Método del Enjambre de Partículas y Evolución Diferencial Para el Diseño de Peso Mínimo de un Panel Naval Reforzado

Guillermo Eduardo Giraldo Fernández

Facultad de Ingenierías
Programa de Ingeniería Mecánica y Mecatrónica
Universidad Tecnológica de Bolívar
Cartagena de Indias D.T. y C.
2011

Cartagena de Indias, Junio de 2011

Señores

Comité Evaluador de Proyectos
Dirección de Programa Ingeniería Mecánica y Mecatrónica
Universidad Tecnológica de Bolívar
L. C.

De la manera más atenta me permito presentar para su consideración el informe de tesis titulado: “METODO DEL ENJAMBRE DE PARTICULAS Y EVOLUCION DIFERENCIAL PARA EL DISEÑO DE PESO MINIMO DE UN PANEL NAVAL REFORZADO”, como requisito para optar por el título de Ingeniero Mecatrónico.

Atentamente,

_____

Guillermo Eduardo Giraldo Fernández

Autorización

Cartagena de Indias, Junio de 2011

Yo, GUILLERMO EDUARDO GIRALDO FERNANDEZ, identificado con la CEDULA DE CIUDADANIA NUMERO 1.143’330.707 de CARTAGENA, autorizo a la UNIVERSIDAD TECNOLOGICA DE BOLIVAR para hacer uso del trabajo titulado “METODO DEL ENJAMBRE DE PARTICULAS Y EVOLUCION DIFERENCIAL PARA EL DISEÑO DE PESO MINIMO DE UN PANEL NAVAL REFORZADO”, y para publicarlo en el catalogo en línea de la biblioteca.

_____

Guillermo Eduardo Giraldo Fernández
CC 1.143’330.707 de Cartagena de Indias

Nota de Aceptación

_____

_____

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Método del Enjambre de Partículas y Evolución Diferencial Para el Diseño de Peso Mínimo de un Panel Naval Reforzado

Guillermo Eduardo Giraldo Fernández

Tesis

Presentada como requisito para optar por el titulo de:

Ingeniero Mecatrónico

Director:

Jairo Useche Vivero, Ph. D.

Facultad de Ingenierías
Programa de Ingeniería Mecánica y Mecatrónica
Universidad Tecnológica de Bolívar
Cartagena de Indias D.T. y C.
2011

El principio de la sabiduría es
el temor de Jehová
(Proverbios 1:7a)

A mi amada familia y amigos

Contenido

Agradecimientos

El trabajo presentado a continuación no es resultado únicamente del esfuerzo de los 4 meses dedicados a aplicar las ideas y técnicas descritas, sino de los varios años que ha tomado completar mi carrera universitaria. Por ende, los agradecimientos se extienden a todas las personas que hicieron posible este gran logro.

En primer lugar, a mi Dios y sustentador, el cual nos ha regalado este mundo bello, con sus misterios y curiosidades: esta es la primera responsabilidad del hombre para con la naturaleza, no solo de los ingenieros e industriales, sino de cada persona el hacer buen uso de los recursos a nuestra disposición.

En segundo lugar a mis padres, Álvaro y Dali, quienes han gastado hasta su último esfuerzo para que este logro sea materialmente posible, con sus consejos, sacrificios y oraciones; estoy seguro de que mi título es una alegría para ellos aun mas grande que para mí. Mis hermanos, tíos y primos, por su gran apoyo y atenciones durante este tiempo de estudio, los cuales no he nombrado por cuestiones de espacio.

A mis profesores, quienes me han inculcado el amor por la ingeniería, la tecnología y el desarrollo, en especial las responsabilidades de un ingeniero en un país y un mundo como el que nos presenta la actualidad. A Jairo Useche por su guía y consejos durante este y otros proyectos realizados, y por la confianza depositada en mí. A Edgardo Arrieta, de quien escuche mis primeras nociones en temas de optimización de forma y métodos de búsqueda.

En último lugar, a los ingenieros que con sus consejos y conversaciones ayudaron a concebir y dar forma a este proyecto, en especial a los trabajadores de Cotecmar CF Fernando Delgado, CC Juan Carlos Galindo y Diana Ortiz.

Y a todas aquellas personas, amigos y compañeros que han hecho parte de mi vida durante este tiempo: gracias a ustedes soy la persona que soy, mis logros son también sus logros.

Lista de Figuras

Lista de Tablas

Resumen

Se presenta la aplicación de las ideas del diseño estructural óptimo al caso de un panel naval reforzado. En este sentido, se busca encontrar la topología del panel reforzado que produzca el diseño de peso mínimo y que al mismo tiempo satisface las restricciones de seguridad y funcionalidad impuestas. Para tal fin, algoritmos de búsqueda basados en las técnicas del Enjambre de Partículas y la Evolución Diferencial son utilizados, valiéndose de un modelo de elementos finitos para el análisis estructural. Resultados de la búsqueda, discusiones y conclusiones al respecto son presentados.

Introducción

En el marco del proyecto de investigación en Estructuras Navales de GIMAT, se concibe la necesidad de desarrollar e implementar metodologías de diseño estructural modernas y prácticas, de forma tal que la industria local de astilleros pueda tomar provecho de ellas, aumentando la calidad de sus productos, disminuyendo sus tiempos de desarrollo, y en fin último, aumentando su competitividad a nivel local y regional.

Mientras los diseños estructurales modernos se tornan más complejos, esto es, sus elementos constitutivos se alejan cada vez mas de las tradicionales armaduras y elementos simples como vigas y columnas, el diseñador encuentra dificultades para determinar la respuesta estructural del sistema y sus estados de esfuerzos, información clave para asegurar la funcionalidad y seguridad de la estructura. Un claro ejemplo de dicha problemática es el elemento de panel reforzado (ver figura 1.1), conformado por una placa y refuerzos prismáticos, y que encuentra amplia utilización en estructuras navales.

Metodologías tradicionales enfrentan el problema realizando aproximaciones teóricas, compensado las incertidumbres con factores de diseño muy altos. Esta aproximación, aunque ha servido su propósito por cierto tiempo, produce resultados por mucho no óptimos en términos de la cantidad de material utilizado, subiendo los costos de fabricación (en términos de materiales), y en el caso de las estructuras navales, disminuyendo la velocidad y reduciendo la capacidad de carga de la embarcación, lo que se refleja en una disminución de los ingresos de la misma (en caso de embarcaciones comerciales) o en la

disminución de la cantidad de elementos tácticos que pueden ser transportados (en el caso de embarcaciones militares).

Las alternativas para lograr diseños óptimos se conocen desde hace varias décadas, pero técnicas como el análisis por elementos finitos y los esquemas de optimización numérica no resultaban rentables en términos del tiempo y esfuerzo del diseño. Con el reciente crecimiento en el poder de cómputo de los ordenadores y su abaratamiento, dichas alternativas deben ser tenidas en cuenta, evaluadas e implementadas.

Surge entonces la necesidad de demostrar los beneficios de estas técnicas frente a las metodologías tradicionales, en términos de tiempo y esfuerzo de diseño y de resultados óptimos. Para tal fin, se implementará una metodología de diseño moderna, asistida por el ordenador, que incluya una metodología de análisis por elementos finitos y un esquema de optimización numérica, junto al tradicional diseño racional, el cual permite aprovechar la experiencia del diseñador y hacer uso optimo de los recursos computacionales.

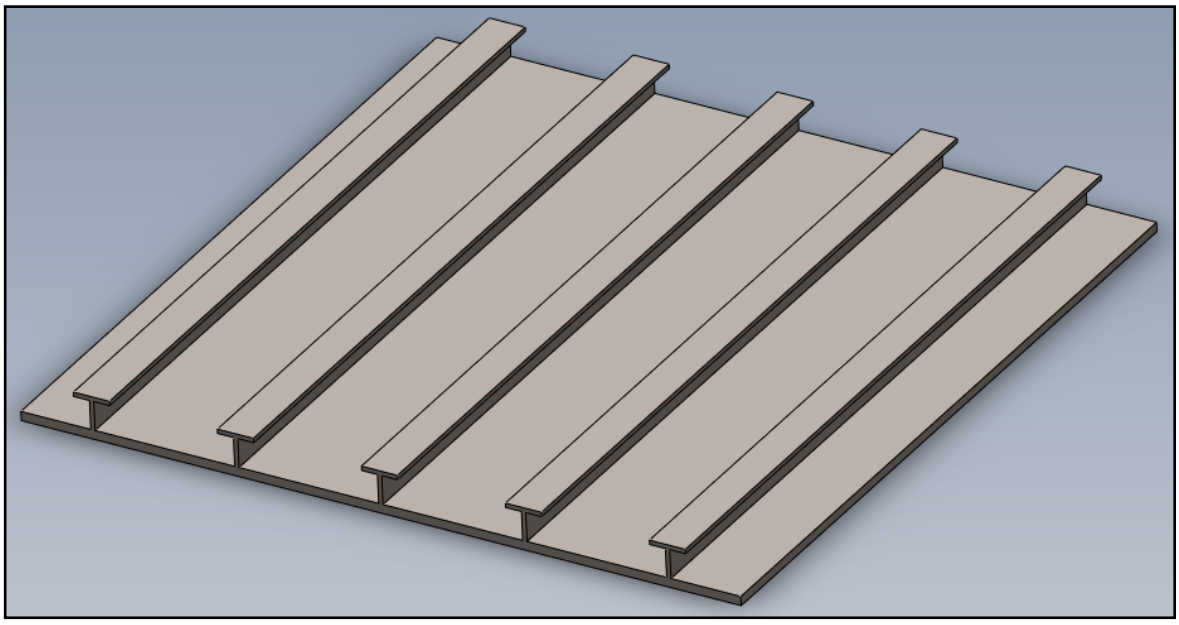

Figura 1.1 Panel Estructural Reforzado con perfiles “T”

Problema de Diseño

El problema de diseño a enfrentar puede enunciarse, en forma de interrogante, como sigue:

“¿Cómo diseñar un elemento estructural, en este caso un panel reforzado, de forma tal que cumpla las necesidades de seguridad e integridad estructural, de acuerdo a sus condiciones de operación estándar como parte de un sistema estructural naval, y al mismo tiempo utilice la menor cantidad de material posible?”

Objetivos

Con el fin de responder al problema de diseño planteado anteriormente, el siguiente objetivo general debe ser alcanzado:

“Encontrar la topología optima de un panel reforzado de aplicación naval, dadas las consideraciones de seguridad y funcionalidad estructural establecidas por las normas internacionales, utilizando una metodología racional, practica y asistida por el ordenador.”

Para alcanzar este fin, se proponen los siguientes objetivos específicos, los cuales determinan cada una de las fases del proyecto a realizar:

1. Definir los requerimientos y parámetros de diseño. Para tal fin se debe implementar una metodología de análisis racional que permita encontrar el estado de cargas sobre el elemento de panel reforzado y los estados límites de las mismas. Además, se debe parametrizar la topología del panel y definir las restricciones sobre tal.

2. Implementar un esquema de Análisis por Elementos Finitos que permita estudiar de forma automatizada el comportamiento del elemento estructural ante el estado de cargas planteadas y determinar si cumple o no los requerimientos del diseño.

3. Implementar un esquema de Optimización Numérica, el cual basado en el criterio de la Mínima Cantidad de Material y en el esquema de análisis del objetivo anterior, permita encontrar la topología optima del elemento estructural.

1. Revisión de la Literatura

En la revisión de la literatura realizada se ha buscado cumplir un objetivo principal, a saber, encontrar las técnicas necesarias para resolver el problema de diseño afrontado, y que las técnicas a ser seleccionadas se hayan probado a sí mismas como competitivas, en el sentido de ser altamente prácticas y eficaces. Dichas características han de ser manifiestas con uno de dos factores: que la técnica utilizada se encuentre arraigada entre las prácticas de diseño de ingeniería utilizadas en la actualidad, esto es, que pueda ser considerada como tradicional; o que la técnica utilizada sea innovadora, y que presente obvias o probadas ventajas frente a técnicas con el mismo propósito utilizadas tradicionalmente. Mientras la primera característica es fácilmente corroborada al recurrir a libros de texto referenciales o a normas clasificadoras, para satisfacer la segunda debe recurrirse a publicaciones de investigaciones recientes, en revistas especializadas o en el vasto recurso que presenta el internet.

En primera instancia, el uso de herramientas computacionales para el análisis y diseño estructural, y en general cualquier diseño de ingeniería, y la búsqueda de la optimización es presentada y sustentada por Hughes. En su libro, se muestra el desarrollo del análisis por elementos finitos y sus consecuencias prácticas para el diseño estructural naval y la gran ventaja que representa el análisis automatizado con miras a la optimización. Los algoritmos de búsqueda presentados reflejan las tendencias de la época, y no tienen en cuenta metaheurísticas.

Con respecto al análisis de paneles reforzados, el texto de Mansour y Liu presenta las aproximaciones analíticas utilizadas con miras al diseño. El método de los elementos finitos es planteado como alternativa, pero no es desarrollado profundamente. Otras aplicaciones y descripciones de

aproximaciones teóricas son presentadas en los trabajos de Brito et al, Campanile et al y Vilnay. En el texto de Hughes también se encuentra el desarrollo de algunas aproximaciones analíticas al análisis de dichos elementos, pero se destaca el desarrollo de un súper-elemento para el análisis computacional. Un caso de éxito en el análisis por elementos finitos para paneles reforzados es presentado en la tesis de maestría de Dongqi.

El uso de metaheurísticas para la optimización de elementos estructurales es evidenciado en la revisión de fuentes en línea y de la revista de optimización estructural y multidisciplinaria de Springer (Structural and Multidisciplinary Optimization). En especial, el trabajo de Vu representa un claro ejemplo del uso de metaheurística para la optimización de peso de un elemento estructural con geometría paramétrica y análisis por elementos finitos. La revisión también muestra un gran uso de otros tipos de optimización de peso, como es la remoción de material. En dicha técnica también se presentan casos del uso de metaheurística como método de evolución de forma (Huang y Xie, Wu y Tseng). Otras interesantes aplicaciones de la metaheurística al diseño estructural encontradas muestran el auge de la técnica entre los diferentes grupos de investigación y la variedad de problemas que pueden manejar.

La optimización de peso de paneles navales reforzados es presentada por Oliveira y Christopoulos, basados en análisis de grillas con deformaciones plásticas, mostrando el interés en la materia. Los elementos para lidiar con un problema similar, utilizando técnicas de optimización actuales (metaheurística) fueron tomados dependiendo de su naturaleza. En especial, Huang y Xie presentan una revisión de las técnicas de optimización topológica de tipo evolutivo.

Una introducción muy comprensiva al tema de la metaheurística es presentada por Luke, junto a modelos de algoritmos para muchos métodos de búsqueda. El método del enjambre de partículas es presentado por Kennedy y Eberhart en un par de trabajos, y varias variantes fueron encontradas (Hart y Vlahopoulos, Li et al, LIU, Moraldo et al). La evolución diferencial y sus posibles variaciones son estudiadas a fondo en el texto de Price, Storn y Lampinen. La técnica de la

función de penalidad es mostrada, junto con resultados de su desempeño en casos de prueba, en el trabajo de Parsopoulos y Vrahatis.

Otras fuentes consultadas, y que de alguna u otra forma influenciaron el presente trabajo también son incluidas en la bibliografía.

2. Marco Teórico

2.1 Diseño Estructural de Ingeniería

Como es planteado por Hughes, el diseño racional de estructuras navales denota la concurrencia de varias técnicas de análisis y diseño de ingeniería, principalmente asistidas por el ordenador, utilizadas en el desarrollo de estructuras navales, y cuyo objetivo principal es la producción de un sistema estructural "optimo". En el presente estudio se utiliza la misma línea de pensamiento y acción para extender las técnicas al diseño estructural en general.

En palabras del propio autor, se define como:

> *"Diseño que es directa y enteramente basado en teoría de estructuras y métodos computacionales de análisis estructural y optimización, y el cual arroja una estructura optima en base a una medida de merito seleccionada por el diseñador."*

La filosofía de diseño presentada por el autor ha visto gran aceptación en la comunidad de ingeniería naval, y representa la base del popular software de diseño asistido por ordenador orientado a estructuras navales MAESTRO.

Para fines de generalización, se presenta la definición del autor del diseño en ingeniería:

> *"La formulación de un modelo preciso del sistema, en orden de analizar su respuesta (interna y externa) a su entorno, y el uso de un método de optimización para determinar las características del sistema que mejor alcancen un objetivo especifico, cumpliendo también ciertas restricciones prescritas en las características y respuesta del sistema."*

De la definición anteriormente presentada, cabe resaltar los siguientes aspectos, que serán de fundamental utilidad en sentar las bases de la filosofía de diseño que será implementada:

1. La importancia de poder determinar de forma **precisa** la respuesta del sistema.
2. La presencia de una medida de merito como orientación del diseño, y el uso de un método de optimización para alcanzarla.
3. El protagonismo de los métodos computacionales de análisis.

El resultado de una metodología capaz de implementar efectivamente estos fundamentos es un diseño confiable, seguro y adaptado a las necesidades de optimalidad.

2.2 Método de los Elementos Finitos

El Método de los Elementos Finitos es un esquema de solución numérica para ecuaciones diferenciales. Desarrollado a mediados del siglo XX por la industria aeroespacial para suplir sus necesidades de análisis estructural, ha conseguido gran éxito en la comunidad de ingeniería por su versatilidad y capacidad de manejar geometrías complejas de manera sencilla.

Históricamente, el método fue desarrollado a partir del Método Matricial de Análisis Estructural, pero formalmente puede ser categorizado entre los esquemas de aproximación de Galerkin. Ambos acercamientos pueden ser utilizadas para desarrollar formalmente el método: la primera es utilizada generalmente cuando se requiere énfasis en la implementación computacional y sencillez en el desarrollo matemático, mientras que la segunda permite mostrar analíticamente la suficiencia del método para aproximar la respuesta y los requerimientos de convergencia. La primera será utilizada en el presente documento por razones de sencillez.

En análisis estructural, el método de los elementos finitos busca aproximar la solución al comportamiento del sistema, esto es, dado un dominio geométrico y un conjunto de cargas y restricciones, encontrar la deformación que sufren los elementos estructurales y los estados de esfuerzos. Para tal fin, se plantea una aproximación numérica en el sentido de Galerkin:

$$u \approx u^* = N_i^e u_i^e$$

Donde se realiza una suma sobre las i-es. La respuesta de deformación "real" u es aproximada por la respuesta u^*, expresada como una combinación lineal del conjunto de soluciones u_i^e. Cada una de las soluciones u_i^e es válida en un dominio reducido, llamado *elemento* (hecho enfatizado con el uso del superíndice e). Los coeficientes N_i^e determinan la influencia y la forma en que la solución en cada elemento aporta a la solución global, y son llamados funciones de forma por esto.

La solución dentro de un elemento u^e (el subíndice es omitido para evitar atiborramiento) es una lista de valores de deformación (conocidos como *grados de libertad*) en ciertos puntos específicos, llamados *nodos del elemento*. De esta forma, la respuesta global aproximada u^* es también una lista de valores de deformación, esta vez sobre el conjunto de *todos* los nodos (y no únicamente los pertenecientes a un elemento). Nótese que al expresar las soluciones en forma de vectores, las funciones de forma N^e deben ser coeficientes matriciales.

De esta forma, se crea el esquema general del método de los elementos finitos: un medio continuo es aproximado geométricamente utilizando elementos geométricos básicos (*elementos*) y puntos (*nodos*), y la solución global de deformación es aproximada como una lista de deformaciones en dichos puntos. La idea es ilustrada en la figura 2.1.

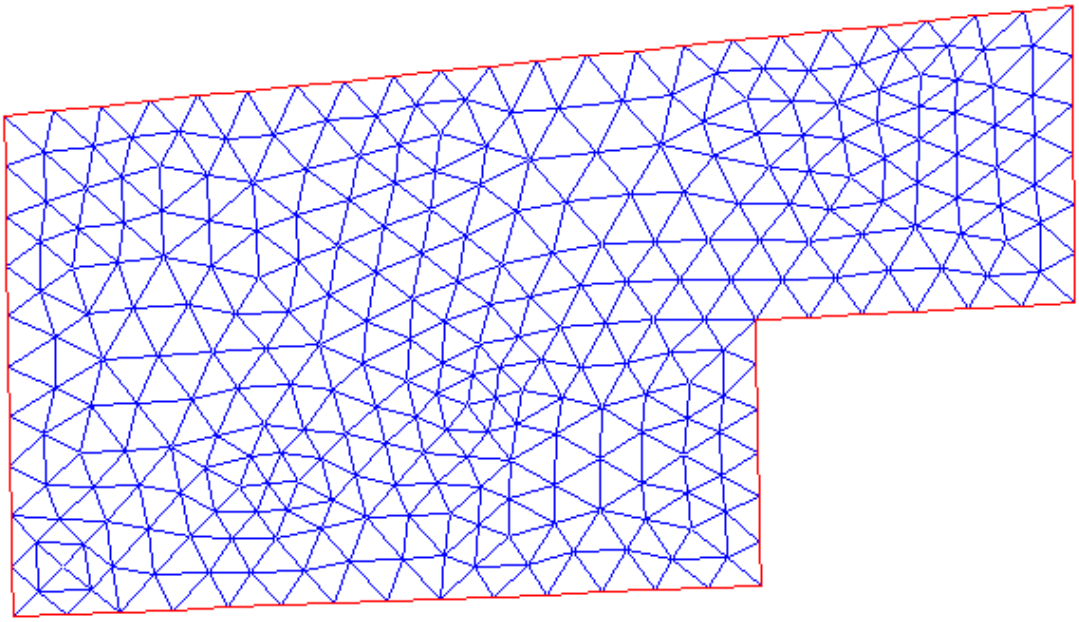

Figura 2.1 Discretización de la geometría para el análisis por elementos finitos.

2.2.1 Mínima Energía Potencial de Deformación

De la mecánica de sólidos, sabemos que un cuerpo se deforma ante una condición de cargas de tal manera que la energía potencial total de la deformación $\Pi(u)$es un punto estacionario:

$$\delta\Pi(u) = 0$$

La energía potencial total de deformación puede ser expresada como la diferencia entre la energía de deformación (energía potencial elástica) y el trabajo realizado por las cargas externas que causan la deformación:

$$\Pi = U - W$$

En términos de los esfuerzos, deformaciones y cargas, las energías pueden calcularse como se muestra:

$$U = \frac{1}{2}\int \sigma^T e d\Omega$$

$$W = \int u^T b d\Omega + \int u^T q d\Gamma$$

En donde Ω denota el dominio geométrico del problema y Γ su contorno; σ y e son el esfuerzo interno y la deformación unitaria representados en forma de vectores; u es la respuesta de deformación representada como vector, b son las fuerzas de cuerpo (actúan en el dominio) y q son las cargas externas (actúan en la frontera).

2.2.2 Ecuación de Rigidez

Con el fin de encontrar la solución al sistema utilizando el principio de la energía mínima potencial y la aproximación por el método de los elementos finitos, se utiliza el siguiente procedimiento general. Por definición:

$$e = Du$$

$$\sigma = Ee$$

En donde D es el operador de gradiente simétrico y E la matriz de esfuerzo-deformación. Enfocando el análisis a un elemento, y utilizando la aproximación del método de los elementos finitos:

$$u = Nu^e$$

$$\Rightarrow e = DNu^e = Bu^e$$

Realizando las respectivas sustituciones, encontramos las expresiones para la energía de deformación del elemento:

$$\Pi^e = U^e - W^e$$

$$U^e = \frac{1}{2} u^{eT} \left[\int_{\Omega^e} B^T E B d\Omega^e \right] u^e = \frac{1}{2} u^{eT} K^e u^e$$

$$W^e = \int_{\Omega^e} u^{e^T} N^T b d\Omega^e + \int_{\Gamma^e} u^{e^T} q d\Gamma^e = u^{e^T} \left[\int_{\Omega^e} N^T b d\Omega^{\mathrm{e}} + \int_{\Gamma^e} N^T q d\Gamma^e \right] = u^{e^T} f^e$$

$$\Pi^e = \frac{1}{2} u^{e^T} K^e u^e - u^{e^T} f^e$$

La solución a la minimización del funcional cuadrático para la energía potencial de deformación es la misma solución a la ecuación vectorial (demostrado al final de la sección):

$$K^e u^e = f^e$$

Que puede ser exitosamente interpretada como una ecuación de rigidez en el dominio del elemento, esto es, una relación lineal entre las cargas aplicadas y las deformaciones producidas. Posteriormente las matrices individuales de rigidez son combinadas en una matriz global de rigidez, aplicando condiciones de continuidad en los grados de libertad comunes entre distintos elementos y de compatibilidad de cargas en los contactos entre elementos.

Un “tipo de elemento” es el resultado de la aplicación del procedimiento mostrado con un conjunto en particular de grados de libertad, funciones de forma y relaciones de esfuerzo-deformación.

2.2.3 Minimización del Funcional Cuadrático

Se desea probar que la solución a la ecuación vectorial:

$$Ax = b$$

También minimiza al funcional cuadrático:

$$f(x) = \frac{1}{2} x^T A x - x^T b + c$$

Para tal fin, considérese una variación ϵ a partir del supuesto punto mínimo x, y calcúlese $f(x + \epsilon)$:

$$f(x+\epsilon) = \frac{1}{2}(x+\epsilon)^T A(x+\epsilon) - (x+\epsilon)^T b + c$$

$$= \frac{1}{2}x^T Ax - x^T b + c + \frac{1}{2}\epsilon^T A\epsilon + \frac{1}{2}x^T A\epsilon + \frac{1}{2}\epsilon^T Ax - \epsilon^T b$$

Si $A^T = A$, y suponiendo $Ax = b \Rightarrow b^T = x^T A^T = x^T A$

$$= \frac{1}{2}x^T Ax - x^T b + c + \frac{1}{2}\epsilon^T A\epsilon + \frac{1}{2}b^T \epsilon + \frac{1}{2}\epsilon^T b - \epsilon^T b$$

$$= \frac{1}{2}x^T Ax - x^T b + c + \frac{1}{2}\epsilon^T A\epsilon$$

Reconociendo el funcional original en los primeros términos, se escribe:

$$f(x+\epsilon) = f(x) + \frac{1}{2}\epsilon^T A\epsilon$$

Donde se advierte claramente que x es efectivamente un punto estacionario del funcional. El comportamiento del funcional en las cercanías de x se encuentra determinado por la naturaleza de la matriz A: si $A \preccurlyeq 0$ (semidefinida negativa), el termino cuadrático en ϵ es negativo y x es el máximo del funcional; si, por el contrario, $A \succcurlyeq 0$ (semidefinida positiva) el termino cuadrático es positivo y x resulta ser el mínimo del funcional.

En particular, la matriz de rigidez obtenida en el meto de los elementos finitos a partir de la energía potencial de deformación es semidefinida positiva, ya que la energía potencial $\frac{1}{2}\int \sigma^T e d\Omega = \frac{1}{2}u^T Ku$ siempre será un numero positivo o al menos cero para $u = 0$.

2.3 Análisis por el Método de los Elementos Finitos

La aplicación del método de los elementos finitos al análisis computacional de estructuras sigue un flujo determinado, presentado a continuación:

1. ***Pre-procesamiento***: en la etapa inicial del análisis es necesario modelar computacionalmente la geometría del problema; luego, una discretización adecuada con miras a los tipos de elementos a utilizar, es realizada. Este procedimiento es comúnmente conocido como “mallado”.
2. ***Solución***: con la malla como insumo de entrada, se procede a aplicar las condiciones de contorno, propiedades materiales, tipos de elementos y de análisis a realizar. Se finaliza con la solución del sistema de ecuaciones para los grados de libertad $Ku = f$.
3. ***Post-procesamiento***: a partir de la solución para los grados de libertad, se construyen los campos de deformaciones y esfuerzos requeridos por el análisis. Los resultados pueden ser procesados como texto o pueden ser generados gráficos para la visualización de las soluciones.

El verdadero interés sobre el método de los elementos finitos es que el flujo de análisis puede ser automatizado. Una pieza de software puede tomar parámetros de la geometría, realizar el modelo, mallar, aplicar parámetros de análisis, solucionar y extraer la solución de forma autónoma. En este sentido, el análisis se convierte en una *función de caja negra*, en la que ante ciertos parámetros de entrada se produce la información necesaria para el desarrollo del diseño.

2.4 Diseño Estructural de Peso Mínimo

Para la selección de la medida de merito de un diseño, el diseñador se encuentra con un abanico de posibilidades. La experiencia del diseñador conjugada con las necesidades y realidades del sistema en estudio determinan el objetivo a perseguir mediante el proceso de optimización. Esto es válido para cualquier diseño de ingeniería. Sin embargo, la búsqueda de un diseño óptimo debe ser un compromiso asumido por el diseñador, brindando un valor agregado a la funcionalidad y seguridad del diseño.

Varias son las medidas de merito comúnmente utilizadas en el diseño estructural de ingeniería, destacándose entre ellas el peso mínimo y la máxima rigidez. Otras medidas de merito pueden ser utilizadas (p. ej. costos de fabricación y de mantenimiento, forma aerodinámica o hidrodinámica, confiabilidad, seguridad), e incluso combinaciones entre varias de estas (técnica conocida como optimización multi-objetivo).

La selección del peso mínimo como medida de merito para la optimización estructural es justificada por el impacto que tiene sobre varios aspectos de la construcción y vida útil de la estructura. En primer lugar, la utilización de la mínima cantidad de material asegura una disminución en los costos de fabricación, al menos en términos de materia prima. En segundo lugar, una estructura más liviana reduce ciertos costos de mantenimiento, como la remoción de corrosión y uso de películas protectoras. Por otro lado, para el diseño estructural de una embarcación es de gran importancia el mantener el peso en vacio del buque al mínimo, ya que dicha característica se refleja en menor necesidad de potencia motriz, menor consumo de combustible, mayor velocidad y mayor capacidad de carga. Para una embarcación militar, se aumenta la capacidad de carga de elementos tácticos y mayor velocidad y maniobrabilidad.

La optimización de peso puede ser realizada de varias formas, como remoción de material. Una alternativa más práctica es asumida en el presente trabajo, y es, teniendo un elemento conceptual, cuya geometría se encuentre parametrizada, encontrar la configuración que minimice la cantidad de material, respetando restricciones de funcionalidad y seguridad.

2.5 Optimización Numérica

Con el fin de encontrar la configuración óptima, es necesario realizar una búsqueda sobre los parámetros de diseño. Tal búsqueda debe ser automatizada, implementada como una rutina computacional reutilizable.

Muchos tipos de algoritmos de búsqueda han sido desarrollados con el tiempo, hecho que ha resultado en una gran variedad de opciones para el diseñador al momento de seleccionar el que mejor se adapte al tipo de problema que enfrenta. Un conocimiento de las debilidades y fortalezas de cada algoritmo es necesario para realizar la mejor selección, mejorando las posibilidades de éxito de la búsqueda.

En términos generales, el problema de optimización puede ser planteado como sigue:

Dado un espacio de búsqueda: $x \in R^n$

Minimizar la función objetivo: $f(x) \in R$

Sujeto a las restricciones: $g_i(x) \leq 0$

El hecho de minimizar en vez de maximizar la función objetivo resulta irrelevante dado que esta sea un número real, esto es, porque minimizar $f(x)$ es equivalente a maximizar $-f(x)$. La representación de las restricciones en la forma planteada no resulta restrictiva, ya que una restricción de la forma $g_i \geq 0$ puede ser representada como $-g_i \leq 0$, y una restricción de la forma $g_i = 0$ puede ser representada como un par de restricciones $g_i \leq 0$ y $-g_i \leq 0$.

2.5.1 Optimización Restringida

Como se ha mostrado, en una optimización relacionada con el diseño de ingeniería, cierto conjunto de restricciones deben ser respetadas. La naturaleza de las restricciones puede estar relacionada con el espacio de búsqueda (por ejemplo, $x_L < x < x_U$; $x_1^2 - 2x_2^2 = 0$), resultando en un acotamiento del mismo, o pueden no estar relacionados directamente con el espacio de búsqueda (por ejemplo, $\sigma_M^{'} < S_y$). En ambos casos, la dependencia de las funciones de restricción en los parámetros de diseño se encuentra presente, explicita o no explícitamente.

La forma en que se incluyan las funciones de restricción en el método de búsqueda parece ser crucial tanto para el desempeño como para el éxito del procedimiento. Las funciones de restricción del tipo 1 (acotamiento del espacio de búsqueda) pueden ser incluidas modificando el método de búsqueda de forma tal que rechace los candidatos de soluciones que no se encuentran en el espacio acotado (que no cumplen las restricciones); pero las restricciones del tipo 2, que no poseen una forma analítica, deben ser tratadas con más cuidado.

2.5.2 Método de la Función de Penalidad

La función de penalidad es una forma elegante de incluir las funciones de restricción (de ambos tipos) en el método de búsqueda, cambiando la función objetivo original por una nueva función a ser optimizada:

$$F(x) = f(x) + h(k)H(x)$$

Donde $f(x)$ es la función objetivo original, y $H(x)$ es un término de penalidad que depende de las funciones de restricción. El término $h(k)$ es un factor que cambia la importancia relativa de la penalización a medida que las iteraciones k de la búsqueda avanzan. Cuando es utilizado, el método es llamado *no estacionario.*

De esta forma, una optimización restringida es enmascarada como una optimización sin restricciones:

Dado un espacio de búsqueda: $x \in R^n$

Minimizar: $F(x) \in R$

El término de penalidad se calcula como sigue:

$$H(x) = \sum_i \theta(q_i(x)) q_i(x)^{\gamma(q_i(x))}$$

$$q_i(x) = \max\{g_i(x), 0\}$$

El factor q_i es la violación relativa de la restricción i: para una restricción no violada $g_i \leq 0 \Rightarrow q_i = 0$ y no hay aporte a la penalización. Si en cambio $g_i > 0$, la restricción ha sido violada y $q_i = g_i$ es una medida de *cuanto* ha sido violada la restricción. El factor θ y la potencia γ cumplen la tarea de escalar la penalidad, de forma tal que resulte alta para violaciones altas de la restricción, y baja para violaciones bajas. De esta forma, el método de búsqueda se aleja rápidamente de las regiones en que la penalización es alta, pero explora detalladamente las zonas en que las penalizaciones son bajas, ya que el óptimo puede encontrarse cerca de un *límite*.

Nótese que la forma en que es calculada la violación relativa de las restricciones induce una no linealidad (un salto) en la nueva función objetivo. Por otro lado, generalmente las funciones θ y γ son seleccionadas como funciones a trozos no continuas, sumando a la cantidad de no linealidades que acarrea la utilización del presente método. Este hecho causa que el método de búsqueda a ser utilizado para minimizar la función de penalidad deba ser seleccionado cuidadosamente. En especial, el método debe ser completamente insensible ante las discontinuidades en la función objetivo, esto es, no requerir información alguna de las derivadas.

2.5.3 Métodos Basados en Derivadas

Los esquemas de optimización clásicos, como el Descenso Empinado, se valen de las derivadas de la función objetivo para explorar el espacio de búsqueda. Empezando en una posición arbitraria en el espacio de búsqueda, se actualiza la posición iterativamente siguiendo la dirección de mayor decrecimiento hasta que se alcanza el mínimo deseado o se termina el tiempo de cómputo:

$$x^{k+1} = x^k - \alpha \nabla f(x)$$

El factor α controla la búsqueda en el siguiente sentido: si el incremento dictado por el gradiente es muy grande (regiones muy empinadas) la búsqueda puede

saltar el optimo; si por el contrario el gradiente es muy pequeño (regiones planas), la búsqueda puede tardar mucho tiempo en avanzar, e incluso quedar “atrapada”. En este sentido, variar α permite ajustar la influencia de la magnitud del gradiente, teóricamente logrando evitar estos comportamientos indeseados.

Varios métodos para variar la influencia de la magnitud del gradiente han sido desarrollados, y esquemas como los Gradientes Conjugados y las Búsquedas de Línea utilizan la información del gradiente e incluso del Hessiano (matriz de segundas derivadas) para hallar el α optimo.

Resulta evidente la fuerte dependencia de este tipo de métodos en la naturaleza de la función. Las primeras (e incluso segundas) derivadas de la función deben existir y ser continuas para asegurar el buen funcionamiento (pero no el éxito) de la búsqueda. Con el fin de mantener la practicidad del método, resulta ideal poseer una forma analítica de la función objetivo en los parámetros, de las funciones de restricción y de sus derivadas (en algunos casos hasta la segunda); sin embargo, algunos métodos se valen de aproximaciones numéricas de las derivadas (diferencias finitas) cuando esta información no se encuentra disponible, induciendo imprecisiones.

En términos generales, no se encuentra garantizada la existencia y continuidad de las derivadas de la función objetivo y de las restricciones en un problema de optimización estructural. Incluso, no se garantiza que exista una forma analítica para el cálculo del merito o de algún parámetro de restricción (por ejemplo, el estado de esfuerzos y deformaciones), siendo necesaria la utilización de métodos de análisis (como Elementos Finitos o Dinámica de Fluidos Computacional) para determinarlos.

2.5.4 Metaheurística

Como plantea Luke, *metaheurística* es un término más bien fuera de lugar, comúnmente utilizado para denotar un campo de estudio más amplio, más precisamente conocido como *optimización estocástica*. Una técnica de esta

familia busca optimizar la función objetivo utilizando cierto grado de aleatoriedad, y por ende de indeterminismo. Dicho indeterminismo ha dificultado la labor de establecer formalmente las características de los métodos de búsqueda.

La principal motivación para la utilización de este tipo de técnicas de optimización, es la no dependencia en la continuidad y derivabilidad de la función objetivo, razón por la cual poseen gran versatilidad entre la cantidad de problemas que pueden afrontar. Sin embargo, con la versatilidad viene la incertidumbre, y en términos generales no está garantizado el éxito de la búsqueda, ni que el optimo encontrado sea el optimo global.

2.5.5 Método del Enjambre de Partículas

El método del enjambre de partículas (originalmente llamado *Particle Swarm Optimization*), pertenece a la familia de los métodos estocásticos de optimización basados en población. Desarrollado por un psicólogo social (James Kennedy) y un ingeniero eléctrico (Russell Eberhart), originalmente fue concebido para simular el comportamiento de una bandada de pájaros o un cardumen de peces mientras buscaban por comida.

La idea detrás del método es que los miembros del enjambre comparten la información de la mejor posición encontrada (en el sentido del mayor merito, en este caso menor distancia hasta el alimento), y que cada miembro tiene en cuenta esta información, junto con el conocimiento de su propia mejor posición, para dirigir su búsqueda. La transición entre la simulación original y el método de búsqueda fue evidente porque la distancia entre cada individuo y la posición del alimento era modelada como una función, que debía ser minimizada al “encontrar” el alimento. Pruebas fueron realizadas y se encontró que *cualquier* función podría ser minimizada utilizando la simulación.

Búsqueda

Como el método se relaciona con la posición espacial de cada individuo y su distancia con el óptimo, el espacio de búsqueda adecuado para el método es, naturalmente, un conjunto de números reales, y la función objetivo un escalar. La posición de un individuo es un vector x_i perteneciente a este espacio, y se actualiza iterativamente utilizando un segundo vector v_i, de *velocidad*, que dirige la partícula hacia la mejor posición conocida hasta el momento por el enjambre x^g y por si misma x_i^p:

$$v_{ij}^{k+1} = \chi\left(\omega v_{ij}^k + c_1 r_1\left(x_j^g - x_{ij}^k\right) + c_2 r_2\left(x_{ij}^p - x_{ij}^k\right)\right)$$

$$x_{ij}^{k+1} = x_{ij}^k + v_{ij}^{k+1}$$

Donde $i = \{1, \dots, Np\}$ indexa cada partícula, $j = \{1, \dots, n\}$ indexa cada dimensión de búsqueda y k corre con las iteraciones del método. La regla es aplicada iterativamente a cada parámetro en cada partícula hasta que es alcanzado el óptimo deseado o se termina el tiempo.

Los parámetros del método: χ, ω, c_1 y c_2, son conocidos como constricción, inercia, cognición y confianza respectivamente. La cognición determina la influencia de la mejor posición global sobre la velocidad, y la confianza determina la influencia de la mejor posición de la partícula; la inercia determina la tendencia de la velocidad a seguir su trayectoria anterior; y la constricción controla que la magnitud de la velocidad se mantenga limitada, evitando que el enjambre explosione. Tradicionalmente:

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}$$

$$\phi = c_1 + c_2 > 4$$

Para terminar, r_1 y r_2 son números aleatorios seleccionados en el intervalo $[0,1]$ cada vez que la regla de actualización es utilizada. Su efecto es variar la

influencia relativa de la cognición y la confianza, añadiendo de esta forma el sabor estocástico a la regla de evolución.

2.5.6 Método de Evolución Diferencial

El método de evolución diferencial (*Differential Evolution* originalmente), al igual que el enjambre de partículas, pertenece a las metaheurísticas basadas en población. En este marco, se ajusta mucho a la plantilla de las Estrategias Evolutivas de optimización, destacándose por su aplicación a los espacios con métricas definidas (las operaciones de suma y multiplicación por un escalar deben existir), como por ejemplo conjuntos de números reales o enteros, o combinaciones de estos.

En las Estrategias Evolutivas, una población de soluciones candidatas *evoluciona* iterativamente hasta que se alcanza el optimo deseado o hasta que se termina el tiempo. Dicha *evolución* es llevada a cabo en tres etapas por cada *generación*, a saber: mutación, cruce y selección. La mutación crea una población alterna de candidatos a partir de la información de la población actual. En el cruce, ambas poblaciones son combinadas para crear una tercera población, llamada de *hijos*. En la selección, se discrimina entre los *padres* (la población original) y los *hijos* para formar la población que pase a la siguiente generación. La evolución diferencial implementa estas estrategias en la siguiente forma:

Mutación

Para cada miembro en la población se crea un mutante, a partir de la combinación de las posiciones de varios individuos seleccionados aleatoriamente. En el caso más sencillo, se seleccionan tres individuos de la población, todos distintos entre sí y distintos al miembro a ser mutado x_i , y se crea el mutante u_i según la regla:

$$u_i = x_1 + Fr(x_2 - x_3)$$

Donde Fr es el factor de escalamiento de la diferencia, un parámetro general del método. La magnitud del factor afecta el comportamiento general del método, logrando que para valores pequeños la búsqueda sea mas explotativa (buscando detalladamente) o mas explorativa (buscando regiones más amplias) para valores más grandes.

Cruce

Cada padre x_i y su respectivo mutante u_i se combinan parámetro a parámetro, según una distribución de probabilidad uniforme, produciendo el hijo v_i:

$$v_{ij} = \begin{cases} u_{ij} \ si \ r \leq Cr \ o \ j = j_{rand} \\ x_{ij} \ en \ otro \ caso \end{cases}$$

El subíndice j corre con los parámetros de búsqueda (miembro a miembro de los vectores). El parámetro Cr es la probabilidad de cruce, dictando la cantidad (estocástica) de parámetros que son heredados al hijo por parte del mutante ($1 - Cr$ es la cantidad relativa de parámetros que son heredados del padre); para tal fin, r debe ser un número aleatorio seleccionado uniformemente en el intervalo $[0,1]$ para cada parámetro a ser determinado. El índice j_{rand} es seleccionado aleatoriamente entre el número de parámetros, y asegura que el hijo herede al menos un parámetro del mutante.

Selección

Para terminar, el lugar en la siguiente generación es tomado por el hijo o el padre, cualquiera de ambos que posea el mejor merito:

$$x_i^{k+1} = \begin{cases} v_i^k \ si \ f\left(v_i^k\right) \leq f(x_i^k) \\ x_i^k \ en \ otro \ caso \end{cases}$$

3. Implementación

3.1 Caso de Estudio

El caso de estudio seleccionado para la optimización consiste en un panel con reforzadores equiespaciados en una sola dirección, de perfil en forma de “L”. La topología y los respectivos parámetros son mostrados en la figura 3.1.

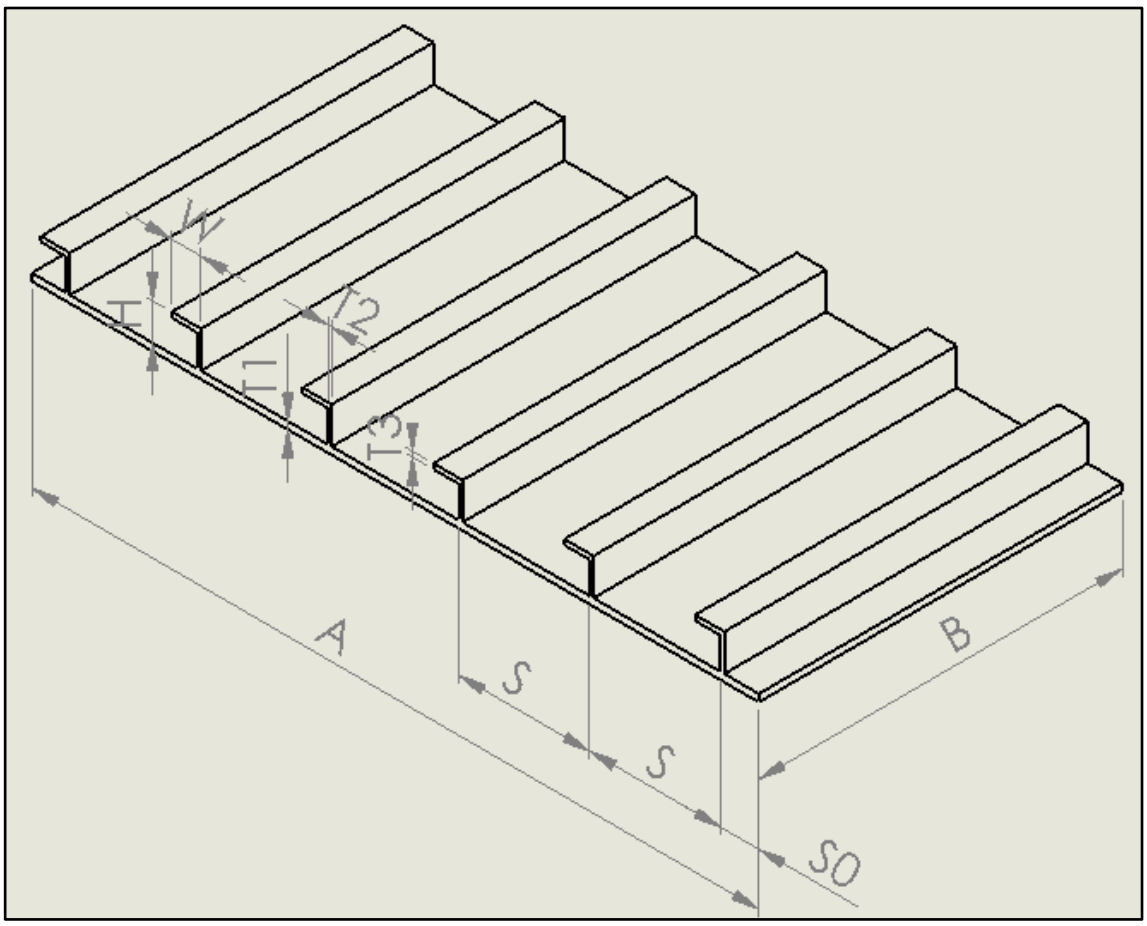

Figura 3.1 Topología del caso de estudio.

3.2 Modelo por el Método de los Elementos Finitos

La solución por el método de los elementos finitos es realizada utilizando Code Aster, un programa de análisis desarrollado por el Ministerio de Energía de Francia, y liberado bajo licencia pública general GPL. El modelado paramétrico de la geometría y el mallado es realizado programáticamente, utilizando las librerías de Salome Meca, software liberado también bajo licencia pública. Como puede verse en la figura 3.2, se utilizaron elementos hexagonales para la discretización de la geometría. Específicamente, el tipo de elementos utilizados corresponde al modelado 3D con elementos HEXA8, de interpolación lineal.

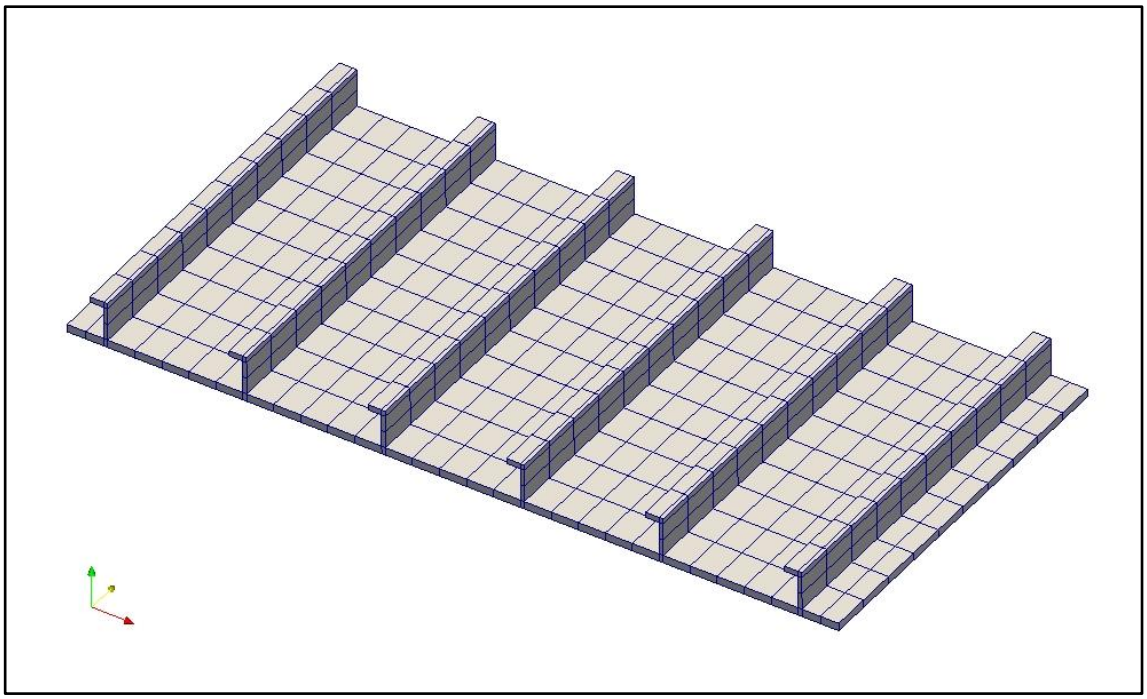

Figura 3.2 Mallado del caso de estudio.

Las condiciones de contorno utilizadas son:

- $DX = DY = DZ = 0$ para todos los nodos en el plano $y = 0$,
- $DX = DY = DZ = 0$ para todos los nodos en el filo inferior del plano $y = B$, esto es, en la recta paralela al eje x que pasa por el punto $(0, B, 0)$,

- Presión $P = 39.2x10^{-6}[MPa]$ distribuida uniformemente sobre la cara inferior, esto es, la coincidente con el plano $z = 0$.

3.3 Validación

3.3.1 Solución de Referencia

La deflexión del panel reforzado modelado con los planteamientos presentados será validada contrastando los resultados del análisis con elementos finitos contra los encontrados utilizando la teoría analítica presentada por Hughes: la deflexión de cada reforzador es encontrada planteando un modelo de viga que tiene en cuenta el aporte de la sección de placa adyacente a la rigidez de la viga (técnica conocida como *Strut Approach Idealization*). La figura 3.3 ilustra el modelado. Nótese que la distribución del área transversal causa que el eje neutro se encuentre en una posición baja.

Parámetro	Valor [mm]
A	1000
B	500
S	180
T1	10
T2	5
T3	7
W	40
H	50

Tabla 3.1 Parámetros de la geometría utilizados en el caso de validación.

La solución debe ser encontrada para dos tipos de vigas: las vigas extremas (primera y última), las cuales no poseen un aporte del área de placa simétrico (el aporte es $S_0 + 0.5S$), y las vigas centrales, las cuales si poseen un aporte simétrico (el aporte es $0.5S + 0.5S$). Los parámetros de la geometría utilizada para la validación se muestran en la tabla 3.1.

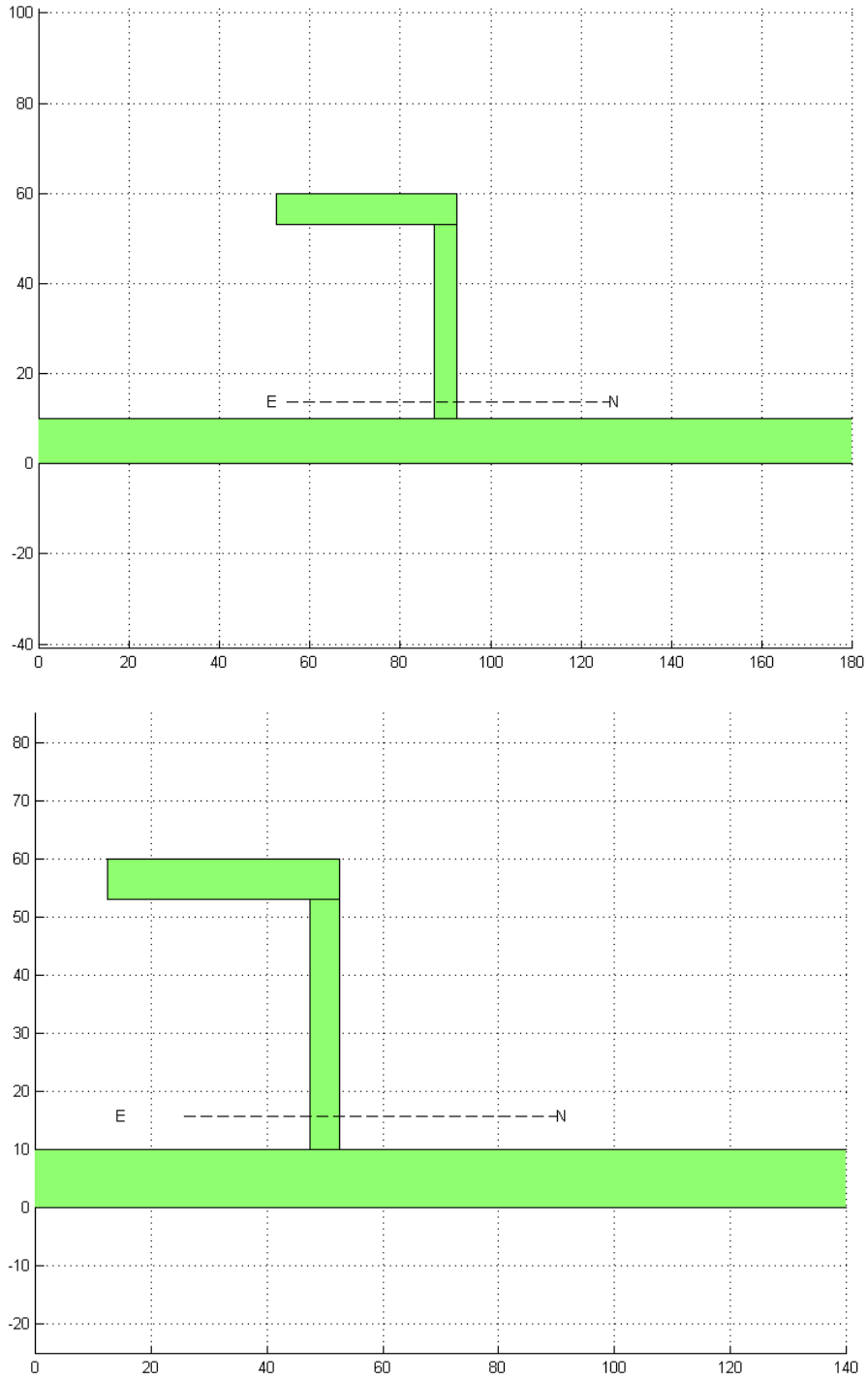

Figura 3.3 Sección transversal de reforzador con base (a) simétrica y (b) no simétrica.

Para la deflexión de una viga elástica con las condiciones de carga y restricciones presentadas (viga bajo carga distribuida uniformemente, empotrada en un extremo y simplemente apoyada en el otro extremo), se conoce la siguiente solución analítica:

$$w(y) = \frac{q}{48EI} y^2(3L^2 - 5Ly + 2y^2)$$

$$w_{max} = \frac{qL^4}{185EI} \; en \; y = 0.5785L$$

La solución es válida para la deflexión en dirección normal a la del eje neutro. Como el caso de estudio presenta una viga con sección transversal asimétrica, las deformaciones deben ser calculadas siguiendo las direcciones indicadas por los ejes principales de inercia, y proyectadas de vuelta a las direcciones originales. La tabla 3.2 muestra un resumen de los resultados numéricos encontrados y la figura 3.4 muestra la forma de deformación sobre las vigas.

Viga	Tipo 1	Tipo 2
$\theta_P \; [deg]$	-2.84	-13.44
$I_{XXP} \; [mm^4]$	7.5614e5	6.0704e5
$I_{YYP} \; [mm^4]$	4.9835e6	2.8076e6
$DZ_{MAX} \; [mm]$	4.9044e-5	5.0671e-5

Tabla 3.2 Resultados de Referencia.

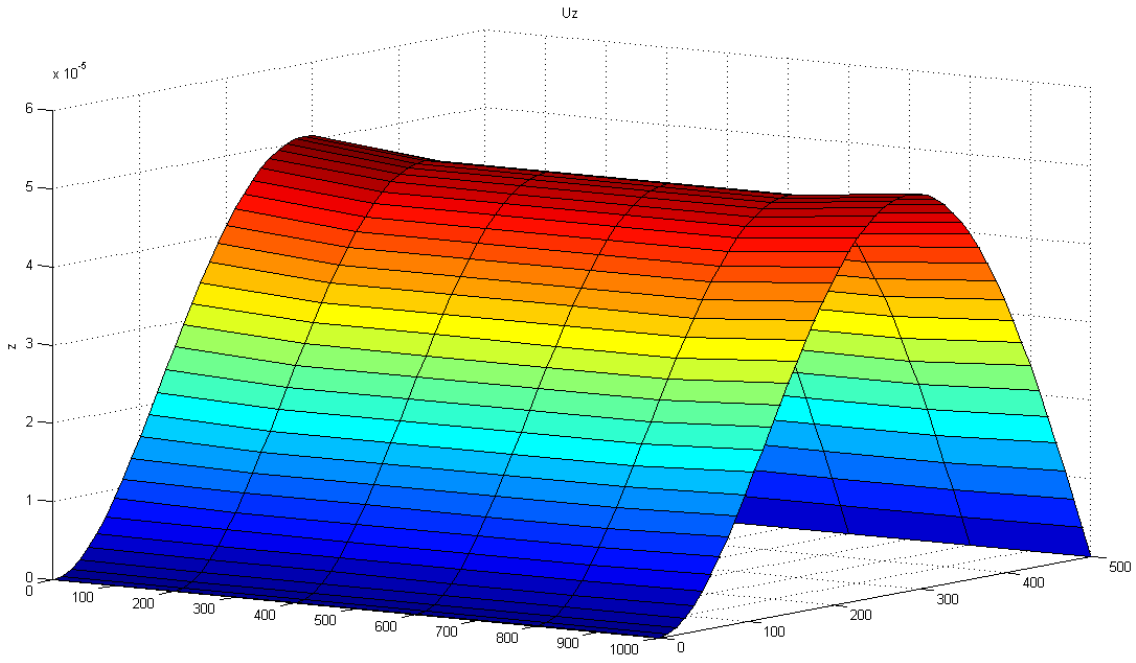

Figura 3.4 Deformación del panel, resultado analítico. Cada línea corresponde a un reforzador.

3.3.2 Solución Numérica y Comparación

En la simulación se encontró la forma de deformación mostrada en la figura 3.5. Con el fin de comparar la solución numérica con la analítica y analizar la convergencia, se registra el valor de deformación transversal y esfuerzo de Von Misses en el punto con coordenadas $(S0 + 2S, B/2,0) = (410,250,0)$. La figura 3.6 muestra las curvas de convergencia y la tabla 3.3 muestra los valores encontrados, y las respectivas diferencias con respecto a la aproximación teórica.

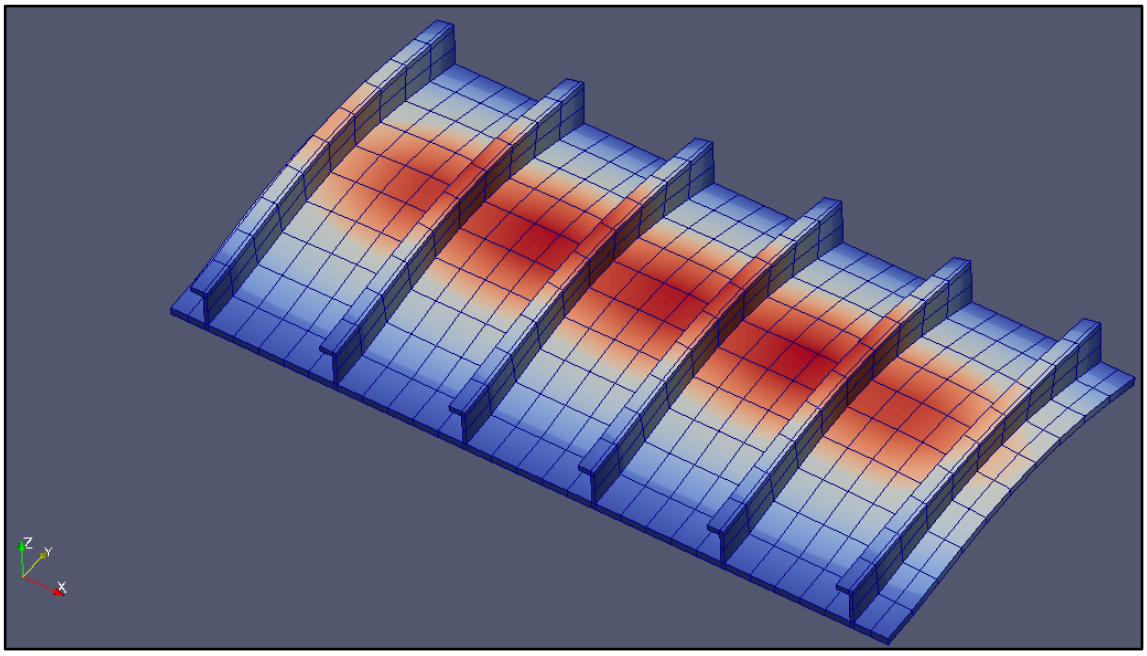

Figura 3.5 Cuerpo deformado, simulación numérica.

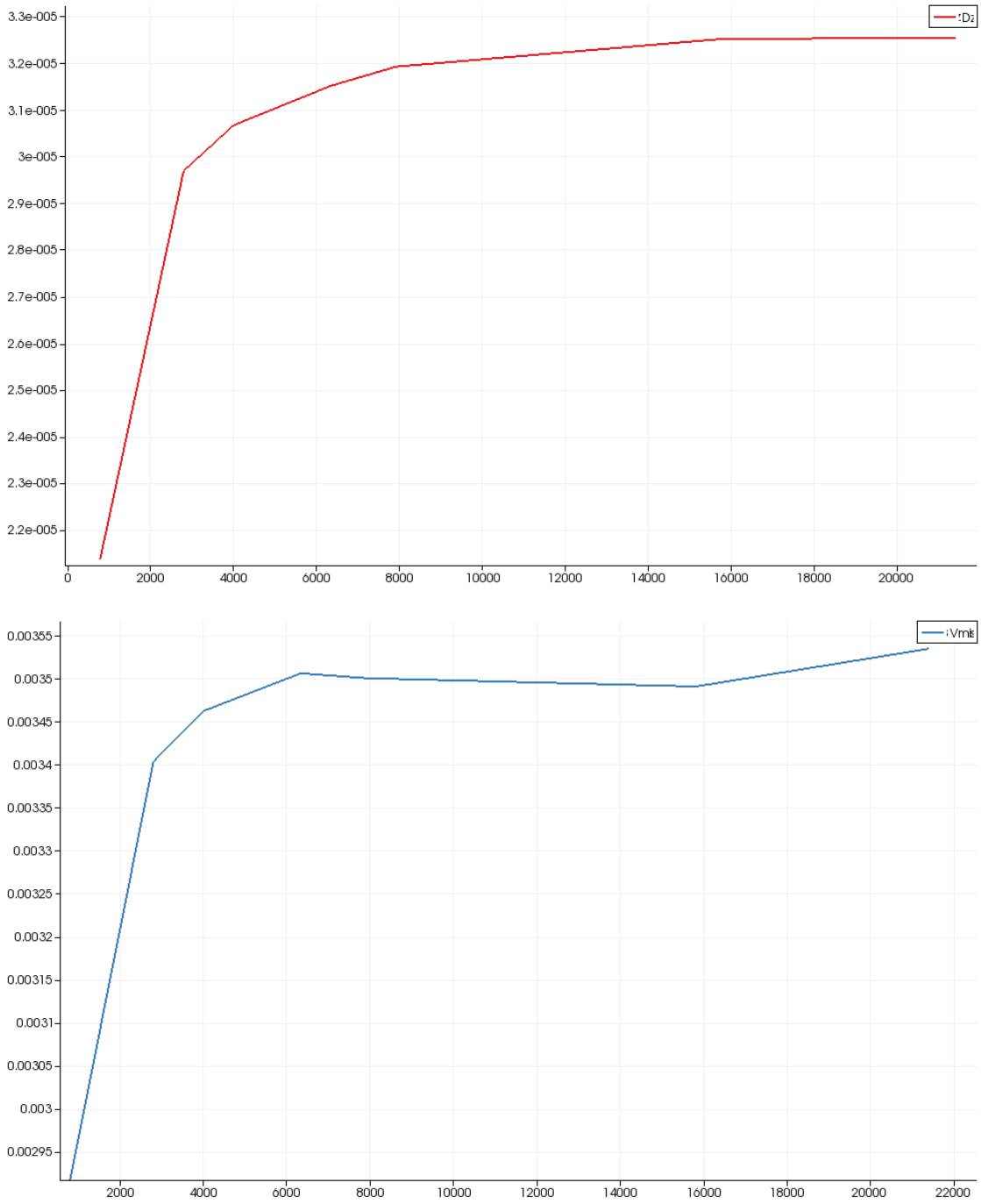

Figura 3.6 Numero de Elementos vs. A) Deflexión y b) Esfuerzo de Von Misses en el punto (400, 250, 0)

P(410,250,0)	FEM	Teórico	% Error
DZ	3.25e-5	4.7126e-5	-31.1

Tabla 3.3 Comparación de resultados teóricos y numéricos.

3.4 Esquema de Optimización

3.4.1 Función objetivo y restricciones

El merito seleccionado para el diseño estructural del caso de estudio es el de peso mínimo. En términos de los parámetros de la geometría, la función objetivo a optimizar se encuentra dada por:

$$f(x) = \rho B\big(A\,T1 + N_S(H\,T2 + (W - T2)\,T3)\big)$$

Donde ρ es la densidad del material utilizado. Todos los parámetros de la geometría serán variables del diseño, a excepción de las longitudes A y B, que serán mantenidas fijas. El número de reforzadores N_S es un número entero, por lo que esta información será reemplazada en el vector de parámetros por la distancia entre reforzadores S. El número de reforzadores y el espaciamiento se encuentran relacionados por la ecuación $N_S = floor(A/S)$. De esta forma el espacio de búsqueda queda determinado por el vector de números reales:

$$x = \{S, T1, T2, T3, W, H\}$$

Dos grupos de restricciones son identificados: las restricciones que mantienen la geometría valida, y las restricciones que aseguran la funcionalidad y seguridad del diseño:

$S > 0$	$T3 > 0$	$T3 < H$	$I_{xx} > I_{min}$
$T1 > 0$	$W > 0$	$W > T2$	$\delta_{max} < \delta_{per}$
$T2 > 0$	$H > 0$	$S_{max} > S > W$	$\sigma^P_{max} < S_y$

Donde I_{xx} es el momento de inercia del área del reforzador, δ_{max} es la máxima deflexión del panel, σ^P_{max} el máximo esfuerzo principal de Von Misses, e I_{min}, δ_{per} y S_y sus respectivos valores limites.

Las restricciones deben ser expresadas en la forma $g_i \leq 0$, con el fin de ser utilizadas en el método de la función de penalidad. Una tolerancia es utilizada

para asegurar que la restricción sea menor que cero, y para evadir errores numéricos. La tabla 3.4 recoge la lista de funciones de restricción.

i	g_i
1	$-S$
2	$-T1$
3	$-T2$
4	$-T3$
5	$-W$
6	$-H$
7	$T3 - H$
8	$T2 - W$
9	$W - S$
10	$S - S_{max}$
11	$I_{min} - I_{xx}$
12	$\delta_{max} - \delta_{per}$
13	$\sigma_{max}^{P} - S_y$

Tabla 3.4 Lista de funciones de restricción en la forma $g_i \leq 0$

3.4.2 Función de Penalidad

Con el fin de incluir la información de las restricciones en la función objetivo, esta será aumentada utilizando el método de la función de penalidad:

$$F(x) = f(x) + H(x)$$

$$H(x) = \sum_i \theta\big(q_i(x)\big) q_i(x)^{\gamma\left(q_i(x)\right)}$$

$$q_i(x) = \max\{g_i(x), 0\}$$

Los factores $\theta(q_i)$ y $\gamma(q_i)$ son seleccionados como funciones definidas por partes, como se muestra a continuación:

$$\gamma(q_i) = \begin{cases} 1 \rightarrow q_i \leq 1 \\ 2 \rightarrow q_i > 1 \end{cases}$$

$$\theta(q_i) = \begin{cases} 100 \rightarrow q_i < 0.001 \\ 150 \rightarrow q_i \leq 0.1 \\ 200 \rightarrow q_i \leq 1 \\ 300 \rightarrow q_i > 1 \end{cases}$$

3.5 Rutina de Optimización

La implementación del algoritmo de búsqueda se realizo como un programa modular, escrito en el lenguaje de programación Python. Cada modulo realiza una tarea específica, como construir la malla, evaluar la función objetivo, extraer los resultados de la simulación, aplicar las reglas de evolución o inicializar el esquema. La figura 3.7 muestra un esquema general de la relación entre los módulos y sus respectivos objetivos (los módulos dependen funcionalmente de sus respectivos inferiores).

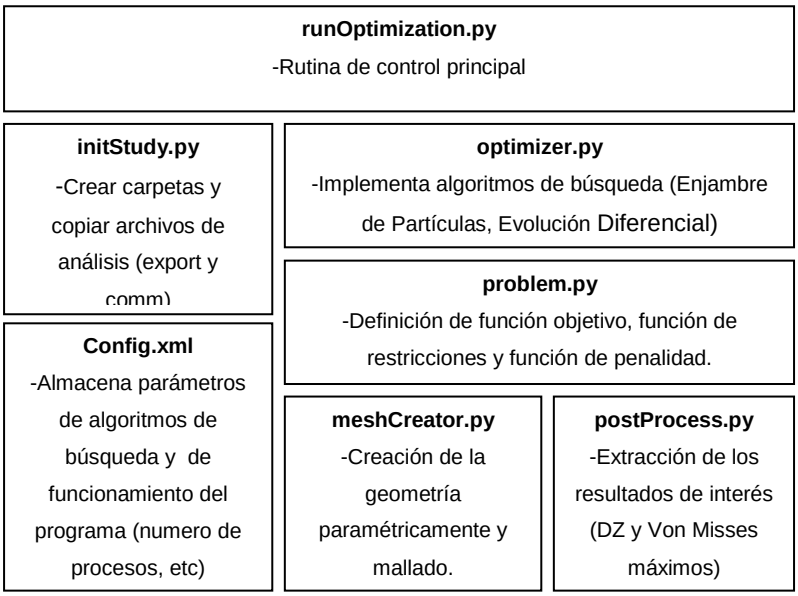

Figura 3.7 Diagrama de pila del esquema de optimización.

La arquitectura implementada permite reutilizar la mayor cantidad de código al momento de cambiar de problema: para tal fin, solo es necesario reescribir los

módulos 35roblema.py y sus dependencias, meshCreator.py y postProcess.py, esto es, actualizar las piezas de código relacionadas con el problema en específico.

3.6 Algoritmo de Búsqueda

Los algoritmos de búsqueda implementados son el Método del Enjambre de Partículas y la Evolución Diferencial. A pesar de que ambos métodos demandan grandes costos computacionales, en especial al utilizar simulaciones numéricas para la evaluación de la función objetivo, se logra realizar una optimización al código para acelerar la búsqueda, esto es, paralelizando la ejecución de ciertas secciones de código. Aunque los métodos no son inherentemente paralelos, la implementación llevada a cabo logra acelerar la ejecución.

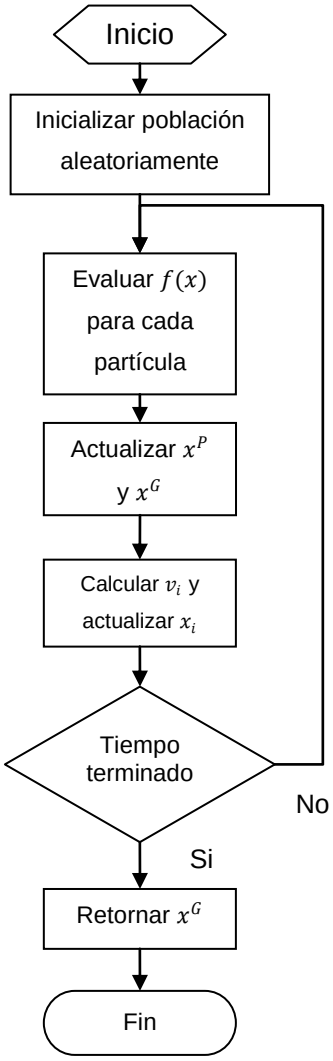

Figura 3.8 Diagrama de flujo de la Optimización por Enjambre de Partículas

La figura 3.8 muestra el diagrama de flujo general del método de enjambre de partículas como es implementado. Aunque la mayoría del flujo del algoritmo sigue en forma secuencial, identificar que el cuello de botella ocurre en la evaluación de la función objetivo para todas las soluciones candidatas permite visualizar la estrategia de optimización, esto es, realizar la evaluación en paralelo.

Idealmente, la mejor solución es evaluar la función objetivo para cada partícula en paralelo, pero el consumo de memoria de cada simulación lo hace prohibitivo. Para seleccionar el número de simulaciones a realizar

simultáneamente, el consumo de memoria máximo realizado por la simulación debe ser registrado, y en base a la razón entre la memoria disponible y tal, debe seleccionarse dicho número, dejando cierto margen para el correcto funcionamiento del sistema. De no respetarse la regla, el sistema se verá obligado a hacer uso de memoria virtual para almacenamiento de datos, produciéndose una penalización en tiempo que sería contraproducente a los fines de la técnica. Una técnica idéntica es implementada para el algoritmo de Evolución Diferencial.

4. Resultados

En primer lugar se propuso el problema de encontrar la topología optima para las condiciones planteadas en el capitulo anterior, y una presión de magnitud unitaria. Luego, el análisis de los resultados encontrados puso en evidencia la necesidad de realizar un segundo estudio, con el fin de optimizar aun más la topología. En el segundo caso de estudio, el grosor de placa $T1$ es mantenido constante, por razones expuestas en la discusión de resultados. Un tercer caso de estudio es realizado, ya que la presión aplicada en los dos primeros casos no representa un caso realista. Para remediar la situación, la presión de una columna de agua de 10 metros es calculada, y un factor de diseño de 5 es aplicado:

$$P = \rho g h = 1000\,\frac{kg}{m^3} * 9.8\,\frac{m}{s^2} * 10\,m * 5 = 490\,000\,\frac{N}{m^2} = 0.49\,MPa$$

$$P = 0.5\,MPa$$

Los resultados para las topologías óptimas encontradas por los algoritmos de búsqueda implementados se presentan de la siguiente forma: la tradicional curva de convergencia, seguida de los parámetros encontrados. Para el tercer caso, se ilustra la topología encontrada.

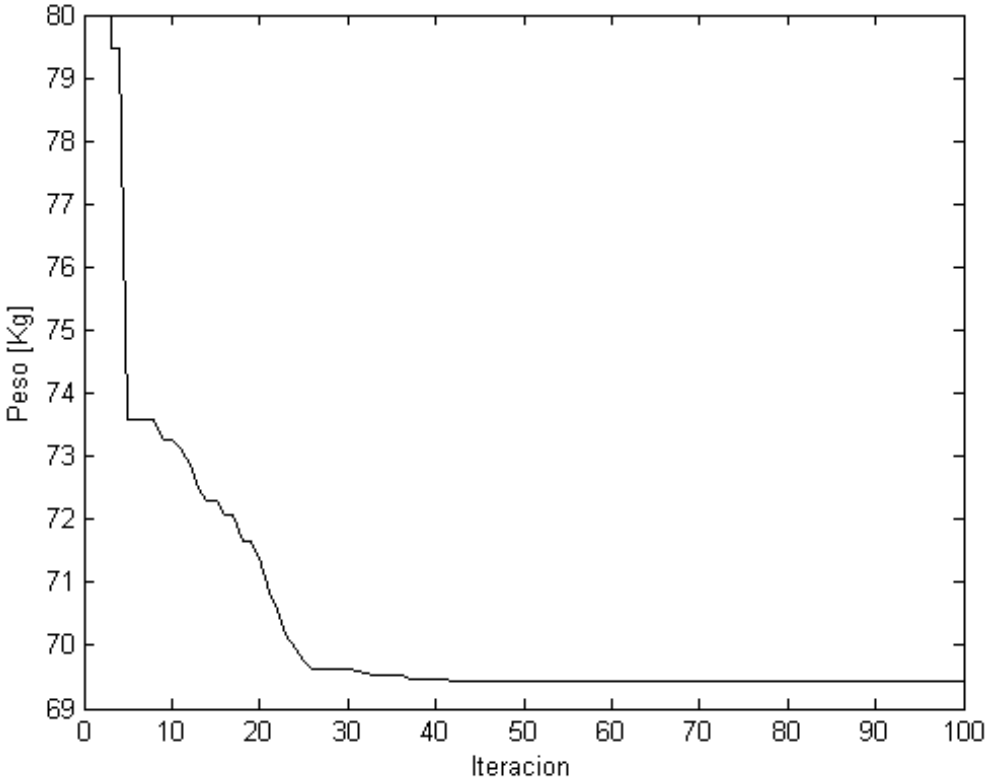

Figura 4.1. Convergencia del Método de Enjambre de Partículas en el caso de estudio 1.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	412.73	25.1	0.1	0.1	22.64	22.34	69.44

Tabla 4.1. Parámetros Encontrados por el Método del Enjambre de Partículas en el caso de estudio 1.

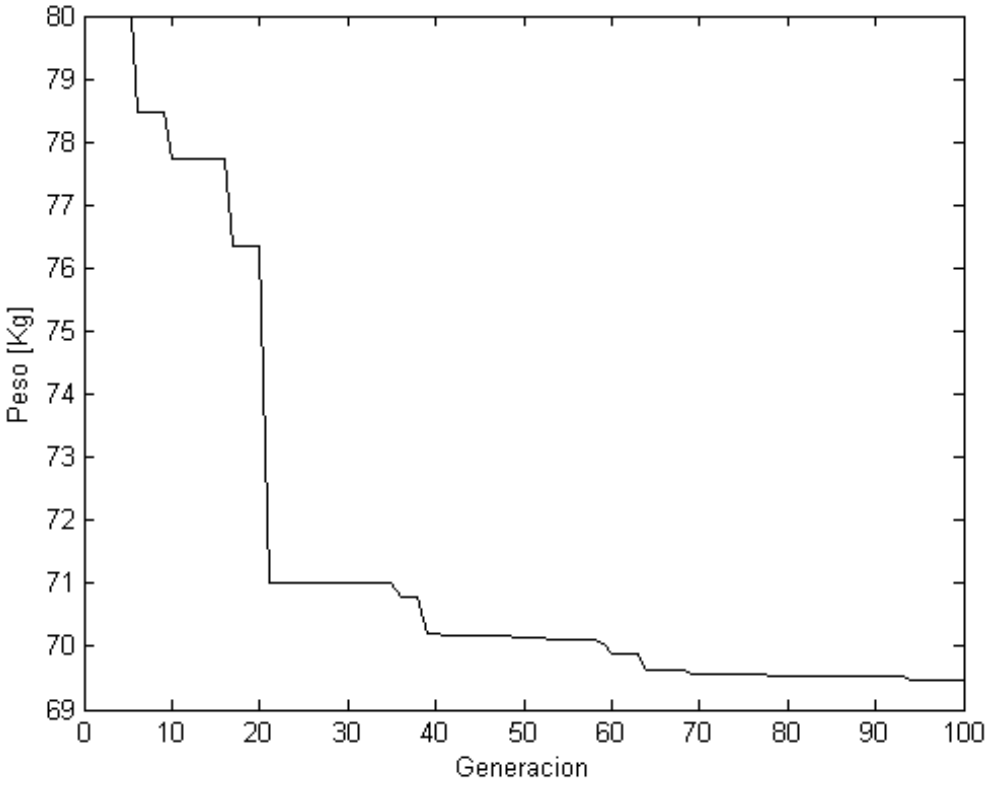

Figura 4.2. Convergencia del Método de Evolución Diferencial en el caso de estudio 1.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	492.31	25.1	1.25	2.27	2.02	4.84	69.47

Tabla 4.2. Parámetros Encontrados por el Método de Evolución Diferencial en el caso de estudio 1.

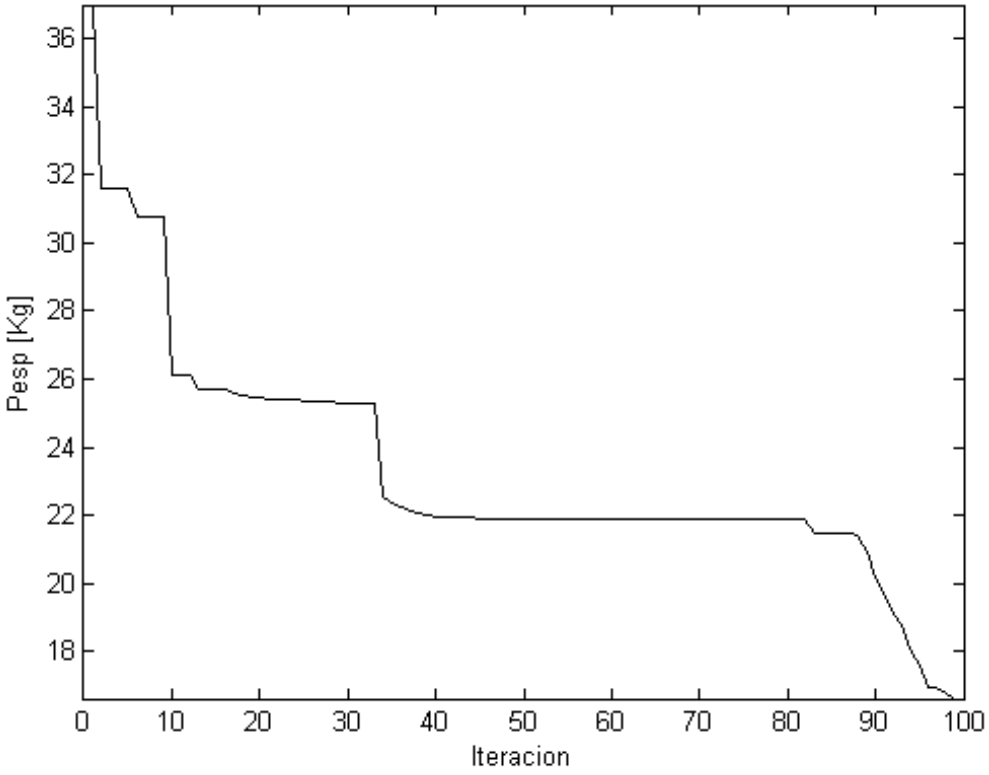

Figura 4.3. Convergencia del Método de Enjambre de Partículas en el caso de estudio 2.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	484.84	6	14.11	2.5e-3	43.25	0.21	16.61

Tabla 4.3. Parámetros Encontrados por el Método del Enjambre de Partículas en el caso de estudio 2.

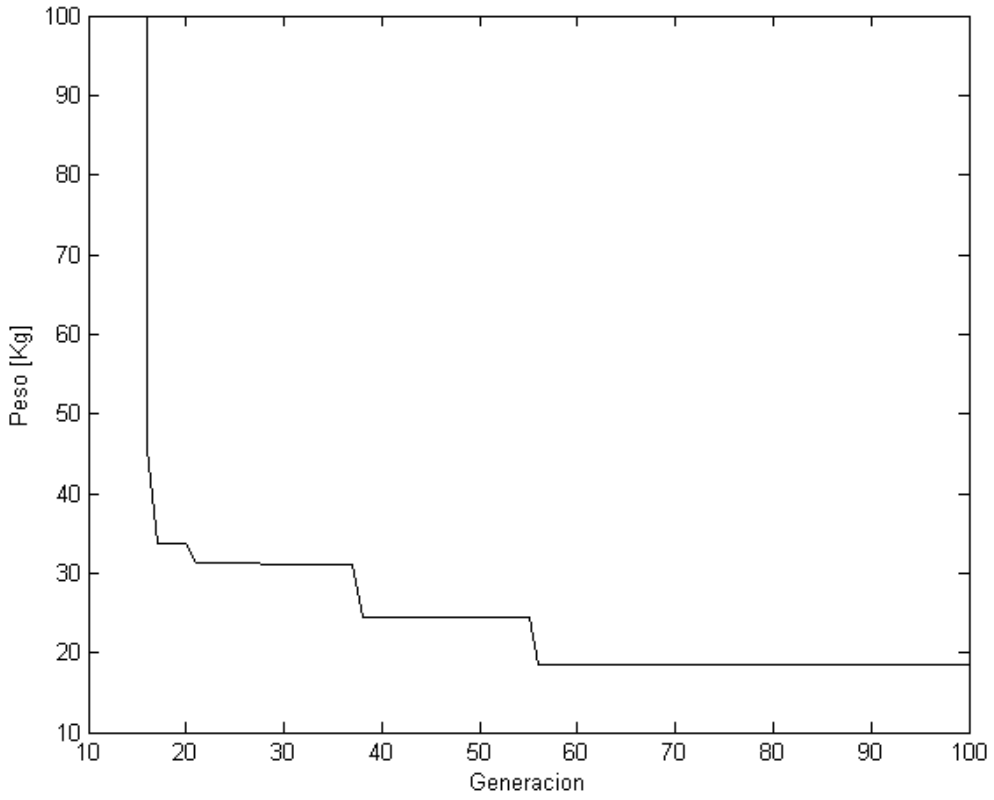

Figura 4.4. Convergencia del Método de Evolución Diferencial en el caso de estudio 2.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	336.28	6	13.34	5.00	13.62	11.42	18.41

Tabla 4.4. Parámetros Encontrados por el Método de Evolución Diferencial en el caso de estudio 2.

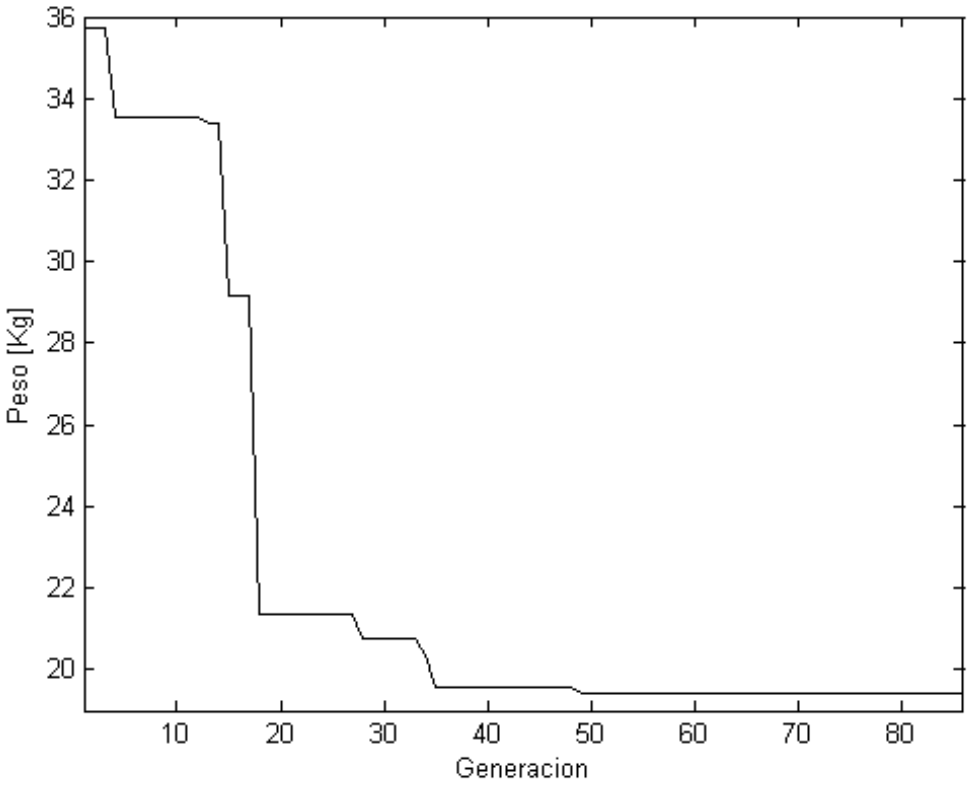

Figura 4.5. Convergencia del Método de Evolución Diferencial en el caso de estudio 3.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	139.56	6	2.24	3.20	22.21	25.20	19.44

Tabla 4.5. Parámetros Encontrados por el Método del Evolución Diferencial en el caso de estudio 3.

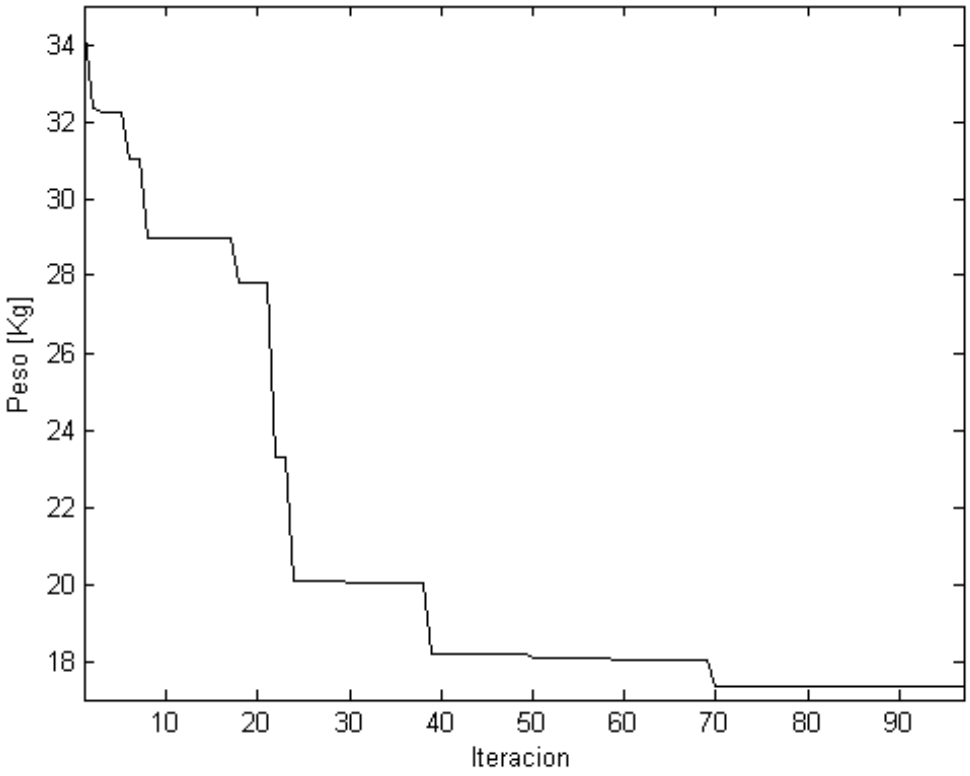

Figura 4.6. Convergencia del Método de Enjambre de Partículas en el caso de estudio 3.

A	B	S	T1	T2	T3	W	H	Peso [Kg]
700	500	143.96	6	0.63	0.12	35.17	53.31	17.33

Tabla 4.6. Parámetros Encontrados por el Método del Enjambre de Partículas en el caso de estudio 3.

4.1 Discusiones

4.1.1 Caso 1

Los resultados arrojados por los algoritmos de búsqueda deben ser examinados e interpretados cuidadosamente. En especial, debe buscarse casos en que las soluciones encontradas contengan restricciones activas. De presentarse tal situación, nuevas experimentaciones deben ser realizadas, con modificaciones que, por ejemplo, cambien las restricciones (de ser posible) o las condiciones de contorno, o la topología del elemento.

Esta situación se ha presentado en la primera experimentación realizada. Según los resultados mostrados en la tabla 4.1, en la topología optima encontrada por el método del enjambre de partículas se encuentran activas tres restricciones, a nombrar, las correspondientes a los espesores de placa y de reforzador. Mientras el espesor de placa $T1$ se eleva hasta su máximo valor permitido, los espesores de reforzador $T2$ y $T3$ son disminuidos hasta el menor valor permitido. Anótese que la tolerancia numérica utilizada para la evaluación del cumplimiento de una restricción es de $0.1\ [mm]$, por lo cual los espesores no son los valores extremos $\{25, 0, 0\}$ sino $\{25.1, 0.1, 0.1\}$. Dicha tolerancia ha sido disminuida en las experimentaciones subsecuentes para aumentar la precisión del algoritmo de búsqueda.

Con respecto a los resultados obtenidos con el método de la evolución diferencial, mostrados en la tabla 4.2, puede notarse que la restricción del grosor de placa máximo también se encuentra activa, con $T1 = 25.1\ [mm]$. En este caso, las restricciones de los grosores mínimos de reforzador $T2$ y $T3$ no se encuentran activas. Por otro lado, como puede apreciarse en la figura 4.2, la convergencia del método de búsqueda en el respectivo caso resultó lenta, y el valor final de la respuesta no perduró muchas generaciones (compárese con la figura 4.1, donde la respuesta final perdura por muchas iteraciones).

Otra particularidad de los resultados es que, mientras el método de enjambre de partículas “intenta desaparecer” los reforzadores haciendo cero los grosores $T2$ y $T3$ mientras mantiene las longitudes W y H relativamente altas, el método de evolución diferencial se direcciona en el mismo sentido, pero esta vez los 4 parámetros son relativamente homogéneos en magnitud.

Teniendo en cuenta todos los aspectos presentados, puede intentarse presentar una interpretación de los resultados:

> *Para las condiciones de carga y las restricciones de geometría impuestas sobre el elemento estructural, el máximo grosor de placa permitido produce una rigidez suficiente como para satisfacer las necesidades de seguridad y funcionalidad; y dado este hecho, remover los reforzadores es la configuración que arroja el mínimo peso posible.*

4.1.2 Caso 2

Los resultados encontrados en el primer caso de experimentación eran inesperados. Con el fin de seguir evaluando los algoritmos de búsqueda y estudiando el elemento estructural, se plantea una segunda experimentación. En este caso, el grosor de placa $T1$ es mantenido fijo en un valor de $6\ [mm]$. Dicho valor es elegido por presentar un caso realista, y por ser muy inferior al grosor obtenido, en el caso 1, la configuración óptima de la topología debe encontrarse variando los parámetros de reforzador.

El método del enjambre de partículas arrojó una solución muy similar a las encontradas para el caso 1, con los parámetros de reforzador $T3 = 2.5x10^{-3}$ y $H = 0.21$ siendo muy bajos. Aunque es la solución de mayor merito encontrada, resulta poco interesante para fines prácticos. Nótese en la grafica 4.3 que al final de la curva se encuentra una pendiente negativa, mostrando que el algoritmo sigue reduciendo material al final de la búsqueda. Sin embargo, es

evidente que la región de búsqueda, y que al parecer contiene al óptimo global, es de poco interés práctico.

Por otro lado, el método de evolución diferencial arroja una topología valida sin parámetros pequeños. Siendo alrededor de un 10% más pesada que la respuesta encontrada por el método de enjambre de partículas, es obvio que no es el óptimo global; sin embargo, el hecho de que sea un óptimo local que puede ser construido, le da un valor agregado muy grande. Sin embargo, esta practicidad en la respuesta no es un hecho premeditado, y debe buscarse una modificación a la búsqueda si se desea encontrar repetitivamente este tipo de resultados.

Ante las dos situaciones planteadas por los algoritmos de búsqueda, parece entonces evidente que es necesario realizar modificaciones. La alternativa planteada es reforzar las restricciones sobre los parámetros de búsqueda, en este caso, aumentar los valores mínimos posibles de los parámetros de búsqueda.

4.1.3 Caso 3

Los resultados del caso numero 3 muestran que las modificaciones realizadas para presentar un caso realista han sido acertadas, en el sentido que producen una configuración cercana a los diseños encontrados en estructuras reales. Las figuras 4.7 a. y b. muestran la ilustración de las topologías optimas para el panel reforzado sometido a las condiciones del caso.

Nótese que la excepción a la regla es el resultado para los grosores de los reforzadores encontrados por el método de enjambre de partículas, los cuales son muy bajos. Si el hecho es tomado como un error, puede ser atribuido a problemas relacionados con el método de los elementos finitos y los bajos grosores combinados con pocos elementos. Por otro lado, puede tomarse como un resultado interesante y ser estudiada posteriormente las implicaciones de aplicar este tipo de topología.

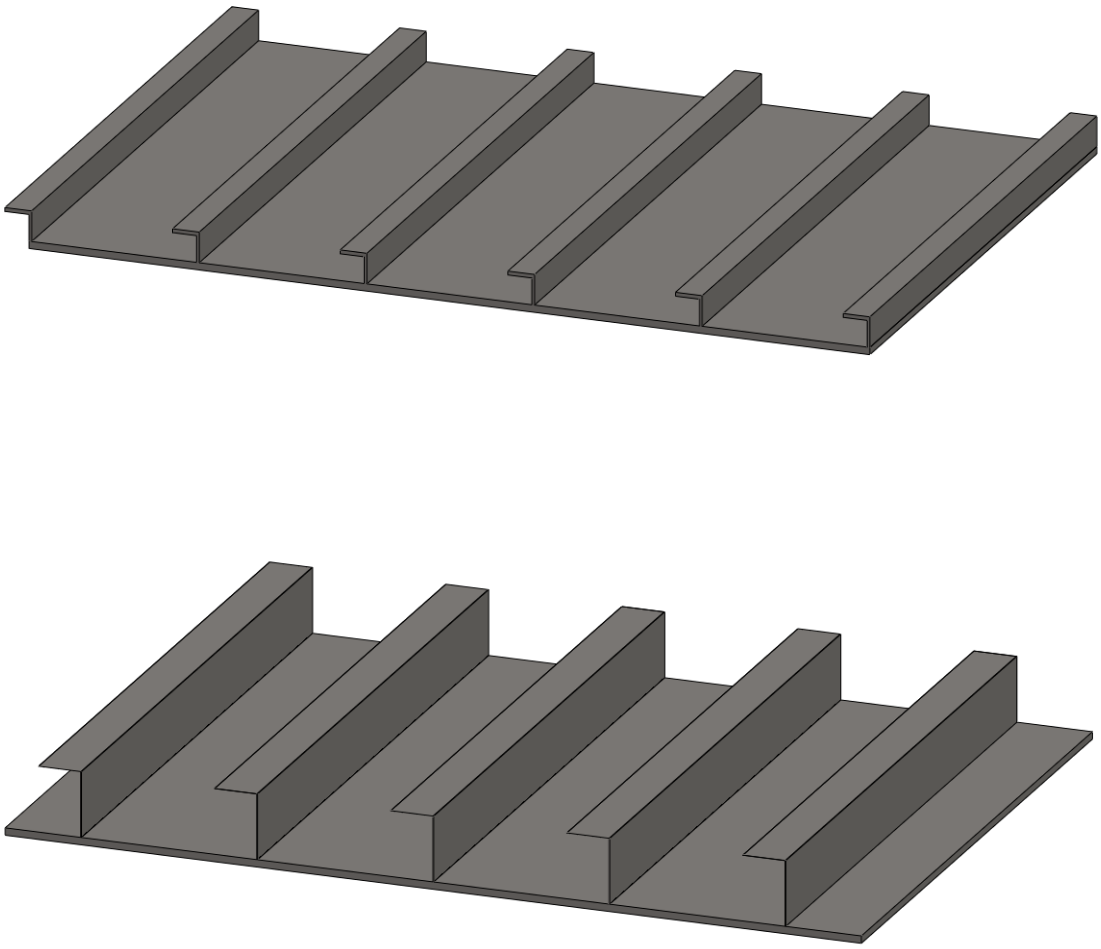

Figura 4.7 Topologías resultado del caso de estudio 3: a. Evolución Diferencial b. Enjambre de Partículas.

4.1.4 Desempeño

Además de los resultados de la búsqueda, para evaluar la practicidad de los métodos de búsqueda implementados es necesario mirar el costo computacional. Los tiempos empleados por las búsquedas cuyos resultados han sido mostrados se presentan en la tabla 4.5. Todas las búsquedas fueron programadas para realizar 100 iteraciones con poblaciones de 25 candidatos, lo cual permite realizar una comparación entre ellas. La gran dispersión en los

tiempos de búsqueda se debe a las decisiones tomadas en los algoritmos, en específico, a que para las soluciones candidatas con parámetros de geometría invalidantes no eran generadas geometrías, mallas ni soluciones; por otro lado, las librerías utilizadas para la generación de geometría y mallado no permitieron la paralelización de dichas tareas. De esta manera, el tiempo gastado en realizar una iteración depende directamente de la cantidad de candidatos que se encuentran en una zona de búsqueda valida; entonces, se interpreta que las búsquedas que tardaron más tiempo son las que eran capaces de llevar más candidatos hacia zonas de búsqueda validas. En comparación, no se ve una aparente dominación de ninguno de los algoritmos empleados para cumplir este objetivo.

La tendencia general observada es que en iteraciones tempranas de los algoritmos esta cantidad de soluciones validas es pequeña y que con la evolución aumentara, siendo las ultimas iteraciones las que tomaron más tiempo en realizarse. De dicho fenómeno también puede evidenciarse que las búsquedas que tardaron menos tiempo fueron las que realizaron menos evaluaciones de soluciones candidatas validas, una característica indeseada.

		Inicio	**Fin**	**Duración**
Caso 1	PSO	04/29/11 11:45:00	04/30/11 05:09:22	17:24
	DE	05/02/11 10:14:55	05/03/11 10:31:32	24:17
Caso 2	PSO	05/25/11 11:42:15	05/28/11 15:27:43	75:46
	DE	05/23/11 15:25:58	05/24/11 16:37:22	25:12
Caso 3	PSO	07/04/11 12:10:22	07/07/11 18:35:30	78:25
	DE	07/01/11 22:19:08	07/04/11 12:05:30	61:46

Tabla 4.5 Duración de las búsquedas

Con respecto al consumo de recursos durante las corridas de búsqueda, las tareas de mallado y solución resultan bastante costosas, sobre todo en el caso de ser paralelizadas. El orden de consumo de memoria se encuentra en los varios gigabytes inmediatos, los cuales se reflejaban en varias decenas

acumuladas al final de la búsqueda. El presente trabajo fue realizado en una maquina multiprocesador con masiva capacidad de memoria, por lo cual dichos impedimentos no resultaron prohibitivos. Sin embargo, una estación de trabajo promedio (con alrededor de una docena de gigabytes de memoria) no sería capaz de realizar las búsqueda en las mismas condiciones; para tal fin debería relajarse la demanda de recursos de memoria, disminuyendo ciertos parámetros de la paralelización, que se reflejarían en mayores tiempos de búsqueda.

5. Conclusiones

De cada uno de los objetivos planteados al inicio del proyecto deben sacarse conclusiones que permitan evaluar los resultados encontrados y juzgar las técnicas y decisiones utilizadas para llegar a ellos, de forma tal que sirvan de referencia para futuros trabajos con aéreas de interés en común.

En primer lugar, la definición de los requerimientos y parámetros de diseños fue relajada, y un caso de prueba genérico fue realizado, en vez de encontrar un caso realista. Dicha decisión fue tomada por cuestiones de tiempo y esfuerzo. Con respecto a la parametrización de la topología del panel, el alcance de las variables de diseño seleccionadas fue suficiente como para que las búsquedas fueran interesantes y permitieran comparar los algoritmos de búsqueda.

En segundo lugar, el esquema de análisis por elementos finitos implementado resulto satisfactorio para los fines de implementación de los algoritmos de búsqueda. Sin embargo, dicha satisfacción no resultó completa por dos razones: la primera es que las librerías utilizadas para generación de la geometría y mallado no permitieron paralelizar dichas tareas, restringiendo el máximo desempeño de los algoritmos como fueron concebidos; la segunda, es que el modelo de elementos finitos utilizado para modelar el panel es más bien simple, conscientemente sacrificando precisión en la respuesta en aras de ahorrar recursos computacionales.

En tercer y último lugar, el esquema de optimización numérica implementado para hallar la topología de menor peso resultó satisfactorio. Ambos algoritmos (método del enjambre de partículas y evolución diferencial) arrojaron buenos resultados que pusieron en evidencia aspectos antes ocultos del problema

estudiado, permitiendo de esta forma ajustar las restricciones y los parámetros del mismo, en miras a encontrar la respuesta óptima. Sin embargo, cabe resaltar que ambos algoritmos son altamente costosos en términos de recursos computacionales, y que su implementación en estaciones de trabajo regulares puede resultar prohibitiva.

En términos generales la metodología como un todo deja un resultado satisfactorio, dejando un campo abierto a futuras investigaciones en miras a mejorar aspectos como la selección de restricciones del problema, mejorar los algoritmos de búsqueda, o implementar nuevos algoritmos y compararlos con los ya implementados.

Bibliografía

[1] Bonte, M. H., Fourment, L., Do, T.-t., Boogaard, A. H., & Huetnik, J. (2010). Optimization of forging processes using Finite Element simulations, a comparision of Sequential Approximate Optimization and other algorithms. *Structural and Multidisciplinary Optimizatons , 2010* (42), 797-810.

[2] Brito, O., & Azevedo, A. d. *Modelo para calculo de resistencia e de estabilidade de grelhas.*

[3] Campanile, A., Mandarino, M., & Piscopo, V. *Application of the orthotropic plate theory to garage deck dimensioning.*

[4] Campanile, A., Mandarino, M., & Piscopo, V. *Considerations on dimensioning of garage decks.*

[5] Carlisle, A., & Dozier, G. *An Off-The-Shelf PSO.*

[6] Challis, V. J. (2010). A discrete level-set topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization , 2010* (41), 453-464.

[7] Colombo, D. (s.f.). An introduction to Code_Aster (Slides). Manchester.

[8] Darwich, W., Gilbert, M., & Tyas, A. (2010). Optimum structure to carry a uniform load between pinned supports. *Structural and Multidisciplinary Optimization , 2010* (42), 33-42.

[9] Dongqi, Z. (2004). *Stiffened plates subjected to in-plane load and lateral pressure.* Singapur.

[10] Eberhart, R. C., & Shi, Y. (2001). *Particle Swarm Optimization: developments, applications and resources.* IEEE.

[11] Gao, T., & Zhang, W. (2010). Topology optimization involving thermo-elastic stress loads. *Structural and Multidisciplinary Optimization , 2010* (42), 725-738.

[12] Gil-Martin, L. M., Hernandez-Montes, E., & Aschheim, M. (2010). Optimization of piers for retaining walls. *Structural and Multidisciplinary Optimization , 2010* (41), 979-987.

[13] Hart, C. G., & Vlahopoulos, N. (2010). An integrated multidisciplinary particle swarm optimization approach to conceptual ship design. *Structural and Multidisciplinary Optimization , 2010* (41), 481-494.

[14] Herencia, J. H., & Haftka, R. T. (2010). Structural optimization with limited number of element properties. *Structural and Multidisciplinary Optimization , 2010* (41), 817-820.

[15] Huang, X., & Xie, Y. M. (2010). Evolutionary topology optimization of continuum structures with an additional displacement constraint. *Structural and Multudisciplinary Optimization , 2010* (40), 409-416.

[16] Huang, X., & Xie, Y.-M. (2010). A further review of ESO type methods for topology optimization. *Structural and Multidisciplinary Optimization , 2010* (41), 671-683.

[17] Hughes, O. (1988). *Ship structural design, a rationally-based, computer-aided optimization aproach.* New Jersey: John Wiley & Sons.

[18] Kaufmann, M., Zenkert, D., & Wennhage, P. (2010). Integrated cost/weight optimization of aircraft structures. *Structural and Multidisciplinary Optimization , 2010* (41), 325-334.

[19] Kaveh, A., & Talatahari, S. (2010). An enhanced charged system search for configuration optimization using the concept of fields of forces. *Structural and Multidisciplinary Optimization* .

[20] Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE international conference on neural networks* (págs. 1942-1948). IEEE.

[21] Klarbring, A., & Torstenfelt, B. (2010). Dynamical systems and topology optimization. *Structural and Multidisciplinary Optimization , 2010* (42), 179-192.

[22] Kruzelecki, J., & Stawlarski, A. (2010). Optimal design of thin-walled columns for buckling under loadings controlled by displacements. *Structural and Multidisciplinary Optimization , 2010* (42), 305-314.

[23] Le, C., Norato, J., Bruns, T., Ha, C., & Tortorelli, D. (2010). Stress-based topology optimization for continua. *Structural and Multidisciplinary Optimization , 2010* (41), 605-620.

[24] Lellep, J., & Paltsepp, A. (2010). Optimization of inelastic cylindrical shells with internal supports. *Structural and Multidisciplinary Optimization , 2010* (41), 841-852.

[25] Li, C., Liu, Y., Zhou, A., Kang, L., & Wang, H. *A fast particle swarm optimization algorithm with cauchy mutation and natural selection strategy.*

[26] Li, Y., Tan, T., & Li, X. (2010). A gradient-based approach for discrete optimum design. *Structural and Multidisciplinary Optimization , 2010* (41), 881-892.

[27] LIU, C.-a. (2009). Dynamic PSO method for nonlinear constrained programming problems. *Journal of Communication and Computer , 6* (4), 6-8.

[28] Luke, S. (2010). *Essentials of Metaheuristics* (Online ed.). Lulu.

[29] Makrodimopoulos, A., Bhaskar, A., & Keane, A. J. (2010). A compliance based design problem of structures under multiple load cases. *Structural and Multidisciplinary Optimization , 2010* (42), 739-743.

[30] Mansour, A., & Liu, D. *Principles of naval architecture: strength of ships.* Society of Naval Architects and Marine Engineers (SNAME).

[31] Marczak, R. J. (2008). Optimization of elastic structures using boundary elements and a topological shape sensitivity formulation. *Latin American Journal of Solid and Structures* , 99-117.

[32] Mazurek, A., Baker, W. F., & Tort, C. (2010). Geometrical aspects of optimim truss like structures. *Structural and Multidisciplinary Optimization* .

[33] Moraldi, S., Fatahi, L., & Razi, P. (2010). Finite element model updating using bees algorithm. *Structural and Multidisciplinary Optimization , 2010* (42), 283-291.

[34] Niu, F., Xu, S., & Cheng, G. (2010). A general formulation of structural topology optimization for maximizing structural stiffness. *Structural and Multidisciplinary Optimization* .

[35] Oliveira, J. G., & Christopoulos, D. A. (1981). A practical method for the minimum weight design of stiffened plates under uniform lateral pressure. *Journal of Computer & Structures , 14* (5-6), 409-421.

[36] Parsopoulos, K. E., & Vrahatis, M. N. *Particle Swarm Optimization Method for Constrained Optimization Problems.* Patras, Greece.

[37] Pathak, K. K., & Sehgal, D. K. (2010). Gradientless shape optimization using artificial neural networks. *Structural and Multidisciplinary Optimization , 2010* (41), 699-709.

[38] Poli, R. (2007). *An analysis of publications on particle swarm optimisation applications.*

[39] Price, K. V., Storn, R. M., & Lampinen, J. A. (2005). *Differential Evolution, A Practical Approach to Global Optimization.* Berlin: Springer.

[40] Qiu, G. Y., & Li, X. S. (2010). A note on the derivation of global stress constraints. *Structural and Multidisciplinary Optimization , 2010* (40), 625-628.

[41] Ramani, A. (2010). A pseudo-sensitivity based discrete-variable approach to structural topology optimization with multiple materials. *Structural and Multidisciplinary Optimization , 2010* (41), 913-934.

[42] Shan, S., & Wing, G. G. (2010). Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization , 2010* (41), 219-241.

[43] Soremekun, G. A. (1997). *Genetic Algorithms for Composite Laminate Design Optimization.* Blacksburg, Virginia, USA.

[44] Stolpe, M. (2010). On some fundamental properties of structural topology optimization problems. *Structural and Multidisciplinary Optimization , 2010* (41), 661-670.

[45] Suresh, K. (2010). A 199-line Matlab code for Pareto-optimal tracing in topology optimization. *Structural and Multidisciplinary Optimization , 2010* (42), 665-679.

[46] Takezawa, A., Nishiwaji, S., Kitamura, M., & Silva, E. C. (2010). Topology optimization for designing strain-gauge load cells. *Structural and Multidisciplinary Optimization , 2010* (42), 387-402.

[47] Vilnay, O. (1983). A full orthotropic-plate method for double-bottom structures. *Journal of Constructional Steel Research , 3* (1), 19-27.

[48] Vu, V. T. (2010). Minimum weight design for toroidal pressure vessels using Differential Evolution and Particle Swarm Optimization. *Structural and Multidisciplinary Optimization , 2010* (42), 351-369.

[49] Wang, L., & Li, L.-p. (2010). An effective differential evolution woth level comparison for constrained engineering design. *Structural and Multidisciplinary Optimization , 2010* (41), 947-963.

[50] Winslow, P., Pellegrino, S., & Sharma, S. B. (2010). Multi-objective optimization of free-form grid structures. *Structural and Multidisciplinary Optimization , 2010* (40), 257-269.

[51] Wu, C.-Y., & Tseng, K.-Y. (2010). Topology optimization of structures using modified binary differential evolution. *Structural and Multidisciplinary Optimization , 2010* (42), 939-953.

[52] Wu, C.-Y., & Tseng, K.-Y. (2010). Truss structure optimization using adaptive multi-population differential evolution. *Structural and Multidisciplinary Optimization , 2010* (42), 575-590.

[53] Zakhama, R., Abdalla, M. M., Gurdal, Z., & Smaoui, H. (2010). Wind load modeling for topology optimization of continuum structures. *Structural and Multidisciplinary Optimization , 2010* (42), 157-164.

Anexos

A continuación se presenta el código fuente correspondiente a las rutinas de optimización y simulación por el método de los elementos finitos implementadas en el presente trabajo. De esta forma, cualquier persona interesada puede repetir las experimentaciones o modificar las rutinas y aprovechar la base implementada para realizar sus propios casos de estudio.

Las fuentes son publicadas aquí para fines de referencia y de uso académico, y bajo ninguna circunstancia pueden ser utilizadas para otro tipo de fines sin una autorización previa por escrito del autor o de la Universidad Tecnológica de Bolívar.

Introducción

El punto de entrada al programa es el script runOptimization.py. Al ejecutarlo, es creada una estructura de directorios conteniendo los archivos necesarios para las simulaciones por elementos finitos en la evaluación del merito de cada miembro de la población, esto es, los archivos de control (.comm) y de ejecución (.export). Los archivos de malla (.mmed) son generados dinámicamente según los parámetros de cada miembro de la población en las respectivas carpetas. Cada miembro contiene su propia carpeta de trabajo, identificado por la letra p seguida de un número entre cero y el número de miembros menos uno. A continuación se ilustra la estructura de directorios:

```
/Optimization/
     -/StudyFiles/
          -command.comm
          -run.export
     -/Script/
          -__init__.py
          -initStudy.py
          -meshCreator.py
          -optimizer.py
          -postprocess.py
          -problem.py
          -config.xml
     -/p0/
```

```
            -mesh.mmed
            -command.comm
            -run.export
            -Result.resu
      -/p1/
            -mesh.mmed
Etc..
```

Para el funcionamiento de las rutinas de mallado y solución, SALOME y Code_Aster deben estar instalados y funcionando en la maquina, junto a la siguiente configuración:

1. Deben existir enlaces simbólicos accesibles desde cualquier nivel de la línea de comandos (esto es, en /bin/) a los programas runSession (de SALOME) y asrun. Dicha configuración es descrita a fondo en el foro de www.caelinux.org.

2. Debe haber al menos una instancia de SALOME ejecutándose en el fondo y prestando sus respectivos servicios, preferiblemente en modo de consola.

3. Una versión de Python reciente debe estar instalada, que contenga las librerías de procesamiento multihilo, más exactamente, la clase Pool de la librería Multiprocessing, de la cual es utilizada la función Pool,map para la paralelización (ver anexo la fuente del script optimizer.py). En general se recomienda utilizar la última versión disponible de Python 2.

```
# Project: Stiffened Plate Optimization
#
# Name: runOptimization.py
#
# Objective: main script
#
# Author: Guille Giraldo, 2011

from time import strftime
from Scripts import initStudy, optimizer

params = initStudy.init() #This is a dictionary!

#Generic parameters
Np = int( params['population'] )
D = int( params['dimension'] )
PoolWorkers = int( params['processcount'] )

#PSO parameters
c1 = float( params['cognition'] )
c2 = float( params['confidence'] )
w = float( params['inertia'] )
K = float( params['constriction'] )
MaxIters = int( params['iterations'] )

#DE parameters
Fd = float( params['scalefactor'] )
Cr = float( params['crprobability'] )
MaxGens = int( params['generations'] )

log = open('Results.txt', 'a')
log.write("\nOptimization Run")
log.write("\nStart time: " + strftime('%x %X'))
log.close()

print "Start time: " + strftime('%x %X')

#Run optimizer
gBest = optimizer.PSoptimize(c1, c2, w, K, Np, D, MaxIters, PoolWorkers)
#gBest = optimizer.DEoptimize(Fd, Cr, Np, D, MaxGens, PoolWorkers)

log = open('Results.txt', 'a')
log.write("\nOptimum = " + str(gBest))
log.write("\nEnd time: " + strftime('%x %X'))
log.close()

print
print "    End of Optimization"
print "End time: " + strftime('%x %X')
print "\n\tOptimization Result: "
print "x* = " + str(gBest[1])
print "f(x*) = " + str(gBest[0])
print
```

```
# Project: Stiffened Plate Optimization
#
# Name: optimizer.py
#
# Objective: Optimization routines: - Particle Swarm Optimization
#                                   - Differential Evolution
# Author: Guille Giraldo, 2011

from copy import copy
from random import random
from problem import F, mesh
from multiprocessing import Pool

def PSoptimize(c1, c2, w, K, Np, D, MaxIters, PoolWorkers):

    print
    print "    Particle Swarm Optimization"
    print
    print "Cognition: " + `c1`
    print "Confidence: " + `c2`
    print "Inertia: " + `w`
    print "Constriction: " + `K`
    print "Particles: " + `Np`
    print "Dimension: " + `D`
    print "MaxIters: " + `MaxIters`
    print "PoolWorkers: " + `PoolWorkers`
    print

    #Data structures
    x = []
    v = []
    pBest = []

    for i in range(Np):
        #D = 6
        #x = [p,d,                t2,          t3,          w,               h]
        xi = [i,50.+450.*random(),20.*random(),20.*random(),10.+40.*random(),10.+40.*random()]
        vi = [50.*(random() - 0.5),20.*(random() - 0.5), 20.*(random() - 0.5), 20.*(random() - 0.5), 20.*(random() - 0.5) ]
        x.append( copy(xi) )
        v.append( copy(vi) )
        pBest.append(copy(xi))
    #end for

    print "    Initializing..."

    if PoolWorkers > 1:
        #Parallel
        pool = Pool(PoolWorkers)
        stats = map(mesh, x)
        fitness = pool.map(F, x)
    elif PoolWorkers == 1:
        #Sequential
        stats = map(mesh, x)
        fitness = map(F, x)
    #end if

    pBestFitness = copy(fitness)

    gBest = copy(x[0])
    gBestFitness = copy(fitness[0])
    persist = 1

    #Optimization Loop
    for it in range(MaxIters):

        print
        print "    Iteration #" + str(it + 1)
        print
        print "x*= " + `gBest`
        print "f(x*)= " + `gBestFitness`

        #Update Fitness
        if it > 0:
            if PoolWorkers > 1:
```

```
                #Parallel
                stats = map(mesh, x)
                fitness = pool.map(F, x)
            elif PoolWorkers == 1:
                #Sequential
                stats = map(mesh, x)
                fitness = map(F, x)
            #end if
        #end if

        #Update gBest and pBest
        newBest = False
        for i in range(Np):
            #gBest
            if fitness[i] < gBestFitness:
                gBest = copy(x[i])
                gBestFitness = copy(fitness[i])
                #newBest = True
            #end if
            #pBest
            if fitness[i] < pBestFitness[i]:
                pBest[i] = copy(x[i])
                pBestFitness[i] = copy(fitness[i])
            #end if
        #end for

        #Update Velocities and Positions
        for i in range(Np):
            for j in range(D):
                #Velocities
                r1 = random()
                r2 = random()
                v[i][j] = K*( w*v[i][j] + c1*r1*(gBest[j+1] - x[i][j+1]) + c2*r2*( pBest[i][j+1] - x[i][j+1]) )
                #Positions
                x[i][j+1] = x[i][j+1] + v[i][j]
        #end for

        #Log file
        log = open('Results.txt', 'a')
        log.write( "\n" + str([it, gBestFitness]) )
        log.close()

        res = open('Evolution.txt', 'a')
        res.write( "\n" + str([gBest, gBestFitness]))
        res.close()

        #Convergence criterium
        #if newBest : persist = 1
        #else : persist = persist + 1
        #if persist > 9:
        #   return [fitne[best], x[best]]
        #end if
    #end Optimization loop

    return [gBestFitness, gBest]
#end PSoptimize()

def DEoptimize(Fd, Cr, Np, D, MaxGens, PoolWorkers):

    print
    print "    Differential Evolution Optimization"
    print
    print "Scale Factor: " + `Fd`
    print "Crossover probability: " + `Cr`
    print "Population: " + `Np`
    print "Dimension: " + `D`
    print "Max Generations: " + `MaxGens`
    print "PoolWorkers: " + `PoolWorkers`
    print

    x = []
    u = []
    fx = []
    fu = []
```

```
149       best = 0
150       persist = 1
151
152       for i in range(Np):
153
154           #x = [p,d,                  t2,          t3,          w,                h]
155           xi = [i,100.+900.*random(),20.*random(),20.*random(),10.+40.*random(),10.+40.*random()]
156           x.append(copy(xi))
157           u.append(copy(xi))
158       #end for
159
160       print "    Initializing..."
161       if PoolWorkers > 1:
162           pool = Pool(PoolWorkers)
163           stast = map(mesh, x)
164           fx = pool.map(F, x)
165       elif PoolWorkers == 1:
166           stats = map(mesh, x)
167           fx = map(F, x)
168       #end if
169
170       for gen in range(MaxGens):
171
172           print
173           print "    Generation #" + str(gen + 1)
174           print
175           print "x*= " + `x[best]`
176           print "f(x*)= " + `fx[best]`
177
178           #Mutation and crossover
179           for i in range(Np):
180
181               rg = int( random() * Np )
182               r0 = (i + rg) % Np
183
184               r1 = copy(r0)
185               r2 = copy(r0)
186
187               while r1 == i or r1 == r0 :
188                   r1 = int( random() * Np )
189               #end while
190
191               while r2 == i or r2 == r0 or r2 == r1:
192                   r2 = int( random() * Np )
193               #end while
194
195               r = random()
196               jrand = int( random()*D )
197
198               for j in range(D):
199
200                   if r < Cr or j == jrand:
201
202                       u[i][j+1] = x[r0][j+1] + Fd * ( x[r1][j+1] - x[r2][j+1] )
203                   else:
204                       u[i][j+1] = x[i][j+1]
205                   #end if
206               #end for
207           #end for
208
209           if PoolWorkers > 1:
210               stats = map(mesh, u)
211               fu = pool.map(F,u)
212           elif PoolWorkers == 1:
213               stats = map(mesh, u)
214               fu = map(F, u)
215           #end if
216
217           #Selection
218           newBest = False
219           for i in range(Np):
220
221               if fu[i] < fx[i]:
222                   x[i] = copy(u[i])
223                   fx[i] = copy(fu[i])
```

```
224             #end if
225             if fx[i] < fx[best]:
226                 best = copy(i)
227                 newBest = True
228             #end if
229         #end for
230 
231         #Log file
232         log = open('Results.txt', 'a')
233         log.write("\n" + str([gen, fx[best]]) )
234         log.close()
235 
236         res = open('Evolution.txt', 'a')
237         res.write( "\n" + str([x[best], fx[best]]))
238         res.close()
239 
240         '''
241         #Convergence criterium
242         if newBest : persist = 1
243         else : persist = persist + 1
244         if persist > 9:
245             return [fx[best], x[best]]
246         #end if
247         '''
248     #end for
249 
250     return [fx[best], x[best]]
251 #end optimize()
```

```python
# Project: Stiffened Plate Optimization
#
# Name: problem.py
#
# Objective: problem definition module: fitness and constraint functions
#
# Author: Guille Giraldo, 2011

import meshCreator
import postprocess
from os import waitpid
from subprocess import Popen
from math import pi

studyDir = "/home/guillermo/OPPR3/"

#Global Parameters
#Geomtric
A = 700.
B = 500.
t1 = 6.
dmin = 200.
Imin = 27500.
dzmax = 10.
dmax = 500.

#Material
Sy = 200. #[MPa]
rho = 7.9e-6 #[Kg/mm3]

#Search
e = 0.1
tol =  -1e-5
penaltyFitness = 100
penaltySigma = 1.1*Sy
penaltyDz = 1.1*dzmax
#end Global Parameters

def I(t2, t3, w, h):
    #Reinforcer Area Moment of Inertia???
    #return ( t2*h**3 )/12 + ( ( 0.5*(h - t3) ) **2 )*( (w - t2)*t3**3 )/12
    return 1.1*Imin
#end I

def mesh(x):

    particle = int(x[0])
    d = x[1]
    t2 = x[2]
    t3 = x[3]
    w = x[4]
    h = x[5]

    #Feasible geomery constraints
    g1 = -d
    g2 = -t2
    g3 = -t3
    g4 = -w
    g5 = -h
    g6 = t3 - h
    g7 = t2 - w
    g8 = w - d
    g9 = d - dmax

    if g1 > tol or g2 > tol or g3 > tol or g4 > tol or g5 > tol or g6 > tol or g7 > tol or g8 > tol or g9 > tol:
        #Invalid geometry, abort
        return -1

    folderName = studyDir + "p" + str(particle) + "/"

    #Create Mesh
    mArgs = str(A) +" "+ str(B) +" "+ str(d) +" "+ str(t1) +" "+ str(t2) +" "+ str(t3) +" "+ str(w) +" "+ str(h) +" "+ str(folderName)

```

```
74      log = open(folderName + "log.txt", 'a')
75      mProc = Popen("runSession python " + studyDir + "Scripts/meshCreator.py " + mArgs, shell=True,
    stdout=log)
76
77      meshingStatus = waitpid(mProc.pid, 0)[1]
78
79      log.close()
80
81      return meshingStatus
82  #end mesh()
83
84  def solve(particle):
85
86      folderName = studyDir + "p" + str(particle) + "/"
87
88      #Solver call
89      log = open(folderName + "log.txt", 'a')
90      sProc = Popen("as_run --run " + folderName + "run.export", shell=True, stdout=log)
91
92      solvingStat = waitpid(sProc.pid, 0)[1]
93      log.close()
94
95      if solvingStat != 0:
96          return [penaltySigma, penaltyDz]
97
98      #Extract results
99      sigma = postprocess.getMaxSigma(folderName)
100     dz = postprocess.getMaxDZ(folderName)
101
102     return [sigma, dz]
103 #end solve(x)
104
105 def f(x):
106
107     d = x[1]
108     t2 = x[2]
109     t3 = x[3]
110     w = x[4]
111     h = x[5]
112     Ns = int(A/d) + 1
113
114     g1 = -d
115     g2 = -t2
116     g3 = -t3
117     g4 = -w
118     g5 = -h
119     g6 = t3 - h
120     g7 = t2 - w
121     g8 = w - d
122     g9 = d - dmax
123
124     if g1 > tol or g2 > tol or g3 > tol or g4 > tol or g5 > tol or g6 > tol or g7 > tol or g8 > tol
    or g9 > tol:
125         return penaltyFitness
126     else:
127         return B*(t1*A + Ns*(t2*h + t3*(w - t2)))*rho
128 #end f
129
130 def g(x):
131     #Constraints in the form g[i] <= 0
132
133     p = int(x[0])
134     d = x[1]
135     t2 = x[2]
136     t3 = x[3]
137     w = x[4]
138     h = x[5]
139     Ns = int(A/d) + 1
140
141     g1 = -d
142     g2 = -t2
143     g3 = -t3
144     g4 = -w
145     g5 = -h
146     g6 = t3 - h
```

```
147       g7 = t2 - w
148       g8 = w - d
149       g9 = d - dmax
150
151       #Avoid treating invalid geometries in stress evaluation
152       if g1 < tol and g2 < tol and g3 < tol and g4 < tol and g5 < tol and g6 < tol and g7 < tol and
      g8 < tol and g9 < tol:
153           [sig, dz] = solve(p)
154       else:
155           sig = penaltySigma
156           dz = penaltyDz
157       #end if
158
159       g10 = dz - dzmax
160       g11 = sig - Sy
161
162       return [g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11]
163   #end g(x)
164
165   def F(x):
166
167       #Constraints
168       q = g(x)
169
170       H = 0.0
171
172       for qi in q[:]:
173           # qi = max{ 0, qi }, with tolerance
174           if qi < tol:
175               qi = 0.0
176           #end qi
177           # gamma
178           if qi < 1.0:
179               gamma = 1.0
180           else:
181               gamma = 2.0
182           #end gamma
183           # theta
184           if qi < 1e-3:
185               theta = 1e2
186           elif qi <= 1e-1:
187               theta = 1.5e2
188           elif qi <= 1.0:
189               theta = 2e2
190           else:
191               theta = 3e2
192           #end theta
193           #penalty
194           H = H + theta*qi**gamma
195       #end for
196
197       fitness =  f(x)
198
199       #line = str(x[0]) + "\t" + str(x[1]) + "\t"  + str(x[2]) + "\t"  + str(x[3]) + "\t" + str(x[4])
      + "\t" + str(x[5]) + "\t" + str(x[6]) + "\t" + str(fitness) + "\t" + str(H)
200       #print line
201
202       return fitness + H
203   #end F(x)
```

```python
# Project: Stiffened Plate Optimization
#
# Name: meshCreator.py
#
# Objective: Mesh Creation Routine
#
# Author: Guille Giraldo, 2011

def createMesh(A, B, S, t1, t2, t3, w, h, folder):

    segments = {'b1' : 2, 'b2' : 1, 'b3' : 1, 'b4' : 4, 'b5' : 1, 'b6' : 5, 'ext' : 20}

    #Begin of calculations
    LVy = B / segments['ext']

    Ns = int(A/S) + 1
    S0 = 0.5 * ( A % S )

    #Geometry
    Vy = geompy.MakeVectorDXDYDZ(0, LVy, 0)
    Vx = geompy.MakeVectorDXDYDZ(100, 0, 0)
    Vz = geompy.MakeVectorDXDYDZ(0, 0, 100)

    pO = geompy.MakeVertex(0, 0 ,0)
    pC = geompy.MakeVertex(0, B, 0)

    OXZ = geompy.MakePlane(pO, Vy, 2.*A)
    CXZ = geompy.MakePlane(pC, Vy, 2.*A)
    OXY = geompy.MakePlane(pO, Vz, A+B)

    b1 = geompy.MakeFaceObjHW(Vy, t1, S0 - 0.5*t2)
    b2 = geompy.MakeFaceObjHW(Vy, t3, 0.5*(w - t2))
    b3 = geompy.MakeFaceObjHW(Vy, t1, t2)
    b4 = geompy.MakeFaceObjHW(Vy, h - t3, t2)
    b5 = geompy.MakeFaceObjHW(Vy, t3, t2)
    b6 = geompy.MakeFaceObjHW(Vy, t1, S - t2)

    b1_t = geompy.MakeTranslation(b1, 0.5*(S0 - 0.5*t2),          0, 0.5*t1)
    b2_t = geompy.MakeTranslation(b2, S0 - 0.5*w + 0.25*(w - t2), 0, t1 + h - 0.5*t3)
    b3_t = geompy.MakeTranslation(b3, S0,                         0, 0.5*t1)
    b4_t = geompy.MakeTranslation(b4, S0,                         0, t1 + 0.5*(h - t3))
    b5_t = geompy.MakeTranslation(b5, S0,                         0, t1 + h - 0.5*t3)
    b6_t = geompy.MakeTranslation(b6, S0 + 0.5*S,                 0, 0.5*t1)

    sub_com_1 = geompy.MakeCompound([b2_t, b3_t, b4_t, b5_t])

    sub_com_2 = geompy.MakeMultiTranslation1D(sub_com_1, Vx, S, Ns)
    sub_com_3 = geompy.MakeMultiTranslation1D(b1_t, Vx, A - S0 + 0.5*t2, 2)
    sub_com_4 = geompy.MakeMultiTranslation1D(b6_t, Vx, S, Ns - 1)

    face0 = geompy.MakeCompound([sub_com_2, sub_com_3, sub_com_4])

    face = geompy.MakeGlueFaces(face0, 0.1)

    faces = geompy.SubShapeAllSorted(face, geompy.ShapeType["FACE"])

    geompy.addToStudy(OXZ, 'OXZ')
    geompy.addToStudy(CXZ, 'CXZ')
    geompy.addToStudy(OXY, 'OXY')
    geompy.addToStudy(face, 'Face')

    #Meshing
    mesh = smesh.Mesh(face)

    alg1D = mesh.Segment()
    alg1D.NumberOfSegments(1)

    mesh.Quadrangle()
    mesh.Hexahedron()

    alg1D_loc = []
    edges = []

    for i,f in enumerate(faces) :

```

```
76          if i == 0 or i == len(faces) - 1:
77              typ = 'b1'
78              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[1] )
79          elif i % 5. == 0:
80              typ = 'b6'
81              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[1] )
82          elif (i - 1) % 5. == 0:
83              typ = 'b5'
84              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[0] )
85          elif (i - 2) % 5. == 0:
86              typ = 'b3'
87              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[0] )
88          elif (i - 3) % 5. == 0:
89              typ = 'b4'
90              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[0] )
91          else:
92              typ = 'b2'
93              edges.append( geompy.SubShapeAllSorted( f, geompy.ShapeType["EDGE"] )[0] )
94          #end if
95
96          alg1D_loc.append( mesh.Segment( edges[i] ) )
97          alg1D_loc[i].NumberOfSegments( segments[typ] )
98          alg1D_loc[i].Propagation()
99      #end for
100
101     done = mesh.Compute()
102
103     if done :
104
105         pE = smesh.PointStruct(0., LVy, 0.)
106         vE = smesh.DirStruct(pE)
107         mesh.ExtrusionSweepObject( mesh, vE, segments['ext'])
108
109         #Groups
110         filter1 = smesh.GetFilter( smesh.NODE, smesh.FT_BelongToPlane, smesh.FT_EqualTo, OXZ)
111         filter2 = smesh.GetFilter( smesh.NODE, smesh.FT_BelongToPlane, smesh.FT_EqualTo, CXZ)
112         filter3 = smesh.GetFilter( smesh.FACE, smesh.FT_BelongToPlane, smesh.FT_EqualTo, OXY)
113
114         ids1 = mesh.GetIdsFromFilter(filter1)
115         ids2 = mesh.GetIdsFromFilter(filter2)
116         ids3 = mesh.GetIdsFromFilter(filter3)
117
118         gREST = mesh.MakeGroupByIds("REST", smesh.NODE, ids1)
119         gREST.Add(ids2)
120         gLOAD = mesh.MakeGroupByIds("LOAD", smesh.FACE, ids3)
121
122         mesh.ExportMED(folder + "mesh.mmed", 0)
123     #end if
124 #end createMesh
125
126 if __name__ == '__main__' :
127
128     import salome
129     import geompy
130     import smesh
131     import sys
132
133     A = float( sys.argv[1] )
134     B = float( sys.argv[2] )
135     d = float( sys.argv[3] )
136     t1 = float( sys.argv[4] )
137     t2 = float( sys.argv[5] )
138     t3 = float( sys.argv[6] )
139     w = float( sys.argv[7] )
140     h = float( sys.argv[8] )
141     folder = sys.argv[9]
142
143     createMesh(A, B, d, t1, t2, t3, w, h, folder)
144 #end if
```

```
1  # Project: Stiffened Plate Optimization
2  #
3  # Name: postprocess.py
4  #
5  # Objective: Solution Post-Prosessing
6  #
7  # Author: Guille Giraldo, 2011
8
9  def getMaxSigma(folderName):
10
11     f = open( folderName + "result.resu" )
12     fcontent = f.read()
13     f.close()
14
15     flines = fcontent.splitlines()
16     line = flines[37] #This is Max Von Misses Eq Stress
17     words = line.split(" ")
18
19     sigma = float( words[12] )
20
21     return sigma
22 #end getMaxSigma
23
24 def getMaxDZ(folderName):
25
26     f = open( folderName + "result.resu" )
27     fcontent = f.read()
28     f.close()
29
30     flines = fcontent.splitlines()
31     line = flines[29]
32     words = line.split(" ")
33
34     DZ = float( words[14] )
35
36     return DZ
37 #end getMaxDZ
```

```python
# Project: Stiffened Plate Optimization
#
# Name: setEnv.py
#
# Objective: create subfolders, copy study files and return study parameters based on config.xml file
#
# Author: Guille Giraldo, 2011

import os
from xml.dom.minidom import parse
from string import replace

def init():
    #Parse config file
    xdoc = parse('./Scripts/config.xml')

    xparams = xdoc.getElementsByTagName("param")

    params = {}

    for xpar in xparams:

        params[xpar.attributes['name'].value] = xpar.attributes['value'].value
    #end for

    Np = int(params['population'])

    xfilenames = xdoc.getElementsByTagName("file")

    sfiles = []

    for xfilename in xfilenames:

        sfiles.append( xfilename.attributes['value'].value )
    #end for

    #Create folders and copy study files in them
    studyDir = "/home/guillermo/OPPR3/"

    for i in range(Np):

        #Create Folder
        folderName = studyDir + "p" + str(i) + "/"
        os.system("mkdir " + folderName)

        #Copy files listed in config.xml
        for sfile in sfiles:
            os.system("cp " + studyDir + "/StudyFiles/" + sfile + " " + folderName + sfile )
        #end for

        #Now modify the export file just copied
        f = open(folderName + "run.export")
        fstr = f.read()
        f.close()

        fstr = replace(fstr, "WORKDIR/", folderName)

        f = open(folderName + "run.export", 'w')
        f.write(fstr)
        f.close()
    #end for

    return params
#end init()
```

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--
# Project: Stiffened Plate Optimization
#
# Name: config.xml
#
# Objective: store optimization parameters
#
# Author: Guille Giraldo, 2011
-->

<config>

    <!-- Problem Parameters -->
    <param name="dimension" value="5" />
    <param name="processcount" value="25" />
    <param name="population" value="25" />

    <!-- PSO Parameters -->
    <param name="cognition" value="2.0" />
    <param name="confidence" value="2.0" />
    <param name="inertia" value="0.8" />
    <param name="constriction" value="0.8" />
    <param name="iterations" value="150" />

    <!-- DE Parameters -->
    <param name="scalefactor" value="0.9" />
    <param name="crprobability" value="0.2" />
    <param name="generations" value="100" />

    <!-- Study Files -->
    <file name="export" value="run.export" />
    <file name="comm" value="command.comm" />

</config>
```

```
1  # Project: 2D Beam With Hole Optimization
2  #
3  # Name: __init__.py
4  #
5  # Objective: define "Scripts" folder as a package
6  #
7  # Author: Guille Giraldo, 2011
8
9  __all__ = ["meshCreator",  "postprocess", "problem", "initStudy"]
```

```
1
2  DEBUT();
3
4  MESH=LIRE_MAILLAGE(UNITE=20,
5                     FORMAT='MED',);
6
7  MESH=DEFI_GROUP(reuse =MESH,
8                  MAILLAGE=MESH,
9                  CREA_GROUP_MA=_F(NOM='ALL',
10                                  TOUT='OUI',),);
11
12 MODEL=AFFE_MODELE(MAILLAGE=MESH,
13                   AFFE=_F(TOUT='OUI',
14                           PHENOMENE='MECANIQUE',
15                           MODELISATION='3D',),);
16
17 STEEL=DEFI_MATERIAU(ELAS=_F(E=2.1e5,
18                             NU=0.3,),);
19
20 MAT=AFFE_MATERIAU(MAILLAGE=MESH,
21                   AFFE=_F(TOUT='OUI',
22                           MATER=STEEL,),);
23
24 BLOCK=AFFE_CHAR_MECA(MODELE=MODEL,
25                      DDL_IMPO=_F(GROUP_NO='REST',
26                                  DX=0,
27                                  DY=0,
28                                  DZ=0,),);
29
30 PRESS=AFFE_CHAR_MECA(MODELE=MODEL,
31                      PRES_REP=_F(GROUP_MA='LOAD',
32                                  PRES=29.4,),);
33
34 RESU=MECA_STATIQUE(MODELE=MODEL,
35                    CHAM_MATER=MAT,
36                    EXCIT=(_F(CHARGE=BLOCK,),
37                           _F(CHARGE=PRESS,),),);
38
39 RESU=CALC_ELEM(reuse =RESU,
40                RESULTAT=RESU,
41                OPTION='EQUI_ELNO_SIGM',);
42
43 RESU=CALC_NO(reuse =RESU,
44              RESULTAT=RESU,
45              OPTION='EQUI_NOEU_SIGM',);
46
47 IMPR_RESU(MODELE=MODEL,
48           RESU=_F(RESULTAT=RESU,
49                   NOM_CHAM=('DEPL','EQUI_NOEU_SIGM',),
50                   VALE_MAX='OUI',),);
51
52 FIN();
```

```
P profastk guillermo@server.der.edf.fr:New
P serveur localhost
P noeud localhost
P username guillermo
P mclient server.der.edf.fr
P uclient guillermo
P version STA10.2
P lang en
P debug nodebug
P mode interactif
P ncpus 1
P mpi_nbcpu 1
P mpi_nbnoeud 1
P classe
P depart
P distrib
P flashdir
P exectool
P nomjob New
P origine ASTK 1.8.1
P display localhost.der.edf.fr:11.0
A args
A memjeveux 64.0
P mem_aster 100.0
A tpmax 900
P memjob 524288
P tpsjob 15
P nbmaxnook 5
P cpresok RESNOOK
P facmtps 1
P corefilesize unlimited
F comm WORKDIR/command.comm D 1
F mmed WORKDIR/mesh.mmed D 20
F resu WORKDIR/result.resu R 8
P consbtc oui
P soumbtc oui
P actions make_etude
```