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

DISEÑO AUTOMATIZADO DE METAMATERIALES MECÁNICOS BLANDOS MEDIANTE EVOLUCIÓN ARTIFICIAL EN UN AMBIENTE SIMULADO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

MARLO ANDRÉS ALEGRÍA MARTÍNEZ

PROFESOR GUÍA:
JUAN CRISTÓBAL SEBASTIÁN ZAGAL MONTEALEGRE

MIEMBROS DE LA COMISIÓN:
BRUNO GROSSI CÓRDOVA
JOAKIN CRISTÓBAL UGALDE CASTRO

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: MARLO ANDRÉS ALEGRÍA MARTÍNEZ
FECHA: 2021
PROF. GUÍA: JUAN CRISTÓBAL SEBASTIÁN ZAGAL MONTEALEGRE

DISEÑO AUTOMATIZADO DE METAMATERIALES MECÁNICOS BLANDOS MEDIANTE EVOLUCIÓN ARTIFICIAL EN UN AMBIENTE SIMULADO

Los **metamateriales mecánicos** se caracterizan por tener propiedades mecánicas mejoradas o fuera de lo común, tales como: Coeficiente de expansión térmica negativo, **auxeticidad** (i.e. módulo de Poisson negativo), multiestabilidad, entre otros. Estas propiedades vienen dadas en mayor medida por la forma de su estructura interna que por la composición del material del que están hechos.

En la última década, estos tipos de metamateriales han sido cada vez más utilizados para proponer nuevas posibilidades y paradigmas de diseño, especialmente dentro del campo de la **robótica blanda**, lo cual ha servido para simplificar el diseño de robots tradicionales y a su vez ahorrar componentes mecánicos y electrónicos.

En el presente trabajo de título se diseña una herramienta computacional, basada principalmente en librerías de **algoritmos genéticos** y simulaciones de objetos de elementos finitos blandos, por medio de la cual se podrán generar, de manera automatizada, distintos diseños de metamateriales auxéticos.

El procedimiento general para lograr esto consiste básicamente en programar una representación genética que describa la estructura interna de un metamaterial, crear un ambiente de simulación que permita medir la respuesta auxética de cada candidato a metamaterial y, por último, programar una función *fitness* adecuada.

Además, para explotar el potencial de esta herramienta, se programan una serie de funciones adicionales que permiten generar diseños en base a distintos tipos de simetrías, emplear ***seeding*** a partir de metamateriales auxéticos ya conocidos y diseñar metamateriales cuya estructura interna varíe tridimensionalmente.

Por último, para verificar la fiabilidad de esta herramienta, se hace uso de una impresora 3D para imprimir algunos de los diseños generados y estudiar su comportamiento real.

Como resultado de aplicar este sistema, se obtienen diseños novedosos y peculiares de metamateriales auxéticos. Además, mediante el uso de *seeding*, se logran diseños con un grado de auxeticidad mayor al obtenido mediante evoluciones aleatorias. Por último se observa cualitativamente que, para la mayoría de los casos, los modelos simulados e impresos se comportan de manera similar.

Sin embargo, cabe destacar que el sistema implementado posee considerables limitantes de diseño relacionadas al tamaño, forma y complejidad de las estructuras modeladas. Además, las distintas funciones que componen este sistema pueden seguir siendo mejoradas para obtener diseños más auxéticos, estables y complejos.

Ofrezco una dedicatoria a toda mi familia, que me educaron e inculcaron excelentes valores para llegar a ser la persona que soy en la actualidad, además de haberme dado todos los recursos y facilidades para estudiar y obtener un título profesional. Muchos de mis logros en la vida se los debo a ustedes, y se los recompensaré con creces el día de mañana.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

Los materiales blandos son usados cada vez más en robótica, con el fin de facilitar un gran número de aplicaciones (principalmente médicas e ingenieriles), así como también para simplificar el diseño de robots e incluso mejorar sus capacidades. Una manera de conseguir esto es mediante el uso de **materiales inteligentes**, que puedan llevar a cabo el control y actuación del robot. Dentro de esta clasificación se encuentran los denominados **metamateriales mecánicos blandos**, los cuales serán el principal objeto de estudio dentro del presente documento.

Los metamateriales mecánicos son estructuras, generalmente arreglos periódicos, que macroscópicamente exhiben propiedades distintas u opuestas a las de aquellos materiales comúnmente usados, como los aceros y polímeros. Dependiendo de su geometría interna, los metamateriales pueden mostrar propiedades tales como: Coeficiente de Poisson negativo (**Auxeticidad**), coeficiente de expansión térmico negativo, configuraciones multiestables, entre otros.

Con tal de mejorar la eficiencia de los mecanismos basados en materiales blandos, hasta el punto de hacerlos factibles y competitivos respecto a los mecanismos tradicionales, es necesario diseñar metamateriales mecánicos blandos que sean capaces de optimizar las propiedades anteriormente mencionadas.

Teniendo lo anterior en mente es que, en el presente trabajo de título, se propone desarrollar una herramienta computacional, basada en **algoritmos genéticos**, que sea capaz de diseñar la estructura interna de un metamaterial mecánico blando, buscando principalmente maximizar su auxeticidad. Adicionalmente, se emplean metamateriales auxéticos ya conocidos para usarlos como base de la evolución genética, y así obtener estructuras mejoradas. Posteriormente, se buscará la forma de explotar las capacidades de esta herramienta programando funciones capaces de generar metamateriales mas diversos y complejos. Como etapa final, se hará uso de una impresora 3D para fabricar los modelos diseñados por el algoritmo y así evaluar cualitativamente su comportamiento auxético real con respecto a las simulaciones. Por útimo, se realizará un análisis y reflexión sobre la confiabilidad y el desempeño de la herramienta desarrollada, así como también de sus limitaciones y aspectos a mejorar.

- **Objetivo General**:

  Implementar un sistema que permita la evolución artificial de metamateriales auxéticos, mediante algoritmos genéticos, en conjunto con un simulador de la física de componentes blandos.

- **Objetivos Específicos**:

  1. Programar distintas representaciones genéticas que permitan generar geometrías de celdas unitarias o de macroestructuras a partir de un genoma.
  2. Elaborar un sistema de evaluación que permita caracterizar el grado de auxeticidad de cada candidato de metamaterial en un ambiente simulado.
  3. Realizar la evolución de distintos metamateriales empleando el sistema implementado en los puntos anteriores
  4. Explorar el potencial del sistema para generar nuevos diseños y mejorar el comportamiento auxético de metamateriales conocidos.
  5. Fabricar los diseños finales mediante el uso de una impresora 3D y estudiar el comportamiento auxético real.

- **Alcances**:

  - El sistema de evolución artificial de metamateriales se desarrollará por lo menos hasta un punto mínimo funcional, sirviéndose de varios *softwares* y librerías para su uso. Es decir, no se desarrollará un *software* nuevo o interfaz gráfica que englobe todos los procesos de manera que su uso sea intuitivo y didáctico para usuarios externos, como si se tratase de un producto comercial. Sin embargo, se tendrá en consideración que el código a programar sea ordenado, legible, explicativo y que sus *inputs* y *outputs* sean fácilmente identificables.
  - Los metamateriales mecánicos a diseñar consistirán básicamente en una matriz de arreglos periódicos y simétricos, mientras que la principal propiedad a optimizar será la auxeticidad. Sin embargo, se espera que el sistema tenga el potencial para que, en un futuro, pueda trabajar con otros tipos de estructuras de metamateriales y optimizar otro tipo de propiedades.
  - El trabajo de título se enfocará únicamente en estudiar comparativamente las propiedades mecánicas relacionadas a la auxeticidad de las estructuras obtenidas por el sistema a desarrollar, como por ejemplo el Coeficiente de Poisson y la reducción de área transversal; excluyendo el estudio de las aplicaciones concretas que estos diseños puedan tener en la robótica blanda u otras áreas de estudio.
  - Por último, algunos resultados se evaluarán en la realidad haciendo uso de una impresora 3D y empleando filamento flexible de TPU. Además, dado que los laboratorios se encuentran cerrados debido a la pandemia sanitaria vivida durante el período de trabajo, los ensayos de compresión de los metamateriales impresos se realizarán con herramientas e implementos que estén a mano en el hogar.

Capítulo 1

Antecedentes

1.1. Metamateriales Auxéticos

Los **metamateriales** se caracterizan por presentar propiedades electromagnéticas, acústicas, ópticas o mecánicas que vienen dadas en mayor medida por su estructura que por la composición del material del que están hechos [3].

Los **metamateriales mecánicos** en particular, muestran propiedades mecánicas mejoradas, tales como una muy alta dureza, o muy alta relación resistencia-peso. O bien, propiedades inusuales como coeficiente de Poisson negativo, coeficiente de expansión térmica negativo, configuraciones multiestables, entre otras. Lo anterior puede significar una gran oportunidad para encontrar un nuevo tipo de aplicaciones y funcionalidades, tales como cambios de forma programables, propiedades mecánicas ajustables, o absorción de energía [3].

La **robótica blanda** se puede considerar como uno de los principales campos en los cuales estas nuevas aplicaciones y funcionalidades suelen encontrar un lugar, liderando así nuevos cambios de paradigma en cuanto a diseño y manufactura. Por ejemplo, en lugar de unir una serie de actuadores, podría ser posible diseñar bloques de metamateriales mecánicos que, mediante un determinado estímulo de entrada, sean capaces de ejecutar una serie de movimientos complejos, programados directamente en su estructura interna, y de esta manera lograr simplificaciones de diseño y ahorro de hardware [3].

Una de las propiedades más notables que presentan algunos metamateriales mecánicos es la **auxeticidad**. En primera instancia, para comprender mejor este concepto, es preciso tener en mente el caso no auxético que se acostumbra observar, por ejemplo, en un disco de acero cuyo diámetro aumenta tras ser comprimido axialmente en una prensa hidráulica. Esto se puede observar gráficamente en la figura 1.1, donde la línea continua corresponde al estado inicial y la línea punteada al final.

En un metamaterial auxético ocurre lo opuesto, es decir, al comprimirlo axialmente, su sección transversal se verá reducida, tal como se muestra en la secuencia de la figura 1.2.

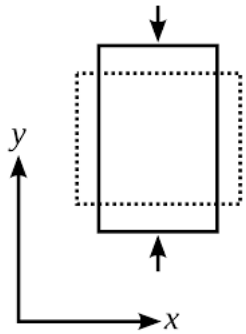


Figura 1.1: Compresión de un material no auxético. Su sección transversal aumenta cuando su altura axial disminuye.

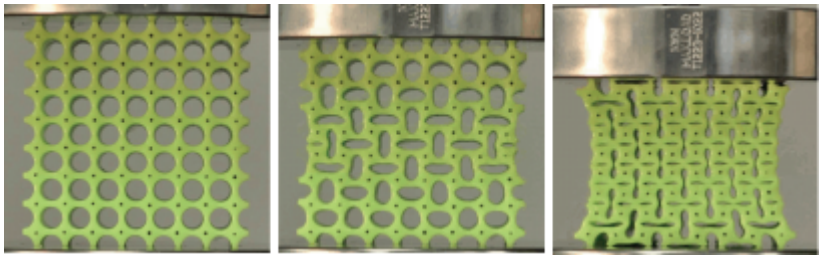


Figura 1.2: Compresión de un metamaterial auxético. Su sección transversal disminuye al igual que su altura axial [7].

Para medir el comportamiento auxético de un metamaterial se hace uso del **Módulo de Poisson** (ν), el cual, tomando como referencia los ejes de la figura 1.1, se define como:

Definición 1.1 ([2]) *Módulo de Poisson*

$$\nu = -\varepsilon_x/\varepsilon_y$$

Donde $\varepsilon_x = x_{final} - x_{inicial}$ es la deformación transversal y $\varepsilon_y = y_{final} - y_{inicial}$ es la deformación axial.

De esta manera, al ejercer una compresión axial ($\varepsilon_y < 0$), para que el material sea auxético ($\nu < 0$), se debe dar que la deformación transversal responda de tal manera que también sea negativa, es decir, comprimiéndose ($\varepsilon_x < 0$), resultando así un módulo de Poisson menor a cero.

Otra manera de obtener un módulo de Poisson negativo es tensar un metamaterial y que su sección transversal responda ensanchándose en lugar de reducirse. A estos metamateriales se les conoce como **alostéricos**.

1.2. Tipos de Metamateriales

Actualmente se han desarrollado diversos tipos de metamateriales mecánicos, los cuales funcionan en base a distintos principios. A continuación se describen 5 de los tipos más conocidos: Metamateriales basados en vigas o varas (*beam-based*), basados en *origami*, basados en *kirigami*, sistemas reforzados y metamateriales multiestables (*snap-based*) [3] [7].

Los **metamateriales basados en vigas** son los más comunes y los más usados a la hora de crear metamateriales auxéticos. Por lo general, estos consisten en arreglos de patrones simétricos que se repiten periódicamente dentro de una malla. En la figura 1.3 A, B y C se pueden observar ejemplos de metamateriales basados en varas y aplicaciones en la robótica blanda.

Los **metamateriales basados en *origami*** consisten en delgadas láminas bidimensionales con dobleces predefinidos, de tal manera que puedan doblarse progresivamente de formas preprogramadas. De esta manera, se pueden obtener propiedades auxéticas o configuraciones multiestables.

Los **metamateriales basados en *kirigami*** son similares al origami, pero se añaden cortes en el arreglo. De esta manera, se pueden lograr deformaciones fuera del plano de la lámina y crear complejas superficies tridimensionales. En la figura 1.3 D, E y F se pueden apreciar ejemplos de metamateriales basados en *origami* y *kirigami*, junto a algunas de sus aplicaciones en robótica blanda.

Los **metamateriales basados en sistemas reforzados** consisten en alineamientos anisotrópicos en direcciones específicas, con tal de favorecer la resistencia a ciertos tipos de pandeo y torsión. Por lo general, este tipo de metamateriales están basados en patrones de tejidos, gracias a su potencial para programar flexibilidad, resistencia y cambios de forma. En la figura 1.3 G, H e I se pueden apreciar algunos ejemplos y aplicaciones.

Por último, se destacan los **metamateriales multiestables**, capaces de "chasquear" (*snap*) entre distintas configuraciones estables según el estímulo que reciban. En la figura 1.4 se observa un metamaterial reconfigurable, que contiene uniones complejas capaces de proveer múltiples grados de libertad y multiestabilidad.

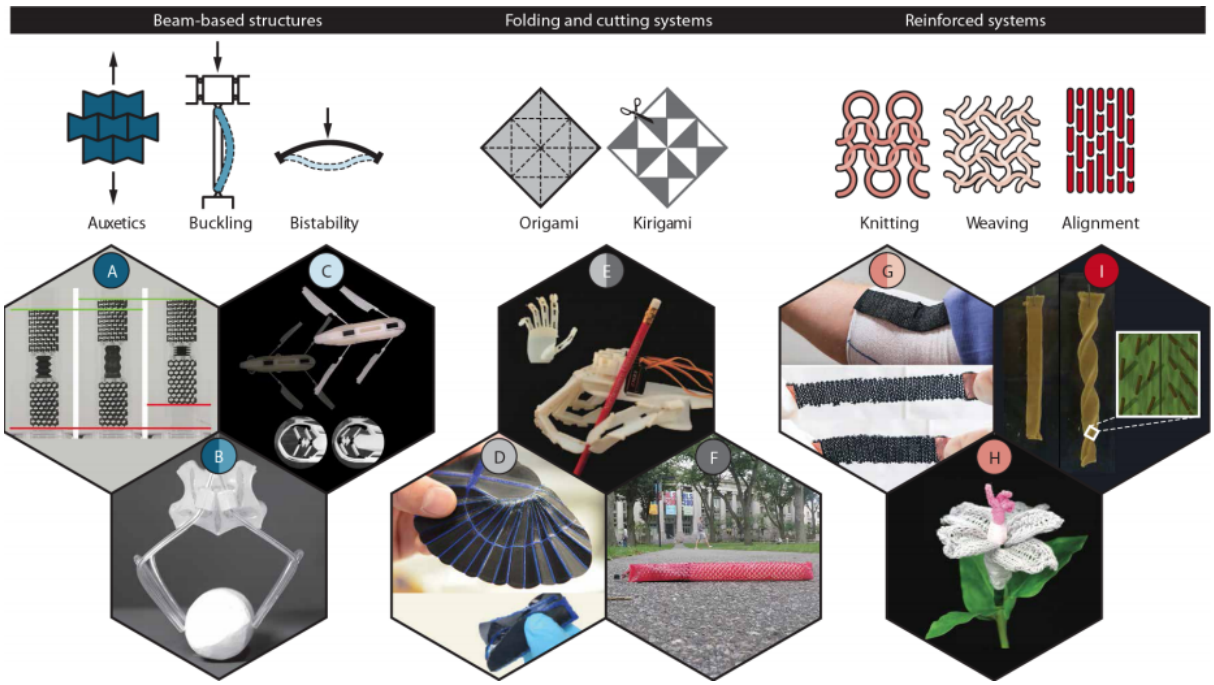


Figura 1.3: Distintos tipos de metamateriales mecánicos blandos junto a algunas de sus aplicaciones más famosas y exitosas dentro del campo de la robótica blanda. (A) Movimiento de gusano usando bloques auxéticos y no auxéticos. (B) Actuadores auxéticos inducidos por pandeo. (C) Propulsión con elementos biestables sensibles a la temperatura. (D) Ala de origami biestable con capacidad de carga y transformación rápida programada. (E) Una mano robótica plegable inspirada en el origami. (F) Locomoción rectilínea mejorada con cobertura de kirigami. (G) Tejido de músculos artificiales. (H) Tejido con forma de flor que florece mediante actuadores SMA (*Shape-Memory Alloys*). (I) Actuadores quirales compuestos hechos de partículas alineadas magnéticamente en una matriz de resina [3].

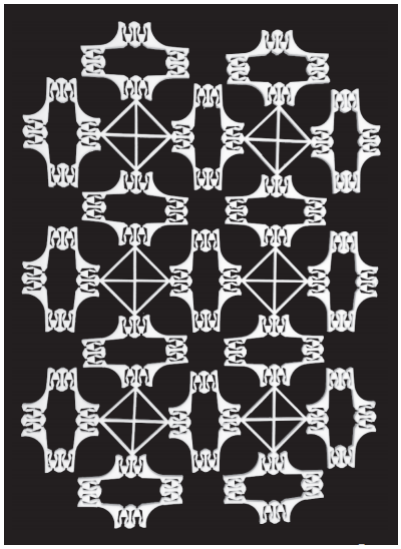


Figura 1.4: Metamaterial reconfigurable que contiene unidades de bisagras complejas que brindan múltiples grados cinemáticos de libertad y multiestabilidad [7].

1.3. Mecanismos Blandos y *Soft Robotics*

Otra de las aplicaciones concretas que poseen los metamateriales mecánicos corresponde al diseño de **mecanismos blandos**. Estos consisten en bloques monolíticos de metamateriales mecánicos, cuya estructura interna se encuentra programada para efectuar combinaciones complejas de torsión, flexión, compresión, entre otros; y así llevar a cabo tareas específicas.

A continuación, se presentan dos ejemplos básicos de este tipo de mecanismos: Una manilla y una pinza, las cuales fueron diseñadas con ayuda de un software de simulación de objetos y fabricados mediante técnicas de impresión 3D [1].

En primer lugar, se describe a la **manilla de puerta metamaterial**, la que se puede observar en la figura 1.5 A. Esta consiste en secciones relativamente densas y rígidas, como la manilla, el marco y el pestillo, y una matriz interna compuesta por distintos patrones que interactúan entre ellos, facilitando la torsión de la manilla y el movimiento lateral del pestillo. Por otro lado, en la figura 1.5 B, se observa una manilla de puerta tradicional, compuesta por un conjunto diverso de elementos mecánicas (eje, palanca, resorte, pestillo, etc), las cuales deben ser manufacturadas por separado y posteriormente ensambladas. Sin embargo, su contraparte metamaterial tiene la ventaja de consistir en una única pieza, cuya fabricación y ensamblaje son básicamente el mismo proceso.

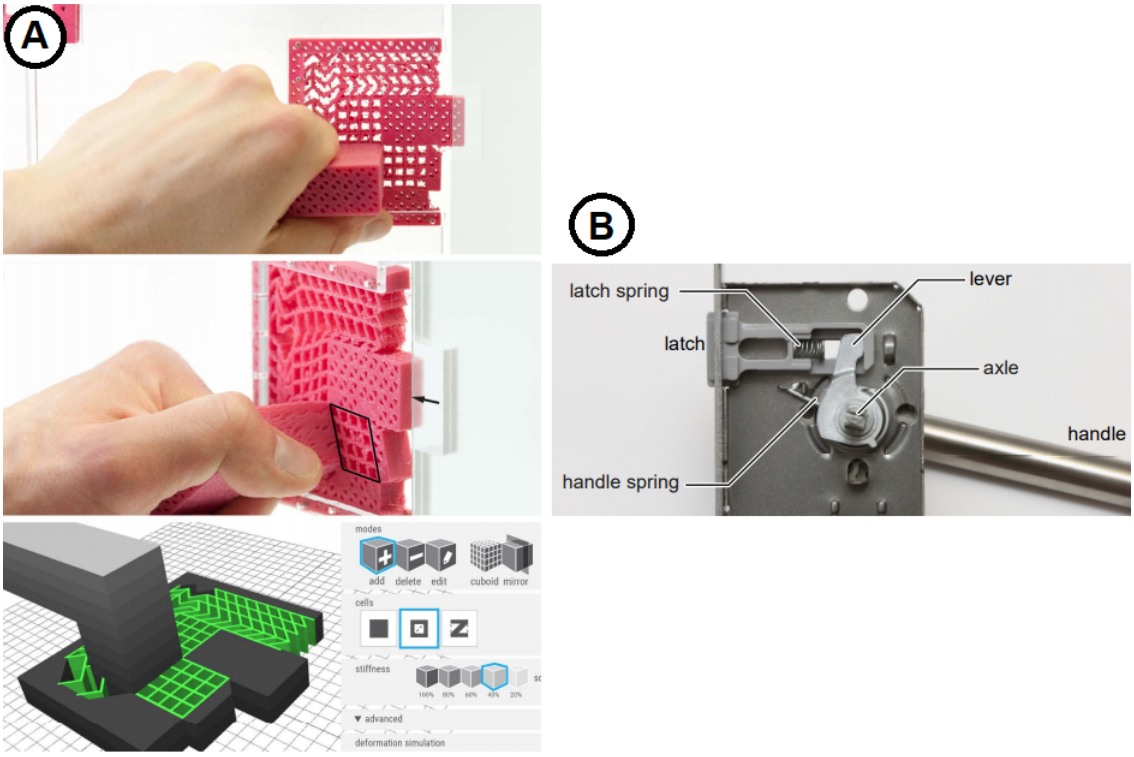


Figura 1.5: (A) Mecanismo blando que se asemeja a una manilla de puerta. (B) Manilla de puerta tradicional y sus componentes [1].

En segundo lugar, se describe la **pinza metamaterial**, la cual puede apreciarse en la figura 1.6. Al comprimir el lado del mango, el cuadrado central adopta la forma de un rombo, transmitiendo simétricamente la deformación de un lado a otro.

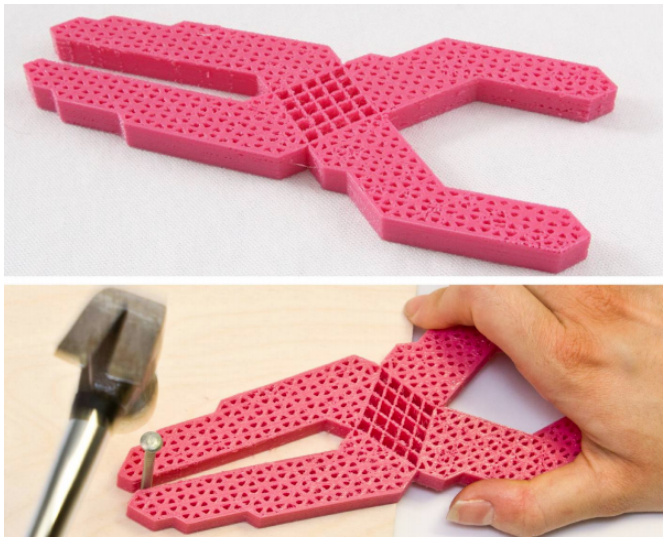


Figura 1.6: Pinza metamaterial. El rombo central transmite la deformación ejercida de un lado a otro [1].

Por último, se entrega un ejemplo de cómo, al emplear metamateriales mecánicos, se puede simplificar el diseño de un robot blando [2]. Dicho robot tiene la funcionalidad de desplazarce por dentro de un ducto empleando un movimineto de traslación similar al de un gusano. (ver figura 1.7). Este diseño cuenta con solamente un actuador y dos bloques de metamateriales, uno auxético y otro no auxético. Idealmente, ambos bloques deben presentar un módulo de Young similar y coeficientes de Poisson de igual magnitud, pero distinto signo.

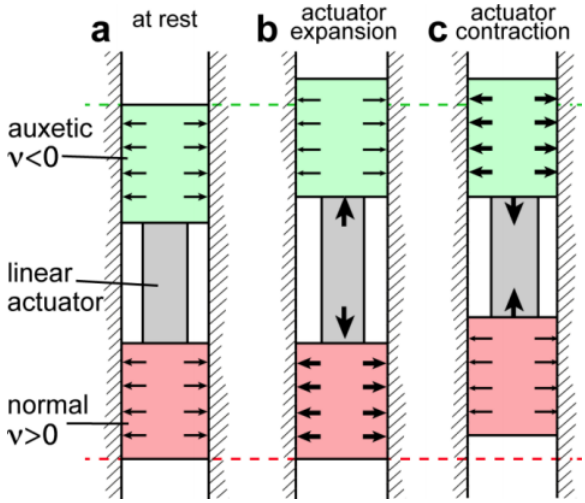


Figura 1.7: Diagrama conceptual de robot blando con movimiento de gusano que puede moverse a través de una cañería [2].

Los bloques de metamateriales que usa este robot se pueden observar en la figura 1.8. A la izquierda (a) se muestra el material auxético, conocido en la literatura como *"Re-entrant Honeycomb"*. Dicho metamaterial ha sido ampliamente utilizado en distintos estudios gracias a su alto grado de auxeticidad. Por otro lado, a la derecha (b) se presenta un metamaterial mecánico no auxético, cuya estructura es de cierta manera compatible y opuesta a la anterior.

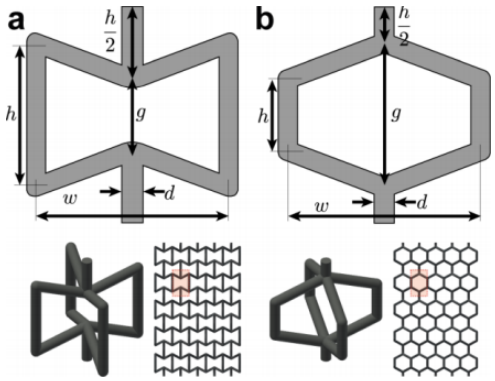


Figura 1.8: Bloques de metamateriales mecánicos blandos usados por el robot gusano explorador de cañerías. a) Metamaterial altamente auxético conocido por el nombre *Re-entrant Honeycomb*. b) Metamaterial no auxético con una estructura similar pero opuesta a la anterior [2].

En la figura 1.9 se puede observar al robot-gusano en acción. Cuando el actuador se expande, comprime de igual manera a ambos bloques, sin embargo, estos exhiben un comportamiento opuesto el uno del otro. En el caso de metamaterial auxético, la compresión axial hace disminuir su sección transversal, lo que resulta en una disminución de la fuerza normal entre el metamaterial y la pared de la cañería. Lo contrario ocurre para el metamaterial no auxético. Por lo tanto, para este caso, la fricción que siente el material no auxético es mayor a la del auxético, haciendo que este último avance mientras el primero se queda fijo. Lo contrario ocurre durante la contracción del actuador, es decir, la fuerza normal del metamaterial material auxético aumenta, mientras que la del material no auxético disminuye, haciendo que el auxético se quede fijo y el no auxético deslice.

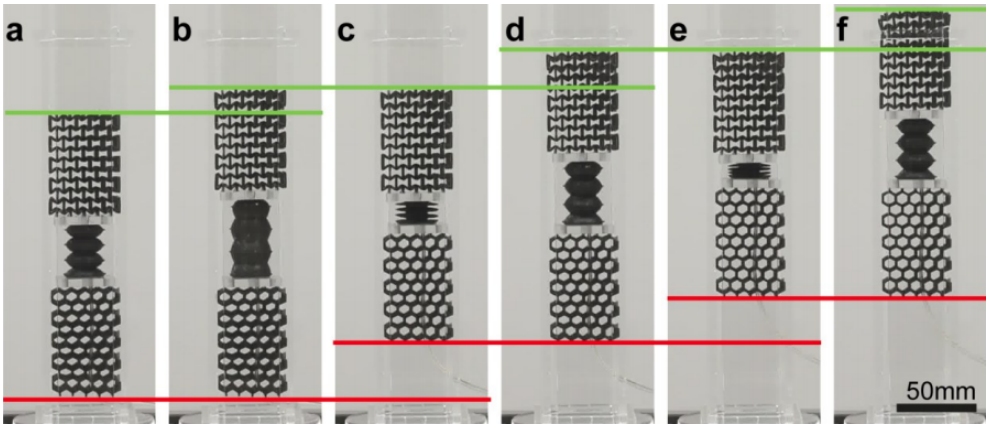


Figura 1.9: Secuencia de movimientos realizada por el robot al desplazarse dentro de una cañería [2].

Normalmente, la elaboración de este tipo de robot requiere de una mayor cantidad de actuadores y otros componentes eléctricos y mecánicos, lo que desemboca en un mayor desafío a la hora de coordinar cada elemento para obtener el movimiento de gusano deseado. Sin embargo, al utilizar metamateriales mecánicos blandos, se logra reducir la cantidad de actuadores a solamente uno, y el problema de la sincronización se resuelve automáticamente gracias al comportamiento opuesto de ambos metamateriales implementados.

De acuerdo a lo anterior, queda en evidencia el alto potencial que poseen los metamateriales al momento de encontrar aplicaciones concretas dentro de la robótica blanda. Es por esto que es importante buscar maneras de mejorar el rendimiento de estos metamateriales y optimizar las propiedades mecánicas que los caracterizan.

1.4. Simulaciones Computarizadas de Metamateriales

Estudios que implementan simulaciones computarizadas de metamateriales auxéticos ya han sido desarrollados con anterioridad [6]. En este caso, se revisa un estudio comparativo de geometrías auxéticas bidimensionales y tridimensionales, confeccionadas en el software *Solid Edge* y posteriormente simuladas usando el método de elementos finitos del software *NX-8.0*. Ambos *softwares* desarrollados por **Siemens PLM Solutions**.

En la figura 1.10 se puede observar la librería de geometrías bidimensionales auxéticas y potencialmente auxéticas desarrolladas en *Solid Edge*. Por su parte, los modelos tridimensionales pueden observarse en la figura 1.11.

En la figura 1.12 se observan cuatro gráficos donde se comparan algunas de las propiedades mecánicas de los metamateriales ya mencionados. En las áreas (A) y (B) de la imagen se relaciona el módulo de Poisson con la máxima reducción de área para los metamateriales bidimensionales y tridimensionales, respectivamente. Por otro lado, en las áreas (C) y (D), se comparan los módulos de Young de los distintos metamateriales bidimensionales y tridimensionales, respectivamente.

A continuación se hace un listado resumiendo los principales resultados y análisis de este estudio.

- Los metamateriales más auxéticos resultaron ser *'re-entrant Masters–Evans v2'* y *'re-entrant cuboid'*, lo cuales obtuvieron módulos de Poisson al rededor de $-1{,}7$ y $-1{,}8$.
- El resto de los metamateriales auxéticos registraron módulos de Poisson entre $-0{,}1$ y $-0{,}9$.
- Los metamateriales más auxéticos registraron una menor reducción de área. esto es, alrededor del 15 %, mientras que el resto de los valores (es decir, para los metamateriales menos auxéticos) estuvieron entre un 25 % y 40 %.
- Respecto al módulo de Young, se puede observar que los metamateriales auxéticos pueden lograr un módulo mayor al del material del que están hechos. Sin embargo, cabe recalcar que dicho parámetro puede variar dependiendo del espesor de pared de la estructura que conforma el metamaterial.

Ahora bien, con respecto al trabajo a desarrollar en el presente documento, es importante mencionar que los resultados del estudio presentado anteriormente servirán para tener una noción y un parámetro de comparación entre estos y los resultados de los metamateriales auxéticos a generar por el sistema a implementar. Por supuesto, teniendo siempre en consideración las diferencias inherentes a la naturaleza de cada estudio.

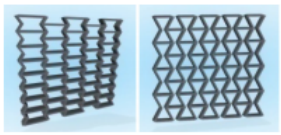
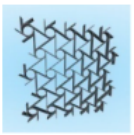
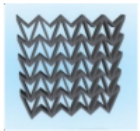
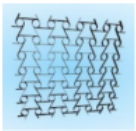
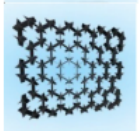
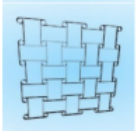
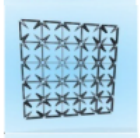
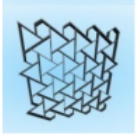
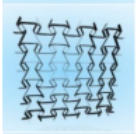
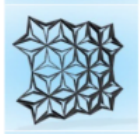
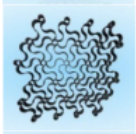
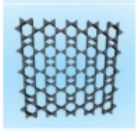
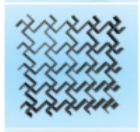
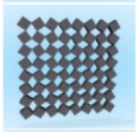
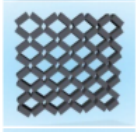
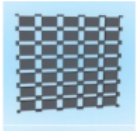
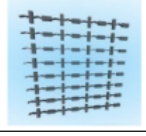
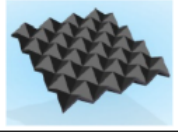
Geometry	CAD model	Geometry	CAD model
Re-entrant Masters–Evans (two models) [9–12, 22]		Chiral circular [16]	
Re-entrant triangular [13]		Chiral circular symmetric [16]	
Re-entrant star 3-n [11]		Chiral square symmetric [16]	
Re-entrant star 4-n [11]		Chiral hexagonal [16]	
Re-entrant star 6-n [11]		Chiral rectangular symmetric [16]	
Re-entrant hexagonal honeycomb [4]		Rotachiral [16]	
Lozenge grid oblong [14]		Rotating unit triangles [17]	
Lozenge grid square [14]		Rotating unit squares [17]	
Square grid [4]		Rotating unit rectangles v1 inspired by [17]	
Re-entrant sinusoidal [4]		Rotating unit rectangles v2 inspired by [17]	
Microporous [4]		Clegg and Vandeperre [17]	
Liquid crystalline [15]		Egg rack mechanism [17, 18]	

Figura 1.10: Librería CAD de metamateriales bidimensionales auxéticos y posiblemente auxéticos, desarrollada en *Solid Edge* [6].

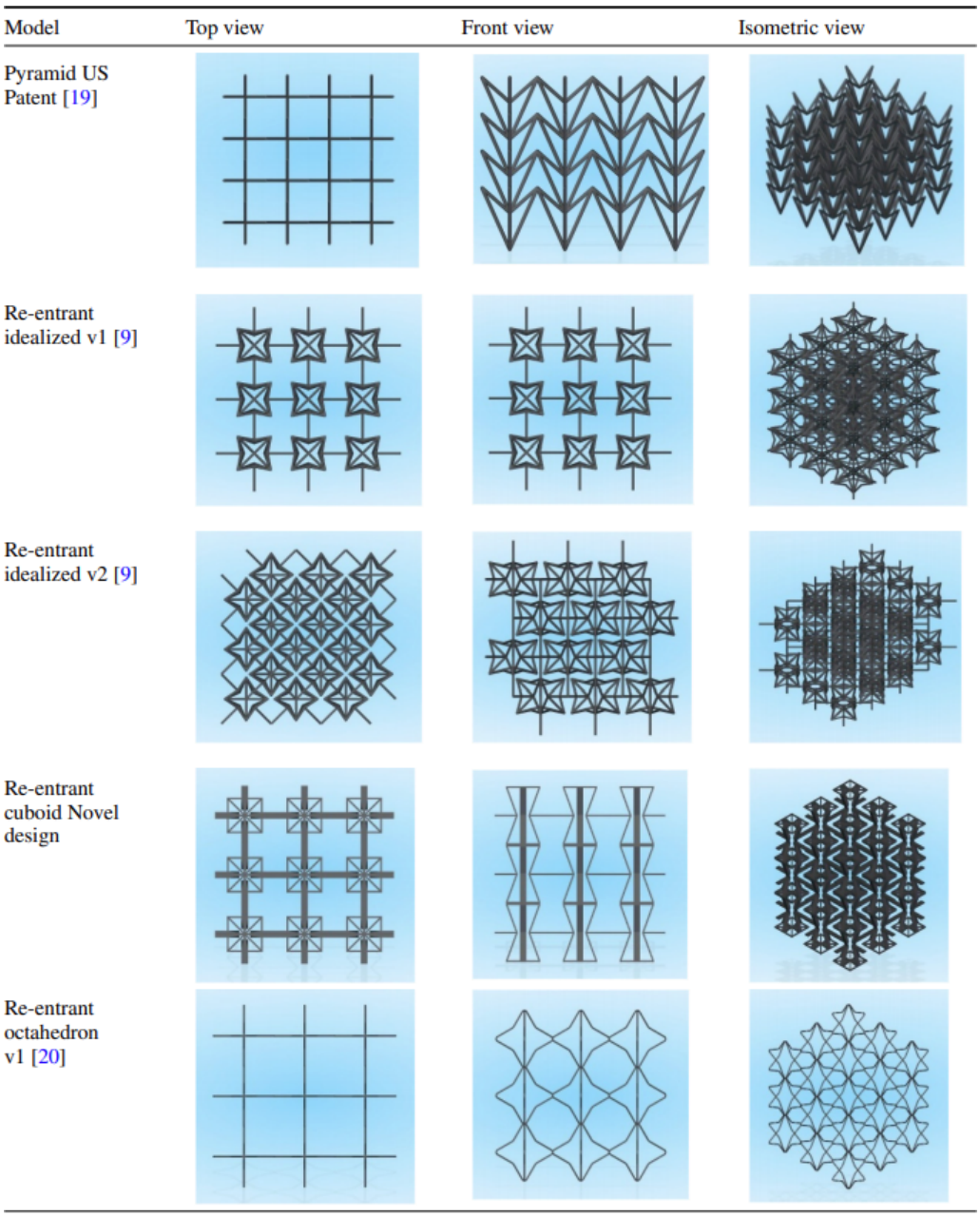


Figura 1.11: Librería CAD de metamateriales tridimensionales auxéticos y posiblemente auxéticos, desarrollada en *Solid Edge* [6].

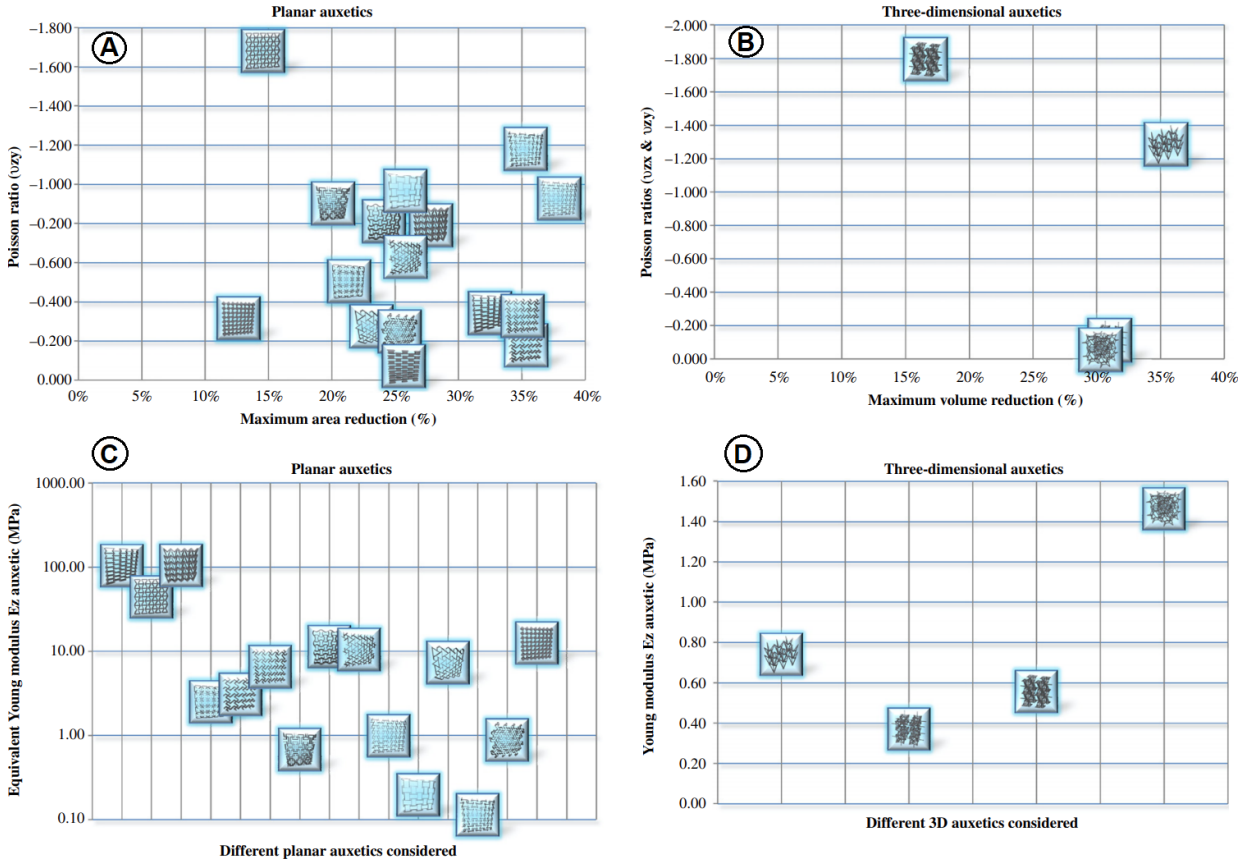


Figura 1.12: Gráfico comparativo de las propiedades de los distintos metamateriales simulados. (A) Módulo de Poisson versus máxima reducción de área para metamateriales bidimensionales. (B) Módulo de Poisson versus máxima reducción de área para metamateriales tridimensionales. (C) Módulo de Young de los metamateriales bidimensionales. (D) Módulo de Young de los metamateriales tridimensionales [6].

Por último, se destaca uno de los estudios más recientes sobre optimización y simulación de metamateriales auxéticos [8], en el cual mediante optimización multi-objetivo de topologías, en base a algoritmos genéticos y métodos de elementos finitos, se modifica la estructura de ciertos metamateriales conocidos para generar nuevas estructuras con una mejor respuesta auxética.

Para ello, se trabaja sobre una cuarta parte del patrón que constituye al metamaterial (ya que estos suelen presentar simetrías axiales en x e y), la cual será representada en una malla de 10x10 elementos finitos. Posteriormente, se limpian los elementos innecesarios de los diseños generados, tal como se muestra en la figura 1.13, para que estos tengan una forma continua y bien contorneada. Finalmente, se hace uso de un *software* de simulaciones computacionales de estática lineal para medir el comportamiento auxético de los candidatos.

En la figura 1.14 se presentan los resultados finales del estudio en cuestión.

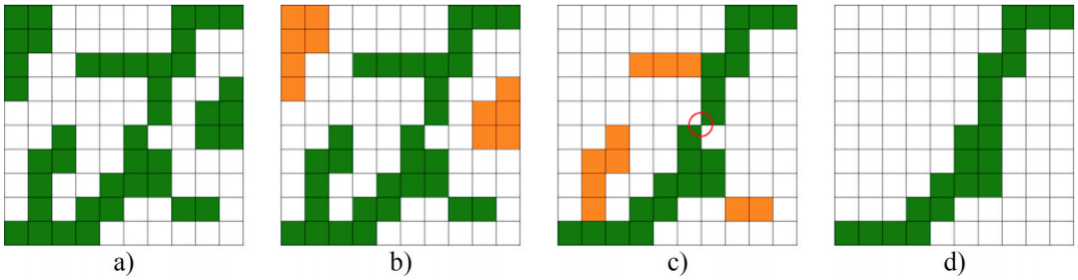


Figura 1.13: Proceso de filtrado. a) Topología base. b) Remoción de regiones inconexas. c) Remoción de partes mecánicamente inactivas e identificación de uniones de bisagra. d) Topología final [8].

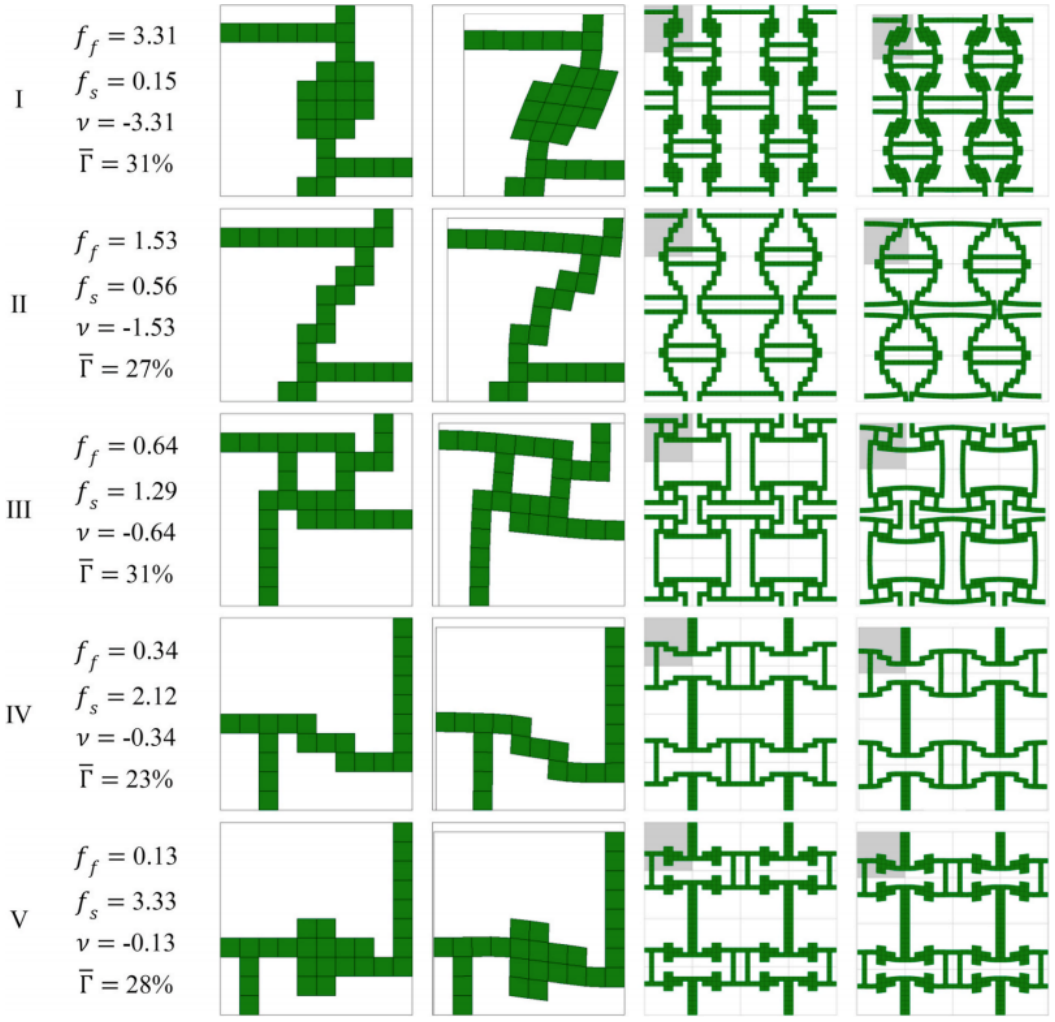


Figura 1.14: Nuevas topologías auxéticas diseñadas por el estudio presentado [8].

A partir del estudio anteriormente presentado, se desprenden ciertas consideraciones que se deben tener en cuenta al momento de diseñar metamateriales auxéticos, tales como la importancia de filtrar aspectos indeseados de los diseños generados y aprovechar las simetrías que se presentan en sus topologías.

1.5. Algoritmos Genéticos

Uno de los factores claves del sistema a desarrollar, por medio del cual se evolucionarán y obtendrán los diseños finales de metamateriales auxéticos, son los denominados **algoritmos genéticos** (GA por sus siglas en inglés) [5]. Estos se definen como algoritmos de búsqueda basados en la mecánica de la *"selección natural"*. Se combinan los conceptos de la *"supervivencia del más apto"*, junto a un intercambio de información aleatorio, pero al mismo tiempo bien estructurado. De esta manera, en cada generación se crea un nuevo set de individuos tomando como base a los más aptos de la generación anterior, y así sucesivamente hasta converger a una solución.

Un algoritmo genético simple consta de tres operaciones fundamentales:

1. **Reproducción**: Operación en que cada individuo es evaluado en la denominada **función objetivo**, de la que se obtiene un parámetro denominado *fitness*, el cual representa qué tan apto es el individuo para resolver el problema en cuestión. Así, aquellos que presenten un mejor *fitness*, tienen una mayor probabilidad de aportar sus genes (generalmente bits de información) en la creación de la siguiente generación.
2. **Crossover**: Una vez identificados los individuos más aptos, estos intercambian bits de información entre ellos de manera aleatoria, creando así nuevos individuos para la siguiente generación.
3. **Mutación**: Para evitar pérdidas de información, o que el algoritmo se encierre dentro de soluciones locales, se añade una pequeña probabilidad de que ocurran mutaciones genéticas aleatorias dentro de un individuo. Por ejemplo, que de manera arbitraria, alguno de sus bits cambie de 1 a 0, o viceversa. De esta manera, se puede explorar si hay mejores soluciones fuera de la línea de evolución que se llevaba siguiendo hasta el momento.

En la figura 1.15 se observa un esquema que muestra gráficamente la interacción que existe entre las operaciones recientemente descritas.

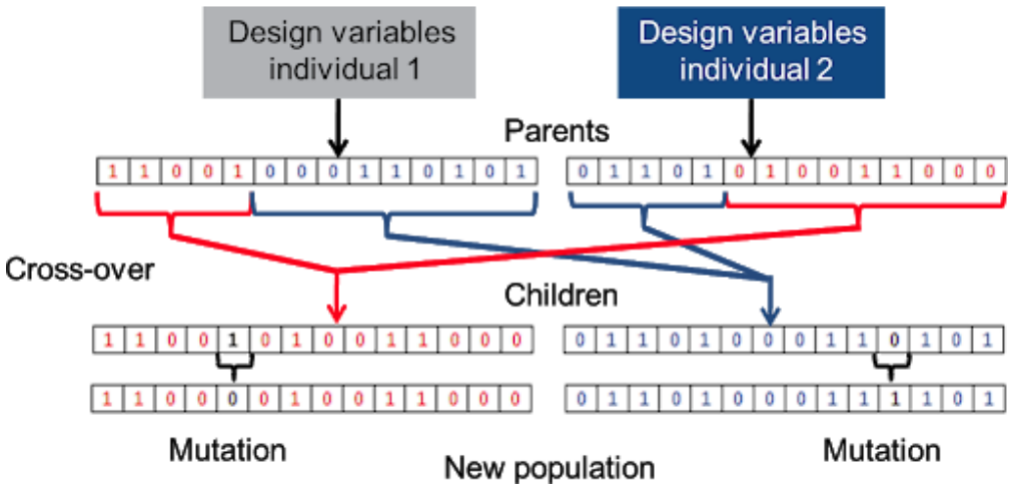


Figura 1.15: Esquema básico que muestra la forma en que opera un algoritmo genético simple [5].

Para ejemplificar el funcionamiento de un GA, se presenta uno de los casos más estudiados y difundidos dentro de este ámbito, el **Problema del vendedor viajero**, o TSP (*Traveler Salesman Problem*) [11]. En dicho problema, un vendedor ambulante desea visitar una determinada cantidad de ciudades exactamente una vez, para posteriormente volver a la ciudad inicial con una cantidad mínima de distancia recorrida. En pocas palabras, un GA simple resolvería dicho problema de la siguiente manera: En primer lugar, se genera un conjunto de rutas aleatorias en base a algún tipo de representación genética, como por ejemplo, una cadena de números reales que indican la distancia relativa entre distintos pares de ciudades, esto teniendo en consideración las restricciones del problema. A continuación, se evalúa cada ruta en la función objetivo, la cual puede definirse, por ejemplo, como la suma de cada elemento de la cadena (Reproducción). Luego, se identifican los individuos que arrojaron un mejor *fitness* para combinar su información genética con la del resto de la población y así generar un nuevo set de posibles soluciones (Crossover). Finalmente, y con una baja probabilidad de ocurrencia, se generan alteraciones puntuales dentro de los individuos, como por ejemplo, cambiar el orden de dos ciudades, esto con la finalidad de explorar nuevas rutas fuera de la línea de convergencia actual (Mutación). El proceso se repite una cierta cantidad de veces hasta obtener una ruta lo suficientemente óptima.

Entre las aplicaciones concretas que tiene este problema se encuentran, por ejemplo: Rutas de vehículos, impresión de circuitos eléctricos, cristalografía, cableado eléctrico, entre otros.

A continuación, se presentan algunos de los distintos tipos de GA comúnmente encontrados en librerías de programación [9]:

- **Simple GA:** En cada generación, el algoritmo crea una población completamente nueva de individuos, a partir del Crossover de los individuos de la generación anterior. Opcionalmente, el mejor individuo de cada generación es trasladado a la siguiente. Este proceso continúa hasta que se cumplen los criterios de terminación.
- **Steady-State GA:** Utiliza poblaciones superpuestas, con un factor de superposición especificado por el usuario. Esto quiere decir que, en cada generación, el algoritmo crea una población temporal de individuos (cuyo tamaño depende del factor de traslape), los agrega a la población anterior, y posteriormente elimina a los peores individuos para devolver la población a su tamaño original.
- **Incremental GA:** Utiliza poblaciones superpuestas, pero con muy poca superposición, reemplazando solamente uno o dos individuos en cada generación, los cuales pueden ser sus padres, un individuo aleatorio, o uno que sea similar.
- **Deme GA:** Este algoritmo genético tiene múltiples poblaciones independientes. Cada población evoluciona utilizando un *Steady-State* GA, pero en cada generación algunos individuos migran de una población a otra. La población principal se actualiza en cada generación con el mejor individuo de cada población.

1.6. *Seeding*

Seeding [11] es una técnica utilizada en algoritmos genéticos que permite manipular la primera generación de individuos de tal manera que, desde un principio, estos presenten un valor de *fitness* suficientemente aceptable. Como consecuencia de esto, el algoritmo converge rápidamente a una solución óptima, sin embargo, el riesgo de que dicha solución sea un óptimo local y no global es mayor.

Existen distintas maneras de hacer *seeding* dependiendo del problema en cuestión. Para ilustrar esto, nuevamente se hará uso del TSP, puesto que se han estudiado varias técnicas de *seeding* para este problema.

- **Random Initialization Technique**: Las ciudades sucesivas de las soluciones iniciales son escogidas de manera aleatoria. Esto sería equivalente a no usar *seeding*.
- **Nearest Neighbor Technique (NN)**: Es una de las técnicas más usadas. Los individuos son construidos tal que la ciudad sucesiva sea la más cercana a la ciudad actual. Los mejores individuos tendrán la oportunidad de mejorar sus rutas en las siguientes generaciones.
- **Gene Bank Technique (GB)**: Con esta técnica se puede generar una solución inicial con calidad y diversidad. El principio bajo el cual trabaja esta técnica es el siguiente: Las N ciudades se permutan y agrupan con tal de construir un banco de genes. El banco de genes es una matriz A cuyas dimensiones son CxN, donde C corresponde a las C ciudades que están más cerca de la ciudad i. El elemento A[i][j] es la j-ésima ciudad más cercana a la ciudad i. Así, el algoritmo puede ir sucesivamente eligiendo entre las j-ésimas ciudades más cercanas a sus respectivas ciudades i.
- **Selective Initialization Technique (SI)**: En esta técnica, una lista de las k-ésimas ciudades más cercanas para cada ciudad es generada con antelación. El valor de k es asignado en base al tamaño del problema. El algoritmo le da mayor prioridad a las rutas con ciudades que pertenecen a las k ciudades más cercanas con respecto a la ciudad actual.
- **Sorted population Technique (SP)**: Esta técnica genera una gran población inicial de soluciones. Luego, estas se ordenan de manera ascendente según su *fitness* y se escoge un cierto porcentaje de soluciones cuyo *fitness* sea mayor al promedio, para finalmente arrancar la evolución a partir de ellas.

En la figura 1.16 se puede observar el mejor rendimiento obtenido con cada técnica de *seeding* para distintas instancias del TSP. Según estos resultados, la técnica NN puede generar individuos con una mayor tasa de convergencia que el resto de las técnicas para todas las instancias del TSP. La siguiente mejor técnica es SI, sin embargo, su rendimiento disminuye a medida que el TSP se vuelve mas complejo. El rendimiento de GB decae abruptamente para TSP más complejos, llegando a valores negativos, lo cual refleja la mala calidad de los individuos generados. Las técnicas Random y SP demuestran ser incapaces de producir individuos con un alto *fitness* en su población inicial. En cualquier caso, **las distintas técnicas de *seeding* suelen tener un mejor rendimiento que las técnicas aleatorias** al momento de encontrar soluciones óptimas mediante la implementación de un algoritmo genético.

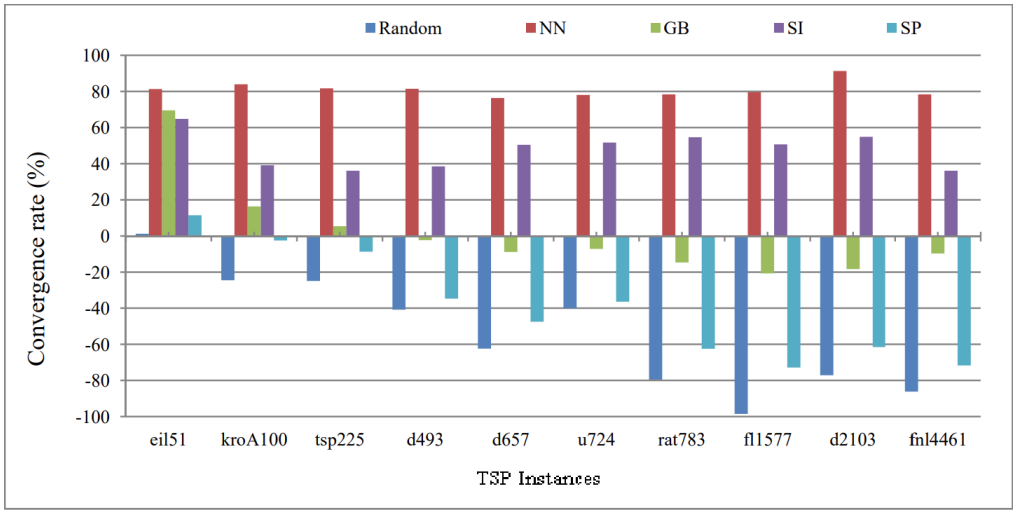


Figura 1.16: Tasa de convergencia (%) de las mejores soluciones generadas por las distintas técnicas de *seeding* [11].

1.7. Curva de Aprendizaje

Una de las maneras que existen para observar gráficamente cómo se llevó a cabo la evolución de un algoritmo genético es mediante las **Curvas de Aprendizaje** [4]. Una curva de aprendizaje es una representación gráfica que relaciona el progreso de cierta tarea (En este caso el *fitness*) con el avance del tiempo (Para este caso, las sucesivas generaciones).

En la figura 1.17 se puede observar la forma típica de una curva de aprendizaje. Esta consta de 3 etapas bien diferenciadas:

1. **Rápido crecimiento**: Al momento de desarrollar una nueva actividad, lo mas normal es que los primeros conocimientos se adquieran a gran velocidad.
2. **Ralentización del crecimiento**: Se está llegando a dominar la nueva actividad.
3. **Meseta**: La ralentización del crecimiento de la curva de aprendizaje termina por estabilizarse en una meseta. Para este caso, significa que el algoritmo está convergiendo a una solución óptima.

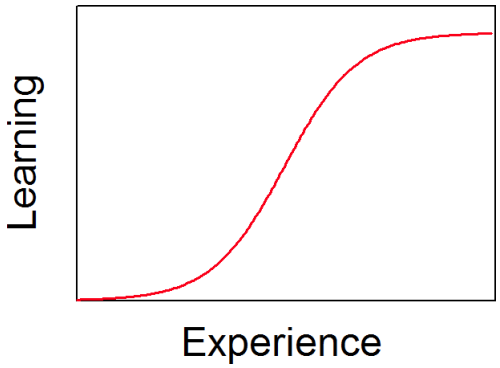


Figura 1.17: Forma típica de una curva de aprendizaje. Esta consta de 3 etapas: Rápido crecimiento, ralentización del crecimiento y meseta [4].

Dentro del contexto de un GA, las curvas de aprendizajes son útiles para determinar si este ha convergido a una solución final. Para esto, se deben comparar las curvas de ***fitness* máximo** y ***fitness* promedio** obtenidos en cada generación. Si ambas curvas no alcanzan el mismo valor en las últimas generaciones, significa que el algoritmo pudo haber seguido evolucionando y eventualmente haber encontrado una mejor solución. En cambio, si ambas curvas llegan a igualarse en las últimas generaciones, eso quiere decir que el GA "terminó de aprender", o en otras palabras, convergió a una solución suficientemente óptima. En la figura 1.18 se puede observar la diferencia entre una curva de aprendizaje incompleta y una exitosa.

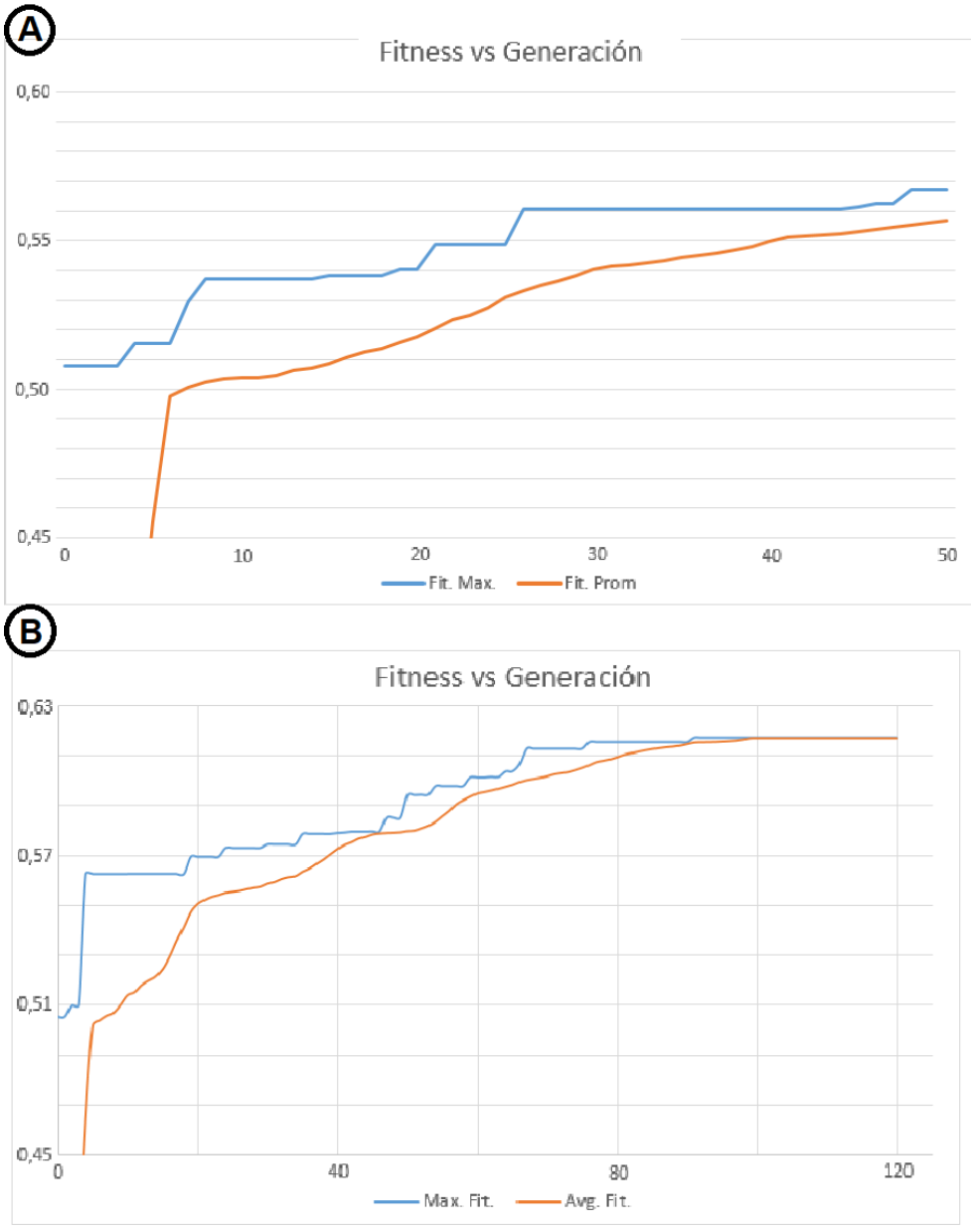


Figura 1.18: (A) Curva de aprendizaje incompleta. El *fitness* promedio (curva roja) está por debajo del *fitness* máximo (curva azul) para cada generación. Por lo tanto, el GA pudo haber ncontrado una mejor solución. (B) Curva de aprendizaje completa. La curva de *fitness* promedio iguala a la de *fitness* máximo en las últimas generaciones. Por lo tanto, el GA ha convergido a una solución suficientemente óptima.

1.8. Evolución Genética en un Ambiente Simulado

La herramienta de evolución genética de metamateriales auxéticos a desarrollar se basará en gran medida en un estudio previo [10], en el cual se llevó a cabo la evolución artificial de robots blandos compuestos de múltiples materiales, los cuales se asemejan a pequeños organismos y que cuya finalidad es desplazarse de un lado a otro de la manera más eficiente posible.

Para este estudio, *Jonathan Hiller* y *Hod Lipson* desarrollaron un software llamado **VoxCAD** [12], el cual ofrece una instancia para simular elementos finitos blandos (denominados *voxels*), cuyas propiedades, tales como: Módulo de Young, Módulo de Poisson, Módulo de Expansión Térmica, Densidad, entre otras, se pueden ajustar a gusto para cada material. En la figura 1.19 se puede observar a uno de estos robots blandos siendo simulados en VoxCAD.

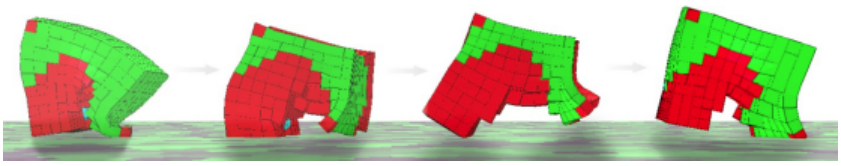


Figura 1.19: Ejemplo de una de las morfologías evolucionadas artificialmente, compuesta por múltiples materiales, desplazándose de izquierda a derecha con un movimiento parecido al trote de un perro [10].

A continuación se describe en detalle las principales herramientas usadas en este estudio, que por cierto, también serán usadas dentro del sistema a diseñar.

- **VoxCAD**: Este simulador puede modelar eficientemente la estática, dinámica y deformaciones no lineales de objetos blandos heterogéneos. Además, es capaz de simular actuadores volumétricos, esto haciendo variar periódicamente la temperatura del entorno y definiendo coeficientes de expansión térmica en ciertos materiales.

  Las características de los materiales usados en este estudio son las siguientes:

  - **Voxels Verdes**: Llevan a cabo una actuación volumétrica periódica. Se asemejan a un músculo que se expande y se contrae.
  - **Voxels Rojos**: Al igual que los verdes, se comporta como un actuador volumétrico, pero a contra-fase. Es decir, se asemejaría a un músculo que se contrae y luego se expande.
  - **Voxels celestes**: Es un material suave y pasivo. Su deformación es a causa de los actuadores anteriormente nombrados. Se asemeja a lo que sería un tejido blando como la piel u órganos.
  - **Voxels azules**: Es un material rígido y pasivo. Se asemeja a lo que sería el esqueleto.
- **GAlib**: Librería de objetos programada en lenguaje C++, que tiene como objetivo construir y diseñar algoritmos genéticos. Posee múltiples tipos de algoritmos, tipos de genomas, criterios de terminación, entre otras características. Además, posee una completa guía de ejemplos para su uso [9].

En la figura 1.20 se observan distintos tipos de robots blandos generados por este estudio, ilustrándose las distintas maneras en que estos avanzan de un punto a otro.

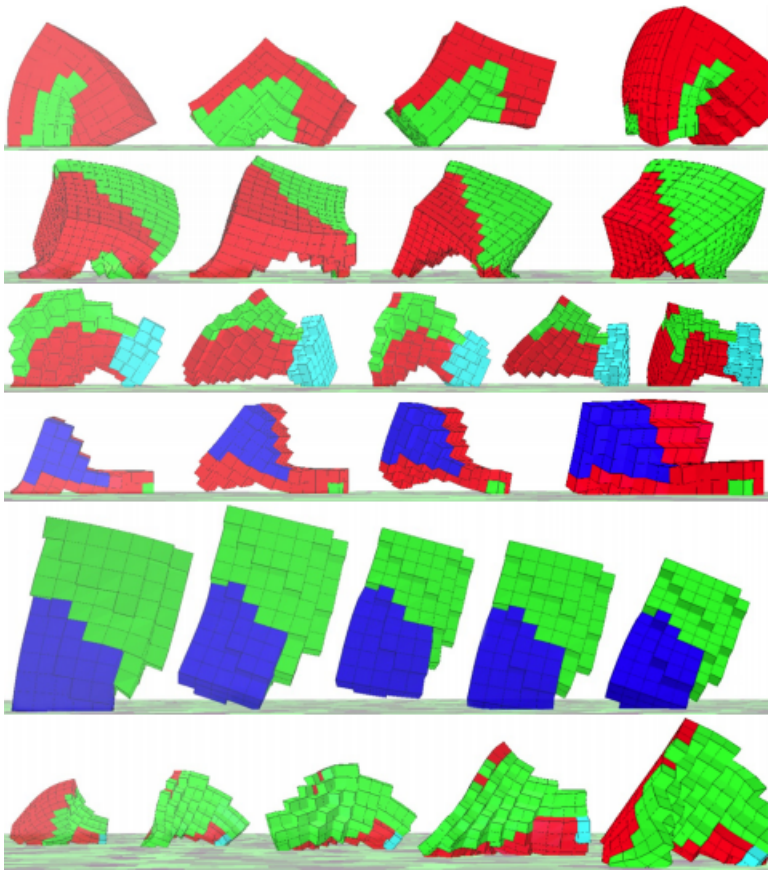


Figura 1.20: Comportamiento de distintos robots blandos moviéndose de izquierda a derecha. De arriba hacia abajo, reciben los siguientes nombres: *Walker, Incher, Push-Pull, Jitter, Jumper,* y *Wings* [10].

Capítulo 2

Metodología

2.1. Organización

El trabajo de título contempla 21 Créditos, los cuales son equivalentes a 35 horas semanales de trabajo. Por lo tanto, este se abordará como si se tratase de un empleo con horario *part-time*.

Estas horas de trabajo incluirán principalmente: Entrenamiento en el uso de *softwares*, programación de código, investigación y reuniones periódicas con la comisión de profesores.

Se procurará tener una constante comunicación con la comisión de profesores, vía correo electrónico o *WhatsApp*, principalmente para reportar la situación actual del trabajo y orientar el avance del proyecto.

2.2. Recursos

- **Equipos**:
  - **Computadora de escritorio**
  - **Impresora 3D**: Con el objetivo de estudiar el comportamiento real de los metamateriales diseñados, se usará una impresora 3D de filamento, modelo *Ender 3 Pro*, desarrollada por la empresa china *Creality*. Por otro lado, a modo de materia prima, se usará *filamento de TPU*, el cual permite imprimir piezas flexibles y elásticas con propiedades similares al caucho. Las especificaciones técnicas de esta impresora y el filamento flexible se pueden encontrar, respectivamente, en los anexos A y B.
- **Softwares**:
  - **VoxCAD**: Simulador de la física de objetos de elementos finitos blandos. Su interfaz gráfica permite observar cómo estos se deforman bajo distintas condiciones físicas. Se usa la versión 0.93 para que las simulaciones sean compatibles con la librería `Voxelyze`.

  - **Ubuntu**: Sistema operativo distribuido por `Linux`. Se usará para programar el código del sistema a implementar, usando lenguaje `C++`.
  - **Fusion360**: *Software* de modelado 3D desarrollado por *Autodesk*. En este programa se pueden elaborar los diseños de metamateriales generados por el sistema a desarrollar y exportarlos en formato `.stl` para su posterior impresión.
  - **Creality Slicer**: *Software* que viene junto a la impresora, en el cual se determinan los distintos parámetros de impresión, tales como la orientación de la pieza a imprimir, el alto de capa, el porcentaje de relleno, entre otros. En este *software* se importan los archivos `.stl` y, una vez establecidos dichos parámetros, se genera un archivo en formato `.gcode`, en el cual se encuentran codificados los parámetros de impresión y la ruta que seguirá el extrusor para imprimir la pieza en cuestión.
- **Librerías C++**:
  - **Voxelyze**: Permite realizar simulaciones físicas de objetos finitos blandos a nivel de programación, es decir, sin necesidad de una interfaz gráfica como VoxCAD.
  - **GAlib**: Librería de objetos que tiene como objetivo diseñar y construir algoritmos genéticos.

2.3. Procedimiento

A continuación se enumera paso a paso el procedimiento que se seguirá en el presente trabajo de título para cumplir los objetivos propuestos. Para esto, se distinguen cuatro etapas:

1. **Aprendizaje**:

   En esta primera etapa se adquieren los conocimientos básicos para el manejo de las distintas herramientas que permitirán llevar a cabo el presente trabajo de título.

   (a) Instalar **WSL** (Windows System for Linux) para poder usar Ubuntu en un computador con *Windows* como sistema operativo [14].

   (b) Familiarizarse con los comandos de Linux para, entre otras cosas, acceder a las carpetas y archivos del computador; y ejecutar, modificar y compilar los códigos en C++.

   (c) Dirigirse al sitio oficial de VoxCAD [12], instalar la versión 0.93 y estudiar los videos tutoriales.

   (d) Estudiar lenguaje de programación en C++ mediante tutoriales gratuitos disponibles en internet.

   (e) Una vez dominada la lectura y escritura en lenguaje `C++`, se procede a estudiar las librerías `GAlib`, `Voxelyze`, y el código desarrollado en el estudio de la referencia [10].

2. **Programación del sistema**:

   Una vez dominadas las herramientas y conocimientos básicos, se da inicio a esta segunda fase que consiste básicamente en modificar el código del estudio de la referencia [10] para que, en lugar de generar y simular robots blandos que se desplazan de un lugar otro, lo haga con metamateriales mecánicos blandos que al comprimirse exhiban una respuesta auxética.

(a) Determinar el tipo de genoma de la librería `GAlib` que se usará para representar la estructura del metamaterial.

(b) Establecer los parámetros y el tipo de algoritmo genético de la librería `GAlib` a implementar.

(c) Programar un generador de geometrías de celdas unitarias o macroestructuras a partir de un genoma.

(d) Implementar `Voxelyze` para elaborar un sistema de evaluación donde sea posible simular la compresión de cada candidato de metamaterial.

(e) Determinar los resultados y mediciones que se obtendrán tras ejecutar la simulación en `Voxelyze`.

(f) Programar una función *fitness* que permita cuantificar el grado de auxeticidad de cada individuo a partir de los resultados de la simulación implementada en los puntos anteriores.

(g) Implementar funciones que lleven registro de las estadísticas de la evolución del algoritmo genético.

(h) Programar líneas de código que impriman en pantalla el comportamiento de las distintas funciones y partes del código, para así identificar posibles fallos o comportamientos no deseados dentro del sistema.

(i) Corregir iterativamente los errores que puedan surgir en cada punto de esta estapa hasta que el sistema opere correctamente.

3. **Testeo del Sistema**:

Una vez que cada parte del sistema implementado pueda ejecutarse y compilar sin problemas, este se pone a prueba realizando distintos testeos y programando nuevas características.

(a) Emplear el sistema implementado en los puntos anteriores para obtener nuevos diseños de metamateriales auxéticos.

(b) Mejorar el desempeño de estructuras ya conocidas usando técnicas de *seeding*.

(c) Explorar el potencial del sistema implementado para generar geometrías más diversas y complejas, considerando principalmente los siguientes puntos:

i. Programar simulaciones alostéricas.

ii. Explorar distintas configuraciones simétricas dentro de las celdas unitarias.

iii. Generar metamateriales cuya estructura varíe tridimensionalmente.

(d) Reportar los resultados de las simulaciones, siguiendo como base el siguiente procedimiento:

i. Establecer parámetros a usar para los testeos finales.

ii. Realizar cada testeo tres veces.

iii. Elegir el mejor de los tres y, a partir de sus estadísticas, generar una curva de aprendizaje.

iv. Exponer gráficamente la evolución del metamaterial a lo largo de la curva de aprendizaje.

v. Estudiar comparativamente la auxeticidad y las propiedades mecánicas de cada metamaterial para los distintos testeos.

4. **Experimentación Real**:

   Corresponde a la última etapa del estudio, en la cual se determinará si efectivamente los metamateriales diseñados son auxéticos en la vida real.

   (a) Estudiar las características técnicas de la impresora y la materia prima a utilizar.

   (b) Establecer los parámetros de diseño de las piezas a fabricar.

   (c) Usar *Fusion 360* para generar modelos 3D de los metamateriales a imprimir.

   (d) Exportar objetos 3D en formato `.stl` y usar *Creality Slicer* para generar los archivos de impresión.

   (e) Elaborar un entorno de experimentación con herramientas e implementos disponibles en el hogar.

   (f) Comprimir los diseños impresos, registrando su deformación en cámara.

   (g) Comparar cualitativamente los resultados reales con los de la simulación.

A continuación, se presenta un diagrama de flujo que resume el procedimiento anteriormente descrito:

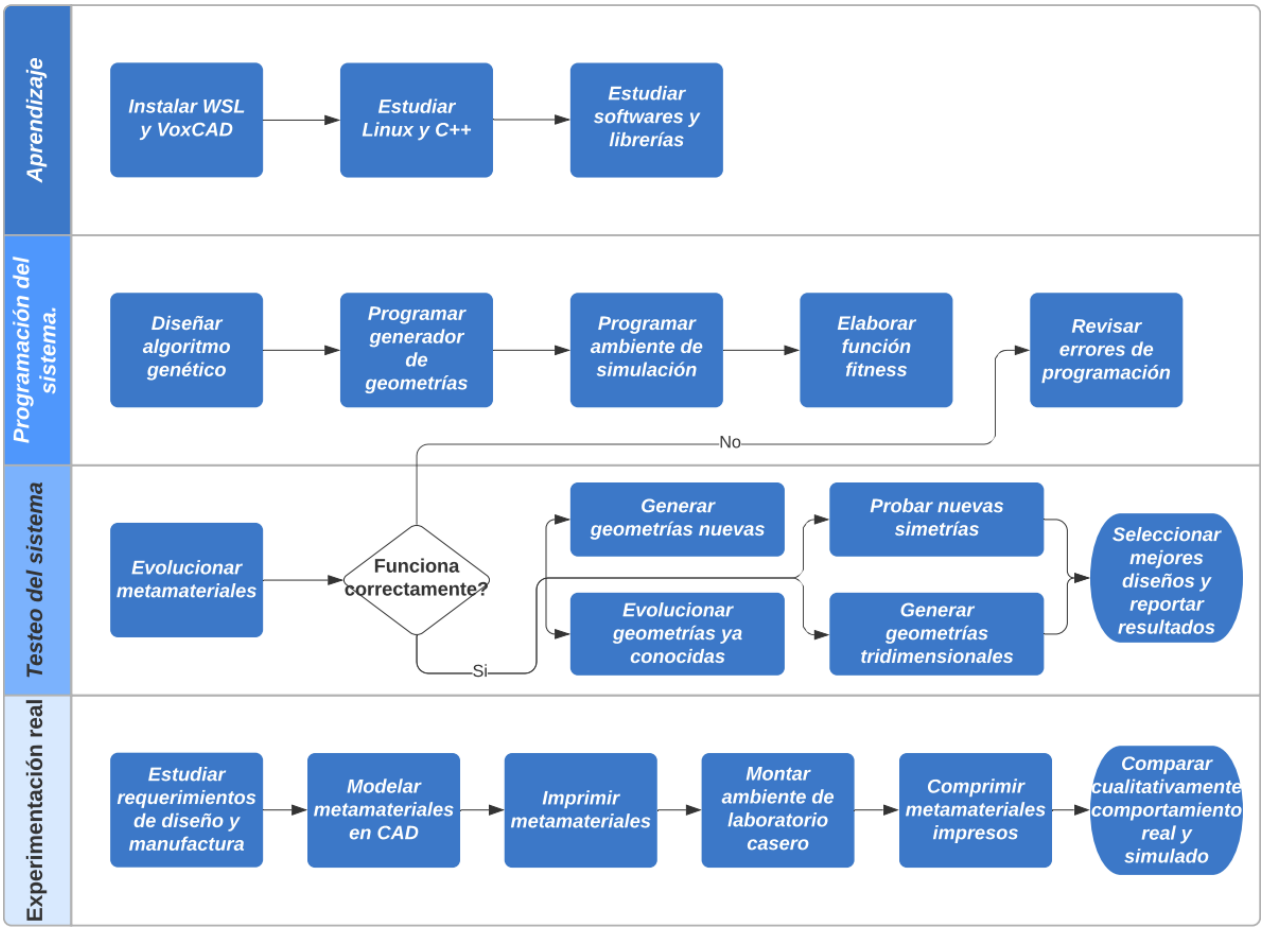


Figura 2.1: Diagrama de flujo de las actividades a llevar a cabo en el presente trabajo de título.

2.4. Carta Gantt

En la figura 2.2 se muestra la programación semanal de las distintas tareas a realizar. S1-14 corresponden a las 14 semanas lectivas del semestre primavera 2020. SE1-2 hacen referencia a las semanas de exámenes, SR alude a la semana recuperativa y, por último, SF corresponde a la entrega del informe final. Se omiten las semanas de vacaciones de mitad de semestre. Las tareas en rojo representan la etapa de aprendizaje, a la cual se le destinan 3 semanas. Las tareas en amarillo corresponden a la fase de programación del sistema y testeos preliminares. La programación del sistema se extiende hasta semana 12 puesto que las modificaciones dentro del código de programación siempre estarán presentes, ya sea para corregir errores, mejorar los algoritmos implementados y programar nuevas funciones. Las tareas en verde están relacionadas a la programación y testeo de las características especiales del sistema. Esta fase comienza una vez que el sistema pueda operar de manera exitosa y sin mayores inconvenientes, y así poder conseguir los primeros resultados preliminares. Las tareas en celeste tienen que ver con la fase del estudio experimental de los metamateriales impresos, la cual comienza inmediatamente después de que el sistema esté completamente funcional. Por último, las tareas en morado corresponden a la preparación de entregables (presentación de avance e informe final) y la recopilación de resultados finales.

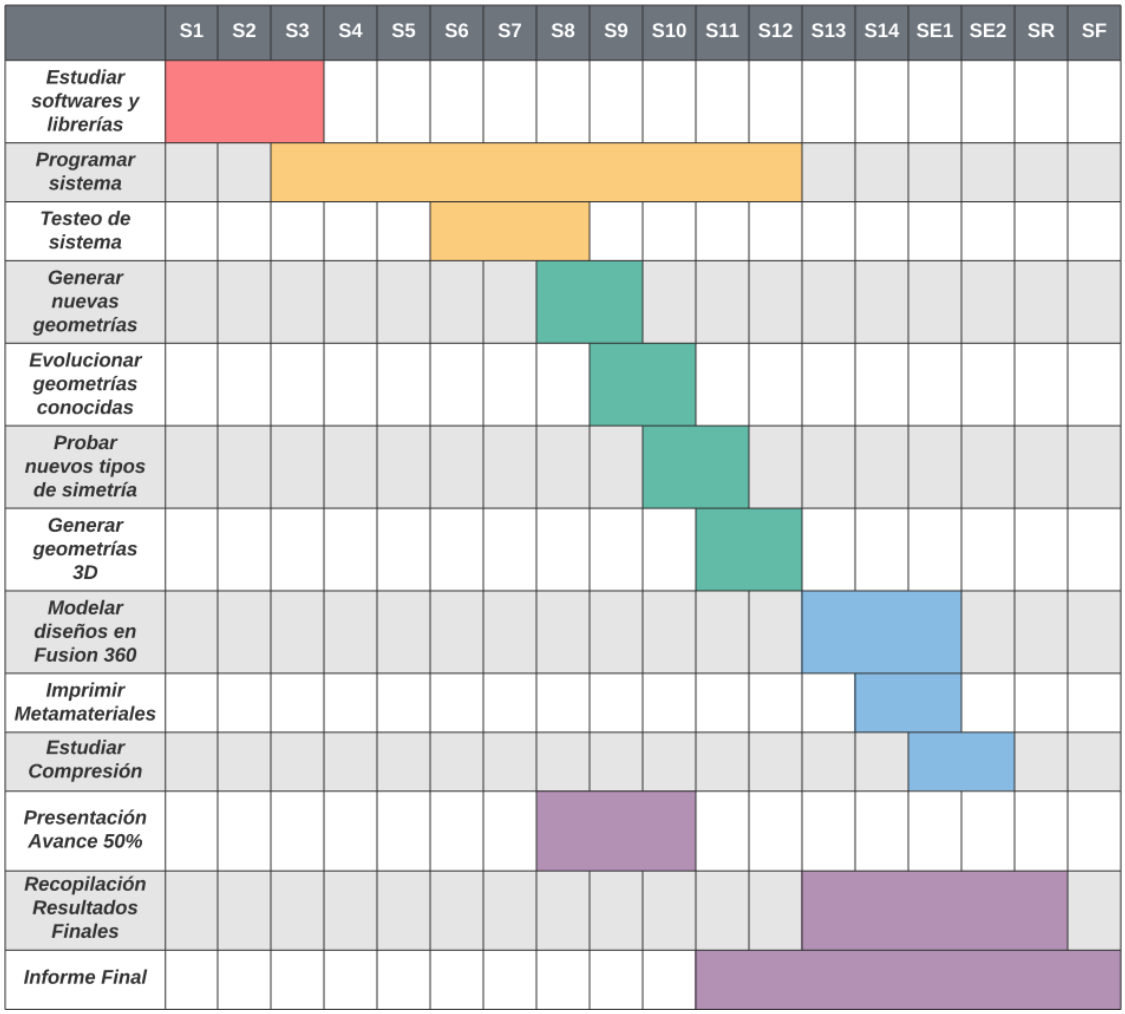


Figura 2.2: Carta Gantt de la programación semanal del trabajo de título.

Capítulo 3

Código del Sistema

Para entender de mejor manera los resultados y las características que los distinguen, es importante tener una noción de cómo opera, a grandes rasgos, el sistema diseñado. A raíz de esto, en las siguientes subsecciones se desglosan y explican algunas de las principales variables, funciones y algoritmos que conforman la herramienta de diseño de metamateriales auxéticos desarrollada a lo largo del presente trabajo de título, llendo desde lo más general a lo más particular. Posteriormente, se muestra esquemáticamente cómo estas partes interactúan entre sí. Por último, en el anexo C, se muestran pseudo códigos de las principales funciones y algoritmos programados en el sistema.

3.1. Parámetros de Entrada

El cuerpo principal del código (conocido como *main* en lenguaje `C++`) comienza de inmediato pidiendo los parámetros de entrada del sistema. Con ellos se controlarán tanto las características físicas y mecánicas del metamaterial, así como las condiciones del ambiente de simulación. A continuación se explican cada una de las variables de entrada, las cuales se agrupan en tres categorías bien definidas:

1. **Parámetros estructurales**:
    - **VoxelSize**: Dimensiones de los **voxels**[1], medidas en metros.
    - **n**: Número de **celdas unitarias**[2]por lado que tendrá el metamaterial a simular.
    - **resolution**: Número de *voxels* por lado que tendrá el **genoma**[3]. Esto se puede interpretar como la resolución de imagen que tendrá el metamaterial, puesto que, así como una imagen digital está compuesta por píxeles, los metamateriales estarán compuestos por una cantidad finita de cubos blandos repartidos ordenadamente dentro de una matriz.

[1]Elementos finitos cúbicos blandos.

[2]Patrón que se repite a lo largo y ancho del metamaterial.

[3]En este caso, el genoma a evolucionar corresponde a la cuarta parte de la celda unitaria, esta última construida a partir de simetrías, rotaciones y/o traslaciones del genoma. Esto se detallará en las secciones subsecuentes.

Esta variable posee múltiples variantes dependiendo del tipo de **representación** escogida para el metamaterial a evolucionar. Estas representaciones son:

- **Representación Bidimensional**: La estructura se desplegará en el plano XY y se extruirá en el eje Z. Por lo tanto, se tienen las variables `resolution_x` y `resolution_y` que representan las dimensiones del genoma en dicho plano, mientras que la variable `resolution_z` indica cuantas veces se repite el diseño en Z. Estas variables se representan gráficamente en la figura 3.1.
- **Representación Tridimensional 1**: La primera representación de metamateriales tridimensionales consiste en entrelazar planos XY y XZ, intercalando las figuras entre planos sucesivos para que estas calcen. Además, es necesario generar ejes de conexión entre las distintas resoluciones para hacer posible su concatenación.

  Por la manera en que se construye esta representación, sus dimensiones en Y y Z serán equivalentes. Por lo tanto, se definen las variables `resolution_yz3d` para su despliegue en los planos XY y XZ; y `resolution_x3d` para el largo axial. Estas variables se ven representadas en la figura 3.2.
- **Representación Tridimensional 2**: En esta representación el genoma deja de ser bidimensional, y pasa a representar un cubo. Por lo tanto, se emplean las variables `resolution_x3d2`, `resolution_y3d2` y `resolution_z3d2` para definir las dimensiones del cubo a evolucionar en cada uno de sus ejes. Dichas variables se muestran gráficamente en la figura 3.3.

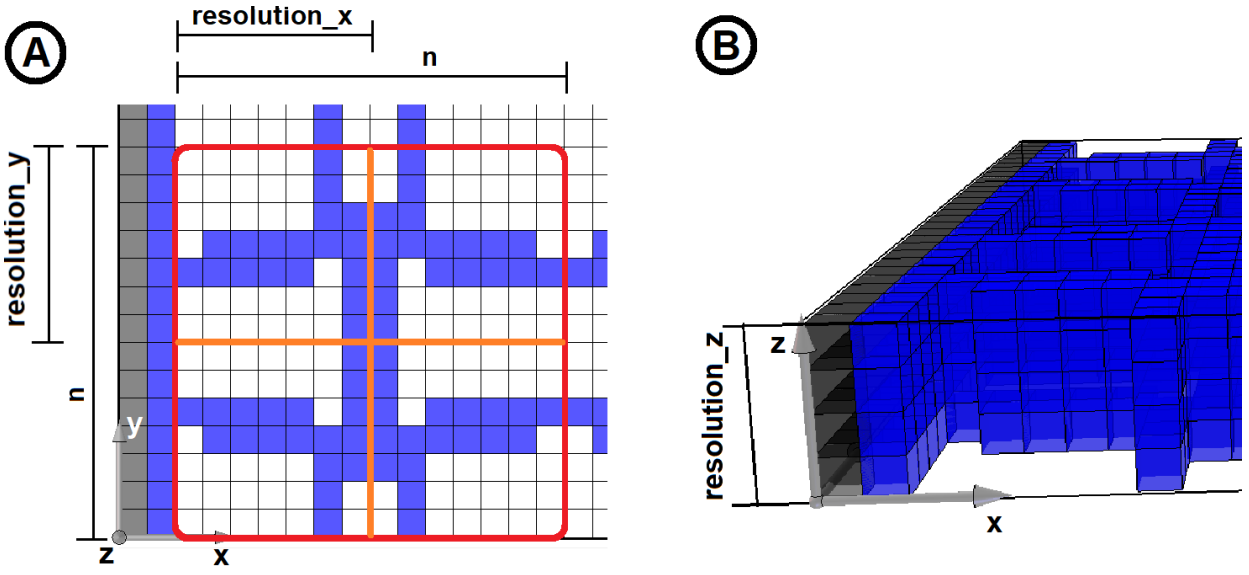


Figura 3.1: Representación gráfica de las variables estructurales que definen a un metamaterial auxético bidimensional. (A) Resolución del genoma (delimitada por líneas naranjas) y tamaño de la celda unitaria generada a partir de este (Delimitada por un cuadrado rojo). (B) Altura del metamaterial.

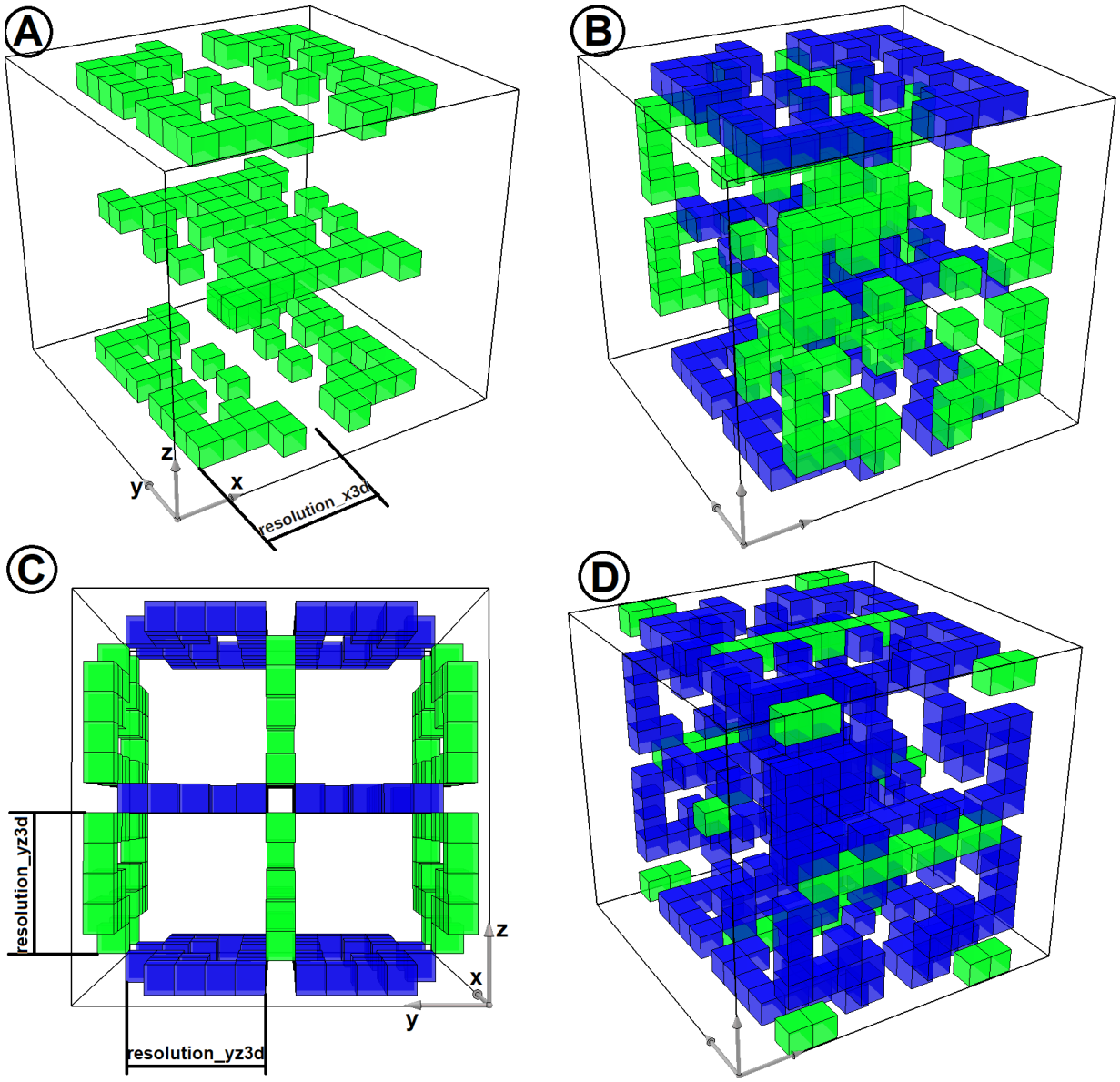


Figura 3.2: Representación gráfica de las variables estructurales para la primera representación de metamateriales auxéticos tridimensionales. (A) En verde se observan los planos XY que conforman al metamaterial diseñado. Para que la posterior concatenación sea exitosa, los planos sucesivos deben presentar la misma topología, pero invertida. (B) Se marcan en verde los planos XZ que van entrelazados al conjunto anterior. (C) Se observa la misma construcción pero desde otro ángulo, quedando en evidencia que, debido a la manera en que se construye el metamaterial, se necesita un eje central en cada intersección de los planos, para que los distintos patrones puedan converger a un mismo punto e interactuar entre ellos. Además, se aprecia de manera gráfica la razón de por qué el largo en Y y Z son equivalentes, justificándose el uso de una única variable `resolution_yz3d` para definir tales dimensiones. (D) Se muestra en color verde la manera en que se construyen los ejes de intersección. Estos se extienden a lo largo del eje X y van, intercaladamente, desde los extremos hacia el centro, y viceversa. Además su construcción se detiene hasta que se topen lateralmente con algún *voxel* de los planos XY o XZ.

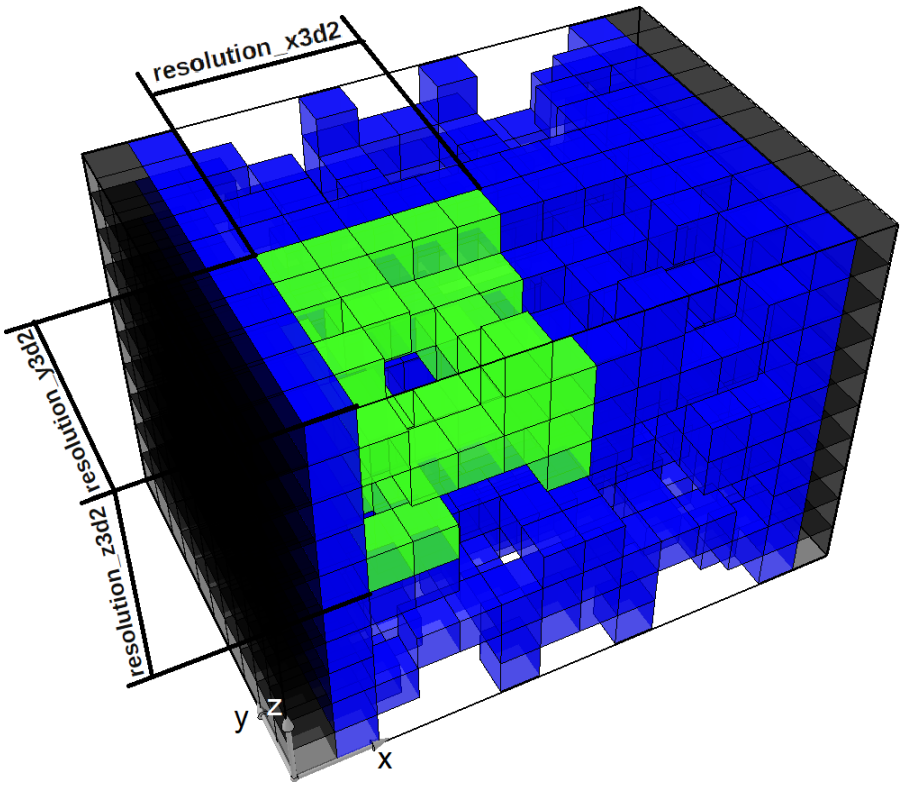


Figura 3.3: Representación gráfica de las variables estructurales para la segunda representación de metamateriales auxéticos tridimensionales. Se destaca en color verde los *voxels* del genoma cúbico.

2. **Parámetros de simulación**:
   - **Dt**: Longitud de la discretización del tiempo de simulación (*timestep*). Representa la fracción de la máxima frecuencia de resonancia del objeto. Valores iguales o mayores a 1 pueden generar divergencias dentro de la simulación.
   - **SimStopTime (SST)**: Cantidad de segundos que deben transcurrir para terminar la simulación.
   - **SimStopStep (SSS)**: Cantidad de *timesteps* que deben transcurrir para terminar la simulación.[4]
   - **maxForce**: Fuerza de compresión, medida en Newtons, con la cual se comprime al candidato a metamaterial dentro de la simulación.
   - **E_meta**: Módulo de elasticidad del metamaterial medido en Pascales.
   - **E_placa**: Módulo de elasticidad de las placas de compresión.
3. **Características Especiales del Sistema (*Features*)**:
   - **isBiased**: Cuando es igual a cero, los candidatos a metamateriales auxéticos serán generados de manera aleatoria. En cambio, si es igual a 1, el sistema usará la técnica de *seeding* para generar metamateriales en base a diseños ya conocidos. Además, para este mismo caso, entran en juego otras dos variables que determinan la manera en que se usa el *seeding*, estas son:

[4]Cuando se cumpla al menos uno de estos dos criterios (SSS o SST), la simulación se detendrá.

  - **thatSeed**: Variable tipo *string* (i.e: cadena de texto) que indica cuál de las "semillas" disponibles se usará como base para la evolución genética.
  - **pbias**: Factor que indica cuánto influirá la semilla en la generación de metamateriales.
- **isAllosteric**: Si es 0, entonces la simulación realizará un ensayo de compresión. Si es 1, entonces la simulación realizará un ensayo de tracción. En cualquier caso, se obtendrán metamateriales con módulo de Poisson negativo.
- **symmetryType**: Esta característica solo está disponible para metamateriales bidimensionales. Acepta parámetros del 0 al 6, y cada uno de ellos representa una manera distinta de usar simetrías dentro de una celda unitaria.
- **is3d**: Si es 0, entonces se generará un metamaterial bidimensional. Si es 1, entonces se usará la primera representación de metamateriales tridimensionales ya descrita. Si es 2, entonces se usará la segunda representación de metamateriales tridimensionales.

En el anexo C.1 se presenta un extracto del código correspondiente al panel de parámetros de entrada.

Mas adelante dentro de la programación del sistema aparece una cuarta sección de parámetros de entrada, correspondiente a la implementación de la librería `GAlib`, en la cual se construye el algoritmo genético (GA por sus siglas en inglés) y se definen los parámetros que lo caracterizan. Las principales variables a considerar son:

- **GARealAlleleSet**: Establece que los posibles valores que adoptarán los genes de los individuos a evolucionar serán números reales, en este caso, contenidos entre 0 y 1.
- **GASteadyStateGA**: Establece que el GA a usar será del tipo *Steady-State* [9].
- **set(gaNpopulationSize, x)**: Setea el tamaño de la población.
- **set(gaNscoreFrequency, x)**: Setea la frecuencia con que se registrarán las estadísticas del GA.
- **set(gaNpMutation, x)**: Setea la probabilidad de mutación.
- **set(gaNpCrossover, x)**: Setea la probabilidad de *Crossover*.
- **set(gaNpConvergence, x)**: Setea el factor de convergencia que debe lograr el algoritmo para que este termine.
- **set(gaNnConvergence, x)**: Setea cuántas de las últimas generaciones se deben considerar para calcular el factor de convergencia.

3.2. Definición del Genoma

Un concepto que se debe tener claro al momento de implementar un GA, es el genoma que representará el tipo de solución que se obtendrá. A continuación se explica simplificadamente, mediante un ejemplo gráfico, las características del genoma que se usará dentro del sistema. Suponiendo que se quiere diseñar un metamaterial bidimensional de resolución 3x3, entonces su genoma será una cadena de números reales entre 0 y 1, de largo $2 \cdot 3x3 = 18$ elementos, es decir, el doble de elementos finitos que conforman la estructura a evolucionar. Esto es así dado que la primera mitad del genoma indica, para cada elemento finito, su probabilidad de estar vacío, mientras que la segunda mitad indica la probabilidad de que estos estén ocupados por un *voxel* de determinado material.

Lo anterior se ejemplifica esquemáticamente en la figura 3.4, en la cual se rellena un genoma con números aleatorios y se muestra la manera en que a partir de él se construye una topología en VoxCAD.

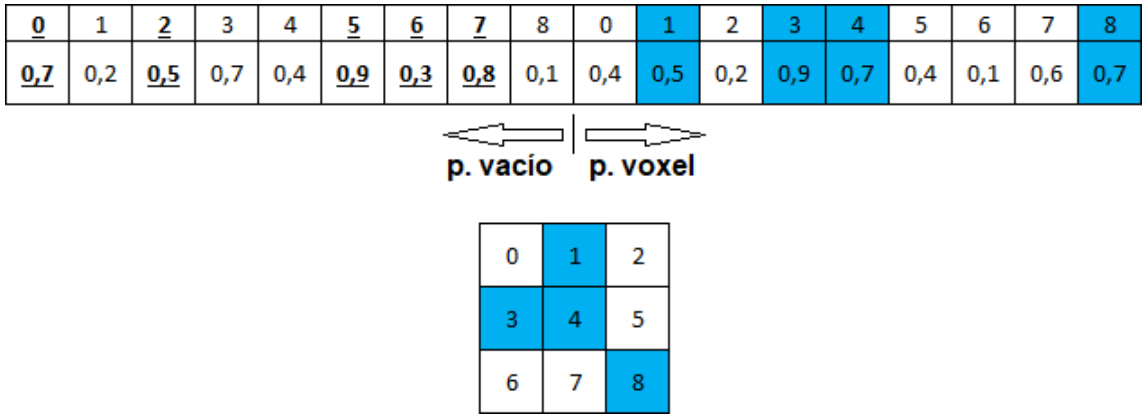


Figura 3.4: Representación gráfica de un genoma bidimensional de 3x3. Los índices de cada elemento se enumeran con módulo 9 para identificar mejor cada mitad. En este caso, para los pares de índices 1, 3, 4 y 8, la probabilidad de *voxel* es mayor que la de vacío, por lo que se colorea en azul para indicar que en dichos elementos se colocará un *voxel*. Finalmente, en la zona inferior de la figura se observa cómo sería su aspecto en la interfaz gráfica de VoxCAD. Cabe recordar que, a partir de esta topología generada a partir de un genoma, se creará posteriormente una estructura más grande mediante la aplicación de simetrías, traslaciones y/o rotaciones de la topología construida inicialmente.

En el anexo C.2 se presenta un extracto del código correspondiente a la implementación del algoritmo genético.

3.3. Función Objetivo

La función objetivo (o función *fitness*) tiene la finalidad de evaluar y cuantificar qué tan auxético es un candidato a metamaterial. Dicha cuantificación queda registrada dentro de una variable llamada *fitness*, la cual permite dilucidar cuáles son los mejores y peores candidatos. Para este sistema, la función objetivo opera básicamente de la siguiente manera:

1. Recibe el genoma y separa cada mitad en dos arreglos independientes. El primero de ellos, denominado como `ContinuousArray`, contiene las probabilidades de vacío para cada elemento finito. Mientras que el segundo arreglo, que recibe el nombre `PassiveSoftArray`, contiene las probabilidades de insertar un *voxel*.
2. Ingresa ambos arreglos a la función `writeVoxelyzeFile` (detallada dentro de los próximos puntos), a partir de la cual se genera un archivo `test_genome.vxa`. Dicho archivo contiene todas las instrucciones y parámetros de la simulación a realizar.
3. Se llama a la librería `voxelyze` para ejecutar el archivo de extensión `.vxa` generado en el punto anterior. Como resultado, se crea un nuevo archivo, llamado `test_fitness.xml`, en el cual se encuentran registrados los resultados y mediciones de la simulación, entre los que se destacan:
   (a) Módulo de Poisson
   (b) Desplazamiento de compresión axial.
   (c) Medición del área transversal inicial
   (d) Medición del área transversal final
4. Se imprimen estos resultados en pantalla.
5. A partir de las mediciones (c) y (d) del listado anterior, se calculan las variables `d1` y `d2`, a partir de las cuales se calculará posteriormente el *fitness* de cada candidato a metamaterial. Dichas distancias se definen como:
   - **d1**: Distancia entre el borde del *workspace*[5]y el borde externo del área transversal del metamaterial, en el estado inicial de la simulación.
   - **d2**: Distancia desde el centro de la celda unitaria hasta el borde externo del área transversal del metamaterial, en el estado final de la simulación.

   En la figura 3.5 se observa una definición gráfica de ambas distancias. Cabe recalcar que tanto `d1` como `d2` están normalizadas respecto a `d`, que se define como la distancia total entre el centro de la celda unitaria y el borde del *workspace*.
6. Se verifica si ocurrieron fallas en la simulación o si se presentó algún factor no deseado. Estos son:
   (a) Si el módulo de Poisson devuelve el string "`-nan`", significa que hubo un error de código asociado a la medición de los resultados de la simulación. Por lo tanto, se imprime un mensaje de error y el *fitness* se iguala a 0.
   (b) Si el metamaterial generado no tiene una forma continua, entonces el *fitness* se iguala a 0.
   (c) Si el metamaterial presenta hileras en los ejes X o Y que estén completamente llenas o vacías, entonces el *fitness* se penaliza según el siguiente criterio:

[5]Área de trabajo que delimita todos los espacios posibles donde se pueden colocar *voxels*. Por ejemplo, en la figura 3.5 correspondería a un cuadrado de 14x18.

i. Si el metamaterial no es auxético, entonces su *fitness* se calcula normalmente y el resultado se divide en 2. Más adelante se verá que los metamateriales no auxéticos siempre tendrán un *fitness* menor a 0,5, por lo que al dividirlo en 2, dicho valor será aún menor de lo normal.

ii. Si el metamaterial es auxético, entonces su *fitness* se iguala a 0,5, que corresponde al peor valor de *fitness* que puede presentar un metamaterial auxético. Esto se hace así puesto que, si bien dicho candidato presenta una característica no deseada, de igual manera es auxético, por lo tanto, puede tener información genética útil que se podría aprovechar.

7. Si el candidato a metamaterial no presenta ninguno de los modos de falla anteriormente descritos, entonces se procede a calcular el *fitness* de la siguiente manera:

Definición 3.1 *Función Fitness*

$$Fitness = \frac{1}{1 + \mathrm{d1/d} + \mathrm{d2/d}}$$

Donde las distancias `d`, `d1` y `d2` están definidas gráficamente en la figura 3.5.

De esta definición se pueden desprender 3 comportamientos:

(a) Si el metamaterial no es ni auxético ni no auxético (En otras palabras, su área transversal no cambia al aplicar esfuerzos de compresión o tracción en la dirección axial. i.e: Módulo de Poisson igual a 0), entonces se tendría que d1 + d2 = d. Al reemplazar dicha expresión en la definición anterior, entonces se tendría un *fitness* igual a 0,5

(b) Si el metamaterial es no auxético, entonces d1 + d2 será mayor a `d`. Si esta suma se hace tender al infinito, entonces la variable *fitness* tenderá a 0.

(c) Si el metamaterial es auxético, entonces d1 + d2 será menor a `d`. Si está suma se hace tender a 0, entonces la función tenderá a 1.

En resumen, mientras más auxético sea un metamaterial, más cercano a 1 será su *fitness*. Por el contrario, mientras mas no auxético sea, más cercano a 0 será su *fitness*. Por su parte, un *fitness* igual a 0,5 sirve de frontera para distinguir entre un candidato auxético y uno no auxético.

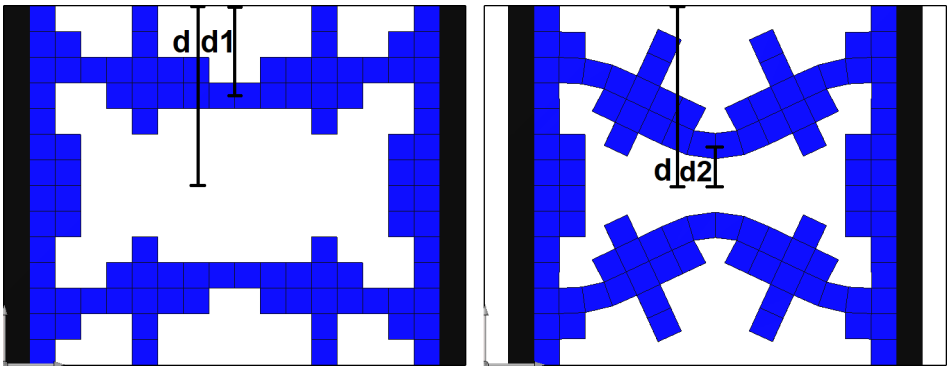


Figura 3.5: Definición gráfica de las distancias d1 y d2, a partir de las cuales se calcula el fitness.

En el anexo C.3 se presenta un pseudo-código que describe la programación de la función objetivo:

3.4. Función writeVoxelyzeFile

Esta función es usada por la función objetivo para que, a partir de la información contenida en el genoma, se genere el archivo de simulación. Esto ocurre de la siguiente manera:

1. Ingresa los arreglos `ContinuousArray` y `PassiveSoftArray` a la función `createArrayForVoxelyze`, la cual entregará un nuevo arreglo llamado `ArrayForVoxelyze`, donde se encuentra codificada la estructura y composición del metamaterial completo.
2. Este nuevo arreglo se ingresa a la función `findVoxelToTrack`, la cual entrega la posición del *voxel* más externo del área transversal del metamaterial. Es el desplazamiento de dicho *voxel* el que necesita ser medido posteriormente dentro de la simulación para determinar los valores de las variables `d1` y `d2`.
3. Se genera un archivo llamado `test_genome.vxa`, dentro del cual se ingresan las instrucciones y parámetros a implementar en la simulación[6], entre los que se destacan:
   (a) **DtFrac**: Duración del *timestep*. Se define en base a la frecuencia máxima de resonancia del objeto. Valores iguales o mayores a 1 causan divergencia dentro de la simulación.
   (b) **BondDampingZ**: Factor de amortiguamiento entre *voxels* contiguos.
   (c) **ColDampingZ**: Factor de amortiguamiento entre objetos que colisionan.
   (d) **SlowDampingZ**: Factor de amortiguamiento entre el objeto y el piso del ambiente simulado.
   (e) **SimStopTime**: Tiempo en segundos para detener la simulación.
   (f) **SimStopStep**: Número de *timesteps* para detener la simulación.
   (g) **TrackVoxel**: Índice del *voxel* cuyo desplazamiento requiere ser medido.
   (h) **ForceX**: Magnitud de la fuerza axial, medida en Newtons, a la cual se someterá el candidato a metamaterial durante la simulación.
   (i) **Material ID**: Número natural asignado a cada uno de los materiales presentes en el objeto a simular.
   (j) **Elastic_Mod**: Módulo de elasticidad del material medido en Pascales.
   (k) **Density**: Densidad del material, medida en Kg/m^3.
   (l) **Poissons_Ratio**: Módulo de Poisson del material.
   (m) **X_Voxels, Y_Voxels y Z_Voxels**: Número de *voxels* que tendrá el *workspace* en sus 3 ejes de coordenadas.
   (n) **Layer**: Se escriben, a lo largo, cada uno de los *voxels* que conforman el plano XY asociado a cada altura en Z del objeto a simular, refiriéndose a cada elemento finito por el ID del material del que están hechos.
4. Por último, se imprime en pantalla el metamaterial generado, usando asteriscos para indicar los elementos en los cuales se insertó un *voxel* y barra espaciadora para los elementos vacíos.

En el anexo C.4 se presenta un pseudo código donde se muestra cómo esta función genera el archivo de simulación.

[6]Esta data se encuentra escrita en lenguaje XML, y puede ser leida tanto por VoxCAD como `voxelyze`.

3.5. Función createArrayForVoxelyze

Esta función recibe los 2 arreglos desprendidos del genoma y, a partir de estos, genera un nuevo arreglo denominado `ArrayForVoxelyze`, el cual cuenta con las dimensiones totales del metamaterial. Además, en cada elemento de este arreglo, en lugar de números reales, se indica el ID del material a insertar. Esta función procede de la siguiente manera:

1. A partir de los parámetros constructivos especificados al principio del código, genera un arreglo lleno de 0's (material vacío, ID=0) con un tamaño suficientemente grande como para representar la totalidad del metamaterial a construir. Cabe destacar que estos arreglos son una cadena unidimensional, y la forma en que esta representa un objeto tridimensional se ve ilustrada en la imagen 3.6.
2. Se generan 2 pares de placas de compresión a ambos lados del metamaterial, teniendo en consideración que sean perpendiculares a la fuerza de compresión de la simulación. El par externo corresponde a un material rígido (ID=2) para simular el contacto con una prensa hidráulica, y el par interior es del mismo material blando del metamaterial (ID=1).
3. Compara, uno a uno, los elementos de los arreglos `ContinuousArray` y `PassiveStiff-Array` de manera tal que, dependiendo de las probabilidades de vacío e inserción de *voxel*, va escribiendo números 1 en los elementos correspondientes. Para optimizar la ejecución del sistema, dicho proceso de inserción ocurre de tal que cada uno de los arreglos involucrados se recorren solamente una vez. Esta mecánica se explicó anteriormente, a grandes rasgos, en la figura 3.4.

   La inserción de los *voxels* se hace tomando en cuenta el tipo de simetría escogida desde un principio. Por ejemplo, para el tipo de simetría estándar, el procedimiento sería:

   (a) Cuadrante 1: Copiar el genoma tal cual como fue generado.
   (b) Cuadrante 2: Aplicar simetría axial respecto al eje Y.
   (c) Cuadrante 3: Aplicar simetría axial respecto al eje X.
   (d) Cuadrante 4: Aplicar simetría central respecto al centro de la celda unitaria.

   En la figura 3.7 se puede ver secuencialmente cómo se va generando la estructura completa del candidato a metamaterial y la forma en que se aplica la simetría para cada cuadrante.
4. Por último, para evitar que los candidatos a metamateriales presenten características o comportamientos no deseados, se ingresa el arreglo `ArrayForVoxelyze` en las funciones `makeOneShapeOnly` y `CheckFullStripes`, las cuales se encargan, respectivamente, de asegurar la continuidad del metamaterial y de verificar la existencia de hileras completamente llenas o vacías.

En el anexo C.5 se presenta un pseudo código simplificado de cómo se generaría un metamaterial bidimensional.

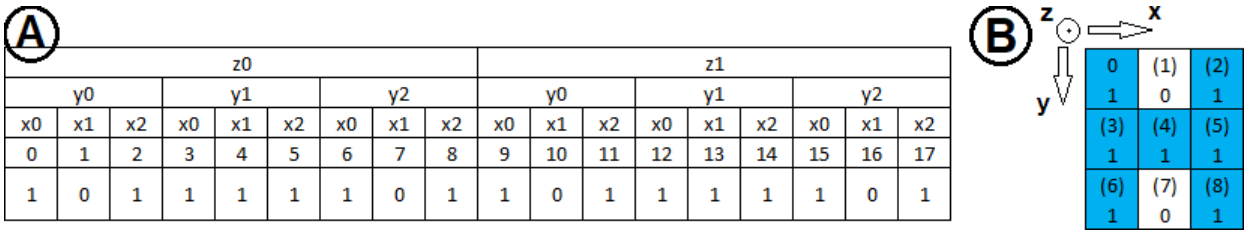


Figura 3.6: Descripción gráfica de cómo a través de una cadena unidimensional se puede representar un objeto tridimensional. (A) Enumerando de abajo hacia arriba, la primera fila corresponde a un ejemplo simplificado de un arrglo `ArrayForVoxelyze`. En la segunda fila se indican los índices de cada elemento. Las filas tercera, cuarta y quinta definen, respectivamente, las coordenadas en X, Y, y Z para cada elemento. (B) Representación gráfica del aspecto que tendría este arreglo en VoxCAD.

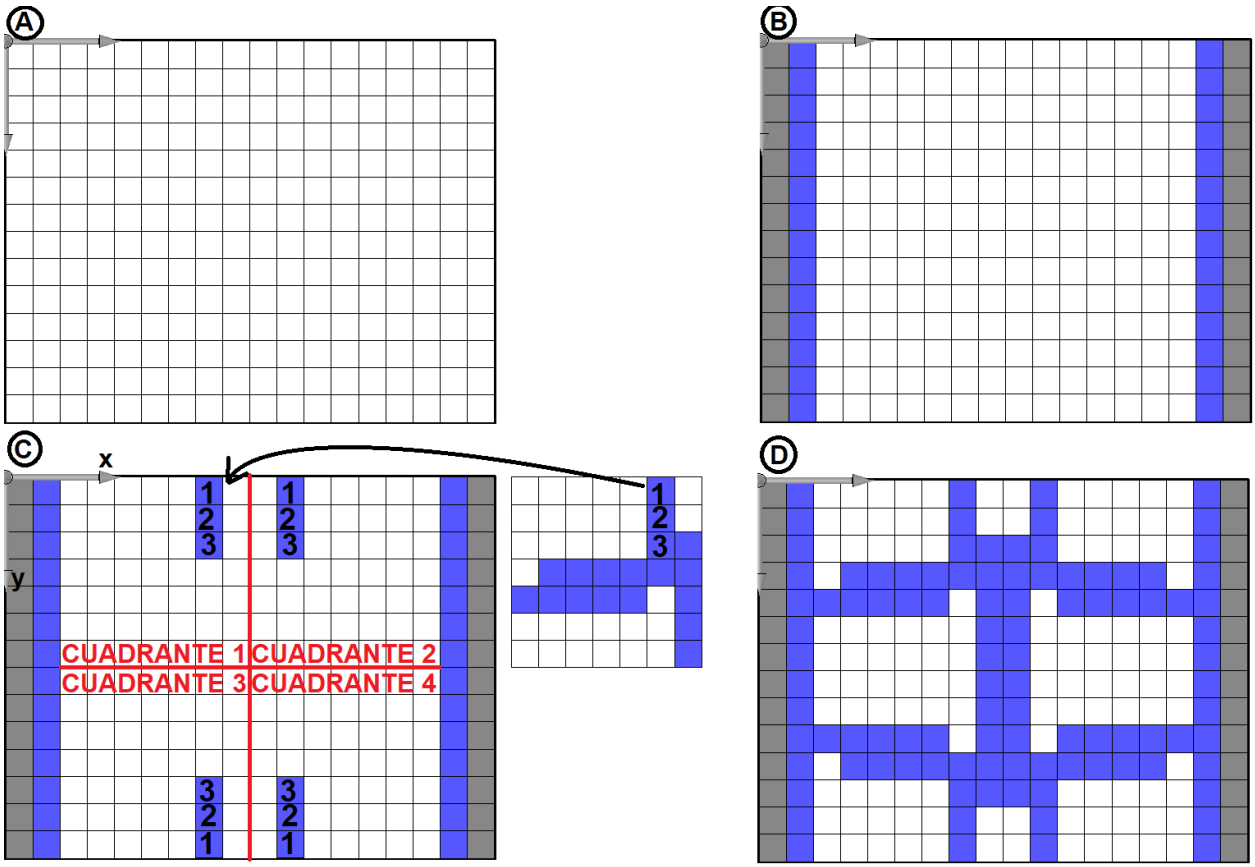


Figura 3.7: Representación gráfica del algoritmo que genera la totalidad del metamaterial. (A) Se genera un arreglo vacío con las dimensiones totales del metamaterial. (B) Se construyen placas de compresión a ambos extremos del metamaterial. (C) Cada elemento del genoma se va insertando directamente en cada uno de los lugares correspondientes, dependiendo del tipo de simetría escogida. (D) Se presenta el metamaterial una vez terminado el proceso anterior.

3.6. Función makeOneShapeOnly

Esta función fue heredada directamente del código original del estudio presentado en la subsección 1.8 [10]. En términos generales, esta función elimina las secciones inconexas del candidato a metamaterial generado, conservando solo aquellas partes que conforman una geometría continua que pueda extenderse de un extremo a otro. En la figura 3.8 se puede observar gráficamente cómo ocurre este proceso.

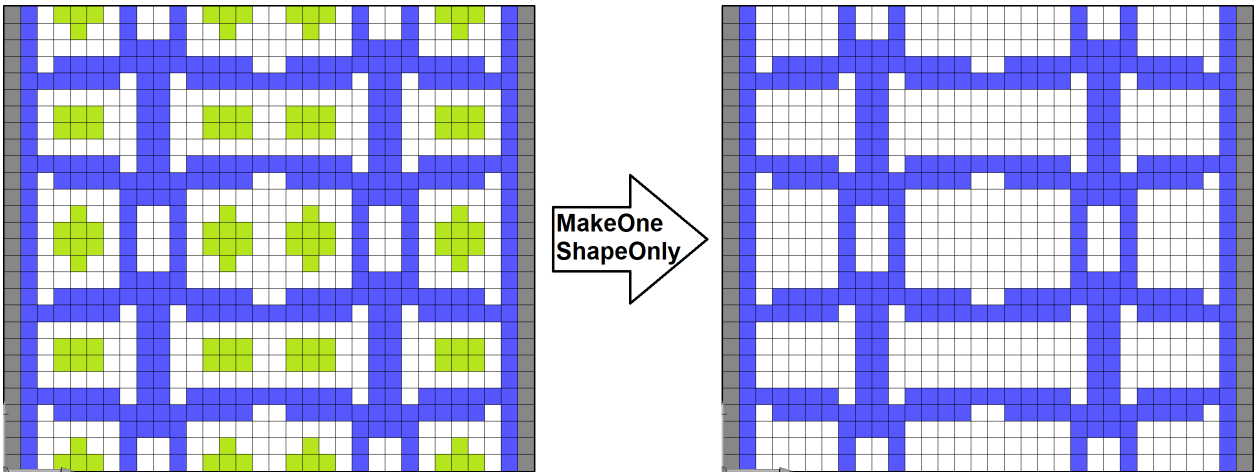


Figura 3.8: Representación gráfica del resultado de usar la función `makeOneShapeOnly`. A la izquierda se muestra el aspecto del metamaterial antes de aplicar la función, indicando con color verde las regiones que no están conectadas al cuerpo principal. A la derecha, en cambio, se muestra el resultado de usar dicha función, obteniendo como resultado un metamaterial cuya topología es continua y uniforme a lo largo de toda la matriz.

Un caso común que ocurre dentro de las primeras generaciones, debido principalmente a la aleatoriedad del proceso de generación de individuos, es que aparecen topologías tan irregulares y dispersas que no cuentan con un cuerpo principal. Como resultado de ello, el candidato a metamaterial no puede arrojar mediciones válidas dentro de la simulación, y por lo tanto su *fitness* debe ser severamente castigado. Este caso puede verse ilustrado en la figura 3.9.

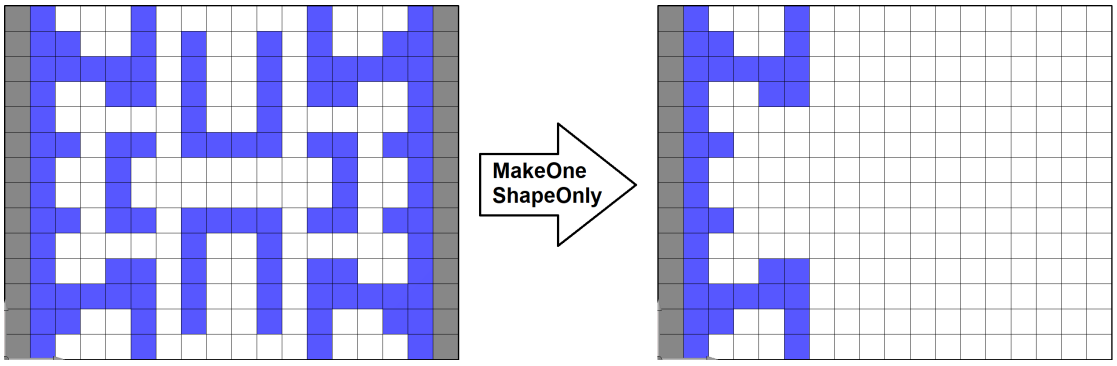


Figura 3.9: Ejemplo de metamaterial discontinuo. Al aplicar la función `makeOneShapeOnly` se pierde casi la totalidad de la estructura del candidato a metamaterial, convirtiéndolo en un diseño infactible para ser estudiado.

3.7. Función checkFullStripes

De no aplicar restricciones de diseño adecuadas, el sistema generará estructuras altamente inestables y con un tipo de deformación que, si bien en las simulaciones se comportan casualmente de manera auxética, en la realidad resultan ser erráticas e impredecibles. Estas topologías, ilustradas en la figura 3.10, se caracterizan por presentar hileras finas que se extienden de un extremo a otro, pasando por medio de amplias regiones de vacío. Dicha inestabilidad se traduce en un valor de *fitness* considerablemente alto con respecto a otras geometrías más estables y factibles, haciendo que el GA opte por descartar estas últimas.

Para evitar este tipo de convergencias, se programa la función `checkFullStripes`, la cual, sirviéndose de la simetría interna de la celda unitaria, recorre el primer cuadrante en busca de filas o columnas completamente llenas o vacías. Esta función opera básicamente de la siguiente manera:

1. Recibe el arreglo `ArrayForVoxelyze`, generado por la función `createArrayForVoxelyze`.
2. Recorre cada una de las filas y columnas del primer cuadrante, realizando un conteo de sus elementos.
3. Si en el recorrido se identifican filas o columnas que están conformadas únicamente de 0's o 1's, entonces aumenta el valor de la variable `fullStripes`.
4. Posteriormente, en la función objetivo, se verificará si el valor de dicha variable es igual a 0 para calcular el *fitness* normalmente, o si es mayor a 0 para aplicar una penalización.

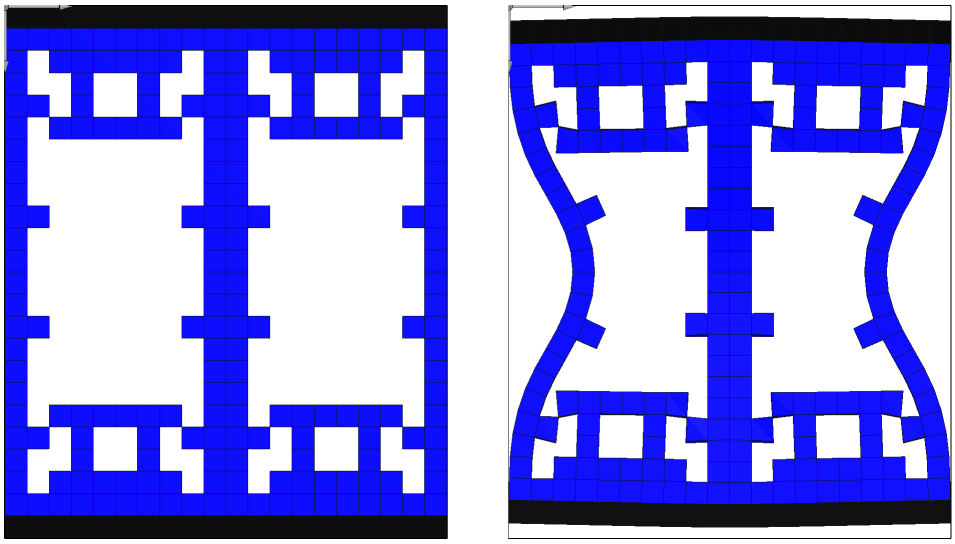


Figura 3.10: Ejemplo de metamaterial que presenta hileras completamente llenas y amplias regiones vacías. Este tipo de topologías presentan una gran disminución de su área transversal frente a pequeños estímulos. Sin embargo, en la realidad, su comportamiento es más bien inestable e impredecible.

En el anexo C.6 se presenta parte del algoritmo de búsqueda de hileras completamente llenas o vacías.

3.8. Función findVoxelToTrack

Con el objetivo de medir la deformación del área transversal, esta función busca y retorna la posición del *voxel* más externo situado en la zona central del metamaterial. Esta función opera básicamente de la siguiente manera.

1. Recibe el arreglo `ArrayForVoxelyze`, previamente generado por la función `createArrayForVoxelyze` y tratado por la función `makeOneShapeOnly`.
2. Antes de realizar la búsqueda, verifica si el material es continuo. Para esto, va a la placa de compresión más alejada del origen y confirma su existencia.
3. Si el paso anterior es positivo, entonces recorre el metamaterial a lo largo del eje X, llevando el conteo de los *voxels* cuyo ID es distinto a 0.
4. Cuando llega a la mitad de la distancia axial, verifica si hay un *voxel*. Si no hay nada, el conteo sigue; si hay un *voxel*, devuelve el valor del conteo. Dicho valor será usado más adelante para ingresarlo al código de la simulación dentro del `archivo test_genome.vxa`, para posteriormente medir su posición inicial y final y así determinar la deformación transversal.
5. Si la placa más lejana al origen no existe, o bien, si en el eje Y recorre una distancia igual a `resolution_y` sin identificar ningún elemento finito, es porque el metamaterial no tiene ningún *voxel* en su área transversal, por lo que no sería un diseño válido. En este caso, la variable `fail` cambia de 0 a 1 y posteriormente, en la función objetivo, el valor del *fitness* para el candidato en cuestión bajará a 0.

En la figura 3.11 se representa gráficamente el método anteriormente descrito.

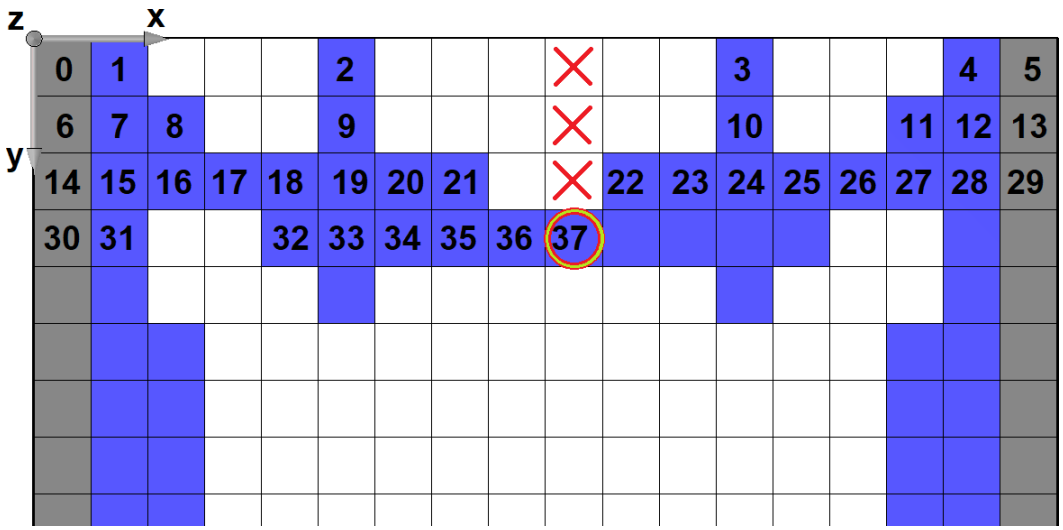


Figura 3.11: Representación gráfica del método de búsqueda del *voxel* a rastrear. El algoritmo realiza un conteo desde el origen, y cada vez que cruza el plano transversal verifica la existencia del *voxel* cuyo desplazamiento requiere ser medido. En este caso, dicho *voxel* corresponde al número 37.

En el anexo C.7 se presenta parte del algoritmo de búsqueda.

3.9. Seeding

La técnica de *seeding* ocurre al comienzo de la función objetivo, al momento de separar el genoma en dos arreglos independientes. Durante este proceso, los elementos de ambos arreglos son modificados para forzar a que más adelante, al momento de generar la estructura completa, esta se parezca en cierto grado a la de un metamaterial auxético conocido. Este proceso se puede separar en dos fases:

1. Preparación de la semilla:
   (a) Buscar diseños de metamateriales mecánicos auxéticos en la literatura [6].
   (b) Representar el metamaterial en VoxCAD.
   (c) Simular la compresión del metamaterial conocido y verificar su auxeticidad.
   (d) Por último, en un archivo de texto, escribir el genoma que generaría la estructura del metamaterial.
2. Implementación de la semilla:
   (a) Se genera un arreglo vacío para luego introducir en él la información genética de la semilla diseñada.
   (b) Se verifica que el metamaterial a generar sea bidimensional y que la variable `isBiased` sea igual a 1.
   (c) Para cada elemento i de la semilla, si este está ocupado por un *voxel*, entonces el elemento i correspondiente al arreglo `PassiveSoftArray` del candidato a metamaterial en cuestión se multiplicará por $1 + pbias$, mientras que el del arreglo `ContinuousArray` se multiplicará por $1 - pbias$. En otras palabras, la probabilidad de inserción de *voxel* aumenta y la de vacío disminuye, ambas por un factor `pbias`.
   (d) Lo contrario ocurre si el elemento i de la semilla corresponde a un *voxel* vacío. Es decir, la probabilidad de vacío aumenta y la de *voxel* disminuye, proporcionales a la variable `pbias`.

El procedimiento anterior se explica gráficamente en las figuras 3.12 y 3.13 para cada etapa respectivamente.

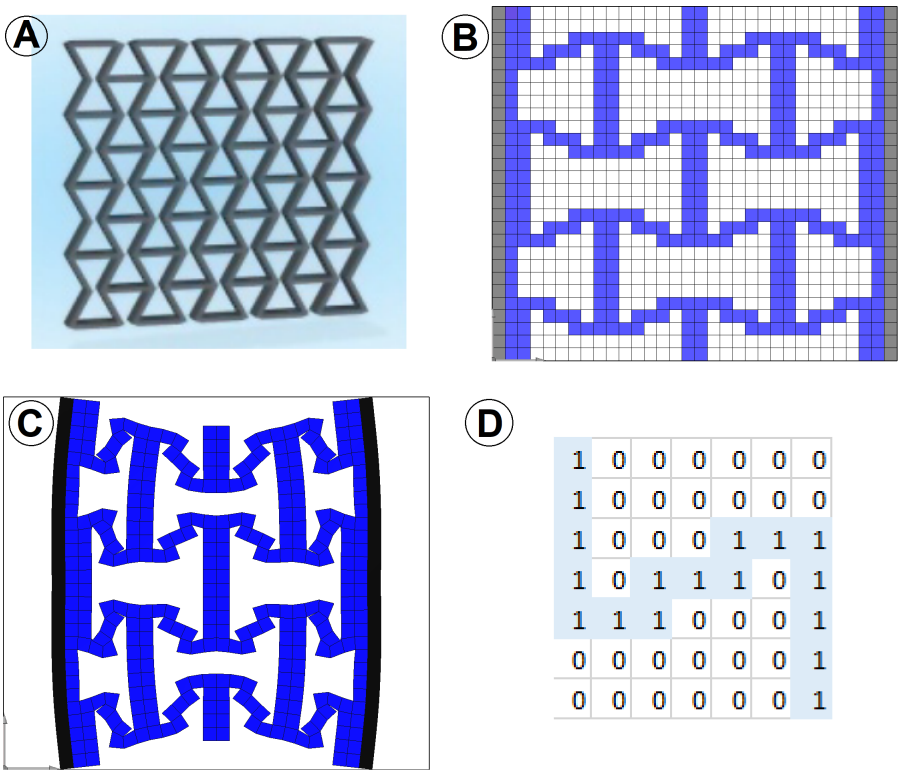


Figura 3.12: Procedimiento para diseñar una semilla. (a) Buscar metamaterial auxético conocido. (b) Representación en VoxCAD. (c) Simulación para verificar su auxeticidad. (d) Representación genética de la semilla.

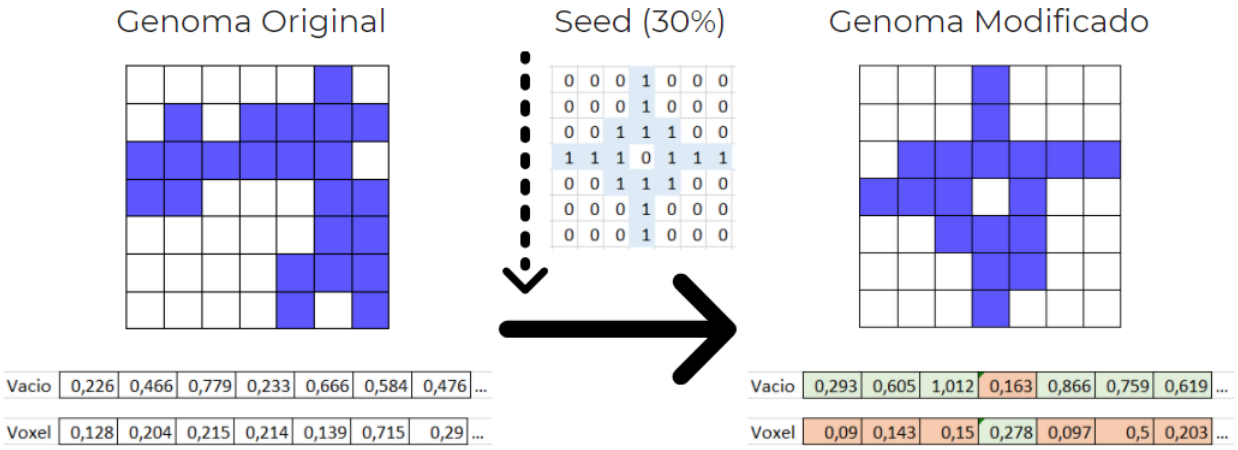


Figura 3.13: Representación gráfica de la implementación de la técnica de *seeding*. A la izquierda se observa de manera gráfica el genoma original, junto a sus probabilidades de vacío e inserción de la primera fila. Luego, se aplica un factor de sesgo del 30 % respecto a la semilla presentada al centro de la figura. Prestando atención a la primera fila de la semilla, se pueden observar tres elementos vacíos, seguidos de un voxel y otros tres elementos vacíos más. A partir de dicha topología, se indica con fondo verde el aumento del 30 % de la probabilidad de vacío para los tres primeros y tres últimos elementos, así como la probabilidad de inserción de *voxel* para el elemento del medio. Lo contrario ocurre para los elementos en fondo rojo.

En el anexo C.8 se muestran las líneas de código agregadas a la función objetivo para la implementación del *seeding*.

3.10. Simulación Alostérica

En este tipo de simulación, el ensayo es de tracción en vez de compresión, y se buscará que la sección transversal aumente en lugar de disminuir. Esto se logra de la siguiente manera:

1. De ser necesario, cambiar el sentido de la fuerza axial aplicada sobre el metamaterial, para asegurar que esta ejerza tracción.
2. Dejar que la función objetivo calcule el *fitness* normalmente, como si fuera un metamaterial auxético.
3. Ya que se desea buscar un comportamiento alostérico, el *fitness* se modifica de la siguiente manera: $fitness = 1 - fitness$. Así, se logra que a medida que el área transversal aumente, el *fitness* irá tendiendo a 1 en lugar de tender a cero, y viceversa.

En el anexo C.9 se muestran las líneas de código adicionales que se agregan al final de la función objetivo.

3.11. Maneras de Construir una Celda Unitaria

En este sistema se programan siete maneras distintas para construir una celda unitaria, esto mediante el manejo de distintas combinaciones de simetrías, traslaciones y rotaciones. Estas se definen de la siguiente manera:

- `SymmetryType=0`: Simetría axial respecto a los ejes X e Y.
- `SymmetryType=1`: Traslación del genoma desde el primer al segundo cuadrante[7]y simetría axial respecto al eje X.
- `SymmetryType=2`: Traslación del genoma desde el primer al tercer cuadrante y simetría axial respecto al eje Y.
- `SymmetryType=3`: Rotaciones en 90°.
- `SymmetryType=4`: Traslación del genoma desde el primer al segundo cuadrante y simetría central en X.
- `SymmetryType=5`: Traslación del genoma desde el primer al tercer cuadrante y simetría central en Y.
- `SymmetryType=6`: Traslación del genoma a todos los cuadrante sin aplicar ningún tipo de simetría.

En la figura 3.14 se ilustran estos siete tipos de construcciones.

En el anexo C.10 se muestra cómo se modifica la función `createArrayForVoxelyze` para generar todas las topologías anteriormente nombradas.

[7]La enumeración de los cuadrantes se puede ver en la figura 3.7.

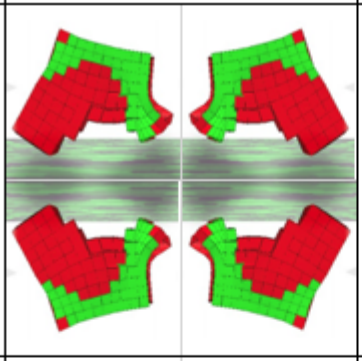
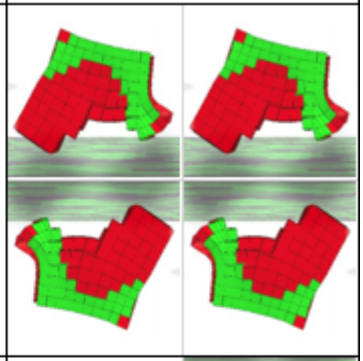
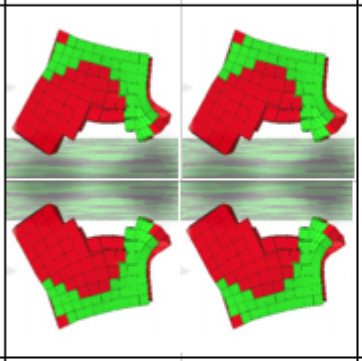
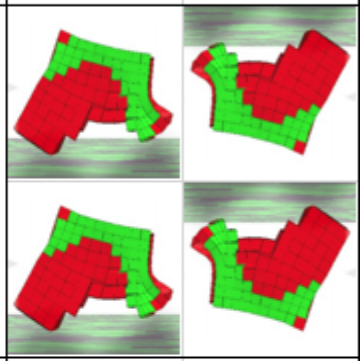
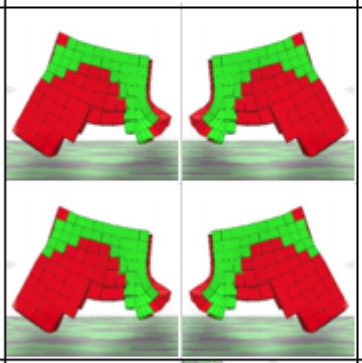
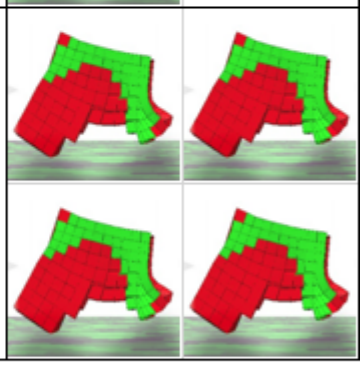
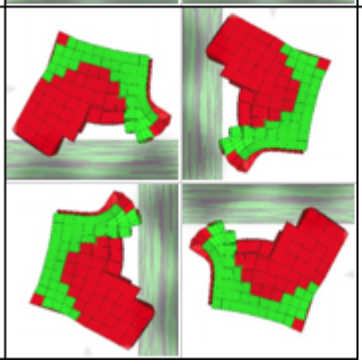
SymmetryType	Representación Gráfica	SymmetryType	Representación Gráfica
0		4	
1		5	
2		6	
3			

Figura 3.14: Representación gráfica de los siete tipos de simetrías programadas para generar celdas unitarias, aplicadas esquemáticamente sobre una figura.

3.12. Impresión en pantalla

En Ubuntu se puede ver en tiempo real la evolución del algoritmo genético, ya que para cada individuo se puede imprimir en pantalla cualquier parámetro del código que uno desee monitorear. En la figura 3.15 se puede apreciar un ejemplo de esto. Los datos actualmente impresos en pantalla corresponden a:

1. El nombre del individuo (test 1)
2. La estructura del metamaterial representada con asteriscos y espacios.
3. Data relacionada con el consumo de recursos computacionales.
4. Módulo de Poisson registrado luego de la simulación.
5. Porcentaje de reducción axial.
6. Porcentaje de *voxels* que han sido llenados.
7. Mensaje de precaución. En este caso, el metamaterial generado tiene filas o columnas totalmente llenas o vacías, por lo que su *fitness* se verá penalizado.
8. Valor del *fitness*. Dado que este metamaterial no es auxético y presenta características indeseadas, su *fitness* es considerablemente menor a 0,5.

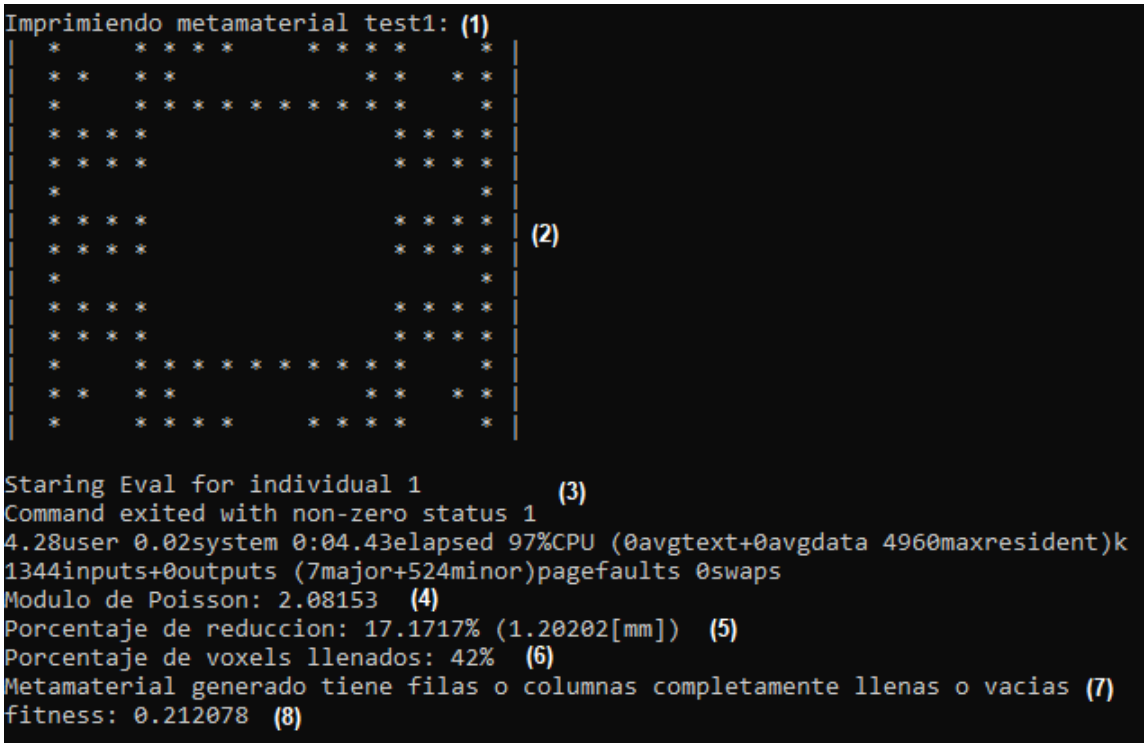


Figura 3.15: Información impresa en pantalla en tiempo real durante la evolución del GA.

3.13. Resumen

Para finalizar, se presenta un esquema que muestra la jerarquía entre las funciones anteriormente descritas y cómo esta se relacionan entre sí.

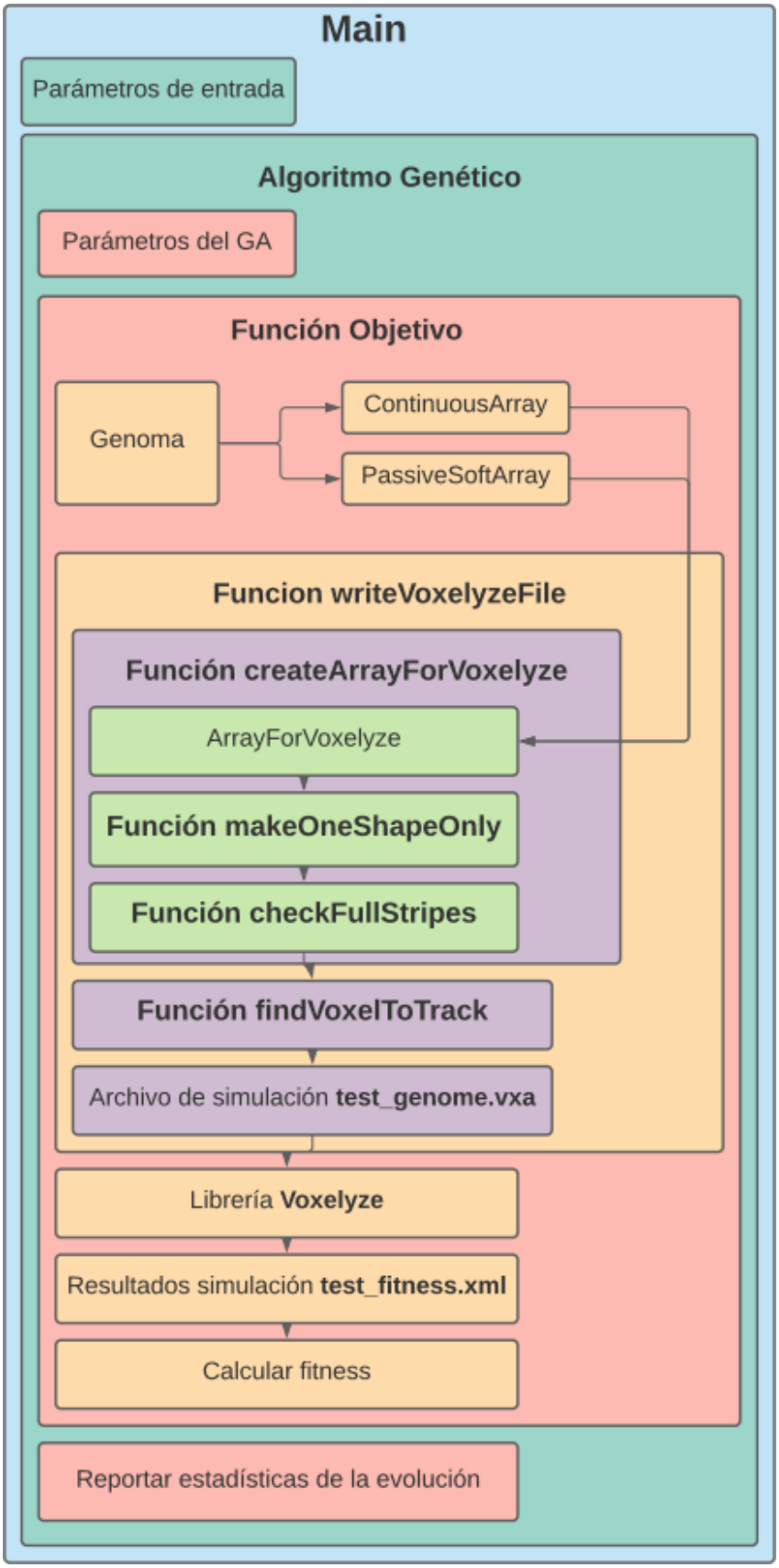


Figura 3.16: Esquema que representa la jerarquía y relación entre las distintas funciones que conforman el código desarrollado.

Capítulo 4

Consideraciones de Diseño y Entorno de Experimentación

4.1. Parámetros Establecidos Para el GA

A continuación se exponen los criterios tomados en cuenta al momento de diseñar el GA:

- Se usa un GA del tipo *Steady-State*, puesto que es el mismo que se usó en el estudio original de *softbots* [10] tomado como base para desarrollar la presente herramienta de diseño. Además, como con este método no se reemplaza la totalidad de la población, el GA puede avanzar más rápido y los mejores individuos se pueden conservar intactos, y así visualizar de mejor manera cómo estos van evolucionando.

- Comúnmente, un GA usa una población de 20 individuos. Sin embargo, para este sistema se decidió subir dicho valor a 50, ya que para las primeras generaciones, la probabilidad de generar individuos discontinuos es considerablemente alta.

- Las probabilidades de mutación y *crossover* se mantienen con valores estándar de 0,01 y 0,8, respectivamente.

- Se decide establecer un criterio de terminación basado en convergencia en lugar de número de generaciones. Esto debido a que, para ciertas evoluciones, un número determinado de generaciones podría ser muy bajo, o bien muy alto, lo que a su vez se traduce, respectivamente, en una curva de aprendizaje incompleta o mantenida en convergencia más tiempo del necesario.

- El factor de convergencia a usar será igual a 0,999, calculado a partir de las últimas 30 generaciones, ya que de esta manera se obtienen curvas de aprendizaje donde el *fitness* promedio alcanza a igualar al *fitness máximo* y mantenerse constante durante una cierta cantidad de generaciones.

En la tabla 4.1 se muestra un resumen de los parámetros del GA implementado.

4.2. Parámetros Establecidos Para la Simulación

Debido a la gran cantidad de parámetros que se manejan dentro de esta herramienta, algunos de estos se tomarán de manera constante para todas las pruebas a realizar. Además, se establecerán ciertos criterios determinados de manera empírica como resultado de observaciones realizadas en los testeos preliminares del sistema. Estos son:

- Se usará una resolución del genoma de 7x7 para los metamateriales bidimensionales, puesto que con dicho valor se pueden generar figuras suficientemente complejas sin perder suficiente estabilidad en la simulación ni tiempo de ejecución del código.
- Para una resolución de 7x7, se podrán simular metamateriales con hasta 2 celdas unitarias por lado, puesto que en caso de agregar una tercera, las simulaciones se vuelven más lentas y difíciles de estabilizar, ralentizando de sobremanera la ejecución del algoritmo genético. En la figura 4.1 se pueden observar los problemas de simulación que surgen para metamateriales demasiado grandes.
- Las dimensiones individuales de cada *voxel* serán de $1[mm]$ por lado.
- Para facilitar la deformación de los metamateriales bidimensionales, y dado que en estas simulaciones es más difícil gatillar efectos de pandeo con respecto a la realidad, estos se extruirán una altura igual a $4[mm]$. O dicho de otro modo, el metamaterial a generar tendrá 4 *voxels* de altura en el eje z.
- A pesar de que el módulo de elasticidad del TPU es de $26[MPa]$ (según ficha técnica adjunta al anexo B), se establece el módulo de elasticidad de los *voxels* con un valor de $3{,}5[GPa]$. Esto porque con órdenes de magnitud inferiores, la deformación del objeto simulado presenta un comportamiento irreal e inestable (al menos para fuerzas de compresión con una magnitud suficientemente alta para que el tiempo de simulación sea de unos cuantos segundos). Este fenómeno se ve ilustrado en la imagen 4.2.
- Para los metamateriales tridimensionales, se disminuye el valor del módulo de elasticidad y se baja la resolución del genoma de 7x7 a 5x5. Estas medidas se aplican con la finalidad de agilizar la ejecución del código y para que el tiempo de simulación sea razonable.

En la tabla 4.2 se resumen los parámetros establecidos para las simulaciones.

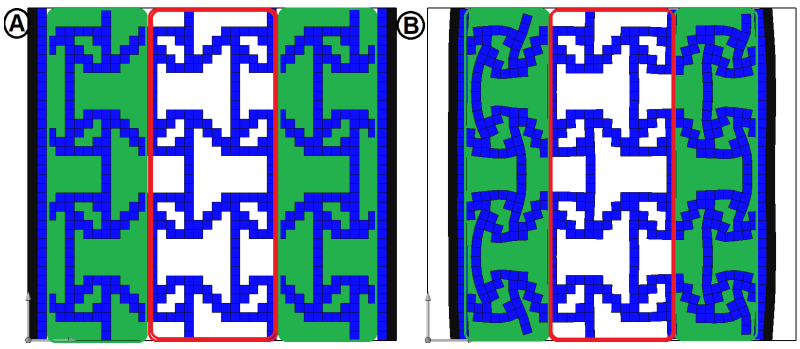


Figura 4.1: Límitación en el tamaño máximo del objetos simulado. (A) Metamaterial de dos celdas unitarias con resolución de genoma de 9x9 al inicio de la simulación. (B) Al comparar las zonas con fondo verde y la zona enmarcada en rojo, se puede notar que la deformación ejercida en los extremos se demora en transmitirse a la parte central del objeto.

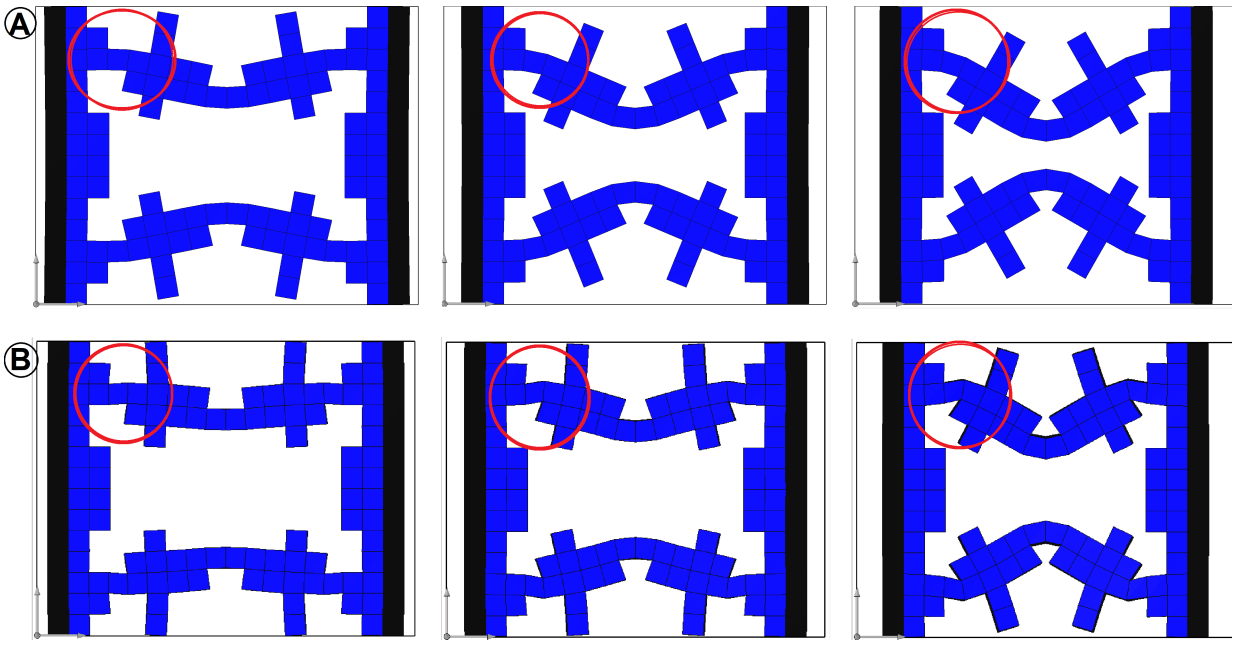


Figura 4.2: Diferencia en la calidad de la simulación según módulo de elasticidad. (A) Módulo de 3,5[GPa]. (B) Módulo de 26[MPa]. En A se observa una mejor transmisión de la deformación entre *voxels* contiguos que en B, logrando una forma más continua. Este efecto puede notarse en las zonas indicadas con un círculo rojo.

Tabla 4.1: Resumen de los parámetros del GA utilizado para evolucionar los resultados finales.

Parámetro	**Valor**
Población	50
p Mutación	0.01
p Crossover	0.8
p Convergencia	0.999
n Convergencia	30

Tabla 4.2: Parámetros de entrada usados para los testeos finales del presente estudio.

	2D n1 7x7	**2D n2 7x7**	**3D n1 5x5**
VoxelSize [m]	0.001	0.001	0.001
Dt	0.95	0.95	0.95
SST [s]	6	5	5
SSS	10000	2000	5000
ForceX [N]	-2500	-25000	-5000
E_meta [MPa]	3500	3500	20000
E_placas [MPa]	300000	300000	300000

4.2.1. Método Para Determinar Parámetros de Simulación

1. En una primera instancia, ingresar valores de manera intuitiva.
2. Ejecutar el código.
3. Dado que se puede ir testeando cada individuo en tiempo real a medida que evoluciona el algoritmo genético, se ejecuta la interfaz gráfica VoxCAD y se abre el simulador para alguno de estos individuos.
4. Ajustar el valor para la fuerza de compresión axial tal que en solo un par de segundos el metamaterial se comprima entre un 10 y un 20 % de reducción de la longitud axial total.
5. Dirigirse a la pestaña *Output* del simulador y ver los parámetros que se indican en la figura 4.3.
6. A partir del dato *step*, se puede obtener un valor inicial para el parámetro SSS. Por ejemplo, un valor de 6000 *steps* sería una buena aproximación para el caso de la figura 3.14 .
7. A partir del dato *rate*, se puede realizar una regla de tres con el dato *step*, y así obtener en dicho caso un SST de 4 segundos.
8. Volver a ejecutar el código e ir ajustando de a poco los valores anteriormente mencionados, procurando que el tiempo que demora el GA en evaluar a cada candidato de metamaterial no se extienda más allá de unos cuantos segundos.

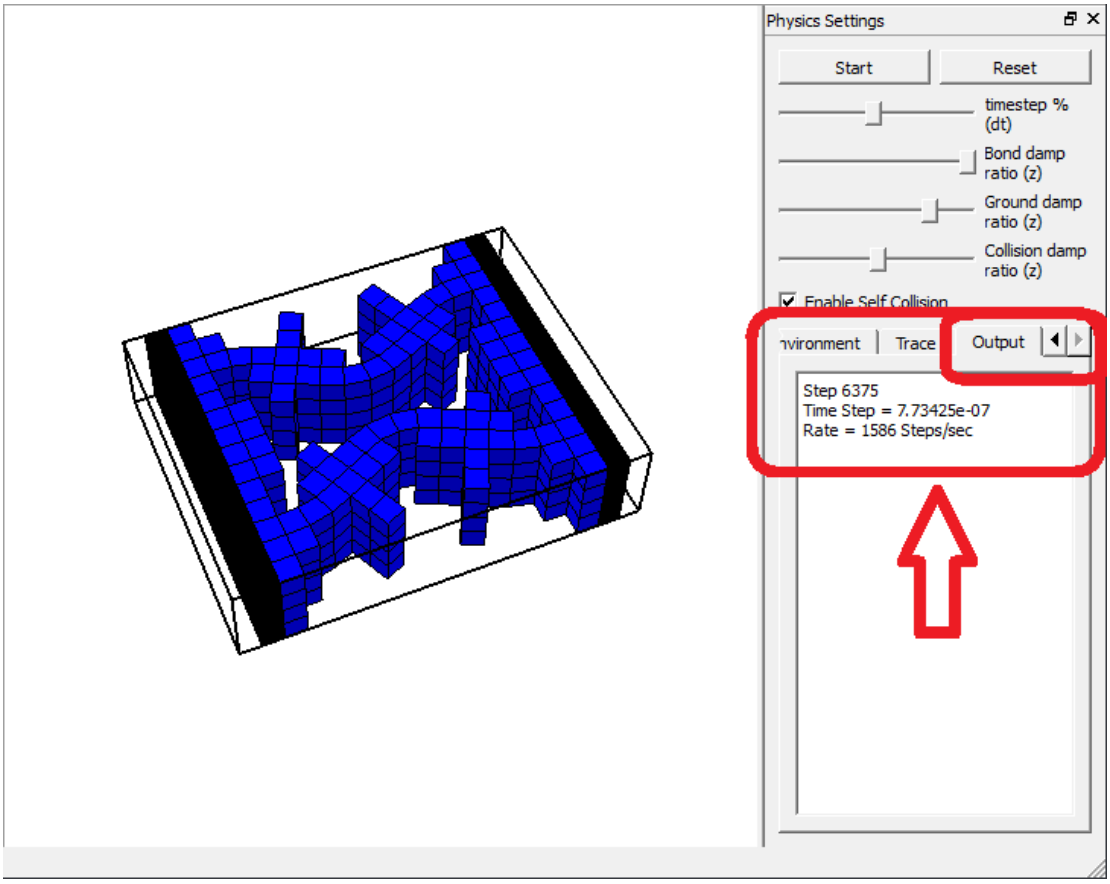


Figura 4.3: Método usado para determinar los parámetros de simulación.

4.3. Consideraciones de Modelado 3D y Manufactura

Al momento de modelar los resultados en CAD, se tomarán los siguientes criterios de diseño:

- Se mantendrán las dimensiones originales del objeto simulado, es decir, *voxels* de $1[mm]$ por lado. Esta medida ayudará a simplificar su elaboración en CAD y posteriormente, al momento de usar *Creality Slicer* para imprimir el metamaterial, será más fácil usar factores de escala para cambiar las dimensiones de este.
- Se usará un radio de curvatura de $0,5[mm]$ para las aristas externas de los *voxels*. Esto para evitar los ángulos rectos que puedan actuar como sumideros de esfuerzos, mejorar las uniones de arista entre *voxels* diagonalmente contiguos y favorecer la estética del metamaterial.
- Para evitar el pandeo en las piezas impresas, estas se extruyen con una altura aproximada a un tercio del largo de la pieza.
- Los diseños se imprimen con un factor de escala de $1,4$ debido a la dificultad para imprimir piezas con detalles finos usando filamento de TPU.
- Solo se imprimirán los metamateriales bidimensionales, ya que los tridimensionales son más complejos de modelar y manufacturar.

4.4. Elaboración del Entorno Para Ensayos de Compresión

Debido a las complicaciones para asistir a laboratorios dentro de la Universidad, que surgen a raíz de la emergencia sanitaria vivida durante el transcurso del presente trabajo de título, se decide montar un entorno de experimentación improvisado con elementos disponibles en el hogar. En dicho ambiente se llevará a cabo la compresión de los metamateriales diseñados y el registro en cámara de su comportamiento. Debido a las limitaciones del entorno fabricado, los resultados y su análisis se realizarán de manera cualitativa.

En la figura 4.4 se muestran y etiquetan los elementos que conforman el entorno improvisado de experimentación. Estos son:

1. Soporte para la iluminación.
2. Lámpara para iluminar la escena.
3. Papel de lija para impedir que el metamaterial se desplace de su sitio.
4. Metamaterial fabricado en impresora 3D con filamento flexible. Sujeto de prueba del experimento.
5. Combo o mazo usado a modo de prensa para comprimir al metamaterial.
6. Soporte para el papel de lija y sujeto de prueba.
7. Cámara de celular que registrará en video el comportamiento del metamaterial al ser comprimido.
8. Sillín cuya altura se puede ajustar para que el lente de la cámara quede cara a cara respecto al metamaterial.
9. Soporte de celular.

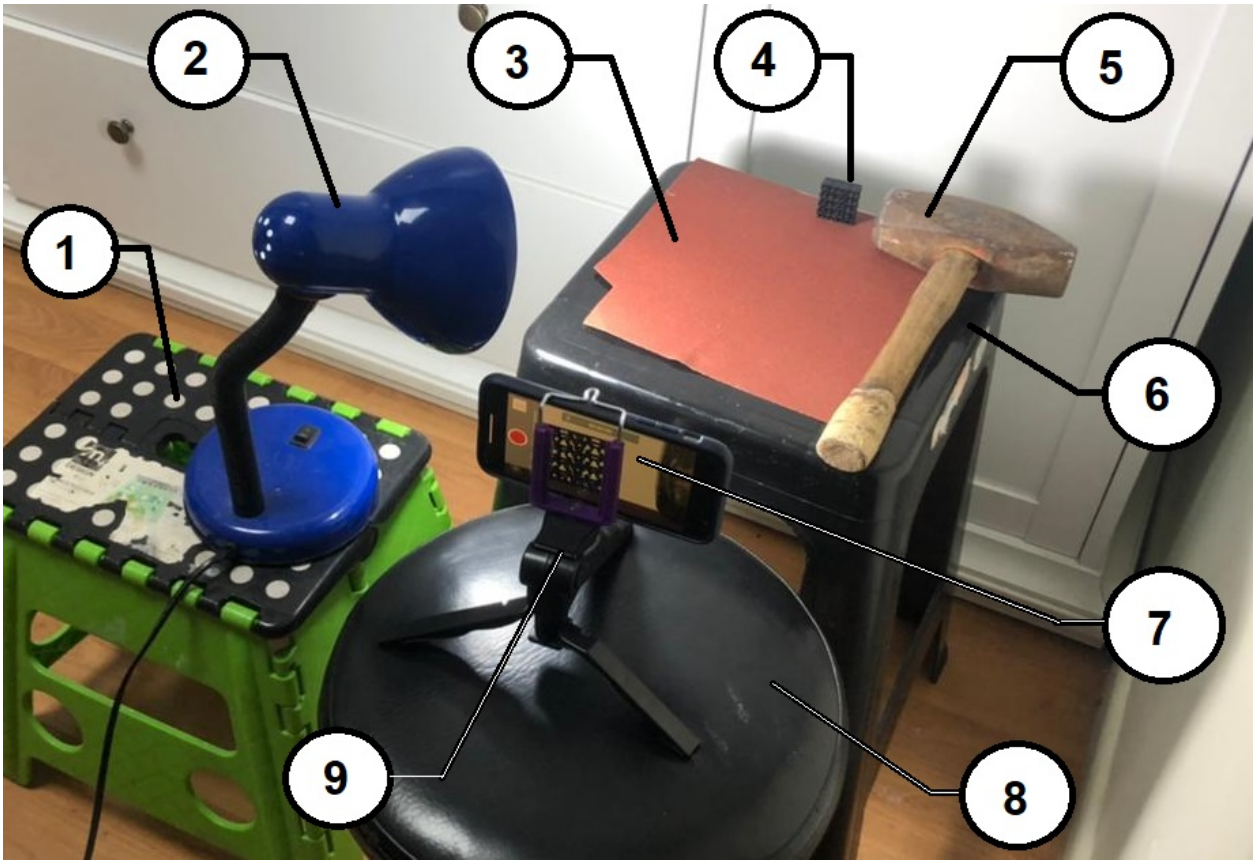


Figura 4.4: Ambiente de laboratorio improvisado en el hogar para llevar a cabo los ensayos de compresión de los metamateriales fabricados.

Capítulo 5

Resultados

En las siguientes secciones se recopilan los diseños de metamateriales auxéticos generados por el sistema desarrollado a lo largo del presente trabajo de título. Los resultados se clasifican de la siguiente manera:

1. Simetrías con 1 celda unitaria.
2. Simetrías con 2 celdas unitarias.
3. Seeding.
4. Metamateriales alostéricos.
5. Geometrías 3D.
6. Resultados experimentales
7. Evolución y curva de aprendizaje.

Para cada una de las pruebas realizadas dentro de estas clasificaciones, se mostrarán las 3 evoluciones llevadas a cabo, indicando cual de las tres corresponde al mejor resultado obtenido, para posteriormente exponer comparativamente sus propiedades mecánicas medidas en las simulaciones.

5.1. Simetrías Para Metamateriales Bidimensionales con 1 Celda Unitaria

En las figuras 5.1 y 5.2 se presenta una recopilación de los resultados finales para cada una de las simulaciones llevadas a cabo, mostrando el estado inicial y final de las mismas. Los *fitness* se ordenan de mayor a menor, clasificándolos respectivamente con las letras A, B y C. Las simulaciones con fondo anaranjado se descartan por ser inestables. Por su parte, aquellas con fondo amarillo son seleccionadas por ser las mejores dentro de los resultados factibles.

Para el conjunto de los resultados factibles, se elabora la tabla 5.1, donde se muestran las propiedades mecánicas derivadas de las mediciones tomadas en la simulación.

En las figuras 5.3, 5.4 y 5.5 se presentan gráficos de barras comparativos a partir de los resultados de la tabla 5.1. En el eje horizontal se ordenan los tipos de simetrías y en el vertical se muestran las propiedades mecánicas medidas, las que corresponden respectivamente a: *fitness*, factor de reducción de área transversal y factor de compresión axial; y Coeficiente de Poisson.

Tabla 5.1: Resultados extraídos de las simulaciones para cada tipo de simetría para metamateriales bidimensionales de una celda unitaria y resolución 7x7.

Symmetry Type	**Fitness**	**Coef. Pisson**	**Factor de Reducción de Área Transversal**	**Factor de Compresión Axial**
0	0,6172	-1,77	0,66	0,21
1	0,5976	-1,93	0,33	0,17
2	-	-	-	-
3	0,6053	-1,83	0,35	0,19
4	0,5960	-1,71	0,32	0,19
5	0,5679	-1,28	0,24	0,19
6	-	-	-	-

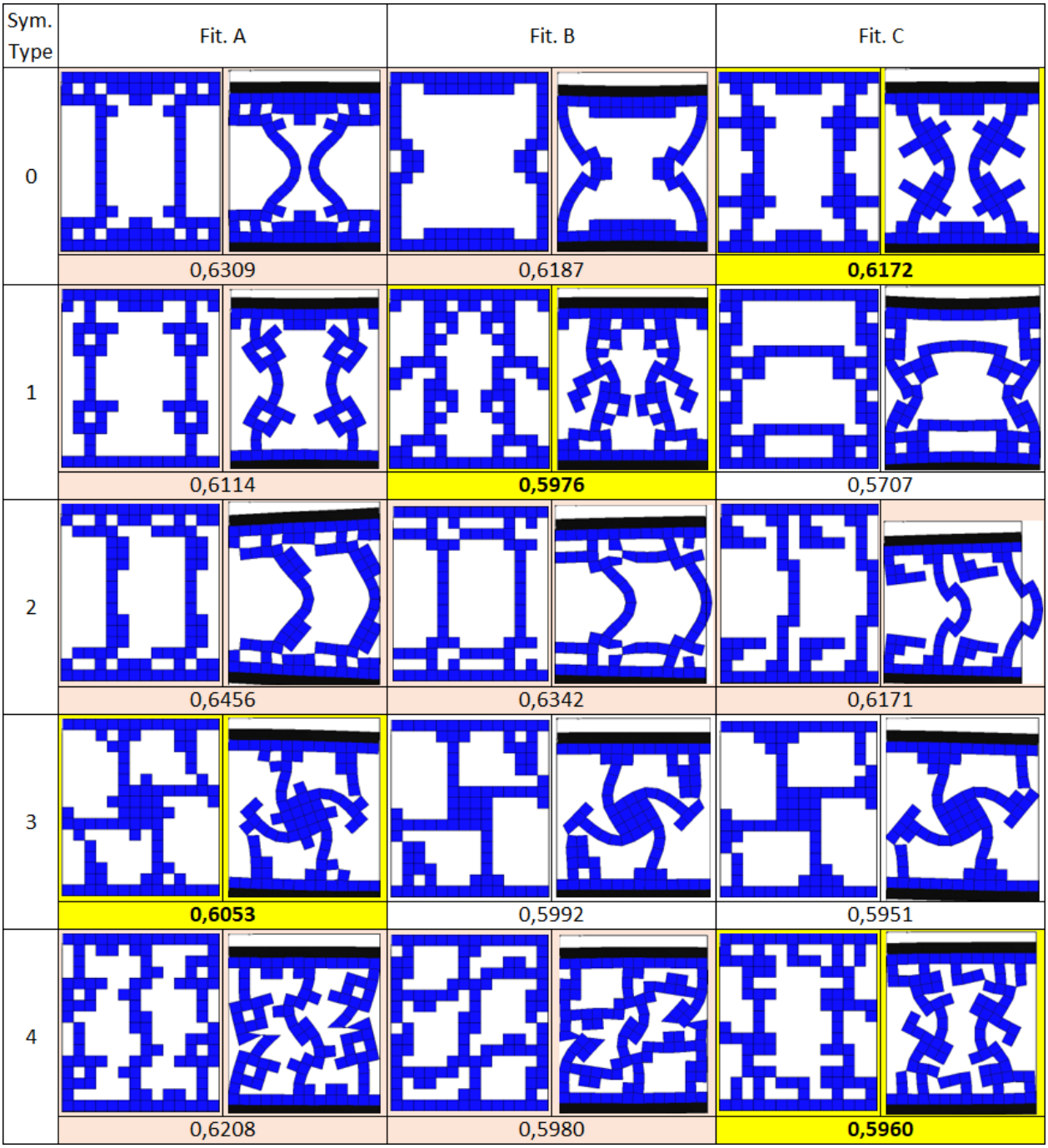


Figura 5.1: Simulaciones según tipo de simetría para metamateriales bidimensionales de 1 cela unitaria y resolución 7x7.

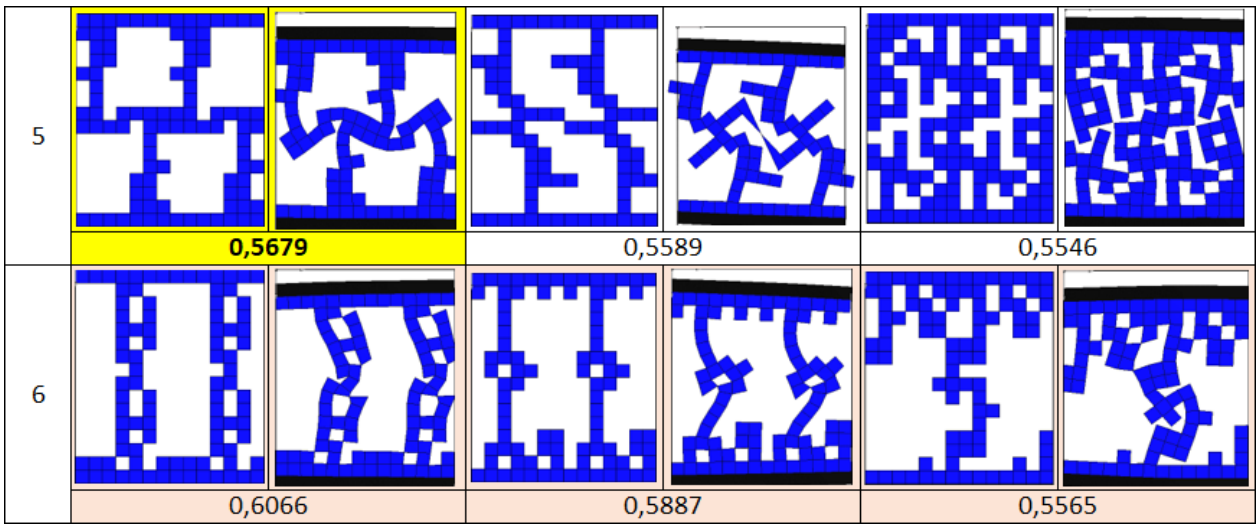


Figura 5.2: Continuación figura 5.1

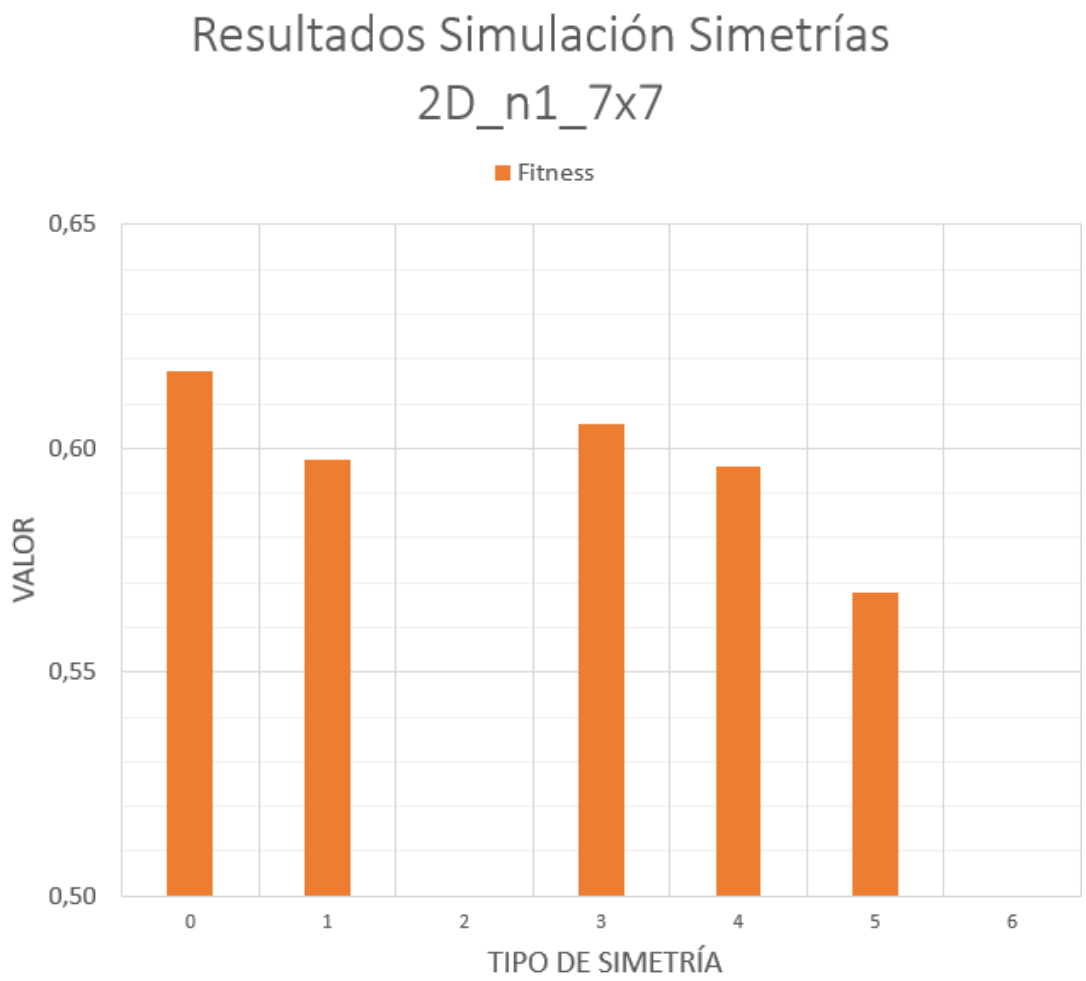


Figura 5.3: Gráfico de barras comparativo de los valores de *fitness* según tipo de simetría para metamateriales bidimensionales de 1 celda unitaria y resolución 7x7.

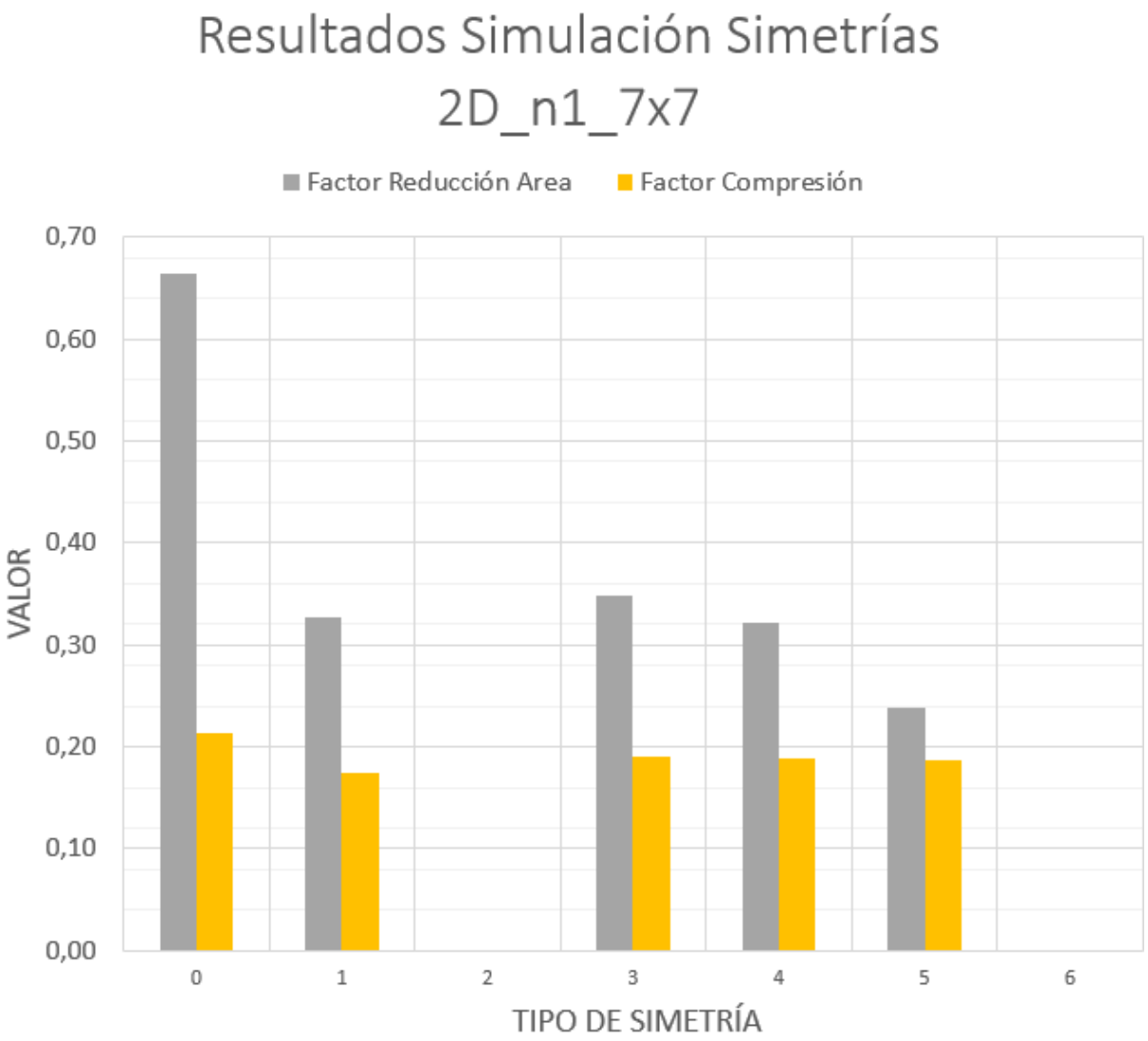


Figura 5.4: Gráfico de barras comparativo de los factores de reducción según tipo de simetría para metamateriales bidimensionales de 1 celda unitaria y resolución 7x7.

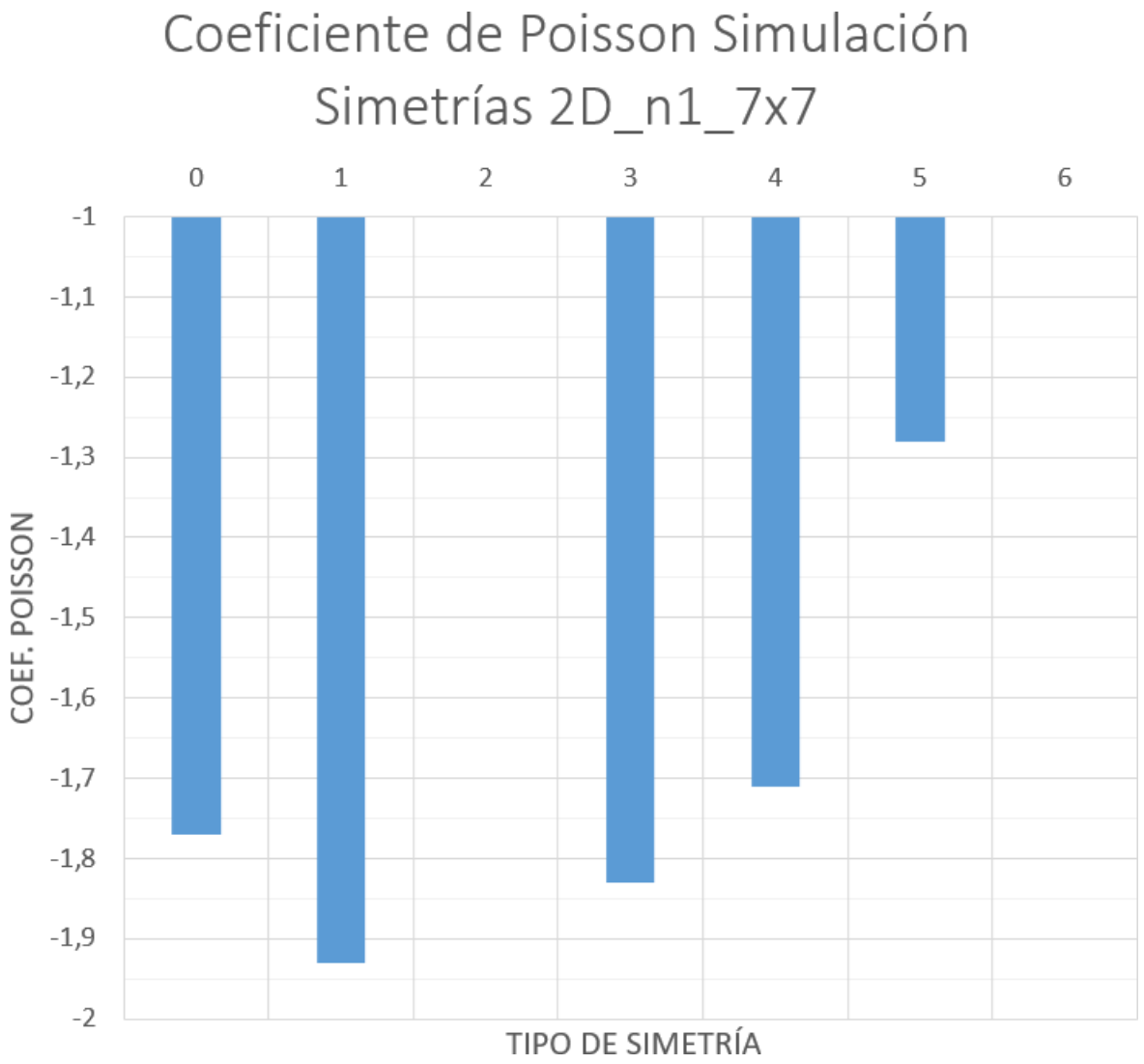


Figura 5.5: Gráfico de barras comparativo de los coeficientes de Poisson según tipo de simetría para metamateriales bidimensionales con 1 celda unitaria y resolución 7x7.

5.2. Simetrías Para Metamateriales Bidimensionales con 2 Celdas Unitarias

En las figuras 5.6 y 5.7 se presenta una recopilación de los resultados finales para cada una de las simulaciones llevadas a cabo, siguiendo la misma lógica que los resultados presentados anteriormente.

Los resultados de las simulaciones para los mejores diseños se recopilan en la tabla 5.2.

Por su parte, en las figuras 5.8, 5.9 y 5.10 se presentan gráficos de barra comparativos para el *fitness*, factores de reducción y coeficiente de Poisson, respectivamente.

Tabla 5.2: Resultados medidos en las simulaciones según tipo de simetría para metamateriales bidimensionales de dos celdas unitarias y resolución 7x7.

Symmetry Type	**Fitness**	**Coef. Pisson**	**Factor de Reducción de Área Transversal**	**Factor de Compresión Axial**
0	0,5499	-0,66	0,13	0,13
1	0,5189	-0,33	0,04	0,10
2	-	-	-	-
3	0,5241	-0,37	0,06	0,12
4	0,5107	-0,17	0,02	0,12
5	0,5312	-0,50	0,06	0,11
6	-	-	-	-

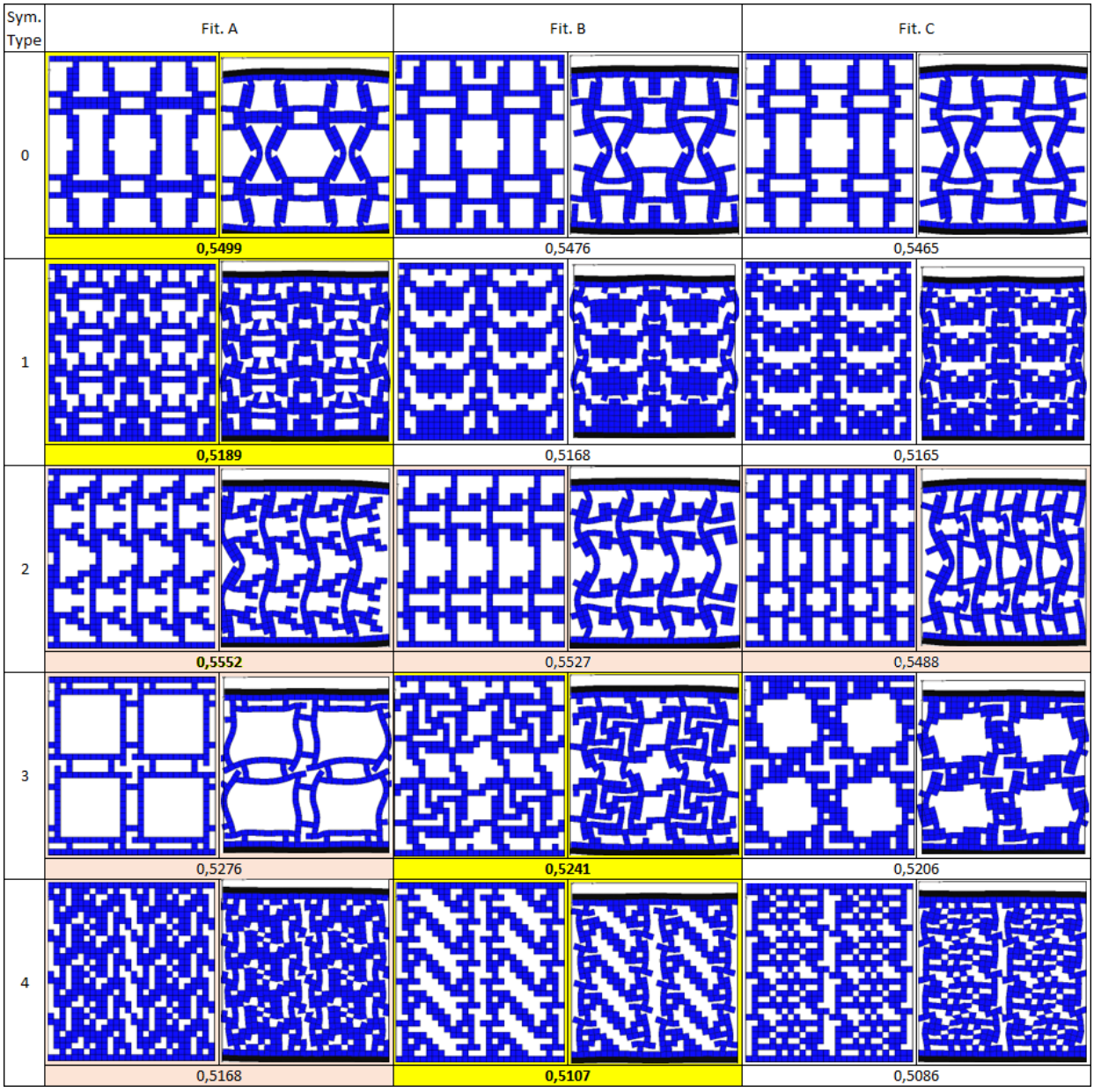


Figura 5.6: Simulaciones según tipo de simetría para metamateriales bidimensionales de 2 celas unitarias y resolución 7x7.

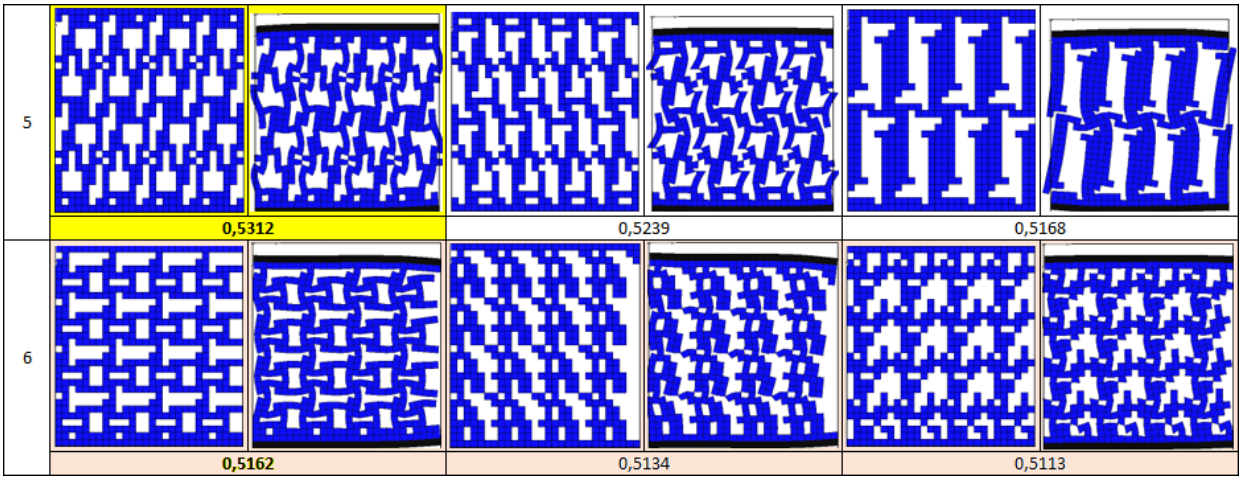


Figura 5.7: Continuación figura 5.1.

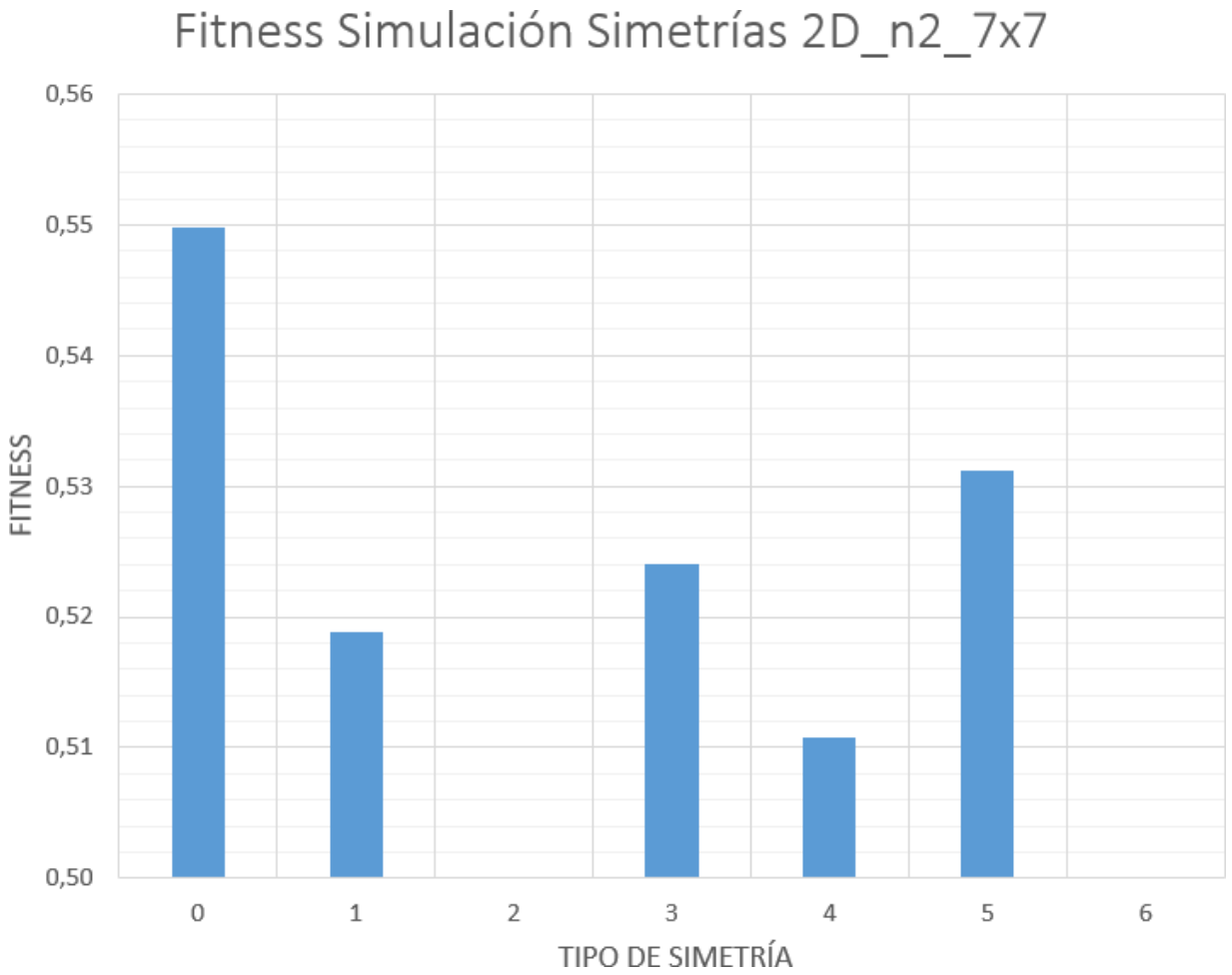


Figura 5.8: Gráfico de barras comparativo de los *fitness* según tipo de simetría para metamateriales bidimensionales de dos celdas unitarias y resolución 7x7.

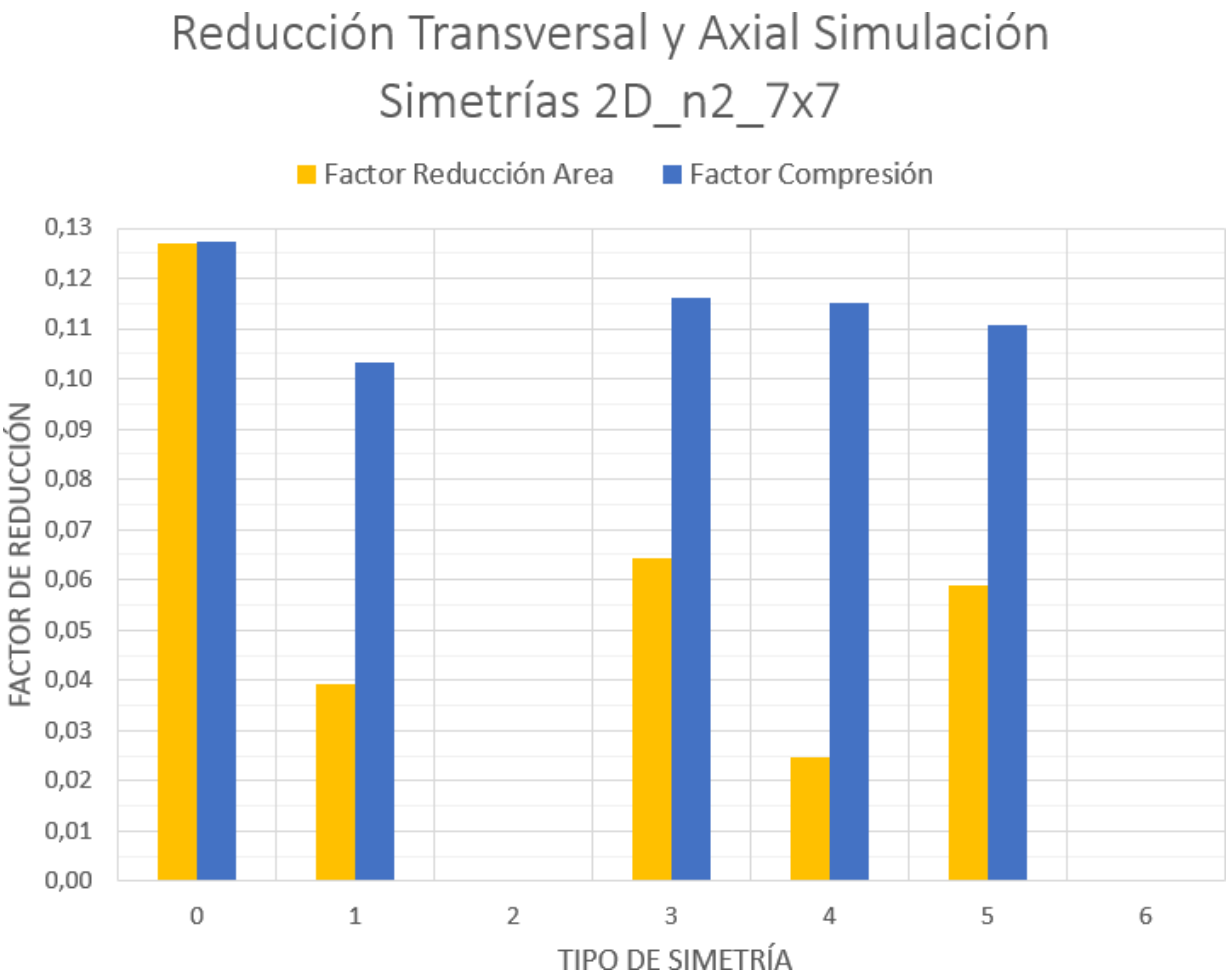


Figura 5.9: Gráfico de barras comparativo de los factores de reducción según tipo de simetría para metamateriales bidimensionales de dos celdas unitarias y resolución 7x7.

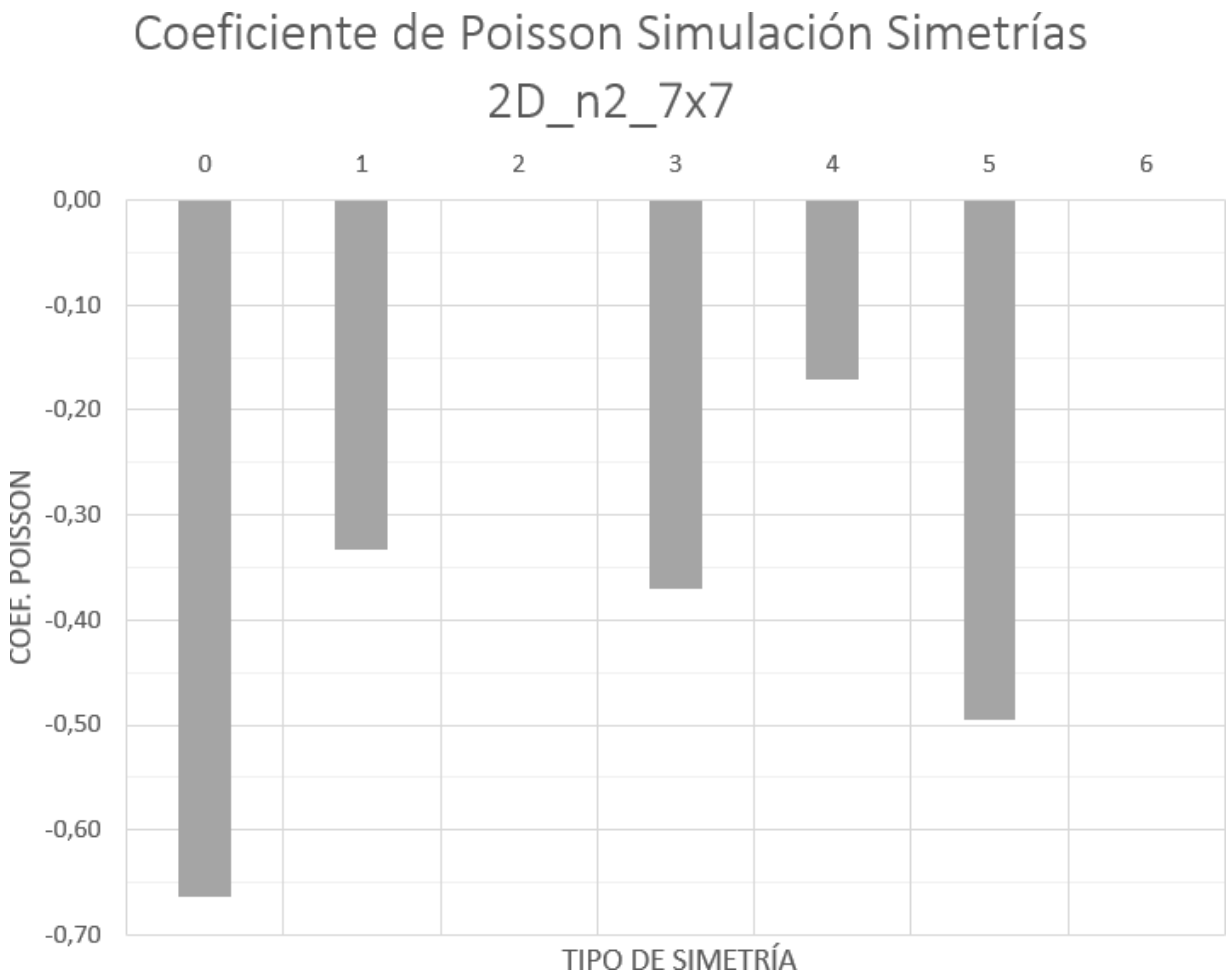


Figura 5.10: Gráfico de barras comparativo para los coeficientes de Poisson según tipo de simetría para metamateriales bidimensionales de dos celdas unitarias y resolución 7x7.

5.3. *Seeding*

En las figuras 5.11, 5.12 y 5.13 se presenta una recopilación de las simulaciones implementando *seeding* en base a 5 semillas distintas. Para los tres testeos realizados por cada semilla se fue variando el parámetro `pbias`, el cual determina el factor de influencia que tendrá la semilla sobre los metamateriales generados. De igual manera que en los testeos anteriores, se destaca con fondo amarillo al mejor resultado de las tres evoluciones realizadas. Por último, cabe mencionar que, para los parámetros de simulación establecidos en un principio, las semillas 4 y 5 no alcanzaron a mostrar un comportamiento auxético, por lo que sus valores de *fitness* no serán considerados.

En primer lugar, se estudia la influencia que tiene la variable `pbias` sobre el tipo de geometría diseñada, para lo cual se consideran los siguientes tres aspectos:

1. Valor de *fitness* logrado en función de la variable `pbias`.
2. Porcentaje de similaridad[1]entre la semilla y el resultado final de la evolución en función de la variable `pbias`.
3. Núnmero de generaciones para cumplir el criterio de convergencia en función de la variable `pbias`.

En la tabla 5.3 se recopilan los datos asociados a estos tres aspectos de estudio.

En las figuras 5.14, 5.15 y 5.16 se grafican, respectivamente, las variaciones en los fitness, similaridad y número de generaciones según el parámetro `pbias`, para cada una de las semillas evolucionadas.

Tabla 5.3: Variación del *fitness*, similaridad y número de generación de los metamateriales generados por *seeding* según el factor `pbias`, para las cinco semillas estudiadas.

	Fitness			Similaridad			N° Generaciones		
seed	p30	p40	p50	p30	p40	p50	p30	p40	p50
1	0,5531	0,5497	0,5224	0,73	0,80	0,92	101	106	36
2	0,5343	0,5508	0,5402	0,65	0,67	0,67	65	82	150
3	0,5402	0,5343	0,5384	0,90	0,96	0,94	66	48	40
4	0,5408	0,5297	0,5372	0,76	0,86	0,90	118	71	66
5	0,5388	0,5308	0,5569	0,80	0,76	0,71	58	76	161

[1]Para calcular este factor, se toma el primer cuadrante de 7x7 de ambos metamateriales y se cuentan cuantos *voxels* tienen en común (Tanto los elementos blandos como los vacíos) para finalmente dividir este valor entre la cantidad total de elementos del genoma (i.e: 49).

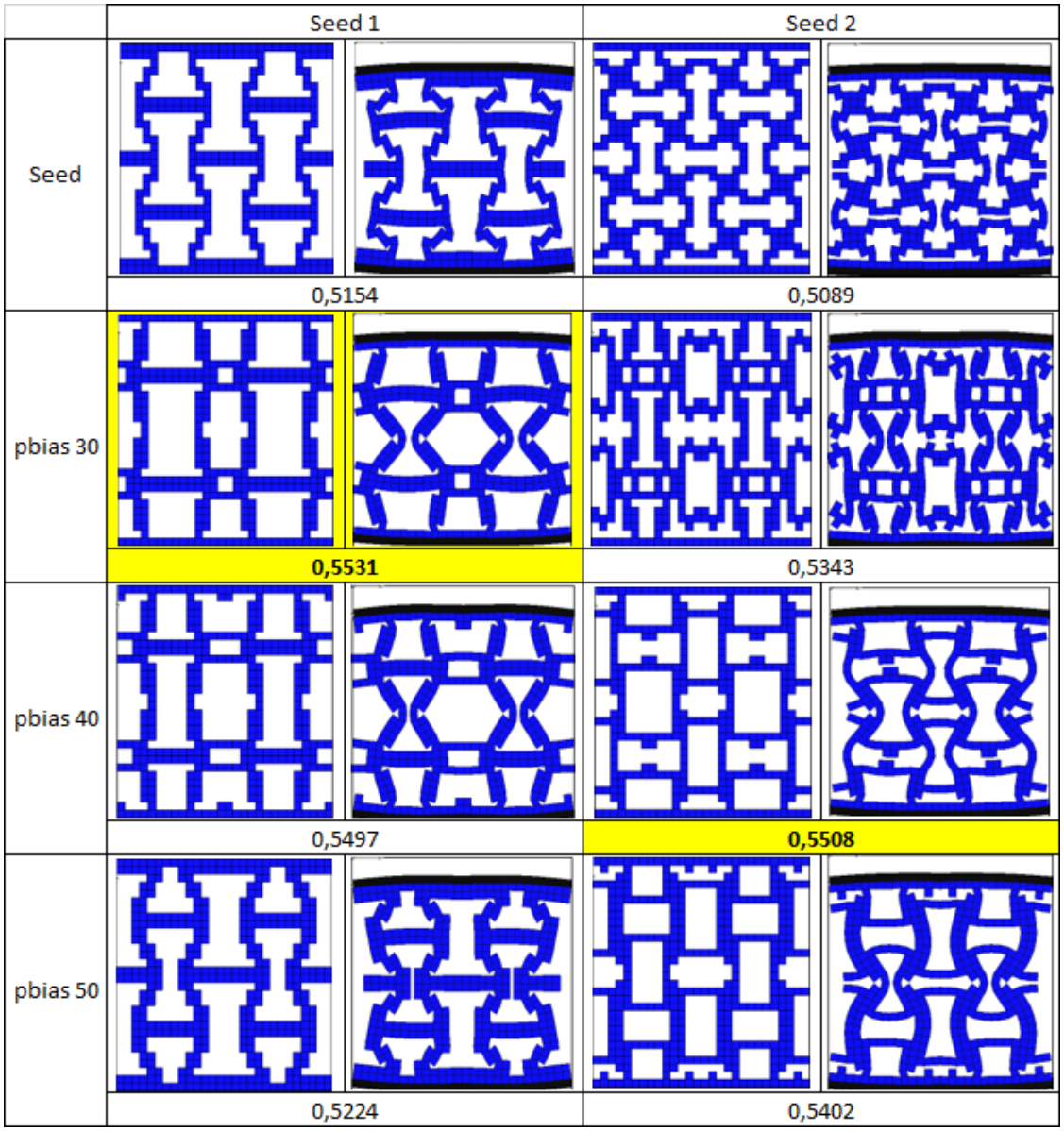


Figura 5.11: Recopilación de los resultados de las simulaciones implementando *seeding*.

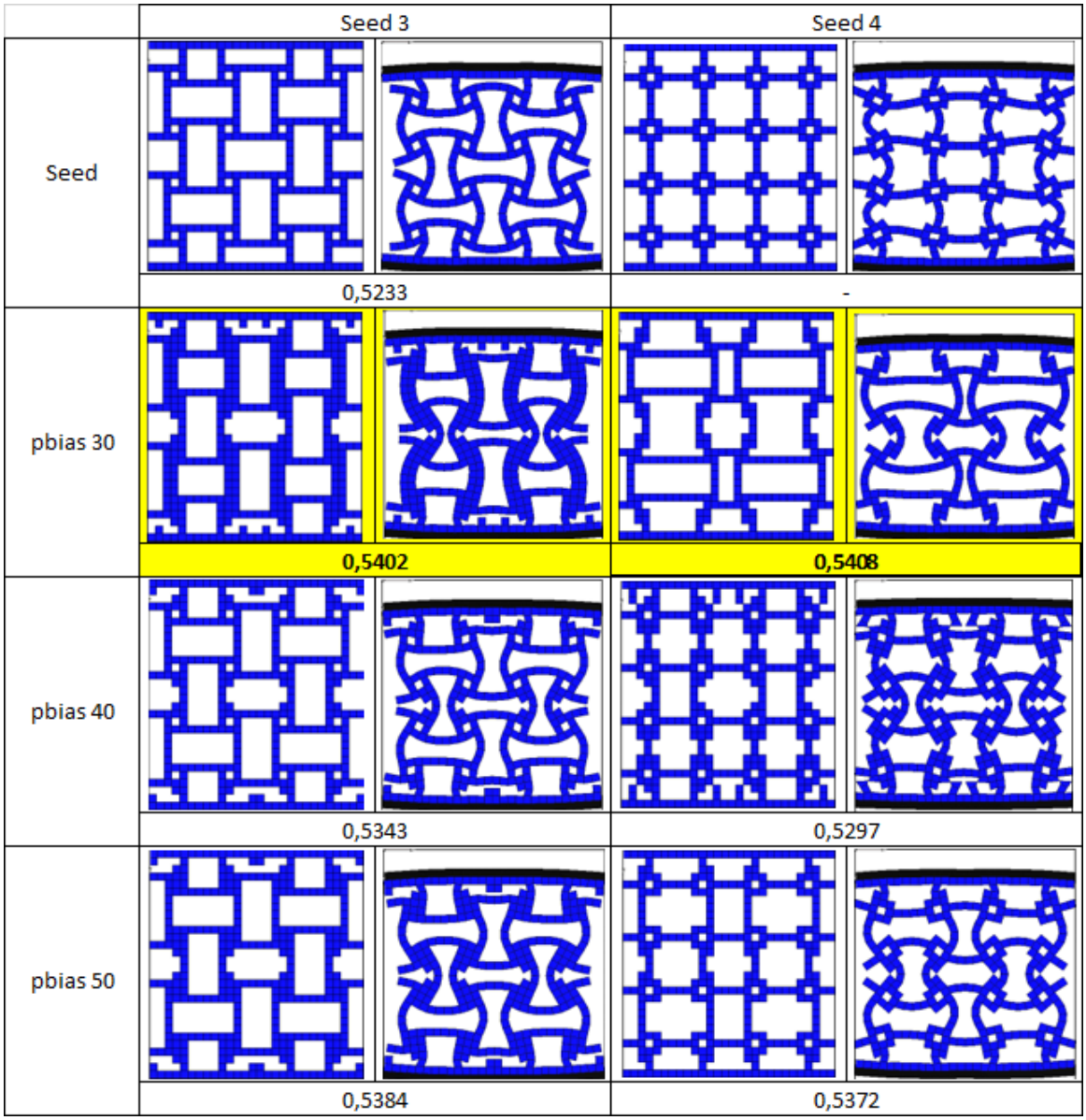


Figura 5.12: Continuación figura 5.11.

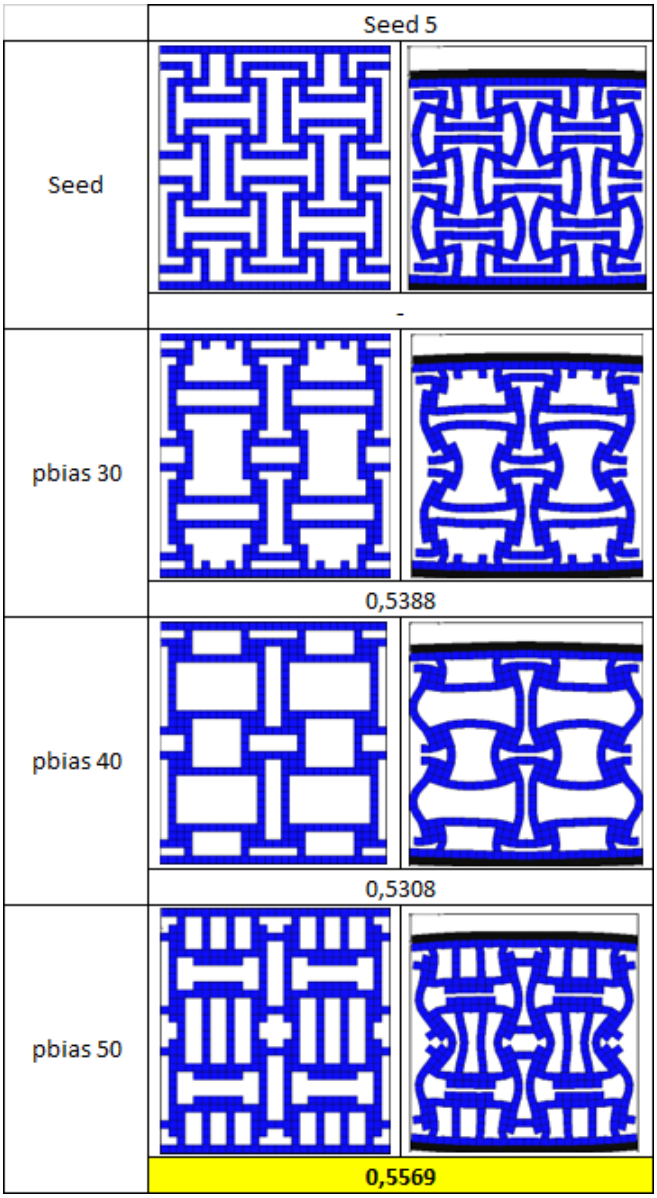


Figura 5.13: Recopilación de los resultados de las simulaciones implementando *seeding*.

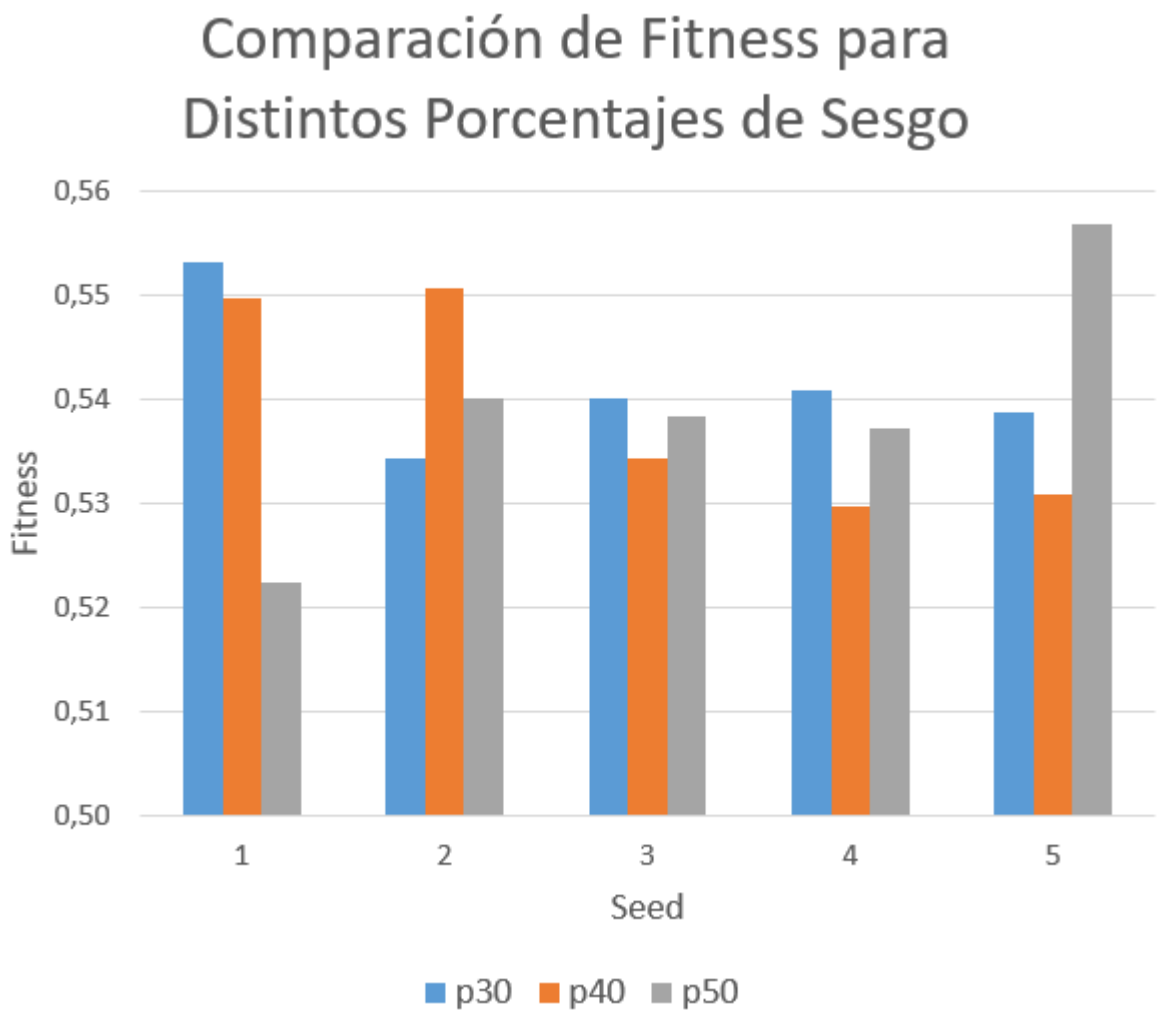


Figura 5.14: Gráfico de barras comparativo para la relación entre el *fitness* y el factor `pbias` para las distintas semillas.

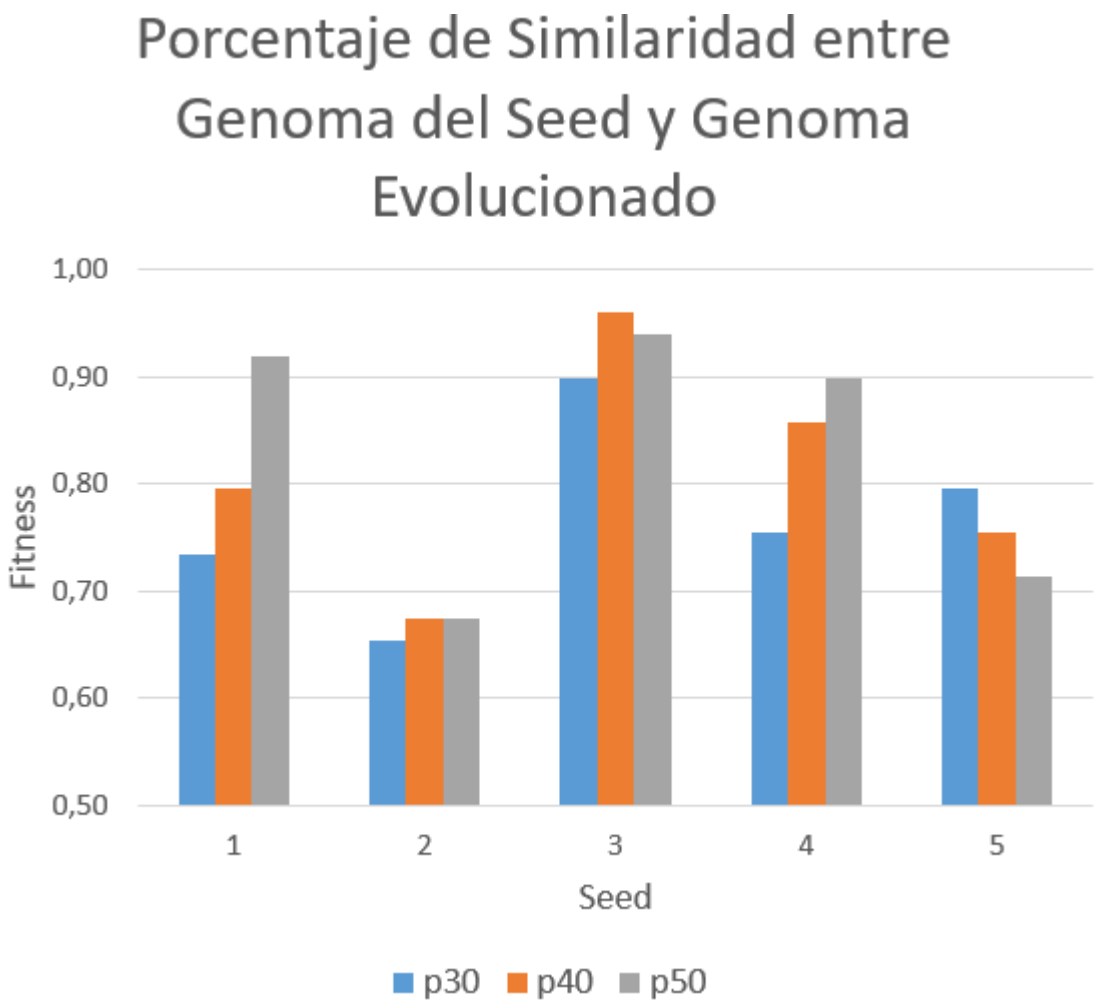


Figura 5.15: Gráfico de barras comparativo para el grado de similaridad entre el genoma de la semilla y el evolucionado, según el factor `pbias`, para las distintas semillas.

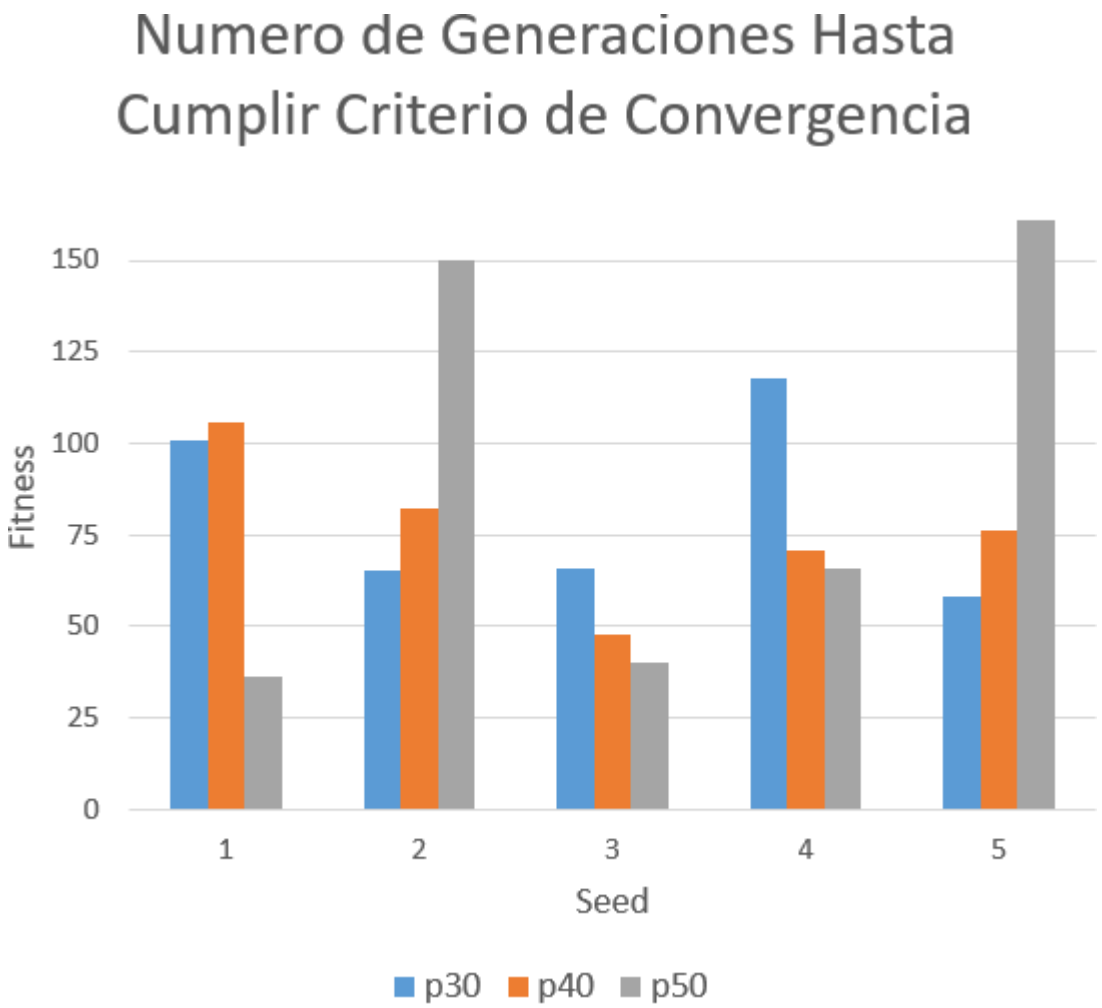


Figura 5.16: Gráfico de barras comparativo para el número de generaciones hasta cumplir el criterio de convergencia, según el factor `pbias`, para las distintas semillas.

Ahora bien, con respecto a los resultados de las simulaciones como tal, estos se recopilan en la tabla 5.4. Para cada tipo de resultado, se presenta el de la semilla junto al de su evolución. Además, para el coeficiente de Poisson y el área de reducción, se presentan los resultados del estudio a partir del cual se obtuvieron las semillas [6] (a excepción de la número 4). Por último, se añade una fila extra para indicar el resultado de una evolución sin *seeding*, importada directamente de la subsección anterior, correspondiente a un metamaterial bidimensional de dos celdas unitarias con simetría tipo 0.

En las figuras 5.17, 5.18, 5.19 y 5.20, se presentan gráficos de barra comparativos para el *fitness*, factor de reducción de área transversal, factor de compresión axial y coeficiente de Poisson, respectivamente.

Tabla 5.4: Resultados de la simulaciones implementando *seeding*, para cinco semillas distintas. Además, se considera el caso en que no se aplica *seeding*.

	Fitness		Coef. Poisson			Reducción Area			Compresión	
Seed N°	Seed	Evolv.	Seed	Evolv.	Estudio	Seed	Evolv.	Estudio	Seed	Evolv.
1	0,5154	0,5531	-0,22	-0,74	-1,68	0,03	0,12	0,32	0,13	0,12
2	0,5089	0,5508	-0,13	-0,71	-0,81	0,03	0,12	0,24	0,12	0,12
3	0,5233	0,5402	-0,32	-0,71	-0,97	0,06	0,10	0,26	0,13	0,12
4	-	0,5408	-	-0,56	-	-	0,10	-	-	0,13
5	-	0,5569	-	-0,84	-0,90	-	0,12	0,20	-	0,11
No Seed	0,5499		-0,66			0,09			0,13	

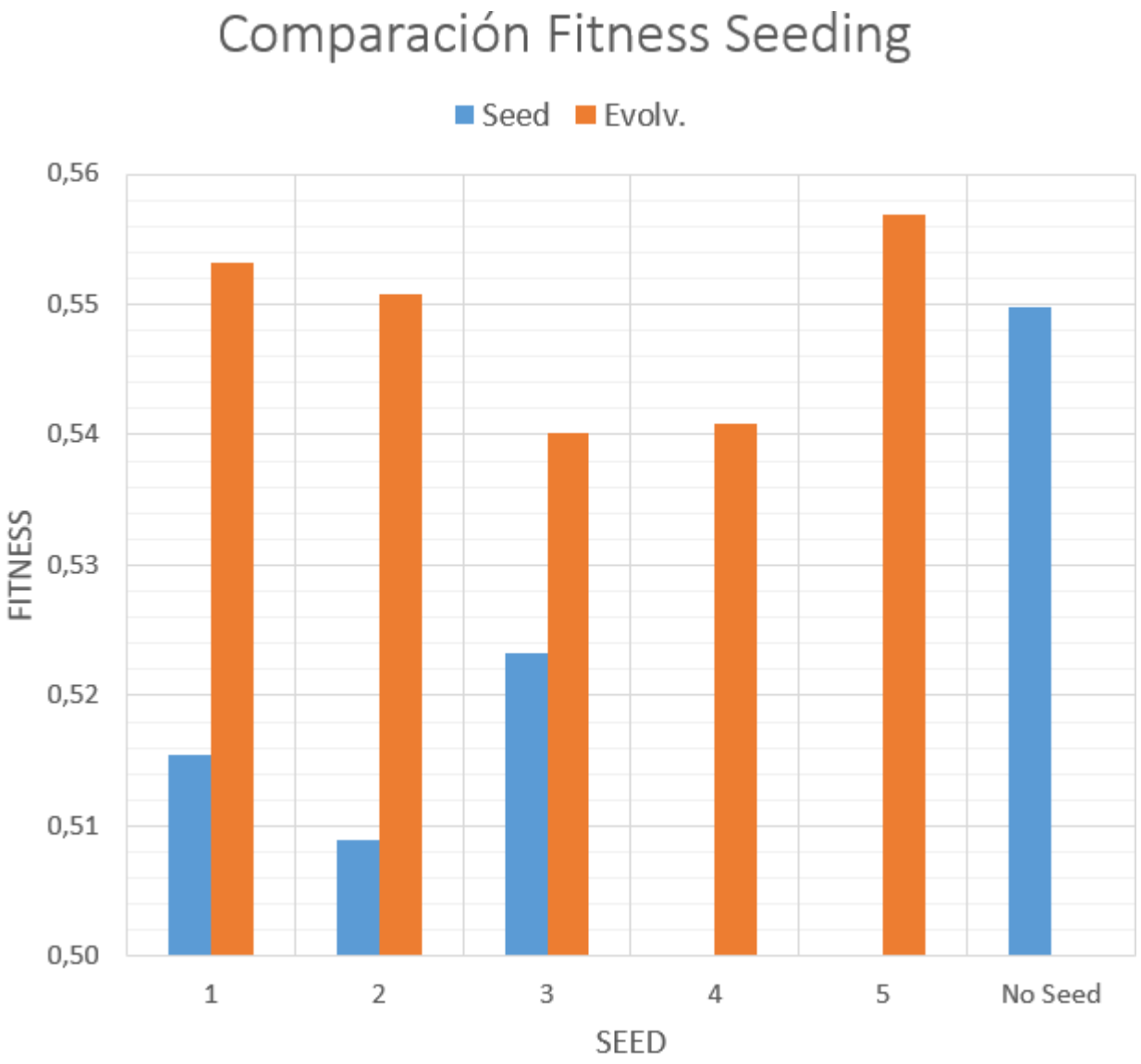


Figura 5.17: Gráfico de barras comparativo para el *fitness* de las semillas y sus evoluciones.

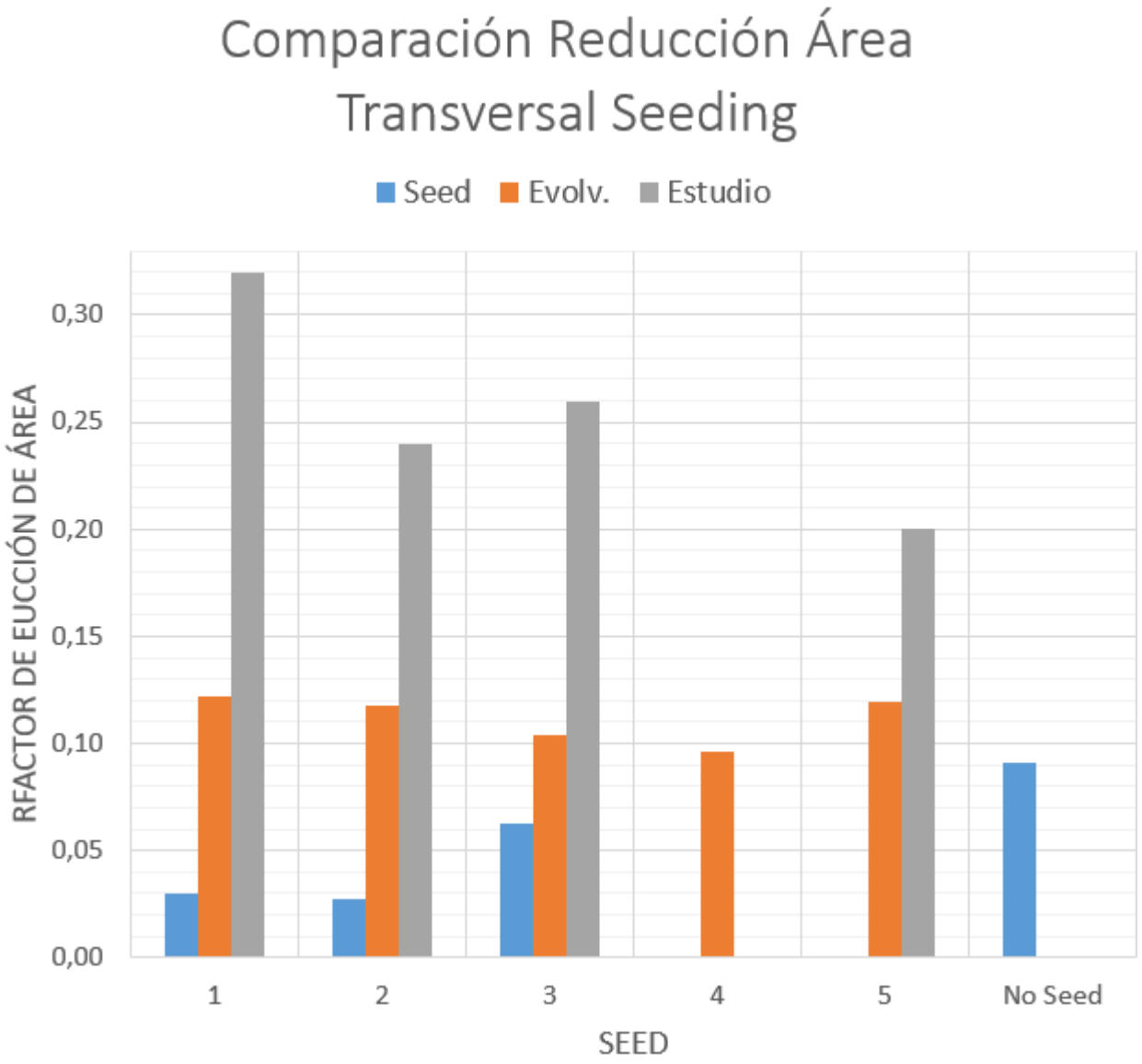


Figura 5.18: Gráfico de barras comparativo para el factor de reducción de área transversal de las semillas y sus evoluciones.

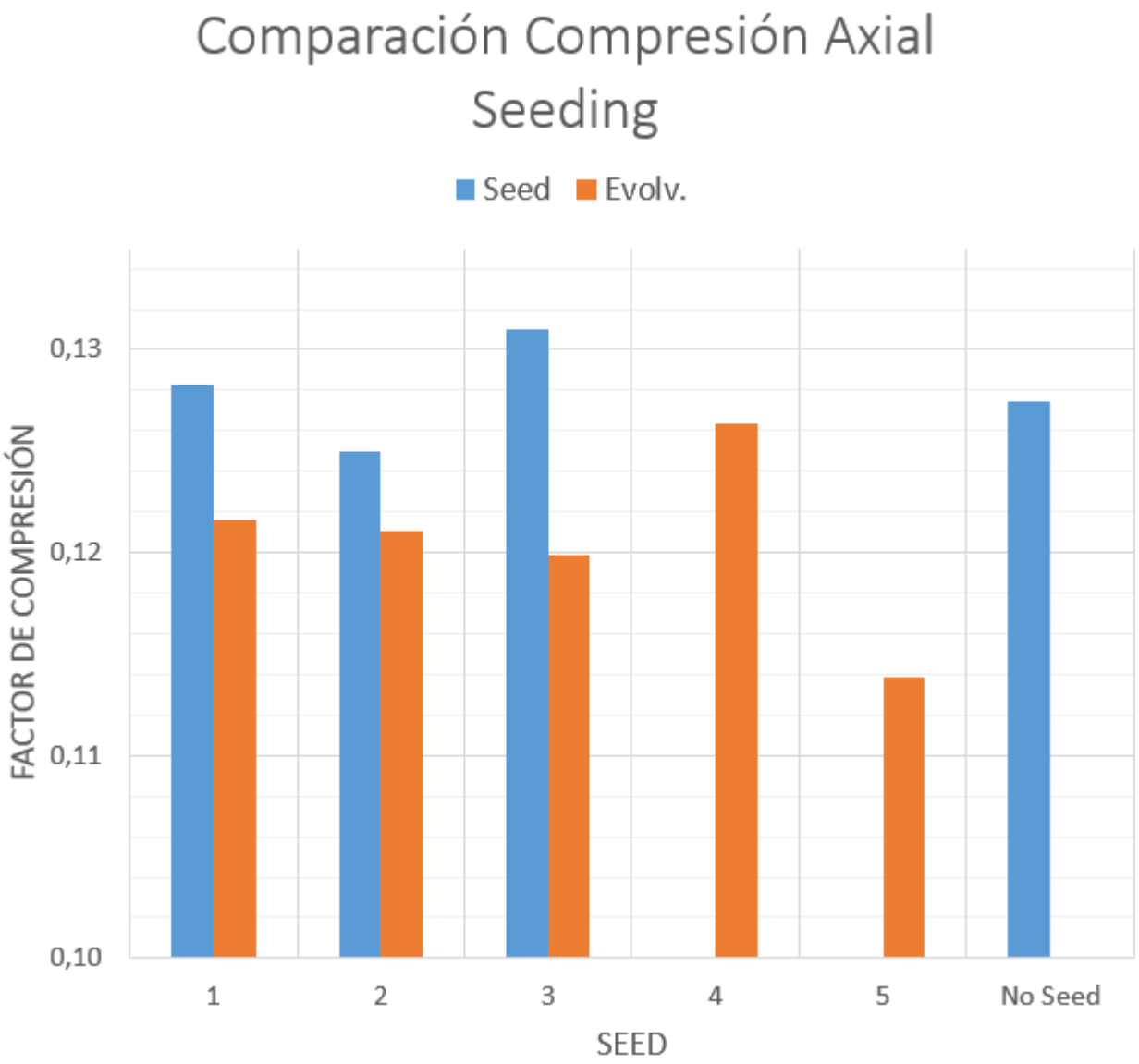


Figura 5.19: Gráfico de barras comparativo para el factor de compresión axial de las semillas y sus evoluciones.

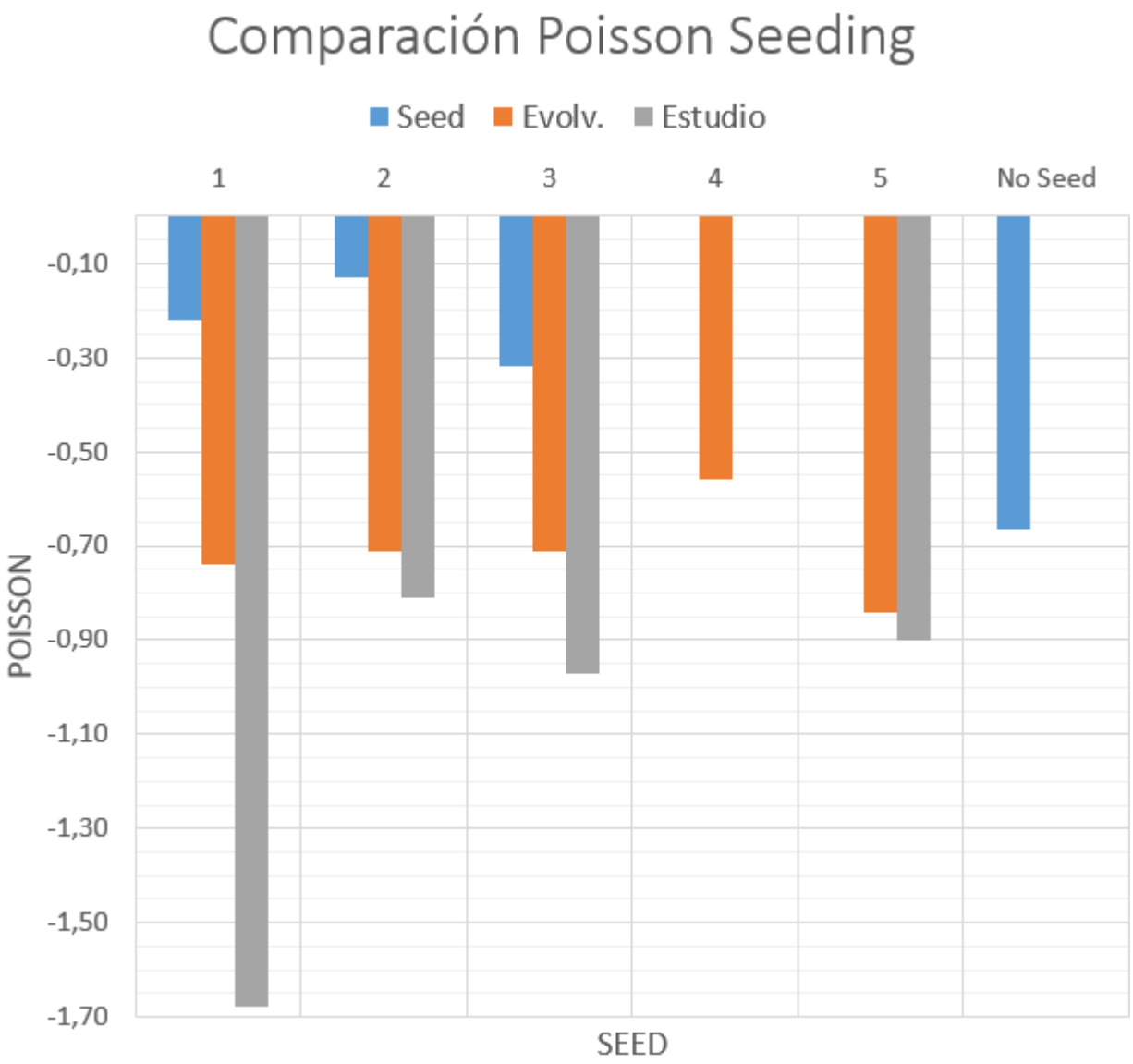


Figura 5.20: Gráfico de barras comparativo para el coeficiente de Poisson de las semillas y sus evoluciones.

5.4. Simulación Alostérica

Dado que el enfoque principal del presente estudio es la auxeticidad, solo se presentan dos casos de evolución alostérica. El primero corresponde a un metamaterial bidimensional de una celda unitaria, con resolución de genoma 7x7 y simetría tipo 0. El segundo caso es idéntico al anterior pero para un metamaterial de 2 celdas unitarias.

En la imagen 5.21 se pueden observar las simulaciones alostéricas llevadas a cabo para ambos casos, mientras que en la tabla 5.5 se recopilan los resultados medidos tras la simulación.

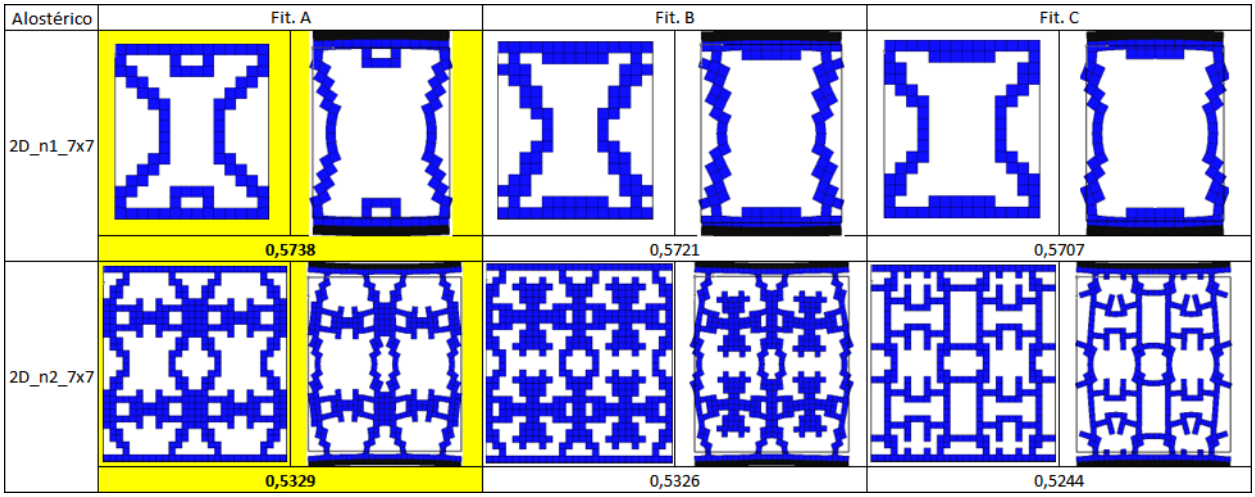


Figura 5.21: Simulaciones alostéricas para metamateriales bidimensionales con simetría tipo 0, para 1 y 2 celdas unitarias.

Tabla 5.5: Resultados de las simulaciones alostéricas.

Alostérico	**Coef. Poisson**	**Factor de Aumento del Área Transversal**	**Factor de Tracción Axial**
2D_n1_7x7	-1,47	0,45	0,09
2D_n2_7x7	-0,52	0,2	0,13

5.5. Geometrías 3D

En la figura 5.22 se observan los tres tests llevados a cabo para la primera representación de metamateriales tridimensionales. En primer lugar se muestra el plano XZ transversal al metamaterial, el cual corresponde al plano donde se exhibirá una mayor respuesta auxética. En segundo lugar, se muestra una vista isométrica del metamaterial diseñado y cómo este responde al momento de comprimirse. Para este caso, se decide orientar el eje de compresión de manera horizontal, ya que la vista isométrica que ofrece la interfaz gráfica de VoxCAD se aprecia mejor de esta manera. Por otro lado, no se identificarán con fondo rosa y amarillo aquellos metamateriales inestables y factibles respectivamente, ya que, a diferencia de los metamateriales bidimensionales, en este caso no es del todo evidente discernir las estructuras inestables de las estables.

Del mismo modo, en la figura 5.25 se observan los 3 tests realizados para la segunda representación de metamateriales tridimensionales. Estos se presentan bajo la misma lógica de la representación anterior.

Por último, en la tabla 5.6 se recopilan los resultados de las simulaciones llevadas a cabo para los mejores diseños de geometrías tridimensionales.

Tabla 5.6: Propiedades de las mejores estructuras tridimensionales obtenidas para ambas representaciones.

Representación	**Fitness**	**Coef. Poisson**	**Factor de Reducción de Área Transversal**	**Factor de Compresión Axial**
3D_1	0,5588	-2,57	0,23	0,09
3D_2	0,6380	-2,49	0,43	0,17

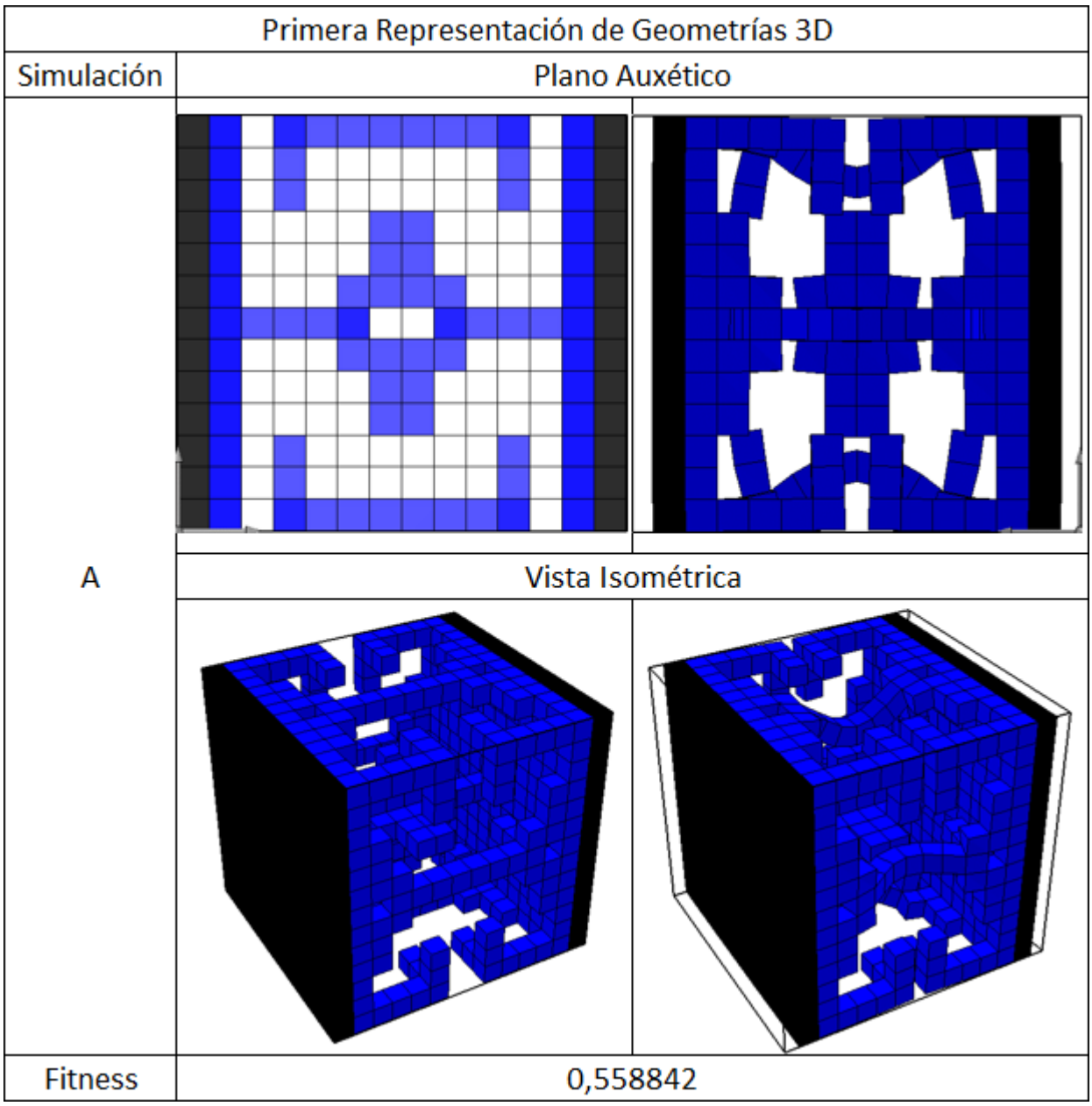


Figura 5.22: Simulaciones para la primera representación de metamateriales tridimensionales, con una celda unitaria, resolución 5x5 y simetría tipo 0.

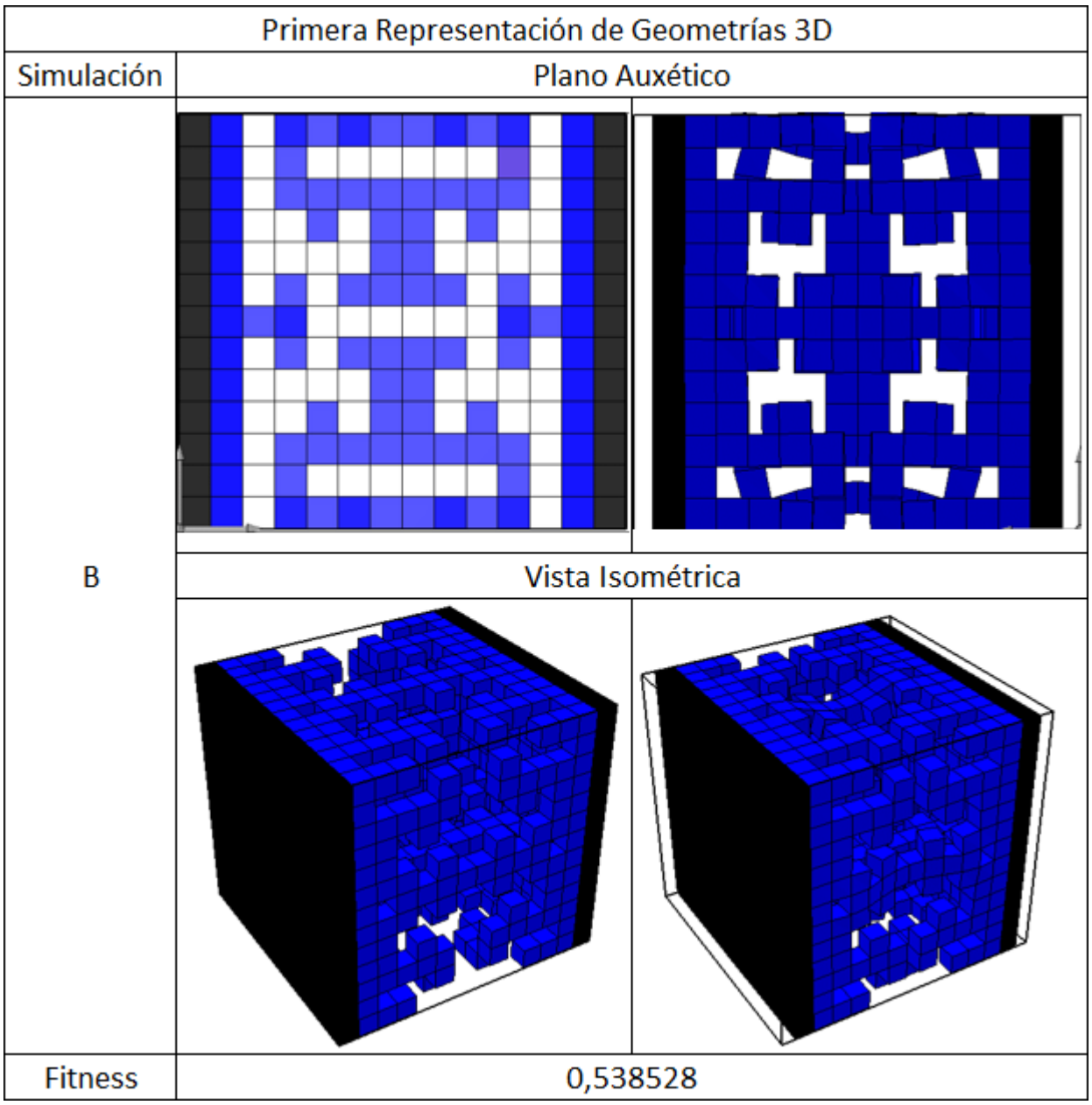


Figura 5.23: Continuación figura 5.22.

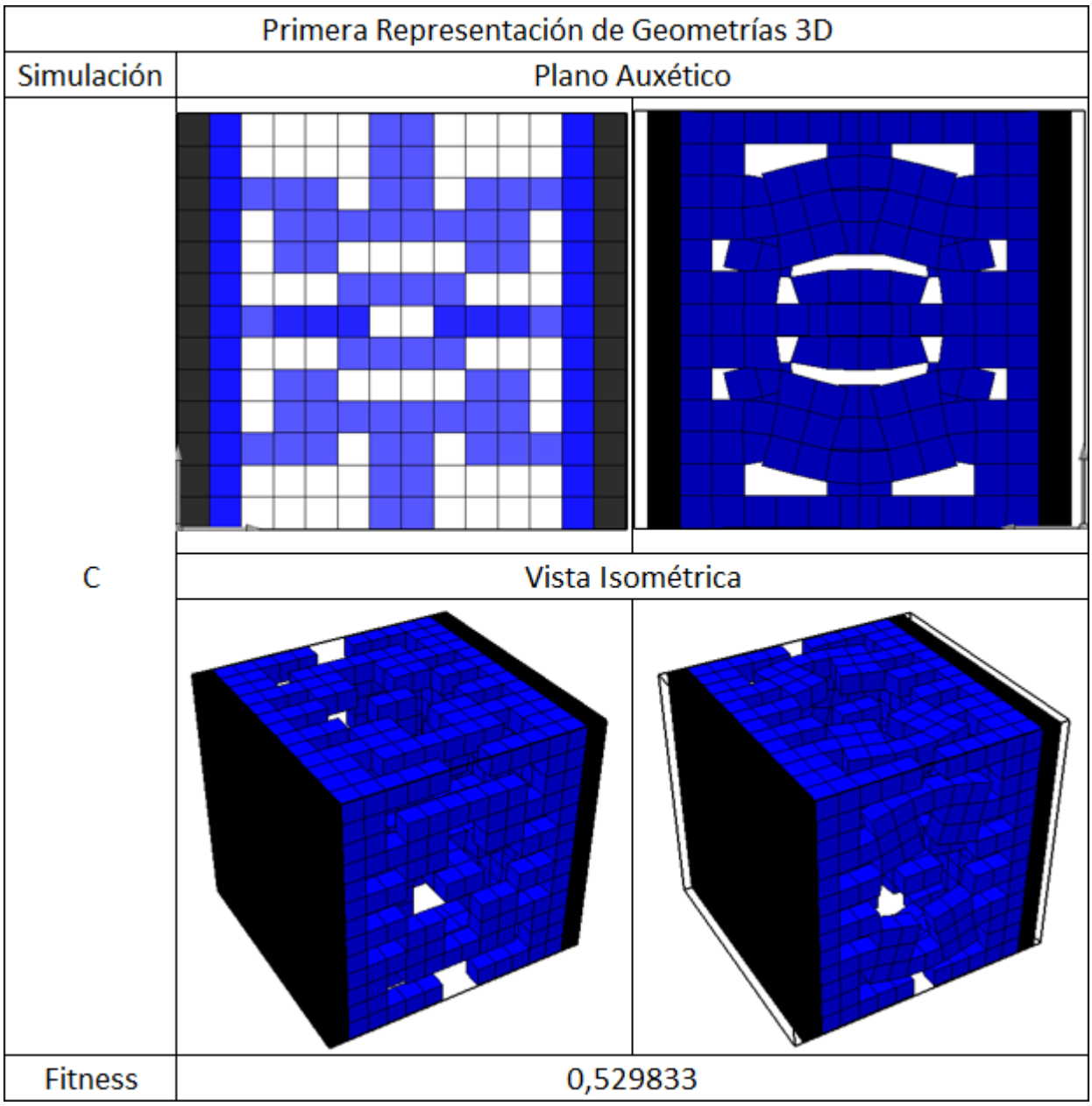


Figura 5.24: Continuación figura 5.22.

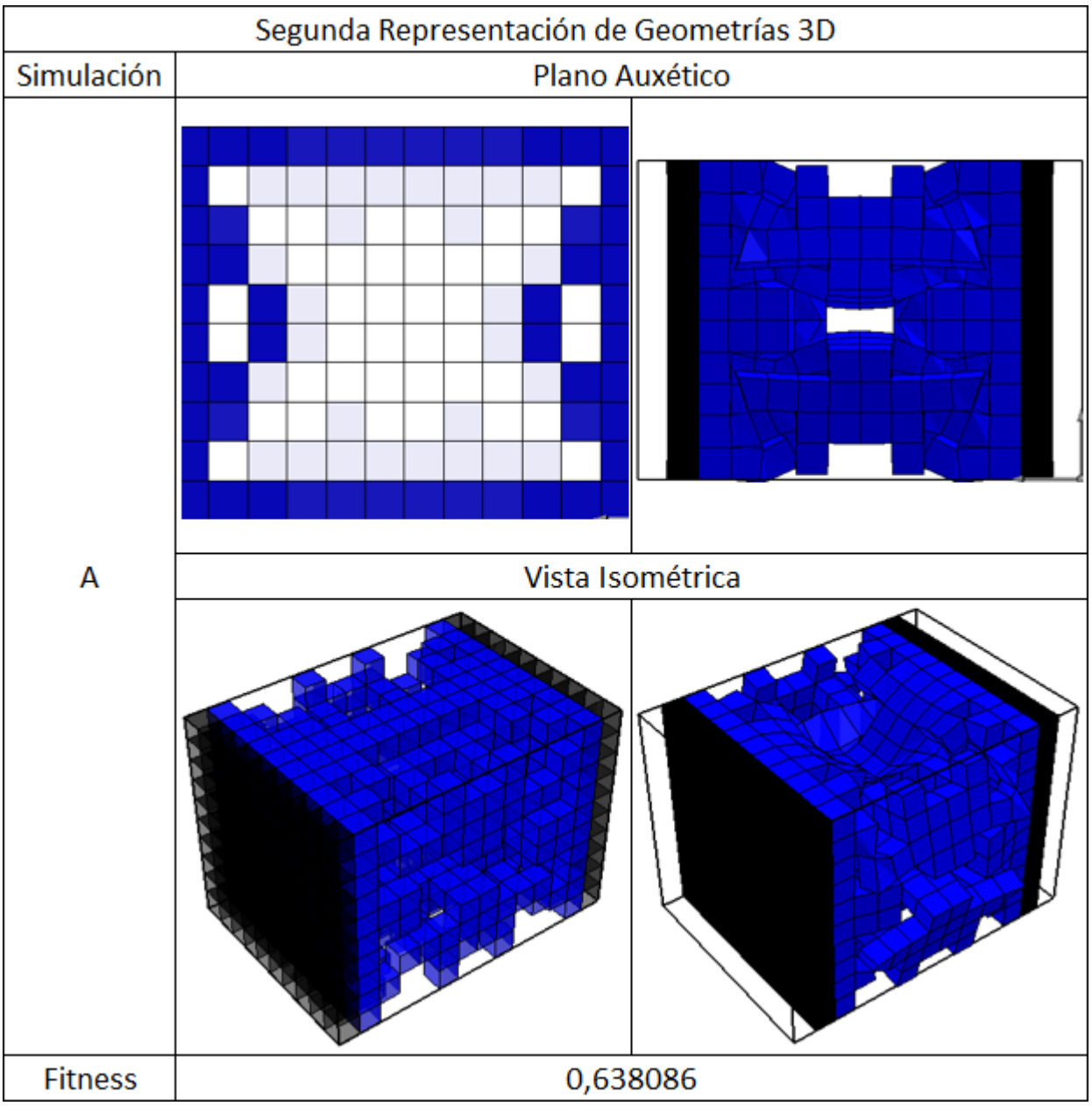


Figura 5.25: Simulaciones para la segunda representación de metamateriales tridimensionales, con una celda unitaria, resolución 5x5 y simetría tipo 0.

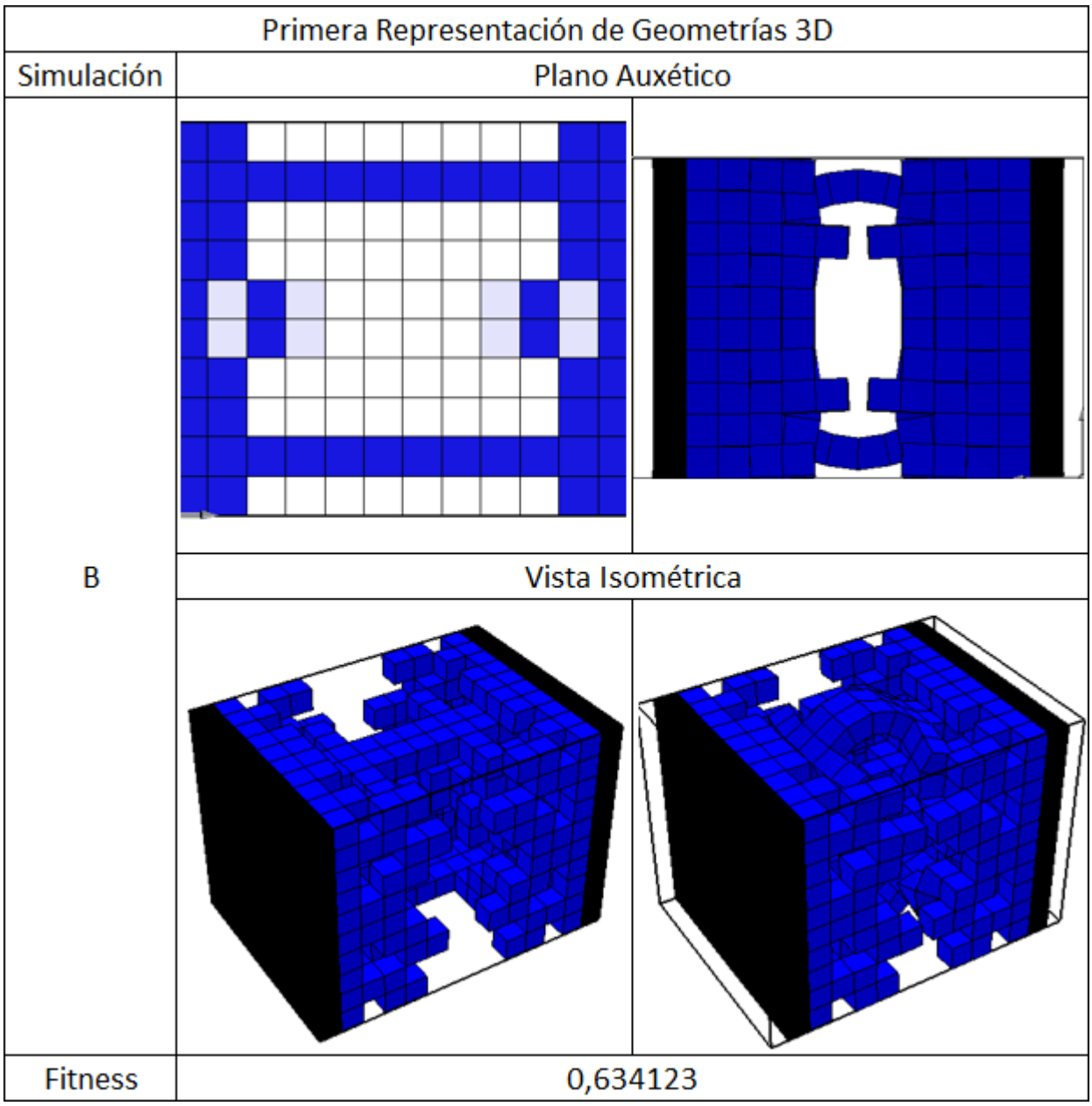


Figura 5.26: Continuación figura 5.25.

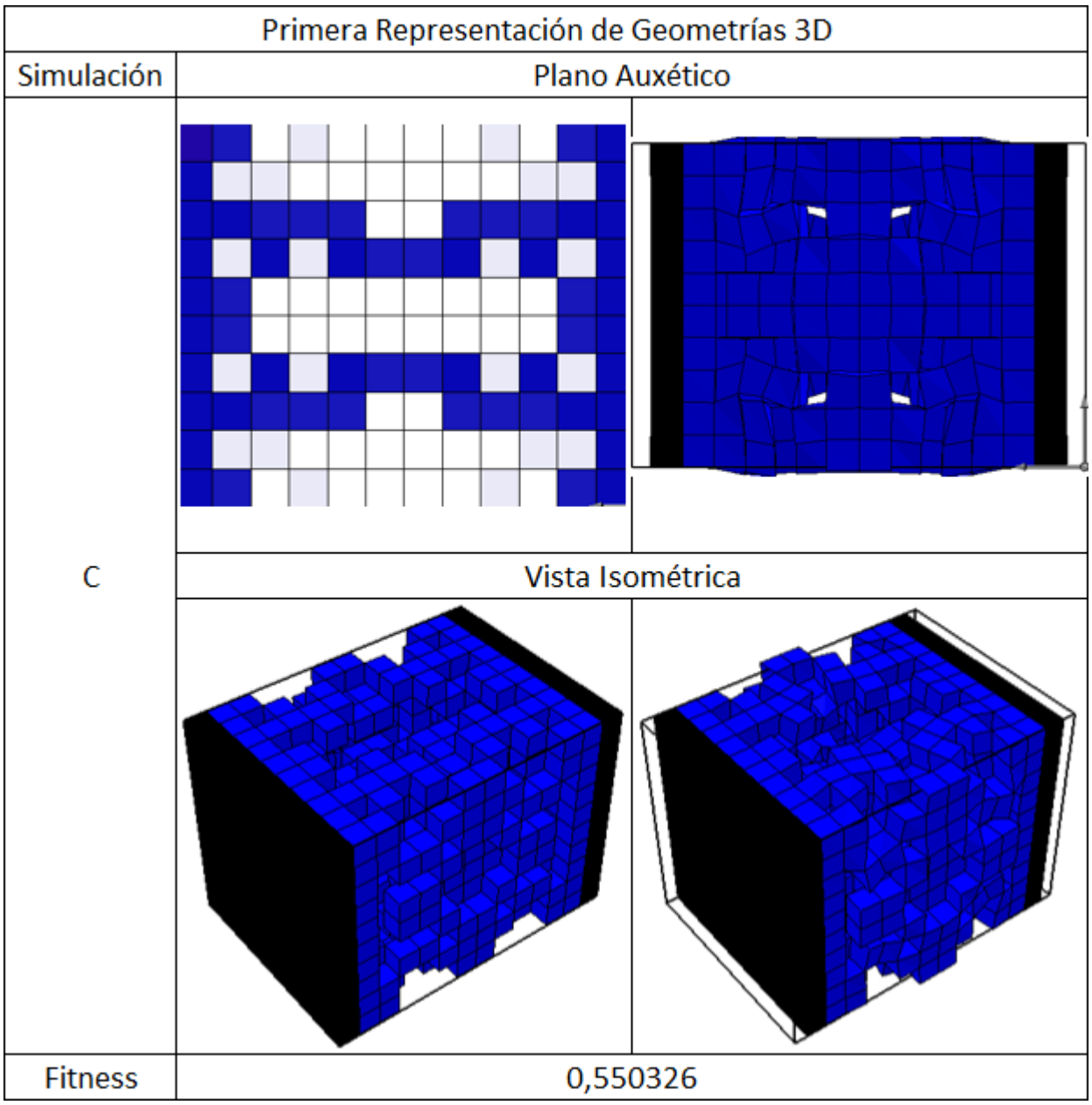


Figura 5.27: Continuación figura 5.25.

5.6. Resultados Experimentales

En la figuras 5.28 a 5.32 se comparan cualitativamente los metamateriales fabricados con los simulados, para los tests de simetrías con una celda unitaria, simetrías con dos celdas unitarias y *seeding*.

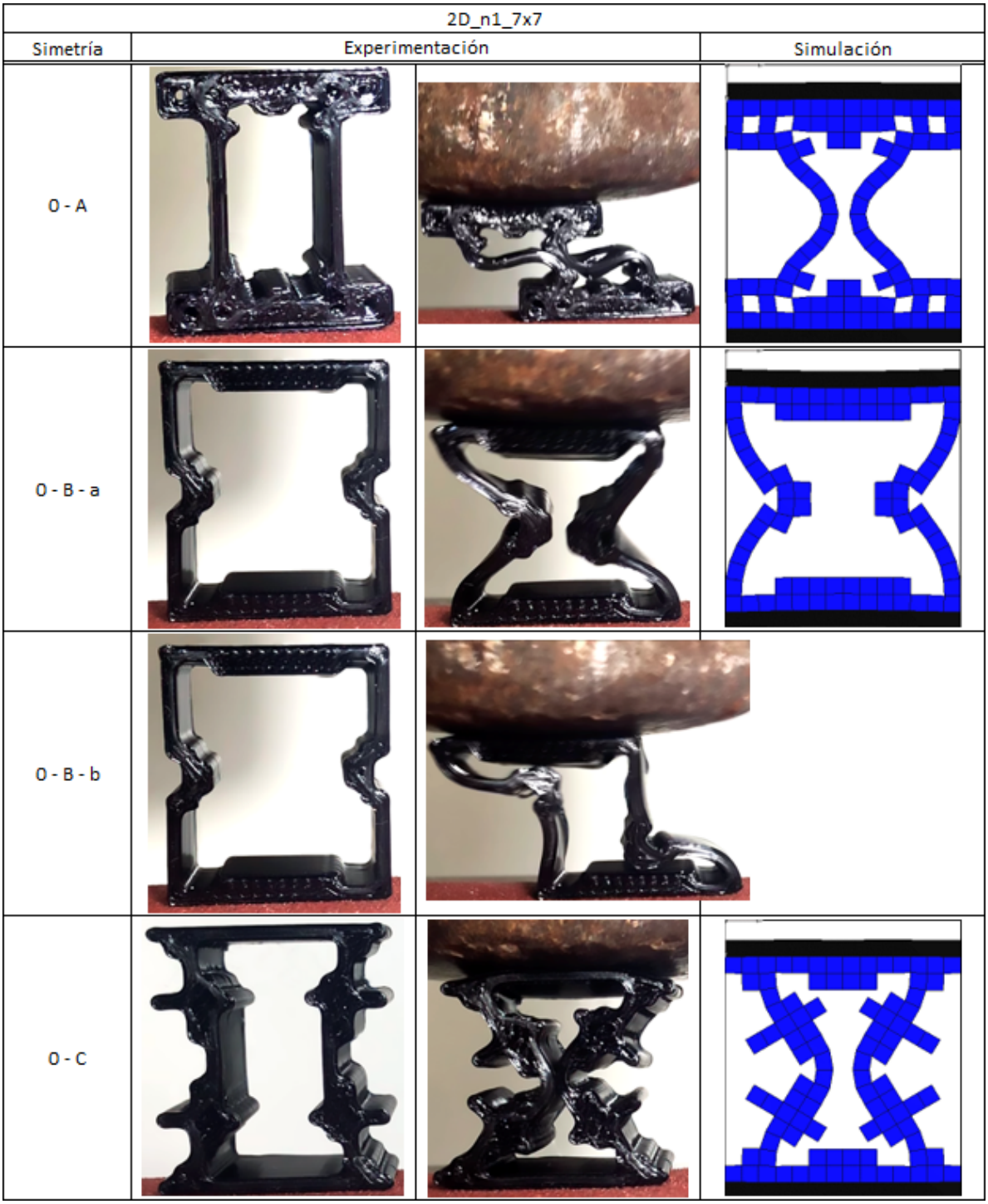


Figura 5.28: Comparación entre el comportamiento auxético real y simulado, para metamateriales bidimensionales de una celda unitaria y resolución 7x7.

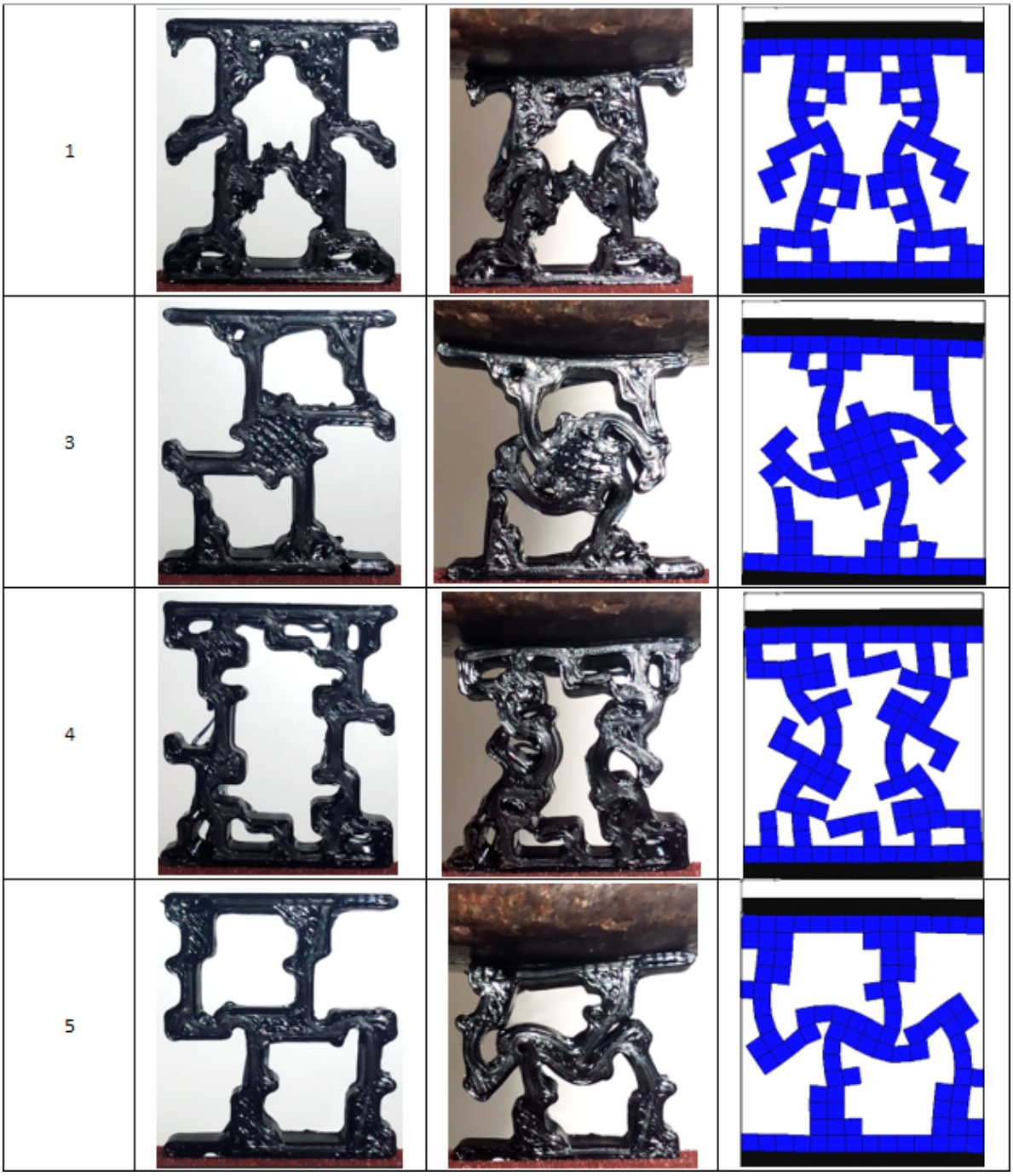


Figura 5.29: Continuación de la figura 5.28.

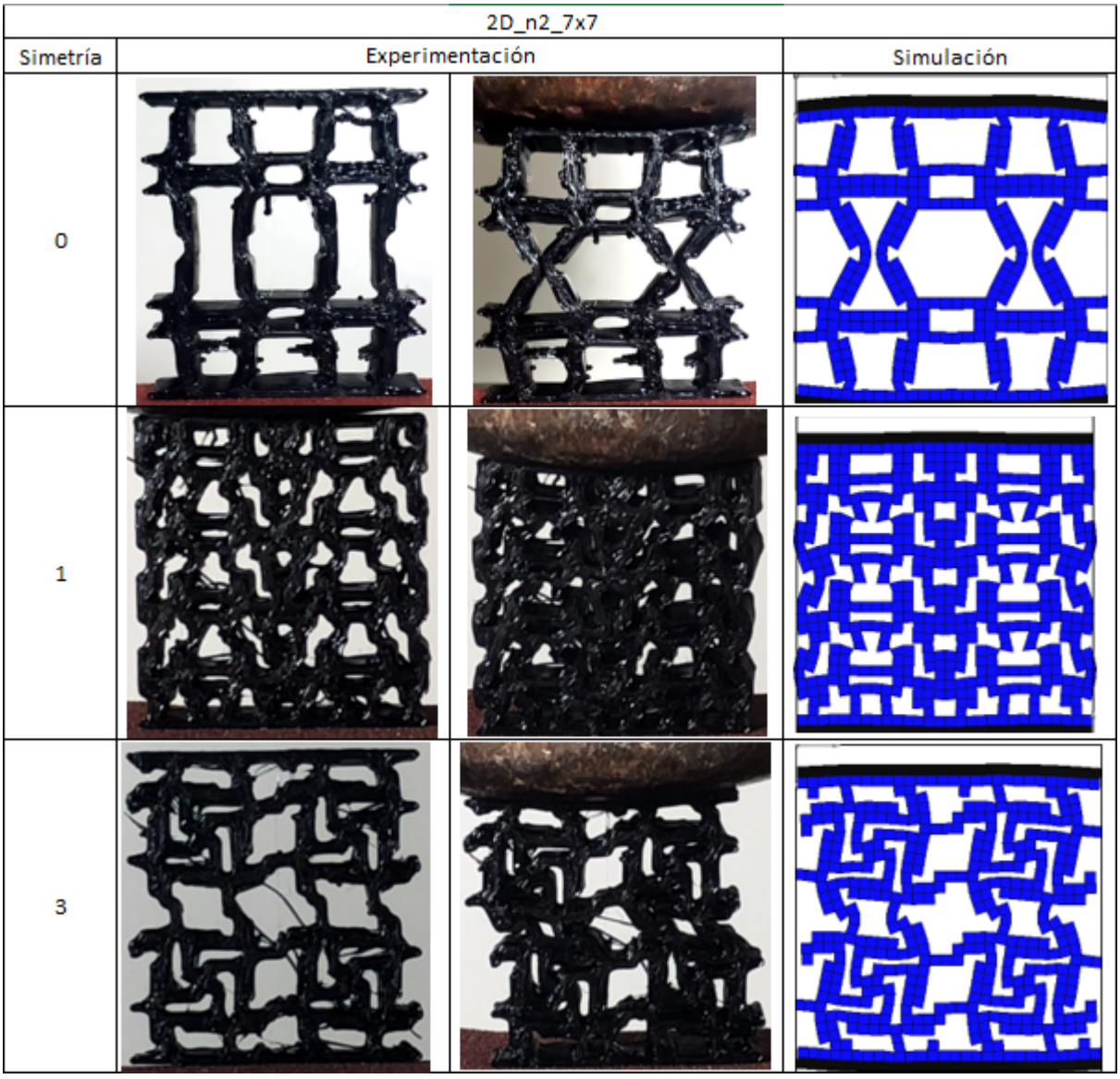


Figura 5.30: Comparación entre el comportamiento auxético real y simulado, para metamateriales bidimensionales de dos celdas unitarias y resolución 7x7.

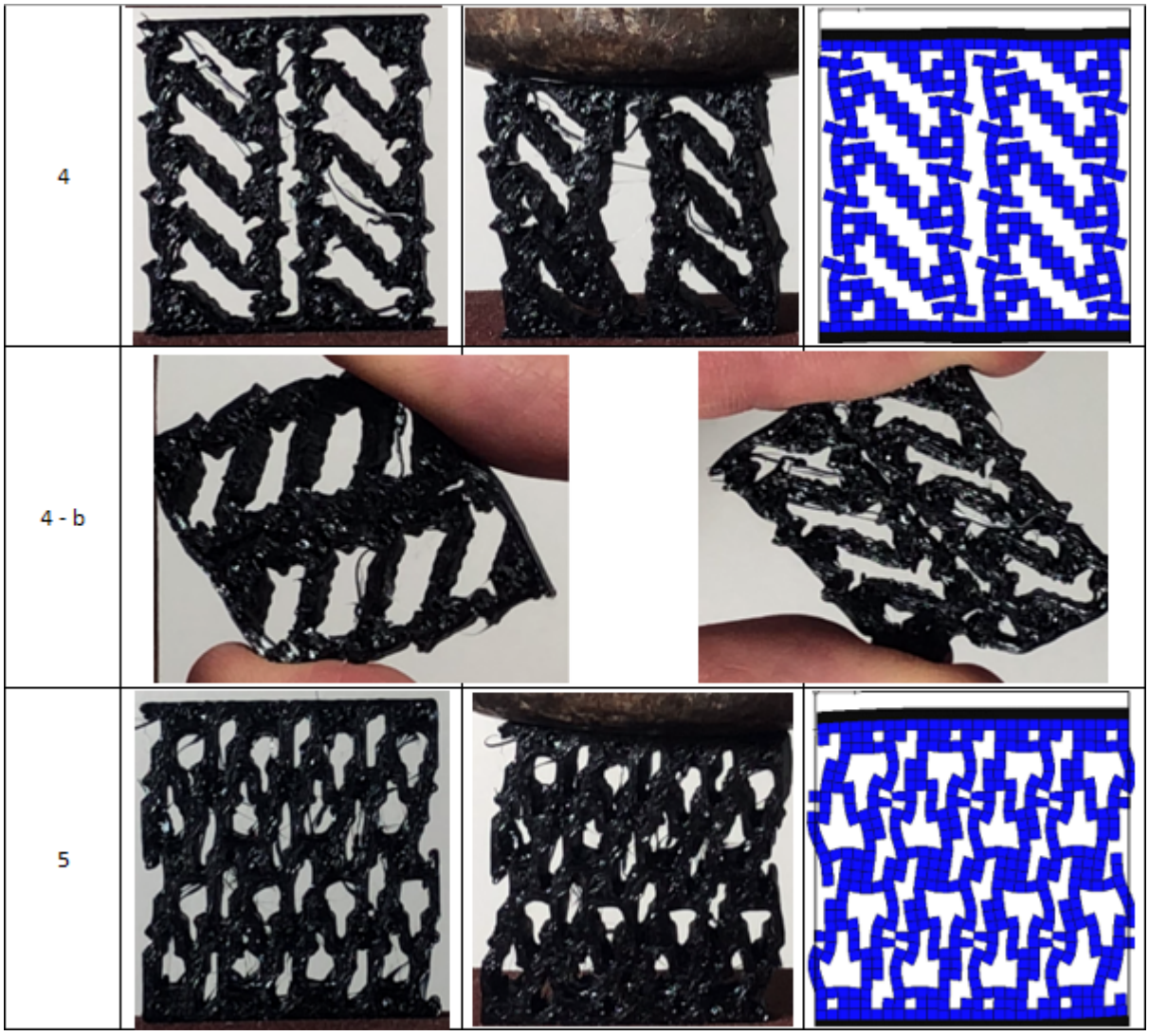


Figura 5.31: Continuación de la figura 5.30.

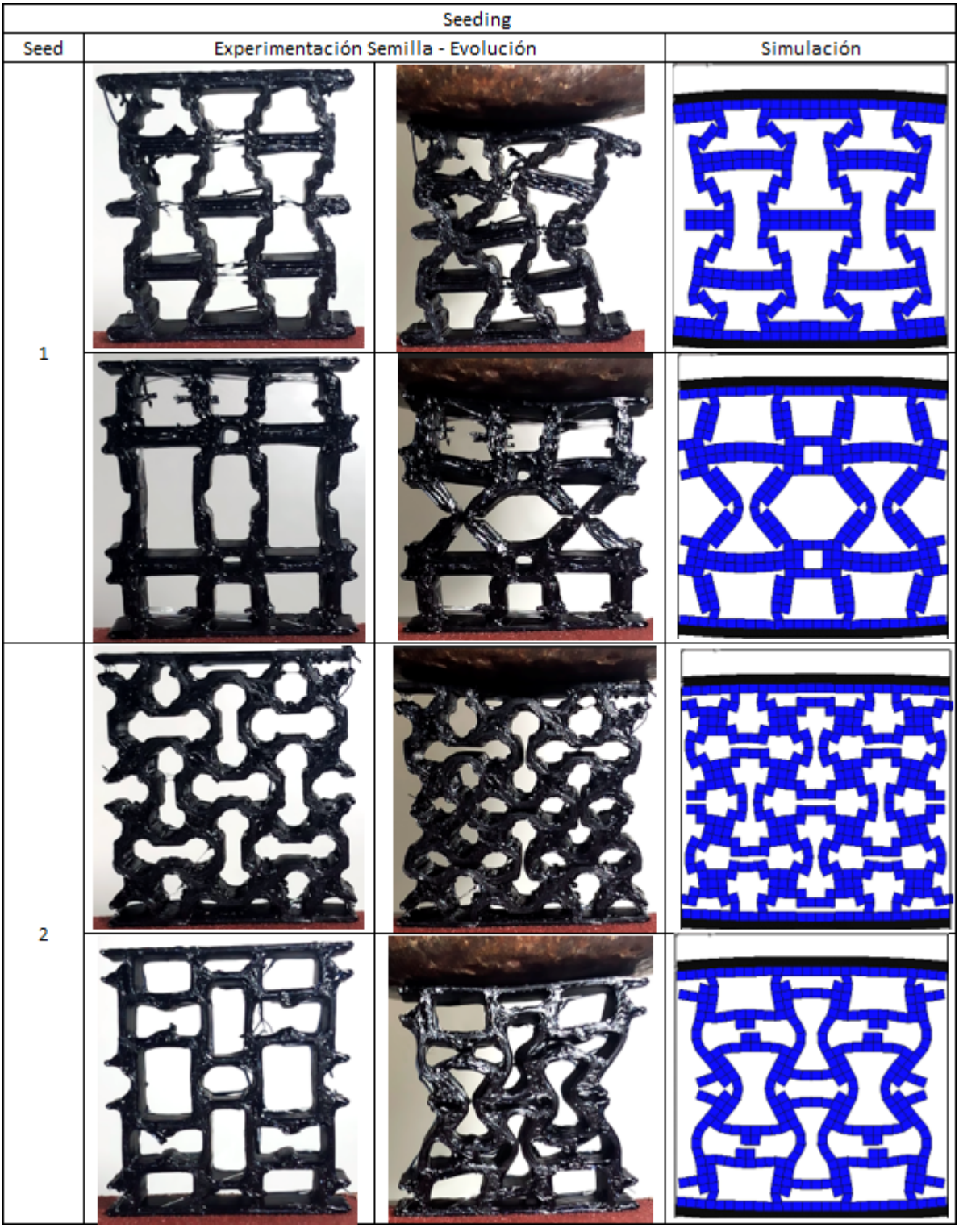


Figura 5.32: Comparación entre el comportamiento auxético real y simulado, para las distintas semillas y sus respectivas evoluciones.

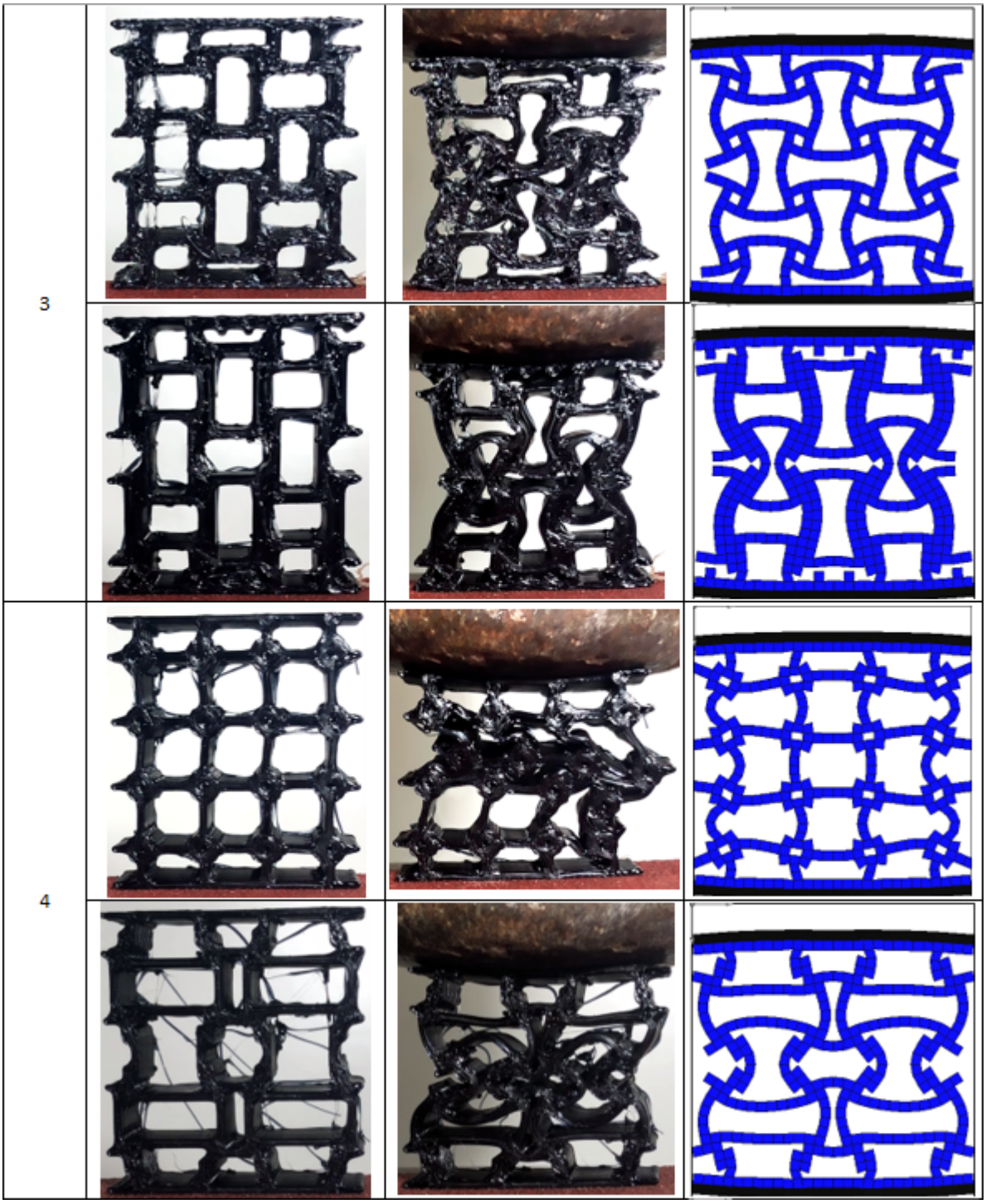


Figura 5.33: Continuación figura de la 5.32.

Figura 5.34: Continuación figura de la 5.32.

5.7. Evolución y Curvas de Aprendizaje

Para ilustrar la manera en que los individuos evolucionan a lo largo del GA, se elaboran videos en los cuales, por un lado, se muestra la curva de aprendizaje y, por el otro, su simulación en VoxCAD. Además, se usa un círculo rojo para indicar a qué parte de la curva de aprendizaje corresponde el candidato a metamaterial simulado. Cada video culmina con una comparación gráfica del comportamiento del metamaterial al ser comprimido, tanto en la simulación como en el entorno experimentación elaborado.

Dichos videos se pueden encontrar en la plataforma de **YouTube**, ingresando por medio del siguiente link:

`https://www.youtube.com/channel/UCs4cMo9XOwwzwm_CfTqDCTA/videos`

Capítulo 6

Análisis y Discusión

6.1. Simetrías Para Metamateriales Bidimensionales con 1 Celda Unitaria

A partir de los diseños con fondo anaranjado de la figura 5.1, se puede observar que, a pesar de haber programado la función `checkFullStripes` para penalizar las topologías inestables, aún así se obtienen metamateriales que consisten en finas hileras que se extienden por medio de amplias regiones vacías. Esto se ve reflejado en las evoluciones A y B del tipo de simetría 0. En el primer caso, se ve que la hilera que se extiende en la cuarta columna burla el sistema ya que en sus extremos posee un elemento vacío. Por otro lado, a partir de la cuarta columna del segundo caso, la amplia región vacía que allí se genera se justifica debido a las barras horizontales construidas en sus extremos superior e inferior. Por lo tanto, queda en evidencia la necesidad de robustecer la programación de dicha función, o bien, buscar otra manera de evitar este tipo de características indeseadas.

Otra observación que cabe realizar en la figura 5.1 es el comportamiento de aquellas topologías que no son simétricas con respecto al eje de compresión, es decir, las simetrías tipo 2, 5 y 6. Estos metamateriales son los que logran el menor *fitness*, para el caso de 5; o bien, son incapaces de generar soluciones factibles, para el caso de 2 y 6. Además, estos dos últimos son auxéticos solo para una de sus mitades, mientras que la otra mitad es no auxética casi en la misma medida, lo cual se traduce en una reducción prácticamente nula del área transversal. Este comportamiento surge debido a que la librería `Voxelyze` solo puede rastrear el desplazamiento de un *voxel* durante la simulación, el cual es elegido mediante la función `findVoxelToTrack`, asumiendo que el comportamiento al otro lado del metamaterial será simétrico y que el resto de los *voxels* se desplazarán en armonía.

Para cerrar el análisis de dicha figura, se destaca la aparición de un tipo de resultado que en un principio no se tenía previsto. Al aplicar simetría tipo 3 (i.e: rotaciones en 90°) el sistema es capaz de generar metamateriales quirales, cuya principal característica consiste en que su geometría interna es capaz de rotar. Además cabe recalcar que, para los 3 testeos, el GA convergió siempre al mismo diseño, pero con leves diferencias entre ellos.

Con respecto a los resultados graficados en la figura 5.3, se observa que el mejor *fitness* corresponde a la simetría tipo 0 (simetría axial en X y Y), la cual es usada por la mayoría de los diseños de metamateriales auxéticos conocidos. Además, considerando que se escogió la evolución C (i.e: la peor del conjunto), aún así es superior al resto, con lo cual queda en evidencia que dicho tipo de construcción es la mejor alternativa si lo que se desea es maximizar la auxeticidad. Sin embargo, esta es también la más susceptible al momento de generar estructuras inestables, por lo que debe usarse con cautela.

En segundo lugar se encuentra la simetría tipo 3, evidenciando la factibilidad que poseen los metamateriales quirales para exhibir un comportamiento auxético a partir de uno rotatorio. Luego, se encuentran las tipo 1 y 4, las cuales presentan simetrías solo respecto al eje de compresión. Estas se destacan del resto por poseer "extremidades" en ambos costados, las que rotan de distinta manera según el tipo de construcción seleccionado. En último lugar, se encuentra el de tipo 5, cuya simetría es con respecto al eje perpendicular al de compresión y por ende presenta un menor *fitness*. Sin embargo, dado que dicha simetría es central en lugar de axial, su compresión también exhibe cierto comportamiento rotatorio, pero en menor medida al resto.

En cuanto a la figura 5.4, se observa que el factor de reducción del área transversal sigue la misma tendencia que el *fitness*. Esta observación se relaciona directamente con la definición 3.1, en la cual se muestra que la función *fitness* le da una mayor prioridad a metamateriales que empiezan con una alta sección transversal y terminan con una sección lo más reducida posible. Nuevamente queda en evidencia la superioridad de la simetría tipo 0 para exhibir cualidades auxéticas.

Por último, en cuanto al factor de compresión axial, cabe mencionar que este entrega una noción relativa a lo que sería el módulo de Young (o módulo de elasticidad) del metamaterial. Cabe recordar que dicho parámetro es directamente proporcional al esfuerzo aplicado e inversamente proporcional a la deformación axial. Dado que la fuerza por unidad de área aplicada a cada metamaterial es la misma dentro de cada test, entonces, mientras más se comprima un metamaterial, menor será su módulo de elasticidad. Y por el contrario, mientras menos se comprima, mayor será dicho módulo. A raíz de esto, se puede apreciar que la simetrías tipo 0 y 3, además de ser las más auxéticas, también son las que ofrecen una menor resistencia a ser comprimidas axialmente.

Para finalizar, se analiza la figura 5.5, donde se comparan los módulos de Poisson registrados para las condiciones de simulación establecidas. Cabe tener en cuenta que, dada la naturaleza no lineal de las deformaciones en metamateriales auxéticos blandos, dicho parámetro es difícil de establecer. Para ello, se debe confeccionar un gráfico de esfuerzo-deformación, función que no ha sido desarrollada dentro del sistema implementado. Sin embargo, de igual manera se observa una tendencia similar al *fitness*, es decir, los mayores módulos son alcanzados por los metamateriales simétricos respecto al eje de compresión y quiral (i.e: simetrías tipo 0, 1, 3 y 4), mientras que el menor módulo se lo lleva el de tipo 5, que no es simétrico respecto a dicho eje. Por otro lado, se destaca la salvedad de que el tipo de simetría 1 tiene un módulo de Poisson más negativo que el de tipo 0. Sin embargo, para asegurar que dicho diseño es más auxético, habría que mirar las curvas de esfuerzo-deformación de los metamateriales simulados.

6.2. Simetrías Para Metamateriales Bidimensionales con 2 Celdas Unitarias

A partir de la figura 5.6 se pueden observar nuevas características a estudiar. En primer lugar, producto de que ahora se están tomando en cuenta las interacciones entre celdas unitarias, los metamateriales poseen una estructura más estable respecto al caso anterior. Por otro lado, debido al aumento en la complejidad de las interacciones entre elementos finitos, los valores de *fitness* logrados son menores. Para el caso de una celda unitaria, estos se encontraban en el rango $[0,56; 0,62]$, mientras que para dos celdas unitarias estos disminuyen a un rango $[0,51; 0,56]$. Por otro lado, se sigue registrando un comportamiento no auxético para las simetrías tipo 2 y 6, debido a las mismas razones expuestas en la subsección anterior.

Según la figura 5.8, se vuelve a tener un mejor valor de *fitness* para la simetría tipo 0, la cual además es considerablemente mayor al del resto. Sin embargo, esta vez el tipo de simetría 5 pasa de ser el peor al segundo mejor. Esto podría indicar que, si bien este tipo de simetría no es eficiente por sí sola, puede lograr una mejor interacción interna a una mayor escala. Ahora bien, respecto al resto, se sigue observando la misma tendencia de los casos con una celda unitaria, en la cual el metamaterial quiral tiene un valor de *fitness* superior a aquellos que son simétricos solamente respecto al eje de compresión, siendo el tipo de simetría 1 (traslación + axial) mayor al de tipo 4 (traslación + central).

En cuanto a la figura 5.9, el factor de reducción de área sigue la misma tendencia respecto al *fitness*, con la salvedad de que la simetría tipo 5 es la tercera mejor en lugar de la segunda. Esta excepción puede parecer contradictoria en un principio, pero no lo es necesariamente, ya que según la definición 3.1, la función objetivo no usa el factor de reducción de área transversal al momento de calcular el *fitness*.

Por otro lado, respecto al factor de compresión axial, se aprecia un comportamiento análogo al visto anteriormente. Esto es, la simetría tipo 0 es la más fácil de comprimir, seguida por las simetrías tipo 3, 4 y 5, siendo la simetría tipo 1 la más rígida del set.

Por último, a partir de la figura 5.10, se analizan los coeficientes de Poisson estimados para las distintas simetrías. Aquí se destaca el hecho de que, a pesar de que la definición de la función *fitness* y la medición del módulo de Poisson son dos procesos totalmente independientes (i.e: no se usa uno como factor para calcular el otro), aún así ambos siguen la misma tendencia, correspondiéndose a su vez casi totalmente con la del factor de reducción de área transversal. Esta coherencia entre los resultados obtenidos evidencia la calidad tanto de la simulación como de la definición de la función objetivo, además de sumarle cierto grado de confiabilidad a la herramienta de diseño implementada.

6.3. Seeding

El análisis para la técnica de *seeding* implementada se subdivide en 2 partes. En primer lugar, se estudia la influencia que tiene el factor `pbias` en el tipo de solución generada, y en segundo lugar, los resultados de la simulación, comparando las semillas implementadas en VoxCAD, su evolución y los resultados originales del estudio base [6].

6.3.1. Relación Entre `pbias` y el Tipo de Solución Generada

En primer lugar, cabe recordar que el factor `pbias` representa el grado de influencia que tendrá la semilla sobre el genoma a evolucionar. En resumidas palabras, a menor `pbias`, menos se forzará al genoma para que se parezca a la semilla, haciendo que la evolución se asemeje al caso aleatorio o sin *seeding*. En caso contrario, mientras más alto sea `pbias`, más similares serán los individuos generados a la semilla implementada.

A partir de la figura 5.14, se puede observar que los valores de *fitness* más altos se obtienen generalmente para un menor valor de `pbias`. Esto se debe a que, mientras menos influencia se ejerza sobre el genoma para que este adopte cierta topología en particular, más probable será que este evolucione en diseños fuera de la línea de convergencia impuesta por la semilla implementada.

En cuanto a la figura 5.15, se observa que a mayor `pbias`, más se parece el genoma del resultado obtenido al genoma de la semilla. Solo en uno de los casos se presenta el comportamiento contrario. Esta tendencia también se ve reflejada cualitativamente en las figuras 5.11 a 5.13.

Por último, respecto a la figura 5.16, se espera que mientras más se fuerce al genoma a que adopte cierta forma bien determinada, debería converger dentro de un número menor de generaciones. Sin embargo, no siempre es ese el caso. Para dos de las tres semillas ocurre totalmente lo contrario.

Si bien para los tres aspectos estudiados anteriormente las tendencias medidas no siempre fueron iguales a las esperadas, es probable que estas coincidan luego de realizar un mayor número de pruebas, hasta tener una estadística lo suficientemente robusta como para obtener conclusiones al respecto.

En resumen, a un menor `pbias`, mayor será el *fitness* obtenido con respecto al de la semilla, su morfología será más distinta y se demorará más en converger. Por el contrario, a un mayor `pbias`, más similar será el resultado de la evolución a la semilla, tanto en *fitness* como morfología, y además la convergencia del algoritmo será más rápida.

6.3.2. Resultados de las Simulaciones Implementando *Seeding*

Al observar el gráfico de la figura 5.17, para los casos 1, 2 y 3, el valor del *fitness* aumenta considerablemente con respecto al de la semilla una vez terminada la evolución. Además, para las semillas 1, 2 y 5, se logra un valor de *fitness* mayor al caso de la evolución aleatoria sin semilla, lo cual confirma la afirmación de que, al usar técnicas de *seeding* adecuadas, el GA puede converger a mejores soluciones.

Luego, respecto a los factores de reducción del área transversal graficados en la figura 5.18, las evoluciones obtienen un valor más alto respecto a sus semillas, para los tres primeros casos. Más aún, en cada una de las semillas implementadas, la evolución final obtuvo un mejor factor de reducción de área transversal que el caso sin semilla, lo cual denota una vez más las ventajas de usar este tipo de técnicas dentro de un GA. Por último, se observa que los resultados logrados con VoxCAD son considerablemente inferiores a los que se encontraron

en la literatura [6], lo cual es de esperarse principalmente por la naturaleza completamente distinta de ambos trabajos, y por otro, debido a la poca cantidad de elementos finitos que componen las estructuras diseñadas en VoxCAD, las cuales no permiten elaborar figuras perfectamente redondas, puntiagudas o incluso paredes diagonales. No así los diseños fabricados en el estudio de referencia, en el cual modelan los diseños de metamateriales usando *SolidEdge* y realizan las simulaciones en un *software* de elementos finitos. Aún así, cabe destacar que las diferencias entre los diseños evolucionados y los del estudio de referencia no llegan a ser abismales, puesto que los factores de reducción de área obtenidos se encuentran al rededor de un 50 % con respecto a los teóricos.

Ahora bien, respecto a los factores de compresión graficados en la figura 5.19, se puede observar que los metamateriales evolucionados suelen ser más rígidos que las mismas semillas, puesto que estos se comprimen en menor medida frente al mismo esfuerzo de compresión. Además, cada una de las evoluciones son también más rígidas que el caso sin semilla. Sin embargo, estas diferencias son más bien finas, puesto que todas las mediciones (excepto para la semilla 5) se encuentran en el rango $[0{,}12; 0{,}13]$, es decir, la distancia de compresión está entre un 12 y un 13 % con respecto al largo total de la pieza. Esto quiere decir que las diferencias en la distancia de compresión para las semillas y sus evoluciones no llegan a ser más de un 1 % del largo total de la pieza, es decir, menor a $0{,}3[mm]$

Por último, se analizan los coeficientes de Poisson graficados en la figura 5.20. De igual manera que en las propiedades anteriormente estudiadas, se nota un aumento en el resultado de la evolución con respecto a la semilla. Además, a excepción de la semilla 4, se lograron coeficientes mayores al caso sin semilla. Sin embargo, se destaca que esta vez, para los casos 2, 3 y 4, la diferencia entre el coeficiente de Poisson de las evoluciones y el de la literatura disminuye notablemente, casi llegando a igualarse en el caso de la semilla 5. Esto podría significar que las deformaciones involucradas, tanto para las evoluciones y los metamateriales del estudio base, ocurren casi a la misma razón, pero estos últimos pueden reducir su área en mayor medida debido a que las paredes de sus estructuras internas son más esbeltas y flexibles, o debido también a que se simularon una mayor cantidad de tiempo. En cualquier caso, esta similitud en los coeficientes de Poisson da cabida a la posibilidad de obtener resultados mejorados si se refinan los diseños de los metamateriales generados por el sistema, o incluso al momento de medir las deformaciones que estos exhiben en la realidad.

6.4. Simulaciones alostéricas

A partir de la figura 5.21 se puede ver que efectivamente se logra un comportamiento alostérico al traccionar el metamaterial. Sin embargo, el área transversal no excede los límites del *workspace*. Por otro lado, en cuanto a los resultados de la tabla 5.5, se puede observar que se logran coeficientes de Poisson similares a los registrados para los metamateriales auxéticos. Además, el metamaterial alostérico de una celda unitaria posee un notable aumento de área transversal.

Si bien este tipo de simulación maximiza el comportamiento alostérico de los metamateriales, es claro que dichos resultados también son válidos para exhibir comportamientos auxéticos. Por lo tanto, dicha función podría tener el potencial de generar metamateriales para aplicaciones en las que se necesite tanto una buena respuesta auxética como alostérica.

6.5. Geometrías 3D

En la figura 5.22, correspondiente a la primera representación de geometrías tridimensionales, se puede observar que una de las principales ventajas de intercalar planos bidimensionales auxéticos es que, al comprimir el objeto, cada una de sus caras laterales exhibe un comportamiento auxético de igual magnitud. Esto principalmente debido a la simetría usada para elaborar y disponer las celdas unitarias.

Por su parte, la segunda representación de metamateriales tridimensionales, recopiladas en la figura 5.25, no presenta un comportamiento tan uniforme como la representación anterior. Debido a su forma irregular, es más complejo lidiar con la programación sus características para generar diseños factibles. Por ejemplo, para este caso, no es del todo evidente restringir la existencia de hileras completamente llenas o vacías, ya que es probable que puedan existir modelos de metamateriales tridimensionales sólidos que necesiten tener ciertas zonas con dicha característica.

Por otro lado, debido a la menor presencia de simetrías internas dentro de la estructura del metamaterial, y eso ligado al hecho de que la librería `voxelyze` solo puede rastrear el desplazamiento de un *voxel* durante la simualción, se tiene como resultado que solo un par de sus caras laterales sea auxética, mientras que con el otro par puede suceder cualquier cosa.

Otra característica preocupante dentro de esta representación es que, por ejemplo, en el test B, la simulación en `voxelyze` determinó que dicho metamaterial es auxético, puesto que su *fitness* es mayor a 0,5. Sin embargo, al momento de ver su comportamiento en VoxCAD, se observa que dicho diseño es en verdad no auxético, lo cual demuestra un serio problema a resolver.

A pesar de las características negativas anteriormente mencionadas, es importante reconocer el alto potencial que tiene esta representación tridimensional, ya que su forma irregular también puede significar un nuevo campo de oportunidades, puesto que se podrían diseñar metamateriales que, por ejemplo, en un par de caras se auxético, y en el otro par sea quiral, dando la posibilidad de generar eventualmente mecanismos blandos.

Por último, respecto a los resultados presentados en la tabla 5.6, se puede observar el elevado coeficiente de Poisson registrado en dichas simulaciones, lo cual puede significar que dichos diseños tienen un alto potencial, o bien, que sus simulaciones son altamente inestables y requieren de un código más robusto para conseguir soluciones factibles.

6.6. Resultados Experimentales

Los resultados experimentales serán un factor decisivo al momento de determinar la factibilidad y el verdadero potencial del sistema implementado, ya que, de nada sirve que los metamateriales sean auxéticos en un simulador de elementos finitos blandos, si estos no pueden emplearse posteriormente para encontrar aplicaciones concretas en la robótica blanda o en la industria.

En la figura 5.28 se pueden observar los 3 testeos relacionados con el metamaterial bidimensional de una celda unitaria y simetría tipo 0. La prueba A, que logró alcanzar un mejor *fitness*, es altamente inestable. La prueba B por su parte podría decirse que es "metaestable", ya que en algunas ocasiones lograba comprimirse de manera análoga a la simulación, pero en otras colapsaba debido a su inestabilidad, o incluso, en algunas ocasiones exhibía un comportamiento no auxético. Por último, se tiene la prueba C, en la cual se consiguió un metamaterial de menor *fitness*, pero con un comportamiento más estable y, por ende, más confiable.

En la figura 5.29 se pueden apreciar los ensayos de compresión realizados en el resto de las simetrías, para los metamateriales bidimensionales de una celda unitaria. Para cada una de estas pruebas, su desempeño real es prácticamente igual al simulado, lo cual permite aseverar que la herramienta desarrollada a lo largo del presente trabajo de título es capaz de diseñar exitosamente nuevos modelos de metamateriales auxéticos blandos, los cuales incluso en algunos casos combinan la auxeticidad con movimientos torsionales.

En la figura 5.30, correspondiente a los metamateriales de 2 celdas unitarias, se puede observar que, para la simetría tipo 0, se obtiene un comportamiento auxético igual al de la simulación. La simetría tipo 1, por su parte, no parece ser del todo auxética, tal y como en la simulación. La simetría tipo 3 se ve que colapsa en su zona central, y las 4 figuras quirales a sus extremos prácticamente son incapaces de rotar, lo cual no se condice con lo que se ve en la simulación, donde sí rotan un poco.

Por otro lado, en la figura 5.31, se observa que la simetría tipo 4 es más inestable de lo que se aprecia en la simulación. Sin embargo, se descubre que, al aplicar un esfuerzo de compresión **paralelo** a los patrones diagonales, el metamaterial exhibe un comportamiento **no auxético**. Mientras que al comprimirlo de manera **perpendicular** a sus patrones diagonales, el metamaterial exhibe un comportamiento **auxético**. Dicha característica podría ser aprovechada, por ejemplo, para crear algún dispositivo de bombeo metamaterial que alterne continuamente ambas respuestas. Por último, respecto a la simetría tipo 5, se logra identificar un leve comportamiento auxético, igual que en la simulación.

A partir de las figuras analizadas, queda en evidencia la dificultad que posee el sistema para generar diseños auxéticos a gran escala, particularmente para los tipos de simetría del 1 al 6. Esto puede deberse al aumento en la complejidad de las interacciones entre celdas unitarias contiguas. Sin embargo, se espera que esta dificultad pueda superarse al robustecer el código del sistema y las consideraciones de diseño establecidas.

Ahora bien, siguiendo con la figura 5.32, se puede observar que tanto las semillas como sus evoluciones presentan un notable comportamiento auxético, aún cuando estas tienen las mismas dimensiones que los metamateriales de dos celdas unitarias y resolución 7x7 analizados anteriormente, los cuales registraron un comportamiento auxético deficiente. Por otro lado, se destaca que solo la semilla número 4 tiene un comportamiento distinto al de su simulación en VoxCAD. Sin embargo, aún así se logra elaborar un metamaterial auxético estable a partir de ella.

6.7. Evolución y curvas de aprendizaje

Con respecto a los videos colgados a la plataforma de YouTube, se puede verificar que las curvas de aprendizaje de cada evolución se encuentran efectivamente completas. Es decir, la curva de *fitness* promedio alcanza a igualar a la de *fitness* máximo en las últimas generaciones, convergiendo de esta manera a diseños prolijos y eficientes. También se puede apreciar a medida que avanza el video que los candidatos a metamateriales son cada vez más auxéticos y su diseño es más uniforme, hasta llegar al campeón del algoritmo.

Por último, en la sección final de cada metraje, donde se compara la respuesta auxética de las simulaciones con la realidad, se observa que estas coinciden en la gran mayoría de los casos. Asegurando así la factibilidad que posee el sistema desarrollado y que vale la pena seguir perfeccionándolo con el objetivo de posibilitar la creación de diseños más estables y con mejor respuesta auxética.

Capítulo 7

Trabajo Propuesto

Por medio de este sistema se han logrado obtener una cantidad no menor de diseños novedosos de metamateriales auxéticos. Además, sus amplias combinaciones de parámetros de entrada podrían dar lugar a un sinnúmero de experimentos posibles. Sin embargo, como se discutió en las secciones anteriores, dicho sistema aún posee un potencial tan grande como inexplorado. Debido a esto, en la presente sección se resumen distintos aspectos del sistema que se podrían testear, modificar o añadir. Estos son:

- **Aspectos a testear**:
  1. Realizar las mismas pruebas anteriormente detalladas, pero estableciendo otras combinaciones de parámetros de entrada. Por ejemplo, probar tiempos de simulación más largos, resoluciones irregulares como pueden ser 5x7, 6x8, etc.
  2. Generar modelos 3D en Fusion360 donde los diseños presentados anteriormente se repitan varias veces a lo largo y a lo ancho, para luego imprimirlos y estudiar cómo el incremento en la escala de repetición afecta a sus propiedades mecánicas.
  3. Medir las deformaciones de los metamateriales fabricados, para luego calcular su reducción de área transversal y coeficiente de Poisson en la realidad, y así compararlos tanto con los resultados de las simulaciones como con los metamateriales auxéticos conocidos en la literatura.
  4. Imprimir los metamateriales diseñados usando resina elástica, para luego verificar si de esta manera presentan una mejor respuesta auxética, puesto que el filamento de TPU podría estar provocando cierto comportamiento anisotrópico en las piezas testeadas.
- **Aspectos a modificar o mejorar**:
  1. Modificar la librería `voxelyze` para que esta sea capaz de rastrear más de un *voxel* dentro de la simulación. Esto podría ser extraordinariamente útil para medir más puntos dentro de un metamaterial, y así abrir camino al desarrollo de prototipos de mecanismos blandos mediante algoritmos genéticos.
  2. La modificación anterior también podría ser útil para ver si las simetrías tipo 2 y 6 son capaces de diseñar metamateriales auxéticos, ya que podrían observarse ambos lados de la sección transversal y penalizar aquellas topologías en las cuales ambas partes se deformen en el mismo sentido.

3. Robustecer y complejizar el generador de geometrías alostéricas para que estas puedan expandirse excediendo el volumen del *workspace*, o que incluso tengan formas tridimensionalmente más elaboradas.
4. Otro elemento que también posee un gran potencial es la segunda representación de geometrías tridimensionales, la cual se puede estudiar mas a fondo para estabilizar su comportamiento, para lo cual sería útil lograr hacer que la librería `voxelyze` acepte más de un *voxel* para rastrear.

- **Aspectos a añadir**:
  1. Programar una función que permita generar un gráfico de esfuerzo-deformación para el resultado final de una evolución genética. De esta manera, se podría tener una mejor comprensión del comportamiento no lineal de los metamateriales diseñados y de su módulo de Poisson.
  2. Diseñar un algoritmo que transforme topologías imposibles de estudiar, como la de la figura 3.9, en diseños válidos. Por ejemplo, rellenando los extremos inconexos y robusteciendo las uniones entre *voxels* conectados diagonalmente. De esta manera, no sería necesario elevar tanto la población inicial del GA, agilizando así la evolución del mismo.
  3. Respecto al *seeding*, podría realizarse una búsqueda intensiva de diseños que sean posibles de implementar en VoxCAD, añadiendo así mas semillas a la colección, a partir de las cuales podrían obtenerse nuevos y mejores diseños de metamateriales auxéticos.
  4. Otra función que se puede añadir con respecto al *seeding* es hacer de este un proceso iterativo, vale decir, usar los mismos metamateriales diseñados por el sistema para usarlos como semillas y de esta manera ver si dichos modelos se pueden seguir mejorando.
  5. Se considera también la posibilidad de generar metamateriales compuestos por múltiples materiales. Es decir, que el algoritmo genético elabore estructuras en las cuales ciertas zonas sean más rígidas y otras más blandas.
  6. Por último, se propone la idea de extrapolar las geometrías de baja resolución de imagen generadas por el sistema, a unas de mayor resolución, y de esta manera obtener formas más continuas, uniformes y con una respuesta auxética potencialmente mayor.

Conclusión

En el presente trabajo de título se logra desarrollar una herramienta computacional capaz de diseñar metamateriales mecánicos auxéticos blandos mediante su evolución dentro de un algoritmo genético. Dicha herramienta posee un amplio abanico de parámetros de entrada que pueden combinarse de múltiples formas y, de esta manera, generar una infinidad de metamateriales distintos y novedosos. Además, dicho sistema ofrece la posibilidad de usar metamateriales auxéticos conocidos para usarlos dentro de la evolución y así generar diseños mejorados.

Otra de las principales fortalezas de este sistema es que, independiente de los parámetros de diseño escogidos, la evolución genética no se detendrá hasta que el algoritmo determine que ha convergido a una solución suficientemente óptima, lo cual garantiza que cada vez que este sistema se ejecute, se obtendrán buenos resultados.

Un aspecto más que cabe considerar es la coherencia que existe entre los resultados medidos en las simulaciones. A pesar de que el *fitness*, el factor de reducción de área transversal y el coeficiente de Poisson se obtienen mediante métodos distintos e independientes, estos presentan tendencias similares en la mayoría de los casos.

La fiabilidad de las soluciones entregadas por este sistema se corrobora al momento de fabricar las piezas diseñadas y observar que, al menos cualitativamente, presentan un comportamiento auxético igual o mejor al de las simulaciones. Este hallazgo es de suma importancia puesto que abre la posibilidad a seguir mejorando el sistema e incluso usarlo como una herramienta más dentro de otros estudios.

Sin embargo, actualmente el sistema implementado posee importantes limitaciones y características indeseadas. Se determinó que para metamateriales cuya estructura tenga alrededor de 30 *voxels* de largo y ancho, las simulaciones comienzan a exhibir comportamientos irreales e inestables. Por ejemplo, las deformaciones cercanas al plano de compresión se tardan más de lo que deberían en transmitirse al resto del cuerpo del metamaterial. Además, al verse obligado a usar una pequeña cantidad de elementos finitos blandos para construir los diseños de metamateriales, se complica la tarea de generar formas redondeadas, puntiagudas o diagonales.

Por otro lado, en ocasiones el sistema implementado suele generar soluciones cuyo comportamiento en el simulador es auxético, pero en la realidad se comportan de manera errática e inestable. Esto evidencia la necesidad de robustecer ciertos algoritmos y consideraciones de diseño dentro del sistema para asegurar la completa factibilidad de los diseños entregados.

Gran parte de las falencias actuales del sistema pueden ser superadas mediante una serie de trabajos propuestos, los que a su vez ayudarían a revelar el verdadero potencial oculto dentro de esta herramienta de diseño. Entre ellos se destacan: En primer lugar, crear modelos CAD de los metamateriales diseñados repitiéndolos numerosas veces tanto a lo largo como a lo ancho y estudiar su comportamiento en la realidad. En segundo lugar, sería útil medir con precisión las deformaciones de los metamateriales fabricados, y comparar su auxeticidad real con otros diseños conocidos en la literatura. Por otro lado, sería extraordinariamente útil lograr que la librería `voxelyze` pueda rastrear el movimiento de más de un *voxel* a la vez, puesto que se abriría un nuevo mundo de posibilidades y aplicaciones complejas, como por ejemplo el diseño evolutivo de mecanismos blandos. Además, se añade la idea de diseñar un algoritmo que sea capaz de convertir topologías imposibles de estudiar en diseños factibles, y así agilizar la evolución del algoritmo genético. Por último, se propone el trabajo de refinar los diseños elaborados por el sistema, de tal manera que estos pasen de verse pixeleados a tener una figura más esbelta y delineada, lo cual podría ayudar a mejorar aún más sus propiedades auxéticas.

Bibliografía

[1] Ion A., Frohnhofen J., Wall L., Kovacs R., Alistar M., Lindsay J., Lopes P., Chen H., and Baudisch. P. Metamaterial mechanisms.

[2] Mark A., Palagi S., Qiu T., and Fischer. P. Auxetic metamaterial simplifies soft robot design. *Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, 2016.

[3] Rafsanjani A., Bertoldi K., and Studart A.R. Programming soft robots with flexible mechanical metamaterials.

[4] School. C. Curva de aprendizaje - diagrama muy útil par los formadores. *https://www.cerem.es/blog/curvas-de-aprendizaje*, 2021.

[5] Goldberg D. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Boston, 1989.

[6] Elipe J.C.A and Lantada. A.D. Comparative study of auxetic geometries by means of computer aided design and engineering. *Smart Materials and Structures*, 2012.

[7] Bertoldi K., Vitelli V., Christensen J., and van Hecke. M. Flexible mechanical metamaterials. *Nat. Rev. Mater*, 2017.

[8] Borovinšek M., Novak N., Vesenjak M., Ren Z., and Ulbin. M. Designing 2d auxetic structures using multi-objective topology optimization. *Materials Science and Engineering: A*, 2020.

[9] Wall. M. Galib: A c++ library of genetic algorithm components. *Leaders for Manufacturing Program*, 1996.

[10] Cheney N., MacCurdy R., Clune J., and Lipson. H. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. *Conference Paper*, 2013.

[11] Victer Paul P., Ramalingam A., Baskaran R., Dhavachelvan P., Vivekanandan K., Subramanian R., and Venkatachalapathy V.S.K. Performance analyses on population seeding techniques for genetic algorithms. *International Journal of Engineering and Technology (IJET)*, 2013.

[12] Sites.google.com. Voxcad. *https://sites.google.com/site/voxcadproject/*.

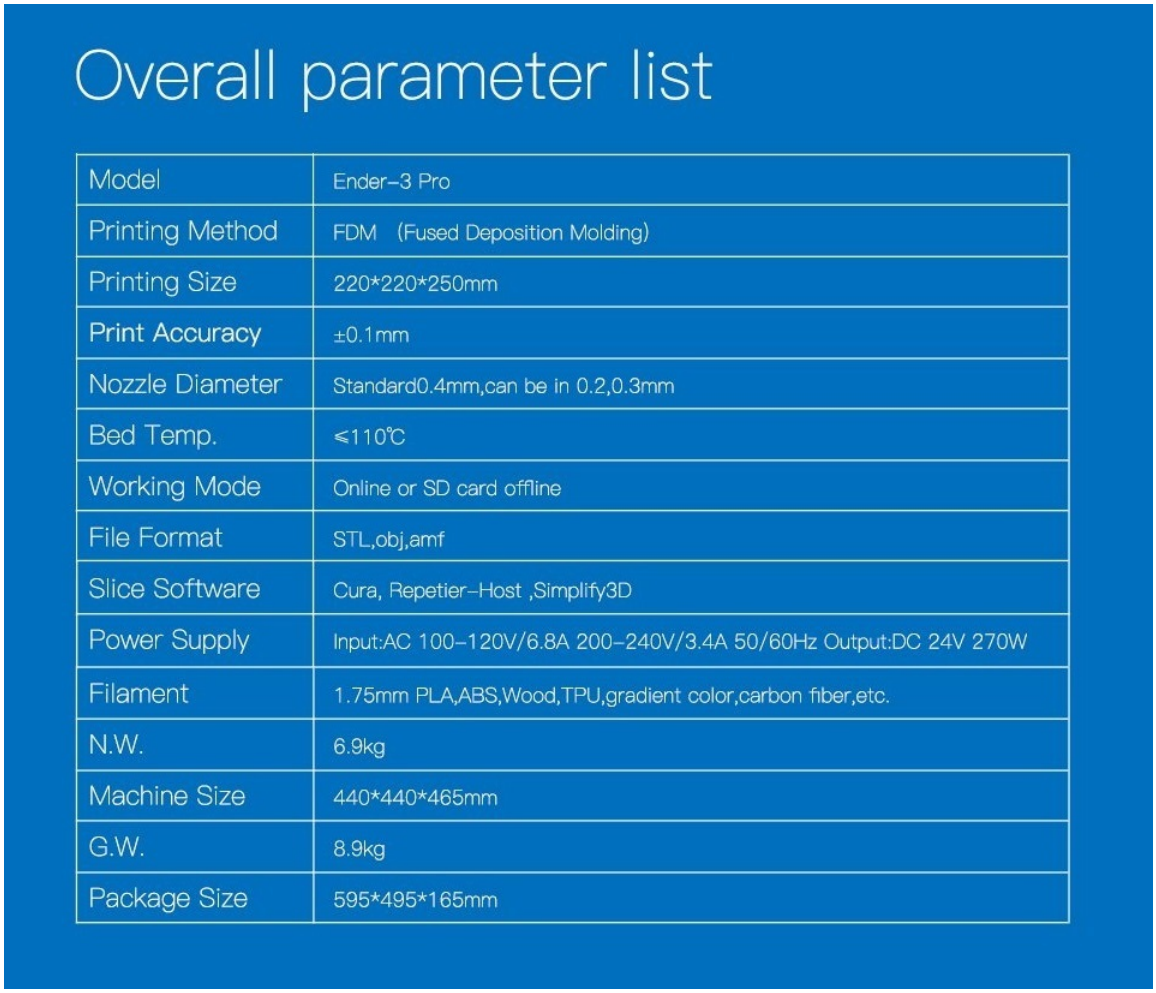
[13] Creality3D Store®. Official store for creality 3d printers and accessories. *https://www.creality3dofficial.com/es/collections/ender-series/products/creality-ender-3-pro-3d-printer?sca_ref=10788.CFDqhuwm3C*.

[14] Ubunlog. Wsl: Cómo instalar y usar el susbistema ubuntu en windows 10. *https://ubunlog.com/wsl-como-instalar-y-usar-el-susbistema-ubuntu-en-windows-10/*.

Anexos

Apéndice A

Especificaciones Técnicas de la Impresora Ender 3 Pro



Overall parameter list

Model	Ender-3 Pro
Printing Method	FDM (Fused Deposition Molding)
Printing Size	220*220*250mm
Print Accuracy	±0.1mm
Nozzle Diameter	Standard0.4mm,can be in 0.2,0.3mm
Bed Temp.	⩽110℃
Working Mode	Online or SD card offline
File Format	STL,obj,amf
Slice Software	Cura, Repetier-Host ,Simplify3D
Power Supply	Input:AC 100-120V/6.8A 200-240V/3.4A 50/60Hz Output:DC 24V 270W
Filament	1.75mm PLA,ABS,Wood,TPU,gradient color,carbon fiber,etc.
N.W.	6.9kg
Machine Size	440*440*465mm
G.W.	8.9kg
Package Size	595*495*165mm

Figura A.1: Especificaciones técnicas de la impresora *Ender 3 Pro* [13].

Figura A.2: Apariencia de la impresora *Ender 3 Pro*.Especificaciones técnicas de la impresora *Ender 3 Pro*[13].

Apéndice B

Especificaciones técnicas del TPU

Ficha de datos técnicos TPU 95A

Ultimaker

Denominación química	Poliuretano termoplástico
Descripción	El filamento de TPU 95A es muy versátil para aplicaciones industriales y es la opción idónea para una amplia gama de proyectos de fabricación que requieren tanto las cualidades del caucho como las del plástico. El TPU 95A se ha diseñado para ofrecer homogeneidad en la impresión 3D y es un filamento semiflexible y resistente a los productos químicos con una fuerte adhesión entre capas. Además, es más fácil y rápido de imprimir que otros filamentos de TPU.
Características principales	Resistencia excepcional al deterioro por uso, alta resistencia a los impactos, dureza Shore A de 95, hasta un 580 % de alargamiento a la rotura y buena resistencia a la corrosión causada por muchos productos químicos y aceites industriales habituales.
Aplicaciones	Prototipado funcional, empuñaduras, guías, bisagras, manguitos, piezas de encaje a presión y carcasas protectoras.
No adecuado para	Aplicaciones en contacto con alimentos e in vivo. Inmersión prolongada en radiación UV y/o humedad y aplicaciones en las que la parte impresa está expuesta a temperaturas superiores a 100 ˚C.

Especificaciones del filamento	Valor	Método
Diámetro	2,90 ± 0,13 mm	Medidor láser de 2 ejes
Desviación de redondez máxima	0,07 mm	Medidor láser de 2 ejes
Peso neto del filamento	750 g	-
Longitud del filamento	~96 m	-

Información sobre el color	Color	Código de color
	TPU 95A blanco	RAL 9010
	TPU 95A negro	RAL 9005
	TPU 95A rojo	RAL 3031
	TPU 95A azul	RAL 5002

Propiedades mecánicas (*)	Moldeo por inyección		Impresión 3D	
	Valor típico	**Método de ensayo**	**Valor típico**	**Método de ensayo**
Módulo de elasticidad a la tracción	-	-	26,0 MPa	ASTM D638
Esfuerzo de tracción a la deformación	-	-	8,6 MPa	ASTM D638
Esfuerzo de tracción a la rotura	-	-	39,0 MPa	ASTM D638
Alargamiento a la deformación	-	-	55,0 %	ASTM D638
Alargamiento a la rotura	-	-	580,0 %	ASTM D638
Resistencia a la flexión	-	-	4,3 MPa	ISO 179
Módulo de flexión	-	-	78,7 MPa	ISO 179
Resistencia a la prueba de impacto Izod, con mella (a 23 ˚C)	-	-	34,4 kJ/m^2	ISO 180
Resistencia a la prueba de impacto Charpy (a 23 ˚C)	-	-	-	-
Dureza	-	-	95 (Shore A) 46 (Shore D)	ASTM D2240 Durómetro
Resistencia a la abrasión	-	-	0,06 g	ASTM D4060 (pérdida de masa, 10 000 ciclos)

Propiedades térmicas	Valor típico	Método de ensayo
Índice de fluidez (MFR)	15,9 g/10 min	ISO 1133 (225 ˚C, 1,2 kg)
Deformación térmica (HDT) a 0,455 MPa	74 ˚C	ASTM D648
Deformación térmica (HDT) a 1,82 MPa	49 ˚C	ASTM D648
Transición vítrea	-24 ˚C	DSC
Coeficiente de expansión térmica	$100 \cdot 10^{-6}$ ˚C^{-1}	ASTM E693
Temperatura de fusión	220 ˚C	DSC
Contracción térmica	-	-

Propiedades eléctricas	Valor típico	Método de ensayo
Resistividad de volumen	10^{11} Ω·m	IEC 60093
Resistencia superficial	$2 \cdot 10^{14}$ Ω	IEC 60093

(*) Ver las notas.

Otras propiedades	Valor típico	Método de ensayo
Gravedad específica	1,22	ASTM D782
Clasificación de llama	Clase HB	ICE 60695-11-10
Absorción de humedad	0,18 %	ASTM D570 (24 h)

Notas

Las propiedades indicadas corresponden a los valores promedio de un lote típico. Las barras para los ensayos de tracción se imprimieron con 2 armazones, flujo de material del 107 %, temperatura de tobera de 260 °C, temperatura del lecho de 45 °C, diámetro de tobera de 0,8 mm, velocidad de relleno de 40 mm/s, velocidad de impresión de 30 mm/s y altura de capa de 0,3 mm. Las barras para los ensayos de flexión e impacto se imprimieron en el plano XY, utilizando el perfil de calidad normal en Cura 2.1, una Ultimaker 2+, una tobera de 0,4 mm, relleno del 90 %, una temperatura de tobera de 235 ˚C y una temperatura de la placa de impresión de 70 ˚C. Los valores son la media de 5 muestras blancas y 5 negras para los ensayos de flexión e impacto. La dureza Shore D se midió en un recuadro de 7 mm de grosor impreso en el plano XY, utilizando el perfil de calidad normal en Cura 2.5, una Ultimaker 3, un núcleo de impresión de 0,4 mm y relleno del 100 %. Ultimaker trabaja constantemente para ampliar la información de las fichas de datos técnicos.

Descargo de responsabilidad

La información o asistencia técnica proporcionadas en esta ficha se facilitan y aceptan por su cuenta y riesgo y Ultimaker y sus filiales no ofrecen ninguna garantía relativa o debida a ellas. Ultimaker y sus filiales no asumen ninguna responsabilidad por el uso de esta información o de ningún producto, método o aparato mencionado y deberá determinar personalmente su idoneidad e integridad para su propio uso, para la protección del medio ambiente y para la salud y la seguridad de sus empleados y los compradores de sus productos. No se ofrece ninguna garantía sobre la capacidad para el comercio o la idoneidad de ningún producto y nada de lo aquí estipulado constituye una renuncia a ninguna de las condiciones de venta de Ultimaker. Las especificaciones están sujetas a modificación sin previo aviso.

Versión Versión 3.010

Fecha 16/05/2017

Apéndice C

Pseudo Códigos

C.1. Parámetros de Entrada

```
//////////////////////////
// PARAMETROS DE ENTRADA //
//////////////////////////
  VoxelSize = 0.001;
  n = 1;

// Bidimensional
  resolution_x = 7;
  resolution_y = 7;
  resolution_z = 4;
// Tridimensional 1
  resolution_x3d = 5;
  resolution_yz3d = 5;
// Tridimensional 2
  resolution_x3d2 = 5;
  resolution_y3d2 = 5;
  resolution_z3d2 = 5;

// Parametros simulacion
  Dt = 0.95;
  SimStopTime = 7;
  SimStopStep = 10000;
  maxForce = -2500;
  E_meta = 3500000000;
  E_placas = 300000000000;

// Features
  isBiased = 0;
  string thatSeed = "seed_1.txt";
  pbias = 0.3;

  isAllosteric = 0;

  symmetryType = 0;

  is3d = 0;
```

C.2. Implementación del GA

```
//////////////////////
// ALGORITMO GENETICO //
//////////////////////
// The allele set is [0,1] and any element of the genome may assume a real
    value [0,1].
  GARealAlleleSet alleles(0, 1);
  GARealGenome genome(length*2, alleles, Objective);

// Parameter list that will be used for the genetic algorithm and genomes.
  GAParameterList params;
  GASteadyStateGA::registerDefaultParameters(params);
  params.set(gaNpopulationSize, 50);
  params.set(gaNscoreFrequency, 1);
  params.set(gaNpMutation, 0.01);
  params.set(gaNpCrossover, 0.8);
  params.set(gaNselectScores, (int)GAStatistics::AllScores);
  params.set(gaNpConvergence, 0.999);
  params.set(gaNnConvergence, 30);
  params.parse(argc, argv, gaFalse);

// Genetic algorithm.
  GASteadyStateGA ga(genome);
  ga.parameters(params);
  ga.terminator(GAGeneticAlgorithm::TerminateUponConvergence);
  ga.set(gaNscoreFilename, "stats.dat");
  ga.initialize();

// Para cada generacion, se guarda la informacion del mejor individuo en pop.
    txt y los fitness max, min y avg en stat.txt
  while (!ga.done())
    {
      reportGA(popfile,statfile,ga,pop,stat);
      indChamp++;
      ga.step();
    }
  reportGA(popfile,statfile,ga2,pop,stat);

// Se imprime al mejor individuo y las estadisticas de la evolucion del
    algoritmo genetico.
  cout << "the ga generated:\n" << ga.statistics().bestIndividual() << endl;
  cout << endl << "GA stats: " << ga.statistics() << endl;
```

C.3. Función Objetivo

```
//////////////////////
// Funcion_objetivo //
//////////////////////
float
Objective(GAGenome& g)
{
  GARealGenome& genome = (GARealGenome&)g;

// Separacion del genoma
  std::vector<float> ContinuousArray (length,0.0);
  std::vector<float> PassiveStiffArray (length,0.0);

  for(int i=0; i<length; i++){
    ContinuousArray[i] = genome.gene(i);
    PassiveStiffArray[i] = genome.gene(i+length);
  }

  string individualID = "test";

// Se obtiene archivo de simulacion
  writeVoxelyzeFile(ContinuousArray, PassiveStiffArray, individualID);

  cout << "Staring Eval for individual " << indNum << endl;

// Se llama a la libreria voxelyze para ejecutar el archivo de simulacion
  std::ostringstream callVoxleyzeCmd;
  callVoxleyzeCmd << "time ../../Voxel_Sim_NoGui/voxelize -f " << "test" << "
      _genome.vxa";

  std::system(callVoxleyzeCmd.str().c_str());

// Se rescatan los resultados de la simulacion
  std::ifstream fitfile;
  fitfile.open("test_fitness.xml");
  std::string line;
  string poisson;
  string compression;
  string outterVox_i;
  string outterVox_f;

// Se calcula d1 y d2
  float d1 = foutterVox_i/(resolution_y*VoxelSize);
  float d2 = ((resolution_y*VoxelSize)-foutterVox_f)/(resolution_y*VoxelSize);

// Se imprime informacion interesante
  cout << "Modulo de Poisson: " << fpoisson << endl;
  cout << "Porcentaje de reduccion: " << 100*((2*fcompression)/((num_x_voxels
      -4)*VoxelSize)) << "%" << " (" << fcompression*1000 << "[mm])" << endl;

// Se indentifican modos de falla, a los cuales se les penalizara el fitness
  if(poisson == "-nan"){
    cout << "Ha ocurrido un error en la simulacion" << endl;
    fitness = 0.0;
  }
```

```
  else if(fail == 1){
    cout << "Metamaterial generado no es valido" << endl;
    fitness = 0.0;
    noValid++;
  }

  else if(fullStripes != 0){
    cout << "Metamaterial generado tiene filas o columnas completamente llenas
        o vacias" << endl;
    fitness = 1/(1+d1+d2);
    if(fpoisson >= 0){
      fitness *= 0.5;
    }
    else{
      fitness = 0.5;
    }
  }

  else{
    fitness = 1/(1+d1+d2);
  }

// Se imprime y retorna el fitness
  cout << "fitness: " << fitness << endl << endl;

  return fitness;
}
```

C.4. Función writeVoxelyzeFile

```
////////////////////////////////
// Funcion_writeVoxelyzeFile    //
// from SoftbotsExperiment.cpp: //
////////////////////////////////
int writeVoxelyzeFile(std::vector<float> ContinuousArray, std::vector<float>
    PassiveStiffArray, string individualID) {

// Se crea el arreglo que define al metamaterial y se define el voxel cuya
    posicion transversal sera monitoreada
  std::vector<int> ArrayForVoxelyze = createArrayForVoxelyze(ContinuousArray,
      PassiveStiffArray);
  voxelToTrack = findVoxelToTrack(ArrayForVoxelyze);

// Se crea un archivo test_genome.vxa que contendra el codigo de simulacion
    definido a continuacion
  std::ofstream myfile;
  std::ostringstream myFileName;
  myFileName << individualID << "_genome.vxa";
  myfile.open (myFileName.str().c_str());
//1000000000
myfile << "\
<?xml␣version=\"1.0\"␣encoding=\"ISO-8859-1\"?>\n\
<VXA␣Version=\"1.0\">\n\
<Simulator>\n\
<Integration>\n\
<Integrator>0</Integrator>\n\
<DtFrac>" << Dt << "</DtFrac>\n\
</Integration>\n\
<Damping>\n\
<BondDampingZ>1</BondDampingZ>\n\
<ColDampingZ>1</ColDampingZ>\n\
<SlowDampingZ>0.015</SlowDampingZ>\n\
</Damping>\n\
<Collisions>\n\
<SelfColEnabled>1</SelfColEnabled>\n\
<ColSystem>3</ColSystem>\n\
<CollisionHorizon>2</CollisionHorizon>\n\
</Collisions>\n\
<Features>\n\
<FluidDampEnabled>0</FluidDampEnabled>\n\
<PoissonKickBackEnabled>0</PoissonKickBackEnabled>\n\
<EnforceLatticeEnabled>0</EnforceLatticeEnabled>\n\
</Features>\n\
<SurfMesh>\n\
<CMesh>\n\
<DrawSmooth>1</DrawSmooth>\n\
<Vertices/>\n\
<Facets/>\n\
<Lines/>\n\
</CMesh>\n\
</SurfMesh>\n\
<SimStop>\n\
<SimStopTime>" << SimStopTime << "</SimStopTime>\n\
<SimStopStep>" << SimStopStep << "</SimStopStep>\n\
</SimStop>\n\
```

```
<Fitness>\n\
<!-- Track a particular voxel -->\n\
<TrackVoxel>" << voxelToTrack << "</TrackVoxel>\n\
<!-- Specify the fitness filename that will be output -->\n\
<FitnessFileNm>" << individualID << "_fitness.xml</FitnessFileNm>\n\
</Fitness>\n\
</Simulator>\n\
<Environment>\n\
<Fixed_Regions>\n\
<NumFixed>0</NumFixed>\n\
</Fixed_Regions>\n\
<Forced_Regions>\n\
<NumForced>2</NumForced>\n\
<FRegion>\n\
<PrimType>0</PrimType>\n\
<X>0.99</X>\n\
<Y>0</Y>\n\
<Z>0</Z>\n\
<dX>0.01</dX>\n\
<dY>1</dY>\n\
<dZ>1</dZ>\n\
<Radius>0</Radius>\n\
<R>0</R>\n\
<G>0</G>\n\
<B>0</B>\n\
<alpha>1</alpha>\n\
<Fixed>0</Fixed>\n\
<ForceX>" << force_1 << "</ForceX>\n\
<ForceY>0</ForceY>\n\
<ForceZ>0</ForceZ>\n\
<DisplaceX>0</DisplaceX>\n\
<DisplaceY>0</DisplaceY>\n\
<DisplaceZ>0</DisplaceZ>\n\
</FRegion>\n\
<FRegion>\n\
<PrimType>0</PrimType>\n\
<X>0</X>\n\
<Y>0</Y>\n\
<Z>0</Z>\n\
<dX>0.01</dX>\n\
<dY>1</dY>\n\
<dZ>1</dZ>\n\
<Radius>0</Radius>\n\
<R>0</R>\n\
<G>0</G>\n\
<B>0</B>\n\
<alpha>1</alpha>\n\
<Fixed>0</Fixed>\n\
<ForceX>" << force_2 << "</ForceX>\n\
<ForceY>0</ForceY>\n\
<ForceZ>0</ForceZ>\n\
<DisplaceX>0</DisplaceX>\n\
<DisplaceY>0</DisplaceY>\n\
<DisplaceZ>0</DisplaceZ>\n\
</FRegion>\n\
</Forced_Regions>\n\
```

```
<Gravity>\n\
<GravEnabled>1</GravEnabled>\n\
<GravAcc>-9.81</GravAcc>\n\
<FloorEnabled>0</FloorEnabled>\n\
</Gravity>\n\
<Thermal>\n\
<TempEnabled>1</TempEnabled>\n\
<TempAmp>25</TempAmp>\n\
<TempBase>25</TempBase>\n\
<VaryTempEnabled>0</VaryTempEnabled>\n\
<TempPeriod>0.1</TempPeriod>\n\
</Thermal>\n\
</Environment>\n\
<VXC␣Version=\"0.93\">\n\
<Lattice>\n\
<Lattice_Dim>" << VoxelSize << "</Lattice_Dim>\n\
<X_Dim_Adj>1</X_Dim_Adj>\n\
<Y_Dim_Adj>1</Y_Dim_Adj>\n\
<Z_Dim_Adj>1</Z_Dim_Adj>\n\
<X_Line_Offset>0</X_Line_Offset>\n\
<Y_Line_Offset>0</Y_Line_Offset>\n\
<X_Layer_Offset>0</X_Layer_Offset>\n\
<Y_Layer_Offset>0</Y_Layer_Offset>\n\
</Lattice>\n\
<Voxel>\n\
<Vox_Name>BOX</Vox_Name>\n\
<X_Squeeze>1</X_Squeeze>\n\
<Y_Squeeze>1</Y_Squeeze>\n\
<Z_Squeeze>1</Z_Squeeze>\n\
</Voxel>\n\
<Palette>\n\
<Material␣ID=\"1\">\n\
<MatType>0</MatType>\n\
<Name>Metamaterial</Name>\n\
<Display>\n\
<Red>0</Red>\n\
<Green>0</Green>\n\
<Blue>0.9</Blue>\n\
<Alpha>0.7</Alpha>\n\
</Display>\n\
<Mechanical>\n\
<MatModel>0</MatModel>\n\
<Elastic_Mod>" << E_meta << "</Elastic_Mod>\n\
<Plastic_Mod>0</Plastic_Mod>\n\
<Yield_Stress>0</Yield_Stress>\n\
<FailModel>0</FailModel>\n\
<Fail_Stress>0</Fail_Stress>\n\
<Fail_Strain>0</Fail_Strain>\n\
<Density>1100000</Density>\n\
<Poissons_Ratio>0.35</Poissons_Ratio>\n\
<CTE>0</CTE>\n\
<uStatic>1</uStatic>\n\
<uDynamic>0.5</uDynamic>\n\
</Mechanical>\n\
</Material>\n\
<Material␣ID=\"2\">\n\
```

```
<MatType>0</MatType>\n\
<Name>Prensa</Name>\n\
<Display>\n\
<Red>0</Red>\n\
<Green>0</Green>\n\
<Blue>0</Blue>\n\
<Alpha>0.5</Alpha>\n\
</Display>\n\
<Mechanical>\n\
<MatModel>0</MatModel>\n\
<Elastic_Mod>" << E_placas << "</Elastic_Mod>\n\
<Plastic_Mod>0</Plastic_Mod>\n\
<Yield_Stress>0</Yield_Stress>\n\
<FailModel>0</FailModel>\n\
<Fail_Stress>0</Fail_Stress>\n\
<Fail_Strain>0</Fail_Strain>\n\
<Density>7850000</Density>\n\
<Poissons_Ratio>0.35</Poissons_Ratio>\n\
<CTE>0</CTE>\n\
<uStatic>1</uStatic>\n\
<uDynamic>0.5</uDynamic>\n\
</Mechanical>\n\
</Material>\n\
</Palette>\n\
<Structure Compression=\"ASCII_READABLE\">\n\
<X_Voxels>" << num_x_voxels << "</X_Voxels>\n\
<Y_Voxels>" << num_y_voxels << "</Y_Voxels>\n\
<Z_Voxels>" << num_z_voxels << "</Z_Voxels>\n\
<Data>\n\
<Layer><![CDATA[";
  int v=0;
  for (int z=0; z<num_z_voxels; z++) {
    for (int y=0; y<num_y_voxels; y++) {
      for (int x=0; x<num_x_voxels; x++) {
        myfile << ArrayForVoxelyze[v];
        md5file << ArrayForVoxelyze[v];
        v++;
      }
    }
    if (z<num_z_voxels-1) {myfile << "]]></Layer>\n<Layer><![CDATA["; md5file
      << "\n";}
  }
  myfile << "]]></Layer>\n\
</Data>\n\
</Structure>\n\
</VXC>\n\
</VXA>";

  myfile.close();

// Se imprime el metamaterial generado
  cout << "Imprimiendo metamaterial " << individualID << indNum << ":\n";

  v = 0;
  for(int j=0; j<num_y_voxels; j++){
    for(int i=0; i<num_x_voxels; i++){
```

```
        if(ArrayForVoxelyze[v] == 2){
        cout << '|';
        }
        cout << (ArrayForVoxelyze[v] == 1 ? '*' : '␣') << "␣";
        v++;
      }
      cout << "\n";
    }
    cout << "\n"; cout.flush();
}
```

C.5. Función createArrayForVoxelyze

```
////////////////////////////////////
// Funcion_createArrayForVoxelyze //
////////////////////////////////////
std::vector<int> createArrayForVoxelyze(std::vector<float> ContinuousArray,
    std::vector<float> PassiveStiffArray) {

  std::vector<int> ArrayForVoxelyze (num_x_voxels*num_y_voxels*num_z_voxels
      ,0.0);

// Se crean placas a ambos extremos del metamaterial
  int v=0;
  for (int z=0; z<num_z_voxels; z++) {
    for (int y=0; y<num_y_voxels; y++) {
      for (int x=0; x<num_x_voxels; x++) {
        if (x==0) {
          ArrayForVoxelyze[v] = 2;
          ArrayForVoxelyze[v+1] = 1;
        }
        if (x==num_x_voxels-1) {
          ArrayForVoxelyze[v] = 2;
          ArrayForVoxelyze[v-1] = 1;
        }
        if (x >= 2 && x <= num_x_voxels-3) {
          ArrayForVoxelyze[v] = 0;
        }
        v++;
      }
    }
  }

// Se genera el metamaterial

////////////////////////
// CASO BIDIMENSIONAL //
////////////////////////
  if(is3d==0){
    for(int i=0; i<resolution_y; i++) {
      for(int j=0; j<resolution_x; j++) {
        for(int k=0; k<num_x_voxels*num_y_voxels; k+=unitCell_y*num_x_voxels)
            {
          for(int l=2; l<num_x_voxels-2; l+=unitCell_x) {
            if(ContinuousArray[i*resolution_x+j] <= PassiveStiffArray[i*
                resolution_x+j]) {
              // CUADRANTE 1
              ArrayForVoxelyze[(k+(i*num_x_voxels))+(l+j)] = 1;
              // CUADRANTE 2
              ArrayForVoxelyze[(k+(i*num_x_voxels))+(l+unitCell_x-1-j)] = 1;}
                  // simetria axial en y
              // CUADRANTE 3
              ArrayForVoxelyze[(k+(num_x_voxels*(unitCell_y-1-i)))+(l+j)] =
                  1;} // simetria axial en x
              // CUADRANTE 4
              ArrayForVoxelyze[(k+(num_x_voxels*(unitCell_y-1-i)))+(l+
                  unitCell_x-1-j)] = 1;} // simetria central
            }
```

```
            }
          }
        }
      }

// Se le da altura al metamaterial
      if(num_z_voxels > 1){
        for(int z=1; z<num_z_voxels; z++) {
          for(int v=0; v<num_x_voxels*num_y_voxels; v++) {
            ArrayForVoxelyze[(z*num_x_voxels*num_y_voxels)+v] = ArrayForVoxelyze
                [v];
          }
        }
      }
    }

// Se usa la funcion makeOneShapeOnly para asegurar la continuidad del
    metamaterial
    ArrayForVoxelyze = makeOneShapeOnly(ArrayForVoxelyze);

// Se checkea la existencia de filas llenas o vacias
    checkFullStripes(ArrayForVoxelyze);

    return ArrayForVoxelyze;
}
```

C.6. Función checkFullStripes

```
////////////////////////////
// Funcion_checkFullStripes //
////////////////////////////
int checkFullStripes(std::vector<int> Array) {

  fullStripes = 0;
  int fullness;

  if(is3d==0){
// Se verifica la existencia de filas completamente llenas o vacias
    for(int j=0; j<resolution_y; j++){
      for(int i=0; i<=num_x_voxels-4-unitCell_x; i++){
        fullness = 0;
        for(int v=0; v<unitCell_x; v++){
          fullness += Array[(j*num_x_voxels)+i+2+v];
        }
        if(fullness==0 || fullness==unitCell_x){
          fullStripes++;
          i+=num_x_voxels;
        }
      }
    }

// Se verifica la existencia de columnas completamente llenas
    for(int i=0; i<resolution_x; i++){
      if(Array[i+2] != 0){
        fullness=0;
        for(int v=0; v<resolution_y; v++){
          fullness += Array[(v*num_x_voxels)+i+2];
        }
        if(fullness == resolution_y){
          fullStripes++;
        }
      }
    }
  }
```

C.7. Función findVoxelToTrack

```
////////////////////////////
// Funcion_findvoxeltotrack //
////////////////////////////
int findVoxelToTrack(std::vector<int> ArrayForVoxelyze) {

  int voxel_pos = -1;
  int stop = 0;
  int i = 0;

  if(ArrayForVoxelyze[num_x_voxels-1]==0){fail=1;}

  else{

  if(is3d==0){
    while(stop == 0){
      if(ArrayForVoxelyze[i] != 0){
        voxel_pos++;
      }
      if(i %num_x_voxels == num_x_voxels/2){
        stop = ArrayForVoxelyze[i];
      }
      if(i/num_x_voxels == resolution_y){
        stop = 1;
        fail = 1;
      }
      i++;
    }
  }
```

C.8. Seeding

```
  for(int i=0; i<length; i++){
    if(isBiased==0){
      ContinuousArray[i] = genome.gene(i);
      PassiveStiffArray[i] = genome.gene(i+length);
    }

    else if(isBiased==1 && is3d==0 && bias[i]==0){
      ContinuousArray[i] = (1+pbias)*genome.gene(i);
      PassiveStiffArray[i] = (1-pbias)*genome.gene(i+length);
    }

    else if(isBiased==1 && is3d==0 && bias[i]==1){
      ContinuousArray[i] = (1-pbias)*genome.gene(i);
      PassiveStiffArray[i] = (1+pbias)*genome.gene(i+length);
    }
  }
```

C.9. Simulación Alostérica

```
// Fitness version alosterico
  if(isAllosteric==1 && poisson != "-nan" && fail != 1 && fullStripes == 0){
    fitness = 1 - fitness;
  }
```

C.10. Tipos de Simetrías

```
//////////////////////
// CASO BIDIMENSIONAL //
//////////////////////
  if(is3d==0){
    for(int i=0; i<resolution_y; i++) {
      for(int j=0; j<resolution_x; j++) {
        for(int k=0; k<num_x_voxels*num_y_voxels; k+=unitCell_y*num_x_voxels)
            {
          for(int l=2; l<num_x_voxels-2; l+=unitCell_x) {
            if(ContinuousArray[i*resolution_x+j] <= PassiveStiffArray[i*
                resolution_x+j]) {
              // CUADRANTE 1
              ArrayForVoxelyze[(k+(i*num_x_voxels))+(l+j)] = 1;
              // CUADRANTE 2
              if(symmetryType==0 || symmetryType==2){ArrayForVoxelyze[(k+(i*
                  num_x_voxels))+(l+unitCell_x-1-j)] = 1;} // simetria axial en
                   y
              else if(symmetryType==3){ArrayForVoxelyze[(k+(j*num_x_voxels))+(
                  l+unitCell_x-1-i)] = 1;} // rotacion 90 grados
              else if(symmetryType==5){ArrayForVoxelyze[(k+(num_x_voxels*(
                  resolution_y-1-i)))+(l+unitCell_x-1-j)] = 1;} // simetria
                  central en y
              else{ArrayForVoxelyze[(k+(i*num_x_voxels))+(l+resolution_x+j)] =
                   1;} // desplazamiento sin simetria
              // CUADRANTE 3
              if(symmetryType==0 || symmetryType==1){ArrayForVoxelyze[(k+(
                  num_x_voxels*(unitCell_y-1-i)))+(l+j)] = 1;} // simetria
                  axial en x
              else if(symmetryType==3){ArrayForVoxelyze[(k+(num_x_voxels*(
                  unitCell_y-1-j)))+(l+i)] = 1;} // rotacion 270 grados
              else if(symmetryType==4){ArrayForVoxelyze[(k+(num_x_voxels*(
                  unitCell_y-1-i)))+(l+resolution_x-1-j)] = 1;} // simetria
                  central en x
              else{ArrayForVoxelyze[(k+(num_x_voxels*(resolution_y+i)))+(l+j)]
                   = 1;} // desplazamiento sin simetria
              // CUADRANTE 4
              if(symmetryType==0 || symmetryType==3 || symmetryType==4 ||
                  symmetryType==5){ArrayForVoxelyze[(k+(num_x_voxels*(
                  unitCell_y-1-i)))+(l+unitCell_x-1-j)] = 1;} // simetria
                  central
              else if(symmetryType==1){ArrayForVoxelyze[(k+(num_x_voxels*(
                  unitCell_y-1-i)))+(l+resolution_x+j)] = 1;} // simetria axial
                   en x
              else if(symmetryType==2){ArrayForVoxelyze[(k+(num_x_voxels*(
                  resolution_y+i)))+(l+unitCell_x-1-j)] = 1;} // simetria axial
                   en y
```

```
                else{ArrayForVoxelyze[(k+(num_x_voxels*(resolution_y+i)))+(l+
                    resolution_x+j)] = 1;} // desplazamiento sin simetria
            }
        }
    }
}
```