Algoritmo de plan de corte para un problema de planificación en un espacio con fronteras

Baldion Góngora, Juan Andres[1], Cascavita Ríos Elquin Mauricio [1], Cabal Millán Álvaro José [1]
Jbaldion06@unisalle.edu.co, ecascavita17@unisalle.edu.co, acabal18@unisalle.edu.co.
Rodríguez, Cesar [2]

***Resumen*— La planificación del diseño del sitio es un procedimiento imperativo que puede afectar significativamente la productividad y la eficiencia de las operaciones logísticas realizadas en un sitio de construcción. El proyecto consiste en implementar un algoritmo genético completo, para obtener la manera más adecuada de ubicar un número de elementos en zonas específicas en un campo de construcción.**

La idea es demostrar que, al ejecutar el algoritmo genético, a medida que las generaciones avanzan, se van obteniendo soluciones óptimas que pueden llegar a ser la solución definitiva del problema y de esta manera se puede escoger por medio del algoritmo, cuál serán las mejores maneras de ubicar los elementos y zonas temporales de un campo de construcción. Esto permitirá tener muchas opciones incluyendo la más adecuada donde aproveche más el espacio.

Finalmente, se obtienen diferentes maneras de ubicación de estos elementos en un espacio definido, mostrando siempre la opción más óptima para que estos lugares queden bien ubicados y así no desperdiciar espacio.

Palabras Clave*— *Genetic Algorithm, Selection, Crossing, Mutation, Initial Population, Best Individuals, Generation, Fitness, Roulette, construction site.

Site design planning is an imperative procedure that can significantly affect the productivity and efficiency of logistic operations performed at a construction site. The project consists of implementing a complete genetic algorithm, to obtain the most appropriate way to locate a number of elements in specific areas in a construction field.

The idea is to demonstrate that, when executing the genetic algorithm, as the generations advance, optimal solutions are obtained that can become the definitive solution of the problem and in this way can be chosen by means of the algorithm, which will be the best ways to locate the elements and temporary zones of a construction field. This will allow you to have many options including the most appropriate where you can take advantage of the space.

Finally, different ways of locating these elements in a defined space are obtained, always showing the most optimal option so that these places are well located and thus not waste space.

Keywords*— *Genetic Algorithm, Selection, Crossing, Mutation, Initial Population, Best Individuals, Generation, Fitness, Roulette, construction site.

I. INTRODUCCION

Los Algoritmos Genéticos (AGs) son métodos que se aplican para resolver problemas donde se tenga que ver la optimización de un proceso, dado que este algoritmo se basa en la búsqueda de encontrar la mejor solución cada vez que itera, su principal fundamento para el desarrollo del algoritmo está concentrado en el proceso genético de los organismos vivos, de aquí su nombre.

Teniendo en cuenta este fundamento el algoritmo es capaz de crear soluciones para problemas del mundo real, dado que tiene como iniciación la naturaleza de los seres vivos ya que tienen padres, hijos y se dice que la evolución en la siguiente descendencia será mejor que la del progenitor, por ello cada vez que genera un nuevo hijo es una mejor solución que la anterior.

Por otra parte, se conoce que en la naturaleza algunos individuos mutan por causa de terceros, por ello en este

[1] Estudiantes de Ingeniería en Automatización.
[2] Docente de Ingeniería en Unisalle.

algoritmo también tiene esta habilidad, pero teniendo la debilidad de no saber si la nueva creación será una mejor que la anterior.

El costo total de transporte, medido en términos de frecuencias de viaje ponderadas por distancia entre las instalaciones, se usa comúnmente como un criterio para evaluar la idoneidad del diseño de la obra. Se utiliza un Algoritmo genético para poder encontrar la forma mas eficiente de ubicar cada elemento en zonas definidas para lograr ganar tiempo y recorridos más precisos y así no desperdiciar espacios los cuales evitaran recorridos excesivos y gastos innecesarios.

II. ASPECTOS TEORICOS

Como tal los algoritmos genéticos se basan en resolver problemas, permitiendo utilizar diferentes tipos de herramientas que tienen en cuenta proceso genético, mismas que proveen elementos fundamentales en la solución del problema.

Se puede decir que los AGs pueden ser aplicados a cualquier campo donde se quiera optimizar una solución dado que muestra la solución de la solución por medio de sus descendientes, a continuación se presentas algunos conceptos básicos que se tuvieron en cuenta para resolver el problema:

El Algoritmo Genético Simple

Primero se tiene el seudocódigo para poder entender sobre el algoritmo en que se va a trabajar:

BEGIN /* Algoritmo Genético Simple */
Generar una población inicial.
Computar la función de evaluación de cada individuo.
WHILE NOT Terminado ***DO***
BEGIN /* Producir nueva generación */
FOR Tamaño˜ población/2 DO
BEGIN /*Ciclo Reproductivo */

Seleccionar dos individuos de la anterior generación, para el cruce (probabilidad de selección proporcional a la función de evaluación del individuo). Cruzar con cierta probabilidad los dos individuos obteniendo dos descendientes. Mutar los dos descendientes con cierta probabilidad. Computar la función de evaluación de los dos descendientes mutados. Insertar los dos descendientes mutados en la nueva generación.

END
IF la población ha convergido ***THEN***
Terminado:= ***TRUE***
END
END

Población

Para este algoritmo es importante el tamaño de la población dado que el trabajar con poblaciones pequeñas no pueden cubrir con el espacio de búsqueda que se desea, mientras que cuando son grandes poblaciones tiene un costo computacional excesivo por otra parte, se tiene la población inicial que se dice que es tomada aleatoriamente de la población total y es la que se le van a generar todo tipo generaciones para encontrar el mejor hijo con las características necesarias para satisfacer el problema

Selección

En esta sección del algoritmo se basa en la selección de individuos con las mejores características para abstraerlos en una nueva población, esto se hace para encontrar cada vez que se genere una nueva generación de individuos se selecciones todos aquellos con las mejores aptitudes, otr4os métodos califican solo a una muestra aleatoria de la población, aunque esto puede tomar demasiando tiempo.

Se utilizan diferentes métodos, pero en todos se pone la prioridad del individuo que tenga mejor aptitud para ser seleccionado.

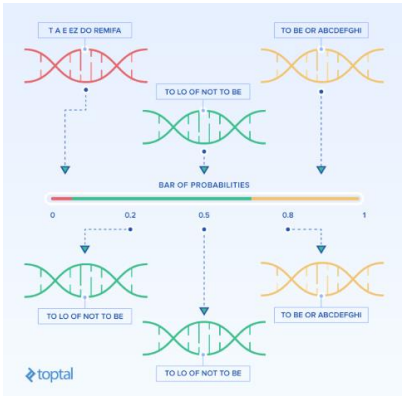

Fig. 1. Selección por probabilidad del mejor individuo.

Cruce

Después de hacer la selección de los individuos se procede hacer el cruce que se basa en escoger diferentes partes de

cada individuo y mezclarlas para generar un nuevo individuo, estas partes se hacen teniendo en cuenta métodos de cruce, de esta manera intercambian secciones que ya están predeterminadas, según investigaciones el mejor cruce es el de dos puntos en un individuo para cruzarlo con otro individuo con la misma situación dado que radica en que el espacio de búsqueda puede ser explorado más fácilmente.

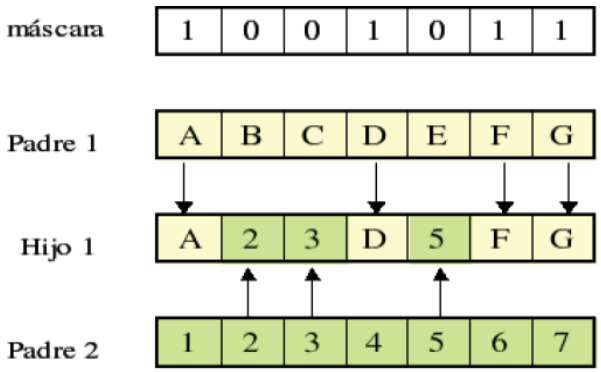

Fig. 2. Cruce de dos padres para generar un hijo.

Mutación

La mutación se considera un operador básico que proporciona una aleatoriedad que ayuda a mejorar la reubicación de los segmentos de los fenotipos, esta permite crear una mejor diversificación de los individuos cuando el algoritmo no encuentra otra solución.

La mutación se puede deducir que puede llegar a ser más importante que el cruce en determinados procesos cuando se busca en tener un valor optimo en la solución, dado que provee descendencia que no se esperaba en la evolución de individuo.

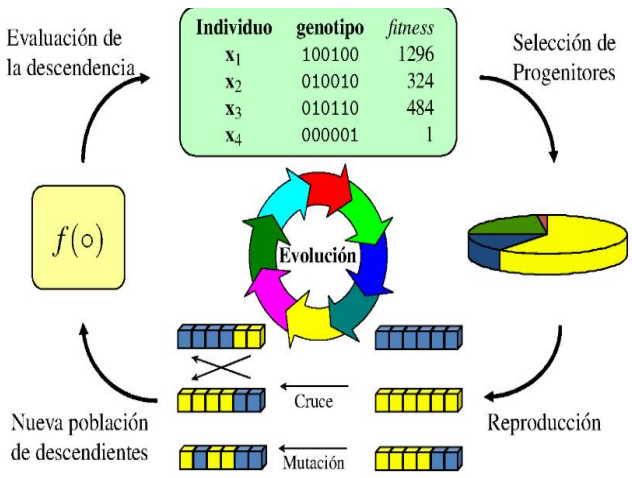

Fig. 3. Muestra de la evolución de una población.

III. Elementos Requeridos

Esta práctica no requiere de elementos físicos como tal, puesto que aparte de los cálculos teóricos solo se requiere el software MATLAB para la realización del algoritmo.

La lista de materiales requeridos para la implementación de esta práctica se mostrará en la Tabla 1.

TABLA I
ELEMENTOS REQUERIDOS PARA LA PRÁCTICA

Elemento	Cantidad
PC	1
Software MATLAB	1

IV. Aspectos Experimentales y Metodologia

Para este proyecto se realizó la implementación de un algoritmo genético para encontrar la mejor forma de ubicar elementos temporales en unas zonas específicas de un proyecto de construcción, de tal manera que el desperdicio de espacio fuera el mínimo.

Para realizar el programa, este fue necesario abordarlo por partes, las cuales se explicarán a continuación de forma detallada.

1. Especificaciones del problema

Esta sección es el inicio y se relaciona con definir cuál es la dimensión del terreno en el que se realizará la construcción. Se introducen las medidas en metros y se especifica un color deseado para el terreno.

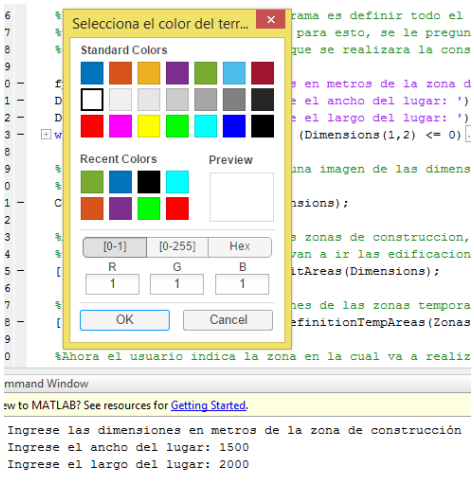

Fig. 4. Definición de medidas del terreno y del color.

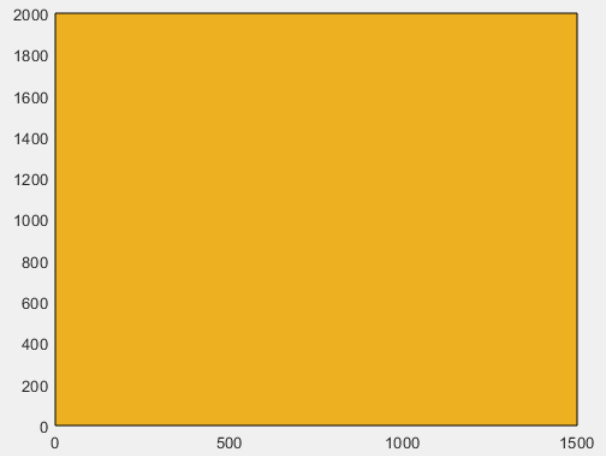

Fig. 5. Terreno obtenido.

Posteriormente, se indica el número de edificaciones que van a ser fijas. Esto sucede porque el problema escogido muestra que se tienen las zonas de construcción fijas y van a haber unas zonas que van a ser temporales en la construcción.

Al introducir el número de construcciones, se irá preguntando en cada una de ellas sus dimensiones, es decir, el largo y el ancho de la construcción, luego se indicará su color y finalmente la posición en el terreno que va a ser colocada esa construcción. Esto se hace dando clic derecho en el terreno, las cuales serán las coordenadas del centro de las construcción.

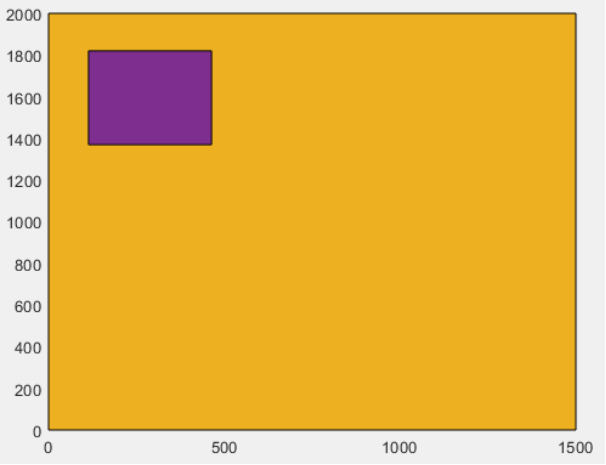

Fig. 6. Construcción puesta en el terreno.

Luego, se le pregunta al usuario si desea incluir piezas adicionales en el terreno creado, estas piezas, al ser de una construcción, pueden indicar tanques, estructuras, tejados o demás. El usuario tiene la opción de incluir círculos, elipses o triángulos, indicando las dimensiones y las coordenadas donde irán estas figuras en el terreno.

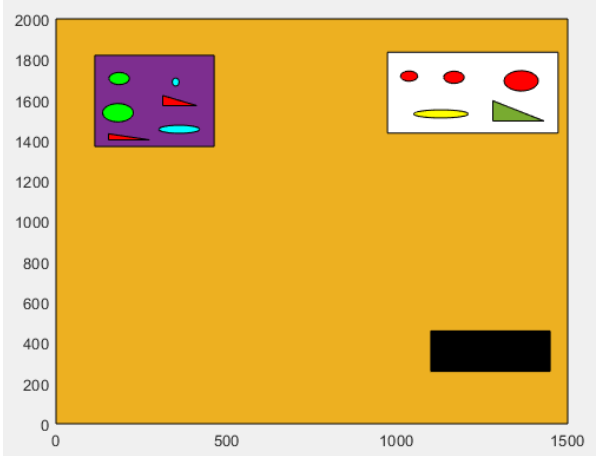

Fig. 7. Varias piezas insertadas en algunas zonas de construcción.

Al ingresar todas estas figuras, el usuario va a poder indicar la cantidad de elementos y las dimensiones que van a ir en las zonas temporales. Estos elementos son aquellos que se colocan en las zonas aledañas de una construcción y son los baños, la zona de máquinas, la zona de descarga y la zona de mezcla.

Terminada la sección anterior, el usuario señala en el mapa la zona en la que estarán estos elementos y aquí es donde empieza como tal el algoritmo genético, pues luego de ingresar la zona temporal, el programa calculará y organizará las piezas de la mejor forma para poder organizar las piezas.

2. Definición de condiciones iniciales.

Esta etapa consiste en definir los parámetros del algoritmo genético como lo son el número de individuos y el número de generaciones. Luego el programa va a organizar en una matriz las zonas que se seleccionaron con sus dimensiones correspondientes y otra matriz que indica el número de zonas de cada tipo.

A continuación, la sección de código que ejecuta lo mencionado anteriormente.

```
%Ahora el usuario indica la zona en la cual va a realizar la construccion y
%las dimensiones de la zona en donde posiciona las zonas temporales. Se
%realiza una validacion de que esas zonas quepan en la zona dada por el
%usuario
[Alto, Ancho, xTA, yTA] = ValidTempAreas(ZonasTemporales, Quantity);

%Se define el area en el que las zonas temporales son puestas y se inicia
%el algoritmo genetico, indicando el numero de individuos y generaciones.
SizeTempAreas=[Ancho Alto];
n_ind=16;
n_gen=16;
best_Ind = [];
```

Fig. 8. Definición de las condiciones iniciales.

3. Generar población inicial.

En esta sección del algoritmo va a colocar todas las zonas que deben ser agregadas sin importar que tanto se está aprovechando o no la superficie del terreno.

En primer lugar se genera un plan de corte el cual va a convertir la orden solicitada en un conjunto de pasos donde especifica el tipo de zona y sus dimensiones.

Para empezar, se coloca la primera zona desde la posición que el usuario haya indicado. Posteriormente se ejecuta un For del tamaño del plan de corte en el cual se van a ir agregando una por una todas las zonas de tal forma que estas queden colocadas juntas, sin sobreponerse, pero sin aprovechar efectivamente las dimensiones del terreno.

Para colocar las zonas, se toman las coordenadas que genera el poner una zona, ya que como las zonas son rectangulares, siempre habrá 3 nuevos puntos posibles para poner la siguiente zona, en la orientación que esta tenga o rotándola, esto para que las zonas puedan ser colocadas una partiendo de la otra.

```
function [cuts,cp] = GenCuttingPlan(parts,order,size_roll,posx,posy)
%Convertir la orden a un plan de cortes
cuts=Order2cuts(parts,order);
%Posicion inicial para colocar la primera pieza
pos_likely=[posx,posy];
%Plan de corte
cp=[];
%Id del primer corte
cut_id=1;
%For que barre todas las piezas de la orden
for i=1:size(cuts,1)
    %Corresponde a las dimensiones de la pieza a cortar
    cut_info=cuts(i,2:4);
    %Seccion que va a intercambiar el largo por el ancho, es como si se
    %rotara la pieza
    if rand()<0.5
        cut_info(2:3)=cuts(i,4:-1:3);
    end
    %La direccion del corte va a ir en 1 de 4 posibles, esta se selecciona
    %aleatoriamente
    loc_dir=randperm(4);
    out=1;
    %Se selecciona el cuadrante que haya quedado en primera posicion
    k=1;
    while out
        for j=1:size(pos_likely,1)
            [out,part]=Check_inter(cut_info,pos_likely(j,:),cp,size_roll,loc_dir(1,k));
            if ~out
                break;
            end
        end
        k=k+1;
    end
    k=k-1;
```

Fig. 9. Código utilizado para la generación de la población inicial Parte 1.

```
    %Se crea el plan de corte indicando el numero de corte, el tipo de pieza
    %y las coordenadas en las que quedo esa pieza
    cp=[cp;[cut_id part.id part.A part.B part.C part.D cut_info loc_dir(1,k)]];
    %Se incrementa el id de pieza para posicionar la siguiente
    cut_id=cut_id+1;
    %Las posiciones para poner nuevas piezas ahora son los 4 extremos de la
    %pieza
    pos_likely=[pos_likely;part.B;part.C;part.D];
    %Quita la primera pieza
    if j==1
        pos_likely=pos_likely(2:end,:);
    else
        pos_likely=[pos_likely(1:j-1,:);pos_likely(j+1:end,:)];
    end
    pos_likely=pos_likely(randperm(size(pos_likely,1)),:);
%    clf
%    DrawCutingPlan(cp,size_roll);
end
```

Fig. 10. código utilizado para la generación de la población inicial Parte 2.

Para garantizar que todas las zonas puedan ser colocadas, se ejecuta un while que evalúa todos los puntos posibles para agregar una zona, si no es posible, entonces se realizan rotaciones de la zona hasta que cada una pueda ser colocada sobre el terreno.

4. Heurística, Fitness y generación de rangos.

Luego de tener una población inicial generada de n número de individuos, se procede a evaluar que tan buenos son o no estos individuos. Para esto se genera un criterio llamado Heurística que nos permite conocer si un individuo es bueno o no.

El criterio de heurística seleccionado fue el siguiente. Todas las zonas son colocadas es un espacio dado por el punto inicial hasta un punto final máximo en x y otro en y.

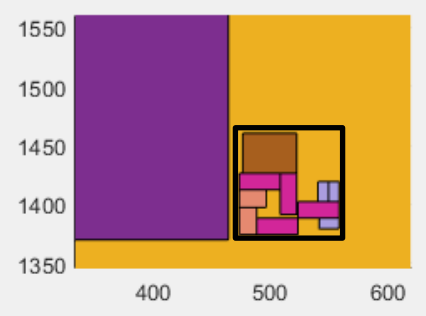

Fig. 11. Explicación del Fitness escogido.

El área negra de la figura 11 muestra que todas las zonas son situadas dentro de un área. Para evaluar que tan bueno es un individuo o no, se implementa la siguiente formula:

$$Heuristica = Area\ Ubicacion\ Piezas - Area\ de\ todas\ las\ piezas$$

Implementando la ecuación, se obtiene una heurística que dice que tan bueno es un individuo o no. Luego, se organizan a todos los individuos por ese valor de Heurística del menor a mayor, obteniendo lo siguiente:

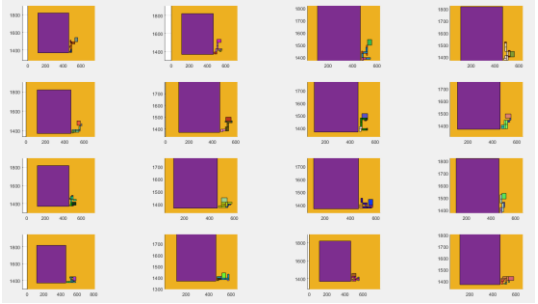

Fig. 12. Población inicial organizada de menor a mayor heurística.

Como se ve en la figura anterior, los individuos están ordenados de izquierda a derecha y de arriba a abajo, por lo tanto, los individuos de la primera fila son los peores ya que son los que más desperdicio de terreno tienen, mientras que los individuos ubicados en la última fila son aquellos que tienen un desperdicio mínimo de terreno.

Utilizando este criterio, se procede a calcula el Fitness. Para el Fitness se utiliza la siguiente ecuación:

$$f(n) = \frac{Heuristica\ (n)}{\sum_{i=1}^{n} Heuristica\ (n)} * 100$$

Esta fórmula muestra que para calcular el Fitness de un individuo n, se toma el valor de heurística que tenga asignado y se divide sobre la heurística total, es decir, la suma de todas las heurísticas. Luego, se multiplica por 100 para tener valores mayores a 0.

Al implementar esta fórmula, se obtiene que el Fitness es muy similar para todos los individuos, es decir, el mejor individuo tiene una diferencia pequeña en Fitness con respecto al peor individuo, por lo que, si se genera un valor aleatorio para escoger individuos (Ruleta) casi cualquier individuo tendría la misma probabilidad de ser escogido.

Para evitar el problema mencionado anteriormente y dar una mayor probabilidad de ser escogido al mejor individuo, se realiza una distribución lineal de probabilidad. Esta distribución se hace cuando se quiere dar prioridad a un valor con respecto a otros, cuando estos son muy similares entre sí, que es tal cual lo que sucede en nuestro caso.

La distribución lineal de probabilidad va a generar una mayor probabilidad al mejor individuo y una menor probabilidad al peor individuo. Para realizar esto, se utiliza la fórmula de la distribución lineal de probabilidad:

$$Probabilidad\ (n) = \frac{2 * (Fitness\ (n) - A)}{(B - A)^2}$$

Donde la probabilidad para un individuo n está dada por el Fitness de ese individuo, A que es el Fitness del peor individuo y B que es el Fitness del mejor individuo. Implementando esta ecuación, se asignan valores de probabilidad acordes al Fitness del individuo.

Al tener todas las probabilidades de cada individuo, se realiza un ajuste para la suma de todas las probabilidades sea 100% y tras realizar esto se asignan los rangos.

El rango de un individuo está dado por:

```
Rango = [probAcum-probabi+0.1 probAcum]
```

Donde Probabi es la probabilidad de ese individuo y probAcum es la suma de las probabilidades de los individuos anteriores a ese individuo.

De esta forma, cada individuo tiene su heurística, su Fitness y su respectivo rango de probabilidad para ser escogido.

Fields	gen	heuristica	fitness	probability	range
1	10x14 double	0.7939	5.5584	0	[100.0001 100]
2	10x14 double	0.8167	5.7182	1.4439	[98.5562 100]
3	10x14 double	0.8196	5.7382	1.6250	[96.9311 98.5561]
4	10x14 double	0.8334	5.8350	2.4993	[94.4318 96.9310]
5	10x14 double	0.8363	5.8551	2.6816	[91.7502 94.4317]
6	10x14 double	0.8575	6.0038	4.0248	[87.7254 91.7501]
7	10x14 double	0.9070	6.3503	7.1561	[80.5693 87.7253]
8	10x14 double	0.9082	6.3589	7.2341	[73.3352 80.5692]
9	10x14 double	0.9275	6.4943	8.4574	[64.8778 73.3351]
10	10x14 double	0.9284	6.5000	8.5089	[56.3689 64.8777]
11	10x14 double	0.9339	6.5388	8.8603	[47.5086 56.3688]
12	10x14 double	0.9372	6.5618	9.0678	[38.4408 47.5085]
13	10x14 double	0.9392	6.5759	9.1949	[29.2459 38.4407]
14	10x14 double	0.9453	6.6185	9.5799	[19.6660 29.2458]
15	10x14 double	0.9484	6.6404	9.7778	[9.8882 19.6659]
16	10x14 double	0.9502	6.6526	9.8881	[1.0000e-04 9.8881]

Fig. 13. Población inicial con la especificación del genotipo, heurística, Fitness y rango de cada individuo.

5. Selección.

Consiste en escoger individuos para que estos sean los nuevos padres que den como fruto hijos y nuevas generaciones.

El proceso de selección se puede hacer de muchas maneras, una de ellas es el elitismo que consiste en escoger únicamente a los mejores individuos, sin tener en cuenta a los demás, pero como en este caso los Fitness son muy similares entre si entonces el proceso de selección se hace totalmente aleatorio.

El proceso de selección escogido fue por ruleta. Por este método es como tener una ruleta de un casino, donde existen huecos de tamaño similar y una bolita que gira por la ruleta, esta al perder velocidad y va a caer en uno de estos huecos. El proceso en la selección es similar solo que ahora el tamaño de los huecos varía según el Fitness del individuo, esto es para que los mejores individuos tengan más probabilidad de ser escogidos.

Para realizar la ruleta en MATLAB se genera un número aleatorio y el valor obtenido se compara con los rangos de cada individuo. El número solo va a poder estar dentro de un rango,

por lo que solo se va a llamar a un individuo cada vez, pero es posible que se llame varias veces al mismo individuo en otros giros de la ruleta, aunque eso no afecta.

La siguiente figura muestra el código utilizado para realizar todo lo explicado anteriormente:

```
function pob=Seleccion(pob)

pobSel = [];

for i=1:size(pob)
    SelNum=rand*100;
    %Seleccion de los individuos por su porcentaje en la ruleta
    for j=1:size(pob)
    Rangi = pob(j).range;
    if SelNum >= Rangi(1,1) && SelNum <= Rangi(1,2)
        %Ahora todos los individuos tienen la misma probabilidad de ser
        %escogidos, ya no hay criterio de seleccion
        ind=GenInd(pob(j).gen,0,0,0,0);
        pobSel = [pobSel;ind];
    end
    end
end
pob=pobSel;
end
```

Fig. 14. Código utilizado para la selección.

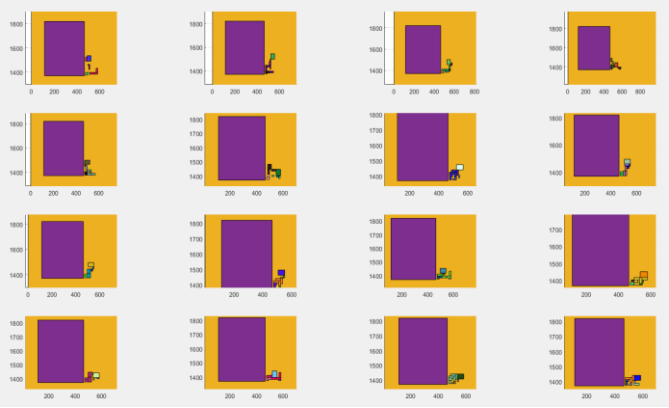

Fig. 15. Población seleccionada.

6. Cruce.

Consiste en combinar características de los dos padres para aplicarlas sobre sus hijos. En este concepto de cruce se tienen dos padres que van a generar dos hijos.

Para mostrar a modo de ejemplo, se dejó que el número de individuos fuera de 16. Por lo tanto, la población inicial fue de 16 individuos y en la selección se seleccionaron 16 individuos. Para realizar el cruce se hace el siguiente procedimiento:

1. Escoger pares de padres.

Para esto, como se tienen 16 individuos entonces los padres se van a escoger de a pares pero cercanos, es decir, dos padres van a ser el individuo 1 y el, otros dos padres van a ser el individuo 3 y 4 y así sucesivamente hasta completar los 16 individuos los cuales van a generar 16 hijos, ya que cada pareja de padres solo genera dos hijos.

2. Seleccionar un punto de cruce.

El tipo de cruce es mono punto, es decir, solo se va a seleccionar un punto para cruzar, Para esto entonces se genera un número aleatorio de 0 a size que es el número de zonas especificado. Según el valor obtenido, se indica que parámetros se van a traer del padre 1 y cuáles del padre 2.

Para poner a modo de ejemplo, se tienen 10 zonas por individuo y el punto de cruce fue 6, entonces esto indica que el primer hijo va a heredar la ubicación de las 6 primeras zonas del padre 1 y también hereda la ubicación de las 4 últimas zonas del padre 2. Entonces, el segundo hijo va a heredar las 6 primeras ubicaciones de zonas del padre 2 y las 4 últimas ubicaciones del padre 1.

3. Insertar elementos de ambos padres en ambos hijos

Esta sección es de mucha importancia, ya que puede existir el caso de que si al insertar un conjunto de zonas del padre 2 en el hijo 1, algunas de estas se sobrepongan a las ya existentes. Por eso, se crea una función que sobrepone las zonas en los campos vacíos y si hay zonas que se sobreponen sobre otras zonas entonces esa zona no se pone y pasa a una lista de zonas que no se pueden poner.

4. Inserción de zonas que no se pudieron sobreponer

Todas las zonas que no se pudieron localizar directamente porque se sobreponía con otra zona, son insertadas en nuevas posiciones aleatorias, similar a como sucede con la generación de individuos.

```
function pob_out=Cruce(pob,size_roll,parts,order)
    %Convertir la orden a un plan de cortes
    cuts=Order2cuts(parts,order);
    %Numero de cortes a realizar
    n_cuts=size(pob(1,1).gen,1);
    cut_info = [];
    %Hijos generados
    Sons = [];

    for i=1:2:size(pob,1)
        %Se trae toda la informacion de dos padres
        Paren1=pob(i,:).gen;
        Paren2=pob(i+1,:).gen;
        %Se escoge un punto de cruze del genotipo
        point_cross=randi(n_cuts);

        %% Primer Hijo
        child1=Paren1(1:point_cross,:);

        %Se crea un arreglo que contendra las partes que no se puedieron
        %colocar de un padre en otro
        cuts_nmatch=[];

        %Funcion que inserta los elementos que se puedan del padre 2 en el
        %hijo 1 y muestra que valores no se pudieron poner
        [child1,cuts_nmatch,cut_info] = CutsNMatch(point_cross,n_cuts,Paren2,child1,size_roll);

        %Se verifica si hubo alguna pieza que no se haya podido insertar
        if size(cuts_nmatch)==0
        else
            pos_likely=Pos_likely_Find(child1);
            child1 = PutNMatch(cuts_nmatch,pos_likely,child1,size_roll,cut_info,cuts);
        end
        ind1=GenInd(child1,0,0,0,0);
        Sons = [Sons;ind1];
```

Fig. 16. Código utilizado para el cruce.

Fig. 17. Población cruzada.

```
for i=1:size(pobCru)
        Cp = [];
        Cp = pobCru(i,:).gen;
        %Genero un parametro de mutacion. Se va a mutar aproximadamente al
        %10% de los individuos
        MutationParam = rand();
        if MutationParam < 0.2
           BgP = [];
           size_Piece = [];
           %Se selecciona una pieza aleatoria para mutar
           RandomPiece = randi(size(Cp,1));
           if RandomPiece == 1
              BgP = [BgP Cp(RandomPiece:end,:)];
           elseif RandomPiece == size(Cp)
              BgP = [BgP Cp(1:RandomPiece-1,:)];
           else
              BgP = [BgP Cp(1:RandomPiece-1,:);Cp(RandomPiece+1:end,:)];
           end
           idPiece = Cp(RandomPiece,1);
           size_Piece = Cp(RandomPiece,12:13);
           cut_info = [idPiece size_Piece];
           loc_dir = Cp(RandomPiece,14);

           Piece = Cp(RandomPiece,:);

           pos_likely=Pos_likely_Find(BgP);

           Mut = PlaceMutation(pos_likely,Piece,BgP,size_roll,cut_info,cuts,loc_dir);

           ind2=GenInd(Mut,0,0,0,0);
           Muts = [Muts;ind2];

           Heu=Fitness(Mut,size_roll,cuts);
           ind3=GenInd(Mut,Heu,0,0,0);
           Muts2 = [Muts2;ind3];
```

Fig. 18. Código utilizado para la mutación.

6. Mutación.

Consiste en alterar, modificar o cambiar algún gen. En este caso, es modificar una única zona de un individuo.

La mutación no se aplica en todos los individuos sino que existe un factor de mutación de 0,2, esto significa que aproximadamente el 20% de los individuos van a ser mutados.

Para determinar cuál se muta y cual no, se genera un número aleatorio entre 0 y 1 y si es menor que 0,2, se muta, si no, no se muta.

Si el valor fue menor a 0,2, entonces se selecciona una pieza aleatoria del individuo a mutar.

La forma para mutar es coger esa zona seleccionada y rotarla en una dirección aleatoria. Si esa rotación es efectiva, es decir, al momento de rotar, la nueva posición de esa zona no tiene intersección con otras zonas o con los límites del terreno entonces la zona queda mutada. Si existe alguna intersección, entonces esa zona se saca y se posiciona en un nuevo lugar dado por todas las coordenadas de los vértices de las demás zonas del individuo, entonces la zona queda ubicada de forma correcta en otro punto del terreno.

Fig. 19. Población mutada.

Después de todo este procedimiento, se organizan todos los individuos por su Fitness, de esta forma, después de la selección, cruce y mutación se obtienen nuevos individuos que teóricamente son mejores que la población inicial, entonces, se organizan a estos nuevos individuos y se saca al mejor.

Luego, como el algoritmo se repite según el número de generaciones, entonces se van sacando los mejores individuos de cada generación para ver como el algoritmo evoluciona.

Para comprobar que el algoritmo evoluciono, se organizan al final todos los mejores individuos y se miran cuantos representan la solución óptima, si al menos 3 individuos de los 16 (Dados por el número de generaciones) son la solución óptima, entonces el algoritmo evoluciona y obtiene después óptimas.

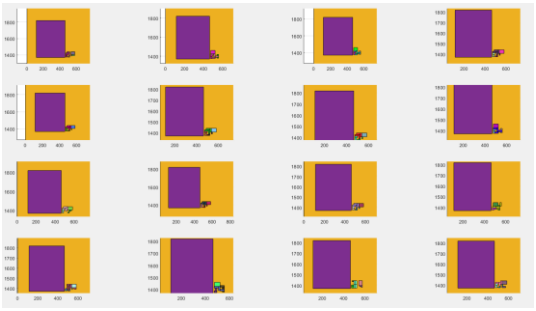

Fig. 20. Individuos obtenidos después de 16 generaciones.

Como se puede ver en la figura 20, algunos de los individuos obtenidos representan una buena solución, por lo tanto, este programa cumple satisfactoriamente con hallar las mejores soluciones de como ubicar correctamente las zonas para obtener el desperdicio mínimo.

Ahora bien, tras comprobar que el programa funciona correctamente, se empieza a variar el número de individuos y de generaciones, esto se hace para ver dos cosas.

La primera es ver si hay un número mínimo de generaciones donde ya se obtenga al mejor individuo y ver en que numero de generaciones ya no es necesario colocar más puesto que ya se alcanzó el límite de mejores individuos obtenidos.

Tabla 1. Comparación de la cantidad de buenos individuos y tiempo empleado para la ejecución del programa según el número de generaciones.

# Buenos Individuos	# de Generaciones	Tiempo Empleado (s)
3	5	00.51.13
8	16	01.34.83
13	24	02.37.62
19	30	03.27.64

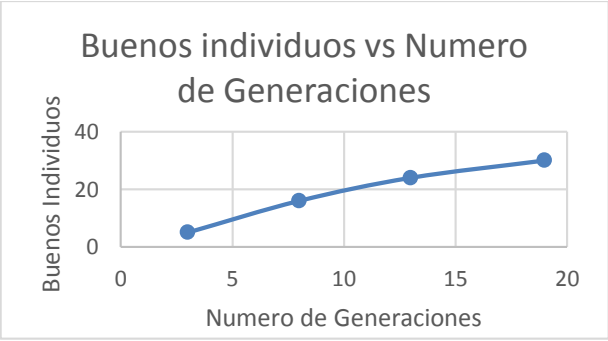

Fig. 21. Comparación del número de individuos VS Número de generaciones.

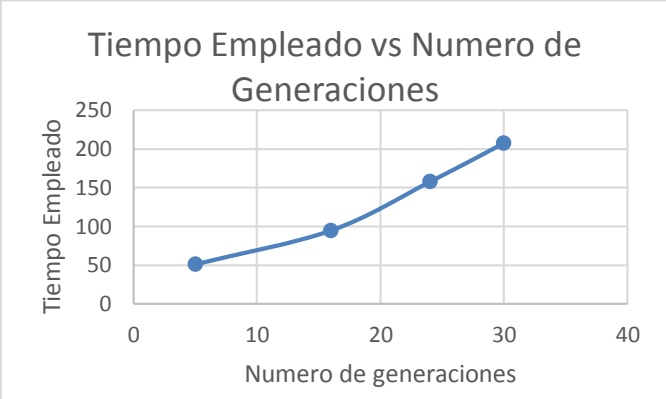

Fig. 22. Tiempo Empleado VS Número de generaciones.

Con base en las gráficas se puede establecer que un número mayor de generaciones no necesariamente garantiza obtener la respuesta óptima, con un número menor se puede hallar una gran cantidad de individuos óptimos.

Por otro lado los tiempos de ejecución son bastante demorados y esto se debe, además de la cantidad ciclos que debe ejecutar el programa, a que también el procesador del computador es Intel Pentium, un procesador de bajas prestaciones lo que ralentiza el proceso.

V. CONCLUSIONES

En general, cada sección del código tiene su papel fundamental a la hora de generar un buen algoritmo genético.

Empezando por la selección inicial, es necesario que esta tenga buenos criterios, es decir, la población inicial debe tener cierta lógica con respecto al problema planteado, ya que una población inicial que no cumpla con las condiciones básicas, hará que el programa funcione erróneamente, generando nuevos individuos que quizá no cumplas con las condiciones dadas por el problema.

A la hora de realizar la selección es necesario definir con claridad el criterio que se va a utilizar, ya que si por ejemplo, en un caso se tienen individuos muy buenos y otros muy malos y ese margen es claramente diferenciado, no sería bueno escoger una selección por ruleta, porque el mismo algoritmo ya nos está diciendo que individuo es bueno y debe ser seleccionado.

En la mutación se debe tener bastante cuidado con el parámetro escogido para mutar y la forma de mutar. Una mala mutación

hace que el algoritmo pueda involucionar. Por ejemplo, en este programa, si la mutación hace que una pieza se desvíe mucho del área donde están todas concentradas, hace que a partir de este punto las nuevas generaciones tengan ese elemento alejado y entonces el programa tendría que gastar más generaciones para llegar a la solución óptima o incluso puede que a partir de ese nueva pieza alejada, se aumenten las distancias de las piezas en las próximas generaciones por lo que el algoritmo ya no cumpliría con su propósito real, evolucionar.

Uno de los aspectos más importantes a tener en cuenta en un algoritmo genético es el Fitness que se haya seleccionado. Este criterio es crucial a la hora del funcionamiento del algoritmo genético. Un mal Fitness hará que el algoritmo no evolucione, es decir, no se vuelva mejor o no llegue a una solución óptima sino que por el contrario, puede que el algoritmo sea involutivo y o que haga es cada vez alejarse de una buena solución. Mientas que un buen Fitness, hará que luego de algunas generaciones, se pueda obtener la solución en la mayoría de los individuos, lo cual indica que fue un buen criterio. Por lo tanto, antes de realizar el algoritmo genético completo, es necesario tener varias opciones de como calcular el Fitness e irlas implementando, una por una, hasta obtener un criterio donde se evidencie que el algoritmo evoluciona y que efectivamente se obtienen soluciones óptimas.

El algoritmo genético se caracteriza porque este va evolucionando a lo largo del tiempo después de un número n de generaciones. Ahora bien, esto tiene un límite y está dado por el dispositivo físico que esta ejecutado el algoritmo, es decir, no es viable colocar 10000 generaciones a un procesador malo, así como tampoco sería bueno colocar muy pocas generaciones si el procesador tiene buena capacidad. La idea es aprovechar de forma óptima los elementos que se tienen a la mano para realizar la implementación y las simulaciones, pero es necesario conocer las restricciones de los elementos físicos pues si no, el compilar el programa podría tardar muchísimo tiempo.

VI. Referencias

[1] L. Davis (ed.) (1991). Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York. Recuperado de: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/tema geneticos.pdf

[2] Sancho, Algoritmos Genéticos: Conceptos Básicos. 15 de noviembre de 2018 Recuperado de: http://www.cs.us.es/~fsancho/?e=65

[3] Imagen obtenida de: https://cmapspublic2.ihmc.us/rid=1KNKM1J52-NJG9K4-19JC/Algoritmos%20Gen%C3%A9ticos.cmap

[4] Tecno Agenda. Algoritmos Genéticos: Búsqueda y Optimización por Selección Natural. 25 de abril de 2017. Recuperado de: http://tecnoagenda.com/algoritmos-geneticos-busqueda-y-optimizacion-por-seleccion-natural/

[5] Russell, S & Norvig, P. Inteligencia artificial un enfoque moderno. 2008. Pearson Education. ISBN: 978-84-205-4003-0

[6] Malagón, C. Búsqueda heurística. (Sin fecha definida). Área de computación e inteligencia artificial. Recuperado de https://www.nebrija.es/~cmalagon/ia/transparencias/busqueda_heuristica.pdf

[7] Revoluciona. Algoritmos para plantear una búsqueda heurística. (Sin fecha definida). Imagen recuperada de: https://www.revolucionia.com/2018/01/Algoritmos_para_busqueda_heuristica.html