Centro Universitario UAEM Zumpango.

Ingeniería en computación.

Algoritmos Genéticos

Laboratorio

Profesor: Dr. Asdrúbal López Chau

Alumno: Axel Valenzuela Juárez

Fecha: 16/05/2020

Conceptos de AG.

Técnicas Clásicas de Optimización

Existen diferentes técnicas para resolver problemas con diferentes características, por eso es importante saber la existencia de estas técnicas. A continuación, se describen brevemente algunas técnicas de optimización.

Método del Descenso Empinado

La idea del método es comenzar de un cierto punto cualquier X1 y luego moverse a lo largo
de las direcciones de descenso más empinado hasta encontrar el óptimo.

Método de Fletcher-Reeves (Gradiente Conjugado)

Un método para resolver sistemas de ecuaciones lineales derivadas de las condiciones estacionarias de una cuadrática.

Cromosoma: Denominamos cromosoma a una estructura de datos que contiene una cadena de parámetros de diseño o genes.

Gen: Se llama gene a una subsección de un cromosoma que (usualmente) codifica

el valor de un solo parámetro.

Genotipo: Se denomina genotipo a la codificación (por ejemplo, binaria) de los parámetros que representan una solución del problema a resolverse.

Fenotipo: Se denomina fenotipo a la decodificación del cromosoma.

Individuo: Se denomina individuo a un solo miembro de la población de soluciones potenciales a un problema.

Aptitud: Se denomina aptitud al valor que se asigna a cada individuo y que indica que tan bueno es este con respecto a los demás para la solución de un problema.

Generación: Llamamos generación a una interacción de la medida de aptitud y a la creación de una nueva población por medio de operadores de reproducción.

Migración: Se llama migración a la transferencia de (los genes de) un individuo de una subpoblación a otra.

Población panmıtica: cualquier individuo puede reproducirse con otro con una probabilidad que depende solo de su aptitud.

Epístasis: Se llama epístasis a la interacción entre los diferentes genes de un cromosoma.

Existen diferentes tipos de operadores de reproducción:

- **Cruza**. Un operador que forma un nuevo cromosoma combinando los cromosomas de los padres.
- **Mutación**. Operador que forma un nuevo cromosoma a través de alteraciones pequeñas del gen.
- **Reordenamiento**. El operador cambia el orden de los genes en el cromosoma.

Elitismo: asegura que la aptitud máxima de la población nunca se reducirá de una generación a la siguiente

Funcionamiento general de un AG

El funcionamiento de un algoritmo genético se basa en el principio de la supervivencia del más apto.

Se necesitan de 5 componentes para implementar un Algoritmo Genético.

1. Una representación de soluciones potenciales al problema.
2. Una forma de crear una población inicial de soluciones potenciales.
3. Una función de evaluación que juega el papel del ambiente, calificando en términos de aptitud.
4. Operadores genéticos que alteran la composición de los descendientes como cruza o mutación.
5. Valores para los diversos parámetros utilizados por el algoritmo genético.

Operadores genéticos: tipos, funcionamiento de cada uno de ellos.

Técnicas de selección

Se pueden clasificar en tres grandes grupos.

Selección proporcional: se eligen individuos de acuerdo con su contribución de aptitud con respecto al total de la población.

Selección mediante torneo: La selección mediante torneo es similar a la de jerarquías en términos de la presión de selección, pero es computacionalmente más adecuada para implementarse en paralelo.

Selección de estado uniforme: se usa en algoritmos genéticos no generacionales, unos cuantos individuos son remplazados en cada generación.

Implementación

decidí hacer una implementación del algoritmo del viajante esta vez aplicado a una colonia de hormigas como ejemplo, cabe destacar que el tutorial para poder implementar este algoritmo lo eh obtenido del siguiente repositorio.

https://github.com/AeroPython/Taller-Algoritmos-Geneticos-PyConEs16

Este algoritmo busca obtener la ruta mas corta entre ciudades.

Para este problema en específico se han determinado las siguientes reglas:

1. Se debe visitar cada ciudad exactamente una vez, excepto la inicial en la que estará dos veces (salida y llegada final);
2. Una ciudad distante tiene menor posibilidad de ser elegida (Visibilidad);
3. Cuanto más intenso es el rastro de feromonas de una arista entre dos ciudades, mayor es la probabilidad de que esa arista sea elegida;
4. Después de haber completado su recorrido, la hormiga deposita feromonas en todas las aristas visitadas, mayor cantidad cuanto más pequeña es la distancia total recorrida;
5. Después de cada generación, algunas feromonas son evaporadas.

Como comentarios adicionales puedo destacar lo interesante que es este problema y complejo a la vez, usando hormigas y sus feromonas para obtener el mejor camino conforme pasan generaciones, en el código se puede observar como se trazan las feromonas y el paso de las hormigas por generaciones, para finalizar calculando el mejor camino posible.

Código de Algoritmo genético para las hormigas, así como las feromonas de ellas.

```python
# -*- coding: utf-8 -*-
"""Algoritmo de Colonia de Hormigas
    Universidad Autónoma del Estado de México
    Centro Universitario UAEM ZUMPANGO
    Ingenieria en Computación
    UA Algoritmos Geneticos 2020-A
    Alumno: Axel Valenzuela Juárez
    Profesor: DR. Asdrúbal López Chau
    Descripcion:
        Laboratorio
        Resumen
    Fecha: 16 de Mayo de 2020
@author: Valenzuela"""

import numpy as np
import matplotlib.pyplot as plt

class Mapa:
    '''Este objeto contiene el mapa de las ciudades que se van a visitar,
    con todos los datos necesarios (como las feromonas y distancias entre ciudades)
    y las hormigas que lo recorren'''
    def __init__ (self, n_ciud = 10, mapsize = 10):
        self.n_ciud = n_ciud
        self.mapsize = mapsize
        self.ciudades = mapsize * np.random.rand(n_ciud, 2)
        self.distances = np.zeros([n_ciud, n_ciud])
        for i in range(n_ciud):
            for j in range(i+1, n_ciud):
                vector = self.ciudades[i,:] - self.ciudades[j,:]
                modulo = np.linalg.norm(vector)
                self.distances[i,j] = modulo
                self.distances[j,i] = modulo
        self.feromap = np.zeros_like(self.distances)
        self.feromultiplier = 1
        self.conjunto_analisis = []
        self.pathdata = []
        self.bestpath = [0,1]
        self.bestpathlenght = n_ciud * mapsize

    def show_distances_matrix(self):
        '''Muestra la matriz de distancias en código de colores'''
        plt.matshow(self.distances)
        plt.title('Matriz de distancias entre ciudades')

    def show_feromones_matrix(self):
        '''Muestra la matriz de feromonas en código de colores'''
        plt.matshow(self.feromap)
        plt.title('Matriz de feromonas entre ciudades')

    def draw_distances(self):
        #Dibuja un mapa con las ciudades, unidas entre sí por líneas que son más gruesas
        #cuanto más cercanas son. Es una manera gráfica de comprobar la matriz de distancias.
        plt.figure(None, figsize=(8,8))
        plt.scatter(self.ciudades[:,0], self.ciudades[:,1], s = 100, c = '#5599FF',zorder=2)
        for i in range(self.n_ciud):
            for j in range(i+1, self.n_ciud):
                path = np.zeros([2,2])
                path[0,:] = self.ciudades[i,:]
                path[1,:] = self.ciudades[j,:]
                dist = self.distances[i,j]
                thickness = 7 - dist * (7 / self.mapsize)
                plt.plot(path[:,0], path[:,1],'#88AA22', linewidth=thickness,zorder=1)
        plt.title('Mapa de ciudades con sus distancias')

    def draw_feromones(self, rescale_lines = True):
        #Dibuja un mapa con las ciudades, unidas entre sí por líneas que son más gruesas
        #cuanto más feromonas contiene la ruta que las une. Es una manera gráfica
        #de comprobar la matriz de feromonas.
        plt.figure(None, figsize=(8,8))
        plt.scatter(self.ciudades[:,0], self.ciudades[:,1], s = 100, c = '#5599FF',zorder=2)
        if rescale_lines:
            maxfer = np.max(self.feromap)
        for i in range(self.n_ciud):
            for j in range(i+1, self.n_ciud):
                path = np.zeros([2,2])
                path[0,:] = self.ciudades[i,:]
                path[1,:] = self.ciudades[j,:]
                if rescale_lines:
                    fer = self.feromap[i,j]
                    if maxfer > 0:
                        fer *= 7/maxfer

                else:
```

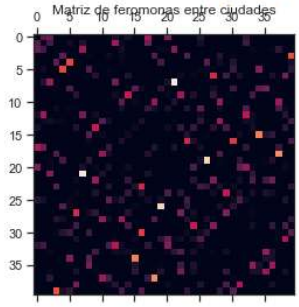
```
82
83                 else:
84                     fer = self.feromap[i,j]
85
86                 plt.plot(path[:,0], path[:,1],'#DD2222', linewidth=fer,zorder=1)
87         plt.title('Mapa de ciudades con sus rastros de feromonas')
88
89     def draw_best_path(self):
90         '''Dibuja un mapa con las ciudades, unidas entre sí por la mejor ruta encontrada hasta el momento.
91         plt.figure(None, figsize=(8,8))
92         plt.scatter(self.ciudades[:,0], self.ciudades[:,1], s = 100, c = '#5599FF',zorder=2)
93         ruta = self.ciudades[[self.bestpath]]
94         plt.plot(ruta[:,0], ruta[:,1],'#2222AA', linewidth=8,zorder=1)
95         plt.title('Mapa de ciudades con mejor ruta encontrada')
96
97     def draw_results(self, relative_scale = False):
98         '''Dibuja la longitud máxima, mínima y media de los caminos que siguen las hormigas,
99         y la longitud mínima que el algoritmo ha encontrado'''
100        plt.figure(None, figsize=(8,5))
101        patharray = np.array(self.pathdata)
102        for i in range(3):
103            plt.plot(patharray[:,i])
104        longx = len(patharray[:,0])
105        plt.plot([0, longx], [self.bestpathlenght, self.bestpathlenght])
106        plt.title('Longitud máxima, mínima, media y mejor camino encontrado')
107        if not relative_scale : plt.ylim(0)
108
109    def draw_best_results(self, relative_scale = False):
110        '''Dibuja la longitud mínima  de los caminos que siguen las hormigas,
111        para todas las veces que el algoritmo se ha ejecutado,
112        y la longitud mínima que el algoritmo ha encontrado'''
113        plt.figure(None, figsize=(8,5))
114        longx = 0
115        for i in range(len(self.conjunto_analisis)):
116            patharray = np.array(self.conjunto_analisis[i])
117            plt.plot(patharray[:,1])
118            longx = max(longx, len(patharray[:,0]))
119
120        patharray = np.array(self.pathdata)
121        longx = max(longx, len(patharray[:,0]))
122        plt.plot(patharray[:,1])
123
124        plt.plot([0, longx], [self.bestpathlenght, self.bestpathlenght])
125        plt.title('Longitud mínima para cada ejecución y mejor camino encontrado')
126        if not relative_scale : plt.ylim(0)
127
128    def swarm_create(self, n_ant = 10):
129        '''Crea una población de hormigas en el mapa'''
130        self.lista_hormigas = []
131        for i in range(n_ant):
132            nueva = Hormiga(self)
133            self.lista_hormigas.append(nueva)
134        del(nueva)
135
136    def swarm_show(self):
137        '''Dibuja un mapa con las ciudades y las hormigas.
138        Es una manera gráfica de comprobar dónde se encuentran.'''
139        plt.figure(None, figsize=(8,8))
140        plt.scatter(self.ciudades[:,0], self.ciudades[:,1], s = 100, c = '#5599FF')
141        ant_pos = np.zeros([len(self.lista_hormigas), 2])
142        for i in range(len(self.lista_hormigas)):
143            hormiga = self.lista_hormigas[i]
144            city = hormiga.position
145            exact_position = self.ciudades[city,:]
146            aprox_position = exact_position + 0.03 *self.mapsize * (np.random.rand(2) - 0.5)
147            #print(exact_position)
148            #print(aprox_position)
149            ant_pos[i,:] = aprox_position
150        plt.scatter(ant_pos[:,0], ant_pos[:,1], s = 5, c = 'k')
151        plt.title('Mapa de ciudades y hormigas')
152
153    def feromone_reset(self):
154        '''Devuelve a 0 el mapa de feromonas para repetir el análisis
155        de un mapa dado.
156        Los datos alcanzados hasta ahora, se guardarán para posterior consulta.'''
157        self.feromap = np.zeros_like(self.distances)
158        self.conjunto_analisis.append(self.pathdata)
159        self.pathdata = []
160
161    def feromone_fine_tune(self):
162        '''Permite controlar en detalle la cantidad de feromonas que se evaporan cada turno
```

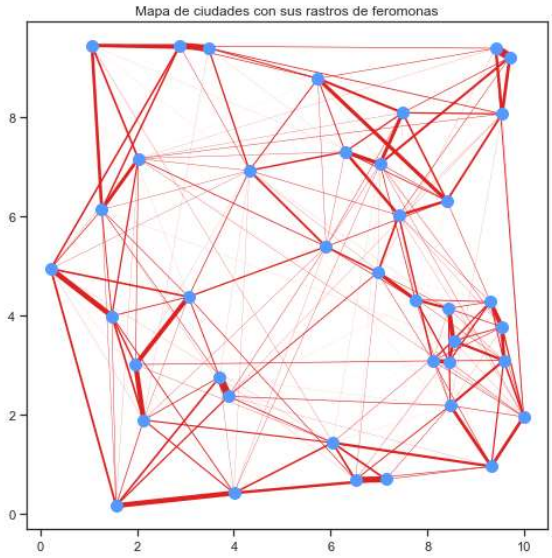
Código de Rutas de hormigas

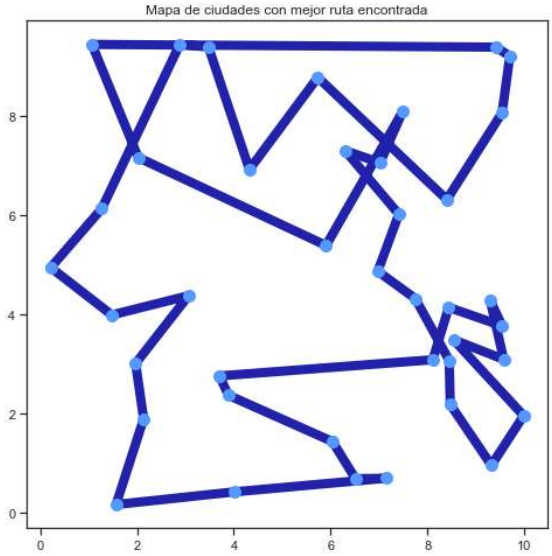
```python
# -*- coding: utf-8 -*-

"""
Algoritmo de Colonia de Hormigas
    Universidad Autónoma del Estado de México
    Centro Universitario UAEM ZUMPANGO
    Ingenieria en Computación
    UA Algoritmos Geneticos 2020-A
    Alumno: Axel Valenzuela Juárez
    Profesor: DR. Asdrúbal López Chau
    Descripcion:
        Laboratorio
        Resumen
    Fecha: 16 de Mayo de 2020
@author: Valenzuela
"""

#paquetes necesarios
import numpy as np
import matplotlib.pyplot as plt
import ants as ants # Aqui están los objetos del algoritmo

#mapa que contiene ciudades
map1 = ants.Mapa(10)
#ver mapa
map1.draw_distances()

map1.swarm_create(100) # Creamos un enjambre de 100 hormigas
#ver hormigas
map1.swarm_show()
#para ver matriz de feromonas y de distancia
map1.show_distances_matrix()
map1.show_feromones_matrix()
#generacion recorre mapa
map1.swarm_generation()
#ver feromonas
map1.show_feromones_matrix()
map1.draw_feromones()
#se hace que multiples generaciones pasen para tener mas datos
for i in range(50):
    print(i, end = '·')
    map1.swarm_generation()
map1.show_feromones_matrix()
map1.draw_feromones()
#se muestra la mejor ruta hasta el momento
map1.draw_best_path()

#se vuelve a crear rutas del algoritmo para ver si cambia
for j in range(3):
    map1.feromone_reset()
    print()
    print('Ejecución', j+1, ', generación: ')
    for i in range(50):
        print(i+1, end = '·')
        map1.swarm_generation()
    map1.draw_feromones()
#se verifica mejor ruta de nuevo
map1.draw_best_path()
#para verificar longitudes minimas
map1.draw_results()
map1.draw_best_results()

# ahora se busca optimizar una ruta entre 40 ciudades
map2 = ants.Mapa(40)
map2.swarm_create(200)
map2.swarm_generation()
map2.show_feromones_matrix()
map2.draw_feromones()

#las feromonas no alcanzan entonces se aumentan
#Con un valor de 5 es suficiente
map2.feromone_fine_tune()

#las hormigas recorren el mapa y se muestran las feromonas
map2.swarm_generation()
map2.show_feromones_matrix()
map2.draw_feromones()

#se hace pasar las generaciones de las hormigas y se terminan mostrando
for i in range(25):
    print(i, end = '·')
    map2.swarm_generation()
map2.show_feromones_matrix()
map2.draw_feromones()
map2.draw_best_path()
map2.draw_results()


```

Resultados

```
In [41]: for i in range(25):
    ...:     print(i, end = '·')
    ...:     map2.swarm_generation()
    ...:
    ...: map2.show_feromones_matrix()
    ...: map2.draw_feromones()
    ...: map2.draw_best_path()
    ...: map2.draw_results()
0·1·2·3·4·5·6·7·8·9·10·11·12·13·14·15·16·17·18·19·20·21·22·23·24·
```







Conclusiones

El uso de AG tiene múltiples usos, un ejemplo es el caso que aquí presente, trazar la mejor ruta posible a partir de múltiples generaciones de hormigas por el paso de sus feromonas. Tal vez este algoritmo no es muy fácil de hacer, pero su utilidad puede ser muy ocupada para trazar rutas del transporte público en una ciudad, los conceptos teóricos que fueron definidos al comienzo de este resumen permiten tener más clara las nociones de cómo funciona un algoritmo genético, así como los pasos o elementos que este debe incluir para ser exitoso.

Bibliografía

ALGORITMOS GENETICOS. (s.f.). Obtenido de http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

Berzal, F. (s.f.). *Algoritmos Genéticos.* Universidad de Granada.

Coello, D. C. (2004). *Introducción a la Computación Evolutiva.* Mexico: CINVESTAV-IPN.