UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Master en Ingeniería de Sistemas Automáticos y Electrónica Industrial

Trabajo de Aplicación de Técnicas de Inteligencia Artificial y Aplicaciones en Automatización

Planificador de una Tarea Productiva con Algoritmos Genéticos

Autores: **Marco Bravo**
Alexander Bermeo
Diego Lascano
Diego Mosquera

Profesor: **Bernardo Morcego**

Mayo 2018

Índice de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Planteamiento del Problema

La planificación de tareas que tienen duraciones arbitrarias, restricciones de precedencia, restricciones de disponibilidad de recursos, etc., es un problema difícil de resolver y computacionalmente costoso. Este proyecto consiste en crear un algoritmo basado en técnicas evolutivas que resuelva este problema eficientemente y de forma óptima o casi óptima.

El proceso productivo que se pretende optimizar consiste en el montaje de un producto formado por 4 componentes. De estos componentes hay 3 que llegan a través de una línea de producción que puede servir hasta 9 tipos de componentes diferentes, el cuarto es común para todos los productos y siempre está disponible. Así, en este proceso se pueden montar productos tipo 'ABC', 'CBA', 'DEF', 'JKL', etc. Los datos más relevantes del proceso son:

- Los componentes llegan a la zona de montaje en lotes de 4 unidades iguales.
- En esta zona se dispone de un almacén con capacidad para 16 productos.
- Un producto tarda un tiempo dado en producirse, pero este tiempo no es fijo, tiene una distribución normal.
- Los pedidos se hacen con fecha de entrega y contienen una relación de los productos que se requieren.
- En un pedido puede haber productos de diferentes tipos.

La siguiente figura muestra el esquema del proceso productivo anteriormente expuesto.

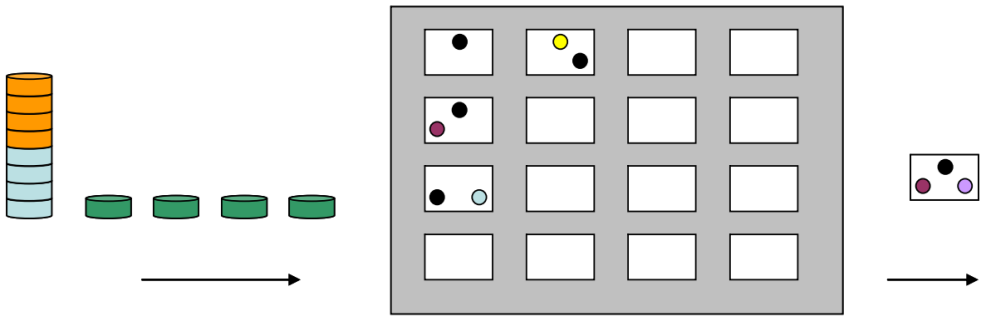

Figura 1. Representación del proceso productivo planteado

1.2. Objetivos

El objetivo general del presente trabajo es diseñar y desarrollar una aplicación, basada en programación con algoritmos genéticos, para encontrar la solución más eficiente y óptima al problema de planificación del proceso productivo planteado.

Para alcanzar este objetivo general se han planteado los siguientes objetivos específicos:

1. Estudiar las diversas técnicas evolutivas y sus posibles aplicaciones en la resolución de sistemas complejos y ejemplos experimentales de su utilización.
2. Utilizar herramientas informáticas para el desarrollo de una aplicación que simule la operación, el funcionamiento, y ejecute los algoritmos de solución del problema.
3. Desarrollar el algoritmo genético que reciba la información, la procese y determine una solución al problema de producción planteado.
4. Realizar pruebas del funcionamiento del algoritmo y depurar el mismo en base a los resultados obtenidos.
5. Desarrollar una memoria técnica de la documentación, procesos, datos, e información utilizada para el desarrollo del trabajo.

1.3. Planificación del Trabajo

Para alcanzar los objetivos planteados, el trabajo se ha organizado en tareas que se han asignado a cada uno de los integrantes del grupo. En la Tabla 1 se detallan las actividades realizadas, su duración y las iniciales de los nombres del o los responsables de llevar a cabo cada una de ellas. En la figura 2 se presenta el diagrama de Gantt que otorga una visualización de los inicios y fin de cada actividad a lo largo del cuatrimestre; en este caso se ha colocado los identificadores de cada actividad de la Tabla 1 en lugar de sus nombres.

Tabla 1. Actividades realizadas por el grupo de trabajo y su duración.

ID	Actividad	Herramientas	Responsable	Horas
1	Reunión de trabajo inicial.	Bibliografía	DM, DL, MB, AB	5
2	Primera reunión con el Tutor.	Entregable N1	DM, DL, MB, AB	1
3	Estudio de la teoría de Algoritmos Genéticos.	Bibliografía	MB, AB	15
4	Estudio de bibliografía de Algoritmos Genéticos orientada a Matlab.	Matlab, Bibliografía	DM, DL	15
5	Identificación y definición de parámetros del problema.	Bibliografía, Descripción del problema	MB, AB	4
6	Definición de función de evaluación.	Bibliografía, Descripción del problema	DM, DL	4
7	Reuniones para Socializar Avances	Bibliografía, Descripción del problema	DM, DL, MB, AB	10
8	Segunda reunión con el tutor.	Entregable N2	DM, DL, MB, AB	1
9	Programación de la función de evaluación	Matlab, Bibliografía	DM, DL	15
10	Programación del programa principal	Matlab, Descripción del problema	DM, DL	15
11	Pruebas de funcionamiento y parámetros de optimización del Algoritmo Genético programado.	Matlab, Descripción del problema	DM, DL	5
12	Desarrollo de la memoria técnica.	Matlab, Bibliografía y Software editor de textos y tablas	MB, AB	25
13	Preparación Presentación	MS Power Point	MB, AB	5
			Total Horas:	**120**

AB: Alexander Bermeo, **MB:** Marco Bravo **DL:** Diego Lascano, **DM:** Diego Mosquera

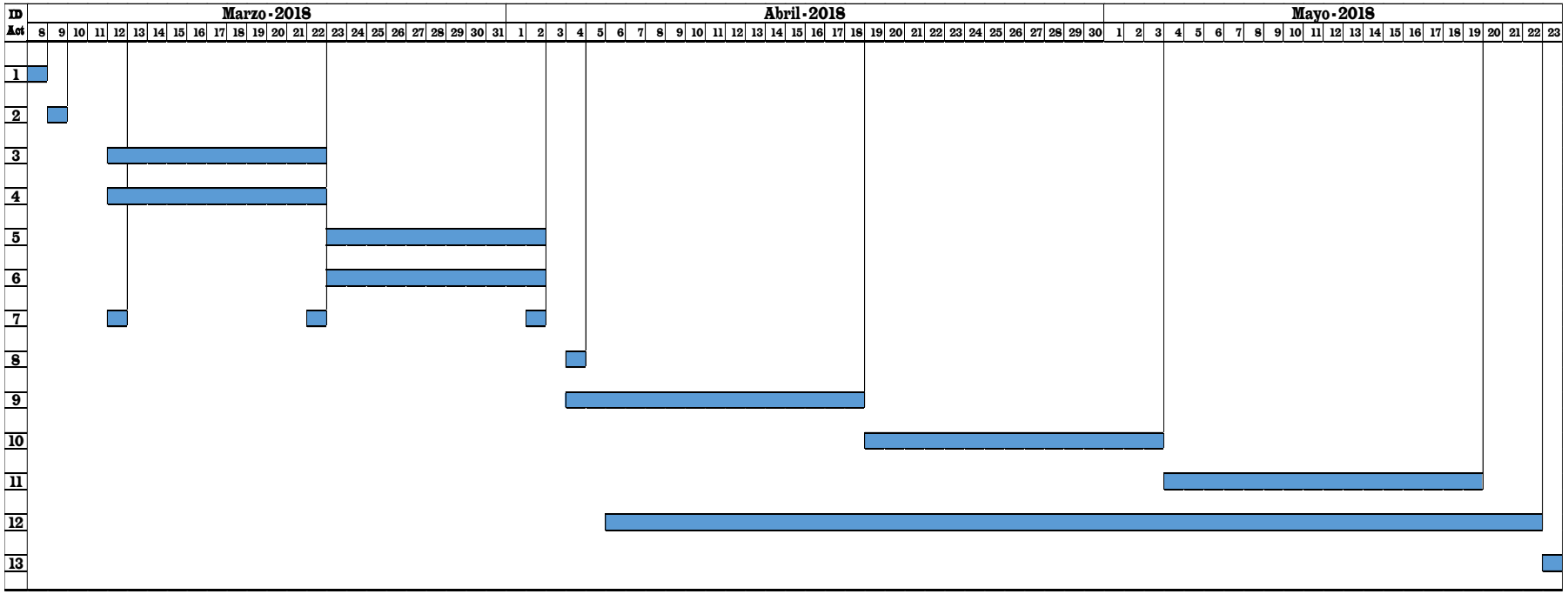

Figura 2. Diagrama de Gantt que representa las actividades realizadas por el grupo de trabajo desde el 8 de marzo al 23 de mayo del 2018

2. Problemas de planificación / programación de tareas (scheduling)

Scheduling es el proceso de asignar recursos a tareas a lo largo del tiempo. Los problemas de scheduling se presentan en áreas tan diversas como planificación de la producción, planificación de personal, diseño de computadores, confección de horarios, etc.

Existen dos familias de Scheduling que son:

- Problema de Scheduling “puros”.
- Problemas de asignación de recursos.

Problema de Scheduling “puros”: la capacidad de cada recurso está definida sobre un cierto número de intervalos temporales y el problema consiste en cubrir las demandas de recursos de las actividades a lo largo del tiempo, sin sobrepasar las capacidades disponibles. Para este problema se conoce a priori qué recursos va a utilizar cada una de las actividades. Siempre dependerá del uso de los recursos por parte de las actividades y se podrá distinguir cuatro patrones de flujo:

Flujo aleatorio (Open-Shop): No hay restricción en cuanto al orden de uso de los recursos por las Actividades de cada uno de los trabajos.

Flujo general (Job-Shop): Cada trabajo, o conjunto de actividades, debe usar los recursos en un orden determinado (restricciones tecnológicas). “N” trabajos deben ser procesadas, una sola vez, por "M' recursos, con un orden, y durante un tiempo dado.

Flujo regular (Flow-Shop): En todos los trabajos se utilizan los recursos en el mismo orden. Es un caso particular del job-shop.

Flujo permutacional (Permutation Flow-Shop): Todos los recursos procesan los trabajos en el mismo orden. Es un caso particular del flow-shop.

Problemas de asignación de recursos: Disponemos de un conjunto de operaciones, y para cada una de ellas se dispone de un conjunto de recursos idénticos (realizan el mismo tipo de operación), pero no equivalentes (pueden requerir distintos tiempos o costes de procesamiento), que pueden ser utilizados. Para este caso no se conoce a prioridad que recurso en concreto va a utilizar para que se cumplan todas las demandas (Galipienso, 2001).

En base al estudio de los problemas de planificación y organización de tareas (scheduling) se puede constatar que el problema sobre el cual se ha desarrollado el presente trabajo pertenece a la familia de problemas Scheduling “puros”. En base a este hecho podemos decir que los algoritmos genéticos representan uno de los métodos con los que se puede darse solución a los problemas scheduling puros.

3. Algoritmos Genéticos

Los algoritmos genéticos (AG) son una técnica de optimización y búsqueda basada en los principios de la genética y la selección natural. La técnica fue descrita inicialmente por John Holland (1975); quien lo presentó como un algoritmo que permite a una población de individuos evolucionar, bajo reglas de selección especificadas, a un estado que maximice la “bondad”, es decir, minimice la función de costo (Sivanandam, 2007).

En el trabajo de John Holland, los AG presentan soluciones a un problema dando valores a un conjunto de parámetros lo cuales se codifican en una cadena de valores denominada cromosoma o individuo de una población. Si se asigna un determinado número de bits a cada gen que conforma el cromosoma, el valor que puede tomar cada uno de los bits pertenecientes a un gen recibe el nombre de alelo. En este caso, al ser una representación binaria solo existen dos valores posibles, 0 o 1. Las distinciones entre cromosoma, gen y alelo se identifican en la figura 3.

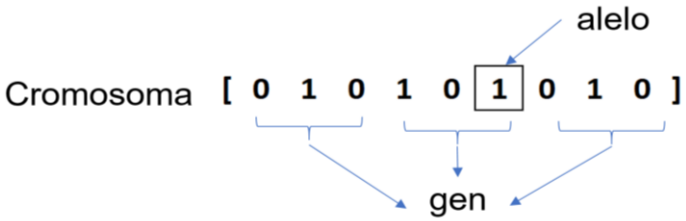

Figura 3. Ejemplo de representación binaria de un cromosoma.

Realizado por: Los Autores.

Una función de costo genera un resultado de un conjunto de variables de entrada (un cromosoma). La función de costo puede ser una función matemática, un experimento o un juego. El principal objetivo es modificar el resultado de alguna manera deseable mediante la búsqueda de los valores adecuados para las

variables de entrada. Un ejemplo es cuando sin pensar llenamos una bañera en agua. El costo es la diferencia entre las temperaturas deseadas y reales del agua. Teniendo dos variables de entrada cuando giramos cada llave como salida de agua fría y caliente. En este caso, la función de costo es el resultado experimental de meter la mano en el agua. Entonces vemos que la determinación de una función de costo apropiada y la decisión de qué variables usar están íntimamente relacionadas (Haupt, 2008).

3.1. La población

Una población es un conjunto de cromosomas, la población es la unidad que usa un Algoritmo Genético para hallar una solución óptima. Esta población inicial servirá de base para las sucesivas generaciones y en caso de poderla generar de forma aleatoria (Haupt, 2008).

La población tiene cromosomas N_{pob} y es una matriz de N_{pob} x N_{bits} que se genera números aleatorios con la función rand de manera uniforme entre cero y uno y la función round esta redondea los números al entero más cercano que en este caso es 0 o 1 de la siguiente manera:

Pob=round(rand(N_{pob}, N_{bits}));

3.2. Evaluación

Para que un AG funcione correctamente debe poseer un proceso o método que indique si los individuos de la población representan o no buenas soluciones. Esto quiere decir que evaluará qué tan buenos son los individuos de la población y codificarlos. El proceso o método de evaluación se realiza mediante una función que proporciona una medida numérica de la bondad de una solución. La medida es nombrada como ajuste o *fitness* y a cada uno de los individuos se le asigna un valor de *fitness* escalar por medio de un procedimiento de evaluación bien definido. Con la ayuda de esta función de evaluación se podrá controlar la aplicación de los operadores genéticos. Permitiendo controlar el número de selecciones, cruces, copias y mutaciones llevadas a cabo (Mendez, 2008).

3.3. Selección

Una vez evaluadas todas las soluciones de la población en una generación, el tipo de selección juega un papel importante ya que esta será la encargada de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no;

imitando lo que ocurre en la naturaleza y otorgando un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto, la selección de un individuo estará relacionada con su valor de *fitness* (Haupt, 2008).

En primer lugar, los costos de N_{pob} y los cromosomas asociados se clasifican de menor costo a mayor costo. Luego, solo los mejores se seleccionan para continuar, mientras que el resto se eliminan. La tasa de selección, X_{rate}, es la fracción de N_{pob} que sobrevive para el siguiente paso de apareamiento.

Entre toda la población N_{pob}, solo el N_{keep} superior sobrevive para el apareamiento, y la N_{pob} inferior se descarta para dar espacio al nuevo descendiente.

A continuación, se presenta varios métodos de selección:

3.3.1. Selección por emparejamiento de arriba a abajo.

Empieza desde la parte superior de una lista, emparejando los cromosomas de dos en dos hasta que los cromosomas superiores de N_{keep} se seleccionen para el apareamiento. Por lo tanto, el algoritmo combina filas impares con filas pares. La madre tiene números de fila en la matriz de población dada por *ma* = 1, 3, 5,…,n y el padre tiene los números de fila pa = 2, 4, 6,…,n. Esta selección no modela bien la naturaleza, pero es fácil de programar (Haupt, 2008).

3.3.2. Selección por emparejamiento aleatorio.

Se generan números aleatorios uniformemente para seleccionar cromosomas. Los números de fila de los padres se encuentran usando:

$ma=ceil(N_{keep}*rand(1, N_{keep}))$

$pa=ceil(N_{keep}*rand(1, N_{keep}))$

donde ceil redondea el valor al siguiente entero más alto. (Haupt, 2008)

3.3.3. Selección por emparejamiento aleatorio ponderado.

Este tipo de ponderación a menudo se denomina ponderación de ruleta. Se crea una “ruleta” con todos los individuos presentes en una generación. A cada individuo se le asigna una porción de la ruleta en función de la puntuación recibida en la etapa de evaluación, es decir, en función del *fitness*, de tal forma que todas las porciones sumen la unidad. La probabilidad de que un individuo sea seleccionado es igual a la probabilidad que tienen todos los demás participantes

por que la selección es al azar y el participante puede ser seleccionado más de una vez. Este método es sencillo pero ineficiente a medida que aumenta el tamaño de la población (Haupt, 2008).

3.3.4. Selección por torneo.

La selección de torneos funciona mejor para tamaños de población muy grandes porque la clasificación consume mucho tiempo para grandes poblaciones. Se hacen subgrupos de individuos de la población, quienes competirán ente ellos, siendo el ganador del subgrupo seleccionado para la reproducción (Haupt, 2008).

3.4. Cruce

El cruce es la creación de uno o más descendientes de los padres seleccionados en el proceso de emparejamiento. La composición genética de la población está limitada por los miembros actuales de la población. El objetivo principal del cruce es obtener una generación con mejores características de ambos padres. una vez compartido las mejores características de dos individuos, la nueva descendencia deberá tener una aptitud mayor que la de sus padres (Haupt, 2008).

Este proceso es análogo a la recombinación de la reproducción sexual biológica, en la que se inspira los AG. A continuación, se explicará dos métodos más utilizados.

3.4.1. Cruce de un punto

Se seleccionan dos individuos, se corta sus cromosomas por un punto seleccionado aleatoriamente generando dos segmentos diferentes, a estos le nombramos cabeza y cola. A continuación, se intercambian las colas entre individuos generando los descendientes. Como se presenta en la figura 4.

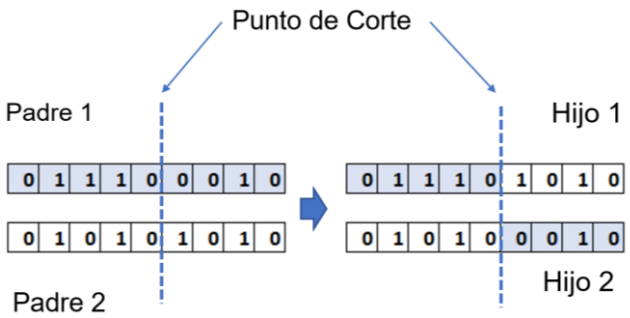

Figura 4. Cruce de 1 punto.
Realizado por: Los Autores.

3.4.2. Cruce de dos puntos

En este caso se corta los cromosomas de los padres mediante 2 puntos generados aleatoriamente. Los cortes no deben coincidir con los extremos de los cromosomas y se pueden obtener 3 segmentos. La descendencia se forma intercambiando el segmento central de los padres. Como se presenta en la figura 5.

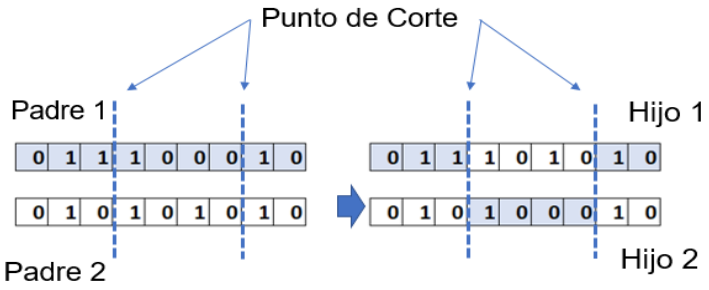

Figura 5. Cruce de 2 puntos.

Realizado por: Los Autores.

3.5. Mutaciones

La mutación es un término que está referido a la selección agresiva, y se define como la introducción de material genético que no se encuentra dentro de la población actual. El objetivo es añadir diversidad genética a la población, siempre se debe considerar que los valores no pueden ser repetidos. Es importante saber que la mutación también se puede realizar juntamente con el cruce. La mutación es la segunda forma en que una AG explora una superficie de costos. Puede introducir rasgos que no están en la población original y evita que el AG converja demasiado rápido antes de muestrear toda la superficie de costos. (Haupt, 2008)

[0111**0**1010] → [0111**1**1010]

Por lo tanto, el bit de la columna 5 del vector de población se muta de un 0 a un 1.

3.6. La próxima generación

Después de que tienen lugar las mutaciones, se calculan los costos asociados con la descendencia y los cromosomas mutados. El proceso descrito se itera. (Haupt, 2008)

3.7. Convergencia

El número de generaciones que evolucionan depende de si se alcanza una solución aceptable o si se excede un número determinado de iteraciones. Después de un tiempo, todos los cromosomas y los costos asociados se volverían iguales si no fuera por mutaciones. En este punto, el algoritmo debe detenerse (Haupt, 2008).

4. Metodología aplicada

Para desarrollar un proceso de solución basado en algoritmos genéticos en primer lugar se debe encontrar la forma más apropiada de representar la información que ingresará al algoritmo. En la figura 6 se muestra el diagrama de flujo que representa la forma de solucionar el problema planteado por medio de algoritmos genéticos.

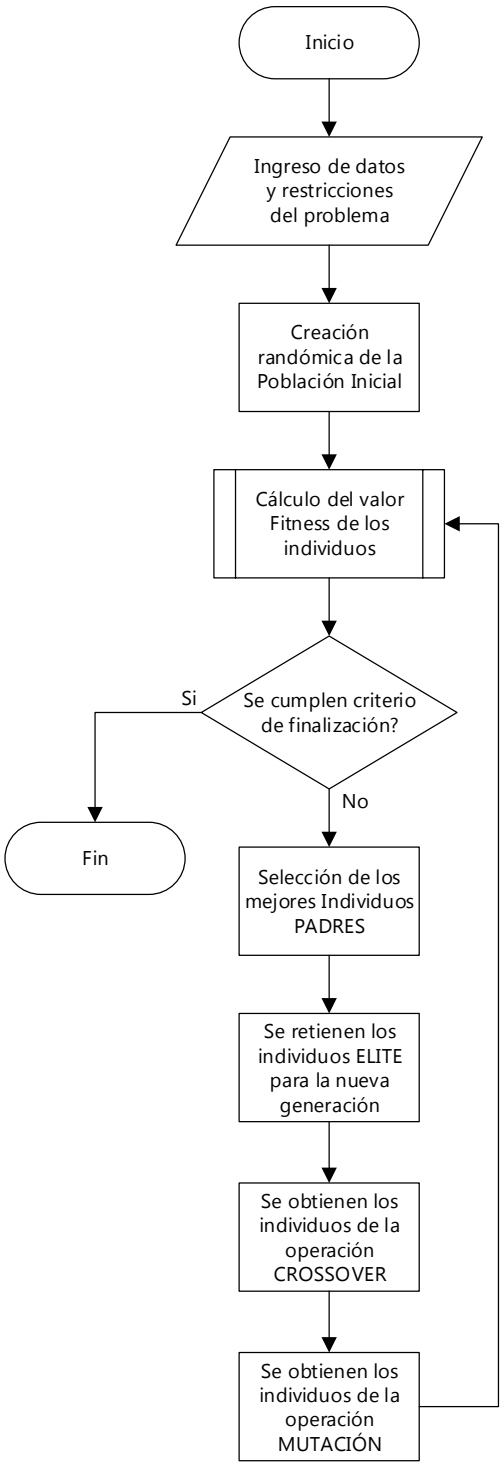

Figura 6. Diagrama de flujo del algoritmo genético que se aplicará para solucionar el problema planteado.

De igual manera, el diagrama de flujo de la función fitness utilizada se presenta en la figura 7.

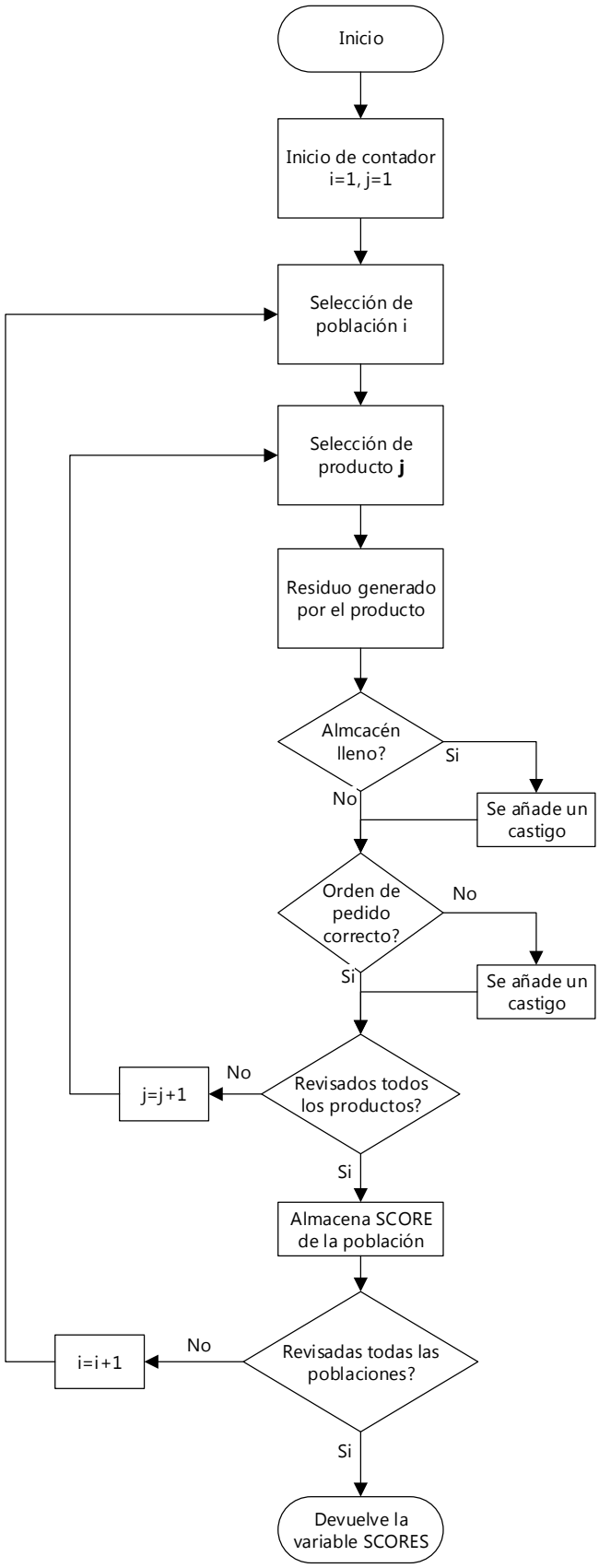

Figura 7. Diagrama de flujo de la función fitness que se utilizará en el algoritmo genético.

La optimización del problema se ha programado en base al comando de algoritmo genético de MatLab (MathWorks, 2018). Dicho comando presenta la siguiente estructura:

[x,fval,exitflag,output,population,scores] = ga(fitnessfcn, nvars, A, b, Aeq, beq, LB, UB, nonlcon, IntCon, options)

Al lado derecho se presentan las entradas al algoritmo, las mismas se detallan a continuación.

fitnessfcn: Función fitness que deberá aceptar un vector de una única fila y tantas columnas como se especifique en el parámetro **nvars**. La función fitness debe devolver un valor escalar.

nvars: Indica el tamaño del vector que espera recibir la función fitness.

A, b: Sirve para aplicar la limitación **A·X<b** en el caso de ser necesario.

Aeq, beq: Sirve para aplicar la limitación **Aeq·X<beq** en caso de ser necesario.

LB, UB: Límites superiores e inferiores de los valores que puede recibir.

nonlcon: Permite aplicar limitaciones para valores no lineales.

IntCon: Vector que indica los posibles valores (enteros).

Options: Permite indicar opciones extras como por ejemplo el método de selección o reproducción.

En el lado izquierdo encontramos las salidas que el algoritmo genético entrega:

X: el mejor valor de entrada para la función fitness

faval: La puntuación que recibe el valor X en la función fitness.

exitflag: es un valor entero que indica la razón por la cual el algoritmo se ha detenido.

output: estructura que contiene la salida de cada generación e información sobre el rendimiento del algoritmo.

population: vector que contiene a los individuos de la población final

scores: vector que contiene los valores de la función fitness de la población final

En base al comando **ga** y sus parámetros se ha desarrollado un programa que nos permitirá darle los valores de entrada a nuestro algoritmo.

```
clear all; clc;

% Ingreso del número de pedidos
ingresos=input('Ingrese el número de pedidos: ');

% Ingreso del número de productos por pedido
productos = input('Ingrese el número de productos por pedido: ');

variantes = 9; %tipos de componentes al ingreso del proceso productivo
componentes = 3; % numero de componentes por producto

% Generación del pedido de forma aleatoria
pedido = randi (variantes, ingresos*productos, componentes);

desde=1; hasta=10;
for i = 1 : ingresos %Aquí se añade el número de pedido a la cuarta
columna
    pedido(desde: hasta,(componentes+1))=i;
    desde=desde+10;
    hasta=hasta+10;
end

% Declaración de las funciones del programa
type permutaciones_creacion.m
type permutacion_cruce.m
type permutacion_mutacion.m
type f_eval.m
type f_resultado.m

FitnessFcn = @(x) f_eval(x,pedido,componentes,ingresos);

% Declaración de variables para generar los gráficos
my_plot1 = @(options, state, flag) gaplotbestf(options, state, flag);
my_plot2 = @(options, state, flag) gaplotscores(options, state, flag);
my_plot3 = @(options, state, flag) gaplotselection(options, state, flag);
my_plot4 = @(options, state, flag) gaplotscorediversity(options, state,
flag);
my_plot5 = @(options, state, flag) gaplotstopping(options, state, flag);
my_plot6 = @(options, state, flag) gaplotgenealogy(options, state, flag);

%% Genetic Algorithm Options Setup
options = optimoptions('ga', 'PopulationType',
'custom','InitialPopulationRange', ...
                           [1;(ingresos*productos)]);

options = optimoptions(options,'CreationFcn', @permutaciones_creacion,
...
                  'CrossoverFcn', @permutacion_cruce, ...
                  'MutationFcn', @permutacion_mutacion, ...
                  'PlotFcn', {my_plot1, my_plot2, my_plot3, my_plot4,
my_plot5}, ...
                  'MaxGenerations',750,'PopulationSize',40, ...
                  'MaxStallGenerations',150,'UseVectorized', true);

numberOfVariables = (ingresos*productos);

[x,fval,reason,output] = ...
    ga(FitnessFcn,numberOfVariables,[],[],[],[],[],[],[],options);

pedido_ordenado = f_resultado(x, pedido, componentes);
```

la función fitness que se ha establecido para la solución de nuestro problema de optimización es la siguiente:

```
function scores = f_eval(x, pedido, componentes, ingresos)

scores = zeros(size(x,1), 1);
nlote = 4; % Numero de componentes repetidos que ingresan al sistema
rangoPedido = size(pedido); %se guarda el # de productos en el pedido
total
t_almacen = 18; %capacidad máxima del almacén de 6 posiciones
%Aqui inicia el análisis de cada población de la nueva generación
for n = 1 : size(x,1) %recorremos todas las poblaciones
    poblacion = x{n};
    %enceramos los residuos de cada elemento al cambiar de población
    rA = 0; rB = 0; rC = 0; rD = 0; rE = 0; rF = 0; rG = 0; rH = 0; rI =
0;
    cBonus=0; cAlmacen=0; cOrden=0; %enceramos los elementos de castigo
    rT = zeros(rangoPedido(1,1),1);
    pedido1 = 0; pedido2 = 0; pedido3 = 0; pedido4 = 0;

    %Aquí inicia el ciclo de comparación de cada producto del pedido
    for m = 1 : rangoPedido(1,1)

      %recorremos todos los productos en el orden de la población
generada por el AG
        i = poblacion(m);

      %recorremos los elementos de un producto (sin tomar en cuenta el #
de pedido)
        for j = 1 : componentes
            switch pedido(i,j)

                case 1 %comparamos si el elemento es tipo A
                    if rA > 0 %si existe residuo, usamos uno en producto
                        rA = rA - 1;
                    elseif rA == 0 %si no hay residuo, ingresa un nuevo
                                   lote menos 1
                        rA = rA + (nlote-1);
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 2 %comparamos si el elemento es tipo B
                    if rB > 0 %si existe el residuo, usamos uno para el
producto
                        rB = rB - 1;
                    elseif rB == 0
                        rB = rB +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 3 %comparamos si el elemento es tipo C
                    if rC > 0 %si existe el residuo, usamos uno para el
producto
                        rC = rC - 1;
                    elseif rC == 0
                        rC = rC +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end
```

```
                case 4 %comparamos si el elemento es tipo D
                    if rD > 0 %si existe el residuo, usamos uno para el
producto
                        rD = rD - 1;
                    elseif rD == 0
                        rD = rD + (nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 5 %comparamos si el elemento es tipo E
                    if rE > 0 %si existe residuo, usamos uno en producto
                        rE = rE - 1;
                    elseif rE == 0
                        rE = rE +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 6 %comparamos si el elemento es tipo E
                    if rF > 0 %si existe residuo, usamos uno en producto
                        rF = rF - 1;
                    elseif rF == 0
                        rF = rF +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 7 %comparamos si el elemento es tipo E
                    if rG > 0 %si hay residuo, usamos uno en el producto
                        rG = rG - 1;
                    elseif rG == 0
                        rG = rG +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 8 %comparamos si el elemento es tipo E
                    if rH > 0 %si hay residuo, usamos uno para producto
                        rH = rH - 1;
                    elseif rH == 0
                        rH = rH +(nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end

                case 9 %comparamos si el elemento es tipo E
                    if rI > 0 %si hay residuo, usamos uno en el producto
                        rI = rI - 1;
                    elseif rI == 0
                        rI = rI + (nlote-1); %si no hay residuo
disponible, ingresa un nuevo lote menos 1
                        cBonus = cBonus+2; %cada que ingresan más
componentes, se castiga esta decisión
                    end
                otherwise
            end
        end
        %al final de analizar cada producto, verificamos si el almacén se
        %lleno
```

```
        if (rA + rB + rC + rD + rE + rF + rG + rH + rI)>t_almacen
            cAlmacen = t_almacen; % si el almacén está lleno, se ingresa
un valor de   castigo
        end

  %En esta parte se analiza cómo se van completando los pedidos en
función de su orden
        If (pedido(i,componentes+1) == 1)
            pedido1 = pedido1+1;
        elseif (pedido(i,componentes+1) == 2)
            pedido2 = pedido2+1;
        elseif (pedido(i,componentes+1) == 3)
            pedido3 = pedido3+1;
        elseif (pedido(i,componentes+1) == 4)
            pedido4 = pedido4+1;
        end

        %comparamos si se están realizando los pedidos finales en primer
        %lugar para castigar la secuencia.
        if (pedido4>pedido2 || pedido4>pedido3 || pedido3>pedido2 ||
pedido2>pedido1
             || pedido4>pedido1 || pedido3>pedido1)
            cOrden = cOrden+8; % Castigo ingresado si se ejecuta pedido
posterior
        end

        rT(i)=rA + rB + rC + rD + rE + rF + rG + rH + rI + cBonus +
cAlmacen + cOrden;
    end

    scores(n)=sum(rT); % Guardamos la calificación de este individuo
analizado
end
```

5. Resultados

5.1. Ejemplo para comprobación de funcionamiento

Con el fin de mejorar el entendimiento se analiza un problema similar al originalmente propuesto, pero de menor magnitud, de tal forma que se pueda encontrar la solución manualmente y sin mayores complicaciones, para luego compararla con la o las soluciones del algoritmo genético. En este caso nos hemos planteado que cada producto tiene 2 componentes de un total de 3 que pueden ser seleccionados. Adicionalmente se ha supuesto que los componentes llegan en lotes de 3. Entonces los datos del problema se presentan a continuación:

- Componentes: A, B, C
- Elementos por Lote: 3
- Componentes de un producto: 2
- Productos que se pueden obtener sin importar el orden de elementos: 6

La representación de un producto se la realiza de forma binaria en un arreglo de datos de 6 posiciones, donde cada posición indica qué componentes están presentes en el producto.

Realizando una analogía a la estructura de los algoritmos genéticos, se interpreta que cada producto está representado por un cromosoma que contiene dos genes (componentes) y el valor de cada gen depende del alelo que se encuentre activo.

En este ejemplo se ha generado aleatoriamente un pedido que contiene 10 productos.

Tabla 2. Pedido de 10 productos, donde cada producto se forma con dos componentes de tres posibles.

	Pedido					
	Componente 1			Componente 2		
	A	B	C	A	B	C
BC	0	1	0	0	0	1
AC	1	0	0	0	0	1
BB	0	1	0	0	1	0
AC	1	0	0	0	0	1
CA	0	0	1	1	0	0
BC	0	1	0	0	0	1
CB	0	0	1	0	1	0
AB	1	0	0	0	1	0
AC	1	0	0	0	0	1
AA	1	0	0	1	0	0

El enfoque considerado en este caso para buscar una solución, es encontrar el orden en el cual los residuos que quedan tras introducir cada lote sean mínimos, y para esto se encontró una función de evaluación (fitness) que cuenta la cantidad de residuo generada en la fabricación de cada producto. Utilizando esta función de evaluación se generan los siguientes valores de residuos tomando en cuenta que el pedido se procesa en el orden inicial sin ordenar.

Tabla 3. Residuos generados si se ingresan los productos en el orden original del pedido

C1	C2	A	B	C	∑R
B	**C**	0	2	2	**4**
A	**C**	2	2	1	**5**
B	**B**	2	0	1	**3**
A	**C**	1	0	0	**1**
C	**A**	0	0	2	**2**
B	**C**	0	2	1	**3**
C	**B**	0	1	0	**1**
A	**B**	2	0	0	**2**
A	**C**	1	0	2	**3**
A	**A**	2	0	2	**4**
Residuo total 28					

Residuo Promedio	2.8
Residuo Máximo	5
Residuo Mínimo	1

Como se observa en la tabla 3, el orden original tiene un costo evaluado por la función fitness de 28, lo que además da un valor medio de 2.8 y un máximo de ocupación en almacén de 5. Un análisis en base a la lógica nos llevaría a establecer un orden como el de la tabla 4, donde el que el residuo sería el mínimo.

Tabla 4. Residuos generados si se reordena el pedido original

C1	C2	A	B	C	∑R
A	**C**	2	0	2	**4**
A	**C**	1	0	1	**2**
C	**A**	0	0	0	**0**
B	**C**	0	2	2	**4**
B	**C**	0	1	1	**2**
B	**C**	0	0	0	**0**
A	**A**	1	0	0	**1**
B	**B**	1	1	0	**2**
A	**B**	0	0	0	**0**
A	**C**	2	0	2	**4**
Residuo total 19					

Residuo Promedio	1.9
Residuo Máximo	4
Residuo Mínimo	0

La siguiente tabla muestra una comparativa entre la resolución obtenida sin utilizar el algoritmo y dos respuestas obtenidas ejecutando el algoritmo genético.

Tabla 5. Comparativa de solución manual y dos soluciones mediante algoritmos genéticos.

Solución Manual

C1	C2	A	B	C	∑
A	C	2	0	2	**4**
A	C	1	0	1	**2**
C	A	0	0	0	**0**
B	C	0	2	2	**4**
B	C	0	1	1	**2**
B	C	0	0	0	**0**
A	A	1	0	0	**1**
B	B	1	1	0	**2**
A	B	0	0	0	**0**
A	C	2	0	2	**4**
Residuo Total 19					

Promedio	1.9
Máximo	4
Mínimo	0

1ra prueba Alg. G.

C1	C2	A	B	C	∑
B	B	0	1	0	**1**
B	C	0	0	2	**2**
B	C	0	2	1	**3**
C	B	0	1	0	**1**
A	B	2	0	0	**2**
A	C	1	0	2	**3**
A	C	0	0	1	**1**
A	C	2	0	0	**2**
A	A	0	0	0	**0**
C	A	2	0	2	**4**
Residuo Total 19					

Promedio	1.9
Máximo	4
Mínimo	0

2da prueba Alg. G.

C1	C2	A	B	C	∑
A	C	2	0	2	**4**
A	C	1	0	1	**2**
C	A	0	0	0	**0**
B	B	0	1	0	**1**
A	A	1	1	0	**2**
A	B	0	0	0	**0**
C	B	0	2	2	**4**
B	C	0	1	1	**2**
B	C	0	0	0	**0**
A	C	2	0	2	**4**
Residuo Total 19					

Promedio	1.9
Máximo	4
Mínimo	0

Se observa que los 3 resultados obtenidos son diferentes en la manera de ordenar los productos, pero en valores de costo son similares, esto es debido a que el algoritmo genético genera valores aleatorios como posibles soluciones y al existir múltiples soluciones posibles, con el mismo costo se puede dirigir hacia cualquiera de estas.

Luego se han realizado variaciones en los parámetros del algoritmo y se analizó como afectan estas variaciones al comportamiento del algoritmo. El análisis se realiza con una constante de 5 individuos y se varía el número máximo de generaciones, en este ejemplo el algoritmo encuentra una solución con el valor de fitness de 19 que es el mínimo posible al crear 30 generaciones. Pero para un número menor de generaciones los valores mínimos que encuentra de fitness son de 22. A continuación se presentan los resultados que entrega el algoritmo al mantener constante una población de 5 individuos **I** y variando el número de generaciones **G**.

Tabla 6. Resultado de reordenamiento de los productos aplicando diferentes números de generaciones en el algoritmo genético

5I 10G

C1	C2	A	B	C	∑
B	B	0	1	0	1
C	A	2	1	2	5
B	C	2	0	1	3
A	C	1	0	0	1
A	B	0	2	0	2
C	B	0	1	2	3
B	C	0	0	1	1
A	C	2	0	0	2
A	A	0	0	0	0
A	C	2	0	2	4
22					

Promedio	2.2
Máximo	5
Mínimo	0

5I 15G

C1	C2	A	B	C	∑
B	B	0	1	0	1
C	B	0	0	2	2
B	C	0	2	1	3
A	B	2	1	1	4
B	C	2	0	0	2
A	A	0	0	0	0
A	C	2	0	2	4
A	C	1	0	1	2
C	A	0	0	0	0
A	C	2	0	2	4
22					

Promedio	2.2
Máximo	4
Mínimo	0

5I 30G

C1	C2	A	B	C	∑
B	B	0	1	0	1
A	B	2	0	0	2
C	A	1	0	2	3
A	C	0	0	1	1
C	B	0	2	0	2
B	C	0	1	2	3
B	C	0	0	1	1
A	C	2	0	0	2
A	A	0	0	0	0
A	C	2	0	2	4
19					

Promedio	1.9
Máximo	4
Mínimo	0

5.2. Optimización del proceso productivo planteado

El problema de optimización objeto de este trabajo, tiene una restricción en el almacén de 16 posiciones, y en cada una de estas posiciones puede almacenarse hasta 3 componentes. El análisis de la ocupación se realizó tomando en cuenta el peor de los casos que se puede presentar para un pedido que son 3 productos seguidos en los que ninguno de sus componentes es igual.

Por ejemplo cuando un pedido tiene 3 productos A B C, D E F, G H I, se generaría una serie de lotes de 4A, 4B, 4C, 4D, 4E, 4F, 4G, 4H, 4I para producir estos productos, pero el residuo de cada uno se acumularía en el almacén como muestra el grafico.

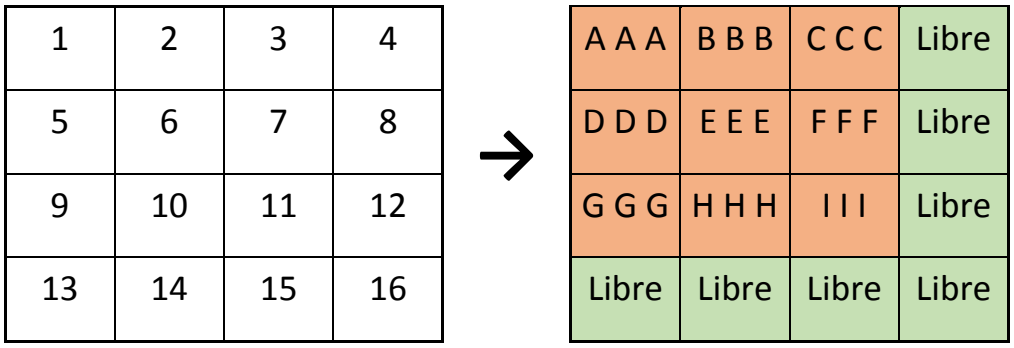

Figura 8. Demostración de que el almacén no se llena ni en el peor de los casos

Posiciones: 16

Elementos por posición: 3

Capacidad almacenamiento: 48 elementos

Máxima ocupación que va a tener: 27

Porcentaje de ocupación: $\frac{27}{48} * 100 = 56\%$

En este ejemplo, que es el peor de los casos, cualquier otro producto que se necesite producir se elaborará a partir de los elementos residuales de los anteriores productos, por lo tanto, el almacén en este punto ya no se llenará más, sino que empezará a vaciarse. Aunque el almacén se reduzca a un tamaño de 10 posiciones, nunca se llenaría, ni siquiera en el peor de los casos.

Con la intención de encontrar casos particulares donde el almacén se llene, se decidió tomar en cuenta solamente 6 posiciones; de esta manera hay casos para los que el algoritmo no encontrará soluciones.

Es así que, tomando el ejemplo del peor de los casos planteado anteriormente, se observa en la siguiente figura, que cuando únicamente se tienen 6 posiciones, el almacén se llenará.

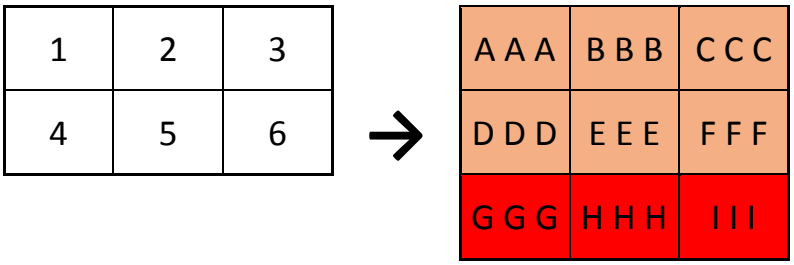

Figura 9. Ilustración de almacén lleno cuando se disminuyen el número de posiciones de almacenaje

Posiciones: 6

Elementos por posición: 3

Capacidad almacenamiento: 18 elementos

Máxima ocupación que podría a tener: 27

Porcentaje de ocupación: $\frac{27}{18} * 100 = 150\%$

5.3. Comprobación final

Una vez reestablecidas las condiciones del problema y se han redefinido las dimensiones del almacén a un total de 6 posiciones, se procede a probar el funcionamiento del algoritmo genético.

Las condiciones que se deben cumplir se detallan en la sección 1.1, con la excepción del tamaño del almacén, el cual hemos modificado. Para la comprobación final se han generado 4 pedidos aleatorios de 10 productos cada uno. Los pedidos se muestran de forma continua en la siguiente tabla.

Tabla 7. Tabla de 4 pedidos aleatorios de 10 productos cada uno.

Orden Original	Componentes			N. Pedido
1	G	A	C	1
2	H	F	G	1
3	A	C	G	1
4	I	E	B	1
5	G	D	F	1
6	A	F	A	1
7	I	F	G	1
8	I	D	G	1
9	D	I	B	1
10	F	D	I	1
11	H	G	B	2
12	A	A	E	2
13	H	H	A	2
14	B	D	H	2
15	H	I	F	2
16	H	C	D	2
17	E	A	F	2
18	E	C	D	2
19	D	C	I	2
20	I	F	G	2
21	G	F	B	3
22	I	G	H	3
23	I	B	C	3
24	A	D	E	3
25	D	F	F	3
26	E	H	A	3
27	A	E	G	3
28	G	G	B	3
29	D	A	B	3
30	B	A	H	3
31	I	H	C	4
32	B	A	H	4
33	A	B	I	4
34	H	H	D	4
35	G	G	I	4
36	B	E	E	4
37	B	I	A	4
38	I	D	D	4
39	A	F	A	4
40	A	I	I	4

En la siguiente figura 10 se muestra el valor fitness promedio de cada población en una dispersión de puntos color azul y, por otra parte, en color negro se indica el mejor fitness obtenido hasta el momento.

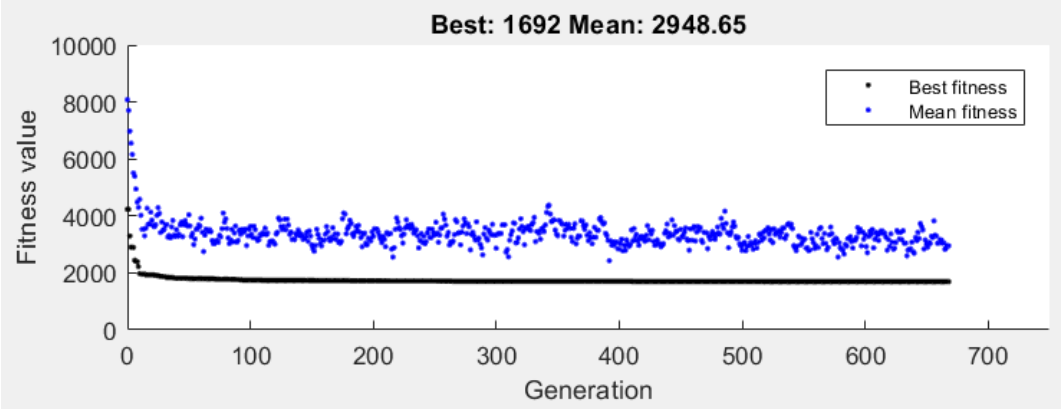

Figura 10. Fitness promedio de cada población (en azul) y el mejor fitness (en negro)

Se puede observar que han transcurrido cerca de 700 generaciones creadas hasta poder llegar a la respuesta más óptima. En la siguiente figura se muestra el avance del criterio de finalización respecto a las generaciones creadas.

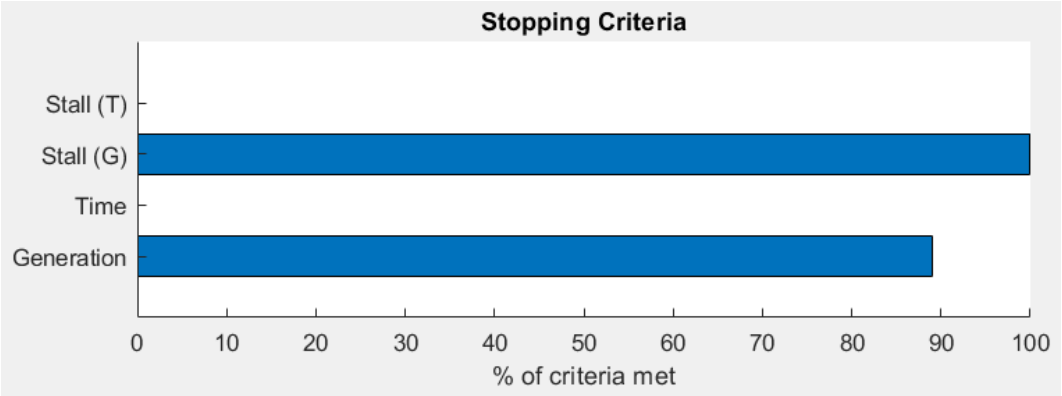

Figura 11. Avance de finalización respecto a generaciones creadas

Se puede evidenciar que, en casos de estudio grandes, es necesario analizar más generaciones para poder llegar a la solución más óptima. En este caso en particular, se ha completado cerca del 90% de generaciones programadas hasta encontrar la solución más óptima.

Tabla 8. Orden óptimo de productos para conseguir el menor residuo de componentes en el almacén.

Nuevo Orden	Componentes			N. Pedido	Ocupación Almacén
1	I	F	G	1	9
2	I	F	G	2	6
3	H	F	G	1	7
4	A	E	G	3	12
5	H	H	A	2	9
6	G	G	I	4	10
7	A	C	G	1	11
8	G	A	C	1	8
9	E	C	D	2	9
10	G	D	F	1	10
11	I	D	G	1	7
12	D	F	F	3	8
13	H	G	B	2	9
14	G	F	B	3	6
15	A	F	A	1	7
16	A	A	E	2	4
17	I	E	B	1	5
18	D	C	I	2	6
19	I	D	D	4	3
20	G	G	B	3	4
21	I	G	H	3	5
22	B	D	H	2	6
23	B	A	H	3	7
24	A	B	I	4	8
25	A	F	A	4	9
26	H	I	F	2	6
27	F	D	I	1	7
28	D	I	B	1	4
29	H	H	D	4	5
30	E	A	F	2	10
31	A	D	E	3	7
32	E	H	A	3	4
33	B	A	H	4	5
34	A	I	I	4	10
35	B	I	A	4	7
36	D	A	B	3	8
37	H	C	D	2	13
38	I	H	C	4	10
39	I	B	C	3	11
40	B	E	E	4	16

Se puede comprobar que la secuencia obtenida, optimiza de una forma muy aceptable la ocupación del almacén, donde los valores de ocupación acumulados nunca llegan a superar la capacidad máxima del almacén de 18 componentes residuales (3 por cada posición), acercándose al valor máximo únicamente al final del proceso de producción debido a que ya no se ocupa el último residuo.

6. Conclusiones

Después de desarrollar el presente trabajo, se ha podido identificar las principales ventajas al aplicar los Algoritmos Genéticos. Se requiere poca información del espacio de búsqueda al iniciar el proceso ya que se trabaja sobre un conjunto de soluciones, su implementación puede ser rápida y sencilla, y mediante la función de evaluación se pueden considerar las restricciones que sean de interés.

Mediante la aplicación diseñada con algoritmos genéticos para la tarea productiva propuesta, se ha podido comprobar la capacidad resolutiva de esta técnica frente a problemas complejos. Sin embargo, a medida que el problema crece en tamaño, se pueden observar valores fitness más elevados y más individuos posibles en cada población. Por ende, la creación de más generaciones para poder llegar al resultado más óptimo es necesaria.

Después de haber realizado diferentes pruebas al algoritmo genético y la función de evaluación diseñada sobre la aplicación de la tarea productiva, se ha podido verificar que a pesar de que el algoritmo converge, en más de un caso la solución óptima no siempre es correcta. Es necesario analizar el resultado obtenido ya que puede ser que la solución más óptima encontrada, llegue a saturar el almacén en algún punto de la secuencia de producción, en especial si el tamaño del almacén es reducido.

La función de evaluación o fitness implementada en el programa ha permitido comprobar, para las distintas secuencias de procesamiento de productos (individuos de cada población), que en el caso original donde se dispone de un almacén de 4x4 posiciones, y considerando las restricciones de precedencia, el almacén no logra llenarse en ningún caso. De esta forma, mediante los algoritmos genéticos, se podría incluso dimensionar de mejor manera el almacén acorde a las condiciones de producción en un proceso productivo cualquiera.

El estudio de problemas complejos mediante Algoritmos Genéticos resulta ser muy conveniente ya que analiza el problema de forma global, no local. Para lo cual, el operador de mutación se convierte en un elemento clave ya que, a pesar de generar una baja cantidad de individuos de la siguiente generación, estos permiten expandir el análisis a otras zonas donde aún no se ha realizado la búsqueda de la solución más óptima.

7. Bibliografía

Galipienso, A., & Isabel, M. (2001). Un modelo de integración de técnicas de CLAUSURA y CSP de restricciones temporales: Aplicación a problemas de Scheduling.

Haupt, R. L., Haupt, S. E., & Haupt, S. E. (1998). *Practical genetic algorithms* (Vol. 2). New York: Wiley.

MathWorks. (2018). Global Optimization Toolbox User's Guide (R2018a). Obtenido de https://www.mathworks.com/help/pdf_doc/gads/gads_tb.pdf

Palma Méndez, J. T., & Marín Morales, R. (2008). *Inteligencia Artificial.* Madrid: McGraw - Hill / Interamericana de España.

Sivanandam, S. N., & Deepa, S. N. (2007). *Introduction to genetic algorithms*. Springer Science & Business Media.