EPD EVALUABLE II

Algorítmica I

Autores:

- **Jaime Baquerizo Delgado (EPD 12)**
- **Víctor Jesús Reina López (EPD 13)**

Fecha de entrega: 12/12/2021

INTRODUCCIÓN

—-----¿Qué se ha hecho?-------
Esta epd consiste en resolver el problema del viajante de comercio(TSP) estudiando coste y complejidad y comparando los algoritmos.

- El código consiste en calcular los tiempos de tres algoritmos diferentes para la solución del problema del TSP. Para ello es necesario leer los ficheros facilitados donde se encuentran los datos. En esta lectura guardaremos el contenido del fichero en una variable de tipo matriz de carácteres para así poder aplicar fácilmente los algoritmos.

- Primeramente experimentamos con el algoritmo Aleatorio 1, es decir, vamos a buscar una solución aleatoria para el problema. Nuestro criterio de parada consiste en un número máximo de iteraciones. Vamos a ir recorriendo las “ciudades” y guardando nuestro mejor camino en una variable, que en caso de encontrar un camino mejor se actualizaría esta.

- Luego aplicamos el algoritmo Aleatorio 2, que es idéntico al anterior pero con un criterio de parada diferente. Este criterio de parada consiste en declarar un contador que comenzará desde 0. Cada vez que encontremos un camino mejor que el actualmente “candidato a ser la solución” se actualiza a 0, y en caso de no encontrarlo esta variable se incrementará en una unidad. Por lo tanto vamos a definir una variable máxima, que nos servirá como límite de búsqueda para detener nuestro algoritmo. Es decir, si en 100 intentos, por ejemplo, no se ha encontrado ningún camino mejor al actual se pararía, ya que si no encontramos una solución nueva en ese número de oportunidades significaría que tenemos un camino es bastante óptimo.

- Por último hemos implementado el algoritmo Voraz, permite al viajante elegir la ciudad no visitada más cercana como próximo movimiento. Este algoritmo devuelve rápidamente una ruta corta, lo cuál puede llegar a ser un problema porque no tiene en cuenta que la siguiente distancia más cercana puede no ser el camino más óptimo.

- Para calcular los tiempos de ejecución, hemos implementado una serie de bucles for, los cuales llaman a los métodos de estos algoritmos repetidas veces un haciendo una sumatorio de todos estos tiempos. Una vez terminado las repeticiones haremos una media que será el tiempo de ejecución medio de nuestro algoritmo.

—---Resultados y conclusiones—---

Los resultados obtenidos han sido que el algoritmo voraz ha encontrado el camino más corto y además en el mejor tiempo, seguido de este, del algoritmo aleatorio con contador y por último el algoritmo aleatorio con un número determinado de iteraciones.

Pseudocódigo Algoritmo voraz:

```
while (recorrido(actual, camino, size)) {
        actual++;
}
next = actual + 1;
        while (next < camino.length - 1) {
                while (recorrido(next, camino, size)) {
                        if (next < camino.length - 1) {
                        next++;
                        }
                }
        if (!recorrido(next, camino, size) && next < camino.length - 1) {
                if (getDistancia(distancias, inicio, actual) > getDistancia(distancias,
        inicio, next)) {
                        actual = next;
                        next++;/
        }else {
                next++;
        }
        }

}
return actual;
```

Explicación algoritmo voraz: Un algoritmo voraz (también conocido como goloso, ávido, devorador o greedy) es una estrategia de búsqueda por la cual se sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento.

Los algoritmos voraces se utilizan para resolver problemas de optimización. ... La clave de estos algoritmos voraces está en la función que selecciona el mejor candidato. Podríamos decir que el algoritmo va seleccionando de forma codiciosa al candidato óptimo en ese momento, despreocupándose del resultado final.

Para solucionar el problema del TSP con este algoritmo primero debemos comprobar que la ciudad a comparar no está ya en nuestro camino con la ayuda del método “recorrido”. Después denominamos la ciudad next como la siguiente de la ciudad actual, comprobamos que no es la última ni la hemos pasado. Seguidamente comprobamos que la distancia de Next es menor que la de la ciudad actual, y en caso afirmativo next pasaría a ser como la nueva ciudad actual, en caso negativo seguimos con nuestra búsqueda sin modificar la ciudad actual. De esta manera buscamos siempre que los caminos con menos coste.

Pseudocódigo Algoritmo Aleatorio con número determinado de Iteraciones:

```
        while (iterador < nItMax) {
                int[] caminoCandidato = getCaminoAleatorio(distancias, posicionInicial);
                if (camino == null || getDistanciaTotal(distancia, caminoCandidato) <
getDistanciaTotal(distancias, camino)) {
                        camino = caminoCandidato;
                }
                iterador++;
        }
      return camino;
```

Explicación Algoritmo Aleatorio con número determinado de iteraciones: Un algoritmo aleatorizado es un algoritmo que toma decisiones aleatorias como parte de su lógica. El análisis probabilístico de algoritmos estima la complejidad computacional de algoritmos o problemas asumiendo alguna distribución probabilística del conjunto de todas las entradas posibles.
Los algoritmos aleatorizados a diferencia de los convencionales tienen una entrada adicional que son los números aleatorios y que se obtienen durante la ejecución, es decir, se usa algún grado de aleatoriedad como parte de su lógica; es importante decir que este tipo de algoritmos son la base esencial de la simulación.

En cuanto al tsp con este algoritmo, lo que hemos hecho ha sido un bucle while, en el que mientras que el número de iteraciones no sea el número máximo de iteraciones, buscará caminos alternativos. Guardamos un camino nuevo que será un candidato a mejorar el camino que tenemos actualmente mediante el método getCaminoAleatorio y por último en un if se comprueba si un camino es mejor que el camino candidato, y el coste de cada uno. Si el coste del camino candidato es menor que nuestro actual camino, pondremos que nuestro actual camino será el camino candidato y en el caso de que el camino actual sea mejor, el iterador seguirá aumentando.

Pseudocódigo Algoritmo Aleatorio con contador:

```
        while (contador < contLimit) {

                int[] caminoCandidato = getCaminoAleatorio(distancia, posicionInicial);
                if (camino == null || getDistanciaTotal(distancia, caminoCandidato) <
getDistanciaTotal(distancia, camino)) {
                        camino = caminoCandidato;
                        contador = 0;
                } else {
                        contador++;
                }
         }

        return camino;
```

En este algoritmo, es idéntico al anterior pero con un criterio de parada diferente. Este criterio de parada consiste en declarar un contador que comenzará desde 0. Cada vez que

encontremos un camino mejor que el actualmente “candidato a ser la solución” se actualiza a 0, y en caso de no encontrarlo esta variable se incrementará en una unidad.

EXPERIMENTACIÓN

—--Experimentación con número de iteraciones y valor límite del contador—--

```
run:
Camino calculado con el algoritmo Aleatorio1: [14,272,164,21,27,
Distancia total: 30946.303
- Tiempo de ejecución: 134858241 ns


Camino calculado con el algoritmo Aleatorio2: [14,36,181,111,212
Distancia total: 31679.842
- Tiempo de ejecución: 34000248 ns
```

- Algoritmo Aleatorio1, Número iteraciones: 1000
- Algoritmo Aleatorio2, Límite Contador: 100

```
Camino calculado con el algoritmo Aleatorio1: [14,145,248,219,2
Distancia total: 30910.572
- Tiempo de ejecución: 79649086 ns


Camino calculado con el algoritmo Aleatorio2: [14,231,8,227,232,
Distancia total: 31891.129
- Tiempo de ejecución: 11788606 ns
```

- Algoritmo Aleatorio1, Número iteraciones: 500
- Algoritmo Aleatorio2, Límite Contador: 50

```
run:
Camino calculado con el algoritmo Aleatorio1: [14,19,148,4
Distancia total: 31924.324
- Tiempo de ejecución: 18816119 ns


Camino calculado con el algoritmo Aleatorio2: [14,92,72,74
Distancia total: 31343.928
- Tiempo de ejecución: 7510771 ns
```

- Algoritmo Aleatorio1, Número iteraciones: 100
- Algoritmo Aleatorio2, Límite Contador: 25

- **—--Graficas—--**

	algVoraz	algAle1	algAle2
KROA150	939224	55001364	9953459
BERLIN52	154499	13689432	2009802
A280	5261808	136509301	25066137
KROA100	396827	33009062	5471420
KROA200	1321110	55231494	9473264
VM1084	185171187	1597656509	269761776
VM1748	1024039509	5766778118	1049637533

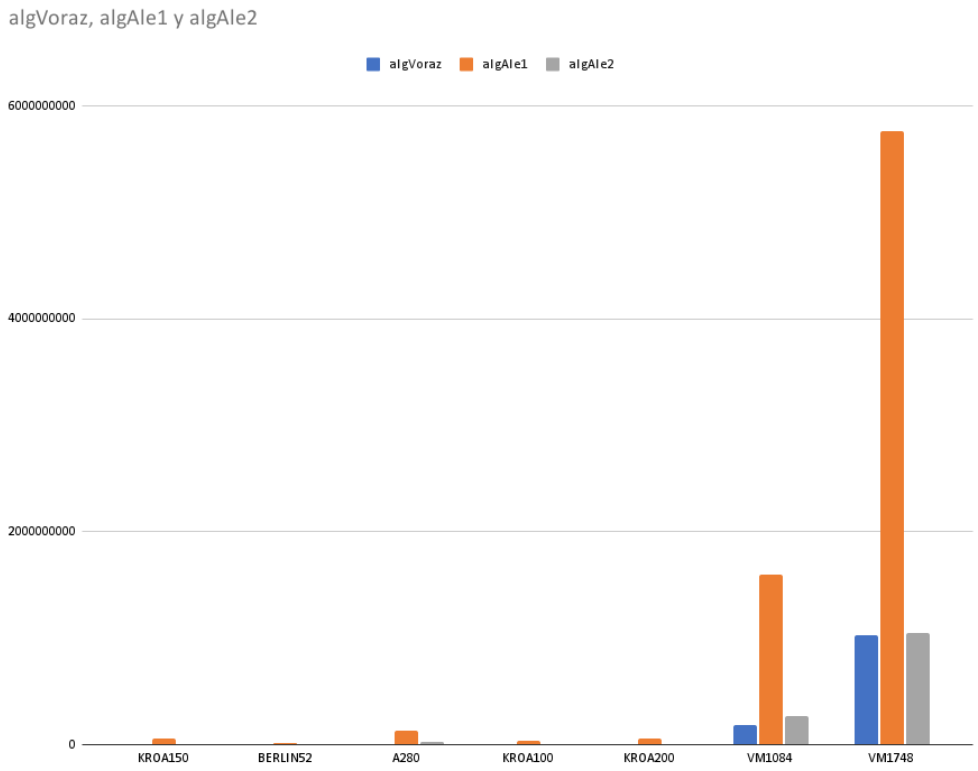
TIEMPOS

	algVoraz	algAle1	algAle2
KROA150	33514,074	226332,36	230690,44
BERLIN52	8980,919	24652,145	25686,176
A280	3284,1362	30993,436	31778,242
KROA100	26679,193	145479,72	139404,81
KROA200	35969,39	291798,3	315932,06
VM1084	301469,16	8244339	8218608,5
VM1748	406401.44	1,45E+14	1,46E+14

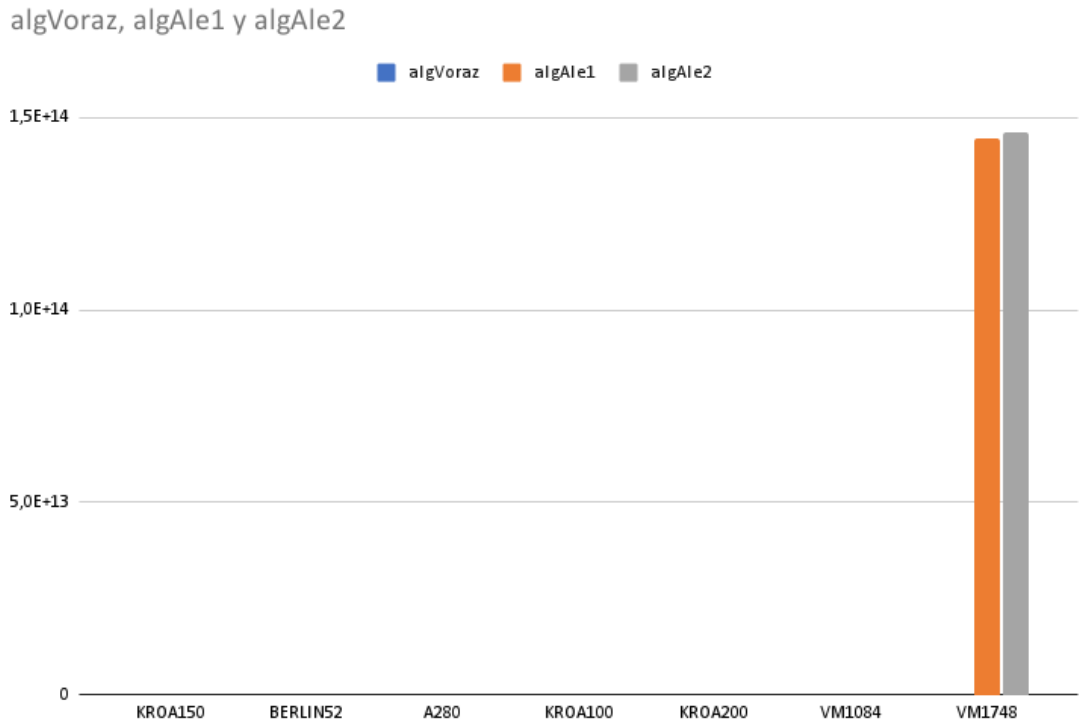
COSTES

No hemos puesto los tiempos y costes del archivo usa13509(insuficiente tiempo de ejecución) ya que al haber tanta diferencia con los demás, no se aprecia bien en los gráficos y por tanto no se puede hacer un buen análisis de los resultados, de hecho pasa lo mismo con los archivos vm1084 y vm1748, aquí podemos verlo:

TIEMPOS

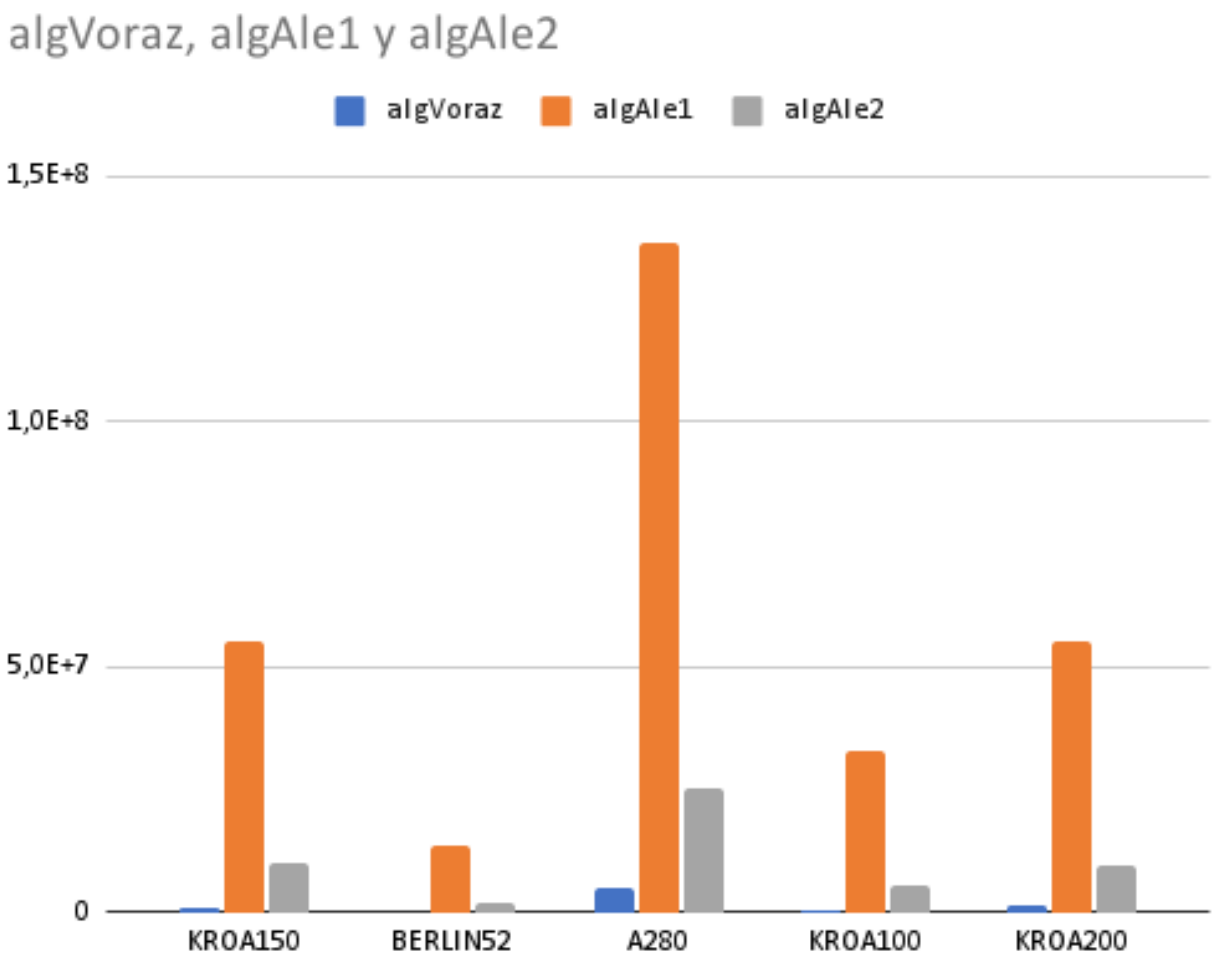


COSTES/DISTANCIAS

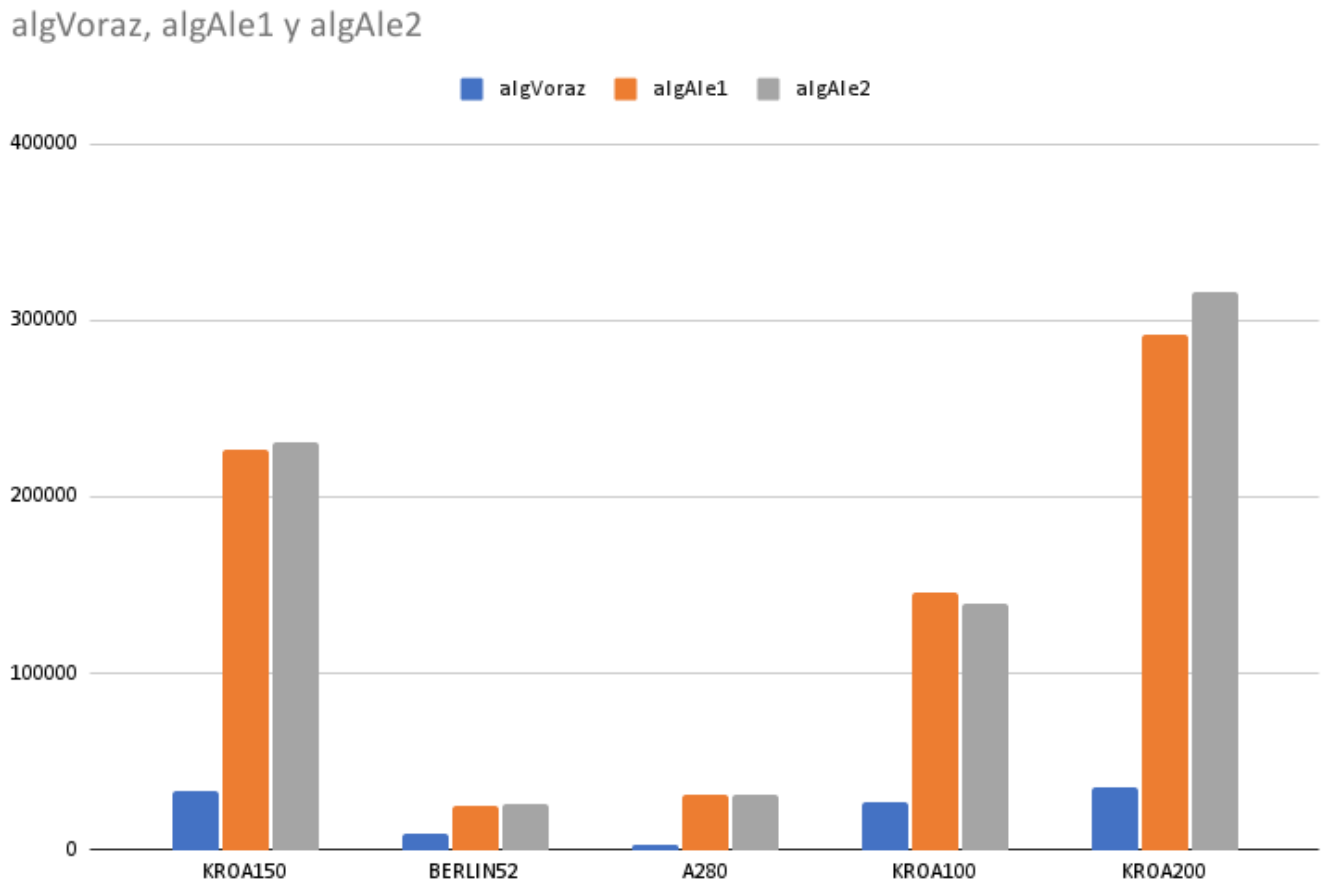


Eliminamos estos dos archivos de la gráfica para poder analizar bien los resultados obtenidos:

TIEMPOS



COSTES/DISTANCIAS



¿Alguno de los dos algoritmos aleatorios garantiza una solución óptima?

Claramente no, ya que como indica el nombre se caracteriza por la aleatoriedad.
Realizan operaciones o toman decisiones distintas de forma aleatoria, se pueden utilizar cuando no es posible explorar todas las posibilidades o resulta ser costosa, por lo que no es óptimo utilizarlo en cualquier problema, sino que hay que saber cuando es eficiente y es útil usarlo.
Por ejemplo, supongamos tenemos dos puertas y detrás de una de ellas hay un premio y en la otra no hay nada, nuestro interés está en descubrir ese premio, un algoritmo determinista abriría una puerta y luego la siguiente, en el peor de los casos las dos puertas serían abiertas. Con un algoritmo aleatorio existe la misma posibilidad que tenemos cuando lanzamos una moneda al aire, es posible que en nuestro primer intento podamos abrir la puerta correcta y obtener el premio sin necesidad de abrir la siguiente.

CONCLUSIONES

Los resultados se pueden observar claramente con los gráficos. Podemos apreciar que los dos algoritmos aleatorios (criterio de parada de iteraciones y contador), los resultados que ofrecen son bastante similares, mientras que los tiempos suelen ser más elevados los del algoritmo aleatorio con contador ya que necesita ejecutarse muchísimas más veces que el de iteraciones.
Lo más claro de los gráficos es que el algoritmo voraz es el más óptimo, ya que tarda mucho menos tiempo y además con el mejor resultado. Es lógico que sea mejor que los otros dos ya que mientras en los algoritmos se realizan aleatoriamente en el caso del algoritmo voraz coge el pueblo más cercano, por lo que siempre va a ser mejor que los otros códigos.
El algoritmo voraz puede no ser la mejor solución aun así, ya que siempre va a coger la ciudad más cercana pero así no sabremos si podría haber una distancia más corta por cualquier otro camino, es decir, tan solo tiene en cuenta la distancia óptima más cercana, sin mirar más allá de esa ciudad.

BIBLIOGRAFÍA

https://silvercorp.wordpress.com/2013/08/20/que-es-un-algoritmo-aleatorio/
https://www.neoteo.com/el-algoritmo-voraz/
EPD7- ALGORITMO VORAZ
EB- ALGORITMO VORAZ
https://www.youtube.com/watch?v=etQN4EfYN7k
https://www.youtube.com/watch?v=YCCE4sbmWrw