definir la matriz de puntajes $\mathbf{R} = \mathbf{WD}$, donde $\mathbf{D}$ es una matriz diagonal con elementos d_{jj} de la forma:

$$d_{jj} = \begin{cases} \frac{1}{c_j} & c_j \neq 0 \\ 0 & c_j = 0. \end{cases}$$

Lo cual nos permite calcular el ranking de todas las páginas como:

$$\mathbf{R}\ \mathbf{x} = \mathbf{x} \tag{3}$$

donde $\mathbf{x} = (x_1, ..., x_i, ..., x_n)^T$. Luego, la ecuación 2 corresponde al elemento i-ésimo $(\mathbf{R}\ \mathbf{x})_i$.

La falla de esta representación es que la navegación de los usuarios es errática y no secuencial en cuanto a los links, por lo cual vamos a tener que darle un giro de tuerca para poder capturar ese comportamiento.

Vamos a considerar un modelo donde el usuario comienza en una página aleatoria del conjunto, y luego en cada página j que visita elige con probabilidad $p \in (0, 1)$ si va a seguir uno de sus links, o con probabilidad 1 - p, si va a pasar a otra página cualquiera del conjunto. Si decidió seguir un link de la página j elige uno al azar con probabilidad $1/\ c_j$, mientras que si decidió pasar a otra página cualquiera entonces decide aleatoriamente con probabilidad 1/n a que página avanzar. Cuando la página j no tiene links salientes, es decir $c_j = 0$, avanza aleatoriamente a cualquier otra página.

Luego, la probabilidad de pasar de la página j a la página i es:

$$a_{ij} = \begin{cases} (1-p)/n + (pw_{ij})/c_j & c_j \neq 0 \\ 1/n & c_j = 0 \end{cases}$$

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$ a la matriz de elementos a_{ij} . Entonces, como adelantamos previamente, el ranking de Page es la solución del sistema: $\mathbf{A}\ \mathbf{x} = \mathbf{x}$ porque cumple $x_i \geq 0$ y $\sum_i x_i = 1$. Por lo tanto, el elemento i-ésimo $(\mathbf{Ax})_i$ es la probabilidad de que el navegante aleatorio esté en la página i sabiendo que x_j es la probabilidad de esté en la página j, para $j = 1 \ldots n$;

Desarmándola en partes, la matriz $\mathbf{A}$ se puede reescribir como:

$$\mathbf{A} = p\mathbf{W}\ \mathbf{D} + \mathbf{ez}^T \tag{4}$$

donde $\mathbf{D}$ es la matriz diagonal mencionada previamente, $\mathbf{e}$ es un vector columna de dimensión n y z es un vector columna cuyos componentes son:

$$z_j = \begin{cases} (1-p)/n & c_j \neq 0 \\ 1/n & c_j = 0 \end{cases}$$

Método de Jacobi:

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, buscamos $x \in \mathbb{R}^n$ tal que

$$\mathbf{A}x = b \tag{5}$$

Suponemos que $a_{ii} \neq 0 \ \forall i = 1, ..., n$

Sea $x^k \in \mathbb{R}^n$. Definimos un próximo punto x^{k+1} de la siguiente manera:

$$x_1^{k+1} = (b_1 - \sum_{\substack{j=1 \\ j \neq 1}}^{n} a_{1j} x_j^k)/a_{11}$$

$$\vdots$$

$$x_i^{k+1} = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^k)/a_{ii}$$

$$\vdots$$

$$x_n^{k+1} = (b_n - \sum_{\substack{j=1 \\ j \neq n}}^{n} a_{nj} x_j^k)/a_{nn}$$

Visto como matrices:

$$\mathbf{A}x = b$$

$$(\mathbf{D} - \mathbf{L} - \mathbf{U})x = b$$

$$\mathbf{D}x = (\mathbf{L} + \mathbf{U})x + b$$

$$x = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})x + \mathbf{D}^{-1}b$$

$$x^{k+1} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})x^k + \mathbf{D}^{-1}b$$

El algoritmo que implementaremos para la resolución del problema y basado en el apartado teórico recién enunciado es el siguiente:

Jacobi(matriz, niter, eps)

```
1:  b = vectorUnos(matriz.rows())
2:  D = matriz.diagonal()
3:  L = -matriz.estrictamenteTriangularInf()
4:  U = -matriz.estrictamenteTriangularSup()
5:  DInv = D.inverse()
6:  LPlusU = L + U
7:  x = vectorRandom(matriz.cols())
8:  for i = 0 hasta niter - 1 do
9:      xAnt = x
10:     x = DInv * (LPlusU * x) + (DInv * b)
11:     if (x - xAnt).norm() <eps then
12:         return x / x.sum()
13:     end if
14: end for
15: return x / x.sum()
```

Método de Gauss Seidel:

Sea $\mathbf{A} \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, buscamos $x \in \mathbb{R}^n$ tal que

$$\mathbf{A}x = b \tag{6}$$

Suponemos que $a_{ii} \neq 0 \ \forall i = 1, ..., n$

Sea $x^k \in \mathbb{R}^n$. Definimos un próximo punto x^{k+1} de la siguiente manera:

$$x_1^{k+1} = (b_1 - \sum_{\substack{j=1 \\ j \neq 1}}^{n} a_{1j} x_j^k)/a_{11}$$

$$\vdots$$

$$x_i^{k+1} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k)/a_{ii}$$

$$\vdots$$

$$x_n^{k+1} = (b_n - \sum_{j=1}^{n-1} a_{nj} x_j^{k+1})/a_{nn}$$

Visto como matrices:

$$\mathbf{A}x = b$$

$$(\mathbf{D} \text{ - } \mathbf{L} \text{ - } \mathbf{U})x = b$$

$$(\mathbf{D} \text{ - } \mathbf{L})x = \mathbf{U}x + b$$

$$x = (\mathbf{D} \text{ - } \mathbf{L})^{-1}\mathbf{U}x + (\mathbf{D} \text{ - } \mathbf{L})^{-1}b$$

$$x^{k+1} = (\mathbf{D} \text{ - } \mathbf{L})^{-1}\mathbf{U}x^k + (\mathbf{D} \text{ - } \mathbf{L})^{-1}b$$

Usaremos el siguiente algoritmo para la resolución del problema basado en el apartado teórico recién señalado:

GaussSeidel(matriz, niter, eps)

```
1: n = matriz.cols()
2: b = vectorUnos(matriz.rows())
3: x = vectorRandom(n)
4: for iter = 0 hasta niter - 1 do
5:     xAnt = x
6:     for i = 0 hasta n - 1 do
7:         sum = 0
8:         for j = 0 hasta n - 1 do
9:             if j = i then
10:                sum += matriz.coeff(i, j) * x(j)
11:            end if
12:        end for
13:        x(i) = (b(i) - sum) / matriz.coeff(i, i)
14:    end for
15:    if (x - xAnt).norm() <eps then
16:        return x / x.sum()
17:    end if
18: end for
19: return x / x.sum()
```

Eliminación Gaussiana:

Un sistema de ecuaciones se resuelve reduciendo el sistema dado a otro equivalente en el que cada ecuación tiene una incógnita menos que la anterior. El método de Gauss transforma la matriz de coeficientes en una matriz triangular superior a través de operaciones básicas(restar filas multiplicandas por un coeficiente). El método de Eliminación Gaussiana continúa el proceso de transformación hasta obtener una matriz diagonal.

Un algoritmo básico sería el siguiente:

Eliminación Gaussiana(matriz)

```
1:  for i = 0 hasta dimFila(matriz) - 1 do
2:      for j = i hasta dimFila(matriz) do
3:          value = matriz[j][i]
4:          if value ≠ 0 then
5:              factor = value / matriz[i][i]
6:              for k = j hasta dimColumna(matriz) do
7:                  matriz[j][k] = matriz[j][k] - (factor * matriz[i][k])
8:              end for
9:          end if
10:     end for
11: end for
```

El algoritmo que implementaremos para la solución del problema aprovecha métodos implementados de la biblioteca Eigen de C++ que ejecuta Eliminación Gaussiana de manera óptima a través de un ordenamiento especial de la matriz esparsa, su consecuente computo y factorización. Para más información sobre la implementación revisar la documentación en LUSolver

2. Experimentación

Como vamos a tener matrices ralas, ya que la mayoría de las páginas no se van a apuntar entre ellas, basaremos fuertemente las implementaciones de las matrices ralas en la biblioteca Eigen de C++. Implementaremos Eliminación Gaussiana (EG), Jacobi (JA) y Gauss Seidel (GS) utilizando también operaciones provistas por la misma biblioteca.Utilizaremos dos criterios de corte para los métodos iterativos y vamos a frenar el método cuando se cumpla el primero. Estos serán una diferencia menor a $1*10^{-6}$ entre la norma de los vectores resultado de dos operaciones consecutivas o 10000 iteraciones. Optamos por mantener estos criterios fijos ya que consideramos que analizar su variación no brindaría información relevante al objetivo del informe. Fijamos el primer criterio en $1*10^{-6}$ ya que significa que comenzaríamos a ver un error con respecto a la solución real a partir del sexto decimal, lo cual consideramos muy preciso. Luego las iteraciones fueron fijadas en 10000 ya que para los casos particulares que analizamos observamos que en el caso de llegar a la última iteración, se debía a que el método divergía, es decir, que se debía cortar la ejecución del algoritmo.

2.1. Instancias

Test 15 segundos y test 30 segundos:

Se trata de 2 instancias provistas por la cátedra, la primera con 2000 paginas y 12000 links y la segunda con 3000 paginas y 18000 links. Vamos a utilizarlas fundamentalmente porque fueron provistas con resultados y resulta útil para revisar el error de nuestros algoritmos.

Sumidero:

Se refiere a la instancia donde toda página tiene solamente un link a la primera pagina, y la primera pagina no tiene links. Presentaremos varias versiones de la misma, entre las cuales varía la cantidad de páginas presentes en el sistema.

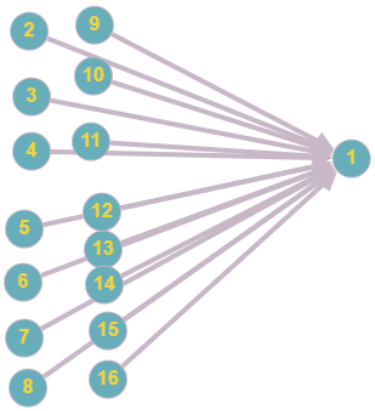

Figura 2: Ejemplo de red sumidero

Random:

Esta instancia contiene 99 páginas entre las cuales creamos links de una página a otra de forma aleatoria según una probabilidad que varia entre 0 y 1. Vamos a utilizar distintas versiones de esta misma instancia, entre las cuales va a variar la cantidad de links presentes en el sistema, de forma que se pueda ver un incremento en densidad entre versiones.

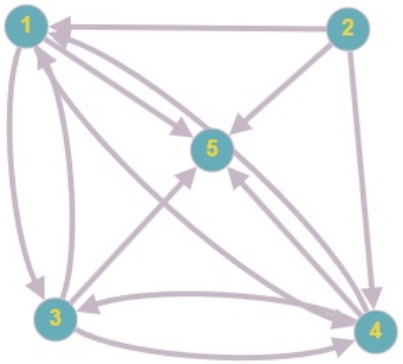

Figura 3: Ejemplo de red random

Densificación:

Nos interesa ver que pasa cuando cuando se le agregan aristas a uno de estos grafos, es decir, cuando su matriz se vuelve mas densa. Vamos a utilizar varias versiones de estos 2 tipos de instancias, aumentando su cantidad de aristas. Esto lo vamos a hacer con dos criterios distintos para luego comparar los resultados. Mientras la instancia random va a tener una cantidad de nodos fija y se le van a ir agregando aristas, a la instancia sumidero le vamos a agregar nodos y aristas para que tenga la misma cantidad entre estos 2. Por este motivo, aunque aumenten las aristas de la matriz sumidero, como también aumentan los nodos la matriz se va volviendo más rala. También agregamos una versión de la red sumidero donde la cantidad de nodos esta fija y se van agregando aristas, para ver su impacto.

2.2. Error de cálculo

Para analizar el error producido por nuestras implementaciones de los algoritmos y la implementación de Eigen, utilizaremos solamente las instancias 15-segundos y 30-segundos, ya que son las únicas instancias lo suficientemente grandes, proporcionadas por la cátedra, como para presentar un error de cálculo. Contrastaremos el resultado obtenido de nuestras implementaciones de cada método contra los resultados de la cátedra. Mediremos el error a través del error cuadrático medio (ECM).

Primero calcularemos el error para la instancia de 15-segundos.

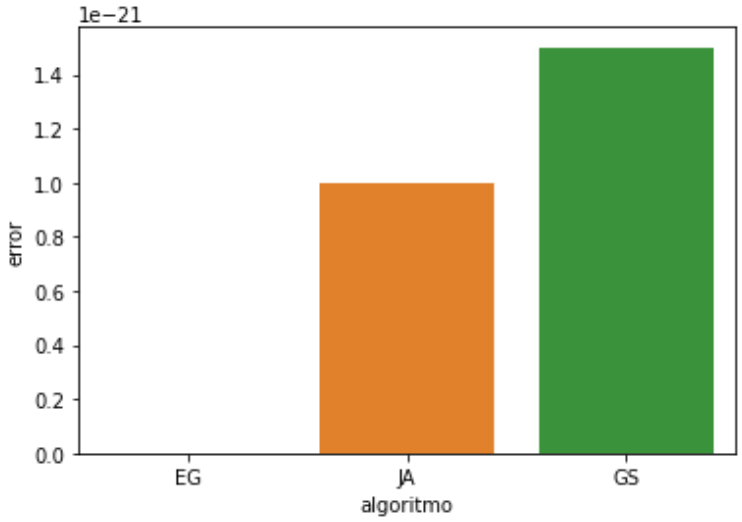

Figura 4: Error de cada método contra los resultados de la cátedra

Se puede observar que el error con el método de Eliminación Gaussiana es nulo. Con Jacobi aumenta y con Gauss Seidel es todavía mayor. Al ser la Eliminación Gaussiana un método exacto, el hecho de que el error sea despreciable parece indicar que la implementación es correcta. El motivo de la diferencia entra Jacobi y Gauss-Seidel no esta claro y es tan chico que deducimos que es arrastre de error de cuentas en la instancia particular. También barajamos la posibilidad de que para esta instancia Jacobi converge ligeramente mejor que Gauss Seidel.

En el caso de la instancia de 30 segundos, la diferencia de los resultados entre los algoritmos es despreciable en comparación a los test de la cátedra. El error es 4.358658e-10 para los tres métodos implementados por nosotros en comparación a la cátedra. Esto indica que los métodos resultan relativamente consistentes entre si, por lo menos tomando los resultados de la cátedra como referencia, ya que los tres tienen el mismo error.

2.3. Jacobi vs Gauss Seidel vs Eliminación Gaussiana

En esta sección apuntamos a analizar la performance de los métodos. Lo vamos a lograr comparando los tiempos de ejecución con cada una de las instancias propuestas anteriormente. Vamos a ejecutar cada experimento 10 veces, y tomaremos el promedio de tiempo entre esas ejecuciones.

En primer lugar queremos analizar el comportamiento de los métodos cuando se les presenta las instancias sumidero, las cuales se mantienen ralas pero incrementan en cantidad de nodos. Esperamos que, en general, a medida que aumentamos la cantidad de nodos el tiempo aumente. En particular, para la Eliminación Gaussiana tenemos la expectativa de que sea menos performante en comparación a los otros métodos, ya que este es un método exacto, mientras que los otros son aproximados y capaces de terminar la ejecución cuando obtienen una respuesta lo suficientemente satisfactoria.

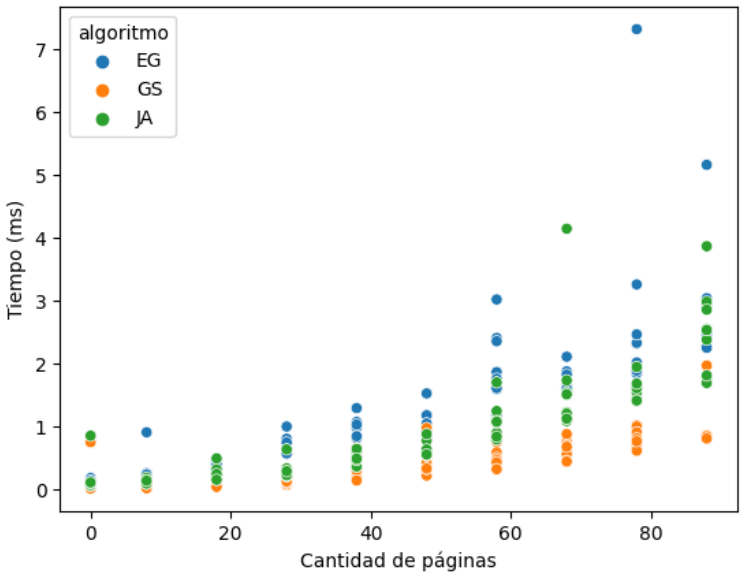

Figura 5: Instancia Sumidero corrida por los 3 métodos 10 veces aumentando la cantidad de páginas

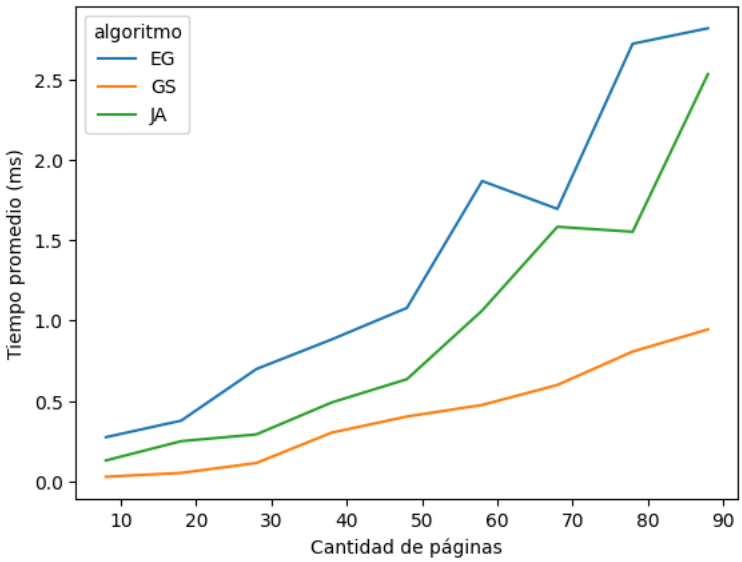

Figura 6: Instancia Sumidero promedio corrida por los 3 métodos 10 veces aumentando la cantidad de páginas
Desvío estándar promedio por algoritmo: EG 1.1118 | GS 0.8437 | JA 1.1118

Tal como predijimos, EG fue el menos performante, dejando de lado algunos outliers, que probablemente se debe a procesos internos de la computadora o a casos borde. Por otro lado, también podemos observar que a medida que aumenta la cantidad de páginas, aumenta el tiempo de ejecución, como habíamos hipotetizado. En particular, Gauss-Seidel triunfó la comparación, seguido por Jacobi, que, curiosamente, se asemejó a EG en pendiente.

Ahora vamos a ver que sucede cuando, con la misma red sumidero, dejamos la cantidad de nodos fija en el máximo y variamos solo la densidad.

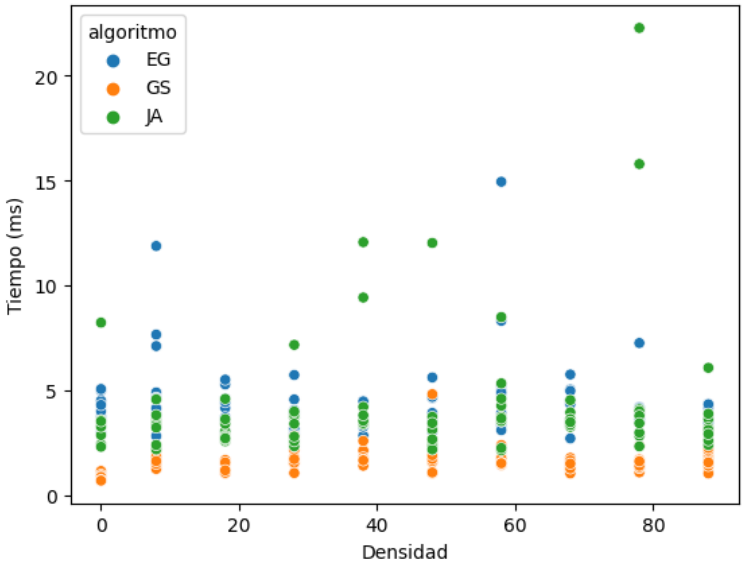

Figura 7: Instancia Sumidero n fijo

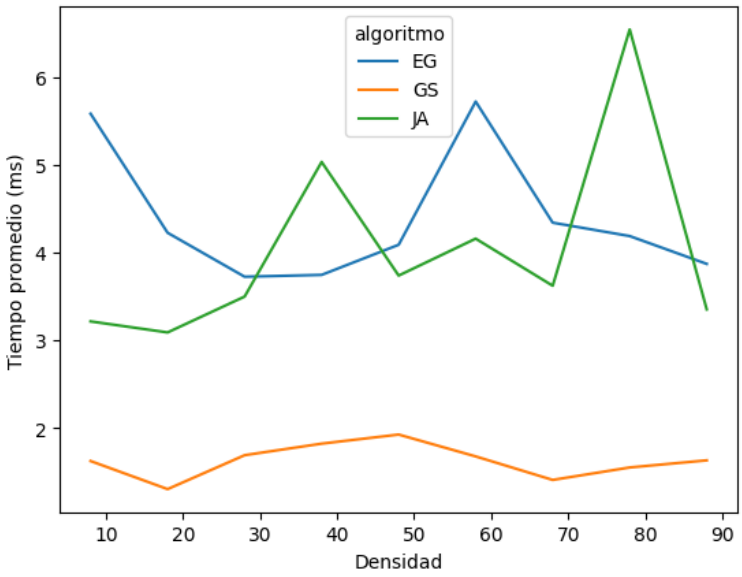

Figura 8: Instancia Sumidero n fijo promedio
Desvío estándar promedio por algoritmo: EG 1.6423 | GS 0.5310 | JA 2.7789

Lo que podemos ver es que hay poca variación de tiempo cuando aumentamos la densidad, y también se ve que se mantiene la superioridad de Gauss-Seidel en cuanto a tiempo de ejecución.

Al analizar utilizando las instancias 'random', vamos a querer ver el comportamiento de los métodos al encontrarse con sistemas cada vez más densos. Nuestra hipótesis, y también la razón por la cual planteamos instancias que crecen en cantidad de nodos, y no en densidad, es que el tiempo de ejecución de cada método no va divergir de un tiempo particular, sino que oscilará cerca de un valor.

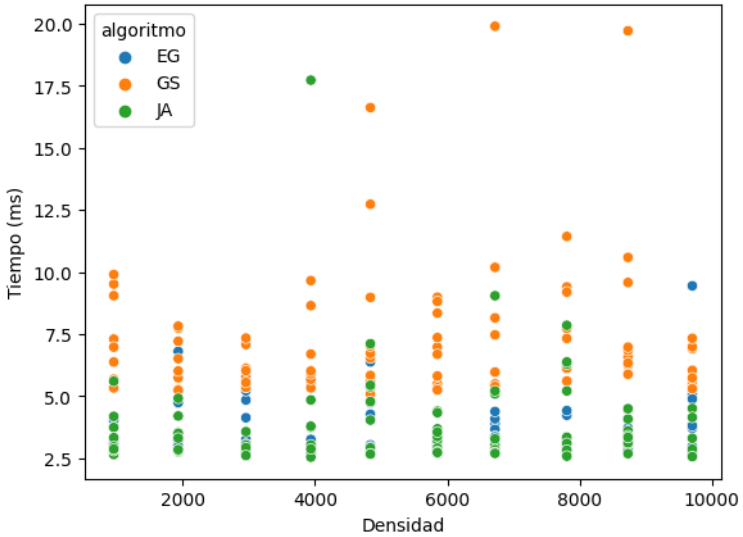

Figura 9: Instancia Random corrida por los 3 métodos 10 veces aumentando la densidad

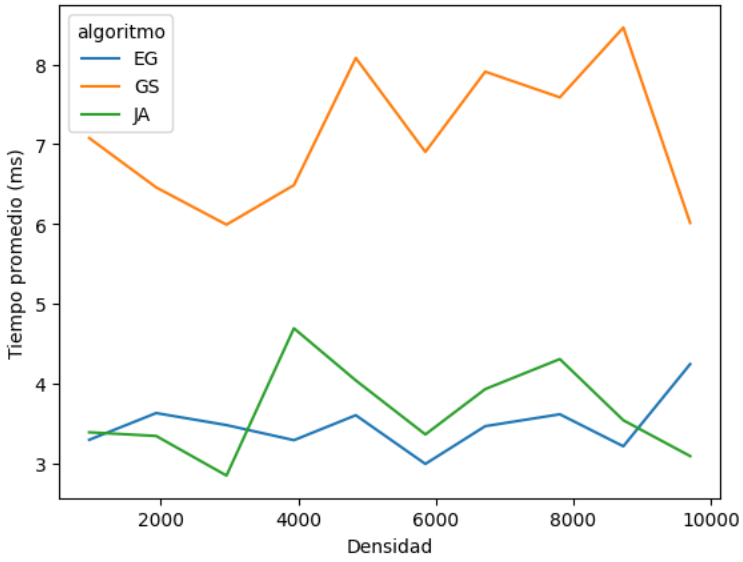

Figura 10: Instancia Random promedio corrida por los 3 métodos 10 veces aumentando la densidad
Desvío estándar promedio por algoritmo: EG 0.9941 | GS 1.8452 | JA 2.5770

Dejando de lado los outliers, que son aquellos puntos que distan mucho de los casos normales, probablemente por procesos internos de la computadora, se prueba nuestra hipótesis. Los tiempos de ejecución al variar la densidad no difieren en gran cantidad dentro del mismo método (10). Es interesante ver que el método de Gauss-Seidel fue el menos performante en esta comparación, a diferencia de las instancias sumidero, mientras que Eliminación Gaussiana y Jacobi batallan el primer lugar con promedios similares. Creemos que esto se debe a las implementaciones de matrices ralas que usamos, pertenecientes a la biblioteca 'Eigen'. Es posible que las mismas favorezcan a un método por encima de otro cuando los números no nulos presentes en las matrices se encuentran en una misma fila, como es el caso de las redes sumidero, o cuando se encuentran repartidos esporádicamente por toda la matriz. Otra observación es que gauss-seidel resulto mas rápido para instancias sumidero, quizás debido a su tamaño.

3. Conclusiones

En lo que respecta a la performance de los algoritmos, las conclusiones son relativamente claras. El impacto de agregarle nodos a una red es mucho mayor al de agregarle aristas. Esto tiene sentido ya que la matriz se volverá mas densa de todas formas al usarla para operar y que cada nuevo nodo que se agrega a una red sumidero de n nodos implica que existen $2 * n - 2$ nuevas aristas posibles. También extraemos de lo experimentado con el error con respecto al de la cátedra que, al aumentar el tamaño de la red, la diferencia entre los métodos disminuye. Aunque para este último resultado haría falta tener más instancias para contrastar, las dos que nos proporciona la cátedra quizás nos dan una conclusión sesgada.