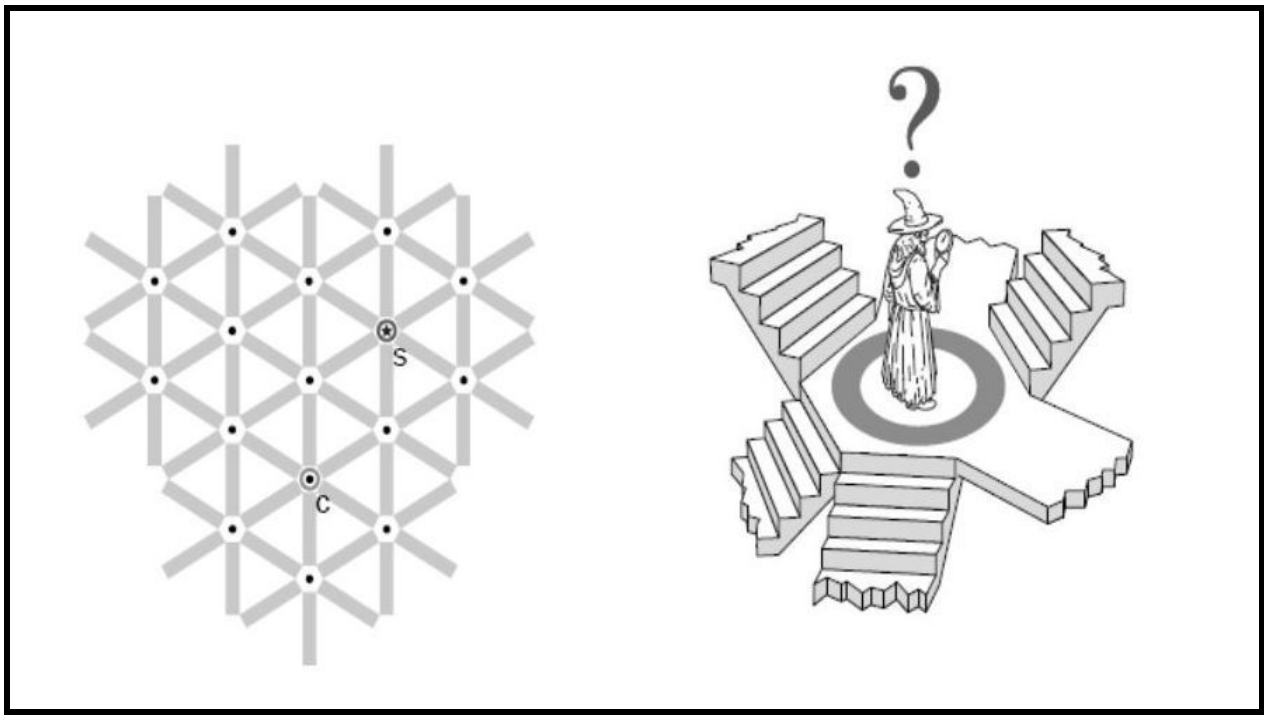
Algoritmos constructivos y búsquedas por entornos

Diseño y Análisis de Algoritmos



Carlos Díaz Calzadilla

26/04/2020

3º Ingeniería Informática

Itinerario Computación

ÍNDICE

1. INTRODUCCIÓN

En primer lugar interesa explicar el problema que se va a tratar. Partimos de un grafo completo G = (V, E), donde V es el conjunto de vértices (|V| = n) y E es el conjunto de aristas (|E| = n(n – 1)/2. Asimismo tenemos que cada arista (i,j) tiene asociada una distancia d(i,j), donde las distancias d(i,j) = d(j,i).

En nuestro problema, el *max-mean dispersion problem* se persigue encontrar el conjunto de vértices S ⊆ V que maximiza la dispersión media dada por:

$$md(S) = \frac{\sum_{i,j \in S} d(i,j)}{|S|}$$

2. ESPECIFICACIONES DEL ORDENADOR

El sistema en el que se han realizado las pruebas tiene las siguientes especificaciones:

- Sistema operativo: Windows 10 Home 64 bits (se ha usado subsistema Linux).
- Procesador: Intel(R) Core(™) i7-4510U CPU @ 2.00GHz (4CPUs), ~2.6GHz.
- Memoria: 16384 MB RAM.

3. DESCRIPCIÓN BREVE DE LOS ALGORITMOS

Cómo en el enunciado se explicaba, se han de implementar 5 algoritmos. En [1] hay un enlace a la documentación del código. Ahora vamos a hacer una explicación de cada uno de ellos:

- **Greedy (pseudocódigo)**: En este algoritmo se parte con una solución inicial que son los nodos que van a tener mayor valor en las aristas que los conectan. Una vez que se tiene lo siguiente se va a iterar hasta que una solución parcial sea igual a la inicial. Con esto luego, vamos a igual la solución parcial a la solución actual y procedemos a buscar el mejor k (denominamos k el valor del nodo que al añadirlo a la solución actual maximiza el resultado), entonces si la dispersión media con la solución actual + ese nodo k es mejor a la dispersión actual con la dispersión media, entonces le vamos incorporar el nodo k a la solución actual. Una vez acabado el bucle se devuelve la solución parcial, que contendrá el conjunto de nodos.

- **Segundo Greedy**: En este caso, se ha implementado una variante del anterior. Para ello, se ha partido de una solución inicial que está formada por el conjunto de nodos total, de tal forma que de forma se hará algo similar a lo anterior, lo único que cabe mencionar es que en vez de añadir nuevas soluciones y probar si ese conjunto maximiza la dispersión media. En este caso, se quitarán elementos del conjunto y se verá si ese nuevo conjunto es maximizado. En caso de que si pues se borrará del conjunto de soluciones ese nodo.

- **GRASP**: Para hacer este algoritmo se ha implementado, además de todo lo necesario, un pre procesamiento. Este básicamente lo que hace es crear un conjunto de candidatos que van a ser aquellos nodos que tienen al menos una arista positiva. Una vez que se hace este, hemos de tener en cuenta que las condiciones de parada son las iteraciones sin y con mejora. Aquí vamos a dividir el algoritmo en diferentes pasos:
    1) **Fase constructiva**: En esta, se ha decidido hacer un número aleatorio de iteraciones (siempre este número aleatorio es generado respecto al tamaño del conjunto de candidatos). En cada iteración se crea un RCL (para crear el RCL, lo que se ha planteado es coger el 60 % número de soluciones respecto a la mejor, es decir, si nuestra mejor solución es 10, pues nuestro RCL va a estar formado por aquellas que tengan un valor mayor a 6).
    2) **Fase de búsqueda local**: Para la búsqueda local, cómo en este caso y en los demás algoritmos se escogerá entre la búsqueda ansiosa (si encuentra una solución la devuelve, no sigue buscando) y la búsqueda greedy (busca la mejor solución y una vez que haya iterado todo, en el caso de que hayan nodos con valores iguales, devuelve uno aleatorio).

  Una vez que se hayan hecho las dos fases, se comprueba cual es la mejor solución (a partir de la dispersión media) y se actualiza en caso de que haya mejorado. Si ha mejorado se establece el iterador que controla las iteraciones sin mejora a 0, pero si no mejora, se incrementa en una unidad.

- **Multiarranque**: Para el multiarranque se ha realizado un pre procesamiento similar al que se implementó en GRASP. Aquí las condiciones de parada también son las iteraciones con y sin mejora. Ahora, dentro del bucle, se procederá a construir la solución que en este caso va a ser pseudoaleatoria. Para ello generamos un número que va a ser la cantidad de veces que vamos a iterar y luego generamos una posición aleatoria del conjunto de candidatos, Comprobamos si ese elemento ya se ha insertado en la solución y, en caso de que

no lo añadimos. Una vez que se haga la construcción se realiza una de las dos búsquedas (ansiosa o greedy). Para acabar, se comprueba si la dispersion media de la nueva solución es mejor que la mejor, en caso de que si la mejor pasa a ser la nueva solución (recordar que si se modifica, las iteraciones con mejora se ponen a 0) y si no pues se incrementa las iteraciones con mejora.

- **VNS**: En el caso de VNS, he decidido implementar la versión básica. Para ello, se crea un vector de candidatos que va a estar formado por todos los nodos (partimos de la idea que tenemos una variable kmax). Ahora volvemos a generar un valor aleatorio entre 0 y el número de nodos. Luego construimos la solución inicial. Tendremos un bucle hasta que un vector auxiliar alcance el valor del número aleatorio que se acaba de generar. Para ello generamos un número que va a ser la cantidad de veces que vamos a iterar y luego generamos una posición aleatoria del conjunto de candidatos, Comprobamos si ese elemento ya se ha insertado en la solución y, en caso de que no lo añadimos. Una vez que se haga la construcción se realiza una de las dos búsquedas (ansiosa o greedy). Y se determina la dispersión media de la solución inicial.

  Ahora se repite el siguiente proceso hasta que se obtenga el número de iteraciones con mejora o sin mejora del número definido. Continuando, tenemos que se repetirá hasta que el valor del tamaño de entorno llegue al máximo definido. En cada iteración se hará el proceso de agitación que básicamente consiste en buscar un elemento aleatorio dentro del entorno (construyendo luego la solución obtenida con ese elemento) y luego la búsqueda local con lo que se haya obtenido. Si la dispersión media con la nueva solución mejora, se actualiza y el tamaño del entorno vuelve al inicial (a su vez las iteraciones sin mejora vuelven a 0), en caso de que no, se incrementa el tamaño del entorno y las iteraciones sin mejora.

4. ESTRUCTURAS DE DATOS USADAS

En este apartado no quiero insistir mucho, pero hay aspectos que son necesarios de comentar. Principalmente se ha desarrollado una clase solución que va a contener el vector, tiempo de ejecución,... Además se ha implementado una clase grafo que va a contener una matriz de distancias. Asimismo, pensé que sería de utilidad implementar una clase File que se va a encargar de gestionar el fichero.Luego, otro aspecto a destacar es que, al haber aplicado el patrón *Strategy*, se ha realizado una clase Exec que va a contener un algoritmo y un fichero. Luego decir que cada algoritmo va a heredar de una clase general algoritmo con un método run que va a ser comun a cada algoritmo.

5. JERARQUÍA DE CLASES

Cómo ya se ha mencionado, se ha aplicado el patrón *Strategy* en esta práctica dado que se iban a implementar distintos algoritmos relacionados y este patrón nos iba a facilitar todo el trabajo. Todos los códigos han sido documentados con doxygen que es una herramienta que a partir de una determinada sintaxis, podemos obtener un html. Además nos ha proporcionado la jerarquía de clases tal y cómo se puede ver en la *figura 1*, además se puede visualizar en [3]:

figura 1: Representación jerarquía de clases (Clase Algorithm)

6. TABLAS

Para la realización de las pruebas, se han cogido 10 iteraciones de cada algoritmo sobre 4 grafos, cada uno con diferente cantidad de nodos (20, 50, 70, 100). Las tablas que se van a mostrar a continuación son los resultados obtenidos de la realización de las medias con estos datos. Aún así, se adjuntará las tablas en formato excel de forma local y en [2] se pondrá una referencia a las mismas en Google Drive. Una vez hecho esto se han almacenado los datos en una tabla y luego, para el muestreo de los datos se han realizado las medias de los datos obtenidos y se han pintado las gráficas.

Otros aspectos importantes a destacar es que, las tablas que se refieren a los algoritmos *GRASP*, *MULTIARRANQUE* y *VNS*, se han realizado con:

- Número de iteraciones total: 100
- Número de iteraciones sin mejora: 10
- Búsqueda local: Greedy

En el caso de *VNS*, comentar que se han hecho dos tablas, para ver cómo influye el tamaño del entorno en el problema. Una se ha hecho con un k igual a 3, y otra con un k mayor, igual a 7.

6.1. ALGORITMO VORAZ (PSEUDOCÓDIGO)

	ALGORITMO VORAZ (PSEUDOCÓDIGO)	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	11,36	0,444
50	20,74	7,131
70	24,22002	19,789
100	35,78041	65,794

6.2. SEGUNDO ALGORITMO VORAZ

	SEGUNDO ALGORITMO VORAZ	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	12,1429	1,700
50	20,89	28,999
70	25,2414	115,4853
100	39,9697	381,465

6.3. ALGORITMO GRASP

	ALGORITMO GRASP	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	12,5357	14,979
50	20,7784	237,734
70	25,93152	921,891
100	38,81555	2.515,666

6.4. ALGORITMO MULTIARRANQUE

	ALGORITMO MULTIARRANQUE	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	13,1667	8,737
50	20,6866	76,455
70	26,71535	285,022
100	39,68624	658,741

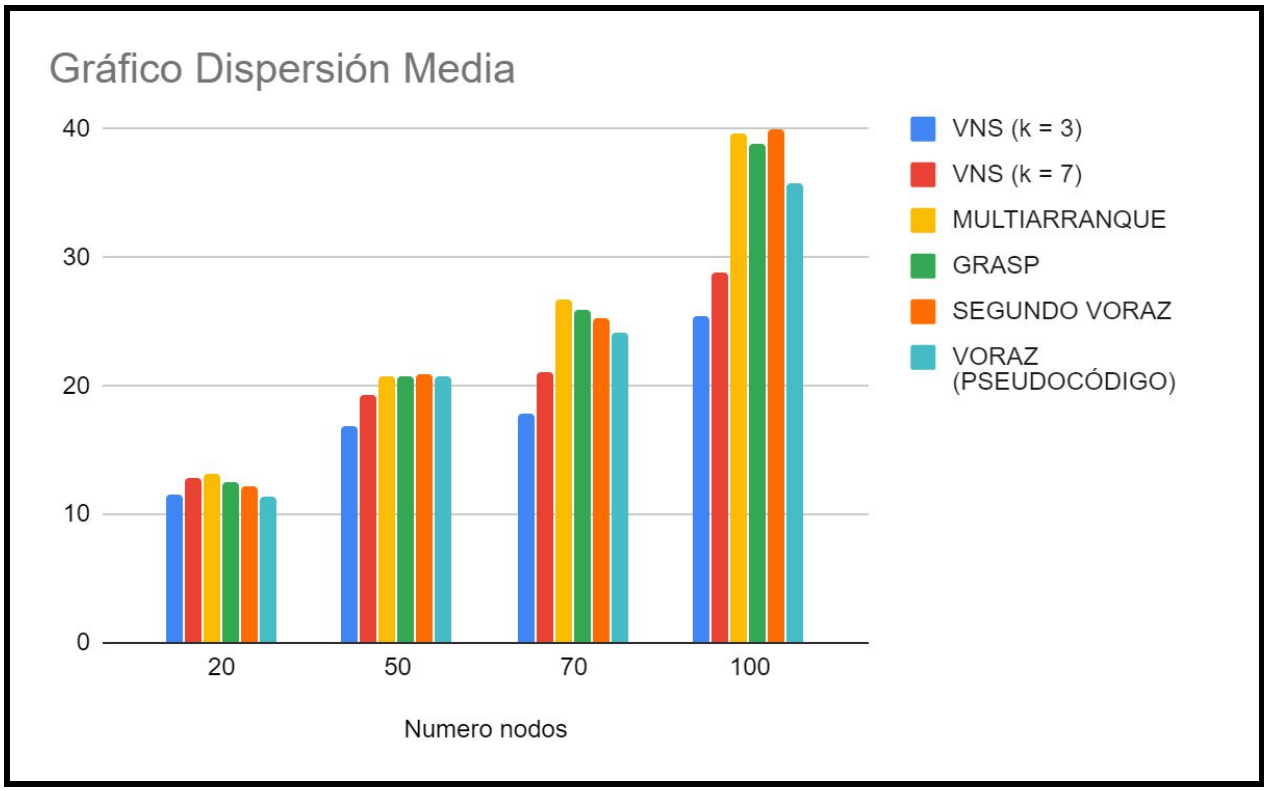
6.5. ALGORITMO VNS

	ALGORITMO VNS (k = 3)	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	11,5000	2,403
50	16,8383	11,188
70	17,77929	21,194
100	25,43849	73,028

	ALGORITMO VNS (k = 7)	
Numero nodos	Media Md	Media Tiempo CPU (ms)
20	12,8000	2,779
50	19,2425	19,203
70	21,09579	45,339
100	28,79981	93,559

7. GRÁFICAS

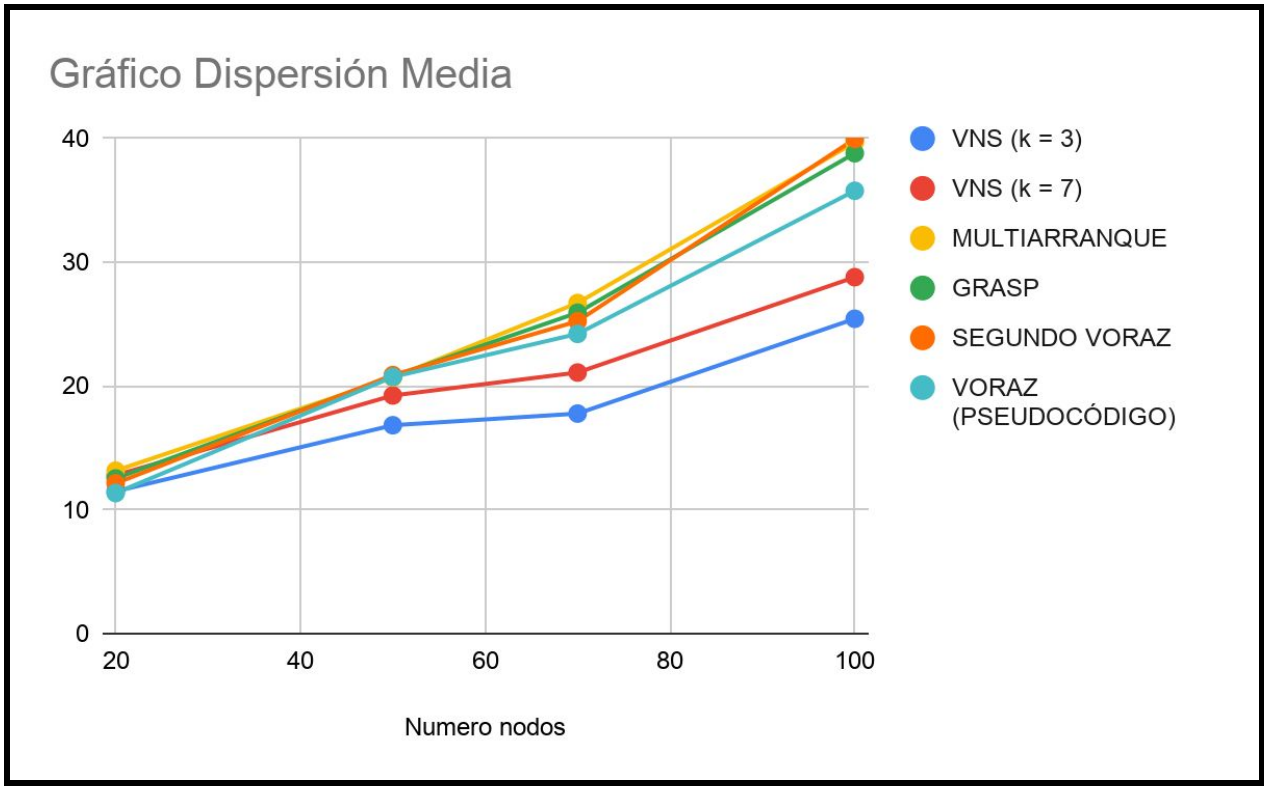
Primero, vamos a empezar con las gráficas referentes al valor de la dispersión media. Para ello, todas las tablas se han juntado en 1 (está visible en el drive o en el excel adjunto localmente), obteniendo dos tipos de gráficas. Vamos a comenzar viendo la referente al diagrama de barras:



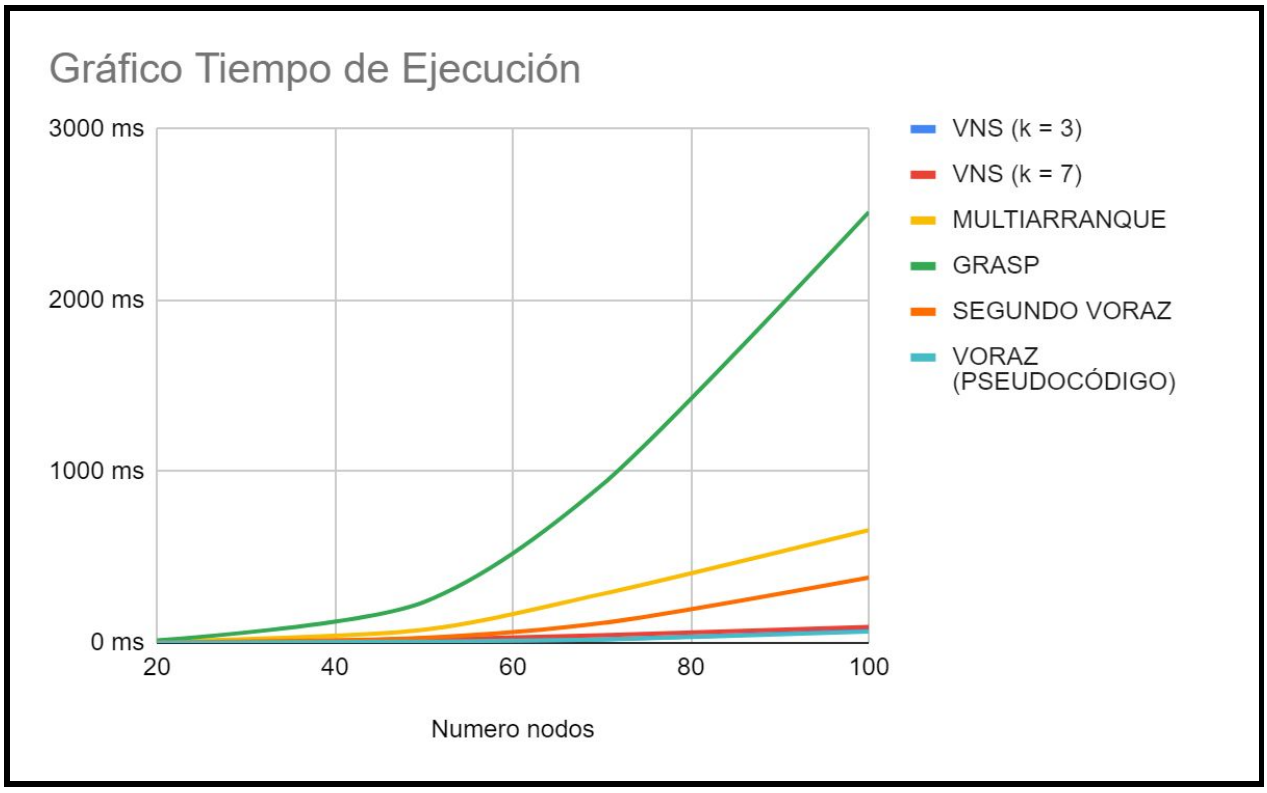
Bueno, tal y cómo vemos, los algoritmos suelen dar valores similares, pero vemos que el caso del color rojo y azul (ambos son los casos de VNS). Vemos que son precisos en cierta medida para problemas pequeños cómo es el caso del de 20 nodos. Ya conforme aumentamos el tamaño, los resultados medios van siendo peores. Esto se puede deber a varios elementos.

Vayamos al caso de la barra de color azul. Aquí nos encontramos que el tamaño máximo que puede tomar el entorno es 3, además sabemos que el número de iteraciones con y sin mejora son bajos. Todo esto influye en la resolución del algoritmo. Por ello, decidí hacer otra prueba de VNS, pero aumentando el tamaño del entorno. En este caso se ve una cierta mejoría. Por lo que deducimos que en mayor medida lo que puede afectar a que nuestros resultados no sean del todo

correctos es porque los valores de las iteraciones son muy bajos. A continuación muestro otro tipo de gráfico para entender mejor lo que se ha hablado:



Una vez que hemos comentado los valores de la dispersión, me gustaría hacer una mención a los tiempos de ejecución. En mi caso, los he determinado en milisegundos para que la visualización de la gráfica fuera más clara. Una vez obtenidas las tablas de una forma similar a la que comentaba anteriormente, así mismo, en el enlace [2] se podrá acceder a la misma y ver los datos completos. El gráfico obtenido es el siguiente:



Haciendo un pequeño análisis, nos damos cuenta que *GRASP*, para ya problemas mayores a 70 nodos, empieza a tardar bastante tiempo. Esto se puede deber a que, las parte de la construcción, donde realiza los RLC, le cuesta bastante computacionalmente hablando. Y los demás algoritmos se mantienen en unos valores normales.

8. REFERENCIAS

[1] Documentación

[2] Tablas Completas

[3] Clase Algoritmo (Herencia)