SEP **SES** **TNM**

INSTITUTO TECNOLÓGICO DE CHIHUAHUA II

“Implementación de programación genética para el diseño de laminados de materiales compuestos”

TESIS
PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

ING. CARLOS HUMBERTO RUBIO RASCÓN

DIRECTOR DE TESIS
M.C. RAFAEL VÁZQUEZ PÉREZ

CO-DIRECTOR DE TESIS
DR. ALBERTO DÍAZ DÍAZ

CHIHUAHUA, CHIH., DICIEMBRE DEL AÑO 2016

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

1.1 Introducción

La inteligencia artificial ha sido utilizada ampliamente en las últimas décadas para conseguir soluciones a problemas que los métodos convencionales no logran resolver. Entre las aplicaciones que se le ha dado a esta disciplina se encuentran los siguientes:

1. El reconocimiento y tratamiento de lenguaje natural.- Utilizar fuentes de lenguaje escritas u orales con el objetivo de obtener información útil o determinar una serie de características.
2. El reconocimiento y tratamiento de imágenes.- Obtener características de una o varias imágenes, tales como el color, los bordes, formas o tamaños para realizar tareas que de forma normal solamente podrían ser manejadas con el ojo humano.
3. La creación y mantenimiento de los sistemas basados en el conocimiento.- Utilizar fuentes de información tales como textos, páginas de internet o expertos en un tema en específico para generar un sistema capaz de tomar decisiones con un cierto nivel de precisión.
4. La optimización de recursos.- Crear algoritmos basados en comportamientos naturales para conseguir las mejores soluciones a un problema comenzando por un conjunto de posibles soluciones aleatorias.

Estas aplicaciones hacen que la inteligencia artificial sea una opción atractiva para solucionar problemas que se encuentran en cualquier área del conocimiento (matemáticas, química, física, etc.) y de la industria (aeroespacial, automotriz, estructural, etc.). Uno de los ámbitos que son beneficiados de estas técnicas es el diseño y simulación de materiales, ya que la necesidad de materiales para la construcción de edificios y productos que tengan mejor rendimiento a los existentes está siempre presente; y tal es el caso del área de los materiales compuestos.

El uso de materiales compuestos en áreas como la industria aeroespacial, de la construcción o la automotriz por mencionar algunas, es cada vez mayor, esto se debe a lo atractivo que resultan ser sus propiedades mecánicas tales como su resistencia y su rigidez. Un ejemplo de ello es la construcción

de autos deportivos, en donde se requieren materiales sumamente resistentes y capaces de soportar las altas cargas de presión sobre su estructura, pero que proporcionen el menor peso posible al vehículo para cumplir con los estándares de eficiencia y seguridad.

Un material compuesto está conformado por un material matriz (generalmente una matriz polimérica) que presenta propiedades mecánicas que permanecen constantes sin importar la dirección en la que se le aplique una fuerza, fusionado con un material de refuerzo, el cual puede ser manejado en forma de esferas, fibras unidireccionales o fibras con dirección aleatoria para mejorar las propiedades del material matriz. Entre estos tipos de inclusiones se utilizan con mayor frecuencia las fibras unidireccionales, dado que permiten mejorar las propiedades mecánicas de un material compuesto en la dirección en la que se hayan dispuesto sus refuerzos. (Jones, 1999)

La industria usa para sus proyectos lo que se conoce como materiales multicapas o laminados compuestos, los cuales se obtienen apilando varios materiales compuestos o varias capas del mismo material compuesto una y otra vez. Este proceso de apilado se realiza hasta que el conjunto de materiales compuestos llega a un tamaño óptimo donde las capas trabajen en conjunto, proporcionando las características adecuadas de rigidez en todas las direcciones requeridas y un peso adecuado para que pueda realizar un trabajo de forma eficiente.

Sin embargo, lo que pareciera un trabajo simple de conseguir un laminado compuesto que satisfaga las necesidades de un diseñador, se puede tornar en una tarea sumamente compleja, debido principalmente a que este tipo de materiales son conocidos como anisotrópicos. Es decir, si las cargas son aplicadas en la misma dirección de orientación del material refuerzo, las propiedades resultantes serán totalmente distintas a las que se obtendrían con una carga asignada en otra dirección. (HU, 2012). Por este motivo se busca que cada una de las capas de un material multicapa tenga una orientación del material refuerzo diferente, haciendo que el conjunto de materiales compuestos tenga las mejores propiedades en todas las zonas y orientaciones de aplicación de fuerza. Es de imaginar que para predecir el comportamiento del material resultante se deben de tomar en cuenta las propiedades de cada uno de los materiales involucrados en cada capa junto con su espesor y

orientación. Esto se traduce directamente en una serie complicada de fórmulas matemáticas que puedan estimar con gran precisión las propiedades finales.

Para resolver este tipo de situación, en el año de 2004 se presenta un software llamado MAC LAM (Mechanical Analysis of Composites and LAMinates), desarrollado en conjunto por el Centro de Investigación en Materiales Avanzados S.C. (CIMAV S.C.) en colaboración del Laboratoire Analyse des Matériaux et Identification (LAMI) de la Escuela Nacional de Puentes y Caminos (ENPC Ecole Nationale des Ponts et Chaussées) de París, Francia. Este software tiene entre sus principales virtudes ser de libre acceso, lo cual permite a cualquier entidad educativa utilizarlo con fines didácticos para que los estudiantes que lleven asignaturas de mecánica puedan comprender de forma más sencilla los conceptos de materiales compuestos y diseños multicapa, así como entrenarlos en el ambiente de diseño de este tipo de estructuras. Además, MAC LAM permite el cálculo de predicciones de comportamientos rápidamente y con una interfaz amigable, por lo que se pueden realizar varios casos de estudio en un espacio reducido de tiempo.

Si bien el programa ha tenido avances muy importantes y las ecuaciones y fórmulas aplicadas son efectivas, el campo de la inteligencia artificial puede aportar funcionalidades especiales a MAC LAM, lo cual lo convertiría en una herramienta de software sumamente poderosa en comparación con sus semejantes. La principal mejora que se le puede implementar a este software es la utilización de algoritmos de optimización que le permitan evaluar múltiples soluciones de diseños multicapa y seleccionar de manera automática las mejores de acuerdo a uno o varios criterios. Sin embargo, existen varios tipos de técnicas de optimización que se pueden implementar en los laminados compuestos. A continuación se presentan algunos ejemplos de dichas formas de optimizar:

- En 2002 se abordó el tema de la maximización de margen de falla de un laminado compuesto usando el algoritmo descendente interactivo, para el cual se planearon el uso de múltiples criterios de evaluación. (P. Kere, 2002)

- Utilizando el algoritmo de recocido simulado (simulated annealing), fue realizada una herramienta en el 2005 que optimiza el esfuerzo máximo en un multicapa con espesor constante. (S. Deng, 2005)
- Se implementó en el año 2010 una optimización del orden de capas de un laminado usando la distribución de probabilidad Lévy en un algoritmo de colonia de hormigas para maximizar la resistencia al pandeo. (R. Candela, 2010)
- En el 2013 fue desarrollada una herramienta basada en el algoritmo "Leaping frog" para optimizar el peso y el costo de los materiales de un laminado compuesto partiendo de sus capas ordenadas y variando su espesor y orientación. (M. Seyyed, 2013)
- Se realizó en 2014 una optimización de orientación de fibras en un laminado compuesto usando algoritmos genéticos con unas cuantas modificaciones de aspectos tanto de selección de parejas como en el cruce de generaciones. (S. Hwang, 2014)

Entre este tipo de programas destacan los algoritmos genéticos debido a su capacidad de experimentar solamente con las mejores soluciones que se van presentando para crear nuevas que se asemejen más a las buscadas. Esto permite una búsqueda de soluciones óptimas sin tener que realizar rastreos exhaustivos por todo el espectro de posibilidades. Además, la versatilidad de un algoritmo genético permite abarcar un amplio espectro de problemas específicos, tomando en cuenta las particularidades de cada uno de ellos para utilizarlas en la búsqueda de las mejores soluciones.

Entre otras, los algoritmos genéticos tienen las siguientes ventajas:

1. Puede optimizar con variables continuas o discretas.
2. No requiere información derivativa.
3. Busca simultáneamente en una amplia gama de posibles soluciones.
4. Puede manejar un gran número de variables.
5. Se puede acomodar para computadoras paralelas.
6. Optimiza variables sumamente complejas.
7. Provee una lista de soluciones óptimas, no solamente una.
8. Puede codificar variables para que la optimización se haga codificada.

9. Trabaja con datos generados numéricamente, datos experimentales o funciones analíticas. (Haupt, 2004)

Por las razones estipuladas anteriormente se concluye que la utilización de algoritmos genéticos para buscar los acomodos y orientaciones óptimas de materiales multicapa para obtener propiedades resultantes que obedezcan un criterio proporcionado por el usuario es una propuesta viable.

1.2 Planteamiento del problema

Para llevar a cabo el diseño de un laminado compuesto, normalmente el diseñador se basa en referencias literarias y su experiencia en este tipo de materiales para llegar a un acomodo multicapa preliminar, el cual es simulado para verificar sus propiedades resultantes, tales como la resistencia, la rigidez y los esfuerzos totales, para luego compararlos con los requeridos.

En caso de que el acomodo no cumpla las características necesarias, se tiene que volver a realizar el proceso ahora con otra configuración en el acomodo de capas, lo cual se traduce en tiempo que podría haberse utilizado para otras tareas. Para los diseñadores que cuentan con poca experiencia en este ámbito, esta actividad se vuelve especialmente extenuante porque sus simulaciones rara vez obtendrán los resultados que éste desea, ocasionando que se tengan que realizar más pruebas hasta que se encuentre el sistema adecuado.

Incluso para un experto en el diseño de materiales multicapa, el contar con una herramienta de software que le permita alcanzar sus objetivos de propiedades mecánicas y espesores en un menor tiempo supone una gran ventaja dado el ahorro de tiempo y la simplificación de la tarea. Por lo dicho anteriormente, la definición del problema puede ser planteada de la siguiente manera:

"Los diseñadores de laminados compuestos encuentran dificultad para encontrar laminados que se adecúen a sus necesidades debido a dos factores: la cantidad abismal de configuraciones posibles y la naturaleza anisotrópica de estos materiales."

1.3 Alcances y Limitaciones

1.3.1 Alcances

- El algoritmo debe encontrar el/los acomodos que se ajusten lo más posible a las propiedades mecánicas que haya indicado el usuario.
- De no existir por lo menos una solución exacta, el sistema debe encontrar el/los resultados que se asemejen más a dicho criterio con un porcentaje de satisfacción dado.

1.3.2 Limitaciones

- El tiempo que dure el algoritmo en mostrar resultados dependerá en gran medida de los recursos que tenga el equipo que lo esté ejecutando.
- Para utilizar el algoritmo, primero se debe tener instalado el software MAC LAM, ya que no se contempla en este proyecto de tesis el desarrollar un programa independiente.
- No se puede garantizar que se va a encontrar una solución que cumpla completamente con lo que haya pedido el usuario, puesto que en ocasiones las propiedades de los materiales proporcionados no tendrán forma de generar laminados compuestos que alcancen los las rigideces solicitadas por el usuario.

1.4 Justificación

El problema de la incertidumbre provocada por la cantidad de posibles combinaciones que pueden tener los materiales compuestos a un criterio de búsqueda determinado puede volverse tan extenso que, incluso después de semanas de trabajo no se logra encontrar la mejor solución para los requisitos que tiene el usuario y peor aún, si el diseñador no tiene la habilidad de detectar la característica o el dato que se debe cambiar para mejorar los resultados, la configuración óptima se puede escapar indefinidamente (Jones, 1999). Esto es especialmente cierto cuando el diseñador carece de experiencia en el área, no se cuenta con la documentación de trabajos anteriores con este material o si el número de capas y materiales es tan grande que la complejidad impide una predicción precisa del comportamiento final. Tanto si el objetivo del diseño tiene fines académicos al entrenar estudiantes de alguna facultad de ingeniería o si se buscan nuevas posibilidades para construir piezas en el ambiente industrial, el tiempo suele ser un factor fundamental, llegando incluso al punto en el que se pueden retrasar procesos completos debido a la falta de un laminado compuesto con propiedades satisfactorias.

Los algoritmos genéticos son una alternativa considerablemente efectiva para este tipo de problemas porque permite, con la ayuda de un sistema computacional, determinar y evaluar cientos o hasta miles de soluciones en un periodo de tiempo mucho menor al que tardaría una persona en hacer los análisis uno por uno. La mayor ventaja que tiene este tipo de algoritmos es la capacidad de ordenar las mejores soluciones que se van presentando en una matriz de datos y darles prioridad, dejando fuera los elementos que resultan poco satisfactorios. El hecho de que un procedimiento como es el de elegir los mejores datos posibles se reduzca de un tiempo indefinido a solamente un par de horas hace que los algoritmos genéticos sean una opción bastante atractiva a considerar.

Por último, la nueva versión del software MAC LAM, con el módulo de inteligencia artificial integrado seguirá a disposición de universidades de manera totalmente gratuita, lo que permitirá que alumnos de todo el mundo puedan beneficiarse de las capacidades que ofrece este programa y del algoritmo genético que se plantea en este documento sin verse obligados a una retribución monetaria.

1.5 Objetivos

1.5.1 Hipótesis

Mediante el uso de algoritmos genéticos es posible encontrar acomodos y orientaciones de capas en un modelo multicapa que permitan cumplir un criterio de resistencia, rigidez y/o esfuerzos totales en un tiempo menor al que se tardaría al realizarlo a mano.

1.5.2 Objetivo General

Desarrollar una herramienta de software que, mediante un algoritmo genético y partiendo de una lista de materiales compuestos (incluidas sus propiedades mecánicas) y sus espesores, encuentre laminados compuestos que satisfagan o se aproximen lo más posible al o a los criterios preestablecidos.

1.5.3 Objetivos específicos

- Desarrollar un algoritmo genético que encuentre modelos multicapas que se asemejen lo más posible a los criterios: El principal resultado esperado de la tesis es un algoritmo genético en el que se escojan los acomodos y orientaciones de capas de un laminado compuesto que cumplan o se aproximen a los valores que haya solicitado el usuario.
- Insertar el algoritmo como un módulo en la nueva versión del software: El algoritmo genético debe desarrollarse como una parte funcional de un programa más grande, brindándole nuevas capacidades a sus usuarios ya existentes y a los nuevos usuarios que lleguen en el futuro.
- Desarrollar una interfaz gráfica amigable para la entrada de datos: Es muy importante presentarle al usuario de manera intuitiva la información que debe proporcionar para que pueda funcionar el algoritmo y, posterior a la búsqueda, mostrarle los resultados del mismo en forma entendible.
- Verificar el comportamiento del algoritmo genético y buscar una ecuación con la que se pueda predecir, con cierto porcentaje de error tolerado, el número de individuos que debería tener el algoritmo genético para resolver un problema concreto, para finalmente comprobar la fiabilidad de dicha ecuación.

CAPÍTULO 2. ESTADO DEL ARTE

2.1 La inteligencia artificial en las áreas de la ciencia

La inteligencia artificial ha tenido sus altas y bajas de popularidad desde la década de los 70s; por una parte, la emoción que provocaba pensar en un sistema que emulara la mentalidad de un humano hacía de los sistemas inteligentes un concepto que se relacionaba con la ciencia ficción, pero dichas emociones generaban decepción en la industria cuando las expectativas que se les encomendaban no se cumplían con suficiente rapidez.

Después del rechazo que sufrieron los proyectos de sistemas inteligentes entre 1987 y 1993 por el fracaso de los sistemas expertos, nuevas ideas de solución de problemas que imitan comportamientos de la naturaleza comenzaron a extenderse apoyándose en el aumento exponencial en el poder de procesamiento y almacenamiento; entre dichas ideas destacaron los algoritmos genéticos por su flexibilidad, potencial de abarcar un amplio espectro de problemas y los buenos resultados que se obtenían.

Los nuevos conceptos que se iban generando fueron aprovechados por la mayoría de las áreas de la ciencia, tales como la química (Leardi, 2001) (Lavine & Moores, 2008) (Venkatasubramanian, Chan, & M., 1994), la biomédica (Ahmad & Bergen, 2010) (Bevilacqua, Mastronardi, & Piscopo, 2007) (Nozari, Rezai Rad, Pourmajidian, & Abdul-Wahab, 2011), entre otras.

2.2 Los algoritmos genéticos en el ambiente de los laminados compuestos

Prueba de esto surge en el año de 1995, cuando fue publicado un artículo llamado “Design of Laminate Composite Layups using Genetic Algorithms” (P. Sargent, 1995) en la revista Engineering with computers, en el cual se discuten tres maneras de diseñar la secuencia de apilado en laminados compuestos utilizando diferentes puntos de vista: métodos de generación y prueba, el recocido simulado y el algoritmo genético.

En el primero de los métodos mencionados se explican los principios de las funciones que evalúan las posibles soluciones del problema y asignan un solo número que determine cuáles de los estudios son

mejores que otros. Luego, se dedica a tomar en cuenta algunas variaciones en los algoritmos de selección de las mejores soluciones y de mejora de las mismas, entre los cuales destacan el algoritmo aleatorio y el algoritmo ambicioso (greedy algorithm).

Para el caso del recocido simulado, el autor realiza la comparación con respecto a los algoritmos anteriormente mencionados, determinando que resulta más conveniente el uso de este tipo de optimización cuando existen un gran número de mínimos locales, gracias a que se asigna una probabilidad a cada solución que no cumpla con las restricciones del estudio usando la distribución Bolzmann. Si una solución dada es solamente un poco peor que la solución actual, entonces una probabilidad relativamente alta es asignada para que dicha solución sea reemplazada. En cambio, si la solución resulta ser mejor que la actual, es una certeza que se tiene que reemplazar.

Al final la atención se torna hacia los algoritmos genéticos, para los cuales se hicieron estudios de eficiencia y robustez. Fue establecido que, si se tiene una cantidad de posibles soluciones a un problema (a la cual se le denomina N), los algoritmos genéticos pueden localizar las soluciones óptimas a dicho problema usando una población inicial menor a $N^{1/3}$, con lo que se reduce de manera sustancial el tiempo que tarda el algoritmo. Por otra parte, se define la importancia de determinar el tipo de cruce de genes entre los miembros de la población. En la *Figura 2.1* se pueden apreciar dos de los más utilizados.

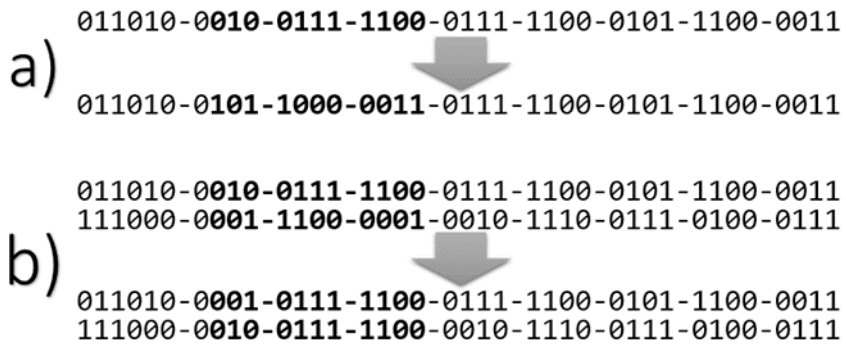

Figura 2.1. a) Ejemplo de inversión de genes. b) Ejemplo de cruce de genes. (Haupt, 2004)

El inciso a corresponde a la primera técnica de cruce, cuyo proceso consiste en tomar cierta cantidad de bits del individuo y simplemente invertirlos, lo cual tiene la ventaja de no requerir la intervención de otros miembros de la población, pero tiene un porcentaje de éxito bastante bajo. En cambio, el inciso b denota una forma más popular de realizar este proceso, y es el cruce entre dos individuos, intercambiando algunos bits entre ellos, creando así dos nuevas posibles soluciones que tienen una alta probabilidad de ser mejores que los individuos originales.

Tres años después, la misma revista publicó otro artículo, esta vez denominado "A Fortran 90 Genetic Algorithm Module for Composite Laminate Structure Design" (M. T. McMahon, 1998) en la que se describe a cierta profundidad el código que se siguió para diseñar un algoritmo genético para optimizar la estructura de un multicapa utilizando FORTRAN 90. La elección de dicho lenguaje de programación es que su antecesor (FORTRAN 77) fue usado muchas veces para realizar implementaciones de algoritmos genéticos para problemas específicos en el diseño de laminados y la nueva versión de FORTRAN contaba con nuevas características como la modularidad, capacidad mejorada en el manejo de arreglos, la habilidad de crear tipos de datos creados por el usuario, potencia para asignar memoria dinámicamente en tiempo de ejecución y, la más importante de todas, es retrocompatible con FORTAN 77, lo cual le permite modularizar segmentos de código ya realizados y mejorar su eficiencia con las nuevas técnicas de programación.

En ese proyecto se realizaron pruebas del nuevo sistema para comprobar su efectividad y su eficiencia en comparación con la implementación realizada totalmente en FORTRAN 77. Los resultados mostraron que ambos algoritmos son igualmente efectivos, pero siendo la nueva versión más lenta en lograr su objetivo que la anterior, lo cual se le atribuyó a que los estilos de programación de alto nivel que proporciona FORTRAN 90 proveen una plataforma más entendible e intuitiva para el ser humano, pero esas facilidades tienen un costo extra en el tiempo de ejecución por las operaciones extras que se tienen que realizar para convertir las instrucciones nuevas a sus contrapartes de bajo nivel.

Para el año 2000 fue desarrollada una herramienta para optimizar el ordenamiento de las capas del laminado especializado en recubrimientos (S. Rajasekaran, 2000), tomando la orientación de las capas

como una variable discreta, es decir, que solamente se toman en cuenta un número limitado de opciones.

Además, se limitaron a elegir espesores de capas constantes para disminuir la complejidad del algoritmo; en la *Tabla 2.1* se aprecian las primeras dos iteraciones del algoritmo genético empleado en esa publicación, especificando los miembros codificados de la población y sus atributos.

Tabla 2.1. Ejemplo de dos generaciones.

#	Población	Ángulos	Frec	Fitness	Cont	Pila	Pareja	CS1	CS2	Cruce
					Generación 1					
1	11100 00110	10,75,10	50.84	0.85	1	11111 00110	7	1	5	10000 00110
2	10110 00100	75,45,75	51.97	0.87	1	10110 00100	3	3	4	10110 00100
3	11110 10110	30,10,30	38.69	0.65	0	11110 10110	2	3	4	11110 10110
4	01011 10010	90,20,90	58.12	0.97	1	01010 10010	6	2	5	01001 10010
5	00001 10110	-45,75,-45	57.67	0.96	1	01110 10110	8	3	4	01100 10110
6	00000 11110	30,-75,30	54.51	0.90	1	00001 11110	4	2	5	00010 11110
7	11110 10110	-75,75,-75	59.98	1.00	2	10000 11101	1	1	5	11111 10110
8	10110 11101	75,-60,75	58.49	0.98	1	10100 11101	5	3	4	10110 11101
					Generación 2					
1	10000 00110	-80,75,-80	60.25	0.98	1	10000 00110	8	3	5	10010 00110
2	10110 00100	75,45,75	51.97	0.85	1	10110 00100	7	1	5	11111 00100
3	11110 10110	-75,75,-75	59.98	0.97	1	11110 10110	4	1	5	11100 10110
4	01001 10010	-20,20,-20	39.37	0.64	0	01100 10110	3	1	5	01110 10110
5	01100 10110	-75,75,-75	61.38	1.00	2	01100 10110	6	3	4	01100 10110
6	00010 11110	10,-75,10	52.44	0.85	1	00010 11110	5	3	4	00010 11110
7	11111 10110	0,75,0	49.18	0.80	1	11111 10110	2	1	5	10110 10110
8	10110 11101	45,-60,45	48.30	0.78	1	10110 11101	1	3	5	10100 11101

La revista “International journal of mechanics and materials in design” en su primera edición en 2004 publicó un artículo denominado “On the determination of mechanical properties of composite laminates using genetic algorithms” (C. Maletta, 2004), el cual presenta un problema de optimización de las propiedades mecánicas de un laminado compuesto.

La peculiaridad de ese proyecto radica en el hecho de que proponen la evaluación de las posibles soluciones en el resolvedor de elemento finito NASTRAN para cada uno de los individuos de la población, en cada una de las iteraciones del algoritmo genético. En la *Figura 2.2* se aprecia el diagrama de flujo correspondiente al análisis de la población, tomando especial consideración en el procesamiento de elementos finitos.

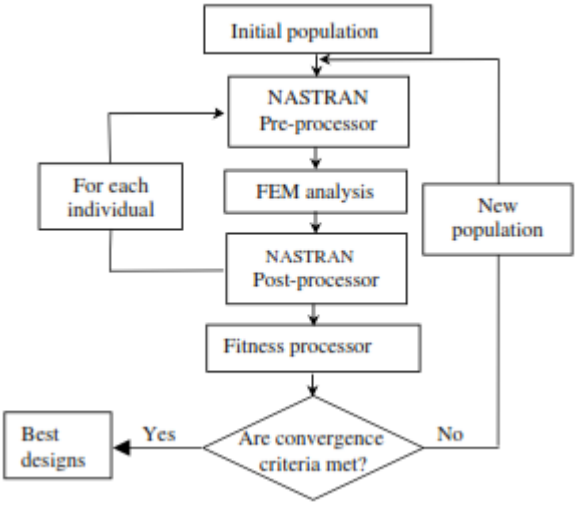

Figura 2.2. Diagrama de flujo con NASTRAN.

Por mencionar uno de los estudios más recientes del tema, en el año 2014 mediante el artículo "A genetic algorithm for the optimization of fiber angles in composite laminates" se dio a conocer un algoritmo genético con una modificación en la selección de generaciones llamado comparación de élite.

Ciertamente los algoritmos genéticos tienen un amplio historial de resolver problemas relacionados con los laminados compuestos, aunque cada uno esté hecho para la solución de un caso específico. Sin embargo, no fueron encontrados antecedentes de una herramienta que aplicara algoritmos genéticos para problemas multiobjetivo con la flexibilidad que se plantea en el presente proyecto de tesis.

CAPÍTULO 3. MARCO TEÓRICO

El problema que se busca resolver en este proyecto de tesis parte de dos áreas de conocimiento, las cuales deben comprenderse para que una evaluación competente pueda llevarse a cabo y así obtener mejores resultados.

El primero de estos temas es comprender la teoría básica de los materiales compuestos y, específicamente, la forma en la que se calculan las propiedades mecánicas de un modelo multicapa, ya que éstas son la base de los criterios de evaluación que utilizará el algoritmo. La segunda parte corresponde al manejo de un algoritmo genético con los ajustes que se le tengan que hacer para que dicho algoritmo lleve a cabo su tarea de la mejor manera posible.

3.1 Cálculos de laminados compuestos

Las propiedades mecánicas de un laminado compuesto se obtienen mediante cálculos matriciales; esta tesis se enfoca en las rigideces del laminado, las cuales se estiman de la siguiente manera:

- La matriz ABBD y su inversa. El comportamiento generalizado del multicapa se escribe:

$$\begin{pmatrix} N_x \\ N_y \\ N_{xy} \\ M_x \\ M_y \\ M_{xy} \end{pmatrix} = \begin{pmatrix} \tilde{\tilde{A}} & \tilde{\tilde{B}} \\ \tilde{\tilde{B}} & \tilde{\tilde{D}} \end{pmatrix} \cdot \begin{pmatrix} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \\ \chi_x \\ \chi_y \\ 2\chi_{xy} \end{pmatrix} \quad (3.1)$$

  donde $\tilde{\tilde{A}}$, $\tilde{\tilde{B}}$ y $\tilde{\tilde{D}}$ son matrices de 3 X 3 que representan las rigideces del laminado compuesto actual; N_x, N_y y N_{xy} son los esfuerzos de membrana; M_x, M_y y M_{xy} son los momentos en el plano del laminado, ε_x, ε_y y ε_{xy} son las deformaciones generalizadas en el plano, mientras que χ_x, χ_y y χ_{xy} son las curvaturas.

- La matriz de rigideces fuera del plano $\tilde{\tilde{F}}$ y su inversa. El comportamiento generalizado del multicapa da lugar a esta matriz:

$$\begin{pmatrix} Q_x \\ Q_y \end{pmatrix} = \tilde{\tilde{F}} \cdot \begin{pmatrix} 2\varepsilon_{xz} \\ 2\varepsilon_{yz} \end{pmatrix} \quad (3.2)$$

donde Q_x, Q_y y Q_{xy} son los cortantes, ε_{xz} y ε_{yz} son las deformaciones generalizadas asociadas a dichos esfuerzos.

- Los módulos de Young aparentes E_X y E_Y en el sistema de coordenadas global (*x*, *y*, *z*).
- Los módulos de esfuerzo cortante aparentes G_{XY} en el sistema de coordenadas global (*x*, *y*, *z*).
- Los módulos de flexión aparentes E_{XB} y E_{YB} en el sistema de coordenadas global.
- Los coeficientes de Poisson aparentes ν_{XY} y ν_{YX} en el sistema de coordenadas global.
- Los coeficientes de dilatación térmica aparentes α_X, α_Y y α_{XY} en el sistema de coordenadas global.
- Los coeficientes de dilatación por absorción de humedad aparentes β_X, β_Y y β_{XY} en el sistema de coordenadas global. (Graeme, 2002)

Debido a la cantidad de posibles combinaciones que puede tener un laminado multicapa tanto en el orden de sus capas como sus orientaciones, el diseño de este tipo de materiales se convierte en un problema de optimización. Para lograr discernir con eficacia la forma en la que se debe resolver este problema, primero se debe definir el tipo de optimización que se requiere.

3.2 Optimización

La optimización se define como el proceso de mejorar un producto o una situación, y consiste en intentar variaciones de un concepto inicial para usar la información obtenida en mejorar la idea original. Una computadora es la herramienta perfecta para la optimización siempre y cuando la idea o variable que influye en la idea pueda ser introducida en formato digital, ajustando las entradas de datos para encontrar la salida máxima o mínima. Las entradas consisten de variables; el proceso que determina qué tan buena es una solución se conoce como función costo o función objetivo y la salida de datos es el costo.

Existen seis categorías que dividen los algoritmos de optimización. Ninguna de estas seis vistas o sus ramas son mutualmente excluyentes en su totalidad. Por ejemplo, un problema de optimización dinámica puede o no tener restricciones. Además, algunas de las variables pueden ser discretas y otras continuas. La *Figura 3.1* muestra las seis categorías con sus ramificaciones, las cuales se explican a continuación:

Figura 3.1. Categorías de optimización. (Haupt, 2004)

1. La optimización a prueba y error se refiere al proceso de ajustar variables que afectan la salida de datos sin saber mucho acerca del proceso que las produce. Un ejemplo simple es ajustar la antena doble de una televisión para obtener la mejor recepción de imagen y sonido. Un ingeniero solamente puede adivinar el por qué ciertas posiciones de las antenas resultan en una mejor imagen y las demás. Los experimentalistas prefieren este método. Muchos grandes descubrimientos, como el refinamiento de la penicilina como antibiótico, fueron resultado de la optimización a prueba y error. En contraste, una fórmula matemática describe la función objetivo en la optimización por función. Varias manipulaciones matemáticas de la función llevan a la solución óptima. Los teóricos utilizan mucho este método.

2. Si solamente hay una variable, la optimización lleva una dimensión. Un problema que tiene más de una variable es de múltiples dimensiones. La optimización se vuelve más difícil conforme aumenta el número de dimensiones. Muchas optimizaciones multidimensionales se pueden generalizar a una serie de aproximaciones de una dimensión.

3. Optimización dinámica significa que el resultado está en función del tiempo, mientras que la estática es independiente del tiempo. Cuando se vive en los suburbios de Boston, existen algunas maneras de conducir hacia el trabajo. ¿Cuál es la mejor ruta? Desde el punto de vista de distancia, el problema es estático, y la solución puede ser encontrada usando un mapa o el odómetro del vehículo. En la práctica, este problema no es simple por la cantidad gigantesca de variaciones en la ruta. La ruta más corta no es necesariamente la más rápida. Encontrar la ruta más rápida es un problema dinámico porque la solución depende de la hora del día, el clima, los accidentes automovilísticos, entre otras variantes. Un problema estático puede ser difícil de resolver, pero añadir la dimensión del tiempo incrementa el reto de resolver un problema dinámico.

4. Un problema de optimización puede ser clasificado también si tiene variables discretas o continuas. Las variables discretas tienen un número finito de valores posibles, mientras que los valores que puede tomar una variable continua son infinitos. Si se tiene que decidir en qué orden hacer una serie de tareas en una lista, la optimización discreta (también conocida como optimización combinacional) es usada. Sin embargo, si se trata de encontrar el valor mínimo de f(x) en una línea de números, es más apropiado ver el problema como continuo.

5. Las variables a veces tienen límites o restricciones. La optimización con restricciones incorpora igualdades y desigualdades dentro de una función costo. La optimización sin restricciones permite que las variables tengan cualquier valor. Una variable restringida puede ser convertida en una no restringida a través de una transformación de variables. La mayoría de las optimizaciones numéricas funcionan mejor con variables no restringidas. Considere como ejemplo la minimización de $f(x)$ restringido al intervalo $-1 < x < 1$. La variable convierte a x en una variable u no restringida haciendo que $x = \sin(u)$ y minimizando $f(\sin(u))$para cualquier valor de u. Cuando la optimización restringida formula variables en términos de ecuaciones lineales y restricciones lineales, se le conoce como programa lineal. Cuando las funciones de costo o las restricciones dejan de ser lineales, el problema se convierte en un programa no lineal.

6. Algunos algoritmos tratan de minimizar el costo comenzando con un conjunto inicial de valores en las variables. Estas búsquedas fácilmente se atoran en mínimos locales pero tienden a terminar rápido. Estos son los algoritmos de optimización tradicionales y son basados generalmente en métodos de cálculo. Moviéndose de un conjunto de variables a otro se basa en alguna secuencia de pasos establecida anteriormente. Por otra parte, los métodos aleatorios usan cálculos probabilísticos para encontrar conjuntos de variables. Tienden a ser más lentos pero suelen ser más exitosos en encontrar un mínimo global.

Entre los algoritmos de optimización que existen en la actualidad, hay una sección de ellos que es especialmente efectivo contra problemas que resultarían ser demasiado complejos para la mayoría de los enfoques, y son los métodos de optimización naturales (Haupt, 2004), los cuales tratan de emular comportamientos que pueden ser encontrados en animales, insectos o plantas. Algunos de estos métodos incluyen los algoritmos genéticos (Holland, 1975), el recocido simulado (Kirkpatrick, 1983), la optimización de enjambre de partículas (Parsopoulos, 2002), la optimización de colonia de hormigas (Dorigo, 1997) y los algoritmos evolutivos (Schwefel, 1995). Estos métodos generan nuevos puntos en el espacio de búsqueda aplicando operadores a los puntos actuales y moviéndose hacia un lugar mejor con métodos estadísticos. Estos algoritmos no requieren derivar la función costo y por lo tanto pueden trabajar con variables discretas y funciones no continuas.

Los algoritmos genéticos han sido utilizados desde hace ya varias décadas y han probado ser muy efectivos en la mayoría de los problemas de optimización. Debido a que este método es utilizado en el presente proyecto, es conveniente explicar un poco más a detalle su funcionamiento y sus principales características para un mejor entendimiento de los puntos con los que se debe tener mayor cuidado en el proceso de optimización.

3.3 Algoritmos genéticos

El algoritmo genético (AG) es una técnica de optimización y búsqueda basada en los principios de la genética y la selección natural. Un AG permite una población compuesta por muchos individuos para

que "evolucionen" bajo ciertas reglas de selección hasta llegar a un estado que maximice su función aptitud (también se conoce como minimizar el costo).

Este tipo de algoritmos ha demostrado con el paso de los años una gran capacidad de resolver problemas sumamente complejos disminuyendo las zonas de búsqueda y además, se obtiene como resultado una lista de las mejores soluciones que se encontraron, no solamente una. (Forrest, 1993)

En la *Figura 3.2* se muestra una simplificación del proceso que conlleva un algoritmo genético comparándolo con la selección natural. Ambos comienzan con una población inicial de individuos (posibles soluciones) generados aleatoriamente. Cada secuencia de datos representa características seleccionadas de un perro en la población. Las propiedades de los ladridos son codificadas de forma binaria para formar lo que se conoce como el "cromosoma" asociado a un perro en particular. Si tratamos de criar al perro con el ladrido más fuerte, entonces solamente algunos de los más ruidosos se quedan para la crianza. Debe existir una manera de determinar los ladridos más fuertes, como por ejemplo, poner a los perros a probar para medir el volumen de su ladrido, asignándoles bajos costos a los miembros más aptos (en este caso, los perros con un ladrido más fuerte). Después, se ordenan los individuos de la población de más apto a menos apto, para luego utilizar un algoritmo de selección para elegir parejas de miembros que cruzarán su material genético, con lo que se obtienen dos cachorros que probablemente tengan ladridos fuertes, dado que ambos padres tienen genes propicios para eso. Estos cachorros reemplazan a los peores perros de la población inicial para mejorar las probabilidades de que las siguientes generaciones de cachorros sean todavía mejores en comparación a los que se encuentran actualmente en la población.

Iterando el proceso mencionado anteriormente lleva a obtener un grupo de perros que ladran realmente fuerte (es decir, con un costo mínimo). Esta optimización natural también puede ser aplicada a procesos de optimización de objetos inanimados. (Haupt, 2004)

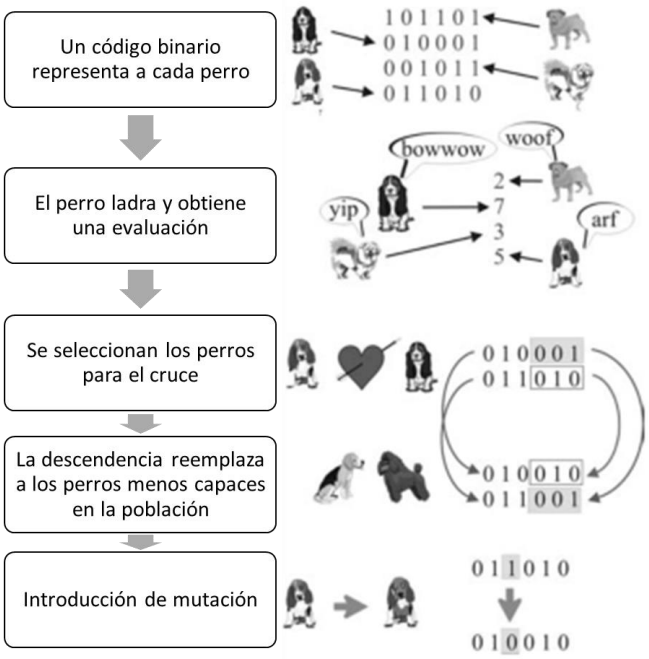

Figura 3.2. Analogía entre un algoritmo genético y la genética biológica. (Melián, Moreno, & Moreno, 2009)

Una vez explicado el proceso básico de un algoritmo genético de manera sencilla, es conveniente definir un poco más a fondo cada uno de los pasos que representa una búsqueda con este tipo de algoritmos (ver *Figura 3.3*).

- Definir función costo, las variables y los parámetros del AG.- Una función costo genera una salida a partir de una serie de variables de entrada (cromosoma). Dicha función puede ser una expresión matemática, un experimento o un juego. El objetivo es modificar las variables de entrada para obtener mejores valores de salida.
- Generar población inicial.- El algoritmo se encarga de crear una cierta cantidad de individuos (posibles soluciones al problema) de manera aleatoria. A este conjunto de individuos se le conoce como población inicial.

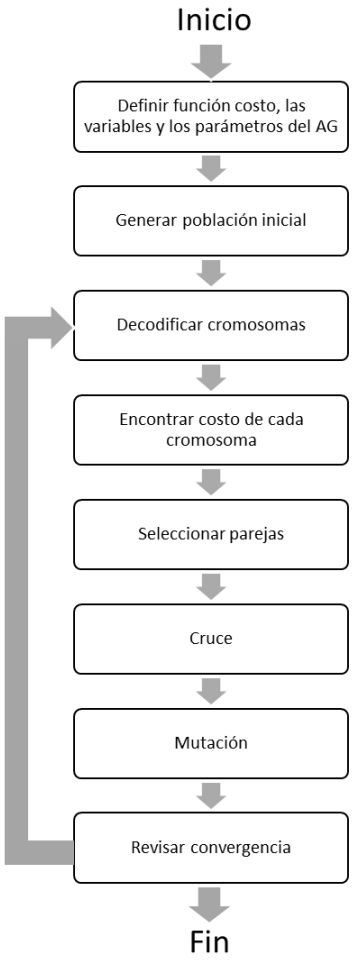

Figura 3.3. Diagrama de flujo general de un algoritmo genético. (Gestal & Daniel, 2010)

- Decodificar cromosomas.- Recordemos que cada individuo es representado por una cadena genética, la cual contiene las características que se consideran para evaluar el costo así que, a fin de encontrar el costo del individuo hay que decodificar cada cromosoma.
- Encontrar el costo de cada cromosoma.- Este paso consiste en aplicar la función a cada individuo de la población para obtener su costo y ordenarlos de menor a mayor para tener las mejores soluciones al inicio de la lista.
- Seleccionar parejas.- Utilizando un criterio definido se obtienen parejas de individuos que serán cruzados.

- Cruce.- Se mezcla el material genético de cada pareja para crear dos nuevos descendientes, los cuales son evaluados y pasan a reemplazar a los peores miembros de la población. Después, la población se vuelve a ordenar de mejor solución a peor.
- Mutación.- Un paso opcional en un AG es establecer una probabilidad muy baja de que un descendiente del cruce cambie un pequeño segmento de su código genético para aumentar la diversidad de la población.
- Revisar convergencia.- Utilizando métodos estadísticos se comprueba el nivel de cambio entre la iteración actual y la anterior. Si el cambio es menor a una tolerancia determinada, el algoritmo termina. De lo contrario, se vuelve al paso de "Decodificar cromosoma" y se realiza otra iteración.

Por supuesto, el proceso descrito anteriormente representa la forma más básica de realizar con éxito una implementación del algoritmo genético. Existen múltiples variaciones que se le pueden realizar para disminuir el tiempo de convergencia en un problema específico, por lo que siempre se debe de tomar en cuenta que es poco probable que un algoritmo que resolvió un problema en el pasado siga sirviendo para los demás proyectos.

Por regla general, los algoritmos genéticos son clasificados por el tipo de los cromosomas que constituyen su código genético:

1. Algoritmos genéticos binarios.- Su cromosoma consiste únicamente de unos y ceros, por lo que casi siempre se tratarán con la codificación/decodificación de las características de los individuos de la población.
2. Algoritmos genéticos continuos.- Pueden manejar datos numéricos en cualquier expresión. Debido a su naturaleza, es posible (pero no recomendado) que se realicen las iteraciones del algoritmo sin tener que codificar las variables que componen a los miembros de la población.

La principal razón por la que se prefiera utilizar algoritmos genéticos binarios para la mayoría de los problemas de optimización es que favorece la aleatoriedad de los métodos. (Goldberg, 1989)

CAPÍTULO 4. DESARROLLO

4.1 Modelado del negocio

El inicio de todo proyecto es la reunión entre el cliente o solicitante y la persona o el equipo de trabajo para discernir los objetivos generales que se pretenden cumplir con dicho proyecto. En este caso el solicitante es el Dr. Alberto Díaz Díaz, quien funge además como codirector de tesis. En la primera reunión con el doctor se abarca el tema de la existencia del software gratuito MAC LAM para el análisis de materiales compuestos y materiales multicapa, explicando que los apilados de materiales compuestos pueden ser complicados de diseñar por la dificultad para predecir sus propiedades resultantes. Por otra parte, se determinó que el propósito principal del software es el de apoyar a los docentes que imparten materias relacionadas con materiales compuestos en el proceso de aprendizaje en el diseño de los materiales multicapa.

Además del Dr. Díaz, las reuniones tuvieron lugar con el M.S.C. Rubén Castañeda Balderas, quien fue el encargado de programar la interfaz gráfica de las primeras dos versiones de MAC LAM, por lo que se tuvo la oportunidad de aclarar dudas con respecto al proceso que se debe llevar a cabo, lo cual auxilia en modelado de los objetivos. La entrevista realizada con el Ing. Rubén se puede ubicar en el ANEXO A. ENTREVISTA. Gracias a las reuniones y a la entrevista se dio forma al diagrama de objetivos que se muestra en la *Figura 4.1*.

Figura 4.1. Diagrama de objetivos.

Debido a que MAC LAM es un software gratuito y está enfocado al ambiente didáctico, el modelado del negocio hace referencia solamente a cumplir los objetivos del proyecto, ya que no se planea una estrategia para conseguir remuneración del sistema.

4.2 Determinación de requerimientos

Posteriormente, las actividades se centraron en organizar la información recopilada en la etapa anterior con el fin de obtener una idea precisa de las necesidades reales del programa. El propósito de este proceso es el de plasmar en diagramas los distintos módulos del software, así como la funcionalidad que debe de tener cada uno de estos para llegar a la meta final.

A la par con el trabajo de análisis obtenido por medio de la observación y las entrevistas, el Dr. Alberto Díaz realizó una serie de peticiones fundamentales se deben cubrir en el desarrollo del software, las cuales son:

- El usuario debe proporcionar una lista de materiales de los cuales el algoritmo podrá construir los individuos de la población inicial.
- Para cada problema de optimización, el usuario podrá seleccionar una combinación de propiedades que desee buscar.
- El software debe contar con una interfaz para que el usuario pueda seleccionar capas y propiedades a optimizar, así como otra para mostrar los resultados del algoritmo, en la cual debe aparecer una lista con los mejores apilados encontrados por el algoritmo genético.
- El programa debe ser capaz de crear un reporte de dichos resultados para que puedan ser leídos posteriormente si el usuario así lo desea.

Respetando las solicitudes antes mencionadas, se hizo una lista con las acciones principales que debe realizar el software. La idea general del software se plasma en el diagrama de casos de uso de la *Figura 4.2*.

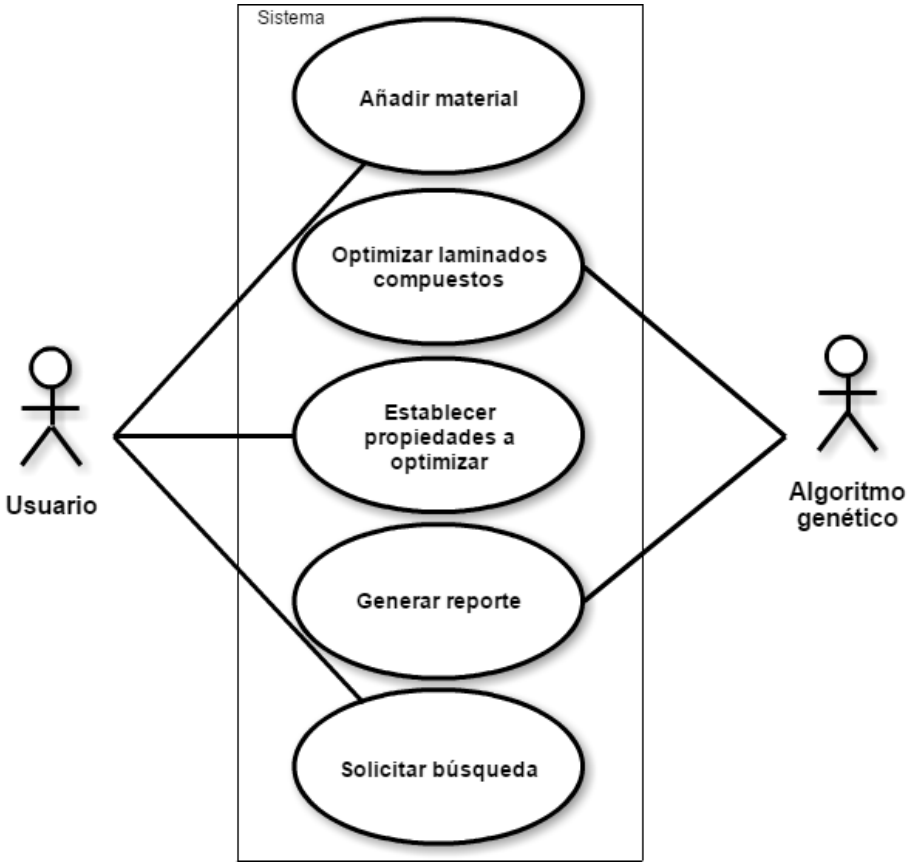

Figura 4.2. Diagrama de casos de uso.

El usuario puede solicitar una o varias de las siguientes propiedades del laminado compuesto para optimizar (maximizar, minimizar, acercar a un valor, buscar que sea menor o mayor a un valor o que esté entre dos valores):

- Módulos de Young aparentes (E_X, E_Y).
- Módulo cortante aparente (G_{XY}).
- Módulos de flexión aparentes (E_XB, E_YB).
- Coeficientes de Poisson aparentes (ν_{XY}, ν_{YX}).
- Coeficientes de dilatación térmica (α_X, α_Y, α_{XY}).
- Coeficientes de dilatación higrotérmicas (β_X, β_Y, β_{XY}).
- Uno o varios de los coeficientes de la matriz ABBD.
- Uno o varios de los coeficientes de la matriz F.

4.3 Análisis

Los diagramas y la redacción del modelado del negocio y la determinación de requerimientos, aunque son fundamentales para el entendimiento básico del proyecto y son un gran apoyo en la comunicación con el cliente, carecen de una visión técnica y de especificaciones más detalladas, ambas características necesarias para realizar la transición entre el funcionamiento teórico y el código real.

Por lo tanto, el proceso de análisis resulta necesario para identificar los problemas que puede traer el proyecto y seleccionar la forma óptima de resolverlos antes de que éstos se presenten. Los aspectos que más requieren de la atención del desarrollador son los correspondientes a la implementación del algoritmo genético. En la *Figura 4.3* se presentan de manera gráfica las partes que se deben tomar en cuenta.

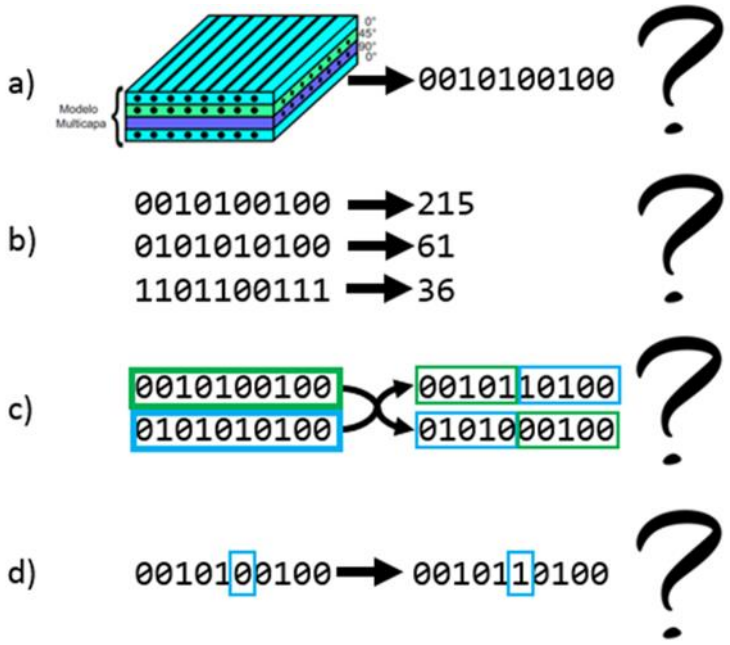

Figura 4.3. Secciones del problema de optimización a tomar en cuenta.

Las secciones del problema desplegados en la figura anterior son explicados de manera más detallada a continuación:

a) El primer elemento a considerar es la forma en la que se representarán los apilados compuestos en la estructura de datos, puesto que es recomendable codificar sus propiedades para que sea más sencillo realizar el cruce de cromosomas.
b) El segundo punto es determinar la función con la que se sabrá cuál apilado es mejor que los demás, teniendo en cuenta los datos proporcionados.
c) Después, se debe elegir o crear un algoritmo para cruzar los genes de los individuos de la población para generar nuevas posibles soluciones con cada generación que pase.
d) Por último, se necesita definir si se va a usar o no un elemento de mutación y, en caso afirmativo, la frecuencia en la que se va a hacer.

4.3.1 Representación del cromosoma

A pesar de que el proceso para crear y resolver un laminado compuesto puede llegar a ser complejo, el proceso de convertir un laminado compuesto en una cadena de unos y ceros es realmente simple, debido a que solamente se deben usar características que puedan ser modificadas sin comprometer datos adicionales. Para este problema de optimización, se cuenta con una lista de materiales disponibles para que sea usada por el algoritmo genético; las propiedades de cada material compuesto no son tomadas en cuenta por el algoritmo para la conversión porque ya no se estarían manejando las restricciones proporcionadas por el usuario, y las propiedades finales del apilado servirán para determinar el costo de un individuo dado. Por otra parte, el índice de cada material compuesto disponible en la lista y la orientación en grados en la que se posiciona son las dos propiedades que pueden ser convertidas a unos y ceros y manejadas sin afectar los datos solicitados por el usuario, por lo que fueron elegidos para formar parte del cromosoma de cada individuo.

Hay que tomar en cuenta otro aspecto importante de los algoritmos genéticos: la longitud del cromosoma. Si el problema que se intenta resolver termina ocasionando que los individuos de la población tengan un cromosoma de diferente tamaño entre sí, esto puede aumentar considerablemente la complejidad de dicho problema, dado que el cruce de información para obtener nuevas generaciones de individuos deja de ser sencillo para convertirse en un reto al tener que considerar la longitud de cada cromosoma y elegir cuáles datos se transfieren. Por lo anterior, se recomienda definir el problema

de tal manera que todos los cromosomas de una ejecución del algoritmo genético sean del mismo tamaño.

Además, existe otra desventaja para el uso del número de capas variable, y es que la mayoría de las variables a optimizar crecen en proporción con el número de capas del apilado. Si el usuario desea maximizar una de estas propiedades y el sistema soporta un número flexible capas, el algoritmo genético tiende a agregar cada vez más capas de material con el fin de aumentar las propiedades solicitadas, en lo que puede terminar como un ciclo infinito.

Para abordar esta situación, se le permitirá al usuario definir un número máximo de capas para cada problema de optimización. De esta manera, el tamaño del cromosoma podrá ser diferente en cada ejecución del algoritmo, pero todos los cromosomas dentro de una sola ejecución tendrán el mismo tamaño. Esto evita los dos riesgos que se mencionaron anteriormente.

El último factor a tratar en esta sección es la manera en que los datos serán codificados y acomodados en el cromosoma. Los genes pueden ser ordenados como números decimales para ser más entendibles para el ser humano, pero esto no es relevante para el algoritmo, donde su eficacia es el aspecto con mayor prioridad en el proyecto. Otra manera para la representación de datos en los algoritmos genéticos es la codificación binaria, es decir, convertir los datos de un individuo usando solamente unos y ceros.

Como se definió al inicio de esta sección, los valores del apilado que en realidad deben ser plasmados en el cromosoma por cada capa son el índice de algún material disponible en la tabla junto con su ángulo en grados. Las capas se agrupan una sobre otra, por lo que la secuencia del cromosoma puede ser determinada fácilmente. Por lo tanto, la codificación de los apilados puede ser representada en binario sin mayor dificultad, haciendo que un algoritmo genético binario sea una buena solución.

La cantidad de bits que tendrán los cromosomas de una ejecución dependerá de tres factores:

1. El número de capas que debe tener el apilado del problema de optimización.

2. La cantidad de materiales disponibles que existan, dado que sus índices deben ser representados en binario. Se usará la menor cantidad de bits que pueda almacenar el número de materiales proporcionados.
3. La cantidad de bits necesarios para almacenar el rango permitido para el ángulo de cada capa.

En la *Figura 4.4* se puede apreciar un ejemplo de un apilado con sus datos y su representación binaria.

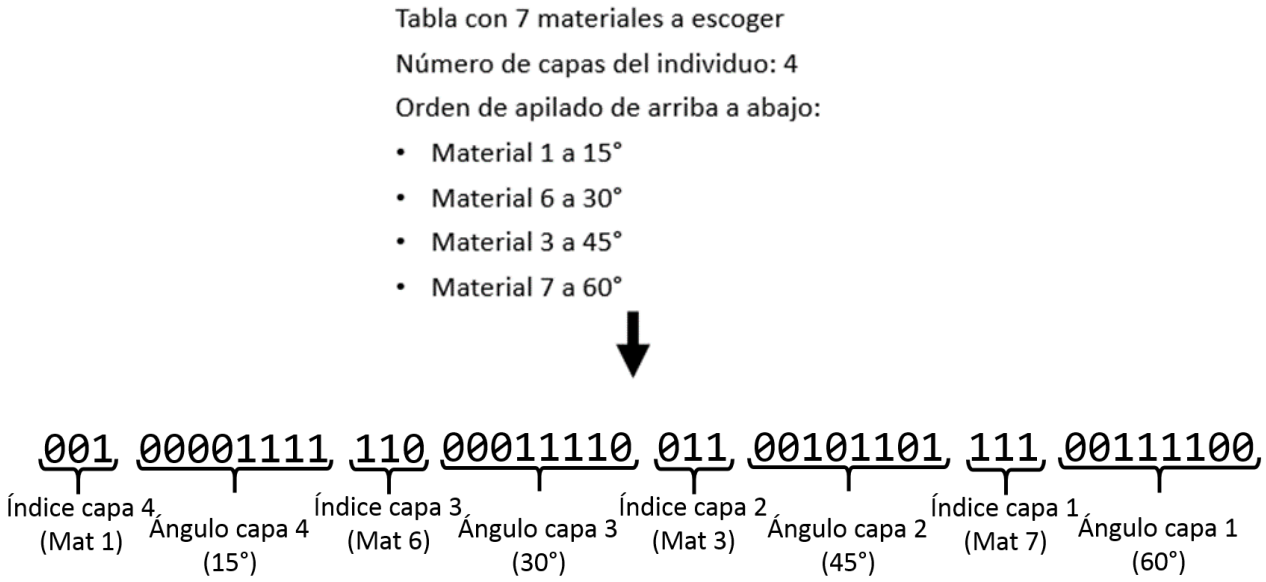

Figura 4.4. Representación de cromosoma en código binario.

En la imagen anterior se puede apreciar que el cromosoma final será codificado en binario y tiene un tamaño estático, definido por el número máximo de capas elegido por el usuario. Además, el cromosoma tiene el siguiente orden: para cada capa del apilado en orden descendente se especifica el índice del material de la tabla (la tabla de este ejemplo tiene 7 materiales, por lo que bastan tres bits de información para establecer ese dato), seguido de su orientación en grados, para el cual se asignan 8 bits (uno para un signo y 7 para albergar números entre cero y 90) con lo que se representan todos los ángulos utilizados para las capas.

Debido a que es posible que los bits puedan contener valores mayores a los válidos, se implementaron instrucciones que obtienen el valor proporcional válido a partir del valor obtenido. Es decir, si el

algoritmo admite un valor de 127 (el valor máximo), este se convierte al valor máximo permitido (90°).

4.3.2 Función costo

Una de las partes más importantes de los algoritmos genéticos es la definición de la función costo (o fitness), con la cual se evalúan todos los individuos de la población en todas las generaciones para asignarles un valor numérico basados en su potencial para solucionar el problema, de tal manera que se puedan identificar los mejores cromosomas para la optimización en cuestión. Esta sección es fundamental porque es quien decide cuáles individuos se toman como buenas soluciones y cuáles serán reemplazados por la siguiente generación.

En este proyecto, la complejidad de la función costo se incrementa de manera considerable, por el hecho de que el valor final debe depender de cualquier combinación de las variables que haya especificado el usuario. Para ejemplificar este problema de optimizaciones dinámicas, a continuación se enlistan tres posibles necesidades de un diseñador de laminados compuestos:

1. Maximizar solamente el módulo de Young E_X.
2. Minimizar el coeficiente de Poisson n, acercar el valor de la matriz A en la posición 2,2 a 45.5 y que el cortante aparente G_{XY} sea menor a 35.
3. Maximizar el módulo de Young E_Y, maximizar el valor de la matriz B en la posición 2,3 y la posición 3,3.

Tomando en cuenta lo anterior, se debe implementar una función generalizada que evalúe correctamente cada individuo de la población sin importar cuántas y cuáles propiedades se deseen optimizar. Dicha función se puede apreciar de manera gráfica en la *Figura 4.5* y se explica a continuación.

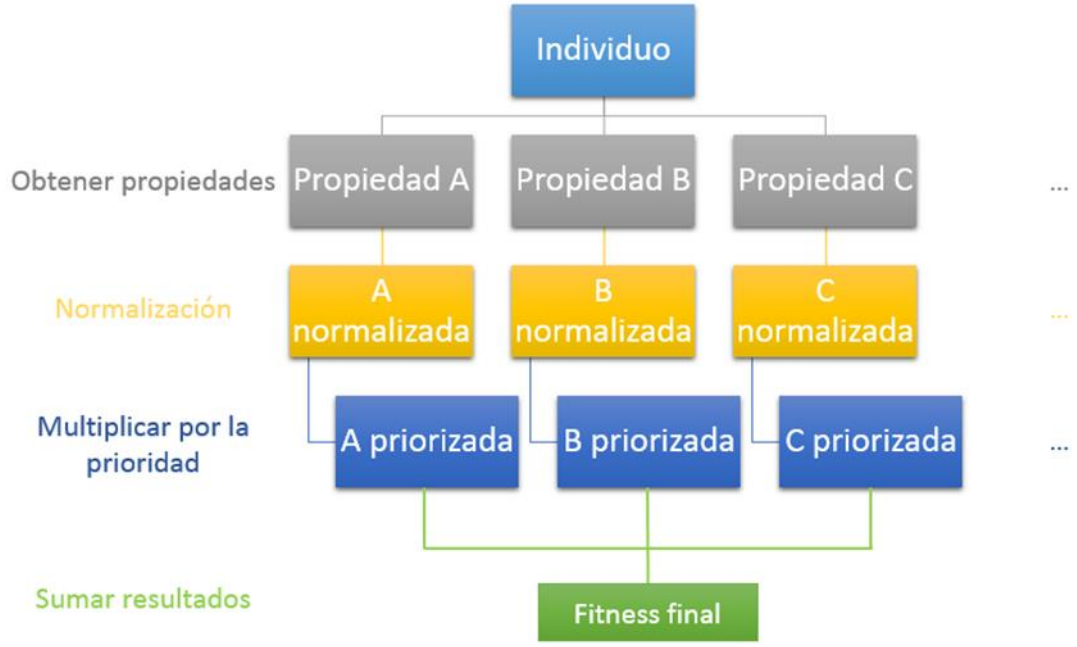

Figura 4.5. Proceso a seguir en la evaluación de un cromosoma.

Para cada individuo que se desee evaluar se deben seguir una serie de pasos:

1. Usar las funciones de laminados compuestos incluidos en MAC LAM para obtener las propiedades que el usuario desea optimizar.
2. Algunas de las propiedades pueden existir en magnitudes diferentes (por ejemplo, el coeficiente de Poisson ν tiene un rango de valores entre 0 y 1, mientras que el módulo de Young E cuenta con valores de varios miles), así que es necesario pasar todos los datos por una fase de normalización para igualar sus magnitudes a un rango de cero como valor mínimo y 1000 como valor máximo. De esta forma, todos los datos adquieren la misma importancia.
3. El usuario es el responsable de definir cuál o cuáles de las variables a optimizar debe tener prioridad sobre las demás dependiendo enteramente de sus necesidades, por lo que los datos son multiplicados por una constante de prioridad especificado por el usuario antes de iniciar la ejecución del algoritmo genético.
4. Debido a que se tiene un resultado para cada una de las propiedades que se desea optimizar, es recomendable aplicar un algoritmo para combinar estas cantidades en una sola, ya que así

es más sencillo realizar comparaciones y así reconocer cuál individuo es mejor que los demás. Para este proyecto se implementaron dos algoritmos para combinar los resultados:

- El primero se dedica solamente a multiplicar cada uno de los valores a optimizar para tener una sola cantidad al final. Esta es la solución más sencilla y resulta correcta cuando solamente se busca obtener la solución que tenga mejor valor, sin importar que se termine optimizando una sola variable en el proceso, dejando las demás sin consideración alguna.
- El segundo algoritmo divide la multiplicación de los resultados entre su media, con lo que se consigue una cantidad que aumente mientras más parecidos son los resultados que se calcularon. Dicha solución se aplica cuando el usuario requiere que se busquen optimizar todas las variables por igual, aunque eso signifique disminuir el valor de alguna de ellas para aumentar otro.

El costo final obtenido por la función anterior se debe emparejar con su cromosoma para poder ordenar la lista de individuos y así tener una idea clara de las mejores soluciones del problema que se tienen hasta ese momento. En la *Tabla 4.1* se puede apreciar una representación simple de una lista ordenada de cromosomas junto con sus costos correspondientes.

Tabla 4.1. Ejemplo de cromosomas evaluados.

Cromosoma	Fitness
11000110101011110110100001101011	103
11101010101010101010101010101110	86
10101011101010101010101011011011	76
10101100110110110111001100100010	48
10101010001010101110101101101111	33
11101011110000011010101011101011	17
00101000101110101010100100011110	15
11010110110011111101010000101001	9
11110101011010000111010110101000	0

4.4 Diseño

La primer interfaz con la que el usuario debe interactuar primero tiene como objetivo permitirle al usuario capturar los materiales que podrá usar el algoritmo genético y definir las propiedades que se desean optimizar, así como las configuraciones generales para el algoritmo (ver *Figura 4.6*).

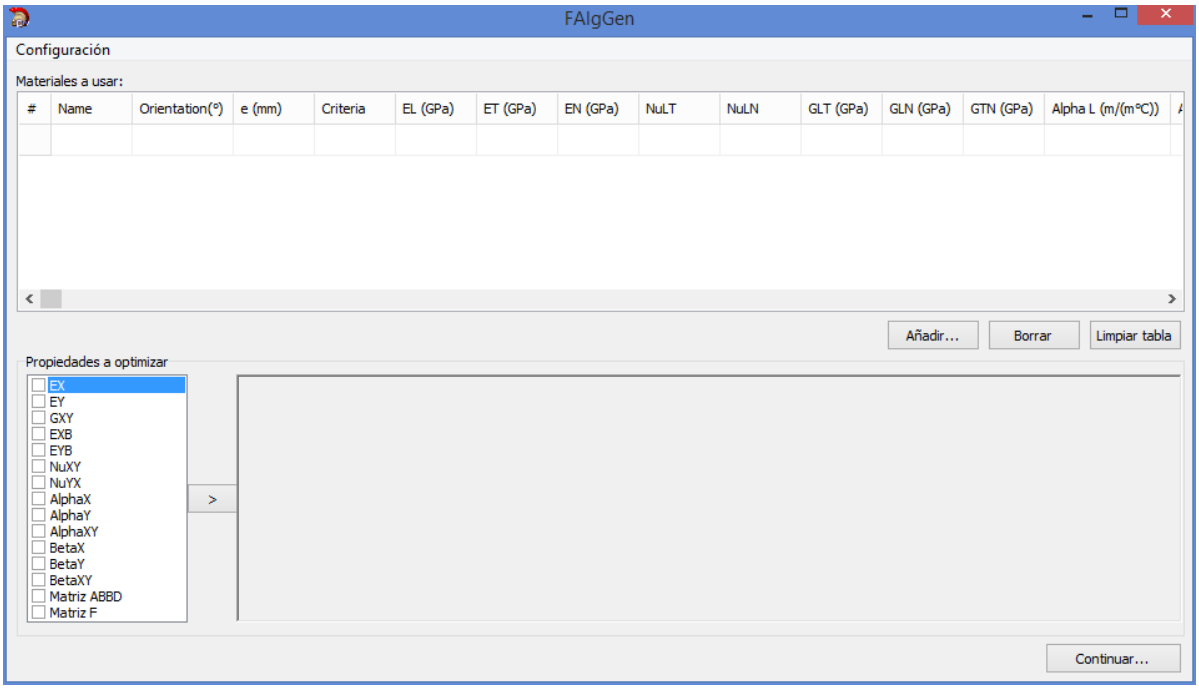

Figura 4.6. Ventana principal del módulo.

La parte superior de la ventana se dedica a manejar la tabla en la que se deben agregar los materiales que el algoritmo podrá utilizar para generar los individuos de la población. Para agregar un material a la tabla el usuario puede hacer click en el botón “Añadir…”, el cual despliega una ventana donde se muestran los materiales existentes en la base de datos de MAC LAM, tal y como se muestra en la *Figura 4.7*.

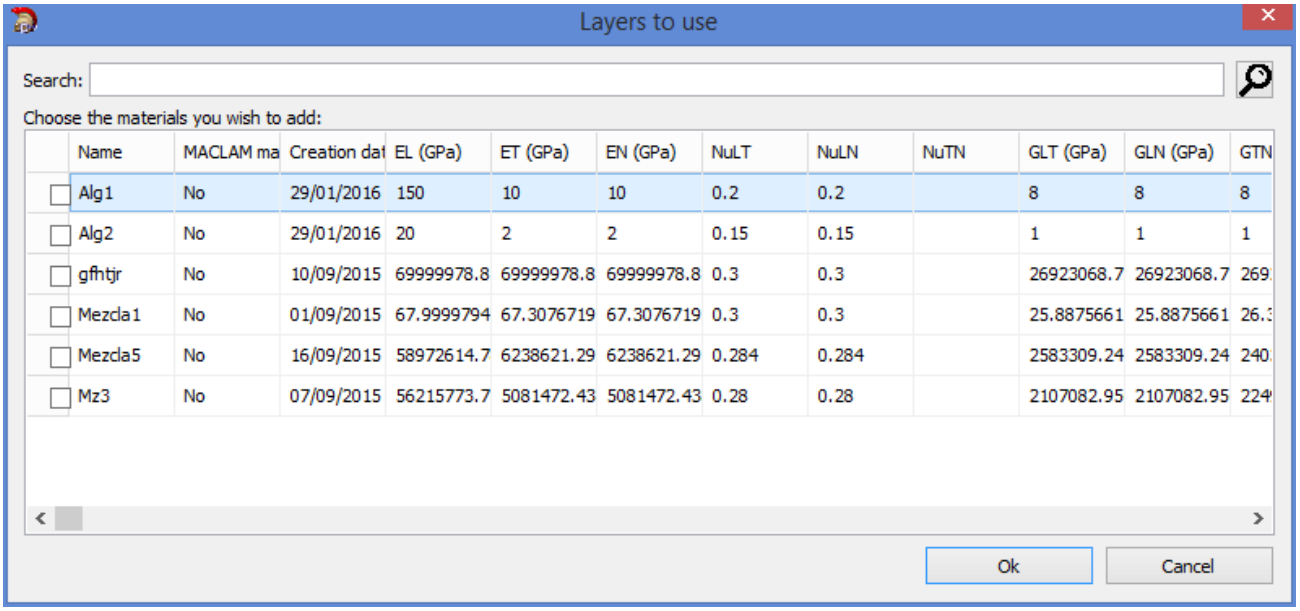

Figura 4.7. Ventana de selección de materiales compuestos.

Una vez el usuario active las casillas de verificación correspondientes a los materiales deseados, debe hacer click en "OK" para terminar su selección, lo cual transfiere la información solicitada a la ventana anterior y le devuelve el control. De la misma forma, es posible borrar materiales de la tabla usando los botones de control ubicados debajo de la misma (ver *Figura 4.8*). El botón "Borrar" elimina el material seleccionado de la tabla, mientras que el botón "Limpiar tabla" elimina todos los registros que se le hayan añadido.

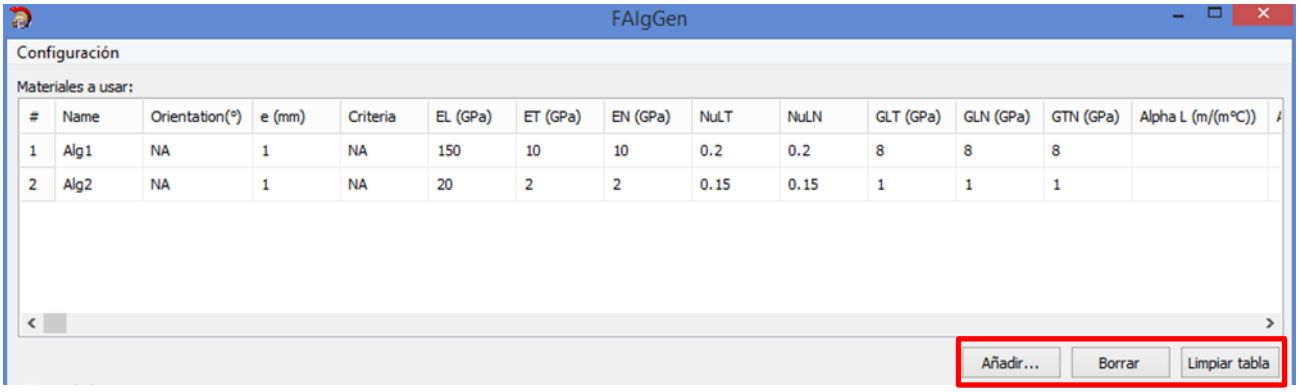

Figura 4.8. Selección de material compuesto para el algoritmo.

La parte inferior de la misma ventana tiene como objetivo administrar las variables que el usuario desea optimizar y proporcionar un ambiente intuitivo para que pueda ser usado por una persona con entendimiento básico y mediano del uso de software (ver *Figura 4.9*). A continuación se explican los aspectos principales de esta sección de la interfaz:

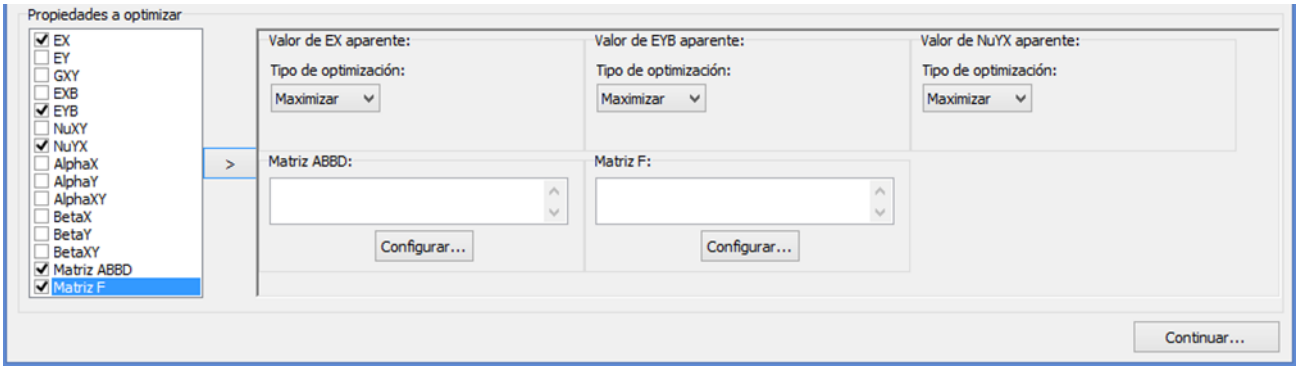

Figura 4.9. Vista de las propiedades a optimizar.

Debido a la cantidad de variables que se pueden optimizar, la idea de mostrar todas las configuraciones posibles resulta poco efectiva, por lo que se decidió presentar las variables disponibles junto con una casilla de verificación en la parte izquierda de la pantalla, lo cual le permite al usuario seleccionar solamente las opciones que desee. A la derecha de esta lista se encuentra el botón “>”, el cual sirve para tomar las configuraciones de cada variable seleccionada y mostrarlas en el panel de la derecha de la pantalla.

Desde ese panel, el usuario tiene la capacidad de elegir qué tipo de optimización desea para cada variable mediante una lista desplegable. Además, si se requiere que el usuario indique algún valor de referencia, la caja de texto correspondiente se muestra junto a su lista desplegable (ver *Figura 4.10*).

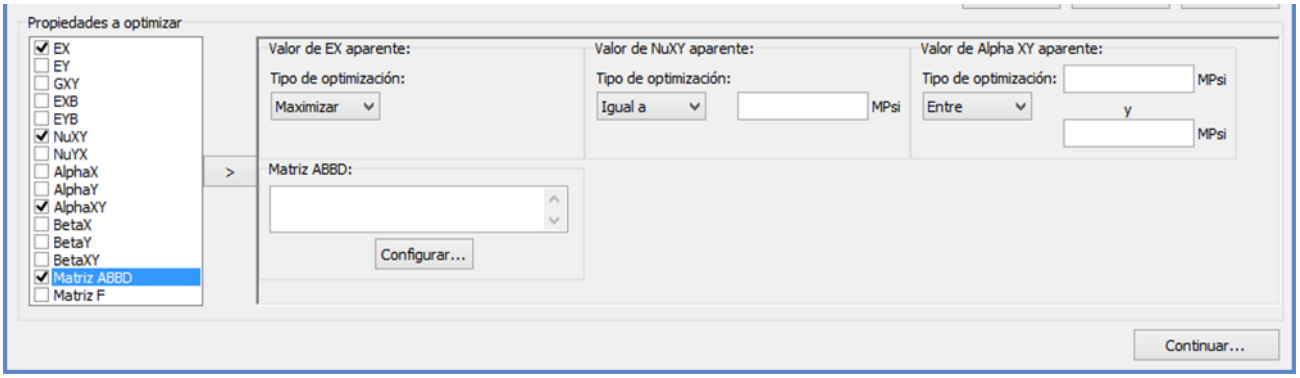

Figura 4.10. Sección para especificar los valores de referencia.

Lo anterior tiene como excepción las opciones para optimizar la matriz ABBD y la matriz F, ya que cada una de estas variables consta de múltiples valores que se pueden optimizar. Si el usuario desea incluir una de estas matrices en el cálculo del algoritmo, debe especificar cuáles y cómo las quiere optimizar. Para ello, debe hacer click en el botón "Configurar…" que se encuentra en su área correspondiente. Esta acción abrirá otra ventana en la que el usuario puede definir los puntos requeridos para su optimización dependiendo de la matriz a configurar. Como ejemplo se muestra en la *Figura 4.11* la ventana correspondiente a la matriz ABBD.

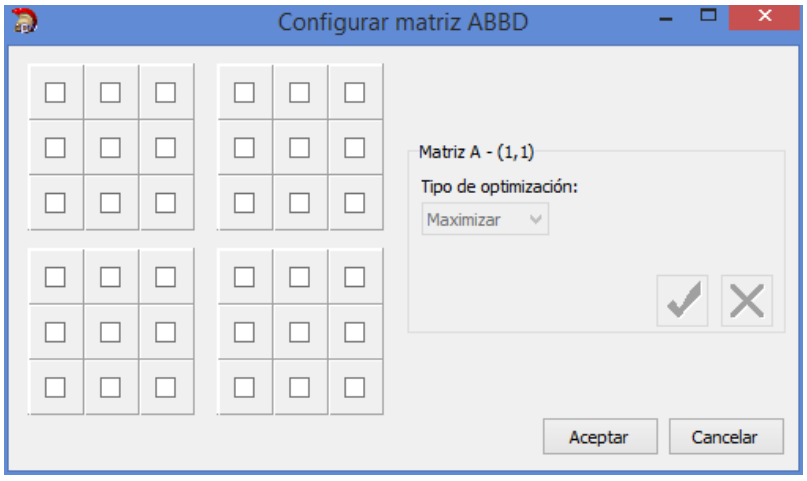

Figura 4.11. Ventana de Matriz ABBD sin configurar.

La variable ABBD se compone de cuatro matrices de 3x3 datos cada una, lo cual da un total de 36 posibles valores a optimizar. Cada uno de estos valores es representado por una casilla de verificación

en la parte izquierda de esta ventana. Si el usuario desea usar alguno de estos elementos, debe activar su casilla correspondiente, habilitando así el grupo de componentes en los que especificará qué tipo de optimización se debe usar con el valor en cuestión. En la *Figura 4.12* se observa un ejemplo de un valor activado.

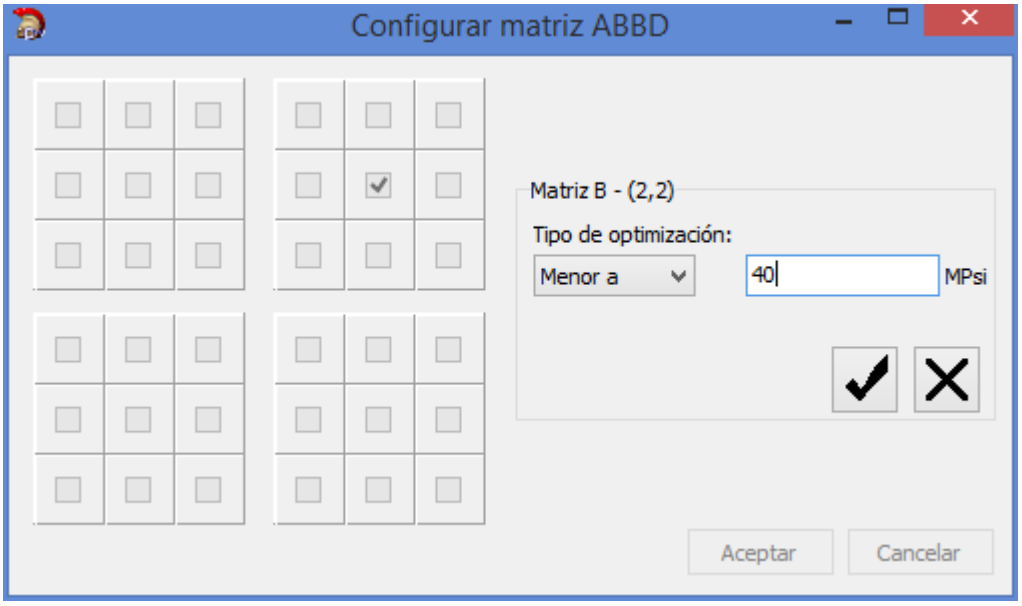

Figura 4.12. Configuración de elemento de la matriz ABBD.

Una vez seleccionado el tipo de optimización, hay que hacer click en el botón de confirmación para guardar los cambios. En cambio, si fue una equivocación el seleccionar este elemento, el usuario puede cancelar la configuración con el botón "X" que está a la derecha del botón de confirmación; esto hace que la casilla de verificación se desactive y el panel de configuración regrese a su estado deshabilitado.

El usuario puede seleccionar tantos valores como necesite y, cuando esté satisfecho con la configuración de la matriz ABBD, terminar con esta ventana usando el botón "Aceptar", el cual guarda toda la selección en un vector temporal para usarlo después. Por otro lado, si no quiere que se guarde dicha información, el botón "Cancelar" cierra la ventana sin almacenar el vector. Al volver a la ventana del algoritmo genético, la información de la matriz ABBD es utilizada para llenar un campo

de texto con un resumen de lo que pidió el usuario para fines de visualización; de esta manera, es fácil tener en cuenta cuáles valores de esta matriz fueron seleccionados para la optimización (ver *Figura 4.13*).

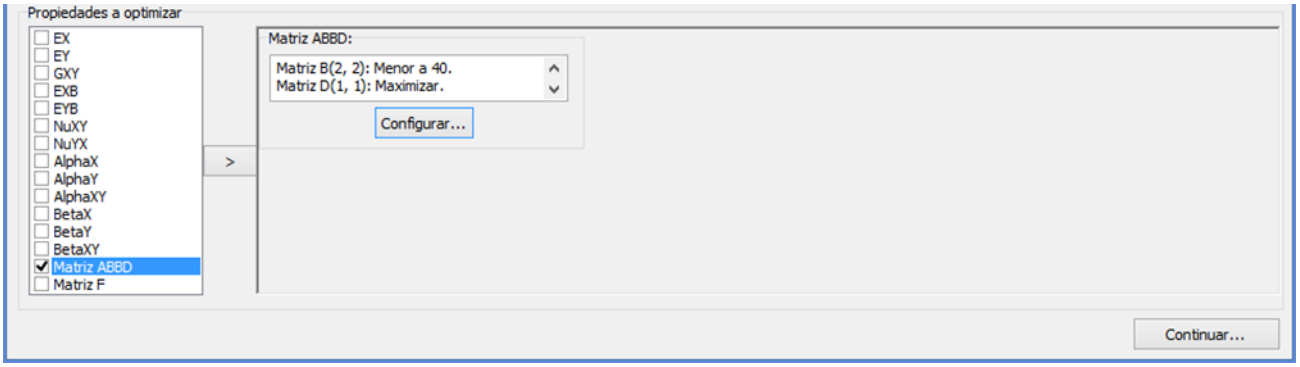

Figura 4.13. Sistema mostrando resumen de la configuración de matriz ABBD.

Después de haber realizado todas las configuraciones necesarias para el algoritmo genético de esta ventana, el usuario debe hacer click en el botón "Continuar…" para avanzar en el proceso. El último paso para la configuración es la asignación de las prioridades para cada una de las variables que se hayan seleccionado (como se muestra en la *Figura 4.14*), esto con la finalidad de establecer cuáles datos son más importantes que los demás.

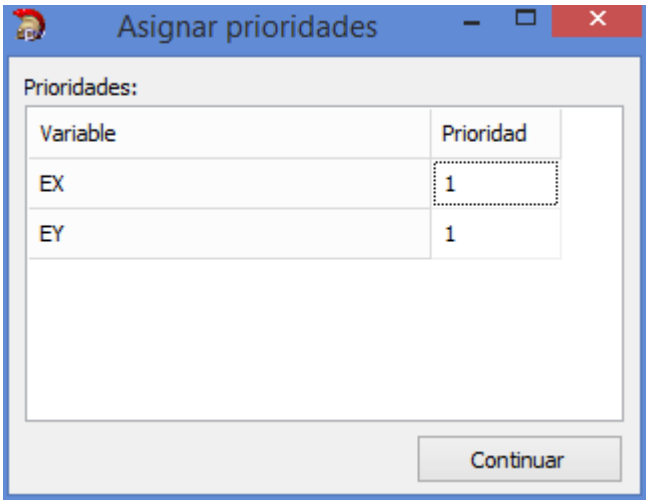

Figura 4.14. Ventana donde se establece la prioridad de cada variable a optimizar.

Al iniciar la ventana, todas las variables son asignadas con un 1 de prioridad. El usuario puede aumentar o disminuir esa cantidad (mayor valor significa mayor prioridad) para uno o varios de los valores y así adecuarlos a sus necesidades. Al terminar, lo único que se debe hacer es hacer click en el botón "Continuar", esto mostrará una última ventana de diálogo para confirmar la decisión de ejecutar el algoritmo genético. Una vez el usuario acepte, solo queda esperar a que el algoritmo termine.

La *Figura 4.15* muestra un ejemplo de la ventana de resultados del algoritmo genético, la cual consta de una pestaña para mostrar la información de la población inicial y otra para la población final (resultados). Ambas pestañas cuentan con los siguientes componentes:

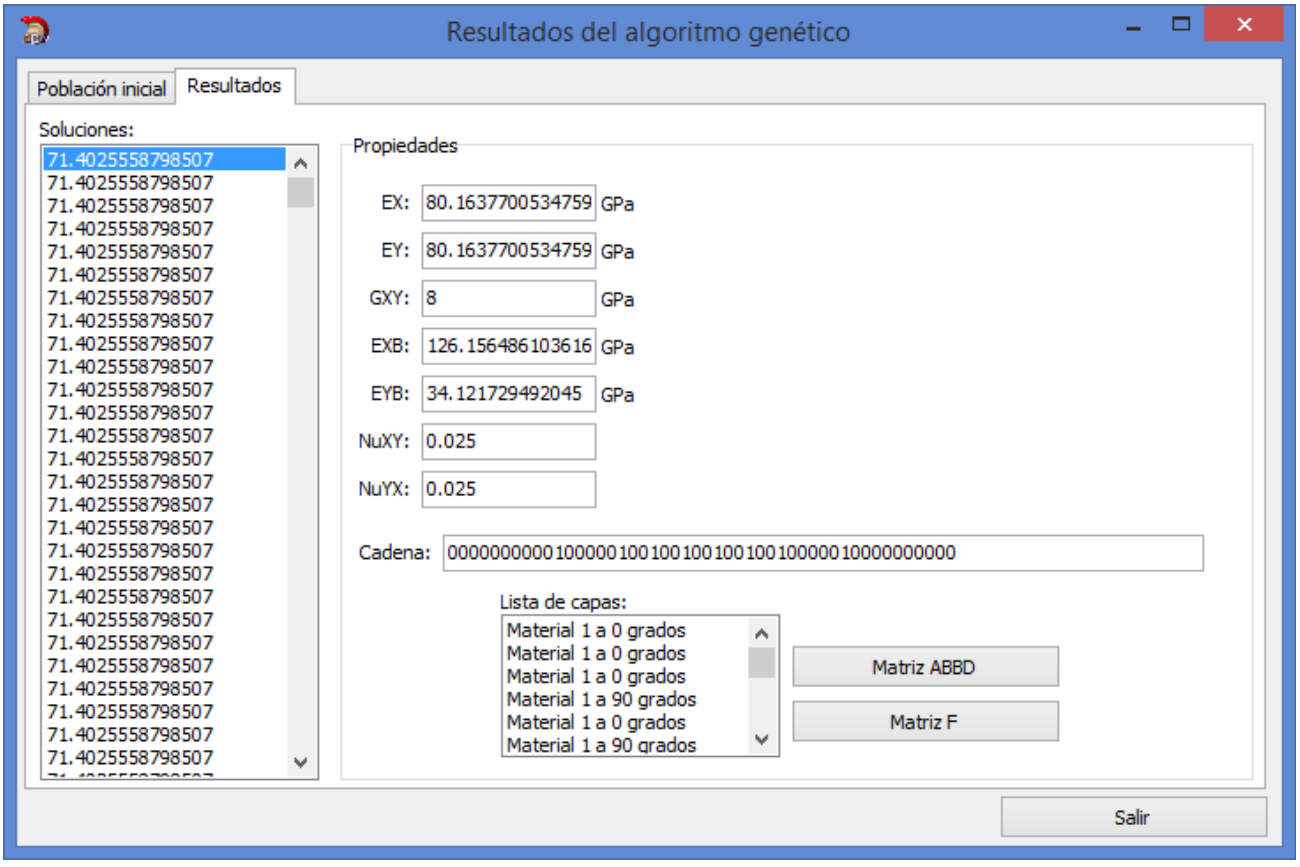

Figura 4.15. Ventana de resultados del algoritmo genético.

- A la izquierda se despliega una lista con las soluciones obtenidas, ordenadas de mejor a peor en base a su costo. De la lista se puede seleccionar una solución para resaltarla.

- A la derecha se encuentra un panel en el que se muestran las propiedades de la solución seleccionada, incluyendo la cadena binaria que la representa y la lista de las capas y su orientación que conforman el apilado.
- Botones para acceder a la matriz ABBD y la matriz F respectivamente.

4.5 Codificación

La programación del proyecto fue dividido en dos partes para un mejor manejo de las instancias y los bloques de memoria. Por un lado se encuentra la programación del algoritmo genético como tal junto a los métodos de inicialización para que dichos procedimientos puedan ser accedidos desde cualquier parte del sistema. Por otra parte, la codificación de los eventos y el manejo de datos en la interfaz gráfica se realizaron pensando en la forma más intuitiva de hacer las cosas para que el usuario pudiera usar el módulo sin tener que ser experto tanto en materiales compuestos como en algoritmos genéticos.

Ambas partes de la codificación son importantes porque brindan la funcionalidad y la accesibilidad del módulo, por lo que vale la pena describirlas.

4.5.1 Codificación de la interfaz gráfica

La principal característica que representa una dificultad al momento de programar los eventos de la interfaz gráfica radica en la cantidad de componentes que pueden ser mostrados y ocultados para su funcionamiento correcto. Debido a que existen un total de 53 propiedades que pueden ser optimizadas por el algoritmo genético, las funciones usadas comúnmente para la visualización y captura de datos resultan poco efectivas, razón por la cual esta pantalla debe ser programada con un enfoque diferente.

Para resolver el problema de la cantidad dinámica de propiedades a optimizar se decidió que solamente se deben mostrar los datos que el usuario debe proporcionar para que el programa funcione mediante el uso de contenedores independientes llamados *TGroupBox* para cada una de ellas, omitiendo todas las propiedades no deseadas.

Se desarrollaron un total de 15 contenedores para lograr este objetivo: uno para cada propiedad separada y uno para cada matriz de propiedades necesarias (ABBD y F). En la *Figura 4.16* se puede apreciar un ejemplo del contenedor del módulo de Young Ex aparente, junto con la lista de los contenedores elaborados en el árbol de componentes del IDE.

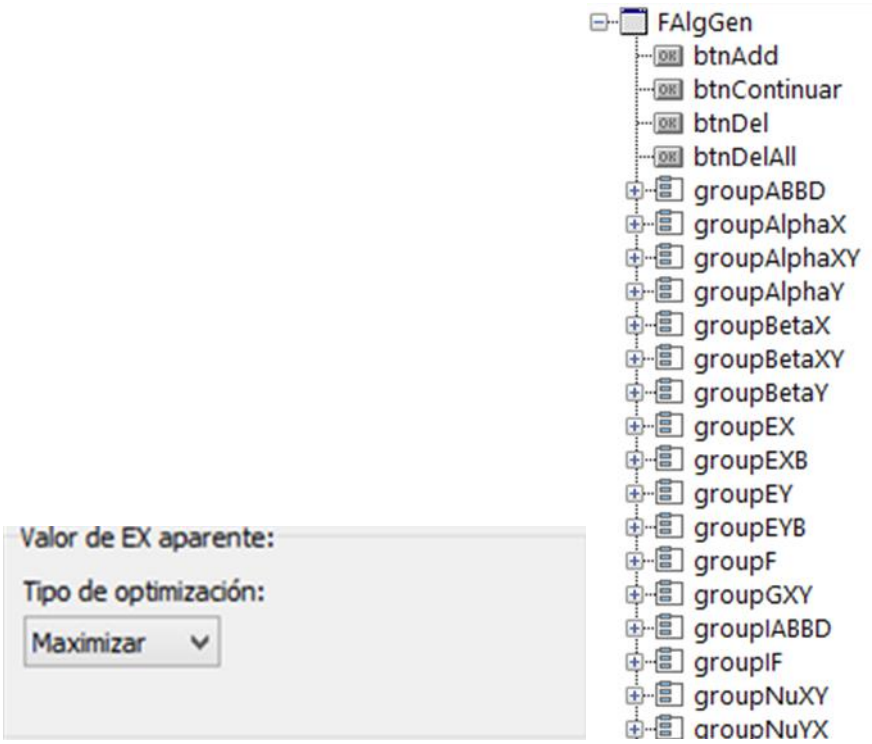

Figura 4.16. Árbol de componentes para los grupos de variables.

De esta manera, los *TGroupBox* pueden habilitarse y deshabilitarse de forma independiente mediante las acciones del usuario. Sin embargo, esto presenta una nueva dificultad al momento de la codificación, ya que generar los eventos necesarios para cada uno de los contenedores representa una cantidad innecesariamente larga de código. Si se desea manejar esta situación de forma óptima, es necesario programar los eventos y las interacciones entre componentes de una forma general que aplique a los más casos posibles.

La primera parte del código a desarrollar es crear una lista que guarde las referencias de los contenedores de las propiedades con la finalidad de obtener la capacidad de iterar dicha lista para realizar tareas y verificaciones. En la *Figura 4.17* se muestra la declaración e inicialización del arreglo de los contenedores.

```
TGroupBox *lstGroups[15];
lstGroups[0] = groupEX;
lstGroups[1] = groupEY;
lstGroups[2] = groupGXY;
lstGroups[3] = groupEXB;
lstGroups[4] = groupEYB;
lstGroups[5] = groupNuXY;
lstGroups[6] = groupNuYX;
lstGroups[7] = groupAlphaX;
lstGroups[8] = groupAlphaY;
lstGroups[9] = groupAlphaXY;
lstGroups[10] = groupBetaX;
lstGroups[11] = groupBetaY;
lstGroups[12] = groupBetaXY;
lstGroups[13] = groupABBD;
lstGroups[14] = groupF;
```

Figura 4.17. Determinación de arreglos de *TGroupBox*.

De la misma forma, se generaron arreglos y matrices para varios de los componentes disponibles en la interfaz gráfica, tales como las listas desplegables y los campos de texto, todo esto con la finalidad de conseguir un conjunto de objetos iterables para desarrollar algoritmos más cortos y eficientes.

Con este tipo de arreglos de componentes gráficos fue posible el desarrollo de eventos comunes para todos los miembros de un arreglo con características en común, definiendo el comportamiento de la mayoría de los procesos de la interfaz gráfica sin necesidad de crear un evento por cada uno de ellos. Un ejemplo de ello es el manejo de las listas desplegables de cada una de las propiedades que se pueden optimizar, en donde se generó un solo evento en el que todas las listas pueden acceder cuando fue seleccionada alguna de sus opciones (ver *Figura 4.18*).

```
void __fastcall TFAlgGen::comboOnChange(TObject *Sender)
{
      //Verificar cual TComboBox fue usado
      TComboBox *gr = (TComboBox *) Sender;
      if (gr == comboEX) {
            enableText(comboEX, txtEXDesde, txtEXIgual, txtEXHasta, lblEXy);
      } else if (gr == comboEY) {
            enableText(comboEY, txtEYDesde, txtEYIgual, txtEYHasta, lblEYy);
      } else if (gr == comboGXY) {
            enableText(comboGXY, txtGXYDesde, txtGXYIgual, txtGXYHasta,
            lblGXYy);
      } else if (gr == comboEXB) {
            enableText(comboEXB, txtEXBDesde, txtEXBIgual, txtEXBHasta,
            lblEXBy);
      } else if (gr == comboEYB) {
            enableText(comboEYB, txtEYBDesde, txtEYBIgual, txtEYBHasta,
            lblEYBy);
      } else if (gr == comboNuXY) {
            enableText(comboNuXY, txtNuXYDesde, txtNuXYIgual, txtNuXYHasta,
            lblNuXYy);
      } else if (gr == comboNuYX) {
            enableText(comboNuYX, txtNuYXDesde, txtNuYXIgual, txtNuYXHasta,
            lblNuYXy);
      } else if (gr == comboAlphaX) {
            enableText(comboAlphaX, txtAlphaXDesde, txtAlphaXIgual,
            txtAlphaXHasta, lblAlphaXy);
      } else if (gr == comboAlphaY) {
            enableText(comboAlphaY, txtAlphaYDesde, txtAlphaYIgual,
            txtAlphaYHasta, lblAlphaYy);
      } else if (gr == comboAlphaXY) {
            enableText(comboAlphaXY, txtAlphaXYDesde, txtAlphaXYIgual,
            txtAlphaXYHasta, lblAlphaXYy);
      } else if (gr == comboBetaX) {
            enableText(comboBetaX, txtBetaXDesde, txtBetaXIgual,
            txtBetaXHasta, lblBetaXy);
      } else if (gr == comboBetaY) {
            enableText(comboBetaY, txtBetaYDesde, txtBetaYIgual,
            txtBetaYHasta, lblBetaYy);
      } else if (gr == comboBetaXY) {
            enableText(comboBetaXY, txtBetaXYDesde, txtBetaXYIgual,
            txtBetaXYHasta, lblBetaXYy);
      }
}
```

Figura 4.18. Comparación para encontrar el componente adecuado.

El evento anterior toma la referencia de la lista desplegable que lo activó para luego compararlo con las diferentes posibilidades. En el momento que encuentra la lista que debe ser cambiada, llama a un método cuyo objetivo es habilitar o deshabilitar los campos de textos donde se definen los parámetros de la optimización. El código de este método se puede apreciar en la *Figura 4.19*.

```
void TFAlgGen::enableText(TComboBox *combo, TLabeledEdit *desde, TLabeledEdit
*igual, TLabeledEdit *hasta, TLabel *y)
{
	desde->Visible = false;
	desde->Text = "";
	igual->Visible = false;
	igual->Text = "";
	hasta->Visible = false;
	hasta->Text = "";
	y->Visible = false;

	switch (combo->ItemIndex) {
		case 2:
		case 3:
		case 4:
			igual->Visible = true;
		break;
		case 5:
			desde->Visible = true;
			hasta->Visible = true;
			y->Visible = true;
	default:
		;
	}
}
```

Figura 4.19. Casos de visibilidad de componentes.

Existen diferentes casos que pueden suceder al momento de determinar cuáles componentes de la propiedad deben ser mostrados:

- Si se requiere maximizar o minimizar, no es necesario mostrar campo de texto alguno, ya que no se precisa información adicional.
- Para acercar a un valor o verificar si el criterio es mayor o menor a un valor, es necesario habilitar el campo de texto central para especificar la cantidad de referencia.
- En caso de requerir que la propiedad se encuentre entre dos valores, se deben activar los dos campos de texto de las orillas para proporcionar los límites superior e inferior de la búsqueda.

Estos casos son aplicados a los componentes que fueron proporcionados por el evento descrito anteriormente, formando un procedimiento totalmente dinámico e independiente de cuál propiedad mecánica se debe optimizar.

Otro ejemplo de la importancia de la codificación dinámica en los grupos de componentes de cada variable se puede apreciar al momento de verificar cuáles de las variables seleccionó el usuario para optimizar con el objetivo de agregar los grupos correspondientes al contenedor de la interfaz gráfica y el usuario pueda interactuar con ellos (ver *Figura 4.20*).

```
void __fastcall TFAlgGen::btnAgregarRestriccionesClick(TObject *Sender)
{
	for (int i = 0; i < checkLstVariables->Count - 4; i++) {
		lstGroups[i]->Parent = NULL;
	}
	int iCont = 0;
	for (int i = 0; i < 13; i++) {
		if (checkLstVariables->Checked[i]) {
			iCont++;
			lstGroups[i]->Parent = pnlVariables;
			comboOnChange(lstCombo[i]);
		}
	}

	for (int i = 13; i < 15; i++) {
		if (checkLstVariables->Checked[i]) {
			iCont++;
			lstGroups[i]->Parent = pnlVariables;
		}
	}

	int res = iCont / 3 + 1;
	pnlVariables->Height = res * 90;
}
```

Figura 4.20. Método para agregar los grupos de componentes al contenedor apropiado.

En lugar de verificar los grupos de componentes uno por uno, el sistema itera los arreglos de dichos grupos para compararlos de forma dinámica y utilizando menos código.

Finalmente, la ejecución del algoritmo genético inicia en el momento en que se hace la llamada al método que se muestra en la *Figura 4.21*.

```
FRecursos->algGen->empezar(nPOP, xRATE, xCONV, xMUTA, nCAPAS,
TAlgoritmoGenetico::PROGRAMACION_META, FAlgGen->lstMaterial);
```

Figura 4.21. Llamada al algoritmo genético.

4.5.2 Codificación del algoritmo genético

El primer paso en la ejecución del algoritmo genético es la llamada al método empezar, el cual contiene la inicialización de las variables de configuración y describe el esqueleto básico del algoritmo como un conjunto. Para poder ejecutar este método se deben proporcionar una serie de datos que sirven para determinar el comportamiento que tendrá la ejecución que está por comenzar; dichos datos se indican como argumentos en el método (ver *Figura 4.22*) y se enlistan a continuación:

1. `int nPOP`.- Representa el número de individuos que tendrá la población de la presente ejecución. En este argumento se puede establecer un número determinado por el usuario o indicar la cantidad calculada por el sistema.
2. `float xRATE`.- Proporción que tendrán los individuos que sobreviven a la siguiente generación y los que son reemplazados. Es un número decimal que se encuentra entre cero y uno que define el porcentaje de "Buenos individuos" y "Malos individuos". Por ejemplo, un valor de *xRATE* = 0.5 indica un 50% de individuos que sobreviven a la siguiente generación contra 50% de individuos que son reemplazados por los mejores individuos de la nueva generación.
3. `float xCONV.-` Criterio de convergencia a cumplir. Es un número decimal mayor a cero y menor a uno que indica el umbral que debe cruzar el criterio de convergencia del algoritmo para determinar que se encontró la solución óptima.
4. `float xMUTA`.- Probabilidad de mutación. Indica la cantidad de bits que deben ser procesados para que un bit cambie de valor por mutación. Por ejemplo, un valor de *xMUTA* = 0.01 indica que uno de cada 100 bits será mutado.
5. `int emparejar.-` Método de emparejamiento.- Se usa para determinar la forma en la que se acumulan los resultados durante el cálculo de la función costo. Puede tener dos valores: *SUMATORIA* = Método que permite balancear la prioridad de los costos en problemas multiobjetivo. *PROGRAMACION_META* = Método que desprecia el balance y simplemente busca el costo óptimo.
6. `TList *materiales`.- Lista de los materiales. En este objeto se indican todos los materiales compuestos que pueden formar parte de las soluciones proporcionadas por el algoritmo genético.

```
bool TAlgoritmoGenetico::empezar(int nPOP, float xRATE, float xCONV,
float xMUTA, int emparejar, TList *materiales)
{
	this->nPOP = nPOP;
	this->xRATE = xRATE;
	this->xCONV = xCONV;
	this->xMUTA = xMUTA;
	this->metodoEmparejar = emparejar;
	nMUTA = 1 / xMUTA;
	iterarMUTA = 0;
	matDisp = materiales;
	float lastCONV = 0, cONV = 100000;
	lastPROM = lastMAX = lastDesv = 0;
	actMAX = actPROM = actDesv = 0;
	float fAux;

	nVariables = 0;
	for (int i = 0; i < 53; i++) {
		if (FRecursos->restricciones[i][5] > 0)
			nVariables++;
	}
	nuevo = false;

	nGOOD = nPOP * xRATE;
	genPOP();
	generacion = 0;

	do {
		generacion++;
		lastPROM = actPROM;
		lastMAX = actMAX;
		lastDesv = actDesv;
		selParejas();
		cruce();
		actPROM = verificarConvergencia();
		if (actPROM <= 0.0)
			return false;
		if (actMAX <= 0.0)
			return false;
		convPROM = (float) FRecursos->abs(lastPROM - actPROM) /
actPROM;
		convMAX = (float) FRecursos->abs(lastMAX - actMAX) / actMAX;
	} while (convPROM > xCONV || convMAX > xCONV);
	return true;
}
```

Figura 4.22. Esqueleto del algoritmo genético en código.

En este segmento de código se localizan las llamadas a todos los pasos que conforman la estructura de un algoritmo genético, de los cuales la mayoría se encuentran dentro de un ciclo *do while* que tiene como condición de salida que se cumpla el criterio de convergencia establecido por el usuario. El archivo completo para el algoritmo genético se puede observar en el *ANEXO C. CÓDIGO DEL*

ALGORITMO GENÉTICO para futuras referencias; en esta sección del documento se van a explicar las partes más importantes de la codificación del algoritmo.

4.5.3 Generar población inicial (*genPOP*)

Para almacenar los individuos de la población inicial son necesarias dos matrices y un vector, tal y como se muestra a continuación (ver *Figura 4.23*):

```
fitInicial = new double[nPOP];
pobInicial = new bool*[nPOP];
resInicial = new double*[nPOP];
for (int i = 0; i < nPOP; i++) {
      sAux = "";
      pobInicial[i] = new bool[nMaxBits];
      resInicial[i] = new double[nVariables];
      for (int j = 0; j < nMaxBits; j++) {
            pobInicial[i][j] = (rand() % 2) == 1;
            sAux += pobInicial[i][j] ? "1" : "0";
      }
      fitInicial[i] = funcionCosto(pobInicial, i, resInicial);
}
```

Figura 4.23. Generación aleatoria de la población inicial.

donde *pobInicial* almacena el apilado codificado en elementos booleanos para representar sus valores de la forma más eficiente, *resInicial* guarda las propiedades calculadas que se desean optimizar y *fitInicial* guarda el resultado de la función costo para cada uno de los individuos. Los elementos de dichos vectores son inicializados con elementos generados aleatoriamente por el sistema, dando lugar a una población de posibles soluciones con la cual se puede comenzar a trabajar.

Debido a que en ocasiones es necesario presentar información de la población inicial para comprobar la evolución de un algoritmo genético, *fitInicial*, *pobInicial* y *resInicial* no son los valores que se utilizan a lo largo de todas las generaciones. En su lugar, una vez evaluados y ordenados los individuos de la población inicial, se obtiene una copia de los valores en otros vectores y matrices para realizar las operaciones necesarias a partir de ese momento; estos campos son *fit*, *res* y *pob*, como se muestra en las siguientes líneas de código (ver *Figura 4.24*).

```
Pob = new bool*[nPOP];
fit = new double[nPOP];
res = new double*[nPOP];
for (int i = 0; i < nPOP; i++) {
      pob[i] = new bool[nMaxBits];
      res[i] = new double[nVariables];
      for (int j = 0; j < nMaxBits; j++) {
            pob[i][j] = pobInicial[i][j];
      }
      fit[i] = fitInicial[i];
      for (int j = 0; j < nVariables; j++) {
            res[i][j] = resInicial[i][j];
      }
}
```

Figura 4.24. Inicialización de la población real en base a la población inicial.

Otra parte importante de la generación de la población inicial es el ordenamiento; cada vez que se realizan cambios en los individuos de una población, es necesario evaluarlos y ordenarlos de mejor a peor para mantener la estructura y aumentar la probabilidad de conseguir mejores individuos con cada generación que pase.

4.5.4 Ordenamiento de individuos

La naturaleza aleatoria de los datos que se asignan a la población es la razón por la que se implementó un algoritmo de ordenamiento HeapSort, el cual básicamente consiste en convertir el arreglo que se busca ordenar en una estructura de montículos (heap) binaria, la cual tiene la propiedad de permitir una extracción y eliminación eficientes del elemento con mayor valor.

De forma iterativa se remueve el elemento mayor del montículo, con lo que se construye una lista ordenada desde atrás hacia adelante.

4.5.5 Función costo

La primera sección de la función costo implica el obtener las propiedades del individuo en cuestión, esto se logra accediendo a las ecuaciones generadas por el Dr. Alberto Díaz las cuales se encuentran en otro archivo dentro del proyecto, tal como se muestra en la *Figura 4.25*.

```
Asigna_memoria(nCapas,sime);
for(int x = 0; x < nCapas; x++)
{
	mezIterar = (Tmezcla *) matDisp->Items[capas[x]];
	captura_datos(mezIterar->getELMezclas(), mezIterar->getETMezclas(),
	mezIterar->getGLNMezclas(),mezIterar->getGLTMezclas(), mezIterar-
	>getGTNParticiones(), mezIterar->getNuLTMezclas(), mezIterar-
	>getEspesor(), ang[x], x, met);
}
free(capas);
capas = NULL;
free(ang);
ang = NULL;
float *apilado = principal_rigideces(false);
```

Figura 4.25. Obtención de propiedades de un apilado.

Para la etapa de normalización se determina qué tipo de optimización se necesita para cada variable (maximizar, minimizar, acercar a un valor, etc.) y se realizan operaciones simples de tal manera que los resultados tengan un rango de valores de cero a 1000 para que puedan ser comparables entre ellos y no exista una propiedad que valga menos que las demás (ver *Figura 4.26*).

```
Resultado[index][contVariables++] = apilado[i];
s += FloatToStr(apilado[i]) + ",";
caso = Frecursos->restricciones[i][0];
vResultados[i] = apilado[i] * 1000 / Frecursos->restricciones[i][2];
if (caso >= 2)
	vx1 = Frecursos->restricciones[i][3] * 1000 / Frecursos-
	>restricciones[i][2];
switch (caso) {
	case 1:
		vResultados[i] = 1000 - vResultados[i];
	break;
	case 2:
		vResultados[i] = 1000 - Frecursos->abs(vx1 - vResultados[i]);
	break;
	case 3:
		if (vResultados[i] < vx1)
			vResultados[i] = 1000;
		else
			vResultados[i] = 1000 - Frecursos->abs(vx1 - vResultados[i]);
	break;
	case 4:
		if (vResultados[i] > vx1)
			vResultados[i] = 1000;
		else
			vResultados[i] = 1000 - Frecursos->abs(vx1 - vResultados[i]);
	break;
	case 5:
```

```
            vx2 = Frecursos->restricciones[i][4] * 1000 / Frecursos-
            >restricciones[i][2];
            if (vResultados[i] > vx1 && vResultados[i] < vx2)
                  vResultados[i] = 1000;
            else if (vResultados[i] < vx1)
                  vResultados[i] = 1000 - Frecursos->abs(vResultados[i] - vx1);
            else
                  vResultados[i] = 1000 - Frecursos->abs(vResultados[i] - vx2);
      break;
      default:
      ;
}
```

Figura 4.26. Normalización de las propiedades.

Finalmente, se realiza una llamada a la función que acumula los resultados obtenidos según la selección que haya hecho el usuario antes de comenzar con el algoritmo, la cual regresa un solo valor y es almacenado para ser comparado al momento de ordenar (ver *Figura 4.27*).

```
Switch (metodoEmparejar) {
      case SUMATORIA:
            resultado = 1;
            for (int i = 0; i < 53; i++) {
                  if (Frecursos->restricciones[i][5] > 0) {
                        cont++;
                        media += res[i];
                        resultado *= res[i];
                  }
            }
            media /= cont;
            resultado /= media;
      break;
      case PROGRAMACION_META:
            resultado = 1;
            for (int i = 0; i < 53; i++) {
                  if (Frecursos->restricciones[i][5] > 0) {
                        if (resultado > 2)
                              resultado /= 1000.0;
                        resultado *= res[i];

                  }
            }
      break;
}
```

Figura 4.27. Acumulación de los valores.

4.5.6 Selección de parejas

El algoritmo que fue implementado para la selección de las parejas en el algoritmo genético corresponde al de la selección por ranking, el cual indica que a cada individuo se le asigna una

probabilidad de ser escogido, donde los mejores individuos tienen una mayor probabilidad de cruzar su material genético, tal y como sucede en la teoría de la supervivencia del más apto.

En esta sección del algoritmo se asignan las propiedades a los individuos y luego se llena un vector con las parejas que deberán cruzar su material genético para generar una nueva generación de posibles soluciones al problema, tal y como se muestra en la *Figura 4.28*.

```
For (int i = 0; i < max; i++) {
      intentos = 0;
      parejas[i] = new int[2];
      for (int j = 0; j < 2; j++) {
            dAux = ((double) rand() / (RAND_MAX));
            parejas[i][j] = -1;
            for (int k = 0; k < nGOOD; k++) {
                  if (ranking[k] > dAux) {
                        parejas[i][j] = k;
                        k = nPOP;
                  }
            }
            if (parejas[i][j] == -1) {
                  parejas[i][j] = nPOP - 1;
            }
            if (j == 1) {
                  hamming = 0;
                  for (int k = 0; k < nMaxBits; k++) {
                        if (pob[parejas[i][0]][k] != pob[parejas[i][1]][k])
                        hamming++;
                  }
                  if (hamming <= (nMaxBits / 4)) {
                        if (++intentos <= 4) {
                              j++;
                        }
                  }
            }
      }
}
```

Figura 4.28. Algoritmo de selección de parejas.

4.5.7 Cruce

Para mantener la estructura de la población normal hasta que sea necesario combinarla con los mejores individuos de la nueva generación, es necesario generar vectores para almacenar la estructura de las nuevas soluciones y así contar con una base sólida para maniobrar sin el peligro de modificar valores

que comprometan el funcionamiento del algoritmo genético. En la *Figura 4.29* se puede apreciar la inicialización de estos vectores.

```
nuevaGen = new bool*[nPOP];
fitNuevaGen = new double[nPOP];
resNuevaGen = new double*[nPOP];
```

Figura 4.29. Inicialización de vectores para almacenar la nueva generación.

Para definir el algoritmo a utilizar en el cruce, es necesario tomar en cuenta que el código genético que es creado por el algoritmo puede llegar a tener un tamaño considerable dependiendo del número de capas que se necesita obtener como resultado, por lo que la mayoría de las técnicas de cruce son poco efectivas para fomentar la aleatoriedad en la creación de nuevas generaciones de la población. Esto llevó a la conclusión que es necesario utilizar un algoritmo de los ya existentes, pero modificado con el fin de satisfacer las necesidades de este proyecto.

Al final se implementó una técnica de cruce simple con punto intermedio aleatorio por cada una de las capas para un mejor desempeño del algoritmo. Este concepto implica que por cada capa de la pareja a cruzar, se genera un punto de cruce usando un número aleatorio, con el cual se determina que todos los bits anteriores a ese punto se van a quedar igual, mientras que los que se encuentren después serán intercambiados para generar dos nuevos apilados: uno con la primera parte del padre uno y segunda parte del padre dos, y uno con la primera parte del padre dos y la segunda parte del padre uno.

Para ejemplificar la técnica de cruce, en la *Figura 4.30* se puede apreciar una pareja de cromosomas elegida para realizar este procedimiento. Como se indicó en el párrafo anterior, el primer paso es generar un punto de cruce aleatorio por cada una de las capas de la pareja; en el ejemplo de la figura estos puntos se representan por las líneas azul y verde.

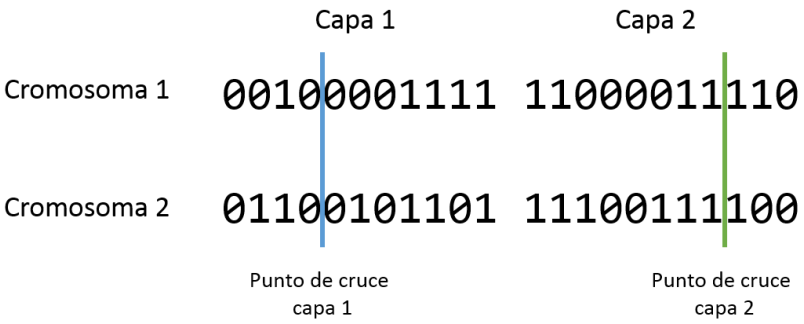

Figura 4.30. Indicación de punto de cruce.

Se generan dos nuevos cromosomas utilizados para la nueva generación, para lo cual se toman los bits a la izquierda del punto de cruce y se ponen en la nueva generación sin modificaciones. Después, se utilizan los bits a la derecha de dicho punto y cambian de lugar, es decir, los bits del cromosoma 2 se pasan al cromosoma 1 y viceversa (ver *Figura 4.31*).

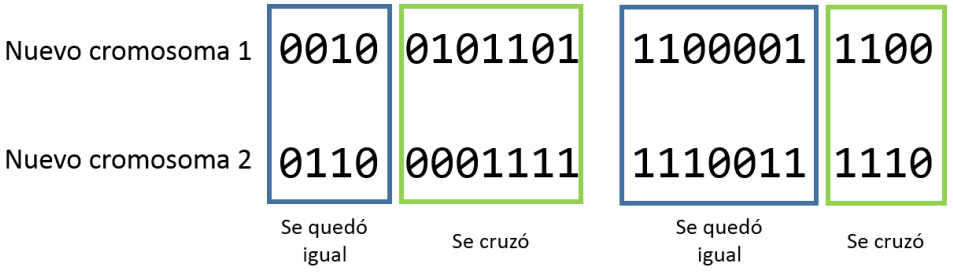

Figura 4.31. Resultado del ejemplo de cruce.

Para implementar este algoritmo en código, se tienen que iterar todas las capas de la pareja, calculando la posición de inicio y fin de cada una de ellas. Después, se define el punto de cruce mediante un número aleatorio y los índices obtenidos anteriormente. Finalmente, se procede a generar los nuevos cromosomas utilizando partes de ambos miembros de la pareja. En la *Figura 4.32* se muestra la parte del código donde se determina el punto de cruce y se cambia el material genético de la pareja.

```
for (int k = 0; k < nMaxCapas; k++) {
      indexIniciar = k * (nBitsCapa + nBITSANGULO);
      indexFinal = (k + 1) * (nBitsCapa + nBITSANGULO);
      dAux = ((double) rand() / (RAND_MAX));
      indexCambiar = FRecursos->redondear(dAux * (indexFinal - indexIniciar) +
      indexIniciar);
```

```
        for (int j = indexIniciar; j < indexCambiar; j++) {
              nuevaGen[indexNuevaGen][j] = pob[parejas[i][0]][j];
              nuevaGen[indexNuevaGen+1][j] = pob[parejas[i][1]][j];
        }
        for (int j = indexCambiar; j < indexFinal; j++) {
              nuevaGen[indexNuevaGen][j] = pob[parejas[i][1]][j];
              nuevaGen[indexNuevaGen+1][j] = pob[parejas[i][0]][j];
        }
}
```

Figura 4.32. Cruce de material genético.

En esta sección del código se verifica por cada uno de los cruces si ya es momento de realizar una mutación en el bit que se está procesando, por lo que es necesario insertar una sección del código que realice esta comprobación y mute el bit actual en caso de que sea necesario (ver *Figura 4.33*).

```
if (iterarMUTA >= nMUTA) {
      iterarMUTA = 0;
      nuevaGen[indexNuevaGen][j] = !nuevaGen[indexNuevaGen][j];
}
```

Figura 4.33. Mutación en el algoritmo de cruce.

4.6 Pruebas

Con el fin de determinar la fiabilidad y eficiencia del algoritmo genético, es necesario realizar una serie de pruebas controladas para obtener un comportamiento representativo del mismo y así determinar información estadística con la que se puede llegar a conclusiones precisas.

4.6.1 Evaluación de las pruebas

Para el diseño de pruebas se implementó un sistema simple que consiste en determinar las condiciones iniciales tales como los materiales, restricciones y configuraciones del algoritmo genético. Una vez asignados, el sistema se encarga de hacer 1000 ejecuciones completas de la optimización para obtener un comportamiento representativo del algoritmo genético.

Las características generales aplicadas en las pruebas presentadas en este documento son:

- Material 1, E_L = 150GPa, E_T = 10GPa, G_{LT} = 8GPa, ν_{LT} = 0.2, Espesor = 1mm.
- Material 2, E_L = 20GPa, E_T = 2GPa, G_{LT} = 1GPa, ν_{LT} = 0.15, Espesor = 1mm.
- Proporción de buenos individuos = 50%. Se refiere a la mejor mitad de los individuos de la población que sobrevivirán a la siguiente generación; el resto serán reemplazados por los mejores individuos de la nueva generación.
- Criterio de convergencia = 1%. Si el cambio que exista en la media y desviación estándar entre generaciones es menor al 1%, el algoritmo genético termina.
- Frecuencia de mutación = 0.1%. Por cada 1000 bits generados por el algoritmo, uno va a cambiar su valor para aumentar la diversidad de los resultados.

Con estas configuraciones se presentan dos casos de estudio en este trabajo. Antes de eso, se muestra una serie de pruebas para predecir el número de individuos que debería utilizarse en una ejecución basándose en el número de capas que se desean y el porcentaje de error que se puede aceptar. Para ello, se hicieron pruebas con las configuraciones mencionadas arriba pero con variaciones en:

- El número de capas del apilado (2, 4, 8 y 16 capas).
- El número de individuos (de 20 a 110 en incrementos de 10 en 10).

4.6.2 Nivel de error

Para realizar esta predicción se realizaron 40 pruebas (un total de 40,000 ejecuciones completas del algoritmo genético) optimizando E_x para tomar todas las combinaciones de las variantes a estudiar. Cada conjunto de 1000 ejecuciones completas duró entre 12 y 15 minutos en terminar su ejecución, y al terminar cada conjunto se obtiene un archivo con información estadística de cada ejecución junto con una gráfica que hace un resumen de los resultados, incluyendo los valores máximo y mínimo obtenidos, la deviación estándar, la media y su desviación normalizada (ver *Figura 4.34*).

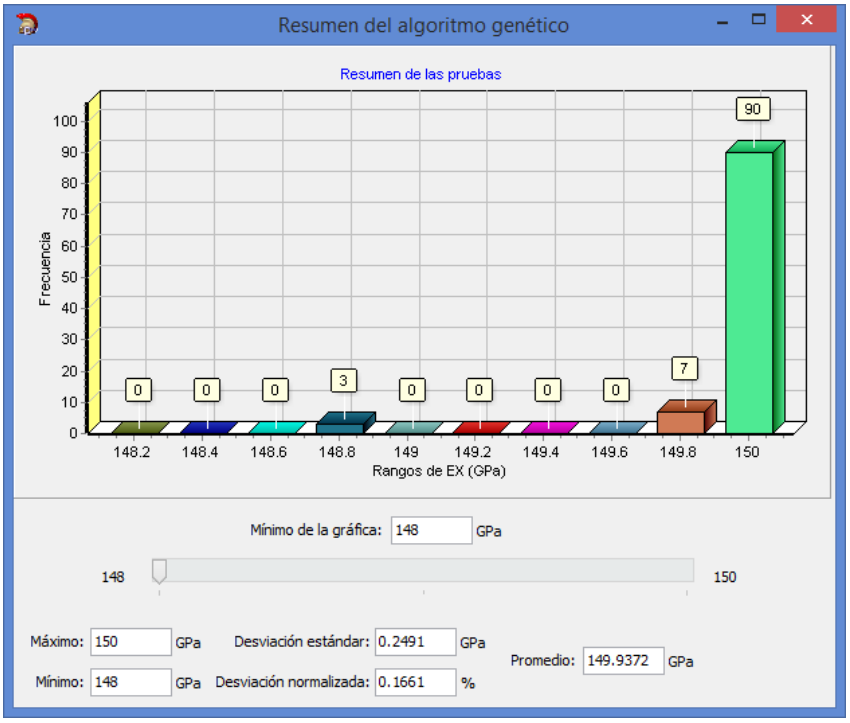

Figura 4.34. Ejemplo de gráfica de resumen.

La desviación normalizada representa el porcentaje de error que se encontró en el conjunto de datos, con la ventaja de que es independiente de las unidades de medida, y se usa para verificar el comportamiento del algoritmo genético y así compararlo con otros casos de prueba. Dicha variable se calcula con la siguiente fórmula:

$$\sigma norm = \frac{\sigma}{\mu} * 100 \quad (4.1)$$

donde σ es la desviación estándar de los valores máximos alcanzados por cada ejecución que se hizo en el conjunto, mientras que μ es la media de dichos máximos.

4.6.3 Predicción de número de individuos de la población

Como se mencionó en las primeras secciones de este documento, una de las razones por las que se decidió realizar esta herramienta es el permitir a las personas que no son expertas en el diseño de materiales compuestos el determinar el mejor arreglo de capas en un laminado compuesto para satisfacer las necesidades que tenga, todo esto mediante un algoritmo genético. Sin embargo, la necesidad de experiencia en el diseño de materiales multicapa fue reemplazada por la necesidad del conocimiento de algunos conceptos de algoritmos genéticos, ya que el usuario debe ser capaz de indicar el número de individuos de la población antes de usar el programa.

Una de las intenciones principales del obtener las desviaciones normalizadas de las pruebas es el hecho de verificar si su comportamiento replicable, es decir, que sea posible modelar las diferencias entre dichas desviaciones para encontrar la forma de predecir el número de individuos que necesita la población de un algoritmo genético para lograr un nivel de error menor o igual a uno determinado por el usuario con un número de capas también predefinido.

De esta forma, se puede seleccionar el número de individuos de la población indicados para las necesidades actuales del usuario, sin necesidad de que esté obligado a indicar este valor bajo el riesgo de que elija un número tan bajo que exista una alta probabilidad de error por parte del algoritmo genético o un valor innecesariamente alto que solamente utilice recursos computacionales sin aportar en gran medida al resultado final.

CAPÍTULO 5. RESULTADOS Y DISCUSIÓN

5.1 Resultados de las pruebas y predicción de individuos

En la *Tabla 5.1* se muestra el condensado de las 40,000 ejecuciones del algoritmo genético mostrando las desviaciones normalizadas para cada una de las combinaciones de número de individuos y número de capas.

Tabla 5.1. Desviaciones normalizadas obtenidas para maximizar Ex.

Individuos	2 capas	4 capas	8 capas	16 capas
20	150.73%	146.65%	130.04%	96.98%
30	68.57%	82.93%	79.59%	65.31%
40	33.61%	47.45%	52.30%	47.01%
50	20.27%	30.72%	37.27%	38.18%
60	12.26%	17.29%	22.49%	28.50%
70	6.73%	11.66%	15.88%	22.71%
80	3.78%	5.44%	13.47%	17.66%
90	1.79%	2.52%	7.86%	15.12%
100	1.32%	2.08%	5.02%	11.92%
110	1.08%	1.50%	4.66%	9.97%

Como se puede apreciar, el porcentaje de error disminuye entre más individuos tenga la población del algoritmo genético, mientras que aumenta junto con el número de capas que se necesita obtener, obteniendo errores menores al 10 por ciento en los siguientes casos:

- Usando dos capas: A partir de 70 individuos.
- Usando cuatro capas: A partir de 80 individuos.
- Usando ocho capas: A partir de 90 individuos.
- Usando dieciséis capas: A partir de 110 individuos.

Esto conduce a la idea de que entre menos individuos tenga la población de un algoritmo genético, el comportamiento del mismo tiende a ser más aleatorio, por lo que los resultados son menos confiables. Mientras tanto, las poblaciones con un número de individuos elevado tienden a tener resultados más predecibles y consistentes, por lo que resultan más útiles para los usuarios. En la *Figura 5.1* se puede apreciar las desviaciones normalizadas graficadas de escala logarítmica para discernir con mayor eficacia las diferencias entre los niveles de error para cada combinación de número de capas y número de individuos.

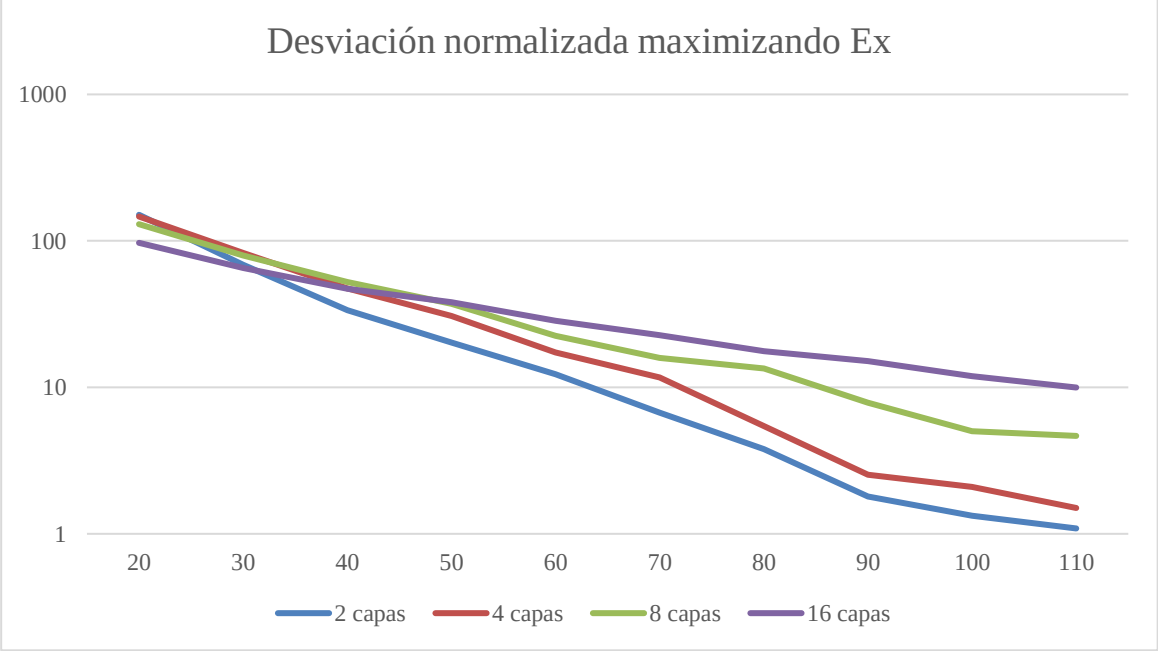

Figura 5.1. Gráfica en escala logarítmica de las desviaciones normalizadas obtenidas.

Los resultados obtenidos en la gráfica sugieren un comportamiento aproximadamente lineal para cada uno de los conjuntos de pruebas por número de capa, por lo que es posible modelar dicho comportamiento utilizando ecuaciones de primer grado, para luego simplificarlas y así conseguir una sola ecuación que permita predecir el número de individuos necesario para lograr un porcentaje de confiabilidad en la ejecución del algoritmo genético proporcionando el número de capas que se desean obtener.

En la *Tabla 5.2* se pueden observar los logaritmos de cada uno de los resultados anteriores, los cuales serán usados para la predicción ya que resultan más efectivos para la predicción debido a su escala de valores.

Tabla 5.2. Logaritmo de los resultados de las pruebas en módulo de Young Ex.

Individuos	**2 capas**	**4 capas**	**8 capas**	**16 capas**
20	2.178	2.166	2.114	1.986
30	1.836	1.918	1.900	1.815
40	1.526	1.676	1.718	1.672
50	1.306	1.487	1.571	1.581
60	1.088	1.237	1.352	1.454
70	0.828	1.066	1.200	1.356
80	0.578	0.736	1.129	1.247
90	0.254	0.402	0.895	1.179
100	0.123	0.319	0.701	1.076
110	0.036	0.176	0.668	0.998

A continuación se muestran las gráficas correspondientes a la tabla anterior, una por cada número de capas que fue muestreado. Para todas estas gráficas se pueden apreciar con puntos los resultados obtenidos, cuya posición está definida por su número de individuos (eje horizontal) y el logaritmo de su desviación normalizada (eje vertical) (ver *Figura 5.2, Figura 5.3, Figura 5.4* y *Figura 5.5*).

Además, junto con los resultados de la tabla se presentan tres atributos adicionales:

- La línea continua que se encuentra cerca de los resultados representa la regresión lineal que modela el comportamiento de los datos proporcionados.
- En la parte superior de la gráfica, se encuentra el coeficiente de correlación R^2, el cual es un valor entre 0 y 1 y sirve para determinar la calidad de dicha ecuación para replicar los resultados y la variación de los datos. En otras palabras, entre más cerca esté este valor a 1, mejor será la aproximación que hace la ecuación que modela el comportamiento deseado.

- Debajo de dicho coeficiente, se muestra la ecuación de la recta indicada anteriormente, cuyos elementos son trabajados más adelante en esta sección.

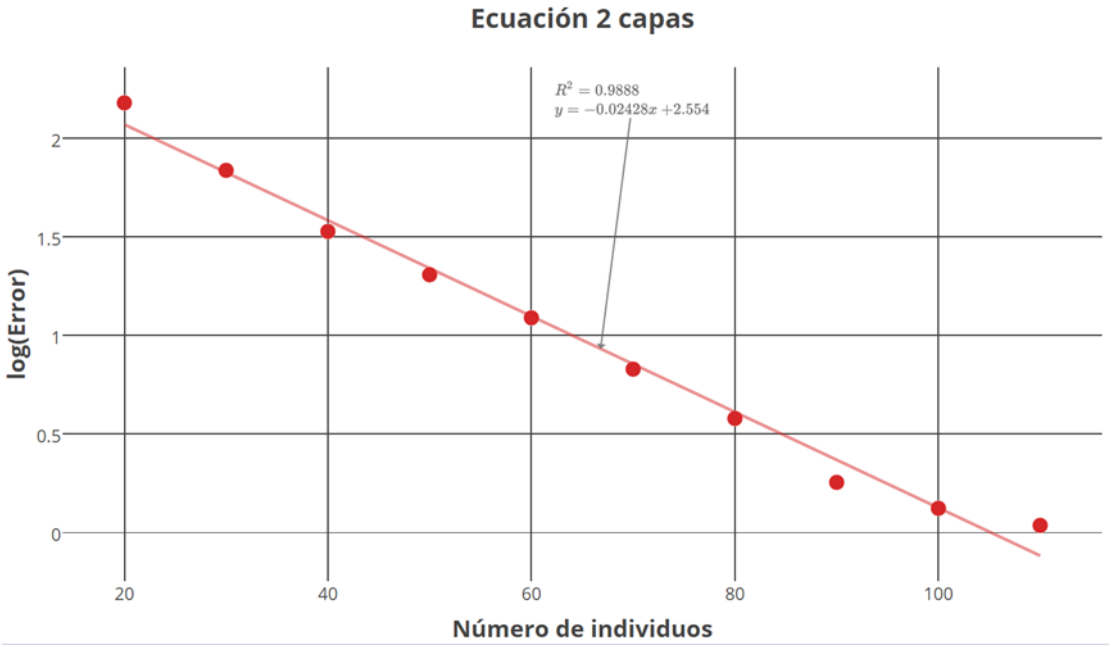

Figura 5.2. Ecuación de la recta para 2 capas.

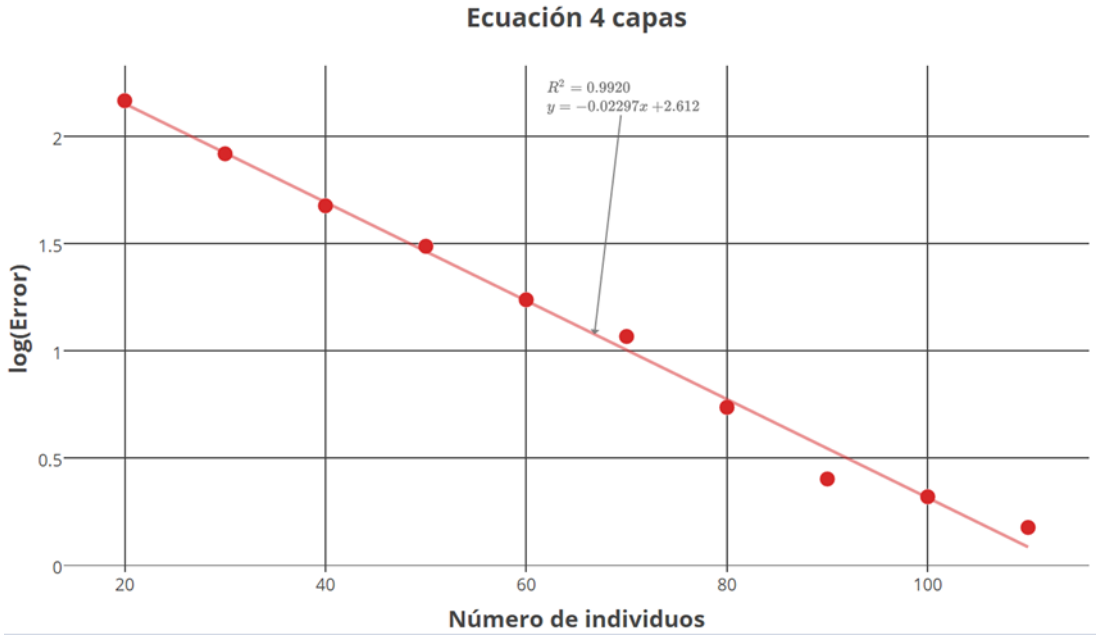

Figura 5.3. Ecuación de la recta para 4 capas.

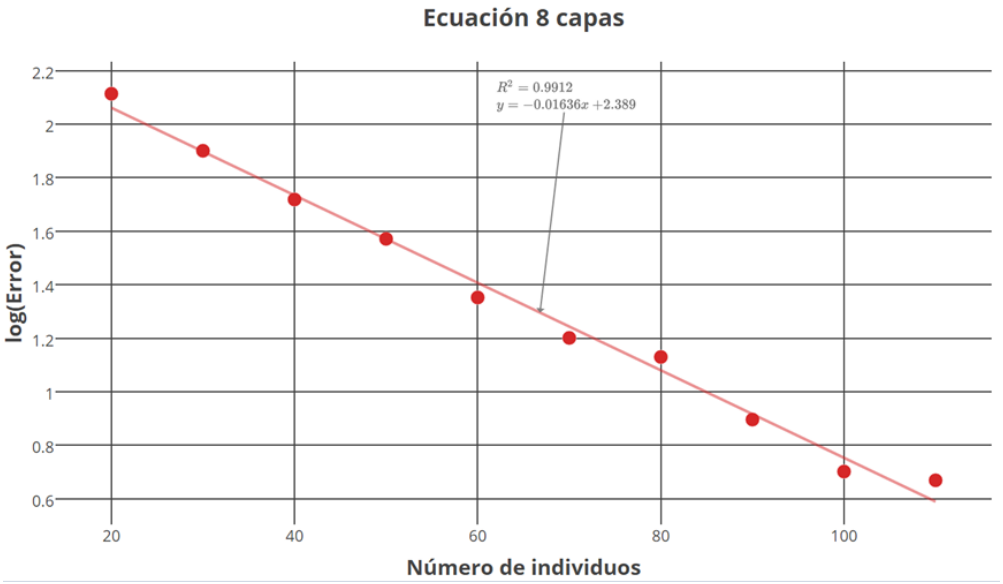

Figura 5.4. Ecuación de la recta para 8 capas.

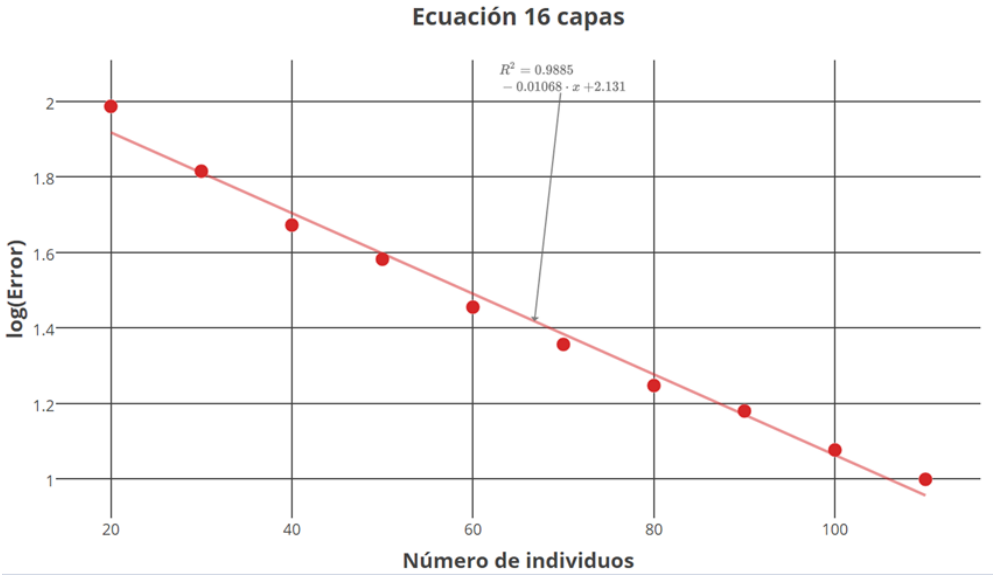

Figura 5.5. Ecuación de la recta para 16 capas.

Debido a que las aproximaciones lineales obtenidas tienen un coeficiente de correlación mayor a 0.98, es decir, tienen un nivel de precisión mayor al 98%, se puede afirmar que las ecuaciones calculadas son confiables para modelar el comportamiento deseado. Esto permite que los sistemas puedan ser simplificados de una mejor manera, teniendo la posibilidad de obtener una sola ecuación para la predicción del número de individuos en el algoritmo genético.

En la siguiente tabla se condensan las propiedades de las ecuaciones que se obtuvieron con el paso anterior (ver *Tabla 5.3*), siendo la columna A la que enlista los coeficientes de dichas ecuaciones, mientras que la columna B despliega las literales de las mismas.

Tabla 5.3. Coeficientes y literales de las ecuaciones calculadas.

Capas	A	B
2	-2.43E-02	2.5543
4	-2.30E-02	2.6118
8	-1.64E-02	2.3889
16	-1.07E-02	2.1312

Estos datos son graficados de la misma forma que los resultados de las pruebas, y se obtienen a su vez sus aproximaciones lineales, junto con su ecuación y coeficiente de correlación R^2 correspondientes con los cuales se puede elaborar la ecuación final de predicción (ver *Figura 5.6* y *Figura 5.7*).

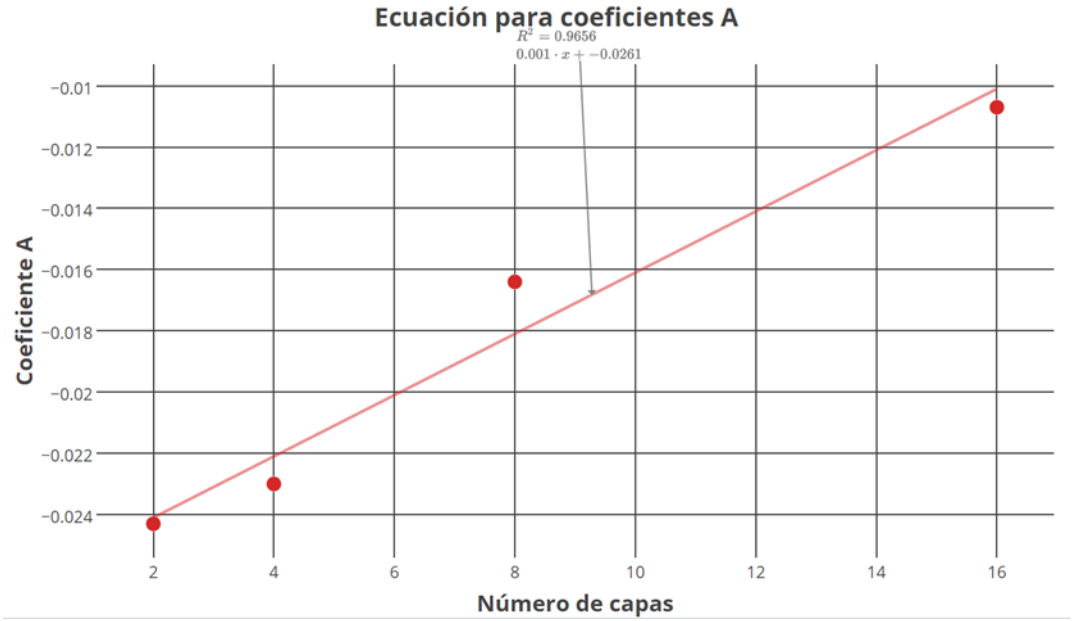

Figura 5.6. Ecuación de la recta para los coeficientes A.

La gráfica anterior muestra un coeficiente de correlación de 0.96 (96% de precisión) y proporciona la ecuación $y = 0.0009x - 0.026$. La variable x en esta ecuación corresponde al número de capas del apilado resultante, mientras que y es el coeficiente calculado de A (Ac); a su vez, el coeficiente de x y la literal pasan a ser conocidas por αA y βA para fines de legibilidad en la ecuación. Entonces, la ecuación definida anteriormente puede ser expresada como $Ac = \alpha A * capas + \beta A$.

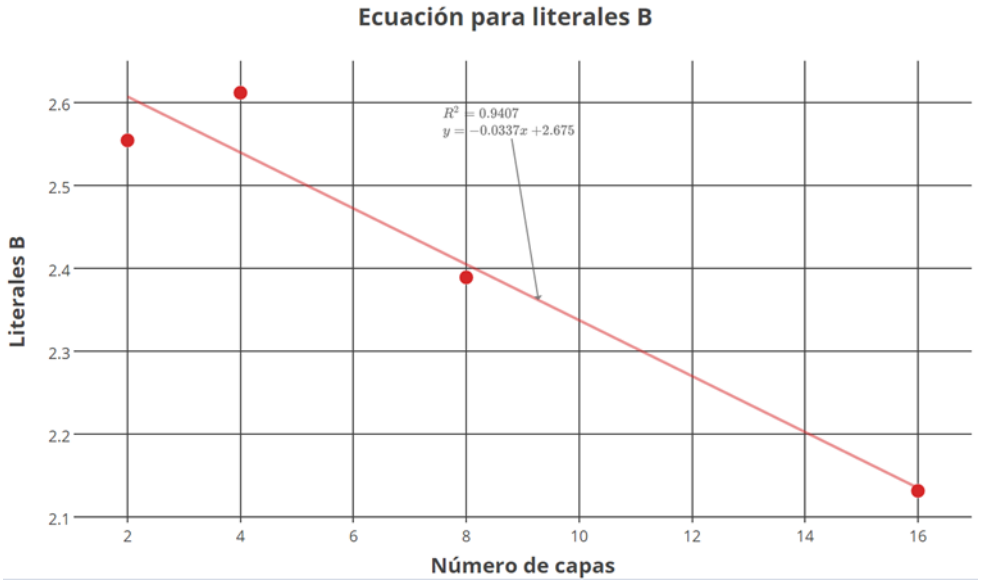

Figura 5.7. Ecuación de la recta para las literales B.

De la misma forma, la ecuación que modela el comportamiento de las literales B puede expresarse como $Bc = \alpha B * capas + \beta B$ para futuras referencias. En la *Tabla 5.4* se enlistan los coeficientes y las literales que utilizan las ecuaciones de Ac y Bc.

Tabla 5.4. Coeficientes y literales de las columnas A y B.

αA	αB	βA	βB
1.00E-03	-2.61E-02	-3.37E-02	2.6746

La última ecuación necesaria para generar la predicción del número de individuos es la que utiliza tanto Ac y Bc para calcular la desviación normalizada del comportamiento. Dicha ecuación se muestra a continuación:

$$\log(\sigma norm) = Ac * \#Individuos + Bc \quad (5.1)$$

Ac y Bc pueden ser calculadas gracias a las ecuaciones obtenidas anteriormente, mientras que $\sigma norm$ es la desviación normalizada que especifica el usuario dependiendo de la precisión que necesita y $\#Individuos$ es la variable a buscar. Reemplazando las variables que pueden ser calculadas y haciendo el despeje de $\#Individuos$, se obtiene la ecuación final para la predicción, la cual se muestra enseguida:

$$\#Individuos = \frac{\log(\sigma Norm) - (\alpha \mathrm{B} * \#Capas + \beta \mathrm{B})}{\alpha \mathrm{A} * \#Capas + \beta A} \quad (5.2)$$

Para comprobar la eficacia de esta función, se realizaron pruebas con las mismas características que las anteriores, diferenciando solamente en dos puntos:

1. El número de individuos.- Ya que el objetivo de estas pruebas es comprobar si el nivel de fiabilidad del algoritmo genético se mantiene, se realizó un cálculo para saber cuántos individuos debe tener la población para conseguir un nivel de error menor al 10% para apilados de 2 capas; cálculo que dio como resultado 56 individuos.
2. La propiedad a optimizar.- Se asignaron varias pruebas optimizando diferentes propiedades a las que se habían hecho hasta el momento para observar el cambio que surge en este ámbito. Las propiedades que fueron puestas a prueba son:
    - Matriz A elemento 1,1.
    - Matriz B elemento 1,1.
    - Matriz B elemento 2,2.
    - Módulo de Young aparente E_y.
    - Módulo cortante aparente G_{xy}.

Los resultados obtenidos de las pruebas se enlistan en la *Tabla 5.5*, la cual se muestra a continuación:

Tabla 5.5. Pruebas de optimización con otras propiedades.

Propiedad a optimizar	**Desviación normalizada**
Matriz A 1,1	0.9883%
Matriz B 1,1	3.1922%
Matriz B 2,2	1.4870%
E_y	8.7027%
G_{xy}	1.6645%

Como se puede apreciar, ninguno de los resultados mostrados contiene una desviación normalizada mayor al 10% solicitado en la ecuación modelada anteriormente, lo cual indica que la predicción realizada es precisa y fiable para la configuración del algoritmo genético.

5.2 Casos de estudio multiobjetivo

Por otro lado, se realizaron pruebas de optimizaciones multiobjetivo para apreciar el comportamiento del algoritmo en otro tipo de situaciones y verificar el campo de soluciones óptimas encontradas con más de una variable a optimizar.

5.2.1 Primer caso de estudio

La primera prueba multiobjetivo busca la maximización simultánea de E_x y E_y, buscando un error normalizado menor al 10% usando los mismos criterios que en las pruebas con un solo objetivo. En la *Figura 5.8* se aprecia la gráfica que contiene los valores óptimos encontrados para ambas propiedades, comparando los módulos de Young aparentes E_x y E_y.

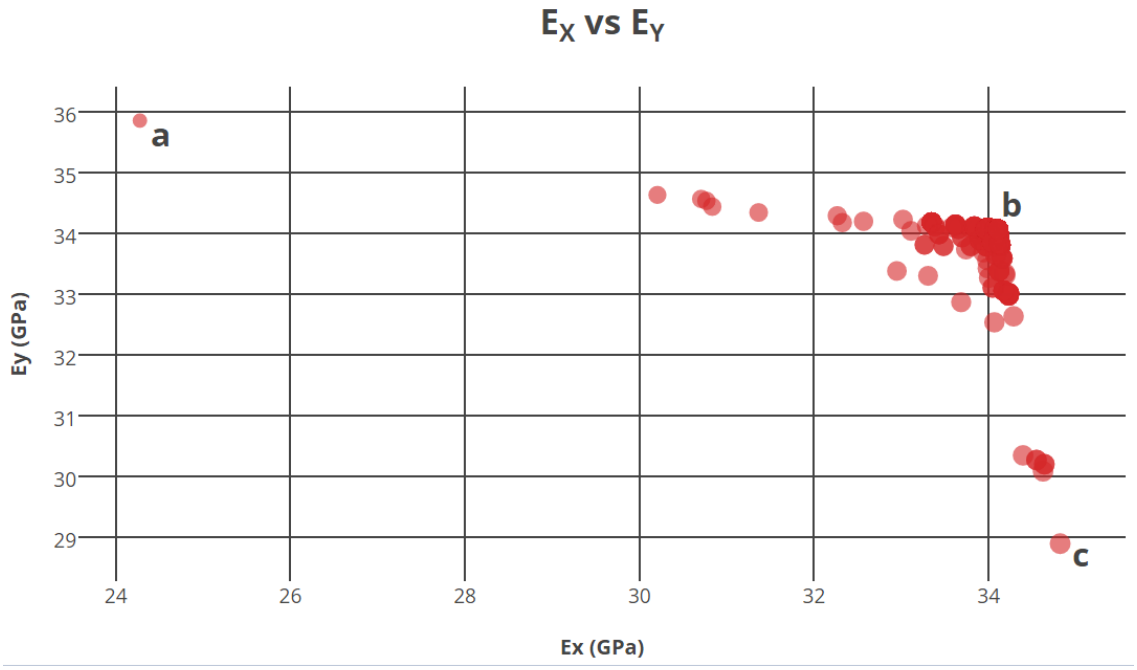

Figura 5.8. Gráfica de los valores óptimos de Ex y Ey al ser maximizados.

En esta gráfica se aprecia que las ejecuciones del algoritmo genético para la optimización de estas dos propiedades tiende hacia un punto en el que se intersectan ambos máximos (inciso b, explicado más adelante), con una desviación normalizada de 3.17%, bastante menor al 10% esperado por los resultados anteriores.

Las soluciones representadas con letras en la imagen corresponden a los siguientes apilados:

a) Material 1 a 22° en capa 1, Material 1 a -85° en capa 2. E_x=24.2 GPa y E_y=35.8 GPa.
b) Material 1 a 90°, Material 1 a 0°. E_x = 34.1 GPa y E_y=34 GPa.
c) Material 1 a 0°, Material 2 a 76°. E_x=34.8 GPa y E_y=28.8 GPa.

5.2.2 Segundo caso de estudio

Por otra parte, al realizar una maximización de E_x y G_{xy} usando los mismos criterios que en la prueba anterior, se obtiene una gráfica diferente (ver *Figura 5.9*).

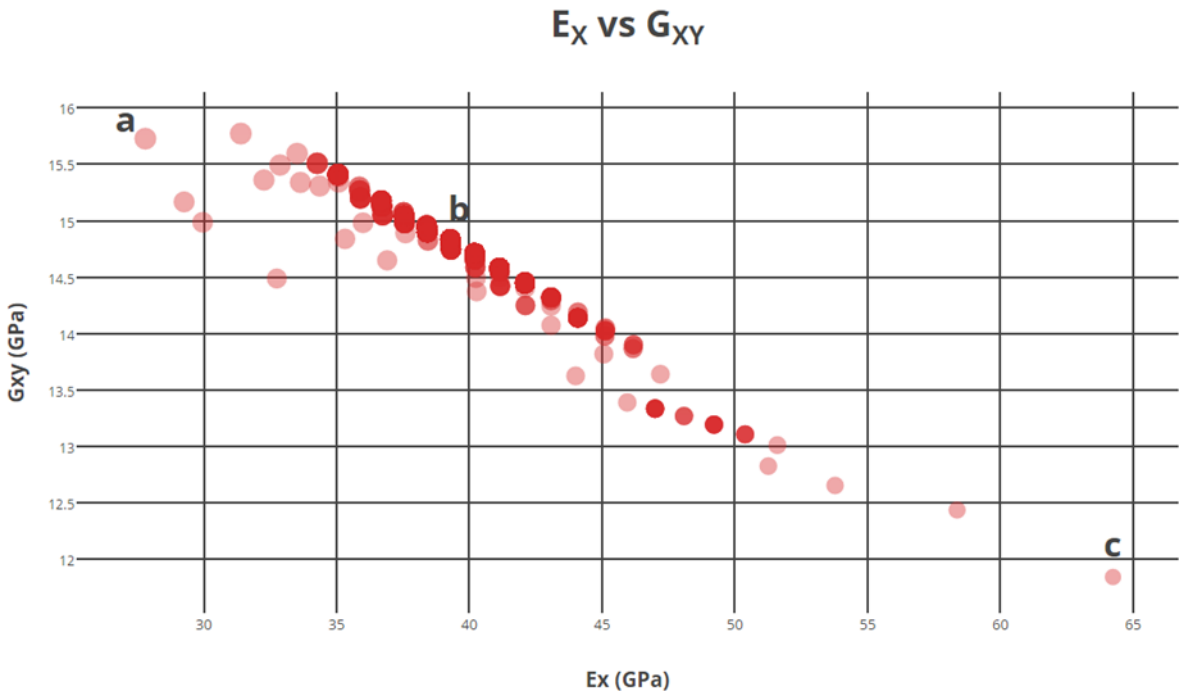

Figura 5.9. Gráfica de los valores óptimos de Ex y Gxy al ser maximizados.

En esta prueba los resultados indican que, en lugar de aproximarse a un punto máximo, los resultados tienden a formar una línea de soluciones óptimas, lo cual tiene como causa la existencia de múltiples soluciones al problema de optimización planteado. Además, su error normalizado fue de 3.09%, nuevamente debajo de lo esperado.

Las soluciones representadas con letras en la gráfica corresponden a los siguientes apilados:

a) Material 1 a -34°, Material 1 a 45°. E_x=27.7 GPa, G_{xy}=15.7 GPa.
b) Material 1 a -31°, Material 1 a 31°. E_x=40.1 GPa, G_{xy}=14.7 GPa.
c) Material 1 a -21°, Material 1 a 21°. E_x= 64.2 GPa, G_{xy}=11.8 GPa.

5.2.3 Tercer caso de estudio

Finalmente, se realizó una prueba maximizando E_x, E_y y E_{XB} al mismo tiempo para ilustrar las capacidades del método. Además, se manejaron apilados de 16 capas de manera que se construyan soluciones con una estructura más extensa (ver *Figura 5.10*).

De nueva cuenta, al realizar esta optimización se encuentra que existen múltiples soluciones pero esta vez, en lugar de formar una línea de soluciones óptimas, los resultados tienden a formar una superficie en el espacio de las tres variables a maximizar.

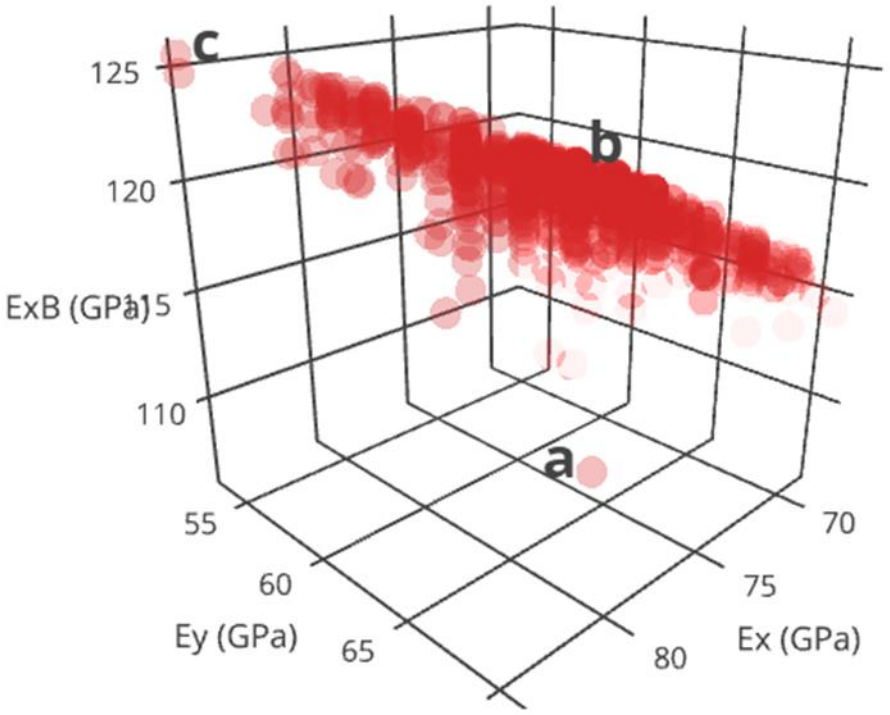

Figura 5.10. Gráfica de los valores óptimos de Ex, Ey y E_{XB} al ser maximizados.

Los resultados marcados con letras en la gráfica corresponden a los siguientes apilados:

a) Orden de las capas: 1 a 0°, 1 a 0°, 2 a 90°, 2 a 0°, 2 a -45°, 2 a 90°, 1 a 90°, 2 a 0°, 1 a 90°, 1 a 90°, 2 a 45°, 2 a 90°, 1 a 0°, 2 a -45°, 2 a 0°, 1 a 0°. E_x=74.1 GPa, E_y=63.5 GPa, E_{XB}=106.4 GPa.

b) Orden de las capas: 1 a 0°, 1 a 0°, 1 a 0°, 2 a 90°, 1 a 90°, 1 a 90°, 2 a 90°, 1 a 90°, 1 a 90°, 1 a 90°, 2 a 90°, 1 a 0°, 1 a 0°, 2 a -45°, 2 a 0°, 1 a 0°. E_x=72.6 GPa, E_y=67.5 GPa, E_{XB}=115.2 GPa.

c) Orden de las capas: 1 a 0°, 2 a 0°, 1 a 0°, 2 a 0°, 1 a -90°, 2 a 90°, 2 a 90°, 2 a -45°, 1 a 90°, 2 a 0°, 1 a 90°, 2 a 90°, 1 a 0°, 2 a -45°, 2 a 0°, 1 a 90°. E_x=83.1837 GPa, E_y=57.3 GPa, E_{XB}=125.2 GPa.

CAPÍTULO 6. CONCLUSIONES

Para este proyecto se utilizó el enfoque de los algoritmos genéticos para generar una herramienta que encuentra materiales multicapa de materiales compuestos que optimicen propiedades relacionadas con la rigidez de los mismos. Las soluciones se encuentran basándose en criterios establecidos por el usuario, tales como una lista de materiales compuestos que el algoritmo puede utilizar, los espesores de dichos materiales, las indicaciones sobre cuáles propiedades se deben optimizar y qué tipo de optimización se desea para cada una.

En comparación a los algoritmos de este tipo que se han realizado anteriormente en el ambiente de los materiales compuestos, el que se presenta en este documento tiene la ventaja de que provee una solución más general a lo establecido. La razón por la que se realiza esta afirmación consta de dos partes:

1 El usuario tiene la capacidad de elegir una o una combinación de las 53 propiedades manejadas por el algoritmo genético, mientras que otros trabajos se centran en una o dos propiedades simultáneas.

2 Para cada una de las propiedades seleccionadas puede elegir qué tipo de optimización desea (maximizar, minimizar, acercar a un valor, buscar un valor mayor o menor a uno establecido o buscar un valor que esté dentro de un rango), a diferencia de otros algoritmos que solamente buscan la maximización o minimización del objetivo.

El principal beneficio que tiene este software es que, una vez implementado en su totalidad en el software MAC LAM, estará disponible a nivel mundial de forma gratuita para su uso, permitiendo la optimización de laminados compuestos en cualquier nivel que se desee aplicar. El enfoque que más se aplica para este programa es didáctico, ya que en él se pueden apoyar estudiantes de materiales para comprender y adquirir nociones del diseño y simulación de los materiales compuestos y los materiales multicapa, pero su uso se puede extender a nivel científico o industrial en caso que se necesite este algoritmo para la creación de nuevos materiales.

En base a los resultados obtenidos, se puede afirmar que el algoritmo genético tiene un desempeño satisfactorio, ya que las pruebas realizadas proporcionan un porcentaje de error menor al 10% que se calculó con la ecuación de predicción del número de individuos, lo cual indica que tanto el algoritmo como la ecuación generada tienen un comportamiento estable. Esto, combinado con el tiempo de ejecución obtenido por el algoritmo, el cual es de menos de un segundo por cada ejecución completa, hace que esta herramienta sea atractiva para la optimización de rigideces en los laminados de materiales compuestos. Tomando en cuenta lo indicado anteriormente, se puede afirmar que el proyecto confirmó la hipótesis y cumplió con los objetivos establecidos.

A futuro se planea que el algoritmo tome en cuenta más necesidades del usuario, como aumentar su flexibilidad al permitirle plantear una función matemática en lugar de una variable para la evaluación de los laminados en la función costo, así como extender la funcionabilidad, implementando la optimización de propiedades de esfuerzos y resistencia y permitir la configuración de los algoritmos de selección de parejas y de cruce.

Por otra parte, la ecuación para la predicción del número de individuos también está contemplada para la aplicación de mejoras, realizando un material que involucre un número de capas cada vez mayor para proporcionarle un comportamiento más representativo del algoritmo genético y así muestre resultados más precisos a los obtenidos hasta el momento.

CAPÍTULO 7. BIBLIOGRAFÍA

Ahmad, S., & Bergen, S. (2010). A genetic algorithm approach to the inverse problem of treatment planning for intensity-modulated radiotherapy. *Biomedical Signal Processing and Control 5*, 189-195.

Bevilacqua, V., Mastronardi, G., & Piscopo, G. (2007). Evolutionary approach to inverse planning in coplanar radiotherapy. *Image and Vision Computing 25*, 196-203.

C. Maletta, L. P. (2004). On the determination of mechanical properties of composite laminates using genetic algorithms. *International Journal of Mechanics and Materials in Design 1*, 199-211.

Dorigo, M. a. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput. 1*, 53-66.

Forrest, S. (1993). Genetic algorithms: principles of natural selection applied to computation. *Science, New Series, Volume 261, Issue 5123*, 872-878.

Gestal, M., & Daniel, R. (2010). *Introducción a los algoritmos genéticos y la programación genética.* La Coruña: Universidad de Coruña.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co.

Graeme, M. (2002). *The Theory of Composites.* Cambridge: Cambridge University Press.

Haupt, R. a. (2004). *Practical genetic algorithms, second edition.* New Jersey, USA: John Wiley & Sons, Inc.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

HU, N. (2012). *Composites and Their Application, first edition.* InTech.

Jones, R. (1999). *Mechanics of composite materials, second edition.* Philadelphia, USA: Taylor & Francis Ltd.

Kirkpatrick, s. C. (1983). Optimization by simulated annealing. *Science 220*, 671-680.

Lavine, B., & Moores, A. (2008). Genetic Algorithms in Analytical Chemistry. *Analytical Letters, 32:3*, 433-445.

Leardi, R. (2001). Genetic algorithms in chemometrics and chemistry: a review. *JOURNAL OF CHEMOMETRICS 15*, 559-569.

M. Seyyed, A. R. (2013). Multi-objective design optimization of composite laminates using discrete shuffled frog leaping algorithm. *Journal of Mechanical Science and Technology 27*, 1791-1800.

M. T. McMahon, L. T. (1998). A Fortran 90 genetic algorithm module for composite laminate structure design. *Engineering with Computers*, 260-273.

Melián, B., Moreno, J., & Moreno, M. (2009). Algoritmos genéticos. Una visión práctica. *Números. Revista de Didáctica de las Matemáticas. Volumen 71*, 29-47.

Nozari, H., Rezai Rad, G. A., Pourmajidian, M., & Abdul-Wahab, A. K. (2011). Parametric Dictionary Design Using Genetic Algorithm for Biomedical Image De-noising Application. *IFMBE Proceedings 35*, 704-707.

P. Kere, J. K. (2002). Multicriterion optimization of composite laminates for maximum failure margins with an interactive descent algorithm. *Struct Multidisc Optim 23*, 436-447.

P. Sargent, D. I. (1995). Design of laminate composite layups using genetic algorithms. *Engineering with Computers*, 59-69.

Parsopoulos, K. E. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural computing. Netherlands: Kluwer Academic 1*, 235-306.

R. Candela, G. C. (2010). Composite laminates buckling optimization through Lévy based ant colony optimization. *IEA/AIE 2010, Part II, LNAI 609*, 288-297.

S. Deng, P. P. (2005). A solution to the stacking sequence of a composite laminate plate with constant thickness using simulated annealing algorithms. *Int J Adv Manuf Technol 26*, 499-504.

S. Hwang, Y. H. (2014). A genetic algorithm for the optimization of fiber angles in composite laminates. *Journal of Mechanical Science and Technology 28*, 3163-3169.

S. Rajasekaran, J. R. (2000). Application of a genetic algorithm to optimal lay-up of shells made of composite laminates. *Mechanics of Composite Materials 36*, 271-278.

Schwefel, H. (1995). *Evolution and Optimum Seeking.* New York: Wiley.

Venkatasubramanian, V., Chan, K., & M., C. J. (1994). Evolutionary Design of Molecules with Desired Properties Using the Genetic Algorithm. *Laboratory for Intelligent Process Systems, School of Chemical Engineering, Purdue University*, West Lafayette, Indiana.

ANEXO A. ENTREVISTA

1.- En sus propias palabras, ¿cuál es el objetivo de MAC LAM?

MacLam tiene como objetivo principal el poder ofrecer un software robusto capaz de simular materiales compuestos de tipo multicapa con refuerzo de fibras unidireccionales para cada una de estas. Entre sus objetivos específicos se encontraba lograr la simulación con lo menor posible de coste computacional, esto debido a que en la época en que el sistema nace, hacer una simulación mediante elemento finito de un material compuesto era casi imposible. Aún hoy en día simular materiales compuestos cuesta mucho computacionalmente hablando pero el sistema logró ese objetivo. Otro de los objetivos específicos era lograr que las universidades o escuelas que enseñan los materiales compuestos tuvieran una herramienta gratuita a la mano como apoyo a la clase teórica.

2.- ¿De qué partes se compone este software?

Principalmente de 3 módulos. El primero de ellos es conocido como micromecánica, el cual a grandes rasgos este módulo se encarga de predecir las propiedades de un material compuesto a partir de la combinación de dos materiales, un material matriz, por lo general una resina, y un material compuesto, en este caso el refuerzo se trata de fibras. No tiene mucho de novedad en el área del cálculo ya que se emplearon fórmulas que se encuentran en la mayoría de la literatura, lo que si era novedoso es que permitía evaluar los diferentes resultados y le permitía al usuario escoger el que considerara más adecuado, aparte de que la salida de este módulo se convertía directamente en la entrada del siguiente.

El segundo módulo se conoce como micromecánica, aquí el software permite realizar capaz de distintos materiales compuestos, en distintos espesores y orientaciones de las fibras, y estas pasan a través de un motor de resolución de ecuaciones el cual permite obtener los resultados de esfuerzos, deformaciones, etc. resultantes de la combinación de las capas. Lo novedoso del sistema radicaba en que era muy fácil formar el multicapa, ya que contaba con herramientas que permitían duplicar capas, apilarlas de acuerdo a una simetría, además de poder guardar los trabajos y resultados.

El tercer módulo se le denominó Deilam, su función principal consistía en poder predecir los esfuerzos interlaminares, es decir los esfuerzos que existen entre capa y capa. Esto aún en la actualidad no existe software que incluya esa predicción entre sus módulos.

3.- ¿Para quién está enfocado?

Tiene dos enfoques principalmente, el primero es para que los alumnos de instituciones educativas que incluyen un plan con materiales compuestos puedan hacer uso de una herramienta con ese poder y sin costo. Este primer objetivo se alcanzó ya que la escuela de puentes y caminos de Francia tiene entre sus materias de materiales compuestos el uso y análisis del software. El segundo enfoque era poder auxiliar a la gente que se dedica al diseño de materiales compuestos como medio de investigación una herramienta poderosa capaz de predecir comportamientos y propiedades de materiales difícilmente simulados.

4.- ¿Qué es lo que les gustaría mejorar del programa?

Existe una lista grande de mejoras, así en este momento puedo decir que le urge un cambio a la interfaz gráfica, ya en este momento es vieja y un poco compleja de usar. La base de datos ya requiere de poder ser compartida para que los mismos usuarios de una red tengan la capacidad de acceder a trabajos de algunas otras personas, aparte que eso lo hacer portable de alguna manera. Otro detalle es que sería interesante tener una herramienta que fuera capaz de predecir propiedades de materiales haciéndole propuestas al usuario de cuál sería el material, capa, espesor, orientación de fibras, más adecuado para el trabajo propuesto. Falta un programa de difusión más amplio, así como el de renovar su sitio web, para hacerlo más atractivo y fácil de acceder.

5.- ¿Cómo quisieran que se implementara esa mejora?

En el caso de la interfaz gráfica se tendría que ver las tendencias actuales en cuanto a componentes para poder determinar cuál es el mejor acomodo, presentación y manejo de datos. El sitio web tendría que ser reconstruido por completo haciendo algo dinámico donde se pudieran captar las opiniones sobre el software. En cuanto a lo del módulo que auxilia al diseñador, sería importante el uso de alguna

técnica de inteligencia artificial que pueda predecir con cierta certidumbre el material óptimo para las condiciones de trabajo propuestas.

6.- ¿Cuál es el proceso normal que se sigue para diseñar un acomodo de capas?

Todo se basa en la experiencia del diseñador y en los requerimientos de trabajo, es decir si estamos buscando un material compuesto muy resistente pero que sea de muy bajo costo, es imposible pensar en fibra de carbono debido a que su resistencia es adecuada pero el costo no lo es, así que lo primero es seleccionar los materiales adecuados, lo segundo es pensar en cómo se acomodaran las cargas o fuerzas a las que será sometido el material, no es lo mismo en pensar un material que soporta fuerzas cortantes a uno que solo será tensionado en sus extremidades. Además hay ciertas reglas de tamaños y número total de capas, sin embargo todo se basa en la experiencia del diseñador y en un trabajo de prueba y error guiado.

7.- ¿De qué depende el tiempo que tardará una persona en encontrar un diseño adecuado?

Principalmente de su experiencia o del uso de material bibliográfico que lo auxilie.

8.- ¿Cuáles datos son los que se piden para predecir el comportamiento de un material multicapa?

Los básicos son los módulos de Young, coeficiente de Poisson, los módulos de cortante, el tamaño de espesor de capa, la dirección del material reforzante, etc.

9.- ¿Qué resultados devuelven sus algoritmos para compósitos laminados?

En el segundo módulo o macromecánica, presenta gráficas de esfuerzo-deformación, gráficas de zonas de tolerancias a la deformación, etc. En el caso de Deilam se presentan gráficas de esfuerzos por medio de mapas de colores, gráficas normales donde se muestran los esfuerzos por regiones, etc.

10.- ¿Cómo saben si el resultado obtenido es bueno o malo?

Por dar un ejemplo, si yo diseño un material cuyo punto de cedencia está en 500MPa y la simulación me arroja que el esfuerzo máximo está por encima de ese punto de cedencia, puedo deducir entonces

que el material va a fallar y que no es un diseño adecuando para mi problema. O si se lograron los resultados pero me excedí en las dimensiones de las capas. En realidad existen muchos criterios que van de la mano con el objetivo principal del diseño.

11.- ¿Existe algún detalle al que se le deba prestar especial atención para el módulo que se va a desarrollar?

En primera instancia debe de ser un módulo muy sencillo de utilizar siguiendo la lógica de la creación del software, segundo sería interesante que ofreciera una o un número de respuestas para que el usuario pudiera elegir la que más convenga a sus intereses. Y siguiendo con la filosofía del software, este módulo más las modificaciones deberían tener la capacidad de ejecutarse en cualquier equipo sin necesidad de un gran poder de cómputo.

ANEXO B. INICIALIZACIÓN DE LA VENTANA DE CONFIGURACIÓN

```
__fastcall TFAlgGen::TFAlgGen(TComponent* Owner)
        : TForm(Owner)
{

        //Tabla
        idiomaComponentes[0][0] = "Nombre";
        idiomaComponentes[0][1] = "Orientación";
        idiomaComponentes[0][2] = "e";
        idiomaComponentes[0][3] = "Criterio";
        idiomaComponentes[0][4] = "EL";
        idiomaComponentes[0][5] = "ET";
        idiomaComponentes[0][6] = "EN";
        idiomaComponentes[0][7] = "NuLT";
        idiomaComponentes[0][8] = "NuLN";
        idiomaComponentes[0][9] = "GLT";
        idiomaComponentes[0][10] = "GLN";
        idiomaComponentes[0][11] = "GTN";
        idiomaComponentes[0][12] = "Alpha L";
        idiomaComponentes[0][13] = "Alpha T";
        idiomaComponentes[0][14] = "Alpha LT";
        idiomaComponentes[0][15] = "Beta L";
        idiomaComponentes[0][16] = "Beta T";
        idiomaComponentes[0][17] = "Beta LT";
        idiomaComponentes[0][18] = "Xt";
        idiomaComponentes[0][19] = "Xc";
        idiomaComponentes[0][20] = "Yc";
        idiomaComponentes[0][21] = "S";
        idiomaComponentes[0][22] = "F12";

        //---------------------Ingles---------------------------------------------
        //Tabla
        idiomaComponentes[1][0] = "Name";
        idiomaComponentes[1][1] = "Orientation";
        idiomaComponentes[1][2] = "e";
        idiomaComponentes[1][3] = "Criteria";
        idiomaComponentes[1][4] = "EL";
        idiomaComponentes[1][5] = "ET";
        idiomaComponentes[1][6] = "EN";
        idiomaComponentes[1][7] = "NuLT";
        idiomaComponentes[1][8] = "NuLN";
        idiomaComponentes[1][9] = "GLT";
        idiomaComponentes[1][10] = "GLN";
        idiomaComponentes[1][11] = "GTN";
        idiomaComponentes[1][12] = "Alpha L";
        idiomaComponentes[1][13] = "Alpha T";
        idiomaComponentes[1][14] = "Alpha LT";
        idiomaComponentes[1][15] = "Beta L";
        idiomaComponentes[1][16] = "Beta T";
        idiomaComponentes[1][17] = "Beta LT";
        idiomaComponentes[1][18] = "Xt";
        idiomaComponentes[1][19] = "Xc";
```

```
idiomaComponentes[1][20] = "Yc";
idiomaComponentes[1][21] = "S";
idiomaComponentes[1][22] = "F12";

variables[0] = "EX";
variables[1] = "EY";
variables[2] = "GXY";
variables[3] = "EXB";
variables[4] = "EYB";
variables[5] = "NuXY";
variables[6] = "NuYX";
variables[7] = "Alpha X";
variables[8] = "Alpha Y";
variables[9] = "Alpha XY";
variables[10] = "Beta X";
variables[11] = "Beta Y";
variables[12] = "Beta XY";
variables[13] = "ABBD";
variables[14] = "F";

lstGroups[0] = groupEX;
lstGroups[1] = groupEY;
lstGroups[2] = groupGXY;
lstGroups[3] = groupEXB;
lstGroups[4] = groupEYB;
lstGroups[5] = groupNuXY;
lstGroups[6] = groupNuYX;
lstGroups[7] = groupAlphaX;
lstGroups[8] = groupAlphaY;
lstGroups[9] = groupAlphaXY;
lstGroups[10] = groupBetaX;
lstGroups[11] = groupBetaY;
lstGroups[12] = groupBetaXY;
lstGroups[13] = groupABBD;
lstGroups[14] = groupF;

lstCombo[0] = comboEX;
lstCombo[1] = comboEY;
lstCombo[2] = comboGXY;
lstCombo[3] = comboEXB;
lstCombo[4] = comboEYB;
lstCombo[5] = comboNuXY;
lstCombo[6] = comboNuYX;
lstCombo[7] = comboAlphaX;
lstCombo[8] = comboAlphaY;
lstCombo[9] = comboAlphaXY;
lstCombo[10] = comboBetaX;
lstCombo[11] = comboBetaY;
lstCombo[12] = comboBetaXY;

lstText = new TLabeledEdit**[13];
for (int i = 0; i < 13; i++) {
      lstText[i] = new TLabeledEdit*[3];
```

```
        }
        lstText[0][0] = txtEXIgual;
        lstText[0][1] = txtEXDesde;
        lstText[0][2] = txtEXHasta;
        lstText[1][0] = txtEYIgual;
        lstText[1][1] = txtEYDesde;
        lstText[1][2] = txtEYHasta;
        lstText[2][0] = txtGXYIgual;
        lstText[2][1] = txtGXYDesde;
        lstText[2][2] = txtGXYHasta;
        lstText[3][0] = txtEXBIgual;
        lstText[3][1] = txtEXBDesde;
        lstText[3][2] = txtEXBHasta;
        lstText[4][0] = txtEYBIgual;
        lstText[4][1] = txtEYBDesde;
        lstText[4][2] = txtEYBHasta;
        lstText[5][0] = txtNuXYIgual;
        lstText[5][1] = txtNuXYDesde;
        lstText[5][2] = txtNuXYHasta;
        lstText[6][0] = txtNuYXIgual;
        lstText[6][1] = txtNuYXDesde;
        lstText[6][2] = txtNuYXHasta;
        lstText[7][0] = txtAlphaXIgual;
        lstText[7][1] = txtAlphaXDesde;
        lstText[7][2] = txtAlphaXHasta;
        lstText[8][0] = txtAlphaYIgual;
        lstText[8][1] = txtAlphaYDesde;
        lstText[8][2] = txtAlphaYHasta;
        lstText[9][0] = txtAlphaXYIgual;
        lstText[9][1] = txtAlphaXYDesde;
        lstText[9][2] = txtAlphaXYHasta;
        lstText[10][0] = txtBetaXIgual;
        lstText[10][1] = txtBetaXDesde;
        lstText[10][2] = txtBetaXHasta;
        lstText[11][0] = txtBetaYIgual;
        lstText[11][1] = txtBetaYDesde;
        lstText[11][2] = txtBetaYHasta;
        lstText[12][0] = txtBetaXYIgual;
        lstText[12][1] = txtBetaXYDesde;
        lstText[12][2] = txtBetaXYHasta;

        pnlVariables->Width = scrBox->Width - 25;
        pnlVariables->Constraints->MinWidth = scrBox->Width - 25;

        lstMaterial = new TList();
}
```

ANEXO C. CÓDIGO DEL ALGORITMO GENÉTICO

```
. //---------------------------------------------------------------------------

#pragma hdrstop

#include "UAlgoritmoGenetico.h"
#include <Classes.hpp>
#include <cstdlib>
#include <cmath>
#include <limits>
#include <math.h>
#include <cmath>
#include "UMezcla.h"
#include "UApilado.h"
#include "FormRecursos.h"

#include <vcl.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)

//--------------calculo de rigideces-----------------------------------------
void captura_datos(double ELC,double ETC,double GLNC,double GLTC,double
GTNC,double NULTC,double espe,double angu, int posi,int Fsabc);
void captura_alpha(int posi,double alphalc,double alphatc,double alphaltc);
void captura_beta (int posi,double betalc,double betatc,double betaltc);
double *principal_rigideces(bool excel);
void asigna_memoria(int ncapas,int sime);

//CONSTRUCTOR
TAlgoritmoGenetico::TAlgoritmoGenetico()
{
      SIGMA = 0.2;
      MU = 0.5;
      nuevo = true;
      nBITSANGULO = 2;
      pasoAngulo = 90.0/127.0;
}
bool TAlgoritmoGenetico::empezar(int nPOP, float xRATE, float xCONV, float
xMUTA, int emparejar, TList *materiales)
{
      this->nPOP = nPOP;
      this->xRATE = xRATE;
      this->xCONV = xCONV;
      this->xMUTA = xMUTA;
      this->metodoEmparejar = emparejar;
      nMUTA = 1 / xMUTA;
      iterarMUTA = 0;
      matDisp = materiales;
      float lastCONV = 0, cONV = 100000;
      lastPROM = lastMAX = lastDesv = 0;
      actMAX = actPROM = actDesv = 0;
      float fAux;
```

```
	nVariables = 0;
	for (int i = 0; i < 53; i++) {
		if (FRecursos->restricciones[i][5] > 0)
			nVariables++;
	}
	nuevo = false;

	nGOOD = nPOP * xRATE;
	genPOP();
	generacion = 0;

	do {
		generacion++;
		lastPROM = actPROM;
		lastMAX = actMAX;
		lastDesv = actDesv;
		selParejas();
		cruce();
		actPROM = verificarConvergencia();
		if (actPROM <= 0.0)
			return false;
		if (actMAX <= 0.0)
			return false;
		convPROM = (float) FRecursos->abs(lastPROM - actPROM) / actPROM;
		convMAX = (float) FRecursos->abs(lastMAX - actMAX) / actMAX;
		//convDesv = (float) FRecursos->abs(lastDesv - actDesv) / actDesv;
	} while (convPROM > xCONV || convMAX > xCONV);
	return true;
}

int TAlgoritmoGenetico::decodificar(bool **datos, int index, int* &capas, int*
&ang)
{
	int posi = 0;
	int aux;
	bool signo;
	capas = new int[nMaxCapas];
	ang = new int[nMaxCapas];
	for (int i = 0; i < nMaxCapas; i++) {
		aux = 0;
		for (int j = nBitsCapa - 1; j >= 0; j--) {
			if (datos[index][posi++]) {
				aux += pow(2.0, j);
			}
		}
		capas[i] = aux * pasoCapa;
		aux = 0;
		//signo = datos[index][posi++];
		for (int j = nBITSANGULO - 1; j >= 0; j--) {
			if (datos[index][posi++]) {
				aux += pow(2.0, j);
			}
```

```
			}
			switch (aux) {
				case 0:
					aux = 0;
					break;
				case 1:
					aux = 45;
					break;
				case 2:
					aux = 90;
					break;
				case 3:
					aux = -45;
					break;
			default:
				;
			}
			//if (!signo)
			//	aux *= -1;
			ang[i] = aux;
		}

		return nMaxCapas;
}

double TAlgoritmoGenetico::acumularResultados(double *res)
{
		double resultado = 0;
		double suma = 0, media = 0;
		int cont = 0;
		switch (metodoEmparejar) {
			case SUMATORIA:
				resultado = 1;
				for (int i = 0; i < 53; i++) {
					if (FRecursos->restricciones[i][5] > 0) {
						cont++;
						media += res[i];
						resultado *= res[i];
					}
				}
				media /= cont;
				resultado /= media;
			break;
			case PROGRAMACION_META:
				resultado = 1;
				for (int i = 0; i < 53; i++) {
					if (FRecursos->restricciones[i][5] > 0) {
						if (resultado > 2)
							resultado /= 1000.0;
						resultado *= res[i];

					}
				}
```

```
			break;
			case OBTENCION_META:
				resultado = 1;
				for (int i = 0; i < 53; i++) {
					if (FRecursos->restricciones[i][5] > 0) {
						suma = 1000.0 * res[i] * FRecursos-
>restricciones[i][5];
						resultado *= suma;
						if (resultado > 2)
							resultado /= 1000.0;
					}
				}
			break;
			case LEXICOGRAFICO:

			break;
			default:
			;
		}
		return resultado;
}


void TAlgoritmoGenetico::ordenar(bool **datos, double *fitness, double
**resultado, bool rand)
{
		heapSort(datos, fitness, resultado);
}

void TAlgoritmoGenetico::genPOP()
{
		FRecursos->memoSalida->Clear();

		int iAux = matDisp->Count - 1;
		nBitsCapa = 1;
		while (iAux > 1) {
			iAux = iAux / 2;
			nBitsCapa++;
		}
		double a = 0;
		for (int i = 0; i < nBitsCapa; i++) {
			a += pow(2.0, i);
		}
		pasoCapa = (matDisp->Count - 1) / a;

		//Normalizar prioridades
		if (metodoEmparejar == OBTENCION_META) {
			iAux = 0;
			for (int i = 0; i < 53; i++) {
				iAux += FRecursos->restricciones[i][5];
			}
			for (int i = 0; i < 53; i++) {
				FRecursos->restricciones[i][5] /= iAux;
```

```
			}
		}

		nMaxBits = nMaxCapas * (nBitsCapa + nBITSANGULO);
		ranking = new double[nGOOD];
		fitInicial = new double[nPOP];
		pobInicial = new bool*[nPOP];
		resInicial = new double*[nPOP];
		int sumatoria = 0;
		for (int i = 1; i <= nPOP; i++) {
			sumatoria += i;
		}
		srand(time(NULL));
		String sAux;
		for (int i = 0; i < nPOP; i++) {
			sAux = "";
			pobInicial[i] = new bool[nMaxBits];
			resInicial[i] = new double[nVariables];
			for (int j = 0; j < nMaxBits; j++) {
				pobInicial[i][j] = (rand() % 2) == 1;
				sAux += pobInicial[i][j] ? "1" : "0";
			}

			fitInicial[i] = funcionCosto(pobInicial, i, resInicial);
		}
		ordenar(pobInicial, fitInicial, resInicial, true);
		pob = new bool*[nPOP];
		fit = new double[nPOP];
		res = new double*[nPOP];
		for (int i = 0; i < nPOP; i++) {
			pob[i] = new bool[nMaxBits];
			res[i] = new double[nVariables];
			for (int j = 0; j < nMaxBits; j++) {
				pob[i][j] = pobInicial[i][j];
			}
			fit[i] = fitInicial[i];
			for (int j = 0; j < nVariables; j++) {
				res[i][j] = resInicial[i][j];
			}
		}
}

//Crear una lista de parejas usando la selección por ranking.
void TAlgoritmoGenetico::selParejas()
{
		double dAux;
		parejas = new int*[nPOP/2];
		int max = nPOP / 2;
		int hamming;
		int intentos;
		double sumatoria = 0;

		for (int i = 0; i < nGOOD; i++) {
```

```
			sumatoria += fit[i];
		}

		for (int i = 0; i < nGOOD; i++) {
			ranking[i] = fit[i] / sumatoria;
		}

		for (int i = 1; i < nGOOD; i++) {
			ranking[i] = ranking[i - 1] + ranking[i];
		}

		for (int i = 0; i < max; i++) {
			intentos = 0;
			parejas[i] = new int[2];
			for (int j = 0; j < 2; j++) {
				dAux = ((double) rand() / (RAND_MAX));
				parejas[i][j] = -1;
				for (int k = 0; k < nGOOD; k++) {
					if (ranking[k] > dAux) {
						parejas[i][j] = k;
						k = nPOP;
					}
				}
				if (parejas[i][j] == -1) {
					parejas[i][j] = nPOP - 1;
				}

				if (j == 1) {
					hamming = 0;
					for (int k = 0; k < nMaxBits; k++) {
						if (pob[parejas[i][0]][k] != pob[parejas[i][1]][k])
							hamming++;
					}
					if (hamming <= (nMaxBits / 4)) {
						if (++intentos <= 4) {
							j++;
						}
					}
				}
			}
		}
	}
}

void TAlgoritmoGenetico::cruce()
{
	nuevaGen = new bool*[nPOP];
	fitNuevaGen = new double[nPOP];
	resNuevaGen = new double*[nPOP];
	int indexNuevaGen = 0;
	double dAux;
	int indexCambiar, indexIniciar, indexFinal;
	for (int i = 0; i < (nPOP / 2); i++) {
		nuevaGen[indexNuevaGen] = new bool[nMaxBits];
```

```
            nuevaGen[indexNuevaGen+1] = new bool[nMaxBits];
            resNuevaGen[indexNuevaGen] = new double[nVariables];
            resNuevaGen[indexNuevaGen+1] = new double[nVariables];
            for (int k = 0; k < nMaxCapas; k++) {
                  indexIniciar = k * (nBitsCapa + nBITSANGULO);
                  indexFinal = (k + 1) * (nBitsCapa + nBITSANGULO);
                  dAux = ((double) rand() / (RAND_MAX));

                  indexCambiar = FRecursos->redondear(dAux * (indexFinal -
indexIniciar) + indexIniciar);
                  for (int j = indexIniciar; j < indexCambiar; j++) {
                        nuevaGen[indexNuevaGen][j] = pob[parejas[i][0]][j];
                        iterarMUTA++;
                        if (iterarMUTA >= nMUTA) {
                              iterarMUTA = 0;
                              nuevaGen[indexNuevaGen][j] =
!nuevaGen[indexNuevaGen][j];
                        }
                        nuevaGen[indexNuevaGen+1][j] = pob[parejas[i][1]][j];
                        iterarMUTA++;
                        if (iterarMUTA >= nMUTA) {
                              iterarMUTA = 0;
                              nuevaGen[indexNuevaGen+1][j] =
!nuevaGen[indexNuevaGen+1][j];
                        }
                  }
                  for (int j = indexCambiar; j < indexFinal; j++) {
                        nuevaGen[indexNuevaGen][j] = pob[parejas[i][1]][j];
                        iterarMUTA++;
                        if (iterarMUTA >= nMUTA) {
                              iterarMUTA = 0;
                              nuevaGen[indexNuevaGen][j] =
!nuevaGen[indexNuevaGen][j];
                        }
                        nuevaGen[indexNuevaGen+1][j] = pob[parejas[i][0]][j];
                        iterarMUTA++;
                        if (iterarMUTA >= nMUTA) {
                              iterarMUTA = 0;
                              nuevaGen[indexNuevaGen+1][j] =
!nuevaGen[indexNuevaGen+1][j];
                        }
                  }
            }

            fitNuevaGen[indexNuevaGen] = funcionCosto(nuevaGen, indexNuevaGen,
resNuevaGen);
            fitNuevaGen[indexNuevaGen + 1] = funcionCosto(nuevaGen,
indexNuevaGen + 1, resNuevaGen);
            indexNuevaGen += 2;
      }
      ordenar(nuevaGen, fitNuevaGen, resNuevaGen, true);
      indexCambiar = 0;
      for (int i = nGOOD; i < nPOP; i++) {
```

```
			for (int j = 0; j < nMaxBits; j++) {
				pob[i][j] = nuevaGen[indexCambiar][j];
			}
			for (int j = 0; j < nVariables; j++) {
				res[i][j] = resNuevaGen[indexCambiar][j];
			}
			fit[i] = fitNuevaGen[indexCambiar++];
		}
		ordenar(pob, fit, res, true);
		for (int i = 0; i < nPOP; i++) {
			free(nuevaGen[i]);
			nuevaGen[i] = NULL;
			free(resNuevaGen[i]);
			resNuevaGen[i] = NULL;
			if (i < (nPOP / 2)) {
				free(parejas[i]);
				parejas[i] = NULL;
			}
		}
		free(nuevaGen);
		nuevaGen = NULL;
		free(fitNuevaGen);
		fitNuevaGen = NULL;
		free(resNuevaGen);
		resNuevaGen = NULL;
		free(parejas);
		parejas = NULL;
}

double TAlgoritmoGenetico::verificarConvergencia()
{
		double suma = 0;
		for (int i = 0; i < nPOP; i++) {
			suma += fit[i];
		}

		actMAX = fit[0];

		return suma / nPOP;
}

//Algoritmo de ordenamiento heapsort
void TAlgoritmoGenetico::heapSort(bool **datos, double *fitness, double
**resultado) {

	/* Inserción al heap */
		for (int heapsize=0; heapsize < nPOP; heapsize++) {
			int n = heapsize; // índice insertado
			while (n > 0) { // mientras no se alcance la raíz del heap
				int p = (n-1)/2; // índice del padre de n
				if (fitness[n] > fitness[p]) { // hijo es mayor que el padre
					cambiarArreglo(datos, fitness, resultado, n, p); //
cambiar hijo con padre
```

```
                        n = p; // revisar padre
                  }
                  else // padre es mayor al hijo
                        break; // el heap está correcto
            }
      }
      /* Remover del heap*/
      for (int heapsize=nPOP; heapsize>0;) {
            cambiarArreglo(datos, fitness, resultado, 0, --heapsize); //
reemplazar la raíz con el último elemento del heap
            int n = 0; // índice del elemento que se mueve del heap
            while (true) {
                  int left = (n*2)+1;
                  if (left >= heapsize) // el nodo no tiene hijo a la izquierda
                        break; // se alcanzó el fondo
                  int right = left+1;
                  if (right >= heapsize) { // el nodo tiene hijo a la izquierda
pero no a la derecha
                        if (fitness[left] > fitness[n]) // si el hijo de la
izquierda es mayor al nodo
                              cambiarArreglo(datos, fitness, resultado, left, n);
// cambiar hijo de la izquierda por el nodo
                        break; // está correcto
                  }
                  if (fitness[left] > fitness[n]) { // (left > n)
                        if (fitness[left] > fitness[right]) { // (left > right) &
(left > n)
                              cambiarArreglo(datos, fitness, resultado, left, n);
                              n = left; continue; // continuar recursividad con
el hijo de la izquierda
                        } else { // (right > left > n)
                              cambiarArreglo(datos, fitness, resultado, right,
n);
                              n = right; continue; // continuar recursividad con
hijo de la derecha
                        }
                  } else { // (n > left)
                        if (fitness[right] > fitness[n]) { // (right > n > left)
                              cambiarArreglo(datos, fitness, resultado, right,
n);
                              n = right; continue; // continuar recursividad con
el hijo de la derecha
                        } else { // (n > left) & (n > right)
                              break; // el nodo es mayor a ambos hijos; está bien
                        }
                  }
            }
      }
      //Invertir la lista.
      int n = nPOP / 2;
      for (int i = 0; i < n; i++) {
            cambiarArreglo(datos, fitness, resultado, i, nPOP - i - 1);
      }
```

```
}

void TAlgoritmoGenetico::cambiarArreglo(bool **datos, double *fitness, double
**resultado, int index1, int index2)
{
	bool bAux;
	double iAux;
	for (int i = 0; i < nMaxBits; i++) {
		bAux = datos[index1][i];
		datos[index1][i] = datos[index2][i];
		datos[index2][i] = bAux;
	}
	iAux = fitness[index1];
	fitness[index1] = fitness[index2];
	fitness[index2] = iAux;

	for (int i = 0; i < nVariables; i++) {
		iAux = resultado[index1][i];
		resultado[index1][i] = resultado[index2][i];
		resultado[index2][i] = iAux;
	}
}

void TAlgoritmoGenetico::escribirArchivo(bool **datos, double *fitness, double
**resultado, String nombre)
{
	TMemo *mem = FRecursos->memoSalida;
	mem->Lines->Clear();
	int *ang, *capas;

	int nCapas;
	String sAux;
	for (int i = 0; i < nPOP; i++) {
		sAux = "(";
		int nCapas = decodificar(datos, i, capas, ang);
		for (int j = 0; j < nCapas; j++) {
			sAux += IntToStr(capas[j] + 1) + "," + IntToStr(ang[j]) + ",";
		}
		sAux = sAux.SubString(0, sAux.Length() - 1);
		sAux += ") - ";
		for (int j = 0; j < nVariables; j++) {
			sAux += FloatToStr(resultado[i][j]) + ",";
		}
		sAux = sAux.SubString(0, sAux.Length() - 1);
		mem->Lines->Add(sAux);
	}
	mem->Lines->SaveToFile(FRecursos->rutaSistema + "Resultados\\" + nombre +
".txt");
}

bool **TAlgoritmoGenetico::getPobInicial() {return pobInicial;}

double *TAlgoritmoGenetico::getFitInicial() {return fitInicial;}
```

```
bool **TAlgoritmoGenetico::getPobFinal() {return pob;}

double *TAlgoritmoGenetico::getFitFinal() {return fit;}

int TAlgoritmoGenetico::getNPOP() {return nPOP;}
double **TAlgoritmoGenetico::getResInicial() {return resInicial;}
double **TAlgoritmoGenetico::getResFinal() {return res;}
TList *TAlgoritmoGenetico::getListaMateriales() {return matDisp;}
String TAlgoritmoGenetico::getCadena(bool **datos, int index)
{
	String str = "";

	for (int i = 0; i < nMaxBits; i++) {
		if (datos[index][i])
			str += "1";
		else
			str += "0";
	}

	return str;
}
int TAlgoritmoGenetico::getNGeneraciones() {return generacion;}
void TAlgoritmoGenetico::limpiarReferencias()
{
	for (int i = 0; i < nPOP; i++) {
		free(pobInicial[i]);
		pobInicial[i] = NULL;
		free(pob[i]);
		pob[i] = NULL;
		free(resInicial[i]);
		resInicial[i] = NULL;
		free(res[i]);
		res[i] = NULL;
	}
	free(pobInicial);
	pobInicial = NULL;
	free(pob);
	pob = NULL;
	free(nuevaGen);
	nuevaGen = NULL;

	free(ranking);
	ranking = NULL;
	free(fitInicial);
	fitInicial = NULL;
	free(fit);
	fit = NULL;
	free(res);
	res = NULL;
	free(resInicial);
	resInicial = NULL;
}
```