Index

Objectius

En aquest PAC es practicaran els conceptes del temari relacionat amb optimització, en una vessant pràctica amb dades sintètiques.

Descripció de la PAC

Un important centre d'investigació en biologia i ciències ambientals està centrant els seus esforços en identificar i controla una plaga d'un determinat fong desconegut fins ara, que ataca de manera diferent a multitud d'espècies vegetals, posant en perill la salut dels boscos del sud d'Europa. Com la majoria de plagues, si es detecta un focus aviat, és fàcil controlar-ho i extingir-ho. A mesura que avança el temps sense actuar sobre un nou focus, més difícil és poder eradicar-ho.
Aquest centre té accés a dades obtingudes gràcies a un satèl·lit especialitzar en motorització de vegetació i biomassa. Les dades que es necessiten per poder identificar un nou focus d'aquest tipus de fong consisteixen en imatges de 5 tipus.

- Imatges RGB
- Imatges tèrmiques
- Imatges infrarojes
- Imatges RADAR
- Imatges LíDAR

Les imatges provinents del satèl·lit es transmeten contínuament en paquets de no més de 1000 imatges en total (la suma de tots els tipus d'imatges), i a on hi ha , almenys, 10 imatges de cada tipus, i com a màxim, 255 imatges d'un mateix tipus.
A mesura que es van rebent imatges, aquestes són processades contínuament per un complex software que determina si hi ha prou indicis en alguna determinada àrea de les imatges que facin sospitar de la presència d'un focus. Aquest pas és clau per a indicar al satèl·lit que se centri en obtenir més fotos d'aquesta zona, amb millor resolució, per tal de, posteriorment, determinar si realment és necessari enviar-hi una brigada d'extinció.
Aquest software està preparat per processar seqüencial-ment (un darrere l'altre) els paquets que van arribant des del satèl·lit, i necessita, per cada àrea que s'explora, una determinada quantitat d'imatges de cada tipus per emetre una estimació sobre si s'ha d'analitzar amb millor resolució aquesta zona o no.
Aquest procés pot trigar hores en dur-se a terme, però aquest temps és variable, en funció de la quantitat d'imatges de cada tipus que conté el paquet que està processant, i la proporció entre els diferents tipus d'imatges.
Els enginyers de software d'aquest programa que, donada una representació de paquet d'imatges, et retorna el temps estima (en ms) que trigarà el software en donar resposta. La intenció d'aquest simulador és fer-ho servir per determinar quina és la millor configuració de paquet (quina és la millor quantitat d'imatges de cada tipus per paquet) per obtenir una la menor quantitat de temps, ja que aquesta configuració desitjada pot ser informada al satèl·lit per a què l'adopti en la generació dels paquets que envia.
És molt important optimitzar aquest procés, atès que el satèl·lit només passa un cop per dia durant un interval limitat de temps per la nostra zona d'interès, així que hem de ser capaços d'indicar ràpidament si cal millorar la resolució de les imatges d'una determinada àrea, o si no haurem d'esperar a que ho faci al dia següent.

Activitat 1 (2 punts)

Dissenyeu una codificació per a la representació d'un paquet de dades, pensant que aquest paquet de dades serà l'element bàsic de treball (un individu o genoma) d'un algorisme genètic que optimitzarà la seva estructura per tal de minimitzar el temps de resposta del simulador. Explique la seva estructura i com la programareu a DEAP.
Recordeu que els paquets consten d'una determinada quantitat d'imatges, i aquestes imatges són de 5 tipus diferents.
Aquesta representació ha de ser diferent a la «trivial» que se us proporciona als fitxers adjunts. Recordeu també que una bona representació ha de permetre els mecanismes de creuament i mutació, i que quant més granular sigui aquesta representació, més efectius són aquests mecanismes. Important: raoneu i justifiqueu la vostra decisió.

Per representar un paquet de dades (individu o genoma) que s'utilitzarà a l'algorisme genètic, hem de tindre en compte que les dades enviades consisteixen en imatges de 5 tipus, de només de 1000 imatges en total i de cada tipus entre 10 i 255, ambdós inclosos. Per tant sempre ha de tindre imatges dels 5 tipus, indicar el tipus degut a que es variable, i el número de cada grup i el total.

Per simplificar-ho es crearà una llista de 10 valors on es succeiran valors del tipus d'imatge i el número d'imatges per a cadascun dels 5 tipus d'imatges. D'aquesta manera es manté el tipus list segons s'indica al codi base.

```
#GA initialization
creator.create('FitnessMin', base.Fitness, weights=(-1.0,))
creator.create('Individual', list, fitness=creator.FitnessMin)

#Register the attributes, individuals, population... according to your individual coding
toolbox.register(, , , )
toolbox.register(, , , , n=)
toolbox.register(, , list, )
```

Activitat 2 (4.5 punts)

El fitxer P3.py serà l'executable del nostre programa. Teniu diversos forats a omplir per completar les següents qüestions. També haureu de completar el fitxer analyzer.py. Aquest últim conté les definicions de les funcions que cridarà el nostre algorisme genètic per dur a terme avaluació de cada individu (és a dir, obtenir quant temps trigaria en ser processat). El funcionament resumit és que a P3 farem evolucionar les poblacions durant un cert nombre de generacions. Per a això, farem servir les funcions analyze_performance i la funció que haureu d'implementar per descodificar (decode) del fitxer analyzer.py.

La funció analyze_performance només l'heu de modificar per canviar la crida a la funció de descodificació, tal i com s'hi indica al propi fitxer. Aquesta funció farà servir la funció simulator per simular el temps de resposta.

IMPORTANT: el codi proporcionat no està preparat per ser executat fins que no es completi, almenys, l'apartat a). Seguiu sempre les indicacions marcades en comentaris precedits per #TO DO. No està permès modificar la funció simulator ni el fitxer proportion_factors.py.

Prepareu l'algorisme genètic amb DEAP per resoldre el problema d'optimització plantejat. Concretament:

a) Completeu els passos que ja teniu disponibles al fitxer P3.py per poder dur a terme el procés evolutiu amb valors variables de nombre de generacions, mida de les poblacions, probabilitat de creuament i probabilitat de mutació. Consulteu la documentació oficial de DEAP per escollir el mètode de mutació que millor s'adapti a les vostres necessitats.

Per a escollir el mètode de mutació, he tingut en compte que existeixen diversos condicionats com un valor mínim d'imatge per cada tipos, un màxim per el total d'imatges d'un paquet i que els valors siguin integers, per tant es podria utilitzar la funció tools.mutShuffleIndexes que només intercanvia el número d'imatges entre els tipus o deap.tools.**mutUniformInt** que permet indicar valors mínim i màxim. Descarto altres com deap.tools.**mutFlipBit** que s'utilitzen per individuals booleans o deap.tools.**mutGaussian** amb valors reals**.** Per tal de simplificar-ho he utilitzat 'deap.tools.mutShuffleIndexes'.

En blau el que he afegit de codi.

```
#Register the attributes, individuals, population... according to your individual coding
toolbox.register('attr_numberImages', random.randint, 0, 255)
toolbox.register('individual', tools.initRepeat, creator.Individual,
                 toolbox.attr_numberImages,
                 n=10)
toolbox.register('population', tools.initRepeat, list, toolbox.individual)

#Register the GA operations
toolbox.register('evaluate', objective_function)
toolbox.register('mate', tools.cxTwoPoint)
toolbox.register('mutate', tools.mutShuffleIndexes, indpb=0.1)
toolbox.register('select', tools.selTournament, tournsize=3)
```

b) Implementeu una funció de decodificació de genoma que rebi un individu codificat segons la vostra proposta de l'activitat anterior, i retorni una tupla de 5 integers (cadascuna de les quantitats d'imatges de cada tipus). Podeu veure un exemple de decodificació trivial, on independentment de l'individu rebut, la decodificació és una tupla d'integers entre 50 i 100.

Per codificar decodificar he extret els valors del número d'imatges i retornat com una tupla de 5 integers.

```
def decode(individual):
    decoded_individual = (None,None,None,None,None)
    #TO DO: Decode the received individual so that this function returns a tuple of 5 integers
    decoded_individual = (individual[1],
                          individual[3],
                          individual[5],
                          individual[7],
                          individual[9])
    return decoded_individual
```

c) Recolçant-vos en la funció de decodificació que acabeu de crear, completeu la implementació de la funció fulfill_constraints, que és el decorador que permet satisfer les restriccions sobre la quantitat d'imatges que conté el paquet de dades, i la quantitat d'imatges de cada tipus. Feu

servir els arguments max_imagery i min_imagery_type. Descomenteu les linies indicades per decorar els mètodes de generació de població, creuament i mutació, i comproveu el correcte funcionament del vostre codi.

Seguint l'ajut primer es comprova el condicionant que el número d'imatge per a cada tipus tingui un valor igual o superior a 10. Per a cada valor de la tupla que retorna (analyzer.decode(child)) identifico els que no compleixen i els assigno un valor aleatori entre 10 i 255 per assegurar que compleix els requisits de l'enunciat.

```
#firstly, minimum 10 images per image type
  for i in range(0,len(decoded_child)):
     if decoded_child[i] < 10:
        decoded_child[i] = random.randint(10, 254)
```

```
Comprovació de mínim detectat i revisat:
MinDectedted:  3
Decoded_child. N_Images[ 578 ]  253   128   128   66   3
Decoded_child_revised. N_Images[ 731 ]  253   128   128   66   156
```

Després es comprova que la suma del número d'imatges de cada decoded_child no sigui de més de 1000 imatges, en cas contrari, assigna un número d'imatges per a cada tipus i torna a fer la comprovació.

```
   #secondly, max amount of images: 1000
    while sum(decoded_child) > max_imagery:
      print("MaxDectedted: ",sum(decoded_child))
      #Recorre els diferents números d'imatges i assigna valors aleatoris entre 10 i 255
      for i in range(0,len(decoded_child)):
         decoded_child[i] = random.randint(10, 255)
```

```
Comprovació de mínim detectat i revisat:
MaxDectedted:  1013
Decoded_child. N_Images[ 1013 ]  230   182   252   133   216
Decoded_child_revised. N_Images[ 682 ]  150   200   103   50   179
```

Per últim he creat una funció encode(decoded_child) que a partir d'una tupla de 5 integers crea una tupla de 10 integers amb el tipos dimatges i el número d'imatges.

```
def encode(decoded_individual):
   encoded_individual = (1,decoded_individual[0],
                         2,decoded_individual[1],
                         3,decoded_individual[2],
                         4,decoded_individual[3],
                         5,decoded_individual[4],)
   return encoded_individual
```

Es crida amb `new_child = analyzer.encode(decoded_child)`.

Activitat 3 (1.5 punts)

a) Expliqueu com funcionen els operadors de creuament i mutació del programa. Quina és la finalitat de cadascun?

Els operadors de creuament el que fan es a partir de dos individus existents, emula la selecció natural, creant individus que poden estar millor adaptat, on la funció 'objective_funtion' donaria un millor resultat. Hi han diferents maneres de creuar on els individus intercanvien les seves longituds, altres que mantenen la longitud i alguns que s'intercanvien valors segons una probabilitat donada. Segons la documentació de DEAP tenim 12 formes de creuar individus. En aquesta PAC s'utilitza cxTwoPoint() que consisteix en intercanviar els valors però mantenint la longitud dels individus.

Els operadors de mutació serveixen per introduir en la població d'individus gens que no es troben i per tant cercar noves solucions que poden ser millors. En aquesta PAC s'utilitzo tools.mutShuffleIndexes que només intercanvia el número d'imatges de valors sencers amb una probabilitat de que succeeixi del 1% indicada amb el paràmetre indpb=0.1.

b) Quines conseqüències tindria prescindir de l'operació de mutació?

En cas que no utilitzéssim la operació de mutació, llavors ens limitem a avaluar la solució optima per un espai de solucions determinat per la població inicial i que només evolucionarà per creuament. Això exclouria la possibilitat de avaluar espais de solucions no sorgits en la població inicial.

c) Expliqueu breument una codificació alternativa a la proposada a l'Activitat 1 que també permetés aplicar aquests operadors.

Com alternativa de codificació proposo la següent:

Llista d'elements on s'inclogui per a cada tipus d'imatge, tant el número d'imatge com les dades de les imatges. El primer serà el número de tipus d'imatge i el següent les dades d'imatge. On l'element Y tingui un valor que representi el tipus d'imatge, entre 0 i 5. L'estructura del paquet de dades es representara com un diccionari {idPaquet: (imageType0,imageType1,imageType2,imageType3,imageType4)} on imageType seria {imageTypeX: (numeroImages, dadesImages)}.

Activitat 4 (2 punts)

Per a aquesta activitat, heu de realitzar un canvi al codi. A la funció analyze_performance(individual) del fitxer analyzer.py, heu de canviar el bucle while, de manera que passarà de ser així:

```
#Exercise 4 - Comment next line:
sim_time,remaining_samples = simulator(samples,remaining_samples)
#Exercise 4 - Uncomment next line:
#sim_time,remaining_samples =simulator(samples,remaining_samples,True)

analysis_time += sim_time
```

A ser així:

```
#Exercise 4 - Comment next line:
#sim_time,remaining_samples = simulator(samples,remaining_samples)
#Exercise 4 - Uncomment next line:
sim_time,remaining_samples = simulator(samples,remaining_samples,True)

analysis_time += sim_time
```

És a dir, comentem una línia, i des-comentem altra.

a) Amb aquesta configuració, és possible que en una generació l'individu guanyador ens doni un temps de simulació superior al de l'individu guanyador de l'anterior generació? Per què?

La diferencia es que amb True, pasem del resultat time de la opció 1 a la 2.

1. times = list(np.array(samples) * np.array(tbase))
2. times = list(map(lambda x: x*random.uniform(0.985,1.015),np.array(samples) * np.array(tbase)))

Mentre que amb el paràmetre per defecte, False el temps de simulació va disminuint o es manté amb cada generació, amb valor True es veu que es pot obtenir un pitjor temps en següents generacions, com es veu a continuació marcat en groc. Això es degut a que en la primera opció el calcul de temps es basa en número d'imatges (samples) i els valors definits amb tbase, però a la segona opció, s'afegeix un component aleatori (x*random.uniform(0.985,1.015)) que pot fer que d'una generació a la següent el temps de simulació sigui superior fins i tot amb els mateixos samples.

Resultat amb False:

```
Decode:  0 (5512791.7919999994,) (255, 239, 146, 108, 114) [1, 255, 2, 239, 3, 146, 4, 108, 5, 114]
Decode:  1 (3453508.7999999993,) (180, 158, 141, 91, 99) [1, 180, 2, 158, 3, 141, 4, 91, 5, 99]
Decode:  2 (2401911.4199999999,) (255, 74, 146, 108, 114) [1, 255, 2, 74, 3, 146, 4, 108, 5, 114]
Decode:  3 (2385111.4199999999,) (255, 74, 141, 108, 114) [1, 255, 2, 74, 3, 141, 4, 108, 5, 114]
Decode:  4 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  5 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  6 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  7 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  8 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  9 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  10 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  11 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  12 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  13 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  14 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  15 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  16 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  17 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  18 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
Decode:  19 (2179575.4199999999,) (255, 74, 141, 91, 99) [1, 255, 2, 74, 3, 141, 4, 91, 5, 99]
```

Resultat amb True:

```
Decode:  1 (3172289.0581556815,) (231, 103, 227, 120, 123) [1, 231, 2, 103, 3, 227, 4, 120, 5, 123]
Decode:  2 (3170843.2085997425,) (231, 103, 227, 120, 123) [1, 231, 2, 103, 3, 227, 4, 120, 5, 123]
Decode:  3 (3033944.7868801835,) (231, 103, 191, 120, 123) [1, 231, 2, 103, 3, 191, 4, 120, 5, 123]
Decode:  4 (3040821.8959164689,) (231, 103, 191, 120, 123) [1, 231, 2, 103, 3, 191, 4, 120, 5, 123]
Decode:  5 (2808079.5749068735,) (231, 103, 131, 120, 123) [1, 231, 2, 103, 3, 131, 4, 120, 5, 123]
Decode:  6 (2811680.6186648975,) (231, 103, 131, 120, 123) [1, 231, 2, 103, 3, 131, 4, 120, 5, 123]
Decode:  7 (2809232.2938311659,) (231, 103, 131, 120, 123) [1, 231, 2, 103, 3, 131, 4, 120, 5, 123]
Decode:  8 (2777336.7425121679,) (231, 103, 131, 120, 120) [1, 231, 2, 103, 3, 131, 4, 120, 5, 120]
```

```
Decode:  9 (2605779.9280828126,) (231, 103, 131, 120, 96) [1, 231, 2, 103, 3, 131, 4, 120, 5, 96]
Decode:  10 (2445486.6555400891,) (231, 103, 131, 120, 77) [1, 231, 2, 103, 3, 131, 4, 120, 5, 77]
Decode:  11 (2446819.2073372491,) (231, 103, 131, 120, 77) [1, 231, 2, 103, 3, 131, 4, 120, 5, 77]
Decode:  12 (2258255.700835309,) (231, 103, 133, 82, 77) [1, 231, 2, 103, 3, 133, 4, 82, 5, 77]
Decode:  13 (2252895.8687718203,) (231, 103, 133, 82, 77) [1, 231, 2, 103, 3, 133, 4, 82, 5, 77]
Decode:  14 (2251774.2386453333,) (231, 103, 133, 82, 77) [1, 231, 2, 103, 3, 133, 4, 82, 5, 77]
Decode:  15 (2237611.3813350643,) (231, 103, 131, 82, 77) [1, 231, 2, 103, 3, 131, 4, 82, 5, 77]
Decode:  16 (2245364.8991698371,) (231, 103, 131, 82, 77) [1, 231, 2, 103, 3, 131, 4, 82, 5, 77]
Decode:  17 (2181217.5115173152,) (231, 83, 131, 82, 77) [1, 231, 2, 83, 3, 131, 4, 82, 5, 77]
Decode:  18 (2247769.5190605498,) (231, 103, 131, 82, 77) [1, 231, 2, 103, 3, 131, 4, 82, 5, 77]
Decode:  19 (2247360.5577938207,) (231, 103, 131, 82, 77) [1, 231, 2, 103, 3, 131, 4, 82, 5, 77]
```

b) Comproveu i discutiu el rendiment de l'algorisme genètic, tenint en compte l'efecte del nombre d'individus per població, nombre de generacions, i taxes de creuament i mutació. Mostreu gràfiques de convergència cap a una solució en funció del nombre de generacions i del nombre d'individus per població per a les 3 combinacions de taxes de creuament i mutació que considereu més interessants. Dueu a terme vàries execucions amb la mateixa configuració abans de treure conclusions.