PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Escuela de Posgrado

Automatización de la optimización del diseño de elementos mecánicos mediante algoritmo genético aplicando ingeniería del conocimiento

Tesis para obtener el grado académico de Magíster en Gestión de la Ingeniería que presenta:

Linder Oskar Jesús Barboza Usco

Asesor:

Héctor Andrés Melgar Sasieta

Lima, 2022

DEDICATORIA

A mis padres, Óscar y Rocío, que con la sabiduría propia de cada uno me guiaron por el camino de la educación y el aprendizaje continuo: la mejor herencia.

A Asuntita, maravillosa consejera, quien me hizo entender la prioridad de la educación por encima de otras opciones.

RESUMEN

El objetivo del presente estudio es desarrollar una herramienta que permita agilizar y generalizar el proceso de diseño mecánico de un componente específico teniendo en consideración que la etapa de diseño es una de las más importantes dentro del proceso productivo de una pieza, pues es en donde se pueden generar los mayores ahorros económicos a través de las cualidades del producto (ergonomía, peso, volumen, calidad, etc.). En ese sentido, encontrar una manera de desarrollar dicho proceso de forma personalizada y con la capacidad de adaptarlo a las condiciones de trabajo de la empresa que busca utilizarlo, mejorará su desempeño.

Para poder lograr implementar esta herramienta se tuvo que vincular tres conceptos: el diseño mecánico propiamente dicho, que son las definiciones técnicas, fórmulas paramétricas y criterios mecánicos que se utilizan al momento de diseñar un elemento mecánico; la ingeniería del conocimiento, que es la rama de la ingeniería que nos dará los conceptos básicos de cómo extraer la información plasmada dentro de un proceso y trasladarla a un flujo de trabajo; y finalmente, los algoritmos bio inspirados, específicamente, el algoritmo genético, que es el que optimizará el proceso de diseño tomando como base los datos de entrada que se captarán previamente.

Palabras clave: diseño, algoritmo genético, ingeniería del conocimiento, elemento mecánico

ABSTRACT

The objective of this study is to develop a tool that allows to streamline and generalize the mechanical design process of a specific component, taking into consideration that the design stage is one of the most important within the production process of a piece, because that is where they can be generated the greatest economic savings through the qualities of the product (ergonomics, weight, volume, quality, etc.). In that sense, finding a way to develop this process in a personalized way and with the ability to adapt it to the working conditions of the company that seeks to use it, will improve its performance.

In order to be able to implement this tool, three concepts had to be linked: the mechanical design itself, which are the technical definitions, parametric formulas and mechanical criteria that are used when designing a mechanical element; knowledge engineering, which is the branch of engineering that will give us the basic concepts of how to extract the information embodied within a process and transfer it to a workflow; and finally, the bio-inspired algorithms, specifically, the genetic algorithm, which is the one that will optimize the design process based on the input data that will be previously captured.

Keywords: design, genetic algorithm, KBE, mechanical component

ÍNDICE

Lista de Ilustraciones

Lista de Tablas

Capítulo 1 GENERALIDADES

1.1 Problemática

El diseño mecánico es un proceso iterativo de creatividad aplicada donde el diseñador debe pensar en su desarrollo teniendo en cuenta los requerimientos y problemas a resolver. En ese sentido el proceso de diseño como tal consiste en idear un plan para resolver un problema o cubrir una necesidad específica. Por ende, de concretarse este plan, el resultado deberá ser funcional, competitivo, útil y confiable, capaz de poder comercializarse o fabricarse de ser necesario (Budynas, s. f.).

La finalidad de un diseño mecánico es de encontrar las dimensiones y la forma de las piezas (o elementos mecánicos) así como seleccionar los materiales de trabajo (Norton, s. f.). Para llegar al diseño final, se debe empezar desde lo más básico, como por ejemplo conociendo el movimiento de los elementos y las fuerzas externas a las cuales el sistema es sometido. Con estos datos es posible realizar un primero bosquejo de los elementos; sin embargo, a este diseño inicial se deberá adicionar más detalles que harán que su diseño sea más preciso. De ahí que se dice que el diseño es iterativo puesto que no trabaja de manera lineal sino más bien en un bucle de desarrollo en donde se harán constantes rediseños hasta encontrar los parámetros óptimos (Norton, s. f.).

Además, el rediseño debe de ser trabajado mediante un experto técnico, el cual debe tener conocimiento en el diseño de piezas mecánicas, y sobre todo la experiencia para poder usar las técnicas de diseño de manera correcta y con las herramientas adecuadas. Por ejemplo, si se desea diseñar un elemento de unión mediante un software de simulación, se debe conocer que estos trabajan mediante cálculo por elementos finitos (FEM por sus siglas en inglés) que es un método para calcular la solución numérica de un conjunto de ecuaciones (Whiteley, 2017); por ende, el diseñador deberá estar en la capacidad de aplicar las sujeciones, cargas y restricciones a dicha simulación; para una vez que lo haya realizado, proceder con las mejoras del producto. Esto se resume, en que para realizar lo antes mencionado, el experto técnico deberá (Mott, s. f.) :

- Tener conocimiento en diseño mecánico.
- Usar software de diseño o manejar plantillas de cálculo.
- Saber las restricción y requerimientos iniciales para los que fue fabricado inicialmente la pieza.
- Entender los nuevos requerimientos funcionales y trasladarlos a conceptos técnicos.

Sin embargo, el uso de herramientas computacionales puede tomar tiempo en trabajarse, esto debido a que la simulación de la pieza en CAD tarda varios minutos y dependerá mucho de la complejidad de la geometría, cargas y restricciones que se apliquen. Si bien es cierto, el uso de análisis por FEM es el más común, existen otras formas como el uso de programación en el lenguaje de mayor dominio del diseñador, así como la herramienta de hoja de cálculo que también es una de las más frecuentes (Gembarski, 2020). Para escoger entre una de estas, es necesario evaluar el esfuerzo del modelado, la capacidad de la herramienta (por ejemplo, es más complicado trabajar un cálculo por elementos finitos desde una hoja de cálculo que desde un software desarrollado exclusivamente para eso) y las competencias del diseñador; teniendo en cuenta que ninguno de estos métodos es superior a los otros (Gembarski, 2020).

Por otro lado, de la misma manera que el proceso de diseño, la optimización del mismo es un proceso que se puede obtener de diferentes maneras, según la complejidad de la búsqueda de la optimización de la pieza. La que usualmente es aplicable es la de ensayo y error (Lampinen, 2003); es decir, generar varios diseños de una misma pieza y evaluar cuán eficientes son unos respecto a los otros para así poder determinar, solo entre los diseños evaluados, el óptimo. No obstante, actualmente el uso de algoritmos computacionales permite una solución que no solo abarca una determinada cantidad de diseños de una pieza, sino que con la ayuda de una computadora es posible evaluar gran mayoría de las combinaciones de los diferentes parámetros que determinan un diseño diferente de una misma pieza. Esto conllevaría al desarrollo de un algoritmo que revise todas las posibles combinaciones de parámetros y posteriormente evalúe el comportamiento de cada diseño sujeto a restricciones que el propio diseñador determine para conseguir el estado óptimo; sin embargo, este trabajo, dependiendo del tamaño del grupo de combinaciones, sería ineficiente. Para

esto el diseñador se puede apoyar en los algoritmos bio-inspirados, los cuales son algoritmos metaheurísticos inteligentes que tratan de replicar la manera en que los organismos biológicos operan para conseguir una respuesta óptima de forma más intuitiva (Kar, 2016). Para el caso, un algoritmo genético, que es un tipo de algoritmo bio-inspirado, se basa en la teoría de la evolución para ayudar en la optimización de la búsqueda de los posibles diseños existentes que cumplan con los requerimientos básicos solicitados y los clasifique de acuerdo a los criterios de optimización.

Finalmente, en base a lo mencionado anteriormente, se revisará la solución al problema del excesivo tiempo de trabajo para el diseño, así como las consideraciones técnicas al momento de diseñar un elemento mecánico; teniendo el conocimiento técnico extraído como un método para un caso general de diseño de una pieza mecánica, además de la optimización del diseño usando un algoritmo genético el cual optimizará el tiempo de búsqueda del diseño óptimo.

1.2 Objetivos de la investigación

1.2.1 Objetivo General

Implementar una herramienta que permita encontrar los parámetros óptimos de manera automática utilizando el conocimiento que existe para el diseño de productos mecánicos usando una única herramienta de software.

1.2.2 Objetivos Específicos

- Implementar un mecanismo que permita representar el conocimiento que existe en el diseño de productos mecánicos.
- Definir los parámetros básicos necesarios para la implementación de un algoritmo genético de diseño de piezas mecánicas.
- Desarrollar un mecanismo capaz de generalizar el diseño de piezas mecánicas mediante un algoritmo genético.

1.3 Resultados esperados

- Para el OE1:
  - Tipos de Conocimiento en el diseño mecánico

- Reglas de diseño aplicado a los componentes mecánicos
- La generalización de las características de cualquier elemento mecánico.

- Para el OE2:
  - El diseño de un cromosoma que modele el comportamiento de un componente mecánico.
  - Una función de fitness general (fitness).
  - Definición del criterio que llevará a la convergencia del algoritmo en un conjunto solución.
- Para el OE3:
  - La adaptación del flujo general al algoritmo genético.

1.4 Métodos y Procedimientos

Para el Objetivo Específico 1, se revisarán diferentes procesos de diseño y optimización y se encontrarán patrones de comportamiento similares con el fin de obtener dicho flujo generalizado. Además, se tomará de base la metodología KBE (*Knowledge-based Engineering*) para la recopilación de los tipos de conocimiento a extraer y las reglas de diseño.

También, para el Objetivo Específico 2, se usarán como bases del algoritmo genético los datos obtenidos en el capítulo 4. Con esto se creará un cromosoma que representará los componentes principales de un elemento mecánico. Además, se deberá definir los criterios de selección de cromosomas a partir de los requerimientos planteados para el elemento. Esto se hará mediante la creación de funciones fitness, que también deberán ser definidas y generalizadas. Luego, se definirá el criterio de convergencia del algoritmo a través de los erros máximos permitidos por cada requerimiento.

Finalmente, Objetivo Específico 3, con el conocimiento extraído en el capítulo 4 y plasmado como base en el capítulo 5, se procede a desarrollar la unión de ambos. Para esto se trabajará un diagrama de flujo que servirá de base para la generalización del código empleado dentro del algoritmo genético.

1.5 Alcances y Limitaciones

El presente trabajo se encuentra en el marco del desarrollo del diseño específicamente de elementos mecánicos; es decir, aquellos en donde el material y los esfuerzos relacionados a este sean determinantes en el comportamiento esperado del componente. El diseño se realizará en función a los requerimientos técnicos que influyen directamente al elemento. No se describirá cómo encontrar esos requerimientos a través de factores externos que impacten en el elemento.

Por otro lado, según los resultados esperados, se desarrollará un algoritmo genético como modelo de optimización que tomará de datos de entrada parámetros generales para diseñar un elemento mecánico. De la misma manera, el modelo de flujo de trabajo planteado, sus etapas y la metodología que se obtendrá están orientados solo al diseño mecánico.

La suma de todos estos puntos engloba el valor de esta investigación que busca generalizar un método de optimización y vincular el desarrollo de algoritmos con el diseño mecánico.

Capítulo 2 MARCO CONCEPTUAL

En este capítulo se presenta el marco conceptual que busca detallar los conceptos de diseño mecánico, extracción del conocimiento y algoritmos genéticos que son necesarios para describir el problema de optimización de diseño de un elemento mecánico. En ese sentido, lo que se busca es explicar cómo cualquier diseño de un elemento mecánico debe pasar por un proceso de diseño utilizando las herramientas que este nos brinda y el conocimiento técnico inmerso, para así extraerlo y plasmarlo en un algoritmo genético definiendo dentro de este los genotipos y la función fitness a desarrollar para obtener un diseño optimizado.

2.1 Diseño mecánico

El diseño mecánico es la aplicación de varias técnicas y principios científicos que permiten definir un elemento o sistema a tal nivel de detalle que permita su realización (Norton, s. f.).

2.1.1 Proceso de diseño

Es el conjunto de actividades ordenadas que permiten el diseño mecánico. Como proceso, este debe ser innovador y altamente iterativo. Dicho proceso comienza con la identificación de necesidades y la determinación para hacer algo al respecto; para después de muchas iteraciones, terminar en un plan para satisfacer dichas necesidades (Budynas, s. f.)

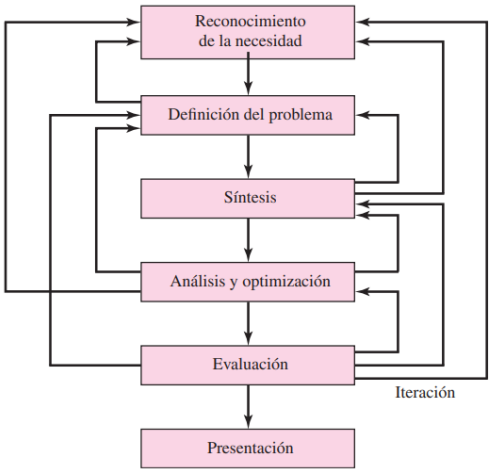

Ilustración 1. Flujo de proceso de diseño. Budynas, s.f.

2.1.2 Herramientas de diseño

Actualmente, el ingeniero de diseño tiene gran cantidad de herramientas que sirven como recursos para su labor de diseño. Desde libros y revistas técnicas de ciencia e ingeniería, catálogos técnicos de proveedores de fabricantes; hasta las computadoras que tienen gran capacidad de diseñar, analizar y simular componentes mecánicos (Budynas, s. f.).

Una herramienta muy utilizada en la actualidad, son las herramientas de diseño asistido por computador (CAD). Estos son capaces de poder crear un elemento tridimensional (como una pieza mecánica) dentro de la computadora y poder extraer de él diferentes propiedades tales como peso, volumen, posición del centro de gravedad.

2.1.3 Optimización de forma

Lo que en este documento se describirá como “optimización”, en el rubro de diseño mecánico es conocido como “optimización de forma”, el cual es un campo relativamente joven y un con un gran potencial (Lampinen, 2003). Lo que busca la optimización de forma es encontrar una estructura que minimice una funcionalidad dada, propia de la geometría del cuerpo (ejemplo: peso de la pieza) y que aun así satisfaga los requerimientos funcionales necesarios (Sokolowski & Zolesio, 1992).

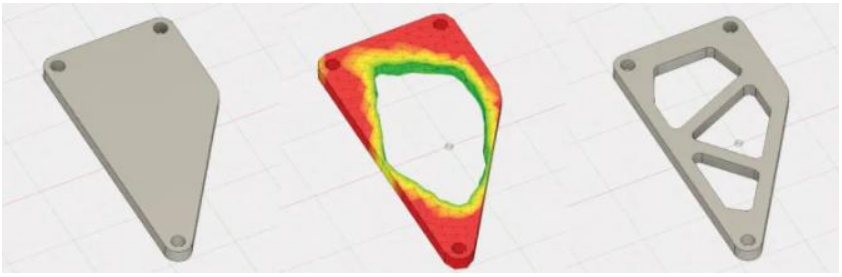

Ilustración 2. Optimización de forma. Imagen tomada de www.moebius-factory.com

2.2 Extracción del conocimiento técnico

2.2.1 Sistema de ingeniería basado en el conocimiento (KBE)

El sistema de ingeniería basado (KBE) está relacionada directamente al conocimiento y por ende está relacionado con la ingeniería del conocimiento y gestión del conocimiento con las que se intersecta, complementa y especializa (Wheeler, s. f.).

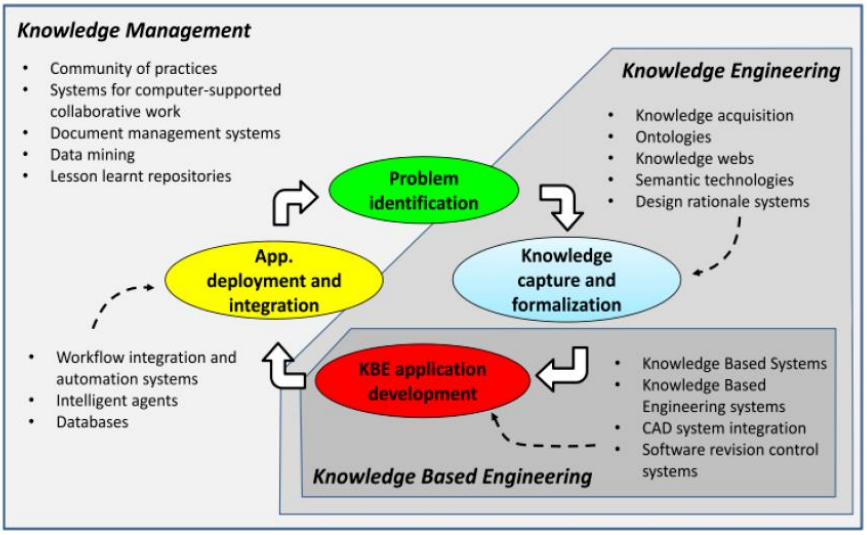

Ilustración 3. Relación entre KBE, Ingeniería y Gestión del Conocimiento. Wheeler, s.f.

Del punto de vista conceptual, KBE es un método de ingeniería que representa una combinación de programación, técnicas de inteligencia artificial y tecnologías CAD dando como resultado una personalización y/o automatización en el diseño. El KBE tiene como fin último capturar las buenas prácticas y experiencia para incorporarlas en una base de conocimiento (Chapman & Pinfold, 1999).

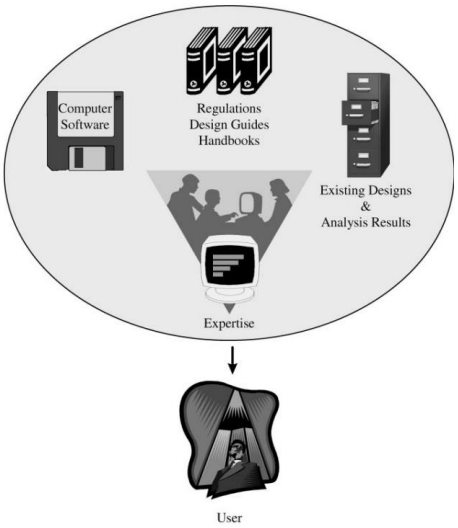

Ilustración 4. Sistema KBE. Chapman & Pinfold, 1999

2.2.2 Habilidades básicas necesarias como diseñador mecánico

Cualquier diseñador debe tener capacidad para crear e innovar; sin embargo, un diseñador mecánico también debe tener una amplia variedad habilidades. Para esto se identifican dos tipos: técnicas y blandas.

Con respecto a las habilidades técnicas, el diseñador debe tener nociones en:

- Trazado y dibujo técnico.
- Uso de herramientas asistidas por computador.
- Propiedades y procesamiento de los materiales.
- Resistencia y dinámica de materiales
- Mecánica de fluidos y transferencia de calor y masa.
- Máquinas hidráulicas, neumáticas y eléctricas.
- Control y automatización industrial.
- Análisis de esfuerzos dentro de los materiales.
- Conocimiento especializado del comportamiento de elementos de máquinas

Por otro lado, respecto a las habilidades blandas:

- Comunicación oral y redacción.
- Trabajo en equipo.
- Creatividad e innovación.
- Solución de problemas.

Dentro de todos estos conocimientos, existe uno que vincula a ambos tipo de habilidades: el conocimiento de los requerimientos y expectativas del cliente (Mott, s. f.). Este último es fundamental para el traslado de los requerimientos comerciales, donde el cliente expresa las necesidades y funcionalidades que necesita, a los requerimientos técnicos, donde el diseñador debe entender y traducir dichos requerimientos comerciales a conceptos mecánicos.

2.2.3 Parámetros a considerar para el diseño

El proceso de diseño necesita de parámetros de entrada para obtener un diseño elaborado, ellos son:

- Requisitos: Datos que el cliente ofrece al diseñador y que la máquina deberá tener.
- Funciones: Describen la finalidad de la máquina de manera general. Por ejemplo: Proteger conexión eléctrica, evitar la corrosión, etc.
- Condiciones de diseño: Estas son cuantitativas y son los valores esperados en los que debería de trabajar la máquina diseñada. Por ejemplo, volumen ocupado, velocidad, rigidez, etc.
- Diseños previamente elaborados: Estos son complementarios, pero en caso se tengan, sirven como base para los nuevos desarrollos en los que se esté trabajando.

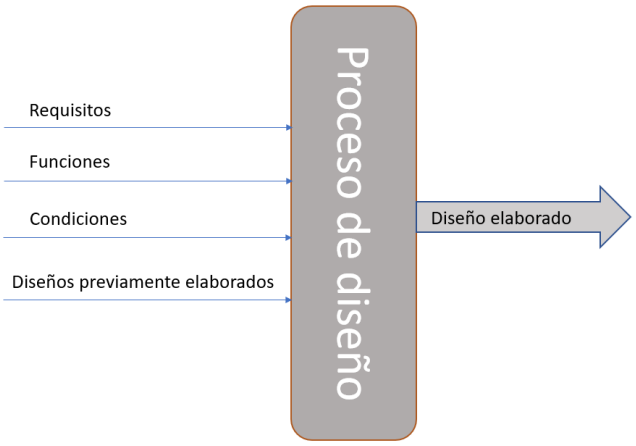

Ilustración 5.Entradas y salidas para el proceso de diseño. Elaboración propia

2.3 Programación genética

La programación genética tiene sus raíces en el año 1948 cuando Alan Turing expresó por primera vez la definición. El indicaba que existe la búsqueda genética o evolutiva mediante la cual una combinación de genes es buscado,

siendo el criterio el valor de supervivencia (Koza & Poli, 2005). Esta definición fue madurando inclusive el mismo autor, dos años después, en 1950, indicaba que no podía esperar una excelente máquina en el primer intento, por ello experimentó con varias verificando si eran mejores o empeoraban; en ese sentido, encontró una conexión entre este proceso y el proceso evolutivo, pues una máquina “hijo” se relaciona con el material hereditario, si se altera uno de sus parámetros para obtener una nueva máquina, se relaciona con el proceso de mutación, dando así al proceso de selección natural (Koza & Poli, 2005).

2.4 Algoritmo genético

En 1960, John Holland con sus estudiantes de la universidad de Michigan desarrollaron el concepto inicial de algoritmo genético. En contraste con la programación genética, la idea inicial de este no era desarrollar un algoritmo para resolver problemas específicos, sino más bien estudiar el fenómeno de adaptación en la naturaleza para así encontrar la manera de adaptar este mecanismo en un sistema computacional (Melanie, s. f.).

Ya en el 2000, se definió como un método que no dependía del dominio inicial para la creación de sus genes y que generaba una población de programas para resolver un problema. Específicamente, los algoritmos genéticos transforman iterativamente una población de programas de computadora en una nueva generación de programas mediante la aplicación de analogías de la naturaleza usando operaciones genéticas. Estas operaciones son las de reproducción, recombinación y mutación, además de otras secundarias como lo son la duplicación y eliminación de genes (Koza & Poli, 2005).

Actualmente, la definición de algoritmo genético es más específica: un algoritmo de búsqueda basados en la teoría de evolución de la naturaleza haciendo uso del proceso natural de selección, reproducción y mutación; esto lo lleva a encontrar soluciones de alta calidad para problemas de búsqueda, optimización y aprendizaje (Wirsansky, 2020).

Entonces, este algoritmo al basarse en la teoría de la evolución sirve como método para moverse de una población de genotipos (o llamados también “cromosomas”) a una nueva población utilizando el proceso de selección natural

a través de sus operadores genéticos: selección, recombinación y mutación (Melanie, s. f.). En la ilustración 5, se observa el flujo básico de un algoritmo genético; en él se aprecia la aplicación de los operadores genéticos, la evaluación mediante la función fitness y posterior evaluación entrando en un bucle hasta que se consiga los valores óptimos.

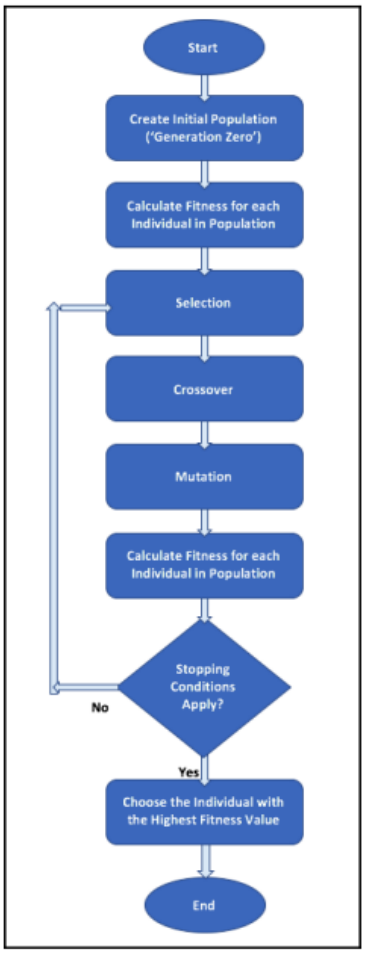

Ilustración 6. Flujo básico de algoritmo genético. Wirsansky, E. (2020)

2.4.1 Genotipo

Un genotipo o individuo es un conjunto de genes agrupados. Este contiene características físicas y mentales (Melanie, s. f.). Del lado de la programación, cada individuo es representado por un cromosoma que a su vez representa un conjunto de genes (Wirsansky, 2020).

Para el caso del diseño de un resorte helicoidal de sección circular, un genotipo sería un modelo que contiene sus parámetros dimensionales, siendo sus genes el diámetro de alambre, diámetro externo, número de vueltas efectivas, número de vueltas totales y el módulo de corte relacionado al material a usar.

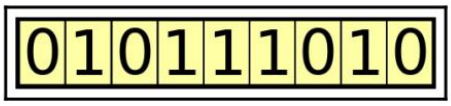

Ilustración 7. Genotipo en código binario. Wirsansky, 2020.

En la ilustración 7 se ve cómo un genotipo también puede ser expresado como un vector binario, para este caso cada conjunto de bits representa una dimensión del genotipo.

2.4.2 Población

Para cualquier instante de la ejecución del algoritmo, existe siempre una población de individuos, los cuales son candidatos a ser seleccionados a pasar a la próxima etapa. Así la población continuará evolucionando teniendo dentro de ella mejores individuos, que serán reemplazados uno por uno (Wirsansky, 2020).

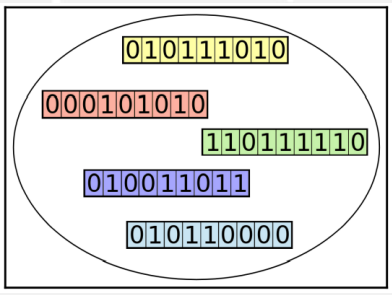

Ilustración 8.Población de individuos representado por genotipos binarios. Wirsansky, 2020

2.4.3 Función fitness

Del lado biológico, el fitness es la probabilidad de que un organismo pueda vivir para reproducirse o una función del número de organismos descendientes que este posee (Melanie, s. f.). Para los algoritmos genéticos, la definición es similar teniendo como fitness a una ecuación que evaluará cuán probable es que un individuo se mantenga a lo largo de las generaciones de poblaciones que se cree. Es un indicador que determinará su existencia frente a otros individuos.

Siguiendo el caso del resorte helicoidal de sección circular, la función fitness que se desarrolló deberá considerar la aproximación del requerimiento que el ensamble necesite ya sea de fuerza, deformación y peso de la pieza. Estos parámetros deberán combinarse para obtener una sola función que los contemple.

2.4.4 Operadores genéticos

La selección de los operadores genéticos dependerá del tipo de algoritmo que se va a desarrollar y de la estrategia de decodificado de los individuos.

Selección

Es el primer paso del proceso evolutivo y consiste en la selección para la reproducción, con cierto grado de aleatoriedad, en donde puede predominar los individuos de mayor fitness (Sivanandam & Deepa, 2007).

Recombinación (*Crossover*)

Cuando se genera la reproducción, lo que sucede con los dos individuos base es dar pase a la creación de otros intercambiando parte de sus cromosomas siendo estos nuevos parte de su descendencia (Wirsansky, 2020).

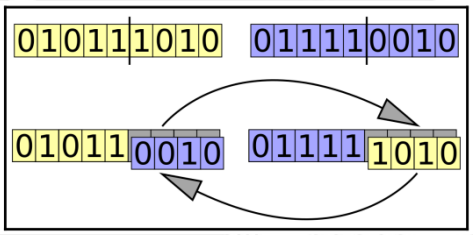

Ilustración 9. Operación de recombinación entre dos genotipos binarios. Wirsansky, 2020

Mutación

La mutación es el proceso por el que, dentro de un cromosoma, un gen es alterado de manera aleatoria para producir un nuevo individuo siendo aleatoria también la posibilidad de la mutación puntualmente (Holland, 1992).

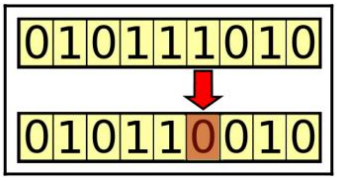

Ilustración 10. Operación de mutación aplicado a un genotipo binario. Wirsansky, 2020

Capítulo 3 ESTADO DEL ARTE

3.1 Introducción

El presente estado del arte se realizó utilizando una revisión sistemática de la información disponible en la web.

La finalidad de la presente revisión es de tener referencias de los objetivos específicos de esta investigación; en ese sentido, los estudios mostrados son antecedentes para el diseño mecánico usando algoritmos computacionales y el desarrollo de un flujo de diseño utilizando ingeniería del conocimiento.

3.2 Método usado para la revisión del estado del arte

El método empleado contempla las siguientes consideraciones:

- Se usaron dos preguntas de investigación: ¿Cómo se han aplicado los algoritmos computacionales para optimizar el diseño de elementos mecánicos? ¿Cómo se ha podido extraer el conocimiento implícito dentro del proceso de diseño?
- Los términos usados para la búsqueda incluían:
  - "Diseño mecánico" O "Optimización" O "algoritmos computacionales"
  - ("KBE" Y "Proceso de diseño") O ("KBE" Y "Componentes mecánicos")
- El proceso de búsqueda de estudios se hizo de manera iterativa.
- Los criterios de inclusión fueron revisando el resumen, la introducción, las *keywords* y las conclusiones de cada estudio.

3.3 ESTUDIO N°1: Optimización de forma de una leva mediante algoritmos genéticos

Este proyecto tiene como objetivo desarrollar el diseño óptimo de una leva mecánica montada en la válvula de admisión de un motor de combustión interna mediante la optimización de forma usando un algoritmo genético.

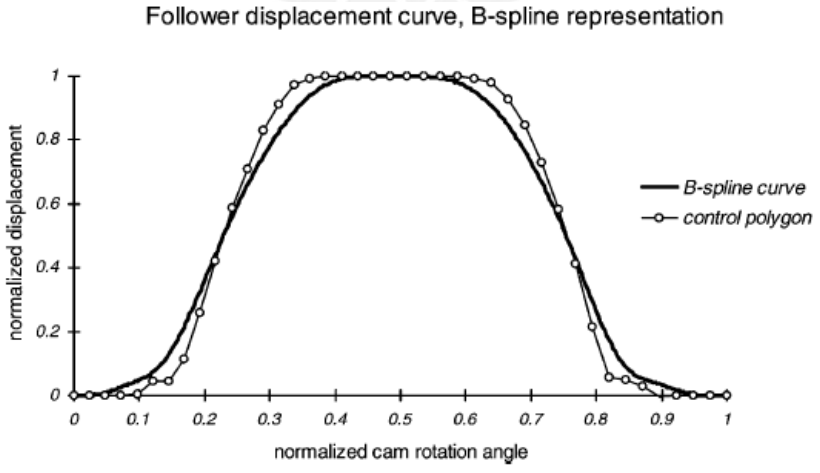

Ilustración 11. Curva de desplazamiento de seguidor de leva. Lampinen, 2003.

Para esto lo que se hace es tratar de replicar la curva dentro de la gráfica ángulo versus desplazamiento normalizado. En ese sentido la curva también se puede representar como un conjunto de puntos (40 puntos para el caso) distribuidos en el plano. Para el caso, trabajar un algoritmo genético es tomar como un individuo dentro de una población como un conjunto de 40 puntos; por ende, cada punto se definiría como un gen de dicho individuo.

Es así, que para poder determinar cuál individuo sobreviviría por encima de los otros, se tuvo que definir una función objetivo, que para el caso se buscó analizar la cinemática de los puntos de la leva al momento del giro, tales como desplazamiento, velocidad, aceleración, frecuencia y amplitud de los puntos, entre otros.

De este estudio se confirma que el uso de un algoritmo genético es más eficiente que el método de ensayo y error cuando de diseñar un elemento se trata. Además, una conclusión propia del estudio es que la aproximación evolutiva no requiere de un ingeniero de diseño que genere y evalúe cada probable diseño. (Lampinen, 2003)

3.4 ESTUDIO N°2: Programación multiobjetivo usando diseño uniforme y algoritmos genéticos

En este estudio el autor toma como base el concepto de optimización de Pareto, el cual describe el comportamiento de una función de optimización compuesta por n subtérminos, conforme se quiera optimizar uno de los términos, los otros perderán eficiencia. Para el caso de solo dos variables, esta descripción se entiende como la curva que delimita el conjunto de valores óptimos que la función puede alcanzar.

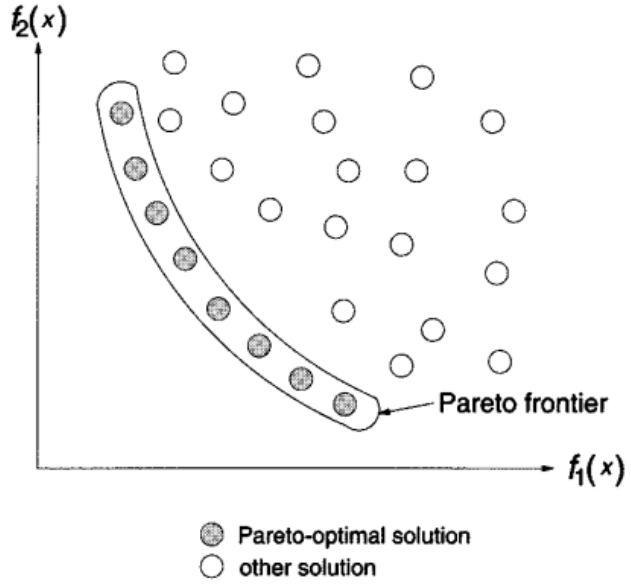

Ilustración 12. Solución de frontera de Pareto de un problema de programación de dos objetivos. Yuping Wang & Yiu-Wing Leung, 2000.

En la gráfica mostrada se tiene una curva de Pareto delimitando los valores mínimos que una función compuesta por $f_1(x)$ y $f_2(x)$ puede tomar. Se deduce que si se desea minimizar f_1, entonces f_2 aumentará y viceversa.

Para el caso de la programación multiobjetivo, se tienen n funciones como parte de una sola función objetivo. Por otro lado, si se desea hablar de una optimización usando algoritmos genéticos, entonces la función objetivo se convierte en función fitness y esta quedaría conformada por las funciones a optimizar internas multiplicadas cada uno por un peso asignado (además que estas funciones deben ser normalizadas para obtener resultados coherentes).

Luego, el diseño uniforme permite distribuir todos los puntos de la población inicial dentro de todo el rango de alcance de las variables, esto para asegurar la obtención de un máximo/mínimo absoluto y no relativo. También, la función fitness es un vector que sin los pesos relativos no tiene una dirección, por lo que es necesario estimar las probables direcciones que este puede tomar para después analizar cuál es la más adecuada.

El uso de la programación multiobjetivo con diseño uniforme sirve para encontrar todas las posibles combinaciones tanto de pesos relativos y valores de subfunciones. Como se entiende, existen demasiadas combinaciones posibles, por lo que el uso de un algoritmo genético para encontrar los valores óptimos

(curva de Pareto) disminuye considerablemente la cantidad de iteraciones que se tienen que hacer (Yuping Wang & Yiu-Wing Leung, 2000).

3.5 ESTUDIO N°3: Ingeniería de diseño, una necesidad de repensar una solución usando ingeniería basado en el conocimiento

En este caso, el documento no se limita a hablar sobre el diseño en sí mismo, sino que describe el flujo completo de creación de un producto y como el proceso de diseño afecta a todo ese flujo, describiendo que los diseñadores deberían tener una idea completa de cómo se fabricará el producto (manufactura), el costo que tendrá, y otros procesos correspondientes al ciclo de vida del producto; de esta manera, todos esos criterios deben ser considerados al momento del diseño. Por encima de todo, el criterio fundamental es la comunicación entre áreas para intercambiar el conocimiento que cada una posee y poder detectar futuros problemas antes de que estos sucedan.

Entonces, si es necesario tener en cuenta el conocimiento de diferentes áreas para el desarrollo del diseño, este se deberá apoyar en una herramienta capaz de tomar en consideración todos esos criterios. Es por ello, que se creó una herramienta de respuesta de análisis de diseño (DART) usando un sistema de ingeniería basado en el conocimiento (KBE).

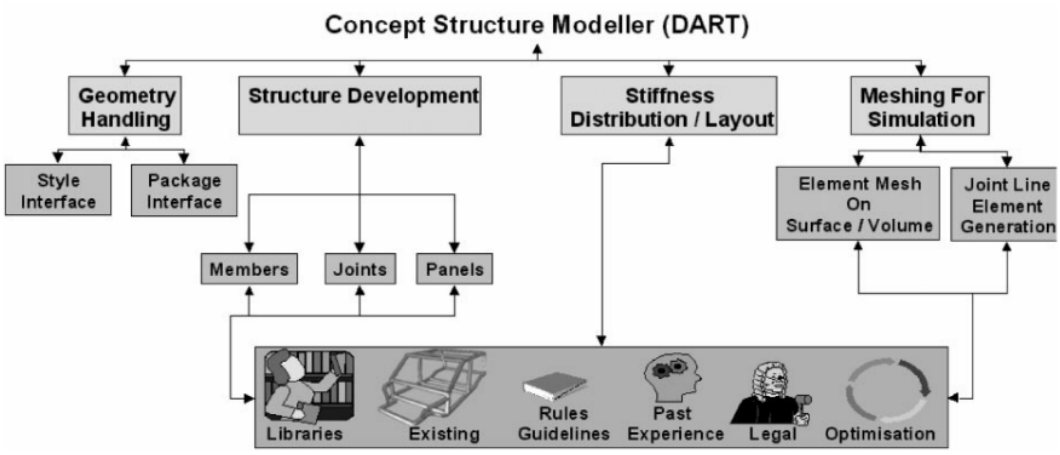

Ilustración 13. Esquema inicial básico para la estructura DART. Chapman & Pinflod, 1999.

La estructura del DART toma como base diseños anteriores, reglas de diseño, librerías, experiencia, parámetros legales y variables de optimización. Toda esta información se pudo estructurar basándose en KBE.

Como resultado, se tiene una interfaz de desarrollo que se ha podido llegar siguiendo una secuencia de etapas:

a. Compresión del proceso a desarrollar: Mediante la identificación del ciclo de vida del producto, y usando el conocimiento especializado de todas las áreas involucradas en el desarrollo del producto.
b. Creación de las partes del objeto como modelos conceptuales: A través del conocimiento compartido y el desarrollo de partes inteligentes, las cuales son objetos con funciones, métodos y reglas.
c. Modelamiento y representación de resultados: Con el apoyo de un software CAD.

(Chapman & Pinfold, 1999)

3.6 ESTUDIO N°4: Investigación de diseño conceptual de productos electromecánicos basados en KBE

Huabo He desarrolló el diseño de una caja de transmisión utilizando como base de su diseño la ingeniería basada en conocimiento. Inicialmente describe el diseño conceptual como el proceso de crear, descomponer y diseñar funciones de acuerdo a los requerimientos del producto en cada etapa del ciclo de vida.

Entonces, para llegar a un correcto KBE, empieza con la adquisición del conocimiento, el cual se pude lograr de dos formas: descubriendo el conocimiento (KD) o minando los datos (DM). El primero es un proceso de búsqueda de información de las diferentes fuentes que se pueda encontrar, ya sea libros, personal de experiencia o de la misma innovación. Mientras que la minería de datos está asociada a la inteligencia artificial, bases de datos, tecnología de visualización y otra información que es necesario descubrir mediante el análisis de esta información.

Con todo lo mencionado, se hace el estudio del sistema de transmisión mecánico utilizando el modelo conceptual de diseño electromecánico:

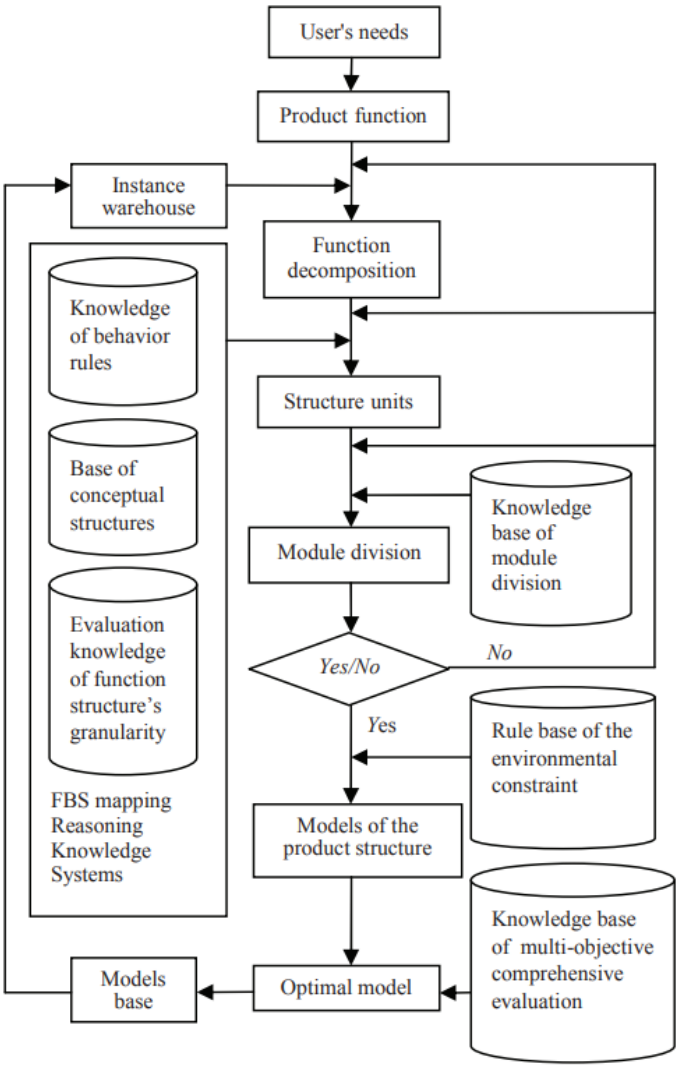

Ilustración 14. Modelo de un diseño conceptual de productos electromecánicos basados en KBE. Huabo He et al., 2011.

El uso de este flujo permite desarrollar el diseño de la caja de transmisión mecánica utilizando parámetros como las propiedades mecánicas, potencia de transmisión, peso, adaptación a altas temperaturas, protección frente al polvo, corrosión, humedad, etc. Todas estas variables pueden ser agrupadas utilizando un análisis de componentes principales (PCA) y descritos luego como condicionales (si se necesita alta precisión y eficiencia, entonces el material deberá ser de bajo carbono o un acero aleado) (Huabo He et al., 2011).

De este trabajo se rescata el desarrollo del flujo de diseño de componentes electromecánicos utilizando KBE.

3.7 ESTUDIO N°5: Un eficiente método de búsqueda basado en simulación para optimización del diseño robusto basado en confiabilidad de elementos mecánicos

Actualmente, la optimización bajo incertidumbre del diseño está teniendo bastante importancia en el campo de la ingeniería. La optimización del diseño robusto basado en confiabilidad (RBDO por sus siglas en inglés) toma dos campos del diseño poco explorados que son la optimización basada en confiabilidad y la optimización del diseño robustos, para así poder encontrar un diseño a base de probabilidad de fallas de un sistema donde los parámetros varían de manera aleatoria, pero conociendo la distribución de la probabilidad.

En el desarrollo del documento, como parte del flujo de trabajo de RBDO, confirma que técnicas de inteligencia artificial, como para el caso las redes neuronales, resultan ser más eficientes que los diseños con FEM (método de elementos finitos). Para llegar a esta conclusión se entrenó una red neuronal con 75 de 101 diseños generados por FEM de una manija de ventana de un auto y se encontró una fuerte correlación entre las variables objetivo (para el caso fueron masa y factor de seguridad) con las variables de diseños (medidas del objeto de diseño).

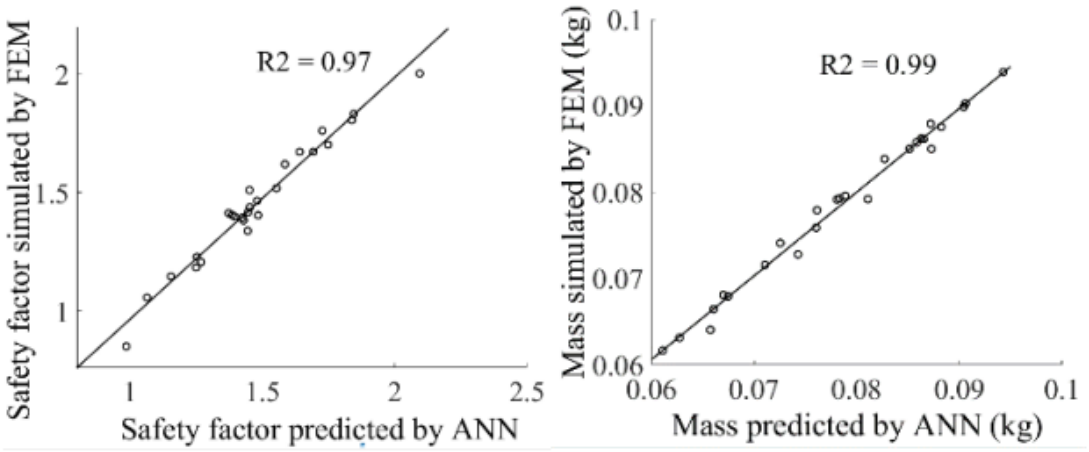

Ilustración 15. Regresiones lineales de la comparación entre el desarrollo con FEM y ANN. Mayda, 2017.

Los resultados confirman que lo obtenido por FEM es muy similar que los resultados obtenidos por ANN (Red Neuronal Artificial) con la diferencia que una vez entrenada la ANN los cálculos generados se obtienen más rápido, y por ende, es más fácil encontrar el diseño óptimo.

Finalmente, los resultados del estudio muestran que el análisis global tuvo un tiempo de cálculo de 20 segundos, lo que es rápido en comparación con lo que tardaría diseñar con FEM. Además, el autor propone otros caminos a seguir como integrar este desarrollo con otros métodos heurísticos de optimización como lo son los algoritmos genéticos (Mayda, 2017).

3.8 ESTUDIO N°6: Metodología de diseño de Ingeniería Basada en Conocimiento (KBE) en niveles de pregrado y posgrado

En este estudio se describe cómo se pueden cumplir con los requerimientos técnicos solicitados para un producto a través de la ingeniería y las decisiones de gestión. Además, cómo es que el proceso de diseño es una de las etapas más importantes en el desarrollo de un producto ya que de este depende los costos, los procesos de fabricación y el valor agregado que lo diferencian. A su vez se muestra que, en el ciclo de vida de un producto, el proceso de diseño es el que más impacto tiene debido a que aquí es donde se encuentra las mayores oportunidades para ahorra en costos. Aproximadamente el 70% de los costos totales se vuelven permanentes una vez acabado el proceso de diseño.

Además, se hace una descripción de cómo se debe segmentar los tipos de conocimiento, como es que de estos pueden generarse reglas de diseño que participarán directamente en el proceso. De esta manera, el proceso de diseño se entrelaza directamente con el conocimiento que se debe aplicar en cada subetapa de esta (Calkins et al., s. f.). Se tiene entonces:

1. Conocimiento

a. Estándar

Es el más básico de todos los conocimientos. Normalmente es conocimiento estándar que se encuentra en los manuales y libros académicos. A diferencia de los demás, este conocimiento es altamente estandarizable por lo que se puede construir una tabla de base de datos con él.

b. Procedimental

Es de dos tipos: el conocimiento algorítmico constituye un conjunto de pasos para poder resolver un problema, donde por lo general para poder hacerlo transforman hechos o parámetros de un estado a otro. Por otro

lado, el conocimiento operativo es el que se encarga de transportar, crear y modificar la información; es el punto de unión entre el desarrollo interno de un programa la interfaz de esta. El conocimiento operativo se encuentra en los métodos de optimización, cálculo por elementos finitos, gestión de base de datos, etc.

c. De juicio

Es aquel que se genera de la experiencia. Por lo general, no es tan sencillo transcribirlo a un programa computacional, debido a que está ligado a la observación, juicio y buenas prácticas descritas en procedimientos. Si se desea usar dentro de una interfaz es necesario detallar la lógica de este conocimiento para describir los principios formales de ese razonamiento.

d. Control

Este es el conocimiento general; es decir, es el conocimiento del conocimiento, el que describe cómo se va a usar los tres conocimientos antes mencionados y en qué etapas del proceso de diseño se usará. Algunos ejemplos de este conocimiento son:

- Anticipación de desarrollos inesperados.
- Saber cómo trabajar frente a la incertidumbre de cierta información.
- Saber escoger un algoritmo frente a diferentes opciones según convenga.

2. Reglas de diseño

Estas reglas son una síntesis del conocimiento recabado y se establecen para determinar cómo este será usado en el modelo desarrollado; es decir, los métodos y los procesos de un ingeniero son imitados por estas reglas.

a. Analíticas – numéricas

Estas son reglas paramétricas. Definen las fórmulas o algoritmos simples a emplear dentro del proceso de diseño.

b. Heurísticas

Son reglas usualmente condicionales, las cuales están relacionadas directamente a las “buenas prácticas” que se han de emplear cuando se tiene el *know-how* del desarrollo de un

producto. Usualmente la conocen empresas o fabricantes con bastante experiencia en el desarrollo de un producto en específico.

c. **Empíricas**

   Estas están basadas en información plasmada en expresiones de curva de ajuste que han sido desarrolladas por información experimental.

d. **De restricciones legales**

   Son reglas relacionadas al producto a desarrollar y que deben de acatarse casi definitivamente debido a que están sujetas a la ley o procesos de ingeniería ya determinados.

De este estudio se rescata la forma en la que se generaliza la información y se plasma en el proceso de diseño. Esta se convertirá en la base para la generalización del proceso de diseño de un elemento mecánico.

Capítulo 4 MECANISMO DE EXTRACCIÓN DEL CONOCIMIENTO DEL PROCESO DE DISEÑO MECÁNICO

4.1 Introducción

En este capítulo se revisará el primer objetivo específico planteado en la investigación. Para esta parte, apoyado en la metodología KBE, se busca extraer el conocimiento inmerso dentro del proceso de diseño enfocado al desarrollo de elementos mecánicos.

Sin embargo, No solo se trabaja el diseño del producto, sino que se debe asegurar la optimización del mismo. En ese sentido se trabajará en el diseño de un producto de calidad y al menor costo.

Cuando nos referimos a un producto de calidad se describe a aquel que cumple con los requerimientos del cliente y a su vez tiene un tiempo de vida lo más extendido posible. En ese sentido, se debe optimizar el resultado de los requerimientos del cliente (los cuales dependerán del elemento mecánico a diseñar) y los esfuerzos de trabajo dentro del material. Mientras menor sea la concentración de esfuerzos, mayor duración tendrá el elemento.

Por otro lado, cuando se analiza el costo de fabricación, para el caso se describirá como la cantidad de material utilizado, se considera que, mientras menor sea el peso del elemento, menor será su costo.

Con la información mostrada, se busca captar los datos de entrada, salida y fórmulas dentro del proceso para el desarrollo del método de optimización mediante algoritmo genético que se verá en el siguiente capítulo.

4.2 Tipos de Conocimiento en el diseño mecánico

4.2.1 Introducción

Tal como se definió en el estudio N°6:” Metodología de diseño de Ingeniería Basada en Conocimiento (KBE) en niveles de pregrado y posgrado”, para la extracción del conocimiento es necesario identificar aquellos que se encuentran de manera implícita y explícita dentro del proceso de diseño.

En ese sentido, se revisará el desglose del conocimiento inmerso en el proceso de diseño enfocado al desarrollo de elementos mecánicos. Esto se tomará como base para describir el caso de estudio de un resorte helicoidal cilíndrico de sección circular.

4.2.2 Resultado alcanzado

Se procede a desarrollar el desglose del conocimiento enfocado al proceso de diseño de elementos mecánicos:

a. Conocimiento estándar

La información estandarizada para elementos mecánicos se encuentra dentro de las normas técnicas asociadas. Tenemos así dentro de las más usadas: ASTM, SAE, AISI, DIN, JIS. Para el caso del Perú, la norma base es la NTP (Norma Técnica Peruana). En este tipo de conocimiento, asociado a las normas, encontraremos las características técnicas de los materiales que posteriormente nos podrán ayudar en el proceso de diseño; siempre y cuando se puedan extraer los parámetros específicos del material, tales como esfuerza de rotura (*tensile strength),* esfuerzo de fluencia *(yield strength),* densidad volumétrica y módulo de elasticidad *(módulo de Young).* Para esto, se recomienda al lector revisar las especificaciones técnicas necesarias de los materiales para soportar el diseño, para posteriormente revisar su funcionamiento. Estas normas, en algunos casos, recomiendan materiales según el tipo de aplicación que se desee trabajar. A su vez, existe material bibliográfico en donde se puede encontrar propiedades de materiales, estos pueden ser manuales o libros especializados en diseño mecánico. Para un mejor desarrollo, es recomendable identificar las propiedades necesarias y estructurar la información dentro de una base de datos para poder consultar y evaluar el comportamiento de los materiales al momento del diseño.

b. Conocimiento Procedimental

En este caso el conocimiento procedimental aplicado al diseño mecánico se muestra como un algoritmo que describe las fórmulas de cálculo para obtener el producto; es decir, el flujo paso a paso de cómo se deben utilizar los datos de entrada para obtener determinados datos de salida.

En el contexto de diseño mecánico, este conocimiento se utiliza para transformar las propiedades de los materiales a usar en conjunto con las dimensiones geométricas y las restricciones del entorno para generar fuerzas y momentos internas propias del equilibrio dinámico o estático. También es considerado como parte de este conocimiento a los algoritmos de optimización, aquellos que buscan mejorar los valores encontrados en función a la maximización o disminución de ciertas variables.

Para el diseño mecánico, esta información se puede encontrar en libros especializados y también de normas técnicas que indiquen este flujo de trabajo.

c. Conocimiento de Juicio

En el diseño mecánico este se puede ver reflejado en las buenas prácticas del diseño el cual suele ser propio del know-how de la industria a la cual pertenezca el elemento mecánico; es decir, las observaciones empíricas que se consiguen con la experiencia práctica y la comparación entre los modelos desarrollados y los productos obtenidos. También, las normas suelen dar recomendaciones para el diseño las cuales no necesariamente deben cumplirse, pero es conocimiento recopilado y expuesto en el documento. Por otro lado, dentro de este conocimiento también está incluido las buenas prácticas orientados al usuario y a la calidad del producto que está recibiendo como entregable. En ese sentido, el uso de una norma de gestión de la calidad servirá para entender los requerimientos del cliente sobre el desarrollo del elemento mecánico que desea.

d. Conocimiento de Control

El metaconocimiento propio del diseño mecánico va a utilizar la información recopilada para saber cómo usarlo y en qué partes del proceso de diseño. En ese sentido, este conocimiento actuará de manera transversal a través de los otros tres.

Para el diseño de un resorte cilíndrico helicoidal de compresión de sección circular el conocimiento de control estará relacionado a cómo usar las normas SAE, ASTM, la información del libro “Diseño en ingeniería mecánica de Shigley” y las buenas prácticas propias del know-how de la industria especializada.

Conocimiento de Control		
Uso adecuado del conocimiento sobre diseño mecánico recopilado en los tres tipos de conocimientos (Ejem: en qué partes del proceso aplicar cada tipo de conocimiento y qué parte de la información será útil).		
Conocimiento Estándar	**Conocimiento Procedimental**	**Conocimiento de Juicio**
- Normas técnicas de materiales (Ejem: ASTM, SAE, AISI). - Libros especializados en diseño mecánico (como recopilación de datos sobre materiales).	- Normas técnicas de procedimiento de diseño (Ejem: SAE, NTP) - Libros especializados en diseño mecánico (como base de fórmulas y flujo de diseño de pieza mecánica).	- Normas técnicas: Recomendaciones de diseño (Ejem: SAE, NTP). - Buenas prácticas de la industria especializada. - Normas orientadas a la gestión de la calidad (Ejem: ISO 9001)

Tabla 1 Tipos de conocimiento aplicado al diseño de elementos mecánicos. Elaboración propia

En la Tabla 1 se muestra el resumen del conocimiento aplicado al diseño de un elemento mecánico.

4.3 Reglas de diseño aplicado a los componentes mecánicos

4.3.1 Introducción

Teniendo los tipos de conocimientos como datos de entrada, es posible moldear reglas de diseño que servirán para saber cómo este conocimiento será usado de manera directa en el proceso de diseño. Para esto, las reglas de diseño creadas se agruparán según el tipo y en qué etapa del proceso de diseño se usen. Como resultado de esto se tendrá un flujo mostrando la información que interactúa a lo largo.

Al igual que en el resultado anterior, una vez definidas las reglas se procede a aplicar estas en el estudio de caso de un resorte cilíndrico helicoidal de sección circular.

4.3.2 Resultado alcanzado

Con la información recopilada se procede a detallar como esta se usará en el diseño de elementos mecánicos mediante reglas:

a. Reglas analíticas-numéricas

Las reglas analíticas para los elementos mecánicos están relacionadas con los parámetros y las fórmulas paramétricas asociadas. Entonces, con respecto a los parámetros, los principales serán las dimensiones del producto que participarán en el proceso de diseño y que a partir de ellos se generarán parámetros adicionales a través de las fórmulas. Dichas fórmulas, son obtenidas del conocimiento procedimental y lo que hacen es transformar los datos de entrada. Para el diseño de un elemento mecánico como mínimo se deberán trabajar los parámetros adicionales de esfuerzo de trabajo (τ_{trab}) y peso (w). Además de esos, se deberán trabajar los parámetros necesarios para satisfacer los requerimientos técnicos solicitados los cuales dependerán del elemento a diseñar.

b. Reglas heurísticas

Las buenas prácticas del desarrollo de un elemento mecánico son recopiladas del conocimiento de juicio extraído anteriormente. Estas están relacionadas a cómo el diseño debe ampliarse y no solo considerar reglas teóricas sino también las prácticas; en ese sentido, se debe tener en cuenta las condiciones de producción del elemento, así como sus restricciones. También ha de considerarse las condiciones de trabajo del elemento y cómo este interactúa con otras partes de la máquina en donde será instalado. Estas condiciones de trabajo podrán ser: espacio de trabajo (restricción dimensional), temperatura, frecuencia, humedad, entre otros.

c. Reglas empíricas

Se usará el conocimiento estándar para la recopilación de las propiedades mecánicas específicas de los materiales. Para esto, como mínimo se deben recopilar propiedades como esfuerzo de rotura del material (*tensile strength)* y la densidad volumétrica, ya que estos son datos necesarios para poder elaborar la optimización del elemento mecánico utilizando algoritmos genéticos (Capítulo 5). Sin embargo, cualquier elemento será diseñado para satisfacer ciertos requerimientos técnicos indicados por el cliente; en ese sentido, se deberán de revisar otras propiedades que se necesiten para asegurar el adecuado funcionamiento del elemento en base a dichos requerimientos. Por ejemplo, para el diseño de un engranaje es necesario revisar la norma AGMA, de donde se

extraen los factores de velocidad (K_v) los cuales dependen del material y la forma desbaste del material, así como el módulo de Young.

d. Reglas de restricción legal

Del conocimiento de juicio se extrae información sobre las recomendaciones técnicas en general que la normas nos pueda suministrar. Por lo general, este tipo de reglas suelen mostrarse en una inecuación en donde se sugiere que algún parámetro del diseño permanezca dentro de un rango establecido con la finalidad de proteger de un mal funcionamiento al elemento mecánico.

Por otro lado, de este tipo de conocimiento también se extrae el conocimiento de los requerimientos técnicos solicitados. Para esto, utilizando la norma ISO 9001 nos podemos apoyar en citas específicas:

- ISO 9001: 2015 Sección 8.3.3 item a: “La organización deberá considerar los requisitos funcionales y de desempeño”.
- ISO 9001: 2015 Sección 8.3.5 item a: “La organización deberá asegurar que las salidas del diseño y desarrollo cumplen con los requisitos de entrada”.

En ese sentido los requerimientos técnicos deberán medirse en función a un porcentaje de **error** entre lo solicitado y lo calculado. Esto se debe cumplir para cada uno de los requerimientos deseados. Además, como consecuencia del proceso de diseño se obtendrá como datos de salida las dimensiones del producto con su respectivo material asignado los cuales deben minimizar el error frente a las demás opciones. A su vez, los valores de esfuerzo y peso están implícitos dentro de estas dimensiones ya que son consecuencia de su geometría; estos también deberán minimizarse respecto a las propiedades del material.

De esta manera, en la Ilustración 16. se muestra el flujo de input/output para el proceso de diseño:

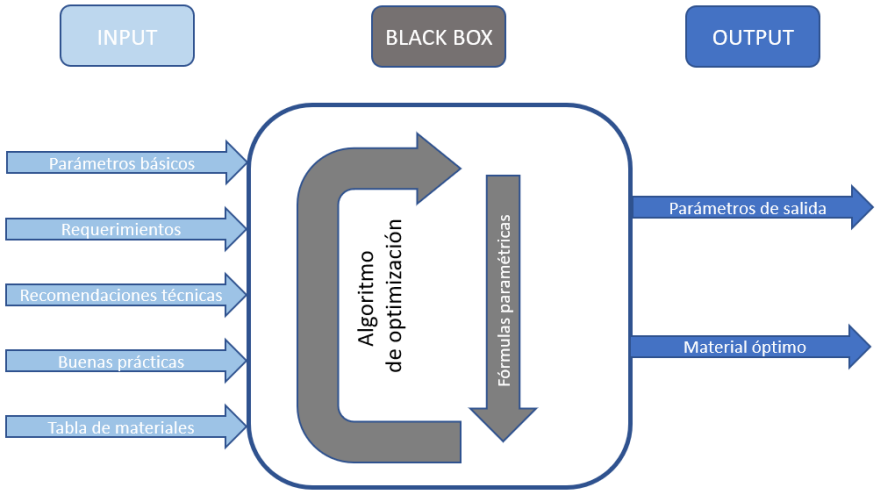

Ilustración 16. Flujo de diseño de elemento mecánico general. Elaboración propia.

4.4 Conclusiones

- Se extrajo el conocimiento implícito dentro del proceso de diseño de un elemento mecánico.
- El conocimiento fue plasmado mediante reglas dentro del proceso de diseño.
- Las reglas fueron asignadas como datos de entrada, datos de salida o caja negra.
- El flujo desarrollado muestra los datos que se deberán trabajar en el algoritmo genético.

Capítulo 5 PARÁMETROS BÁSICOS PARA IMPLEMENTAR UN ALGORITMO GENÉTICO DE DISEÑO DE PIEZAS MECÁNICAS

5.1 Introducción

En este capítulo se aborda los pasos iniciales para definir un algoritmo genético. Este está compuesto por una población de cromosomas, un indicador de un buen o mal resultado llamado función fitness y finalmente el proceso de convergencia que es el que define cuando el bucle de búsqueda se detendrá.

Basándonos en esos puntos clave, el primer resultado esperado obtendremos la definición de un componente mecánico convertido a un cromosoma; en el segundo resultado esperado el desarrollo de la función fitness que recopile todos los requerimientos propios del diseño mecánico; y en el último resultado esperado se mostrará la estructura que presenta el criterio de convergencia del algoritmo.

5.2 Diseño de un cromosoma que modele el comportamiento de un componente mecánico

5.2.1 Introducción

Un cromosoma recopila el conjunto de características que tiene un objeto y que introducidos dentro del algoritmo genético estarán variando de manera continua a lo largo de las generaciones. A su vez, en cada generación se tendrá una población como conjunto de cromosomas que posteriormente se compararán entre sí.

Las características inmersas dentro de cada cromosoma son las que le darán un valor a este, por lo que generalmente tienen una representación numérica. Por ende, para el caso de un componente mecánico, las propiedades del material y su geometría serán las que se tomen como características del cromosoma. Para poder conseguir esa información, se toma como base las reglas de diseño descritas en el capítulo anterior.

5.2.2 Resultado alcanzado

El diseño del cromosoma debe contemplar los datos de entrada necesarios para un proceso común de diseño. De esta manera, las reglas de diseño a usar en esta parte serán los **parámetros básicos** definidos y la **tabla de materiales**. Estos están controlados por las restricciones en el desempeño del elemento; en ese sentido, también se deberán considerar las **recomendaciones técnicas** y las **buenas prácticas** propuestas.

Como se explicó en el capítulo anterior, los parámetros básicos son las dimensiones geométricas que el elemento tiene para ser descrito. Estas a su vez, actuarán como elementos de entrada para la generación de fórmulas posteriores. Entonces, cada una de las dimensiones que el elemento tendrá se mostrará en el algoritmo como un gen dentro del cromosoma.

Por otro lado, del material seleccionado se rescatan sus propiedades mecánicas. De esa manera, cada una de las propiedades mecánicas del material se muestren como un gen dentro del cromosoma.

Con lo anterior, se consigue el perfil del cromosoma en la Ilustración 18. donde la longitud de este será igual al número de dimensiones básicas a analizar más el número de propiedades mecánicas de cada material.

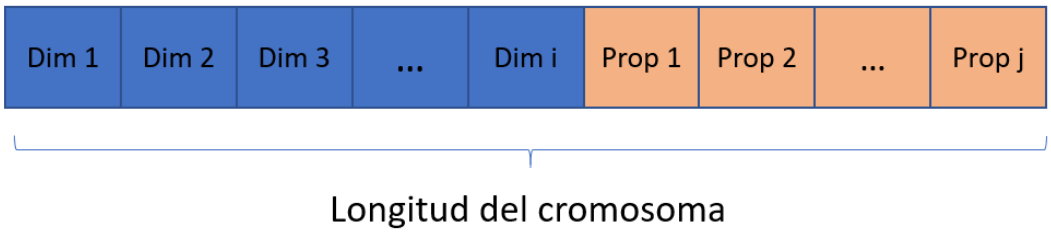

Ilustración 17. Longitud del cromosoma para diseño de elementos mecánicos. Elaboración propia

Se debe tomar en cuenta, las restricciones que existen en cada una de las dos partes que conforman al cromosoma. Para esto se analizarán por separado:

- Dimensiones: El tipo de variación que cada dimensión podría tener según el rango para la selección de la dimensión óptima podría ser continua o intermitente. De ser continua solo se tendría que definir el valor máximo y mínimo permitido; mientras que de ser intermitente además de los valores

máximo y mínimo también se debe considerar el valor de la intermitencia. Otra forma de restringir las dimensiones es si esta se encuentra dentro de una lista de valores predefinidos. Se puede tomar como ejemplo las dimensiones de stock del diámetro de un eje el cual no requiere proceso de mecanizado (solo se tienen en milímetros o pulgadas).

- Propiedades: Los materiales son los que definirán todas las propiedades que se utilizarán dentro del cromosoma, por lo que estas estarán entrelazadas entre sí. Esto debe considerarse a la hora de generar el cromosoma de manera aleatoria: las propiedades no podrán variar para un mismo material; es decir, no se pueden asignar datos aleatoria que no estén entrelazados.

	Material 1	Material 2	Material 3
	Propiedad 11	Propiedad 12	Propiedad 13
	Propiedad 21	Propiedad 22	Propiedad 23
	Propiedad 31	Propiedad 32	Propiedad 33

Tabla 2. Ejemplo de propiedades asignadas a cada material. Elaboración propia

De la tabla 4, se observa que para cada material existen tres propiedades asociadas por lo que nunca se tendrá en un solo cromosoma la propiedad12 con la propiedad23, ya que estas pertenecen a diferentes materiales. Entonces, se puede reacomodar el cromosoma como se observa en la ilustración 19:

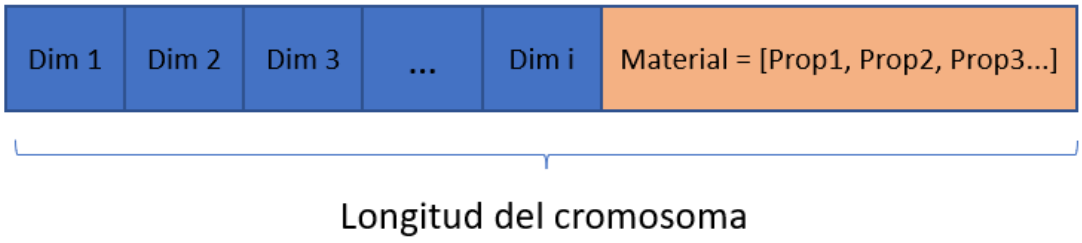

Ilustración 18. Longitud del cromosoma para un solo material. Elaboración propia

5.3 Desarrollo de una función fitness general

5.3.1 Introducción

Para la evaluación de cada cromosoma, será necesario asignar un valor cuantitativo que represente cuán bueno o malo es dicho cromosoma; para esto,

la población deberá ser comparada en cada generación rescatando aquellos cromosomas con mejores cualidades.

Por otro lado, para definir si un cromosoma es bueno o malo esto dependerá de cuán próximo esté su desempeño actual a los requerimientos deseados. Asimismo, el desempeño del cromosoma dependerá de sus condiciones geométricas y mecánicas (para el caso de un elemento mecánico). En ese sentido, el desempeño esperado dependerá no solo de un parámetro, sino de un conjunto de ellos, los cuales deben ser convertidos en un único número que los represente para su posterior comparación (algoritmo multiobjetivo).

5.3.2 Resultado alcanzado

Con la información del capítulo anterior, se rescatan las buenas prácticas y las recomendaciones técnicas para desarrollar restricciones en la búsqueda del cromosoma ideal. Además, dicha búsqueda llegará a su fin si los requerimientos son satisfechos; sin embargo, al existir un conjunto de requerimientos, no se puede indicar que existe un solo cromosoma que satisfaga por encima de todos los demás, a todos los requerimientos planteados; en ese sentido, existirá una lista de soluciones que se pueden considerar como óptimas. Posteriormente, el diseñador deberá escoger cuál de todas esas opciones se acomoda mejor a los valores deseados.

Para alcanzar el conjunto solución, se necesita revisar los conceptos de restricciones y requerimientos.

a. Restricciones

Estas impedirán que se creen cromosomas que contengan información, que, sin ser evaluada, de por si ya se encuentran descartada. Las razones por las cuales se descartaría están relacionadas a las condiciones de trabajo del elemento. Por ejemplo, si el elemento fuese un tornillo que se desea insertar dentro de una cavidad de diámetro “d”, el diámetro del tornillo no podría ser mayor a esta, ya que físicamente sería imposible introducirlo. Otro ejemplo sería que la pieza trabajase a una temperatura “T°” sabiendo que el material no soporta dicha temperatura.

Viendo esos ejemplos, queda definido que las restricciones serán aplicables para la geometría de la pieza (características dimensionales) y para el material

asignado, ambos son componentes del cromosoma, por lo que al momento de crearlos aleatoriamente se debe revisar si existe alguna restricción en cada gen y se deberá de colocar de forma expresa.

Dim1 mín. ≤ Dim1 ≤ Dim1 máx.	Dim2 mín. ≤ Dim2 ≤ Dim2 máx.	Dim3 mín. ≤ Dim3 ≤ Dim3 máx.	...	Dimi mín. ≤ Dimi ≤ Dimi máx.

Ilustración 19. Restricciones asignadas a cada gen. Elaboración propia.

Si en caso no se tuviera restricción mínima o máxima en algún gen, esta tiene que ser ingresada ya sea con algún criterio de diseño o en su defecto por consideración propia del diseñador (las dimensiones no pueden tomar un valor negativo o nulo ni tampoco dimensiones muy altas en comparación a las demás). Esto es necesario debido a que en el momento de crearse los genes de manera aleatoria estos deben tener un rango de valores de donde se escogerá.

b. Requerimientos

Los requerimientos son condiciones que el cromosoma deberá de alcanzar o aproximarse lo más posible. Cada uno de estos se convertirá en una función fitness que describirá el comportamiento del cromosoma en función a ese objetivo (requerimiento); posteriormente, todos deberán ser introducidos en una solo función fitness global que los represente. Ésta será la que se compare con otros cromosomas para determinar quién persiste en la próxima generación.

Se debe considerar dos tipos de requerimientos:

- Requerimiento de funcionamiento mecánico

Como se describió en párrafos anteriores, los elementos mecánicos serán analizados y optimizados basándose en dos conceptos básicos: esfuerzos internos (capacidad del material de soportar las cargas asignadas al elemento) y el peso (asociado al costo del producto). En ambos casos lo que se busca es minimizarlo; sin embargo, se toma en cuenta que estos valores nunca podrán llegar a ser nulos (cero), por lo que deben tener un nivel de referencia con el fin de compararlo respecto a este.

Para el caso del esfuerzo, este se deberá comparar con el esfuerzo de fluencia correspondiente al material asignado multiplicado por un factor de corrección del esfuerzo:

$$f_\tau: función\ fitness\ de\ esfuerzo$$

$$\tau < b * S_y$$

De esta manera, se puede generar un indicador que relacione el esfuerzo de trabajo del elemento con el permisible por el material. Este valor será asociado a la función fitness:

$$f_\tau = \frac{\tau}{b * S_y}$$

Por otro lado, para el caso del peso, este depende del volumen y la densidad volumétrica del material. El primero se calcula en función a las características geométricas de la pieza, mientras que el segundo es consecuencia del material a utilizar. Debido a que no se tiene un peso máximo ni mínimo de referencia (a menos que las restricciones revisadas anteriormente así lo contemplen), es posible encontrar el peso máximo si se tiene los extremos de cada gen; es decir, el valor mínimo y máximo que estos tendrán al momento de crear la pieza de manera aleatoria. Por lo general, el peso máximo se encuentra utilizando los valores máximos de cada gen, con esto se maximizará el volumen de la pieza. Por otro lado, para el material, solo se deberá escoger aquel que dentro de la lista tenga la mayor densidad volumétrica.

$$w \rightarrow depende\ de\ \rho_v\ y\ V$$

$$f_w: función\ fitness\ de\ peso$$

$$w < w_{max}$$

De la misma manera que con el esfuerzo, se genera un indicador que relacione el peso asociado a los parámetros del cromosoma y el peso máximo que los cromosomas podrían generar en caso de escoger los valores extremos de cada gen.

$$f_w = \frac{w}{w_{max}}$$

En ambos casos, al dividir las condiciones de trabajo de esfuerzo y peso respecto a un valor máximo no solo se generan las funciones fitness, sino que también estamos normalizándolas para que estas puedan interactuar posteriormente sin asignarle mayor peso a alguna por el momento.

- Requerimientos particulares del elemento

Este tipo de requerimiento está asociado directamente al correcto funcionamiento del elemento tal cual se ha solicitado. Para este caso, no se busca maximizar o minimizar un parámetro, sino más bien, aproximarse lo más que se puede a un valor fijo. En ese sentido, se debe encontrar una forma de convertirlo en un problema de optimización (máximos o mínimos). Esto se alcanza introduciendo el concepto de error como la función fitness asociada, entonces, lo que se busca sería la minimización del error.

$$f_i: función\ fitness\ de\ requerimiento\ i$$

Se tiene como objetivo que el requerimiento generado por un cromosoma se aproxime lo más que se pueda al requerimiento esperado:

$$Req_{real_i} \approx Req_{esperado_i}$$

La expresión anterior se puede transformar en una ecuación:

$$Req_{real_i} + error_i = Req_{esperado_i}$$

Debido a que el error no se afecta por el signo, se debe considerar el valor absoluto de este:

$$error_i = \left|Req_{esperado_i} - Req_{real_i}\right|$$

Entonces, conociendo el error, es posible normalizarlo tomando como base el requerimiento esperado. Esta sería la función fitness para cualquier requerimiento:

$$f_i = \frac{error_i}{Req_{esperado_i}}$$

Se debe considerar que puede haber tantos requerimientos particulares como sea solicitado y son estos los que tienen mayor prioridad sobre los de funcionamiento mecánico ya que se relacionan directamente a la calidad del elemento y como este cumple las expectativas. Para esto se debe recordar las reglas de restricción legal.

Con los requerimientos específicos ya definidos se necesita ahora insertarlos en una sola función: función fitness global. Ésta debe contemplar cada una de las funciones fitness encontradas antes y transformarlas en un único número que califique al cromosoma. Se tiene entonces:

$$f_T : f(f_\tau, f_w, f_1, f_2, \dots, f_n)$$

Donde n es el número de requerimientos particulares.

La manera en cómo estas funciones fitness van a interactuar entre sí, dependerá del nivel de importancia que se le asigne a cada una. Además, se debe tomar en cuenta que todas las funciones antes revisadas buscan reducir su valor lo más posible, por lo que se está hablando de un problema de minimización en todos los casos.

Se han revisado dos opciones que el diseñador puede optar para encontrar la función fitness global.

a. Suma de funciones fitness

$$f_T = w_1 * f_\tau + w_2 * f_w + w_3 f_1 + \cdots + w_{i+2} f_i$$

Según el requerimiento inicial, cada función fitness aporta un valor, el cual se espera minimizar. Al sumarse dichos valores esta función se comportará como una compuerta “OR”, en donde suficiente es que uno de los valores sea alto en comparación con los otros, para que estos sean absorbidos por el de mayor valor.

b. Producto de funciones fitness

$$f_T = {f_\tau}^{w_1} * {f_w}^{w_2} * {f_1}^{w_3} * \ldots {f_i}^{w_{i+2}}$$

En este tipo, todas las funciones fitness aportan en función al peso (w_i) asociado a cada una. Si una de ellas presenta un valor alto y otra un valor muy bajo, al multiplicarse se obtendrá un valor intermedio que promedie el impacto de ambas en la función global.

El uso de cualquiera de las dos formas, dependerá de cómo estén los requerimientos relacionados entre sí y el grado de error que se espera obtener en cada uno. Por ejemplo, si se tiene dos funciones fitness en donde el error esperado es del 1% y 0.001% (error permitido por cada función para considerar al cromosoma como óptimo), si estas se suman, el primer resultado absorbería al segundo (1.001%≈1%), mientras que, si se multiplica, el resultado sería la interacción de ambas dando un valor aún más pequeño (0.00001%).

En ambos casos es necesario encontrar las constantes que mejor se acomoden al algoritmo y que reduzcan el número de generaciones.

5.4 Selección de criterio de convergencia

5.4.1 Introducción

Con la función fitness y el modelo de cromosoma definido es posible implementar el algoritmo; sin embargo, al ser este un bucle, deberá existir una condición que lo detenga en el momento oportuno y entregar los resultados deseados.

5.4.2 Resultado alcanzado

La condición de detención del algoritmo está directamente asociada a las funciones fitness creadas, ya que si se espera una lista de cromosomas que cumplen los requerimientos solicitados, estos deberán tener el menor error posible. Con esa premisa, se llega a la conclusión que la condición de detención está asociada al error que los cromosomas podrán tener para ser aceptados.

Los requerimientos de funcionamiento mecánico no presentan un error de por medio, lo que se busca en ellos es encontrar cromosomas donde estos valores sean los menores posibles sin afectar a los requerimientos particulares. Por otro lado, serán los requerimientos particulares quienes detengan el algoritmo. Para esto, se necesita que por cada función fitness de requerimiento particular se tenga un error máximo aceptable, y para poder detener el algoritmo, todos los requerimientos deberán estar por debajo de los errores asociados a cada una de las funciones. Entonces, al tener “i” requerimientos se tendrá “i” funciones fitness particulares e “i” errores máximos.

$$e = [e_{1_{max}}, e_{2_{max}}, e_{3_{max}}, \dots, e_{i_{max}}]$$

Al ser una población, se deberá comparar estos errores con el valor máximo respectivo de cada función fitness de cada cromosoma.

N: $Generación\ actual$

p: $Número\ de\ cromosomas\ en\ la\ población$

$$N = \{Cr_1, Cr_2, Cr_3, \dots Cr_p\}$$

Entonces, se tiene una matriz de valores fitness donde las filas indican el número de cromosomas y las columnas el número de funciones fitness que posee cada cromosoma.

$$f = \begin{bmatrix} f_{11} & \cdots & f_{p1} \\ \vdots & \ddots & \vdots \\ f_{1i} & \cdots & f_{pi} \end{bmatrix}$$

$$f_i = [f_{1i}, f_{2i}, \dots, f_{pi}] \rightarrow max\{f_i\} < e_{i_{max}}$$

5.5 Conclusiones

- Se ha diseñado la estructura de los cromosomas tomando como base la información que contendrán.
- Se han desarrollada las funciones fitness de funcionamiento mecánico y particulares. En ambos casos se busca su minimización.
- La condición de detención del algoritmo dependerá exclusivamente del error máximo obtenido en una generación de cromosomas por cada función fitness requerida.

Capítulo 6 MECANISMO PARA GENERALIZAR EL DISEÑO MECÁNICO USANDO UN ALGORITMO GENÉTICO

6.1 Introducción

Teniendo el conocimiento plasmado en datos de entrada para un algoritmo, ahora es necesario introducir toda la información para su posterior procesamiento y obtención de resultados. Dichos resultados no tendrán una solución única, ya que, como se vio en el capítulo anterior, existen varias soluciones para la optimización multiobjetivo; es decir, es poco probable obtener una solución que optimice todos los requerimientos, en su defecto, se obtendrá una lista de soluciones que finalmente deberá ser verificada por el diseñador según criterio.

En ese sentido, en esta sección se revisará el desarrollo del algoritmo genético descrito, usando los parámetros del elemento mecánico como datos de entrada, los cuales deberán tener límites (inferior y superior), el uso de fórmulas paramétricas para el procesamiento de los datos de entrada, y a través de los requerimientos, la obtención de una lista de soluciones que se elaborará en función a dichos requerimientos.

6.2 Adaptación del flujo general al algoritmo genético.

Para el desarrollo de este mecanismo de generalización de diseño se ha elaborado un flujo de actividades (Ilustración 20) el cual abarca todo lo que el algoritmo deberá contener para un correcto funcionamiento.

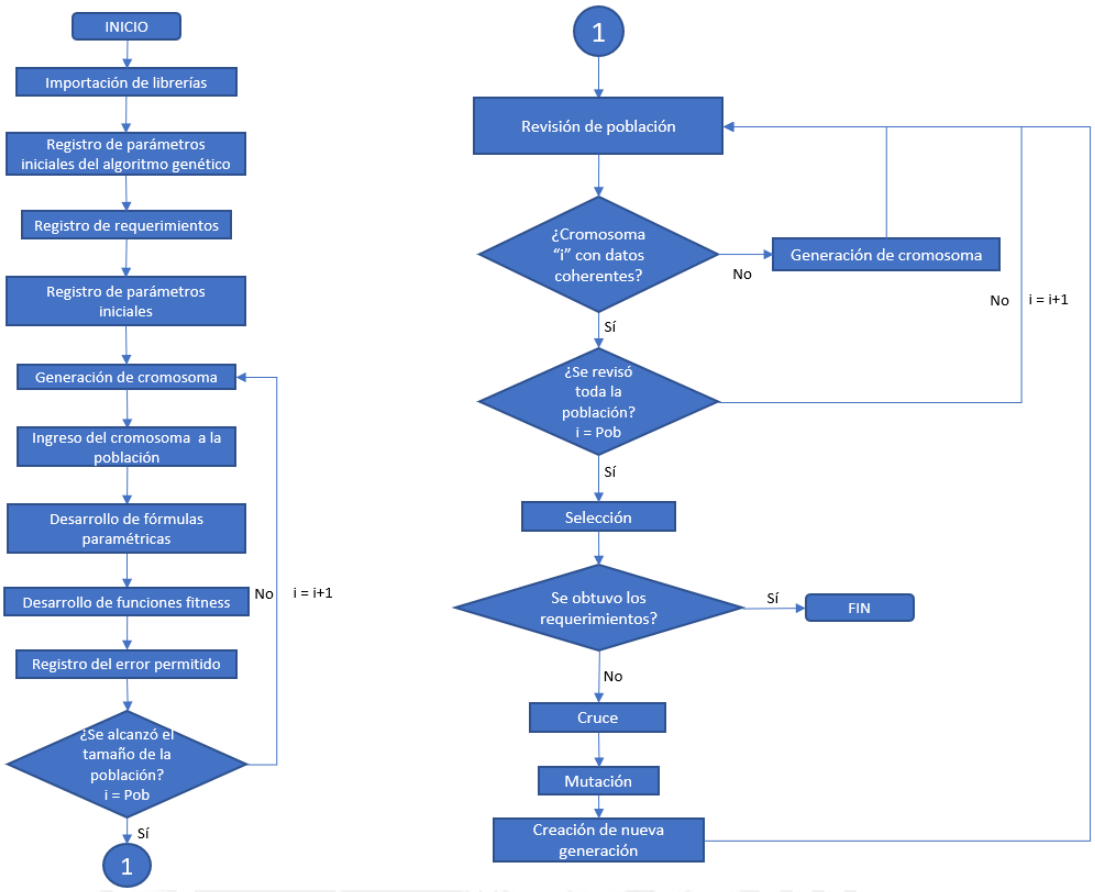

Ilustración 20. Flujo de desarrollo de un algoritmo genético. Elaboración propia.

6.2.1 Importación de librerías

Para desarrollar este tipo de algoritmo es necesario trabajar con valores aleatorios, fórmulas matemáticas, gráficas (para una mejor visualización del avance) y matrices. En ese sentido se deberán instalar las siguientes librerías:

```
1   from cmath import pi
2   import random
3   import matplotlib.pyplot as plt
4   import numpy as np
```

Ilustración 21. Librerías básicas a utilizar en el algoritmo generalizado. Elaboración propia.

6.2.2 Parámetros de control del algoritmo genético

Para el caso de un algoritmo genético, este tiene dos etapas que es necesario que ocurran de manera aleatoria; en ese sentido, deberá existir una probabilidad de ocurrencia del cruce y de la mutación. Dicha probabilidad es asignada por el diseñador en función a las pruebas de ajuste que se vayan haciendo.

También, el algoritmo debe registrar un número de cromosomas por población y cuántos de ellos pasarán a ser escogidos en cada iteración dentro de un número

de generaciones máximas permitidas (este último valor se deberá ingresar como dato adicional para evitar que el algoritmo caiga en bucle infinito.

Por otro lado, si se necesita revisar el código de una forma seudo aleatoria, se tendrá entonces que utilizar una semilla dentro de la función random().

```
#-----------------------------------------------------------
#Parametros de control del algoritmo genético
p_cruce = 0.8
p_mutacion = 0.7
pobinit = 200
bestpob = 0.7*pobinit
GEN = 1000
random.seed(123)
```

Ilustración 22. Parámetros básicos para el control del algoritmo generalizado. Elaboración propia.

6.2.3 Requerimientos

Los requerimientos deberán de ser enumerados e ingresados en un vector de requerimientos. Para esto se crea la función requerimientos(). En caso un requerimiento contemple más de un valor en conjunto (ejemplo: para este valor específico deberá entregar otro valor específico) estos deberán ocupar un espacio dentro del vector y ser ingresados como un subvector dentro.

```
def requerimientos():
    req = []
    req1 = valor1
    req2 = valor2
    req3 = [valor31, valor32]
    ...
    reqk = valork
    req.append(req1)
    req.append(req2)
    req.append(req3)
    ...
    req.append(reqk)

    return req
```

Ilustración 23. Creación de función de requerimientos generales. Elaboración propia.

6.2.4 Parámetros iniciales de diseño

Como se ha visto en capítulos anteriores, para el caso de diseño mecánico, el cromosoma podrá tener los genes de dos tipos: característica geométrica o material (este último como vector de almacenamiento de “n” propiedades).

En el caso que fuese una característica geométrica se debe tomar en cuenta que existen cuatro tipos de valores que esta puede adoptar:

- Entero (0)

- Decimal (1)
- Valor de lista (2)
- Intermitente (3)

```
#Si es entero
random.randint(inicio,fin) #selección de valores enteros

#Si es decimal
random.uniform(inicio,fin) #selección de valores decimales

#Si es valor de una lista
random.choice(rangon) #selección de valores de una lista

#Si es intermitente
random.randint(intermitencia*inicio, intermitencia*fin)/intermitencia
```

Ilustración 24. Clasificación del tipo de característica geométrica según los valores que puede tomar. Elaboración propia.

En ese sentido, se deberá incluir el tipo de variable que se manejará como un dato de entrada adicional (donde cada uno será representado por un valor entero). Se crea entonces la función parametrosIniciales():

```
def parametrosIniciales():
    rango1 = [inicio1, fin1, variable1, (intermitencia1)]
    rango2 = [inicio2, fin2, variable2, (intermitencia2)]
    ...
    rangon = [inicion, finn, variablen, (intermitencian)]
    rangos =[]
    rangos.append(rango1)
    rangos.append(rango2)
    ...
    rangos.append(rangon)
    return rangos
```

Ilustración 25. Definición de reglas según el tipo de característica geométrica. Elaboración propia.

Para el caso en que el gen sea el material se crea la función materiales() que tiene dentro el registro de una matriz:

```
def materiales():
    m = [["Material1", prop11, prop12, ..., prop1j],
        ["Material2", prop21, prop22, ..., prop2j],
        ...
        ["Materiali", propi1, propi2, ..., propij]]

    #i: Número de materiales a analizar
    #j: Número de propiedades de cada material
    return m
```

Ilustración 26. Creación de la función de materiales según la tabla 2. Elaboración propia.

6.2.5 Generación de población

Al crearse los cromosomas de manera aleatoria y teniendo ya el rango de donde puede ser escogido cada uno de sus genes, se crea la función aleatorio():

```python
def aleatorio():
    rand = []
    rangos = parametrosIniciales()
    for i in range(len(rangos)):
        if(rangos[i][2])==0:
            rand.append(random.randint(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==1:
            rand.append(random.uniform(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==2:
            rand.append(random.choice(rangos[i][0]))
        elif(rangos[i][2])==3:
            rand.append(random.randint(rangos[i][0]/rangos[i][3],
            rangos[i][1]/rangos[i][3])*rangos[i][3])

    rand.append(int(random.randint(0,len(materiales())-1)))
    return rand
```

Ilustración 27. Creación del cromosoma una vez seleccionado el tipo de característica geométrica. Elaboración propia.

Cada uno de los cromosomas que serán creados podrían ser soluciones al problema de optimización. Entonces, la población deberá almacenar el número de “soluciones” que se haya indicado en la población definida en los parámetros iniciales de diseño.

```python
def aleatorioAppend():
    solutions = []
    for s in range(pobinit): #Aca se debe definir extremos de cada variable
        solutions.append(aleatorio())

    solutions = np.array(solutions)
    return solutions
```

Ilustración 28. Creación de la población a partir de los cromosomas creados aleatoriamente. Elaboración propia.

6.2.6 Fórmulas paramétricas

En esta parte se definen la función propia del diseño del elemento mecánico deseada. La función que se desarrolle tiene por objetivo la transformación de datos de entrada en datos de salida. No obstante, los valores que esta función va a retornar son los datos de entrada, salida y los requerimientos; esto debido a la posterior evaluación de los datos mediante un filtro.

```python
def diseno(solution):
    req = requerimientos()
    material = materiales()
    parinit = parametrosIniciales()

    #Fórmulas paramétricas
    ...
    ...
    ...

    #Datos de salida
    output = [op1, op2, ..., opj]
    #j es el numero de outputs
    input = [ip1, ip2, ..., ipi]
    #i es el numero de inputs

    return  input, req, output
```

Ilustración 29. Algoritmo base para el desarrollo de las fórmulas paramétricas. Elaboración propia.

6.2.7 Desarrollo de funciones fitness

Ya que se comprobó en el capítulo anterior que todas las funciones fitness eran errores normalizados, entonces, cada una de las funciones que se cree tendrá que hacer uso de las funciones paramétricas y extraer los outputs y requerimientos asociados entre sí (recordar que la función diseño() retorna una matriz donde el segundo y tercer dato son los vectores de requerimiento y outputs respectivamente.

```
194  def fiti(inputs):
195      datos = diseno(inputs)
196      reqi = datos[1][i] #requerimiento asociado a funcion fitness i
197      opj = datos[2][j] #output asociado al requerimiento i
198      ei = abs((opj-reqi)/reqi) #error absoluto calculado
199
200      return ei
```

Ilustración 30. Función fitness "i" asociada a un requerimiento. Elaboración propia.

También, se usará el producto de funciones fitness para hallar la función fitness global. En ese sentido, se creará un vector para el almacenamiento de las funciones fitness y otro para el valor de la potencia a la que estará elevado en la productoria.

```
216  def fitness(inputs):
217      fitness = []
218      fitness.append(fit1(inputs))
219      fitness.append(fit2(inputs))
220      ...
221      fitness.append(fiti(inputs))
222
223      w=[w1, w2, ..., wi]
224
225      fit=1
226      for i in range(len(fitness)):
227          fit = fit*fitness[i]**w[i]
228
229      return fit
```

Ilustración 31. Cálculo de fitness global a partir de las otras funciones fitness calculadas. Elaboración propia.

6.2.8 Registro del error permitido

Aquellas funciones fitness que tengan un error permitido deberán ser registrados y puestos en el mismo orden en que las funciones han sido asignadas. Por otro lado, en el caso de las funciones que no tengan un valor de error deberán ser creadas al final, esto para que la condición de parada no las tome en cuenta.

```
#Error máximo permitido
def errorPermitido():
    e = [e1,e2,...,ej]
    #j es el número de funciones
    # fitness con error máximo permitido.
    return e
```

Ilustración 32. Registro del error máximo permitido por cada requerimiento. Elaboración propia.

6.2.9 Filtro de población

Con la población inicial creada y las funciones paramétricas establecidas, aún es necesario asegurarse que los datos ingresados tengan coherencia, esto con la finalidad de evitar que el algoritmo converja en soluciones inexistentes o físicamente imposibles de crear. Además, para asegurar que la solución no converja a un solo valor, se debe evitar que existan soluciones iguales y que de ser así se vuelva a buscar otra potencial solución.

```
#Filtro de soluciones
def filterSolution(solutions):
    bool =[]
    for s in range(pobinit):
        filtro= diseno(solutions[s])
        filtro = np.array(filtro)
        for i in range(len(solutions[s])):
            solutions[s][0] = filtro[0][i]

        for j in range(s):
            bool1 = all(solutions[s] == solutions[j])
            if bool1:
                bool.append([bool1,s])
                for i in range(len(aleatorio())-1):
                    solutions[s][i] = aleatorio()[i]

                filtro= diseno(solutions[s])
                for i in range(len(solutions[s])):
                    solutions[s][0] = filtro[0][i]
    return solutions
```

Ilustración 33. Filtro de posibles soluciones previas al proceso de algoritmo genético. Elaboración propia.

6.2.10 Selección

Este es uno de los operadores dentro del algoritmo genético donde se busca evaluar a la población en una generación específica y rescatar de ellos los mejores candidatos a resolver el problema de optimización. Para determinar quién se queda o no se usa el valor obtenido en la función fitness global.

```
#Selección
def selection(solutions):
    rankedsolutions = []
    for s in solutions:
        rankedsolutions.append(fitness(s),
                               fit1(s),
                               fit2(s),
                               ...,
                               fiti(s))

    rankedsolutions.sort()
    bestsolutions = rankedsolutions[:int(bestpob)]

    return bestsolutions
```

Ilustración 34. Operador de selección del algoritmo genético. Elaboración propia.

6.2.11 Condición de parada

Un algoritmo genético puede concluir por un número máximo de generaciones o por que alcanzó los valores esperados. Para el primer caso, este valor ya ha sido asignado al momento de registrar los parámetros básicos del algoritmo. En caso del segundo, esto se deberá definir usando los valores de error permitido y comparándolos con el valor máximo de cada función fitness evaluada. Necesariamente, para que el algoritmo se detenga, todos los valores deberán ser inferiores.

```
296   def stopCondition(bestsolutions):
297       npbestsol = np.array(bestsolutions)
298       e = errorPermitido()
299       det = 1
300       for i in range(len(e)):
301           det = det*(max(npbestsol[:,i])<e[i])
302
303       if det:
304           print("Se interrumpio algoritmo: ")
305           print(*bestsolutions, sep = "\n")
306
307           return True
```

Ilustración 35. Función de evaluación de fin de programa. Elaboración propia.

6.2.12 Cruce

Del proceso de selección se obtuvo un conjunto de soluciones que no solo llevan consigue los genes iniciales, sino que también se han incluido los valores de las funciones fitness asociadas. En ese sentido, se deberá solo usar los genes iniciales para el proceso de cruce. Para el caso de este algoritmo se está usando la operación de cruce de un solo corte.

```
312   #Cruce
313   def crossOver(bestsolutions):
314       f = i+1 #i es el numero de funciones fitness
315       #se suma uno debido al espacio de la funcion fitness global
316       crosSolution = []
317       for i in range(int(bestpob/2)):
318           a = bestsolutions[2*i][f:len(bestsolutions[2*i])]
319           b = bestsolutions[2*i+1][f:len(bestsolutions[2*i+1])]
320           separ = random.randint(1,len(a)-1)
321           sepran = random.random()
322
323           if p_cruce < sepran:
324               x = []
325               x.append(b[:separ])
326               x[0] = x[0] + a[separ:]
327               y = []
328               y.append(a[:separ])
329               y[0] =y[0] + b[separ:]
330               crosSolution.append(x[0])
331               crosSolution.append(y[0])
332           else:
333               crosSolution.append(a)
334               crosSolution.append(b)
335
336       return crosSolution
337
```

Ilustración 36. Operador de cruce del algoritmo genético. Elaboración propia.

6.2.13 Mutación

El operador de mutación usará los valores obtenidos en la operación de cruce y mediante una probabilidad modificará uno de los genes de manera aleatoria. No es necesario filtrar el valor en caso de tener una solución incoherente, ya que esto se hará en la generación siguiente.

```
340  #Mutación
341  def mutate(crosSolution):
342      mutSolution = []
343      for i in crosSolution:
344          mutran = random.random()
345          mut = random.randint(0,len(crosSolution[0])-1)
346
347          if mutran > p_mutacion:
348              varmut = aleatorio()[mut]
349              j=[]
350              j = i[:mut]
351              j = np.insert(j,len(j),varmut)
352              j = np.concatenate([j,i[mut+1:]])
353              mutSolution.append(j)
354          else:
355              mutSolution.append(i)
356
357      return mutSolution
```

Ilustración 37. Operador de mutación del algoritmo genético. Elaboración propia.

6.2.14 Creación de nueva generación

En esta última etapa se revisarán las soluciones obtenidas después del operador de selección y mutación. Estos dos grupos de soluciones serán evaluados nuevamente mediante el fitness y se escogerán tantos como población inicial hubo al inicio. De esta manera se asegura rescatar lo mejor del proceso así como mantener el número de soluciones por generación.

Terminado este proceso se tendrá que volver al paso de filtro de población así hasta que se active la condición de parada o en su defecto se llegue al número de generación máxima permitida.

```python
#Nueva generación
def newSolution(mutSolution, bestSolution):
    newGen = []
    f = i+1 #i es el numero de funciones fitness
    #se suma uno debido al espacio de la funcion fitness global
    for i in mutSolution:
        newGen.append(np.asarray(i))

    for i in bestSolution:
        newGen.append(np.asarray(i[5:10]))
    newGen = np.array(newGen)

    newGen2 = []
    n = 0
    for s in newGen:
        n = n+1
        newGen2.append((fitness(s),
                        fit1(s),
                        fit2(s),
                            ...,
                        fiti(s),
                            s))

    newGen2.sort()
    newGen2 = np.array(newGen2)
    solutions = newGen2[:pobinit,f:]

    return solutions
```

Ilustración 38. Creación de nuevas soluciones para la siguiente generación. Elaboración propia.

6.2.15 Función principal

A partir de la función de filtro de población, todas las funciones antes creadas deberán estar descritas y conectadas en una función principal que internamente tiene un bucle de generaciones.

```python
#Main
solutions = initialPopulation()
for z in range(GEN): # N° generaciones

    solutions = filterSolution(solutions)

    bestsolutions = selection(solutions)

    graph(bestsolutions)

    if stopCondition(bestsolutions) == True:
        break

    crosSolution = crossOver(bestsolutions)

    mutSolution = mutate(crosSolution)

    solutions = newSolution(mutSolution,bestsolutions)

plt.waitforbuttonpress()
```

Ilustración 39. Función principal donde se ejecutan todas las funciones del algoritmo generalizado. Elaboración propia.

Capítulo 7 ESTUDIO DE CASO: RESORTE HELICOIDAL DE SECCIÓN CIRCULAR

7.1 Introducción

Para el estudio caso se tomará como pieza mecánica de ejemplo a un resorte helicoidal de sección circular que trabaje a compresión. Este tipo de resortes es muy común a nivel industrial y comercial. Los podemos encontrar en los lapiceros que con un botón esconden o sacan la punta de la tinta, en la suspensión de locomotoras, como base de amortiguación en edificaciones, así como en la suspensión automotriz.

El resorte, como elemento mecánico, tiene como requerimientos principales la rigidez y la altura instalada. Respecto a la rigidez, esta se mide en kg/mm e indica cada cuántos kilogramos el resorte pierde una altura de 1mm (si la rigidez es de 5kg/mm significa que por cada 5kg el resorte descenderá 1mm). Se entiende que mientras mayor sea la rigidez, el resorte tendrá mayor resistencia a la deformación. Por otro lado, la altura instalada indica la posición de trabajo unitaria a la que el resorte soportará determinada carga (a una altura de 100mm el resorte deberá cargar 200kg). Esta altura es la necesaria para que la máquina o mecanismo trabaje en condiciones normales.

Para el caso se consideran los siguientes requerimientos:

- Rigidez: 2.5 kg/mm
- Altura instalada: 280mm a una fuerza de 260kg
- Trabaja dentro de una cavidad de 300mm.
- La instalación no permite que el resorte tenga una altura mayor de 500mm.
- No se considera la temperatura como factor influyente.

En base a ello, se procederá con el desarrollo general, desde la extracción del conocimiento, hasta la entrega de los parámetros óptimos.

7.2 Desarrollo de mecanismo

7.2.1 Extracción de conocimiento

a. Conocimiento estándar

Se trabajará en función a la norma ASTM para revisar los materiales. En ella encontramos tres tipos: el A227 el cual es la especificación estándar para alambres de acero para conformado en frío para resortes mecánicos; el A313 que es la especificación estándar para alambre de acero inoxidable para resortes; y el A228 que es la especificación estándar para alambre de acero de calidad de cuerda de piano (usado para resortes de requerimiento de alto esfuerzo). De estos usaremos sus propiedades de esfuerzo de rotura, densidad volumétrica y módulo de elasticidad. A su vez, el libro “Diseño en ingeniería mecánica de Shigley” en su octava edición en su sección 10-6 “Materiales para resortes”; este nos brindará información adicional de las propiedades.

b. Conocimiento procedimental

Se extrae la información del libro “Diseño en ingeniería mecánica de Shigley” en su octava edición en su sección 10-4 “Resortes de compresión”; además de la norma SAE HS 795 la cual se encuentra dentro del “Spring Design Manual”. En ellos se encuentran el flujo y las fórmulas para el diseño de resortes de compresión que nos servirán como inputs para el siguiente capítulo.

Con respecto al conocimiento operacional, para el caso se usará un modelo de optimización de algoritmos genéticos el cual recibirá las fórmulas propuestas y optimizará las variables de salida entregando una lista de posibles soluciones que deberán ser escogidas basados en el conocimiento del diseño y requerimientos.

c. Conocimiento de juicio

Para el caso, se usarán las recomendaciones propias de la norma SAE HS 795 además de las buenas prácticas obtenidas de la industria especializada en diseño y producción de resortes helicoidales de sección circular. También se considerará a la norma ISO 9001:2015 en su sección 8.3 “Diseño y desarrollo de los productos y servicios”.

d. Conocimiento de control

Tomando esto como referencia, se elabora el cuadro de conocimiento para el diseño de un resorte cilíndrico helicoidal de sección circular.

Conocimiento de Control		
Uso correcto de normas técnicas ASTM y SAE; además de identificar las buenas prácticas en diseño de resortes propias de la fabricación de estos por parte de industria especializada en resortes.		
Conocimiento Estándar	Conocimiento Procedimental	Conocimiento de Juicio
- ASTM A227 - ASTM A228 - ASTM A313 - Shigley's 8va Edición 10-6	- SAE HS 795: Método de diseño - Shigley's 8va Edición 10-4	- SAE HS 795: Recomendaciones de diseño - *Know-how* de diseño en industria especializada en resortes - ISO 9001:2015 Sección 8.3

Tabla 3. Tipos de conocimiento aplicado al diseño de un resorte helicoidal de sección circular. Elaboración propia

7.2.2 Reglas de diseño

a. Reglas analíticas-numéricas

Para un resorte cilíndrico helicoidal las dimensiones principales serán el diámetro de alambre, diámetro externo, número de vueltas totales y longitud libre del resorte.

$$Input \rightarrow d, D_{ext}, N_t, L_o$$

Para el cálculo de los esfuerzos y el peso del material se trabajará con las fórmulas:

$$D_m = D_{ext} - d$$

$$C = \frac{D_m}{d}$$

$$K_B = \frac{4C + 2}{4C - 3}$$

$$\tau_{trab} = K_s \frac{8FD_m}{\pi d^3}$$

$$LDA = \pi D_m N_t$$

$$w = (\frac{\pi d^2}{4})(LDA)\rho_{vol}$$

Para el caso del requerimiento de fuerza y rigidez, se deben usar las fórmulas:

$$K = \frac{d^4 G}{8{D_m}^3 N_a}$$

$$F_{ins} = K(L_o - L_{ins})$$

Como complemento a las fórmulas se usará una tabla donde, dependiendo de los tipos de terminales que tiene el resorte, ciertos parámetros cambiarán su valor.

	Tipo de extremo			
Término	Plano	Plano y esmerilado	A escuadra y cerrado	A escuadra y esmerilado
Espiras activas, N_a	N_t	N_t-1	N_t-2	N_t-2
Longitud bloqueo, L_{bloq}	$d(N_t + 1)$	dN_t	$d(N_t + 1)$	dN_t

Tabla 4. Tipos de extremos de resortes. Budynas, s.f.

Basándonos en esta tabla, se considera que el resorte a desarrollar tiene los extremos a escuadra y cerrados.

b. Reglas heurísticas

Las condiciones de trabajo deberán considerar:

- Lugar de instalación: Si trabaja con un eje interno, las tolerancias de diseño deben asegurar que el diámetro interno del resorte sea mayor al diámetro del eje. Por el contrario, si trabajase dentro de una cavidad, entonces el diámetro externo será menor al diámetro de la cavidad.

$$Eje \rightarrow D_{int} > D_{eje}$$

$$Cavidad \rightarrow D_{ext} < D_{cavidad}$$

- Temperatura: Si la temperatura de trabajo es menor a 120°C, es posible usar los materiales A227 y A229.

$$T°_{trabajo} > 120°C \rightarrow A313$$

c. Reglas empíricas

Los datos de materiales a extraer serán la densidad volumétrica y módulo de corte de los tres materiales a revisar: A228, A313 y A227 todos de la norma

ASTM. Además, para el cálculo del esfuerzo de rotura, este será consecuencia de una fórmula que depende del diámetro del alambre y dos constantes: A y m. Estos se mostrarán en una tabla para identificar cuál de estos es el óptimo en base a las reglas heurísticas.

Materiales	A	m	Densidad volumétrica (kg/m3)	Módulo de Corte
A227	1.783	0.19	7861.09	80.7
A228	2.211	0.145	7861.09	82.7
A313	1.867	0.146	7916.45	69

Tabla 5. Propiedades de materiales aplicado al diseño de un resorte helicoidal de sección circular. Elaboración propia.

d. Reglas de restricción legal

Basándonos en las indicaciones de la ISO 9001: 2015 Sección 8.3, los requerimientos técnicos solicitados son que para una altura instalada del resorte le corresponde una fuerza instalada; además que la rigidez axial del resorte tenga el mínimo error respecto a lo solicitado.

$$L_{ins} \rightarrow F_{ins}$$

$$e_K \cong 0$$

También, conforme a las recomendaciones técnicas, se planea trabajar fórmulas de esfuerzos permisibles que deberán ser menores al 70% del esfuerzo de fluencia (*yield strength)* del material; y el coeficiente de arrollamiento deberá estar en 4 y 12 para asegurar trabajar con los menores esfuerzos recomendados.

$$\tau_{perm} \leq 0.7\, S_y$$

$$4 \leq C \leq 12$$

Los datos de salida del proceso de diseño serán las dimensiones óptimas del resorte que para el caso se traducen en diámetro de alambre, diámetro externo, número de vueltas totales y longitud libre del resorte.

$$Output\ 1 \rightarrow d_{opt}, D_{ext_{opt}}, N_{t_{opt}}, L_{o_{opt}}$$

$$Output\ 2 \rightarrow Material\ óptimo$$

Con estos valores se logrará minimizar los esfuerzos de trabajo y el peso del resorte.

7.2.3 Flujo de diseño

De la información proporcionada por el conocimiento y las reglas de diseño generadas, se desarrolla el flujo para el caso de estudio de un resorte helicoidal de sección circular.

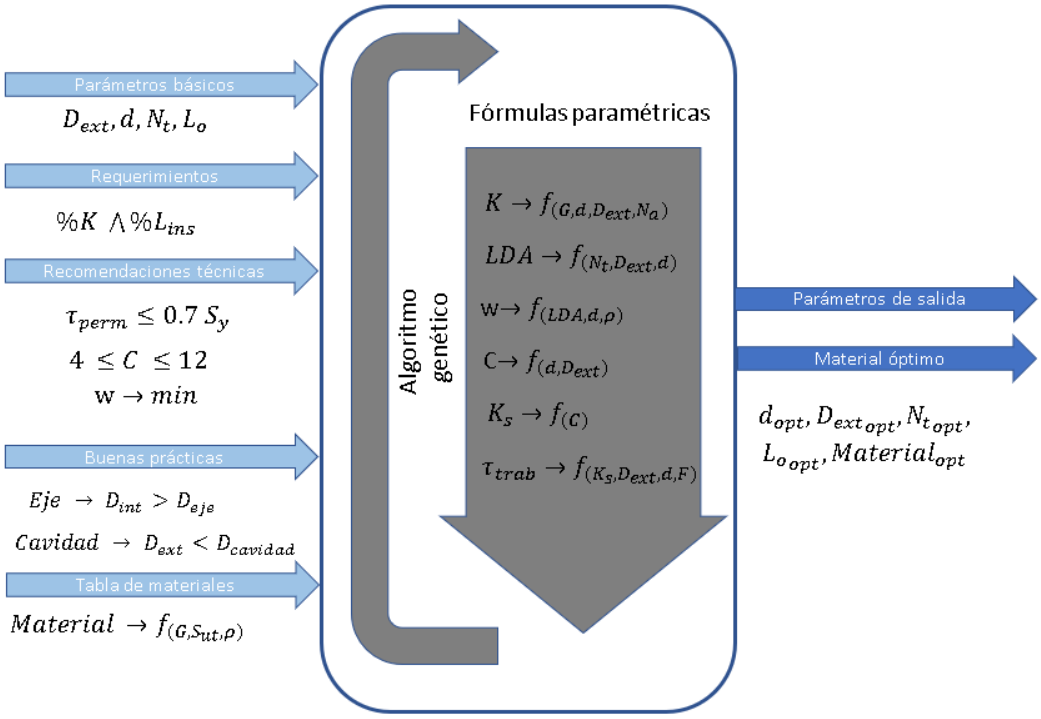

Ilustración 40. Flujo de diseño de un resorte cilíndrico helicoidal de sección circular. Elaboración propia.

7.2.4 Desarrollo de cromosoma

Las dimensiones serán las características geométricas que se extrajeron en la sección anterior: diámetro de alambre (d), diámetro externo (D_{ext}), número de **vueltas** (N_t) **y longitud** libre (L_o). Por otro lado, para el material se extrajeron las propiedades a utilizar: factor A, factor m, densidad volumétrica y módulo de corte.

De esta manera, el cromosoma queda representado según la ilustración 20:

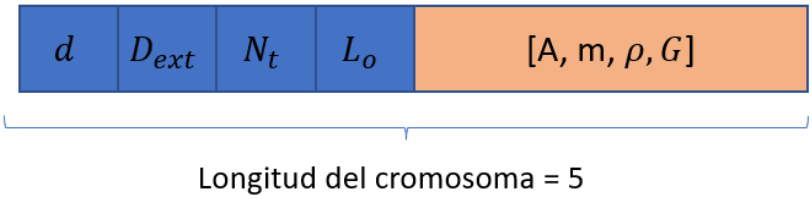

Ilustración 41. Longitud del cromosoma para un resorte helicoidal de sección circular. Elaboración propia

También será necesario considerar el tipo de variación que presenta cada gen con carácter dimensional dentro del cromosoma.

- Diámetro de alambre (d)

Este proviene de un alambre trefilado y las medidas que se encuentran disponibles en el sistema internacional son cada 0.5mm (Ejemplo: 10, 10.5, 11, 11.5, 12…)

- Diámetro externo (D_{ext})

Si bien es cierto, este puede tener variaciones próximas a las centésimas de milímetro, para el caso se trabajará con variación de 1mm debido a que los resultados entre dos valores muy cercanos no tendrían diferencia significativa.

- Número de vueltas (N_t)

Este valor es usualmente una fracción y en el diseño de resortes se contempla una variación mínima de un octavo de vuelta. (Ejemplo: 5, 12.5, 10.75, 8.125)

- Longitud libre (L_o)

Similar al diámetro externo, la variación será de 1mm para tener valores con diferencias significativas.

7.2.5 Desarrollo de función fitness

Las restricciones encontradas se limitan solo a la altura libre y al diámetro exterior máximos. En ese sentido, de acuerdo a criterio del diseñador se deberá escoger un rango para cada uno de los genes con cierto criterio:

$$10 \leq d \leq 25$$

$$100 \leq D_{ext} \leq 300$$

$$3 \leq N_t \leq 10$$

$$100 \leq L_o \leq 500$$

Teniendo estos datos como base, se procede a revisar los requerimientos.

- Requerimiento de esfuerzo

Para este caso el esfuerzo de fluencia va a ser variado ya que depende del material que se escogerá, el cual, a su vez, tiene como datos los parámetros A y m para el cálculo en función al diámetro del alambre. Por otro lado, para el cálculo de resortes a compresión se considera un factor de corrección de esfuerzo de 0.7.

$$f_\tau = \frac{\tau}{0.7 * (\frac{A}{d^m})}$$

Queda entendido que los parámetros d, A y m serán asignados de manera aleatoria (A y m asociados a un material) al momento de generarse los cromosomas.

- Requerimiento de peso

La función fitness de peso requiere el dato del peso máximo que se encontrará en el rango de valores definidos. Entonces, se deben escoger los valores máximos de las dimensiones y el material con la mayor densidad.

De la fórmula:

$$w = (\frac{\pi d^2}{4})(LDA)\rho_{vol}$$

Se asignan los valores máximos

$$w_{max} = (\frac{\pi {d_{max}}^2}{4})(\pi D_{m_{max}} N_{t_{max}})\rho_{max}$$

Entonces, la función fitness del peso quedará representada por:

$$f_w = \frac{w}{(\frac{\pi {d_{max}}^2}{4})(\pi D_{m_{max}} N_{t_{max}})\rho_{max}}$$

- Requerimiento de rigidez

Al ser un requerimiento particular del resorte será el error quien modele está función:

$$f_k = \frac{|K_{real} - K_{esperado}|}{K_{esperado}}$$

- Requerimiento de altura instalada

Al igual que la rigidez, será el error quien determine el comportamiento de la función:

$$f_l = \frac{\left|L_{o_{real}} - L_{o_{esperado}}\right|}{L_{o_{esperado}}}$$

Con todas las funciones fitness creadas ahora se necesita compilar todas dentro de la función global, para esto se trabajará con la función fitness global de multiplicación debido a que la precisión de cada uno de las funciones es diferente:

$$f_T = {f_\tau}^{w_1} * {f_w}^{w_2} * {f_k}^{w_3} * {f_l}^{w_4}$$

Para la asignación de los parámetros w_i se debe entender el comportamiento del elemento. Al ser particulares, los requerimientos de rigidez y altura instalada deberán tener mayor peso. Para poder encontrar los valores óptimos, es necesario ejecutar el algoritmo y revisar los parámetros ideales. Para el caso se encontró:

$$w_1 = 1, w_2 = 1, w_3 = 4, w_4 = 4$$

$$f_T = f_\tau * f_w * {f_k}^4 * {f_l}^4$$

7.2.6 Criterio de convergencia

Es necesario asignar un error máximo permitido para cada una de las funciones fitness:

- Error máximo permitido de requerimiento de rigidez: 4%
- Error máximo permitido de requerimiento de altura instalada: 4%
- Error máximo permitido de esfuerzo: 1

Para el caso del peso no se tiene un error por lo que solo se buscara su minimización.

Entonces, la condición de detención será:

$$f_\tau < 1 \; and \; f_k < 0.04 \; and \; f_l < 0.04$$

7.3 Desarrollo de algoritmo genético

Lo explicado en el capítulo 6, será plasmado tomando como ejemplo el resorte helicoidal como elemento mecánico:

7.3.1 Importación de librerías y parámetros de control

Las librerías se mantienen a excepción de “pandas” la cual se está incluyendo para mostrar los resultados (opcional). Por otro lado, se insertan las funciones usadas en otros subprogramas creados. También, se indican los parámetros iniciales del algoritmo genético.

```python
from cmath import pi
import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from diseno import parametrosIniciales, materiales
from fitness import fit1, fit2, fit3, fit4, fitness, errorPermitido
from diseno import diseno

#-----------------------------------------------------------------
#Parámetros de control del algoritmo genético
p_cruce = 0.8
p_mutacion = 0.7
pobinit = 200
bestpob = 0.7*pobinit
random.seed(123)
nval = 15
GEN = 1000
parinit = parametrosIniciales()
```

Ilustración 42. Librerías usadas en el caso de estudio. Elaboración propia.

7.3.2 Requerimientos

Para este caso se crea un vector de requerimientos teniendo internamente un vector que agrupa la fuerza y altura instalada, y el requerimiento de rigidez.

```python
def requerimientos():
    req = []
    #Altura y fuerza instalada
    req1 = [280, 260*9.81]
    #Rigidez
    req2 = 2.5*9.81
    req.append(req1)
    req.append(req2)
    return req
```

Ilustración 43. Requerimientos planteados en el caso de estudio. Elaboración propia.

7.3.3 Parámetros iniciales

Estos corresponden a las características geométricas del elemento, donde se define el rango y la intermitencia de los datos. En la sección 7.2.5 se reviso el rango y la intermitencia de los valores; ahora estos deberán ser introducidos en

el algoritmo. Para el caso del material, se usan los descritos en la sección 7.2.2.c con sus respectivas propiedades.

```python
def parametrosIniciales():
    rangos = []
    rangoal = [10,25,3,0.5]
    rangodext = [100,300,0,1]
    rangont = [3,10,3,0.125]
    rangolo = [100,500,0,1]

    rangos.append(rangoal)
    rangos.append(rangodext)
    rangos.append(rangont)
    rangos.append(rangolo)

    return rangos

def materiales():
    m = [["A227", 1783, 0.19, 7861.09, 80.7],
        ["A228", 2211, 0.145, 7861.09, 82.7],
        ["A313", 1867, 0.146, 7916.45, 69]]

        #Nombre del material,
        #A, m, Densidad(kg/m3),
        # Módulo de Corte (GPa)
    return m
```

Ilustración 44. Registro de datos de las características geométricas y selección de materiales. Elaboración propia.

7.3.4 Generación de población inicial

En esta parte del algoritmo no existe variación con lo descrito en el capítulo anterior. La creación de la población inicial dependerá de la aleatoriedad dentro de los rangos determinados para cada parámetro.

```python
#-------------------------------------------------------------------
#Creación de cromosoma
def aleatorio():
    rand = []
    rangos = parametrosIniciales()
    for i in range(len(rangos)):
        if(rangos[i][2])==0:
            rand.append(random.randint(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==1:
            rand.append(random.uniform(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==2:
            rand.append(random.choice(rangos[i][0]))
        elif(rangos[i][2])==3:
            rand.append(random.randint(int(rangos[i][0]/rangos[i][3]),
            int(rangos[i][1]/rangos[i][3]))*rangos[i][3])

    rand.append(int(random.randint(0,len(materiales())-1)))
    return rand

#-------------------------------------------------------------------
#Población inicial
def initialPopulation():
    solutions = []
    for s in range(pobinit): #Aca se debe definir extremos de cada variable
        solutions.append(aleatorio())
    solutions = np.array(solutions)
    return solutions

#-------------------------------------------------------------------
```

Ilustración 45. Generación de población inicial creada a partir de cromosomas generados aleatoriamente. Elaboración propia.

7.3.5 Fórmulas paramétricas

Esta sección es la que más variaciones tiene debido a que cualquier elemento presenta fórmulas de desarrollo distintas y distintos requerimientos a optimizar.

El resultado de esta función son los requerimientos, los datos de salida e inclusive los datos de entrada que podrían estar modificados en caso se encuentre alguna inconsistencia al momento de ejecutar las fórmulas.

```python
def diseno(a,dext,nt,lo,i):
    #Declarar condiciones iniciales
    req = requerimientos()
    material = materiales()
    ans = parametrosIniciales()
    li = req[0][0]
    fio = req[0][1]
    ko = req[1]

    #Fórmulas
    i = int(i)
    G = int(material[i][4]*1000) #MPa
    ty = material[i][1]/(a**material[i][2])
    dm = dext - a
    n = nt - 1.5
    k = G*a**4/(8*n*dm**3) #N/mm
    c = dm/a

    lm = lo - a
    p = lm/n
    xc = (p-a)*n
    fc = xc*k

    while fc < fio or lo < li:
        dext = random.randint(int(ans[1][0]/ans[1][3]),
                              int(ans[1][1]/ans[1][3]))*ans[1][3]
        nt = random.randint(int(ans[2][0]/ans[2][3]),
                            int(ans[2][1]/ans[2][3]))*ans[2][3]
        lo = random.randint(int(ans[3][0]/ans[3][3]),
                            int(ans[3][1]/ans[3][3]))*ans[3][3]
        dm = dext - a
        n = nt - 1.5
        c = dm/a
        k = G*a**4/(8*n*dm**3) #N/mm
        lm = lo - a
        p = lm/n
        xc = (p-a)*n
        fc = xc*k

    wc = (4*c-1)/(4*c-4)+0.615/c
    tc = 8*dm*fc/(pi*a**3)*wc
    lda = dm*pi*nt+200
    peso = lda/1000*pi/4*(a/1000)**2*material[i][3]

    return k,ko,lo,li,fc,fio,ty,tc,peso,a,dext,nt
```

Ilustración 46. Desarrollo de fórmulas paramétricas para el caso de estudio. Elaboración propia.

7.3.6 Funciones fitness y error permitido

Para el caso se cuenta con dos requerimientos y dos variables a optimizar: rigidez, altura instalada, esfuerzo y peso respectivamente. A su vez, todas las funciones asociadas se encuentren dentro de la función fitness global.

```python
def fit1(a,dext,nt,lo,i): #error en rigidez
    ans = diseno(a,dext,nt,lo,i)
    ko = ans[1]
    ek = abs((ans[0]-ko)/ko)

    return ek

def fit2(a,dext,nt,lo,i): #error en fuerza
    ans = diseno(a,dext,nt,lo,i)
    li = ans[3]
    fio = ans[5]
    fi = ans[0]*(ans[2] - li)
    fc = ans[4]
    ef = abs((fi-fio)/fio)

    return ef

def fit3(a,dext,nt,lo,i): #Esfuerzos minimos
    ans = diseno(a,dext,nt,lo,i)
    ty = ans[6] #MPa
    et = abs((ans[7])/ty)

    return et

def fit4(a,dext,nt,lo,i): #Peso mínimo
    ans = diseno(a,dext,nt,lo,i)
    parinit = parametrosIniciales()
    material = materiales()
    dmax = parinit[0][1]
    dextmax = parinit[1][1]
    nmax = parinit[2][1]
    ldamax = (dextmax-dmax)*pi*nmax+200
    pesomax = ldamax/1000*pi/4*(dmax/1000)**2*material[int(i)][3]
    ep = ans[8]/pesomax

    return ep

def fitness(a,dext,nt,lo,i):
    fitness = []
    fitness.append(fit1(a,dext,nt,lo,i))
    fitness.append(fit2(a,dext,nt,lo,i))
    fitness.append(fit3(a,dext,nt,lo,i))
    fitness.append(fit4(a,dext,nt,lo,i))

    w=[4,4,1,1]

    fit=1
    for i in range(len(fitness)):
        fit = fit*fitness[i]**w[i]

    return fit
```

Ilustración 47. Generación de funciones fitness en función a los requerimientos del caso de estudio. Elaboración propia.

También se incluyen los errores (en %) que debe tener el resultado de cada función fitness.

```
55  def errorPermitido():
56      e = [0.04,0.04]
57      #j es el número de funciones
58      # fitness con error máximo permitido.
59      return e
```

Ilustración 48. Errores asociados a los requerimientos particulares del caso de estudio. Elaboración propia.

7.3.7 Filtro de soluciones

Esta etapa no se tiene modificaciones respecto al algoritmo general.

```
77  #Filtro de soluciones
78  def filterSolution(solutions):
79      bool =[]
80      for s in range(pobinit):
81          filtro= diseno(solutions[s][0],
82                         solutions[s][1],
83                         solutions[s][2],
84                         solutions[s][3],
85                         solutions[s][4])
86          filtro = np.array(filtro)
87          solutions[s][0] = filtro[9]
88          solutions[s][1] = filtro[10]
89          solutions[s][2] = filtro[11]
90          solutions[s][3] = filtro[2]
91
92          for j in range(s):
93              bool1 = all(solutions[s] == solutions[j])
94              if bool1:
95                  bool.append([bool1,s])
96                  for i in range(len(aleatorio())-1):
97                      solutions[s][i] = aleatorio()[i]
98
99                  filtro= diseno(solutions[s][0],
100                                 solutions[s][1],
101                                 solutions[s][2],
102                                 solutions[s][3],
103                                 solutions[s][4])
104                 solutions[s][0] = filtro[9]
105                 solutions[s][1] = filtro[10]
106                 solutions[s][2] = filtro[11]
107                 solutions[s][3] = filtro[2]
108     return solutions
109
110 #-----------------------------------------------------
```

Ilustración 49. Filtro de soluciones previas al algoritmo genético. Elaboración propia.

7.3.8 Selección, Cruce y Mutación

De los tres operadores básicos del algoritmo genético, solo el de selección se ve ligeramente alterado ya que es necesario registrar manualmente todas las funciones fitness que participarán. Si es necesario trabajar otra forma de operadores (tales como la ruleta para selección o de doble punto para el cruce) se puede incluir sin problema alguno.

```python
#Selección
def selection(solutions):
    rankedsolutions = []
    for s in solutions:
        rankedsolutions.append((fitness(s[0],s[1],s[2],s[3],s[4]),
                                fit1(s[0],s[1],s[2],s[3],s[4]),
                                fit2(s[0],s[1],s[2],s[3],s[4]),
                                fit3(s[0],s[1],s[2],s[3],s[4]),
                                fit4(s[0],s[1],s[2],s[3],s[4]),
                                    s[0],s[1],s[2],s[3],s[4]))

    rankedsolutions.sort()
    bestsolutions = rankedsolutions[:int(bestpob)]

    return bestsolutions

#-----------------------------------------------------------------
```

Ilustración 50. Operador de selección del algoritmo genético aplicado al caso de estudio. Elaboración propia.

7.3.9 Condición de parada

La sección de condición de parada se mantendrá constante para cualquier elemento mecánico.

```python
#-----------------------------------------------------------
#Condición de parada
def stopCondition(bestsolutions):
    npbestsol = np.array(bestsolutions)
    e = errorPermitido()
    det = 1
    for i in range(len(e)):
        det = det*(max(npbestsol[:nval,i+1])<e[i])
    if det:
        solsinrep  = np.unique(bestsolutions,axis = 0)
        solsinrep = np.array(solsinrep)
        print("Se interrumpio algoritmo: ")
        print(*solsinrep[nval:], sep = "\n")
        showResults(solsinrep)
        return True
```

Ilustración 51. Condición de parada aplicado al caso de estudio. Elaboración propia.

7.3.10 Creación de nueva generación y función principal

De la misma manera que en la función de selección, la creación de nueva generación solo se tendrá que alterar para ingresar manualmente todas las funciones fitness con sus respectivos inputs. Para la función principal, está no se verá afectada.

```python
#Nueva generación
def newSolution(mutSolution, bestSolution):
    newGen = []
    for i in mutSolution:
        newGen.append(np.asarray(i))

    for i in bestSolution:
        newGen.append(np.asarray(i[5:10]))
    newGen = np.array(newGen)

    newGen2 = []
    n = 0
    for s in newGen:
        n = n+1
        newGen2.append((fitness(s[0],s[1],s[2],s[3],s[4]),
                        fit1(s[0],s[1],s[2],s[3],s[4]),
                        fit2(s[0],s[1],s[2],s[3],s[4]),
                        fit3(s[0],s[1],s[2],s[3],s[4]),
                        fit4(s[0],s[1],s[2],s[3],s[4]),
                            s[0],s[1],s[2],s[3],s[4]))

    newGen2.sort()
    newGen2 = np.array(newGen2)
    solutions = newGen2[:pobinit,5:]

    return solutions

#-------------------------------------------------------------------
```

Ilustración 52. Desarrollo de nueva población para próxima generación. Elaboración propia.

7.3.11 Mostrar resultados

Con la librería “pandas” se procede a filtrar los datos encontrados en función al error permitido y posteriormente a enviar los datos a una hoja de cálculo.

```python
# Mostrar resultados
def showResults(solutions):
    e = errorPermitido()
    df = pd.DataFrame({"Fitness Global": solutions[:,0],
                       "Fitness1": solutions[:,1],
                       "Fitness2": solutions[:,2],
                       "Fitness3": solutions[:,3],
                       "Fitness4": solutions[:,4],
                       "Alambre": solutions[:,5],
                       "D. externo": solutions[:,6],
                       "Nº espiras": solutions[:,7],
                       "Longitud libre": solutions[:,8],
                       "Tipo de material": solutions[:,9]
                        })
    df2 = df[(df['Fitness1']<e[0])&(df['Fitness2']<e[1])&(df['Fitness3']<1)]
    df2.to_csv("D:/Programas/EjemplosGA/Genetic-Algotihm/fitness2.csv")
#-------------------------------------------------------------------
```

Ilustración 53. Función de conversión de datos a dataframe y envío a hoja de cálculo. Elaboración propia.

7.4 Resultados

Al ejecutarse el algoritmo se ha obtenido una lista de soluciones la cual deberá ser analizada por el diseñador para determinar cuál de todas las opciones se adaptará mejor al trabajo.

Fitness Global	Fit 1	Fit 2	Fit 3	Fit 4	Alambre	D. exterior	N° espiras	Longitud libre	Tipo de material
5.65E-19	2.30E-03	2.15E-02	6.55E-01	1.45E-01	15	176	6.625	382	1
1.33E-18	2.30E-03	2.65E-02	6.66E-01	1.45E-01	15	176	6.625	387	1
3.29E-31	2.06E-04	2.06E-04	6.67E-01	1.53E-01	15.5	191	6	384	1
1.43E-24	2.06E-04	9.41E-03	6.65E-01	1.53E-01	15.5	191	6	383	1
1.71E-24	2.06E-04	9.82E-03	6.69E-01	1.53E-01	15.5	191	6	385	1
2.38E-23	2.06E-04	1.90E-02	6.63E-01	1.53E-01	15.5	191	6	382	1
2.63E-23	2.06E-04	1.94E-02	6.72E-01	1.53E-01	15.5	191	6	386	1
1.32E-22	2.06E-04	2.91E-02	6.74E-01	1.53E-01	15.5	191	6	387	1
3.87E-22	2.06E-04	3.83E-02	6.58E-01	1.53E-01	15.5	191	6	380	1
4.15E-22	2.06E-04	3.87E-02	6.76E-01	1.53E-01	15.5	191	6	388	1
8.66E-21	7.27E-03	2.27E-03	5.60E-01	2.07E-01	17	216	6	385	1
2.15E-25	1.08E-03	1.08E-03	2.97E-01	3.81E-01	19	210	9.375	384	1
2.06E-21	1.08E-03	1.07E-02	2.99E-01	3.81E-01	19	210	9.375	385	1
1.68E-20	1.08E-03	1.82E-02	2.95E-01	3.81E-01	19	210	9.375	382	1
2.69E-20	1.08E-03	2.03E-02	3.00E-01	3.81E-01	19	210	9.375	386	1
1.27E-19	1.08E-03	3.00E-02	3.01E-01	3.81E-01	19	210	9.375	387	1
3.00E-19	1.08E-03	3.74E-02	2.92E-01	3.81E-01	19	210	9.375	380	1
3.90E-19	1.08E-03	3.96E-02	3.03E-01	3.81E-01	19	210	9.375	388	1
7.20E-20	3.94E-02	6.16E-04	5.59E-01	3.73E-01	20.5	270	6	380	0
1.69E-22	9.06E-03	6.43E-04	3.79E-01	3.88E-01	20.5	270	6.25	383	1
3.62E-18	1.45E-02	4.99E-03	2.64E-01	4.98E-01	21	247	8.5	382	1
2.32E-19	2.71E-03	1.24E-02	2.57E-01	7.14E-01	23	266	9.5	385	0
1.46E-20	4.05E-03	4.05E-03	2.78E-01	7.26E-01	23.5	284	8.625	384	0

Tabla 6. Base de datos exportada desde el programa a una hoja de cálculo. Elaboración propia.

CONCLUSIONES

- Se describió todos los tipos de conocimiento que deberán estar presentes al momento de la extracción del conocimiento global.
- Se desarrolló una regla de diseño en función a cada conocimiento. Estas permiten que los conceptos teóricos sean trasladados a conceptos matemáticos a ser almacenadas en un programa.
- Con las reglas se armó un flujo de entrada y salida incluyendo el proceso de transformación de datos (fórmulas paramétricas).
- Se implementó una herramienta que hace uso del conocimiento existente de diseño de productos mecánicos para encontrar los parámetros óptimos que satisfagan sus requerimientos.
- El cromosoma, como parte básica del algoritmo genético, fue diseñado en función a los conceptos mecánicos de características dimensionales y material asignado. Son estos valores los que estarán almacenados dentro del cromosoma y los que definirán las soluciones que se obtendrán.
- Los requerimientos fueron trasladados a conceptos de algoritmo genético tal como lo es la función fitness. Al ser más de un requerimiento, se crearon varias funciones fitness y una función global que las agrupa. Para ello, cada una de las funciones debe estar definida en un dominio y rango; además de normalizada.
- Debido a los diversos requerimientos que se puede considerar, la solución se transforma en un conjunto de soluciones que posteriormente deberá ser filtrada por el diseñador.
- El criterio de convergencia a definir es fundamental para encontrar los valores óptimos dentro del algoritmo. Por ende, los errores máximos permitidos asociados a cada requerimiento deberán ser ingresados por el diseñador en base a su conocimiento en el diseño del elemento.
- Se desarrolló la base del algoritmo genético siendo esta la unión entre el conocimiento extraído de diseño mecánico convertido a datos de entrada.
- La implementación de un algoritmo genético generalizado permite a los diseñadores tomarlo como base para el desarrollo de cualquier elemento mecánico y adecuarlo a los parámetros, fórmulas y requerimientos solicitados.

RECOMENDACIONES

- El uso del código generalizado expuesto en esta investigación se encuentra limitado al diseño de elementos **mecánicos**. Si se necesita usar en otro tipo de elemento (eléctrico, electrónico, etc.), es posible hacer siempre y cuando se tengan en cuenta los nuevos datos de entrada y los requerimientos generales sean modificados conforme a las características que se busca del elemento.
- Combinar el flujo de algoritmo genético planteado en esta tesis, con el uso del método de elementos finitos es una de las mejores pendientes que tiene esta investigación. De esta manera los cálculos numéricos son más precisos y se puede adaptar a formas más complejas que se extienden fuera de las fórmulas paramétricas.
- Con la información mostrada, es posible armar una base de datos con los datos de entrada y salida de cada diseño que se elabore para una pieza mecánica en específico. Así, esta información puede ser procesada por una inteligencia artificial, como una red neuronal, que permita encontrar formas de procesar la información a mayor velocidad o encuentre mejoras dentro del algoritmo.
- Cada cierto tiempo aparecen nuevos algoritmos de optimización; en ese sentido, se deberá comprobar si existe algún otro algoritmo que pueda mejorar la herramienta de optimización de manera que siempre se esté actualizando. En esta investigación, el algoritmo genético es parte de la herramienta, y siempre será discutible el uso de alguna otra forma de optimización.

REFERENCIAS BIBLIOGRÁFICAS

- Budynas, R. G. (s. f.). *Diseño en ingeniería mecánica de Shigley*. 1092.
- Calkins, D. E., Egging, N., & Scholz, C. (s. f.). *Knowledge-Based Engineering (KBE) Design Methodology at the Undergraduate and Graduate Levels*. 18.
- Chapman, C. B., & Pinfold, M. (1999). Design engineering—A need to rethink the solution using knowledge based engineering. *Knowledge-Based Systems*, *12*(5-6), 257-267. https://doi.org/10.1016/S0950-7051(99)00013-1
- Gembarski, P. C. (2020). THREE WAYS OF INTEGRATING COMPUTER-AIDED DESIGN AND KNOWLEDGE-BASED ENGINEERING. *Proceedings of the Design Society: DESIGN Conference*, *1*, 1255-1264. https://doi.org/10.1017/dsd.2020.313
- Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press. https://doi.org/10.7551/mitpress/1090.001.0001
- Huabo He, Ke Wang, & Yimin Deng. (2011). Research of conceptual design of electromechanical products based on KBE. *2011 Second International Conference on Mechanic Automation and Control Engineering*, 558-561. https://doi.org/10.1109/MACE.2011.5986985
- Kar, A. K. (2016). Bio inspired computing – A review of algorithms and scope of applications. *Expert Systems with Applications*, *59*, 20-32. https://doi.org/10.1016/j.eswa.2016.04.018
- Koza, J. R., & Poli, R. (2005). Genetic programming. En *Search methodologies* (pp. 127-164). Springer.
- Lampinen, J. (2003). Cam shape optimisation by genetic algorithm. *Computer-Aided Design*, *35*(8), 727-737. https://doi.org/10.1016/S0010-4485(03)00004-6

- Mayda, M. (2017). An Efficient Simulation-Based Search Method for Reliability-Based Robust Design Optimization of Mechanical Components. *Mechanics*, *23*(5), 696-702. https://doi.org/10.5755/j01.mech.23.5.15745
- Melanie, M. (s. f.). *An Introduction to Genetic Algorithms*. 162.
- Mott, R. L. (s. f.). *Diseño de elementos de máquinas*. 946.
- Norton, R. L. (s. f.). *Diseño de máquinas*. 1057.
- Sivanandam, S. N., & Deepa, S. N. (2007). *Introduction to genetic algorithms*. Springer.
- Sokolowski, J., & Zolesio, J.-P. (1992). Introduction to shape optimization. En J. Sokolowski & J.-P. Zolesio, *Introduction to Shape Optimization* (Vol. 16, pp. 5-12). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-58106-9_1
- Wheeler, K. (s. f.). *Methodological Support for Knowledge Based Engineering Application Development*. 92.
- Whiteley, J. (2017). *Finite Element Methods*. Springer International Publishing. https://doi.org/10.1007/978-3-319-49971-0
- Wirsansky, E. (2020). *Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems*. Packt Publishing Ltd.
- Yuping Wang & Yiu-Wing Leung. (2000). Multiobjective programming using uniform design and genetic algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, *30*(3), 293-304. https://doi.org/10.1109/5326.885111

ANEXOS

Anexo A: Código de algoritmo en python

```
from cmath import pi
import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from diseno import parametrosIniciales, materiales
from fitness import fit1, fit2, fit3, fit4, fitness, errorPermitido
from diseno import diseno

#---------------------------------------------------------------------
#Parámetros de control del algoritmo genético
p_cruce = 0.8
p_mutacion = 0.7
pobinit = 200
bestpob = 0.7*pobinit
random.seed(123)
nval = 15
GEN = 1000
parinit = parametrosIniciales()
#numero de items que deben estar por debajo de las especificaciones para
detener el algoritmo

#---------------------------------------------------------------------
#Gráfica
plt.ion()
fig, ax = plt.subplots()
x, y = [],[]
sc = ax.scatter(x,y)
plt.xlim(0,0.2)
plt.ylim(0,2)
ax.plot(x,y,"r")
plt.draw()

#Función de actualización de gráfica
def graph(bestsolutions):
    npbestsol = np.array(bestsolutions)
    x = npbestsol[:,1]
    y = npbestsol[:,2]
    plt.xlim(0,max(x))
    plt.ylim(0,max(y))
    texto = "Generación: " + str(z)
    plt.xlabel(texto)
    sc.set_offsets(np.c_[x,y])
    fig.canvas.draw_idle()
```

```
    plt.pause(0.1)



#---------------------------------------------------------------------
#Creación de cromosoma
def aleatorio():
    rand = []
    rangos = parametrosIniciales()
    for i in range(len(rangos)):
        if(rangos[i][2])==0:
            rand.append(random.randint(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==1:
            rand.append(random.uniform(rangos[i][0],rangos[i][1]))
        elif(rangos[i][2])==2:
            rand.append(random.choice(rangos[i][0]))
        elif(rangos[i][2])==3:
            rand.append(random.randint(int(rangos[i][0]/rangos[i][3]),
            int(rangos[i][1]/rangos[i][3]))*rangos[i][3])

    rand.append(int(random.randint(0,len(materiales())-1)))
    return rand

#---------------------------------------------------------------------
#Población inicial
def initialPopulation():
    solutions = []
    for s in range(pobinit): #Aca se debe definir extremos de cada
variable
        solutions.append(aleatorio())
    solutions = np.array(solutions)
    return solutions

#---------------------------------------------------------------------
#Filtro de soluciones
def filterSolution(solutions):
    bool =[]
    for s in range(pobinit):
        filtro= diseno(solutions[s][0],
                       solutions[s][1],
                       solutions[s][2],
                       solutions[s][3],
                       solutions[s][4])
        filtro = np.array(filtro)
        solutions[s][0] = filtro[9]
        solutions[s][1] = filtro[10]
        solutions[s][2] = filtro[11]
        solutions[s][3] = filtro[2]
```

```
        for j in range(s):
            bool1 = all(solutions[s] == solutions[j])
            if bool1:
                bool.append([bool1,s])
                for i in range(len(aleatorio())-1):
                    solutions[s][i] = aleatorio()[i]

                filtro= diseno(solutions[s][0],
                               solutions[s][1],
                               solutions[s][2],
                               solutions[s][3],
                               solutions[s][4])
                solutions[s][0] = filtro[9]
                solutions[s][1] = filtro[10]
                solutions[s][2] = filtro[11]
                solutions[s][3] = filtro[2]
    return solutions

#---------------------------------------------------------------------
#Selección
def selection(solutions):
    rankedsolutions = []
    for s in solutions:
        rankedsolutions.append((fitness(s[0],s[1],s[2],s[3],s[4]),
                                fit1(s[0],s[1],s[2],s[3],s[4]),
                                fit2(s[0],s[1],s[2],s[3],s[4]),
                                fit3(s[0],s[1],s[2],s[3],s[4]),
                                fit4(s[0],s[1],s[2],s[3],s[4]),
                                     s[0],s[1],s[2],s[3],s[4]))

    rankedsolutions.sort()
    bestsolutions = rankedsolutions[:int(bestpob)]

    return bestsolutions

#---------------------------------------------------------------------
#Condición de parada
def stopCondition(bestsolutions):
    npbestsol = np.array(bestsolutions)
    e = errorPermitido()
    det = 1
    for i in range(len(e)):
        det = det*(max(npbestsol[:nval,i+1])<e[i])
    if det:
        solsinrep  = np.unique(bestsolutions,axis = 0)
        solsinrep = np.array(solsinrep)
        print("Se interrumpio algoritmo: ")
        print(*solsinrep[nval:], sep = "\n")
        showResults(solsinrep)
```

```
        return True

#---------------------------------------------------------------------
# Mostrar resultados
def showResults(solutions):
    e = errorPermitido()
    df = pd.DataFrame({"Fitness Global": solutions[:,0],
                       "Fitness1": solutions[:,1],
                       "Fitness2": solutions[:,2],
                       "Fitness3": solutions[:,3],
                       "Fitness4": solutions[:,4],
                       "Alambre": solutions[:,5],
                       "D. externo": solutions[:,6],
                       "N° espiras": solutions[:,7],
                       "Longitud libre": solutions[:,8],
                       "Tipo de material": solutions[:,9]
                         })
    df2 =
df[(df['Fitness1']<e[0])&(df['Fitness2']<e[1])&(df['Fitness3']<1)]
    df2.to_csv("D:/Programas/EjemplosGA/Genetic-Algotihm/fitness2.csv")
#---------------------------------------------------------------------
#Cruce
def crossOver(bestsolutions):
    crosSolution = []
    for i in range(int(bestpob/2)):
        a = bestsolutions[2*i][5:len(bestsolutions[2*i])]
        b = bestsolutions[2*i+1][5:len(bestsolutions[2*i+1])]
        separ = random.randint(1,len(a)-1)
        sepran = random.random()

        if p_cruce < sepran:
            x = []
            x.append(b[:separ])
            x[0] = x[0] + a[separ:]
            y = []
            y.append(a[:separ])
            y[0] =y[0] + b[separ:]
            crosSolution.append(x[0])
            crosSolution.append(y[0])
        else:
            crosSolution.append(a)
            crosSolution.append(b)

    return crosSolution

#---------------------------------------------------------------------
#Mutación
def mutate(crosSolution):
    mutSolution = []
```

```
    for i in crosSolution:
        mutran = random.random()
        mut = random.randint(0,len(crosSolution[0])-1)

        if mutran > p_mutacion:
            varmut = aleatorio()[mut]
            j=[]
            j = i[:mut]
            j = np.insert(j,len(j),varmut)
            j = np.concatenate([j,i[mut+1:]])
            mutSolution.append(j)
        else:
            mutSolution.append(i)

    return mutSolution

#---------------------------------------------------------------------
#Nueva generación
def newSolution(mutSolution, bestSolution):
    newGen = []
    for i in mutSolution:
        newGen.append(np.asarray(i))

    for i in bestSolution:
        newGen.append(np.asarray(i[5:10]))
    newGen = np.array(newGen)

    newGen2 = []
    n = 0
    for s in newGen:
        n = n+1
        newGen2.append((fitness(s[0],s[1],s[2],s[3],s[4]),
                        fit1(s[0],s[1],s[2],s[3],s[4]),
                        fit2(s[0],s[1],s[2],s[3],s[4]),
                        fit3(s[0],s[1],s[2],s[3],s[4]),
                        fit4(s[0],s[1],s[2],s[3],s[4]),
                             s[0],s[1],s[2],s[3],s[4]))

    newGen2.sort()
    newGen2 = np.array(newGen2)
    solutions = newGen2[:pobinit,5:]

    return solutions

#---------------------------------------------------------------------
#Main
solutions = initialPopulation()
for z in range(GEN): # N° generaciones
```

```
        solutions = filterSolution(solutions)

        bestsolutions = selection(solutions)

        graph(bestsolutions)

        if stopCondition(bestsolutions) == True:
            break

        crosSolution = crossOver(bestsolutions)

        mutSolution = mutate(crosSolution)

        solutions = newSolution(mutSolution,bestsolutions)

plt.waitforbuttonpress()

#Fin de codigo
```