Ciencia & Desarrollo, Vol. 17, 22 (1) 50 - 56 (2018)
ISSN 2304 – 8891 Versión impresa
Depósito legal N° 2017-16148

OPTIMIZACIÓN DE RUTA CORTA USANDO ALGORITMO GENÉTICO GENERACIONAL

OPTIMIZATION OF A SHORT ROUTE USING GENETIC GENERIC ALGORITHM

[1]Raúl Huarote Zegarra, [2]Yensi Vega Luján, [3]Hilario Aradiel Castañeda, [4]Jhony Valverde Flores

RESUMEN

El presente artículo de investigación tiene como objetivo utilizar algoritmo genético generacional, propio de la inteligencia artificial, donde se aprovecha el proceso evolutivo para optimizar el recorrido de los *n* puntos o nodos, a un coste de computador menor. Una de las ventajas de este método evolutivo es que no compiten todos contra todos, sino que se crea y procesa una porción de la población total, con el objetivo de encontrar la posible mejor ruta, o también llamada búsqueda local. Se ha considerado para el método de la ruleta, cruce por intercambio de 2 puntos, mutación por intercambio y método de parada de acuerdo a la cantidad de generaciones. En un computador de regular característica se logró implementar el algoritmo genético generacional en lenguaje Matlab 8.3, tomando como ejemplo las distancias de cada ciudad. Se obtuvieron como resultados para n puntos o nodos, en cada evaluación, las posibles mejores rutas basándose en el modelo evolutivo. Las pruebas realizadas desde 100 hasta 1000 ciudades resultaron en tiempos de 3,2006 s y 27,924 s, respectivamente. De acuerdo al conjunto de pruebas, demuestra un incremento de manera polinómica de nivel 2; por lo que esta investigación se centra en el incremento de los dos métodos de parada a una secuencia lineal.

Palabras clave: Algoritmo genético, complejidad computacional, ruta, optimización.

ABSTRACT

The objective of this paper is use a generational genetic algorithm, typical of artificial intelligence, where the evolutionary process is used to optimize the path of the n points or nodes, at a lower computer cost. One of the advantages of this evolutionary method is that they do not compete against each other, but that a portion of the total population is created and processed, where their objective is to find the best possible optimal route, or also called local search. It has been considered for the roulette method, crossing by exchange of 2 points, mutation by exchange, stop method according to the number of generations. In a computer of regular characteristic, the generational genetic algorithm was implemented in Matlab 8.3 language and taking as an example the distances of each city, achieving as a result for n points or nodes, in each evaluation the best possible optimal routes based on the evolutionary model. The tests carried out from 100 to 1000 cities resulted in 3,200 seconds and 27,924 seconds respectively. According to the set of tests, it shows an increase in polynomial way of level 2. Finding in this investigation the increase of the two stop methods to a linear sequence.

Keywords: Genetic algorithm, computational complexity, route, optimization.

[1]Escuela de Ingeniería de Sistemas. Universidad César Vallejo. Lima-Perú. E-mail: raulhuarote@ucv.edu.pe
[2]Escuela de Ingeniería Informática. Universidad Nacional de Trujillo. La Libertad-Perú.
[3]Escuela de Ingeniería de Sistemas. Universidad de Ingeniería. Lima - Perú. E-mail: aradielc@hotmail.com
[4]Escuela de Ingeniería Ambiental. Universidad César Vallejo. Lima-Perú. E-mail: jhoval1@yahoo.es

Presentado: 31/03/18 Aprobado: 16/06/18

INTRODUCCIÓN

Encontrar la ruta más corta de un conjunto de n nodos es un problema y otro es cuando se extiende en la cantidad de nodos, pues se requiere hardware de mayor capacidad.

Existen diferentes tipos de algoritmos para resolver problemas de optimización de ruta. Para Díaz (2012), es el problema del agente viajero asimétrico (PVA); o para Hahsler (2007), el del agente viajero (TSP) que debe recorrer todos los nodos y terminar en el mismo nodo de partida, donde se han planteado diferentes métodos tales como el método húngaro, el cual, según Martello (2010), está diseñado para la resolución de problemas de minimización; el método de la fuerza bruta que consiste en explorar todos los recorridos posibles; así también, Micó (1996) menciona que el método del vecino más cercano es un algoritmo heurístico, que aunque no asegura una solución óptima, suele proporcionar buenas soluciones; y el método de Branch and Bound, según Kleinberg (2005) donde a medida que aumente el tamaño de la red, el método puede tardar gran cantidad de tiempo en resolverse, no obstante, para redes de mediano tamaño es una excelente alternativa, considerando que tiene una complejidad computacional de O(n3). El objetivo de este trabajo es usar un método alternativo donde la complejidad es O(n2), tal como nos ofrece el algoritmo genético generacional y representar esto en un lenguaje de programación Matlab 8.3.

INTELIGENCIAARTIFICIAL

Es la manera de emular en lo posible cada una de las capacidades propias del ser humano, y para su implementación se agencia de las tecnologías de información, hardware y software.

Ponce (2010) menciona que si bien es imposible pronosticar con precisión lo que se puede esperar de esta disciplina en el futuro, es evidente que las computadoras con una inteligencia a nivel humano o superior tendrán repercusiones importantes en la vida diaria, así como en el devenir de la civilización.

Por otro lado, Gómez (2013) afirma que la inteligencia artificial puede definirse como "el medio por el cual las computadoras, los robots y otros dispositivos realizan tareas que normalmente requieren de la inteligencia humana".

Así también Romero, Daforte, Gómez y Penousal (2007) sostienen que la inteligencia artificial "es la rama de la ciencia que se encarga del estudio de la inteligencia en elementos artificiales y, desde el punto de vista de la ingeniería, propone la creación de elementos que posean un comportamiento inteligente".

ALGORITMO GENÉTICO GENERACIONAL

Aprovechando la ley de la selva "Sobrevive el más fuerte y adaptado a los embates de la naturaleza", que no es más una representación de Darwin (1859) "supervivencia del más apto", modelo evolutivo que puede utilizarse en la programación, Holland (1992) lo llama "algoritmos genéticos", como una rama de la inteligencia artificial, que no es más que la programación evolutiva.

Goldberg (1989) señala que "los Algoritmos Genéticos son métodos adaptativos, generalmente usados en problemas de búsqueda y optimización de parámetros, basados en la reproducción sexual y en el principio de supervivencia del más apto".

FUNCIONES DEL ALGORITMO GENÉTICO A UTILIZAR

A fin de encontrar la ruta más corta o si se desea optimizar la ruta, se usará el algoritmo genético. Para ello, se considera todas las funciones propias del proceso evolutivo a partir de un cromosoma, fitness, población inicial, evaluación de adaptación (según el fitness de cada cromosoma), selección, cruce, mutación, condición de parada hasta encontrar una solución óptima (Michell, 1997; Eibe, 2010).

A continuación, se muestra cada una de estas funciones:

Cromosoma: Es la representación de una posible solución, donde se considera las distancias de cada uno de los nodos.

Hay que tomar en cuenta que los nodos asignados en un cromosoma se generan de manera aleatoria, "pero no tan aleatorio", esto porque puede aleatoriamente repetir los nodos; por tanto, se le considera nodo no valido, lo que se considera en la creación de cromosoma válido.

Y tiene la siguiente estructura:

Cromosoma=[Nodo1, Nodo2, Nodo3, Nodo4, …, Nodon]
o

Cromosoma = [Ciudad1, Ciudad2, Ciudad3, Ciudad4,…, Ciudadn]

Analizando la estructura del cromosoma, indica que uno de los posibles caminos de ruta óptima es el nodo1, seguido del nodo 2… hasta el nodo *n*. Esto refleja la distancia de nodo1 a nodo 2, más la distancia de nodo 2 a nodo 3, así seguir sumando hasta la distancia del nodo(n-1) hasta nodon.

Fitness: Es el valor de adaptación de cada cromosoma, lo cual permite corroborar qué cromosoma es más apto entre la población. Esto se refleja con la siguiente ecuación:

$$Fitness = 1 - distancia/(máximaDistancia*tamCromosoma)$$

Probabilidad de cruce: Es el valor entre 0 y 1, en el cual se indica la probabilidad de cruce entre los dos padres, si el valor aleatorio que se realiza en ese momento cae dentro del valor de probabilidad de cruce, entonces sí se realiza el cruce; caso contrario, los padres pasan automáticamente a ser hijos.

La figura 1 muestra que sí se va a realizar el cruce (generar 2 hijos), siempre y cuando el valor aleatorio generado en ese momento esté entre 0 y Pc, y no se realiza el cruce (es decir, los padres pasan a ser hijos) si el valor aleatorio actual generado cae entre Pc y 1.

Figura 1. Representación de la probabilidad de cruce.
Fuente: Elaboración propia

Cabe resaltar que el valor de Pc es cerca de 1, para que pueda darse más casos en relación al cruce y buscar además otras posibilidades de valores cercanos al óptimo.

Probabilidad de mutación: Es el valor entre 0 y 1, que indica la probabilidad de la mutación en el cromosoma, si es que el valor aleatorio que se realiza en ese momento cae dentro del valor de probabilidad de mutación, entonces sí se realiza la mutación; caso contrario, el hijo pasa automáticamente a la siguiente generación sin producir variación alguna en su cromosoma.

La figura 2 revela que sí se va a realizar la mutación (alteración en alguno de sus genes del cromosoma), siempre y cuando el valor aleatorio generado en ese momento esté entre 0 y Pm; y no se realiza la mutación (lo que significa que no se alteran los genes del cromosoma) si es que el valor aleatorio actual generado cae entre Pm y 1.

Figura 2. Representación de la probabilidad de mutación.
Fuente: Elaboración propia.

Población inicial: Se considera una población inicial a una porción del total de las posibilidades representados en cromosomas, por tanto, se escoge de manera aleatoria los cromosomas. Se considera la población inicial con el formato de la tabla 1.

Tabla 1. *Representación de la población inicial*

poblacion =	Cromosoma (1)
	Cromosoma (2)
	Cromosoma (3)
	Cromosoma (4)
	……..
	Cromosoma (tamPoblacion-1)
	Cromosoma (tamPoblacion)

Fuente: Elaboración propia.

Evaluación inicial: Es crear la valuación fitness de cada uno de los cromosomas, se puede ordenar desde el más apto de manera descendente, dándole mayor posibilidad de ser escogido, según el criterio de selección. Por tanto, para la representación de la evaluación se considera el fitness en la tabla 2.

Tabla 2. *Representación de la población fitness.*

poblacion =	Cromosoma (1)	Fitness (1)
	Cromosoma (2)	Fitness (2)
	Cromosoma (3)	Fitness (3)
	Cromosoma (4)	Fitness (4)
	……..	Fitness (5)
	Cromosoma (tamPoblacion-1)	Fitness (tamPoblacion-1)
	Cromosoma (tamPoblacion)	Fitness (tamPoblacion)

Fuente: Elaboración propia.

Selección: Es necesario considerar el método de la ruleta para el proceso de selección, de acuerdo al fitness; por tanto, el que tiene mejor *fitness* es quien tendrá mayor posibilidad de ser escogido (figura 3).

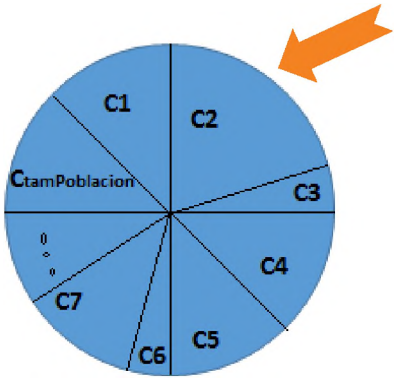

Figura 3. Representación de la ruleta para el proceso de selección de padres.
Fuente: Elaboración propia

Cruce: Una vez seleccionados los padres en una generación determinada, se realiza el cruce, para determinar a los 2 hijos que pasarían a la siguiente generación. Para realizar este proceso se considera el formato del cromosoma, pues tal como está podría generar inconsistencias, específicamente se puede encontrar con 2 ciudades ya visitadas. Por tanto, se debe corregir esas inconsistencias. El proceso de cruce se realiza por medio de 2 puntos (escogidos al azar entre 0 y tamCromosoma), por los cuales se realiza el cruce, a su vez después de realizar el cruce, se tiene que evaluar y corregir si es que hay inconsistencia en el cromosoma resultante.

Mutación: Una vez obtenido el hijo en el proceso de selección, es necesario considerar de acuerdo al valor de probabilidad de mutación (Pm) si es que cada hijo se va a mutar. Que no significa otra cosa que alterar alguno(s) de sus cromosomas. El proceso de mutación se hace por intercambio, ya que se quiere evitar las inconsistencias.

Condición de parada: El proceso evolutivo tiene que realizar una parada, es necesario para darle fin de evaluación del cromosoma más apto. Para ello tenemos 2 métodos posibles: el primero es el método de la varianza, esto quiere decir que si los resultados del mejor individuo (*fitness*) de las m últimas generaciones no varía, entonces nos encontramos en el caso de que el proceso evolutivo ya tiene que hacer una parada, según Cartusia (1998) encontramos la siguiente formula de la varianza:

$$S^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x})^2 \qquad (1)$$

Donde S^2 es la varianza, x_i es el término del conjunto de datos, x es la media de la muestra y m el tamaño de la muestra.

El segundo método se refiere a la cantidad de generaciones, es decir, definir exactamente cuántas generaciones debe procesar el algoritmo genético. Este método ahorra el proceso de evaluación estadística, pues solo se ingresa el número de generaciones. Si bien se ahorra el proceso de evaluar constantemente en caso haya cambios en una porción de un conjunto de datos, no garantiza la cantidad necesaria a procesar; puesto que puede estar evaluando por exceso o por defecto la cantidad de generaciones.

La comparativa de estos métodos se presenta en este gráfico, para los siguientes casos:

DIAGRAMA DEL ALGORITMO GENÉTICO

Para la evaluación del resultado del algoritmo genético que muestre la ruta más corta, es necesario generar un diagrama que permita un orden del proceso evolutivo (figura 4).

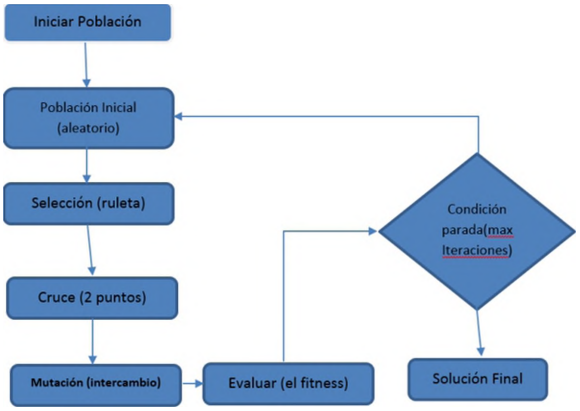

Figura 4. Diagrama del proceso evolutivo para encontrar la ruta más corta.
Fuente: Elaboración propia.

PROTOTIPO DE DESARROLLO

Para la realización del prototipo se han utilizado los scripts y el entorno grafico (GUIDE) del Matlab, tal como se muestra en la figura 5.

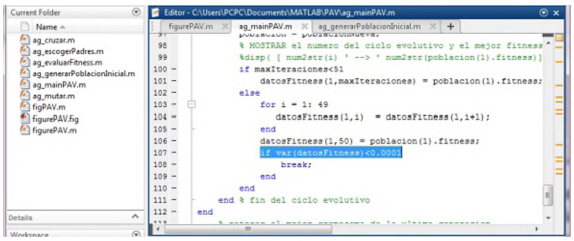

Figura 5. Parte del código fuente donde se realiza la evaluación de la varianza.
Fuente: Elaboración propia.

Para la realización de la matriz de datos, que representa las distancias entre la $ciudad_i$ con la $ciudad_j$, esta tiene que ser simétrica, para mantener la consistencia; la $distancia_i$ a la $distancia_i$ es cero, puesto que está en la misma ciudad:

$$distancia(ciudad_i, ciudad_j) = distancia(ciudad_j, ciudad_i)$$
$$d(ciudad_i, ciudad_i) = 0$$

En la figura 6 se visualiza de manera general cómo se realiza el proceso de encontrar la ruta óptima.s necesarias y demoró en procesarlo 2,4585 s.

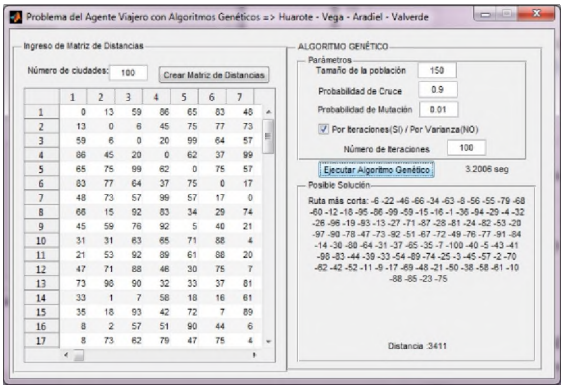

Figura 6. Interfaz gráfica del algoritmo genético implementado en Matlab, definida con 100 iteraciones.
Fuente: Elaboración propia.

En el checkbox "Por Iteraciones(SI)/Por Varianza(NO)", se indica que si el check está marcado, entonces se va a realizar el método de parada por la cantidad de generaciones escrita en la caja de texto. Si el check está desmarcado, implica que va a realizar el método de parada por la varianza de una porción de datos. Cabe mencionar que para este método la porción de datos fue de tamaño 50, implicando que se va a evaluar la varianza con los últimos 50 datos. Para que los valores del *fitness* tengan una variación mínima (casi 0), se puso un umbral de 0,0001 de varianza, tal como muestra la figura 7, dando como resultado 69 iteraciones necesarias y demoró en procesarlo 2,4585 s.

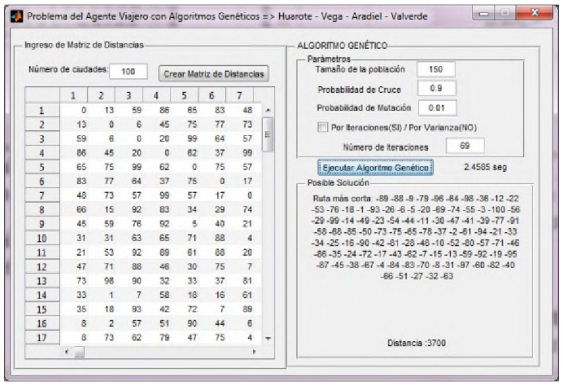

Figura 7. Interfaz gráfica del algoritmo genético implementado en Matlab, sin definir cantidad de iteraciones.

Fuente: Elaboración propia.

El botón "Crear Matriz de Distancias" genera una matriz diagonal de n ciudades (en este caso se prueba con 100), donde se muestra en una tabla los 100 valores de distancia entre ciudad 1 y ciudad 2, etc. En la parte derecha se muestra los parámetros propios del algoritmo genético, en el cual el tamaño de la población es de 150, probabilidad de cruce 0,9, la probabilidad de mutación es de 0,01, el número de iteraciones que para este proceso se está colocando es 100. El botón llamado "Ejecutar Algoritmo Genético" permite todo el proceso evolutivo con los parámetros presentados anteriormente, incluido el método de parada. En la parte derecha de este botón se muestra el tiempo (en segundos) que demora dependiendo de la cantidad de ciudades para evaluar la ruta óptima, en la pantalla se muestra 3,2006 s. En la parte de posible solución, se revela la mejor posible solución en índices de ciudades (o nodos) y la distancia que se va a recorrer según esa secuencia de ciudades. Para este proceso, los valores de los parámetros no necesariamente son estrictos, sino recomendados.

Los datos de las distancias de las ciudades y los parámetros del algoritmo genético van a generar una tabla que muestra el tiempo (en segundos).

Cuánto demora en realizar el proceso evolutivo hasta encontrar la ruta óptima usando algoritmo genético, según detalla la tabla 3 se realizaron las 10 pruebas con nodos de 100, 200...1000, con el fin de determinar el tiempo que demora en encontrar la ruta óptima, y mostrar además el crecimiento que se realiza al cambiar de 100 nodos a 200 nodos, de 200 nodos a 300, y así sucesivamente hasta llegar a 1000 nodos.

Tabla 3. *Tabla de datos de cantidad de ciudades vs de procesamiento con cantidad de iteraciones definidas a 100.*

Prueba	Cantidad Ciudades	Tiempo (Seg)	Crecimiento
1	100	3,2006	-
2	200	3,9982	0,7616
3	300	5,2178	1,2196
4	400	7,0306	1,8128
5	500	9,3941	2,3635
6	600	12,3571	2,963
7	700	15,4821	3,125
8	800	19,1503	3,6682
9	900	23,1621	4,0118
10	1000	27,924	4 7619

Fuente: Elaboración propia.

En función de la tabla 3 se representa el crecimiento de orden 2 para el valor del tiempo que demora en realizar el proceso evolutivo de encontrar la ruta más corta en función del algoritmo genético; según va creciendo la cantidad de ciudades, este crecimiento refleja una parábola tal como se muestra en la figura 8:

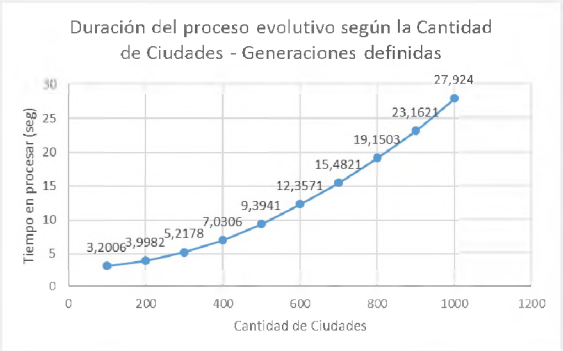

Figura 8. Crecimiento del tiempo de orden 2 para el procesamiento del algoritmo genético – Generaciones definidas.

Fuente: Elaboración propia.

Tabla 4. *Datos de cantidad de ciudades vs tiempo de procesamiento con cantidad de iteraciones sin definir.*

Prueba	Cantidad Ciudades	Tiempo (Seg)	Crecimiento
1	100	2,4585	-
2	200	2,6521	0,3622
3	300	3,481	1,1289
4	400	4,4798	0,9988
5	500	6,1413	1,6615
6	600	7,8533	1,712
7	700	10,1766	2,3233
8	800	12,1011	1,9245
9	900	14,8206	2,7195
10	1000	18,445	3,6244

Fuente: Elaboración propia.

En función de la tabla 4 se representa el crecimiento de orden 2 para el valor del tiempo que demora en realizar el proceso evolutivo de encontrar la ruta más corta en función del algoritmo genético; según va creciendo la cantidad de ciudades, este crecimiento refleja una parábola, tal como se muestra en la figura 9.

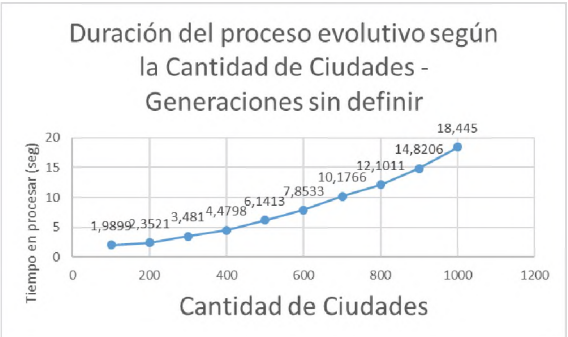

Figura 9. Crecimiento del tiempo de orden 2 para el procesamiento del algoritmo genético – Generaciones sin definir.

Fuente: Elaboración propia.

Tabla 5. *Datos que refleja la diferencia en tiempo de procesamiento con cantidad uno de los métodos de parada.*

Cantidad Ciudades	Tiempo (Seg) – Generación Definida	Tiempo (Seg) - Generación Sin Definir	Diferencia
100	3.2006	2,4585	0,7421
200	3.9982	2,6521	1,3461
300	5.2178	3,481	1,7368
400	7.0306	4,4798	2,5508
500	9.3941	6,1413	3,2528
600	12.3571	7,8533	4,5038
700	15.4821	10,1766	5,3055
800	19.1503	12,1011	7,0492
900	23.1621	14,8206	8,3415
1000	27.924	18,445	9,479

Fuente: Elaboración propia.

En la tabla 5 se representa el crecimiento de orden 1 para la diferencia del tiempo que demoran los dos métodos de parada en realizar el proceso evolutivo de encontrar la ruta más corta en función del algoritmo genético generacional, este crecimiento refleja una línea, tal como se muestra en la figura 10.

Figura 10. Crecimiento del tiempo de orden 1 para el procesamiento del algoritmo genético – entre ambos métodos de parada.

Fuente: Elaboración propia

CÓDIGO FUENTE

Para realizar el algoritmo genético y encontrar la ruta corta entre el conjunto de posibilidades que existe, ha sido necesario realizar la codificación en un lenguaje que permita trabajar con funciones, esto para separar cada una de las funciones propias del algoritmo genético:

Funcion ag_generarPoblacionInicial:

```
function poblacion = ag_generarPoblacionInicial(tamPoblacion )
%AG_GENERARPOBLACIONINICIAL Summary of this function goes here
%   Detailed explanation goes here
    poblacion = struct( 'cromosoma' , [], 'fitness',[]);
    for i=1:tamPoblacion
        % Llenar la poblacion de cromosomas validos
        poblacion(i).cromosoma = generarCromosomaValido();
    end
end
```

Funcion ag_mutar:

```
function hijoMutado = ag_mutar( hijo, probMutacion )
%AG_MUTAR Summary of this function goes here
%   Detailed explanation goes here

    global tamCromosoma;

    % si no hay mutacion, el hijo sale igual
    hijoMutado = hijo;

    %Mutación por intercambio
    if ( rand < probMutacion )
        %primer punto de intercambio
        pto1 = ceil( rand * tamCromosoma );
        %segundo punto de intercambio
        pto2 = ceil( rand * tamCromosoma );


        hijoMutado.cromosoma(pto1) = hijo.cromosoma(pto2);
        hijoMutado.cromosoma(pto2) = hijo.cromosoma(pto1);
    end
end
```

Función ag_cruzar:

```
function [ hijo1 hijo2 ] = ag_cruzar( padre1, padre2, probCruce)
%AG_CRUZAR Summary of this function goes here
%   Detailed explanation goes here

    %declaracion de los hijos como estructuras
    hijo1 = struct( 'cromosoma' , [], 'fitness',[]);
    hijo2 = struct( 'cromosoma' , [], 'fitness',[]);

    global tamCromosoma;
    %cruzamiento con un punto de cruce
    if( rand < probCruce )
        %generacion del punto de cruce
        ptoCruce = ceil( rand * (tamCromosoma-1) );

        %copiar parte del padre1 en el hijo1
        hijo1.cromosoma(1:ptoCruce) = padre1.cromosoma(1:ptoCruce);
        %copiar parte del padre2 en el hijo1
        hijo1.cromosoma(ptoCruce+1:tamCromosoma) = padre2.cromosoma(ptoCruce+1:tamCromosoma);

        %copiar parte del padre2 en el hijo2
        hijo2.cromosoma(1:ptoCruce) = padre2.cromosoma(1:ptoCruce);
        %copiar parte del padre1 en el hijo2
        hijo2.cromosoma(ptoCruce+1:tamCromosoma) = padre1.cromosoma(ptoCruce+1:tamCromosoma);

        %CORREGIR inconsistencias ( cromosomomas no validos )
        hijo1 = corregirInconsistencias( hijo1 );
        hijo2 = corregirInconsistencias( hijo2 );
```

Función ag_escogerPadres:

```
function [ padre1 padre2] = ag_escogerPadres( poblacion )
%AG_ESCOGERPADRES Summary of this function goes here
%   Se utiliza el metodo de la ruleta para escoger los padres a ser
%   cruzados.

    sumFitness = 0;
    [ x tamPoblacion ]= size(poblacion);
    %calculo de la sumatoria de fitness
    for p=1:tamPoblacion
        sumFitness = sumFitness + poblacion(p).fitness;
    end

    %aca se guarda los indices de los 2 padres
    indPadres = zeros(1,2);

    for i = 1:2
        %valor al azar de la ruleta
        ruleta =  round( rand*sumFitness) ;

        % ira sumando los fitness de los individuos hasta encontrarse en el
        % valor de la ruleta
        actualPosicionRuleta = 0;
        j = 1;
        while ( ( ruleta >= actualPosicionRuleta ) && ( j <= tamPoblacion ) )
            %mientras ruleta sea mayor que la actualPosicionRuleta
            actualPosicionRuleta = actualPosicionRuleta + poblacion(j).fitness;
            j = j+1;
        end
```

Función ag_evaluarFitness:

```
function poblacionEvaluada = ag_evaluarFitness( poblacion, matrizDistancias )
%AG_EVALUARFITNESS Summary of this function goes here
%   Calcula el valor de adaptacion (fitness) de cada individuo
%   y posiciona al mejor individuo (Elite) al inicio de la poblacion
    %variables globales necesarias en esta funcion
    global tamCromosoma;
    global maximaDistancia;

    [ x tamPoblacion ] = size(poblacion);
    %calculo del fitness de un cromosoma
    mayorFitness = 0;
    posMayorFitness = 0;

    for p=1:tamPoblacion
        % extraer el cromosoma del individuo p
        cromosoma = poblacion(p).cromosoma;

        %calculo de la distancia de la ruta representada en cromosoma
        % inicialmente la distancia entre la primera y ultima ciudad
        distancia = matrizDistancias( cromosoma(1) , cromosoma(tamCromosoma) );
        %acumular la distancia entre las demas ciudades
        for i = 1:(tamCromosoma-1)
            distancia = distancia + matrizDistancias( cromosoma(i) , cromosoma(i+1) );
        end

        %tamCromosoma = numero de ciudades del problema
        % funcion fitness
        poblacion(p).fitness = 1 - distancia / ( maximaDistancia * tamCromosoma);
```

Función btnAG_Callback

```
figurePAV.m | ag_mainPAV.m | ag_generarPoblacionInicial.m
function btnAG_Callback(hObject, eventdata, handles)
tic;
%capturar los parametros desde los controles

%incremento el tamaño de la poblacion en 1, para el ELITE
tamPoblacion = str2double( get( handles.txtTamPoblacion, 'string' ) ) +1;
probCruce = str2double( get( handles.txtProbCruce, 'string' ));
probMutacion = str2double( get(handles.txtProbMutacion, 'string' ));
maxIteraciones = str2double( get(handles.txtMaxIteraciones, 'string' ));
matrizDistancias = get(handles.tableDistancias,'Data');

metodoParada = get(handles.chkmetodoparada,'Value');%1 por iteracciones / 0 por varianza
%Llamada al Algoritmo Genetico
% retornna el mejor individuo al finalizar el proceso evolutivo del
% Algoritmo Genetico
[posibleSolucion maxIteraciones]= ag_mainPAV(tamPoblacion, matrizDistancias, probCruce, pro

% string para reporte de la Mejor ruta del Individuo ganador
strRuta = ' ';

%sacar el tamaño del cromosoma.
[x tamCromosoma] = size(posibleSolucion.cromosoma);

strRuta = 'Ruta más corta: ';
```

CONCLUSIONES

Bajo el modelo evolutivo, se puede encontrar la ruta más corta de n puntos y no presenta complicaciones por la cantidad de nodos a evaluar, considerando que se llegó a probar con 1000 nodos y demoró 27,924 segundos.

Según la cantidad de ciudades para encontrar la ruta más corta con cantidades de generaciones definidas a 100, la demora en realizar el proceso evolutivo está en función de la siguiente fórmula polinómica de segundo nivel, $y = 2x^2/10^5 + 0{,}0017x + 2{,}7169$. Esta ecuación se genera a partir de los datos de la tabla 3.

Para encontrar la ruta más corta con cantidades de generaciones sin definir, la demora en realizar el proceso evolutivo, está en función de la siguiente fórmula polinómica de segundo nivel: $y = 2x^2/10^5 + 0{,}0008x + 1{,}7269$. Esta ecuación se genera a partir de los datos de la tabla 4.

Se realizó la comparación entre los dos tiempos que dura el proceso en encontrar la ruta más corta usando los 2 métodos de parada y se encontró un diferencial de tipo lineal, y = 0,992x - 1,0251. Esta ecuación se genera a partir de los datos de la tabla 5.

BIBLIOGRAFÍA

Cartusia, L. (1998). *Bioestadística – Métodos y Aplicaciones*. Malaga: U.D. Bioestadística. Facultad de Medicina.

Darwin, C. (1859). *El origen de las especies*. London: College de Cambridge

Díaz, J. (2012). *Algoritmos para la optimización de rutas*. España: Eae Editorial Academia Espanola.

Eibe, A. & Smith, J. (2010). *Introduction to Evolutianary Computing*. United Satate: Springer. Cap. 3: "Genetic Algorithms".

Golberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. MA: Addison-Wesley Professional.

Gómez, R. (2013). *La inteligencia artificial. ¿Hacia dónde nos lleva? ¿Cómo ves?*. México: UNAM.

Hahsler, M. & Hornik, K. (2007). *TSP – Infrastructure for the Traveling Salesperson Problem*. United State: Journal of Statistical Software 23.

Holland, J. (1992). *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press

Kleinberg, J. & Tardos, E. (2005): *Algorithm Design*. United Satate: Addison-Wesley

Martello, S (2010). From the origins of the Hungarian algorithm to satellite communication. Central European: *Journal of Operations Research*, 18, 47–58.

Matworks. (2018). *Algoritmo genético. Determinación de los mínimos globales para problemas altamente no lineales*. Massachusetts: Matworks Inc. Recuperado de: https://la.mathworks.com/discovery/genetic-algorithm.html.

Micó, M. (1996). *Algoritmos de búsqueda de vecinos más próximos en espacios métricos*. Valencia: Universidad Politécnica.

Mitchell, T. (1997). *Machine Learning*. Illinois: McGraw-Hill. Cap. 9: Genetic Algorithms.

Ponce, P. (2010). *Inteligencia artificial con aplicaciones a la ingeniería*. México: Alfaomega Grupo Editores S.A. de C.V., México.

Romero, J., Daforte, C., Gómez, A., y Penousa, F.J. (2007). *Inteligencia artificial y computación avanzada*. Santiago de Compostela: Fundación Alfredo Brañas.