AG2 – Actividad Guiada 2

03MIAR – Algoritmos de Optimización

Abrir el cuaderno de google colab:

https://colab.research.google.com/drive/15LKBrH_aHrbVfPCcAg6QFRkn5gC5-SL9?usp=sharing

viu | Universidad Internacional de Valencia

AG2 – Actividad Guiada 2

Programación Dinámica

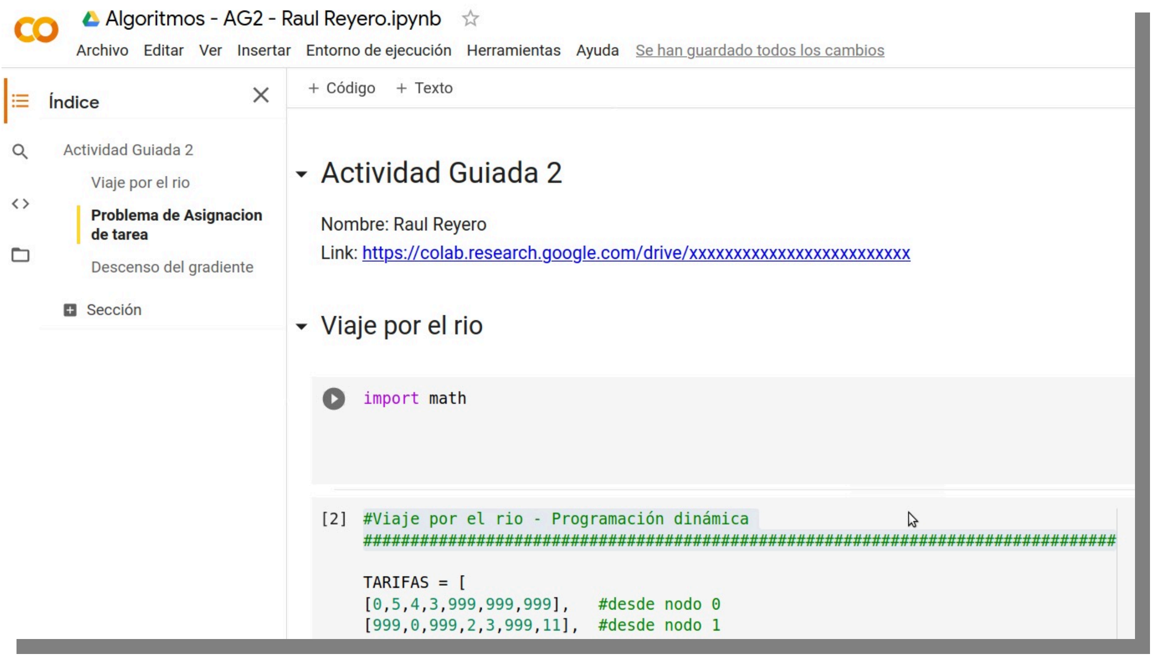
Agenda

- Nuevo tema en el Foro
- Práctica: Programación dinámica(Viaje por el rio).
- Teoría: Ramificación y Poda
- Práctica: Búsqueda en grafos, ramificación y poda(asignación de tareas).
- Practica: Descenso del gradiente.

viu | Universidad Internacional de Valencia

Google Colab

https://colab.research.google.com/drive/15LKBrH_aHrbVfPCcAg6QFRkn5gC5-SL9?usp=sharing



Programación dinámica (I)

importante

- **Definición:** Es posible dividir el problema en subproblemas más pequeños, guardando las soluciones para ser utilizadas más adelante.
- Características que permiten identificar problemas aplicables:
  - ✓ Es posible almacenar soluciones de los subproblemas para ser reutilizadas.
  - ✓ Debe verificar el **principio de optimalidad** de Bellman: "*en una secuencia optima de decisiones, toda sub-secuencia también es óptima*" (*)
  - ✓ La necesidad de guardar la información acerca de las soluciones parciales unido a la recursividad provoca la necesidad de preocuparnos por la complejidad espacial (cuantos recursos de espacio usaremos)

(*) Ojo! No significa que si tenemos soluciones optimas de los sub-problemas las podamos combinar para el optimo del problema principal.

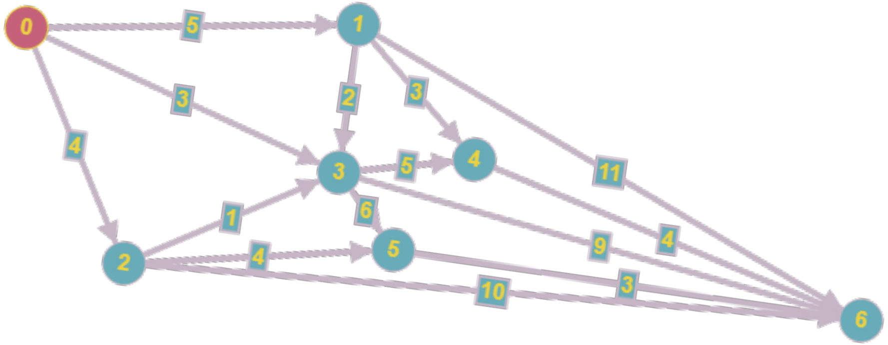
Repaso

Programación dinámica (II)

Problema: Viaje por el rio

- Consideramos una tabla T(i,j) para almacenar todos los precios que nos ofrecen los embarcaderos
- Si no es posible ir desde i a j daremos un valor alto para garantizar que ese trayecto no se va a elegir en la ruta óptima(modelado habitual para restricciones)
- Establecer una tabla intermedia(P(i,j)) para guardar soluciones óptimas parciales para ir desde i a j.

$$P(i,j) = \min \{T(i,j) \text{ , } P(i,k)+T(k,j) \text{ para todo } i<k<=j \}$$



Universidad Internacional de Valencia

Programación dinámica (III)

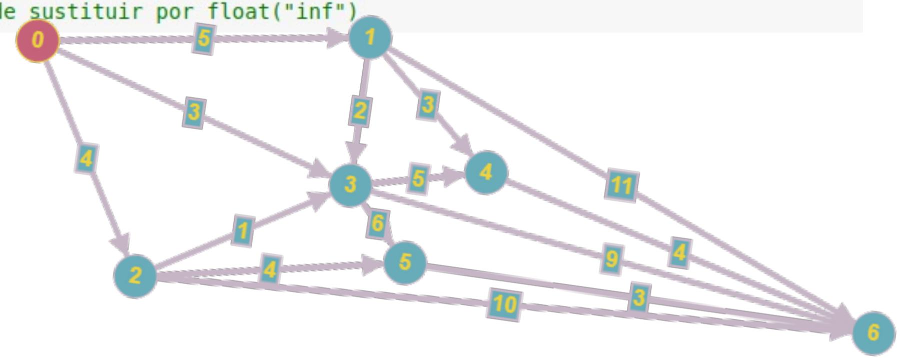
Problema: Viaje por el rio

- Establecemos las tarifas:

```
#Viaje por el rio - Programación dinámica
################################################################################

TARIFAS = [
[0,5,4,3,999,999,999],
[999,0,999,2,3,999,11],
[999,999, 0,1,999,4,10],
[999,999,999, 0,5,6,9],
[999,999, 999,999,0,999,4],
[999,999, 999,999,999,0,3],
[999,999,999,999,999,999,0]
]

#999 se puede sustituir por float("inf")
```



Universidad Internacional de Valencia

Programación dinámica (IV)

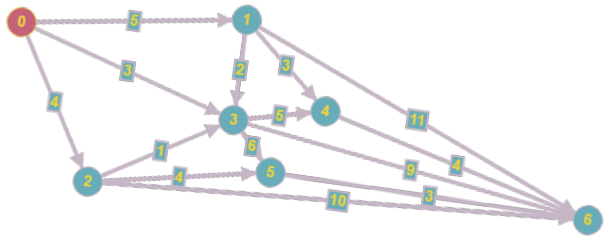
```
#Calculo de la matriz de PRECIOS y RUTAS
#  PRECIOS - contiene la matriz del mejor precio para ir de un nodo a otro
#  RUTAS   - contiene los nodos intermedios para ir de un nodo a otro
#################################################################
def Precios(TARIFAS):
#################################################################
  #Total de Nodos
  N = len(TARIFAS[0])

  #Inicialización de la tabla de precios
  PRECIOS = [ [9999]*N for i in [9999]*N]  #n x n
  RUTA = [ [""]*N for i in [""]*N]

  #Se recorren todos los nodos con dos bucles(origen - destino)
  #  para ir construyendo la matriz de PRECIOS
  for i in range(N-1):
    for j in range(i+1, N):
      MIN = TARIFAS[i][j]
      RUTA[i][j] = i

      for k in range(i, j):
        if PRECIOS[i][k] + TARIFAS[k][j] < MIN:
            MIN = min(MIN, PRECIOS[i][k] + TARIFAS[k][j] )
            RUTA[i][j] = k
        PRECIOS[i][j] = MIN

  return PRECIOS,RUTA
```



Operaciones

n

n

n

$3 \cdot n^3$

```
[ ] TARIFAS = [
    [0,5,4,3,999,999,999],
    [999,0,999,2,3,999,11],
    [999,999, 0,1,999,4,10],
    [999,999,999, 0,5,6,9],
    [999,999, 999,999,0,999,4],
    [999,999, 999,999,999,0,3],
    [999,999,999,999,999,999,0]
    ]
```

(*) En lugar de 999

```
import math
math.inf
```

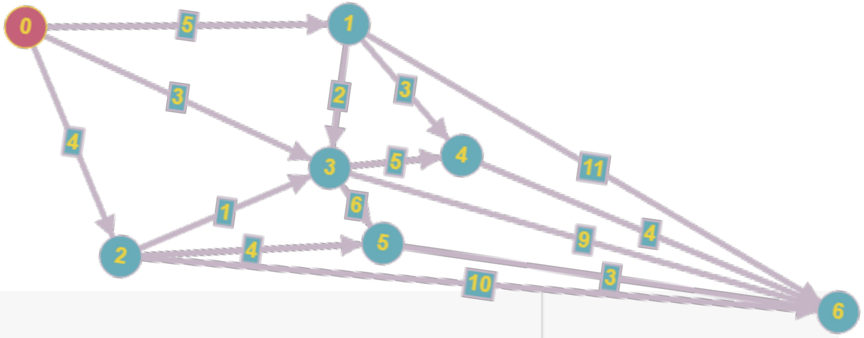
P(i,j) = min {T(i,j) , P(i,k)+T(k,j) para todo i<k<=j }

Universidad Internacional de Valencia

Programación dinámica (V)

- **RUTA contiene la mejor opción intermedia para ir de un nodo a otro**

```
RUTA
['', 0, 0, 0, 1, 2, 5]
['', '', 1, 1, 1, 3, 4]
['', '', '', 2, 3, 2, 5]
['', '', '', '', 3, 3, 3]
['', '', '', '', '', 4, 4]
['', '', '', '', '', '', 5]
['', '', '', '', '', '', '']
```




```python
def calcular_ruta(RUTA, desde, hasta):
  if desde == hasta:
    #print("Ir a :" + str(desde))
    return desde
  else:
    return str(calcular_ruta(RUTA, desde, RUTA[desde][hasta]) ) +  ',' + str(RUTA[desde][hasta])

print("\nLa ruta es:")
calcular_ruta(RUTA, 0,6)
```

Recursividad

AG2 – Actividad Guiada 2

Ramificación y Poda

viu | Universidad Internacional de Valencia

Agenda

- Nuevo tema en el Foro
- Práctica: Programación dinámica(Viaje por el rio).
- Teoría: Ramificación y Poda
- Práctica: Búsqueda en grafos, ramificación y poda(asignación de tareas).
- Practica: Descenso del gradiente.

Ramificación y Poda.

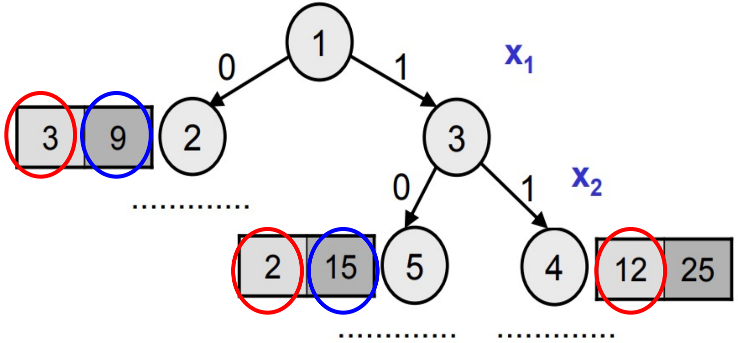
Problema: Asignación de tareas

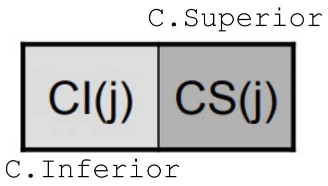
- El problema consiste en **maximizar** el rendimiento (o **minimizar** los costes) en cuanto a la asignación de N tareas a N agentes. Cada tarea solo puede ser asignado a un agente.
- Los beneficios que se obtienen al realizar la **tarea 1 por el agente A** es 9
- La matriz de beneficios es la que se muestra en la figura
- Aplicando Ramificación y Poda, obtener la asignación que maximice los beneficios.

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(V)

- **Estrategia de poda: Poda i si** $CS(i) \leq CI(j)$ Para algún `j` ya generado





C=12(max. de las cotas inferiores)
El nodo **2** no puede mejoran(valor 9) las peores predicciones de **4**(valor 12 = max{todo CI() })
El nodo **5** quizá pueda mejorar al nodo **4**

Revisión

Ramificación y Poda.

Problema: Asignación de tareas

- P.ej. Si asignamos el agente A a la tarea1
  - Cota Inferior para la tarea2 es 4
  - Cota Superior para la tarea2 es 8

	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Ramificación y Poda.

Problema: Asignación de tareas

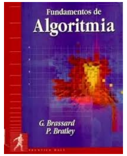
- ¿Cómo diseñamos un estado?
- ¿Como expandimos nodos (ramificación)?
- ¿Como podamos?
- ¿Como recorremos el árbol completo de estados?
- ¿Qué complejidad tiene? (*)

(*) es difícil calcular el número de operaciones exacto. Calcular el **mejor** caso y el **peor** caso.
Peor caso = tengo que expandir todos los niveles.
Mejor caso = puedo podar todos los nodos de un nivel menos uno.
En general podemos asumir una complejidad exponencial

Ramificación y Poda. Practica

```
#Del libro Fundamentos de algoritmia - G. Brassard & P. Bratley pg 349, sec. 9.7.1
'''Hay que asignar n tareas a n agentes, de forma que a cada agente le corresponda una tarea.
Se dispone de una tabla de costes: el coste para el agente i al realizar la tarea j es cij>=0.
Se busca la asignación que reduzca el coste total.
Diferentes aplicaciones,

Ejemplo de tabla de costes para n=4. Agentes: a, b, c y d . Tareas: 1,2,3 y 4
'''
```




```
#   TAREAS
#  A
#  G
#  E
#  N
#  T
#  E
#  S

COSTES=[[11,12,18,40],
        [14,15,13,22],
        [11,17,19,23],
        [17,14,20,28]]
```

La tupla: (1, 0, 3, 2) representa la solución :

Agente 0 a la tarea 1
“ 1 “ 0
“ 2 “ 3
“ 3 “ 2

Universidad Internacional de Valencia

Ramificación y Poda. Practica

Función objetivo

- S es una solución parcial P.ej (1,0,)
- C es la matriz de costes(o beneficios)

```
def valor(S,COSTES):
  VALOR = 0
  for i in range(len(S)):
    VALOR += COSTES[i][S[i]]
  return VALOR

valor((0, 1, 2, 3),COSTES)
```

```
#   TAREAS
#  A
#  G
#  E
#  N
#  T
#  E
#  S

COSTES=[[11,12,18,40],
        [14,15,13,22],
        [11,17,19,23],
        [17,14,20,28]]
```

Ramificación y Poda. Practica

Fuerza Bruta:

```
@calcular_tiempo
def fuerza_bruta(COSTES):
  #Representacion de la solucion será una tupla donde cada valor en la cordenada i-sima es la tarea asignado al agente i
  #  Ejemplo (1,2,3,4) Tiene valor 11+15+19+28=73
  #
  #¿Cuantas posibilidades hay? n! -> Complejidad factorial(exponecial)
  #Con dimension 11 se va a 1 minuto de ejecucion

  mejor_valor = 10e10
  mejor_solucion = ()

  for s in list(itertools.permutations(range(len(COSTES)))):
    #print(s,valor(s,COSTES))
    valor_tmp = valor(s,COSTES)
    if valor_tmp < mejor_valor:
        mejor_valor = valor_tmp
        mejor_solucion = s

  print("La mejor solucion es :", mejor_solucion, " con valor:", mejor_valor  )
```

```
#   TAREAS
#  A
#  G
#  E
#  N
#  T
#  E
#  S

COSTES=[[11,12,18,40],
        [14,15,13,22],
        [11,17,19,23],
        [17,14,20,28]]
```

Ramificación y Poda. Practica

Función para estimar una cota inferior para una solución parcial:

```
#Coste inferior para soluciones parciales
#  (1,3,) Se asigna la tarea 1 al agente 0 y la tarea 3 al agente 1

def CI(S,COSTES):
  VALOR = 0
  #Valores establecidos
  for i in range(len(S)):
    VALOR += COSTES[i][S[i]]

  #Estimacion
  for i in range( len(S), len(COSTES)   ):
    VALOR += min( [ COSTES[j][i] for j in range(len(S), len(COSTES))  ])
  return VALOR
```

Ej. La tupla (2,0,)

Representa un nodo intermedio con :
- Agente 0 asignado a 2
- Agente 1 asignado a 0

Ramificación y Poda. Practica

Función para ramificar:

```
#Genera tantos hijos como como posibilidades haya para la siguiente elemento de la tupla
#(0,) -> (0,1), (0,2), (0,3)
def crear_hijos(NODO, N):
  HIJOS = []
  for i in range(N ):
    if i not in NODO:
      HIJOS.append({'s':NODO +(i,)    })
  return HIJOS
```

Ramificación y Poda. Practica

Proceso principal (I):

```
@calcular_tiempo
def ramificacion_y_poda(COSTES):
#Construccion iterativa de soluciones(arbol). En cada etapa asignamos un agente(ramas).
#Nodos del grafo  { s:(1,2),CI:3,CS:5  }
  #print(COSTES)
  DIMENSION = len(COSTES)
  MEJOR_SOLUCION=tuple( i for i in range(len(COSTES)) )
  CotaSup = valor(MEJOR_SOLUCION,COSTES)
  #print("Cota Superior:", CotaSup)

  NODOS=[]
  NODOS.append({'s':(), 'ci':CI((),COSTES)    } )

  iteracion = 0
```

Inicializamos con una solución cualquiera(opcional)

NODOS guarda los nodos que debemos explorar(ramificación)

continua

Ramificación y Poda. Practica

Proceso principal (II):

```
while( len(NODOS) > 0):
  iteracion +=1

  nodo_prometedor = [ min(NODOS, key=lambda x:x['ci']) ][0]['s']
  #print("Nodo prometedor:", nodo_prometedor)

  #Ramificacion
  #Se generan los hijos
  HIJOS =[ {'s':x['s'], 'ci':CI(x['s'], COSTES)   } for x in crear_hijos(nodo_prometedor, DIMENSION) ]

  #Revisamos la cota superior y nos quedamos con la mejor solucion si llegamos a una solucion final
  NODO_FINAL = [x for x in HIJOS if len(x['s']) == DIMENSION  ]
  if len(NODO_FINAL ) >0:
    #print("\n********Soluciones:",  [x for x in HIJOS if len(x['s']) == DIMENSION  ] )
    if NODO_FINAL[0]['ci'] < CotaSup:
      CotaSup = NODO_FINAL[0]['ci']
      MEJOR_SOLUCION = NODO_FINAL

  #Poda
  HIJOS = [x for x in HIJOS if x['ci'] < CotaSup   ]

  #Añadimos los hijos
  NODOS.extend(HIJOS)

  #Eliminamos el nodo ramificado
  NODOS =  [  x for x in NODOS if x['s'] != nodo_prometedor    ]

print("La solucion final es:" ,MEJOR_SOLUCION , " en " , iteracion , " iteraciones" , " para dimension: " ,DIMENSION  )
```

1 De entre los nodos pendientes a explorar, elegimos el mas prometedor

2

3

4

5

6

Universidad Internacional de Valencia

Ramificación y Poda. Practica

Análisis para **mejorar nota:**

- Generar matrices con valores aleatorios de mayores dimensiones (5,6,7,…) y ejecutar ambos algoritmos.
- ¿A partir de que dimensión el algoritmo por fuerza bruta deja de ser una opción?
- ¿Hay algún valor de la dimensión a partir de la cual el algoritmo de ramificación y poda también deja de ser una opción válida?

importante

+1 para mejoar (9/10)

AG2 – Actividad Guiada 2

Descenso del Gradiente

Universidad Internacional de Valencia

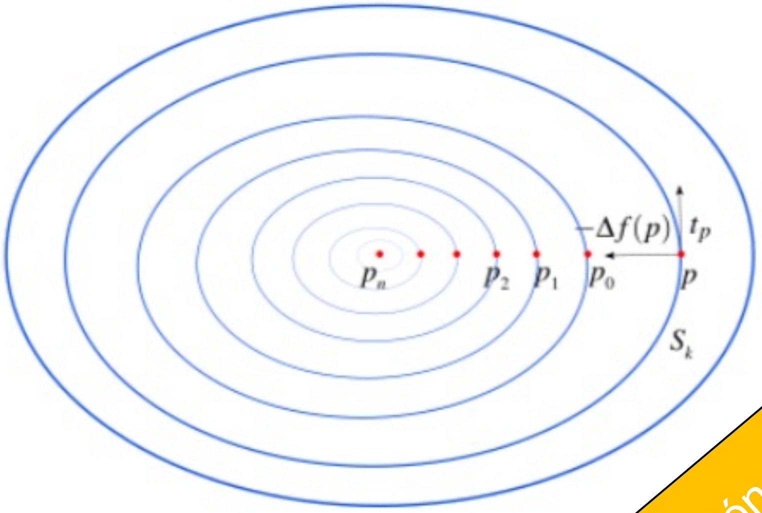
Agenda

- Nuevo tema en el Foro
- Práctica: Programación dinámica(Viaje por el rio).
- Teoría: Ramificación y Poda
- Práctica: Búsqueda en grafos, ramificación y poda(asignación de tareas).
- Practica: Descenso del gradiente.

Descenso del Gradiente – AGD

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

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



Donde el parámetro α_t se selecciona

para que p_{t+1} sea solución

Y $f(p_t) \geq f(p_{t+1})$

Revisión

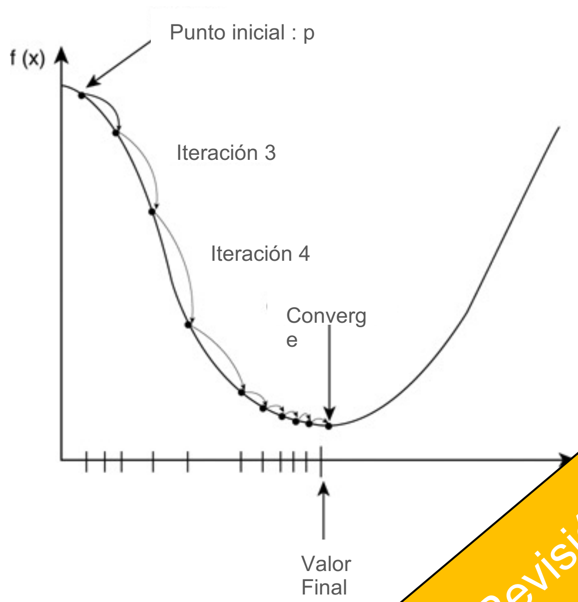
Descenso del Gradiente – AGD

- En general se va reduciendo el valor de α_t dinámicamente a medida que nos aproximamos a la solución que podemos deducir por:

  - la magnitud del gradiente

  - cantidad de iteraciones que hemos realizado

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



Revisión

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

Revisión

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

Revisión

Descenso del gradiente. Práctica

Preparar entorno

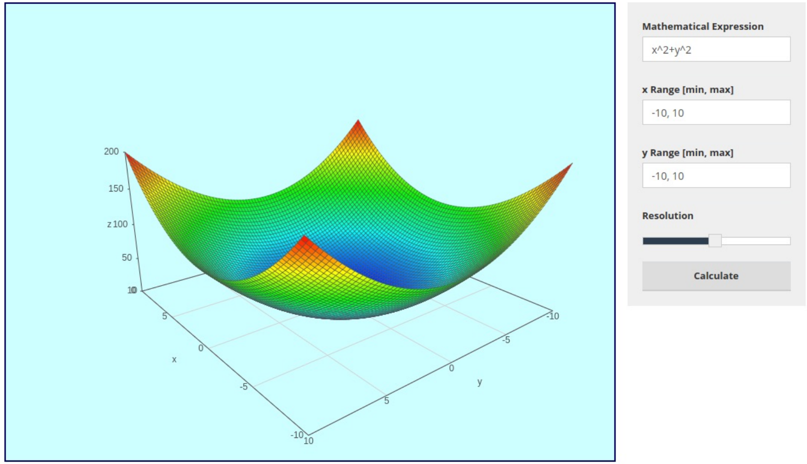
```
import math                       #Funciones matematicas
import matplotlib.pyplot as plt   #Generacion de gráficos (otra opcion seaborn)
import numpy as np                #Tratamiento matriz N-dimensionales y otras (fundamental!)
#import scipy as sc

import random
```

Descenso del gradiente. Práctica

La función a minimizar. Paraboloide

```python
f  = lambda X:       X[0]**2+X[1]**2        #Funcion
df = lambda X: [2*X[0] , 2*X[1]]            #Gradiente
```



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

Descenso del gradiente. Práctica

Prepara los datos para el gráfico

```
#Prepara los datos para dibujar mapa de niveles de Z
resolucion = 100
rango=2.5
X=np.linspace(-rango,rango,resolucion)
Y=np.linspace(-rango,rango,resolucion)
Z=np.zeros((resolucion,resolucion))
for ix,x in enumerate(X):
  for iy,y in enumerate(Y):
    Z[iy,ix] = f([x,y])

#Pinta el mapa de niveles de Z
plt.contourf(X,Y,Z,resolucion)
plt.colorbar()
```

Descenso del gradiente. Práctica

Generamos un Punto aleatorio

```
#Generamos un punto aleatorio
P=[random.uniform(-2,2  ),random.uniform(-2,2 ) ]
plt.plot(P[0],P[1],"o",c="white")
```

Descenso del gradiente. Práctica

Iteramos el algoritmo

```
#Tasa de aprendizaje
TA=.1

#Iteraciones
for _ in range(500):
  grad = df(P)
  #print(P,grad)
  P[0],P[1] = P[0] - TA*grad[0] , P[1] - TA*grad[1]
  plt.plot(P[0],P[1],"o",c="red")
```

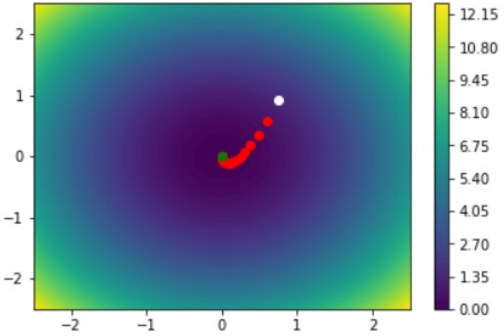
$$p_{t+1} = p_t - \alpha_t \nabla f(p_t)$$

Descenso del gradiente. Práctica

Pintamos el gráfico con las iteraciones

```
plt.plot(P[0],P[1],"o",c="green")
plt.show()
print("Solucion:" , P , f(P))
```

```
Punto Inicial: [0.7569403161678125, 0.9223403659681089]
```

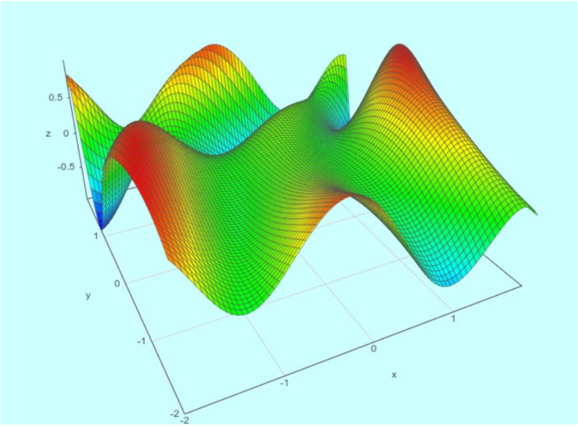



```
Solucion: [-0.004999999999999998, -0.005000000000000004] 5.000000000000002e-05
```

Descenso del gradiente. Práctica

Otra función a minimizar.

```
#Definimos la funcion
#sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2*x + 1 - E^y)
f  = lambda X: np.sin(1/2 * X[0]**2 - 1/4 * X[1]**2 + 3) * np.cos(2 * X[0] + 1 - np.e**X[1])
```

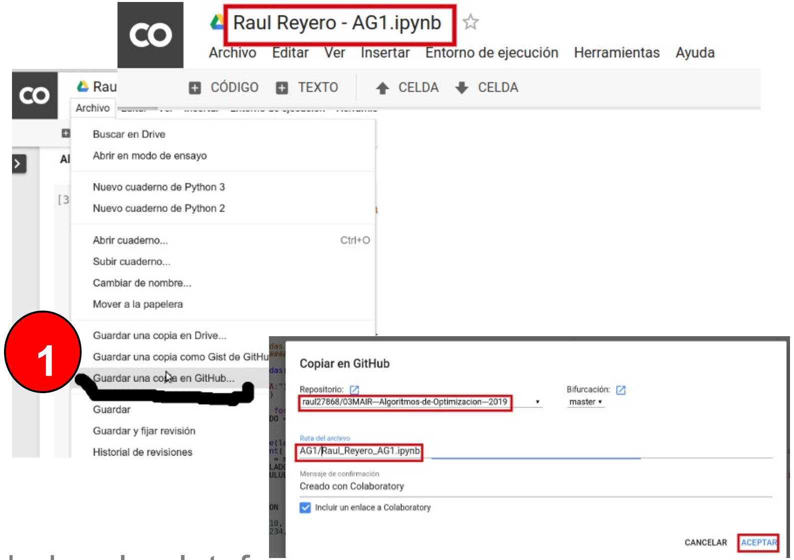


sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2*x + 1 - E^y)

Online 3D Function Visualizer: http://al-roomi.org/3DPlot/index.html

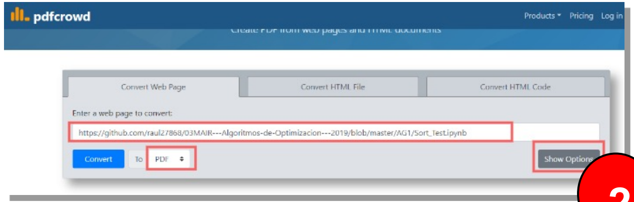
Finalizar la actividad. Grabar, subir a GitHub, Generar pdf (I)

- Guardar en GitHub
  Repositorio: 03MIAR ---Algoritmos de Optimizacion
  Ruta de Archivo con **AG2**



1

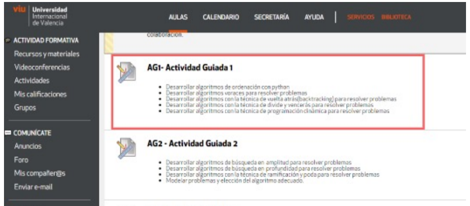
- Generar pdf (con https://pdfcrowd.com)



2

- Descargar pdf y adjuntar el documento generado a la actividad en la plataforma
  - Adjuntar .pdf en la actividad
  - URL GitHub en el texto del mensaje de la actividad

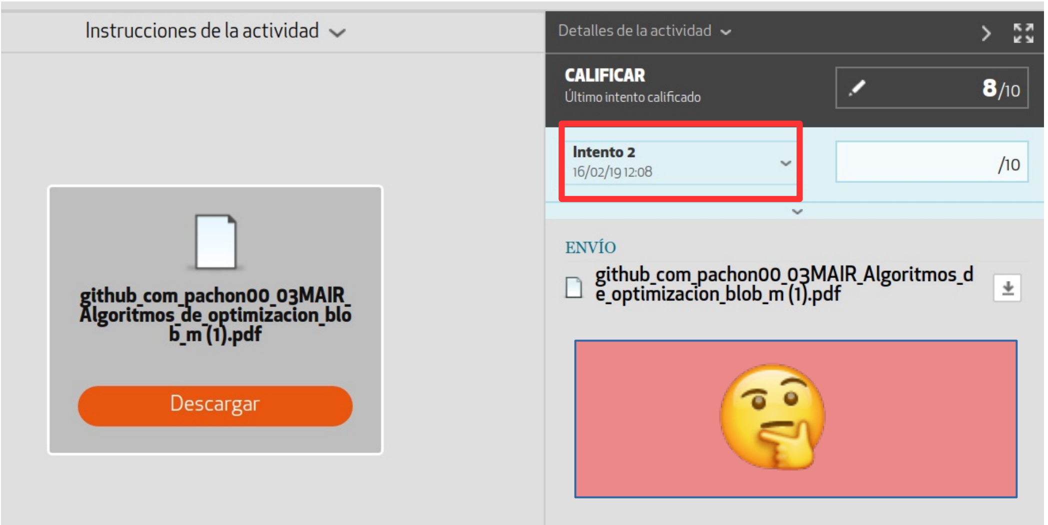
3



viu Universidad Internacional de Valencia

Actividades Guiadas.

- Si hay aportaciones, **añadir explicación**
- Si es la 2ª entrega, **añadir explicación**
- Si tienes dudas de que la 2ª entrega sea valida, **enviame un correo.**



Ramificación y Poda. Practica

Análisis para mejorar nota:

- ¿Que complejidad tiene el algoritmo por fuerza bruta?
- Generar matrices con valores aleatorios de mayores dimensiones (5,6,7,…) y ejecutar ambos algoritmos.
- ¿A partir de que dimensión el algoritmo por fuerza bruta deja de ser una opción?
- ¿Hay algún valor de la dimensión a partir de la cual el algoritmo de ramificación y poda deja de ser una opción válida?

importante

8/10

+1 para mejoar

Descenso del gradiente. Practica

Práctica para **mejorar nota:**

- Minimizar la función por descenso del gradiente

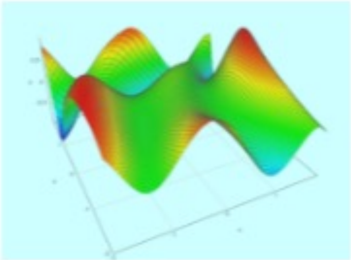
importante

¿Te atreves a optimizar la función?:

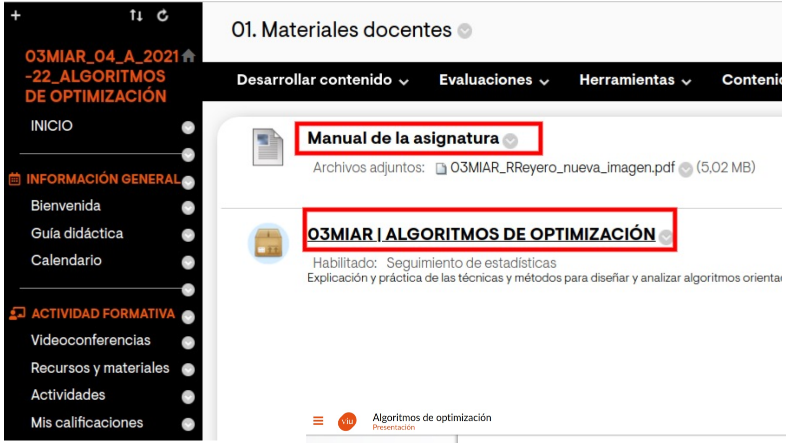
$$f(x) = sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2 * x + 1 - e^y)$$

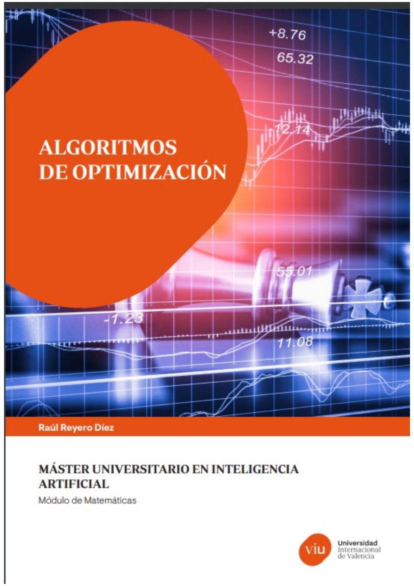
8/10

+1 para mejoar



Manual de la asignatura





Ampliación de conocimientos y habilidades

- Bibliografía

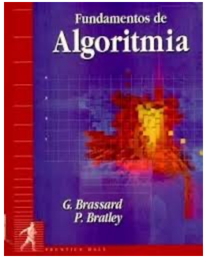
-Brassard, G., y Bratley, P. (1997). Fundamentos de algoritmia. *ISBN 13: 9788489660007*

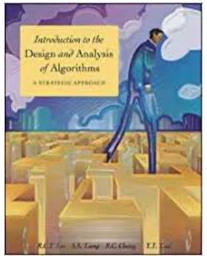
-Guerequeta, R., y Vallecillo, A. (2000). Técnicas de diseño de algoritmos.(http://www.lcc.uma.es/~av/Libro/indice.html)

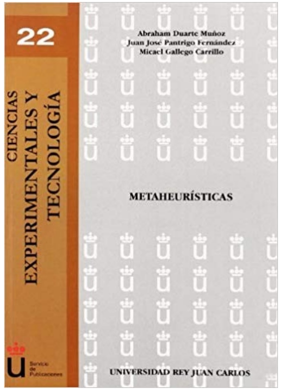
-Lee, R. C. T., Tseng, S. S., Chang, R. C., y Tsai, Y. T. (2005). Introducción al diseño y análisis de algoritmos. *ISBN 13: 9789701061244*

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

- Practicar







Próxima clase. VC4

Introducción a las metaheurísticas

1. Búsqueda aleatoria
2. Búsqueda basada en trayectorias
3. Métodos basados en trayectorias. Búsqueda Tabú
4. Métodos basados en trayectorias. Recocido Simulado
5. Métodos constructivos. Multiarranque
6. Métodos constructivos. GRASP
7. Métodos poblacionales. Colonia de hormigas

Gracias

jcamacho@professor.universidadviu.com