

VC4 – Video Conferencia 4

03MAIR – Algoritmos de Optimización

Agenda

- Revisión de algoritmos exactos.
- Revisión de la Complejidad. Cálculo de operaciones.
- Descenso del Gradiente
- Algoritmos paralelos
- Problemas del trabajo del Seminario
- Revisión de objetivos de la asignatura

VC4 – Repaso de Algoritmos exactos

03MAIR – Algoritmos de Optimización

Tipos de Algoritmos



- Atendiendo a **la forma** en la que calculan la solución
 - Algoritmos **deterministas**: Siempre la misma solución para los mismos datos de entrada (instancia).
 - Algoritmos **probabilistas**: Pueden generar soluciones diferentes para los mismos datos de entrada (instancia).
- Atendiendo al **tipo de solución** :
 - Algoritmos **exactos**: Proporcionan la solución óptima.
 - Algoritmos **aproximados**: Proporcionan buenas soluciones con un grado de aproximación al óptimo
 - Algoritmos **heurísticos**: Proporcionan buenas soluciones pero no podemos determinar si es la óptima.

Desarrollo e implementación de algoritmos



● Modelar

- Determinar los supuestos del problema y los datos de inicio.
- Determinar la estructura de datos que represente la información del problema
- No siempre es posible ajustarse al 100% de la realidad. Será necesario evaluar si con el modelo elegido podemos llegar a soluciones deseadas.

● Diseñar

- Aplicar técnicas conocidas para la resolución de problemas.

● Analizar

- Determinar las cantidades de operaciones que realizará el diseño realizado.

Algoritmos de ordenación y Diseño de Algoritmos

1. Algoritmos de ordenación
2. Algoritmos voraces.(elegir en cada etapa la decisión óptima)
3. Búsqueda en Grafos I (Prim y Kruskal)
4. Técnica de Vuelta atrás(exploración exhaustiva por etapas, Ej: N-reinas)
5. Técnica de Divide y Vencerás
6. Técnica de Programación Dinámica (sub-problemas, Bellman)
7. Búsqueda en Grafos II (BFS, A*, DFS, ramificación y poda)

Problemas Tipo

1. Programación lineal (Simplex)
2. Programación lineal entera (ojo con el redondeo de soluciones!)
3. Problema del agente viajero(TSP)
4. Problema de la mochila (enrutamiento, dieta, asignación, ...)
5. Otros problemas,...

VC4 – Repaso de Complejidad

03MAIR – Algoritmos de Optimización

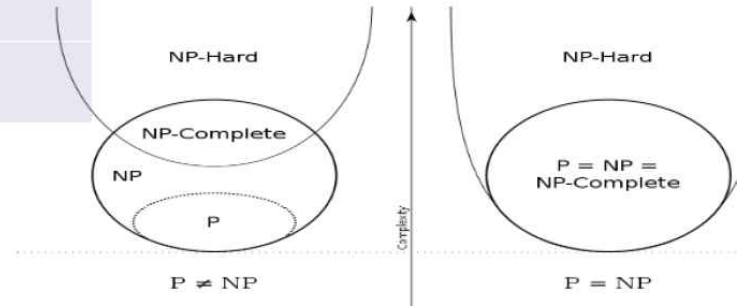
Introducción(Simplificada) a la teoría de la complejidad

- ¿P=NP? O, ¿Qué problemas se pueden resolver fácilmente con computación?

Tipo de problema	Verificable en tiempo P	Solución en tiempo P	Dificultad
P	SI	SI	-
NP	SI	Algunos	
NP-Completos	SI	Desconocido	
NP-Duros	Algunos	Desconocido	+

P= tiempo polinomial

- Problema del millón de dolares



Introducción(Simplificada) a la teoría de la complejidad

- Ejemplos:

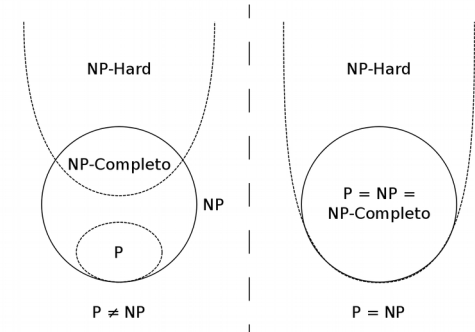
- NP-Completo : Satisfacibilidad booleana(SAT)(1971-Stephen Cook).

- * Todos los NP-Completo se “*reducen*” a SAT

- https://es.wikipedia.org/wiki/Anexo:Problemas_NP-completos

- NP-Duro

- * Problema del Agente Viajero(TSP)



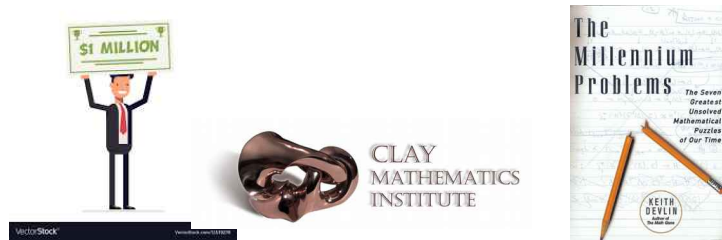
Introducción(Simplificada) a la teoría de la complejidad

- ¿P=NP?

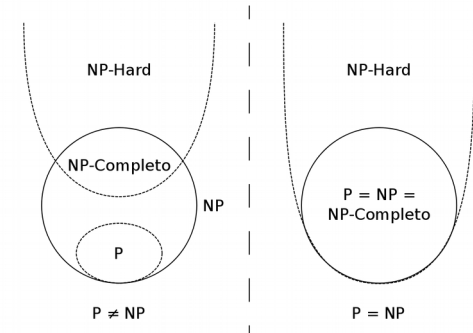
Resolverlo supondría encontrar “atajos” para resolver problemas NP a partir de problemas P (reducción de un problema a otro)

Todo indica a que P no es igual a NP por lo que “casi” asumimos que no podemos abordar estos problemas computacionalmente(*) (con algoritmos exactos!)

Es muy probable que solo podamos resolver estos problemas por métodos aproximados.



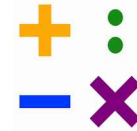
<http://www.claymath.org/millennium-problems/p-vs-np-problem>



Análisis de Algoritmos

- Contar operaciones elementales

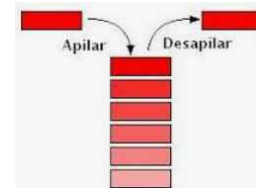
- Suma, resta, multiplicación y división
- Asignaciones
- Condiciones , llamadas a funciones externas y retornos
- Accesos a estructuras de datos
- Ojo con los bucles anidados! 2 bucles $\Rightarrow n^2$



a+=b	a = a + b
a-=b	a = a - b
a*=b	a = a * b
a/=b	a = a / b
a%=b	a = a % b

- Orden de complejidad

- ✓ Decimos que un algoritmo tiene orden de complejidad $O(f)$ si su tiempo de ejecución en operaciones elementales está acotado, salvo por una constante, por f para todos los tamaños de entrada a partir de un cierto n_0



Análisis de Algoritmos

- Contar operaciones elementales. Sin recursividad. Ejemplo 1

```
#Fuerza Bruta

def distancia_fuerza_bruta(L):
    mejor_distancia = 100000e10

    A,B = (),()

    for i in range(len(L)):
        for j in range(i+1, len(L)):
            D = distancia(L[i],L[j])
            if D < mejor_distancia:
                A,B=L[i],L[j]
                mejor_distancia = D

    return [A,B]

distancia_fuerza_bruta(LISTA_2D)
```

Contar cuantas realiza la función distancia: 2
(en 1-dimensión)

Total Operaciones

Complejidad

$$7 \cdot n \cdot (n-1) \sim n^2$$

$$\text{Total: } 7 \cdot n^2 + 4$$



$$O(n^2)$$

Análisis de Algoritmos

- Ecuaciones en recurrencia. Ejemplo 1(*)

```
#Sucesión de Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la sucesión de Fibonacci
def Fibonacci(N:int):
    if N < 2:
        return 1
    else:
        return Fibonacci(N-1)+Fibonacci(N-2)

Fibonacci(5)
```

8

$$T(n) = T(n-1) + T(n-2)$$

$$T(n) - T(n-1) - T(n-2) = 0$$

Ecuación característica:

$$x^2 - x - 1 = 0$$

$$x_1 = \frac{1 + \sqrt{5}}{2} \quad x_2 = \frac{1 - \sqrt{5}}{2}$$

La solución es

$$F(n) = A_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + A_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

Hallar A_1 y A_2 dependiendo de los términos iniciales

(*) El termino n -simo depende de los términos anteriores

Análisis de Algoritmos

- Ecuaciones en recurrencia. Con recursividad. Ejemplo 3(*)

```
#quick_sort
A = [9187, 244, 4054, 9222, 8373, 4993, 5265, 5470, 4519, 7182, 2035, 3506, 4337, 7580, 2554, 2824, 8357, 4447, 7379]

def quick_sort(A):
    if len(A) == 1:
        return A
    if len(A) == 2:
        return [min(A), max(A)]

    IZQ=[]
    DER=[]

    #pivot = (A[0]+ A[1] + A[2])/3
    pivot = sum(A)/len(A)

    #Falla en algunos casos, cuando hay valores repetidos y coinciden al principio de alguna lista
    #Buena opción. La mejor opción sería la mediana pero esto pasa por ordenar la lista
    # que es precisamente lo que queremos hacer.

    for i in A:
        if i <= pivot :
            IZQ.append(i)
        else:
            DER.append(i)

    return quick_sort(IZQ) + quick_sort(DER)

@calcular_tiempo
def QS(A):
    return quick_sort(A)

print(QS(A))
```

Ecuaciones

$$T_n = \begin{cases} c \cdot n^k, 1 \leq n < b \\ a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k, n \geq b \end{cases}$$

Soluciones

$$T(n) = O(n^k) \text{ si } a < b^k$$

$$T(n) = O(n^k \cdot \log(n)) \text{ si } a = b^k$$

$$T(n) = O(n^{\log_b(a)}) \text{ si } a > b^k$$

$$T(n) = 2 \cdot T(n/2) + n$$

$$a=2$$

$$b=2$$

$$c=1$$

$$k=1$$

En el mejor caso!

Análisis de la complejidad temporal

- Ordenes de complejidad (de menor a mayor)

$O(1)$	Orden constante
$O(\log n)$	Orden logarítmico
$O(n)$	Orden lineal
$O(n \log n)$	Orden cuasi-lineal
$O(n^2)$	Orden cuadrático
$O(n^3)$	Orden cúbico
$O(n^k)$	Orden polinómico
$O(2^n)$	Orden exponencial
$O(n!)$	Orden factorial



Tabla 1

Comparación de los órdenes de complejidad según el tamaño del problema

n	$\log n$	n^2	2^{2^n}	$n!$
2	1	4	4	2
8	3	64	256	40.320
32	5	1.024	$4,3 \times 10^9$	$2,6 \times 10^{35}$
100	6	10^4	$1,2 \times 10^{27}$	$9,3 \times 10^{177}$

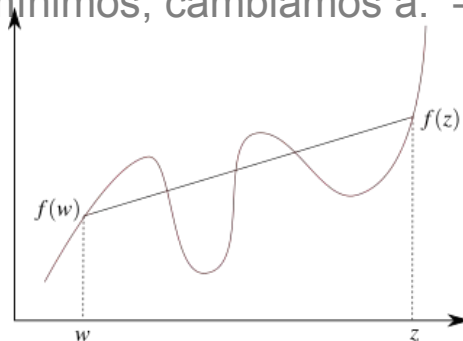
VC4 – Descenso del Gradiente

03MAIR – Algoritmos de Optimización

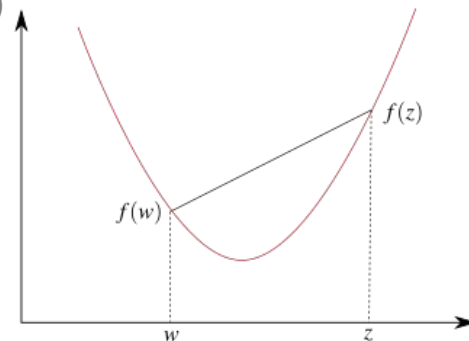
Descenso del Gradiente – AGD

importante

- Es un algoritmo **iterativo** de optimización para encontrar valores mínimos para funciones multi-variables **convexas** y **diferenciables** (y por tanto continuas).
- Convexidad
 - Permitir asegurar que la solución que encontremos es global frente a que solo sea local
 - ¿El segmento que une dos puntos está siempre por encima de la función?
 - Si es cóncava y buscamos mínimos, cambiamos a: $-f(x)$



a) Función no convexa



b) Función convexa

Descenso del Gradiente – AGD

- **Diferenciable:** derivable en n variables(dimensiones)
- **Gradiente:** Es un vector(diferente para cada punto = campo vectorial) cuyas coordenadas son las derivadas parciales:

$$\nabla f(\mathbf{r}) = \left(\frac{\partial f(\mathbf{r})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{r})}{\partial x_n} \right)$$

Ejemplo: Dada la función $f(\mathbf{x}) = x^2 + 2x + y^2 + y^3 + xy$
el gradiente será:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (2x + y + 2, 2y + 3y^2 + x)$$

En el punto (x,y)

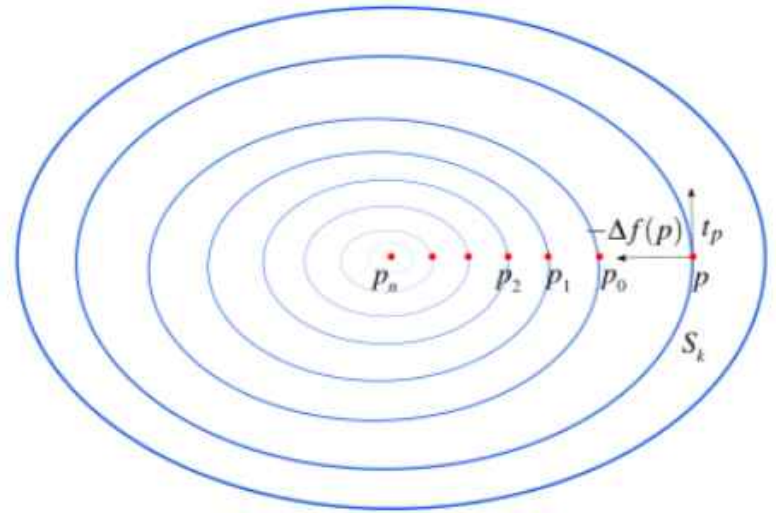
Descenso del Gradiente – AGD

El procedimiento parte de un punto **p** como **solución aproximada(inicial)** y da pasos en el sentido opuesto al gradiente de la función en dicho punto.

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \alpha_t \cdot \Delta \mathbf{f}(\mathbf{p}_t)$$

La elección de α_t estará condicionada para que :

- \mathbf{p}_{t+1} sea solución factible
- mejore el valor de f respecto a \mathbf{p}_t : $f(\mathbf{p}_{t+1}) \leq f(\mathbf{p}_t)$

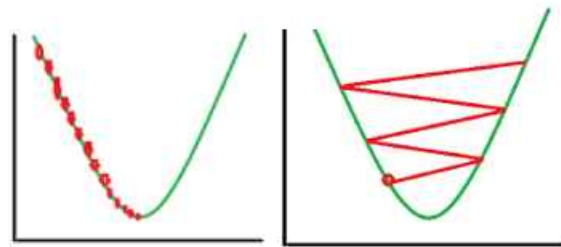


Descenso del Gradiente – AGD

La elección del parámetro α_t , llamado **tasa de aprendizaje (learning rate)**, es importante para hacer efectivo el proceso de acercamiento a la solución óptima.

- Un valor demasiado pequeño puede provocar exceso de iteraciones(Figura 1)
- Un valor demasiado alto puede provocar que el proceso no se acerca lo suficiente(Figura 2)

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \alpha_t \cdot \Delta \mathbf{f}(\mathbf{p}_t)$$



learning rate demasiado pequeño

(Figura 1)

learning rate demasiado grande

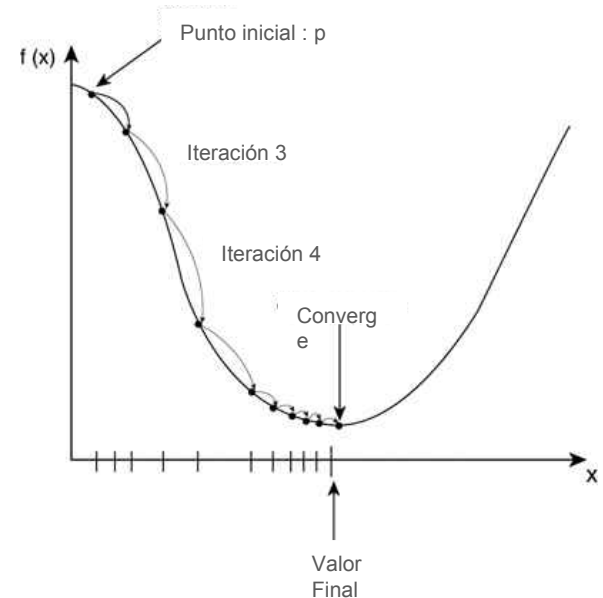
(Figura 2)

Descenso del Gradiente – AGD

En general es buena estrategia ir reduciendo el valor de α_t dinámicamente a medida que nos aproximamos a la solución

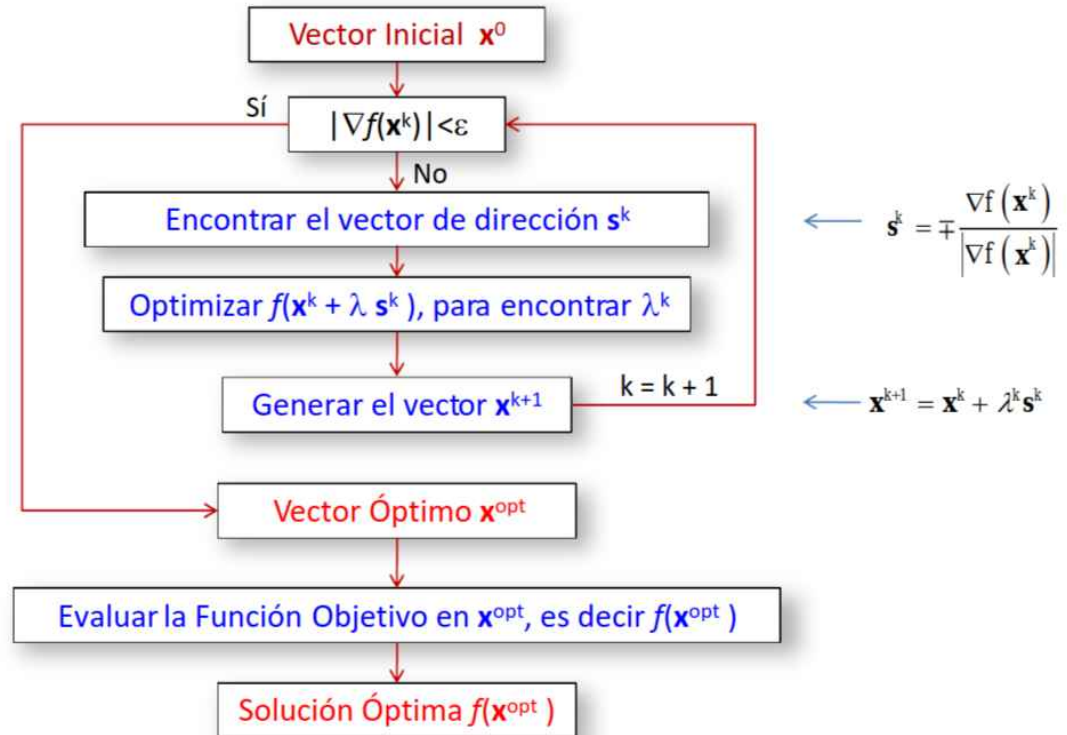
- ¿Como sabemos que nos acercamos?:
 - la magnitud del gradiente
 - cantidad de iteraciones que hemos realizado

$$\mathbf{p}_{t+1} = \mathbf{p}_t - \alpha_t \cdot \Delta \mathbf{f}(\mathbf{p}_t)$$



Descenso del Gradiente – AGD

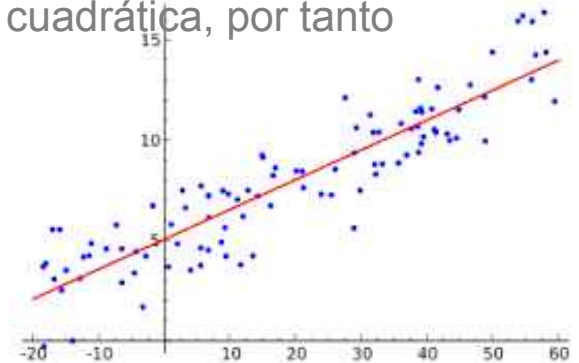
- Diagrama de resolución



Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

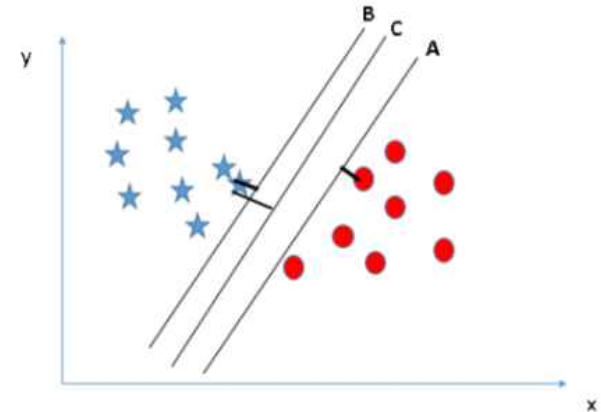
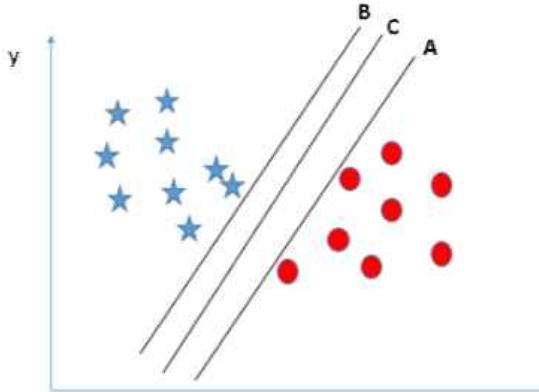
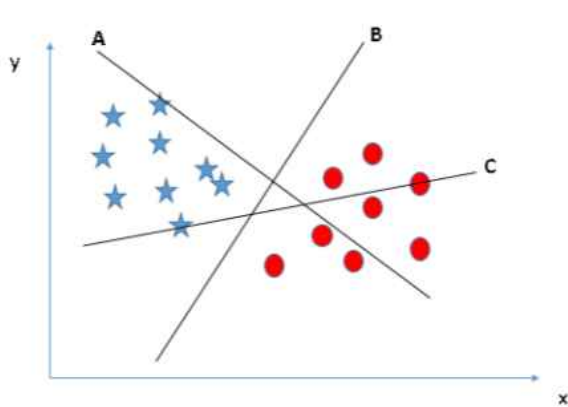
- ✓ En problemas de clasificación tratamos de clasificar los puntos en sus categorías correspondientes de tal manera que se minimice el error.
- ✓ Para la medida de este error solemos utilizar el error cuadrático medio(regresión lineal) pero es posible usar otras medidas(por ejemplo: distancia Euclidea, Manhattan, ...)
- ✓ La función que acumula los errores para los valores conocidos la llamamos función de coste. En el caso de la regresión línea es cuadrática, por tanto diferenciable y podemos obtener el gradiente.



Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

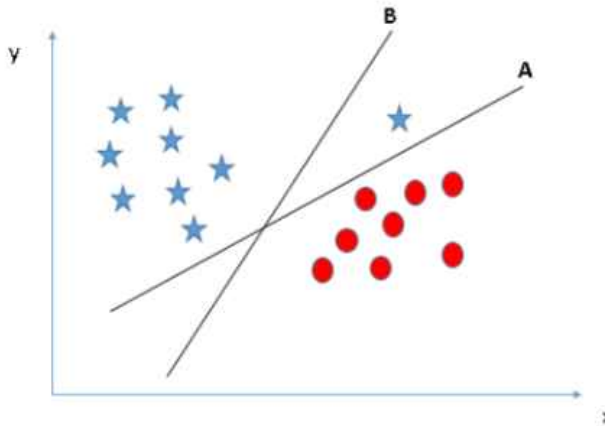


*Seleccionar el que esté
a mayor distancia de ambos conjuntos*

Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?



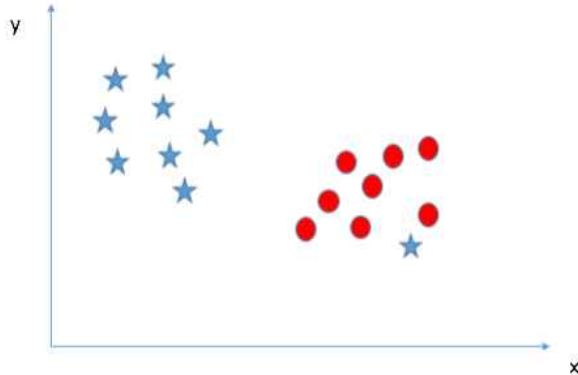
1º . Seleccionar el que mejor clasifique

2º . Seleccionar el que está a mas distancia

Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

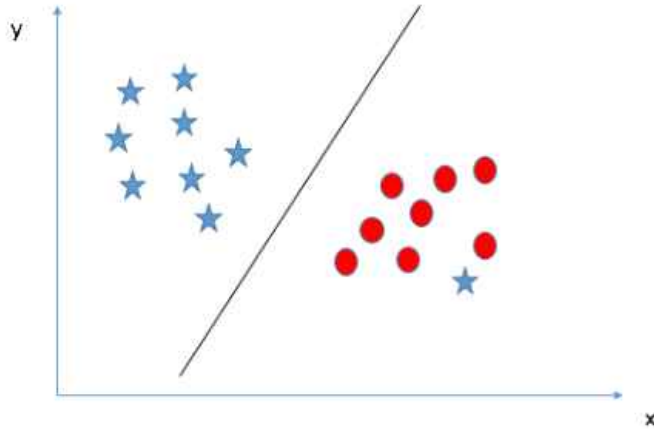


- *No siempre es posible clasificar 100%*
- *¿valores atípicos?*

Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

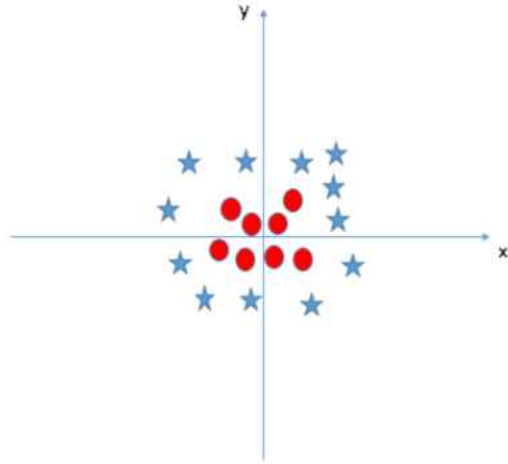


- *Ignorar valores atípicos*

Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

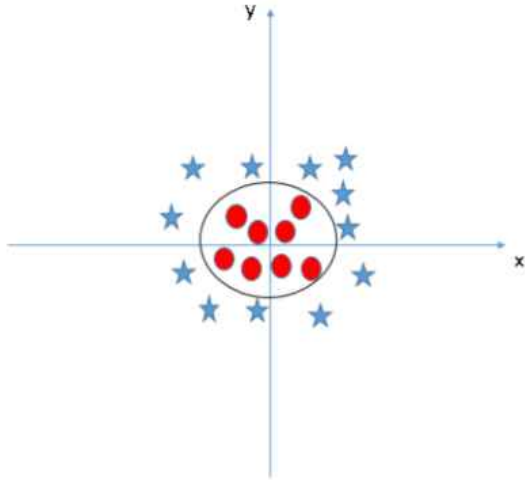


- *No parecen valores atípicos*
- *Parece que si hay una clasificación*
- *Es imposible con hiperplanos lineales*

Descenso del Gradiente – AGD

- Aprendizaje supervisado y la regresión lineal

Podemos extender el procedimiento a funciones no lineales!



Una función como $f(x,y) = x^2 + y^2$ lo resuelve

Descenso del Gradiente – AGD

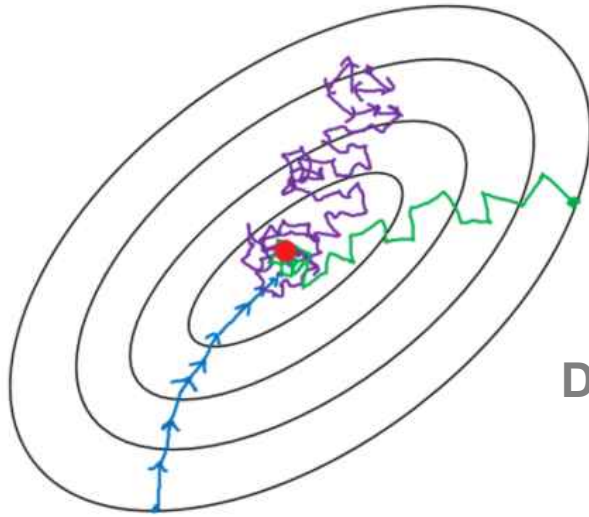
- Otra opción para función de coste(no cuadrática)
 - Entropía cruzada(cross-entropy)
 - Para problemas con soluciones binarias
 - **Definición:** Número de bits diferentes. Mide como de diferentes son dos elementos.
 - Usada en algoritmos de clasificación de imágenes

Descenso del Gradiente – AGD

- **Dependiendo del Volumen de Datos**
 - ✓ Descenso del gradiente por lotes (**batch gradient descent**)
Calcula la desviación para todos los puntos en cada iteración!!!
 - ✓ Descenso del gradiente estocástico(**stochastic gradient descent**)
Calcula la desviación para un punto en cada iteración!!!
 - ✓ Descenso del gradiente por lotes reducido(**mini-batch gradient descent**)
Mezcla de ambos conceptos

Descenso del Gradiente – AGD

- Dependiendo del Volumen de Datos

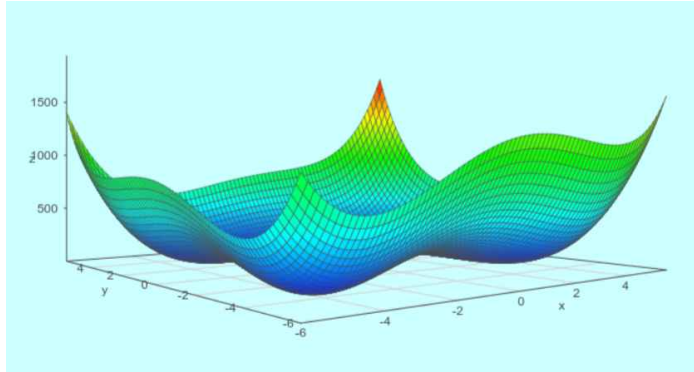


- Batch gradient descent (directo pero muy lento)
- Stochastic gradient descent(rápido pero muy disperso)
- Mini-batch gradient descent(equilibrio)

Decisión: Elección de mini-batch size

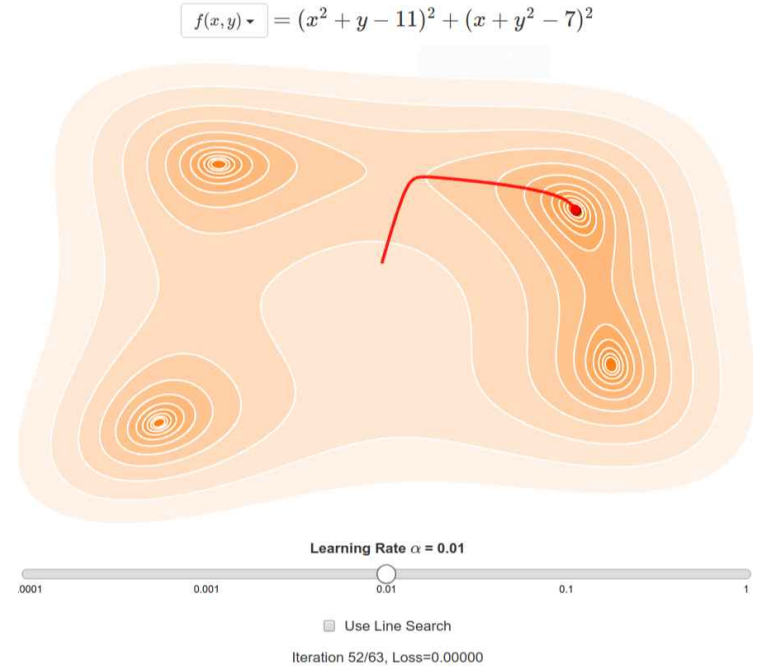
Descenso del Gradiente – AGD

Visualización



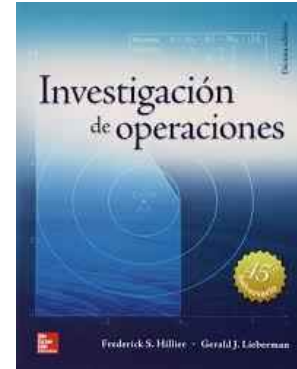
$$(x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

<http://al-roomi.org/3DPlot/index.html>



Ampliación de conocimientos

- ¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial | DotCSV
https://www.youtube.com/watch?v=A6FiCDoz8_4
- Hillier, F. S., y Lieberman, G. J. Investigación de Operaciones. Ciudad de México



Algoritmos paralelos

- **Definición:** Algoritmo que pueden hacer uso de computación en paralelo para ser **ejecutado por partes** y unirlos para dar el resultado final.
- No siempre es posible plantear la resolución de un problema por partes, pues deben ser abordadas **secuencialmente**.
- Toman cada vez más protagonismos dado que la evolución en la mejora de la velocidad de los procesadores empieza a estar “acotada”.
- En general son mas complicados por:
 - Control central
 - Compartir datos (en algunos casos)

VC4 – Problemas del seminario

03MAIR – Algoritmos de optimización

Problema 1. Organizar sesiones de doblaje

- Se precisa coordinar el doblaje de una película. Los actores del doblaje deben coincidir en las tomas en las que sus personajes aparecen juntos en las diferentes tomas. Los actores de doblaje cobran todos la misma cantidad por cada día que deben desplazarse hasta el estudio de grabación independientemente del número de tomas que se graben. No es posible grabar más de 6 tomas por día. El objetivo es planificar las sesiones por día de manera que se gaste por los servicios de los actores de doblaje sea el menor posible. Los datos son:

Número de actores: 10

Número de tomas : 30

Actores/Tomas : <https://bit.ly/36D8luK>

1 indica que el actor participa en la toma
0 en caso contrario

Toma	Actor									
	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	0	0	0	0	0
2	0	0	1	1	1	0	0	0	0	0
3	0	1	0	0	1	0	1	0	0	0
4	1	1	0	0	0	0	1	1	0	0
5	0	1	0	1	0	0	0	1	0	0
6	1	1	0	1	1	0	0	0	0	0
7	1	1	0	1	1	0	0	0	0	0
8	1	1	0	0	0	1	0	0	0	0
9	1	1	0	1	0	0	0	0	0	0
10	1	1	0	0	0	1	0	0	1	0
11	1	1	1	0	1	0	0	1	0	0
12	1	1	1	1	0	1	0	0	0	0
13	1	0	0	1	1	0	0	0	0	0
14	1	0	1	0	0	1	0	0	0	0
15	1	1	0	0	0	0	1	0	0	0
16	0	0	0	1	0	0	0	0	0	1
17	1	0	1	0	0	0	0	0	0	0
18	0	0	1	0	0	1	0	0	0	0
19	1	0	1	0	0	0	0	0	0	0
20	1	0	1	1	1	0	0	0	0	0
21	0	0	0	0	0	1	0	1	0	0
22	1	1	1	1	0	0	0	0	0	0
23	1	0	1	0	0	0	0	0	0	0
24	0	0	1	0	0	1	0	0	0	0
25	1	1	0	1	0	0	0	0	0	1
26	1	0	1	0	1	0	0	0	1	0
27	0	0	0	1	1	0	0	0	0	0
28	1	0	0	1	0	0	0	0	0	0
29	1	0	0	0	1	1	0	0	0	0
30	1	0	0	1	0	0	0	0	0	0



Problema 2. Organizar los horarios de partidos de La Liga

- Desde la La Liga de fútbol profesional se pretende organizar los horarios de los partidos de liga de cada jornada. Se conocen algunos datos que nos deben llevar a diseñar un algoritmo que realice la asignación de los partidos a los horarios de forma que **maximice la audiencia**.
- Los horarios disponibles se conocen a priori y son los siguientes:

Viernes	20
Sábado	12,16,18,20
Domingo	12,16,18,20
Lunes	20

Problema 2. Organizar los horarios de partidos de La Liga

- En primer lugar se clasifican los equipos en tres categorías según el numero de seguidores(que tiene relación directa con la audiencia). Hay **3 equipos** en la **categoría A**, **11 equipos** de **categoría B** y **6 equipos** de **categoría C**.
- Se conoce estadísticamente la audiencia que genera cada partido según los equipos que se enfrentan y en horario de sábado a las 20h (el mejor en todos los casos)

	Categoría A	Categoría B	Categoría C
Categoría A	2 Millones	1,3 Millones	1 Millones
Categoría B		0.9 Millones	0.75 Millones
Categoría C			0.47 Millones

Problema 2. Organizar los horarios de partidos de La Liga

- Si el horario del partido no se realiza a las 20 horas del sábado se sabe que se reduce según los coeficientes de la siguiente tabla
- Debemos asignar obligatoriamente siempre un partido el viernes y un partido el lunes

	Viernes	Sábado	Domingo	Lunes
12h	-	0.55	0.45	-
16h	-	0.7	0.75	-
18h	-	0.8	0.5	-
20h	0.4	1	1	0.4

Problema 2. Organizar los horarios de partidos de La Liga

- Es posible la coincidencia de horarios pero en este caso la audiencia de cada partido se verá afectada y se estima que se reduce en porcentaje según la siguiente tabla dependiendo del número de coincidencias:

Coincidencias	-%
0	0%
1	25%
2	45%
3	60%
4	70%
5	75%
6	78%
7	80%
8	80%

Problema 2. Organizar los horarios de partidos de La Liga

Los cálculos asociados a una jornada de ejemplo se realizan según se muestra en la siguiente tabla:

Partido	Categorías	Horario	Base(Mill.)	Ponderación	Base*Ponderación	Corrección Coincidencia
Celta - Real Madrid	B-A	V20	1,3	0,4	0,52	0,52
Valencia - R. Sociedad	B-A	S12	1,3	0,55	0,72	0,72
Mallorca - Eibar	C-C	S16	0,47	0,7	0,33	0,33
Athletic - Barcelona	B-A	S18	1,3	0,8	1,04	1,04
Leganés - Osasuna	C-C	S20	0,47	1	0,47	0,47
Villarreal - Granada	B-C	D16	0,75	0,75	0,56	0,42
Alavés - Levante	B-B	D16	0,9	0,75	0,68	0,51
Espanyol - Sevilla	B-B	D18	0,9	0,85	0,77	0,77
Betis - Valladolid	B-C	D20	0,75	1	0,75	0,75
Atlético - Getafe	B-B	L20	0,9	0,4	0,36	0,36

Total: 5,88

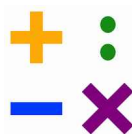
$$=0,56 \times 0,75$$

$$=0,68 \times 0,75$$

Problema 3. Combinar cifras y operaciones

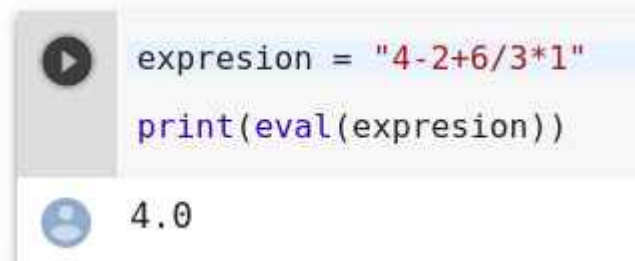
- El problema consiste en **analizar** el siguiente problema y **diseñar** un algoritmo que lo **resuelva**.
- Disponemos de las 9 cifras del 1 al 9 (excluimos el cero) y de los 4 signos básicos de las operaciones fundamentales: suma(+), resta(-), multiplicación(*) y división(/)
- Debemos **combinarlos alternativamente sin repetir ninguno de ellos** para obtener una cantidad dada. Un ejemplo sería para obtener el 4:

$$4+2-6/3*1 = 4$$



Problema 3. Combinar cifras y operaciones

- Debe analizarse el problema para encontrar todos los **valores enteros** posibles planteando las siguientes cuestiones:
 - ¿Qué valor **máximo y mínimo** se pueden obtener según las condiciones del problema?
 - ¿Es posible encontrar **todos los valores enteros posibles** entre dicho mínimo y máximo ?
- Nota: Es posible usar la función de python “**eval**” para evaluar una expresión:



```
expression = "4-2+6/3*1"
print(eval(expression))
```

4.0

Problema del Seminario

- Importante!. Además de resolver el problema será necesario responder algunas preguntas relacionadas con:
 - Complejidad
 - Justificar la estructura de datos elegida
 - Generar y probar con diferentes juegos de datos de entrada

Problemas del Seminario. Evaluación.

Total 13 cuestiones:

6 obligatorias(*) , aseguran 7/10

- 7 opcionales , añaden 2 puntos más: 9/10
 - 1 punto por presentación, descripción
 - lenguaje claro
 - código comentado
 - acompaña ilustraciones si es necesario(imágenes)
- ...

Fecha limite de entrega 1ª convocatoria: 25/01/2021

Fecha limite de entrega 2ª convocatoria: 11/02/2021

VC4 – Repaso de objetivos

03MAIR – Algoritmos de Optimización

Sistema de Evaluación

Evaluación	Evaluación continua 60% (mínimo 5/10)	3 Actividades Guiadas(*) (<u>Evaluables</u>): 10%
		Trabajo del Seminario(*)(<u>Evaluable</u>): 40%
		Participación en Foro(<u>Evaluable</u>): 10%
	Evaluación sumativa 40% (mínimo 5/10)	Examen final(*) Prueba <u>sumativa</u> y final teórico-práctica (preguntas abiertas, preguntas de prueba objetiva, examen truncado, etc.)

*Es requisito indispensable para superar la asignatura aprobar cada apartado.

Actividades Guiadas(*)



- Desarrollar, modelar y analizar algoritmos según diferentes técnicas para resolver el problemas planteados en la asignatura de manera guiada
- Entrega de PDF
- Fecha limite de entrega 1ª convocatoria: 18/01/2021
- Fecha limite de entrega 2ª convocatoria: 11/02/2021

Seminario. Refuerzo del Conocimiento(*)



40%

- Desarrollar, modelar y analizar algoritmos según diferentes técnicas para resolver el problema planteado en la asignatura.
- Resolver un problema real. Entrega de PDF
- Deben identificarse los aspectos teóricos en la entrega
- Fecha limite de entrega 1ª convocatoria: 25/01/2021
- Fecha limite de entrega 2ª convocatoria: 11/02/2021

Foro

10%

viu Universidad Internacional de Valencia

AULAS CALENDARIO SECRETARÍA | SERVICIOS BIBLIOTECA

Raul Reyero Diez
Menu personal 12

20. ALGORITMOS DE OPTIMIZACIÓN

INICIO

INFORMACIÓN GENERAL

- Bienvenida
- Guía didáctica
- Calendario

ACTIVIDAD FORMATIVA

- Recursos y materiales
- Videoconferencias
- Actividades
- Mis calificaciones
- Grupos

COMUNICATE

- Anuncios

discusiones. Más ayuda

Crear foro Buscar 11

Eliminar	Foro	Descripción	Publicaciones totales	Participantes totales
<input type="checkbox"/>	Cuestiones de la asignatura(No evaluable)	En este foro se recogerán las participaciones que no serán evaluables.	0	0
<input type="checkbox"/>	Aportaciones extraordinarias(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al contenido de la asignatura al margen de los temas de debate.	0	0
<input type="checkbox"/>	Foro para Tema 1 de debate semana 15-julio(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al tema de debate de la semana 15-jul	0	0
<input type="checkbox"/>	Foro para Tema 2 de debate semana 22-jul(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al tema de debate de la semana 22-jul	0	0

Eliminar

Examen



Fecha 1ª convocatoria : 18/01/2021 desde 21:00(ventana 48 horas)

Fecha 2ª convocatoria : 11/02/2021 desde 20:00(ventana 48 horas)

- Duración: **1 hora y 30 minutos**
- 10 preguntas: 9 tipo test + 1 práctica de desarrollo

Convocatorias y entregas tardías

- * Si es la **primera convocatoria** de la actividad:

En este caso, la entrega se ignora y se suspende directamente con un 0. El alumno puede presentar el mismo trabajo, si así lo desea, en segunda convocatoria, en cuyo caso se corregirá de forma normal sin penalización.

- * Si es la **segunda convocatoria** de la actividad (pero se ha entregado antes de publicar las actas):

se procederá a corregir la actividad, pero se quedará en 5 sobre 10.

- * Si es la **segunda convocatoria** de la actividad pero se ha entregado tras publicar las actas:
se ignorará la entrega y se considerará la actividad como no presentada.

Resumiendo..., es obligatorio:

- Entregar y aprobar(*) las 3 actividades guiadas. La entrega asegura 8/10
- Entregar y aprobar(*) trabajo del seminario.
- Aprobar(*) el examen : 9 test(8 puntos) + 1 práctica corta(2 puntos)

(*) Debe obtener 5/10 mínimo

Revisar “Materiales Docentes”. Guía interactiva



Sobre las actividades guiadas



Ampliación de conocimientos y habilidades

■ Bibliografía

-Brassard, G., y Bratley, P. (1997). Fundamentos de algoritmia. ISBN 13: 9788489660007

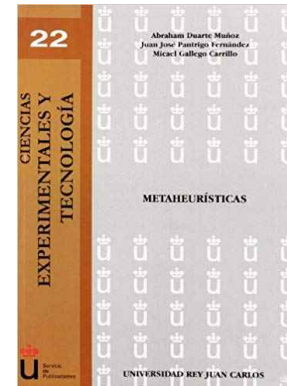
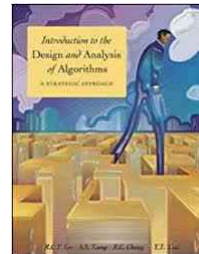
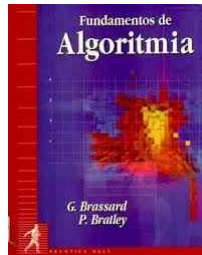
-Guerequeta, R., y Vallecillo, A. (2000). Técnicas de diseño de algoritmos.

<http://www.lcc.uma.es/~av/Libro/indice.html>

-Lee, R. C. T., Tseng, S. S., Chang, R. C., y Tsai, Y. T. (2005). Introducción al diseño y análisis de algoritmos. ISBN 13: 9789701061244

-**Abraham Duarte,.. Metaheurísticas. ISBN 13: 9788498490169**

• Practicar



Próximo día, prácticas sobre :

- Programación dinámica
- Búsqueda en grafos, ramificación y poda.
- Descenso del gradiente

Gracias

raul.reyero@campusviu.es