

Doble Grado en Ingeniería Informática y Matemáticas

Algorítmica

Práctica 2 - Amistad en la granja

David Kessler Martínez

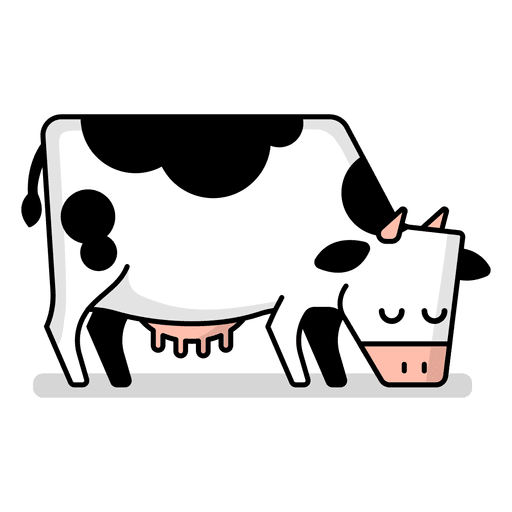
[dkesslerm03@correo.ugr.es](mailto:dkesslerm03@correo.ugr.es)

Santiago López Cerro

[santiloce@correo.ugr.es](mailto:santiloce@correo.ugr.es)

Antonio Javier Rodríguez Romero

[antoniojrr@correo.ugr.es](mailto:antoniojrr@correo.ugr.es)



Granada, curso 2022/2023

ÍNDICE

[**0. Introducción 3**](#_8eoju0duuci6)

[0.1. Objetivos 3](#_xbg6s8xq9o0d)

[0.2. Definición del problema 3](#_tg1se67ysq3n)

[0.2.1. Entorno de análisis 4](#_zb6ub7wi2t8i)

[Santiago López Cerro 4](#_t4s09tmx7yko)

[David Kessler Martínez 5](#_nqrwkvdjevuo)

[Antonio Rodríguez Romero 5](#_3hb0b4ygmxff)

[0.2.2. Obtención de tiempos 5](#_ofxykke5mnwh)

[**1. Algorítmico específico 6**](#_p45quo6kcw6a)

[1.1. Estudio teórico 7](#_8ah6a0jjqoiq)

[1.2. Estudio empírico 7](#_izposo98jgsy)

[1.3. Estudio híbrido 8](#_1h0pid61zfz)

[**2.Algoritmo DyV 9**](#_z75azyrs13gn)

[2.1. Explicación 9](#_unn3pemwgsmf)

[2.2. Estudio teórico 10](#_n8dsfflhfwwj)

[2.3. Estudio empírico 11](#_epkikhcet7wj)

[2.4. Estudio híbrido 11](#_ox4kp9u9s2m7)

[2.5. Cálculo de los umbrales 12](#_5o2vrm8j208i)

[2.6. Cálculo del umbral óptimo 13](#_pdbzclqiejfk)

[2.7. Umbrales de tanteo 13](#_rztuxsaq9d51)

[**3. Conclusiones**](#_6hljqoln0267) **15**

# 

# 

# 0. Introducción

Porcentajes del trabajo realizado por cada alumno:

David Kessler Martínez - 32%

Santiago López Cerro - 33%

Antonio Javier Rodríguez Romero - 35%

David se ha encargado del diseño e implementación del algoritmo específico, además del análisis de la eficiencia teórica, empírica e híbrida de este algoritmo. Santiago ha diseñado e implementado el generador de casos, además de calcular los umbrales para las implementaciones de algoritmos realizadas. Finalmente, Antonio ha estudiado en profundidad el problema de “Amistad en la granja”, además de diseñar e implementar el algoritmo que usa la técnica “Divide y Vencerás” y ha realizado el análisis de la eficiencia teórica, empírica e híbrida de este algoritmo. Los tres hemos contribuido a la elaboración de la memoria para documentar el trabajo realizado, además de supervisar el trabajo realizado por cada uno de los miembros del grupo. Así, afirmamos que los tres miembros entendemos a la perfección todas las partes de la práctica, independientemente de quién se haya encargado de ellas.

## 0.1. Objetivos

El objetivo de esta práctica es comprender el funcionamiento de la técnica de diseño de algoritmos “Divide y Vencerás”, al mismo tiempo que comprobamos su efecto en la eficacia de estos. Para ello se plantea un problema, definido más adelante, el cual debe ser resuelto utilizando un algoritmo específico y un algoritmo que aplique la técnica “Divide y Vencerás”, de forma que con este contexto se nos brinda la posibilidad de estudiar los tiempos de resolución del problema con estas dos opciones distintas.

## 

## 0.2. Definición del problema

En nuestro caso nos encargaremos de abarcar el problema propuesto de amistad en la granja. En este el problema considera la situación de un amplio terreno, por donde las vacas de un granjero se encuentran repartidas. Sabiendo que dos vacas son más afines cuanto más cerca se encuentra la una de la otra, lo que pretendemos averiguar es cual es aquella pareja de vacas que es más afín. Así pues nuestro objetivo será calcular dicho par de vacas para cada distribución de la misma.

Para ello consideramos el terreno como el plano euclídeo, en donde siendo las vacas puntos en el plano podremos determinar la distancia entre ellas.Así pues nuestra entrada de datos será un vector del tipo pair<int,int>, en donde cada elemento del vector corresponderá a una vaca distinta con su posición correspondiente en el plano. Así pues con estos datos y con un método que nos calcule la distancia entre dos vacas en el plano podremos empezar a considerar situaciones:

* En el primer caso del problema, implementaremos una solución que no recurra al método Divide y Vencerás, por lo que es de esperar que su eficiencia sea mucho menor. Para este caso probaremos a buscar la pareja más afín considerando vectores de distintos tamaños, desde 1000 hasta 20000 implementando saltos de 1000.

Una vez realizada la implementación, se llevará a cabo un análisis de eficiencia teórica del algoritmo, que contrastaremos con un estudio empírico de la eficiencia y, más tarde, híbrido

* En el segundo caso será cuando implementemos en nuestro algoritmo la técnica de Divide y Vencerás. Ante el interés de comparar los resultados de este caso con el anterior, tomaremos los mismos números de vacas sobre los que trabajar. Así pues, igual que antes, con el algoritmo implementado y los datos obtenidos nos dedicaremos al correspondiente análisis de eficiencia teórico, empírico e híbrido, con sus respectivas comparativas.

Por último, realizaremos un resumen de las conclusiones obtenidas a lo largo de la práctica y en base a los resultados reflejados.

### 0.2.1. Entorno de análisis

Estos los distinto entornos de cada uno de los compañeros en donde se ha realizado el estudio de los algoritmos:

#### Santiago López Cerro

* Dispositivo: HP Pavilion x360 Convertible 14-dw1xxx
* Procesador: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
* RAM instalada: 8,00 GB (7,65 GB usable)
* Arquitectura: Sistema operativo de 64 bits, procesador basado en x64
* CPU(s): 4
* Hilo de procesamiento por núcleo: 1
* Socket(s): 1
* Núcleos por socket: 4

Especificaciones de Windows:

* Sistema Operativo: Windows 10 Home

Acerca de mi maquina virtual:

* Memoria de base: 2048 MB
* Procesadores: 4
* Orden de arranque: Disquete, Óptica, Disco Dura

#### 

#### David Kessler Martínez

Dispositivo: Lenovo IdeaPad 5 15ALC05

Procesador: AMD Ryzen 5 5500U with Radeon Graphics

RAM: 16 GB

Arquitectura: x86\_64

Sistema operativo: Ubuntu 22

CPU(s):12

Hilo de procesamiento por núcleo: 2

Socket(s): 1

Núcleos por socket: 6

#### Antonio Rodríguez Romero

Dispositivo: ASUS TUF Dash F15

Procesador: 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30 GHz

RAM: 16GB

Arquitectura: x86\_64

Sistema operativo: Ubunto 22

CPU(s): 8

Hilo de procesamiento por núcleo: 2

Socket(s): 1

Núcleos por socket: 4

En cuanto a la compilación, hemos utilizado el compilador g++ con nivel de optimización 1.

### 

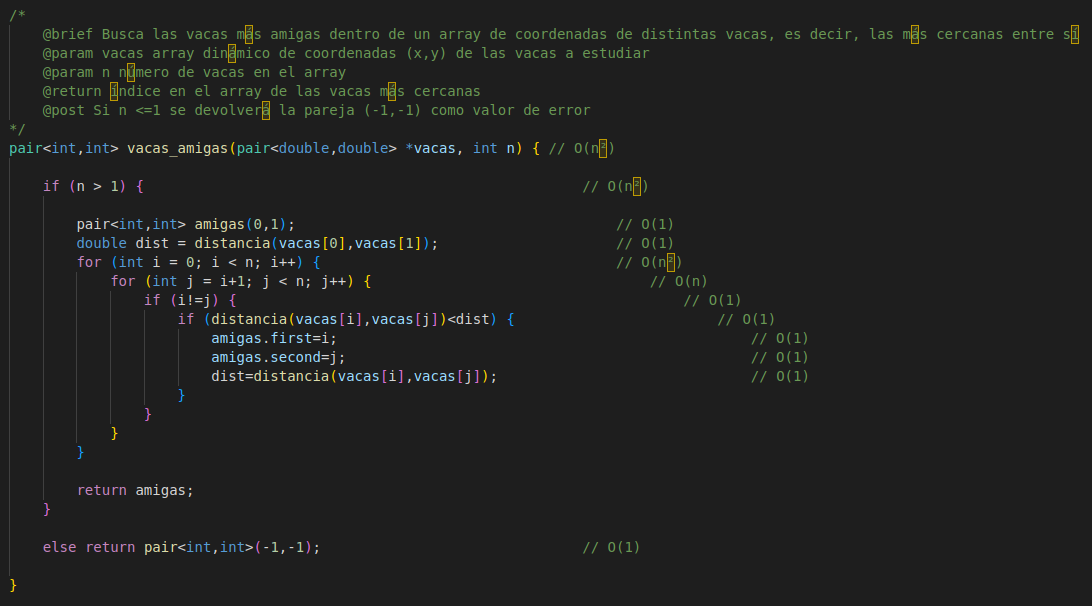
### 0.2.2. Obtención de tiempos

En lo relativo al método de medición de tiempo hemos decidido optar por el uso de la biblioteca ctime, implementando una llamada a la función clock() antes y después de la ejecución del método que nos proporciona la pareja más afín y dividiendo posteriormente el resultado de la resta entre el tiempo final menos el inicial entre CLOCKS\_PER\_SEC. Para conseguir que los datos además sean lo más representativos posible, el algoritmo se ejecutará 5 veces para después dividir este tiempo entre 5. De esta forma suavizamos la disparidad de tiempos de una toma a otra.

# Algorítmico específico

Nuestro algoritmo específico, o base, se trata de un algoritmo muy simple de orden O(n²) el cual se basa en la comparación de distancias entre todas las vacas que hay en el problema, guardando el índice de las vacas con la distancia menor y esta distancia. Una vez se hayan comparado todas las parejas de vacas posibles, tendremos los índices de las más cercanas entre sí.

Este algoritmo consta de dos bucles anidados, uno que comenzará en el principio del *array* que guarda las coordenadas de las vacas, y otro que comenzará en el elemento siguiente al que esté situado el bucle exterior. Su implementación en C++ viene dada por el siguiente código:



Hemos utilizado dos objetos de tipo pair distintos. Cabe explicar la diferencia entre ambos. El objeto pair<double,double> \*vacas es un vector dinámico de pares de doubles, que almacena las coordenadas de las vacas con las que trabajamos. En cambio, pair<int,int> amigas indica el par de vacas que consideramos mejores amigas, según su posición dada por el vector \*vacas.

Finalmente, cabe mencionar también que llamamos a una función distancia(), la cual calcula la distancia entre dos vacas según la fórmula , donde (x1, y1) son las coordenadas de la primera vaca a comparar y (x2, y2) son las coordenadas de la segunda vaca. Ahora sí, pasamos al análisis de la eficiencia teórica, empírica e híbrida.

## 

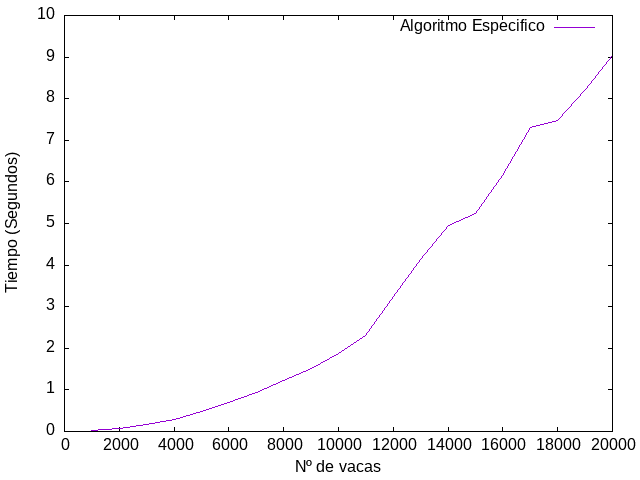
## 1.1. Estudio teórico

Vemos que nuestro algoritmo se encontrará dentro del bucle interno durante la mayor parte del tiempo de ejecución. Llamemos a este tiempo “t”. De esta forma, con n elementos en el *array* vacas, tenemos que

y vemos como obviamente . Por lo tanto, diremos que se trata de un algoritmo de orden O(n²) y que la función que mejor ajustará los tiempos de este algoritmo será un polinomio de segundo grado, que tendrá la forma . Pasemos al estudio empírico.

## 

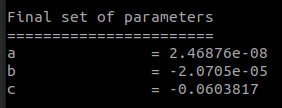
## 1.2. Estudio empírico

Escalando el problema a distinto número de casos, comenzando con 1000 vacas y llegando hasta 20.000, con saltos de 1000 vacas, hemos estudiado los tiempos de ejecución del algoritmo. Para conseguir un resultado lo más representativo posible, cada uno de estos tiempos se ha obtenido como la media de 5 mediciones. Así, hemos obtenido los siguientes resultados graficados:

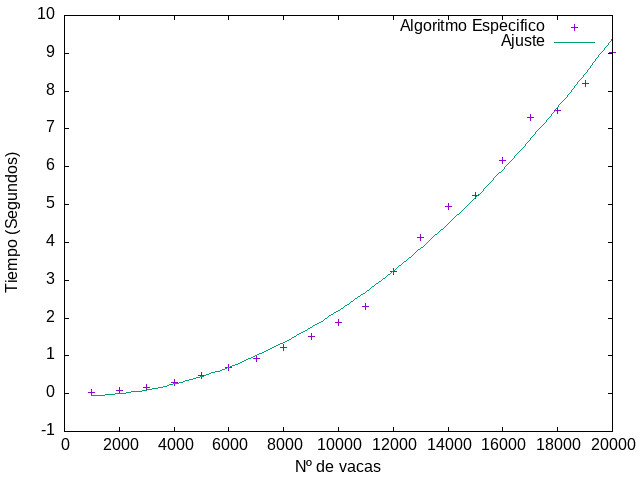
# 

## 1.3. Estudio híbrido

Como hemos visto en el análisis teórico, se trata de un algoritmo de orden cuadrático, luego el mejor ajuste que podemos realizar teóricamente sobre esta gráfica será con la función polinómica . Estudiemos primero las constantes ocultas. Con la ayuda de gnuplot, obtenemos:



Por tanto la función de ajuste es f(x) = 2.46876e-08\*x² - 2.0705e-05\*x - 0.0603817. Además vemos que se trata de un buen ajuste pues la varianza tiende a cero→ Varianza: 0.0739729

Hemos realizado un ajuste por regresión mediante mínimos cuadrados con los datos obtenidos, lo que vemos en la siguiente gráfica. 

Vemos que, según el ajuste híbrido, nuestro algoritmo específico tarda 24.796 segundos en encontrar las vacas con una mayor amistad en un campo de 1.000.000 de vacas. Esto será curioso a la hora de compararlo con el algoritmo DyV.

# 2. Algoritmo DyV

## 2.1. Explicación

Nuestro algoritmo basado en la técnica *Divide y Vencerás* se trata de un algoritmo de orden , que consiste en una función recursiva que hemos denominado vacas\_amigas\_DyV() la cual recibe como parámetros un array de coordenadas de vacas del tipo de dato pair<double, double>, que representan sus coordenadas en el plano, y un valor del tipo int referido al tamaño del vector, devolviendo finalmente las coordenadas de aquellas vacas más afines.

En lo referente a la implementación del código, comenzamos creando un dato tipo pair<pair<double,double>,pair<double,double>> denominado extrem. En este, mediante una función auxiliar buscamos las esquinas opuestas de la zona del plano que contendrá todas las vacas pasadas en el vector.

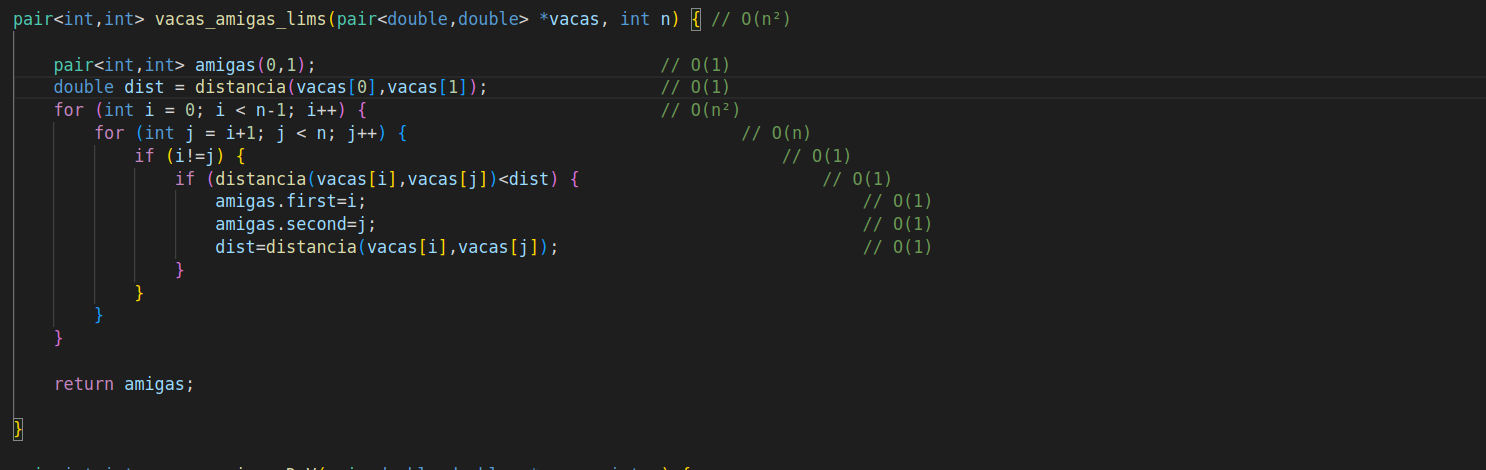
Una vez tomado este dato, comprobaremos si el número de vacas es menor que el umbral que hayamos establecido, de forma que si esto se cumple, se ejecutará el algoritmo cuadrático. Este debe definirse con un mínimo de consideración ya que, al realizar llamadas recursivas a una función que creará vectores dinámicos por cada llamada, si el umbral es excesivamente pequeño, la pila se llenará y no podrá ejecutarse, mientras que si es demasiado grande, no conseguiremos gran ganancia de tiempo con respecto al algoritmo de orden cuadrático. Para realizar esta división de la zona en dos, utilizaremos una función auxiliar *separar(),* la cual se encargará de conseguir cada una de las vacas de cada sector y devolver la frontera entre estos a través de una pareja de puntos, de forma que estos representarán la recta divisoria. Por tanto, por cada llamada se reduce en dos mitades el área inicial. Finalmente, para hallar cual es la pareja más afín de un cuadrante no nos bastará con solo hallar los pares de cada subcuadrante generado, sino que además consideraremos la posibilidad de que la pareja más afín esté en la frontera. Para asegurar el correcto funcionamiento del algoritmo, comprobaremos la afinidad de las vacas fronterizas con otra función auxiliar *vacas\_amigas\_frontera(),* que nos devolverá las coordenadas de la pareja cercana a la frontera con más afinidad, de forma que compararemos las tres parejas para encontrar la más afín.

Figura 5: Función vacas\_amigas\_lims()

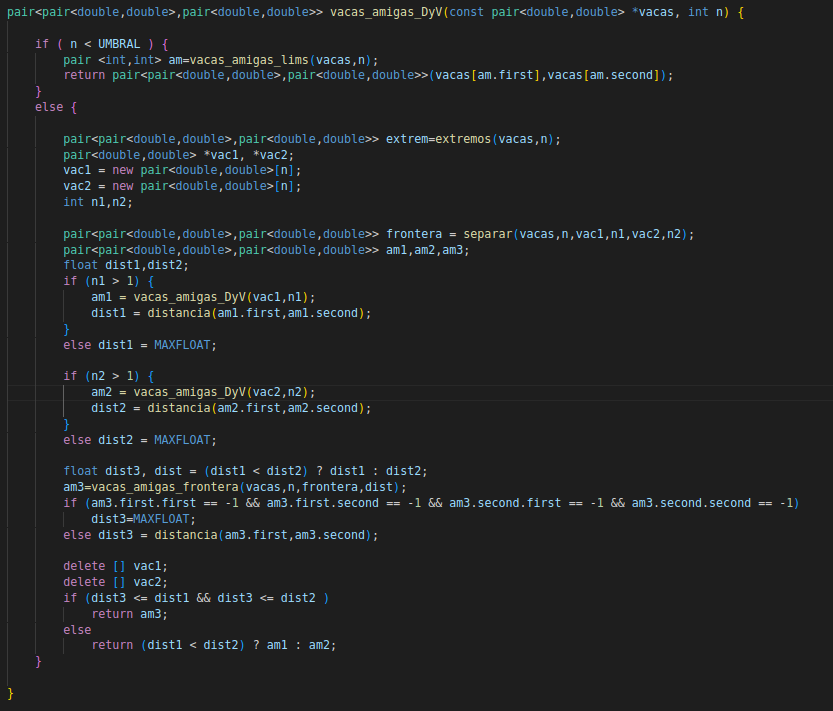


Figura 6: Función vacas\_amigas\_DyV()

## 2.2. Estudio teórico

Nuestro algoritmo sigue un desarrollo muy parecido al algoritmo de ordenación Mergesort: divide el vector vacas en dos partes que, de media, supondremos iguales entre sí, y se vuelve a aplicar de forma recurrente a cada una de ellas. Una vez hecho esto, encuentra la pareja de vacas con mayor afinidad en cada subvector, hasta llegar a la pareja de vacas con mayor afinidad del vector original. Si el número de vacas del vector es menor que nuestra constante UMBRAL, entonces se encuentra la pareja de vacas con mayor afinidad mediante nuestro algoritmo base, que es cuadrático. Más adelante daremos una versión alternativa para el caso base que se nos ha ocurrido sobre la marcha. Esto es:

si

Haciendo el cambio de variable obtenemos:

si

Renombrando , resulta

La ecuación de recurrencia que se deriva de esta ecuación es:

Luego la solución general será:

Deshaciendo el cambio de variable, obtenemos:

Por tanto

## 2.3. Estudio empírico

Para realizar una buena comparativa entre ambos algoritmos, tomaremos los mismos números de vacas que en el análisis del específico aunque, claramente, los resultados obtenidos en este caso son considerablemente más pequeños.

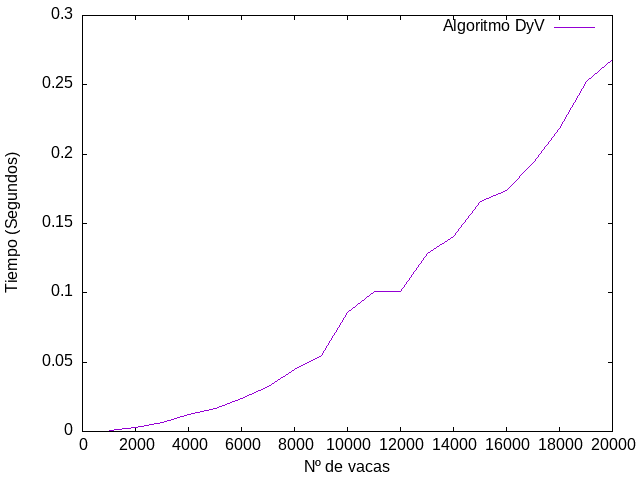


Figura 7: Algoritmo DyV

## Vemos que, como esperábamos, encontramos una tendencia creciente con el aumento de individuos a estudiar, aunque con un crecimiento mínimo en comparación al algoritmo que no aplica la técnica DyV.

## 2.4. Estudio híbrido

Como hemos visto en el análisis teórico, se trata de un algoritmo de orden *O(n\*log(n))*, luego el mejor ajuste que podemos realizar teóricamente sobre esta gráfica será con la función logarítmica . Estudiaremos primero el valor de las constantes ocultas. Con la ayuda del software *gnuplot*, que realizará un ajuste por mínimos cuadrados, obtenemos:

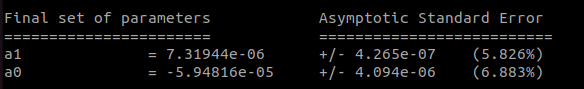


Figura 8: Constantes ocultas

Con lo cual, la función que mejor ajustará los resultados de este algoritmo, según los datos obtenidos, será:

Vemos que, aunque encontramos algo de dispersión entre los puntos, se trata de un buen ajuste puesto que la varianza es muy cercana al cero (Varianza=6.82113e-08).

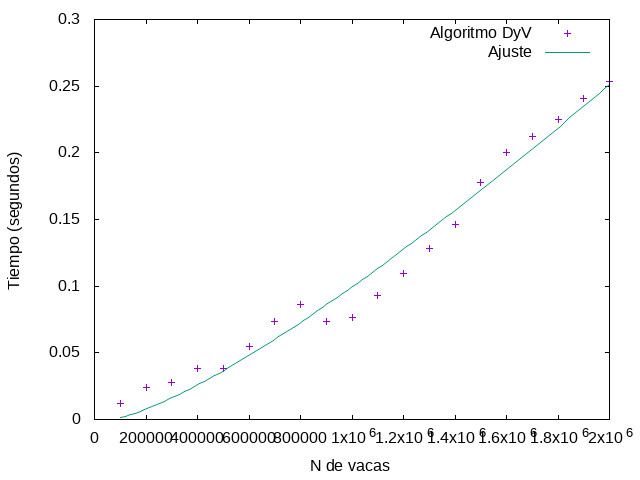


Figura 9: Análisis híbrido DyV

Como observamos con el caso específico, para buscar la mayor afinidad entre 1.000.000 de vacas, tardamos 24.796 segundos. El análisis híbrido nos permite consultar cuánto se tardaría en buscar la mayor afinidad entre 1.000.000 de vacas para el algoritmo DyV, obteniendo un resultado de 0,0768232 segundos.

## 2.5. Cálculo de los umbrales

Para un algoritmo basado en la técnica DyV, sabemos que la elección de un buen umbral es vital para el correcto funcionamiento del mismo. Procedemos a calcular el umbral teórico, óptimo, y de tanteo.

## 2.6. Cálculo del umbral óptimo

Una vez conseguido el ajuste del algoritmo DyV, podemos crear una inecuación que relacione los tiempos de ambos algoritmos de forma que obtengamos el número de vacas tal que, desde 0 hasta este, el algoritmo cuadrático tarde menos que el logarítmico, de forma que este será nuestro umbral óptimo



Tratándola de resolver esta inecuación nos damos cuenta de que, teóricamente, para cualquier umbral con sentido (UMBRAL >= 2), el algoritmo DyV tardará menos que el cuadrático. Veamos si empíricamente llegamos a la misma conclusión.

## 2.7. Umbrales de tanteo

Para ver la tendencia general de los tiempos en cuanto a los umbrales, primero hemos tomado algunos un poco mayores al umbral óptimo teórico, que es 2. Los datos obtenidos, resultantes de hacer la media de 5 medidas, son:

**Umbral n1 n2 n3**

10 0.0033625 0.027694 0.246937

20 0.0042175 0.0541025 0.314466

30 0.0039515 0.0357525 0.320485

40 0.002717 0.060728 0.164334

50 0.002847 0.065381 0.139543

60 0.002545 0.06776 0.302947

70 0.0023575 0.0718115 0.287

80 0.0024915 0.051142 0.361689

90 0.0023205 0.048637 0.366134

100 0.002411 0.048 0.358338

110 0.0024935 0.0511625 0.36332

120 0.0023295 0.048715 0.361243

130 0.001987 0.049084 0.355686

140 0.0018945 0.051364 0.343566

150 0.0021755 0.0453715 0.31

160 0.0023615 0.029603 0.321562

170 0.002566 0.032846 0.31653

180 0.002118 0.032787 0.314787

190 0.00217 0.0334095 0.306975

200 0.0022075 0.0328305 0.3047

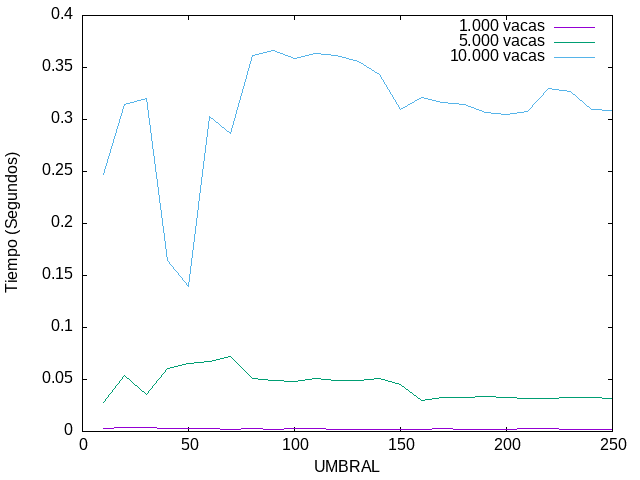


Figura 10: Análisis de umbrales

Observamos que tomando gran número de datos a estudiar, existe un pico negativo bastante significativo en los tiempos tomados con el algoritmo en torno al umbral 50.

Es también aconsejable tener en cuenta que, al trabajar el algoritmo dividiendo la zona de estudio y no las vacas arbitrariamente, dependiendo de la distribución de estas, que es pseudoaleatoria, se puede dar gran variación en los resultados obtenidos.

Por otro lado, como en tamaños bajos de población de estudio no encontramos grandes variaciones con la elección de los distintos umbrales, tomaremos como óptimo el más eficiente para poblaciones grandes, es decir, el mencionado anteriormente (*UMBRAL=50*).

# 3. Conclusiones

Teniendo en cuenta los resultados obtenidos a lo largo de esta práctica, parece evidente que el aplicar la técnica *DyV* en nuestro algoritmo conseguimos una abismal mejoría en tanto que a los tiempos de ejecución obtenidos.

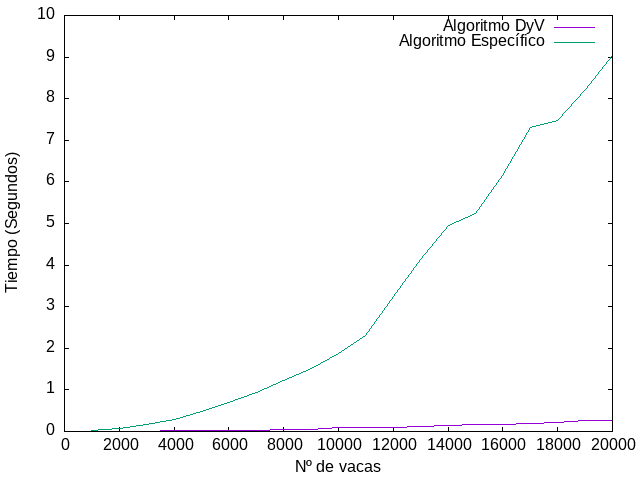


Figura 11: Comparativa de tiempos

Sin embargo, tampoco puede quedar en el olvido la buena elección de un algoritmo específico, ya que, al fin y al cabo, este será la base de nuestra segunda implementación de la solución del problema planteado.

Por otro lado, cabe resaltar que es igualmente importante realizar un estudio acerca de la elección de un umbral u otro, ya que este afectará en gran medida a la eficacia de nuestro algoritmo.