Optimización estructural y topológica de estructuras morfológicamente no definidas mediante algoritmos genéticos

SAMUEL SÁNCHEZ CABALLERO

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Ingeniería Mecánica y de Materiales

OPTIMIZACIÓN ESTRUCTURAL Y TOPOLÓGICA DE ESTRUCTURAS MORFOLÓGICAMENTE NO DEFINIDAS MEDIANTE ALGORITMOS GENÉTICOS

Tesis Doctoral

Por

Samuel Sánchez Caballero

Abril 2012

Esta editorial es miembro de la UNE, lo que garantiza la difusión y comercialización de sus publicaciones a nivel nacional e internacional.

Primera edición, 2012

© de la presente edición:
Editorial Universitat Politècnica de València
www.editorial.upv.es

ISBN: 978-84-8363-894-0 (versión impresa)

UNIVERSITAT POLITÈCNICA DE VALENCIA

DEPARTAMENTO DE INGENIERÍA MECÁNICA
Y DE MATERIALES

OPTIMIZACIÓN ESTRUCTURAL Y TOPOLÓGICA DE ESTRUCTURAS MORFOLÓGICAMENTE NO DEFINIDAS MEDIANTE ALGORITMOS GENÉTICOS

Tesis Doctoral

Autor: Samuel Sánchez Caballero
Ingeniero en Organización Industrial

Directores: Dr. D. Vicente Jesús Seguí Llinares
Doctor Ingeniero en Organización Industrial
Dr. D. José Enrique Crespo Amorós
Doctor Ingeniero en Materiales
Dr. D. Miguel Ángel Sellés Cantó
Doctor Ingeniero en Organización Industrial

A mi mujer Noelia,

y a mi pequeño Diego.

Agradecimientos

Me gustaría agradecer en primer lugar al profesor Mihir Sen de la *University of Notre Dame*, Indiana, USA, por haber sembrado la semilla que ha germinado en el presente trabajo.

A los profesores de la *University of Oradea* Adrian Pop y Flavius Ardelean por hacer mi estancia en Rumanía tan agradable.

A mis directores de tesis por sus aportaciones y sabios consejos.

A mi amigo y mentor Rafael Pla Ferrando por tantas y tantas cosas.

A Mathew Wall por crear una librería de Algoritmos Genéticos tan maravillosa.

A Juan Rada-Vilela por su sencilla y útil librería de lógica Fuzzy.

A Adolfo Hilario por introducirme en el mundillo del LaTeX y por su paciencia resolviendo mis interminables dudas.

A mis padres Asensio y Dolores, por haberme dado una educación y valores que me acompañarán durante toda la vida.

Especialmente a mi mujer Noelia y mi hijo Diego por todo el tiempo que les he quitado y por soportar mi mal humor durante el desarrollo de este trabajo.

Resumen

La optimización de estructuras ha sido una disciplina muy estudiada por numerosos investigadores durante los últimos cuarenta años. A pesar de que durante los primeros veinte años las técnicas de *Programación Matemática* fueron la herramienta fundamental en este campo, estas han ido perdiendo fuelle frente a un nuevo conjunto de técnicas *metaheurísticas* basadas en la *Computación Evolutiva*. De entre destacan, de manera significativa, los *Algoritmos Genéticos*.

Durante estos años la optimización de estructuras ha ido evolucionando de la mera optimización del tamaño, a la optimización del número óptimo de barras y coordenadas de los nudos (topología), y finalmente a la optimización de ambos tipos de variables de forma simultánea.

La irrupción de estas nuevas técnicas en el campo de la optimización de estructuras es debida, en gran medida, a las dificultades que tenía la programación matemática para realizar la optimización simultánea de las variables de diseño debido a la elevada alinealidad de estas y sus restricciones.

Las técnicas metaheurísticas por contra, son matemáticamente más sencillas y mas fáciles de implementar computacionalmente, ya que no requieren de otra información a parte de la función objetivo que determina la aptitud de las soluciones. No requieren por lo tanto de complejos procesos de linealización ni trabajar con las derivadas de la función.

El objetivo fundamental del presente trabajo es ir un poco más allá en el proceso de la optimización simultánea de las variables de diseño, definiendo un algoritmo que no parte de una estructura predefinida y que incorpora los parámetros que determinan la geometría. A diferencia de los métodos actuales, el algoritmo desarrollado no requiere de ningún tipo de estructura inicial ni otro tipo de infor-

mación adicional, aparte de la definición de los puntos de aplicación de las cargas, los puntos de apoyo y el tipo de apoyo.

El nuevo algoritmo desarrollado se justifica según la siguiente hipótesis: *La definición previa de la forma, geometría, regla o modelo preconcebido en una estructura suponen restricciones del diseño en sí mismas y por lo tanto aquel algoritmo que no se encuentre sujeto a estas deberá poder generar diseños necesariamente mejores, o al menos tan buenos como los existentes.*

A partir de esta hipótesis se desarrolla un nuevo algoritmo con una codificación mixta, adaptada a cada grupo de variables de diseño, donde los diferentes operadores se definen y actúan de forma independiente para grupo. El algoritmo incorpora además diferentes operadores que aseguran la legalidad de las soluciones a evaluar, así como un conjunto de estrategias orientadas a mantener la diversidad de las soluciones y a reducir las necesidades computacionales del algoritmo, el talón de Aquiles de las técnicas metaheurísticas.

Una vez desarrollado el algoritmo se procede a su validación mediante un problema de optimización de estructuras clásico: la optimización de una estructura de diez barras y seis nodos en voladizo, sujeta a restricciones de tensión y desplazamiento. Mediante este problema se evalúa de forma individualizada las diferentes estrategias implementadas en el algoritmo para operador en cada grupo de variables de diseño. Como resultado a este apartado se logra el diseño de menor peso obtenido hasta la fecha de entre un total de sesenta artículos científicos indexados donde se ha empleado como técnica de validación.

Del estudio del proceso evolutivo realizado en el apartado de validación, y su comparación con los diseños óptimos publicados en anteriores trabajos, se puede concluir que efectivamente el diseño del propio algoritmo condicionó el resultado de los mismos, probando la hipótesis inicial. Además se han podido corroborar los teoremas de Fleron, y se ha conseguido generalizar la topología óptima para la estructura analizada ya que esta ha permanecido invariable en todas las ejecuciones realizadas.

Entre las principales aportaciones del presente trabajo, aparte del desarrollo del propio algoritmo destacan el desarrollo de nuevos operadores genéticos, aplicados deforma individualizada a cada grupo de variables de diseño, la definición de una nueva función de penalización, la posibilidad de adquirir información adicio-

nal mediante el proceso evolutivo debido a la propia definición del algoritmo y finalmente la obtención de un nuevo mínimo para la estructura analizada, un un 10,92% inferior al mejor resultado publicado hasta la fecha.

Resum

L'optimització d'estructures ha sigut una disciplina molt estudiada per nombrosos investigadors durant els darrers quaranta anys. A pesar que durant els primers vint anys les tècniques de *Programació Matemàtica* van ser l'eina fonamental en aquest camp, aquestes han anat perdent força enfront d'un nou conjunt de tècniques *metaheurísticas* basades en la *Computació Evolutiva*. Entre aquestes destaquen, de manera significativa, els *Algoritmes Genètics*.

Durant aquests anys l'optimització d'estructures ha anat evolucionant de la mera optimització de la grandària, a l'optimització del nombre òptim de barres i coordenades dels nusos (topologia), i finalment a l'optimització de tots dos tipus de variables de forma simultània.

La irrupció d'aquestes noves tècniques en el camp de l'optimització d'estructures és deguda, en gran part, a les dificultats que tenia la programació matemàtica per a realitzar l'optimització simultània de les variables de disseny a causa de l'elevada alinealitat d'aquestes i les seues restriccions.

Les tècniques metaheurísticas per contra, són matemàticament més senzilles i més fàcils d'implementar computacionalment, donat que requereixen d'altra informació a part de la funció objectiu que determina l'aptitud de les solucions. No requereixen per tant de complexos processos de linealització ni treballen amb les derivades de la funció.

L'objectiu fonamental d'aquest treball és anar una mica més enllà en el procés de l'optimització simultània de les variables de disseny, definint un algoritme que no parteix d'una estructura predefinida i que incorpora els paràmetres que determinen la geometria. A diferència dels mètodes actuals, l'algoritme desenvolupat no requereix cap tipus d'estructura inicial ni cap altre tipus d'informació addicional,

a part de la definició dels punts d'aplicació de les càrregues, els punts de suport i el tipus de suport.

El nou algoritme desenvolupat es justifica segons la hipòtesi següent: *La definició prèvia de la forma, geometria, regla o model preconcebut en una estructura suposen restriccions del disseny en si mateixes i, per tant, aquell algoritme que no es trobe subjecte a aquestes haurà de poder generar dissenys necessàriament millors, o almenys tan bons com els existents.*

A partir d'aquesta hipòtesi es desenvolupa un nou algoritme amb una codificació mixta, adaptada a cada grup de variables de disseny, on els diferents operadors es defineixen i actuen de forma independent per a grup. L'algoritme incorpora a més diferents operadors que asseguren la legalitat de les solucions a avaluar, així com un conjunt d'estratègies orientades a mantenir la diversitat de les solucions i a reduir les necessitats computacionals de l'algoritme, el taló d'Aquil · les de les tècniques metaheurístiques.

Una vegada desenvolupat l'algoritme, es valida mitjançant un problema d'optimització d'estructures clàssic: l'optimització d'una estructura de deu barres i sis nodes en volada, subjecta a restriccions de tensió i desplaçament. Mitjançant aquest problema s'avaluen de manera individualitzada les diferents estratègies implementades en l'algoritme per a operador en cada grup de variables de disseny. Com a resultat, en aquest apartat s'aconsegueix el disseny de menor pes obtingut fins avui entre un total de seixanta articles científics indexats en què s'ha emprat com a tècnica de validació.

De l'estudi del procés evolutiu realitzat en l'apartat de validació, i la comparació d'aquest amb els dissenys òptims publicats en treballs anteriors, es pot concloure que efectivament el disseny del mateix algoritme en va condicionar el resultat, fet que prova la hipòtesi inicial. A més s'han pogut corroborar els teoremes de Fleron, i s'ha aconseguit generalitzar la topologia òptima per a l'estructura analitzada, ja que aquesta ha romàs invariable en totes les execucions realitzades.

Entre les principals aportacions d'aquest treball, a part del desenvolupament del mateix algoritme, destaquen el desenvolupament de nous operadors genètics, aplicats de manera individualitzada a cada grup de variables de disseny, la definició d'una nova funció de penalització, la possibilitat d'adquirir informació addicional del procés evolutiu a causa de la mateixa definició de l'algoritme i, finalment,

l'obtenció d'un nou mínim per a l'estructura analitzada, un 10,92% inferior al millor resultat publicat fins ara.

Abstract

Structural optimization has been widely studied over the last forty years. Although *Mathematical Programming* has been the main tool during the first twenty years, it ran out of steam in front of a new group *metaheuristic* techniques based in *Evolutionary Computation*. Among them, *Genetic Algorithms* are meaningfully highlighted.

During the last years structural optimization has evolved from mere size optimization, to number of bars and joints placement (topology) optimization, and finally to simultaneous optimization.

The irruption of these new techniques in the field of structural optimization is owed, to a large extent, to mathematical programming lack of capacity to manage the simultaneous optimization owed to constraint and design variables high nonlinearities.

However, metaheuristic techniques are mathematically simpler and computationally easier because it is not required any extra information, aside from the objective function, to evaluate the solution fitness. Therefore, there are not required any linearization process neither function derivatives.

The main aim of present work is to go further a little in the simultaneous optimization of design variables, defining a new algorithm that does not need a predefined structure and covers the geometry parameters. Unlike current methods, the developed algorithm does not require any initial structure neither another type of additional information, apart from the load and support keypoints and support class definition.

The new algorithm lays assuming the following hypothesis: *The previous definition of the form, geometry, rule or preconceived model implies a design constraint by*

themselves and so such algorithm, which is not subjected to that constraint must generate better designs, or at least as good as the previous ones.

A new algorithm was developed using this hypothesis, using a mixed code, adapted to each group of design variables, defining different operators who act independently on each group. The algorithm incorporates, in addition, some operators to ensure the solution's legality before being evaluated, as well as a group of strategies oriented to keep the solution diversity and to reduce the computational effort, the Achilles' heel of metaheuristics techniques.

Later, the algorithm was proven by a classical structural optimization problem: the ten-bar and six nodes cantilever structure, considering displacement and stress constraints. Using this structure as a benchmark, the different strategies implemented in the algorithm, for each operator over each group of design variables, were individually evaluated. As a result, the lower weight design reported, until now, was obtained, considering until sixty scientific papers where the structure was used as a benchmark.

Through the study of the evolutionary process developed during the benchmarking, and by comparison between the previously reported designs, it can be concluded that the previous reported algorithms were constrained by their own definition, conditioning so their results, so the initial hypothesis was proven. Moreover, the theorems of Fleron were proven, and also it was possible to generalize the optimum topology because it remained unchanged during all runs.

Between the main contributions of the present work, apart from the algorithm development, it can be stood: the new genetic operators, the penalty function, the possibility of acquire new information by the evolutionary process (owed to the algorithm definition) and finally obtaining a new minimum weight for the benchmark structure, a 10,92% lower than any other reported before.

Índice

Listado de Símbolos

Ψ	Función de finalización del algoritmo. 84
Φ	Fitness o aptitud del individuo. 84, 90, 92, 132, 133
$\Phi_{\max}$	Fitness o aptitud del mejor individuo. 97, 132, 133
$\Phi_{\min}$	Fitness o aptitud del peor individuo. 96, 97, 132
$\overline{\Phi}$	Valor promedio de la aptitud de la población. 150, 151
i	Subíndice. Elemento. 92, 99, 100
κ_i	Número de veces que un individuo i espera ser seleccionado para la reproducción. 92–96, 150
κ_{pla}	Número máximo de veces que el individuo más apto de la población es seleccionado para la reproducción. 98
λ	Número de hijos generados. 71
l_i	Valor mínimo que puede tomar el gen $i \in \{l_i, u_i\}$. 122, 124, 125
μ	Tamaño de la población. 71, 84, 91, 92, 94, 95, 97, 98, 149
$\mathbb{N}$	Números naturales. 161, 164, 166, 204

n_f	Número de fenotipos presentes en el genoma $\boldsymbol{s}$. 103–105
n_g	Número de genes presentes en el genoma $\boldsymbol{s}$. Longitud del genoma. 103–105, 121, 122, 124, 125, 133
n_{sel}	Número de individuos que participan en el proceso de selección. 91
n_{tor}	Número de individuos que participan en cada torneo. 99, 100
p_c	Probabilidad de cruce o crossover. 102
p_m	Probabilidad de mutación. 84, 121, 122, 124, 125
ϕ	Presión selectiva. 99, 133
p_s	Probabilidad de selección de un individuo. 92, 93, 95, 96, 98, 99, 149
p_w	Probabilidad de ganar en un torneo binario. 100
$\mathbb{R}$	Números reales. 66, 151, 158, 159, 164
r	Número aleatorio. 103, 107, 113
$\mathbb{R}^+$	Números reales positivos. 107, 113, 146, 147, 161, 164
rnd(0,1)	Función que genera un número aleatorio normal gausiano comprendido en el intervalo [0,1]. 92, 93, 97, 98, 100, 107, 113, 118, 122–125
$\boldsymbol{s}$	Genoma correspondiente a un individuo. xxii, 86, 103, 118
$\boldsymbol{s}_h$	Genoma del hijo resultante proceso de recombinación. 107–110, 112, 113, 115, 116, 118
$\boldsymbol{s}_p$	Genoma del padre en el proceso de recombinación. 106–110, 112–115, 117, 118
T	Número de iteraciones o generaciones que se deja correr el algoritmo. 99, 134

t	Número de iteración o generación. 90, 92, 99, 135
u_i	Valor máximo que puede tomar el gen $i \in [l_i, u_i]$. 122, 124, 125
$\boldsymbol{\chi}$	Vector población. $X : \{\chi_1, \cdots, \chi_\mu\}$. 84, 91–93, 95, 97, 98, 100
$\overline{\boldsymbol{\chi}}$	Conjunto de padres seleccionados mediante el operador de selección. 84, 91–93, 95, 97, 98, 100, 115
$\boldsymbol{\chi'}$	Conjunto de hijos engendrados. 84, 122, 124, 125
$\boldsymbol{\chi''}$	Conjunto de hijos engendrados tras sufrir mutaciones. 84, 91, 122, 124, 125
$\boldsymbol{\chi^o}$	Vector población ordenada según un criterio dado. $X^o : \{\chi_1^o, \cdots, \chi_\mu^o\}$. 97, 98

Listado de Acrónimos

ACO	Ant Colony Optimization. Optimización por Colonias de Hormigas. 17, 18, 24, 50, 61
BB	Branch and Bound. Ramificación y poda. 46, 55
BBBC	Big Bang Big Crounch. Gran Estallido-Gran Crujido. 50, 191
BFS	Breadth-First Search. Método de búsqueda de la teoría de grafos empleado para la detección de la conectividad. 170, 171
BLX-α	α - Blend Crossover. Operador α de cruce por mezcla. xv, 110, 112, 118, 119, 166, 201, 203
DE	Differential Evolution. Evolución Diferencial. 50, 192
DFS	Depth-First Search. Método de búsqueda de la teoría de grafos empleado para la detección de la conectividad. 170
EA	Evolutionary Algorithms. Algoritmos evolutivos. 71, 121, 136, 137, 147, 148
EC	Evolutionary Computation. Computación evolutiva. 71
EP	Evolutionary Programming. Programación evolutiva. 71

ES	Evolutionary Strategies. Estrategias evolutivas. 72, 131, 145
FIFO	First in Firs out. Método de ordenación de una cola o pila. 170
GA	Genetic Algorithm. Algoritmo genético. 71, 90
GASOP	Genetic Algorithm for Structural Optimization. 153
GGA	Generational Genetic Algorithm. Algoritmo genético con reemplazo generacional. 126, 155, 156
GP	Genetic Programming. Programación genética. 72
HPSO	Harmonic Search + Particle Swarm Optimization. Técnica formada por la combinación de HS y PSO. 64, 192
HPSOACO	Harmonic Search + Particle Swarm Optimization + Ant Colony Optimization. Técnica formada por la combinación de HS, PSO y ACO. 64, 192
HS	Harmonic Search. Búsqueda Armónica. 50, 64, 191
MOGA	Multi Objective Genetic Algorithm. Algoritmo Genético Multiobjetivo. 191
MP	Mathematical Programming. Programación Matemática. 13, 17, 18
OX	Order Crossover. Operador de cruce parento-céntrico. 120
PCX	Parent-Centric Crossover. Operador de cruce parento-céntrico. 115, 120, 166, 201, 203

PMX Partially Mapped Crossover. Operador de cruce de relación parcial. 120

PPX Precedence Preservative Crossover. Operador de cruce con preservación de la precedencia. 120

PSO Particle Swarm Optimization. Optimización por Enjambre de Partículas. 17, 18, 24, 50, 63, 191

RCGA Real Coded Genetic Algorithm. Algoritmo genético con codificación real. 85

RWS Roulette Wheel Selection. Operador de selección de ruleta. xv, 92, 94, 95, 100, 198

SA Simulated Annealing. Recocido Simulado. xiv, 13, 17, 18, 24, 47, 48, 57

SBX-β β - Simulated Binary Crossover. Operador β de cruce binario simulado. xv, 112, 119, 166, 194, 201–203, 232, 238

SLP Sequential Linear Programming . Programación Lineal Secuencial. 46

SPX Simplex Crossover. Operador de cruce simplex. 117–120, 166, 201, 203

SQP Sequential Quadratic Programming . Programación Cuadrática Secuencial. 42, 46

SSGA Steady State Genetic Algorithm. Algoritmo genético con brecha generacional. 126, 155, 156, 194, 208

SUS Stochastic Universal Sampling. Operador de selección de muestreo universal estocástico. xv, 94, 198

UNDX-m Unimodal Normal Distribution Crossover. Operador de cruce unimodal de distribución normal. 114–117, 119, 120, 166, 201, 203, 293

Glosario

cromosoma	Cadena binaria o numérica. 76
epístasis	Interacción de dos o más genes en la formación de un genotipo. 80, 102, 105
esquema	Es un patrón o esquema de repetido en varios cromosomas. En un cromosoma binario puede especificarse con una cadena de igual longitud del cromosoma donde cada gen puede tomar una de los valores [0,1,@] donde el símbolo @ representa un gen que no forma parte del patrón. 85, 86, 89, 91, 104, 105, 111, 112, 114
fitness	Cuantificación del grado de adaptación del individuo al medio. 84
función de aptitud	Función que evalúa al individuo en base a su genotipo. 84, 135
gen	Rasgo, caracter, parámetro del problema. 83
generación	Iteración. 82
individuo	Representa una de las soluciones del problema. 83
mapear	Codificar o decodificar, relacionar dos conjuntos o funciones mediante determinadas reglas. 83, 86, 88, 94, 98

población	Conjunto de individuos para los cuales se evalúa la función objetivo. 82
presión selectiva	Es el cociente entre la aptitud máxima y la media de las aptitudes de una población. 93, 96, 98, 99, 133
recocido simulado	Técnica de optimización basada en la analogía del enfriamiento del acero según una distribución de Boltzman. 99
schema	Esquema. 85

1

Introducción y objetivos

1.1 Reseña histórica

Desde la antigüedad el ser humano ha tenido interés en diseñar edificios y estructuras seguros. Uno de los escritos más antiguos que se conservan, el código Hammurabi (1760 a.C), recoge que si una casa se derrumbaba y mataba al dueño, alguno de sus hijos o esclavos; el arquitecto-constructor, su hijo o su esclavo debía ser condenado a la pena de muerte (leyes 229-231) [221]. Durante la época del imperio romano, las pruebas de carga de los puentes eran realizadas colocando a su arquitecto debajo. Esto constituía, en cierto modo, un mecanismo de selección natural [92]: solo los mejores diseñadores sobrevivían.

Desde muy antiguo, antes incluso del desarrollo de las matemáticas ya se construyeron estructuras magníficas mediante la reproducción de modelos a escala. Ya en la antigua Mesopotamia se construyeron bóvedas de piedra con un arco de más de 20 metros que se mantienen en pie aun hoy en día.

Para construir un edificio los arquitectos griegos relacionaban todas sus dimensiones con una medida llamada módulo. El módulo era, en cada caso, la medida del diámetro de la columna del edificio a construir. El resto de las dimensiones como el largo, ancho y altura del edificio, se calculaban multiplicando una cierta cantidad de veces ese módulo. Cada uno de los órdenes griegos se caracteriza entre otras cosas por utilizar un módulo diferente.

Ante la falta de teorías que permitieran realizar diseños más precisos, el diseño estructural se reducía a repetir diseños probados. Dichos diseños se fueron acumulando durante siglos dando lugar a tratados como el *De Architectura libri decem* de Vitruvio (I a.C.). Éste fue el único tratado del período clásico que sobrevivió tras la caída de Roma, aunque no fue el único como el propio autor relató, sirviéndose de muchos textos, principalmente griegos, de los cuales sólo ha quedado su mención.

Durante la edad media fue copiado y conservado en las bibliotecas de los monasterios, siendo la fuente de inspiración del trabajo de Leon Battista Alberti *De re aedificatoria* (1485), considerado el tratado más importante del renacimiento.

La falta de una teorías matemáticas no impidió, no obstante, la ejecución de obras colosales como el Panteón de Agripa (27 a.C.), que ha resistido durante

diecinueve siglos sin reformas o refuerzos, o la cúpula de Santa María del Fiore (1418) proyectada por Brunelleschi e inspirada en el anterior.

Los primeros métodos científicos para crear modelos matemáticos y verificarlos mediante experimentos fueron enunciados por Francis Bacon (1561-1626) y Galileo Galilei (1564-1642). El propio Galileo en su obra [125] *Discorsi e dimonstrazioni matematiche, intorno, a due nuove scienze attenenti alla mecanica et i movimenti locali* (1638) puso en práctica el modelo científico en el campo de la construcción. Nuestra teoría moderna de la construcción es en buena medida sucesora directa de la teoría de la solidez de las construcciones presentada en ella.

La investigación de las construcciones fue desgajada del resto de la teoría arquitectónica durante siglos, dando lugar a un gremio separado: el de los ingenieros.

El nombre *ingeniero*, viene de la palabra latina *ingenium*, y tiene las siguientes acepciones: *genio*, *un producto del genio* e *invención*. Esta palabra ya fue empleada en la Edad Media para los arquitectos hábiles. Esta palabra fue adoptada por el marqués de Vauban cuando fundó un departamento de construcciones, *Corps des ingénieurs*, en el ejército francés en 1675. En este tiempo era habitual para los ingenieros militares diseñar castillos, planos de ciudades e incluso iglesias. Esta nueva profesión especializada en cuestiones de construcción quedó organizada bastante rápidamente y en 1747 fue fundada en París una escuela especial, Ecole des Ponts et Chaussées. Escuela en la cual se licenció Claude-Louis Navier en 1806, considerado uno de los padres de la teoría de la elasticidad.

Las figuras centrales del desarrollo de la teoría matemática de la construcción fueron Robert Hooke (1635-1703), Jakob Bernoulli (1654-1705) y Leonhard Euler (1707-1783). Desde Euler en adelante, la teoría de la elasticidad se desarrolló codo con codo junto a la teoría matemática. A estos les seguirían otros ilustres investigadores: Saint Venant, Navier, Poisson, Cauchy, Lamé y más recientemente Timoshenko.

El desarrollo comercial del hierro como elemento estructural se inicia finales del siglo XVIII en la construcción de puentes. Sin embargo, no es hasta las últimas del siglo XIX tras el desarrollo del convertidor Bessemer (1856) y del horno de hogar abierto (1867) que permitieron la fabricación de acero estructural, cuando empieza a emplearse en construcciones. En España, son construcciones significativas de esta

época: El Palacio de Cristal del Parque del retiro (1887), la estación de Atocha (1892), la torre Eiffel (1889).

En la población de Alcoi todavía se conservan construcciones de la época como el puente del viaducto de Canalejas (1907) y el edificio de la antigua escuela industrial (1936), sede de la Escuela Politécnica superior de Alcoi hasta el 2007.

Posteriormente, tras el desarrollo de la tecnología de la soldadura, que se inició con la construcción de los buques cisterna *Liberty Ships* [220] durante la segunda guerra mundial, el avance de las construcciones metálicas ha sido extraordinario, substituyendo a materiales tradicionales de construcción en muchos casos.

Ha sido durante las últimas cuatro décadas, gracias al desarrollo del ordenador y de los métodos de cálculo de estructuras adaptados a este, cuando han empezado a emplearse diferentes técnicas de optimización en las estructuras metálicas.

1.2 Tamaño, forma y topología

El problema de la optimización en estructuras se puede abordar desde diferentes enfoques, que se dividen fundamentalmente los siguientes:

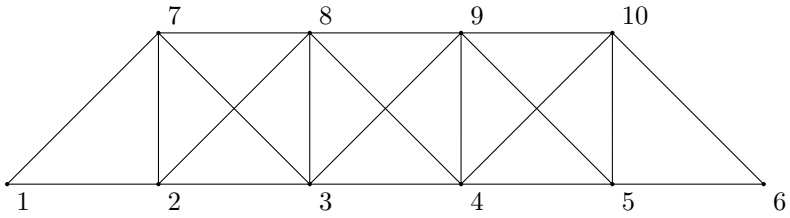
1. Optimización del tamaño. Trata de determinar las medidas óptimas de los elementos que constituyen una estructura de forma predefinida. Es el enfoque más sencillo ya que el número de variables del problema es bastante limitado.

2. Optimización de la forma. Trata de buscar la forma óptima de una estructura de topología fija. En este enfoque, partiendo de una forma predefinida se busca optimizar esta mediante la modificación de la conectividad entre elementos o mediante la eliminación de los elementos menos esforzados.

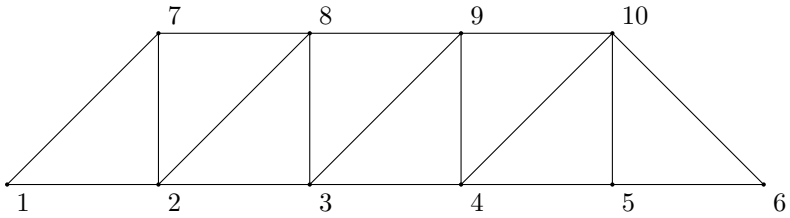
3. Optimización de la topología. Trata de buscar la distribución óptima del material en una estructura, mediante la supresión de elementos y/o la modificación de la conectividad y/o de las coordenadas nodales. Este enfoque es el más complicado porque la cantidad de variables que aborda es elevado. En todos los trabajos publicados hasta la fecha se parte siempre de una morfología previa.

La la figura 1.1 muestra un ejemplo de los tres enfoques sobre una misma estructura.

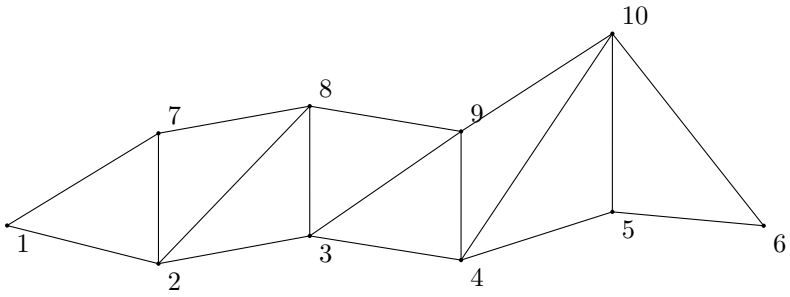
En el presente trabajo se pretende abordar por primera vez la optimización simultánea del tamaño y la topología de una estructura sin partir de ningún tipo de forma previa.



(a) Optimización del tamaño



(b) Optimización de la forma



(c) Optimización de la topología

Figura 1.1: Diferentes tipos de optimización estructural

1.3 La necesidad de optimización

La escasez de materias primas y el incremento de la demanda de estas empuja a las empresas a abaratar costes, que en el caso de las estructuras pasa por reducir el peso de las mismas.

En sectores como la automoción, la reducción de peso no solo interesa a los fabricantes, sino también a los compradores donde el peso está directamente relacionado con el consumo de combustible, cuyo precio se encuentra también al alza como se puede apreciar en la figura 1.2.

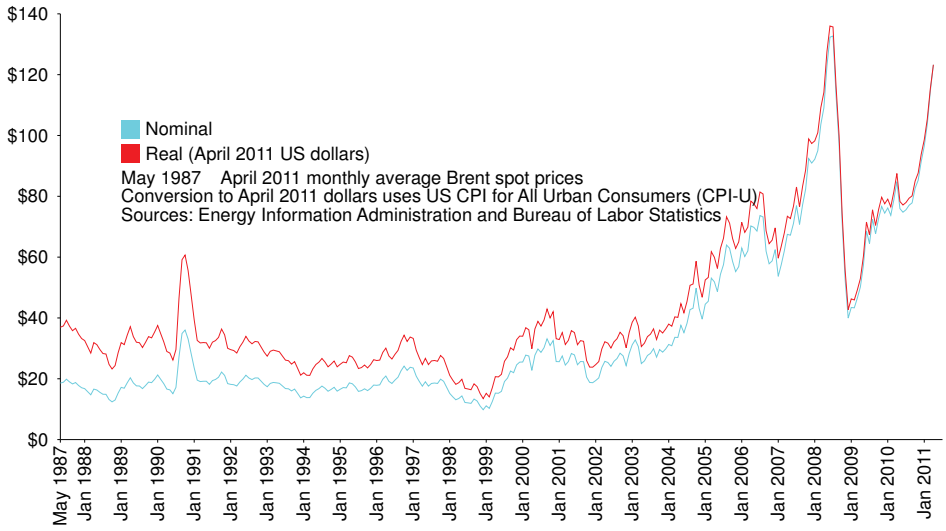


Figura 1.2: Evolución de los precios del barril de petróleo Brent. Figura bajo licencia de Creative Commons Attribution-ShareAlike 3.0 Unported

Por otra parte el desarrollo de los automóviles en los últimos años ha provocado un incremento de significativo de su peso debido fundamentalmente a la incorporación de sistemas de seguridad (ABS, ESP, Airbag, etc ...) y de confort (climatizador, CAN-Bus, etc...). De este modo, por ejemplo, el Volkswagen Golf ha pasado de 790 a 1318 kg en 35 años, según se aprecia en la figura 1.3. Este incremento habría sido mayor de no haber sido por la reducción del peso del chasis y demás elementos estructurales.

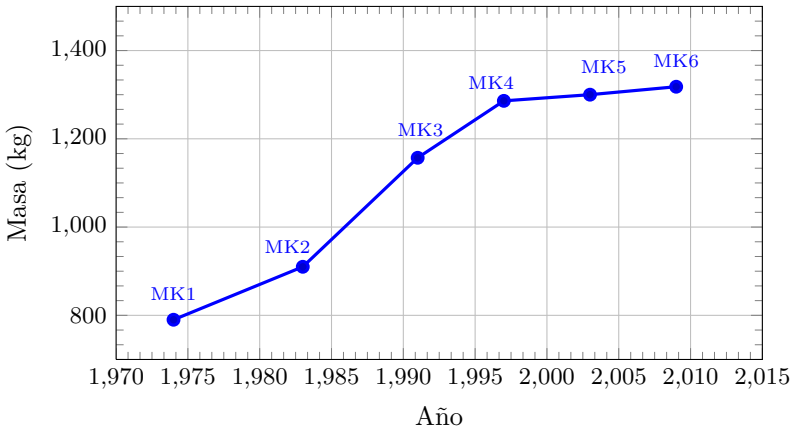


Figura 1.3: Evolución del peso del Wolkswagen Golf. Fuente VW Group

En el transporte de mercancías, se estima que en promedio una reducción de 1 kN en el peso de la estructura supondría un ahorro anual de 3600 €[1] [255] debido al ahorro en combustible.

Se hace pues necesaria la optimización de los recursos disponibles lo cual se traduce en el caso de las estructuras metálicas en el abaratamiento de los costes de producción y sobre todo de los materiales.

El diseño tradicional de estructuras se basa en un conjunto de reglas o teorías, como la resistencia de materiales, que permiten al proyectista establecer los materiales y geometrías necesarias para soportar un conjunto de esfuerzos y reacciones. Ahora bien, estos diseños generalmente no serán óptimos desde el punto de vista económico o de ahorro de materiales, ya que se encuentran afectados por una serie de decisiones subjetivas durante el proceso de diseño. De este modo, dos proyectistas diferentes llegarán a dos soluciones diferentes perfectamente factibles desde el punto de vista de la resistencia estructural o del cumplimiento de normas. Sin embargo estas estructuras seguramente tendrán costes y pesos diferentes.

Una de las primeras decisiones subjetivas que adoptarán los diseñadores será la definición de la topología de la estructura, para lo cual seguramente emplearán el método romano, mediante el cual se repiten los modelos llevados a cabo con éxito anteriormente. Un ejemplo de este condicionamiento del diseño es la estructura presente aún hoy en día en los semirremolques, también presente en los automóviles

[1] Costes actualizados al 2011

hasta mediados del siglo pasado. Esta estructura hoy en día tiene la forma de escalera típica de los carros de la edad media.

El proceso de optimización, basado en el diseño tradicional, es un proceso iterativo donde poco a poco se va modificando la geometría de las secciones del diseño inicial, realizado generalmente de forma subjetiva. De este modo en cada etapa se comprueba el cumplimento de las restricciones de seguridad y diseño impuestas a la estructura y se propone de un modo más o menos científico las modificaciones a realizar en la geometría para la siguiente etapa. Este proceso es extremadamente costoso para un cálculo manual y su aparición es relativamente reciente debida a la utilización de los ordenadores personales. El proceso tradicional de optimización generalmente solo aborda la optimización de la geometría de la estructura y suele presentar una única solución óptima.

Los métodos de optimización más modernos están basados en técnicas heurísticas o metaheurísticas porque como se verá posteriormente, los métodos tradicionales son incapaces de abordar espacios de soluciones tan grandes, con la complejidad adicional de ser dispersos y no convexos. Estas técnicas, al igual que el método tradicional también pueden estar afectadas por decisiones subjetivas o prejuicios del diseñador o simplemente por la capacidad de computación disponible en ese momento. Una de esas decisiones subjetivas es la definición de la topología inicial del problema. A diferencia del método tradicional alguna de estas técnicas si que modifican la morfología del problema, además de su geometría, pero ya parten de una decisión subjetiva que condicionará sin lugar a dudas el proceso de búsqueda.

Como se analizará en los sucesivos capítulos, el presente trabajo logra un método de diseño automatizado, basado en los Algoritmos Genéticos que proporciona al diseñador un conjunto de soluciones óptimas, desde el punto de vista del peso de la estructura, que cumpla con los requerimientos de resistencia y rigidez impuestos por este. Este método no requiere de ningún tipo de idea preconcebida a parte de las especificaciones de los puntos de carga y soporte de la estructura. Con esto se eliminan prácticamente los inconvenientes de todos los procesos de optimización de estructuras presentes hasta el momento. De este modo, no se requerirá de ningún tipo de habilidad o conocimiento especial por parte del diseñador, a parte de los necesarios para poder interpretar y hacer uso de los resultados. La labor del diseñador será la de seleccionar la mejor solución, de entre las propuestas, en base a otros criterios de tipo económico o estético no contemplados en el algoritmo.

1.4 Estado del arte sobre la optimización topológica y estructural simultánea

Como punto de partida para el presente trabajo científico, es necesario conocer los trabajos previos publicados referidos al tema a estudiar. Este estudio permitirá conocer la viabilidad y el interés científico y social del mismo. Por este motivo, se ha realizado un estudio bibliográfico sobre la optimización de estructuras formadas por barras o vigas, con objeto de fundamentar este trabajo.

1.4.1 Fuentes de información

De las diversas bases de datos existentes destaca por su volumen y calidad *SciVerse Hub*. Esta base de datos de reciente creación realiza las búsquedas en las bases de datos de artículos como *Scopus* y *Sciencedirect*, así como en bases de datos de congresos como *ISI proceedings* entre otras. También realiza búsquedas en bases de datos de tesis doctorales y de master via *NDLTD* así como diferentes bases de datos de patentes y la propia web.

A continuación se detalla el rango de búsqueda de *SciVerse Hub* sobre artículos los científicas relacionados con el tema de investigación:

- 10,3 millones de artículos científicos de ScienceDirect
- 691.000 artículos científicos de SAGE Publications
- 524.000 artículos científicos de Scitation
- 252.000 artículos científicos de Wiley-Blackwell
- 661.000 artículos científicos de Nature Publishing Group
- 477.000 artículos científicos de American Physical Society
- 337.000 artículos científicos de IOP Publishing
- 66.400 artículos científicos de Royal Society Publishing
- 23.300 artículos científicos de Maney Publishing
- 1,6 millones de tesis doctorales y de master via NDLTD

- 78.500 documentos de MIT OpenCourseWare
- 25.400 technical reports de NASA
- 657.000 e-prints de ArXiv.org
- 26.600 documentoss de Caltech Coda
- 3.400 e-prints de Cogprints
- 4,2 millones de documentos de Digital Archives
- 110.000 artículos de Project Euclid
- 34.700 artículos de Hindawi Publishing Corporation
- 6.400 documentos de HKUST Institutional Repository
- 87.000 documentos de The University of Hong Kong
- 29.400 documentos de IISc
- 157.000 documentos de WaY
- 19.000 documentos de Humboldt Universität
- 21.400 documentos de DiVA
- 24.700 documentos de University of Toronto T-Space
- 9.100 artículos de SIAM

El rango de búsqueda de patentes es de 24.4 millones contenidos en la base de datos de patentes *LexisNexis*.

El rango de búsqueda de *SciVerse Hub* en la web incluye sobre 410 millones de webs de contenido científico, incluyendo:

- 140 millones de sitios .edu
- 40 millones de sitios .org
- 23 millones de sitios .ac.uk
- 38 millones de sitios .com

- 39 millones de sitios .gov
- 136 millones de sitios relevantes pertenecientes a instituciones de investigación y universidades a lo largo del mundo

También se han utilizado otras bases de datos espacializadas en inteligencia artificial como *Mendeley*, recientemente incorporada en *Scopus*, *CiteseerX* incorporada a *SciVerse Hub* y *ACM Digital Library*.

1.4.2 Criterios de búsqueda

La búsqueda se ha realizado a lo largo de todos los campos de búsqueda posibles y sobre todos los documentos registrados en *SciVerse Hub*.

Si bien la realización de búsquedas en *SciVerse Hub* es sencilla, el tratamiento de la información no lo es tanto ya que es bastante común que la base de datos devuelva decenas o cientos de miles de referencias. Cabe pues establecer unos criterios de búsqueda adecuados para conseguir acotar los resultados dentro del campo científico objeto de estudio.

La primera forma de acotación de la búsqueda es la utilización de las palabras clave o *keywords*. En este caso y dado que se pretende optimizar el tamaño y la topología de estructuras formadas por barras o vigas se han empleado palabras clave como *truss*, *topology*. Estas palabras clave se relacionan entre sí mediante operadores binarios como *AND*, *OR*, *NOT*, etc . . . La tabla 1.1 muestra los patrones de búsqueda resultantes de la combinación de las palabras clave empleadas con los operadores binarios. En los citados patrones no se emplean otras palabras más generales como *structural* porque estas abren demasiado el abanico de búsqueda hacia otros campos como es la optimización estructural de medios continuos (ESO), la cual no está relacionada con el presente trabajo. Esto no supone una limitación en la búsqueda ya que, como se ha comentado anteriormente, *SciVerse Hub* realiza la búsqueda en todos los campos de cada referencia y será bastante difícil que un artículo que verse sobre la optimización de estructuras formadas por barras o vigas no contengan estas palabras en ningún campo.

La segunda forma de acotación es la definición de las áreas de búsqueda o *Subject Areas*. El presente estudio ha reducido la búsqueda a las siguientes áreas: *Computer Science*, *Engineering* y *Matematics*.

La tabla 1.1 muestra los resultados obtenidos mediante los criterios de búsqueda descritos.

1.4.3 Estudio de la producción científica global

A partir de La tabla 1.1 los resultados se clasifican en dos grupos de patrones de búsqueda.

El primer grupo de patrones de búsqueda, cuyos resultados se reproducen en la figura 1.4, representa todos los trabajos publicados relacionados con la optimización topológica de estructuras formadas por vigas o barras. Como se puede apreciar los trabajos publicados utilizan principalmente métodos de *Programación Matemática* (MP) y Algoritmos Genéticos (GA) en una proporción bastante similar. A estos le siguen los trabajos relacionados con el *Recocido Simulado* (SA), siendo especialmente destacable la producción en Tesis la cual es proporcionalmente mucho mayor. El resto de métodos tiene una relevancia mucho menor hasta el punto de no tener tesis desarrolladas al respecto.

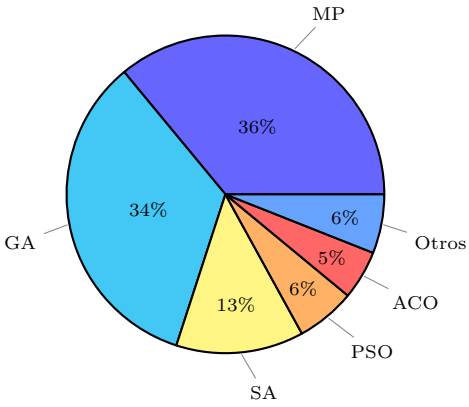
El segundo grupo, representado en La figura 1.5, es un subconjunto del primero y está formado por todos los trabajos donde se realiza una optimización simultánea del tamaño y la topología de estructuras formadas por vigas o barras. Este último es el campo donde se enmarca el presente trabajo. Como se puede apreciar la producción científica de trabajos relacionados con los Algoritmos Genéticos es superior al resto, siendo especialmente destacable la producción en congresos lo cual es tradicionalmente habitual en las publicaciones sobre este método de optimización. Posteriormente le sigue en importancia la *Programación Matemática* y el *Recocido Simulado*, al igual que en el grupo anterior. El resto de métodos tiene una relevancia mucho menor hasta el punto de no tener tesis, ni publicaciones en congresos relacionadas con el tema.

Cabe destacar la pequeña cantidad de patentes desarrolladas en ambos grupos, de las cuales ninguna está relacionada con el campo que se pretende estudiar.

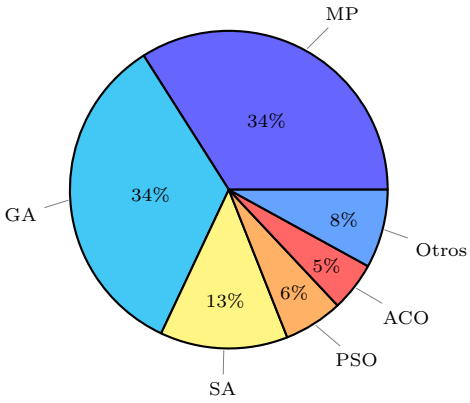
Este segundo grupo representa menos del 12% del primer grupo, lo cual muestra la poca investigación existente en este campo durante los últimos años. Sin embargo, según los resultados de los trabajos de Ebenau et al. [100], Balling et al. [21] y Tang et al. [384] muestran que es precisamente este camino el que logra mejores resultados en el campo de la optimización de estructuras.

Tabla 1.1: Patrones de búsqueda

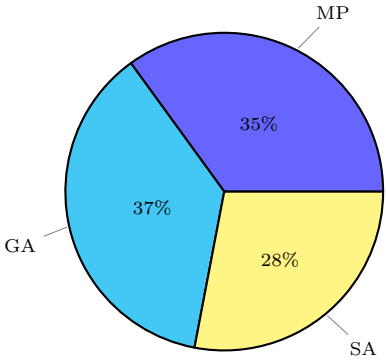
Patrones		**Palabras clave de búsqueda**		**Resultados**					
		Claves generales	Clave específica	Total	Artículos	Tesis	Congresos	Patentes	Otras Web
Primer grupo	Total	truss AND topology AND	optimization	3455	2010	36	248	8	1153
	MP		math. programming	1260	690	16	36	0	518
	GA		genetic algorithm	1184	692	17	89	0	386
	SA		simulated annealing	456	267	13	26	0	150
	PSO		particle swarm	202	111	0	13	0	78
	ACO		ant colony	162	94	0	8	0	60
Segundo grupo	Total	truss AND topology AND simultaneous AND (size OR sizing) AND	optimization	407	215	8	10	1	173
	MP		math. programming	227	102	6	2	0	117
	GA		genetic algorithm	256	115	6	7	0	128
	SA		simulated annealing	121	47	3	2	0	69
	PSO		particle swarm	57	25	0	0	0	32
	ACO		ant colony	41	18	0	0	0	23



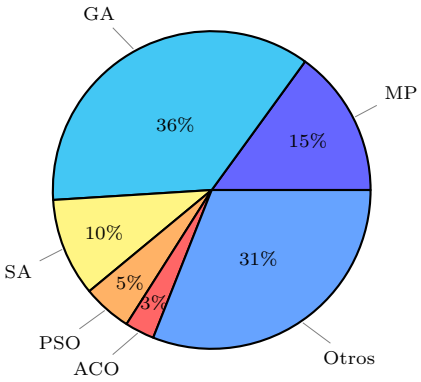
(a) Total



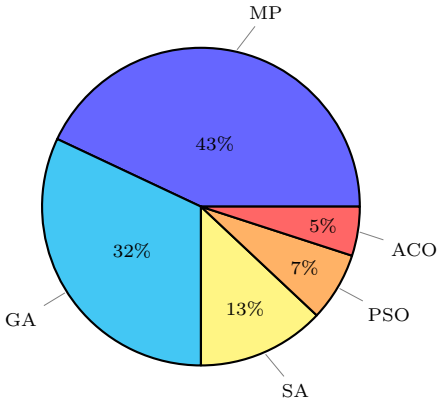
(b) Artículos científicos



(c) Tesis



(d) Congresos



(e) Otras webs

Figura 1.4: Producción científica. Patrones del primer grupo.

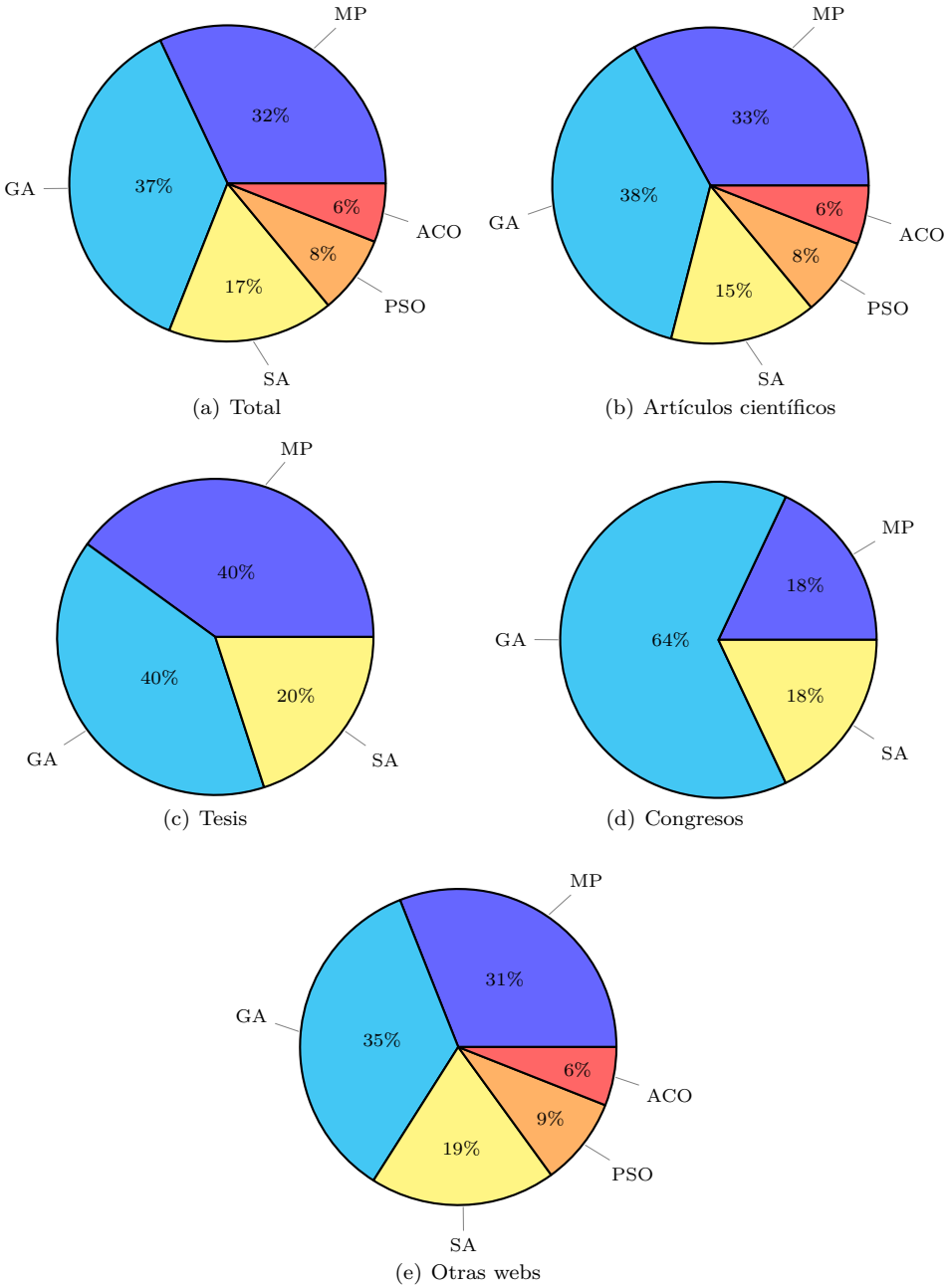


Figura 1.5: Producción científica. Patrones del segundo grupo.

1.4.4 Estudio de la producción científica por año de publicación

Los resultados globales obtenidos en el apartado anterior no son suficientes para determinar que métodos de optimización pueden resultar más interesantes ya que no muestra la tendencia. De hecho, de los diferentes métodos analizados la *Optimización por Colonias de Hormigas* (ACO) y la *Optimización por Enjambre de Partículas*(PSO) son relativamente recientes, los Algoritmos Genéticos y el SA se desarrollaron en la década de los 90 y la MP se inició en la década de los sesenta. Es presciso pues analizar también cual es el desarrollo temporal de la investigación con el fin de determinar las tendencias que puedan surgir.

El análisis de tendencia se realiza estudiando los artículos en revistas indexadas publicados según el patrón de búsqueda correspondiente con el segundo grupo, objeto de estudio del presente trabajo. Se ha empleado la producción en revistas y no el número total de documentos ya que son las primeras las que realmente nos indican en que se está investigando en la actualidad mientras que el segundo caso suele ser un indicativo de su aplicación actual.

Como se puede apreciar en la figura 1.6, a diferencia de lo que ocurre cuando se consideran los números absolutos de documentos o artículos, la producción científica actual relacionada con la *Programación Matemática* es bastante escasa. La producción actual se centra fundamentalmente en el empleo de los GA, la PSO y el SA. En el último año se ha visto incluso superada por una técnica relativamente nueva como la ACO

1.4.5 Estudio de la producción científica en revistas indexadas

Existen muchas revistas científicas especializadas que tratan el tema de la optimización de estructuras, sin embargo no tantas han publicado artículos relacionados con la optimización simultánea de topología y tamaño de dichas estructuras. Dado que una de las tareas fundamentales de cualquier trabajo de investigación científica es su divulgación, es necesario conocer en que revistas se están publicando trabajos relacionados con el campo de estudio para que esta investigación tenga la máxima difusión posible.

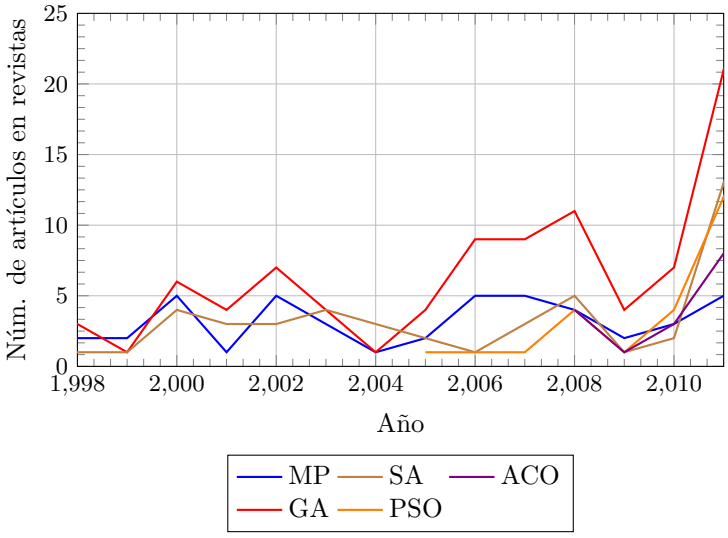


Figura 1.6: Número de documentos publicados sobre optimización topológica. Patrones del segundo grupo.

La tabla 1.2 muestra un extracto de las revistas donde se ha publicado mayor número de artículos relacionados con la temática del presente trabajo. De esta tabla se desprende que las revistas donde fundamentalmente se publican artículos relacionados con el campo de estudio son:

- Computers and Structures.
- Computer Methods in Applied Mechanics and Engineering.
- Applied Soft Computing.
- Advances in Engineering Software.
- International Journal of Solids and Structures.
- Discrete Applied Mathematics

La cantidad de artículos publicados por una revista no es garantía suficiente de su difusión. Además debe tenerse en cuenta otros factores que miden la calidad y difusión de la revista como son el factor de impacto o los factores SJR y SNIP de reciente creación.

Tabla 1.2: Principales publicaciones relacionadas con el campo de estudio

Revista	Temática					
	GA	**MP**	**SA**	**PSO**	**ACO**	**Total**
Advanced Engineering Informatics	2	0	1	1	1	5
Advances in Engineering Software	7	2	3	1	1	14
Applied Soft Computing	5	2	3	4	3	17
Automatica	0	1	1	0	0	2
CIRP Annals - Manufacturing Technology	1	0	1	1	1	4
Composite Structures	0	0	1	0	1	2
Comprehensive Composite Materials	1	0	1	0	0	2
Computer Methods in Applied Mechanics and Engineering	10	16	4	1	0	31
Computer-Aided Design	0	1	1	0	0	2
Computers and Structures	16	15	7	5	3	46
Discrete Applied Mathematics	3	3	3	0	0	9
Encyclopedia of Information Systems	1	0	1	0	0	2
Encyclopedia of Physical Science and Technology	1	1	1	0	0	3
Engineering Applications of Artificial Intelligence	2	0	0	1	0	3
Engineering Structures	3	0	0	0	2	5
Expert Systems with Applications	0	1	0	1	0	2
Finite Elements in Analysis and Design	2	5	0	0	0	7
International Journal of Solids and Structures	4	3	2	1	0	10
Journal of Computational Physics	1	1	0	0	0	2
Journal of Constructional Steel Research	0	0	0	1	1	2
Journal of Sound and Vibration	3	0	0	0	0	3
Materials & Design	0	1	1	0	0	2
Mathematical and Computer Modelling	2	0	2	1	0	5

La tabla 1.3 muestra diferentes indicadores de calidad obtenidos para las diferentes revistas mediante la *ISI Web of Knowledge*. Según esta tabla, las revistas más interesantes donde publicar son: Computer Methods in Applied Mechanics and Engineering, International Journal of Solids and Structures y Applied Soft Computing por este orden si se tienen en cuenta otros ratios además de el factor de impacto del 2010.

Estos indicadores sin embargo no nos muestran la evolución de la revista. Es importante también conocer la tendencia de las mismas para lograr maximizar la difusión del artículo en el futuro. Con este fin se empleó la herramienta *Journal Analyzer* de la base de datos *Scopus* cuyos resultados aparecen en las figuras 1.7 y 1.8.

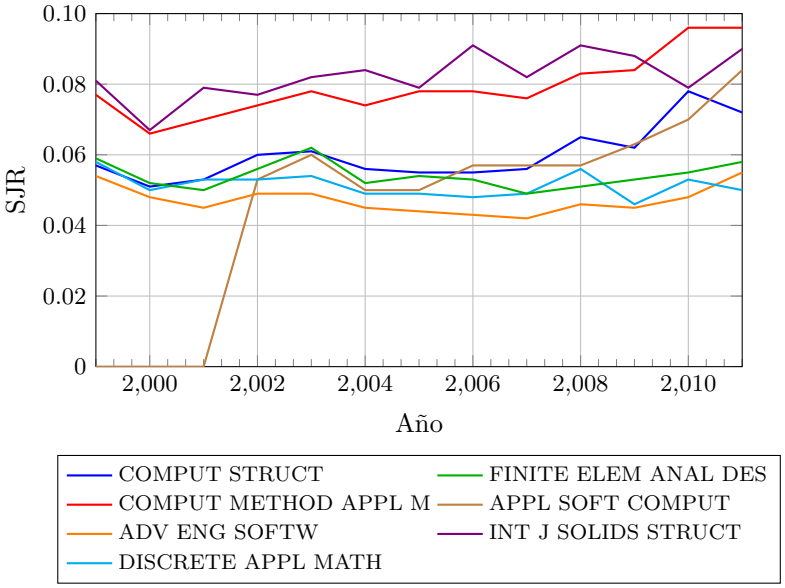


Figura 1.7: Evolución temporal del indicador SJR

Según el indicador SJR[2], que pondera el número de citas por documento durante los cuatro años anteriores, las revistas más interesantes son *International Journal of Solids and Structures*, *Computer Methods in Applied Mechanics and Engineering* y *Applied Soft Computing* por este orden.

[2]SCIMago Journal Rank

Tabla 1.3: Indicadores de calidad de las revistas mas relevantes

Revista	**ISSN**	**FI-2010**	**FI-5**	**Immediacy**	**Eigenfactor**	**Influence**	**Ranking**
Computers and Structures	0045-7949	1,719	1,846	0,244	0,01546	0,911	Q1/Q2
Computer Methods in ...	0045-7825	2,085	2,503	0,714	0,04039	1,263	Q1
Applied Soft Computing	1568-4946	2097	2,108	0,322	0,00457	0,511	Q1/Q2
Advances in Engineering ...	0965-9978	1,004	1,112	0,229	0,00382	0,409	Q3
International Journal of Solids ...	0020-7683	1,677	2,067	0,365	0,03618	0,931	Q1
Discrete Applied Mathematics	0166-218X	0,822	0,894	0,541	0,01347	0,149	Q2
Finite Elements in Analysis ...	0168-874X	1,03	1,348	0,596	0,00485	0,125	Q2/Q3

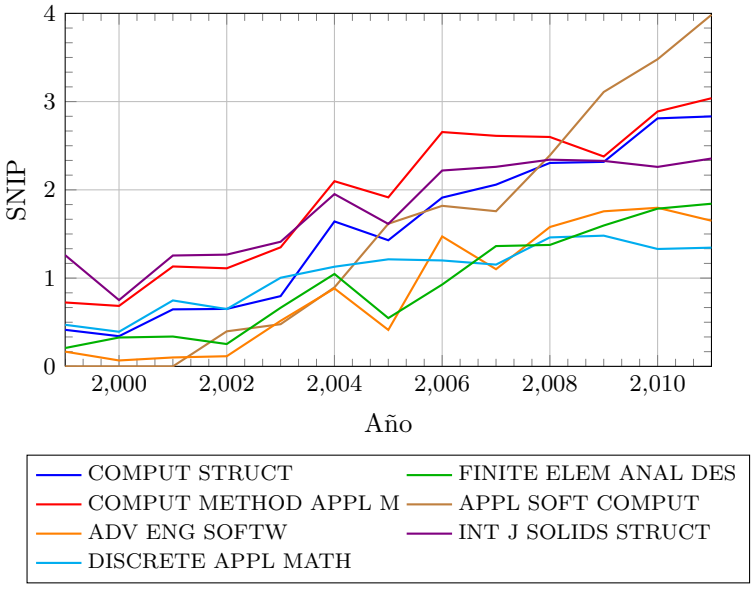


Figura 1.8: Evolución temporal del indicador SNIP

Sin embargo según el indicador SNIP, las revistas más interesantes son *Applied Soft Computing*, *Computer Methods in Applied Mechanics and Engineering* y *Computers and Structures*.

Como se puede apreciar, existe un factor común entre todos los indicadores formado por las revistas *Computer Methods in Applied Mechanics and Engineering* y *Applied Soft Computing*.

1.4.6 Principales grupos de investigación

La tabla 1.4 muestra a los autores mas prolíficos según los resultados obtenidos para los diversos patrones de búsqueda del segundo grupo, objeto de estudio en el presente trabajo. Como puede apreciarse no existen grupos importantes como en otras disciplinas, sino un conjunto de autores destacados. Esto es debido a que este tipo de investigación no requiere de laboratorios y equipos costosos, sino que puede realizarse de forma individual. Esto es bastante habitual dentro del campo de la optimización estructural. También es relativamente frecuente que los principales autores de este campo escriban de forma conjunta artículos o libros de investigación, generalmente de revisión.

Como también se aprecia en la citada tabla, muchos autores han realizado investigaciones empleando diversos métodos de optimización. Cabe señalar que no existe ningún autor español destacado en el campo de estudio del presente trabajo, solo existen unos pocos autores cuyos trabajos están relacionados con la optimización del tamaño de las estructuras.

Seguidamente se indican los autores más destacados en el campo de la optimización de estructuras en general:

Goldberg, David Edward Profesor y director del Illinois Genetic Algorithms Laboratory de la *University of Illinois* (USA).

Subramaniam D. Rajan Profesor y director del Computational Mechanics Lab de la *Arizona State University* (USA).

Hajela, Prabhat Profesor de Ingeniería Aeroespacial en el *Rensselaer Polytechnic Institute* (USA).

Pezeshk, Shahram y Camp, Charles V. Profesores de Ingeniería Civil en la *University of Memphis* (USA).

Adeli, Hojjat Profesor en la *Ohio State University* (USA).

Kicinger, Rafal Profesor en la *George Mason University* (USA).

Achtziger, Wolfgang Profesor en la *University of Dortmund* (Alemania).

Kaveh, Alí y Talatahari, S. Profesor de la *Iran University* e investigador de la *University of Tabriz* (Irán).

Ebenau, Garsten Ingeniero de la empresa *Hamelmann-Karvanek-Thierauf* (Alemania).

Balling, Richard J. Profesor de Estructuras y Optimización en la *Brigham Young University* (USA).

Ohsaki Profesor en la *Kyoto University* (Japón).

Topping, Barry H.V. Profesor emérito de Mecánica Computacional en la *Heriot-Watt University* (UK).

Tang, Wenyan Visiting Scholar en la *University of Sydney* (Australia).

Barbosa, Helio J.C. y Lemonge, Afonso C.C. Profesor en la *Universidade Federal de Juiz de Fora* (Brasil).

Erbatur, Fuat y Hasançebi, Oguzhan Profesor emérito en la *Middle East Technical University* (Turquía).

Papadrakakis, Manolis y Hasançebi, Oguzhan Profesores en la *National Technical University of Athens* (Grecia).

Jenkins, W.M. Profesor en la *University of Hertfordshire* (UK).

Greiner Sánchez, David y Emperador Azola, Jose Maria Investigadores en el *Instituto Universitario de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería de la Universidad de las Palmas de Gran Canaria* (España).

1.4.7 Conclusiones

A partir del estudio del estado del arte se puede concluir que en la actualidad no existe un único método para abordar el estudio planteado. El análisis de los artículos publicados en los últimos años parece indicar que las principales líneas de investigación actual se centran en los Algoritmos Genéticos, a cierta distancia la PSO, el SA y a mayor distancia la ACO. Este análisis demuestra que cuarenta años después de su invención los Algoritmos Genéticos se encuentran en plena vigencia, al menos en este campo.

Los resultados obtenidos sin embargo con estas técnicas no son igualmente satisfactorios en todos los casos. Un estudio más detallado del contenido de los trabajos publicados nos permite apreciar como los resultados obtenidos empleando PSO o SA se encuentran bastante lejos de los por los Algoritmos Genéticos, como muestran las tablas 6.1 y 6.2.

Gran parte del interés por la PSO y el ACO es debido a que requieren un número menor de evaluaciones de la función objetivo. Aun así se trata de técnicas incipientes en este campo.

En base a un análisis similar al descrito, se decide al inicio del presente trabajo en 2008 utilizar esta técnica como método de optimización topológica.

Tabla 1.4: Autores mas prolíficos en la optimización topológica de estructuras

	Total	**GA**	**MP**	**SA**	**PSO**	**ACO**
Primer grupo	Stolpe, M.	Kaveh, A.	Kaveh, A.	Cagan, J.	Kaveh, A.	Kaveh, A.
	Achtziger, W.	Talatahari, S.	Talatahari, S.	Shea, K.	Talatahari, S.	Talatahari, S.
	Guo, X.	Lin, Chun-Yi	Lin, Chun-Yi	Lin, C.-Y.	Liu, F.	Lin, Chun-Yi
	Cheng, G.	Luh, G.	Luh, G.	Luh, G.	Aravinthan, T.	Luh, G.
	Bureerat, S.	Shahrouzi, M.	Shahrouzi, M.	Lamberti, L.	Awad, Ziad K.	Shahrouzi, M.
	Fan, Z.	Aravinthan, T.	Aravinthan, T.	Baumann, B.	Dogan, E.	Aravinthan, T.
	Fleury, C.	Awad, Ziad K.	Awad, Ziad K.	Chiandussi, G.	Erdal, F.	Awad, Ziad K.
	Gengdong, C.	Dogan, E.	Dogan, E.	Kost, B.	Gonzalez, F.	Dogan, E.
	Gui, L.	Erdal, F.	Erdal, F.	Luh, G.	Hasançebi, O.	Erdal, F.
	Jiang, J.	Gonzalez, F.	Gonzalez, F.	Luo, Y.	Li, L.	Gonzalez, F.
Segundo grupo	Kaveh, A.	Kaveh, A.	Guest, James K.	Erbatur, F.	Kaveh, A.	Kaveh, A.
	Hajela, P.	Ashida, F.	Xie, Y.M.	Lamberti, L.	Ahangaran, M.	Ahangaran, M.
	Ohsaki, M.	Erbatur, F.	Ananthasuresh, G.K.	Liu, Xiaojian	Alinia-ziazi, A.	Bureerat, Sujin
	Xie, Y.M.	Farajpour, I.	Ashida, F.	Seepersad, C.C.	Bureerat, Sujin	Farajpour, I.
	Allen, J.K.	Hajela, P.	Farajpour, I.	Alinia-ziazi, A.	El Alem, W.	Farhoudi, N.
	Ashida, F.	Kravanja, S.	Hajela, P.	Barron, Sarah	El Hami, A.	Fayazbakhsh, K.
	Bruns, T.E.	Lamberti, L.	Hinton, E.	Barthelat, F.	Ellaia, R.	Fujiwara, J.
	Bureerat, S.	Liu, X.	Hira, A.	Begg, D. W.	Farajpour, I.	Ghiasi, H.
	Erbatur, F.	Luo, Zhen	Kravanja, S.	Beom, H.G.	Farhoudi, N.	Hinduja, S.
	Farajpour, I.	Ohsaki, M.	Liu, Xiaojian	Beyer, H.	Farkas, J.	Lamberti, L.

1.5 Organización de la tesis

La tesis se organiza, de forma similar al desarrollo de la investigación, en torno a seis capítulos:

- El primer capítulo, el presente, introductorio.
- El segundo capítulo detalla los objetivos y el plan de trabajo seguido en el desarrollo de la misma.
- El tercer capítulo describe los principales trabajos sobre de optimización estructural que se han publicado hasta el momento y plantea el problema de optimización.
- El cuarto capítulo detalla el método de los Algoritmos Genéticos así como los operadores y parámetros fundamentales de ajustes de los mismos ya que estos conocimientos son necesarios para abordar el siguiente capítulo.
- El quinto capítulo describe la implementación de un nuevo Algoritmo Genético libre de los condicionantes descritos en este capítulo.
- El sexto capítulo realiza la validación del nuevo algoritmo mediante su prueba contra uno de los problemas de optimización clásicos: la optimización de la estructura de diez barras y seis nodos, logrando mejorar los resultados existentes.
- El séptimo capítulo detallan las conclusiones obtenidas, en base al capítulo anterior.
- El octavo capítulo describe las futuras líneas de investigación del presente trabajo.
- Los diferentes anexos detallan los parámetros, cálculos y procedimientos adicionales, empleados en el cálculo o el desarrollo del nuevo algoritmo. También detalla el código de cálculo de la solución óptima en lenguaje APDL de Ansys para su verificación. Finalmente aparece la clase en C++ fundamental para la implementación del algoritmo con el fin de verificar y entender su funcionamiento. No se detallan todas las otras clases adicionales desarrolladas debido a que su extensión total es superior a la propia tesis. El autor pone a disposición del que lo solicite el código fuente completo a través del correo electrónico `sasanca@dimm.upv.es`sasanca@dimm.upv.es

2

Objetivos

2.1 Objetivos

El objetivo principal de la tesis es *desarrollar un algoritmo eficiente y robusto que sea capaz de generar un conjunto de estructuras óptimas en base a las cargas que estas tengan que soportar y los apoyos que las sustenten, sin partir de ninguna forma, geometría, regla o modelo preconcebido para las mismas, sujetas a ciertas restricciones de resistencia y rigidez.*

La necesidad de desarrollar dicho algoritmo se fundamenta en la siguiente hipótesis:

> La definición previa de la forma, geometría, regla o modelo preconcebido en una estructura suponen restricciones del diseño en sí mismas y por lo tanto aquel algoritmo que no se encuentre sujeto a estas deberá poder generar diseños necesariamente mejores, o al menos tan buenos como los existentes.

Dicha hipótesis se sustenta en las indicaciones de un trabajo previo realizado por Rajan [310], el cual constituye el primer aporte relevante de los Algoritmos Genéticos a la optimización de tamaño y topología simultáneas. Dado que todos los algoritmos de optimización existentes hasta la fecha requieren de la definición de una topología previa, si se logra cumplir este primer objetivo, será posible diseñar estructuras más óptimas a las existentes en la actualidad.

Por otra parte, al plantear el diseño de estructuras de una forma totalmente abierta, se logrará otro objetivo secundario y no por ello menos importante como es *lograr la generalización de la topología óptima de las estructuras analizadas.*

La implementación del algoritmo en una aplicación informática permitirá alcanzar además otro objetivo: *el desarrollo de una herramienta que permita a los diseñadores obtener un conjunto de estructuras óptimas, sin que se requiera el conocimiento de una estructura previa, en un periodo de tiempo razonablemente corto.*

2.2 Planificación de la investigación

Con el fin de alcanzar los objetivos planteados se han seguido una serie de pasos reflejados a continuación:

Estudio del estado del arte

El primer paso para abordar el presente trabajo fue estudiar el estado actual de la técnica de optimización de estructuras con el fin de determinar el método de optimización más adecuado para lograr los objetivos propuestos. Para ello se realizó un estudio exhaustivo donde se procedió a revisar los trabajos publicados relacionados con la optimización de estructuras.

Para lograr los objetivos de la tesis fue necesario recabar mas de 600 referencias, de las cuales sobre 426 aparecen reflejadas en este trabajo. A diferencia de otras disciplinas, gran parte del trabajo publicado, sobre todo en lo referente a la investigación básica, se encuentra en informes técnicos, congresos, proceedings, etc...

Como conclusión a este estudio, se puede afirmar que el método de los Algoritmos Genéticos es uno de los más empleados en la optimización de estructuras y previsiblemente lo seguirá siendo a falta del desarrollo de nuevos métodos.

En este estudio se ha podido constatar como muchos artículos presentan carencias de tipo teórico incluso en la actualidad. El presente trabajo, sin pretender ser un manual sobre el tema, describe en el tercer capítulo toda la base teórica necesaria para el desarrollo del algoritmo descrito y validado en los capítulos sucesivos. Esta descripción, aunque más extensa que muchos libros relacionados con el tema, no aborda de forma deliberada aquellos contenidos teóricos no empleados en el presente trabajo.

Estudio de las librerías de Algoritmos Genéticos

Tras concluir el apartado anterior el siguiente paso fue estudiar el conjunto de aplicaciones y librerías existentes en la actualidad con el objetivo de determinar si debía partirse de cero o podía aprovecharse parte del trabajo existente.

La cantidad de aplicaciones de sofware libre es muy importante, siendo la más destacable de todas, con notable diferencia, la librería GAlib que Mattew Wall

desarrolló en el MIT. Esta librería fue desarrollada por el autor, dentro de un proyecto del MIT, a partir de una librería previa escrita por Kazu Saito en el MIT CADlab. Mas tarde el autor emplearía esta librería para el desarrollo de su tesis doctoral [404]. Desde entonces numerosos doctorandos en todo el mundo han desarrollado sus tesis a partir de esta librería.

El lenguaje de programación de la librería es el C++, lo cual le proporciona toda la versatilidad y garantías de eficiencia de este lenguaje de programación. El paquete dispone de la capacidad de compilar toda la librería en una librería estática por lo que esta puede ser llamada desde cualquier lenguaje de programación si se estudian sus procedimientos.

El lenguaje empleado para el desarrollo de la aplicación informática que implementa el algoritmo desarrollado en el presente trabajo ha sido también el C++, empleando el compilador de C++Builder para la compilación. La ejecución del programa no contiene ninguna GUI gráfica ya que se ha pretendido primar la versatilidad y portabilidad a otros sistemas operativos.

El código desarrollado es compatible con ANSI C++ y POSIX por lo que puede ser compilado en diferentes sistemas operativos. El ejecutable resultante ha sido probado con éxito en *Linux Ubuntu x64* y *Windows 7 x64*.

Definición e implementación del algoritmo

Una vez concluidos los pasos anteriores se procedió a definir e implementar el algoritmo. El algoritmo aquí descrito es el resultado de numerosas pruebas, no descritas en el presente trabajo. El proceso ha sido iterativo, retroalimentado tras cada fracaso por nuevos artículos de investigación básica, hasta configurar el algoritmo definitivo. Sin esta investigación básica y un conocimiento completo de la propia programación de la librería GAlib este trabajo no habría sido posible.

Durante las etapas iniciales de este proceso se trabajó con un genoma único de codificación entera, con todas las limitaciones que ello supone y que se detallan en los capítulos posteriores. También se emplearon exclusivamente los operadores genéticos proporcionados por la librería GAlib, con las limitaciones derivadas del empleo de la codificación entera.

Finalmente el Algoritmo Genético definitivo se configuró mediante una codificación mixta y el empleo de diversos operadores genéticos no implementados por GAlib, obteniéndose unos resultados bastante satisfactorios.

Validación del algoritmo

Finalmente se procedió a validar[1] en algoritmo en base a un problema clásico de optimización de estructuras: la estructura de diez barras y seis nudos, obteniéndose como un resultado un peso sensiblemente inferior al mejor resultado publicado hasta la fecha.

[1] En los países de influencia anglosajona se prefiere emplear el término *verificar* cuando se comparan los resultados entre modelos teóricos, mientras que el término *validar* se emplea al comparar modelos teóricos con resultados experimentales. No obstante, en este trabajo, se ha optado por el término *validar* porque el término *verificar* no se encuentra todavía extendido en el ámbito científico hispano.

3

Estado del arte de la optimización de estructuras

3.1 Breve reseña histórica

El primer trabajo destacable sobre optimización ya estaba relacionado con la optimización de estructuras y fue realizado por Galileo Galilei(1564-1642) en su obra, publicada en 1638, *Discorsi e dimonstrazioni matematiche, intorno, a due nuove scienze attenenti alla mecanica et i movimenti locali* [125]. Este versaba sobre la forma óptima de una viga en voladizo, con una carga puntual en su extremo libre.

Más tarde, el desarrollo del cálculo infinitesimal de Leibniz(1646-1716) y el cálculo de variaciones de Lagrange(1736-1813) sentarían las bases de la optimización de funciones moderna.

Posteriormente llegaría el *principio de mínima acción* de Hamilton (1808-1865).

Ya en el siglo XX Michell [268] establecería los principios fundamentales para el diseño óptimo de barras de peso mínimo. Para ello se valió de un teorema previo desarrollado por Clerk Maxwell [256], estableciendo un nuevo teorema según el cual:

> "Una estructura alcanza el límite absoluto en la economía de material si el espacio ocupado por esta puede ser sometido a una deformación pequeña apropiada tal que la deformación unitaria de todas las barras de la estructura se incrementa no menos que el cambio de longitud relativo de cualquier elemento en el espacio".

Las estructuras de Michell, sin embargo, presentaban el problema de ser siempre estáticamente determinadas, y generalmente con un gran número de barras, por lo que en muchos casos son impracticables desde el punto de vista práctico.

Este trabajo fue analizado y discutido por otros, destacando los trabajos de Cox [74], Owen [291], y Parkes [294]. Chan [59] desarrolló un conjunto de técnicas para la construcción gráfica de los campos de deformaciones unitarias descritos por Michell. Prager [302] desarrollaron un conjunto de técnicas para mejorar la eficiencia de estructuras cercanas al óptimo y definió un nuevo criterio de diseño para los elementos en una estructura de Michell discretizada. Posteriormente E.W. y Parkes [111] de forma simultánea a Prager [303] introdujeron el efecto de la rigidez de las uniones en las estructuras.

En 1957 Barta [25] publica una investigación para determinar los conjuntos de barras redundantes con el fin de probar el teorema de Sved [380] según el cual *mediante la eliminación adecuada de las barras redundantes de una estructura es posible obtener una estructura estáticamente determinada con el mínimo peso* para un estado de carga dado. Pearson [295] empleó un generador de números aleatorios para variar los elementos redundantes hasta alcanzar soluciones óptimas, advirtiendo que *solo las estructuras estáticamente determinadas carecen de barras con secciones nulas.*

Prager y Rozvany [304, 305, 325] desarrollaron una teoría para la distribución óptima de estructuras reticulares utilizando utilizando una analogía con la teoría de Michell. Estos trabajos representaron la primera aproximación a la modificación de la distribución de las estructuras y, a pesar de ser completamente diferente a la filosofía de los posteriores trabajos de programación matemática, sirvieron para el desarrollo de estos.

3.2 Técnicas de Optimización de Estructuras

A continuación se enumeran las diferentes técnicas de optimización de estructuras empleadas en los últimos cuarenta años. No se procede a realizar una descripción exhaustiva de ellas ya que el alcance y la extensión de esta tarea excede los límites del presente trabajo. Simplemente se detallan los trabajos más relevantes relacionados con la optimización de estructuras.

3.2.1 Programación matemática

Gracias al desarrollo de la programación matemática, en 1968 Hemp y Chan [60, 168–170], y en paralelo Dorn, Gomory y Greenberg [97], superaron los problemas de las estructuras de Michell mediante el desarrollo de una malla de puntos (*Ground Structure*) que incluía las uniones estructurales, los apoyos y los puntos de aplicación de carga. Demostraron que si una estructura articulada sujeta a un campo de deformaciones virtuales cumple con las restricciones impuestas, entonces esta es óptima. Los desplazamientos virtuales de los diversos puntos fueron modificados utilizando técnicas de programación lineal donde la función objetivo estaba formada por los trabajos virtuales de las barras. Algunas de las barras menos cargadas y sus nudos eran eliminadas hasta llegar a estructuras determinadas o con

un grado de indeterminación reducido, logrando que las barras restantes alcanzaran la máxima deformación virtual. Demostraron que, para estructuras sujetas a un simple estado de carga, la forma óptima era estáticamente determinada.

En 1964 Fleron [116] desarrolló un proceso iterativo para la reducción de peso donde cada paso se correspondía con una subestructura estáticamente determinada, llegando a dos conclusiones para estructuras sujetas a un único estado de carga donde las tensiones admisibles para cada miembro pueden ser diferentes:

1. Si una estructura tiene menor peso que las demás, esta estructura será necesariamente estáticamente determinada.
2. Si una estructura óptima tiene un grado de indeterminación p, existen al menos $p+1$ estructuras óptimas estáticamente determinadas.

Más tarde Topping demostraría que si la malla de puntos inicial incluía un número suficiente de puntos, era posible encontrar múltiples topologías para un mismo óptimo, lo cual permitía elegir entre diferentes soluciones [23, 394].

Dorn, Gomory and Greenberg [97] concluyeron que el tamaño de la malla tenía un efecto significativo en la reducción de peso y la topología final. Para evitar este inconveniente procedieron a incorporar las coordenadas nodales como variables de diseño dentro del proceso de optimización, abriendo un nuevo camino en la optimización de estructuras.

A partir del trabajo de Dorn, Gomory and Greenberg las técnicas de programación matemáticas se clasificaron en tres grupos:

1. Las técnicas basadas en una malla de puntos denominadas técnicas de *Ground Structure*, donde se eliminan las barras de la estructura.
2. Las técnicas donde las coordenadas y las propiedades de las secciones se introducen como variables de diseño denominadas técnicas geométricas.
3. Los métodos híbridos donde se realizan ambas tareas a la vez. El presente trabajo se enmarca dentro de estos últimos.

Técnicas basadas en una malla de puntos o Ground Structure

La técnica mas sencilla es el *método del diseño completamente esforzado* o *Fully Stressed Design Stress-Ratio* donde en cada iteración las secciones de los elementos estructurales son redimensionados en base a la proporción entre la tensión que soporta cada miembro y su tensión admisible:

$$A_i(t) = A_i(t-1)\frac{\sigma_i(t-1)}{\sigma_i^{\text{adm}}} \tag{3.1}$$

donde $A_i(t)$ y σ^{adm} son la sección y tensión admisible del elemento i en la iteración t, $A_i(t-1)$ y $\sigma_i(t-1)$ son la sección y la tensión del elemento i en la iteración $t-1$.

Mediante esta aproximación cuando la sección de una barra de la estructura se reduce a cero, se procede a eliminarla. Los resultados obtenidos mediante esta técnica para estructuras sometidas a un único estado de cargas, son similares a los obtenidos por Hemp y Chan [60, 168–170] anteriormente, y generalmente estáticamente determinados.

A principios de los 60, Pearson [295] y Chan [60] consideraron la posibilidad de que la estructura soportara diferentes estados de carga. En este caso Pearson [295], mediante un algoritmo estocástico, señaló que las estructuras tendían a ser estáticamente indeterminadas y que cada elemento era sometido a la carga máxima, al menos en un estado de cargas. Chan [60] llegó a una conclusión similar empleando técnicas de programación lineal.

Más tarde, Lev [231], Schmidt [335], Hemp [169], Spillers y Lev [366] y Rozvany [325] corroboraron estas conclusiones.

Sin embargo el trabajo de Topping [394] demostró que los resultados obtenidos con el *Fully Stressed Design Stress-Ratio* no son siempre óptimos. Esto es debido a que el método no considera la compatibilidad de las estructuras resultantes y si la distribución resultante es indeterminada, la sección de los elementos no asegura la compatibilidad de la estructura sin sobretensionar alguno de los elementos. Para evitar este inconveniente Reinschmidt y Russell [316, 317, 326] formularon de forma exitosa una técnica iterativa que redimensionaba la distribución obtenida mediante los métodos de programación lineal y el *Fully Stressed Design Stress-Ratio*.

Con el fin de evitar el problema de la compatibilidad estructural y la incorporación de restricciones de desplazamientos Dobbs y Felton [94] emplearon un algoritmo basado en el método de la máxima pendiente. Posteriormente Hajela y Ashley [153] presentaron un algoritmo de búsqueda con una formulación de restricciones acumulativa.

Según Topping [395] las técnicas de *ground structure* son especialmente recomendables en el diseño de estructuras modulares, las cuales emplean elementos y sistemas estructurales estandarizados.

En 1992, Kirsch [210] demuestra que cuando la sección se aproxima a cero, las restricciones de tensión desaparecen repentinamente y generándose soluciones factibles degeneradas. Demostró que es difícil obtener una solución óptima ya que estas soluciones degeneradas pueden activar o desactivar las restricciones de tensión de forma abrupta, dificultando el proceso de optimización.

En 2009, Hagishita y Ohsaki [152] propusieron una variación del método al que llamaron *Growing Ground Structure Method* donde se añaden o eliminan barras en base a cinco estrategias propuestas.

Las técnicas geométricas

Con este enfoque, la función objetivo resultante se vuelve extremadamente alineal, ya que las longitudes de los miembros varían al variar las coordenadas de los nudos.

Uno de los primeros trabajos con este enfoque fue el de Schmit [336]. En este consideró las coordenadas de los nudos como variables al formular el clásico problema de optimización de tres barras. Para su resolución empleó un algoritmo no lineal que utilizaba el método de la máxima pendiente. En las conclusiones a dicho trabajo fue el primero en indicar que en una estructura estáticamente indeterminada, sometida a varios estados de carga, no todos los miembros están completamente esforzados al menos en un estado.

Posteriormente, Schmit y Morrow [341] introdujeron restricciones de pandeo y demostraron que cuando estas restricciones se encuentran activas las distribuciones óptimas son mas rígidas, pudiendo emplearse un material más ligero.

En 1963, Schmit y Mallett [338] publicaron un artículo donde los ángulos entre los elementos, las áreas y los materiales eran tratados como variables continuas.

El método consideraba restricciones de tensión y deformación junto con múltiples estados de carga. Para su resolución emplearon un algoritmo que utilizaba el método de la máxima pendiente, donde la dirección de búsqueda era determinada de forma aleatoria dentro del plano de peso constante. La inclusión de las variables mejoró el resultado óptimo y demostró que incluso bajo estados de carga múltiples este podía ser estáticamente determinado. También demostró que empleando diferentes estructuras iniciales era posible obtener diferentes diseños finales, de modo similar a lo que ocurre con las técnicas heurísticas.

En 1968, Sved y Ginos [381] investigaron el problema de tres barras estudiado previamente por Schmit [336], demostrando que era posible obtener el óptimo global eliminando una de las barras, violando la restricción de tensión para la misma. Sugirieron que *si hay una redundancia simple, es necesario buscar sistemáticamente todas las estructuras que pueden ser obtenidas de la original omitiendo un elemento*. En este trabajo no mostraron una técnica apropiada de búsqueda, pero demostraron que un algoritmo que no explorara la eliminación de cada uno de sus miembros no podía garantizar la obtención de un óptimo global.

En 1970, Corcoran [71, 72] replanteó el problema de tres barras de Schmit incluyendo las coordenadas horizontales de los tres nudos y las secciones de las barras como variables de diseño. Dos de los nudos coincidieron y la configuración óptima se convirtió en un problema de dos barras. Para resolver el problema empleó dos algoritmos no lineales: la técnica de proyección del gradiente y la técnica de minimización secuencial no restringida[1], teniendo mejor comportamiento el primero. Su enfoque supuso un avance y fue aplicado a un conjunto de estructuras demostrando que era posible obtener estructuras óptimas introduciendo las coordenadas de los nudos como variables, con ahorros significativos de material. El método permitía que los nudos coincidieran fusionándose, evitando los problemas de inestabilidad que aparecían cuando se eliminaban barras de la estructura.

Posteriormente Pedersen [296] publicó una técnica iterativa, basada en la programación lineal, para el diseño de estructuras planas sometidas a un estado de carga simple donde las áreas de los elementos y las coordenadas nodales fueron empleadas como variables de diseño. Al emplear un solo estado de cargas las soluciones obtenidas fueron estáticamente determinadas. El método empleaba un análisis

[1] Método de Powell

de sensibilidad de las derivadas parciales con respecto a las áreas, considerando también el peso propio, así como restricciones de tensión y pandeo.

En 1972, Pedersen [297] extendió su trabajo añadiendo múltiples estados de carga así como restricciones de deformación, aplicando su algoritmo a la estructura de un puente y una viga en voladizo. Un año más tarde añadió límites a las variables de diseño sin incrementar el tamaño del problema de programación lineal y extendiendo su trabajo al espacio [298]. En las conclusiones generales de su trabajo indicaba que en los miembros de la estructura donde las restricciones de pandeo estaban activas, tendían a ser más cortos y que las restricciones de desplazamiento activas tendían a extender la distribución óptima. Sin restricciones de desplazamiento los diseños óptimos a menudo estaban completamente tensionados. Sin embargo, si existía un conjunto de nudos fijos, estas afirmaciones no son siempre estrictamente ciertas aunque la diferencia en peso entre los diseños no fue significativa.

En 1973, Schmit y Farshi [337], optimizaron el problema de la estructura clásica de diez barras y seis nodos empleada en el apartado de validación de este trabajo.

En 1977, Thomas y Brown [388] emplearon una función de penalización combinada con una técnica de optimización secuencial no restringida para optimizar varios tipos de tejado. Los elementos que se aproximaban a la mínima tensión admisible eran eliminados, excepto los necesarios para mantener la estabilidad de la estructura. Una vez el elemento era eliminado, la estructura era reoptimizada y se volvía a estudiar la posibilidad de eliminar nuevos elementos.

En 1980, Saka [328] aplicó de forma exitosa a un conjunto de estructuras típicas, una técnica para el diseño de estructuras sujetas a múltiples estados de carga donde las áreas de los elementos y las coordenadas de los nudos fueron introducidas como variables de diseño. El método agrupaba los diferentes elementos por rigidez y teniendo en cuenta las restricciones de desplazamiento, tensión y pandeo. Además los desplazamientos de los nodos fueron introducidos como variables de diseño, evitando tener que analizar la estructuras. El algoritmo empleado linealizaba las restricciones y empleaba el método *simplex* de forma iterativa.

En 1981, Imai y Schmit [185] desarrollaron un método *primal-dual* al que denominaron método de los multiplicadores, el cual es una extensión del enfoque de la función de penalización cuadrática. Para evitar la elevada alinealidad de los problemas realizaron un desarrollo de polinomios de Taylor para los desplazamientos

a lo largo de la dirección de búsqueda. Se consideraron como restricciones la tensión, los desplazamientos y el pandeo de Euler. A diferencia de los anteriores, este algoritmo no necesita partir de una estructura inicial factible.

Posteriormente Lin et al. [238, 239] publicaron un algoritmo basado en las condiciones de Kuhn-Tucker para la optimización de una estructura sometida a restricciones de tensión y resonancia. Incluyeron como variables de diseño las coordenadas de los nudos así como las secciones de los elementos. Como conclusión establecieron que las estructuras analizadas eran más sensibles a las cargas dinámicas que a las estáticas.

En 1985, Haftka [150], Haftka y Kamat [151] emplearon el método del gradiente conjugado combinado con el método de Newton para minimizar la función de penalización.

En 1986, Ringertz [320] empleó un algoritmo de *Branch and Bound* sujeto a restricciones de tensión y desplazamiento. A partir de una *Ground Structure* se generaban y analizaban una serie de barras candidatas. Las secciones transversales y los desplazamientos se trataron como variables independientes por lo que fue posible que en determinadas barras estos tuvieran un valor nulo. El problema de optimización se resolvió empleando tres métodos diferentes, destacando la SQP.

En 2002, Missoum y Gürdal [270], Missoum et al. [271, 272] ampliaron el trabajo de McKeown y lo extendieron a problemas no lineales.

Técnicas híbridas

El diseño simultáneo de la topología el tamaño de la estructura conduce a un gran número de variables de diseño con diferentes dominios que dan lugar a un gran rango de sensibilidades. Además, como demostraron Sved y Ginos [381], el espacio de soluciones factibles suele ser disjunto, haciendo muy difícil la búsqueda del óptimo global. Para evitar estos problemas algunos investigadores dividen las variables de diseño en dos espacios de diseño mientras que otros solo las consideran a intervalos alternos del proceso de diseño.

En 1972, Vanderplaats y Moses [400] desarrollaron una técnica aplicada a estructuras sometidas a múltiples estados de carga donde emplearon el método del *Fully Stressed Design Stress-Ratio* para dimensionar la estructura mientras mantenía la

topología fija y posteriormente el *método de la máxima pendiente* para desplazar las coordenadas de los nudos hacia una posición óptima. El algoritmo empleaba ambas técnicas de forma consecutiva e iterativa. Como restricciones de diseño tomaron el rango de tamaños y tensiones para cada miembro así como el pandeo. Como conclusión a su trabajo afirmaron que el empleo de dos espacios de diseño separados pero independientes reducía sensiblemente el número de variables de diseño.

Más tarde extendieron sus investigaciones a estructuras con restricciones de desplazamiento [398, 399]. En esta ocasión se empleó del *método de las direcciones factibles* para optimizar la estructura con la topología fija.

En 1973, Kuan-Chen [217] desarrolló otro algoritmo para múltiples estados de carga donde empleó una técnica que combinaba una búsqueda basada en el gradiente con una aleatoria. Investigó las propiedades de una estructura moviendo uno de los nudos y demostró que la función objetivo en ese caso era convexa.

En 1974, Lipson y Agrawal [241] emplearon el *método complex* para diseñar estructuras sometidas a restricciones de tensión y múltiples estados de cargas. Emplearon como variables las coordenadas de los nudos y las secciones de los elementos. Posteriormente [242, 243] añadieron restricciones de deformación. Permitieron la eliminación de elementos de la estructura cuando la fuerza aplicada sobre los mismos tendía a cero y no generaba inestabilidad en la estructura. Emplearon el método del *Fully Stressed Design Stress-Ratio* junto a una técnica de escalado del desplazamiento mientras mantenían la geometría constante y el *método complex* mientras fijaban la topología. Concluyeron que para emplear el *método complex* era conveniente separar las variables de diseño y solo considerar la sección de los miembros o las coordenadas de los nudos cada vez.

En 1975, Spillers y Friedland [367–369] desarrollaron otra serie de técnicas para estructuras determinadas e indeterminadas estáticamente. Derivaron un conjunto de expresiones basadas en el gradiente de la función objetivo respecto a las coordenadas nodales y las áreas de los elementos de la estructura. Emplearon un esquema iterativo basado en el método de newton y las condiciones de Kuhn-Tucker manteniendo dos espacios de diseño separados.

En 1980, Spillers y Kountouris [370] emplearon las fórmulas de tensión admisible de la AISC para definir una relación entre el peso/longitud con la carga segura de cada sección.

En 1981, Lev [232] planteó un enfoque heurístico al problema de optimización geométrica. Demostró que las soluciones obtenidas mediante el software de Spillers and Vanderplaats se estancaban en óptimos relativos. Sugirió que en la optimización de estructuras sometidas a un único estado de cargas, primero se debía optimizar la topología empleando el *Fully Stressed Design Stress-Ratio*. También sugirió que para estructuras sometidas bajo estados de carga múltiples, la topología inicial debía estar compuesta de una combinación de las topologías óptimas para los diferentes estados de carga simples.

En 1989, y posteriormente en 1998 McKeown [259] introdujo el peso de las uniones en la formulación del problema. El problema se formulaba en dos fases, el problema interior y el exterior. Durante la optimización del problema interior se optimizaba el coste sujeto a las condiciones de equilibrio. En el problema exterior, se calculaban los desplazamientos que minimizaban la función de coste sometida a las restricciones de tensión y desplazamiento.

A diferencia de los anteriores, los siguientes autores utilizaron una malla de puntos en sus algoritmos, alternando la optimización de tamaño y topología.

En 1970, Reinschmidt y Russell [316, 317, 326] utilizaron una malla de puntos sobre la que aplicaron diversas técnicas de programación lineal y el *Fully Stressed Design Stress-Ratio* de forma sucesiva asegurando el cumplimento de las restricciones de tensión y desplazamiento, obteniendo resultados más satisfactorios que con el empleo del *Fully Stressed Design Stress-Ratio*.

En 1972, Sheu y Schmit Jr. [351] basaron su método en la comparación del límite superior de la configuración obtenida mediante la técnica de la dirección factible y el límite inferior obtenido mediante el algoritmo *primal-dual* con la restricción de compatibilidad relajada.

En 1974, Farshi y A. [112] desarrollaron un método para evitar los problemas derivados de un espacio de soluciones factibles disjunto. En este método introdujeron como variables de diseño las fuerzas de los elementos redundantes y las áreas de todos los elementos, incluyendo en las restricciones los límites máximo y mínimo

para los tamaños de los elementos. Inicialmente resolvieron el problema empleando el *método simplex* obviando la restricción de compatibilidad, introduciendo posteriormente los requerimientos de compatibilidad en forma de restricción.

En 1977, Lev [230] analizó como diseñar estructuras estáticamente determinadas sometidas a dos estados de carga. Para ello partió de una superposición de las topologías óptimas para cada estado de cargas, lo cual generalmente genera una estructura estáticamente indeterminada. La estructura es convertida en una estructura estáticamente determinada mediante un proceso similar al *método simplex* donde se determina en que orden deben eliminarse los diferentes miembros de la estructura. Este método proporciona un conjunto de óptimos estáticamente indeterminados con un grado de indeterminación decreciente.

En 1974, Majid y Elliott [249–251] desarrollaron una técnica que empleaba el método de la máxima pendiente para optimizar el peso de una estructura sometida a diferentes estados de carga, considerando restricciones de tensión y deformación. Emplearon los *Teoremas de Variación Estructural*, calculando la estructura una sola vez en todo el proceso. Los análisis posteriores obtenían los resultados mediante la aplicación de los mencionados teoremas combinados con un conjunto de coeficientes de influencia aplicados en los extremos de cada miembro. Durante el proceso de optimización inicialmente se mantuvo la topología fija, empleando los teoremas y coeficientes de influencia con el fin de determinar en que orden debían eliminarse los elementos de la estructura.

Posteriormente Majid y Saka [252, 329] ampliaron los teoremas formulados por Majid y Elliott para el análisis de estructuras no articuladas. Emplearon el *método del plano de corte* como método de optimización, eliminando de la estructura aquellos elementos cuya sección se aproximaba a cero.

En 1982, Kirsch [209] empleó un algoritmo en el que la condición de compatibilidad era ignorada durante la optimización de la geometría, reduciendo el número de variables de diseño al emplear un procedimiento de enlazado de variables.

En 1989, Ringertz [321] formuló un diseño de peso mínimo en estructuras con comportamiento no lineal de dos modos distintos. Bien todas las ecuaciones de equilibrio o bien unas pocas de ellas fueron tomadas como restricciones. En ambos casos las variables de diseño y los desplazamientos fueron tomados como variables.

En 1991, Bendsøe et al. [41, 43] propusieron dos métodos para el diseño de estructuras con la máxima rigidez, dentro de un volumen prescrito. Las variables empleadas fueron las secciones transversales y los desplazamientos nodales.

En 1992, Achtziger et al. [5] y más recientemente Bendsøe y Sigmund [42] reformularon el diseño estructural minimizando el grado de violación y empleando las secciones de las barras como variables de diseño y el volumen total como restricción. De este modo redujeron el problema a un problema de Programación Lineal equivalente con los desplazamientos nodales como variables de diseño. En ese mismo año, Orozco y Ghattas [287] y más recientemente [288] publicaron un método SQP con Hessiano reducido.

En 1994, Kirsch y Rozvany [211] presentaron varias formulaciones alternativas que incluían un criterio de optimalidad (OC).

En 1995, Larsson y Rönnqvist [222] emplearon el método del Lagrangiano aumentado para evitar la degeneración causada por en proceso iterativo en la función de penalización.

En 1996, Achtziger [1] introdujo el cálculo con barras con diferentes propiedades a la tracción y a la compresión.

Otros trabajos destacables posteriormente son:

Achtziger [2, 3], Bendsøe y Sigmund [42] emplearon la Programación Lineal Secuencial (SLP). Tin-Loi [389, 390, 391] empleó el método del Gradiente Reducido Generalizado (GRG). Schulz y Bock [343], Dreyer et al. [99], Ohsaki y Swan [281], Stolpe y Svanberg [376], Schulz [342], Wang y Arora [406] emplearon diversos métodos de Programación Cuadrática Secuencial (SQP). Ben-Tal y Roth [40], Jarre et al. [191], Maar y Schulz [247], Herskovits [173], Herskovits et al. [174], Hoppe et al. [181] desarrollaron algoritmos basados en el método del punto interior. Achtziger y Stolpe [4], Ohsaki y Katoh [280] emplearon el método de *Branch and Bound* (BB) para resolver el problema con secciones transversales con valores discretos. Cheng y Guo [66], Guo y Cheng [148], Guo et al. [149], Stolpe y Svanberg [375] emplearon el método ϵ-relajado.

3.2.2 Técnicas metaheurísticas

Las técnicas metaheurísticas, a diferencia de las técnicas de optimización tradicionales, no siguen unos métodos o reglas preestablecidas de búsqueda. A pesar de no seguir ningún tipo de procedimiento deductivo son capaces de proporcionar soluciones buenas en un periodo de tiempo razonablemente corto. El principal inconveniente de este tipo técnicas es que no garantizan la localización del óptimo absoluto, por lo que para tener la certeza de haberlo obtenido (o al menos un punto muy próximo) debe ejecutarse el algoritmo varias veces.

Estos métodos son de aplicación cuando no existe un algoritmo específico y determinístico para la resolución del problema que asegure la obtención del óptimo absoluto.

Durante las últimas décadas han aparecido, y con seguridad seguirán apareciendo, innumerables técnicas de las cuales destacan, en el campo de la optimización de estructuras las descritas a continuación.

3.2.2.1 Recocido Simulado (SA)

El recocido simulado es un algoritmo estocástico cuya idea procede de la física estadística. Este intenta imitar el proceso de recocido de un metal líquido lentamente hasta su estado sólido. Si el enfriamiento es suficientemente lento, las moléculas se organizan de modo que la función de energía alcanza un mínimo global. Si por el contrario es demasiado rápido la función de energía alcanza un mínimo local.

Esta técnica sigue esta estrategia relativamente simple. Se toma un diseño de prueba, generado normalmente de forma aleatoria, y se evalúa la función objetivo para ese punto. Si el diseño de prueba no es factible este es descartado y se genera un nuevo diseño. Si el punto de prueba es factible y la función objetivo proporciona un resultado mejor que el mejor resultado obtenido hasta el momento, el diseño es aceptado y se anota el resultado como nuevo récord. Si el diseño de prueba es factible pero la función objetivo arroja un resultado peor que el mejor resultado obtenido, entonces el diseño es aceptado o rechazado en función de un criterio probabilístico que estima si el diseño puede mejorar en las siguientes iteraciones. El cálculo de la probabilidad de aceptación se realiza en función de un parámetro denominado *Temperatura*, por analogía con el citado proceso. Esta temperatura establece el umbral de aceptación, e inicialmente toma un valor grande para ir

decreciendo conforme avanza el proceso iterativo, siguiendo una ley similar a la ley de enfriamiento del metal. De este modo la probabilidad de selección decrece con la temperatura hasta hacerse nula.

El SA ha sido ampliamente estudiado en problemas de optimización de estructuras por su simplicidad y facilidad para encontrar el óptimo global incluso con un gran número de variables de diseño. La utilidad de esta técnica quedó demostrada con los trabajos clásicos de Balling [20] y Bennage y Dhingra [44].

Durante los últimos quince años se han publicado numerosos trabajos relacionados con la optimización de estructuras, relacionados con esta técnica, de entre los que destacan:

En 1997, Shea et al. [349, 350] desarrollaron un método de diseño de barras basado en una combinación de SA y reglas de transformación geométrica y lo aplicaron al diseño de cerchas. A diferencia del algoritmo tradicional donde la probabilidad de selección depende de un valor predefinido, se utilizó un método adaptativo.

Posteriormente, Shim y Manoochehri [354] resolvieron un conjunto de problemas de optimización de tamaño realizando una linealización de las restricciones de tensión mediante esta técnica. Para asegurar que los diseños fueran factibles, añadieron un factor corrector a los términos dependientes de la tensión basado en el gradiente máximo de la tensión linealizada.

En el 2000, Pantelides y Tzan [292] emplearon este método para optimizar diversas estructuras sometidas a cargas dinámicas. El algoritmo incluía un análisis de sensibilidad para identificar que variables de diseño tenían más influencia en las deformaciones globales. Cada vez que un nuevo diseño establecía un nuevo récord se modificaba el dominio del problema con el fin de que los nuevos diseño generados aleatoriamente estuvieran próximos a la región del mejor diseño.

En 2002, Hasançebi y Erbatur [166] demostraron la posibilidad de realizar una optimización simultánea de tamaño y topología empleando este método empleando el esquema tradicional de funcionamiento del SA.

En este mismo año, Chen y Su [63] apuntaron que la técnica tradicional SA requiere de demasiados análisis para converger hasta el diseño óptimo. Este hecho es un hándicap ya que el coste computacional de los análisis suele ser elevado. Con el fin de mitigar el inconveniente propusieron dos mejoras a la formulación clásica:

estimar las regiones del espacio de diseño factible a partir de la linealización de las restricciones e iniciar el proceso de optimización a partir de un diseño factible.

En 2005, Erdal y Sonmez [107] utilizaron un conjunto de geometrías predefinidas en lugar de un solo punto. Cada vez que un nuevo diseño era mejor que el peor de dicho conjunto, este era sustituido por el nuevo. Aplicaron en algoritmo al diseño de laminados en materiales compuestos.

En 2008, Lamberti [219] empleó una variación del método denominada *Corrected Multi-Level and Multi-Point Simulated Annealing* (CMLPSA). Al igual que el método de Erdal, este utiliza una población de individuos. En este algoritmo los nuevos puntos se generaron perturbando todas las variables de diseño de forma simultánea o local, realizando una búsqueda global o local en función de la tendencia del proceso de optimización observada.

En 2011, Noilublao y Bureerat [277] aplicaron una variación del método denominada *Archived Multiobjective Simulated Annealing* (AMOSA) a una torre esbelta tridimensional así como a otros casos mas sencillos. En este trabajo empleó otras técnicas como *Strength Pareto Evolutionary Algorithm* (SPEA2) y *Population-Based Incremental Learning* (PBIL).

El elevado coste computacional del método, acompañado del hecho de que a diferencia de otros métodos como GA, ACO, PSO y HS, este método solo proporciona una solución óptima, ha provocado que los investigadores estén perdiendo interés por el mismo en detrimento de otros métodos de optimización heurística, como se ha podido apreciar en el estudio previo recogido en el apartado apartado 1.4.

3.2.2.2 Computación evolutiva

La computación evolutiva nace como consecuencia de los inconvenientes y limitaciones de los métodos más formales como la *Programación Matemática* o el *Método del Criterio de Optimalidad* [205] al ser aplicados a problemas complejos. Estas técnicas tradicionales requieren de funciones continuas y derivables. Tal situación no siempre es posible posible cuando se abordan problemas de optimización de estructuras.

Durante mediados de los 60 y los 70, un conjunto de investigadores de entre los que destacan Lawrence J. Fogel [118], Ingo Rechenberg [313, 314], y John Henry

Holland [178], empezaron a plantear de forma teórica la posibilidad de desarrollar algoritmos de optimización inspiradas en la naturaleza. Fue el último de estos el que acuñó por primera vez el término *Algoritmo Genético*.

Posteriormente, en 1992 Marco Dorigo [96] desarrolló en su tesis doctoral una nueva técnica a la que denominó *Optimización por Colonias de Hormigas* (ACO).

En 1995 Kennedy, Eberhart y Shi [204, 352] desarrollaron otra técnica conocida como *Optimización por Enjambre de Partículas*(PSO), la cual es una de las técnicas más prometedoras en en el futuro de la optimización de estructuras.

Mas recientemente han aparecido otras técnicas como la *Evolución Diferencial* (DE)[2] de Storn y Price [377] en 1997, la *Búsqueda Armónica* (HS)[3] de Geem et al. [127] en 2001, o el Gran *Estallido-Gran Crujido* (BBBC)[4] de Erol y Eksin [108] en 2006, todavía con un desarrollo muy incipiente respecto a la optimización de estructuras.

A continuación se detallan los trabajos más destacados de la optimización de estructuras donde se han aplicado con éxito estas técnicas.

Los Algoritmos Evolutivos

El primer tipo *Algoritmo Evolutivo* desarrollado fueron los Algoritmos Genéticos. Estos comenzaron a aplicarse de forma práctica a mediados de los 70 y principios de los 80. Fue precisamente David Goldberg, un discípulo de Holland quien alcanzó mayor éxito a este respecto. Según parece tras asistir a una de los seminarios de Holland, Goldberg que estaba interesado en encontrar el diseño óptimo de líneas para el transporte de gas, le planteó la posibilidad de emplear los Algoritmos Genéticos con este fin, a lo cual Holland le replicó que tal aplicación era excesivamente compleja. Poco tiempo después logró llevar a cabo dicha tarea en su tesis doctoral [133, 134].

Posteriormente publicaría un libro [135] que se convertiría un hito en el campo de la optimización mediante Algoritmos Genéticos y que recogía numerosas aplicaciones, muchas de ellas relacionadas con la optimización de estructuras, como el clásico

[2]Differential Evolution
[3]Harmony Search
[4]Big Bang Big Crunch

problema de optimización de diez barras y seis nudos, aunque sin la restricción de desplazamientos del problema original.

Los Algoritmos Genéticos trabajan con poblaciones de individuos (soluciones) y se basan en los conceptos Darwinianos de supervivencia de los individuos más aptos. Los diseños se representan en forma de cadenas numéricas generalmente llamadas *cromosomas* donde cada elemento de la cadena recibe en nombre de *gen*[5]. Los cromosomas son generados y modificados siguiendo los mecanismos naturales de la evolución: la reproducción, el cruce y la mutación. Después de varias generaciones, los diseños representado por los individuos más aptos representan los diseños óptimos.

El éxito de esta técnica de optimización en todos los ámbitos del saber ha sido incuestionable, superando en muchos casos al resto de técnicas de optimización. Una simple búsqueda de los términos *Genetic Algorithm* en *SciVerse Hub* arroja más de 361.000 resultados frente a los casi 212.000 que devuelve los términos *Mathematical Programming* que no representa una única técnica sino un conjunto de técnicas mucho más antiguas que la primera.

A diferencia de otras técnicas como la Programación Matemática, el planteamiento de los problemas de optimización de estructuras realiza la optimización simultánea del tamaño y topología de las mismas ya que esto no supone un problema añadido para este tipo de algoritmos. Prácticamente la totalidad de los artículos comentados a continuación realizan este tipo de optimización simultánea. No obstante, el planteamiento fundamental a la hora de codificar las variables de diseño es heredado de la MP empleando técnicas basadas en *Ground Structure* o híbridas fundamentalmente.

A continuación se detallan los trabajos más importantes relacionados con la optimización de estructuras y los Algoritmos Genéticos.

En 1982, Lawo y Thierauf [223] intentaron optimizar el diseño de una estructura sometida a cargas dinámicas mediante una búsqueda aleatoria. Aunque el algoritmo empleado no fue un Algoritmo Genético, este trabajo representa el primer intento por resolver un problema de optimización de estructuras mediante una técnica heurística.

[5]Estas afirmaciones son de tipo general pudiendo variar en función de la codificación

En 1986, Goldberg y Samtani [140] publican la primera aplicación de los Algoritmos Genéticos a la optimización de estructuras al problema clásico de optimización de estructuras conocido como *the ten-bar and six node problem*[6] propuesto inicialmente por Venkayya et al. [401], en plena vigencia como banco de pruebas actualmente.

En 1990, Hajela y Shih [157] estudiaron la aplicabilidad de los Algoritmos Genéticos como método de optimización de estructuras.

En 1991, Jenkins [192] emplearía el código en Pascal de un Algoritmo Genético Simple publicado por Goldberg [135] para estudiar una cercha.

En 1992, Rajeev y Krishnamoorthy [311] emplearon un Algoritmo Genético binario para optimizar una estructura de tres barras y la clásica estructura de diez barras analizada por Goldberg, pero añadiendo las restricciones de desplazamiento del problema original. También aplicaron el algoritmo para optimizar una torre de transmisión de 160 barras.

Ese mismo año Hajela y Lin [156] emplearon un Algoritmo Genético binario que empleaba variables discretas para minimizar una estructura de diez barras sujeta a restricciones de desplazamiento. Al año siguiente utilizaron un Algoritmo Genético con dos estrategias de cruce diferentes para optimizar estructuras a gran escala, aplicando el algoritmo a estructuras de 25 y 72 barras [237].

En 1993, Adeli y Cheng [6] utilizaron un Algoritmo Genético binario con un cruce por dos puntos y una función de penalización cuadrática de un solo coeficiente constante para optimizar varias estructuras.

En este mismo año, Grierson y Pak [145] emplearon un Algoritmo Genético binario para optimizar el tamaño y la topología de diversos pórticos. Este es uno de los primeros trabajos en analizar estructuras con elementos de viga.

Sakamoto y Oda [331] presentaron una técnica de optimización topológica empleando un método híbrido que combinaba los Algoritmos Genéticos con un método basado en el Criterio de Optimalidad para optimizar la topología y el tamaño de una estructura de forma simultánea. En esta aproximación el Algoritmo Genético optimizaba la topología mientras que el Criterio de Optimalidad optimizaba el tamaño.

[6] El problema 10 barras y seis nudos

Hajela y Lee [154, 155], Hajela et al. [158] emplearon los Algoritmos Genéticos en la optimización topológica de estructuras reticuladas mediante un proceso de dos etapas. En la primera etapa emplearon criterios de estabilidad cinemática para identificar la configuraciones estables. En una segunda etapa añadían o quitaban elementos y redimensionaban el tamaño de los mismos.

En 1994, Adeli y Cheng [7] utilizaron un Algoritmo Genético binario empleando la transformación Lagrangiana aumentada para convertir el problema en no restringido, aplicando el algoritmo a estructuras tridimensionales. El mismo año Dhingra y Lee [91] emplearon un Algoritmo Genético en un problema de optimización donde coexistían variables continuas y discretas, comparando los resultados con los obtenidos empleando técnicas basadas en el gradiente. Koumousis y Georgiou [214] emplearon un Algoritmo Genético binario en la optimización de cerchas comparando los resultados obtenidos por un Algoritmo Genético con brecha generacional con otro sin esta.

En 1995, Rajan [310] presenta un trabajo seminal en el que extiende la magnitud del problema de optimización del tamaño a la forma y topología de forma simultánea, reduciendo en un 44% el mejor resultado obtenido hasta la fecha para el problema de diez barras. En la actualidad este resultado se encuentra todavía en un nada despreciable cuarto lugar a una corta distancia del resto según se puede apreciar en la tabla 6.1. La función de penalización empleaba un coeficiente constante resultante de la multiplicación del peso mínimo de entre todos los diseños factibles y un coeficiente multiplicador que debe ajustarse al problema. En sus observaciones ya apreciaba que un valor del coeficiente multiplicador demasiado grande restringia el espacio de diseño, mientras que un valor demasiado pequeño no es suficiente para empujar a los individuos a la zona factible. En dicho trabajo realizaba las siguientes indicaciones:

1. Según su experiencia en la resolución de problemas de optimización, en los problemas donde se combina la optimización de la topología con el tamaño, aproximadamente entre el 10 y el 30% de las estructuras resultantes son inestables. Asi mismo indicaba que debería emplearse el Método de los Elementos Finitos como método de cálculo. Por otra parte planteaba la necesidad de emplear un coeficiente de penalización dinámico con el fin de obtener la topología óptima, asumiendo por lo tanto que la topología obtenida no era la óptima.

2. La solución final está fuertemente influenciada por la estructura inicial. También reconocía no poder proporcionar ningún tipo de guía de como generar la estructura inicial adecuada.

3. La calidad y diversidad de la población inicial tiene un gran peso en la dirección que el algoritmo genético toma. Si se emplea una solución previa como punto de partida para realizar una reinicialización del algoritmo la solución obtenida está muy próxima a la solución previa. Reconocía la necesidad de determinar un procedimiento adecuado para inicializar la población.

Pese a no estar suficientemente demostradas, estas se han tenido especialmente en cuenta a la hora de abordar la implementación del algoritmo descrito en el presente trabajo.

También Ohsaki [279] presentó un método de para optimizar la topología y el tamaño de las estructuras empleando un Algoritmo Genético, pero de menor importancia que el anterior.

En 1996, Soh y Yang [361] emplearon un Algoritmo Genético híbrido regulado por un control fuzzy y un sistema experto, logrando un requerimiento computacional inferior, mejorando la eficiencia del algoritmo.

También Galante [124] aplicó los Algoritmos Genéticos al diseño de estructuras bidimensionales, empleando el operador de renacimiento propuesto por Goldberg. Aplicó el algoritmo a la estructura de 10 barras y la torre de transmisión de 160 barras.

En 1997, Yang y Soh [421] emplearon una Algoritmo Genético con un operador de selección por torneo en el diseño de estructuras, demostrando que en la optimización de estructuras, este operador es más eficiente que el operador convencional de selección por ruleta.

Rajeev y Krishnamoorthy [312] publicaron una metodología para la optimización simultánea de topología y tamaño. Su metodología permitía emplear variables continuas y discretas empleando una codificación de longitud variable que permitía variar la topología y el tamaño.

Huang y Arora [182] aplicaron un Algoritmo Genético binario al diseño de estructuras empleando perfiles comerciales. Compararon tres estrategias: una optimi-

zación con variables continuas empleando SA, otra empleando GA y una última empleando BB. Aplicaron las estrategias al clásico problema de 10 barras, a un pórtico de 2 vanos y seis plantas y a una estructura de 200 barras.

Jenkins [193] empleó un Algoritmo Genético binario en la optimización de un pórtico de varias plantas, sugiriendo que el rendimiento de los Algoritmos Genéticos se vería mejorado con un control adaptativo del cruce y la mutación.

Cheng y Li [65] publicaron un método de optimización multiobjetivo restringida donde combinaron un Algoritmo Genético de Pareto con una función de penalización fuzzy. El Algoritmo Genético de Pareto estaba compuesto de cinco operadores genéticos: reproducción, cruce, mutación, nicho y un filtro de frente de Pareto cuya función era almacenar los individuos no dominados de cada generación.

En 1998, Leite y Topping [228] presentaron varias modificaciones sobre los operadores genéticos convencionales, demostrando que de esta manera se podía reducir sensiblemente el número de evaluaciones de la función objetivo. Aplicaron el algoritmo en la resolución de varios problemas como el clásico de diez barras (aunque sin emplear restricciones de deformación máxima), un diseño de una unión soldada, una viga continua de tres vanos y una sección en I pretensada.

Camp et al. [54] desarrollaron una herramienta nueva de diseño FEAPGEN como módulo del programa de cálculo por elementos finitos FEAP[7], el cual sigue siendo actualizado y mantenido por los autores en la Universidad de Berkley. Entre las características de FEAPGEN destacan: posibilidad de utilizar variables de diseño discretas, formato abierto para la definición de las restricciones, cálculo de las tensiones según las especificaciones de AISC-ASD, posibilidad de aplicar múltiples estados de carga, incorporación de una base de datos con los perfiles estandarizados según la AISC. La herramienta se aplicó a varios casos de diseño entre los que se encuentran el problema de diez barras, un pórtico de un vano y ocho plantas y otro de tres vanos y tres plantas.

Chen y Rajan [62] desarrollaron un software de diseño de pórticos basado en un Algoritmo Genético Simple binario que incorporaba un operador de cruce por un punto que actuaba de forma específica sobre cada variable. El principal objetivo de este operador era evitar el efecto disruptivo que producía otros tipos de operadores

[7] http://www.ce.berkeley.edu/projects/feap/

de cruce. Los resultados mostraron que esta estrategia era más eficiente que las anteriores.

Ohmori y Kito [278] propusieron una nueva metodología de optimización de estructuras donde la topología de la estructura se expresaba como una combinación de triángulos unidos. Este enfoque pretendía evitar topologías con elementos inútiles, colineales o inestables. Adicionalmente propuso una estrategia en donde varias poblaciones evolucionaban en paralelo con la idea de proporcionar un *efecto ambiental* al algoritmo con el fin de mejorar la eficiencia del mismo. Es el primer caso de optimización de estructuras donde se describe los beneficios de utilizar múltiples poblaciones con el fin de incrementar la diversidad de las mismas.

Leite y Topping [228] publicaron una serie de modificaciones de los operadores genéticos de cruce y mutación. También plantearon un método de reproducción que generaba más de dos hijos, aplicando su algoritmo al problema clásico de diez barras, aunque sin restricciones de desplazamiento, a una unión soldada, a un resorte a compresión, a una viga continua de tres vanos y a un perfil en I pretensado.

Saka [330] empleó un Algoritmo Genético Simple para minimizar el peso de una estructura reticular sometida a restricciones de tensión y deformación, considerando los efectos de la cortadura y el alabeo.

Hajela et al. [159] aplicaron también un Algoritmo Genético Simple para determinar la distribución óptima de estructuras reticulares bi y tridimensionales, sometidas a restricciones de tensión y desplazamiento. El algoritmo funcionaba en dos niveles. En el primer nivel se aplicaban restricciones de estabilidad, mientras que en el segundo se aplicaban restricciones de tensión y desplazamiento. Aplicaron el algoritmo a la estructura de cola de un helicóptero, comparando los resultados con los obtenidos mediante programación lineal.

Soh y Yang [362] desarrollaron un Algoritmo Genético de dos etapas para diseñar estructuras de puentes. En la primera etapa se optimizaba el tamaño de los elementos de la estructura, mientras que en la segunda se optimizaba la topología para posteriormente volver a optimizar el tamaño.

En 1999, Groenwold et al. [146] desarrollaron uno de los primeros Algoritmos Genéticos en aplicar una codificación no binaria. En esta caso se empleó una codi-

ficación entera. Denominaron a su algoritmo Algoritmo Genético Regional porque partía de una región próxima al óptimo. Emplearon el operador de renacimiento propuesto por Galante [124] como estrategia de selección. Se aplicó el algoritmo a la clásica estructura de diez barras y seis nodos, a una estructura de 25 barras, a una estructura de 36 barras, a un poste de tendido eléctrico de 160 barras y finalmente a una estructura de 200 barras.

Botello et al. [49] propusieron un algoritmo híbrido que empleaba un Algoritmo Genético Simple y el SA. Tras la selección y cruce, el algoritmo compara la población resultante con la previa aceptando o rechazando los nuevos individuos en función de un operador de aceptación controlado mediante SA. En su trabajo analizaron la clásica estructura de 10 barras, comparando sus resultados con algunos previos, así como un pórtico reticulado, una cercha, una torre de tendido eléctrico y una estructura reticular curva tridimensional.

En el 2000, Erbatur et al. [106] aplicaron un Algoritmo Genético binario al diseño de estructuras de barras y vigas al que denominaron GAOS. Emplearon una función de penalización basada en el modelo de Joines y Houck [195] y emplearon una base de datos con los perfiles normalizados AISC. Aplicaron el algoritmo a una estructura de 25 barras, a otra de 72 barras, a una cúpula de acero de 112 barras, a una estructura de 22 barras en voladizo y a la estructura metálica de una nave industrial.

Hasançebi y Erbatur [165] realizaron una evaluación de las diversas técnicas de cruce empleadas en el diseño de estructuras. Para ello emplearon un Algoritmo Genético binario. Para realizar dicha evaluación emplearon las estructuras clásicas de 10, 25 y 72 barras. De este estudio no pudieron concluir que técnica era la más eficiente ya que los resultados dependían del tipo de estructura e incluso del estado de carga.

Pezeshk et al. [301] publicaron un artículo donde se analizaban pórticos con comportamiento no lineal mediante un Algoritmo Genético Simple binario. Definieron un nuevo operador de selección: el operador de selección por grupo. También fueron los primeros en intentar introducir un operador de cruce adaptativo basado en una estrategia coevolutiva. Para ello implementaron en el propio genoma dos dígitos adicionales que controlaban el número de puntos de cruce que deberían producirse al generar los descendientes. Aplicaron el algoritmo a un pórtico de dos vanos y tres plantas.

En 2001, Deb y Gulati [89] emplearon por primera vez el *Operador β de cruce binario simulado* formulado por el propio Deb [85] unos años atrás. El algoritmo empleado fue un Algoritmo Genético Simple binario, aplicándolo a los clásicos problemas de optimización de 10 y 25 barras, a una estructura de 45 barras, y a otra de 39.

Greiner et al. [143] estudiaron el efecto de las diferentes estrategias de reemplazo sobre los Algoritmos Genéticos. El algoritmo utilizado fue un Algoritmo Genético binario, y las estrategias comparadas fueron: la Simple, la de brecha generacional y una *Estrategia Evolutiva* CHC propuesta en este artículo. Para analizar el efecto de estas estrategias emplearon una estructura de pórtico, obteniendo como conclusión que la más adecuada era la estrategia de brecha generacional.

En 2002, Azid et al. [16] fueron los primeros en plantear un Algoritmo Genético con una codificación real para resolver un problema de optimización de estructuras. El operador de cruce propuesto fue diferente a los convencionales para codificaciones reales. En este caso el cruce se realizaba superponiendo los genomas de modo que los genes que fueran bastante similares eran copiados directamente. Para probar su algoritmo emplearon el problema de Michell de la viga en voladizo y la estructura clásica de 25 barras.

En 2003, Kaveh y Kalatjari [196] emplearon una combinación del *Método de Flexibilidad*, la *Teoría de Grafos* u los Algoritmos Genéticos para la optimización de estructuras. Aplicaron dicho algoritmo a las estructuras clásicas de 10 y 25 barras, así como otra de 31.

En 2004, Greiner et al. [144] emplearon un operador de renacimiento autoadaptativo inspirado en el propuesto por Goldberg [135]. El algoritmo empleado fue un Algoritmo Genético con brecha generacional y codificación real. El operador de reinicialización trabajaba almacenando a los mejores individuos una vez se producía la convergencia del algoritmo, para posteriormente reinicializar la población volviendo a incluir a los individuos almacenados. Para probar su algoritmo emplearon diferentes estructuras de tipo pórtico.

Lemonge y Barbosa [229] emplearon un Algoritmo Genético Simple binario con una nueva función de penalización aplicada a la optimización estructural la cual se describe en profundidad en apartado 4.9. Para evaluar dicho algoritmo emplearon un conjunto de funciones matemáticas, una unión soldada, un depósito a presión,

una viga en voladizo de sección variable, los problemas clásicos de diez barras, de 25, de 72 y finalmente otro de 52 barras. Los resultados de esta función de penalización fueron variables dependiendo del problema, en algunos casos no muy buenos como reconocieron los propios autores.

Wu y Lin [416] emplearon un Algoritmo Genético Simple binario con una estrategia de penalización a la que llamó *SOAPS-II*. Empleó el un conjunto de funciones para evaluar dicha estrategia así como el problema clásico de 10 barras.

En 2005, Tang et al. [384] publicaron un trabajo en el que emplearon un Algoritmo Genético mixto con variables continuas, con codificación real, y discretas, con codificación entera. Emplearon un operador de cruce por dos puntos, un operador de mutación gaussiana una función compleja basada en funciones logarítmicas y exponenciales detallada en [235]. Aplicaron el algoritmo a una estructura de 15 barras, al clásico problema de 25 barras y al problema de 10 barras, obteniendo el mejor resultado hasta el presente día.

Prendes Gero et al. [307, 308] emplearon un Algoritmo Genético binario elitista donde emplearon un operador de cruce por fenotipo, con cierta similitud al empleado anteriormente por Chen y Rajan [62], comparándolos con los operadores de cruce por genotipo tradicionales, demostrando la superioridad de los primeros. El presente trabajo tiene en cuenta los resultados de estos autores en el desarrollo de la codificación del algoritmo.

Canyurt y Hajela [55] utilizaron un autómata celular basado en un Algoritmo Genético para predecir los resultados FEM de estructuras continuas y discretas obteniendo resultados muy prometedores.

Lingyun et al. [240] presentaron un Algoritmo Genético sometido a restricciones de frecuencia, aplicándolo a la estructura de diez barras, un puente estudiado previamente por Wang y Arora [406] y una cúpula de 52 barras.

En 2006, Balling et al. [21] publicaron un trabajo donde desarrollaron un Algoritmo Genético con codificación real donde la función de aptitud estaba influenciada por la topología de los individuos, con el fin de intentar aumentar la diversidad de la población. Para probar su algoritmo analizaron varias estructuras como la clásica de diez barras, obteniendo el segundo mejor registro en la optimización de esta

estructura. También analizó el problema de Kirsch, un pórtico de cuatro plantas y tres vanos, así como un puente de tres vanos.

Kaveh y Rahami [197] propusieron una combinación de un Algoritmo Genético con el *Método de Flexibilidad* para el análisis de estructuras con comportamiento no lineal del material o la geometría. Con dicho algoritmo analizaron un pórtico de tres barras, las clásicas estructuras de 10 y 25 barras, una viga continua con tres vanos y una estructura de 31 barras.

Kaveh y Shahrouzi [198] emplearon un Algoritmo Genético binario donde utilizaron una codificación indexada mediante una tabla de correspondencias en lugar de la tradicional codificación binaria, obteniendo un comportamiento más eficiente del algoritmo comparado con una codificación binaria tradicional. Aplicaron el algoritmo a la clásica estructura de diez barras, con un resultado aceptable, así como a la también clásica estructura de 25 barras.

Dominguez et al. [95] aplicaron un Algoritmo Genético al diseño real de la estructura de una grúa.

Togan y Daloglu [392] emplearon un Algoritmo Genético en la optimización de estructuras tridimensionales. Demostraron la influencia de la adaptación de los parámetros de los operadores genéticos en la eficiencia del algoritmo, empleando una función de penalización adaptativa. Aplicaron el algoritmo a la estructura de una cúpula de 112 barras, a una estructura de 200 barras y a una torre de tendido eléctrico de 244 barras.

En 2007, Srinivas y Ramanjaneyulu [373] plantearon la interesante idea de emplear una red neuronal para predecir la aptitud de los individuos de un Algoritmo Genético. Lamentablemente no aportaron aplicaciones prácticas de su trabajo.

Xu et al. [419] analizaron la forma de diseñar estructuras sujetas a restricciones de resonancia mediante Algoritmos Genéticos. Para probar dicho algoritmo lo aplicaron a varias estructuras demostrando que era posible reducir el peso de las mismas sin tener problemas de resonancia.

Belloli y Ermanni [37] aplicaron un Algoritmo Genético para determinar el emplazamiento óptimo de un sensor piezoeléctrico con el fin de reducir la vibración de estructuras altamente restringidas. Aplicaron dicho algoritmo al alerón trasero de un coche de carreras.

En 2008, Togan y Daloglu [393] plantearon un Algoritmo Genético con un operador de inicialización basado en una técnica heurística. Para evaluar el algoritmo emplearon las estructuras clásicas de diez y 25 barras, una torre de tendido eléctrico de 244 barras, una estructura de 200 barras,y una cúpula de 120 barras. Como resultado de su trabajo concluyeron, de modo similar a Rajan [310], que los resultados se ven altamente afectados por la diversidad de la población inicial y su distancia al óptimo.

Kaveh y Shahrouzi [199] realizaron una combinación de un Algoritmo Genético binario con un algoritmo ACO, de modo que los mejores individuos de la población exploraban el espacio de soluciones empleando este último.

En 2009, Guo et al. [147] propusieron un Algoritmo Genético mejorado donde la población inicial es generada mediante la triangulación de Delaunay y una técnica heurística similar a la de Togan y Daloglu [393]. Para evaluar el algoritmo se empleó una estructura de 16 barras.

En 2011, Noilublao y Bureerat [277] emplearon un Algoritmo Genético con una codificación mixta real-entera para optimizar una torre tridimensional sujeta a restricciones de resonancia, además de las tradicionales de tensión y desplazamiento.

Hajirasouliha et al. [160] aplicaron un Algoritmo Genético al diseño de estructuras sismoresistentes.

El resto de *Algoritmos Evolutivos* como la *Programación Evolutiva*, las *Estrategias Evolutivas* y la *Programación Genética* han tenido mucha menor repercusión en el campo de la optimización de estructuras.

No obstante, cabe destacar el importante papel que tienen las *Estrategias Evolutivas* en el campo de la optimización estructural de medios continuos, donde prácticamente dominan al resto de técnicas.

Los trabajos más relevantes de estas técnicas, relacionados con campo de la optimización estructural son:

En 1998, Papadrakakis et al. [293] propusieron una *Estrategia Evolutiva* combinada con una red neuronal con el fin de reducir el número de evaluaciones de la función objetivo del algoritmo. Emplearon un pórtico de seis pisos y otro de veinte para probar su algoritmo.

En 2002, Lagaros et al. [218] compararon diferentes algoritmos en el diseño de estructuras a gran escala. Entre estos utilizaron Algoritmos Genéticos, un μGA, un Algoritmo Genético segregado, una ES, así como varios algoritmos híbridos que combinaban la SPQ con los GA o las ES. Emplearon pórtico de seis pisos y otro de veinte para probar su algoritmo obteniendo los mejores resultados al combinar SPQ con los GA o las ES.

En 2004 y 2007, Kicinger y Arciszewski [206], Kicinger et al. [207], Kicinger [208] publicaron una *Estrategia Evolutiva* aplicada al diseño de estructuras metálicas de edificios altos. En este empleó una representación de la estructura como una composición de celdillas dentro de las cuales podrían encontrarse una barra en una u otra diagonal, en V o Λ, en X simple o unida al centro. También codificó de forma diferente el tipo de unión dependiendo de si era una articulación o un nudo. A cada una de estas representaciones le asignó un número y con este procedió a codificar la estructura entera. Gracias a esta representación pudo analizar un edificio de 36 plantas y 3 vanos.

En 2005, Ebenau et al. [100] publicaron una *Estrategia Evolutiva* $(\mu + 1)$ con una función de penalización adaptativa. Para probar su algoritmo emplearon el problema clásico de 10 barras, obteniendo el mejor resultado hasta la realización del presente trabajo, así como a una estructura de 120 barras.

En 2008, Hasançebi [164] aplicó con éxito una *Estrategia Evolutiva* a una torre de 26 plantas y 942 barras, así como a varias estructuras de puente.

Isaacs et al. [186] aplicaron una ES en dos etapas. En una primera etapa eliminaba aquellos elementos con una energía de deformación inferior a un umbral predefinido, mientras que en la segunda optimiza el tamaño de la estructura. Para probar el algoritmo analizaron las clásicas estructuras de 10 y 25 barras. La técnica no presenta resultados mucho mejores a los publicados, pero en contraposición requiere un número de evaluaciones de mucho menor.

La Optimización por enjambre de partículas (PSO)

La PSO está basada en el comportamiento social de animales como los bancos de peces, enjambres de insectos y bandadas de pájaros. Este comportamiento está relacionado con los grupos y las fuerzas sociales que dependen de la memoria de cada individuo y de la inteligencia del grupo.

El algoritmo está formado por una serie de partículas que forman el enjambre, las cuales son inicializadas aleatoriamente dentro del espacio de búsqueda de la función objetivo. Cada partícula representa una posible solución del problema. Las partículas se mueven por el espacio de búsqueda atraídas por la posición de mayor aptitud lograda por la partícula (óptimo local) así como la mejor aptitud lograda en todo el enjambre (óptimo global), durante cada iteración del algoritmo, de modo similar al de un enjambre.

Comparada con el resto de técnicas de *Computación Evolutiva* esta técnica es muy simple y requiere de pocos parámetros de ajuste. Durante los últimos años ha sido una técnica bastante estudiada en la optimización de estructuras, destacando los siguientes trabajos:

En 2003, Schutte y Groenwold [344] aplicaron por primera vez[8] con éxito un algoritmo PSO a la optimización de estructuras. El algoritmo imitaba el comportamiento social de una bandada de pájaros donde cada pájaro intercambia con los mas próximos su posición velocidad y aptitud mientras el comportamiento global de la bandada tiende a desplazarse a las regiones de mayor aptitud. Para evaluar el algoritmo emplearon las clásicas estructuras de diez y 25 barras, así como una estructura de 36 barras. Su algoritmo es logró el mejor resultado para la estructura clásica de 10 barras de entre todos los PSO publicados hasta hoy.

En 2007, Li et al. [233] emplearon un algoritmo PSO para optimizar de estructuras articuladas. En este trabajo intentaron evitar uno de los inconvenientes de los PSO: su rápida convergencia provoca que, en problemas complejos, la solución converja a un óptimo local. Para probar su algoritmo lo aplicaron sobre las estructuras clásicas de 10 y 25 barras, así como una estructura de 22 y 72 barras.

[8] Aunque existe un precedente publicado en un congreso [120], este suele ser tomado como el primer trabajo

Perez y Behdinan [299] utilizaron un algoritmo PSO para la optimización de estructuras donde emplearon la función de penalización de Lemonge y Barbosa [22, 229]. Validaron el algoritmo con las tradicionales estructuras de 10, 25 y 72 barras.

En 2009, Kaveh y Talatahari [200, 201] presentaron un algoritmo híbrido HPSOACO que combinaba una *Búsqueda Armónica* (HS), un algoritmo PSO y un algoritmo ACO para la optimización de estructuras. El algoritmo emplea el PSO para la búsqueda global y el ACO para la búsqueda local. La Búsqueda Armónica se empleó para manejar las restricciones. Para probar su algoritmo utilizaron la estructura de 10 barras, obteniendo unos resultados bastante pobres, así como una cúpula de 120 barras y una torre de 582 barras. La ventaja de este algoritmo frente a los publicados anteriormente es el bajo número de evaluaciones de la función objetivo requeridas.

Li et al. [234] publicaron un algoritmo HPSO que empleaba una técnica de *Búsqueda Armónica* (HS) para acelerar la convergencia del algoritmo. Utilizaron las estructuras clásicas de 10 y 25 barras, así como otras de 52 y 72 para probar la convergencia del algoritmo. Este algoritmo proporcionó el mejor resultado después de Schutte entre los PSO para la estructura de 10 barras.

En 2011, Luh y Lin [245] presentaron un algoritmo donde la optimización de la estructura se realizaba en dos etapas. En una primera etapa se optimizaba la topología mediante un PSO binario para posteriormente modificar el tamaño mediante un PSO inspirado en la atracción y repulsión de partículas. Para evaluar el funcionamiento del algoritmo empleo una *Ground Structure* de 39 barras y 12 nodos. A pesar de afirmar en las conclusiones que el algoritmo proporcionaba mejores resultados que los publicados anteriormente, no se indica referencia alguna a los mismos.

Martins y Gomes [254] emplearon un algoritmo PSO aplicado a estructuras sometidas a restricciones de resonancia. Para probar su algoritmo emplearon las estructuras clásicas de 10 y 72 barras, así como un puente de 37 y una cúpula de 52.

A pesar de su simplicidad, los PSO presentan el inconveniente de su rápida convergencia, lo cual suponen un serio problema para el presente campo de estudio.

La Optimización por colonia de hormigas (ACO)

Este tipo de algoritmos pretenden imitar el comportamiento natural de las colonias de hormigas, abejas o avispas. Estas colonias están formadas por individuos que desarrollan diversas tareas como la exploración en busca de comida, el transporte de la misma, la construcción de nidos y la defensa. Cada miembro de la colonia realiza su propia tarea interaccionando con el resto de individuos de la colonia, de modo que si un individuo no es capaz de realizar su tarea, la colonia en su conjunto si lo hace. El proceso de división de las tareas entre individuos es más efectivo que la tarea realizada por un individuo aislado.

El primer ACO desarrollado se inspiró en la forma en que las hormigas sortean los obstáculos a la hora de transportar comida. Cuando una hormiga encuentra la forma de sortear el obstáculo, en poco tiempo toda la colonia sabe que camino tomar gracias al sistema de transmisión de información basado en las feromonas marcadoras de pistas que estas poseen. El autor aplicó este mismo mecanismo al problema del viajante de comercio.

Debido a su sencillez durante los últimos años han sido objeto de investigación en cuanto a su aplicación en la optimización de estructuras. Seguidamente se detallan los trabajos mas importantes de la optimización de estructuras basados en esta técnica:

En 2004, Camp y Bichon [53] publicaron el primer trabajo relacionado con la ACO. Para probar su algoritmo emplearon las estructuras clásicas de diez, 25 y 72 barras. Obtuvieron el mejor registro, para la estructura de 10 barras, hasta la actualidad para un algoritmo ACO sin hibridizar.

En 2006, Serra y Venini [347] realizaron un trabajo donde se empleaba un ACO para la optimización de la estructuras, probando dicho algoritmo con la estructura clásica de 10 barras, pero sin restricciones de desplazamiento.

En 2010, Chen et al. [64] emplearon un PSO para la optimización de estructuras donde utilizaron las estructuras clásicas de 10 y 25 barras como método de prueba, con un resultado bastante pobre.

En 2011, Sonmez [364] emplearon un ACO inspirado en las colonias de abejas. Para probar la eficacia emplearon la estructura clásica de 10 barras, con un resultado

peor pero cercano al de Camp, así como una estructura en voladizo de 18 barras, otra de 25 barras, una torre de 72 barras y finalmente una estructura de 200 barras.

Los resultados obtenidos por esta técnica se encuentran lejos de los obtenidos por la PSO y aun mas lejos de los Algoritmos Genéticos, aunque bien es cierto que todavía falta un estudio más riguroso de esta técnica dentro de este campo.

3.3 Formulación del problema de optimización de estructuras

La determinación de la función objetivo en la tarea de optimización no es una cuestión trivial ya que de esta dependerá los resultados que se obtengan. En la mayoría de los casos se optimiza una sola característica del problema, como el peso, pero también se podría incluir otras características adicionales como el coste económico, la duración, etc ...

Sin embargo, la mayoría de estudios analizados se centra en la minimización del peso de las estructuras y será en este aspecto en el que se enmarque el presente trabajo.

El diseño de estructuras requiere además que estas posean ciertas propiedades de resistencia, rigidez o resonancia que restringen el espacio de soluciones posibles. El proceso de optimización requiere pues de un conjunto de variables de diseño que proporcionen el peso mínimo sin violar las restricciones mencionadas.

La formulación matemática del problema de optimización suele expresarse del siguiente modo:

Encontrar: $\boldsymbol{X} \in \mathbb{R}^k$
que minimizan

$$W = F(\boldsymbol{X})$$

sujeto al conjunto de restricciones:

$$g_i(\boldsymbol{X}) \leq 0 \quad \forall i = 1, 2, \ldots, m$$
$$X_j^L \leq X_j \leq X_j^U \quad \forall j = 1, 2 \ldots, k$$

donde: W es el peso de la estructura, $\boldsymbol{X}$ es el conjunto de variables de diseño formado por el material, la sección transversal y la longitud de cada barra, k es el número de variables de diseño, $g_i(\boldsymbol{X})$ es una de las restricciones de diseño, m es el número total de restricciones de diseño, X_j^L, X_j^U son los límites inferior y superior de la variable de diseño j.

4

Optimización mediante Algoritmos Genéticos

4.1 Breve introducción a la Computación Evolutiva

La Computación evolutiva (EC) es una rama de la inteligencia artificial, inspirada en los mecanismos de evolución biológica, que involucra problemas de optimización combinatoria. Está considerada como una técnica metaheurística y por lo tanto es adecuada para la resolución de problemas con espacios de búsqueda extensos, dispersos, no lineales y no convexos, en donde otros métodos no son capaces de encontrar soluciones en un tiempo razonable.

Los Algoritmos Evolutivos (EA) incluyen a todas las técnicas relacionadas con la optimización metaheurística inspiradas en la teoría de la evolución. Las primeras simulaciones de la evolución mediante Algoritmos Evolutivos y vida artificial empezaron con el trabajo de Nils Aall Barricelli [24] durante la década de los 60. Su trabajo fue continuado por otros como Alex Fraser [121], Ingo Rechenberg [313, 314], Lawrence J. Fogel [118] y John Henry Holland [178] entre otros, durante los 60 y principios de los 70 sentando las bases de esta técnica.

Aunque la cantidad de técnicas y algoritmos existentes dentro de esta clasificación aumenta día a día, existen cuatro paradigmas fundamentales de los Algoritmos Evolutivos que son, a saber:

1. Los *Algoritmos Genéticos* (GA): Fueron desarrollados por John H. Holland y sus colaboradores. Utilizaron inicialmente la codificación binaria, aunque en la actualidad también se ha extendido a la codificación con números reales. Emplea operadores genéticos de selección, recombinación y mutación teniendo mayor relevancia los operadores de selección y recombinación (reproducción).

2. La *Programación Evolutiva* (EP): Fue desarrollada por Lawrence J. Fogel con la idea de simular la evolución como medio de aprendizaje con el fin de obtener una inteligencia artificial. Esta trabaja directamente con las variables de diseño y los nuevos individuos son generados a partir de un solo padre mediante el uso exclusivo de la mutación. En cierto modo es una variación de los Algoritmos Genéticos con reproducción asexual. Emplean solamente los operadores genéticos de mutación y selección $(\mu + \lambda)$ [1].

[1] Tamaño de la población + Número de hijos generados

3. Las *Estrategias Evolutivas* (ES): Fueron desarrolladas por Ingo Rechenberg Rechenberg [313, 314], Hans-Paul Schwefel Schwefel [345] y sus colaboradores. Utilizan generalmente la codificación con números reales. La función de codificación transcribe dos tipos de variables: las variables objeto y las estratégicas. Las variables objeto se corresponden con las del problema que se desea resolver mientras que las estratégicas son los parámetros mediante los cuales se gobierna el proceso evolutivo (tasa de mutación). Emplea operadores genéticos de selección $(\mu + \lambda)$ o (μ, λ) y mutación.

4. La *Programación Genética* (GP): Fue desarrollada por Nichael Cramer y John Koza [216] con la idea de generar automáticamente de programas de ordenador en diferentes lenguajes y formas como los árboles de decisión y grafos empleando operadores genéticos para tal fin.

La Computación Evolutiva incluye otros algoritmos, no clasificables dentro del paradigma de los Algoritmos Evolutivos, de entre los que destaca la Inteligencia de Enjambre (SI). Inicialmente desarrollada por Gerardo Beni y Jing Wang en 1989 con objeto de ser aplicadas a sistemas robóticos. Se basa en el estudio del comportamiento colectivo, descentralizado, de sistemas autoorganizados naturales o artificiales. Están constituidos por una población de individuos que interactúan entre sí y el medio. Los individuos siguen unas reglas muy simples y a pesar de no existir una estructura de control debido a las interacciones entre los individuos aparece una especie de *inteligencia colectiva*. En la naturaleza encontramos este tipo de mecanismo en las colonias de hormigas, bandadas de pájaros, bancos de peces o el crecimiento de bacterias.

Engloba a un conjunto de algoritmos de entre los que destacan: Optimización mediante colonias de hormigas o abejas, sistemas inmunes artificiales, algoritmo de búsqueda gravitacional, algoritmos luciérnaga, optimización de partículas de enjambre (PSO), dinámica de formación de ríos, partículas autopropulsadas, gotas de agua inteligentes y la búsqueda de difusión estocástica. La mayoría de estos algoritmos están todavía en desarrollo y suelen tener aplicaciones muy específicas.

Como se analiza en el primer capítulo, los Algoritmos Genéticos son el método más extendido en la optimización de estructuras. Adicionalmente, existe un gran número de librerías contrastadas en diferentes lenguajes de programación que permiten llevar a cabo este tipo de simulaciones. De entre ellas destaca la librería

desarrollada por el doctor del MIT Matthew Wall: `http://lancet.mit.edu/ga/` que es la empleada, con modificaciones, en este trabajo.

4.2 Antecedentes históricos de los Algoritmos Genéticos

Los Algoritmos Genéticos son un tipo de algoritmo de búsqueda heurística inspirados en la genética y la selección natural enunciada por el naturalista inglés Charles Darwin en el libro *"The Origin of Species by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life"*(El Origen de las Especies) [78]. Según esta teoría los individuos más aptos de una población son los que sobreviven al adaptarse más fácilmente a los cambios que se producen en su entorno.

Los Algoritmos Genéticos fueron desarrollados por John Henry Holland, sus colaboradores (de entre los que destaca De Jong [83] de la Universidad de Michigan), a finales de los sesenta [75, 178, 179]. Después de estudiar un libro escrito por Fisher [115], titulado *"La teoría genética de la selección natural"*, Holland aprendió que la evolución es una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar un algoritmo que permitía a los ordenadores imitar el proceso de la evolución. Los objetivos fundamentales de su investigación fueron explicar de forma rigurosa los procesos adaptativos de los sistemas naturales y diseñar sistemas artificiales que imiten los mecanismos de estos últimos. De estos estudios dedujo que:

1. La evolución es un proceso que opera sobre cromosomas más que sobre los seres vivos que estos codifican.
2. El proceso de reproducción es el punto en el que se produce la evolución.
3. El cruce genera en los hijos cromosomas diferentes por recombinación de los cromosomas de sus padres.
4. Las mutaciones generan en los hijos genes diferentes a los de sus padres.

Posteriormente, en 1989, Goldberg que también fue discípulo de Holland publicó un libro [135] que le dio una fuerte base científica, citando no menos de 73 aplicaciones exitosas de los algoritmos genéticos.

Desde entonces numerosos artículos, tesis [48, 50, 61, 83, 208, 313, 408] y libros [15, 26, 69, 70, 80, 82, 104, 117, 126, 130, 137, 167, 176, 216, 246, 257, 258, 263, 300, 315, 332, 355] han establecido su validez como técnica de optimización. Han demostrado ser robustos eficientes y computacionalmente simples. Además no están sujetos a restricciones sobre el espacio de búsqueda como la necesidad de continuidad o la existencia de derivadas de la función objetivo.

Los mecanismos de búsqueda de los Algoritmos Genéticos se basan en la genética y la selección natural. Combinan el concepto de supervivencia artificial de los individuos más adaptados (o aptos) con operadores genéticos imitados de la naturaleza como el cruce o la mutación. Los Algoritmos Genéticos difieren de los métodos tradicionales de optimización de varias formas [135]:

- Utilizan funciones de objetivo para determinar la aptitud del individuo (punto del dominio) a evaluar, no derivadas u otro tipo de información adicional. Los Algoritmos Genéticos son ciegos.
- No trabajan con las variables de diseño, sino con una codificación de estas.
- Utilizan un conjunto de puntos del dominio en contraposición con los métodos tradicionales que se basan en un único punto. Esto se traduce en que en cada iteración los Algoritmos Genéticos procesan y evalúan un determinado número de diseños.
- Utilizan reglas u operadores estocásticos en lugar de las tradicionales reglas determinísticas.
- Existe la certeza matemática de que el algoritmo es capaz de obtener siempre un óptimo global, si no existe una limitación temporal en el cálculo [179].

La principal cualidad de este tipo de algoritmo es su robustez y el equilibrio entre eficiencia y eficacia en la determinación de puntos óptimos. Esto les permite ser aplicados, con ciertas restricciones, a un amplio abanico de problemas de optimización. También los hace ideales cuando la búsqueda tiene lugar en entornos altamente cambiantes (p.e. la función objetivo varía con el tiempo) o altamente restringidos, donde las restricciones suelen ser contrapuestas entre sí.

El propio Goldberg define los Algoritmos Genéticos como:

> ≪Algoritmos de búsqueda basados en los mecanismos de selección y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana≫

Otra buena definición de Algoritmo Genético sería la dada por Koza [215]:

> ≪Un Algoritmo Genético es un algoritmo matemático fuertemente paralelizado que transforma una **población**[2], cuyos individuos tienen un **fitness**[3] asociado, en una nueva población (siguiente **generación**) utilizando operaciones basadas en los principios darwinianos de reproducción y supervivencia de los mejores (más aptos)≫.

Cualquier Algoritmo Genético, independientemente del problema planteado, está compuesto por los siguientes cinco componentes:

1. Una representación de las potenciales soluciones del problema.
2. Un método para crear la población inicial de posibles soluciones.
3. Una función de evaluación que juega el papel de medio calificando las soluciones en base a su fitness.
4. Operadores genéticos que alteran la composición de los hijos.

4.3 Antecedentes biológicos

La genética es la ciencia que se encarga de dilucidar los mecanismos responsables de las similaridades y diferencias entre las especies. Los fundamentos de los Algoritmos Genéticos provienen de la teoría de la evolución y la genética.

A continuación se describe la terminología básica empleada en estos campos:

[2]Conjunto de individuos (soluciones) en una generación
[3]Aptitud. Cuantificación del grado de adaptación al medio

4.3.1 La célula

Las células constituyen el elemento de menor tamaño que puede considerarse vivo. Está presente en todos los seres vivos pudiendo ser clasificados en función del número de células que poseen como: unicelulares (protozoos, bacterias, etc ...) o pluricelulares. En este último caso los organismos pueden tener de unos pocos cientos a cientos de billones como en caso del ser humano (10^{14}).

La figura 4.1 muestra la anatomía de una célula animal.

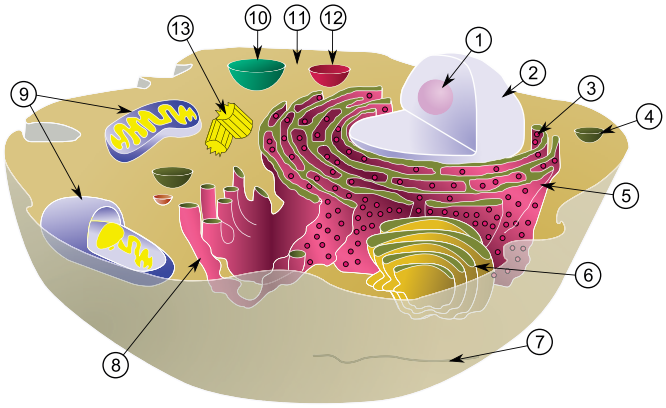


Figura 4.1: Diagrama de una célula animal: 1. Nucléolo, 2. Núcleo, 3. Ribosoma, 4. Vesícula, 5. Retículo endoplasmático rugoso, 6. Aparato de Golgi, 7. Citoesqueleto (microtúbulos), 8. Retículo endoplasmático liso, 9. Mitocondria, 10. Vacuola, 11. Citoplasma, 12. Lisosoma. 13. Centríolos. Figura bajo licencia de Creative Commons Atribución 3.0, no adaptada

4.3.2 Los cromosomas

El núcleo de la célula contiene la mayor parte del material genético celular en forma de múltiples moléculas lineales de ácido desoxirribonucleico (ADN) conocidas como cromatina, que durante la división celular (mitosis) se organiza formando pequeños cuerpos en forma de bastoncillos conocidos como cromosomas. Por lo tanto, cromatina y cromosoma son dos aspectos morfológicamente distintos de una misma entidad celular.

La figura 4.2 muestra la anatomía de los cromosomas durante la mitosis.

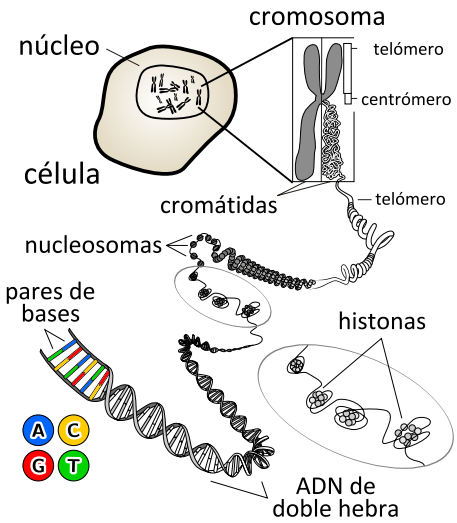


Figura 4.2: Situación del ADN dentro de una célula. Figura bajo licencia de Creative Commons Atribución 3.0, no adaptada

4.3.3 Los genes

El ADN es la molécula portadora de la información genética. Es un ácido nucleico que está formado por la unión de moléculas más sencillas llamadas nucleótidos que se unen formando cadenas.

Un nucleótido está constituido por estas tres moléculas:

1. Una base nitrogenada. Hay cinco bases nitrogenadas diferentes:
   - Adenina (A)
   - Guanina (G)
   - Citosina (C)
   - Timina (T)
2. Un azúcar de cinco carbonos (pentosa).
3. Un ácido fosfórico.

La pentosa puede ser la ribosa o la desoxiribosa. La presencia de una y otra forma dos tipos de ácidos: el desoxiribonucleico o ADN, constituyente fundamental del núcleo, o el ARN, abundante en el citoplasma y nucleolo.

Estas cadenas se unen entre sí de forma que cada nucleótido de una de las cadenas se une de forma específica con otro nucleótido de la otra cadena. Los enlaces se establecen entre las bases hidrogenadas y son siempre: A – T y C – G. El resultado son dos cadenas complementarias que se enrollan en espiral, formando una doble hélice, conformando así la estructura característica de una molécula de ADN.

La complementariedad de las bases nitrogenadas permite que la secuencia de una cadena sencilla de ADN actúe como un molde para la formación de una copia complementaria de ADN (replicación) o de ARNm (transcripción).

Un gen es un segmento de ADN que contiene la información necesaria para la síntesis de una macromolécula con función celular específica, normalmente proteínas, pero también ARNm, ARNr y ARNt. Esta función puede estar vinculada al desarrollo o funcionamiento de una función fisiológica, como por ejemplo, el color de los ojos. La posición del gen dentro del cromosoma se denomina *locus*. Los genes pueden tener formas alternativas, denominadas alelos, que se pueden manifestar en modificaciones concretas de la función de ese gen.

La mayoría de organismos pluricelulares son diploides, es decir, tienen dos juegos de cromosomas. Uno procede de la madre y el otro del padre. Los organismos diploides tienen una copia de cada gen (y por lo tanto de cada alelo) en cada cromosoma. Si ambos alelos son iguales se denominan homocigóticos. Si son diferentes heterocigóticos. La longitud de los cromosomas homólogos es idéntica (excepto los cromosomas que determinan el sexo) y la posición de los genes o alelos en los mismos invariable.

Cuando los alelos heterocigóticos están presentes, solo uno de los dos estará presente en el fenotipo (rasgos) del individuo. Este gen o alelo se denomina dominante, en contraposición con el alelo recesivo. Este último si bien está presente en el individuo no lo hace de forma observable, pudiendo ser transmitido a su descendencia.

El conjunto de posibles alelos presentes en una población particular forma la reserva genética. El tamaño de la reserva genética determina la diversidad de la población.

El conjunto de genes de una especie, y por tanto de los cromosomas que los componen, se denomina genoma.

4.3.4 La genética

Toda la información genética contenida en los cromosomas de un individuo se conoce como genotipo, sin embargo dicha información puede o no manifestarse en el individuo. El fenotipo es cualquier característica o rasgo observable de un organismo, como su morfología, desarrollo, propiedades bioquímicas, fisiología y comportamiento, resultantes de la decodificación del genotipo. El fenotipo está determinado por el genotipo y por factores del medio. En este sentido, la interacción entre el genotipo y el fenotipo ha sido descrita usando la simple ecuación que se expone a continuación:

$$\text{Ambiente} + \text{Genotipo} + \text{Ambiente} * \text{Genotipo} = \text{Fenotipo}$$

La figura 4.3 muestra un ejemplo de Diversidad genotípica, tanto en el color como en el patrón geométrico, de los individuos de la especie de moluscos *Donax variabilis* debida a una gran diversidad de alelos. Un punto interesante de la evolución

Figura 4.3: Diversidad genotípica de los moluscos *Donax variabilis* debida a una gran diversidad de alelos. Figura bajo licencia de Creative Commons Atribución 3.0, no adaptada

es que mientras la selección siempre se basa en el fenotipo, la reproducción recom-

bina el genotipo, pudiendo heredarse los genes recesivos. La capacidad diploide de estos cromosomas proporciona un mecanismo de memoria muy útil en medios cambiantes. Sin embargo, la mayoría de Algoritmos Genéticos utilizan cromosomas haploides (un solo juego de cromosomas) porque son mucho más sencillos de construir y procesar. En la representación haploide, los genes solo se almacenan una vez por lo que no es necesario determinar que gen es el dominante y cual es el recesivo.

La interacción de dos o más genes en la formación de un fenotipo se denomina *epístasis*.

4.3.5 La reproducción

La reproducción es el proceso biológico mediante el cual se generan nuevos organismos. Existen dos tipos básicos:

1. Reproducción asexual
   Se basa fundamentalmente en el mecanismo de la mitosis, por el cual una célula se divide formando una nueva. El resultado esencial de la mitosis es la continuidad de la información hereditaria de la célula madre en cada una de las dos células hijas. La mitosis completa es un mecanismo de multiplicación celular que participa en el desarrollo, crecimiento y regeneración de un organismo. La figura 4.4 muestra este mecanismo.

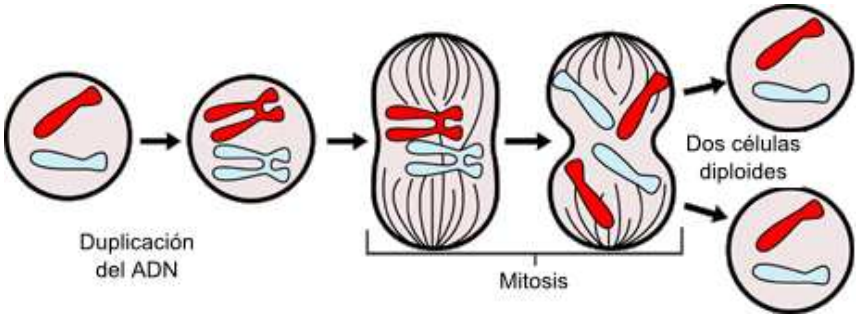


Figura 4.4: Mecanismo de la mitosis

2. Reproducción sexual
   Es el mecanismo reproductivo más habitual en los organismos pluricelulares. Se pueden definir tres tipos:

- Singamia
  Es el proceso reproductivo más complejo de todos. Consiste en la fusión de dos gametos[4] para producir un nuevo organismo. En los animales el proceso requiere de la fusión de un óvulo y un espermatozoide y finaliza con la generación de un embrión.

- Meiosis
  Es el proceso de división celular por el cual una célula diploide experimenta dos divisiones sucesivas, con la capacidad de generar cuatro células haploides. Mediante este mecanismo se producen los óvulos y espermatozoides (gametos). Este proceso se lleva a cabo en dos divisiones nucleares y citoplasmáticas, llamadas primera y segunda división meiótica o simplemente meiosis I y meiosis II.

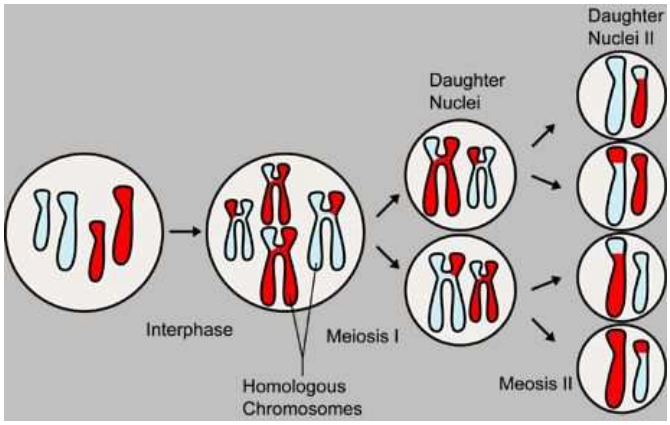


Figura 4.5: Mecanismo de la meiosis

- Recombinación genética
  Es el proceso por el cual una hebra de material genético es rota y luego unida a una molécula de material genético diferente. De entre las formas de recombinación genética, la más habitual es el entrecruzamiento cromosómico, donde se recombinan los cromosomas heredados de los padres. La recombinación puede producirse con baja probabilidad en cualquier lugar del cromosoma siendo posible la recombinación en múltiples puntos.

[4]Células sexuales

4.3.6 La selección natural

El *Origen de las Especies* se basa en el principio de preservación de los individuos mas fuertes (adaptados al medio) y la eliminación de los débiles. La selección natural puede actuar sobre cualquier rasgo fenotípico heredable y cualquier aspecto del entorno puede producir presión selectiva, esto incluye la selección sexual y la competición con miembros tanto de la misma como de otra especie.

Por lo tanto, los individuos más aptos tienen más probabilidad de contribuir con descendientes a la siguiente generación, mientras que los menos aptos tendrán menos descendientes o morirán antes de alcanzar la edad adulta. Como resultado, los alelos que en promedio conllevan mejor adaptación (aptitud) son más abundantes en la siguiente generación, mientras que los alelos que tienden a perjudicar a los individuos portadores, también tienden a desaparecer.

En cada generación se producen mutaciones y recombinaciones que producen un nuevo espectro de genotipos. Estos dos mecanismos son los responsables fundamentales del mantenimiento de la diversidad en la población. Cada nueva generación se enriquece con la abundancia de alelos que contribuyen a los rasgos que fueron anteriormente favorecidos por la selección natural, mejorando así gradualmente estos rasgos durante generaciones sucesivas.

La selección natural requiere pues tiempo y un medio relativamente estable para realizar su trabajo. Bajo estas dos premisas es uno de los mecanismos de adaptación más poderosos de la naturaleza.

4.4 Terminología de los Algoritmos Genéticos

4.4.1 La población

Es un conjunto de individuos que representan el conjunto de soluciones evaluadas durante una generación (iteración). Idealmente la primera población debería estar formada por individuos contenidos en todo el espacio de búsqueda. Es por esto que la generación de la población inicial usualmente es aleatoria para impedir una convergencia prematura hacia soluciones próximas a óptimos locales. No obstante, en determinados casos donde se conoce la proximidad del óptimo global, se suelen emplear soluciones conocidas previamente.

La determinación del tamaño de la población no es una tarea fácil. Este depende de la complejidad del problema, de modo que cuando mayor es el espacio de búsqueda, mayor debe ser tamaño de la población para evitar un estancamiento prematuro en un óptimo relativo. El tamaño típico de las poblaciones en los problemas de optimización estructural suele estar en torno a los 100 individuos.

4.4.2 Los individuos

Un individuo representa una de las soluciones del problema planteado. Dada la dualidad existente entre el genotipo y el fenotipo de los individuos estas soluciones también lo son. La solución genotípica viene representada por los cromosomas, que es con lo que trabajan los Algoritmos Genéticos.

Por otro lado la solución fenotípica representa el modelo o la estructura en que la solución genotípica se decodifica. La función de codificación, mapeo o morfogénesis que relaciona cada genotipo con su fenotipo puede no ser biyectiva. Es decir un cromosoma (genotipo) solo puede representar una solución (fenotipo), pero una solución (fenotipo) puede ser representado por varios cromosomas (genotipos). En este caso se dice que la función de morfogénesis está degenerada. Cierto grado de degeneración es inevitable en la resolución de problemas de gran tamaño pero puede ser muy perjudicial para el funcionamiento del algoritmo ya que puede conducir a una convergencia prematura y estancamiento en la población de fenotipos mientras la población de genotipos es muy diversa.

4.4.3 Los genes

Es una porción del cromosoma que generalmente codifica el valor de un solo parámetro acotado dentro de un cierto rango o dominio. La cantidad de valores que puede tomar un parámetro suele ser discreta y debe tenerse en cuenta que a mayor número más complejo tiende a ser el espacio de búsqueda. La estructura del gen depende de la función de codificación o mapeo empleada.

4.4.4 La Función de aptitud

La adaptación de cada individuo de una población al medio se realiza mediante la función de aptitud Φ o *fitness*. Para cuantificar esta adaptación o fitness primero debe decodificarse el cromosoma, ya que esta función está definida en el espacio fenotípico. El grado de adaptación de un individuo será el que determine su probabilidad de reproducción o incluso su eliminación de la población.

4.5 Definición formal de un Algoritmo Genético estándar

Dado el espacio de individuos $I \notin \{\emptyset\}$ del cual se genera inicialmente la población $\boldsymbol{\chi} \in I$ de tamaño μ, formada por los individuos: $\{a_1, \ldots, a_\mu\}$ que será evaluada por una función Φ que asigna a cada individuo una aptitud. En base a la aptitud de los individuos se seleccionará mediante determinadas reglas un conjunto de padres $\overline{\boldsymbol{\chi}} = \{\overline{a}_1, \ldots, \overline{a}_\mu\}$ los cuales engendrarán un conjunto de hijos $\boldsymbol{\chi'} = \{a'_1, \ldots, a'_\mu\}$, que sufrirán procesos de mutación que los transformarán en un nuevo conjunto $\boldsymbol{\chi''} = \{a''_1, \ldots, a''_\mu\}$, el cual tras ser evaluado por la función Φ formará la población $\boldsymbol{\chi}$ de la siguiente generación. Este último proceso se repetirá hasta que se cumpla determinada condición de terminación Ψ.

El algoritmo 1 muestra en forma de pseudocódigo del mismo.

Algoritmo 1 Pseudocódigo de Algoritmo Genético

```
t = 0
inicializar: χ(0) = {a_1(0), ..., a_μ(0)}
evaluar: χ(0) : {Φ(a_1(0)), ..., Φ(a_μ(0))}
while Ψ(Φ(t) ≠ true) do
    seleccionar: ā_k(t) = f(s_{p_s}, Φ(a_k(t)))   ∀k ∈ {1, ..., μ} ∧ ā_k ∈ χ̄
    recombinar: a'_k(t) = f(r_{p_c}, Φ(ā_k(t)))   ∀k ∈ {1, ..., μ} ∧ a'_k ∈ χ'
   mutar: a''_k(t) = p_m(a'_k(t))   ∀k ∈ {1, ..., μ} ∧ a''_k ∈ χ''
   evaluar: χ''(t) : {Φ(a''_1), ..., Φ(a''_μ)}
   seleccionar nueva pob.: a_k(t+1) = f(χ(t), χ''(t))   ∀k ∈ {1, ..., μ} ∧ a_k ∈ χ

   t= t+1
end while
return χ(t)
```

4.6 Codificación de las variables de diseño

4.6.1 Clasificación de los diferentes tipos de codificación

Desde la aparición de los Algoritmos Genéticos se han utilizado diferentes formas de codificación binaria, gray, real, hexadecimal, octal, arbórea, ...) [156, 192, 194, 315].

Inicialmente se extendió la codificación binaria, donde las variables eran codificadas en formato binario o grey, por varias razones. En primer lugar, el teorema fundamental de los Algoritmos Genéticos: el *Teorema Fundamental del Esquema* (*The Fundamental Theorem of schemata*) establece que el alfabeto binario ofrece el máximo número de *schemata*[5] por bit de información de todos los sistemas de codificación [135]. En segundo lugar esta codificación facilita el análisis teórico del algoritmo y permite desarrollar operadores de forma elegante.

Sin embargo la mejora evolutiva es independiente de la codificación, como demuestra el artículo de Antonisse [14] o como ya el propio Goldberg advertía [136]:

> ≪El empleo de codificaciones reales o de punto flotante tiene una historia controvertida en la historia de los esquemas de búsqueda genética y evolutiva, y su uso parece estar últimamente al alza. Este incremento en su uso a sorprendido a los investigadores familiarizados la teoría fundamental de los Algoritmos Genéticos, porque un análisis simple parece sugerir que el uso de alfabetos de baja cardinalidad mejora el procesamiento de esquemas, lo cual es aparentemente una contradicción directa con los resultados empíricos donde codificaciones reales han funcionado bien en un gran número de problemas prácticos.≫

Además la codificación binaria presenta una serie de desventajas que supera la codificación real (RCGA). En primer lugar, la codificación binaria requiere de la determinación a priori del número de bits necesario. Esto no supone mayor problema que la discretización del problema, pero esta discretización requiere de

[5] Es un patrón o esquema de repetido en varios cromosomas. En un cromosoma binario puede especificarse con una cadena de igual longitud del cromosoma donde cada gen puede tomar una de los valores [0,1,@] donde el símbolo @ representa un gen que no forma parte del patrón

un esfuerzo y tiempo de computación en la codificación y decodificación que influye en el rendimiento del algoritmo. Además la discretización binaria acarrea siempre un error de discretización. Para conseguir que este sea pequeño la longitud de la cadena binaria puede ser excesivamente larga. Por ejemplo, si se desease discretizar cien variables en un dominio [-500,500] con seis dígitos de precisión, la longitud del vector binario (genoma) sería de 3000 y el espacio de búsqueda de alrededor de 10^{1000}.

Los experimentos realizados por Michalewicz [263] donde comparó el desempeño en la resolución de una serie de problemas mediante codificación binaria y real le llevaron a enunciar la siguiente conclusión:

> «Los experimentos llevados a cabo indican que la codificación real es más rápida y consistente y proporciona mayor precisión (especialmente cuando los dominios son grandes, donde la representación binaria requiere de representaciones extraordinariamente largas).»

Desde entonces, numerosos trabajos [82, 85, 189, 263, 309, 403, 405, 413] han demostrado que la codificación real es computacionalmente mas eficiente que la codificación binaria o gray. Por otro lado el trabajo de Eshelman y Schaffer [109] logró la generalización del *teorema del esquema* planteado por Holland [178] dotando a la codificación entera y continua de un bagaje teórico teórico tan fuerte como el desarrollado por Holland y Goldberg

Los genes suelen representarse en forma de cadena, aunque otras representaciones como la matricial o arbórea también son posibles. La representación de cadena viene definida por un vector $\boldsymbol{s} = \{s_1, s_2, \ldots, s_j, \ldots, s_n\}$ donde s_j es el gen de la posición j. Los diferentes valores de j representan los alelos de ese gen.

La tabla 4.1 muestra un ejemplo de codificación de las dimensiones del perfil en U mostrado en la figura 4.6 de dimensiones $\boldsymbol{s} = (a, b, c, e_1, e_2, e_3) = (50, 100, 50, 3, 2, 3)$.

Los algoritmos genéticos trabajan en dos espacios diferentes de manera simultánea: el espacio de código y el de soluciones, o lo que es lo mismo en el espacio de genotipos y fenotipos. Los operadores genéticos trabajan en el espacio de genotipos mientras que la evaluación y selección tiene lugar en el espacio de fenotipos. El tipo y forma de codificación o mapeo tiene una influencia significativa en el rendimiento del algoritmo, pudiendo generar soluciones ilegales.

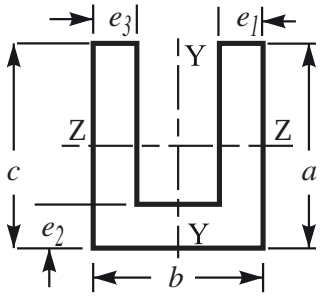


Figura 4.6: Perfil en U

Tabla 4.1: Comparativa de los diferentes tipos de codificación de un mismo conjunto de genes

Tipo de codificación	**Ejemplo** (a, b, c, e_1, e_2, e_3)
Entera	(50,100,50,3,2,3)
Real	(50.0,100.0,50.0,3.0,2.0,3.0)
Binaria	(0110010,1100100,0110010,0000011,0000010,0000011)
Hexadecimal	(32,64,32,03,02,03)
Octal	(062,144,062,003,002,003)

Es importante distinguir entre dos conceptos, que a menudo se confunden en la literatura, como son la infactibilidad (infeasible) y la ilegalidad. Una solución no factible se genera cuando esta no cumple con alguna o varias de las restricciones que se imponen al problema, es decir, de no existir estas sería una solución factible. La factibilidad de una solución depende de las restricciones y no de la codificación. En estos casos existen diversas técnicas[6] que fuerzan a los individuos a evolucionar hacia la zona de soluciones posibles [100, 130, 229, 264, 275, 422]. La figura 4.7 muestra de forma gráfica ambos conceptos.

Por otro lado una solución ilegal se genera debido a la naturaleza de la codificación. Dado que los cromosomas ilegales no pueden traducirse al espacio de fenotipos no pueden ser evaluados y por lo tanto se no puede emplear una técnica de penalización. En este caso pueden aplicarse otras técnicas como la utilización de técnicas de reparación que transforman los genotipos ilegales en genotipos legales, aunque puede que no factibles. Otra técnica consiste en comprobar la legalidad del indi-

[6]Generalmente basadas en métodos de penalización

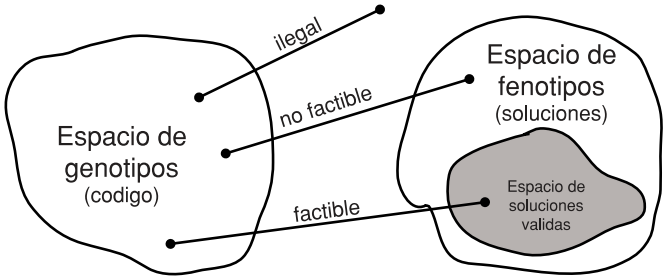


Figura 4.7: Invalidez vs ilegalidad

viduo tras su nacimiento, el cual en caso de ser ilegal será eliminado y sustituido por un nuevo individuo.

4.6.2 Propiedades de las codificaciones

Con el fin de evitar la generación de individuos ilegales la codificación debe cumplir con una serie de requisitos, según se desprende de los estudios realizados [67, 313, 314]. A continuación se detallan los principios a seguir:

1. No redundancia
   El mapeado entre individuos debe ser biyectivo (1 a 1), es decir a cada genotipo le corresponde uno y solo un fenotipo y viceversa. Cuando el mapeado es suprayectivo (n a 1), se pierde tiempo en la búsqueda ya que la misma solución se evalúa n veces porque el algoritmo considera que los individuos son diferentes.Ya por último cuando el mapeado es inversamente suprayectivo (1 a n) el algoritmo funciona de forma errática y es muy probable que no converja ya que un mismo individuo puede poseer varias aptitudes. La figura 4.8 muestra de forma gráfica los tres tipos de mapeado.

2. Legalidad
   Cualquier permutación de genes dentro del genotipo da lugar a un fenotipo. Esta propiedad permite la aplicación de los diferentes operadores genéticos.

3. Totalidad
   A cada solución le corresponde una codificación. Es decir, el mapeado no puede ser simplemente inyectivo. A cada fenotipo le debe corresponder al menos un genotipo para asegurar que el algoritmo abarca todo el espacio de búsqueda.

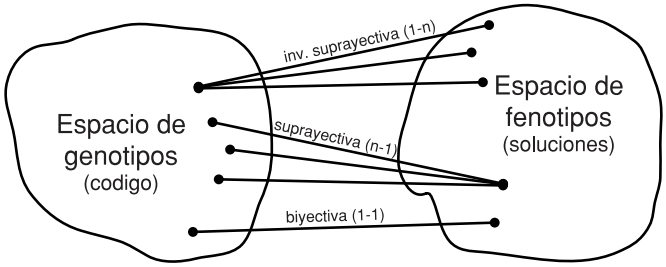


Figura 4.8: Mapeado entre individuos

4. Propiedad Lamarckiana
   El significado de los alelos de un gen no es dependiente del contexto. Expresado de otro modo, es la propiedad de los padres de transmitir información genética a su descendencia. Una representación tiene la propiedad Lamarckiana si puede pasar sus méritos a las generaciones siguientes a través de los operadores genéticos. Las codificaciones de los Algoritmos Genéticos deben ser imprescindiblemente Lamarckianas ya que de lo contrario no se formarían y transmitirían esquemas exitosos a las siguientes generaciones haciendo que el algoritmo funcione de forma errática.

5. Causalidad
   Pequeñas variaciones en el espacio de genotipos debidas a la mutación generan pequeñas variaciones en el espacio de fenotipos. Esta propiedad fue formulada por Rechenberg [313] y se centra en la conservación de estructuras vecinas. Los procesos de búsqueda que no destruyen las estructuras vecinas son denominados de *fuerte causalidad*. Una fuerte causalidad puede resultar interesante para incrementar la diversidad de la población pero por contra dificultará la convergencia del algoritmo. Sendhoff, Kreuts y Seelen desarrollaron un método que permite cuantificar esta propiedad [346].

4.7 Operadores genéticos

Son los encargados de mimetizar los procesos naturales en los *Algoritmos Evolutivos*. El número y tipo de estos da origen a los cuatro paradigmas de estos algoritmos. En el caso de los algoritmos genéticos se encuentran presentes los siguientes operadores: inicialización, reproducción (selección + recombinación) y mutación.

4.7.1 El operador de inicialización

Se trata normalmente de un operador muy sencillo que se encarga de generar la población inicial, o de reinicializar una población en determinadas condiciones. Esta inicialización se suele realizar generando el conjunto de genes de cada individuo de forma aleatoria, dentro del dominio para el cual se ha definido cada gen.

Pueden utilizarse otras técnicas heurísticas adaptadas al problema que se desea solucionar pero en general no merecen la pena por el esfuerzo computacional que ello implica. Observando la figura 4.9, donde se representa el comportamiento típico de un GA, apreciamos como en sólo k generaciones se consigue mejorar sensiblemente la aptitud desde Φ_0 a Φ_k. Además debe tenerse en cuenta que es conveniente que la población inicial esté suficientemente dispersa con el fin de abarcar todo el espacio de búsqueda.

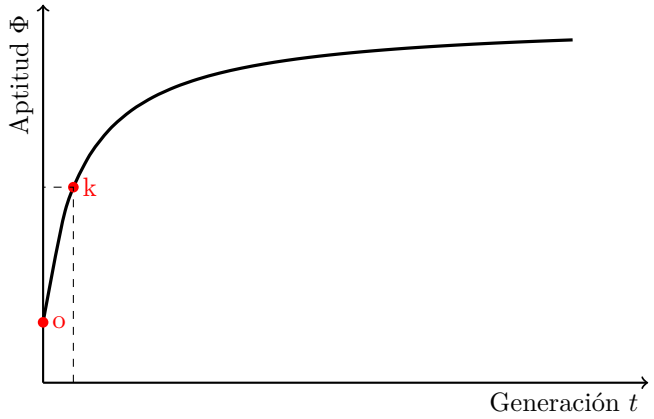


Figura 4.9: Evolución temporal típica de un Algoritmo Genético

También pueden utilizarse soluciones conocidas para inicializar la población, pero esto normalmente solo conducirá a óptimos locales: *si buscas resultados distintos no hagas siempre lo mismo*[7].

[7]Definición de demencia [13]: *es repetir los mismos errores y esperar resultados diferentes*

4.7.2 Los operadores de reproducción

Los Algoritmos Genéticos se fundamentan en el proceso de la reproducción, durante el cual el proceso de búsqueda genera individuos nuevos y mejores (más adaptados). Son estos operadores los que dominan el proceso de búsqueda, y de su selección y configuración dependerán los procesos de exploración y explotación. La exploración hace referencia a la búsqueda de soluciones más allá de las soluciones codificadas por los padres, mientras que la explotación se refiere a la búsqueda del óptimo guiada y acotada por las soluciones de los padres.

El dilema *exploración vs explotación* ya fue planteado por el propio Holland y plantea la problemática de los Algoritmos Genéticos. Una exploración demasiado intensiva hace el proceso de búsqueda lento al no generar suficientes esquemas[8], mientras que una explotación excesiva puede provocar la convergencia prematura en un óptimo relativo. Por otra parte el ajuste entre ambos es dependiente del tipo de problema a resolver por lo que debe ser establecido en cada caso [411].

El ciclo reproductivo consta de tres pasos: la selección de los padres, el cruce (recombinación) de los padres para generar nuevos individuos (hijos), y la mutación que alterará en mayor o menor medida el genoma de los nuevos hijos. Cada uno de estos pasos es implementado dentro del algoritmo con un operador diferente.

4.7.2.1 El operador de selección

La selección es el proceso mediante el cual se selecciona por diferentes procedimientos n_{sel}[9] individuos de la población (padres) para participar en el proceso de la reproducción. El conjunto de los individuos seleccionados $\overline{\boldsymbol{\chi}}$ formará los hijos $\boldsymbol{\chi}''$ tras aplicarle los operadores de cruce y mutación.

La forma en que se produce el proceso de selección la lugar a los diferentes operadores de selección [30, 33, 138, 269]:

[8] Herencia de características deseables

[9] Suele ser igual al número de individuos de la población $\boldsymbol{\chi}$. En el presente trabajo se considera en todos los casos $n_{\text{sel}}=\mu$

La selección aleatoria

Esta técnica selecciona aleatoriamente los padres de la nueva población. Presenta la ventaja de mantener la diversidad de la población y el inconveniente de hacer extremadamente lento el proceso de convergencia.

La selección por ruleta (RWS)

También conocida como selección (o muestreo) estocástica con remplazo o como selección proporcional a la aptitud. Es el uno de los operadores tradicionales. Fue introducido por De Jong [83] y ha sido extensamente estudiado desde entonces. Con este operador la probabilidad de selección de un individuo i es directamente proporcional a su aptitud Φ_i. De este modo la probabilidad de selección p_s de un individuo i en la generación t se calcula como:

$$p_{s_i}(t) = \frac{\Phi_i(t)}{\sum_{j=1}^{\mu} \Phi_j(t)} \tag{4.1}$$

El número de vecesκ_i que se espera que un individuo i sea seleccionado para la reproducción en la población $\boldsymbol{\chi}$ en la generación t viene dado por la siguiente expresión:

$$\kappa_i(t) = \mu \frac{\Phi_i(t)}{\sum_{j=1}^{\mu} \Phi_j(t)} = \mu p_{s_i}(t) \tag{4.2}$$

La implementación del operador se realiza del siguiente modo: se toma una ruleta de casino y se divide en partes proporcionales a la aptitud de cada individuo de modo que cada porción coincide con la probabilidad de selección p_s del mismo. Seguidamente se lanza la ruleta μ veces, donde μ es el número de individuos de la población, seleccionando en cada tirada un progenitor formando de este modo las parejas que engendrarán los hijos. Estos suelen ser almacenados en un vector $\overline{\boldsymbol{\chi}}$ donde los índices impares son los padres y los pares las madres. El algoritmo 2 muestra la implementación de este operador.

A pesar de ser un operador muy extendido presenta una serie de inconvenientes:

- La complejidad computacional de este operador varía entre $\mathcal{O}(\mu^2)$ y $\mathcal{O}(\mu \log \mu)$.

[10] Función que genera un número aleatorio normal gausiano comprendido en el intervalo [0,1]

Algoritmo 2 Pseudocódigo del operador de selección por ruleta

```
inicializar: χ̄ = {∅}
evaluar: p_s = {p_{s_1}, ..., p_{s_μ}}
for i=1 to μ do
  r = rnd(0,1)^10
  s = j = 0
  while s ≤ r do
    j = j + 1
    s = s + p_{s_j}
  end while
  χ̄_i = χ_j
end for
return χ̄
```

- Los mejores individuos toman el control de la población rápidamente, disminuyendo la diversidad de esta y causando una **convergencia prematura** del algoritmo hacia un óptimo local [91].

- Cuando los valores de la aptitud de los individuos están muy próximos entre sí desaparece la **presión selectiva**[11] haciendo que este se comporte como un operador de selección aleatoria. De este modo cuando el algoritmo empieza a converger y los individuos menos aptos desaparecen el algoritmo se comporta como un algoritmo de búsqueda aleatoria (random walk), donde la exploración es llevada a cabo mediante la mutación.

- Alta diversidad estocástica. La distribución de la selección de los individuos de la población no se corresponde siempre con su valor esperado κ_i provocando según que casos procesos de convergencia demasiado rápida o demasiado lenta.

De las observaciones anteriores se desprende que existen dos cuestiones fundamentales en la búsqueda del óptimo global: la diversidad de la población y la presión selectiva. Estos dos factores se encuentran fuertemente relacionados de forma inversa [409]: una fuerte presión selectiva genera una convergencia rápida del algoritmo mientras que una presión selectiva demasiado débil hace el proceso de búsqueda ineficaz. Es pues importante encontrar un equilibrio entre ambos factores, el cual depende del operador de selección empleado. La elección del operador de selección es pues vital para la resolución del problema planteado.

[11] Es el cociente entre la aptitud máxima y la media de las aptitudes de una población

La selección por muestreo universal estocástico (SUS)

También conocida como selección (o muestreo) estocástica sin remplazo. Fue definida originalmente por De Jong [83] y estudiada por Baker [18, 142] con el fin de mejorar la mala distribución de los individuos de la población en función de los valores esperados κ_i obtenidos mediante el operador de selección RWS.

La implementación del operador se realiza del siguiente modo: se mapean los individuos en segmentos contiguos de una misma línea de manera que cada segmento es proporcional en tamaño a su aptitud[12], de modo similar a la selección por ruleta. La longitud total de la línea será por tanto igual a la unidad. A continuación se colocan tantos punteros sobre la línea como individuos desean seleccionarse (generalmente μ), distribuidos de forma equidistante a una distancia de $1/\mu$. Seguidamente se genera un número aleatorio en el intervalo [0,1/μ] y partiendo del origen, se selecciona al individuo (segmento) que coincide con esa posición. A continuación se incrementa la posición en $1/\mu$ y se selecciona al siguiente individuo que coincide con dicha posición el proceso se repite μ veces hasta seleccionar todos los individuos.

La figura 4.10 muestra un ejemplo de este operador y El algoritmo 3 la implementación del mismo.

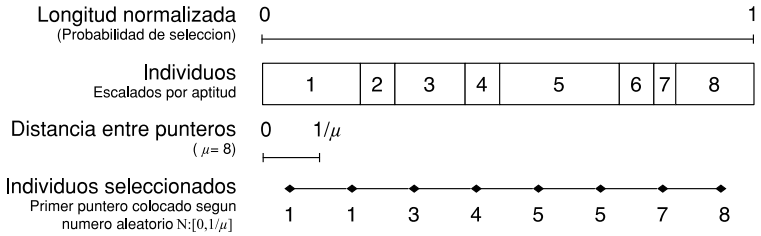


Figura 4.10: Ejemplo la selección por muestreo universal aleatorio

Este operador presenta tres ventajas fundamentales. La primera es que reduce la alta variabilidad estocástica presente en el RWS. La segunda que reduce el tiempo de computación ya que el número de selecciones necesarias es de orden [246] $\mathcal{O}(\mu)$ y no $\mathcal{O}(\mu \log \mu)$ como en el caso RWS. Por último y no menos importante, garantiza

[12]La longitud del segmento respecto a la longitud total de la línea es igual a su probabilidad de selección

Algoritmo 3 Pseudocódigo del operador de selección por muestreo universal estocástico

```
inicializar: χ̄ = {∅}
evaluar: p_s = {p_{s_1}, ..., p_{μ-1}}
i = 1
u = p_{s_1}
for j = 1 to μ do
  r = rnd[0, 1/μ]
  v = r + (j − 1)μ^{-1}
  while v > u do
    i = i + 1
    u = u + p_{s_i}
  end while
  χ̄_i = χ_j
end for
return χ̄
```

que si un individuo es suficientemente apto[13], será seleccionado incluso en varias ocasiones.

La selección por muestreo determinístico

Se trata de una variación del operador RWS. En este caso cada individuo es seleccionado una cantidad de veces igual a la parte entera del valor esperado κ_i. A continuación se ordena la población según la parte no entera del valor esperado κ_i. Los individuos restantes en el proceso de selección se determinan en función del ordenamiento anterior.

La selección por muestreo estocástico del resto con remplazo

Se trata de una variación del operador por muestreo determinístico. En este caso los individuos son también seleccionados en función de la parte entera del valor esperado κ_i. A continuación la parte no entera es utilizada para calcular la probabilidad de selección según el procedimiento de la ruleta de selección (RWS) que es utilizada para seleccionar el resto de individuos.

[13]Tiene una probabilidad de selección mayor o igual a $1/\mu$

La selección por muestreo estocástico del resto sin remplazo

Se trata de una variación del operador por muestreo determinístico. En este caso los individuos son también seleccionados en función de la parte entera del valor esperado κ_i. A continuación la parte no entera es utilizada para lanzar monedas donde la probabilidad de que salga una cara u otra[14] está ponderada por la parte no entera del valor esperado κ_i. De este modo, si por ejemplo, el número de copias esperadas de un individuo es de 1,7 este será seleccionado al menos una vez y tendrá una probabilidad del 70% de volver a ser seleccionado. El proceso es repetido hasta seleccionar el resto de individuos. El trabajo de Booker [48] parece demostrar su superioridad frente al resto de operadores estocásticos.

Selección por grupos

También llamada selección por bloques, fue propuesta por Thierens y Goldberg [387]. Cuando presión selectiva es muy baja, los operadores de selección proporcionales a la aptitud ralentizan la convergencia del algoritmo. El operador de selección por bloques intenta evitar este inconveniente. La implementación se realiza del siguiente modo: se divide la población en k grupos, asignándose una probabilidad de selección a cada grupo. La probabilidad de selección de un individuo de un grupo dado se obtiene dividiendo la probabilidad de selección del grupo por el número de individuos de dicho grupo.

La figura 4.11 muestra un ejemplo de aplicación del operador. A un 30% de la población se le asigna una probabilidad de selección del 75% mientras que al otro 70% se le asigna el 25% restante. De este modo, si la población es de 10 individuos, el primer grupo estará formado por tres individuos y la probabilidad de selección de cada uno de ellos será: $p_s = 0,75/3 = 0,25$.

La selección por rango

Fue definida originalmente por Baker [17] con el fin de superar los inconvenientes de la selección por ruleta [18]. Este operador determina la probabilidad de selección de un individuo en función del rango que este posee dentro de la población en lugar de su aptitud. La determinación del rango se realiza del siguiente modo: el elemento con peor aptitud ($\Phi_{\min} = \chi_n | f(\chi_n)$) tiene un rango de 1 y el más apto

[14] Que el individuo sea seleccionado o no

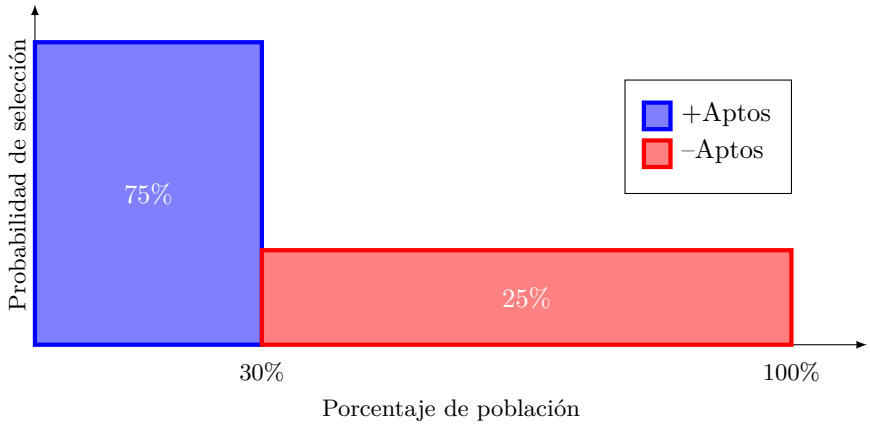


Figura 4.11: Seleccion por grupos o bloques

$(\Phi_{\max} = \chi_m | f(\chi_m))$recibe un rango μ[15]. Seguidamente se seleccionan aleatoriamente dos individuos de la población y mediante un torneo definido por alguna regla relacionada con el rango se determina que individuo será el primer padre. A continuación se vuelven a seleccionar dos individuos y aplicando la misma regla se selecciona el segundo padre. El proceso se repite hasta generar todos los hijos. Los algoritmos 4 y 5 muestran el operador siguiendo dos reglas diferentes.

Algoritmo 4 Pseudocódigo del operador de selección por rango. Regla 1

```
inicializar: χ̄ = {∅}
ordenar: χ → χ^o : {χ^o_1 = χ_m|f(χ_m) = Φ_max, ..., χ^o_μ = χ_n|f(χ_n) = Φ_min}
for i=1 to μ do
  for j=1 to 2 do
    r_j = ||μ · rnd(0,1)||          \\ Selección aleatoria de dos individuos
  end for
  r = rnd(0,1)
  if r < δ then
    χ̄_i = χ^o_{r1}
  else
    χ̄_i = χ^o_{r2}
  end if
end for
return χ̄
```

[15]Tamaño de la población

Algoritmo 5 Pseudocódigo del operador de selección por rango. Regla 2

```
inicializar: χ̄ = {∅}
ordenar: χ → χ^o : {χ^o_1 = MAX(f(χ)), ..., χ^o_μ = MIN(f(χ))}
for i=1 to μ do
  for j=1 to 2 do
    r_j = ‖μ · rnd(0,1)‖        \\ Selección aleatoria de dos individuos
  end for
  if r_1 < r_2 then
    χ̄_i = χ^o_{r_1}
  else
    χ̄_i = χ^o_{r_2}
  end if
end for
return χ̄
```

También es posible establecer reglas donde el rango se establezca en función de una relación lineal o exponencial con la aptitud del individuo. En este caso si se obtiene la relación entre la probabilidad de selección y el rango es posible aplicar el RWS con dichas probabilidades.

Baker estableció la probabilidad de selección de un individuo de la población ordenada por rango $\boldsymbol{\chi^o}$ en función del tamaño de la población μ, el número máximo de hijos del individuo más apto κ_{pla} y su rango i en la población ordenada. Para un mapeado lineal entre el rango y la aptitud, la probabilidad de selección se obtiene mediante la ecuación:

$$p_{s_i} = \frac{2 - \kappa_{\text{pla}}}{\mu + 1} + \frac{2i(\kappa_{\text{pla}} - 1)}{\mu(\mu + 1)} \tag{4.3}$$

En aquellas ocasiones en las que se requiere mayor presión selectiva se suele emplear el mapeado es exponencial desarrollado por Michalewicz [263], donde la probabilidad de selección de un individuo i de la población ordenada $\boldsymbol{\chi^o}$ viene dada por la ecuación:

$$p_{s_i} = c(1 - c)^{i-1} \tag{4.4}$$

Donde $c \in [0, 1]$ representa la probabilidad de que un individuo de rango 1 sea seleccionado. Cuanto mayor sea el valor de c, mayor será la presión selectiva.

Esta técnica presenta el inconveniente de hacer más lento el proceso de convergencia y las ventajas de prevenir una convergencia prematura, mantener la presión

selectiva cuando la varianza de la aptitud es baja así como de preservar la diversidad de la población.

La selección de Boltzmann

Utiliza la misma analogía que el recocido simulado. En este caso la temperatura se relaciona de forma inversa con la presión selectiva logrando mantener un equilibrio entre esta y la diversidad de la población. Siguiendo la distribución de Boltzmann, la probabilidad de selección de un individuo i en la generación t viene dado por la ecuación:

$$p_{si,t} = e^{-\frac{f_{\max_{t-1}} - f_i}{\hat{T}}} \tag{4.5}$$

Donde $\hat{T} = T_0 \left(1 - \alpha\right)^{\gamma}$, $T_0 \in [5, 100]$, $\alpha \in [0, 1]$, $\gamma = 1 + 100t/T$, t es el número de iteración o generación y T es el número de iteraciones o generaciones que se deja correr el algoritmo.

La selección por torneo

Propuesta por Wetzel [408], estudiada inicialmente por Brindle [50] en su tesis y popularizada por Deb y Goldberg [87, 132] es de las mas efectivas y sencillas de implementar. Se seleccionan dos o más individuos de la población de modo que compitan entre sí para ser uno de los padres de la nueva generación. El vencedor será el individuo con mayor aptitud, repitiendo este proceso hasta elegir todos los padres.

El algoritmo 6 muestra esta implementación.

Este algoritmo tiene la ventaja de no requerir la ordenación de la población en función de su aptitud, con el correspondiente ahorro computacional. Al no utilizar la aptitud[16] para realizar la selección se asegura la diversidad de la población y, por consiguiente, el mantenimiento de la presión selectiva cuyo valor esperado será :

$$\phi = 2^{n_{\text{tor}}-1} \tag{4.6}$$

Debido a sus características [31, 47] es uno de los operadores de selección más eficientes y suele conducir a la obtención de óptimos absolutos [409].

[16] Solo se utiliza para determinar si un individuo es mejor que su oponente

Algoritmo 6 Pseudocódigo del operador de selección por torneo

```
inicializar: χ̄ = {∅}
for i=1 to μ do
  for j=1 to n_tor do
    r_j = ‖μ · rnd(0,1)‖          \\ Selección aleatoria de n_tor individuos
  end for
  s = 1
  for j = 1 to n_tor do
    for k = j+1 to n_tor do
      if f_{r_j} < f_{r_k} then
        s = k                      \\ Determinación del individuo más apto
      end if
    end for
  end for
  χ̄_i = χ_{r_s}
end for
return χ̄
```

Existe una variante del operador propuesta por Michalewicz [263] basado en la selección de Boltzmann, para el torneo binario, donde los dos individuos i y j compiten entre sí y donde la probabilidad de que gane el individuo j viene determinada por la fórmula:

$$p_{w_j} = \frac{1}{1 + e^{\frac{f_i - f_j}{\tilde{T}}}} \tag{4.7}$$

Seguidamente se lanza una moneda donde la probabilidad de que salga una cara u otra está ponderada[17] según la probabilidad p_w determinada por la ecuación (4.7), seleccionando al individuo que salga de este lanzamiento. Nótese que cuando las aptitudes de ambos individuos se aproximan (convergencia del algoritmo) la probabilidad de ser el ganador se iguala. Así mismo, cuando el tiempo transcurrido es grande, la probabilidad de selección también tiende a igualarse aunque las aptitudes de los individuos sean muy diferentes. Esto último permite evitar el estancamiento de la población al permitir a individuos alejados de la zona de búsqueda entrar en el proceso de selección. En cierto modo realiza el mismo trabajo que las técnicas de escalado que se verán posteriormente.

La tabla 4.2 muestra un resumen de la complejidad computacional de los diferentes operadores de selección [138]. De la observación de esta puede deducirse que los operadores menos adecuados son los de selección por ruleta y ranking.

[17] Moneda de bernoulli

Tabla 4.2: Coste computacional de los diferentes operadores de selección

Tipo de operador de selección	Tiempo o complejidad
Ruleta	$\mathcal{O}(\mu^2)$
Muestreo universal estocástico	$\mathcal{O}(\mu)$
Muestreo estocástico del resto[18]	$\mathcal{O}(\mu)$
Muestreo determinístico	$\mathcal{O}(\mu)$
Selección por grupos	$\mathcal{O}(\mu)$
Ranking	$\mathcal{O}(\mu \ln \mu)$
Torneo	$\mathcal{O}(\mu)$

4.7.2.2 El operador de cruce

También llamado operador de recombinación o *crossover*, es el responsable de realizar el apareamiento de dos individuos[19] para generar dos o más hijos. El principio del mecanismo de cruce es muy sencillo: el apareamiento de dos individuos con ciertas características diferentes y deseables genera hijos que combinan ambas características. Las técnicas de cruce han sido ampliamente utilizadas también por los humanos durante siglos con el objeto de *mejorar* las especies animales (perros, caballos, etc...) y vegetales.

A pesar de que la reproducción asexual está presente en la naturaleza en unos pocos organismos inferiores, esta no aparece en ningún organismo superior [257, 258]. Esto es indicativo de que la recombinación sexual es una forma superior de reproducción.

Este es el principal operador de búsqueda en los algoritmos genéticos[20]. La combinación de la información genética de los padres permite realizar búsquedas en otros puntos del espacio de soluciones. Cuanto más diferentes sean los padres, tanto mayor será la influencia de este operador en el funcionamiento del algoritmo. Por lo tanto, la eficiencia de la búsqueda dependerá en gran medida de la presión selectiva y la diversidad de la población.

Aunque generalmente intervienen dos padres y se generan dos hijos, esta no es la forma exclusiva de cruce. El cruce podría extenderse a un caso más general donde intervienen varios padres [102] o se genera un solo individuo.

[18] Con o sin reemplazo
[19] Aunque sin sentido biológico, también es posible el cruce de más de dos padres [101–103]
[20] En el resto de Algoritmos Evolutivos tiene un peso inferior o incluso nulo

El operador de cruce realiza las siguientes operaciones fundamentales: primero se toma aleatoriamente un par de individuos seleccionados previamente por el operador de selección para realizar el cruce. Después se genera un número aleatorio entre 0 y 1 comparándose con la probabilidad de cruce p_c. Si este número es menor que p_c se selecciona uno o varios puntos de la cadenas de los padres y generan dos hijos mediante determinadas reglas de intercambio de genes. En caso contrario se generan los hijos de forma asexual, es decir los hijos son simples copias de los padres.

A la hora de realizar el corte puede procederse de dos formas: realizando un corte por genotipo, es decir cortando por cualquier punto de la cadena, o por fenotipo. En este segundo caso los genes se agrupan en función del fenotipo que estos definen y el corte se realiza entre estos bloques de genes. Está comprobado [62, 306] que el corte por genotipo es muy disruptivo en problemas donde existen epístasis[21] entre sus genes, generando una gran cantidad de individuos ilegales por lo que es preferida la segunda opción.

Existen diferentes métodos para definir la localización del punto (o los puntos) de corte así como la longitud del material genético a intercambiar. Ello da lugar a los diferentes operadores de cruce:

Operadores de cruce determinísticos

Dentro de esta denominación se engloban todos aquellos operadores donde los genomas de los hijos se obtienen mezclando, mediante determinadas reglas determinísticas, los genes de los padres. De este modo el gen i de un hijo se corresponde con el gen i del primer o del segundo padre.

[21] Interacción de dos o más genes en la formación de un genotipo

Cruce por un punto Es el operador de cruce más simple. Fue propuesto inicialmente por Holland [178] y estudiado por De Jong [83]. En el caso de que el corte se realice por genotipo, la implementación del algoritmo se realiza del siguiente modo: se genera un número aleatorio $r \in [1, n_g - 1]$, donde n_g es la longitud del genoma $\boldsymbol{s}$. Seguidamente se cortan los genomas (cadenas) de los padres por ese punto y se permutan, generando los hijos.

En el caso de que se desee un corte por fenotipo, la implementación se realiza del mismo modo pero sustituyendo n_g por n_f, donde n_f es el número de fenotipos presentes en un genoma.

Las figuras 4.12 y 4.13 muestran un ejemplo del funcionamiento del operador, donde las comas se han eliminado de los vectores para facilitar su lectura:

Padres	**Corte**	**Hijos**
$\boldsymbol{s}_{p_1} = \{1\ 2\ 3\ 4\ 5 \vert \mathbf{6\ 7\ 8\ 9}\}$	$\{1\ 2\ 3\ 4\ 5\ \mathbf{6\ 7\ 8\ 9}\}$	$\boldsymbol{s}_{h_1} = \{1\ 2\ 3\ 4\ 5 \vert 4\ 3\ 2\ 1\}$
$\boldsymbol{s}_{p_2} = \{9\ 8\ 7\ 6\ 5 \vert \mathbf{4\ 3\ 2\ 1}\}$	$\{9\ 8\ 7\ 6\ 5\ \mathbf{4\ 3\ 2\ 1}\}$	$\boldsymbol{s}_{h_2} = \{9\ 8\ 7\ 6\ 5 \vert 6\ 7\ 8\ 9\}$

Figura 4.12: Cruce por un punto. Corte por genotipo.

Padres	**Corte**	**Hijos**
$\boldsymbol{s}_{p_1} = \{\boxed{1\ 2\ 3}\ \boxed{4\ 5\ 6\ 7} \Vert \boxed{\mathbf{8\ 9}}\}$	$\{1\ 2\ 3\ 4\ 5\ 6\ 7\ \mathbf{8\ 9}\}$	$\boldsymbol{s}_{h_1} = \{1\ 2\ 3\ 4\ 5\ 6 \vert 2\ 1\}$
$\boldsymbol{s}_{p_2} = \{\boxed{9\ 8\ 7}\ \boxed{6\ 5\ 4\ 3} \Vert \boxed{\mathbf{2\ 1}}\}$	$\{9\ 8\ 7\ 6\ 5\ 4\ 3\ \mathbf{2\ 1}\}$	$\boldsymbol{s}_{h_2} = \{9\ 8\ 7\ 6\ 5\ 4\ 3 \vert 8\ 9\}$

Figura 4.13: Cruce por un punto. Corte por fenotipo.

Este operador tiene la desventaja de que los segmentos intercambiados se corresponden siempre con los extremos del genoma, tendiendo a mantener juntos los genes próximos a la cabeza o a la cola. Este fenómeno se conoce como sesgo posicional y sus efectos han sido analizados extensamente desde el punto de vista teórico y experimental [110, 365].

Cruce por *n* puntos La técnica de cruce por un punto puede generalizarse fácilmente a n puntos de corte de la cadena. En este caso se permutan los genes comprendidos entre los puntos de corte.

Cuando el corte se realiza por genotipo es posible realizar tantos puntos de corte n como n_g - 1 genes posee el genoma. Cuando el corte se realiza por fenotipo, es posible realizar tantos puntos de corte n como n_f - 1 genes posee el genoma.

Las figuras 4.14 y 4.15 muestran un ejemplo del funcionamiento del operador, donde las comas se han eliminado de los vectores para facilitar su lectura:

Padres	Corte	Hijos
$\boldsymbol{s}_{p_1} = \{1\ 2 \vert \mathbf{3\ 4\ 5} \vert 6\ 7\ 8\ 9\}$	$\{1\ 2 \quad 6\ 7\ 8\ 9\}$ $\mathbf{3\ 4\ 5}$	$\boldsymbol{s}_{h_1} = \{1\ 2 \vert 7\ 6\ 5 \vert 6\ 7\ 8\ 9\}$
$\boldsymbol{s}_{p_2} = \{9\ 8 \vert \mathbf{7\ 6\ 5} \vert 4\ 3\ 2\ 1\}$	$\{9\ 8 \quad 4\ 3\ 2\ 1\}$ $\mathbf{7\ 6\ 5}$	$\boldsymbol{s}_{h_2} = \{9\ 8 \vert 3\ 4\ 5 \vert 4\ 3\ 2\ 1\}$

Figura 4.14: Cruce por dos puntos. Corte por genotipo.

Padres	Corte	Hijos
$\boldsymbol{s}_{p_1} = \{\boxed{1\ 2\ 3} \Vert \boxed{\mathbf{4\ 5\ 6\ 7}} \Vert \boxed{8\ 9}\}$	$\{1\ 2\ 3 \quad 8\ 9\}$ $\mathbf{4\ 5\ 6\ 7}$	$\boldsymbol{s}_{h_1} = \{1\ 2\ 3 \vert 6\ 5\ 4\ 3 \vert 8\ 9\}$
$\boldsymbol{s}_{p_2} = \{\boxed{9\ 8\ 7} \Vert \boxed{\mathbf{6\ 5\ 4\ 3}} \Vert \boxed{2\ 1}\}$	$\{9\ 8\ 7 \quad 2\ 1\}$ $\mathbf{6\ 5\ 4\ 3}$	$\boldsymbol{s}_{h_2} = \{9\ 8\ 7 \vert 4\ 5\ 6\ 7 \vert 2\ 1\}$

Figura 4.15: Cruce por dos puntos. Corte por fenotipo.

Este tipo de cruce tiene la ventaja de combinar un mayor número de esquemas y que los segmentos intercambiados no se correspondan con los extremos del genoma si n es par. Para entender esta ventaja imaginemos que los genomas de las figuras 4.13 y 4.15 se corresponden con las dimensiones de una estructura de tres barras donde cada bloque representa a cada barra y que se desea realizar una optimización del peso del conjunto. Supongamos también que el proceso de optimización se encuentra avanzado y que la primera y última barra han alcanzado su peso óptimo pero no así la central en el primer progenitor, mientras que para el segundo la central ha alcanzado el óptimo y las otras dos no. Si se realiza el cruce con un solo corte se permutará la información de la primera y la segunda barra o bien de la segunda y la tercera pero no de la segunda barra que es donde debe centrarse el proceso de búsqueda prolongando por lo tanto dicho proceso.

Por otro lado, el corte por genotipo es altamente disruptivo en problemas donde existen epístasis entre sus genes [62]. Para entender en que consiste la disrupción volvamos al ejemplo anterior y comparemos el corte por genotipo y por fenotipo. Si observamos el fenotipo, cada barra representa un bloque de información del cual se puede extraer un esquema. Si el corte se realiza por fenotipo será posible que el algoritmo combine las características deseables de los padres y forme un esquema, mientras que si se realiza por genotipo no será posible la formación de esquemas y el algoritmo no funcionará adecuadamente. La principal desventaja de este operador consiste en que tiende a mantener juntos los genes vecinos, al igual que el operador de cruce por un punto, produciendo también por tanto un sesgo posicional.

Cruce uniforme Este operador fue desarrollado por Syswerda [382]. Su funcionamiento es diferente a los anteriores. En este caso se genera una máscara binaria de forma aleatoria de igual longitud que el genoma de los padres n_g en el caso de realizar el cruce por genotipo o de longitud igual al número de fenotipos n_f en el caso de caso de hacer el cruce por fenotipo. Seguidamente se realiza el cruce siguiendo esta máscara, eligiendo para el primer hijo al primer padre cuando en la máscara hay un 1 y al segundo cuando hay un 0 y viceversa con el segundo hijo[22]. Este operador es una generalización de los anteriores.

Las figuras 4.16 y 4.17 muestran un ejemplo del funcionamiento del operador, donde las comas se han eliminado de los vectores para facilitar su lectura.

Padres	**Hijos**
$\boldsymbol{s}_{p_1} = \{1\,2\,3\,4\,5\,6\,7\,8\,9\}$	$\boldsymbol{s}_{h_1} = \{1\,2\,7\,4\,5\,4\,7\,2\,9\}$
$\{1\,1\,0\,1\,0\,0\,1\,0\,1\}$	
$\boldsymbol{s}_{p_2} = \{9\,8\,7\,6\,5\,4\,3\,2\,1\}$	$\boldsymbol{s}_{h_2} = \{9\,8\,3\,6\,5\,6\,3\,8\,1\}$

Figura 4.16: Cruce uniforme. Corte por genotipo.

Este operador tiene la ventaja de no presentar un sesgo posicional ya que tiene una fuerte tendencia a mezclar los genes vecinos[23].

[22] Esta implementación es equivalente a lanzar una moneda para cada gen o grupo de genes eligiendo al primer o segundo padre según salga cara o cruz

[23] La probabilidad de cruce de un gen vecino es del 50%

Padres	Hijos
$\boldsymbol{s}_{p_1} = \{\boxed{1\ 2\ 3}\|\boxed{\mathbf{4\ 5\ 6\ 7}}\|\boxed{8\ 9}\}$	$\boldsymbol{s}_{h_1} = \{1\ 2\ 3\ 6\ 5\ 4\ 3\ 2\ 1\}$
$\{\ 1\ \ \ 0\ \ \ 0\ \}$	
$\boldsymbol{s}_{p_2} = \{\boxed{9\ 8\ 7}\|\boxed{\mathbf{6\ 5\ 4\ 3}}\|\boxed{2\ 1}\}$	$\boldsymbol{s}_{h_2} = \{9\ 8\ 7\ 4\ 5\ 6\ 7\ 8\ 9\}$

Figura 4.17: Cruce uniforme. Corte por fenotipo.

Cruce plano Este operador fue definido por Radcliffe [309] con la idea de minimizar las disrupciones producidas por el operador de cruce uniforme. Para ello se basó en el concepto de *schemata* (esquemas) binarios planteado por Holland.

Un *schema* es un patrón común entre dos o más individuos de una población de modo que si seleccionamos dos padres de la población p.e. $\boldsymbol{s}_{p1} = \{10111001\}$ y $\boldsymbol{s}_{p1} = \{11011101\}$ su forma es $\mathcal{F}\{1\#\#11\#01\}$. La implementación del operador se realiza entonces del siguiente modo:

1. Seleccionar aleatoriamente dos padres y determinar su *Forma* $\mathcal{F}$
2. Buscar los huecos (#) de la *Forma* $\mathcal{F}$ y reemplazarlos por un número aleatorio $r \in [0, 1]$
3. Repetir el proceso hasta obtener el número de hijos deseado.

Este algoritmo es mejor que los anteriores porque evita la disrupción y amplia el espacio de búsqueda del operador, pero presenta el inconveniente de requerir una codificación binaria ya que no es extensible a la extrapolación del concepto de esquema a $\mathbb{R}$. Aunque por su comportamiento binario tenga una clasificación de operador determinístico, al decodificar las variables a una representación entera o de punto flotante el operador no se comporta de forma determinística como ilustra la figura 4.27.

Operadores de cruce aritmético

Los operadores de cruce anteriores son adecuados para codificaciones binarias o enteras, pero no así para codificaciones de punto flotante. Este operador, junto con los siguientes se desarrollaron para la codificación de este tipo de variables.

Este operador se fundamenta en una analogía con la teoría de conjuntos convexos [26], donde de forma general la media ponderada de dos vectores se calcula

mediante la expresión:

$$\boldsymbol{x}' = \lambda_1 \boldsymbol{x}_1 + \lambda_2 \boldsymbol{x}_2 \tag{4.8}$$

Si además los multiplicadores son restringidos a la condición: $\lambda_1 + \lambda_2 = 1$, la ecuación (4.8) dará lugar a una *combinación afín*. Si se añade la condición de que ambos multiplicadores sean positivos: $\lambda_1 > 0,\ \lambda_2 > 0$, dará lugar a una *combinación convexa*. Si por último, añadimos la condición de que ambos multiplicadores se encuentren en el dominio $\mathbb{R}^+$: $\lambda_1, \lambda_2 \in \mathbb{R}^+ - \{0\}$, dará lugar a una *combinación lineal* de ambos vectores. Aplicando la primera condición (4.8) puede simplificarse obteniendo la siguiente expresión:

$$\boldsymbol{x}' = \lambda_1 \boldsymbol{x}_1 + (1 - \lambda_1) \boldsymbol{x}_2 \tag{4.9}$$

Siguiendo la analogía de los conjuntos convexos, el cruce puede realizarse siguiendo las mismas combinaciones descritas anteriormente, según la siguiente expresión propuesta por Radcliffe [309]:

$$\begin{aligned} \boldsymbol{s}_{h1} &= \lambda_1 \boldsymbol{s}_{p_1} + (1 - \lambda_1) \boldsymbol{s}_{p_2} \\ \boldsymbol{s}_{h2} &= \lambda_1 \boldsymbol{s}_{p_2} + (1 - \lambda_1) \boldsymbol{s}_{p_1} \end{aligned} \tag{4.10}$$

Dependiendo de los valores que tome λ_1 tendremos diferentes tipos de operadores de cruce aritmético:

- Si $\lambda_1 \notin [0, 1]$ el operador de cruce será afín. Un caso particular de operador afín es el estudiado por Wright [413], donde $\lambda_1 = 1,5\ ,\ \lambda_2 = -0,5$. Otro tipo de operador afín es el propuesto por Mühlenbein y Schlierkamp-Voosen [274], denominado ***cruce de línea extendida***, donde λ_1 es el resultante de un número aleatorio r generado en el intervalo: $r \in [-d, 1+d]$ donde d suele tomar el valor de 0,25 .

- Si $\lambda_1 \in]0, 1]$ el operador de cruce será convexo.

- Si $\lambda_1, \lambda_2 \in \{\mathbb{R}^+ - \{0\}\} : 0 \leq \lambda_1, \lambda_2 \leq 1$, el operador de cruce será lineal. Nótese que si: $\lambda_1 = 0$ ó $\lambda_1 = 1$, no se produce recombinación heredando cada hijo el genoma de uno de los padres de modo similar a la reproducción asexual. El valor de λ_1 suele elegirse de manera aleatoria, de modo que $\lambda_1 = \text{rnd}(0,1)$. Este operador se suele denominar ***cruce heurístico*** y fue propuesto inicialmente por Wright.

También se utilizan otros métodos como el propuesto por Davis [82] y Schwefel [345] al que llamaron ***cruce intermedio*** o ***cruce promedio***. Otros autores como Eiben y Smith [104] lo denominan ***recombinación uniforme aritmética***. En caso de que $\lambda_1 = 0,5$ los genomas resultantes se corresponden con la media aritmética de sus padres. Este método ha demostrado ser bueno en varios problemas interesantes [263].

Dependiendo del número de genes combinados, se definen tres tipos de recombinación aritmética para los operadores descritos anteriormente:

Recombinación simple La implementación se realiza del siguiente modo: se selecciona un gen aleatorio i del genoma (o del fenotipo). A continuación, para el primer hijo, se copia el genoma del primer padre hasta el gen i formando el resto del genoma mediante alguno de los operadores anteriormente descritos. Para el segundo hijo es generado del mismo modo pero invirtiendo los genomas de los padres.

La ecuación (4.11) y la figura 4.18 muestran la implementación de este operador.

$$\begin{aligned}
\boldsymbol{s}_{p1} &= \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \\
\boldsymbol{s}_{p2} &= \{q_1, q_2, \ldots, q_i, \ldots, q_n\} \\
\boldsymbol{s}_{h1} &= \{p_1, p_2, \ldots, p_i, \lambda_1 q_{i+1} + (1-\lambda_1)p_{i+1}, \ldots, \lambda_1 q_n + (1-\lambda_1)p_n\} \\
\boldsymbol{s}_{h2} &= \{q_1, q_2, \ldots, q_i, \lambda_1 p_{i+1} + (1-\lambda_1)q_{i+1}, \ldots, \lambda_1 p_n + (1-\lambda_1)q_n\}
\end{aligned} \tag{4.11}$$

Padres	**Hijos**
$\boldsymbol{s}_{p_1} = \{0,1\ \ 0,2\ \ 0,3\ \ 0,4\ \ 0,5\ \ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,1\ \ 0,2\ \ 0,3\ \ 0,4\ \ 0,35\ \ 0,35\}$
$\boldsymbol{s}_{p_2} = \{0,6\ \ 0,5\ \ 0,4\ \ 0,3\ \ 0,2\ \ 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,6\ \ 0,5\ \ 0,4\ \ 0,3\ \ 0,35\ \ 0,35\}$

Figura 4.18: Cruce aritmético lineal con recombinación simple. $\lambda_1 = 0,5$, $i = 4$.

Recombinación aritmética simple La implementación se realiza del siguiente modo: se selecciona un punto aleatorio i del genotipo (o del fenotipo). A continuación, para el primer hijo, se copia todo el genoma del primer padre con excepción del gen i el cual se formará mediante alguno de los operadores anteriormente descritos. Para el segundo hijo es generado del mismo modo pero invirtiendo los genomas de los padres.

La ecuación (4.12) y la figura 4.19 muestran la implementación de este operador.

$$\begin{aligned} \boldsymbol{s}_{p1} &= \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \\ \boldsymbol{s}_{p2} &= \{q_1, q_2, \ldots, q_i, \ldots, q_n\} \\ \boldsymbol{s}_{h1} &= \{p_1, p_2, \ldots, p_{i-1}, \lambda_1 q_i + (1-\lambda_1)p_i, p_{i+1}, \ldots, p_n\} \\ \boldsymbol{s}_{h2} &= \{q_1, q_2, \ldots, q_{i-1}, \lambda_1 p_i + (1-\lambda_1)q_i, p_{i+1}, \ldots, q_n\} \end{aligned} \tag{4.12}$$

Padres	**Hijos**
$\boldsymbol{s}_{p_1} = \{0,1 \;\; 0,2 \;\; 0,3 \;\; 0,4 \;\; 0,5 \;\; 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,1 \;\; 0,2 \;\; 0,3 \;\; 0,35 \;\; 0,5 \;\; 0,6\}$
$\boldsymbol{s}_{p_2} = \{0,6 \;\; 0,5 \;\; 0,4 \;\; 0,3 \;\; 0,2 \;\; 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,6 \;\; 0,5 \;\; 0,4 \;\; 0,35 \;\; 0,2 \;\; 0,1\}$

Figura 4.19: Cruce aritmético lineal con recombinación aritmética simple. $\lambda_1 = 0,5$, $i = 4$.

Recombinación aritmética completa En este caso todos los genes de los hijos se forman mediante la combinación de ambos padres tal y como describe la ecuación (4.10). Debe tenerse especial cuidado de no realizar un cruce intermedio ($\lambda_1 = 0,5$) ya que en este caso ambos hijos son iguales, pudiendo provocar la pérdida de diversidad en la población.

La ecuación (4.13) y la figura 4.20 muestran la implementación de este operador.

Nótese como en el ejemplo de la figura 4.20 ambos descendientes son iguales. Si se empleara este operador el 50% de la población seria prácticamente igual[24] al otro 50% lo cual acarrearía una pérdida de diversidad importante.

[24] El operador de mutación puede introducir ligeras diferencias

$$\begin{aligned} \boldsymbol{s}_{p1} &= \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \\ \boldsymbol{s}_{p2} &= \{q_1, q_2, \ldots, q_i, \ldots, q_n\} \\ \boldsymbol{s}_{h1} &= \{\lambda_1 q_1 + (1-\lambda_1)p_1, \ldots, \lambda_1 q_n + (1-\lambda_1)p_n\} \\ \boldsymbol{s}_{h2} &= \{\lambda_1 p_1 + (1-\lambda_1)q_1, \ldots, \lambda_1 p_n + (1-\lambda_1)q_n\} \end{aligned} \tag{4.13}$$

Padres	Hijos
$\boldsymbol{s}_{p_1} = \{0,1\ \ 0,2\ \ 0,3\ \ 0,4\ \ 0,5\ \ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,35\ \ 0,35\ \ 0,35\ \ 0,35\ \ 0,35\ \ 0,35\}$
$\boldsymbol{s}_{p_2} = \{0,6\ \ 0,5\ \ 0,4\ \ 0,3\ \ 0,2\ \ 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,35\ \ 0,35\ \ 0,35\ \ 0,35\ \ 0,35\ \ 0,35\}$

Figura 4.20: Cruce aritmético lineal con recombinación aritmética completa. $\lambda_1 = 0,5$.

Cruce geométrico

Fue definido originalmente por Michalewicz et al. [267]. En este caso solo se genera un hijo por cada pareja cuyos genes se forman mediante la combinación cuadrática de ambos padres tal y como describe la ecuación (4.14). La figura 4.21 muestra un ejemplo de aplicación de este operador.

$$\begin{aligned} \boldsymbol{s}_{p1} &= \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \\ \boldsymbol{s}_{p2} &= \{q_1, q_2, \ldots, q_i, \ldots, q_n\} \\ \boldsymbol{s}_{h} &= \{\sqrt{p_1 q_1} \ldots, \sqrt{p_i q_i}, \ldots, \sqrt{p_n q_n}\} \end{aligned} \tag{4.14}$$

Padres	Hijo
$\boldsymbol{s}_{p_1} = \{0,1\ \ 0,2\ \ 0,3\ \ 0,4\ \ 0,5\ \ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,2\ \ 0,3\ \ 0,3\ \ 0,3\ \ 0,3\ \ 0,2\}$
$\boldsymbol{s}_{p_2} = \{0,6\ \ 0,5\ \ 0,4\ \ 0,3\ \ 0,2\ \ 0,1\}$	

Figura 4.21: Cruce geométrico.

Cruce por mezcla alfa (BLX-α)

Fue definido originalmente por Eshelman y Shaffer. Dados los genomas de los padres $\boldsymbol{s}_{p_1} = \{p_1 \ldots, p_i, \ldots, p_n\}$ y $\boldsymbol{s}_{p_2} = \{q_1, \ldots, q_i, \ldots, q_n\}$ la distancia rectilínea o de Manhattan entre los genes de los padres es:

$$d_i = |p_i - q_i| \tag{4.15}$$

El algoritmo utiliza esta distancia para generar los genes de los hijos de forma aleatoria dentro del intervalo $[L_i, U_i]$ a partir de los valores proporcionados por la

ecuación (4.16).

$$\begin{aligned} L_i &= \min(p_i, q_i) - \alpha d_i \\ U_i &= \max(p_i, q_i) + \alpha d_i \end{aligned} \tag{4.16}$$

El valor de α representa la distancia, más allá de los límites establecidos por los genes de los padres, que los genes de los hijos pueden tomar. Este método permite por lo tanto que los genes de los hijos tomen valores más allá del rango de sus padres pero sin permitir tampoco que estén excesivamente alejados. El valor $\alpha = 0,5$ es el más utilizado normalmente, pero es dependiente del problema a resolver, por lo que debe ajustarse de algún modo en cada problema.

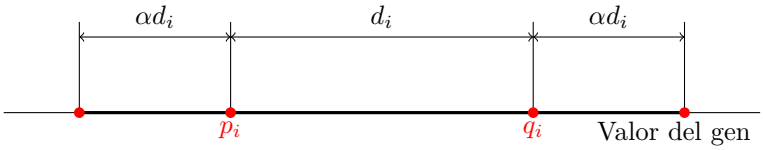


Figura 4.22: Cruce de mezcla del gen i de un solo gen

La figura 4.22 muestra un cruce por mezcla del gen i. El valor αd_i se corresponde con la exploración que puede realizar el operador, mientras que el valor d_i se corresponde con la explotación. Para un valor $\alpha = 1$ no hay exploración, solo hay explotación. El valor α es pues el responsable de la exploración de este operador de cruce

Este operador es una generalización del operador de cruce aritmético, donde el parámetro λ varía para cada gen de modo que para el gen i tomará el valor $\lambda_i \in \text{rnd}(L_i, U_i)$. Una vez calculado el valor λ_i cada gen de los hijos se formará según la ecuación (4.10).

La figura 4.23 muestra un ejemplo de la implementación de este operador.

Según la crítica realizada por Deb y Agrawal [85] este operador tiene el inconveniente de no respetar siempre los esquemas (características deseables) de los padres a la hora de generar a los hijos, con lo cual en determinados casos puede ralentizar el proceso de convergencia.

Padres	Hijos
$\boldsymbol{s}_{p_1} = \{0,1\ \ 0,2\ \ 0,3\ \ 0,4\ \ 0,5\ \ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,8\ \ 0,3\ \ 0,5\ \ 0,4\ \ 0,8\ \ 0,6\}$
$\boldsymbol{s}_{p_2} = \{0,6\ \ 0,5\ \ 0,4\ \ 0,3\ \ 0,2\ \ 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,2\ \ 0,5\ \ 0,4\ \ 0,4\ \ 0,6\ \ 0,6\}$
max,min	**X,Y**
$\{0,6\ \ 0,5\ \ 0,4\ \ 0,4\ \ 0,5\ \ 0,6\}$	$\{-0,1\ \ 0,2\ \ 0,3\ \ 0,3\ \ 0,2\ \ 0,1\}$
$\{0,1\ \ 0,2\ \ 0,3\ \ 0,3\ \ 0,2\ \ 0,1\}$	$\{\ \ 1,1\ \ 0,8\ \ 0,5\ \ 0,5\ \ 0,8\ \ 1,1\}$

Figura 4.23: Cruce aritmético lineal con recombinación aritmética completa. $\lambda_1 = 0,5$.

Cruce binario simulado (SBX-β)

Fue propuesto inicialmente por Deb [85, 86, 88] con el objetivo de simular el comportamiento del operador de cruce por un punto en codificaciones binarias y de diseñar un operador para codificaciones reales[25] que respetara la construcción de esquemas a diferencia del operador BLX-α.

En su trabajo define un factor β denominado *factor de propagación* que mide la propagación de los genes de los hijos respecto a los de sus padres y que se calcula, para un gen i, como el cociente entre la distancia entre los genes de los hijos generados dividido por la distancia entre los genes de los padres.

Dados los padres $\boldsymbol{s}_{p_1} = \{p_1, \ldots, p_i, \ldots, p_n\}$ y $\boldsymbol{s}_{p_2} = \{q_1, \ldots, q_i, \ldots, q_n\}$ y los hijos que deseamos obtener $\boldsymbol{s}_{h1} = \{u_1, u_2, \ldots, u_i, \ldots, u_n\}$ y $\boldsymbol{s}_{h2} = \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$ el factor de propagación β_i se calcula como:

$$\beta = \left| \frac{\boldsymbol{s}_{p1} - \boldsymbol{s}_{p2}}{\boldsymbol{s}_{h1} - \boldsymbol{s}_{h2}} \right| \tag{4.17}$$

Este factor clasifica los operadores de cruce en tres tipos:

1. Si $\beta < 1$ el operador de cruce es contractivo, es decir los genes de los hijos se encuentran localizados entre los padres, por lo que el algoritmo se centrará en la explotación.

2. Si $\beta > 1$ el operador de cruce es expansivo, es decir los genes de los padre se encuentran localizados entre los genes de los hijos, lo cual se traduce en que estos explorarán zonas donde no alcanzaban sus padres por lo que el algoritmo se centrará en la exploración.

[25] Los términos real, continuo o de coma flotante suelen emplearse como sinónimos

3. Si $\beta = 1$ el operador de cruce es estacionario, es decir los genes de los hijos son los mismos que los de los padres padres.

Analizando la probabilidad de obtener un comportamiento contractivo, estacionario o expansivo se obtuvieron unas ecuaciones de ajustes polinómicas, para el cruce por un punto, que proporcionan la distribución de probabilidad de obtener un cruce estacionario o expansivo.

De este modo la distribución de probabilidad propuesta por Deb viene dada por la ecuación (4.18).

$$P(\beta) = \begin{cases} 0,5(\eta+1)\beta^{\eta} & \beta \leq 1 \\ 0,5(\eta+1)\dfrac{1}{\beta^{\eta+2}} & \beta > 1 \end{cases} \tag{4.18}$$

donde $\eta \in \mathbb{R}^+$ determina la probabilidad de generar nuevos individuos próximos o alejados de sus padres. Valores de η pequeños generan hijos muy alejados de sus padres mientras que valores de η grandes los generarán muy próximos a ellos. Para valores $2 \leq \eta \leq 5$ el comportamiento del operador será similar al de un operador de cruce simple. $\eta = 1$ genera cruces tanto expansivos como contractivos, siendo por lo tanto el valor usualmente empleado.

Seguidamente, la implementación se realiza del siguiente modo: primero se genera un número aleatorio $r \in$ rnd(0,1). A continuación se calcula el valor de β que tiene la misma probabilidad acumulada que r mediante la ecuación (4.19).

$$\beta = \begin{cases} 2r^{\frac{1}{\eta+1}} & r \leq 0,5 \\ \left[\dfrac{1}{2(1-r)}\right]^{\frac{1}{\eta+1}} & r > 0,5 \end{cases} \tag{4.19}$$

Una vez determinado β_{q_i} los hijos se calculan mediante la ecuación (4.20).

$$\begin{aligned} \boldsymbol{s}_{h1} &= 0,5[(1+\beta)\boldsymbol{s}_{p1} + (1-\beta)\boldsymbol{s}_{p2}] \\ \boldsymbol{s}_{h2} &= 0,5[(1-\beta)\boldsymbol{s}_{h1} + (1+\beta)\boldsymbol{s}_{h2}] \end{aligned} \tag{4.20}$$

Las figuras 4.24 y 4.25 muestran un ejemplo expansivo y contractivo generados por diferentes valores del número aleatorio r.

Padres	Hijos
$\boldsymbol{s}_{p_1} = \{0,1\ 0,2\ 0,3\ 0,4\ 0,5\ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,0\ 0,2\ 0,3\ 0,4\ 0,5\ 0,7\}$
$\boldsymbol{s}_{p_2} = \{0,6\ 0,5\ 0,4\ 0,3\ 0,2\ 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,7\ 0,5\ 0,4\ 0,3\ 0,2\ 0,0\}$

Figura 4.24: Cruce binario simulado $r = 0,4$, $\eta = 1 (\beta = 1,27)$.

Padres	Hijos
$\boldsymbol{s}_{p_1} = \{0,1\ 0,2\ 0,3\ 0,4\ 0,5\ 0,6\}$	$\boldsymbol{s}_{h_1} = \{0,2\ 0,3\ 0,3\ 0,4\ 0,4\ 0,5\}$
$\boldsymbol{s}_{p_2} = \{0,6\ 0,5\ 0,4\ 0,3\ 0,2\ 0,1\}$	$\boldsymbol{s}_{h_2} = \{0,5\ 0,4\ 0,4\ 0,3\ 0,3\ 0,2\}$

Figura 4.25: Cruce binario simulado $r = 0,1$, $\eta = 1 (\beta = 0,63)$.

Operadores de cruce multiparentales

Cruce unimodal de distribución normal (UNDX-m) Fue propuesto por Ono y Kobayashi [284–286] para una combinación de 3 padres y ampliado por Kita [212, 213] a m padres. Fue diseñado con la idea de desarrollar un operador robusto que soportara epístasis entre sus genes ya que ninguno de los anteriores se comporta bien cuando esta se hace presente. En estos casos, los operadores de cruce tradicionales dificultan la formación de esquemas provocando lo que Holland denominó como *el engaño* que se traduce en una falta de convergencia o la caída en un óptimo local.

El procedimiento del operador es el siguiente:

1. Seleccionar $m-1$ padres $\boldsymbol{s}_{p1}, \ldots, \boldsymbol{s}_{p_{m-1}}$ aleatoriamente de la población.
2. Calcular el centro de masas de esos padres:

$$\overline{\boldsymbol{s}} = \frac{1}{m-1} \sum_{i=1}^{m-1} \boldsymbol{s}_{p_i} \tag{4.21}$$

3. Calcular el subespacio vectorial formado por los vectores resultantes de la diferencia entre los padres seleccionados y el dentro de masas:

$$\boldsymbol{d}_i = \boldsymbol{s}_{p_i} - \overline{\boldsymbol{s}}\ \forall i \in \{1, \ldots, m-1\}$$

Este subespacio vectorial constituye la dirección primaria de búsqueda.

4. Seleccionar aleatoriamente otro padre $\boldsymbol{s}_{pm}$ de la población restante y calcular su distancia subespacio vectorial $\boldsymbol{d}_i$: $D = |\boldsymbol{s}_{p_m} - \overline{\boldsymbol{s}}|$.

5. Calcular la base ortonormal $\boldsymbol{e}_j \quad \forall j \in \{1, \ldots, \mu - m\}$ que es ortogonal a $\boldsymbol{d}_i \; \forall i \in \{1, \ldots, m\}$. El subespacio vectorial generado por esta base constituye la dirección secundaria de búsqueda.

6. Generar los hijos siguiendo la expresión:

$$\boldsymbol{s}_h = \overline{\boldsymbol{s}} + \sum_{i=1}^{m} \omega_i \boldsymbol{d}_i + \sum_{j=1}^{\mu-m} \upsilon_j D \boldsymbol{e}_j \tag{4.22}$$

   donde: ω_i y υ_i son números aleatorios normales que siguen las distribuciones $N(0, \sigma_\xi^2)$ y $N(0, \sigma_\eta^2)$ respectivamente y σ_ξ y σ_η son parámetros que controlan la extensión de la búsqueda.

   Los valores propuestos para σ_ξ y σ_η por Kita para preservar la matriz de covarianza de la población de padres $\overline{\boldsymbol{\chi}}$, confirmados por los resultados experimentales son: $\sigma_\xi = 1/\sqrt{m}$ y $\sigma_\eta = 0,35\sqrt{\mu - m}$.

   El valor de m que mejores resultados ha dado empíricamente ha sido $m = 4 \sim 5$.

   La figura 4.26 muestra el funcionamiento del operador con tres padres. En ella se ha simplificado la representación de los vectores de los padres y el hijo por puntos para facilitar la interpretación de la misma. El vector $\overline{\boldsymbol{s}}$ se corresponde con el primer término de la ecuación (4.22). La componente $\boldsymbol{u}$ se corresponde con el segundo término y representa la dirección primaria de búsqueda, mientras que la componente $\boldsymbol{v}$ ortogonal se corresponde con el tercer término y representa la dirección secundaria de búsqueda. La tonalidad del color indica la probabilidad de tener un hijo en esa zona. Esta probabilidad viene definida por σ_ξ y σ_η.

Cruce parento-céntrico (PCX) Fue desarrollado por Deb et al. [90] y es una variación del UNDX-m donde en lugar de generar los hijos alrededor del centro de masas de los padres seleccionados, estos son generados alrededor de los padres.

El procedimiento del operador es el siguiente:

1. Para cada hijo $\boldsymbol{s}_{hi}$ seleccionar aleatoriamente un padre $\boldsymbol{s}_p$ y calcular la distancia de este al centro de gravedad: $d_p = \boldsymbol{s}_p - \overline{\boldsymbol{s}}$.

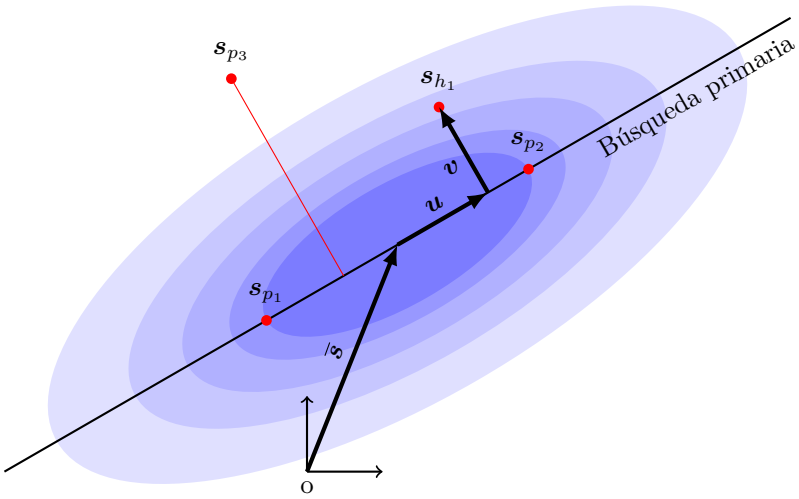


Figura 4.26: Cruce unimodal de distribución normal UNDX-m

2. Calcular, para el resto de $\mu-1$ padres, la distancias D_i $\forall i \neq p, i \in \{0,\ldots,\mu-1\}$ mínimas[26] a la recta d_p.

3. Calcular el valor de la distancia promedio $\overline{D}$.

4. Calcular la base ortonormal $\boldsymbol{e}_i$ $\forall i \neq p, i \in \{0,\ldots,\mu-1\}$ que es ortogonal a $\boldsymbol{d}_p$.

5. Generar los hijos siguiendo la expresión:

$$\boldsymbol{s}_h = \boldsymbol{s}_p + \omega_i|\boldsymbol{d}_p| + \sum_{i=1,i\neq p}^{\mu} \upsilon_j \overline{D}\boldsymbol{e}_j \tag{4.23}$$

donde: ω_i y υ_i son dos funciones normales que tienen el mismo significado que en el operador UNDX-m.

[26] Perpendicular

Cruce simplex (SPX) Fue propuesto por Tsutsui y Goldberg [396] y por Renders y Bersini [318] con diferentes implementaciones. A diferencia del UNDX-m, que genera los hijos alrededor del *cdm*, este lo hace sobre la superficie comprendida entre los padres

El método de Renders y Bersini sigue los siguientes pasos:

1. Seleccionar $m > 2$ padres y determinar cual es el mejor y el peor padre[27]: $\boldsymbol{s}_{p\max}$ y $\boldsymbol{s}_{p\min}$ respectivamente.
2. Calcular el centro de masas $\overline{\boldsymbol{s}}$ excluyendo al peor padre:

$$\overline{\boldsymbol{s}} = \frac{1}{m-1} \sum_{i=1, i \neq \min}^{m} \boldsymbol{s}_{p_i} \tag{4.24}$$

3. Calcular el *punto reflejado* $\boldsymbol{s}_r$ mediante la siguiente expresión:

$$\boldsymbol{s}_r = \overline{\boldsymbol{s}} + (\overline{\boldsymbol{s}} - \boldsymbol{s}_{p\min}) \tag{4.25}$$

4. Si la aptitud del punto reflejado $\boldsymbol{s}_r$ es mejor que la de $\boldsymbol{s}_{p\max}$ se genera el *punto expandido* $\boldsymbol{s}_e$ calculado mediante la siguiente expresión:

$$\boldsymbol{s}_e = \boldsymbol{s}_r + (\boldsymbol{s}_r - \overline{\boldsymbol{s}}) \tag{4.26}$$

5. Si $\boldsymbol{s}_e$ es mejor que $\boldsymbol{s}_r$ el hijo es $\boldsymbol{s}_e$. En caso contrario es $\boldsymbol{s}_r$.
6. Si la aptitud del punto reflejado $\boldsymbol{s}_r$ es peor que la de $\boldsymbol{s}_{p\max}$ y que $\boldsymbol{s}_{p\min}$ calcular el *punto contraído* $\boldsymbol{s}_c$ mediante la ecuación:

$$\boldsymbol{s}_c = \frac{\boldsymbol{s}_{\min} + \overline{\boldsymbol{s}}}{2} \tag{4.27}$$

7. Si $\boldsymbol{s}_c$ es mejor que $\boldsymbol{s}_{\min}$ el descendiente es $\boldsymbol{s}_c$. En caso contrario el descendiente se calcula mediante la ecuación:

$$\boldsymbol{s}_c = \frac{\boldsymbol{s}_{\min} + \boldsymbol{s}_{\max}}{2} \tag{4.28}$$

El método de Tsutsui y Goldberg realiza los siguientes pasos:

[27] La bondad se determina en base a la aptitud

1. Seleccionar m padres y determinar su centro de masas $\overline{\boldsymbol{s}}$.

2. Generar el vector números aleatorios $\boldsymbol{r} : \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_i, \ldots, \boldsymbol{r}_m\}$ según la expresión:

$$r_i = u^{1/i} \quad \forall i \in \{1, \ldots, m\} \tag{4.29}$$

   donde u es un número aleatorio $u \in \text{rnd}(0,1)$

3. Calcular el vector $\boldsymbol{Y}$ mediante la expresión:

$$Y_i = \overline{\boldsymbol{s}} + \epsilon(\boldsymbol{s}_{pi} - \overline{\boldsymbol{s}}) \quad \forall i \in \{1, \ldots, m\} \tag{4.30}$$

   donde ϵ es la tasa de expansión que controla el SPX

4. Calcular C_n mediante la siguiente fórmula recursiva:

$$C_i = \begin{cases} 0 & k = 1 \\ r_{i-1}(Y_{i-1} - Y_i + C_{i-1}) & \forall k \in \{2, \ldots, m\} \end{cases} \tag{4.31}$$

5. Generar el descendiente: $\boldsymbol{s}_h = Y_n + C_n$

A modo de resumen, la figura 4.27 muestra una comparativa del espacio de búsqueda abarcado por los diversos operadores de cruce que utilizan dos padres $\boldsymbol{s}_1, \boldsymbol{s}_1$ para engendrar a los hijos. En este ejemplo cada padre tiene dos genes y los hijos son representados por puntos o por zonas en función del tipo de operador. Los hijos generados por los diferentes operadores son los descritos a continuación:

- Puntos 1: hijos generados por los operadores de cruce determinísticos.
- Puntos 2 y 4': hijos generados por el operador de cruce promedio. Este operador es un tipo especial de operador aritmético donde $\lambda_1 = 0,5$. En este caso ambos hijos son iguales.
- Puntos 3,3': hijos generados por de cruce aritmético lineal. $0 \leq \lambda_1, \lambda_2 \leq 1$.
- Línea que une los padres $\boldsymbol{s}_{p1}$ y $\boldsymbol{s}_{p2}$: hijos generados por el operador de cruce heurístico.
- Zona amarilla: hijos generados por el operador de cruce plano y por el BLX-α para $\alpha = 0$.

- Línea que une los puntos 3,3': hijos generados por el operador de cruce aritmético lineal.
- Zonas azul y magenta: hijos generados por el operador de cruce SBX-β.
- Ventana exterior delimitada por los puntos 4-4": hijos generados por el operador de cruce BLX-α. La proporción entre la ventana exterior y la amarilla es igual α (zona de exploración).

Por lo tanto, de todos los operadores anteriores, el operador que abarca un mayor espacio de búsqueda es el BLX-α. Además con este operador se puede seleccionar cual es el espacio de exploración, frente al de explotación, que se desea abarcar siendo aconsejable su uso para problemas donde existan numerosos mínimos relativos aunque sin epístasis entre sus variables. El operador SBX-β también permite cierta exploración pero más centrada en las zonas próximas a los padres y puede ser conveniente para acelerar la convergencia del algoritmo.

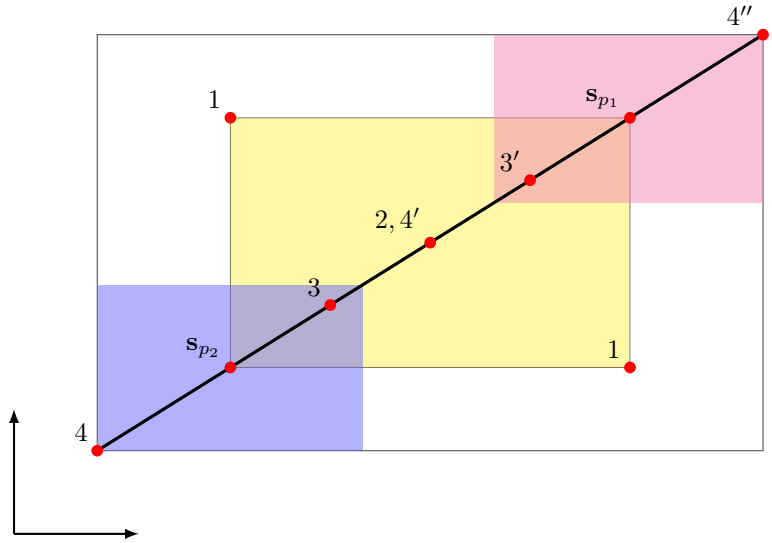
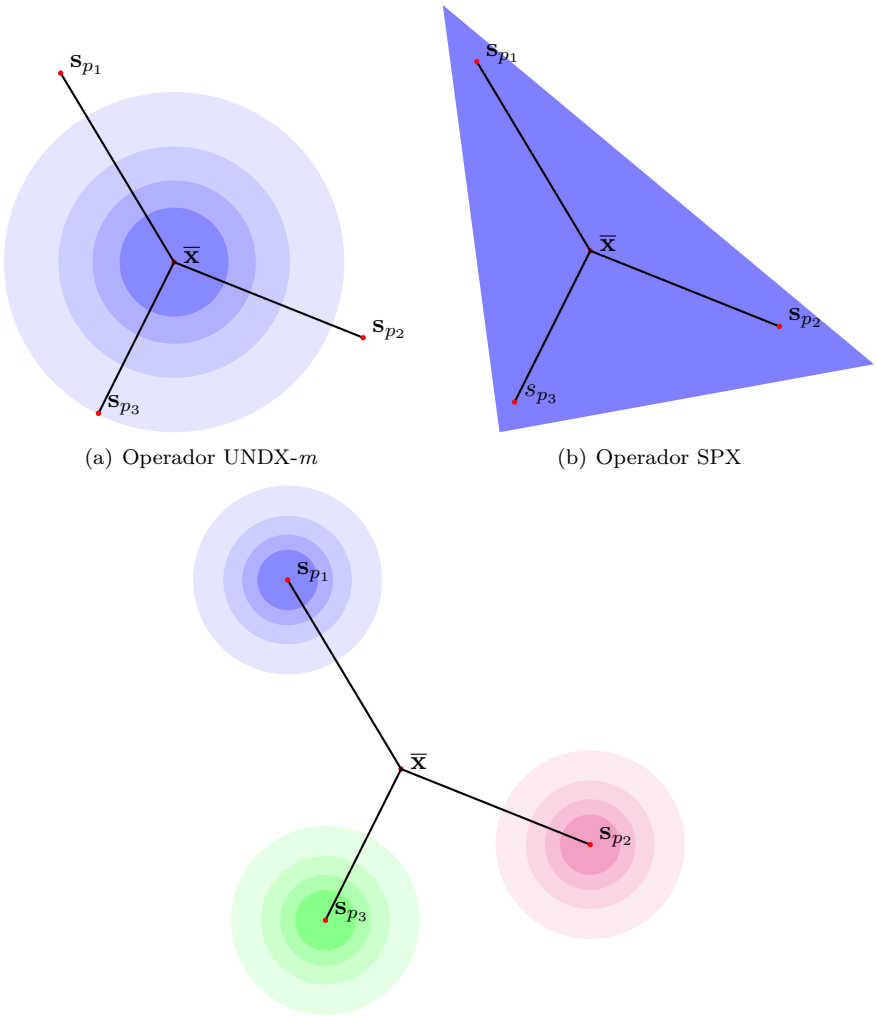


Figura 4.27: Comparativa entre los operadores de cruce basados en dos padres

La figura 4.28 muestra la comparativa entre los operadores de 3 o más padres, donde se puede apreciar la diferencia de enfoque de cada operador. No podemos afirmar que ninguno de ellos sea mejor o peor que los otros ya que su eficiencia dependerá del contexto en que se use. El UNDX-m será mejor en aquellos problemas donde exista una fuerte epístasis entre sus variables. El SPX será mejor en

aquellos problemas donde prioricemos la exploración, es decir se desee aumentar el espacio de búsqueda, y el PCX será mejor cuando prioricemos la explotación.



(a) Operador UNDX-m

(b) Operador SPX

(c) Operador PCX

Figura 4.28: Comparativa entre los tres operadores multiparentales

Además de los operadores expuestos, existen muchos otros aunque la mayoría sirven a usos específicos como la ordenación, búsqueda de listas o secuenciación y están basados en permutaciones de los genes como el PPX [45, 46] o el PMX [139, 162, 410], el operador de cruce por orden (OX) [82], el de cruce cíclico [282], el de cruce por arista [82], etc...

También existe un sinfín de variaciones de los operadores anteriores que no se reflejan por quedar fuera del alcance del presente trabajo. En cualquier caso y al igual que ocurre con el resto de operadores genéticos siguiendo el principio de *no hay café para todos*[28] no existe un operador que sea mejor a los demás en todos los casos, sino que estos operadores son dependientes del tipo de problema.

4.7.3 El operador de mutación

Este operador es el responsable de realizar búsquedas más allá de los puntos generados por los genomas de los padres. Es responsable por lo tanto de la exploración y, cuando el algoritmo converge y la población pierde diversidad, es casi el único mecanismo explorativo de este. En algunos EA como la *Programación Evolutiva* es el único operador de búsqueda, añadiendo otros operadores para que no se convierta en una búsqueda aleatoria.

Al igual que las mutaciones naturales los hijos mutados usualmente tienen una aptitud muy inferior a los padres, por lo que el porcentaje de individuos mutados no puede ser muy elevado. Pero también es cierto que muchas veces sirven para saltar de un óptimo local a otro e incluso al óptimo global. En el caso de los Algoritmos Genéticos no podemos confiar la exploración exclusivamente a este operador porque en este caso se comportaría como una búsqueda aleatoria, lo cual es del todo indeseable.

Técnicamente es un operador unario porque solo actúa sobre un solo individuo, a diferencia de los operadores reproductivos. La probabilidad de mutación, de cada gen de un individuo, recomendada suele ser $p_m \in [0.001, 0.01]$, aunque algunos autores como Schaffer et al. [334] sugieren emplear siguiente expresión empírica:

$$p_m \approx \frac{1,75}{\mu n_g} \tag{4.32}$$

o como Hesser y Männer [175] que propone:

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \frac{\exp(-\gamma t/2)}{\mu n_g} \tag{4.33}$$

donde α, β, γ son parámetros constantes.

[28] Traducción libre del autor del concepto *no free lunch theorems* enunciado por Wolpert y Macready [411]

La forma en que se seleccionan los genes que mutarán es la siguiente: A partir del primer gen se genera un número aleatorio $r \in [0,1]$ de modo que si $r \leq p_m$ se producirá la mutación de ese gen. El proceso se reproduce hasta alcanzar el último gen.

Si cada gen es mutado con una probabilidad p_m, la probabilidad de que un individuo sea mutado vendrá dada por la ecuación (4.34):

$$p_{mi} = 1 - (1 - p_m)^{n_g} \tag{4.34}$$

A continuación se describen los operadores más empleados habitualmente para codificaciones enteras o reales:

4.7.3.1 Mutación aleatoria uniforme

Fue descrita por Michalewicz [263]. Si el gen i resulta elegido para la mutación, se genera un número aleatorio dentro del dominio del gen, es decir, $r \in [l_i, u_i]$, donde l_i y u_i representan los valores mínimo y máximo que puede tomar dicho gen. El algoritmo 7 muestra en forma de pseudocódigo de este operador.

Algoritmo 7 Pseudocódigo del operador de mutación aleatoria uniforme

$\boldsymbol{\chi'} = \{a'_1, \ldots, a'_\mu\} : a'_k \in \mathbb{R} \quad \forall k \in \{1, \ldots, \mu\}$
inicializar: $\boldsymbol{\chi''}(0) = \{a''_1(0), \ldots, a''_\mu(0)\}$
for $k = 0$ to μ **do**
  for $i = 0$ to n_g **do**
    calcular: $r = \text{rnd}(0,1)$
    if $r \leq p_m$ **then**
      calcular: $r' \in [l_i, u_i]$
      $a''_{k_i} = r'$
    else
      $a''_{k_i} = a'_{k_i}$
    end if
  end for
end for
return $\boldsymbol{\chi''}$

4.7.3.2 Mutación aleatoria no uniforme

El operador de mutación aleatoria uniforme tiene el problema de que puede mutar genes muy alejados del gen inicial, dificultando la *explotación* del algoritmo. Este operador sacrifica la *exploración* para mejorar la *explotación*.

Luke [246] propone el siguiente algoritmo, aplicado a números enteros, para generar una proporción de individuos más dispersa que el anterior operador. En primer lugar se establece la proporción de individuos alejados del gen inicial mediante el valor $p_w \in [0, 1]$. A continuación, si el gen es elegido para sufrir una mutación se genera un número aleatorio $r' = \text{rnd}(0,1)$ de modo que si $r' < p_w$ se lanza una moneda al aire. Si sale cara se suma uno al valor inicial del gen y si sale cruz se le resta uno. Este proceso se repite hasta que $r' \geq p_w$.

El algoritmo 8 muestra en forma de pseudocódigo de este operador.

Michalewicz [263] propone una variación del método, aplicable a todos los reales, donde en lugar de sumarle un uno le suma un valor aleatorio entre el valor del gen y el rango máximo o mínimo del gen dependiendo de la tirada de la moneda.

El algoritmo 9 muestra en forma de pseudocódigo de este operador.

4.7.3.3 Mutación de convolución gaussiana

Es uno de los operadores más empleados en la actualidad. En este caso la mutación es realizada mediante la adición de ruido gaussiano al gen que se desea alterar. Este operador sitúa el gen modificado próximo al gen inicial en la mayoría de los casos pero es también posible, aunque con menor probabilidad, que genere puntos más alejados de este.

El ruido gaussiano se genera mediante una distribución normal $N(\hat{\mu}, \sigma^2)$ donde el valor de $\hat{\mu}$ suele ser igual a cero y la varianza σ^2 controla la cantidad de genes alejados a la media generados. La figura 4.29 muestra el efecto de la varianza en la distribución de los genes alterados. Dado que sería posible generar valores fuera de los límites del dominio del gen es necesario rechazar o truncar estos valores.

El algoritmo 10 muestra en forma de pseudocódigo de este operador.

Algoritmo 8 Pseudocódigo del operador de mutación aleatoria no uniforme para números enteros

$\boldsymbol{\chi'} = \{a'_1, \ldots, a'_\mu\} : a'_k \in \mathbb{R} \quad \forall k \in \{1, \ldots, \mu\}$
inicializar: $\boldsymbol{\chi''}(0) = \{a''_1(0), \ldots, a''_\mu(0)\}$
for $k = 0$ to μ **do**
  for $i = 0$ to n_g **do**
    calcular: $r = \text{rnd}(0,1)$
    $n = a'_{k_i}, n \in \mathbb{Z}$;
    if $r \leq p_m$ **then**
      repeat
        calcular: $r' = \text{rnd}(0,1)$
        calcular: $r'' = \text{rnd}(0,1)$
        if $r'' \leq 0.5$ **then**
          if $n + 1 \leq u_i$ **then**
            $n = n + 1$
          end if
        else
          if $n - 1 \geq l_i$ **then**
            $n = n - 1$
          end if
        end if
      until $r' \geq p_w$
    end if
    $a''_{k_i} = n$
  end for
end for
return $\boldsymbol{\chi''}$

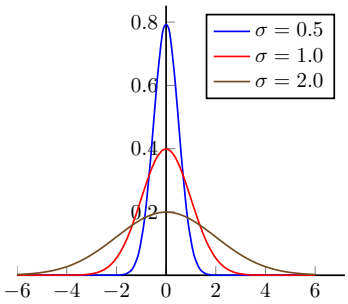


Figura 4.29: Distribución de genes alterados alrededor del gen original.

Algoritmo 9 Pseudocódigo del operador de mutación aleatoria no uniforme para números reales

$\boldsymbol{\chi'} = \{a'_1, \ldots, a'_\mu\} : a'_k \in \mathbb{R} \quad \forall k \in \{1, \ldots, \mu\}$
inicializar: $\boldsymbol{\chi''}(0) = \{a''_1(0), \ldots, a''_\mu(0)\}$
for $k = 0$ to μ **do**
  for $i = 0$ to n_g **do**
    calcular: $r = \text{rnd}(0,1)$
    if $r \leq p_m$ **then**
      calcular: $r' = \text{rnd}(0,1)$
      if $r'' \leq 0.5$ **then**
        calcular: $r'' = \{0, u_i - a'_{k_i}\}$
        $a''_{k_i} = a'_{k_i} + r''$
      else
        calcular: $r'' = \{0, a'_{k_i} - l_i\}$
        $a''_{k_i} = a'_{k_i} - r''$
      end if
    end if
    $a''_{k_i} = n$
  end for
end for
return $\boldsymbol{\chi''}$

Algoritmo 10 Pseudocódigo del operador de mutación de convolución gaussiana

$\boldsymbol{\chi'} = \{a'_1, \ldots, a'_\mu\} : a'_k \in \mathbb{R} \quad \forall k \in \{1, \ldots, \mu\}$
inicializar: $\boldsymbol{\chi''}(0) = \{a''_1(0), \ldots, a''_\mu(0)\}$
for $k = 0$ to μ **do**
  for $i = 0$ to n_g **do**
    calcular: $r = \text{rnd}(0,1)$
    if $r \leq p_m$ **then**
      repeat
        calcular: $r' = \text{rnd}(0,1)$
        obtener $z : p(z) = r', \quad z \sim N(\mu, \sigma)$
      until $l_i \leq z + a'_{k_i} \leq u_i$
      $a''_{k_i} = z + a'_{k_i}$
    else
      $a''_{k_i} = a'_{k_i}$
    end if
  end for
end for
return $\boldsymbol{\chi''}$

4.7.4 El operador de remplazo

Existen dos clases fundamentales de Algoritmos Genéticos dependiendo de la estrategia de reemplazo empleada: los Algoritmos Genéticos con reemplazo generacional (GGA), también denominados Algoritmos Genéticos Simples (SGA) y los Algoritmos Genéticos con brecha generacional (SSGA). En los primeros la población de padres es completamente reemplazada por la población de hijos una vez estos han sido engendrados y mutados[29]. En los segundos existe una proporción de padres, la brecha generacional [83], que sobrevive. Los GGA tienen nula brecha generacional, debiendo emplear operadores elitistas para evitar la pérdida de los mejores individuos, mientras que en los SSGA las brechas generacionales pueden llegar a ser bastante amplias.

Las diferencias estrategias adoptadas para los SSGA dan lugar a los diferentes operadores de reemplazo:

4.7.4.1 Reemplazo de los menos aptos

Fue definido por De Jong y Sarma [84] dentro de un modelo de Algoritmo Genético incremental denominado GENITOR. Con este operador los hijos engendrados reemplazan a los individuos menos aptos de la población.

4.7.4.2 Reemplazo aleatorio

Definido por Syswerda [82]. Con este operador los hijos engendrados reemplazan de forma aleatoria a los individuos de la población.

[29] Asumiendo que no hay elitismo

4.7.4.3 Torneo a muerte

Definido por Smith [357, 360]. Con este operador se selecciona aleatoriamente a un conjunto de individuos de la población, donde el menos apto es sustituido por un hijo.

4.7.4.4 Reemplazo del individuo más viejo

Con este operador se reemplaza al individuo más viejo[30]. Se corre el riesgo de eliminar al mas apto si no se incorpora ningún operador de elitismo.

4.7.4.5 Selección conservativa

Definido por Smith combina la estrategia de reemplazo del más viejo con un torneo de selección binaria. Este operador realiza un torneo entre el individuo más viejo y otro seleccionado aleatoriamente dentro de la población. Si el más viejo es más apto sobrevive y el otro es sustituido por un hijo. En caso contrario es el viejo el reemplazado. Esta estrategia tiene la ventaja de no reemplazar nunca al individuo más apto si es también el más viejo.

4.7.4.6 Reemplazo de los progenitores

Este operador mediante una estrategia de supervivencia predefinida decide si el hijo reemplaza a alguno de los padres o bien ambos progenitores sobreviven. La estrategia puede estar basada en la aptitud, distancia entre genomas, etc...

4.7.4.7 Elitismo

El operador elitista tiene por objeto evitar la eliminación del individuo o grupo de individuos más aptos de la población. La eliminación del individuo más apto puede provocar una pérdida significativa de explotación dificultando la convergencia del algoritmo. Casi todos los operadores de reemplazo actuales y algunos de cruce incorporan estrategias elitistas.

[30] El que ha sobrevivido durante mas generaciones

4.7.5 Ajuste de los parámetros de los operadores genéticos

Todos los operadores genéticos están dotados de parámetros que deben ajustarse y de cuyo valor depende en gran medida la eficiencia y eficacia del Algoritmo Genético implementado. Existe dos alternativas para determinar estos parámetros: el ajuste *offline* y *online* o adaptativo.

4.7.5.1 Ajuste offline

Tiene un enfoque determinístico y el ajuste de los parámetros se basa en la experiencia previa en problemas similares. Tiene la ventaja de ser muy sencillo de implementar pero el inconveniente de que los valores no tienen porque ser los buenos para nuestro problema.

Ya desde los trabajos iniciales de Holland, la determinación de los parámetros adecuados en los Algoritmos Genéticos constituyó un primer problema. El mismo De Jong [83] desarrolló un conjunto de funciones, utilizadas aún hoy en día, como medida de la eficiencia de los Algoritmos Genéticos con el fin de determinar los valores de ajuste óptimos.

Al trabajo de De Jong le siguieron el de Greffenstette [141], Goldberg [135], Woon et al. [412] entre otros muchos. Finalmente los trabajos de Hart y Belew [163], y de Wolpert y Macready [411] establecieron el teorema de *no hay café para todos*. Este teorema determina que en promedio, para todos los problemas todos los algoritmos de búsqueda, incluida la búsqueda aleatoria, funcionan igual. Esto quiere decir que si los Algoritmos Genéticos funcionan bien con cierto tipo de problemas no lo harán con otros, mientras que otras técnicas que no funcionen bien donde dominan los Algoritmos Genéticos puede que si lo hagan en otros campos. La figura 4.30 muestra de forma gráfica este teorema.

De igual modo unos parámetros de ajuste para un problema o conjunto de problemas no funcionarán bien para otros. Ya el propio Goldberg en su primer trabajo explicaba de forma gráfica este concepto, aunque de manera errónea, porque situaba los Algoritmos Genéticos con un desempeño superior a la búsqueda aleatoria.

La tabla 4.3 muestra diferentes valores de ajuste propuestos por autores para problemas relacionados con diversos campos. Tradicionalmente los parámetros se ajustaban manualmente del siguiente modo: se fijaban todos los parámetros menos

Tabla 4.3: Valores de ajuste para los Algoritmos Genéticos propuestos por varios autores

Autor/es	Población	Probabilidad cruce	Probabilidad de mutación	Número máximo de generaciones
De Jong [83] (1975)	No menciona	0,60-0,80	0,010-0,020	No menciona
Greffenstette [141] (1986)	80	0,45	0,010	20
Wang [407] (1991)	100	1,00	0,010	50
Janson y Frenzel [190] (1993)	100	No menciona	0,001	500
Ball et al. [19] (1993)	100	0,30	0,006	180
Koumousis y Georgiou [214] (1994)	20	0,60	0,010	100
Srinivas y Patnaik [372] (1994)	30-200	0,50-1,00	0,001-0,05	No menciona
Croce et al. [76] (1995)	300	1,00	0,030	3000
Dorndorf y Pesch [98] (1995)	200	0,65	0,001	300
Tate y Smith [385] (1995)	100	0,25	0,750	2000
Keane [203] (1995)	100	0,80	0,005	10
Suresh et al. [379] (1996)	40-60	0,50-0,70	0,100	No menciona
Carroll [58] (1996)	100-200	0,60	0,010-0,005	20
Carroll [58] (1996)	50	0,50	0,020-0,040	12
Roston y R.H. [324] (1996)	200	0,50	0,010	150
Furuta et al. [122] (1996)	No menciona	0,25	0,010	No menciona
Liu et al. [244] (1997)	50	0,80	0,010	100
Roman et al. [323] (2000)	25-125	0,60-0,90	0,005-0,05	100
Nanakorn y Meesomklin [275] (2001)	40	0,8	0,001	100
Nanakorn y Meesomklin [275] (2001)	50	0,85	0,05	100
Annicchiarico y Cerrolaza [12] (2002)	100	1,00	0,020	50
Victoria y Martí [402] (2005)	20	0,8	0,003	300

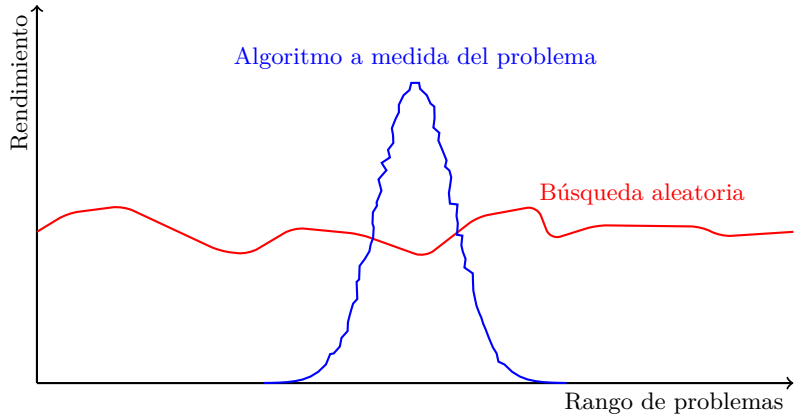


Figura 4.30: Comparativa entre algoritmos de búsqueda

uno, el cual se modificaba y generaba la curva de eficiencia de ese parámetro. Después se continuaba con el siguiente y así sucesivamente. Este enfoque parte de un error y es que no tiene en cuenta las interrelaciones entre parámetros, que en la mayoría de los casos es muy importante. La realización de experimentos variando diferentes parámetros a la vez resulta harto compleja y requeriría de un diseño de experimentos por lo que usualmente no es un procedimiento empleado. Debe tenerse en cuenta que, como se ha comentado anteriormente, incluso en el caso de que se encuentre un ajuste óptimo es muy probable que este sea únicamente útil para el problema planteado.

Por otra parte, el uso de parámetros estáticos conceptualmente está en contra del mismo proceso evolutivo. Es intuitivamente obvio que los valores de los parámetros deberían variar durante el proceso de la evolución como demuestran los trabajos de Davis [81], Syswerda [383], Bäck [27–29] , Hesser y Männer [175], Schaffer y Eshelman [333], Jain y Fogel [188], Schwefel [345], Smith y Fogarty [358, 359], Stephens et al. [374] entre otros. Por ejemplo una tasa alta de mutaciones es adecuada en las etapas iniciales del algoritmo para favorecer la exploración y por el contrario debería ser más baja posteriormente para favorecer la explotación. También es deseable tener una tasa alta cuando la diversidad de la población disminuye, cosa que por otro lado ocurre cuando el algoritmo converge y se incrementa la presión selectiva.

Una primera aproximación a la adaptación muy rudimentaria fue la transformación de los parámetros determinísticos de estáticos en dinámicos, es decir, haciéndolos de alguna forma dependientes del número de generaciones transcurridas. Sin embargo este enfoque plantea un problema aun mayor. Al requerir de un ajuste manual, la determinación de una regla determinística realmente útil que rija la modificación del parámetro en función del número de generaciones es mucho más compleja. Esta regla determina como se comportará el operador sin tener la menor noción de lo que está haciendo el algoritmo, pudiendo actuar de forma improcedente.

4.7.5.2 Ajuste online

El ajuste online se basa en un concepto totalmente diferente. Es el propio algoritmo el cual, por diferentes mecanismos, se encarga de ir ajustando los parámetros del Algoritmo Genético. Esta técnica fue empleada inicialmente en las ES.

Los mecanismos de control actuales pueden ser fundamentalmente de dos tipos: mediante un mecanismo de retroalimentación heurístico basado en ciertos parámetros que se calculan cada generación como la diversidad, variación de la aptitud de la población, etc...; o bien incorporando los propios parámetros en la propia codificación, dejando que sea el propio proceso evolutivo el que se encargue de realizar el ajuste. A este último mecanismo se le denomina también autoadaptación.

Retroalimentación heurística

Esta parte de la idea de utilizar un parámetro dinámico al cual se le añade un mecanismo de retroalimentación. Uno de los primeros principios de adaptación empleados fue el de Rechenberg [314] donde modificó online la tasa de mutación y Davis [81] que modificó la probabilidad y punto de cruce del operador de cruce.

De entre los diferentes medidas de la evolución de un Algoritmo Genético destacan [426]:

1. La convergencia. En ocasiones también denominada *calidad* del algoritmo. La convergencia cuantifica en que medida se ha alcanzado un valor óptimo, que no tiene por que ser el global, en la medida en que la aptitud de la

población no mejora. Existen diferentes procedimientos para determinar la convergencia:

(a) La aptitud del mejor individuo. Se determina en función de la variación de la aptitud del mejor individuo durante G generaciones[31]:

$$C(t) = \frac{\Phi_{\max}(t)}{\Phi_{\max}(t-G)} \tag{4.35}$$

Este criterio garantiza la obtención de al menos un óptimo, aunque no asegura que sea el absoluto.

(b) La aptitud del peor individuo. Se determina en función de la variación de la aptitud del peor individuo durante G generaciones:

$$C(t) = \frac{\Phi_{\min}(t)}{\Phi_{\min}(t-G)} \tag{4.36}$$

(c) La suma de aptitudes de la población. Se determina en función de la suma de la aptitudes de la población durante G generaciones:

$$C(t) = \frac{\sum_{i=1}^{\mu} \Phi_i(t)}{\Phi_i(t-G)} \tag{4.37}$$

En determinados casos este procedimiento puede resultar contraproducente debido a la incursión constante de individuos no aptos.

(d) La aptitud media. Se determina en función de la aptitud media de la población durante G generaciones:

$$C(t) = \frac{\overline{\Phi}(t)}{\overline{\Phi}(t-G)} \tag{4.38}$$

2. La diversidad:
La diversidad cuantifica en que medida son diferentes los individuos de una población. Existen diferentes procedimientos para determinar la diversidad de una población:

(a) Medida de la diversidad genotípica. Basados en la distancia genotípica entre individuos. Emplean normalmente la distancia de Hamming[161].

[31] Se asume proceso de minimización. Para maximizar se utiliza la relación inversa.

La distancia de Hamming D entre dos individuos $s^1 = \{a_1^1, a_2^1, \ldots, a_i^1, \ldots, a_\mu^1\}$ y $s^2 = \{a_1^2, a_2^2, \ldots, a_i^2, \ldots, a_\mu^2\}$ viene dada por la fórmula:

$$D^{1-2} = \sum_{k=1}^{n_g} |a_k^1 - a_k^2| \tag{4.39}$$

Para calcular la distancia es conveniente normalizar la distancia de Hamming para el gen k. La distancia de Hamming máxima entre dos genes vendrá dada por los valores máximo u_k y mínimo l_k de ese gen. Cuando ambos genes se encuentren a esa distancia la distancia de Hamming normalizada será 1. Por otro lado, si el vector que representan los genomas es de longitud n_g, la distancia de Hamming de los dos individuos deberá ser también dividida por n_g. La ecuación que determina la distancia de Hamming normalizada será por lo tanto:

$$D^{1-2} = \frac{1}{n_g} \sum_{k=1}^{n_g} \frac{|a_k^1 - a_k^2|}{u_k - l_k} \tag{4.40}$$

La diversidad de población vendrá dado por la suma de las distancias de Hamming entre todos los individuos de la población dividido por el número de combinaciones posibles: $n_g(n_g - 1)/2$. De este modo la diversidad de la población viene determinada por la ecuación:

$$D(t) = \frac{2}{n_g(n_g - 1)} \sum_{i=1}^{\mu} \sum_{j=i+1}^{\mu} \sum_{k=1}^{n_g} \frac{|a_k^i - a_k^j|}{u_k - l_k} \tag{4.41}$$

(b) Medida de la diversidad fenotípica. La presión selectiva ϕ es uno de los procedimientos típicos de medida de la diversidad[31]:

$$D(t) = \phi(t)^{-1} = \frac{\Phi_{\max}(t)}{\overline{\Phi}(t)} \tag{4.42}$$

Otro método también empleado utiliza la desviación estándar de la aptitud:

$$D(t) = \sqrt{\frac{\sum_{i=1}^{\mu} (\overline{\Phi}(t) - \Phi_i(t))^2}{\mu - 1}} \tag{4.43}$$

Control determinístico A partir de las diferentes medidas descritas anteriormente, diversos autores [10, 17, 371] plantean diferentes reglas para la tasa de mutación, el punto o tipo de cruce así como su tasa, etc. . . en función de una serie de parámetros determinísticos.

Otros tipos de control A partir de las diferentes medidas descritas anteriormente, otros autores plantean un control de los parámetros basado en redes neuronales o lógica difusa [171, 184, 363, 403].

Auto-adaptación

Existen fundamentalmente dos variantes de auto-adaptación:

Coevolución En esta se incluyen los parámetros que regulan los operadores del algoritmo dentro de la propia codificación del mismo. De este modo se pueden incluir no solo las probabilidades de selección, cruce o mutación, sino también los propios operadores cambiando de uno a otro modelo de forma automática. El inconveniente de este método es que en ocasiones resulta excesivamente lento y puede ocurrir que se produzca una convergencia prematura antes de que se haya adaptado. Diversos autores como Tuson y Ross [397], y Bäck [32] han empleado con éxito esta técnica.

micro-Algoritmo Genético (μGA) Esta variante utiliza otro algoritmo genético, mas tosco, que se encarga de optimizar y reajustar los parámetros de los operadores del primero. Diversos autores [30, 227, 261, 348] han empleado con éxito esta técnica en diferentes campos.

4.8 Criterios de detención

Una vez se ha detectado la convergencia, el algoritmo debe detenerse para evitar que el bucle se perpetúe indefinidamente. Entre los criterios de detención más habituales están:

1. Número máximo de generaciones o tiempo transcurrido máximo. El algoritmo se detiene cuando ha alcanzado un número de generaciones T predeterminado o cuando ha transcurrido un determinado tiempo. Se utiliza

normalmente para evitar que el algoritmo continúe realizando el bucle si no es detenido por ninguno de los criterios posteriores.

2. Estabilidad de la aptitud durante t generaciones. El algoritmo se detiene si la aptitud de uno o varios individuos de la población se mantiene constante tras un número determinado de generaciones t.

3. Estancamiento durante t generaciones. El algoritmo se detiene si la aptitud de uno o varios individuos de la población no mejora tras un número determinado de generaciones t.

4.9 La función de aptitud o función objetivo

La función de aptitud constituye el núcleo central de los Algoritmos Genéticos. Esta es una representación del problema a resolver donde se cuantifica la bondad (aptitud) de cada solución (individuo). Juega el papel de *medio* calificando a los individuos en función de adaptación al mismo.

Aunque inicialmente los Algoritmos Genéticos se emplearon para la optimización de funciones no restringidas, lo más habitual es que no sea así. En problemas de optimización estructural lo habitual es que esta se halle sujeta a diferentes restricciones de resistencia y rigidez. El tratamiento de estas restricciones es muy variado y en muchos casos también, dependiente del problema.

Matemáticamente, el proceso de optimización se puede expresar como:

Encontrar el vector de soluciones $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ que optimiza[32] la función $f(\boldsymbol{x})$ sujeto a:

$$\begin{aligned} \tilde{g}_i(\boldsymbol{x}) \leq 0, \forall i \in \{1, \ldots, m\} \\ \tilde{h}_j(\boldsymbol{x}) = 0, \forall j \in \{1, \ldots, n\} \end{aligned} \tag{4.44}$$

Cuando una restricción satisface la condición $\tilde{g}_i = 0$ actúa limitando el espacio de soluciones factibles. En este caso se dice que dicha restricción está *activa*. Por el contrario, si no tiene ninguna influencia en la solución se clasifica como *inactiva*.

[32] Maximiza, minimiza

Resulta una práctica bastante común el transformar las igualdades en inecuaciones mediante la expresión:

$$|\tilde{h}_j(\boldsymbol{x})| - c_j \leq 0 \tag{4.45}$$

donde c_j es la tolerancia admitida para la igualdad. Generalmente un valor muy pequeño.

También resulta habitual normalizar las restricciones, de modo que puedan ser comparadas entre sí[33]. Esta se realiza dividiendo la restricción por el valor que activa la restricción y restándole la unidad. De este modo las restricciones normalizadas pueden escribirse como:

$$g_j(\boldsymbol{x}) = \frac{\tilde{g}_j(\boldsymbol{x})}{b_j} - 1 \leq 0 \tag{4.46}$$

$$h_j(\boldsymbol{x}) = \frac{|\tilde{h}_j(\boldsymbol{x})|}{c_j} - 1 \leq 0 \tag{4.47}$$

4.9.1 Manejo de las restricciones

En la bibliografía existe un sinfín de métodos para manejar las restricciones del problema. De entre estas destacan las descritas en los siguientes subapartados:

4.9.1.1 Funciones de penalización

Las funciones de penalización constituyen uno de los métodos de manejo de las restricciones más empleados en los EA. Fueron propuestas inicialmente por Courant [73] y posteriormente ampliadas por Carroll [57] y por Fiacco y McCormick [114]. La idea fundamental del método consiste en transformar un problema restringido en uno sin restricciones añadiéndole (o restándole) un determinado valor a la función objetivo dependiendo del grado de violación de las restricciones presentes en una determinada solución.

En los métodos de optimización clásicos existen dos tipos diferentes de funciones de penalización: las exteriores y las interiores. En las primeras se parte soluciones localizadas en el espacio de soluciones no factibles y desde aquí el algoritmo va desplazando las soluciones hacia el espacio de soluciones factibles. Por el contrario

[33] La normalización es imprescindible en las funciones de penalización

en las segundas se parte de soluciones factibles eligiendo un factor de penalización que es muy pequeño y que crece hasta el infinito en la frontera entre el espacio de solucione factibles y no factibles. De este modo las restricciones actúan como barreras del proceso de optimización.

El método más empleado en los EA, y especialmente en la optimización de estructuras, es la penalización exterior ya que en muchos casos es muy difícil partir de una solución factible, así como generar nuevos individuos factibles a partir de las poblaciones iniciales.

De este modo, la formulación de la función de aptitud modificada por una función de penalización exterior es:

$$\Phi(\boldsymbol{x}) = f(\boldsymbol{x}) \pm P(\boldsymbol{x}) \tag{4.48}$$

$$P(\boldsymbol{x}) = \left[\sum_{i=1}^{m} r_i G_i + \sum_{j=1}^{m} r_j L_j \right] \tag{4.49}$$

donde $f(\boldsymbol{x})$ es la función de aptitud, el signo positivo se emplea en la búsqueda de mínimos y el negativo en la de máximos, $P(\boldsymbol{x})$ es la función de penalización $\Phi(\boldsymbol{x})$ es la función de aptitud modificada, G_i y L_j son funciones de las restricciones $g(\boldsymbol{x})$ y $h(\boldsymbol{x})$ respectivamente, y r_i y r_j son constantes positivas denominadas factores de penalización.

La elección del grado de penalización no es trivial, ya que una penalización demasiado baja o demasiado alta puede hacer muy difícil para el Algoritmo Genético la obtención de un óptimo absoluto [80, 224, 225].

Según el principio de la *regla de la penalización mínima* [79, 224, 356] la penalización debe mantenerse lo más baja posible, justo por encima del límite por debajo del cual las soluciones no factibles son óptimas. Esto es debido al hecho de que si la penalización es demasiado alta el Algoritmo Genético será empujado rápidamente hacia el espacio de soluciones factibles, evitando la exploración del espacio de soluciones no factibles. Si por ejemplo existen varias zonas de soluciones factibles aisladas entre sí, el algoritmo se verá empujado hacia una de ellas siendo prácticamente imposible el salto y la exploración de las otras zonas. Por otra parte, esta regla no es fácil de implementar porque la frontera entre el espacio de soluciones factibles y no factibles es usualmente desconocido.

Si por el contrario la penalización es demasiado baja, se perderá demasiado tiempo explorando el espacio de soluciones factibles ya que la penalización será irrelevante para la función objetivo [35]. Este aspecto es importante en los Algoritmos Genéticos ya que muchos problemas tienen sus óptimos localizados en el límite del espacio de soluciones factibles.

La figura 4.31 muestra un ejemplo de aplicación de la regla de la penalización mínima. Una penalización demasiado baja dirigirá la búsqueda hacia el origen, mientras que una penalización demasiado alta lo hará hacia el punto d.

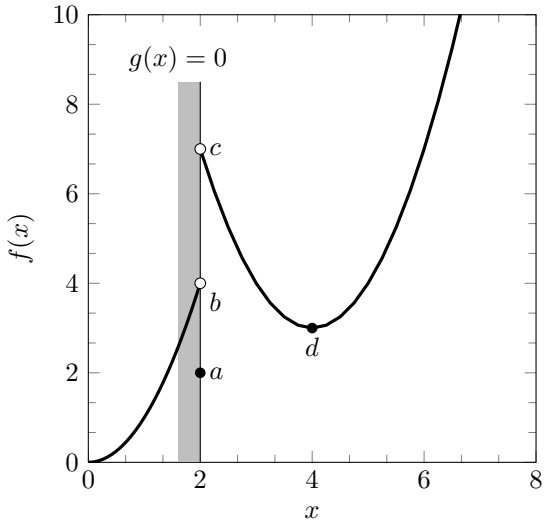


Figura 4.31: Ejemplo de optimización restringida

Por lo tanto la elección de los parámetros de penalización no es sencilla, y en todo caso tampoco podrá ser generalizable al igual que no lo eran los parámetros de los operadores genéticos.

En base a los razonamientos anteriores se han desarrollado los siguientes enfoques en el tratamiento de la función de penalización

Penalización estática

En este caso los factores de penalización son determinísticos, es decir se establecen a priori y permanecen constantes durante todo el proceso de optimización.

Existen diferentes planteamientos en la función de penalización. De entre ellos destaca el de Homaifar et al. [180], donde establece varios niveles de violación de las restricciones, de modo que el grado de penalización se incrementa con el grado de violación. Según este criterio, la función de penalización es:

$$P(\boldsymbol{x}) = \sum_{i=1}^{m} R_{k,i} H(g_i(\boldsymbol{x})) g_i(\boldsymbol{x})^2 \ , \ k = \{1, 2, \ldots, l\} \tag{4.50}$$

donde $H(\bullet)$ es la función de Heaviside:

$$H(x) = \begin{cases} 0, \ x \leq 0 \ , \text{ la solución es factible} \\ 1, \ x > 0 \ , \text{ en caso contrario} \end{cases} \tag{4.51}$$

siendo $R_{k,i}$ son los coeficientes de penalización, l es el número de niveles de violación establecidos y m el número total de restricciones, transformando las igualdades en inecuaciones. Este método tiene la ventaja de tratar a las restricciones de forma diferente, permitiendo una mayor penalización en unas que en otras[34]. Por contra, tiene el inconveniente de que el número de parámetros a determinar es extraordinariamente grande: $m(2l + 1)$.

Otro planteamiento curioso es el de Morales y Quezada [273], donde la función de aptitud modificada se plantea según la expresión:

$$\Phi(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) \quad \text{si la solución es factible} \\ K - \sum_{i=1}^{s} \dfrac{K}{m} \quad \text{en caso contrario} \end{cases} \tag{4.52}$$

donde K es un número grande, usualmente $K = 10^9$. Nótese que cuando un individuo no es factible, no es necesario calcular la aptitud $f()$, lo cual la hace conveniente en aquellos problemas donde el coste computacional de la función de aptitud es elevado y es posible evaluar las restricciones de forma separada. Por otra parte todos los individuos que violan el mismo número de restricciones tienen

[34] Este es un tipo de aproximación arcaico a la optimización multiobjetivo

la misma aptitud independientemente de su proximidad al espacio de soluciones factibles, lo cual contradice las reglas básicas de Richardson et al. [319] sobre la definición de funciones de penalización.

Hoffmeister y Sprave [177] plantean la siguiente función de penalización:

$$P(\boldsymbol{x}) = \sqrt{\sum_{i=1}^{m} H(g_i(\boldsymbol{x}))g_i(\boldsymbol{x})^2} \quad , \quad H : \mathbb{R} \rightarrow \{0, 1\} \tag{4.53}$$

donde $H(\bullet)$ es la función de Heaviside vista en la la ecuación (4.51).

El inconveniente de la aproximación de Hoffmeister es que asume que las soluciones no factibles serán siempre evaluadas peor que las soluciones factibles lo cual no siempre ocurre [264].

Otros investigadores [34, 56, 140, 183, 283, 319, 386, 423] han empleado funciones de penalización basadas en la distancia al espacio de soluciones factibles, pero en todos los casos hacen falta parámetros de penalización que impiden su generalización al ser dependientes del problema.

Para concluir, Prendes Gero [306], Prendes Gero et al. [307] plantea una función de penalización a tramos donde los individuos factibles son fuertemente penalizados y los no factibles son penalizados mediante una función exponencial:

$$r_i = \begin{cases} 0 & g_i(\boldsymbol{x}) = 0 \\ e^{2-g_i(\boldsymbol{x})} & 0 < g_i(\boldsymbol{x}) < 1 \\ 1 & g_i(\boldsymbol{x}) = 1 \\ 1000 g_i(\boldsymbol{x}) & g_i(\boldsymbol{x}) > 1 \end{cases} \tag{4.54}$$

donde r_i es el coeficiente de penalización de la ecuación (4.49).

Este enfoque presenta el inconveniente de la eliminación rápida de las soluciones factibles, provocando un desplazamiento de la población hacia la zona de soluciones no factibles, dificultando la localización de un óptimo factible.

Penalización dinámica

En este caso los factores de penalización son dependientes del número de generaciones transcurridas, de forma que dichos factores se incrementan con el tiempo.

Joines y Houck [195] proponen la siguiente función de penalización para la generación t:

$$P(\boldsymbol{x}, t) = (Ct)^{\alpha} \left[\sum_{i=1}^{m} D_i^{\beta}(\boldsymbol{x}) + \sum_{j=1}^{m} D_j(\boldsymbol{x}) \right] \tag{4.55}$$

donde C, α, β son constantes y D_i, D_j se calculan según las ecuaciones (4.56) y (4.57). Joines empleó $C = 0,5, \alpha \in [1,2], \beta \in [1,2]$.

$$D_i(\boldsymbol{x}) = \begin{cases} 0, & g_i(\boldsymbol{x}) \leq 0 \\ |g_i(\boldsymbol{x})|, & g_i(\boldsymbol{x}) > 0 \end{cases} \quad 1 \leq i \leq m \tag{4.56}$$

$$D_j(\boldsymbol{x}) = \begin{cases} 0, & -\epsilon \leq h_j(\boldsymbol{x}) \leq \epsilon \\ |h_j(\boldsymbol{x})|, & \text{en caso contrario} \end{cases} \quad 1 \leq j \leq n \tag{4.57}$$

Este enfoque presenta los mismos inconvenientes que la penalización estática, es decir, la determinación de los parámetros adecuados. Algunos autores como Michalewicz [265], Michalewicz y Nazhiyath [266] señalan que el método produce convergencia prematura en casos distintos al indicado por los autores. Otros como Dasgupta y Michalewicz [79] señalan que la técnica converge hacia zonas del espacio de soluciones factibles o no factibles, alejadas del óptimo global.

Kazarlis y Petridis [202] propusieron el siguiente enfoque alternativo:

$$P(\boldsymbol{x}, t) = V(t) \left[A \sum_{i=1}^{m} H\left(g_i(\boldsymbol{x})\right) w_i \phi(\Delta g_i(\boldsymbol{x})) + B \right] \left[1 - \prod_{i=1}^{m} H(-g_i(\boldsymbol{x}) \right] \tag{4.58}$$

donde A es un factor de *severidad* de la penalización, B es un umbral mínimo de penalización, $H(\bullet)$ es la ecuación de Heaviside descrita en la ecuación (4.51), w_i es un factor de ponderación para la restricción i, $\Delta g_i(\boldsymbol{x})$ es el grado de violación de la restricción i, $\phi(\cdot)$ es una función de penalización que depende del grado de violación de i. $V(t)$ es un parámetro que depende del número de generaciones transcurridas. Los autores recomiendan utilizar:

$$V(t) = \left(\frac{t}{T} \right)^2 \tag{4.59}$$

Esta técnica ha funcionado bien en varios de los problemas expuestos por los autores pero todavía no hay trabajos concluyentes sobre su uso en otros campos. Al igual que las anteriores depende de una serie de parámetros que son dependientes del problema.

Penalización adaptativa

Al igual que ocurría con los parámetros de los operadores genéticos, la función de penalización puede adaptarse a la evolución del algoritmo a partir de diferentes medidas tomadas de este. La primera aproximación a este tipo de penalización la realizaron Bean y Ben Hadj-Alouane [36, 39]. En ella la función de penalización se retroalimenta según la fórmula:

$$P(\boldsymbol{x}, t) = \lambda(t) \left[\sum_{i=1}^{m} g_i(\boldsymbol{x})^2 + \sum_{j=1}^{n} |h_j(\boldsymbol{x})| \right] \tag{4.60}$$

donde $\lambda(t)$ se actualiza cada generación del siguiente modo:

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\lambda(t) & \text{caso 1} \\ \beta_2 \lambda(t) & \text{caso 2} \\ \lambda(t) & \text{casos restantes} \end{cases} \tag{4.61}$$

donde, el caso 1 se produce cuando el mejor individuo es factible durante k generaciones, el caso 2 cuando no es nunca factible durante k generaciones, β_1, β_2 son constantes que deben cumplir las condiciones: $\beta_2 > 1, \beta_1 > \beta_2, \beta_1 \neq \beta_2$. De este modo la penalización disminuye si los mejores individuos de las últimas k generaciones han sido factibles y se incrementa cuando no lo han sido.

Al igual que en los casos anteriores el inconveniente del método es la elección de los parámetros adecuados y dependencia de estos con el tipo de problema.

Gen y Cheng [129] realizaron una variante del método planteado por Yokota et al. [424] donde se emplea un factor de penalización multiplicativo, en lugar de aditivo como el resto de los métodos. En este caso la función de adaptación modificada es de la forma:

$$\Phi(\boldsymbol{x}, t) = f(\boldsymbol{x}) P(\boldsymbol{x}, t) \tag{4.62}$$

En una primera aproximación la función de penalización presentaba la forma:

$$P(\boldsymbol{x},t) = 1 \pm \frac{1}{m}\sum_{i=1}^{m}\left(\Delta g_i(\boldsymbol{x},t)\right)^k \tag{4.63}$$

Mas tarde presentaron la siguiente variante de la función de penalización :

$$P(\boldsymbol{x},t) = 1 \pm \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\Delta g_i(\boldsymbol{x},t)}{\Delta g_i^{\max}}\right)^k \tag{4.64}$$

$$\Delta g_i^{\max} = \max[\Delta g_i(\boldsymbol{x}_j,t)] \quad , \quad j = \{1,2,\ldots,\mu\} \tag{4.65}$$

donde $\Delta g_i(\boldsymbol{x})$ es el grado de violación de la restricción i, $\Delta g_i^{\max}$ es el grado de violación máxima de la restricción i en toda la población durante la generación t y k suele tomar valores entre 1 y 2.

La ventaja de esta técnica es que consigue preservar la diversidad de la población, evitando penalizar demasiado las soluciones no factibles. Según los autores el método es independiente del problema, pero no se encuentra suficientemente probado.

Barbosa y Lemonge [22], Lemonge y Barbosa [229] proponen otro método libre de parámetros en el cual emplean la siguiente función de aptitud modificada:

$$\Phi(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{si la solución es factible} \\ \tilde{f}(\boldsymbol{x}) + \displaystyle\sum_{i=1}^{m} k_i \Delta g_i(\boldsymbol{x}) & \text{si la solución no es factible} \end{cases} \tag{4.66}$$

donde

$$\tilde{f}(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & f(\boldsymbol{x}) > \overline{f}(\boldsymbol{x}) \\ \overline{f}(\boldsymbol{x}) & f(\boldsymbol{x}) \leq \overline{f}(\boldsymbol{x}) \end{cases} \tag{4.67}$$

$$k_j = \frac{\displaystyle\sum_{i=1}^{\mu} |f(\boldsymbol{x}_i)|}{\displaystyle\sum_{l=1}^{\mathrm{m}} \left[\sum_{i=1}^{\mu} \Delta g_l(\boldsymbol{x}_i)\right]^2} \sum_{i=1}^{\mu} \Delta g_j(\boldsymbol{x}_i) \tag{4.68}$$

siendo $\overline{f}(\boldsymbol{x})$ la media de los valores de la función de aptitud sin modificar de la población, k_j el factor de penalización de la restricción j, y $\Delta g_j(\boldsymbol{x}_i)$ el grado de violación de la restricción j por el individuo i.

El objetivo de este método es que aquellas restricciones más difíciles de satisfacer tengan un coeficiente de penalización relativamente mayor. Este método ha sido aplicado con éxito a la optimización de estructuras, pero no ha sido suficientemente analizado en otros campos.

Penalización coevolutiva

Este método plantea la incorporación de los parámetros de penalización en el propio proceso evolutivo, siendo este el encargado de seleccionar los más adecuados.

Coello Coello [68] propusieron la siguiente función de penalización:

$$P(\boldsymbol{x}) = \alpha w_1 + \beta w_2 \tag{4.69}$$

donde $w_1, w_2 \in \mathbb{N}+$ son factores de penalización, $\beta \in \mathbb{N}+$ es igual al número de restricciones violadas y α es la suma de los valores del grado de violación $\Delta g_i(\boldsymbol{x})$ de las restricciones:

$$\alpha = \sum_{i=1}^{m} \Delta g_i(\boldsymbol{x}) \tag{4.70}$$

A continuación Coello utilizó de forma simultánea dos poblaciones de tamaños diferentes. La segunda de estas llevaba codificados los factores de penalización w_1, w_2 que serviría para evaluar la aptitud de la primera, de forma que la segunda población se encarga de optimizar los factores de penalización y la primera de optimizar el problema en cuestión. Los factores de penalización optimizados por la segunda población se aplican a la primera durante k generaciones ya que el proceso evolutivo es lento.

El principal inconveniente de este enfoque es que los valores de los tamaños de población y el factor k son dependientes del problema y una selección incorrecta de los mismos acarrea un gran número de evaluaciones de la función de aptitud.

Algoritmo Genético segregado

Propuesto por Le Riche et al. [225] utiliza dos subpoblaciones y dos funciones de penalización a diferencia de los otros métodos. El procedimiento de este método es el siguiente:

1. Se genera una población de tamaño 2μ.
2. Se evalúa a cada individuo según dos funciones de penalización, una con mayores factores de penalización y otra más relajada.
3. Se generan dos listas de tamaño μ donde los individuos son clasificados según la aptitud obtenida con una u otra función de aptitud.
4. Se juntan las dos listas y se eliminan los peores μ individuos.
5. Los μ individuos restantes conforman la población que será evolucionada según el método tradicional. Esta técnica es denominada *superelitismo* y proviene de las ES [345].

La ventaja de este método es que permite explorar mejor la frontera entre el espacio de soluciones factibles e no factibles. El inconveniente reside en la elección de los factores de penalización, la cual plantea el mismo problema que el resto de métodos anteriores.

Pena de muerte

Es el modo más sencillo de manejar las restricciones. Consiste en asignar una aptitud nula si se pretende maximinzar o muy grande si se pretende minimizar, cuando un individuo viola una restricción. Es un método que irremediablemente arrastrará la solución hacia un óptimo local, pero puede ser un método bueno cuando es muy difícil obtener soluciones factibles.

Es el método empleado, si no se aplica algún algoritmo de reparación, para eliminar los individuos ilegales del problema.

Penalización Fuzzy

Wu et al. [414] proponen un método de variación de los coeficientes de penalización basado en la lógica difusa o *Fuzzy*. La función de penalización utilizada es de la forma:

$$P(\boldsymbol{x}) = r\left[\sum_{i=1}^{m} H(g_i(\boldsymbol{x}))g_i(\boldsymbol{x}) + \sum_{j=1}^{m} |h_j(\boldsymbol{x})|\right] \tag{4.71}$$

donde $r = [0, r_{max}] \in \mathbb{R}^+$ es un coeficiente de penalización que se ajustará mediante una función Fuzzy y H es la función de Heaviside.

La función *Fuzzy* empleada esta compuesta de las variables f y p asociadas a valores difusos como *muy grande, grande, mediano, pequeño, muy pequeño* y están relacionadas con las funciones $f(\boldsymbol{x})$ y $P(\boldsymbol{x})$. Los rangos de estas variables son:

$$R_f = [f_{\min}, f_{\max}] \tag{4.72}$$

$$R_p = [0, p_{\max}] \tag{4.73}$$

donde $f_{\min}$ y $f_{\max}$ son los valores máximo y mínimo de la función de aptitud sin modificar en la generación t y $p_{\min}$ y $p_{\max}$ es el valor mínimo de la función de penalización en la generación t, es decir:

$$f_{\min} = \min(F(\boldsymbol{x}, t)) \tag{4.74}$$

$$f_{\max} = \max(F(\boldsymbol{x}, t)) \tag{4.75}$$

$$p_{\max} = \max(P(\boldsymbol{x}, t)) \tag{4.76}$$

Seguidamente definen los conjuntos borrosos A_i y $B_i, i = \{1, \ldots, m\}$ asociados a las variables f y p respectivamente. Finalmente se divide el rango de las variables f y p y del coeficiente de penalización en $k \in \mathbb{N}^+$ intervalos y se asigna una regla asociada a las variables f y p a cada intervalo del tipo: « Si f es *bajo* y p es *bajo* entonces $r = r_1$ ». El inconveniente de este método al igual que los demás es la determinación de los parámetros adecuados para $r_{\max}$ y k. Aunque los autores recomiendan $r_{\max} = f_{\max} - f_{\min}$ no hay ninguna base teórica o empírica para tal afirmación.

4.9.1.2 Algoritmos de reparación

En ciertos problemas como por ejemplo la secuenciación de tareas o rutas, es relativamente sencillo *reparar* individuos no factibles que sustituirán, según una probabilidad determinada de antemano, al individuo original en la población. No existe un algoritmo general de reparación ya que este depende del tipo de problema y su codificación.

Algunos de los trabajos más destacables sobre el uso de este tipo de algoritmos se encuentran en [225, 236, 262, 289, 290, 385, 417, 418]

4.9.1.3 Métodos híbridos

Dentro de esta categoría se incluyen aquellos métodos que no utilizan los EA para manejar las restricciones. Entre ellos destacan los siguientes:

Multiplicadores de Lagrange

Propuesto por Adeli y Cheng [7] el algoritmo combina una función de penalización con el método primal-dual. Esta aproximación se basa en la optimización secuencial de los multiplicadores de Lagrange, donde la función de penalización es de la forma:

$$P(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m}\gamma_i(\max[0, g_i(\boldsymbol{x}) + \delta_i])^2 \tag{4.77}$$

donde $\lambda_i > 0$ y δ_i es un parámetro relacionado el grado de violación máximo $\Delta g_j^{\max}$ de la restricción j en la iteración anterior:

$$\delta_i(t) = \beta\Delta g_j^{\max}(t-1) \tag{4.78}$$

donde $\beta \in \mathbb{R}^+$ es una constante predefinida por el usuario. Así mismo el parámetro γ_i es función de β y el propio valor en la iteración anterior:

$$\gamma_i(t) = f(\beta, \gamma_i(t-1) \tag{4.79}$$

A continuación se aplica el método primal-dual, calculando multiplicadores de Lagrange del siguiente modo: $\lambda_i = \gamma_i\delta_i$. En las pruebas realizadas por Adeli y Cheng el algoritmo tuvo un comportamiento satisfactorio, pero también es cierto

que este no evita el problema de selección de parámetros de las funciones de penalización.

Optimización restringida por evolución aleatoria (CORE)

Fue propuesta por Belur [38] y se basa en la idea de combinar un EA con una técnica de programación matemática como el método simplex de Nelder y Mead [276], de modo que cuando una solución no es factible, se procede a minimizar la siguiente función de penalización:

$$P(\boldsymbol{x}) = \sum_{i \in C_1} \tilde{h}_i(\boldsymbol{x})^2 - \sum_{j \in C_2} \tilde{g}_j(\boldsymbol{x}) \tag{4.80}$$

donde

$$C_1 = \{i = \{1, 2, \ldots, n\} \ : \ |\tilde{h}_i(\boldsymbol{x})| \leq \epsilon\} \tag{4.81}$$

$$C_2 = \{j = \{1, 2, \ldots, m\} \ : \ \tilde{g}_j(\boldsymbol{x}) \leq 0\} \tag{4.82}$$

donde ϵ es la tolerancia admitida en las restricciones de igualdad. Esta técnica tiene la ventaja de no requerir ningún tipo de parámetro, pero tiene los mismos inconvenientes que los algoritmos de reparación.

4.9.2 Técnicas de escalado

La resolución de un problema mediante Algoritmos Genéticos no siempre es sencilla ni viable. La elección del tipo de codificación, los operadores genéticos y sus parámetros, así como la función de aptitud condicionan la resolución del problema. Además de los condicionantes expuestos anteriormente, las funciones de aptitud presentan otros problemas.

Uno de los problemas más comunes de los Algoritmos Genéticos es su rápida convergencia a un óptimo local si la aptitud de los individuos es demasiado diferente. Esto es especialmente importante en aquellos problemas donde el espacio de soluciones factibles es pequeño en comparación con el global.

Otro de los problemas surge cuando el algoritmo converge y los individuos son muy próximos en términos de aptitud, lo cual hace que la presión selectiva desparezca.

Finalmente el algoritmo evoluciona de forma diferente para versiones traspuestas de una misma función de aptitud. De este modo dadas dos funciones de aptitud $f_1(x)$ y $f_2(x) = f_1(x) + K$ la segunda tiene la misma forma que la primera desplazada en el eje vertical, ambas funciones alojarán los diferentes máximos y mínimos en los mismos valores de x, pero sin embargo el proceso de optimización será diferente para cada función. La razón estriba en que la probabilidad de selección de los individuos es diferente en ambas funciones. Si la probabilidad de selección, mediante un operador de selección de ruleta, del individuo i evaluado por con la función de aptitud f_1 es según la ecuación (4.1):

$$p_{s1}^{\ i} = \frac{f_1^i}{\sum_{j=1}^{\mu} f_1^j} \tag{4.83}$$

la probabilidad de selección del mismo individuo, pero evaluado por la función de aptitud f_2 será:

$$p_{s2}^{\ i} = \frac{f_2^i}{\sum_{j=1}^{\mu} f_2^j} = \frac{f_1^i + K}{\sum_{j=1}^{\mu} f_1^j + K} = \frac{f_1^i + K}{K + \sum_{j=1}^{\mu} f_1^j} \tag{4.84}$$

de donde puede deducirse fácilmente que si $K > 0$ la probabilidad de selección de los individuos más aptos disminuye para f_2 y si $K < 0$ lo hace paraf_1.

Con el fin de evitar, en la medida de lo posible, estos problemas se desarrollaron las denominadas *técnicas de escalado*. Estas fueron clasificadas en tres grandes grupos por Goldberg: escalado lineal, truncado sigma, y escalado potencial.

4.9.2.1 Escalado lineal

Para evitar los problemas anteriores, Goldberg propuso emplear la técnica del escalado lineal. Esta técnica constituye una traslación de la función de aptitud original, con el fin de evitar el tercer problema, y una rotación con el fin de evitar o minorar los dos primeros:

$$\Phi'(\boldsymbol{x}) = a\Phi(\boldsymbol{x}) + b \tag{4.85}$$

donde $\Phi(\boldsymbol{x})$ es la función de aptitud original, $\Phi'(\boldsymbol{x})$ es la función de aptitud escalada, a y b representan la traslación y la rotación de la función de aptitud original.

La función de aptitud escalada puede definirse de diferentes maneras, aunque se debe cumplir la condición de que el valor promedio de la aptitud de la población escalada $\overline{\Phi}'$ debe ser igual al de la población sin escalar $\overline{\Phi}$:

$$\overline{\Phi}' = \overline{\Phi} \tag{4.86}$$

si los valores promedio de las funciones de aptitud son:

$$\overline{\Phi} = \frac{\sum_{i=1}^{\mu} \Phi(\boldsymbol{x}_i)}{\mu} \tag{4.87}$$

$$\overline{\Phi}' = \frac{\sum_{i=1}^{\mu} \Phi'(\boldsymbol{x}_i)}{\mu} \tag{4.88}$$

entonces

$$\sum_{i=1}^{\mu} \Phi(\boldsymbol{x}_i) = \sum_{i=1}^{\mu} \Phi'(\boldsymbol{x}_i) \tag{4.89}$$

De lo que se deduce que los individuos *promedio* de la población tendrán el mismo peso en la reproducción (ec. 4.86), mientras que los menos aptos verán aumentada su probabilidad reductora y los más aptos la verán disminuida (ec. 4.89).

Un segundo criterio para determinar los parámetros a y b es la limitación del número de hijos máximos C_{mult} que puede engendrar el individuo más apto. Para lograr esto se parte de la ecuación (4.2) donde se establecía el número de veces que el individuo era seleccionado para la reproducción κ_i en promedio. Combinando la ecuación (4.2) con las ecuaciones (4.85) y (4.86) podemos escribir el número de hijos en función de $\overline{\Phi}$:

$$C_{\text{mult}} = \frac{\Phi'_{\max}}{\overline{\Phi}'} = \frac{\Phi'_{\max}}{\overline{\Phi}} \tag{4.90}$$

donde $\Phi'_{\max}$ es la aptitud escalada del individuo más apto. Para tamaños de población pequeños ($n \in [50, 100]$) Goldberg recomienda $C_{\text{mult}} \in [1.2, 2]$ A partir

de esta relación los valores a y b pueden expresarse fácilmente como:

$$a = \left[\frac{C_{\text{mult}} - 1}{\Phi'_{\text{max}} - \overline{\Phi}}\right] \overline{\Phi} \tag{4.91}$$

$$b = \left[\frac{\Phi'_{\text{max}} - C_{\text{mult}}\overline{\Phi}}{\Phi'_{\text{max}} - \overline{\Phi}}\right] \overline{\Phi} \tag{4.92}$$

4.9.2.2 Truncado sigma

Uno de los problemas del escalado lineal es que no asegura que el parámetro a sea positivo, pudiendo producirse una traslación de la función de aptitud que genere valores de aptitud negativos, lo cual viola el principio de no negatividad de la función de aptitud. Este efecto suele producirse cuando el algoritmo se acerca a la convergencia, produciendo a partir de ese momento soluciones aberrantes.

Para evitar este problema Forrest y Tanese [119] sugieren emplear la desviación típica de la aptitud de la población para preprocesar la aptitud antes del escalado. De este modo la función de aptitud es modificada previamente mediante la expresión:

$$\Phi'(\boldsymbol{x}) = \Phi(\boldsymbol{x}) - (\overline{\Phi} - c\sigma) \tag{4.93}$$

donde $\Phi'(\boldsymbol{x})$ es la función de aptitud modificada previa al escalado, σ es la desviación típica de la aptitud de la población y c es un múltiplo razonable de la desviación estándar que usualmente toma valores $c \in [1, 3] \in \mathbb{R}$.

Finalmente, en caso de que la función de aptitud modificada fuera negativa, se le asignaría un valor nulo (truncado). La nueva función de aptitud que realiza el escalado será:

$$\Phi''(\boldsymbol{x}) = H(\Phi'(\boldsymbol{x}))(a'\Phi'(\boldsymbol{x}) + b') \tag{4.94}$$

donde $H(\bullet)$ es la función de Heaviside que realiza el truncado y a', b' son dos nuevas constantes de escalado que se calculan de forma análoga a la ecuación (4.92)

4.9.2.3 Escalado potencial

En este caso la función de aptitud original es escalada por un exponente k cercano a la unidad:

$$\Phi'(\boldsymbol{x}) = \Phi(\boldsymbol{x})^k \tag{4.95}$$

La selección del exponente k es dependiente del problema. El valor más comúnmente empleado es $k = 1,005$

5

Implementación del Algoritmo Genético GASOP

5.1 La librería GAlib

La librería de componentes de Algoritmos Genéticos GAlib fue desarrollada por Mattew Wall [404] como herramienta para llevar a cabo su tesis doctoral en el Massachusetts Institute of Technology. Esta librería está desarrollada en lenguaje C++, y su descarga y uso es gratuito bajo las condiciones reflejadas en la licencia escritas en la página `http://lancet.mit.edu/ga/`.

Se trata de una librería muy completa y compleja que permite utilizar diferentes tipos de codificaciones: binaria, gray, real, arbórea y listas. También incorpora diferentes tipos de operadores adaptados a cada codificación, así como la posibilidad de derivar nuevos tipos de codificaciones y operadores a partir de la librería de clases[1] original.

De entre todas las características, se destacan las empleadas en el desarrollo del presente trabajo:

Operadores de inicialización La librería incluye los siguientes operadores de inicialización: aleatoria, ordenada, puesta a cero y puesta a uno (para codificación binaria).

Operadores de selección La librería contiene los siguientes operadores de selección: por ruleta, por rango, por torneo, por muestreo universal estocástico, por muestreo estocástico del resto sin remplazo, por muestreo determinístico. Además, el operador puede ser fácilmente personalizado.

Operadores de cruce La librería contiene los siguientes operadores de cruce: de 1 o 2 puntos y uniforme para la codificación binaria y entera, recombinación uniforme aritmética (o cruce intermedio) y cruce por mezcla alfa ($\alpha = 0,5$) para la codificación real.

Operadores de mutación La librería incorpora los siguientes operadores de mutación: inversión de bits e intercambio de bits para codificación binaria y mutación gaussiana para codificación real.

Operadores de reemplazo La librería incorpora dos tipos de reemplazo: el simple o generacional (GGA) y el de brecha generacional (SSGA), pudiendo selec-

[1] En C++ es un conjunto de variables y procedimientos relacionados entre sí agrupados en un solo tipo

cionar el porcentaje o número de individuos de reemplazo. Incorpora también las siguientes estrategias de reemplazo para los SSGA: el del padre, del menos apto, del más apto y aleatorio. Se puede decidir si se emplea o no elitismo en los GGA. Además, el operador puede ser fácilmente personalizado.

Técnicas de escalado La librería también incorpora las siguientes técnicas de escalado de la aptitud de la población: lineal, truncado sigma y potencial.

Criterios de detención La librería dispone de los siguientes criterios de finalización: tras t generaciones, tras la convergencia de la aptitud durante t generaciones y estancamiento durante t generaciones.

5.2 La implementación de las clases GASOPGenome y GAMulti

La complejidad del problema a abordar no permite la utilización directa de las librería GALib, ya que esta es de propósito general y solo puede ser directamente aplicable para la resolución de problemas simples.

Seguidamente se detalla la implementación del algoritmo genético, adaptado a la resolución del tipo de problemas que se pretende resolver, en dos nuevas clases escritas en lenguaje C++ GASOPGenome y GAMulti, derivadas de las clases GAGenome y GADeme respectivamente. Estas clases incorporan una serie de operadores y procedimientos no presentes en la la librería original y constituyen el núcleo del Algoritmo Genético empleado en el presente trabajo.

5.2.1 Codificación de las variables de diseño

Dado que se pretende optimizar de forma simultánea el tamaño y la topología de una estructura, las variables serán múltiples y de muy diferente tipo como se verá posteriormente. Algunas de estas, como la conectividad entre nudos, estarán mejor representadas por un alfabeto binario, mientras otras como el tipo de sección serán mejor representadas por una codificación entera u otras como las coordenadas de los nudos lo serán por una codificación real.

Se impone pues el empleo de una codificación mixta, alejada de la codificación única tradicional. Al adaptar la codificación al tipo de problema, es posible em-

plear los operadores genéticos más apropiados a cada codificación, permitiendo al algoritmo adaptarse mucho más en la búsqueda de la solución que los algoritmos de codificación única como se verá en el capítulo posterior.

Esta codificación mixta, inspirada en el trabajo de Chen y Rajan [62], además separará por grupos a las variables de diseño, lo cual permitirá la utilización de operadores genéticos individualizados. A diferencia del método tradicional de codificación única, no solo se podrá definir un operador de cruce especializado para cada grupo de variables, sino también una estrategia de inicialización, de renacimiento, etc ...

Las variables que definen un problema de optimización estructural, pueden clasificarse en dos tipos: las topológicas y las estructurales. Seguidamente se detallan por separado, simplemente por mantener cierto orden y claridad. La codificación computacional en C++ sigue la misma cronología y criterios expresados a continuación.

5.2.1.1 Codificación de las variables topológicas

Tradicionalmente las variables topológicas se encuentran previamente predefinidas ya que se parte de diseños previos ya realizados (método romano), con lo cual el planteamiento y la resolución del problema se simplifica bastante.

Para introducir la topología como variable del problema a resolver debemos de definir previamente una ventana de trabajo que determine el espacio máximo que puede ocupar la estructura ya que de lo contrario el problema sería inabarcable. En el presente trabajo se predefine un contorno cúbico para el espacio de soluciones, aunque el algoritmo también sería adaptable a la definición de otro tipo de contornos. Así mismo se delimita también el número máximo de nudos j que puede tener la estructura con el fin también de limitar el espacio de búsqueda ya que el número de barras máximo k que puede tener la estructura está directamente relacionado con esta mediante la expresión:

$$k = \frac{j(j-1)}{2} \tag{5.1}$$

La topología quedará restringida por la localización de los nudos fijos: restricciones y puntos de aplicación de carga restringidos, así como, los nudos semifijos:

restricciones y puntos de aplicación de carga que están restringidos a una linea o un plano.

Los tipos de variables que por lo tanto definen la topología son: las direcciones libres de los nudos semifijos, la posición de los nudos móviles y la conectividad (barras) entre estos. Cada uno de estos será codificada de manera diferente, constituyendo un cromosoma independiente, de forma análoga a lo que ocurre en la naturaleza.

A continuación se detalla la codificación de cada uno de estos tipos:

Codificación de los nudos móviles Representan aquellos nudos sin restricciones. Los nudos móviles se pueden codificar mediante una matriz donde cada fila se corresponde con las coordenadas x, y, z de cada nudo móvil:

$$[\boldsymbol{N}_m] = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \cdots & \cdots & \cdots \\ x_m & y_m & z_m \end{bmatrix}_{3\times m} \quad \forall x, y, z \in \tilde{\boldsymbol{C}} \tag{5.2}$$

donde m representa el número de nudos móviles de la estructura. La cantidad de nudos móviles se obtendrá restando al número máximo de nudos j predefinido, el número de nudos fijos y semifijos. $\tilde{\boldsymbol{C}}$ representa el contorno del espacio de soluciones. Este espacio es continuo y por lo tanto la codificación se realizará dentro del dominio $\mathbb{R}$

La codificación tradicional binaria de los nudos limita sensiblemente las posibilidades de optimización del algoritmo. Al tener que discretizar las variables, se genera una dependencia de la solución con respecto a la resolución de la codificación, de forma similar a la dependencia respecto a la malla de puntos de las técnicas de programación lineal basadas en la *Ground Structure* descritas anteriormente [97]. La codificación real evita este inconveniente. La matriz de nudos móviles se codifica computacionalmente como un vector ya que aunque la librería permite utilizar codificaciones matriciales resulta más sencillo el manejo de vectores que de matrices. Su representación en forma vectorial será de la forma:

$$\boldsymbol{N}_m = \begin{pmatrix} x_1 & y_1 & z_1 & x_2 & y_2 & z_2 & \cdots & x_m & y_m & z_m \end{pmatrix}^T_{3m} \forall x, y, z \in \tilde{\boldsymbol{C}} \tag{5.3}$$

Codificación de los nudos semifijos Representan a aquellos nudos con algún grado de libertad restringido. Los nudos semifijos se codifican mediante un vector donde los elementos representan las coordenadas no restringidas (móviles). Estos se almacenan simplemente de forma consecutiva:

$$\boldsymbol{N}_s = \begin{pmatrix} c_1 & c_2 & c_3 & \cdots & c_s \end{pmatrix}_s^T \quad \forall c_i \in \boldsymbol{C} \quad , \quad i = \{1, 2, \ldots, s\} \tag{5.4}$$

donde s es el número de coordenadas no restringidas. $\tilde{\boldsymbol{C}}$ representa el contorno del espacio de soluciones. Este espacio es continuo y por lo tanto la codificación se realizará dentro del dominio $\mathbb{R}$

Codificación de la conectividad entre nudos La conectividad entre nudos se puede representar como una matriz binaria simétrica cuyas filas y columnas representan los nudos de la estructura:

$$[\boldsymbol{C}] = \begin{bmatrix} e_{11} & e_{12} & \ldots & e_{1p} & \ldots & e_{1q} & \ldots & e_{1j} \\ e_{21} & e_{22} & \ldots & e_{2p} & \ldots & e_{2q} & \ldots & e_{2j} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ e_{p1} & e_{p2} & \ldots & e_{pp} & \ldots & e_{pq} & \ldots & e_{pj} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ e_{q1} & e_{q2} & \ldots & e_{qp} & \ldots & e_{qq} & \ldots & e_{qj} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ e_{j1} & e_{j2} & \ldots & e_{jp} & \ldots & e_{jq} & \ldots & e_{jj} \end{bmatrix}_{j \times j} \tag{5.5}$$

Cuando existe conectividad entre los nudos p y q, es decir cuando existe una barra que une ambos nudos, el los elementos e_{pq} y e_{qp} de la matriz son iguales a la unidad. Si por el contrario no existe conectividad estos elementos serán iguales a cero. Por otra parte todos los elementos de la diagonal serán iguales a la unidad. Dada esta definición la codificación de la matriz de conectividad deberá realizarse de forma binaria

Esta forma de proceder es computacionalmente deficiente ya que se almacena información redundante, ocupando más espacio de almacenamiento y tiempo de computación del necesario. Una forma más eficiente de representar la matriz es

empleando la matriz diagonal superior de conectividad:

$$[\boldsymbol{C'}] = \begin{bmatrix} \epsilon_1 & \epsilon_2 & \dots & \dots & \dots & \epsilon_{j-1} \\ 0 & \epsilon_j & \dots & \dots & \dots & \epsilon_{2j-3} \\ 0 & 0 & \epsilon_{2j-1} & \dots & \dots & \epsilon_{\frac{3j-6}{2}} \\ 0 & 0 & 0 & \epsilon_{pj-\frac{p(p+1)}{2}} & \dots & \epsilon_{j(p-1)+j-\frac{p(p+1)}{2}} \\ \dots & \dots & \dots & \dots & \ddots & \dots \\ 0 & 0 & 0 & \dots & \dots & \epsilon_{\frac{j(j-1)}{2}} \end{bmatrix}_{(j-1)\times(j-1)} \tag{5.6}$$

donde ϵ_1 representa la conectividad entre el nudo 1 y el 2 y así sucesivamente. El subíndice p representa una fila de la matriz. Esta representación tiene la ventaja de tener un tamaño inferior y un acceso mas rápido al anidar dos bucles for . . . next relacionando el inicio del segundo con el índice del primero. Sin embargo no es una solución óptima en cuanto al tamaño y los tiempos de computación, ya que si bien no se requiere el acceso a los elementos de la diagonal inferior, es necesario inicializarlos.

La codificación empleada es una representación de tipo vectorial donde se almacenan solo los elementos de la matriz diagonal superior $[\boldsymbol{C'}]$. Las representaciones anteriores se han desarrollado para dar más claridad a este último vector y para facilitar la comprensión del método de detección de nudos inconexos explicado con posterioridad.

La representación definitiva del vector de conectividad es:

$$\boldsymbol{C} = \begin{pmatrix} \epsilon_1 & \epsilon_2 & \dots & \epsilon_{j-1} & \epsilon_j & \dots & \epsilon_{2j-3} & \epsilon_{2j-2} & \dots & \epsilon_{\frac{j(j-1)}{2}} \end{pmatrix}_k^T \quad \forall \epsilon \in \{0,1\} \tag{5.7}$$

Una vez definido el vector de conectividad, la conectividad e_{pq} entre los nudos p y q de la matriz de conectividad original vendrá almacenada en el elemento ϵ_r del vector de conectividad. Esta relación se puede deducir mediante la observación de la ecuación (5.6) :

$$e_{pq} = \epsilon_r \tag{5.8}$$

donde r viene dado por la ecuación:

$$r = j(p-1) + q - \frac{p(p+1)}{2} \quad \forall p < q \tag{5.9}$$

donde j es el número máximo de nudos de la estructura. Si no se cumple la condición $p < q$ bastará con permutar ambos índices dado la simetría de la matriz de conectividad.

5.2.1.2 Codificación de las variables estructurales

Dentro de este conjunto de variables se engloban aquellas variables que intervienen en un proceso de optimización de estructuras convencional, como son: materiales, tipo de sección y características geométricas de las mismas.

Codificación de los materiales En los procesos de optimización estructural no es habitual incorporar como variables directamente las características mecánicas de los materiales tales como: límite de resistencia a la fluencia o rotura, módulo de elasticidad, coeficiente de Poisson, etc... ya que a la postre se precisa trabajar con material existente en el mercado, o peor aún, con un conjunto de limitado de estos. Es por esto que la variable de diseño suele ser el tipo de material. Su implementación es bastante simple, basta con elaborar una lista y definir como variable la posición del material en la lista. Por lo tanto, la codificación del vector de materiales debe pertenecer al dominio $\mathbb{N}$.

Teniendo en cuenta este criterio, la codificación de los materiales se realiza mediante el vector:

$$\boldsymbol{M} = \begin{pmatrix} m_1 & m_2 & \dots & m_q & \dots & m_k \end{pmatrix}_k^T \quad \forall m_q \in \mathbb{N} \mid m_q \rightarrow \boldsymbol{L}_m \tag{5.10}$$

donde k es el número máximo de barras, m_q representa el material de la barra q, el cual se haya registrado en la fila m_q de la matriz $\boldsymbol{L}_m$ que conforma la lista de materiales.

Codificación del tipo de sección Al igual que los materiales, el tipo de sección se realiza mediante una lista ordenada, donde cada posición de la lista se corresponde con un tipo de sección. El tipo de sección se codifica de forma separada a la geometría de la sección porque la primera requiere que la codificación pertenezca al dominio $\mathbb{N}$, mientras que la segunda pertenece al dominio $\mathbb{R}^+$.

El vector de tipo de sección se representa como:

$$\boldsymbol{T} = \begin{pmatrix} t_1 & t_2 & \dots & t_q & \dots & t_k \end{pmatrix}_k^T \quad \forall t_q \in \mathbb{N} \mid t_q \rightarrow \boldsymbol{L}_s \tag{5.11}$$

donde t_q representa el tipo de sección de la barra q, el cual se haya registrado en la fila t_q de la matriz $\boldsymbol{L}_s$ que conforma la lista de tipos de sección. A continuación se reproduce la lista con los tipo de sección incorporadas al algoritmo GASOP, dentro de la clase *torsión*, representadas de forma ordenada:

1. ASEC: Sección arbitraria.
2. RECT: Rectangulo.
3. QUAD: Quadrilatero
4. CSOLID: Eje sólido
5. CTUBE: Tubo circular
6. CHAN: Perfil en U
7. I: Perfil en I
8. Z: Perfil en Z
9. L: Perfil en L
10. T: Perfil en T
11. HATS: Perfil de sombrero
12. HREC: Perfil rectangular hueco

Las características geométricas de estas secciones pueden encontrarse en apéndice A

Codificación de la geometría de la sección Esta codificación incorpora todos los parámetros geométricos necesarios para definir las secciones de la lista de tipos de sección más un parámetro extra colocado al final que sirve para definir la orientación del perfil. Resulta evidente que no se requieren el mismo número de parámetros para una sección circular que para una de forma arbitraria. Aunque sería posible definir una codificación donde el número de parámetros fuera variable,

esto plantearía ciertos problemas a la hora de implementar los operadores genéticos. Por esta razón se decidió emplear un enfoque diferente. Realizando de nuevo una analogía con la naturaleza, la mayoría de organismos celulares son diploides como se comentó anteriormente. El empleo de una codificación de longitud fija, sin ser una representación puramente diploide, puede asemejarse a esta. Si el individuo realmente queda definido por una parte de los genes del cromosoma (genes *activos*) se puede considerar que el resto del cromosoma (genes *inactivos*) actúan en cierto modo como una representación diploide proveyendo las características de memoria y diversidad genotípicas planteadas por Goldberg [135].

En base a estas consideraciones, la geometría de la sección se representa mediante la matriz:

$$[\boldsymbol{S}] = \begin{bmatrix} s_1^1 & s_2^1 & \dots & s_p^1 & \dots & s_{10}^1 \\ s_1^2 & s_2^2 & \dots & s_p^2 & \dots & s_{10}^2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ s_1^q & s_2^q & \dots & s_p^q & \dots & s_{10}^q \\ \dots & \dots & \dots & \dots & \dots & \dots \\ s_1^k & s_2^k & \dots & s_p^k & \dots & s_{10}^k \end{bmatrix}_{10\times k} \tag{5.12}$$

donde el p representa el número de parámetro y q el número de barra y k el numero máximo de barras. Los primeros nueve parámetros se definen la geometría de la sección y el último su orientación. La colocación de los parámetros dentro de la matriz no es arbitraria ya el valor numérico de los parámetros puede ser muy diferente incluso dentro de una misma sección. En un perfil en I, por ejemplo, los parámetros que definen las medidas de las alas y la altura tienen un valor muy grande en comparación con el espesor. Si estos se introdujeran sin ningún criterio dentro de la matriz, los operadores genéticos producirían una gran cantidad de individuos ilegales o no factibles, generando una gran disrupción en el funcionamiento del algoritmo.

Para evitar este problema, se establecen dos grupos de parámetros: un primer grupo compuesto por los cuatro primeros parámetros donde se codifican las variables de mayor tamaño y un segundo grupo formado por las cinco siguientes donde se codifican las variables de tamaño menor. De este modo, al actuar los operadores genéticos alterarán las variables dentro de rangos similares evitando la generación de elementos ilegales o no factibles. La existencia de genes *inactivos* intercalados

entre los activos dentro del genoma no plantea mayor problema desde el punto de vista del algoritmo que su programación.

Al igual que en los casos anteriores, la matriz se codifica en forma de vector que toma la siguiente forma:

$$\boldsymbol{S} = \begin{pmatrix} s_1^1 & \dots & s_{10}^1 & s_1^2 & \cdots & s_{10}^2 & \dots s_1^k & \cdots & s_{10}^k \end{pmatrix}_{10k}^T \quad \forall s_q^p \in \mathbb{R}^+ \tag{5.13}$$

La codificación de este vector puede pertenecer al dominio $\mathbb{R}^+$ en el caso más genérico, aunque también al dominio $\mathbb{N}^+$ o bien a un conjunto discreto de medidas si se desea emplear medidas comerciales.

Una vez definidos todos los cromosomas, el genotipo completo vendrá definido por la agregación de estos:

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{N}_m & \boldsymbol{N}_s & \boldsymbol{C} & \boldsymbol{M} & \boldsymbol{T} & \boldsymbol{S} \end{pmatrix}_{\omega}^T \tag{5.14}$$

El número de elementos del vector será:

$$\omega = 3m + s + 13\frac{j(j-1)}{2} \tag{5.15}$$

5.2.2 Los operadores genéticos

A pesar de que la librería recoge una gran cantidad de operadores genéticos, estos no son suficientes para abordar el tipo de problemas que se pretende resolver. Es por esto que se incorporan gran parte de los operadores descritos en el capítulo anterior y que se describen a continuación.

5.2.2.1 El operador de inicialización

Dado que en la codificación implementada se utilizan cromosomas con diferentes tipos de codificación, es necesario adaptar la inicialización a cada tipo de codificación. La inicialización empleada en todos los casos es del tipo uniforme, donde se genera un número aleatorio comprendido entre los límites inferior y superior de la variable. El número generado pertenecerá al dominio binario, $\mathbb{N}$, $\mathbb{R}$ o $\mathbb{R}^+$ dependiendo del tipo de codificación.

La evaluación de la aptitud de los individuos inicializados requiere del cálculo de tensiones y deformaciones de toda la estructura codificada por dicho individuo. Este cálculo requiere de un tiempo de computación importante, sobre todo si los tamaños de población o las estructuras a analizar son grandes. Este tiempo de computación de la función objetivo es muy superior al del propio algoritmo por lo que resulta de vital importancia que el individuo cuya aptitud va a ser evaluada sea legal. De no incorporar un procedimiento que detecte la incorporación de los individuos ilegales de la población el tiempo de cálculo total puede incrementarse sensiblemente, o lo que es peor, puede conducir al algoritmo fuera del espacio de soluciones factibles como se describe en el apartado 5.2.3.1.

Resulta pues imprescindible que el operador de inicialización incorpore algún mecanismo de detección de los individuos ilegales. Dado que los diferentes cromosomas son independientes entre sí la inicialización de cada uno de ellos se realiza por separado, comprobando la legalidad del cromosoma mediante bucles do ... until donde la condición será que el cromosoma en cuestión sea legal. Esto acelera el proceso de inicialización respecto a enfoques tradicionales ya que no es necesario inicializar todos los genes cada vez que un individuo es evaluado como ilegal, sino solo el cromosoma que genera la ilegalidad del individuo.

Además de la comprobación de la legalidad de los individuos, el operador de inicialización incorpora el siguiente mecanismo, no descrito en ningún trabajo previo, cuyo fin es incrementar la diversidad de las poblaciones generadas y ampliar el espacio de búsqueda. Dado que el espacio de búsqueda y la dificultad de generar individuos factibles es muy grande, conviene que los individuos generados inicialmente estén lo más distantes posibles entre sí. Para ello el algoritmo incorpora una rutina de cálculo que determina la distancia de Hamming existente entre el nuevo individuo generado y el resto de individuos de la población. Si este valor no supera un determinado valor que puede ser ajustado, el individuo es rechazado y se genera uno nuevo. Este mecanismo se ajusta a la tercera indicación de los trabajos de Rajan [310] y más recientemente Togan y Daloglu [393]

En el cálculo de la distancia de Hamming además se tiene en cuenta la presencia de genes inactivos, como se verá posteriormente, presentes en la codificación. Estos genes inactivos no son tenidos en cuenta a la hora de calcular dicha distancia.

5.2.2.2 Los operadores de reproducción

Los operadores de selección incorporados en la librería contemplan prácticamente todos los existentes en la actualidad y han sido suficientes para la validación del algoritmo, por lo que no se ha implementado ningún operador adicional.

Sin embargo los operadores de cruce de la librería son insuficientes para el trabajo con números reales por lo que se han incorporado los siguientes operadores de cruce descritos en el capítulo anterior:

- Cruce por mezcla alfa (BLX-α).
- Cruce binario simulado (SBX-β).
- Cruce unimodal de distribución normal (UNDX-m).
- Cruce parento-céntrico (PCX).
- Cruce simplex (SPX).

Además de estos operadores se ha incorporado una rutina de comprobación de la legalidad de los cromosomas generados. Esta forma de operar realiza en cierto modo las funciones de un operador de reparación ya que en lugar de descartar el individuo y volver a seleccionar y cruzar los padres, lo que se hace es volver a cruzar los padres de nuevo si el cromosoma no es legal. Esto supone una ventaja desde el punto de vista computacional y de convergencia como se verá en el próximo capítulo.

Finalmente, la separación del genotipo en varios cromosomas permite emplear los operadores de cruce más adecuados para cada cromosoma, cosa que no permite la codificación tradicional. Esto se traduce en una mejora de la convergencia y su velocidad como se verá en le próximo capítulo.

5.2.2.3 Los operadores de mutación

A los existentes en la librería se ha incorporado el operador de mutación aleatoria no uniforme con el fin de aplicarlo en los cromosomas con codificación dentro de $\mathbb{N}$.

Al igual que en el resto de operadores, sobre cada cromosoma actuará el operador de mutación más adecuado, comprobando la validez de estos una vez mutados. Si el cromosoma resultante no es legal, el operador volverá a actuar hasta generar un cromosoma legal.

5.2.2.4 El operador de migración

Además de los operadores tradicionales se ha incorporado un operador de migración. Este operador tiene la función de ayudar a mantener la diversidad de la población.

La aplicación de este operador requiere que el algoritmo evolucione de forma simultánea varias poblaciones. Para ello se desarrolla una nueva clase en C++, la clase GAMulti derivada de GADeme, que se encarga, entre otras tareas, de evolucionar varias poblaciones y regular la migración entre ellas.

El operador de migración tradicional denominado *stepping-stone* migra uno o varios individuos de una población a la población más próxima cada t generaciones.

Este operador presenta el inconveniente de que al migrar a los mejores individuos al final todas las poblaciones acaban convergiendo y comportándose como una sola con lo cual la diversidad solo se mantiene durante los primeros estadios del ejecución del algoritmo que es precisamente cuando existe mayor diversidad. Presenta además el inconveniente de que si una población encuentra rápidamente un óptimo local, al migrar esta solución rápidamente domina al resto de poblaciones que pierden su diversidad, convergiendo todas hacia una misma solución.

Para evitar estos inconvenientes se desarrolla un nuevo operador donde se copian los mejores individuos de todas las poblaciones menos la última denominada *población maestra*. Los individuos copiados reemplazan a los peores individuos de la *población maestra*. Mediante este método se logra que la *población maestra* realice las tareas de explotación, mientra que el resto realiza la exploración, ayudando a incrementar la diversidad de la *población maestra* cuando esta converge. Este operador transforma al resto de poblaciones, en cierto modo, en un micro-Algoritmo Genético.

5.2.2.5 El operador de renacimiento

Se trata de un operador no convencional sugerido por el propio Goldberg e introducido por Galante [124] y extendido por Greiner et al. [144] con el objetivo de emplearlo en con un micro-Algoritmo Genético.

Su funcionamiento es muy simple: cuando el algoritmo se estanca en una población, se reserva al mejor individuo y se reinicializa el resto. Este modo de proceder es contrario a la tercera conclusión del trabajo de Rajan [310], por lo que en el presente trabajo se implementa de un modo diferente.

El objetivo del operador de Greiner era proveer de diversidad a la población una vez se producía el estancamiento. Por otra parte la crítica realizada por Rajan [310] a este operador mantiene que la población resultante está muy cerca de la original, con lo cual el incremento de la diversidad se realiza a costa de un gasto computacional muy grande.

Para evitar ese inconveniente el algoritmo desarrollado en el presente trabajo reinicializa cualquier población menos la *población maestra* que al ser retroalimentada por las otras poblaciones verá incrementada su diversidad. Por otro lado, para evitar que el resto de poblaciones converjan rápidamente hacia una solución cercana a la inicial, como describió Rajan, la reinicialización se realiza eliminando completamente a todos los individuos del resto de poblaciones. Esta estrategia permitirá además que estas poblaciones exploren otras zonas del espacio de soluciones incrementando la capacidad de exploración del algoritmo sin perder capacidad de explotación.

5.2.3 La evaluación de legalidad de los individuos

Como se ha dicho anteriormente, la detección de individuos ilegales en la población resulta vital para el funcionamiento del algoritmo. La legalidad de un individuo depende de dos aspectos, uno topológico y otro geométrico: la existencia de nudos fijos o semifijos inconexos y la existencia de parámetros geométricos inválidos.

5.2.3.1 Detección de nudos y subestructuras inconexos

La existencia de un nudo de carga inconexo puede tener consecuencias catastróficas en la convergencia del algoritmo. Si un uno de estos nudos no está conexo al resto de la estructura, los motores de cálculo continuarán con el cálculo de la estructura que soportará evidentemente una carga menor. Al retornar los valores de tensión y deformación estos serán por lo general mucho más bajos que en el resto de individuos, empujando al algoritmo a reproducir dicho individuo con lo cual la población entera se desplazará fuera del espacio de soluciones factibles del problema.

La detección de los nudos inconexos se realiza de un modo relativamente sencillo mediante la matriz de conectividad. Aquellos nudos cuya fila (o columna) tengan todos sus elementos iguales a cero excepto el situado en la diagonal[2] se encontrará inconexo. Las figuras 5.1 y 5.2 muestran un ejemplo.

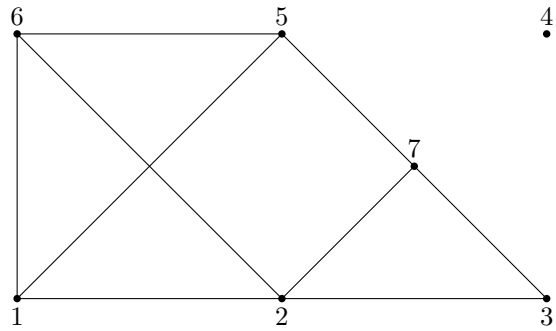


Figura 5.1: Nodo inconexo

$$[\boldsymbol{C}] = \begin{bmatrix} 1 & 1 & 0 & \mathbf{0} & 1 & 1 & 0 \\ 1 & 1 & 1 & \mathbf{0} & 0 & 0 & 1 \\ 0 & 1 & 1 & \mathbf{0} & 0 & 0 & 1 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 & \mathbf{0} & 1 & 1 & 1 \\ 1 & 1 & 0 & \mathbf{0} & 1 & 1 & 0 \\ 0 & 1 & 1 & \mathbf{0} & 1 & 0 & 1 \end{bmatrix}_{7\times 7}$$

Figura 5.2: Matriz de conectividad de la la figura 5.1

[2]representa la conectividad consigo mismo

La existencia de nudos móviles inconexos no supone ningún problema, de hecho es el mecanismo que tiene el algoritmo para reducir el número de nudos total de la estructura que inicialmente se fija de un modo arbitrario.

Sin embargo este método no nos previene de la existencia de subestructuras inconexas dentro de la estructura como el mostrado en la figura 5.3. Su implementación se justifica por el hecho de evitar un cálculo más laborioso en muchos casos. La detección de grupos inconexos puede realizarse mediante el empleo de la teoría de grafos, donde se recogen dos métodos: la búsqueda en anchura BFS, la búsqueda en profundidad DFS. El presente algoritmo implementa el primero por ser mas sencillo de programar. Este algoritmo no requiere de ningún tipo de información o guía aparte de la matriz de conectividad. Es de tipo determinístico y partiendo de un nodo cualquiera (*nodo padre*) se expande hacia los *nodos hijos*[3] añadiéndolos a una cola FIFO. Una vez añadidos todos los hijos a la cola, el nodo padre es apuntado en una lista que contendrá todos los nodos interconectados. Seguidamente los *nodos hijo* se convierten en *nodos padre* siguiendo la ordenación FIFO de la cola, añadiendo al final de la cola los nuevos hijos encontrados a la cola si estos no se encuentran en la lista. El proceso finaliza cuando se vacía la cola. De este modo, el algoritmo explora de modo exhaustivo todo el grafo (estructura).

Si el número de nodos de la lista coincide con el número total de nodos, toda la estructura estará conectada. En caso contrario existirán nodos o subestructuras desconectadas y por lo tanto el cromosoma o individuo será ilegal. En este punto puede optarse por dos procedimientos: sustituir el individuo por uno nuevo o bien reparar la estructura eliminando la subestructura desconectada (cromosomas) en caso de que no contenga ningún nodo de carga o restricción (operador de reparación). El algoritmo desarrollado implementa la segunda opción ya que esta permitirá una reducción del número de nodos definido inicialmente.

El algoritmo 11 muestra la implementación del mismo y la figura 5.3 muestra un ejemplo de aplicación. A continuación se ilustra el procedimiento del algoritmo a partir de la matriz de conectividad de la figura 5.3:

Siguiendo el algoritmo descrito anteriormente se realiza el siguiente procedimiento:

1. Partiendo del primer elemento e_{11} se introduce en la lista $\boldsymbol{v} = \{e_{11}\}$.

[3] nodos conectados con el *nodo padre*

Algoritmo 11 Pseudocódigo del algoritmo BFS

inicializar la lista : $\boldsymbol{v} = \{\emptyset\}$
crear la cola : $\boldsymbol{Q} = \{\emptyset\}$
introducir e_{11} en la cola Q: $e_{11} \rightarrow q_1 \in \boldsymbol{Q}$
apuntar e_{11} en la lista $\boldsymbol{v}$: $e_{11} \rightarrow v_1 \in \boldsymbol{v}$
while $Q \neq \{\emptyset\}$ **do**
  extraer el primer elemento q_1 de la cola Q: $e_{pq} \leftarrow q_1 \in \boldsymbol{Q} = \{q_1, q_2, \cdots, q_n\}$
  for $k = 0$ to j **do**
    if $e_{pk} \notin \boldsymbol{Q}$ **and** $e_{pk} \notin \boldsymbol{v}$ **then**
      if $e_{pk} = 1$ **then**
        introducir e_{pk} al final de la cola: $e_{pk} \rightarrow q_{n+1} \in \boldsymbol{Q} = \{q_1, q_2, \cdots, q_n\}$
        apuntar e_{pk} en la lista $\boldsymbol{v}$: $e_{pk} \rightarrow v_{n+1} \in \boldsymbol{v} = \{v_1, \cdots, v_n\}$
      end if
    end if
  end for
end while
return $\boldsymbol{v}$

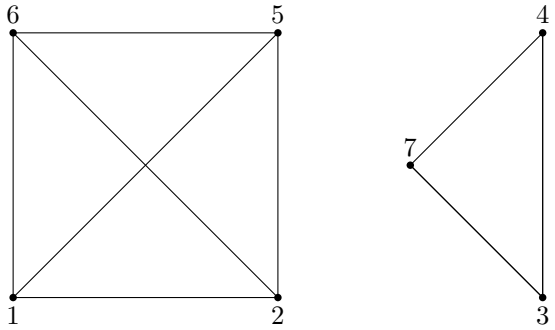


Figura 5.3: Subestructura inconexa

$$[\boldsymbol{C}] = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ & 1 & 0 & 0 & 1 & 1 & 0 \\ & & 1 & 1 & 0 & 0 & 1 \\ & & & 1 & 0 & 0 & 1 \\ & & & & 1 & 1 & 0 \\ & & & & & 1 & 0 \\ & & & & & & 1 \end{bmatrix}_{7\times 7}$$

Figura 5.4: Matriz de conectividad de la figura 5.3

2. Se introducen de forma secuencial los nudos conexos a e_{11} en la cola $\boldsymbol{Q} = \{e_{12}, e_{15}, e_{16}\}$ y en la lista $\boldsymbol{v} = \{e_{11}, e_{12}, e_{15}, e_{16}\}$.

3. Se extrae el primer elemento de la cola: e_{12}.

4. Se introducen de forma secuencial los nodos conexos a e_{12} que no estén previamente en la lista $\boldsymbol{v}$. Dado que no existe ninguno, no se introduce ningún nudo en la cola $\boldsymbol{Q} = \{e_{15}, e_{16}\}$.

5. Se introducen de forma secuencial los nodos conexos a e_{15} que no estén previamente en la lista $\boldsymbol{v}$. Dado que no existe ninguno, no se introduce ningún nudo en la cola $\boldsymbol{Q} = \{e_{16}\}$.

6. Se introducen de forma secuencial los nodos conexos a e_{16} que no estén previamente en la lista $\boldsymbol{v}$. Dado que no existe ninguno, no se introduce ningún nudo en la cola $\boldsymbol{Q} = \{\emptyset\}$.

7. La cola $\boldsymbol{Q} = \{\emptyset\}$ está vacía por lo que los nudos conectados son los contenidos en $\boldsymbol{v} = \{e_{11}, e_{12}, e_{15}, e_{16}\}$. Los nodos { 1, 2, 5, 6 } se encuentran unidos entre sí mientras que nodos restantes: { 3, 4, 7 } no están contendidos en la lista y por lo tanto constituyen una subestructura disjunta.

La implementación real del algoritmo es algo más compleja porque se trabaja con el vector de conectividad en lugar de la matriz, aunque esta complejidad no se traduce en un incremento del tiempo de computación.

5.2.3.2 Detección de parámetros geométricos inválidos

Como se ha descrito anteriormente, el cromosoma que aloja los parámetros geométricos almacena parámetros de tipos de secciones muy diferentes por lo que es posible que el rango fijado para cada gen del cromosoma no sea adecuado en todos los casos. Es por esto que resulta necesario comprobar que para cada uno de los tipos de secciones del individuo a evaluar se tengan unos parámetros geométricos coherentes.

5.2.4 Ajuste de los parámetros de los operadores genéticos

Como se vio en el apartado anterior, los parámetros de los operadores genéticos deben ser ajustados de algún modo para cada problema. Los parámetros a ajustar en el presente algoritmo son la probabilidad de cruce y de mutación.

De los tipos de ajuste existentes, se ha elegido un ajuste online con retroalimentación heurística, en concreto un control mediante lógica difusa, ya es un método robusto suficientemente probado en diversos trabajos [171, 172, 184, 363, 403].

Siguiendo el modelo desarrollado por Herrera y Lozano [171] se implementa un control de lógica difusa que ajusta las tasas de cruce y de mutación en función de dos medidas de la convergencia: distancia al umbral de aptitud (TF) y número de generaciones sin mejora de la aptitud del mejor individuo (UN); y una medida de la diversidad: la varianza de la aptitud (VF).

La tabla 5.1 muestra las reglas de control establecidas para las funciones de pertenencia de las variables difusas de entrada y salida reproducidas en las figuras 5.5 y 5.6. La implementación del proceso de fuzzyficación y defuzzyficación de los

Tabla 5.1: Reglas para el control Fuzzy de la tasas de mutación y cruce

TF	**UN**	**VF**	p_m	p_c
bajo	-	-	bajo	alto
medio	bajo	-	bajo	alto
medio	medio	-	medio	medio
alto	bajo	-	bajo	alto
alto	medio	-	medio	medio
-	alto	bajo	alto	bajo
-	alto	medio	alto	bajo
-	alto	alto	bajo	bajo

conjuntos borrosos descritos se ha realizado empleando la librería opensource[4], de libre uso y distribución *FuzzyLite v1.01. A Fuzzy Inference System written in C++* perteneciente a *Foundation for the Advancement of Soft Computing* que puede encontrarse en `http://code.google.com/p/fuzzy-lite/`. Las funciones de pertenencia empleadas, tal y como aparecen en las figuras son de tipo triangular o trapezoidal. El sistema borroso implementado es del tipo *Mamdani* [253] ,

[4]De código abierto

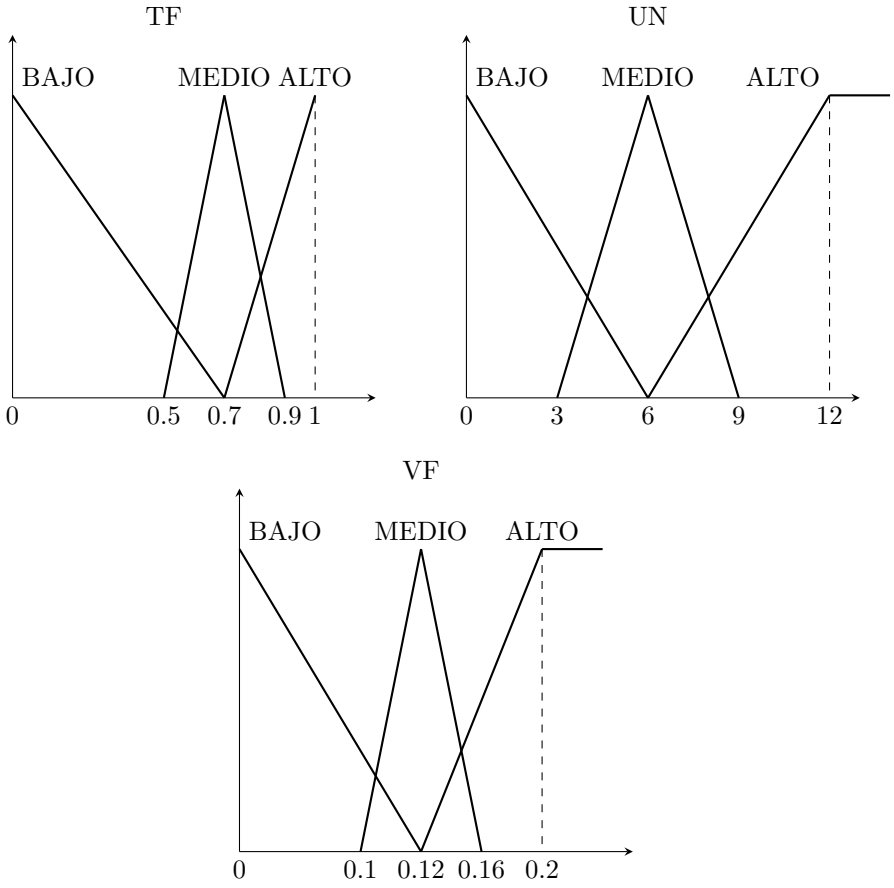


Figura 5.5: Significado lingüístico de las variables de entrada

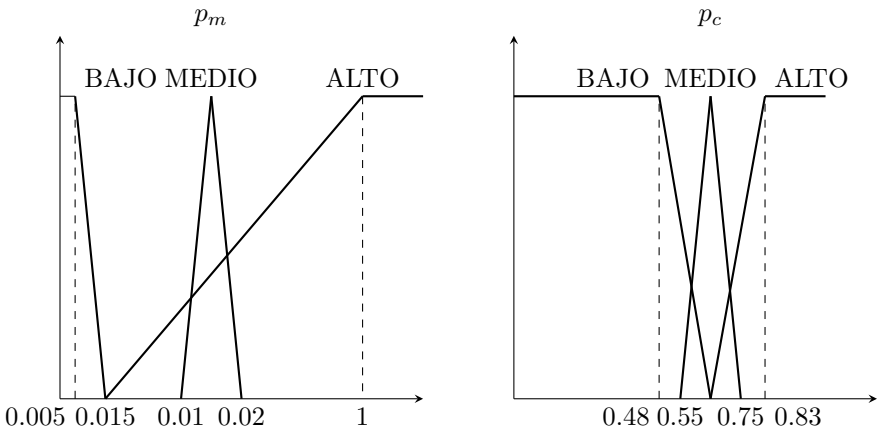


Figura 5.6: Significado lingüístico de las variables de salida

aunque puede ser fácilmente reemplazado por otros sistemas como el de Sugeno y Kang [378]. Las técnicas de inferencia[5] o razonamiento serán del tipo MAXMIN o truncamiento y la desborrosificación se realiza mediante el centro de gravedad (COG).

5.2.5 La función de aptitud

Una vez codificadas las variables de diseño y tras generar las poblaciones de individuos, el siguiente paso es evaluar la aptitud de los mismos. En los problemas de optimización estructural en los que se desea minimizar el peso o volumen de los elementos que componen una estructura, la aptitud viene proporcionada por el peso o volumen de la misma. Evidentemente, este proceso estará sujeto a ciertas restricciones ya que de lo contrario la solución óptima sería la trivial.

Las restricciones implementadas generalmente en los problemas de optimización estructural son de dos tipos: tensión máxima y deformación máxima. La limitación de tensión se establece normalmente en función de las características mecánicas del material empleado, mientras que las de deformación pueden venir limitadas por normas o para prevenir problemas de resonancia estructural. El presente algoritmo incorpora y comprueba ambas restricciones para cada una de las barras de la estructura. No se incorporan otro tipo de restricciones (p.e. asociados a normas de construcción) ya que se pretende generar un algoritmo de propósito general

[5]Determinación del grado de certeza de un conjunto de premisas de un conjunto borroso

aplicable no solo a estructuras de construcción sino también a otro tipo de estructuras como bastidores de vehículos, etc.... No obstante, es posible implementar sin demasiado esfuerzo otro tipo de restricciones.

El manejo de las restricciones se realiza mediante una función de penalización diferente a ninguna existente previamente y que constituye una de las aportaciones de la tesis. Como se analiza en el próximo capítulo, las funciones de penalización tradicionales no son adecuadas para el presente algoritmo. Debe recordarse que, según se describió anteriormente, la propia función de penalización no solo es dependiente del tipo de problema sino del propio algoritmo en sí.

El principal inconveniente de las funciones de penalización existentes reside la *regla de la penalización mínima* descrita en el capítulo anterior. El tipo de problemas a resolver añaden tres inconvenientes a esta regla:

En primer lugar, el espacio a explorar es muy grande mientras que el espacio de soluciones factibles está formado por regiones dispersas, no convexas y generalmente muy pequeñas en proporción con el espacio de soluciones no factibles, por lo que la probabilidad de encontrar una solución factible es también muy pequeña. Si el coeficiente de penalización es muy grande, el algoritmo convergerá rápidamente a un óptimo local al encontrar una solución factible, independientemente de lo lejos que se encuentre esta de el contorno del espacio de soluciones factibles.

En segundo lugar, la solución del problema no restringido es trivial, por lo que el óptimo global se encontrará localizado en el contorno del espacio de soluciones factibles. Además del óptimo global, es posible encontrar una gran cantidad de óptimos locales, algunos de ellos bastante alejados numéricamente (aunque puede que próximos en cuanto a localización) del global. Evidentemente el contorno del espacio de soluciones factibles no es sencillo de definir en términos de funciones finitas.

En tercer lugar, al ser la solución del problema restringido no trivial, si el coeficiente de penalización resulta demasiado pequeño no se generará nunca ningún individuo factible y la población entera permanecerá alejada de el contorno del espacio de soluciones factibles.

Estos tres factores hacen que sea del todo imposible la utilización de funciones de penalización estáticas, y que las funciones dinámicas existentes en la actualidad no

funcionen correctamente, como se analizará en el próximo capítulo. Esta situación queda perfectamente descrita por las figuras 5.7 a 5.9.

La función de penalización implementada incorpora un enfoque nuevo basado en el contorno del espacio de soluciones factibles. Esta fija un umbral de penalización que se puede establecer en base al problema a resolver o bien en base a simulaciones previas. Como es bien sabido, cuando se emplea una técnica heurística, esta debe ser ejecutada varias veces antes de considerar que se ha obtenido un óptimo global o un punto cercano a este[6], por lo que es posible utilizar estas ejecuciones previas para ajustar el umbral.

Una vez determinado el umbral ζ, tras seleccionar, cruzar y generar los nuevos individuos, se sigue el siguiente procedimiento con la población formada por la agregación de los individuos existentes mas los nuevos:

En primer lugar se calcula el peso de todos los individuos así como el grado de violación (normalizad) de cada restricción.

A continuación se modifica la aptitud de toda la población mediante la siguiente variante del método de penalización de Gen y Cheng [129] visto en la ecuación (4.63) del capítulo anterior:

$$P(\boldsymbol{x}, t) = 1 \pm \frac{r}{m} \sum_{i=1}^{m} (\Delta g_i(\boldsymbol{x}, t)) \tag{5.16}$$

donde r es un factor de penalización adaptativo. En el capítulo de siguiente se toma $r = 1$.

Seguidamente se ordena toda la población según la aptitud resultante del paso anterior y se busca al primer individuo no penalizado. Si este tiene un peso inferior al umbral ζ, este remplaza dicho valor.

A continuación se busca al primer individuo penalizado de la población y se registra su masa m_{vbest} y su grado de violación global Δg_{vbest}:

$$\Delta g_{\text{vbest}} = \frac{r}{m} \sum_{i=1}^{m} (\Delta g_i(\boldsymbol{x}, t)) \tag{5.17}$$

[6]Salvo en funciones simples que puedan ser trazadas, nunca se puede tener la certeza matemática de que se ha alcanzado el óptimo global.

Después se calcula un nuevo factor de penalización mediante la expresión:

$$r = \begin{cases} 1 & , \; m_{\text{vbest}} \geq \zeta \\ \dfrac{1}{\Delta g_{\text{vbest}}} \left[\dfrac{\zeta}{m_{\text{vbest}}} - 1 \right] & , \; m_{\text{vbest}} < \zeta \end{cases} \tag{5.18}$$

A continuación se vuelve a evaluar y ordenar toda la población mediante la ecuación (5.16) utilizando el coeficiente r resultante de la ecuación (5.18).

Finalmente se eliminan los individuos sobrantes según el operador de reemplazo seleccionado.

La figura 5.7 muestra gráficamente el primer inconveniente. El coeficiente de penalización r es igual a la inversa de la pendiente de la recta oblicua. Si el coeficiente es demasiado grande la pendiente se hace pequeña escalando la aptitud de todos los puntos que violan la restricción $g(x)$ por encima de la aptitud del mínimo local d. Si durante la evolución se produce algún individuo factible, el algoritmo encontrará rápidamente el punto d expulsando a todos los individuos no factibles de la búsqueda, con lo cual se eliminará la capacidad de exploración del algoritmo de la zona intermedia (contorno). Sólo será posible alcanzar el punto a de forma aleatoria (mutación), lo cual hace bastante improbable la obtención del mínimo absoluto.

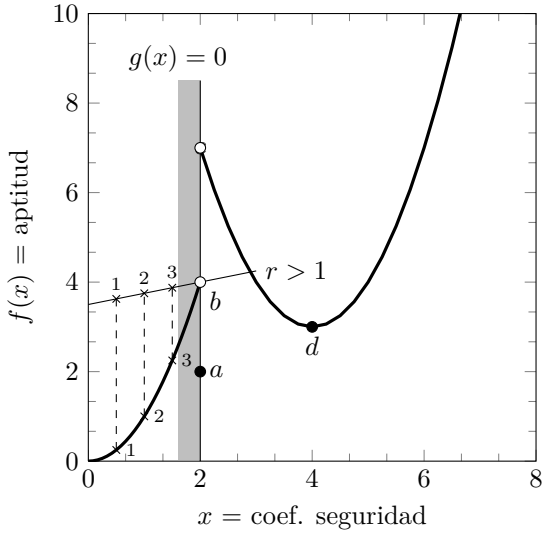


Figura 5.7: Efecto de una función de penalización demasiado restrictiva

La figura 5.8 muestra gráficamente el tercer inconveniente. El escalado de la aptitud de los individuos no factibles es insuficiente. Si bien se evita la caída en el óptimo local d, tampoco es posible aproximarse al punto a porque los nuevos individuos generados se agruparán cerca de la solución trivial ya que esta, aun penalizada, tiene una aptitud superior al punto a. Debido a esto aún en el caso de localizar este punto, será desechado por tener una aptitud superior a la de la solución trivial.

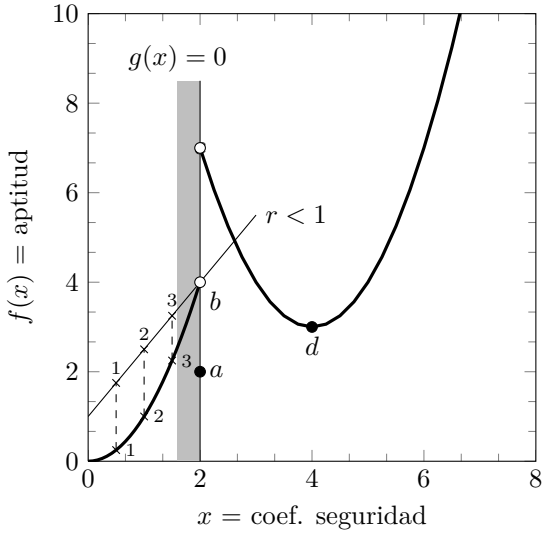


Figura 5.8: Efecto de una función de penalización demasiado laxa

La figura 5.9 muestra un ejemplo gráfico del funcionamiento de la función de penalización descrita. En primer lugar se establece un umbral en un punto cercano a la solución a. Dado que esta es desconocida, se localiza algo por debajo del mínimo relativo conocido d. Si la población contiene a los individuos 1,2,3 el individuo de menor peso será el 1. A continuación se le asigna a este individuo una aptitud igual al umbral, penalizando a todos los individuos que se encuentren bajo el umbral según la nueva recta r'. Como puede apreciarse en la figura, tras el reescalado, todos los puntos se encuentran sobre el umbral relativamente próximos entre sí en términos de aptitud. También se encuentran próximos, en términos de aptitud, al mínimo relativo d por lo que este tampoco es expulsado de la población pudiendo finalmente lograrse individuos localizados entre ambos (gracias al operador de cruce) y por lo tanto asegurando la localización del punto óptimo a.

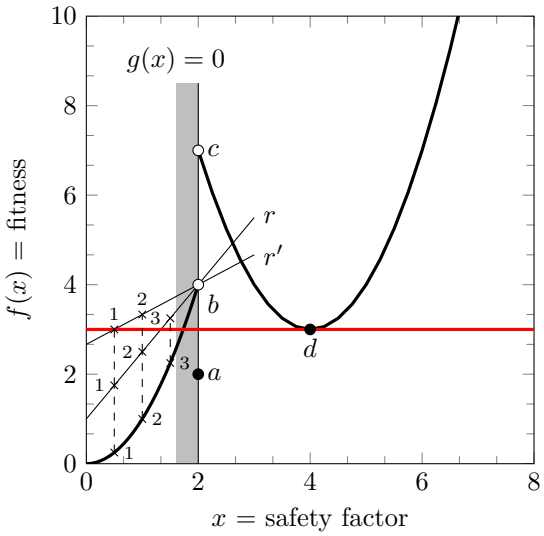


Figura 5.9: Ejemplo del funcionamiento de la función de penalización. Escalado inicial

5.2.6 El procesamiento en paralelo

Dado que la complejidad de los problemas a resolver es elevada, resulta imprescindible el empleo de más de un ordenador para realizar dicha tarea. Se hace pues necesario el desarrollo de un método de procesamiento en paralelo. Los métodos de procesamiento en paralelo son muy diversos y van desde el procesamiento de la aptitud de los individuos de una población en paralelo a la evolución de poblaciones en paralelo.

El procesamiento de la aptitud en paralelo presenta varios problemas que pueden comprometer el rendimiento del algoritmo. En primer lugar, si el motor de cálculo es externo, cada vez que tiene que realizar un cálculo existe un tiempo muerto de lectura e inicialización del programa que puede superar incluso el propio tiempo de cálculo. En segundo lugar, para aplicar el operador de remplazo necesitamos haber evaluado la población completa, por lo que el tiempo de evaluación total se corresponderá con el tiempo máximo empleado en evaluar un individuo. Si el clúster empleado para el cálculo no está formado por máquinas de características similares, el tiempo de procesamiento dependerá de la mas lenta.

A diferencia del procesamiento en paralelo, la evolución de poblaciones en paralelo nos permite realizar un procesamiento asíncrono ya que cada ordenador evolu-

ciona sus propias poblaciones de forma independiente. En el presente trabajo se ha desarrollado un modo de trabajo en paralelo muy simple, pero extraordinariamente robusto que permite utilizar máquinas con configuraciones diferentes, parar temporalmente o incluso conectar y desconectar alguna o todas estas máquinas sin perjuicio del proceso evolutivo.

El procedimiento seguido para implementar la evolución de poblaciones en paralelo es el siguiente: al iniciar el programa se selecciona mediante una opción que se va a ejecutar en modo *esclavo*, indicándole la dirección de las máquinas hacia las que debe apuntar. Esta dirección puede ser una dirección de red o simplemente una dirección IP donde se puedan copiar archivos. Una vez en marcha el programa este almacena físicamente durante cada generación el genotipo del mejor individuo de todas las poblaciones de esa generación y a continuación copia dicho archivo en la dirección a la que apunta el esclavo.

La máquina que actúa como *maestro* al final de cada generación lee los ficheros que le han enviado los *esclavos*, incorporándolos en la última población. La máquina que actúa como *maestro* puede ser a su vez esclavo de otras *máquinas*. La máquina que actúa como *maestro* no tiene ningún requerimiento especial, ni siquiera de velocidad de procesamiento. El único requisito es que el tamaño de su población debe ser lo suficientemente grande en relación al número de nodos *esclavos* para evitar que estos saturen su población.

El método de trabajo en paralelo desarrollado permite utilizar máquinas con diferentes configuraciones, sistemas operativos o motores de cálculo, lo cual lo hace tremendamente versátil. La estructura del clúster de cálculo es totalmente abierta y reconfigurable en caliente ya que los mejores genomas quedan almacenados físicamente, pudiendo reiniciar el programa apuntando hacia otros nodos. Tras la parada, si se desea, al reiniciar el programa es posible releer los genomas almacenados del mismo o cualquier otro nodo, con lo cual este continuaría el proceso evolutivo donde lo dejó.

5.3 El motor de cálculo estructural

El motor de cálculo empleado para la estructura ha sido el Ansys APDL v12.1. Aunque a primera vista pudiera parecer que un motor de cálculo integrado en el propio programa pudiera ser más rápido que la llamada a un motor o programa externo, esto no es necesariamente cierto. La versión citada permite utilizar la tecnología *multithreading*, utilizando al máximo los diferentes núcleos de los procesadores actuales. Sin el empleo del *multithreading*, el motor de cálculo solo utilizará uno de los núcleos de procesamiento. Por otro lado la versión 13 incorpora la tecnología *Kuda* que permite utilizar las GPU de la tarjeta gráfica para acelerar los cálculos a realizar. Alguna de las tarjetas con este tipo de tecnología incorporan hasta 300 núcleos de procesamiento, convirtiendo a la máquina en un pequeño clúster de cálculo. La programación de un multithreading o la utilización del Kuda de forma eficiente para este tipo de cálculos en C++ no es una tarea trivial.

Por otro lado, se trata de un programa de eficiencia más que probada que permite realizar cálculos estructurales complejos con materiales con comportamiento no lineal. Las librerías en C++ de cálculo de estructuras basadas en el método de los elementos finitos existentes en la actualidad no son tan eficientes ni versátiles como este programa, por lo que finalmente se ha optado por incorporarlo como motor de cálculo. No obstante, como se ha dicho anteriormente, el algoritmo es independiente del motor de cálculo, por lo que sería relativamente sencillo la incorporación de otros motores de cálculo al mismo. Para ello solo sería necesario programar un traductor adecuado entre el nuevo motor de cálculo y el algoritmo.

El tipo de elemento empleado para representar la estructura ha sido el BEAM44: *Viga asimétrica ahusada elástica tridimensional*. Este elemento permite modelizar cualquier tipo de estructura de forma suficientemente precisa. Tiene la capacidad de soportar tracción, compresión torsión y flexión. Cada elemento tiene seis grados de libertad: traslaciones y rotaciones alrededor de los tres ejes. Permite tener diferentes geometrías asimétricas en cada extremo del elemento así como el descentramiento de la geometría respecto del eje centroidal. La figura 5.10 esquematiza las características geométricas de este elemento.

La elección de un elemento de viga en lugar de la barra convencional se corresponde con la primera indicación del trabajo de Rajan [310] que establecía que entre el 10

y el 30% de las de las barras resultantes eran inestables y que debería emplearse el Método de los Elementos Finitos como método de cálculo empleando elementos del tipo viga.

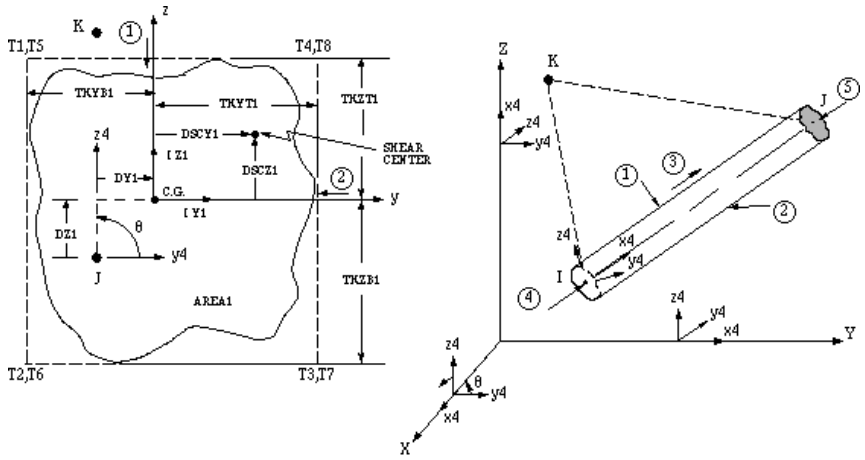


Figura 5.10: Geometría del elemento BEAM44

La definición del elemento requiere, además de los parámetros correspondientes y los nodos extremos, de un nodo de orientación K que proporcionará la orientación de la sección. Los parámetros que definen la geometría se encuentran codificadas en los nueve primeros parámetros reales del cromosoma que codifica la geometría, mientras que la orientación se encuentra almacenada en la décima posición. La definición del tipo de geometría se define mediante el comando SECTYPE cuyas geometrías se aprecian en la figura 5.11 y que se corresponde con la codificación realizada en el cromosoma de tipos de sección.

Este comando se combina con el comando SECDATA definiendo de manera completa la geometría de cada sección. En el anexo apéndice A se detallan los parámetros necesarios para cada tipo de sección.

Una vez escritos todos los datos del problema: materiales, posición de los nudos, interconexión de estos, geometrías, parámetros, etc..., en un lenguaje comprensible para el APDL el motor realiza el cálculo de las tensiones y deformaciones y realiza la escritura de un archivo donde quedan registradas la tensión combinada de tracción flexión máxima en la barra así como el momento torsor máximo de la barra, el volumen y la deformación máxima de la misma.

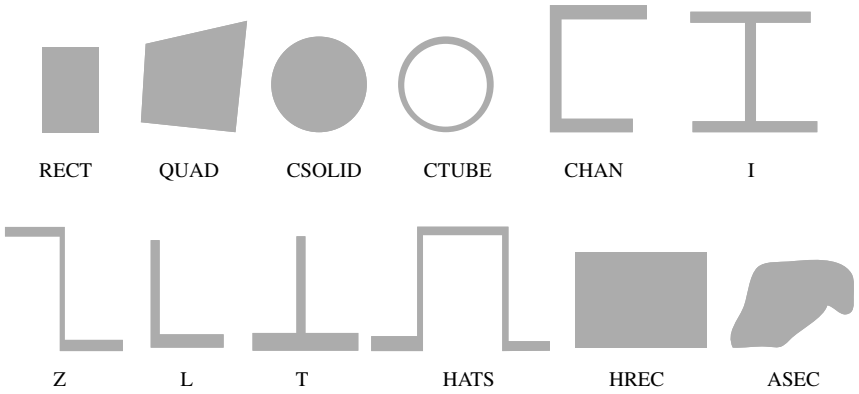


Figura 5.11: Tipos de sección seleccionables mediante el comando SECTYPE

Tras la escritura de los ficheros de resultados el motor devuelve el control al programa que pasa a interpretar los resultados. Cuando se inicializa el cromosoma que contiene los valores geométricos almacenados en el genoma se calculan los parámetros geométricos, según la formulación reflejada en el libro *Roark's formulas for stress and strain* [425], que permiten el cálculo de la tensión de cortadura generada por la torsión y que se reproducen en el anexo apéndice A. Para realizar este cálculo, se implementa la clase *torsión*.

Una vez calculadas las tensiones de cortadura se calcula la tensión equivalente de von Mises, para la misma, según la fórmula:

$$\sigma_{eq} = \sqrt{\sigma^2 + 3\tau^2} \tag{5.19}$$

donde σ es la tensión de flexión y tracción (o compresión) combinadas y τ es la tensión de cortadura generada por la torsión y la flexión simple.

Una vez calculada la tensión equivalente de Von Mises y la deformación máxima de cada barra se procede a comparar estos valores con la tensión y deformación máxima admisible, calculando el grado de violación de cada restricción y a partir de este el global, según se describió anteriormente. El valor del grado de violación es almacenado como un parámetro más del genoma y sirve para evaluar la aptitud del elemento.

A partir del volumen de cada barra, multiplicando este por la densidad de cada barra, se obtiene el peso total de la estructura que constituye el valor de la aptitud del individuo sin contemplar el grado de violación de las restricciones.

6

Validación del Algoritmo Genético GASOP

6.1 El método de validación

Ya desde los inicios de los Algoritmos Genéticos, los investigadores intentaron desarrollar métodos para validar sus algoritmos. El primero de ellos fue De Jong [83] que estableció una serie de funciones donde los métodos de optimización convencionales fallaban. El criterio establecido fue que si un algoritmo era capaz de localizar los puntos óptimos de esas funciones en menor tiempo (o menor número de evaluaciones de la función objetivo) que otro, este último era mejor. Sin embargo el teorema Wolpert y Macready [411] establece que un mismo algoritmo no puede funcionar bien en todos los casos, por lo que a partir de ese momento se empiezan a establecer otro tipo de funciones o problemas tipo en función del tipo de problema a resolver.

En la optimización de estructuras mediante Algoritmos Genéticos uno de los primeros trabajos al respecto fue el de Rajan [310], el cual estableció una serie de problemas tipo que han continuado utilizándose durante los años posteriores para validar y evaluar la bondad de los Algoritmos Genéticos aplicados a la optimización estructural. Este trabajo se ha convertido en un referente en la literatura de optimización estructural

Entre estos problemas tipo destaca uno de los problemas clásicos más populares en la optimización de estructuras: el problema de la estructura de seis nudos y diez elementos, propuesto por Venkayya et al. [401], representado en la figura 6.1. Según muestra muestran las tablas 6.1, 6.2 y 6.3 se trata de un problema ampliamente estudiado y empleado como banco de prueba por numerosos investigadores y de plena vigencia en la actualidad.

Para esta estructura, la distancia entre los nodos 1-2 , 2-3 y 3-4 es de 9140 mm quedando el resto de la geometría definida por estos nodos. La fuerza F aplicada en los nodos 2 y 3 es de 445 kN, siendo el material empleado para la fabricación de la estructura un aluminio con un módulo de elasticidad E = 68950 MPa, un coeficiente de Poisson $\nu = 0{,}33$, una tensión máxima admisible $\sigma_{\text{adm}} = 172$ MPa y una densidad $\rho = 2,76 \cdot 10^{-6} \frac{\text{kg}}{\text{m}^2}$. Los nudos restringidos son el 1 y el 6 que trabajan como una articulación. La deformación máxima admisible para la estructura medida sobre los tres ejes es de $\delta_{\max} = 50,8$ mm. El tipo de elemento empleado comunmente para la resolución es la viga, que considera los efectos axiales pero no la flexión.

Este problema es bien conocido, y ha sido resuelto por métodos de optimización convencionales y hasta la aparición del trabajo de Rajan el peso mínimo de la estructura estaba en torno a los 22,17 kN. Previamente al trabajo de Rajan ya hubieron otros intentos de resolver el problema mediante el empleo de Algoritmos Genéticos. El propio Goldberg [140] ya realizó una aproximación aunque sin restricciones de desplazamiento. Más tarde Rajeev y Krishnamoorthy [311] también resolvieron el problema con un peso de 24,98 kN, una marca algo peor que la existente. La diferencia entre el trabajo de Rajan y los anteriores trabajos es que, por primera vez, se aborda la optimización topológica del problema, lográndose a partir de ese momento una disminución drástica del peso de la estructura.

En el problema planteado por Rajan, a diferencia de los trabajos anteriores, se permitió variar la topología de la estructura estructura añadiendo y quitando elementos y nudos excepto los nodos de aplicación de cargas y restricciones. Además se permitió variar la posición del nudo 6 en el eje vertical, manteniendo fijas la posiciones de los nodos 1,2 y 3.

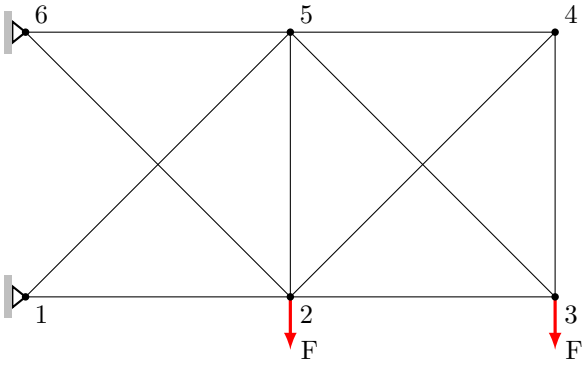


Figura 6.1: Estructura de diez barras y seis nudos

Este problema ha sido ampliamente estudiado desde entonces por numerosos autores cuyos resultados se recogen en las tablas 6.1 a 6.3, siendo el mejor resultado obtenido hasta la fecha el proporcionado por [100] con un peso de 12,04 kN.

Debido a su amplia aceptación como problema de referencia, el presente trabajo utiliza este problema como método de validación aunque con una pequeña variación. Como se comentó anteriormente, se pretende que el algoritmo sea útil para casos más generales donde interviene la flexión. Además, el problema tradicional adolece de una grave deficiencia. Las luces resultantes entre nudos son demasiado

Tabla 6.1: Comparativa entre los resultados obtenidos por diferentes autores para el problema de diez barras.

Autor	**Año**	**Peso** (kN)	$\boldsymbol{\delta_{max}}$ (mm)	$\boldsymbol{\sigma_{max}}$ Mpa	**Algoritmo**	**Codificación**	**Tamaño pob.**	**Numero de generaciones**
Ebenau et al. [100]	2005	12,04	50,800	131,5	ES	real	30	n.d.
Balling et al. [21]	2006	12,17	50,800	131,9	GA	real	n.d.	n.d.
Tang et al. [384]	2005	12,52	50,795	127,1	GA	real	70	544
Rajan [310]	1995	14,27	50,546	107,2	GA	real	40	n.d.
Ai y Wang [9]	2011	14,31	n.d.	n.d.	GA	binaria	n.d.	n.d.
Groenwold et al. [146]	1999	18,66	n.d.	n.d.	GA	entera	20	n.d.
Kaveh y Shahrouzi [198]	2006	19,27	n.d.	n.d.	GA	binaria	30	300
Schutte et al. [344]	2003	20,50	n.d.	n.d.	PSO	-	-	-
Lee y Geem [226]	2005	20,80	n.d.	n.d.	HS	-	t	-
Li et al. [233]	2007	20,81	n.d.	n.d.	PSO	-	-	-
Wu y Tseng [415]	2010	21,05	50,800	128,5	DE	-	-	-
Kaveh y Shahrouzi [199]	2008	22,06	n.d.	n.d.	GA+ACO	binaria	50	100
Deb y Gulati [89]	2001	21,06	50,800	131,6	GA	real	450	189
Nanakorn et al. [275]	2001	22,08	n.d.	n.d.	GA	n.d.	70	100
Isaacs et al. [186]	2008	22,10	n.d.	n.d.	ES	real	5	-
Ruy et al. [327]	2001	21,10	n.d.	n.d.	MOGA	n.d.	500	n.d.
Memari y Fuladgar [260]	1994	22,17	52,068	n.d.	MP	-	-	-
Galante [123]	1992	22,19	51,511	n.d.	GA	binaria	200,0	-
Camp y Bichon [53]	2004	22,22	n.d.	n.d.	ACO	-	-	-
El-Sayed y Jang [105]	1994	22,31	51,133	n.d.	MP	-	-	-
Camp [52]	2007	22,36	n.d.	n.d.	BBBC	-	-	100
Perez y Behdinan [299]	2007	22,36	n.d.	n.d.	PSO		-	-
Adeli y Kamal [8]	1991	22,48	51,295	n.d.	GA	n.d.	n.d.	n.d.
Sonmez [364]	2011	22,50	50,800	n.d.	ACO	n.d.	n.d.	n.d.

Tabla 6.2: Comparativa entre los resultados obtenidos por diferentes autores para el problema de diez barras (continuación).

Autor	**Año**	**Peso** (kN)	$\boldsymbol{\delta_{max}}$ (mm)	$\boldsymbol{\sigma_{max}}$ Mpa	**Algoritmo**	**Codificación**	**Tamaño pob.**	**Numero de generaciones**
Farshi et al. [113]	2010	22,50	n.d.	n.d.	MP	-	-	-
Kaveh et al. [200]	2009	22,50	n.d.	n.d.	HPSOACO	-	-	-
Kaveh et al. [201]	2009	22,50	n.d.	n.d.	HPSOACO	-	-	-
Togan y Daloglu [393]	2008	22,50	50,917	n.d.	GA	n.d.	n.d.	n.d.
da Silva et al. [77]	2008	22,50	n.d.	n.d.	DE	-	-	-
Li et al. [233]	2007	22,50	n.d.	n.d.	HPSO	-	-	-
Ali et al. [11]	2003	22,50	n.d.	n.d.	GA	n.d.	n.d.	
Schmit y Miura [340]	1976	22,59	50,797	n.d.	MP	-	-	-
Rizzi [322]	1976	22,59	50,351	n.d.	MP	-	-	-
Xu et al. [420]	2010	22,60	n.d.	n.d.	GA	binaria	2000	400
Rajeev et al. [312]	1997	22,60	n.d.	n.d.	GA	binaria	50	171
Galante [124]	1996	22,60	n.d.	148,2	GA	binaria	200	160
Dobbs y Nelson [93]	1976	22,61	50,799	n.d.	MP	-	-	-
Venkayya et al. [401]	1969	22,63	50,764	n.d.	MP	-	-	-
Schmit y Farshi [337]	1973	22,65	50,760	n.d.	MP	-	-	-
Ghasemi et al. [131]	1999	22,68	51,092	n.d.	GA	binaria	n.d.	n.d.
Jenkins [194]	2002	22,70	48,260	n.d.	GA	real	20	200
Cai y Thierauf [51]	1996	22,70	n.d.	n.d.	GA	real	(10+10)-ES	n.d.
Schmit y Miura [339]	1975	22,73	50,798	n.d.	MP	-	-	-
Gellatly et al. [128]	1971	22,75	50,800	n.d.	MP	-	-	-
Ismail et al. [187]	2010	22,90	n.d.	n.d.	MP	-	-	-
Shih [353]	1997	23,97	51,771	n.d.	Fuzzy	-	-	-

Tabla 6.3: Comparativa entre los resultados obtenidos por diferentes autores para el problema de diez barras (continuación).

Autor	**Año**	**Peso** (kN)	$\boldsymbol{\delta_{max}}$ (mm)	$\boldsymbol{\sigma_{max}}$ Mpa	**Algoritmo**	**Codificación**	**Tamaño pob.**	**Numero de generaciones**
Galante [124]	1996	24,29	n.d.	n.d.	GA	n.d.	n.d.	n.d.
Wu y Lin [416]	2004	24,43	n.d.	n.d.	GA	real	100	500
Mahfouz et Al. [248]	1999	24,43	n.d.	n.d.	GA	binaria	80	100
Chen et al. [64]	2010	24,43	n.d.	n.d.	ACO	-	-	-
Li et al. [234]	2009	24,60	n.d.	n.d.	HPSO	-	-	-
Rajeev et Al. [311]	1992	24,98	n.d.	n.d.	GA	binaria	30	20
Missoum et al. [272]	2002	42,36	50,800	-	MP	-	-	-
Botello et al. [49]	1999	58,68	n.d.	n.d.	GA	n.d.	5	1000
Kaveh [196]	2003	96,85	n.d.	n.d.	GA	binaria	50	100

importantes para despreciar el efecto de la flexión, más aun cuando uno de los condicionantes de la estructura es la deformación máxima. Este efecto es tan importante que para determinadas secciones las deformaciones asociadas a la flexión pueden ser mucho mayores que las generadas por las cargas axiales desvirtuando totalmente el proceso de búsqueda. Si se pretende validar un algoritmo aplicable a estructuras reales resulta pues imprescindible que el algoritmo trabaje con elementos tipo viga. En cualquier caso, la solución obtenida empleando un elemento tipo viga cumplirá las condiciones establecidas en el problema tradicional al tratarse de un caso más restrictivo.

6.2 Configuración óptima

La configuración óptima del algoritmo que ha permitido disminuir el peso de la estructura, frente a los trabajos publicados con anterioridad es la siguiente:

1. Tamaño de la población: $\mu = 20$ individuos.
2. Número de poblaciones: 4.
3. Operador de inicialización: aleatorio con comprobación de legalidad y distancia al resto de individuos de la población.
4. Operador de selección: selección por torneo.
5. Operadores de cruce: Operador β de cruce binario simulado SBX-β ($\eta = 1$) para los cromosomas móviles y semifijos, uniforme para el cromosoma de conectividad, uniforme real para los cromosomas de material y tipo de sección, Operador β de cruce binario simulado truncado SBX-β ($\eta = 1$).
6. Operador de mutación: convolución gaussiana adaptado a cada codificación.
7. Operador de remplazo: de brecha generacional SSGA con remplazo del individuo menos apto.
8. Operador de migración: migración de los mejores individuos del resto de poblaciones a la principal, descrita en el capítulo anterior.
9. Operador de renacimiento: reinicialización completa de la población (menos la principal) cuando la diversidad genotípica de esta, calculada según la

ecuación (4.43), y normalizada mediante la expresión la ecuación (6.2) cae por debajo de 10^{-4}

10. Ajuste de la tasa de cruce y mutación: Fuzzy, descrita en el capítulo anterior

11. Manejo de las restricciones: función de penalización basada en el umbral de penalización desarrollada para el presente trabajo en el capítulo anterior.

Mediante esta configuración se ha conseguido obtener un nuevo peso mínimo de 10,725 kN, el cual es un 10,92% menor que el mejor valor publicado hasta la fecha. Como se verá más adelante, no se ha obtenido un solo individuo, sino once en un intervalo comprendido entre el mejor y 10,745 kN.

En los siguientes apartados se procede a analizar para cada uno de los operadores y parámetros descritos su influencia en el comportamiento del algoritmo. Para poder comparar los resultados se han mantenido constantes el resto de parámetros salvo el que se desea analizar respecto de la configuración óptima descrita para el algoritmo.

6.3 Análisis del operador de inicialización

Como se ha descrito anteriormente el operador de inicialización implementado tiene una doble vertiente: por un lado genera individuos aleatoriamente comprobando su legalidad y repitiendo el proceso hasta conseguir un individuo legal y por otro se asegura que el nuevo individuo se encuentre alejado del resto de individuos de la población.

Las figuras 6.2 y 6.3 muestran el comportamiento del algoritmo frente a tres formas de inicialización de los individuos de la población: la óptima implementada en el algoritmo, la aleatoria con comprobación de legalidad y la aleatoria convencional. Sobre el eje horizontal se representa el número de generaciones transcurridas y sobre el vertical la aptitud normalizada resultante. Se emplea este método para poder comparar gráficamente la influencia de los diversos factores. La normalización se realiza mediante la ecuación:

$$f_{\text{norm}} = \frac{f_{\text{best}}}{f_{\text{eval}}} \tag{6.1}$$

donde $f_{\text{best}} = 12,04$ kN se corresponde con el mejor resultado de aptitud publicado hasta la fecha, mientras que f_{eval} es la aptitud obtenida con el presente algoritmo bajo diferentes parámetros u operadores.

La aptitud representada se corresponde con la de la última población, que como se ha descrito, contiene a los mejores individuos. La representación se corresponde con el mejor valor obtenido en cinco ejecuciones. Las figuras han sido suavizadas para apreciar mejor las tendencias.

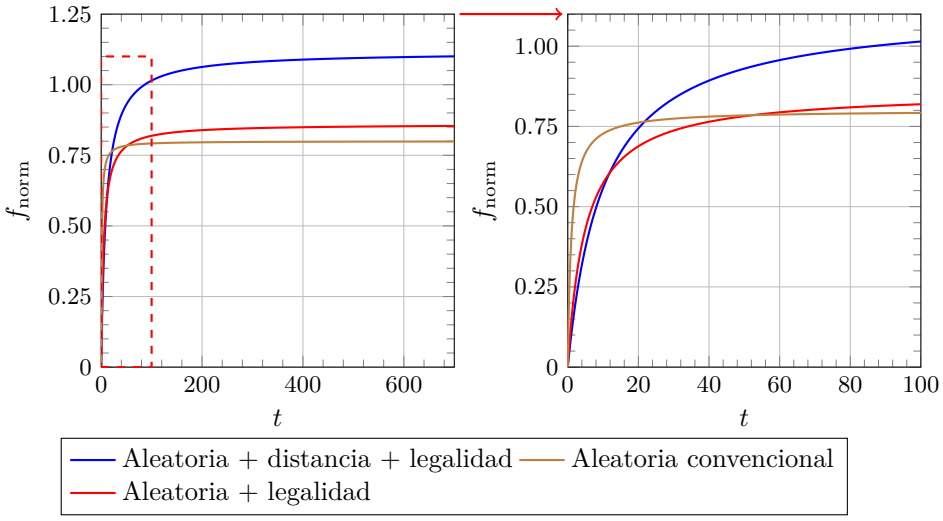


Figura 6.2: Influencia del operador de inicialización en la aptitud

Observando las figuras 6.2 y 6.3 puede verse como una inicialización aleatoria tradicional presenta problemas ya que la diversidad inicial de la población es muy pequeña, cayendo rápidamente en un óptimo relativo y estancándose a las pocas generaciones. Con una inicialización aleatoria con comprobación de la legalidad de los individuos, la diversidad inicial es algo mayor, pero no lo suficiente como para asegurar la exploración necesaria, cayendo también rápidamente en un óptimo local. La dispersión de los resultados entre las sucesivas ejecuciones del algoritmos es bastante importante por lo que en estos casos se comporta más como una búsqueda aleatoria que a un procedimiento de optimización en sí.

Otro tipo de representación muy empleado para medir la eficiencia de un algoritmo es la representación de la aptitud versus el número de evaluaciones de la función de aptitud. A diferencia de la anterior, esta última nos da una idea más aproximada

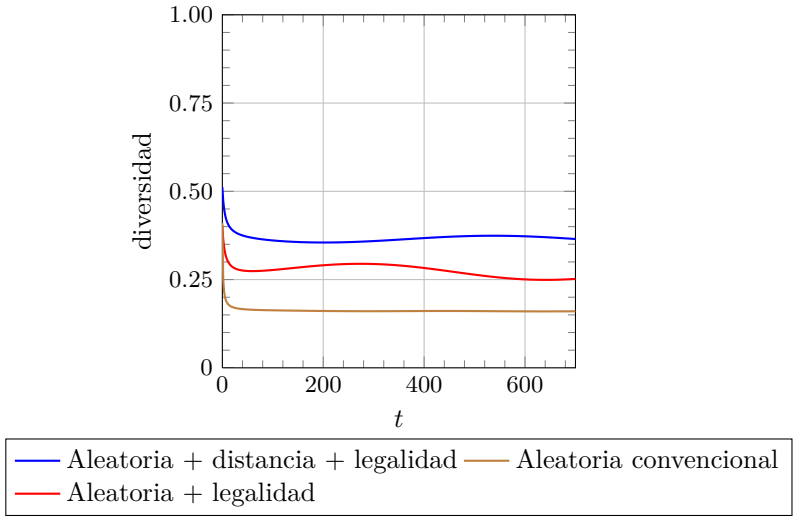


Figura 6.3: Influencia del operador de inicialización en la diversidad

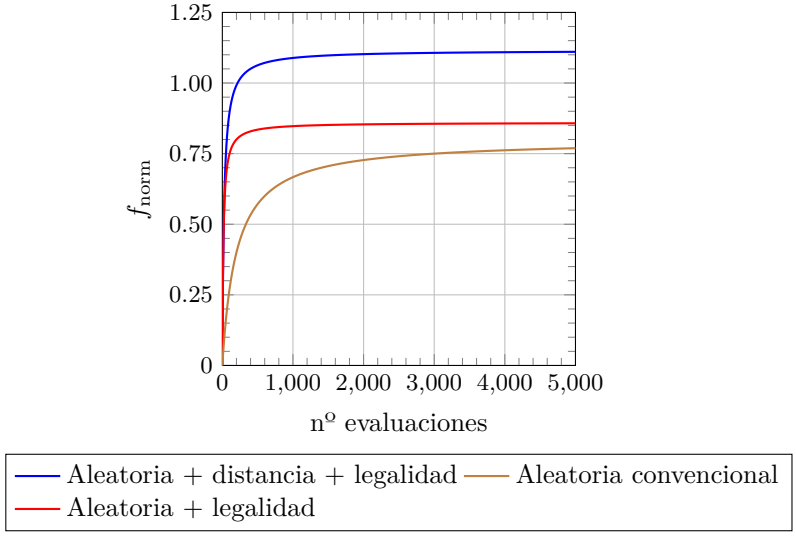


Figura 6.4: Número de evaluaciones de la función de aptitud para diversos operadores de inicialización

del tiempo de computación del algoritmo. Observando la figura 6.4 se puede apreciar el incremento de eficiencia del algoritmo al incorporar la comprobación de la legalidad de los individuos generados. Mientras que con una inicialización sin comprobación de la legalidad necesitamos 5000 evaluaciones para alcanzar la convergencia, con la comprobación de la legalidad solo se requieren unas 1000.

6.3.1 Análisis de los operadores de selección

En la figura 6.5 puede comprobarse que el comportamiento de todos los operadores es bastante similar, presentando el mejor comportamiento la selección por torneo y peor la selección por ruleta y el muestreo determinístico.

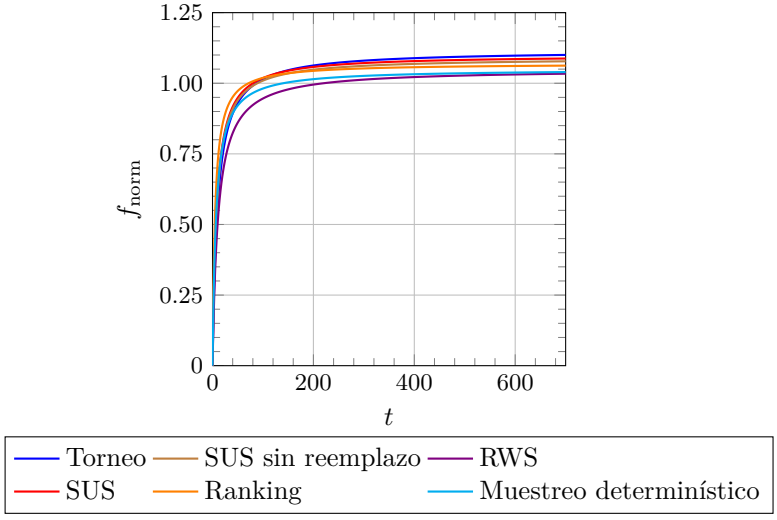


Figura 6.5: Influencia del operador de selección en la aptitud

6.3.2 Análisis de los operadores de cruce

Las figuras 6.6 y 6.7 analizan en primer lugar el cruce por genotipo, sin considerar operadores diferenciados para cada cromosoma. Dado que el genotipo está compuesto por diferentes codificaciones solo pueden utilizarse operadores de corte por uno o varios puntos. Como se puede apreciar el operador de cruce por un punto presenta una rápida convergencia acompañada con una pérdida rápida de diversidad en la población. El operador de cruce por dos puntos presenta un comporta-

miento algo mejor, con una convergencia algo más lenta y un mantenimiento algo mayor de la diversidad de la población. Estos operadores tienen una capacidad de exploración muy reducida, por lo que su convergencia es muy rápida lo que se traduce en un rendimiento del algoritmo pobre.

El operador de cruce uniforme presenta una convergencia muy lenta. Esto es debido a que es altamente disruptivo, sobre todo en su actuación sobre el cromosoma que almacena la geometría de la sección, impidiendo la formación de los esquemas imprescindibles en el proceso evolutivo. En cualquier caso presenta un rendimiento muy pobre bastante lejos de la aptitud óptima.

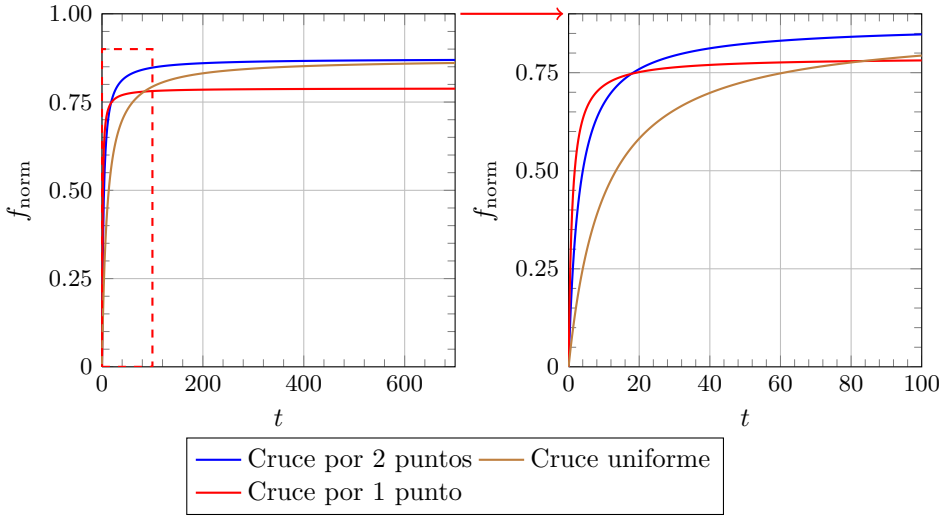


Figura 6.6: Influencia del operador de cruce por genotipo en la aptitud

Las figuras 6.8 a 6.12 analizan los diferentes operadores de cruce por cromosoma para cada grupo de cromosomas con idéntica codificación. Se omite la representación de la diversidad en aquellos operadores donde no es significativa.

Como puede apreciarse en la figura 6.8, para el cromosoma de conectividad, el operador de cruce uniforme tiene mejor comportamiento de los tres, al contrario de lo que ocurría en el cruce por genotipo. Esto es debido a que este tipo de cruce sí permite incrementar el espacio de búsqueda en la codificación binaria sin el inconveniente de afectar al resto de la codificación.

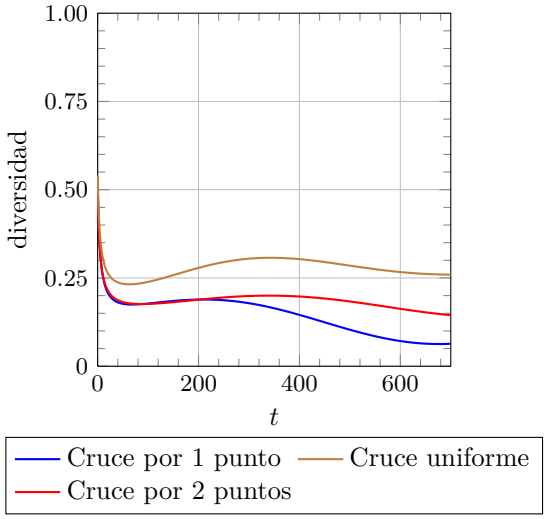


Figura 6.7: Influencia del operador cruce genotípico en la diversidad

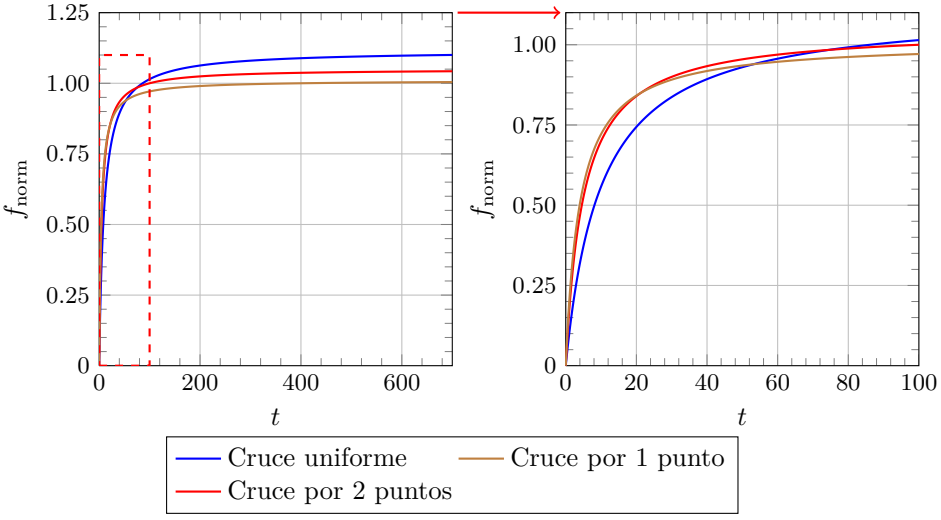


Figura 6.8: Influencia del operador de cruce sobre el cromosoma de conectividad y la aptitud

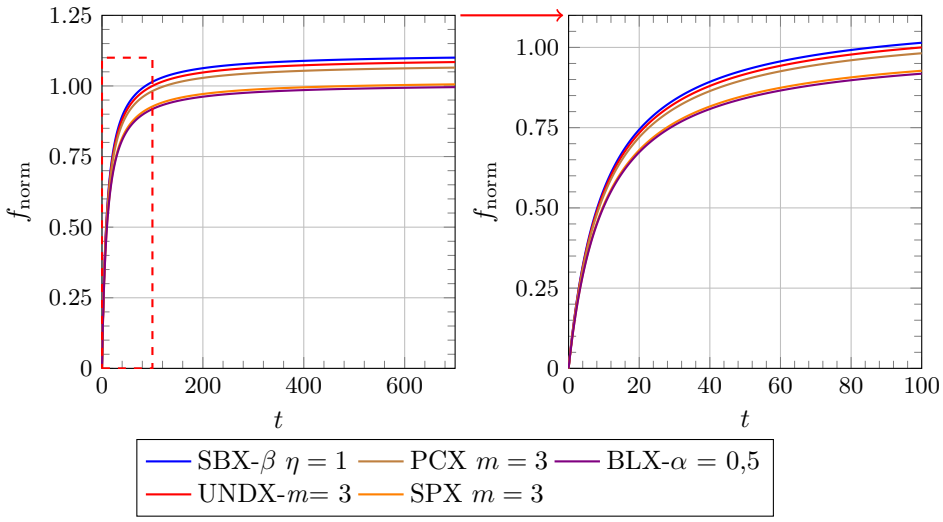


Figura 6.9: Influencia del operador de cruce sobre los cromosomas de nudos móviles y semifijos y la aptitud

Según la figura 6.9 todos los operadores de cruce tienen un comportamiento similar para los cromosomas de nudos móviles y semifijos, teniendo un comportamiento ligeramente mejor el SBX-β ($\eta = 1$) y peor el BLX-α = 0,5 debido a que en ocasiones es más disruptivo. Dado que el operador SBX-β requiere de menos cálculos y tiene mejor comportamiento de todos es el elegido como óptimo. Este resultado coincide con el publicado en un trabajo previo por Deb y Gulati [89] quien también evaluó la estructura de diez barras y cuyos resultados aparecen en la tabla 6.1.

Dado que los cromosomas de tipos de sección y material tienen una codificación entera y los tamaños de las listas asociadas a los cromosomas no son muy elevadas, se implementa un cruce por uno o varios puntos. La figura 6.10 muestra el comportamiento frente a los diferentes tipos de cruce de estos cromosomas. Al igual que ocurría con el cromosoma de conectividad, el cruce uniforme presenta un mejor comportamiento que los otros dos.

Finalmente para el cromosoma que contiene los parámetros geométricos se implementan los operadores de cruce aplicados a números reales, pero truncados a números enteros ya que la geometría de la sección así lo requiere. No se emplean los operadores de cruce por uno o varios puntos ya que el rango de las variables

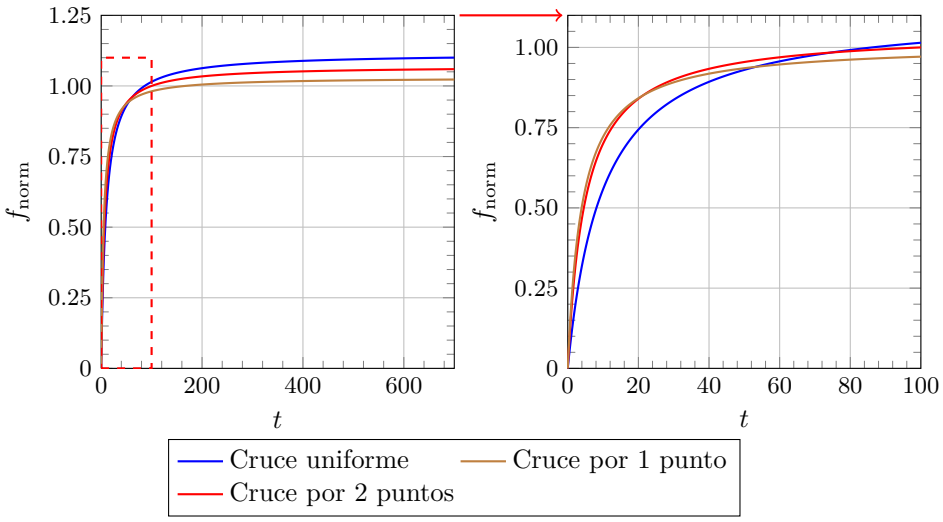


Figura 6.10: Influencia del operador de cruce sobre los cromosomas de material y tipo de sección

codificadas puede ser bastante grande y la experiencia previa con una codificación con cruce por genotipo indica que este tipo de operadores no son adecuados para este cromosoma.

Las figuras 6.11 y 6.12 muestra los efectos de los diferentes operadores. Como se puede apreciar, aquellos que son más explorativos presentan un peor comportamiento frente a los operadores parentocéntricos lo cual concuerda con el comportamiento frente al cruce por uno o varios puntos. La mezcla de los parámetros de sección en este caso dificulta la búsqueda por la generación de individuos legales pero tan deformes que no son factibles. Puede apreciarse en estos casos que le convergencia es mucho más elevada debido precisamente a este efecto, al producirse pocos individuos estos copan rápidamente el proceso evolutivo. El operador con mejor comportamiento es el SBX-β ($\eta = 1$), el cual fue diseñado específicamente con el fin de favorecer la formación de esquemas en el proceso evolutivo, tal y como se vio anteriormente.

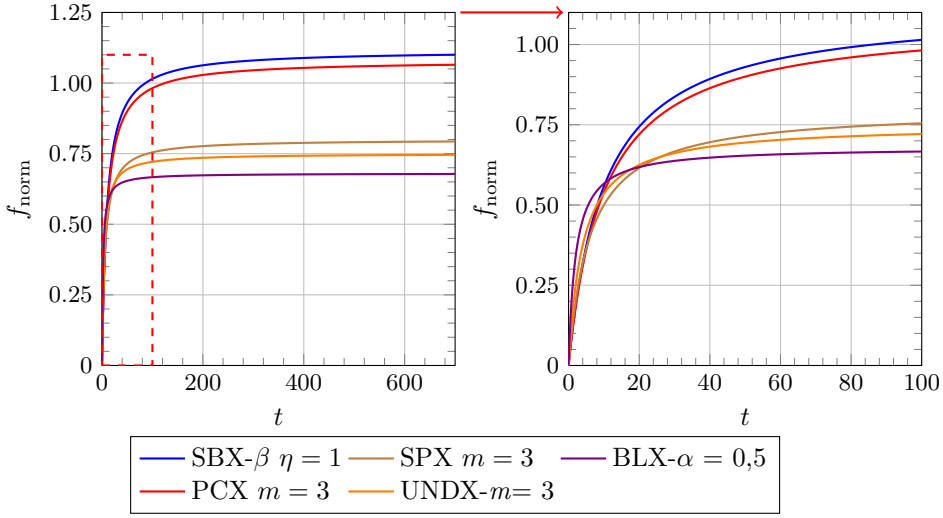


Figura 6.11: Influencia del operador de cruce sobre el cromosoma de geometría y la aptitud

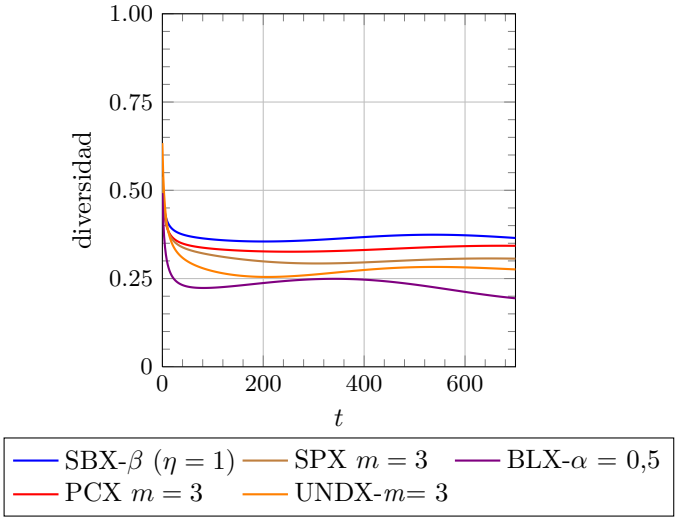


Figura 6.12: Influencia del operador de cruce sobre el cromosoma de geometría y la diversidad

6.4 Los operadores de mutación

El operador de mutación empleado para todos los cromosomas ha sido el operador de convolución gaussiana, adaptado a cada codificación. A pesar de incorporar el operador de mutación aleatoria para los cromosomas con codificación $\mathbb{N}$, este no ha tenido mejor comportamiento que el operador de convolución gaussiana, el cual gracias a la dispersión de los genes resultantes permite mejorar las capacidades exploratorias del algoritmo, a costa de ralentizar ligeramente la convergencia del algoritmo. Como se puede apreciar en la figura 6.13 el incremento del parámetro varianza que controla el operador mejora el comportamiento de este fundamentando el razonamiento anterior. La figura 6.14 indica también que una mayor varianza mejora la diversidad de la población.

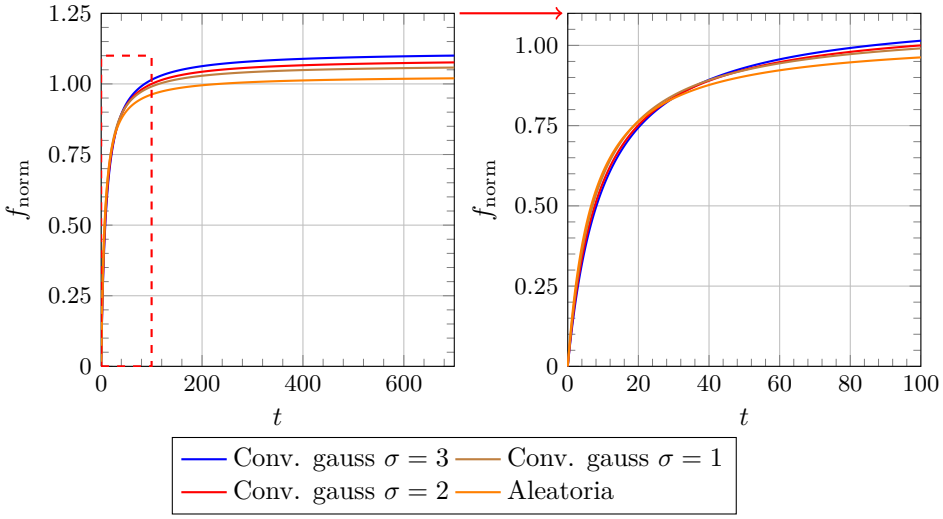


Figura 6.13: Influencia del operador de mutación en la aptitud

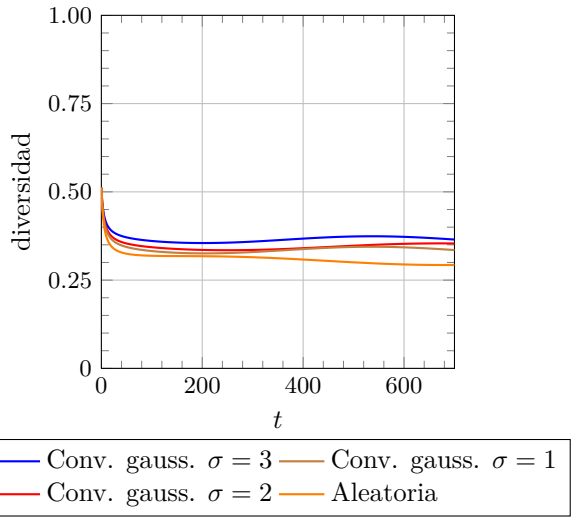


Figura 6.14: Influencia del operador de mutación en la diversidad

6.5 El operador de migración

La migración es uno de los métodos que poseen los Algoritmos Genéticos para mantener la diversidad de sus poblaciones. Las figuras 6.15 y 6.16 muestran el comportamiento del algoritmo sin migración, con una migración del tipo *stepping-stone* y con una migración de todas las poblaciones a la principal, según se describió anteriormente. Como se puede apreciar en la figura 6.15 este operador tiene una importancia decisiva en el rendimiento del algoritmo.

La migración del tipo *stepping-stone* mejora la diversidad del algoritmo durante las primeras generaciones, pero dado la dificultad en la búsqueda de individuos factibles, en el momento que se localiza un individuo factible, este rápidamente copa a todas las poblaciones. Esto se traduce en una rápida pérdida de la diversidad necesaria para continuar con la búsqueda quedando esta únicamente en manos del operador de mutación. La figura 6.16 muestra este efecto, donde se aprecia además que la falta de migración o una migración inadecuada provoca una pérdida continuada de diversidad de la población. Este último fenómeno corrobora el mal funcionamiento del modelo *stepping-stone* cuando el algoritmo inicia su convergencia.

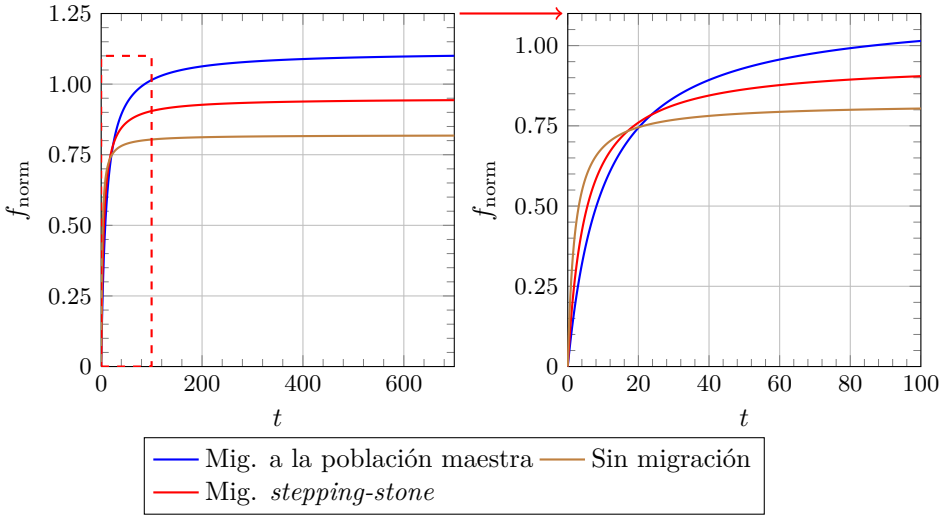


Figura 6.15: Influencia del operador de migración sobre la aptitud

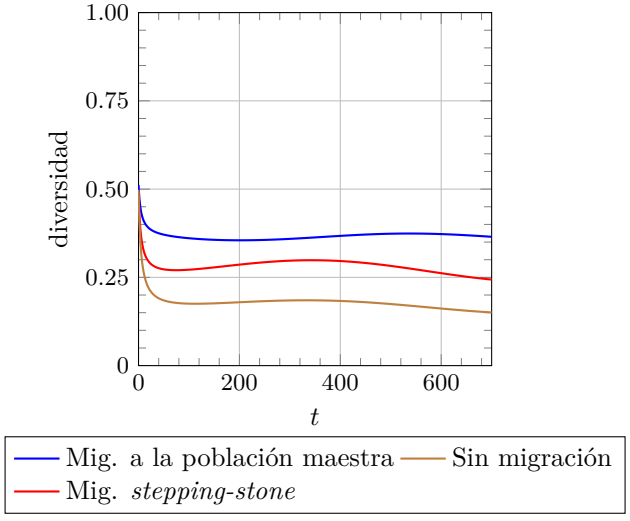


Figura 6.16: Influencia del operador de migración sobre la diversidad

6.6 El operador de renacimiento

La función de este operación es evitar el estancamiento de las poblaciones que alimentan a la *población maestra*, ya que en este caso, estas poblaciones serían una carga computacional innecesaria. La condición de reinicio empleada ha sido la diversidad genotípica población, calculada según la ecuación (4.43), y normalizada mediante la expresión:

$$D_{\text{norm}} = \frac{D}{\hat{\mu}} \tag{6.2}$$

Cuando este valor cae por debajo de 10^{-4} se produce la reinicialización completa de esa población. Esta nueva población vuelve a alimentar a la *población maestra* aunque los individuos generados por la primera estén muy alejados del mejor individuo de la maestra. Sin embargo, esta pequeña aportación de material genético es suficiente para que con el tiempo se produzcan los cruces entre los individuos migrados y la población maestra, enriqueciendo la diversidad de esta última. Las figuras 6.17 y 6.18 muestran la importante aportación de este operador al mantenimiento de la diversidad de la *población maestra*, donde puede apreciarse una convergencia mucho más rápida hacia un óptimo relativo, así como una menor diversidad ante la ausencia de renacimiento.

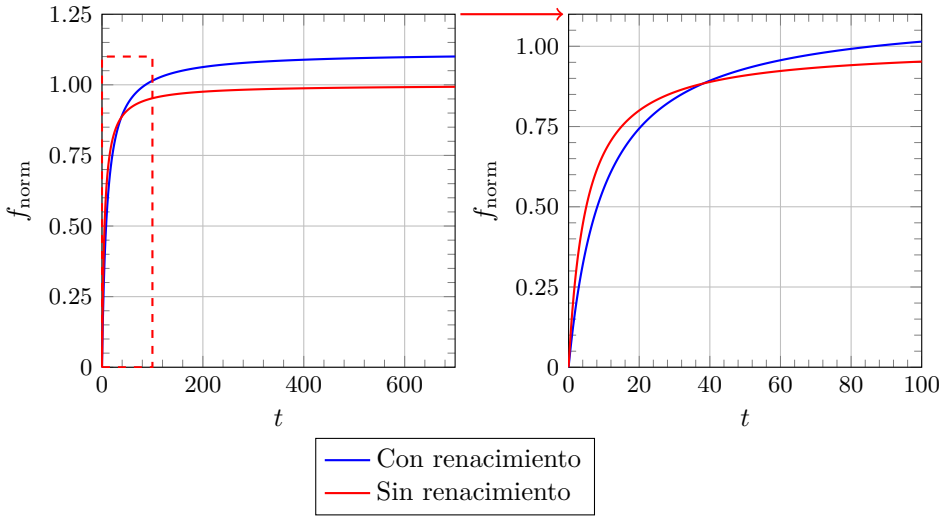


Figura 6.17: Influencia del operador de renacimiento sobre la aptitud

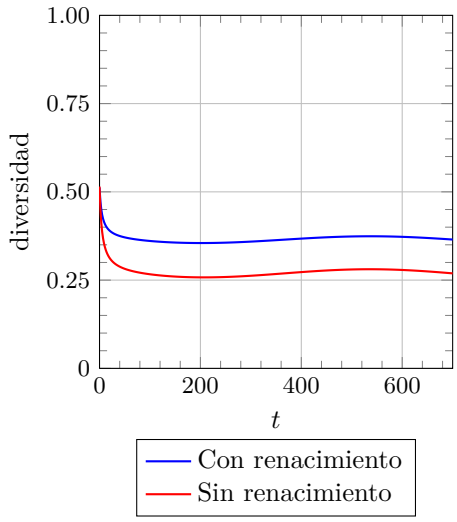


Figura 6.18: Influencia del operador de migración sobre la diversidad

6.7 El operador de remplazo

El operador de remplazo empleado es el de brecha generacional SSGA, analizándose las estrategias de remplazo padre, del menos apto, del más apto y aleatorio.

Las figuras 6.19 y 6.20 muestran el efecto de las diferentes estrategias, de las cuales la que mejor comportamiento ha tenido ha sido la de remplazo del individuo menos apto. El resto de estrategias han funcionado bastante mal, ya que la eliminación de los individuos más aptos, especialmente cuando es difícil encontrar individuos factibles, dificulta la formación de los esquemas necesarios para la convergencia del algoritmo. Por otro lado, la eliminación de los mejores individuos tampoco aporta un incremento en la diversidad de la población que asegure una convergencia futura.

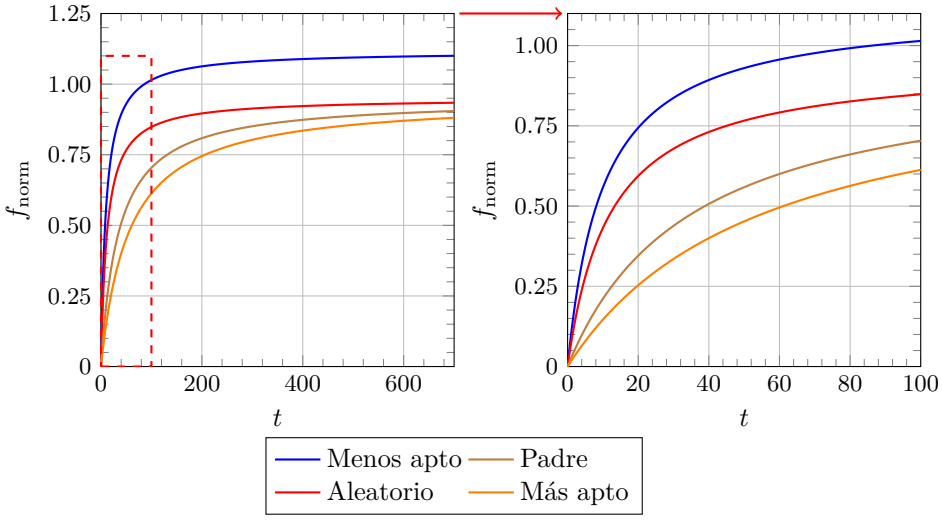


Figura 6.19: Influencia del operador de remplazo sobre la aptitud

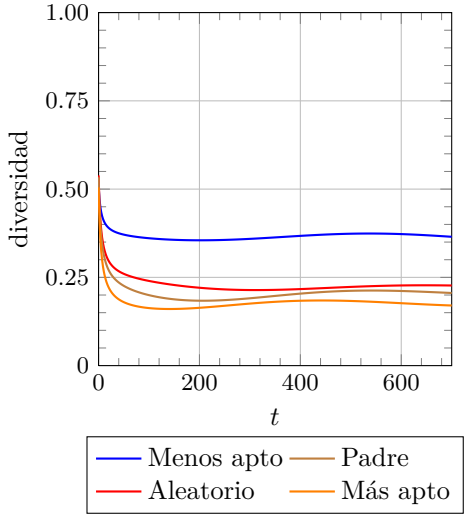


Figura 6.20: Influencia del operador de remplazo sobre la diversidad

6.8 El ajuste de los parámetros del Algoritmo Genético

Los parámetros a determinar en el algoritmo descrito son: el tamaño de la población, la tasas de cruce y mutación. Lamentablemente el ajuste de estos parámetros solo podrán tomarse como referencia para la resolución de este problema sin que tengan que ser necesariamente los óptimos para todos los problemas de optimización estructural, tal y como se vió anteriormente. No obstante, siempre sirven de indicador frente a nuevos problemas.

A continuación se describen los resultados obtenidos para cada parámetro.

6.8.1 El tamaño de la población

Este parámetro es fundamental para la resolución del problema, ya que un tamaño de población excesivamente grande requiere demasiadas evaluaciones de la función objetivo, mientras que un tamaño de población demasiado pequeño no es capaz de almacenar la diversidad genética suficiente y suele provocar una convergencia prematura del algoritmo.

Las tablas 6.1 y 6.2 muestran que los tamaños de población empleados para la resolución de este problema. Estos tamaños oscilan entre los 30 a los 200 individuos, siendo un valor habitual 30-40 individuos.

Las figuras 6.21 y 6.22 muestran como, a partir de 20 individuos, no se produce un incremento en la aptitud resultante, mientras que los valores de diversidad son bastante similares. Por debajo de 20 individuos, la diversidad de la población no es adecuada provocando la convergencia prematura del algoritmo.

Por otro lado, como puede verse en la figura 6.23, el número de evaluaciones hasta alcanzar la convergencia es significativamente menor para un tamaño de población de 20 individuos que para un tamaño de 30 ó 40 por lo que el tamaño de población óptimo será de 20. Resulta pues evidente que el algoritmo desarrollado tiene una influencia directa, gracias al mantenimiento de la diversidad por diversos mecanismos, en la reducción del tamaño de población.

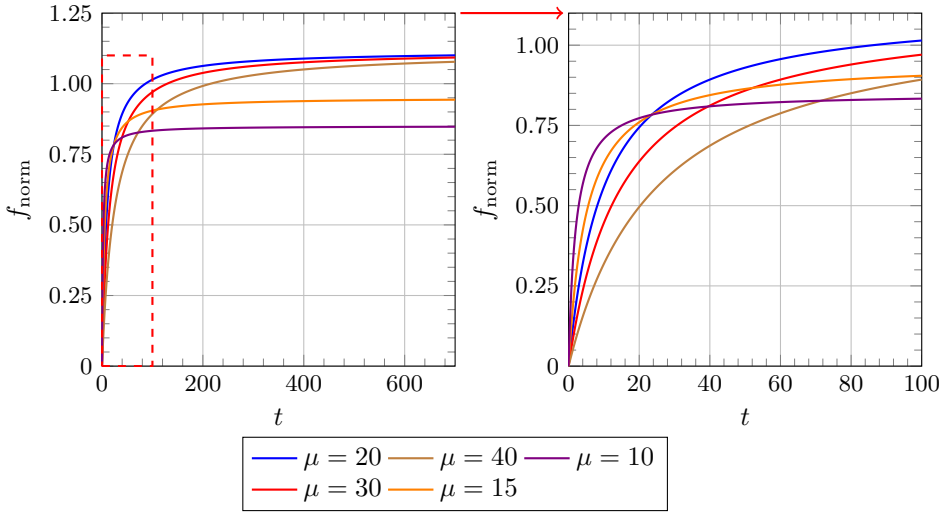


Figura 6.21: Influencia del tamaño de población en la aptitud

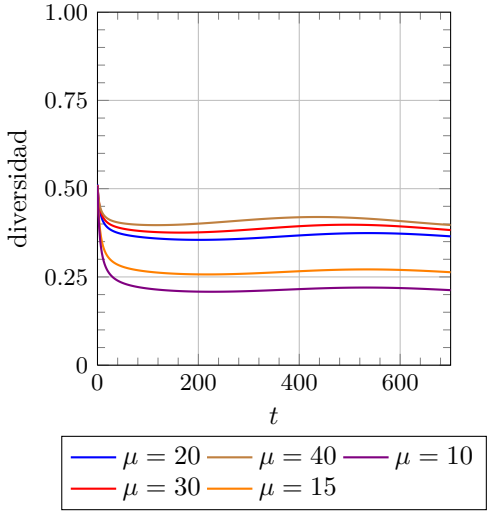


Figura 6.22: Influencia del tamaño de población en la diversidad

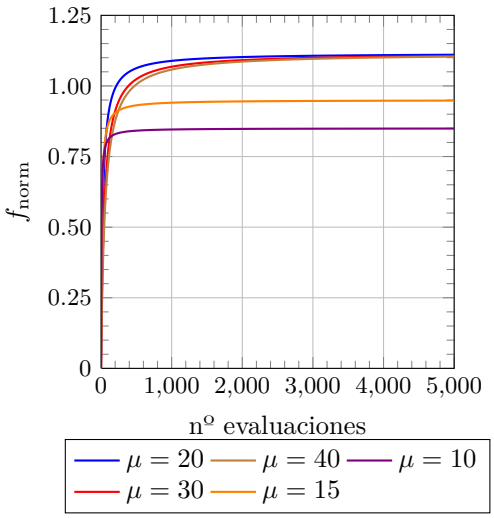


Figura 6.23: Número de evaluaciones de la función de aptitud para diferentes tamaños de población

6.8.2 La probabilidad de cruce

Como se indicó anteriormente se ha incorporado un control fuzzy sobre este parámetro en base a experiencias de trabajos anteriores. Las figuras 6.24 y 6.25 muestran como una probabilidad de cruce demasiado baja produce una convergencia prematura del algoritmo, mientras que una probabilidad demasiado alta provoca lo contrario. Por contra, el control *Fuzzy* presenta un comportamiento mucho mejor ya que permite asignar una probabilidad alta hasta alcanzar la convergencia, mejorando la exploración del algoritmo, y baja posteriormente, permitiendo acelerar el proceso de convergencia.

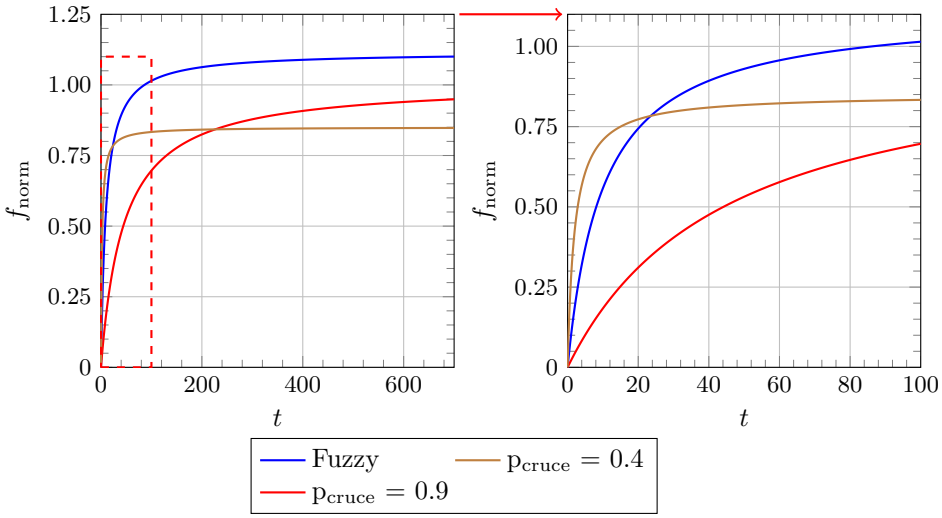


Figura 6.24: Influencia de la probabilidad de cruce en la aptitud

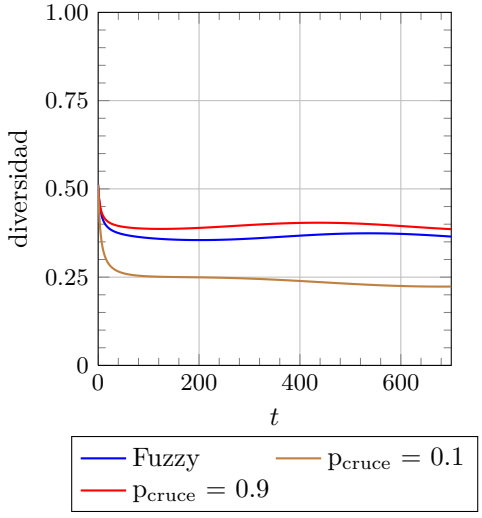


Figura 6.25: Influencia de la probabilidad de cruce en la diversidad

6.8.3 La probabilidad de mutación

Al igual que en el parámetro anterior se ha incorporado un control fuzzy sobre el mismo. Las figuras 6.26 y 6.27 muestran como una probabilidad de mutación demasiado baja no permite una exploración suficiente, alcanzando valores de aptitud deficientes, mientras que una probabilidad demasiado alta provoca una convergencia demasiado lenta al dificultar la formación de esquemas.

Como en en el caso anterior, el control fuzzy presenta un comportamiento mucho mejor ya que permite asignar una probabilidad baja durante las primeras generaciones, acelerando la convergencia del algoritmo, y alta posteriormente, permitiendo mantener la diversidad de la población.

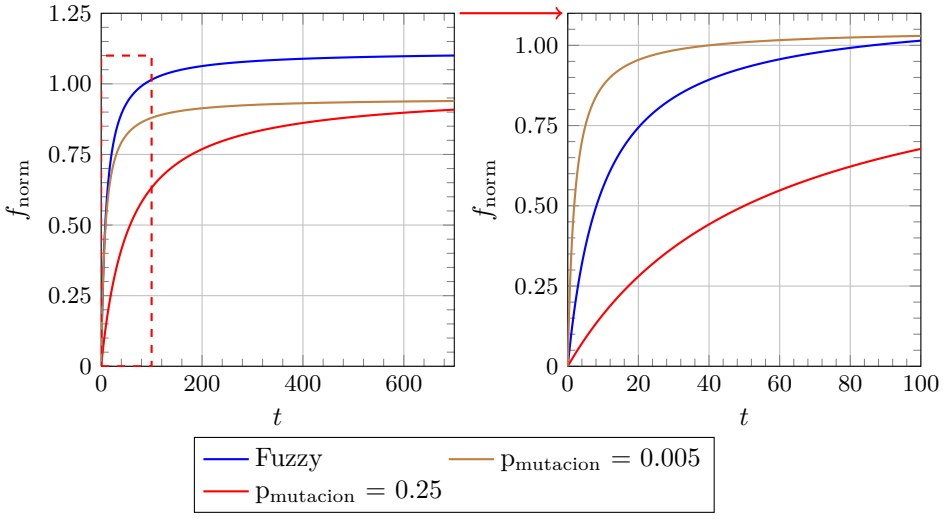


Figura 6.26: Influencia de la probabilidad de mutación en la aptitud

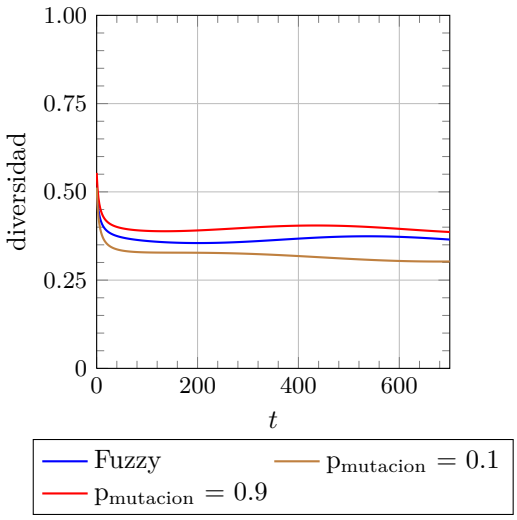


Figura 6.27: Influencia de la probabilidad de mutación en la diversidad

6.9 La función de penalización

Si la definición de los operadores y parámetros anteriores ha sido importante para determinar la eficiencia del algoritmo, la definición de una nueva función de penalización ha sido capital. Ninguna de las funciones de penalización empleadas anteriormente ha podido acercarse, ni de lejos, a los resultados obtenidos mediante la función de penalización propuesta.

En la figura 6.28 se representan los resultados obtenidos para las diferentes funciones de penalización. Las soluciones no factibles, en aquellos casos donde no ha sido posible obtener una solución factible, se representan a trazos. Estas representan la solución con menor grado de violación entre todos los individuos de la población. Como puede observarse, el método de penalización empleado tiene un desempeño muy superior al resto de métodos, permitiendo obtener un nuevo mínimo para el problema analizado.

A continuación se detallan los inconvenientes de los diferentes modelos analizados:

Critica del modelo de Hoffmeister y Sprave El modelo no incorpora ningún factor de ponderación que permita ajustar la función de penalización, lo cual conduce a una situación donde la población entera está compuesta por individuos no

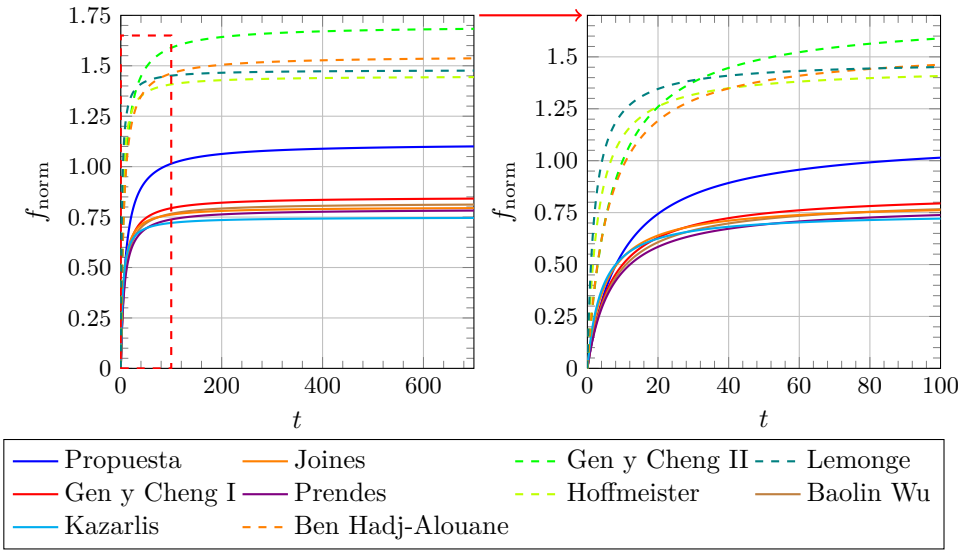


Figura 6.28: Influencia de la función de penalización en la aptitud

factibles y alejados del contorno de soluciones factibles, tal y como ya advertía Michalewicz [264]. Esto debido fundamentalmente a que el problema presenta una solución trivial fuera de dicho contorno.

Critica del modelo de Prendes Este modelo busca penalizar las soluciones sobredimensionadas con el fin de acelerar la convergencia del problema. El algoritmo propuesto por Prendes Gero [306] es muy simple ya que la búsqueda se limita a una lista de perfiles sin alterar tampoco la topología de la estructura. La aplicación de esta función de penalización al presente algoritmo provoca que la población entera se desplace hacia la región de soluciones no factibles, sin posibilidad de localizar ningún óptimo relativo factible.

Critica de los modelos de Joines y Houck y Kazarlis y Petridis El modelo incorpora un factor de penalización C que permite incrementar la penalización de la función, desplazando toda la población hacia la región de soluciones factibles. Además este factor es decreciente con el número de generaciones lo cual permite penalizar menos las soluciones no factibles, una vez la población supuestamente se ha desplazado a la región de soluciones factibles. El inconveniente de esta aproximación reside en el hecho de que una vez el problema ha convergido hacia la región de soluciones factibles, las soluciones no factibles son rápidamente

expulsadas aunque su penalización sea decreciente con el número de generaciones, provocando una convergencia prematura del algoritmo.

Critica del modelo de Ben Hadj-Alouane Este método presenta un exponente cuadrático para la función de penalización. En general todas las funciones de penalización con exponentes superiores a la unidad suelen funcionar mal con el algoritmo desarrollado en este trabajo. Tras estos exponentes subyace la idea de penalizar más aquellos individuos que estén mas alejados, lo cual ocurre durante las primeras generaciones. Pero lo cierto es que transcurridas las primeras generaciones, dado que las restricciones se encuentran normalizadas, los valores de la violación suelen ser inferiores a la unidad provocando precisamente el efecto contrario. Además de este inconveniente, la elección de los parámetros β_1, β_2 no permite ajustar fácilmente el funcionamiento del algoritmo, pasando rápidamente de una convergencia prematura a no tener individuos factibles.

Critica del modelo de Gen y Cheng La primera versión del mismo utiliza una función de penalización multiplicativa normalizada sin factor de ponderación de las restricciones. Al no incorporar un factor de ponderación, presenta el inconveniente de no poder penalizar las restricciones más allá de la unidad por lo que en determinados casos como el estudiado la penalización resulta demasiado baja y no se obtienen soluciones factibles. El segundo modelo tiene un comportamiento aún peor ya que en lugar de normalizar las restricciones, divide el grado de violación de cada restricción por el grado de violación máximo de esa restricción en la población, el cual tiende a ser muy grande cuando existen individuos con una aptitud muy pequeña y un gran grado de violación mantenido en el tiempo. Esto genera un denominador grande que hace que la penalización sea aún menor provocando que el algoritmo se estanque lejos del contorno del espacio de soluciones factibles.

Crítica del modelo de Lemonge y Barbosa Para entender los inconvenientes del método lo reduciremos a una única restricción:

$$k_j = \frac{\sum_{i=1}^{\mu} |f(\boldsymbol{x}_i)|}{\left[\sum_{i=1}^{\mu} \Delta g_l(\boldsymbol{x}_i)\right]^2} \sum_{i=1}^{\mu} \Delta g_j(\boldsymbol{x}_i) = \frac{\sum_{i=1}^{\mu} |f(\boldsymbol{x}_i)|}{\sum_{i=1}^{\mu} \Delta g(\boldsymbol{x}_i)} = \frac{\overline{f}(\boldsymbol{x}}{\Delta \overline{g}(\boldsymbol{x})} \tag{6.3}$$

donde $\overline{f}(\boldsymbol{x})$ es la media de los valores de la función de aptitud y $\Delta\overline{g}(\boldsymbol{x})$ es la media de los grados de violación de los individuos de la población.

Se observa claramente que el factor k_j es inversamente proporcional media de los grados de violación, lo cual provoca que tras aparecer un individuo factible este domine rápidamente la solución al provocar la caída del grado de violación medio. La disminución de este valor provoca que el denominador se aproxime a cero generando por lo tanto un multiplicador infinito que rápidamente elimina las soluciones no factibles haciendo que el problema converja prematuramente. Ya el propio Lemonge y Barbosa [229] advertía que el método no funcionaba de forma satisfactoria para el problema estudiado.

Crítica del modelo de Baolin Wu El principal problema de este modelo es la determinación del significado lingüístico de las variables de entrada y salida. Este presenta el mismo problema para determinar los coeficientes de penalización de los casos anteriores. Además, como las anteriores, tampoco contempla la distancia al contorno de la región de soluciones factibles como método de corregir la penalización. En este caso como en los anteriores el algoritmo bien converge prematuramente o bien no es capaz de generar individuos factibles, sin conseguir aproximarse al óptimo global.

6.10 La solución óptima

Tras ajustar los operadores y parámetros del algoritmo siguiendo el método descrito en los apartados anteriores se ha obtenido una solución óptima mostrada en la figura la figura 6.29. Las coordenadas de los nodos 5,6 respecto del nodo 1 se corresponden la las indicadas para la *solución 1* de la tabla 6.5. El listado de comandos con la definición y el cálculo del mejor individuo se encuentra detallado en apéndice B.

Dado que el método de búsqueda empleado es heurístico no existe la certeza matemática de que se trata de un mínimo absoluto, pero existen una serie de indicadores que apuntan a que, de no serlo, estará muy cerca. Uno de los indicadores es el nivel de tensiones y deformaciones alcanzado. La deformación máxima es de 50,776 mm y los valores de tensión, que se recogen en la tabla 6.4, son bastante elevados lo cual corrobora la afirmación de que la solución se encuentra cerca de la frontera de la región de soluciones factibles. También es indicativo de que nos

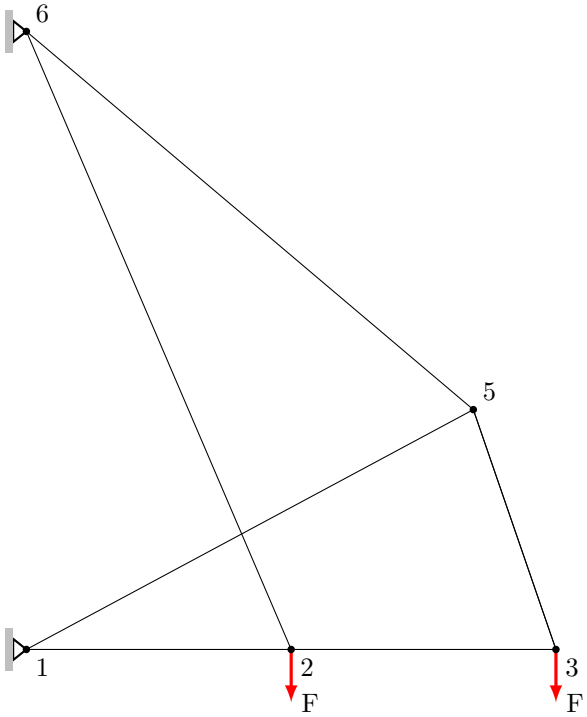


Figura 6.29: Estructura óptima formada por seis barras y cinco nudos

encontramos cerca del óptimo ya que peso y resistencia son dos variables contrapuestas.

Tabla 6.4: Nivel de tensión en los elementos de la solución óptima

Número de elemento	Nudos inicial	Nudos final	Tensión Mpa	Grado de carga
1	1	2	113,33	65,9%
2	2	6	134,10	78,0%
3	1	3	76,70	44,6%
4	3	5	77,89	45,3%
5	1	5	77,96	45,3%
6	5	6	71,15	41,4%

Otro indicador lo constituyen el resto de mínimos relativos obtenidos. Tras la resolución del problema en cinco tentativas independientes se obtiene un conjunto de soluciones con una topología muy similar como se puede apreciar en la tabla 6.5. La conectividad entre elementos es exactamente la misma para todos los individuos,mientras que las coordenadas de los nudos móviles y semifijos están muy próximas entre sí, por lo que todas las soluciones tendrán un aspecto muy similar a la solución final. Las diferencias entre los individuos (soluciones) reside en los genes de los cromosomas de tipo de sección y geometría fundamentalmente. Este último no ha sido recogido en la tabla debido a su gran extensión.

El siguiente indicador es el parecido en la topología obtenida con las soluciones obtenidas previamente. Las topologías de Balling et al. [21] y Ebenau et al. [100], obtenidas mediante diferentes algoritmos, son muy similares a la obtenida mediante este algoritmo como muestra la figura 6.30. A diferencia del modelo de Ebenau, donde solo variaba la coordenada y de los nodos superiores, el presente algoritmo no tiene esta limitación y por lo tanto ha sido capaz de encontrar un óptimo aun mejor. Aquí se puede apreciar claramente como la forma de plantear el problema restringe el proceso de búsqueda impidiendo encontrar mejores soluciones, quedando demostrada la hipótesis inicial que mantenía que el planteamiento sin condicionantes iniciales de forma o geometría, del problema de optimización estructural aportaría necesariamente mejores soluciones que las existentes.

La topología de la estructura resultante, a pesar de utilizar elementos de tipo viga en lugar de barra, respeta los teoremas de Fleron [116], siendo estáticamente determinada y de múltiples soluciones.

Tabla 6.5: Cromosomas de los mejores individuos

Sol.	Coor. nudo semifijo	Coord. nudo libre x	y	z	Conectividad	Tipos de sección	Peso (kN)	Identificador UID
1	21340,9	15435,6	8285,09	0	0 1 1 0 1 0 1 0 1 1	4 8 9 3 9 3 11 7 11 6	10,725	21340-23720-14-15-7895
2	21340,9	15435,6	8285,09	0	0 1 1 0 1 0 1 0 1 1	4 8 9 3 9 3 11 7 11 6	10,725	21340-23720-14-15-8019
3	21341,4	15435,6	8289,31	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 9 11 6	10,726	21341-23724-14-15-9269
4	21341,6	15435,5	8286,52	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 7 11 6	10,737	21341-23721-14-15-9313
5	21341,6	15435,5	8286,52	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 7 11 6	10,737	21341-23721-14-15-9368
6	21338,9	15435,6	8296,19	0	0 1 1 0 1 0 1 0 1 1	4 9 9 11 9 3 11 7 11 6	10,738	21338-23731-14-15-8528
7	21338,9	15435,6	8296,19	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 7 11 6	10,738	21338-23731-14-15-8605
8	21338,9	15435,6	8296,19	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 7 11 6	10,738	21338-23731-14-15-8671
9	21338,9	15435,6	8296,19	0	0 1 1 0 1 0 1 0 1 1	4 8 9 11 9 3 11 7 11 6	10,738	21338-23731-14-15-8842
10	21341,9	15435,5	8383,21	0	0 1 1 0 1 0 1 0 1 1	2 8 9 10 9 2 11 7 11 6	10,745	21341-23818-14-15-9203
11	21348,6	15437,0	8310,21	0	0 1 1 0 1 0 1 0 1 1	4 8 8 11 9 2 11 4 11 6	10,746	21348-23747-14-15-7857
12	21359,6	15436,8	7458,81	0	0 1 1 0 1 0 1 0 1 1	4 8 9 8 9 6 11 6 11 6	10,803	21359-22894-14-15-7739
13	21349,6	15436,4	7440,56	0	0 1 1 0 1 0 1 0 1 1	2 8 9 11 9 8 11 6 11 7	10,861	21349-22876-14-15-8548
14	21349,6	15436,4	7440,56	0	0 1 1 0 1 0 1 0 1 1	2 8 9 11 9 8 11 6 11 6	10,868	21349-22876-14-15-8227
15	21349,6	15436,4	7440,56	0	0 1 1 0 1 0 1 0 1 1	2 8 9 11 9 11 11 5 11 6	10,890	21349-22876-14-15-8389
16	21347,0	15436,4	7441,72	0	0 1 1 0 1 0 1 0 1 1	2 9 9 11 9 7 11 6 11 6	10,936	21347-22877-14-15-8623
17	21399,8	15436,5	7577,48	0	0 1 1 0 1 0 1 0 1 1	4 3 9 11 9 6 11 10 11 6	11,012	21399-23013-14-15-8481
18	21972,3	15436,1	7578,88	0	0 1 1 0 1 0 1 0 1 1	4 3 9 8 9 7 11 3 11 6	11,011	21972-23014-14-15-8285
19	21386,7	15436,3	7394,71	0	0 1 1 0 1 0 1 0 1 1	2 8 9 11 9 8 11 5 11 6	11,017	21386-22830-14-15-7928
20	21379,2	15436,5	7585,37	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 8 11 10 11 5	11,017	21379-23021-14-15-8500
21	21317,9	15437,2	7557,00	0	0 1 1 0 1 0 1 0 1 1	2 9 9 11 9 6 11 5 11 5	11,049	21317-22994-14-15-8037
22	21313,2	15393,1	7581,31	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 8 11 2 11 5	11,061	21313-22974-14-15-8659
23	21311,9	15436,5	7585,32	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 8 11 10 11 6	11,069	21311-23021-14-15-8312
24	21317,9	15472,0	7557,00	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 6 11 5 11 5	11,082	21317-23029-14-15-8385
25	21317,9	15437,2	7557,00	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 2 11 5 11 5	11,094	21317-22994-14-15-7439
26	21306,6	15436,9	7189,02	0	0 1 1 0 1 0 1 0 1 1	2 3 9 11 9 4 11 5 11 5	11,127	21306-22625-14-15-8386

Existe aún otra diferencia entre la solución óptima y las anteriores, y esta radica en el hecho de considerar la flexión de los elementos estructurales. En las soluciones previas existe una barra que conecta los nodos 1-2 y otra que conecta los nodos 2-3. En la solución nueva existe una barra que conecta los nodos 1-2 y otra que conecta los nodos 1-3. Esta diferencia es debida al hecho de que la flexión de las barras produce mayores deformaciones que las consideradas en los modelos más simples. Para evitar que el nudo 2 se deforme excesivamente el algoritmo ha desarrollado dos barras independientes.

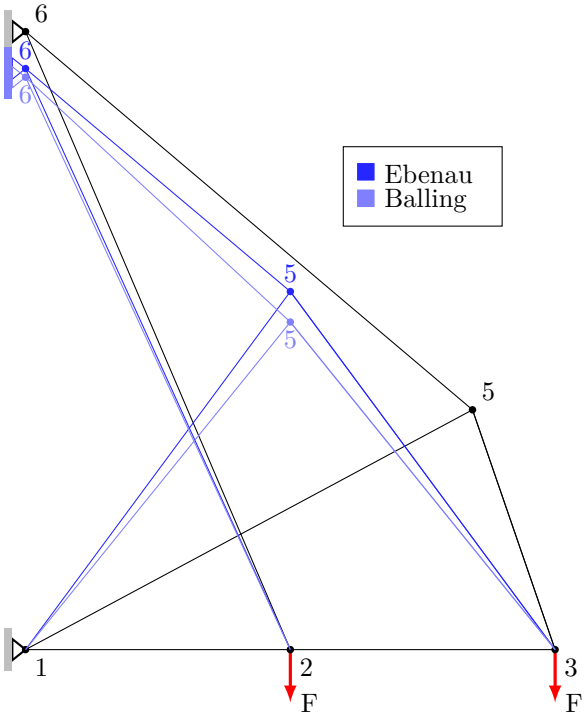


Figura 6.30: Comparativa entre la solución actual y las dos mejores soluciones previas

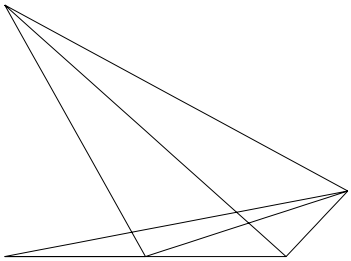
Finalmente el propio proceso evolutivo es un indicador en sí mismo. Las figuras 6.31 a 6.35 representan a los mejores individuos en términos de aptitud durante el proceso evolutivo. Cada vez que aparece un nuevo individuo con una aptitud mejor que el resto, este genoma es anotado con todas sus características en un archivo separado para poder analizarlo posteriormente. En estas figuras se representan de forma consecutiva estos individuos hasta la generación 167. A partir

de aquí se representa un individuo intermedio (figura 6.35(e)) y el mejor (figura 6.35(f)) ya que el resto no aporta ninguna información gráfica.

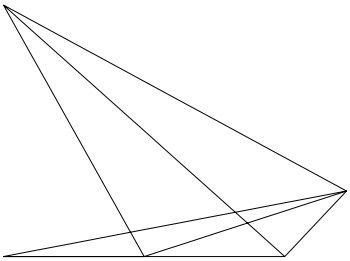
Como se puede observar en la sucesión de figuras, el algoritmo se centra inicialmente en variar la topología del problema, ya que de este modo se produce una mayor disminución en peso de la estructura. Esta exploración de la topología se produce aproximadamente hasta la generación número 100. A partir de la generación número 133 los mejores individuos generados ya son factibles y la topología de los individuos sucesivos apenas varía. Es precisamente en esta generación cuando se supera el mínimo relativo publicado en el trabajo de Ebenau et al. [100]. Este mecanismo coincide con las observaciones realizadas por Lev [230] para estados de carga simples empleando métodos de programación matemática.

La disminución de peso a partir de este punto se logra explorando el tipo de sección y la geometría, extendiéndose durante un periodo de tiempo muy superior, obteniendo un mínimo absoluto en la generación 654. A partir de este punto se produce la convergencia del algoritmo y no fue posible localizar otro óptimo aunque la simulación se extendió durante 1000 generaciones.

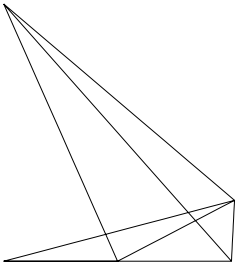
Para finalizar, en base a todo lo descrito, se puede apreciar como el proceso evolutivo no se encuentra sujeto a la arbitrariedad o prejuicios del diseñador. Al no partir de una geometría inicial definida, este proceso evoluciona de forma libre hasta localizar un óptimo, de topología muy similar a la de los diseños anteriores de Ebenau et al. [100] y Balling et al. [21] pero que, al no estar sujeta a restricciones de diseño, logra un carácter mucho más general que estas, pudiendo afirmarse de forma general que esta topología será la óptima para este problema, lográndose el segundo de los objetivos planteados.



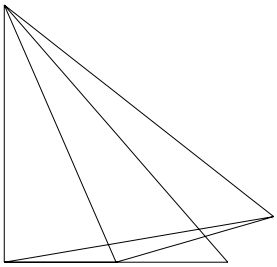
(a) t=0,W=88,210 kN, σ_{max}=46,55 MPa, δ_{max} = 24,5 mm



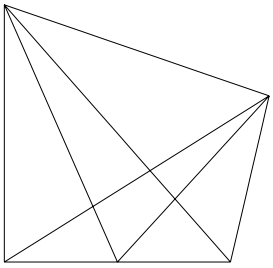
(b) t=2,W=72,68 kN, σ_{max}=150,1 MPa, δ_{max} = 51,6 mm



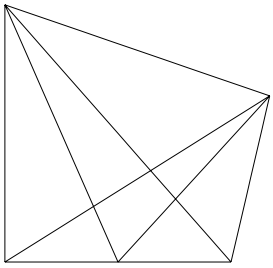
(c) t=5,W=48,616 kN, σ_{max}=39,4 MPa, δ_{max} = 20,2 mm



(d) t=7,W=37,155 kN, σ_{max}=91,9 MPa, δ_{max} = 58 mm

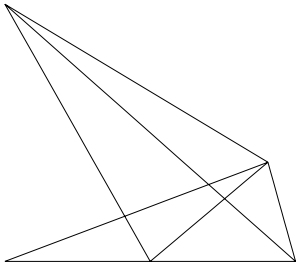


(e) t=9,W=29,951 kN, σ_{max}=78,3 MPa, δ_{max} = 44,1 mm

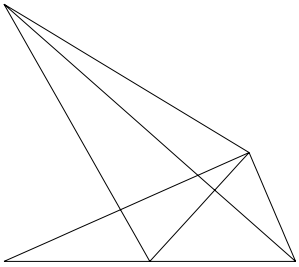


(f) t=11,W=29,618 kN, σ_{max}=75,9 MPa, δ_{max} = 44,93 mm

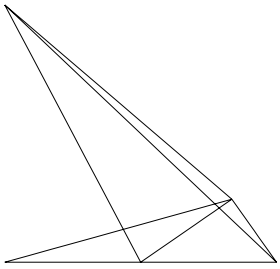
Figura 6.31: Mejores individuos a lo largo de la evolución. Generación 0 a 11



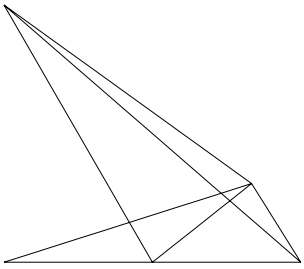
(a) t=13,W=16,665 kN,σ_{max}=94,65 MPa, δ_{max} = 60,163 mm



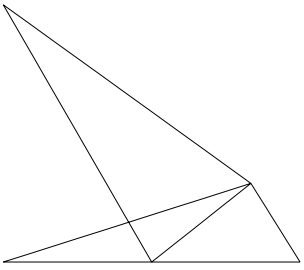
(b) t=14,W=18,107 kN,σ_{max}=94,09 MPa, δ_{max} = 46,31 mm



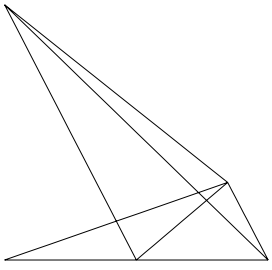
(c) t=16,W=13,833 kN,σ_{max}=142,40 MPa, δ_{max} = 64,57 mm



(d) t=21,W=16,594 kN,σ_{max}=127,15 MPa, δ_{max} = 53,16 mm

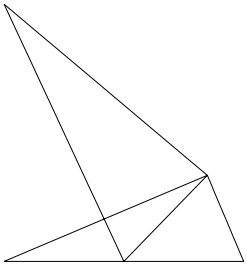


(e) t=22,W=12,113 kN,σ_{max}=107,19 MPa, δ_{max} = 69,20 mm

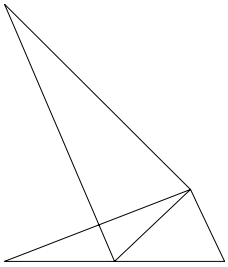


(f) t=23,W=15,157 kN,σ_{max}=210,47 MPa, δ_{max} = 55,23 mm

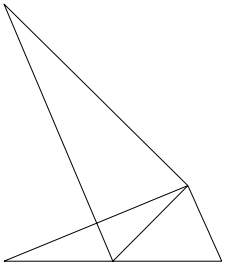
Figura 6.32: Mejores individuos a lo largo de la evolución. Generación 13 a 23



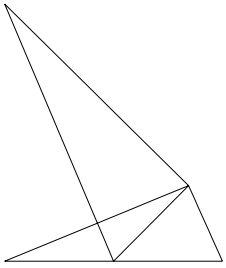
(a) t=24,W=12,203 kN,σ_{max}=98,7 MPa, δ_{max} = 59,58 mm



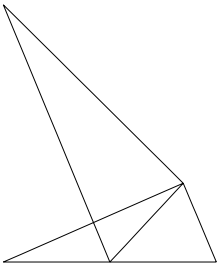
(b) t=35,W=10,441 kN,σ_{max}=126,4 MPa, δ_{max} = 66,77 mm



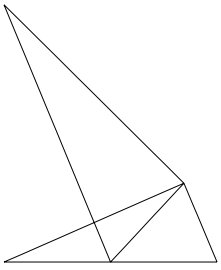
(c) t=38,W=10,042 kN,σ_{max}=218,0 MPa, δ_{max} = 65,69 mm



(d) t=40,W=10,513 kN,σ_{max}=215,9 MPa, δ_{max} = 62,47 mm

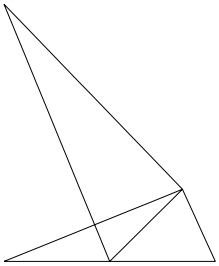


(e) t=42,W=10,712 kN,σ_{max}=218,0 MPa, δ_{max} = 60,15 mm

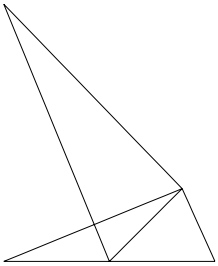


(f) t=43,W=10,426 kN,σ_{max}=223,6 MPa, δ_{max} = 60,2 mm

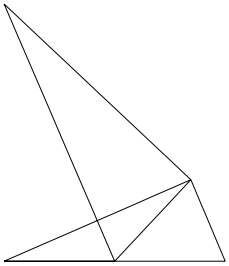
Figura 6.33: Mejores individuos a lo largo de la evolución. Generación 24 a 43



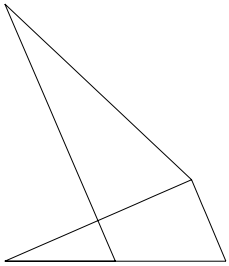
(a) t=63,W=12,092 kN,σ_{max}=135,3 MPa, δ_{max} = 51,7 mm



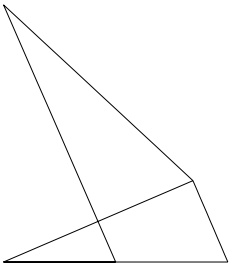
(b) t=74,W=12,135 kN,σ_{max}=135,5 MPa, δ_{max} = 51,27 mm



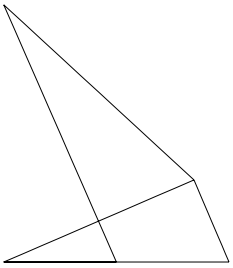
(c) t=87,W=11,763 kN,σ_{max}=122,1 MPa, δ_{max} = 52,56 mm



(d) t=100,W=11,104 kN,σ_{max}=136,6 MPa, δ_{max} = 52,64 mm

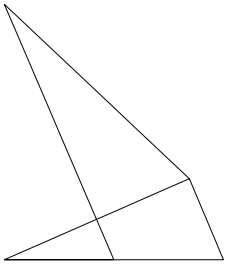


(e) t=119,W=11,600 kN,σ_{max}=136,9 MPa, δ_{max} = 51,18 mm

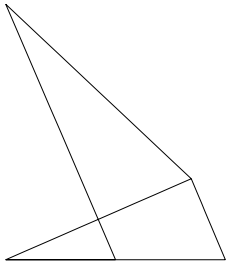


(f) t=121,W=11,236 kN,σ_{max}=218,0 MPa, δ_{max} = 51,20 mm

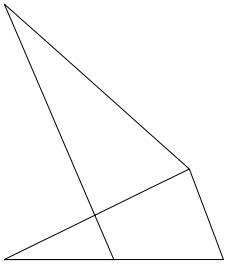
Figura 6.34: Mejores individuos a lo largo de la evolución. Generación 63 a 121



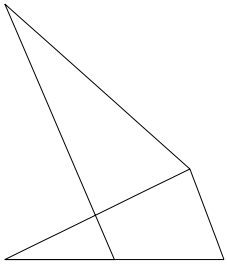
(a) t=133,W=11,756 kN,σ_{max}=131,3 MPa, δ_{max} = 49,94 mm



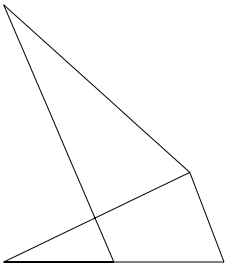
(b) t=135,W=11,580 kN,σ_{max}=139,6 MPa, δ_{max} = 50,51 mm



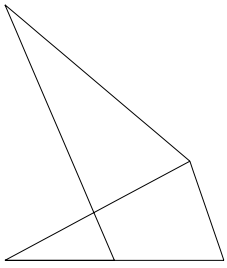
(c) t=149,W=11,251 kN,σ_{max}=131,3 MPa, δ_{max} = 49,93 mm



(d) t=167,W=11,094 kN,σ_{max}=131,3 MPa, δ_{max} = 50,64 mm



(e) t=404,W=10,865 kN,σ_{max}=133,6 MPa, δ_{max} = 50,76 mm



(f) t=654,W=10,725 kN,σ_{max}=134,1 MPa, δ_{max} = 50,78 mm

Figura 6.35: Mejores individuos a lo largo de la evolución. Generación 133 a 654

7

Conclusiones

En base al objetivo principal planteado y a la vista de los resultados obtenidos, ha sido posible probar la hipótesis planteada, demostrando por lo tanto que la definición de una estructura previa condiciona la eficiencia del algoritmo en la búsqueda del óptimo global.

Respecto al resto de objetivos planteados, también pueden establecerse las siguientes conclusiones:

1. Dado que se han seguido las indicaciones proporcionadas por Rajan [310] obteniéndose los mejores resultados hasta la fecha para la estructura de diez barras y seis nodos, se puede afirmar por lo tanto que tales indicaciones, entre las cuales se encuentra la hipótesis del presente trabajo, eran correctas.

2. El algoritmo no proporciona una única solución, sino un conjunto de soluciones, de topología similar y estáticamente determinadas, en consonancia con los teoremas de Fleron [116], aun tratándose de elementos de tipo viga.

3. La comparación de la topología resultante con las existentes permite establecer que restricciones de diseño le han impedido localizar un óptimo mejor. En los trabajos de Balling et al. [21] and Ebenau et al. [100] las restricciones de diseño impidieron el movimiento lateral del nudo 5.

4. A diferencia del resto de algoritmos, el algoritmo desarrollado permite obtener información del propio proceso evolutivo. Esto permite al diseñador conocer cual de los dos, la topología o la geometría y tamaño de las secciones, tiene mayor influencia en la reducción de peso. En el caso estudiado ha sido la topología, lo cual concuerda con los resultados obtenidos previamente por Lev Lev [232] y Rajan [310] donde al permitir a sus algoritmos variar la topología de sus estructuras, consiguieron reducir sensiblemente el peso de las mismas.

5. Dado que, en diferentes ejecuciones, se ha obtenido una topología muy similar a las existentes, sin partir de ningún tipo de geometría inicial, se puede generalizar que esta topología debe ser la óptima para este problema.

6. Al no necesitar de un diseño preliminar de la estructura, no se requiere de ningún tipo de habilidad o conocimiento especial por parte del diseñador, a parte de los necesarios para poder interpretar y hacer uso de los resultados.

Además, el presente trabajo ha permitido conocer los diferentes factores que intervienen en la eficiencia de los Algoritmos Genéticos aplicados a la optimización de estructuras, y que aparecen descritos a continuación:

1. El empleo de una codificación mixta provee de un desempeño muy superior del algoritmo al posibilitar una definición diferenciada de los operadores genéticos, permitiéndole adaptarse de mejor forma al problema de optimización de estructuras. Las codificaciones continuas evitan los problemas derivados de la discretización similares a los descritos por Dorn et al. [97].

2. El operador de inicialización no debe ser un simple generador de números aleatorios, sino que debe incorporar una comprobación de validez de los individuos generados así como un requerimiento de distancia mínima al resto de individuos de la población.

3. El empleo de una codificación mixta permite adaptar el operador de cruce adecuado para cada cromosoma, permitiendo de este modo realizar un cruce por cromosoma. Esto facilita una construcción de esquemas de forma mas rápida y eficiente mejorando sensiblemente la eficiencia del algoritmo, de forma similar al cruce por fenotipo descrito por Prendes Gero et al. [307]. Los operadores de cruce óptimos, para la estructura de diez barras y seis nudos, han sido: Cruce binario uniforme para el cromosoma de conectividad, cruce entero uniforme para los cromosomas de material y tipo de sección, cruce SBX-β para los cromosomas de nudos móviles y semifijos y cruce SBX-β truncado para el cromosoma de geometría. Este resultado coincide con un trabajo previo publicado por [89] quien también evaluó la estructura de diez barras y cuyos resultados aparecen en la tabla 6.1.

4. El empleo de operadores genéticos avanzados, provenientes de la teoría básica de los Algoritmos Genéticos ha permitido al presente trabajo dar un salto cualitativo y cuantitativo respecto a los trabajos anteriores.

5. El operador de mutación óptimo es el de convolución gaussiana con $\sigma = 3$, adaptado a cada codificación, debido a su mayor capacidad explorativa.

6. La incorporación de un operador de migración hacia la *población maestra* mejora sensiblemente la diversidad de la población, lo cual influye directamente en la eficiencia.

7. La combinación de los operador de migración y renacimiento permiten dotar a la población de la diversidad necesaria para explorar otras zonas del espacio de soluciones incluso cuando la convergencia del algoritmo es elevada, sin perjuicio en la eficiencia, del modo descrito por Rajan [310], mejorando por lo tanto de manera sensible en la eficiencia del mismo.

8. La definición del algoritmo tiene una influencia directa en el mantenimiento de la diversidad de la población, y por lo tanto en el tamaño de esta, requiriéndose un tamaño sensiblemente menor a los valores publicados hasta la fecha.

9. El ajuste de las probabilidades de cruce y mutación en base a los resultados de Herrera y Lozano [171] ha sido satisfactorio. El rendimiento del algoritmo ha sido sensiblemente superior que el obtenido mediante valores predefinidos.

10. Ninguna de las funciones de penalización publicadas hasta la fecha funcionan de modo correcto con el algoritmo descrito. Esto es debido al planteamiento abierto del mismo, donde el espacio de búsqueda es muy superior a los algoritmos empleados hasta la fecha. Debido a esto se desarrolla una nueva función de penalización basada en el concepto de *umbral de penalización* que permite explorar la región de soluciones no factibles sin caer en la solución trivial.

Las principales aportaciones del presente trabajo son:

1. El planteamiento del problema de optimización de estructuras sin otras restricciones de diseño que la definición de los puntos de aplicación de carga y reacción así como las limitaciones de tensión y deformación necesarias para garantizar la seguridad de las mismas. Esto supone que no se requiere ningún tipo de geometría inicial en el proceso de búsqueda, lo cual no se había logrado hasta el momento.

2. La generalización de las topologías resultantes en virtud del propio proceso de búsqueda.

3. El empleo de elementos estructurales de tipo viga, en lugar de barra, incorporando la geometría de la sección y sus propias características geométricas en la codificación.

4. La aplicación y evaluación rigurosa de los diferentes operadores genéticos desarrollados en la actualidad. Desafortunadamente la mayoría de los trabajos publicados no se fundamentan en la teoría general de los Algoritmos Genéticos y emplean operadores genéticos y codificaciones en general no apropiados al tipo de problema a resolver.

5. El desarrollo de un sistema de codificación mixto, adaptado a los problemas de optimización de estructuras, con un desempeño superior a los existentes.

6. La definición de un nuevo operador de migración combinado con uno de renacimiento completo.

7. La definición de una nueva función de penalización aplicable a los problemas de optimización de estructuras.

8. La obtención de información del propio proceso evolutivo. Al no partir de una forma predefinida, este nos indica como se puede lograr una mayor reducción de peso en la estructura.

9. La obtención de un nuevo mínimo, un 10,92% inferior a la existente establecida hace seis años, para el problema clásico de optimización de estructuras de diez barras y seis nodos.

Concluding remarks

Based on the main objective and the results, it has been possible to test the main hypothesis, showing therefore, that the definition of a prior structure affects the efficiency of the algorithm in finding the global optimum.

The following conclusions can be also stated:

1. Since the guidelines given by Rajan [310] were followed, obtaining the best results to date for the ten-bar and six nodes benchmark problem, it can be stated that such guidelines were correct.

2. The algorithm does not provide a single solution but a set of solutions, with similar topology and statically determinate, which is consistent with the theorems of Fleron [116], even dealing with beam-type elements.

3. Comparing the final topology-set with the previously reported ones, allows to state that design constraints prevented the previous algorithms from locating a better optimum solution. In Balling et al. [21] and Ebenau et al. [100] reports, the design constraint which stopped the improvement was the lateral constraint at node 5.

4. Unlike other algorithms, the algorithm developed allows to obtain information from the evolutionary process itself. This allows the designer to know which of both, the topology or section geometry and size, has greater influence on reducing weight. For the benchmark problem, topology had greater influence, which agrees with the previously obtained results by Lev [232] and Rajan [310] where a significantly weight reduction was achieved by allowing its algorithm to modify the structure topology.

5. Since without defining any previous structure, in different runs, a very similar topology has been obtained, it can be stated that this must be the optimal one for this problem.

6. By not requiring a preliminary design of the structure, special skill or knowledge is not required by the designer, apart from the one needed to interpret and make use of the results.

Furthermore, this work has allowed to know the different factors affecting the efficiency of GA applied to structural optimization, which are described below:

1. Mixed encoding allows to enhance the algorithm performance by a differentiated definition of genetic operators, enabling a better adaptation to the structural optimization problem. Continuous encoding avoid discretization problems similar to those described by Dorn et al. [97].

2. The initialization operator should not be a simple random number generator, but must include a check of legality of the generated individuals, as well as a minimum distance requirement to other individuals in the population.

3. Mixed coding allows to adapt the crossover operator suitable for each chromosome, thus allowing a chromosome crossover. This makes scheme's formation easier in a faster and more efficient way enhancing, to a substantial extent, the algorithm efficiency, in a similar way to the phenotype crossover described by Prendes Gero et al. [307]. Optimal crossover operators, for the benchmark structure, were: uniform crossover for the binary connectivity chromosome, uniform integer crossover for material section type chromosomes, SBX-β crossover for mobile and semi-mobile joint's chromosome and truncated SBX-β for geometry chromosome. This result agrees with the previous work reported by [89] who also used the benchmark structure (see la tabla 6.1).

4. The use of advanced genetic operators, from the basic GA theory, enabled this work to achieve a qualitative and quantitative improvement over previous work.

5. The optimum mutation operator was the $\sigma = 3$ Gaussian convolution operator, adapted to each chromosome, owed to its higher explorative capacities.

6. Population diversity was highly enhanced by a migration operator where best individuals were migrated to the *master population*, which is closely linked with the algorithm efficiency.

7. The combination of migration and rebirth operators allow the population to provide the diversity needed to explore other areas of the solution space even when the algorithm convergence is high, without prejudice to the efficiency, as described by Rajan [310], thus enhancing the efficiency sensitively.

8. The algorithm definition has a direct influence in maintaining the diversity of the population, which is closely linked with the population size, requiring a significantly smaller size than previously reported values.

9. Crossover and mutation probabilities adjustment based on the results of Herrera y Lozano [171] were pleasing. The algorithm performance was significantly higher than that obtained with predefined values.

10. None of the penalty functions reported to date worked properly. This is due to the algorithm definition, where the search space is far higher than the previous ones. So a new penalty function based on the concept of *penalty threshold* was developed, in order to explore the non-feasible region of the solution's space without falling into the trivial solution.

The main contributions of this work are:

1. The approach of the structural optimization problem with no other design constraints than load or support keypoints, stress and displacement constraints to ensure the safety of the structure. This means that it does not require a predefined geometry for the search process, which had not been achieved so far.

2. The generalization of the topologies obtained under the search process itself.

3. The use of beam-like structural elements, rather than truss, and the incorporation of the section geometry and its own intrinsic characteristics in the coding.

4. The application and rigorous evaluation of the different genetic operators developed to date. Unfortunately most of the published works are not based on the general theory of GA and use genetic operators and encodings which generally are not suitable.

5. The development of a mixed coding system adapted to structural optimization problems, with enhanced performance to existing ones.

6. The definition of a new operator of migration combined with a full rebirth operator.

7. A new penalty scheme for all structural optimization problems.

8. Obtaining information from the evolutionary process itself allows to know how a greater weight loss can be achieved.

9. A new global minimum, at least 10,92% lower than the existing one established six years ago, for the classic problem of structural optimization of ten bars and six nodes.

8

Futuras líneas de investigación

Los resultados del presente trabajo han sido plenamente satisfactorios al demostrar que es posible trabajar con problemas de optimización de estructuras con planteamientos abiertos. No obstante, se abren nuevas vías de trabajo y futuras líneas de investigación de entre las que merece la pena destacar las siguientes:

1. Incorporar el estudio de varios estados de carga.
2. Incorporar otros tipos de carga, además de las puntuales.
3. Incorporar otros tipos de restricciones como las de simetría.
4. Incorporar un motor de cálculo FEM propio.
5. Incorporar técnicas de predicción de la aptitud que alivien el coste computacional. Se han planteado algunas como los autómatas celulares [55] o las redes neuronales [373] que se encuentran a falta de desarrollar de modo práctico.
6. Incorporar un operador de inicialización basado en PSO, de modo que cada individuo de la población inicial proceda de optimizaciones previas realizadas mediante PSO de rápida convergencia, sobre individuos alejados entre sí en el espacio de diseño.
7. Incorporar una interfaz gráfica que facilite el manejo del programa.
8. Adaptar el algoritmo a contornos diferentes del espacio cúbico de soluciones, ya que esta restricción de diseño puede ser necesaria para resolver ciertos problemas.
9. Añadir la posibilidad restringir la geometría de las secciones a perfiles normalizados.
10. Incorporar otro tipo de restricciones de diseño como las asociadas a vibraciones, euronormas, etc ...
11. Incorporar restricciones que tengan en cuenta aspectos económicos y constructivos.
12. Evaluar de la función de penalización con otros problemas de optimización, diferentes a la estructural, donde el óptimo se encuentre localizado en la frontera de la región de validez.

13. Incorporar un μGA que explore los valores óptimos de la geometría de las secciones definidas para cada individuo ya que una vez se ha alcanzado la topología óptima, el rendimiento del algoritmo disminuye sensiblemente.

14. Aplicar el algoritmo a problemas reales de optimización de estructuras como es la construcción de bastidores de semiremolques o estructuras de obra civil.

15. Adecuar el algoritmo al diseño de elementos de máquinas como el diseño de engranajes, considerando estos elementos dentro del conjunto de la máquina, lo cual permitirá evaluar la interacción con el resto de elementos de la misma. Se pretende desarrollar esta aplicación en colaboración con la empresa KISSSoft Hirnware.

16. Aplicar el algoritmo al calibrado de cámaras. Esta línea se encuentra en estado de desarrollo incipiente y forma parte de un proyecto más ambicioso que se está llevando a cabo en el Instituto de Diseño y Fabricación de la UPV, cuyo objetivo es la detección de defectos en las carrocerías de vehículos.

Future research

The results of this work have been fully pleasant to demonstrate that it is possible to work with structural optimization problems without a predefined geometry. However, it opens new research lines. Among them, it is worth remarking the following ones:

1. Integrate multiple loads.
2. Integrate other loads, besides point loads.
3. Integrate other constraints, like symmetry ones.
4. Integrate an own FEM calculation engine.
5. Incorporate fitness prediction techniques to reduce the number of function calls. Some of them were previously reported like cellular automata [55] or neural networks [373] but they fail to develop in a practical way.
6. Incorporate a PSO-based initialization operator, so that each individual in the initial population comes from previous optimizations performed by *fast convergence PSO* on individuals far apart in the design space.
7. Incorporate a graphical interface in order to make the work easier for the user.
8. Adapt the algorithm to other contours different from cubic space of solutions, since this design constraint may be necessary to solve certain problems.
9. Add the possibility of using normalized section profiles.
10. Incorporate other design constraints such as those associated with vibrations, regulations, etc...
11. Incorporate other design constraint to take into account economic and constructive aspects.
12. Assess the penalty function with another kind of optimization problems where the optimum is located on the border of the feasible region.
13. Include a μGA to explore the geometry optimum values for each individual because, once the optimal topology has been reached, the algorithm performance decreases significantly.

14. Apply the algorithm to real structural optimization problems such as trailer construction.

15. Adapt the algorithm to the design of machine elements such as gear design, considering these elements in a whole way, evaluating the interaction between elements. It is intended to develop this application in collaboration with the company KISSsoft Hirnware.

16. Apply the algorithm to camera calibration. This research line is in an incipient stage of development and is part of a larger project being carried out at *Instituto de Diseño y Fabricación*, belonging to UPV, aimed at defect's detection in vehicle bodies.

Bibliografía

[1] W. Achtziger. Truss topology optimization including bar properties different for tension and compression. *Structural Optimization*, 12(1):63–74, 1996. ISSN 09344373.

[2] W. Achtziger. Local stability of trusses in the context of topology optimization. part i: Exact modelling. *Structural Optimization*, 17(4):235–246, 1999. ISSN 09344373. doi: 10.1007/s001580050056.

[3] W. Achtziger. Local stability of trusses in the context of topology optimazation. part ii: A numerical approach. *Structural Optimization*, 17(4):247–258, 1999. ISSN 09344373.

[4] W. Achtziger y M. Stolpe. Truss topology optimization with discrete design variables-guaranteed global optimality and benchmark examples. *Structural and Multidisciplinary Optimization*, 34(1):1–20, 2007. ISSN 1615147X. doi: 10.1007/s00158-006-0074-2.

[5] W. Achtziger, M. P. Bendsøe, A. Ben-Tal, y J. Zowe. Equivalent displacement based formulations for maximum strength truss topology design. *IMPACT of Computing in Science and Engineering*, 4(4):315–345, 1992. ISSN 08998248.

[6] H. Adeli y N.-T. Cheng. Integrated genetic algorithm for optimization of space structures. *Journal of Aerospace Engineering*, 6(4):315–328–, 1993. doi: 10.1061/(ASCE)0893-1321(1993)6:4(315).

[7] H. Adeli y N.-T. Cheng. Augmented lagrangian genetic algorithm for structural optimization. *Journal of Aerospace Engineering*, 7(1):104–118–, 1994. doi: 10.1061/(ASCE)0893-1321(1994)7:1(104).

[8] H. Adeli y O. Kamal. Efficient optimization of plane trusses. *Advances in Engineering Software and Workstations*, 13(3):116–122, 1991.

[9] S. Ai y Y. Wang. Application of improved genetic algorithms in structural optimization design. In M. Zhu, editor, *Information and Management Engineering*, volume 236 of *Communications in Computer and Information Science*, pages 480–487. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24097-3. doi: 10.1007/978-3-642-24097-3_72.

[10] S. Aine, R. Kumar, y P. Chakrabarti. Adaptive parameter control of evolutionary algorithms to improve quality-time trade-off. *Applied Soft Computing*, 9(2):527 – 540, 2009. ISSN 1568-4946. doi: 10.1016/j.asoc.2008.07.001.

[11] N. Ali, K. Behdinan, y Z. Fawaz. Applicability and viability of a ga based finite element analysis architecture for structural design optimization. *Computers & Structures*, 81(22–23):2259 – 2271, 2003. ISSN 0045-7949. doi: 10.1016/S0045-7949(03)00255-4.

[12] W. Annicchiarico y M. Cerrolaza. An evolutionary approach for the shape optimization of general boundary elements models. *Area*, 266(2):251–266, 2002.

[13] N. Anonymous. *Narcotics Anonymous.* World Service Office, Van Nuys, CA, U.S.A., 1988.

[14] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In *Proceedings of the third international conference on Genetic algorithms*, pages 86–91, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3. doi: http://dl.acm.org/citation.cfm?id=93126.93156.

[15] J. S. Arora. *Optimization of structural and mechanical systems.* World Scientific, Hackensack, NJ, 2007.

[16] I. Azid, A. Kwan, y K. Seetharamu. An evolutionary approach for layout optimization of a three-dimensional truss. *Structural and Multidisciplinary Optimization*, 24:333–337, 2002. ISSN 1615-147X. 10.1007/s00158-002-0244-9.

[17] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages

101–111, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc. ISBN 0-8058-0426-9.

[18] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc. ISBN 0-8058-0158-8.

[19] N. R. Ball, P. M. Sargent, y D. O. Ige. Genetic algorithm representations for laminate layups. *Artificial Intelligence in Engineering*, pages 99–108, 1993.

[20] R. J. Balling. Optimal steel frame design by simulated annealing. *Journal of structural engineering New York, N.Y.*, 117(6):1780–1795, 1991. ISSN 07339445.

[21] R. J. Balling, R. R. Briggs, y K. Gillman. Multiple optimum size/shape/-topology designs for skeletal structures using a genetic algorithm. *Journal of Structural Engineering*, 132(7):1158–1165–, 2006. doi: 10.1061/(ASCE)0733-9445(2006)132:7(1158).

[22] H. Barbosa y A. Lemonge. A new adaptive penalty scheme for genetic algorithms. *Information Sciences*, 156(3-4):215–251, 2003. ISSN 00200255. doi: 10.1016/S0020-0255(03)00177-4.

[23] M. Barnes, B. Topping, y D. Wakefield. Aspects of form finding by dynamic relaxation. *International Conference on Slender Structures*, 1977.

[24] N. A. Barricelli. *Symbiogenetic evolution processes realized by artificial methods.* S.l., 1957.

[25] J. Barta. On the minimum weight of certain redundant structures. *Acta Tech. Acad. Sci. Hung.*, 18:67–76, 1957.

[26] M. S. Bazaraa, J. J. Jarvis, y H. D. Sherali. *Linear programming and network flows (2nd ed.).* John Wiley & Sons, Inc., New York, NY, USA, 1990. ISBN 0-471-63681-9.

[27] T. Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer y B. Manderick, editors, *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature*. North-Holland, Amsterdam, 1992.

[28] T. Bäck. Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, pages 263–271. MIT Press, 1992.

[29] T. Bäck. Optimal mutation rates in genetic search. In *Proceedings of the fifth International Conference on Genetic Algorithms*, pages 2–8. Morgan Kaufmann, 1993.

[30] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *In Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62. IEEE Press, 1994.

[31] T. Bäck. Generalized convergence models for tournament- and (mu, lambda)-selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 2–8, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-370-0.

[32] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

[33] T. Bäck y F. Hoffmeister. Extended selection mechanisms in genetic algorithms. pages 92–99. Morgan Kaufmann, 1991.

[34] T. Bäck y S. Khuri. An evolutionary heuristic for the maximum independent set problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 531–535. IEEE Press, 1994.

[35] T. Bäck, D. B. Fogel, y Z. Michalewicz. *Handbook of evolutionary computation*. Institute of Physics Pub. ; Oxford University Press, Bristol; Philadelphia; New York, 1997.

[36] J. Bean y A. Ben Hadj-Alouane. A dual genetic algorithm for bounded integer programs. *Technical Report 9253*, 1992.

[37] A. Belloli y P. Ermanni. Optimum placement of piezoelectric ceramic modules for vibration suppression of highly constrained structures. *Smart Materials and Structures*, 16(5):1662, 2007.

[38] S. V. Belur. *CORE: Constrained Optimization by Random Evolution*, pages 280–286. Stanford Bookstore, 1997.

[39] A. Ben Hadj-Alouane y J. Bean. A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45(1):92–101, 1997.

[40] A. Ben-Tal y G. Roth. A truncated log barrier algorithm for large-scale convex programming and minmax problems: Implementation and computational results. *Optimization Methods and Software*, 6(4):283–312, 1996. ISSN 10556788.

[41] M. P. Bendsøe y C. Mota Soares. Topology design of structures. *Proc. NATO ARW*, 1992.

[42] M. P. Bendsøe y O. Sigmund. *Topology optimization : theory, methods, and applications*. Springer, Berlin; New York, 2004.

[43] M. P. Bendsøe, A. Ben-Tal, y R. T. Haftka. New displacement-based methods for optimal truss topology design. *Proceedings of the 32nd AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Material Conference, Baltimore, MD*, pages 684–696, 1991.

[44] W. Bennage y A. Dhingra. Single and multiobjective structural optimization in discrete-continuous variables using simulated annealing. *International Journal for Numerical Methods in Engineering*, 38(16):2753–2773, 1995. ISSN 1097-0207. doi: 10.1002/nme.1620381606.

[45] C. Bierwirth, D. C. Mattfeld, y H. Kopfer. On permutation representations for scheduling problems. In *In 4th PPSN*, pages 310–318. Springer-Verlag, 1996.

[46] J. L. Blanton y R. L. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-299-2.

[47] T. Blickle y L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[48] L. Booker. *Intelligent Behavior as a Adaptation to the Task Environment*. PhD thesis, University of Michigan, 1982.

[49] S. Botello, J. L. Marroquin, E. Oñate, y J. V. Horebeek. Solving structural optimization problems with genetic algorithms and simulated annealing. *International Journal for Numerical Methods in Engineering*, 45(8):1069–1084,

1999. ISSN 1097-0207. doi: 10.1002/(SICI)1097-0207(19990720)45:8⟨1069::AID-NME620⟩3.0.CO;2-E.

[50] A. Brindle. *Genetic algorithms for function optimization*. PhD thesis, Dept. of Computing Science, University of Alberta, Edmonton, 1981.

[51] J. Cai y G. Thierauf. A parallel evolution strategy for solving discrete structural optimization. *Advances in Engineering Software*, 27(1-2):91–96, 1996. ISSN 09659978.

[52] C. V. Camp. Design of space trusses using big bang–big crunch optimization. *Journal of Structural Engineering*, 133(7):999–1008, 2007. doi: 10.1061/(ASCE)0733-9445(2007)133:7(999).

[53] C. V. Camp y B. J. Bichon. Design of space trusses using ant colony optimization. *Journal of Structural Engineering*, 130(5):741–751, 2004. doi: 10.1061/(ASCE)0733-9445(2004)130:5(741).

[54] C. V. Camp, S. Pezeshk, y G. Cao. Optimized design of two-dimensional structures using a genetic algorithm. *Journal of Structural Engineering*, 124 (5):551–559, 1998. ISSN 07339445.

[55] O. Canyurt y P. Hajela. A cellular framework for structural analysis and optimization. *Computer Methods in Applied Mechanics and Engineering*, 194 (30–33):3516 – 3534, 2005. ISSN 0045-7825. doi: 10.1016/j.cma.2005.01.014.

[56] S. E. Carlson. A general method for handling constraints in genetic algorithms. In *In Proceedings of the Second Annual Joint Conference on Information Science*, pages 663–667, 1995.

[57] C. W. Carroll. The created response surface technique for optimizing nonlinear, restrained systems. *Operations Research*, 9:169–185, 1961.

[58] D. L. Carroll. Chemical laser modeling with genetic algorithms. *AIAA Journal*, 34(2):338–346, 1996.

[59] A. Chan. *The design of Michell optimum structures,*. H.M.S.O., London, 1962.

[60] H. Chan. *Optimum structural design and linear programming*. College of Aeronautics, Cranfield, Bedfordshire, 1964.

[61] D. Chen. *Least weight design of 2-D and 3-D geometrically nonlinear framed structures using a genetic algorithm*. PhD thesis, 1997.

[62] S. Chen y S. Rajan. Improving the efficiency of genetic algorithms for frame designs. *Engineering Optimization*, 30(3):281 – 307–, 1998. ISSN 0305-215X.

[63] T.-Y. Chen y J.-J. Su. Efficiency improvement of simulated annealing in optimal structural designs. *Advances in Engineering Software*, 33(7–10):675 – 680, 2002. ISSN 0965-9978. doi: 10.1016/S0965-9978(02)00058-3.

[64] X. Chen, S. Liu, y S. He. The optimization design of truss based on ant colony optimal algorithm. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 2, pages 720 –723, aug. 2010. doi: 10.1109/ICNC.2010.5583282.

[65] F. Y. Cheng y D. Li. Multiobjective optimization design with pareto genetic algorithm. *Journal of Structural Engineering*, 123(9):1252–1261–, 1997. doi: 10.1061/(ASCE)0733-9445(1997)123:9(1252).

[66] G. Cheng y X. Guo. Epsilon-relaxed approach in structural topology optimization. *Structural and Multidisciplinary Optimization*, 13(4):258–266, 1997. ISSN 1615147X.

[67] R. Cheng, M. Gen, y Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & Industrial Engineering*, 30(4):983 – 997, 1996. ISSN 0360-8352. doi: 10.1016/0360-8352(96)00047-2.

[68] C. A. Coello Coello. Treating constraints as objectives for single-objective evolutionary optimization. *Engineering Optimization*, 32:275–308, 1999.

[69] C. A. Coello Coello, G. B. Lamont, y D. A. Van Veldhuisen. *Evolutionary algorithms for solving multi-objective problems*. Springer, New York, 2007.

[70] D. A. Coley. *An introduction to genetic algorithms for scientists and engineers*. World Scientific, Singapore ; River Edge, NJ, 1999.

[71] P. Corcoran. Configurational optimization of structures. *International Journal of Mechanical Sciences*, 12:459–462, 1970.

[72] P. Corcoran. *The Design of Minimum Weight Structures of Optimum Configuration*. PhD thesis, The City University, at London, 1970.

[73] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. 1943.

[74] H. L. Cox. *The design of structures of least weight,*. Pergamon Press, Oxford; New York, 1965.

[75] J. W. Crichton, M. R. Finley, y H. J. Henry. Machine adaptive systems : quarterly report no. 3 : 15 october-15. Technical report, University of Michigan, 1963.

[76] F. D. Croce, R. Tadei, y G. Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15 – 24, 1995. ISSN 0305-0548. doi: 10.1016/0305-0548(93)E0015-L.

[77] E. da Silva, H. Barbosa, y A. Lemonge. An adaptive constraint handling technique for differential evolution with dynamic use of variants in engineering optimization. *Optimization and Engineering*, 12:31–54, 2011. ISSN 1389-4420. doi: 10.1007/s11081-010-9114-2. 10.1007/s11081-010-9114-2.

[78] C. Darwin. *The origin of species by means of natural selection, or the preservation of favoured races in the struggle for life.* J. Murray, London, 1866.

[79] D. Dasgupta y Z. Michalewicz. *Evolutionary Algorithms in Engineering Applications*, volume 2. Springer, 1997.

[80] L. Davis. *Genetic algorithms and simulated annealing.* Pitman ; Morgan Kaufmann Publishers, London Los Altos, Calif., 1987.

[81] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 61–69, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3.

[82] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[83] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[84] K. A. De Jong y J. Sarma. Generation gaps revisited. In *Foundations of Genetic Algorithms.* Morgan Kaufmann, 1992.

[85] K. Deb y R. Agrawal. Simulated binary crossover for continuous search space. Technical report, 1994.

[86] K. Deb y H.-g. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evol. Comput.*, 9:197–221, June 2001. ISSN 1063-6560. doi: 10.1162/106365601750190406.

[87] K. Deb y D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3.

[88] K. Deb y M. Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.

[89] K. Deb y S. Gulati. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elements in Analysis and Design*, 37(5):447–465, 2001. ISSN 0168874X. doi: 10.1016/S0168-874X(00)00057-3.

[90] K. Deb, D. Joshi, y A. Anand. Real-coded evolutionary algorithms with parent-centric recombination. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 61 –66, may 2002. doi: 10.1109/CEC.2002.1006210.

[91] A. K. Dhingra y B. Lee. A genetic algorithm approach to single and multiobjective structural optimization with discrete and continuous variables. *Int. J. Numer. Meth. Engng.*, 37(23):4059–4080, 1994. ISSN 1097-0207.

[92] A. D. Dimarogonas. *Vibration for engineers*. Prentice Hall, Upper Saddle River, N.J., 1996.

[93] M. Dobbs y R. Nelson. Application of optimality criteria to automated structural design. *AIAA Journal*, 14(10):1436–1443, 1976.

[94] M. W. Dobbs y L. Felton. Optimization of truss geometry. *Journal of the Structural Division, ASCE*, 95:2105–2118, Oct 1969.

[95] A. Dominguez, I. Stiharu, y R. Sedaghati. Practical design optimization of truss structures using the genetic algorithms. *Research in Engineering Design*, 17:73–84, 2006. ISSN 0934-9839. doi: 10.1007/s00163-006-0020-8.

[96] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, 1992.

[97] W. Dorn, R. Gomory, y H. Greenberg. Automatic design of optimal structures. *Journal De Mecanique*, 3(6):2552, 1964.

[98] U. Dorndorf y E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22(1):25–40, 1995.

[99] T. Dreyer, B. Maar, y V. Schulz. Multigrid optimization in applications. *Journal of Computational and Applied Mathematics*, 120(1-2):67–84, 2000. ISSN 03770427. doi: 10.1016/S0377-0427(00)00304-6.

[100] G. Ebenau, J. Rottschäfer, y G. Thierauf. An advanced evolutionary strategy with an adaptive penalty function for mixed-discrete structural optimisation. *Advances in Engineering Software*, 36(1):29–38, 2005. ISSN 09659978. doi: 10.1016/j.advengsoft.2003.10.008.

[101] A. Eiben. Multi-parent recombination. In *Handbook of Evolutionary Computation*, pages 3–3. IOP Publishing Ltd. and Oxford University Press, 1997.

[102] A. Eiben, C. van Kemenade, y J. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. 1995.

[103] A. Eiben, T. Bäck, y N. Bohrweg. An empirical investigation of multi-parent recombination operators in evolution strategies. 1997.

[104] A. E. Eiben y J. E. Smith. *Introduction to evolutionary computing : with 28 tables*. Springer, Berlin [u.a.], 2007.

[105] M. El-Sayed y T. Jang. Structural optimization using unconstrained nonlinear goal programming algorithm. *Computers & Structures*, 52(4):723 – 727, 1994. ISSN 0045-7949. doi: 10.1016/0045-7949(94)90353-0.

[106] F. Erbatur, O. Hasançebi, I. Tütüncü, y H. KiIiç. Optimal design of planar and space structures with genetic algorithms. *Computers & Structures*, 75 (2):209–224–, 2000. ISSN 0045-7949.

[107] O. Erdal y F. O. Sonmez. Optimum design of composite laminates for maximum buckling load capacity using simulated annealing. *Composite Structures*, 71(1):45 – 52, 2005. ISSN 0263-8223. doi: 10.1016/j.compstruct.2004.09.008.

[108] O. K. Erol y I. Eksin. A new optimization method: Big bang–big crunch. *Advances in Engineering Software*, 37(2):106 – 111, 2006. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2005.04.005.

[109] L. J. Eshelman y J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In D. Whitley, editor, *Foundation of Genetic Algorithms 2*, pages 187–202, San Mateo, CA, 1993. Morgan Kaufmann.

[110] L. J. Eshelman, R. A. Caruana, y J. D. Schaffer. Biases in the crossover landscape. In *Proceedings of the third international conference on Genetic algorithms*, pages 10–19, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3.

[111] E.W. y Parkes. Joints in optimum frameworks. *International Journal of Solids and Structures*, 11(9):1017 – 1022, 1975. ISSN 0020-7683. doi: 10.1016/0020-7683(75)90044-X.

[112] B. Farshi y S. L. A. Minimum weight design of stress limited trusses. *ASCE J Struct Div*, 100(ST1):97–107, 1974.

[113] B. Farshi y A. Alinia-ziazi. Sizing optimization of truss structures by method of centers and force formulation. *International Journal of Solids and Structures*, 47(18–19):2508 – 2524, 2010. ISSN 0020-7683. doi: 10.1016/j.ijsolstr.2010.05.009.

[114] A. V. Fiacco y G. P. McCormick. Extensions of sumt for nonlinear programming : Equality constraints and extrapolation extensions of sumt for nonlinear programming : Equality constraints and extrapolation. *Management Science*, 12(11):816–828, 1966. doi: 10.1287/mnsc.12.11.816.

[115] R. A. Fisher. *The genetical theory of natural selection,*. The Clarendon Press, Oxford, 1939.

[116] P. Fleron. The minimum weight of trusses. *Bygningsstatiske Meddelser*, 35: 81–96, 1964.

[117] D. B. Fogel, T. Bäck, y Z. Michalewicz. *Evolutionary computation. Vol. 1, Basic algorithms and operators*. Institute of Physics Pub., Bristol; Philadelphia, 2000.

[118] L. Fogel, A. Owens, y M. Walsh. Artificial intelligence through a simulation of evolution. In A. Callahan, M. Maxfield, y L. Fogel, editors, *Biophysics and Cybernetic Systems*, pages 131–156. Spartan, Washington DC, 1965.

[119] S. Forrest y R. Tanese. *Documentation for prisoners dilemma and norms programs that use the genetic algorithm.* [s. n.], [United States?], 1986.

[120] P. Fourie y A. Groenwold. The particle swarm optimization in topology optimization. *Fourth World Congress of Structural and Multidisciplinary Optimization, No. Paper No. 154*, 2001.

[121] A. Fraser. Monte carlo analyses of genetic models. *Nature*, 181(4603):208–209, Jan. 1958. doi: 10.1038/181208a0.

[122] H. Furuta, J. He, y E. Watanabe. A fuzzy expert system for damage assessment using genetic algorithms and neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 11(1):37–45, 1996. ISSN 1467-8667. doi: 10.1111/j.1467-8667.1996.tb00307.x.

[123] M. Galante. Structures optimization by a simple genetic algorithm. *Numerical methods in engineering and applied sciences*, pages 862–70, 1992.

[124] M. Galante. Genetic algorithms as an approach to optimize real-world trusses. *International Journal for Numerical Methods in Engineering*, 39(3): 361–382, 1996. ISSN 1097-0207. doi: 10.1002/(SICI)1097-0207(19960215)39:3⟨361::AID-NME854⟩3.0.CO;2-1.

[125] G. Galilei. *Discorsi e dimostrazioni matematiche intorno a due nuove scienze.* Louis Elsevier, 1638.

[126] R. H. Gallagher y O. Zienkiewicz. *Optimum structural design; theory and applications.* Wiley, London, New York„ 1973.

[127] Z. Geem, J. Kim, y G. Loganathan. A new heuristic optimization algorithm: Harmony search. *Simulation*, 76(2):60–68, 2001. ISSN 00375497.

[128] R. A. Gellatly, L. Berke, A. F. F. D. L. (U.S.), y B. A. Company. *Optimal structural design.* Air Force Flight Dynamics Laboratory, Air Force Systems Command, United States Air Force, Wright-Patterson Air Force Base, Ohio, 1971.

[129] M. Gen y R. Cheng. Interval programming using genetic algorithms. In *Proceedings of the Sixth International Symposium on Robotics and Manufacturing*, 1996.

[130] M. Gen y R. Cheng. *Genetic algorithms and engineering optimization*. Wiley, New York, 2000.

[131] M. Ghasemi, E. Hinton, y R. Wood. Optimization of trusses using genetic algorithms for discrete and continuous variables. *Engineering Computations*, 16(3):272–303, 1999.

[132] D. Goldberg, K. Deb, y B. Korb. Don't worry, be messy. In R. Belew y L. Booker, editors, *Proc. of the 4th International Conference on Genetic Algorithms*, pages 24–30. Morgan Kaufmann, 1991.

[133] D. E. Goldberg. *Computer-aided gas pipeline operation using genetic algorithms and rule learning*. PhD thesis, University of Michigan, Ann Arbor, MI., 1983.

[134] D. E. Goldberg. Computer-aided pipeline operation using genetic algorithms and rule learning. 1984.

[135] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., Reading, Mass., 1989.

[136] D. E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems*, 5:139–167, 1990.

[137] D. E. Goldberg. *The design of innovation : lessons from and for competent genetic algorithms*. Kluwer Academic Publishers, Boston, 2002.

[138] D. E. Goldberg y K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991.

[139] D. E. Goldberg y R. Lingle. Alleles, loci, and the traveling salesman problem. In J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages –. Lawrence Erlbaum Associates, Publishers, 1985.

[140] D. E. Goldberg y M. Samtani. Engineering optimization via genetic algorithm. *Ninth Conference on Electronic Computation*, (August):1–9, 1986.

[141] J. J. Greffenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122 –128, jan. 1986. ISSN 0018-9472. doi: 10.1109/TSMC.1986.289288.

[142] J. J. Greffenstette y J. E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 20–27, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-066-3.

[143] D. Greiner, G. Winter, y J. Emperador. Optimising frame structures by different strategies of genetic algorithms. *Finite Elements in Analysis and Design*, 37(5):381 – 402, 2001. ISSN 0168-874X. doi: 10.1016/S0168-874X(00)00054-8.

[144] D. Greiner, J. Emperador, y G. Winter. Single and multiobjective frame optimization by evolutionary algorithms and the auto-adaptive rebirth operator. *Computer Methods in Applied Mechanics and Engineering*, 193(33-35):3711–3743, 2004. ISSN 00457825. doi: 10.1016/j.cma.2004.02.001.

[145] D. Grierson y W. Pak. Optimal sizing, geometrical and topological design using a genetic algorithm. *Structural and Multidisciplinary Optimization*, 6 (3):151–159–, 1993. doi: 10.1007/BF01743506.

[146] A. Groenwold, N. Stander, y J. Snyman. A regional genetic algorithm for the discrete optimal design of truss structures. *International Journal for Numerical Methods in Engineering*, 44(6):749–766, 1999. ISSN 1097-0207. doi: 10.1002/(SICI)1097-0207(19990228)44:6⟨749::AID-NME523⟩3.0.CO;2-F.

[147] L. Guo, W. Tang, y M. Yuan. Simultaneous topology and sizing optimization of trusses by an improved genetic algorithm. In *Technology and Innovation Conference 2009 (ITIC 2009), International*, pages 1 –5, oct. 2009. doi: 10.1049/cp.2009.1469.

[148] X. Guo y G. Cheng. Epsilon-continuation approach for truss topology optimization. *Acta Mechanica Sinica/Lixue Xuebao*, 20(5):526–533, 2004. ISSN 05677718.

[149] X. Guo, G. Cheng, y K. Yamazaki. A new approach for the solution of singular optima in truss topology optimization with stress and local buckling constraints. *Structural and Multidisciplinary Optimization*, 22(5):364–372, 2001. ISSN 1615147X. doi: 10.1007/s00158-001-0156-0.

[150] R. T. Haftka. Simultaneous analysis and design. *AIAA journal*, 23(7):1099–1103, 1985. ISSN 00011452.

[151] R. T. Haftka y M. P. Kamat. *Elements of structural optimization*. M. Nijhoff ; Distributors for the U.S. and Canada, Kluwer Academic Publishers, The Hague; Boston; Hingham, MA, USA, 1985.

[152] T. Hagishita y M. Ohsaki. Topology optimization of trusses by growing ground structure method. *Structural and Multidisciplinary Optimization*, 37:377–393, 2009. ISSN 1615-147X. doi: 10.1007/s00158-008-0237-4.

[153] P. Hajela y H. Ashley. Hybrid optimization of truss structures with strength and buckling constraints. pages 3. 11–3. 18, Tucson, AZ, USA, 1981. Conference of International Symposium on Optimum Structural Design. 11th ONR Naval Structural Mechanics Symposium (US Office of Naval Research).; Conference Code: 499.

[154] P. Hajela y E. Lee. Genetic algorithms in topological design of grillage structures. *Proc., IUTAM Symp. on Discrete Structural Systems, IUTAm, Zakopane, Poland*, 1993.

[155] P. Hajela y E. Lee. Genetic algorithms in truss topological optimization. *International Journal of Solids and Structures*, 32(22):3341 – 3357, 1995. ISSN 0020-7683. doi: 10.1016/0020-7683(94)00306-H.

[156] P. Hajela y C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural and Multidisciplinary Optimization*, 4(2):99–107–, 1992.

[157] P. Hajela y C. Shih. Multiobjective optimum design in mixed integer and discrete design variable problems. *AIAA Journal*, 28(4):670–675, 1990. ISSN 00011452.

[158] P. Hajela, E. Lee, y C. Lin. Genetic algorithms in structural topology optimization. *Topology Design of Structures*, pages 117–133, 1993.

[159] P. Hajela, E. Lee, y H. Cho. Genetic algorithms in topologic design of grillage structures. *Computer-Aided Civil and Infrastructure Engineering*, 13(1):13–22, 1998. ISSN 1467-8667. doi: 10.1111/0885-9507.00081.

[160] I. Hajirasouliha, K. Pilakoutas, y H. Moghaddam. Topology optimization for the seismic design of truss-like structures. *Computers & Structures*, 89(7–8): 702 – 711, 2011. ISSN 0045-7949. doi: 10.1016/j.compstruc.2011.02.003.

[161] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[162] P. Hancock. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In D. Whitley y J. Schaffer, editors, *Proceedings of the International Workshop on Combination of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 108–122, Los Alamitos, CA, 1992. IEEE Computer Society Press.

[163] W. E. Hart y R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195. Morgan Kaufmann, 1991.

[164] O. Hasançebi. Adaptive evolution strategies in structural optimization: Enhancing their computational performance with applications to large-scale structures. *Computers and Structures*, 86(1-2):119–132, 2008. ISSN 00457949. doi: 10.1016/j.compstruc.2007.05.012.

[165] O. Hasançebi y F. Erbatur. Evaluation of crossover techniques in genetic algorithm based optimum structural design. *Computers & Structures*, 78 (1-3):435–448–, 2000. ISSN 0045-7949.

[166] O. Hasançebi y F. Erbatur. Layout optimisation of trusses using simulated annealing. *Advances in Engineering Software*, 33(7–10):681 – 696, 2002. ISSN 0965-9978. doi: 10.1016/S0965-9978(02)00049-2.

[167] R. L. Haupt y S. E. Haupt. *Practical genetic algorithms*. John Wiley, Hoboken, N.J., 2004.

[168] W. Hemp. Studies in the theory of michell structures. In *Proceedings, International Congress of Applied Mechanics*, Munich, West Germany„ 1964.

[169] W. Hemp. *Optimum structures*. Clarendon Press, Oxford, 1973.

[170] W. Hemp y H. Chang. *Optimum design of pin-jointed frameworks,*. H.M. Stationery Off., London, 1970.

[171] F. Herrera y M. Lozano. Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. *Soft Computing - A Fusion of Foundations, Methodologies and Applications Soft Computing*, 7(8):545–562, 2003. ISSN 1432-7643.

[172] F. Herrera, M. Lozano, y J. Verdegay. Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convergence of real-coded genetic algorithms. *International Journal of Intelligent Systems*, 11(12):1013–1040, 1996. ISSN 08848173.

[173] J. Herskovits. A mathematical programming algorithm for multidisciplinary design optimization. volume 4, pages 2246–2255, Albany, NY, 2004. ISBN 1563477165. Conference of Collection of Technical Papers - 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference; Conference Date: 30 August 2004 through 1 September 2004; Conference Code: 65014.

[174] J. Herskovits, P. Mappa, y L. Juillen. An interior point algorithm for simultaneous analysis and design optimization. 2001.

[175] J. Hesser y R. Männer. Towards an optimal mutation probability for genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 23–32, London, UK, 1991. Springer-Verlag. ISBN 3-540-54148-9.

[176] F. S. Hillier y G. J. Lieberman. *Introduction to mathematical programming*. McGraw-Hill, New York, 1990.

[177] F. Hoffmeister y J. Sprave. Problem-independent handling of constraints by use of metric penalty functions. *Optimization*, 1996.

[178] J. H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor, 1975.

[179] J. H. Holland y L. of Computers Group. *Hierarchical descriptions, universal spaces and adaptive systems.* Defense Technical Information Center, Ft. Belvoir, 1968.

[180] A. Homaifar, C. Qi, y S. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242, 1994. doi: 10.1177/003754979406200405.

[181] R. Hoppe, S. Petrova, y V. Schulz. Primal-dual newton-type interior-point method for topology optimization. *Journal of Optimization Theory and Applications*, 114(3):545–571, 2002. ISSN 00223239. doi: 10.1023/A:1016070928600.

[182] M. Huang y J. S. Arora. Optimal design of steel structures using standard sections. *Structural and Multidisciplinary Optimization*, 14:24–35, 1997. ISSN 1615-147X. doi: 10.1007/BF01197555. 10.1007/BF01197555.

[183] W.-C. Huang, C.-Y. Kao, y J.-T. Horng. A genetic algorithm approach for set covering problems. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 569 –574 vol.2, jun 1994. doi: 10.1109/ICEC.1994.349997.

[184] S. Im y J. Lee. Adaptive crossover, mutation and selection using fuzzy system for genetic algorithms. *Artificial Life and Robotics*, 13(1):129–133, 2008. ISSN 14335298. doi: 10.1007/s10015-008-0545-1.

[185] K. Imai y L. Schmit. Configuration optimization of trusses. *Proc. ASCE*, 107(ST5):745–756, 1982.

[186] A. Isaacs, T. Ray, y W. Smith. An efficient hybrid algorithm for optimization of discrete structures. In X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. Tan, J. Branke, y Y. Shi, editors, *Simulated Evolution and Learning*, volume 5361 of *Lecture Notes in Computer Science*, pages 625–634. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-89693-7. doi: 10.1007/978-3-540-89694-4_63.

[187] Ismail y Farajpour. Constrained optimization of structures with displacement constraints under various loading conditions. *Advances in Engineering Software*, 41(4):580 – 589, 2010. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2009.11.005.

[188] A. Jain y D. Fogel. Case studies in applying fitness distributions in evolutionary algorithms. ii. comparing the improvements from crossover and gaussian mutation on simple neural networks. In *Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on*, pages 91 –97, 2000. doi: 10.1109/ECNN.2000.886224.

[189] C. Z. Janikow y Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In R. K. Belew y L. B. Booker, editors, *Proc. of the 4th International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann, 1991.

[190] D. Janson y J. Frenzel. Training product unit neural networks with genetic algorithms. *IEEE Expert*, 8(5):26 –33, oct 1993. ISSN 0885-9000. doi: 10.1109/64.236478.

[191] F. Jarre, M. Kocvara, y J. Zowe. Optimal truss design by interior-point methods. *SIAM Journal on Optimization*, 8(4):1084–1107, 1998. ISSN 10526234.

[192] W. Jenkins. Towards structural optimization via the genetic algorithm. *Computers & Structures*, 40(5):1321–1327–, 1991. ISSN 0045-7949.

[193] W. Jenkins. On the application of natural algorithms to structural design optimization. *Engineering Structures*, 19(4):302–308–, 1997. ISSN 0141-0296.

[194] W. Jenkins. A decimal-coded evolutionary algorithm for constrained optimization. *Computers and Structures*, 80(5-6):471–480, 2002. ISSN 00457949. doi: 10.1016/S0045-7949(02)00021-4.

[195] J. Joines y C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 579 –584 vol.2, jun 1994. doi: 10.1109/ICEC.1994.349995.

[196] A. Kaveh y V. Kalatjari. Topology optimization of trusses using genetic algorithm, force method and graph theory. *International Journal for Numerical Methods in Engineering*, 58(5):771–791–, 2003. ISSN 1097-0207. doi: 10.1002/nme.800.

[197] A. Kaveh y H. Rahami. Nonlinear analysis and optimal design of structures via force method and genetic algorithm. *Computers & Structures*, 84(12): 770 – 778, 2006. ISSN 0045-7949. doi: 10.1016/j.compstruc.2006.02.004.

[198] A. Kaveh y M. Shahrouzi. Simultaneous topology and size optimization of structures by genetic algorithm using minimal length chromosome. *Engineering Computations (Swansea, Wales)*, 23(6):644–674, 2006. ISSN 02644401. doi: 10.1108/02644400610680351.

[199] A. Kaveh y M. Shahrouzi. Optimal structural design family by genetic search and ant colony approach. *Engineering Computations (Swansea, Wales)*, 25 (3):268–288, 2008. ISSN 02644401. doi: 10.1108/02644400810857092.

[200] A. Kaveh y S. Talatahari. Hybrid algorithm of harmony search, particle swarm and ant colony for structural design optimization. In Z. Geem, editor, *Harmony Search Algorithms for Structural Design Optimization*, volume 239 of *Studies in Computational Intelligence*, pages 159–198. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-03449-7. doi: 10.1007/978-3-642-03450-3_5.

[201] A. Kaveh y S. Talatahari. Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures. *Computers & Structures*, 87(5-6):267 – 283, 2009. ISSN 0045-7949. doi: 10.1016/j.compstruc.2009.01.003.

[202] S. Kazarlis y V. Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *Parallel Problem Solving from Nature V—PPSN V*, pages 211–220. Springer-Verlag, 1998.

[203] A. Keane. Genetic algorithm optimization of multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9(2): 75 – 83, 1995. ISSN 0954-1810. doi: 10.1016/0954-1810(95)95751-Q.

[204] J. Kennedy y R. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks, Perth*, 4:1942–1948, 1995.

[205] N. Khot y L. Berke. Structural optimization using optimality criteria methods. pages 47–74, Tucson, AZ, USA, 1984. John Wiley & Sons (Wiley Series in Numerical Methods in Engineering), Chichester, Engl. ISBN 0471902918. Conference of New Directions in Optimum Structural Design (Proceedings of the Second International Symposium on Optimum Structural Design).; Conference Code: 8355.

[206] R. Kicinger y T. Arciszewski. Multiobjective evolutionary design of steel structures in tall buildings. volume 2, pages 820–831, Chicago, IL, 2004. ISBN 156347719X. Conference of Collection of Technical Papers - AIAA 1st Intelligent Systems Technical Conference; Conference Date: 20 September 2004 through 23 September 2004; Conference Code: 64930.

[207] R. Kicinger, S. Obayashi, y T. Arciszewski. Evolutionary multiobjective optimization of steel structural systems in tall buildings. *Lecture Notes*

in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4403 LNCS:604–618, 2007. ISSN 03029743. Conference of 4th International Conference on Evolutionary Multi-Criterion Optimization, EMO 2007; Conference Date: 5 March 2007 through 8 March 2007; Conference Code: 70795.

[208] R. P. Kicinger. *Emergent engineering design : design creativity and optimality inspired by nature*. PhD thesis, 2004.

[209] U. Kirsch. Synthesis of structural geometry using approximation concepts. *Computers and Structures*, 15(3):305–314, 1982. ISSN 00457949.

[210] U. Kirsch. Fundamental properties of optimal topologies. In *Proc. of NATO / ARW on Topology Design of Structures*, 1992.

[211] U. Kirsch y G. Rozvany. Alternative formulations of structural optimization. *Structural Optimization*, 7(1-2):32–41, 1994.

[212] H. Kita, I. Ono, y S. Kobayashi. Theoretical analysis of the unimodal normal distribution crossover for real-coded genetic algorithms. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 529–534, 1998.

[213] H. Kita, I. Ono, y S. Kobayashi. Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 3 vol. (xxxvii+2348), 1999. doi: 10.1109/CEC.1999.782672.

[214] V. K. Koumousis y P. G. Georgiou. Genetic algorithms in discrete optimization of steel truss roofs. *Journal of Computing in Civil Engineering*, 8(3): 309–325–, 1994. doi: 10.1061/(ASCE)0887-3801(1994)8:3(309).

[215] J. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, 1994. ISSN 09603174. doi: 10.1007/BF00175355.

[216] J. R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. MIT Press, Cambridge, Mass., 1992.

[217] F. Kuan-Chen. An application of search technique in truss configurational optimization. *Computers and Structures*, 3(2):315–328, 1973. ISSN 00457949.

[218] N. D. Lagaros, M. Papadrakakis, y G. Kokossalakis. Structural optimization using evolutionary algorithms. *Computers & Structures*, 80(7-8):571–589–, 2002. ISSN 0045-7949.

[219] L. Lamberti. An efficient simulated annealing algorithm for design optimization of truss structures. *Computers & Structures*, 86(19-20):1936 – 1953, 2008. ISSN 0045-7949. doi: 10.1016/j.compstruc.2008.02.004.

[220] F. C. Lane, B. D. Coll, y U. S. M. Commission. *Ships for victory : a history of shipbuilding under the U.S. Maritime Commission in World War II*. John Hopkins Press, Baltimore, Md., 1951.

[221] F. Lara Peinado. *Código de Hammurabi*. F.L. Peinado, Madrid, 1986.

[222] T. Larsson y M. Rönnqvist. Simultaneous structural analysis and design based on augmented lagrangian duality. *Structural Optimization*, 9(1):1–11, 1995. ISSN 09344373. doi: 10.1007/BF01742636.

[223] M. Lawo y G. Thierauf. Optimal design for dynamic stochastic loading: A solution by random search. optimization in structural design. university of siegen. *Bibl. Inst. Mannheim*, 346-352, 1982.

[224] R. G. Le Riche y R. T. Haftka. Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm. *AIAA Journal*, 31(5): 951–956, 1993.

[225] R. G. Le Riche, C. Knopf-Lenoir, y R. T. T. Haftka. *A Segregated Genetic Algorithm for Constrained Structural Optimization*, pages 558–565. Morgan Kaufmann, 1995.

[226] K. S. Lee y Z. W. Geem. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194(36–38):3902 – 3933, 2005. ISSN 0045-7825. doi: 10.1016/j.cma.2004.09.007.

[227] M. A. Lee y H. Takagi. *A Framework for Studying the Effects of Dynamic Crossover, Mutation, and Population Sizing in Genetic Algorithms*, volume 1011, pages 111–126. Springer Verlag, 1995.

[228] J. Leite y B. Topping. Improved genetic operators for structural engineering optimization. *Advances in Engineering Software*, 29(7–9):529 – 562, 1998. ISSN 0965-9978. doi: 10.1016/S0965-9978(98)00021-0.

[229] A. Lemonge y H. Barbosa. An adaptive penalty scheme for genetic algorithms in structural optimization. *International Journal for Numerical Methods in Engineering*, 59(5):703–736, 2004. ISSN 00295981. doi: 10.1002/nme.899.

[230] O. E. Lev. Optimum choice of determinate methods under multiple loads. *ASCE J Struct Div*, 103(2):391–403, 1977.

[231] O. E. Lev. Topology and optimality of certain trusses. *ASCE J Struct Div*, 107(2):383–393, 1981.

[232] O. E. Lev. Sequential geometric optimization. *ASCE J Struct Div*, 107(10): 1935–1943, 1981. ISSN 00448001.

[233] L. Li, Z. Huang, F. Liu, y Q. Wu. A heuristic particle swarm optimizer for optimization of pin connected structures. *Computers & Structures*, 85(7-8): 340 – 349, 2007. ISSN 0045-7949. doi: 10.1016/j.compstruc.2006.11.020.

[234] L. Li, Z. Huang, y F. Liu. A heuristic particle swarm optimization method for truss structures with discrete variables. *Computers & Structures*, 87(7-8): 435 – 443, 2009. ISSN 0045-7949. doi: 10.1016/j.compstruc.2009.01.004.

[235] X. Li. An efficient solution to non-differentiable optimization problem. *China Science*, 24:371–377, 1994.

[236] G. E. Liepins y M. D. Vose. Representational issues in genetic optimization. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(2):101–115, 1990. doi: 10.1080/09528139008953717.

[237] C.-Y. Lin y P. Hajela. Genetic search strategies in large scale optimization. Number pt 4, pages 2437–2447, La Jolla, CA, USA, 1993. Publ by AIAA, Washington, DC, United States. Conference of 34th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference; Conference Date: 19 April 1993 through 22 April 1993; Conference Code: 18619.

[238] J. Lin, W. Che, y Y. Yu. Structural optimization on geometrical configuration and element sizing with statical and dynamical constraints. pages

2. 11–2. 19, Tucson, AZ, USA, 1981. Conference of International Symposium on Optimum Structural Design. 11th ONR Naval Structural Mechanics Symposium (US Office of Naval Research).; Conference Code: 499.

[239] J. Lin, W. Che, y Y. Yu. Structural optimization on geometrical configuration and element sizing with statical and dynamical constraints. *Computers and Structures*, 15(5):507–515, 1982. ISSN 00457949.

[240] W. Lingyun, Z. Mei, W. Guangming, y M. Guang. Truss optimization on shape and sizing with frequency constraints based on genetic algorithm. *Computational Mechanics*, 35:361–368, 2005. ISSN 0178-7675. doi: 10.1007/s00466-004-0623-8.

[241] S. L. Lipson y K. M. Agrawal. Weight optimization of plane trusses. *ASCE J Struct Div*, 100(ST5):865–879, 1974.

[242] S. L. Lipson y L. B. Gwin. Discrete sizing of trusses for optimal geometry. *ASCE J Struct Div*, 103(5):1031–1046, 1977.

[243] S. L. Lipson y L. B. Gwin. The complex method applied to optimal truss configuration. *Computers and Structures*, 7(3):461–468, 1977. ISSN 00457949.

[244] C. Liu, A. Hammad, y Y. Itoh. Maintenance strategy optimization of bridge decks using genetic algorithm. *Journal of Transportation Engineering*, 123 (2):91–100, 1997. doi: 10.1061/(ASCE)0733-947X(1997)123:2(91).

[245] G.-C. Luh y C.-Y. Lin. Optimal design of truss-structures using particle swarm optimization. *Computers & Structures*, 89(23–24):2221 – 2232, 2011. ISSN 0045-7949. doi: 10.1016/j.compstruc.2011.08.013.

[246] S. Luke. *Essentials of metaheuristics : a set of undergraduate lecture notes*. Lulu, [S.l.], 2009.

[247] B. Maar y V. Schulz. Interior point multigrid methods for topology optimization. *Structural and Multidisciplinary Optimization*, 19(3):214–224, 2000. ISSN 1615147X. doi: 10.1007/s001580050104.

[248] S. Mahfouz. *Design optimization of structural steelwork*. PhD thesis, 1999.

[249] K. Majid. *Optimum design of structures.* Newnes-Butterworths, London, 1974.

[250] K. Majid y D. Elliott. Forces and deflexions in changing structures. *Structural Engineer*, 51(3):93–101, 1973. ISSN 14665123.

[251] K. Majid y D. Elliott. Topological design of pin jointed structures by nonlinear programming. *Proc Inst Civ Eng (London)*, 55(Part 2):129–149, 1973.

[252] K. Majid, M. Saka, y T. Celik. Theorems of structural variation generalized for rigidly jointed frames. *Proc Inst Civ Eng (London)*, 65(pt 2):839–856, 1978. ISSN 00203262.

[253] E. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Trans. Comput.*, 26:1182–1191, December 1977. ISSN 0018-9340. doi: 10.1109/TC.1977.1674779.

[254] H. Martins y Gomes. Truss optimization with dynamic constraints using a particle swarm algorithm. *Expert Systems with Applications*, 38(1):957 – 968, 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2010.07.086.

[255] J. Masiá Vañó. *Investigación de la caracterización experimental de las uniones entre componentes de bastidores de vehículos pesados*. PhD thesis, Editorial de la Universidad Politécnica de Valencia, Valencia, 2005.

[256] C. J. Maxwell. *On reciprocal figures, frames, and diagrams of forces...* printed by Neill and C°, Edinburgh, 1870.

[257] J. Maynard Smith. *The evolution of sex*. Cambridge University Press, Cambridge [Eng.]; New York, 1978.

[258] J. Maynard Smith y E. Szathmáry. *The major transitions in evolution*. W.H. Freeman Spektrum, Oxford; New York, 1995.

[259] J. McKeown. The design of optimal trusses via sequences of optimal fixed displacement structures. *Eng. Opt.*, 14:159–178, 1989.

[260] A. Memari y A. Fuladgar. Minimum weight design of trusses by behsaz program. pages 179–185, Athens, Greece, 1994. Civil-Comp Limited, Edinburgh, United Kingdom. Conference of Proceedings of the 2nd International Conference on Computational Structures Technology. Part 1 (of 4); Conference Date: 30 August 1994 through 1 September 1994; Conference Code: 42517.

[261] R. Mercer y J. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978. doi: 10.1108/eb005486.

[262] H. Mühlenbein. Parallel genetic algorithm in combinatorial optimization, 1992.

[263] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer, Berlin u.a., 1992.

[264] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. MIT Press, 1995.

[265] Z. Michalewicz. *Genetic Algorithms , Numerical Optimization , and Constraints*, pages 151–158. Citeseer, 1995.

[266] Z. Michalewicz y G. Nazhiyath. Genocop iii: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 2, pages 647 –651 vol.2, nov-1 dec 1995. doi: 10.1109/ICEC.1995.487460.

[267] Z. Michalewicz, G. Nazhiyath, y M. Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Evolutionary Programming'96*, pages 305–312, 1996.

[268] A. Michell. The limits of economy of material in frame-structures. *Philosophical Magazine Series 6*, 8(47):589–597, 1904. doi: 10.1080/14786440409463229.

[269] B. L. Miller y D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evol. Comput.*, 4:113–131, June 1996. ISSN 1063-6560. doi: 10.1162/evco.1996.4.2.113.

[270] S. Missoum y Z. Gürdal. Displacement-based optimization for truss structures subjected to static and dynamic constraints. *AIAA Journal*, 40(1): 154–161, 2002. ISSN 00011452.

[271] S. Missoum, Z. Gürdal, y W. Gu. Optimization of nonlinear trusses using a displacement-based approach. *Structural and Multidisciplinary Optimization*, 23(3):214–221, 2002. ISSN 1615147X. doi: 10.1007/s00158-002-0179-1.

[272] S. Missoum, Z. Gürdal, y L. Watson. A displacement based optimization method for geometrically nonlinear frame structures. *Structural and Multidisciplinary Optimization*, 24(3):195–204, 2002. ISSN 1615147X. doi: 10.1007/s00158-002-0229-8.

[273] A. Morales y C. Quezada. A universal eclectic genetic algorithm for constrained optimization. In *Proceedings of 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT)*, pages 518–522, 1998.

[274] H. Mühlenbein y D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evol. Comput.*, 1: 25–49, March 1993. ISSN 1063-6560. doi: 10.1162/evco.1993.1.1.25.

[275] P. Nanakorn y K. Meesomklin. An adaptive penalty function in genetic algorithms for structural design optimization. *Computers Structures*, 79(29-30):2527–2539, 2001.

[276] J. Nelder y R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965. doi: 10.1093/comjnl/7.4.308.

[277] N. Noilublao y S. Bureerat. Simultaneous topology, shape and sizing optimisation of a three-dimensional slender truss tower using multiobjective evolutionary algorithms. *Computers & Structures*, 89(23–24):2531 – 2538, 2011. ISSN 0045-7949. doi: 10.1016/j.compstruc.2011.08.010.

[278] H. Ohmori y N. Kito. Structural optimization of truss topology by genetic algorithms. *Published by Publication Committee of NCTAM Proceedings*, pages 331–340, 1998.

[279] M. Ohsaki. Genetic algorithm for topology optimization of trusses. *Computers & Structures*, 57(2):219 – 225, 1995. ISSN 0045-7949. doi: 10.1016/0045-7949(94)00617-C.

[280] M. Ohsaki y N. Katoh. Topology optimization of trusses with stress and local constraints on nodal stability and member intersection. *Structural and Multidisciplinary Optimization*, 29:190–197, 2005. ISSN 1615-147X. doi: 10.1007/s00158-004-0480-2. 10.1007/s00158-004-0480-2.

[281] M. Ohsaki y C. Swan. *Topology and geometry optimization of trusses and frames*. American Society of Civil Engineers, 2002. ISBN 0784406367.

[282] I. Oliver, D. Smith, y J. H. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 224–230, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc. ISBN 0-8058-0158-8.

[283] A. Olsen. Penalty functions and the knapsack problem. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 554 –558 vol.2, jun 1994. doi: 10.1109/ICEC.1994.350000.

[284] I. Ono y S. Kobayashi. *A Real Coded Genetic Algorithm for Function Optimization Using Unimodal Normal Distributed Crossover*, volume 14, page 1146–1155. Morgan Kaufmann, 1997.

[285] I. Ono, H. Kita, y S. Kobayashi. A robust real-coded genetic algorithm using unimodal normal distribution crossover augmented by uniform crossover: Effects of self-adaptation of crossover probabilities. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, y R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 496–503, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann. ISBN 1-55860-611-4.

[286] I. Ono, H. Kita, y S. Kobayashi. *A real-coded genetic algorithm using the unimodal normal distribution crossover*, pages 213–237. Springer-Verlag New York, Inc., New York, NY, USA, 2003. ISBN 3-540-43330-9.

[287] C. Orozco y O. Ghattas. Sparse approach to simultaneous analysis and design of geometrically nonlinear structures. *AIAA journal*, 30(7):1877–1885, 1992. ISSN 00011452.

[288] C. Orozco y O. Ghattas. A reduced sand method for optimal design of non-linear structures. *International Journal for Numerical Methods in Engineering*, 40(15):2759–2774, 1997. ISSN 00295981.

[289] D. Orvosh y L. Davis. Shall we repair? genetic algorithmscombinatorial optimizationand feasibility constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 650–, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-299-2.

[290] D. Orvosh y L. Davis. Using a genetic algorithm to optimize problems with feasibility constraints. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 548 –553 vol.2, jun 1994. doi: 10.1109/ICEC.1994.350001.

[291] J. Owen. *The analysis and design of light structures,*. American Elsevier Pub. Co., New York, 1965.

[292] C. Pantelides y S.-R. Tzan. Modified iterated simulated annealing algorithm for structural synthesis. *Advances in Engineering Software*, 31(6):391 – 400, 2000. ISSN 0965-9978. doi: 10.1016/S0965-9978(00)00007-7.

[293] M. Papadrakakis, N. D. Lagaros, y Y. Tsompanakis. Structural optimization using evolution strategies and neural networks. *Computer Methods in Applied Mechanics and Engineering*, 156(1-4):309–333–, 1998. ISSN 0045-7825.

[294] E. Parkes. *Braced frameworks; an introduction to the theory of structures*. Pergamon Press, Oxford; New York, 1965.

[295] C. Pearson. Structural design by high-speed computing machines. *Proceedings of the First Conference on Electronic Computation*, pages 417–436, 1958.

[296] P. Pedersen. On the minimum mass layout of trusses. *Proc. AGARD Symposium 1970, Istanbul, AGARD-CP-36-7.*, 1970.

[297] P. Pedersen. On the optimal layout of multi-purpose trusses. *Computers and Structures*, 2(5-6):695–712, 1972. ISSN 00457949.

[298] P. Pedersen. Optimal joint positions for space trusses. *ASCE J Struct Div*, 99(St12):2459–2476, 1973.

[299] R. Perez y K. Behdinan. Particle swarm approach for structural design optimization. *Computers and Structures*, 85(19-20):1579–1588, 2007. ISSN 00457949. doi: 10.1016/j.compstruc.2006.10.013.

[300] S. Pezeshk y C. V. Camp. *State of the art on the use of genetic algorithms in design of steel structures*. American Society of Civil Engineers, 2002. ISBN 0784406367.

[301] S. Pezeshk, C. V. Camp, y D. Chen. Design of nonlinear framed structures using genetic optimization. *Journal of Structural Engineering*, 126(3):387–388, 2000. ISSN 07339445 (ISSN).

[302] W. Prager. A note on discretized michell structures. *Computer Methods in Applied Mechanics and Engineering*, 3(3):349 – 355, 1974. ISSN 0045-7825. doi: 10.1016/0045-7825(74)90019-X.

[303] W. Prager. Optimal layout of cantilever trusses. *Journal of Optimization Theory and Applications*, 23:111–117, 1977. ISSN 0022-3239. doi: 10.1007/BF00932301. 10.1007/BF00932301.

[304] W. Prager y G. Rozvany. Optimal layout of grillages. *ASME J. Struct. Mech.*, 5:1–18, 1976.

[305] W. Prager y G. Rozvany. Optimization of structural geometry. In *Proceedings of University of Florida international symposium*, page 265–294, Gainesville, 1977.

[306] M. B. Prendes Gero. *Optimización del diseño y construcción de edificios metálicos en base a algoritmos genéticos*. PhD thesis, Universidad de Oviedo, Servicio de Publicaciones, Oviedo, 2002.

[307] M. B. Prendes Gero, A. Bello García, y J. J. del Coz Díaz. A modified elitist genetic algorithm applied to the design optimization of complex steel structures. *Journal of Constructional Steel Research*, 61(2):265–280–, 2005. ISSN 0143-974X.

[308] M. B. Prendes Gero, A. Bello García, y J. J. del Coz Díaz. Design optimization of 3d steel structures: Genetic algorithms vs. classical techniques. *Journal of Constructional Steel Research*, 62(12):1303–1309–, 2006. ISSN 0143-974X.

[309] N. J. Radcliffe. Forma analysis and random respectful recombination. In *ICGA'91*, pages 222–229, 1991.

[310] S. D. Rajan. Sizing, shape, and topology design optimization of trusses using genetic algorithm. *Journal of Structural Engineering*, 121(10):1480–1487–, 1995. doi: 10.1061/(ASCE)0733-9445(1995)121:10(1480).

[311] S. Rajeev y C. S. Krishnamoorthy. Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering*, 118(5):1233–1250–, 1992. doi: 10.1061/(ASCE)0733-9445(1992)118:5(1233).

[312] S. Rajeev y C. S. Krishnamoorthy. Genetic algorithms-based methodologies for design optimization of trusses. *Journal of Structural Engineering*, 123 (3):350–358, 1997. doi: 10.1061/(ASCE)0733-9445(1997)123:3(350).

[313] I. Rechenberg. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1970.

[314] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

[315] C. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Sci., 1993.

[316] K. F. Reinschmidt y A. D. Russell. Linear methods in structural optimisation. Technical Report R 70-41, Department of Civil Engeneering, Cambridge, July 1970.

[317] K. F. Reinschmidt y A. D. Russell. Applications of linear programming in structural layout and optimization. *Computers and Structures*, 4(4):855–869, 1974. ISSN 00457949.

[318] J.-M. Renders y H. Bersini. Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 312 –317 vol.1, jun 1994. doi: 10.1109/ICEC.1994.349948.

[319] J. T. Richardson, M. R. Palmer, G. E. Liepins, y M. Hilliard. *Some Guidelines for Genetic Algorithms with Penalty Functions*, pages 191–197. Morgan Kaufmann Publishers, 1989.

[320] U. Ringertz. A branch and bound algorithm for topology optimization of truss structures. *Eng. Opt.*, 10:111–124, 1986.

[321] U. Ringertz. Optimization of structures with nonlinear response. *Engineering Optimization*, 14:179–188, 1989.

[322] P. Rizzi. Optimization of multi-constrained structures based on optimality criteria. In *Proceedings of Structural Dynamics, and Materials Conference, 17th, King of Prussia, Pa*, pages 448–462. American Institute of Aeronautics and Astronautics, Inc, May 5-7 1976.

[323] G. Roman, M. Uebersax, y O. König. Structural optimization tool using genetic algorithms and ansys. *Structural Optimization*, pages 1–10, 2000.

[324] G. Roston y S. R.H. Genetic algorithm synthesis of four-bar mechanics. *Analysis and Manufacturing*, 10:371–390, 1996.

[325] G. Rozvany. *Optimal design of flexural systems : beams, grillages, slabs, plates, and shells*. Pergamon Press, Oxford; New York, 1976.

[326] A. D. Russell y K. F. Reinschmidt. Discussion of "optimum design of trusses for ultimate loads". *Journal of the Structural Division*, 97(9):2437–2442, September 1971.

[327] W.-S. Ruy, Y.-S. Yang, G.-H. Kim, y Y.-S. Yeun. Topology design of truss structures in a multicriteria environment. *Computer-Aided Civil and Infrastructure Engineering*, 16(4):246–258, 2001. ISSN 1467-8667. doi: 10.1111/0885-9507.00230.

[328] M. Saka. Shape optimization of trusses. *J Struct Div ASCE*, 80(5 ST), 1980.

[329] M. Saka. Optimum design of rigidly jointed frames. *Computers and Structures*, 11(5):411–419, 1980. ISSN 00457949.

[330] M. Saka. Optimum design of grillage systems using genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, 13(4):297–302, 1998. ISSN 1467-8667. doi: 10.1111/0885-9507.00108.

[331] J. Sakamoto y J. Oda. Technique for optimal layout design for truss structures using genetic algorithm. Number pt 4, pages 2402–2408, La Jolla, CA, USA, 1993. Publ by AIAA, Washington, DC, United States. Conference of 34th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference; Conference Date: 19 April 1993 through 22 April 1993; Conference Code: 18619.

[332] M. Sakawa. *Genetic algorithms and fuzzy multiobjective optimization*. Kluwer Academic Publishers, Boston, 2002.

[333] J. D. Schaffer y L. J. Eshelman. On crossover as an evolutionarily viable strategy. In *ICGA'91*, pages 61–68, 1991.

[334] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, y R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 51–60, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3.

[335] L. Schmidt. Minimum weight layouts of elastic, statically determinate, triangulated frames under alternative load systems. *Journal of the Mechanics and Physics of Solids*, 10(2):139 – 149, 1962. ISSN 0022-5096. doi: 10.1016/0022-5096(62)90017-0.

[336] L. Schmit. Structural design by systematic synthesis. *Proceeding of the 2nd Conference on Electric Computation*, pages 105–122, 1960.

[337] L. Schmit y B. Farshi. Some approximation concepts for structural synthesis. *AIAA Journal*, 12(5):692–699, 1973.

[338] L. Schmit y R. Mallett. Structural synthesis and design parameter hierarchy. *Journal of the Structural Division*, 89:269–300, 1963.

[339] L. Schmit y H. Miura. A new structural analysis/synthesis capability - access. *AIAA Journal*, 14(5):661–671, 1975.

[340] L. Schmit y H. Miura. Approximation concepts for efficient structural synthesis. *NASA CR2552*, (CR-2552), 1976.

[341] L. Schmit y W. Morrow. Structural synthesis with buckling constraints. *Journal of the Structural Division, ASCE*, 89:107–126, April 1963.

[342] V. Schulz. Simultaneous solution approaches for large optimization problems. *Journal of Computational and Applied Mathematics*, 164(1):629–641, 2004. ISSN 03770427. doi: 10.1016/j.cam.2003.09.011.

[343] V. Schulz y H. Bock. Partially reduced sqp methods for large-scale nonlinear optimization problems. *Nonlinear Analysis, Theory, Methods and Applications*, 30(8):4723–4734, 1997. ISSN 0362546X.

[344] J. Schutte y A. Groenwold. Sizing design of truss structures using particle swarms. *Structural and Multidisciplinary Optimization*, 25:261–269, 2003. ISSN 1615-147X. doi: 10.1007/s00158-003-0316-5. 10.1007/s00158-003-0316-5.

[345] H.-P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981. ISBN 0471099880.

[346] B. Sendhoff, M. Kreutz, y W. V. Seelen. A condition for the genotype-phenotype mapping: Causality. In *Proc. International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufman, 1997.

[347] M. Serra y P. Venini. On some applications of ant colony optimization metaheuristic to plane truss optimization. *Structural and Multidisciplinary Optimization*, 32:499–506, 2006. ISSN 1615-147X. doi: 10.1007/s00158-006-0042-x. 10.1007/s00158-006-0042-x.

[348] K. Shahookar y P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(5):500–511, 1990. doi: 10.1109/43.55180.

[349] K. Shea y J. Cagan. The design of novel roof trusses with shape annealing: assessing the ability of a computational method in aiding structural designers with varying design intent. *Design Studies*, 20(1):3 – 23, 1999. ISSN 0142-694X. doi: 10.1016/S0142-694X(98)00019-2.

[350] K. Shea, J. Cagan, y S. Fenves. A shape annealing approach to optimal truss design with dynamic grouping of members. *Journal of Mechanical Design*, 119(3):388–394, 1997. doi: 10.1115/1.2826360.

[351] C. Sheu y L. Schmit Jr. Minimum weight design of elastic redundant trusses under multiple static loading conditions. *AIAA Journal*, 10(2):155–162, 1972. ISSN 00011452.

[352] Y. Shi y R. Eberhart. A modified particle swarm optimizer. *IEEE World Congress on Computational Intelligence. Evolutionary Computation Proceedings, 1998.*, pages 69–73, 1998.

[353] C. Shih. Fuzzy and improved penalty approaches for multiobjective mixed-discrete optimization in structural systems. *Computers and Structures*, 63 (3):559–565, 1997. ISSN 00457949.

[354] P. Shim y S. Manoochehri. Generating optimal configurations in structural design using simulated annealing. *International Journal for Numerical Methods in Engineering*, 40(6):1053–1069, 1997. ISSN 00295981.

[355] S. Sivanandam y S. Deepa. *Introduction to genetic algorithms*. Springer, Berlin; New York, 2008.

[356] A. E. Smith y D. M. Tate. *Genetic Optimization Using A Penalty Function*, pages 499–505. Morgan Kaufmann Publishers Inc., 1993.

[357] J. Smith. On replacement strategies in steady state evolutionary algorithms. *Evol. Comput.*, 15:29–59, March 2007. ISSN 1063-6560. doi: 10.1162/evco.2007.15.1.29.

[358] J. Smith y T. Fogarty. Recombination strategy adaptation via evolution of gene linkage. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 826 –831, may 1996. doi: 10.1109/ICEC.1996.542708.

[359] J. Smith y T. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 318 –323, may 1996. doi: 10.1109/ICEC.1996.542382.

[360] J. Smith y F. Vavak. Replacement strategies in steady state genetic algorithms: Static environments. In *FOGA'98*, pages 219–234, 1998.

[361] C. K. Soh y J. Yang. Fuzzy controlled genetic algorithm search for shape optimization. *Journal of Computing in Civil Engineering*, 10(2):143–150, 1996. doi: 10.1061/(ASCE)0887-3801(1996)10:2(143).

[362] C.-K. Soh y J. Yang. Optimal layout of bridge trusses by genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, 13(4):247–254, 1998. ISSN 1467-8667. doi: 10.1111/0885-9507.00103.

[363] Y. Song, G. Wang, A. Johns, y P. Wang. Improved genetic algorithms with fuzzy logic controlled crossover and mutation. Number 427 /1, pages 140–144, Exeter, UK, 1996. IEE, Stevenage, United Kingdom. Conference of

Proceedings of the 1996 UKACC International Conference on Control. Part 1 (of 2); Conference Date: 2 September 1996 through 5 September 1996; Conference Code: 45597.

[364] M. Sonmez. Artificial bee colony algorithm for optimization of truss structures. *Applied Soft Computing*, 11(2):2406 – 2418, 2011. ISSN 1568-4946. doi: 10.1016/j.asoc.2010.09.003.

[365] W. M. Spears y K. A. D. Jong. An analysis of multi-point crossover. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 301–315, San Mateo, CA, 1991. Morgan Kaufmann.

[366] W. Spillers y O. E. Lev. Design for two loading conditions. *International Journal of Solids and Structures*, 7(9):1261 – 1267, 1971. ISSN 0020-7683. doi: 10.1016/0020-7683(71)90066-7.

[367] W. R. Spillers. Iterative design for optimal geometry. *ASCE J Struct Div*, 101(7):1435–1442, 1975.

[368] W. R. Spillers. *Iterative structural design*. North-Holland Pub. Co. ; American Elsevier Pub. Co., Amsterdam; New York, 1975.

[369] W. R. Spillers y L. Friedland. On adaptive structural design. *Journal of the Structural Division, ASCE*, 98(ST10):2155–2163, October 1971.

[370] W. R. Spillers y G. E. Kountouris. Geometric optimization using simple code representation. *ASCE J Struct Div*, 106(5):959–971, 1980.

[371] M. Srinivas y L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994. ISSN 00189472. doi: 10.1109/21.286385.

[372] M. Srinivas y L. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17 –26, jun 1994. ISSN 0018-9162. doi: 10.1109/2.294849.

[373] V. Srinivas y K. Ramanjaneyulu. An integrated approach for optimum design of bridge decks using genetic algorithms and artificial neural networks. *Advances in Engineering Software*, 38(7):475 – 487, 2007. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2006.09.016.

[374] C. R. Stephens, I. G. Olmedo, J. M. Vargas, y H. Waelbroeck. Self-adaptation in evolving systems. *Artif. Life*, 4:183–201, April 1998. ISSN 1064-5462. doi: 10.1162/106454698568512.

[375] M. Stolpe y K. Svanberg. On the trajectories of the epsilon-relaxation approach for stress-constrained truss topology optimization. *Structural and Multidisciplinary Optimization*, 21(2):140–151, 2001. ISSN 1615147X. doi: 10.1007/s001580050178.

[376] M. Stolpe y K. Svanberg. A note on stress-constrained truss topology optimization. *Structural and Multidisciplinary Optimization*, 25:62–64, 2003. ISSN 1615-147X. doi: 10.1007/s00158-002-0273-4.

[377] R. Storn y K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328.

[378] M. Sugeno y G. Kang. Structure identification of fuzzy model. *Fuzzy Sets Syst.*, 28:15–33, October 1988. ISSN 0165-0114. doi: 10.1016/0165-0114(88)90113-3.

[379] G. Suresh, V. Vinod, y S. Sahu. A genetic algorithm for assembly line balancing. *Production Planning & Control*, 7(1):38–46, 1996. doi: 10.1080/09537289608930323.

[380] G. Sved. The minimum weight of certain redundant structures. *Australian Journal Of Applied Science*, 5(1-8), 1954.

[381] G. Sved y Z. Ginos. Structural optimization under multiple loading. *International Journal of Mechanical Sciences*, 10(10):803–805, 1968.

[382] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-066-3.

[383] G. Syswerda. A study of reproduction in generational and steady-state genetic algorithms. *Foundation of Genetic Algorithms*, pages 94–101, 1991.

[384] W. Tang, L. Tong, y Y. Gu. Improved genetic algorithm for design optimization of truss structures with sizing, shape and topology variables. *International Journal for Numerical Methods in Engineering*, 62(13):1737–1762–, 2005. ISSN 1097-0207. doi: 10.1002/nme.1244.

[385] D. M. Tate y A. E. Smith. A genetic approach to the quadratic assignment problem, 1995.

[386] S. R. Thangiah. An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 536–545, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-370-0.

[387] D. Thierens y D. E. Goldberg. Convergence models of genetic algorithm selection schemes. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature*, PPSN III, pages 119–129, London, UK, 1994. Springer-Verlag. ISBN 3-540-58484-6.

[388] H. Thomas y D. Brown. Optimum least-cost design of a truss roof system. *Computers and Structures*, 7(1):13–22, 1977. ISSN 00457949.

[389] F. Tin-Loi. A smoothing scheme for a minimum weight problem in structural plasticity. *Structural and Multidisciplinary Optimization*, 17(4):279–285, 1999. ISSN 1615147X.

[390] F. Tin-Loi. On the numerical solution of a class of unilateral contact structural optimization problems. *Structural Optimization*, 17(2):155–161, 1999. ISSN 09344373. doi: 10.1007/s001580050047.

[391] F. Tin-Loi. Optimum shakedown design under residual displacement constraints. *Structural and Multidisciplinary Optimization*, 19(2):130–139, 2000. ISSN 1615147X. doi: 10.1007/s001580050093.

[392] V. Togan y A. T. Daloglu. Optimization of 3d trusses with adaptive approach in genetic algorithms. *Engineering Structures*, 28(7):1019 – 1027, 2006. ISSN 0141-0296. doi: 10.1016/j.engstruct.2005.11.007.

[393] V. Togan y A. T. Daloglu. An improved genetic algorithm with initial population strategy and self-adaptive member grouping. *Computers & Structures*,

86(11–12):1204 – 1218, 2008. ISSN 0045-7949. doi: 10.1016/j.compstruc.2007.11.006.

[394] B. Topping. *The Application of Dynamic Relaxation to the Design of Modular Space Structures*. PhD thesis, The City University, London, Uk., 1978.

[395] B. Topping. Shape optimization of skeletal structures: A review. *Journal of Structural Engineering*, 109(8):1933–1951–, 1983.

[396] S. Tsutsui y D. E. Goldberg. Simplex crossover and linkage identification: Single-stage evolution vs. multi-stage evolution. In *in: Proceedings IEEE International Conference on Evolutionary Computation, 2002*, pages 974–979, 2002.

[397] A. Tuson y P. Ross. Co-evolution of operator settings in genetic algorithms. *EVOLUTIONARY COMPUTATION*, 6:161–184, 1996.

[398] G. N. Vanderplaats. Design of structures for optimum geometry. Technical Report TMX-62, National Aeronautics and Space Administration, August 1975.

[399] G. N. Vanderplaats. Structural optimization via a design space hierarchy. *International Journal for Numerical Methods in Engineering*, 10(3):713–717, 1976. ISSN 00295981.

[400] G. N. Vanderplaats y F. Moses. Automated design of trusses for optimum geometry. *ASCE J Struct Div*, 98(ST3):671–690, 1972.

[401] V. B. Venkayya, N. S. Khot, V. S. Reddy, y A. F. F. D. L. (U.S.). *Energy distribution in an optimum structural design*. Air Force Flight Dynamics Laboratory, Air Force Systems Command, United States Air Force, Wright-Patterson Air Force Base, Ohio, 1969.

[402] M. Victoria y P. Martí. Optimización conjunta de forma y topología de estructuras continuas bidimensionales mediante algoritmos evolucionarios en multi-nivel. In *Congreso de Métodos Numéricos en Ingeniería*, 2005.

[403] H.-M. Voigt, H. Mühlenbein, y D. Cvetkovic. Fuzzy recombination for the breeder genetic algorithm. In *Proc. Sixth Int. Conf. on Genetic Algorithms*, pages 104–111. Morgan Kaufmann Publishers, 1995.

[404] M. B. Wall. A genetic algorithm for resource-constrained scheduling by. *Design*, page 62, 1996.

[405] G. Walters, G. A., y D. K. Smith. Evolutionary design algorithm for optimal layout of tree networks. *Engineering Optimization*, 24(4):261–281, Aug. 1995. ISSN 0305-215X. doi: 10.1080/03052159508941193.

[406] Q. Wang y J. S. Arora. Alternate formulations for structural optimization. volume 2, pages 1413–1423, Palm Springs, CA, 2004. Conference of Collect. of Pap. - 45th AIAA/ASME/ASCE/AHS/ASC Struct., Struct. Dyn. and Mater. Conf.; 12th AIAA/ASME/AHS Adapt. Struct. Conf.; 6th AIAA Non-Deterministic Approaches Forum; 5th AIAA Gossamer Spacecraft Forum; Conference Date: 19 April 2004 through 22 April 2004; Conference Code: 64536.

[407] Q. J. Wang. The genetic algorithm and its application to calibrating conceptual rainfall-runoff models. *Water Resources Research*, 27(9):2467–2471, 1991.

[408] A. Wetzel. *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization*. PhD thesis, University of Pittsburgh, 1983.

[409] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

[410] D. Whitley. Functions as permutations: Implications for no free lunch, walsh analysis and statistics. In *In*, pages 169–178. Springer-Verlag, 2000.

[411] D. Wolpert y W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[412] S. Woon, O. Querin, y G. Steven. On improving the ga step-wise shape optimization method through the application of the fixed grid fea paradigm. *October*, 25(2003):270–278, 2003.

[413] A. Wright. Genetic algorithms for real parameter optimization. In G. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218, San Mateo, CA, 1991. Morgan Kaufmann.

[414] B. Wu, X. Yu, y L. Liu. Fuzzy penalty function approach for constrained function optimization with evolutionary algorithms. 2003.

[415] C.-Y. Wu y K.-Y. Tseng. Truss structure optimization using adaptive multipopulation differential evolution. *Structural and Multidisciplinary Optimization*, 42:575–590, 2010. ISSN 1615-147X. doi: 10.1007/s00158-010-0507-9. 10.1007/s00158-010-0507-9.

[416] W.-H. Wu y C.-Y. Lin. The second generation of self-organizing adaptive penalty strategy for constrained genetic search. *Advances in Engineering Software*, 35(12):815–825, 2004. ISSN 09659978. doi: 10.1016/j.advengsoft.2004.06.014.

[417] J. Xiao, Z. Michalewicz, y L. Zhang. Evolutionary planner/navigator: operator performance and self-tuning. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 366 –371, may 1996. doi: 10.1109/ICEC.1996.542391.

[418] J. Xiao, Z. Michalewicz, L. Zhang, y K. Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *Evolutionary Computation, IEEE Transactions on*, 1(1):18 –28, apr 1997. ISSN 1089-778X. doi: 10.1109/4235.585889.

[419] B. Xu, J. Jiang, y J. Ou. Integrated optimization of structural topology and control for piezoelectric smart trusses using genetic algorithm. *Journal of Sound and Vibration*, 307(3–5):393 – 427, 2007. ISSN 0022-460X. doi: 10.1016/j.jsv.2007.05.057.

[420] T. Xu, W. Zuo, T. Xu, G. Song, y R. Li. An adaptive reanalysis method for genetic algorithm with application to fast truss optimization. *Acta Mechanica Sinica*, 26:225–234, 2010. ISSN 0567-7718. doi: 10.1007/s10409-009-0323-x. 10.1007/s10409-009-0323-x.

[421] J. Yang y C. K. Soh. Structural optimization by genetic algorithms with tournament selection. *Journal of Computing in Civil Engineering*, 11(3): 195–200–, 1997. doi: 10.1061/(ASCE)0887-3801(1997)11:3(195).

[422] G. Yen. An adaptive penalty function for handling constraint in multiobjective evolutionary optimization. *Studies in Computational Intelligence*, 198:121–143, 2009. ISSN 1860949X. doi: 10.1007/978-3-642-00619-7_6.

[423] A. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005. ISSN 1300686X.

[424] T. Yokota, M. Gen, K. Ida, y T. Taguchi. Optimal design of system reliability by an improved genetic algorithm. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 79(2):41–51, 1996. ISSN 1520-6440. doi: 10.1002/ecjc.4430790205.

[425] W. C. Young, R. G. Budynas, y R. J. Roark. *Roark's formulas for stress and strain*. McGraw-Hill, New York, 2002.

[426] K. Zhu y Z. Liu. Population diversity in permutation-based genetic algorithm. In J.-F. Boulicaut, F. Esposito, F. Giannotti, y D. Pedreschi, editors, *Machine Learning: ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 537–547. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-23105-9. doi: 10.1007/978-3-540-30115-8_49.

Índice de figuras

Índice de tablas

Apéndices

A

Propiedades geométricas de las secciones implementadas

A.1 Sección rectangular

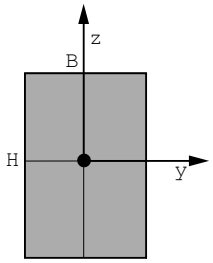


Figura A.1: Tipo de sección rectangular (RECT)

A.1.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- B = Ancho
- H = Alto
- Nb = Número de divisiones en la malla a lo largo de la anchura (2 por defecto)
- Nh = Número de divisiones en la malla a lo largo de la altura (2 por defecto)

Los únicos parámetros necesarios para definir la sección son los dos primeros, mientras que los otros dos controlan la malla, por lo que solo estos dos son almacenados en el genoma.

A.1.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Ambos parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma: $s_1 = B, s_2 = H$.

A.1.3 Cálculo de la tensión de torsión

Siguiendo las expresiones de Young et al. [425], se puede deducir el cálculo de la tensión mediante las siguientes expresiones:

$$\tau_{\max} = \frac{T}{K} \tag{A.1}$$

donde T es el par torsor y K se calcula según la ecuación:

$$K = \frac{ab^2}{3}\left[1 + 0,6095\lambda + 0,886\lambda^2 - 1,8023\lambda^3 + 0,91\lambda^4\right]^{-1} \tag{A.2}$$

donde $\lambda = \dfrac{b}{a}$

A.2 Sección rectangular hueca (HREC)

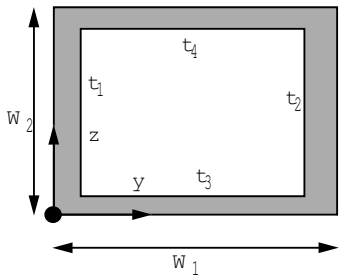


Figura A.2: Tipo de sección rectangular (HREC)

A.2.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- W_1 = Ancho
- W_2 = Alto
- t_1 = Espesor de un lado.
- t_2 = Espesor de un lado.
- t_3 = Espesor de un lado.

- t_4 = Espesor de un lado.

Todos los parámetros son necesarios para definir la sección, por lo que estos son almacenados en el genoma.

A.2.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los dos primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que el resto serán de un valor numérico pequeño codificándose en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_5 = t_1$, $s_6 = t_2$, $s_6 = t_3$, $s_7 = t_4$.

A.2.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K se puede deducir a partir de las expresiones de Young et al. [425].

$$K_0 = \left(W_1 - \frac{t_1 + t_2}{2}\right)\left(W_2 - \frac{t_3 + t_4}{2}\right) \tag{A.3}$$

$$K_1 = 2t_1 K_0 \tag{A.4}$$

$$K_2 = 2t_2 K_0 \tag{A.5}$$

$$K_3 = 2t_3 K_0 \tag{A.6}$$

$$K_4 = 2t_4 K_0 \tag{A.7}$$

$$\tag{A.8}$$

Finalmente el valor de K se obtiene mediante la expresión:

$$K = \min\{K_1, K_2, K_3, K_4\} \tag{A.9}$$

A.3 Sección circular

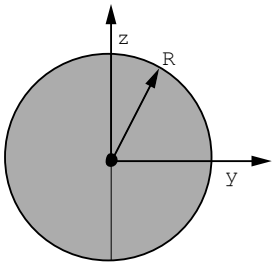


Figura A.3: Tipo de sección circular (CSOLID)

A.3.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- R = Radio
- N = Número de divisiones de la circunferencia
- T = Número de divisiones a lo largo del radio

El único parámetro necesario para definir la sección es el primero, mientras que los otros dos controlan la malla, por lo que sólo este es almacenados en el genoma.

A.3.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

El parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma: $s_1 = \mathrm{R}$.

A.3.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K en este caso es bien conocido y se calcula según la ecuación:

$$K = \frac{\pi R^3}{2} \tag{A.10}$$

A.4 Sección tubular

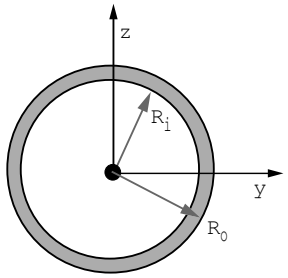


Figura A.4: Tipo de sección circular (CTUBE)

A.4.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- R_i = Radio interior
- R_o = Radio exterior
- N = Número de divisiones de la circunferencia

Los únicos parámetros necesarios para definir la sección son los dos primeros, mientras que el otro controla la malla, por lo que solo estos dos son almacenados en el genoma.

A.4.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Ambos parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma: $s_1 = R_i$, $s_2 = R_o$.

A.4.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K en este caso es bien conocido y se calcula según la ecuación:

$$K = \frac{\pi(R_o^4 - R_i^4)}{2R_o} \tag{A.11}$$

A.5 Sección en T

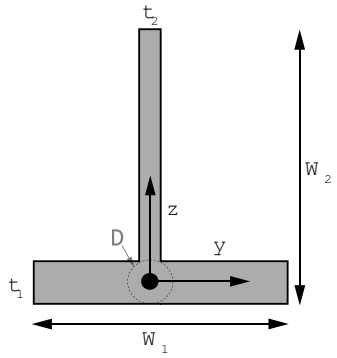


Figura A.5: Tipo de sección en T (T)

A.5.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- W_1 = Ancho del ala
- W_2 = Altura total
- t_1 = Espesor del ala

- t_2 = Espesor del alma

Todos los parámetros son necesarios para definir la sección y por lo tanto son almacenados en el genoma.

A.5.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los dos primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que los otros dos tendrán un valor numérico inferior por lo que se almacenarán en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_5 = t_1$, $s_6 = t_2$.

A.5.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K se deduce a partir de las expresiones de Young et al. [425] mediante las siguientes ecuaciones:

$$K_1 = W_1 t_1^3 \left[\frac{1}{3} - 0,21\lambda_1 \left(1 - \frac{1}{12}\lambda_1^4\right)\right] \tag{A.12}$$

donde $\lambda_1 = \dfrac{t_1}{W_1}$

$$K_2 = (W_2 - t_1) t_2^3 \left[\frac{1}{3} - 0,105\lambda_2 \left(1 - \frac{1}{192}\lambda_2^4\right)\right] \tag{A.13}$$

donde $\lambda_2 = \dfrac{t_2}{W_2 - t_1}$

$$\alpha = 0,15\frac{t_2}{t_1} \tag{A.14}$$

$$D = \frac{t_1^2 + \frac{t_2^2}{4}}{t_1} \tag{A.15}$$

$$C = \frac{D}{1 + \left(\dfrac{\pi D^2}{4A}\right)} \tag{A.16}$$

donde A es el área del perfil:

$$A = W_1 t_1 + (W_2 - t_1) t_2 \tag{A.17}$$

$$K_3 = K_1 + K_2 + \alpha D^4 \tag{A.18}$$

Finalmente K se calcula como:

$$K = \frac{K_3}{C} \tag{A.19}$$

A.6 Sección en L

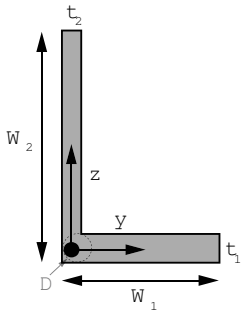


Figura A.6: Tipo de sección en L (L)

A.6.1 Parámetros del comando SECDATA

Los parámetros para esta sección son:

- W_1 = Ancho de un ala
- W_2 = Ancho de otra ala
- t_1 = Espesor de un ala
- t_2 = Espesor de otra ala

Todos los parámetros son necesarios para definir la sección y por lo tanto son almacenados en el genoma.

A.6.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los dos primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que los otros dos tendrán un valor numérico inferior por lo que se almacenarán en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_5 = t_1$, $s_6 = t_2$.

A.6.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K se calcula según Young et al. [425] mediante las siguientes ecuaciones:

$$K_1 = W_1 t_1^3 \left[\frac{1}{3} - 0,21\lambda_1 \left(1 - \frac{1}{12}\lambda_1^4\right)\right] \tag{A.20}$$

donde $\lambda_1 = \dfrac{t_1}{W_1}$

$$K_2 = (W_2 - t_1) t_2^3 \left[\frac{1}{3} - 0,105\lambda_2 \left(1 - \frac{1}{192}\lambda_2^4\right)\right] \tag{A.21}$$

donde $\lambda_2 = \dfrac{t_2}{W_2 - t_1}$

$$\alpha = 0,07\frac{t_2}{t_1} \tag{A.22}$$

$$D = 2(t_1 + t_2 - \sqrt{2t_1 t_2}) \tag{A.23}$$

$$C = \frac{D}{1 + \left(\dfrac{\pi D^2}{4A}\right)} \tag{A.24}$$

donde A es el área del perfil:

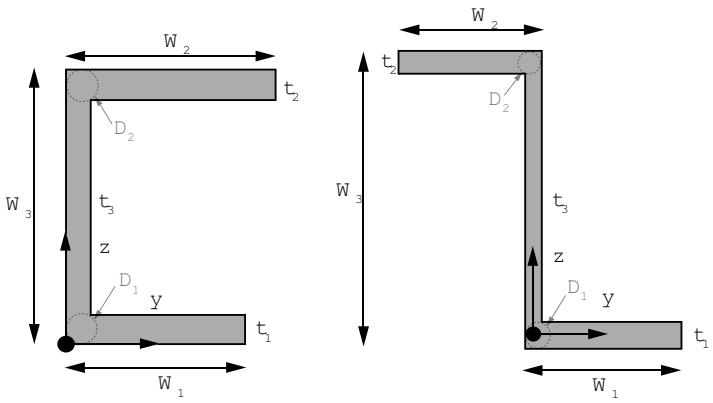
$$A = W_1 t_1 + (W_2 - t_1) t_2 \tag{A.25}$$

$$K_3 = K_1 + K_2 + \alpha D^4 \tag{A.26}$$

Finalmente K se calcula como:

$$K = \frac{K_3}{C} \tag{A.27}$$

A.7 Sección en U o en Z



(a) Sección en U (CHAN) (b) Sección en Z

Figura A.7: Secciones U,Z

A.7.1 Parámetros del comando SECDATA

Los parámetros para ambos tipos de sección son iguales:

- W_1 = Ancho de un ala
- W_2 = Ancho de otra ala
- W_3 = Altura total del perfil
- t_1 = Espesor de un ala
- t_2 = Espesor de otra ala
- t_3 = Espesor del alma

Todos los parámetros son necesarios para definir la sección y por lo tanto son almacenados en el genoma.

A.7.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los tres primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que los otros tres tendrán un valor numérico inferior por lo que se almacenarán en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_3 = W_3$, $s_5 = t_1$, $s_6 = t_2$, $s_7 = t_3$.

A.7.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K deduce de las expresiones de Young et al. [425] mediante la agregación de dos perfiles en L:

$$K_{11} = W_1 t_1^3 \left[\frac{1}{3} - 0,21\lambda_1 \left(1 - \frac{1}{12}\lambda_1^4\right)\right] \tag{A.28}$$

donde $\lambda_{11} = \dfrac{t_1}{W_1}$

$$K_{12} = \frac{W_3 - t_1 - t_2}{2} t_3^3 \left[\frac{1}{3} - 0,105\lambda_2 \left(1 - \frac{1}{192}\lambda_2^4\right)\right] \tag{A.29}$$

donde $\lambda_{12} = \dfrac{t_3}{0,5(W_3 - t_1 - t_2)}$

$$\alpha_1 = 0,07\frac{t_3}{t_1} \tag{A.30}$$

$$D_1 = 2(t_1 + t_3 - \sqrt{2t_1 t_3}) \tag{A.31}$$

$$C_1 = \frac{D_1}{1 + \left(\dfrac{\pi D_1^2}{4A}\right)} \tag{A.32}$$

donde A es el área del perfil:

$$A = W_1 t_1 + W_2 t_2 + (W_3 - t_1 - t_2)t_3 \tag{A.33}$$

$$K_{13} = K_{11} + K_{12} + \alpha D_1^4 \tag{A.34}$$

$$K_1 = \frac{K_{13}}{C_1} \tag{A.35}$$

$$K_{21} = W_2 t_1^3 \left[\frac{1}{3} - 0,21\lambda_{21} \left(1 - \frac{1}{12}\lambda_1^4 \right) \right] \tag{A.36}$$

donde $\lambda_{21} = \dfrac{t_2}{W_2}$

$$K_{22} = K_{21} \tag{A.37}$$

$$\alpha_2 = 0,07\frac{t_3}{t_2} \tag{A.38}$$

$$D_2 = 2(t_2 + t_3 - \sqrt{2t_2 t_3}) \tag{A.39}$$

$$C_2 = \frac{D_2}{1 + \left(\dfrac{\pi D_2^2}{4A} \right)} \tag{A.40}$$

$$K_{23} = K_{21} + K_{22} + \alpha_2 D_2^4 \tag{A.41}$$

$$K_2 = \frac{K_{23}}{C_2} \tag{A.42}$$

Finalmente el factor K se calculará mediante la expresión:

$$K = K_1 + K_2 \tag{A.43}$$

A.8 Sección en I

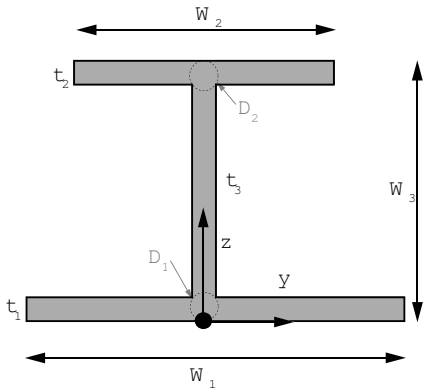


Figura A.8: Sección en I

A.8.1 Parámetros del comando SECDATA

Los parámetros para ambos tipos de sección son iguales:

- W_1 = Ancho de un ala
- W_2 = Ancho de otra ala
- W_3 = Altura total del perfil
- t_1 = Espesor de un ala
- t_2 = Espesor de otra ala
- t_3 = Espesor del alma

Todos los parámetros son necesarios para definir la sección y por lo tanto son almacenados en el genoma.

A.8.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los tres primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que los otros tres tendrán un valor numérico inferior por lo que se almacenarán en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_3 = W_3$, $s_5 = t_1$, $s_6 = t_2$, $s_7 = t_3$.

A.8.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K deduce de las expresiones de Young et al. [425] mediante la agregación de dos perfiles en L. Los factores K_{11}, K_{12}, C_1, A ,K_{13}, K_1, K_{21}, K_{22}, C_2, K_{23} y K se calculan según las ecuaciones (A.28), (A.29), (A.32) a (A.37), (A.40), (A.41) y (A.43) respectivamente. Los siguientes factores difieren de los anteriores y deben ser calculados:

$$\alpha_1 = 0,15\frac{t_3}{t_1} \tag{A.44}$$

$$D_1 = \frac{t_1^2 + \frac{t_3^2}{4}}{t_1} \tag{A.45}$$

$$\alpha_2 = 0,15\frac{t_3}{t_2} \tag{A.46}$$

$$D_2 = \frac{t_2^2 + \frac{t_3^2}{4}}{t_2} \tag{A.47}$$

A.9 Sección en omega (HATS)

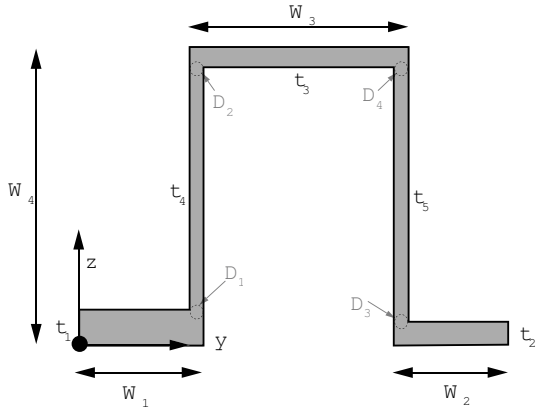


Figura A.9: Sección en omega (HATS)

A.9.1 Parámetros del comando SECDATA

Los parámetros para ambos tipos de sección son iguales:

- W_1 = Ancho de un ala
- W_2 = Ancho de otra ala
- W_3 = Anchura de la copa de la omega
- W_4 = Altura de omega
- t_1 = Espesor de un ala
- t_2 = Espesor de otra ala
- t_3 = Espesor un alma
- t_4 = Espesor del alma opuesta

Todos los parámetros son necesarios para definir la sección y por lo tanto son almacenados en el genoma.

A.9.2 Equivalencia entre los parámetros del cromosoma de geometría y SECDATA

Los cuatro primeros parámetros suelen tener un valor numérico elevado por lo que se codifican dentro de los primeros cuatro parámetros del cromosoma, mientras que los otros cuatro tendrán un valor numérico inferior por lo que se almacenarán en el segundo grupo: $s_1 = W_1$, $s_2 = W_2$, $s_3 = W_3$, $s_4 = W_4$, $s_5 = t_1$, $s_6 = t_2$, $s_7 = t_3$, $s_8 = t_4$.

A.9.3 Cálculo de la tensión de torsión

El cálculo de la tensión de cortadura se realiza mediante la ecuación (A.1). El factor K deduce de las expresiones de Young et al. [425] mediante la agregación de dos perfiles en Z.

B

Listado de comandos para el análisis con ANSYS APDL del mejor individuo

El siguiente listado muestra el conjunto de comandos que definen y calculan el mejor individuo de la población. Puede cargarse en cualquier versión de ANSYS APDL mediante la opción File>Read Input From o bien mediante línea de comandos empleando el siguiente comando:

ansysxxx -np 8 -b -i nombrearchivo.dat -o nombrearchivo.out

donde *xxx* es la versión de programa instalada.

```
! *************************** INICIO DEL LISTADO ********************
/NOLIST
/NERR,,,-1
/CWD,'C:\Simul'
/FILNAME,21340-23720-14-15-7895,0
/PREP7
PIVCHECK, OFF
/TITLE,21340-23720-14-15-7895

ANTYPE,STATIC
*AFUN,DEG

ET,1,BEAM44
*SET,Y, 21340.9
MP,EX,1,68950
MP,NUXY,1,0.33
MP,DENS,1,2.76E-6

SECTYPE, 2, BEAM, L, S2, 0
SECOFFSET, CENT
SECDATA, 305, 311, 3, 3
SECTYPE, 3, BEAM, T, S3, 0
SECOFFSET, CENT
SECDATA, 164, 98, 15, 15
SECTYPE, 5, BEAM, T, S5, 0
SECOFFSET, CENT
SECDATA, 46, 252, 12, 6
SECTYPE, 7, BEAM, HREC, S7, 0
SECOFFSET, CENT
SECDATA, 157, 231, 6, 9, 11, 8
SECTYPE, 9, BEAM, HREC, S9, 0
SECOFFSET, CENT
SECDATA, 346, 28, 2, 15, 5, 4
SECTYPE, 10, BEAM, I, S10, 0
SECOFFSET, CENT
SECDATA, 355, 41, 283, 9, 1, 14
```

```
K,1,9140,0,0
FK,1,FY, -444800
K,2,18280,0,0
FK,2,FY, -444800

K,3,0,0,0
DK,3, , , ,0,UX ,UY ,UZ
K,4,0,Y,0
DK,4, , , ,0,UX ,UY ,UZ
K,5,15435.6,8285.09,0
LSTR, 1,3
LSTR, 1,4
LSTR, 2,3
LSTR, 2,5
LSTR, 3,5
LSTR, 4,5

WPSTYLE,,,,,,1
CSYS,4
LSEL,,,,1
LWPLAN,-1,1,0.5
K,12,100,225
LATT,1,,,,12,12,2
LSEL,,,,2
LWPLAN,-1,2,0.5
K,13,100,135
LATT,1,,,,13,13,3
LSEL,,,,3
LWPLAN,-1,3,0.5
K,15,100,315
LATT,1,,,,15,15,5
LSEL,,,,4
LWPLAN,-1,4,0.5
K,17,100,270
LATT,1,,,,17,17,7
```

```
LSEL,,,,5
LWPLAN,-1,5,0.5
K,19,100,270
LATT,1,,,,19,19,9
LSEL,,,,6
LWPLAN,-1,6,0.5
K,20,100,135
LATT,1,,,,20,20,10

ALLSEL, ALL
LESIZE,ALL, , ,10, ,1, , ,1,
LMESH, ALL
CSYS,4

/SOL
SOLVE

/POST1

SABS,1
ETABLE,SMAXI,NMISC, 1
ETABLE,SMAXJ,NMISC, 3
SMAX,SMAX1,SMAXI,SMAXJ,1,1,
ETABLE,SMINI,NMISC, 2
ETABLE,SMINJ,NMISC, 4
SMAX,SMAX2,SMINI,SMINJ,1,1,
SMAX, SAX, SMAX1,SMAX2,1,1,
ETABLE, MOMXI,SMISC,4
ETABLE, MOMXJ,SMISC,10
SMAX, TOR, MOMXI,MOMXJ,1,1,
ETABLE,DX,U,X
ETABLE,DY,U,Y
ETABLE,DZ,U,Z
SMAX, D1, DX,DY,1,1,
SMAX, D2, DY,DZ,1,1,
SMAX, DEF, D1,D2,1,1,
```

```
ETABLE,VOL,VOLU

LSEL,S,,,1
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN1,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
*VSCFUN,VOL1,SUM,VLM
*DEL,VLM

*VMASK,NMASK
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR1,MAX,TORQUE
*DEL,TORQUE

LSEL,S,,,2
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN2,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
*VSCFUN,VOL2,SUM,VLM
*DEL,VLM

*VMASK,NMASK
```

```
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR2,MAX,TORQUE
*DEL,TORQUE

LSEL,S,,,3
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN3,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
*VSCFUN,VOL3,SUM,VLM
*DEL,VLM

*VMASK,NMASK
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR3,MAX,TORQUE
*DEL,TORQUE

LSEL,S,,,4
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN4,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
*VSCFUN,VOL4,SUM,VLM
*DEL,VLM
```

```
*VMASK,NMASK
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR4,MAX,TORQUE
*DEL,TORQUE

LSEL,S,,,5
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN5,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
*VSCFUN,VOL5,SUM,VLM
*DEL,VLM

*VMASK,NMASK
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR5,MAX,TORQUE
*DEL,TORQUE

LSEL,S,,,6
ESLL,S
*VGET,NMASK,ELEM,1,esel
*VMASK,NMASK
*VGET,ST,ELEM, ,ETAB,SAX, , ,2
*VABS,1,1,,
*VSCFUN,TEN6,MAX,ST
*DEL,ST

*VMASK,NMASK
*VGET,VLM,ELEM, ,ETAB,VOL , , ,2
```

```
*VSCFUN,VOL6,SUM,VLM
*DEL,VLM

*VMASK,NMASK
*VGET,TORQUE,ELEM, ,ETAB,TOR, , ,2
*VSCFUN,TOR6,MAX,TORQUE
*DEL,TORQUE

*VGET,DP,ELEM, ,ETAB,DEF, , ,2
*VABS,1,1,,
*VSCFUN,DEF,MAX,DP

ALLSEL,ALL

PARSAV,SCALAR,'C:\Simul\21340-23720-14-15-7895','res',

! *************************** FIN DEL LISTADO ********************
```

C

Código fuente de la clase GASOPGenome

El siguiente listado muestra la clase principal del algoritmo utilizado. No se incluye el resto de código del programa porque la extensión del presente trabajo resultaría excesiva[1]. El código completo puede solicitarse al autor dirigiéndose al correo electrónico `sasanca@dimm.upv.es`

```
1  //GASOPGenome.h
2  //Cabecera de la Clase GASOPGenome
3  #if !defined(_GASOPGENOME_H_)
4  #define _GASOPGENOME_H_
5  #include <string>
6  #include <iostream>
7  #include <strstream>
8
9  #include <ga/GAGenome.h>
10 #include <ga/GARealGenome.h>
11 #include "crossover.h"
12 #include "Funciones.h"
13 #include "torsion.h"
14 #include "etokenizer.h"
15
16 // Generamos objetos con los tipos de perfil
17 L perfil_l;
18 I perfil_i;
19 T perfil_t;
20 CHAN perfil_z;
21 CHAN perfil_u;
22 HATS perfil_omega;
23 RECTANGULO r;
24 HREC r_hueco;
25 CTUBE eje;
26 CTUBE eje_hueco;
27
28 class GAsOPGenome : public GAGenome {
29 public:
30     GADefineIdentity("GAsOPGenome", 201);
31     static void GAsOPInitializer(GAGenome &);
32     static int GAsOPMutator(GAGenome & , float);
```

[1]El código completo está compuesto por más de quince mil líneas

```
33      static float GAsOPComparator(const GAGenome & , const ↩
           GAGenome &);
34      static float GAsOPEvaluator(GAGenome &);
35      static int GAsOPCrossover(const GAGenome & a, const ↩
           GAGenome & b,
36           GAGenome *c, GAGenome *d);
37
38      // Constructores
39      GAsOPGenome(const int &, const GARealAlleleSetArray & ,
40           const GARealAlleleSetArray & , const ↩
              GARealAlleleSetArray &,
41           const GARealAlleleSetArray & , const ↩
              GARealAlleleSetArray & ,
42           const GARealAlleleSetArray & , GAGenome::Evaluator f ↩
              = NULL);
43
44      GAsOPGenome(const GAsOPGenome &); // constructor copia
45      GAsOPGenome& operator = (const GAGenome & g); // operador↩
           de asignación
46
47      // Metodos que ocultan a los de la clase GAGenome
48      virtual GAGenome* clone(GAGenome::CloneMethod)const;
49      virtual void copy(const GAGenome &);
50      virtual int equal(const GAGenome &g)const;
51      virtual int write(ostream & os)const;
52      virtual int read(istream & is);
53      virtual ~GAsOPGenome();
54
55      // Retorno variables protegidas
56      int getnumnodos()const {
57          return numnodos;
58      }
59
60      int getnumelementos()const {
61          return numelementos;
62      }
63
64      // Retorno cromosomas protegidos
65      GARealGenome & getcons()const {
```

```
66          return *cons;
67      }
68
69      GARealGenome & getnod()const {
70          return *nod;
71      }
72
73      GARealGenome & getcon()const {
74          return *con;
75      }
76
77      GARealGenome & getmat()const {
78          return *mat;
79      }
80
81      GARealGenome & gettsec()const {
82          return *tsec;
83      }
84
85      GARealGenome & getsec()const {
86          return *sec;
87      }
88
89      // Métodos propios
90      string get_uid()const {
91          return uid;
92      };
93      void set_uid();
94
95      float get_massvol()const {
96          return massvol;
97      };
98
99      void set_massvol(float mv) {
100         massvol = mv;
101     };
102
103     float get_gviol()const {
104         return gviol;
```

```
105     };
106
107     void set_gviol(float gv) {
108         gviol = gv;
109     };
110
111     float get_area(int numarea)const {
112         return Area[numarea];
113     };
114
115     vector<float>get_Ktorsion()const {
116         return Ktorsion;
117     }; // Devuelve el vector entero
118
119     void set_tension(vector<float> &ten){
120       Tension.clear();
121       Tension = ten;
122     }
123     bool write_tension(ostream &) const;
124     bool write_Ktorsion(ostream &) const;
125
126     bool check_all();
127     bool check_geom(); // Comprobación de la geometría
128     bool check_con();
129
130     float genedistance(const GAGenome& , const GAGenome& , ↩
          int ngen);
131
132     bool loadfile(const string &, int numgen = 1);
133     void write_conection_array(ostream &)const;
134     bool check_genome(ostream & os)const; // Comprobación del↩
          genoma
135
136 public:
137
138 protected:
139     // Funciones
140     bool parametros_seccion();
141
```

```
142     // Variables
143     static int numelementos; // Número de barras de la ↩
            estructura
144     static int numnodos; // Número de nudos de la estructura
145     string uid; // Unique IDentifier. Identificador del ↩
            individuo
146     float massvol; // Peso o masa
147     float gviol; // Grado de violación normalizado
148     bool feasible;
149     float maxdef; // Máxima deformación de la estructura
150     vector<float>Ktorsion; // Vector de constantes de torsión
151     vector<float>Tension;  // Vector de tensiones ↩
            equivalentes de cada barra
152     vector<float>Area;
153
154     // Cromosomas
155     GARealGenome *cons; // Vector restricciones móviles real
156     GARealGenome *nod; // Vector nodal real
157     GARealGenome *con; // Vector conectividad binario
158     GARealGenome *mat; // Vector materiales  entero
159     GARealGenome *tsec; // Vector tipo sección entero
160     GARealGenome *sec; // Vector dimensiones sección real o ↩
            entero
161
162 };
163
164 int GAsOPGenome::numelementos = 0;
165 int GAsOPGenome::numnodos = 0;
166 #endif
```

```
1  //GASOPGenome.cpp
2  //Fuente de la Clase GASOPGenome
3
4  #include "GAsOPGenome.h"
5
6  GAsOPGenome::~GAsOPGenome() {
7      delete cons;
8      delete nod;
9      delete con;
10     delete mat;
11     delete tsec;
12     delete sec;
13 };
14
15 GAsOPGenome::GAsOPGenome(const int &numnod,
16     const GARealAlleleSetArray & alcons, const ↩
          GARealAlleleSetArray & alnod,
17     const GARealAlleleSetArray & alcon, const ↩
          GARealAlleleSetArray & almat,
18     const GARealAlleleSetArray & altsec, const ↩
          GARealAlleleSetArray & alsec,
19     GAGenome::Evaluator f) : GAGenome(GAsOPInitializer, ↩
          GAsOPMutator,
20     GAsOPComparator) {
21     evaluator(f);
22     crossover(GAsOPCrossover);
23
24     // Número de barras y nodos de la estructura
25     GAsOPGenome::numelementos = numnod * (numnod - 1) / 2;
26     GAsOPGenome::numnodos = numnod;
27
28     // Creación del vector de restricciones móvies
29     cons = new GARealGenome(alcons);
30     // Creación del vector de coordenadas nodales
31     nod = new GARealGenome(alnod);
32
33     // Creación del vector de conectividad
34     // La longitud depende del número de elementos ↩
          Numelementos
```

```
35      // que depende a su vez del número de nudos
36      con = new GARealGenome(alcon);
37      // Creación del vector de materiales
38      mat = new GARealGenome(almat);
39
40      // Creación del vector de tipos de seccion
41      tsec = new GARealGenome(altsec);
42
43      // Creación del vector de dimensiones de las secciones
44      sec = new GARealGenome(alsec);
45  }
46
47  GAsOPGenome::GAsOPGenome(const GAsOPGenome & orig) { // ↩
        constructor copia
48      cons = new GARealGenome(orig.getcons());
49      nod = new GARealGenome(orig.getnod());
50      con = new GARealGenome(orig.getcon());
51      mat = new GARealGenome(orig.getmat());
52      tsec = new GARealGenome(orig.gettsec());
53      sec = new GARealGenome(orig.getsec());
54      copy(orig);
55  }
56
57  GAsOPGenome& GAsOPGenome:: operator = (const GAGenome & g) {
58      // operador de asignacíón
59      copy(g);
60      return *this;
61  }
62
63  GAGenome* GAsOPGenome::clone(GAGenome::CloneMethod)const { //↩
        constructor clon
64      return new GAsOPGenome(*this);
65  }
66
67  void GAsOPGenome::copy(const GAGenome &c) {
68      if (&c != this && sameClass(c)) {
69          GAGenome::copy(c);
70          GAsOPGenome & bc = (GAsOPGenome&)c;
71
```

```
72          // Copia de los cromosomas (6)
73          cons->copy(*(bc.cons));
74          nod->copy(*(bc.nod));
75          con->copy(*(bc.con));
76          mat->copy(*(bc.mat));
77          tsec->copy(*(bc.tsec));
78          sec->copy(*(bc.sec));
79
80      // Copia de las variables (7) + evaluacion
81          uid = bc.get_uid();
82          feasible = bc.feasible;
83          gviol = bc.gviol; // Grado de violación normalizado
84          massvol = bc.massvol;
85          Ktorsion = bc.Ktorsion;
86          Area = bc.Area;
87          _evaluated = bc._evaluated;
88      }
89  }
90
91  int GAsOPGenome::equal(const GAGenome &g)const {
92      GAsOPGenome & genome = (GAsOPGenome&)g;
93      int comparacion;
94      comparacion = *cons == *genome.cons && *nod == *genome.↩
            nod && *con ==
95      *genome.con && *mat == *genome.mat && *tsec == *genome.↩
            tsec && *sec ==
96      *genome.sec && uid == genome.uid && feasible == genome.↩
            feasible &&
97      gviol == genome.gviol && massvol == genome.massvol && ↩
            Ktorsion ==
98      genome.Ktorsion && Area == genome.Area && _evaluated == ↩
            genome._evaluated;
99
100     return comparacion;
101 }
102
103 void GAsOPGenome::GAsOPInitializer(GAGenome & g) {
104     GAsOPGenome & gen = (GAsOPGenome&)g;
105     GARealGenome::UniformInitializer(gen.getcons());
```

```
106     GARealGenome::UniformInitializer(gen.getnod());
107
108     // Repetimos la inicialización del vector de conexión ↩
            hasta encontrar uno legal
109     do {
110         GARandomSeed();
111         GARealGenome::UniformInitializer(gen.getcon());
112     }
113     while (!gen.check_con());
114     GARealGenome::UniformInitializer(gen.getmat());
115     GARealGenome::UniformInitializer(gen.gettsec());
116
117     // Repetimos la inicialización del vector de geometría ↩
            hasta encontrar uno legal
118     do {
119         GARandomSeed();
120         GARealGenome::UniformInitializer(gen.getsec());
121     }
122     while (!gen.check_geom());
123
124     gen.set_uid();
125     gen._evaluated = true;
126 }
127
128 int GAsOPGenome::GAsOPMutator(GAGenome & g, float pmut) {
129     GAsOPGenome & gen = (GAsOPGenome&)g;
130     int nmut, tnmut;
131     nmut += GARealGaussianMutator(gen.getcons(), pmut);
132     nmut += GARealGaussianMutator(gen.getnod(), pmut);
133
134     do {
135         tnmut = GARealGaussianMutator(gen.getcon(), pmut);
136     }
137     while (!gen.check_con());
138     nmut += tnmut;
139
140     nmut += GARealGaussianMutator(gen.getmat(), pmut);
141     nmut += GARealGaussianMutator(gen.gettsec(), pmut);
142
```

```
143     do {
144         tnmut = GARealGaussianMutator(gen.getsec(), pmut);
145     }
146     while (!gen.check_geom());
147     nmut += tnmut;
148 
149     if (nmut) {
150         gen.set_uid();
151         gen._evaluated = false;
152     }
153     return nmut;
154 }
155 
156 float GAsOPGenome::GAsOPComparator(const GAGenome& p1, const ↩
    GAGenome& p2) {
157     // Comparación entre los alelos de los genomas
158     // retorna 0 si son iguales y un float que representa % ↩
       de diferencia
159     GAsOPGenome & g1 = (GAsOPGenome&)p1;
160     GAsOPGenome & g2 = (GAsOPGenome&)p2;
161     float diff = 0;
162     float nev = 0; // Número de veces que se evalua la ↩
       diferencia
163     float temp; // Registro temporal
164 
165     // Comparación de restricciones móviles:
166     for (int i = 0; i < g1.cons->length(); i++) {
167         temp = g1.genedistance(g1.getcons(), g2.getcons(), i)↩
            ;
168         if (temp > 0) { // No computamos los genes que solo ↩
            pueden tener un valor (sin rango)
169             diff += temp;
170             nev++;
171         }
172     }
173 
174     // Comparación de los nodos móviles
175     for (int i = 0; i < g1.nod->length(); i++) {
176         temp = g1.genedistance(g1.getnod(), g2.getnod(), i);
```

```
177             if (temp > 0) {
178                 diff += temp;
179                 nev++;
180             }
181         }
182 
183         // Comparación de las conexiones
184         for (int i = 0; i < g1.con->length(); i++) {
185             temp = g1.genedistance(g1.getcon(), g2.getcon(), i);
186             if (temp > 0) {
187                 diff += temp;
188                 nev++;
189             }
190         }
191 
192         // Comparación de los materiales
193         for (int i = 0; i < g1.mat->length(); i++) {
194             temp = g1.genedistance(g1.getmat(), g2.getmat(), i);
195             if (temp > 0) {
196                 diff += temp;
197                 nev++;
198             }
199         }
200 
201         // Comparación de los tipos de sección
202         for (int i = 0; i < g1.tsec->length(); i++) {
203             temp = g1.genedistance(g1.gettsec(), g2.gettsec(), i)↩
                    ;
204             if (temp > 0) {
205                 diff += temp;
206                 nev++;
207             }
208         }
209 
210         // La comparación de las secciones se hace por separado ↩
                para eliminar los genes inactivos
211         // de la misma
212         for (int i = 0, index = -1; i < g1.nod->length(); i++, ↩
                index += 10) {
```

```
213         // De la primera a la última barra. index recorre las↩
                fila s
214         if (g1.tsec->gene(i) == g2.tsec->gene(i)) {
215             // Si las secciones son iguales
216             // Si las secciones son iguales continuamos ↩
                    evaluando la distancia
217             // el primer gen está presente en todas las ↩
                    secciones
218             switch((int)g1.tsec->gene(i)) {
219             case 1:
220                 break;
221             case 2,
222                 4 : // {1,2}
223                 temp = g1.genedistance(g1.getsec(), g2.getsec↩
                        (), index + 1);
224                 temp += g1.genedistance(g1.getsec(), g2.↩
                        getsec(), index + 2);
225                 if (temp > 0) {
226                     diff += temp;
227                     nev++;
228                 }
229                 break;
230             case 3: // 1
231                 temp = g1.genedistance(g1.getsec(), g2.getsec↩
                        (), index + 1);
232                 if (temp > 0) {
233                     diff += temp;
234                     nev++;
235                 }
236                 break;
237             case 5,
238                 6, 7 : // {1,2,3,5,6,7}
239                 { // Generamos un ámbito privado para poder ↩
                        formar la matriz
240                     int f[] = {
241                         1, 2, 3, 5, 6, 7
242                     }; // Método para hacer un for each
243                     for (int k = 0; k < 6; k++) {
```

```
244                         temp = g1.genedistance(g1.getsec(), ↩
                               g2.getsec(),
245                             index + *(f + k));
246                         if (temp > 0) {
247                             diff = diff + temp;
248                             nev++;
249                         }
250                     }
251                 }break;
252             case 8,
253                 9 : // {1,2,5,6}
254                 {
255                     int f[4] = {
256                         1, 2, 5, 6
257                     }; // Método para hacer un for each
258                     for (int k = 0; k < 4; k++) {
259                         temp = g1.genedistance(g1.getsec(), ↩
                               g2.getsec(),
260                             index + *(f + k));
261                         if (temp > 0) {
262                             diff = diff + temp;
263                             nev++;
264                         }
265                     }
266                 }break;
267             case 10:
268                 for (int k = 1; k < 10; k++) {
269                     temp = g1.genedistance(g1.getsec(), g2.↩
                           getsec(), index + k);
270                     if (temp > 0) {
271                         diff = diff + temp;
272                         nev++;
273                     }
274                 }
275                 break;
276             case 11: // {1,2,5,6,7,8}
277                 {
278                     int f[5] = {
279                         2, 5, 6, 7, 8
```

```
280                     }; // Método para hacer un for each
281                     for (int k = 0; k < 5; k++) {
282                         temp = g1.genedistance(g1.getsec(), ↩
                               g2.getsec(),
283                              index + *(f + k));
284                         if (temp > 0) {
285                              diff = diff + temp;
286                              nev++;
287                         }
288                     }
289                 }break;
290             }
291         }
292
293         else { // Si las secciones son diferentes
294             diff++;
295             nev++;
296         }
297
298     }
299
300     if (!nev)
301         nev++; // Si los vectores son coincidentes  diff y ↩
                   nev serán cero
302
303     return diff / nev;
304 }
305
306 int GAsOPGenome::GAsOPCrossover(const GAGenome & a, const ↩
    GAGenome & b,
307     GAGenome *c, GAGenome *d) {
308     GAsOPGenome & mom = (GAsOPGenome&)a;
309     GAsOPGenome & dad = (GAsOPGenome&)b;
310     GAsOPGenome & sis = (GAsOPGenome&) * c;
311     GAsOPGenome & bro = (GAsOPGenome&) * d;
312     // Si en lugar de hacer los else if, poneemos if, los ↩
           hijos no son simétricos, es decir, los hijos
313     // se producen en dos uniones diferendes de las ↩
           cualessale un hijo en cada una
```

```
314
315     int n; // Número de hijos generados
316
317     if (c && d) { // Si hay cruce
318         SBXCrossover(mom.getnod(), dad.getnod(), &sis.getnod↩
                ()), &bro.getnod());
319         SBXCrossover(mom.getcons(), dad.getcons(), &sis.↩
                getcons(),
320              &bro.getcons(), 1, 1);
321         do {
322              GARealGenome::UniformCrossover(mom.getcon(), dad.↩
                     getcon(),
323                   &sis.getcon(), &bro.getcon());
324         }
325         while (!sis.check_con() && !bro.check_con());
326
327         GARealGenome::UniformCrossover(mom.getmat(), dad.↩
                getmat(),
328              &sis.getmat(), &bro.getmat());
329         GARealGenome::UniformCrossover(mom.gettsec(), dad.↩
                gettsec(),
330              &sis.gettsec(), &bro.gettsec());
331
332         do {
333              SBXCrossover(mom.getsec(), dad.getsec(), &sis.↩
                     getsec(),
334                   &bro.getsec(), 1, 1);
335         }
336         while (!sis.check_geom() && !bro.check_geom());
337
338         sis.set_uid();
339         sis._evaluated = false;
340         bro.set_uid();
341         bro._evaluated = false;
342         n = 2;
343     }
344
345     else if (c) { // Si duplicamos a la madre
346         GAsOPGenome & sis = (GAsOPGenome&) * c;
```

```
347
348         SBXCrossover(mom.getcons(), dad.getcons(), &sis.↩
                getcons(), 0);
349         SBXCrossover(mom.getnod(), dad.getnod(), &sis.getnod↩
                (), 0);
350
351         do {
352             GARealGenome::UniformCrossover(mom.getcon(), dad.↩
                    getcon(),
353                 &sis.getcon(), 0);
354         }
355         while (!sis.check_con());
356
357         GARealGenome::UniformCrossover(mom.getmat(), dad.↩
                getmat(),
358             &sis.getmat(), 0);
359         GARealGenome::UniformCrossover(mom.gettsec(), dad.↩
                gettsec(),
360             &sis.gettsec(), 0);
361         do {
362             SBXCrossover(mom.getsec(), dad.getsec(), &sis.↩
                    getsec(), 0, 1, 1);
363         }
364         while (!sis.check_geom());
365
366         sis.set_uid();
367         sis._evaluated = false;
368         n = 1;
369     }
370     else if (d) { // Si duplicamos al padre
371         GAsOPGenome & bro = (GAsOPGenome&) * d;
372         SBXCrossover(mom.getcons(), dad.getcons(), 0, &bro.↩
                getcons());
373         SBXCrossover(mom.getnod(), dad.getnod(), 0, &bro.↩
                getnod());
374
375         do { // Operador de reparacion
376             GARealGenome::UniformCrossover(mom.getcon(), dad.↩
                    getcon(), 0,
```

```
377                 &bro.getcon());
378         }
379         while (!bro.check_con());
380 
381         GARealGenome::UniformCrossover(mom.getmat(), dad.↩
                getmat(), 0,
382             &bro.getmat());
383         GARealGenome::UniformCrossover(mom.gettsec(), dad.↩
                gettsec(), 0,
384             &bro.gettsec());
385         do {
386             SBXCrossover(mom.getsec(), dad.getsec(), 0, &bro.↩
                    getsec(), 1, 1);
387         }
388         while (!bro.check_geom());
389 
390         bro.set_uid();
391         bro._evaluated = false;
392         n = 1;
393     }
394 
395     return n; // número de individuos generados
396 }
397 
398 float GAsOPGenome::GAsOPEvaluator(GAGenome & a) {
399 
400 }
401 
402 int GAsOPGenome::write(ostream & os)const {
403   // Escribe en el stream las características del gen
404 
405   os << *cons << *nod << *con << *mat << *tsec << *sec;
406     os << endl;
407     os << "UID individuo: " << get_uid();
408     os << " Fitness: " << score();
409     os << " Mass\\Vol: " << get_massvol();
410     os << " Grado violacion: " << get_gviol();
411     os << endl << endl;
```

```
412     return os.fail() ? 1 : 0; // Devuelve 1 si se produce un ↩
            fallo de escritura
413 }
414
415 int GAsOPGenome::read(istream & is) {
416   // Lectura del cromosoma desde un stream
417
418     char cfitness[100];
419     char cmassvol[100];
420     char cgviol[100];
421     char trash[100];
422     is >> *cons >> *nod >> *con >> *mat >> *tsec >> *sec;
423     // Lectura de los cromosomas
424     is >> trash >> trash >> trash >> trash >> cfitness >> ↩
            trash;
425     is >> cmassvol >> trash >> trash >> cgviol;
426
427     set_uid();
428     score(atof(cfitness));
429     massvol = atof(cmassvol);
430     gviol = atof(cgviol);
431
432     return is.fail() ? 1 : 0; // Devuelve 1 si se produce un ↩
            fallo de lectura
433 }
434
435 void GAsOPGenome::set_uid() {
436     int i, id, id1, id2, id3, id4, id5;
437     id1 = id2 = id3 = id4 = id5 = 0;
438     for (i = 0; i < cons->length(); i++) {
439         id1 += cons->gene(i);
440     }
441
442     for (i = 0; i < nod->length(); i++) {
443         id2 += nod->gene(i);
444     }
445
446     for (i = 0; i < con->length(); i++) {
447         id = con->gene(i);
```

```
448         if (!(i % 2)) {
449             id3 += 3 * id;
450         }
451         else
452             id3 += id;
453     }
454
455     for (i = 0; i < mat->length(); i++) {
456         id = mat->gene(i);
457         if (!(i % 2)) {
458             id4 += 2 * id;
459         }
460         else
461             id4 += id;
462     }
463
464     for (i = 0; i < sec->length(); i++) {
465         id5 += sec->gene(i);
466     }
467
468     uid = IntToStr(id1) + "-" + IntToStr(id2) + "-" + ↩
        IntToStr(id3) + "-";
469     uid += IntToStr(id4) + "-" + IntToStr(id5);
470 }
471
472 bool GAsOPGenome::loadfile(const string &nomarch, int numgen)↩
    {
473   // Lectura de individuos desde un archivo externo
474     int ngen = 1;
475     ifstream lfile(nomarch.c_str());
476     if (!lfile) {
477         cout << "No se puede cargar el archivo." << endl;
478         pause();
479         return false;
480     }
481     // read(lfile);
482     while (!lfile.eof()) {
483         string cad;
484         getline(lfile, cad);
```

```
485         etokens texto(cad);
486         if (texto.tipoparam(0) == 1) {
487             if (ngen == numgen) {
488                 getline(lfile, cad);
489                 etokens texto(cad);
490                 score(StrToF(texto[4]));
491                 StrToF(texto[6]);
492                 StrToF(texto[8]);
493                 uid = get_uid();
494                 return true;
495             }
496             ngen++;
497         }
498
499     }
500 }
501
502 bool GAsOPGenome::check_all() {
503     feasible = check_con() && check_geom();
504     return feasible; // true si existe conexión y la ↩
        geometría es correcta
505 }
506
507 bool GAsOPGenome::check_geom() {
508     // Borramos los datos almacenados
509     Ktorsion.clear();
510     Area.clear();
511
512     // Creacion de las secciones
513       for (int i = 0, index = -1; i < numelementos; i++, ↩
          index += 10) {
514
515         // De la primera a la última barra
516         // index es el número de fila de la matriz. La ↩
            primera fila
517         // es -1 para que el orden den gen coicida con el ↩
            parámetro y facilitar depuración
518         if (con->gene(i)) { // Si hay elemento
519             float p1, p2, p3, p4, p5, p6, p7, p8, p9;
```

```
520             switch((int)tsec->gene(i)) {
521             case 1: //
522                 return false;
523                 break;
524             case 2: // Rectangulo
525                 p1 = sec->gene(index + 1);
526                 p2 = sec->gene(index + 2);
527                 if (!p1 || !p2)
528                     return false; // Si algún parámetro es ↩
                        nulo el individuo es incorrecto
529                 if (p1 > p2)
530                     r.carga(p1, p2);
531                 else
532                     r.carga(p2, p1);
533                 Ktorsion.push_back(r.Ktorsion());
534                 Area.push_back(r.get_area());
535                 break;
536             case 3: // Eje macizo
537                 eje.carga(sec->gene(index + 1));
538                 Ktorsion.push_back(eje.Ktorsion());
539                 Area.push_back(eje.get_area());
540                 break;
541             case 4: // Eje hueco
542                 p1 = sec->gene(index + 1);
543                 p2 = sec->gene(index + 2);
544                 if (p1 >= p2) {
545                     //return false; // Desactivar comentario ↩
                        si no se desea correccion
546                     eje_hueco.carga(p1, p2);
547                     // Correccion de los genes defectuosos
548                     sec->gene(index + 1, p2);
549                     sec->gene(index + 2, p1);
550                 }
551                 else
552                     eje_hueco.carga(p2, p1);
553                 Ktorsion.push_back(eje_hueco.Ktorsion());
554                 Area.push_back(eje_hueco.get_area());
555                 break;
556             case 5: // Perfil en U
```

```
557                 p1 = sec->gene(index + 1); // W1
558                 p2 = sec->gene(index + 5); // t1
559                 p4 = sec->gene(index + 7); // t3
560                 p5 = sec->gene(index + 2); // W2
561                 p6 = sec->gene(index + 6); // t2
562                 p3 = (sec->gene(index + 3) - p2 - p6) / 2; //↩
                        (W3-t1-t2)/2
563                 if (p1 <= p4 || p5 <= p4)
564                     return false;
565                 if (p3 <= 0 || !p1 || !p2 || !p4 || !p5 || !↩
                       p6)
566                     return false;
567                 perfil_u.carga(p1, p2, p3, p4, p5, p6);
568                 Ktorsion.push_back(perfil_u.Ktorsion());
569                 Area.push_back(perfil_u.get_area());
570                 break;
571             case 6: // Perfil en I
572                 p1 = sec->gene(index + 1); // W1
573                 p2 = sec->gene(index + 5); // t1
574                 p4 = sec->gene(index + 7); // t3
575                 p5 = sec->gene(index + 2); // W2
576                 p6 = sec->gene(index + 6); // t2
577                 p3 = (sec->gene(index + 3) - p2 - p6) / 2; //↩
                        (W3-t1-t2)/2
578                 if (p1 <= p4 || p5 <= p4)
579                     return false;
580                 if (p3 <= 0 || !p1 || !p2 || !p4 || !p5 || !↩
                       p6)
581                     return false;
582                 perfil_i.carga(p1, p2, p3, p4, p5, p6);
583                 Ktorsion.push_back(perfil_i.Ktorsion());
584                 Area.push_back(perfil_i.get_area());
585                 break;
586             case 7: // Perfil Z
587                 p1 = sec->gene(index + 1); // W1
588                 p2 = sec->gene(index + 5); // t1
589                 p4 = sec->gene(index + 7); // t3
590                 p5 = sec->gene(index + 2); // W2
591                 p6 = sec->gene(index + 6); // t2
```

```
592                 p3 = (sec->gene(index + 3) - p2 - p6) / 2;
593                 // (W3-t1-t2)/2
594                 if (p1 <= p4 || p5 <= p4)
595                     return false;
596                 if (p3 <= 0 || !p1 || !p2 || !p4 || !p5 || !↩
                        p6)
597                     return false;
598                 perfil_z.carga(p1, p2, p3, p4, p5, p6);
599                 Ktorsion.push_back(perfil_z.Ktorsion());
600                 Area.push_back(perfil_z.get_area());
601                 break;
602             case 8: // Perfil L
603                 p1 = sec->gene(index + 1); // W1
604                 p2 = sec->gene(index + 5); // t1
605                 p3 = sec->gene(index + 2) - p2; // W2-t1
606                 p4 = sec->gene(index + 6); // t2
607                 if (p1 <= p4)
608                     return false;
609                 if (p3 <= 0 || !p1 || !p2 || !p4)
610                     return false;
611                 perfil_l.carga(p1, p2, p3, p4);
612                 Ktorsion.push_back(perfil_l.Ktorsion());
613                 Area.push_back(perfil_l.get_area());
614                 break;
615             case 9: // Perfil T
616                 p1 = sec->gene(index + 1); // W1
617                 p2 = sec->gene(index + 5); // t1
618                 p3 = sec->gene(index + 2) - p2; // W2 - t1
619                 p4 = sec->gene(index + 6); // t2
620                 if (p1 <= p4)
621                     return false;
622                 if (p3 <= 0 || !p1 || !p2 || !p4)
623                     return false;
624                 perfil_t.carga(p1, p2, p3, p4);
625                 Ktorsion.push_back(perfil_t.Ktorsion());
626                 Area.push_back(perfil_t.get_area());
627                 break;
628             case 10: // Perfil omega
629                 p1 = sec->gene(index + 1); // W1
```

```
630                 p2 = sec->gene(index + 5); // t1
631                 p4 = sec->gene(index + 8); // t4
632                 p5 = sec->gene(index + 3) / 2; // W3/2
633                 p6 = sec->gene(index + 7); // t3
634                 p3 = (sec->gene(index + 4) - p2 - p6) / 2;
635                 // (W4-t1-t3)/2
636                 p7 = sec->gene(index + 9); // t5
637                 p8 = sec->gene(index + 2); // W2
638                 p9 = sec->gene(index + 6); // t2
639                 if (p1 <= p4 || p8 <= p7 || 2 * p5 < (p4 + p7↩
                        ))
640                     return false;
641                 if (sec->gene(index + 4) <= p9 + p6)
642                     return false;
643                 if (p3 <= 0 || !p1 || !p2 || !p4 || !p5 || !↩
                        p6 || !p7 || !p8 ||
644                     !p9)
645                     return false;
646                 perfil_omega.carga(p1, p2, p3, p4, p5, p6, p7↩
                        , p8, p9);
647                 Ktorsion.push_back(perfil_omega.Ktorsion());
648                 Area.push_back(perfil_omega.get_area());
649                 break;
650             case 11: // Perfil rectangular hueco
651                 p1 = sec->gene(index + 1); // W1
652                 p2 = sec->gene(index + 2); // W2
653                 p3 = sec->gene(index + 5); // t1
654                 p4 = sec->gene(index + 6); // t2
655                 p5 = sec->gene(index + 7); // t3
656                 p6 = sec->gene(index + 8); // t4
657                 if (p1 <= (p3 + p4) || p2 <= (p5 + p6))
658                     return false;
659                 if (p1 <= 0 || p2 <= 0 || p3 <= 0 || p4 <= 0 ↩
                        || p5 <= 0 ||
660                     p6 <= 0)
661                     return false;
662                 r_hueco.carga(p1, p2, p3, p4, p5, p6);
663                 Ktorsion.push_back(r_hueco.Ktorsion());
664                 Area.push_back(r_hueco.get_area());
```

```
665                 break;
666             }
667         }
668     }
669     return true;
670 }
671
672 bool GAsOPGenome::check_con() { // true: conectividad , false: sin conectividad
673     bool *check_flag = new bool[numnodos];
674
675     bool checkflag = true;
676     int index = 0; //
677     int noelvector; // Número de elemento del vector
678     int maxnode = numnodos - nod->length() / 3;
679     // maxnode es el numero de nodo máximo restringido o con carga
680
681     // Inicialización del vector de conectividad
682     for (int i = 0; i < numnodos; i++)
683         check_flag[i] = false; // Ponemos a cero todos los elementos
684
685     for (int j = 0; j < numnodos; j++) { // Columnas
686           for (int i = 0; i < j; i++) { // Nos movemos por la columna
687             if (!check_flag[i]) {
688                 if (con->gene(noelvector)) {
689                     check_flag[j] = true;
690                     check_flag[i] = true; // Existe conexión entre i , j
691                 }
692             }
693             noelvector += numnodos - i - 2; // Salto sobre columna
694         }
695         noelvector++;
696
```

```
697         for (int k = j + 1; k < numnodos; k++) { // Nos ↩
                movemos por la fila
698             if (!check_flag[k]) {
699                 if (con->gene(noelvector)) {
700                     check_flag[j] = true;
701                     check_flag[k] = true; // Existe conexión ↩
                            entre j , k
702                 }
703             }
704             noelvector++;
705         }
706     }
707
708     // Comprobamos que todos los elementos estén conectados ↩
            entre sí
709     for (int i = 0; i < maxnode; i++)
710         if (!check_flag[i]) {
711             checkflag = false;
712         }
713     delete[] check_flag;
714
715     return checkflag;
716
717 }
718
719 float GAsOPGenome::genedistance(const GAGenome& g1, const ↩
        GAGenome& g2,
720     int ngen) {
721     const GA1DArrayAlleleGenome<float>&p1 = DYN_CAST
722         (const GA1DArrayAlleleGenome<float>&, g1);
723     const GA1DArrayAlleleGenome<float>&p2 = DYN_CAST
724         (const GA1DArrayAlleleGenome<float>&, g2);
725
726     // Retorna la distancia de Hamming normalizada. Si a se ↩
            situa en c, y b en d están a la máxima distancia
727     // y su valor normalizado será 1. Si los genes si son ↩
            diferentes tomará el valor de la distancia de Hamming,
728     // 0  si son iguales y -1 si el rango es nulo
729     float distancia;
```

```
730     float a, b, c, d;
731     a = p1.gene(ngen);
732     b = p2.gene(ngen);
733     c = p1.alleleset(ngen).lower(); // valor mínimo del gen
734     d = p1.alleleset(ngen).upper(); // valor máximo del gen
735
736     if (abs(d - c)) { // Si el rango no es nulo
737         distancia = abs(a - b) / abs(d - c);
738     }
739     else // si el rango es nulo
740         distancia = -1; // Anotamos un rango nulo
741     return distancia; // Restamos el número de rangos nulos ↩
        para no desvirtuar la medida
742 }
743
744 void GAsOPGenome::write_conection_array(ostream & os)const {
745
746     os << "Vector de conectividad" << endl;
747     getcon().write(os);
748     os << endl << endl;
749
750     os << "Matriz de conectividad posterior" << endl;
751     int ncols = numnodos - 1;
752     int numcon, lastnode = 0;
753     os << "   ";
754     for (int i = 0; i < numnodos; i++) {
755         os << i + 1;
756     }
757     os << endl;
758
759     for (int i = 0, k = 1; i < ncols; i++, k++) {
760         numcon = 0;
761         os << k << string(k + 1, ' ');
762         for (int j = 0; j < ncols - i; j++) {
763             if (getcon().gene(lastnode)) {
764                 numcon++;
765             };
766             os << getcon().gene(lastnode++);
767         }
```

```
768             os << "   " << numcon << endl;
769         }
770         os << numnodos << endl << endl;
771 
772         os << "Matriz de conectividad anterior (traspuesta)" << endl;
773         ncols = numnodos - 1;
774         os << "   ";
775         for (int i = 0; i < numnodos; i++) {
776             os << i + 1;
777         }
778         os << endl;
779         os << "1 " << endl;
780         for (int i = 1, k = 2; i < numnodos; i++, k++) {
781             os << k << " ";
782             lastnode = i - 1;
783             numcon = 0;
784             for (int j = 0; j < i; j++) {
785 
786                 if (getcon().gene(lastnode)) {
787                     numcon++;
788                 };
789                 os << getcon().gene(lastnode);
790                 lastnode += ncols - j - 1;
791             }
792             os << string(numnodos - k + 3, ' ') << numcon << endl;
793         }
794         os << endl << endl;
795 
796         os << "Matriz de conectividad completa" << endl;
797         bool *check_flag = new bool[numnodos];
798 
799         // Inicialización del vector de conectividad
800         for (int i = 0; i < numnodos; i++)
801             check_flag[i] = false; // Ponemos a cero todos los elementos
802         for (int j = 0; j < numnodos; j++) { // Columnas
803             lastnode = j - 1;
```

```
804         for (int i = 0; i < j; i++) { // Nos movemos por la ↩
                columna
805             os << getcon().gene(lastnode);
806             lastnode += numnodos - i - 2; // Salto sobre ↩
                    columna
807         }
808         os << " ";
809         lastnode++;
810         for (int k = j + 1; k < numnodos; k++) { // Nos ↩
                movemos por la fila
811             os << getcon().gene(lastnode);
812             lastnode++;
813         }
814         os << endl;
815     }
816 }
817
818 bool GAsOPGenome::check_genome(ostream & os)const {
819     int i, k;
820     os << "Cromosoma de nodos fijos y semifijos" << endl;
821     os << *cons << endl;
822     os << "Cromosoma de nodos libres" << endl;
823     os << *nod << endl;
824     os << "Cromosoma de conectividad" << endl;
825     os << *con << endl;
826     os << "Cromosoma de materiales" << endl;
827     os << *mat << endl;
828     os << "Cromosoma de tipos de sección" << endl;
829     os << *tsec << endl;
830     os << "Cromosoma de secciones" << endl;
831     for (i = 0, k = 9; i < sec->length(); i++) {
832         os << sec->gene(i) << " ";
833         if (i == k) {
834             k += 10;
835             os << endl;
836         }
837     }
838     os << endl;
839     os << "UID individuo: " << get_uid();
```

```
840     os << " Fitness: " << score();
841     os << " Mass\\Vol: " << get_massvol();
842     os << " Grado violacion: " << get_gviol();
843     os << endl << endl;
844     return os.fail() ? 1 : 0; // Devuelve 1 si se produce un ↩
            fallo de escritura
845 }
846
847 bool GAsOPGenome::write_tension(ostream & os)const {
848
849     vector<float>::iterator itr;
850     int i = 1;
851     vector<float> & tempTension =  const_cast<vector<float> ↩
            &>(Tension);
852
853     os << "Tensión en las barras:" << endl;
854     for (itr = tempTension.begin(); itr != tempTension.end();↩
            itr++) {
855         os << "Tension en la barra " << i++ << ": " << *itr ↩
                << " MPa." << endl;
856     }
857     os << endl;
858     return os.fail() ? 1 : 0;
859 };
860
861
862 bool GAsOPGenome::write_Ktorsion(ostream & os)const {
863
864     vector<float>::iterator itr;
865     int i = 1;
866     vector<float> & tempK =  const_cast<vector<float> &>(↩
            Ktorsion);
867
868     os << "Constantes de torsión de las barras:" << endl;
869     for (itr = tempK.begin(); itr != tempK.end(); itr++) {
870         os << "Constante de torsión de la barra " << i++ << "↩
                : " << *itr << endl;
871     }
872     os << endl;
```

```
873     return os.fail() ? 1 : 0;
874 };
```