UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA CIVIL

ALGORITMOS EVOLUTIVOS PARA EL DISEÑO ESTRUCTURAL: ESTADO DEL ARTE Y CASO ESTUDIO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL

NICOLÁS ANTONIO GONZÁLEZ VALENZUELA

PROFESOR GUÍA:
FABIÁN ROJAS BARRALES

MIEMBROS DE LA COMISIÓN:
RAFAEL RUIZ GARCÍA
LEONARDO MASSONE SÁNCHEZ

Este trabajo ha sido financiado por el proyecto: FONDECYT REGULAR N° 1200709

SANTIAGO DE CHILE
2020

Resumen de la memoria para optar al título de:
Ingeniero Civil.
Por: Nicolás Antonio González Valenzuela
Fecha: 28/09/2020
Profesor guía: Fabián Rojas B.

Algoritmos evolutivos para el diseño estructural: Estado del arte y caso estudio

En el presente trabajo se realiza un estudio de los algoritmos genéticos y evolutivos centrado en el diseño estructural, analizando las investigaciones y trabajos previamente realizados en esta materia respecto a su formulación teórica, características claves, variantes y aplicaciones prácticas, permitiendo esclarecer sus ventajas y diferencias respecto a otros tipos de algoritmos y métodos de trabajo.

El objetivo del trabajo es desarrollar un estudio del estado del arte de los métodos de diseño basados en algoritmos generativos y evolutivos, entregando sus componentes principales y variantes en diversas investigaciones, con el fin de clasificar dichos métodos en torno a sus posibilidades de aplicación. Para finalmente, aplicar uno de estos algoritmos sobre una estructura de estudio, para evaluar su posible desempeño en edificios de hormigón armado.

Inicialmente, se recopila la bibliografía relacionada al diseño estructural, la cual se analiza identificando las principales características, ventajas y desventajas para cada método de diseño. Luego, se genera una clasificación de los distintos métodos en función de sus aplicaciones.

Posteriormente, se selecciona el problema de estudio que se busca solucionar, junto con el algoritmo aplicable más adecuado dentro de los entregados en el estado del arte recién estudiado. Finalmente, se analizan las soluciones entregadas por el método evolutivo o generativo.

Dentro de los resultados esperados de este trabajo, se encuentra no solo el desarrollo del estado del arte de los algoritmos evolutivos sino la comprensión e identificación del mejor método de optimización basado en dichos algoritmos, para resolver un problema estructural, gracias al análisis de las distintas características que poseen los distintos métodos; Además, se espera la obtención de candidatos a solución óptimos de acuerdo al problema planteado para posterior análisis e identificación de problemas en el desarrollo del proceso.

Agradecimientos

Este trabajo es la culminación de un largo (y muchas veces agotador) proceso el cual me permitió conocer a personas increíbles, adquirir nuevas habilidades, y desarrollarme como profesional y como persona. Todo esto no hubiera sido posible sin el apoyo incondicional de todos los que me rodean.

Quiero agradecer a mi familia quienes siempre me alentaron para seguir trabajando duro, dándome aliento en los momentos más difíciles, y apoyándome anímica y económicamente a lo largo de todo este proceso. A mis hermanos, siempre preocupados de mis avances y mi salud física y emocional. A mi madre quién ha sido un pilar fundamental y que siempre me ha dado el respaldo necesario para seguir. A mi padre, quién siempre creyó en mí, y a quién le dedico este trabajo.

Agradezco también a mis amigos. Por el apoyo y la ayuda cuando estudiábamos. Por estar ahí cuando era necesario desahogarse y desconectarse de todo. Y por todos esos momentos en que compartimos que hacían que la universidad no fuera solo estudio y trabajo sino un lugar ameno del que me puedo llevar bonitos recuerdos.

Finalmente, agradecer al profesor Rojas y a todos quienes fueron parte de este trabajo, por la confianza y la ayuda brindada, y por la oportunidad de conocer y ser parte del mundo de la investigación científica y el aprendizaje que esta conlleva.

Tabla de contenido

Índice de ilustraciones

1.- Introducción

En los últimos años se ha visto un aumento generalizado del uso de tecnologías relacionadas al campo de la inteligencia artificial en todas las áreas de la sociedad. El campo del diseño estructural no es la excepción, y distintas aplicaciones y estudios han surgido con alternativas y métodos de trabajo que permitan generar diseños más eficientes.

Uno de los métodos de optimización que ha ganado popularidad en los últimos años, debido principalmente al aumento de la capacidad de cómputo, son los denominados algoritmos evolutivos, los cuales tienen la ventaja de ser muy versátiles a la hora de trabajar con problemas en los que la calidad de las soluciones está sujetas a un amplio número de variables y en la que se presentan muchas restricciones, como lo son los problemas del campo del diseño estructural.

Frente a esto, se decide elaborar un análisis del estado del arte de este método de optimización que permita identificar de manera adecuada el alcance del método en el campo del diseño estructural, junto con las ventajas y desventajas que conlleva su uso.

Para ello, se hace una revisión de las investigaciones desarrolladas en el área desde su génesis hasta la actualidad, con el fin de comprender la evolución del método y entender así su posición respecto a otros tipos de modelos de optimización y herramientas de inteligencia artificial, tanto por la forma en la que actúa como en las ventajas y particularidades que posea. A partir de esto se busca establecer el tipo de problemas a los que es afín y evaluar la aplicabilidad del método en dicho tipo de problemas.

Finalmente, en miras de evaluar la respuesta del algoritmo, se establece un problema a optimizar y se analiza el desempeño del método tanto en la ejecución de este con en las soluciones entregadas por el proceso. A partir de ahí, se modifican aspectos claves del método para evaluar la adaptabilidad del proceso a un problema diferente.

1.1.- Objetivos

El objetivo principal de este trabajo es elaborar un estado del arte de los distintos algoritmos evolutivos enfocados en el diseño estructural, para posteriormente aplicarlo sobre un caso estudio para someterlo a evaluación. Para ello, se establecen objetivos específicos por cumplir a lo largo de este trabajo:

- Realizar una revisión bibliográfica de los distintos algoritmos generativos y evolutivos.
- Comparar y catalogar los distintos tipos de algoritmos en función de sus aplicaciones y procesos de optimización.
- Identificar las principales diferencias, similitudes, deficiencias y ventajas de los distintos tipos de métodos de diseño.
- Establecer un caso de estudio e identificar la clase de algoritmo que mejor se adecúe a este tipo de problema.
- Evaluar los resultados de la aplicación del algoritmo generativo y evolutivo al caso de estudio y los costes computacionales de las distintas fases del proceso.

1.2.- Metodología

Para lograr los objetivos planteados, se comenzará con el estudio del estado del arte. En este punto se realiza una revisión bibliográfica de diversos estudios relacionados con algoritmos generativos y evolutivos utilizados en el campo del diseño estructural, junto con métodos utilizados de manera similar fuera de este campo con el fin de tener una referencia sobre los posibles alcances de cada tipo de método y algoritmo.

Luego, para los distintos algoritmos estudiados, se identificará su función, métodos de optimización asociados y posibilidades de aplicación, que permitan compararlos y lograr generar clasificaciones y determinar las características esenciales de cada método analizado.

Posteriormente, se comenzará a trabajar en el caso de estudio. Para esto, se identificará el problema a solucionar, variables modificables y restricciones a cumplir. Teniendo claros estos conceptos, se establecerán procesos para la generación, evaluación y modificación de las distintas soluciones al problema de estudio. Todo esto apoyándose en el análisis de los distintos algoritmos presentados en la parte anterior.

Finalmente, se analizarán los resultados encontrados en cada parte del proceso de diseño, y se identificarán las principales dificultades del método utilizado, y se evaluarán las soluciones entregadas y se entregarán recomendaciones y líneas de trabajo a seguir.

2.- Marco teórico

2.1.- IA en ingeniería estructural

En el campo de la ingeniería estructural, diversos métodos relacionados a la inteligencia artificial han tomado gran interés, dentro de los que se pueden destacar el *machine learning*, la computación evolutiva y *fuzzy logic* (Salehi & Burgueño, 2018). Ya sea para el aprendizaje y reconocimiento o para optimización estocástica, los trabajos relacionados a estos campos han ido en aumento en los últimos años, lo que queda evidenciado en el trabajo realizado por Salehi y Burgueño esquematizado en la siguiente figura.

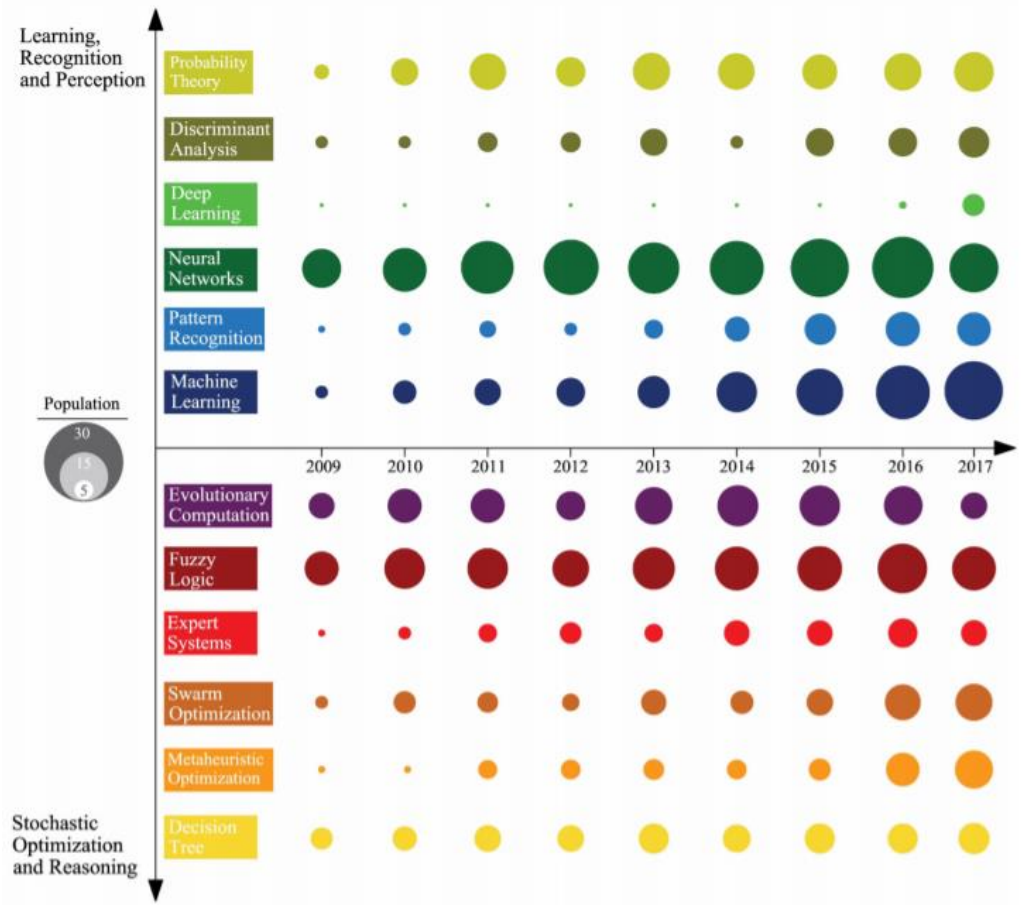

Figura 1: Publicaciones sobre ramas de IA en ingeniería estructural (Salehi & Burgueño, 2018).

Cada una de estas ramas tiene ventajas y desventajas respecto a la capacidad de uso en problemas relacionados al área de la ingeniería estructural, por lo que sus aplicaciones van desde la optimización en diseño estructural a la detección y control de daños es estructuras. En los siguientes párrafos se abordarán las ramas más estudiadas, y se comentará respecto a sus potenciales aplicaciones y ventajas.

Machine learning

El *machine learning* es un campo de la inteligencia artificial centrado en el estudio, diseño y desarrollo de algoritmos que pueden aprender de los datos entregados, y hacer predicciones de acuerdo con lo aprendido. En sí, se refiere a la capacidad de los computadores para aprender sobre algo, sin haber sido explícitamente programados para ello.

Es posible identificar cuatro subcampos del *machine learning*:

1. **Aprendizaje supervisado:** Este busca elaborar un modelo o función para predecir un resultado de acuerdo con ciertos datos de prueba. El entrenamiento de este algoritmo se hace por medio de un subconjunto de datos de prueba, el cual asocia valores de salida del algoritmo a cada conjunto de datos de entrada presentados. El algoritmo se clasifica en función de su objetivo, pudiendo ser de regresión o clasificación dependiendo si la variable a predecir es de un dominio continuo o discreto respectivamente.
2. **Aprendizaje no supervisado:** Con este método se busca separar los datos de entrada en conjuntos o *clusters*, de tal manera que todos los elementos en un conjunto tengan el mayor nivel de proximidad posible, pero que cada conjunto tenga un bajo nivel de proximidad con el resto. Esto se hace sin explicitar los conjuntos objetivos a formar, por lo que no hay un aprendizaje explícito.
3. **Aprendizaje reforzado:** En el aprendizaje reforzado no hay información relacionada a los objetivos buscados, sino que hay un agente que determina el comportamiento ideal del método en un contexto específico. Así, el agente recibe un pago relacionado a la calidad de los datos de salida asociados a cada ejecución frente a ciertos datos de entrada. De esta forma el método busca los objetivos óptimos en base al ensayo y error, de modo que el agente reciba la mayor cantidad de pagos en el tiempo.
4. ***Deep Learning**:* En el *Deep learning* se aprende de manera no supervisada sobre datos que no están estructurados, en busca de encontrar las características representativas de datos de entrada. En esencia, las estructuras que ocupa el *Deep learning* son redes neuronales con más de una capa oculta. Las más utilizadas en la comunidad de ingeniería estructural son las redes neuronales convolucionales (CNN).

La estructura central del machine learning se conoce como redes neuronales, estas son una estructura informática compuesta de múltiples unidades de procesamiento singulares interconectadas entre sí, permitiendo la realización de muchos cálculos independientes para resolver un problema más complejo. Cada unidad de procesamiento se denomina "Axón". En esencia, el axón es una transformación sobre ciertos parámetros de entrada para formar un valor de salida. La transformación en un axón opera en tres etapas:

1. Primero, se pondera cada parámetro de entrada (Q_i) por un peso (W_i). Estos pesos son la parte central de las redes neuronales.
2. Luego, se suman los valores ponderados. Esto representa una transformación lineal de los valores de entrada.
3. Con el fin de no perder la no linealidad en un problema, se aplica una "Función de activación" sobre el valor ponderado. Suele ser usada la función sigmoide, aunque existen varias alternativas.

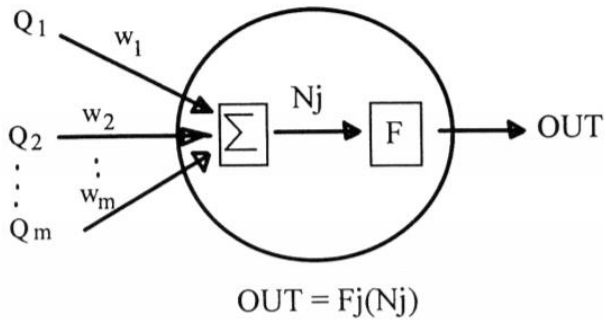

Figura 2: Esquema Axón (Berrais, 1999)

La ventaja de las redes neuronales radica en la posibilidad de conectar múltiples axones para formar estructuras complejas para resolver problemas de gran dificultad. Existen diversas arquitecturas de redes neuronales, pero la más utilizada es la de *back propagation.* En esta, los axones se agrupan por "capas", formando tres grupos: Capa de entrada, Capa de salida, y Capa oculta. Esta última representa una o más capas de axones intermedias entre la capa de entrada y la de salida.

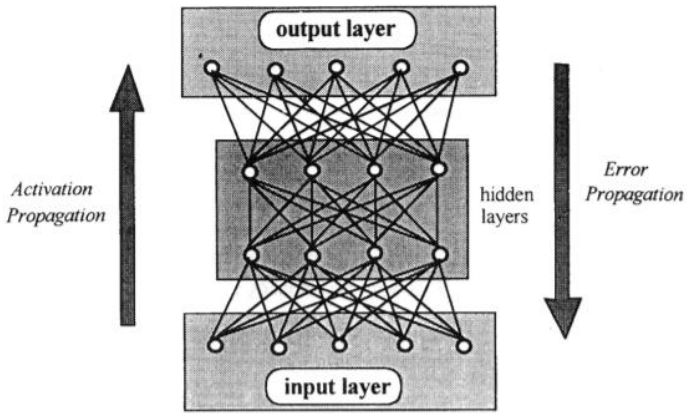

Figura 3: Esquema Red Neuronal (Berrais, 1999)

Esta estructura permite predecir el valor de salida de un conjunto de datos de entrada a través de un proceso iterativo: Primero, se generan pesos entre axones de manera aleatoria, de modo de calcular todos los valores de salida de cada función de activación. Esto se repite hasta obtener los valores de salida de la capa de salida. Luego, se calcula el error entre los valores esperados de salida y los reales y se distribuye entre los pesos entre la penúltima capa y la capa de salida. Luego, se compara el valor esperado de la penúltima capa, con los valores reales, y se distribuye entre los pesos de la antepenúltima capa. Esto se repite hasta distribuir el error entre los pesos de la capa de entrada. El proceso se ejecuta nuevamente hasta que el error converge a un mínimo, y se guardan los valores de los pesos de la red.

***Fuzzy logic*:**

La lógica difusa es un método que opera sobre valores de entrada, pero contextualizados sobre sí. Esto se logra mediante el uso de cuantificadores sobre la cualidad de los valores entregados. A modo de ejemplo, en un conjunto de datos distribuidos normalmente con media 10 y varianza 1, un valor igual a 10 puede considerarse promedio, mientras que un valor 15 se considera alto y 50 muy alto. Luego es capaz de tomar decisiones mediante uso de subconjuntos difusos por medio de relaciones heurísticas "Sí" y "Entonces". Siguiendo con el ejemplo anterior, los valores 50 y sus cercanos pertenecen al conjunto de valores Muy Alto, y una regla sería: "Sí" valor es "Muy alto", "Entonces" reduzco "Mucho" el valor.

El proceso puede separarse en 4 etapas principales (Akkurt, Gokmen, & Can, 2004):

- **Difuminación**: Esta etapa pasa los datos de entrada por funciones de pertenencia, los cuales los clasifican de acuerdo con cuantificadores previamente establecidos. Es común el uso de cuantificadores del tipo "Alto" – "Muy alto", y funciones de pertenencia lineales simples, como las triangulares.

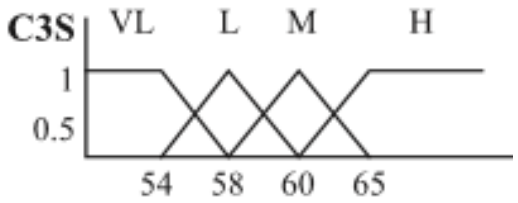

Figura 4: Función de pertenencia lineal, desde "Very low" a "High"

- **Reglas difusas**: Se establecen las reglas que incluyen todas las relaciones posibles entre los datos de entrada y de salida. Es en este punto donde se utilizan relaciones heurísticas del tipo "Si" y "Entonces". En este método no existen parámetros de modelamiento o ecuaciones de matemáticas, por lo que todas las incertidumbres se encuentran incluidas en las relaciones.
- **Motor de inferencia difusa**: Este, toma en cuenta todas las reglas difusas y aprende a transformar los datos de entrada en datos de salida.
- **Des difuminado**: Esta parte del proceso convierte los datos de salida del motor de inferencia en valores numéricos. Dado que el proceso opera sobre conjuntos, hay muchos métodos para esta etapa desarrollados sobre área, dentro de los que se destacan: Bisectriz, promedio de máximos, y centro de gravedad.

Cabe destacar que es usual incorporar otros métodos de IA en algunas etapas del proceso de lógica difusa, como el uso de redes neuronales para asignar reglas difusas, o para la elaboración del motor de inferencia.

Aplicaciones y desventajas

El *machine learning* ha sido muy utilizado en aplicaciones de monitoreo de salud estructural. Para eso, suele utilizarse la respuesta basada en datos obtenidos mediante sensores o mediciones, y es comparada con la respuesta esperada del modelo original de la estructura. Esto es particularmente eficiente al tener largos volúmenes de datos disponibles, o si las características físicas de la estructura son desconocidas o difíciles de modelar (Salehi & Burgueño, 2018). Suelen usarse otros métodos de IA para ayudar a disminuir la incertidumbre o ruido en los datos.

Además, algoritmos de *machine learning* son usados para predecir las propiedades mecánicas del concreto. Ya sea para modelar fórmulas que predigan la resistencia a compresión de concreto de alta resistencia (Yeh & Lien, 2009), o para modelar la resistencia a tracción y compresión de concreto expuesto al fuego (Chen, Chang, Shih, & Wang, 2009).

Por otro lado, el método de Fuzzy logic resulta particularmente útil para procesos de alta complejidad, no lineales, o con parámetros cualitativos o poco precisos, junto con procesos para los que no existe un modelamiento matemático explícito. Debido a esto, en ingeniería estructural es utilizado para análisis de fiabilidad y predicción de resistencia de elementos. En este último caso, es usual intentar predecir la resistencia del hormigón a partir de la presencia de ciertos compuestos (Akkurt, Gokmen, & Can, 2004).

Sin embargo, estos métodos no ofrecen alternativas para la optimización del diseño estructural. El método de machine learning puede ser usado para esto, pero la calidad del proceso de optimización está ligado a la cantidad de datos de entrenamiento necesarios, por lo que un adecuado proceso de optimización puede requerir una gran base de datos previa. Ante esto, se optará por utilizar un mecanismo de optimización por exploración denominado computación evolutiva.

2.2.- Marco histórico

Introducción

La computación evolutiva es una técnica que utiliza modelos computacionales de procesos de evolución Darwiniana y selección natural, codificados en algoritmos evolutivos. Lo que hoy se entiende como campo de la computación evolutiva, es el resultado de la interacción de distintos enfoques relacionados a las estrategias evolutivas en el campo de la programación y resolución de problemas. El uso de modelos computacionales que permitieran una mejor comprensión del proceso de evolución natural puede rastrearse hasta inicio de la década de 1950 (De Jong, Fogel, & Schwefel, 1997).

Dentro de los primeros trabajos asociados al campo de computación evolutiva se observan las primeras descripciones de un proceso evolutivo centrado en la solución de problemas por computador. Se destacan los trabajos de Alex Fraser (1957) y de Richard Friedberg, quién analizó el uso de un algoritmo evolutivo para la programación automatizada (Friedberg, 1958).

También es posible encontrar los trabajos de Hans-Joachim Bremermann, quién elaboró las primeras teorías sobre los algoritmos evolutivos. Además, desarrolló trabajos para aplicar la evolución simulada a problemas de optimización numérica, tanto para optimización convexa como para ecuaciones no lineales simultaneas (Bremermann, 1962).

Además, en este período se ven los primeros intentos de una aplicación de técnicas evolutivas en el diseño y análisis de experimentos industriales de la mano de George Box, quién desarrolló las ideas de lo que llamo el operador evolutivo (EVOP, por sus siglas en inglés). Si bien esta idea no llegó a convertirse en un algoritmo computacional, sirvió de base a muchos trabajos posteriores centrados en esa área (Box, 1957).

Estos juntos con otros trabajos fueron los primeros acercamientos teóricos y prácticos, y fueron recibidos con escepticismo por parte de la comunidad. A mediados de la década de 1960 en adelante, fueron desarrollados trabajos en lo que se conoce como las tres ramas principales de algoritmos evolutivos: Programación evolutiva, algoritmos genéticos y estrategias evolutivas.

Programación evolutiva

La idea de programación evolutiva fue desarrollada por Lawrence J Fogel en la década de 1960. En ese entonces, la inteligencia artificial estaba centrada en el desarrollo de redes neuronales primitivas. Fogel estimaba que la idea de inteligencia debía basarse en la capacidad de adaptarse un rango de ambientes para cumplir metas específicas, entendiendo un comportamiento inteligente como la habilidad de predecir un entorno, y de traducir esa predicción a una respuesta que se adapte a los objetivos buscados (Fogel, Owens, & Walsh, 1966).

Bajo esta línea, se desarrolló un problema evolutivo que consistía en desarrollar un algoritmo que pudiera operar sobre una secuencia de símbolos (Entorno). La idea es que el algoritmo fuera capaz de predecir el siguiente símbolo de la secuencia, lo cual era evaluado de acuerdo con una función de pago (Un criterio de evaluación) previamente definida.

La componente evolutiva radica en el uso de máquinas de estados finitos padres y descendientes. Así, se muta la información las maquinas padres, formando descendencia. El tipo, probabilidad y número de mutaciones que sufre cada padre se establece previamente. Luego, la descendencia es evaluada junto con las maquinas padres. Posteriormente, las maquinas que tengan las mejores calificaciones se mantendrán para ser los padres de la siguiente generación, y el proceso se repite.

Este procedimiento fue aplicado a problemas de predicción, identificación y control automático, para luego simular poblaciones coevolucionando. Otros trabajos fueron realizados por Lutter y Huntsinger (1969), Burgin (1969), Atmar (1976), Dearholt (1976) y Takeuchi (1980) en torno a evolucionar máquinas de estados finitos para predicción de secuencias, reconocimiento de patrones y el campo de los juegos. Más tarde, Fogel extendió el procedimiento para soluciones del problema del viajante, optimización de funciones continuas, planeación de rutas, selección de subconjuntos óptimos y entrenamiento de redes neuronales (De Jong, Fogel, & Schwefel, 1997).

Debido a que este tipo de algoritmo no tiene restricciones con respecto al uso de tipos de datos en sus atributos, y a que posee una estructura de programa fija que le permite parámetros numéricos para evolucionar, es muy usado en para problemas de predicción, generalización, juegos y control automatizado (Coello C. , 2005).

Algoritmos genéticos

Los primeros trabajos relacionados a algoritmos genéticos fueron llevados a cabo por John Holland a principios de los años 1960 quien buscaba entender los principios subyacentes de los sistemas adaptativos, aquellos que son capaces de auto modificarse en respuesta a cambios en el ambiente en el cual cumplen su función. Para él, la característica clave de un sistema adaptativo es la correcta aplicación de la idea de competición e innovación para responder a eventos anticipados y ambientes variantes (De Jong, Fogel, & Schwefel, 1997). Algunos modelos simples de evolución biológica capturan estas ideas a través de nociones sobre selección natural y producción continua de descendencia.

A mediados de 1960 las ideas de John Holland fueron utilizadas por tesistas de doctorado que trabajaban con él. Estas aplicaciones tenían componentes genéticos ya que la información de los objetos que serían evolucionados era representada a través de genomas, mientras que los mecanismos de reproducción y traspaso de la información eran abstracciones de operadores genéticos, tales como mutación y apareamiento.

Dentro de estos trabajos se encuentran las primeras experimentaciones con operadores de cruzamiento. En primer lugar, la tesis de John Bagley presenta los primeros usos de mecanismos de selección y representaciones diploides para el cruzamiento de la información de los padres. Estas últimas son representaciones asociadas a la información contenida en los alelos presentes en el genoma, los cuales vienen mitad del padre y mitad de la madre (Bagley, 1967); Al mismo tiempo, Richard Rosenberg trabajaba en la simulación de la evolución de un ambiente bioquímico compuesto por células capaces de producir enzimas, cuya información eran representada como diploide, para producir las concentraciones químicas apropiadas en dicho ambiente (Rosenberg, 1967).

Trabajos posteriores se centraron en mecanismos de selección y apareamiento, dentro de los que se encuentra la tesis de Daniel Cavicchio quién ahondó en formas de selección elitista, junto con experimentos en los que se adaptan las tasas de apareamiento y mutación (Cavicchio, 1970); También se destaca la tesis de Ralf Hollstien quién estudió la selección alternada y esquemas de apareamiento, experimentando con estrategias de reproducción inspiradas en técnicas usadas por criaderos de animales. Además, Hollstien incluyó en su trabajo los primeros acercamientos sobre codificación binaria del genoma y observaciones sobre codificación de Gray (Hollstien, 1971).

Todos estos trabajos fueron realizados mientras Holland desarrollaba una teoría general de sistemas adaptativos, quién articulo su información para realizar un análisis teórico sobre los que él llamó planes reproductivos, lo que en esencia son representaciones simples de un algoritmo genético (Holland, 1975).

Los algoritmos genéticos son los algoritmos evolutivos más utilizados, sobre todo para problemas de *machine learning*, reconocimiento de patrones y optimización (Coello C. , 2005). Los operados de mutación y recombinación son inherentes en este tipo de algoritmo.

Estrategias evolutivas

La primera versión de estrategia evolutiva, denominada (1+1) ES, fue presentada por Ingo Rechenberg el año 1964 mientras trabajaba en estudios de aerotecnología junto a P. Bienert y H.P. Schwefel. La idea era desarrollar una clase de robot que pudiese simular experimentos sobre un cuerpo flexible y delgado en un túnel de viento para minimizar su arrastre. La primera idea del grupo falló al testear el robot buscando la forma óptima de una placa articular como modelo de prueba.

El posterior planteamiento de Rechenberg consistía en utilizar una población de un ancestro y un descendiente por cada generación, con mutaciones en el ancestro seleccionadas de manera aleatoria. La selección se haría mediante el uso de un dado, lo que les otorgaba a las mutaciones un carácter discreto con una distribución de probabilidad binomial (Rechenberg, 1965). El resultado fue que se alcanzó la forma esperada de manera analítica, no sin antes estancarse algunas generaciones en una geometría distinta, un óptimo local.

Schwefel continuó el trabajo, desarrollando investigaciones analíticas sobre las mutaciones enteras distribuidas de manera binomial, demostrando que este tipo de distribuciones pueden estancarse prematuramente y dando paso al uso de mutaciones normalmente distribuidas. Aun así, esto no se volvió norma, y el uso de distribución binomial continuó en la práctica (Schwefel, 1965). Posteriormente, en el contexto del diseño de una boquilla de agua que maximizara la eficiencia energética, Schwefel incorporó la duplicación y eliminación genética en su estrategia (Schwefel, 1968).

Posteriormente, Rechenberg centró su trabajo en el desarrollo de una teoría de tasa de convergencia, que terminaría creando una regla para adaptar la desviación estándar de las mutaciones, para luego incluir el concepto de población y recombinación, resultando en la primera estrategia evolutiva multi miembro (Rechenberg, 1971). Esta fue denominada (μ+1) ES, pues posee múltiples ancestros.

Schwefel orientó su trabajo de manera similar llegando a resultados análogos a los encontrados por Rechenberg. Sin embargo, formuló la (μ+λ) ES la cual permite la incorporación de parámetros de mutación variantes, logrando que el algoritmo posea la capacidad de auto adaptar parámetros internos (Schwefel, 1974). Posteriormente, realizó análisis empíricos detallados sobre la auto adaptación, demostrando la importancia de la recombinación, dando lugar a la noción de auto adaptación para el aprendizaje colectivo (Schwefel, 1987).

Producto de esta capacidad de auto adaptación y a la representación de soluciones mediante vectores en números reales, las estrategias evolutivas son eficientes para obtener la expresión actual de algún atributo sin necesidad de generar una codificación específica para ello. Esto hace que las estrategias evolutivas sean utilizadas en problemas de enrutamiento y redes, bioquímica, óptica y diseño ingenieril (Coello C. , 2005).

Enfoques finales

Las investigaciones relacionadas a estrategias evolutivas disminuyeron posteriormente debido a la falta de capacidad de cómputo, panorama que se mantuvo hasta la década de 1990. A partir de este punto, diversos trabajos fueron realizados generando muchos conceptos paralelos, presentando estrategias similares, pero con conceptos ligeramente variantes. Fue Thomas Bäck quien introdujo un algoritmo estándar para todas las ramas de algoritmos evolutivos (Bäck, 1996).

Estas ramas de trabajo se desarrollaron de manera independiente hasta inicio de la década de 1990. A partir de este punto se inició un esfuerzo desde la comunidad para generar instancias de interacción entre los distintos investigadores de algoritmos evolutivos. Este esfuerzo derivó en la realización del taller internacional "*Parallel Problem Solving from Nature*", realizado el año 1991 en Dortmund (Schwefel, 1991). Este evento propició la cooperación de las comunidades de algoritmos evolutivos, lo que generó un aumento en los eventos, foros y talleres relacionados al área.

La interacción llevó al consenso respecto al nombre del nuevo campo de estudio, denominada computación evolutiva, lo cual quedó demostrado con la publicación de la revista del MIT del mismo nombre el año 1993, y las conferencias realizadas en 1994 en el congreso mundial de inteligencia computacional IEEE WCCI (Michalewicz, 1994).

2.3.- Algoritmos evolutivos genéricos

Si bien presentan diferencias, los métodos recién nombrados poseen características clave en común. Estos son, en esencia, modelos computacionales basados en la evolución natural, los cuales son diseñados para generar soluciones óptimas de un problema que no puede solucionarse fácilmente de manera analítica, ya sea por ser problemas que incorporen una gran cantidad de variables, que dichas variables sean discretas o cuyas soluciones deban ser evaluadas de manera cualitativa. Más aún, los algoritmos evolutivos son especialmente eficientes frente a problemas en los que no es posible usar estrategias heurísticas, ya que su formulación representa una optimización estocástica mediante conceptos metaheurísticos (Jangid, Puri, & Kumar, 2019).

Descripción del algoritmo

En esencia, el algoritmo evolutivo opera como un proceso de optimización que busca la solución óptima a un problema en un espacio de candidatos a soluciones previamente definido, de manera iterativa. Para esto, se selecciona una "Población" de soluciones en el espacio de búsqueda, la cual pasa por un proceso de cambios graduales que dependen de la calidad de los "Individuos" de la población, entendiendo a la calidad como la aptitud de la solución en el "Entorno" del problema. La evaluación de la calidad de las soluciones se obtiene mediante una función objetivo previamente definida.

La estructuración canónica de un algoritmo evolutivo consta de seis pasos claves que se muestran a continuación (Kicinger, Arciszewski, & De Jong, 2005):

1. **Inicialización** de la población (Aleatorio).
2. **Evaluación** de los individuos de la población.
3. **Selección** de los individuos de la población para ser ancestros.
4. **Generación** de nuevos individuos por operadores de variación.
5. Evaluación de nuevos individuos.
6. **Reemplazo** de individuos en la población, y repetir proceso desde 2.

En primer lugar, genera una población de individuos de manera usualmente aleatoria, con el fin de no forzar soluciones similares a la población inicial. Dichos individuos son evaluados mediante una función objetivo, o función de costo previamente definida, que refleje la calidad de las soluciones frente al problema. Posteriormente, se seleccionan aquellos individuos que se comporten de mejor manera, es decir, aquellos que tengan la mejor calificación entregada por la función objetivo para convertirse en "Ancestros" de la nueva generación. El número de individuos seleccionados en cada iteración puede variar a lo largo del proceso.

Luego, se utilizan los operadores de variación que se ejecutan sobre copias de los ancestros variando parte de la información que define a dichos individuos. Estas copias transformadas se convierten en la "Descendencia" de la nueva generación, y son evaluados de la misma manera que los ancestros.

Finalmente, se reemplaza a la población original por la población mezclada de ancestros y descendientes, manteniendo o no a cierto porcentaje de individuos de la población original. Si los individuos de la nueva población cumplen con los objetivos del problema, se obtiene una gamma de candidatos a soluciones que se adaptan a las condiciones del problema, permitiendo una selección no automatizada a través de un criterio cualitativo. En caso de no cumplir con los objetivos, se repite el proceso desde la etapa de selección.

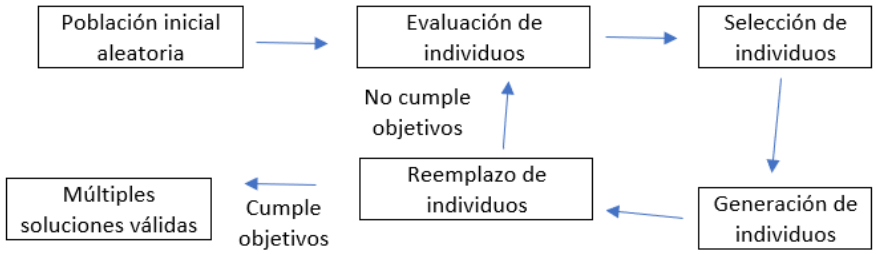

Figura 5: Esquema algoritmo evolutivo canónico

2.3.1.- Descripción de etapas

Codificación

La codificación representa la parte más importante dentro de un algoritmo evolutivo, a la vez que no es parte de este. La codificación es la forma de representar las características o variables presentes en las soluciones que serán modificadas para explorar el espacio de soluciones posibles en búsqueda del óptimo. En otras palabras, son las características que contienen la información a partir de las cuales se generarán las soluciones del problema de optimización (Kicinger, Arciszewski, & De Jong, 2005).

Este método de caracterización de soluciones permite la generación de tantos modelos como sea posible según los grados de libertad, variables o características posibles se entregue. Una mayor cantidad de variables permitirá generar soluciones más complejas y en mayor cantidad. Esto puede significar en soluciones más cercanas a la óptima.

A modo de ejemplo, la codificación más usual en los algoritmos genéticos es la caracterización binaria para denotar la existencia o ausencia de un elemento. De esta forma, por ejemplo, es posible generar una estructura de edificio metálico en la que cada piso posea (1) o no posea (0) elementos cruzados como se ve en el siguiente ejemplo:

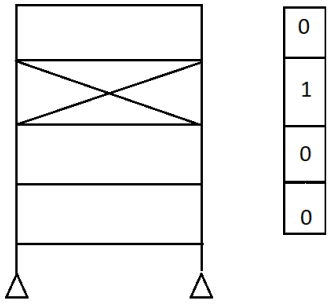

Figura 6: Codificación binaria

En los algoritmos genéticos el código (En este caso 0 1 0 0) representa un "gen" a partir del cual se genera una solución, y cada valor representa un "alelo" posible. En caso de que la codificación sea binaria, la cantidad de soluciones que pueden ser generadas es 2^n donde n es la cantidad de alelos (en este caso, de pisos). Así, simplemente con variar un valor del gen se puede construir una solución distinta, hasta llegar a aquella óptima según los criterios definidos.

Es por esto por lo que la codificación es tan importante pues establece las variaciones posibles que pueden tener las soluciones limitando, a su vez, el espacio de soluciones en los que se buscará el óptimo del problema. Debido a esto la codificación varía según el problema buscado, desde la existencia o ausencia de elementos, hasta una caracterización de los tipos de elementos y dimensiones de estos.

Por otro lado, la posibilidad de crear múltiples soluciones a partir de una codificación ha abierto la puerta al diseño automatizado a gran escala, tomando el nombre de diseño generativo. Acá, el diseño de un elemento estructural, mecánico o la carcasa de un producto puede modificarse por completo a partir de una modificación de algunas componentes clave previamente codificadas, como el tamaño de una pantalla o la altura del alma de una sección transversal de viga, permitiendo la exploración de múltiples diseños para ser seleccionados de forma automatizada o manual asegurando diseños cercanos a los buscados originalmente, pero optimizados de acuerdo con criterios establecidos previamente.

Generación

Establecida la codificación para la generación de soluciones, se crea una población inicial de individuos. En general la primera población es usualmente creada de manera aleatoria, con el fin de asegurar que estas cubran todo el espacio de soluciones posibles, o, dicho de otra forma, que sean lo suficientemente distintas unas de otras para garantizar la creación de todos los tipos de solución posibles.

La generación de una mayor cantidad de soluciones asegura la aparición de múltiples tipos de solución lo que se traduce en evitar óptimos locales. Sin embargo, cada solución aumenta el coste computacional necesario para generarla y evaluarla, y a partir de cierto punto el coste computacional puede ser demasiado elevado sin aumentar la calidad de las soluciones finales obtenidas.

Evaluación

Una vez generadas las soluciones, estas pasan por un criterio de evaluación previamente establecido. El criterio de evaluación más común es el coste, ya sea de recursos, dinero, computacional, etc. Otros criterios utilizados son el tiempo necesario para la ejecución, huella de carbono, mano de obra necesaria.

Además, es posible establecer criterios cualitativos, los cuales suelen ser usados en diseño de productos o arquitectura. Este puede incluir el buen aspecto del producto, cosa que varía según el criterio de cada diseñador.

Cabe destacar que es posible usar múltiples criterios de evaluación a la vez con el fin de encontrar un diseño óptimo. De esta forma, es posible buscar un diseño de bajo coste, baja huella de carbono y bajo tiempo de construcción. Este tipo de problemas se denominan problemas de optimización multi objetivos, y en los cuales la solución por medio de algoritmos evolutivo resulta útil en comparación a una búsqueda de soluciones de manera analítica.

Para cuantificar la calidad de las soluciones de acuerdo con estos criterios, suele usarse una función de evaluación, de manera de dotar de un peso a cada característica de la solución, con el fin de que cada solución tenga asociado un valor o magnitud en función de las características que presente y su cercanía con lo buscado.

Selección

Una vez evaluadas las soluciones, se seleccionan aquellas que pasaran a la siguiente fase del algoritmo. Para esto, hay distintos métodos los cuales pueden afectar en la heterogeneidad de las soluciones finales del proceso y en los tiempos de cómputo. Así, métodos de selección más estrictos pueden ocasionar que las soluciones futuras sean similares a una solución inicial con una buena calificación en su evaluación en comparación con el resto, pudiendo caer en óptimos locales. Por otro lado, otros métodos de selección pueden evitar la similitud entre soluciones, haciendo el proceso completo más lento.

Dentro de los métodos de selección más utilizados, se encuentran (Kicinger, Arciszewski, & De Jong, 2005):

- **Ruleta**: En este método, posibilidad de ser seleccionado es proporcional a la calificación normalizada obtenida en la evaluación. De este modo, las soluciones más aptas tienen más probabilidad de ser seleccionadas para pasar a la próxima generación.
  Debido a esto, es posible converger rápidamente a una solución apta de un óptimo local que haya sido generada aleatoriamente, lo que se conoce como convergencia prematura.
- **Ranking**: Con el fin de evitar la convergencia prematura, este método le asigna a cada solución evaluada una posición, dejando a la mejor solución con una puntuación N, y a la peor solución con una puntuación de 1. Luego, la posibilidad de ser seleccionadas es proporcional a su puntuación.
  Si bien esto disminuye la posibilidad de caer en óptimos locales al reducir la diferencia de puntuación entre la mejor y peor solución, también baja la velocidad de convergencia pues también decae la diferencia de puntuación que hay entre las mejores soluciones.
- **Torneo**: En este método N soluciones son elegidas de manera aleatoria para "competir" entre ellas: Aquella solución con mejor calificación en la evaluación pasa a la siguiente fase del algoritmo. Esto se repite hasta haber alcanzado la cantidad de soluciones buscada.
  Dado que todas las soluciones tienen la misma probabilidad de ser elegidas para un torneo, la selección puede ser muy variada. Esto puede ajustarse con la proporción de soluciones elegidas en cada torneo. Mientras más grande sea cada torneo, mayor probabilidad de seleccionar a las soluciones con mejor calificación, limitando la aparición de soluciones más variadas.
- **Truncamiento**: En este método se selecciona solo una porción de los mejores individuos de cada población. Es el método más apreciado en las comunidades de estrategias evolutivas para los modelos (μ,λ), en los que se seleccionan los mejores individuos entre la descendencia, y (μ+λ), en el que se seleccionan los mejores individuos de la totalidad de la población.
  De manera similar a lo que ocurre en el método de ruleta, al elegir sólo las mejores soluciones puede producirse la convergencia prematura, y la restricción sobre soluciones puede evitar el escape de óptimos locales. Sin embargo, garantiza una rápida convergencia.
- **Boltzmann**: En el método Boltzmann la probabilidad que tiene una solución de ser elegida es proporcional a su calificación de manera exponencial. Sin embargo, esta probabilidad se ve regulada por un parámetro que varía en función de la generación (Iteración) en el que se ejecute el algoritmo, de la forma $P = \exp\left(-\frac{f_{máx} - f(X_i)}{T}\right)$, donde T es el parámetro que varía con la generación.

En las generaciones iniciales, el parámetro es alto lo que implica una selección menos estricta, lo que permite que sean elegidas soluciones iniciales con baja calificación facilitando la exploración del espacio de soluciones.
En las generaciones finales, el parámetro es bajo restringiendo más el proceso de selección. Esto aumenta la velocidad de convergencia al descartar soluciones con calificaciones distintas a la solución más apta.

Reproducción

Para continuar con el proceso de optimización es necesario crear una nueva generación de soluciones que contengan la información de los individuos seleccionados en el paso anterior. Los operadores más utilizados son el de mutación y el de recombinación (Jangid, Puri, & Kumar, 2019).

La mutación es un operador unario pues actúa sobre un único individuo. Acá se crea una copia de los individuos seleccionados, para luego variar su información. Este nuevo individuo modificado representa la descendencia de la población.

La recombinación es un operador binario que trabaja sobre 2 individuos padres, combinando su información para generar descendencia. La forma clásica de recombinación es intercambiar bloques del código (Genoma) entre las soluciones, como se muestra en el esquema siguiente:

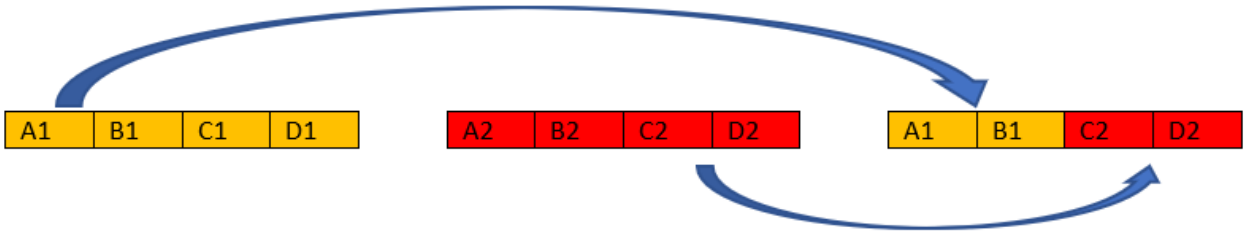

Figura 7: Ejemplo de recombinación

Cabe destacar que esta forma de recombinación no siempre es posible. Puede presentarse un código en el que la información de una parte de este dependa de otra sección, y que esta forma de reemplazo genere una solución cuyo código viole dicha restricción. Por lo mismo, el método de recombinación depende directamente de la forma en que la codificación de las soluciones se realice.

Mutación

Si en algún momento del proceso iterativo de optimización alguna de las soluciones se encuentra en la vecindad de un óptimo local, al traspasar la información a las próximas generaciones las soluciones creadas a partir de esta tenderán a tener una mejor calificación que el resto de los individuos. Dichos individuos traspasaran su información a las siguientes generaciones, haciendo que el conjunto de soluciones converja a ese óptimo local.

Para evitar esto, una vez terminada la fase de reproducción, las soluciones pasan por un proceso de mutación alterando la información de los individuos. Esto permite que, pese a que alguna de las soluciones se encuentre en un óptimo local, una solución tenga la oportunidad de moverse más cerca del óptimo global a través de una alteración de su información de manera aleatoria, haciendo que el proceso continúe de manera correcta. Cabe destacar que este proceso se realiza cuando la reproducción no se hace a través de mecanismos de mutación unaria.

La forma clásica de llevar a cabo este proceso es que cada individuo tiene una probabilidad de mutar, cambiando alguno de los valores insertos en su código. Un ejemplo de esto es el siguiente, con una probabilidad de mutación de 10% (Si $P \leq 0.1$).

Figura 8: Ejemplo de mutación

En esta parte del proceso puede presentarse el mismo problema que en la reproducción por recombinación: Si una parte del código depende de una sección anterior, la forma de mutación recién presentada puede violar dicha restricción, por lo que el proceso de mutación también depende de la forma en que se genera el código.

Reemplazo e iteración

Finalmente, se introduce a la descendencia reemplazando a la población original. Si el reemplazo es sobre la totalidad de la población, se dice que el algoritmo es generacional. Por otro lado, si la descendencia reemplaza solo a un subconjunto de la población, se dice que el algoritmo es de estado estacionario (Kicinger, Arciszewski, & De Jong, 2005).

Teóricamente, estos pasos se repiten hasta alcanzar el óptimo global. Sin embargo, en la práctica dicho óptimo global se desconoce y dado que el algoritmo evolutivo es un proceso estocástico, este continúa pese a encontrar el óptimo (Jangid, Puri, & Kumar, 2019). Debido a esto, el proceso se repite hasta que la evaluación del conjunto de soluciones generadas converge a un máximo (O mínimo si se busca).

Por otro lado, este proceso puede variar a medida que se avanza en el número de iteraciones. Esto suele verse en problemas multi objetivos y multi variables, en el que la evaluación u otra parte del proceso se modifica.

A modo de ejemplo, en un problema en el que se busque un diseño con el menor costo y tiempo de ejecución los tiempos de cómputo pueden ser muy altos. Para disminuir esto, puede desarrollarse un proceso que en sus inicios sea exigente al momento de buscar soluciones que optimicen el coste, pero menos exigente con el tiempo de ejecución, con el fin de explorar el espacio de soluciones. Luego, una vez avanzado en el proceso, modificar la función de evaluación para que sea más exigente respecto al tiempo de ejecución, pero con soluciones más cercanas al óptimo que las generadas en sus inicios para forzar la convergencia al óptimo.

2.3.2.- Desventajas

Gracias a los mecanismos que operan en el algoritmo evolutivo, este es capaz de optimizar una amplia variedad de funciones que a través de medios analíticos u otro tipo de procesos de optimización serían muy costosos si no imposibles de desarrollar. Sin embargo, el algoritmo evolutivo canónico presenta ciertas desventajas en su uso, dentro de las que se pueden destacar las siguientes:

- En problemas en los que se desea optimizar una amplia variedad de elementos, la representación de las soluciones requiere codificaciones o genomas extensos que permitan parametrizar la totalidad de la solución, lo que conlleva a un aumento de la capacidad de cómputo necesaria para llevar a cabo el proceso. Debido a esto, es necesario adecuar la codificación de las soluciones de manera de minimizar los tiempos de cómputo necesarios, pero sin llegar al punto de sub representar el espacio de soluciones, lo que podría llevar a individuos finales con una calidad inferior a la entregada por el óptimo global.
- La calidad de las soluciones se ve enormemente influenciada por la función de evaluación utilizada. Sin embargo, puede ser complicado encontrar funciones de evaluación que trabajen de manera correcta con múltiples variables que impactan de manera diferente la calidad de las soluciones entregadas. Ante esto, se requieren mecanismos de trabajo adicionales que permitan ajustar la función de evaluación, de manera que el proceso evolutivo se desarrolle de manera óptima.
- Al ser un proceso de optimización por exploración, los tiempos de cómputo necesarios para su desarrollo pueden ser demasiado elevados. Además, dado que consta de muchas etapas en las que actúan mecanismos diferentes, es posible observar instancias en las que el algoritmo ocupa parte importante del tiempo de cómputo total, lo que genera problemas de escalabilidad del algoritmo.

2.4.- Variantes en algoritmos evolutivos

Como se ha visto, los algoritmos evolutivos presentan elementos claves que le permiten llevar a cabo el proceso de optimización. Sin embargo, es posible adaptar el modelo dependiendo de las características del problema, con el fin de obtener resultados más deseables. Ya sea en la forma de caracterizar el espacio de soluciones, o en el proceso evolutivo en sí, las distintas variantes al algoritmo permiten resolver problemas de mayor complejidad o con distintas restricciones. A continuación, se observarán algunas características variantes al modelo genérico, junto con ejemplos que permitan identificar posibilidades de uso y aplicaciones útiles.

2.4.1.- Representaciones

El acercamiento computacional al diseño estructural conceptual requiere de una formulación adecuada del espacio de búsqueda de los candidatos a soluciones, por lo que esta formulación es un proceso crítico. Esto, pues cada diseño contempla atributos que describe las características estructurales, forma, función, requerimientos y restricciones que ha adquirido a lo largo del proceso evolutivo, atributos que eventualmente serán capaces de generar no solo las soluciones de diseño esperadas para el problema, sino que también será capaz de generar soluciones inesperadas pero factibles que presenten una mejor afinidad al problema.

Es por esto que los objetivos del diseñador influencian enormemente el tipo de soluciones entregadas. Así, cuando el objetivo es encontrar diseños óptimos, los diseñadores suelen limitarse a un concepto específico, o a distintos conceptos de un diseño previamente conocido; Por otro lado, cuando se busca un diseño novedoso, es necesario aumentar el espacio de búsqueda creando representaciones más generales y complejas (Arciszewski, Michalski, & Wnek, 1995).

Para diseños de ingeniería enfocados en optimizar la totalidad o una parte del proceso, el uso de representaciones paramétrica de sus partes es práctica común, codificando sus componentes en termino de valores binarios o continuos. Sin embargo, a medida que los sistemas se vuelven complejos, o se esperan diseños con un grado importante de originalidad. Debido a esto, se han estudiado múltiples alternativas de representaciones indirectas o "generativas" inspiradas en el proceso de morfogénesis (Hornby, 2003).

Uno de los métodos generativos para representar sistemas complejos más populares es el uso de autómatas celulares. Estos son sistemas compuestos por una configuración de células (valores), y un set de reglas de transformación que permite a estos valores variar en el tiempo, dándole la posibilidad al sistema de explorar todas las posibles combinaciones de valores de células.

La manera en que operan las reglas sobre el valor de una célula específica suele depender del valor que tengan las células aledañas a esta. Así, es posible generar un a configuración específica si se conocen las reglas de transformación y la configuración inicial adecuada. En el ejemplo de variación de células autómatas presentada a continuación, es posible observar el comportamiento en 2 períodos de un sistema de 6 células capaces de tomar 2 valores (Blanco y negro) al ser operadas por alguna de las 8 reglas de transformación. En este caso la transformación de cada célula dependerá del valor de la célula a su izquierda y derecha.

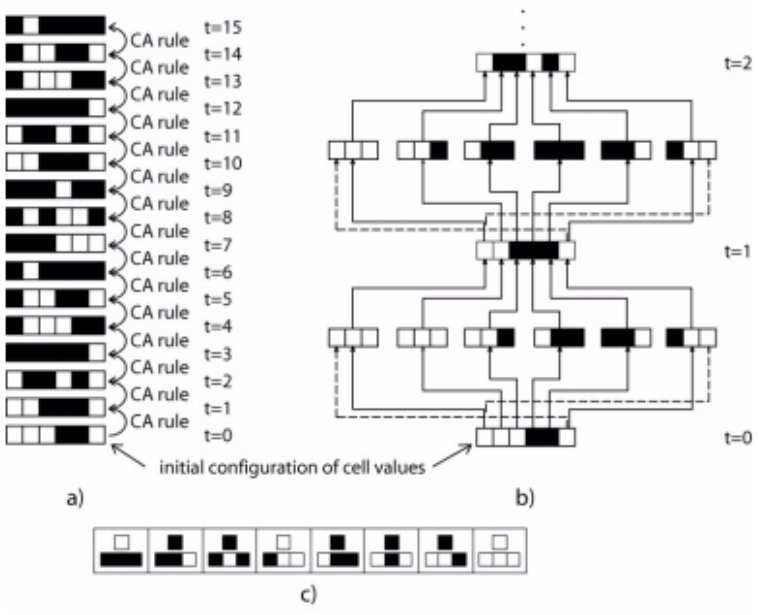

Figura 9: Proceso iterativo de célula autómata (Kicinger, Arciszewski, & De Jong, 2005)

La principal ventaja de este enfoque es la opción de reducir el número de células implicadas en la representación de la solución a un menor grupo de células en un estado inicial, junto con las reglas de transformación, permitiendo modificar la totalidad de la estructura al variar un valor único, ya sea del estado inicial o las reglas de transformación.

Por otro lado, la desventaja de este método es la escalabilidad. El aumento del tamaño de la vecindad que influirá en las reglas de transformación, o los valores que pueden tomar las células generará un aumento exponencial de la cantidad de reglas necesarias para abarcar todas las combinaciones posibles, a fin de poder explorar el espacio de candidatos a solución de manera eficaz. Ante esto surge el concepto de Autómata Celular Totalística, la cual asume que el valor de las células en el próximo período depende del promedio del valor de actual de la célula y su vecindad, y no del valor de la célula en sí (Wolfram, 2002).

Sin embargo, gracias a la manera indirecta en que el diseño es representado, la escalabilidad de los diseños con representación generativa es significativamente mayor que los representados de manera paramétrica (Hornby, 2003). Y como se verá más adelante, la calidad de los diseños generados representados de manera generativa ya sea con Células Autónomas estándar o totalísticas, puede ser mucho mayor que aquellos representados de manera paramétrica (Kicinger, Arciszewski & De Jong, 2005).

La aplicabilidad de los autómatas celulares en el diseño estructural recae en la posibilidad de separar la estructura en subsistemas de arriostramientos, vigas, columnas y apoyo junto con sus respectivas reglas de transformación, reduciendo la totalidad de los diseños generados a dichas reglas y un estado inicial de los elementos estructurales recién mencionados.

2.4.2.- Modelos

El aumento de la capacidad de cómputo y el desarrollo de nuevas tecnologías de procesamiento de datos han influido en un aumento del uso de algoritmos evolutivos no solo en el ámbito de la ingeniería, sino que en las más diversas áreas asociadas a la optimización estocástica, posibilitando la utilización en sistemas cada vez más complejos, con mayor cantidad de restricciones de diseño, y con más variables trabajando en conjunto.

Sin embargo, esto puede aumentar dramáticamente la necesidad de cómputo de los algoritmos evolutivos tradicionales, haciendo que la aplicabilidad de estos disminuya para sistemas demasiado grandes. En consecuencia, se desarrollaron modelos alternativos a los algoritmos evolutivos genéricos centrados en problemas en los que la dimensionalidad del problema sea muy alta, o en los que la interrelación de las distintas componentes del problema sea demasiado compleja.

Algoritmos multi objetivos

En sistemas complejos es recurrente tener múltiples variables, que se relacionen de manera distinta entre sí. En muchos casos, el aumento del valor de una variable genera un aumento de la calidad de la solución (Por ejemplo, el uso de un mejor hormigón aumenta la resistencia), pero genera un cambio en otra variable que disminuye dicha calidad (En el mismo ejemplo, aumenta también el coste asociado). Ante esto, se utilizan algoritmos especializado en realizar tareas con múltiples objetivos incompatibles como los que se destacan a continuación:

1. **Algoritmo genético multi objetivo (MOGA)**
   Este acercamiento propuesto por Goldberg utiliza la idea de asignar calidad utilizando un criterio de Pareto (Goldberg, 1989). Acá, se plantea el uso de un ranking de acuerdo con la posición de cada individuo respecto a la frontera Pareto formada por los mejores individuos de esa generación. Estos individuos, denominas no dominados, se interpolan de acuerdo con algún criterio (Por ejemplo, una interpolación lineal) para dotar a los individuos que no pertenecen a la frontera de

Pareto (Individuos dominados) una evaluación normalizada. Esto hace que las soluciones sean calificadas respecto a su distancia con la frontera Pareto.
La principal desventaja de esto es que se requiere una etapa adicional que permita conocer la calidad de cada individuo de acuerdo con cada función de evaluación. Además, es posible que ciertas soluciones sean sobre/sub calificadas debido a la naturaleza de las distintas funciones de evaluación y como los individuos se distribuyen en estas (i.e el dominio de dicha función).

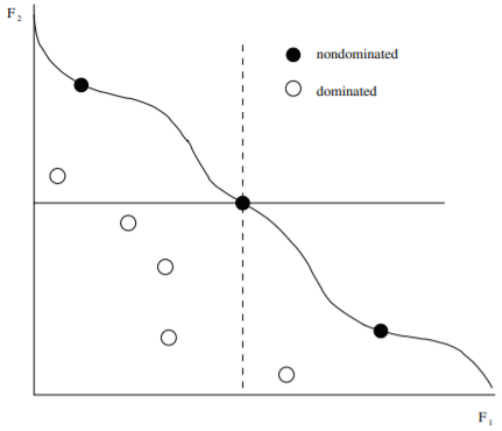

Figura 10: Ejemplo de dominancia en frontera Pareto (Coello, 2007)

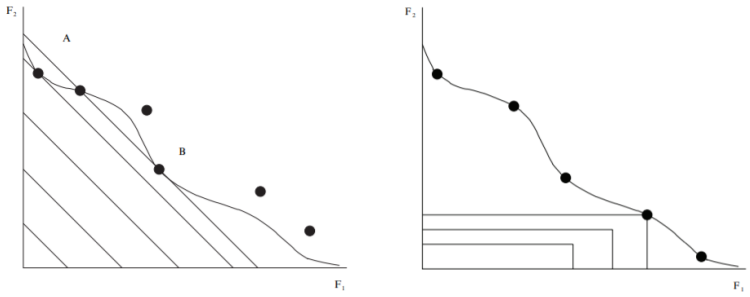

Figura 11: Métodos de interpolación lineal/bi objetivo

2. Modelo Coevolutivo

En sistemas complejos es común que su diseño sea llevado a cabo por un equipo multidisciplinario de profesionales quienes le otorgan al problema requerimientos de diseño asociados a cada disciplina. Debido a esto, se requiere una actualización manual de dichos requerimientos a medida que el proceso avanza de manera de adaptar la población de soluciones a los requerimientos de diseño actualizados, lo que puede aumentar drásticamente el tiempo de trabajo necesario para optimizar el diseño.
Una forma de evitar esta problemática es el uso de algoritmos coevolutivos. Este tipo de algoritmos es, en esencia, el algoritmo evolutivo estándar. Sin embargo, en este caso se utilizan distintas poblaciones con baja interacción entre sí, y la calidad de la solución es un valor subjetivo, pues depende de su interacción con los individuos de otra subpoblación.
Así es posible tener distintas secciones de un problema las cuales van evolucionando de manera paralela de acuerdo con criterios propios, a la vez que transforman a los individuos de otras secciones cada cierto número de iteraciones.

Este modelo se asemeja a las poblaciones de animales inter-especies cuyas características evolutivas están íntimamente relacionadas, y el tipo de relaciones que se presentan son las siguientes (Coello, 2007):

Tabla 1: Relaciones inter-especies coevolucionando

	A	B	
Neutralismo	0	0	Poblaciones A y B son independientes entre sí y no interactúan
Cooperativismo	+	+	Ambas Poblaciones se benefician de la relación
Comensalismo	+	0	Una Población se beneficia de la relación, mientras que la otra no mejora ni empeora su situación
Competición	-	-	Ambas especies tienen efectos negativos entre ellas debido a que compiten por los recursos
Depredador-Presa	+	-	El depredador (A) se beneficia mientras que la pesa (B) se ve afectada negativamente
Parasito-Huésped	+	-	Similar al modelo depredador-presa, el parásito (A) se beneficia mientras que el huésped (B) se ve afectado negativamente

La principal limitante de este método es que se requiere procesos que sean relativamente independientes entre sí, de manera de no generar limitantes al momento de separar el problema en dichos procesos, lo que reduce el universo de problemas que pueden ser efectivamente solucionados a través de este.

Algoritmos genéticos paralelos

Al tratar de modelar un sistema de ingeniería complejo pueden presentarse distintos problemas que impliquen un aumento drástico en la capacidad de cómputo requerida. Esto se puede deber al uso de grandes poblaciones de candidatos a solución, haciendo que la memoria requerida para almacenar a cada individuo sea considerable; También se ha visto problemas complejos en los que el tiempo requerido para evaluar a los individuos aumenta a 1 año CPU en una sola iteración (Luke,1997); Otra razón es el estancamiento del proceso en una región óptima local impidiendo una mejora del espacio de soluciones. Bajo esta problemática, se desarrollaron una seria de modificaciones al algoritmo evolutivo genérico, dando paso a los denominados algoritmos genéticos paralelos.

Como su nombre lo indica, esta clase de algoritmos realizan una parte o la totalidad del proceso de optimización en estructuras diferenciadas, las cuales se comunican entre ellas después de cierto número de iteraciones con el fin de compartir el conocimiento adquirido en el proceso de optimización. Esto permite que diversas unidades de procesamiento (CPUs) lleven a cabo de manera paralela el proceso, aumentando la velocidad de este. Además, esta paralelización en muchos casos implica un aumento de la calidad de las soluciones obtenidas, incluso aunque el paralelismo sea modelado artificialmente en un computador convencional.

Los algoritmos paralelos pueden clasificarse en tres tipos:

1. **Amo esclavo**
   Esta variante separa el proceso en 2 partes, siendo común aislar la etapa de evaluación del resto debido a que esta etapa duele conllevar el mayor requerimiento de cómputo. De esta forma, el Amo

lleva a cabo todo el proceso hasta obtener la población sin evaluar, para luego ser enviada al Esclavo quien ejecutará el cálculo de evaluación. Posteriormente la población evaluada es enviada de vuelta al Amo quien sigue el proceso, para luego repetir.
De esta forma, es posible obtener múltiples esclavos para realizar distintas evaluaciones lo que da pie a que los individuos de la población sean enfrentados a distintas funciones de evaluación a lo largo del proceso.

Sin embargo, esto implica la necesidad de utilizar un elemento "comunicacional" que lleve los individuos del Amo al Esclavo, y devuelva las evaluaciones de los esclavos al Amo. Para ciertos problemas este elemento comunicacional puede implicar un coste computacional, implicando que en algunos casos sea una opción peor que la realización normal. Frente existen casos en los que los esclavos llevan a cabo una parte mayor del proceso, como evaluación y reemplazo de población con el fin de aliviar los costes computacionales entre iteraciones.
Cabe destacar que el uso de distintas funciones de evaluación puede generar un efecto de cuello de botella, en que el esclavo que tiene la función de evaluación con el mayor coste computacional seguirá trabajando cuando todo el resto de los esclavos finalice su actividad. Ante esto es posible tomar 2 decisiones: Si se elige esperar a que todos los Esclavos terminen su proceso (En este caso, que todas las evaluaciones sean llevadas a cabo) antes de continuar con el Amo, se denomina proceso "Sincronizado"; En caso contrario, se puede optar por no esperar a que los Esclavos más lento terminen su proceso trabajando así sobre una fracción de la población. En este caso se denomina proceso Desincronizado.

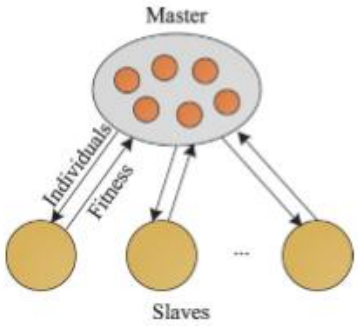

Figura 12: Ejemplo de Algoritmo Evolutivo Amo Esclavo (Yue-Jiao et al., 2015)

2. Modelos de Superposición

Esta variante de algoritmos paralelos busca alterar las condiciones en la que evolucionan los candidatos a solución, tomando en cuenta el uso de múltiples individuos para una misma evaluación o múltiples evaluaciones para un mismo individuo. Esto permite una superposición de la información logrando que la información adquirida a lo largo del proceso se traspase de iteración a iteración (Yue-Jiao et al., 2015).
Dentro de esta variante se encuentran el modelo paralelo de "Piscina". Acá se utilizan múltiples funciones de evaluación (Por lo general asociada a la cantidad de procesadores), las cuales operan sobre individuos aleatorios de la población, asegurando que la totalidad de los individuos ejecute al menos una evaluación. Esto separa a la función en distintas subpoblaciones independientes, salvo por aquellos individuos que son evaluados por más de una función. Estos últimos le otorgan al

algoritmo la superposición necesaria para que la información entre subpoblaciones se propague a la nueva generación.
Otro modelo útil es el modelo celular. En este los individuos son afectos a la misma evaluación (Es decir, sin subpoblaciones), pero solo compiten contra los individuos en su vecindad. Debido a esto, cada individuo es compite N veces, donde N es el tamaño de la vecindad, superposición que permite evaluar a la totalidad de la población. Cabe destacar que la escalabilidad de este método es baja debido a la necesidad de reevaluar a cada individuo, por lo que suele usarse cuando es posible utilizar procesadores distintos para cada individuo de la población.
Este método también posee una clasificación síncrona o asíncrona en función de si la evaluación se realiza sobre los individuos todos a la vez, o uno a uno.

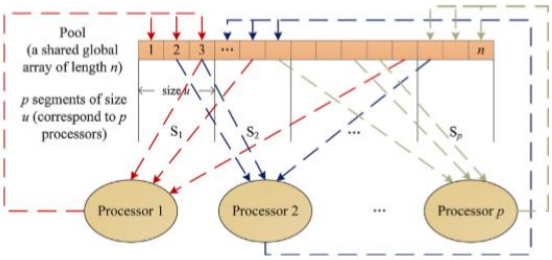

Figura 13: Modelo de Piscina

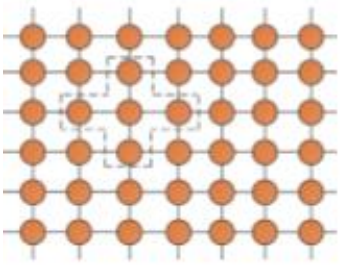
Figura 14: Modelo Celular

3. Modelos Migratorios

Finalmente, se tiene una variante en la que la separación se realiza sobre la población con el fin de lograr una "Aislación geográfica", creando subpoblaciones cuyos individuos compitan solo dentro de ellas. Existe, además, un componente migratorio que permite que individuos de una subpoblación pasen a otra, logrando un traspaso de la información adquirida durante el proceso entre subpoblaciones.
Este modelo también se clasifica como síncrono si todas las migraciones se producen en conjunto a intervalos de tiempo fijos, o asíncrono si cada población genera migraciones cuando se cumplen criterios predefinidos (Como un aumento en cierta magnitud de la calidad promedio de los individuos). Por otro lado, se diferencia entre la posibilidad de "viaje" de los individuos migratorios: El modelo migratorio más popular, el modelo de isla, permite que los individuos migrantes lleguen a cualquier otra subpoblación; Por otro lado, el modelo de escalón solo permite a los migrantes viajar a subpoblaciones dentro de la vecindad de la subpoblación de origen.

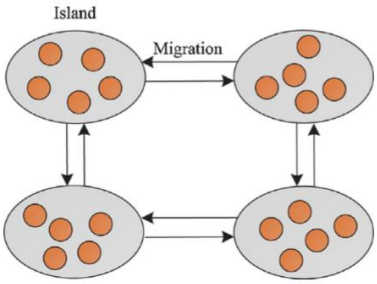

Figura 15: Modelo migratorio

2.5.- Resumen del método

Los modelos presentados engloban los distintos métodos de algoritmos evolutivos, sus características principales y posibilidad de aplicación. A partir de esto se elabora un mapa conceptual que clarifique los métodos dentro de los modelos de optimización y dentro de los algoritmos evolutivos.

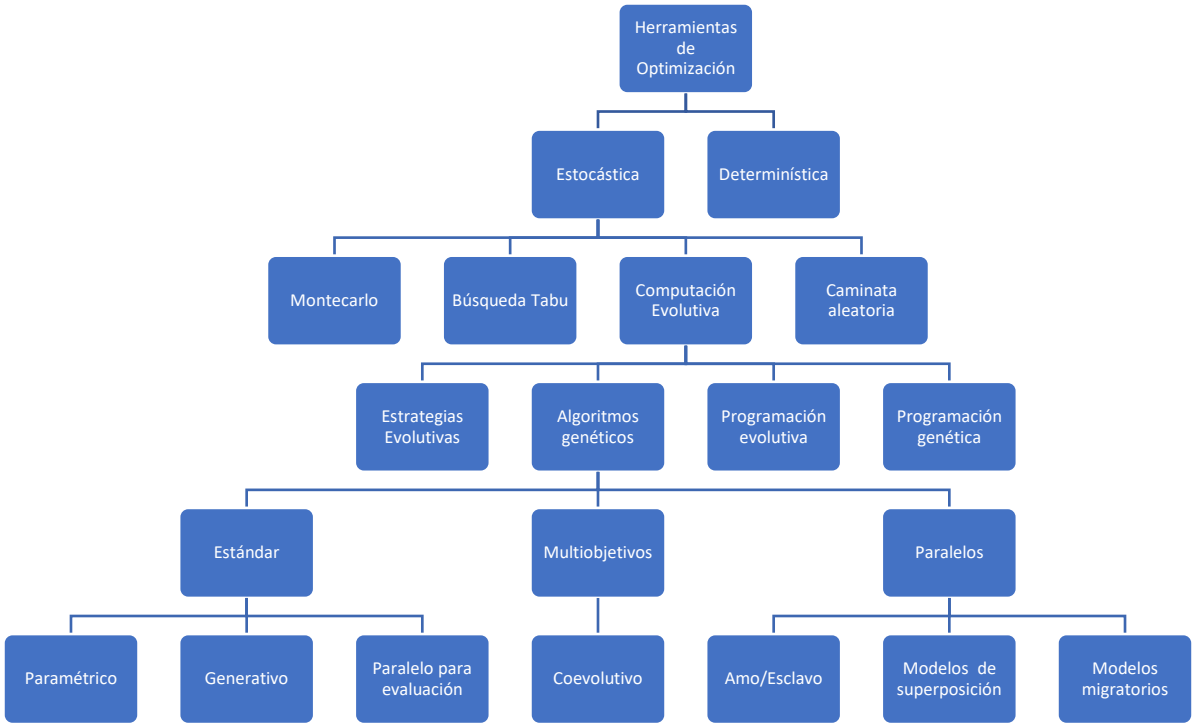

Figura 16: Mapa conceptual algoritmos evolutivos

En resumen, los algoritmos evolutivos son métodos de optimización estocástica basado en exploración. Sin embargo, a diferencia de otros métodos de optimización por exploración como el método de caminata aleatoria o el método de Montecarlo, los algoritmos evolutivos realizan la exploración en una vecindad de los candidatos a solución previamente estudiados.

Además, el algoritmo evolutivo es la conjunción de distintos métodos de optimización de manera iterativa con estrategias basadas en la conducta evolutiva de los individuos de poblaciones en la naturaleza.

Finalmente, se observa que el modelo evolutivo puede tomar distintas características que mejoren la respuesta de este frente a problemas particulares, variando tanto en la forma de representar el espacio de búsqueda de candidatos a solución, como en la posibilidad de estructurarlos dando paso a la utilización de múltiples unidades de procesamiento para realizar tareas de manera paralela.

2.6.- Aplicaciones en ingeniería estructural

Como se ha visto, al ser un método de optimización, los algoritmos evolutivos pueden ser usados en una amplia variedad de problemas de todas las disciplinas. En esta sección se ejemplificarán usos en el marco de la ingeniería estructural para distintos modelos.

1. **Diseño Generativo:** Con el fin de estudiar la respuesta del uso de representaciones generativas, tanto estándar como totalísticas frente a las representaciones paramétricas, Kicinger, Arciszewski y De Jong elaboraron patrones estructurales de edificios de acero. Los resultados indicaron que la calidad de las soluciones es significativamente mejor al usar representaciones generativas. Además, estas presentan una mayor cantidad de patrones repetitivos en su estructuración. Por otro lado, se observa que las representaciones generativas estándar tienen una menor calidad, pero una mayor variación que las representaciones totalísticas.

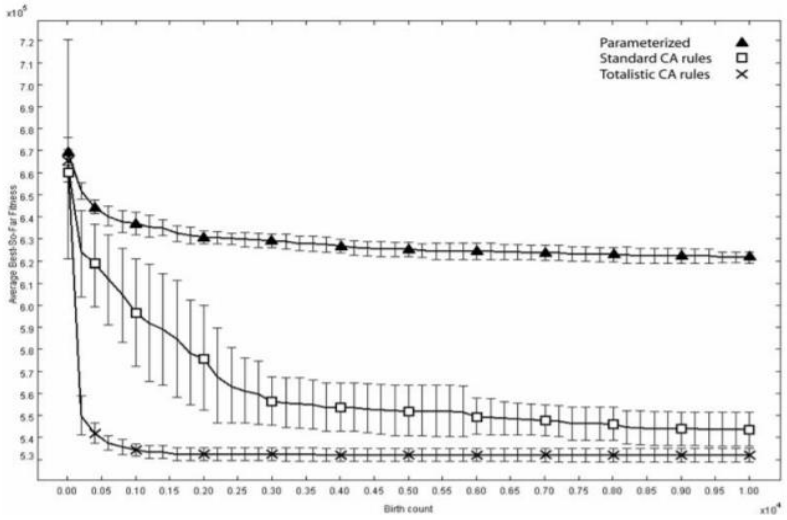

Figura 17: Puntuación de la población en el tiempo (Menos es mejor). (Kicinger, Arciszewski, & De Jong, 2005)

2. **Algoritmo genético multiobjetivo basado en entropía**

   Uno de los problemas observados en algoritmos genéticos es la aparición de "clusters" asociados a cada variable. Es decir, grupos de candidatos a solución en los cuales los miembros de cada grupo son cercanos entre ellos en término de calidad y valor de variables, a la vez que son distintos a los miembros de grupos diferentes. En otras palabras, al optimizar un diseño estructural que priorice una alta resistencia y bajo coste es posible encontrar grupos de soluciones muy resistentes, pero de precio elevado, y otro grupo de bajo coste, pero una menor resistencia.

   Esto puede resultar un problema cuando se requiere una gamma de soluciones finales a ser comparadas, por ejemplo, bajo un criterio cualitativo, en el que sean requeridas soluciones que se distribuyan de manera más uniforme a lo largo de la frontera de soluciones óptimas. Por otro lado, la habilidad de evitar esta "clusterización" de las soluciones puede ayudar escapar de óptimos locales en los que las soluciones queden atrapadas.

Una técnica común es generar soluciones adicionales para rellenar áreas no representadas a lo largo de la frontera Pareto, proyectando candidatos a solución aleatoriamente elegidos cerca de los bordes de un área no representada para rellenar esa sección. Sin embargo, juzgar si la calidad de las soluciones proyectadas mejora la calidad promedio de las soluciones a lo largo de la frontera es complicado si no se tiene una heurística clara de generación de soluciones.
Para solventar este problema de clusterización, se hace uso de una analogía al comportamiento de los gases en expansión en espacios cerrados (Farhang-Mehr & Azarm, 2002). Los gases se expanden de manera aleatoria, en el sentido de que no poseen conocimiento respecto a la geometría de su contenedor, hasta alcanzar un estado de equilibrio uniforme y homogéneo de máxima entropía. Por tanto, se busca llevar esto a la generación de soluciones para hacer que estas se adapten a su contenedor (Es decir, a las restricciones del problema) formando una estructura lo más uniforme posible, cubriendo la mayor parte e la frontera Pareto.
Para lograr esto, se pensarán a las soluciones evolucionando como partículas de un gas ideal en un espacio N-dimensional, donde N es la dimensión del espacio de soluciones, propagándose en el medio y colisionando con los bordes (Las restricciones del problema) hasta llegar a un estado de equilibro.
En primer lugar, se le asigna a cada solución una velocidad que se obtendrá a partir de una función de distribución de velocidad, análoga a la función de distribución de Maxwell. Esta velocidad se mantiene a lo largo de todo el proceso de optimización de forma similar a lo que ocurre con los gases ideales, quienes solo varían la dirección de desplazamiento al colisionar, sin cambiar su velocidad.
Posteriormente, se calcula el vector de propagación. Este vector representa la variación esperada de las mejores soluciones de una generación a la siguiente. Para esto, se separa a las soluciones en grupos de soluciones no dominadas, es decir, grupos donde todas las soluciones tengan una evaluación similar entre ellas, pero mejor que el resto. Posteriormente, se calcula un vector "A" de empuje, que representa el desplazamiento esperado de las partículas debido a la "colisión" con las partículas de su mismo grupo. El vector de propagación "C" será el vector de empuje "B" entre las partículas de un grupo y otro, menos el vector de empuje A, como se muestra en la siguiente figura:

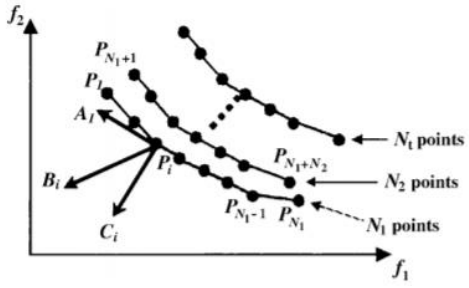

Figura 18: Vector de propagación "C"

Este vector es calculado para cada partícula. Luego, se selecciona una dirección perpendicular a dicho vector en el espacio. Este vector perpendicular representará el vector de desplazamiento que sufrirá la partícula. Así, cada solución, o un porcentaje de la población, es desplazado a lo largo de su vector desplazamiento, una magnitud que dependerá de la velocidad de estas. Esta nueva posición representa el valor de las variables de un descendiente de dicha solución.

Finalmente, se comprueba la no violación de restricciones. En caso de que la nueva partícula desplazada se encuentre fuera del espacio de soluciones, es decir, que no cumpla una restricción, se recalcula su posición asumiendo una "reflexión" al colisionar con el borde geográfico entregado por las restricciones del problema, como se observa en la figura a continuación.

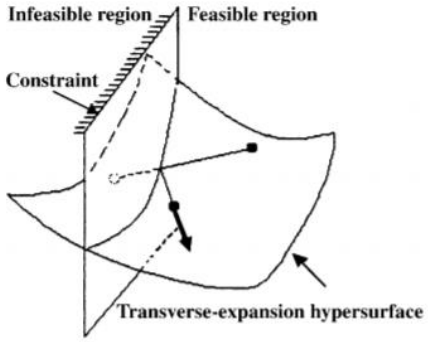

Figura 19: Colisión al violar restricciones

En resumen, esta variante modela la evolución esperada de las mejores soluciones (Las soluciones de Nivel 1) al compararlas con su posición respecto a las soluciones no tan buenas (Soluciones de Nivel 2) para luego desplazar perpendicularmente a las partículas, formando así una frontera virtual de soluciones óptimas, pero más alejadas unas de otras en comparación al método tradicional. Esta variante ha demostrado entregar soluciones bien distribuidas a lo largo de una frontera Pareto para problemas de enrejados, en comparación al método estándar.

3. **Algoritmo Coevolutivo**: Hofmeyer y Dávila evaluaron la respuesta de un algoritmo coevolutivo frente a un algoritmo genético para el diseño estructural de 2 edificios sometidos a distintas condiciones de carga. En primer lugar, un edificio de 24 metros de altura y una base de 9 x 9 metros sometido únicamente a una carga vertical. En segundo lugar, un edificio de 9 metros de altura, y una base de 24 x 9 metros, sometido a una carga combinada: vertical y de viento. El algoritmo coevolutivo utilizado optimizaba la presencia de elementos estructurales en cada piso, a la vez que optimizaba la topología por piso de acuerdo con las últimas modificaciones. De esta forma, se buscaba disminuir la energía de deformación y el volumen estructural.
La cantidad de soluciones entregadas por el algoritmo coevolutivo es muy baja en relación con el algoritmo genético, pero presentan una mejor calidad que estos últimos. Por otro lado, los algoritmos genéticos son capaces de alcanzar niveles cercanos al óptimo del algoritmo coevolutivo, pero después de un tiempo de cómputo significativamente mayor.

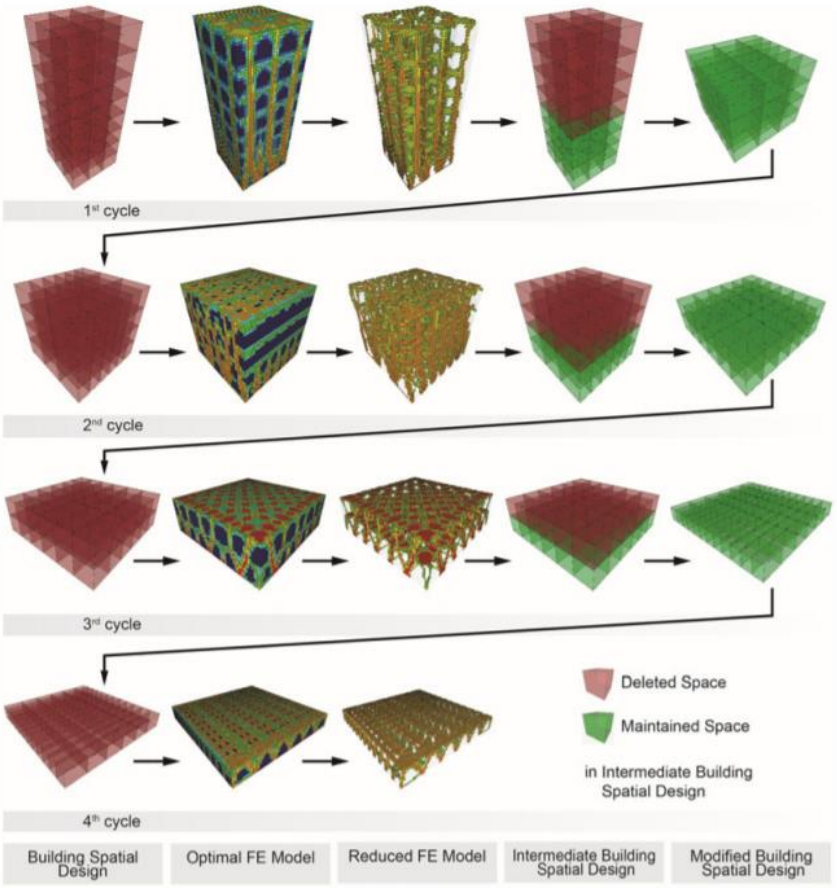

Figura 20: Esquema proceso coevolutivo

4. **Algoritmos genéticos estándar**: Abdallah y otros optimizaron la variación de altura de una viga con el fin de estudiar su impacto en el coste. Para ello separaron la viga en tramos, de modo que cada tramo disminuyera su altura en un cierto incremento fijo, y generaron un algoritmo con el fin de reducir el volumen total a utilizar, lo que implica una disminución de coste y huella de carbono. Los resultados indican que el coste debido a la optimización se reduce entre un 40% y un 52%.

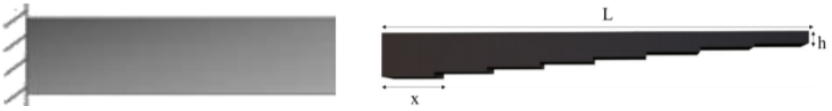

Figura 21: Viga modificada (Abdallah et al. 2019)

Por otro lado, Abdelrehim y otros investigadores desarrollaron una optimización de la sección transversal del arco de un puente utilizando algoritmos genéticos. Para esto, dividieron la estructura en elementos finitos triangulares y optimizaron la posición de los nodos de dichos elementos manera de minimizar el volumen estructural. De esta forma, fueron capaces de reducir entre un 35% y 40% el peso de la estructura en comparación a una de sección transversal estándar.

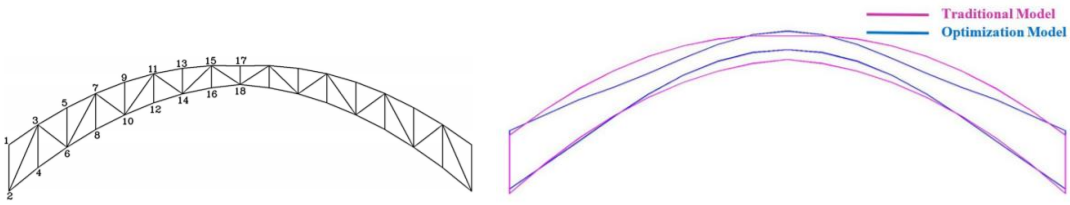

Figura 22: Estructura de arco modelada / optimizada (Abdelrehim et al. 2019)

5. **Otras aplicaciones:** A continuación, se presentan algunas aplicaciones de algoritmos evolutivos en el diseño estructural, separados en cuanto a el enfoque de aplicación (Las referencias se encuentran posterior a la bibliografía):

Tabla 2: Aplicaciones de algoritmos evolutivos en el diseño estructural

Referencia	Tipo de algoritmo	Aplicación	Aplicación genérica
1.- Kicinger (2004)	Algoritmo genético Multiobjetivo	Optimización de topología estructural de edificios de acero en altura.	Topología
2.- Kicinger (2004)	Algoritmo paralelo Migratorio		
3.- Kicinger (2004)	Algoritmo evolutivo con representación generativa		
4.- Gang Li (2010)	Algoritmo genético y criterios de optimalidad	Optimización de diseño estructural de edificios de hormigón armado en altura.	Edificio en altura Hormigón armado
5.- Pullman (2003)	Algoritmos evolutivos y *Fuzzy logic*		
6.- Sarma (2001)	Algoritmos paralelos Migratorios y de superposición, y *Fuzzy Logic*	Optimización de diseño estructural de edificios de acero en altura.	Edificio en altura acero
7.- Tomeia (2018)	Algoritmo evolutivo estándar	Optimización de diseño basado en grilla triangular	
8.- Pezeshk (2000)	Algoritmo genético estándar	Optimización de marco bidimensional con elementos discretos en rango no lineal	Marco
9.- Talaslioglu (2008)	Algoritmo evolutivo paralelo con búsqueda de intervalos	Optimización de enrejado 3D y comparación con otros métodos	Enrejado
10.- Shengli (2011)	Algoritmo genético y gradiente conjugado	Optimización de elementos de enrejado bidimensional	
11.- Johan (2019)	Algoritmo genético en diseño generativo	Optimización de enrejado según tipo de material	
12.- Zhang You (2017)	Algoritmo evolutivo estándar con chequeo estructural	Optimización de diseño de muro de corte compatible con arquitectura	Muro de Corte
13.- Tafraouta (2019)	Algoritmo evolutivo Multiobjetivo	Optimización de diseño de muro de corte en entorno BIM	

3.- Caso de estudio

Con el fin de evaluar la aplicabilidad de los algoritmos evolutivo se realiza la optimización del diseño estructural de dos plantas estructurales las cuales están compuestas por muros de hormigón armado de posición predefinida, y largo fijo o variable en la primera y segunda planta respectivamente. La finalidad del algoritmo evolutivo es optimizar la existencia y longitud de los muros en la primera y segunda planta respectivamente, con el fin de disminuir tanto el peso estructural de la planta como la respuesta de esta frente a una carga sísmica.

La primera planta es una estructura simétrica en ambos ejes y puede poseer muros en casi la totalidad de su perímetro, llegando a niveles de rigidez importantes. En este caso el algoritmo solo puede decidir en cuanto a la existencia o ausencia de muros. Estas características hacen que una gran variedad de plantas generadas presente respuestas estructurales y pesos similares, lo que puede derivar en que existan muchas estructuras similares a la óptima en cuanto a su respuesta. De esta forma es posible analizar el desarrollo del algoritmo, en particular la convergencia del proceso y la obtención de estructuras optimizadas, considerando el comportamiento exploratorio que este presenta, para así identificar posibles dificultades del algoritmo y mecanismos que deban ser modificados o replanteados.

En el caso de la segunda planta estructural, esta también posee simetría en uno de sus ejes, y además una acumulación de muros en uno de sus lados. Esto permite analizar como la selección de distintos parámetros del algoritmo afecta a las plantas entregadas por el proceso evolutivo. Cabe destacar que en esta planta existen restricciones por arquitectura, de manera que los muros no puedan tomar dimensiones tales que, por ejemplo, se introduzcan en el área definida como pasillo por arquitectura.

Las plantas de la estructura tienen por sí mismas 3 grados de libertad: Desplazamiento en las dos direcciones del plano de planta y la torción de esta. La estructura es sometida a un espectro sísmico de aceleraciones para evaluar los desplazamientos máximos entre la losa superior de la planta y la base de esta, junto con los valores torsionales. Sin embargo, debido a la baja magnitud de los desplazamientos torsionales, se trabaja con la excentricidad entre el centro de rigidez y el centro de masa de la estructura. Para obtener los niveles de respuesta se realiza un análisis de torsión en planta siguiendo el método de análisis modal espectral. El desarrollo de este método se encuentra en el ANEXO.

Debido a la relativa simplicidad de las estructuras a optimizar, se utiliza el algoritmo evolutivo canónico con el fin de disminuir los tiempos de cómputo asociados a procesos más complejos. Esto permite, además, identificar instancias en las que mecanismos propios de las variantes del algoritmo canónico pueden mejorar la respuesta del proceso. Sin embargo, se incluye una etapa que aumenta la diversidad de las poblaciones de candidatos a solución para solucionar problemas de falta de convergencia evidenciados en las primeras pruebas realizadas en este trabajo.

El esquema de trabajo del algoritmo evolutivo es el mismo para ambas estructuras, pero los mecanismos que actúan en el proceso presentan algunas diferencias debido a la variación de la codificación de ambos problemas y a las variaciones que el algoritmo puede realizar a la estructura en ambos casos. A continuación, se presenta un esquema del algoritmo evolutivo, cuyas etapas se describirán en los capítulos siguientes.

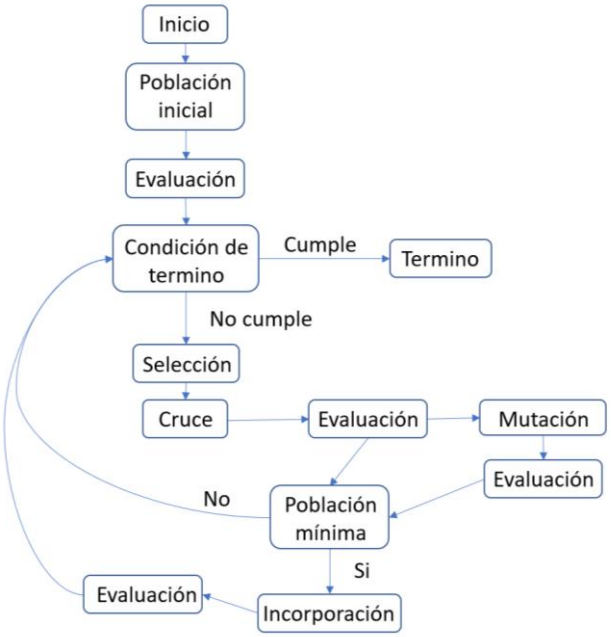

Figura 233: Esquema algoritmo evolutivo

El algoritmo tiene como objetivo encontrar aquella disposición de muros que minimice el peso estructural minimizando el porcentaje de área de muro en cada dirección principal, a la vez que reduce la excentricidad entre el centro de rigidez y el centro de masa de la planta, con el fin de reducir la torsión en planta. Además, se minimiza la respuesta de la estructura frente a una carga sísmica.

El algoritmo recibe los siguientes datos de entrada:

- Tamaño máximo de la población (100 Por defecto).
- Cantidad de sobrevivientes entre generaciones (1 Por defecto).
- Espectro sísmico de aceleraciones.
- Número de iteraciones (2000 Por defecto).
- Tasa de mutación inicial (0.3 y 0.6 Por defecto para la primera y segunda planta, respectivamente).
- Tasa de reemplazo (0.99 y 0.25 Por defecto para la primera y segunda planta, respectivamente).
- Altura de muros (2.4m Por defecto).
- Espesor de muros (0.2m Por defecto en la primera planta).

El algoritmo cuenta además con un *recorder* que almacena la puntuación del mejor individuo obtenido por el proceso hasta el momento, en cada iteración, junto con el esquema de la planta asociada al mejor individuo obtenido por el proceso completo. El algoritmo desarrolla la cantidad de iteraciones establecidas, para finalmente entregar una visualización de la evolución histórica del mejor individuo a cada iteración, el esquema del mejor individuo obtenido por el proceso, junto con los valores de excentricidad, porcentaje de área de muros y respuesta, todo para cada dirección principal de la planta.

3.1.- Caso Base

3.1.1.- Problema y estructuración

Esta estructura base corresponde a una planta de 13m x 8m en cuyo contorno se disponen muros con una longitud de 1m y altura igual a 3m. Además, se cuenta con sectores de ventanales de 2m de largo en los que no puede haber muros.

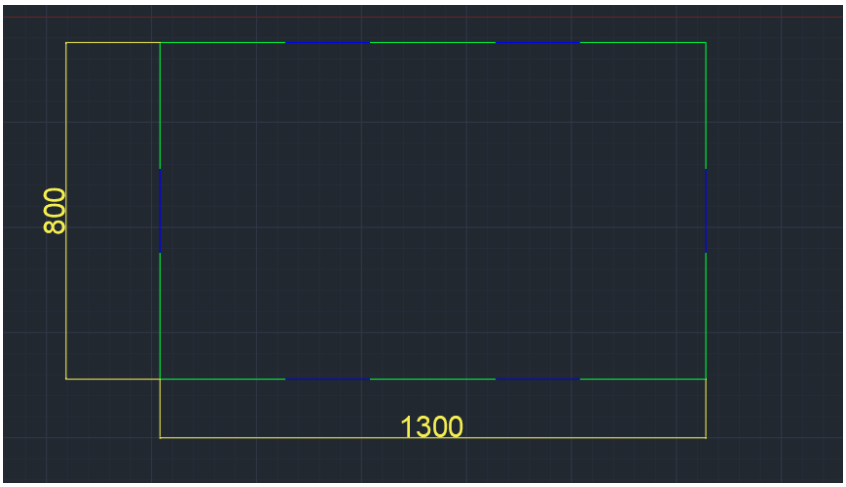

Figura 244: Estructura base. Ventanales (Azul) y Muros variables (Verde)

.

3.1.2.- Desarrollo del algoritmo

En este caso, el algoritmo busca la posición de los muros de largo predefinido igual a 1m que minimicen los criterios de porcentaje de área de muro, excentricidad y respuesta definidos previamente. Para ello, consta de las siguientes etapas y características:

Variables

El algoritmo base cuenta con 30 variables binarias, las cuales corresponden a la posibilidad de ausencia o existencia de cada muro. La disposición de muros es en sentido horario desde la esquina superior izquierda, por lo que las primeras 3 variables corresponden a la totalidad del muro vertical inferior izquierdo.

Generación

EL algoritmo contempla una función "Constructor". Esta función toma como entrada un valor P (Inicialmente igual al tamaño máximo de la población definido al inicio de la ejecución), y entrega un vector de P individuos generados aleatoriamente. Cada individuo es, a su vez, un vector con las 30 variables binarias de dicho individuo las cuales pueden ser 0 (No hay muro) o 1 (Hay Muro). Estos valores se obtienen de manera aleatoria para cada muro.
La primera generación del algoritmo es generada con dicha función, por lo que es aleatoria, y cuenta con una población de 100 individuos.

$$P\ individuos \begin{cases} [0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1] \\ [1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1] \\ \vdots \\ [1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0] \end{cases}$$

Figura 25: Ejemplo población inicial caso base

Evaluación

La función de evaluación utilizada es una suma ponderada de las características que se buscan minimizar, pero sujeto a una penalización si estás características toman valores que se alejen mucho de magnitudes límites previamente definidas. De esta forma, se trabajará con 3 características:

1) **Cuantía de muro máxima – Cuantía de muro estructura**: El porcentaje de área de muro de la estructura no tendrá penalización, ya que el valor del porcentaje máximo es el de aquella estructura que posee muros en la totalidad de su contorno, es decir, 4 muros de 3 metros de largo en vertical, y 6 muros de 3 metros de largo en horizontal.
   De esta forma, el valor de esta característica aumenta a medida que la cuantía de muro de la estructura se minimiza, que es una de las finalidades de este proceso de optimización.
2) **Desplazamiento máximo – Desplazamiento estructura:** En este caso, se restringe el desplazamiento de la estructura a un valor igual a h/1000, donde h es la altura de muros de la estructura. En caso de que el desplazamiento máximo de la estructura (En cualquier dirección) supere dicho valor, se restará el valor exponencial entre la diferencia del desplazamiento máximo de la estructura y el desplazamiento límite.
   De esta forma, el valor de esta característica aumenta a medida que el desplazamiento máximo de la estructura se minimiza, que es una de las finalidades de este proceso de optimización, mientras que toma un valor negativo, cada vez más grande en magnitud, a medida que el desplazamiento máximo de la estructura sobrepase el desplazamiento límite.
3) **Excentricidad máxima – Excentricidad estructura**: La excentricidad de la estructura en cada dirección se calcula como la diferencia entre la posición del centro de masa y el centro de rigidez, dividido por la mitad de la longitud de la estructura en dicha dirección, por lo que es un valor porcentual. La excentricidad límite varía a lo largo del proceso con el fin de controlar la velocidad de convergencia del algoritmo, y el valor inicial será del 5%. En caso de que la excentricidad de la estructura (En cualquier dirección) supere dicho valor, se restará el valor exponencial entre la diferencia de la excentricidad de la estructura y la excentricidad límite.
   De esta forma, el valor de esta característica aumenta a medida que la excentricidad de la estructura se minimiza, que es una de las finalidades de este proceso de optimización, mientras que toma un valor negativo, cada vez más grande en magnitud, a medida que la excentricidad de la estructura sobrepase la excentricidad límite.

En resumen, las características de desplazamiento y excentricidad se formulan de la siguiente manera:

$$D = \begin{cases} \frac{h}{1000} - desplazamiento\ máximo & si\ desplazamiento\ máximo < \frac{h}{1000} \\ -e^{desplazamiento\ máximo - \frac{h}{1000}} & si\ desplazamiento\ máximo > \frac{h}{1000} \end{cases}$$

$$E = \begin{cases} 5\% - excentricidad\ máxima & si\ excentricidad\ máxima < 5\% \\ -e^{excentricidad\ máxima - 5\%} & si\ excentricidad\ máxima > 5\% \end{cases}$$

$$P = (Cuantía\ muro\ máxima\ - Cuantía\ muro\ estructura)$$

$$f = F_{dx}D_x + F_{dy}D_y + F_{ex}E_x + F_{ey}E_y + F_{px}P_x + F_{py}P_y$$

Figura 26: Función de evaluación

Donde F_{di}, F_{ei}, F_{pi} son factores de importancia. Los factores de importancia utilizados fueron obtenidos como se detalla a continuación.

Evaluación (Factores de importancia)

En primer lugar, los factores de importancia se fijan en 1 de manera de observar qué parámetros son optimizados con mayor énfasis por el algoritmo, y a cuáles les asigna menor importancia. Posteriormente se utiliza la función Constructor para generar 1000 individuos aleatorios. Estos individuos son evaluados y se clasifican en 3 grupos: Mejor 5% de la población, Peor 5% de la población, y el restante 90%. Los resultados de dicha evaluación se muestran a continuación:

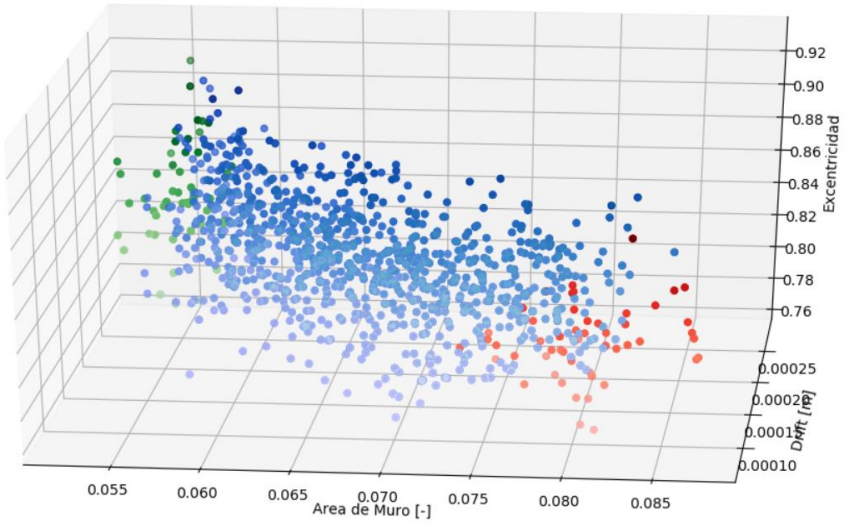

Figura 27: Evaluación de 1000 individuos aleatorios. $F_d = 1, F_e = 1, F_p = 1$

Analizando los resultados, se observa una distribución relativamente homogénea de los individuos, en que el mejor 5% de los resultados (Verdes) se encuentra en cuantías de muro bajas, mientras que el peor 5% de los resultados (Rojos) en cuantías estructurales altas. Sin embargo, no se observan diferencias en cuanto a distintos valores de excentricidad y desplazamientos. Esto se debe a que la variación del valor de la cuantía de muro entre individuos es de una magnitud mayor a la que presentan los desplazamientos y excentricidad, por lo que el valor de la función evaluación se ve fuertemente influenciado por dicho puntaje.

Debido a esto, se realizan pruebas con distintos valores de factores de importancia hasta alcanzar uno que sea capaz de identificar de mejor manera aquellos individuos que responden mejor a los objetivos del proceso evolutivo. Esto se logra con los factores de importancia $F_d = 10, F_e = 5, F_p = 1$. Es decir, el desplazamiento es 10 veces más influyente que el peso en la función de evaluación, y 2 veces más influyente que la excentricidad. Con estos valores de factores de influencia, se evalúan nuevamente 1000 individuos generados aleatoriamente y se muestran sus resultados.

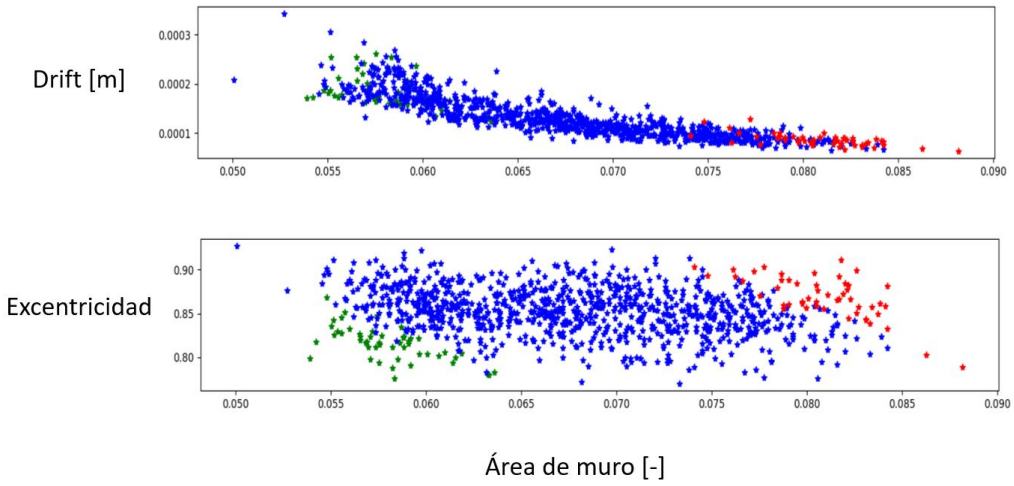

Figura 28: Evaluación de 1000 individuos aleatorios. $F_d = 10, F_e = 5, F_p = 1$

En este caso se observa que el mejor 5% de la población se encuentra en valores bajos de excentricidad, desplazamiento y peso estructural, mientras que el peor 5% de la población se encuentra en valores altos de desplazamiento, excentricidad y peso estructural. Se desprende, por tanto, que estos valores de factores de importancia responden de manera correcta al proceso de evaluación.

Al usar estas características, la función de evaluación aumenta su valor al evaluar las mejores soluciones, por lo que este será un proceso de maximización de puntuación. La función objetivo final se formula de la siguiente manera:

$$f = 10D_x + 10D_y + 5E_x + 5E_y + P_x + P_y$$

Selección

La selección de los mejores individuos que pasarán a ser parte de la siguiente generación se hace utilizando el método de ruleta proporcional. Así, se normalizará la puntuación de todos los individuos respecto a la puntuación total de la población, y posteriormente se seleccionarán 2 individuos, los cuales se cruzarán mediante una recombinación binaria para generar a los nuevos individuos de la población. Esta selección se hace de manera aleatoria, pero cada padre tiene una probabilidad de ser elegido igual a su puntuación normalizada. Este proceso se repite hasta alcanzar N pares de padres, los cuales son entregados en un vector de "Ancestros".

Sin embargo, para asegurar la diversidad del proceso y evitar convergencias prematuras, cada vez que un par de padres es seleccionado se evalúa la diferencia entre estos. Para ello, verifica en posean al menos un 10% de su genoma distinto entre ellos, lo que corresponde a 3 muros diferentes. En caso de que la diferencia entre estos no se cumpla, se seleccionará un nuevo par de padres. Esto se repite hasta que el par de padres seleccionado posea este nivel de diferencia, o hasta que se hayan realizado 20 intentos. Si en estos intentos no se logra encontrar un par de padres diferentes, se asume que la población es lo suficientemente homogénea como para no poder encontrar un par diferente, por lo que los padres seleccionados pasan directamente al vector de ancestros.

La cantidad de pares de padres seleccionados es igual al 50% del valor obtenido al multiplicar el tamaño de la población por la tasa de reemplazo. En este caso, al ser la tasa de reemplazo igual a 99%, la cantidad de pares de padres inicial será del 49% de la población original, y dado que cada par de padres genera un par de nuevos individuos, el tamaño de la población en la siguiente iteración será un 98% al tamaño original. De esta forma, es posible asociar la tasa de reemplazo con la tasa de disminución del tamaño de la población entre iteraciones. Así, es posible modificar la tasa de reemplazo para obtener niveles de reducción diferentes.

La ventaja de utilizar este método radica en la metodología utilizada para las etapas de cruce e incorporación explicitadas en los párrafos siguientes. La capacidad de convergencia del mecanismo de ruleta proporcional respecto al resto de los métodos de selección combinada con la capacidad exploratoria otorgada por el mecanismo de cruce e incorporación hacen que el algoritmo pueda desarrollarse de manera eficiente. Además, este método de sección requiere menor tiempo de cómputo que el resto de los métodos, y es muy simple de implementar.

A continuación, se presenta un esquema del proceso de selección de padres.

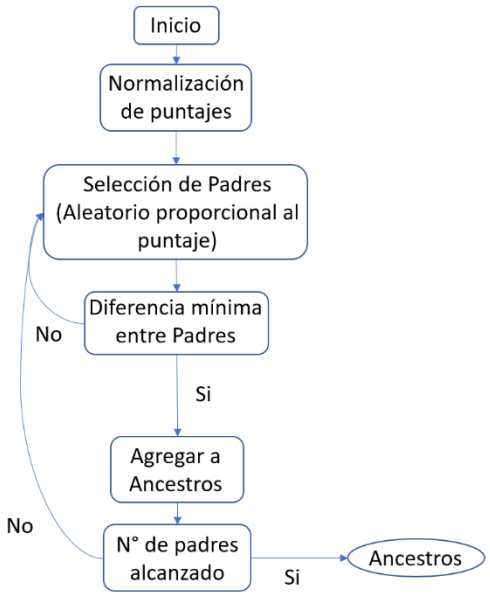

Figura 29: Esquema de selección de ancestros

Cruce

Para formar los individuos de la nueva generación, se utiliza el mecanismo de recombinación binaria. A través de este método cada se selecciona cada par de padres entregado en el vector de ancestros y se genera a partir de cada pareja dos nuevos individuos descendientes.

La manera en que se generan los nuevos descendientes es la siguiente: Cada elemento (Muro) de los nuevos individuos descendientes será obtenido desde uno de los padres de manera aleatoria, mientras que el otro descendiente recibe el muro del padre no seleccionado. Esto garantiza que los muros que serán seleccionados varíen a lo largo del proceso de manera aleatoria, por lo que el mismo par de padres, si bien sólo genera dos nuevos individuos, tiene la posibilidad de generar múltiples pares de individuos diferentes.

En particular, dado que la codificación de este problema consta de 30 variables (Muros), es posible generar 2^{30} pares de nuevos descendientes a partir del mismo par de ancestros.

Pese a que este método requiere la utilización de un paso adicional para poder seleccionar a los padres de manera aleatoria, le otorga al proceso evolutivo una mayor capacidad exploratoria, a la vez que ayuda a la heterogeneidad de las poblaciones finales del proceso. Incluir menos variabilidad entre los posibles descendientes obtenidos a partir del mismo par de padres, genera la posibilidad de que los nuevos individuos creados sean muy similares a individuos explorados en fase previas del proceso evolutivo, haciendo que aumente el coste computacional del proceso sin aumentar la exploración de nuevos candidatos a solución.

Finalizada la generación de individuos de todos los pares de padres seleccionados en la etapa anterior, se evalúa esta nueva población de individuos. A continuación, se presenta un esquema del método de cruce.

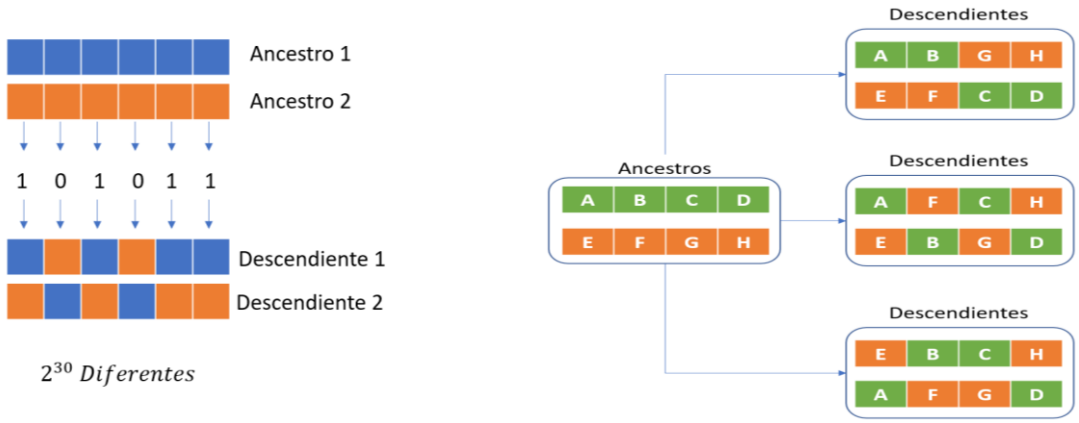

Figura 30: Esquema mecanismo de cruce

Mutación

Cada uno de los nuevos individuos generados en el proceso de cruce pueden pasar por un proceso de mutación con una probabilidad igual a la tasa de mutación previamente definida como 30%. En dicho caso, este individuo cambia alguno de sus componentes de manera aleatoria, es decir, incorpora un nuevo muro donde antes no existía o elimina un muro preexistente. Posteriormente el individuo mutado es evaluado y reincorporado a la población.

Reemplazo

La nueva generación será la unión entre los individuos mejor evaluados de la generación anterior, y los individuos creados en el proceso de cruce después del proceso de mutación, en caso de ocurrir. Por tanto, dado que la nueva generación reemplaza sólo a una fracción de la generación anterior, este será un proceso de reemplazo estacionario. La cantidad de individuos seleccionados para mantenerse desde la generación anterior a la nueva fue definida al inicio del proceso como “Cantidad de sobrevivientes”.

Incorporación

Con el fin de dotar al proceso de una mayor diversidad, la cantidad de individuos seleccionados para cruce, los seleccionados a la próxima generación, y los generados a partir del cruce están establecidos de manera que la población disminuya de generación en generación. Debido a esto, se establece una población mínima fijada como el 50% de la población original. En caso de que el tamaño de la población obtenida posterior a la etapa de reemplazo sea inferior a este valor, se generan, evalúan e incorporan a la nueva población nuevos individuos de generados de manera aleatoria a través de la función Constructor hasta que esta población tenga el mismo tamaño que la población original.

Esta etapa puede verse como una mutación en masa, lo que permite evitar óptimos locales, disminuir la homogeneidad de la población y la convergencia prematura y explorar el espacio de soluciones de mejor manera.

Condición de Término

La población final, ya sea la generada a partir del reemplazo, o la generada post incorporación, se utiliza como nueva generación para repetir el proceso nuevamente. Esto se repite hasta que se cumpla el número de iteraciones definido como 4000.

3.1.3.- Resultados

Al analizar los candidatos a solución generados por el algoritmo, se observa que estas se centran colocar muros de manera que la distribución de área de muros sea simétrica en cuanto a magnitud, pero sin hacer hincapié en una simetría geométrica de estos. Es más, se generan estructuraciones en que la cantidad de muros es igual a ambos lados de la división diagonal de la planta, haciendo que la cantidad de muros en las caras superior e izquierda combinadas sea la misma que en la cara inferior y derechas combinadas. Esto hace que, pese a que no se presente simetría respecto a las direcciones principales de la planta, la excentricidad del centro de rigidez (Punto azul) respecto al centro de masa (Punto rojo) sea casi nula.

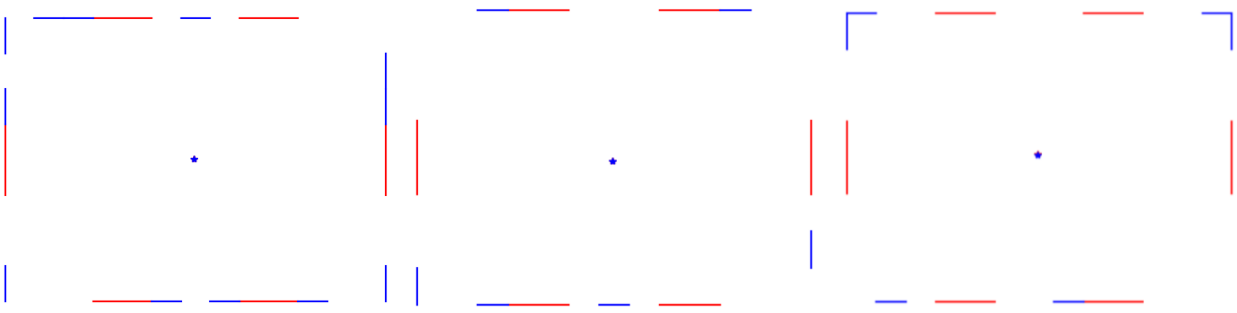

Figura 31: Candidatos a solución caso base

Además, se observa que en ningún caso se utiliza la longitud de muro completa, lo que deriva del intento del proceso en disminuir el área de muro para aumentar el puntaje sin un aumento de los desplazamientos que afecte negativamente al mismo.

Por otro lado, con el fin de analizar los puntajes de los mejores candidatos a solución a lo largo del proceso evolutivo, se ejecuta el algoritmo veces y se grafica la evolución del puntaje del mejor individuo obtenido por el proceso hasta el momento en cada iteración. Así, es posible observar estancamientos en el que no se presentan cambios, para posteriormente ver "saltos" en los que el puntaje aumenta considerablemente. Estos momentos de estancamiento representan instancias de exploración sobre el espacio de soluciones a través de las mutaciones y el proceso de cruce, y puede deberse a la caída en óptimos locales que son evitados por el proceso luego de múltiples iteraciones, cuando logra encontrar a un candidato a solución que mejore la respuesta a la encontrada en dicho óptimo local.

Además, para comparar el desempeño del método evolutivo frente a uno de exploración aleatoria, cada 200 iteraciones se generan y evalúa una cantidad de individuos aleatorios igual a la cantidad de individuos que ha sido analizado por el algoritmo evolutivo hasta ese momento. De esta forma, luego de las primeras 200 iteraciones con 100 individuos iniciales se han testeado aproximadamente 17000 individuos, por lo que se generan (utilizando la función Constructor) 17000 nuevos individuos, se evalúan, y se grafica el puntaje del mejor. Posteriormente se realiza con los siguientes 17000 individuos, y se grafica el puntaje del mejor entre ellos solo si es mejor que el mejor obtenido hasta el momento. Esto se hace hasta las 4000 iteraciones y se representa por la línea roja en el gráfico.

A través de esto, es posible apreciar que hasta las 2000 iteraciones es posible que el proceso evolutivo genere soluciones subóptimas en comparación a las que podrían ser entregadas por un mecanismo de exploración aleatorio. Sin embargo, a medida que la cantidad de iteraciones aumenta, la diferencia de puntuación entre el proceso evolutivo y la exploración aleatoria crece de igual manera.

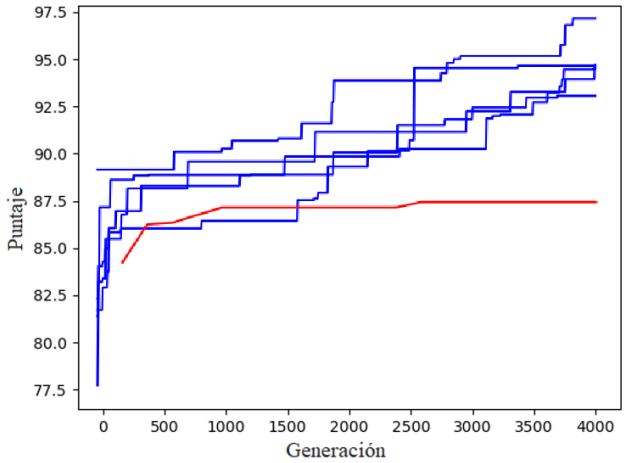

Figura 32: Puntaje del mejor individuo

Posteriormente, se realiza este mismo ejercicio al cambiar el número de sobrevivientes al doble o al aumentar el número de iteraciones. Los resultados se grafican a continuación:

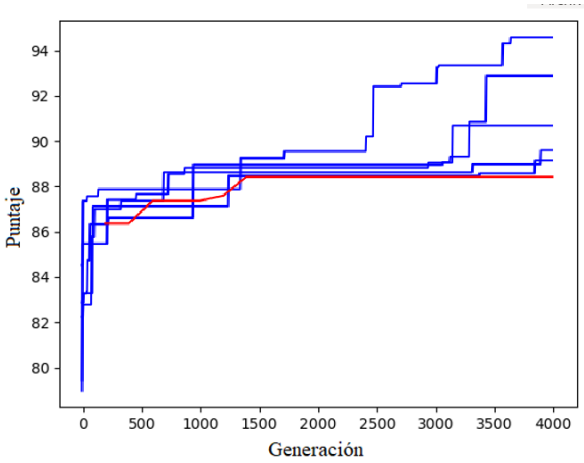

Figura 33: Puntaje Mejor individuo Sobrevivientes x2

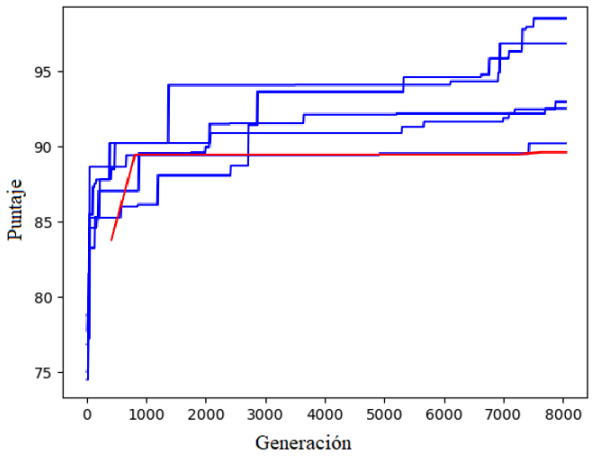

Figura 34: Puntaje Mejor Individuo Generaciones x2

En primer lugar, se observa un peor comportamiento al aumentar el número de sobrevivientes de una generación a otra, pese a que los individuos de la población inicial tienen un mejor puntaje que en caso base. Esto se debe al estancamiento observado en la fase media del proceso que tiene una duración de aproximadamente 1000 iteraciones. Este estancamiento es producido por la convergencia prematura hacia los mejores individuos, los cuales ahora sobreviven durante más tiempo en la población ocasionando que esta se asemeje más a ellos. El estancamiento se debe a que dichos individuos pueden representar un óptimo local, al cual el algoritmo logra escapar en las fases más tardías del proceso a través de mecanismos de exploración. En la etapa final se observan aumentos considerables de la puntuación, a partir de este punto el proceso evolutivo se desarrolla de mejor manera que un mecanismo de exploración aleatoria.

Por otro lado, al aumentar el número de iteraciones sin variar el número de individuos por población se observa una distribución relativamente similar del aumento de puntajes a lo largo del proceso en comparación con el caso base, salvo por un estancamiento en la fase media del proceso.

Los grandes aumentos de puntaje pueden deberse a una mutación que mejore, por ejemplo, los niveles de simetría de la estructura, o a una inyección de individuos en los que esté incluida una solución mejor a las obtenidas en el proceso. En contraparte, las pequeñas variaciones en pocas iteraciones son producto de la mutación de los individuos, generando pequeños cambios en la estructura que impactan de baja manera a la puntación de dicho individuo.

Al aumento de iteraciones sólo le puede ser atribuido un aumento del puntaje final del proceso al tener más tiempo para buscar el óptimo. Alcanzar niveles similares a los obtenidos en el caso final puede indicar un estancamiento en un óptimo (Global o local), o a un mayor gasto de tiempo en producto de la distribución homogénea en la que fueron requeridas más iteraciones para tener lo mismos niveles de aumentos de puntaje que en el caso inicial.

3.2.- Caso Estudio

3.2.1.- Problema y estructuración

El caso estudio corresponde a una planta de 8m por 37m en los que disponen muros de largo variable en ambas direcciones. El modelo cuenta con muros predefinidos que no pueden ser modificados como es el caso de los muros de la caja de ascensores. Además, cuenta con sectores de balcón en los que no pueden disponerse muros. A continuación, se presenta un esquema con la estructuración.

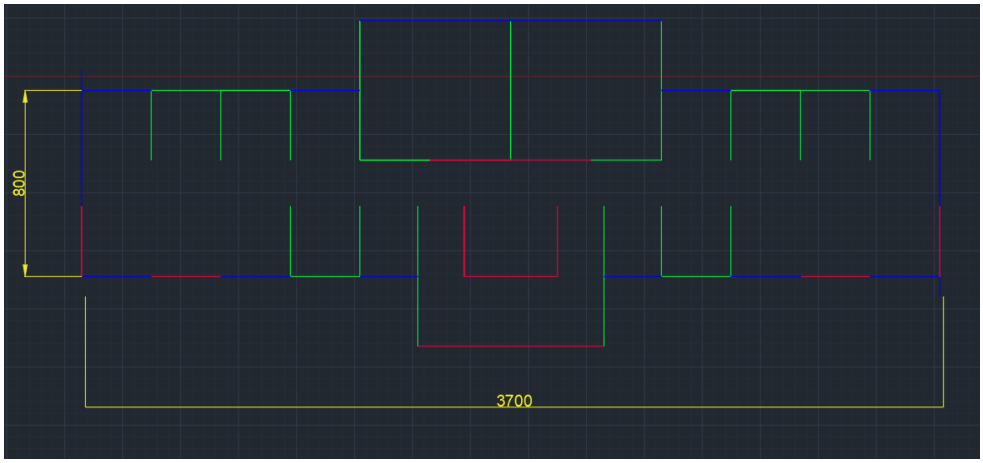

Figura 35: Esquema de planta de estructura de caso estudio. Muros variables (Verde), Muros Fijos (Rojos), Terrazas (Azul).

3.2.2.- Desarrollo del algoritmo

El esquema general del algoritmo no presenta diferencias destacables en cuanto los procesos, sino más bien en las variables y algunas operaciones.

Variables

El algoritmo de estudio cuenta con un genoma de 33 variables continuas. De estas, el último elemento del genoma corresponde al espesor de muros de la estructura, que puede tomar valores de 15cm, 20cm, 25cm, 30cm; nueve elementos corresponden a muros que no pueden ser variados, con longitudes que van desde los 3m hasta los 8m; los 23 elementos restantes corresponden a muros variables que pueden o no existir, y de hacerlo pueden tener longitudes de hasta 3m, con excepción de cinco muros que pueden llegar a los 6m.

Generación

Se mantiene el uso de la función “Constructor”, la cual es modificada para poder parametrizar el problema actual. En este caso para cada elemento variable (Muro) se le asigna un número aleatorio entre 0 y N de máximo un decimal, donde N es el largo máximo que puede tener dicho muro. Posteriormente, se agregan las componentes de los muros fijos, preestablecidas con largos entre los 3m y 8m. Finalmente, se asigna uno de los espesores de muro posibles (0.15m, 0.2m, 0.25m, 0.3m) de manera aleatoria al final de la codificación. De este modo la población se representa como se muestra a continuación:

[0, 1, 1.2, 3, 4.1, 5, 1, 0, 2, 3, 2.1, 0, 4.2, 5.1, 6, 1, 0, 2, 1.1, 2.8, 3, 0, 1, 3, 3, 3, 3, 3, 7, 4, 8, 3, 0.15]
[1.1, 0, 1.5, 0, 5, 5.2, 0, 0, 1, 2.4, 1.6, 0, 5, 4.1, 0, 2, 1, 0, 1.7, 1.8, 0, 0, 2, 3, 3, 3, 3, 3, 7, 4, 8, 3, 0.3]
⋮
[2, 1.5, 0, 2, 5, 2, 1, 2, 1.2, 0, 3, 1.1, 6, 0, 6, 1.8, 2, 0, 2.1, 3, 0, 1.9., 1, 3, 3, 3, 3, 3, 7, 4, 8, 3, 0.2]

Figura 36: Ejemplo población inicial

Evaluación

Se establecen los mismos valores para los factores de importancia que operan en la función de evaluación, por lo que la función final es:

$$f = 10D_X + 10D_y + 5E_X + 5E_Y + P_y + P_X$$

Selección

El mecanismo de selección actúa de la misma manera. Sin embargo, con el fin de controlar las incorporaciones de individuos en masa al alcanzar los niveles mínimos de población exigidos, la cantidad de ancestros seleccionados ahora varía a lo largo del proceso. Así, en las primeras generaciones la cantidad de pares de ancestros escogidos es igual al 25% del total. Esto pues dado que cada par de ancestros generan dos nuevos individuos, la siguiente generación tendrá aproximadamente un 50% del tamaño que la población original. Así, como la incorporación de individuos se realiza al llegar a la mitad de la población de inicio, en las primeras etapas se realizan incorporaciones de individuos aleatorios en todas las iteraciones.

La cantidad de pares de ancestros aumenta hasta llegar a un 50% en las últimas generaciones. En este caso, los ancestros generan una cantidad de nuevos individuos igual a la población original, por lo que el tamaño se la población se mantiene constante y no se producen incorporaciones de individuos en ninguna generación.

Gracias a esto, en las primeras etapas se produce una actualización constante de los individuos, no sólo por la gran cantidad de individuos de una generación que no es capaza de pasar sus características a la próxima generación, sino que se integra una cantidad de individuos aleatorios. Todo que se traduce en una mayor exploración del espacio de soluciones.

En contraparte, en las etapas finales la actualización cesa y comienza un proceso de convergencia a los mejores individuos producto de la no incorporación de nuevos individuos aleatorios y al traspaso de información a la próxima generación de los individuos mejor evaluados a través de cruces entre ellos.

Finalmente, se hace una modificación a la forma de evaluar la diferencia entre individuos para contrarrestar el hecho de que los muros tienen un largo variable, por lo que la diferencia entre un muro inexistente (L = 0) y uno corto (L = 1) debe ser menor a la diferencia entre un muro inexistente y uno de largo máximo (L = 3 / 6). Para ello, se calcula la diferencia porcentual de largo de muros, de acuerdo con:

$$P = \frac{Muro\ mayor - Muro\ menor}{Muro\ máximo\ posible}$$

Posteriormente, se calcula el promedio de estos valores. Este valor promedio representa la diferencia relativa entre el “mayor” de los padres y el “menor” de los padres. Sólo se acepta un par de padres cuando este valor esta diferencia es mayor al 10%, es decir, cuando $P_{promedio} > 1.1$

Cruce

No se realizan cambios al mecanismo de cruce. En este caso al ser 24 elementos variables (Los otros 9 son muros de largo predefinido), cada par de padres puede generar 2^{24} pares de individuos diferentes.

Mutación

Los niveles de mutación ahora varían a lo largo del proceso. Al igual que en el mecanismo de selección se impone una tasa de mutación del 60% al inicio del proceso con el fin de mejorar la exploración en las etapas iniciales. Esta tasa de mutación decae hasta llegar a un 30% en las etapas finales del proceso con el fin de permitir una exploración, pero sin perjudicar a la convergencia al mejor individuo.

Reemplazo

El mecanismo de reemplazo no varía.

Incorporación

El mecanismo de incorporación no varía.

Condición de Término

La condición de termino se reduce a 2000 iteraciones debido a los mayores costes computacionales requeridos para desarrollar este modelo.

3.2.3.- Resultados

Al analizar la evolución del mejor individuo en cada generación se observa una gran mejora al inicio del proceso, y luego un estancamiento general. Es posible notar una disminución en los niveles de aumento del puntaje a medida que el proceso avanza, lo que se corresponde con la mayor influencia que toma la exploración a través de la mutación dejando de lado la exploración por incorporación de nuevos individuos.

Esto también permite explicar el estancamiento al final del proceso, en el que la disminución de mecanismos de exploración y un aumento de la convergencia hacen que el proceso tienda al óptimo, ya sea global o local.

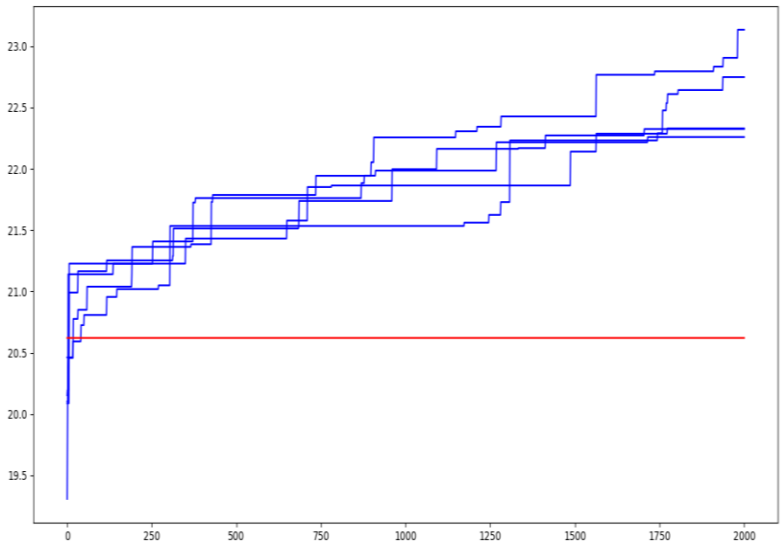

Figura 37: Puntaje mejor individuo

También se realizan pruebas sobre una mayor población para cada generación. En este caso se observa como el puntaje aumenta rápidamente para quedar estancado. Este estancamiento se mantiene a lo largo de una parte importante del proceso hasta las etapas finales, lo que puede coincidir con la obtención de un nuevo mejor individuo a través de mutaciones.

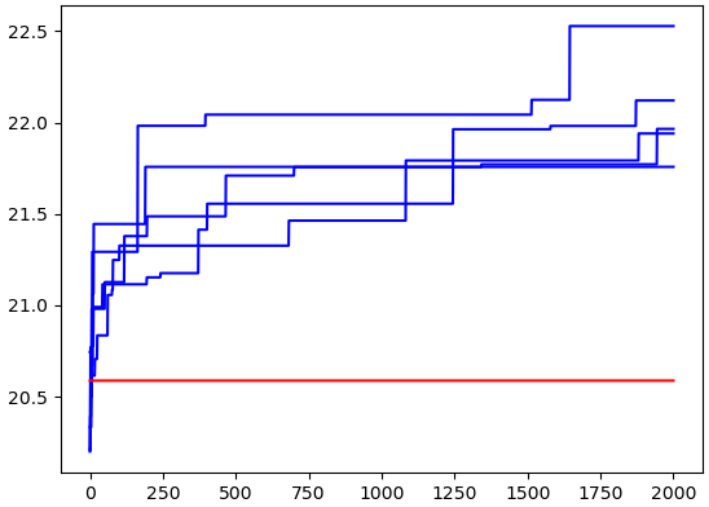

Figura 38: Puntaje mejor individuo.
Población = 200

Finalmente, se evalúa el impacto de la variación en la población mantenida de generación a generación. Al analizar la respuesta al mantener al 10% de la población original se observa un aumento más paulatino del puntaje, lo que puede deberse a que existen más individuos que se mantienen entre generación dotando de más subespacios de búsqueda explorables por medio de mutaciones. Se observa también una reducción de los niveles de variación a lo largo del proceso.

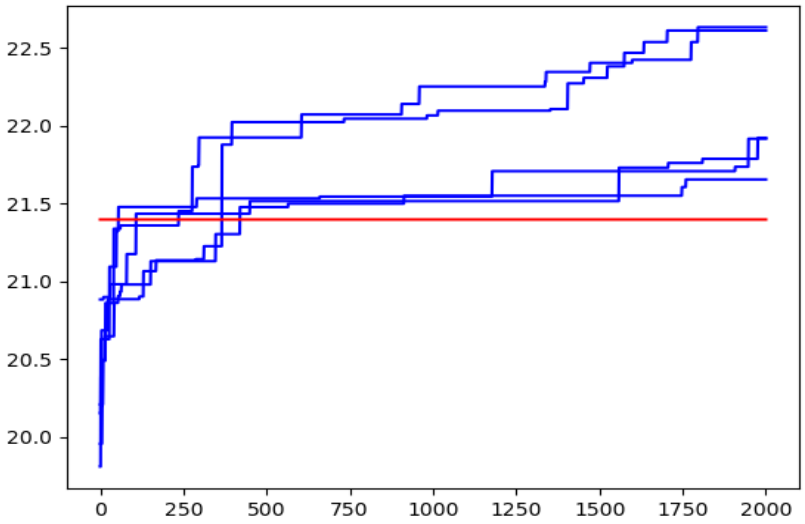

Figura 39: Puntaje mejor individuo.
Mantención del 10%

En cuanto a los candidatos a solución entregados por el algoritmo es posible observar un esfuerzo por garantizar simetría, pese a que no está programado para ello. Esto se explica pues la estructura no presenta ningún nivel de simetría en uno de sus ejes, generando importantes excentricidades en dicha dirección, por lo que una estructuración simétrica en el otro eje ayuda a reducir los niveles finales de excentricidad y aumentar la puntuación.

Se observa, además, el uso de muros de menor espesor en aquellas estructuras con menor grado de simetría, en un afán de minimizar los niveles de peso estructural para contrarrestar la pérdida de puntaje debido a la excentricidad.

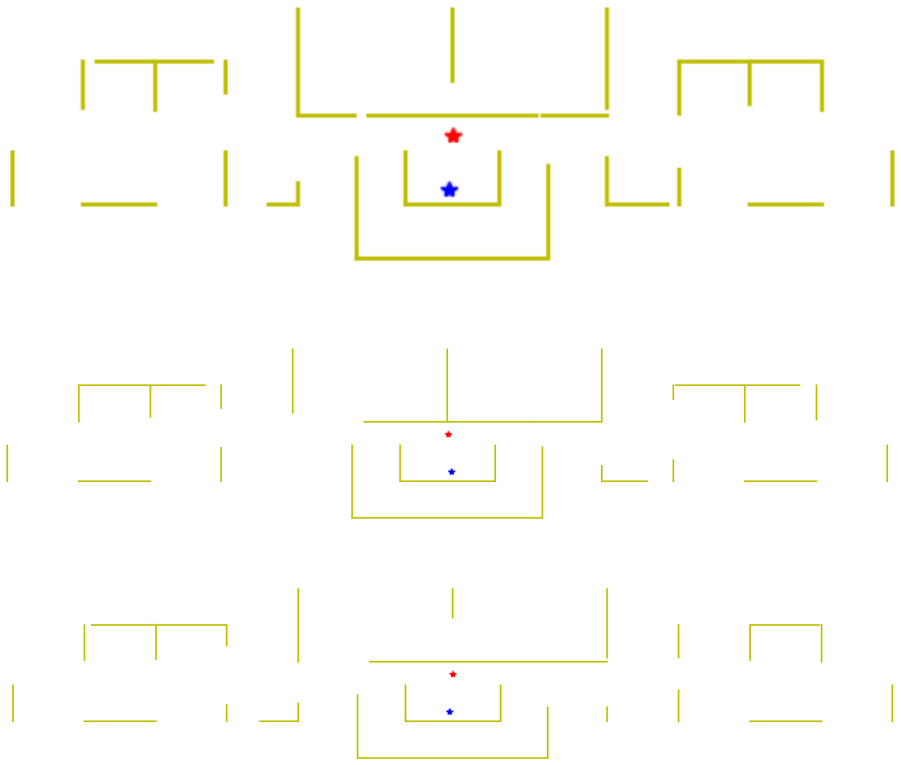

Figura 40: Estructuraciones caso de estudio

4.- Cierre y conclusiones

4.1.- Estado del arte

En los últimos años se ha visto un aumento en el uso de tecnologías relacionadas al campo de la inteligencia artificial en el área del diseño estructural y construcción, en las que el machine learning se ha vuelto una de las arquitecturas de optimización más populares en el último tiempo. Sin embargo, las dificultades propias de usar este y otros métodos de inteligencia artificial hacen que sea más atractivo el uso de técnicas de optimización por exploración. En base a eso, este trabajó abordó las principales componentes de los algoritmos evolutivos utilizados en el diseño estructural, desde sus orígenes como método de optimización hasta aplicaciones prácticas en el área, considerando tanto su posición en el campo de la inteligencia artificial como similitudes, diferencias y ventajas respecto a otros tipos de métodos de optimización centrados en el diseño.

Se identificaron y detallaron los diferentes mecanismos que se ven involucrados en un proceso optimización evolutivo y como estos afectan en los resultados obtenidos tanto en la homogeneidad de dichos resultados como en la velocidad de convergencia hacia el óptimo global y la posibilidad de evitar óptimos locales. Se estudiaron, además, diversas formas de desarrollar estos mecanismos utilizando distintos métodos a la vez que se identificó la implicancia de estos cambios en el resultado final del proceso evolutivo.

Se analizó la introducción de variantes al proceso evolutivo canónico las que permiten desarrollar un abanico más amplio de problemas, ya sea resolviendo problemas al optimizar variables inversas o que posean múltiples restricciones. También se revisó como estas variantes aportan en la disminución de tiempo de cómputo o en la homogeneidad de las soluciones entregadas por el proceso evolutivo.

Finalmente se describieron algunas aplicaciones prácticas de algoritmos evolutivos en el área del diseño estructural, en cuanto a el problema a solucionar como en los distintos métodos utilizados para alcanzar los objetivos buscados, para culminar con un pequeño cuadro resumen con otras aplicaciones prácticas clasificadas según el tipo de método utilizado y el tipo de problema a resolver, demostrando así la versatilidad de los algoritmos evolutivos y su buena sinergia con otros métodos de inteligencia artificial como la lógica difusa o el machine learning.

4.2.- Caso estudio

La realización de los casos de estudio permite experimentar con los diversos métodos y características estudiadas durante la elaboración del estado del arte para evaluar de forma práctica su impacto en los procesos evolutivos generados para resolver el problema.

Al observar la variación de la puntuación de los individuos a lo largo del proceso evolutivo es posible observar como ocurre la optimización de la población de candidatos a solución en el tiempo, e identificar puntos en los que esta optimización se ve ralentizada o detenida.

La realización de múltiples ejecuciones del algoritmo, y el estudio de la distribución de los candidatos a solución en la función evaluación permitieron identificar los cambios necesarios a aplicar sobre el algoritmo canónico que permitiera resolver de mejor manera el problema planteado.

Se observa que la estructura inicial entregada por el algoritmo evolutivo base no presenta niveles de simetría respecto a ninguno de sus ejes pese a que el problema original posee doble simetría. Tomando esto en consideración, y notando que la estructura entregada no ocupa toda el área de muro disponible, y que no posee niveles de excentricidad importantes entre el centro de masa y el centro de rigidez, se observa que el proceso hace énfasis en optimizar respecto a peso y excentricidad más que en los niveles de desplazamiento que la estructura pueda presentar.

En cuanto a la estructura del problema de estudio se observa que hay mayores niveles de simetría que en el caso original que en el problema base, debido probablemente a que solo puede presentar simetría en uno de sus ejes y esta es la única manera de reducir los niveles de excentricidad general. También es posible apreciar una disminución del tamaño de muros al disminuir los niveles de simetría, lo que implica que el algoritmo no sólo prioriza la optimización por área de muros y excentricidad, sino que parece ignorar la influencia de la respuesta de la estructura frente a la solicitación sísmica.

Respecto a esto último, la razón por la cuál no se hace hincapié en la respuesta es porque los niveles de rigidez de ambas estructuras son muy elevados. En el caso base, se tiene una planta de poco más de 90 m2 en la que los muros perimetrales pueden llegar a tener 30 metros de longitud; Mientras que en el caso estudio, se considera una planta estructural de casi 300 m2. Ante esto, los niveles de desplazamiento de la estructura son mínimos, por lo que al llevar a cabo el proceso de optimización la variación de desplazamiento entre los individuos disminuye en el orden de micrómetro, por lo que en esencia la gran mayoría de las estructuras exploradas por el algoritmo no tiene problemas de desplazamientos generando una gamma muy grande de soluciones distintas entre si en las que todas cumplen los requisitos de desplazamiento, dificultando así el proceso de optimización.

4.3.- Trabajos futuros

Debido al aumento explosivo de la capacidad de procesamiento en los últimos años, como la aparición de servicios de nubes de cómputo, hace atractivo el uso de mecanismo de exploración para el diseño estructural, lo que se ve reflejado en el aumento en el uso de diseño generativo como herramienta de trabajo. Los distintos métodos que permitan relacionar la optimización evolutiva con plataformas como BIM, junto con aplicaciones de diseño generativo en el campo de la construcción y el diseño estructural son líneas de estudio a seguir en torno a la computación evolutiva como ayuda al diseño estructural.

En cuanto a los casos prácticos de trabajo, el uso de estructuras más complejas, como la estructura utilizada pero evaluada como un edificio en altura o cualquier estructura en que existan individuos rechazados por la función evaluación, permitiría obtener resultados más decisivos a la hora de discernir entre un candidato a solución y otro más eficiente, por lo que esta también deberá ser una línea de trabajo para futuras aplicaciones.

5.- Bibliografía

Abdallah, H., Farah, E., Haddad, A., Salami, G., Sanboskani, H., Dabaghi, M., & Hamzeh, F. (2019). *Employing generative design for suistanable construction.* Creative Construction Conference, 2019.

Abdelrehim, M., Eid, M., & Sayed, M. (2019). *Structural optimization of concrete arch bridges using Genetic Algortihms.* Ain Shams Engineering Journal.

Akkurt, S., Gokmen, T., & Can, S. (2004). *Fuzzy logic model for the prediction of cement compressive strenght* . Cement and Concrete Research 34 (2004) 1429–1433.

Arciszewski, T., Michalski, R., & Wnek, J. (1995). *Constructive introduction: the key to design creativity.* Third International Round-Table Conference on Computational Models of Creative Design.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice* . New York: Oxford University Press.

Bagley, J. D. (1967). *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms.* PhD Thesis, University of Michigan.

Berrais, A. (1999). *Artificial Neural Networks in Structural Engineering: Concept and applications* .

Box, G. E. (1957). *Evolutionary Operation: A Method for Increasing Industrial Productivity.*

Bremermann, H.-J. (1962). *Optimization through evolution and recombination.*

Cavicchio, D. J. (1970). *Adaptive Search Using Simulated Evolution.* PhD Thesis, University of Michigan.

Chen, B.-T., Chang, T.-P., Shih, J.-Y., & Wang, J.-J. (2009). *Estimation of exposed temperature for fire-damaged.* Comput Mater Sci 2009;44:913–20.

Coello, C. (2005). *An Introduction to Evolutionary Algorithms and Their Applications.* Advanced Distributed Systems Lecture Notes in Computer Science Volume 3563, 2005, pp 425-442.

Coello, C., Lamont, G., & Veldhuizen, D. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems* . Springer.

De Jong, K., Fogel, D., & Schwefel, H.-P. (1997). *A history of evolutionary computation.*

Farhang-Mehr, A., & Azarm, S. (2002). *Entropy-based multi-objetive genetic algorithm for design optimization.*

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* John Wiley & Sons.

Friedberg, R. M. (1958). *A Learning Machine: Part I.*

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company.

Hofmeyer, H., & Dávila, J. (2015). *Coevolutionary and genetic algorithm based building spatial and structural design.* Artificial Intelligence for Engineering Design, Anaysis and Manufacturing (2015), 29, 351-370.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.*

Hollstien, R. B. (1971). *Artificial Genetic Adaptation in Computer Control System.* PhD Thesis, University of Michigan.

Hornby, G. (2003). *Generative representations for evolutionary design automation.* Ph.D. Dissertation, Department of Computer Science, Brandeis University.

Jangid, S., Puri, R., & Kumar, T. (2019). *Evolutionary Algorithms: A Critical Review and its Future Prospects.* National Conference on Research Trends in Big Data and Intelligent Computing, Indian Academy Degree College Autonomous.

Kicinger, R., Arciszewski, T., & De Jong, K. (2005). *Evolutionary computation and structural design: A survey of the state-of-the-art.*

Kicinger, R., Arciszewski, T., & De Jong, K. (2005). *Parametrized versus Generative Representations in Structural Desing: An Empirical Comparison.*

Luke, S. (1997). *Genetic programming produced competitive soccer softbot teams for robocup97.*

Maher, M. (1994). *Creative Design Using A Genetic Algorithm.* Computing in Civil Engineering, ASCE, pp 2014-2021.

Michalewicz, Z. (1994). *1st IEEE Conf. on Evolutionary Computation.*

Rechenberg, I. (1965). *Cybernetic Solution Path of an Experimental Problem.* Royal Aircraft Establishment Library.

Rechenberg, I. (1971). *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* PhD. Thesis.

Rosenberg, R. (1967). *Simulation of Genetic Populations with Biochemical Properties.* PhD Thesis, University of Michigan.

Salehi, H., & Burgueño, R. (2018). *Emerging artificial intelligence methods in structural engineering.* Engineering Structures 171, 170-189.

Schwefel. (1968). *WExperimentelle Optimierung einer Zweiphasenduse Teil IW.* AEG Research Institute Projec.

Schwefel. (1987). *Collective phenomena in evolutionary systems Problems of Constancy and Change—the Complementarity of Systems Approaches to Complexity.* Papers Presented at the 31st Ann. Meeting Int. Society Gen. Syst. Res. vol 2, ed P Checkland and I Kiss (Budapest: International Society for General System Research) pp 1025–33.

Schwefel. (1991). *Parallel Problem Solving from Nature.* 1st Workshop, PPSN I, Dortmund, FRG, October 1-3, 1990, Proceedings.

Schwefel. (1965). *Kybernetische Evolution als Strategie der experimentellen Forschung in der Stromungstechnik.* Thesis, Technical University of Berlin, Hermann Fottinger Institute for Hydrodynamics.

Schwefel. (1974). *Adaptive Mechanismen in der biologischen Evolution und ihr Einflußauf die Evolutionsgeschwindigkeit.* Technical University of Berlin Working Group of Bionics and

Evolution Techniques at the Institute for Measurement and Control Technology Technical Report Re 215/3.

Wolfram, S. (2002). *A new kind of science.* Wolfram Media, Champaing.

Yeh, I.-C., & Lien, L.-C. (2009). *Knowledge discovery of concrete material using genetic operation trees.* Expert Syst Appl 2009;36:5807–12.

Yue-Jiao, G., Wei-Neng, C., Zhi-Hui, Z., Yun, L., Qingfu, Z., & Jing-Jing, L. (2015). *Distributed Evolutionary Algorithms and their models: A survey of the state-of-the-art.* Aplied Soft Computing 34.

Bibliografía de aplicaciones en ingeniería estructural:

1. Kicinger, R., & Arciszewski, T. (2004). Multiobjective evolutionary design of steel structures in tall buildings. In Proceedings of the AIAA 1st Intelligent Systems Technical Conference, Chicago, Illinois.
2. Kicinger R, Arciszewski T, De Jong KA.. Distributed evolutionary design: island-model based optimization of steel skeleton structures in tall buildings. In: Beucke K, Firmenich B, Donath D, Fruchter R, Roddis K, editors. Proceedings of the 10th International Conference on Computing in Civil and Building Engineering (ICCCBEX), Weimar, Germany; 2004. p. 190.
3. Kicinger R, Arciszewski T, De Jong KA. Morphogenesis and structural design: cellular automata representations of steel structures in tall buildings. In: Proceedings of the Congress on Evolution Computat (CEC2004), Portland, Oregon; 2004. p. 411–8.
4. Gang Li, Haiyan Lu, Xiang Liu. “A hybrid genetic algorith and optimality criteria methods for optimum design of RC tall buildings under multi-load cases”. Struct. Design Tall Spec. Build. 19, 656–678 (2010)
5. Pullmann T, Skolicki Z, Freischlad M, Arciszewski T, De Jong KA, Schnellenbach-Held M. Structural design of reinforced concrete tall buildings: evolutionary computation approach using fuzzy sets. In: Ciftcioglu O, Dado E, editors. Proceedings of the 10th International Workshop of the European Group for Intelligent Computing in Engineering (EG-ICE), Delft, The Netherlands; 2003. p. 53–61.
6. Sarma KC, Adeli H. Bilevel parallel genetic algorithms for optimization of large steel structures. Comput-Aided Civil Infrastruct Eng 2001;16:295–304.
7. Valentina Tomeia, Maura Imbimbob, Elena Melec . “Optimization of structural patterns for tall buildings: The case of diagrid”. Engineering Structures 171 (2018) 280–297
8. Pezeshk S, Camp CV, Chen D. Design of framed structures by genetic optimization. J Struct Eng 2000; 126(3):382–8.
9. Talaslioglu Tugrul. “A New Genetic Algorithm Methodology for Design Optimization of Truss Structures: Bipopulation-Based Genetic Algorithm with Enhanced Interval Search”. Hindawi Publishing Corporation, Modelling and Simulation in Engineering
10. Ai Shengli, Wang Yude. “Application of Improved Genetic Algorithms in Structural Optimization Design”. M. Zhu (Ed.): ICCIC 2011, Part VI, CCIS 236, pp. 480–487, 2011.
11. Johan R, et al. “Building intelligence through generative design”. 24th CAADRIA Conference, At Victoria University of Wellington
12. Zhang Yu, Mueller Caitlin. Shear wall layout optimization for conceptual design of tall buildings. Engineering Structures 140 (2017) 225–240

13. S. Tafraouta , N. Bourahlaa , Y. Bourahlab , A. Mebarkic. “Automatic structural design of RC wall-slab buildings using a genetic algorithm with application in BIM environment”. Automation in Construction 106 (2019) 102901

6.- ANEXO

ANEXO A.- Método Combinación Modal

El método modal corresponde a la solución del problema homogéneo para la a la ecuación de equilibrio dinámico sin disipación. Es decir:

$$[M]\{\ddot{v}(t)\} + [C]\{\dot{v}(t)\} + [K]\{v(t)\} = \{P(t)\}$$

$$[M]\{\ddot{v}(t)\} + [K]\{v(t)\} = 0$$

Donde $[K]$ es la matriz de rigidez y $[M]$ la matriz de masa inercial de la estructura. La solución a esta ecuación es conocida:

$$\{v(t)\} = \{\Phi\}\, y_0\, sen(wt)$$

$$\{\ddot{v}(t)\} = -w^2\{\Phi\}\, y_0\, sen(wt)$$

$$\left[[K] - w^2[M]\right]\{\Phi\} = 0$$

Donde $\{\Phi\}$ (Los valores propios de la ecuación) son los modos de vibrar de la estructura, y w_i (Los vectores propios de la ecuación) son las frecuencias de oscilación de los respectivos modos. Debido a la ortogonalidad de los distintos modos de vibrar, la superposición de estos entrega la respuesta de la estructura. A partir de los modos de vibrar, es posible obtener la matriz de masa modal de la estructura:

$$[M_m] = [\Phi]^T[M][\Phi]$$

Esta matriz corresponde a la cantidad de masa que se ve influenciada por los distintos modos de vibrar.

Por otro lado, se utiliza el vector de influencia {r}, el cual representa el desplazamiento estático de las masas de la estructura al producir un movimiento unitario en el terreno en la dirección de un grado de libertad.

Con estos datos es posible calcular el factor de participación modal L_m, el cual representa la medida en que cada modo de vibrar participa en la respuesta de la estructura. Este se obtiene como:

$$\{L_m\} = [\Phi]^T[M]\{r\}$$

Este valor es utilizado para obtener el vector de masa modal efectiva, el cual es representa la masa que mueve cada modo, por lo que la suma de todas las masas modales corresponde a la masa total de la estructura. Además, la norma sísmica de edificios NCh433 establece que sólo es necesario trabajar con los primeros modos de vibrar tal que la suma de sus respectivas masas modales efectivas sea de al menos el 90% de la masa total de la estructura. La formulación de la masa modal efectiva es:

$$M_{meff_i} = \frac{L_{m_i}^2}{M_{m_i}}$$

Finalmente, una vez obtenida la cantidad de modos de vibrar necesarios a utilizar, se calcula la respuesta real. Cuando se conoce la excitación que sufre la estructura en el tiempo, las respuestas modales también son función del tiempo y el problema finaliza. Sin embargo, cuando se diseña una estructura utilizando espectros de aceleración (Como en este trabajo), lo que se obtiene es cuánto será la aceleración máxima en cada modo de la estructura, lo que podría pasar en tiempos distintos de la excitación.

Para solucionar esto, se combinan el desplazamiento de cada grado de libertad producido por cada modo de vibrar (De ahí el nombre combinación modal). En este caso, se utiliza el método denominado SRSS, la cual obtiene la respuesta real como la raíz de la suma de los cuadrados de las respuestas modales.

De esta forma, la respuesta real se formula como sigue:

$$y_{máx_i} = \frac{L_{m_i}}{M_{m_i}} \frac{S_{a_i}}{w_i^2}$$

$$\left\{v_{max}^{(i)}\right\} = \{\Phi_i\}\, y_{máx_i}$$

$$v_{máx_i} = \sqrt{\sum_{j=1}^{N} (v_{max_i}^{(j)})^2}$$

Donde N es la cantidad de grados de libertad del problema, el subíndice hace referencia al grado de libertad, el superíndice hace referencia al modo de vibrar, y_{max} es la excitación máxima en cierto grado de libertad, y S_{a_i} el valor de la aceleración de la excitación para el período del modo de vibrar i.

ANEXO B.- Códigos Python

Respuestas Estructural

```
import numpy as np
import pandas as pd
S = pd.read_csv('Espectro.txt', sep="\t", header=None)

## Coordenadas
def Coordenadas(Individuo):
    M = np.full(32,None)
    #Muro interno hacia arriba
    #3m
    M[0] = [-9.5, -4]
    M[1] = [-6.5, -4]
    M[2] = [6.5,  -4]
    M[3] = [9.5,  -4]
    #6m
    M[4] = [-4, -7]
    M[5] = [4,  -7]
    #Muro interno hacia abajo
    #3m
    M[6]   = [-15.5, 4]
    M[7]   = [-12.5, 4]
    M[8]   = [-9.5,  4]
    M[9]   = [9.5,   4]
    M[10]  = [12.5,  4]
    M[11]  = [15.5,  4]
```

```
    #6m
    M[12]  = [-6.5, 7]
    M[13]  = [0,    7]
    M[14]  = [6.5,  7]
    #Muro interno a derecha
    M[15] = [-12.5, 4]
    M[16] = [12.5,  4]
    M[17] = [6.5,  -4]
    M[18] = [-6.5,   1]
    #Muro interno a izquierda
    M[19] = [-12.5, 4]
    M[20] = [12.5,  4]
    M[21] = [-6.5, -4]
    M[22] = [6.5,  1]
    #Muro Fijo
    #Hacia Abajo
    M[23] = [-18.5, -2.5]
    M[24] = [-2,    -2.5]
    M[25] = [2,     -2.5]
    M[26] = [18.5,  -2.5]
    #Hacia Derecha
    M[27] = [-14, -4]
    M[28] = [0,    1]
    M[29] = [0,   -4]
    M[30] = [0,   -7]
    M[31] = [14,  -4]

    for _muro in range(23):
        if _muro < 6:
            M[_muro][1] += Individuo[_muro]/2
        elif _muro < 15:
            M[_muro][1] -= Individuo[_muro]/2
        elif _muro < 19:
            M[_muro][0] += Individuo[_muro]/2
        else:
            M[_muro][0] -= Individuo[_muro]/2
    return(np.array(M))

# Centro de masa respecto al centro geometrico
# Losa de amarre = 0.55 ton/m2
def CentrodeMasa(Muros,h,rho = 2.8):
    e = Muros[-1]
    Mlosa = 0.55*9.7*37
    cmx = 0
    cmy = 0
    Mt = Mlosa + (e*h*rho*sum(Muros[:-1:]))
    for muro in range(len(Muros)-1):
```

```
        Coord = Coordenadas(Muros)
        if Muros[muro] > 0:
            cmx += Muros[muro]*Coord[muro][0]
            cmy += Muros[muro]*Coord[muro][1]
    cmx *= e*h*rho
    cmy *= e*h*rho
    return([cmx/Mt,cmy/Mt])

# Matriz de Rigidez
# Centro de Rigidez respecto al centro geometrico
def Rigidez(Muros,h,E = 3400000, ediafragma = np.inf):
    ##Unidades Ton, m
    e = Muros[-1]
    CM = CentrodeMasa(Muros,e,h)
    Rxx = 0
    Ryy = 0
    Rtx = 0
    Rty = 0
    Rtt = 0 #RigidezTorcional(Ltx,Lty,ediafragma)
    RxxY = 0
    RyyX = 0
    Coord = Coordenadas(Muros)
    for muro in range(len(Muros)-1):
        if Muros[muro] > 0:
            if muro < 15 or (muro > 22 and muro < 27):
                Lx = e
                Ly = Muros[muro]
                [dx,dy] = Coord[muro]
            else:
                Lx = Muros[muro]
                Ly = e
                [dx,dy] = Coord[muro]
            Rxx += (E*Ly*(Lx**3))/(h**3)
            Ryy += (E*Lx*(Ly**3))/(h**3)
            Rtx += (dy)*(E*Ly*(Lx**3))/(h**3)
            Rty += (dx)*(E*Lx*(Ly**3))/(h**3)
            Rtt += (((dy)**2)*(E*Ly*(Lx**3))/(h**3))+(((dx)**2)*(E*Lx*(Ly**3))/(h**3))
    ey = ((Rtx/Rxx)-CM[1])/(0.5*8)
    ex = ((Rty/Ryy)-CM[0])/(0.5*37)
    return(np.array([[Rxx, 0, Rtx],
            [0, Ryy, Rty],
            [Rtx, Rty, Rtt]]),[ex,ey])

def Masas(Muros,e=0.2,h=2.4,rho=2.8):
    ##Unidades ton, m
    e = Muros[-1]
    M = 0.55*37*9.7
```

```
    Mx = sum(Muros[:-1:])*e*h*rho
    My = sum(Muros[:-1:])*e*h*rho
    Mt = M*(37**2+9.7**2)/12 #Ton*m2
    Mx += M
    My += M
    return(np.array([[Mx,0,0],
            [0,My,0],
            [0,0,Mt]]))

# Recibe un espectro en g y un vector de frecuencias, y entrega las aceleraciones respectivas en g
def Espectros(T,S):
    cargas = np.zeros(len(T))
    for _periodo in range(len(T)):
        for _espectro in range(len(S)-1):
            if T[_periodo] >= S[0][_espectro] and T[_periodo] <= S[0][_espectro+1]:
                cargas[_periodo] = ((S[1][_espectro+1]-S[1][_espectro])/(S[0][_espectro+1]-
S[0][_espectro]))*(T[_periodo]-S[0][_espectro])+S[1][_espectro]
                break
    return(cargas)

def Reordenar(W2entrada,phientrada):
    phisalida = np.zeros([len(phientrada),len(phientrada)])
    W2salida = np.zeros(len(W2entrada))
    aux = np.copy(W2entrada)
    modomin = np.where(W2entrada == min(aux))[0]
    cambios = 0
    while cambios < len(W2entrada):
        for modo in range(len(modomin)):
            for fila in range(len(W2entrada)):
                phisalida[fila][cambios] = phientrada[fila][modomin[modo]]
                W2salida[cambios] = W2entrada[modomin[modo]]
            cambios += 1
            aux[modomin[modo]] = np.inf
        modomin = np.where(W2entrada == min(aux))[0]
    return(W2salida,phisalida)

def Respuesta(Muros,h,e,S,rho = 2.4, E= 3400000,ediafragma=np.inf):
    M = Masas(Muros)
    Rig = Rigidez(Muros,h,E,ediafragma)
    K = Rig[0]
    [ex,ey] = abs(np.array(Rig[1]))
    [w2,phi] = np.linalg.eig((np.linalg.inv(M))@(K))
    [w2,phi] = Reordenar(w2,phi)
    T = 2*np.pi/np.sqrt(w2)
    Sa = Espectros(T,S)
    Mm = np.transpose(phi)@M@phi
    phi = phi@(np.diag(np.diag(Mm)**-0.5))
```

```
Mm = np.transpose(phi)@M@phi ## identidad
r = np.ones(len(phi))
Lm = np.transpose(phi)@M@r
Mmef = np.ones(len(Mm))
for _modo in range(len(Mm)):
   Mmef[_modo] = Lm[_modo]**2
#Masa total
Mt = 0
for _masa in M:
   Mt += sum(_masa)
#Mada modal efectiva acumulada
Mmefneed = 0
#Modos de vibrar necesarios
modos = 0
for _modo in range(len(Mmef)):
   Mmefneed += Mmef[_modo]
   modos += 1
   if Mmefneed >= 0.9*Mt:
      break
#Respuesta modal
ymax = np.zeros(modos)
for _modo in range(modos):
   ymax[_modo] = Lm[_modo]*Sa[_modo]/w2[_modo]
#Desplazamiento modal en cada GDL
vmmax = np.full(modos,None)
for _modo in range(modos):
   vmmax[_modo] = np.transpose(phi)[_modo]*ymax[_modo]
#Desplazamiento GDL [m,m,Rad]
vmax = np.zeros(len(M))
for _modo in range(modos):
   for _gdl in range(len(M)):
      vmax[_gdl] += vmmax[_modo][_gdl]**2
vmax = vmax**0.5

#carga modal en cada GDL
fmmax = np.full(modos,None)
for _modo in range(modos):
   fmmax[_modo] = M@(np.transpose(phi)[_modo])*Lm[_modo]*Sa[_modo]
#carga GDL [tonf,tonf,tonf*m]
fmax = np.zeros(len(M))
for _modo in range(modos):
   for _gdl in range(len(M)):
      fmax[_gdl] += fmmax[_modo][_gdl]**2
fmax = fmax**0.5
return([vmax,fmax,[ex,ey]])
```

Funciones del algoritmo

```
import numpy as np
import Respuestas
import pandas as pd
S = pd.read_csv('Espectro.txt', sep="\t", header=None)
## Funcion que genera P individuos aleatorio
def Constructor(P):
    Pob = np.full(P,None)
    for _individuo in range(P):
        Pob[_individuo] = np.zeros(33)
        for muro in range(23):
            if muro in [4,5,12,13,14]:
                Pob[_individuo][muro] = int(50*np.random.random())/10
            else:
                Pob[_individuo][muro] = int(20*np.random.random())/10
            Pob[_individuo][muro] += 1 if Pob[_individuo][muro]>0 else 0

        for muro in [23,24,25,26,27,31]:
            Pob[_individuo][muro] = 3
        Pob[_individuo][28] = 7
        Pob[_individuo][29] = 4
        Pob[_individuo][30] = 8
        Pob[_individuo][32] = [0.15,0.2,0.25,0.3][np.random.randint(4)]
    return(Pob)



## Cubicador entrega un vector con el porcentaje de area de muros de cada individuo en cada direccion
def Cubicador(Poblacion):
    LargosX = np.zeros(len(Poblacion))
    LargosY = np.zeros(len(Poblacion))
    for _individuo in range(len(Poblacion)):
        LargosY[_individuo] = Poblacion[_individuo][-1]*
sum(Poblacion[_individuo][:15:])+sum(Poblacion[_individuo][23:27:])
        LargosX[_individuo] = Poblacion[_individuo][-
1]*sum(Poblacion[_individuo][15:23:])+sum(Poblacion[_individuo][27::])
    LargosX /= 8*37
    LargosY /= 8*37
    return ([LargosX,LargosY])

## Evaluador entrega un vector con los puntajes de cada individuo
def Evaluador(Poblacion,h,S):
    Puntajes = np.zeros(len(Poblacion))
    [Ax,Ay] = Cubicador(Poblacion)
    [A,B,C,D,E,F] = [10,10,5,5,1,1]
    for _individuo in range(len(Poblacion)):
```

```
        e = Poblacion[_individuo][-1]
        R = Respuestas.Respuesta(Poblacion[_individuo],h,e,S)
        driftX = R[0][0]
        driftY = R[0][1]
        [ex,ey] = R[2]
        Pdx = 1-((driftX-(8.22e-05))/(1.064e-04))
        Pdy = 1-((driftY-(1.1e-05))/(1.8e-04))
        Pex = 1-abs(ex)
        Pey = 1-abs(ey)
        Pax = 1-((Ax[_individuo] - 0.0169)/0.0162)
        Pay = 1-((Ay[_individuo] - 0.0081)/0.0405)
        #print([Pdx,Pdy,Pex,Pey,Pax,Pay])
        Puntajes[_individuo] = (A*Pdx)+(B*Pdy)+(C*Pex)+(D*Pey)+(E*Pax)+(F*Pay)
    return(Puntajes)


#Retorna una lista matriz donde el primer elemento es un vector con las calificaciones de los mejores
#individuos, y el segundo es un vector con los mejores individuos, ordenados de mayor a menor
calificacion
def Mejores(N,Pob,Notas):
    aux = np.copy(Notas)
    #Vector que almacena los indices de las mejores calificaciones en Notas
    Score = []
    #Vector de las calificaciones de los mejores individuos
    MejoresNotas = np.zeros(N)
    #Vector de los mejores individuos
    MejoresPop = np.full(N,None)
    Nmejores = len(Score)
    while Nmejores < N:
        C = np.where(Notas == max(aux))[0]
        #Si los mejores individuos seleccionados son mas que los que se pueden agregar
        if len(C) > N-Nmejores:
            C = C[0:N-Nmejores:]
        for indice in C:
            Score.append(indice)
            aux[indice] = -np.inf
        Nmejores = len(Score)
    for _ in range (N):
        MejoresNotas[_] = Notas[Score[_]]
        MejoresPop[_] = Pob[Score[_]]
    return(MejoresNotas,MejoresPop)

## Diferencia: Calcula la diferencia entre 2 individuos:
def Diferencia(individuo1,individuo2):
    k = 0
    for muro in range(23):
        Mt = 6 if muro in [4,5,12,13,14] else 3
```

```
        Mmin = min(individuo1[muro],individuo2[muro])
        Mmax = max(individuo1[muro],individuo2[muro])
        k += (Mmax-Mmin)/Mt
    return(k/23)

## Seleccion: Entrega un vector con los indices (en Mejores) de los N pares de padres que generaran cada hijo
# Cada par de padres genera 2 hijos
def Seleccion(Notas,N,Mej):
    Notas = np.array(Notas)/sum(Notas)
    Padres = np.full(N,None)
    for _individuo in range(len(Notas)):
        Notas[_individuo] += sum(Notas[_individuo+1::])
    _Par = 0
    intentos = 0
    while _Par < N:
        P1 = np.random.random()
        Padre1 = max(np.where(Notas > P1)[0])
        P2 = np.random.random()
        Padre2 = max(np.where(Notas > P2)[0])
        if intentos == 20:
            intentos = 0
            Padres[_Par] = np.array([Padre1,Padre2])
        elif Padre1 == Padre2:
            _Par -= 1
            intentos +=1
        elif Diferencia(Mej[Padre1],Mej[Padre2]) < 0.1:
            _Par -= 1
            intentos +=1
        else:
            Padres[_Par] = np.array([Padre1,Padre2])
            intentos = 0
        _Par += 1
    return(Padres)


##Toma los mejores individuos, y los cruza segun dicten los indices de Padres. Entrega un vector con todos los hijos
def Cruce(Mejores,Padres):
    _Hijos = np.full(2*len(Padres),None)
    for _Par in range(len(Padres)):
        cambios = np.zeros(33)
        Hijo1 = np.zeros(33)
        Hijo2 = np.zeros(33)
        for alelo in range(33):
            cambios[alelo] = np.random.randint(0,2)
        for alelo in range(33):
```

```
            if cambios[alelo] == 0:
                Hijo1[alelo] = Mejores[Padres[_Par][0]][alelo]
                Hijo2[alelo] = Mejores[Padres[_Par][1]][alelo]
            else:
                Hijo1[alelo] = Mejores[Padres[_Par][1]][alelo]
                Hijo2[alelo] = Mejores[Padres[_Par][0]][alelo]
        _Hijos[2*_Par] = Hijo1
        _Hijos[2*_Par+1] = Hijo2

    return(_Hijos)


def Mutador(Pop, mr):
    Pob = np.copy(Pop)
    for individuo in range(len(Pob)):
        mutacion = np.random.random()
        if mutacion < mr:
            alelo = np.random.randint(0,23)
            Pob[individuo][alelo] = int(50*np.random.random())/10 if alelo in [4,5,12,13,14] else
int(20*np.random.random())/10
            Pob[individuo][alelo] += 1 if Pob[individuo][alelo] > 0 else 0

    return(Pop)


def Variacion(Pop):
    k = 0
    comparaciones = 0
    for _individuo in range(len(Pop)-1):
        for _individuo2 in range(_individuo+1,len(Pop)):
            k += Diferencia(Pop[_individuo],Pop[_individuo2])
            comparaciones += 1
    return(k/comparaciones)

## Ploter de individuos
def Plotear(individuo,h,rho=2.8):
    import matplotlib.pyplot as plt
    e = individuo[-1]
    CM = Respuestas.CentrodeMasa(individuo,h,rho)
    CR = Respuestas.Rigidez(individuo,h)[1]
    C = Respuestas.Coordenadas(individuo)
    print(CM)
    print(CR)
    if e == 0.15:
        color = 'b-'
    elif e == 0.2:
        color = 'm-'
```

```
    elif e == 0.25:
        color = 'r-'
    else:
        color = 'y-'
    for muro in range(32):
        #Verticales
        if muro < 15 or muro in [23,24,25,26]:
            plt.plot([C[muro][0],C[muro][0]],[C[muro][1]-
(0.5*individuo[muro]),C[muro][1]+(0.5*individuo[muro])],color)
        #Horizontales
        else:
            plt.plot([C[muro][0]-
(0.5*individuo[muro]),C[muro][0]+(0.5*individuo[muro])],[C[muro][1],C[muro][1]],color)
    plt.plot(CM[0],CM[1],'r*')
    plt.plot(CM[0]+(0.5*37*CR[0]),CM[1]+(0.5*8*CR[1]),'b*')
    plt.ylim([-20,20])

    plt.show()
```

Ensamblaje del algoritmo

```
import Respuestas
import Evolver
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
Tiempo0 = time.perf_counter()
S = pd.read_csv('Espectro.txt', sep="\t", header=None)
#Tamano de poblacion
lenP = 100
#Sobrevivientes
survivors = 1
#Iteraciones
It = 2000
#Tasa de mutacion inicial (Tasa final = 0.1)
mr = 0.6
#Tasa de reemplazo Tr:[0.25:0.025:0.5] (Generaciones por cambio = ln(0.5)/ln(2*Tr)) -> [0,34:inf]
Tr = 0.25
# Dimensiones Estructura en m
Lx = 30
Ly = 8
h = 2.4

## Recorders
```

```
PuntajeHistorico = np.full(It+1,None)
MejorPuntaje = 0
MejorIndividuo = None

## Caso inicial
Pop = Evolver.Constructor(lenP)
lenMej = int(lenP/2)
Puntajes = Evolver.Evaluador(Pop,h,S)
PuntajeHistorico[0] = [max(Puntajes), Evolver.Variacion(Pop)]
MejorPuntaje = max(Puntajes)
MejorIndividuo = np.copy(Pop[np.where(Puntajes == max(Puntajes))[0]])
CambiosHistoricos = MejorIndividuo
print("Tiempo inicio de iteración = " + str(time.perf_counter()-Tiempo0))
Tiempo0 = time.perf_counter()

## Iteraciones
for _iteracion in range(It):
    [MejoresNotas, MejoresPop] = Evolver.Mejores(lenMej,Pop,Puntajes)
    Tr = 0.25 + 0.025*int(10*_iteracion/(It-1))
    Padres = Evolver.Seleccion(MejoresNotas,int(Tr*(len(Pop)-survivors)),MejoresPop)
    Hijos = Evolver.Cruce(MejoresPop,Padres)
    NotasHijos = Evolver.Evaluador(Hijos,h,S)
    Hijos = Evolver.Mutador(Hijos,(mr+(_iteracion)*(0.1-mr)/(It-1)))
    NotasHijos = Evolver.Evaluador(Hijos,h,S)[0]
    Pop = np.full(len(Hijos)+survivors,None)
    Puntajes = np.zeros(len(Hijos)+survivors)
    aux = Evolver.Mejores(survivors,MejoresPop,MejoresNotas)
    Pop[:survivors:] = aux[1]
    Pop[survivors::] = Hijos
    Puntajes[:survivors:] = aux[0]
    Puntajes[survivors::] = NotasHijos
    if len(Pop) <= 0.5*lenP:
        Nuevos = Evolver.Constructor((lenP-len(Pop)))
        Pop = np.full(lenP,None)
        Pop[:survivors:] = aux[1]
        Pop[survivors:len(Hijos)+survivors:] = Hijos
        Pop[len(Hijos)+survivors::] = Nuevos
        NotasNuevos = Evolver.Evaluador(Nuevos,h,S)
        Puntajes = np.zeros(lenP)
        Puntajes[:survivors:] = aux[0]
        Puntajes[survivors:len(Hijos)+survivors:] = NotasHijos
        Puntajes[len(Hijos)+survivors::] = NotasNuevos
    PuntajeHistorico[_iteracion+1] = [max(Puntajes), Evolver.Variacion(Pop)]
    if max(Puntajes) > MejorPuntaje:
        MejorPuntaje = max(Puntajes)
        MejorIndividuo = np.copy(Pop[min(np.where(Puntajes == max(Puntajes))[0])])
        CambiosHistoricos = np.insert(CambiosHistoricos,len(CambiosHistoricos),None)
```

```
        CambiosHistoricos[-1] = MejorIndividuo
    lenMej = int(len(Pop)/2)
    Titeracion = time.perf_counter()-Tiempo0
    if _iteracion == It -1:
        print("Tiempo de ejecución iteración = "+str(Titeracion/(1+_iteracion)))


##Plot Evolución
Puntos = np.zeros(len(PuntajeHistorico))
Varianza = np.zeros(len(PuntajeHistorico))
for i in range(len(PuntajeHistorico)):
    [Puntos[i],Varianza[i]] = PuntajeHistorico[i]
fig1, ax1 = plt.subplots()
color = 'tab:blue'
ax1.set_xlabel('Generacion')
ax1.set_ylabel('Puntaje', color=color)
ax1.plot(Puntos, color=color)
ax1.tick_params(axis='y', labelcolor=color)
fig1.tight_layout()
plt.show()

## Plot Mejor Individuo
if len(MejorIndividuo) == 1:
    MejorIndividuo = MejorIndividuo[0]
Evolver.Plotear(MejorIndividuo, 0.2, 2.4)
```