UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de la Construcción y de Proyectos de Ingeniería Civil

OPTIMIZACIÓN HEURÍSTICA ECONÓMICA DE MARCOS PREFABRICADOS DE HORMIGÓN ARMADO

Trabajo Fin de Máster

Máster Universitario en Ingeniería del Hormigón

AUTOR/A: Ruiz Vélez, Andrés

Tutor/a: Yepes Piqueras, Víctor

Cotutor/a: Alcalá González, Julián

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE LA CONSTRUCCIÓN Y DE PROYECTOS DE INGENIERÍA CIVIL

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL HORMIGÓN

TRABAJO FIN DE MÁSTER

CURSO ACADÉMICO 2021/2022

OPTIMIZACIÓN HEURÍSTICA ECONÓMICA DE MARCOS PREFABRICADOS DE HORMIGÓN ARMADO

Autor/a: Andrés Ruiz Vélez

Tutor/a: Víctor Yepes Piqueras

Cotutor/a: Julián Alcalá González

Valencia, Sept. de 2022

DEPARTAMENTO DE INGENIERÍA DE LA CONSTRUCCIÓN Y PROYECTOS DE INGENIERÍA CIVIL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

European Accreditation of Engineering Programmes EUR-ACE®

MUIH

Agradecimientos

Este trabajo da cierre a mi paso por el Máster Universitario en Ingeniería del Hormigón, etapa muy enriquecedora en la que he conseguido excelentes resultados académicos fomentados por el gran trabajo llevado a cabo por el profesorado. Es por esto que me gustaría mostrar mi más sincero agradecimiento a la unidad docente del máster, vuestra experiencia e inquietud se transmite y consigue despertar el interés de los estudiantes.

Víctor Yepes me ha mostrado lo que significa ser una persona proactiva, apasionada y dedicada a su trabajo. Muchas gracias por confiar en mi y promover mi continuidad en esta línea de investigación, gracias a tu labor he descubierto un campo que me apasiona y en el que deseo seguir formándome, consiguiendo objetivos comunes.

Julián Alcalá ha sido un apoyo constante, directo y eficaz. Tu cercanía me ha ayudado mucho a la hora de afrontar el trabajo, y tu experiencia ha sido clave en la resolución de muchos de los problemas propios del desarrollo de un proyecto de estas características.

También me gustaría agradecer la ayuda aportada por compañeros del grupo de investigación, especialmente a David Martínez, compañero que me acogió en mi entrada al ICITECH, y me ha guiado y ayudado en gran medida durante el desarrollo del trabajo.

Finalmente, me gustaría dar las gracias a las personas que, incluso cuando la distancia se hace notar, me han acompañado en mi día a día. Este trabajo va dedicado a mis padres, a mi hermano, y a María, sinceramente creo que una gran parte de mis éxitos se debe a vuestra compañía y apoyo incondicional. Gracias a vosotros he llegado hasta aquí con el claro objetivo de seguir mejorando.

Optimización heurística económica de marcos prefabricados de hormigón armado

Andrés Ruiz Vélez

Trabajo Final de Máster
Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil
Escuela Técnica Superior de Ingeniería de Caminos Canales y Puertos, Universidad Politécnica de Valencia

Resumen: Tradicionalmente, el diseño de estructuras se ha fundamentado en los conocimientos, habilidad y experiencia del ingeniero responsable, algo que da lugar a diseños estructuralmente seguros, pero que no hacen un uso óptimo de los materiales. Fundamentado en la necesidad de mejorar el proceso de diseño, el presente estudio busca diseños más sostenibles mediante la optimización de los recursos consumidos en una de las estructuras con un uso más extendido en la infraestructura de transporte. En este contexto, se lleva a cabo la optimización económica de un marco prefabricado de hormigón armado mediante tres técnicas heurísticas. El trabajo comprende el estudio del estado del arte, modelado matemático, y el desarrollo y posterior aplicación de software propio de cálculo, verificación y optimización estructural. Los resultados muestran diseños no tradicionales con secciones de canto reducido y bajo coste, que validan la aplicación del software desarrollado. La posibilidad de configurar diversas tipologías estructurales, en conjunto con la consideración simultánea de varias funciones objetivo hace del software desarrollado una potente herramienta de optimización multiobjetivo que permitirá avances en investigaciones posteriores.

Palabras Clave: Optimización, Marco, Heurística, Sostenibilidad, Hormigón.

Optimització heurística econòmica de marcos prefabricats de formigó armat

Andrés Ruiz Vélez

Trabajo Final de Máster
Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil
Escuela Técnica Superior de Ingeniería de Caminos Canales y Puertos, Universidad Politécnica de Valencia

Resum: Tradicionalment, el disseny déstructures s'ha fonamentat en els coneixements, habilitat i experiència de lénginyer responsable, alguna cosa que dona lloc a dissenys estructuralment segurs, però que no fan un ús òptim dels materials. Fonamentat en la necessitat de millorar el procés de disseny, el present estudi busca dissenys més sostenibles mitjançant lóptimització dels recursos consumits en una de les estructures amb un ús més estés en la infraestructura de transport. En aquest context, es duu a terme lóptimització econòmica dún marc prefabricat de formigó armat mitjançant tres tècniques heurístiques. El treball comprén léstudi de léstat de lárt, modelatge matemàtic, i el desenvolupament i posterior aplicació de programari propi de càlcul, verificació i optimització estructural. Els resultats mostren dissenys no tradicionals amb seccions de cant reduït i baix cost, que validen láplicació del programari desenvolupat. La possibilitat de configurar diverses tipologies estructurals, en conjunt amb la consideració simultània de diverses funcions objectiu fa del programari desenvolupat una potent eina dóptimització multiobjectiu que permetrà avanços en investigacions posteriors.

Paraules Clau: Optimització, Marco, Heurísticas, Sostenibilitat, Formigó.

Economic heuristic optimization of precast reinforced concrete frames

Andrés Ruiz Vélez

Trabajo Final de Máster
Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil
Escuela Técnica Superior de Ingeniería de Caminos Canales y Puertos, Universidad Politécnica de Valencia

Abstract: Traditionally, structural design has been conditioned by the responsible engineer's knowledge, technical skills and previous experiences. This results in structurally safe designs which do not optimize resource consumption. Based on the need to improve the design process, the present study seeks more sustainable designs by optimizing the resources consumed during the fabrication of one of the common structures in the transportation infrastructure. In this context, the economic optimization of precast reinforced concrete frames is carried out using three heuristic techniques. The work comprises the study of the state of the art, mathematical modelling, and development and subsequent application of proprietary structural calculation, verification and optimization software. The results show non-traditional designs with reduced depth sections and low cost, validating the application of the developed software. The possibility of configuring different structural typologies, together with the simultaneous consideration of several objective functions, show that the developed software is a powerful multiobjective optimization tool that will allow advances in further research.

Key Words: Optimization, Frame, Heuristic, Sustainability, Concrete.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto de la investigación

Los recursos naturales no son ilimitados, es bien sabido que un uso abusivo de estos tiene consecuencias directas, generando un impacto negativo sobre el medio que nos rodea [1, 2]. En la actualidad la construcción se sitúa como una de las actividades industriales más contaminante a nivel mundial, algo directamente relacionado con la ingente cantidad de recursos materiales requeridos para un desarrollo adecuado de su actividad [3–5].

La evolución de un país depende, de forma directa, de que su infraestructura sea capaz de sustentar y acompañar ese avance. Consecuentemente, la construcción, y posterior mantenimiento, de las infraestructuras que nos rodean son acciones necesarias para un correcto desarrollo de las actividades socio económicas del ser humano. Sin embargo, la naturaleza imprescindible de estas actividades no hace que el uso irresponsable de los ya mencionados recursos naturales sea tolerable [6, 7].

Con origen etimológico latín *ingenium*, un ingeniero es una persona que hace uso del ingenio para resolver problemas [8, 9]. La magnitud de los problemas medioambientales mencionados anteriormente hace evidente que, como ingenieros, debemos hacer uso de las herramientas a nuestro alcance para buscar una solución que permita solventar, o al menos reducir, estos problemas.

Una estructura puede considerarse sostenible cuando, a lo largo de todo su ciclo de vida, mantiene un equilibrio adecuado entre su coste económico, su impacto

medioambiental y el impacto social que genera [10, 11]. Este trabajo es, pues, una búsqueda de respuestas, fundamentadas en el conocimiento técnico y la experiencia, que permitan el avance hacia una industria de la construcción menos nociva y más sostenible.

1.2. Alcance y objetivos de la investigación

El presente trabajo final de máster busca profundizar y mejorar los conocimientos actuales sobre las técnicas de optimización de estructuras de hormigón armado. Una de las estructuras comunes en la red nacional de carreteras son los marcos prefabricados. Esta tipología estructural se usa habitualmente como solución al cruce de caminos así como obras de drenaje.

En vista de lo presentado en el apartado anterior, el estudio busca establecer relaciones, fundamentadas en resultados representativos, entre la aplicación de metodologías innovadoras en el proceso de diseño de estructuras de hormigón armado, y la mejora en la utilización de materiales en su construcción. Además, se pretende diferenciar ciertas características que permiten a dichos diseños, resultados del proceso de optimización, ser estructuralmente seguras manteniendo presupuestos reducidos.

En este contexto, el objetivo general es la optimización económica de un marco de paso de carretera prefabricado mediante el uso de técnicas metaheurísticas, en concreto la aplicación de tres algoritmos de optimización heurística. Esto comprende el uso de un algoritmo genético de Recocido Simulado o *Simulated Annealing* (SA), Aceptación por Umbrales o *Threshold Accepting* (TA), y finalmente Algoritmo del Solterón o *Old Bachelor Acceptance* (OBA).

El hecho de que la optimización sea económica está fundamentado en que el coste de la estructura es el método más directo de cuantificar la cantidad de recursos consumidos [12]. La restricción de la cantidad de materiales utilizados, y consecuente disminución del impacto ambiental, es inherente la reducción del coste económico del marco.

Para lograr la consecución del objetivo general del trabajo, se establecieron los siguientes objetivos específicos:

1. Análisis del proceso de diseño estructural y su optimización mediante la apli-

cación de metodologías metaheurísticas.

2. Análisis y selección de técnicas heurísticas a aplicar en la optimización económica del marco prefabricado.

3. Desarrollo de software propio de verificación y optimización estructural que permita la optimización económica del marco prefabricado.

4. Obtención y validación de resultados, para un posterior estudio y análisis de los mismos.

1.3. Estructura del trabajo

El trabajo se divide en seis capítulos diferenciados, el presente Capítulo I contextualiza y define el objetivo de la investigación.

Tras esto, en el Capítulo II se presenta una imagen clara del proceso de diseño estructural y el papel de la optimización en el mismo. Además, se desarrolla la clasificación y características de las principales técnicas empleadas en la optimización de estructuras hasta el momento. El capítulo se cierra con un breve estado del arte donde se hace mención de los trabajos más relevantes en el ámbito de la optimización de estructuras de hormigón armado, así como aquellos que han llevado a cabo la aplicación de una o varias técnicas heurísticas para su resolución.

El Capítulo III es dedicado a formular el problema de optimización, esto comprende la definición de parámetros y variables consideradas, así como los rangos de valores permitidos para estos. Establecido esto, se detallan las comprobaciones a realizar, se define la función objetivo del problema y se describen de forma detallada los problemas estudiados.

En el Capítulo IV se presenta el software desarrollado para el cálculo estructural del marco, así como la verificación de los estados límite correspondientes. Además, se detalla la aplicación de los algoritmos considerados para la optimización del marco prefabricado. Esto engloba la completa definición, y adecuación a las características del problema, de los parámetros de cada uno de los algoritmos considerados, así como el proceso seguido para la obtención de los resultados de la optimización.

El Capítulo V presenta los resultados de la optimización, inicialmente se justifica la metodología seguida en lo referente al número de ejecuciones llevado a cabo en la resolución de cada problema planteado. Tras esto, la parte final del capítulo consiste en el análisis y comentario de los resultados obtenidos.

Finalmente el Capítulo VI es dedicado a establecer las conclusiones generales del estudio, así como una serie de conclusiones específicas derivadas de las anteriores. Como cierre del trabajo final de máster, se establecen futuras líneas de investigación derivadas del estudio que permitirán completar y ampliar los resultados.

De esta forma, queda establecido el contenido general comprendido en cada uno de los diferentes capítulos que componen el presente trabajo.

Capítulo 2

Optimización de estructuras de hormigón y métodos heurísticos

2.1. Optimización en el diseño de estructuras

2.1.1. Proceso de diseño estructural

Tradicionalmente, el diseño de estructuras se ha fundamentado en los conocimientos técnicos, habilidad, y experiencia previa del ingeniero responsable [13]. Mediante el método tradicional, este diseña una estructura inicial que ha de verificar los estados límites pertinentes, de no ser así, los materiales o su cantidad son modificados. Esto se repite en un proceso de prueba y error que, si bien nos permite obtener estructuras seguras que verifican los requerimientos establecidos en norma, no son óptimas en el uso de materiales.

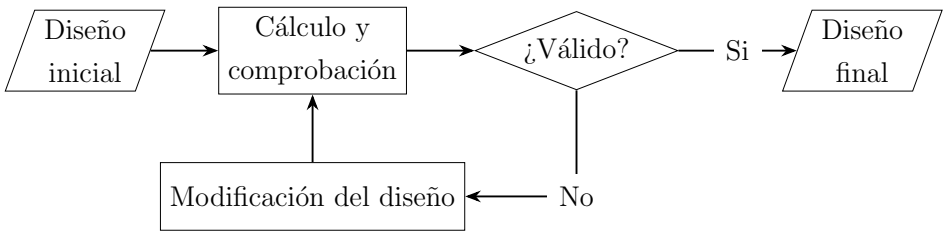


Figura 2.1: Diagrama de flujo del proceso de diseño estructural tradicional (Elaboración propia basado en Martí Albiñana [14]).

Como ingenieros, es nuestra responsabilidad no sólo crear diseños que se adecúen a los requerimientos estructurales, sino que también tengan un impacto positivo a nivel social y el mínimo impacto medioambiental posible [15–17].

Sin embargo, el hecho de que la experiencia previa de los ingenieros sea el pilar fundamental de la metodología tradicional del diseño de estructuras no es algo fortuito. El diseño de una estructura es un problema muy complejo, con una gran incertidumbre y un número elevado de variables a tener en cuenta, por lo que la sistematización del proceso de obtención de soluciones válidas es complicada, y poco viable sin el uso de herramientas computacionales avanzadas. Esto da lugar a la predisposición intrínseca de crear estructuras con diseños similares a otras ya existentes.

En este contexto, el continuo desarrollo tecnológico y consecuente mejora de los recursos computacionales a disposición del ingeniero promedio, permite la aparición y desarrollo de nuevas técnicas de diseño que, dan la importancia necesaria a los diversos factores mencionados con anterioridad [18]. Estas metodologías consistentes en la aplicación de técnicas metaheurísticas al diseño de estructuras dan lugar a la creación de diseños no tradicionales que, cumpliendo los requisitos estructurales, consiguen minimizar o maximizar una serie de factores establecidos anteriormente tales como el coste económico, las emisiones de CO_2 asociadas, o la energía consumida en la construcción de la estructura diseñada [12–14, 19, 20].

Estas nuevas técnicas presentan una muy buena evolución en las últimas décadas, conformando un campo de investigación en continuo desarrollo. En definitiva, la optimización estructural nace motivada por la búsqueda de soluciones que permitan reducir el impacto negativo que la industria de la construcción tiene sobre nuestro planeta, tarea facilitada por el rápido desarrollo tecnológico, que actúa como un potente catalizador del proceso.

A diferencia de la metodología tradicional, el diseño de estructuras con técnicas de optimización permite, en cierta medida, la automatización del proceso. Bajo el control de un ingeniero con conocimientos técnicos adecuados, un software de diseño óptimo permite obtener resultados considerablemente superiores. Sin embargo, aunque este tipo de software conforma una herramienta muy poderosa para el diseño estructural, el usuario ha de entender de forma íntegra el trasfondo técnico para, de esta forma, poder compensar aquellos aspectos fuera del alcance del software.

Citando el manual del software *SAP2000*, uno de los productos de uso extendido en el ámbito de la ingeniería estructural, el uso de un software de optimización ha de ir acompañado de la siguiente advertencia, "*Los resultados producidos por el software deben ser validados por un ingeniero de diseño experimentado. Este los ha de verificar de forma independiente aportando respaldo profesional y siendo responsable del uso de la información que el programa proporciona*" [21].

Partiendo del hecho de que existen múltiples soluciones para un mismo problema de diseño, es relevante introducir el concepto de estructura óptima. En el contexto de la optimización estructural, multitud de autores entienden como estructura óptima aquella que, para una función predefinida denominada función objetivo, obtiene el valor más adecuado [22]. Es común identificar la función objetivo de un problema de diseño con el coste final de la estructura, esto se debe en gran medida a que, a nivel industrial, la optimización tiene un gran potencial a la hora de reducir costes y mejorar los márgenes de beneficio de las empresas [12, 23, 24].

Teniendo en cuenta que el trasfondo del coste final de una estructura viene determinado, principalmente, el tipo y cantidad de materiales que se usan en su construcción, en conjunto con la actual búsqueda de construcciones sostenibles y de bajo impacto medioambiental, da lugar a que la consideración simultánea de varias funciones objetivo que valoren tanto el aspecto económico como los mencionados anteriormente, sea de especial interés.

2.1.2. Modelización matemática de un problema de optimización

A la hora de abordar un problema de optimización, hemos de entender que este, a su vez, conlleva la resolución de un proceso de modelización. La realidad es muy difícil de modelar, los problemas de diseño tienen una gran complejidad y en ocasiones es difícil discernir qué partes de esta son y no son relevantes para nuestro problema. Por esto, hemos de sopesar bien la fase de creación del modelo pues, si este es demasiado simple, no será suficientemente representativo, sin embargo, un modelo demasiado complejo puede llegar a impedir que distingamos relaciones básicas existentes en nuestro problema. Es por esto que la modelización está fundamentada en identificar y determinar de forma adecuada los límites del problema que tratamos de resolver.

Un modelo matemático de optimización adecuado ha de contar siempre con una serie de componentes diferenciados y relacionados entre sí.

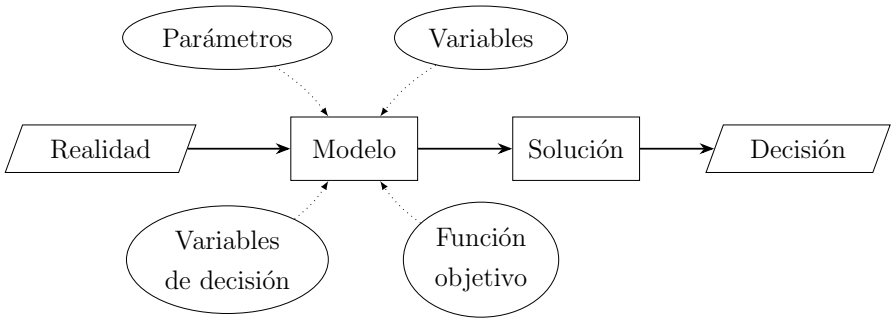


Figura 2.2: Diagrama de flujo del modelo matemático de un problema de optimización con sus diferentes componentes (Elaboración propia).

- El primero de estos componentes son los **parámetros**, se trata de magnitudes de valor constante durante el proceso de optimización. Es información conocida cuyo valor no es objeto de optimización. Algunos de los parámetros del presente estudio son, por ejemplo, el módulo de elasticidad de los materiales o las dimensiones del marco.

- El segundo componente del problema son las **variables de diseño**, estas son una serie de datos objeto de optimización que, en conjunto con los parámetros, permiten la definición completa de la estructura. El conjunto formado por los diferentes valores que las variables pueden adoptar a lo largo del proceso de optimización es denominado **espacio de soluciones** de nuestro problema. En el caso del marco algunas de las variables objeto de optimización son el diámetro y número de barras de la armadura, o el ancho de losas y hastiales.

- Tanto parámetros como variables han de cumplir una serie de **restricciones**, estas son condiciones responsables de asegurar que el diseño óptimo sea representativo de la realidad, cumpla los requerimientos estructurales establecidos, y verifique los estados límite pertinentes. Entre otras, las restricciones particulares del problema que se presentan son las comprobaciones de estados límite últimos, o las comprobaciones geométricas de cada sección.

- La definición de parámetros y restricciones, así como el conocimiento del valor de las variables, permite evaluar la **función objetivo**. Esta, como el nombre indica, determina el objetivo global de la optimización, pudiendo haber una o varias en caso de que se trate de optimización multiobjetivo. En el presente estudio la función objetivo representa el coste económico del marco, y es calculada como el sumatorio de la medición de cada uno de los materiales multiplicado por su coste unitario.

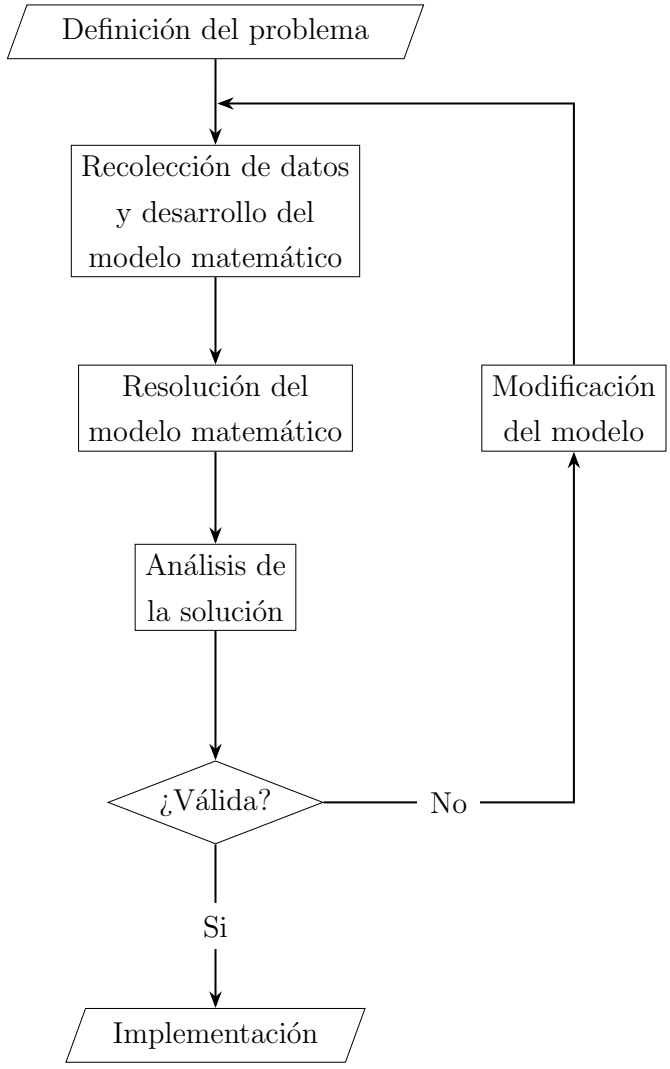


Figura 2.3: Proceso de modelización matemática de un problema de optimización (Elaboración propia basado en Martí Albiñana [14]).

En este contexto, la metodología a aplicar en la creación de un modelo matemático para la optimización de un diseño estructural parte de identificar las diferentes componentes del modelo. Estos son los parámetros y variables que se relacionan entre sí mediante expresiones matemáticas. De esta forma, una vez identificados, el siguiente paso natural es determinar la estructura matemática del modelo, es decir, entender y definir de forma completa las expresiones matemáticas que relacionan todas las componentes.

Esto proporciona una imagen general del funcionamiento interno del modelo, y permite dar paso al establecimiento de un principio de elección. Este puede consistir en la relativamente simple verificación de una serie de comprobaciones, o como en el caso que nos concierne, la evaluación de una función objetivo.

Una vez establecido el principio de elección, el modelo matemático se ha definido de forma completa, pudiendo dar paso al proceso de optimización. En función de las características propias de cada técnica, se generan un conjunto de soluciones alternativas, que permiten mediante un análisis de sensibilidad, valorar el conjunto para así aceptarlas o rechazarlas. Esto consiste en, bajo la consideración de un escenario compuesto por una serie de suposiciones, seguir cursos de acción establecidos que respalden la toma de decisiones.

Los modelos matemáticos de optimización pueden clasificarse en función de las características del grado de conocimiento del problema, variables, restricciones y función objetivo, pudiendo distinguirse principalmente los siguientes [18].

- En función de la **certeza** con la que se conocen los datos del problema el modelo puede clasificarse como:
  - Determinístico, donde todos los datos se conocen con certeza.
  - Estocástico, modelo en el que no todos los datos se conocen con certeza.
- En función de las **restricciones** del modelo se diferencia entre:
  - No restringidos.
  - Restringido, donde, a su vez, se pueden considerar dos clases.
    - Restricciones no lineales.

        - Restricciones lineales, modelo en el que todas quedan expresadas como suma o diferencia de constantes conocidas multiplicadas por una variable.

- En función de la **función objetivo**, se puede distinguir de nuevo entre:
    - Función objetivo no lineal.
    - Función objetivo lineal.
- Finalmente, en función de las **variables** del modelo, podemos distinguir entre modelos en los que:
    - Variables enteras, es decir, números naturales.
    - Variables continuas.

2.2. Técnicas heurísticas más relevantes

La optimización estructural consiste en la resolución de un problema de optimización combinatoria. En este, las variables de decisión son enteras, dando lugar a un espacio de soluciones conformado por números naturales. El objetivo es encontrar un máximo o mínimo de una determinada función sobre un conjunto de soluciones que, si bien es finito, es de tamaño considerablemente grande. De esta forma, "*la optimización combinatoria contiene los dos elementos que hacen atractivo un problema a los matemáticos: un planteamiento sencillo y dificultad de resolución*" (Garfinkel, 1985 [25]).

En función de la respuesta proporcionada, podemos clasificar los algoritmos de resolución en dos grandes grupos.

- **Exactos**, algoritmos capaces de alcanzar la solución óptima del problema. Uno de los principales algoritmos de resolución exactos son la programación lineal, también denominada método *simplex*, un caso particular de programación matemática donde todas las funciones del modelo son lineales [26]. Se trata de un método aplicable a multitud de procesos que han de ser optimizados, sin embargo su planteamiento altamente restrictivo limita las aplicaciones en el mundo real.

- **Heurísticos**, también denominados métodos aproximados, son algoritmos sencillos que consiguen encontrar soluciones de alta calidad para un problema concreto, pero sin la certeza de que estas sean la óptima. El principal problema de estos algoritmos es el gran esfuerzo computacional derivado de su resolución [27]. Además de los algoritmos heurísticos, existen las técnicas **metaheurísticas**, métodos generales que emulan estrategias eficientes de la naturaleza y que, guiando a las técnicas heurísticas, son de carácter general, pudiendo ser aplicados a un amplio abanico de problemas [18].

A continuación se detallan los tres grupos que conforman las principales técnicas heurísticas y metaheurísticas aplicadas al problema de optimización estructural, estos quedan representados en la figura 2.4. El hecho de que sean esta tipología la que se usa de forma común, y no los algoritmos exactos, está fundamentado en el que estas proporcionan una mayor flexibilidad, permitiendo así la imposición de condiciones difíciles de modelar.

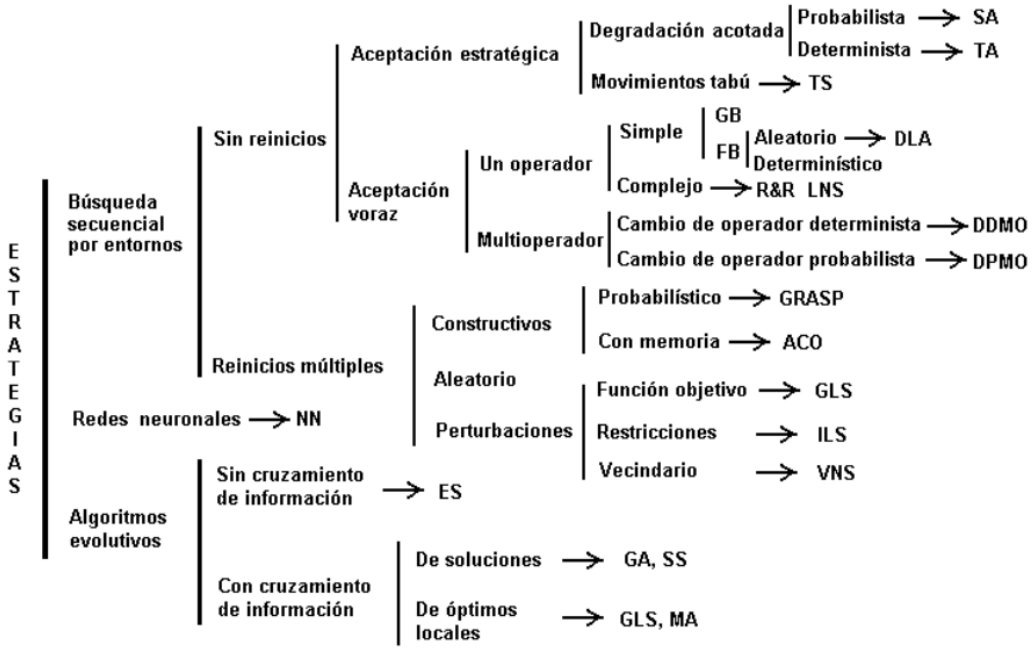


Figura 2.4: Taxonomía de estrategias empleadas en la resolución aproximada de problemas de optimización combinatoria sobre la base de soluciones iniciales (Yepes, 2002 [18])

2.2.1. Métodos de búsqueda secuencial por entornos

Los métodos de búsqueda secuencial por entornos parten de una solución inicial S_0 aleatoria, a la que se aplica una ligera modificación en una o varias variables, para así obtener una nueva solución S_1. Esta nueva solución es evaluada y aceptada o rechazada en función del criterio de aceptación preestablecido. En caso de que S_1 sea aceptada, esta se establece como nueva solución inicial, para luego repetir el proceso desde el inicio. La modificación de las variables que se lleva a cabo para pasar de una solución a la siguiente es denominada **movimiento**. Este ha de modificar las variables en una magnitud suficientemente pequeña como para evitar la pérdida de las cualidades positivas de la solución inicial. El conjunto de nuevas soluciones a las que se puede llegar mediante un movimiento es el denominado **entorno** de la solución.

2.2.1.1. Búsqueda por gradiente

El primero de los algoritmos de búsqueda secuencial por entornos que se detallan en este capítulo es el algoritmo de búsqueda por gradiente, *Gradient Descent* (GD). Este algoritmo recorre el espacio de soluciones del problema encontrando soluciones del entorno que consiguen mejorar la actual. En el caso de que la función objetivo sea el coste de la estructura, este método circula por el espacio de soluciones en busca de gradientes negativos que indiquen un coste inferior al actual.

Existen dos posibilidades a la hora de definir el movimiento en este tipo de algoritmos. La primera de ellas, denominada *First Best* (FB), acepta la primera solución que mejora la actual sin necesidad de evaluar el resto de soluciones del entorno. La segunda posibilidad, denominada *Global Best* (GB), a diferencia de la anterior, evalúa todas las posibles soluciones dentro del entorno y selecciona aquella que conlleva una mayor mejora.

El principal problema asociado a este algoritmo es la posibilidad de que finalice en un óptimo local, algo que proporcionaría una solución de baja calidad debido a que el algoritmo ha tenido una convergencia prematura [28].

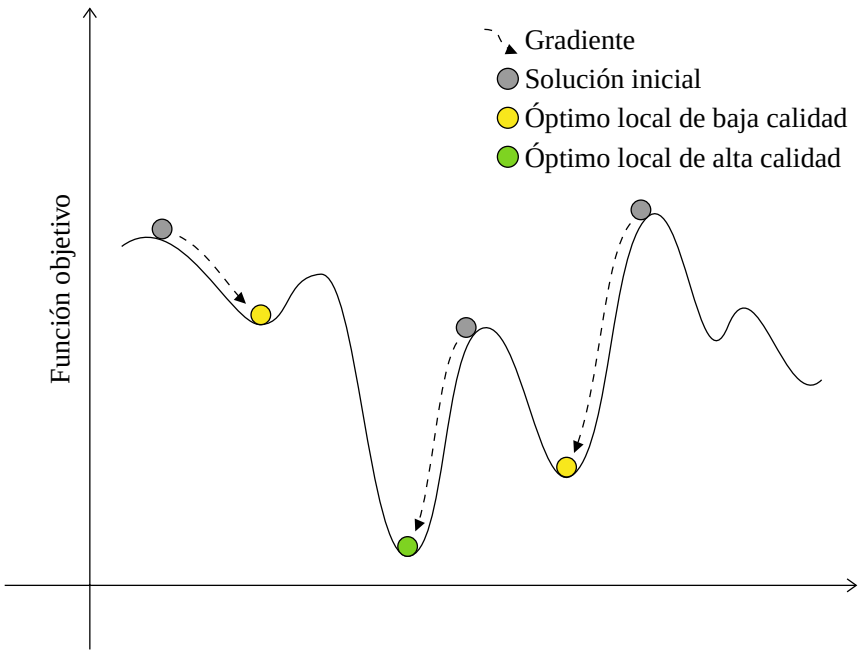


Figura 2.5: Funcionamiento del algoritmo de búsqueda por gradiente (Elaboración propia).

Existen estrategias que buscan salvar esta limitación haciendo que, en ciertas circunstancias, el algoritmo acepte soluciones de menor calidad que la actual. A continuación se detallan los algoritmos de Recocido Simulado o *Simulated Annealing* (SA) y Aceptación por Umbrales o *Threshold Accepting* (TA), ambos de los cuales permiten una cierta degradación de la solución con el objetivo de evitar el problema mencionado anteriormente [12, 18].

2.2.1.2. Recocido simulado

Esta estrategia, propuesta inicialmente por Kirkpatrick, Gelatt y Vecchi [29] en 1983 también fue formulada independientemente por Cerny en 1985 [30], simula el proceso de recocido usado para modificar las características mecánicas de los metales.

En el recocido, el metal es sometido a una temperatura inicial muy elevada, para luego someterlo a un enfriamiento lento y controlado. Durante esta segunda fase de enfriamiento, tiene lugar el proceso de cristalización del material, en este su micro-estuctura interna sufre modificaciones asociadas a la creación de cristales derivada de la búsqueda de estados energéticos de menor nivel. Los estados alcanzados, y en

consecuencia, las características finales del material serán función de los parámetros propios del proceso tales como la temperatura inicial o velocidad de enfriamiento.

De esta forma, el algoritmo equipara la función objetivo con el nivel energético asociado a cada uno de los estados alcanzados por el material durante el proceso de cristalización, tratando de avanzar hacia soluciones más estables cuyo nivel de energía es menor, ya que esto conlleva una solución de mejor calidad.

El proceso seguido por el SA consiste, en primer lugar, en la generación de una solución aleatoria S_0 y la definición de la temperatura inicial del problema T_0. Tras esto, durante el periodo en el que no se cumpla el criterio de parada establecido, el algoritmo continúa generando nuevas soluciones. En caso de que la nueva solución S_1 mejore la anterior, esta es aceptada de forma directa. De no ser así, el criterio de aceptación se basa en la comparación de un valor aleatorio en el intervalo (0,1) con el valor resultado de la expresión de Boltzmann:

$$e^{-(\Delta E/T)} \tag{2.1}$$

Donde ΔE es la diferencia de energía entre dos niveles, equivalente a la diferencia de coste entre dos soluciones y T es la temperatura del problema en el momento de cálculo.

$$\Delta E = |f(S_1) - f(S_0)| \tag{2.2}$$

La nueva solución es aceptada si el número aleatorio generado es inferior al valor de la función de Boltzmann.

Esto se repite, a una misma temperatura, durante un número finito de iteraciones que se denomina Cadena de Markov. Al finalizar la cadena, el parámetro T se reduce a razón del coeficiente de enfriamiento k. Tras el establecimiento de una nueva temperatura, inferior a la anterior, se inicia una nueva cadena. Este descenso de la temperatura conlleva una menor tasa de aceptación de nuevas las soluciones S_1 que no mejoran la solución actual S_0.

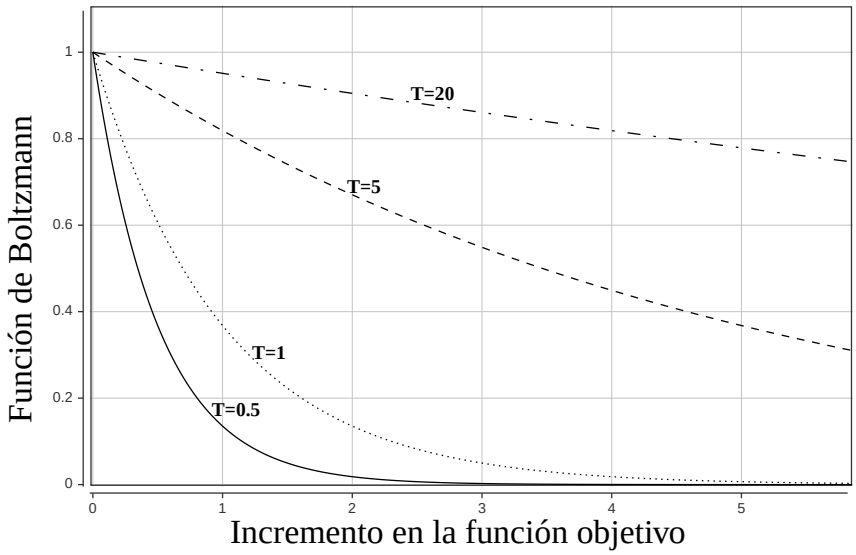


Figura 2.6: Probabilidad de aceptación de una solución inferior a la actual en función del incremento de la función objetivo en un proceso de recocido simulado (Elaboración propia).

El final del proceso puede establecerse mediante tres criterios de parada diferenciados. El primero de ellos establece el máximo número de cadenas sin mejora que pueden tener lugar, el segundo define la temperatura mínima que se puede alcanzar, siendo esta habitualmente un porcentaje bajo de la temperatura inicial. El tercer y último criterio de parada, es común a cualquier algoritmo y consiste en determinar un tiempo de computación máximo.

Este algoritmo consigue, gracias al criterio de aceptación de soluciones que empeoran la actual, evitar la convergencia prematura y consecuente aceptación de soluciones de baja calidad que conforman óptimos relativos. De acuerdo con lo mencionado anteriormente, el esquema general de un algoritmo SA es el presentado en la figura 2.7.

Al igual que el proceso físico de recocido, la correcta definición de los parámetros del proceso va a dictaminar la calidad de los resultados. Un metal sometido a un enfriamiento demasiado rápido no puede llevar a cabo los procesos de cristalización que le permiten alcanzar configuraciones muy estables de bajo nivel energético. De igual forma, el uso de un algoritmo SA para resolver un problema de optimización requiere una correcta definición de sus parámetros que lo definen, algo que comprende el correcto establecimiento de criterios que determine la temperatura inicial, coeficiente

de enfriamiento, y parada. Es igualmente importante almacenar todas las soluciones generadas pues puede que como consecuencia de la aceptación de soluciones de menor calidad, la solución final no sea el óptimo de nuestro problema.

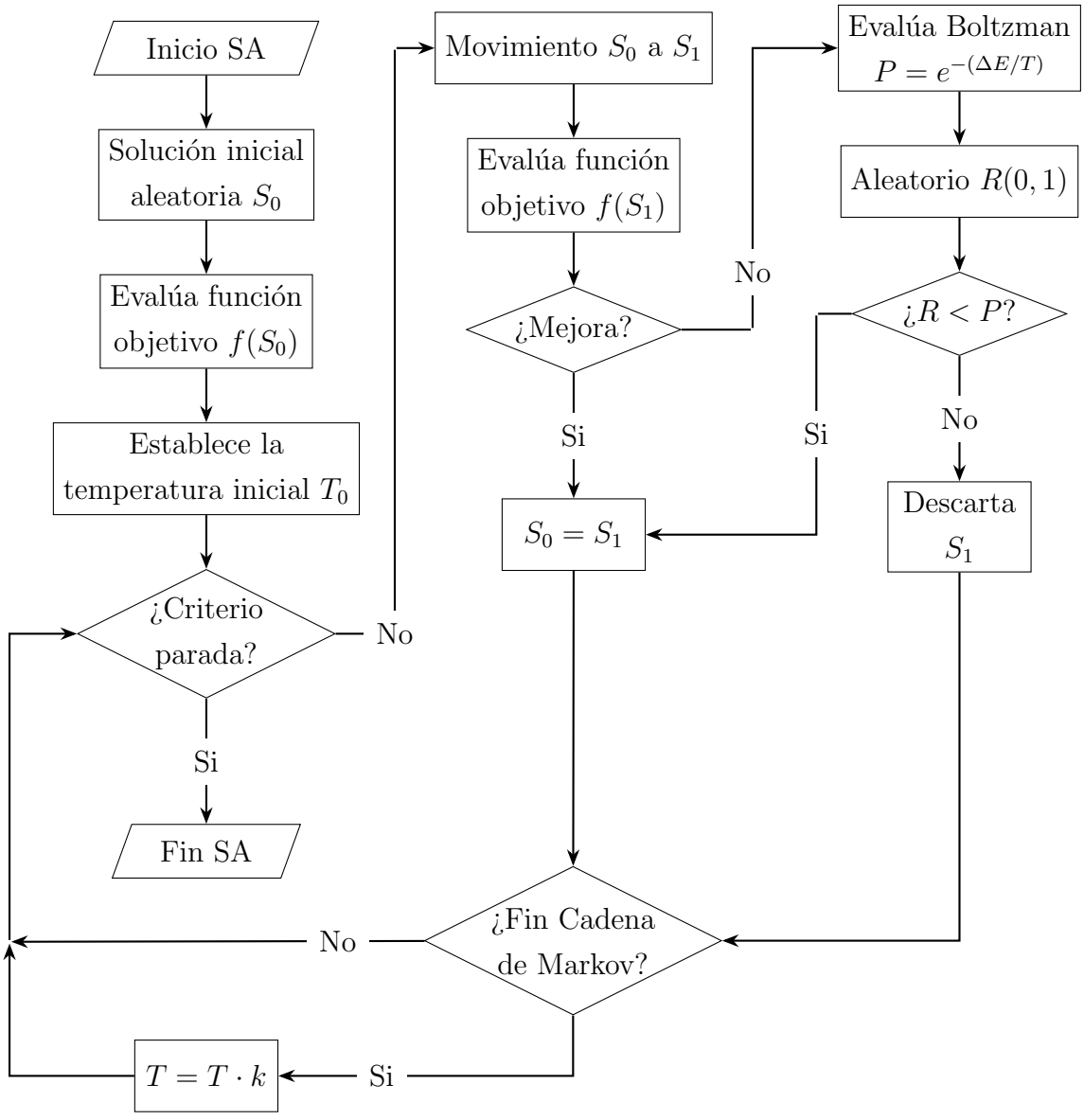


Figura 2.7: Diagrama de flujo del funcionamiento de un algoritmo de recocido simulado (Elaboración propia).

2.2.1.3. Aceptación por umbrales

Años después de la aparición del SA, motivado por la eficacia demostrada, surgen nuevas variantes que modifican el criterio de aceptación del algortmo. El algoritmo *Threshold Accepting*, TA, es desarrollado en el año 1990 de forma independiente tanto por Dueck y Scheuer [31], como por Moscato y Fontanari [32]. La principal diferencia del TA respecto al SA es el hecho de que el criterio de aceptación del TA es determinístico, mientras que el del SA es probabilístico.

Este nuevo criterio determina que cualquier nueva solución puede ser válida, siempre que se encuentre dentro de un determinado umbral U, es decir S_1 sustituye a S_0 si la mejora o si, en caso de que la empeore, se cumple la expresión:

$$f(S_1) - f(S_0) < U \tag{2.3}$$

Esto se repite un número finito de iteraciones tras el cual se disminuye U a razón de un parámetro de control. Al igual que en el caso del recocido simulado, debido a la aceptación de soluciones de menor nivel, es necesario almacenar todas las soluciones generadas ya que la solución final podría no ser la óptima.

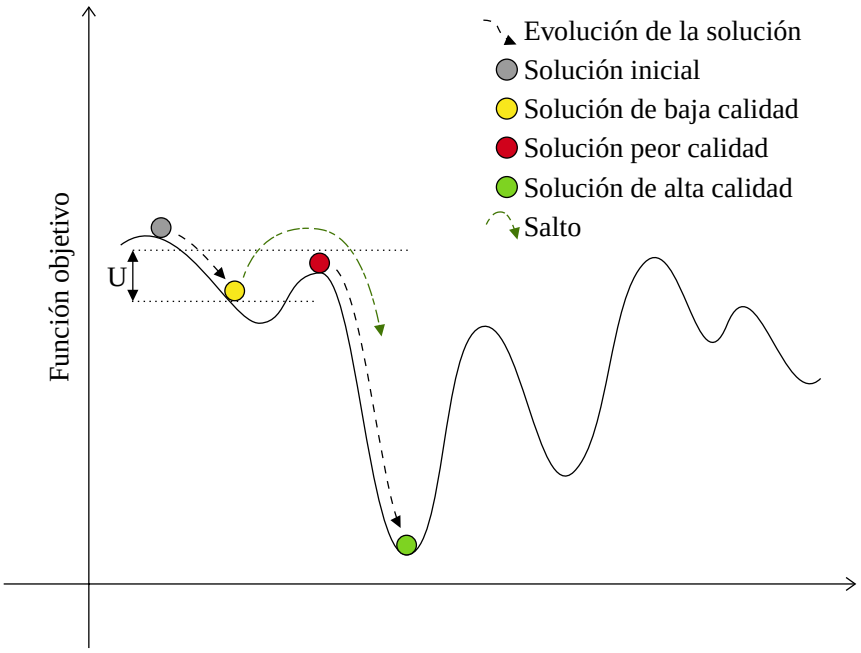


Figura 2.8: Funcionamiento del algoritmo de aceptación por entornos (Elaboración propia).

En conjunto con el SA y TA, existen otros algoritmos de búsqueda por entornos que modifican alguna de las características de estos consiguiendo buenos resultados [33–36].

2.2.1.4. Algoritmo del solterón

Los algoritmos SA y TA descritos anteriormente comparten el hecho de que la calidad de su solución va a depender, en gran medida, del equilibrio que se pueda alcanzar entre la fase inicial de diversificación y la fase de intensificación final. En estos algoritmos la diferenciación entre esas fases viene determinada por la correcta calibración del parámetro de control, temperatura para el SA, y umbral para el TA.

En este contexto, el algoritmo del solterón o *Old Bachelor Acceptance* (OBA) tiene un principio consistente en una variación del uso de umbrales propuesto por el TA, que le permite adaptar las fases de intensificación y diversificación en función de la calidad de las soluciones que el algoritmo esté obteniendo en cada instante. Propuesto por primera vez en 1995 por Hu et al. [37], se trata de un algoritmo que permite una exploración amplia del espacio de soluciones, pudiendo llegar a obtener buenos resultados en la resolución de problemas de optimización.

De esta forma, el funcionamiento del algoritmo parte de la generación aleatoria de una solución inicial S_0. Tras esto, se genera una nueva solución S_1 y se compara la función objetivo de ambas. En caso de que la nueva solución mejore la actual, es aceptada de forma directa, por otra parte, si la nueva solución es de peor calidad, sólo es aceptada si la diferencia entre ambas queda dentro de un umbral determinado. Es precisamente el umbral del OBA lo que le permite variar entre fases de exploración e intensificación. Se parte de un umbral nulo ($U = 0$) para, tras esto, variar su magnitud en función de las soluciones obtenidas.

En caso de que la nueva solución sea de mejor calidad, el algoritmo entiende que se encuentra en una región del espacio de soluciones donde puede existir un óptimo de buena calidad. Por ello, es de interés empezar una fase de intensificación. Para conseguir esto, el umbral actual es reducido a razón de un factor preestablecido denominado factor de variación ($\Delta_{(-)}$), cada vez que una nueva solución es mejor que la anterior.

Por otra parte, en caso de que la nueva solución sea de peor calidad, el OBA

considera que esa región del espacio de soluciones ha de ser explorada hasta encontrar un posible óptimo, para ello, cada vez que la nueva solución es peor que la anterior, el umbral actual aumenta a razón del ya mencionado factor de variación, en este caso ($\Delta_{(+)}$). Este funcionamiento puede observarse en el diagrama de flujo representado en la figura 2.9.

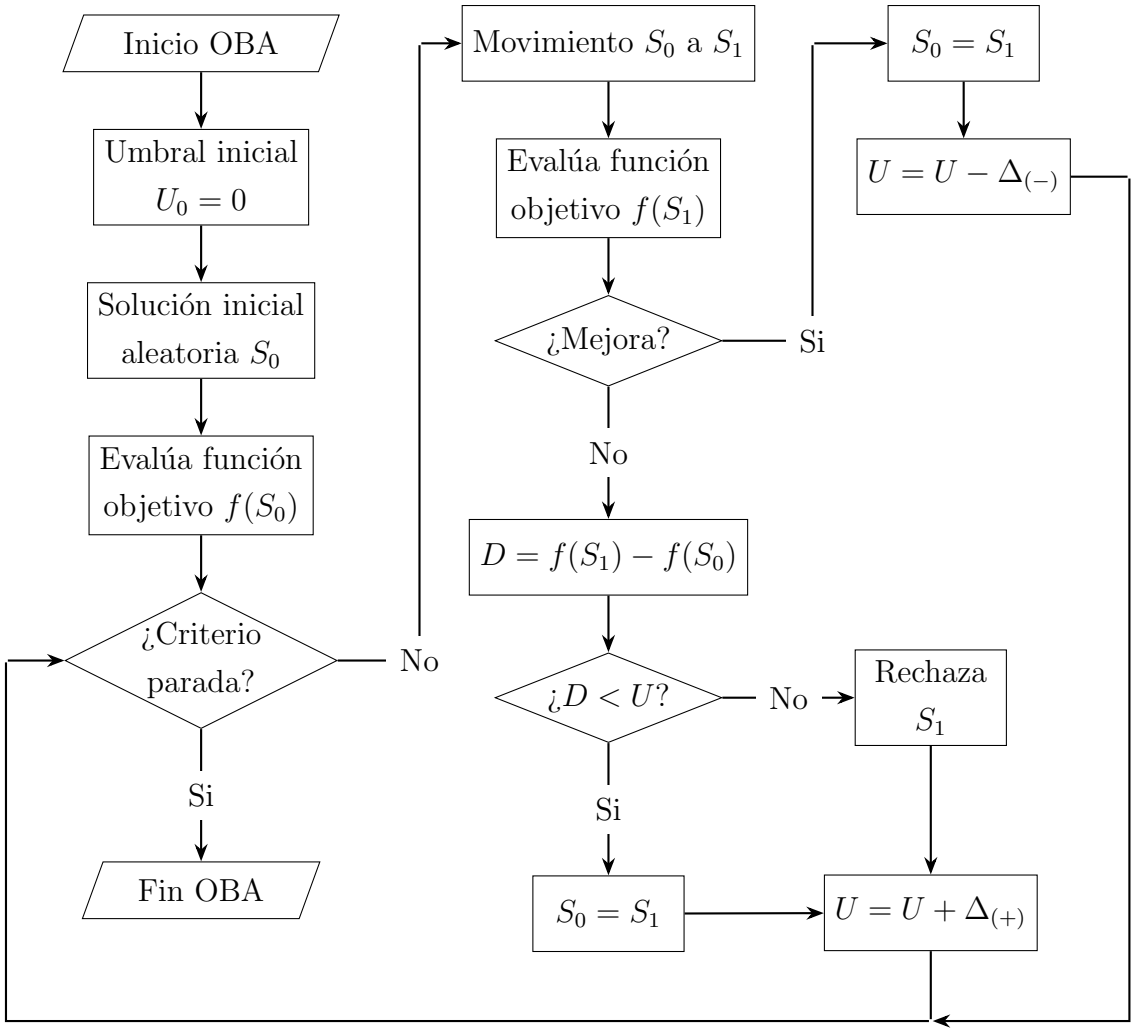


Figura 2.9: Diagrama de flujo del funcionamiento de un algoritmo *Old Bachelor Acceptance* (Elaboración propia).

Es especialmente relevante considerar el hecho de que, en su formulación inicial, los umbrales pueden llegar a tomar valores negativos. Esta característica hace que las fases de intensificación, en las que el algoritmo se acerca a un óptimo, tengan una

velocidad de convergencia considerablemente rápida.

Como cabe entender, al no contar con un parámetro de control que se reduce en el tiempo, este algoritmo no tiene un punto final que pueda asemejarse a los del SA o TA. Es por esto que el criterio de parada común para un algoritmo OBA es el limitar la duración del proceso, ya sea mediante el establecimiento de un tiempo límite, o un número máximo de iteraciones (M).

2.2.2. Algoritmos evolutivos

En 1753 Georges-Louis Leclerc de Buffon, valora por primera vez la idea de que todas las especies existentes fueron generadas a partir de una única especie que se ha ido perfeccionando y degenerando a lo largo del tiempo [38]. En este periodo, el naturalista contemporáneo Jean-Baptiste Lamark afirmó que las características físicas adquiridas por un organismo durante su vida, pueden ser transmitidas a sus descendientes [39]. Esta afirmación fue rebatida cincuenta años después por Charles Darwin, quien, en contra del Lamarkismo, afirmaba que la evolución consiste en la conjunción de cambios aleatorios en las características de los individuos, y un proceso de selección natural [40].

Durante ese mismo periodo Gregor Mendel desarrolló sus famosos experimentos de hibridación vegetal, enunciando las leyes básicas de la transmisión de la herencia [41]. Teoría relacionada con el denominado Efecto Baldwin, desarrollado por James Baldwin, este enuncia que, un organismo con altas capacidades de aprendizaje tendrá una mayor probabilidad de supervivencia y, por lo tanto, dará lugar a una mayor transmisión de genes del aprendizaje [42, 43].

El concepto de evolución, derivado en gran parte de los estudios mencionados anteriormente, es el fundamento de la rama de algoritmos de resolución que se presenta a continuación. Los algoritmos evolutivos basan su funcionamiento en la generación de poblaciones de soluciones que, mediante una serie de reglas aplicadas al individuo, son seleccionadas, combinadas y remplazadas por nuevas generaciones. En este tipo de algoritmos es necesario el concepto de muerte del individuo, algo que permite el paso de una generación a la siguiente. El resto de la sección es dedicada a presentar los tres principales tipos de algoritmos evolutivos.

2.2.2.1. Programación evolutiva

La programación evolutiva aparece en el año 1960 de mano de Lawrence J. Fogel [44]. Esta estrategia, en la que cada individuo consiste en un vector de longitud fija y valores reales, se caracteriza por enfatizar los nexos entre padres e hijos, así como el hecho de que no existe cruzamiento entre soluciones.

El proceso consiste en la creación de una población inicial de individuos, compuesta por una serie de soluciones aleatorias. Tras esto, se genera una nueva población con el mismo número de individuos, algo que se consigue aplicando una mutación Gaussiana a cada uno de los individuos. Se evalúa entonces la aptitud de cada uno de los individuos de ambas generaciones, y se selecciona, mediante torneo estocástico, qué individuos han de permanecer. Esto consiste en la elección de pares de padres e hijos que son enfrentados, sobreviviendo aquel que tenga una mayor aptitud.

2.2.2.2. Estrategias evolutivas

Fundamentados en un proceso similar a la ya mencionada programación evolutiva, las estrategias evolutivas fueron desarrolladas de forma conjunta por Ingo Rechenberg y Hans-Paul Schwefel [45, 46]. En este caso, al ser un algoritmo extintivo, el proceso de selección es determinista, de forma que el individuo menos capaz tiene probabilidad nula de supervivencia. Las soluciones siguen siendo un vector de longitud fija y valores reales, a los que se aplica, además de una mutación Gaussiana, un proceso de recombinación que se describe más adelante.

El **proceso original** toma un padre, se le aplica una mutación Gaussiana $N(0, \sigma)$ de forma que se genera un hijo. Tras esto, debido a que es una estrategia extintiva (EE), sólo sobrevive aquél que cuente con mejor aptitud de los dos.

$$(1 + 1) - EE \tag{2.4}$$

Este proceso fue modificado, introduciendo el operador recombinación, este consiste en la creación de los hijos como resultado de combinar dos individuos de la anterior generación. Esto no elimina el operador mutación, que es aplicado a cada uno de los hijos generados tras su creación. El operador recombinación puede ser de dos tipos:

- **Sexuales**, donde se toma, de forma aleatoria, dos individuos de la generación

de padres para generar un hijo.

- **Panmíticos**, eligiendo un padre fijo que se combina con diferentes individuos elegidos aleatoriamente para, con cada uno de ellos, dar lugar a un hijo.

La primera modificación de la estrategia consiste en la toma de ν padres, estos generan un único hijo, y el individuo menos apto de todos los presentes, muere.

$$(\mu + 1) - EE \tag{2.5}$$

Esta modificación deriva en la siguiente, donde tomándose ν padres, se generan λ hijos. Tras ello, se eliminan uno a uno los peores individuos del conjunto formado por ambas generaciones hasta que el número total de individuos vuelve a ser μ.

$$(\mu + \lambda) - EE \tag{2.6}$$

Finalmente, se desarrolla la estrategia donde, al igual que en el caso anterior, μ padres generan λ hijos. Sin embargo, en este caso sólo sobreviven μ individuos de la nueva generación. Por esto ha de cumplirse que $\lambda \geq \mu$.

$$(\mu, \lambda) - EE \tag{2.7}$$

2.2.2.3. Algoritmos genéticos

Dando paso ahora al tercer y último tipo de algoritmos evolutivos, los algoritmos genéticos son la tipología de algoritmo evolutivo más utilizada en la optimización estructural. Desarrollados por Holland (1975) [47], los algoritmos genéticos se basan en los fundamentos de la evolución enunciados por Darwin. Según estos, la evolución es consecuencia de dos factores principalmente, la **selección natural** y la **reproducción sexual**. Este segundo factor engloba tanto el denominado **cruzamiento** entre padre y madre, como las **mutaciones aleatorias** consistentes en la modificación espontánea de la información genética, pudiendo ser estas de carácter positivo o negativo [40].

La evolución se basa entonces en una serie de unidades distinguidas:

- **Genes**, la unidad básica de la herencia consistente en fragmentos de información encadenados.
- **Genotipo**, resultado de la combinación de los genes.
- **Fenotipo**, expresión física del genotipo.
- **Locus**, es la posición que un determinado gen ocupa dentro del cromosoma.
- **Alelo**, la cantidad de variables que pueden ser ocupadas por un mismo locus.
- **Aptitud**, la capacidad de sobrevivir.

Naturaleza		**Algoritmo Genético**
Fenotipo	-	Solución
Genotipo	-	Estructura
Cromosoma	-	Vector
Gen	-	Elemento del vector
Alelo	-	Valor del elemento
Locus	-	Posición del elemento
Aptitud	-	Función objetivo

Tabla 2.1: Unidades de la evolución y sus equivalencias en un algoritmo genético.

Los algoritmos genéticos desarrollados por Holland se componen de dos elementos necesarios, el primero son cada una de las soluciones expresadas en forma de vector binario, algo que denomina ***string***. El segundo es la aptitud de cada uno de esos vectores, esta determina la capacidad de supervivencia que tiene cada uno de ellos y se denomina ***fitness*** [47].

De forma similar, pero considerando la utilización de estructuras distintas, que no tienen por qué ser binarias, las soluciones de De-Jong eran **grafos** [48].

0	1	1	0	0	0	1	0	0	1	1	1	1	0	0	0
Cadena 1				**Cadena 2**				**Cadena 3**				**Cadena 4**			

Figura 2.10: Vector de soluciones donde se pueden ver los diferentes *strings* (Elaboración propia).

El funcionamiento de los algoritmos genéticos pasa por la generación de una población inicial de soluciones de forma aleatoria. Es importante que esta población inicial tenga características variadas y diferenciadas de forma que sea representativa para muchos tipos de soluciones del problema. Sin embargo, la necesidad de que sea representativa puede dar lugar a la elección de un número innecesariamente alto de individuos, algo que aumenta considerablemente el tiempo de cálculo del problema. En este contexto se ha de valorar que, por una parte la población ha de ser suficientemente grande como para evitar problemas de convergencia prematura del algoritmo, pero por otra, no ha de ser demasiado grande ya que esto daría lugar a un proceso de computación muy largo.

Tras esto se da paso a la evaluación de cromosomas donde, tras calcular el valor de la función objetivo ($V_{objetivo}$), se mide la denominada infactibilidad de soluciones (I), en conjunto, estos dos factores, permiten calcular la aptitud (A) de cada individuo. Es importante destacar que la infactibilidad no descarta un individuo de forma directa, sino que es penalizada mediante un factor ($K_{penal.}$).

$$A = V_{objetivo} - (K_{penal.} \cdot I) \tag{2.8}$$

Una vez determinada la aptitud de cada individuo, se lleva a cabo la **selección de los padres** para dar lugar a una nueva generación. La probabilidad de que un padre sea elegido para el cruzamiento es directamente proporcional a su aptitud, cumpliendo el denominado principio de la ruleta.

Tras aplicar el principio de selección, ha de tener lugar el **cruzamiento** de individuos. Existen diferentes formas de las que este proceso puede desarrollarse, es decir, hay distintos operadores de cruzamiento.

- Un punto, donde se corta la cadena de ambos padres en un punto de forma aleatoria y se intercambia.

- Dos puntos, mismo procedimiento que el anterior, pero en este caso se corta en dos puntos.

- Uniforme, donde cada uno de los bits de la cadena hijo es tomado de un padre.

- PMX (*Partially-Matched Crossover*) y SEX (*Strategic Edge Crossover*), operadores algo más complejos que mezclan y aleatorizan los descritos anteriormente.

Mediante el cruzamiento de padres, se consigue una nueva generación de hijos cuyas características son resultado de la combinación de las presentes en la generación anterior. Además, gracias a que los padres con mayor aptitud son elegidos prioritariamente, aquellas características que sean positivas para la solución prevalecerán y aparecerán de forma más reiterada en la generación de hijos.

Sin embargo, basándose en lo enunciado por Darwin, el proceso de evolución no está completo pues hace falta considerar las **mutaciones** que modifiquen, de forma aleatoria, características de ciertos individuos de la nueva generación [40]. En la naturaleza estas mutaciones consisten en el duplicado o eliminación de un cromosoma. En los algoritmos genéticos esto equivale en tomar una variable de la solución y cambiar el orden en el que aparece [47, 48].

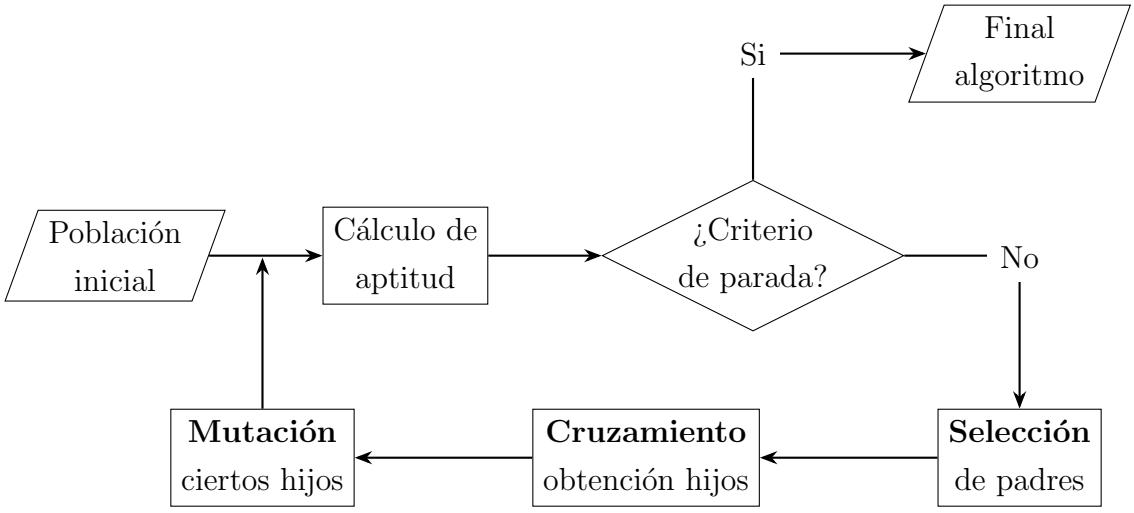


Figura 2.11: Diagrama del funcionamiento general de los algoritmos genéticos (Elaboración propia).

Es importante valorar de forma adecuada el proceso de mutación en lo que respecta tanto a la cantidad de individuos que mutan en cada generación, como la forma en la que estos individuos han de mutar. En caso de que la generación sufra mutaciones que modifiquen sus características de forma demasiado severa, esta podría separarse de la generación anterior. Por otra parte, si las mutaciones no son suficientemente

notables, podrían perderse rasgos de interés que no aparecerán por no estar presentes en la generación inicial de padres.

2.2.3. Redes neuronales artificiales

Finalmente, la metodología más reciente aplicada en la resolución de problemas de optimización son las denominadas Redes Neuronales Artificiales o *Artificial Neural Network* (ANN). Una ANN es un sistema computacional basado en los procesos biológicos que constituyen el cerebro animal. Está compuesto por unidades elementales denominadas neuronas que, al estar interconectadas entre sí de forma muy densa, reciben y transmiten información a través de la red que conforma el sistema hasta producir un estímulo de salida. Mediante procesos en paralelo tienen una gran capacidad para entender, simular y llegar a predecir procesos lineales, pudiendo aplicar este aprendizaje derivado de la experiencia de casos anteriores, a casos nuevos.

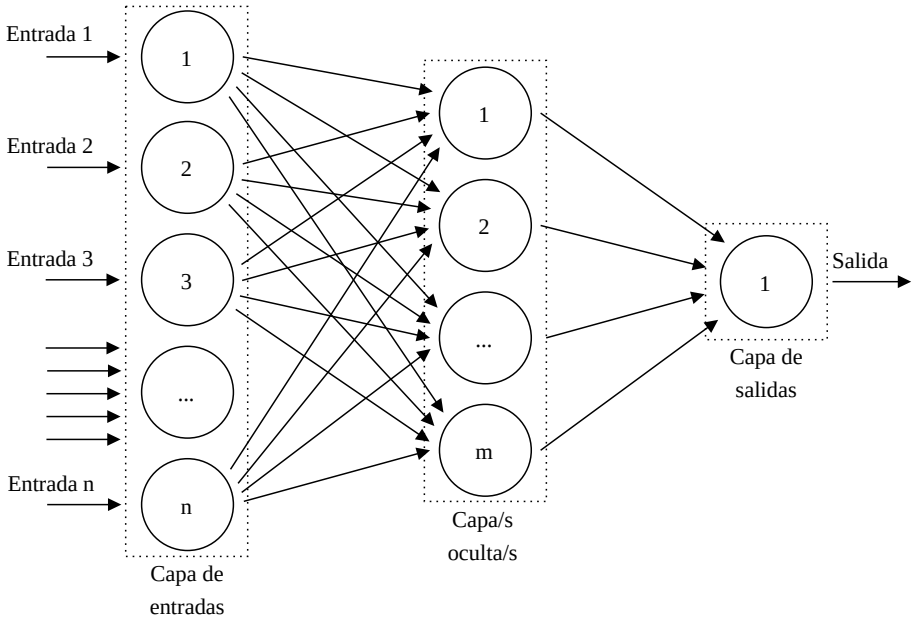


Figura 2.12: Esquema general de una red neuronal artificial (Elaboración propia).

El primer modelo de neurona artificial fue presentado en 1946 por McChulloch y Pitts [49], sin embargo no fue hasta 1985 y 1989 esta tecnología no fue aplicada a problemas de optimización combinatoria y de optimización estructural, por Hopfield

y Tank, y Adeli y Yeh, respectivamente [50–52].

Las ANN son adecuadas para aquellos procesos donde no es sencillo llevar a cabo la modelización del problema, pero se dispone de una serie de casos iniciales cuya respuesta es conocida. Esto es así gracias a que las neuronas son elementos con procesos internos que, generalmente no son lineales, algo que permite el modelado de problemas que tampoco lo son. Además, mediante el proceso de aprendizaje supervisado, consiguen modelar de forma relativamente rápida las relaciones entre entrada y salida con una gran flexibilidad y adaptabilidad. Este modelado interno consigue que sean muy eficaces a nivel computacional y bastante insensibles a datos con ruido.

2.3. Estado del arte

Como cierre de este capítulo, una vez presentado el problema de optimización estructural, así como las principales técnicas heurísticas aplicadas en su resolución. A continuación se presenta un breve resumen de la revisión bibliográfica llevada a cabo durante el desarrollo de la investigación. Algo que comprende la aparición, evolución y desarrollo de diferentes técnicas de optimización aplicadas a distintos elementos estructurales.

2.3.1. Antecedentes y evolución de la optimización de estructuras

El concepto de optimización en el diseño de estructuras existe desde hace siglos. Si bien es cierto que la aplicación de nuevas metodologías ha conseguido un desarrollo y resultados muy superiores, científicos como Leonardo da Vinci o Galileo Galilei ya desarrollaron estudios sobre el procedimiento para reducir el peso de estructuras de madera [53]. Es decir, en los siglos XV y XVI ya existía el interés de optimizar ciertas características, en esos casos el peso, de los diseños estructurales de la época.

Durante siglos posteriores, científicos tan conocidos como Newton, Euler, Lagrange o Bernoulli también indagaron en la optimización de ciertos tipos de estructuras básicas [54]. Sin embargo, el inicio del siglo XX marca el comienzo de la optimización estructural como disciplina científica. El estudio de arcos y cerchas metálicas llevado a cabo por Maxwell, Levi y Mitchell en los años 1869 a 1904 conforma las

bases científicas de la optimización estructural [55–57]. Estos autores buscaban, para un sistema de cargas determinado, arcos con forma óptima, y cerchas metálicas de mínimo peso. Considerando estructura óptima aquella que verifica algún criterio tal como que todos sus elementos trabajen a una tensión similar, o que realice un trabajo virtual mínimo.

La relevancia de estos trabajos va más allá de los resultados obtenidos pues, independientemente de que la construcción de algunas de las estructuras derivadas de los estudios no sea viable, establecieron una metodología que permitía la obtención de soluciones. Algo que consiguió despertar el interés por la optimización estructural en muchos autores que, posteriormente, fueron los pioneros en el desarrollo de este campo.

En este contexto, en el año 1955 Klein B. enunció que la optimización estructural debía plantearse como un problema de optimización condicionada [58]. Esto, en conjunto con la aparición y rápida mejora de las herramientas computacionales de la época, marca un punto de inflexión en el proceso y metodología aplicados a la optimización estructural.

En 1960 Schmidt es el primer autor en plantear el problema de optimización estructural haciendo uso de elementos finitos y programación lineal, algo que denominó "*síntesis estructural*" [59]. Como se ha visto en apartados anteriores, el problema de optimización estructural es muy complejo, cuenta con muchas variables y suele ser no lineal. Esto hace que el uso del denominado proceso de "*síntesis estructural*" conlleve la simplificación, en gran medida, del problema estudiado.

Buscando una solución al problema de la no linealidad, autores como Karush, o Kuhn y Tucker desarrollan, durante los años setenta, los criterios de optimalidad [60, 61]. Técnicas que, mediante su correcta aplicación, permitían la resolución de problemas no lineales.

Los resultados de los trabajos mencionados anteriormente, en conjunto con las primeras aplicaciones de técnicas como el gradiente generalizado reducido o los multiplicadores de Lagrange aumentados, permiten el posterior desarrollo de este interesante campo de investigación.

2.3.2. Estructuras de hormigón armado

La mejora en las técnicas permite a los autores enfrentarse a problemas cada vez más complicados, obteniendo resultados cada vez más prometedores. Esto proporciona el marco científico necesario para que los autores mencionados a continuación puedan aplicar estas metodologías a las estructuras de hormigón armado.

De esta forma, los resultados de la optimización económica de estructuras aporticadas llevada a cabo por Moragues en 1980 fundamentan una serie de criterios de predimensionamiento enunciados por el autor [22]. Este estudio impone, a elementos de sección rectangular, la única restricción de cumplir el Estado Límite Último de agotamiento bajo solicitaciones normales. En conjunto con el hecho de que el número máximo de variables consideradas fuera siete, es evidente que obviar la influencia del cortante o los Estados Límite de Servicio conforma una simplificación considerable del problema.

En el año 1989 Arenas y Villegas llevan a cabo el dimensionamiento, mediante un proceso de análisis basado en prueba y error, del viaducto gallego de Cruzul, en Lugo [62]. Los cálculos realizados consideraban la no linealidad geométrica y de los materiales. Poco después, durante 1991, Gash lleva a cabo la optimización de forjados reticulares sometidos a cargas en dirección vertical [63]. Se trata de uno de los primeros trabajos en los que la optimización de la estructura cuenta con dos niveles. En una primera fase se fijaba la sección del forjado, siendo el número de soportes el objeto de optimización. Tras esto, una vez fijada la topología óptima, se procedía a la optimización de la sección.

El conjunto de estos y otros trabajos desarrollados hasta el año 1994 están contenidos en la revisión de la literatura llevada a cabo por Cohn y Dinovitzer [64]. Las principales conclusiones de la revisión establecen la necesidad de aplicar un enfoque mas práctico en el proceso de optimización. Algo que permitiría un acercamiento de este tipo de metodologías al ingeniero medio que desempeña su labor en el campo de la ingeniería estructural. Además de esto, los autores remarcan la necesidad de que estas técnicas sean aplicadas a estructuras de hormigón pues, hasta ese momento, algo mas del noventa por ciento de los estudios realizados estaba dirigido a estructuras metálicas.

La necesidad de aplicar estas técnicas a estructuras de hormigón armado, así

como de considerar cargas, secciones y restricciones más aplicables en la realidad, funciona como motor para que en los siguientes años un gran número de autores tales como Balling y Yao, Torrano o Rodrigues, mejoren en gran medida los conocimientos de este campo [65–67]. Esto es así gracias a los resultados obtenidos en el estudio de pórticos de edificación, secciones arbitrarias y edificios multiplanta desarrollados por los autores mencionados, respectivamente.

2.3.3. Optimización heurística de estructuras de hormigón armado

Dentro de las investigaciones que abordan la optimización de estructuras de hormigón armado, es especialmente relevante el desarrollo de la aplicación de técnicas heurísticas para la resolución del problema de optimización estructura.

En 1986 la optimización de celosías articuladas de diez barras mediante algoritmos genéticos llevados a cabo por Goldberg y Samtani conforman la primera aplicación de métodos metaheurísticos a la optimización estructural [68, 69]. Estudios posteriores tales como los desarrollados por Hajela, Jenkins o Rajeev y Krishnamoorthy aplican los algoritmos *Simple Genetic Algorythm* (SGA) definidos por Goldberg a estructuras sencillas [70–75]. Las principales conclusiones de estos estudios destacan la buena capacidad de este tipo de métodos para la resolución de problemas discretos. Además, hacen mención de que requieren el cálculo de la función objetivo en un número de ocasiones muy elevado, algo que se podrá solventar gracias a la mejora de las herramientas computacionales.

Desde la publicación de los primeros estudios mencionados anteriormente, hasta la actualidad, los algoritmos genéticos son la técnica metaheurística más se ha estudiado para la resolución del problema de optimización estructural. En 1997 Coello et al. [76], aplican por primera vez algoritmos genéticos para la optimización de vigas de sección rectangular de hormigón armado.

A partir de ese momento, existe un gran número de investigaciones que centran la aplicación de estas metaheurísticas en la resolución de optimización de estructuras de edificación. En India, autores como Chakrabarty [77], y, posteriormente, los ya mencionados Rajeev y Krishnamoorthy [74, 75], aplican algoritmos genéticos a la optimización de pórticos planos. Años despúes Srinivas y Ramanjaneyulu establecen

un procedimiento para el diseño óptimo de tableros viga pretensados basado en el uso de algoritmos genéticos [78]. Este estudio es especialmente interesante gracias al uso de una red neuronal que, una vez entrenada, permite predecir la factibilidad de las soluciones obtenidas.

Contemporáneamente, autores localizados en Estados Unidos tales como Coello o Camp [76, 79], aplican algoritmos genéticos a la optimización de vigas biapoyadas, y pórticos planos de hormigón armado, respectivamente. Estos estudios´, enfocados en la reducción del coste económico, consiguen un mejor modelado del problema de diseño, considerando un número más elevado de varibales, así como restricciones estipuladas en el código del *American Concrete Institute* (ACI).

Al otro lado del océano Atlántico, distintas universidades como la de Londres o Praga, amplían tanto el rango de estructuras al que se aplican las técnicas existentes, como el número de metodologías aplicables [80]. Se desarrollan así algoritmos combinados que, basados en los algoritmos genéticos, combinan las características fundamentales de estos con las de otras heurísticas como el recocido simulado. Además del desarrollo de nuevas técnicas, el interés por este campo da pie a su aplicación más allá de la optimización económica.

Fundamentado en la necesidad de controlar el impacto ambiental generado por la industria de la construcción, algo ya mencionado en el capítulo introductorio, grupos de investigación tales como el conformado por los docentes y personal de investigación del Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil de la Universidad Politécnica de Valencia centran sus esfuerzos en ampliar los objetivos del problema de optimización estructural más allá del ámbito económico.

De esta forma, desde el inicio de los años 2000, este departamento enfatiza la necesidad de considerar funciones objetivo con carácter ambiental. Esto conlleva la optimización, además del coste de la estructura, de las emisiones de CO_2 asociadas, o la energía consumida, durante su construcción. La consideración de más de una función objetivo conforma un problema de optimización multiobjetivo, algo de gran interés y especial relevancia en el panorama actual.

En este contexto, algunas de las primeras publicaciones del grupo tales como las presentadas por Perea et al. [81–84], durante los años 2004 a 2010 establecen las bases de la presente línea de investigación, llevando a cabo una forma inicial de

optimización económica de marcos de hormigón armado. Para ello hace uso de técnicas como el Recocido Simulado o Aceptación por Umbrales, obteniendo resultados positivos. Sin embargo, estas investigaciones se vieron limitadas por la necesidad de desarrollar un software de cálculo propio que mejorase la capacidad de computación de forma considerable. Esto es así debido a que, hasta ese momento, la parte referente al cálculo estructural y comprobación del marco era llevado a cabo por un software de cálculo externo, algo que conllevaba un mayor tiempo de cálculo.

Desde 2008 hasta 2015, Yepes et al. [13, 85], llevan a cabo el diseño óptimo y estudio paramétrico de muros de contención mediante Recocido Simulado. Además de conseguir muros con diseños óptimos desde el punto de vista económico y ambiental, estos estudios permitieron estimar la influencia de distintos parámetros sobre las características finales y comportamiento estructural del diseño. Durante el año final del periodo mencionado, se desarrolla una aproximación que permite la obtención y reducción del óptimo de Pareto en problemas de optimización multiobjetivo. Esta técnica es validada mediante su aplicación al caso de una viga en I de hormigón armado, consiguiendo diseños que, consiguiendo una durabilidad e impacto ambiental óptimos, son factibles en la realidad.

En ese mismo periodo temporal Payá et al. [86], establece una metodología para el diseño óptimo de marcos de edificación de hormigón armado. Esta se basa en la aplicación de un Algoritmo Multiobjetivo de Recocido Simulado (*MOSA*) que permite la consideración de cuatro funciones objetivo diferenciadas. Estas funciones comprenden el coste económico, la constructibilidad, el impacto ambiental, así como la seguridad general de la estructura. Dado que el principal objetivo de estos estudios es facilitar el acceso a este tipo de técnicas al ingeniero promedio, durante el año 2010 se establece una metodología para determinar el número de pruebas numéricas necesarias para que un algoritmo de Recocido Simulado proporcione una solución óptima con una precisión determinada.

Otros estudios tales como Carbonell et al. [87–89], o Torres-Machí et al. [90], desarrollan metodologías para la optimización de bóvedas de hormigón armado y vigas de alta resistencia, respectivamente. Estos estudios optimizan costes, emisiones de CO_2 y consumo de energía asociado a la construcción, para ello hacen uso de diferentes algoritmos tales como la Aceptación por Umbrales, Búsqueda Local Iterativa,

Búsqueda Reducida por Entornos, o Búsqueda por Entornos Variables Básica, entre otros.

Investigaciones posteriores a las mencionadas, tales como las llevadas a cabo por Martí Albiña et al. [91–93], o García Segura et al. [94–96], durante los años 2013 a 2016, desarrollan nuevas técnicas así como aplicaciones novedosas para este tipo de metodologías. De esta forma, se consigue llevar a cabo la optimización económica de puentes de carretera con prefabricados pretensados, así como tableros de puente de viga artesa pretensada prefabricada. Además, se desarrolló un nuevo algoritmo denominado Enjambre de Luciérnagas Híbrido (SAGSO) que, combinado con Recocido Simulado, permite la obtención de resultados de alta calidad en la resolución de problemas de optimización estructural. Algo validado por los resultados obtenidos en la optimización económica, así como de emisiones de CO_2 de vigas en I.

Finalmente es relevante destacar el trabajo que, a día de hoy, se sigue desarrollando en el departamento. En conjunto con el continuo flujo de publicaciones y estudios desarrollado por algunos de los autores mencionados anteriormente, publicaciones como Martínez et al. [20, 97], siguen presentando resultados de gran calidad en el campo del diseño óptimo de distintas tipologías estructurales.

Capítulo 3

Descripción particularizada del problema de optimización

Este capítulo presenta la estructura de estudio, algo que comprende un breve resumen de las tipologías existentes y principales usos, así como la definición de los diferentes parámetros y variables que delimitan el problema de optimización. Algo que permite la completa definición de la geometría del marco, así como de las secciones, la armadura pasiva, materiales y proceso de determinación del coste final de la estructura.

3.1. Introducción

Los marcos prefabricados de hormigón armado son estructuras de geometría rectangular muy versátiles. El uso generalizado de los marcos prefabricados de hormigón armado con geometría rectangular está extendido a la construcción de pasos inferiores. Sin embargo su gran versatilidad permite que, en función de sus dimensiones, pueda ser considerado como solución para drenajes transversales, desagües y saneamientos, o arquetas de grandes dimensiones.

La prefabricación de este elemento estructural permite una rápida puesta en obra. Además, el control sobre el proceso de fabricación en una planta de prefabricados es mucho más intensivo que el que se puede realizar en obras *in situ*. Esto, en conjunto con la estabilidad de las condiciones de curado del hormigón, permite la obtención

de productos con una calidad muy elevada y gran uniformidad.

Dentro de las aplicaciones mencionadas anteriormente, el estudio considera un marco destinado a un paso inferior. Las principales soluciones estructurales para pasos inferiores de carretera son los pórticos, **estructuras de tipo marco** o bóveda, y finalmente los pasos circulares de acero galvanizado.

Existen diferentes tipologías de marcos para pasos de carretera, en primer lugar es relevante distinguir entre los marcos monocelda y los marcos multicelda. La primera tipología permite, en conjunto, salvar luces de mayor tamaño.

El presente estudio considera el caso de un marco monocelda. Dentro de esta tipología se pueden distinguir, nuevamente, dos clases de marco. El primero de ellos es el denominado marco cerrado, este consiste en una única pieza de hormigón armado que constituye la totalidad de la sección. El marco cerrado puede ser adecuado para su uso en otro tipo de aplicaciones, sin embargo la necesidad de unas dimensiones mínimas que permitan el adecuado tránsito de vehículos y personas a través del paso inferior da lugar a problemas relacionados con su transporte hasta la obra.

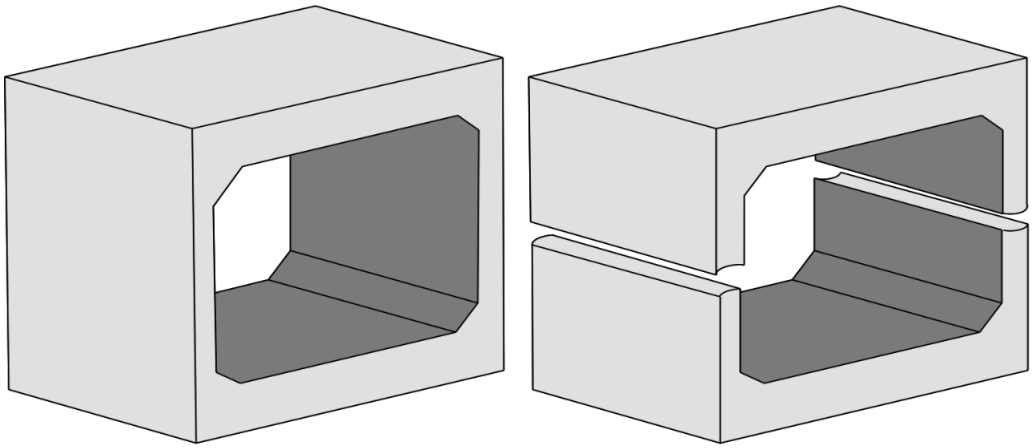


Figura 3.1: Ilustración de dos marcos prefabricados monocelda donde: a) Marco monocelda cerrado; b) Marco monocelda articulado (Elaboración propia).

Como solución a este problema, existe la tipología en la que se centra la investigación, el marco monocelda articulado. Compuesto por dos piezas con sección en U, este se transporta hasta la obra de forma más sencilla, evitando la necesidad de solicitar permisos para transportes especiales, reduciendo considerablemente los

tiempos de puesta en obra.

Una vez en su lugar de montaje, su ensamblado se lleva a cabo haciendo uso de grúas elevadoras de forma que cada una de las secciones necesarias quedan fijas mediante un sistema de unión machihembrada.

3.2. Parámetros

Los parámetros considerados en la optimización económica del marco prefabricado han de permitir la definición completa del problema. Estos pueden ser relativos a la geometría del marco, las acciones actuantes, los coeficientes de seguridad, grado de exposición ambiental y cualquier otra característica cuya definición es necesaria para el cálculo y verificación de la estructura, así como la evaluación de la función objetivo.

3.2.1. Parámetros geométricos y de armado

La geometría del marco queda definida por tres parámetros que se corresponden con la altura del marco, luz salvada y altura a la que se encuentra la unión articulada. Las medidas estipuladas se corresponden con las distancias entre las líneas medias de las secciones finales del marco. Además de estos tres, considerando que el cálculo de marcos se realiza por metro lineal, ya que así lo estipula la Guía de cimentaciones en obras de carretera del Gobierno de España, se establece un parámetro correspondiente al ancho de la estructura.

Además de estos, se establecen una serie de parámetros referentes a la disposición de armaduras de cortante, así como longitudes de los redondos de refuerzo en esquinas y flexión negativa en la losa superior. Gracias a la naturaleza simétrica de la estructura es posible definir ambos lados del marco haciendo uso de los mismos parámetros.

El conjunto de los once parámetros geométricos que definen el marco, así como ciertas características necesarias para llevar a cabo el proceso de optimización de su armadura, pasiva se indican en la tabla 3.1 y en la figura 3.2, junto con los valores concretos considerados.

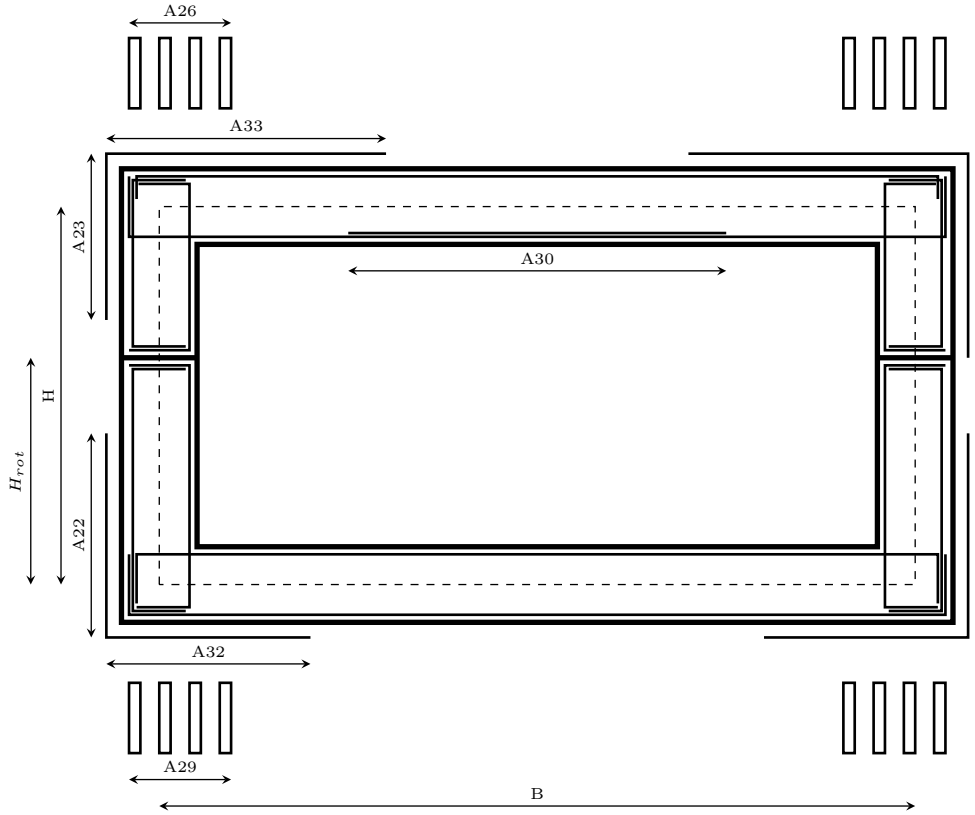


Figura 3.2: Representación gráfica de los parámetros geométricos del marco (Elaboración propia).

Descripción	**Parámetro**	**Valor**
Altura del marco (m)	H	5
Luz salvada (m)	B	10
Altura de la unión articulada (m)	H_{rot}	3
Anchura de cálculo del marco (m)	b	1
Long. del refuerzo de la esquina inf. dir. vertical (m)	A22	2
Long. del refuerzo de la esquina inf. dir. horizontal (m)	A32	2
Long. del refuerzo de la esquina sup. dir. vertical (m)	A23	1.5
Long. del refuerzo de la esquina sup. dir. horizontal (m)	A33	3
Disposición armadura de cortante en la losa sup. (m)	A26	3
Disposición armadura de cortante en la losa inf. (m)	A29	2
Long. armadura de refuerzo de flexión en losa sup. (m)	A30	5

Tabla 3.1: Parámetros geométricos del problema junto con los valores concretos considerados.

3.2.2. Parámetros relativos a las acciones

Las acciones actuantes sobre el marco son aquellas cargas, o conjunto de cargas que, impuestas sobre la estructura, dan lugar a estados tensionales. Estas acciones son definidas por las normas pertinentes, en el caso de estudio están recogidas en la norma UNE-EN 1992-1-1 (Eurocódigo 2) [98], así como en la Guía para el Proyecto de Cimentaciones en Obras de Carretera del Gobierno de España [99].

Los parámetros relativos a las acciones actuantes sobre la estructura, junto con los valores concretos considerados en el estudio quedan indicados en la tabla 3.2.

Descripción	**Parámetro**	**Valor**
Peso específico del hormigón armado (kN/m^3)	γ_{HA}	25
Peso específico de la tierra de relleno (kN/m^3)	γ_T	20
Profundidad de enterramiento del marco (m)	e_T	1.5
Ángulo de rozamiento del relleno ($^\circ$)	SU_{LS}	30
Coeficiente de empuje al reposo	K_{SU}	0.5
Carro pesado IAP (kN/m^2)	CL_{carro}	150
Dimensiones carro pesado IAP (m	L_{carro}	1.2
Sobrecarga uniforme IAP (kN/m^2)	SU_{IAP}	10

Tabla 3.2: Parámetros relativos a las acciones junto con los valores concretos considerados en el caso de estudio.

Los parámetros mostrados en la tabla permiten la completa definición de las acciones que actúan sobre la estructura. Es relevante mencionar que, durante el proceso de modelado del problema es necesario plantear una serie de hipótesis relativas a las acciones. Estas permiten adecuar el proceso de modelado al estudio, adecuándolo para cumplir los requisitos establecidos en norma.

- Las dimensiones del marco su naturaleza prefabricada, en conjunto con la consideración de que hay presentes juntas de dilatación suficientemente próximas hace posible despreciar los efectos de las acciones reológicas y térmicas.

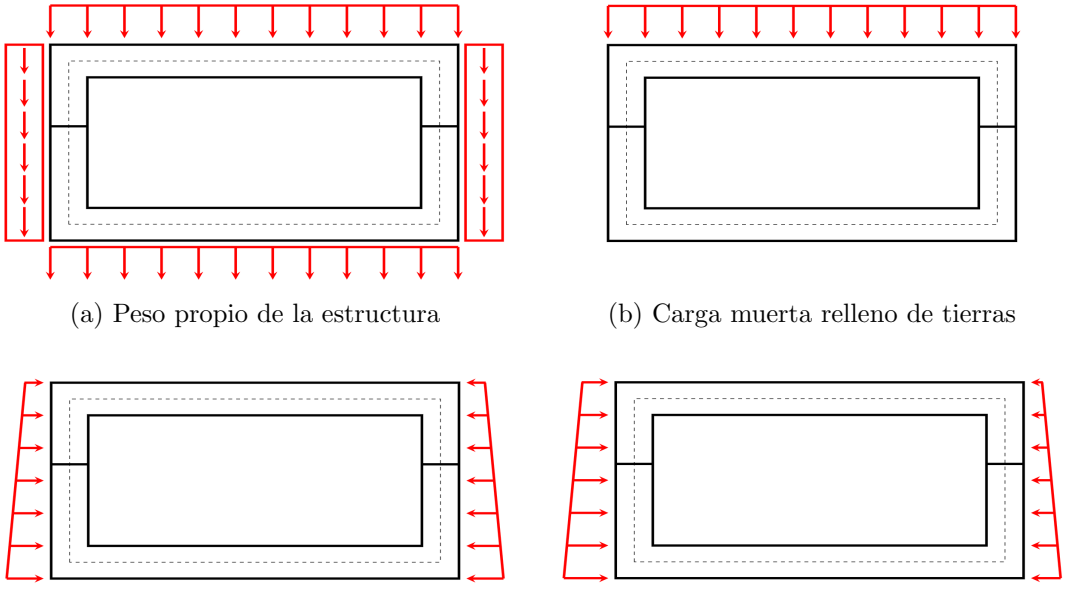
- Los asientos diferenciales de la cimentación de la estructura son despreciables.

- La localización del marco permite considerar que este se encuentra en una zona no sísmica, y por lo tanto no es necesario considerar los posibles efectos de este tipo de acciones.

- Se considera que la magnitud de la sobrecarga producida por el Efecto Marston toma el valor límite establecido en norma. Este efecto debido al rozamiento negativo es modelado mediante un coeficiente de ponderación que se aplica a la carga muerta sobre la losa superior.

- El vehículo pesado de la IAP es modelado mediante una carga distribuida localizada en el centro de la losa superior, la magnitud y longitud de aplicación son establecidas conforme lo presentado en la Guía de Cimentaciones del Gobierno de España.

Cada uno de los casos de carga contemplados en el presente problema queda representado en las figura 3.3 y 3.4.



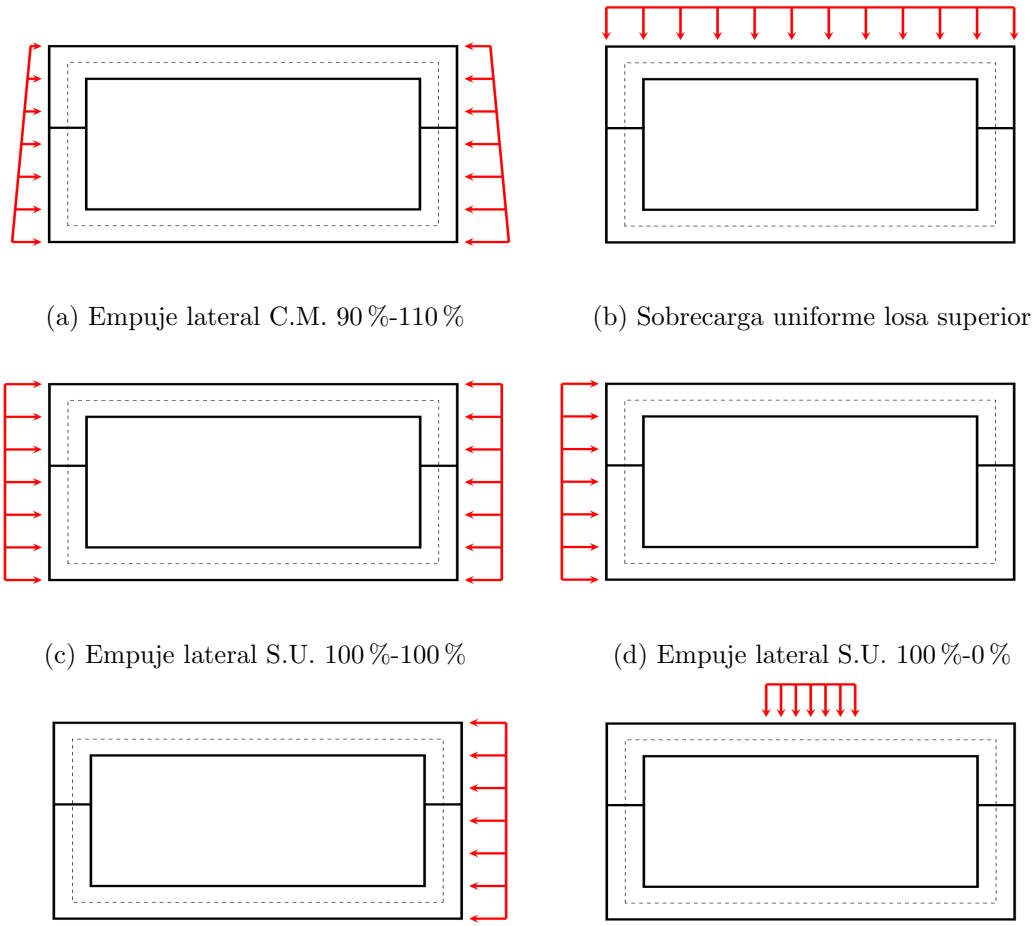
(a) Peso propio de la estructura

(b) Carga muerta relleno de tierras

(c) Empuje lateral C.M. 100 %-100 %

(d) Empuje lateral C.M. 110 %-90 %

Figura 3.3: (A) Casos de carga considerados en el cálculo del marco prefabricado (Elaboración propia).



(a) Empuje lateral C.M. 90 %-110 %

(b) Sobrecarga uniforme losa superior

(c) Empuje lateral S.U. 100 %-100 %

(d) Empuje lateral S.U. 100 %-0 %

(e) Empuje lateral S.U. 0 %-100 %

(f) Carga localizada carro pesado

Figura 3.4: (B) Casos de carga considerados en el cálculo del marco prefabricado (Elaboración propia).

3.2.3. Parámetros relativos a los coeficientes de seguridad y grado de exposición de la estructura

La aplicación del "Método de los Estados Límite" para el cálculo y comprobación del marco hace necesario la consideración de una serie de parámetros relativos a los coeficientes de minoración de los materiales (γ_c, γ_s), y mayoración de acciones ((γ_f)).

Además de estos parámetros se ha de establecer el recubrimiento de las armaduras pasivas, parámetro función de las condiciones ambientales a las que se verá sometida

la estructura. En el caso de estudio, tratándose de una estructura enterrada, la clase de exposición estipulada en el EC2 es la XC2.

En el caso del coeficiente de mayoración de acciones, este se establece en función del de control de ejecución, el nivel considerado, así como el valor concreto del recubrimiento nominal de las armaduras pasivas y el resto de parámetros mencionados quedan contenidos en la tabla 3.3.

Descripción	**Parámetro**	**Valor**
Coeficiente de minoración de resistencia (hormigón)	γ_C	1.5
Coeficiente de minoración de resistencia (acero)	γ_S	1.15
Nivel de Control de Ejecución	NCE	Normal
Clase de Exposición	CE	XC2
Recubrimiento nominal de las armaduras (mm)	r_{nom}	35

Tabla 3.3: Parámetros relativos a los coeficientes de seguridad de los materiales y grado de exposición de la estructura.

3.3. Variables de diseño

Un total de treinta y una variables de diseño han sido consideradas en la optimización del marco.

Las tres primeras variables son geométricas, se trata del canto de los hastiales (h_H), el canto de la losa inferior (h_{LI}), y el canto de la losa superior (h_{LS}). En conjunto con los parámetros geométricos mencionados en el apartado anterior, estas variables permiten la completa definición de la geometría del marco.

Las dos siguientes variables de diseño se corresponden con los materiales, estas definen el tipo de hormigón del marco(C), y tipo de acero de las armaduras pasivas (S).

Finalmente, las veintiséis variables de diseño restantes se corresponden con la configuración de las armaduras pasivas del marco. Gracias a la naturaleza simétrica de la estructura es posible definir la armadura completa mediante variables que consideren una única mitad, para posteriormente igualar la opuesta. Estas variables de armado permiten la definición de la cuantía del total de las i armaduras dispuestas,

tratándose del diámetro de los redondos (ϕ_i), así como el número de barras (n_i) en el caso de armaduras longitudinales, y la separación entre ramas (s_i) en el caso de las armaduras de cortante dispuestas en las losas.

Las variables geométricas, así como las de separación entre ramas de la armadura de cortante son discretizadas para su correcta definición. Las variables correspondientes al canto pueden tomar valores cada dos centímetros dentro del rango propuesto, por otra parte la separación entre ramas de cortante puede tomar valores cada cinco centímetros en el rango establecido.

Finalmente, el diámetro de barras de armadura puede tomar los valores normalizados de uso común entre 10 y 32 milímetros, y el número de barras puede ser cualquier entero en el rango definido. De esta forma, la figura 3.5 muestra la configuración de variables considerada, junto con la denominación de cada una de las armaduras del marco prefabricado.

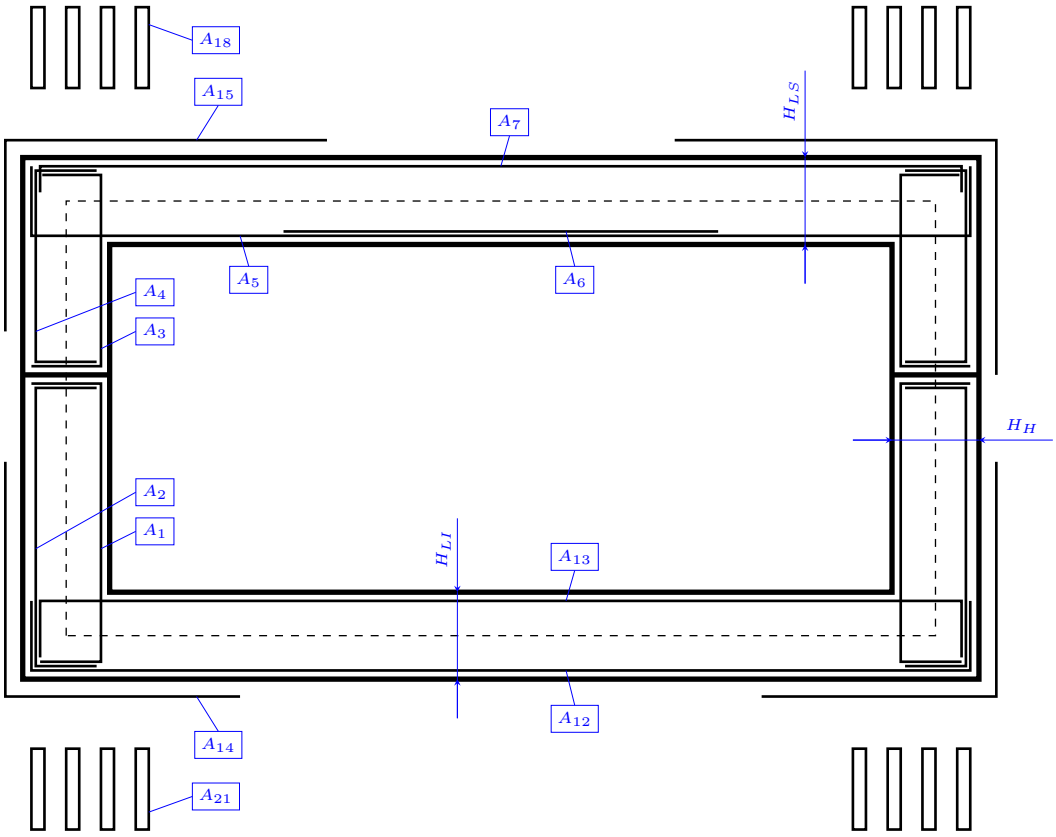


Figura 3.5: Ilustración de las variables de diseño del marco (Elaboración propia).

Finalmente, el conjunto de las 31 variables de diseño, en conjunto con el número de valores que estas pueden tomar en la resolución del problema, así como los límites establecidos, quedan resumidos en la tabla 3.4.

Descripción		**Variable**	**Número de valores**	**Rango de valores**
Canto hastiales	(m)	h_H	41	(0.4, 1.2)
Canto losa inferior	(m)	h_{LI}	41	(0.4, 1.2)
Canto losa superior	(m)	h_{LS}	41	(0.4, 1.2)
Tipo de hormigón	(MPa)	C	4	(20, 40)
Tipo de acero	(MPa)	S	2	(400, 500)
Arm. long. A1	(mm)	ϕ_{A_1}	6	(10, 32)
	$(barras)$	n_{A_1}	9	(4, 12)
Arm. long. A2	(mm)	ϕ_{A_2}	6	(10, 32)
	$(barras)$	n_{A_2}	9	(4, 12)
Arm. long. A3	(mm)	ϕ_{A_3}	6	(10, 32)
	$(barras)$	n_{A_3}	9	(4, 12)
Arm. long. A4	(mm)	ϕ_{A_4}	6	(10, 32)
	$(barras)$	n_{A_4}	9	(4, 12)
Arm. long. A5	(mm)	ϕ_{A_5}	6	(10, 32)
	$(barras)$	n_{A_5}	9	(4, 12)
Arm. de ref. A6	(mm)	ϕ_{A_6}	6	(10, 32)
	$(barras)$	n_{A_6}	10	(3, 12)
Arm. long. A7	(mm)	ϕ_{A_7}	6	(10, 32)
	$(barras)$	n_{A_7}	9	(4, 12)
Arm. long. A12	(mm)	$\phi_{A_{12}}$	6	(10, 32)
	$(barras)$	$n_{A_{12}}$	9	(4, 12)
Arm. long. A13	(mm)	$\phi_{A_{13}}$	6	(10, 32)
	$(barras)$	$n_{A_{13}}$	9	(4, 12)
Arm. de ref. A14	(mm)	$\phi_{A_{14}}$	6	(10, 32)
	$(barras)$	$n_{A_{14}}$	10	(3, 12)
Arm. de ref. A15	(mm)	$\phi_{A_{15}}$	6	(10, 32)
	$(barras)$	$n_{A_{15}}$	10	(3, 12)
Arm. de cort. A18	(mm)	$\phi_{A_{18}}$	7	(8 a 32)
	(m)	$s_{A_{18}}$	7	(0.1 a 0.4)
Arm. de cort A21	(mm)	$\phi_{A_{21}}$	7	(8 a 32)
	(m)	$s_{A_{21}}$	7	(0.1 a 0.4)

Tabla 3.4: Variables de diseño del marco prefabricado.

3.4. Restricciones de comportamiento y diseño

La imposición de una serie de restricciones al problema tiene el objetivo de asegurar que el marco resultado de la optimización tenga sentido físico y verifique los requerimientos estructurales establecidos en norma.

En este contexto, la Norma UNE-EN-1992-1-1 (EC2) establece la comprobación de la seguridad estructural mediante el uso del "Método de los Estados Límite". El procedimiento asociado a esta metodología en conjunto con las consideraciones realizadas para la resolución del problema son definidos en los siguientes apartados.

3.4.1. Principios del Método de los Estados Límite

El "Método de los Estados Límite" permite considerar la naturaleza aleatoria de la realidad estructural, algo asociado a las solicitaciones, resistencia de los materiales y dimensiones de la estructura. Esto se consigue mediante el uso de coeficientes de seguridad cuyo valor concreto está contenido en la norma.

El proceso para la aplicación de esta metodología comienza con la completa definición de las acciones actuantes. Tras esto, se han de definir una serie de Estados Límites que la estructura ha de verificar. La no superación de alguno de estos estados se corresponde con aquellas situaciones en las que la estructura deja de verificar alguna de las funciones para las que ha sido proyectada. En este contexto se distinguen dos tipos de estados límite:

- Estados Límite Últimos (E.L.U.)
- Estados Límite de Servicio (E.L.S.)

Tras el establecimiento de los estados que la estructura de estudio ha de verificar, se procede a su comprobación. Este proceso comprende el cálculo de los efectos que las combinaciones de acciones generan una vez aplicadas sobre la estructura, así como la respuesta de la estructura frente a las mismas. Una vez conocida la respuesta estructural, se comprueba el cumplimiento del estado límite, su verificación significa que la respuesta estructural es, con suficiente fiabilidad, superior al efecto de las acciones aplicadas.

Finalmente, existe una serie de consideraciones relativas a la disposición de la armadura asociadas a verificaciones geométricas que aseguren la viabilidad de las secciones diseñadas, así como cumplimiento de cuantías mínimas y máximas.

3.4.2. Acciones consideradas en el cálculo

En base a lo expuesto anteriormente, las acciones consideradas en el cálculo del marco son las presentadas en el apartado 3.2.2 "Parámetros relativos a las acciones". La adecuación del valor final de cálculo de cada una de las acciones conlleva las siguientes consideraciones.

1. Acciones permanentes (G)

   a) Peso propio de la estructura, el valor viene definido por las dimensiones de la estructura y el peso específico del hormigón armado.

   $$PP = V_C \cdot \gamma_C \tag{3.1}$$

   b) Carga muerta del relleno de tierras sobre la losa superior, cuya magnitud viene determinada por el volumen de tierras sobre el marco, función de la profundidad de enterramiento de la estructura, y el peso específico del relleno.

   $$CM_T = e_T \cdot B \cdot b \cdot \gamma_T = V_T \cdot \gamma_T \tag{3.2}$$

   c) Empuje lateral del terreno sobre los hastiales, mediante la consideración de un ángulo de rozamiento $\phi = 30°$, y un coeficiente de empuje al reposo $K_{SU} = 0{,}5$.

   $$K_T = V_T \cdot E_{SU} \tag{3.3}$$

2. Acciones variables (Q)

   - Sobrecarga uniforme, con una magnitud de 10 kN/m^2 actuante en la losa superior.
   - Empuje lateral sobre hastiales debido a la sobrecarga uniforme, calculado de forma análoga al empuje lateral del terreno.

- Carro pesado modelado como una carga localizada que se distribuye a lo largo de 1.2 metros en el centro de la losa superior, con una magnitud de 150 kN/m.

- El rozamiento negativo se modela mediante la aplicación de un factor multiplicador aplicado a la carga muerta sobre la losa superior. Este factor es calculado como el máximo valor establecido en la expresión de la Guía de Cimentaciones del Gobierno de España.

$$r = \gamma_{ap} \cdot \left(\frac{D+H}{2}\right) \cdot \left(\frac{1}{f}\right) \leq 0{,}3 \cdot \gamma_{ap} \cdot \frac{D^2}{B} \tag{3.4}$$

Es relevante mencionar la consideración de los estados de desequilibrio estipulados en norma, algo equivalente a modelar los empujes ponderados por un factor que permite distinguir la aplicación de la carga entre un 90 % y 110 % para los empujes debidos a las acciones permanentes. Y entre un 100 % y 0 % para los empujes debidos a las acciones variables.

La combinación del conjunto de todas las acciones mencionadas se lleva a cabo siguiendo las indicaciones estipuladas en el capítulo sexto de la Norma EN 1990, siendo consideradas un total de 48 diferentes combinaciones de casos de carga para la comprobación de estados límite.

3.4.3. Verificación de los estados límite

La verificación de los estados límite se lleva a cabo partiendo de un análisis elástico lineal de la estructura. Este considera un comportamiento elástico y lineal de los materiales, así como la existencia de equilibrio de la estructura sin deformar. El proceso de verificación parte de los esfuerzos de cada una de las secciones del marco, la modelización de la estructura, así como el proceso seguido para su cálculo queda detallado en capítulos posteriores.

De esta forma, este apartado presenta las hipótesis consideradas, así como procedimiento seguido en la verificación de cada estado límite, considerando conocidos los esfuerzos internos de cada una de las secciones.

3.4.3.1. Estado Límite Último de agotamiento por cortante

En primer lugar, se procede a la comprobación del E.L.U. de agotamiento por cortante. Siguiendo el procedimiento estipulado en el apartado 6.2 de la norma UNE-EN 1992-1-1 se comprueba cada una de las secciones del modelo.

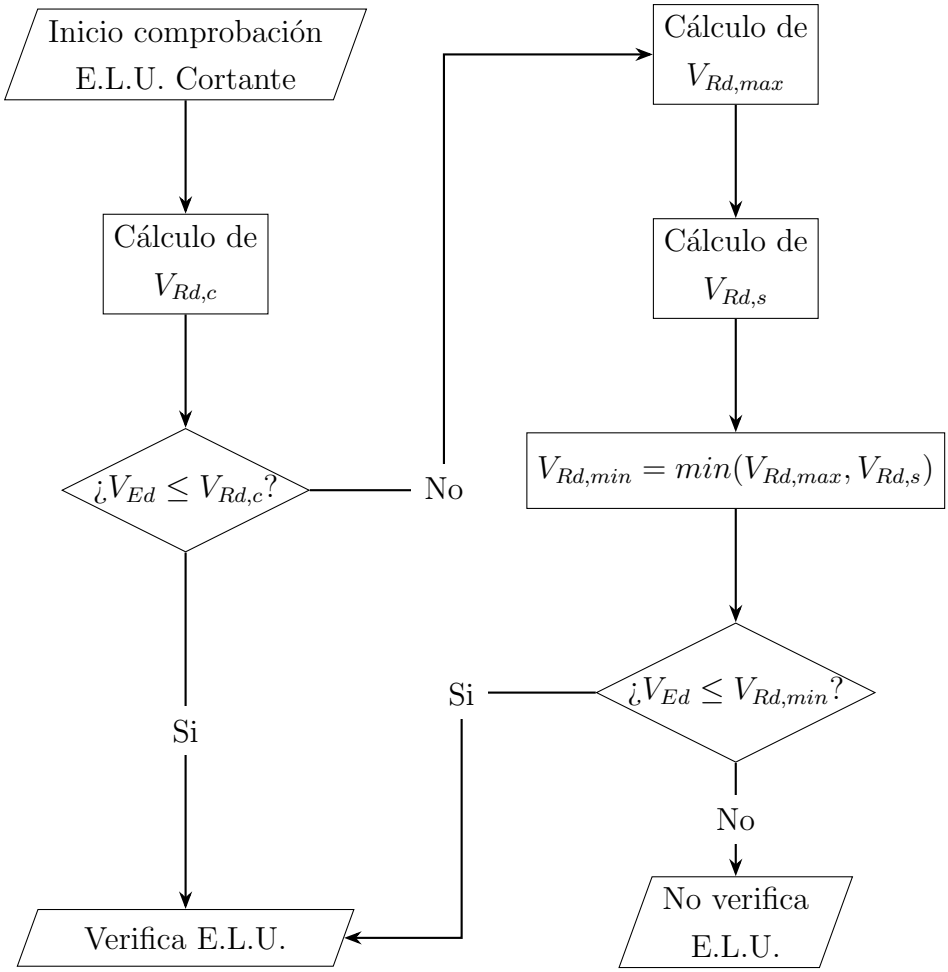


Figura 3.6: Diagrama de flujo del proceso de comprobación del estado límite de agotamiento por cortante (Elaboración propia).

El procedimiento de comprobación pasa en primer lugar por establecer si las sección requiere armadura de cortante. Para ello se obtiene el calor de cálculo para la resistencia a cortante de elementos que no requieren armadura de cortante $V_{Rd,c}$. Esta es comparada con el esfuerzo cortante actuante V_{Ed}, en caso de que el esfuerzo

actuante sea inferior, la sección verifica el estado limite, ya que no requiere armadura de cortante. De no ser así, pueden darse dos casos, el primero de ellos es que la sección no cuente con armadura de cortante, por lo que no verificaría el estado límite de forma directa. El segundo es que sí hay presencia de armadura de cortante, siendo necesario entonces verificar que el esfuerzo actuante es menor o igual al mínimo de los valores entre la resistencia de agotamiento por compresión oblicua $V_{Rd,max}$, y la resistencia de agotamiento por tracción en el alma $V_{Rd,s}$. Este proceso se muestra en la figura 3.4, los valores considerados, así como las expresiones necesarias para la comprobación se presentan a continuación.

El proceso comienza con la obtención del valor de cálculo de la resistencia a cortante de elementos sin armadura de cortante según la expresión 6.2.a de la norma.

$$V_{Rd,c} = [C_{Rd,c} \cdot k \cdot (100 \cdot \rho_1 \cdot f_{ck})^{1/3} + k_1 \cdot \sigma_{cp}] \cdot b_w \cdot d \tag{3.5}$$

con valor mínimo de

$$V_{Rd,c} = (\nu_{min} + k_1 \cdot \sigma_{cp}) \cdot b_w d \tag{3.6}$$

donde los valores considerados en la resolución son los siguientes

$C_{Rd,c} = \dfrac{0{,}18}{\gamma_C}$ es el valor recomendado en norma;

$k = 1 + \dfrac{200}{d^{0,5}} \leq 2$ con d en mm;

$\rho_1 = \dfrac{A_{sl}}{b_w \cdot d} \leq 0{,}02$;

$\sigma_{cp} = \dfrac{N_{Ed}}{A_c} < 0{,}2 \cdot f_{cd}$ con f_{cd} en MPa;

$\nu_{min} = \dfrac{0{,}075}{\gamma_C} \cdot k^{3/2} \cdot f_{ck}^{1/2}$

Una vez calculado el valor de la resistencia a cortante de elementos de hormigón sin armadura, esta es comparada con el esfuerzo cortante actuante sobre la sección.

$$V_{Ed} \leq V_{Rd,c} \tag{3.7}$$

En caso de que este sea menor al resistente, la sección cumplirá en todo caso, de no

ser así, el hormigón agota por tracción en el alma y es necesario continuar con la comprobación.

La norma UNE-EN 1992-1-1 basa el cálculo de elementos con armadura de cortante en un modelo de celosía plana cuyas bielas tienen una inclinación θ determinada. Este modelo permite el cálculo de de la resistencia de agotamiento por compresión oblicua de las armaduras, así como la de agotamiento por tracción en el alma.

La resistencia de agotamiento por compresión oblicua del caso de estudio, donde las armaduras de cortante son rectas ($\alpha = 90°$), se obtiene mediante la expresión 6.9 de la norma.

$$V_{Rd,max} = \frac{\alpha_{cw} \cdot b_w \cdot z \cdot \nu_1 \cdot f_{cd}}{cot(\theta) + tan(\theta)} \tag{3.8}$$

Donde los valores considerados son los siguientes

$\alpha_{cw} = 1$ dado que la estructura no es pretensada;

$z = 0{,}9 \cdot d$ es la aproximación del brazo mecánico;

$\nu_1 = 0{,}6 \cdot (1 - \frac{f_{ck}}{250})$ con f_{ck} en MPa;

$\theta = 45°$ dado que $cot(\theta) = 1$ es el caso más restrictivo.

Y la resistencia de agotamiento por tracción en el alma de hormigón con armadura de cortante se obtiene mediante la expresión 6.8 de la norma.

$$V_{Rd,s} = \frac{A_{sw}}{s} \cdot f_{ywd} \cdot cot(\theta) \tag{3.9}$$

Donde los valores del problema son

A_{sw} es el área de la sección transversal de la armadura de cortante;

s es la separación entre cercos;

$z = 0{,}9 \cdot d$ es el ya mencionado brazo mecánico aproximado;

$f_{ywd} = \frac{f_{yd}}{\gamma_S}$

Una vez determinados los valores de estas resistencias, la sección verifica el estado límite en caso de que el esfuerzo actuante sea igual o menos al mínimo de los valores

calculados anteriormente.

$$V_{Ed} \leq min(V_{Rd,max}, V_{Rd,s}) \tag{3.10}$$

De esta forma, mediante la repetición de este procedimiento para cada una de las secciones consideradas, se determina la verificación del estado límite de agotamiento por cortante para el marco prefabricado.

Establecido esto, antes de dar paso a la verificación del E.L. de solicitaciones normales, es importante considerar el incremento de tracción en la armadura longitudinal generado por el esfuerzo cortante. El esfuerzo de tracción adicional ΔF_{td} se puede calcular mediante la expresión 6.18 de la norma.

$$\Delta F_{td} = 0{,}5 \cdot V_{Ed} \cdot (cot(\theta) - cot(\alpha)) \tag{3.11}$$

El efecto de este incremento puede entonces ser estimado decalando la ley de momentos actuantes en la sección un determinado ΔM_{Ed}.

$$\Delta M_{Ed} = \Delta F_{td} \cdot z \tag{3.12}$$

Una vez aplicada la ley de decalaje se puede dar paso a la comprobación del estado límite frente a solicitaciones normales.

3.4.3.2. Estado Límite Último frente a solicitaciones normales

El proceso de comprobación del estado límite frente a solicitaciones establecido en el apartado 6.1 de la norma UNE-EN 1992-1-1 conlleva la obtención del diagrama de interacción de cada una de las secciones de interés. Este diagrama representa puntos cuyas coordenadas se corresponden con el momento y axil último resistente de la sección.

La determinación de esos pares, axil y momento, está fundamentada en el modelo de dominios de deformación, empleando las siguientes hipótesis:

1. El plano de deformación de la sección seguirá siendo plano.

2. El agotamiento del material está asociado a una serie de deformaciones establecidas en norma. Esto permite evitar que, por la naturaleza de la curva resistente

del hormigón, haya dos deformaciones asociadas a una misma tensión.

a) Deformación unitaria de agotamiento a tracción de la armadura pasiva.

$$\varepsilon_{uk} = 0{,}010 \tag{3.13}$$

b) Deformación unitaria asociada al límite elástico de las armaduras pasivas.

$$\varepsilon_{ud} = \frac{f_{yk}}{\gamma_S} \tag{3.14}$$

c) Deformación unitaria de agotamiento por compresión en flexión compuesta del hormigón.

$$\varepsilon_{c3} = 0{,}00175 \tag{3.15}$$

d) Deformación unitaria de agotamiento por compresión simple del hormigón.

$$\varepsilon_{cu3} = 0{,}0035 \tag{3.16}$$

3. El alargamiento en la armadura pasiva adherente es igual al del hormigón circundante, tanto en compresión como en tracción.

4. Se desprecia la resistencia a tracción del hormigón.

Las solicitaciones aplicadas a la sección de estudio dan lugar a un plano de deformaciones que, a su vez, genera unas tensiones en el material que buscan equilibrar las fuerzas aplicadas. Esto se consigue en caso de que las fuerzas actuantes sean inferiores a las de rotura, valores que son determinados por los dominios de deformación de agotamiento representados en la figura 3.7.

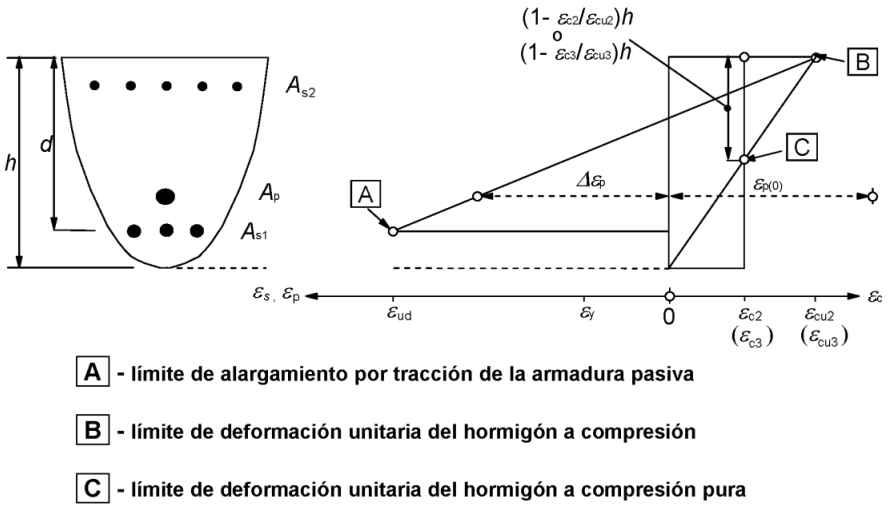


Figura 3.7: Dominios de deformación en estado límite último (Figura 6.1 norma UNE-EN 1992-1-1 [98]).

Las tensiones generadas por el plano de deformación sobre los materiales en cada uno de los puntos de la sección son función de la distancia entre este y la fibra neutra. Por ello, cuanto más alejado esté el punto de estudio de la fibra neutra, mayor será la deformación respecto al plano no deformado y, como consecuencia, las tensiones serán de mayor magnitud. Es por esto que cada plano de deformación entonces puede definirse mediante una única variable, la profundidad de la fibra neutra.

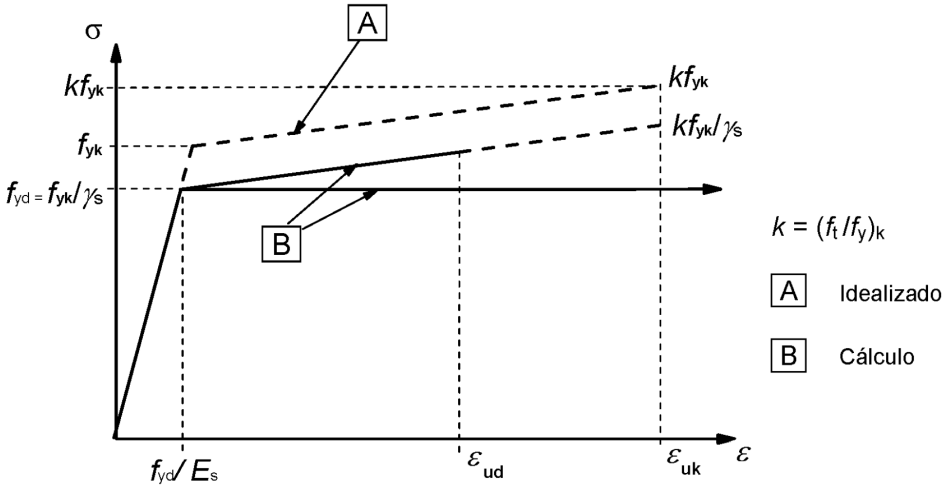
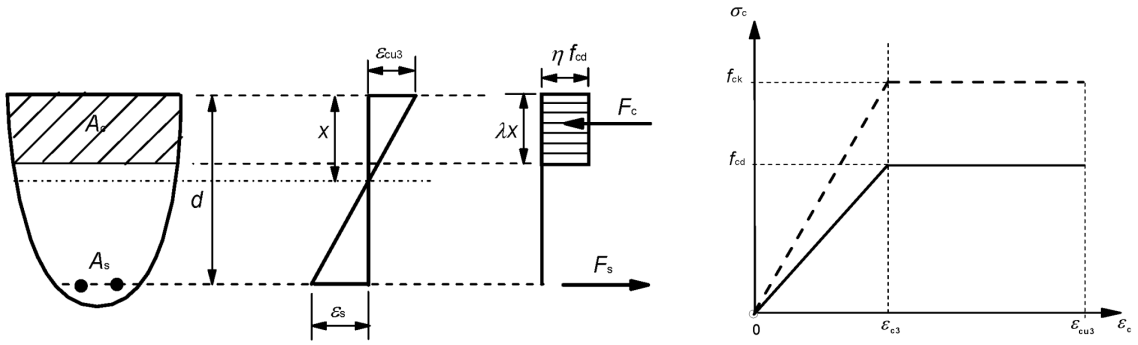


Figura 3.8: Diagrama tensión-deformación de cálculo de las armaduras pasivas (Figura 3.8 norma UNE-EN 1992-1-1 [98]).

En el caso de las armaduras pasivas, el cálculo tensional considera el diagrama representado en la figura 3.8. Cada barra se puede tratar como un elemento puntual cuya tensión es la asociada a la deformación en el punto medio de su sección. Mediante el sumatorio de las deformaciones de cada una de las barras de armadura de la sección, multiplicadas por su tensión, se obtiene el axil de la armadura. De igual forma, si cada uno de los axiles es multiplicado por la distancia hasta el eje de la sección, se obtiene el momento asociado a las armaduras.

En el caso del hormigón, la norma UNE-EN 1992-1-1 establece, mediante el uso del diagrama parábola rectángulo, la forma adecuada de considerar el hecho de que se trata de un elemento no puntual. Por ser continuo, tiene diferentes deformaciones en cada uno de los puntos de la sección, algo que solventa la integración del ya mencionado diagrama parábola rectángulo en toda la sección.



(a) Diagrama parábola rectángulo. (b) Diag. tensión deformación.

Figura 3.9: Diagramas para el cálculo del hormigón (Figura 3.5 (a) y Figura 3.4 (b) norma UNE-EN 1992-1-1 [98]).

De esta forma, se puede considerar un diagrama bilineal de tensión deformación cuyos puntos clave se corresponden con las deformaciones unitarias que definen, en conjunto con las del acero, cada uno de los dominios de deformación.

Mediante el cálculo del axil y momento últimos de la sección, como suma de los asociados a las armaduras y el hormigón, se puede determinar de forma completa el diagrama de interacción. Una vez hecho esto, el proceso de comprobación del estado limite queda reducido a verificar que el punto determinado por el axil y momento flector actuantes, una vez aplicado el decalaje pertinente, queda contenido dentro del diagrama de la sección. En caso de ser así, esta verifica el estado límite frente

a solicitaciones normales. La figura 3.10 presenta, a modo de ejemplo, uno de los diagramas de interacción de las secciones del marco obtenido mediante el software desarrollado.

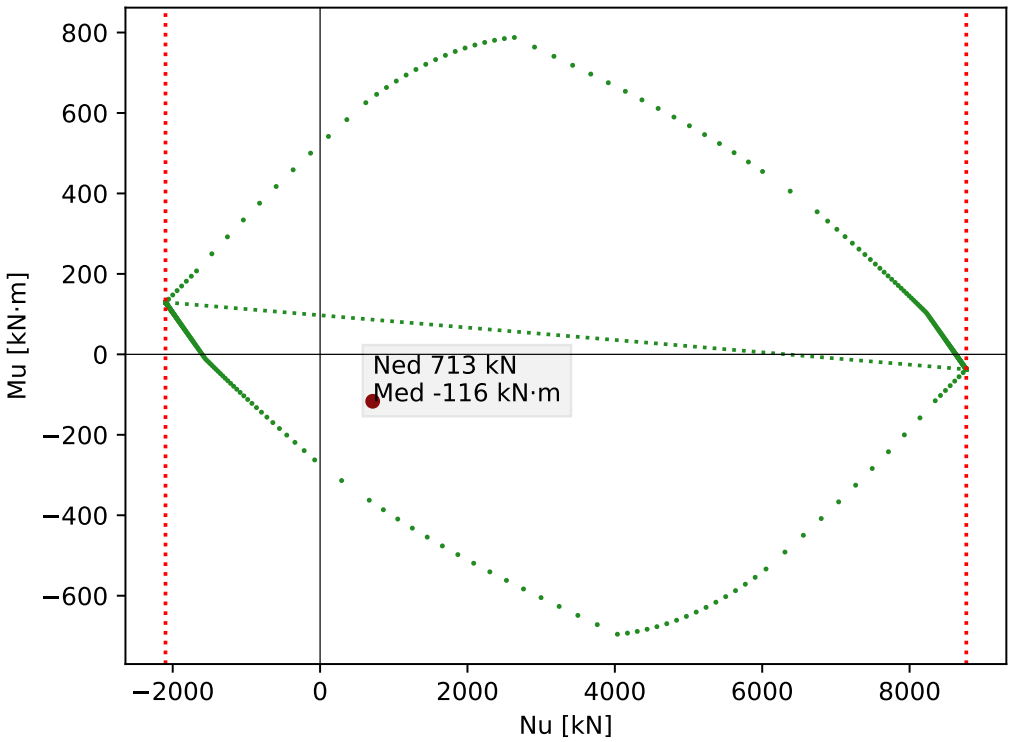


Figura 3.10: Diagrama de interacción de una sección del marco prefabricado obtenido mediante el software desarrollado (Elaboración propia).

3.4.3.3. Estado Límite de Servicio de fisuración

Una vez establecido el procedimiento a seguir para la comprobación de los estados límite último del marco, se da paso a la explicación del procedimiento seguido en la comprobación del estado límite considerado. En el caso del marco, la verificación del estado límite de fisuración cuenta con una serie de consideraciones diferenciadas.

En primer lugar se han de considerar las limitaciones tensionales impuestas a los materiales. Estas buscan evitar la aparición de fisuras estableciendo unos valores

de tensiones límite que no han de ser superados. En función del tipo de fisura cuya aparición se trata de evitar, se imponen diferentes limitaciones. En el caso del marco prefabricado estas limitaciones son las siguientes.

- Longitudinales y de compresión, limitando las tensiones en el hormigón bajo la combinación de estado límite de servicio característica.

$$\sigma_{c,max}(N_{car.}, M_{car.}) \leq 0{,}6 \cdot f_{ck} \tag{3.17}$$

- Tracción en las armaduras, buscando mantener una apariencia aceptable establece un límite en la tensión de las armaduras pasivas.

$$|\sigma_{s,max}(N_{car.}, M_{car.})| \leq |0{,}8 \cdot f_{syk}| \tag{3.18}$$

La comprobación del cumplimiento de estas limitaciones pasa pues, por el cálculo de las tensiones en cada elemento de la sección. Partiendo de lo establecido en la verificación del estado límite frente a solicitaciones normales, este cálculo tensional conlleva, en primer lugar, la determinación de la profundidad de la fibra neutra de la sección (X).

$$X = n \cdot \rho_1 \cdot \left(1 + \frac{\rho_2}{\rho_1}\right) \cdot \left(-1 + \sqrt{1 + 2 \cdot \frac{1 + \frac{\rho_2 \cdot d'}{\rho_1 \cdot d}}{n \cdot \rho_1 \cdot \left(1 + \frac{\rho_2}{\rho_1}\right)^2}}\right) \cdot d \tag{3.19}$$

Donde

$n = \dfrac{E_s}{E_c}$ la relación entre los módulos elásticos de acero y hormigón;

$d = h - r_{nom} - \dfrac{\phi}{2}$ el canto útil de la sección;

$d' = h - d \cdot d$ el recubrimiento de la armadura pasiva;

$\rho_1 = \dfrac{A_{s1}}{b \cdot d}$ la cuantía de armadura longitudinal A_{s1};

$\rho_2 = \dfrac{A_{s2}}{b \cdot d}$ cuantía de armadura longitudinal A_{s2}.

Una vez determinada la profundidad de la fibra neutra X para la sección, la tensión del hormigón puede calcularse mediante la siguiente expresión.

$$\sigma_c = M_k \cdot \frac{X}{I_{cr}} \tag{3.20}$$

Siendo necesario antes calcular el momento de fisuración de la sección ($M_{fis} = M_k$), así como la inercia de la sección fisurada (I_{cr}).

$$M_k = f_{ctm} \cdot I_b \cdot \frac{2}{h} \tag{3.21}$$

$$I_{cr} = n \cdot A_{s1} \cdot (d - X) \cdot \left(d - \frac{X}{3}\right) + n \cdot A_{s2} \cdot (X - d') \cdot \left(\frac{X}{3} - d'\right) \tag{3.22}$$

Pudiendo entonces comprobar la verificación de la limitación de tensiones de compresión sobre el hormigón.

$$\sigma_c = M_k \cdot \frac{X}{I_{cr}} \leq 0{,}6 \cdot f_{ck} \tag{3.23}$$

Tras esto, la tensión de las armaduras pasivas de la sección puede ser calculada aplicando la expresión.

$$\sigma_{s1} = n \cdot \sigma_c \cdot \frac{d - X}{X} \tag{3.24}$$

Pudiendo de nuevo verificar de forma directa el cumplimiento de la limitación tensional en las armaduras.

$$\sigma_{s1} = n \cdot \sigma_c \cdot \frac{d - X}{X} \leq 0{,}8 \cdot f_{yk} \tag{3.25}$$

Una vez verificadas las limitaciones tensionales, el procedimiento de comprobación del estado límite de fisuración consiste en el cálculo de la abertura de fisura (W_k) de la sección para, posteriormente, compararlo con el valor máximo estipulado en norma.

La abertura de fisura se calcula como el múltiplo de separación entre fisuras y la diferencia entre los alargamientos unitarios medios del acero y hormigón.

$$W_k = S_{r,max} \cdot (\varepsilon_{sm} - \varepsilon_{cm}) \tag{3.26}$$

Donde

$S_{r,max}$ la separación máxima entre fisuras;

ε_{sm} el alargamiento unitario medio de la armadura;

ε_{cm} el alargamiento unitario medio del hormigón.

El primer término, correspondiente a la separación máxima entre fisuras puede calcularse mediante la siguiente expresión.

$$S_{r,max} = k_3 \cdot c + k_1 \cdot k_2 \cdot k_4 \cdot \frac{\phi}{\rho_{p,eff}} \tag{3.27}$$

Donde

c es el recubrimiento de la armadura longitudinal;

$k_1 = 0{,}8$ para barras de alta adherencia;

$k_2 = 0{,}5$ para cálculo en flexión;

$k_3 = 3{,}4$ valor de uso en España;

$k_4 = 0{,}425$ valor de uso en España;

ϕ es el diámetro de la barra;

$\rho_{p,eff}$ definido a continuación en el cálculo del segundo término.

El segundo término, correspondiente a la diferencia de alargamientos unitarios medios puede obtenerse mediante la siguiente expresión.

$$(\varepsilon_{sm} - \varepsilon_{cm}) = \frac{\sigma_s - k_t \cdot \frac{f_{ct,eff}}{\rho_{p,eff}} \cdot (1 + \alpha_e \cdot \rho_{p,eff})}{E_s} \geq 0{,}6 \cdot \frac{\sigma_s}{E_s} \tag{3.28}$$

Donde

σ_s la tensión de la armadura de tracción con la sección fisurada;

α_e es la relación entre módulos elásticos E_s/E_{cm};

$$\rho_{p,eff} = \frac{A_s + \xi_1 \cdot A'_p}{A_{c,eff}} \text{ donde}$$

A_p área de armadura en $A_{c,eff}$;

$A_{c,eff}$ es el área eficaz de hormigón en tracción;

$\xi_1 = \sqrt{\xi \cdot \frac{\phi_s}{\phi_p}}$;

$k_t = 0{,}4$ coeficiente para carga a largo plazo.

De esta forma, mediante el cálculo de la abertura de fisura de la sección, y posterior comparación con los valores tabulados en norma para diferentes condiciones, se puede verificar de forma completa el estado límite de servicio. El procedimiento completo se muestra en el diagrama correspondiente a la figura 3.11.

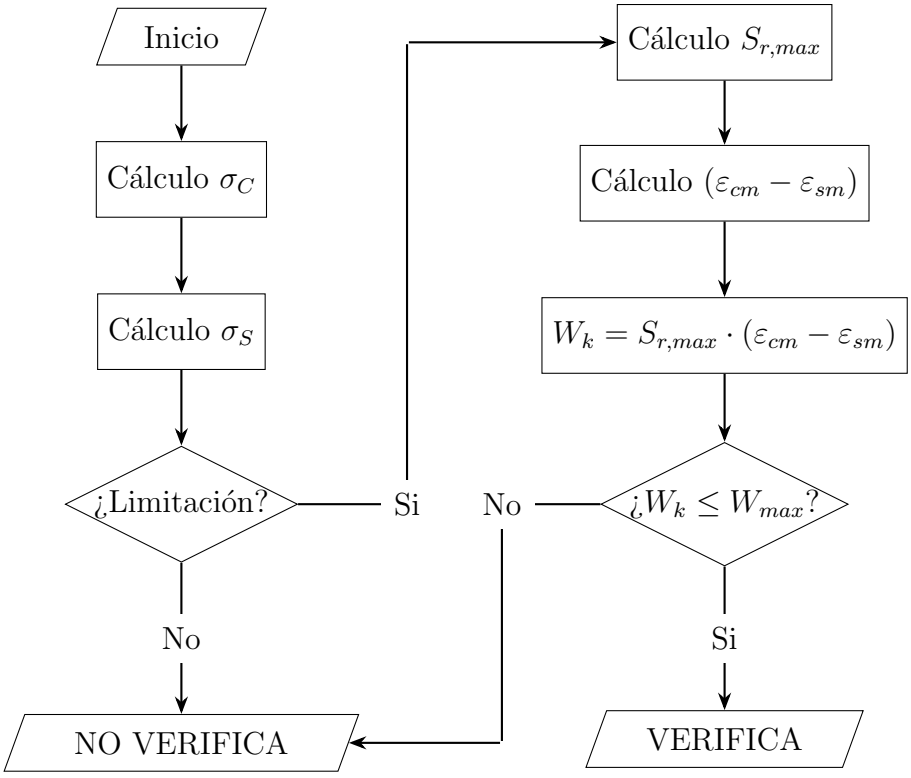


Figura 3.11: Diagrama del proceso de verificación del estado límite de servicio de fisuración (Elaboración propia).

Finalmente, es necesario evaluar el cumplimiento de la armadura mínima para control de fisuración en la estructura, para ello se hace uso de la siguiente expresión.

$$A_{s,min} \cdot \sigma_s = k_c \cdot k \cdot f_{ct,eff} \cdot A_{ct} \tag{3.29}$$

Donde

σ_s la tensión de la armadura f_{yk};

$k = 1$ en el caso del marco;

$f_{ct,eff}$ resistencia media a tracción en la aparición de fisura;

A_{ct} área de hormigón dentro de la zona de tracción;

$k_c = 0{,}4 \cdot \left(1 - \dfrac{\sigma_c}{k_1 \cdot (h/h^*) \cdot f_{ct,eff}}\right) \leq 1$ donde

σ_c tensión calculada en el hormigón;

k_1 coeficiente efecto de fuerzas axiles;

$h^* = h$ si $h < 1m$;

$h^* = h$ si $h \geq 1m$;

$f_{ct,eff}$ ya descrita.

De esta forma, mediante la comprobación de las limitaciones tensionales en el hormigón y las armaduras, así como la comprobación de la abertura mediante comparación con la abertura máxima, en conjunto con la verificación del armado mínimo para control de fisuración, el estado límite de servicio de fisuración es verificado para cada una de las secciones del marco.

3.4.3.4. Disposiciones relativas a armaduras

Dando fin a las comprobaciones necesarias para la verificación de cada una de las secciones de interés del marco prefabricado, es necesario considerar las cuantías máximas y mínimas estipuladas en norma. Esto comprende las siguientes verificaciones referentes a losas macizas y muros, contenidas en los artículos 7.3.2, 9.4 y 9.6

del Eurocódigo 2 [98]:

- Armadura longitudinal de tracción mínima. Queda determinado por el máximo de los valores derivados del cálculo de armadura mínima longitudinal, y armadura mínima para el control de fisuración.

$$A_{s_{min}} = max(A_{s_{min,fis}}, A_{smin,long}) \tag{3.30}$$

  El cálculo del área mínima para control de fisuración fue detallado en el apartado anterior, y el área mínima longitudinal se obtiene mediante la expresión de uso en España.

$$A_{s_{min,long}} = \frac{W}{z} \cdot \frac{f_{ctm,fl}}{f_{yd}} \tag{3.31}$$

- Armadura longitudinal máxima. El área de armadura longitudinal no ha de ser superior al cuatro por ciento del área de la sección.

$$A_{s_{max}} \leq 0{,}04 \cdot A_{sec} \tag{3.32}$$

- Separación máxima entre barras de armadura longitudinal. La distancia entre barras contiguas ha de ser suficientemente grande como para permitir un vertido y compactado adecuado, sin embargo está limitado por la siguiente expresión.

$$s_{max} = min(400mm, 3h) \tag{3.33}$$

- Armadura mínima de cortante. Según lo estipulado en el artículo 9.2.2 de la norma, la cuantía de armadura de cortante mínima se determina aplicando la siguiente expresión.

$$\frac{A_{sw,min}}{s} = \rho_{min} \cdot b_w \cdot sen(\alpha) \tag{3.34}$$

  Donde

$$\rho_{min} = \frac{f_{ctm}}{7{,}5 \cdot f_{yk}}$$

- Separación máxima entre cercos. Además de la necesidad de cumplir la cuantía mínima de armadura de cortante, se establece la separación máxima longitu-

dinal entre estribos mediante la expresión.

$$s_{l,max} = 0{,}75 \cdot d \cdot (1 + cot(\alpha)) \tag{3.35}$$

- Armadura transversal mínima. Las losas y muros del marco han de contar con una armadura transversal mínima, siendo su área función de las armaduras longitudinales dispuestas. De esta forma, la presencia de armadura transversal no conforma parte de la verificación del problema, su magnitud es calculada en función del diseño de armado y considerada en la función objetivo del problema.

3.5. Función objetivo

El coste del marco se evalúa mediante el sumatorio de los costes unitarios de cada uno de los materiales que lo conforman, multiplicados por las mediciones correspondientes a cada uno de ellos.

$$C_{total} = M_H \cdot P_H + M_S \cdot P_S \tag{3.36}$$

Donde

M_H es la medición de hormigón en m^3;
M_S es la medición de acero en kg;
P_C es el precio unitario del hormigón en €/m^3;
P_S es el precio unitario del acero en €/kg.

Considerando el alcance del estudio, se evalúa el coste del marco como función de la cantidad de hormigón y acero usados en su construcción, la naturaleza prefabricada de la estructura permite obviar costes de encofrado, además, la consideración del hecho de que el transporte hasta obra es independiente de la optimización del diseño del marco, permite también obviar ese coste asociado a la construcción.

Cada uno de los precios unitarios considerados en la optimización del marco fue obtenido de la base de datos BEDEC [100], estos quedan contenidos en la tabla 3.5.

Unidad	Descripción	Coste unitario (€)
m^3	Hormigón HA-25	88.86
m^3	Hormigón HA-30	97.80
m^3	Hormigón HA-35	101.83
m^3	Hormigón HA-40	104.08
kg	Acero B-400S	1.40
kg	Acero B-500S	1.42

Tabla 3.5: Precios unitarios considerados en el problema (BEDEC [100]).

De esta forma, el problema de optimización a abordar queda definido de forma completa, en este capítulo se han presentado los parámetros, variables y restricciones que lo definen, así como la función objetivo considerada.

3.6. Dimensión del espacio de soluciones

Considerando los posibles valores que pueden adoptar cada una de las 31 variables de diseño del problema, el número total de soluciones, y consecuentemente la dimensión del espacio de soluciones, es el siguiente.

$$n_{hh} \cdot n_{hls} \cdot n_{hli} \cdot n_{\phi_{Ai}} \cdot n_{n_{Ai}} \cdot n_{phi_{Aj}} \cdot n_{s_{Aj}} \cdot n_{concrete} \cdot n_{steel} \tag{3.37}$$

$$41 \cdot 41 \cdot 41 \cdot 41 \cdot 6^8 \cdot 9^8 \cdot 6^3 \cdot 10^3 \cdot 7^4 \cdot 4 \cdot 2 \approx 2{,}0675 \times 10^{28}$$

Considerando que la comprobación de cada una de las soluciones toma, de forma aproximada, un tiempo de computación de 0,33 segundos, sería necesario un periodo de $2{,}185 \times 10^{21}$ milenios. Esto hace que sea evidente concluir que, con los medios computacionales actuales, la verificación del espacio de soluciones al completo no es una opción viable.

Capítulo 4

Aplicación de los métodos heurísticos seleccionados

En el capítulo anterior se hizo una definición completa de los parámetros, variables, y simplificaciones consideradas en la resolución del problema de optimización. Establecido esto, en este capítulo se detalla el fundamento teórico tras el software de cálculo estructural desarrollado, así como sus características y limitaciones.

Además, se muestra el procedimiento seguido en el proceso de calibración de los parámetros de cada uno de los algoritmos genéticos aplicados en la resolución del problema de optimización. Tras esto, como final del capítulo, se describen los pasos seguidos en el proceso de obtención de los resultados que se presentan y analizan en los siguientes capítulos.

4.1. Recocido simulado (*SA*)

4.1.1. Introducción

En el segundo capítulo del trabajo se detalló el funcionamiento general de un algoritmo SA, además se presentaron una serie de trabajos en los que su aplicación ha conseguido resultados de alta calidad. En este contexto, un algoritmo genético de recocido simulado fue la primera de las técnicas heurísticas aplicadas en la resolución del problema de optimización económica del marco prefabricado. A continuación se

describe el funcionamiento concreto del algoritmo SA utilizado en el estudio.

4.1.2. Aplicación del algoritmo *SA*

La versión de algoritmo SA implementada en este trabajo parte de una solución inicial factible S_0, generada de forma aleatoria, para la que se evalúa el coste económico $f(S_0)$. A partir de esta solución, se inicia el proceso de establecimiento y adecuación de la temperatura inicial del proceso.

Para determinar la temperatura inicial se hizo uso del método propuesto por Medina [101], dado que se trata de un procedimiento sencillo y respaldado por los buenos resultados obtenidos en trabajos con objetivos similares a este [13, 82, 102]. El procedimiento en cuestión parte de una temperatura inicial arbitraria para luego corregir su valor durante una serie de cadenas de Markov.

La corrección se aplica en función de la tasa de aceptación ($A_{Boltzmann}$) de respuestas de la cadena mediante el factor de Boltzmann $\left(e^{-(\Delta E/T)}\right)$. Es decir, la relación entre el número de soluciones aceptadas con un coste superior a la anterior, y el número total de soluciones con un coste superior evaluadas mediante la comparación de dicho factor y un número aleatorio de una distribución $U(0,1)$.

$$A_{Boltzmann} = \frac{(S_1)\ con\ (\Delta E > 0)\ aceptadas}{Total\ de\ (S_1)\ con\ (\Delta E > 0)\ evaluadas} \cdot 100 \tag{4.1}$$

Si tras la cadena de Markov esta tasa se sitúa entre dos límites, superior e inferior, fijados anteriormente, el valor de la temperatura inicial T_0 es aceptado. En caso de que la tasa esté por debajo del límite inferior, establecido en la aceptación de un 20 % de soluciones de peor calidad, proceso está "frío", y el valor de la temperatura se multiplica por dos. En caso opuesto, si la tasa de aceptación está por encima del límite superior, establecido en un 40 %, el problema está demasiado "caliente" por lo que el valor de la temperatura se divide a la mitad. Tras una serie de pruebas iniciales, se establece que la temperatura de inicio arbitraria tenga una magnitud igual al 5 % del coste de la solución inicial generada de forma aleatoria.

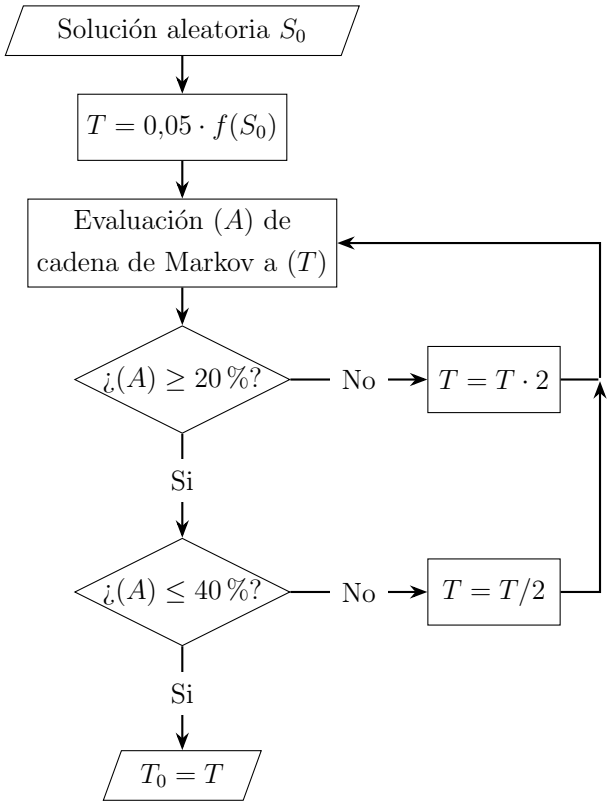


Figura 4.1: Proceso de establecimiento de la temperatura inicial del algoritmo de recocido simulado programado (Elaboración propia basado en Medina et al. [101]).

El coeficiente de enfriamiento k tiene un valor constante inferior a la unidad, su aplicación tras cada una de las cadenas de Markov da lugar a una reducción geométrica de la temperatura del problema. El valor de este coeficiente, junto con la longitud de la cadena de Markov, ha sido objeto de estudio durante el desarrollo del algoritmo. Una correcta calibración de estos parámetros permite obtener resultados de mejor calidad.

En lo que al fin del algoritmo respecta, se establecen dos criterios diferenciados que determinan el final del proceso de optimización. El primero de ellos es el criterio de parada, este consiste en el establecimiento de un número máximo de iteraciones en las que el algoritmo no consigue mejorar la solución óptima actual. Además de esto, se establece un criterio de terminación del algoritmo, este establece el final del proceso en el momento que la temperatura del problema sea igual o inferior a un determinado porcentaje de la temperatura inicial.

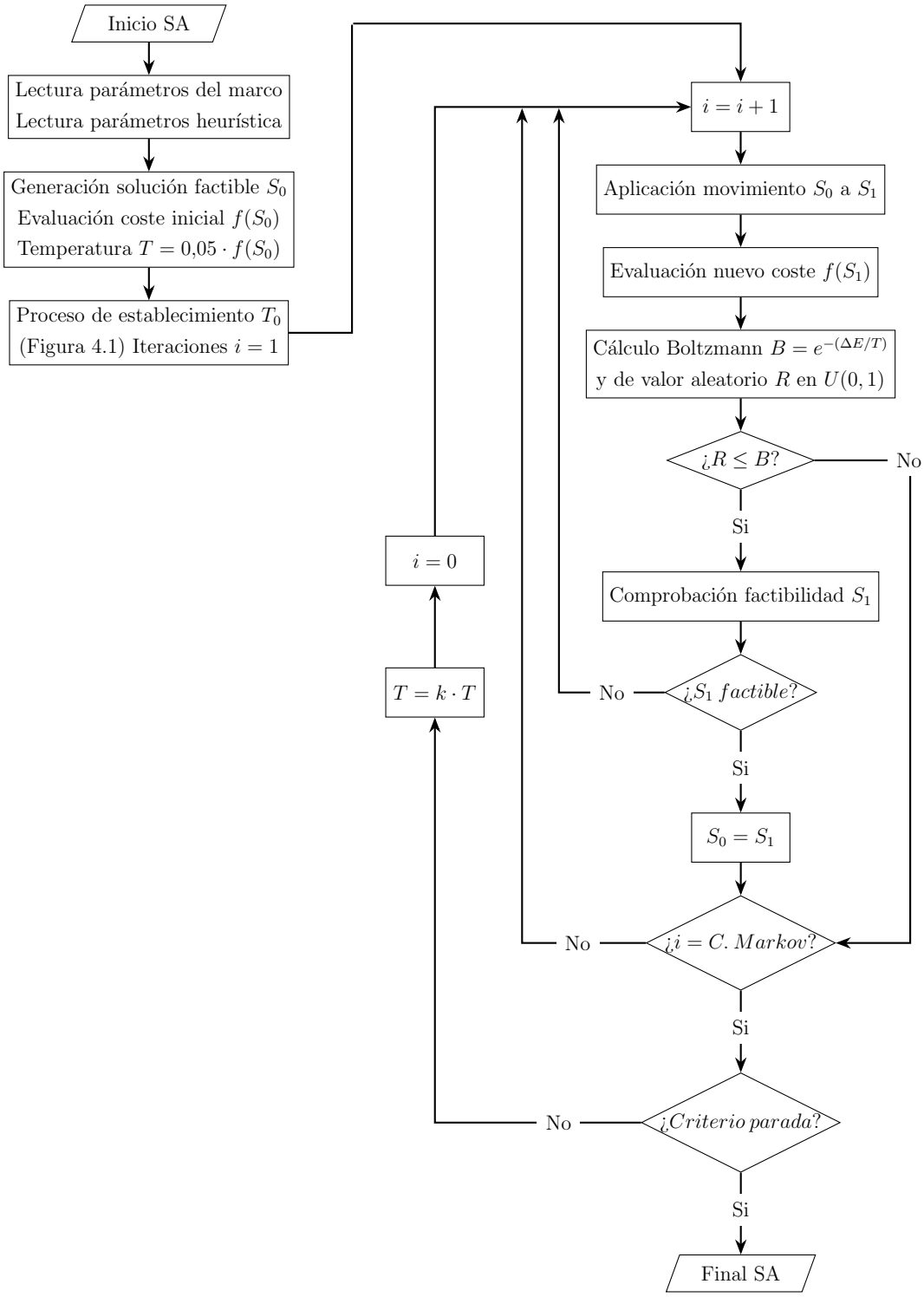


Figura 4.2: Funcionamiento interno del algoritmo SA programado (Elaboración propia).

La figura 4.2 muestra el funcionamiento interno del algoritmo programado para la resolución del problema.

4.1.3. Calibración de parámetros SA

Una vez descrito el algoritmo SA concreto programado para la resolución del problema, este apartado presenta el proceso de calibración de los parámetros de la heurística. Una correcta calibración del algoritmo permitirá que este presente dos fases diferenciadas. La primera de ellas es la **fase de diversificación**, momento en el que la temperatura del problema es suficientemente alta como para asegurar que el algoritmo pueda explorar el espacio de soluciones evitando así incurrir en una convergencia temprana, algo que puede dar lugar a la aceptación de un óptimo de baja calidad. Tras esta fase, conforme la temperatura del problema desciende, el algoritmo ha de centrar sus esfuerzos en soluciones similares, siendo menos probable la aceptación de nuevas propuestas que empeoren de forma considerable la actual. Este periodo es la denominada **fase de intensificación** del algoritmo.

Tras las primeras pruebas del algoritmo, el criterio de parada fue el primero de los parámetros del SA que requirió ser ajustado. En un inicio se estableció un número máximo de dos cadenas de Markov sin mejora, sin embargo, esto daba lugar a una finalización demasiado temprana del proceso, evitando que el algoritmo pudiera alcanzar la fase de intensificación, no llegando a superar la fase de diversificación inicial. Con el objetivo de solventar este problema, se establece un nuevo criterio de parada, combinado con un criterio de terminación. Este consiste en permitir que la heurística pueda detenerse tras un máximo de cinco cadenas de Markov sin mejora consecutivas, siempre que la temperatura sea suficientemente baja. Por otra parte, se establece la terminación del proceso en el momento que la temperatura del problema sea igual o inferior a un cinco por ciento de la temperatura inicial, considerando como tal la temperatura ajustada tras el proceso de adecuación de la temperatura inicial descrito en el inicio del apartado y desarrollada por Medina [101].

De esta forma, gracias a los criterios de parada y terminación establecidos, se consigue que el algoritmo SA pueda superar la fase de diversificación inicial. Además, este cuenta con fases de intensificación suficientemente prolongadas como para asegurar la convergencia del problema, sin ser, por el contrario, excesivamente largas. La

figura 4.3 muestra el proceso genérico obtenido en una ejecución normal del algoritmo de recocido simulado considerado.

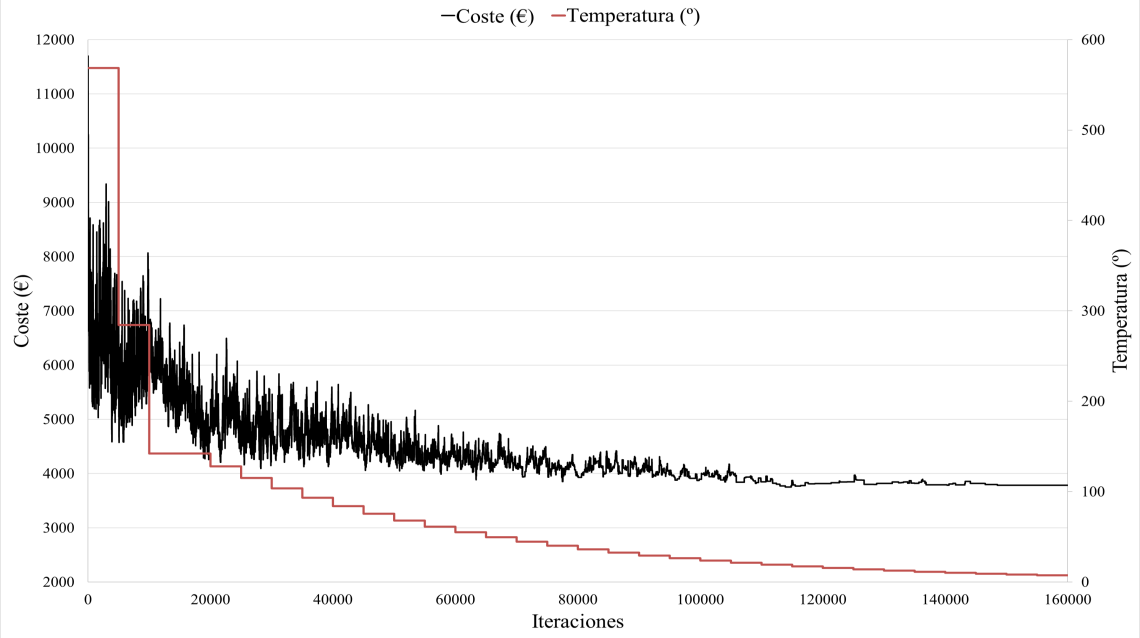


Figura 4.3: Evolución de una ejecución genérica del algoritmo SA programado (Elaboración propia).

En esta se ve, de forma clara, el proceso de establecimiento, y consecuente ajuste, de la temperatura inicial del problema. Además, puede apreciarse la convergencia del algoritmo desde la fase inicial donde existe una considerable aceptación de soluciones de peor calidad, siendo este tipo de aceptaciones cada vez menos común conforme se desarrolla el proceso. La fase inicial de diversificación finaliza al acercarse a las 12×10^4 iteraciones, dando comienzo a la fase de intensificación final del problema.

Una vez establecidos los parámetros mencionados, es necesario establecer el tipo de movimiento que permite al algoritmo desplazarse desde la solución actual hasta la nueva solución. En este caso se ha considerado que cada movimiento puede modificar un máximo de cinco de las treinta y una variables del problema. De forma previa al movimiento, se selecciona de forma aleatoria el número de variables que van a ser modificadas, siendo siempre igual o inferior a cinco. Tras esto, las variables a modificar son seleccionadas de forma aleatoria entre el conjunto, tras la selección, su valor sube o baja, de forma arbitraria, un escalón dentro del rango de valores

establecidos previamente.

Finalmente, los parámetros del algoritmo SA por definir son la longitud de cadena de Markov, así como el coeficiente de enfriamiento k. Con el objetivo de conseguir un rango amplio de respuestas por parte del algoritmo, se considera la aplicación de diferentes valores que permitieron la obtención de resultados de alta calidad en el estudio de diversas estructuras [12, 14, 19]. De esta forma, los parámetros de la heurística varían entre 0,80 y 0,95 para el coeficiente de enfriamiento k, y entre 500 y 5,000 para la longitud de la cadena de Markov. Los resultados obtenidos, en conjunto con su análisis y comentario son presentados en el siguiente capítulo.

4.2. Threshold accepting (TA)

4.2.1. Introducción

Tras la consideración inicial del algoritmo de recocido simulado, un algoritmo genético de aceptación por umbrales fue la segunda de las técnicas heurísticas aplicadas en la resolución del problema de optimización económica del marco prefabricado. A continuación se describe el funcionamiento concreto del algoritmo TA utilizado en el estudio.

4.2.2. Aplicación del algoritmo TA

El algoritmo TA considerado en la resolución del problema comparte muchas de las características presentadas para el algoritmo SA aplicado inicialmente. Esto comprende tanto el proceso de adecuación de la temperatura inicial, siendo en este caso el umbral inicial, así como el consecuente proceso de descenso geométrico del umbral a razón de un coeficiente de reducción k.

Como se describió en el segundo capítulo del trabajo, la principal diferencia entre el algoritmo SA y el TA es el criterio de aceptación de soluciones que empeoran la actual, siendo una aceptación de tipo determinístico en el caso del TA.

En este contexto, el algoritmo TA parte de una solución inicial factible S_0 generada de forma aleatoria, cuyo coste $f(S_0)$ es evaluado para establecer un umbral inicial arbitrario que, de forma análoga a lo presentado para el algoritmo SA, es co-

rregido durante una serie de cadenas de Markov en función de la tasa de aceptación de soluciones de menor calidad del algoritmo.

Este proceso puede observarse en el diagrama de flujo del proceso de establecimiento del umbral inicial representado en la figura 4.4.

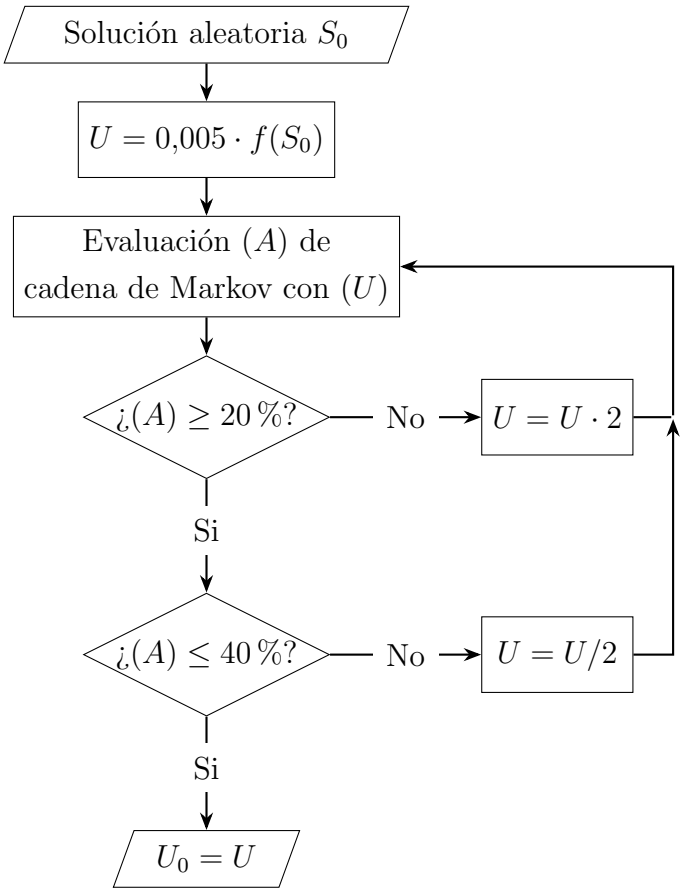


Figura 4.4: Proceso de establecimiento del umbral inicial del algoritmo de aceptación por umbrales programado (Elaboración propia basado en Medina et al. [101]).

El coeficiente de reducción del umbral k tiene un valor constante inferior a la unidad, su aplicación tras cada una de las cadenas da lugar a una reducción geométrica del umbral de aceptación del algoritmo. De forma similar al caso del algoritmo SA, el valor de este coeficiente, en conjunto con la longitud de la cadena es objeto de calibración y se adoptaron diferentes valores en la resolución del problema.

La figura 4.5 muestra el funcionamiento interno del algoritmo programado para la resolución del problema.

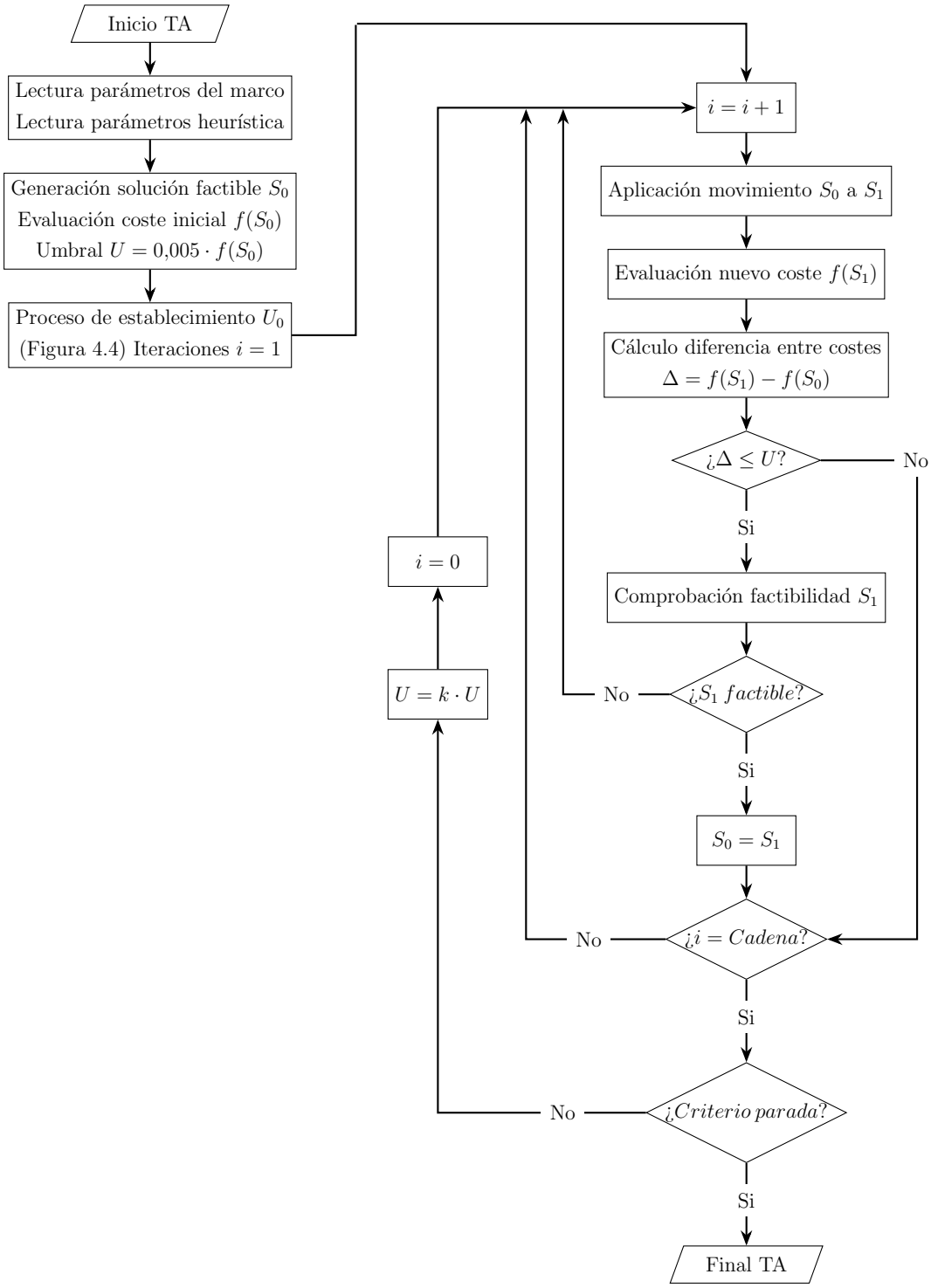


Figura 4.5: Funcionamiento interno del algoritmo TA programado (Elaboración propia).

4.2.3. Calibración de parámetros TA

Una vez establecido el funcionamiento concreto del algoritmo TA programado, el siguiente paso es la calibración de los parámetros de la heurística. Este proceso comprende la adecuación del umbral inicial, así como de los criterios de parada del algoritmo.

Tras una serie de ejecuciones iniciales se establece que el umbral inicial del algoritmo sea equivalente a un diez por ciento del coste del marco inicial factible generado de forma aleatoria. Partiendo de ese umbral, de forma análoga al proceso presentado en en algoritmo SA, propuesto por Medina [101], este umbral es adecuado al problema concreto durante las cadenas que sean necesarias.

Al igual que en la calibración de la heurística anterior, tanto la longitud de cadena como el coeficiente de reducción de umbral son modificados con el objetivo de establecer los valores más adecuados para la optimización económica del marco. Se adoptan pues, valores que permitieron la obtención de resultados de alta calidad en el estudio de diversas estructuras [12, 14, 19]. En este caso la longitud de cadena varía entre las 5,000 y 500 iteraciones, y el coeficiente de reducción de umbral entre 0,80 y 0,95, valores similares a los probados en la resolución mediante el algoritmo SA.

En vista de lo probado con el ya mencionado algoritmo de recocido simulado, se establecen criterios de parada y terminación iguales a los desarrollados en el apartado anterior. Además de esto, es relevante mencionar que el movimiento considerado para el algoritmo de aceptación por umbrales tiene un funcionamiento análogo al del algoritmo de recocido simulado.

De esta forma, habiendo calibrado los criterios de parada y terminación del algoritmo TA, en conjunto con el valor arbitrario del que parte el proceso de establecimiento del umbral inicial, además de la completa definición del movimiento que permite la obtención de nuevas soluciones, el algoritmo queda completamente definido. Siendo la longitud de cadena, así como el coeficiente de reducción de umbral, parámetros de la heurística objeto de prueba y estudio en la resolución del problema.

La figura 4.6 presenta la evolución de una ejecución genérica del algoritmo TA desarrollado, en esta se distingue de forma clara el proceso de establecimiento del umbral inicial. Proceso tras el cual tiene lugar la fase de exploración y consecuente

convergencia hacia soluciones que cada vez son más similares entre sí.

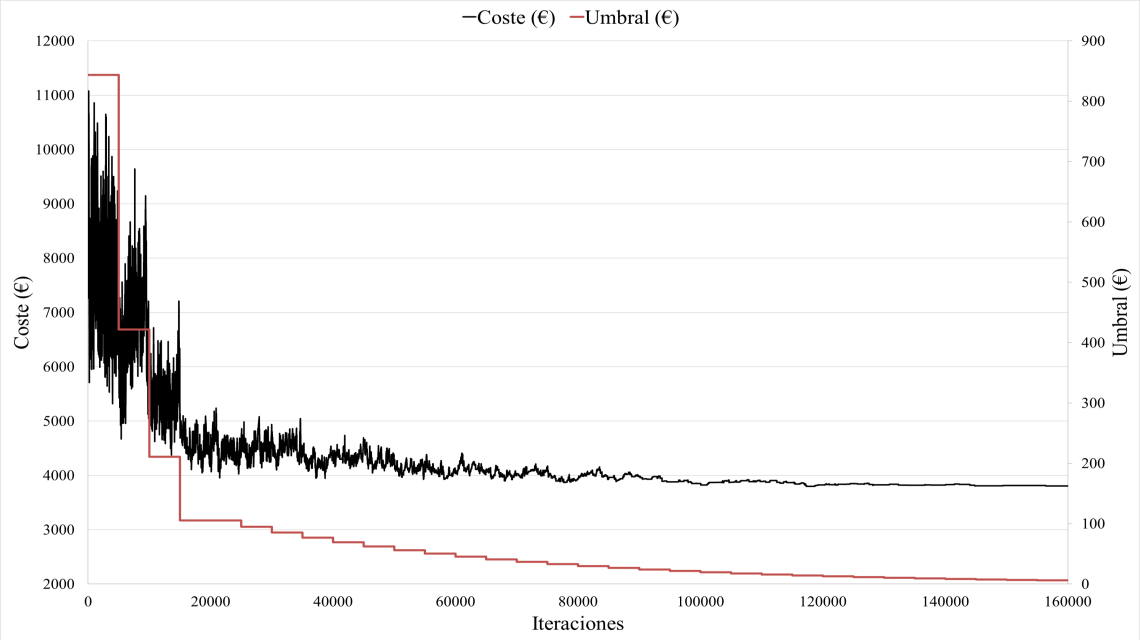


Figura 4.6: Evolución de una ejecución genérica del algoritmo TA programado (Elaboración propia).

4.3. Old bachelor acceptance (OBA)

4.3.1. Introducción

Dando paso al tercer y último algoritmo genético aplicado en la resolución del problema de optimización económica del marco prefabricado, este apartado está dedicado a presentar las particularidades del algoritmo OBA programado.

4.3.2. Aplicación del algoritmo OBA

Como se mencionó en la descripción general del algoritmo OBA, contenida en el segundo capítulo del trabajo, este comparte ciertas similitudes con las otras dos heurísticas aplicadas en la resolución del problema. El algoritmo parte de una solución inicial factible S_0 generada de forma aleatoria y evalúa su coste $f(S_0)$. Tras esto, se aplica un movimiento de características análogas a las desarrolladas para

los algoritmos SA y TA, para así generar una nueva solución S_1. Durante el proceso de evaluación de la primera solución, el algoritmo parte de un umbral inicial U_0 nulo cuya evolución dependerá de la aceptación o no consideración de las soluciones generadas durante la resolución.

En este contexto, el coste de la nueva solución $f(S_1)$ es evaluado y comparado con el de la solución actual, en caso de que el nuevo coste sea inferior, la solución es aceptada de forma directa. La mejora en el coste de la nueva solución frente a la inicial indica un acercamiento a un óptimo del espacio de soluciones. Es por esto que, tras la aceptación de la nueva solución, el algoritmo reduce el umbral inicial a razón del factor de variación $\Delta_{(-)}$. De esta forma, el umbral U a considerar en la siguiente evaluación de soluciones modifica su valor respecto a la anterior.

Tras la primera iteración, una vez el umbral tiene un valor no nulo, el procedimiento continúa mediante la aplicación de movimientos para la generación de nuevas soluciones. Estas son evaluadas y aceptadas si presentan un coste menor, o si la diferencia entre el nuevo coste y el actual es inferior al umbral actual. Cada vez que una nueva solución es aceptada, el umbral se reduce a razón de $\Delta_{(-)}$.

$$[f(S_1) \leq f(S_0)\, o\, (f(S_1) - f(S_0)) \leq U] \rightarrow [U = U - \Delta_{(-)}] \tag{4.2}$$

Por otra parte, cada vez que una solución que empeora la actual presenta un coste cuya diferencia es superior al umbral del algoritmo en ese momento, la heurística procede a descartar dicha nueva solución. Esta ocurrencia puede indicar que el algoritmo no se encuentra cerca de un óptimo del espacio de soluciones del problema, por lo que es interesante empezar un proceso de exploración mediante el aumento del umbral a razón de $\Delta_{(+)}$.

$$[(f(S_1) - f(S_0)) > U] \rightarrow [U = U + \Delta_{(+)}] \tag{4.3}$$

Este proceso se repite durante un número de iteraciones establecido previamente, de forma que, en una ejecución correcta, el algoritmo alternará procesos de exploración del espacio de soluciones, con periodos de intensificación entorno a un óptimo. El funcionamiento concreto del algoritmo OBA programado para la resolución del problema de optimización económica del marco es detallado en la figura 4.7.

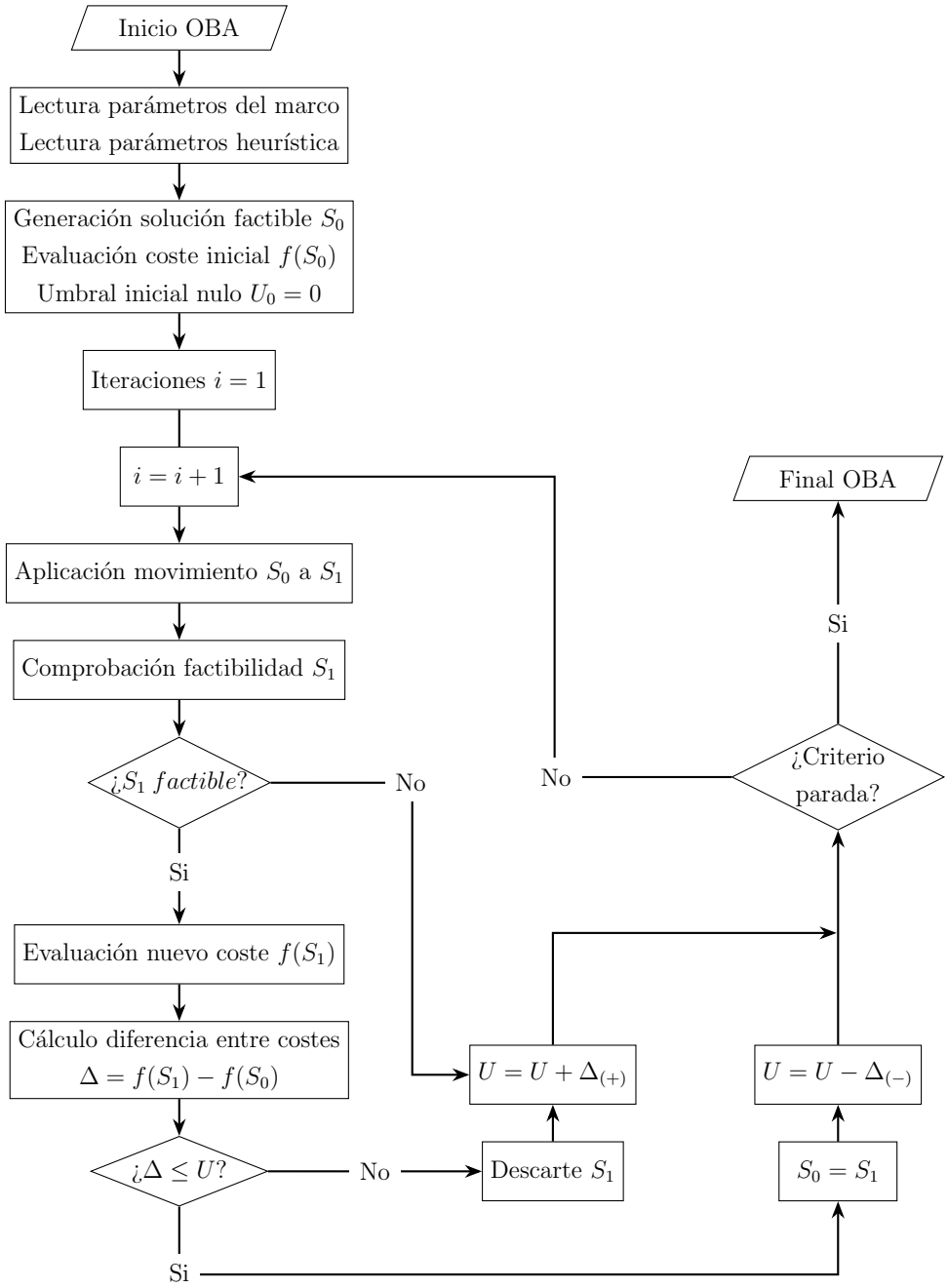


Figura 4.7: Funcionamiento interno del algoritmo OBA programado (Elaboración propia).

4.3.3. Calibración de parámetros OBA

Una vez presentado el funcionamiento concreto del algoritmo OBA desarrollado para la resolución del problema de optimización de interés, se da paso a la calibración de los parámetros de la heurística.

En este caso, el algoritmo OBA no cuenta con un parámetro de control cuyo valor se reduce de forma geométrica a lo largo del proceso. En este contexto, los parámetros objeto de calibración del algoritmo son el criterio de parada, así como los parámetros de modificación del umbral. Considerando el tiempo de computación asociado a la resolución del problema, se establece como criterio de parada del algoritmo un número máximo de iteraciones equivalente a cincuenta mil marcos.

Por otra parte, los valores $\Delta_{(-)}$ y $\Delta_{(+)}$ fueron calibrados mediante una serie de ensayos. Inicialmente se estableció un parámetro de modificación común para el caso de aceptación y no aceptación de la solución equivalente a cinco unidades monetarias.

$$\Delta_{(-)} = \Delta_{(+)} = 5\,€ \tag{4.4}$$

Es decir, el umbral aumentaba o disminuía a razón de ese mismo valor en función de la aceptación o descarte de la nueva solución propuesta.

Tras una serie de reinicios, los resultados obtenidos haciendo uso de esta configuración de parámetros dieron lugar a que fuera descartada. Como puede apreciarse en la figura 4.8, el algoritmo no contaba con la capacidad necesaria para desarrollar de forma adecuada fases de intensificación que le permitieran estudiar óptimos del espacio de soluciones. Esto se debe a que, debido a la magnitud del parámetro $\Delta_{(+)}$, cuando el algoritmo OBA descartaba una solución S_1, o esta no era factible, el umbral aumentaba excesivamente. Esto conducía a series de fases de exploración consecutivas, y la consecuente falta de convergencia del algoritmo.

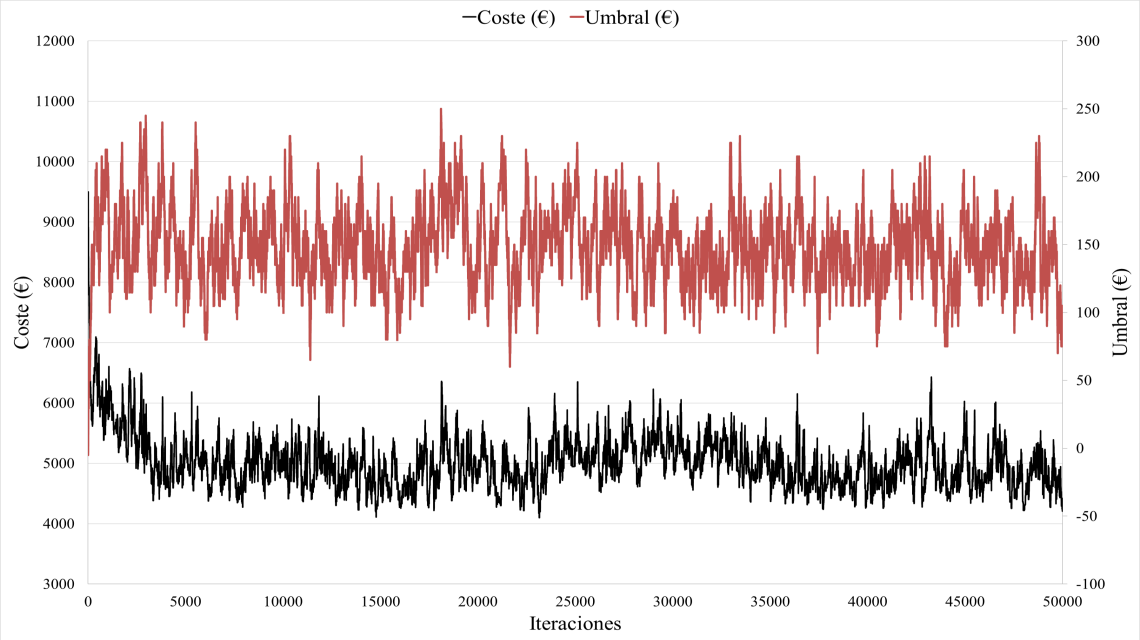


Figura 4.8: Desarrollo genérico del algoritmo OBA sin una convergencia adecuada (Elaboración propia).

Con el objetivo de evitar este tipo de problemas, y poder asegurar la convergencia del algoritmo, se toma la aproximación desarrollada por Alcalá [12]. Esta consiste en la consideración de valores diferentes para los parámetros $\Delta_{(-)}$ y $\Delta_{(+)}$. El fundamento de esta configuración consiste en el hecho de que, si restringimos suficientemente la capacidad del algoritmo para entrar en fases de exploración, el algoritmo tendrá una mayor facilidad para entrar en fases de intensificación y así poder converger adecuadamente.

Esto se consigue mediante la reducción relativa de $\Delta_{(+)}$ respecto a $\Delta_{(-)}$. De esta forma, la calibración del algoritmo OBA da lugar a que la configuración final del mismo cuente con un parámetro $\Delta_{(+)}$ equivalente al veinte por ciento del parámetro $\Delta_{(-)}$.

$$\Delta_{(+)} = 1\,€ \tag{4.5}$$

$$\Delta_{(-)} = 5\,€ \tag{4.6}$$

Haciendo uso de esta configuración el algoritmo OBA desarrollado presenta una mejor convergencia, con fases de intensificación cuya duración es suficiente como

para asegurar el estudio de óptimos del espacio de soluciones. Además, como se puede observar en la figura 4.9, el algoritmo mantiene la capacidad de intercalar fases de exploración e intensificación, convergiendo adecuadamente.

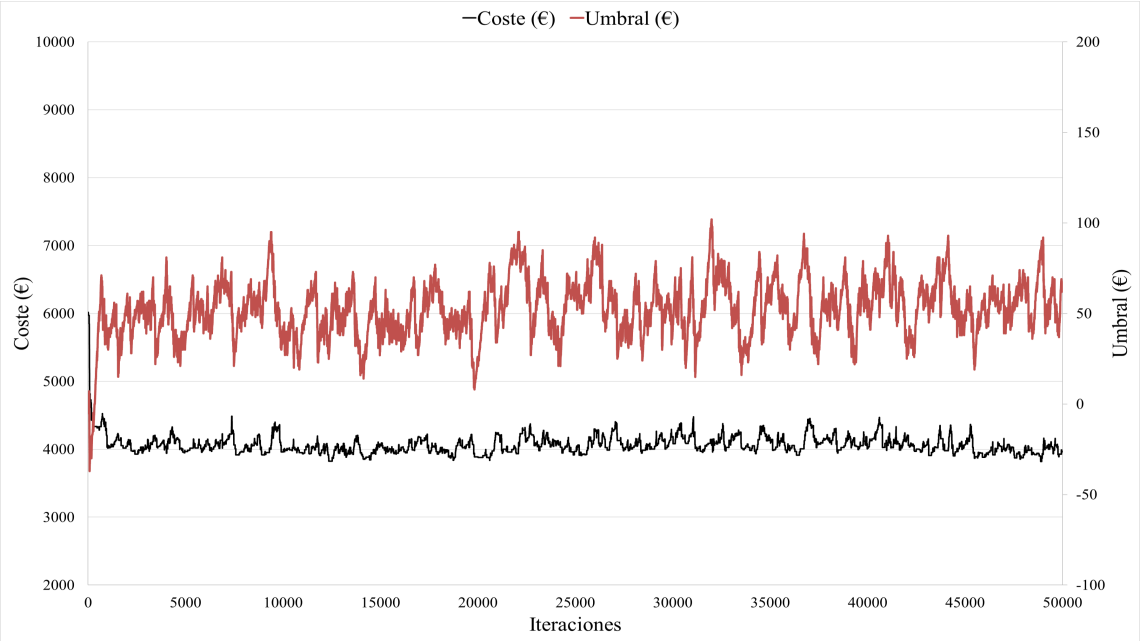


Figura 4.9: Evolución de una ejecución genérica del algoritmo OBA una vez calibrado (Elaboración propia).

4.4. Software de cálculo estructural y verificación de estados límite

Los tres primeros apartados del capítulo han detallado el funcionamiento y calibración de cada uno de los tres algoritmos considerados en la resolución del problema. Una vez establecido esto, el resto del capítulo describe en profundidad el funcionamiento interno, así como los fundamentos teóricos del software desarrollado.

El software de optimización estructural diseñado se ha desarrollado haciendo uso del lenguaje de programación *Python* [103]. Este lenguaje multiparadigma de alto nivel es uno de los más utilzados en la actualidad gracias, entre otras de sus características positivas, a la buena legibilidad de su código, o que posee una licencia *Open Source*.

El software de optimización diseñado cuenta con tres bloques principales, dentro de los que hay, a su vez, una serie de funciones internas. El primero de los bloques es el denominado **Módulo base de datos**, este contiene los parámetros del problema de optimización, tanto los de la estructura como los del algoritmo de recocido simulado.

El segundo bloque, denominado **Módulo de verificación**, es el encargado de llevar a cabo los cálculos estructurales necesarios para la comprobación de los estados límite pertinentes. Esto comprende el cálculo de esfuerzos, verificaciones geométricas y de armado, así como estados límite últimos y de servicio. Su funcionamiento queda detallado en los siguientes apartados del capítulo.

El tercer y último bloque base del software se denomina **Módulo de optimización**, consiste en la aplicación del algoritmo genético a la resolución del problema. Los tres bloques están interconectados entre sí, de forma que, mediante la transmisión de información entre ellos, consiguen llevar a cabo la resolución del problema de optimización propuesto. Dado que se han aplicado tres heurísticas diferentes en la resolución del problema, fue necesario desarrollar un módulo de optimización propio para cada uno de ellos. El código general del software *Python* desarrollado queda contenido en el apéndice denominado "Software desarrollado", localizado al final del documento.

4.4.1. Módulo base de datos

El módulo base de datos consiste, de forma básica, en la enumeración y completa definición, de los parámetros y variables del problema de optimización. En el caso de los parámetros esto puede consistir en establecer el valor mediante un número entero, o una expresión derivada de otros ya definidos.

En el caso de las variables, consiste, principalmente, en establecer un rango de valores válidos que estas podrán tomar a lo largo del proceso de resolución del problema. A nivel práctico esto consiste en vectores que contengan las diferentes posibilidades.

Además de parámetros y variables propios de la geometría del marco, sus materiales, o las acciones de cálculo establecidas en norma, así como sus coeficientes de seguridad y combinación correspondientes, también se establece el **modelo de marco** considerado para el cálculo estructural. Se define el número de barras en las que se discretiza cada parte de la estructura, siendo posible definir la localización

de cada uno de los nodos, o la distribución equidistante de estos entre los nodos extremos de cada componente.

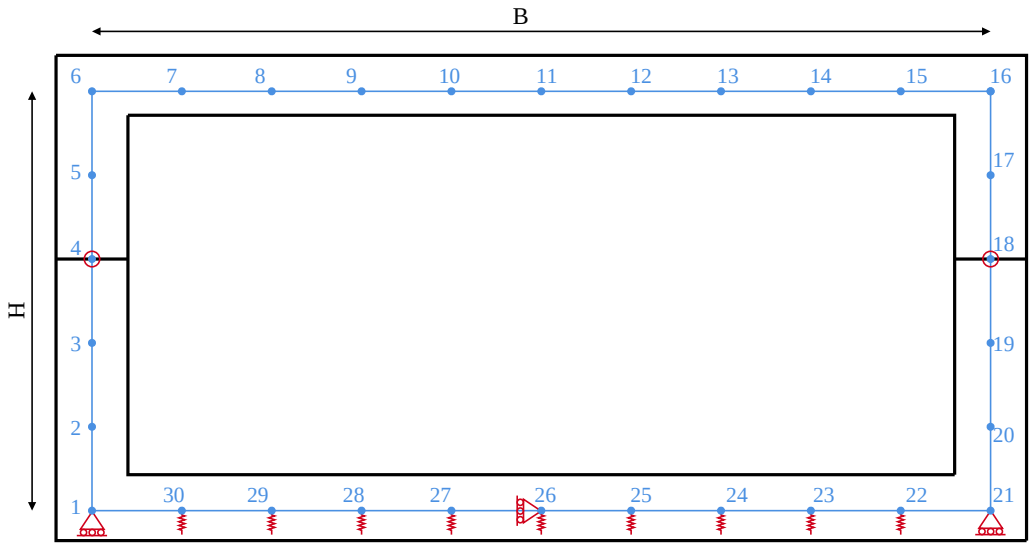


Figura 4.10: Representación gráfica del modelo de marco articulado considerado en el estudio (Elaboración propia).

Las condiciones de contorno aplicadas al modelo consisten en la restricción de tres desplazamientos. Dos de ellos en dirección vertical, impuestas a los nodos extremos, y la tercera en dirección horizontal, impuesta al nodo central de la losa inferior. Además de esto, con objeto de simular el medio elástico sobre el que se dispone la losa inferior del marco, cada uno de los nodos de la losa inferior cuyo desplazamiento vertical no está restringido, cuenta con un muelle de rigidez equivalente al módulo de balasto del terreno.

Además, dado que el marco de estudio es de tipología articulada, los nodos correspondientes a la unión entre la pieza que conforma la parte superior de los hastiales y la losa superior, con la pieza que conforma la parte inferior de los hastiales y la losa inferior, queda modelada mediante una articulación. Es decir, las barras que llegan y parten de ese nodo, tienen liberado el grado de libertad correspondiente al giro.

El proceso inicial del software consiste pues, en la lectura completa de la base de datos, algo que permite la **generación del modelo** del marco considerado. Esto comprende la definición completa de cada una de las secciones del marco. Con el

objetivo de facilitar la visualización de las características de cada una de las secciones se desarrolla una función que permita visualizar el armado y dimensiones de las diferentes partes del marco. La figura 4.11 muestra dos secciones genéricas de un marco determinado obtenidas mediante el software desarrollado.

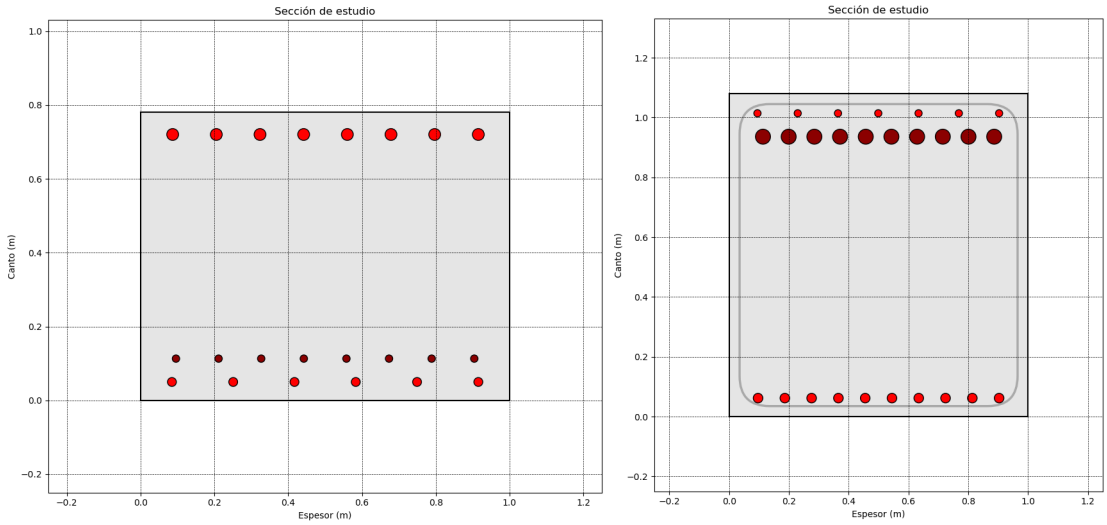


Figura 4.11: Representación gráfica de dos secciones de un marco genérico obtenida mediante el software desarrollado (Elaboración propia).

4.4.2. Módulo de verificación

El segundo módulo del software es, con diferencia, el módulo que, debido a la complejidad de los procesos internos que se han de llevar a cabo para su uso, ocupa un mayor porcentaje del tiempo de computación necesario para cada iteración.

4.4.2.1. Cálculo de esfuerzos internos

La obtención de esfuerzos se lleva a cabo mediante la resolución del Método Directo de Rigidez (MDR), un método matricial de uso común en aplicaciones software que hacen uso del Método de los Elementos Finitos o *Finite Element Method* (FEM).

El MDR permite, partiendo de la matriz de rigidez de un elemento tipo barra, en conjunto con las acciones actuantes sobre los nodos extremos de este, la obtención de los esfuerzos internos del elemento. Es por esto que el marco ha de ser discretizado

en una serie de elementos barra, los parámetros derivados de esto quedan definidos en la base de datos.

El cálculo consiste, en términos generales, en la multiplicación de la matriz de rigidez reducida de la estructura, con el vector de desplazamientos reducido de la misma.

$$\Big|F\Big| = \Big[K\Big] \cdot \Big|U\Big| \tag{4.7}$$

Donde

$|F|$ es el vector de cargas nodales;

$[K]$ es la matriz de rigidez de la estructura;

$|U|$ es el vector desplazamientos nodales.

En el caso de estudio cada uno de los nodos del modelo tiene tres grados de libertad asociados a los desplazamiento en dirección longitudinal del elemento barra, desplazamiento en dirección perpendicular al elemento barra, y giro entorno al eje perpendicular al plano de desplazamientos. De esta forma, cada uno de los elementos barra que componen el modelo cuenta con seis grados de libertad cuyos desplazamientos están asociados a sus esfuerzos de tipo axil, cortante y momento flector.

En este contexto, la primera de las funciones internas que componen el bloque Verificador es la encargada del **cálculo de la matriz de rigidez reducida de la estructura**. En primer lugar se calcula la matriz de rigidez K_{ij} de cada uno de los elementos tipo barra del modelo. Esta matriz, considera el módulo elástico del material que compone el elemento barra, su longitud, así como el área e inercia de la sección. La matriz de rigidez de cada una de las barras que componen el modelo es simétrica y tiene una dimensión de 6×6.

De forma genérica el nodo inicial de la barra es denominado i y el final j, de forma que la ecuación descrita anteriormente aplicada a un elemento barra genérico tendría la siguiente forma.

$$\begin{vmatrix} F_{hi} \\ F_{vi} \\ M_i \\ F_{hj} \\ F_{vj} \\ M_j \end{vmatrix} = \begin{bmatrix} \frac{E \cdot A}{L} & 0 & 0 & -\frac{E \cdot A}{L} & 0 & 0 \\ 0 & \frac{12 \cdot E \cdot I}{L^3} & \frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{12 \cdot E \cdot I}{L^3} & -\frac{6 \cdot E \cdot I}{L^2} \\ 0 & \frac{6 \cdot E \cdot I}{L^2} & \frac{4 \cdot E \cdot I}{L} & 0 & -\frac{6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} \\ -\frac{E \cdot A}{L} & 0 & 0 & \frac{E \cdot A}{L} & 0 & 0 \\ 0 & -\frac{12 \cdot E \cdot I}{L^3} & -\frac{6 \cdot E \cdot I}{L^2} & 0 & \frac{12 \cdot E \cdot I}{L^3} & -\frac{6 \cdot E \cdot I}{L^2} \\ 0 & \frac{6 \cdot E \cdot I}{L^2} & \frac{2 \cdot E \cdot I}{L} & 0 & -\frac{6 \cdot E \cdot I}{L^2} & \frac{4 \cdot E \cdot I}{L} \end{bmatrix} \cdot \begin{vmatrix} U_i \\ V_i \\ \theta_i \\ U_j \\ V_j \\ \theta_j \end{vmatrix}$$

Tras el cálculo de la matriz de rigidez de cada una de las barras, se procede al ensamblaje de la matriz de rigidez primaria K_p, esta es resultado de la unión de las anteriores considerando la posición que cada elemento toma en el modelo del marco. Finalmente, con el objetivo de obtener la matriz de rigidez de la estructura K_s la matriz primaria es reducida eliminando las filas y columnas correspondientes a los grados de libertad restringidos por las condiciones de contorno impuestas al modelo.

Una vez obtenida la matriz de rigidez reducida, una segunda función procede al **cálculo el vector reducido de fuerzas de la estructura**. Esta función parte de los parámetros que definen las acciones actuantes sobre la estructura, es decir, lee la información contenida en la base de datos, para luego proceder a su tratamiento. En primer lugar es necesario calcular la magnitud de las cargas actuantes haciendo uso de los parámetros del problema y las expresiones desarrolladas en el capítulo anterior.

Tras esto, las acciones son combinadas según lo establecido en norma para así obtener cada uno de los casos de carga de la estructura. Una vez establecidos los casos de carga, es necesario calcular las fuerzas de empotramiento perfecto de cada uno de los elementos barra del modelo para cada uno de los casos. Finalmente, se procede al ensamblaje y posterior reducción mediante eliminación de las filas asociadas a los grados de libertad restringidos por las condiciones de contorno, obteniendo así el vector de fuerzas de la estructura F.

Una vez conocido el vector reducido de fuerzas de la estructura F, así como su matriz de rigidez reducida K_s, una tercera función interna procede a la resolución para el **cálculo del vector reducido de desplazamientos** de la estructura. Siendo posible completar con valor nulo las posiciones asociadas a los desplazamientos restringidos en el vector, obteniendo el vector completo de desplazamientos de la estructura.

Finalmente, una vez determinado de forma completa los desplazamientos nodales de la estructura es posible obtener los **esfuerzos internos** en las secciones correspondientes a los nodos.

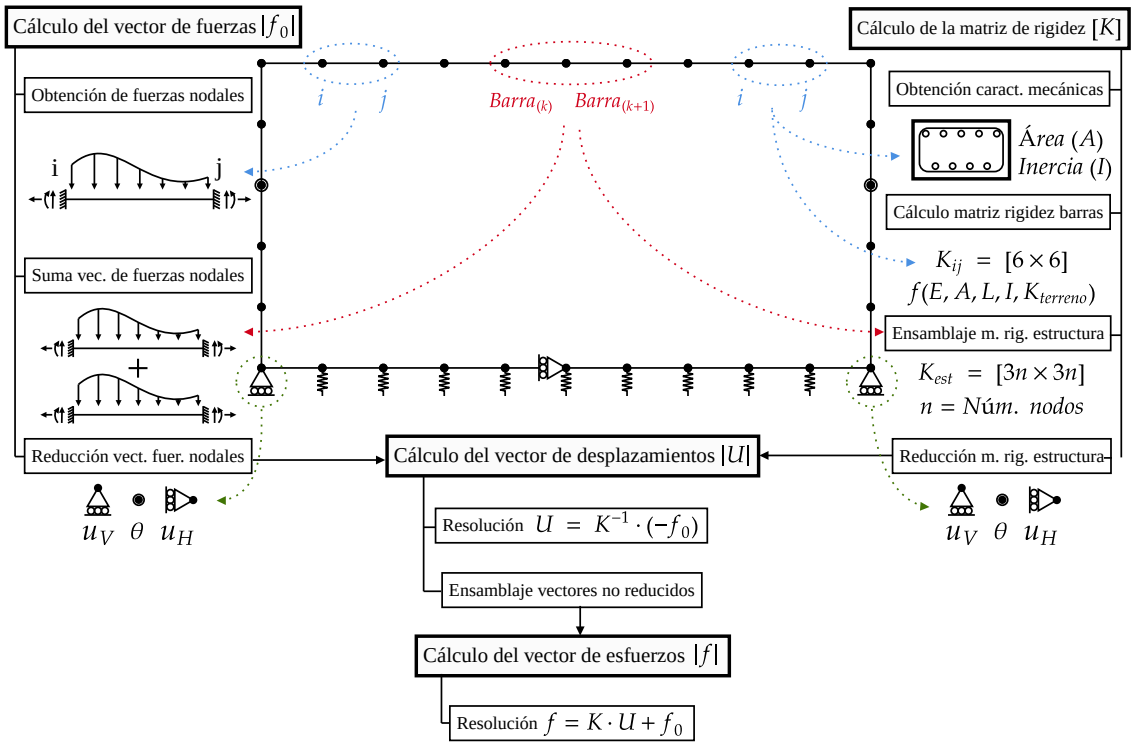


Figura 4.12: Proceso de cálculo de esfuerzos mediante el software desarrollado (Elaboración propia).

Con el objetivo de poder visualizar los resultados, y así facilitar un medio sencillo de comprobación de la validez de los esfuerzos obtenidos, se desarrollan funciones que permiten obtener representaciones gráficas tanto de los esfuerzos, como de la deformada de la estructura, ejemplificado en las figuras 4.13 a 4.17.

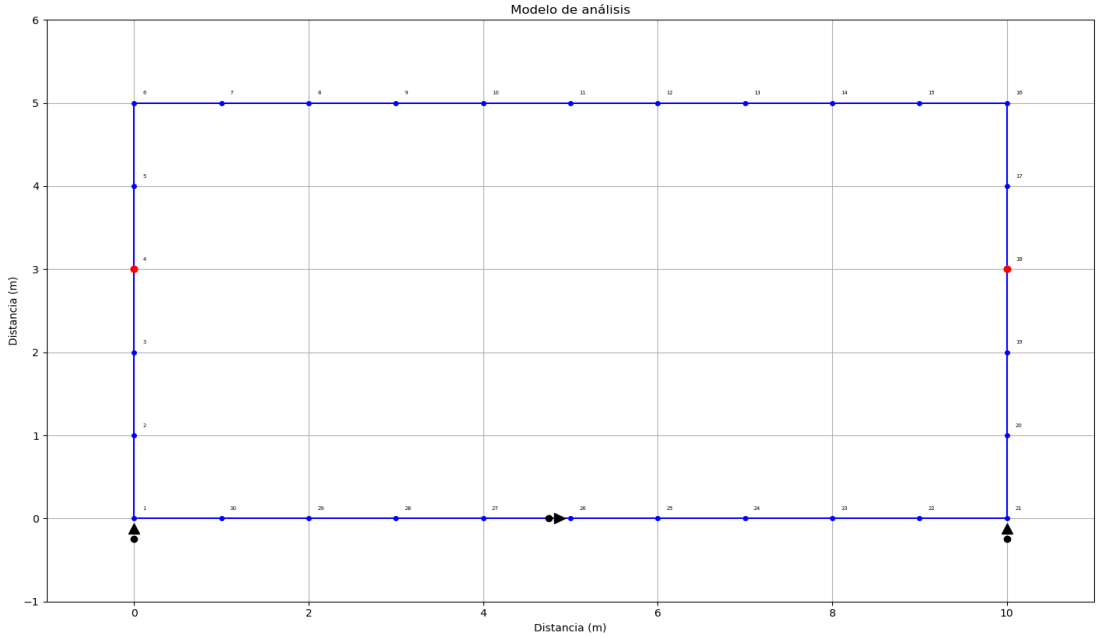


Figura 4.13: Representación gráfica del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

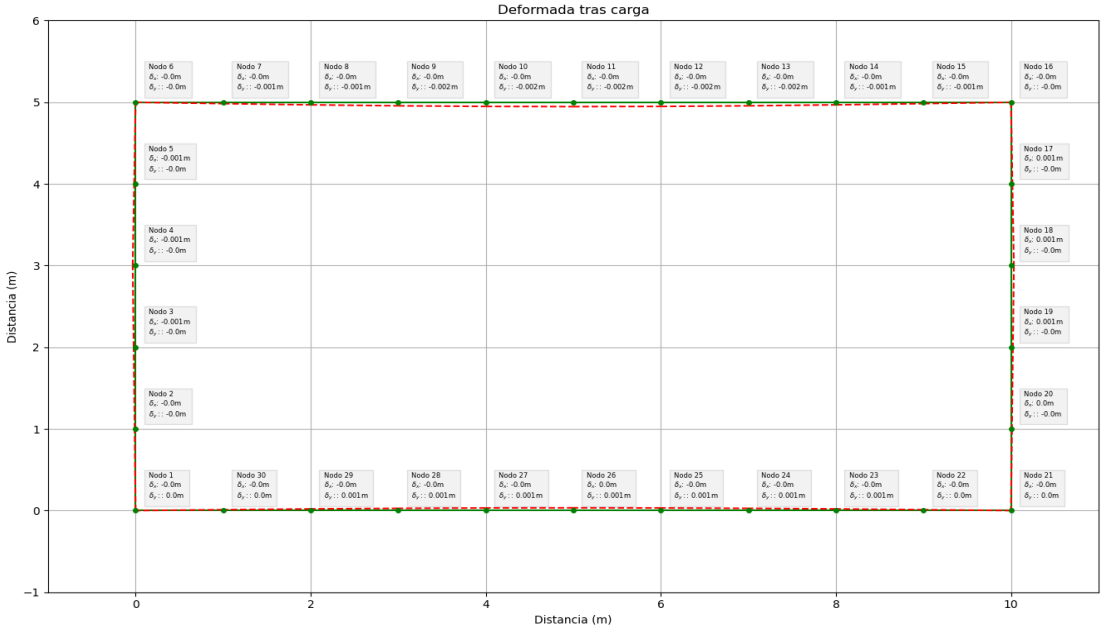


Figura 4.14: Representación gráfica de la deformada del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

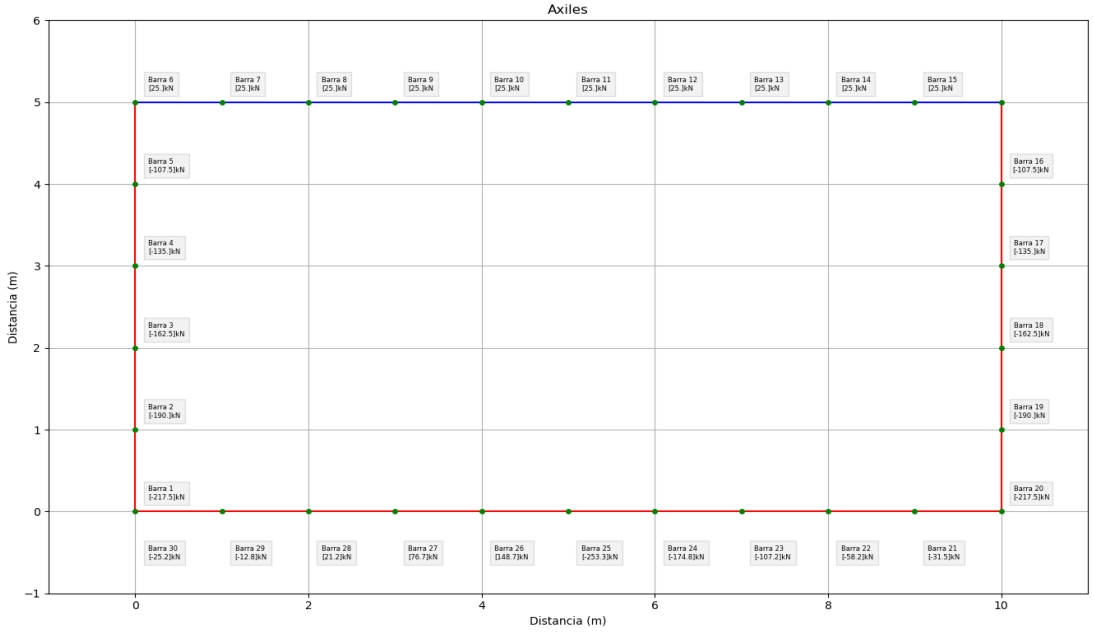


Figura 4.15: Representación gráfica del esfuerzo axil del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

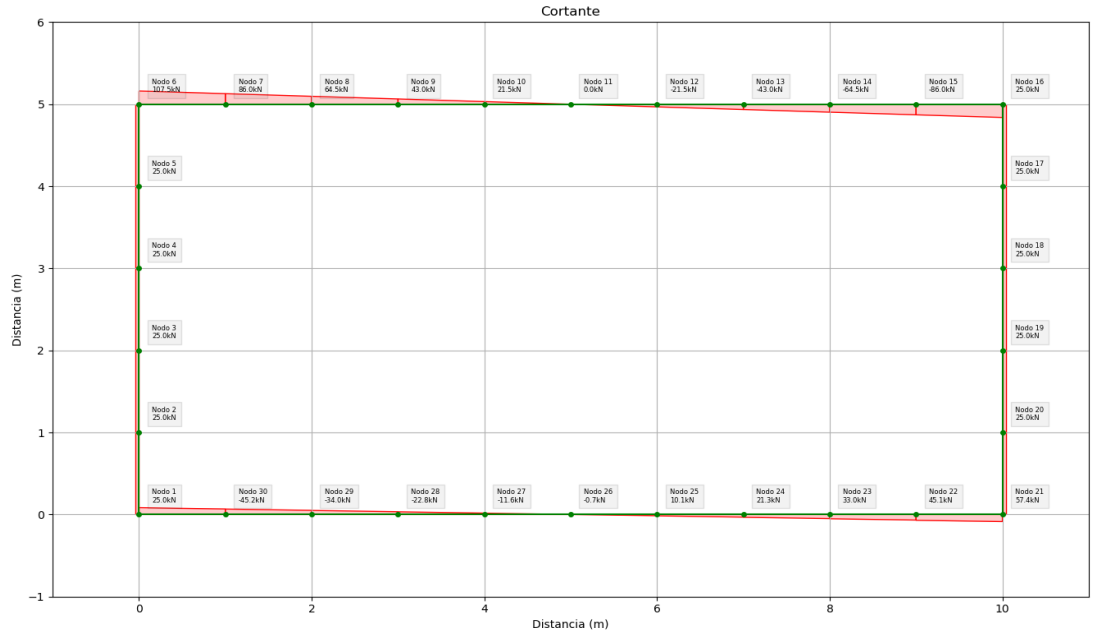


Figura 4.16: Representación gráfica del esfuerzo cortante del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

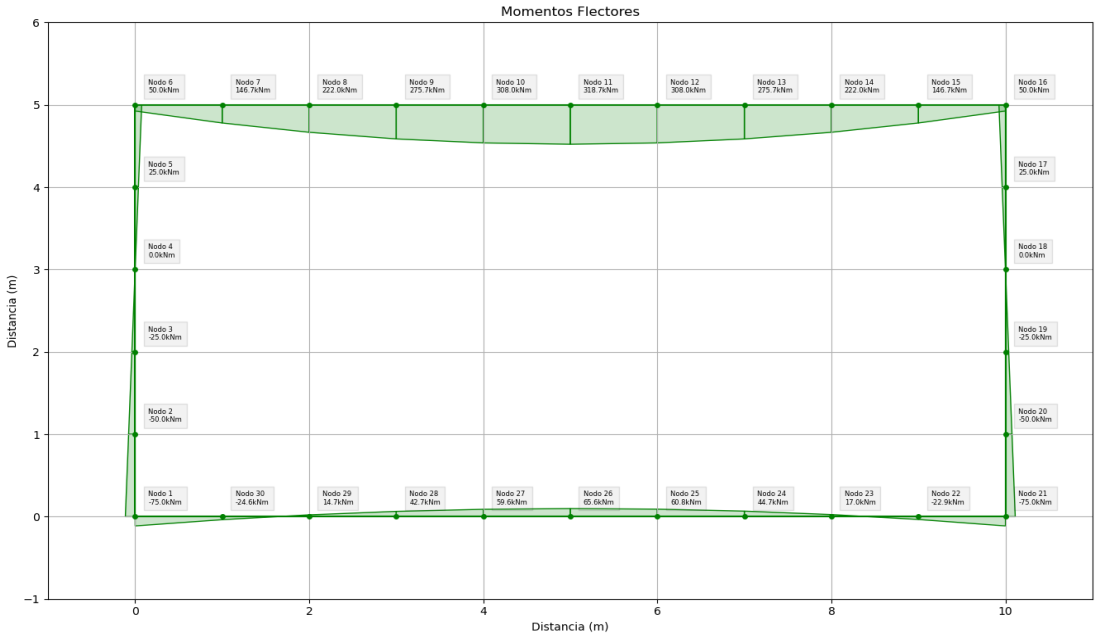


Figura 4.17: Representación gráfica del momento flector del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

4.4.2.2. Verificación de los estados límite

Una vez determinados los esfuerzos internos de la estructura, el siguiente paso consiste en la verificación de los estados límite pertinentes. Los cálculos necesarios fueron detallados en el capítulo anterior, de forma que a continuación se desarrolla el orden de verificación así como una serie de comentarios relevantes sobre el funcionamiento interno del software desarrollado.

Una vez han sido calculados los esfuerzos internos de la estructura para cada uno de los casos de carga, el software procede al **cálculo de las envolventes** necesarias para la verificación de los estados límite establecidos en norma. De forma similar a lo presentado en el cálculo de esfuerzos, con el objetivo de poder visualizar los resultados de forma sencilla, se desarrollan funciones que representen las envolventes de esfuerzos del modelo. En este contexto, la figuras 4.18 a 4.20 se corresponden con las envolventes de esfuerzos de un marco genérico obtenidas mediante el software desarrollado.

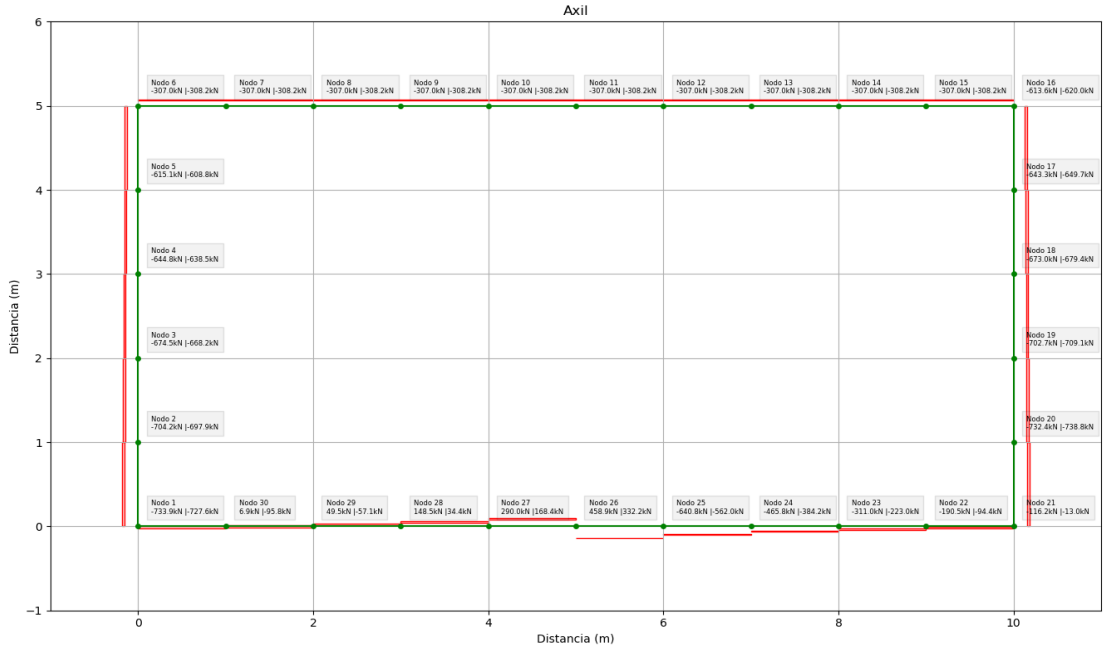


Figura 4.18: Representación gráfica del esfuerzo axil del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

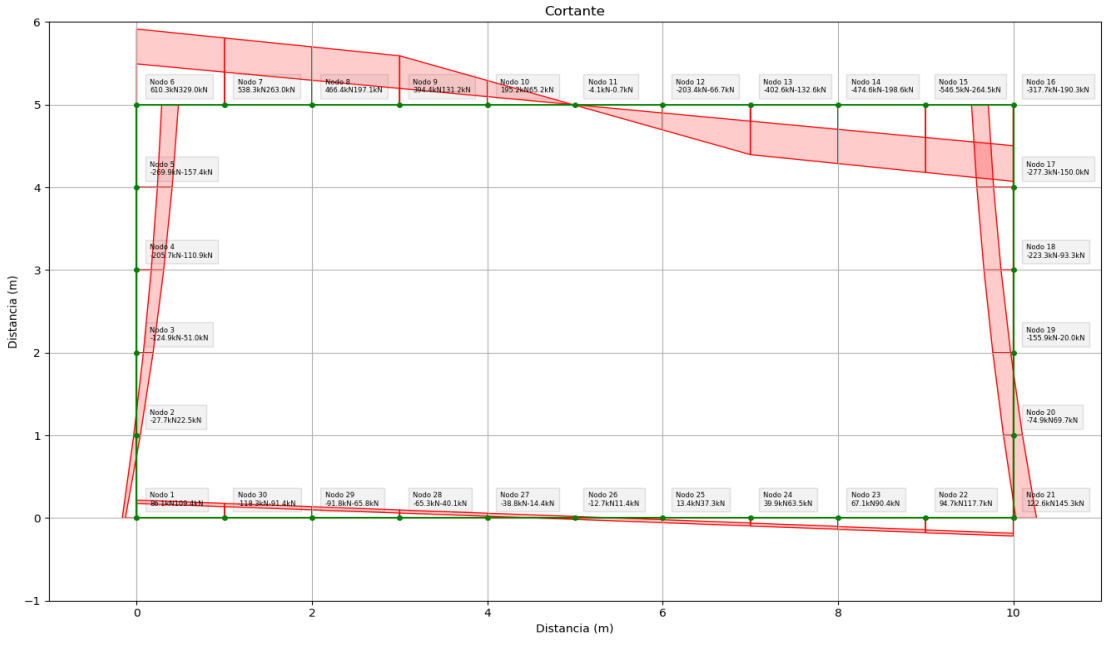


Figura 4.19: Representación gráfica del esfuerzo cortante del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

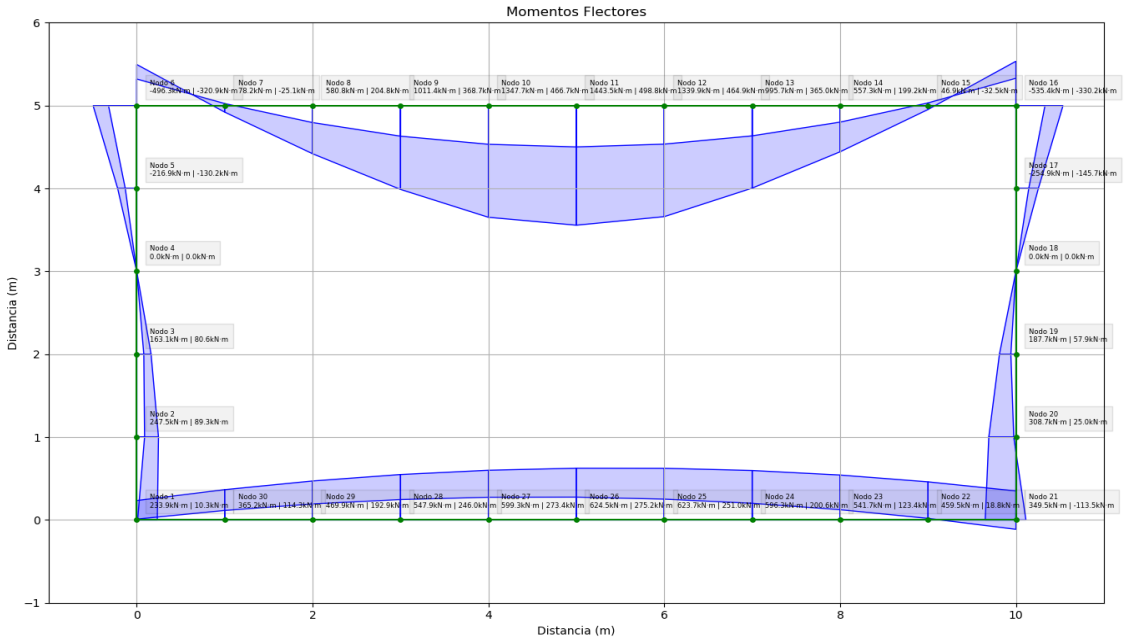


Figura 4.20: Representación gráfica del momento flector del modelo considerado para el cálculo de los esfuerzos del marco prefabricado (Elaboración propia).

Dando paso a la comprobación de los estados límite pertinentes, en primer lugar se verifica el **estado límite de agotamiento a cortante**, de esta forma es posible aplicar la ley de decalaje pertinente. Tras esto, se procede a la verificación del **estado límite frente a solicitaciones normales**. Este procedimiento cuenta con una primera parte en la que una función iterativa genera el diagrama de interacción de cada una de las secciones, para tras esto, comprobar que el punto correspondiente al axil y momento actuantes se sitúa dentro de la región cerrada delimitada por el diagrama de interacción de la sección. Tras la verificación frente a esfuerzos normales, se procede a la comprobación del cumplimiento del ultimo estado límite considerado, el **estado límite de fisuración**.

Llegados a este punto es relevante mencionar el hecho de que, si bien es cierto que la comprobación del estado límite de fatiga es un punto importante en el proceso de diseño de estructuras enterradas con la tipología considerada, el modelado de las cargas necesarias para su verificación quedó fuera del alcance del presente estudio. De esta forma, la verificación de este estado límite así como la consideración de cargas asociadas al transporte y montaje del marco constituyen una línea de investigación

en la que profundizarán estudios posteriores, algo que se detalla en el capítulo final del documento.

Reanudando el proceso de comprobación, una vez verificados todos los estado límite, es necesario realizar el cálculo y comparación de las armaduras de la estructura con las **armaduras mínimas y máximas** estipuladas en norma. Este proceso es relativamente directo ya que el diseño de armado de la estructura es conocido por lo que simplemente consiste en la comparación directa y verificación de cumplimiento de los límites establecidos en la norma UNE-EN 1992-1-1 [98].

De esta forma, siguiendo los pasos descritos, el software desarrollado verifica las restricciones de comportamiento y diseño establecidas. La figura 4.21 presenta un esquema general del funcionamiento interno del módulo de verificación estructural del software desarrollado.

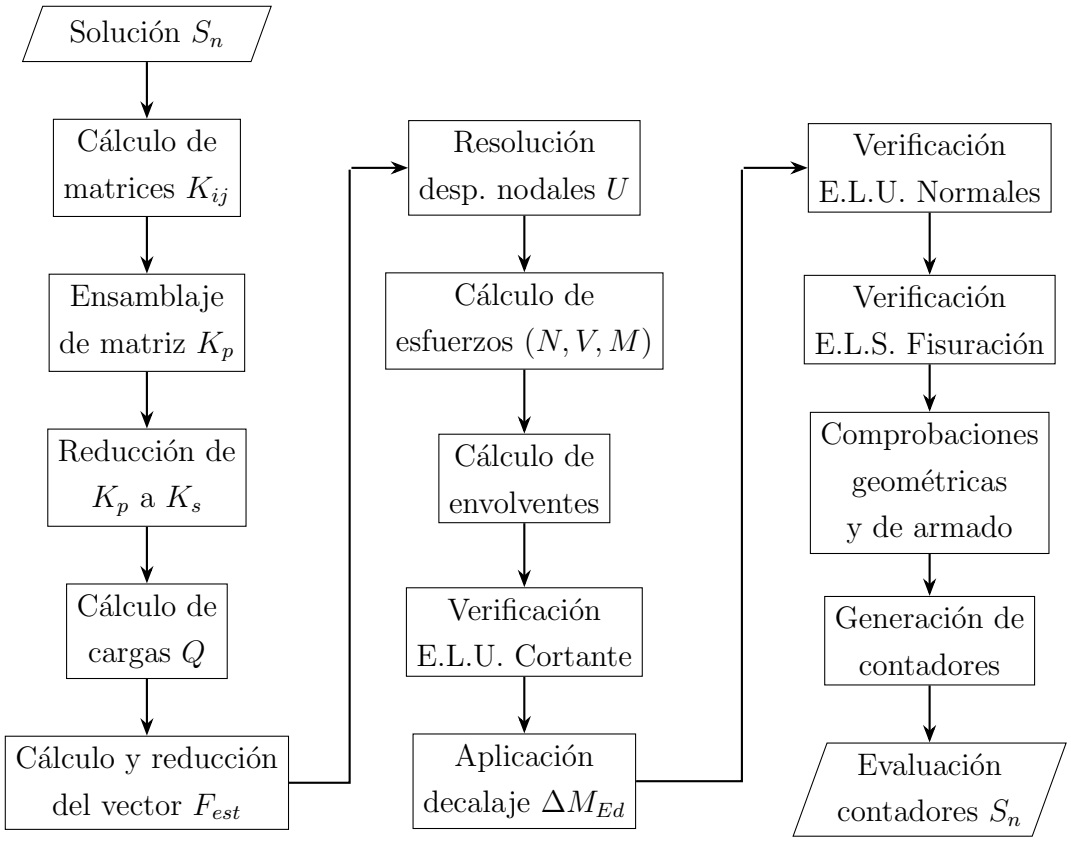


Figura 4.21: Diagrama de flujo del funcionamiento interno del módulo de verificación del software desarrollado (Elaboración propia).

El resultado final arrojado por el módulo de verificación consiste en una serie de contadores cuyo valor determina el número de secciones que no verifican cada una de las restricciones impuestas. De esta forma, si todos los contadores son nulos, la estructura cumple los requerimientos y se considera una solución factible. En caso de que alguno de los contadores no sea cero, la estructura no cumple las restricciones necesarias y no puede considerarse como una solución factible al problema.

4.4.3. Módulo de optimización

El tercer y último bloque principal del software desarrollado es el **módulo de optimización**. Este hace uso de la información contenida en la base de datos, así como de los resultados arrojados por el módulo de verificación.

Los procesos internos del software varían en función del algoritmo aplicado en la resolución del problema, sin embargo, el funcionamiento general del módulo de optimización es común para todos ellos. El módulo conforma un proceso iterativo en el que, durante el proceso de generación y aceptación de soluciones, se hacen llamadas al módulo de verificación, y se modifica la información contenida en la base de datos.

De esta forma, tras la lectura y comprobación de la primera solución, el módulo de optimización genera una nueva solución aplicando un movimiento determinado a la solución inicial. Tras esto, se ejecuta una llamada al módulo de verificación, que comprueba la validez de la nueva solución y devuelve el valor de cada uno de los contadores asociados a las distintas restricciones del problema.

En caso de que todos los contadores proporcionados por el módulo de verificación sean nulos, la nueva solución es aceptada. En este momento, mediante la función interna denominada cambio de variables, el módulo de optimización sobrescribe la información contenida en la base de datos, sustituyendo de esta forma la solución antigua por la nueva solución aceptada.

Este proceso de llamada a verificación y escritura de datos se repite de forma reiterada durante un número de iteraciones determinado por los parámetros del algoritmo. Una vez se cumple el criterio de parada o terminación del algoritmo, el módulo de optimización genera dos documentos Excel cuyo contenido es el valor de cada una de las variables objeto de optimización en cada una de las iteraciones. El

primer documento almacena la solución óptima obtenida hasta el momento en cada una de las iteraciones, y el segundo almacena cada una de las soluciones aceptadas durante el funcionamiento del algoritmo. El funcionamiento general del módulo de optimización descrito en este apartado puede verse en la figura 4.22.

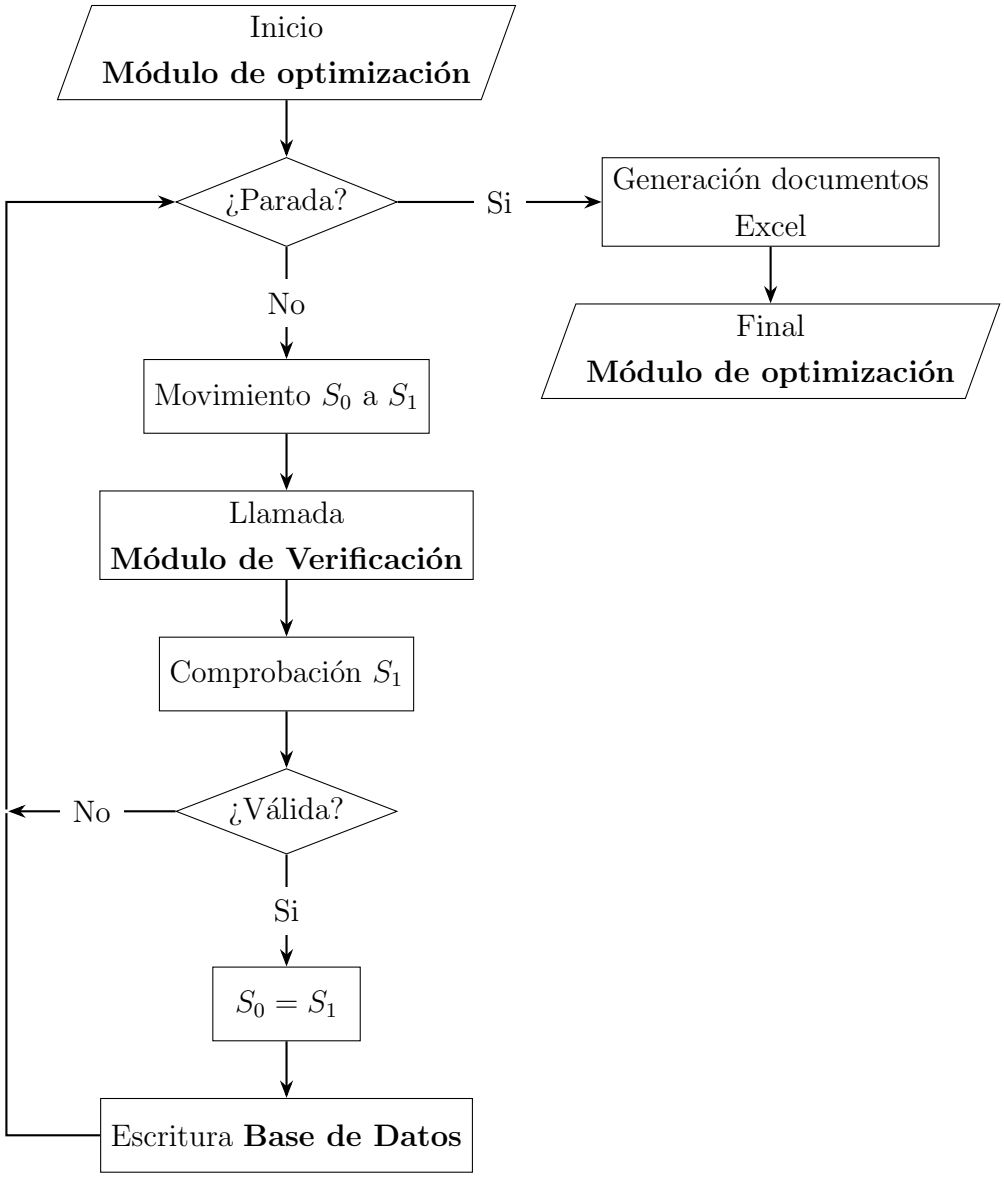


Figura 4.22: Funcionamiento del módulo de optimización del software desarrollado (Elaboración propia).

4.5. Características del proceso de obtención de resultados

El proceso de obtención de resultados consiste, a término general, en la aplicación reiterada de cada uno de los algoritmos de resolución. Para ello se ha hecho uso de un dispositivo con sistema operativo Windows 10 de 64 bits, con un procesador Intel(R) Core(TM) i7-8700K y 32GB de memoria RAM, contando también con una tarjeta gráfica NVIDA GeForce GTX 1060 de 6GB. Las principales características del dispositivo así como del entorno utilizado en la programación y desarrollo del software propio quedan resumidas en la tabla 4.1.

Elemento	**Modelo**
Sistema operativo	Windows 10 64 bits
Procesador	Intel(R) Core(TM) i7-8700K
Memoria RAM	32 GB
Tarjeta gráfica	NVIDIA GeForce GTX 1060 6GB
Entorno Python	PyCharm Community Edition 2020.3.2
Intérprete Python	3.8

Tabla 4.1: Principales características del dispositivo y entorno de desarrollo utilizados.

Cuando el software desarrollado se ejecuta en el dispositivo descrito, cada iteración tiene un tiempo medio de ejecución cercano a 0,35 segundos, permitiendo calcular aproximadamente tres marcos por segundo.

En este contexto, la tabla 4.2 presenta el tiempo consumido en cada uno de los procesos para la comprobación de la primera solución inicial válida S_0 generada de forma aleatoria para un algoritmo OBA. Estos valores quedan representados en la figura 4.23, donde, de forma visual, se muestra el porcentaje de tiempo de computación asociado a cada uno de los procesos contenidos en la tabla anterior.

Con un consumo porcentual total del 58.33 %, el módulo de verificación comprende la verificación de estados límite, incluyendo comprobaciones asociadas a la armadura, procesos que conllevan un mayor coste computacional. Esto se debe a la

necesidad de ejecutar subprocesos iterativos compuestos por bucles embebidos con un gran número de condiciones.

	Tiempo (*Segundos*)
Generación del modelo	0.000998
Cálculo M.D.R.	0.118680
Cálculo de envolventes	0.039892
Verificación ELU Cortante	0.065822
Verificación ELU Sol. Normales	0.088760
Verificación ELS Fisuración	0.068814
Tiempo total	**0.382966**

Tabla 4.2: Tiempo de computación consumido por cada uno de los principales procesos internos del software desarrollado.

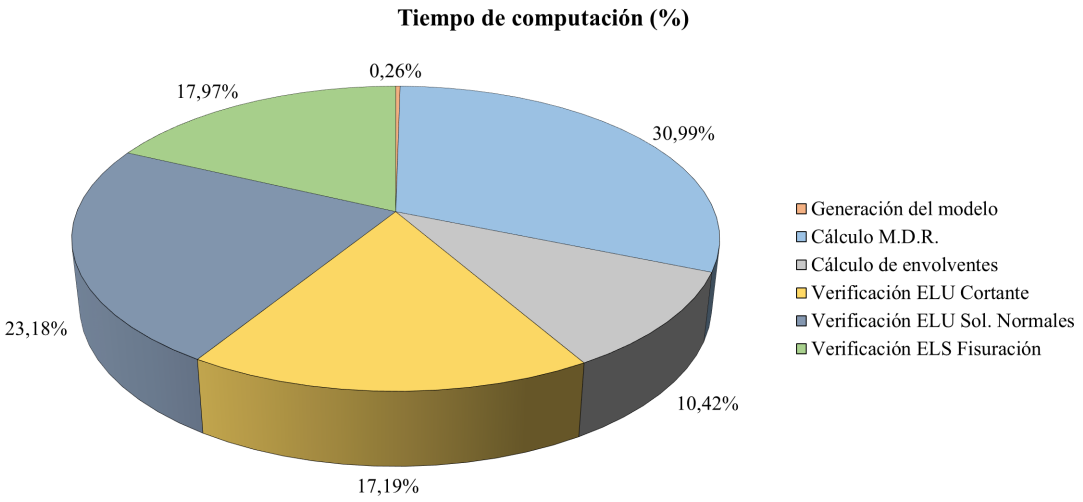


Figura 4.23: Porcentaje de tiempo de computación consumido por cada uno de los principales procesos internos del software desarrollado (Elaboración propia).

Es relevante mencionar que existen diferentes configuraciones, así como modificaciones en el software desarrollado, que permiten reducir los tiempos de computación. El estudio y optimización de estas configuraciones con objetivo de reducir los tiempos globales de cálculo serán objeto de estudio en investigaciones posteriores.

Capítulo 5

Resultados de la optimización económica

Una vez establecido el funcionamiento interno del software desarrollado, así como sus fundamentos teóricos, el quinto capítulo del presente trabajo final de máster es dedicado a la presentación de los resultados obtenidos en la optimización económica del marco prefabricado mediante el uso de un algoritmo de recocido simulado (SA), un algoritmo de aceptación por umbrales (TA), y un tercer algoritmo del solterón (OBA).

5.1. Justificación del número de ensayos

Una de las cuestiones derivadas del uso de algoritmos para la resolución del problema de optimización es el número de reinicios necesarios para poder asegurar, con una probabilidad suficientemente alta, que la solución obtenida es de gran calidad. En el caso de la optimización estructural, la determinación del número de ejecuciones no es algo sencillo ni directo. La forma concreta de determinar esto sería la comparación directa de las soluciones obtenidas con el óptimo global del problema, algo que no es viable dado que este no es conocido.

En este contexto, se plantea la idea de que, si tras una serie de ejecuciones, las soluciones obtenidas son una muestra suficientemente representativa de la población origen, podemos asumir que la mejor solución obtenida en esa muestra será repre-

sentativa de la mejor solución del problema. Es decir, a partir de cierto número de ejecuciones, la probabilidad de encontrar soluciones óptimas de mejor calidad se reduce, ya que estas tienden a ser resultados cercanos al valor promedio de los ya obtenidos.

Perea et al. [104], plantea el estudio de este acontecimiento mediante un análisis estadístico de resultados en el que se mide la media y desviación típica de las soluciones óptimas obtenidas en cada ejecución. Las conclusiones del estudio enuncian que, cuando la media y varianza de las soluciones óptimas se estabilizan, estas constituyen una muestra representativa del espacio de soluciones completo, algo que, en los casos analizados, podía alcanzarse con un número de entre veinte y treinta y cinco ejecuciones.

Sin embargo, estudios como los desarrollados por Martí Albiña [14], profundizan en este tema. Para ello se estudia la evolución de la desviación estándar de las soluciones óptimas para veinte ejecuciones un algoritmo SA. Los resultados muestran que tras nueve ejecuciones, tanto la media como la desviación se estabilizan a partir de la séptima ejecución, siendo suficientemente estables en la novena.

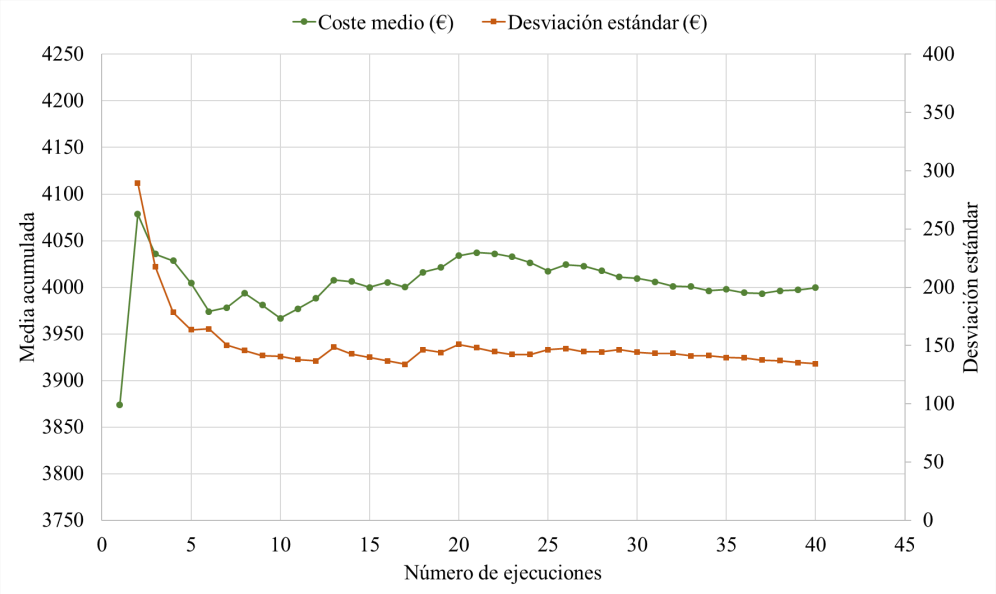


Figura 5.1: Evolución de la media acumulada y la desviación estándar con el número de ejecuciones (Elaboración propia).

Con el objetivo de reducir los tiempos de computación necesarios para la ob-

tención de resultados representativos, se lleva a cabo un proceso análogo para el problema que nos concierne.

De esta forma, se ejecuta un algoritmo SA un número total de cuarenta ejecuciones, obteniendo los resultados que se presentan en la figura 5.1, correspondientes a los valores de las tablas 5.1 y 5.2.

Ejec.	**Coste (€)**	**Med. Ac.**	**Var. (%)**	**Desv. Est.**	**Var. (%)**
1	3873,93	3873,93	-	0,00	-
2	4283,34	4078,63	-	289,49	-
3	3949,92	4035,73	-	217,77	-
4	4006,69	4028,47	-	178,40	-
5	3908,47	4004,47	-	163,56	-
6	3821,57	3973,99	-2,52	164,24	-100,00
7	4004,64	3978,37	2,52	150,38	92,51
8	4100,25	3993,60	1,05	145,74	49,42
9	3880,94	3981,08	1,19	141,41	26,16
10	3838,04	3966,78	0,95	140,78	16,18
11	4081,03	3977,17	-0,08	137,93	19,08
12	4108,15	3988,08	-0,24	136,84	9,90
13	4241,81	4007,60	-0,35	148,72	-2,00
14	3990,41	4006,37	-0,63	142,96	-1,08
15	3910,80	4000,00	-0,83	139,95	0,60
16	4084,48	4005,28	-0,70	136,84	0,79
17	3921,56	4000,36	-0,31	134,05	2,08
18	4287,13	4016,29	-0,22	146,56	1,47
19	4112,50	4021,35	-0,37	144,13	-0,81
20	4272,55	4033,91	-0,84	151,11	-7,39

Tabla 5.1: Resultados de las veinte primeras ejecuciones control del algoritmo SA.

Ejec.	Coste (€)	Med. Ac.	Var. (%)	Desv. Est.	Var. (%)
21	4105,22	4037,31	-0,79	148,11	-7,61
22	4009,66	4036,05	-0,88	144,66	-7,34
23	3957,97	4032,66	-0,41	142,27	3,02
24	3883,97	4026,46	-0,13	142,41	1,21
25	3802,32	4017,49	0,41	146,44	3,19
26	4191,12	4024,17	0,33	147,47	0,43
27	3983,56	4022,67	0,33	144,82	-0,11
28	3886,07	4017,79	0,37	144,44	-1,50
29	3819,84	4010,96	0,39	146,52	-2,80
30	3966,43	4009,48	0,20	144,20	1,56
31	3896,00	4005,82	0,46	143,23	2,96
32	3857,88	4001,20	0,54	143,31	1,05
33	3983,58	4000,66	0,43	141,09	2,37
34	3849,35	3996,21	0,37	141,34	3,67
35	4055,62	3997,91	0,29	139,60	3,29
36	3860,04	3994,08	0,29	139,50	2,68
37	3964,70	3993,29	0,20	137,63	4,12
38	4105,73	3996,24	0,11	136,98	3,00
39	4032,21	3997,17	-0,02	135,29	4,47
40	4099,23	3999,72	-0,05	134,51	3,78

Tabla 5.2: Resultados de las veinte últimas ejecuciones control del algoritmo SA.

Como se puede observar, existe una variación inicial significativa, sin embargo tras la séptima ejecución tanto la media como la varianza empiezan a estabilizarse. Esto da lugar a que en la novena ejecución estos valores sean suficientemente estables como para considerar que la solución es representativa de la muestra del espacio de soluciones completo.

Observando los resultados presentados en la tabla 5.2 se puede comprobar que el óptimo global del conjunto de ejecuciones se obtiene en el vigésimo quinto reinicio, con un tiempo computacional de 11 horas, 13 minutos y 1 segundo. Sin embargo,

este valor no difiere en gran medida del óptimo de mejor calidad conseguido dentro de los nueve primeros reinicios, correspondiente a la sexta ejecución, con un tiempo computacional muy inferior, de tan sólo 3 horas, 38 minutos y 57 segundos.

Con un coste de 3821.57€, el óptimo de mejor calidad dentro de las nueve primeras ejecuciones tiene un coste un 0.506270 % superior al del óptimo global del conjunto de resultados obtenido en la vigésimo quinta iteración. Una diferencia poco notable considerando que el tiempo de computación requerido es casi cinco veces superior.

El tiempo global de computación de las cuarenta iteraciones realizadas para el algoritmo SA de control fue de 17 horas, 27 minutos y 54 segundos. En vista de la considerable reducción en tiempo de computación asociada a realizar nueve reinicios frente a los cuarenta realizados en el control, sumado al hecho de que la las nueve primeras soluciones óptimas conforman una muestra representativa del espacio de soluciones, es viable afirmar que el uso de un número finito de nueve reinicios para cada uno de los casos de estudio abordados en la optimización del marco prefabricado es justificado.

5.2. Presentación y análisis de los resultados

Una vez justificado el número de ensayos, el resto del capítulo presenta los resultados obtenidos tras la aplicación de cada una de las heurísticas consideradas en la optimización económica del marco prefabricado. En primer lugar se presentan los resultados obtenidos mediante la aplicación del algoritmo de recocido simulado, en segundo el algoritmo de aceptación por umbrales, y finalmente el algoritmo del solterón.

Con el objetivo de proporcionar una imagen clara sobre los resultados obtenidos, la presentación los mismos comprende una serie de tablas, representaciones gráficas, así como análisis estadísticos, de características comunes para los tres algoritmos genéticos.

5.2.1. Algoritmo de recocido simulado (SA)

Comenzando con la presentación de los resultados fruto del estudio desarrollado, en primer lugar se detallan los obtenidos mediante la aplicación del algoritmo de recocido simulado en la resolución del problema.

En este contexto, la tabla 5.3 muestra la configuración de los parámetros de cada uno de los diferentes algoritmos de recocido simulado aplicados en la optimización del marco prefabricado. Cada una de estas configuraciones comparte un mismo tipo de movimiento ya descrito en capítulos anteriores, este modifica un máximo de cinco variables del marco para así generar una nueva solución. Los parámetros que diferencian cada uno de los algoritmos SA son pues, la longitud de cadena de Markov, y la magnitud del coeficiente de enfriamiento del problema.

Heurística	**C. Markov**	**C. Enfriamiento**	**Tiempo(s)**	**C. mínimo (€)**
SA1	500	0.80	13.136,95	3.821,57
SA2	500	0.90	24.527,61	3.755,27
SA3	500	0.95	46.262,48	3.713,99
SA4	1000	0.80	27.590,72	3.709,59
SA5	1000	0.90	45.887,28	3.761,14
SA6	1000	0.95	84.024,97	3.708,99
SA7	5000	0.80	124.321,11	3.691,30
SA8	5000	0.90	246.138,28	3.707,36
SA9	5000	0.95	489.242,56	3.683,84

Tabla 5.3: Parámetros definitorios de cada una de las heurísticas SA aplicadas, junto con los costes mínimos de la mejor solución encontrada por cada una de ellas (Elaboración propia).

Cada una de las nueve heurísticas de recocido simulado es aplicada en consecuencia con lo enunciado en la parte inicial del capítulo. De esta forma, los resultados asociados a cada algoritmo son resultado de nueve reinicios diferenciados, algo que permite asegurar la validez de la respuesta del algoritmo, así como evaluar y verificar su robustez. La tabla 5.4 y figura 5.2 se corresponden con la evolución de los resultados obtenidos en cada una de las nueve iteraciones del algoritmo SA8.

Reinicio	Coste (€)	Tiempo s
1	3.751,73	29.714,11
2	3.847,22	28.574,27
3	3.820,43	26.886,72
4	3.707,37	28.252,66
5	3.804,61	26.563,56
6	3.755,92	27.926,85
7	3.783,16	27.545,16
8	3.707,36	25.268,31
9	3.745,12	25.406,64
Valor medio	3.768,93	27.313,61
Valor mínimo	3.707,36	25.268,31
Desviación típica	48,56	1466.99

Tabla 5.4: Evolución de los resultados de la heurística SA8 para cada uno de los nueve reinicios.

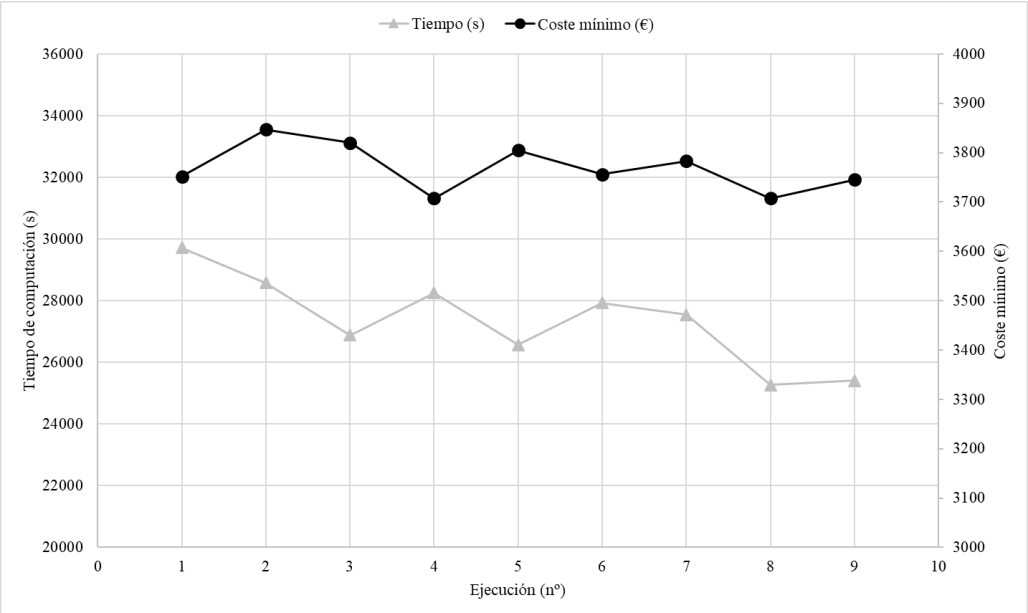


Figura 5.2: Evolución del coste mínimo y tiempo de computación de la heurística SA8 para cada uno de los nueve reinicios (Elaboración propia).

Dando paso a la presentación general de los resultados, la figura 5.3 muestra cada uno de los marcos de menor coste obtenidos para los diferentes algoritmos, en función del tiempo de computación incurrido durante el reinicio en el que se obtuvieron.

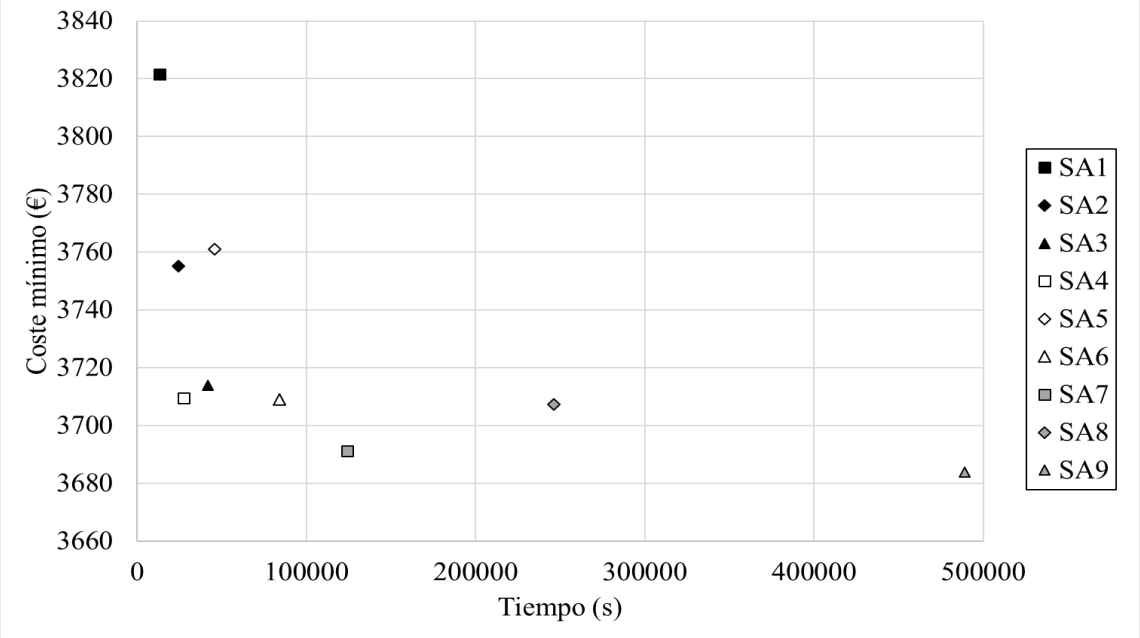


Figura 5.3: Coste mínimo del marco prefabricado en función del tiempo de computación asociado de cada uno de los distintos algoritmos SA aplicados (Elaboración propia).

Es viable afirmar que, con carácter general, las heurísticas que conllevan un mayor tiempo de computación consiguen alcanzar menores costes. Por otra parte, se puede observar que algunos de los algoritmos con menor coste computacional arrojan soluciones de gran calidad.

Sin embargo, es relevante destacar que la evaluación de los valores mínimos no permite llevar a cabo comparaciones representativas entre las diferentes heurísticas. Es por esto que es especialmente relevante comparar los valores medios tanto de coste como de tiempo de computación, algo que permite obtener una visión más representativa de los resultados. De esta forma, la figura 5.4 presenta el coste medio en función del tiempo medio de ejecución para cada una de los nueve algoritmos SA aplicados.

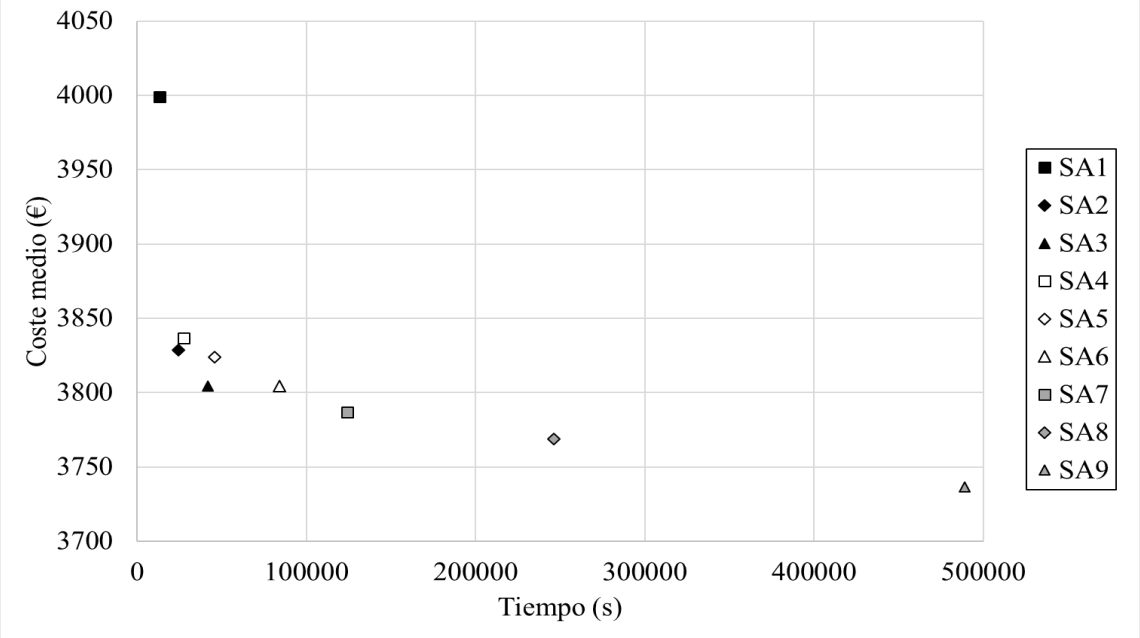


Figura 5.4: Coste medio del marco prefabricado en función del tiempo medio de computación de cada uno de los distintos algoritmos SA aplicados (Elaboración propia).

El marco con un menor coste obtenido mediante la aplicación de un algoritmo de recocido simulado para la resolución del problema se corresponde con la quinta iteración de la heurística SA9. Con un coste de 3.683,84 euros, fue necesario un tiempo de computación acumulado de 74 horas, 5 minutos y 24 segundos para su obtención.

La tabla 5.5 contiene los valores de cada una de las 31 variables objeto de optimización, valores que permiten la completa definición de la solución de mejor calidad asociada al marco representado en la figura 5.5.

Tras esto, como cierre de la presentación de los resultados obtenidos mediante el uso de los distintos algoritmos de recocido simulado desarrollados, las tablas 5.6 a 5.8 contienen un resumen general de la totalidad de los resultados obtenidos en el proceso.

Descripción		**Variable**	**Valor**
Canto hastiales	(m)	h_H	0.40
Canto losa inferior	(m)	h_{LI}	0.40
Canto losa superior	(m)	h_{LS}	0.82
Tipo de hormigón	(MPa)	C	25
Tipo de acero	(MPa)	S	500
Arm. long. A1	(mm)	ϕ_{A_1}	16
	$(barras)$	n_{A_1}	7
Arm. long. A2	(mm)	ϕ_{A_2}	12
	$(barras)$	n_{A_2}	7
Arm. long. A3	(mm)	ϕ_{A_3}	10
	$(barras)$	n_{A_3}	8
Arm. long. A4	(mm)	ϕ_{A_4}	16
	$(barras)$	n_{A_4}	7
Arm. long. A5	(mm)	ϕ_{A_5}	20
	$(barras)$	n_{A_5}	9
Arm. de ref. A6	(mm)	ϕ_{A_6}	20
	$(barras)$	n_{A_6}	9
Arm. long. A7	(mm)	ϕ_{A_7}	12
	$(barras)$	n_{A_7}	11
Arm. long. A12	(mm)	$\phi_{A_{12}}$	20
	$(barras)$	$n_{A_{12}}$	5
Arm. long. A13	(mm)	$\phi_{A_{13}}$	10
	$(barras)$	$n_{A_{13}}$	4
Arm. de ref. A14	(mm)	$\phi_{A_{14}}$	12
	$(barras)$	$n_{A_{14}}$	8
Arm. de ref. A15	(mm)	$\phi_{A_{15}}$	10
	$(barras)$	$n_{A_{15}}$	4
Arm. de cort. A18	(mm)	$\phi_{A_{18}}$	32
	(m)	$s_{A_{18}}$	0.35
Arm. de cort A21	(mm)	$\phi_{A_{21}}$	10
	(m)	$s_{A_{21}}$	0.35

Tabla 5.5: Variables del marco prefabricado con menor coste resultado de la optimización mediante el algoritmo SA desarrollado.

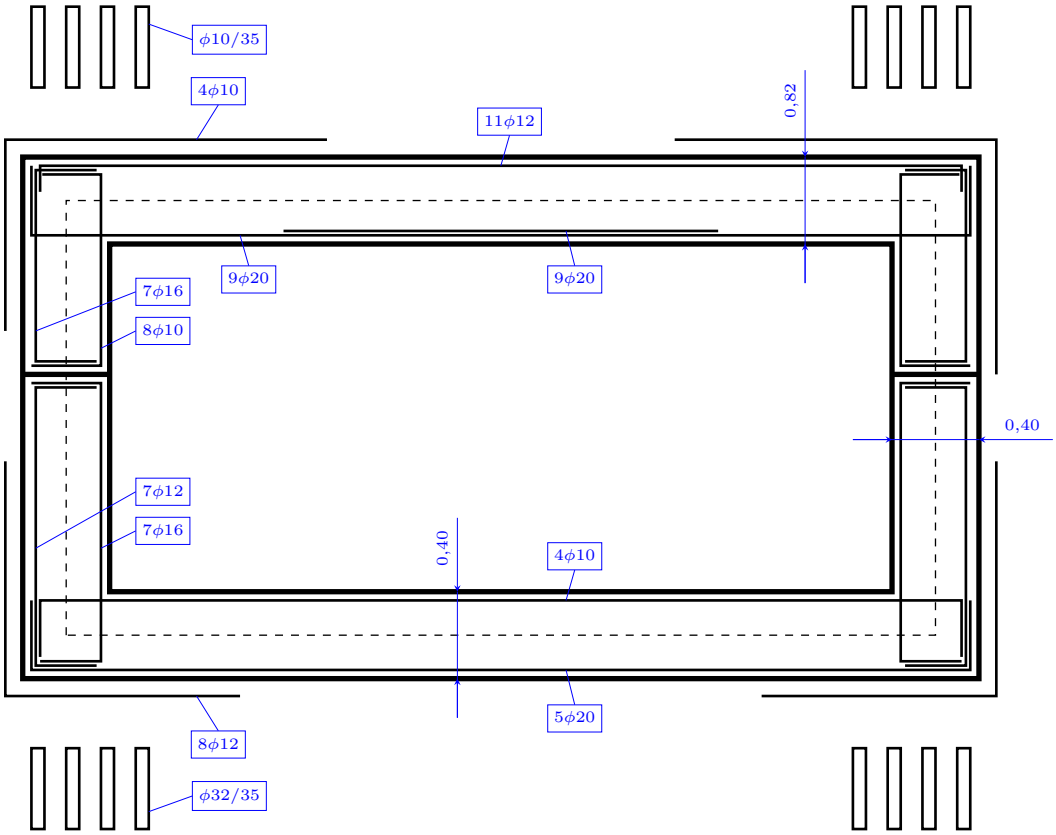


Figura 5.5: Ilustración del marco prefabricado con menor coste resultado de la optimización mediante el algoritmo SA desarrollado (Elaboración propia).

Longitud de cadena = 500 iteraciones						
	Coeficiente de enfriamiento					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.873,93	1.257,46	3.852,09	3.039,78	3.775,55	4.757,78
2	4.483,34	1.229,98	3.927,73	3.200,11	3.792,47	3.941,83
3	3.949,92	1.372,98	3.760,16	2.678,57	3.871,84	4.449,33
4	4.006,69	1.370,12	3.851,10	2.569,69	3.788,02	4.350,11
5	3.908,47	1.528,26	3.838,79	2.758,78	3.840,04	4.775,05
6	3.821,57	1.541,97	3.755,27	2.333,30	3.783,25	4.905,37
7	4.004,64	1.684,34	3.790,97	2.761,88	3.817,92	5.473,90
8	4.100,25	1.609,56	3.842,54	2.585,33	3.860,75	4.765,01
9	3.880,94	1.542,29	3.840,76	2.600,18	3.713,99	4.559,99
C. mínimo	3.821,57	1.541,97	3.755,27	2.333,30	3.713,99	4.559,99
C. medio	3.999,13	1.451,88	3.828,49	2.714,50	3.804,59	4.647,38

Tabla 5.6: Resultados del algoritmo de recocido simulado (SA) para una longitud de cadena de Markov de 500 iteraciones.

Longitud de cadena = 1000 iteraciones						
	Coeficiente de enfriamiento					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.900,67	3.007,48	3.846,04	5.305,29	3.824,78	10.002,63
2	3.709,59	3.140,26	3.806,37	5.257,97	3.851,55	9.838,66
3	3.947,79	3.043,06	3.790,06	5.186,84	3.725,28	9.077,43
4	3.788,62	2.939,31	3.764,58	5.204,75	3.708,99	9.188,82
5	3.772,46	3.588,33	3.762,18	5.053,51	3.810,49	9.912,85
6	3.860,66	3.098,34	3.882,41	4.910,15	3.821,23	8.829,53
7	3.875,76	2.933,96	3.981,72	5.226,23	3.819,42	8.864,13
8	3.794,70	2.858,97	3.761,14	4.606,76	3.843,32	8.968,33
9	3.887,73	2.981,01	3.824,73	5.135,78	3.837,69	9.342,58
C. mínimo	3.709,59	3.140,26	3.761,14	4.606,76	3.708,99	9.188,82
C. medio	3.836,89	3.059,44	3.823,76	5.094,23	3.804,44	9.325,92

Tabla 5.7: Resultados del algoritmo de recocido simulado (SA) para una longitud de cadena de cadena de Markov de 1000 iteraciones.

Longitud de cadena = 5000 iteraciones						
	Coeficiente de enfriamiento					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.805,72	15.245,31	3.751,73	29.714,11	3.774,38	55.024,72
2	3.823,34	14.501,79	3.847,22	28.574,27	3.699,51	54.303,55
3	3.810,26	14.011,98	3.820,43	26.886,72	3.819,86	51.866,17
4	3.817,03	14.449,62	3.707,37	28.252,66	3.767,13	53.993,98
5	3.817,99	11.914,92	3.804,61	26.563,56	3.683,84	51.535,49
6	3.845,26	12.970,27	3.755,92	27.926,85	3.755,78	54.092,16
7	3.691,30	13.893,95	3.783,16	27.545,16	3.720,51	58.112,41
8	3.770,26	12.974,09	3.707,36	25.268,31	3.703,39	55.221,91
9	3.704,37	14.359,18	3.745,12	25.406,64	3.707,37	55.092,16
C. mínimo	3.691,30	13.893,95	3.707,36	25.268,31	3.683,84	51.535,49
C. medio	3.786,93	13.779,10	3.786,94	27.313,61	3.736,63	54.329,57

Tabla 5.8: Resultados del algoritmo de recocido simulado (SA) para una longitud de cadena de cadena de Markov de 5000 iteraciones.

5.2.2. Algoritmo de aceptación por umbrales (TA)

Dando paso ahora a los resultados obtenidos mediante la aplicación del algoritmo TA en la resolución del problema, la tabla 5.9 presenta cada una de las diferentes configuraciones consideradas. Es relevante mencionar que todas ellas cuentan con el mismo tipo de movimiento que, al igual que en el caso del algoritmo SA, modifica un número máximo de cinco variables.

En el caso del algoritmo TA el menor coste ha sido de 3.675,73 euros, obtenido en el séptimo reinicio del algoritmo TA2, la obtención de este óptimo conlleva un tiempo computacional acumulado de 10 horas, 44 minutos y 32 segundos.

Heurística	Cadena	Coef. reducción	Tiempo (s)	Coste mínimo (€)
TA1	500	0.80	16.599,61	3.805,03
TA2	500	0.90	27.982,12	3.675,73
TA3	500	0.95	46.262,48	3.719,86
TA4	1000	0.80	29.237,13	3.759,20
TA5	1000	0.90	47.026,41	3.766,73
TA6	1000	0.95	79.795,56	3.705,82
TA7	5000	0.80	145.426,05	3.682,57
TA8	5000	0.90	211.780,40	3.697,23
TA9	5000	0.95	484.939,59	3.678,59

Tabla 5.9: Parámetros definitorios de cada una de las heurísticas TA aplicadas, junto con los costes mínimos de la mejor solución encontrada por cada una de ellas.

En la figura 5.6 se puede observar el coste mínimo obtenido mediante cada uno de los algoritmos TA desarrollados en función del tiempo de computación incurrido en ese reinicio.

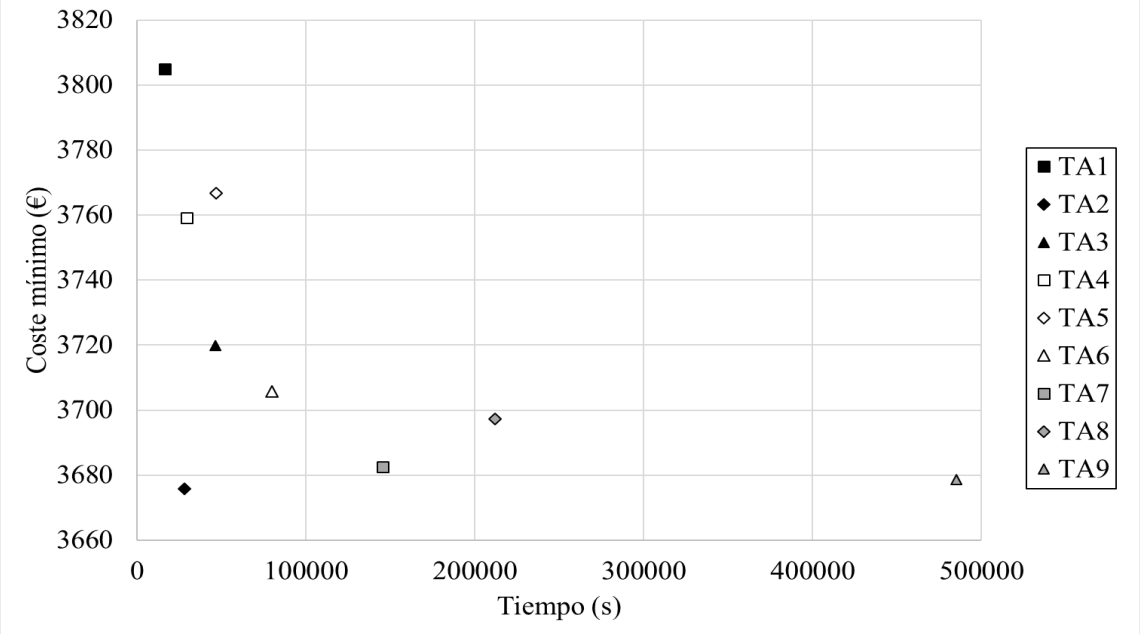


Figura 5.6: Costes mínimos del marco en función del tiempo asociado para cada uno de los distintos algoritmos TA programados (Elaboración propia).

Sin embargo, como se mencionó anteriormente, el valor mínimo de cada uno de los algoritmos TA no es representativo de la calidad de las soluciones obtenidas mediante su uso. Con el objetivo de poder hacer comparaciones representativas y, consecuentemente, obtener conclusiones adecuadas relativas a las diferentes parametrizaciones, es preferible comparar los valores medios. En este contexto, la figura 5.7 presenta el coste medio en función del tiempo de computación medio para cada uno de los algoritmos TA desarrollados.

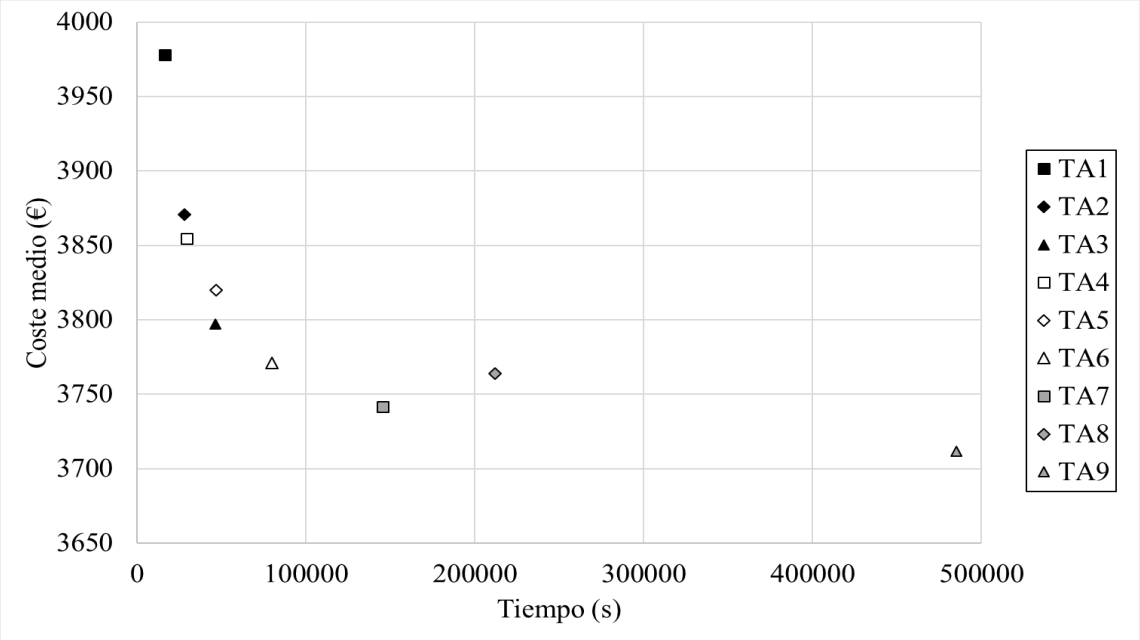


Figura 5.7: Costes medios del marco en función del tiempo medio para cada uno de los distintos algoritmos TA programados (Elaboración propia).

La tabla 5.10 contiene los resultados de las nueve ejecuciones llevadas a cabo para el algoritmo TA9. Se puede observar que la desviación típica tanto del coste como del tiempo de ejecución tiene un valor adecuado, algo que permite asegurar la robustez tanto del algoritmo como del método de aplicación del mismo.

La evolución del coste mínimo y tiempo de computación asociados en función del reinicio queda representado en la figura 5.8.

Reinicio	Coste (€)	Tiempo (s)
1	3.736,28	52.171,96
2	3.688,75	56.298,01
3	3.729,34	55.212,90
4	3.697,35	55.611,22
5	3.749,69	54.483,10
6	3.684,50	50.959,33
7	3.702,96	49.550,99
8	3.738,42	57.889,00
9	3.678,59	52.763,06
Valor medio	3.711,68	53.821,43
Valor mínimo	3.678,59	49.550,99
Desviación típica	26,73	2703,98

Tabla 5.10: Evolución de la heurística TA9 para cada uno de los nueve reinicios.

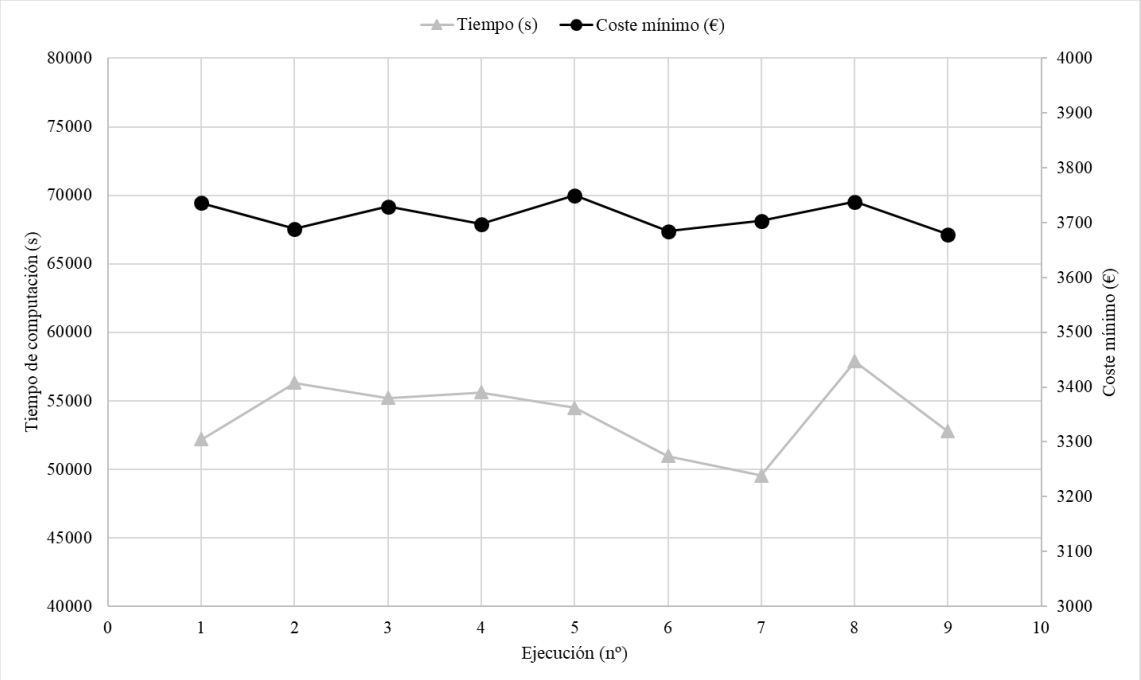


Figura 5.8: Evolución del coste mínimo y el tiempo de computación de la heurística TA9 para cada uno de los nueve reinicios (Elaboración propia).

Descripción		**Variable**	**Valor**
Canto hastiales	(m)	h_H	0.40
Canto losa inferior	(m)	h_{LI}	0.40
Canto losa superior	(m)	h_{LS}	0.80
Tipo de hormigón	(MPa)	C	25
Tipo de acero	(MPa)	S	500
Arm. long. A1	(mm)	ϕ_{A_1}	16
	$(barras)$	n_{A_1}	7
Arm. long. A2	(mm)	ϕ_{A_2}	12
	$(barras)$	n_{A_2}	6
Arm. long. A3	(mm)	ϕ_{A_3}	10
	$(barras)$	n_{A_3}	9
Arm. long. A4	(mm)	ϕ_{A_4}	16
	$(barras)$	n_{A_4}	7
Arm. long. A5	(mm)	ϕ_{A_5}	25
	$(barras)$	n_{A_5}	6
Arm. de ref. A6	(mm)	ϕ_{A_6}	20
	$(barras)$	n_{A_6}	9
Arm. long. A7	(mm)	ϕ_{A_7}	12
	$(barras)$	n_{A_7}	10
Arm. long. A12	(mm)	$\phi_{A_{12}}$	16
	$(barras)$	$n_{A_{12}}$	8
Arm. long. A13	(mm)	$\phi_{A_{13}}$	10
	$(barras)$	$n_{A_{13}}$	4
Arm. de ref. A14	(mm)	$\phi_{A_{14}}$	16
	$(barras)$	$n_{A_{14}}$	5
Arm. de ref. A15	(mm)	$\phi_{A_{15}}$	10
	$(barras)$	$n_{A_{15}}$	4
Arm. de cort. A18	(mm)	$\phi_{A_{18}}$	32
	(m)	$s_{A_{18}}$	0.35
Arm. de cort A21	(mm)	$\phi_{A_{21}}$	8
	(m)	$s_{A_{21}}$	0.40

Tabla 5.11: Variables del marco prefabricado con menor coste resultado de la optimización mediante el algoritmo TA desarrollado.

Dando paso a la presentación del marco con el menor coste obtenido mediante la aplicación del algoritmo TA en la resolución del problema, la tabla 5.11 presenta el valor concreto de cada una de las 31 variables objeto de optimización. La figura 5.9 se corresponde con la representación de dicho marco.

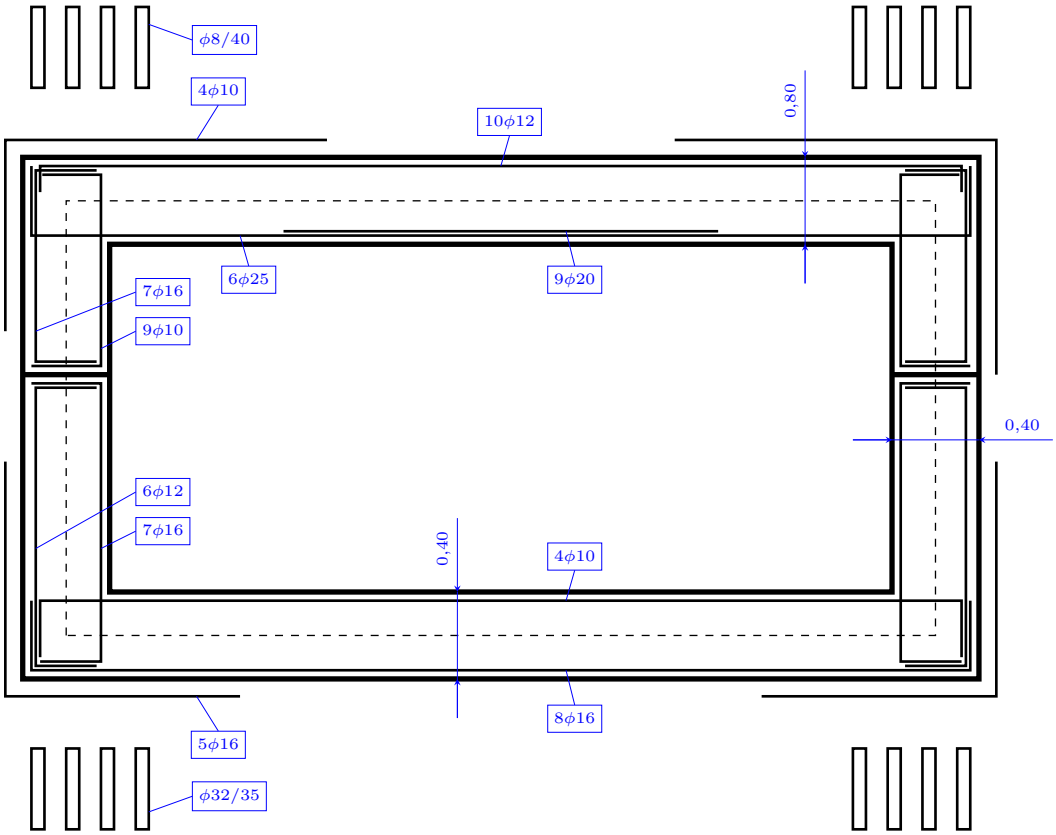


Figura 5.9: Ilustración del marco prefabricado con menor coste resultado de la optimización mediante el algoritmo TA desarrollado (Elaboración propia).

Finalmente, las tablas 5.10 a 5.12 contienen la totalidad de los resultados obtenidos mediante cada una de las configuraciones desarrolladas en el apartado. De forma general puede observarse que los valores obtenidos son adecuados al problema, sin la presencia de valores anómalos resultado de una calibración inadecuada del algoritmo.

Longitud de cadena = 500 iteraciones

	Coeficiente de reducción de umbral					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.873,28	1.721,76	3.779,23	2.890,84	3.822,24	5.956,90
2	3.816,16	2.058,49	3.811,81	3.279,98	3.822,48	5.243,62
3	4.072,71	2.472,77	4.227,55	3.053,79	3.776,87	5.583,24
4	4.038,57	1.937,21	3.965,54	3.196,77	3.723,10	5.463,68
5	3.924,76	1.612,80	3.971,33	3.636,46	3.854,27	5.577,87
6	4.264,60	1.585,80	3.841,56	3.074,89	3.812,91	5.427,09
7	3.826,14	1.724,24	3.675,73	2.939,19	3.719,86	4.914,39
8	3.805,03	1.781,87	3.749,65	3.282,79	3.836,48	4.240,09
9	4.211,93	1.704,67	3.838,88	2.627,42	3.808,59	3.855,60
C. mínimo	3.805,03	1.781,87	3.675,73	2.939,19	3.719,86	4.914,39
C. medio	3.978,13	1.827,55	3.870,50	3.097,38	3.797,15	5.095,91

Tabla 5.12: Resultados del algoritmo de aceptación por umbrales (TA) para una longitud de cadena de 500 iteraciones.

Longitud de cadena = 1000 iteraciones

	Coeficiente de reducción de umbral					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.759,21	3.457,62	3.766,73	5.589,24	3.782,54	9.785,31
2	3.870,93	3.443,53	3.849,44	5.708,44	3.735,23	9.288,41
3	3.842,46	3.323,80	3.849,99	5.639,74	3.816,60	10.078,01
4	3.851,92	3.223,77	3.767,00	5.423,47	3.778,19	9.519,56
5	3.876,89	3.446,48	3.783,80	5.896,78	3.856,71	10.477,11
6	3.834,35	3.338,51	3.797,68	4.323,53	3.736,08	7.115,02
7	3.906,19	3.093,54	3.831,54	5.093,85	3.757,74	7.795,24
8	3.951,53	3.315,83	3.860,45	4.741,74	3.705,83	7.389,91
9	3.800,08	2.594,05	3.872,48	4.609,62	3.774,31	8.347,00
C. mínimo	3.759,21	3.457,62	3.766,73	5.589,24	3.705,83	7.389,91
C. medio	3.854,47	3.237,37	3.819,70	5.197,98	3.771,23	8.787,80

Tabla 5.13: Resultados del algoritmo de aceptación por umbrales (TA) para una longitud de cadena de 1000 iteraciones.

Longitud de cadena = 5000 iteraciones						
	Coeficiente de reducción de umbral					
	0.8		0.9		0.95	
Ejecución	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**	**Coste (€)**	**Tiempo (s)**
1	3.760,22	16.370,04	3.697,23	28.310,40	3.736,28	52.171,96
2	3.798,66	16.363,76	3.802,15	21.540,70	3.688,75	56.298,01
3	3.684,01	16.027,94	3.727,93	23.759,66	3.729,34	55.212,90
4	3.794,98	16.455,70	3.712,15	22.390,34	3.697,35	55.611,22
5	3.735,98	15.162,70	3.740,28	22.702,04	3.749,69	54.483,10
6	3.682,57	15.672,13	3.799,09	23.851,69	3.684,50	50.959,33
7	3.728,27	16.574,45	3.824,16	24.050,41	3.702,96	49.550,99
8	3.765,63	16.524,79	3.773,08	23.634,47	3.738,42	57.889,01
9	3.723,99	16.274,53	3.802,15	21.540,70	3.678,59	52.763,06
C. mínimo	3.682,57	15.672,13	3.697,23	28.310,40	3.678,59	52.763,06
C. medio	3.741,38	16.152,32	3.764,00	23.458,00	3.711,68	53.821,43

Tabla 5.14: Resultados del algoritmo de aceptación por umbrales (TA) para una longitud de cadena de 5000 iteraciones.

5.2.3. Algoritmo del solterón (OBA)

Dando paso a la presentación de los resultados obtenidos mediante la aplicación de la tercera y última técnica heurística, es relevante mencionar que, durante el desarrollo del proceso de calibración del algoritmo OBA, se hizo evidente el hecho de que esta heurística proporcionaba resultados de una calidad considerablemente inferior a las anteriores.

Heurística	**Criterio parada**	**Coef. inc. umbral**	**Coef. red. umbral**
OBA	50.000 iteraciones	$\Delta_{(+)} = 1$€	$\Delta_{(-)} = 5$€

Tabla 5.15: Parámetros definitorios del algoritmo OBA desarrollado.

Esto dio lugar al extenso proceso de calibración y modificación del algoritmo detallado en capítulos anteriores, llegando a conseguir resultados que, si bien eran de mejor calidad que los iniciales, no son comparables con los obtenidos mediante la aplicación de las heurísticas SA y TA.

En este contexto, el algoritmo OBA aplicado tras las modificaciones y proceso de calibración detalladas en capítulos anteriores, cuenta con las características presentadas.

Mediante la aplicación de este algoritmo, se obtienen resultados asociados a nueve reinicios diferenciados. Estos resultados quedan resumidos en la tabla 5.16, y representados en la figura 5.10.

De esta forma, se concluye la presentación general de los resultados obtenidos mediante la aplicación de las diferentes técnicas heurísticas consideradas en la optimización económica del marco prefabricado.

Reinicio	**Coste (€)**	**Tiempo** s
1	3.909,94	18.046,68
2	3.804,83	14.859,00
3	3.808,72	14.850,59
4	3.847,45	14.831,98
5	3.819,65	20.159,23
6	3.874,68	18.752,80
7	3.830,58	17.428,04
8	3.866,87	14.889,60
9	3.934,59	14.786,09
Valor medio	3.855,02	16.396,90
Valor mínimo	3.804,83	14.859,00
Desviación típica	231,16	2.104,87

Tabla 5.16: Evolución de los resultados para los nueve reinicios del algoritmo OBA aplicado.

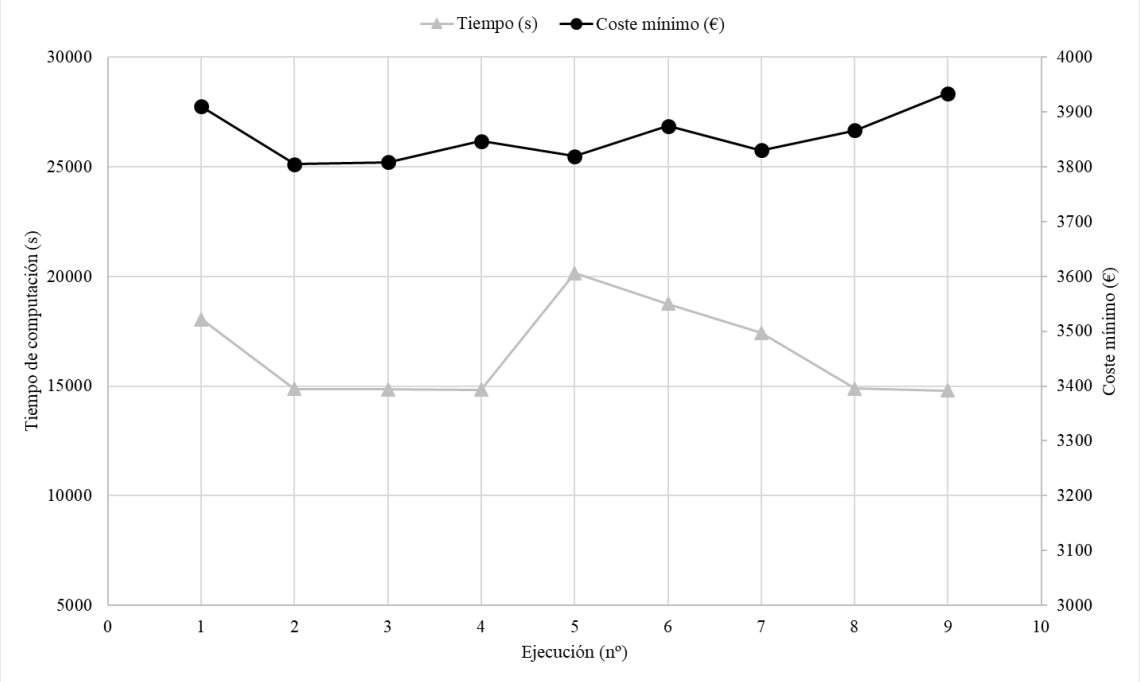


Figura 5.10: Evolución del coste mínimo y tiempo de computación asociado de cada uno de los nueve reinicios del algoritmo OBA (Elaboración propia).

5.3. Discusión de los resultados

Una vez presentados los resultados, es necesario llevar a cabo un análisis que permita desarrollar conclusiones representativas. En este contexto, a modo de resumen de los resultados ya presentados, la figura 5.11 se corresponde con los costes medios en función de los costes medios obtenidos mediante la aplicación de cada algoritmo.

A la hora de evaluar de forma adecuada los distintos algoritmos aplicados en la optimización económica del marco, es necesario considerar que este ha de satisfacer varios objetivos de forma conjunta. La heurística debe ser evaluada considerando, no sólo la calidad de las soluciones obtenidas, algo que consiste en este caso en evaluar el coste económico final del marco, sino también su rapidez, es decir, el tiempo de computación incurrido durante el proceso de obtención de ese resultado.

Esto constituye una evaluación multiobjetivo en la que ambas consideraciones, como se ha visto anteriormente, entran en conflicto. En este contexto es relevante introducir el concepto de óptimo de Pareto, propio de la economía, este es aplicable

tanto a ciencias sociales como a la ingeniería. Se considera eficiente aquella situación en la que no es posible beneficiar a más individuos de un sistema sin perjudicar a otros.

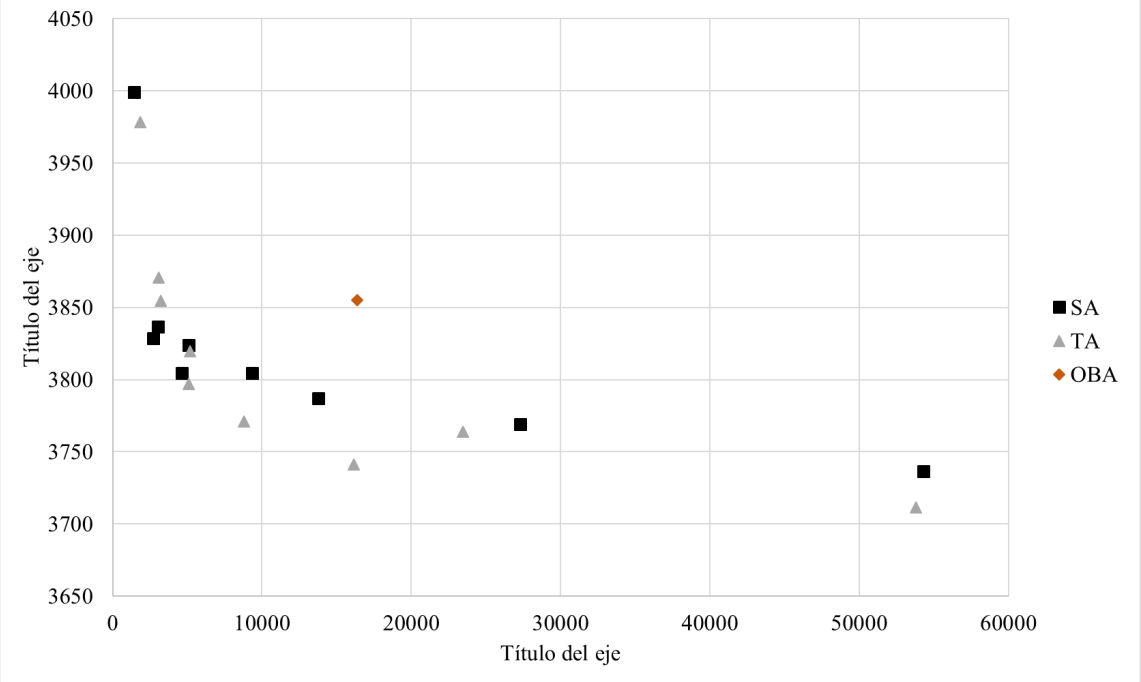


Figura 5.11: Coste medio en función del tiempo medio de computación para el conjunto global de los algoritmos aplicados (Elaboración propia).

En el caso de estudio, un óptimo de Pareto será aquella solución óptima que permita reducir el coste, sin incurrir en un tiempo de computación superior. La unión de cada uno de los distintos marcos que cumplen esta condición permite trazar las denominadas fronteras de Pareto, algo representado en la figura 5.12.

El análisis de estas fronteras es de especial relevancia ya que se trata de valores que cuentan con las mejores características conjuntas de coste mínimo, y menor tiempo de computación. En el caso de estudio la frontera está compuesta por tres algoritmos SA y cuatro algoritmos TA.

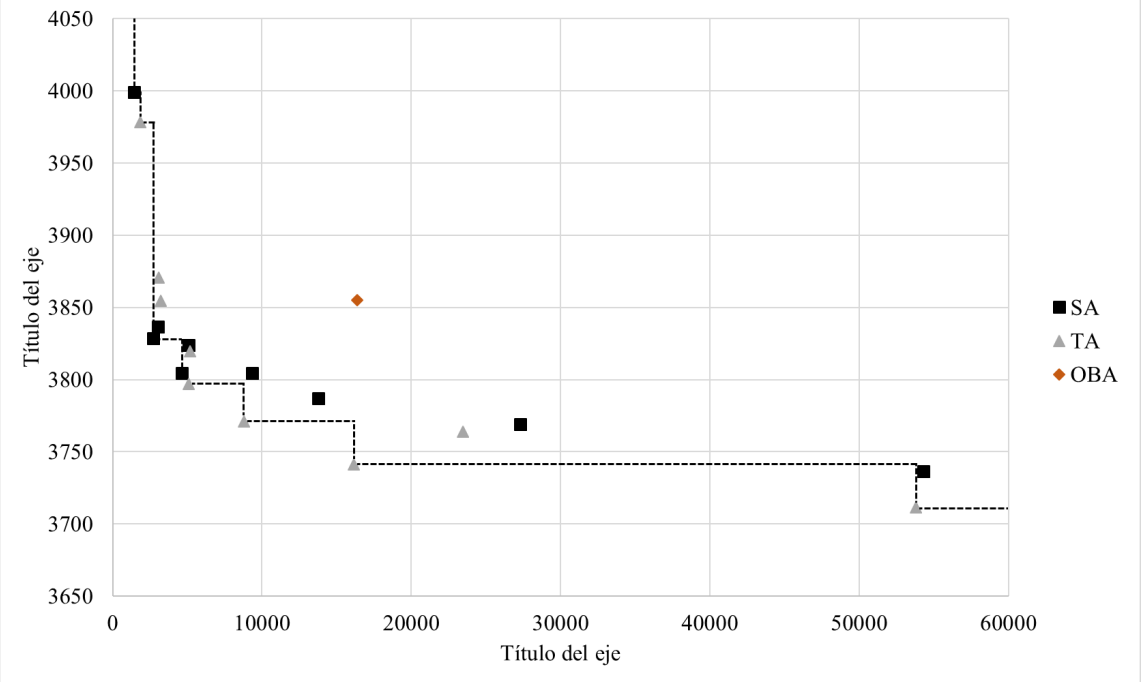


Figura 5.12: Representación gráfica de la frontera de Pareto del problema (Elaboración propia).

Dentro de cada uno de los algoritmos que establece esta frontera, el algoritmo TA7 es especialmente interesante ya que logra obtener un óptimo de calidad muy similar al mínimo, sin incurrir en un coste computacional elevado. De igual forma puede verificarse lo enunciado anteriormente en lo que al algoritmo OBA respecta, ya que este se sitúa en una zona lejana a la frontera, algo que denota la menor calidad de los resultados obtenidos frente a los algoritmos SA y TA.

En lo que a las características de los marcos óptimos respecta, es interesante mencionar que el conjunto global de los marcos con menor coste para cada una de las heurísticas consideradas presenta cantos reducidos respecto a diseños tradicionales.

En el caso del algoritmo de recocido simulado, el marco óptimo tiene un coste final de 3.683,84 euros, las principales características geométricas de este son un canto de 0,40 metros tanto en hastiales como en la losa inferior, tomando este un valor de 0,82 metros en el caso de la losa superior. El marco es fabricado con hormigón de 25 MPa, y armaduras pasivas de 500 MPa.

Por otra parte, la aplicación del algoritmo de aceptación por umbrales deriva

en la obtención de un marco con un coste final ligeramente inferior, siendo este 3.675,73 euros. El marco óptimo comparte ciertas de las características geométricas mencionadas en el caso del resultado de la optimización mediante SA, con un canto de 0,40 metros en hastiales y losa inferior, y 0,80 metros en el caso de la losa superior. Además, los materiales usados en su construcción son los mismos, haciendo uso de hormigón de 25 MPa y armaduras pasivas de 500 MPa.

Finalmente, en el caso del algoritmo del solterón, el marco óptimo tiene un coste algo superior, de 3.804,83 euros. En este caso la losa inferior cuenta con un canto de 0,42 metros, 0,40 metros en el caso de los hastiales, y 0,82 metros en el caso de la losa superior. Una vez más, los materiales usados en la construcción del marco óptimo son hormigón de 25 MPa y acero de 500 MPa.

El conjunto de soluciones óptimas conseguido mediante la aplicación de cada una de las tres técnicas heurísticas consideradas cuenta con una serie de características comunes. En los tres casos el marco se fabrica con hormigón de tipo HA-25, así como con armaduras pasivas de grado B500, el uso general de estos materiales es indicativo de que, con los requerimientos geométricos y estructurales establecidos, estos materiales conforman la elección óptima en lo que a la fabricación del marco respecta.

Por otra parte, una de las principales características del conjunto de soluciones con menor coste es la esbeltez de las secciones de hastiales y losa inferior. En un número considerable de ocasiones el algoritmo concluye que el menor de los valores permitidos conforma la elección óptima en lo que a canto de hastiales y losa inferior respecta. Esta reducción en el canto se ve sustentada por un incremento en la cuantía de armadura pasiva. En vista de estos resultados, una consecuente reducción en el límite inferior de estas variables podría permitir que el algoritmo alcance soluciones válidas aún más esbeltas donde la disposición de armado sea aún mayor.

Los valores habituales para el canto de la losa superior e inferior en la tipología estructural considerada se sitúa entre una décima y una quinceava parte de la luz a salvar. En el caso del marco estos valores se corresponden con un canto de 1 metro y 0,67 metros. En el caso de la losa superior de los marcos óptimos, su canto se encuentra dentro del rango de valores habituales, sin embargo, la losa inferior presenta cantos considerablemente inferiores, siendo un 40 % inferior al mínimo de

los valores habituales. Por otra parte, los hastiales suelen presentar un canto de magnitud inferior a la considerada en las losa superior, o una doceava parte de la luz salvada, algo equivalente a 0,83 metros en el caso de estudio. Las soluciones óptimas pues, muestran una esbeltez característica, solventada mediante un diseño de armado considerablemente denso.

Finalmente, es relevante mencionar que, en la solución de menor coste, correspondiente al algoritmo de aceptación por umbrales TA2, el ratio de armadura pasiva para el conjunto general de la estructura es de 91,01 kg/m^3. En conjunto con la serie de características generales de los marcos óptimos descritos anteriormente, es viable afirmar que los resultados que presentan un menor coste cuentan con secciones esbeltas y un diseño de armado denso.

Capítulo 6

Conclusiones y futuras líneas de investigación

Como cierre del presente trabajo, este capítulo es dedicado a enumerar una serie de conclusiones derivadas del estudio llevado a cabo, así como diferentes líneas de investigación que, tras la finalización del mismo, serán desarrolladas en estudios posteriores.

El objetivo general del estudio no es otro que la mejora y ampliación del conocimiento actual sobre las técnicas de optimización de estructuras de hormigón armado. En este contexto, se optó por, como fase inicial de un proyecto de mayor magnitud, llevar a cabo la optimización económica de un marco prefabricado mediante la aplicación de tres técnicas heurísticas cuya aplicación en estudios similares ha permitido obtener resultados de gran calidad. La optimización económica del marco comprende, tanto el desarrollo de software propio que permita un posterior desarrollo de la línea de investigación, como la aplicación de las tres técnicas heurísticas seleccionadas.

En este contexto, este capítulo es dedicado a mostrar un resumen general del contenido presentado, mostrar la novedad aportada por el trabajo desarrollado, así como a enunciar las principales conclusiones. Como cierre se exponen una serie de consideraciones referentes a la continuidad del proyecto, enunciando futuras líneas de investigación.

6.1. Resumen del contenido

El primer capítulo del trabajo establece el marco de la investigación desarrollada. Esto comprende la contextualización y motivación del estudio, el correcto establecimiento del alcance y los objetivos de la investigación, así como la completa estructuración del presente documento, resultado de la misma.

El segundo capítulo introduce, de forma general, los problemas de optimización y el modelo matemático inherente a su planteamiento, así como una serie de técnicas aplicables en su resolución. Además, se particulariza este tipo de problema, contextualizando el lugar que toma el proceso de diseño estructural dentro del ámbito de la optimización. Tras la introducción del problema, una gran parte del capítulo es dedicada al desarrollo y completa definición de algunas de las técnicas aplicadas hasta el momento en la resolución de este tipo de problemas. En conjunto con el estudio del estado del arte, la completa definición de los algoritmos más relevantes permite una posterior selección fundamentada de las técnicas metaheurísticas aplicadas en la optimización económica del marco prefabricado.

En el tercer capítulo se da paso a la descripción particularizada del problema considerado en el estudio. Inicialmente, se fundamenta la elección de la tipología estructural considerada, haciendo mención de algunos de los casos de uso así como posibles tipologías relacionadas. Tras esto, el capítulo se centra en la completa definición del problema de optimización, detallando los distintos parámetros, variables y restricciones, así como la función objetivo considerada. Además, se establecen los rangos de valores que estos pueden tomar en el proceso de optimización, fundamentando las hipótesis y simplificaciones consideradas.

Tras esto, el cuarto capítulo detalla en profundidad las características de cada una de las distintas técnicas heurísticas consideradas, así como el proceso de adecuación y calibración aplicado a cada algoritmo. La segunda parte del capítulo constituye una completa descripción de los fundamentos teóricos, así como del funcionamiento interno del software desarrollado para el cálculo, verificación y optimización económica del marco. Como cierre del capítulo, se menciona el dispositivo utilizado para el desarrollo y obtención de resultados.

Dando paso al quinto capítulo, este es dedicado a la presentación general, análisis y discusión, de los resultados obtenidos en el estudio. Esto comprende la completa

justificación del número de ejecuciones realizadas para cada una de las distintas configuraciones de los algoritmos considerados. De esta forma, en conjunto con el presente capítulo, este documento constituye una recolección general de los resultados obtenidos durante el desarrollo de la línea de investigación.

6.2. Novedad y aporte original del trabajo

A lo largo del documento se ha hecho referencia a una serie de trabajos previos elaborados por el grupo de investigación que promueve este proyecto. Dichas investigaciones aplican metodologías similares en el estudio de diversas estructuras, obteniendo resultados de alta calidad [12–14, 19]. Entre estos es especialmente relevante destacar el trabajo desarrollado en Perea (2008) [81], donde se lleva a cabo el estudio y optimización económica de estructuras con características similares a las consideradas en el presente estudio. En este contexto es relevante destacar las diferencias, mejoras y ampliaciones que el presente estudio plantea, quedando justificada de esta forma la ampliación del conocimiento actual al respecto, así como la aportación original del autor. Las estructuras consideradas en estudios anteriores consistían en marcos de hormigón armado ejecutados in situ. El problema de optimización planteado partía, para todas las heurísticas aplicadas, de una solución inicial propuesta anteriormente, algo que condiciona el resultado final del proceso. Estas eran evaluadas mediante las expresiones establecidas en la Norma EHE-08 [105], algo que muestra, en cierta medida, la necesidad de desarrollar una línea de investigación similar que considere las normas vigentes actualmente.

De esta forma, considerando que la prefabricación permite obtener piezas con una calidad superior gracias al ambiente controlado de fabricación y curado, el presente estudio centra sus esfuerzos en la mejora del diseño de este tipo de estructuras prefabricadas. Esta consideración abre la puerta al estudio de una tipología estructural diferente. En esta, la unión articulada de las piezas que lo componen supone un factor de desarrollo de la investigación especialmente interesante, pasando a ser su posición en los hastiales una nueva variable cuyo valor sea objeto de optimización en estudios posteriores. En este contexto, considerando el hecho de que el diseño de una estructura prefabricada será iterado y utilizado en un número muy superior al

de una estructura particular ejecutada in situ, es viable afirmar que una mejora en el diseño de la tipología estructural considerada puede llegar a derivar en reducciones superiores en las emisiones de gases contaminantes, así como conseguir ahorros económicos considerables.

El presente estudio fundamenta las restricciones estructurales en las normas vigentes en la actualidad, además el proceso de optimización planteado parte de iteraciones aleatorias, algo que reduce la predisposición del algoritmo a converger hacia soluciones con unas características determinadas.

El desarrollo de software propio de cálculo, verificación y optimización estructural conlleva una serie de mejoras tales como reducciones en los tiempos de computación. Además, gracias a la naturaleza generalista con la que se ha programado, el software desarrollado abre la puerta a la evaluación de diversas funciones objetivo, y lo que es más importante, a la consideración simultánea de varias de estas funciones. En conjunto con el hecho de que, con una correcta configuración, el software permite el estudio de cualquier tipología estructural, es viable afirmar que se trata de una herramienta de optimización multiobjetivo muy potente.

En vista de lo mencionado anteriormente, la aportación original del autor queda fundamentada en el desarrollo y posterior aplicación del software propio. Siendo además respaldada por la obtención de los resultados presentados en el estudio. En conjunto con algunas de las líneas de investigación abiertas durante el desarrollo del trabajo, enumeradas al final del capítulo, el trabajo constituye un aporte original que busca mejorar el conocimiento sobre la optimización de las estructuras de hormigón armado.

6.3. Conclusiones derivadas de la optimización económica del marco

La necesidad de evaluar y mejorar el consumo de recursos humanos en la industria de la construcción, en conjunto con la importancia social y económica asociada al desarrollo de la infraestructura de un país, fundamenta la necesidad de desarrollar líneas de investigación cuyos resultados deriven en un uso más responsable y sostenible de los materiales.

Conclusión 1: *Como ingenieros hemos de involucrarnos y tomar parte en la reducción de las consecuencias asociadas al consumo de recursos naturales por parte de la industria de la construcción. En este contexto, la optimización estructural conforma una herramienta de gran capacidad para la mejora del conocimiento sobre este tipo de problemas.*

La fase inicial del proyecto consistió en el estudio del estado del arte relativo a la optimización de estructuras de hormigón armado mediante la aplicación de técnicas metaheurísticas. El análisis de la literatura permite afirmar que, actualmente, la optimización genera un gran interés en el ámbito académico.

Conclusión 2: *La aplicación de técnicas heurísticas para la optimización del proceso de diseño estructural constituye un campo de investigación activo en desarrollo. Fundamentado en la dificultad intrínseca del cálculo, y, consecuentemente, del modelado de estructuras de hormigón armado, estas técnicas han sido aplicadas tradicionalmente a estructuras metálicas. Sin embargo, en la última década existe una clara tendencia hacia la consideración de estructuras de hormigón armado.*

En conjunto con la vasta experiencia de los tutores del trabajo, el correcto análisis de la literatura permitió determinar la falta de desarrollo, y consecuente necesidad de ampliar y mejorar el conocimiento actual sobre la tipología estructural considerada dentro del ámbito de la optimización.

Conclusión 3: *Los trabajos de optimización de marcos de hormigón armado llevados a cabo hasta el momento constituyen versiones muy simplificadas del problema real. Actualmente no existe literatura en la que el proceso de cálculo y verificación estructural se lleve a cabo haciendo uso de software propio, algo que permite una notable mejora en el proceso de obtención de resultados, así como la posibilidad de considerar diferentes configuraciones y tipologías estructurales.*

El establecimiento y completa definición de la línea de investigación a desarrollar, da paso a la necesidad intrínseca de definir una serie de objetivos y establecer un plan de trabajo que asegure su consecución. El presente trabajo conforma la fase inicial del estudio en profundidad de la optimización de marcos prefabricados de hor-

migón armado. Esto comprende la consideración de factores económicos, ambientales o sociales, así como el análisis del ciclo de vida de la estructura.

Conclusión 4: *La economía de una estructura es un indicador directo del uso de materiales incurrido en su construcción. Consecuentemente, la optimización económica de la tipología estructural considerada constituye un paso inicial adecuado en la mejora del conocimiento sobre la optimización de marcos de hormigón armado.*

El desarrollo de software propio de cálculo, verificación y optimización estructural conlleva, inicialmente, un proceso de análisis de las tareas asociadas al cálculo y verificación de la tipología estructural considerada, procesos inherentes al diseño de cualquier estructura de hormigón armado. Este análisis ha de permitir el desarrollo de un modelo matemático representativo de la realidad, siendo necesario definir de forma adecuada los parámetros, variables, y restricciones del problema de optimización.

Conclusión 5: *La elaboración de un modelo matemático representativo constituye una de las tareas más relevantes a la hora del desarrollo de software de cálculo y verificación estructural. El desarrollo de un modelo adecuado para la optimización del marco ha sido un paso fundamental en el desarrollo del estudio.*

El desarrollo de software propio cuenta con una serie de complejidades intrínsecas tanto al ya mencionado proceso de creación de un modelo representativo, como al propio proceso de programación de módulos óptimos que funcionen de forma correcta. De nuevo, es relevante destacar la importancia de haber llevado a cabo un estudio detallado del problema a resolver, de forma que tanto el modelo matemático como la estructura del software permitan la obtención de resultados representativos.

Dando paso a la selección de las tres técnicas heurísticas aplicadas en la resolución del problema de optimización económica del marco, el estudio del estado del arte hizo notable una clara tendencia hacia el uso de los algoritmos de recocido simulado y aceptación por umbrales. Por otra parte, la aplicación del algoritmo del solterón resultó interesante dado que su principio de funcionamiento es similar a los mencionados anteriormente, sin embargo, la calidad de los resultados no justifica la continuación de su uso.

Conclusión 6: *Las técnicas heurísticas de recocido simulado y aceptación por umbrales cuentan con una considerable capacidad en lo que a la resolución de problemas de optimización estructural respecta. La calidad de los resultados obtenidos mediante su aplicación justifica su consideración para el proceso de optimización del marco.*

Una vez desarrollado el software de cálculo y optimización estructura, así como seleccionadas las técnicas heurísticas a aplicar en la resolución del problema, es necesario establecer el proceso de obtención de resultados. Una correcta y completa definición de este permitirá asegurar la representatividad de los marcos obtenidos.

En lo que a la aplicación de las técnicas heurísticas seleccionadas respecta, el estudio, adecuación y correcta calibración de los parámetros definitorios de cada uno de los algoritmos ha demostrado ser una de las consideraciones más relevantes a la hora de obtener resultados de alta calidad.

Conclusión 7: *El movimiento diseñado para cada uno de los algoritmos permite la generación de nuevas soluciones de forma adecuada. La modificación de un número superior a cinco variables puede dar lugar a la pérdida de cualidades interesantes de la solución actual, por otra parte, un número inferior puede conllevar la convergencia prematura del algoritmo hacia un óptimo de baja calidad.*

Dando paso al proceso de obtención de resultados, es relevante mencionar que, considerando el alcance y la limitación temporal del presente trabajo, la aplicación de cada uno de los algoritmos se ha visto limitada. Si bien esto no modifica la calidad de las soluciones obtenidas, sí que conlleva la aceptación de un número finito de configuraciones a aplicar. En este contexto, el estudio estadístico detallado en la justificación del número de ensayos facilita la obtención de resultados, reduciendo notablemente el tiempo de computación.

Conclusión 8: *El coste computacional del proceso de optimización es un factor determinante en la obtención de resultados. En este contexto la realización de un estudio estadístico permite justificar la ejecución de un número finito de nueve ejecuciones, asegurando la representatividad de las soluciones obtenidas.*

La primera de las técnicas heurísticas aplicados en la optimización económica del

marco fue un algoritmo de recocido simulado. Durante las fases iniciales de aplicación fue necesario adecuar el valor de los parámetros definitorios al problema concreto del marco. Tras esto, la calidad de los resultados obtenidos respalda la robustez tanto del modelo como del algoritmo desarrollado. Algunas de las consideraciones más relevantes para la correcta obtención de soluciones pasan por la definición adecuado de la temperatura inicial del problema, así como el establecimiento de un criterio de parada.

Conclusión 9: *El proceso de establecimiento de la temperatura inicial basado en la tasa de aceptación permite un correcto funcionamiento del algoritmo. Tras las aplicaciones iniciales fue evidente la necesidad de establecer un criterio de parada que permitiera el desarrollo completo del algoritmo hacia fases de intensificación suficientemente largas.*

El algoritmo de aceptación por umbrales demostró tener un proceso de calibración muy similar al de recocido simulado. Al fin y al cabo el criterio de aceptación es la única de sus características que los diferencia. Por otra parte, los resultados obtenidos mediante el algoritmo OBA no presentan una calidad suficiente como para justificar el coste computacional asociado al proceso.

Conclusión 10: *Las técnicas heurísticas de recocido simulado y aceptación por umbrales permiten la obtención de resultados de una calidad elevada.*

Conclusión 11: *Con tiempos de computación superiores, el algoritmo del solterón no arroja soluciones de calidad similar. A pesar de que tras la modificación de su principio de funcionamiento los resultados fueron algo más alentadores, es necesario seguir desarrollando modificaciones que puedan dar lugar a la obtención de soluciones de mejor calidad.*

Finalmente, es relevante destacar que un posterior estudio paramétrico de las soluciones obtenidas a lo largo del trabajo, permitirá entender las características de interés de los marcos óptimos. Dentro de su alcance, el presente estudio ha permitido establecer las bases de esta línea de investigación, obteniendo resultados positivos que justifican la continuidad del proyecto.

Conclusión 12: *El desarrollo de software propio de cálculo, verificación y optimización estructural conforma una mejora considerable en las herramientas disponibles. El trabajo conforma una aportación original al grupo de investigación y conforma una base sólida que sirve como inicio de un proyecto de investigación con mayor alcance.*

6.4. Futuras líneas de investigación

Dando paso al cierre del trabajo, es relevante hacer mención de las diferentes líneas de investigación abiertas tras la finalización del presente estudio.

Como parte del presente estudio, se ha desarrollado un software de cálculo y comprobación estructural propio basado en el lenguaje de programación *Python*. Las características de este han sido adecuadas al alcance y necesidades actuales del estudio, algo que conlleva la aceptación de una serie de simplificaciones y adaptaciones del problema estudiado. Es por esto que una ampliación del estudio conlleva una inherente ampliación y mejora del software desarrollado.

Esta ampliación comprende la consideración de cargas asociadas al transporte y montaje del marco prefabricado, así como la consideración de ciertas mejoras en el proceso de modelado de las cargas ya existentes. Por otra parte, es especialmente interesante dar paso a la consideración del estado límite último de fatiga, proceso cuyo modelado tiene cierta complejidad, pero de especial relevancia en estructuras enterradas.

El software actual modela el medio elástico sobre el que se sitúa la losa inferior mediante una serie de muelles con rigidez equivalente asociada al módulo de balasto del terreno. Si bien es cierto que se trata de un modelo adecuado que permite la obtención de resultados veraces, es interesante mejorar este aspecto. Por esto, una línea de investigación que actualmente se está desarrollando consiste en el modelado del medio elástico mediante la resolución de las ecuaciones para vigas flotantes sobre medio elástico propuestas por J.A. Jiménez Salas en Geotecnia y Cimientos III [106]. La resolución y correcta aplicación de estas ecuaciones permite la obtención de una matriz de rigidez que ya considera el apoyo de la losa inferior sobre el medio elástico.

Otra de las líneas de ampliación del trabajo más evidentes consiste en la apli-

cación de un mayor número de técnicas heurísticas para la resolución del problema. El uso de nuevos algoritmos genéticos, así como algoritmos evolutivos puede proporcionar no sólo mejores soluciones al problema, sino información relevante sobre qué tipo de heurística permite obtener soluciones de mejor calidad. Además de esto, sería especialmente interesante el uso de redes neuronales ya que, como se indicó en el segundo capítulo, con un buen entrenamiento, estas cuentan con una gran capacidad y permitirían reducir en gran medida los tiempos de computación, así como la obtención de soluciones de gran calidad.

El estudio desarrollado cuenta con un límite temporal definido, algo que limita, en cierta medida, la aplicación de las heurísticas elegidas. En este contexto, además de la ya mencionada consideración de nuevos algoritmos, sería interesante trabajar con configuraciones que permitan a las heurísticas aplicadas (SA, TA y OBA) ser ejecutadas durante periodos temporales más largos. Esto consistiría, en el caso de las dos primeras, en establecer longitudes de cadena superiores, pudiendo llegar a las 50.000 iteraciones por cadena. En el caso del algoritmo OBA esto consistiría tanto en aumentar el límite de iteraciones como tratar de mejorar el algoritmo mediante modificaciones propias. Esta variación en los parámetros de las heurísticas debería ser acompañada de una mejora en los dispositivos usados para la resolución del problema ya que, como cabe esperar, un ordenador con características superiores reduce los tiempos de computación de forma considerable.

Finalmente, considerando la especial relevancia de los medidores de sostenibilidad de las estructuras tienen en la actualidad, es relevante mencionar otra de las líneas de investigación que actualmente se están desarrollando. Esta consiste en la consideración de optimización multicriterio en la que, además de la función objetivo asociada al coste, se consideran más funciones objetivo relacionadas con las emisiones de CO_2, o el consumo energético asociados a la creación, transporte y construcción de la estructura. Este tipo de consideraciones se ve enmarcada en el estudio del ciclo de vida completo de la estructura, algo comprendido en el estudio del Análisis de Ciclo de Vida o *Life Cycle Analysis* (LCA) de una estructura.

Bibliografía

[1] Wu Z.; Yu A.T.W.; Poon C.S. An off-site snapshot methodology for estimating building construction waste composition — a case study of hong kong. *Environmental Impact Assessment Review*, 77:128–135, 2019.

[2] Scherer A.G.; Palazzo G.; Seidl D. Managing legitimacy in complex and heterogeneous environments: Sustainable development in a globalized world. *Journal of Management Studies*, 50(2):259–284, 2013.

[3] Ramesh T.; Prakash R.; Shukla K. K. Life cycle energy analysis of buildings: An overview. *Energy Build*, 42(10):1592–1600, 2010.

[4] Liu S.; Tao R.; Tam C.M. Optimizing cost and co_2 emission for construction projects using particle swarm optimization. *Habitat International*, 37:155–162, 2013.

[5] Chong W.K.; Kumar S.; Haas C.T.; Beheiry S.M.A.; Coplen L. Oey M. Understanding and interpreting baseline perceptions of sustainability in construction among civil engineers in the united states. *Journal of Management in Engineering*, 25(3):143–154, 2009.

[6] Valdes-Vasquez R.; Klotz L. Social sustainability considerations during planning and design: A framework of processes for construction projects. *Journal of Construction Engineering and Management*, 139(1):80–89, 2013.

[7] Sierra L.; Pellicer E.; Yepes V. Social sustainability in the life cycle of chilean public infrastructure. *Journal of Construction Engineering and Management*, 142(5), 2016.

[8] Real Academia Española. Diccionario de la lengua española, definición de ingenio, 2022. URL https://dle.rae.es/ingenio.

[9] Real Academia Española. Diccionario de la lengua española, definición de ingeniero, 2022. URL https://dle.rae.es/ingeniero.

[10] Zhong Y.; Wu P. Economic sustainability, environmental sustainability and constructability indicators related to concrete- and steel-projects. *Journal of Cleaner Production*, 108:748–756, 2015.

[11] Yepes V. and Navarro Martínez I. *Trends in Sustainable Buildings and Infrastructure*. IJERPH, 2021.

[12] Julián Alcalá González. *Optimización Heurística Económica de Tableros de Puentes Losa Pretensados.* PhD thesis, Universidad Politécnica de Valencia, 2009.

[13] Yepes V.; Alcalá J.; Perea C.; González-Vidosa F. A parametric study of optimum earth-retaining walls by simulated annealing. *Engineering Structures*, 30(3):821–830, 2008.

[14] José Vicente Martí Albiñana. *Diseño Óptimo de Tableros Isostáticos de Vigas Artesas Prefabricadas Pretensadas.* PhD thesis, Universidad Politécnica de Valencia, 2010.

[15] Navarro I.J.; Martí J.V.; Yepes V. Anp-based sustainability-oriented indicator for bridges in aggressive environments. *26th International Congress on Project Management and Engineering (AEIPRO)*, 2022.

[16] Navarro I.J.; Villalba I.; Yepes V. Development of social criteria for the social life cycle assessment of railway infrastructures. *26th International Congress on Project Management and Engineering (AEIPRO)*, 2022.

[17] Yepes-Bellver V.J.; Alcalá J.; Yepes V. Study of solutions for the design of a footbridge based on a hierarchical analytical process. *26th International Congress on Project Management and Engineering (AEIPRO)*, 2022.

[18] Víctor Yepes Piqueras. *Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW.* PhD thesis, Universidad Politécnica de Valencia, 2002.

[19] Ignacio Payá Zaforteza. *Optimización Heurística de Pórticos de Edificación de Hormigón Armado.* PhD thesis, Universidad Politécnica de Valencia, 2007.

[20] Martínez-Muñoz D.; Martí J.V. ; Yepes V. Social impact assessment comparison of composite and concrete bridge alternatives. *Sustainability*, 14(9), 2022.

[21] Computers and Structures Inc. *SAP2000 Linear and Nonlinear, Static and Dynamic Analyisis and Design of Three-Dimensional Structures. Getting Started - User Manual.* Computers and Structures Inc., 2004.

[22] Moragues J. *Diseño óptimo de estructuras aporticadas de hormigón armado.* PhD thesis, Universidad Politécnica de Valencia, 1980.

[23] Víctor Yepes Piqueras. *Coste, producción y mantenimiento de maquinaria para la construcción.* Editorial Universitat Politècnica de Valencia, 2015.

[24] Víctor Yepes Piqueras. *Productivity and Performance.* Editorial Warsaw University of Technology, 2008.

[25] Garfinkel R.S. The traveling salesman problem: A guide tour of combinatorial optimization. *Motivation and Modeling*, 1985.

[26] Bazaraa M.S.; Jarvis J.J.; Sherali H.D. *Programación lineal y flujo en redes.* Limusa, 1998.

[27] Díaz A.; Glover F.; Ghaziri H.M.; González J.L.; Laguna M.; Moscato P.; Tseng F.T. *Optimización Heurística y Redes Neuronales en Dirección de Operaciones de Ingeniería.* Paraninfo, 1996.

[28] Laguna M.; Feo T.; Elrod H. A greedy randomized adaptive search procedures for the 2-partion problem. *Operations Research*, 42(4):677–687, 1997.

[29] Kirkpatrick S.; Gelatt C.D.; Vecchi M.P. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[30] Černy V. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[31] Dueck G.; Scheuer T. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[32] Moscato P.; Fontanari F.J. Stochastic versus deterministic update in simulated annealing. *Physics Letters A*, 146(4):204–208, 1990.

[33] Glover F. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[34] Dueck G. "the great deluge algorithm and the record to record travel. *Journal of Computation Physics*, (104):86, 1993.

[35] Hart J.P.; Shogan A.W. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

[36] Mladenovich N.; Hansen P. Variable neighborhood search. *Journal of Computation Physics*, (24):1097–1100, 1997.

[37] Hu T.C.; Kahng A.B.; Tsao; C-W A. Old bachelor acceptance: a new class of non-monotone threshold accepting methods. *ORSA Journal on Computing*, 7: 417–425, 1995.

[38] Georges-Louis Leclerc de Buffon. *Histoire naturelle, générale et particuliére.* Imprimerie Royale, 1753.

[39] Jean-Baptiste Lamarck. *Philosophie zoologique ou exposition des consedérations relatives à l'histoire naturelle des animaux.* Musée d'Histoire Naturelle, 1809.

[40] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoureed Races in the Struggle for Life.* John Murray, 1859.

[41] Gregor Mendel. *Experiments on Plant Hybridization.* Brünn Natural History Society, 1866.

[42] Baldwin J.M. A new factor in evolution. *The American Naturalist*, 30(354): 441–451, 1896.

[43] Baldwin J.M. Organic selection. *Science*, 5(121):634–636, 1897.

[44] Fogel L.J. *Artificial Intelligence through Simulated Evolution.* John Wiley, 1965.

[45] Ingo Rechenberg. *Evolutionsstrategie.* Stuttgart Frommann-Holzboog, 1994.

[46] Hans-Paul Schwefel. *Evolution and optimum seeking.* John Wiley, 1995.

[47] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to bilolgy, control, and artificial intelligence.* University of Michigan Press, 1975.

[48] Kenneth A. De Jong. *Evolutionary computation: an unified approach.* University of Michigan Press, 2006.

[49] McCulloch W.; Pitts W. A logical calculus of the ideas immanent in nervous system. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[50] Hopfield J.J.; Tank D. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[51] Adeli H.; Yeh C. Perceptron learning in engineering design. *Microcomputers in Civil Engineering*, 4:247–256, 1989.

[52] Adeli H. Neuronal networks in civil engineering: 1989-2000. *Computer-aided civil and Infraestructural Engineering*, 16:126–142, 2001.

[53] Galileo Galilei. *Discorsi e dimostrazioni matematiche : intorno a due nuove scienze attenenti alla mecanica i movimenti locali.* Appresso gli Elsevirii, 1638.

[54] Adeli H. *Advances in Design Optimization.* Chapman Hall, 1994.

[55] Maxwell C. Scientific papers 2. *Dover Publications*, pages 175–177, 1869.

[56] Levy M.M. *La statique graphique et ses applications aux constructions.* Gauther-Villars, 1874.

[57] Mitchell A.G.M. The limits of economy of material in frame structures. *Philosophical Magazine*, 8(47):589–597, 1904.

[58] Klein B. Direct use of external principles in solving certain optimization problems involving inequalities. *Journal of the Operations Research Society of America*, 3:168–175, 1955.

[59] Schmidt L.A. Structural design by systematic synthesis. *Proceedings, 2nd Conference on Electronic Computation*, 36:11–19, 1960.

[60] Karush W. *Minima of Funcions of Several Variables with Inequalities as Side Constraints.* PhD thesis, Dept. Mathematics, University of Chicago, 1939.

[61] Kuhn H.W.; Tucker A.W. Nonlinear programming. *Proceedings, 2nd Berkeley Symposium. University of California Press*, 1951.

[62] Arenas J.; Villegas L. Análisis en teoría de segundo orden de las pilas del viaducto de cruzul. *Hormigón y Acero*, 171:33–55, 1989.

[63] Gasch M.S. *Optimización de estructuras de forjados reticulares.* PhD thesis, Universidad Politécnica de Valencia, 1991.

[64] Cohn M.Z.; Dinovitzer A.S. Application of structural optimization. *ASCE Journal of Structural Engineering*, 120(2):617–650, 1994.

[65] Cohn M.Z.; Dinovitzer A.S. Optimization of reinforced concrete frames. *ASCE Journal of Structural Engineering*, 123(2):193–202, 1994.

[66] Torrano M. *Diseño óptimo de secciones y elementos estructurales de hormigón armado.* PhD thesis, Universidad Politécnica de Cartagena, 2003.

[67] Kuhn H.W.; Tucker A.W. Optimum design of tall buildings in reinforced concrete subjected to wind forces. *Proceedings, 6th Congress of Structural and Multidisciplinary Optimization. Rio de Janeiro*, 2005.

[68] Goldberg D.E.; Samtani. Engineering optimizarion via genetic algorythms. *Proceedings, 9th Conference on Electronic Computation*, pages 471–482, 2005.

[69] Goldberg D.E. *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[70] Hajela P. *Genetic algorithms in automated structural synthesis.* NATO Advanced Study Institute on Optimization and Decision Support Systems, 1989.

[71] Hajela P. Genetic search - an approach to the nonconvex optimization problem. *AIAA Journal*, 26(7):1205–1210, 1990.

[72] Jenkins W.M. Towards structural optimisation via the genetic algorithm. *Computers Structures*, 40(5):1321–1327, 1991.

[73] Jenkins W.M. Structural optimisation with genetic algorithm. *The Structural Engineer*, 69(24):418–422, 1991.

[74] Rajeev S.; Krishnamoorthy C.S. Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering*, 118(5):1223–1250, 1992.

[75] Rajeev S.; Krishnamoorthy C.S. Genetic algorithm-based methodology for design optimization of reinforced concrete frames. *Computer-Aided Civil and Infrastructure Engineering*, 13:63–74, 1998.

[76] Coello C.A.; Christiansen A.D.; Hernández F.S. A simple genetic algorithm for the design of reinforced concrete beams. *Engineering with Computers*, 13: 185–196, 1997.

[77] Chakrabarty B.K. A model for optimal design of reinforced concrete beam. *ASCE Journal of Structural Engineering*, 118(1):3238–3242, 1992.

[78] Srinivas V.; Ramanjaneyulu K. An integrated approach for optimum design of bridge decks using genetic algorithms and artificial neural networks. *Advances in Engineering Software*, 38:475–487, 2007.

[79] Camp. C.; Pezeshk S.; Hansson H. Flexural design of reinforced concrete frames using a genetic algorithm. *ASCE Journal of Structural Engineering*, 130(5): 741–751, 2003.

[80] Leps M.; Sejnoha M. New approach to optimizarion of reinforced concrete beams. *Computers and Structures*, 81(18):1957–1966, 2003.

[81] Cristian Perea de Dios. *Heuristic optimization of reinforced concrete frame bridges.* PhD thesis, Universidad Politécnica de Valencia, 2007.

[82] Perea C.; Alcalá J.; Yepes V.; González-Vidosa F.; Hospitaler A. Design of reinforced concrete bridge frames by heuristic optimization. *Advances in Engineering Software*, 39(8):676–688, 2007.

[83] Perea C.; Yepes V.; Alcalá J.; Hospitaler A.; González-Vidosa F. A parametric study of optimum road frame bridges by threshold acceptance. *Indian Journal of Engineering Material Sciences*, 17(6):427–437, 2010.

[84] Cristian Perea de Dios. *Heuristic optimization of reinforced concrete road box frames.* Informe interno CST/GPRC-03 Universidad Politécnica de Valencia, Dep. Ingeniería de la construcción, 2004.

[85] Yepes V.; González-Vidosa F.; Alcalá J.; Villalba P. co_2-optimization design of reinforced concrete retaining walls based on a vns-threshold acceptance strategy. *Journal of Computing in Civil Engineering ASCE*, 26(3):378–386, 2012.

[86] Payá I.; Yepes V.; González-Vidosa F.; Hospitaler A. Multiobjective optimization of reinforced concrete building frames by simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 23(8):596–610, 2008.

[87] Carbonell A.; Yepes V.; González-Vidosa F. Búsqueda exhaustiva por entornos aplicada al diseño económico de bóvedas de hormigón armado. *Revista*

Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería, 27 (3):227–235, 2011.

[88] Carbonell A.; González-Vidosa F.; Yepes V. Design of reinforced concrete road vaults by heuristic optimization. *Advances in Engineering Software*, 42 (4):151–159, 2011.

[89] Carbonell A.; Yepes V.; González-Vidosa F. Automatic design of concrete vaults using iterated local search and extreme value estimation. *Latin American Journal of Solids and Structures*, 9(6):675–689, 2012.

[90] Torres-Machí C.; Yepes V.; González-Vidosa F. Optimization of high-performance concrete structures by variable neigborhood search. *International Journal of Civil Engineering*, 11(2):90–99, 2013.

[91] Martí J.V.; González-Vidosa F.; Yepes V.; Alcalá J. Design of prestressed concrete precast road bridges with hybrid simulated annealing. *Engineering Structures*, 48:342–352, 2013.

[92] Martí J.V.; Yepes V.; González-Vidosa F.; Luz A. Diseño automático de tableros óptimos de puentes de carretera de viga artesa prefabricadas mediante algoritmos meméticos híbridos. *Revista Internacional de Métodos Numéricos para el Cálculo y Diseño en Ingeniería*, 30(3):145–154, 2014.

[93] Martí J.V.; Yepes V.; González-Vidosa F. A memetic algorithm approach to designing of precast-prestressed concrete road bridges with steel fiber reinforcement. *Journal of Structural Engineering ASCE*, 141(2), 2015.

[94] García-Segura T.; Yepes V.; Martí J.V.; Alcalá J. Optimization of concrete i-beams using a new hybrid glowworm swarm algorithm. *Latin American Journal of Solids and Structures*, 11(7):1190–1205, 2014.

[95] García-Segura T.; Yepes V.; Alcalá J. Lice-cycle greenhouse gas emissions of blended cement concrete including carbonation and durability. *The International Journal of Life Cycle Assessment*, 19(1):3–12, 2014.

[96] Pérez-López E. García-Segura T.; Yepes V.; Alcalá J. Hybrid harmony search for sustainable design of post-tensioned concrete box-grider pedestrian bridges. *Engineering Structures*, 92:112–122, 2015.

[97] Martínez-Muñoz D.; Martí J.V.; Yepes V. Steel-concrete composite bridges: Design, lyfe cycle assessment, maintenance, and decision-making. *Advances in Civil Engineering*, 2020, 2020.

[98] Comité Europeo de Normalización. *Eurocódigo EC-2: Proyectos de Estructuras de Hormigón*. UNE-ENV 1992-2. AENOR, 1992.

[99] Dirección General de Carreteras en conjunto con el Laboratorio de Geotecnia del CE-DEX. *Guía para el proyecto de cimentaciones en obras de carretera.* Gobierno de España, 2002.

[100] Catalonia Institute of Construction Technology. BEDEC ITEC, material database, 2022. URL https://metabase.itec.cat/vide/es/bedec.

[101] Medina J.R. Estimation of incident and reflected waves using simulated annealing. *ASCE Journal of Waterway, Port, Coastal and Ocean Engineering.*, 127(4):213–221, 2001.

[102] Payá I.; Yepes V.; Clemente J.J.; González-Vidosa F. Optimización heurística de pórticos de edificación de hormigón armado. *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, 202, 2006.

[103] Python Software Foundation. Open source python programming language, 2022. URL https://www.python.org/.

[104] Perea C.; Alcalá J.; Martínez F.J.; Yepes V.; González-Vidosa F. Optimum design of the box frame tunnel of the underground line palma-uib. *Proceedings, 10th International Conference on Project Engineering*, pages 12–15, 2006.

[105] Ministerio de Fomento. *Comisión Permanente del Hormigón Ïnstrucción de Hormigón Estructural".* Gobierno de España, 2008.

[106] Jiménez Salas J.A.; de Justo Alpañes J.L.; Serrano González A.A. *Geotecnia y Cimientos III.* Rueda, 1980.

Apéndice A

Software desarrollado

A.1. Main SA

```
"""
Author: Andrés Ruiz Vélez - aruivel@posgrado.upv.es
Date: February 2022

Instituto Universitario de Ciencia y Tecnología del Hormigón
Universitat Politècnica de València

Programa de calculo y optimización de marco prefabricado
"""

import numpy as np
import matplotlib.pyplot as matplot
import time

import data as dat
import plotting as plot
import modelgenerator as model
import sectionproperties as secprop
import stiffnesscalculator as stiff
import loadcalculator as load
import forcevectorcalculator as force
import stresscalculator as stress
import envelopescalculator as envelopes
```

```
import NMdiagramgenerator as NMdiagram
import sectionverificator as verif
import modelverificator as modelverif
import objectivefunction as objfunc

import maincalculationsmodule as calcmodule
import simulatedannealing_ARV as sa_ARV
import toexcel as toexcel


if __name__ == '__main__':

    # Inicialización de modelo del marco
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha, dat.condense)

    # Variables geométricas de las secciones
    hh = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hls = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hli = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    # Variables del armado del marco
    fiA1 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA1 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA2 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA2 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA3 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA3 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA4 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA4 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA5 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA5 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA6 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA6 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA7 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA7 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA12 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA12 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA13 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA13 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
```

```
fiA14 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
nA14 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
fiA15 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
nA15 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
fiA18 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
sA18 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
fiA20 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
sA20 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
# Variables de los materiales del marco
concrete = np.array((25, 30, 35, 40))  # En MPa
steel = np.array((400, 500))

variables_marco = [hh, hls, hli,

                   fiA1, nA1, fiA2, nA2, fiA3, nA3, fiA4, nA4, fiA5, nA5,
                   fiA6, nA6, fiA7, nA7, fiA12, nA12, fiA13, nA13, fiA14,
                   nA14, fiA15, nA15, fiA18, sA18, fiA20, sA20,

                   concrete, steel]

data_labels = [
    'markov_chain',

    'iteration',

    'cost', 'CO2', 'energy',

    'temperature',

    'it_time',

    'elapsed_time',

    'hh', 'hls', 'hli', 'fiA1', 'nA1', 'fiA2', 'nA2', 'fiA3', 'nA3', 'fiA4',
    ↪  'nA4', 'fiA5',
    'nA5', 'fiA6', 'nA6', 'fiA7', 'nA7', 'fiA12', 'nA12', 'fiA13', 'nA13',
    ↪  'fiA14', 'nA14',
    'fiA15', 'nA15', 'fiA18', 'sA18', 'fiA20', 'sA20', 'concrete', 'steel',
```

```
    'Vconc', 'Vsteel'

]

combinaciones = 1
long_vector = 0
for i in range(0, len(variables_marco)):
    long_vector = 0
    for j in range(0, len(variables_marco[i])):
        long_vector += 1
    if long_vector != 0:
        combinaciones = combinaciones * long_vector
print('Nº de combinaciones: %s' % combinaciones)

replicas = 9
incio_tiempo = time.time()
for i in range(replicas):
    cadena_markov = 10000
    desv_estandar = 0.3
    numero_variables = 5
    coeficiente_enfriamiento = 0.90
    criterio_de_parada = 5  # Cadenas sin mejora
    criterio_terminacion = 0.05
    t0 = 0.05
    rango_inferior = 0.20
    rango_superior = 0.40

    mh_variables = [cadena_markov, desv_estandar, numero_variables,
    ↪  coeficiente_enfriamiento, criterio_de_parada,
                    criterio_terminacion, t0, rango_inferior, rango_superior]

    data, data_best = sa_ARV.sa(variables_marco, mh_variables, data_labels)

    book_name = 'SA%s_%s_%s_%s_%s_%s' % (i, cadena_markov, desv_estandar,
    ↪  numero_variables,
                                         coeficiente_enfriamiento,
                                         ↪  criterio_de_parada)
    book_name_pro = 'SA_process%s_%s_%s_%s_%s_%s' % (i, cadena_markov,
    ↪  desv_estandar, numero_variables,
```

```
                                                coeficiente_enfriamiento,
                                                ↪ criterio_de_parada)
        toexcel.write_3d_array(book_name, 1, data_labels, len(data_best),
        ↪ data_best)
        toexcel.write_3d_array(book_name_pro, 1, data_labels, len(data), data)
    fin_tiempo = time.time()
    tiempo_completo = fin_tiempo - incio_tiempo
    print('Tiempo empleado en el cálculo: ', tiempo_completo)
```

A.2. Main TA

```
"""
Author: Andrés Ruiz Vélez - aruivel@posgrado.upv.es
Date: February 2022

Instituto Universitario de Ciencia y Tecnología del Hormigón
Universitat Politècnica de València

Programa de calculo y optimización de marco prefabricado
"""

import numpy as np
import matplotlib.pyplot as matplot
import time

import data as dat
import plotting as plot
import modelgenerator as model
import sectionproperties as secprop
import stiffnesscalculator as stiff
import loadcalculator as load
import forcevectorcalculator as force
import stresscalculator as stress
import envelopescalculator as envelopes
import NMdiagramgenerator as NMdiagram
import sectionverificator as verif
import modelverificator as modelverif
import objectivefunction as objfunc
```

```
import maincalculationsmodule as calcmodule
import thresholdaccepting_ARV as ta_ARV
import toexcel as toexcel

if __name__ == '__main__':

    # Inicialización de modelo del marco
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha, dat.condense)

    # Variables geométricas de las secciones
    hh = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hls = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hli = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    # Variables del armado del marco
    fiA1 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA1 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA2 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA2 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA3 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA3 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA4 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA4 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA5 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA5 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA6 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA6 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA7 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA7 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA12 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA12 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA13 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA13 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA14 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA14 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA15 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA15 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA18 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
```

```
sA18 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
fiA20 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
sA20 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
# Variables de los materiales del marco
concrete = np.array((25, 30, 35, 40))  # En MPa
steel = np.array((400, 500))

variables_marco = [hh, hls, hli,

                   fiA1, nA1, fiA2, nA2, fiA3, nA3, fiA4, nA4, fiA5, nA5,
                   fiA6, nA6, fiA7, nA7, fiA12, nA12, fiA13, nA13, fiA14,
                   nA14, fiA15, nA15, fiA18, sA18, fiA20, sA20,

                   concrete, steel]

data_labels = [
    'markov_chain',

    'iteration',

    'cost', 'CO2', 'energy',

    'temperature',

    'it_time',

    'elapsed_time',

    'hh', 'hls', 'hli', 'fiA1', 'nA1', 'fiA2', 'nA2', 'fiA3', 'nA3', 'fiA4',
    ↪  'nA4', 'fiA5',
    'nA5', 'fiA6', 'nA6', 'fiA7', 'nA7', 'fiA12', 'nA12', 'fiA13', 'nA13',
    ↪  'fiA14', 'nA14',
    'fiA15', 'nA15', 'fiA18', 'sA18', 'fiA20', 'sA20', 'concrete', 'steel',

    'Vconc', 'Vsteel'
]

combinaciones = 1
long_vector = 0
```

```
for i in range(0, len(variables_marco)):
    long_vector = 0
    for j in range(0, len(variables_marco[i])):
        long_vector += 1
    if long_vector != 0:
        combinaciones = combinaciones * long_vector
print('Nº de combinaciones: %s' % combinaciones)

replicas = 9

incio_tiempo = time.time()

for i in range(replicas):
    cadena_umbral = 10000
    desv_estandar = 0.3
    numero_variables = 5
    coeficiente_enfriamiento = 0.90
    criterio_de_parada = 5  # Cadenas sin mejora
    criterio_terminacion = 0.05
    u0 = 0.10
    rango_inferior = 0.20
    rango_superior = 0.40

    mh_variables = [cadena_umbral, desv_estandar, numero_variables,
    ↪  coeficiente_enfriamiento, criterio_de_parada,
                    criterio_terminacion, u0, rango_inferior, rango_superior]

    data, data_best = ta_ARV.ta(variables_marco, mh_variables, data_labels)

    book_name = 'TA%s_%s_%s_%s_%s_%s' % (i, cadena_umbral, desv_estandar,
    ↪  numero_variables,
                                         coeficiente_enfriamiento,
                                         ↪  criterio_de_parada)
    book_name_pro = 'TA_process%s_%s_%s_%s_%s_%s' % (i, cadena_umbral,
    ↪  desv_estandar, numero_variables,
                                         coeficiente_enfriamiento,
                                         ↪  criterio_de_parada)
    toexcel.write_3d_array(book_name, 1, data_labels, len(data_best),
    ↪  data_best)
```

```
        toexcel.write_3d_array(book_name_pro, 1, data_labels, len(data), data)

    fin_tiempo = time.time()
    tiempo_completo = fin_tiempo - incio_tiempo

    print('Tiempo empleado en el cálculo: ', tiempo_completo)
```

A.3. Main OBA

```
"""
Author: Andrés Ruiz Vélez - aruivel@posgrado.upv.es
Date: February 2022

Instituto Universitario de Ciencia y Tecnología del Hormigón
Universitat Politècnica de València

Programa de calculo y optimización de marco prefabricado
"""

import numpy as np
import matplotlib.pyplot as matplot
import time

import data as dat
import plotting as plot
import modelgenerator as model
import sectionproperties as secprop
import stiffnesscalculator as stiff
import loadcalculator as load
import forcevectorcalculator as force
import stresscalculator as stress
import envelopescalculator as envelopes
import NMdiagramgenerator as NMdiagram
import sectionverificator as verif
import modelverificator as modelverif
import objectivefunction as objfunc

import maincalculationsmodule as calcmodule
```

```
import oldbacheloracceptance_ARV as oba_ARV
import toexcel as toexcel

if __name__ == '__main__':

    # Inicialización de modelo del marco
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha, dat.condense)

    # Variables geométricas de las secciones
    hh = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hls = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    hli = np.linspace(0.4, 1.2, int(np.ceil((1.2 - 0.4) / 0.02) + 1))
    # Variables del armado del marco
    fiA1 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA1 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA2 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA2 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA3 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA3 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA4 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA4 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA5 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA5 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA6 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA6 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA7 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA7 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA12 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA12 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA13 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA13 = np.array((4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA14 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA14 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA15 = np.array((10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    nA15 = np.array((3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
    fiA18 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
    sA18 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
    fiA20 = np.array((8e-03, 10e-03, 12e-03, 16e-03, 20e-03, 25e-03, 32e-03))
```

```
sA20 = np.array((0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40))
# Variables de los materiales del marco
concrete = np.array((25, 30, 35, 40))  # En MPa
steel = np.array((400, 500))

variables_marco = [hh, hls, hli,

                   fiA1, nA1, fiA2, nA2, fiA3, nA3, fiA4, nA4, fiA5, nA5,
                   fiA6, nA6, fiA7, nA7, fiA12, nA12, fiA13, nA13, fiA14,
                   nA14, fiA15, nA15, fiA18, sA18, fiA20, sA20,

                   concrete, steel]

data_labels = [
    'iteration',

    'cost', 'C02', 'energy',

    'temperature',

    'it_time',

    'elapsed_time',

    'hh', 'hls', 'hli', 'fiA1', 'nA1', 'fiA2', 'nA2', 'fiA3', 'nA3', 'fiA4',
    ↪  'nA4', 'fiA5',
    'nA5', 'fiA6', 'nA6', 'fiA7', 'nA7', 'fiA12', 'nA12', 'fiA13', 'nA13',
    ↪  'fiA14', 'nA14',
    'fiA15', 'nA15', 'fiA18', 'sA18', 'fiA20', 'sA20', 'concrete', 'steel',

    'Vconc', 'Vsteel'

]

combinaciones = 1
long_vector = 0
for i in range(0, len(variables_marco)):
    long_vector = 0
    for j in range(0, len(variables_marco[i])):
```

```
            long_vector += 1
        if long_vector != 0:
            combinaciones = combinaciones * long_vector
    print('Nº de combinaciones: %s' % combinaciones)

    replicas = 1

    incio_tiempo = time.time()

    for i in range(replicas):
        desv_estandar = 0.3
        numero_variables = 5
        coeficiente_variacion = 100
        criterio_terminacion = 50000

        mh_variables = [desv_estandar, numero_variables, coeficiente_variacion,
        ↪  criterio_terminacion]

        data, data_best = oba_ARV.oba(variables_marco, mh_variables, data_labels)

        book_name = 'OBA%s_%s_%s_%s' % (i, criterio_terminacion, desv_estandar,
        ↪  numero_variables)
        book_name_pro = 'OBA_process%s_%s_%s_%s' % (i, criterio_terminacion,
        ↪  desv_estandar, numero_variables)
        toexcel.write_3d_array(book_name, 1, data_labels, len(data_best),
        ↪  data_best)
        toexcel.write_3d_array(book_name_pro, 1, data_labels, len(data), data)

    fin_tiempo = time.time()
    tiempo_completo = fin_tiempo - incio_tiempo

    print('Tiempo empleado en el cálculo: ', tiempo_completo)
```

A.4. Módulo base de datos

```
"""
Author: Andrés Ruiz Vélez - aruivel@posgrado.upv.es
Date: February 2022
```

```
Instituto Universitario de Ciencia y Tecnología del Hormigón
Universitat Politècnica de València

Programa de calculo y optimización de marco prefabricado
"""

import numpy as np

"""GEOMETRÍA DEL MARCO"""

# Geométricos (m)

B = 10                      # Anchura del marco (m)
H = 5                       # Altura del marco (m)
rotula = 2/3                # Posición relativa de la unión en hastiales (m/m)

b = 1                       # Ancho de análisis por norma (1 m)

hls = 0.9                  # Canto de la losa superior (m)
hli = 0.9                  # Canto de la losa inferior (m)
hh = 0.60                  # Canto de los hastiales (m)

rec = 0.035                # Recubrimiento de las armaduras (m)


L = np.array((3*H/5,        # Longitud tramo 1 (Hastial izquierdo)
              2*H/5,
              B,            # Longitud tramo 2 (Losa superior)
              2*H/5,        # Longitud tramo 3 (Hastial derecho)
              3*H/5,
              B))           # Longitud tramo 4 (Losa inferior)

nb = np.array((3,           # Número de barras en el tramo 1
               2,
               10,          # Número de barras en el tramo 2
               2,           # Número de barras en el tramo 3
               3,
               10))         # Número de barras en el tramo 4
```

```
# Inclinación de la barra respecto al eje X global

alpha = np.array((90,           # Alpha tramo 1
                  90,
                  0,            # Alpha tramo 2
                  270,          # Alpha tramo 3
                  270,
                  180))         # Alpha tramo 4

"""Indicar el nodo a condensar:
        Giro libre en extremo inicial (1)
        Giro libre en extremo final (2)"""

condense = np.array((2,     # Condensación giro en el tramo 1
                     1,
                     0,     # Condensación giro en el tramo 2
                     2,     # Condensación giro en el tramo 3
                     1,
                     0))    # Condensación giro en el tramo 4


"""ACCIONES CONSIDERADAS PARA EL CÁLCULO"""

cct = 10
# 1. ACCIONES PERMANENTES
# 1.1. Peso propio
pconc = 25e03                    # Peso específico del hormigón armado (N/m3)
# 1.2. Carga muerta
ppav = 23e03                     # Peso específico del pavimento (N/m3)
epav = 0.1                       # Espesor del pavimento (m)
ptr = 20e03                      # Peso específico del terreno de relleno (N/m3)
etr = 1.5                        # Profudidad del marco respecto a la superficie
↪  (m)

# 2. ACCIONES PERMANENTES DE VALOR NO CONSTANTE
# 2.1. Empuje lateral del terreno
phi = np.deg2rad(30)                      # Ángulo de rozamiento del terrreno con
↪  la estructura (º)
```

```
# CARRO DE LA IAP

qsuIAP = 10000
qcarro = 150000
Lccarro = 1.2
Lacarro = B / 2

"""MATERIALES DEL MARCO"""

# HORMIGÓN EUROCÓDIGO 2 (UNE-EN 1992-1-1) Tabla 3.1 Características mecánicas en
↪ función del tiempo

fck = 25                                        # Resistencia característica del
↪ hormigón (Mpa)
csconcrete = 1.5                                # Coeficiente de seguridad del
↪ hormigón

age = 7                                         # Edad del hormigón en días
↪ (días)
s = 0.25                                        # Coeficiente velocidad de
↪ endurecimiento normal
betacc = np.exp(s * (1 - np.sqrt(28 / age)))
HR = 75                                         # Humedad relativa (%)
vc = 0.2                                        # Coeficiente de poisson del
↪ hormigón
vs = 0.3                                        # Coeficiente de poisson del
↪ acero
cterm = 1 * 10 ** (-5)                          # Coeficiente térmico (ºC-1)

fcd = fck / csconcrete                          # Resistencia de cálculo del
↪ hormigón (MPa)
fcm = fck + 8                                   # Resistencia media del hormigón
↪ (MPa)
if fck <= 50:
    fctm = 0.3 * fck ** (2 / 3)
else:
    fctm = 2.12 * np.log(1 + (fcm / 10))
```

```
fcmt = betacc * fcm                                    # Resistencia media en función
↪  del tiempo (MPa)
fctmt = betacc * fctm                                  # Resistencia media a tracción en
↪  función del tiempo (MPa)
fckt = fcmt - 8                                        # Resistencia característica en
↪  función del tiempo (MPa)

Ecm = 22 * (fcm / 10) ** 0.3 * 10 ** 9                 # Módulo elástico longitudinal
↪  del hormigón (MPa)                                 # Módulo elástico transversal del
↪  hormigón (MPa)
E = Ecm
Ecmt = (fcmt / fcm) ** 0.3 * Ecm                       # Modulo elástico longitudinal
↪  del hormigón en función del tiempo (MPa)


# ARMADURAS PASIVAS EUROCÓDIGO 2 (UNE-EN 1992-1-1)

fyk = 500                                              # Resistencia característica del
↪  acero (MPa)
gammas = 1.15                                          # Coeficiente de seguridad del
↪  acero
fyd = fyk / gammas                                     # Resistencia de cálculo (MPa)
Es = 200e09
sdensity = 7858.2                                      # Densidad del acero de las
↪  armaduras pasivas (Kg/m3)

# Longitudes de REFUERZO en esquinas
A22 = 2    # Longitud de refuerzo A14 en Z
A32 = 2    # Longitud de refuerzo A14 en X
A23 = 1.5  # Longitud de refuerzo A15 en Z
A33 = 3    # Longitud de refuerzo A15 en X
A24 = A23  # Longitud de refuerzo A16 en Z
A34 = A33  # Longitud de refuerzo A16 en X
A25 = A22  # Longitud de refuerzo A17 en Z
A35 = A32  # Longitud de refuerzo A17 en X

# Distancia armaduras de CORTANTE
A26 = 3    # Distancia armadura cortante A18 losa superior
A27 = A26  # Distancia armadura cortante A19 losa superior
```

```
A28 = 2    # Distancia armadura cortante A20 losa inferior
A29 = A28  # Distancia armadura cortante A21 losa inferior

# Longitud armadura de REFUERZO losa superior
A30 = 5

################################## VARIABLES del ARMADO del MARCO
↪  ######################################################
# Armado BASE del HASTIAL IZQUIERDO
# Parte INFERIOR
fiA1 = 32e-03       # Diámetro armadura inferior
nA1 = 8             # Número de barras armadura inferior
fiA2 = 32e-03       # Diámetro armadura superior
nA2 = 8             # Número de barras armadura superior
# Parte SUPERIOR
fiA3 = 32e-03       # Diámetro armadura inferior
nA3 = 8
fiA4 = 32e-03
nA4 = 8

# Armado BASE y REFUERZO de la LOSA SUPERIOR
fiA5 = 32e-03       # Diámetro armadura inferior
nA5 = 8             # Número de barras armadura inferior
fiA6 = 25e-03       # Diámetro armadura de refuerzo
nA6 = 2             # Número de barras armadura de refuerzo
fiA7 = 32e-03       # Diámetro armadura superior
nA7 = 8             # Número de barras armadura superior

# Armado BASE del HASTIAL DERECHO
# Parte SUPERIOR
fiA8 = fiA3         # Diámetro armadura inferior
nA8 = nA3           # Número de barras armadura inferior
fiA9 = fiA4         # Diámetro armadura superior
nA9 = nA4           # Número de barras armadura superior
# Parte INFERIOR
fiA10 = fiA1        # Diámetro armadura inferior
nA10 = nA1          # Número de barras armadura inferior
fiA11 = fiA2        # Diámetro armadura superior
nA11 = nA2          # Número de barras armadura superior
```

```
# Armado BASE de la LOSA INFERIOR
fiA12 = 32e-03      # Diámetro armadura inferior
nA12 = 8            # Número de barras armadura inferior
fiA13 = 32e-03      # Diámetro armadura superior
nA13 = 8            # Número de barras armadura superior

# Armado de REFUERZO en las ESQUINAS
# Esquina INFERIOR IZQUIERDA
fiA14 = 20e-03      # Diámetro armadura de refuerzo esquina
nA14 = 8            # Número de barras armadura de refuerzo esquina
# Esquina SUPERIOR IZQUIERDA
fiA15 = 20e-03      # Diámetro armadura de refuerzo esquina
nA15 = 8            # Número de barras armadura de refuerzo esquina
# Esquina SUPERIOR DERECHA
fiA16 = fiA15       # Diámetro armadura de refuerzo esquina
nA16 = nA15         # Número de barras armadura de refuerzo esquina
# Esquina INFERIOR DERECHA
fiA17 = fiA14       # Diámetro armadura de refuerzo esquina
nA17 = nA14         # Número de barras armadura de refuerzo esquina

# Armado de CORTANTE en las LOSAS
fiA18 = 20e-03      # Diámetro cercos de armadura a cortante
sA18 = 0.1          # Separación entre cercos de armadura a cortante
fiA19 = fiA18       # Diámetro cercos de armadura a cortante
sA19 = sA18         # Separación entre cercos de armadura a cortante
fiA20 = 20e-03      # Diámetro cercos de armadura a cortante
sA20 = 0.1          # Separación entre cercos de armadura a cortante
fiA21 = fiA20       # Diámetro cercos de armadura a cortante
sA21 = sA20         # Separación entre cercos de armadura a cortante
```

A.5. Main módulo de verificación

```
import numpy as np
import matplotlib.pyplot as matplot
import time

import data as dat
```

```
import plotting as plot
import modelgenerator as model
import sectionproperties as secprop
import stiffnesscalculator as stiff
import loadcalculator as load
import forcevectorcalculator as force
import stresscalculator as stress
import envelopescalculator as envelopes
import NMdiagramgenerator as NMdiagram
import sectionverificator as verif
import modelverificator as modelverif
import objectivefunction as objfunc


def generation(members, nodes, thetas):

    reinforcment = secprop.SectionReinforcementGenerator(nodes, thetas, dat.L)
    canto = secprop.SectionDimensionGenerator(nodes, thetas)

    E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2, firefesq, nsrefesq, Asrefesq,
    ↪  firef, nsref, Asref, Asw, sw = \
        secprop.nodeMechanicalPorperties(members, nodes, reinforcement, canto)

    return reinforcement, canto, E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2,
    ↪  firefesq, nsrefesq, Asrefesq, firef, nsref, Asref, Asw, sw


def objectivefunction(nodes, members, lenghts, thetas):

    reinforcement, canto, E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2, firefesq,
    ↪  nsrefesq, Asrefesq, firef, \
    nsref, Asref, Asw, sw = generation(members, nodes, thetas)

    costetotal, mediciones = objfunc.meassurements(nodes, members, dat.nb, A,
    ↪  canto, dat.L, lenghts, As1, As2, Asrefesq, Asref, Asw, sw, dat.fck,
    ↪  dat.fyk)

    return costetotal, mediciones
```

```
def verification(nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF,
↪  restrains):

    start_time = time.time()

    reinforcement = secprop.SectionReinforcementGenerator(nodes, thetas, dat.L)
    canto = secprop.SectionDimensionGenerator(nodes, thetas)

    E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2, firefesq, nsrefesq, Asrefesq,
    ↪  firef, nsref, Asref, Asw, sw = \
        secprop.nodeMechanicalPorperties(members, nodes, reinforcement, canto)

    mg_time = time.time()
    modelgeneration_time = mg_time - start_time

    geomverif = secprop.SectionGeometryVerificator(nodes, reinforcement, canto)
    coefminAs, coefmaxAs = secprop.minimumReinforcementVerification(members,
    ↪  nodes, As1, Asref, canto, I, dat.fctm,
                                                                    dat.fyd)
    Astr = secprop.additionalReinforcementValues(dat.nb, dat.L, As1, Asref,
    ↪  canto)

    s1_time = time.time()

    Kp = stiff.buildKp(members, pins, E, A, I, L=lenghts, theta=thetas)
    Ks = stiff.buildKs(Kp, pinDoF, restrainedDoF)

    q, qx, q1t, q2t, ql, La, Lb, Lc = load.loadcalculatorIteration(members,
    ↪  nodes, dat.nb, E, h, A, I, lenghts, thetas)
    feq = force.forceVectorcalculatorIteration(members, pins, q, qx, q1t, q2t,
    ↪  ql, La, Lb, Lc, thetas, lenghts)

    UG = stress.solveDisplacementsIteration(feq, Ks, pinDoF, restrainedDoF,
    ↪  members)
    FG = stress.solveReactions(Kp, UG)
    N, V, M = stress.solveMemberForcesIteration(UG, members, pins, E, A, I,
    ↪  lenghts, thetas)
```

```
FG_f = stress.removeENAfromReactionsIteration(members, FG, feq, thetas)
Nreal, Vreal, Mreal = stress.removeENAfromMemberActionsIteration(members, N,
↪  V, M, q, qx, q1t, q2t, ql, La, Lb, Lc, lenghts, pins, thetas)

stiffnes_time = time.time() - s1_time

s2_time = time.time()

ELUf, N_ELU, V_ELU, M_ELU, UG_ELU, env = envelopes.stressEnvelopes(members,
↪  Nreal, Vreal, Mreal, UG)
ELScar, N_ELScar, V_ELScar, M_ELScar, UG_ELScar, envELScar, \
ELSc, N_ELSc, V_ELSc, M_ELSc, UG_ELSc, envELSc =
↪  envelopes.stressEnvelopesELS(members, Nreal, Vreal, Mreal, UG)

envelope_time = time.time() - s2_time
s3_time = time.time()

coefVELU, incMedELU = verif.VverificatorELUIteration(nodes, members, ELUf, h,
↪  dat.rec, As1, As2, Asrefesq, Asref, dat.fck, dat.fcd, dat.fyk, N_ELU,
↪  M_ELU, V_ELU, Asw, sw)

V_ELU_time = time.time() - s3_time

######## Comprobación ELU SOLICITACIONES NORMALES (NM) ####################

s4_time = time.time()

M_ELUdec = verif.DecalajeApplication(members, incMedELU, M_ELU, ELUf)

Nufp, Mufp, Nufn, Mufn, Nuinf, Nusup, sep =
↪  NMdiagram.NMdiagramIteration(nodes, dat.Es, b, h, As2, As1, Asrefesq,
↪  Asref, dat.rec, dat.fcd, dat.fyd, dat.fck,  dat.fyk, 250)

coefNELU, coefMELU = verif.NMverificatorELUIteration(nodes, Nufp, Mufp, Nufn,
↪  Mufn, Nuinf, Nusup, N_ELU, M_ELUdec, ELUf)

normal_check_time = time.time() - s4_time
s5_time = time.time()
```

```
######## Comprobación ELS FISURACIÓN ########################################

coefELScar, coefELScua = verif.FisuracionverificatorELSIteration(nodes,
↪  ELScar, ELSc, As1, As2, Asrefesq, Asref, fi1, ns1, fi2, ns2, canto, I,
↪  M_ELScar, M_ELSc, 0.2)

ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
    modelverif.CoefficientsCounter(nodes, geomverif, coefminAs, coefmaxAs,
    ↪  ELUf, ELScar, ELSc,
                                   coefVELU, coefNELU, coefMELU, coefELScar,
                                   ↪  coefELScua)

ELS_time = time.time() - s5_time
check_time = time.time() - s3_time

total_time = time.time() - start_time

p_generation = modelgeneration_time / total_time * 100
p_stiffness = stiffnes_time / total_time * 100
p_envelope = envelope_time / total_time * 100
p_shear_check = V_ELU_time / total_time * 100
p_normal_check = normal_check_time / total_time * 100
p_fisure_check = ELS_time / total_time * 100
p_check = check_time / total_time * 100


print('\n*************** Porcentajes de tiempo ***************')
print('Tiempo total:                          %2.6f segundos' % total_time)
print('Porcentaje generación del modelo:      %2.6f' % p_generation)
print('Porcentaje MDR:                        %2.6f' % p_stiffness)
print('Porcentaje envolventes:                %2.6f' % p_envelope)
print('Porcentaje comprobaciones:             %2.6f' % p_check)
print('Porcentaje ELU cortante:               %2.6f' % p_shear_check)
print('Porcentaje ELU normales:               %2.6f' % p_normal_check)
print('Porcentaje ELS fisuración:             %2.6f' % p_fisure_check)

print('\n***************** Tiempos de proceso *****************')
print('Tiempo total:                          %2.6f segundos' % total_time)
```

```
print('Tiempo generación del modelo:            %2.6f' %
↪  modelgeneration_time)
print('Tiempo MDR:                              %2.6f' % stiffnes_time)
print('Tiempo envolventes:                      %2.6f' % envelope_time)
print('Tiempo comprobaciones:                   %2.6f' % check_time)
print('Tiempo ELU cortante:                     %2.6f' % V_ELU_time)
print('Tiempo ELU normales:                     %2.6f' % normal_check_time)
print('Tiempo ELS fisuración:                   %2.6f' % ELS_time)

costetotal, mediciones = objfunc.meassurements(nodes, members, dat.nb, A,
↪  canto, dat.L, lenghts, As1, As2, Asrefesq, Asref, Asw, sw, dat.fck,
↪  dat.fyk)


######### GRÁFICAS Y DIAGRAMAS ##############################################

ccNM = int(env[2, 0])

for secti in range(0, len(members)):
    plot.plotNMdiagram(Nufp[secti, :], Mufp[secti, :], Nufn[secti, :],
    ↪  Mufn[secti, :], Nuinf[secti], Nusup[secti],
                       N_ELU[ccNM, secti] / 1000, M_ELUdec[ccNM, secti, 0] /
                       ↪  1000, sep[secti, :])

sectionIndex = 5
plot.plotSection(nodes, members, reinforcement, h, dat.b, sectionIndex)

"""Grafica las envolventes del marco"""
plot.plotstructure(nodes, members, pins, restrains)
plot.plotBMDenvelopes(nodes, members, M_ELU[env[2, 0]], M_ELU[env[2, 1]],
↪  thetas, lenghts)
plot.plotShearsenvelopes(nodes, members, V_ELU[env[1, 0]], V_ELU[env[1, 1]],
↪  thetas, lenghts)
plot.plotForcesenvelopes(nodes, members, N_ELU[env[0, 0]], N_ELU[env[0, 1]],
↪  thetas, lenghts)
plot.plotDeflection(nodes, members, UG_ELU[:, env[2, 0]])

cc = 0
plot.plotstructure(nodes, members, pins, restrains)
```

```
    plot.plotDeflection(nodes, members, UG[:, cc])
    plot.plotForces(nodes, members, Nreal[cc], thetas)
    plot.plotShears(nodes, members, Vreal[cc], thetas, lenghts)
    plot.plotBMD(nodes, members, Mreal[cc], thetas, lenghts)

    return ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS
```

A.5.1. Generación del modelo

```
import numpy as np
import data as dat
import random as random

def SectionDimensionGenerator(nodes, thetas):
    canto = np.zeros((len(nodes)))
    h = - 1
    hrandomhastial = dat.hh
    hrandomlosasup = dat.hls
    hrandomlosainf = dat.hli
    for i, node in enumerate(nodes):
        if thetas[i] == np.deg2rad(90) or thetas[i] == np.deg2rad(270):
            h = hrandomhastial
        if thetas[i] == np.deg2rad(0):
            h = hrandomlosasup
        if thetas[j] == np.deg2rad(180):
            h = hrandomlosainf
        canto[k] = h

    return canto


def SectionReinforcementGenerator(nodes, thetas, L):

    # Inicialización del armado base
    fiA1 = dat.fiA1
    fiA2 = dat.fiA2
    fiA3 = dat.fiA3
    fiA4 = dat.fiA4
    fiA5 = dat.fiA5
```

```
fiA7 = dat.fiA7
fiA8 = fiA3
fiA9 = fiA4
fiA10 = fiA1
fiA11 = fiA2
fiA12 = dat.fiA12
fiA13 = dat.fiA13
nA1 = dat.nA1
nA2 = dat.nA2
nA3 = dat.nA3
nA4 = dat.nA4
nA5 = dat.nA5
nA7 = dat.nA7
nA8 = nA3
nA9 = nA4
nA10 = nA1
nA11 = nA2
nA12 = dat.nA12
nA13 = dat.nA13

# Inicialización de las armaduras de refuerzo en esquinas
fiA14 = dat.fiA14
fiA15 = dat.fiA15
fiA16 = fiA15
fiA17 = fiA14
nA14 = dat.nA14
nA15 = dat.nA15
nA16 = nA15
nA17 = nA14

# Inicialización de las armaduras de refuerzo de flexión
fiA6 = dat.fiA6
nA6 = dat.nA6

# Inicialización de las armaduras de cortante
fiA18 = dat.fiA18
fiA19 = fiA18
fiA20 = dat.fiA20
fiA21 = fiA20
```

```
sVA18 = dat.sA18
sVA19 = sVA18
sVA20 = dat.sA20
sVA21 = sVA20

# Longitudes de REFUERZO en esquinas
A22 = dat.A22  # Longitud de refuerzo A14 en Z
A32 = dat.A32  # Longitud de refuerzo A14 en X
A23 = dat.A23  # Longitud de refuerzo A15 en Z
A33 = dat.A33  # Longitud de refuerzo A15 en X
A24 = A23  # Longitud de refuerzo A16 en Z
A34 = A33  # Longitud de refuerzo A16 en X
A25 = A22  # Longitud de refuerzo A17 en Z
A35 = A32  # Longitud de refuerzo A17 en X

# Distancia armaduras de CORTANTE
A26 = dat.A26  # Distancia armadura cortante A18 losa superior
A27 = A26  # Distancia armadura cortante A19 losa superior
A28 = dat.A28  # Distancia armadura cortante A20 losa inferior
A29 = A28  # Distancia armadura cortante A21 losa inferior

# Longitud armadura de REFUERZO losa superior
A30 = dat.A30

# Inicialización de variables y control de errores
fibase1 = - 1
fibase2 = - 1
nbase1 = - 1
nbase2 = - 1
firefesq = - 1
nrefesq = - 1
firef = - 1
nref = - 1
fiV = - 1
sV = - 1

# Inicialización de los vectores de almacenamietno de cada variable
reinforcement = np.zeros((10, len(nodes)))
```

```
for i, node in enumerate(nodes):
    x = nodes[i, 0]
    z = nodes[i, 1]

    # HASTIAL IZQUIERDO
    if thetas[i] == np.deg2rad(90):
        if z <= L[0]:
            # Armadura BASE inferior y superior
            fibase1 = fiA1
            fibase2 = fiA2
            nbase1 = nA1
            nbase2 = nA2

            # Armadura REFUERZO de las ESQUINAS
            if z <= A22:
                firefesq = fiA14
                nrefesq = nA14
            else:
                firefesq = 0
                nrefesq = 0

            # Armadura REFUERZO FLEXIÓN
            firef = 0
            nref = 0

            # Armadura de CORTANTE
            if z == 0:
                fiV = fiA21
                sV = sVA21
            else:
                fiV = 0
                sV = 0
        else:
            # Armadura BASE inferior y superior
            fibase1 = fiA3
            fibase2 = fiA4
            nbase1 = nA3
            nbase2 = nA4
```

```
            # Armadura REFUERZO de las ESQUINAS
            if z >= (dat.H - A23):
                firefesq = fiA15
                nrefesq = nA15
            else:
                firefesq = 0
                nrefesq = 0

            # Armadura REFUERZO FLEXIÓN
            firef = 0
            nref = 0

            # Armadura de CORTANTE
            if z == dat.H:
                fiV = fiA18
                sV = sVA18
            else:
                fiV = 0
                sV = 0

    # LOSA SUPERIOR
    if thetas[i] == np.deg2rad(0):

        # Armadura BASE inferior y superior
        fibase1 = fiA5
        fibase2 = fiA7
        nbase1 = nA5
        nbase2 = nA7
        # Armadura REFUERZO de las ESQUINAS
        if x <= A33:
            firefesq = fiA15
            nrefesq = nA15
        elif x >= (dat.B - A34):
            firefesq = fiA16
            nrefesq = nA16
        else:
            firefesq = 0
            nrefesq = 0
```

```
        # Armadura REFUERZO FLEXIÓN
        liminfrefls = (dat.B - A30) / 2  # Posición X donde empieza refuerzo
        ↪  de flexión
        limsuprefls = liminfrefls + A30  # Posición X donde termina refuerzo
        ↪  de flexión
        if liminfrefls <= x <= limsuprefls:
            firef = fiA6
            nref = nA6
        else:
            firef = 0
            nref = 0

        # Armadura de CORTANTE
        if x <= A26:
            fiV = fiA18
            sV = sVA18
        elif x >= (dat.B - A27):
            fiV = fiA19
            sV = sVA19
        else:
            fiV = 0
            sV = 0

    # HASTIAL DERECHO
    if thetas[i] == np.deg2rad(270):

        if z > (dat.H - L[3]):
            # Armadura BASE inferior y superior
            fibase1 = fiA8
            fibase2 = fiA9
            nbase1 = nA8
            nbase2 = nA9

            # Armadura REFUERZO de las ESQUINAS
            if z >= (dat.B - A24):
                firefesq = fiA16
                nrefesq = nA16
            else:
                firefesq = 0
```

```
            nrefesq = 0

        # Armadura REFUERZO FLEXIÓN
        firef = 0
        nref = 0

        # Armadura de CORTANTE
        if z == dat.H:
            fiV = fiA19
            sV = sVA19
        else:
            fiV = 0
            sV = 0

    else:
        # Armadura BASE inferior y superior
        fibase1 = fiA10
        fibase2 = fiA11
        nbase1 = nA10
        nbase2 = nA11

        # Armadura REFUERZO de las ESQUINAS
        if z <= A25:
            firefesq = fiA17
            nrefesq = nA17
        else:
            firefesq = 0
            nrefesq = 0

        # Armadura REFUERZO FLEXIÓN
        firef = 0
        nref = 0

        # Armadura de CORTANTE
        if z == 0:
            fiV = fiA20
            sV = sVA20
        else:
            fiV = 0
```

```
            sV = 0

# LOSA INFERIOR
if thetas[i] == np.deg2rad(180):

    # Armadura BASE inferior y superior
    fibase1 = fiA12
    fibase2 = fiA13
    nbase1 = nA12
    nbase2 = nA13

    # Armadura REFUERZO de las ESQUINAS
    if x >= (dat.B - A35):
        firefesq = fiA17
        nrefesq = nA17
    elif x <= A32:
        firefesq = fiA14
        nrefesq = nA14
    else:
        firefesq = 0
        nrefesq = 0

    # Armadura REFUERZO FLEXIÓN
    firef = 0
    nref = 0

    # Armadura de CORTANTE
    if x >= (dat.B - A28):
        fiV = fiA20
        sV = sVA20
    elif x <= A29:
        fiV = fiA21
        sV = sVA21
    else:
        fiV = 0
        sV = 0

reinforcement[0, i] = fibase1
reinforcement[1, i] = fibase2
```

```
            reinforcement[2, i] = nbase1
            reinforcement[3, i] = nbase2
            reinforcement[4, i] = firefesq
            reinforcement[5, i] = nrefesq
            reinforcement[6, i] = firef
            reinforcement[7, i] = nref
            reinforcement[8, i] = fiV
            reinforcement[9, i] = sV

            print(' fibase1', fibase1, '\n',
                  'fibase2', fibase2, '\n',
                  'nbase1', nbase1, '\n',
                  'nbase2', nbase2, '\n',
                  'firefesq', firefesq, '\n',
                  'nrefesq', nrefesq, '\n',
                  'firef', firef, '\n',
                  'nref', nref, '\n',
                  'fiV', fiV, '\n', )

        return reinforcement


    def SectionGeometryVerificator(nodes, reinforcement, canto):

        b = dat.b
        rec = dat.rec
        sepmin = 0.025

        fibase1 = reinforcement[0, i]
        fibase2 = reinforcement[1, i]
        nbase1 = reinforcement[2, i]
        nbase2 = reinforcement[3, i]
        firefesq = reinforcement[4, i]
        nrefesq = reinforcement[5, i]
        firef = reinforcement[6, i]
        nref = reinforcement[7, i]
        fiV = reinforcement[8, i]

        coefgeom = np.zeros((len(nodes)))
```

```
    for i, node in enumerate(nodes):
        sepinf = (b - 2 * rec - 2 * fiV[i] - nbase1[i] * fibase1[i]) / (nbase1[i]
        ↪  - 1)
        sepsup = (b - 2 * rec - 2 * fiV[i] - nbase2[i] * fibase2[i]) / (nbase2[i]
        ↪  - 1)

        if firefesq[i] != 0:
            seprefesq = (b - 2 * rec - 2 * fiV[i] - nrefesq[i] * firefesq[i]) /
            ↪  (nrefesq[i] - 1)
        else:
            seprefesq = 1e10
        if firef[i] != 0:
            sepref = (b - 2 * rec - 2 * fiV[i] - nref[i] * firef[i]) / (nref[i] -
            ↪  1)
        else:
            sepref = 1e10

        if min(sepinf, sepsup, seprefesq, sepref) >= sepminima:
            coefgeom[i] = 1
        elif min(sepinf, sepsup, seprefesq, sepref) < sepminima:
            coefgeom[i] = - 1

    return coefgeom


def nodeMechanicalPorperties(members, nodes, reinforcement, canto):
    E = np.zeros((len(members)))
    b = np.zeros((len(members)))
    h = canto[:]
    A = np.zeros((len(members)))
    I = np.zeros((len(members)))
    fi1 = np.zeros((len(members)))
    fi2 = np.zeros((len(members)))
    ns1 = np.zeros((len(members)))
    ns2 = np.zeros((len(members)))
    As1 = np.zeros((len(members)))
    As2 = np.zeros((len(members)))
    firefesq = np.zeros((len(members)))
    nsrefesq = np.zeros((len(members)))
```

```
    Asrefesq = np.zeros((len(members)))
    firef = np.zeros((len(members)))
    nsref = np.zeros((len(members)))
    Asref = np.zeros((len(members)))
    fiw = np.zeros((len(members)))
    Asw = np.zeros((len(members)))
    sw = np.zeros((len(members)))
    for i, node in enumerate(nodes):
        E[i] = dat.E
        b[i] = dat.b
        h[i] = canto[i]
        A[i] = b[i] * h[i]
        I[i] = (b[i] * h[i] ** 3) / 12
        fi1[i] = reinforcement[0, i]
        fi2[i] = reinforcement[1, i]
        ns1[i] = reinforcement[2, i]
        ns2[i] = reinforcement[3, i]
        As1[i] = ns1[i] * np.pi * (fi1[i] / 2) ** 2
        As2[i] = ns2[i] * np.pi * (fi2[i] / 2) ** 2
        firefesq[i] = reinforcement[4, i]  # Armadura de refuerzo
        nsrefesq[i] = reinforcement[5, i]
        Asrefesq[i] = nsrefesq[i] * np.pi * (firefesq[i] / 2) ** 2
        firef[i] = reinforcement[6, i]
        nsref[i] = reinforcement[7, i]
        Asref[i] = nsref[i] * np.pi * (firef[i] / 2) ** 2
        fiw[i] = reinforcement[8, i]
        Asw[i] = np.pi * (fiw[i] / 2) ** 2
        sw[i] = reinforcement[9, i]

    return E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2, firefesq, nsrefesq,
    ↪  Asrefesq, firef, nsref, Asref, Asw, sw

def minimumReinforcementVerification(members, nodes, As1, Asref, h, I, fctm,
↪  fyd):
    coefminAs = np.zeros((len(nodes)))
    coefmaxAs = np.zeros((len(nodes)))

    for i, node in enumerate(nodes):
        z = 0.8 * h[i]
```

```
            W = I[i] / (h[i] / 2)
            fctmfl = max((1.6 - h[i]) * fctm, fctm)
            Asmin = W / z * fctmfl / fyd

            Asmax = 0.04 * dat.b * h[k]

            Smax = min((3 * h[i]), 0.4)

            coef = 0
            if (As1[i] + Asref[i]) >= Asmin:
                coef = 1
            elif (As1[i] + Asref[i]) < Asmin:
                coef = - 1
            coefminAs[i] = coef

            coef2 = 0
            if (As1[i] + Asref[i]) <= Asmax:
                coef2 = 1
            elif (As1[i] + Asref[i]) > Asmax:
                coef2 = - 1

            coefmaxAs[i] = coef2

        return coefminAs, coefmaxAs


    def additionalReinforcementValues(nb, L, As1, Asref, h):

        Astr1H = (0.0032 * L[0] * h[0]) * 0.40
        Astr2H = (0.0032 * L[0] * h[0]) * 0.60
        AstrHastiales = Astr1H + Astr2H

        seccl = nb[0] + nb[i] + int(nb[2] / 2)
        Astr1L = 0.20 * (As1[seccl] + Asref[seccl]) * L[2]
        Astr2L = 0.20 * (As1[seccl] + Asref[seccl]) * L[2]
        AstrLosas = Astr1L + Astr2L

        Atransversal = 2 * (AstrLosas + AstrHastiales)
```

```
    return Atransversal
```

A.5.2. Cálculo de esfuerzos internos

```
import numpy as np
import data as dat


def calculateK(E, A, L, I, theta):
    c = np.cos(theta)
    s = np.sin(theta)
    TM = np.array(((c, s, 0, 0, 0, 0),
                   (- s, c, 0, 0, 0, 0),
                   (0, 0, 1, 0, 0, 0),
                   (0, 0, 0, c, s, 0),
                   (0, 0, 0, - s, c, 0),
                   (0, 0, 0, 0, 0, 1)))

    k11 = E * A / L
    k12 = 0
    k13 = 0
    k14 = -E * A / L
    k15 = 0
    k16 = 0
    k21 = 0
    k22 = 12 * E * I / L ** 3
    k23 = -6 * E * I / L ** 2
    k24 = 0
    k25 = -12 * E * I / L ** 3
    k26 = -6 * E * I / L ** 2
    k31 = 0
    k32 = -6 * E * I / L ** 2
    k33 = 4 * E * I / L
    k34 = 0
    k35 = 6 * E * I / L ** 2
    k36 = 2 * E * I / L
    k41 = -E * A / L
    k42 = 0
```

```
    k43 = 0
    k44 = E * A / L
    k45 = 0
    k46 = 0
    k51 = 0
    k52 = -12 * E * I / L ** 3
    k53 = 6 * E * I / L ** 2
    k54 = 0
    k55 = 12 * E * I / L ** 3
    k56 = 6 * E * I / L ** 2
    k61 = 0
    k62 = -6 * E * I / L ** 2
    k63 = 2 * E * I / L
    k64 = 0
    k65 = 6 * E * I / L ** 2
    k66 = 4 * E * I / L

    K11 = np.array(((k11, k12, k13), (k21, k22, k23), (k31, k32, k33)))
    K12 = np.array(((k14, k15, k16), (k24, k25, k26), (k34, k35, k36)))
    K21 = np.array(((k41, k42, k43), (k51, k52, k53), (k61, k62, k63)))
    K22 = np.array(((k44, k45, k46), (k54, k55, k56), (k64, k65, k66)))

    top = np.concatenate((K11, K12), axis=1)
    btm = np.concatenate((K21, K22), axis=1)
    Kl = np.concatenate((top, btm), axis=0)

    Kg = TM.T.dot(Kl).dot(TM)

    K11g = Kg[0:3, 0:3]
    K12g = Kg[0:3, 3:6]
    K21g = Kg[3:6, 0:3]
    K22g = Kg[3:6, 3:6]

    return K11g, K12g, K21g, K22g


def calculateKpl(E, A, L, I, theta):
    c = np.cos(theta)
    s = np.sin(theta)
```

```
TM = np.array(((c, s, 0, 0, 0),
               (- s, c, 0, 0, 0),
               (0, 0, c, s, 0),
               (0, 0, - s, c, 0),
               (0, 0, 0, 0, 1)))

k11 = E * A / L
k12 = 0
k13 = -E * A / L
k14 = 0
k15 = 0
k21 = 0
k22 = 3 * E * I / L ** 3
k23 = 0
k24 = -3 * E * I / L ** 3
k25 = -3 * E * I / L ** 2
k31 = -E * A / L
k32 = 0
k33 = E * A / L
k34 = 0
k35 = 0
k41 = 0
k42 = -3 * E * I / L ** 3
k43 = 0
k44 = 3 * E * I / L ** 3
k45 = 3 * E * I / L ** 2
k51 = 0
k52 = -3 * E * I / L ** 2
k53 = 0
k54 = 3 * E * I / L ** 2
k55 = 3 * E * I / L

K11 = np.array(((k11, k12), (k21, k22)))
K12 = np.array(((k13, k14, k15), (k23, k24, k25)))
K21 = np.array(((k31, k32), (k41, k42), (k51, k52)))
K22 = np.array(((k33, k34, k35), (k43, k44, k45), (k53, k54, k55)))

top = np.concatenate((K11, K12), axis=1)
btm = np.concatenate((K21, K22), axis=1)
```

```
    Kl = np.concatenate((top, btm), axis=0)

    Kg = TM.T.dot(Kl).dot(TM)

    K11g = Kg[0:2, 0:2]
    K12g = Kg[0:2, 2:5]
    K21g = Kg[2:5, 0:2]
    K22g = Kg[2:5, 2:5]

    return K11g, K12g, K21g, K22g


def calculateKpr(E, A, L, I, theta):
    c = np.cos(theta)
    s = np.sin(theta)
    TM = np.array(((c, s, 0, 0, 0),
                   (- s, c, 0, 0, 0),
                   (0, 0, 1, 0, 0),
                   (0, 0, 0, c, s),
                   (0, 0, 0, - s, c)))

    k11 = E * A / L
    k12 = 0
    k13 = 0
    k14 = -E * A / L
    k15 = 0
    k21 = 0
    k22 = 3 * E * I / L ** 3
    k23 = -3 * E * I / L ** 2
    k24 = 0
    k25 = -3 * E * I / L ** 3
    k31 = 0
    k32 = -3 * E * I / L ** 2
    k33 = 3 * E * I / L
    k34 = 0
    k35 = 3 * E * I / L ** 2
    k41 = -E * A / L
    k42 = 0
    k43 = 0
```

```
    k44 = E * A / L
    k45 = 0
    k51 = 0
    k52 = -3 * E * I / L ** 3
    k53 = 3 * E * I / L ** 2
    k54 = 0
    k55 = 3 * E * I / L ** 3

    K11 = np.array(((k11, k12, k13), (k21, k22, k23), (k31, k32, k33)))
    K12 = np.array(((k14, k15), (k24, k25), (k34, k35)))
    K21 = np.array(((k41, k42, k43), (k51, k52, k53)))
    K22 = np.array(((k44, k45), (k54, k55)))

    top = np.concatenate((K11, K12), axis=1)
    btm = np.concatenate((K21, K22), axis=1)
    Kl = np.concatenate((top, btm), axis=0)

    Kg = TM.T.dot(Kl).dot(TM)

    K11g = Kg[0:3, 0:3]
    K12g = Kg[0:3, 3:5]
    K21g = Kg[3:5, 0:3]
    K22g = Kg[3:5, 3:5]

    return K11g, K12g, K21g, K22g


def calculateKET(Kbalasto, b, L):

    k11 = 0
    k12 = Kbalasto * b * L
    k13 = 0
    k14 = 0
    k15 = Kbalasto * b * L
    k16 = 0
    k21 = Kbalasto * b * L
    k22 = 0
    k23 = 0
    k24 = Kbalasto * b * L
```

```
k25 = 0
k26 = 0
k31 = 0
k32 = 0
k33 = 0
k34 = 0
k35 = 0
k36 = 0
k41 = Kbalasto * b * L
k42 = 0
k43 = 0
k44 = 0
k45 = Kbalasto * b * L
k46 = 0
k51 = Kbalasto * b * L
k52 = 0
k53 = 0
k54 = Kbalasto * b * L
k55 = 0
k56 = 0
k61 = 0
k62 = 0
k63 = 0
k64 = 0
k65 = 0
k66 = 0
K11 = np.array(((k11, k12, k13), (k21, k22, k23), (k31, k32, k33)))
K12 = np.array(((k14, k15, k16), (k24, k25, k26), (k34, k35, k36)))
K21 = np.array(((k41, k42, k43), (k51, k52, k53), (k61, k62, k63)))
K22 = np.array(((k44, k45, k46), (k54, k55, k56), (k64, k65, k66)))

top = np.concatenate((K11, K12), axis=1)
btm = np.concatenate((K21, K22), axis=1)
Kg = np.concatenate((top, btm), axis=0)

K11g = Kg[0:3, 0:3]
K12g = Kg[0:3, 3:6]
K21g = Kg[3:6, 0:3]
K22g = Kg[3:6, 3:6]
```

```
    return K11g, K12g, K21g, K22g


def buildKp(members, pins, E, A, I, L, theta):
    nDoF = int(np.amax(members * 3))
    Kp = np.zeros((nDoF, nDoF))

    for i, member in enumerate(members):

        # Definimos el nodo inicial y final de cada miembro para indexarlos
        node_i = int(member[0])
        node_j = int(member[1])

        # Rótula en el Nodo j del miembro i+1 (Extremo derecho de la barra)
        if pins[i, 1] == 0:
            K11, K12, K21, K22 = calculateKgPinRight(E[i], A[i], L[i], I[i],
            ↪  theta[i])
            # Indexado de los nodos para establecer la posición que ocupan dentro
            ↪  del modelo
            ia = 3 * node_i - 3  # index 0 (Ej. nodo 1) - sin rótula
            ib = 3 * node_i - 1  # index 2 (Ej. nodo 1) - sin rótula
            ja = 3 * node_j - 3  # index 3 (Ej. nodo 2) - sin rótula
            jb = 3 * node_j - 2  # index 4 (Ej. nodo 1) - con rótula
        # Rótula en el Nodo i del miembro i+1 (Extremo izquierdo de la barra)
        elif pins[i, 0] == 0:
            K11, K12, K21, K22 = calculateKgPinLeft(E[i], A[i], L[i], I[i],
            ↪  theta[i])
            # Indexado de los nodos para establecer la posición que ocupan dentro
            ↪  del modelo
            ia = 3 * node_i - 3  # index 0 (Ej. nodo 1) - sin rótula
            ib = 3 * node_i - 2  # index 1 (Ej. nodo 1) - con rótula
            ja = 3 * node_j - 3  # index 3 (Ej. nodo 2) - sin rótula
            jb = 3 * node_j - 1  # index 5 (Ej. nodo 1) - sin rótula
        # Barra empotrada-empotrada sin rótulas
        else:
            K11, K12, K21, K22 = calculateKg(E[i], A[i], L[i], I[i], theta[i])
            if theta[i] == np.deg2rad(180):
```

```
                K11muelle, K12muelle, K21muelle, K22muelle =
                ↪  calculateKgElasticTerrain(10000e03, dat.b, L[i])
                K11 = K11 + K11muelle
                K12 = K12 + K12muelle
                K21 = K21 + K21muelle
                K22 = K22 + K22muelle
            # Indexado de los nodos para establecer la posición que ocupan dentro
            ↪  del modelo
            ia = 3 * node_j - 3  # index 0 (Ej. nodo 1) - sin rótula
            ib = 3 * node_j - 1  # index 1 (Ej. nodo 1) - sin rótula
            ja = 3 * node_i - 3  # index 3 (Ej. nodo 2) - sin rótula
            jb = 3 * node_i - 1  # index 5 (Ej. nodo 1) - sin rótula

        # Ensamblaje de la matriz de rigidez primaria del modelo del marco
        Kp[ia:ib + 1, ia:ib + 1] = Kp[ia:ib + 1, ia:ib + 1] + K11
        Kp[ia:ib + 1, ja:jb + 1] = Kp[ia:ib + 1, ja:jb + 1] + K12
        Kp[ja:jb + 1, ia:ib + 1] = Kp[ja:jb + 1, ia:ib + 1] + K21
        Kp[ja:jb + 1, ja:jb + 1] = Kp[ja:jb + 1, ja:jb + 1] + K22

    return Kp


def buildKs(Kp, pinDoF, restrainedDoF):
    removedDoF = restrainedDoF + pinDoF
    removedIndex = [x - 1 for x in removedDoF]
    Ks = np.delete(Kp, removedIndex, 0)
    Ks = np.delete(Ks, removedIndex, 1)

    return Ks

def loadcalculator(members, nodes, nb, E, h, A, I, L, thetas, cc):
    qVector = np.zeros(len(members))
    axialqVector = np.zeros(len(members))
    trapq1Vector = np.zeros(len(members))
    trapq2Vector = np.zeros(len(members))
    localqVector = np.zeros(len(members))
    localLaVector = np.zeros(len(members))
    localLbVector = np.zeros(len(members))
    localLcVector = np.zeros(len(members))
```

```
if cc == 0:
    for i, member in enumerate(members):
        # Vhor = A[i] * L[i]
        Vhor = A[i]
        if np.rad2deg(thetas[i]) == 90:
            axialqVector[i] = Vhor * dat.pconc
        if np.rad2deg(thetas[i]) == 270:
            axialqVector[i] = - Vhor * dat.pconc
        elif np.rad2deg(thetas[i]) == 0 or np.rad2deg(thetas[i]) == 180:
            qVector[i] = Vhor * dat.pconc

if cc == 1:
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 0:
            # qVector[i] = (dat.etr * dat.B * 1) * dat.ptr
            qVector[i] = dat.etr * dat.ptr * (1 + 0.3 * dat.etr / dat.B)

if cc == 2:
    nbhizq = nb[0] + nb[1]
    nbhdch = nb[3] + nb[4]
    conti = 0
    contd = 0
    K0 = (1 - np.sin(dat.phi))

    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 90:
            # qsup = (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr * K0
            qsup = (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
            qinf = (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            slopei = (qsup - qinf) / nbhdch
            trapq1Vector[i] = qinf + conti * slopei
            trapq2Vector[i] = qinf + (conti + 1) * slopei
            conti = conti + 1
        elif np.rad2deg(thetas[i]) == 270:
            # qsup = (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr * K0
            qsup = (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
```

```
            qinf = (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            sloped = (qinf - qsup) / nbhizq
            trapq1Vector[i] = qsup + contd * sloped
            trapq2Vector[i] = qsup + (contd + 1) * sloped
            contd = contd + 1
        else:
            trapq1Vector[i] = 0
            trapq2Vector[i] = 0

if cc == 3:
    nbhizq = nb[0] + nb[1]
    nbhdch = nb[3] + nb[4]
    conti = 0
    contd = 0
    K0 = (1 - np.sin(dat.phi))

    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 90:
            # qsup = 0.9 * (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = 0.9 * (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr
            ↪  * K0
            qsup = 0.9 * (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
            qinf = 0.9 * (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            slopei = (qsup - qinf) / nbhdch
            trapq1Vector[i] = qinf + conti * slopei
            trapq2Vector[i] = qinf + (conti + 1) * slopei
            conti = conti + 1
        elif np.rad2deg(thetas[i]) == 270:
            # qsup = 1.1 * (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = 1.1 * (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr
            ↪  * K0
            qsup = 1.1 * (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
            qinf = 1.1 * (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            sloped = (qinf - qsup) / nbhizq
            trapq1Vector[i] = qsup + contd * sloped
            trapq2Vector[i] = qsup + (contd + 1) * sloped
            contd = contd + 1
        else:
            trapq1Vector[i] = 0
```

```
            trapq2Vector[i] = 0

if cc == 4:
    nbhizq = nb[0] + nb[1]
    nbhdch = nb[3] + nb[4]
    conti = 0
    contd = 0
    K0 = (1 - np.sin(dat.phi))

    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 90:
            # qsup = 1.1 * (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = 1.1 * (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr
            ↪  * K0
            qsup = 1.1 * (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
            qinf = 1.1 * (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            slopei = (qsup - qinf) / nbhdch
            trapq1Vector[i] = qinf + conti * slopei
            trapq2Vector[i] = qinf + (conti + 1) * slopei
            conti = conti + 1
        elif np.rad2deg(thetas[i]) == 270:
            # qsup = 0.9 * (dat.etr + h[i] / 2) * dat.B * 1 * dat.ptr * K0
            # qinf = 0.9 * (dat.etr + h[i] / 2 + dat.H) * dat.B * 1 * dat.ptr
            ↪  * K0
            qsup = 0.9 * (dat.etr + h[i] / 2) * 1 * dat.ptr * K0
            qinf = 0.9 * (dat.etr + h[i] / 2 + dat.H) * 1 * dat.ptr * K0
            sloped = (qinf - qsup) / nbhizq
            trapq1Vector[i] = qsup + contd * sloped
            trapq2Vector[i] = qsup + (contd + 1) * sloped
            contd = contd + 1
        else:
            trapq1Vector[i] = 0
            trapq2Vector[i] = 0

if cc == 5:
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 0:
            qVector[i] = dat.qsuIAP * (1 / (1 + dat.etr))
```

```
if cc == 6:
    K0 = (1 - np.sin(dat.phi))
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 90 or np.rad2deg(thetas[i]) == 270:
            qVector[i] = dat.qsuIAP * K0

if cc == 7:
    K0 = (1 - np.sin(dat.phi))
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 90:
            qVector[i] = dat.qsuIAP * K0

if cc == 8:
    K0 = (1 - np.sin(dat.phi))
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 270:
            qVector[i] = dat.qsuIAP * K0

if cc == 9:
    La = dat.Lacarro
    Lc = dat.Lccarro + 2 * dat.etr * np.sin(np.deg2rad(30))
    a = La - Lc / 2
    b = La + Lc / 2
    ql = (2 * dat.qcarro) / (np.pi * dat.etr ** 2)
    for i, member in enumerate(members):
        if np.rad2deg(thetas[i]) == 0:
            xi = nodes[i, 0]
            xf = nodes[i + 1, 0]
            if xi > b or xf < a:
                localqVector[i] = 0
                localLaVector[i] = 0
                localLbVector[i] = 0
                localLcVector[i] = 0
            elif xi <= a and xf >= b:
                localqVector[i] = ql
                localLaVector[i] = La - xi - Lc / 2
                localLbVector[i] = L[i] - localLaVector[i]
                localLcVector[i] = Lc
            elif xi <= a and xf < b:
```

```
                    localqVector[i] = ql
                    localLcVector[i] = xf - a
                    localLaVector[i] = L[i] - localLcVector[i] / 2
                    localLbVector[i] = L[i] - localLaVector[i]
                elif xi >= a and xf <= b:
                    localqVector[i] = ql
                    localLaVector[i] = L[i] / 2
                    localLbVector[i] = L[i] / 2
                    localLcVector[i] = L[i]
                elif xi > a and xf > b:
                    localqVector[i] = ql
                    localLcVector[j] = b - xi
                    localLaVector[j] = localLcVector[i] / 2
                    localLbVector[j] = L[i] - localLcVector[i] / 2

            else:
                localqVector[i] = 0
                localLaVector[i] = 0
                localLbVector[i] = 0
                localLcVector[i] = 0

    return qVector, axialqVector, trapq1Vector, trapq2Vector, localqVector,
    ↪  localLaVector, localLbVector, localLcVector


def loadcalculatorIt(members, nodes, nb, E, h, A, I, L, thetas):
    cct = dat.cct

    storageqVector = np.zeros((cct, len(members)))
    storageaxialqVector = np.zeros((cct, len(members)))
    storagetrapq1Vector = np.zeros((cct, len(members)))
    storagetrapq2Vector = np.zeros((cct, len(members)))
    storagelocalqVector = np.zeros((cct, len(members)))
    storagelocalLaVector = np.zeros((cct, len(members)))
    storagelocalLbVector = np.zeros((cct, len(members)))
    storagelocalLcVector = np.zeros((cct, len(members)))

    for cc in range(0, cct):
```

```
            storageqVector[cc, :], storageaxialqVector[cc, :],
            ↪ storagetrapq1Vector[cc, :], storagetrapq2Vector[cc, :],
            ↪ storagelocalqVector[cc, :], storagelocalLaVector[cc, :],
            ↪ storagelocalLbVector[cc, :], storagelocalLcVector[cc, :] =
            ↪ loadcalculator(members, nodes, dat.nb, E, h, A, I, L, thetas, cc)

    return storageqVector, storageaxialqVector, storagetrapq1Vector,
    ↪ storagetrapq2Vector, storagelocalqVector, storagelocalLaVector,
    ↪ storagelocalLbVector, storagelocalLcVector

def ENAtotbeamload(q, L, pins):
    fhi = 0
    fvi = 0
    mi = 0
    fhj = 0
    fvj = 0
    mj = 0

    if pins[0] == 1 and pins[1] == 1:
        fhi = 0
        fvi = - ((q * L) / 2)
        mi = ((q * L ** 2) / 12)
        fhj = 0
        fvj = - ((q * L) / 2)
        mj = (-((q * L ** 2) / 12))

    elif pins[0] == 1 and pins[1] == 0:
        fhi = 0
        fvi = - (5 * (q * L) / 8)
        mi = ((q * L ** 2) / 8)
        fhj = 0
        fvj = - (3 * (q * L) / 8)
        mj = 0

    elif pins[0] == 0 and pins[1] == 1:
        fhi = 0
        fvi = - (3 * (q * L) / 8)
        mi = 0
        fhj = 0
```

```
        fvj = - (5 * (q * L) / 8)
        mj = ((q * L ** 2) / 8)

    return fhi, fvi, mi, fhj, fvj, mj


def ENAaxialload(qx, L):
    fhi = - qtx * L / 2
    fvi = 0
    mi = 0
    fhj = - qtx * L / 2
    fvj = 0
    mj = 0

    return fhi, fvi, mi, fhj, fvj, mj


def ENAtrapload(q1, q2, L, pins):
    fhi = 0
    fvi = 0
    mi = 0
    fhj = 0
    fvj = 0
    mj = 0

    if pins[0] == 1 and pins[1] == 1:
        if q1 > q2:
            fhi = 0
            fvi = - ((L / 20) * (7 * q1 + 3 * q2))
            mi = ((L ** 2 / 60) * (3 * q1 + 2 * q2))
            fhj = 0
            fvj = - ((L / 20) * (3 * q1 + 7 * q2))
            mj = -((L ** 2 / 60) * (2 * q1 + 3 * q2))

        elif q1 < q2:
            fhi = 0
            fvi = - ((L / 20) * (3 * q1 + 7 * q2))
            mi = ((L ** 2 / 60) * (2 * q1 + 3 * q2))
            fhj = 0
```

```
            fvj = - ((L / 20) * (7 * q1 + 3 * q2))
            mj = -((L ** 2 / 60) * (3 * q1 + 2 * q2))

    elif pins[0] == 1 and pins[1] == 0:
        if q1 > q2:
            fhi = 0
            fvi = - ((9 * q1 + 16 * q2) * L / 40)
            mi = ((7 * q1 + 8 * q2) * L ** 2 / 120)
            fhj = 0
            fvj = - ((11 * q1 + 4 * q2) * L / 40)
            mj = 0

        elif q1 < q2:
            fhi = 0
            fvi = - ((22 * q1 + 8 * q2) * L / 80)
            mi = ((7 * q1 + 8 * q2) * L ** 2 / 120)
            fhj = 0
            fvj = - ((9 * q1 + 16 * q2) * L / 40)
            mj = 0

    elif pins[0] == 0 and pins[1] == 1:
        if q1 > q2:
            fhi = 0
            fvi = - ((11 * q1 + 4 * q2) * L / 40)
            mi = 0
            fhj = 0
            fvj = - ((9 * q1 + 16 * q2) * L / 40)
            mj = ((7 * q1 + 8 * q2) * L ** 2 / 120)

        elif q1 < q2:
            fhi = 0
            fvi = - ((9 * q1 + 16 * q2) * L / 40)
            mi = 0
            fhj = 0
            fvj = - ((22 * q1 + 8 * q2) * L / 80)
            mj = ((7 * q1 + 8 * q2) * L ** 2 / 120)

    return fhi, fvi, mi, fhj, fvj, mj
```

```
def ENAloclo(qloc, L, La, Lb, Lc):
    fhi = 0
    fvi = - (2 * qloc / L ** 3) * (L ** 3 - 3 * La ** 2 * L - (Lc / 2) ** 2 * L +
                                   2 * La ** 3 + 2 * La * (Lc / 2) ** 2)
    mi = (2 * qloc / 12 * L) * (12 * La * Lb ** 2 + 4 * (Lc / 2) ** 2 * (L - 3 *
    ↪  Lb))
    fhj = 0
    fvj = - (2 * qloc / L ** 3) * (3 * La ** 2 * L + (Lc / 2) ** 2 * L -
                                   2 * La ** 3 - 2 * La * (Lc / 2) ** 2)
    mj = - (2 * qloc / 12 * L) * (12 * La ** 2 * Lb + 4 * (Lc / 2) ** 2 * (L - 3
    ↪  * La))

    return fhi, fvi, mi, fhj, fvj, mj


def forceVectorcalc(members, pins, q, qx, q1t, q2t, ql, La, Lb, Lc, thetas,
↪  lenghts):
    DoF = int(np.amax(3 * members))
    feq = np.array([np.zeros(DoF)]).T

    for i, member in enumerate(members):
        node_i = int(member[0])
        node_j = int(member[1])

        c = np.cos(thetas[i])
        s = np.sin(thetas[i])

        TM = np.array(((c, s, 0, 0, 0, 0),
                       (-s, c, 0, 0, 0, 0),
                       (0, 0, 1, 0, 0, 0),
                       (0, 0, 0, c, s, 0),
                       (0, 0, 0, -s, c, 0),
                       (0, 0, 0, 0, 0, 1)))

        fhi1, fvi1, mi1, fhj1, fvj1, mj1 = ENAtotbeamload(q[i], lenghts[i],
        ↪  pins[i])

        fhi2, fvi2, mi2, fhj2, fvj2, mj2 = ENAaxialload(qx[i], lenghts[i])
```

```
        fhi3, fvi3, mi3, fhj3, fvj3, mj3 = ENAtrapload(q1t[i], q2t[i],
        ↪  lenghts[i], pins[i])

        fhi4, fvi4, mi4, fhj4, fvj4, mj4 = ENAlocload(ql[i], lenghts[i], La[i],
        ↪  Lb[i], Lc[i])

        Fhi = fhi1 + fhi2 + fhi3 + fhi4
        Fvi = fvi1 + fvi2 + fvi3 + fvi4
        Mri = mi1 + mi2 + mi3 + mi4
        Fhj = fhj1 + fhj2 + fhj3 + fhj4
        Fvj = fvj1 + fvj2 + fvj3 + fvj4
        Mrj = mj1 + mj2 + mj3 + mj4

        ia = 3 * node_i - 3
        ja = 3 * node_j - 3

        np.set_printoptions(3)
        ENA_local = np.array(([Fhi, Fvi, Mri, Fhj, Fvj, Mrj])).T
        ENA_global = np.matmul(TM.T, ENA_local)

        feq[ia] = feq[ia] + ENA_global[0]
        feq[ia + 1] = feq[ja + 1] + ENA_global[1]
        feq[ia + 2] = feq[ia + 2] + ENA_global[2]
        feq[ja] = feq[ja] + ENA_global[3]
        feq[ja + 1] = feq[ja + 1] + ENA_global[4]
        feq[ja + 2] = feq[ja + 2] + ENA_global[5]

    return feq


def forceVectorcalculatorIteration(members, pins, q, qx, q1t, q2t, ql, La, Lb,
↪  Lc, thetas, lenghts):
    cct = dat.cct
    DoF = int(np.amax(3 * members))

    storagefeq = np.zeros((DoF, dat.cct))

    for cc in range(0, cct):
```

```
        storagefeq[:, cc] = np.ravel(
            forceVectorcalculator(members, pins, q[cc, :], qx[cc, :], q1t[cc, :],
            ↪  q2t[cc, :], ql[cc, :], La[cc, :],
                                  Lb[cc, :], Lc[cc, :], thetas, lenghts))

    return storagefeq

def solveDisplacements(forceVector, Ks, pinDoF, restrainedDoF, members):
    removedDoF = restrainedDoF + pinDoF
    removedIndex = [x - 1 for x in removedDoF]

    f = np.delete(forceVector, removedIndex, 0)

    U = np.linalg.inv(Ks).dot(f)

    DoF = int(np.amax(members * 3))
    UG = np.zeros(DoF)
    aux = 0

    for i in np.arange(DoF):
        if i in removedIndex:
            UG[i] = 0
        else:
            UG[i] = U[aux]
            aux = aux + 1

    UG = np.array([UG]).T

    return UG


def solveDisplacementsIteration(forceVector, Ks, pinDoF, restrainedDoF, members):
    cct = dat.cct
    DoF = int(np.amax(members * 3))
    storageUG = np.zeros((DoF, cct))

    for cc in range(0, cct):
        storageUG[:, cc] = np.ravel(solveDisplacements(forceVector[:, cc], Ks,
        ↪  pinDoF, restrainedDoF, members))
```

```
    return storageUG


def solveReactions(Kp, UG):

    FG = np.matmul(Kp, UG)

    return FG


def solveReactionsIteration(Kp, UG):
    cct = dat.cct
    DoF = int(np.amax(members * 3))
    storageFG = np.zeros((DoF, cct))

    for cc in range(0, cct):
        storageFG[:, cc] = np.ravel(solveReactions(Kp, UG[:, cc]))

    return storageFG


def solveMemberForces(UG, members, pins, E, A, I, L, thetas):
    memberForces = np.array(())
    memberShears = np.zeros(members.shape)
    memberMoments = np.zeros(members.shape)


    for i, member in enumerate(members):

        node_i = int(member[0])
        node_j = int(member[1])

        c = np.cos(thetas[i])
        s = np.sin(thetas[i])

        if pins[i, 1] == 0:
            ia = 3 * node_i - 3
            ib = 3 * node_i - 1
```

```
        ja = 3 * node_j - 3
        jb = 3 * node_j - 2

        TM = np.array(((c, s, 0, 0, 0),
                       (- s, c, 0, 0, 0),
                       (0, 0, 1, 0, 0),
                       (0, 0, 0, c, s),
                       (0, 0, 0, - s, c)))

        disp = np.array([[UG[ia], UG[ia + 1], UG[ib], UG[ja], UG[jb]]]).T
        disp_local = np.matmul(TM, disp)

        F_axial = (A[i] * E[i] / L[i]) * (disp_local[3] - disp_local[0])

        K11, K12, K21, K22 = stiff.calculateKgPinRight(E[i], A[i], L[i],
        ↪  I[i], thetas[i])
        top = np.concatenate((K11, K12), axis=1)
        btm = np.concatenate((K21, K22), axis=1)
        Kg = np.concatenate((top, btm), axis=0)
        Kl = TM.dot(Kg).dot(TM.T)

        Mi = Kl[2, :].dot(disp_local)
        Mj = 0

        Vi = Kl[1, :].dot(disp_local)
        Vj = Kl[4, :].dot(disp_local)

        memberForces = np.append(memberForces, F_axial)
        memberShears[j, 0] = Vi
        memberShears[j, 1] = Vj
        memberMoments[j, 0] = Mi
        memberMoments[j, 1] = Mj

    elif pins[k, 0] == 0:
        ia = 3 * node_i - 3
        ib = 3 * node_i - 2
        ja = 3 * node_j - 3
        jb = 3 * node_j - 1
```

```
        TM = np.array(((c, s, 0, 0, 0),
                       (- s, c, 0, 0, 0),
                       (0, 0, c, s, 0),
                       (0, 0, - s, c, 0),
                       (0, 0, 0, 0, 1)))

        disp = np.array([[UG[ia], UG[ib], UG[ja], UG[ja + 1], UG[jb]]]).T
        disp_local = np.matmul(TM, disp)

        F_axial = (A[i] * E[i] / L[i]) * (disp_local[2] - disp_local[0])

        K11, K12, K21, K22 = stiff.calculateKgPinLeft(E[i], A[i], L[i], I[i],
        ↪  thetas[i])
        top = np.concatenate((K11, K12), axis=1)
        btm = np.concatenate((K21, K22), axis=1)
        Kg = np.concatenate((top, btm), axis=0)
        Kl = TM.dot(Kg).dot(TM.T)

        Mi = 0
        Mj = Kl[4, :].dot(disp_local)

        Vi = Kl[1, :].dot(disp_local)
        Vj = Kl[3, :].dot(disp_local)

        memberForces = np.append(memberForces, F_axial)
        memberShears[i, 0] = Vi
        memberShears[i, 1] = Vj
        memberMoments[i, 0] = Mi
        memberMoments[i, 1] = Mj

    else:
        ia = 3 * node_i - 3
        ib = 3 * node_i - 1
        ja = 3 * node_j - 3
        jb = 3 * node_j - 1

        TM = np.array(((c, s, 0, 0, 0, 0),
                       (- s, c, 0, 0, 0, 0),
                       (0, 0, 1, 0, 0, 0),
```

```
                        (0, 0, 0, c, s, 0),
                        (0, 0, 0, - s, c, 0),
                        (0, 0, 0, 0, 0, 1)))

            disp = np.array([[UG[ia], UG[ia + 1], UG[ib], UG[ja], UG[ja + 1],
            ↪  UG[jb]]]).T
            disp_local = np.matmul(TM, disp)

            F_axial = (E[i] * A[i] / L[i]) * (disp_local[3] - disp_local[0])

            K11, K12, K21, K22 = stiff.calculateKg(E[i], A[i], L[i], I[i],
            ↪  thetas[i])
            top = np.concatenate((K11, K12), axis=1)
            btm = np.concatenate((K21, K22), axis=1)
            Kg = np.concatenate((top, btm), axis=0)
            Kl = TM.dot(Kg).dot(TM.T)

            Mi = Kl[2, :].dot(disp_local)
            Mj = Kl[5, :].dot(disp_local)

            Vi = Kl[1, :].dot(disp_local)
            Vj = Kl[4, :].dot(disp_local)

            memberForces = np.append(memberForces, F_axial)
            memberShears[j, 0] = Vi
            memberShears[j, 1] = Vj
            memberMoments[i, 0] = Mi
            memberMoments[i, 1] = Mj

    return memberForces, memberShears, memberMoments


def solveMemberForcesIteration(UG, members, pins, E, A, I, L, thetas):
    cct = dat.cct
    storagememberForces = np.zeros((cct, len(members)))
    storagememberShears = np.zeros((cct, len(members), 2))
    storagememberMoments = np.zeros((cct, len(members), 2))

    for cc in range(0, cct):
```

```
        storagememberForces[cc], storagememberShears[cc],
        ↪  storagememberMoments[cc] = solveMemberForces(UG[:, cc], members,
        ↪  pins, E, A, I, L, thetas)

    return storagememberForces, storagememberShears, storagememberMoments


def removeENAfromReactions(members, FG, feq, thetas):

    DoF = int(np.amax(3 * members))
    FG_f = np.array([np.zeros(DoF)]).T

    for i, member in enumerate(members):
        node_i = int(member[0])
        node_j = int(member[1])

        ia = 3 * node_i - 3
        ja = 3 * node_j - 3

        ENA_global = np.array([[feq[ia], feq[ia + 1], feq[ia + 2], feq[ja],
        ↪  feq[ja + 1], feq[ja + 2]]]).T

        FG_f[ia] = FG[ia] - ENA_global[0]
        FG_f[ia + 1] = FG[ia + 1] - ENA_global[1]
        FG_f[ia + 2] = FG[ia + 2] - ENA_global[2]
        FG_f[ja] = FG[ja] - ENA_global[3]
        FG_f[ja + 1] = FG[ja + 1] - ENA_global[4]
        FG_f[ja + 2] = FG[ja + 2] - ENA_global[5]

    return FG_f


def removeENAfromReactionsIteration(members, FG, feq, thetas):
    cct = dat.cct
    DoF = int(np.amax(members * 3))
    storageFG_f = np.zeros((DoF, cct))

    for cc in range(0, cct):
```

```
        storageFG_f[:, cc] = np.ravel(removeENAfromReactions(members, FG[:, cc],
        ↪  feq[:, cc], thetas))

    return storageFG_f


def removeENAfromMemberActions(members, mbrForces, mbrShears, mbrMoments, q, qx,
↪  q1t, q2t, ql, La, Lb, Lc, lenghts, pins, thetas):

    mbrForcesReal = np.array([np.zeros(len(members))]).T
    mbrShearsReal = np.zeros(members.shape)
    mbrMomentsReal = np.zeros(members.shape)

    for i, member in enumerate(members):

        Ni = mbrForces[i]
        Vi = mbrShears[i, 0]
        Mi = mbrMoments[i, 0]
        Nj = mbrForces[i]
        Vj = mbrShears[i, 1]
        Mj = mbrMoments[i, 1]

        c = np.cos(thetas[i])
        s = np.sin(thetas[i])

        TM = np.array(((c, s, 0, 0, 0, 0),
                       (-s, c, 0, 0, 0, 0),
                       (0, 0, 1, 0, 0, 0),
                       (0, 0, 0, c, s, 0),
                       (0, 0, 0, -s, c, 0),
                       (0, 0, 0, 0, 0, 1)))

        fhi1, fvi1, mi1, fhj1, fvj1, mj1 = force.ENAtotbeamload(q[i], lenghts[i],
        ↪  pins[i])

        fhi2, fvi2, mi2, fhj2, fvj2, mj2 = force.ENAaxialload(qx[i], lenghts[i])

        fhi3, fvi3, mi3, fhj3, fvj3, mj3 = force.ENAtrapload(q1t[i], q2t[i],
        ↪  lenghts[i], pins[i])
```

```
        fhi4, fvi4, mi4, fhj4, fvj4, mj4 = force.ENAlocload(ql[i], lenghts[i],
        ↪  La[i], Lb[i], Lc[i])

        if thetas[i] == np.deg2rad(270):
            fhi2 = - 1 * fhi2
            fhj2 = - 1 * fhj2

        Fhi = fhi1 + fhi2 + fhi3 + fhi4
        Fvi = fvi1 + fvi2 + fvi3 + fvi4
        Mri = mi1 + mi2 + mi3 + mi4
        Fhj = fhj1 + fhj2 + fhj3 + fhj4
        Fvj = fvj1 + fvj2 + fvj3 + fvj4
        Mrj = mj1 + mj2 + mj3 + mj4

        ENA_local = np.array([[Fhi, Fvi, Mri, Fhj, Fvj, Mrj]]).T
        ENA_global = np.matmul(TM.T, ENA_local)

        calActions_local = np.array([[Ni, Vi, Mi, Nj, Vj, Mj]]).T
        calActions_global = np.matmul(TM.T, calActions_local)

        finalActions_local = np.matmul(TM, calActions_global - ENA_global)

        if thetas[i] != np.deg2rad(270):
            mbrForcesReal[i] = finalActions_local[0]
            mbrShearsReal[i, 0] = finalActions_local[1]
            mbrShearsReal[i, 1] = finalActions_local[4]
        else:
            mbrForcesReal[i] = - 1 * finalActions_local[0]
            mbrShearsReal[i, 0] = - 1 * finalActions_local[1]
            mbrShearsReal[i, 1] = - 1 * finalActions_local[4]

        mbrForcesReal[j] = finalActions_local[0]
        mbrShearsReal[i, 0] = finalActions_local[1]
        mbrShearsReal[i, 1] = finalActions_local[4]
        mbrMomentsReal[j, 0] = finalActions_local[2]
        mbrMomentsReal[j, 1] = finalActions_local[5]

    return mbrForcesReal, mbrShearsReal, mbrMomentsReal
```

```
def removeENAfromMemberActionsIteration(members, mbrForces, mbrShears,
↪  mbrMoments, q, qx, q1t, q2t, ql, La, Lb, Lc, lenghts, pins, thetas):
    cct = dat.cct
    storagembrForcesReal = np.zeros((cct, len(members), 1))
    storagembrShearsReal = np.zeros((cct, len(members), 2))
    storagembrMomentsReal = np.zeros((cct, len(members), 2))

    for cc in range(0, cct):
        storagembrForcesReal[cc], storagembrShearsReal[cc],
        ↪  storagembrMomentsReal[cc] = removeENAfromMemberActions(members,
        ↪  mbrForces[cc, :], mbrShears[cc], mbrMoments[cc], q[cc, :], qx[cc, :],
        ↪  q1t[cc, :], q2t[cc, :], ql[cc, :], La[cc, :], Lb[cc, :], Lc[cc, :],
        ↪  lenghts, pins, thetas)

    return storagembrForcesReal, storagembrShearsReal, storagembrMomentsReal
```

A.6. Cálculo de envolventes

```
import numpy as np
import stresscalculator as stress
import data as dat


def stressEnvel(members, N, V, M, UG):

    N_pp = N[0]
    V_pp = V[0]
    M_pp = M[0]

    N_cm = N[1]
    V_cm = V[1]
    M_cm = M[1]

    N_et1 = N[2]
    V_et1 = V[2]
    M_et1 = M[2]
```

```
N_et2 = N[3]
V_et2 = V[3]
M_et2 = M[3]

N_et3 = N[4]
V_et3 = V[4]
M_et3 = M[4]

N_suIAP1 = N[5]
V_suIAP1 = V[5]
M_suIAP1 = M[5]

N_suIAP2 = N[6]
V_suIAP2 = V[6]
M_suIAP2 = M[6]

N_suIAP3 = N[7]
V_suIAP3 = V[7]
M_suIAP3 = M[7]

N_suIAP4 = N[8]
V_suIAP4 = V[8]
M_suIAP4 = M[8]

N_vehIAP = N[9]
V_vehIAP = V[9]
M_vehIAP = M[9]

cp = 1.35
cv = 1.5
cn = 0

ELUf = np.array(((cp, cp, cv, cn, cn, cn, cn, cn, cn, cn),
                 (cp, cp, cv, cn, cn, cn, cn, cn, cn, cv),
                 (cp, cp, cv, cn, cn, cn, cn, cn, cv, cn),
                 (cp, cp, cv, cn, cn, cn, cn, cn, cv, cv),
                 (cp, cp, cv, cn, cn, cn, cn, cv, cn, cn),
                 (cp, cp, cv, cn, cn, cn, cn, cv, cn, cv),
```

(cp, cp, cv, cn, cn, cn, cv, cn, cn, cn),
(cp, cp, cv, cn, cn, cn, cv, cn, cn, cv),
(cp, cp, cv, cn, cn, cv, cn, cn, cn, cn),
(cp, cp, cv, cn, cn, cv, cn, cn, cn, cv),
(cp, cp, cv, cn, cn, cv, cn, cn, cv, cn),
(cp, cp, cv, cn, cn, cv, cn, cn, cv, cv),
(cp, cp, cv, cn, cn, cv, cn, cv, cn, cn),
(cp, cp, cv, cn, cn, cv, cn, cv, cn, cv),
(cp, cp, cv, cn, cn, cv, cv, cn, cn, cn),
(cp, cp, cv, cn, cn, cv, cv, cn, cn, cv),
(cp, cp, cn, cv, cn, cn, cn, cn, cn, cn),
(cp, cp, cn, cv, cn, cn, cn, cn, cn, cv),
(cp, cp, cn, cv, cn, cn, cn, cn, cv, cn),
(cp, cp, cn, cv, cn, cn, cn, cn, cv, cv),
(cp, cp, cn, cv, cn, cn, cn, cv, cn, cn),
(cp, cp, cn, cv, cn, cn, cn, cv, cn, cv),
(cp, cp, cn, cv, cn, cn, cv, cn, cn, cn),
(cp, cp, cn, cv, cn, cn, cv, cn, cn, cv),
(cp, cp, cn, cv, cn, cv, cn, cn, cn, cn),
(cp, cp, cn, cv, cn, cv, cn, cn, cn, cv),
(cp, cp, cn, cv, cn, cv, cn, cn, cv, cn),
(cp, cp, cn, cv, cn, cv, cn, cn, cv, cv),
(cp, cp, cn, cv, cn, cv, cn, cv, cn, cn),
(cp, cp, cn, cv, cn, cv, cn, cv, cn, cv),
(cp, cp, cn, cv, cn, cv, cv, cn, cn, cn),
(cp, cp, cn, cv, cn, cv, cv, cn, cn, cv),
(cp, cp, cn, cn, cv, cn, cn, cn, cn, cn),
(cp, cp, cn, cn, cv, cn, cn, cn, cn, cv),
(cp, cp, cn, cn, cv, cn, cn, cn, cv, cn),
(cp, cp, cn, cn, cv, cn, cn, cn, cv, cv),
(cp, cp, cn, cn, cv, cn, cn, cv, cn, cn),
(cp, cp, cn, cn, cv, cn, cn, cv, cn, cv),
(cp, cp, cn, cn, cv, cn, cv, cn, cn, cn),
(cp, cp, cn, cn, cv, cn, cv, cn, cn, cv),
(cp, cp, cn, cn, cv, cv, cn, cn, cn, cn),
(cp, cp, cn, cn, cv, cv, cn, cn, cn, cv),
(cp, cp, cn, cn, cv, cv, cn, cn, cv, cn),
(cp, cp, cn, cn, cv, cv, cn, cn, cv, cv),
(cp, cp, cn, cn, cv, cv, cn, cv, cn, cn),

```
                (cp, cp, cn, cn, cv, cv, cn, cv, cn, cv),
                (cp, cp, cn, cn, cv, cv, cv, cn, cn, cn),
                (cp, cp, cn, cn, cv, cv, cv, cn, cn, cv)))

N_combELUf = np.zeros((len(ELUf), len(members), 1))
V_combELUf = np.zeros((len(ELUf), len(members), 2))
M_combELUf = np.zeros((len(ELUf), len(members), 2))
UG_combELUf = np.zeros((len(members) * 3, len(ELUf)))

for i in range(0, len(ELUf)):
    for j in range(0, dat.cct):
        N_combELUf[i] = N_combELUf[i] + ELUf[i, j] * N[j]
        V_combELUf[i] = V_combELUf[i] + ELUf[i, j] * V[j]
        M_combELUf[i] = M_combELUf[i] + ELUf[i, j] * M[j]
        if ELUf[i, j] != cn:
            UG_combELUf[:, i] = UG_combELUf[:, i] + UG[:, j]
        elif ELUf[i, j] == cv or cp:
            UG_combELUf[:, i] = UG_combELUf[:, i] + 0

combNmax = - 1
combNmin = - 1
combVmax = - 1
combVmin = - 1
combMmax = - 1
combMmin = - 1

env = np.zeros((3, 2))

Nmax = 0
Nmin = - 1e10
for i in range(0, len(ELUf)):
    Nmaxaux = min(N_combELUf[i, :])
    Nminaux = max(N_combELUf[i, :])
    if Nmaxaux < Nmax:
        Nmax = Nmaxaux
        combNmax = i
    if Nminaux > Nmin:
        Nmin = Nminaux
        combNmin = i
```

```
    Vmax = 0
    Vmin = 1e10
    for i in range(0, len(ELUf)):
        Vmaxaux = V_combELUf[i, 4, 1]
        Vminaux = V_combELUf[i, 4, 1]
        if Vmaxaux > Vmax:
            Vmax = Vmaxaux
            combVmax = i
        if Vminaux < Vmin:
            Vmin = Vminaux
            combVmin = i

    Mmax = 0
    Mmin = 1e10
    for i in range(0, len(ELUf)):
        Mmaxaux = M_combELUf[i, 10, 0]
        Mminaux = M_combELUf[i, 10, 0]
        if Mmaxaux > Mmax:
            Mmax = Mmaxaux
            combMmax = i

        if Mminaux < Mmin:
            Mmin = Mminaux
            combMmin = i

    env = np.array(((combNmax, combNmin),
                    (combVmax, combVmin),
                    (combMmax, combMmin)))

    return ELUf, N_combELUf, V_combELUf, M_combELUf, UG_combELUf, env


def stressEnvelopesELS(members, N, V, M, UG):

    cp = 1.00
    cv = 1.00
    cn = 0
```

```
p0 = 0.60
p1 = 0.50
p2 = 0.20

ELScar = np.array(((cp, cp, cv, cn, cn, cn, cn, cn, cn, cn),
                   (cp, cp, cv, cn, cn, cn, cn, cn, cn, p0),
                   (cp, cp, cv, cn, cn, cn, cn, cn, p0, cn),
                   (cp, cp, cv, cn, cn, cn, cn, cn, p0, p0),
                   (cp, cp, cv, cn, cn, cn, cn, p0, cn, cn),
                   (cp, cp, cv, cn, cn, cn, cn, p0, cn, p0),
                   (cp, cp, cv, cn, cn, cn, p0, cn, cn, cn),
                   (cp, cp, cv, cn, cn, cn, p0, cn, cn, p0),
                   (cp, cp, cv, cn, cn, p0, cn, cn, cn, cn),
                   (cp, cp, cv, cn, cn, p0, cn, cn, cn, p0),
                   (cp, cp, cv, cn, cn, p0, cn, cn, p0, cn),
                   (cp, cp, cv, cn, cn, p0, cn, cn, p0, p0),
                   (cp, cp, cv, cn, cn, p0, cn, p0, cn, cn),
                   (cp, cp, cv, cn, cn, p0, cn, p0, cn, p0),
                   (cp, cp, cv, cn, cn, p0, p0, cn, cn, cn),
                   (cp, cp, cv, cn, cn, p0, p0, cn, cn, p0),
                   (cp, cp, cn, cv, cn, cn, cn, cn, cn, cn),
                   (cp, cp, cn, cv, cn, cn, cn, cn, cn, p0),
                   (cp, cp, cn, cv, cn, cn, cn, cn, p0, cn),
                   (cp, cp, cn, cv, cn, cn, cn, cn, p0, p0),
                   (cp, cp, cn, cv, cn, cn, cn, p0, cn, cn),
                   (cp, cp, cn, cv, cn, cn, cn, p0, cn, p0),
                   (cp, cp, cn, cv, cn, cn, p0, cn, cn, cn),
                   (cp, cp, cn, cv, cn, cn, p0, cn, cn, p0),
                   (cp, cp, cn, cv, cn, p0, cn, cn, cn, cn),
                   (cp, cp, cn, cv, cn, p0, cn, cn, cn, p0),
                   (cp, cp, cn, cv, cn, p0, cn, cn, p0, cn),
                   (cp, cp, cn, cv, cn, p0, cn, cn, p0, p0),
                   (cp, cp, cn, cv, cn, p0, cn, p0, cn, cn),
                   (cp, cp, cn, cv, cn, p0, cn, p0, cn, p0),
                   (cp, cp, cn, cv, cn, p0, p0, cn, cn, cn),
                   (cp, cp, cn, cv, cn, p0, p0, cn, cn, p0),
                   (cp, cp, cn, cn, cv, cn, cn, cn, cn, cn),
                   (cp, cp, cn, cn, cv, cn, cn, cn, cn, p0),
                   (cp, cp, cn, cn, cv, cn, cn, cn, p0, cn),
```

```
                    (cp, cp, cn, cn, cv, cn, cn, cn, p0, p0),
                    (cp, cp, cn, cn, cv, cn, cn, p0, cn, cn),
                    (cp, cp, cn, cn, cv, cn, cn, p0, cn, p0),
                    (cp, cp, cn, cn, cv, cn, p0, cn, cn, cn),
                    (cp, cp, cn, cn, cv, cn, p0, cn, cn, p0),
                    (cp, cp, cn, cn, cv, p0, cn, cn, cn, cn),
                    (cp, cp, cn, cn, cv, p0, cn, cn, cn, p0),
                    (cp, cp, cn, cn, cv, p0, cn, cn, p0, cn),
                    (cp, cp, cn, cn, cv, p0, cn, cn, p0, p0),
                    (cp, cp, cn, cn, cv, p0, cn, p0, cn, cn),
                    (cp, cp, cn, cn, cv, p0, cn, p0, cn, p0),
                    (cp, cp, cn, cn, cv, p0, p0, cn, cn, cn),
                    (cp, cp, cn, cn, cv, p0, p0, cn, cn, p0)))

ELSc = np.array(((cp, cp, cv, cn, cn, cn, cn, cn, cn, cn),
                 (cp, cp, cv, cn, cn, cn, cn, cn, cn, p2),
                 (cp, cp, cv, cn, cn, cn, cn, cn, p2, cn),
                 (cp, cp, cv, cn, cn, cn, cn, cn, p2, p2),
                 (cp, cp, cv, cn, cn, cn, cn, p2, cn, cn),
                 (cp, cp, cv, cn, cn, cn, cn, p2, cn, p2),
                 (cp, cp, cv, cn, cn, cn, p2, cn, cn, cn),
                 (cp, cp, cv, cn, cn, cn, p2, cn, cn, p2),
                 (cp, cp, cv, cn, cn, p2, cn, cn, cn, cn),
                 (cp, cp, cv, cn, cn, p2, cn, cn, cn, p2),
                 (cp, cp, cv, cn, cn, p2, cn, cn, p2, cn),
                 (cp, cp, cv, cn, cn, p2, cn, cn, p2, p2),
                 (cp, cp, cv, cn, cn, p2, cn, p2, cn, cn),
                 (cp, cp, cv, cn, cn, p2, cn, p2, cn, p2),
                 (cp, cp, cv, cn, cn, p2, p2, cn, cn, cn),
                 (cp, cp, cv, cn, cn, p2, p2, cn, cn, p2),
                 (cp, cp, cn, cv, cn, cn, cn, cn, cn, cn),
                 (cp, cp, cn, cv, cn, cn, cn, cn, cn, p2),
                 (cp, cp, cn, cv, cn, cn, cn, cn, p2, cn),
                 (cp, cp, cn, cv, cn, cn, cn, cn, p2, p2),
                 (cp, cp, cn, cv, cn, cn, cn, p2, cn, cn),
                 (cp, cp, cn, cv, cn, cn, cn, p2, cn, p2),
                 (cp, cp, cn, cv, cn, cn, p2, cn, cn, cn),
                 (cp, cp, cn, cv, cn, cn, p2, cn, cn, p2),
                 (cp, cp, cn, cv, cn, p2, cn, cn, cn, cn),
```

```
                    (cp, cp, cn, cv, cn, p2, cn, cn, cn, p2),
                    (cp, cp, cn, cv, cn, p2, cn, cn, p2, cn),
                    (cp, cp, cn, cv, cn, p2, cn, cn, p2, p2),
                    (cp, cp, cn, cv, cn, p2, cn, p2, cn, cn),
                    (cp, cp, cn, cv, cn, p2, cn, p2, cn, p2),
                    (cp, cp, cn, cv, cn, p2, p2, cn, cn, cn),
                    (cp, cp, cn, cv, cn, p2, p2, cn, cn, p2),
                    (cp, cp, cn, cn, cv, cn, cn, cn, cn, cn),
                    (cp, cp, cn, cn, cv, cn, cn, cn, cn, p2),
                    (cp, cp, cn, cn, cv, cn, cn, cn, p2, cn),
                    (cp, cp, cn, cn, cv, cn, cn, cn, p2, p2),
                    (cp, cp, cn, cn, cv, cn, cn, p2, cn, cn),
                    (cp, cp, cn, cn, cv, cn, cn, p2, cn, p2),
                    (cp, cp, cn, cn, cv, cn, p2, cn, cn, cn),
                    (cp, cp, cn, cn, cv, cn, p2, cn, cn, p2),
                    (cp, cp, cn, cn, cv, p2, cn, cn, cn, cn),
                    (cp, cp, cn, cn, cv, p2, cn, cn, cn, p2),
                    (cp, cp, cn, cn, cv, p2, cn, cn, p2, cn),
                    (cp, cp, cn, cn, cv, p2, cn, cn, p2, p2),
                    (cp, cp, cn, cn, cv, p2, cn, p2, cn, cn),
                    (cp, cp, cn, cn, cv, p2, cn, p2, cn, p2),
                    (cp, cp, cn, cn, cv, p2, p2, cn, cn, cn),
                    (cp, cp, cn, cn, cv, p2, p2, cn, cn, p2)))

N_combELScar = np.zeros((len(ELScar), len(members), 1))
V_combELScar = np.zeros((len(ELScar), len(members), 2))
M_combELScar = np.zeros((len(ELScar), len(members), 2))
UG_combELScar = np.zeros((len(members) * 3, len(ELScar)))

N_combELSc = np.zeros((len(ELSc), len(members), 1))
V_combELSc = np.zeros((len(ELSc), len(members), 2))
M_combELSc = np.zeros((len(ELSc), len(members), 2))
UG_combELSc = np.zeros((len(members) * 3, len(ELSc)))

##### CARACTERÍSTICA ############################################################

for i in range(0, len(ELScar)):
    for j in range(0, (dat.cct - 1)):
        N_combELScar[i] = N_combELScar[i] + ELScar[i, j] * N[j]
```

```
        V_combELScar[i] = V_combELScar[i] + ELScar[i, j] * V[j]
        M_combELScar[i] = M_combELScar[i] + ELScar[i, j] * M[j]
        if ELScar[i, j] != cn:
            UG_combELScar[:, i] = UG_combELScar[:, i] + UG[:, j]
        elif ELSc[i, j] == cv or cp:
            UG_combELScar[:, i] = UG_combELScar[:, i] + 0

combNmaxcar = - 1
combNmincar = - 1
combVmaxcar = - 1
combVmincar = - 1
combMmaxcar = - 1
combMmincar = - 1

Nmaxcar = 0
Nmincar = 1e10
for i in range(0, len(ELScar)):
    Nmaxauxcar = max(N_combELScar[i])
    Nminauxcar = min(N_combELScar[i])
    if Nmaxauxcar > Nmaxcar:
        Nmaxcar = Nmaxauxcar
        combNmaxcar = i
    if Nminauxcar < Nmincar:
        Nmincar = Nminauxcar
        combNmincar = i

Vmaxcar = 0
Vmincar = 1e10
for i in range(0, len(ELScar)):
    Vmaxauxcar = max(max(V_combELScar[i, :, 0]), max(V_combELScar[i, :, 1]))
    Vminauxcar = abs(min(min(V_combELScar[i, :, 0]), min(V_combELScar[i, :,
    ↪  1])))
    if Vmaxauxcar > Vmaxcar:
        Vmaxcar = Vmaxauxcar
        combVmaxcar = i
    if Vminauxcar < Vmincar:
        Vmincar = Vminauxcar
        combVmincar = i
```

```
Mmincar = 1e10
for i in range(0, len(ELScar)):
    Mmaxauxcar = max(max(M_combELScar[i, :, 0]), abs(min(M_combELScar[i, :,
    ↪  1])))
    Mminauxcar = min(min(M_combELScar[i, :, 0]), abs(max(M_combELScar[i, :,
    ↪  1])))
    if Mmaxauxcar > Mmaxcar:
        Mmaxcar = Mmaxauxcar
        combMmaxcar = i
    if Mminauxcar < Mmincar:
        combMmincar = i

envcar = np.array(((combNmaxcar, combNmincar),
                   (combVmaxcar, combVmincar),
                   (combMmaxcar, combMmincar)))

##### FRECUENTE ##################################################################

for i in range(0, len(ELSf)):
    for j in range(0, (dat.cct - 1)):
        N_combELSf[i] = N_combELSf[i] + ELSf[i, j] * N[j]
        V_combELSf[i] = V_combELSf[i] + ELSf[i, j] * V[j]
        M_combELSf[i] = M_combELSf[i] + ELSf[i, j] * M[j]
        if ELSf[i, j] != cn:
            UG_combELSf[:, i] = UG_combELSf[:, i] + UG[:, j]
        elif ELSf[i, j] == cv or cp:
            UG_combELSf[:, i] = UG_combELSf[:, i] + 0

combNmaxf = - 1
combNminf = - 1
combVmaxf = - 1
combVminf = - 1
combMmaxf = - 1
combMminf = - 1

Nmaxf = 0
Nminf = 1e10
for i in range(0, len(ELSf)):
```

```
    Nmaxauxf = max(N_combELSf[i])
    Nminauxf = min(N_combELSf[i])
    if Nmaxauxf > Nmaxf:
        Nmaxf = Nmaxauxf
        combNmaxf = i
    if Nminauxf < Nminf:
        Nminf = Nminauxf
        combNminf = i

Vmaxf = 0
Vminf = 1e10
for i in range(0, len(ELSf)):
    Vmaxauxf = max(max(V_combELSf[i, :, 0]), max(V_combELSf[i, :, 1]))
    Vminauxf = abs(min(min(V_combELSf[i, :, 0]), min(V_combELSf[i, :, 1])))
    if Vmaxauxf > Vmaxf:
        Vmaxf = Vmaxauxf
        combVmaxf = i
    if Vminauxf < Vminf:
        Vminf = Vminauxf
        combVminf = i

Mmaxf = 0
Mminf = 1e10
for i in range(0, len(ELSf)):
    Mmaxauxf = max(max(M_combELSf[i, :, 0]), abs(min(M_combELSf[i, :, 1])))
    Mminauxf = min(min(M_combELSf[i, :, 0]), abs(max(M_combELSf[i, :, 1])))
    if Mmaxauxf > Mmaxf:
        Mmaxf = Mmaxauxf
        combMmaxf = i
    if Mminauxf < Mminf:
        combMminf = i

envf = np.array(((combNmaxf, combNminf),
                 (combVmaxf, combVminf),
                 (combMmaxf, combMminf)))

##### CUASIPERMANENTE ########################################################

for i in range(0, len(ELSc)):
```

```
    for j in range(0, (dat.cct - 1)):
        N_combELSc[i] = N_combELSc[i] + ELSc[i, j] * N[j]
        V_combELSc[i] = V_combELSc[i] + ELSc[i, j] * V[j]
        M_combELSc[i] = M_combELSc[i] + ELSc[i, j] * M[j]
        if ELSc[i, j] != cn:
            UG_combELSc[:, i] = UG_combELSc[:, i] + UG[:, j]
        elif ELSc[i, j] == cv or cp:
            UG_combELSc[:, i] = UG_combELSc[:, i] + 0

combNmaxc = - 1
combNminc = - 1
combVmaxc = - 1
combVminc = - 1
combMmaxc = - 1
combMminc = - 1

Nmaxc = 0
Nminc = 1e10
for i in range(0, len(ELSc)):
    Nmaxauxc = max(N_combELSc[i])
    Nminauxc = min(N_combELSc[i])
    if Nmaxauxc > Nmaxc:
        Nmaxc = Nmaxauxc
        combNmaxc = i
    if Nminauxc < Nminc:
        Nminc = Nminauxc
        combNminc = i

Vmaxc = 0
Vminc = 1e10
for i in range(0, len(ELSc)):
    Vmaxauxc = max(max(V_combELSc[i, :, 0]), max(V_combELSc[i, :, 1]))
    Vminauxc = abs(min(min(V_combELSc[i, :, 0]), min(V_combELSc[i, :, 1])))
    if Vmaxauxc > Vmaxc:
        Vmaxc = Vmaxauxc
        combVmaxc = i
    if Vminauxc < Vminc:
        Vminc = Vminauxc
        combVminc = i
```

```
    Mmaxc = 0
    Mminc = 1e10
    for i in range(0, len(ELSc)):
        Mmaxauxc = max(max(M_combELSc[i, :, 0]), abs(min(M_combELSc[i, :, 1])))
        Mminauxc = min(min(M_combELSc[i, :, 0]), abs(max(M_combELSc[i, :, 1])))
        if Mmaxauxc > Mmaxc:
            Mmaxc = Mmaxauxc
            combMmaxc = i
        if Mminauxc < Mminc:
            combMminc = i

    envc = np.array(((combNmaxc, combNminc),
                     (combVmaxc, combVminc),
                     (combMmaxc, combMminc)))

    return ELScar, N_combELScar, V_combELScar, M_combELScar, UG_combELScar,
    ↪  envcar, \
           ELSc, N_combELSc, V_combELSc, M_combELSc, UG_combELSc, envc
```

A.7. Verificación de los estados límite

```
import numpy as np
import time

import data as dat
import modelgenerator as model
import sectionproperties as secprop
import stiffnesscalculator as stiff
import loadcalculator as load
import forcevectorcalculator as force
import stresscalculator as stress
import envelopescalculator as envelopes
import NMdiagramgenerator as NMdiagram
import sectionverificator as verif

def InitialModelGeneration():
```

```
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha, dat.condense)
    initialreinforcement = secprop.SectionReinforcementGenerator(nodes, thetas,
    ↪  dat.L)
    initialcanto = secprop.SectionDimensionGenerator(nodes, thetas)

    return nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF,
    ↪  restrains, initialreinforcement, initialcanto


def ModelVerification(nodes, members, pins, pinDoF, thetas, lenghts,
↪  restrainedDoF, restrains, reinforcement, canto):
    E, b, h, A, I, fi1, fi2, ns1, ns2, As1, As2, firefesq, nsrefesq, Asrefesq,
    ↪  firef, nsref, Asref, Asw, sw = \
        secprop.nodeMechanicalPorperties(members, nodes, reinforcement, canto)
    geomverif = secprop.SectionGeometryVerificator(nodes, reinforcement, canto)
    coefminAs, coefmaxAs = secprop.minimumReinforcementVerification(members,
    ↪  nodes, As1, Asref, canto, I, dat.fctm, dat.fyd)

    Kp = stiff.buildKp(members, pins, E, A, I, L=lenghts, theta=thetas)
    Ks = stiff.buildKs(Kp, pinDoF, restrainedDoF)

    q, qx, q1t, q2t, ql, La, Lb, Lc = load.loadcalculatorIteration(members,
    ↪  nodes, dat.nb, E, h, A, I, lenghts, thetas)
    feq = force.forceVectorcalculatorIteration(members, pins, q, qx, q1t, q2t,
    ↪  ql, La, Lb, Lc, thetas, lenghts)

    UG = stress.solveDisplacementsIteration(feq, Ks, pinDoF, restrainedDoF,
    ↪  members)
    FG = stress.solveReactions(Kp, UG)
    N, V, M = stress.solveMemberForcesIteration(UG, members, pins, E, A, I,
    ↪  lenghts, thetas)

    FG_f = stress.removeENAfromReactionsIteration(members, FG, feq, thetas)
    Nrea, Vrea, Mrea = stress.removeENAfromMemberActionsIteration(members, N, V,
    ↪  M, q, qx, q1t, q2t, ql, La, Lb, Lc, lenghts, pins, thetas)

    ELUf, N_ELU, V_ELU, M_ELU, UG_ELU, env = envelopes.stressEnvelopes(members,
    ↪  Nreal, Vreal, Mreal, UG)
```

```
ELScar, N_ELScar, V_ELScar, M_ELScar, UG_ELScar, envELScar, \
ELSc, N_ELSc, V_ELSc, M_ELSc, UG_ELSc, envELSc =
↪  envelopes.stressEnvelopesELS(members, Nreal, Vreal, Mreal, UG)

coefVELU, incMedELU = verif.VverificatorELUIteration(nodes, members, ELUf, h,
↪  dat.rec, As1, As2, Asrefesq, Asref,
                                                    dat.fck, dat.fcd,
                                                    ↪  dat.fyk, N_ELU,
                                                    ↪  M_ELU, V_ELU, Asw,
                                                    ↪  sw)

######## Comprobación ELU SOLICITACIONES NORMALES (NM) ####################


# Aplicación de DECALAJE por incremento de tracciones debido a esfuerzo
↪  cortante
M_ELUdec = verif.DecalajeApplication(members, incMedELU, M_ELU, ELUf)

# Generación de los diagramas NM de cada una de las secciones del modelo
Nufp, Mufp, Nufn, Mufn, Nuinf, Nusup, sep =
↪  NMdiagram.NMdiagramIteration(nodes, dat.Es, b, h, As2, As1, Asrefesq,

                                                                ↪  Asref,
                                                                ↪  dat.rec,
                                                                ↪  dat.fcd,
                                                                ↪  dat.fyd,
                                                                ↪  dat.fck,

                                                                ↪  dat.fyk,
                                                                ↪  12500)
# Verificación mediante comparación con diagrama NM generado
coefNELU, coefMELU = verif.NMverificatorELUIteration(nodes, Nufp, Mufp, Nufn,
↪  Mufn, Nuinf, Nusup, N_ELU, M_ELUdec,
                                                    sep, ELUf)

######## Comprobación ELS FISURACIÓN ######################################

coefELScar, coefELScua = verif.FisuracionverificatorELSIteration(nodes,
↪  ELScar, ELSc, As1, As2, Asrefesq, Asref,
```

```
                                                                    fi1, ns1,
                                                                    ↪ canto,
                                                                    ↪ I,
                                                                    ↪ M_ELScar,
                                                                    ↪ M_ELSc,
                                                                    ↪ 0.2)

    return geomverif, coefminAs, coefmaxAs, coefVELU, coefNELU, coefMELU,
    ↪ coefELScar, coefELScua


def CoefficientsCounter(nodes, geomverif, coefminAs, coefmaxAs, ELUf, ELScar,
↪ ELScua, coefVELU, coefNELU, coefMELU,
                        coefELScar, coefELScua):
    ncGEOM = 0
    ncAsMin = 0
    ncAsMax = 0

    for n, node in enumerate(nodes):
        # Verificación GEOMÉTRICA
        if geomverif[n] < 1:
            ncGEOM = ncGEOM + 1
        else:
            ncGEOM = ncGEOM + 0

        # Verificación ARMADURA MÍNIMA
        if coefminAs[n] < 1:
            ncAsMin = ncAsMin + 1
        else:
            ncAsMin = ncAsMin + 0

        # Verificación ARMADURA MÁXIMA
        if coefmaxAs[n] < 1:
            ncAsMax = ncAsMax + 1
        else:
            ncAsMax = ncAsMax + 0

    # Estado Límite Último (E.L.U.)
    ncVELU = 0
```

```
    ncNELU = 0
    ncMELU = 0

    for i, ELU in enumerate(ELUf):
        for j, node in enumerate(nodes):
            # Verificación ELU CORTANTE
            if coefVELU[i, j] < 1:
                ncVELU = ncVELU + 1
            else:
                ncVELU = ncVELU + 0

            # Verificación ELU AXIL
            if coefNELU[i, j] < 1:
                ncNELU = ncNELU + 1
            else:
                ncNELU = ncNELU + 0

            # Verificación ELU FLECTOR
            if coefMELU[i, j] < 1:
                ncMELU = ncMELU + 1
            else:
                ncMELU = ncMELU + 0

    ncFELS = 0
    for i, ELS in enumerate(ELScar):
        for j, node in enumerate(nodes):
            # Verificación ELS FISURACIÓN combinación CARACTERÍSTICA
            if coefELScar[i, j] < 1 or coefELScua[i, j] < 1:
                ncFELS = ncFELS + 1
            else:
                ncFELS = ncFELS + 0

    return ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS

import numpy as np
from bisect import bisect
```

```
def NMdiagram(Es, b, h, As2, As1, Asrefesq, Asref, rec, fcd, fyd, fck, fyk,
↪  num_iter):
    Nu = np.array(())
    Mu = np.array(())
    d = h - rec

    As2 = As2 + Asrefesq
    As1 = As1 + Asref

    fcd = fcd * 1e03
    fyd = fyd * 1e03
    Es = Es * 1e-03

    # HORMGIÓN
    EPcuc = 0.00175
    EPcuf = 0.0035

    # ACERO
    EPud = 0.01
    EPsy = fyd / Es

    Nu_neg = - (As2 + As1) * fyd
    if EPcuc < EPsy:
        sigS = Es * EPcuc
    else:
        sigS = fyd
    Nu_pos = (As2 + As1) * sigS + fcd * b * h

    Xab = (EPcuf / (EPcuf + EPud)) * d
    Xlim = (EPcuf / (EPcuf + EPsy)) * d

    x = - 800e-03
    aux = 0

    for i in range(0, num_iter):
        x = x + 1e-02

        if x <= 0:
            # Deformación
```

```
        EPs1 = EPud
        EPs2 = EPud * ((rec - x) / (d - x))
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd
        # Axil
        Nc = 0
        Ns1 = - As1 * sigAs1
        Ns2 = - As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)
        # Momento
        Mc = 0
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)
        dom = 1

    elif 0 < x <= Xab:
        # Deformación
        EPs1 = EPud
        EPs2 = EPud * ((k - rec) / (d - x))
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd
        # Axil
        Nc = fcd * 0.8 * x * b
        Ns1 = - As1 * sigAs1
        Ns2 = As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)
```

```
        # Momento
        Mc = Nc * (d / 2 - 0.8 * x / 2)
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)
        dom = 2

    elif Xab < x <= Xlim:
        # Deformación
        EPs1 = EPcuf * ((d - x) / x)
        EPs2 = EPcuf * ((x - rec) / x)
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd
        # Axil
        Nc = fcd * 0.8 * x * b
        Ns1 = - As1 * sigAs1
        Ns2 = As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)
        # Momento
        Mc = Nc * (h / 2 - 0.4 * x)
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)
        dom = 3

    elif Xlim < x <= d:
        # Deformación
        EPs1 = EPcuf * ((d - x) / x)
        EPs2 = EPcuf * ((x - rec) / x)
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
```

```
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd
        # Axil
        Nc = fcd * 0.8 * i * b
        Ns1 = - As1 * sigAs1
        Ns2 = As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)
        # Momento
        Mc = Nc * (j / 2 - 0.4 * x)
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)

    elif d < x <= h:
        x = x + 1e-02
        # Deformación
        EPs1 = EPcuf * ((x - d) / x)
        EPs2 = EPcuf * ((x - rec) / x)
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd
        # Axil
        Nc = fcd * 0.8 * x * b
        Ns1 = As1 * sigAs1
        Ns2 = As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)
        # Momento
        Mc = Nc * (h / 2 - 0.4 * x)
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)
```

```
    elif h < x:
        x = x + 1e-02
        # Deformación
        EPs1 = EPcuc * ((x - d) / (x - h / 2))
        EPs2 = EPcuc * ((x - rec) / (x - h / 2))
        if EPs1 < EPsy:
            sigAs1 = Es * EPs1
        else:
            sigAs1 = fyd
        if EPs2 < EPsy:
            sigAs2 = Es * EPs2
        else:
            sigAs2 = fyd

        Nc = fcd * (1 - (1 - 0.8) * h / x) * h * b
        Ns1 = As1 * sigAs1
        Ns2 = As2 * sigAs2
        Nu = np.append(Nu, Nc + Ns1 + Ns2)

        Mc = Nc * (h / 2 - (1 - (1 - 0.8) * h / x) * h / 2)
        Ms1 = Ns1 * (h / 2 - d)
        Ms2 = Ns2 * (h / 2 - rec)
        Mu = np.append(Mu, Mc + Ms1 + Ms2)
    aux = aux + 1
Nu_pos = Nu[aux - 1]

a = abs(np.max(Nu)) - abs(np.min(Nu))
b = abs(Mu[0]) - abs(Mu[aux - 1])
Nufn = - 1 * Nu[::-1] + a
if Mu[0] > Mu[aux - 1]:
    Mufn = - 1 * Mu[::-1] + b
else:
    Mufn = - 1 * Mu[::-1] - b

xi = Nu[0]
xf = Nu[aux - 1]
zi = Mu[0]
zf = Mu[aux - 1]
m = (zf - zi) / (xf - xi)
```

```
    sep = m * Nu[0]
    z0 = Mu[0] - sep
    separation = z0 + m * Nu

    return Nu, Mu, Nufn, Mufn, Nu_neg, Nu_pos, separation


def NMdiagramIteration(nodes, Es, b, h, As2, As1, Asrefesq, Asref, rec, fcd, fyd,
↪  fck, fyk, num_iter):
    Nufpstor = np.zeros((len(nodes), num_iter))
    Mufpstor = np.zeros((len(nodes), num_iter))
    Nufnstor = np.zeros((len(nodes), num_iter))
    Mufnstor = np.zeros((len(nodes), num_iter))
    Nu_negstor = np.zeros((len(nodes)))
    Nu_posstor = np.zeros((len(nodes)))
    separationstor = np.zeros((len(nodes), num_iter))

    prop = np.zeros((len(nodes), 5))

    for i, member in enumerate(nodes):
        prop[j] = np.array((h[i], As2[i], As1[i], Asrefesq[i], Asref[i]))
        copy = np.where((prop == prop[i]).all(1))[0]

        if i == 0:
            Nufp, Mufp, Nufn, Mufn, Nu_neg, Nu_pos, separation = NMdiagram(Es,
            ↪  b[i], h[i], As2[i], As1[i], Asrefesq[i], Asref[i], rec, fcd, fyd,
            ↪  fck, fyk, num_iter)

            Nufpstor[i] = Nufp
            Mufpstor[i] = Mufp
            Nufnstor[i] = Nufn
            Mufnstor[i] = Mufn
            Nu_negstor[i] = Nu_neg
            Nu_posstor[i] = Nu_pos
            separationstor[i] = separation

        elif i != 0 and (prop[i] == prop[i - 1]).all() == True:
```

```
            """Si la sección es igual a la contigua, o si ya existe una sección
            ↪  de las mismas características calculada, toma sus valores de la
            ↪  base de datos y no la calcula,
            esto permite ahorrar tiempo y optimizar el proceso."""
            Nufpstor[i] = Nufpstor[i - 1]
            Mufpstor[i] = Mufpstor[i - 1]
            Nufnstor[i] = Nufnstor[i - 1]
            Mufnstor[i] = Mufnstor[i - 1]
            Nu_negstor[i] = Nu_negstor[i - 1]
            Nu_posstor[i] = Nu_posstor[i - 1]
            separationstor[i] = separationstor[i - 1]

        elif i != 0 and (prop[i] == prop[i - 1]).all() == False:
            Nufp, Mufp, Nufn, Mufn, Nu_neg, Nu_pos, separation = NMdiagram(Es,
            ↪  b[i], h[i], As2[i], As1[i], Asrefesq[i], Asref[i], rec, fcd, fyd,
            ↪  fck, fyk, num_iter)

            Nufpstor[j] = Nufp
            Mufpstor[i] = Mufp
            Nufnstor[i] = Nufn
            Mufnstor[j] = Mufn
            Nu_negstor[i] = Nu_neg
            Nu_posstor[i] = Nu_pos
            separationstor[j] = separation

    return Nufpstor, Mufpstor, Nufnstor, Mufnstor, Nu_negstor, Nu_posstor,
    ↪  separationstor
```

A.8. Módulo de optimización

A.8.1. Evaluación de la función objetivo

```
import data as dat
import numpy as np


def meassurements(nodes, members, nb, h, A, L, l, As1, As2, Asrefesq, Asref, Asw,
↪  sw, fck, fyk):
    costetotal = np.zeros((3, 3))
```

```
datoshorm = np.array([[25, 88.86, 256.66, 402.44],  # medido en m3
                      [30, 97.80, 277.72, 428.29],
                      [35, 101.03, 278.04, 429.95],
                      [40, 104.08, 278.04, 429.95]])

datosaceros = np.array([[400, 1.40, 0.70, 3.38],  # medido en kg
                        [500, 1.42, 0.70, 3.38]])

concrete2 = 0
steel = 0
b = dat.b

sls = nb[0] + nb[1] + 1
sli = nb[0] + nb[1] + nb[2] + nb[3] + nb[4] + 1
vch = (b * h[0] * L[0] + L[1] - h[sls] / 2 - h[sli] / 2)
vcls = b * h[sls] * (L[2] - h[0])
vcli = b * h[sli] * (L[5] - h[0])
concrete1 = 2 * vch + vcls + vcli
concrete = concrete1

for i in range(0, len(datoshorm)):
    aux = datoshorm[i, 0]
    if aux == fck:
        for j in range(0, len(costetotal[0])):
            costetotal[0, j] = concrete * datoshorm[i, j + 1]

shs = nb[0] + 1
vshi = (As1[0] + As2[0]) * (L[0] + h[sli] / 2 + 4 * h[0])
vsresqhi = Asrefesq[0] * (dat.A22 + h[sli] / 2)

vshs = (As1[shs] + As2[shs]) * (L[1] + h[sls] / 2 + 4 * h[shs])
vsresqhs = Asrefesq[nb[0] + nb[1] - 1] * (dat.A23 + h[sls] / 2)

vsls = (As1[sls] + As2[sls]) * (L[2] + h[shs] + 4 * h[sls])
vsrls = Asref[sls] * dat.A30
vsresqls = Asrefesq[nb[0] + nb[1] + 1] * (dat.A33 + dat.A34 + h[shs])
vscls = 2 * (Asw[nb[0] + nb[1] + 1] * b * h[sls] * (dat.A26 + h[sls]) /
↪  sw[nb[0] + nb[1] + 1])
```

```
vsli = (As1[sli] + As2[sli]) * (L[5] + h[shs] + 4 * h[sli])
vsresqli = Asrefesq[0] * (dat.A35 + dat.A32 + h[shs])
vscli = 2 * (Asw[0] * b * h[sli] * (dat.A28 + h[sli]) / sw[0])

Astr1H = (0.0032 * L[0] * h[0]) * 0.40
Astr2H = (0.0032 * L[0] * h[0]) * 0.60
AstrHastiales = Astr1H + Astr2H

seccl = nb[0] + nb[1] + int(nb[2] / 2)
Astr1L = 0.20 * (As1[seccl] + Asref[seccl]) * L[2]
Astr2L = 0.20 * (As1[seccl] + Asref[seccl]) * L[2]
AstrLosas = Astr1L + Astr2L

steelT = 2 * (AstrLosas + AstrHastiales)

steelH = 2 * (vshi + vsresqhi + vshs + vsresqhs)
steelLS = vsls + vsrls + vsresqls + vscls
steelLI = vsli + vsresqli + vscli

steelV = steelH + steelLS + steelLI + steelT
steel = steelV * dat.sdensity

for i in range(0, len(datosaceros)):
    aux = datosaceros[i, 0]
    if aux == fyk:
        for j in range(0, len(costetotal)):
            costetotal[1, j] = steel * datosaceros[i, j + 1]


for i in range(0, len(costetotal[0])):
    costetotal[2, i] = sum(costetotal[:, i])

mediciones = np.zeros(2)

mediciones[0] = concrete
mediciones[1] = steel

return costetotal, mediciones
```

A.8.2. Algoritmo de recocido simulado SA

```
def cambia_variables(vector_variables):
    ################################ VARIABLES de la SECCIÓN TRANSVERSAL
    ↪  ##############################################
    dat.hh = vector_variables[0]
    dat.hls = vector_variables[1]
    dat.hli = vector_variables[2]

    ################################ VARIABLES del ARMADO del MARCO
    ↪  ######################################################
    # Armado BASE del HASTIAL IZQUIERDO
    # Parte INFERIOR
    dat.fiA1 = vector_variables[3]
    dat.nA1 = vector_variables[4]
    dat.fiA2 = vector_variables[5]
    dat.nA2 = vector_variables[6]
    # Parte SUPERIOR
    dat.fiA3 = vector_variables[7]
    dat.nA3 = vector_variables[8]
    dat.fiA4 = vector_variables[9]
    dat.nA4 = vector_variables[10]

    # Armado BASE y REFUERZO de la LOSA SUPERIOR
    dat.fiA5 = vector_variables[11]
    dat.nA5 = vector_variables[12]
    dat.fiA6 = vector_variables[13]
    dat.nA6 = vector_variables[14]
    dat.fiA7 = vector_variables[15]
    dat.nA7 = vector_variables[16]

    # Armado BASE de la LOSA INFERIOR
    dat.fiA12 = vector_variables[17]
    dat.nA12 = vector_variables[18]
    dat.fiA13 = vector_variables[19]
    dat.nA13 = vector_variables[20]

    # Armado de REFUERZO en las ESQUINAS
    # Esquina INFERIOR IZQUIERDA
    dat.fiA14 = vector_variables[21]
```

```
    dat.nA14 = vector_variables[22]
    # Esquina SUPERIOR IZQUIERDA
    dat.fiA15 = vector_variables[23]
    dat.nA15 = vector_variables[24]

    # Armado de CORTANTE en las LOSAS
    dat.fiA18 = vector_variables[25]
    dat.sA18 = vector_variables[26]
    dat.fiA20 = vector_variables[27]
    dat.sA20 = vector_variables[28]

    ################################ VARIABLES de los MATERIALES del MARCO
    ↪  ############################################

    dat.fck = vector_variables[29]
    dat.fyk = vector_variables[30]


def movimiento_sa(variables_marco, sinicial, desv_estandar, numero_variables):
    sol1 = np.array(sinicial)
    control_cambios = np.zeros((len(sol1), 1))
    numero_variables = round(numero_variables * (1 + desv_estandar / 100 *
    ↪  np.random.rand()))

    i = 0

    while i < numero_variables:
        j = np.random.randint(0, len(sol1) - 1)
        if control_cambios[j] == 0:
            if np.random.rand() < 0.5:  # Valor aleatorio distribución uniforme
            ↪  entre 0 y 1
                control_cambios[j] = 1
            else:
                control_cambios[j] = - 1
        i = i + 1

    for i in range(0, len(sol1)):
        for j in range(len(variables_marco[i])):
            if sol1[i] == variables_marco[i][j]:
```

```
                if control_cambios[i] == 1:
                    if j != (len(variables_marco[i]) - 1):
                        sol1[i] = variables_marco[i][j + 1]
                    else:
                        dice = np.random.choice((1, 2, 3))
                        if dice == 1:
                            sol1[i] = variables_marco[i][j]
                        elif dice == 2:
                            sol1[i] = variables_marco[i][0]
                        else:
                            sol1[i] = np.random.choice(variables_marco[i])
                elif control_cambios[i] == - 1:
                    if j != 0:
                        sol1[i] = variables_marco[i][j - 1]
                    else:
                        dice = np.random.choice((1, 2, 3))
                        if dice == 1:
                            sol1[i] = variables_marco[i][j]
                        elif dice == 2:
                            sol1[i] = variables_marco[i][j - 1]
                        else:
                            sol1[i] = np.random.choice(variables_marco[i])
                break

    return sol1


def solutionstorage(contador_cadenas, iteracion, coste, temperatura,
↪  time_iteration, time_elapsed, merco,
                    mediciones):
    data_aux = []
    data_i = []

    data_aux.extend([contador_cadenas, iteracion, coste, temperatura,
    ↪  time_iteration, time_elapsed, merco,
                     mediciones])

    counter = 0
    for i in range(len(data_aux)):
```

```
            if isinstance(data_aux[i], int) or isinstance(data_aux[i], float):
                data_i.append(data_aux[i])
                counter += 1
            else:
                for j in range(len(data_aux[i])):
                    data_i.append(data_aux[i][j])
                    counter += 1

        return data_i


    def sa(variables_marco, mh_variables, data_labels):
        # Inicialización de modelo del marco
        nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
        ↪  model.model(dat.L, dat.nb, dat.alpha,

                                                                                    ↪  da
        start_time = time.time()

        # Generación del vector de soluciones
        marcoinicial = np.zeros(31)

        # Identificación de las variables de la heurística
        cadena_markov = mh_variables[0]
        desv_estandar = mh_variables[1]
        numero_variables = mh_variables[2]
        coeficiente_enfriamiento = mh_variables[3]
        criterio_de_parada = mh_variables[4]
        criterio_terminacion = mh_variables[5]
        t0 = mh_variables[6]
        rango_inferior = mh_variables[7]
        rango_superior = mh_variables[8]

        # Generación del vector de datos
        data = np.zeros((0, len(data_labels)))
        data_best = np.zeros((0, len(data_labels)))

        factible = False
        factibles = 0
```

```
no_factibles = 0
iteracion = 1

##### GENERACIÓN DE LA SOLUCIÓN INICIAL (S0)
↪  ##############################################################################
tiempo_inicial = time.time()
while not factible:

    ##### Genera marco inicial aleatorio ##### (Marco Inicial random)
    for i in range(0, len(variables_marco)):
        marcoinicial[i] = np.random.choice(variables_marco[i])
    cambia_variables(marcoinicial)

    ##### Comprueba si el marco inicial es válido ##### (¿Marco Inicial =
    ↪  S0?)
    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF, thetas,
        ↪  lenghts, restrainedDoF, restrains)

    print('Comprobación del marco inicial:', '\n',
          'GEO', ncGEOM, '\n',
          'AsMin', ncAsMin, '\n',
          'AsMax', ncAsMax, '\n',
          'ELU(V)', ncVELU, '\n',
          'ELU(N)', ncNELU, '\n',
          'ELU(M)', ncMELU, '\n',
          'ELS', ncFELS, '\n')

    if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0 and
    ↪  ncNELU == 0 \
            and ncMELU == 0 and ncFELS == 0:
        factibles += 1
    else:
        no_factibles += 1

    ##### Solución inicial establecida ##### (S0)
    if factibles == 1:
        tiempo_final = time.time()
        tiempo_medio = (tiempo_final - tiempo_inicial) / iteracion
```

```
            factible = True

            print('\n*************** Solución aleatoria generada
            ↪  ***************')
            print('Nº de iteración:                              %s' % iteracion)
            print('Nº de soluciones factibles:                   %s' % factibles)
            print('Nº de soluciones no factibles:                %s' %
            ↪  no_factibles)
            print('Tiempo medio:                                 %2.2f' %
            ↪  tiempo_medio)

        iteracion += 1

    ##### ESTABLECIMIENTO DE LA TEMPERATURA INICIAL
    ↪  ##############################################################################
    if factible:

        ##### Evaluación de función objetivo de la solución inicial ##### f(S0)
        costetotal, mediciones = calcmodule.objectivefunction(nodes, members,
        ↪  lenghts, thetas)

        coste_S0 = costetotal[2]
        marco_S0 = marcoinicial
        mediciones_S0 = mediciones

        marco_r = marco_S0
        coste_r = coste_S0
        mediciones_r = mediciones_S0

        ##### Temperatura inicial arbitraria #####
        temperatura = t0 * coste_S0[0]

        check = True
        iteracion = 0
        contador_cadenas = 0

        ##### Adecuación de la  temperatura inicial ##### (Aceptación f(T) =
        ↪  (0.2, 0.4))
```

```
while check:
    movimiento = 0
    aceptadas_cadena = 0
    validas_cadena = 0
    empeora_S0 = 0
    aceptada_empeora = 0
    contador_cadenas += 1

    ##### Inicio de Cadena de Markov #####
    while movimiento < cadena_markov:
        iteracion += 1
        movimiento += 1
        time_0i = time.time()

        ##### Movimiento de S0 a S1 #####
        marco_S1 = movimiento_sa(variables_marco, marco_S0,
        ↪  desv_estandar, numero_variables)
        cambia_variables(marco_S1)

        ##### Evaluación de función objetivo de la solución S1 #####
        ↪  f(S1)
        costetotal, mediciones = calcmodule.objectivefunction(nodes,
        ↪  members, lenghts, thetas)
        coste_S1 = costetotal[2]

        ##### Criterio de aceptación Boltzmann #####
        A = np.random.rand()
        B = np.exp(-1 * (coste_S1[0] - coste_S0[0]) / temperatura)

        if B < 1:
            empeora_S0 += 1

        if A < B:
            if B < 1:
                aceptada_empeora += 1
            aceptadas_cadena += 1
            ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
                calcmodule.verification(nodes, members, pins, pinDoF,
                ↪  thetas, lenghts, restrainedDoF, restrains)
```

```
            ##### Verificación de la solución propuesta S1 #####
            if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU
            ↪  == 0 and ncNELU == 0 \
                    and ncMELU == 0 and ncFELS == 0:
                factibles += 1
                validas_cadena += 1

                ##### Aceptación de la solución S1 como nueva S0 #####
                ↪  (S0 = S1)
                marco_S0 = marco_S1
                coste_S0 = coste_S1
                mediciones_S0 = mediciones

                ##### Comprobación marco récord #####
                if coste_S1[0] < coste_r[0]:
                    coste_r = coste_S1
                    marco_r = marco_S1
                    mediciones_r = mediciones

            else:
                no_factibles += 1

        if movimiento % 100 == 0:
            perc_factible = validas_cadena / movimiento * 100
            perc_boltzmann = (aceptada_empeora / empeora_S0) * 100
            print('\n********** Definición de la temperatura inicial
            ↪  **********')
            print('Temperatura:
            ↪  %2.2f' % temperatura)
            print('Cadena de markov nº:                                %s' %
            ↪  contador_cadenas)
            print('Nº de movimientos:                                  %s' %
            ↪  movimiento)
            print('Nº de aceptadas en la cadena:                       %s' %
            ↪  aceptadas_cadena)
            print('Nº de movimientos factibles:                        %s' %
            ↪  validas_cadena)
```

```
                print('Porcentaje factibles:
                ↪  %2.2f' % perc_factible + str('%'))
                print('Porcentaje Boltzmann:
                ↪  %2.2f' % perc_boltzmann + str('%'))
                print('Coste actual f(S0):
                ↪  %2.2f' % coste_S0[0])
                print('Coste record f(Sr):
                ↪  %2.2f' % coste_r[0])

            time_0it = time.time() - time_0i
            time_0el = time.time() - start_time

            """ALMACENAMIENTO SOLUCION"""
            ##### Solución récord ##### (Sr)
            data_i_best = solutionstorage(contador_cadenas, iteracion,
            ↪  coste_r, temperatura, time_0it, time_0el,
                                          marco_r, mediciones_r)
            data_best = np.append(data_best, [data_i_best], axis=0)
            data_i_best.clear()

            ##### Solución actual ##### (S0)
            data_i = solutionstorage(contador_cadenas, iteracion, coste_S0,
            ↪  temperatura, time_0it, time_0el,
                                     marco_S0, mediciones_S0)
            data = np.append(data, [data_i], axis=0)
            data_i.clear()

        iteracion += 1

        ##### Condiciones establecimiento de temperatura inicial #####
        if (aceptada_empeora / empeora_S0) < rango_inferior:
            temperatura = 2 * temperatura
        else:
            if (aceptada_empeora / empeora_S0) > rango_superior:
                temperatura = 0.5 * temperatura
            else:
                check = False

temperatura_inicial = temperatura
```

```
print('\nTemperatura inicial fijada en ', temperatura_inicial, 'ºC')

##### ENFRIAMIENTO DEL PROBLEMA
↪  ##########################################################################################
cadenas_sin_mejora = 0
# contador_cadenas = 0
# iteracion = 0
check = True

while check:
    movimiento = 0
    movimiento_sin_mejora = 0
    validas_cadena = 0
    contador_cadenas += 1

    while movimiento < cadena_markov:
        iteracion += 1
        movimiento += 1
        i_time = time.time()

        ##### Movimiento de S0 a S1 #####
        marco_S1 = movimiento_sa(variables_marco, marco_S0, desv_estandar,
        ↪  numero_variables)
        cambia_variables(marco_S1)

        ##### Evaluación de función objetivo de la solución S1 ##### f(S1)
        costetotal, mediciones = calcmodule.objectivefunction(nodes, members,
        ↪  lenghts, thetas)
        coste_S1 = costetotal[2]

        ##### Criterio de aceptación Boltzmann #####
        A = np.random.rand()
        B = np.exp(-1 * (coste_S1[0] - coste_S0[0]) / temperatura)

        if A < B:
            check = True
            ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
                calcmodule.verification(nodes, members, pins, pinDoF, thetas,
                ↪  lenghts, restrainedDoF, restrains)
```

```
            ##### Verificación de la solución propuesta S1 #####
            if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0
            ↪  and ncNELU == 0 \
                    and ncMELU == 0 and ncFELS == 0:
                factibles += 1
                validas_cadena += 1

                ##### Aceptación de la solución S1 como nueva S0 ##### (S0 =
                ↪  S1)
                marco_S0 = marco_S1
                coste_S0 = coste_S1
                mediciones_S0 = mediciones

                ##### Comprobación marco récord #####
                if coste_S1[0] < coste_r[0]:
                    coste_r = coste_S1
                    marco_r = marco_S1
                    mediciones_r = mediciones

                else:
                    movimiento_sin_mejora += 1

            else:
                no_factibles += 1

        if movimiento % 100 == 0:
            perc_factible = validas_cadena / movimiento * 100
            print('\n***************** Enfriamiento del problema
            ↪  *****************')
            print('Temperatura:                                    %2.2f' %
            ↪  temperatura)
            print('Cadena de markov nº:                            %s' %
            ↪  contador_cadenas)
            print('Nº de movimientos:                              %s' %
            ↪  movimiento)
            print('Nº de movimientos aceptados:                    %s' %
            ↪  validas_cadena)
```

```
            print('Porcentaje factibles:                                %2.2f' %
            ↪  perc_factible + str('%'))
            print('Coste actual f(S0):                                  %2.2f' %
            ↪  coste_S0[0])
            print('Coste record f(Sr):                                  %2.2f' %
            ↪  coste_r[0])

        it_time = time.time() - i_time
        el_time = time.time() - start_time

        """ALMACENAMIENTO SOLUCION"""
        ##### Solución récord ##### (Sr)
        data_i_best = solutionstorage(contador_cadenas, iteracion, coste_r,
        ↪  temperatura, it_time, el_time,
                                      marco_r, mediciones_r)
        data_best = np.append(data_best, [data_i_best], axis=0)
        data_i_best.clear()

        ##### Solución actual ##### (S0)
        data_i = solutionstorage(contador_cadenas, iteracion, coste_S0,
        ↪  temperatura, it_time, el_time,
                                 marco_S0, mediciones_S0)
        data = np.append(data, [data_i], axis=0)
        data_i.clear()

    ##### Reducción temperatura para siguiente Cadena de Markov ##### (T =
    ↪  k*T)
    temperatura = coeficiente_enfriamiento * temperatura

    ##### Verificación criterios de parada del problema #####
    if movimiento_sin_mejora == cadena_markov:
        cadenas_sin_mejora += 1
    else:
        cadenas_sin_mejora = 0

    if cadenas_sin_mejora >= criterio_de_parada or temperatura /
    ↪  temperatura_inicial < criterio_terminacion:
        check = False
```

```
    return data, data_best
```

A.8.3. Algoritmo de aceptación por umbrales TA

```
import numpy as np
import time

import data as dat
import modelgenerator as model
import maincalculationsmodule as calcmodule

def cambia_variables(vector_variables):
    ################################ VARIABLES de la SECCIÓN TRANSVERSAL
    ↪  ################################################
    dat.hh = vector_variables[0]
    dat.hls = vector_variables[1]
    dat.hli = vector_variables[2]

    ################################ VARIABLES del ARMADO del MARCO
    ↪  ########################################################
    # Armado BASE del HASTIAL IZQUIERDO
    # Parte INFERIOR
    dat.fiA1 = vector_variables[3]
    dat.nA1 = vector_variables[4]
    dat.fiA2 = vector_variables[5]
    dat.nA2 = vector_variables[6]
    # Parte SUPERIOR
    dat.fiA3 = vector_variables[7]
    dat.nA3 = vector_variables[8]
    dat.fiA4 = vector_variables[9]
    dat.nA4 = vector_variables[10]

    # Armado BASE y REFUERZO de la LOSA SUPERIOR
    dat.fiA5 = vector_variables[11]
    dat.nA5 = vector_variables[12]
    dat.fiA6 = vector_variables[13]
    dat.nA6 = vector_variables[14]
    dat.fiA7 = vector_variables[15]
    dat.nA7 = vector_variables[16]
```

```
    # Armado BASE de la LOSA INFERIOR
    dat.fiA12 = vector_variables[17]
    dat.nA12 = vector_variables[18]
    dat.fiA13 = vector_variables[19]
    dat.nA13 = vector_variables[20]

    # Armado de REFUERZO en las ESQUINAS
    # Esquina INFERIOR IZQUIERDA
    dat.fiA14 = vector_variables[21]
    dat.nA14 = vector_variables[22]
    # Esquina SUPERIOR IZQUIERDA
    dat.fiA15 = vector_variables[23]
    dat.nA15 = vector_variables[24]

    # Armado de CORTANTE en las LOSAS
    dat.fiA18 = vector_variables[25]
    dat.sA18 = vector_variables[26]
    dat.fiA20 = vector_variables[27]
    dat.sA20 = vector_variables[28]

    ################################# VARIABLES de los MATERIALES del MARCO
    ↪  ##############################################

    dat.fck = vector_variables[29]
    dat.fyk = vector_variables[30]


def movimiento_ta(variables_marco, sinicial, desv_estandar, numero_variables):
    sol1 = np.array(sinicial)
    control_cambios = np.zeros((len(sol1), 1))
    numero_variables = round(numero_variables * (1 + desv_estandar / 100 *
    ↪  np.random.rand()))

    i = 0

    while i < numero_variables:
        j = np.random.randint(0, len(sol1) - 1)
        if control_cambios[j] == 0:
```

```
            if np.random.rand() < 0.5:  # Valor aleatorio distribución uniforme
            ↪ entre 0 y 1
                control_cambios[j] = 1
            else:
                control_cambios[j] = - 1
        i = i + 1

    for i in range(0, len(sol1)):
        for j in range(len(variables_marco[i])):
            if sol1[i] == variables_marco[i][j]:
                if control_cambios[i] == 1:
                    if j != (len(variables_marco[i]) - 1):
                        sol1[i] = variables_marco[i][j + 1]
                    else:
                        dice = np.random.choice((1, 2, 3))
                        if dice == 1:
                            sol1[i] = variables_marco[i][j]
                        elif dice == 2:
                            sol1[i] = variables_marco[i][0]
                        else:
                            sol1[i] = np.random.choice(variables_marco[i])
                elif control_cambios[i] == - 1:
                    if j != 0:
                        sol1[i] = variables_marco[i][j - 1]
                    else:
                        dice = np.random.choice((1, 2, 3))
                        if dice == 1:
                            sol1[i] = variables_marco[i][j]
                        elif dice == 2:
                            sol1[i] = variables_marco[i][j - 1]
                        else:
                            sol1[i] = np.random.choice(variables_marco[i])
                break

    return sol1


def solutionstorage(contador_cadenas, iteracion, coste, temperatura,
↪ time_iteration, time_elapsed, merco,
```

```
                mediciones):
    data_aux = []
    data_i = []

    data_aux.extend([contador_cadenas, iteracion, coste, temperatura,
    ↪  time_iteration, time_elapsed, merco,
                mediciones])

    counter = 0
    for i in range(len(data_aux)):
        if isinstance(data_aux[i], int) or isinstance(data_aux[i], float):
            data_i.append(data_aux[i])
            counter += 1
        else:
            for j in range(len(data_aux[i])):
                data_i.append(data_aux[i][j])
                counter += 1

    return data_i


def ta(variables_marco, mh_variables, data_labels):
    # Inicialización de modelo del marco
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha,

                                                                    ↪  da
    start_time = time.time()

    # Generación del vector de soluciones
    marcoinicial = np.zeros(31)

    # Identificación de las variables de la heurística
    cadena_umbral = mh_variables[0]
    desv_estandar = mh_variables[1]
    numero_variables = mh_variables[2]
    coeficiente_reduccion = mh_variables[3]
    criterio_de_parada = mh_variables[4]
    criterio_terminacion = mh_variables[5]
```

```
u0 = mh_variables[6]
rango_inferior = mh_variables[7]
rango_superior = mh_variables[8]

# Generación del vector de datos
data = np.zeros((0, len(data_labels)))
data_best = np.zeros((0, len(data_labels)))

factible = False
factibles = 0
no_factibles = 0
iteracion = 1

##### GENERACIÓN DE LA SOLUCIÓN INICIAL (SO)
↪  ##############################################################################
tiempo_inicial = time.time()
while not factible:

    ##### Genera marco inicial aleatorio ##### (Marco Inicial random)
    for i in range(0, len(variables_marco)):
        marcoinicial[i] = np.random.choice(variables_marco[i])
    cambia_variables(marcoinicial)

    ##### Comprueba si el marco inicial es válido ##### (¿Marco Inicial =
    ↪  SO?)
    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF, thetas,
        ↪  lenghts, restrainedDoF, restrains)

    print('Comprobación del marco inicial:', '\n',
          'GEO', ncGEOM, '\n',
          'AsMin', ncAsMin, '\n',
          'AsMax', ncAsMax, '\n',
          'ELU(V)', ncVELU, '\n',
          'ELU(N)', ncNELU, '\n',
          'ELU(M)', ncMELU, '\n',
          'ELS', ncFELS, '\n')
```

```
    if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0 and
    ↪  ncNELU == 0 \
            and ncMELU == 0 and ncFELS == 0:
        factibles += 1
    else:
        no_factibles += 1

    ##### Solución inicial establecida ##### (S0)
    if factibles == 1:
        tiempo_final = time.time()
        tiempo_medio = (tiempo_final - tiempo_inicial) / iteracion

        factible = True

        print('\n*************** Solución aleatoria generada
        ↪  ***************')
        print('Nº de iteración:                              %s' % iteracion)
        print('Nº de soluciones factibles:                   %s' % factibles)
        print('Nº de soluciones no factibles:                %s' %
        ↪  no_factibles)
        print('Tiempo medio:                                 %2.2f' %
        ↪  tiempo_medio)

    iteracion += 1

##### ESTABLECIMIENTO DEL UMBRAL INICIAL
↪  ###############################################################################
if factible:

    ##### Evaluación de función objetivo de la solución inicial ##### f(S0)
    costetotal, mediciones = calcmodule.objectivefunction(nodes, members,
    ↪  lenghts, thetas)

    coste_S0 = costetotal[2]
    marco_S0 = marcoinicial
    mediciones_S0 = mediciones

    marco_r = marco_S0
    coste_r = coste_S0
```

```
mediciones_r = mediciones_S0

##### Umbral inicial arbitrario #####
umbral = u0 * coste_S0[0]
print(umbral)

check = True
iteracion = 0
contador_cadenas = 0

##### Adecuación del umbral inicial ##### (Aceptación f(U) = (0.2, 0.4))
while check:
    movimiento = 0
    aceptadas_cadena = 0
    validas_cadena = 0
    empeora_S0 = 0
    aceptada_empeora = 0
    contador_cadenas += 1

    ##### Inicio de Cadena de Umbral #####
    while movimiento < cadena_umbral:
        iteracion += 1
        movimiento += 1

        time_0i = time.time()

        ##### Movimiento de S0 a S1 #####
        marco_S1 = movimiento_ta(variables_marco, marco_S0,
        ↪  desv_estandar, numero_variables)
        cambia_variables(marco_S1)

        ##### Evaluación de función objetivo de la solución S1 #####
        ↪  f(S1)
        costetotal, mediciones = calcmodule.objectivefunction(nodes,
        ↪  members, lenghts, thetas)
        coste_S1 = costetotal[2]

        ##### Criterio determinista Umbral #####
        D = np.abs(coste_S1[0] - coste_S0[0])
```

```
if coste_S1[0] > coste_S0[0]:
    empeora_S0 += 1

if coste_S1[0] <= coste_S0[0] or D < umbral:
    if coste_S1[0] > coste_S0[0]:
        aceptada_empeora += 1
    aceptadas_cadena += 1
    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF,
        ↪  thetas, lenghts, restrainedDoF, restrains)

    ##### Verificación de la solución propuesta S1 #####
    if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU
    ↪  == 0 and ncNELU == 0 \
            and ncMELU == 0 and ncFELS == 0:
        factibles += 1
        validas_cadena += 1

        ##### Aceptación de la solución S1 como nueva S0 #####
        ↪  (S0 = S1)
        marco_S0 = marco_S1
        coste_S0 = coste_S1
        mediciones_S0 = mediciones

        ##### Comprobación marco récord #####
        if coste_S1[0] < coste_r[0]:
            coste_r = coste_S1
            marco_r = marco_S1
            mediciones_r = mediciones

    else:
        no_factibles += 1

if movimiento % 100 == 0:
    perc_factible = validas_cadena / movimiento * 100
    perc_boltzmann = (aceptada_empeora / empeora_S0) * 100
    print('\n*************** Definición del umbral inicial
    ↪  ***************')
```

```
            print('Umbral:
            ↪  %2.2f' % umbral)
            print('Cadena de Umbral nº:                                %s' %
            ↪  contador_cadenas)
            print('Nº de movimientos:                                  %s' %
            ↪  movimiento)
            print('Nº de aceptadas en la cadena:                       %s' %
            ↪  aceptadas_cadena)
            print('Nº de movimientos factibles:                        %s' %
            ↪  validas_cadena)
            print('Porcentaje factibles:
            ↪  %2.2f' % perc_factible + str('%'))
            print('Porcentaje Umbral:
            ↪  %2.2f' % perc_boltzmann + str('%'))
            print('Coste actual f(S0):
            ↪  %2.2f' % coste_S0[0])
            print('Coste record f(Sr):
            ↪  %2.2f' % coste_r[0])

        time_0it = time.time() - time_0i
        time_0el = time.time() - start_time

        """ALMACENAMIENTO SOLUCION"""
        ##### Solución récord ##### (Sr)
        data_i_best = solutionstorage(contador_cadenas, iteracion,
        ↪  coste_r, umbral, time_0it, time_0el,
                                      marco_r, mediciones_r)
        data_best = np.append(data_best, [data_i_best], axis=0)
        data_i_best.clear()

        ##### Solución actual ##### (S0)
        data_i = solutionstorage(contador_cadenas, iteracion, coste_S0,
        ↪  umbral, time_0it, time_0el,
                                 marco_S0, mediciones_S0)
        data = np.append(data, [data_i], axis=0)
        data_i.clear()

    iteracion += 1
```

```
        ##### Condiciones establecimiento del umbral inicial #####
        if empeora_S0 == 0:
            umbral = 0.5 * umbral
        else:
            if (aceptada_empeora / empeora_S0) < rango_inferior:
                umbral = 2 * umbral
            else:
                if (aceptada_empeora / empeora_S0) > rango_superior:
                    umbral = 0.5 * umbral
                else:
                    check = False

umbral_inicial = umbral
print('\nUmbral inicial fijado en ', umbral_inicial, '€')

##### ENFRIAMIENTO DEL PROBLEMA
↪  ###########################################################################################
cadenas_sin_mejora = 0
# contador_cadenas = 0
# iteracion = 0
check = True

while check:
    movimiento = 0
    movimiento_sin_mejora = 0
    validas_cadena = 0
    contador_cadenas += 1

    while movimiento < cadena_umbral:
        iteracion += 1
        movimiento += 1
        i_time = time.time()

        ##### Movimiento de S0 a S1 #####
        marco_S1 = movimiento_ta(variables_marco, marco_S0, desv_estandar,
        ↪  numero_variables)
        cambia_variables(marco_S1)

        ##### Evaluación de función objetivo de la solución S1 ##### f(S1)
```

```
costetotal, mediciones = calcmodule.objectivefunction(nodes, members,
↪  lenghts, thetas)
coste_S1 = costetotal[2]

##### Criterio determinista Umbral #####
D = np.abs(coste_S1[0] - coste_S0[0])

if coste_S1[0] <= coste_S0[0] or D < umbral:
    check = True
    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF, thetas,
        ↪  lenghts, restrainedDoF, restrains)

    ##### Verificación de la solución propuesta S1 #####
    if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0
    ↪  and ncNELU == 0 \
            and ncMELU == 0 and ncFELS == 0:
        factibles += 1
        validas_cadena += 1

        ##### Aceptación de la solución S1 como nueva S0 ##### (S0 =
        ↪  S1)
        marco_S0 = marco_S1
        coste_S0 = coste_S1
        mediciones_S0 = mediciones

        ##### Comprobación marco récord #####
        if coste_S1[0] < coste_r[0]:
            coste_r = coste_S1
            marco_r = marco_S1
            mediciones_r = mediciones

    else:
        no_factibles += 1

if movimiento % 100 == 0:
    perc_factible = validas_cadena / movimiento * 100
    print('\n********** Enfriamiento del problema
    ↪  **********')
```

```
            print('Umbral:                                          %2.2f' %
            ↪  umbral)
            print('Cadena de Umbral nº:                             %s' %
            ↪  contador_cadenas)
            print('Nº de movimientos:                               %s' %
            ↪  movimiento)
            print('Nº de movimientos aceptados:                     %s' %
            ↪  validas_cadena)
            print('Porcentaje factibles:                            %2.2f' %
            ↪  perc_factible + str('%'))
            print('Coste actual f(S0):                              %2.2f' %
            ↪  coste_S0[0])
            print('Coste record f(Sr):                              %2.2f' %
            ↪  coste_r[0])

        it_time = time.time() - i_time
        el_time = time.time() - start_time

        """ALMACENAMIENTO SOLUCION"""
        ##### Solución récord ##### (Sr)
        data_i_best = solutionstorage(contador_cadenas, iteracion, coste_r,
        ↪  umbral, it_time, el_time,
                                      marco_r, mediciones_r)
        data_best = np.append(data_best, [data_i_best], axis=0)
        data_i_best.clear()

        ##### Solución actual ##### (S0)
        data_i = solutionstorage(contador_cadenas, iteracion, coste_S0,
        ↪  umbral, it_time, el_time,
                                 marco_S0, mediciones_S0)
        data = np.append(data, [data_i], axis=0)
        data_i.clear()

    ##### Reducción umbral para siguiente Cadena de Umbral ##### (T = k*T)
    umbral = coeficiente_reduccion * umbral

    ##### Verificación criterios de parada del problema #####
    if movimiento_sin_mejora == cadena_umbral:
        cadenas_sin_mejora += 1
```

```
        else:
            cadenas_sin_mejora = 0

        if cadenas_sin_mejora >= criterio_de_parada or umbral / umbral_inicial <
        ↪  criterio_terminacion:
            check = False

    return data, data_best
```

A.8.4. Algoritmo del solterón OBA

```
import numpy as np
import time

import data as dat
import modelgenerator as model
import maincalculationsmodule as calcmodule

def cambia_variables(vector_variables):
    ################################ VARIABLES de la SECCIÓN TRANSVERSAL
    ↪  ################################################
    dat.hh = vector_variables[0]
    dat.hls = vector_variables[1]
    dat.hli = vector_variables[2]

    ################################ VARIABLES del ARMADO del MARCO
    ↪  ########################################################
    # Armado BASE del HASTIAL IZQUIERDO
    # Parte INFERIOR
    dat.fiA1 = vector_variables[3]
    dat.nA1 = vector_variables[4]
    dat.fiA2 = vector_variables[5]
    dat.nA2 = vector_variables[6]
    # Parte SUPERIOR
    dat.fiA3 = vector_variables[7]
    dat.nA3 = vector_variables[8]
    dat.fiA4 = vector_variables[9]
    dat.nA4 = vector_variables[10]
```

```
    # Armado BASE y REFUERZO de la LOSA SUPERIOR
    dat.fiA5 = vector_variables[11]
    dat.nA5 = vector_variables[12]
    dat.fiA6 = vector_variables[13]
    dat.nA6 = vector_variables[14]
    dat.fiA7 = vector_variables[15]
    dat.nA7 = vector_variables[16]

    # Armado BASE de la LOSA INFERIOR
    dat.fiA12 = vector_variables[17]
    dat.nA12 = vector_variables[18]
    dat.fiA13 = vector_variables[19]
    dat.nA13 = vector_variables[20]

    # Armado de REFUERZO en las ESQUINAS
    # Esquina INFERIOR IZQUIERDA
    dat.fiA14 = vector_variables[21]
    dat.nA14 = vector_variables[22]
    # Esquina SUPERIOR IZQUIERDA
    dat.fiA15 = vector_variables[23]
    dat.nA15 = vector_variables[24]

    # Armado de CORTANTE en las LOSAS
    dat.fiA18 = vector_variables[25]
    dat.sA18 = vector_variables[26]
    dat.fiA20 = vector_variables[27]
    dat.sA20 = vector_variables[28]

    ################################## VARIABLES de los MATERIALES del MARCO
    ↪  ##############################################

    dat.fck = vector_variables[29]
    dat.fyk = vector_variables[30]


def movimiento_sa(variables_marco, sinicial, desv_estandar, numero_variables):
    sol1 = np.array(sinicial)
    control_cambios = np.zeros((len(sol1), 1))
```

```
numero_variables = round(numero_variables * (1 + desv_estandar / 100 *
↪  np.random.rand()))

i = 0

while i < numero_variables:
    j = np.random.randint(0, len(sol1) - 1)
    if control_cambios[j] == 0:
        if np.random.rand() < 0.5:  # Valor aleatorio distribución uniforme
        ↪  entre 0 y 1
            control_cambios[j] = 1
        else:
            control_cambios[j] = - 1
    i = i + 1

for i in range(0, len(sol1)):
    for j in range(len(variables_marco[i])):
        if sol1[i] == variables_marco[i][j]:
            if control_cambios[i] == 1:
                if j != (len(variables_marco[i]) - 1):
                    sol1[i] = variables_marco[i][j + 1]
                else:
                    dice = np.random.choice((1, 2, 3))
                    if dice == 1:
                        sol1[i] = variables_marco[i][j]
                    elif dice == 2:
                        sol1[i] = variables_marco[i][0]
                    else:
                        sol1[i] = np.random.choice(variables_marco[i])
            elif control_cambios[i] == - 1:
                if j != 0:
                    sol1[i] = variables_marco[i][j - 1]
                else:
                    dice = np.random.choice((1, 2, 3))
                    if dice == 1:
                        sol1[i] = variables_marco[i][j]
                    elif dice == 2:
                        sol1[i] = variables_marco[i][j - 1]
                    else:
```

```
                            sol1[i] = np.random.choice(variables_marco[i])
                break

    return sol1


def solutionstorage(iteracion, coste, temperatura, time_iteration, time_elapsed,
↪  merco,
                    mediciones):
    data_aux = []
    data_i = []

    data_aux.extend([iteracion, coste, temperatura, time_iteration, time_elapsed,
    ↪  merco,
                     mediciones])

    counter = 0
    for i in range(len(data_aux)):
        if isinstance(data_aux[i], int) or isinstance(data_aux[i], float):
            data_i.append(data_aux[i])
            counter += 1
        else:
            for j in range(len(data_aux[i])):
                data_i.append(data_aux[i][j])
                counter += 1

    return data_i


def oba(variables_marco, mh_variables, data_labels):
    # Inicialización de modelo del marco
    nodes, members, pins, pinDoF, thetas, lenghts, restrainedDoF, restrains =
    ↪  model.model(dat.L, dat.nb, dat.alpha,

                                                                              ↪  da
    start_time = time.time()

    # Generación del vector de soluciones
    marcoinicial = np.zeros(31)
```

```
# Identificación de las variables de la heurística
desv_estandar = mh_variables[0]
numero_variables = mh_variables[1]
coeficiente_variacion = mh_variables[2]
criterio_terminacion = mh_variables[3]

# Generación del vector de datos
data = np.zeros((0, len(data_labels)))
data_best = np.zeros((0, len(data_labels)))

factible = False
factibles = 0
no_factibles = 0
iteracion = 1

##### GENERACIÓN DE LA SOLUCIÓN INICIAL (SO)
↪  ##############################################################################
tiempo_inicial = time.time()
while not factible:

    ##### Genera marco inicial aleatorio ##### (Marco Inicial random)
    for i in range(0, len(variables_marco)):
        marcoinicial[i] = np.random.choice(variables_marco[i])
    cambia_variables(marcoinicial)

    ##### Comprueba si el marco inicial es válido ##### (¿Marco Inicial =
    ↪  SO?)
    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF, thetas,
        ↪  lenghts, restrainedDoF, restrains)

    print('Comprobación del marco inicial:', '\n',
          'GEO', ncGEOM, '\n',
          'AsMin', ncAsMin, '\n',
          'AsMax', ncAsMax, '\n',
          'ELU(V)', ncVELU, '\n',
          'ELU(N)', ncNELU, '\n',
          'ELU(M)', ncMELU, '\n',
```

```
            'ELS', ncFELS, '\n')

        if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0 and
        ↪  ncNELU == 0 \
                and ncMELU == 0 and ncFELS == 0:
            factibles += 1
        else:
            no_factibles += 1

        ##### Solución inicial establecida ##### (S0)
        if factibles == 1:
            tiempo_final = time.time()
            tiempo_medio = (tiempo_final - tiempo_inicial) / iteracion

            factible = True

            print('\n**************** Solución aleatoria generada
            ↪  ***************')
            print('Nº de iteración:                              %s' % iteracion)
            print('Nº de soluciones factibles:                   %s' % factibles)
            print('Nº de soluciones no factibles:                %s' %
            ↪  no_factibles)
            print('Tiempo medio:                                 %2.2f' %
            ↪  tiempo_medio)

        iteracion += 1

    if factible:

        ##### Evaluación de función objetivo de la solución inicial ##### f(S0)
        costetotal, mediciones = calcmodule.objectivefunction(nodes, members,
        ↪  lenghts, thetas)

        coste_S0 = costetotal[2]
        marco_S0 = marcoinicial
        mediciones_S0 = mediciones

        marco_r = marco_S0
        coste_r = coste_S0
```

```
mediciones_r = mediciones_S0

check = True
iteracion = 0
movimiento = 0
aceptados = 0
rechazados = 0

while check:
    iteracion += 1
    i_time = time.time()

    if iteracion <= criterio_terminacion:
        check = True
    elif iteracion > criterio_terminacion:
        check = False

    if iteracion == 1:
        umbral = 0

    ##### Movimiento de S0 a S1 #####
    marco_S1 = movimiento_sa(variables_marco, marco_S0, desv_estandar,
    ↪  numero_variables)
    cambia_variables(marco_S1)

    ncGEOM, ncAsMin, ncAsMax, ncVELU, ncNELU, ncMELU, ncFELS = \
        calcmodule.verification(nodes, members, pins, pinDoF, thetas,
        ↪  lenghts, restrainedDoF, restrains)

    ##### Verificación de la solución propuesta S1 #####
    if ncGEOM == 0 and ncAsMin == 0 and ncAsMax == 0 and ncVELU == 0 and
    ↪  ncNELU == 0 \
            and ncMELU == 0 and ncFELS == 0:
        movimiento += 1

        ##### Evaluación de función objetivo de la solución S1 #####
        ↪  f(S1)
        costetotal, mediciones = calcmodule.objectivefunction(nodes,
        ↪  members, lenghts, thetas)
```

```
            coste_S1 = costetotal[2]

            ##### Criterio de aceptación OBA #####
            D = coste_S1[0] - coste_S0[0]

            if D < 0 or np.abs(D) < umbral:
                aceptados += 1

                ##### Aceptación de la solución S1 como nueva S0 ##### (S0 =
                ↪  S1)
                marco_S0 = marco_S1
                coste_S0 = coste_S1
                mediciones_S0 = mediciones

                ##### Comprobación marco récord #####
                if coste_S1[0] < coste_r[0]:
                    coste_r = coste_S1
                    marco_r = marco_S1
                    mediciones_r = mediciones

                ##### Modificación del umbral de aceptación ##### (REDUCCIÓN)
                umbral = umbral - 0.10 * coeficiente_variacion

            else:
                rechazados += 1
                ##### Modificación del umbral de aceptación ##### (AUMENTO)
                umbral = umbral + 0.02 * coeficiente_variacion

        else:
            no_factibles += 1

        if iteracion % 100 == 0:
            perc_factible = movimiento / iteracion * 100
            print('\n********** Proceso Old Bachelor Acceptance
            ↪  **********')
            print('Umbral actual:                                    %2.2f' %
            ↪  umbral)
            print('Nº de iteraciones:                                %s' %
            ↪  iteracion)
```

```
                print('Nº de movimientos:                                %s' %
                ↪  movimiento)
                print('Nº de movimientos no factibles:                   %s' %
                ↪  rechazados)
                print('Porcentaje factibles:                             %2.2f' %
                ↪  perc_factible + str('%'))
                print('Coste actual f(S0):                               %2.2f' %
                ↪  coste_S0[0])
                print('Coste record f(Sr):                               %2.2f' %
                ↪  coste_r[0])

            it_time = time.time() - i_time
            el_time = time.time() - start_time

            """ALMACENAMIENTO SOLUCION"""
            ##### Solución récord ##### (Sr)
            data_i_best = solutionstorage(iteracion, coste_r, umbral, it_time,
            ↪  el_time,
                                          marco_r, mediciones_r)
            data_best = np.append(data_best, [data_i_best], axis=0)
            data_i_best.clear()

            ##### Solución actual ##### (S0)
            data_i = solutionstorage(iteracion, coste_S0, umbral, it_time,
            ↪  el_time,
                                     marco_S0, mediciones_S0)
            data = np.append(data, [data_i], axis=0)
            data_i.clear()

    return data, data_best
```