UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Ingeniería Mecánica y de Materiales

Trabajo de Fin de Máster:

Integración y paralelización de algoritmos de optimización estructural

Autor:
David Muñoz Pellicer

Dirigido por:
Juan José Ródenas García
José Albelda Vitoria

Fecha:
Septiembre 2018

Agradecimientos

Quiero darle las gracias a mis tutores, Juanjo y Pepe, por haberme ayudado todo lo posible durante la realización de este trabajo y por involucrarse mucho en su desarrollo.
También quiero agradecer a Fede, por haberme ayudado a montar la comunicación entre ordenadores y por darme tan valiosos consejos.

Abstract

The structural optimization processes involve high computational costs and, in many situations, the accomplishment of several sequential shape optimizations (first a topological optimization to define the layout of the problem and then a shape optimization to define completely the contour of the mechanical component). In order to carry out this task, different computational tools are used. Those tools were developed in different times and platforms. This is why the implementation of a work environment that allows to integrate those tools and eases the transfer of information between them is needed. In reference to the problem's solution, its highl demand of computational cost would be reduced through the introduction of surrogate models and new tools that allow to distribute in parallel certain tasks to several computers. This would make it possible to solve, in a reasonable time, structural optimization problems that are nowadays impossible to be solved in just one computer. The work proposed will consist in the development of this optimization integrated environment, that allows to define and control complex sequences of optimization, and to include the computational tools available to reduce the computational cost and parallelization tools adapted to the hardware available at the Department of Mechanical Engineering.

Resumen

Los procesos de optimización estructural involucran un coste computacional muy elevado y en muchas ocasiones la realización de varias optimizaciones de forma secuencial (primero una optimización topológica para definir la parametrización adecuada del problema y una optimización de forma posterior para definir completamente el componente mecánico). Para realizar estas tareas suelen utilizarse diferentes herramientas computacionales que se han desarrollado en diferentes momentos y plataformas. Esto justifica el desarrollo de un entorno de trabajo que permita integrar estas herramientas y facilite la transferencia de información entre ellas. En lo referente a la solución del problema asociado al alto coste computacional en el entorno de trabajo se quiere introducir, tanto los modelos subrogados desarrollados previamente en el departamento, como nuevas herramientas que permitan distribuir en paralelo ciertas tareas de cálculo entre diferentes ordenadores. Esto supondría poder abordar en un tiempo razonable problemas de optimización estructural que actualmente no se pueden resolver con una única máquina. El trabajo propuesto consistirá en el desarrollo de dicho entorno integrado de optimización, que permita definir y controlar secuencias complejas de optimización, y que incluya las herramientas computacionales disponibles para reducir el coste computacional y herramientas de paralelización adaptadas al *hardware* disponible en el Departamento de Ingeniería Mecánica.

Resum

Els processos d'optimització estructural involucren un cost computacional molt elevat i moltes vegades la realització de diverses optimitzacions de forma seqüencial (primer una optimització topològica per a definir la parametrización adequada del problema i una optimització de forma posterior per a definir completament el component mecànic) . Per a realitzar aquestes tasques solen utilitzar-se diferents eïnes computacionals que s'han desenvolupat en diferents moments i plataformes. Açò justifica el desenvolupament d'un entorn de treball que permitisca integrar aquestes eïnes i facilite la transferència d'informació entre elles. Pel que fa a la solució del problema associat a l'alt cost computacional en l'entorn de treball es vol introduir, tant els models subrogats desenvolupats prèviament en el departament, com noves eïnes que permitisquen distribuir en paral·lel certes tasques de càlcul entre diferents ordinadors. Açò suposaria poder abordar en un temps raonable problemes d'optimització estructural que actualment no es poden resoldre amb una única màquina. El treball proposat consistirà en el desenvolupament de l'anterior entorn integrat d'optimització, que permitisca definir i controlar seqüències complexes de càlculs d'optimització, i que incloga les eïnes computacionals disponibles per a reduir el cost computacional i eines de paral·lelització adaptades als ordinadors disponibles en el Departament d'Enginyeria Mecànica.

Índice general

1 Introducción

El objetivo de este trabajo ha sido el de desarrollar una nueva metodología de análisis en paralelo que aprovechase los recursos informáticos del Área de Ingeniería Mecánica del Departamento de Ingeniería Mecánica y Materiales de la UPV e implementarla en un entorno de análisis programado en Matlab, orientado principalmente a su utilización en problemas de optimización de forma estructural.

En este capítulo se presentarán brevemente los principales conceptos en los que se basa el desarrollo de la tesis, que son:

- Optimización.
  - Algoritmos Estocásticos.
  - Algoritmos Deterministas.
- Modelos Subrogados.
- Hibridación.
- Método de los elementos finitos mediante mallas cartesianas.
- Paralelización

1.1. Optimización

La optimización es el campo de conocimiento en el que se busca y selecciona el mejor entre un conjunto determinado de elementos. Por lo tanto, los algoritmos de optimización son los pasos que se deben seguir hasta encontrarlo. Un problema de optimización estándar consiste en maximizar o minimizar una función dada, modificando sistemáticamente las variables de diseño que definen el problema. A su vez, éste puede estar sometido a una serie de restricciones que dependen también de las variables de diseño. A continuación, se representa un ejemplo genérico de problema de minimización monoobjetivo:

$$\begin{aligned} & \min_x f(x) \\ \textit{sujeto a:} \quad & \\ & g_i(x) \leq 0 \\ & h_i(x) = 0 \\ & x_{min} \leq x \leq x_{max} \end{aligned}$$

La optimización que se verá en este trabajo estará orientada a los componentes mecánicos y estructurales sometidos a una serie de cargas y restricciones. Este sistema estará enmarcado dentro de la elasticidad lineal por lo tanto, las magnitudes con las que se podrán trabajar serán: tensiones, deformaciones, desplazamientos, etc. Las siguientes funciones y variables están siempre presentes en un problema de optimización estructural [1]:

- *Función objetivo (f)*: Esta función es usada para clasificar los diseños. Para cada posible diseño f devuelve un escalar que indica su bondad. Habitualmente se trata de buscar los valores más pequeños de *f*, puesto que el problema más usual es el de minimización, que es el que se utilizará en todo este trabajo. Mediante modificaciones simples de esta función, un problema de maximización puede convertirse en otro de minimización, por lo que este planteamiento es completamente general.
- *Variable de diseño (x)*: Un vector de escalares que describe el diseño, y que debe ser modificado a lo largo de la optimización. Puede definir tanto el contorno de una superficie, como las densidades relativas de distintas zonas del diseño o la sección de barras.
- *Variable de estado (y)*: Para una estructura determinada, i.e., para un diseño *x*, *y* es una función que representa la respuesta de la estructura. Para un diseño mecánico, puede ser desplazamiento, tensiones, deformaciones o fuerzas.

Un problema de optimización estructural ($\mathbb{OE}$) en su forma genérica se representa tal que:

$$(\mathbb{OE}) = \left| \begin{array}{ll} \textit{minimizar f(x) respecto a x} & \\ \textit{sujeto a} & \left| \begin{array}{l} \textit{Restricciones de comportamiento en y} \\ \textit{Restricciones de diseño en x} \\ \textit{Restricciones de equilibrio} \end{array} \right. \end{array} \right.$$

Es posible que se piense en un problema con varias funciones objetivo, llamado *criterio múltiple*, o problema de *optimización multi-objetivo*:

$$minimizar(f_1(x), f_2(x), \ldots, f_i(x)), \tag{1.1}$$

donde i es el número de funciones objetivos y las restricciones son las mismas que en ($\mathbb{OE}$). Habitualmente, cada una de estas funciones no son minimizadas para el mismo x e y. En lugar de esto, se busca la llamada *optimización de Pareto*, que consiste en obtener la frontera de Pareto, constituida por el conjunto de puntos del espacio de trabajo cuyos puntos de alrededor no son mejores respecto de las funciones objetivo. Otro metodo de obtener el diseño x en este tipo de problema es transformar el problema multi-objetivo en uno mono-objetivo y alcanzar una relación de compromiso entre todas las funciones objetivo $f_i(x)$ de forma que se convierte en un f escalar objetivo equivalente:

$$\sum_{i=1}^{n} p_i f_i(x), \tag{1.2}$$

donde $p_i \geq 0, i = 1, \ldots, n$, son los llamados factores de peso que satisfacen $\sum_{i=1}^{n} p_i = 1$. La modificación de los pesos conduce a otra solución diferente (otro punto de la frontera de Pareto).
En este trabajo solo se considerarán problemas de optimización estructural de la misma forma que ($\mathbb{OE}$), es decir, se estudiarán problemas mono-objetivo.
Tres tipos de restricciones son consideradas en ($\mathbb{OE}$): (1) *Restricciones de comportamiento* son las referentes a la variable de estado y. Normalmente se escriben como $g_i(x) \leq 0$, donde g es una función que represente, por ejemplo, la tensión de Von Mises. (2) *Restricciones de diseño* involucran a las variables de diseño, pueden ser restricciones tipo caja o también aquellas que representan la relación entre las propias variables de diseño. Finalmente, en un problema discreto que además es lineal, las *restricciones de equilibrio* son:

$$\mathbf{K}(x)\mathbf{u} = \mathbf{F}(x), \tag{1.3}$$

donde $\mathbf{K}$(x) es la matriz de rigidez de la estructura, que generalmente es función del diseño, $\mathbf{u}$ es el vector de desplazamientos y $\mathbf{F}$(x) es el vector de fuerzas, que pueden depender también del diseño. En la formulación definida en ($\mathbb{OE}$) se toman x y y como variables independientes, siendo esta la llamada *formulación simultánea*. Sin embargo, puede darse el caso que y depende únicamente de x, si se trata $\mathbf{u}$(x) como un función dada, la restricción de equilibrio se puede eliminar de ($\mathbb{OE}$) y la variable de estado y puede ser sustituida por esta, dando como resultado:

$$(\mathbb{OE})_{fa} = \begin{cases} \min\limits_{x} & f(x, \mathbf{u}(x)) \\ s.a & g_i(x, \mathbf{u}(x)) \leq 0, \end{cases}$$

donde $s.a$ denota *"sujeto a"* y se asume que todas las restricciones de estado pueden ser expresadas como $g_i(x, \mathbf{u}(x)) \leq 0$. Esta formulación es la llamada *formulación*

anidada y será el punto de partida para los métodos numéricos vistos en este trabajo. Cuando se trata $(\mathbb{OE})_{fa}$ numéricamente, es posible que se necesiten las derivadas de f y de g respecto a x. Para encontrar tales derivadas se usará el *análisis de sensibilidades* o algún método de aproximación, puesto que el cálculo de éstas no es trivial, debido a que $\mathbf{u}$(x) esta definido implícitamente.
Dependiendo de la característica geométrica que x represente, los problemas de optimización estructural se dividen en tres diferentes clases:

- *Optimización de tamaño*: cuando x es alguna dimensión asociada a la definición de las secciones en estructuras de barras, o la distribución de espesores en unas láminas. Un problema de optimización de tamaño de una estructura de barras puede verse en la Figura 1.1.

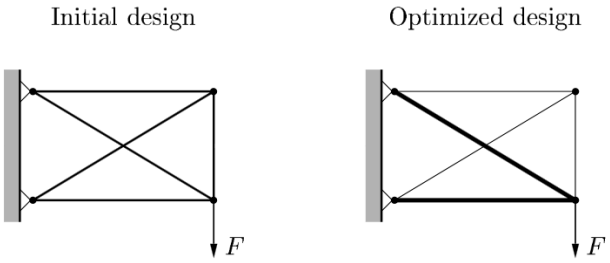


Figura 1.1: Problema de optimización de tamaño.

- *Optimización de forma*: en este caso x define la forma del contorno de alguna parte de la frontera del dominio estructural. La optimización consiste en seleccionar el dominio de integración para las ecuaciones diferenciales de un modo óptimo. Cabe destacar, que no se modifica la topología del dominio, es decir, no se crearán nuevas fronteras. Un problema de optimización de forma en dos dimensiones aparece en la Figura 1.2.

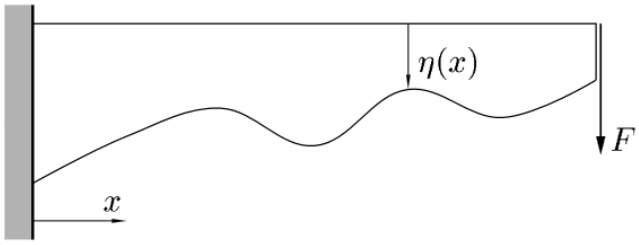


Figura 1.2: Problema de optimización de forma.

- *Optimización topológica*: este es el caso de optimización estructural más general. La configuración más habitual de este problema es considerar que x sea una variable que represente la densidad de los elementos discretos. Pudiendo

ser esa igual a 0, en el caso de que el elemento deba desaparecer, y 1 en caso contrario. En la Figura 1.3 se detalla este tipo de problema.

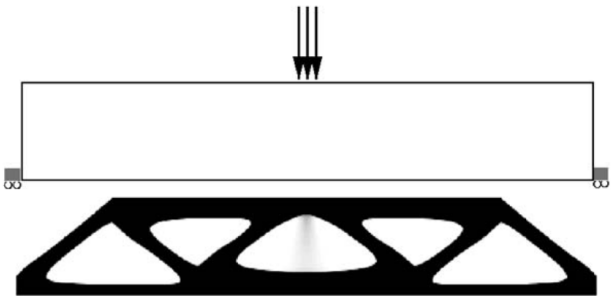


Figura 1.3: Problema de optimización topológica.

Puesto que la mayoría de problemas de optimización estructural son de una escala considerable, la resolución de estos exige la utilización de técnicas numéricas para evaluar la función objetivo como las funciones de restricción. Tanto la obtención de las funciones objetivo f como de las variables de estado y se realizará a través del *Método de los Elementos Finitos (MEF)*, del cual se hablará más adelante. En el caso de la optimización también se utilizarán métodos numéricos para encontrar el mejor diseño, la gran mayoría de estos métodos se resuelven de manera iterativa y se pueden separan en dos grandes grupos, algoritmos estocásticos y algoritmos deterministas.

- *Algoritmos estocásticos* [2]: Habitualmente estos solo necesitan el valor de la función objetivo y el de las restricciones para poder continuar el proceso. La principal característica es que de una ejecución a la siguiente el camino recorrido por el proceso iterativo puede ser completamente distinto, puesto que utilizan un procedimiento en cierto grado, aleatorio. Suelen ser algoritmos con convergencia al mínimo global puesto que evalúan un amplio rango del campo de posibles soluciones. Se conocen un gran número de algoritmos estocásticos entre los cuales podemos destacar los siguientes:

  - *Algoritmos evolutivos*[3], como el genético. Estos algoritmos simulan la evolución de una población de individuos sometiéndola a agentes aleatorios, como por ejemplo, mutaciones y recombinación genética.
  - *Ascensión de montañas*, se basa en variar incrementalmente la solución actual con el fin de encontrar una mejor.
  - *Algoritmo de recocido simulado*[4], se inspira en el proceso de recocido de aceros, el cual consiste en calentar y luego enfriar lentamente el acero para variar sus propiedades físicas y alcanzar un estado de menor energía. Aplicado a la optimización, se define un vecindario compuesto por los estados que puede alcanzar un individuo en base a un cambio en la conformación del sistema, hecho esto, se evalúa si alguno de estos vecinos es

mejor (menor energía) que el inicial, si es así se evalua un nuevo vecindario.

- *Algoritmo de la colonia de hormigas*[5], surge a partir de la observación de las hormigas cuando estas buscan el camino más corto entre la colonia y una fuente de alimentos. Partiendo de caminos aleatorios, la hormiga que tenga exito en su búsqueda dejará un rastro de feromonas para que el resto de la colonia siga ese camino, pero si es excesivamente largo, esas feromonas se evaporarán a lo largo del tiempo, lo cual impide que el algoritmo converja en un mínimo local.

- *Algoritmos deterministas*: Usualmente para poder aplicar estos se necesita de información adicional para que funcionen adecuadamente, como puede ser el gradiente de las funciones objetivo, de las restricciones, o incluso el Hessiano de ambas. Debido a esta característica se definen como deterministas puesto que se puede repetir con exactitud un proceso de optimización. Cabe destacar los siguiente algoritmos:

  - *Método de Newton*[6], su funcionamiento se basa en la *expansión de Taylor* de segundo grado de la función objetivo. Al derivar e igualar a cero, resulta que el incremento que debe llevarse a cabo en las variables de diseño depende del gradiente y del hessiano de la función objetivo.
  - *Método de Quasi-Newton*[7], a diferencia del anterior, no es necesario evaluar el hessiano, éste se actualiza en base al análisis de una sucesión del vector gradiente.
  - *Método del descenso de gradiente*[8], la actualización de variables en este algoritmo se hace considerando directamente el gradiente de la función objetivo, siendo el próximo individuo, el actual sumado el gradiente por un término que representa la magnitud del incremento llevado a cabo.

  Es necesario añadir una serie de algoritmos, utilizados mayoritariamente para problemas no convexos en los cuales para cada iteración generan a partir del probleam original uno convexo y lo minimizan. De esta forma el elemento óptimo del subproblema será el siguiente elemento a evaluar en el problema original. Entre los más destacas aparecen los siguientes:

  - *Sequential Linear Programming (SLP)* [9], en este caso se resuelven una serie de aproximaciones de primer orden del modelo inicial.
  - *Sequential Quadratic Programming (SQP)* [9], igual que el anterior pero en este caso se crea un problema cuadrático convexo.
  - *Method of Moving Asymptotes (MMA)* [10] , a cada iteración se minimiza un problema aproximado en cual las funciones han sido remplazadas por ciertas funciones convexas definidas por unas asíntotas laterales.

1.1.1. Hibridación o ejecución secuencial de diferentes algoritmos

Con el fin de obtener el mayor rendimiento posible de los algoritmos de optimización detallados en apartados anteriores, se ha propuesto un método de hibridación. Este método hace referencia a la posibilidad de concatenar varios algoritmos entre sí, independientemente de su naturaleza, de esta forma se aprovecharán las ventajas de cada uno de los algoritmos. El ejemplo más básico y funcional podría ser comenzar el proceso de optimización mediante un algoritmo estocástico, como por ejemplo el genético y una vez se haya encontrado una zona en la que posiblemente se encuentre el mínimo global se puede lanzar un algoritmo basado en gradiente, ver Figura 1.4. Los beneficios de esta ejecución serían por un lado, aprovechar el barrido del espacio de soluciones que realiza el algoritmo genético con la velocidad de convergencia del algoritmo basado en gradiente.

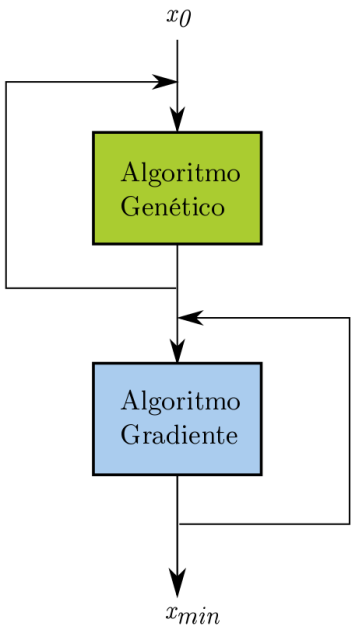


Figura 1.4: Ejemplo de hibridación de un algoritmo genético y uno basado en gradiente.

1.2. Modelos subrogados

Un modelo subrogado es un modelo ajustado a un número de puntos y se emplea en el campo de la ingeniería cuando un resultado de interés no puede ser medido directamente, porque la obtención de éste se prolonga excesivamente en el tiempo. La mayoría de problemas de diseño en ingeniería requieren de experimentos y/o simulaciones para evaluar el objetivo de diseño y funciones de restricción como

función de las variables de diseño. En el caso que se verá en este trabajo se utilizarán modelos subrogados bien para obtener directamente los resultados del análisis de elementos finitos o bien para obtener una relación entre entrada y salida mediante un estudio paramétrico.
Existe un amplio catálogo de posibles métodos para la definición de un modelo subrogado, como los siguientes:

- *Regresión* [11], en base al ajuste de una función polinómica (habitualmente con el método de los mínimos cudrados) sobre una nube de puntos, se crea un modelo que permite interpolar la solución de las variables de diseño.
- *Máquinas de vector soporte* [12], dado un subconjunto de puntos clasificados en una serie de categorías, la MVS construye un modelo que se encarga de predecir a que categoría pertenecerá un nuevo punto. Para ello, busca un hiperplano que separe de forma óptima todas las catergorías existentes.
- *Redes Neuronales Artificiales*[13], se seleccionarán éstas últimas para su implementación en el programa, por lo tanto se detallarán en profundidad a continuación.

Las **Redes Neuronal Artificiales (RNA)** son un artificio matemático vagamente inspirado en las redes neuronales biológicas que constituyen el cerebro de los animales. Una RNA esta basada en una colección de nodos conectados entre sí, llamados **neuronas artificiales**. Las neuronas pueden transmitir señales de unas a otras, en las aplicaciones más comunes, estas conexiones son un número real, pero la salida de cada una de estas neuronas es calculada por un función no-lineal de la suma de las entradas.
Estas neuronas se agrupan entre sí dando como resultado lo que se conoce como **capa**. El conjunto de capas forma la red neuronal propiamente dicha, ver Figura 1.5. Habitualmente se distinguen tres tipos de capas: la capa de entrada, que tiene tantas neuronas como variables de diseño nuestro problemas, las capas ocultas que son las encargadas de darle complejidad a nuestra solución, puede haber tantas y con tantas neuronas como uno desee, y por último, la capa de salida que tiene tantas neuronas como resultados esperemos obtener.

1.2.1. Componentes de una red neuronal artificial

Neuronas
Una neurona cualquiera $\mathbf{j}$ recibiendo una entrada $\mathbf{p_j}(t)$ de la neurona predecesora estaría formada por:

- una *activación* $\mathbf{a_j}(t)$, dependiente de un parámetro discreto de tiempo,
- una *función de activación* $\mathbf{f}$, que computa la nueva activación a un tiempo $t+1$ a partir de $\mathbf{a_j}(t)$ y de la entrada a la neurona $\mathbf{p_j}(t)$, siguiendo la siguiente relación $\mathbf{a_j}(t+1) = \mathbf{f}(\mathbf{a_j}(t), \mathbf{p_j}(t))$

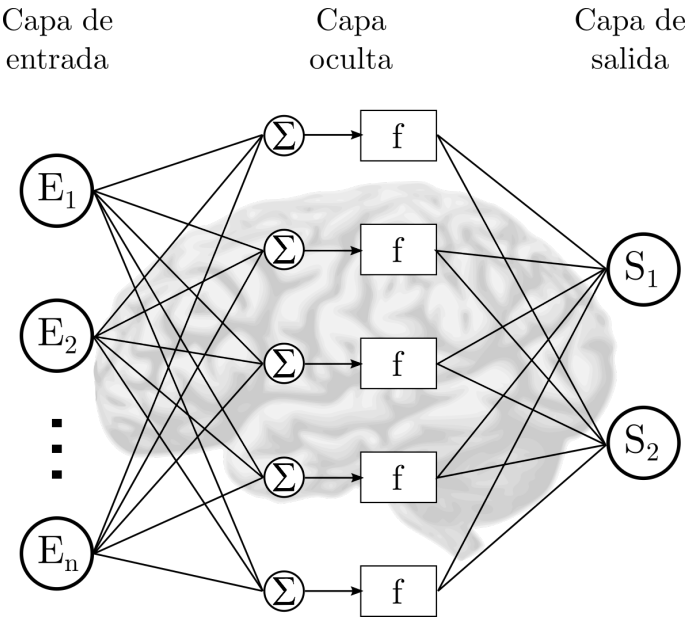


Figura 1.5: Representación de una red neuronal artificial con una capa oculta.

- una *función de salida* $\mathbf{f_{out}}$, que calcula la salida a partir de la activación $\mathbf{o_j}(t) = \mathbf{f_{out}}(\mathbf{a_j}(t))$

Habitualmente la *función de salida* es simplemente la función identidad.

Conexiones y pesos
La red consiste en conexiones, cada conexión transfiere la salida de una neurona $\boldsymbol{i}$ hacia la entrada de la neurona $\boldsymbol{j}$. A cada conexión se le asigna un peso $\mathbf{w_{ij}}$.

Función de propagación
La *función de propagación* calcula la entrada $\mathbf{p_j}(t)$ de una neurona $\boldsymbol{j}$ a partir de la salida $\mathbf{o_i}(t)$ de la neurona predecesora y usualmente tiene la siguiente forma

$$\mathbf{p_j}(t) = \sum_i \mathbf{o_i}(t)\mathbf{w_{ij}} \tag{1.4}$$

Regla de aprendizaje
La *regla de aprendizaje* es una regla o algoritmo que se encarga de modificar ciertos parámetros de la RNA, de modo que se obtenga un resultado más favorable a partir de unas determinadas entradas. Este proceso de aprendizaje habitualmente modifica los pesos $\mathbf{w_{ij}}$ existentes en cada una de las conexiones.

1.3. Método de elementos finitos mediante mallas cartesianas

En cuanto a los análisis numéricos, estos se han realizado a través del método de los elementos finitos. En este trabajo, se han utilizado tanto programas comerciales de *MEF*, por ejemplo *ANSYS*, como *software* desarrollado íntegramente en el Centro de Investigación en Ingeniería Mecánica (*CIIM*), bautizado como *cgFEM*.
El método de los elementos finitos mediante mallas cartesianas, *cgFEM* [14], requiere una apropiada combinación de diferentes técnicas para conseguir nuestros objetivos. La principal característica de *cgFEM* es que trata de suprimir la tarea de mallado, que es una de las que más recursos consume el método tradicional de mallado conforme a geometría del método de los elementos finitos (MEF).
La malla usada para resolver el problema de MEF será independiente de la geometría del componente a ser analizado. En el marco de *cgFEM* se tienen dos dominios, el dominio del problema Ω y el dominio de la malla Ω_E, que es cuadrado (cúbico en 3D) rodeando Ω, ver Figura 1.6. Este proceso evita el tedioso proceso de mallado del MEF tradicional. Incluso geometrías muy complejas puede ser representadas con éxito con elementos de gran calidad sin necesidad de manipulación por el usuario.
Dejando a un lado la robustez del método, el siguiente asunto es la eficiencia. La

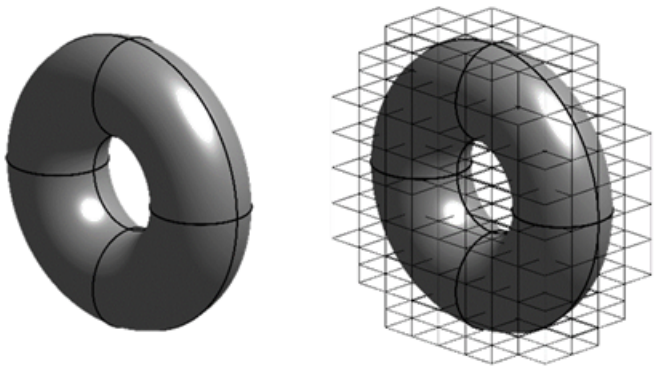


Figura 1.6: Dominio del problem Ω y dominio de la malla Ω_E.

eficiencia en comparación con códigos tradicionales de MEF puede ser obtenida a través de dos frentes diferentes: la generación del modelo numérico de MEF y su resolución. El primero de ellos hace referencia a la creación del problema numérico de MEF y el segundo a la manera de resolverlo. En *cgFEM* para construir el modelo numérico, confiamos en la estructura de la malla y la estructura de datos jerárquica implementada a este tipo de mallas para reducir la cantidad de cálculos a realizar. En general, en el MEF tradicional, cada elemento de la malla es diferente, y por ende, diferentes matrices de rigidez y puntos de integración por elementos. Por lo tanto, cada elemento debe ser evaluado por separado. En *cgFEM* la malla está formada por elementos con una geometría similar. La similitud geométrica entre los elementos conlleva a que algunos de los resultados de integración y en especial, la

matriz de rigidez están relacionados por un factor de escala, por lo tanto, solo un elemento es evaluado. Sin embargo, los elementos que están cortados por el contorno del dominio del problema $\partial\Omega$ necesitan un tratamiento individual especial. Este proceso evita numerosos cálculos innecesarios mejorando el rendimiento del método en comparación con el MEF tradicional.
Respecto a la resolución del sistema de ecuaciones global, se pueden diferenciar dos casos: método directo de resolución y iterativo. Cuando se usa un método directo es muy interesante tener la matriz del sistema de ecuaciones adecuadamente reordenada. En *cgFEM* el reordenamiento utilizado está basado en *Nested Domain Decomposition (NDD)*. En los métodos iterativos se utilizan técnicas de proyección para enviar la información desde una malla anterior. El uso de esta estructura de datos permite refinar las mallas a través de un procedimiento h-adaptativo, con el beneficio de que es un procedimiento automático a diferencia de otros métodos, ver Figura 1.7. El procedimiento típico de mallado que se seguiría en el marco de *cgFEM* sería el siguiente:

1. Se genera una secuencia de mallas cartesianas que contengan el dominio del problema Ω, esta secuencia de mallas ha sido denominada *pila*.

2. La malla resultante se construye a partir de un conjunto de elementos no-superpuestos de diferentes tamaños, obtenidos de la *pila* de mallas. La malla h-adaptativa resultante es no-conforme, por lo tanto la continuidad C^0 no está garantizada. Este problema se puede resolver a través de restricciones multi-punto (MPCs).

3. Existen tres tipos diferentes de elementos generados por la malla anterior: *internos*, *externos* y *de contorno*. Los elementos externos, se encuentran fuera del dominio y por lo tanto no se consideran en el análisis. Los elementos internos, se evalúan como elementos FE estándar. Por último, los elementos situados a lo largo del contorno $\partial\Omega$ solo parte de estos permanece dentro del dominio Ω. La evaluación de la matriz de rigidez de estos elementos requiere analizar la intersección de la geometría y los lados del elemento, para detectar la cantidad de elemento situada dentro del dominio.

4. Tras la intersección del contorno, se puede llevar a cabo la integración de la matriz de rigidez y las condiciones de contorno sobre los elementos de la malla.

5. Puesto que los nodos de los elementos no están situados sobre el contorno del dominio, las condiciones de contorno de *Dirichlet* no pueden ser aplicadas directamente sobre los nodos. Este incoveniente se soluciona utilizando un método de estabilización desarrollado en el *DIMM* [15].

6. Se reordena la matriz del sistema de ecuaciones y se resuelve, a través de un método directo o iterativo.

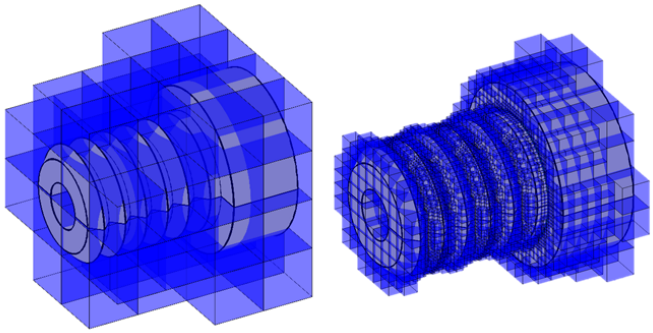


Figura 1.7: Malla basta y malla h-adaptada sobre geometría.

1.4. Paralelización

La computación en paralelo [16] es un forma de cómputo en la cual muchos de los cálculos o la ejecución de ciertos procesos son llevados a cabo simultáneamente. Los problemas más grandes en muchas ocasiones pueden ser subdividos en otros más pequeños, que a su vez pueden ser calculados al mismo tiempo. Adicionalmente al claro ahorro de tiempo en los cálculos se debe añadir un menor consumo de energía (con su respectiva reducción de calor generado), lo que convierte a la computación en paralelo en el paradigma predominante en arquitectura de computadores, principalmente en el desarrollo de procesadores con varios núcleos. Existen diferentes formas de computación en paralelo: nivel de bit, nivel de instrucción, nivel de datos y nivel de tareas. También se pueden encontrar en distintos formatos de *hardware*: en una sola máquina, con varios núcleos o procesadores, y usando varios computadores para trabajar en una tarea como por ejemplo,*clusters*, *MPPs* y *grids*.
A pesar de las bondades explicadas anteriormente, cabe destacar un inconveniente y este es que el trasiego de información llevado a cabo durante la comunicación de los diferentes elementos de la red consume una serie de recursos, lo cual se suele traducir en un mayor tiempo de computación global en todos los elementos de la red en caso de computación en serie (servidor). Por lo tanto, cuando el procesado de datos requiera de una cantidad de recursos y tiempo significativa, el intercambio de información y comunicación no es importante y la paralelización debería funcionar. Por el contrario, si el tamaño de problema fuese muy pequeño, paralelizarlo podría no ser la mejor opción, puesto que la duración del proceso de computación puede aumentar
La paralelización llevada a cabo no se enmarcaría dentro de ninguna de las clasificaciones anteriores. Se ha usado un ordenador, llamado *servidor*, una serie de ordenador provenientes de distintas aulas, llamados *clientes* y un protocolo de red, en concreto un *modelo TCP/IP*. El proceso fundamental de paralelización sobre este sistema sería aignarle una tarea al *servidor* y los *clientes* que deseemos. El *servidor* sería el encargado de establecer comunicación con cada uno de los *clientes* a través de *Internet* y el conocimiento de las direcciones *IP* de cada uno de ellos. El *ser-*

vidor gestionará un *buffer* de tareas que repartirá equitativamente entre cada uno de los *clientes* disponibles, estos realizarán la tarea encomendada y devolverán los resultados que estemos buscando, ver Figura 1.8. Tal y como se ha visto en ante-

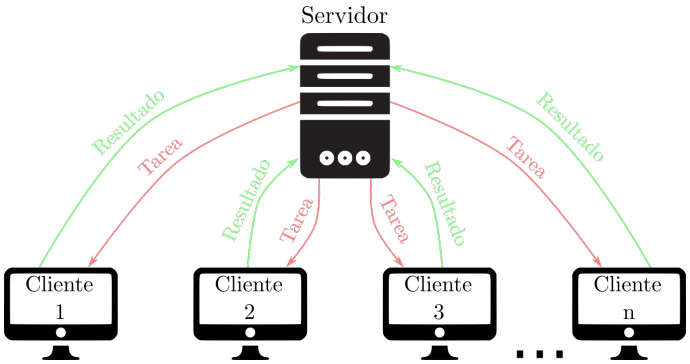


Figura 1.8: Esquema de reparto de tareas

riores apartados, existe una gran cantidad de algoritmos con distintas naturalezas de funcionamiento, esto se traduce en que no todos ellos presentarán las mismas facilidades a la hora de paralelizar el cálculo de individuos.
En concreto, los algoritmos pertenecientes a la familia de los metaheurísticos, en este trabajo el algoritmo genético, serían facilmente paralelizables. Esto es debido a que a cada iteración del proceso de optimización, el algoritmo genera una población con un determinado número de individuos, siendo estos independientes entre sí. Al no existir ninguna dependencia entre individuos dentro de cada población, ésta se puede paralelizar de modo que se enviará cada individuo a un cliente distinto.
Sin embargo, los algoritmos que basan la actualización de variables en las derivadas de la función objetivo o restricciones, no pueden ser paralelizables, a priori, puesto que cada uno de los individuos depende directamente del anterior. De modo que este algoritmo pertenece a la familia de los algoritmos tipo *line search* porque su esquema de actualización es igual al que aparece en la ecuación 1.5

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1.5}$$

donde, x_{k+1} y x_k, son el valor de la variable de diseño en las iteraciones *k+1* y *k* respectivamente, α, es el parámetro que mide el tamaño de paso entre iteraciones, que puede ser obtenido a través de un proceso de *backtracking line search* [17] o usando las *condiciones de Wolfe* [18] y por último, d_k que define la dirección que debe ser seguida y habitualmente depende los gradientes o hessianos de la función objetivo.
El único método plausible mediante el cual paralelizar este tipo de algoritmos sería inicializarlo en clientes diferentes con distintos puntos de inicio con el fin de abarcar el mayor rango del espectro de posibles soluciones.

2

Desarrollo de metodología e implementación

En este capítulo, se detallará la metodología utilizada para implementar las herramientas presentadas en la introducción para dar forma a un *software* de optimización con un modulo que permita paralelizar procesos, tanto iterativos (*Optimización*), como otros como por ejemplo procesar un listado de tareas a realizar simultáneamente.
La aplicación se ha desarrollado en *MATLAB* por lo tanto nos valdremos de funciones *built-in* para llevar a cabo la mayoría del desarrollo del *software*. Entre ellas cabe destacar las funciones que incluyen gran cantidad de algoritmos de optimización como pueden ser: *fmincon*, *ga* y *globalsearch*, que más adelante se detallarán. También serán fundamentales las funciones que se encargarán de la gestión y comunicación entre ordenadores, que son entre otras: *timer*, *tcpip* y *BytesAvailableFunction*. Y, por último, las funciones encargadas de entrenar y evaluar las redes neuronales, que serán de vital importancia, como por ejemplo *feedforwardnet*, *train* y *perform*.
En la Figura 2.1, se muestra un diagrama descriptivo de la metodología implementada. Este ejemplo se centra en el caso de concreto de una optimización, aunque algunas de las herramientas pueden utilizarse en otros ámbitos.
De esta forma, la metodología a seguir para realizar un proceso iterativo de optimización comenzaría con la selección de algoritmos, y tomando el primero de ellos como inicio de la ejecución. Una vez dentro de un determinado algoritmo, se procederá con el cálculo del individuo actual. En caso de que existan *RNA* entrenadas el resultado se obtendrá a través de éstas, si no, el análisis numérico ha de realizarse. Si se dispone de ordenadores adicionales y el algoritmo lo permite, se enviará el individuo a uno de estos, y si solo se dispone del servidor, los análisis numéricos se efectuarán en serie en éste. Tras la obtención de resultados se comprobará la convergencia del actual algoritmo, en caso de converger se tomará el mínimo alcanzado como valor inicial del siguiente algoritmo. Este proceso se repetirá hasta la convergencia del último de los algoritmos.

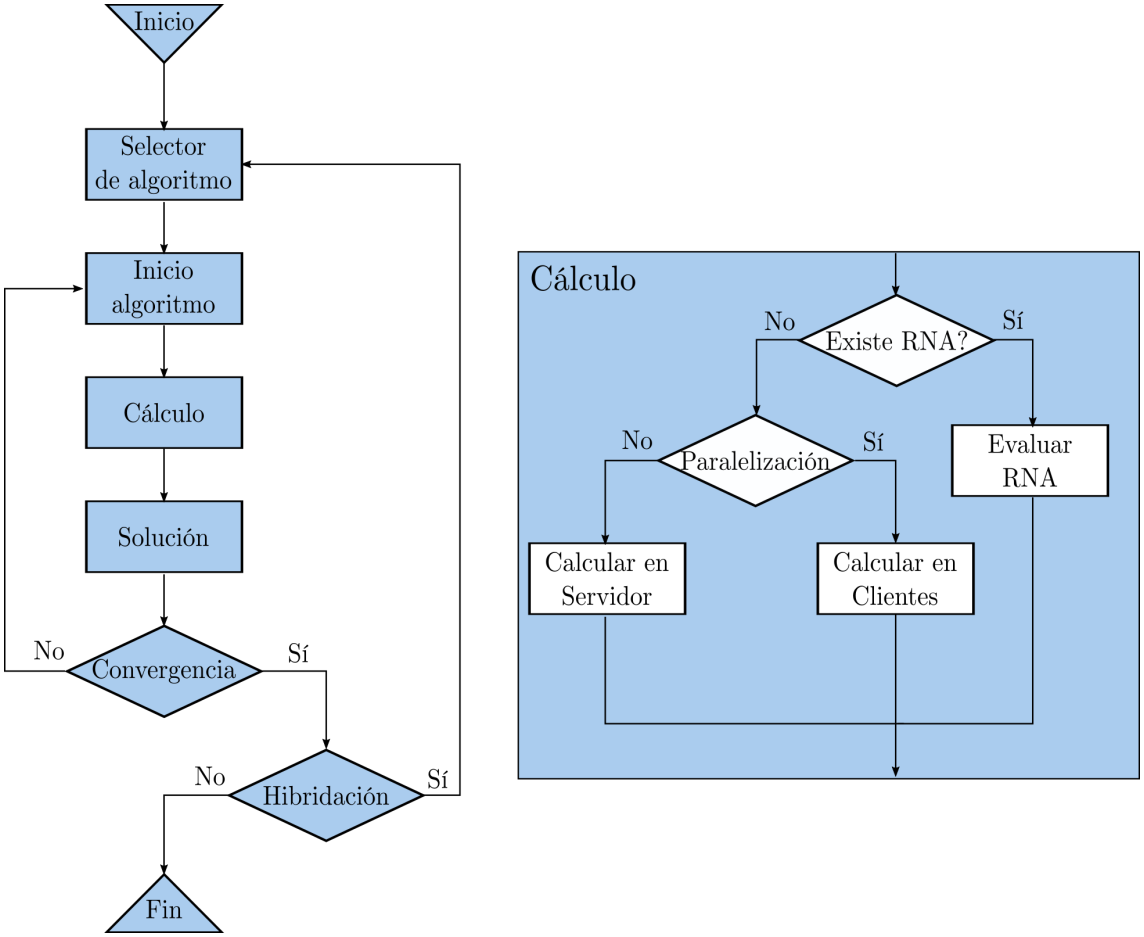


Figura 2.1: Flujo de trabajo de la aplicación.

2.1. Integración algoritmos de optimización

En lo referente a los algoritmos de optimización y partiendo de la clasificación hecha anteriormente trataremos con tres funciones principales. Todas ellas pertenecen al *toolbox* de optimización de *MATLAB* La primera de ellas es *fmincon*, esta función se encuentra dentro de la familia de funciones que necesitan (o simulan) los gradientes o hessianos tanto de la función objetivo como de las posibles restricciones. La función es capaz de encontrar un mínimo local en problemas que tengan la siguiente estructura:

$$\min_x f(x) \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

donde *f(x)* es la función objetivo (*en nuestro caso, volumen o peso*), *c(x)* y *ceq(x)* corresponde a las restricciones (por ejemplo *tensión de Von Mises*) de desigualdad e igualdad no lineales respectivamente, *A, Aeq, b* y *beq* son parámetros que representan las restricciones de desigual e igualdad lineales entre las variables de diseño.
Entre los atributos modificables por el usuario de *fmincon* cabe destacar algunos de los más importantes, como por ejemplo *Algorithm* que es el tipo de algortimo matemático que debe seguir el proceso de optimización, entre los cuales se encuentra *Interior Point Method* [19] y *Sequential Quadratic Programming*.
De la misma familia, existen otras funciones como *fminunc*, *fminsearch* y *fgoalattain*. El uso de *fmincon* frente a otros algoritmos de optimización dentro de la familia familia se debe a que:

- Asumen funciones objetivo y restricciones continuas. En el caso tratado en este trabajo será así, pero no siempre será asumible esa hipotesis.
- Al igual que *fmincon* todos ellos pueden converger únicamente a soluciones locales.
- Ninguna de ellas son compatibles con el uso de *GlobalSearch*, a diferencia de *fmincon*.

¿Por qué es importante el uso de *GlobalSearch*? Esta es la segunda función del *toolbox* de *MATLAB* y permite encontrar la solución global óptima. La forma de actuar de ésta es a través del inicio inteligente de diversas instancias de *fmincon*. Los puntos de inicio del algoritmo se toman de modo que hagan un barrido del espectro de soluciones lo más óptimo posible. Esta función tiene una serie de atributos modificables por el usuario que son:

- *NumTrialPoints*, define la cantidad de puntos de inicio potenciales, además de x0 (*punto de inicio sugerido por el usuario*). Estos puntos se someten a una serie de pruebas y en caso de superar cierto criterio será considerados como válidos.
- *NumStageOnePoints*, indica cuántos de los puntos válidos son evaluados en la primera etapa del proceso.
- *StartPointsToRun*, es un filtro adicional mediante el cual se discriminan los puntos de inicio.

El diagrama de trabajo de *GlobalSearch* sería similar al mostrado a continuación:

1. Ejectua *fmincon* desde el punto sugerido por el usuario.
2. Genera el vector de puntos de prueba potenciales.
3. **Etapa 1**. Ejectua *fmincon* desde el mejor de los puntos de prueba.

4. **Etapa 2**. Evalúa el resto de puntos y ejecuta *fmincon* si un punto satisface las condiciones adecuadas.

5. Crea el vector *GlobalOptimSolutions* que recolecta todas las soluciones obtenidas.

En lo referente a algoritmos basados en gradiente, con las funciones anteriormente descritas el *software* ya sería capaz de ejecutar un proceso de optimización. Con el fin de aprovechar la hibridación de algoritmos, también se ha implementado otro algoritmo del tipo metaheurístico para poder ser utilizado en el *software*. En concreto se usará la función *ga* también del *toolbox* de optimización de *MATLAB*. La función *ga* ejecuta un algoritmo genético que se enmarcan dentro de los algoritmos evolutivos, los cuales son llamados así porque su funcionamiento está inspirado en las reglas de evolución biológica, sobretodo en su base genético-molecular.
La optimización toma en cada iteración una población de individuos, también llamadas generaciones. El algoritmo hace evolucionar a la población de acuerdo a una serie de acciones de base aleatorias, como las mutaciones y la recombinación genética, tal y como ocurre en la naturaleza. Adicionalmente, se someten las poblaciones a un criterio de selección para definir cuales son los individuos más adaptados y que sobreviven, ver Figure 2.2.
Por la propia naturaleza de este tipo de algoritmos, son especialmente útiles si se

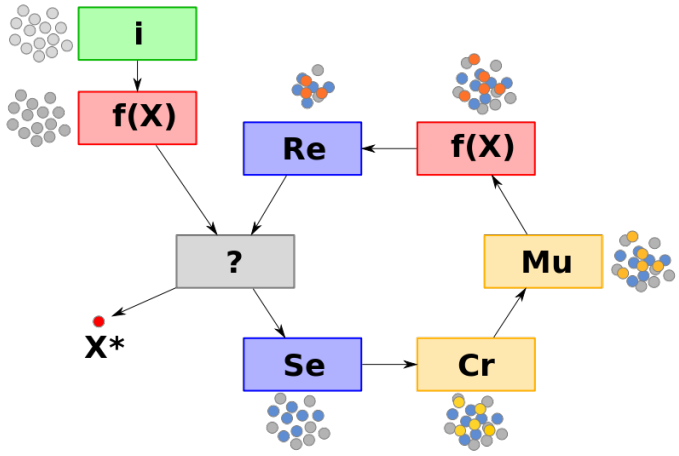


Figura 2.2: Esquema del proceso iterativo del algoritmo genético, donde **I** es la población inicial, **f(X)** la evaluación de la población inicial, **?** la comprobación de convergencia, **Se** la selección de los mejores individuos, **Cr** la recombinación de los individuos, **Mu** la mutación de algunos individuos, **Re** el reemplazo de nuevos individuos élite y **X** el individuo óptimo.

desea encontrar la solución global óptima. Esto se debe a que las primeras poblaciones suelen cubrir un amplio espectro del campo de posibles soluciones. El algoritmo

una vez encuentra individuos élite con poca variación entre sí, focaliza la búsqueda hasta que al final converge a la solución óptima, habitualmente la global.
Un vez detallados los distintos algoritmos a utilizar en el software de optimización veremos como integrar estas tecnologías en un mismo código. La definición de los atributos de los diferentes algoritmos pueden realizarse tanto a través de la interfaz de usuario como modificando directamente sobre el código, para un usuario no experimentado se recomendará la primera. Independiente del método de definición la hibridación se realizará a través de un bucle (*for*), siendo de vital importancia la correcta definición del criterio de parada de cada uno de los algoritmos que entrarán en juego a lo largo del proceso. Los posibles criterios de parada pueden ser:

- **Número de iteraciones**: la cantidad máxima de iteraciones que puede realizar cada uno de los algoritmos.
- **Tiempo**: criterio de parada en función del tiempo de ejecución.
- **Tolerancia**: si la variación de una solución a la anterior es menor o igual que cierto valor definido por el usuario.
- **Sin criterio**: cuando solo se ejecuta un algoritmo.

2.2. Matriz de resultados. Beneficios

Con el fin de tener un histórico del proceso cargado en memoria se creará una variable de resultados (*ResultData*). En ella estará disponible toda la información útil que pueda ser solicitado por el *software* en cualquier momento.
El uso de esta variable aporta varios beneficios. Así, entre las posibles opciones que tiene a su disposición el usuario en el entorno desarrollado, existe la recuperación de los resultados obtenidos de un análisis anterior en caso de que el individuo se repita, en la Figura 2.3 se muestra un breve diagrama del modulo.

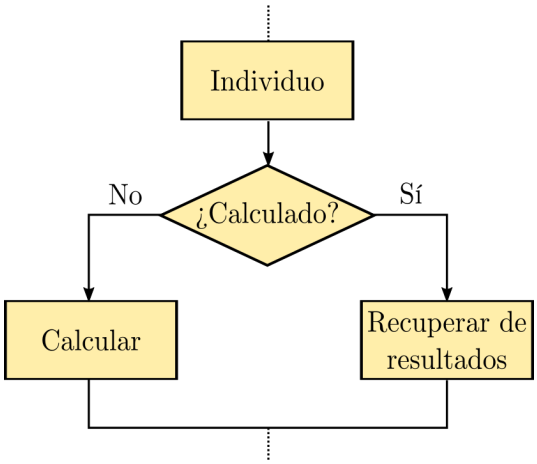


Figura 2.3: Diagrama de la obtención de resultados a partir de individuos ya calculados.

La adición de esta opción se debe a que los algoritmos de *MATLAB* tienden a llamar repetidas veces a un mismo individuo ya sea porque quiere obtener por un lado el valor de la función objetivo y por otro el de la restricción, o porque el propio algoritmo les conduce a un individuo que ha aparecido previamente. De la misma manera que el usuario puede elegir si arrancar el proceso a partir de un análisis previo o de iniciar uno nuevo.

2.3. Redes Neuronales Artificiales. Entrenamiento y uso

Otras de las opciones a disposición del usuario es la creación y uso de Redes Neuronales Artificiales (*RNA*), siguiendo el diagrama de la Figura 2.4. Anteriormente se ha detallado el funcionamiento de estas de un modo conceptual, a continuación se explicará como hacerlas funcionar a través de las funciones de *MATLAB*. La implementación de esta herramienta puede dividirse en dos etapas claramente diferenciadas, por un lado tendríamos la creación de la *RNA* y por otro su uso. En cuanto a la creación de las *RNA*, su aplicación se ha planteado de modo que exista una *RNA* para el cálculo de la función objetivo y otra para las restricciones, en el caso de optimización estructural será el volumen y la tensión máxima, respectivamente.
El primer paso para crear la *RNA* es tener una base de datos debidamente organizada, puesto que ya almacenamos una variable con toda la información del proceso (*ResultData*), solo será necesario recopilar los datos necesarios. De la matriz habría que extraer los únicos individuos que han sido obtenidos a través de *FEM*, el resultados de este análisis es considerado como el exacto a pesar de tener errores de discretización del modelo continuo. Por un lado se extraen las variables de diseño

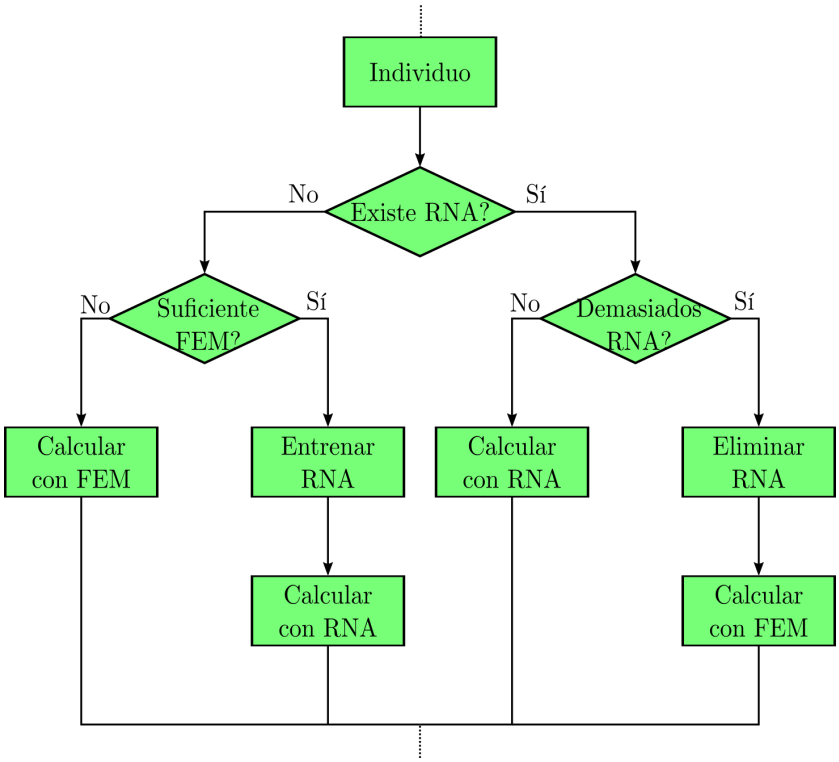


Figura 2.4: Diagrama de la obtención de resultados a partir de *RNA*.

ya que estas serán los *inputs* de la *RNA*, y por otro el valor de la función objetivo y restricciones por separado, ya que serán los *outputs* de las *RNA* respectivamente. Luego, es importante definir los parámetros que rigen el proceso de creación de las *RNA* como peuden ser la cantidad de neuronas en la capa oculta o la función de entrenamiento. Por defecto, los valores de cada una de estas variables son, por un lado, 10 neuronas por capa oculta y por otro, una función de entrenamiento definida como *'trainlm'*. Esta función está basada en el algoritmo Levenberg–Marquardt usado para resolver problemas de mínimos cuadrados no-lineales, como esquema para la actualización de los pesos en las conexiones. Adicionalmente, se definirá un parámetro extra que dicta la cantidad de *RNA* que deben ser creadas, esto se debe a que una vez se hayan entrenado las *RNA* que dicte ese parámetro se comprobará el rendimiento de todas ellas y nos quedaremos con la mejor de todas, desechando el resto.

Una vez disponemos de todos los ingredientes para crear un *RNA*, procedemos a llamar las instrucciones de *MATLAB* que nos permitirás crear, entrenar y validar las redes. La primera de ellas es la función *feedforwardnet* que genera una red por defecto a partir del número de neuronas en la capa oculta y las función de entrenamiento. En lugar de esta función se puede usar *cascadeforwardnet* que crea una conexión directa entre la entrada y la salida, puenteando la capa oculta o *network* mediante la cual se puede crear una red a medida de nuestras necesidades. Los beneficios de usar esta función es que la mayoría de parámetros ya están predifinidos

como por ejemplo el número de capas o las funciones de transferencia que existen en cada una de ellas. En la Figura 2.5 se aprecia estas funciones de transferencia así como la cantidad de capas que forman las red y las neuronas de cada una de estas.
Después, se añadirán algunos parámetros extra, que tiene relación con la división

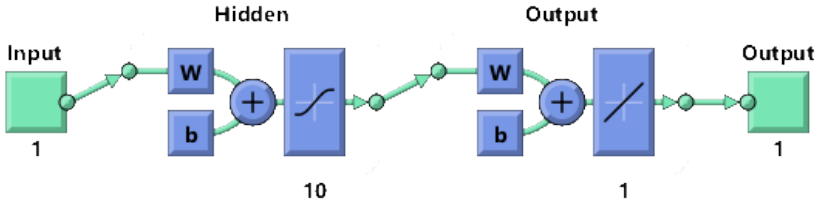


Figura 2.5: Esquema de la *RNA* generado por *MATLAB*.

de los datos de entrenamiento. En nuestro ejemplo, se usará el 70 % de la información para entrenar la red, un 15 % para la validación cruzada y el 15 % restante en *testear* la red entrenada, esta configuración es tan válida como otra cualquiera. Tras esto, se llamará a la función *train* que se encargará de entrenar la *RNA* a partir de los parámetros definidos anteriormente. Una vez se haya entrenado, obtendremos los valores *output* a partir del *input* de la red con el fin de medir el rendimiento y fiabilidad de la red comparando este *output* con los valores de entrenamiento. Todo este proceso estará dentro de un bucle con el fin de quedarnos con la mejor de las redes entrenadas. El último de los pasos será el de almacenar la *RNA* en el disco duro ya que para tenerla cargada en memoria se debería definir como una *variable global* y son excesivamente pesadas.
En cuanto al uso de las *RNA* el procedimiento es simple, cuando puedan ser utilizadas se cargarán en memoria desde el disco duro, esto generára un objeto que puede ser llamado como una función habitual, siendo el *input* las variables de diseño y el *output* el valor de la función objetivo o de las restricciones en función de la red que se haya cargado previamente.
Del mismo modo que las *RNA* deben ser creadas con un cantidad de análisis de *FEM* determinada para que la aproximación sea lo suficientemente fiable, las *RNA* solo puede ser usadas cierta cantidad de veces para obtener resultados. Esto se debe a que el ajuste se ha realizado con un rango de variables de diseño demasiado amplio, en el momento que el algoritmo de optimziación se focaliza en una zona óptima, las *RNA* dejan de tener utilidad ya que no se han entrenado para aproximar con detalle esa zona concreta. La *RNA* deberá ser desechada y tras otro número considerable de análisis de *FEM* podrá volver a ser creada esta vez aproximando una zona más concreta de todo el espectro de soluciones.

2.4. Paralelización. Gestión y comunicación

La paralelización que se ha desarrollado en este trabajo, como se ha comentado anteriormente, basa su funcionamiento en la tecnología *TCP/IP* con la finalidad de comunicar el servidor con cada uno de los clientes que se tenga a disposición. Dando unas pequeñas pinceladas, este proceso quedaría de la siguiente forma: el *servidor* enviaría un individuo o tarea a una determinada dirección *IP* que es el indicativo de cada uno de los clientes, con la información recibida éste sería capaz de resolver el cálculo o completar la tarea y devolver los resultados obtenidos al *servidor* valiéndose de su dirección *IP*, ver Figura 2.6

El planteamiento de la comunicación por parte del servidor y el cliente nace a

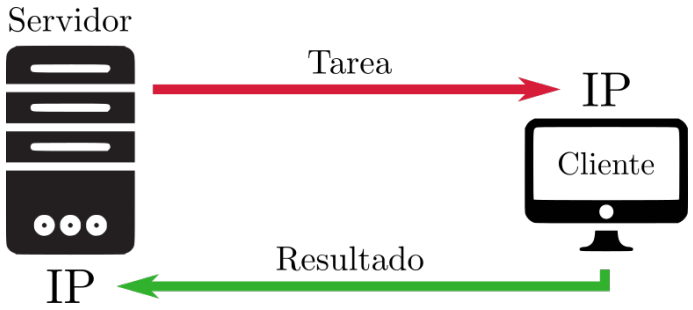


Figura 2.6: Representación de comunicación a través de la dirección IP.

partir de la emulación de un diálogo corriente entre personas. Este planteamiento se traduce de modo que en función de el tipo de información recibida se actuará de una determinada manera, del mismo modo que se devolverá la respuesta acertada para cada momento concreto. Todo este intercambio de información puede realizarse con un sencillo criterio de clasificación de órdenes, de modo que cualquier dato que se transfiera durante la comunicación siempre irá precedido con un valor numérico que dictará como debe actuar cada uno de los interlocutores, es decir el flujo de datos se conducirá hasta la función apropiada.

A continuación, en la Figura 2.7, se muestra un diagrama en el que se representa el flujo de información llevado a cabo entre el servidor y uno de los clientes.

Con el fin de ofrecer algo más de información sobre el diagrama de comunicación, a continuación se muestra con mayor detalle los pasos llevados a cabo durante un comunicación exitosa:

1. **Servidor**: Envía la orden de actualización.
2. **Clientes**: Comprueba la versión de *software* disponible en el directorio y se informa al servidor ello.
3. **S**: Toma la versión de los clientes, en caso de que sea una versión del *software* diferente de la disponible en el servidor el cliente deberá ser actualizado. Enviar orden de actualización junto con la última versión del *software*. Si la versión es la correcta, saltar al paso *6*.

4. **C**: Re-arranca *MATLAB* con la versión adecuada de los ficheros. El cliente estará a la espera hasta que se realice nuevamente la conexión.

5. **S**: Repite los pasos anteriores hasta que los clientes estén actualizados. En ese momento se envía la orden de iniciar el proceso a los clientes.

6. **C**: Solicitan al servidor el envío de una tarea del *buffer* de tareas.

7. **S**: Gestiona el *buffer* de tareas para evitar enviar trabajos que ya está realizando otro cliente y envía una instrucción para cada uno de los clientes. En caso de que haya clientes parados, se reenviarán las tareas por si alguno de los otros clientes falla en su cometido (Error al calcular, corte de luz, apagado involuntario, etc.).

8. **C**: En función del tipo de tarea, realizará una determinada acción u otra. El caso más habitual será el de lanzar un análisis de *FEM*, tanto a través del *software* desarrollado por el centro (*FEAVox*), como por *software* comercial (*ANSYS*, *ABAQUS*, etc). Informa al servidor de que se ha comenzado el cálculo.

9. **S**: Se comprueba: 1) si se han obtenido resultados de esa misma tarea a través de otro cliente y 2) si se ha excedido el tiempo máximo de cálculo determinado por el usuario. En función de la comprobación anterior pueden suceder dos cosas:

    - Si se cumplen:, se envía la orden de abortar el proceso de cálculo al cliente que todavía no haya concluido
    - Si no se cumplen, se envía una orden al cliente para que informe del estado de la tarea.

10. **C**: Existen dos escenarios posibles:

    - En caso de "matar.$^{\text{el}}$ proceso, se ejecuta el procedimiento para concluir un cálculo, habitualmente a través del *Símbolo del sistema*. Si ha sido porque otro cliente ha acabado antes de procesar exactamente el mismo cálculo, simplemente se solicitará una tarea nueva. Pero si ha sido a causa de un exceso de tiempo de cálculo, se informa al servidor de que ha ocurrido algún error.
    - En cambio, si el servidor solicita información acerca del estado de la tarea, se le informará tanto si la tarea ha concluido como si no. En caso de que no haya acabado el servidor hará llamadas recursivas al clientes hasta que la tarea haya finalizado.

11. **S**: Con el cálculo concluido, el servidor solicitará al cliente que recopile los resultados.

12. **C**: Recupera los resultados generados por la tarea y los envía con un orden determinado, para ser fácilmente interpretado por el servidor.

13. **S**: Recibe los resultados que, habitualmente, se almacenan en una matriz en la que se refleje el histórico de todo el proceso (*ResultData*). Se envía una nueva tarea a los clientes y se repiten los pasos anteriores.

Adicionalmente a lo visto anteriormente, se ha añadido un nuevo nivel de comunicación entre el servidor y los clientes, que podría denominarse *nivel de estado*. En la Figura 2.8, se representa los distintos niveles en los que se ha dividido la aplicación, dos niveles de comunicación independientes entre sí, *estado* y *comunicación* y el encargado de completar las tareas, *cálculo*. El propósito de la adición de este nuevo nivel es el de poder interrogar a cada uno de los clientes sobre su estado, y que estos devuelvan la información al servidor para un correcto funcionamiento del programa. Este nivel de comunicación será transparente al ya existente, de modo que no interferirá y solamente informará al servidor si el cliente está en ejecución o no. En caso de estarlo, dirá si se encuentra trabajando o está a la espera de tareas nuevas. En caso de que el cliente, al que se le ha ordenado trabajar, no esté en ejecución, el servidor debe buscar la forma de volver a incluir ese cliente en el listado de clientes activos. Puesto que durante la elaboración de este trabajo no se disponían permisos de administrador sobre los clientes, la capacidad de reacción ante esta situación es re-arrancar *MATLAB* o en caso de que el ordenador se haya apagado habría que volver a conectarlo. Actualmente, ambas tareas se harían de forma manual. También puede darse el caso de que al principio de la ejecución se consideren ciertos clientes que, por cualquier circunstancia, no están preparados para establecer la comunicación, este nivel también sería el encargado de establecer las nuevas conexiones en el momento de que los clientes estén preparados.

Figura 2.8: Esquema de niveles de la aplicación de paralelización.

Puesto que el *software* de paralelización no tiene porqué estar limitado a algoritmos de optimización, se creará una función para romper esas barreras y poder ser usado en otros ámbitos. Esta función será un gestor de tareas, llamada *QueueManager*, y entre otras, sus aplicaciones pueden ir desde la optimización topológica hasta el estudio paramétrico.
La función *QueueManager* es una mezcla de tecnologías preintegradas en *MATLAB* ordenadas e interconectadas de tal forma que crean sinergias entre sí y permiten

Tabla 2.1: Representación de la información de cada de los clientes.

Nombre	IP	Puerto	Directorio	Versión
PC1	XXX.XXX.XXX.XXX	1024	\\PC1\RecursoCompartido	1
PC2	XXX.XXX.XXX.XXX	1025	\\PC2\RecursoCompartido	1
...	...	...	...	...
PCn	XXX.XXX.XXX.XXX	n	\\PCn\RecursoCompartido	1

crear un gestor de tareas y reparto eficiente. La primera de ellas es la interacción con el *software* que gestiona textos, en nuestro caso, el programa leerá datos a partir de una hoja *Excel*. Esto se debe a que el usuario necesita una interfaz para gestionar las información de los clientes. En esa hoja de *Excel* aparecerán los datos de todos los clientes disponibles: nombre, *IP*, puerto de conexión, el directorio donde se almacenan los ficheros y la versión del calculador usado, ver Tabla 2.1.
De entre todos los clientes disponibles el usuario debe seleccionar aquellos que quiera incluir en el proceso de paralelización. Para ello se han creado dos hojas distintas en un mismo fichero *Excel*, en una de ellas se mostrará todas la información de todos los clientes, mientras que en la otra se hará la selección de determinados clientes, está será la hoja que leerá el programa. Se creará una estructura de datos en la que se almacene toda la información de cada uno de los clientes que intervienen en la ejecución para que sea ágilmente llamada por cada uno de los diferentes módulos.
El resto del programa esta apoyado sobre un objeto de *MATLAB* llamado *timer*. Este objeto será el encargado de: en una primera ejecución establecer la comunicación con los clientes disponibles en ese momento, comprobar periodicamente el estado de todos los clientes y establecer nuevas conexiones en caso de que sea necesario.
Este tipo de objetos están definidos por tres atributos que se detallan a continuación:

- ***ExecutionMode***: entre otras, cabe destacar dos opciones básicas, *singleShot*, donde la función del *timer* es ejecutada una sola vez y *fixedSpacing*, que lo hace cada determinado lapso de tiempo.

- ***Period***: especifica, en segundos, el lapso de tiempo entre ejecuciones de la función del *timer*.

- ***TimerFcn***: definida como la función del *timer*. Ésta será ejecutada siguiendo las propiedades deifinidas en los otros atributos, *ExecutionMode* y *Period*.

La función del *timer* será el pilar sobre el cual se asienten el resto de modulos del programa, referentes a la gestión de tares y comunicación entre clientes.
La primer modulo del que se hablará, comprueba el estado de los clientes, lanza un *ping* a cada uno de los clientes seleccionados por el usuario. De entre todos los clientes que devuelvan el *ping* al servidor, se comprobará la existencia de un archivo de texto en cada uno de los clientes. Ese archivo de texto, llamado *Available.txt*, el cual se genera una vez se ejecute la función de conexión localizada en cada cliente,

se detallará más adelante.
Una vez se conocen los clientes potenciales, se procederá a la conexión entre ellos y el servidor. El procedimiento es tal que, se llama a la función *tcpip* de *MATLAB* creando el objeto *TCP/IP Servidor*. Una vez creado, se abrirá el puerto definido en la hoja *Excel*, con el comando *fopen* de *MATLAB*, para permitir la comunicación. Simultáneamente, se enviará un fichero de texto a cada uno de los clientes, este fichero sirve como aviso para crear el objeto *TCP/IP Cliente*, abrir el puerto de comunicación del cliente y por último, el contenido de ese fichero sirve para configurar el tipo de problema a paralelizar.
La propiedad del objeto *TCP/IP* llamada *BytesAvailableFcn* sirve como base para la comunicación entre el servidor y los clientes. Esta propiedad no es más que una función que trabaja a modo de *callback function*, lo cual significa que se ejecuta cada vez que exista cierta cantidad de *bytes* en el *buffer* de entrada. Dentro de esta función cada una de las acciones es gobernada por un *switch* que es alimentado por el identificador de instrucción que acompaña a cada transferencia de información, como se ha visto anteriormente. Cada una de estas acciones corresponden a las vistas en el diagrama de comunicación ya descrito.
Del mismo modo que el servidor se vale de este tipo de funciones, cada uno de los clientes se debe crear una función que trabaje del mismo modo. La diferencia principal entre estas dos funciones es que la que pertenece al servidor hace un especial hincapié en la gestión de los resultados obtenidos, mientras que la del los clientes, en el cálculo de los individuos o la realización de las tareas recibidas.

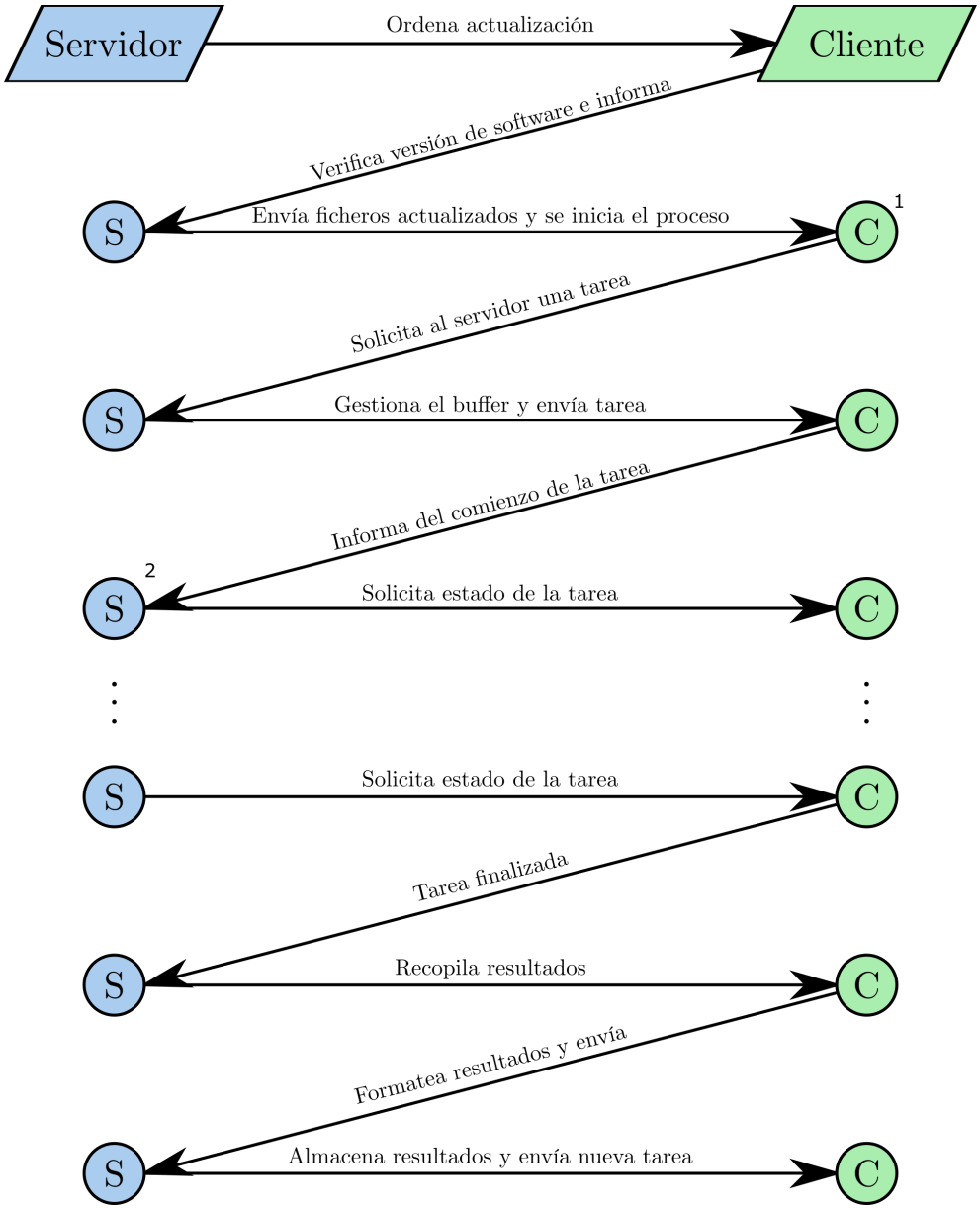


Figura 2.7: Representación esquemática de la comunicación entre el servidor y uno de los clientes. [1]Si está desactualizado, se re-arrancará *MATLAB* y el cliente permanecerá a la espera del servidor. [2]El servidor comprobará si la tarea debe ser abortada o no.

3

Resultados y ejemplos

En el siguiente capítulo, se mostrará el funcionamiento del *software* a través de una serie de ejemplos de diversa índole, todos ellos se beneficiarán de la paralelización descrita anteriormente:

- Optimización topológica mediante celdas.
- Estudio paramétrico junto con *RNA*.
- Optimización de forma con algoritmos híbridos.
- Optimización de forma mediante algoritmos genéticos.

Todos estos ejemplos se han llevado a cabo contando con los equipos disponibles entre las distintas aulas de informática que se encuentran en el Departamento de Ingeniería Mecánica y Materiales de la UPV, por ese mismo motivo, las características de los equipos utilizados son bastante heterogéneas. Éstas se resumen a continuación:

- Equipos Tipo **A**.
  - Intel$^{(R)}$ Core$^{(TM)}$ i5-3470 con 4 núcleos a 3,2 GHz.
- Equipos Tipo **B**.
  - Intel$^{(R)}$ Core$^{(TM)}$ i3-4150 con 4 núcleos a 3,5 GHz.
- Equipos Tipo **C**.
  - Intel$^{(R)}$ Core$^{(TM)}$ i3-530 con 4 núcleos a 2.93 GHz.

Todos estos equipos comparten una serie de características comunes descritas a continuación:

- Windows 10 Enterprise 64 bits.
- Memoria RAM de 8 Gb.

- MATLAB R2017a.
- ANSYS 18.1.

3.1. Optimización topológica mediante celdas

La optimización topológica apoya su esquema de actualización de variables en un algoritmo basado en gradiente, tal y como se ha detallado anteriormente este tipo de algoritmo no es fácilmente paralelizable. Una forma de distribuir el problema de optimización entre varios equipos es discretizando el problema en celdas y tomando cada una de estas como un problema independiente de optimización topológica.
Este procedimiento, da como resultado un problema definido en dos niveles. Por un lado se encuentra el dominio global del problema y por otro, el dominio de cada una de las celdas.
A continuación se muestran, a grandes rasgos, el procedimiento a seguir para resolver un problema de optimización topológica mediante este procedimiento:

1. Se somete al dominio global a una fase completa de optimización topológica. Puesto que no interesa penalizar la distribución de densidades, se podría decir que no se usa el esquema de SIMP [20] o que se usa con un valor de penalización igual a 1. Esto es debido a que se busca una distribución de densidades lo más suave posible.

2. Se recogen los resultados del anterior problema y, siguiendo un proceso iterativo sobre la fracción de volumen se determina la fracción de volumen mínima que garantice que las tensiones de Von Mises no superen la tensión de fluencia, S_y.

3. Los anteriores pasos, dan como resultado una distribución de densidades sobre el dominio global. Ese campo de densidades será tomado por cada celda, respectivamente como fracción de volumen de su propio problema de optimización.

4. Además de la fracción de volumen, se debe definir las condiciones de contorno de cada una de las celdas. Eso se consigue postprocesando el campo de tensiones obtenido en el dominio global a través del diagrama de Maxwell [21]. Este procedimiento dará como resultado un campo de tracciones equilibrado en cada una de las celdas y entre cada una de las celdas, de manera que cada una de ellas se podrá evaluar de manera independiente, pudiendo realizarse en paralelo.

5. Previo a la paralelización, es vital generar una base de datos con la cantidad de información suficiente como para que el *gestor de colas* sea capaz de enviar a cada cliente la celda adecuada con su respectiva información adicional, que no es otra que la obtenida en pasos anteriores.

6. Se enviarán recursivamente las celdas a los clientes hasta que se complete el listado de tareas.

7. A través de una recopilación de los resultados obtenidos, se generará la estructura óptima en el dominio global, situando cada una de las celdas en su respectivas posiciones.

En este problema se tiene un dominio rectangular, que representa un estructura empotrada por uno de sus extremos y con una combinación de cargas en el extremo opuesto. En la combinación de cargas contribuyen una carga parabólica vertical y una carga horizontal. Este problema se puede representar con un esquema como el que aparece en la Figura 3.1.

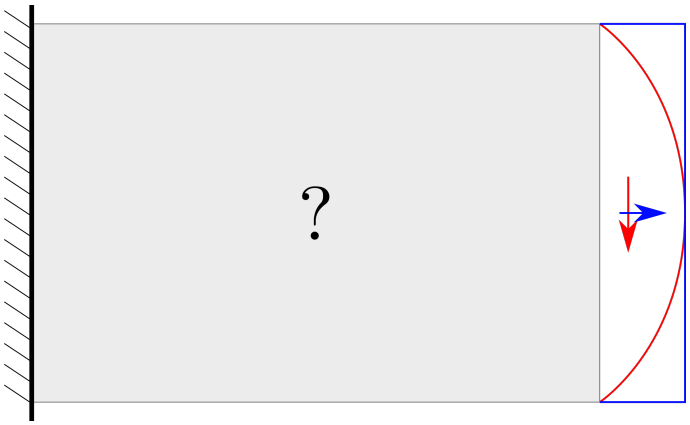


Figura 3.1: Esquema del problema de optimización topológica con sus cargas aplicadas.

Todo el cálculo realizado en el servidor sirve para generar una base de datos rica en información. El resultado obtenido, entre otros, es una distribución suave de densidades intermedias en el dominio, como puede verse en la Figura 3.2.

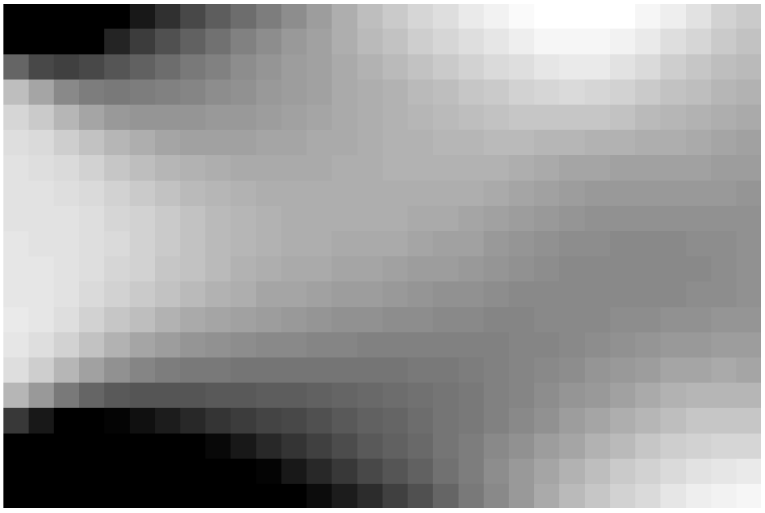


Figura 3.2: Distribución de densidades en el dominio global del problema.

Como se ha comentado anteriormente, tras este paso, se debe realizar un proceso de equilibrado de tracciones en cada una de las celdas. Con esta información y la distribución de densidades, se creará un base de datos con la información necesaria para paralelizar el problema. Enviando a cada cliente una celda distinta y siendo el servidor el encargado de recopilar la información resultante. Se obtendrá como resultado una estructura como la que aparece en la Figura 3.3.

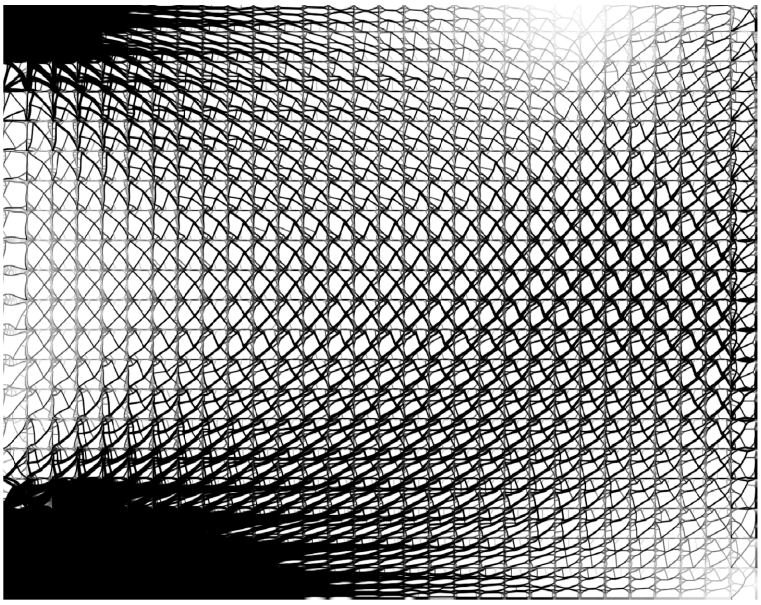


Figura 3.3: Dominio global óptimo generado por los resultados de las celdas.

Una simplificación del problema anteriormente descrito se ha analizado un determinado número de veces variando en cada una de ellas la cantidad de clientes

disponibles para el cálculo, con el objetivo de obtener una relación entre la cantidad de ordenadores y el tiempo de procesamiento. Por lo tanto el problema a resolver no será el explicado anteriormente sino el que se muestra en la Figura 3.5 que representa la solución óptima obtenida.

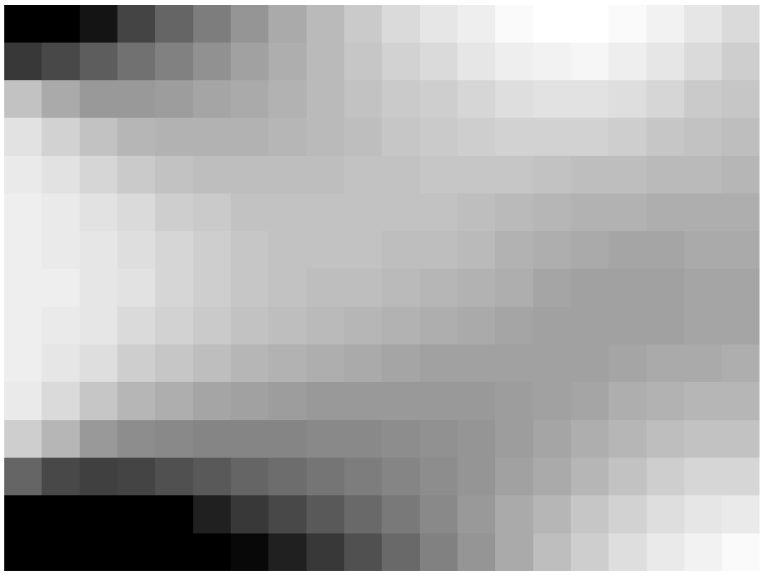


Figura 3.4: Distribución de densidades en el dominio global del problema simplificado.

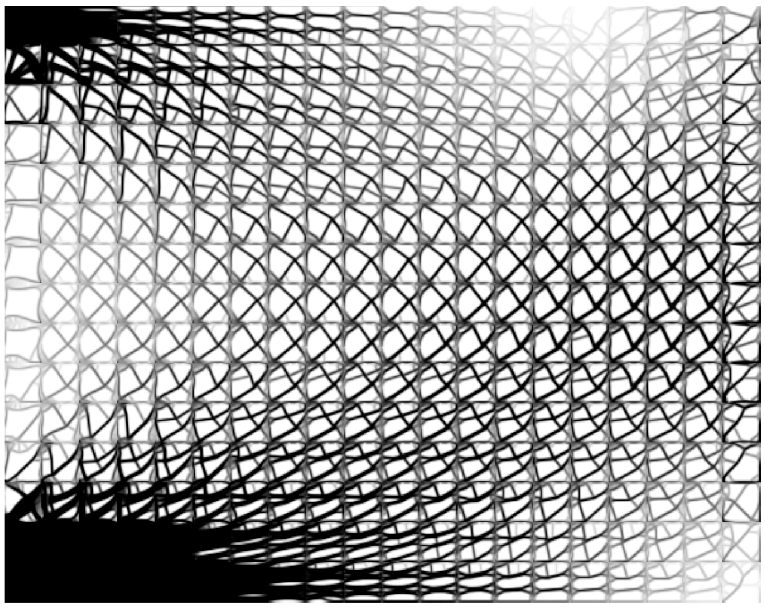


Figura 3.5: Dominio global óptimo de un problema simplificado generado por los resultados de las celdas.

Los resultados de este estudio pueden verse reflejados en la gráfica que aparece en la Figura 3.6, que recoge los datos mostrados en la tabla 3.1. Esta tabla, se alimenta de la experimentación a través de distintas pruebas. En cada una de estas pruebas, se considera un determinado número de clientes, y una selección aleatoria diferente de estos. También, se ha mostrado el estudio de tiempos en términos de *speed-up*, como medio para relativizar los resultados, este puede verse en la Figura 3.7.

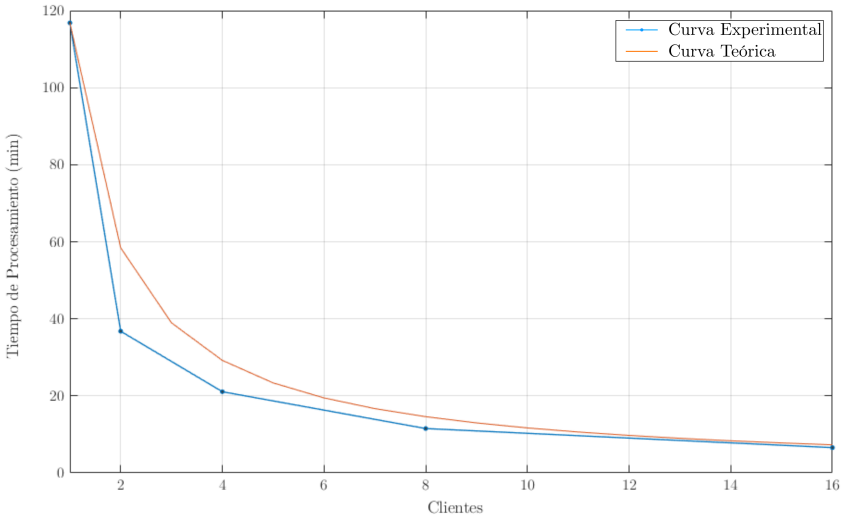


Figura 3.6: Curva de Tiempo de Procesamiento en función del número de clientes.

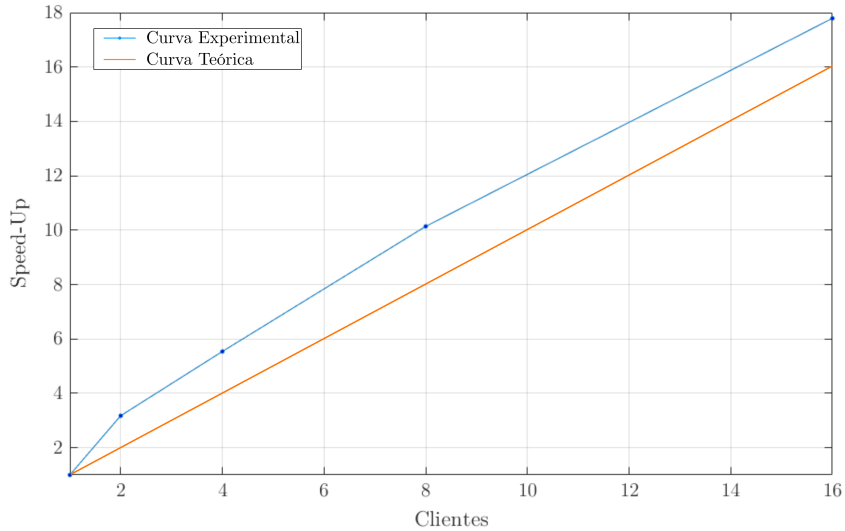


Figura 3.7: Curva de *Speed-up* en función del número de clientes.

Cabe destacar brevemente, que en el caso de un solo cliente la ejecución se ha

Clientes	16	8	4	2	1
E1	00:06:56	00:12:54	00:21:05	00:24:35	01:24:50
E2	00:05:33	00:08:04	00:16:07	00:52:07	01:41:53
E3	00:06:45	00:13:33	00:27:37	00:33:19	02:43:44
E4	00:06:51	00:11:32	00:19:33	00:37:03	-
Media Sexagesimal	00:06:34	00:11:31	00:21:05	00:36:46	01:56:49
Media decimal (min)	6,57	11,52	21,08	36,77	116,82
Speed-Up	17,79	10,14	5,54	3,177	-

Tabla 3.1: Tabla de tiempos.

realizado en serie en lugar de en paralelo, ya que carece de sentido práctico. Y también, que en el caso de un solo cliente se han hecho tres pruebas, porque son los tres tipos de equipos de los que se dispone. Por lo tanto, cada uno de los experimentos realizados en serie con un solo cliente, *E1, E2 y E3*, se ha realizado sobre un tipo de ordenador diferente. De este modo en caso de disponer más cliente, en cada uno de los experimentos, se ha tomado una selección aleatoria de los diferentes equipos. Esto explica el porqué se puede conseguir un *speed-up* por encima del máximo teórico, al igual que un tiempo de procesamiento menor que el teórico.
Las conclusiones que se pueden extraer de este estudio es que, según dicta el sentido común, a medida que se usa un mayor número de clientes, el tiempo de procesamiento es menor, en concreto, tiene una relación inversamente proporcional (con el doble de clientes, la mitad de tiempo, en el caso teórico). Sin embargo, se aprecia que la regla experimental que gobierna esta relación no es tan lineal como la teórica. Esta perturbación puede deberse a los tiempos invertidos en la conexión de los clientes y en el tiempo invertido por ciertas subrutinas para buscar nuevos clientes a los que conectarse, sin embargo el factor más decisivo es la heterogeneidad de los equipos a la hora de seleccionar grupos de trabajo para los experimentos. Si se estudia en detenimiento la tabla 3.1 se muestra que la selección de unos equipos u otros marca por completo el rendimiento de la red de comunicación creada. Por lo tanto, se concluye que la mayor variabilidad concerniente al tiempo de ejecución se debe a la selección de clientes. Con el fin de demostrar cuanto de vital es el criterio de selección de equipos, se rescata de la tabla 3.1 los resultados obtenidos mediante los experimentos $E3$ de la selección de cuatro clientes y el $E1$ de la selección de dos clientes. En este caso el primero de ellos invierte un mayor tiempo en la obtención solución que el segundo, a pesar de contar con el doble de clientes.
Puestos a obtener una referencia del potencial ahorro que se puede alcanzar con esta aplicación, se ha analizado el problema definido en su distribución inicial de densidades por la Figura 3.2 y el resultado obtenido a partir de ésta en la Figura 3.3. El resumen de tiempos de ejecución puede verse en la tabla 3.2. Para este análisis se ha tomado el equipo con las mejores características disponibles y se ha enfrentado a la paralelización del mismo problema en 23 ordenadores, todos los disponibles, con características heterogéneas.

Clientes	1	Todos (23)
Tiempo	15:41:23	00:49:50
Speed-Up	1	18,89

Tabla 3.2: Tabla de tiempos.

3.2. Optimización de forma con algoritmos híbridos

Como se ha explicado en anteriores secciones, la *hibridación* consiste en conectar varios algoritmos de optimización, de distinta naturaleza o no, con el fin de aprovechar las bondades de estos reduciendo al máximo sus debilidades. Se ha comentado anteriormente que los algoritmos metaheurísticos, como el genético, son fácilmente paralelizables, pero en cambio los que basan su actualización de variables en los gradientes de la función objetivo presentan algunas dificultades. Con el fin de paralelizar el proceso se considerarán diferentes métodos de actuación en función de la naturaleza del algoritmo, en caso de que se use un algoritmo de los primeros simplemente se repartirá cada individuo a un equipo diferente, pero en caso de los segundos, habrá dos opciones, se podrá lanzar un proceso completo en el servidor o se podrán lanzar varios procesos en equipos distintos, esta última opción facilita el encontrar un mínimo global. Puesto que en anterior apartado se ha realizado un estudio de tiempos referentes a la paralelización, en éste no será necesario, dando por hecho que se consigue un ahorro razonable y se estudiará solamente la hibridación.
El primero de los ejemplos que se mostrará en este apartado se lanzará un algoritmo híbrido formado por un algoritmo genético y uno basado en gradiente (SQP). Adicionalmente, se paralelizará el tramo del algoritmo de optimización gobernado por el genético, los motivos por los cuales solo se paralelizará esta parte y no al completo han sido expuestos anteriormente.
Los parámetros de configuración que gobiernan este ejemplo se muestran a continuación, mientras que el resultado que se obtiene puede ver en la figura 3.10:

- Problema: Cilindro hueco sometido a presión interna, ver Figura 3.8.
- Función objetivo: Volumen.
- Restricción: Tensión máxima admisible = 2 MPa.
- Variables de diseño: 1. (Representa el radio externo del cilindro).
- *Solver*: *ANSYS*.
- Diseño inicial (x_0): 15 m.
- Ejecución sin entrenamiento de *RNA*.
- Hibridación de algoritmos:

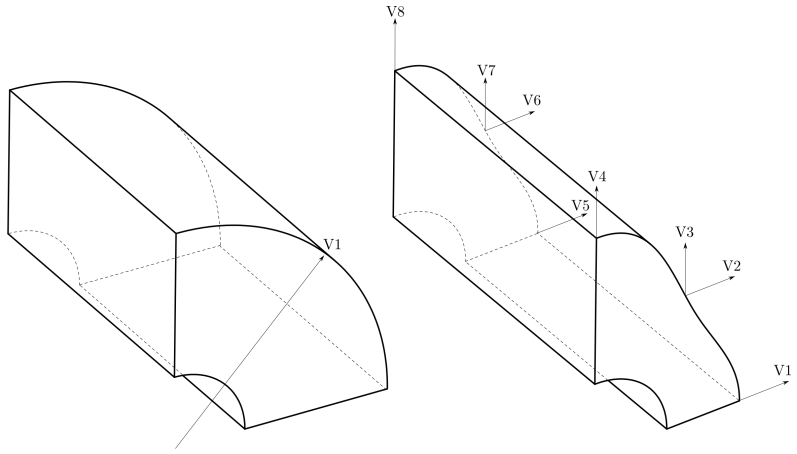


Figura 3.8: Representación esquemática del problema, a la izquierda, geometría definida con una variable de diseño y a la derecha, geometría definida con ocho variables de diseño.

- **Algoritmo Genético**:
  - Tamaño de población = 50 individuos.
  - Tolerancia = $1x10^{-4}$.
  - Número de generaciones máximas = 1.
  - **Computación en paralelo**:
    - 5 Clientes activos.
- **Algoritmo basado en gradiente**:
  - Algoritmo interno = SQP.
  - Tolerancia = $1x10^{-6}$.
  - Rango de acción = Local.
  - Convergencia por tolerancia.

Algunos de los parámetros anteriores merecen ser detallados, no tanto su significado sino el porqué de tomar unos valores o otros. Para empezar, el tamaño de la población es relativamente alto para un problema que solo consta de una variable de diseño el motivo por el cual se ha hecho una selección de un número de individuos tan elevado es debido a la paralelización. Si se toman valores de población excesivamente pequeños, los clientes calcularán una población en muy poco tiempo y además cabe la posibilidad de que dentro de esa población existan muchos individuos repetidos, disminuyendo el número de tareas a paralelizar notablemente. Con esto, se consigue que la mayoría de clientes pasen una gran parte de su tiempo o en espera o re-calculando individuos que ya han sido enviados a otro cliente, por el mero hecho de quedarse sin tareas con mucho rapidez. Se desconoce cual es la cantidad de individuos óptima para este problema con la configuración que la define, habitualmente se toman valores de 10 veces el número de variables de diseño [22].

En este ejemplo, se han considerado esa misma referencia pero multiplicándola por el número de clientes activos, que como se ve en la definición de parámetros es 5. Este nuevo parámetro, el número de clientes, se ha definido como 5 por un mero hecho de visualización de resultados, los que aparecen en la Figura 3.9. En cuanto a la tolerancia, que representa el criterio de parada habitual a no ser que se indique otro específicamente, en el primer algoritmo se ha tomado un valor relativamente alto puesto que este primer tramo solo servirá para concretar una zona en la que es posible que se encuentre el mínimo global, al contrario que con el segundo algoritmo que sirve para afinar lo máximo posible al encontrar el mínimo global, por eso el valor es más alto en este caso.
El criterio de parada es el parámetro que define, cuándo y porqué se debe pasar de un algoritmo al siguiente, en caso de un algoritmo genético este parámetro puede considerarse como el número máximo de poblaciones a estudiar antes de converger, éste toma el valor de '1' porque con una generación realizada sobre el campo de posibles soluciones de una variable de diseño es suficiente para conocer por donde se encuentra el mínimo global. El último de los parámetros que caben ser detallados es el de rango de acción del algoritmo basado en gradiente, éste toma el valor de 'Local' (pudiendo ser 'Local' o 'Global') porque el algoritmo genético se ha encargado de confinar la zona en la que se encuentra el mínimo global. Debido a esto el algoritmo basado en gradiente puede ejecutarse de modo que haga una búsqueda de soluciones local en lugar de global por todo el campo de soluciones.
Este ejemplo se ha ejecutado para representar como avanza la ejecución del algoritmo, por eso se ha tomado una única variable de diseño, ya que ésta puede ser fácilmente representada en un gráfico. En concreto, la ejecución de este análisis da como resultado el gráfico que puede verse en la Figura 3.9. En este gráfico aparecen dos secciones fácilmente reconocibles, la primera de ellas, el algoritmo genético. Éste se ejecuta al comienzo del algoritmo de optimización híbrido, el porqué de la aparición de varias curvas se debe a la paralelización. Cada una de las curvas en esta sección representa la evolución de la solución en un cliente distinto, por ese motivo no sería correcto hablar de iteraciones sino de un concepto que represente las tandas de reparto de tareas desde el servidor a los clientes. En la figura, se ve que esta paralelización permite considerar como máximo tantos individuos como clientes activos. En la gráfica también se aprecia un concepto explicado anteriormente, y es el que aparece en la quinta población a repartir, o si se mira del gráfico, quinta iteración, cuatro de los clientes convergen sobre un mismo punto, esto se debe a la inexistencia de nuevas tareas a realizar de modo que se re-envían individuos de los que no se dispongan resultados todavía a los clientes parados. Este criterio de funcionamiento no es nada más que un red de seguridad, en caso de que se produzca un error en un cliente este mismo cálculo ya está razonablemente avanzado en otro cliente. La convergencia final de todos los clientes se debe a la convergencia propia del algoritmo genético, a pesar de que en solo una generación es imposible considerar un valor óptimo, si que se dispone de una población más evolucionada que la anterior, y focalizada en una zona concreta del espectro de valores de la variable de diseño.

Puesto que para el próximo tramo se ejecutará un algoritmo basado en gradiente, se debe reducir el tamaño de la población a un solo individuo que será el x_0 del próximo algoritmo. De todos los individuos simplemente se escoge el mejor de todos ellos.
Una vez concluida la primera parte del proceso, se pasa a la segunda sección del algoritmo de optimización, la que está gobernada por un algoritmo cuya actualización de variables basada en los gradientes del problema (función objetivo y restricciones). En este tramo las iteraciones si que tienen el significado propio de la palabra y todas ellas se están realizando sobre el servidor. Existen distintas configuraciones de funcionamiento como que esta parte se haya realizado sobre el mejor de los clientes anteriormente conectados, basando su rendimiento en función de tiempo de cálculo o también se podría haber ejecutado el mismo algoritmo con variaciones en su x_0 en cada uno de los clientes. Mediante este último método se alcanzaría un mayor grado de seguridad a la hora de saber si el óptimo alcanzado es el global. Como se ve en la gráfica, la variación no es excesivamente amplia a lo largo de su ejecución, esto es debido a que el algoritmo genético a suministrado un x_0 muy próximo al mínimo global real. Por este mismo motivo, se ha realizado un ampliación situada en la misma figura para que se observe la evolución del algoritmo conforme avanzan las iteraciones. Se puede ver como oscila sobre el valor óptimo hasta que finalmente converge, la amplitud de esta oscilación puede ser más o menos controlada modificando los parámetros de penalización de individuos, pero no podrá ser evitada por completo a no ser que se utilice un método de barrera y se descarten los individuos que no cumplan la restricción. El algoritmo converge en una solución definida por la variable de diseño tomando el valor de $11,85$, este valor, se puede comparar con el mínimo analítico [23], siendo este igual a $10,67$, representado en la gráfica por la línea horizontal de color rojo. La diferencia entre ambos valores puede verse principalmente afectada por el error de discretización llevado a cabo en la resolución numérica del problema.
Asimismo, el resultado obtenido como resultado de la ejecución del anterior análisis se puede ver en la Figura 3.10. En ella, se muestra la geometría inicial aleatoria (gris) y la geometría óptima (azul). Con el fin de comprobar que no existen limitaciones espaciales en cuanto al número de variables de diseño, se ha realizado el mismo ejemplo que el visto anteriormente, sin embargo, en este caso se ha parametrizado la superficie externa con ocho variables de diseño en lugar de con una, que corresponde al ejemplo visto anteriormente.

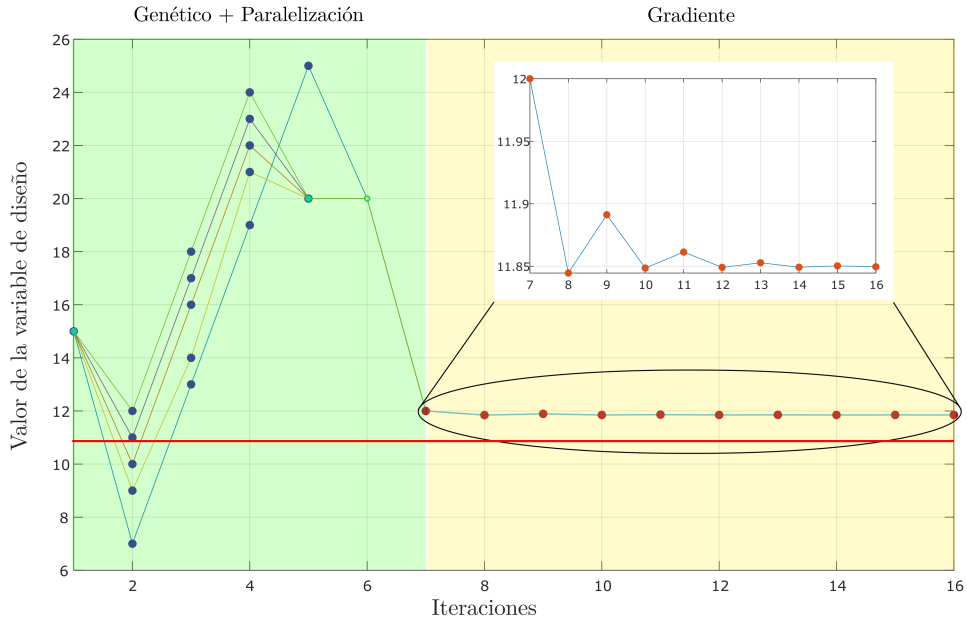


Figura 3.9: Curva de valores que toma la variable de diseño conforme avanza el algoritmo de optimización. 1º) Algoritmo genético y evolución de la solución en cada procesador. 2º) Algoritmo tipo gradiente.

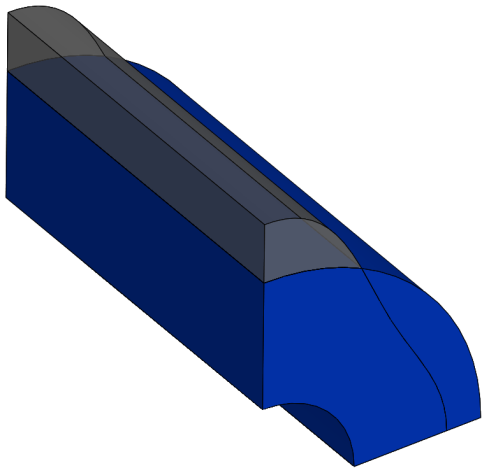


Figura 3.11: Solución del problema (en azul) y la geometría original (en gris).

Cabe mencionar que se han modificado ligeramente los parámetros de ejecución en este último ejemplo. En concreto, se ha modificado la cantidad de individuos por población, que ha pasado a ser de 100 individuos. Del mismo modo, que se han cambiado la cantidad de poblaciones máximas, siendo este ahora de 10 poblaciones. Y por último, se ha actualizado el valor y cantidad de las variables de diseño de

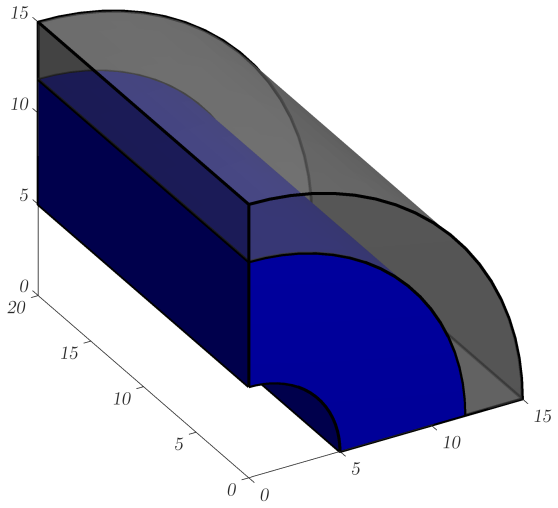


Figura 3.10: Solución del problema (en azul) y la geometría original (en gris).

uno a ocho y tomando un x_0 igual a $[10, 7, 7, 15, 10, 7, 7, 15]$. También cabe añadir que se han usado una serie de restricciones lineales entre las variables de diseño en este problema para reducir el campo de soluciones, junto con limitaciones inferior y superior en el valor que pueden alcanzar estas variables, ver Figura 3.12.

	Variables	Lower b.	Upper b.
1	V1	10	15
2	V2	7	12
3	V3	7	12
4	V4	10	15
5	V5	10	15
6	V6	7	12
7	V7	7	12
8	V8	10	15

	Constant	Variable	Constant	Variable	Operator	Constant
1	-0.7500	V1	1	V2	<=	-0.5000
2	1	V2	-1	V3	<=	0.1000
3	1	V1	-1	V4	<=	0.1000
4	-0.7500	V4	1	V3	<=	-0.5000
5	-0.7500	V5	1	V6	<=	-0.5000
6	1	V6	-1	V7	<=	0.1000
7	1	V5	-1	V8	<=	0.1000
8	-0.7500	V8	1	V7	<=	-0.5000

Figura 3.12: Límites inferior y superior de las variables de diseño (izq.) y restricciones lineales entre estas (der.).

3.3. Optimización con *RNA* y el resto de mejoras

Nota: Este problema se ha paralelizado teniendo en cuenta los servidores de cálculo en lugar de los ordenadores de las aulas. Esto es debido a un problema técnico puntual. El procedimiento es exactamente igual salvo que la eficiencia se verá reducida, puesto que la frecuencia de sus procesadores no es tan alta.

En este último aparatado de resultados, nos serviremos de otro problema para estudiar el uso de RNA, puesto que se han mostrado los beneficios de las anteriores mejoras, también se implementarán en este ejemplo.
En concreto, ese problema será una biela, definida como en la Figura 3.13, es decir considerando una simetría transversal para simplificar los cálculos, aplicando una presión en la superficie cilíndrica, y empotrando uno de los puntos para eliminar desplazamientos de sólido rígido.
Los parámetros dimensionales del problema son: $AB = 11$, $C = 4$, $AD = 20$,

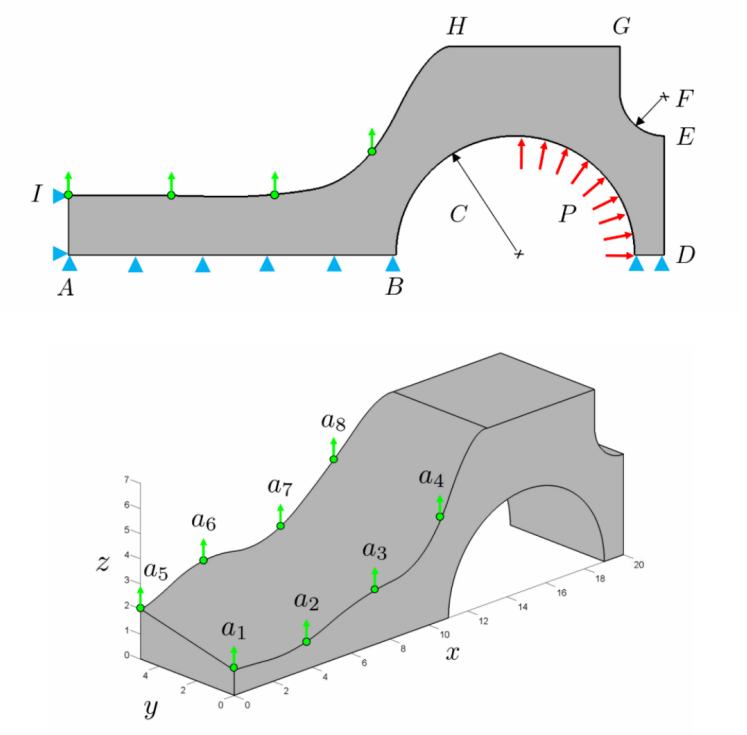


Figura 3.13: Definición del problema de la biela. Arriba, las dimensiones, las condiciones de contorno y las cargas aplicadas. Abajo, la parametrización realizada sobre la superficie externa, *Nota: se ha considerado simetría entre las varibles de diseño para reducir el campo de soluciones, tomando cuatro variables independientes.*

$DE = 4$, $F = 1{,}5$, $DG = 7$, $HG = 5{,}5$. El módulo de *Young* es $E = 10^5$, y el coeficiente de *Poisson* $\nu = 0{,}333$. La presión es $P = 100$ en la dirección normal al medio arco mostrado en la Figura 3.13.
Como se ha comentado anteriormente, este problema esta definido por cuatro variables de diseño, siendo éstas las encargadas de definir la superficie externa de la geometría. Con el fin de restringir el espacio de soluciones, se ha definido un límite inferior tal que, $[1, 1, 1{,}2, 7]$; mientras que el límite superior es, $[7, 7, 7, 7]$. Siendo ambos vectores columna con cuatro elementos, cada uno de ellos cerca el espacio de variación para cada una de las variables. Adicionalmente, se han definido una serie

de restricciones lineales entre las variables de diseño de tal forma que cada una de ellas debe ser mayor o igual que la anterior, es decir, tomaría la siguiente representación matemática: $V_1 \leq V_2 \leq V_3 \leq V_4$. También, se ha añadido una restricción no lineal, que hace referencia a la tensión máxima admisible, siendo esta igual a 900, tomando el valor del límite de fluencia, S_y.
En base a la anterior definición y considerando todas las restricciones, la solución de dicho problema es la que aparece en la Figura 3.14, estando la geometría definida por las variables de diseño tomando los siguientes valores: $[1, 1, 1{,}2, 2{,}7862]$.
Cabe destacar el hecho de que las dos primeras variables tomen el valor de los

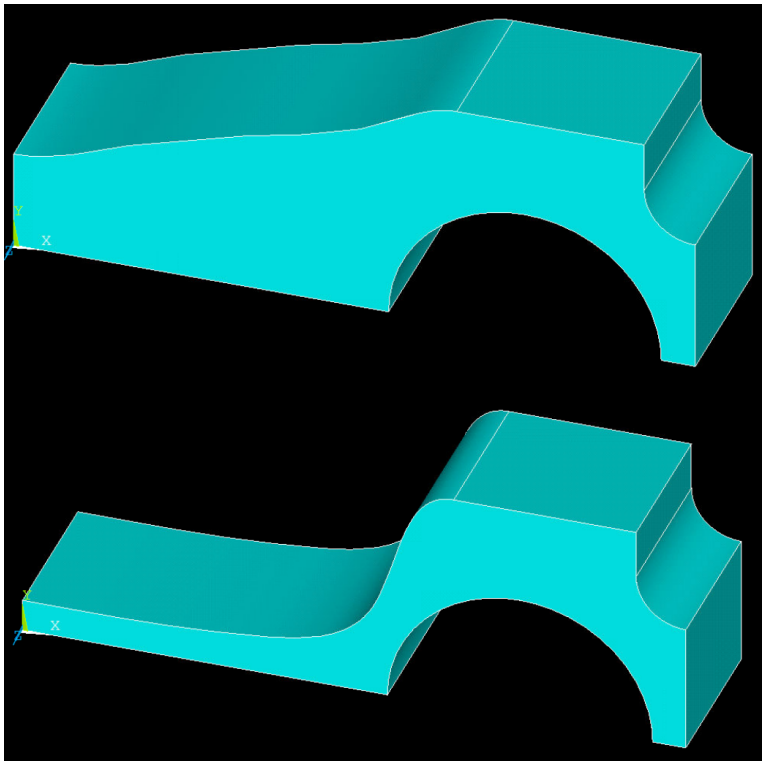


Figura 3.14: Solución del problema de la biela. Arriba, la geometría inicial, abajo, la geometría óptima.

límites inferiores, esto es debido a que esa zona de la geometría no está sometida a una gran carga, concluyendo que existe un potencial de mejora si se reduce aún más el valor de dichas variables. En cuanto a las otras variables, la tercera también toma el valor límite inferior a pesar de definir, junto con la cuarta variable la zona curva de la superficie donde es factible que aparezca un concentrador de tensiones, debido a esto, la variación de estas variables definará la existencia o no de una singularidad en la geomtría. Esto, tiene como consecuencia, en el algoritmo de optimización, que un pequeña variación de una de estas variables puede generar una gran evolución de la tensión a la que la pieza está sometida, generando unos gradientes desproporcionados en relación a la variación de la variable. Lo cual, se traduce en que la curva de

convergencia no es suave sino que esta formada por picos. Rescatando la evolución de la función objetivo y restricción de la sección gobernada por el algoritmo de gradiente, se muestra el tipo de convergencia anteriormente descrito, ver Figura 3.15.
El anterior análisis se ha ejecutado en base a la siguiente configuración y se tomará

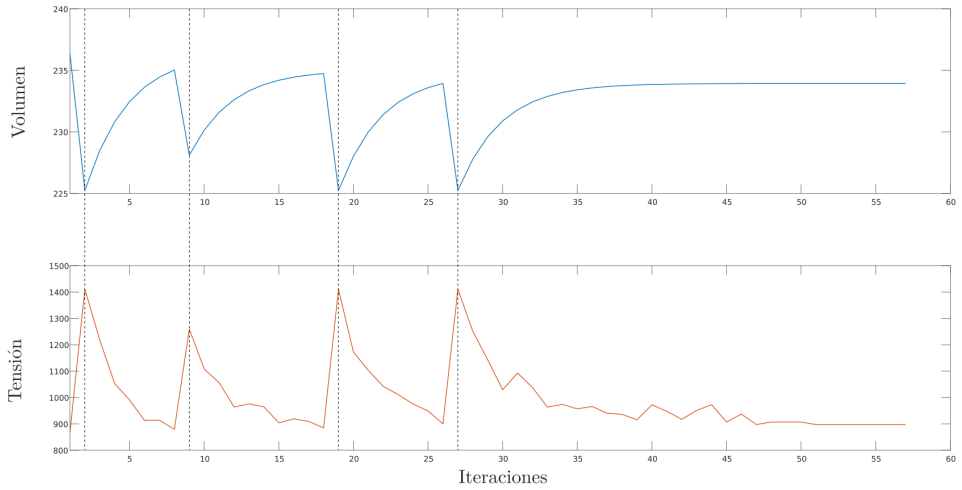


Figura 3.15: Evolución del análisis en la sección del algoritmo gradiente. Arriba, la función objetivo y abajo, la restricción.

como referencia para comprobar como afecta el uso de *RNA*, por ello en éste no se considerarán:

- Valor inicial, $x_0 = [3, 4, 5, 6]$.
- Hibridación:
  - Algoritmo Genético:
    - 75 individuos por población.
    - 5 poblaciones hasta convergencia.
    - La tolerancia toma un valor de $1x10^{-4}$.
    - Paralelización:
      - 4 servidores de cálculo.
    - *ANSYS* como *software* de cálculo.
  - Algoritmo de gradiente:
    - Búsqueda local de soluciones.
    - *SQP*, como algoritmo de actualización.
    - Tolerancia con valor de $1x10^{-6}$.
    - *ANSYS* como *software* de cálculo.

A parte, del anterior análisis, que se tomará como referencia de la fiabilidad de las *RNA*, se han ejecutado dos análisis más con 2 configuraciones de *RNA*, en ambas se ha considerado el mismo número de análisis permitidos seguidos mediante el uso de las *RNA*, que es 25, pero en uno de ellos se han tomado 50 valores de análisis como suficientes para su entrenamiento, mientras que en otro se han considerado 75.
En base a lo anterior, se han recopilado todos los datos relevantes de cada uno de estos análisis, generando la tabla 3.3 con el fin de hacer una comparación entre estos. Basándose en los resultados obtenidos y representados en la tabla 3.3, se

	No RNA	***RNA 50/25***	***RNA 75/25***
Solución	[1 1 1,2 2,7862]	[1 1 1,2 2,9148]	[1 1 1,2 2,8240]
Algoritmo Genético			
Paralelo	594 ind.	395 ind.	475 ind.
Serie	0 ind.	0 ind.	0 ind.
RNA	0 ind.	151 ind.	140 ind.
Algoritmo Gradiente			
Serie	57 ind.	2 ind.	41 ind.
RNA	0 ind.	11 ind.	8 ind.
Total	651 ind.	559 ind.	664 ind.
Tiempo	00:58:31	00:26:18	00:45:36
Error	0 %	3,84 %	1,13 %

Tabla 3.3: Tabla de resultados.

pueden extraer una serie de conclusiones que se resumen a continuación:

- Las *RNA* representan un ahorro sustancial en coste computacional y en consecuencia, en tiempo, comparándolo con un problema que no use este tipo de modelos de predicción.
- Ese ahorro de tiempo no es gratis y se debe suplir con un aumento en el error de la solución obtenida, respecto a una referencia dada.
- Si se consigue un cierto balance entre ahorro y precisión, como puede ser el último caso, se puede conseguir un ahorro de tiempo nada despreciable frente a un error totalmente asumible.
- En cuanto al número de individuos evaluado en los distintos ejemplos, no parece que las *RNA* tengan un efecto directo en el número de individuos considerado.

4 Conclusiones

Durante la realización de este trabajo se han desarrollado y analizado una serie de herramientas computacionales que permiten manejar problemas que suponen un número elevado de análisis de MEF en un tiempo de ejecución razonable, sobretodo problemas relacionados con la optimización de componentes estructural a través de técnicas de paralelización. Todo ello se ha desarrollado usando los recursos informáticos que podrían estar disponibles en el Área de Ingeniería Mecánica del Departamento de Ingeniería Mecánica y Materiales. Las conclusiones obtenidas a partir de la implementación de cada una de estas herramientas serán descritas a continuación:

- La conclusión más obvia y directa en cuanto al método de paralelización es que da como resultado una disminución dramática en el tiempo de ejecución de la mayoría de problemas. Por supuesto, estos problema han de ser paralelizables, siendo éste el único de los inconvenientes encontrados. Los *speed-up* alcanzados en algunos ejemplos muestran el potencial del software desarrollado que permitirá sacar un mayor provecho a los recursos computacionales (aulas de informática) del Área de Ingeniería Mecánica.

- En referencia a la hibridación de algoritmos, se ha mostrado que se pueden beneficiar algoritmos de distinta naturaleza entre sí. En los ejemplos vistos, el algoritmo genético es capaz de computar con eficiencia la gran mayoría del campo de soluciones, direccionando al algoritmo de gradiente hacía una zona con alta probabilidad de encontrar un mínimo global.

- Se ha mostrado el ahorro de tiempo que se consigue mediante el uso de modelos subrogados, en concreto de las *Redes Neuronales Artificiales*. A pesar de la clara disminución del coste computacional, se debe tener en cuenta que este tipo de modelos matemáticos producen cierto error, el cual debe ser controlado para que los resultados obtenidos se puedan considerar fiables.

- Cabe destacar que mediante la implantación tanto, del modelo de paralelización como del uso de las *RNA* el usuario final es capaz de trabajar y lanzar

análisis que requieran ciertos requisitos computacionales, en un ordenador con unas características mucho más modestas que si necesitara ejecutar esos mismos cálculos en su propio ordenador.

- El programa implementado puede ser utilizado también en otros tipos de cálculo en ingeniería que sean paralelizables además de en problemas de optimización, como por ejemplo la resolución de un sistema de ecuaciones mediante *Nested Domain Decomposition*.

5 Trabajos Futuros

Tras la realización e implementación de esta aplicación y, sobretodo, a través de un gran uso de éste, es inevitable observar posible focos de mejora en nuevas versiones de la aplicación, entre otras cabe destacar las siguientes:

- La mejora fundamental que debe llevarse a cabo para que el uso de la aplicación de paralelización pueda ser facilmente por los usuarios es la de tener control sobre los clientes. Esto se traduce en adquirir permisos administrativos sobre las aulas de informática, algo que debe ser meditado ampliamente. Con estas nuevas funcionalidades el usuario será capaz de arrancar y apagar equipos de un modo remoto, así como lanzar *scripts* que ejecuten ciertas instrucciones, por supuesto, todas estas tareas serán transparente al usuario.
  Hasta ahora el procedimiento habitual de funcionamiento es el arranque manual de los equipos. Una vez arrancados, se realiza una conexión por escritorio remoto con un determinado usuario de *Windows* a cada uno de los clientes. Con el acceso a cada equipo, ahora es posible ejecutar *MATLAB* y posteriormente lanzar el *script* que gestiona la conexión con el servidor. Un proceso nada práctico.

- Debido a la inevitable heterogeneidad de los equipos usados como clientes de cálculo, se ha comprobado que las características individuales de cada uno de ellos influyen en exceso en el rendimiento final obtenido por la paralelización. Partiendo de la anterior afirmación, sería muy recomendable desarrollar un sistema de clasificación de equipos, asignándoles distintos niveles de prioridad en función de su rendimiento. De esta forma se trataría de enviar la mayoría de tareas a los equipos con un funcionamiento superior. La implementación de un procedimiento adicional que sea capaz de asignar pesos a las tareas a ejecutar en función de su complejidad, crearía sinergias con la anterior mejora incrementando en mayor medida, si cabe, el rendimiento de la aplicación.

- Actualmente esta herramienta permite paralelizar cierta cantidad de problemas diferentes, y en esencia, todos los vistos en este trabajo tienen una natura-

leza iterativa, mediante la cual cada cliente se encarga de calcular un problema completo. La potencial mejora que se puede alcanzar es la de paralelizar partes de un mismo problema extremadamente pesado.

- Otro de los items que deben ser estudiados en profundidad en el momento de desarrollar una nueva versión de esta aplicación es la gestión de los cálculos por parte del cliente. Actualmente, si el problema a calcular es independiente de *MATLAB*, ya sea porque se calcula en un software externo, como por ejemplo *ANSYS* o *ABAQUS*, o porque se ejecute un código compilado, en ningún momento se pierde el control sobre el cliente y la comunicación no debería tener ningún *delay*. En cambio, si se trata de ejecutar un problema en *MATLAB*, éste bloqueará la linea de comandos y por tanto cortará la comunicación hasta que el cálculo haya concluido. La solución implementada en esta versión es que todo el cálculo que deba ser lanzado a través de *MATLAB*, será enviado a un *worker* y a su vez, se creará un *job*, para no perder la trazabilidad del problema. De esta forma el cálculo será lanzado en segundo plano a la actual sesión de *MATLAB*, que está siendo ocupada por la comunicación, aprovechando los distintos núcleos de procesamiento que disponen los ordenadores.

- Además, como posible foco de mejora, es la sustitución del objeto *timer* como procedimiento gobernante del proceso de comunicación. Esto es debido a que si se introduce un periodo de ejecución excesivamente pequeño es posible ralentizar o incluso bloquear una cantidad de tiempo demasiado grande, lo cual puede imposibilitar la correcta comunicación mediante el servidor y los clientes. Sin embargo, es un método muy cómodo para comprobar el estado de los clientes, sobretodo de los caídos, y también para descubrir nuevos clientes disponibles.

6 ANEXOS

6.1. Manual de Usuario

Nota: Esta versión del manual de usuario se ha realizado en base a la actual versión del programa, en caso de actualización importante, este manual deberá ser renovado.

6.1.1. Instalación

Para el correcto funcionamiento se deberá copiar la carpeta que contiene el programa de optimización al completo en un directorio de usuario, intentado evitar directorios de administrador. Esto se debe a que el programa tratará de crear y eliminar una serie de archivos y es posible que en un directorio de este tipo se necesiten permisos más elevados o de administrador para realizar esas tareas.

6.1.2. Interfaz de usuario

A continuación se presenta la interfaz de usuario que gobierna las funcionalidades del programa, ver Figura 6.1. Se explicarán por encima algunas de las opciones que están disponibles, aunque se verán más adelante cuando se haga un ejemplo de ejecución.
La interfaz tiene un menú desplegable en la parte superior con una serie de opciones. La primera de ellas, *file*, permite abrir una nueva instancia así como cargar una configuración ejecutada anteriormente desde memoria. También permite guardar la configuración de algoritmos en el directorio que cada usuario desee con el nombre personalizado a modo descriptivo. Y, por supuesto, también permite cerrar la interfaz de usuario, ver Figura 6.2.
La siguiente opción disponible en el menú superior es la de *edit*, ver Figura 6.3, ésta al desplegarse nos da la opción de modificar las restricciones lineales existentes entre las variables de diseño y también especificar los límites inferior y superior que toman estas variables.

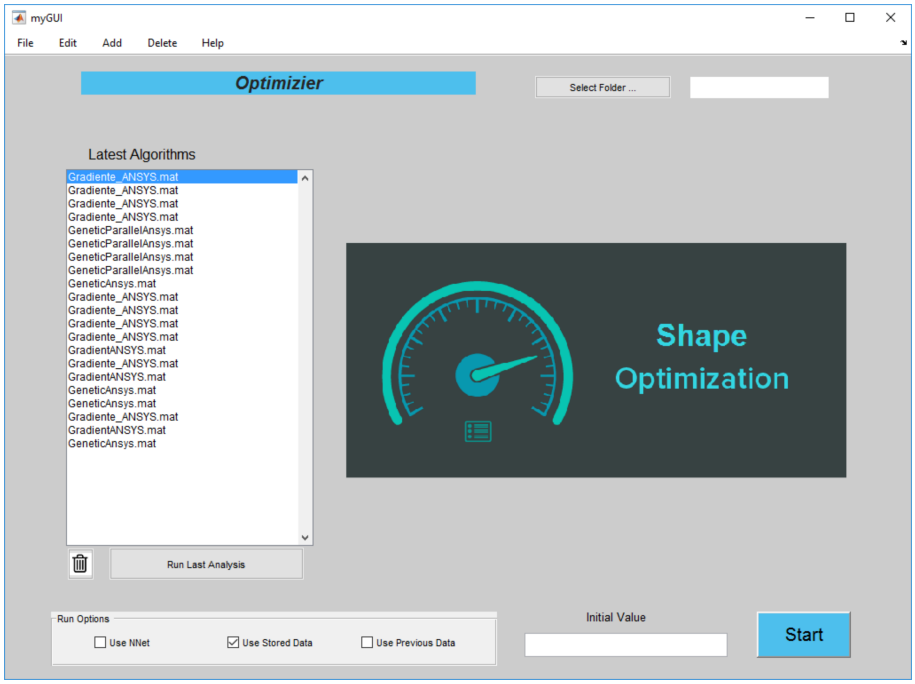


Figura 6.1: Interfaz de usuario del programa.

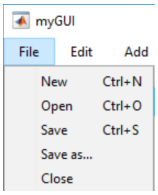


Figura 6.2: Menú desplegable de la opción *file*.

Si se *clicka* sobre la único opción disponible actualmente, aparecerá una ventana

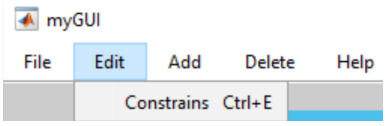


Figura 6.3: Menú desplegable de la opción *edit*.

nueva, ver Figura 6.4, en ella aparecen dos tablas. La primera de las tablas sirve para definir las restricciones de caja de las variables de diseño, estas restricciones hacen referencia a los valores límite inferior y superior que estas pueden tomar. La segunda tabla se utilizada para definir las relaciones lineales que existen entre las distintas variables, seleccionando las variables, el operador necesario y las constantes que entren en juego.

Con el fin de facilitar su uso, ya que la definición de restricciones es algo laborioso, se han creado una serie de opciones para poder cargar de memoria, así como de un

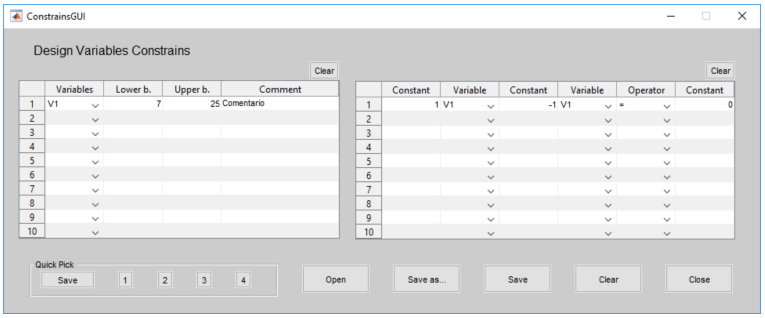


Figura 6.4: Ventana emergente de edición de restricciones.

menú rápido que almacene las restricciones más usadas.
Continuando en el menú superior, la siguiente funcionalidad es la de crear los distintos tipos de algoritmos disponibles, en concreto se podrá crear un algoritmo genético o un algoritmo basado en gradiente (*fmincon*), ver Figura 6.5.
En función de la opción que *clickemos* se creará un tipo de pestaña o otro. Es im-

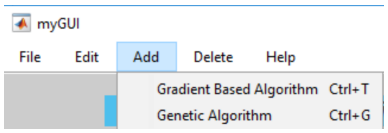
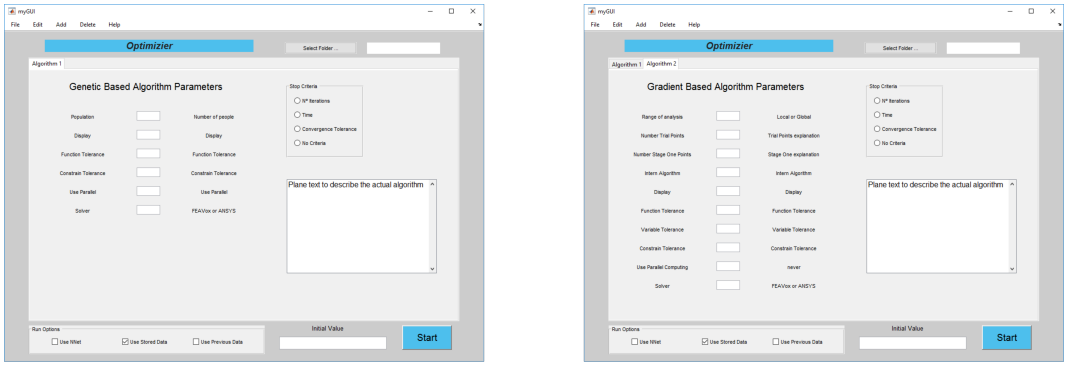


Figura 6.5: Menú desplegable de la opción *add*.

portante conocer de antemano el orden en el que se desean ejecutar los algoritmos, puesto que el orden en el que los creemos será el orden en el que posteriormente se ejecuten, una vez creados no se podrá modificar el orden. Los dos algoritmos posibles se muestran en la Figura 6.6.



a) b)

Figura 6.6: Configuración de dos algoritmos diferentes: a) Algoritmo genético; b) Algoritmo basado en gradiente.

En el momento que se ejecute un problema de ejemplo se detallarán cada uno

de estos parámetros y cuales suelen ser los valores que habitualmente estos toman. En la Figura 6.7, se muestran el resto de funcionalidades que serán detalladas a continuación:

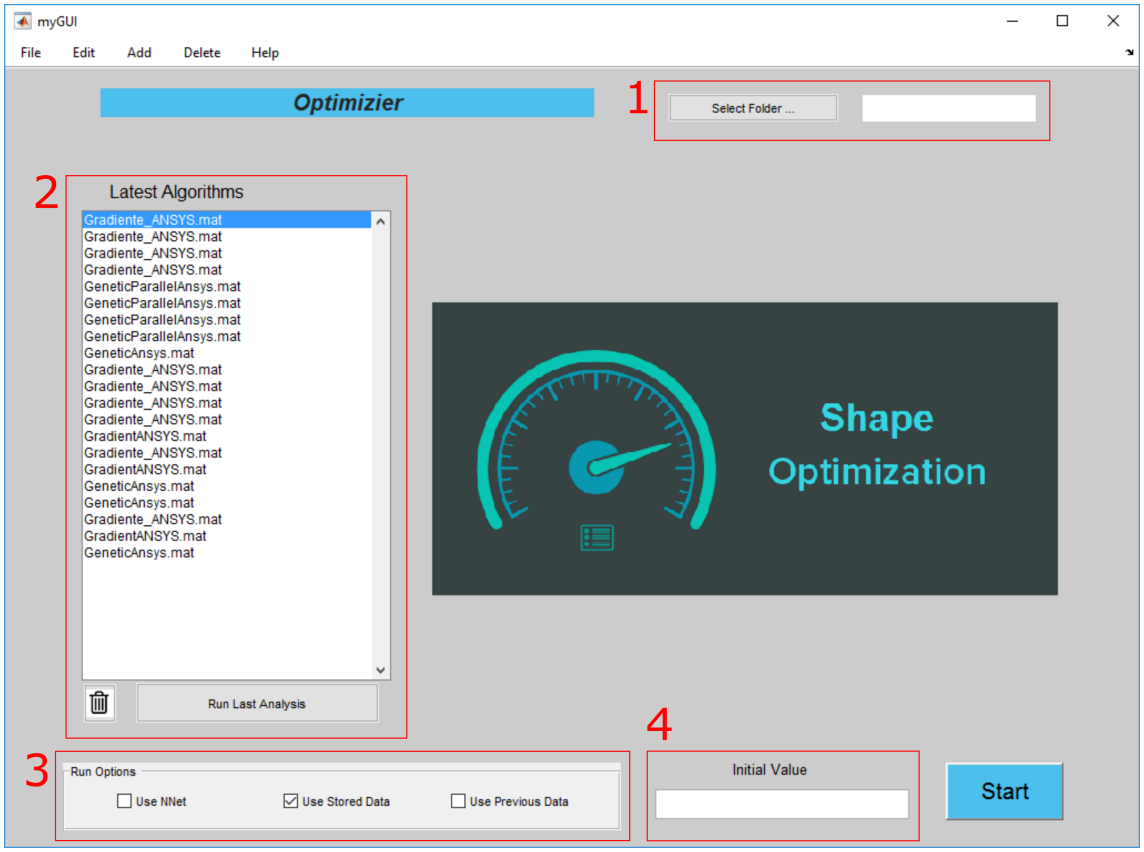


Figura 6.7: Funcionalidades de la interfaz de usuario.

1. Esta opción permite al usuario elegir el directorio del problema, en el momento de *clickar* aparecerá una ventana emergente para que se seleccione el directorio de trabajo. En él se almacenarán todos los resultados obtenidos a lo largo del proceso de optimización. También, se buscarán los ficheros del problema necesarios para la correcta ejecución.

2. Se ha creado un *widget* mediante el cual aparece un listado de las últimas configuraciones de algoritmos ejecutadas, seleccionando cualquiera de éstas, se cargará directamente la configuración. Existe la posibilidad de borrar el historial de configuraciones y también la de lanzar la ultima ejecución completamente configurada con todos los parámetros. La diferencia entre la última opción y el listado de configuraciones es que ésta, sólo almacena la información de los algoritmos, el resto se debe volver a seleccionar.

3. En la sección de opciones de ejecución se seleccionan una serie de funcionalidades que se usarán durante el proceso. Por defecto, la opción de recuperar de la

matriz de resultados un individuo que ya se haya ejecutado está seleccionada, se puede desmarcar sin problema, aunque no es recomendable puesto que estos algoritmos tienden a llamar un cierto número de veces a los mismos individuos. También, se puede seleccionar la opción de entrenar y usar una *RNA*. Y, por último, la opción de comenzar desde una ejecución previa que no ha llegado a concluir por cualquier motivo, para que este proceso sea consistente se debe configurar de la misma manera que la ejecución que se pretende continuar.

4. Los algoritmos usados en esta aplicación necesitan una sugerencia inicial para comenzar el proceso iterativo de optimización, se introducirá ese vector en el espacio reservado para ello. En caso de que el problema este definido por más de una variable de diseño, éstas se deberán introducir de la siguiente forma: $[V_1, V_2, ..., V_n]$.

Una vez configurado el problema, solo quedará *clickar* en el botón *Start* para comenzar la ejecución.
En el momento de lanzar la aplicación, si se ha seleccionado la funcionalidad del cálculo en paralelo, aparecerá una nueva ventana, ver Figura 6.8.
En ésta, se muestra un listado de los clientes seleccionados, junto a ellos, aparece un indicador luminoso, éste representa cuatro de los estados que pueden tomar los clientes:

- **Rojo**, desconectado.
- **Verde**, conectado y a la espera.
- **Amarillo**, conectado y trabajando.
- **Negro**, conectado y tarea eliminada.

Adicionalmente, en la ventana también aparece una barra de progreso que representa las tareas realizadas en relación al total de éstas. Por si esto no fuera lo suficientemente claro, se ha acompañado la barra con un indicador numérico que representa las tareas realizadas en términos porcentuales. Justo a esto, también se ha insertado un temporizador que informa del tiempo que la comunicación lleva trabajando.

6.1.3. Configuración

En este apartado se explicará la configuración previa necesaria para que el programa funcione correctamente.
Si el problema que queremos lanzar se va a calcular en un solo ordenador no sería necesaria ninguna configuración adicional más que la propia del problema. Ésta hace referencia al análisis de elementos finitos efectuado:

- En caso de querer ejecutar *ANSYS* como *software* de cálculo, se debe tener en cuenta previamente que para el correcto funcionamiento debe existir en el

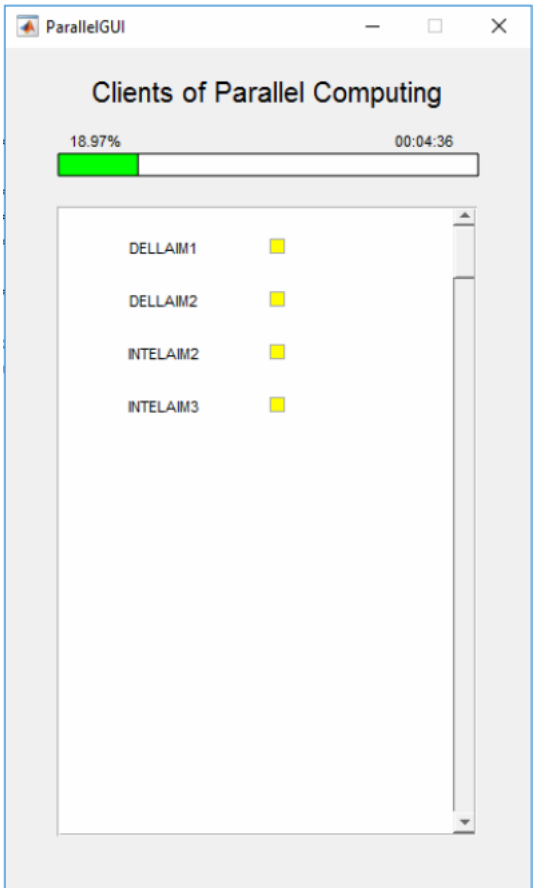


Figura 6.8: Ventana informativa de la comunicación en paralelo.

directorio de trabajo una macro correctamente parametrizada. La implementación actual de lectura y posterior generación de macros en *ANSYS* funciona en base a dos ficheros, por un lado una macro parametrizada y por otro, un fichero de texto en el que se definen cuáles son esos parámetros a sustituir. El programa se encargará de buscar el lugar de cada uno de los parámetros en la macro y sustituirlo por el valor de la variable de diseño correspondiente.

- En caso de que se vaya a usar *cgFEM* como *software* de cálculo, se debe disponer de un fichero *".mat"* que describa la geometría, así como las condiciones de contorno correctamente definidas en las funciones específicamente creadas para tal fin.

Sin embargo, si se pretende lanzar un problema que necesite paralelización, es necesario copiar las funciones que se encargan de la comunicación entre los ordenadores en cada uno de los clientes que vayamos a utilizar, en caso de que no estuvieran ya disponibles en el directorio de trabajo. Si estos ficheros ya se encuentran en los clientes, el programa automáticamente comprobará la versión y los actualizará en caso de que sea necesario. Por supuesto, tanto el servidor como los clientes deben

tener una conexión funcional a *internet*.

6.1.4. Resultados

El siguiente apartado detalla cuáles son los resultados obtenidos mediante la ejecución del programa y que significa cada uno de ellos.
Los resultados se almacenan en un directorio creado en base a la ruta especificada del problema, en ésta se creará una carpeta llamado *Results* y dentro de ésta, se crearán tantas carpetas de resultados como procesos de optimización se lancen. El nombre de éstas carpetas se formará en función de la fecha actual para que sea lo suficientemente descriptivo.
Si se lanza un problema básico de optimización, que no utilice la paralelización, en la carpeta anteriormente mencionada se crearán tres ficheros:

- Un fichero de texto llamado *Iteration Data.txt*, en el se almacena la información de cada una de las iteraciones que el programa ha realizado, como por ejemplo el valor de las variables de diseño, el valor de la función objetivo y restricciones, cual ha sido el método usado para obtener los resultados, etc.

- Otro fichero de texto llamado *Solution Data.txt*, en él simplemente se almacena toda la información disponible del mínimo obtenido. También se hace un cómputo global de tiempos y comparación entre los distintos métodos de cálculo.

- Un fichero tipo *".mat"*, en el que se almacena la matriz de resultados del problema lanzado. En ella, se almacena toda la información del proceso de un modo "tangible"para poder realizar gráficas o informes.

También es posible que aparezca otro fichero de texto llamado *Error Data.txt* en el que, en caso de que hubiera ocurrido algún error, éste se almacenaría en el fichero con el fin de evaluarlo más adelante.
Adicionalmente, si se ejecuta una instancia con cálculos en paralelo, se crearán dos ficheros extra:

- Un fichero llamado *Comunication Report.txt* en el que aparece cada una de las instrucciones enviadas y recibidas entre el servidor y los clientes, todas ellas etiquetadas con el momento exacto en el que se han llevado a cabo.

- Otro fichero llamado *Parallel Computing Report.txt* en el que se almacena la tarea o individuo enviado a cada uno de los clientes. También se almacena los resultados obtenidos de una determinada tarea y la información del cliente que la ha evaluado.

6.1.5. Procedimiento. Ejemplo

En este último apartado del manual se ejecutará un problema que tenga en cuenta la mayoría de las funcionalidades. De forma guiada, se explicará cada uno de los pasos a realizar. En concreto, se ejecutará una optimización de una biela con hibridación de algoritmos, el primero de ellos será un algoritmo genético el cual se ha paralelizado su calculo usando cuatro clientes y el segundo un algoritmo basado en gradiente. A lo anterior, se ha añadido el entrenamiento y uso de *RNA*. Los pasos básicos serán los que se explican a continuación:

1. Puesto que se va ejecutar una instancia completa del problema, debemos preparar a los clientes para la comunicación con el servidor. Esto conlleva, arrancarlos, cerciorarnos de que existen los ficheros de los que se ha hablado en el apartado de *Configuración* y, una vez hecho esto, se debe ejecutar en Matlab la función *ParallelComputingTCPIP.m*. Actualmente este paso se deber hacer físicamente, encendiendo manualmente los ordenadores y ejecutando la función o en caso de disponer permisos más elevados, se pueden arrancar los clientes a través de la *Intranet* personal de cada uno y tras esto, conectarse usando una conexión tipo escritorio remoto. En este momento, cada uno de los clientes está preparado para conectarse en el momento en que el servidor lo considere necesario.

2. Dentro de la carpeta de computación en paralelo del programa, se encuentra el *Excel* donde está la información necesaria de cada uno de los clientes. El siguiente paso a realizar será el de seleccionar aquellos clientes que entren en juego en la ejecución. Estos clientes no tienen porque ser necesariamente aquellos anteriormente arrancados, puede considerarse un mayor número de estos y tras el arranque de la instancia se pueden ir encendiendo el resto de clientes que actualmente se encuentre apagados, el algoritmo irá buscando nuevos clientes cada cierto tiempo. De este modo, se conectará a los nuevos clientes cuando estén disponibles. En el ejemplo concreto que se está tratando no hay forma de arranque manual puesto que los clientes son servidores de cálculos así que se conectará por escritorio remoto, se buscará el directorio de trabajo y se ejecutará la función *ParallelComputingTCPIP.m*.

3. Debido a que también se ha considerado el uso de *RNA* en este ejemplo, se explica como modificar los parámetros relacionados con las condiciones de uso de estos. El resto de parámetros que involucran al entrenamiento y validación de las redes no debería ser modificado por el usuario. Los parámetros a modificar son:

    - *FeedNNetResults*, que define la cantidad de análisis previos que deben realizarse antes de disponer de la suficiente información como para entrenar la *RNA*.

- *MaxNNetUses*, puesto que las *RNA* tienen una vida óptima debido a que a medida que la solución evoluciona, la red no es capaz de representar fielmente ese nuevo espacio de soluciones al no disponer de información, se define un parámetros que configura la cantidad máxima de cálculos que se pueden realizar con la red antes de ser eliminada.

Estos parámetros no se han añadido a la interfaz de usuario y deben ser modificados manualmente a través de la edición de la función llamada *Data Initialization.m*.
En ella, también se pueden modificar otros parámetros de interés como la máxima tensión admisible del problema que se vaya a lanzar.

4. El resto de la configuración necesaria para lanzar satisfactoriamente un problema se puede realizar usando la interfaz de usuario mostrada en el apartado *Interfaz de Usuario*.
El orden en el que se definan los parámetros mostrados a continuación no tiene ninguna relevancia, salvo en el caso de los algoritmos, que se deben crear en el orden en el que serán ejecutados.

   a) Se crea la serie de algoritmos a ejecutar y también los parámetros de configuración de cada uno de estos. Primero se describe el algoritmo genético, con sus parámetros de configuración más básicos:

   - Número de individuos por población, igual a 75.
   - Tolerancias, igual a $1x10^{-4}$.
   - Mostrar información por pantalla, al final.
   - Paralelización, activada.
   - Calculador usado, *ANSYS*

   Puesto que se ha añadido un algoritmo que debe continuar tras el ya definido, se debe seleccionar un criterio de parada. Existen varias opciones para elegir, en este caso, se seleccionará la cantidad de poblaciones a calcular, tomando el valor de 5.
   Ahora, se detallarán los parámetros de funcionamiento del algoritmo basado en gradiente:

   - Espacio de búsqueda, en este caso *Local*. Puesto que se ha tomado valor, los dos parámetros siguientes no tienen relevancia y por lo tanto no importa el valor que se les asigne. Estos parámetros hacen referencia al arranque iterativo del algoritmo desde distintos valores iniciales.
   - Algoritmo usado, existen varias opciones, pero para este problema se seleccionará *sqp*, que recordemos hace referencia al *Sequential Quadratic Programming*.
   - Información a mostrar por pantalla, al igual que en el anterior se hará al final del algoritmo.

- Tolerancias, en este caso será de $1x10^{-6}$, puesto que este algoritmo ya decidirá cual es el mínimo alcanzado, queremos una mayor precisión en la solución.
- Paralelización, desactivado por lo motivos detallados en la memoria.
- Calculador usado, al igual que en el anterior algoritmo será *ANSYS*.

Como tras la convergencia de este algoritmo, se dará por terminada la ejecución del problema, no sería necesario seleccionar ningún criterio de parada, mas que el propio por defecto del algoritmo, aunque ésta se podría definir.

b) A continuación, se seleccionará el problema a estudiar. Al *clickar* se abrirá un selector de directorios, se usará el menú de navegación para encontrar el directorio de trabajo que deseamos y posteriormente seleccionarlo.

c) El siguiente paso, será el de definir las restricciones que existen entre las variables de diseño. Para ello, nos valemos del menú superior y abrimos la ventana de modificación de restricciones. En la tabla de izquierda se definen los límites inferior y superior que pueden tomar las variables, en este caso, serán cuatro variables siendo sus límites inferiores igual a $[1, 1, 1{,}2, 2]$ y los superiores tal que $[7, 7, 7, 7]$. En cuanto a las restricciones lineales entre variables, simplemente se ha definido que una variable debe ser mayor o igual que la anterior, esta restricción se puede representar de la siguiente manera: $V_1 \leq V_2 \leq V_3 \leq V_4$.

d) Los siguientes parámetros a definir serán aquellos que estén relacionados con las opciones de ejecución:

- Uso de RNA, activado.
- Recuperación de resultados ya calculados, activado.
- Partir de otro análisis anterior, desactivado.

e) El último valor que debe ser introducido a través de la interfaz de usuario es la sugerencia inicial, es decir, por donde empezará el primero de los algoritmos a ejecutar. En nuestro caso este valor equivale a $[3, 4, 5, 6]$.

5. Cuando estemos seguros que todos los parámetros se han introducido correctamente, solo quedaría *clickar* en el botón *Start* para comenzar el análisis. En ese momento, saltará una ventana emergente de confirmación, dándonos una segunda oportunidad para modificar alguno de los parámetros. Tras esto, es posible que aparezca otra ventana emergente, ésta se muestra porque ha detectado una matriz de resultados que ha concluido su ejecución de una manera inusual, por ejemplo debido a un corte de luz o que el usuario ha interrumpido la ejecución manualmente. Nos preguntará si queremos continuar la ejecución desde donde se quedó la anterior. Seleccionamos la opción que más nos convenga e iniciamos el análisis.

6. Por último, aparecerá la ventana informativa sobre la comunicación en paralelo. Puesto que carece de interactividad con el usuario, simplemente es a título informativo y sirve para seguir visualmente el proceso de paralelización.

Bibliografía

[1] Peter W. Christensen and Anders. Klarbring. *An Introduction to Structural Optimization*, volume 153. Springer, 2009.

[2] James C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., Hoboken, NJ, USA, mar 2003.

[3] Pradnya A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265. IEEE, dec 2016.

[4] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, may 1983.

[5] Amira Hamdi, V Antoine, Nicolas Monmarché, Adel Alimi, and Mohamed Slimane. Artificial Ant: From Collective Intelligence to Real-life Optimization and Beyond. pages 265–290. ISTE, 2010.

[6] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publications, 1976.

[7] Robby Haelterman. Analytical study of the Least Squares Quasi-Newton method for interaction problems. pages XXI, 224, 2009.

[8] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, jan 1988.

[9] Nocedal Jorge and J Wright Stephen. *Numerical Optimization*. Springer, 2006.

[10] K. Svanberg. The Method of Moving Asymptotes (MMA) with Some Extensions. 1998.

[11] J. Scott Armstrong. Illusions in regression analysis. *International Journal of Forecasting*, 28(3):689–694, jul 2012.

[12] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[13] Marcel van Gerven and Sander Bohte. Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Frontiers in Computational Neuroscience*, 11, dec 2017.

[14] Enrique Nadal. Cartesian grid FEM (cgFEM): High performance h-adaptive FE analysis with efficient error control. Application to structural shape optimization. page 312, 2014.

[15] M. Tur, J. Albelda, E. Nadal, and J. J. Ródenas. Imposing Dirichlet boundary conditions in hierarchical Cartesian meshes by means of stabilized Lagrange multipliers. *International Journal for Numerical Methods in Engineering*, 98(6):399–417, may 2014.

[16] George S. Almasi and Allan. Gottlieb. *Highly parallel computing*. Benjamin/Cummings, 1989.

[17] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, jan 1966.

[18] Philip Wolfe. Convergence Conditions for Ascent Methods. II: Some Corrections. *SIAM Review*, 13(2):185–188, 1971.

[19] Sanjay Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, 2(4):575–601, nov 1992.

[20] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, apr 2001.

[21] P. Ladevèze and E. A.W. Maunder. A general method for recovering equilibrating element tractions. *Computer Methods in Applied Mechanics and Engineering*, 137(2):111–151, oct 1996.

[22] Joseph F. Hair, Jr., William C. Black, Barry J. Babin, Rolph E. Anderson. Multivariate Data Analysis. *Faculty Publications*, feb 2010.

[23] J. J. Ródenas, G. Bugeda, J. Albelda, and E. Oñate. On the need for the use of error-controlled finite element analyses in structural shape optimization processes. *International Journal for Numerical Methods in Engineering*, 87(11):1105–1126, sep 2011.