Algoritmos genéticos

1ro Diego Fung
Ingeniería en computación
Instituto Tecnológico de Costa Rica
San José, Costa Rica
2019308467@estudiantec.cr

2ndo Johel Pérez
Ingeniería en computación
Instituto Tecnológico de Costa Rica
San José, Costa Rica
pfjohelu@estudiantec.cr

***Abstract*—In our third and final Analysis of Algorithms project we must create a genetic mutation simulator for flowers and bees, bees must evolve depending on the path that gets them the most flowers in the less amount of distance, on the other hand flowers must evolve depending on the preference of the bees, color, position and such. To accomplish this we must create an algorithm capable of searching for flowers given a bee, and optimize it in order to create a whole generation of bees.**

***Index Terms*—Algoritmo,Python,Mutación Genética, ADN, Algoritmo de Evolución, Flores, Abejas**

I. Introducción

Para nuestro tercer y último proyecto de análisis de algoritmos debíamos de crear una simulación de mutación genética para abejas y flores, las abejas debían de evolucionar para favorecer el camino que les diera más flores en la menor cantidad de distancia y las flores por el otro lado debían de evolucionar dependiendo de las preferencias de las abejas tanto en color como en su posición.

II. Trabajo relacionado

Para el trabajo relacionado consideramos que debíamos investigar de dos cosas relevantes para nuestro proyecto básicamente que es un algoritmo genético y que lo define y además cuál sería la mejor manera de programar la búsqueda, para lo cual concluimos que un grafo sería la mejor manera.

A. Algoritmos genéticos

Buscando en el internet encontramos un papel sin autor simplemente titulado algoritmos genéticos, en el cual nos basamos bastante, este dice que: ”Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes” y además de esto nos da un esquema en pseudocódigo para seguir al crear un Algoritmo Genético el cual es el siguiente:

BEGIN /* Algoritmo Genetico Simple */
Generar una poblacion inicial.
Computar la funcion de evaluacion de cada individuo.
WHILE NOT Terminado DO
BEGIN /* Producir nueva generacion */
FOR Tamano˜ poblacion/2 DO
BEGIN /*Ciclo Reproductivo */
Seleccionar dos individuos de la anterior generacion,
para el cruce (probabilidad de seleccion proporcional
a la funcion de evaluacion del individuo).
Cruzar con cierta probabilidad los dos

individuos obteniendo dos descendientes.

Mutar los dos descendientes con cierta probabilidad.

Computar la funcion de evaluacion de los dos descendientes mutados.

Insertar los dos descendientes mutados en la nueva generacion.

END

IF la poblacion ha convergido THEN

Terminado := TRUE

END

END

B. Grafo

Para el grafo se buscó código para poder implementar la búsqueda de una manera efectiva y óptima debido a la cantidad de abejas que tienen que ser, se concluyo que un grafo sería la mejor manera de lograr esto, el grafo utilizado es una combinación de código encontrado en github y en runestone academy (ver bibliografía). Realmente la investigación de esta parte fue determinar cuál sería la mejor estructura de datos por utilizar, se concluyo que grafo debido a sus búsquedas las cuales en nuestra opinión nos iba a beneficiar más que las búsquedas de árboles.

III. Métodos

Al comenzar a disecar el proyecto nos planteamos la hipótesis de que este podía ser resuelto de la siguiente manera:

- Creación de objetos
- Creación de búsquedas
- Creacion de GUI
- Pruebas

En la sección de métodos hablaremos de los primeros tres y que fue planteado para resolverse y en la sección de análisis de resultados hablaremos de la cuarta la cual son las pruebas.

A. Creación de objetos

Al iniciar el proyecto nos percatamos de que se podía dividir en diversos objetos con atributos para simplificar la creación del programa, abejas, flores, el grafo y la población, esta división fue hecha para que cada uno de los objetos se encargue solamente de lo que tiene que hacer y contener y de esta manera dividir el proceso y no complicarse mucho a largo plazo, la estructura de cada una es la siguiente:

1) Abejas: Las abejas tienen el polen el cual llevan, su color, una identificación, su dirección favorita, su puntaje, sus flores recorridas y su distancia total recorrida, esta se encarga de calcular la dirección válida que la abeja puede tomar en coordenadas y pasar los parámetros para la búsqueda.

2) Flores: Las flores contienen su color, la posición en la cual se encuentran, el polen de las otras flores y un booleano para saber si la flor fue seleccionada o no, la flor se encarga de su proceso de selección solamente.

3) Población: La población ya es un objeto un poco más pesado, ya que este se encarga de tener las flores, las abejas, las direcciones, los posibles colores, la población se encarga de mezclar el ADN, crear flores, crear las abejas, crear las generaciones, etc. básicamente se encarga de todo lo relacionado que afecta tanto a las abejas como a las flores, además de esto también se encarga de hacer las búsquedas e inserciones en el grafo.

4) Grafo: El grafo se encarga de crearlo para poder ser usado en las búsquedas, en la siguiente sección se hablará del razonamiento detrás de las búsquedas a más detalle, el código del grafo básicamente es una combinación de lo visto en el trabajo relacionado.

B. Creación de búsquedas

1) Búsqueda de Anchura: La búsqueda de anchura fue la primera que fue creada esta funciona revisando los vértices

a ver si estos están fuera del radio o no se encuentran en el radio, con esto visita todos los vértices que no cumplen estas dos condiciones, intercambia polen y por último calcula todas las distancias recorridas con distancia entre puntos. Este lo hace agregando a la cola los vértices que tienen conexión con el vértice actual

2) Búsqueda de Profundidad: La búsqueda de profundidad funciona igual en términos de condiciones es esta funciona revisando los vértices a ver si estos están fuera del radio o no se encuentran en el radio, con esto visita todos los vértices que no cumplen estas dos condiciones, intercambia polen y por último calcula todas las distancias recorridas con distancia entre puntos, sin embargo hay una diferencia clave, esta lo hace pasando por todos los vértices de manera recursiva hasta haber pasado por todos.

C. Creacion de GUI

La GUI es muy simple se presenta un botón para generar las generaciones deseadas, dos labels para mostrar las flores y distancia total de una generación y un mapa arriba a la izaquierda de 100x100 el cual nos muestra el campo de flores, con esto se cumplen todas las condiciones necesitadas.

IV. Análisis de resultados

Al revisar algunas pruebas podemos ver inclusive en pocas generaciones que el algoritmo si va mejorando la distancia y la cantidad de flores poco a poco, un ejemplo:

Al hacer 5 generaciones se aprecia que

1 Gen

Distancia total: 353.561210860908

Flores totales: 8

2 Gen

Distancia total: 339.07013138176353

Flores totales: 8

3 Gen

Distancia total: 415.5940094111713

Flores totales: 10

4 Gen

Distancia total: 231.8372713088136

Flores totales: 11

5 Gen

Distancia total: 281.54990928962155

Flores totales: 13

Como se puede apreciar la primera generación es simplemente la primera, la segunda mejora la distancia, la tercera sacrifica distancia por más flores, la cuarta mejora tanto flores como distancia, y la quinta de nuevo sacrifica distancia por más flores, pero todas presentan una mejora, esto podría ser solo coincidencia entonces veamos otro ejemplo:

1 Gen

Distancia total: 282.3115950620419

Flores totales: 13

2 Gen

Distancia total: 305.59822901622834

Flores totales: 13

3 Gen

Distancia total: 414.61411172868833

Flores totales: 16

4 Gen

Distancia total: 387.7085683822044

Flores totales: 16

5 Gen

Distancia total: 387.7085683822044

Flores totales: 16

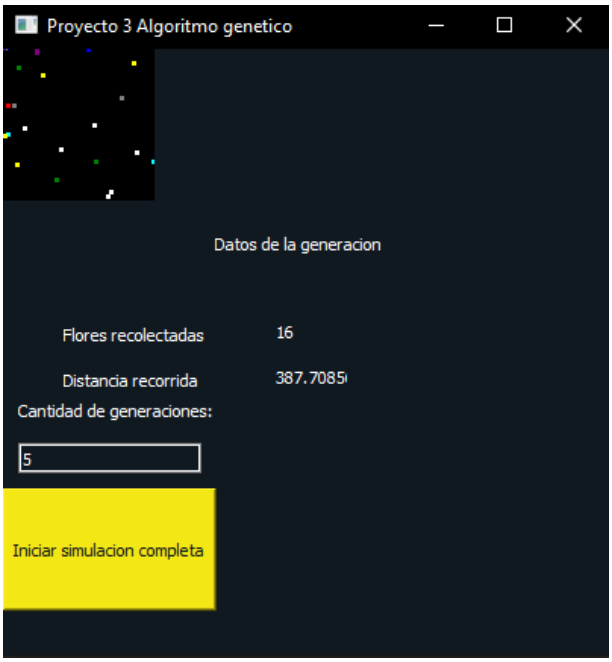
Ahora se ve un panorama un tanto distinto, el algoritmo igual mejora, pero más lento, esto debido a la probabilidad de que salgan mutaciones no favorables como se puede ver en la segunda generación, en la tercera si sacrifica distancia

por flores y en la cuarta mejora la distancia, vemos genes no favorables pero tampoco negativos en la quinta generación.

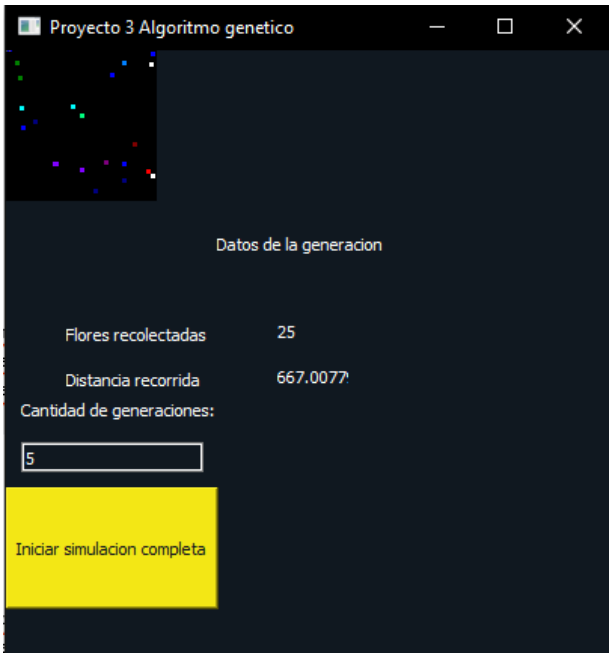
A. *Pruebas*

Pruebas de ejecución:

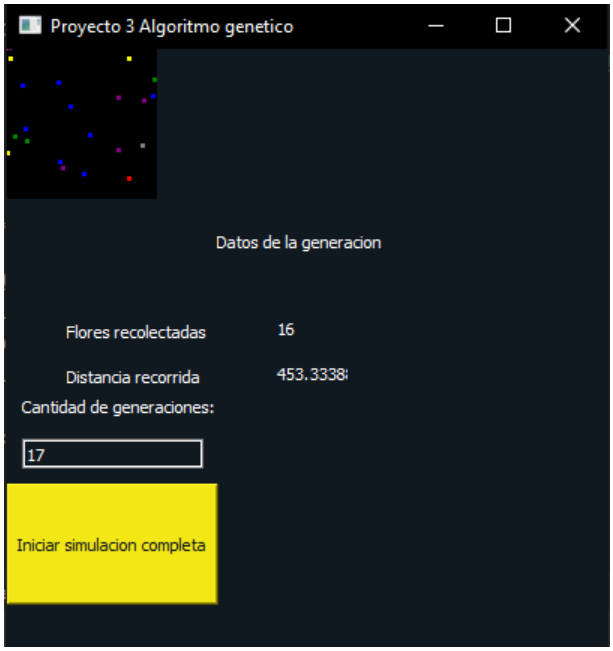
Prueba A



Prueba B



Prueba C



V. Conclusión

Al crear un algoritmo de este tipo mucho debe de ser tomado en cuenta, su complejidad es uno de los aspectos más importantes, ya que si se tiene un algoritmo no optimizado este simplemente no dará la talla al ser puesta con una simulación de esta magnitud, además se tiene que tener cuidado con los cruces para asegurar que el ADN al cual se le da prioridad sea el que da mejores resultados. El modelo utilizado para resolver el problema si dio fruto de una manera simple por lo que se puede decir que es recomendable para resolver un problema de este tipo, sin embargo al final la clave está en la optimización del programa.

References

[1] Solución de problemas con algoritmos y estructuras de datos https://runestone.academy/runestone/static/pythoned/Graphs/Implementacion.html

[2] Recorrido o búsqueda en un Grafo en anchura en Python 3 https://gist.github.com/codigosdeprogra/74154438c7370d551ccd7f6340b78fc8

[3] Algoritmos Genéticos http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf