

VC4 – Descenso del Gradiente

03MIAR – Algoritmos de Optimización

Agenda

- Aportaciones en el foro.
- Revisión de la Complejidad. Cálculo de operaciones.
- Descenso del Gradiente
- Problemas del Trabajo Práctico

Aportaciones en el foro

Cuándo usar técnica la técnica voraz en el cambio de moneda



Belén Jiménez García

Hace 9 días

Cuándo usar técnica la técnica voraz en el cambio de moneda

La técnica voraz no siempre funciona bien para el problema del cambio de moneda. De hecho va a depender del sistema monetario utilizado. Para los sistemas monetarios como el Euro o el Dólar va a funcionar bien, son los denominados canónicos.

Los sistemas canónicos nos van a garantizar el correcto funcionamiento de la técnica voraz en este problema. Para determinar si un sistema de monedas es canónicos podemos recurrir al algoritmo de complejidad $O(n^3)$ de Pearson presentado en el artículo "A Polynomial-Time Algorithm for the Change-Making Problem" [1]. Este algoritmo es lo mejor que tenemos hasta la fecha. Se basa en otros dos artículos. El primero de Chang y Gill [2] en el que muestran que esto se puede resolver en polinomios de tiempo en el tamaño de la moneda más grande y en el número de monedas. El segundo de Kozen and Zaks [3] propone un algoritmo más eficiente y plantea si esto se puede resolver en tiempo polinomial del tamaño del sistema monetario de entrada. En este artículo finalmente lo que se hace es implementar este algoritmo de tiempo polinomial.

Finalmente, quiero sugerir una solución basada en programación dinámica, que garantiza encontrar siempre la solución más óptima.

Google Colaboratory



[1] Pearson, D. "A Polynomial-time Algorithm for the Change-Making Problem" Theor. Comput. Sci. (1994)
[type=pdf](#)

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.3243&rep=rep1&type=pdf>

[2] Chang, S. K., A. Gill, "Algorithmic solution of the change-making problem," J. Assoc. Comput. Mach. 17:1 (January 1970), pp. 113-122.

[3] Kozen, D., S. Zaks, "Optimal bounds for the change-making problem," Theor. Comput. Sci. 123 (1994), pp. 377-388.

<https://www.cs.cornell.edu/~kozen/Papers/change.pdf>

Aportaciones en el foro

Algoritmos de Prim y Kruskal para obtener árboles de recubrimiento mínimo



David Bataller Sendra

Hace 9 días

Algoritmos de Prim y Kruskal para obtener árboles de recubrimiento mínimo

Para los que estéis interesados en los algoritmos de Prim y Kruskal, que no llegamos a ver implementados en clase, he sacado (y ligeramente modificado) sus implementaciones del pdf https://ocw.ehu.es/pluginfile.php/46102/mod_resource/content/1/O3_Algoritmos_Voraces/O3_Algoritmos_Voraces.pdf

El algoritmo de Prim sería el siguiente:

```
def Prim(num_vertices, grafo):  
    # Se inicializa el árbol de recubrimiento mínimo como una lista vacía, y se comienza seleccionando el vértice 0.  
    arbol = []  
    vertices_seleccionados = [0]  
    coste = 0  
  
    while len(vertices_seleccionados) < num_vertices:  
        # Se obtiene la arista de menor coste tal que un vértice ya ha sido seleccionado previamente y el otro no. También se obtiene este último vértice, y el coste de la arista.  
        nueva_arista, nuevo_vertice, nuevo_peso = obtener_siguiete_Prim(vertices_seleccionados, grafo)
```

Aportaciones en el foro

Aplicaciones de la programación entera - problemas de cubrimiento



Karen Oliveros Félez

Hace 5 días

Aplicaciones de la programación entera - problemas de cubrimiento

Una de las aplicaciones de la programación entera que hemos mencionado en clase son los problemas de cubrimiento. Curiosamente mi trabajo de fin de grado de la carrera trataba sobre las distintas modelizaciones y formulaciones de un problema de cubrimiento en específico. Según mi trabajo <https://zaguán.unizar.es/record/77836> (basado en papers):

Los problemas de cubrimiento surgen al decidir dónde ubicar instalaciones que van a proveer servicios a usuarios, que solo podrán ser atendidos por la instalación más cercana si ésta se sitúa a una distancia menor que una dada D .

Así pues, diremos que un individuo está cubierto si está a una distancia menor que D de una instalación.

En este trabajo se describen distintos tipos de problemas de cubrimiento según cómo formemos la función objetivo y si queremos minimizar o maximizar:

- LSCP: problema de cubrimiento de conjuntos que consiste en minimizar el coste de localización suponiendo que ubicar una instalación en un vértice u otro tiene costes distintos
- MCLP: problema de cubrimiento maximal, cuyo objetivo es identificar la ubicación de un número específico de instalaciones de manera que se maximice la demanda cubierta
- LSCP-Implicit: problema de cubrimiento de conjuntos implícito, que es como el LSCP pero tiene en cuenta si la instalación que cubre un conjunto de nodos de demanda j está disponible en el momento en el que es requerido por un usuario.
- CLSCP: problema de cubrimiento de conjuntos con fraccionamiento, en el que se tienen en cuenta restricciones de capacidad.
- PSCP: problema de cubrimiento de conjuntos probabilístico, en el que algunos de los parámetros son variables (como la fracción de tiempo en el que la instalación está ocupada dando servicio)

VC4 – Complejidad

03MIAR – Algoritmos de Optimización

Agenda

- Revisión de la Complejidad. Cálculo de operaciones.
- Descenso del Gradiente
- Problemas del Trabajo Práctico

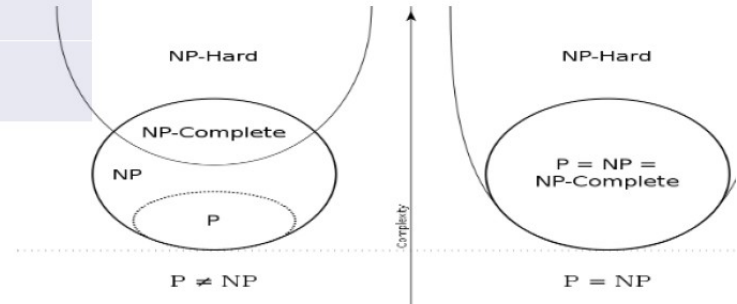
Introducción(Simplificada) a la teoría de la complejidad

- ¿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

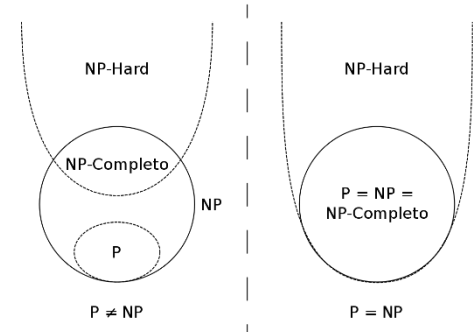
- Problema del millón de dólares



Introducción(Simplificada) a la teoría de la complejidad

- Ejemplos:

- NP-Completo : Satisfacibilidad booleana(SAT)(1971-Stephen Cook).
 - * Todos los NP-Completo se “*reducen*” a SAThttps://es.wikipedia.org/wiki/Anexo:Problemas_NP-completos
- NP-Duro
 - * Problema del Agente Viajero(TSP)



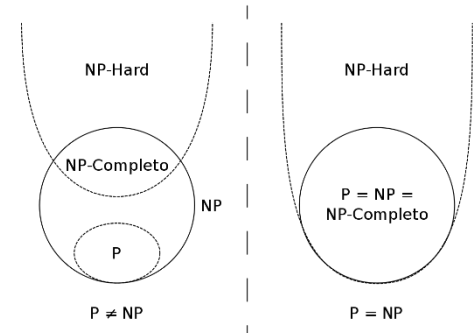
Introducción(Simplificada) a la teoría de la complejidad

- ¿P=NP?

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

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

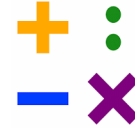
Es muy probable que solo podamos resolver estos problemas por métodos aproximados o usando heurísticas.



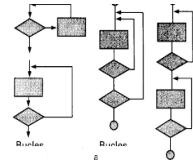
Análisis de Algoritmos

- 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 $\Rightarrow n^2$

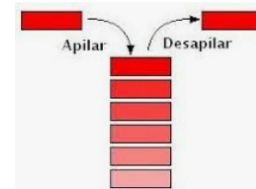


$a+=b$	$a = a + b$
$a-=b$	$a = a - b$
$a*=b$	$a = a * b$
$a/=b$	$a = a / b$
$a\%=b$	$a = a \% b$



- 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. Sin recursividad

Contar operaciones elementales.

```
#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

El termino n-simo depende de los términos anteriores

```
#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

$$T(n) = A \cdot \left(\frac{1 + \sqrt{5}}{2} \right)^n + B \cdot \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

A y B dependen de los términos iniciales

Análisis de Algoritmos. Ecuaciones en recurrencia

El termino n-simo depende de una división de los términos anteriores

```
#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=[]

    #pivote = (A[0]+ A[1] + A[2])/3    #Falla en algunos casos, cuando hay valores repetidos y coinciden al principio de alguna lista
    pivote = sum(A)/len(A)           #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 <= pivote :
            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!

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

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
$O(n^4)$	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^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

03MIAR – Algoritmos de Optimización

Agenda

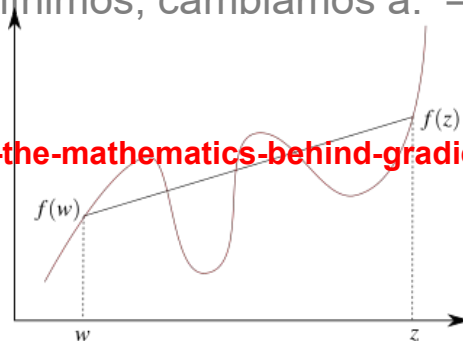
- Revisión de la Complejidad. Cálculo de operaciones.
- **Descenso del Gradiente**
- Problemas del Trabajo Práctico

Descenso del Gradiente – AGD

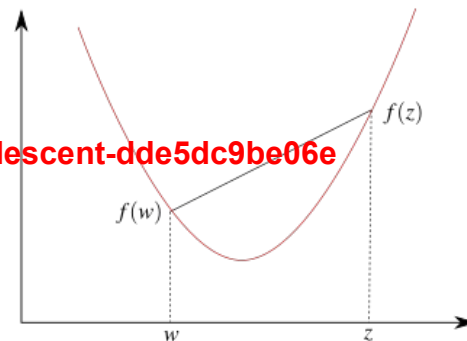
importante

- 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 óptimo 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: $-f(x)$

<https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e>



a) Función no convexa



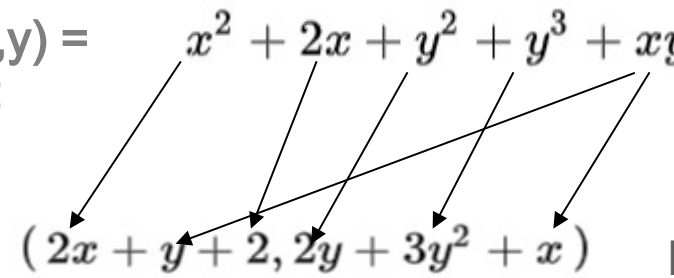
b) Función convexa

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 $f(x,y) =$
el gradiente será:


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

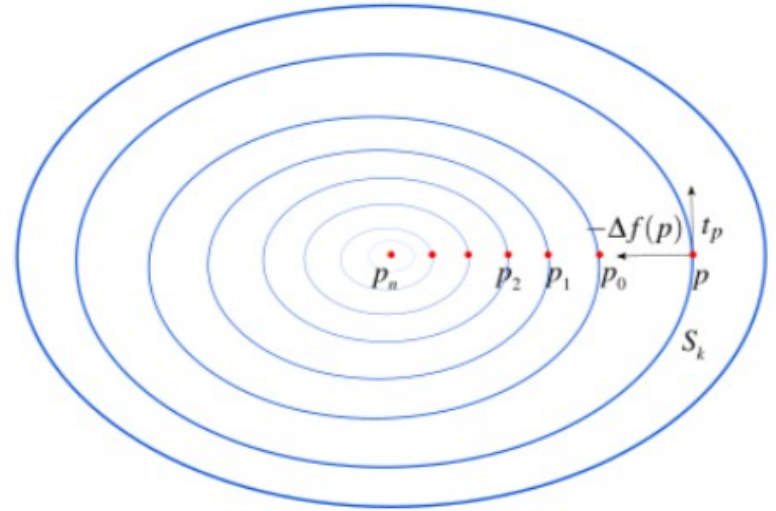
Descenso del Gradiente – AGD

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

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$

La elección de α_t estará condicionada para que :

- p_{t+1} sea solución factible
- mejore el valor de f respecto a p_t : $f(p_{t+1}) \leq f(p_t)$

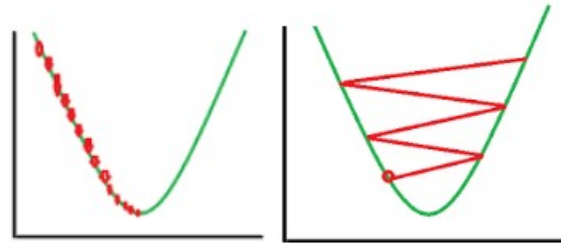


Descenso del Gradiente – AGD

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 acerque lo suficiente(Figura 2)

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$



learning rate demasiado pequeño
(Figura 1)

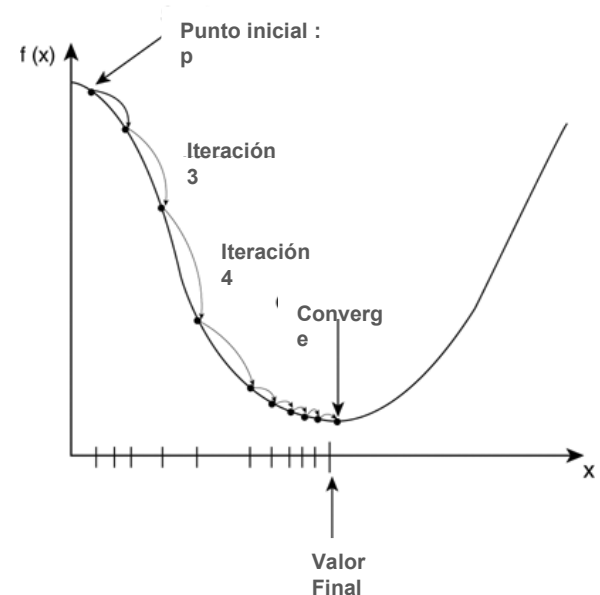
learning rate demasiado grande
(Figura 2)

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

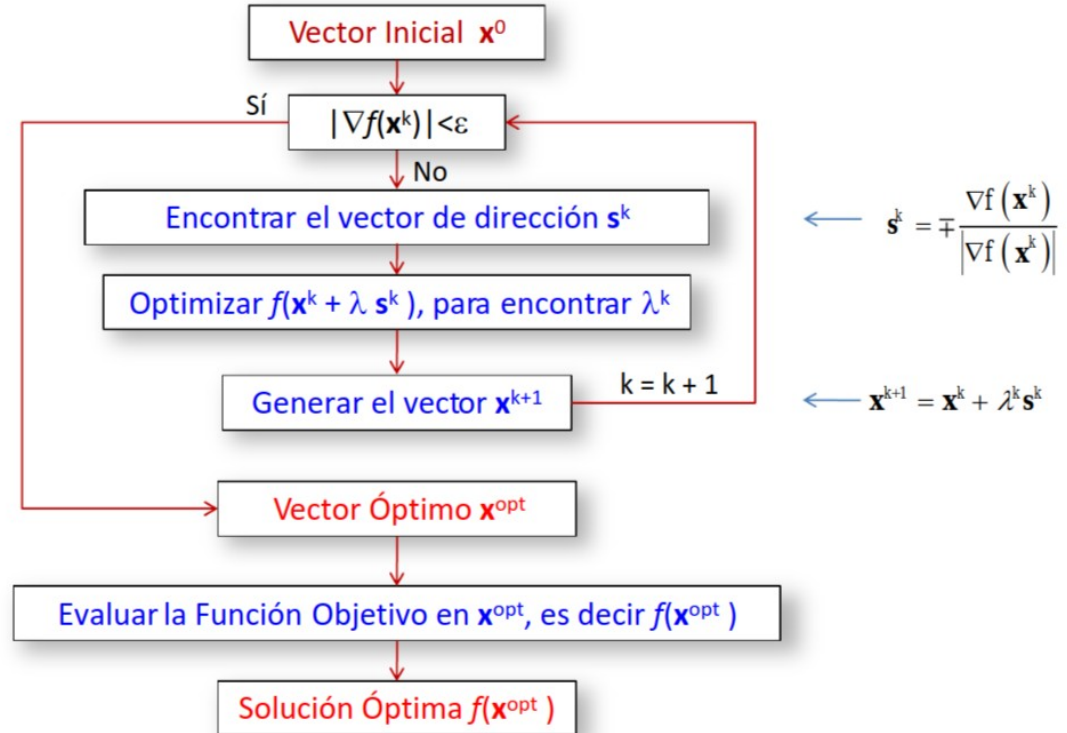
- ¿Como sabemos que nos acercamos?:
 - la magnitud del gradiente
 - cantidad de iteraciones que hemos realizado

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$



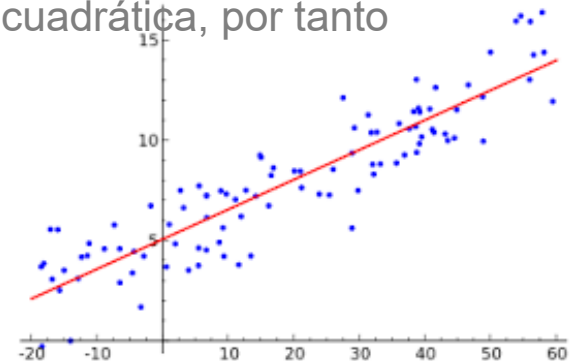
Descenso del Gradiente – AGD

- Diagrama de resolución



Descenso del Gradiente – AGD

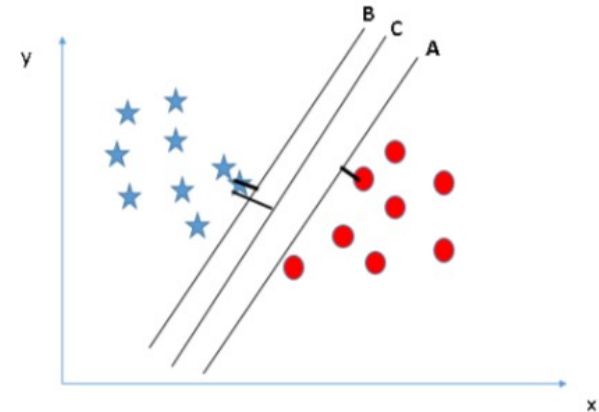
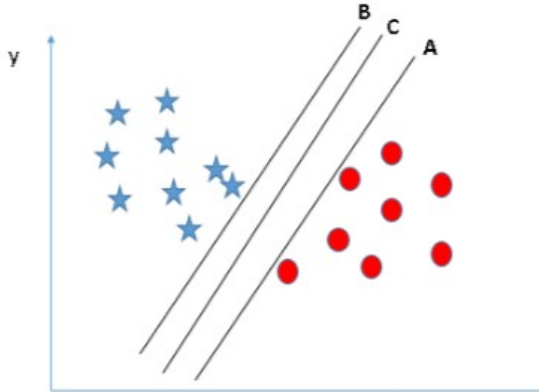
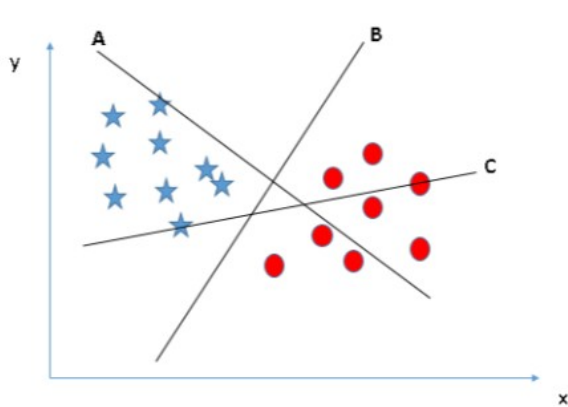
- Aprendizaje supervisado y la regresión lineal
 - ✓ En problemas de clasificación tratamos de clasificar los puntos en sus categorías correspondientes de tal manera que se minimice el error.
 - ✓ Para la medida de este error solemos utilizar el **error cuadrático medio**(regresión lineal) pero es posible 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?

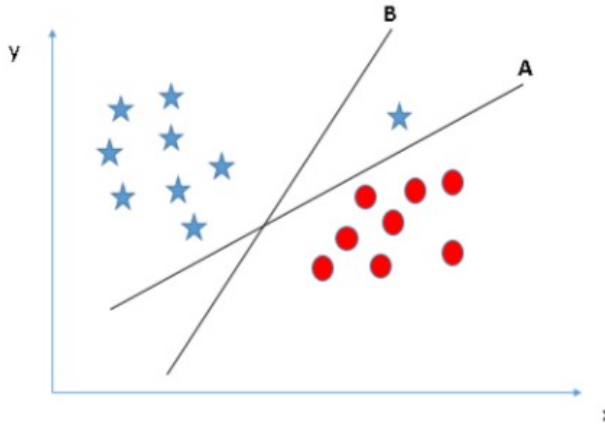


*Seleccionar el que esté
a mayor distancia de ambos conjuntos*

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?

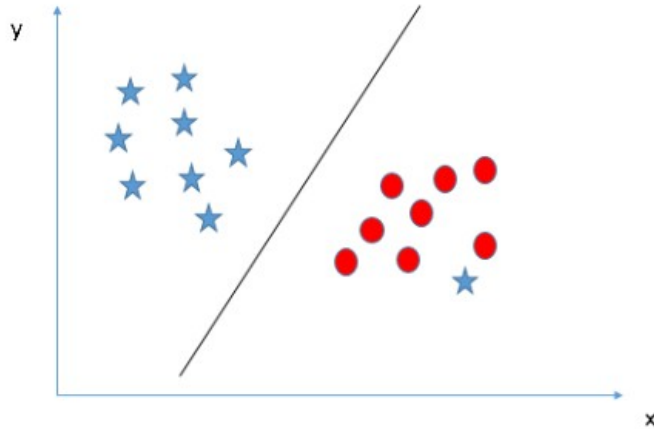


- *No siempre es posible clasificar 100%*
- *¿valores atípicos?*

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?

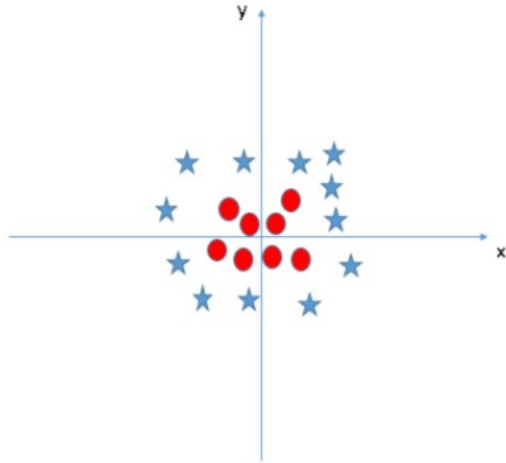


- *Ignorar valores atípicos*

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?

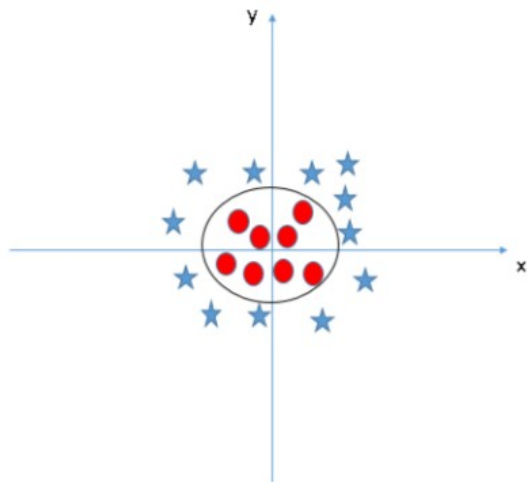


- *No parecen valores atípicos*
- *Parece que si hay una clasificación*
- *Es imposible con hiperplanos lineales*

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

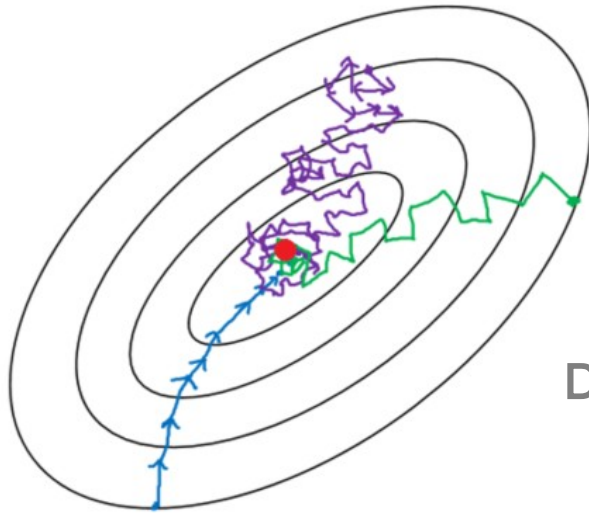
- Otra opción para función de coste(no cuadrática)
- Entropía cruzada(cross-entropy)
 - Para problemas con soluciones binarias
 - **Definición:** Número de bits diferentes. Mide como de diferentes son dos elementos.
 - Usada en algoritmos de clasificación de imágenes

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

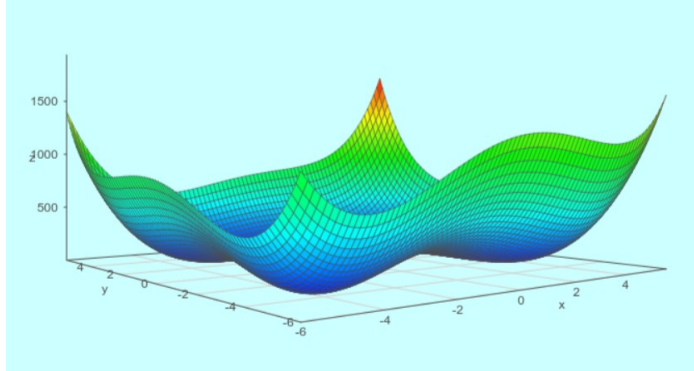


- Batch gradient descent (directo pero muy lento)
- Stochastic gradient descent(rápido pero muy disperso)
- Mini-batch gradient descent(equilibrio)

Decisión: Elección de mini-batch size

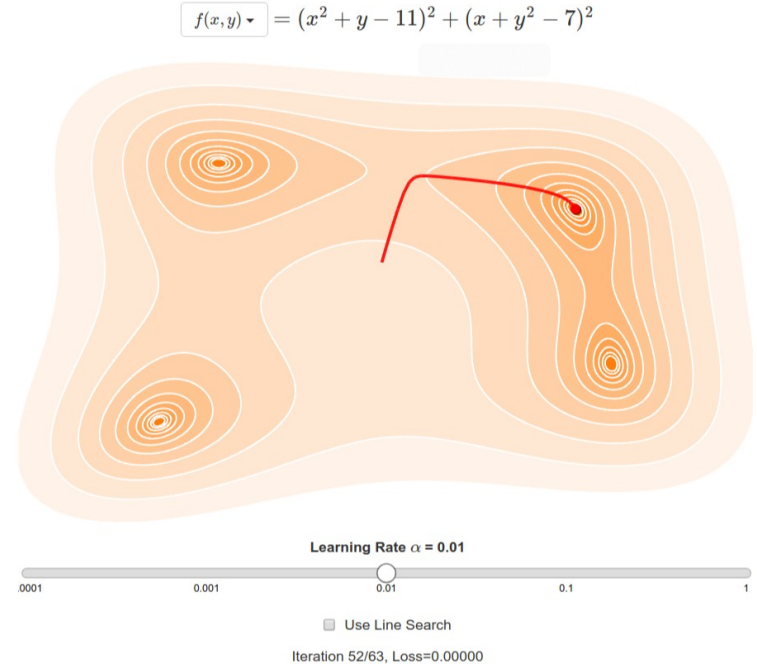
Descenso del Gradiente – AGD

Visualización



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

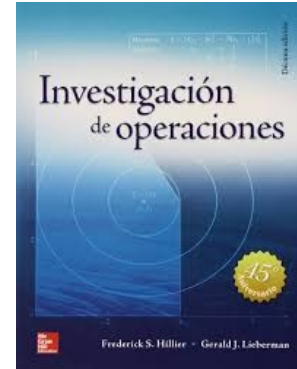
<http://al-roomi.org/3DPlot/index.html>



<https://www.benfrederickson.com/numerical-optimization/>

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



VC4 – Problemas del Trabajo Práctico

03MIAR – Algoritmos de optimización

Agenda

- Revisión de la Complejidad. Cálculo de operaciones.
- Descenso del Gradiente
- Problemas del Trabajo Práctico

Problema 1. Organizar sesiones de doblaje(I)

- 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 el gasto 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(I)

- 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(II)

- 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(III)

- 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	0.85	-
20h	0.4	1	1	0.4

Problema 2. Organizar los horarios de partidos de La Liga(IV)

- 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(IV)

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 Trabajo Práctico

ACTIVIDAD FORMATIVA

Videoconferencias

Recursos y
materiales

Actividades

Mis calificaciones

COMUNICACIÓN

Anuncios

Foro

Desarrollar algoritmos con la técnica de búsqueda aleatoria
Desarrollar algoritmos con la técnica de búsqueda local
Desarrollar algoritmos con la técnica de recocido simulado(simulated annealing)(SA)
Desarrollar algoritmos con la técnica de colonia de hormigas(ACO)
Desarrollo de algoritmos genéticos(AG)



Trabajo Práctico

Desarrollar, modelar y analizar algoritmos según diferentes técnicas para resolver el problema planteado en la asignatura.

Haz una copia de la platilla siguiente de Google Colab para realizar el trabajo:

<https://colab.research.google.com/drive/1NVFHsnmrE-wFLX8y1SC3tKlh2et5FOz8>

Al finalizar, genera el .pdf y adjuntalo como respuesta.

Plantilla: <https://colab.research.google.com/drive/1NVFHsnmrE-wFLX8y1SC3tKlh2et5FOz8>

Problema del Trabajo Práctico

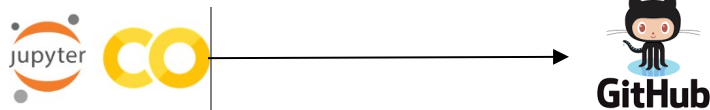
- Importante!. Además de resolver el problema será necesario responder algunas preguntas relacionadas con:
 - Complejidad
 - Justificar la estructura de datos elegida
 - Generar y probar con diferentes juegos de datos de entrada
- Fecha limite de entrega 1ª convocatoria: **17/02/2022**
- Fecha limite de entrega 2ª convocatoria: **11/03/2022**

Problema del Trabajo Práctico. Entregable

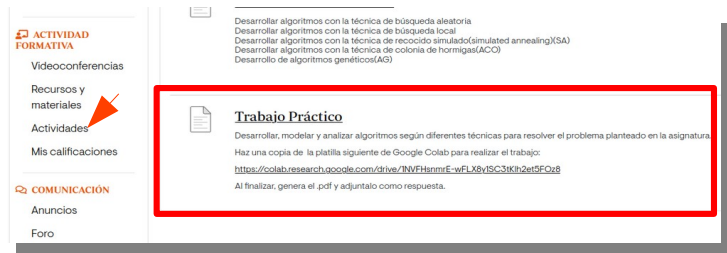


30%

- Generar un Notebook en GitHub (carpeta SEMINARIO)



- Entrega de documento .pdf con en Notebook (como las A. Guiadas)

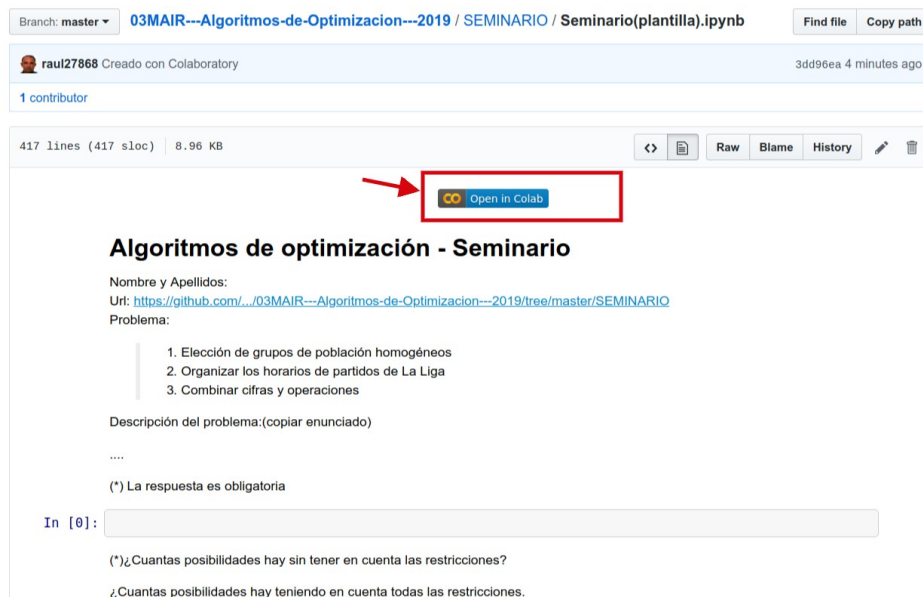


Problema del Trabajo Práctico. Entregable

- Plantilla para el documento

<https://colab.research.google.com/drive/1NVFHsnmrE-wFLX8y1SC3tKlh2et5FOz8>

30%



Branch: master ▾ 03MAIR---Algoritmos-de-Optimizacion---2019 / SEMINARIO / Seminario(plantilla).ipynb Find file Copy path

raul27868 Creado con Colaboratory 3dd96ea 4 minutes ago

1 contributor

417 lines (417 sloc) 8.96 KB

Open in Colab

Algoritmos de optimización - Seminario

Nombre y Apellidos:
Url: <https://github.com/.../03MAIR---Algoritmos-de-Optimizacion---2019/tree/master/SEMINARIO>
Problema:

1. Elección de grupos de población homogéneos
2. Organizar los horarios de partidos de La Liga
3. Combinar cifras y operaciones

Descripción del problema:(copiar enunciado)

....

(*) La respuesta es obligatoria

In [0]:

(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Problema del Trabajo Práctico. Entregable



30%

- Cabecera

Algoritmos de optimización - Seminario

Nombre y Apellidos:

Url: <https://github.com/.../03MAIR---Algoritmos-de-Optimizacion---2019/tree/master/SEMINARIO>

Problema:

- ~~1. Elección de grupos de población homogéneos~~
- ~~2. Organizar los horarios de partidos de La Liga~~
3. Combinar cifras y operaciones

Descripción del problema: (copiar enunciado)

...

Añadir texto del enunciado



(*) La respuesta es obligatoria

[]

Problema del Trabajo Práctico. Entregable

- Pregunta – Respuesta (texto + Python)

30%

(*)¿Cuantas posibilidades hay sin tener en cuenta las restricciones?
¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta

	Texto
[]	Código python

Obligatoria

Problema del Trabajo Práctico. Entregable. Ejemplo



Pregunta – Respuesta (texto + Python)

(*)¿Cuántas posibilidades hay sin tener en cuenta las restricciones?

¿Cuántas posibilidades hay teniendo en cuenta todas las restricciones.

Respuesta:

Para calcular el número posible de soluciones es necesario algo de combinatoria y saber contar.

Suponemos que el número de ciudades es N y que partimos de una ciudad dada. Podemos suponer un procedimiento que vaya construyendo todas las soluciones.

Para el primer viaje disponemos de $N-1$ ciudades ya que debemos eliminar la ciudad de partida como posible candidata. Para la segunda ciudad a visitar disponemos de $N-2$ posibilidades. Por tanto ya tenemos $(N-1) \times (N-2)$ para visitar 2 ciudades.

Si seguimos el razonamiento deducimos que hay $(N-1)!$ (factorial de $N-1$) posibilidades.

Puesto que el camino es circular (comienza y termina en la misma ciudad) debemos tener en cuenta que cada ruta tiene una ruta inversa semejante. La primera se convierte en la última, la segunda en la penúltima y así sucesivamente. Por tanto sin no queremos tener en cuenta esta repetición en total tenemos $(N-1)!/2$

Problema del Trabajo Práctico. Preguntas(1/3)



Pregunta – Respuesta (texto + Python)

- (*)¿Cuántas posibilidades hay sin tener en cuenta las restricciones?
- ¿Cuántas posibilidades hay teniendo en cuenta todas las restricciones.
- (*) ¿Cual es la estructura de datos que mejor se adapta al problema? Argumenta la respuesta
(Es posible que hayas elegido una al principio y veas la necesidad de cambiar, argumenta)
- (*)¿Cual es la función objetivo?
- (*)¿Es un problema de maximización o minimización?

Problema del Trabajo Práctico. Preguntas(2/3)



Pregunta – Respuesta (texto + Python)

- Diseña un algoritmo para resolver el problema por fuerza bruta
- Calcula la complejidad del algoritmo por fuerza bruta
- (*)Diseña un algoritmo que mejore la complejidad del algoritmo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta
- (*)Calcula la complejidad del algoritmo
- Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorio.

Problema del Trabajo Práctico. Preguntas(3/3)



Pregunta – Respuesta (texto + Python)

- Aplica el algoritmo al juego de datos aleatorio generado.
- Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo
- Describe brevemente en unas líneas como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño.

Problema del Trabajo Práctico. 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(imágenes) si es necesario
- ...

Fecha limite de entrega 1ª convocatoria: **17/02/2022**

Fecha limite de entrega 2ª convocatoria: **11/03/2022**



Ampliación de conocimientos y habilidades

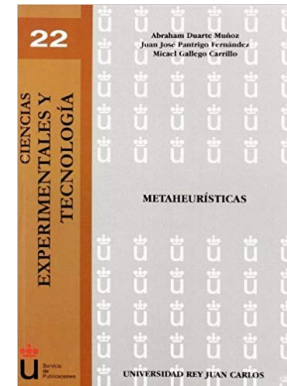
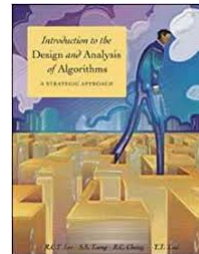
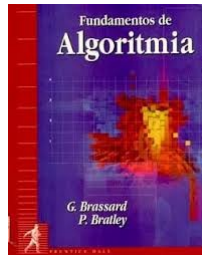
■ Bibliografía

-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



Próximo día, prácticas sobre :

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

Gracias

raul.reyero@campusviu.es