prog_bio_5_2

July 23, 2018

1 Programación para la Bioinformática

1.1 Unidad 5: ADN, ARN, secuencias y motivos (Parte 2)

1.1.1 Instrucciones de uso

A continuación se presentará la teoría y algún ejemplo de algoritmo genético. Recordad que podéis ir ejecutando los ejemplos para obtener sus resultados.

1.1.2 Algoritmos inspirados en la naturaleza

Existe una categoría de algoritmos que utilizan conceptos basados o inspirados en la naturaleza estableciendo una metáfora que los hace más comprensibles para los humanos. Muy populares en algoritmos de inteligencia artificial, empezaron a aparecer en la década de los 70 del siglo pasado y en la última década han explotado hasta convertirse en métodos casi estándares.

Una familia de algoritmos de inteligencia artificial inspirados en la naturaleza muy populares son los **algoritmos genéticos**. Los algoritmos genéticos utilizan conceptos de la genética, como son las mutaciones, los mecanismos de selección o los cruces. Los algoritmos genéticos se utilizan con el objetivo de optimizar valores de una función cualquiera en su espacio de valores. El funcionamiento básico del algoritmo está descrito en la siguiente figura (fuente Wikipedia - https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico):

- **Inicialización** (I). Se genera aleatoriamente una población inicial, constituida por un conjunto de cromosomas (o también llamados genes) que representan posibles soluciones del problema. Esta población deberá tener una diversidad inicial lo suficientemente rica para garantizar que el algoritmo no converja de forma prematura en soluciones no óptimas.
- **Evaluación** (?). Para cada uno de los cromosomas, lo evaluaremos en el espacio de búsqueda (aplicaremos la función que deseamos optimizar) y después calcularemos la distancia a la solución que queremos obtener. Esta solución objetivo es muy importante y está codificada en la función de *fitness* que dirigirá la evolución hacia esa solución óptima (podemos conocerla o no, en este segundo caso, la expresaremos en forma de función: cuán rápido es un coche, cuál es la cantidad monetaria más grande, etc.). Deberemos, además, definir las condiciones de parada del algoritmo para no entrar en bucle infinito: o bien acotando el número de pasos del algoritmo o bien cuando en la población ya no haya cambios.
- **Selección** (Se). Si no se ha dado la condición de parada, se procede a elegir los cromosomas que serán cruzados en la siguiente generación, para ello, seleccionaremos los mejores cromosomas ordenándolos por su aptitud.

- **Cruce** (Cr). Representa en esta metáfora la reproducción sexual y opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.
- **Mutación** (Mu). Modifica de forma aleatoria parte del cromosoma de los individuos de la población para añadir diversidad y poder salir de pozos locales en el espacio de búsqueda.
- **Reemplazo** (Re). Una vez aplicados los operadores genéticos, se seleccionan los mejores individuos para conformar la población de la generación siguiente y continuar con otro paso de la simulación.

1.1.3 Ejercicio 1

El siguiente código es una implementación de un algoritmo genético que optimiza la búsqueda de un string, es decir, dado un string **objetivo**, intentad encontrar esa cadena empezando desde diversas cadenas con caracteres aleatorios.

```
In [10]: import random
         import string


         objetivo = "python"


         GENES = 20
         MAX_GENERACION = 600


         class Individuo(object):
             def __init__(self, adn, fitness):
                 self.adn = adn
                 self.fitness = fitness


         def calcular_fitness(origen, valor_objetivo):
             fitness = 0
             for i in range(0, len(origen)):
                 fitness += (ord(valor_objetivo[i]) - ord(origen[i])) ** 2
             return fitness


         def mutacion(padre1, padre2):
             adn_hijo = padre1.adn[:]

             start = random.randint(0, len(padre2.adn) - 1)
             stop = random.randint(0, len(padre2.adn) - 1)
             if start > stop:
                 stop, start = start, stop

             adn_hijo[start:stop] = padre2.adn[start:stop]
```

```
    posicion = random.randint(0, len(adn_hijo) - 1)
    adn_hijo[posicion] = chr(ord(adn_hijo[posicion]) + random.randint(-1, 1))
    fitness_hijo = calcular_fitness(adn_hijo, objetivo)
    return Individuo(adn_hijo, fitness_hijo)


def padre_al_azar(poblacion):
    return poblacion[int(random.random() * random.random() * (GENES - 1))]


def escribe_generacion(generacion, poblacion):
    print('Pasos de simulación: %d' % generacion)
    print()
    print('  Fitness         ADN')
    print('------------------------')
    for candidato in poblacion:
        print("%6i %15s" % (candidato.fitness, ''.join(candidato.adn)))
    print()


def inicializa_poblacion():
    poblacion = []
    for i in range(0, GENES):
        adn = [random.choice(string.printable[:-5]) for _ in range(0, len(objetivo))]
        fitness = calcular_fitness(adn, objetivo)
        candidate = Individuo(adn, fitness)
        poblacion.append(candidate)
    return poblacion


def simulacion():
    poblacion = inicializa_poblacion()
    generacion = 0
    while True and generacion < MAX_GENERACION:
        generacion += 1
        poblacion.sort(key=lambda candidate: candidate.fitness)

        if poblacion[0].fitness == 0:
            break

        padre1 = padre_al_azar(poblacion)
        padre2 = padre_al_azar(poblacion)

        hijo = mutacion(padre1, padre2)
        if hijo.fitness < poblacion[-1].fitness:
            poblacion[-1] = hijo
```

```
            if generacion == MAX_GENERACION:
                print(u'Se ha alcanzado el máximo de generaciones')
            escribe_generacion(generacion, poblacion)


        simulacion()
```

```
Pasos de simulación: 575

  Fitness         ADN
-------------------------
     0          python
     1          oython
     1          oython
     1          pzthon
     1          oython
     1          oython
     1          oython
     2          ozthon
     2          pxthom
     2          oxthon
     2          oxthon
     2          pxthom
     2          oythpn
     2          ozthon
     2          pxthoo
     2          oytgon
     2          oyshon
     2          qzthon
     2          oyshon
     2          oythpn
```

Es muy importante, tanto en bioinformática como en programación en general, leer e interpretar código de otros programadores. Por ello, en este ejercicio se os pide que comentéis el código anterior con comentarios en el propio código que expliquen las partes más importantes de este.

1.1.4 Ejercicio 2

Escribe una función de fitness alternativa. Recuerda que fitness=0 indica que la cadena objetivo se ha conseguido. Explica en qué consiste tu función de fitness.

```
In [11]: # Respuesta
```

1.1.5 Ejercicio 3

Representa utilizando matplotlib el máximo fitness, el mínimo y la media por paso de la simulación en un gráfico:

In [12]:
```
%matplotlib inline

# Respuesta
```