Computación Evolutiva

Carlos Artemio Coello Coello
Editor

ACADEMIA MEXICANA DE COMPUTACIÓN, A, C.

Computación Evolutiva
Editor: Carlos Artemio Coello Coello.

En colaboración con la Academia Mexicana de Computación:
Coordinador: Luis Enrique Sucar Succar.

Segunda Edición 2019
Academia Mexicana de Computación, A.C.

ISBN: 978-607-97357-9-1

Colaboradores capítulo 1: Carlos A. Coello Coello
Colaboradores capítulo 2: Rafael Rivera López y Efrén Mezura Montes
Colaboradores capítulo 3: Alicia Morales Reyes
Colaboradores capítulo 4: Mario Graff, Daniela Moctezuma, Eric S. Tellez y Sabino Miranda Jiménez
Colaboradores capítulo 5: Carlos Segura y Joel Chacón Castillo
Colaboradores capítulo 6: Carlos A. Brizuela y Julio Juárez
Colaboradores capítulo 7: Hugo Terashima Marín, Santiago Enrique Conant Pablos, José Carlos Ortiz Bayliss e Iván Mauricio Amaya Contreras

Este libro se realizó con el apoyo del CONACyT, Proyecto I1200/28/2019.

Impreso en México.

Computación Evolutiva

Autores:
Amaya Contreras, Iván Mauricio
Brizuela, Carlos A.
Chacón Castillo, Joel
Coello Coello, Carlos Artemio
Conant Pablos, Santiago Enrique
Graff, Mario
Juárez, Julio
Mezura Montes, Efrén
Miranda Jiménez, Sabino
Moctezuma, Daniela
Morales Reyes, Alicia
Ortiz Bayliss, José Carlos
Rivera López, Rafael
Segura, Carlos
Tellez, Eric S.
Terashima Marín, Hugo

Prólogo

La computación evolutiva es una rama de la inteligencia artificial en la cual se simula la evolución natural (basada en el principio de la "supervivencia del más apto" de Charles Darwin) para resolver problemas complejos de búsqueda, optimización y aprendizaje.

En México existe un número importante de investigadores trabajando en diferentes áreas de la computación evolutiva y este libro intenta proporcionar un asomo a la diversidad de temas en los que se hace investigación en nuestro país. Los siete capítulos que conforman este libro están escritos como tutoriales breves que pretender introducir al lector a ciertos temas específicos de computación evolutiva. En todos los casos, se describe brevemente parte del trabajo que han realizado algunos grupos de investigación en México en torno a estos temas.

El capítulo 1 presenta una introducción general a la computación evolutiva, abarcando sus antecendentes históricos, así como una breve descripción de sus tres paradigmas principales (las estrategias evolutivas, la programación evolutiva y los algoritmos genéticos). También se incluye una pequeña sección en la que se describen brevemente otras metaheurísticas bio-inspiradas (cúmulos de partículas, sistemas

inmunes artificiales, evolución diferencial y colonia de hormigas), así como un recuento suscinto de la investigación que se ha realizado y se realiza actualmente en México en torno a estos temas.

El capítulo 2 presenta una introducción a la "evolución diferencial" que es un tipo de algoritmo evolutivo que se ha vuelto muy popular en la solución de problemas de optimización no lineal. Los autores proporcionan una descripción general de este tipo de algoritmo evolutivo, incluyendo el funcionamiento de sus operadores de cruza y mutación y las diferencias entre sus variantes principales.

El capítulo 3 describe los conceptos básicos de los "algoritmos genéticos paralelos", a partir de una taxonomía muy sencilla, basada en la granularidad de la paralelización, enfatizando las diferencias que existen entre los esquemas homogéneos y heterogéneos, así como entre el paralelismo síncrono y el asíncrono.

El capítulo 4 se enfoca a una variante del algoritmo genético denominada "programación genética" en la cual se usa una representación de árbol (la cual permite evolucionar programas) en vez de las cadenas binarias de longitud fija del algoritmo genético tradicional. La programación genética ha tomado una gran relevancia en años recientes, sobre todo por su aplicabilidad a problemas de aprendizaje y clasificación.

El capítulo 5 se enfoca en un tema muy importante en computación evolutiva, del cual no suele hablarse mucho: la importancia de la diversidad en el diseño de un algoritmo evolutivo. Los autores hacen una revisión de diversas técnicas que se han propuesto para lidiar con la convergencia prematura (la cual suele ser consecuencia de una

pérdida abrupta de diversidad en la población). Adicionalmente, se incluye un pequeño estudio en el que se analizan dos técnicas para mantener diversidad.

El capítulo 6 presenta un tutorial de otra área que ha tomado gran relevancia en años recientes: la *optimización evolutiva multi-objetivo*, la cual se enfoca a la solución de problemas de optimización que tienen dos o más objetivos (normalmente en conflicto entre sí) que deseamos optimizar al mismo tiempo. Además de proporcionar los conceptos básicos de esta área, se describen brevemente los algoritmos evolutivos multi-objetivo más utilizados en la actualidad, así como las medidas principales de desempeño que se utilizan en la literatura especializada. En la parte final del capítulo se describen también algunos de los temas de investigación en los que se está trabajando más dentro de esta área en la actualidad.

El capítulo 7 está dedicado a las *hiper-heurísticas*, que son técnicas que buscan proporcionar más flexibilidad que los algoritmos evolutivos tradicionales al ser capaces de adaptarse a diversos tipos de problemas y a diferentes instancias de una misma clase de problemas. Las hiper-heurísticas son el eslabón intermedio entre las limitantes teóricas relacionadas con la aplicabilidad de las metaheurísticas y el anhelado sueño del diseño automatizado de algoritmos y, constituyen, sin lugar a dudas, uno de los temas más ricos de investigación dentro de las metaheurísticas.

Carlos Artemio Coello Coello

Abreviaturas

AE	Algoritmo Evolutivo
AG	Algoritmo Genético
AGc	Algoritmo Genético Celular
AGd	Algoritmo Genético Distribuido
AGP	Algoritmo Genético Paralelo
EE	Estrategia Evolutiva
PE	Programación Evolutiva
PG	Programación Genética

Contenido

Capítulo 1

Introducción a la Computación Evolutiva

La computación evolutiva es un área de las ciencias de la computación que se enfoca al estudio de las propiedades de una serie de metaheurísticas estocásticas (a las cuales se les denomina "Algoritmos Evolutivos") inspiradas en la teoría de la evolución de las especies formulada por Charles Darwin, según la cual los individuos "más aptos" a su ambiente tienen una mayor probabilidad de sobrevivir.

Una metaheurística es un procedimiento de alto nivel que aplica una regla o conjunto de reglas que se basa(n) en una fuente de conocimiento. El énfasis de las metaheurísticas es explorar el espacio de búsqueda de manera relativamente eficiente; es decir, mejor que una búsqueda exhaustiva, aunque no necesariamente óptima. Las metaheurísticas suelen considerarse también como técnicas de búsqueda y optimización de uso general, pese a sus restricciones teóricas [492].

Además, suelen requerir de poca información específica del problema y son útiles cuando el espacio de búsqueda es muy grande, poco conocido y/o con dificultades para explorarlo. Cuando se usan para optimización, no pueden garantizar, en general, que convergerán a la mejor solución posible (es decir, al óptimo global del problema), si bien en la práctica suelen producir aproximaciones razonablemente buenas en tiempos razonablemente cortos.

Los Algoritmos Evolutivos (AEs) son metaheurísticas estocásticas porque su comportamiento se basa en el uso de números aleatorios y, por lo tanto, no necesariamente generan el mismo resultado cada vez que se ejecutan. Los AEs se han aplicado exitosamente a diversos tipos de problemas [155]. A lo largo de los años, se han desarrollado al menos 2 tipos de AEs con base en el tipo de problemas que pretenden resolver: (1) aquellos diseñados para resolver problemas de optimización (sobre todo no lineal y combinatoria) y (2) aquellos diseñados para resolver problemas de aprendizaje de máquina (sobre todo de clasificación). Sin embargo, la investigación en computación evolutiva no se ha limitado al desarrollo de nuevos algoritmos, sino también al diseño de nuevos operadores, mecanismos de selección y de representación, así como al modelado matemático de los AEs, el estudio de las fuentes de dificultad en un espacio de búsqueda y su correlación con los operadores de un AE y la hibridación de los AEs con otro tipo de técnicas.

El resto de este capítulo está organizado de la siguiente manera. Primero se revisan los antecedentes históricos de los AEs. Posteriormente, se describen los tres principales tipos de AEs que existen

(las estrategias evolutivas, la programación evolutiva y los algoritmos genéticos), así como una variante muy popular de los algoritmos genéticos conocida como "programación genética". En dicha descripción se incluyen algunas aplicaciones relevantes, así como una breve discusión de trabajos de investigación que se han realizado en México. En la parte final del capítulo se mencionan otras metaheurísticas bio-inspiradas con las que actualmente se trabaja en nuestro país. También se proporciona una breve discusión sobre los orígenes de esta área de investigación en México y se concluye con una breve discusión de sus perspectivas futuras.

1.1 Antecedentes Históricos

La computación evolutiva tiene una historia larga y fascinante. En sus orígenes, varios investigadores propusieron de manera totalmente independiente diversos modelos computacionales con objetivos específicos, los cuales tenían en común su inspiración en la evolución natural. Con el tiempo, las evidentes similitudes entre estas técnicas llevaron a acuñar el término genérico "Algoritmo Evolutivo" en la década de los noventa del siglo pasado [156]. En esta sección, se discuten brevemente los acontecimientos más importantes de su historia. Cabe indicar que muchas de las bases de la computación evolutiva se desarrollaron en los sesenta y los setenta, cuando se propusieron los tres enfoques más populares dentro de la computación evolutiva: i) la programación evolutiva, ii) las estrategias evolutivas y iii) los algoritmos genéticos. Si bien los tres enfoques se inspiraron en las mismas ideas, difieren en

sus detalles, como el esquema para representar soluciones y el tipo de operadores adoptados. Adicinalmente se discuten otras propuestas que se consideran importantes por su valor histórico, pese a que su uso no llegó a popularizarse.

Wright [493] y Cannon [68] fueron los primeros investigadores que vieron a la evolución natural como un proceso de aprendizaje en el cual la información genética de las especies se cambia continuamente a través de un proceso de ensayo y error. De esta forma, la evolución puede verse como un método cuyo objetivo es maximizar la capacidad de adaptación de las especies en su ambiente. Wright fue más lejos e introdujo el concepto de "paisaje de aptitud" como una forma de representar el espacio de todos los genotipos posibles junto con sus valores de aptitud. Este concepto se ha utilizado extensamente en computación evolutiva para estudiar el desempeño de los AEs [384]. Adicionalmente, Wright también desarrolló uno de los primeros estudios que relacionan a la selección con la mutación, y concluyó que éstas deben tener un balance adecuado para tener éxito en el proceso evolutivo. Las ideas de Wright se extendieron por Campbell [67], quien afirmaba que el proceso de variación ciega combinado con un proceso de selección adecuado constituye el principio fundamental de la evolución.

Turing puede ser considerado también uno de los pioneros de la computación evolutiva, pues reconoció una conexión (que a él le parecía obvia) entre la evolución y el aprendizaje de máquina [458]. Específicamente, Turing afirmaba que para poder desarrollar una computadora capaz de pasar la famosa prueba que lleva su nombre (con lo

cual sería considerada "inteligente"), podría requerirse un esquema de aprendizaje basado en la evolución natural. Turing ya había sugerido antes que este tipo de métodos basados en la evolución podrían usarse para entrenar un conjunto de redes análogas a lo que hoy conocemos como "redes neuronales" [457]. De hecho, Turing incluso usó el término "búsqueda genética" para referirse a este tipo de esquemas. Sin embargo, debido a que su supervisor consideró que este documento lucía más como un ensayo juvenil que como una propuesta científica [64] el trabajo se publicó hasta 1968 [144], cuando existían ya algunos algoritmos evolutivos y las ideas de Turing al respecto acabaron por ser ignoradas.

Hacia finales de los cincuenta y principios de los sesenta se realizaron varios avances relevantes que coincidieron con el período en que las computadoras digitales se volvieron más accesibles. Durante dicho período, se llevaron a cabo varios análisis de la dinámica poblacional que aparece durante la evolución. Una revisión del estado del arte en torno a este tema se presentó por Crosby en 1967 [102], quien identificó tres tipos de esquemas:

1. Métodos basados en el estudio de formulaciones matemáticas que emergen en el análisis determinista de la dinámica poblacional. Estos esquemas usualmente presuponen un tamaño de población infinito y analizan las distribuciones de los genes[1] bajo distintas circunstancias.

[1]Un **gene** es una secuencia de ácido desoxirribonucleico (ADN) que ocupa una posición específica en un cromosoma. A su vez, un **cromosoma** es una estructura dentro del núcleo de una célula que contiene el material genético que conforma a un individuo.

2. Enfoques que simulan la evolución pero que no representan explícitamente una población [101]. En estos métodos, las frecuencias de los diferentes genotipos[2] potenciales se almacenan para cada generación, resultando en un esquema no escalable en términos del número de genes. Adicionalmente, se requiere una representación matemática del sistema evolutivo.

3. Esquemas que simulan la evolución que mantienen una representanción explícita de la población. Fraser [164] fue pionero en este tipo de métodos, los cuales se describen en una serie de artículos publicados a lo largo de una década [165]. Las conclusiones principales que se derivaron de dichos estudios se reúnen en un libro seminal sobre el tema [163]. Desde su creación, este modelo se aceptó ampliamente y se adoptó por varios investigadores. De hecho, aunque existen varios detalles de implementación en los que las simulaciones de Fraser difieren de los algoritmos evolutivos modernos, algunos autores consideran al trabajo de Fraser como la primera propuesta de un algoritmo genético [156]. Cabe mencionar que aunque a nivel de la implementación la diferencia principal entre la propuesta de Fraser y el algoritmo genético moderno radica en que la primera usaba cromosomas diploides (es decir, un par de cromosomas por individuo) mientras que el segundo usa cromosomas haploides (es decir, un solo cromosoma por individuo), conceptualmente, el trabajo de Fraser se centró

[2]Se llama **genotipo** a la información genética de un organismo. Esta información (contenida en los cromosomas del organismo) puede o no ser manifestada u observada en el individuo.

en analizar la dinámica poblacional y no en resolver problemas, que es la orientación típica de los algoritmos evolutivos.

En este mismo período otros investigadores propusieron esquemas similares a un algoritmo evolutivo. De entre ellos, destacan Friedman, Box y Friedberg por el impacto que su trabajo tuvo posteriormente. Friedman [168] propuso el diseño automático de circuitos de control usando un esquema inspirado en la evolución natural, al que denominó "retroalimentación selectiva." Su propuesta era muy diferente a los algoritmos evolutivos modernos. Por ejemplo, no manejaba la noción de población[3] ni de generación.[4] Además, no implementó su método y planteó ciertas presuposiciones que resultaron demasiado optimistas. No obstante, su trabajo se considera pionero dentro de un área que hoy se conoce como "hardware evolutivo", en la que se busca diseñar circuitos con algoritmos evolutivos (en los que se adopta frecuentemente implementaciones en hardware).

Box [50] propuso una técnica a la que denominó "Operación Evolutiva" (*Evolutionary Operation*, o EVOP) para optimizar el manejo de procesos de la industria química. EVOP usa una solución base (el "padre") para generar varias soluciones adicionales (los "hijos"), modificando unos cuantos parámetros de producción a la vez. Posteriormente, un individuo se selecciona para pasar a la siguiente generación, a partir de cierta evidencia estadística. Este sistema no era autónomo,

[3]La mayor parte de los algoritmos evolutivos modernos operan sobre un conjunto de soluciones potenciales al problema que están resolviendo. A dicho conjunto de soluciones se le denomina "población."

[4]Los algoritmos evolutivos se ejecutan durante un cierto número de iteraciones, a las cuales se les denomina "generaciones."

pues requería de asistencia humana en los procesos de variación y selección. De tal forma, EVOP puede considerarse como el primer algoritmo evolutivo interactivo.

La contribución más importante de Friedberg [166] fue su intento de generar automáticamente programas de computadora mediante un esquema evolutivo. En sus primeros trabajos [167], no identificó ninguna relación específica entre su propuesta y la evolución natural; sin embargo, dicha relación se identificó en publicaciones posteriores de sus co-autores [127]. En sus primeros intentos, el objetivo era generar automáticamente programas pequeños con algunas funcionalidades simples basadas en un conjunto de instrucciones hechas a la medida. Para poder dirigir la búsqueda a regiones prometedoras del espacio de búsqueda, se incorporaba información dependiente del problema, mediante la definción de un esquema de variación diseñado específicamente para el problema a resolverse. Pese al uso de operadores especializados, el éxito de esta técnica fue limitado. Sin embargo, algunos de los análisis realizados por Friedberg y sus co-autores fueron particularmente importantes para algunos de los logros posteriores de la computación evolutiva. Entre sus contribuciones más importantes, destacan las siguientes:

- La idea de dividir las soluciones candidatas en clases es precursora de las nociones de paralelismo implícito[5] y de esquema[6] que propusiera años después John Holland [214].

- Se probaron varios conjuntos de instrucciones, mostrándose, por primera vez, la influencia del mapeo del genotipo al fenotipo.

- Se utilizó un algoritmo de asignación de crédito para medir la influencia de las instrucciones, de manera aislada. Esta idea está muy relacionada con el trabajo que realizara años después Holland con los algoritmos genéticos y los sistemas de clasificadores.

Hay varios artículos más que no llamaron mucho la atención cuando se publicaron originalmente, pero que propusieron técnicas que se reinventaron varios años después. Por ejemplo, el trabajo de Reed, Toombs y Barricelli [381] proporcionó varias innovaciones que se asemejan mucho a los conceptos de auto-adaptación,[7] cruza[8] y coevolución[9] que se usan hoy en día en computación evolutiva.

[5]El "paralelismo implícito" es una propiedad atribuída a los algoritmos genéticos, la cual les permite explorar más eficientemente el espacio de búsqueda mediante un proceso de estimación de la calidad de las soluciones.

[6]Un "esquema" es una abstracción usada para representar las soluciones que procesa un algoritmo genético. Holland usó este concepto para analizar matemáticamente la forma en la que opera un algoritmo genético.

[7]La "auto-adaptación" es un mecanismo que permite que un algoritmo evolutivo adapte por sí mismo los valores de sus parámetros.

[8]La "cruza" es un operador utilizado en los algoritmos evolutivos para hacer que dos individuos intercambien material genético y produzcan descendientes (o "hijos") que combine dicho material.

[9]La "coevolución" es un modelo en computación evolutiva en el cual se usan dos poblaciones que interactúan entre sí de tal forma que la aptitud de los miembros de una de ellas depende de la otra población y vice versa. En otras palabras, en este modelo, la evolución de una especie está condicionada a la de otra especie.

1.2 Programación Evolutiva

Este tipo de algoritmo evolutivo se propuso originalmente por Lawrence Fogel a principios de los sesenta, cuando realizaba investigación básica en torno a inteligencia artificial para la *National Science Foundation*, en Estados Unidos [159]. En esa época, la mayor parte de los intentos existentes para generar comportamientos inteligentes usaban al hombre como modelo. Sin embargo, Fogel consideró que, puesto que la evolución fue capaz de crear a los humanos y a otras criaturas inteligentes, su simulación podría conducir a comportamientos inteligentes [160].

Las ideas básicas de Fogel eran similares a las de algunos trabajos que describimos anteriormente: usar mecanismos de selección y variación para evolucionar soluciones candidatas que se adaptaran mejor a las metas deseadas. Sin embargo, dado que Fogel desconocía dichos trabajos, su propuesta puede considerarse una re-invención de los esquemas basados en la evolución. Filosóficamente, las estructuras de codificación utilizadas en la programación evolutiva son una abstracción del fenotipo[10] de diferentes especies [154]. Consecuentemente, la codificación de soluciones candidatas se puede adaptar libremente para satisfacer los requerimientos del problema (es decir, no se requiere codificar las soluciones en un alfabeto específico como ocurre, por ejemplo, con el Algoritmo Genético, en donde las soluciones normalmente se codifican usando un alfabeto binario). Dado

[10]El "fenotipo" es un término que se usa en genética para denotar los rasgos físicos visibles de un individuo. En algoritmos evolutivos, el fenotipo se refiere al valor decodificado de una solución candidata.

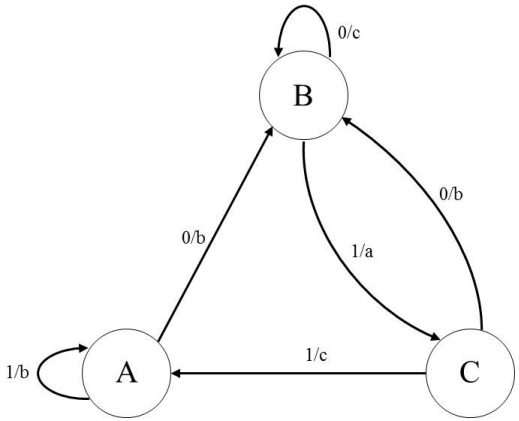


Figura 1.1: Máquina de estados finitos como las usadas por Lawrence Fogel en sus primeros experimentos con la Programación Evolutiva. Los símbolos a la izquierda y a la derecha del "/" son de entrada y de salida respectivamente. El estado inicial en este caso es el **C**.

que no es posible la recombinación sexual entre especies diferentes, la Programación Evolutiva no incorpora un operador de cruza o recombinación. La ausencia del operador de cruza, la libertad de adaptar la codificación y el uso de un operador de selección probabilístico[11] (el cual no se incorporó en las primeras versiones del algoritmo) son los rasgos principales que distinguen a la programación evolutiva de los demás algoritmos evolutivos.

En sus primeras versiones de la Programación Evolutiva, Fogel se percató de que uno de los principales rasgos que caracterizan el comportamiento inteligente es la capacidad de predecir el ambiente, junto con un mecanismo que permita traducir las predicciones a una respuesta adecuada de manera que se logre el objetivo deseado. Por otra

[11]El uso de un operador de selección probabilístico implica que los individuos menos aptos de la población tienen una probabilidad distinta de cero de ser seleccionados.

Estado Actual	**C**	**B**	**C**	**A**	**A**	**B**
Símbolo de entrada	0	1	1	1	0	1
Símbolo siguiente	B	C	A	A	B	C
Símbolo de salida	b	a	c	b	b	a

Tabla 1.1: Esta tabla ilustra las transiciones de la máquina de estados finitos que se muestra en la figura 1.1.

parte, un transductor de estados finitos es una máquina que permite transformar una secuencia de símbolos de entrada en una secuencia de símbolos de salida, como la que se ilustra en la figura 1.1. Dependiendo de los estados y transiciones, esta máquina se puede usar para modelar diferentes situaciones y producir diferentes transformaciones. Por tanto, Fogel la consideró como un mecanismo adecuado para lidiar con el problema de generar comportamiento inteligente. En sus primeros experimentos, su objetivo fue desarrollar una máquina capaz de predecir el comportamiento de un ambiente, el cual se representaba como una secuencia de símbolos de un alfabeto de entrada; es decir, dados los símbolos generados previamente por el ambiente, predecir el siguiente símbolo que emergerá del ambiente. Pensemos, por ejemplo, que queremos obtener un transductor de estados finitos como el que se muestra en la figura 1.1. Para este efecto se requiere crear una tabla de transiciones como la que se muestra en la tabla 1.1. En ésta se indica, para cada estado, el sigiente estado según la entrada, y el símbolo de salida. Fogel intentó generar estas máquinas de manera automatizada (simulando la evolución), usando como entrada los símbolos de salida de la tabla de transiciones. El objetivo era que la Programación Evolutiva produjera una máquina de estados finitos que generara los símbolos de salida deseados, sin intervención humana.

El algoritmo original de Programación Evolutiva funcionaba de la manera siguiente: Primero, se producía de manera aleatoria una población con N máquinas de estados finitos. Posteriormente, cada miembro de la población era mutado, creándose una población de hijos del mismo tamaño de la de los padres. Fogel adoptó cinco tipos de operadores de mutación: (1) agregar un estado, (2) borrar un estado, (3) cambiar el estado siguiente de una transición, (4) cambiar el símbolo de salida de una transición o (5) cambiar el estado inicial. Los operadores de mutación se elegían aleatoriamente[12] y, en algunos casos, los hijos se mutaban más de una vez. Finalmente, los hijos eran evaluados y se seleccionaban las N mejores máquinas (de entre los padres e hijos). Cada máquina de estados finitos era evaluada en términos de su capacidad para predecir correctamente los siguientes símbolos en las secuencias conocidas. Inicialmente, la aptitud[13] se calculaba considerando un número pequeño de símbolos, pero conforme la evolución progresaba, se agregaban más símbolos al conjunto de entrenamiento. En sus primeros experimentos, Fogel realizó pruebas con diferentes tipos de tareas de predicción: secuencias periódicas de números, secuencias con ruido, ambientes no estacionarios, etc. Posteriormente, consideró tareas más complejas como el reconocimiento de patrones, la clasificación y el diseño de sistemas de control. Todo este trabajo originó el que se considera el primer libro de computación

[12]Fogel experimentó con diversos esquemas para elegir el operador de mutación más adecuado, pero al final, decidió adoptar un método de selección aleatorio.

[13]La "aptitud" de un individuo es una medida que permite comparar una solución con respecto a las demás. Si el problema a resolverse es de optimización, normalmente la aptitud se define en términos de la función objetivo que se desea optimizar.

evolutiva de la historia [161]. Este trabajo fue muy importante porque presenta, entre otras cosas, una de las primeras aplicaciones de la coevolución. También es importante enfatizar que en la mayor parte de los primeros estudios en torno a la Programación Evolutiva, los resultados computacionales presentados no eran muy amplios, debido al limitado poder de cómputo de la época. La mayor parte de estos experimentos iniciales fueron recapitulados y extendidos varios años después, proporcionando una mayor comprensión de los verdaderos alcances de la Programación Evolutiva [161].

En los años setenta la mayor parte de la investigación en torno a la Programación Evolutiva se realizó bajo la tutela de Dearholt. Una de las principales contribuciones de su trabajo fue la aplicación de Programación Evolutiva a problemas prácticos. Por ejemplo, al reconocimiento de patrones en expresiones regulares [278] y en caracteres manuscritos [98], así como para clasificar diferentes tipos de electrocardiogramas [111]. Estos trabajos incorporaron varias novedades algorítmicas. De entre ellas, tal vez las más importantes hayan sido el uso simultáneo de varios operadores de mutación y la adaptación dinámica de las probabilidades asociadas con los diferentes esquemas de mutación.

A principios de los ochenta, la Programación Evolutiva se diversificó usando otras representaciones arbitrarias de las soluciones candidatas, a fin de resolver diferentes tipos de problemas. Eventualmente se utilizó, por ejemplo, para resolver problemas de ruteo mediante una codificación basada en el uso de permutaciones de enteros [161], y para generar programas de computadora de manera automática, adop-

tando una codificación de árbol [80]. Asimismo, se usaron vectores de números reales para resolver problemas de optimización continua [500] y para entrenar redes neuronales [370]. Durante este período, se aceptaba comúnmente que el diseñar representaciones específicas para un problema, junto con operadores especializados, permitía realizar una búsqueda más eficiente [160]. De hecho, varios investigadores defendían la posición de que el diseño de esquemas de mutación inteligentes podía evitar el uso de los operadores de recombinación [157].

En los noventa, el uso de la Programación Evolutiva en problemas de optimización continua se extendió considerablemente. Durante este período, se hicieron enormes esfuerzos por desarrollar esquemas auto-adaptativos. Cuando se usa auto-adaptación, algunos parámetros del algoritmo se codifican junto con las variables del problema, de tal manera que los operadores evolutivos actúen sobre ellos y definan sus valores. Bajo este tipo de esquema, las variables del problema son llamadas parámetros objeto, mientras que los parámetros del algoritmo evolutivo son denominados parámetros de la estrategia. En los primeros intentos por incorporar auto-adaptación a la Programación Evolutiva, se adaptaba el factor de escalamiento de la mutación Gaussiana [158]. En otras variantes más avanzadas se consideran varios operadores de mutación simultáneamente [500]. Puesto que los esquemas auto-adaptativos resultaron tener un desempeño superior a las variantes tradicionales, este mecanismo se volvió muy popular en muchas de las implementaciones usadas para resolver problemas del mundo real, particularmente cuando se adoptaba codificación mediante vectores de números reales [155]. Cabe destacar que el mecanismo

de auto-adaptación de la Programación Evolutiva para optimización continua presenta varias similitudes con respecto al mecanismo adoptado en las Estrategias Evolutivas. La diferencia más significativa es que la Programación Evolutiva no incorpora recombinación. Adicionalmente, algunos detalles de implementación eran también diferentes en las primeras variantes auto-adaptativas de la Programación Evolutiva [132]. Por ejemplo, el orden en el que se sometían a mutación los parámetros objeto y los de la estrategia era diferente en la Programación Evolutiva y las Estrategias Evolutivas, si bien estas diferencias acabaron por desvanecerse con el tiempo. Además, existen casos documentados en los que se ha podido mostrar que el uso exclusivo del operador de mutación produce mejores resultados que cuando se incorpora recombinación y viceversa [385]. Hay evidencia que indica que es benéfico adaptar también los parámetros asociados con el operador de cruza usado por las Estrategias Evolutivas [221], lo cual hace que disminuyan las diferencias entre la Programación Evolutiva y las Estrategias Evolutivas. Se hace notar, sin embargo, que el primer mecanismo de auto-adaptación de parámetros para optimización continua se produjo en las Estrategias Evolutivas, por lo cual este tema se discutirá a mayor detalle cuando hablemos de dicho tipo de algoritmo evolutivo.

En la actualidad, la Programación Evolutiva es el tipo de algoritmo evolutivo menos usado a nivel mundial. Aunque su uso original fue en problemas de predicción, hoy en día se les suele usar más para optimización en espacios continuos. Esto se refleja también en México,

en donde existe muy poca evidencia de su uso para optimización mono-objetivo [91] y multi-objetivo [92] en espacios continuos.

1.3 Estrategias Evolutivas

A mediados de la década de los sesenta, tres estudiantes de la Universidad Técnica de Berlín (Bienert, Schwefel y Rechenberg) estudiaban problemas prácticos de mecánica de fluídos y otras áreas similares con el fin de construir robots que pudieran resolver de manera automatizada problemas de ingeniería [155]. Para este efecto, formularon dichas tareas como problemas de optimización y desarrollaron máquinas autónomas que se modificaban automáticamente bajo la aplicación de ciertas reglas. También intentaron aplicar algoritmos clásicos de optimización. Sin embargo, debido a que los problemas que querían resolver tenían ruido y eran multimodales, estos algoritmos fueron incapaces de producir soluciones aceptables. Para enfrentar esta problemática decidieron aplicar métodos de mutación y selección análogos a la evolución natural. Rechenberg publicó el primer reporte en torno a la nueva técnica, denominada "estrategia evolutiva", aplicada a la minimización del arrastre total de un cuerpo en un túnel de viento [378]. Posteriormente, resolvieron otros problemas tales como el diseño de tubos y de boquillas hidrodinámicas [155].

Filosóficamente, la codificación de soluciones adoptada por las Estrategias Evolutivas es una abstracción del fenotipo de los individuos [154]. Por esta razón se puede adoptar libremente la codificación de soluciones candidatas, a fin de adaptarse mejor a los requerimientos

del problema. Además, en las Estrategias Evolutivas se permite la recombinación de individuos, si bien este operador no fue implementado en la versión original del algoritmo.

A la primera versión de este algoritmo se le conoce como la $(1+1)$-EE (EE significa "Estrategia Evolutiva"). Su funcionamiento es el siguiente: primero, se crea una solución inicial de manera aleatoria y se considera como "el padre". Esta solución es mutada (es decir, el valor de una de sus variables es modificado aleatoriamente) y al individuo mutado se le denomina "el hijo". La mutación se aplicaba siguiendo una distribución binomial, a fin de aplicar mutaciones pequeñas más frecuentemente que las mutaciones grandes, como ocurre en la naturaleza. La solución mutada se vuelve el padre si es mejor o igual que el individuo que la originó. El nuevo padre se vuelve a mutar, repitiéndose este proceso hasta alcanzar un cierto criterio de paro. Los experimentos realizados por los creadores de esta técnica mostraron que resultaba mejor que los métodos de optimización clásicos en problemas de alta complejidad.

En 1965, Hans-Paul Schwefel implementó por primera vez una Estrategia Evolutiva en una computadora de uso general y la aplicó a problemas de optimización continua [406]. La contribución más importante de Schwefel fue que incorporó una mutación Gaussiana con media cero, y este operador se sigue usando a la fecha. El operador de mutación se podía controlar mediante el ajuste de las desviaciones estándar (es decir, el tamaño de paso) de la distribución Gaussiana. Los estudios realizados por Schwefel en torno al tamaño de paso originaron los primeros análisis teóricos de la Estrategia Evolutiva.

Hacia finales de los sesenta se introdujo una versión de la Estrategia Evolutiva que usaba una población con varios individuos. Esta primera versión poblacional fue denominada $(\mu + 1)$-EE, y adoptaba μ individuos, los cuales se usaban para crear un nuevo individuo por medio de operadores de recombinación y mutación. Posteriormente, los μ mejores individuos de entre los $\mu + 1$ existentes se seleccionaban para sobrevivir. Este esquema es muy similar a la selección de estado uniforme [490] que se volvió popular años más tarde en los algoritmos genéticos. Con el tiempo, las Estrategias Evolutivas permitieron adoptar cualquier número de padres y de hijos. Además, se comenzaron a utilizar dos esquemas de selección: la $(\mu + \lambda)$-EE y la (μ, λ)-EE. En la primera, μ padres generan λ hijos y sobreviven los μ mejores de la unión de ellos (padres e hijos). En la segunda, μ padres generan λ hijos y sobreviven los μ mejores hijos.

Las primeras publicaciones sobre las Estrategias Evolutivas fueron escritas en alemán, lo cual limitó su acceso. Sin embargo, en 1981, Schwefel publicó el primer libro sobre este tema en inglés [407]. Desde entonces, un número importante de investigadores en el mundo las han estudiado y aplicado extensamente. En México, se han utilizado para resolver problemas de procesamiento de imágenes [186] y para optimización no lineal con restricciones [298].

El libro de Schwefel se enfoca en el uso de las Estrategias Evolutivas para optimización continua, que es de hecho el área en la que esta técnica ha tenido mayor éxito. En éste se discute, entre otros temas, el uso de diferentes tipos de operadores de recombinación y la

adaptación de las distribuciones Gaussianas utilizadas para el operador de mutación.

En los estudios iniciales del operador de recombinación se propusieron cuatro esquemas posibles, combinando dos propiedades diferentes. Primero, se proporcionaron dos opciones para el número de padres involucrados en la creación de un hijo: i) recombinación bisexual (o local) y ii) recombinación multisexual (o global). En la primera se seleccionan dos padres y éstos se recombinan entre sí, produciendo así la solución candidata del hijo. En la segunda, se seleccionan diferentes padres para ir generando cada variable de la solución candidata del hijo. También hay dos opciones para combinar los valores de los padres: la denominada "recombinación discreta", en la que uno de los valores se selecciona aleatoriamente, y la denominada "recombinación intermedia", en la que se calcula el promedio de los valores de los padres. En estos esquemas de recombinación no se puede especificar la cantidad de padres que toman parte en cada recombinación. Rechenberg [380] propuso una generalización que cambia esta restricción. En este caso, se usa un nuevo parámetro, denominado ρ para especificar el número de padres que tomarán parte en la creación de cada individuo. Esto originó las variantes denominadas $(\mu/\rho, \lambda)$-EE y $(\mu/\rho + \lambda)$-EE. Posteriormente, se propusieron otras versiones que asignaban pesos a la contribución de cada uno de los individuos que participaban en la recombinación [33].

Pese a utilizar recombinación, el operador principal de las Estrategias Evolutivas es la mutación. Como se indicó anteriormente, este operador se suele basar en una distribución Gaussiana con media cero.

En la mayor parte de los casos, los individuos se perturban mediante la adición de un vector aleatorio generado de una distribución Gaussiana multivariada. En otras palabras, se usa la expresión siguiente:

$$x_i^{(t+1)} = x_i^t + N_i(0, C) \tag{1.1}$$

donde t es la iteración actual, $x_i^{(t+1)}$ es el nuevo individuo, generado a partir de x_i^t, C representa una matriz de covarianza y $N(0, C)$ devuelve un número aleatorio con media cero. De tal forma, el nuevo individuo (es decir, el "hijo") se crea a partir del anterior (el "padre") mediante una perturbación aleatoria a una de sus variables. Desde los orígenes de las Estrategias Evolutivas, resultó claro que este tipo de mecanismo podría tener un efecto significativo en el desempeño del algoritmo. Consecuentemente, se han desarrollado diversos métodos de adaptación durante la ejecución del algoritmo. Los primeros esquemas de este tipo se originaron de los estudios realizados por Rechenberg [379], quien analizó las propiedades de convergencia de las Estrategias Evolutivas en relación a la frecuencia relativa de las mutaciones exitosas (si el hijo es mejor que su padre en términos del valor de la función objetivo). Este trabajo condujo al primer esquema adaptativo en el cual el tamaño de paso global se ajusta mediante un procedimiento en línea con el objetivo de producir mutaciones exitosas con una tasa igual a 1/5. Este procedimiento se conoce como la "regla de éxito 1/5". Esta regla hace ajustes a la desviación estándar cuando la tasa de éxito no es exactamente 1/5, bajo la premisa que esta tasa de mutaciones exitosas es la ideal. Es importante hacer notar que dicha regla no es general, lo cual originó las versiones auto-adaptativas de las Estrate-

gias Evolutivas. En el primer esquema de este tipo, el único parámetro adaptado fue el tamaño de paso global. Sin embargo, con los años, se desarrollaron esquemas auto-adaptativos mucho más sofisticados.

Otro punto interesante en torno a las Estrategias Evolutivas es que si se adapta toda la matriz de covarianza, pueden inducirse correlaciones entre las mutaciones realizadas, en las cuales se involucran diferentes parámetros, lo cual hace más apropiado su uso para lidiar con funciones no separables. El primer esquema en el cual se adapta toda la matriz de covarianza fue propuesto por Schwefel [407]. En este caso, el sistema coordenado en el cual se realiza el control del tamaño de paso, así como los tamaños de paso mismos, se auto-adaptan. Se hace notar, sin embargo, que este esquema no ha sido muy exitoso, porque requiere de tamaños de población muy grandes para operar adecuadamente. Una alternativa más eficiente es la Estrategia Evolutiva con un esquema de adaptación de la matriz de covarianza propuesta por Hansen y Ostermeier conocida como CMA-ES [203]. En este esquema se combinan dos técnicas diferentes de auto-adaptación con el objetivo de construir matrices de covarianza que maximicen la creación de vectores de mutación que fueron exitosos en generaciones anteriores. Esta técnica es muy compleja y requiere de varios parámetros, lo cual ha dado pie a variantes más simples y con menos parámetros [44].

Finalmente, existen también esquemas que favorecen ciertas direcciones de búsqueda sin adaptar toda la matriz de covarianza. Por ejemplo, Poland y Zell [366] propusieron utilizar la dirección principal de descenso para guiar la búsqueda. La mayor ventaja de este tipo de esquemas es que reducen la complejidad computacional en términos

de tiempo y espacio. Sin embargo, dado que el tiempo para calcular las matrices de covarianza no es significativo con respecto al de la evaluación de la función objetivo (sobre todo, si el problema es costoso en términos de tiempo de cómputo), este tipo de esquemas no ha sido muy popular, aunque puede resultar útil en problemas con un gran número de variables, pues en ese caso, el costo de calcular matrices de covarianza se incrementa de manera significativa.

Las Estrategias Evolutivas son una opción muy buena para resolver problemas de optimización continuos en los que todas las variables son números reales, aunque en años recientes han sido desplazadas por metaheurísticas como la evolución diferencial [373], que resultan más fáciles de implementar y usar.

1.4 Algoritmos Genéticos

John Holland [214] identificó la relación entre el proceso de adaptación que existe en la evolución natural y la optimización, y conjeturó que dicho proceso podría jugar un papel crítico en otras áreas tales como el aprendizaje, el control automático o las matemáticas. Su objetivo inicial era estudiar de manera formal la adaptación y propuso un entorno algorítmico inspirado en estas ideas, al cual denominó originalmente "planes reproductivos y adaptativos", pero que después renombró como "Algoritmo Genético". El funcionamiento de estos algoritmos se basa en la modificación progresiva de un conjunto de estructuras de datos con el objetivo de adaptarlas a un cierto ambiente. La forma específica de realizar dichas adaptaciones está inspirada en la genética y

en la evolución natural. Sin embargo, en poco tiempo, los Algoritmos Genéticos se volvieron solucionadores de problemas en áreas tales como el aprendizaje de máquina[14] y el control automático. Sin embargo, su uso más extendido ha sido en optimización, tanto en espacios discretos como en espacios continuos.

Filosóficamente, la codificación utilizada en los Algoritmos Genéticos es una abstracción del genotipo de los individuos. En este caso, sin importar el tipo de variables que tenga el problema, éstas se deben codificar en binario. Aunque hoy en día es posible usar otras codificaciones, Holland argumentó que el alfabeto binario es universal y presenta varias ventajas, tanto de tipo práctico como teórico. Sin embargo, pese a sus ventajas tiene la desventaja evidente de que se requiere de un proceso de decodificación para convertir la cadena binaria en la variable que codifica. Asimismo, el uso de la codificación binaria puede en algunos casos introducir problemas debido a la discretización de un espacio de búsqueda que originalmente no era discreto (por ejemplo, cuando las variables son números reales). Tales problemas van desde usar una precisión inadecuada hasta el manejo de cadenas binarias de gran tamaño, con las cuales es más ineficiente operar dentro del Algoritmo Genético.

A cada cadena binaria que codifica una variable se le denomina *gene*, y cada bit dentro de un gene se conoce como *alelo*. A la concatenación de todos los genes (es decir, al conjunto de todas las variables del problema) se le denomina *cromosoma*, como se ilustra en la

[14]Los Algoritmos Genéticos dieron pie a los **Sistemas de Clasificadores**, en los cuales se evolucionaron conjuntos de reglas del tipo IF-THE-ELSE para resolver problemas de clasificación.

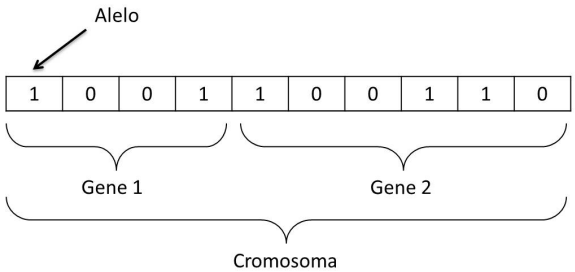


Figura 1.2: Estructura de una cadena cromosómica para un problema con dos variables ("gene 1" y "gene 2" corresponden a dichas variables).

figura 1.2. Un individuo está formado normalmente por un cromosoma (es decir, se considera una estructura haploide, pese a que en la naturaleza es muy común que las especies tengan una estructura diploide, es decir, con dos cromosomas).

La forma de operar de un Algoritmo Genético es bastante simple. Se parte de un conjunto de soluciones que se generan aleatoriamente al que se conoce como la "población inicial". Para cada una de estas soluciones (o "individuos") se debe calcular un valor de *aptitud*. Dicho valor está en función del objetivo que se quiere optimizar, aunque por lo general, requiere algún proceso de normalización a fin de evitar valores de aptitud negativos o divisiones entre cero. Las parejas de padres que se reproducirán se seleccionan de acuerdo a su valor de aptitud (los mayores de toda la población). En la versión original del Algoritmo Genético, se utiliza un esquema de selección proporcional, lo que significa que los individuos más aptos tienen mayor probabilidad de sobrevivir. Sin embargo, los menos aptos tienen una probabilidad distinta de cero de ser seleccionados y podrían volverse padres también. Los individuos seleccionados se recombinan mediante un operador de

cruza para generar la población de hijos, a la cual se aplica el operador de mutación. Los hijos mutados conforman la nueva población que reemplaza por completo a la población anterior. El proceso se repite hasta cumplirse un cierto criterio de paro, que normalmente es un número máximo de generaciones (o iteraciones). La tabla 1.2 muestra una población hipotética de 5 individuos de un algoritmo genético, incluyendo los cromosomas y las aptitudes correspondientes.

Individuo No.	**Cromosoma**	**Aptitud**	**% del Total**
1	11010110	254	24.5
2	10100111	47	4.5
3	00110110	457	44.1
4	01110010	194	18.7
5	11110010	85	8.2
Total		1037	100.0

Tabla 1.2: Ejemplo de una población de 5 individuos, cuyos cromosomas contienen 8 bits cada uno. En la tercera columna se muestra la aptitud (calculada mediante una función hipotética).

La figura 3 ilustra el funcionamiento de un método de selección proporcional conocido como "la ruleta". El triángulo negro es un pivote. En la selección por el método de la ruleta, se simula el giro aleatorio de un disco que contiene a los individuos de la población y en cada giro se selecciona como padre al individuo que marca el pivote al detenerse la ruleta.

La porción que cada individuo ocupa en la ruleta corresponde a su aptitud, de tal forma que los individuos más aptos tengan mayor probabilidad de seleccionarse y recombinarse mediante alguno de los varios tipos de cruzas disponibles. Las figuras 1.4 y 1.5 ilustran el

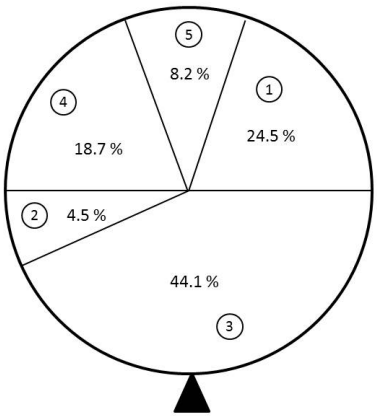


Figura 1.3: Método de selección de la ruleta (uno de los métodos de selección proporcional que se usan con los algoritmos genéticos). Los 5 individuos de la tabla 1.2 se colocan en una rueda, en la cual cada subdivisión corresponde a su proporción de aptitud con respecto al total. En esta técnica de selección se simula el giro de una ruleta, de tal forma que seleccionamos como padre al individuo donde la ruleta se detiene. Evidentemente, los individuos más aptos tienen mayor probabilidad de ser seleccionados, pero los menos aptos también podrían ser seleccionados como padres, aunque su probabilidad es muy baja.

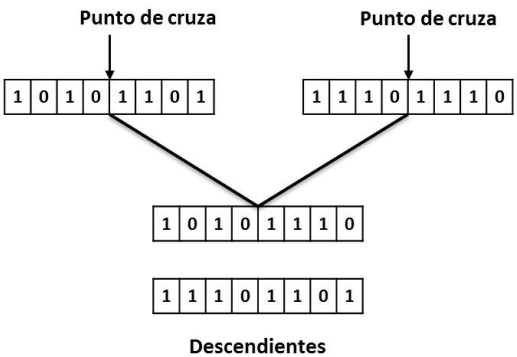


Figura 1.4: Cruza de un punto. Nótese cómo la cruza entre dos padres, produce dos hijos. El hijo 1 tiene los primeros 4 bits del padre 1 y los otros 4 del padre 2. El hijo 2 contiene los primeros 4 bits del padre 2 y los otros 4 del padre 1.

funcionamiento de la cruza de un punto y la cruza de 2 puntos, respectivamente.

Finalmente, cada uno de los hijos que se producen mediante la cruza es mutado, como se ilustra en la figura 1.6.

Con los años, se desarrollaron distintas variantes del Algoritmo Genético que involucran el uso de esquemas de selección deterministas y de estado uniforme (en los cuales se reemplaza a un solo individuo de la población a cada iteración, en vez de reemplazarlos a todos), codificaciones no binarias, operadores de cruza y de mutación muy sofisticados y/o altamente especializados para un dominio en particular, cromosomas diploides y multiploides, así como otro tipo de operadores tales como la inversión, la traslocación y la segregación [184].

Un aspecto distintivo de los Algoritmos Genéticos es que se ha considerado tradicionalmente que el operador de cruza es su fuente de

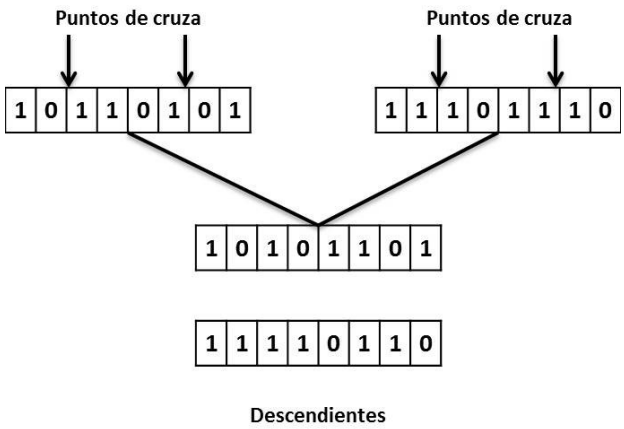


Figura 1.5: Cruza de dos puntos. El hijo contiene tiene los 2 primeros y 2 últimos últimos bits 2 del padre 1 y los bits complementarios del padre 2. El hijo 2 tiene los 2 primeros y 2 últimos bits del padre 2 y los bits complementarios del padre 1.

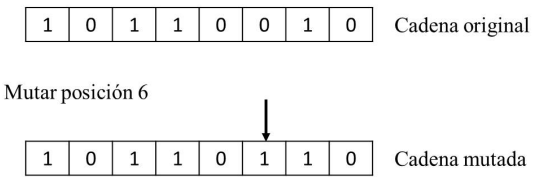


Figura 1.6: Ejemplo del operador de mutación de un algoritmo genético. En este caso, se muta la posición 6 de la cadena original. El operador aplica un "NOT" a la posición original, con lo cual se cambia el cero original a uno.

mayor poder, y que el operador de mutación es sólo complementario, como un esquema para mantener diversidad y para mantener el espacio de búsqueda completamente conectado. Esta filosofía se contrapone a la de las Estrategias Evolutivas en las que la mutación es el operador principal y la cruza es sólo un operador secundario y a la Programación Evolutiva en la que sólo se usa la mutación.

En su libro de 1975, Holland trató de modelar el funcionamiento del Algoritmo Genético a través de su "Teorema de los Esquemas" en el que deriva una fórmula que proporciona la probabilidad de que un cierto patrón de bits (denominado "esquema") sobreviva a los efectos de la selección, la cruza y la mutación [214]. Con el paso de los años, se desarrollaron modelos matemáticos mucho más sofisticados para intentar explicar el funcionamiento de un Algoritmo Genético, incluyendo aquellos que se basan en cadenas de Markov y los que se basan en mecánica estadística [132]. La primera demostración de convergencia de un Algoritmo Genético ordinario [397] (llamado "canónico") mostró que debe usarse un operador conocido como "elitismo" para poder garantizar convergencia. Este operador consiste en retener al mejor individuo de cada generación y pasarlo intacto a la generación siguiente, sin cruzarlo ni mutarlo.

En México, se ha hecho investigación muy relevante en torno a los Algoritmos Genéticos, tanto en el contexto de aprendizaje de máquina[15] como en el de optimización mono-objetivo y multi-objetivo.[16] Tam-

[15]Por ejemplo, hay grupos de investigación en el ITESM Campus Monterrey y el INAOE.

[16]Hay grupos de investigación que han abordado estos temas en la Universidad Veracruzana, la Universidad de Guadalajara, el CINVESTAV-IPN, la Universidad Autónoma de Querétaro, el CICESE, el CIMAT y la UNAM.

bién se hace investigación en torno al uso de algoritmos genéticos y sus variantes en electrónica (el denominado "hardware evolutivo"),[17] visión,[18] bioinformática,[19] finanzas,[20] criptografía,[21] procesamiento de imágenes,[22] y diseño.[23] Asimismo, hay grupos de investigación que han abordado el estudio de los operadores evolutivos[24] y de aspectos teóricos de los algoritmos genéticos.[25] En nuestro país se ha abordado también el estudio de la **programación genética**,[26] una variante del algoritmo genético propuesta por John Koza [240] en la que se usa una codificación de árbol para evolucionar programas.

1.5 Otras Metaheurísticas Bio-Inspiradas

En México se han estudiado también varias otras metaheurísticas bio-inspiradas, de entre las que destacan las siguientes:

Cúmulos de Partículas: Propuesta por Kennedy y Eberhart a mediados de los noventa [230] simula el movimiento de un conjunto de partículas en un fluído (agua o aire). Las trayectorias se definen a partir de una fórmula muy simple que involucra la velocidad de cada

[17] En el INAOE y el CIMAT.
[18] En el CICESE.
[19] En el CICESE y el ITESM Campus Estado de México.
[20] En el ITESM Campus Monterrey, el CINVESTAV-IPN y el Banco de México.
[21] En el CIC-IPN y el CINVESTAV-IPN.
[22] En el ITESM Campus Guadalajara, el CIMAT, el CICESE y el CINVESTAV-IPN.
[23] En el Instituto Tecnológico Autónomo de México (ITAM).
[24] En la Universidad de Guadalajara y el CINVESTAV-IPN.
[25] En la UNAM y el CINVESTAV-IPN
[26] En la UNAM, INFOTEC, CINVESTAV-IPN y el CICESE.

partícula, un factor de inercia y la influencia de la mejor solución que se haya obtenido para una partícula individual y para todo el cúmulo. Hay grupos de investigación que han usado la optimización mediante cúmulos de partículas, sobre todo para optimización mono-objetivo y multi-objetivo.[27]

Sistemas Inmunes Artificiales: Busca simular el comportamiento de nuestro sistema inmune, el cual nos defiende de las enfermedades que invaden nuestro cuerpo en forma de "antígenos". Para ello, el sistema inmune genera células denominadas "anticuerpos", que tienen la capacidad de acoplarse a los antígenos a fin de anularlos. Los primeros modelos computacionales del sistema inmune datan de los ochenta y fueron popularizaron por Stephanie Forrest, quien realizó aportaciones clave en esta área [323]. Hoy en día existen diversas propuestas de modelos computacionales basados en sistemas inmunes, de entre los que destacan la selección negativa, la selección clonal y el modelo de red inmune. De ellos, la selección clonal ha sido el más utilizado en la literatura especializada (sobre todo para resolver problemas de optimización). Los sistemas inmunes artificiales se han usado por grupos de investigación en México para abordar problemas de criptografía y de optimización mono-objetivo y multi-objetivo.[28]

Evolución Diferencial: Propuesta por Kenneth Price y Rainer Storn a mediados de los noventa [373], se puede ver más bien como un método de búsqueda directa, en el que se trata de estimar el gradiente

[27] En CINVESTAV-IPN, unidades Tamaulipas, Guadalajara y Zacatenco.
[28] En el ITESM Campus Estado de México, el CIC-IPN y el CINVESTAV-IPN.

en una región (y no en un punto), mediante el uso de un operador que recombina a 3 individuos. Pese a que su diseño luce muy simple, esta técnica es sumamente poderosa para realizar optimización en espacios continuos, y produce resultados equiparables e incluso superiores a los obtenidos por las mejores Estrategias Evolutivas que se conocen, pero usando menos parámetros y con implementaciones mucho más sencillas. Hay grupos de investigación en México que han utilizado la evolución diferencial en diferentes problemas de optimización mono-objetivo y multi-objetivo.[29]

Colonia de Hormigas: Propuesta por Marco Dorigo a principios de los noventa [124], simula el comportamiento de un grupo de hormigas que salen de su nido a buscar comida. Durante su recorrido segregan una sustancia llamada "feromona", la cual pueden oler las demás hormigas. Al combinarse los rastros de feromona de toda la colonia, la ruta resultante es la más corta del nido a la comida. En el modelo computacional desarrollado por Dorigo se considera un factor de evaporación de la feromona y se debe poder evaluar una solución parcial al problema. En su versión original, esta metaheurística se podía utilizar sólo para problemas que se mapearan al problema del viajero. Sin embargo, con los años se desarrollaron versiones más sofisticadas que se pueden aplicar a otros tipos de problemas combinatorios e incluso a problemas continuos. En México, existen grupos de investigación que han utilizado esta metaheurística para resolver problemas de diseño de

[29]En la Universidad Veracruzana y el CINVESTAV-IPN.

circuitos y de optimización mono-objetivo y multi-objetivo en espacios continuos.[30]

1.6 La Computación Evolutiva en México

Los orígenes de la investigación en computación evolutiva en México se remontan a finales de los años ochenta en el Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM) Campus Monterrey y en la Universidad Nacional Autónoma de México (UNAM). En el ITESM se creó el Centro de Inteligencia Artificial en 1989 (hoy llamado Centro de Sistemas Inteligentes) el cual, casi desde su inicio, tuvo interés en estudiar Sistemas Inspirados en la Naturaleza. Manuel Valenzuela Rendón (doctorado en 1989 bajo la supervisión de David E. Goldberg en la Universidad de Alabama, Estados Unidos) fue tal vez el primer doctor especializado en computación evolutiva en México. De hecho, Goldberg lo menciona en los agradecimientos de su famoso libro de algoritmos genéticos [184].

En la UNAM existen tesis de licenciatura y doctorado de finales de los ochenta y principios de los noventa, dirigidas por Germinal Cocho Gil, que utilizan el método de Monte Carlo y algoritmos genéticos [303, 23, 304], si bien las primeras tesis que utilizan el término "algoritmo genético" en su título, se remontan a 1993.[31]

[30] En CINVESTAV-IPN.

[31] Comunicación personal de la Dra. Katya Rodríguez Vázquez, quien realizó diversas consultas a la base de datos de tesis doctorales de la UNAM para proporcionar este dato.

Hacia la segunda mitad de los noventa, existían ya cursos sobre computación evolutiva en instituciones tales como el Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN), la Maestría en Inteligencia Artificial de la Universidad Veracruzana y la UNAM.

Actualmente, existen diversos posgrados en México en los cuales es posible obtener una maestría y un doctorado en computación con especialidad en computación evolutiva. Por ejemplo, en el Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN), el Centro de Investigación en Matemáticas (CIMAT), el Instituto de Investigación en Matemáticas Aplicadas y Sistemas de la UNAM (IIMAS-UNAM), el CIC-IPN, la Universidad de Guadalajara, el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), el Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), INFOTEC (Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación) y en el ITESM.

1.7 Perspectivas Futuras

La investigación en torno a la computación evolutiva ha tenido un crecimiento importante en México en los últimos años. Hoy en día, existe un número considerable de investigadores que trabajan con distintos tipos de algoritmos evolutivos y con otras metaheurísticas bioinspiradas, no sólo en torno a aplicaciones, sino también en investigación básica (por ejemplo, diseño de nuevos algoritmos, análisis

teórico y estudio de operadores). Como se indicó anteriormente, existen también varios programas de maestría y doctorado reconocidos en el Padrón Nacional de Posgrado de CONACyT, en los cuales es posible especializarse en computación evolutiva. Asimismo, existen grupos de investigación que colaboran con investigadores de varios países[32] y que tienen amplia visibilidad internacional.

En años recientes, ha habido también una creciente participación de estos grupos en los congresos internacionales más importantes del área: la *Genetic and Evolutionary Computation Conference*, el *IEEE Congress on Evolutionary Computation* y *Parallel Problem Solving from Nature*. Sin embargo, son todavía muy pocos los grupos de investigación de México que publican en las revistas más reconocidas del área: *IEEE Transactions on Evolutionary Computation*, *Evolutionary Computation* y *Genetic Programming and Evolvable Machines*. Esto puede deberse a que varios de los grupos de investigación están conformados por doctores jóvenes en proceso de consolidación y también porque algunos de los grupos actuales están orientados al desarrollo de aplicaciones y prefieren publicar en revistas no especializadas en computación evolutiva.

Finalmente, es deseable que los grupos de investigación en computación evolutiva existentes en México que son relativamente recientes se consoliden y que su visibilidad aumente; sin duda existen la

[32]Todos los grupos de investigación de México mencionados a lo largo de este capítulo han tenido colaboraciones con investigadores de países tales como Estados Unidos, Reino Unido, Australia, España, Chile, Francia, Holanda, Alemania, China y Japón.

masa crítica y el potencial para que eso ocurra, dada la cantidad de colaboraciones internacionales con las que ya cuenta.

Capítulo 2

Evolución Diferencial: Una Breve Introducción

En este capítulo se hace una descripción general del algoritmo de Evolución Diferencial. Este tipo de algoritmo evolutivo representa una excelente alternativa para encontrar soluciones cercanas al óptimo a problemas de optimización continua complejos. Debido a la simplicidad de su implementación y sus excelentes habilidades de exploración y explotación, diversas variantes de este algoritmo se han aplicado para resolver problemas con diferentes características. En este capítulo se describen los elementos básicos del algoritmo, así como las diferentes variantes que se han desarrollado para mejorar su desempeño.

2.1 Introducción

Con el nombre de *algoritmos evolutivos* se identifica a un grupo de procedimientos que se utilizan para resolver muchos problemas prácticos y que se distinguen de otras técnicas porque están inspirados en las teorías que sintetizan la evolución Darwiniana y la herencia genética descrita por Mendel [93]. Los algoritmos evolutivos han probado ser herramientas poderosas para encontrar soluciones aceptables a problemas donde otras técnicas fallan o consumen una excesiva cantidad de tiempo. Estos algoritmos se destacan por combinar exitosamente dos mecanismos de búsqueda: *exploración* y *explotación*. La exploración permite identificar áreas prometedoras del espacio de búsqueda y la explotación lleva a cabo una búsqueda especializada dentro de estas áreas para encontrar soluciones cercanas al óptimo. Su aplicación se extiende en prácticamente todas las áreas de la actividad humana como por ejemplo en salud [469], educación [416], ciencias [497], industria [484], economía [52] y defensa [499]. Los algoritmos evolutivos también han sido utilizados en áreas emergentes como la ciencia de datos [293], la inteligencia de negocios [149] y el big data [45]. En estos algoritmos un conjunto de soluciones candidatas evoluciona para alcanzar una solución cercana al óptimo por la aplicación de procedimientos que permiten recombinar y alterar sus valores. En la jerga de los especialistas del área, una solución candidata se conoce como *individuo*, al conjunto de soluciones se denomina *población* y los procedimientos involucrados en el proceso evolutivo se identifican como operadores de *selección*, *cruzamiento* y *mutación*. Comúnmente dos individuos son seleccionados y sus valores son recombinados, y algunas

veces mutados, para crear nuevas soluciones candidatas. Los nuevos individuos son considerados parte de una nueva población que sustituye a la anterior. Entonces, una secuencia de nuevas poblaciones son creadas hasta que una condición de término es alcanzada y la mejor solución de las última población es seleccionada como resultado del algoritmo. Este proceso evolutivo es guiado por una función de aptitud que determina la calidad de cada individuo en la población. La figura 2.1 muestra el esquema general de un algoritmo evolutivo. A lo

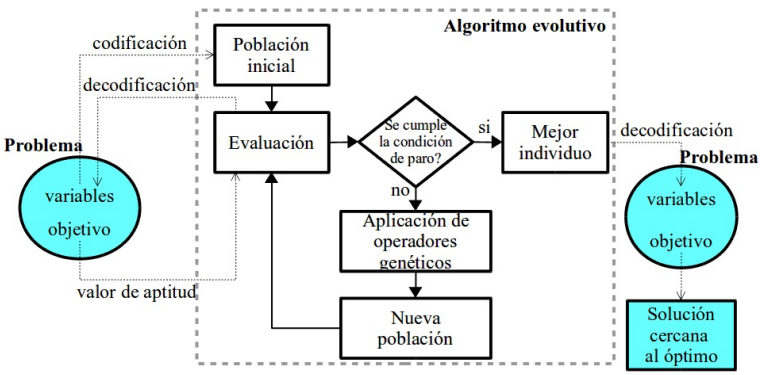


Figura 2.1: Esquema general de un algoritmo evolutivo.

largo de su desarrollo histórico, varios tipos de algoritmos evolutivos han sido implementados difiriendo en la forma de representar a sus individuos y en la implementación de sus operadores genéticos. Aún cuando en los años 50 varios autores introducen los términos *evolución* y *proceso evolutivo* en sus propuestas para resolver diversos problemas [50, 166, 54], es en los años 60 cuando se sientan las bases del desarrollo de los algoritmos evolutivos con la introducción de tres técnicas: la *Programación Evolutiva*, las *Estrategias Evolutivas* y los *Algoritmos Genéticos*. Fogel propone en 1962 [159] la Programación Evolutiva

donde las soluciones candidatas se representan con máquinas de estados finitos y Rechenberg [378] y Schwefel en 1965 [406] desarrollan las Estrategias Evolutivas que codifican sus individuos con vectores de números reales. Los Algoritmos Genéticos introducidos en 1962 por Holland [213] son sin duda el tipo de algoritmo evolutivo más popular. Estos algoritmos tradicionalmente utilizan una cadena binaria para representar a sus individuos, aunque también se han utilizado cadenas de enteros [193] y secuencias de números reales [488]. Desarrollos posteriores a estas técnicas han aportado al éxito de los algoritmos evolutivos, destacándose la *Programación Genética* [239], los *Algoritmos Coevolutivos* [212, 371], los *Algoritmos Culturales* [382], el algoritmo de *Evolución Diferencial* [435], los *Algoritmos de Estimación de Distribuciones* [318], la *Evolución Gramatical* [398] y la *Programación de Expresión de Genes* [151]. Cada uno de estos algoritmos tienen sus ventajas y sus propios desafíos, y han sido utilizados para resolver diferentes tipos de problemas. La figura 2.2 muestra un esquema del desarrollo histórico de los algoritmos evolutivos. En particular, el algoritmo de Evolución Diferencial ha demostrado ser muy competitivo y exitoso para resolver problemas numéricos complejos en comparación con otros algoritmos [472, 354, 82].

2.2 El algoritmo de Evolución Diferencial

Evolución Diferencial (DE por sus siglas en inglés) es un tipo de algoritmo evolutivo que fue originalmente diseñado por Storn y Price en 1995 para resolver problemas de optimización continua [435]. En par-

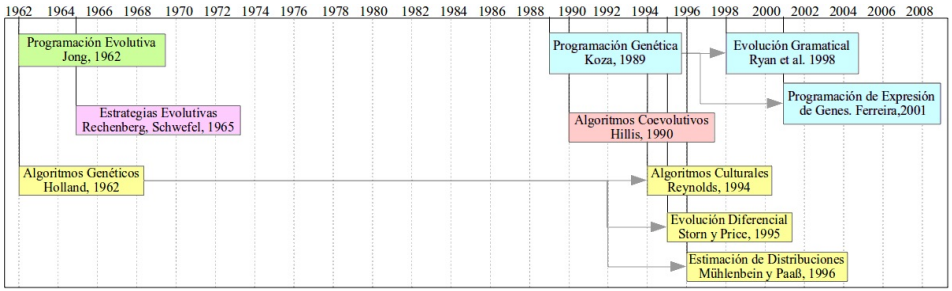


Figura 2.2: Línea del tiempo del desarrollo de los algoritmos evolutivos.

ticular, en este algoritmo cada individuo usualmente se representa con un vector de valores continuos $x = \left(x_1, x_2, \ldots, x_n\right)$ de n parámetros. Además, en lugar de implementar los operadores de cruza y mutación tradicionales, el algoritmo crea cada nuevo individuo con una combinación lineal de los valores de dos o más individuos de la población actual que son seleccionados al azar. La forma en que se implementan los operadores de cruza y mutación se ha utilizado en la literatura para identificar las diferentes variantes del algoritmo. `DE/A/B/C` es la notación comúnmente usada para nombrar estas variantes, donde `A` y `B` determinan el tipo de mutación implementado, y `C` identifica el tipo de cruza implementado por la variante. En particular, `A` representa el procedimiento de selección de los individuos para construir el individuo mutado, y `B` es el número de vectores diferencia usados para el operador de mutación. DE puede considerarse un proceso de tres pasos que incluye una fase de inicialización, el proceso evolutivo y el paso final que determina el resultado obtenido. La fase de inicialización involucra la selección y evaluación de un grupo de NP individuos que son generados aleatoriamente de un espacio de búsqueda finito $\Omega \subseteq \mathbb{R}^n$ los

cuales se utilizan como población inicial del algoritmo, conocido como X_0. Si para cada $j \in \{1, \ldots, n\}$, $x_j^{\min}$ y $x_j^{\max}$ son los valores mínimo y máximo del j-ésimo parámetro en Ω, el valor x_j^i del i-ésimo individuo x^i en la población inicial es determinado como sigue:

$$x_j^i = x_j^{\min} + r\left(x_j^{\max} - x_j^{\min}\right) \tag{2.1}$$

donde r es un número aleatorio uniformemente distribuido dentro del rango $[0, 1]$. El proceso evolutivo implementa un esquema iterativo para evolucionar la población inicial. En cada iteración de este proceso, conocido como una *generación*, una nueva población de individuos es generada utilizando información de la población previamente creada. A diferencia de otros algoritmos evolutivos, el orden de aplicación de los operadores genéticos en el algoritmo de Evolución Diferencial es diferente. Primero, se aplica la mutación, después la cruza y al final se lleva a cabo la selección del mejor individuo. Para cada $i \in \{1, \ldots, \mathrm{NP}\}$ en la generación g, x^i es obtenido de la población X_{g-1} y es utilizado para crear un nuevo vector u^i aplicando los operadores de cruza y mutación. Los vectores x^i y u^i son conocidos como el *vector objetivo* y el *vector de prueba*, respectivamente. Estos vectores son evaluados por el operador de selección para actualizar la nueva población X_g. Los operadores de mutación, cruza y selección son descritos en detalle en los siguientes párrafos. En el paso final, cuando una *condición de paro* ω es alcanzada, el algoritmo devuelve el mejor individuo en la población actual. La condición de paro más comúnmente utilizada es comparar el número de generaciones construidas

contra un valor fijo, pero otros criterios pueden ser implementados, como aquellos descritos por Zielinski y Laur [518].

2.2.1 Mutación

Un *vector mutado* v^i es construido al combinar los valores de varios individuos x^{r_j} aleatoriamente seleccionados de X_{g-1}. El número de individuos seleccionados para realizar la combinación depende del operador de mutación implementado por la variante del algoritmo. En alguna de estas variantes, el individuo con la mejor aptitud x^{best} en la población X_{g-1} es también utilizado por el operador de mutación. Entre los diferentes operadores de mutación descritos en la literatura, destacan los siguientes:

$$\text{DE/rand/1:} \quad v^i = x^{r_1} + \mathrm{F}\big(x^{r_2} - x^{r_3}\big), \tag{2.2}$$

$$\text{DE/best/1:} \quad v^i = x^{\text{best}} + \mathrm{F}\big(x^{r_1} - x^{r_2}\big), \tag{2.3}$$

$$\text{DE/current-to-best/1:} \quad v^i = x^i + \mathrm{F}\big(x^{\text{best}} - x^{r_1}\big) + \mathrm{F}\big(x^{r_2} - x^{r_3}\big), \tag{2.4}$$

$$\text{DE/rand/2:} \quad v^i = x^{r_1} + \mathrm{F}\big(x^{r_2} - x^{r_3}\big) + \mathrm{F}\big(x^{r_4} - x^{r_5}\big), \tag{2.5}$$

y

$$\text{DE/best/2:} \quad v^i = x^{\text{best}} + \mathrm{F}\big(x^{r_1} - x^{r_2}\big) + \mathrm{F}\big(x^{r_3} - x^{r_4}\big), \tag{2.6}$$

donde F es un valor definido por el usuario que representa un factor de escala aplicado para controlar la variación diferencial. Storn y Price señalan que F puede fijarse entre 0 y 2 [436], pero otros autores

recomiendan valores entre 0.4 y 1 [106]. En la figura 2.3 se presenta un ejemplo de la aplicación del operador de mutación DE/rand/1. En este ejemplo $x^{r_1}=\big(10.56, 3.16\big)$, $x^{r_2}=\big(2.53, 9.03\big)$ y $x^{r_3}=\big(5.6, 6.26\big)$. Usando F igual a 0.5, el vector mutado resultante es $v^i=\big(9.02, 4.54\big)$. La mutación aporta a las habilidades de explotación y exploración del

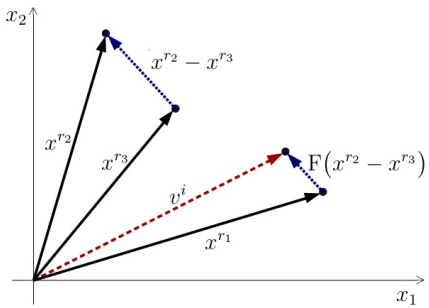


Figura 2.3: Ejemplo de la creación de un vector mutado con el esquema DE/rand/1.

método. Debido a que al inicio del proceso evolutivo los individuos se encuentran relativamente dispersos en el espacio de búsqueda, la diferencia $x^{r_2} - x^{r_3}$ entre los vectores seleccionados es grande, por lo que los vectores mutados pueden ubicarse en diferentes áreas del espacio de búsqueda, pero a medida que el proceso evolutivo avanza, las soluciones se vuelven más cercanas y las diferencias entre ellos son más pequeñas, lo que produce que los vectores mutados se encuentran concentrados en áreas promisorias del espacio de búsqueda [150, 327].

2.2.2 Cruza

El operador de cruza se utiliza para combinar la información entre el vector objetivo y el vector mutado para construir el vector de prueba.

Por cada $j \in \{1, \ldots, n\}$, x_j^i o v_j^i se seleccionan para construir u^i usando un factor de cruza CR $\in [0, 1]$ el cual es también un valor especificado por el usuario. Dos tipos de cruza se utilizan en este caso: cruza binomial y cruza exponencial.

Cruza binomial: En este operador, cada valor asignado a u^i es seleccionado de v^i o de x^i comparando el valor CR con un valor $r \in [0, 1]$ seleccionado al azar. Para garantizar que al menos un parámetro de u^i toma un valor de v^i, este operador utiliza un valor $l \in \{1, \ldots, n\}$ también seleccionado al azar. Formalmente, la cruza binomial se define como:

$$\text{bin:} \quad u_j^i = \begin{cases} v_j^i & \text{si } r \leq \text{CR or } j = l, \\ x_j^i & \text{de otra forma,} \end{cases} \tag{2.7}$$

Tomando el vector mutado $v^i = (9.02, 4.54)$ del ejemplo de la figura 2.3 y un vector objetivo $x^i = (4.06, 7.66)$, en la figura 2.4 se muestra un ejemplo de la creación de un vector de prueba usando la cruza binomial. El vector de prueba generado en este ejemplo es $u^i = (4.06, 4.54)$.

Cruza exponencial: Este operador de cruza asigna una secuencia de valores de v^i a u^i. Los valores restantes que son asignados a u^i son obtenidos de x^i. El tamaño de la secuencia se determina con dos valores: 1) un índice l que representa la posición inicial de la secuencia en v^i, seleccionado aleatoriamente dentro del rango $\{1, \ldots, n\}$, y 2) el número de intentos consecutivos $L \leq n$ en los cuales $r \leq$ CR determina la posición final de la secuencia, como

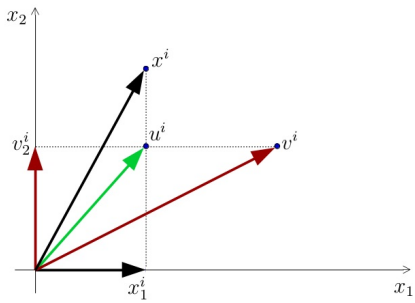


Figura 2.4: Ejemplo de la creación de un vector de prueba usando la cruza binomial.

es definida en la Ec. (2.8). La cruza exponencial es similar al operador de cruza de dos puntos introducido por Cavicchio [75].

$$\text{exp:} \quad u_j^i = \begin{cases} v_j^i & \text{si } l \leq j \leq \min(l+L-1, n), \\ x_j^i & \text{de otra forma,} \end{cases} \tag{2.8}$$

Por ejemplo, si se tienen dos vectores de cinco parámetros $v^i{=}\big(2.4, 3.6, 1.2, 7.8, 0.2\big)$ y $x^i{=}\big(3.4, 9.6, 1.2, 5.2, 3.1\big)$ y se aplica el operador de cruza exponencial usando un factor de cruza de 0.6, entonces se debe obtener el valor de l y después generar una secuencia de valores aleatorios para r. Si $l{=}2$ y la secuencia de valores de r es 0.6, 0.3, 0.4 y 0.6, y el vector de prueba que se construye es $u^i{=}\big(2.4, 9.6, 1.2, 5.2, 0.2\big)$.

La cruza ayuda al algoritmo a mantener la diversidad de la población y eventualmente evitar que caiga en óptimos locales [106]. El valor de CR es complicado de determinar, pero algunos autores señalan que su valor puede ajustarse entre 0.3 y 0.9 [171].

2.2.3 Selección

Un torneo uno-a-uno es aplicado para determinar cual vector, entre x^i y u^i, es seleccionado como miembro de la nueva población X_g. Este esquema de selección permite garantizar que el proceso evolutivo mantenga una mejora continua [150].

2.2.4 Estructura del Algoritmo

El Algoritmo 1 muestra la estructura de una implementación clásica del algoritmo. Éste requiere de tres parámetros para controlar el proceso evolutivo (CR, F, y NP), y también la selección de un espacio de búsqueda para construir la población inicial, y la definición del tamaño de individuo, de una función objetivo para evaluar cada individuo y de una condición de paro para finalizar el proceso evolutivo.

DE tiene varias ventajas en comparación con otros algoritmos evolutivos, como la simplicidad de su implementación, su habilidad para producir mejores resultados que aquellos obtenidos por los demás, y su baja complejidad espacial [106].

2.3 Modificaciones al algoritmo de Evolución Diferencial

Aunque DE requiere de la definición de un número menor de parámetros en comparación con el número de parámetros requeridos por otros algoritmos evolutivos, su desempeño es sensible a los valores seleccionados de CR, F, y NP [519]. Varios autores señalan que los val-

Algoritmo 1 Estructura del algoritmo de introducido por Storn y Price en [435].

función EvolucionDiferencial(CR, F, NP)
Entrada:
CR: Porcentaje de cruza
F: Factor de escala
NP: Tamaño de la población

Salida:
x^{best}: Mejor individuo en la población actual

Fase de inicialización
$g \leftarrow 0$
$X_g \leftarrow \varnothing$
para each $i \in \{1, \ldots, \text{NP}\}$ **hacer**
  $x^i \leftarrow$ Un individuo generado aleatoriamente usando la Ec. (2.1)
  $X_g \leftarrow X_g \cup \{(i, x^i)\}$
fin para

Proceso evolutivo
mientras ω no se ha alcanzado **hacer**
  $g \leftarrow g + 1$
  $X_g \leftarrow \varnothing$
  para each $i \in \{1, \ldots, \text{NP}\}$ **hacer**
    $x^i \leftarrow$ Vector objetivo obtenido aleatoriamente de X_{g-1}
    $v^i \leftarrow$ Vector mutado generado usando alguna de las Ecs. (2.2)–(2.6)
    $u^i \leftarrow$ Vector de prueba construido usando alguna de las Ecs. (2.7)–(2.8)
    si $f(u^i)$ es mejor que $f(x^i)$ **entonces**
      $X_g \leftarrow X_g \cup \{(i, u^i)\}$
    si no
      $X_g \leftarrow X_g \cup \{(i, x^i)\}$
    fin si
  fin para
fin mientras

Paso final
$x^{\text{best}} \leftarrow$ El mejor individuo en X_g
devolver x^{best}
fin función

ores adecuados para estos parámetros pueden variar para diferentes problemas [261] y es conocido que la definición de estos parámetros depende de las características propias de cada problema [55]. Además, Lampinen y Zelinka [247] señalan que este algoritmo puede presentar el problema de *estancamiento* debido a una incorrecta selección de sus parámetros. El estancamiento en un algoritmo se presenta cuando éste no converge a alguna solución y la población presenta una gran diversidad de individuos [327]. Por lo anterior, la mayoría de los estudios realizados para mejorar el desempeño del algoritmo de Evolución Diferencial se relacionan con técnicas para ajustar los valores de sus parámetros (F, CR y NP) y para combinar las ventajas de las diferentes variantes del algoritmo, usando un subconjunto de ellas en lugar de solamente aplicar una variante. En los siguientes párrafos se describen las propuestas para mejorar el desempeño del algoritmo que se consideran los más representativos en la literatura especializada. Para simplificar su descripción, estas propuestas se agrupan de acuerdo al criterio utilizado: Ajuste de parámetros y combinación de variantes. Por un lado, debido a que F y CR son los parámetros del algoritmo más estudiados y donde más variantes de mejora han sido desarrolladas, estos estudios se agrupan en: enfoques que usan parámetros globales y enfoques que usan parámetros para cada individuo. Adicionalmente se presentan los enfoques relacionados con el ajuste del tamaño de la población y también aquellos donde se modifica la forma que generar los vectores mutados dentro del proceso evolutivo. La figura 2.5 presenta un esquema del desarrollo histórico de estos enfoques.

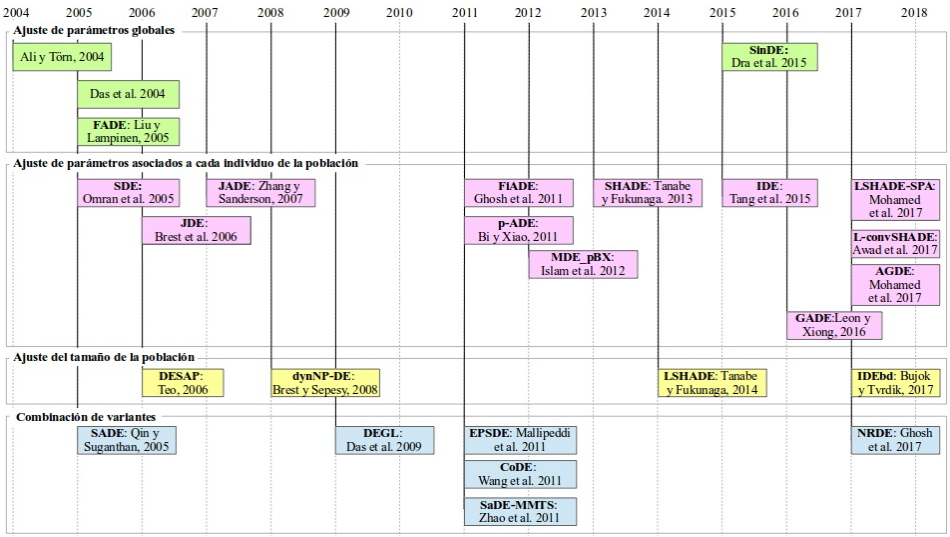


Figura 2.5: Línea del tiempo de las modificaciones a DE.

2.3.1 Ajuste de parámetros globales

En estos enfoques, los operadores genéticos utilizan los mismos valores para F y CR en todos los individuos de una misma población, como se ilustra en la figura 2.6. Estos valores se actualizan al finalizar cada generación, con el ánimo de mejorar las habilidades de exploración y explotación del algoritmo. Dos enfoques se concentran en ajustar el valor de F, manteniendo fijo el valor de CR: En 2004, Ali y Törn [19] lo actualizan considerando la aptitud del mejor y del peor individuo en la población, y Das *et al.* en 2005 [104] proponen dos enfoques para modificar su valor: el primero lo modifica de forma aleatoria, y el segundo reduce su valor de forma lineal conforme el proceso evolutivo avanza. Por otro lado, Liu y Lampinen en 2005 [262] proponen un enfoque diferente al cual denominan FADE (Fuzzy Adaptive DE algorithm). En este algoritmo, los valores de F y CR se ajustan usando un

sistema de control lógico difuso. El sistema utiliza las diferencias entre generaciones sucesivas, tanto de los valores de los individuos como de sus valores de aptitud. Finalmente, Dra *et al.* [126] en 2015 proponen una versión alternativa denominada SinDE (Sinusoidal DE) en donde los valores de F y CR se ajustan utilizando fórmulas trigonométricas, permitiendo cambiar la dirección del vector diferencia dentro del operador de mutación.

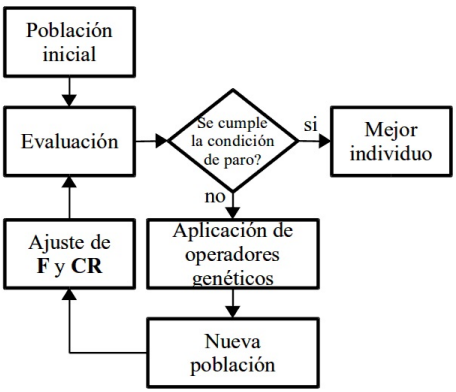


Figura 2.6: Esquema de ajuste de parámetros globales del algoritmo.

2.3.2 Ajuste de parámetros asociados a cada individuo de la población

En este tipo de algoritmos, cada individuo de la población tiene asociado su propio conjunto de parámetros, como se muestra en la figura 2.7. Estos parámetros se ajustan de forma independiente dentro del proceso evolutivo, como se esquematiza en la figura 2.8.

En 2005, Omran *et al.* [335] asocian el valor de F a cada individuo de la población en su método denominado SDE (self-adaptive DE).

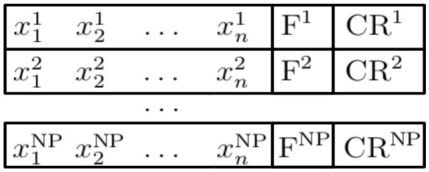


Figura 2.7: Parámetros asociados a cada individuo de la población.

Este método actualiza el valor de F combinando los valores de F de tres individuos de la población actual que son seleccionados aleatoriamente. En 2006, Brest *et al.* [55] describen su método denominado jDE, donde los valores de F y CR se pueden modificar en cada iteración del proceso evolutivo, basados en una decisión estocástica utilizando un par de umbrales previamente definidos. En 2007, Zhang y Sanderson [509] implementan un método conocido como JADE, donde F y CR son ajustados utilizando distribuciones normales independientes cuya media es calculada usando el promedio de los vectores mutados mejor adaptados en cada iteración. En 2011, Ghosh *et al.* [180] actualizan los valores de estos parámetros en base al valor de aptitud de los individuos en la población, en un método denominado FiADE (Fitness-Adaptive DE). Una versión parecida es presentada en 2015 por Tang *et al.* [443] en el método llamado IDE (DE with an individual-dependent mechanism). En 2011, Bi y Xiao [46] describen el método p-ADE (pbest vector-based self-adaptive DE variant) donde el vector mutado se crea usando el mejor individuo en la población y el mejor de los vectores que se utilizaron para crear al vector objetivo. Los valores de F y CR de cada individuo son ajustados usando un factor asociado a las mejores y peores aptitudes de los individuos en la población. En 2012, Islam *et al.* [220] introdujeron

el método llamado MDE_pBX (modified DE with p-best crossover) donde se modifica la variante DE/current-to-best/1 al utilizar el mejor elemento de un grupo de individuos seleccionados aleatoriamente de la población actual, en lugar de usar el mejor individuo de la población. Los valores de los parámetros se ajustan en base a un par de distribuciones independientes que son actualizadas al final de cada generación. En 2013, Tanabe y Fukunaga [442] implementan el método SHADE (Success-History based Adaptive DE) que, basándose en JADE, actualiza los valores de F y CR usando una memoria de los vectores mutados que han sido exitosamente seleccionados previamente dentro del proceso evolutivo. A partir de SHADE, varias versiones se han desarrollado como el método L-convSHADE (SHADE using covariance matrix learning with Euclidean Neighborhood) [29] y el método LSHADE-SPA (LSHADE with semi parameter adaptation) [306]. En 2016, Leon y Xiong [251] introducen una estrategia voraz para actualizar los valores de los parámetros de cada individuo en el método llamado GADE (Greedy Adaptive DE). Se utilizan los valores de los parámetros de las soluciones vecinas para ajustar los parámetros del nuevo vector mutado. Finalmente, Mohamed y Mohamed en 2017 [307] proponen el método denominado AGDE (Adaptive guided DE).

2.3.3 Ajuste del tamaño de la población

En los enfoques descritos previamente solamente se ajustan los valores de F y CR, manteniendo el tamaño de la población constante. Diversos estudios indican que el tamaño de la población también afecta al desempeño de un algoritmo evolutivo [401, 283], por lo que existen

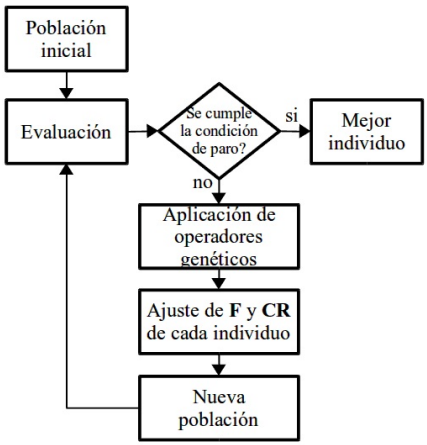


Figura 2.8: Esquema de ajuste de parámetros para cada individuo de la población.

trabajos que se han centrado en ajustar el tamaño de la población para garantizar la convergencia del algoritmo. Existen algunos enfoques donde se ajusta el tamaño de la población y se describirán a continuación. En la figura 2.9 se muestra el esquema general en que estos enfoques ajustan el tamaño de la población durante el proceso evolutivo.

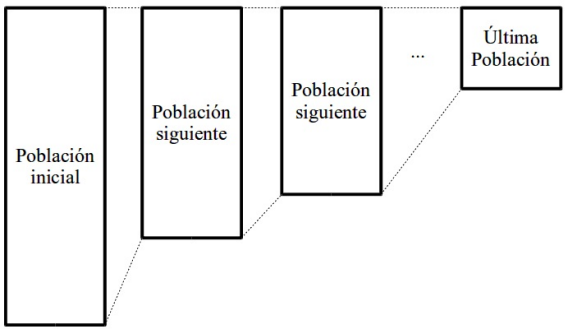


Figura 2.9: Esquema general del ajuste del tamaño de la población en DE.

En 2006, Teo [446] describe el método DESAP (Differential Evolution with self-adapting populations) donde el tamaño de la población se ajusta usando un factor distribuido uniformemente entre 0 y 1. En 2008, Brest y Sepesy Maučec [56] presentan el algoritmo dynNP-DE (DE con NP dinámico) donde van reduciendo a la mitad el tamaño de la población conforme el proceso evolutivo progresa. Utilizando el número total de evaluaciones y un parámetro que indica el número de posibles reducciones, el algoritmo establece en qué generaciones el algoritmo reducirá el tamaño de la población. En 2014, Tanabe y Fukunaga [442] presentaron una versión con ajuste de población del método SHADE, denominada L-SHADE, donde se aplica una reducción lineal del tamaño de la población, previo a una ordenación a los individuos por su aptitud, que permite eliminar solo los individuos peor adaptados. En 2017, Bujok y Tvrdík [63] describen el método IDEbd (IDE with Population Size Adaptation) en donde el tamaño de la población es adaptado usando un factor calculado en base al nivel de diversidad de la población actual.

2.3.4 Combinación de variantes

Debido a la forma de combinar los individuos para crear el vector de prueba de cada una de las variantes de DE, estás presentan diferentes capacidades de exploración y explotación, además de que son más efectivas para diferentes tipos de problemas [297]. Por ejemplo, la variante DE/rand/1 es mejor para explorar el espacio de búsqueda pero tiene una velocidad de convergencia lenta, y DE/best/1 converge más rápido hacia soluciones cercanas a la óptima pero solo para prob-

lemas con un único óptimo global (unimodales) [485]. Debido a que usar diferentes estrategias de mutación durante diferentes momentos del proceso evolutivo pueden ser mejor que usar una sola variante durante todo el proceso, varios autores se han enfocado en desarrollar estrategias para combinar diferentes variantes de DE y lograr con esto que cooperen en la creación de un mejor vector objetivo [284]. La figura 2.10 muestra un esquema general de la forma en que se aplican estas estrategias dentro del algoritmo de Evolución Diferencial. En 2005, Qin y Suganthan [376] describieron una versión conocida

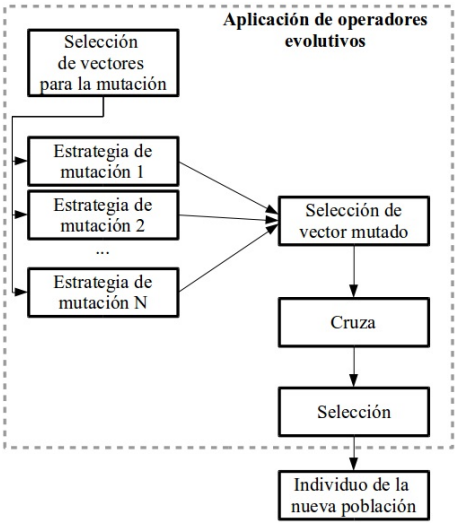


Figura 2.10: Esquema general de la estrategia de combinación de variantes de DE.

como SADE (Self-Adaptive DE algorithm for numerical optimization), donde introducen un esquema que selecciona de forma alternada una de dos variantes del algoritmo, de acuerdo a su tasa de éxito dentro del proceso evolutivo. En 2009, Das *et al.* [103] presentaron el método llamado DEGL (DE with Global and Local Neighborhoods)

donde cada nuevo vector mutado se crea combinando dos vectores: uno construido usando una vecindad del vector objetivo (vector local) y otro creado usando el mejor individuo de la población (vector global). En 2011, Mallipeddi *et al.* [284] asignaron a cada individuo de la población el tipo de variante para generar el vector mutado y los valores para F y CR, en el método EPSDE (Ensemble of mutation strategies and parameters in DE). Tanto el tipo de variante como los valores de los parámetros se escogen aleatoriamente de un grupo de ellos que es previamente definido. Dependiendo del éxito de la aplicación de dichos valores, éstos se pueden cambiar durante el proceso evolutivo. También en ese mismo año, Wang *et al.* [486] en su método CoDE (Composite DE) crean tres vectores para cada vector objetivo. Cada vector es construido usando variantes diferentes y con parámetros generados aleatoriamente. El mejor de los tres vectores se usa como vector de prueba para aplicar el operador de selección. Adicionalmente, Zhao *et al.* [515] implementaron un método muy similar denominado SaDE-MMTS (SaDE with a modified multi-trajectory search) donde el vector mutado se selecciona de entre los vectores generados por tres diferentes variantes del algoritmo, pero la selección se hace en base a una probabilidad que iterativamente se ajusta en base a la cantidad de vectores mutados que son exitosamente seleccionados. En 2017, Ghosh *et al.* [181] propusieron el método NRDE (Noise Resilient DE) donde cada individuo puede ser mutado por alguna de dos variantes con igual probabilidad. Los valores de F y CR son ajustados para cada individuo basado en un rango preestablecido previamente.

2.4 Evolución Diferencial para otros tipos de problemas

Aunque DE fue definido originalmente para resolver problemas de optimización continua sin restricciones, también se ha utilizado con éxito para resolver otros tipos de problemas, como aquellos donde los valores de las variables del problema son discretos [445, 302, 258], donde el espacio de búsqueda está limitado por una o más restricciones [295, 502] y aquellos donde se manejan dos o mas objetivos [20, 37]. Prado *et al.* [372] describe varias adaptaciones al algoritmo de Evolución Diferencial para problemas de optimización combinatoria, como aquellos que implementan una matriz de permutación [373], los que obtienen una secuencia ordenada de valores [444, 375] y los que aplican una transformación de los valores continuos al espacio discreto [337]. En el caso de los problemas de optimización con restricciones, Mezura-Montes *et al.* [296] describen varios enfoques que se han utilizado para aplicar DE en este tipo de problemas, como aquellos que utilizan técnicas para eliminar o reparar soluciones infactibles [511, 352], y los que implementan técnicas de penalización [260]. Finalmente, para problemas de optimización multi-objetivo, Mezura-Montes *et al.* [299] resume las principales técnicas basadas en el algoritmo de Evolución Diferencial para manejar dos o más objetivos, como los enfoques basados en dominancia de Pareto [400] y en jerarquización de Pareto [279], así como los métodos que no se basan en Pareto [254] y los enfoques que combinan diferentes estrategias [248]. Hoy en día existen también algoritmos basados en descomposición [255] y en indicadores de desempeño [393].

2.5 DE en México

En el país se tiene mucha actividad en investigación relacionada con DE, particularmente el grupo EVOCINV de CINVESTAV Zacatenco estudia a este algoritmo en optimización multi-objetivo [299] y temas afines al manejo de múltiples objetivos como algoritmos basados en indicadores [393], modelos subrogados [26], operadores de selección [291], operadores de variación [16], adaptación de parámetros [243], entre otros temas.

En el Centro de Investigación en Inteligencia Artificial de la Universidad Veracruzana se desarrolla investigación en optimización con restricciones usando DE [296], aplicaciones de DE en mecatrónica [77] y en control automático [475], DE y búsqueda local [123], optimización dinámica multi-objetivo [289], DE asistida por subrogados en espacios restringidos [130], entre otras temáticas.

Recientemente, en el Instituto Tecnológico de Veracruz se desarrollan aplicaciones de DE en aprendizaje automático [386], en el Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC) del IPN se aplica DE en problemas de mecatrónica [66] y en el Centro de Investigación en Matemáticas se aplica en procesos químicos [482].

Todas estas menciones sólo son representativas, pues la DE, al ser un algoritmo fácil de implementar y que provee resultados competitivos, ha sido particularmente atractiva para su estudio y aplicación en instituciones mexicanas.

2.6 Retos y Perspectivas

Si bien la investigación sobre DE hoy en día es abundante, existen retos importantes aún por resolver. Entre ellos se pueden destacar los siguientes:

- **Teoría**: Existen trabajos destacados para fortalecer los fundamentos teóricos de la DE [339]. Sin embargo, se requieren mayores esfuerzos para poder establecer, por ejemplo, tiempos de convergencia en problemas complejos de optimización.

- **Optimización combinatoria**: Combinar los operadores originales de la DE, diseñados para espacios continuos, con las representaciones de soluciones propias de problemas combinatorios sigue siendo un reto importante [338, 286]. Es válido optar por la transformación del espacio de búsqueda discreto a uno continuo usando esquemas tipo llaves aleatorias [38] y mantener la mutación diferencial. Por otro lado, se puede usar una representación para optimización combinatoria, pero el uso de penalizaciones o reparadores es necesario [18], y ahí se disminuye el efecto de la mutación diferencial.

- **Nuevos problemas de optimización**: Las adaptaciones de DE en nuevos espacios de búsqueda como la optimización binivel [199], la optimización dinámica restringida [17], dinámica multi-objetivo [289], la optimización asistida por subrogados [130] y la optimización robusta sobre el tiempo [197] siguen siendo temas poco explorados.

- **Problemas de optimización con alta dimensionalidad**: La escalabilidad que ofrece DE en espacios de muchas dimensiones [290] es un nicho fértil por explorar.

Aún con los retos antes mencionados, y considerando la gran cantidad de investigación sobre este algoritmo, se vislumbra que en los próximos años la DE se consolide como uno de los algoritmos bio-inspirados más confiables y robustos, y que sea de fácil acceso vía herramientas de software para optimización, tal como lo son ahora otras técnicas de programación matemática.

2.7 Conclusiones

El algoritmo de Evolución Diferencial es un enfoque exitoso para resolver problemas de optimización continua sin restricciones. Los individuos que evolucionan al aplicar los operadores de este algoritmo tradicionalmente se codifican como secuencias de números reales, pero otras representaciones se han utilizado también. Además, con la inclusión de diversas estrategias para manejar varios objetivos y conjuntos de restricciones, DE se puede aplicar para resolver otro tipo de problemas. DE se distingue por depender de la definición de un menor número de parámetros que otros algoritmos evolutivos, pero esto también lo hace muy sensible a la definición de los valores para esos parámetros. Diversos variantes del algoritmo han sido desarrolladas para mejorar el desempeño del algoritmo, generando un abanico de alternativas para resolver casi cualquier tipo de problema. El estudio del comportamiento de este algoritmo y la definición de nuevas

propuestas de desarrollo, sin duda aportará a la resolución más eficiente de problemas complejos que interesan a la comunidad científica.

2.8 Para saber más

Para aquellos lectores que deseen abundar en el tema, las siguientes referencias sobre DE pueden ser de interés:

- El libro sobre DE escrito por los autores originales del algoritmo [373].
- Una revisión muy completa de la literatura de DE [106] y su versión actualizada [105].
- Otra revisión de la literatura acompañada de un estudio experimental [327].
- Una revisión de estudios teóricos sobre DE [339].
- Un libro editado con temas diversos y recientes sobre DE [79].
- DE para resolver problemas de optimización multi-objetivo [299].
- DE y su papel importante en espacios restringidos [296].

E

Capítulo 3

Algoritmos Genéticos Paralelos

3.1 Introducción

Las técnicas evolutivas son aptas para su paralelización tanto a nivel algorítmico como a nivel de ejecución. En particular, los algoritmos genéticos (AGs) han sido investigados desde esta perspectiva y esquemas paralelos han surgido desde el inicio de su desarrollo [277]. La paralelización de los AGs persigue la mejora de su desempeño algorítmico en términos de eficiencia y eficacia y la reducción de tiempos de ejecución en sus distintos dominios de aplicación.

Una clasificación general de los Algoritmos Genéticos Paralelos (AGPs) los agrupa por su granularidad en AGPs de grano fino y grueso. Los AGPs de grano grueso dividen la población agrupada en panmixia [1] en varias sub-poblaciones permitiendo la migración de individuos entre éstas. De esta forma, se reduce el número de individ-

[1] Una población en panmixia es aquella donde todos sus individuos, sin restricciones pueden reproducirse entre si.

uos en cada unidad de procesamiento. En consecuencia, se establece un fuerte vínculo entre la algoritmia y la implementación de estas técnicas. Sin embargo, utilizar una única unidad de procesamiento para la evolución de varias sub-poblaciones ha sido un contexto también explorado como un mecanismo de mejora del desempeño algorítmico [12].

Los AGPs de grano grueso, también llamados Algoritmos Genéticos distribuidos (AGds) han permitido la exploración de nuevas vertientes de la versión estándar de los AGs. Tener diferentes parámetros de configuración para realizar las operaciones genéticas, implementar granularidades distintas, incorporar el operador de migración, han sido características a explorar que pueden culminar en un mejor desempeño de los AGs [6, 13, 69, 452].

En la figura 3.1 se muestra un diagrama de un AG distribuido. Cada población es centralizada y se necesitan definir criterios en términos de tasa y frecuencia de migración. También se necesita definir si la migración será síncrona o asíncrona. Herrera et al. [210] presentaron un esquema de un AGP distribuido con poblaciones heterogéneas. Utilizaron la estructura de un hipercubo posicionando en cada vértice una sub-población. Dependiendo de la ubicación de cada sub-población la búsqueda implica mayor explotación o exploración. Esto se logra de manera interna vía la configuración de los operadores genéticos. De esta forma, se procura mantener la diversidad de soluciones de forma global y así mejorar la calidad de las soluciones encontradas.

La principal diferencia entre los AGP de grano fino con respecto a los AGP distribuidos o de grano grueso es la descentralización de su

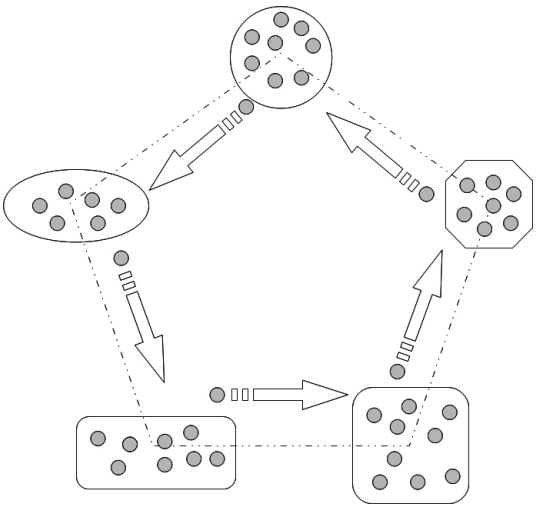


Figura 3.1: Diagrama poblacional de un AGP distribuido o de grano grueso

población y, en consecuencia, la localización del proceso de selección de individuos. En la figura 3.2 se muestra un esquema poblacional de un AGP de grano fino, también conocido como AG celular (AGc). De manera común se utiliza una topología de malla conectada de forma toroidal como lo muestra la figura. Un individuo de la población se ubica en cada cruce de la malla y alrededor de él se define un vecindario cuyas soluciones participan en los procesos de selección, recombinación y mutación para la evolución de éste. En la figura 3.2 se ilustra un vecindario tipo *Moore* con 9 individuos.

En los AGs celulares, cada individuo o solución interactúa con sus vecinos más cercanos, logrando de esta forma el esparcimiento suave de las soluciones a través de la malla debido al traslape de vecindarios [35]. En los AGs distribuidos, la asociación entre soluciones se pierde en comparación con los AGs celulares. Esta es otra característica

importante de los AGc que permite la exploración global del espacio de búsqueda mientras que localmente los individuos explotan la región cercana a su ubicación.

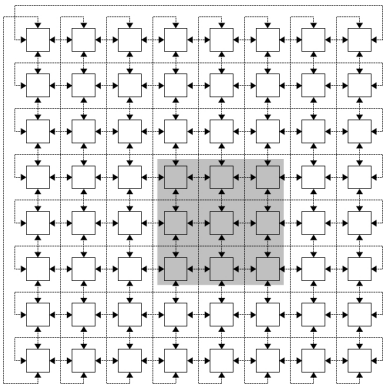


Figura 3.2: Diagrama poblacional del AG celular o de grano fino

La paralelización de los algoritmos evolutivos, en particular de los algoritmos genéticos, persigue distintos objetivos. Por ejemplo, la evolución simultánea de sub-poblaciones, lo cual permite descubrir nueva rutas para solucionar un problema y en consecuencia, reducir el número de generaciones para converger a la solución óptima. Aunado a esto, los tiempos de procesamiento pueden también acortarse [8]. La paralelización de AGs se puede realizar a distintos niveles. Así mismo, es posible la combinación de sub-poblaciones con distintas dinámicas internas. Por ejemplo, algunas sub-poblaciones podrían evolucionar en un esquema de panmixia y otras en un esquema celular. Este tipo de esquemas poblacionales se discutirán a detalle en la subsección 3.1.1.

Una operación importante en los procesos evolutivos de los AGs distribuidos es la migración de los individuos entre sub-poblaciones. Durante el proceso de búsqueda, uno o más individuos son selec-

cionados para migrar y reemplazar a otros individuos de otras sub-poblaciones. Los criterios de migración requieren no sólo de la tasa y la frecuencia de migración, sino también de criterios de selección y reemplazo de individuos al llegar a las otras sub-poblaciones [70]. El proceso de migración puede realizarse de forma síncrona o asíncrona. La migración síncrona supone que todas las sub-poblaciones están en el mismo estado de la búsqueda y la migración de individuos entre sub-poblaciones ocurre de manera simultánea. Por otro lado, la migración asíncrona supone que cada sub-población determina el momento en que los individuos deben migrar. En la sub-sección 3.2.4 se aborda el efecto del sincronismo en los mecanismos de migración.

Cantú-Paz et al. [72] presentó un estudio completo en AGs distribuidos, tanto desde el punto de vista teórico como del práctico. Debido al número de parámetros que se deben definir en los AGds, Cantú-Paz et al. iniciaron con la solución de un problema de optimización para determinar el tamaño adecuado de las sub-poblaciones para alcanzar una cierta calidad de la misma. Utilizaron un modelo matemático considerando la hipótesis de los bloques constructores propuesta por Holland y extendida por Goldberg [184]. El estudio se verifica en varios problemas con un grado alto de precisión, mostrándose una buena respuesta a la escalabilidad.

Para determinar escenarios de operación límite en términos de eficiencia, analizaron la interacción de las sub-poblaciones y su efecto en la calidad de las soluciones [72]. Previamente se ha investigado el cambio en la calidad de las soluciones debido a la forma de interacción de las sub-poblaciones. Tener sub-poblaciones evolucionando de manera

independiente con muy poca o nula interacción entre ellas, resulta en soluciones menos precisas, mientras mejores soluciones se obtienen vía el esquema de panmixia tradicional. Sin embargo, cuando se definen criterios de migración adecuados y un cierto número de soluciones migran entre sub-poblaciones, a una frecuencia media o baja, el desempeño en general de los AGds mejora, alcanzando en un número de casos aceleraciones super-lineales y mejor calidad de las soluciones encontradas. Se puede inferir entonces la importancia del operador de migración, el cual será analizado en detalle en la sub-sección 3.1.2.

3.1.1 AGds homogéneos y heterogéneos

Existen distintos escenarios en los que se pueden definir y aplicar AGds con esquemas poblaciones homogéneos y heterogéneos, es decir, combinaciones de poblaciones centralizadas y descentralizadas. A nivel algorítmico, se puede definir la misma o una diferente configuración de operadores genéticos. A partir de esas configuraciones distintas, diferentes configuraciones de parámetros se pueden determinar para ejecutar la búsqueda en cada sub-población. Por ejemplo, distintas definiciones del operador de recombinación o bien de las probabilidades de mutación pueden operar sobre los individuos de cada sub-población. La comunidad en el área ha incluso explorado escenarios donde las soluciones se representan de manera distinta entre sub-poblaciones. Por otro lado, a nivel de implementación se han explorado esquemas con sub-poblaciones heterogéneas. Por ejemplo, Alba et al. [5] realizó un estudio en distintas plataformas de cómputo ejecutando sub-poblaciones en panmixia y celulares.

Como se puede apreciar, el desarrollo de un esquema de AG paralelos ya sean distribuidos o descentralizados conlleva la definición de un número de parámetros que afectará de manera importante su desempeño algorítmico. Otro ejemplo, es la solución de problemas de optimización en espacios continuos que se han abordado a través de un esquema de AGP utilizando una versión mejorada con cromosomas codificados con números reales [12]. Para esto, se definió una topología doble con un hipercubo que se comparó con una topología sencilla. Las sub-poblaciones ubicadas en sus vértices promovían mayor o menor exploración u explotación de las regiones vía la configuración de los operadores genéticos. La migración entre las sub-poblaciones afectó directamente los tiempos de ejecución por medio de la reducción del número de evaluaciones. La capacidad de búsqueda de la topología doble superó a la de la topología sencilla [12].

Cuando pensamos en paralelizar alguna técnica algorítmica, nos viene a la mente acelerar los procesos de ejecución. Sin embargo, la paralelización de los algoritmos evolutivos podría mejorar tanto los procesos de búsqueda algorítmica como los tiempos de ejecución. Es así que a nivel de implementación, se ha utilizado cómputo heterogéneo para la evaluación de esquemas de AGPs. Por ejemplo, Alba et al. [5] propusieron utilizar una plataforma heterogénea en la cual distintos sistemas operativos con distintas restricciones de ejecución implementaban un modelo de AGPs distribuidos. Las sub-poblaciones se conectaron en un arreglo de anillo unidireccional, cada sub-población ejecutaba un modelo poblacional de estado estable. En cada generación se producía un individuo que reemplazaba a la peor solución

obtenida. Se estableció un esquema de migración asíncrona que demostró mejores resultados, ver sub-sección 3.3 . En general, se observó un mejor desempeño de la ejecución del esquema paralelo en la plataforma heterogénea en comparación con la homogénea, se redujo el número de evaluaciones debido a la explotación de soluciones a distintas tasas de migración.

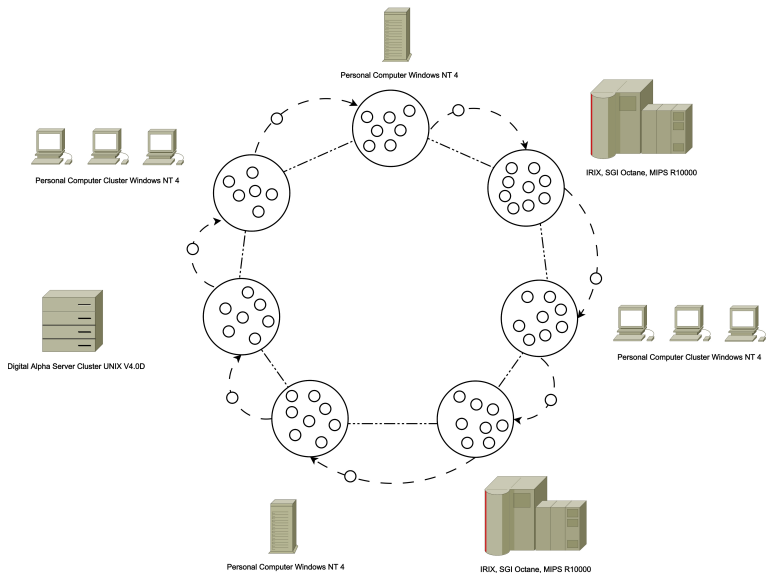


Figura 3.3: AGP con topología poblacional de anillo unidireccional implementada en una plataforma de cómputo heterogénea. La migración entre sub-poblaciones es asíncrona y responde a una dinámica de estado estable.

3.1.2 Migración en AGPs

Como se explicó anteriormente, la incorporación de un nuevo operador llamado migración se hace latente en los esquemas paralelos de

los AGs. En los esquemas de grano grueso, donde se tienen sub-poblaciones de grano fino y grueso evolucionando simultáneamente, la migración se debe considerar como un mecanismo que introduce nuevos individuos y, por lo tanto, promueve o restaura la diversidad entre sub-poblaciones [377]. De forma distinta, en los AGs celulares o de grano fino, un mecanismo de migración implícito trabaja durante el proceso de búsqueda debido al traslape de vecindarios, que permite la difusión de soluciones a lo largo y ancho de la topología definida para la población. Este mecanismo de migración es inherente a la estructura de los AGs celulares y no requiere de un parámetro de frecuencia o tasa de migración o ningún otro mecanismo de selección y reemplazo.

Revisaremos ahora evaluaciones empíricas que se han realizado a criterios de migración definidos para AGPs de grano grueso. Aunque estos criterios son definidos para este tipo de esquemas paralelos, también es posible aplicar un segundo criterio migratorio en AGs celulares, además del mecanismo natural inherente a estos.

Para establecer un criterio de migración adecuado y mejorar el desempeño algorítmico de los AGPs de grano grueso, se examinaron los casos límite para aplicar el proceso de migración. Se sugiere mantener un cierto nivel de aislamiento entre las sub-poblaciones. Si la sub-población evoluciona en completo aislamiento, las mejoras en diversidad son mínimas y, por ende, la capacidad de búsqueda se ve reducida. Por otro lado, si se aplica el operador de migración con una frecuencia máxima, el costo computacional se incrementa de manera significativa. Por lo tanto, el objetivo de la migración es evitar

el aislamiento entre sub-poblaciones mientras se mantiene un nivel aceptable de comunicación entre éstas.

Es posible calcular experimentalmente el número adecuado de unidades de procesamiento para minimizar el tiempo de ejecución de un esquema de AGPs [73]. El análisis considera casos límite en términos de conectividad entre sub-poblaciones, es decir, el caso de tener frecuencias de migración mínimas y máximas. La conclusión teórica para una población indica que el número óptimo de unidades de procesamiento está dado por: $O\left(\sqrt{\frac{nT_f}{T_c}}\right)$, donde n es el tamaño de la población, T_f es el tiempo de cómputo de la función objetivo de un individuo y T_c es el tiempo promedio de comunicación para una unidad de procesamiento. Esto quiere decir que dividiendo una población en sub-poblaciones distribuidas en múltiples unidades de procesamiento reduce el tiempo global de ejecución. La misma conclusión se logró para un esquema de múltiples poblaciones considerando además la evaluación de funciones objetivo separables [71]. Esto quiere decir que utilizar un número de unidades de procesamiento en un esquema evolutivo paralelo tiene un impacto positivo en la reducción de los tiempos de ejecución.

Es también importante analizar el efecto de los criterios de migración que se utilizan en esquemas de AGPs de grano grueso. El objetivo principal es determinar cómo el operador de migración afecta la presión de selección inducida e implícitamente poder explicar las aceleraciones super-lineales que se pueden modelar para este tipo de esquemas. La forma en que los individuos se seleccionan para migración y reemplazo tienen un efecto considerable en la presión de selección y se ha observado que tasas altas de migración producen

un incremento en ésta. Tener una presión de selección alta afecta la búsqueda al producir una convergencia prematura a un óptimo local.

Aunque la operación de migración se ha investigado principalmente para esquemas de AGPs de grano grueso, también se ha revisado su efecto en AGPs de grano fino, considerando tanto el mecanismo de migración implícita vía el traslape de vecindarios, como la definición de criterios para migración de individuos desde y hacia distintas posiciones de la topología de malla definida para la población. Por ejemplo, se han utilizado estructuras de árboles binarios para dividir una topología de malla de dos dimensiones en formaciones concéntricas de individuos, definiendo para cada formación criterios de migración distintos [357]. El intercambio de individuos entre estas formaciones definidas por su estructura alenta el proceso de búsqueda, permitiendo una mayor exploración. Los resultados mostraron un mejor desempeño en la solución de problemas de optimización con restricciones; el operador de mutación se aplica para evitar la presencia de super individuos que pueden dirigir la búsqueda al estancamiento, conquistando la topología de malla mucho más rápido. En la siguiente sección se abordará el comportamiento de los AG celulares o de grano fino.

3.2 AGs celulares o de grano fino

Los AGs celulares o de grano fino se construyen a partir de poblaciones descentralizadas donde los individuos interactúan con otros individuos ubicados dentro de un vecindario definido. Es decir, interactúan con los vecinos más cercanos. Usualmente, los AGCs se implementan

en topologías de mallas de n-dimensionales, manteniendo conexiones toroidales entre los cruces de malla utilizando formas comunes como una malla lineal, cuadrada o rectangular.

El procesamiento de los AG celulares corresponde al de los Autómatas Celulares (ACs). Estos determinan una regla global que define un comportamiento específico que se construye a partir de reglas locales. Se han explorado ACs de una, dos y tres dimensiones en combinación con AGs con la finalidad de evolucionar el comportamiento global requerido [59]. Esta combinación de un AC y un AG originó el nombre de AG celular, con la diferencia de tener una representación más elaborada en cada cruce de la malla: cromosomas. Se observó que la información de las reglas locales o individuos alcanzaba distancias lejanas dentro de la malla definida, esto debido al traslape de los vecindarios. De esta forma se determinó que la información se mueve entre vecindarios, permitiendo acelerar el proceso de crear nuevos comportamientos ad-hoc al problema que se esté solucionando y con una pérdida mínima de información [31].

El pseudocódigo 2 es el estándar de un AG celular. En un primer paso, se genera una población inicial de forma aleatoria ubicando cada individuo en un cruce de la topología de malla. Esta población inicial se evalúa de acuerdo a la función objetivo del problema a resolver. Posteriormente da inicio, el ciclo reproductivo principal. Se define un vecindario local con individuos ubicados de forma cercana. A partir de éstos y del individuo ubicado en el cruce de la malla, se seleccionan a los padres para reproducirse. Existen distintos métodos de selección que se utilizan en los AG celulares. Se pueden utilizar los que común-

mente se aplican en los AGs con poblaciones centralizadas como el torneo binario, ruleta o selección determinista. Existen también otros métodos específicos a los AG celulares, como son la selección céntrica o anisotrópica [418, 419]. Una vez seleccionados, los padres se recombinan y los hijos se mutan. No hay diferencia entre la aplicación de los operadores de recombinación y mutación con respecto a los AG tradicionales.

Debido a la descentralización de la población, el reemplazo de ésta puede ocurrir de forma síncrona o asíncrona. El reemplazo asíncrono requiere del almacenaje temporal de la población hasta que todos los individuos concluyen su ciclo reproductivo. El reemplazo síncrono requiere de la definición del criterio para implementarla. Los distintos criterios que se han utilizado para el reemplazo se explicarán en la sub-sección 3.2.4.

3.2.1 Diversidad a partir de propiedades estructurales

Debido a la descentralización de la población y la definición de una topología de malla, las características estructurales de los AG celulares tienen un impacto importante en los procesos de búsqueda. La forma y la dimensión de la topología, el número de individuos dentro del vecindario, entre otras son características de la estructura que necesitan definirse. Además, otras operaciones durante el ciclo evolutivo como el proceso de selección local, el reemplazo de individuos, criterios de migración, entre otras, están también asociadas a estas características de estructura.

Pseudocódigo 2 AG celular

procedimiento CGA
  $(x) \leftarrow random\,(x_0)$ ▷ población inicial
  $(f) \leftarrow evaluation\,(x)$ ▷ evaluación
  mientras $k \leftarrow 1, generaciones$, or $\bar{f} <= umbral$ **hacer**
    para $i \leftarrow 1$, tamañoPoblación **hacer**
      $(f_1, f_2) \leftarrow seleccion\,(f, f_N, f_E, f_S, f_W)$ ▷ selección de padres
      $(x_1, x_2) \leftarrow seleccion\,(x, x_N, x_E, x_S, x_W)$ ▷ vecinario L5
      $(x'_1, x'_2) \leftarrow recombinacion\,(x'_1, x'_2)$ ▷ recombinación
      $(x''_1, x''_2) \leftarrow mutacion\,(x'_1, x'_2)$ ▷ mutación
      $(f_{new}) \leftarrow evaluacion\,(x''_1, x''_2)$ ▷ evaluación
      $(f, x) \leftarrow reemplazo\,(f_{new}, x_{new})$ ▷ reemplazo
    fin para
  fin mientras
fin procedimiento

La topología de malla, que es comúnmente utilizada en los AG celulares se muestra en la figura 3.2. Se ha reportado que distintas formas de malla en combinación con distintos tipos de vecindarios inducen una mayor o menor presión de selección [402, 451, 15]. Para modelar el efecto de ambas estructuras, se utiliza un patrón de dispersión con p puntos (la posición de cada solución en la malla) a partir de una posición central con coordenadas (x_0, y_0). Se calcula la medida de dispersión, debido a que otras medidas como el radio de una circunferencia, indicaría el mismo valor para topologías distintas. La relación entre el vecindario y la topología o radii se calcula de la siguiente forma:

$$D = \sqrt{\frac{\sum(x_i - \bar{x})^2 + \sum(y_i - \bar{y})^2}{p}}$$

donde $(\bar{x}) = \frac{\sum_{i=1}^{p} x_i}{p}$ y $(\bar{y}) = \frac{\sum_{i=1}^{p} y_i}{p}$. La razón o radii entre el vecindario y la topología de malla está dada por:

$$NGR = \frac{D_{vecindario}}{D_{malla}}$$

Esta medida se conoce como la razón vecindario-malla (RVM). Formas y tamaños diferentes de vecindarios y topologías proporcionan una medida distinta de RVM, ver figura 3.4.

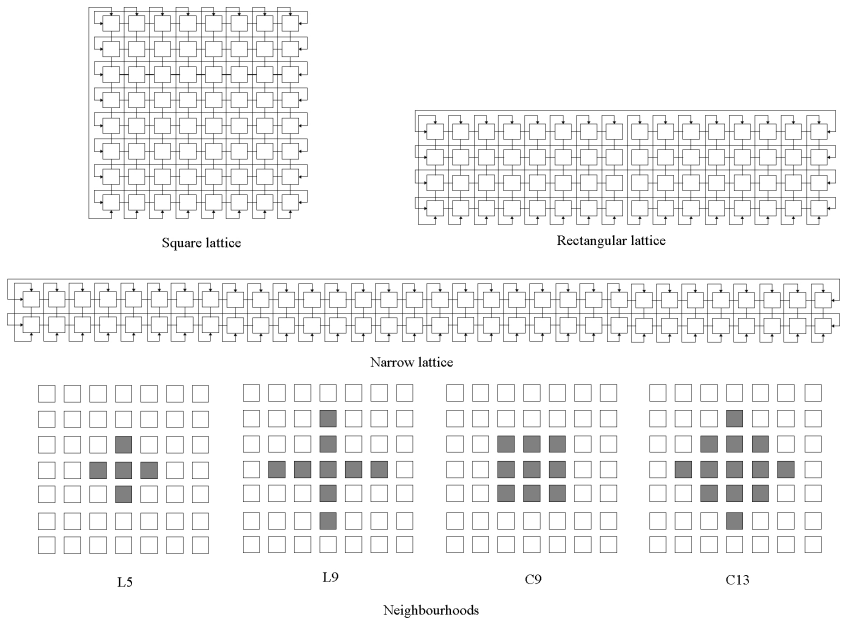


Figura 3.4: Ejemplos de formas de vecindarios y topologías de malla.

Una topología de malla cuadrada con un vecindario $L5$ es una configuración comúnmente utilizada en la implementación de AG celulares. En la figura 3.5, la razón vecindario-malla se ilustra para dis-

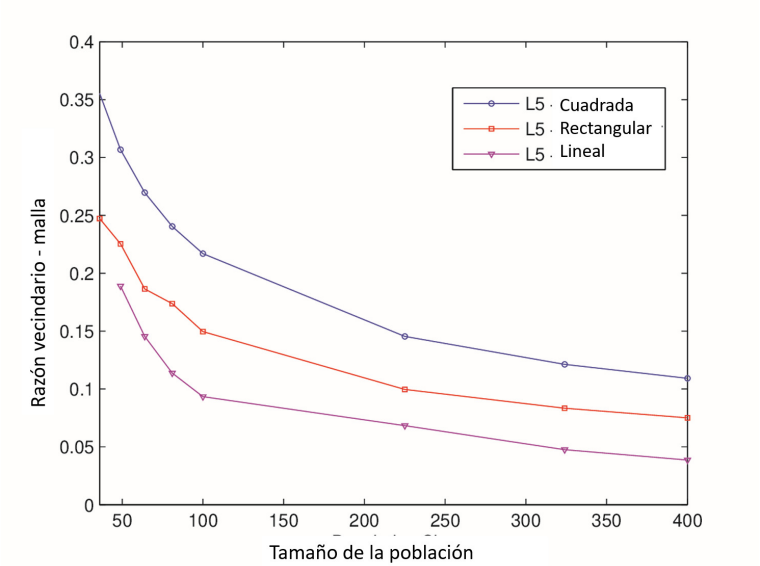


Figura 3.5: Razón vecindario-malla para diferentes topologías de malla y tamaños de población con un vecindario L5.

tintos tamaños de población con vecindario $L5$. Tres formas de malla se evalúan: cuadrada, rectangular y lineal. Las curvas de la RVM muestran razones altas para mallas cuadradas y razones bajas para topologías lineales. La RVM se asocia a la presión de selección inducida en AG celulares debido a sus características estructurales. Para comprender esto de mejor manera, otro concepto importante es necesario: el número de generaciones que toma al mejor individuo de una población inicial esparcir su solución en todas las direcciones de la topología de malla, es decir, conquistarla a partir de aplicar únicamente el operador de selección. Un número de generaciones alto indica una baja presión de selección es decir una búsqueda principalmente explorativa. Por el contrario, un número de reducido de generaciones

implica una alta presión de selección y por tanto una búsqueda orientada a la explotación del espacio de búsqueda.

Los AG celulares son técnicas flexibles con un número de parámetros que pueden configurarse. Por ejemplo, es posible pensar en modificar en tiempo de búsqueda las topologías de malla, tamaños y formas de vecindarios como una forma de control de la presión de selección inducida. Las modificaciones de la configuración pueden realizarse de manera constante, definiendo un número de generaciones o bien de forma adaptativa, midiendo generación a generación el comportamiento, por ejemplo de la diversidad en el espacio de fenotipos y genotipos, modificando entonces las configuraciones de estructura de manera que se promueva la exploración o la explotación del espacio de búsqueda.

3.2.1.1 Presión de selección por características estructurales

El matemático belga Pierre Verhulst en el siglo XIX, estudió modelos logísticos de crecimiento para describir sistemas biológicos bajo condiciones de recursos limitados. Considerando una población en panmixia de cierto tamaño $tamPob$ y un número de copias del mejor individuo en un tiempo t discreto dado por $N(t)$, la tasa de crecimiento se expresa a través del siguiente modelo de recurrencia discreto:

$N(0) = 1,$
$N(t) = N(t-1) + p_s tamPob N(t-1)\left(1 - \left(\frac{1}{tamPob}\right) N(t-1)\right)$
donde p_s es la probabilidad de selección de un individuo y $N(t)$ se puede aproximar por la ecuación logística continua [415]:

$N(t) = \frac{tamPob}{1+\left(\frac{tamPob}{N(0)}-1\right)e^{-\alpha t}}$
donde α es la probabilidad p_s. Aunque se obtienen curvas de crecimiento similares, el comportamiento en poblaciones estructuradas o descentralizadas no es exponencial sino polinomial como fue sugerido por Spiessen y Manderick in [431]. De este modo, las características estructurales necesitan considerarse en el modelo matemático de la tasa de crecimiento para AG celulares. Si se evalúa un caso límite donde el mejor individuo dentro del vecindario se selecciona siempre para reemplazar al individuo actual o central (al vecindario), es decir $p_s = 1$; y se define un tamaño de población $tamPob$, donde $\sqrt{tamPob} \times \sqrt{tamPob}$ determina una topología de malla cuadrada y un vecindario local con radio r. La tasa de crecimiento del mejor individuo está dada por:

$$
\begin{aligned}
&N(0) = 1,\\
&N(t) = N(t-1) + 4r^2t - 2r(r+1), \quad 0 \leq t \leq \frac{\left(\sqrt{tamPob}-1\right)}{2},\\
&N(t) = N(t-1) - 4r^2t + 4r\sqrt{tamPob} - 2r(r+1), \quad t \geq \frac{\left(\sqrt{tamPob}-1\right)}{2}
\end{aligned}
$$

que en su forma cerrada queda como:

$$
\begin{aligned}
&N(t) = 2r^2t^2 + 2r(2r+1)t + 1, \quad 0 \leq t \leq \frac{\left(\sqrt{tamPob}-1\right)}{2},\\
&N(t) = -2r^2t^2 + 2r\left(2\sqrt{tamPob} - 3r - 1\right)t + 1, \quad t \geq \frac{\left(\sqrt{tamPob}-1\right)}{2}
\end{aligned}
$$

En la figura 3.6 se ilustra la razón de crecimiento para una población de 81 individuos con un vecindario local tipo $C9$, ver figura 3.4.

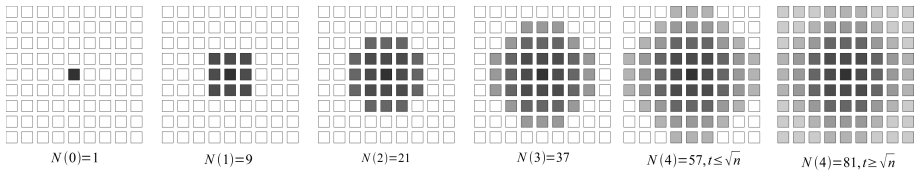


Figura 3.6: Tasa de crecimiento teórica en una topología de malla toroidal cuadrada y un vecindario local tipo Moore con probabilidad de selección $p_s = 1$

Si se considera la probabilidad variable p_s, y cada individuo tiene probabilidades de selección distintas: p_0 probabilidad para el individuo central y $p_1, p_2, p_3, p_4 ... p_8$ para cada individuo dentro de un vecindario tipo $C9$. El modelado de la recurrencia exacta se vuelve muy complicado. Para poder construir un modelo completo, se recurre a modelos más simples, en donde la tasa de crecimiento se describe como la expansión de un cuadrado rotado, con una longitud por lado de $s = \sqrt{N(t)}$, y diagonal $d = \sqrt{\frac{N(t)}{2}}$. La tasa de crecimiento de los individuos con probabilidad variable p_i contenidos dentro del cuadrado rotado está dada por la siguiente recurrencia:

$$
\begin{aligned}
&N(0) = 1, \\
&N(t) = N(t-1) + 4p_i\sqrt{\tfrac{N(t-1)}{2}}, \quad N(t) \leq \tfrac{tamPob}{2}, \\
&N(t) = N(t-1) + 4p_i\sqrt{tamPob - N(t-1)}, \quad N(t) > \tfrac{tamPob}{2}
\end{aligned}
$$

Encontrar una forma cerrada de este modelo es una tarea complicada de acuerdo a autores reconocidos del área [451]. En la figura 3.7 se muestra la tasa de crecimiento probabilista como ejemplo de la dificultad que representa su modelado matemático.

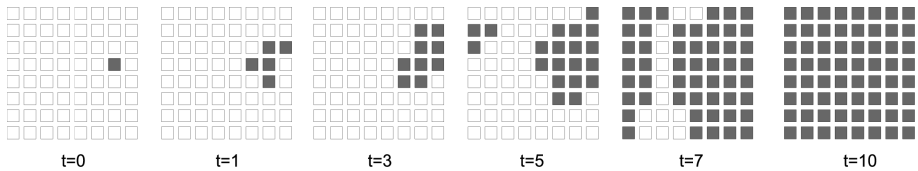


Figura 3.7: Tasa de crecimiento probabilista para una topología de malla toroidal cuadrada con vecindario local y probabilidad de selección variable p_s

3.2.1.2 Análisis empírico

Es posible tener una aproximación empírica del tiempo para el dominio de la topología de malla. A continuación se presenta un ejemplo utilizando como método de selección local, torneo binario y 50 ejecuciones por configuración fueron evaluadas. En la figura 3.8 se muestran las distintas tasas de crecimiento para la conquista de la malla, considerando un tamaño de la población de 400 individuos, con las siguientes formas de topología de malla: 1) $\sqrt{400} \times \sqrt{400}$ cuadrada, 2) $10 \times \frac{400}{10}$ rectangular y 3) $4 \times \frac{400}{4}$ lineal. La diferencia entre los tiempos de dominio es evidente. Topologías de malla lineales proveen una difusión más lenta del mejor individuo para conquistar la malla y por ende menor presión de selección y un comportamiento principalmente explorativo. Por otro lado, en topologías de malla cuadradas, la solución del mejor individuo conquista la malla rápidamente, en un número menor de generaciones. Es decir, tiene un comportamiento con alta presión de selección y por ende una búsqueda caracterizada por una mayor explotación del espacio de búsqueda. Desde este punto de vista, se ha investigado también la posibilidad de modificación dinámica de

las propiedades estructurales en tiempo de búsqueda. Se han reportado mejores desempeños algorítmicos bajo estos criterios [4].

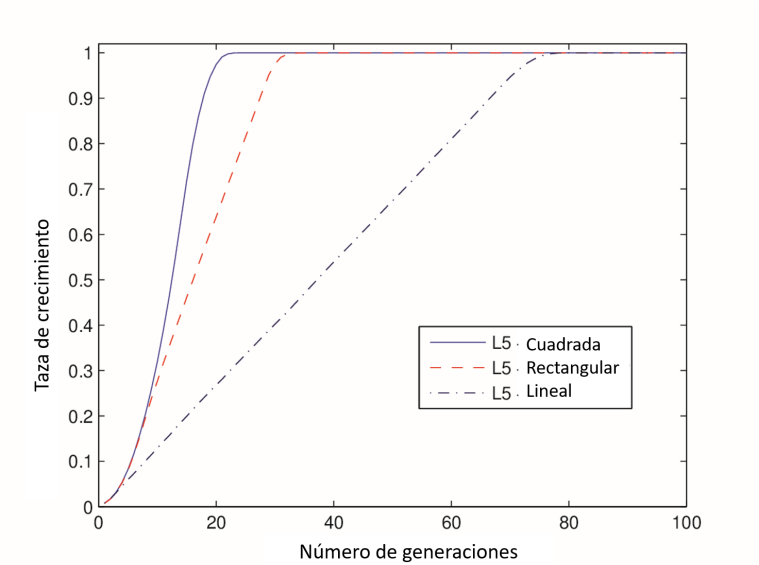


Figura 3.8: Tasas de crecimiento promedio del mejor individuo en topologías de malla cuadrada, rectangular y lineal con vecindario $L5$. La tasa promedio de crecimiento se calcula con base en 50 experimentos y aplicando únicamente el operador de selección.

3.2.2 Aceleración en AGP

Un tema controversial en AGPs son las aceleraciones que esquemas distribuidos y celulares pueden alcanzar. El cálculo teórico de estas aceleraciones, es una medida aceptada para determinar su eficiencia. Calcular la razón entre el tiempo promedio de ejecución de la mejor versión secuencial de un AG y otra técnica de búsqueda no evolutiva, con respecto al tiempo promedio de ejecución de un AG paralelo corriendo en un determinado número de procesadores, proporciona

la aceleración alcanzada por este último. Aceleraciones sub-lineales indican que esta razón es menor que el número de procesadores utilizados en la ejecución del AGP. Aceleraciones lineales indican que esta relación es igual mientras aceleraciones super-lineales indican que la razón es mayor que el número de procesadores utilizados [6].

Un medida estricta para medir las aceleraciones de los AGPs involucraría una comparación con la mejor versión secuencial de un AG, lo cual es difícil de determinar [3]. Considerar el desempeño de un AG secuencial como referencia en conjunto con una condición de paro que evalúe la misma calidad de soluciones en todas las muestras experimentales, podría permitir una comparación justa, en términos de aceleración, entre esquemas paralelos y secuenciales.

Los AGPs tienen tres características que permiten su aceleración: 1) dividir el espacio de búsqueda en diversas áreas de exploración implementadas en distintas unidades de procesamiento, 2) reducir la carga de procesamiento por medio de la distribución de la población en un número de unidades de procesamiento y 3) operadores genéticos aplicados a estructuras de datos reducidas [7].

Al analizar comparativamente las tasas de aceleración de los AG distribuidos con sub-poblaciones en panmixia en comparación con el desempeño de un AG secuencial, las aceleraciones calculadas resultan excesivas. Sin embargo, cuando se considera la distribución de las sub-poblaciones en un número de unidades de procesamiento, las tasas de aceleración son moderadas [7].

Una de las características más importantes en los AGPs es el operador de migración. Analizar el comportamiento de este operador

bajo distintos criterios como la sincronía o asincronía de migrar individuos entre las sub-poblaciones, afecta directamente el desempeño de la búsqueda. Entre los comportamientos con importancia significativa se ha observado que utilizar bajas frecuencias de migración en combinación con una comunicación asíncrona entre sub-poblaciones permite una mayor aceleración. Este resultado considera sub-poblaciones en panmixia con la misma configuración de parámetros [3].

3.2.3 Concepto de sincronismo en AGPs

En los AGPs tanto distribuidos como celulares, el concepto de sincronía aplica de forma distinta. En un AGP distribuido, el operador de migración está directamente relacionado con la comunicación entre sub-poblaciones y por ende la sincronía con que ésta ocurre. Por otro lado, en AGP de grano fino o celulares, la sincronía está relacionada con la actualización de soluciones durante el proceso evolutivo. Esto se explica a continuación a mayor detalle.

3.2.3.1 En AGPs distribuidos vía migración

Varios autores han investigado el efecto del sincronismo a través de la migración de individuos entre sub-poblaciones. Se han considerado varios escenarios tales como tener un conjunto de sub-poblaciones en panmixia y tener sub-poblaciones descentralizadas; un tercer escenario que se ha considerado es tener una combinación de sub-poblaciones en panmixia y descentralizadas [14, 10]. Los tres esquemas han sido evaluados no sólo a nivel algorítmico sino de implementación en plataformas de cómputo para procesamiento paralelo [5, 12].

Utilizando un esquema de procesamiento de instrucciones múltiples - datos múltiples, con un esquema homogéneo de sub-poblaciones, se verificó el comportamiento de distintas frecuencias de migración definidas como múltiplos del tamaño de la población. Esto representa que la migración es nula cuando la frecuencia es cero, lo que implica que la evolución es independiente de las sub-poblaciones. Localmente los operadores genéticos fueron configurados de la misma forma. En general, los resultados mostraron un mejor desempeño cuando la comunicación entre sub-poblaciones es asíncrona y la frecuencia de migración es baja. En este sentido, el tener sub-poblaciones descentralizadas permite una mejor eficacia que cuando se tienen sub-poblaciones en panmixia. Las tasas de aceleración también se analizaron. Bajas frecuencias de migración reportaron aceleraciones lineales para sub-poblaciones descentralizadas y super-lineales para sub-poblaciones en panmixia [14].

Se ha podido verificar el funcionamiento de los AGPs ampliando la gama de problemas de prueba a aquellos del dominio continuo. El uso de un esquema paralelo distribuido con sub-poblaciones en panmixia y un ciclo evolutivo de estado estable donde solamente una solución se actualiza en cada generación logró obtener soluciones de mejor calidad. Además de la influencia de la frecuencia con la que se ejecuta la operación de migración, la selección aleatoria de individuos migrantes demostró también un mejor desempeño que la selección de los mejores individuos para migrar. El mejor desempeño continúa siendo a partir de la aplicación de frecuencias de migración bajas ahora en combinación con la selección aleatoria de soluciones para migrar. En problemas

de optimización en el dominio continuo con óptimos locales engañosos en sus espacios de búsqueda, los esquemas de sub-poblaciones descentralizadas superaron el desempeño de aquellos con sub-poblaciones en panmixia, además de responder mejor a frecuencias de migración altas. Las tasas de aceleración también se relacionan con la operación de la migración; a frecuencias de migración alta se obtienen mayores tasas de aceleración en ambos esquemas poblacionales.

Es posible también considerar un esquema de AGPs con sub-poblaciones tanto en panmixia como descentralizadas, utilizando un esquema de cómputo heterogéneo distribuido. Las tasas de aceleración que se alcanzan en estos caso son super-lineales vía el esquema sub-poblacional mixto.

3.2.3.2 En AGP celulares vía criterios de actualización

En los AGPs de grano fino o celulares, el concepto de sincronismo se asocia a los criterios de actualización de las soluciones. Se consideran diversos escenarios como son: 1) barrido línea a línea, 2) barrido fijo aleatorio, 3) barrido aleatorio, 4) selección uniforme. Cada unos de estos criterios de actualización de soluciones se puede ejecutar de manera síncrona o asíncrona. La actualización síncrona se refiere a que la población completa se genera a partir de los individuos actuales. En contraste, la actualización asíncrona quiere decir que los individuos se actualizan después de su proceso evolutivo a nivel local, entonces los individuos en una generación han evolucionado de hijos generados durante esta.

El barrido línea a línea es la forma más sencilla de actualizar soluciones en una topología de malla con conexión toroidal. Haciendo el barrido secuencial de ubicación de las soluciones por línea o por columna, se ejecuta la actualización de estas. El barrido aleatorio fijo consiste en elegir de forma aleatoria una solución sin reemplazo con una probabilidad uniforme. En cada generación se determina una distribución diferente para la actualización de los individuos. La actualización uniforme elige de forma aleatoria con una probabilidad uniforme una solución con reemplazo. Esto quiere decir que una solución puede ser actualizada más de una vez en un mismo ciclo reproductivo.

También se ha buscado modelar formalmente el comportamiento de los distintos criterios de actualización de soluciones considerando ambos criterios de sincronismo [451]. Los criterios síncronos de actualización presentan menores tasas de crecimiento de la mejor solución en la topología de malla. Es decir, la búsqueda se ejecuta promoviendo la exploración cuando se sigue un criterio de actualización uniforme. Aplicar un barrido aleatorio o lineal para la actualización de soluciones representa tasas más altas de crecimiento pero no alcanza a aquellas de las poblaciones en panmixia. La principal característica que se aprecia es que las poblaciones en panmixia promueven la explotación del espacio de búsqueda en mayor grado que los AGPs descentralizados o celulares [182].

Finalmente, se puede concluir después de la evaluación de los criterios síncronos y asíncronos en una variedad de problemas de optimización combinatorios y del dominio continuo que, la actualización asíncrona de soluciones mejora el desempeño algorítmico en términos

de eficiencia y eficacia en comparación con el criterio asíncrono en términos del número de generaciones necesarias para converger a la solución del problema.

3.2.4 Discusión

En este capítulo se presentaron los distintos esquemas poblaciones que se consideran en la paralelización de los AGs. Se introdujo el principal operador que distingue a estos esquemas evolutivos: el operador de migración y su efecto en los procesos de búsqueda. También se introdujeron de manera detallada los esquemas paralelos de AGs de grano fino o celulares y la manera en que las características estructurales de estos pueden afectar directamente la exploración y explotación del espacio de búsqueda. Se abordó también el tema de las aceleraciones que se pueden alcanzar vía la paralelización de grano fino y grueso de los AGPs. Este análisis incluye los escenarios teóricos y reales de las tasas de aceleración que se pueden alcanzar. Finalmente, se revisó el concepto de sincronismo en ambos esquemas de paralelización el cual se observó que afecta de manera positiva el desempeño algorítmico.

Los esquemas paralelos de los algoritmos genéticos permiten obtener un mejor desempeño de los procesos de búsqueda. Sin embargo, se debe considerar una serie de parámetros que se necesitan definir y que requerirán la intervención de un usuario. Es importante destacar también, que la mejora en el desempeño se puede apreciar tanto a nivel algorítmico como a nivel de procesamiento. Es necesario también indicar que la selección de la plataforma de cómputo paralelo ya

sea distribuida o no, implicará costos de comunicación que deben ser considerados para las tasas de aceleración que se pretendan alcanzar.

3.2.5 Investigación de AGPs en México

En México, investigadoras e investigadores en distintas instituciones y centros de investigación se han dado a la tarea de explorar esquemas evolutivos que utilizan AGPs. Por ejemplo, esquemas de AGPs de grano fino para solucionar problemas de optimización mono-objetivo sin restricciones y problemas combinatorios, han sido desarrollados en [314, 313, 312]. El objetivo en esos trabajos es el estudio del efecto de las características estructurales en las topologías de población descentralizadas, utilizando mallas conectadas de forma toroidal. Distintos criterios fueron definidos para modificar en tiempo de búsqueda la forma de la topología y sin incrementar el costo computacional verificar la mejora del desempeño algorítmico al evaluar una cama prueba con problemas mono-objetivo sin restricciones y combinatorios. Los resultados obtenidos fueron alentadores y a partir de estos se extendió el estudio de las propiedades estructurales a manipular también la dimensión de la topología de malla y modificarla en tiempo de búsqueda, resultados positivos de este estudio fueron reportados en [315].

Debido al costo computacional de los AGs en general, la aceleración de estos utilizando plataformas de cómputo especializadas ha sido un área de investigación también desarrollada en México. Particularmente, el diseño e implementación de un arreglo de procesadores flexible para la aceleración de los AGs celulares, utilizando dispositivos reconfigurables conocidos como arreglos de compuertas programables

en el campo (FPGAs, por sus siglas en Inglés) se reportó en [252]. Este trabajo presenta una forma novedosa de dividir a la población descentralizada en un número determinado de unidades de procesamiento especialmente diseñadas para operadores evolutivos, manteniendo las conexiones toroidales entre soluciones. La flexibilidad del diseño consiste en poder definir de acuerdo a las necesidades de usuario el número de unidades de procesamiento. De este modo, un número mayor de unidades de procesamiento permitirá acelerar la ejecución del algoritmo y un número menor de estas permitirá reducir el uso de recursos de *hardware*.

En el ámbito de la optimización multi-objetivo desde la perspectiva del cómputo evolutivo utilizando esquemas poblacionales paralelos, investigadores en México han desarrollado un esquema con base en algoritmos genéticos utilizando múltiples resoluciones [271]. La propuesta algorítmica llamada "Algoritmo genético multi-objetivo con resolución múltiple" (MRMOGA por sus siglas en Inglés), consiste de un conjunto de sub-poblaciones las cuales a nivel representación de soluciones evolucionan con resoluciones distintas, lo cual implica la división del espacio de búsqueda en regiones acotadas y traslapadas. El desempeño alcanzado por MRMOGA supera los resultados reportados en esquemas evolutivos paralelos en términos de convergencia y evidencia la necesidad de profundizar el estudio de la escalabilidad de esquemas evolutivos paralelos.

Los esquemas paralelos de técnicas evolutivas en general y en particular de algoritmos genéticos paralelos han sido ampliamente estudiados por la comunidad en sus versiones distribu idas, existe sin

embargo, menos investigación en trabajos reportados en sus aproximaciones descentralizadas, de grano fino o celulares [437]. L os contextos de optimización más abordados son de un objetivo existiendo un número menor de investigaciones en las áreas de optimización con múltiples objetivos, desde esta perspect iva de distribución o descentralización de las soluciones [276]. Dentro del área, uno de los retos importantes es la exploración de esquemas di stribuidos y descentralizados que ataquen problemas con múltiples y con muchos objetivos en contextos restringidos de optimización estáticos y dinámicos [271]. En este sent ido y aunque de forma inicial, técnicas evolutivas establecidas como los algoritmos genéticos fueron los que dirigieron el desarrollo de estos esquemas paralelos. Sin embargo y como retos dentro del área, la comunidad ha explorado también otras técnicas no sólo evolutivas sino del ámbito bio-inspirado siguiendo distintas dinámicas poblacionales con resultados competitivos [516, 25, 494].

Por otro lado, el rápido desarrollo de arquitecturas *hardware-software* para procesamiento paralelo masivo ha permitido también el escalamiento de técnicas algorítmicas evol utivas en particular de algoritmos genéticos para su implementación y la verificación de desempeño en entornos condicionados en el tiempo [473, 152]. Así también, de sde la perspectiva del desarrollo de técnicas algorítmicas que se adapten a este tipo de plataformas de procesamiento, existen nichos de oportunidad importantes para el desarrollo d e técnicas evolutivas cuyas soluciones tanto en esquemas distribuidos como descentralizados tomen ventaja de estas tecnologías mejorando no sólo la calidad de solu-

ciones obtenidas s ino reduciendo los tiempos de procesamiento para llegar a ellas.

3.2.6 Retos y perspectivas

Los esquemas paralelos de técnicas evolutivas en general y en particular de algoritmos genéticos paralelos han sido ampliamente estudiados por la comunidad en sus versiones distribuidas, existe sin embargo, menos investigación en trabajos reportados en sus aproximaciones descentralizadas, de grano fino o celulares [437]. Los contextos de optimización más abordados son de un objetivo existiendo un número menor de investigaciones en las áreas de optimización con múltiples objetivos, desde esta perspectiva de distribución o descentralización de las soluciones [276]. Dentro del área, uno de los retos importantes es la exploración de esquemas distribuidos y descentralizados que ataquen problemas con múltiples y con muchos objetivos en contextos restringidos de optimización estáticos y dinámicos [271]. En este sentido y aunque de forma inicial, técnicas evolutivas establecidas como los algoritmos genéticos fueron los que dirigieron el desarrollo de estos esquemas paralelos. Sin embargo y como retos dentro del área, la comunidad ha explorado también otras técnicas no sólo evolutivas sino del ámbito bio-inspirado siguiendo distintas dinámicas poblacionales con resultados competitivos [516, 25, 494].

Por otro lado, el rápido desarrollo de arquitecturas *hardware-software* para procesamiento paralelo masivo ha permitido también el escalamiento de técnicas algorítmicas evolutivas en particular de algoritmos genéticos para su implementación y la verificación de desempeño en en-

tornos condicionados en el tiempo [473, 152]. Así también, desde la perspectiva del desarrollo de técnicas algorítmicas que se adapten a este tipo de plataformas de procesamiento, existen nichos de oportunidad importantes para el desarrollo de técnicas evolutivas cuyas soluciones tanto en esquemas distribuidos como descentralizados tomen ventaja de estas tecnologías mejorando no sólo la calidad de soluciones obtenidas sino reduciendo los tiempos de procesamiento para llegar a ellas.

3.2.7 Para saber más

Dentro de la comunidad científica dedicada a la investigación en las áreas de las técnicas algorítmicas evolutivas que desarrollan el concepto de paralelismo, se han generado recursos bibliográficos diversos de entre los cuales destacan:

- Algoritmos genéticos paralelos por Gabriel Luque y Enrique Alba [277].
- Algoritmos evolutivos estructurados espacialmente por Marco Tomassini [451].
- Algoritmos genéticos celulares por Bernabé Dorronsoro y Enrique Alba [11].
- Algoritmos genéticos paralelos eficientes y precisos por Erick Cantú-Paz [72].

Por otro lado, entre los grupos de investigación que se desarrollan en el área destaca el liderado por el investigador Enrique Alba de la

Universidad de Málaga en España, quien en colaboración con otros investigadores y estudiantes, a puesto a disposición una serie de recursos relacionados en [328]. En particular, en la sección de *software* se encuentran disponibles herramientas para el diseño e implementación de algoritmos evolutivos paralelos tanto de grano grueso como de grano fino.

Capítulo 4

Programación Genética

4.1 Introducción

En este capítulo se presenta una introducción a Programación Genética (PG) así como la descripción de cada uno de los aspectos que la comprenden, las perspectivas de este campo y las áreas donde se ha aplicado en México. PG es un algoritmo de búsqueda inspirado en los principios de la evolución de las especies de Charles Darwin (ver [240, 367]). Particularmente, PG engloba a un conjunto de técnicas de cómputo evolutivo, las cuales tienen como objetivo solucionar automáticamente problemas partiendo de ejemplos del problema por resolver a través de expresiones evaluables o programas.

La Programación Genética ha sido ampliamente utilizada en diferentes dominios, siendo capaz de resolver una gran variedad de problemas de alta complejidad muchas veces mejorando las soluciones encon-

tradas por humanos. Por ejemplo, en la competencia –"Humies[1]" celebrada anualmente en el marco de la *Genetic and Evolutionary Computation Conference (GECCO)*, la cual premia sistemas bio-inspirados que mejoran las soluciones encontradas por humanos a problemas difíciles —desde 2004 y hasta 2017, PG ha sido galardonada con 8 medallas de oro, 5 de plata y 1 de bronce. En particular, en esta competencia y utlizando PG, México ha obtenido una medalla de bronce [454]. En competencias de categorización de texto se ha obtenido primer lugar en identificación de humor [349] y en detección de agresividad [188].

En México, la PG ha sido estudiada y aplicada sobre una gran variedad de dominios, empezando por aplicaciones tradicionales como el diseño de circuitos lógicos [2], regresión simbólica [428], identificación de sistemas [391, 153], modelado de sistemas de manchas solares [392], lluvia [389], temperatura del agua [24], turbinas de gas [27, 143, 396], absorción de partículas [504], medición de la capacidad de automovilistas para el manejo de vehículos [268, 267], sistemas caóticos [390], detección de fallas en transformadores de potencia [76] y modelado cualitativo de sistemas [413].

La PG ha probado su efectividad con respecto a algoritmos tradicionales de aprendizaje de máquina en problemas como predicción de series de tiempo [187, 74], clasificación [325, 192], generación de ensambles de clasificadores [1], generación de prototipos [462, 138, 137, 394], generación de heurísticas [427, 425, 342], navegación autónoma [326], generación de rejillas topográficas [387] y memorias asociativas [476]. En el área de procesamiento de lenguaje natural, la PG ha sido uti-

[1] http://www.human-competitive.org

lizada para mejorar la representación del texto en un espacio vectorial [139], aplicada a problemas de categorización de texto en análisis de sentimientos [191], intensidad de una emoción [188], identificación de humor [349] y detección de agresión [188], entre otras. En al área de visión computacional, la PG ha servido para desarrollar nuevas representaciones siguiendo un enfoque de redes profundas [388], representación de palabras visuales [139], estimación de exponentes Hölder en imágenes [454], detección de puntos de interés [456, 334, 333], reconocimiento de objetos [332], análisis de imágenes hiper-espectrales [125] y estimación de vegetación [374]. En el país también se realiza investigación teórico-práctica sobre la creación de modelos que permitan predecir su rendimiento como en [288, 189, 190], métodos de muestreo en la función objetivo [287], mejoras al algoritmo como en [266, 169] y control de código superfluo [455]. El lector interesado en la teoría de PG puede revisar los siguientes trabajos paradigmáticos [368, 433, 369], que son referentes indispensables sobre el tema.

La idea detrás de la PG es evolucionar programas partiendo de una descripción de lo que debe hacer el programa. Tradicionalmente, esta descripción se realiza mediante un conjunto de pares de entrada-salida, tal como se haría con la descripción de una función. Esta descripción es similar a la usada en *Aprendizaje Supervisado*, con la diferencia de que en PG no existen tipos definidos para las entradas y las salidas; por otro lado, en aprendizaje supervisado, las entradas están en un espacio vectorial y su salida es un número natural o real. Utilizando esta descripción, la funcionalidad de un programa se define como el conjunto $\mathcal{X} = \{(x_0, y_0), \ldots, (x_i, y_i), \ldots, (x_N, y_N)\}$, donde x_i

representa la i-ésima entrada y y_i es su correspondiente salida. En cierta forma, la idea es encontrar un programa f tal que aplicado a un conjunto de entradas x_i se evalúe a las predicciones $\hat{y}_i$ de tal forma que éstas se aproximen a los valores originales y_i, donde la similitud se mide por medio de una función de aptitud. Tal y como sucede en otros algoritmos de aprendizaje, la efectividad de f no estará en aprender los datos de entrenamiento, sino en su capacidad de generalizar a entradas nunca antes vistas. El procedimiento de aprendizaje y predicción se detallará en el resto de este capítulo.

La PG comparte muchas similitudes con los Algoritmos Genéticos (AG). De hecho, se podría decir que la PG es una generalización del AG; la diferencia radica en que la PG utiliza una representación variable para sus soluciones y el AG tiene una representación fija; esta diferencia tiene varias implicaciones las cuales se verán a detalle en el transcurso del capítulo. Por el momento, veamos las similitudes que comparten ambos enfoques. La PG es un algoritmo evolutivo poblacional, es decir, utiliza un conjunto de soluciones, llamado *población*, para hacer un muestreo del espacio de búsqueda. La búsqueda es guiada mediante la selección de individuos en la población y los operadores genéticos, como son la recombinación y la mutación, se encargan de generar nuevos individuos, es decir, nuevos puntos en el espacio de búsqueda.

El resto del capítulo detalla un sistema de PG típico, empezando por los tipos de evolución que pueden realizar (sección 4.2), la representación que se utiliza (sección 4.3), seguida por la creación de la población inicial (sección 4.4), los operadores que se utilizan para

generar un individuo (sección 4.5) y finalmente, las técnica utilizada para hacer la selección (sección 4.6).

4.2 Proceso evolutivo

La evolución de la población de programas en la PG puede realizarse de dos maneras: mediante una evolución *generacional* o mediante una evolución de *estado estable*. Ambos tipos de evolución requieren una población inicial, y la diferencia entre ellas radica en el uso de dicha población para generar individuos y la actualización de la misma. En la evolución *generacional* existen dos tipos de poblaciones: la de los padres y la de los nuevos individuos. La población de padres es utilizada para generar nuevos individuos y, por lo general, cuando la población de nuevos individuos tiene el mismo número de elementos que la población de padres, se procede a mezclar estas poblaciones para convertirse en la población de padres de la siguiente generación. Por otro lado, la evolución de *estado estable* no contempla esta diferencia entre población de padres y de nuevos individuos, dado que el individuo generado reemplaza un individuo de la población actual.

El algoritmo 3 muestra el pseudocódigo de la evolución generacional. Como se había mencionado, se requiere de una población inicial, denominada $\mathcal{P}$, la cual será utilizada para generar individuos y será actualizada mientras no se cumpla un Criterio de Paro establecido. Lo primero que se observa en el algoritmo son dos ciclos anidados (líneas 1 y 2). El primero itera hasta que se cumpla la condición de paro y el segundo se encarga de generar la población de hijos (línea 3),

$\mathcal{P}_h$. Al final de este segundo ciclo, se tienen dos poblaciones $\mathcal{P}$ y $\mathcal{P}_h$. Estas dos poblaciones se combinan (línea 5) para formar una nueva población $\mathcal{P}$ utilizada en la siguiente generación. La función Mezcla combina estas poblaciones, y puede ser tan simple como seleccionar todo $\mathcal{P}_h$ para reemplazar $\mathcal{P}$, o pueden adoptarse estrategias más complejas como remplazo con elitismo, donde se realiza el remplazo pero se asegura que los mejores individuos de ambas poblaciones se preserven en la siguiente generación.

Algoritmo 3 Evolución generacional

Entrada: $\mathcal{P}$ ▷ Población

1: **mientras no** Criterio de Paro($\mathcal{P}$) **hacer**
2: **para** $i = 0$ hasta $|\mathcal{P}|$ **hacer**
3: $\mathcal{P}_h(i) \leftarrow$ Operadores Genéticos($\mathcal{P}$) ▷ Ver algoritmo 7
4: **fin para**
5: $\mathcal{P} \leftarrow$ Mezcla($\mathcal{P}, \mathcal{P}_h$)
6: **fin mientras**
7: **devolver** $\mathcal{P}$

La evolución generacional es muy utilizada tanto en computación evolutiva como en PG. Sin embargo, hasta la fecha no existen ventajas teóricas para decidir utilizar un tipo de evolución sobre el otro. Considerando lo anterior, este capítulo se enfocará en el uso de evolución de estado estable debido a que su implementación es más simple.

El algoritmo 4 muestra el pseudocódigo de una evolución de estado estable. $\mathcal{P}$ es utilizada para generar un individuo, *hijo*, mediante la aplicación de operadores genéticos (línea 2). El hijo reemplaza (línea 4) a un elemento de $\mathcal{P}$ que es seleccionado utilizando una selección negativa, es decir, se selecciona un individuo con una aptitud baja (línea

3). Este proceso continúa hasta que el criterio de paro es alcanzado. Dicho criterio puede ser variado, tal como alcanzar un número máximo de individuos generados, convergencia de una función de error, encontrar una solución adecuada o suficiente, entre otras opciones.

Algoritmo 4 Evolución estado estable

Entrada: $\mathcal{P}$ ▷ Población
1: **mientras no** Criterio de Paro($\mathcal{P}$) **hacer**
2: hijo ← Operadores Genéticos($\mathcal{P}$) ▷ Ver algoritmo 7
3: k ← Selección Negativa($\mathcal{P}$) ▷ Ver algoritmo 11
4: $\mathcal{P}(k)$ ← hijo ▷ Reemplaza el k-ésimo individuo
5: **fin mientras**
6: **devolver** $\mathcal{P}$

4.3 Representación

Tomando en cuenta que el objetivo de PG es evolucionar programas, es normal pensar que la PG evoluciona programas escritos en algún lenguaje de programación, p.ej., Python, utilizando un intérprete o compilador y máquina que lo ejecute, dependiendo del caso. Aunque esto es factible y existen sistemas de PG que han seguido este camino, no es lo más común. En este capítulo seguimos una de las prácticas más comunes, popularizada en el libro de Koza [240], la cual consiste en representar un programa utilizando un árbol de expresión. Un árbol de expresión cuenta con tres nodos distintivos: la raíz la cual es la salida de la expresión, los nodos internos y las hojas. La función de la raíz y los nodos internos es aplicar operaciones a sus hijos y las hojas representan las entradas, ya sean variables o constantes. Un

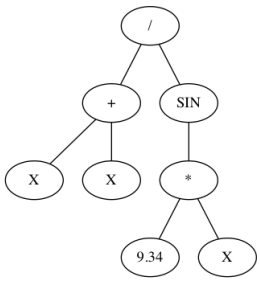


Figura 4.1: Expresión $\frac{x+x}{\sin(9.34x)}$ representada en un árbol de expresiones.

ejemplo de un árbol de expresión se muestra en la figura 4.1, donde la expresión $\frac{x+x}{\sin(9.34x)}$ es representada en el árbol.

Los árboles de expresiones se forman partiendo de dos conjuntos: uno de terminales, $\mathcal{T}$, que contiene las variables, constantes y funciones sin argumentos; y otro de funciones, $\mathcal{F}$, que contiene las funciones u operadores que reciben argumentos. Por ejemplo, en una regresión simbólica, la cual consiste en encontrar la función que más se asemeja a una serie de puntos, el conjunto de terminales podría ser $\mathcal{T} = \{x, \Re\}$, donde x es la variable independiente y $\Re$ representa un número aleatorio; y el conjunto de funciones podría ser: $\mathcal{F} = \{+, -, \cdot, /, \sqrt{}, \sin, \cos\}$. Continuando con este ejemplo, se puede observar que la expresión representada en la figura 4.1 pudo haber sido generada con los mencionados conjuntos de terminales y funciones.

Una codificación simple de un árbol de expresión puede ser representado en un arreglo o una lista, es decir, un árbol de expresión puede estar codificado de manera lineal. Esta codificación se logra recorriendo el árbol a lo profundo. Por ejemplo, la expresión representada en la figura 4.1 puede ser representada como $(/, +, x, x, \sin, \times, 9.34, x)$.

Esta codificación será la utilizada en este capítulo para representar los árboles de expresiones.

Dado un árbol de expresión se requiere tener un procedimiento para su evaluación. Dicho procedimiento se muestra en el algoritmo 5, Evaluación, el cual recibe como entrada un árbol de expresión representado en una estructura de datos lineal como un arreglo o lista. La expresión es codificada en $\mathbf{x}$, donde $\mathbf{x}_i$ es la i-ésima posición, e i es una variable global. Evaluación es una función recursiva, donde la recursión termina cuando $\mathbf{x}_i$ es una constante o variable (ver líneas 15-19). Cuando $\mathbf{x}_i$ es una función (v.g., $\mathbf{x}_i \in \mathcal{F}$), línea 1, se debe de conocer la cantidad de argumentos que requiere, línea 3, y empieza la recursión. En caso de que requiera un argumento, líneas 4-7, se incrementa la posición i en uno y se llama a Evaluación terminando con la evaluación de la función $\mathbf{x}_i$ en el valor regresado en la recursión. Para más argumentos, líneas 9-14, se invoca Evaluación tantas veces como argumentos requiera la función, y el valor es guardado en una estructura, $\mathbf{w}$, que podría ser un arreglo o lista. La línea 14 evalúa la función f, v.g., $\mathbf{x}_i$, en los argumentos $\mathbf{w}$. Por ejemplo, sea f la suma, entonces la línea 14 realizaría $\mathbf{w}_0 + \mathbf{w}_1$.

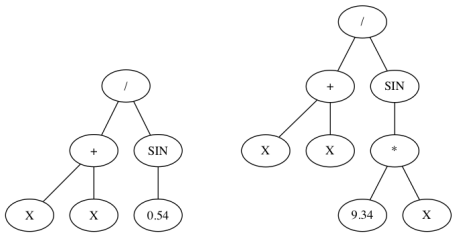
4.4 Población inicial

Una vez descrita la representación que tendrán los individuos dentro de la población por medio de árboles de expresiones, la codificación de éstos mediante una estructura de datos lineal y el pseudocódigo para

Algoritmo 5 Evaluación

Entrada: $\mathbf{x}$ ▷ Individuo
Entrada: i ▷ Posición, variable global

si $\mathbf{x}_i$ es una función **entonces**
    $f \leftarrow \mathbf{x}_i$
    $d \leftarrow \#\text{argumentos}(f)$
    si $d = 1$ **entonces**
        $i \leftarrow i + 1$
        $w \leftarrow \text{Evaluación}(\mathbf{x})$
        devolver $f(w)$
    fin si
    $\mathbf{w}$ ▷ Estructura de datos, p.ej., arreglo o lista
    para $j = 0$ hasta d **hacer**
        $i \leftarrow i + 1$
        $\mathbf{w}_j \leftarrow \text{Evaluación}(\mathbf{x})$
    fin para
    devolver $f(\mathbf{w})$
si no si $\mathbf{x}_i$ es una variable **entonces**
    devolver el valor de la variable $\mathbf{x}_i$
si no
    devolver la constante $\mathbf{x}_i$
fin si



(a) Balanceado (b) Desbalanceado

Figura 4.2: Árboles de expresiones.

evaluar un individuo, ya es posible describir el procedimiento para crear la población inicial.

En PG existen dos procedimientos tradicionales para generar un individuo: 1) generar un árbol de expresiones balanceado (*Full*) o 2) generar un árbol desbalanceado (*Grow*); ambos se definen con una altura máxima. La definición de un árbol balanceado es aquel donde los hijos de cualquier nodo tienen siempre la misma altura. La figura 4.2 muestra un árbol balanceado y un árbol no balanceado. En la figura 4.2a se observa cómo la definición de balanceo solo considera la altura sin importar el número de nodos. Por ejemplo, la raíz tiene tres nodos en su hijo izquierdo y dos nodos en su hijo derecho y es un árbol balanceado.

Estos dos procedimientos para generar un individuo comparten muchas similitudes. Por esta razón, el algoritmo 6 presenta ambos procedimientos, decidiendo cuál utilizar mediante una bandera, Método. El proceso de generar un individuo utiliza el conjunto de funciones y terminales y la altura deseada h. Éste es un procedimiento recursivo donde la recursión termina cuando la altura $h = 0$ llega a su valor mínimo (líneas 1-3) o en el caso en que se utilice el método desbalanceado,

(líneas 3-5) y un número aleatorio sea menor que $\frac{|\mathcal{T}|}{|\mathcal{T}|+|\mathcal{F}|}$. Si ninguna de las dos condiciones se cumplen, empieza la recursión, primero decrementando h (línea 6), después seleccionando de manera aleatoria una función del conjunto $\mathcal{F}$ (línea 7) y contando sus argumentos (línea 8). La estructura, **w**, definida en la línea 9 es la encargada de guardar los componentes de la regresión. Dicha estructura podría ser una lista o un conjunto que mantiene el orden de inserción. El procedimiento continúa creando un ciclo del número de argumentos (líneas 10-12), donde se hace la recursión y el resultado de la recursión se añade a **w**, línea 11. Finalmente el individuo creado está en **w** y se regresa en la línea 13.

Algoritmo 6 Creación de Individuo

Entrada: $\mathcal{F}$ ▷ Conjunto de funciones
Entrada: $\mathcal{T}$ ▷ Conjunto de terminales
Entrada: h ▷ Altura
Entrada: Método
1: **si** $h = 0$ **entonces**
2: **devolver** $x \in \mathcal{T}$
3: **si no si** Método es desbalanceado y Número Aleatorio() $< \frac{|\mathcal{T}|}{|\mathcal{T}|+|\mathcal{F}|}$ **entonces**
4: **devolver** $x \in \mathcal{T}$
5: **fin si**
6: $h \leftarrow h - 1$
7: $f \in \mathcal{F}$
8: $d \leftarrow$ #argumentos(f)
9: $\mathbf{w} = [f]$ ▷ Estructura de datos, e.g. lista
10: **para** $j = 0$ hasta d **hacer**
11: $\mathbf{w} \leftarrow \mathbf{w} \cup$ Creación de Individuo$(h,$ Método$)$
12: **fin para**
13: **devolver** **w**

Es probable que uno de los métodos más tradicionales para crear la población inicial sea el método *ramped half-and-half*, propuesto por Koza [240], el cual consiste en construir la mitad de la población utilizando un método balanceado y la otra mitad utilizando un método desbalanceado. Además de lo anterior, la altura del árbol varía desde un valor de altura mínima hasta un valor máximo de manera lineal tomando en cuenta el tamaño de la población.

4.5 Operadores Genéticos

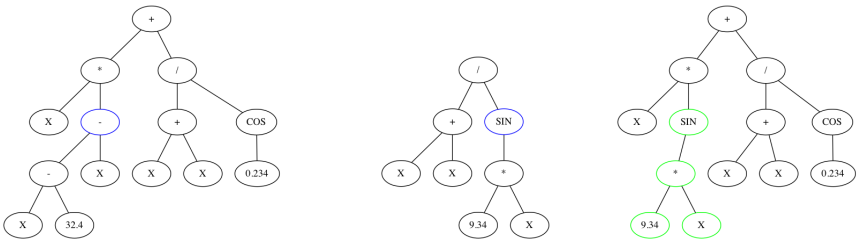
Habiendo descrito dos procedimientos para la creación de la población inicial, es tiempo de introducir los operadores genéticos que permitirán la creación de nuevos individuos basados en la población actual. En PG se acostumbra tener dos operadores genéticos: mutación (ver Sección 4.5.2) y recombinación (ver Sección 4.5.1), que a diferencia del AG, son excluyentes y seleccionados de manera aleatoria.

El algoritmo 7 muestra el procedimiento para generar un individuo utilizando mutación y recombinación. Lo primero que se realiza es la selección de los individuos que servirán como padres en este proceso (línea 1 y línea 3). Esto es seguido por la decisión de generar un individuo mediante mutación o recombinación, la cual se realiza de manera aleatoria y se controla con un parámetro: la probabilidad de recombinación p_{xo} (línea 2). En caso de aplicarse recombinación, se requieren dos elementos (línea 4) y en el caso de mutación solamente se requiere un elemento (línea 6).

Algoritmo 7 Operadores genéticos

Entrada: $\mathcal{P}$ ▷ Población

1: $a \leftarrow$ Selección$(\mathcal{P})$ ▷ Ver algoritmo 11
2: **si** Número Aleatorio() $< p_{xo}$ **entonces**
3: $b \leftarrow$ Selección$(\mathcal{P})$ ▷ Ver algoritmo 11
4: $\mathbf{w} \leftarrow$ Recombinación$(\mathcal{P}(a)), \mathcal{P}(b))$ ▷ Ver algoritmo 9
5: **si no**
6: $\mathbf{w} \leftarrow$ Mutación$(\mathcal{P}(a))$ ▷ Ver algoritmo 10
7: **fin si**
8: **devolver** w



(a) Primer padre (b) Segundo padre (c) Hijo

Figura 4.3: Proceso de recombinación. Los nodos azules representan los puntos de recombinación y el hijo se construye reemplazando el subárbol del primer padre con el subárbol del segundo padre.

4.5.1 Recombinación

La recombinación es la operación genética que toma dos individuos de la población, denominados padres, y los combina para generar un nuevo individuo. Tomando en cuenta que los individuos son árboles de expresiones, entonces la recombinación se plantea como la generación de un nuevo árbol de expresión. La idea es escoger en cada árbol un punto de recombinación e intercambiar los subárboles cuya raíz sean dichos puntos de recombinación.

La figura 4.3 muestra el procedimiento de recombinación. En dicha figura, los dos padres son los primeros dos árboles de la izquierda, ver figuras 4.3a y 4.3b. El nuevo individuo es el árbol de la derecha, ver figura 4.3c; es común llamar hijo al producto de la recombinación. Los nodos de color azul representan los puntos de recombinación, siendo el procedimiento para generar un nuevo individuo el cambiar los subárboles cuya raíz son los nodos en azul. En particular, aquí se crea solamente un individuo, el cual es generado al reemplazar el subárbol del primer padre por el subárbol del segundo padre.

El procedimiento de recombinación requiere de un método que permita conocer donde termina un subárbol dada su raíz. Esto es debido a que se está codificando un árbol de expresión en una estructura lineal. Este procedimiento es simple y solamente se requiere el contar el número de argumentos por cada función encontrada en el camino. El algoritmo 8 presenta el pseudocódigo para hacer el recorrido por el subárbol. Este procedimiento recibe un individuo y la posición de la raíz del subárbol de interés. El ciclo de las líneas 2-12, realiza todo el recorrido del subárbol terminando cuando las funciones encontradas en el recorrido han satisfecho todos sus argumentos, líneas 9-11, regresando la posición del último nodo del subárbol. En cada iteración se sabe que se consume un argumento, línea 5, y se incrementa la posición final del subárbol, línea 4. Además se verifica si el nodo actual corresponde a una función. En caso afirmativo, se incrementa el número de argumentos, líneas 6-8, y todo este proceso continua hasta que los argumentos requeridos por la funciones han sido cubiertos.

Algoritmo 8 Recorre

```
Entrada: x                                          ▷ Individuo
Entrada: i                                          ▷ Posición
 1: c ← 1                                           ▷ Número de argumentos
 2: mientras Verdadero hacer
 3:     x ← x_i
 4:     i ← i + 1
 5:     c ← c − 1
 6:     si x es una función entonces
 7:         c ← c + #argumentos(x)
 8:     fin si
 9:     si c = 0 entonces
10:         devolver i
11:     fin si
12: fin mientras
```

Ahora que se conoce la función auxiliar, Recorre, es momento de describir el pseudocódigo de la recombinación, el cual se muestra en el algoritmo 9. Lo primero que se puede observar es que la recombinación requiere dos individuos, $\mathbf{x}, \mathbf{y}$; el segundo paso es seleccionar un nodo de cada uno de estos individuos, líneas 1 y 2. Las líneas 3 y 4 calculan las posiciones donde terminan cada uno de los subárboles que corresponden a los nodos seleccionados. Finalmente, el individuo generado corresponde al reemplazo de la subcadena que representa el subárbol del primer padre con la subcadena que representa el subárbol del segundo padre, ver línea 5.

Algoritmo 9 Recombinación

Entrada: $\mathbf{x}$ ▷ Primer padre
Entrada: $\mathbf{y}$ ▷ Segundo padre
1: $i \leftarrow$ Punto de recombinación$(\mathbf{x})$
2: $j \leftarrow$ Punto de recombinación$(\mathbf{y})$
3: $i_k \leftarrow$ Recorre$(\mathbf{x}, i)$ ▷ Ver algoritmo 8
4: $j_k \leftarrow$ Recorre$(\mathbf{y}, j)$ ▷ Ver algoritmo 8
5: **devolver** $\mathbf{x}_{0,\ldots,i} \cup \mathbf{y}_{j,\ldots,j_k} \cup \mathbf{x}_{i_k,\ldots,|\mathbf{x}|}$

4.5.2 Mutación

Existen diferentes tipos de mutación. Una de las más utilizadas es la *mutación de subárbol*, la cual consiste en hacer recombinación de un individuo de la población con un individuo aleatorio. La idea es reemplazar un subárbol del individuo de la población por un subárbol aleatorio y esto se logra utilizando métodos previamente definidos.

La mutación de subárbol se presenta en el algoritmo 10. Este procedimiento requiere un padre, $\mathbf{x}$. Después se genera un individuo aleatorio utilizando el método Creación de Individuo. Por lo general, se usa una altura menor a siete y después se regresa la Recombinación de los individuos (línea 2).

Algoritmo 10 Mutación

Entrada: $\mathbf{x}$ ▷ Padre
1: $\mathbf{y} \leftarrow$ Creación de Individuo$(4, \text{Balanceado})$ ▷ Ver algoritmo 6
2: **devolver** Recombinación$(\mathbf{x}, \mathbf{y})$ ▷ Ver algoritmo 9

La mutación de subárbol no es el único tipo de mutación. Otro tipo de mutación muy utilizada es la mutación de punto, que tiene su homólogo en el AG. La mutación de punto funciona iterando por todos

los nodos del individuo y seleccionando aleatoriamente algunos nodos para ser mutados. Un nodo seleccionado para ser mutado cambia su valor de la siguiente manera. Un nodo que tiene una función cambia la función por otra de la misma cardinalidad. Por otro lado, un nodo que tiene una terminal cambia su valor por otra terminal seleccionada del conjunto de terminales.

4.6 Selección

Hasta este momento hemos revisado los procedimientos para hacer una evolución de estado estable, generar una población inicial y modificar la población utilizando operadores genéticos. En esta sección se muestra el proceso de selección de los individuos padres que generarán mediante operadores genéticos un nuevo individuo.

La selección, como su nombre lo indica, debe escoger un individuo de la población con base en su *aptitud*; es decir, se busca aquel individuo que resuelve de mejor manera el problema. En el caso de la *selección negativa* se busca el individuo que resuelve el problema de la peor manera.

Recordemos que la forma en que se describe la funcionalidad de un programa es mediante un conjunto de pares entrada-salida, v.g., $\mathcal{X} = \{(x_0, y_0), \ldots, (x_i, y_i), \ldots, (x_N, y_N)\}$, donde x_i representa la i-ésima entrada y y_i es su salida correspondiente. Utilizando $\mathcal{X}$ se puede definir la aptitud de un programa p de la siguiente manera.

$$\mathsf{Aptitud}(p) = \sum_{(x,y)\in\mathcal{X}} \mathsf{Mide\ Approximación}(y, p(x)) \tag{4.1}$$

donde $p(x)$ representa la salida del programa p cuando la entrada es x y Mide Approximación es una función que captura que tanto se parece la predicción $p(x)$ a la salida esperada y.

Existen varios procedimientos de selección. Uno de los más comunes es la selección proporcional o por ruleta. La idea de esta selección es asignar a cada individuo una probabilidad de ser seleccionado directamente proporcional a su aptitud. Otro tipo de selección muy utilizado en la comunidad es la selección por torneo, donde la idea es seleccionar al ganador de un grupo de individuos seleccionados aleatoriamente. Al tamaño del grupo se le conoce como el tamaño del torneo.

El algoritmo 11 presenta el pseudocódigo de la selección por torneo. El procedimiento requiere la población, $\mathcal{P}$, y el tamaño del torneo, t. Se empieza seleccionando un individuo de la población, v.g., el k-ésimo, para el cual se obtiene su aptitud g_a, líneas 1 y 2. El individuo $\mathcal{P}(k)$ es momentáneamente el mejor elemento conocido. El ciclo principal ejecuta el torneo, líneas 3-10. En cada iteración se selecciona un elemento de la población junto con su aptitud (línea 4 y 5). En caso que el nuevo individuo tenga una mejor aptitud que la conocida, éste se convierte en el ganador del torneo, líneas 6-9. El procedimiento termina regresando la posición del mejor individuo.

4.7 Espacio de búsqueda

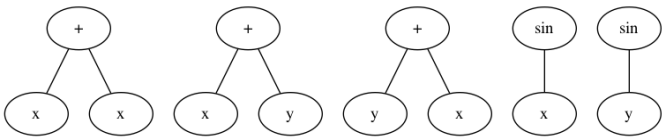
La PG es un algoritmo de búsqueda inspirado en los principios de la evolución. En esta sección se describe cómo es el espacio de búsqueda, Ω. Existen dos diferencias principales entre la PG y el AG: 1) la PG

Algoritmo 11 Selección

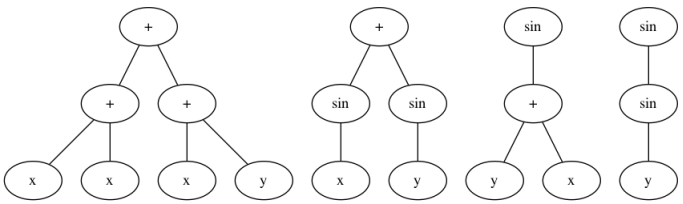
Entrada: $\mathcal{P}$ ▷ Población
Entrada: t ▷ Tamaño del torneo
1: $k \leftarrow$ Número Aleatorio() ▷ $0 \leq$ Número aleatorio entero $< |\mathcal{P}|$
2: $g_a \leftarrow$ Aptitud$(\mathcal{P}(k))$ ▷ Ver ecuación 4.1
3: **para** $i = 1$ **hasta** t **hacer**
4: $c_k \leftarrow$ Número Aleatorio()
5: $c_a \leftarrow$ Aptitud$(\mathcal{P}(c_k))$ ▷ Ver ecuación 4.1
6: **si** $c_a > g_a$ ▷ En selección negativa se cambia la comparación, i.e. $c_a < g_a$ **entonces**
7: $k \leftarrow c_k$
8: $g_a \leftarrow c_a$
9: **fin si**
10: **fin para**
11: **devolver** k

busca en un espacio posiblemente infinito de programas, mientras que el AG en un espacio finito; y 2) las soluciones de la PG tienen una codificación de longitud variable y las del AG son de longitud fija.

El espacio de búsqueda de la PG se construye partiendo del conjunto de funciones, $\mathcal{F}$, y del conjunto de terminales, $\mathcal{T}$, de la siguiente manera. Sea $\mathcal{F}_c \subseteq \mathcal{F}$ el subconjunto de funciones con cardinalidad c del conjunto de funciones; y Ω^i los elementos creados en la i-ésima iteración, iniciando en $i = 0$. Usando esta notación, los primeros elementos del espacio de búsqueda, v.g., $i = 0$, son $\Omega^0 = \bigcup_c \{f(x_1, \cdots, x_c) \mid x_i \in \mathcal{T}, f \in \mathcal{F}_c\}$. El resto de los elementos se construyen iterativamente usando $\Omega^i = \bigcup_c \{f(x_1, \cdots, x_c) \mid x_i \in \Omega^{i-1}, f \in \mathcal{F}_c\}$; por lo tanto, el espacio de búsqueda está definido como $\Omega = \bigcup_i \Omega^i$.



(a) Todos los elementos de Ω^0.



(b) Algunos elementos de Ω^1.

Figura 4.4: Ejemplo del espacio de búsqueda cuando $\mathcal{T} = \{x, y\}$ y $\mathcal{F} = \{+, \sin\}$.

Con el objetivo de ejemplificar la estructura del espacio de búsqueda, la figura 4.4 presenta unos elementos del espacio de búsqueda cuando el conjunto de terminales es $\mathcal{T} = \{x, y\}$ y el conjunto de funciones es $\mathcal{F} = \{+, \sin\}$. La figura 4.4a muestra todos los elementos de Ω^0, los cuales corresponden a las combinaciones entre el operador + y las entradas, así como a la función sin y las posibles entradas. La parte inferior de la figura muestra algunos elementos de Ω^1. Siguiendo la definición dada, la primera expresión corresponde al segundo elemento de Ω^1 y el último elemento corresponde al último elemento generado en Ω^1.

4.8 Retos y Perspectivas

PG ha sido utilizado en una gran diversidad de áreas y aplicaciones como se ha descrito en la Sección 4.1. Aun así consideramos de partic-

ular importancia mencionar brevemente dos áreas que han tenido un auge muy importante en los últimos años. En primer lugar trataremos, dentro de aprendizaje computacional, el problema de aprendizaje supervisado y en segundo lugar se tocará el tema de programación automática que es una de las tareas mas anheladas en el área y en particular el tema de la mejora de software mediante PG.

PG ha sido utilizada en diferentes tareas de aprendizaje supervisado mostrando ser un enfoque competitivo contra otras técnicas tradicionales de aprendizaje supervisado, como máquinas de soporte vectorial, árboles de decisión, entre otras. Sin embargo, recientemente ha habido una explosión en el uso de redes neuronales y en particular redes neuronales profundas que atacan este tipo de problemas. Esta tendencia no ha tenido el mismo impulso en la comunidad de PG aun y cuando PG tiene todas las características para competir y contribuir con el área de redes neuronales. Es de esperar que en los siguientes años existan una contribución de técnicas de PG al área de redes profundas y además que existan alternativas basadas en PG al aprendizaje profundo como se puede observar en [388].

Han pasado mas de 25 años desde la publicación del libro de Koza [240], uno podría esperar que en todo este tiempo los avances en programación automática fueran una constante en PG, sin embargo esto no ha sido así. Aunque recientemente existen avances notables en el desarrollo de sistemas de PG capaces de generar programas, está área se ha enfocado a la mejora software ya sea mediante la corrección automática de errores, mejoras en rendimiento, reducción de memoria

e incluso el incluir nuevas características no existentes en el software original.

4.9 Conclusiones

En este capítulo se describió la importancia y potencial que tiene la PG en diversas tareas y aplicaciones. Además, se describieron las partes que conforman un sistema de programación genética tradicional, equivalente a los descritos en [240, 367]). También se describieron los tipos de evolución más comunes, y con base en la evolución de estado estable, se detalló el resto de las partes necesarias para llevar a cabo el proceso evolutivo, como son: la representación de los individuos, la creación de la población inicial, los operadores genéticos y el tipo de selección. Finalmente, se describió el espacio de búsqueda de la PG que se genera partiendo de los conjuntos de funciones y terminales.

Como todo los sistemas, la PG ha evolucionado. Actualmente, existen diferentes propuestas para mejorar sus componentes, empezando desde la creación de la población inicial, p.ej., [465], modificaciones a los operadores genéticos tradicionales, p.ej., [122], operadores semánticos [466, 241, 464], estudios sobre la selección [148] y evolución guiada por una función diferente a la función objetivo [250, 325].

4.10 Para saber más

Para una revisión amplia sobre la PG, se sugiere al lector la siguiente bibliografía. La PG fue descrita inicialmente en el libro de John Koza [240], esta descripción se complementa con el libro escrito por Wolfgang Banzhaf *et. al.* [36] y, más reciente, Riccardo Poli *et. al.* [367] presenta una introducción actualizada al área.

Con respecto a artículos científicos es importante comentar que existe una colección bibliográfica en PG compilada y administrada por William Langdon, John Koza y Steven Gustafson.[23] Esta colección cuenta con más de 13,000 referencias a trabajos de investigación en el tema, incluyendo los trabajos realizados en México.

[2] http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html

[3] http://www.cs.bham.ac.uk/~wbl/biblio/

Capítulo 5

Importancia de la Diversidad en el Diseño de Algoritmos Evolutivos

La convergencia prematura es una de las mayores problemáticas que afectan al rendimiento de las metaheurísticas poblacionales por lo que a la hora de diseñar algoritmos evolutivos es un aspecto a tener en cuenta. En este capítulo se enumeran diferentes técnicas que se han propuesto a lo largo de las últimas décadas para lidiar con este problema. Una de las alternativas más exitosas consiste en examinar los efectos que los diferentes componentes del algoritmo evolutivo tienen sobre la diversidad mantenida en la población y con base en ello rediseñarlos para modificar su comportamiento de forma dinámica. Con el objetivo de ilustrar de una forma detallada este último grupo de técnicas, se discuten dos mecanismos que pertenecen a este grupo. El

primero se aplica en el ámbito de evolución diferencial y consiste en una estrategia de reemplazo que combina una población élite con un mecanismo para mantener la diversidad de forma explícita. El segundo caso está enfocado a los operadores de cruza, donde concretamente se analiza y extiende el Operador de Cruza basado en Simulación Binaria (Simulated Binary Crossover - SBX). En las extensiones se considera el criterio de paro para modificar de forma dinámica el comportamiento del operador con el propósito de inducir un cambio gradual desde exploración hacia intensificación en el proceso de búsqueda. La validación experimental se realizó con algunos de los problemas de prueba más populares del ámbito mono-objetivo y multi-objetivo, alcanzándose mejoras significativas en ambos casos.

5.1 Introducción

Los Algoritmos Evolutivos (Evolutionary Algorithms — EAs) son considerados como uno de los enfoques con mayor eficacia para resolver distintas categorías de problemas de optimización. Se han desarrollado diversas variantes que han sido aplicadas en múltiples campos, como en transporte, economía o ingeniería. Particularmente, se han aplicado tanto en problemas del dominio continuo [183] como del dominio discreto [408]. En general, los EAs han sido especialmente exitosos en la resolución de problemas complejos en los que los enfoques exactos no son actualmente aplicables, como por ejemplo, en problemas NP-completos con espacios de búsqueda grandes [79]. Para el ámbito de este capítulo se hace uso de problemas continuos mono-objetivo

y multi-objetivo con restricciones de caja que se pueden definir tal y como se indica en la Ecuación (5.1).

$$\begin{aligned} Minimizar \quad & \mathbf{F}(\mathbf{X}) \\ Sujeto \quad a \quad & \mathbf{X} \in \Omega \end{aligned} \tag{5.1}$$

donde $\mathbf{X}$ es un vector compuesto por D variables continuas de decisión $\mathbf{X} = [x_1, x_2, ..., x_D]$, cada variable pertenece al conjunto de los reales $x_i \in \Re$, D es la dimensión correspondiente al espacio de las variables de decisión, $\mathbf{F}$ es un vector compuesto por M funciones objetivo $\mathbf{F} = [f_1(\mathbf{X}), f_2(\mathbf{X}), ..., f_M(\mathbf{X})]$, Ω es el espacio factible cuyo límite inferior es $x_i^{(L)}$ y límite superior es $x_i^{(U)}$, es decir $\Omega = \prod_{j=1}^{D}[x_j^{(L)}, x_j^{(U)}]$. Cada vector de variables es mapeado con M-funciones objetivo $\mathbf{F}(\mathbf{X})(\mathbf{F} : \Omega \subseteq \Re^D \rightarrow \Re^M)$ al espacio $\Re^M$ que es conocido como el espacio objetivo.

Actualmente, los EAs son una de las metaheurísticas más conocidas [183], pero a pesar de su éxito y de su uso tan extendido, adaptarlas a nuevos problemas implica la toma de varias decisiones de diseño complejas. Particularmente, a la hora de diseñar de forma apropiada un EA, se ha visto que es muy importante conseguir inducir un balanceo adecuado entre la exploración e intensificación del espacio de búsqueda [209]. Nótese en este punto que, de manera informal, la exploración del espacio de búsqueda consiste en evaluar regiones del espacio de búsqueda que no han sido muestreadas con el fin de detectar regiones promisorias, y la explotación consiste en muestrear en zonas ya evaluadas previamente para realizar una búsqueda más profunda con el fin de encontrar soluciones más refinadas y de mayor calidad.

Cuando en los algoritmos evolutivos todas o casi todas las soluciones están en regiones distantes — alta diversidad — se produce habitualmente una búsqueda exploratoria, es decir, muchas de las nuevas soluciones evaluadas serán distantes a las ya evaluadas anteriormente. Sin embargo, cuando casi todas las soluciones están en una o en unas pocas regiones, se produce una búsqueda intensificadora. Uno de los problemas a la hora de diseñar los algoritmos evolutivos es que en muchos casos no se comprenden todas las implicaciones que los diferentes componentes tienen sobre el mantenimiento de la diversidad de la población y por tanto, sobre el balanceo entre exploración e intensificación [467]. Por ello, analizar el comportamiento y rediseñar en base a lo que está ocurriendo en este aspecto, es parte del proceso de diseño de los EAs.

Relacionado con lo anterior, aparece el concepto de convergencia prematura [467]. Se dice que un algoritmo converge de forma prematura cuando mucho antes de alcanzar el criterio de paro, todas las soluciones están en una zona muy pequeña del espacio de búsqueda. En este sentido, a partir de ese momento es difícil seguir mejorando las soluciones de forma significativa ya que con alta probabilidad sólo se va a realizar un muestreo de soluciones en dicha región. Por ello, es importante detectar si esto ocurre y en tal caso rediseñar algunos aspectos del EA para preservar una mayor diversidad. Sin embargo, si la población es muy diversa durante todo el proceso de búsqueda, se podría no alcanzar un grado adecuado de intensificación y por lo tanto se tendría una convergencia lenta que posiblemente también resultaría en soluciones de baja calidad. Por esta razón, Mahfoud [107] utilizó el

concepto de diversidad útil para referirse a la cantidad de diversidad necesaria para generar soluciones de alta calidad.

En relación al diseño de EAs, se puede observar que en sus inicios la mayoría de enfoques fueron esquemas generacionales [109] en los que las soluciones hijas reemplazaban a la población anterior sin importar su respectiva aptitud o grado de diversidad. En estos esquemas iniciales se usaba la selección de padres para promover que el proceso de muestreo se realizara con mayor probabilidad en las regiones más promisorias encontradas. Por ello, con el propósito de alcanzar un balanceo adecuado entre exploración e intensificación, se desarrollaron muchas estrategias de selección de padres que permitían centrarse con mayor o menor velocidad en las regiones promisorias. Además, se desarrollaron alternativas que modifican otros aspectos como la estrategia de variación [211] y/o el modelo poblacional [9]. En la mayor parte de EAs más recientes, se introduce además una fase de reemplazamiento [132] por lo que la nueva población no tiene que formarse exclusivamente con los hijos, más bien se usan mecanismos para combinar la población anterior con la población hija y determinar así los nuevos sobrevivientes. En este contexto, se suele introducir elitismo en los algoritmos, es decir, el mejor individuo encontrado sobrevivirá a la siguiente generación [140]. De esta forma, ahora también se puede modificar esta última fase para conseguir el balanceo apropiado entre exploración e intensificación.

En los últimos años, algunos de los trabajos más exitosos en el área de evitación de convergencia prematura se han basado en considerar el criterio de parada y evaluaciones realizadas para balancear entre

exploración e intensificación [411]. Esto se fundamenta en la premisa de que el grado entre exploración e intensificación debería variar a lo largo de la ejecución, por lo tanto tiene sentido que las decisiones tomadas por las componentes varíen en función del instante de ejecución. Particularmente, en este capítulo se describen dos aportaciones que pertenecen a este grupo de técnicas. La primera, que se aplica en el área de *Evolución Diferencial* (Differential Evolution — DE), recibe el nombre de DE *Mejorado con Mantenimiento de Diversidad* (DE with Enhanced Diversity Maintenance — DE-EDM) e integra una estrategia de reemplazo que maneja la diversidad de forma explícita con una población élite. En la fase de reemplazo que incorpora el DE-EDM se promueve un balanceo dinámico entre exploración e intensificación, teniendo en cuenta para ello el criterio de paro y las evaluaciones transcurridas para la toma de decisiones. Concretamente, en las primeras etapas se induce un grado de exploración alto ya que los individuos sobrevivientes son diversificados y, posteriormente, conforme transcurren las generaciones se induce un mayor grado de intensificación. La segunda aportación está enfocada a la extensión de operadores de cruza. Particularmente se analizan los componentes que conforman al *Operador de Cruza basado en Simulación Binaria* (Simulated Binary Crossover - SBX), y se propone una variante dinámica (Dynamic Simulated Binary Crossover — DSBX) en la que el comportamiento interno es alterado en base al criterio de paro y a las evaluaciones realizadas hasta el momento.

El resto de este capítulo está organizado de la siguiente forma. En primer lugar, en la sección 5.2 se revisan algunas de las técnicas más

populares que se han propuesto para promover el mantenimiento de diversidad, exponiendo una de las clasificaciones más extendidas. Posteriormente, en la sección 5.3 se describe y valida experimentalmente la aportación basada en evolución diferencial donde se considera la diversidad de forma explícita en la fase de reemplazamiento. Siguiendo esta misma línea en la sección 5.4 se lleva a cabo un análisis del operador de cruza SBX mencionando algunas de sus implicaciones en la diversidad, y se expone en base a ello una variante dinámica que se valida experimentalmente con problemas multi-objetivo. En la sección 5.5 se exponen algunos de los principales retos que encontramos en esta área de investigación. Finalmente, en la sección 5.6 se exponen las conclusiones trazadas a partir de los resultados obtenidos.

5.2 Preservación de diversidad en algoritmos evolutivos

La convergencia prematura es una problemática muy conocida en el ámbito de los EAs por lo que se han desarrollado gran cantidad de técnicas para lidiar con la misma [353]. Estas técnicas modifican de manera directa o indirecta la cantidad de diversidad mantenida por el algoritmo [468] y varían desde técnicas generales hasta mecanismos dependientes de un problema dado. En este apartado se revisan algunas de las técnicas generales más populares. Inicialmente, se describen algunas maneras de clasificar a este tipo de estrategias, para posteriormente describir mecanismos clásicos específicos, así como algunas estrategias más novedosas que se basan en modificar la estrategia de

reemplazamiento. Finalmente, dado que en este capítulo se extiende DE, se revisan también los trabajos de esta área que tienen una relación estrecha con el manejo de diversidad. Se recomienda a los lectores que requieran conocer estos mecanismos con más detalle consultar [468] así como los artículos específicos en que cada método es propuesto.

5.2.1 Clasificaciones de mecanismos para promover la diversidad

Debido a la gran cantidad de métodos desarrollados en esta área, se han propuesto varias clasificaciones de los mismos. Liu et al. [263] propuso diferenciar entre los enfoques uni-proceso y multi-proceso. En el enfoque uni-proceso se modifica la preservación de la diversidad actuando sobre un único componente del EA. Es importante destacar que en los enfoques uni-proceso no se excluye el uso de otros componentes en el proceso de exploración y/o intensificación, sino que más bien, si no se consigue el balanceo adecuado, sólo se modifica una componente hasta conseguir el comportamiento deseado. Por otra parte, en los enfoques multi-proceso se tiene en cuenta las implicaciones que varios componentes provocan sobre el balanceo, y se actúa modificando o rediseñando varios de ellos hasta conseguir el comportamiento adecuado. Los esquemas uni-proceso son mucho más habituales actualmente [467], y en particular las dos propuestas incluidas en este capítulo son mecanismos uni-proceso.

Extendiendo a lo anterior, se propuso una clasificación más específica [467], en la que se tiene en cuenta cuál es la componente que

se cambia para categorizar a cada método. En este sentido, los más populares son los siguientes:

- **Enfoques basados en la selección**: son los más clásicos y se basan en cambiar la presión de selección que se produce hacia las zonas promisorias a la hora de realizar la selección de padres.

- **Enfoques basados en población**: se modifica el modelo poblacional utilizando algunas técnicas como variar el tamaño de la población de forma dinámica, eliminar individuos duplicados, utilizar técnicas de infusión o establecer un estrategia de islas con migraciones.

- **Enfoques basados en la cruza y/o la mutación**: se basan en rediseñar los operadores de cruza y/o mutación, considerando en algunos casos información específica del problema. También se incluyen en este grupo opciones más generales como aplicar restricciones sobre el emparejamiento y/o incluir operadores disruptivos que podrían ser utilizados sólo en ciertos instantes del proceso de optimización.

5.2.2 Esquemas clásicos para administrar la diversidad

Los primeros EAs se basaron principalmente en esquemas generacionales que no incluían fase de reemplazo. En estos esquemas, la selección de padres era la principal responsable de que se muestrearan con mayor probabilidad las zonas más promisorias encontradas hasta el momento,

y por tanto muchos de los primeros esquemas que trataron de evitar la convergencia prematura se basaron en modificar el proceso de selección de padres. Así, en los 90s se desarrollaron varios esquemas que alteraban la presión de selección [132] de forma estática o dinámica. Sin embargo, con base en varios estudios teóricos y experimentales se observó que, generalmente, actuar exclusivamente sobre el operador de selección no es suficiente, especialmente cuando se quieren realizar ejecuciones a largo plazo, ya que se requerirían poblaciones excesivamente grandes para mantener un grado adecuado de diversidad.

Otra alternativa fue modificar los modelos poblacionales, encontrando en este grupo los esquemas basados en islas [9], los celulares o, más recientemente, los basados en *agrupaciones o clústeres* [172]. La idea de introducir restricciones en el emparejamiento, principalmente con base en la ubicación de los individuos en el espacio de búsqueda también ha sido bastante exitosa aunque controlar los mismos para obtener el balanceo apropiado ante diferentes criterios de parada es bastante complejo. En algunos casos resultó ser más prometedor promover el emparejamiento entre individuos no similares [140], mientras que en otros escenarios se hace exactamente lo opuesto [116]. Otro problema común de muchas de las estrategias anteriores es que suelen introducir parámetros adicionales, por lo que el proceso de ajuste de parámetros, que ya de por sí es un problema importante en los EAs, se vuelve aún más complejo. Es importante resaltar que todas estas estrategias clásicas no evitan por completo la convergencia sino que la idea es disponer de mecanismos para acelerarla o retrasarla.

Otra alternativa diferente ha sido adaptar la fase de variación. En este sentido se han desarrollado diversas técnicas para controlar los parámetros que se consideran en la variación con el propósito de adaptar el balanceo entre exploración e intensificación. En algunos casos esto se consigue usando distintos valores en los parámetros para distintas etapas a lo largo del proceso de optimización [501], mientras que en otros casos se hacen cambios más drásticos y se consideran varios operadores con distintas propiedades [265]. También existen mecanismos adaptativos que usan una memoria para almacenar información histórica sobre los efectos de la variación y con base en ello ir modificándola [503]. Cabe destacar que en la mayor parte de estos esquemas no se considera la diversidad de forma directa, sino que sólo se considera para analizar el comportamiento y con base en ello se procede a un rediseño.

Finalmente, un esquema muy sencillo pero no por ello menos importante es el basado en reinicios. En estos esquemas, en lugar de evitar la convergencia acelerada, se aplica un reinicio total o parcial de la población cada cierto número de generaciones o cuando se detecta que la población ha convergido. Con base en esto se han propuesto diversas estrategias para establecer los puntos de reinicio [224]. Estos esquemas se implementan de forma muy sencilla y en algunos casos han proporcionado mejoras significativas [238] por lo que es un método a tener en cuenta, al menos como alternativa inicial. Es común combinar las estrategias basadas en reinicio con algunas de las técnicas anteriores, ya que dichas técnicas están basadas en mantener la

diversidad, mientras que en esta última el objetivo es recuperar la diversidad.

5.2.3 Esquemas de reemplazamiento basados en diversidad

Recientemente se han propuesto diversos mecanismos que modifican la fase de reemplazo para preservar la diversidad. La idea principal de estos esquemas es inducir un grado de exploración adecuado diversificando a los individuos sobrevivientes, de forma que los operadores de reproducción puedan generar nuevas soluciones en diferentes regiones en las siguientes generaciones. Estos métodos están basados en el principio de que los operadores de cruza tienen un efecto de exploración al considerar individuos distantes y de intensificación al considerar individuos próximos [133].

El esquema de pre-selección propuesto por Cavicchio [194] es uno de los primeros estudios que utilizan la fase de reemplazamiento para controlar la diversidad. El esquema inicial de Cavicchio se extendió para generar el esquema denominado *amontonamiento o crowding* [110], el cual ha sido muy popular en los últimos años [280, 294]. El principio del crowding se basa en que los nuevos individuos que entren en la población sustituyan a individuos similares de generaciones anteriores, y con base en este principio, se han formulado diversas implementaciones.

En esta misma línea, se han propuesto otras estrategias de reemplazo con el propósito de promover la diversidad. Uno de los procedimientos más populares es la *Estrategia de Limpieza* (Clearing Strat-

egy - CLR) [274]. En el procedimiento CLR se agrupan a los individuos en grupos denominados nichos, y los mejores individuos de cada nicho son preservados e incluidos en la población de la siguiente generación. Un inconveniente de este procedimiento es que los casos en que se detectan muchos nichos provocan una fuerte inmovilización de la población. Por ello, Petrowski [362] propuso una variante para únicamente seleccionar a individuos cuya aptitud sea mejor que la media de la población.

Otros métodos de este grupo consideran funciones de aptitud que combinan la función objetivo original con la diversidad. Sin embargo, es complejo construir una función compuesta ya que las dos mediciones podrían no ser directamente compatibles, y por lo tanto, las funciones adecuadas suelen depender de cada problema. Una forma de suavizar este inconveniente fue propuesto en el algoritmo de combinación (COMB) [474] donde los individuos son ordenados y categorizados con base en su aptitud y contribución a la diversidad, y la función compuesta se diseña con base en el orden y no con base en los valores de función objetivo y contribución a diversidad. La principal desventaja del COMB es que requiere dos parámetros de usuario, aunque independientemente de esto, se ha usado con bastante éxito. Otra alternativa es el procedimiento *Reemplazamiento Basado en Contribución a la Diversidad y Sustitución del Peor* (Contribution of Diversity/Replace Worst - CD/RW) [274]. En el método CD/RW un nuevo individuo reemplaza a un miembro de la población cuyo rendimiento sea peor tanto en aptitud como en contribución a la diversidad. En caso de no encontrar un peor individuo bajo estos dos

criterios, se procede a reemplazar al peor individuo en la población considerando únicamente a la aptitud. Una última alternativa se basa en considerar a la contribución a la diversidad como un objetivo adicional y aplicar un esquema de optimización multi-objetivo [62] [317]. Estos enfoques son identificados como algoritmos multi-objetivo basados en diversidad. Existen varias estrategias para calcular el objetivo auxiliar [409]. Uno de los enfoques más populares consiste en calcular la contribución a la diversidad de cada individuo con base en la *Distancia al Vecino más Cercano* (Distance to the Closest Neighbor - DCN) [411] de entre los individuos que ya hayan sido seleccionados como supervivientes.

5.2.4 Diversidad en evolución diferencial

Los algoritmos basados en DE son altamente susceptibles a la pérdida de diversidad debido a que se basan en una estrategia de selección muy elitista. Debido a ello, se han desarrollado varios análisis para lidiar con este problema. Dado que en el área se conoce, al menos de manera general, las implicaciones que tiene cada parámetro sobre la diversidad, algunos autores han trabajado en estimar de forma teórica cuáles deben ser los parámetros adecuados para que se produzca cierto tipo de comportamiento [506]. Otros autores han estudiado el efecto que tiene la norma de los vectores de diferencia sobre la mutación [309] y con base en ello se han propuesto mecanismos que prohíben ciertos movimientos que pueden resultar perjudiciales para la diversidad [311]. En este último estudio, el tipo de movimientos aceptados varía a lo largo de la ejecución, descartando a los movimientos cuyos desplaza-

mientos sean menores a un umbral el cual es decrementado conforme transcurren las generaciones. Además, se han propuesto otras formas para establecer los movimientos aceptados [48].

Una alternativa distinta se basa en alterar el operador de selección [399]. Específicamente, con el propósito de mantener mayor diversidad en la población se altera la presión de selección utilizando una selección probabilística que permite escapar en algunos casos de las bases de atracción de óptimos locales. Sin embargo, este método no es demasiado robusto debido a que considera la aptitud para definir las probabilidades para seleccionar a un individuo por lo que ciertas transformaciones de la función pueden modificar de forma drástica el tipo de búsqueda que se realiza.

Finalmente, la variante DE con *Diversidad de la Población Auto-Mejorado* (*Auto-Enhanced Population Diversity* - AEPD) mide la diversidad de forma explícita y cuando se detecta la existencia de un nivel bajo de diversidad en la población, se lanza un mecanismo de diversificación [498]. Esta propuesta se ha extendido para considerar diferentes esquemas de perturbación [514].

Es interesante hacer notar que las variantes de DE que alcanzaron los primeros lugares en varias competencias de optimización durante los últimos años no consideran estas modificaciones, y además estas variantes no han sido incorporadas en las herramientas de optimización más populares. Esto puede deberse a que muchos de los concursos están orientados a obtener resultados en un número de evaluaciones bastante limitado en lugar de a largo plazo, que es el ámbito en el que más beneficios suelen dar los mecanismos de control de diversidad.

5.3 Diseño de evolución diferencial basado en diversidad

5.3.1 Evolución diferencial: Conceptos básicos

Esta sección está dedicada a revisar la variante clásica de DE y a introducir varios términos importantes que son utilizados en el campo de DE. El esquema clásico de DE es identificado como DE/rand/1/bin y ha sido ampliamente usado como base para el desarrollo de variantes más complejas [106]. De hecho, la propuesta que se presenta en este capítulo como ejemplo extiende al DE/rand/1/bin. DE fue propuesto como un método de búsqueda directa para optimización continua mono-objetivo y es el ámbito en que se usará en este capítulo, es decir, se considera la Ecuación (5.1), fijando $M = 1$.

DE es un algoritmo estocástico basado en población, por lo que en cada instante maneja un conjunto de soluciones candidatas que van evolucionando de forma iterativa. En DE dichas soluciones candidatas son usualmente conocidas como vectores. En la variante básica de DE, para cada miembro de la población (conocidos como *vectores objetivo*) se genera un nuevo vector que es conocido como el *vector mutado*. A continuación, el vector mutado se combina con el vector objetivo para generar al *vector de prueba* y finalmente se procede con la fase de selección para elegir a los vectores sobrevivientes. De esta forma, las generaciones transcurren de forma iterativa hasta cumplir el criterio de paro. En esta capítulo, el i-ésimo vector de la población en la generación G se denota como$\mathbf{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, ..., x_{D,i,G}]$. A continuación se explica en más detalle cada componente de DE.

5.3.1.1 Inicialización

Usualmente, DE inicia el proceso de optimización con una población de NP vectores que son creados de forma aleatoria. Habitualmente los vectores de la población inicial son generados con base en una distribución uniforme, ya que usualmente no se posee información sobre cuáles son las zonas más promisorias del espacio de búsqueda. Por lo tanto, el j-ésimo componente del i-ésimo vector es inicializado de la forma $x_{j,i,0} = x_j^{(L)} + rand_{i,j}[0,1](x_j^{(U)} - x_j^{(L)})$, donde $rand_{i,j}[0,1]$ es un número aleatorio uniformemente distribuido entre 0 y 1.

5.3.1.2 Operador de mutación

Por cada vector objetivo se genera un vector mutado para lo cual se han propuesto múltiples estrategias con la particularidad de que en cierta forma se usen las diferencias entre vectores. La variante clásica de DE aplica la estrategia conocida como rand/1, en la cual se crea un vector mutado $V_{i,G}$ de la siguiente forma:

$$\mathbf{V}_{i,G} = \mathbf{X}_{r1,G} + F \times (\mathbf{X}_{r2,G} - \mathbf{X}_{r3,G}) \quad r1 \neq r2 \neq r3 \tag{5.2}$$

En la Ecuación (5.2) los índices $r1, r2, r3 \in [1, NP]$ deben ser enteros distintos y son generados de forma aleatoria en el rango $[1, NP]$. Además, estos índices son distintos al índice i. Es importante hacer notar que la diferencia entre los vectores es escalada por medio del parámetro F, el cual usualmente se define en el intervalo $[0.4, 1]$. Posteriormente, el vector de diferencia (escalado) es agregado a un tercer vector, lo que significa que los vectores mutados son similares a los vectores objetivo si el grado de diversidad es bajo, ya que los vectores

de diferencias tienen norma pequeña. Como consecuencia de esto es claro que es crítico mantener un grado mínimo de diversidad en DE.

5.3.1.3 Operador de cruza

El operador de cruza se aplica con el objetivo de combinar la información de distintas soluciones candidatas y de incrementar la diversidad de los vectores. Específicamente, cada vector objetivo $\mathbf{X}_{i,G}$ se mezcla con su correspondiente vector mutado $\mathbf{V_{i,G}}$ para generar un vector de prueba $\mathbf{U_{i,G}} = [u_{1,i,G}, u_{2,i,G}, ..., u_{D,i,G}]$. La estrategia de cruza más típica es conocida como *cruza binomial* y actúa de la siguiente forma:

$$\mathbf{U}_{j,i,G} = \begin{cases} \mathbf{V}_{j,i,G}, & \text{si} \quad (rand_{i,j}[0,1] \leq CR \quad o \quad j = j_{rand}) \\ \mathbf{X}_{j,i,G}, & \text{de otra forma} \end{cases} \tag{5.3}$$

En la Ecuación (5.3) $rand_{i,j}[0,1]$ es un número uniformemente distribuido, j_{rand} es un índice seleccionado aleatoriamente que asegura que $\mathbf{U}_{i,G}$ genera al menos un componente de $\mathbf{V}_{i,G}$ y $CR \in [0,1]$ es la proporción de cruza.

5.3.1.4 Operador de selección

Finalmente, se aplica una selección elitista para determinar los vectores sobrevivientes que participarán en la siguiente generación. Específicamente, cada vector de prueba se compara con su correspondiente vector objetivo y sobrevive el que tiene la mejor aptitud:

$$\mathbf{X}_{j,i,G+1} = \begin{cases} \mathbf{U}_{i,G}, & \text{si} \quad f(\mathbf{U}_{i,G}) \leq f(\mathbf{X}_{i,G}) \\ \mathbf{X}_{i,G}, & \text{de otra forma} \end{cases} \tag{5.4}$$

Debido a la forma de seleccionar a los sobrevivientes en cada generación los miembros de la población permanecen iguales o mejoran. Por ello, se considera que DE hace uso de una selección muy elitista y es una de las razones por la que en ciertos casos puede sufrir de una convergencia demasiado acelerada.

5.3.2 Propuesta basada en diversidad

El método que se presenta en esta sección está motivado principalmente por dos trabajos. El primero de ellos es un estudio empírico desarrollado por Montgomery et al. [311] que confirma que DE sufre de convergencia prematura en varios problemas. El segundo es un trabajo propuesto por Segura et al. [411] que proporciona mejoras significativas en el campo de optimización combinatoria utilizando para ello una fase de reemplazo conocida como *Reemplazo con Control de Diversidad Dinámica Basada en Varios Objetivos* (Replacement with Multi-objective based Dynamic Diversity Control - RMDDC). Particularmente, el RMDDC controla el grado de diversidad, relacionándolo con el criterio de paro y generaciones transcurridas. Con base en las conclusiones de cada uno de los trabajos, en esta sección se ilustra una propuesta que es una variante novedosa de DE que incluye un mecanismo de reemplazo que sigue los mismos principios que guiaron el diseño de RMDDC. Este nuevo algoritmo recibe el nombre de *Evolución Diferencial con Mantenimiento Mejorado de Diversidad* (Differ-

ential Evolution with Enhanced Diversity Maintenance - DE-EDM) y su código fuente está disponible [1].

DE-EDM (ver Algoritmo 12) es bastante similar a la versión clásica de DE; de hecho, la forma en que se crean los vectores de prueba (líneas 5 y 6) se mantiene intacta. La novedad del DE-EDM es que incorpora una población élite (E) y una estrategia de reemplazo basada en diversidad. Específicamente, con el propósito de seleccionar a los miembros de la población élite se considera el operador de selección elitista de la versión clásica de DE (línea 7). Posteriormente, se aplica la estrategia de reemplazo (línea 8) para determinar a los sobrevivientes. Siguiendo la misma filosofía que en el RMDDC, los individuos que contribuyen muy poco a la diversidad no deberían ser aceptados como miembros de la siguiente generación. Para conseguir este propósito se considera tanto el criterio de paro como las generaciones transcurridas en la selección. Así, se establecerá un grado de diversidad mínimo deseado dependiente del tiempo de ejecución.

La estrategia de reemplazo (ver Algoritmo 13) funciona de la siguiente forma. Inicialmente, recibe a la población padre (vectores objetivo), a la población de hijos (vectores de prueba) y a los vectores élite. De entre todos ellos, debe seleccionar a NP vectores para formar la siguiente población de padres. En primer lugar, con base en el número de evaluaciones de función transcurridas y el criterio de parada, se calcula una distancia mínima deseada (D_t) para mantener en la población (línea 2). A continuación, se juntan las tres poblaciones en un conjunto de miembros candidatos (línea 3) que contiene

[1] El código en C++ puede ser descargado en la siguiente dirección https://github.com/joelchaconcastillo/Diversity_DE_Research.git

Algoritmo 12 Esquema general del DE-EDM

1: Inicializar de forma aleatoria a la población con NP individuos, donde cada uno es distribuido de forma uniforme.
2: $G = 0$
3: **mientras** El criterio de paro no sea alcanzado **hacer**
4: **para** $i = 1$ to NP **hacer**
5: Mutación: Generar al vector mutado ($V_{i,G}$) de acuerdo a la Ecuación (5.2).
6: Cruza: Utilizar la recombinación para generar al vector de prueba ($U_{i,G}$) de acuerdo a la Ecuación (5.3).
7: Selección: Actualizar al vector élite ($E_{i,G}$ en lugar de $X_{i,G}$) de acuerdo a la Ecuación (5.4).
8: **fin para**
9: Reemplazo: Seleccionar a los vectores objetivo (X_{G+1}) de acuerdo al Algoritmo 13.
10: $G = G + 1$
11: **fin mientras**

en cada momento a los vectores candidatos que podrían ser seleccionados para sobrevivir. Posteriormente, se inicializa tanto el conjunto de individuos sobrevivientes como los penalizados con el conjunto vacío (línea 4). Para seleccionar a los NP sobrevivientes se repite el proceso iterativo (líneas 5 - 13) que se describe a continuación. En cada iteración se comienza seleccionando como sobreviviente al mejor individuo del *Conjunto de Candidatos*, es decir al individuo que tiene la mejor aptitud. Este individuo se mueve al *Conjunto de Sobrevivientes* y los individuos del *Conjunto de Candidatos* cuya distancia al individuo seleccionado sea menor que D_t se transfieren al *Conjunto de Penalizados* (línea 9). La forma de calcular la distancia entre dos individuos es con base en la distancia Euclidiana normalizada descrita en

la Ecuación (5.5), donde D es la dimensión del problema, y $x_d^{(L)}, x_d^{(U)}$ son los límites menores y mayores de la dimensión d. En los casos donde el *conjunto de Candidatos* queda vacío antes de seleccionar a NP individuos, el *Conjunto de Sobrevivientes* se llena seleccionando en cada iteración al individuo *Penalizado* con la mayor distancia al individuo más cercano del *Conjunto de Sobrevivientes* (líneas 10 - 13).

$$distancia(x_i, x_j) = \frac{\sqrt{\sum_{d=1}^{D} \left(\frac{x_i^d - x_j^d}{x_d^{(U)} - x_d^{(L)}} \right)^2}}{\sqrt{D}} \tag{5.5}$$

Con el propósito de completar la descripción de DE-EDM se especifica la forma en que se calcula D_t y el procedimiento con el cual se actualizan a los individuos élite. El resto del algoritmo se mantiene igual que la variante clásica de DE. El valor de D_t se utiliza para alterar el grado entre exploración e intensificación, por lo tanto el valor adecuado para este parámetro depende del instante de ejecución. Específicamente, este valor debería ser reducido conforme se alcanza el criterio de paro con el objetivo de promover mayor intensificación en las últimas etapas de optimización. En nuestro esquema se requiere asignar un valor inicial para D_t (D_I), y a continuación de manera similar a como se hace en [411], se realiza una reducción lineal de D_t considerando las evaluaciones transcurridas y el criterio de paro. Particularmente, en este trabajo, el criterio de paro se asigna con base en las evaluaciones a función y la reducción se calcula de tal forma que al 95% del número máximo de evaluaciones el valor de diversidad mínimo requerido es cero, lo que quiere decir que en la última fase la diversidad no es considerada para nada. Entonces,

Algoritmo 13 Fase de Reemplazo

1: Entrada: *Población* (*Vectores Objetivo*), *Hijos* (*Vectores de prueba*), y *Elite*
2: Actualizar $D_t = D_I - D_I * (nfes/(0.95 * max\_nfes))$
3: *Candidatos* = *Población* $\cup$ *Hijos* $\cup$ *Elite*.
4: *Sobrevivientes* = *Penalizados* = $\emptyset$.
5: **mientras** $|Sobrevivientes| < NP$ y $|Candidatos| > 0$ **hacer**
6: *Seleccionados* = Seleccionar al mejor individuo de *Candidatos*.
7: Eliminar *Seleccionado* de *Candidatos*.
8: Copiar *Seleccionado* a *Sobrevivientes*.
9: Encontrar a los individuos de *Candidatos* cuya distancia a *Seleccionados* sea menor que D_t y moverlos a *Penalizados*. En esta parte se considera la distancia normalizada (Ecuación 5.5).
10: **fin mientras**
11: **mientras** $|Sobrevivientes| < NP$ **hacer**
12: *Seleccionado* = Seleccionar al individuo de *Penalizados* con la mayor distancia al individuo más cercano a *Sobrevivientes*.
13: Eliminar *Seleccionado* de *Penalizados*.
14: Copiar *Seleccionado* a *Sobrevivientes*
15: **fin mientras**
16: **devolver** *Sobrevivientes*

denotando max_nfes al número máximo de evaluaciones y $nfes$ al número de evaluaciones trascurridas, D_t es calculado de la siguiente forma $D_t = D_I - D_I * (nfes/(0.95 * max\_nfes))$.

La distancia inicial (D_I) afecta de forma considerable al rendimiento de DE-EDM. Si este parámetro es elevado, el algoritmo maximiza la diversidad de la población en las primeras etapas de optimización resultando en una exploración adecuada que es muy importante en varios tipos de problemas como los multi-modales y deceptivos. Sin embargo, un valor demasiado elevado de D_I podría inducir un grado excesivo de exploración y en consecuencia la intensificación podría fallar. Por otra parte, un valor muy pequeño de D_I provocaría muy poca exploración y por lo tanto sería más difícil evitar óptimos locales. El valor óptimo de D_I podría variar dependiendo del tipo de problema y el criterio de paro. Sin embargo, en esta primera versión no se adapta el valor de D_I para cada problema. En su lugar, se realizó un análisis previo para determinar un valor adecuado, usándose el valor $D_I = 0.3$ en todos los problemas.

Al igual que en la versión estándar de DE, en DE-EDM es necesario asignar una probabilidad de cruza (CR) y un factor de mutación (F). De acuerdo a varios estudios desarrollados por Montgomery y otros [310] la probabilidad de cruza tiene un efecto muy importante. Estos autores mostraron de forma empírica que los valores extremos de CR resultan en comportamientos muy distintos entre sí, pero de gran utilidad. Así, los valores pequeños de CR resultan en una búsqueda alineada a un número reducido de ejes e induce pequeños desplazamientos. Esto provoca una mejora gradual con convergencia lenta

que en algunos escenarios podría resultar beneficioso. Por otra parte, los valores elevados de CR en general, proporcionan soluciones de mayor calidad con una menor probabilidad, ya que se tiende a realizar desplazamientos largos. Sin embargo, cuando son exitosos puedan mejorar la aptitud de forma significativa y permiten explorar zonas distantes. Con base en esto, en nuestra propuesta se emplean los dos mecanismos, es decir, valores elevados y pequeños de CR tal y como se muestra en la Ecuación (5.6).

$$CR = \begin{cases} Normal(0.2, 0.1), & \text{si} \quad rand[0,1] \leq 0.5 \\ Normal(0.9, 0.1), & \text{de otra forma} \end{cases} \tag{5.6}$$

Por otro lado, siguiendo los principios de distintas variantes del SHADE [28, 57], se consideran las evaluaciones transcurridas para generar el factor de mutación F aplicado. Particularmente, cada valor F se muestrea de una distribución Cauchy (Ecuación 5.7).

$$Cauchy(0.5, 0.5 * nfes/max\_nfes) \tag{5.7}$$

El efecto de esta distribución es que en las primeras etapas de optimización se generen valores de F cercanos a 0.5. Posteriormente, conforme la ejecución transcurre, la función de densidad sufre una transformación gradual ya que la varianza se incrementa. Esto implica que se generen valores fuera del intervalo $[0.0, 1.0]$ con una probabilidad más alta. En los casos en que los valores son mayores a 1.0, se utiliza el valor 1.0, mientras que si se genera un valor negativo, se vuelve a muestrear el valor. Uno de los efectos de este enfoque es el de incrementar la probabilidad de generar valores elevados de F conforme

transcurren las generaciones, lo que ayuda a evitar la convergencia acelerada en las últimas etapas de optimización.

5.3.3 Resultados de DE-EDM

En esta sección se presenta la validación experimental de DE-EDM. Específicamente, se muestra que se pueden mejorar los resultados de los algoritmos del estado-del-arte controlando de forma explícita la diversidad. Particularmente, se consideraron los conjuntos de prueba de los concursos de optimización continua organizados en el CEC 2016 y CEC 2017. Cada uno está compuesto de treinta problemas distintos. En la comparativa incluimos a los algoritmos que obtuvieron los primeros lugares en cada año, así como una versión de DE que usa la misma parametrización que DE-EDM pero que no incluye la población élite ni el reemplazamiento basado en diversidad. Los algoritmos considerados del CEC 2016 son el UMOEAs-II [134] y L-SHADE-EpSin [28], que alcanzaron el primero y el segundo lugar respectivamente, mientras que del CEC 2017 se consideran el EBOwithCMAR [244] y el jSO [58]. Todos estos algoritmos fueron probados con los dos conjuntos de prueba como se sugiere en [308]. Debido a que todos los algoritmos son estocásticos se realizaron 51 ejecuciones con distintas semillas y en cada caso, el criterio de paro fue asignado a $25,000,000$ de evaluaciones de la función objetivo. Además se consideraron diez variables en cada función ($D = 10$). La evaluación de los algoritmos se realizó siguiendo los lineamientos de las competencias del CEC, de forma que se asignó un error igual a 0 si la diferencia entre la mejor solución encontrada y la solución óptima era menor que 10^{-8}. Para

cada algoritmo se utilizó la parametrización indicada por sus autores y que se describe a continuación:

- **EBOwithCMAR**: Para la parte EBO, el tamaño máximo de la población de $S_1 = 18D$, el tamaño mínimo de la población de $S_1 = 4$, el tamaño máximo de la población de $S_2 = (146.8)D$, el tamaño mínimo de la población de $S_2 = 10$, el tamaño de la memoria histórica H=6. Para la parte de CMAR el tamaño de la población $S_3 = 4+3log(D)$, $\sigma = 0.3$, CS = 50, la probabilidad de búsqueda local $pl = 0.1$ y $cfe_{ls} = 0.4 * FE_{max}$.

- **UMOEAs-II**: Para la parte de MODE, el tamaño máximo de la población de $S_1 = 18D$, el tamaño mínimo de la población de $S_1 = 4$, el tamaño de la memoria histórica H=6. Para la parte del CMA-ES el tamaño de la población $S_2 = 4+\lfloor 3log(D) \rfloor$, $\mu = \frac{PS}{2}$, $\sigma = 0.3$, CS = 50. Para la búsqueda local, $cfe_{ls} = 0.2 * FE_{max}$.

- **jSO**: El tamaño máximo de la población = $25log(D)\sqrt{D}$, el tamaño de la memoria histórica H= 5, valor de mutación inicial de la memoria $M_F = 0.5$, probabilidad inicial de la memoria $M_{CR} = 0.8$, tamaño mínimo de la población = 4, valor inicial p-best = $0.25 * N$, valor final p-best = 2.

- **L-SHADE-EpSin**: Tamaño máximo de la población = $25log(D)\sqrt{D}$, tamaño de la memoria histórica H= 5, valor de la mutación inicial de la memoria $M_F = 0.5$, probabilidad inicial de la memoria $M_{CR} = 0.5$, frecuencia inicial de la memoria $\mu_F = 0.5$, tamaño mínimo de la población = 4, valor inicial

p-best $= 0.25 * N$, valor final p-best $= 2$, generaciones de la búsqueda local $G_{LS} = 250$.

- **DE-EDM**: $D_I = 0.3$, tamaño de la población $= 250$.
- **Standard-DE**: Tamaño de la población $= 250$ (mismos operadores que en DE-EDM).

Nuestro análisis experimental se realizó teniendo en cuenta la diferencia entre la solución óptima y la mejor solución obtenida y para comparar los resultados estadísticamente, se siguió un procedimiento similar que el propuesto en [128]. Concretamente, en primer lugar se utilizó la prueba Shapiro-Wilk para comprobar si los resultados se ajustaban a una distribución Gaussiana. En los casos en que sí se ajustaban, se utilizó la prueba de Levene para comprobar la homogeneidad de las varianzas, procediendo con la prueba de ANOVA en caso positivo o con la prueba de Welch en caso negativo. Por otro lado, para los casos que no se ajustaban a distribuciones Gaussianas, se utilizó la prueba de Kruskal-Wallis. En todos los casos se fijó el nivel de confianza al 95%. Se considera que un algoritmo X es superior a un algoritmo Y, si el procedimiento anterior reporta diferencias significativas y si la media y mediana del error obtenido por el método X son inferiores a las obtenidas por el método Y.

En las tablas 5.1 y 5.2 se presenta un resumen de los resultados obtenidos para el CEC 2016 y el CEC 2017 respectivamente. La columna etiquetada con "Siempre Resuelto" muestra el número de funciones en que se obtuvo un error de cero en las 51 ejecuciones. La columna etiquetada con "Al menos una vez resuelto" muestra el

número de funciones que se resolvieron en al menos una ejecución. Prácticamente todas las funciones del CEC 2017 (28 de ellas) fueron resueltas al menos una vez por nuestra propuesta. Además, se resolvieron 21 funciones del CEC 2016 al menos una vez. Esto representa una diferencia sustancial con los resultados obtenidos por el resto de algoritmos, pues todos ellos resolvieron muchas menos funciones. Con el objetivo de confirmar la superioridad del DE-EDM, se ejecutaron las pruebas estadísticas por pares. La columna etiquetada con el símbolo $\uparrow$ muestra el número de veces en que cada método fue superior, mientras que la columna etiquetada con $\downarrow$ cuenta el número de casos donde el método fue inferior. Finalmente, la columna etiquetada con $\longleftrightarrow$ muestra el número de comparaciones en las que las diferencias no fueron significativas. Las pruebas estadísticas indican que el DE-EDM alcanzó los mejores resultados en los dos años. De hecho el número de comparaciones en que nuestra propuesta ganó en el CEC 2016 y el CEC 2017 fue de 77 y 88 respectivamente y sólo perdió en 25 y 6, respectivamente, que son valores muy superiores a los del resto de algoritmos. La última columna etiquetada con "Puntaje" muestra la puntuación siguiendo los lineamientos propuestos en las competencias del CEC. Particularmente, este método de evaluación combina dos puntajes como se indica en la Ecuación (5.8), definiendo el puntaje final como $Score = Score_1 + Score_2$.

$$\begin{aligned} Score_1 &= \left(1 - \frac{SE_i - SE_{min}}{SE_i}\right) \times 50, \\ Score_2 &= \left(1 - \frac{SR_i - SR_{min}}{SR_i}\right) \times 50, \end{aligned} \tag{5.8}$$

Específicamente, SE_i es la sumatoria de errores del i-ésimo algoritmo ($SE_i = \sum_{j=1}^{30} error\_\hat{f}_j$), además $error\_\hat{f}_j$ es el error promedio en la j-ésima función. SE_{min} es la mínima sumatoria de errores entre todos los algoritmos. Por su parte, SR_i es la sumatoria del rango de cada función en el i-ésimo algoritmo ($SR_i = \sum_{j=1}^{30} rango_j$). Además, SR_{min} es la sumatoria mínima de los rangos entre todos los algoritmos. Nuestra propuesta alcanzó el mejor puntaje de 100.00 en los dos años, demostrando su superioridad. Adicionalmente, es destacable que la versión estándar de DE alcanzó resultados buenos, de hecho obtuvo el tercer y el segundo lugar en los años 2016 y 2017, respectivamente. Esto muestra que el rendimiento de los algoritmos del estado-del-arte se deteriora al considerar ejecuciones a largo plazo pues esos algoritmos que fueron muy efectivos a corto plazo, no lo son tanto a largo plazo. La superioridad del DE-EDM es clara con base en todos los tipos de comparativas descritas.

Además, con el fin de proporcionar resultados que puedan usar otros autores para realizar comparativas, en las tablas 5.3 y 5.4 se reporta el mejor, peor, mediana, media, desviación estándar y razón de éxito para el CEC 2016 y 2017, respectivamente. En estas tablas se observa que nuestra propuesta resuelve todos los problemas unimodales y que en la mayor parte de los problemas multi-modales se obtienen resultados muy aceptables con respecto a los reportados por otros métodos. Así, nuestra propuesta resolvió y mejoró significativamente varias funciones complejas que no fueron resueltas por ninguno de los algoritmos restantes.

Tabla 5.1: Resumen de los resultados - CEC 2016

Algoritmo	**Siempre Resuelto**	**Resuelto al menos una vez**	**Pruebas Estadísticas**			**Puntaje**
			$\uparrow$	$\downarrow$	$\longleftrightarrow$	
EBOwithCMAR	8	14	35	56	59	50.28
jSO	9	17	47	51	52	55.43
UMOEAs-II	9	14	51	31	68	62.45
L-SHADE-Epsilon	7	13	20	71	59	50.12
DE-EDM	13	21	77	25	48	100.00
Standard-DE	11	19	50	46	54	56.29

Tabla 5.2: Resumen de los resultados - CEC 2017

Algoritmo	**Siempre Resuelto**	**Resuelto al menos una vez**	**Pruebas Estadísticas**			**Puntaje**
			$\uparrow$	$\downarrow$	$\longleftrightarrow$	
EBOwithCMAR	9	18	34	46	70	37.14
jSO	8	15	29	55	66	29.30
UMOEAs-II	11	15	43	40	67	26.89
L-SHADE-Epsilon	8	19	7	81	62	32.78
DE-EDM	21	28	88	6	56	100.00
Standard-DE	12	21	56	29	65	42.91

Tabla 5.3: Resultados del DE-EDM con los problemas del CEC 2016

	Mejor	**Peor**	**Mediana**	**Media**	**Sd**	**Razón de éxito**
f_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_4	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_6	0.00E+00	3.60E-02	4.00E-03	7.39E-03	1.15E-02	3.92E-01
f_7	2.00E-02	1.02E-01	5.90E-02	5.77E-02	4.93E-02	0.00E+00
f_8	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{10}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{11}	0.00E+00	6.00E-02	0.00E+00	5.88E-03	1.90E-02	9.02E-01
f_{12}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{13}	1.00E-02	8.00E-02	5.00E-02	4.67E-02	2.60E-02	0.00E+00
f_{14}	1.00E-02	5.00E-02	3.00E-02	2.82E-02	2.13E-02	0.00E+00
f_{15}	0.00E+00	4.70E-01	2.20E-01	1.99E-01	1.55E-01	1.96E-02
f_{16}	4.00E-02	1.50E-01	8.00E-02	8.47E-02	4.96E-02	0.00E+00
f_{17}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{18}	0.00E+00	2.00E-02	1.00E-02	7.65E-03	6.32E-03	3.14E-01
f_{19}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{20}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{21}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{22}	0.00E+00	3.00E-02	0.00E+00	3.73E-03	2.76E-02	7.65E-01
f_{23}	0.00E+00	1.00E+02	0.00E+00	2.55E+01	5.10E+01	7.45E-01
f_{24}	0.00E+00	6.90E-01	0.00E+00	2.61E-02	1.33E-01	9.61E-01
f_{25}	1.00E+02	1.00E+02	1.00E+02	1.00E+02	0.00E+00	0.00E+00
f_{26}	8.00E-02	1.00E+02	5.29E+01	5.20E+01	3.19E+01	0.00E+00
f_{27}	2.50E-01	9.10E-01	5.40E-01	5.60E-01	2.92E-01	0.00E+00
f_{28}	0.00E+00	3.57E+02	3.43E+02	2.76E+02	1.60E+02	1.96E-01
f_{29}	1.00E+02	1.00E+02	1.00E+02	1.00E+02	0.00E+00	0.00E+00
f_{30}	1.84E+02	1.84E+02	1.84E+02	1.84E+02	3.25E-02	0.00E+00

Tabla 5.4: Resultados del DE-EDM con los problemas del CEC 2017

	Mejor	**Peor**	**Mediana**	**Media**	**Sd**	**Razón de éxito**
f_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_4	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_6	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_8	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{10}	0.00E+00	1.20E-01	0.00E+00	1.65E-02	3.39E-02	7.45E-01
f_{11}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{12}	0.00E+00	2.20E-01	0.00E+00	6.37E-02	1.76E-01	6.67E-01
f_{13}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{14}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{15}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{16}	0.00E+00	2.10E-01	0.00E+00	2.47E-02	7.27E-02	8.82E-01
f_{17}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{18}	0.00E+00	1.00E-02	0.00E+00	1.96E-03	4.47E-03	8.04E-01
f_{19}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{20}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{21}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{22}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{23}	0.00E+00	3.00E+02	0.00E+00	3.49E+01	1.03E+02	8.82E-01
f_{24}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{25}	0.00E+00	1.00E+02	0.00E+00	3.92E+00	2.00E+01	9.61E-01
f_{26}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{27}	0.00E+00	3.87E+02	3.87E+02	2.05E+02	2.68E+02	1.96E-02
f_{28}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.00E+00
f_{29}	1.45E+02	2.26E+02	2.18E+02	1.99E+02	4.21E+01	0.00E+00
f_{30}	3.95E+02	3.95E+02	3.95E+02	3.95E+02	2.10E-01	0.00E+00

5.4 Diseño de operadores de cruza basados en diversidad

La segunda propuesta se aplica en el campo de optimización multi-objetivo y está basada en la extensión del operador de cruza SBX. En esta sección se introducen, en primer lugar, varios conceptos relacionados con los algoritmos evolutivos multi-objetivo, para posteriormente describir algunas de las clasificaciones más populares de operadores de cruza. Finalmente, se realiza un análisis del operador SBX y con base en el mismo se propone una variante dinámica denominada *Operador de Cruza Dinámico basado en Simulación Binaria* (Dynamic Simulated Binary Crossover - DSBX), el cual es validado experimentalmente.

5.4.1 Algoritmos evolutivos multi-objetivo

Los EAs han sido utilizados frecuentemente para lidiar con problemas de optimización multi-objetivo (Multi-objective Optimization Problems - MOPs). Particularmente, un MOP de dominio continuo y que es basado en minimización puede ser definido como se indica en la Ecuación (5.1) de la sección 5.1, fijando la M a un valor mayor que uno. Dadas dos soluciones $\mathbf{x}$, $\mathbf{y} \in \Omega$, $\mathbf{x}$ domina a $\mathbf{y}$, denotado por $\mathbf{x} \prec \mathbf{y}$, si y solo si $\forall m \in 1, 2, ..., M : f_m(x_i) \leq f_m(y_i)$ y $\exists m \in 1, 2, ..., M : f_m(x_i) < f_m(y_i)$. Una solución $\mathbf{x}^* \in \Omega$ es conocida como solución óptima de Pareto si no existe otra solución $\mathbf{x} \in \Omega$ que domine a $\mathbf{x}^*$. El conjunto de Pareto es el conjunto de todas las soluciones óptimas de Pareto y el frente de Pareto está formado por

las imágenes del conjunto de Pareto. El propósito de un *Algoritmo Evolutivo Multi-Objetivo* (Multi-objective Evolutionary Algorithm - MOEA) es, esencialmente, obtener un conjunto de soluciones bien distribuidas y cercanas a las soluciones del frente de Pareto.

En los últimos años, se han diseñado una gran cantidad de MOEAs siguiendo distintos principios. A raíz de esto se han propuesto varias taxonomías [39] y, por ejemplo, en base a sus principios de diseño se pueden clasificar en basados en la dominancia de Pareto, indicadores y/o descomposición [363]. Hay algoritmos muy competitivos de cada uno de los grupos por lo que en esta sección se consideraron MOEAs de todos los grupos para realizar la validación. Particularmente, la validación experimental se desarrolló incluyendo a los algoritmos *Algoritmo Genético basado en Ordenación de los No-Dominados* (Non-Dominated Sorting Genetic Algorithm - NSGA-II) [118], el *Algoritmo Evolutivo Multi-objetivo basado en descomposición* (MOEA based on Decomposition - MOEA/D) [512] y el *Algoritmo Evolutivo Multi-objetivo basado en la Métrica-S* (the S-Metric Selection Evolutionary Multi-objective Optimization Algorithm - SMS-EMOA) [43]. Estos algoritmos son representativos de los basados en dominancia, basados en descomposición y basados en indicadores respectivamente.

5.4.2 Operadores de cruza

Los operadores de cruza son utilizados para generar soluciones hijas utilizando la información de las soluciones padre. Así, estos operadores combinan las características de dos o más soluciones padre con el propósito de generar nuevas soluciones candidatas. En base en que

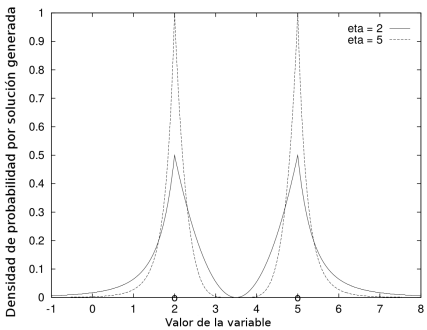


Figura 5.1: Función de densidad del operador de cruza SBX con índices de distribución 2 y 5. Las soluciones padre están ubicadas en 2 y 5 respectivamente.

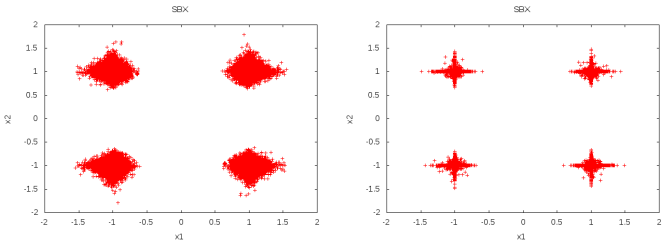


Figura 5.2: Simulaciones del operador SBX con un índice de distribución igual a 20. Las soluciones padre están ubicadas en $P_1 = (-1.0, -1.0)$ y $P_2 = (1.0, 1.0)$. En la parte izquierda la simulación consiste en alterar cada variable con probabilidad 1.0 y en la parte derecha con probabilidad 0.1 (parámetro δ_1 en el Algoritmo 14).

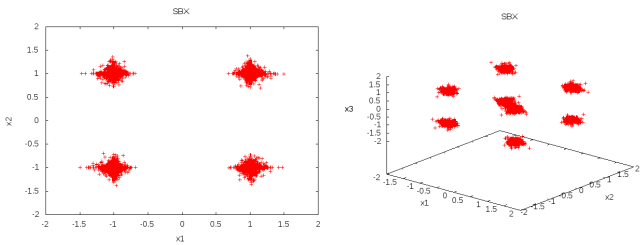


Figura 5.3: Simulaciones del operador SBX con un índice de distribución de 20. Las soluciones padre están ubicadas en $P_1 = (-1.0, -1.0)$, $P_2 = (1.0, 1.0)$ (parte izquierda) y $P_1 = (-1.0, -1.0, -1.0)$, $P_2 = (1.0, 1.0, 1.0)$ (parte derecha).

en la literatura existen diversos operadores de cruza, se han propuesto varias taxonomías para clasificarlos. Particularmente, las taxonomías se basan en varias características tales como la ubicación relativa entre padres e hijos o el tipo de relaciones existentes entre las variables.

Una clasificación popular hace distinción entre operadores de cruza *basados en las variables* y *basados en los vectores*. En los *basados en las variables*, cada variable de las soluciones padre son combinadas para crear nuevos valores de forma independiente, siendo ideales para lidiar con problemas separables. Algunos operadores que pertenecen a esta categoría son el *Operador de Cruza por Mezcla* (the Blend Crossover - BLX) [141] y el SBX [113]. Por otra parte, los operadores de recombinación *basados en vectores* tienen en cuenta la relación que existe entre las variables, por lo que la combinación no es independiente. Así, este tipo de operadores pueden por ejemplo, realizar combinaciones lineales de las soluciones involucradas. Algunos operadores que pertenecen a esta categoría son el *Operador de Cruza Unimodal Normalmente Dis-*

tribuido (Unimodal Normally Distributed Crossover - UNDX) [336], y el *Operador de Cruza Simplex* (Simplex Crossover - SPX) [436].

Adicionalmente, los operadores de cruza pueden ser clasificados como *centrados en los padres* y *centrados en la media* [222]. En los operadores centrados en los padres, las soluciones hijas tienden a ser creadas alrededor de cada solución padre, mientras que en los operadores centrados en la media se tiende a crear a las soluciones hijas alrededor de la media de los valores de las soluciones padres.

5.4.2.1 El operador de cruza basado en simulación binaria - SBX

Entre los operadores de cruza, el *Operador de Cruza Basado en Simulación Binaria* (Simulated Binary Crossover - SBX) [113] es uno de los más utilizados en dominios continuos y fue el que se decidió extender en nuestra propuesta, por lo que esta sección se centra en este operador de cruza. El operador SBX es clasificado como un operador centrado en los padres, por lo que los valores asociados a los hijos (c_1 y c_2) tienden a ser cercanos a los valores de los padres (p_1 y p_2). Específicamente, el proceso para generar los valores de las soluciones hijas se basa en utilizar una distribución de probabilidad. Esta distribución controla el factor de dispersión $\beta = |c_1 - c_2|/|p_1 - p_2|$, el cual es definido como la razón entre la distancia de los valores de las soluciones hijas y la distancia entre los valores de las soluciones padre. Esta función de densidad se define con base en un índice de distribución η_c (es un parámetro de control especificado por el usuario) el cual altera la capacidad de exploración. Específicamente, un índice pequeño induce una proba-

bilidad elevada de crear valores de las soluciones hijas distantes de los valores de las soluciones padre, mientras que índices elevados tienden a crear soluciones muy similares a las soluciones padre, lo cual se ilustra en la figura 5.1. La definición matemática de la distribución y la forma de generar los valores de las soluciones hijas pueden ser estudiados en [113]. Las ecuaciones iniciales fueron formuladas con base en un problema de optimización sin límites en las variables. Sin embargo, en muchos problemas, cada variable está limitada dentro de un límite inferior y superior. Para considerar los límites del espacio de decisión se propuso una modificación de la distribución de probabilidad [115], que es la que se usa en este artículo. Para el caso de nuestro método, no es demasiado importante conocer la fórmula exacta, sino el efecto del valor η_c que se ha ilustrado previamente.

Es importante aclarar que la primera versión del SBX fue diseñada con base en una sola variable. Posteriormente, los autores consideraron aplicarlo a problemas con múltiples variables [113], considerando una estrategia simple para escoger las variables que se van a cruzar [336]. Específicamente, con base en los principios del operador de cruza uniforme, cada variable es cruzada con una probabilidad igual a 0.5. A pesar de su sencillez y de no tener en cuenta las posibles dependencias entre variables, hoy en día ésta es la forma más común de aplicar el SBX.

5.4.2.2 Implementación y análisis del operador SBX

En este apartado se discuten algunas de las principales características de la implementación más utilizada del operador SBX para problemas

Algoritmo 14 Operador de Cruza basado en Simulación Binaria (SBX)

Entrada: Soluciones padre (P_1, P_2), Índice de distribución (η_c), Probabilidad de cruza (P_c).
Salida: Soluciones hijo (C_1, C_2).
si $U[0,1] \leq P_c$ **entonces**
    para para cada variable d **hacer**
        si $U[0,1] \leq \delta_1$ **entonces**
            Generar $C_{1,d}$ utilizando las distribuciones dadas en [115].
            Generar $C_{2,d}$ utilizando las distribuciones dadas en [115].
            si $U[0,1] \leq (1 - \delta_2)$ **entonces**
                Intercambiar $C_{1,d}$ con $C_{2,d}$.
            fin si
        si no
            $C_{1,d} = P_{1,d}$.
            $C_{2,d} = P_{2,d}$.
        fin si
    fin para
si no
    $C_1 = P_1$.
    $C_2 = P_2$.
fin si

con múltiples variables. Esencialmente, se consideran tres componentes clave que podrían afectar al rendimiento de los MOEAs. En primer lugar, como ya se mencionó anteriormente, cada variable es alterada con una probabilidad fija igual a 0.5. Si este valor de probabilidad se incrementa, entonces los hijos tienden a crearse a mayor distancia de los padres debido a que se modifican más variables de forma simultánea. En los problemas separables, en general interesa modificar sólo una variable mientras que en los no separables suele ser más conveniente modificar varias variables a la vez. En la figura 5.2 se pueden observar las implicaciones de modificar esta probabilidad considerando un problema con dos variables. Particularmente, en la parte derecha se muestra que una probabilidad pequeña provoca que algunos valores permanezcan intactos, es decir, hay una tendencia de generar desplazamientos paralelos a los ejes, lo cual suele ser ideal para problemas separables. Por otra parte, en la parte izquierda se muestra que utilizando una probabilidad elevada existe un comportamiento de búsqueda distinto donde la tendencia anterior desaparece, lo cual podría ser oportuno para problemas no separables. Es importante destacar que existe una relación entre esta probabilidad y el índice de distribución, ya que estos dos factores tienen un efecto directo en la similitud que existe entre las soluciones padres e hijas.

El segundo aspecto importante es que después de generar los dos valores de las soluciones hijas, estos valores son intercambiados con una probabilidad fija (usualmente es 0.5), es decir el valor de la solución hija c_1 no siempre es heredado a partir de la solución padre más cercana p_1. Esta es una característica no muy discutida, sin embargo

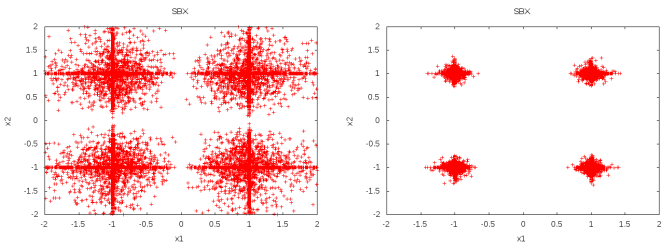


Figura 5.4: Simulación del operador SBX donde las soluciones padre están ubicadas en $P_1 = (-1.0, -1.0)$ y $P_2 = (1.0, 1.0)$. En la parte izquierda se consideró un índice de distribución de 2 y en la derecha de 20.

es un aspecto muy relevante que afecta al rendimiento del algoritmo. En algunos contextos esta probabilidad se denomina "Probabilidad de cruza uniforme por variable" (Variable uniform crossover probability) [459] o "Recombinación Discreta" (Discrete Recombination) [319]. Dado que en el ámbito multi-objetivo se promueve más diversidad en las variables de decisión de forma implícita, estos intercambios podrían ser altamente disruptivos. De hecho, debido a esto, no es totalmente claro que el SBX deba ser categorizado como un operador centrado en los padres. Estos intercambios que existen entre los valores de las soluciones hijas tienen un efecto de realizar múltiples "reflexiones" en el espacio de búsqueda. Así, conforme se incrementa la dimensionalidad en el espacio de las variables, el número de regiones cubiertas crece de forma exponencial como se puede observar en los casos de dos y tres dimensiones en la figura 5.3. Es importante notar que esta característica tiene un efecto relevante en la distancia entre las soluciones padre y las soluciones hijas.

Finalmente, se discute el índice de distribución que es quizás la característica más conocida del operador SBX. Un índice de distribución pequeño provoca un grado de exploración elevado. De hecho, un índice de distribución igual a uno tiene un efecto similar al *Operador de Recombinación Difusa* (Fuzzy Recombination Operator) [478]. En la figura 5.4 se puede observar el efecto de aplicar distintos índices de distribución. Particularmente, en la parte izquierda se considera un índice de distribución pequeño, mientras en la parte derecha se considera un índice de distribución grande. Se observa que este último genera soluciones candidatas similares a las soluciones padre.

En el Algoritmo 14 se muestra la implementación del operador SBX. Este pseudocódigo está basado en la implementación que está integrada en el código del NSGA-II propuesto por Deb et al. [118], la cual se considera como la variante más popular. En los parámetros de entrada se requieren dos soluciones padre (P_1, P_2), y éste crea dos soluciones hijas (C_1, C_2). El primero y el segundo componente mencionados previamente corresponden a las líneas 5 y 9 respectivamente. El caso clásico del operador SBX se configura con los parámetros $\delta_1 = \delta_2 = 0.5$ y $\eta_c = 20$. Es importante hacer notar que la implementación clásica no considera la dimensionalidad de las variables o el criterio de paro como parte de sus parámetros internos.

5.4.3 Propuesta - DSBX

Con base en el análisis anterior y con el propósito de inducir un balanceo entre exploración e intensificación de forma dinámica y que dependa del instante de ejecución, se proponen las siguiente modifica-

ciones. En primer lugar, se modifica la probabilidad de alterar una variable (δ_1) durante la ejecución. La intención de esta modificación es estimular la capacidad de exploración considerando inicialmente una probabilidad elevada para alterar más variables de forma simultánea, mientras que conforme la ejecución avanza se reduce esta probabilidad, con el fin de reducir el número de variables que se modifican y promover así movimientos más pequeños. El valor de δ_1 se cambia con base en un modelo lineal decreciente, donde inicialmente se fija a 1.0 y se decrementa de forma que a la mitad del total de generaciones alcanza el valor 0.5. Este último valor es mantenido hasta el final de la ejecución, es decir, desde la mitad de la ejecución este parámetro se fija al mismo valor que el de la implementación tradicional del SBX. Para asignar el valor δ_1 se utiliza la Ecuación (5.9), donde $G_{Transcurridas}$ corresponde a la generación actual y G_{Total} corresponde al número total de generaciones.

El segundo cambio está relacionado con la probabilidad de aplicar reflexiones ($1 - \delta_2$). En este caso, δ_2 es actualizado de acuerdo a la Ecuación (5.9), por lo tanto la probabilidad de aplicar una reflexión se incrementa de 0.0 a 0.5 durante la ejecución. Esta modificación se realiza con el propósito de evitar el comportamiento disruptivo de intercambiar variables en las primeras generaciones ya que esto provocaría modificaciones muy drásticas. De esta forma, sería más sensato aplicar estas reflexiones una vez que los individuos convergen en cierto grado. Nótese que se incrementa hasta el valor 0.5 el cual es el valor utilizado en la implementación del SBX estándar, ya que no se quiere disponer de un comportamiento excesivamente disruptivo. Sin em-

Tabla 5.5: Puntos de referencias para el indicador HV

Instancias	Punto de referencia
WFG1-WFG9	$[2.1, ..., 2m + 0.1]$
DTLZ 1, 2, 4	$[1.1, ..., 1.1]$
DTLZ 3, 5, 6	$[3, ..., 3]$
DTLZ7	$[1.1, ..., 1.1, 2m]$
UF 1-10	$[2, ..., 2]$

bargo, en el futuro sería interesante realizar estudios con otros valores finales.

$$\delta_1 = \delta_2 = max\left(0.5, 1.0 - \frac{G_{Transcurridas}}{G_{Total}}\right) \tag{5.9}$$

Finalmente, el índice de distribución también es modificado durante la ejecución. En las primeras etapas se promueve un índice de distribución pequeño con el propósito de incrementar la capacidad de exploración del SBX. Posteriormente se decrementa de forma lineal, lo cual tiene el efecto de que la curva de distribución se cierre, y por lo tanto se promueve un mayor grado de intensificación en las etapas finales. El incremento lineal es llevado a cabo usando la Ecuación (5.10), por lo tanto el índice de distribución es alterado de 2 a 22. Es importante aclarar que ya se han considerado modificaciones similares al índice de distribución [525], [198], aunque las otras propiedades que se modifican nunca han sido estudiadas en profundidad.

$$\eta_c = 2 + 20 \times \left(\frac{G_{Transcurridas}}{G_{Total}}\right) \tag{5.10}$$

5.4.4 Resultados

En este apartado se analizan los resultados obtenidos con las variantes dinámicas del SBX (DSBX). El nuevo operador de cruza se integró en

Tabla 5.6: Información estadística de las métricas considerando dos objetivos

	NSGA-II						MOEA/D						SMS-EMOA					
	1	2	3	4	5	DE	1	2	3	4	5	DE	1	2	3	4	5	DE
HV promedio	0.88	0.90	0.90	0.91	0.93	**0.94**	0.87	0.87	0.87	0.90	**0.91**	**0.91**	0.88	0.89	0.87	0.91	0.92	**0.93**
IGD+ promedio	0.12	0.09	0.11	0.07	0.06	**0.05**	0.14	0.12	0.14	0.09	0.08	**0.07**	0.13	0.11	0.14	0.08	0.07	**0.05**

Tabla 5.7: Información estadística de las métricas considerando tres objetivos

	NSGA-II						MOEA/D						SMS-EMOA					
	1	2	3	4	5	DE	1	2	3	4	5	DE	1	2	3	4	5	DE
HV promedio	**0.87**	0.84	**0.87**	**0.87**	**0.87**	0.85	0.84	0.84	0.84	**0.86**	**0.86**	0.85	0.90	0.89	0.88	**0.91**	**0.91**	**0.91**
IGD+ promedio	0.13	0.16	0.13	**0.12**	**0.12**	0.13	0.15	0.14	0.15	**0.11**	**0.11**	0.13	0.11	0.11	0.13	**0.09**	**0.09**	0.13

los algoritmos NSGA-II, MOEA/D y SMS-EMOA. En primer lugar, se analiza cada una de las tres modificaciones propuestas de forma aislada. Posteriormente se construye un caso donde se consideran las dos modificaciones que ofrecieron mejores resultados de forma simultánea. Como parte de la validación experimental se consideran los problemas de prueba WFG [215], DTLZ [119] y UF [513]. Además, con el propósito de comparar nuestra extensión del SBX con otros operadores se incluye la variante de evolución diferencial conocida como DEMO [459].

Para llevar a cabo el análisis experimental, y dado que todos los algoritmos son estocásticos, cada caso fue ejecutado 35 veces con distintas semillas. La configuración global que se aplicó a todos los algoritmos fue la siguiente. Se asignó el criterio de paro a $25,000$ generaciones, el tamaño de la población a 100, se configuraron los problemas de prueba WFG con dos y tres objetivos considerando 24 variables, donde 20 variables son parámetros de distancia y 4 son parámetros de posición. Como es sugerido en [119] se consideró $n = M + r - 1$ variables de decisión en los problemas de prueba DTLZ, donde para los problemas DTLZ1, DTLZ2 - DTLZ6 y DTLZ7 se consideraron

Tabla 5.8: Resumen de las pruebas estadísticas

NSGA-II															
	1			2			3			4			5		
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$
HV-2obj	16	29	47	6	61	25	28	19	45	31	23	38	**54**	3	35
HV-3obj	15	19	42	12	50	14	17	15	44	**33**	10	33	26	9	41
IGD-2obj	14	30	48	4	60	28	25	17	50	33	19	40	**52**	2	38
IGD-3obj	14	18	44	13	44	19	18	15	43	**33**	15	28	23	9	44

MOEA/D															
	1			2			3			4			5		
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$
HV-2obj	15	33	44	10	60	22	25	26	41	39	18	35	**57**	9	26
HV-3obj	10	22	44	12	39	25	11	19	46	24	10	42	**38**	5	33
IGD-2obj	16	31	45	9	60	23	23	27	42	37	17	38	**57**	7	28
IGD-3obj	12	22	42	13	43	20	13	24	39	30	9	37	**40**	10	26

SMS-EMOA															
	1			2			3			4			5		
	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$	$\uparrow$	$\downarrow$	$\longleftrightarrow$
HV-2obj	9	35	48	7	43	42	16	31	45	41	9	42	**53**	8	31
HV-3obj	7	21	48	9	35	32	13	21	42	27	6	43	**31**	4	41
IGD-2obj	10	34	48	15	48	29	12	33	47	41	12	39	**55**	6	31
IGD-3obj	8	20	48	13	30	33	9	19	48	22	5	49	**27**	5	44

$r = \{5, 10, 20\}$ respectivamente. En el caso de los problemas de prueba UF se utilizaron 30 variables de decisión. Finalmente, se hizo uso del operador de mutación polinomial con una probabilidad de mutación de $1/n$ y con un índice de distribución igual a 50, mientras que el operador de cruza SBX se utilizó con una probabilidad de cruza igual a 0.9 y un índice de distribución de 20.

A continuación se especifica la parametrización adicional propia de cada algoritmo:

- **DEMO**: CR = 0.3 y F = 0.5.
- **SMS-EMOA**: desplazamiento para calcular el HV = 100.
- **MOEA/D**: tamaño de la vecindad = 10, el número de actualizaciones por subproblema (nr) = 2 y $\delta = 0.9$.

Para comparar los frentes obtenidos por cada método se utiliza el hipervolumen normalizado (HV), que se debe maximizar, y la *Distancia Generacional Invertida Modificada* (Inverted Generational Distance Plus - IGD+), que se debe minimizar. En la tabla 5.5 se presentan los puntos de referencia utilizados para el indicador del hipervolumen los cuales son similares a los utilizados en [42, 517]. Por otra parte, para comparar los resultados estadísticamente (valores del IGD+ y HV), se siguió un procedimiento similar al que se propuso en [128], siendo el mismo que se aplicó para validar la primera propuesta.

5.4.5 Análisis de cada modificación en el operador SBX

En este apartado se analiza el efecto que cada modificación propuesta tiene de forma independiente sobre los resultados obtenidos. Para ello basados en el Algoritmo 14 se proponen cuatro casos:

- **Caso 1**: se aplica la versión estándar del operador SBX donde $\delta_1 = \delta_2 = 0.5$ y $\eta_c = 20$.
- **Caso 2**: se actualiza el valor δ_1 como se indica en la Ecuación (5.9), $\delta_2 = 0.5$ y $\eta_c = 20$.
- **Caso 3**: se actualiza el valor δ_2 como se indica en la Ecuación (5.9), $\delta_1 = 0.5$ y $\eta_c = 20$.
- **Caso 4**: se actualiza el índice de distribución de acuerdo a la Ecuación (5.10), $\delta_1 = \delta_2 = 0.5$.

En las tablas 5.6 y 5.7 se muestra información del HV normalizado [525] y del IGD+ [219] para cada caso, con dos y tres objetivos respectivamente (el Caso 5 se comenta más adelante). Específicamente, se muestra la media del HV e IGD+ para todos los problemas en dos y tres objetivos. Se observa que considerando dos y tres objetivos el Caso 4 mejora al Caso 1, al Caso 2 y al Caso 3 en todos los algoritmos. Por lo tanto, se aprecian de forma clara los beneficios de incrementar el índice de distribución durante la ejecución. Por otra parte, al considerar tres objetivos, el Caso 2 presentó un rendimiento menor que el Caso 1, posiblemente debido a que al alterar tantas variables de forma

simultánea, se produce un comportamiento excesivamente disruptivo. Quizás los resultados podrían mejorar si el parámetro δ_1 es alterado de una forma distinta, sin embargo esto se deja como trabajo futuro. Los análisis anteriores únicamente consideran la media obtenida en todos los problemas de prueba. En la tabla 5.8 se muestran los resultados de las pruebas estadísticas (el Caso 5 se discute en la siguiente sección). Particularmente, se realizan comparaciones por pares en base a las pruebas estadísticas ya mencionadas entre los cinco casos. Este procedimiento se realizó de forma independiente para el NSGA-II, MOEA/D y el SMS-EMOA. Para cada algoritmo y para cada caso, la columna "$\uparrow$" reporta el número de comparaciones donde las pruebas estadísticas confirmaron la superioridad del caso correspondiente, mientras que en la columna "$\downarrow$" se reporta el número de veces donde este caso fue inferior y la columna "$\longleftrightarrow$" indica el número de comparaciones donde las diferencias no fueron significativas. Con base en estos resultados se puede observar que el caso 3 y caso 4 aportan con respecto al caso 1, mientras que el caso 2 no es robusto, siendo en varios algoritmos inferior al caso 1. Adicionalmente, se anexan resultados detallados para facilitar que otros investigadores puedan realizar comparativas con un mayor nivel de detalle[2].

5.4.6 Modificación simultánea de varios componentes

Con base en los análisis realizados con anterioridad se propone una variante del operador SBX que incorpora las modificaciones propuestas en el Caso 3 y Caso 4 de forma simultánea, es decir, se incorpora

[2]https://github.com/joelchaconcastillo/SBX_CEC2018.

un cambio dinámico tanto en el parámetro δ_2 como en el índice de distribución. Debido a los resultados de baja calidad del Caso 2 el parámetro δ_1 no se modifica de forma dinámica. Particularmente, el Caso 5 se construye con base en el Algoritmo 14 asignando el parámetro δ_1 a 0.5 debido a que es la forma de la versión estándar del SBX, mientras que δ_2 es actualizado de acuerdo a la Ecuación (5.9) y el parámetro η_c es actualizado como se indica en la Ecuación (5.10).

De acuerdo a la media del HV y del IGD+ que se obtuvieron en el Caso 5 (ver tablas 5.6 y 5.7), es claro que integrar los Casos 3 y 4 proporciona beneficios importantes. En el caso de dos objetivos se observa una ventaja significativa. Sin embargo, en el caso de tres objetivos los Casos 4 y 5 son muy similares con base en la media. Además, al considerar tres objetivos los resultados que se obtuvieron en el Caso 5 son superiores a los obtenidos con DE, mientras que al considerar la versión estándar de SBX se observa un deterioro. Por lo tanto, si el operador SBX es configurado adecuadamente puede generar resultados similares o superiores que el algoritmo DEMO.

Finalmente, en la tabla 5.8, se muestran los resultados de las pruebas estadísticas. Los beneficios obtenidos por el Caso 5 son muy claros. De hecho, únicamente el Caso 4 obtuvo mejores resultados que el Caso 5 al considerar tres objetivos y al algoritmo NSGA-II. Esto demuestra que se pueden obtener mejores resultados si se integran varias modificaciones dinámicas de forma simultánea. Además, los resultados confirman las ventajas de la versión dinámica frente al caso estándar (Caso 1). El único caso que no presenta ventajas importantes frente

a la versión estándar del SBX es el Caso 2, el cual fue discutido con anterioridad.

5.5 Retos y perspectivas

El campo de diseño de EAs es un campo muy activo y en el que hay muchas ideas aún por explorar. Los retos en este campo, pasan por hacer el trabajo de diseño un proceso mucho más metódico y algorítmico. Así, parece importante que en este campo no sólo se publiquen los métodos finales que han dado buenos resultados en un problema dado, sino también cómo se realizó el desarrollo completo. Actualmente, el desarrollo de buenos optimizadores es un proceso iterativo en el que en cada fase se van incluyendo nuevas componentes que producen mejoras. Crear guías sobre cómo realizar todo este proceso es un gran reto actual.

En lo que concierne al caso más específico de las estrategias de control de diversidad, el análisis de los resultados reportados en la literatura muestra que son estrategias que ofrecen grandes ventajas a largo plazo, es decir, cuando la cantidad de recursos computacionales que se pueden utilizar son bastante grandes. Dado que cada vez se cuenta con mayores clústeres de computación, es importante abordar el tema de cómo desarrollar optimizadores paralelos con estrategias de control de diversidad. En este línea existen algunos trabajos recientes, pero realmente los avances aún son mínimos. Otros aspecto importantes a estudiar está relacionado con qué se puede hacer para reducir el esfuerzo computacional requerido para conseguir resultados prom-

etedores. La mayor parte de técnicas actuales son métodos dinámicos no adaptativos, y dadas las ventajas que han ofrecido los métodos adaptativos en otras áreas, desarrollar métodos adaptativos de control de diversidad parece realmente prometedor. Finalmente, un aspecto negativo de las estrategias vistas enjoel: este capítulo y de la mayor parte de métodos de gestión de diversidad, es que se introducen nuevos parámetros, por lo que es importante combinar estos mecanismos con estrategias de control de parámetros para facilitar su utilización. Esto es importantísimo ya que un reto global del área de metaheurísticas es conseguir que los usuarios no expertos puedan hacer uso, por sí mismos, de las estrategias desarrolladas por los investigadores.

5.6 Conclusiones

La convergencia prematura es una de las debilidades más importantes de los EAs. Una de las formas de lidiar con este problema consiste en inducir un balanceo adecuado entre exploración e intensificación. Sin embargo, obtener este balanceo no es una tarea trivial debido a que cada problema de optimización tiene características distintas y por ello han surgido numerosos esquemas para tratar esta problemática. Con base en esto se han desarrollado varias taxonomías con el propósito de clasificar las diferentes formas en que se puede preservar y promover la diversidad. En este trabajo, con el fin de ilustrar como se pueden realizar diseños de EAs que tomen en cuenta el balanceo entre exploración e intensificación se presentan dos ejemplos. De forma general, las propuestas establecen un comportamiento dinámico de algún com-

ponente del EA teniendo en cuenta para ello el criterio de paro y el instante en que se encuentra la ejecución. Así, se obtuvo un balanceo adecuado en el que en las primeras fases se promueve más exploración y en las últimas se promueve más la intensificación. De modo más específico, en la primera propuesta se modifica evolución diferencial por medio de una nueva fase de reemplazo con el propósito de conseguir este balanceo. Además, esta contribución incorpora una población élite con el propósito de proporcionar soluciones de calidad tanto a mediano como a largo plazo. Con base en la validación experimental llevada a cabo, se observa que esta propuesta supera a los mejores algoritmos que se habían propuesto en una competición que consideró las funciones de prueba usadas en este capítulo. Posteriormente, se presenta un análisis del operador de cruza SBX y se desarrolló una variante que modifica varios componentes de forma dinámica considerando también el criterio de paro y las generaciones transcurridas. Con base en los resultados obtenidos con problemas muy populares del ámbito de optimización multi-objetivo, se muestra que usar el operador SBX modificado ofrece ventajas muy importantes en varios MOEAs y para diferentes indicadores. De forma general se observa que obtener un balanceo apropiado entre exploración e intensificación es realmente importante, y que esto se puede conseguir utilizando técnicas radicalmente diferentes aunque siguiendo los mismos principios de diseño.

Para saber más

- Para saber más sobre la historia de los algoritmos evolutivos se sugiere consultar [156, 93].
- Puede consultar información sobre tipos de metaheurísticas y diseño e implementación de las mismas en [440].
- Si desea consultar información básica sobre los fundamentos de los algoritmos evolutivos se sugiere consultar [132, 265, 32].
- Para profundizar más en el tópico de manejo de diversidad, exploración e intensificación, se sugiere consultar [467].
- Se puede consultar material relacionado con operadores de cruza y algoritmos evolutivos multi-objetivo en [90, 112, 90].
- Para obtener más información relacionada al tema de evolución diferencial se sugiere consultar [373, 79].
- Si desea consultar aplicaciones de algoritmos evolutivos multi-objetivo se puede consultar [525, 30].

Capítulo 6

Optimización Evolutiva Multi-objetivo

La mayor parte de los problemas de optimización del mundo real involucran dos o más objetivos en conflicto que se desean optimizar en forma simultánea. La solución a esta clase de problemas está conformada por un conjunto de soluciones entre las cuales no se puede establecer una sola como la mejor. El paradigma de cómputo evolutivo se presenta como una opción prometedora para resolver este tipo de problemas por el hecho de manejar en forma simultánea un multi conjunto de soluciones. Este capítulo presenta una breve introducción a los conceptos básicos de optimización evolutiva multi-objetivo, sin pretender cubrir en forma exhaustiva los desarrollos llevados a cabo en el área en los últimos años. Busca más bien plasmar una visión que los autores tienen del área y hacer notar algunos aspectos que consideramos relevantes. Se presenta también un listado de problemas

todavía no resueltos que consideramos importante explorar. Así también se hace una breve mención a la contribución de grupos mexicanos en esta área del conocimiento.

6.1 Introducción

Un gran número de problemas del mundo real involucran la optimización simultánea de varios objetivos que se encuentran en conflicto unos con otros. Estos problemas vienen de distintas disciplinas como la economía, ingeniería, transporte, biología, farmacología, medio ambiente, por citar algunas. Por ejemplo al diseñar un fármaco [246] se desea maximizar su potencia, minimizar su costo de producción y toxicidad así como los efectos secundarios de éste. Aumentar la potencia del fármaco estará normalmente en conflicto con disminuir la toxicidad del mismo. Por otro lado, en las actividades cotidianas se realiza el transporte de mercaderías o materiales peligrosos de un punto a otro en una región, en un modelo similar al que ocurre en internet cuando se desea movilizar grandes cantidades de datos de un nodo a otro. En esta clase de problemas no sólo se busca la ruta más corta, sino también la más segura (en el caso de datos) y la menos riesgosa en el caso de traslado de materiales peligrosos. En general, la ruta más segura no será la más corta y viceversa. Para esta clase de problemas no existe un único valor óptimo de las funciones objetivo sino un conjunto de soluciones con sus respectivas funciones objetivo que no son mejores entre sí bajo todos los criterios al mismo tiempo.

Para abordar esta clase de problemas se han establecido condiciones y propuesto métodos para resolver problemas que satisfacen dichas condiciones. Por ejemplo, cuando las funciones objetivo y las restricciones son lineales en las variables de decisión, éstas pueden ser abordados con técnicas de programación lineal multi-objetivo (MO) [508, 434, 275], donde, entre otros métodos, se recurren a extensiones del método Simplex para programación lineal de un solo objetivo. En el caso de que se tengan funciones objetivo y una región factible convexas, existen algoritmos eficientes para su solución [51].

Desafortunadamente, la mayoría de los problemas del mundo real no son convexos y deben aplicarse métodos específicos caso por caso [300, 356]. Por otro lado, el cómputo evolutivo ofrece una alternativa al diseño de métodos de caso por caso y propone un marco de trabajo general para la solución de problemas de este tipo. La primera propuesta con un enfoque evolutivo surge en los 1980s con la tesis doctoral de Schaffer [403]. A partir de ahí el interés en el área se incrementa cada vez de manera más acelerada. Prueba de ello son las conferencias bianuales especializadas [520, 162, 453], las monografías [112, 90, 441] sobre estas técnicas y sus aplicaciones y el volumen de literatura sobre el tema actualmente disponible.[1]

El capítulo está organizado de la siguiente manera. Los conceptos básicos de optimización multi-objetivo son presentados en la Sección 6.2. La Sección 6.3 presenta una breve introducción a los algoritmos evolutivos. Los algoritmos evolutivos multi-objetivo son presentados en la Sección 6.4. La Sección 6.5 presenta las medidas de calidad

[1] http://emoo.cs.cinvestav.mx/

más comúnmente usadas para comparar el desempeño de nuevas propuestas. La Sección 6.6 presenta una revisión parcial de las contribuciones realizadas al área por los grupos de investigación en México. Los retos y perspectivas de investigación en el área se presentan en la Sección 6.7. Finalmente, la Sección 6.8 sugiere literatura adicional relevante y se menciona de manera sucinta las plataformas de programación disponibles tanto para implementar algoritmos evolutivos multi-objetivo como para el cálculo de criterios de calidad.

6.2 Conceptos Básicos

Sin pérdida de generalidad, un problema de optimización multi-objetivo se define de la siguiente manera:

$$Minimizar \quad \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T$$

sujeto a:

Restricciones de desigualdad:

$$g_i(\mathbf{x}) \geq 0 \qquad i = 1, 2, \ldots, k \tag{6.1}$$

de igualdad:

$$h_i(\mathbf{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{6.2}$$

Donde $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ es un vector cuyos componentes son las variables de decisión o variables de diseño. Las expresiones (6.1) y (6.2) se denominan restricciones y son parte inherente del problema, pueden ser restricciones físicas, de tiempo, de presupuesto, de energía, o de cualquier otro tipo. Al conjunto de las $\mathbf{x}$ que satisfacen estas restricciones se las denomina soluciones factibles y ellas definen la región factible del problema. Las m funciones f_i son los objetivos a minimizar de manera simultánea.

Para entender los conceptos de optimalidad en el contexto de varios objetivos es necesario primero entender cómo comparar dos soluciones bajo este paradigma.

Para dos vectores solución cualesquiera $\mathbf{a}$ y $\mathbf{b}$ existen cuatro posibles relaciones entre ellos (para ejemplificar y sin pérdida de generalidad supondremos minimización para cada una de las funciones objetivo $f_i(\cdot)$):

1. La solución $\mathbf{a}$ *domina* a la solución $\mathbf{b}$ ($\mathbf{a} \prec \mathbf{b}$), si cada componente de $\mathbf{f}(\mathbf{a})$ es igual o mejor al componente correspondiente de $\mathbf{f}(\mathbf{b})$, $f_i(\mathbf{a}) \leq f_i(\mathbf{b}) \forall i = 1, 2, \ldots, m$; y $f_i(\mathbf{a}) \neq f_i(\mathbf{b})$ para al menos un i.

2. La solución $\mathbf{a}$ *es dominada* por la solución $\mathbf{b}$ ($\mathbf{a} \succ \mathbf{b}$). Para cada componente de $\mathbf{f}(\mathbf{a})$ los componentes de $\mathbf{f}(\mathbf{b})$ son iguales o mejores, $f_i(\mathbf{a}) \geq f_i(\mathbf{b}) \forall i = 1, 2, \ldots, m$; y $f_i(\mathbf{a}) \neq f_i(\mathbf{b})$ para al menos un i.

3. La solución $\mathbf{a}$ *es igual* a la solución $\mathbf{b}$ ($\mathbf{a} = \mathbf{b}$). En cada uno de los componentes de $\mathbf{f}(\mathbf{a})$, los componentes correspondientes de $f(\mathbf{b})$ son los mismos, $f_i(\mathbf{a}) = f_i(\mathbf{b}) \forall i = 1, 2, \ldots, m$.

4. La solución $\mathbf{a}$ *es no comparable* con la solución $\mathbf{b}$ ($\mathbf{a} \succ\prec \mathbf{b}$). Este caso ocurre cuando ninguno de los tres primeros casos sucede. En este caso se dice que ambas soluciones son igual de buenas (o igual de malas).

La solución a un problema de optimización multi-objetivo es entonces el conjunto de soluciones no dominadas del conjunto de todas las soluciones factibles del problema. Este se denomina también conjunto de soluciones *no-inferiores* [434] o decisiones Pareto óptimas [356].

Definición 1 *Para un problema de optimización multi-objetivo, el conjunto óptimo de Pareto (*$\mathbf{P}$*) se define como*

$$\mathbf{P} := \{\mathbf{x} \in \mathbf{D} | \nexists \mathbf{y} \in \mathbf{D} \quad \mathbf{f}(\mathbf{y}) \prec \mathbf{f}(\mathbf{x})\}$$

$\mathbf{D}$ *es el conjunto de soluciones que cumplen con (6.1) y (6.2), v.g., el conjunto de soluciones factibles.*

Definición 2 *Para un problema de optimización multi-objetivo y un conjunto óptimo de Pareto* $\mathbf{P}$*, el frente Pareto se define como:*

$$\mathbf{PF} := \left\{ \mathbf{u} = \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T | \mathbf{x} \in \mathbf{P} \right\}$$

Donde en la formulación del problema multi-objetivo anterior minimizar $\mathbf{f}(\mathbf{x})$ sujeto a (6.1) y (6.2) significa encontrar el conjunto óptimo de Pareto.

Es importante notar que las variables de decisión y las funciones objetivo definen dos espacios a considerar:

1. El espacio de las variables de decisión $\mathbf{x}$ (decision space), el cual es de dimensión n. Cada punto en este espacio corresponde a un vector $\mathbf{x}$.

2. El espacio de criterios $\mathbf{f}(\mathbf{x})$ (criteria space), el cual es de dimensión m. Cada punto es la correspondiente imagen, a través de $\mathbf{f}(\cdot)$, de una solución $\mathbf{x}$ al problema.

Por lo tanto, la diferencia entre el conjunto óptimo de Pareto y el frente Pareto es que el primero se encuentra en el espacio de decisión y el frente Pareto se encuentra en el espacio de criterios. Esto hace a los problemas multi-objetivo inherentemente diferentes a los problemas de un solo objetivo.

En resumen, al enfrentarse a un problema multi-objetivo, lo que se quiere obtener es el conjunto de soluciones que se mapean al frente Pareto. Si se está resolviendo un problema multi-objetivo con funciones continuas, lo que se quiere encontrar es una gama de puntos que cubran (discretamente) lo más posible del frente (continuo) Pareto.

Definición 3 *El Punto Ideal o Vector Ideal es el punto* $\mathbf{z}^*$ *que está compuesto por los mejores valores objetivo que se pueden obtener. Esto es,* $z_j^* = \min\{f_j(\mathbf{x}) | \mathbf{x} \in \mathbf{A}\}$, $\mathbf{A}$ *es el conjunto (sin restricciones) de*

todos los posibles valores de $\mathbf{x}$, $j = 1, 2, \ldots, m$, *donde* $m > 1$ *es el número de funciones objetivo.*

Basándonos en la Definición 3 se puede presentar el concepto de funciones objetivo en conflicto.

Definición 4 *Dos funciones objetivo están en conflicto si la distancia Euclidiana desde el punto ideal al Frente Pareto es diferente de cero.*

Definición 5 *Tres o más funciones objetivo están en conflicto si están en conflicto por pares.*

El hecho de que las funciones objetivo estén en conflicto hace que se tengan varias soluciones para el mismo problema multi-objetivo. Es decir, mientras un problema mono-objetivo se resuelve encontrando el mejor valor (que no implica una única solución) para una función dada; un problema multi-objetivo se resuelve encontrando un conjunto de vectores que no pueden ser comparados entre sí.

Es importante tener en cuenta que si las funciones f_i son convexas, las g_i son cóncavas y las h_i lineales, entonces el problema anterior es un problema de optimización convexa para los cuales se cuentan con algoritmos eficientes para resolverlos de manera exacta. Para detalles ver el trabajo de Boyd y Vandenberghe [51].

Existe una vasta literatura sobre optimización multi-objetivo. Para el caso continuo tenemos los enfoques clásicos de Steuer [434] y Miettinen [300]. Para problemas combinatorios tenemos los trabajos de Ehrgott [131] y Pardalos [356]. Los enfoques evolutivos empezaron a ser más frecuentes después de las monografías de Deb [112] y Coello [90].

6.3 Algoritmos evolutivos

La optimización evolutiva toma su nombre precisamente de emplear algoritmos evolutivos como técnica fundamental de optimización. Estos algoritmos constituyen un subárbol muy importante de la familia de enfoques estocásticos de optimización. Decimos subárbol porque éste, a su vez, pertenece a la rama del cómputo natural (o bio-inspirado) y se divide en otras como algoritmos genéticos, estrategias evolutivas, programación evolutiva, entre muchas otras[2]. Estas técnicas están inspiradas en los principios de la evolución neodarwinista (en su mayoría; también hay trabajos que se basan en algún tipo de evolución lamarckista).

La analogía es simple: se codifica un vector de decisión $\mathbf{x}$ (el fenotipo) en una representación abstracta (genotipo), al que se denomina individuo, y su imagen $f(\mathbf{x})$ en el espacio de criterios captura su aptitud. Entonces, de una población de individuos, aquellos individuos más aptos serán los que tengan una mayor probabilidad de tener descendencia y preservar sus genes en la siguiente generación. Los algoritmos evolutivos fueron originalmente pensados sin una consideración explícita para manejar múltiples funciones objetivo. No obstante, y sin modificar la esencia de la analogía, muchos de éstos han sido hábilmente adaptados para solucionar problemas de optimización multi-objetivo.

La aptitud suele reflejar qué tan bueno es un individuo con respecto a la función objetivo que estamos optimizando. Cuando hablamos

[2]Cabe aclarar que el uso de los algoritmos evolutivos, o del cómputo natural, no es exclusivo de los problemas de optimización aunque sí es de lo más popular.

de optimización mono-objetivo, la aptitud se acostumbra medir reciclando el valor de su función objetivo y es trivial decidir qué individuo es más apto que otro. Sin embargo, esto cambia drásticamente cuando contemplamos múltiples objetivos. Por ejemplo, suponiendo que estamos minimizando dos objetivos $f_1(\cdot)$ y $f_2(\cdot)$, dados un par de individuos $\mathbf{x}$ y $\mathbf{x}'$ mutuamente no dominados, con valores $f_1(\mathbf{x}) < f_1(\mathbf{x}')$ y $f_2(\mathbf{x}') < f_2(\mathbf{x})$, ¿cuál de los dos es el más apto? Claramente, el individuo $\mathbf{x}$ es más apto que el individuo $\mathbf{x}'$ con respecto a $f_1(\cdot)$; mientras que el caso inverso ocurre con respecto a $f_2(\cdot)$. Recordemos que la solución de un problema de optimización multi-objetivo equivale a encontrar el conjunto óptimo de Pareto $\mathbf{P}$ el cual se transforma, a través de las funciones objetivo, a un frente de Pareto $\mathbf{PF}$. De manera que el modelo de aptitud que se adopte debe propiciar, o al menos no entorpecer, la búsqueda de este conjunto.

El esquema general de un algoritmo evolutivo (ver Algoritmo 15) se compone de cinco módulos: Inicializar-población, Evaluación-de-aptitud, Selección, Variación, y Reemplazo. Los pormenores de cada módulo son inherentes al tipo de técnica evolutiva que se desee implementar. No obstante, de manera general, cumplen con la siguiente descripción de funcionamiento. Inicializar-población genera, de manera (semi-)aleatoria, la población que será sujeta al proceso de evolución. Evaluación-de-aptitud es la rutina que calcula y asigna el valor de aptitud a cada individuo del conjunto evaluado. Selección es la rutina encargada de extraer un subconjunto de individuos (padres) de la población que serán sometidos a los operadores de variación. Variación es la rutina que lleva a cabo las operaciones

de recombinación y/o mutación, del subconjunto de la población seleccionada, que da como resultado la descendencia. Finalmente, REEMPLAZO es la rutina que se encarga de preservar aquellos individuos más aptos, ya sea de la población más la descendencia, o sólo de la descendencia.

Algoritmo 15 Algoritmo Evolutivo genérico

1: $P_0 \leftarrow$ INICIALIZAR-POBLACIÓN(N)
2: EVALUACIÓN-DE-APTITUD(P_0)
3: $t \leftarrow 0$
4: **mientras** no Criterio de Paro($\mathcal{P}$) **hacer**
5: $S_t \leftarrow$ SELECCIÓN(P_t)
6: $P'_t \leftarrow$ VARIACIÓN(S_t)
7: EVALUACIÓN-DE-APTITUD(P'_t)
8: $P_{t+1} \leftarrow$ REEMPLAZO(P_t, P'_t)
9: $t \leftarrow t + 1$
10: **fin mientras**

Una observación importante: aunque los algoritmos evolutivos se clasifican como métodos estocásticos esto no implica que estén haciendo una búsqueda aleatoria. Otra forma de decir esto es que el éxito de aproximar el frente Pareto, en el caso multiobjetivo, no se debe atribuir meramente a la casualidad. Como ya hemos visto, estos algoritmos implementan mecanismos de selección para la recombinación y la preservación de soluciones basadas en su calidad.

6.4 Técnicas de optimización evolutiva multi-objetivo

A más de treinta años de la primer propuesta [403], existe una larga lista de técnicas evolutivas para la optimización multiobjetivo. La comunidad científica que trabaja en esta área coincide en que hemos sido testigos de al menos dos generaciones de estas técnicas y que estamos a la espera de una tercera.

El problema de aproximar el frente Pareto es en esencia multi-objetivo. Se desea, por un lado, minimizar la distancia generacional de una población mientras que, por otro lado, se desea maximizar la diversidad de ésta sobre el frente aproximado. La metodología con la que se persigue este frente, en el marco del esquema presentado en la sección anterior, da origen a las diferentes técnicas evolutivas de optimización multi-objetivo.

A continuación, a manera de antología, revisaremos brevemente algunas de las técnicas evolutivas de segunda generación (mayormente) más utilizadas para la optimización multi-objetivo.

6.4.1 Funciones de agregación

Esta metodología aunque no es de segunda generación, sí es una de las más sencillas y directas de entender e implementar. Consiste en combinar el valor de los múltiples objetivos en un solo valor escalar. De esta manera, el problema se devuelve al dominio mono-objetivo, lo que facilita su implementación en un algoritmo evolutivo genérico. Una función de agregación s se define como una función

$s : \mathbb{R}^k \to \mathbb{R}$ en la que se corresponde un vector $f(x) \in \mathbb{R}^k$ a un valor real $s(f(\mathbf{x})) \in \mathbb{R}$. Para ello, se requiere adicionalmente un vector de pesos $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_m]^T$ con un peso por cada objetivo (suponiendo que estamos optimizando m objetivos). Entre las implementaciones más populares están la suma ponderada [179, 505] y la de Tchebycheff [49]; otras técnicas de este enfoque pueden encontrarse en el libro de Miettinen [300]. Un ejemplo de la suma ponderada luce como

$$s(f(\mathbf{x})) = \sum_{i=1}^{m} \lambda_i f_i(\mathbf{x}),$$

donde $\lambda_i \geq 0$ es el coeficiente que determina el peso que se otorga al i-ésimo objetivo. Por lo general también se establece que $\sum_i \lambda_i = 1$. Es importante señalar que las soluciones resultantes dependen de los valores que componen al vector λ, pudiendo variar drásticamante al variar los valores del vector. Adicionalmente, para encontrar un frente distribuido es necesario hacer varias corridas con variaciones al vector de pesos.

6.4.2 NSGA-II

El NSGA-II fue propuesto por Deb y colaboradores a principios de siglo [114, 118]. Como su nombre sugiere, este método es la segunda propuesta de la estirpe NSGA (*Non-dominated Sorting Genetic Algorithm*); que hasta ahora ha sido, por mucho, la más popular de tres [432, 118, 117]. Es un método basado en dominancia (Pareto) en donde la población se particiona en un número variable k de subcon-

juntos, llamados frentes, de individuos no-dominados entre sí[3]. Los k frentes son categorizados y ordenados como $F_1, F_2, \cdots, F_k$ de manera que para todo individuo $x \in F_i$, no existe otro individuo x' en algún frente con índice mayor que domine a x, es decir, $\nexists x' \in F_{i+j} | x' \prec x$ para $j = 1, 2, \cdots, k - i$. Una vez particionada la población en frentes no-dominados, contamos con una medida de aptitud para comparar individuos conocida como profundidad de dominación. La profundidad de dominación es un escalar que responde a qué frente pertenece un individuo, donde a menor profundidad mayor aptitud (ver figura 6.1). Además, proponen un método, al que llaman distancia de amontonamiento, para discernir entre individuos de un mismo frente no-dominado con el objetivo de maximizar la diversidad sobre el frente. Esta distancia pretende capturar, por frente, qué tan distante está un individuo con respecto a sus vecinos más próximos en cada objetivo. Un individuo con una mayor distancia de amontonamiento implica un mayor aislamiento en su frente. El criterio de comparación de individuos se hace de manera binaria y en dos niveles; primero se compara por profundidad de dominancia, en caso de empate, se compara por distancia de amontonamiento.

6.4.3 SPEA2

El SPEA2, propuesto por Zitzler y colaboradores [523], también es una segunda generación de un método que refina los detalles de su propuesta previa, el SPEA (*Strenght Pareto Evolutionary Algorithm*) [524, 525]. También es un método basado en dominancia; con la car-

[3]Sugerido por David Goldberg.

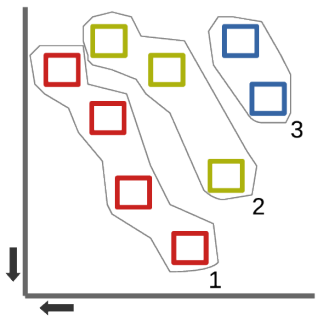


Figura 6.1: Profundidad de dominación. En rojo, el frente F_1 de menor profundidad. En amarillo, el frente F_2 de profundidad media. En azul, el frente F_3 de mayor profundidad.

acterística de que implementa y mantiene un conjunto externo de individuos no-dominados, denominado archivo A_t, en paralelo con la población regular P_t. A cada individuo de la población y del archivo se le asigna un valor llamado fuerza (*strength*) definido como $S(i) = |\{j \in P_t \cup A_t | i \prec j\}|$, es decir, el conteo de dominancia de i. A partir de los valores de fuerza, se calcula el valor bruto de aptitud (*raw fitness*) definido como $R(i) = \sum_{j \in P_t \cup A_t | j \prec i} S(j)$. Así, los individuos i con valor bruto de aptitud $R(i) = 0$ son aquellos no-dominados, mientras que aquellos con los valores $R(i)$ más altos son los que están dominados por varios individuos también dominados (ver figura 6.2). Adicionalmente, como medida de densidad, por cada individuo i, se calcula una razón de la distancia σ_i^k a su k-ésimo vecino más cercano en el espacio de criterios, esto es, $D(i) = \frac{1}{\sigma_i^k + 2}$. Finalmente, la aptitud de un individuo i está definida como $F(i) = R(i) + D(i)$, donde su

parte entera describe dominancia (sobre éste) y su parte fraccionaria describe su aislamiento.

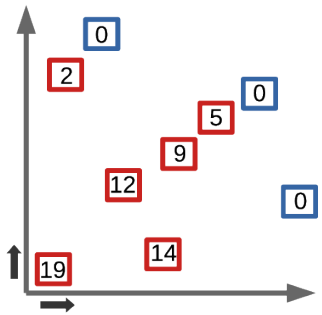


Figura 6.2: Valor bruto de aptitud. En azul, el frente no dominado; en rojo, la imagen de las soluciones dominadas. Todos los elementos del conjunto no dominado tienen un valor bruto de aptitud de cero. Aquellas soluciones dominadas tienen un valor bruto de aptitud igual a la suma del conteo de dominancia de quienes las dominan.

En cada iteración, un nuevo archivo A_{t+1} admite un número fijo N de individuos no-dominados del conjunto unión de la población anterior y el archivo anterior $(P_t \cup A_t)$. El caso interesante ocurre cuando hay más de N individuos no-dominados y es necesario truncar. Este truncamiento se hace con base en la aptitud $(F(\cdot))$ de los individuos, asegurándonos de conservar siempre a los individuos en los extremos de los frentes.

6.4.4 MOEA/D

El MOEA/D (*Multi-Objective Evolutionary Algorithm based on Decomposition*), propuesto por Zhang y Li [512], es un método basado

en descomposición en el que, explícitamente, se descompone un problema multi-objetivo en N subproblemas mono-objetivo que son optimizados simultáneamente en una sola ejecución. La descomposición se realiza mediante N funciones de agregación $s_i(\cdot)$ diferenciadas por su vector de pesos correspondiente λ^i, con $i = 1, 2, \cdots, N$ (ver Subsección 6.4.1). De manera que cada subproblema i optimiza su propia función de agregación $s_i(\cdot)$ como función objetivo. La población $P_t = \{x^1, x^2, \cdots, x^N\}$ se compone de N soluciones, siendo x^i la mejor solución encontrada al momento t para $s_i(x^i)$. La idea detrás de este método es que la diversidad inherente de cada subproblema promueva la diversidad en la población. Dado que cada vector de peso optimiza un subproblema, la variación suele ser otro aspecto clave de este método. Para esto, se calcula el vecindario de cada individuo x^i, definido como $B(i) = \{i_1, i_2, \cdots, i_T\}$ tal que $\{\lambda^{i_1}, \lambda^{i_2}, \cdots, \lambda^{i_T}\}$ son los T vectores más cercanos a λ^i. Por cada individuo x^i, se seleccionan aleatoriamente dos índices $k, \ell \in B(i)$ para producir un nuevo individuo y a partir de x^k y x^ℓ. Luego se actualizan las soluciones del vecindario. Por cada vecino $k \in B(i)$, si $s_k(y) \leq s_k(x^k)$, entonces $x^k = y$. Además de lo anterior, el MOEA/D también implementa un archivo A_t como población externa que se actualiza al final de cada iteración t y es la manera en la que interactúan los subproblemas traslapantes. De manera que si $f(y) \prec f(z) \in A_t$, entonces se retira z del archivo. Finalmente, si $\nexists z \in A_t | f(z) \prec f(y)$, entonces se agrega y al archivo A_t.

6.4.5 SMS-EMOA

El SMS-EMOA (*S metric selection evolutionary multiobjective optimisation algorithm*), propuesto por Beume y colaboradores [43], es una metodología basada en indicadores. Este tipo de metodologías conciben el problema de optimización multi-objetivo en términos de algún indicador que mide la calidad del frente resultante (se describen algunos de estos indicadores en la Sección 6.5). Particularmente, el SMS-EMOA emplea un indicador conocido como hipervolumen que, dado un punto de referencia, captura el tamaño de la región de dominancia de un frente (ver figura 6.3). La idea es entonces encontrar una población P que maximice el hipervolumen $\mathcal{HV}(P)$. El SMS-EMOA, al igual que como se hace en el NSGA-II, particiona su población en frentes no-dominados ordenados $F_1, F_2, \cdots, F_k$. Durante el reemplazo, se descarta un individuo de aquel frente no-dominado con el índice más grande; particularmente, se descarta el $\arg\min_{i \in F_k}(\mathcal{HV}(F_k) - \mathcal{HV}(F_k \backslash \{i\}))$. Es por esto último que, como invariante de este método, el valor del hipervolumen no empeora con cada nueva generación $t+1$, es decir, $\mathcal{HV}(P_t) \leq \mathcal{HV}(P_{t+1})$. Otra técnicas basadas en este indicador son el MO-CMA-ES [217] y el HypE [34].

6.4.6 Observaciones

Es importante mencionar que estos métodos se conocen como técnicas *a posteriori* [4], donde el proceso de toma de decisión ocurre al finalizar la ejecución luego de obtener los resultados. Es por ello fundamental

[4]Existen otras dos clasificaciones de técnicas que no revisaremos en este capítulo, éstas son las técnicas *a priori* y las técnicas progresivas.

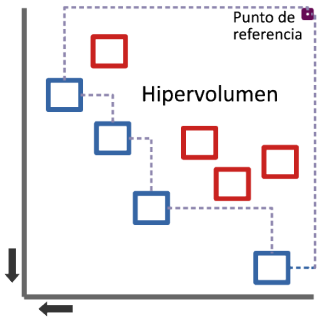


Figura 6.3: Hipervolumen, en este caso de dos dimensiones, el área del polígono delimitado por el perímetro punteado. En azul, el frente no dominado; en rojo, la imagen de las soluciones dominadas. En morado, el punto de referencia.

que el conjunto Pareto de soluciones encontradas estén lo más dispersas como sea posible sobre el frente.

Otro aspecto importante es que estas técnicas fallan, en general, en conseguir un buen cubrimiento del frente Pareto, cuando se trata de optimizar más de tres objetivos simultáneamente [231, 216]. Incluso se ha reportado que algunas técnicas de segunda generación tienen un desempeño comparable con el de una búsqueda aleatoria en dimensiones tan bajas como diez objetivos [233]. Esto no es coincidencia, se ha reconocido un paradigma anidado en el paradigma multi-objetivo conocido como muchos-objetivos (*many-objective*) [479]. Para ello se han propuesto técnicas especializadas en este paradigma, como el NSGA-III [117]. Se habla más sobre la perspectiva de esta área en la Sección 6.7.

Las técnicas descritas en esta sección, con excepción de las funciones de agregación, además de ser muy utilizadas para la optimización de problemas multi-objetivo, son comúnmente empleadas para comparar su desempeño ante los nuevos métodos propuestos. No obstante, no hemos mencionado cómo hacer comparaciones cuantitativas de calidad o desempeño entre métodos.

Cuando se cuenta con dos o más métodos distintos para resolver un problema multi-objetivo, es importante saber cuál produce los mejores resultados. Esto no es tan fácil como en el caso mono-objetivo, en el que cada método produce una solución única y el mejor método es el que produce la solución menor o mayor según corresponda (minimizar o maximizar). En el caso multi-objetivo, la comparación debe realizarse entre *conjuntos de soluciones*. Con el objetivo de poder comparar estos conjuntos se han propuesto varios criterios, los cuales se explican en la siguiente sección.

6.5 Criterios para la evaluación de calidad de conjuntos no-dominados

El término métrica empleado en [232] y en [471] no cumple todas las propiedades requeridas por una definición formal de dicho concepto como se establece en análisis funcional [170]. Por esta razón las llamaremos de aquí en adelante criterios de calidad. Los criterios que se utilizan son un mapeo que toma como argumento un conjunto de soluciones no dominadas y le asigna un número real. Además, estos

criterios están pensados para evaluar distintos aspectos de los conjuntos de soluciones:

1. Cercanía al frente Pareto. Evalúa qué tan alejadas del frente Pareto están las soluciones obtenidas. Los criterios que caen en esta categoría son: Razón de Error, Distancia Generacional, Distancia Generacional Invertida y Error Máximo del Frente Pareto.

2. Distribución. Evalúa qué tanto del frente Pareto está siendo cubierto por las soluciones obtenidas. El hipervolumen es un ejemplo de esto.

3. Dispersión. Indica qué tan equidistantes sobre el frente Pareto están las soluciones obtenidas. Por ejemplo, el criterio espaciado (*spacing*). Este criterio está pensado para frentes continuos.

Desafortunadamente, no se conoce un criterio único que evalúe los tres aspectos a la vez. Por el contrario, es un problema multi-objetivo decidir qué conjunto de soluciones es el mejor debido a que puede haber dos o más conjuntos mejores dependiendo del criterio seleccionado.

Antes de presentar los diferentes criterios, se introducen primero algunas definiciones [202].

Si se tienen dos conjuntos $\mathbf{A}$ y $\mathbf{B}$, $ND(\mathbf{A})$ ={Conjunto de soluciones No-Dominadas de $\mathbf{A}$} se define:

Definición 6 Superioridad débil.
$\mathbf{A}O_W\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ *y* $\mathbf{A} \neq \mathbf{B}$. $\mathbf{A}$ supera débilmente a $\mathbf{B}$ *si todos los puntos de* $\mathbf{B}$ *son 'cubiertos' por los puntos de* $\mathbf{A}$ *('cubiertos' significa domina o es igual) y hay al menos un punto en* $\mathbf{A}$ *que no está en* $\mathbf{B}$. *Esto se ilustra en la figura 6.4 (Izquierda).*

Definición 7 Superioridad fuerte.
$\mathbf{A}O_S\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ *y* $\mathbf{B} \backslash ND(\mathbf{A} \bigcup \mathbf{B}) \neq \emptyset$. $\mathbf{A}$ supera fuertemente a $\mathbf{B}$ *si todos los puntos en* $\mathbf{B}$ *son cubiertos por los puntos de* $\mathbf{A}$ *y algún punto en* $\mathbf{B}$ *es dominado por un punto en* $\mathbf{A}$. *La figura 6.4 (Centro) muestra un ejemplo.*

Definición 8 Superioridad completa.
$\mathbf{A}O_C\mathbf{B} \Leftrightarrow ND(\mathbf{A} \bigcup \mathbf{B}) = \mathbf{A}$ *y* $\mathbf{B} \bigcap ND(\mathbf{A} \bigcup \mathbf{B}) = \emptyset$. $\mathbf{A}$ supera completamente a $\mathbf{B}$ *si cada punto en* $\mathbf{B}$ *es dominado por un punto en* $\mathbf{A}$. *La figura 6.4 (Derecha) ilustra esta definición.*

Como se ve de las definiciones 6, 7 y 8: $\mathbf{A}O_C\mathbf{B} \Rightarrow \mathbf{A}O_S\mathbf{B} \Rightarrow \mathbf{A}O_W\mathbf{B}$. Esto es, *superioridad completa* es la más estricta de las relaciones y *superioridad débil* es la menos estricta.

En [202] se proponen los siguientes tipos de compatibilidad:

Definición 9 Compatibilidad débil.
Una relación de comparación $\preceq_R$ *es* débilmente compatible *con una medida de desempeño* R *si para cada par de conjuntos no-dominados* $\mathbf{A}, \mathbf{B}$ *con* $\mathbf{A} \preceq_R \mathbf{B}$, R *evaluará que* $\mathbf{A}$ *no es mejor que* $\mathbf{B}$.

Definición 10 Compatibilidad.
Una relación de comparación $\preceq_R$ *es* compatible *con una medida de desempeño* R *si para cada par de conjuntos no-dominados* $\mathbf{A}$ *y* $\mathbf{B}$, *tales que* $\mathbf{A} \preceq_R \mathbf{B}$, R *evaluará que* $\mathbf{A}$ *es mejor que* $\mathbf{B}$.

Las definiciones anteriores nos sirven para saber qué características tiene un criterio R en particular. Si, por ejemplo, el criterio R es compatible con O_W significa que al tener dos conjuntos ($\mathbf{A}$ y $\mathbf{B}$) cuyas

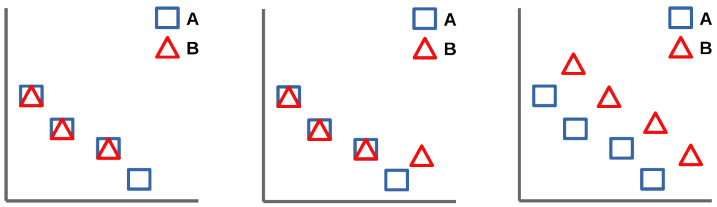


Figura 6.4: Relaciones de superioridad. Izquierda: Superioridad débil ($\mathbf{A}O_W\mathbf{B}$). Centro: Superioridad fuerte ($\mathbf{A}O_S\mathbf{B}$). Derecha: Superioridad completa ($\mathbf{A}O_C\mathbf{B}$).

soluciones se ven como en la figura 6.4 (Izquierda), R va a evaluar correctamente que **A** es el mejor conjunto. Si por el contrario, R no es compatible con O_W entonces evaluará que el mejor conjunto es **B**.

Los criterios de calidad se clasifican en unarios y binarios, según cuántos conjuntos no domidados toman como entrada. Los unarios mapean un conjunto no dominado a un valor real, y la comparación entre conjuntos no dominados se realiza mediante la comparación directa de estos valores reales. Los criterios binarios son aquellos que comparan dos conjuntos no dominados y producen un valor real, el valor de ese número informa sobre el resultado de la comparación.

6.5.1 Criterios unarios

Para todos los criterios siguientes, sea **A** el conjunto soluciones no dominadas. Todos los criterios se aplican sobre este conjunto.

6.5.1.1 Hipervolumen ($\mathcal{HV}$)

El $\mathcal{HV}$ propuesto originalmente por Zitzler y Thiele [525] calcula el hipervolumen de la región multidimensional encerrada por $\mathbf{A}$ y un punto de referencia $\mathbf{r} = (r_1, \cdots, r_m) \in R^m$. Éste se calcula de la siguiente manera [60]:

$$HV(\mathbf{A}, \mathbf{r}) = \lambda(\bigcup_{a \in \mathbf{A}} [f_1(a), r_1] \times \cdots \times [f_m(a), r_m]), \tag{6.3}$$

donde $\lambda(S)$ es la medida de Lebesgue del conjunto S. Este criterio está diseñado para evaluar la distribución del frente no-dominado. Mientras mayor sea el valor que el conjunto obtiene en este criterio, mejor es el conjunto (en cuanto a distribución). Este criterio es compatible con O_W siempre y cuando el punto de referencia se escoja de forma que cualquier punto no-dominado válido sea evaluado positivo. Una desventaja de este indicador es la selección del punto de referencia [521], una inadecuada selección de esta referencia puede dar lugar a evaluaciones erróneas. Debido al alto costo computacional de calcular este criterio, varios algoritmos han sido propuestos para hacer el cálculo cada vez más eficiente. Ejemplos de estos son las propuestas de Lacour et al. [245] y Jaszkiewicz [225].

6.5.1.2 Razón de Error (Error Ratio - ER)

Está definido en [463] como $\sum_{i=1}^{n} \mathbf{a}_i/n$ donde n es el número de vectores en el conjunto de soluciones $\mathbf{A}$; $\mathbf{a}_i = 0$ si el vector i está en $\mathbf{PF}$ y 1 de otra forma. Este criterio se enfoca en el aspecto de cercanía al frente Pareto. Un valor de 0 significa que todos los puntos están

en el frente Pareto real. Es débilmente compatible con O_C y no es débilmente compatible con O_S u O_C. Este criterio es fácil de entender y de calcular, aunque se necesita conocimiento del **PF**.

6.5.1.3 Distancia Generacional (Generational Distance - GD)

Descrito en [463] está dado por $GD = \sqrt{\sum_{i=1}^{n} d_i^2}/n$, donde n es el número de vectores en el conjunto de soluciones $\mathbf{A}$, y d_i es la distancia en el espacio de criterios entre el vector i y el miembro *más cercano* de **PF**. Esta es otra forma eficiente de medir la cercanía al frente Pareto. Si el conjunto tiene todos sus puntos en el frente Pareto, obtendrá un valor de 0 en este criterio. No es débilmente compatible con O_W, pero es compatible con O_S; sin embargo, viola la monotonía débil (que exige que al agregar una solución no dominada, el criterio no empeore). Al verificar la cercanía al frente real se requiere conocimiento de **PF**.

6.5.1.4 Distancia Generacional Invertida (Inverted Generational Distance - IGD)

Dada una muestra $\mathbf{P}^*$ del frente Pareto **PF** y un frente no dominado $\mathbf{A}$ generado por un algoritmo, la distancia generacional invertida se define como:

$$IGD(\mathbf{P}^*, \mathbf{A}) = \sum_{v \in P^*} d(v, \mathbf{A})/|\mathbf{P}^*|, \tag{6.4}$$

donde $d(v, \mathbf{A})$ es la distancia euclidiana de la solución v a la solución más cercana en $\mathbf{A}$. En un trabajo reciente [218], Ishibuchi y colaboradores mostraron que, al igual como ocurre con el punto de referencia

para el cálculo del hipervolumen, esta medida puede introducir sesgos no deseados en función de la selección de la muestra $\mathbf{P}^*$ del **PF**.

6.5.1.5 Error Máximo del Frente Pareto (Maximum Pareto Front Error - MPFE)

De acuerdo a [463] el criterio MPFE se define como:

$$MPFE = \max_j(\min_i(\left|f_1^i(\mathbf{x}) - f_1^j(\mathbf{x}^*)\right|^p + \left|f_2^i(\mathbf{x}) - f_2^j(\mathbf{x}^*)\right|^p))^{1/p} \quad (6.5)$$

donde $\mathbf{x}^* \in \mathbf{P}$ y además $i = 1, \ldots, n_1$ y $j = 1, \ldots, n_2$ son los índices de los vectores en **A** y en **PF**, respectivamente. MPFE mide la mayor distancia entre cualquier vector en **A** y su vector correspondiente más cercano en **PF**. Este criterio también mide la cercanía al frente Pareto. Los conjuntos que obtienen menores valores son mejores, ya que un valor de 0 en este criterio significa que todos los puntos están en el frente Pareto. No es débilmente compatible con cualquier relación de superioridad y viola la monotonía débil. Una ventaja de este criterio es que es fácil de calcular. Aunque una desventaja es que también requiere conocimiento del **PF**.

6.5.1.6 Espaciamiento de Schott (Schott's Spacing metric - SS)

Este criterio se define como [404]:

$$SS = \sqrt{\frac{1}{n-1}\sum_{i+1} n(\bar{d} - d_i)^2} \quad (6.6)$$

donde $d_i = \min_j(|a_1^i - a_1^j| + |a_2^i - a_2^j| + \ldots + |a_m^i - a_m^j|)$. $i, j = 1, \ldots, n$. $\bar{d}$ es la media de todas las d_i y $n = |\mathbf{A}|$. SS intenta medir qué tan uniformemente distribuidos están los puntos, por lo que un valor de 0 indica que todos los puntos son equidistantes. El criterio no es débilmente compatible con O_W y tiene un bajo costo computacional. Este criterio implícitamente supone que el Frente Pareto del problema analizado está uniformemente distribuido.

6.5.2 Criterios binarios

6.5.2.1 La cobertura de conjunto $\mathcal{C}$ (The $\mathcal{C}$ metric) [234]

Sean $\mathbf{A}, \mathbf{B}$ dos conjuntos de vectores no-dominados. $\mathcal{C}$ mapea el par ordenado $(\mathbf{A}, \mathbf{B})$ al intervalo [0,1]:

$$\mathcal{C}(\mathbf{A}, \mathbf{B}) = \frac{|\{\mathbf{b} \in \mathbf{B} | \exists \mathbf{a} \in \mathbf{A} : \mathbf{a} \preceq \mathbf{b}\}|}{|\mathbf{B}|} \tag{6.7}$$

El valor $\mathcal{C}(\mathbf{A}, \mathbf{B}) = 1$ significa que todos los vectores de $\mathbf{B}$ son dominados por $\mathbf{A}$. Lo opuesto, $\mathcal{C}(\mathbf{A}, \mathbf{B}) = 0$, significa que ninguno de los puntos de $\mathbf{B}$ es dominado por $\mathbf{A}$. Pero $\mathcal{C}(\mathbf{A}, \mathbf{B})$ no es necesariamente $1 - \mathcal{C}(\mathbf{B}, \mathbf{A})$; esto se puede apreciar en la figura 6.5. Este es un criterio que intenta explicar cuál es la relación entre el conjunto $\mathbf{A}$ y el conjunto $\mathbf{B}$. En general $\mathcal{C}$ no es compatible con O_W. Sin embargo, sí es compatible con O_S y con O_C. Si hay dos conjuntos en los cuales no se cumple que $\mathcal{C}(\mathbf{A}, \mathbf{B}) = 1$ y tampoco $\mathcal{C}(\mathbf{B}, \mathbf{A}) = 1$ entonces indica que los dos conjuntos son incomparables en términos de la relación de superioridad débil. Este criterio tiene la ventaja de ser independiente de la escala y además no requiere conocer el **PF**.

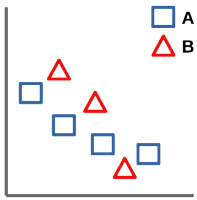


Figura 6.5: Ejemplo de un caso en que $\mathcal{C}(\mathbf{A},\mathbf{B}) \neq 1 - \mathcal{C}(\mathbf{B},\mathbf{A})$. $\mathcal{C}(\mathbf{A},\mathbf{B}) = 2/3$. $\mathcal{C}(\mathbf{B},\mathbf{A}) = 1/4$

6.5.2.2 Criterio $\mathbf{I}_\epsilon(\mathbf{A},\mathbf{B})$ (Epsilon indicator)

Este criterio se define como [521]:

$$\mathcal{I}_\epsilon(\mathbf{A},\mathbf{B}) = \inf_{\epsilon \in R}\{\forall b \in \mathbf{B} | \exists \mathbf{a} \in \mathbf{A} : \mathbf{a} \preceq_\epsilon \mathbf{b}\} \tag{6.8}$$

Este indicador se base en la dominancia epsilon la cual se define como [521]:

$$\mathbf{b} \preceq_\epsilon \mathbf{a} \Longleftrightarrow \forall i \in 1..n : f_i(\mathbf{b}) \leq \epsilon f_i(\mathbf{a}) \tag{6.9}$$

Esto considereando minimización y que todos los puntos son positivos en todos los objetivos.

6.5.2.3 Criterio R1 (R1 metric)

Definido en [202] como:

$$R1(\mathbf{A},\mathbf{B},\mathbf{U},p) = \int_{u \in \mathbf{U}} C(\mathbf{A},\mathbf{B},u)p(u)du$$

$$C(\mathbf{A},\mathbf{B},u) = \begin{cases} 1 & u^*(\mathbf{A}) > u^*(\mathbf{B}) \\ 1/2 & u^*(\mathbf{A}) = u^*(\mathbf{B}) \\ 0 & u^*(\mathbf{A}) < u^*(\mathbf{B}) \end{cases}$$

donde $\mathbf{A}$ y $\mathbf{B}$ son dos conjuntos de soluciones, $\mathbf{U}$ es un conjunto de funciones de utilidad que asignan a cada punto en el espacio de criterios una medida de utilidad, $p(u)$ es la función de densidad de probabilidad de la utilidad $u \in \mathbf{U}$, y $u^*(\mathbf{A}) = \max_{\mathbf{z}\in\mathbf{A}}\{u(\mathbf{z})\}$ y lo mismo para $\mathbf{B}$. Este criterio introduce cierto conjunto de funciones de utilidad, tratando de "adivinar" las preferencias de quien toma las decisiones. Estas preferencias dependen tanto del problema como de la forma de la solución y las estudia el que toma las decisiones (*decision maker* o *DM* es como suele llamársele, dentro del área de Investigación de Operaciones). Lo que $\mathbf{R}1$ hace es calcular la probabilidad de que $\mathbf{A}$ sea mejor que $\mathbf{B}$ *sobre* este conjunto de funciones de utilidad. Sin embargo, la funcionalidad de este criterio depende de la selección del conjunto de funciones de utilidad, las cuales se pueden escoger sin conocer **PF**. En cuanto a la compatibilidad con las relaciones de superioridad si $\mathbf{U}(A > B) = \{u \in \mathbf{U} | u^*(\mathbf{A}) > u^*(\mathbf{B})\}$ y si $p(\mathbf{u})$ es tal que la probabilidad de seleccionar una función de utilidad $u \in \mathbf{U}(\mathbf{A} > \mathbf{B})$ es positiva siempre que $\mathbf{U}(\mathbf{A} > \mathbf{B}) \neq \emptyset$ entonces **R1** es compatible con O_W.

Es importante resaltar que los criterios unarios más aceptados son el $\mathcal{HV}$ y las GD y IGD, mientras que los binarios más usados son $\mathcal{I}_\epsilon(\mathbf{A},\mathbf{B})$ y $C(\mathbf{A},\mathbf{B})$. Estos dos últimos son usados cuando se requiere comparar conjuntos no dominados para problemas donde no se conoce el **PF**, en estos casos el $\mathcal{HV}$ también es frecuentemente usado.

6.6 Optimización evolutiva multi-objetivo en México

En México, existe una comunidad científica muy activa que ha encontrado su nicho en la optimización evolutiva multi-objetivo. Esta comunidad creció y maduró, de la mano de un puñado de investigadores, hasta consolidar grupos de trabajo reconocidos dentro y fuera del país. Los grupos que más se destacan se encuentran en instituciones académicas tales como el CINVESTAV-IPN, CINVESTAV-Tamaulipas, CICESE, CIMAT, UAM-Cuajimalpa, UAM-Azcapotzalco, y CIIA-UV.

Sin duda, un gran porcentaje de las contribuciones en el área de multi-objetivo en México se deben al grupo de Carlos Coello y colaboradores, en CINVESTAV-IPN[5]. Contribuciones importantes en el manejo de restricciones [85, 86, 205], extensiones de algoritmos mono-objetivo a versiones multi-objetivo [89, 96, 92], resolución de problemas en distintas disciplinas [88], solo por citar una muestra. Una descripción detallada de los aportes de Coello y quienes se formaron bajo su tutela requeriría de un espacio no disponible en este capítulo.

De los trabajos realizados predominantemente en México, se enlistan a continuación algunos de los más relevantes y más recientes en el área; nos enfocamos en aplicaciones, métodos novedosos, indicadores de calidad, multi-objetivización y optimización de muchos objetivos.

[5]Popularmente conocidos como el grupo de *EVOCINV*; su página se encuentra en: http://www.cs.cinvestav.mx/~EVOCINV.

Aplicaciones: Entre éstas se encuentran la predicción de estructura de proteínas [177, 178], optimización estructural [84], optimización de forma [359], diseño aerodinámico (*airfoil design*) [273], enrutamiento de vehículos [173, 174], generación de prototipos (*prototype generation*) [223], diseño de antenas [355], diseño de secuencias de ADN [78], clasificación y diseño de péptidos [41, 40], enrutamiento de microfluidos en biochips digitales [229], formación de equipos [228] y respuesta a preguntas fundamentales en bioquímica computacional [97].

Métodos novedosos: Algunas propuestas de métodos novedosos, variaciones, e híbridos son: μGA [95, 94], un método de optimización evolutiva multi-objetivo que usa tamaños de población muy pequeños e implementa un mecanismo de reinicio; CAEP multi-objetivo [92], la primer propuesta que adapta un algoritmo cultural para la optimización multi-objetivo; MOPSO [96], la primer propuesta que adapta el método PSO para la optimización multi-objetivo; IS-PAES [205], una extensión del método PAES [235] para el manejo de problemas con restricciones; MRMOGA [270, 271], un método paralelizable de optimización evolutiva multi-objetivo basado en un enfoque de islas; HCS para algoritmos evolutivos [249], una hibridación de un algoritmo cultural con un procedimiento de búsqueda local para optimización multi-objetivo; MOMBI [206], un método de optimización evolutiva para muchos objetivos (*many-objective*) basado en el indicador R2; y recientemente, Δ_p-MOEA [292], un método basado en un indicador de distancia de Hausdorff; IGD$^+$-EMOA [285], un método basado en una variante del indicador IGD; RDS-NSGA-II [207], es una hibridación

entre un método de búsqueda local y una variante del método NSGA-II que contempla puntos de referencia; LIBEA [507], un método de optimización evolutiva multi-objetivo para problemas continuos con restricciones de intervalo, basado en un indicador de Lebesgue.

Otros: Sobre propuestas de indicadores de calidad, multiobjetivización, y optimización de muchos objetivos. Existen algunas propuestas de indicadores como G-metric [264], un indicador de calidad n-ario, y Δ_p [405], un indicador de calidad que mide la distancia Hausdorff promedio entre el frente resultante y el frente Pareto. También, algunas técnicas para el manejo de muchos objetivos como, la inclusión de preferencias en la búsqueda [272], métodos de clasificación alternativos (al de dominancia Pareto) como el indicador R2 [121], y métodos que preservan diversidad como la evolución (semi-)independiente de subpoblaciones [176]. Además de, un par de revisiones de literatura en el tema de multi-objetivización [409] con y sin restricciones [410].

6.7 Retos y perspectivas de investigación

A pesar del éxito alcanzado por la optimización evolutiva multi-objetivo durante sus más de tres décadas y de haber alcanzado un importante grado de madurez, esta área se encuentra lejos de estar cerrada. Existen numerosos retos importantes, algunos de éstos acompañan al área desde sus orígenes, siendo heredados desde su fuente, el cómputo evolutivo. A continuación enlistamos los retos que los autores consideramos más relevantes:

1. El desarrollo de suficientes bases teóricas que acompañen a la evidencia empírica. Una de las críticas más fuertes provenientes de otras áreas de optimización es justamente la falta de teoría sólida que justifique un diseño de experimento, o que ayude a explicar los resultados más allá de un análisis a posteriori.

2. El establecimiento de un puente entre las técnicas de optimización evolutiva multi-objetivo (como aquellas mencionadas anteriormente) y las preferencias del tomador de decisiones. Es común que el tomador de decisiones tenga de antemano una idea de qué tipo de soluciones pudiera preferir sobre otras. Aunque existen algunas propuestas (como los métodos interactivos), aún hay un reto muy grande en cómo incorporar eficazmente estas preferencias tempranas del usuario dentro de los métodos de optimización evolutiva multi-objetivo.

3. El diseño de criterios para la evaluación de calidad de los conjuntos no-dominados. Anteriormente se mencionó que a la fecha no se conoce criterio alguno que pueda evaluar unificadamente los aspectos de cercanía al frente Pareto, distribución, y dispersión, siendo probablemente el hipervolumen la mejor aproximación; lo que supone un problema al momento de comparar la calidad y el desempeño de las propuestas nuevas y las ya existentes. Queda claro que buscar un único criterio podría ser un esfuerzo sin mucho sentido, sin embargo, encontrar un conjunto pequeño de éstos podría ayudar a establecer un marco de comparación más justo.

4. A nivel de comunidad un reto asociado a la voluntad de los investigadores, en especial a la de los editores de las revistas, es tener mejores prácticas para hacer disponibles públicamente y sin costo, las implementaciones de los algoritmos y los datos empleados. El día que esto sea rutinario podremos considerar que el área evolucionó hacia el siguiente nivel de madurez. Es contra-intuitivo ver que en revistas de áreas de Ciencias de la Vida relacionadas con el desarrollo de algoritmos sea mandatorio hacer disponible programas y datos usados en la investigación y que para revistas asociadas al área de cómputo evolutivo siga siendo opcional.

Por otra parte, es amplia la avenida de líneas de investigación que resultan prometedoras en optimización evolutiva multi-objetivo, entre ellas podemos nombrar la optimización de muchos objetivos, problemas con funciones objetivo costosas, problemas en aprendizaje profundo y en biología computacional y la multiobjetivización. A continuación hacemos una breve descripción de cada una de estas áreas.

Many Objective. Un área floreciente de investigación dentro de optimización evolutiva multi-objetivo es la denominada optimización de muchos objetivos (*many-objective*, MaO) [479, 253, 257]. Esta incluye problemas de optimización con cuatro o más objetivos en conflicto que deben ser optimizados de forma simultánea. El primer problema que surge en optimización MaO es el elevado porcentaje de soluciones no dominadas que se obtienen desde las primeras generaciones del proceso evolutivo. Esto dificulta definir una dirección hacia dónde avanzar en la búsqueda. Para mitigar este problema, los algo-

ritmos para problemas MaO se han enfocado en desarrollar métodos basados en: relaciones de preferencia, agregación, descomposición, o basados en reducción de dimensionalidad del espacio de criterios. A la fecha varios algoritmos han sido propuestos para abordar los nuevos desafíos introducidos por los problemas MaO. Para un estudio comparativo de métodos recientes ver el trabajo de Li et al. [257]. Von Lucken et al. [480] presentan una descripción sucinta de aspectos relevantes donde se requieren de nuevos enfoques. Ellos destacan, entre otros problemas la visualización, el proceso de la selección de sobrevivientes y el desarrollo de técnicas de reducción de dimensionalidad. Mencionan también la necesidad de contar con mecanismos eficientes de preservación de diversidad para mantener una muestra uniforme de soluciones no dominadas.

Funciones objetivo y/o restricciones de alto costo. Los métodos para abordar esta clase de problemas son denominados metamodelos o subrogados de la función objetivo, ya que se basan en el uso de un modelo para estimar el valor de la función objetivo en una solución dada. Existen problemas para los cuales calcular la calidad de un solución puede ser muy demandante, por ejemplo, calcular la respuesta de un diseño de una válvula para un cohete espacial puede requerir de varias horas de simulación para una dinámica de fluidos computacional. Simular por dinámica molecular el movimiento por 90 ns de una proteína de 350 amino ácidos puede llevar 24 horas de tiempo de cómputo en una supercomputadora. Para estos casos pensar en evaluar cada individuo es simplemente irrealizable y se debe recurrir a modelos aproximados de la función de costo. El área inicia

hace poco más de 15 años [227] y existen una gran cantidad de problemas por resolver, entre ellos el desarrollo de meta-modelos efectivos para el cálculo de problemas dinámicos. Algunos aspectos básicos de optimización MO con métodos subrogados son presentados en [481]. Recientemente, Wang et al. [483] proponen un ensamble de métodos subrogados para problemas de optimización, donde la elección del algoritmo a usar del ensamble se basa en datos históricos y un modelo de aprendizaje de máquina. En un esquema denominado subrogado multi-problemas, Min et al. [301] exploran lo que denominan transferencia de conocimiento. En su enfoque proponen un esquema para utilizar información de optimización de problemas similares para producir el subrogado de la función objetivo del problema a optimizar, y el enfoque es aplicado a problemas multi-objetivo. Este trabajo es un primer paso y queda mucho por explorar en esta dirección, desde nuevos modelos de cómo transferir el conocimiento hasta el tipo de información a utilizar. Las preguntas fundamentales del área son [21]: i) qué meta-modelo usar, ii) qué aproximar y iii) cómo manejar las aproximaciones.

Aprendizaje Profundo. Hay una considerable cantidad de trabajo que explora la aplicación de técnicas de aprendizaje de máquina para mejorar los algoritmos evolutivos [510], así también los hay aplicando técnicas de cómputo evolutivo para mejorar algoritmos de aprendizaje de máquina; en particular en aprendizaje por refuerzo [489], selección de características [195, 40], clustering multi-objetivo [320], entre otros. Para una revisión general del área ver el trabajo de Xue

et al [496]. Se han aplicado también técnicas evolutivas tanto para aprendizaje supervisado como no supervisado y minería de datos en general [321, 322]. Un área que ha presentado un crecimiento acelerado en los últimos años es el aprendizaje profundo, con casos de sonado éxito en el recocimiento de objetos en imágenes [242, 324]. Un trabajo reciente [414] explora el uso de cómputo evolutivo en la construcción de redes neuronales profundas. Dado el éxito de la aplicación de estas redes y su rápida expansión a otros dominios del conocimiento podría ser interesante explorar el uso de optimización evolutiva multi-objetivo para el diseño de la arquitectura y el entrenamiento de estas redes.

Biología Computacional. Existe un gran número de problemas en esta área del conocimiento que pueden abordarse con técnicas de computación evolutiva multi-objetivo. Un problema central es el diseño computacional de proteínas que tendrá una repercución sin precedentes en el diseño de fármacos y la bionanotecnología, el principal cuello de botella lo constituyen las funciones de energía que no modelan correctamente el problema. Un camino posible es generalizar el trabajo pionero de Widera et al. [491] y extenderlo para permitir una mayor libertad de los términos que componen la función y hacerlo desde una perspectiva MO. Para este propósito la programación genética MO se presenta como el mejor candidato. Otra tarea pendiente del área es el modelado del acoplamiento molecular como un problema MO. Un primer intento de esto fue propuesto por García et al. [175], quienes proponen varios caminos posibles por donde avanzar en el tema. A la fecha hay evidencias de que los algoritmos evolutivos han sido efectivos en el diseño de fármacos desde la perspectiva

de pequeñas moléculas [120], donde también enfoques MO han sido explorados [330].

Multiobjetivización. Hay evidencias que muestran que multi-objetivizar problemas mono-objetivo ayudan a resolverlos de manera más sencilla, esto se ha mostrado para el problema de predicción de estructuras [201, 200], el problema de calendarización en ambiente *Job-Shop* [226] y ruteo de vehículos [487]. Un ejemplo adicional de esto es lo reportado en el trabajo de Knowles [236] donde se muestra que agregar un objetivo puede ayudar a reducir el número de óptimos locales para el ampliamente conocido problema del viajante. Por otro lado, es claro que problemas combinatorios que en su versión mono-objetivo pertenecen a la clase P, en su versión multi-objetivo pertenecen a la clase NP-difícil. Un ejemplo ilustrativo de este caso es el problema del camino más corto entre dos vértices dados en un grafo [412]. Sería entonces interesante poder caracterizar la clase de problemas, o incluso, casos de un problema que disminuirán el número de óptimos locales al volverlos un problema multi-objetivo. Existen estudios teóricos para el conteo de óptimos locales como los realizados por Hernando y colaboradores [208]. Se puede realizar un estudio similar estimando el número de óptimos locales para los casos en los que se agreguen funciones objetivo adicionales.

6.8 Para saber más

Este capítulo contiene sólo una breve introducción a la optimización evolutiva multi-objetivo con lo que consideramos básico y relevante

del tema. No obstante, existe una extensa fuente de información disponible que sugerimos a todo aquel lector que desee profundizar en el área.

6.8.1 Literatura selecta

Para saber más en cuanto a la optimización evolutiva multi-objetivo, se sugiere al lector los siguientes libros:

- **Multi-objective optimization using evolutionary algorithms** [112].
  Kalyanmoy Deb.

- **Applications of Multi-objective Evolutionary Algorithms** [83].
  Carlos A. Coello Coello & Gary B. Lamont.

- **Multiobjective Evolutionary Algorithms and Applications** [441].
  Kay Chen Tan, Eik Fun Khor & Tong Heng Lee.

- **Evolutionary algorithms for solving multi-objective problems** [90].
  Carlos A. Coello Coello, Gary B. Lamont & David A. Van Veldhuizen.

- **Multiobjective Optimization: Interactive and Evolutionary Approaches** [53].
  J. Branke, K. Deb, K. Miettinen, and R. Slowiński (Eds.).

Si desea conocer más, a través de un formato más compacto, se sugiere consultar el contenido de tutoriales especializados como el de Coello [87], Zitzler et al. [522], Konak et al. [237], Brockhoff y Wagner [61], y recientemente, Emmerich y Deutz [135]. Además, existe un repositorio Web, disponible en el sitio `http://emoo.cs.cinvestav.mx/`, donde se encuentra una colección de referencias de libros, artículos de revista, artículos de conferencia, tesis, software, entre otros.

6.8.2 Herramientas de software para la optimización evolutiva

Adicionalmente, existen plataformas de software que cuentan con los algoritmos y herramientas necesarias para experimentar directamente con algunas estrategias de optimización evolutivas mono y multi-objetivo. A continuación se presentan y se describen brevemente algunas de estas plataformas.

- JMetal [129]: es una plataforma orientada a objetos en Java para optimización multi-objetivo con metaheurísticas. Incluye algoritmos evolutivos mono y multi-objetivo, algunos de ellos paralelizados. Incluye además las funciones de prueba más utilizadas y algunos problemas combinatorios, además de indicadores de calidad.

- MOEA Framework: es una librería de código abierto en Java que ofrece la colección de algoritmos evolutivos para optimización mono y multi-objetivo más grande que hay. Incorpora algunos algoritmos de JMetal y de PISA.

- PISA [47]: Es una interfaz de texto para algoritmos de búsqueda. Esta herramienta divide el proceso de optimización en dos módulos, lo que da una mayor flexibilidad para implementar entre módulos. Por un lado, un módulo contiene las partes específicas del problema de optimización (evaluación de soluciones, representación y operadores de variación). Por el otro lado, el segundo módulo contiene las partes que son independientes del problema de optimización (básicamente, el mecanismo de selección).

- KEA: es una paquetería que ofrece una variedad de algoritmos evolutivos multi-objetivo. Permite definir las funciones objetivo tanto en Java como en C++.

- Guimoo: es un software libre dedicado al análisis de resultados de la optimización multi-objetivo. Entre sus características permite la visualización en tiempo real de la aproximación al frente Pareto.

Capítulo 7

Hiper-heurísticas en la Solución de Problemas de Optimización

7.1 Introducción

Cuando nos enfrentamos a problemas complejos, como aquellos del tipo NP o de optimización combinatoria, pueden presentarse dos escenarios diferentes. En el primero estamos dispuestos a invertir todos los recursos disponibles para resolver, de la mejor manera posible, un problema particular. Este escenario se caracteriza porque cada vez que un nuevo problema aparece, se invierte una gran cantidad de recursos para obtener una solución a esa instancia dada. En el segundo escenario, estamos dispuestos a sacrificar algo de la calidad de la solución para economizar recursos. Por lo tanto, se utilizan métodos genéricos

y capaces de obtener soluciones aceptables mediante un uso racionado de los recursos. En este caso, el riesgo es que el método genérico no sea adecuado para el problema y que produzca una solución de mala calidad. Está claro que, en promedio, el primer escenario generará mejores soluciones, pero no es viable en la mayoría de los problemas reales debido al alto consumo de recursos. Las hiper-heurísticas, por otra parte, son generalmente útiles en el segundo escenario, pues buscan proporcionar estrategias genéricas con buena calidad.

Las hiper-heurísticas son métodos emergentes de búsqueda heurística que intentan producir un nivel más alto de flexibilidad e independencia del dominio en que se aplican, de tal forma que sean capaces de adaptarse a distintos tipos de problemas o a las distintas instancias de un mismo tipo. Esto lo automatizan frecuentemente incorporando métodos de aprendizaje automático y lo ejecutan a través de un proceso de selección dinámica de heurísticas más simples durante la solución de los problemas, o bien, mediante la generación de nuevas heurísticas a partir de la combinación o adaptación de otras ya existentes (o de sus componentes). Actualmente hay diversos modelos de hiper-heurísticas, los cuales pueden clasificarse de acuerdo a las categorías propuestas por Burke et al. [65]. Esta clasificación divide a las hiper-heurísticas según su forma de aprendizaje, así como según su mecanismo de obtención de las soluciones, entre otros. Aunque el término 'hiper-heurística' fue acuñado como tal en el 2001 [100], hubieron esfuerzos previos con ideas similares. Entre ellos cabe destacar a los portafolios (o selección) de algoritmos [136, 351, 361], y a la configuración específica de algoritmos [282, 281]. Desde entonces, la actividad

de investigación en hiper-heurísticas ha avanzado y propagado en diversas instituciones en el mundo, y ha sido tema de presentación y discusión en diversas conferencias y revistas internacionales.

Por otra parte, y en lo que respecta a la investigación en hiper-heurísticas en México, son pocos los investigadores e instituciones que se han vinculado a ella. La mayor parte de la investigación realizada en el tema se concentró inicialmente en el Tecnológico de Monterrey, donde se han desarrollado hiper-heurísticas para corte y empacado [447, 449], satisfacción de restricciones [448, 345] y ruteo de vehículos [142]. El Tecnológico de Monterrey es también pionero en el tema de la transformación de características y en hiper-heurísticas para resolver problemas multi-objetivo. Las hiper-heurísticas desarrolladas en el Tecnológico de Monterrey se han fundamentado en diversas estrategias de aprendizaje, que incluyen algoritmos genéticos [345, 449], redes neuronales [343, 344] y programación genética [341, 426], entre otras. Recientemente, el Instituto Politécnico Nacional (IPN) y el CINVESTAV han logrado importantes avances en lo relacionado a hiper-heurísticas para problemas multi-objetivo [145] y en el uso de hiper-heurísticas basadas en evolución diferencial [360]. Como resultado de la colaboración del Tecnológico de Monterrey y el CINVESTAV, en el 2017 comenzó a explorarse la transformación de características como mecanismo para mejorar el desempeño de hiper-heurísticas [22]. El Instituto Tecnológico de León y la Universidad de Guanajuato ha colaborado para desarrollar múltiples trabajos sobre hiper-heurísticas. Los dominios de problemas que han abordado con hiper-heurísticas incluyen calendarización de eventos académi-

cos [421, 424, 423] y ruteo de vehículos [196]. Adicionalmente, la Universidad de Guanajuato ha incursionado en el uso de redes neuronales pre-entrenadas para acelerar el proceso de selección de las heurísticas [420] además de proponer una metodología para la adecuada selección del conjunto de heurísticas que debe estar disponible para una determinada hiper-heurística [422]. El Instituto Tecnológico de Ciudad Madero, en colaboración con el Instituto Tecnológico de León, ha realizado importantes avances en la generación semiautomática de heurísticas para el problema de empacado, utilizando diversos métodos como evolución diferencial [429] y evolución gramatical [430].

Este capítulo está encaminado a presentar el concepto de hiper-heurísticas para entender sus características, fundamentos, modelos, fortalezas y debilidades, así como algunas de sus áreas de aplicación. El capítulo integra en la Sección 7.2 los fundamentos de las hiper-heurísticas considerando principalmente dos tipos principales: aquéllas que seleccionan heurísticas y aquéllas que generan nuevas heurísticas. En la Sección 7.3 se presentan algunos modelos de hiper-heurísticas, de los cuales unos están basados en esquemas evolutivos y otros están basados en redes neuronales. La Sección 7.4 presenta un ejemplo con la idea de mostrar paso a paso el proceso para generar hiper-heurísticas, en particular para el problema de partición balanceada. En la Sección 7.5 se discuten algunos aspectos avanzados de las hiper-heurísticas, tales como la solución de problemas multi-objetivo y la transformación de características. En la Sección 7.6 se describe algunas tendencias futuras de hiper-heurísticas. En la Sección 7.7 se discuten algunos de los retos y perspectivas en esta área. Algunos re-

cursos disponibles relacionados con las hiper-heurísticas se presentan en la Sección 7.8 y, finalmente, en la Sección 7.9 se proporcionan las conclusiones.

7.2 Fundamentos de las Hiper-heurísticas

Las hiper-heurísticas trabajan sobre la idea de que es posible reutilizar los componentes de métodos existentes –generalmente simples, como son las heurísticas– para construir métodos de búsqueda más robustos y que se adapten a las condiciones del problema actual. De esta forma se busca obtener soluciones que, en la medida de lo posible y de acuerdo al tiempo disponible, sean de buena calidad. Ahora bien, la forma de adaptación a los problemas da lugar a los dos grandes tipos de hiper-heurísticas: las que seleccionan heurísticas y las que generan nuevas heurísticas.

Hiper-heurísticas que seleccionan heurísticas. En este tipo de adaptación se busca cambiar la heurística usada para atacar la instancia del problema; cambios que pueden ocurrir cada vez que el proceso requiere tomar una decisión. Por ejemplo, en el caso del problema del vendedor viajero (TSP), se pretende encontrar el tour de mínimo costo para visitar todas y cada una de las ciudades de un conjunto dado. Si se inicia con un tour vacío, la toma de decisiones consiste en decidir la siguiente ciudad a visitar. Mientras que con una heurística estaríamos a merced de una sola forma de decisión (por ejemplo, visitar la ciudad más cercana o la más lejana a la actual, que todavía no

se haya visitado), con una hiper-heurística tenemos la flexibilidad para utilizar heurísticas diferentes en cada toma de decisión (por ejemplo, iniciar con la más cercana y luego tomar la más lejana). De esta manera, es común que una hiper-heurística utilice conjuntos y órdenes de heurísticas distintos para resolver diferentes instancias de un problema. Desde esta perspectiva, una hiper-heurística es el método de alto nivel que controla cómo y cuándo usar heurísticas de bajo nivel.

Hiper-heurísticas que generan nuevas heurísticas. Este segundo tipo de adaptación busca combinar heurísticas existentes (o sus componentes) en la generación de heurísticas de alto nivel. Por ejemplo, asumiendo de nuevo el problema del vendedor viajero donde se construye la solución a partir de un tour vacío, una heurística podría decidir la nueva ciudad a agregar como aquella que minimice la distancia d_x con respecto al eje coordenado x a la última ciudad visitada en el tour parcial actual. Sin embargo, otra heurística podría seleccionar la ciudad basada en la distancia d_y con respecto al eje coordenado y. La ciudad recomendada por ambas heurísticas podría ser la misma o una distinta. Otra alternativa sería combinar los criterios de ambas heurísticas, por ejemplo, que se agregue la ciudad que minimice la función $f(x) = d_x \times d_y$. Mediante esta sencilla función, es posible combinar los criterios de decisión de cada heurística y formar una que se adapte a diferentes condiciones del problema. De esta forma, la hiper-heurística es un método que genera nuevas heurísticas.

A continuación describiremos con más detalle cada uno de estos tipos de hiper-heurísticas.

7.2.1 Hiper-heurísticas que seleccionan heurísticas

La mayor parte de los trabajos realizados en México se ha enfocado en este primer tipo de adaptación, y por ello optamos por detallarlas más. Las hiper-heurísticas de selección, como nos referiremos a ellas a partir de este momento, están inspiradas en la idea de que existe un algoritmo más adecuado para cada problema, lo que en la literatura se conoce como el problema de la selección de algoritmos [383]. En este sentido, las hiper-heurísticas de selección exploran la forma en que diversas heurísticas (u otros métodos simples y rápidos de ejecutar) pueden combinarse para generar estrategias de solución robustas y con un buen desempeño general.

En este punto cabe mencionar que las diferencias entre los portafolios de algoritmos [136, 351, 361], la configuración específica de algoritmos [282, 281] y las hiper-heurísticas de selección son sutiles, pero aún así pueden apreciarse. Por ejemplo, tanto las hiper-heurísticas como los portafolios de algoritmos deciden cuál de los métodos disponibles para resolver el problema debe utilizarse en un momento dado. La diferencia principal es la naturaleza de los métodos disponibles, donde los portafolios de algoritmos trabajan usualmente con colecciones de métodos de solución especializados para ciertos tipos de problemas, mientras que las hiper-heurísticas usualmente administran métodos más generales, como son las heurísticas. La configuración específica de algoritmos (ISAC, por sus siglas en inglés) busca optimizar los

parámetros de los algoritmos para que se adapten a las características del problema que intentan resolver. Si consideramos dos instancias de un mismo algoritmo con configuraciones distintas pueden considerarse como dos métodos diferentes, entonces ISAC es, en esencia, un portafolio de algoritmos.

Como ya se mencionó, una hiper-heurística de selección decide, en determinados momentos del proceso de solución, qué heurística debe utilizar. A partir de este momento usaremos el término 'punto de decisión' para referirnos a esos momentos específicos durante la búsqueda cuando la hiper-heurística puede cambiar la heurística a utilizar. Los puntos de decisión pueden manejarse de muchas formas. Por ejemplo, podría realizarse un cambio de heurística cada vez que una variable deba asignarse, o sólo después de cierto número de veces que deba asignarse, o sólo si se cumple alguna condición particular. Una forma muy común en la que se usan las hiper-heurísticas 'perturbativas' –aquellas que incrementalmente aplican modificaciones locales (movimientos) a soluciones completas (aproximaciones a la solución deseada) que representan los estados actuales de la búsqueda– es cambiar de heurística perturbativa únicamente si la calidad de la aproximación actual está disminuyendo o si se ha mantenido estable por cierto tiempo.

De forma muy general, una hiper-heurística de selección se comporta como se muestra en el Algoritmo 16.

Como puede observarse en el Algoritmo 16, la hiper-heurística tiene la capacidad de aplicar una heurística al problema para modificar su estado (buscando acercarlo a un estado meta o solución deseada). Ya que un problema específico de búsqueda depende del estado inicial de

Algoritmo 16 Estructura genérica de una hiper-heurística de selección.

Entrada: `heurísticas`, el conjunto de heurísticas a utilizar. `problema`, la instancia del problema a resolver.
Salida: `solución`, la solución de la instancia.

```
1:
2: mientras el problema no se haya resuelto hacer
3:     heurística ← Selecciona(heurísticas,problema)
4:     problema ← AplicaHeurística(heurística,problema)
5: fin mientras
6: devolver problema.solución
```

la misma, si consideramos a la solución resultante de aplicación de la heurística actual al estado actual como el estado inicial del resto de la búsqueda, estamos modificando el problema a resolver y este quizás podría ser resuelto de una mejor manera con una heurística distinta a la actual.

La elección de la heurística a utilizar en cada punto de decisión depende completamente de la función `Selecciona`, la cual busca en el espacio de heurísticas la más adecuada de acuerdo al estado actual del problema. Como puede inferirse del pseudocódigo, diferentes implementaciones de la función `Selecciona` resultarán en diferentes comportamientos de la hiper-heurística. Por ejemplo, una hiper-heurística completamente aleatoria implementaría la función `Selecciona` como se muestra en el Algoritmo 17.

Algoritmo 17 Selecciona una heurística de forma aleatoria.

```
1: función Selecciona(heurísticas,problema)
2: devolver escoge aleatoriamente un elemento de heurísticas
3: fin función
```

En el caso de una hiper-heurística aleatoria, la rutina ignora por completo el estado del problema, ya que no lo requiere para determinar qué heurística utilizar. Como es de esperarse, el desempeño de una hiper-heurística aleatoria es, en la mayoría de los casos, malo si el conjunto de heurísticas simples contiene una o más heurísticas cuyo comportamiento es deficiente en los problemas a resolver.

Otra posible idea para implementar una hiper-heurística es usar una secuencia predefinida de heurísticas como estrategia de decisión dentro de la función `Selecciona`. La idea detrás de esta hiper-heurística es que, en algunos casos, la relación entre heurísticas depende de su encadenamiento (una secuencia ordenada de heurísticas). Por ejemplo, en un problema de empacado, donde objetos de diferentes volúmenes deben ser empacados en un número finito de contenedores de forma que se minimice el número de contenedores utilizados, podría ser conveniente el utilizar una secuencia de heurísticas donde después de empacar el objeto más pequeño siempre se deba empacar el objeto más grande. Una hiper-heurística basada en secuencias predefinidas requiere conocer la secuencia específica de heurísticas a aplicar y la última heurística aplicada. El Algoritmo 18 muestra una posible implementación de este tipo de hiper-heurísticas.

Algoritmo 18 Selecciona una heurística basándose en una secuencia predefinida de heurísticas.

```
1: función Selecciona(heurísticas,problema)
2:     heurísticas ← última heurística aplicada en problema
3: devolver la que aparece después de heurística en heurísticas
4: fin función
```

En caso de que una heurística deba aplicarse dos o más veces seguidas, esta deberá aparecer dos o más veces en la secuencia predefinida de heurísticas. Debido a que la longitud de la secuencia de heurísticas es mucho menor que el número de puntos de decisión en el problema, hay que definir lo que se hará al llegar al final de la secuencia. Una alternativa es que se vuelva a recorrer la misma secuencia desde la primera posición. Otra opción es que al llegar al final de la secuencia, desde ese momento se use sólo la última heurística de la secuencia.

Una tercera versión de hiper-heurística (Algoritmo 19) representa la selección inteligente del método más adecuado dadas las condiciones actuales del problema. En esta versión de hiper-heurística, la función `Selecciona` contiene un mapeo de situaciones en las que cada heurística debería ser utilizada.

Algoritmo 19 Selecciona una heurística de acuerdo al estado actual del problema.

```
1: función Selecciona(heurísticas,problema)
2:     estado ← caracteriza el estado actual del problema
3: devolver escoge una heurística con base en estado
4: fin función
```

Como puede apreciarse en el Algoritmo 19, la función debe evaluar el estado actual del problema, para determinar qué heurística aplicar. Hasta este momento no hemos proporcionado detalles sobre cómo se realiza la selección de la heurística más adecuada para un determinado estado del problema, pero podemos adelantar que es el punto más sensible del proceso.

Entre los modelos más utilizados para producir hiper-heurísticas de selección se encuentran los algoritmos genéticos [345, 447, 449] y las redes neuronales [316, 344, 461]. Más adelante en este capítulo se describirán a detalle cómo funciona cada uno de estos modelos y cómo se implementan.

7.2.2 Hiper-heurísticas que generan nuevas heurísticas

Este segundo tipo de adaptación hace que las hiper-heurísticas funcionen como generadores semi-automáticos de heurísticas. De esta forma, la combinación de componentes de decisión se plasma en una función que, por sí misma, representa una nueva heurística. En el contexto de heurísticas perturbativas puede tratarse de la combinación de dos o más operadores al mismo tiempo, cada uno de ellos enfocados a modificar distintas secciones de la aproximación actual o bien, a la aplicación en secuencia de dos o más operadores perturbativos al estado actual de la búsqueda. La situación es diferente para hiper-heurísticas 'constructivas' –aquellas que gradualmente construyen la solución al problema generando soluciones parciales cada vez más completas a partir de una solución inicialmente vacía–, ya que, en este caso, las heurísticas constructivas tienen la tarea de asignar un valor a cada elemento que puede integrarse a la solución de forma que se agregue el elemento que maximice (o minimice, según sea el caso) dicho valor. Por ejemplo, si estamos utilizando un hiper-heurística constructiva para resolver una instancia del problema del agente viajero, podemos considerar una heurística que agrega la ciudad con la menor distancia

a la última ciudad visitada en la solución parcial actual. Para ello, se debe calcular la distancia de cada una de las ciudades que todavía no han sido visitadas, a la última ciudad del tour parcial actual. Luego de ello, la heurística recomendará agregar aquella con la menor distancia.

De forma muy general, una hiper-heurística de generación se comporta como se muestra en el Algoritmo 20.

Algoritmo 20 Estructura genérica de una hiper-heurística de generación.

```
Entrada: heurísticas, el conjunto de heurísticas a utilizar.
 1:              problema, la instancia del problema a resolver.
Salida: solución, la solución de la instancia.
 2:
 3: heurística ← Genera(heurísticas,problema)
 4: mientras el problema no se haya resuelto hacer
 5:     problema ← AplicaHeurística(heurística,problema)
 6: fin mientras
 7: devolver problema.solución
```

Por supuesto, en este momento no se brindan detalles de cómo implementar la función `Genera`, que es la responsable de generar la heurística que se utilizará para resolver el problema actual. Entre los métodos más populares en la comunidad para implementar hiper-heurísticas de generación se cuentan tanto programación genética [329, 495] como evolución diferencial [259, 477].

7.3 Modelos de Hiper-heurísticas

En esta sección se presentarán las ideas generales detrás de las hiper-heurísticas que se ha implementado de diversas formas y con apoyo

de diferentes meta-heurísticas o distintos modelos de aprendizaje automático.

7.3.1 Modelos Evolutivos

La computación evolutiva se ha utilizado tanto para construir hiper-heurísticas de generación (mediante programación genética y evolución diferencial) como para construir hiper-heurísticas de selección (mediante algoritmos genéticos y estrategias evolutivas). A continuación describiremos cómo se han implementado algunos casos de éxito de hiper-heurísticas mediante computación evolutiva.

7.3.1.1 Hiper-Heurísticas basadas en Algoritmos Genéticos

Los modelos evolutivos generan reglas que relacionan puntos específicos del espacio de características del problema con una heurística específica, que ha demostrado ser útil ante instancias con características similares. Estas reglas pueden obtenerse de diversas formas – incluso pueden ser definidas por el usuario. Sin embargo, los algoritmos genéticos han resultado muy útiles para esta tarea, y por esta razón su uso será detallado a continuación.

El algoritmo genético mantiene una población de individuos de longitud variable que representan hiper-heurísticas en potencia. Durante cada generación, estos individuos son combinados y mutados con la finalidad de mejorar la calidad de la población. Según sea el caso, los operadores genéticos pueden o no ser adaptados para el problema. Por ejemplo, como el número de reglas en las hiper-heurísticas es variable, los individuos en la población también tienen una longitud

variable. Es necesario, entonces, adaptar el operador de cruza para que produzca sólo individuos válidos. La función objetivo contiene un conjunto de instancias a resolver. El desempeño de cada individuo en dicho conjunto se convierte en su aptitud. Con el paso de las generaciones, se espera que los individuos que mejor resuelvan el conjunto de entrenamiento sean favorecidos por el proceso genético. El mejor individuo de la última generación es seleccionado como la hiper-heurística resultante del proceso evolutivo.

Los modelos hiper-heurísticos basados en algoritmos genéticos fueron unas de las primeras opciones que se consideraron para generar heurísticas. El primer trabajo que utilizó un algoritmo genético para la exploración del espacio de heurísticas en problemas de calendarización fue publicado en 1994 [146, 147], años antes de que el término hiper-heurística fuera utilizado por primera vez. Pocos años más tarde, esta idea fue retomada en [395] y [450]. A la fecha, gran parte de los trabajos relacionados con hiper-heurísticas utilizan alguna forma de computación evolutiva como parte del mecanismo de generación y ajuste de las hiper-heurísticas [449, 345, 269].

7.3.1.2 Hiper-Heurísticas basadas en Programación Genética

Las hiper-heurísticas basadas en programación genética tienen la capacidad de combinar los componentes de heurísticas existentes para construir nuevas heurísticas que funcionen adecuadamente ante diversos grupos de instancias del problema. La inclusión de una gramática tipo BNF en este proceso es una tendencia reciente. Un problema de satisfacción de restricciones (CSP) se define como un conjunto de

variables X, asociado a un conjunto de dominios de valores (D), y a un conjunto de restricciones entre las variables. El objetivo es una asignación completa y consistente de valores a variables de la manera que se cumplan todas las restricciones. Los CSPs han sido aplicados para el modelado y solución de problemas en diversos dominios. Aquí describimos una plataforma basada en gramáticas para generar automáticamente nuevas heurísticas de selección de variables para CSPs. El espacio de posibilidades se define generando oraciones válidas de acuerdo a la gramática que se utiliza.

Un enfoque inicial en este sentido es el presentado en el trabajo de Ortiz-Bayliss et al. [340] que provee la evidencia de que es posible delegar, al menos para ciertas tareas, la generación de heurísticas a un proceso automático. En el trabajo se usan características provenientes de las heurísticas previamente diseñadas y se utiliza programación genética para combinar esos atributos y generar nuevas heurísticas. La idea es encapsular el conocimiento del experto humano de diversas formas para obtener mejores soluciones. Dado que en la gramática hay que definir un conjunto de funciones y un conjunto de terminales, las funciones utilizadas en esta investigación incluyen suma (+), resta (-), multiplicación (*) y división protegida (%) (si el divisor es cero no se ejecutar la operación y el resultado es 1). Con este conjunto simple de operadores es suficiente para generar heurísticas competitivas. La definición de la gramática se puede observar en la Figura 7.1.

En un esfuerzo más avanzado en este respecto, el modelo presentado en [426], presenta una gramática que contiene elementos más sofisticados como el estatuto `if-then-else`, operadores aritméticos

<S> $\rightarrow$ <E>
| <E> $\rightarrow$ <Op> <F> <Op>
| <Op> $\rightarrow$ (<E>) | <T>
| <F> $\rightarrow$ + | - | * | %
| <T> $\rightarrow$ $deg(x)$ | $conflicts(x)$ | $dom(x)$ | $\kappa(x)$ | r

Figura 7.1: Gramática propuesta.

y lógicos, lo que permite generar mayor cantidad de posibilidades de combinación, además de funciones sobrecargadas capaces de manejar diversas dimensionalidades de entrada, lo que provee mayor flexibilidad y un mayor espacio de soluciones. En este trabajo se usaron tres diferentes estrategias para evolucionar las heurísticas: programación genética (GP), búsqueda local iterada (ILS) y búsqueda de alpinista en paralelo (PHC). En los experimentos se usaron tanto instancias sintéticas como reales para comparar el rendimiento entre las heurísticas generadas y las del estado del arte diseñadas por el humano. El estudio considera varios asuntos como la composición del conjunto de entrenamiento hacia la generalidad de las heurísticas y su rendimiento sobre instancias no vistas. También se considera la evolución de heurísticas sobre instancias pequeñas capaces de generalizar sobre instancias más grandes y no vistas, y sobre otras instancias del mundo real. La Figura 7.2 muestra un ejemplo de heurística generada por programación genética.

7.3.2 Modelos Neuronales

A diferencia de los modelos evolutivos que generan reglas que mapean características del problema a heurísticas específicas, las hiper-

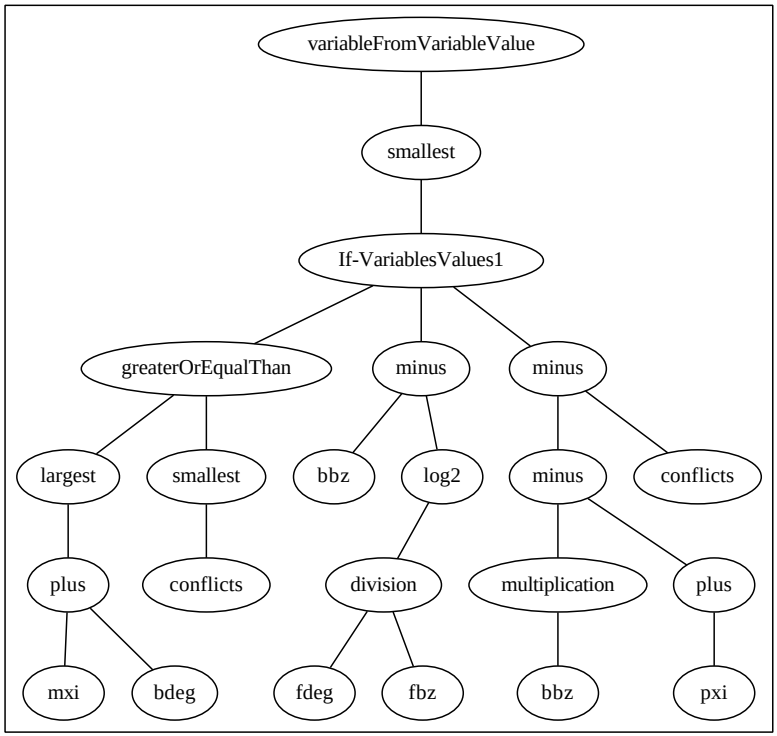


Figura 7.2: Ejemplo de heurística generada automáticamente por un proceso de Programación Genética para el problema de CSP.

heurísticas basadas en modelos neuronales aprenden las relaciones existentes entre características y heurísticas, a partir de casos de ejemplo. De esta manera, los modelos de hiper-heurísticas basados en redes neuronales requieren un conjunto de entrenamiento donde se contengan las relaciones a aprender. De manera general, la hiper-heurística contiene una red neuronal que aprende a clasificar instancias, según sus características. La clase a la que pertenece la instancia representa la heurística a utilizar en un momento particular de la búsqueda.

Probablemente uno de los primeros trabajos que exploraron el uso de redes neuronales para la selección de heurísticas fue el presentado por Moriarty [316], en donde se presentó el modelo SANE (Symbiotic, Adaptive Neuro-Evolution) para encontrar redes neuronales que representaban heurísticas para el ordenamiento de valor en problemas de satisfacción de restricciones. Años más tarde, Corr [99] propuso utilizar redes neuronales como selectores de heurísticas constructivas para el problema de la calendarización de exámenes. No fue, sino hasta el 2009, que se publicó el primer trabajo de redes neuronales que explícitamente mencionaba el término 'hiper-heurística'. En dicho trabajo, Ortiz-Bayliss et al. [348] retomaron la idea de usar redes neuronales como selectores de heurísticas constructivas, pero esta vez aplicado al problema de satisfacción de restricciones y con una red neuronal con aprendizaje mediante retro-propagación.

Los modelos neuronales también se han usado dentro de las hiper-heurísticas para agilizar el entrenamiento. Por ejemplo, Li et al. [256] utilizaron una red neuronal para estimar el costo de una función objetivo, de forma que se minimizara el número de evaluaciones reales durante el proceso de entrenamiento de la hiper-heurística.

En cuanto a la arquitectura de la red neuronal, se han utilizado diferentes opciones. Una parte significativa de los trabajos ha utilizado el perceptrón multicapa [343, 461]. Algunos otros trabajos han optado por arquitecturas más específicas a ciertos problemas, como son los mapas auto-organizados [99, 344] y redes neuronales con retraso temporal [460]. En algunas ocasiones se ha explorado la idea de combinar redes neuronales con algoritmos genéticos para ajus-

tar la topología de la red y maximizar el desempeño de las hiper-heurísticas [346, 347, 348].

7.4 Un ejemplo

A continuación presentamos un breve ejemplo, donde mostramos la forma en la que las hiper-heurísticas pueden ser de utilidad, y el proceso para aplicarlas.

7.4.1 El problema

Para este ejemplo tomaremos el problema de partición balanceada (Balanced Partition), que consiste en encontrar la distribución de elementos que minimiza a Q, donde Q es el índice de calidad dado por la ecuación 7.1, y donde $item_x^y$ representa el elemento x del subconjunto y, y N_y es el número de elementos en el subconjunto y. Así, el mínimo de Q corresponde al valor óptimo de la separación.

$$Q = \left| \sum_{i=1}^{N_1} item_i^1 - \sum_{j=1}^{N_2} item_j^2 \right| \tag{7.1}$$

7.4.2 Las alternativas

Para atacar este sencillo problema podemos considerar un enfoque constructivo a través de dos heurísticas muy simples: seleccionar el elemento de carga máxima (MAX) o mínima (MIN). Partiremos de una solución vacía, al considerar que nuestra solución es el subconjunto 2 (por tanto toda la carga será asignada al subconjunto 1), por lo que

la solución se construirá al escoger los elementos que deben moverse al subconjunto 2 para balancear la carga de todo el sistema. Adicionalmente, nuestra solución estará completa cuando el subconjunto 2 represente a la mitad o más de la carga total de la instancia del problema, pues nuestras heurísticas no permiten retornar elementos.

Es importante resaltar que estas heurísticas no son necesariamente buenas ni malas, pues dependen mucho de la instancia del problema en la que se apliquen. Consideremos, por ejemplo, un comportamiento como el de la Tabla 7.1, donde observamos que la heurística MAX se desempeñó ligeramente mejor que la heurística MIN. Esto se debe a que en algunas instancias la heurística MAX se desempeñó mucho mejor que la heurística MIN (por ejemplo, en las instancias 7 y 10), y viceversa (por ejemplo, en las instancias 3 y 9). Es precisamente esta diferencia en desempeño lo que motivó la idea de las hiper-heurísticas, y donde se evidencia su eventual beneficio. De hecho, la Tabla 7.1 incluye una columna con el desempeño de una hiper-heurística simple (que detallaremos más adelante), donde se puede observar que es mucho mejor que sus contrincantes.

7.4.3 La información del problema

Ya conocemos el tipo de problema a resolver y las opciones que tenemos para hacerlo. Ahora bien, requerimos también alguna manera de obtener información de cada instancia en particular, para así poder analizarla y determinar qué alternativa tomar (MAX o MIN). Para ello, debemos definir al menos una característica del problema, relacionada principalmente a las condiciones actuales de la instancia (de

Tabla 7.1: Desempeño de las heurísticas (y de una hiper-heurística) sobre 10 instancias de mayor extensión.

Instancia	Q_{HH}	Q_{MAX}	Q_{MIN}
1	11	9	7
2	2	8	6
3	1	13	3
4	2	0	0
5	3	5	13
6	1	9	3
7	3	1	13
8	9	3	9
9	9	11	3
10	5	1	11
Total	**46**	**60**	**68**
Promedio	**4.6**	**6**	**6.8**

manera que se actualice a medida que progresa la solución). En este ejemplo utilizaremos la relación entre la carga actual del subconjunto 2, y la carga máxima, como se muestra en la ecuación 7.2. Con esto ya tenemos todo lo necesario para empezar a crear nuestras hiper-heurísticas.

$$\frac{\sum_{j=1}^{N_2} item_j^2}{\sum_{i=1}^{N_1} item_i^1 + \sum_{j=1}^{N_2} item_j^2} \in [0,1] \tag{7.2}$$

7.4.4 Nuestra primer hiper-heurística

Nuestro objetivo con la hiper-heurística será el de tratar de dar un orden lógico al proceso. Así, buscaremos pasar los elementos más grandes al inicio de la solución (es decir, cuando el subconjunto está

vacío), para luego refinarla con elementos más pequeños (para ajustar las pequeñas diferencias). Por tanto, podemos crear una hiper-heurística de dos reglas, como se muestra en la Figura 7.3. En la parte derecha de la figura podemos observar la zona de influencia de cada regla. Así, siempre que el valor de la característica (definida en la ecuación 7.2) sea inferior a 0.25, se usará la heurística MAX para seleccionar el siguiente elemento. Pero, si es superior, se usará la heurística MIN.

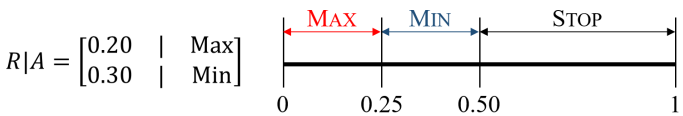


Figura 7.3: Primer modelo de hiper-heurística con dos reglas y una sola característica (izquierda). Zona de influencia de cada regla (derecha)

En este punto es bueno recordar la columna que traíamos pendiente de la Tabla 7.1. Es justamente esta hiper-heurística la que se usó para obtener los desempeños allí mostrados, donde una vez más se observa que supera a las heurísticas simples. Sin embargo, es importante resaltar que no todo es perfecto. En algunos casos, por ejemplo en la instancia 1, la hiper-heurística generó una solución peor que cualquiera de las dos heurísticas. Es más, en casos como el de la instancia 4, fue la única que no logró generar un balanceo perfecto de la carga. Esto se debe, principalmente, a la simplicidad de nuestra hiper-heurística, pues solamente cuenta con dos reglas y con una sola característica. Con ello se llega a que el espacio de características se divida en regiones simples, como la mostrada en la parte derecha de la Figura 7.3. Por otra parte, el desempeño de la hiper-heurística que estamos usando se encuentra limitado por su proceso de entrenamiento, pues los

valores fueron asignados manualmente y con base en lo que estimamos como mejor.

7.4.5 Otras hiper-heurísticas

Revisemos ahora cómo puede afectar el proceso de entrenamiento al desempeño de esta hiper-heurística en particular. Consideremos, por ejemplo, las hiper-heurísticas mostradas en la Figura 7.4. La primera de ellas corresponde a nuestro modelo original (el que hemos venido usando). Las otras dos tienen una estructura similar, pero el punto de corte de sus regiones de influencia se encuentra en 0.15 y 0.35, respectivamente. Como se muestra en la Tabla 7.2, el hecho de usar un enfoque más agresivo (es decir, HH3) sobre este conjunto de instancias resulta en un beneficio, pues se obtiene un valor de Q que, en promedio, es casi dos unidades menor al alcanzado con la hiper-heurística original (HH). Asimismo, el otro enfoque empeoró el resultado respecto al logrado con HH. Sin embargo, incluso así el resultado logrado es mejor que el obtenido por las heurísticas independientes.

$$\begin{aligned}
&\mathsf{HH:} && R|A = \begin{bmatrix} 0.20 & | & \text{Max} \\ 0.30 & | & \text{Min} \end{bmatrix} \\
&\mathsf{HH2:} && R|A = \begin{bmatrix} 0.10 & | & \text{Max} \\ 0.20 & | & \text{Min} \end{bmatrix} \\
&\mathsf{HH3:} && R|A = \begin{bmatrix} 0.30 & | & \text{Max} \\ 0.40 & | & \text{Min} \end{bmatrix}
\end{aligned}$$

Figura 7.4: Tres configuraciones diferentes de hiper-heurísticas.

Tabla 7.2: Desempeño de dos heurísticas simples y de tres modelos de hiper-heurísticas sobre un conjunto de 10 instancias.

Instancia	Q_{HH}	Q_{MAX}	Q_{MIN}	Q_{HH2}	Q_{HH3}
1	11	9	7	11	1
2	2	8	6	10	2
3	1	13	3	5	1
4	2	0	0	4	0
5	3	5	13	3	7
6	1	9	3	3	1
7	3	1	13	5	3
8	9	3	9	5	9
9	9	11	3	5	3
10	5	1	11	5	1
Total	**46**	**60**	**68**	**56**	**28**
Promedio	**4.6**	**6**	**6.8**	**5.6**	**2.8**

7.4.6 Las hiper-heurísticas como un problema de optimización

Las hiper-heurísticas mostradas han sido entrenadas 'a mano', definiendo ciertos valores críticos según nuestro criterio y conocimiento del problema. Sin embargo, el proceso se puede automatizar si se plantea como un problema de optimización. Para ello, podemos definir la función objetivo como el desempeño de la hiper-heurística. Esto se muestra en la ecuación 7.3, donde N_i es el número de instancias de entrenamiento y Q_j es el nivel de calidad dado por la hiper-heurística en la instancia j. Como se puede observar, el proceso de optimización implica probar cada hiper-heurística candidata sobre el conjunto de entrenamiento. Observemos también que en el caso más simple se

cuenta con cuatro variables de diseño, pero si aumentamos el número de características a, por ejemplo, tres, el número de incógnitas se duplica. Si en este punto agregamos un par de reglas más (para crear regiones de acción más complejas), el total de variables se duplica una vez más, llegando a 16. Así, el uso de técnicas de optimización clásicas como Newton-Rapshon o Descenso empinado queda descartado para la mayoría de problemas prácticos (debido a que se requiere un punto de inicio cerca de la solución y a que quedan atrapados en óptimos locales).

$$F_{obj}(F_T, A) = \frac{\sum_{j=1}^{N_i} Q_j}{N_i}; F_T = [F_1, F_2, F_3, ..., F_{N_F}] \wedge Q_j = Q_j(F_T, A) \tag{7.3}$$

Sin embargo, existe otra alternativa: las metaheurísticas. Para evitar desviar la atención del lector, omitiremos los detalles relacionados con el optimizador, y nos centraremos en el proceso de entrenamiento.

Vamos a imaginar que tenemos una herramienta capaz de construir una solución mejorada del problema, usando como base una solución existente que no es necesariamente buena. Imaginemos también que dicha solución corresponde a una hiper-heurística. Por tanto, podemos definir una solución (e.g. una hiper-heurística) candidata inicial (generada de forma aleatoria), y utilizar dicha herramienta para crear una refinada. Para determinar la ganancia dada por nuestra herramienta, basta con evaluar ambas soluciones (tanto la aleatoria como la refinada) en la ecuación 7.3, y comparar los resultados. Luego de esto, podemos tomar la solución refinada como base y usar nuestra herramienta para crear una mejorada. Es justamente este proceso iterativo lo que nos permite operar sobre la solución aleatoria hasta

lograr un resultado con la calidad deseada. A manera de ejemplo, consideremos un conjunto de 30 instancias aleatorias del problema de partición balanceada (las cuales no se muestran por motivos de espacio). Si generamos 10 soluciones aleatorias (como las descritas anteriormente), obtendremos hiper-heurísticas con desempeños buenos y malos, como se muestra en la Figura 7.5 (iteración 1). Sin embargo, podemos usar nuestra herramienta de optimización sobre cada una de estas hiper-heurísticas para generar unas mejoradas (iteración 2). Es más, podemos continuar iterativamente y, luego de 15 iteraciones, tendremos hiper-heurísticas cuyo desempeño es bueno y muy similar (iteración 16). Note que por la naturaleza estocástica de nuestra herramienta, en ocasiones el desempeño mejora muy rápidamente, pero en otros casos tarda varias iteraciones en mejorar. Además, tengamos en cuenta que muchas metaheurísticas trabajan de forma poblacional, es decir evalúan y mejoran múltiples soluciones al mismo tiempo con el ánimo de encontrar una única que sea lo suficientemente buena. Sin embargo, hemos omitido este tipo de detalles en la gráfica para evitar complicar nuestro análisis y para facilitar la comprensión por parte del lector.

7.5 Aspectos Avanzados

En esta sección se abordan algunos temas avanzados de las hiper-heurísticas, como son sus aplicaciones en problemas multi-objetivo y a la ciencia de datos, los modelos híbridos y algunas tendencias futuras del área.

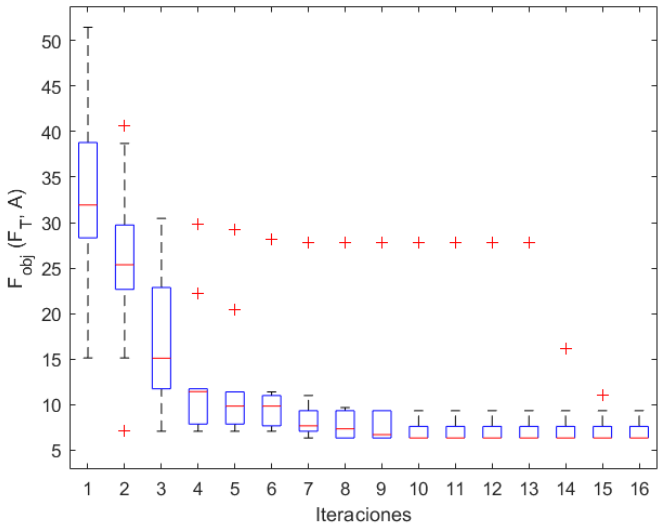


Figura 7.5: Ejemplo de entrenamiento de hiper-heurísticas usando metaheurísticas. La iteración 1 representa el resultado luego de generar la población inicial. Los datos mostrados corresponden a 10 réplicas sobre 30 problemas generados de forma aleatoria, con 100 elementos y una carga máxima de 100 por elemento.

7.5.1 Problemas Multi-objetivo

Muchos trabajos en la literatura relacionados con hiper-heurísticas para solucionar problemas de optimización, consideran solamente un objetivo que guía la búsqueda de soluciones. Sin embargo, es natural pensar que en estos problemas existen varios objetivos en conflicto y que la idea es optimizarlos al mismo tiempo.

Esto debe plantearse como un problema multi-objetivo y por lo tanto, la búsqueda de soluciones se debe hacer con un esquema adecuado que facilite ese proceso. Para resolver un problema con esa perspectiva es importante definir el problema, sus objetivos y el dis-

eño del algoritmo que genere la solución esperada, aunque en realidad lo que un algoritmo de este tipo produce es una serie de soluciones que corresponden a los mejores compromisos posibles entre los objetivos en conflicto. A estas soluciones se les denomina conjunto de óptimos de Pareto y sus valores correspondientes de las funciones objetivo conforman el denominado frente de Pareto. Para ejemplificar este tema, usaremos un problema de corte y empacado en dos dimensiones, donde se quiere empacar pequeñas figuras de diferentes tipos y formas, en hojas de material rectangulares. Lo que se requiere es minimizar el número de hojas de materia prima, pero al mismo tiempo minimizar también el tiempo en que se ejecuta el proceso. Estos son claramente dos objetivos que se contraponen: por un lado, se pueden poner las piezas muy rápidamente pero esto no será efectivo en cuanto al número de hojas de material; y por el otro lado, si queremos reducir el número de hojas de material, quizá requiera más tiempo para terminar el proceso. En [269] se usó un modelo evolutivo simplificado para representar el estado del problema donde el cromosoma representa un conjunto de puntos en el espacio y cada punto es etiquetado con una heurística. Bajo este esquema, el modelo representa una receta para resolver el problema que se puede describir como (a) determinar el estado actual E del problema (b) encontrar el punto más cercano a él, (c) aplicar la heurística anexa al punto, (d) actualizar el estado. Usamos esas ideas y en combinación con un algoritmo evolutivo multi-objetivo (AEMO) (NSGA-II, SPEA2, GDE3) se busca encontrar el conjunto S de cromosomas en el frente de Pareto que son capaces de encontrar soluciones a una amplia variedad de problemas, tomando en cuenta

los dos objetivos. El conjunto S contiene las hiper-heurísticas que se están buscando.

En la Figura 7.6 se muestra el esquema utilizado para resolver problemas multi-objetivo para el problema de corte y empacado.

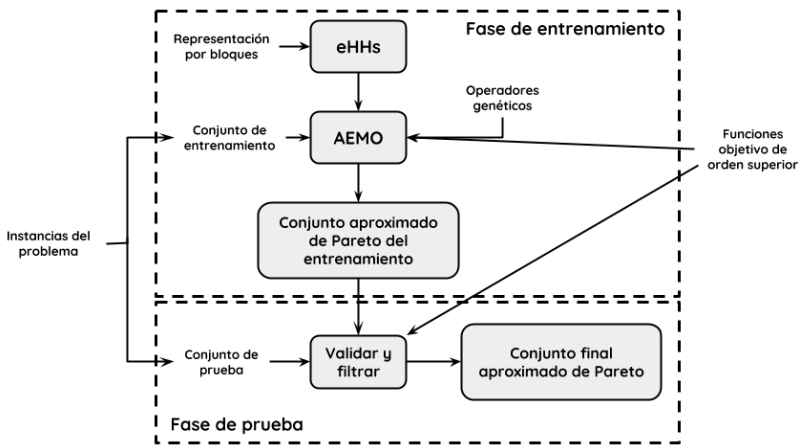


Figura 7.6: Modelo general multi-objetivo mostrando las fases de entrenamiento y prueba.

En la determinación del estado actual del problema E es importante caracterizarlo. Para este caso, fueron seleccionadas 10 características de un conjunto mayor y que son las siguientes:

1. Número de piezas restantes por empacar.

2. Área promedio de las piezas restantes.

3. Varianza en el área de las piezas restantes.

4. Media de la rectangularidad de las piezas restantes.

5. Varianza en la rectangularidad de las piezas restantes.

6. Media de la altura de las piezas restantes.

7. Varianza en el ancho de las piezas restantes.

8. Fracción de las piezas restantes cuya área es mayor a $\frac{1}{2}$ del área del objeto.

9. Media del grado de concavidad de las piezas restantes.

10. Fracción del total de las piezas restantes.

En la plataforma propuesta, para encontrar el conjunto S de mejores hiper-heurísticas en el frente de Pareto, se divide el conjunto de instancias en dos tipos: entrenamiento y prueba. El modelo evolucionaba poblaciones de hiper-heurísticas tomando en cuenta el AEMO. La fase de entrenamiento inicia con la generación de hiper-heurísticas en forma aleatoria para el AEMO. Este algoritmos evoluciona la población de acuerdo a su propia metodología y aplica operadores genéticos para generar nuevas poblaciones, considerando el conjunto de entrenamiento para evaluar su rendimiento.

Una vez que las mejores hiper-heurísticas se tienen, se prueban entonces con las instancias de prueba para comparar los modelos multi-objetivo, entre ellos y con respecto a las heurísticas simples adaptadas para el problema, usando diferentes métricas.

Los resultados obtenidos indican que los modelos eHH-MO son adecuados para construir un conjunto de reglas que permiten resolver el problema bi-objetivo de diversas maneras. Los resultados de los modelos eHH-MO han sido comparados contra los obtenidos con las

heurísticas simples, bajo una serie de experimentos usando tres diferentes métricas como son la distancia generacional (GD) [118, 470], el hipervolumen (HV) [525] y D [118] (dispersión y distribución). Desde diferentes perspectivas los modelos multi-objetivo superan claramente a los modelos de heurísticas simples. Los detalles de la serie de experimentos realizados y su discusión general pueden consultarse en [185].

7.5.2 Transformación de Características en Hiper-heurísticas

Una técnica que se ha implementado recientemente se enfoca en el uso de transformaciones matemáticas para mejorar el desempeño del modelo de selección. Al trabajar con una heurística de selección, las características juegan un papel preponderante en la eficiencia del proceso, pues el selector representa una matriz de puntos clave en el espacio de características. Además, debemos tener presente que cada problema de optimización es diferente, por lo que los puntos críticos de una característica difícilmente corresponderán con los de otra. Así, se inició un proceso para identificar la forma en la que se pudiera establecer una transformación adecuada a cada problema particular. Como resultado, se encontró que el problema principal que puede resolver la transformación es el mostrado en la Figura 7.7. Allí, se muestra la ubicación ideal de cuatro reglas (marcadores redondos) y su distancia al estado actual del problema (cuadro blanco). Dos de estas reglas tienen posiciones muy similares (la roja y la azul), por lo que el desempeño del selector se torna muy sensible a pequeños errores en la ubicación real de dichas reglas.

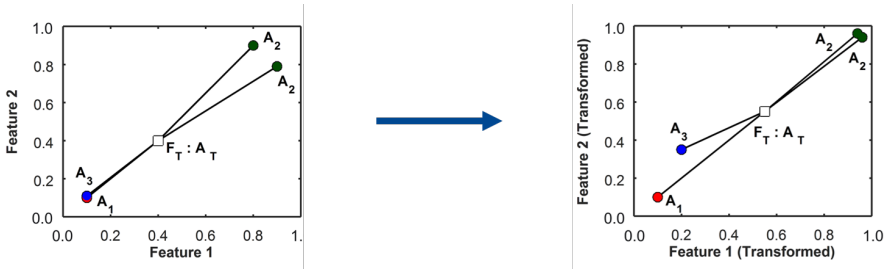


Figura 7.7: Sensibilidad de hiper-heurísticas a errores en los valores del selector y su solución a través de transformación de características. Izquierda: Ubicación ideal de las reglas en el espacio original. Derecha: Ubicación ideal de las reglas en el espacio transformado.

La Figura 7.7 también muestra la forma como se puede solucionar este problema con la transformación de características. La idea es transformar el espacio de cada característica del problema, de tal forma que las regiones en conflicto se expandan y las relativamente irrelevantes se compriman. Para el ejemplo de la figura, esto implica tener mayor resolución en el espacio cercano a las reglas azul y roja, lo que permite un mayor margen de maniobra y por tanto reduce la sensibilidad de la hiper-heurística. De forma similar, la zona que pertenece a las dos reglas verdes, se comprime. Dado que se trata de reglas que ejecutan la misma acción, su región de influencia combinada no será tan sensible a pequeños errores de ubicación. Consideremos ahora los casos mostrados en la Figura 7.8. En la izquierda se muestran dos selectores (uno con dos reglas y otro con cuatro), ambos con dos acciones disponibles: roja y azul. Al hacer una transformación de características podemos ampliar zonas en conflicto (como ya se mencionó arriba), lo que representa una distribución de las reglas la cual puede, incluso, modificar la apariencia de las zonas de influencia

de cada selector. Para nuestro ejemplo, dicha transformación se ve en las imágenes del centro, donde la apariencia ha sido parcialmente preservada, pero donde se da mayor enfoque a las zonas conflictivas. Claro está, este proceso no puede ser arbitrario, pues podemos también generar transformaciones poco útiles como las de la derecha de la figura, donde hemos perdido la mayoría de la información acerca de las zonas conflictivas, por lo que seguramente tendremos un bajo desempeño de la hiper-heurística. Detalles de esta investigación puden ser revisados en [22].

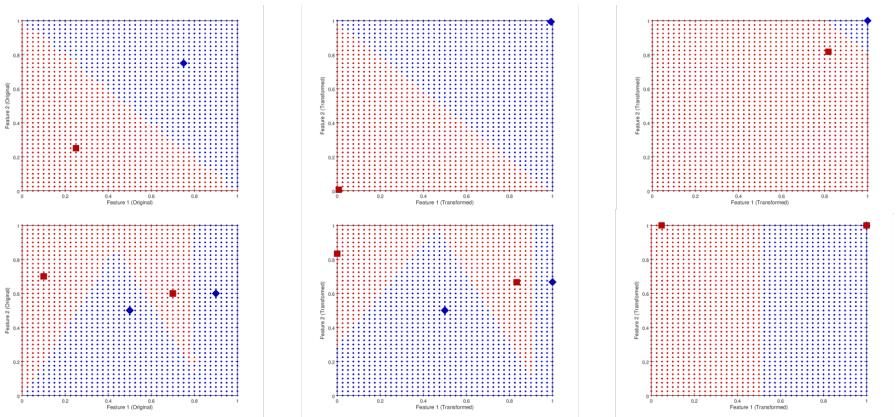


Figura 7.8: Ejemplo de la forma como la transformación de características puede afectar la apariencia de las zonas de influencia de un selector. Izquierda: Forma original. Centro: Transformación útil que amplía las zonas en conflicto. Derecha: Transformación poco útil que destruye información sobre las zonas en conflicto.

7.6 Tendencias Futuras

Los sistemas inteligentes híbridos [305] son sistemas de software que emplean, en paralelo, una combinación de métodos y técnicas de sub-

campos de la Inteligencia Artificial. La hibridación de los sistemas inteligentes es un campo de investigación prometedora de la Inteligencia Computacional moderna para el desarrollo de la próxima generación de sistemas inteligentes. Comunidades científicas y académicas están cada vez más conscientes de que enfoques combinados podrían ser necesarios si se quiere seguir resolviendo los problemas difíciles de la Inteligencia Artificial. La integración de distintas técnicas inteligentes y su adaptación para superar las limitaciones de los métodos individuales y lograr efectos sinérgicos a través de la hibridación o fusión de estas técnicas, ha contribuido a una variedad de nuevos diseños dentro del campo de la optimización [256, 204]. Sin embargo, la mayoría de estos enfoques de hibridación siguen un diseño metodológico ad-hoc que se justifica por el éxito en ciertos dominios de aplicación. No obstante, se requiere de metodologías más robustas para la construcción de los sistemas híbridos requeridos para resolver problemas contemporáneos y futuros cada vez más complejos y más grandes.

Las hiper-heurísticas [65], como técnicas de búsqueda que tratan de automatizar el proceso de generación, selección y combinación de otras heurísticas, son un excelente vehículo para la construcción de sistemas inteligentes híbridos avanzados. Recientemente han aparecido los primeros trabajos que proponen formas muy simples y limitadas de combinación de hiper-heurísticas [417]. Se requiere de estudios más profundos y serios que combinen generación con selección, así como estrategias constructivas con perturbativas [65], en formas más elaboradas que mejoren los resultados obtenidos.

El desafío de la hibridación hiper-heurística es enorme y complejo. Reside primeramente en el modelado de los problemas y de las soluciones (parciales o aproximadas) de forma que, a partir de descripciones paramétricas o no paramétricas, el método híbrido pueda ir decidiendo la combinación de métodos heurísticos que irá utilizando para guiar la búsqueda de la solución. Parte importante del desafío también es la selección inteligente de las heurísticas específicas a los dominios de los problemas [422], las meta-heurísticas y las hiper-heurísticas [439] que formarán el método híbrido. Otro elemento de complejidad es que cada técnica heurística incluye parámetros que deben ser ajustados adecuadamente para que trabajen de forma correcta y eso podría ser requerido durante la ejecución del método. Finalmente, como se quieren métodos híbridos que maximicen su desempeño, también se deberán seleccionar o diseñar las métricas para medir la efectividad y eficiencia de los algoritmos utilizados.

Son demasiadas posibilidades las que hay que considerar y demasiada información relacionada tanto con las instancias de los problemas a resolver, los métodos de solución, sus hiper-parámetros, su hibridación y el efecto de sus decisiones durante la solución de los problemas. Los investigadores ya están abordando algunos esfuerzos en estas direcciones a través del aprendizaje automático [81] y el uso de técnicas estadísticas, pero mucha de la práctica es primordialmente artesanal. Se requiere el uso de metodologías científicas más maduras y desarrolladas para manejar la gran cantidad de datos que se genera en el proceso.

Dado que la Ciencia de Datos es la capacidad de extraer conocimientos e ideas de grandes cantidades de datos y datos complejos, la construcción de métodos híbridos para optimización mediante hiper-heurísticas son un buen motivo para su aplicación.

La Ciencia de Datos intenta comprender las complejidades de las actividades del mundo real y el contexto a partir del análisis de los datos para mejorar los beneficios que pueden obtenerse de ellos. Los algoritmos de optimización en tiempo real están estrechamente vinculados al contexto de sus datos. Luego, utilizar las técnicas de Ciencia de Datos permitirá aprovechar el conjunto histórico de datos, así como los datos generados en tiempo real, durante la búsqueda de soluciones para los problemas de optimización [358]. El proceso de Ciencia de Datos se puede aplicar al ajuste automático de los algoritmos, su configuración y su adaptación para crear mejores métodos de optimización.

El desarrollo de algoritmos de búsqueda poderosos se asemeja a la solución de los problemas de optimización. La hibridación de optimizadores mediante hiper-heurísticas incluye una gran cantidad de parámetros y generalmente ofrecen muchas opciones. El desafío para la Ciencia de Datos es más complejo debido a la naturaleza estocástica de los métodos de solución. Para explotar el poder real de estas técnicas y resolver los problemas de manera eficiente, el desarrollador debe seleccionar los mejores valores y tomar las decisiones correctas.

En la última década se han desarrollado varias técnicas para la optimización de parámetros, la construcción automática de algoritmos y la creación de hiper-heurísticas para generar o seleccionar algorit-

mos, pero la Ciencia de Datos tiene mucho que aportar para explotar al máximo el potencial de las técnicas novedosas de búsqueda existentes. No se debe de buscar un método ideal sino crear metodologías adecuadas a los diferentes dominios de problemas. Estas metodologías deben ser robustas de manera que permitan construir métodos poderosos a partir de la descripción de esquemas de recolección, procesamiento, exploración, visualización, aprendizaje y explotación de datos adecuados a los problemas específicos a resolver.

Aunque actualmente hay un boom en el desarrollo de aplicaciones de Ciencia de Datos en muchos dominios, y hay un gran esfuerzo de grupos de personas para desarrollar métodos de búsqueda para dominios específicos, la aplicación de la Ciencia de Datos en la construcción de métodos de optimización requiere habilidades inherentemente diferentes y hay muy pocos trabajos específicos sobre el tema. Iniciativas como 'Data Science meets Optimization' [108] promovida por grupos creados hace poco, como el grupo europeo EWG DSO, confirman la relevancia y el potencial del tema.

7.7 Retos y perspectivas

Aunque las hiper-heurísticas han llegado un punto de madurez en años recientes y a ser identificada como un área de importancia en la comunicad científica, algunos temas faltan todavía por extender, y otros por explorar. Podemos mencionar algunos retos y perspectivas a futuro que permitirán consolidar el tema para tener una mayor contribución e impacto en la resolución de diversos problemas:

- Hibridación inteligente de sistemas que emplean en paralelo una combinación de métodos y técnicas provenientes de diversos campos de la IA. Ésto es un campo de investigación fertil para la generación de técnicas modernas de la Inteligencia Computacional sirva para el desarrollo de herramientas y modelos de nueva generación. La comunidad científica está consciente de que enfoques combinados podrían ser necesarios si se quiere seguir resolviendo los problemas complejos y retadores de la Inteligencia Artificial. La integración de distintas técnicas inteligentes y su adaptación para superar las limitaciones de los métodos individuales y lograr efectos sinérgicos a través de la hibridación o fusión de estas técnicas, ha contribuido a una variedad de nuevos diseños dentro del campo de la optimización [204, 256]. En consecuencia, se requiere de metodologías más robustas para la construcción de los sistemas híbridos requeridos para resolver problemas contemporáneos y futuros cada vez más complejos y de mayor tamaño.

- Las hiper-heurísticas [65], como técnicas de búsqueda que tratan de automatizar el proceso de generación, selección y combinación de otras heurísticas, son un excelente vehículo para la construcción de sistemas inteligentes híbridos avanzados. Particularmente se requieren estudios más profundos y serios para generar sistemas híbridos que combinen modelos de selección con generación, modelos constructivos y perturbativos, mecanismos que combinen aprendizaje *offline* con *online*, entre otros, que mejoren el comportamiento de los algoritmos y sus resultados.

- La Ciencia de Datos intenta comprender las complejidades de las actividades del mundo real y el contexto a partir del análisis de los datos para mejorar los beneficios que pueden obtenerse de ellos. Los algoritmos de optimización en tiempo real están estrechamente vinculados al contexto de sus datos. Por ello, utilizar las técnicas de Ciencia de Datos permitiría aprovechar el conjunto histórico de datos, así como los datos generados en tiempo real, durante la búsqueda de soluciones para los problemas de optimización [358]. El proceso de Ciencia de Datos se puede aplicar al ajuste automático de los algoritmos, su configuración y su adaptación para crear mejores métodos de optimización.

- A la fecha, las hiper-heuristicas han generado soluciones muy alentadoras para deiversos dominios y problemas, Sin embargo, su proceso ha sido más bien artesanal en la generación y adaptación de este tipo de algoritmos para la solución de problemas particulares. Es necesario en el futuro desarrollar procesos más sistemáticos y metodológicos que permitan resolver de manera robusta problemas de diversa índole, no solo de optmimización, y que puedan producir reglas complejas automatizadas para la aplicaciópn de cierto tipo de métodos a ciertos tipos de problemas. Es seguro que hiper-heurísticas puede aplicarse a diversos dominios, no necesariamente ligados a optimización.

Adicionalmente a los tópicos de investigación y retos en temas de hiper-heuristicas presentados en el párrafo anterior, seguro que hay muchas otras oportunidades que abren diversas líneas de investigación

en el tema. Cada vez hay mayor número de grupos e investigadores en múltiples instituciones que han establecido el tópico dentro de sus intereses y portafoilio de investigación. Pillay and Qu recientemente han publicado un libro sobre hiper-heurísticas [365] que resume lo realizado a la fecha en el tópico, y discuten perspectivas de investigación al futuro y sus aplicaciones.

7.8 Para saber más

El número de trabajos relacionados a hiper-heurísticas se ha incrementado significativamente en la última década. Como resultado, una gran cantidad de recursos se encuentran disponibles en diversas fuentes. El lector interesado en conocer a más detalle los diferentes tipos de hiper-heurísticas y sus aplicaciones puede consultar estudios de referencia como [65] y [350]. Una lista muy completa sobre los trabajos relacionados a hiper-heurísticas se encuentra disponible en línea, en `https://mustafamisir.github.io/hh.html`.

Debido al interés que las hiper-heurísticas han despertado en los investigadores, diversas conferencias y talleres han dedicado espacios para este tipo de trabajos. Por ejemplo: *Genetic and Evolutionary Computation Conference* (GECCO), que desde el 2011 ha proporcionado espacios en sus categorías *Self-* Search* (2011 al 2016) y *Search-Based Software Engineering* (a partir del 2017); *International Conference on Parallel Problem Solving from Nature* (PPSN), con talleres en el tema en las ediciones del 2008 y 2016; *IEEE Congress on Evolutionary Computation* (CEC), con sesiones especiales sobre

hiper-heurísticas en sus ediciones del 2013 y 2016. En cuanto a competencias internacionales sobre la materia, destacan las dos ediciones del *Cross-domain Heurisic Search Challenge* (CHeSC). La competencia consistía en diseñar estrategias de alto nivel que controlaran la aplicación de heurísticas de bajo nivel para dominios específicos. Durante el reto, las heurísticas de bajo nivel serían diferentes para cada dominio del problema pero la estrategia de alto nivel debía ser la misma para todos los dominios. El reto se enfocaba en el desempeño de los métodos de optimización en diferentes dominios de problemas, en lugar de concentrarse sólo en uno.

En cuanto a herramientas para la implementación de hiper-heurísticas, la literatura contiene pocos ejemplos, entre los que destacan Hyperion [438], HyFlex [331] y EvoHyp [364]. No todas las herramientas se encuentran disponibles en este momento, ya que algunos de los sitios para su distribución han sido descontinuados. En cualquier caso, es posible contactar a los autores para solicitar acceso a las herramientas.

Adicionalmente a estos recursos, en el 2013 se formó la *IEEE Task Force on Hyper-heuristics*[1], dedicada al desarrollo y aplicaciones de hiper-heurísticas y a su intersección con la inteligencia artificial, la inteligencia computacional y la investigación de operaciones. El sitio del grupo de trabajo proporciona muchos recursos útiles en el tema de hiper-heurísticas.

[1] https://sites.google.com/site/ieeetaskforceonhyperheurisitcs/home

7.9 Conclusiones

Este capítulo ha presentado el tema de hiper-heurísticas desde diferentes perspectivas con la idea de dar al lector una panorámica general, pero suficientemente detallada para que pueda conocer el tema, y pueda empezar a realizar avances a través de la investigación en el área o en la aplicación de las técnicas para resolver problemas en el dominio de optimización. El capítulo ha explicado el detalle los principales tipos de hiper-heurísticas y sus fundamentos, y los diversos modelos que se han generado en la literatura, principalmente basados en modelos evolutivos y neuronales, y entendiendo que no son los únicos. El capítulo provee discusión sobre aspectos avanzados en hiper-heurísticas y que se han derivado en trabajos recientes, como es el tratamiento de hiper-heurísticas en problemas multi-objetivo y la transformación de características que permite mejorar el rendimiento de esquemas de hiper-heurísticas. En trabajos recientes se han presentado resultados al hibridizar tipos de hiper-heurísticas, como por ejemplo las de selección con las de generación, las de construcción con las de generación, los tipos de aprendizaje, etc. Hay todavía un espacio para continuar investigando dentro de ese contexto. Con el reciente impacto del tema de *big-data*, se ha empezado a discutir en la comunidad científica la relación de la Ciencia de Datos con optimización. Aquí cabe naturalmente el tópico de hiper-heurísticas, dado que la Ciencia de Datos puede apoyar para generar mejores algoritmos de optimización, y ciertamente, los modelos de optimización pueden incidir favorablemente para mejorar los procesos en la Ciencia de Datos.

Al final, el capítulo presentó una serie de recursos que pueden servir como base a estudiantes e investigadores para perseguir el tema de hiper-heurísticas.

Bibliografía

[1] Niusvel Acosta-Mendoza, Alicia Morales-Reyes, Hugo Jair Escalante, and Andrés Gago-Alonso. Learning to Assemble Classifiers via Genetic Programming. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(7):1460005, September 2014.

[2] Arturo Hernández Aguirre and Carlos A. Coello Coello. Evolutionary Synthesis of Logic Circuits Using Information Theory. *Artificial Intelligence Review*, 20(3/4):445–471, 12 2003.

[3] E. Alba. Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance. *Information Processing Letters*, 82(1):7–13, April 2002.

[4] E. Alba and B. Dorronsoro. The Exploration/Exploitation Tradeoff in Dynamic Cellular Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.

[5] E. Alba, A. J. Nebro, and J. M. Troya. Heterogeneous Computing and Parallel Genetic Algorithms. *Journal of Parallel and Distributed Computing*, 62(9):1362–1385, September 2002.

[6] E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.

[7] E. Alba and J. M. Troya. Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms. *Statistics and Computing*, 12(2):91–114, April 2002.

[8] Enrique Alba. *Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos*. PhD thesis, Universidad de Málaga, Spain, March 1999. (in Spanish).

[9] Enrique Alba. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley & Sons, 2005. ISBN 978-047167806-9.

[10] Enrique Alba, Carlos Cotta, and José Ma Troya. Numerical and real time analysis of parallel distributed GAs with structured and panmictic populations. In *1999 IEEE Congress on Evolutionary Computation (CEC'1999)*, pages 1019–1026, Washington, DC, USA, July 6-9 1999. IEEE Press. ISBN 0-7803-5536-9.

[11] Enrique Alba and Bernabe Dorronsoro. *Cellular Genetic Algorithms*. Springer, USA, 2008. ISBN 978-0-387-77609-5.

[12] Enrique Alba, Francisco Luna, and Antonio Nebro. Advances in Parallel Heterogeneous Genetic Algorithms for Continuous Optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3):317–333, 2004.

[13] Enrique Alba and José M. Troya. A Survey of Parallel Distributed Genetic Algorithms. *Complexity*, 4(4):31–52, March/April 1999.

[14] Enrique Alba and José Ma Troya. An analysis of synchronous and asynchronous parallel distributed genetic algorithms with structured and panmictic islands. In José Rolim et al., editor, *Parallel and*

Distributed Processing, 11th IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, pages 248–256. Springer. Lecture Notes in Computer Science Vol. 1586, San Juan de Puerto Rico, USA, April 12-16 1999.

[15] Enrique Alba and José Ma Troya. Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 29–38, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[16] Isolina Alberto, Carlos A. Coello Coello, and Pedro M. Mateo. A comparative study of variation operators used for evolutionary multi-objective optimization. *Information Sciences*, 273:33–48, July 20 2014.

[17] María-Yaneli Ameca Alducin, Efrén Mezura-Montes, and Nicandro Cruz-Ramírez. Dynamic differential evolution with combined variants and a repair method to solve dynamic constrained optimization problems: an empirical study. *Soft Computing*, 22(2):541–570, 2018.

[18] Ismail M. Ali, Daryl Essam, and Kathryn Kasmarik. A novel differential evolution mapping technique for generic combinatorial optimization problems. *Applied Soft Computing*, 80:297–309, 2019.

[19] M. M. Ali and A. Törn. Population set-based global optimization algorithms: some modifications and numerical studies. *Computers & Operations Research*, 31(10):1703–1725, September 2004.

[20] Musrrat Ali, Patrick Siarry, and Millie Pant. An efficient Differential Evolution based algorithm for solving multi-objective optimization problems. *European Journal of Operational Research*, 217(2):404–416, March 1 2012.

[21] Richard Allmendinger, Michael T. M. Emmerich, Jussi Hakanen, Yaochu Jin, and Enrico Rigoni. Surrogate-assisted multicriteria optimization: Complexities, prospective solutions, and business case. *Journal of Multi-Criteria Decision Analysis*, 24(1-2):5–24, January-April 2017.

[22] Ivan Amaya, José Carlos Ortiz-Bayliss, Alejandro Rosales-Pérez, Andrés Eduardo Gutiérrez-Rodríguez, Santiago Enrique Conant-Pablos, Hugo Terashima-Marín, and Carlos A. Coello Coello. Enhancing Selection Hyper-Heuristics via Feature Transformations. *IEEE Computational Intelligence Magazine*, 13(2):30–41, May 2018.

[23] Jorge Humberto Arce Rincón. *Estados de Mínima Energía en una Red a Temperatura Cero*. PhD thesis, Doctorado en Ciencias (Física), 1990.

[24] Maritza Arganis, Rafael Val, Jordi Prats, Katya Rodríguez, Ramón Domínguez, and Josep Dolz. Genetic Programming and Standardization in Water Temperature Modelling. *Advances in Civil Engineering*, 2009, 2009. Article ID 353960.

[25] Alfredo Arias Montaño and Carlos A. Coello Coello. pMODE-LS+SS: An Effective and Efficient Parallel Differential Evolution Algorithm for Multi-Objective Optimization. In *Parallel Problem Solving from Nature–PPSN XI, 11th International Conference, Proceedings, Part*

II, pages 21–30. Springer, Lecture Notes in Computer Science Vol. 6239, Kraków, Poland, September 2010.

[26] Alfredo Arias Montaño, Carlos A. Coello Coello, and Efrén Mezura-Montes. MODE-LD+SS: A Novel Differential Evolution Algorithm Incorporating Local Dominance and Scalar Selection Mechanisms for Multi-Objective Optimization. In *2010 IEEE Congress on Evolutionary Computation (CEC'2010)*, pages 3284–3291, Barcelona, Spain, July 18–23 2010. IEEE Press.

[27] V. Arkov, C. Evans, P.J. Fleming, D.C. Hill, J.P. Norton, I. Pratt, D. Rees, and K. Rodríguez-Vázquez. System identification strategies applied to aircraft gas turbine engines. *Annual Reviews in Control*, 24:67–81, 2000.

[28] Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, and Robert G. Reynolds. An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 2958–2965, Vancouver, British Columbia, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0624-3.

[29] Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Robert G. Reynolds, and Ali M. Shatnawi. A novel differential crossover strategy based on covariance matrix learning with Euclidean neighborhood for solving real-world problems. In *2017 IEEE Congress on Evolutionary Computation (CEC'2017)*, pages 380–386, San Sebastian, Spain, 5-8 June 2017. IEEE Press. ISBN 978-1-5090-4602-7.

[30] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice. Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, New York, USA, 1996. ISBN:0-19-509971-0.

[31] Thomas Bäck and Ron Breukelaar. Using Genetic Algorithms to Evolve Behavior in Cellular Automata. In Cristian S. Calude, Michael J. Dinneen, Gheorghe Păun, Mario J. Pérez-Jimenez, and Grzegorz Rozenberg, editors, *Unconventional Computation, 4th International Conference, UC 2005*, pages 1–10, Sevilla, Spain, October 3-7 2005. Springer-Verlag. Lecture Notes in Computer Science Vol. 3699. 978-3-540-29100-8.

[32] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators.* CRC Press, Boca Raton, Florida, USA, 2000. ISBN 978-075030664-5.

[33] Thomas Bäck and Hans-Paul Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, March 1993.

[34] Johannes Bader and Eckart Zitzler. HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evolutionary Computation*, 19(1):45–76, Spring 2011.

[35] Shumeet Baluja. Structure and Performance of Fine-Grain Parallelism in Genetic Search. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 155–162, San Mateo, California, USA, 1993. Morgan Kaufmann Publishers.

[36] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming : An Introduction. On the Auto-*

matic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers, San Francisco, California, USA, 1998. 1-55860-510-X.

[37] P. Baraldi, G. Bonfanti, and E. Zio. Differential evolution-based multi-objective optimization for the definition of a health indicator for fault diagnostics and prognostics. *Mechanical Systems and Signal Processing*, 102:382–400, 2018.

[38] James C. Bean. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[39] Slim Bechikh, Rituparna Datta, and Abhishek Gupta, editors. *Recent Advances in Evolutionary Multi-objective Optimization*. Springer International Publishing, Switzerland, 2017. ISBN 978-3-319-42977-9.

[40] Jesus A. Beltran, Longendri Aguilera-Mendoza, and Carlos A. Brizuela. Feature weighting for antimicrobial peptides classification: A multi-objective evolutionary approach. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM'2017)*, pages 276–283, Kansas City, Missouri, USA, November 13-16 2017. IEEE Press. ISBN 978-1-5090-3051-4.

[41] Jesus A. Beltran and Carlos A. Brizuela. Design of selective cationic antibacterial peptides: A multiobjective genetic algorithm approach. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 484–491, Vancouver, British Columbia, Canada, July 24-29 2016. IEEE Press. ISBN 978-1-5090-0624-3.

[42] José A. Molinet Berenguer and Carlos A. Coello Coello. Evolutionary Many-Objective Optimization Based on Kuhn-Munkres' Algorithm.

In *Evolutionary Multi-Criterion Optimization, 8th International Conference, EMO 2015*, pages 3–17. Springer. Lecture Notes in Computer Science Vol. 9019, Guimarães, Portugal, March 29 - April 1 2015.

[43] Nicola Beume, Boris Naujoks, and Michael Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 16 September 2007.

[44] Hans-Georg Beyer and Bernhard Sendhoff. Covariance Matrix Adaptation Revisited - The CMSA Evolution Strategy -. In Günter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature - PPSN X*, Lecture Notes in Computer Science Vol. 5199, pages 123–132. Springer, 2008.

[45] M. Bhattacharya, R. Islam, and J. Abawajy. Evolutionary optimization: A big data perspective. *Journal of Network and Computer Applications*, 59:416 – 426, 2016.

[46] X-J Bi and J. Xiao. Classification-based self-adaptive differential evolution with fast and reliable convergence performance. *Soft Computing*, 15(8):1581–1599, 2011.

[47] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 494–508, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[48] Antonio Bolufé-Röhler, Suilan Estévez-Velarde, Alejandro Piad-Morffis, Stephen Chen, and James Montgomery. Differential evolution with thresheld convergence. In *2013 IEEE Congress on Evolutionary Computation (CEC'2013)*, pages 40–47, Cancún, México, 20-23 June 2013. IEEE Press. ISBN 978-1-4799-0453-2.

[49] V. Joseph Bowman Jr. On the Relationship of the Tchebycheff Norm and the Efficient Frontier of Multiple-Criteria Objectives. In Hervé Thiriez and Stanley Zionts, editors, *Multiple Criteria Decision Making*, pages 76–86. Springer. Lecture Notes in Economics and Mathematical Systems Vol. 130, Jouy-en-Josas, France, 1976. ISBN 978-3-540-07794-7.

[50] G. E. P. Box. Evolutionary Operation: A Method for Increasing Industrial Productivity. *Applied Statistics*, 6(2):81–101, 1957.

[51] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[52] J. Branke, B. Scheckenbach, M. Stein, K. Deb, and H. Schmeck. Portfolio optimization with an envelope-based multi-objective evolutionary algorithm. *European Journal of operational Research*, 199(3):684–693, December 16 2009.

[53] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors. *Multiobjective Optimization. Interactive and Evolutionary Approaches*. Springer. Lecture Notes in Computer Science Vol. 5252, Berlin, Germany, 2008.

[54] H. J. Bremermann. Optimization through evolution and recombination. In M. C. Yovits, G. T. Jacobi, and G. D. Golstine, editors,

Proceedings of the Conference on Self-Organizing Systems, pages 93–106, Washington, DC, USA, 1962. Spartan Books.

[55] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[56] J. Brest and M. Sepesy Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, December 2008.

[57] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 1188–1195, Vancouver, British Columbia, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0624-3.

[58] Janez Brest, Mirjam Sepesy Maučec, and Borko Bošković. Single objective real-parameter optimization: Algorithm jSO. In *2017 IEEE Congress on Evolutionary Computation (CEC'2017)*, pages 1311–1318, San Sebastian, Spain, 5-8 June 2017. IEEE Press. ISBN 978-1-5090-4602-7.

[59] Ron Breukelaar and Thomas Bäck. Using a Genetic Algorithm to Evolve Behaviour in Multi Dimensional Cellular Automata. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'2005)*, pages 107–114, New York, USA, 2005. ACM Press. ISBN 1-59593-010-8.

[60] Dimo Brockhoff, Tobias Friedrich, and Frank Neumann. Analyzing Hypervolume Indicator Based Algorithms. In *Parallel Problem Solving from Nature–PPSN X*, pages 651–660. Springer. Lecture Notes in Computer Science Vol. 5199, Dortmund, Germany, September 2008.

[61] Dimo Brockhoff and Tobias Wagner. GECCO 2016 Tutorial on Evolutionary Multiobjective Optimization. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 201–227, New York, USA, July 20-24 2016. ACM Press. ISBN 978-1-4503-4323-7.

[62] Lam Thu Bui, Hussein A Abbass, and Jürgen Branke. Multiobjective optimization for dynamic environments. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2349–2356. IEEE, 2005.

[63] Petr Bujok and Josef Tvrdík. Enhanced Individual-dependent Differential Evolution with Population Size Adaptation. In *2017 IEEE Congress on Evolutionary Computation (CEC'2017)*, pages 1358–1365, San Sebastian, Spain, 5-8 June 2017. IEEE Press. ISBN 978-1-5090-4602-7.

[64] M. Burgin and E. Eberbach. Recursively Generated Evolutionary Turing Machines and Evolutionary Automata. In X.-S. Yang, editor, *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pages 201–230. Springer, Berlin, Germany, 2013.

[65] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, December 2013.

[66] Maria Bárbara Calva-Yanez, Paola Andrea Niño-Suarez, Edgar Alberto Portilla-Flores, Jorge Alexander Aponte-Rodríguez, and Eric Santiago-Valentin. Reconfigurable Mechanical System Design for Tracking an Ankle Trajectory Using an Evolutionary Optimization Algorithm. *IEEE Access*, 5:5480–5493, 2017.

[67] D.T. Campbell. Blind Variation and Selective Survival as a General Strategy in Knowledge-Processes. In M.C. Yovits and S. Cameron, editors, *Self-Organizing Systems*, pages 205–231. Pergamon Press, New York, USA, 1960.

[68] W.B. Cannon. *The Wisdom of the Body.* W.W. Norton & Company, Inc., 1932.

[69] Erick Cantú-Paz. A Summary of Research on Parallel Genetic Algorithms. Technical Report IlliGAL report 95007„ University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1995.

[70] Erick Cantú-Paz. Migration Policies and Takeover Times in Parallel Genetic Algorithms. In Wolfgang Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1, page 775. Morgan Kaufmann Publishers, Orlanda, Florida, USA, July 13-17 1999. ISBN 1-55860-611-4.

[71] Erick Cantú-Paz. Topologies, Migration Rates, and Multi-population Parallel Genetic Algorithms. In Wolfgang Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1, pages 91–98, Orlanda, Florida, USA, July 13-17 1999. Morgan Kaufmann Publishers. ISBN 1-55860-611-4.

[72] Erick Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, USA, 2000. ISBN 978-0-7923-7221-9.

[73] Erick Cantú-Paz and David E. Goldberg. On the Scalability of Parallel Genetic Algorithms. *Evolutionary Computation*, 7(4):429–449, Winter 1999.

[74] Mauro Castelli, Leonardo Trujillo, Leonardo Vanneschi, and Aleš Popovič. Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67–74, September 2015.

[75] D.J. Cavicchio. *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, USA, 1970.

[76] Jaime Cerda, Alberto Avalos, and Mario Graff. Limitations of Genetic Programming Applied to Incipient Fault Detection: SFRA as Example. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 498–503. IEEE Press, 7-9 December 2015. ISBN 978-1-4673-9795-7.

[77] Héctor Cervantes-Culebro, Carlos A. Cruz-Villar, María-Guadalupe Martínez-Peñaloza, and Efrén Mezura-Montes. Constraint-handling techniques for the concurrent design of a five-bar parallel robot. *IEEE Access*, 5(1):23010–2302, 2017.

[78] Victor M. Cervantes-Salido, Oswaldo Jaime, Carlos A. Brizuela, and Israel M. Martinez-Perez. Improving the design of sequences for DNA computing: A multiobjective evolutionary approach. *Applied Soft Computing*, 13(12):4594–4607, December 2013.

[79] Uday K. Chakraborty, editor. *Advances in Differential Evolution*. Springer, Berlin, Germany, 2008. ISBN 978-3-540-68827-3.

[80] Kumar Chellapilla. Evolving Computer Programs Without Subtree Crossover. *IEEE Transactions on Evolutionary Computation*, 1(3):209–216, September 1997.

[81] Shin Siang Choong, Li-Pei Wong, and Chee Peng Lim. Automatic design of hyper-heuristic based on reinforcement learning. *Information Sciences*, 436-437:89 – 107, 2018.

[82] P. Civicioglu and E. Besdok. A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, 39(4):315–346, 2013.

[83] Carlos A. Coello Coello and Gary B. Lamont, editors. *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore, 2004. ISBN 981-256-106-4.

[84] Carlos A Coello Coello and Gregorio Toscano Pulido. Multiobjective structural optimization using a microgenetic algorithm. *Structural and Multidisciplinary Optimization*, 30(5):388–403, 2005.

[85] Carlos A. Coello Coello. Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering and Environmental Systems*, 17:319–346, 2000.

[86] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32(3):275–308, 2000.

[87] Carlos A. Coello Coello. A Short Tutorial on Evolutionary Multiobjective Optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 21–40. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[88] Carlos A Coello Coello and Ricardo Landa Becerra. Evolutionary multiobjective optimization in materials science and engineering. *Materials and manufacturing processes*, 24(2):119–129, 2009.

[89] Carlos A. Coello Coello and Nareli Cruz Cortés. Solving Multiobjective Optimization Problems using an Artificial Immune System. *Genetic Programming and Evolvable Machines*, 6(2):163–190, June 2005.

[90] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.

[91] Carlos A. Coello Coello and Ricardo Landa Becerra. Adding Knowledge and Efficient Data Structures to Evolutionary Programming: A Cultural Algorithm for Constrained Optimization. In W.B. Langdon et al., editor, *Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 201–209, San Francisco, California, USA, July 2002. Morgan Kaufmann Publishers.

[92] Carlos A. Coello Coello and Ricardo Landa Becerra. Evolutionary Multiobjective Optimization using a Cultural Algorithm. In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 6–13, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[93] Carlos A. Coello Coello, Carlos Segura, and Gara Miranda. History and Philosophy of Evolutionary Computation. In Plamen Angelov, editor, *Handbook on Computational Intelligence*, volume 2, chapter 14, pages 509–545. World Scientific, Singapore, 2016. ISBN 978-981-4675-04-8.

[94] Carlos A. Coello Coello and Gregorio Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[95] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[96] Carlos A. Coello Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga. Handling Multiple Objectives With Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, June 2004.

[97] Jose Colbes, Sergio A. Aguila, and Carlos A. Brizuela. Scoring of Side-Chain Packings: An Analysis of Weight Factors and Molecular Dynamics Structures. *Journal of Chemical Information and Modeling*, 58(2):443–452, 2018.

[98] F. N. Cornett. An Application of Evolutionary Programming to Pattern Recognition. Master's thesis, New Mexico State University, Las Cruces, New Mexico, USA, 1972.

[99] P. Corr, B. McCollum, M. McGreevy, and P. McMullan. A new neural network based construction heuristic for the examination timetabling problem. In Thomas Runarsson, Hans-Georg Beyer, Edmund Burke, Juan Merelo-Guervós, L. Whitley, and Xin Yao, editors, *Parallel Problem Solving from Nature (PPSN IX)*, volume 4193 of *Lecture Notes in Computer Science*, pages 392–401. Springer, 2006.

[100] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In Edmund Burke and Wilhelm Erben, editors, *Practice and Theory of Automated Timetabling III*, pages 176–190, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[101] J.L. Crosby. The Use of Electronic Computation in the Study of Random Fluctuations in Rapidly Evolving Populations. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 242(697):550–572, 1960.

[102] J.L. Crosby. Computers in the study of evolution. *Science Progress Oxford*, 55:279–292, 1967.

[103] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential Evolution Using a Neighborhood-Based Mutation Operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, 2009.

[104] Swagatam Das, Amit Konar, and Uday K. Chakraborty. Two Improved Differential Evolution Schemes for Faster Global Search. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*, pages 991–998, Washington DC, USA, 25-29 June 2005. ACM Press. ISBN 1-59593-010-8.

[105] Swagatam Das, Sankha Subhra-Mullick, and P.N. Suganthan. Recent advances in differential evolution – an updated survey. *Swarm and Evolutionary Computation*, 27:1–30, April 2016.

[106] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, February 2011.

[107] Dipankar Dasgupta and Zbigniew Michalewicz. *Evolutionary Algorithms in Engineering Applications*. Springer, Berlin, Germany, 1997. ISBN 978-3-540-62021-1.

[108] Patrick De Causmaecker. Data Science Meets Optimization. In Antonio Sforza and Claudio Sterle, editors, *Optimization and Decision Science: Methodologies and Applications*, pages 13–20. Springer International Publishing, Cham, Switzerland, 2017. ISBN 978-3-319-67308-0.

[109] Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. The MIT Press, Cambridge, Massachusetts, USA, 2006. ISBN 0-262-04194-4.

[110] Kenneth Alan De Jong. *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, USA, August 1975.

[111] D. W. Dearholt. Some experiments on generalization using evolving automata. In *9th Hawaii Internation Conference on System Sciences*, pages 131–133, Honolulu, Hawaii, USA, 1976. Western Periodicals.

[112] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.

[113] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995.

[114] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[115] Kalyanmoy Deb and Hans-Georg Beyer. Self-Adaptive Genetic Algorithms with Simulated Binary Crossover. *Evolutionary Computation*, 9(2):197–221, Summer 2001.

[116] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, USA, June 1989. Morgan Kaufmann Publishers.

[117] Kalyanmoy Deb and Himanshu Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Non-dominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, August 2014.

[118] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[119] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Test Problems for Evolutionary Multiobjective Optimization. In *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, pages 105–145. Springer, USA, 2005.

[120] R. Vasundhara Devi, S. Siva Sathya, and Mohane Selvaraj Coumar. Evolutionary algorithms for *de novo* drug design–A survey. *Applied Soft Computing*, 27:543–552, February 2015.

[121] Alan Díaz-Manríquez, Gregorio Toscano-Pulido, Carlos A. Coello Coello, and Ricardo Landa-Becerra. A Ranking Method Based on the $R2$ Indicator for Many-Objective Optimization. In *2013 IEEE Congress on Evolutionary Computation (CEC'2013)*, pages 1523–1530, Cancún, México, 20-23 June 2013. IEEE Press. ISBN 978-1-4799-0454-9.

[122] Stephen Dignum and Riccardo Poli. Sub-tree Swapping Crossover and Arity Histogram Distributions. In *European Conference on Genetic Programming EuroGP 2010*, pages 38–49. Springer, Berlin, Heidelberg, 2010.

[123] Saúl Domínguez-Isidro and Efrén Mezura-Montes. A cost-benefit local search coordination in multimeme differential evolution for constrained numerical optimization problems. *Swarm and Evolutionary Computation*, 39:249–266, 2017.

[124] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004. ISBN 0-262-04219-3.

[125] León Dozal, José L. Silván-Cárdenas, Daniela Moctezuma, Oscar S. Siordia, and Enrique Naredo. Evolutionary Approach for Detection of Buried Remains Using Hyperspectral Images. *Photogrammetric Engineering & Remote Sensing*, 84(7):435–450, 7 2018.

[126] A. Draa, S. Bouzoubia, and I. Boukhalfa. A sinusoidal differential evolution algorithm for numerical optimisation. *Applied Soft Computing*, 27:99–126, 2015.

[127] B. Dunham, D. Fridshal, R. Fridshal, and J. H. North. Design by natural selection. *Synthese*, 15(1):254–259, 1963.

[128] J.J. Durillo, A.J. Nebro, C.A. Coello Coello, J. Garcia-Nieto, F. Luna, and E. Alba. A Study of Multiobjective Metaheuristics When Solving Parameter Scalable Problems. *IEEE Transactions on Evolutionary Computation*, 14(4):618–635, August 2010.

[129] Juan J. Durillo and Antonio J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, October 2011.

[130] Mariana Edith-Miranda-Varela and Efrén Mezura-Montes. Constraint-handling techniques in surrogate-assisted evolutionary optimization. an empirical study. *Applied Soft Computing*, 73:215–229, 2018.

[131] Matthias Ehrgott. *Multicriteria Optimization*. Springer, Berlin, Germany, 2005. ISBN 978-3-540-21398-7.

[132] A.E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, Germany, 2003. ISBN 978-3-642-07285-7.

[133] Agoston E. Eiben and Cornelis A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, August 1998.

[134] Saber Elsayed, Noha Hamza, and Ruhul Sarker. Testing united multi-operator evolutionary algorithms-II on single objective optimization problems. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 2966–2973, Vancouver, British Columbia, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0624-3.

[135] Michael T.M. Emmerich and André H. Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17(3):585–609, September 2018.

[136] Susan L. Epstein, Eugene C. Freuder, Richard Wallace, Anton Morozov, and Bruce Samuels. The Adaptive Constraint Engine. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP'02, pages 525–542, London, UK, 2002. Springer-Verlag. ISBN 3-540-44120-4.

[137] Hugo Jair Escalante, Mario Graff, and Alicia Morales-Reyes. PGGP: Prototype Generation via Genetic Programming. *Applied Soft Computing*, 40:569–580, March 2016.

[138] Hugo Jair Escalante, Maribel Marin-Castro, Alicia Morales-Reyes, Mario Graff, Alejandro Rosales-Perez, Manuel Montes y Gomez, Carlos A. Reyes, and Jesus A. Gonzalez. MOPG: A Multi-Objective

Evolutionary Algorithm for Prototype Generation. *Pattern Analysis and Applications*, 20(1):33–47, February 2017.

[139] Hugo Jair Escalante, Jose Martinez-Carraza, Sergio Escalera, Victor Ponce-Lopez, and Xavier Baro. Improving bag of visual words representations with genetic programming. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 7 2015.

[140] Larry J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Gregory J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, volume 1, pages 265 – 283. Morgan Kaufmann Publishers, 1991.

[141] Larry J. Eshelman and J. David Schaffer. Real-coded Genetic Algorithms and Interval-Schemata. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, San Mateo, California, USA, 1993.

[142] David Espinoza-Nevárez, José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Gustavo Gatica. Selection and generation hyper-heuristics for solving the vehicle routing problem with time windows. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, GECCO '16 Companion, pages 139–140, New York, NY, USA, 2016. ACM.

[143] C. Evans, P.J. Fleming, D.C. Hill, J.P. Norton, I. Pratt, D. Rees, and K. Rodriguez-Vazquez. Application of system identification techniques to aircraft gas turbine engines. *Control Engineering Practice*, 9(2):135–148, 2 2001.

[144] C.R. Evans and A.D.J. Robertson, editors. *Cybernetics: Key Papers*. University Park Press, Baltimore, Maryland, USA, 1968.

[145] Jesús Guillermo Falcón-Cardona and Carlos A. Coello Coello. A Multi-Objective Evolutionary Hyper-Heuristic Based on Multiple Indicator-Based Density Estimators. In *2018 Genetic and Evolutionary Computation Conference (GECCO'2018)*, pages 633–640, Kyoto, Japan, July 15–19 2018. ACM Press. ISBN: 978-1-4503-5618-3.

[146] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 375–382, San Mateo, California, USA, 1993. Morgan Kaufmann Publishers.

[147] Hsiao-Lan Fang, Peter Ross, and David Corne. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. In *ECAI'94 Proceedings of the 11th European Conference on Artificial Intelligence*, pages 590–594, Amsterdam, The Netherlands, August 8-12 1994. John Wiley & Sons, Ltd.

[148] Yongsheng Fang and Jun Li. A Review of Tournament Selection in Genetic Programming. In Zhihua Cai, Chengyu Hu, Zhup Kang, and Yong Liu, editors, *Advances in Computation and Intelligence, 5th International Symposium, ISICA 2010*, pages 181–192. Springer. Lecture Notes in Computer Science Vol. 6382, Wuhan, China, October 22-24 2010.

[149] H. Faris, B. Al-Shboul, and N. Ghatasheh. A Genetic Programming Based Framework for Churn Prediction in Telecommunication Indus-

try. In D. Hwang, J. J. Jung, and N. T. Nguyen, editors, *Computational Collective Intelligence. Technologies and Applications*, pages 353–362, Cham, 2014. Springer International Publishing.

[150] V. Feoktistov. *Differential Evolution: In Search of Solutions*. Springer Optimization and Its Applications. Springer, 2007.

[151] C. Ferreira. Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13(2):87–129, 2001.

[152] Filomena Ferrucci, Pasquale Salza, and Federica Sarro. Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models. *Evolutionary Computation*, 26(4):535–567, Winter 2018.

[153] Juan J. Flores and Mario Graff. System Identification Using Genetic Programming and Gene Expression Programming. In Pínar Yolum, Tunga Güngör, Fikret Gürgen, and Can Özturan, editors, *Computer and Information Sciences - ISCIS 2005, 20th International Symposium*, pages 503–511. Springer. Lecture Notes in Computer Science Vol. 3733, Istanbul, Turkey, October 26-28 2005.

[154] David B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994.

[155] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, New Jersey, USA, 1995.

[156] David B. Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, 1998.

[157] D.B. Fogel and J.W. Atmar. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63(2):111–114, 1990.

[158] D.B. Fogel, L.J. Fogel, and J.W. Atmar. Meta-Evolutionary Programming. In *Twenty-Fifth Asilomar Conference on Signals, Systems & Computers*, pages 540–545, Pacific Grove, California, USA, 4-6 November 1991. IEEE Press. ISBN 0-8186-2470-1.

[159] Lawrence J. Fogel. Autonomous automata. *Industrial Research Magazine*, 4:14–19, 1962.

[160] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution. Forty Years of Evolutionary Programming.* John Wiley & Sons, Inc., New York, 1999.

[161] Lawrence J. Fogel, Alvin J. Owens, and Michael John Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons, 1966.

[162] Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors. *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003.* Springer. Lecture Notes in Computer Science. Volume 2632, Faro, Portugal, April 2003.

[163] A. S. Fraser and D. Burnell. *Computer models in genetics.* McGraw-Hill, New York, USA, 1970.

[164] A.S. Fraser. Simulation of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological Systems*, 10:484–491, 1957.

[165] A.S. Fraser and D. Burnell. Simulation of Genetic Systems XII. Models of Inversion Polymorphism. *Genetics*, 57:267–282, 1967.

[166] R. M. Friedberg. A Learning Machine: Part I. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[167] R. M. Friedberg, B. Dunham, and J. H. North. A Learning Machine: Part II. *IBM Journal of Research and Development*, 3(3):282–287, 1959.

[168] G. J. Friedman. Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy. Master's thesis, University of California, Los Angeles, 1956.

[169] Edgar Galván-López, Efrén Mezura-Montes, Ouassim Ait ElHara, and Marc Schoenauer. On the Use of Semantics in Multi-Objective Genetic Programming. In *Parallel Problem Solving from Nature – PPSN XIV, 14th International Conference*, pages 353–363. Springer. Lecture Notes in Computer Science Vol. 9921, Edinburgh, UK, September 17-21 2016. ISBN 978-3-319-45822-9.

[170] Theodore W. Gamelin and Robert Everist Greene. *Introduction to Topology*. Dover Publications, second edition, 1999. ISBN 978-0486406800.

[171] R. Gämperle, S.D. Müller, and P. Koumoutsakos. A parameter study for differential evolution. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 10(10):293–298, 2002.

[172] Weifeng Gao, Gary G. Yen, and Sanyang Liu. A Cluster-Based Differential Evolution With Self-Adaptive Strategy for Multimodal Op-

timization . *IEEE Transactions on Cybernetics*, 44(8):1314–1327, August 2014.

[173] Abel García-Nájera, John A. Bullinaria, and Miguel A. Gutiérrez-Andrade. An evolutionary approach for multi-objective vehicle routing problems with backhauls. *Computers & Industrial Engineering*, 81:90–108, March 2015.

[174] Abel Garcia-Najera and Antonio Lopez-Jaimes. An Investigation into Many-Objective Optimization on Combinatorial Problems: Analyzing the Pickup and Delivery Problem. *Swarm and Evolutionary Computation*, 38:218–230, February 2018.

[175] José García-Nieto, Esteban López-Camacho, María Jesús García-Godoy, Antonio J Nebro, and José F Aldana-Montes. Multi-objective ligand-protein docking with particle swarm optimizers. *Swarm and Evolutionary Computation*, 44:439–452, February 2019.

[176] Mario Garza-Fabre, Carlos A. Coello Coello Gregorio Toscano-Pulido, and Eduardo Rodríguez-Tello. Effective Ranking + Speciation = Many-Objective Optimization. In *2011 IEEE Congress on Evolutionary Computation (CEC'2011)*, pages 2115–2122, New Orleans, Louisiana, USA, 5-8 June 2011. IEEE Service Center.

[177] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. Constraint-handling through multi-objective optimization: The hydrophobic-polar model for protein structure prediction. *Computers & Operations Research*, 53:128–153, January 2015.

[178] Mario Garza-Fabre, Gregorio Toscano-Pulido, and Eduardo Rodriguez-Tello. Multi-objectivization, fitness landscape trans-

formation and search performance: A case of study on the hp model for protein structure prediction. *European Journal of Operational Research*, 243(2):405–422, June 1 2015.

[179] Saul Gass and Thomas Saaty. The computational algorithm for the parametric objective function. *Naval research logistics quarterly*, 2(1-2):39–45, 1955.

[180] A. Ghosh, S. Das, A. Chowdhury, and R. Giri. An improved differential evolution algorithm with fitness-based adaptation of the control parameters. *Information Sciences*, 181(18):3749–3765, 2011.

[181] Arka Ghosh, Swagatam Das, Bijaya Ketan Panigrahi, and Asit Kr. Das. A noise resilient Differential Evolution with improved parameter and strategy control. In *2017 IEEE Congress on Evolutionary Computation (CEC'2017)*, pages 2590–2597, San Sebastian, Spain, 5-8 June 2017. ISBN 978-1-5090-4602-7.

[182] Mario Giacobini, Marco Tomassini, Andrea Tettamanzi, and Enrique Alba. Selection Intensity in Cellular Evolutionary Algorithms for Regular Lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505, October 2005.

[183] Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2003.

[184] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishers, 1989.

[185] Juan Carlos Gomez and Hugo Terashima-Marín. Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems. *Genetic Programming and Evolvable Machines*, 19(1):151–181, June 2018.

[186] Héctor Fernando Gomez Garcia, Arturo González Vega, Arturo Hernández Aguirre, José Luis Marroquín Zaleta, and Carlos A. Coello Coello. Robust Multiscale Affine 2D-Image Registration through Evolutionary Strategies. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villacañas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature VII*, pages 740–748. Springer-Verlag. Lecture Notes in Computer Science Vol. 2439, Granada, Spain, September 2002.

[187] Mario Graff, Hugo Jair Escalante, Fernando Ornelas-Tellez, and Eric S. Tellez. Time series forecasting with genetic programming. *Natural Computing*, 16(1):165–174, 3 2017.

[188] Mario Graff, Sabino Miranda-Jiménez, Eric S Tellez, and Daniela Moctezuma. INGEOTEC at SemEval-2018 Task 1: EvoMSA and microTC for Sentiment Analysis. In *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, pages 146–150, 2018.

[189] Mario Graff and Riccardo Poli. Practical performance models of algorithms in evolutionary program induction and other domains. *Artificial Intelligence*, 174(15):1254–1276, October 2010.

[190] Mario Graff, Riccardo Poli, and Juan J. Flores. Models of Performance of Evolutionary Program Induction Algorithms Based on Indicators of Problem Difficulty. *Evolutionary Computation*, 21(4):533–560, Winter 2013.

[191] Mario Graff, Eric S. Tellez, Hugo Jair Escalante, and Sabino Miranda-Jiménez. Semantic Genetic Programming for Sentiment Analysis. In Oliver Schütze, Leonardo Trujillo, Pierrick Legrand, and Yazmin Maldonado, editors, *NEO 2015: Results of the Numerical and Evolutionary Optimization Workshop NEO 2015 held at September 23-25 2015 in Tijuana, Mexico*, pages 43–65. Springer, Cham, Switzerland, 2017. ISBN 978-3-319-44002-6.

[192] Mario Graff, Eric S. Tellez, Sabino Miranda-Jimenez, and Hugo Jair Escalante. EvoDAG: A semantic Genetic Programming Python library. In *2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC'2016)*, Ixtapa, Mexico, 9-11 November 2016. IEEE Press. ISBN 978-1-5090-3794-0.

[193] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic Algorithms for the Traveling Salesman Problem. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 160–168, Hillsdale, New Jersey, USA, July 24-26 1985. Lawrence Erlbaum Associates, Publishers. ISBN 0-8058-0426-9.

[194] John J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, January 1986.

[195] Shenkai Gu, Ran Cheng, and Yaochu Jin. Feature selection for high-dimensional classification using a competitive swarm optimizer. *Soft Computing*, 22(3):811–822, 2018.

[196] Juan Adolfo Montesino Guerra, Héctor José Puga Soberanes, Marco Aurelio Sotelo Figueroa, Juan Martín Carpio Valadez, Manuel Or-

nelas Rodríguez, Jorge Alberto Soria Alcaraz, and Raúl Santiago Montero. Comportamiento Sinérgico En Hiperheurística de Selección para la Solución de los Problemas del Agente Viajero. *Programación Matemática y Software*, 8(3):30–41, 2016.

[197] José-Yair Guzmán-Gaspar and Efrén Mezura-Montes. Robust Optimization Over Time with Differential Evolution using an Average Time Approach. In *2019 IEEE Congress on Evolutionary Computation (CEC'2019)*, pages 1548–1555, Wellington, New Zealand, 10-13 June 2019. IEEE Press. ISBN 978-1-7281-2153-6.

[198] Mohammad Hamdan. The distribution index in polynomial mutation for evolutionary multiobjective optimisation algorithms: An experimental study. In *International Conference on Electronics Computer Technology, Kanyakumari, India*, 2012.

[199] Guanghong Han and Xi Chen. A Bi-level Differential Evolutionary Algorithm for Constrained Optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC'2019)*, pages 1628–1633, Wellington, New Zealand, 10-13 June 2019. IEEE Press. ISBN 978-1-7281-2153-6.

[200] Julia Handl, Simon C. Lovell, and Joshua Knowles. Multiobjectivization by Decomposition of Scalar Cost Functions. In *Parallel Problem Solving from Nature–PPSN X*, pages 31–40. Springer. Lecture Notes in Computer Science Vol. 5199, Dortmund, Germany, September 2008.

[201] Julia Handl, Simon C. Lovell, and Joshua D. Knowles. Investigations into the Effect of Multiobjectivization in Protein Structure Prediction. In *Parallel Problem Solving from Nature–PPSN X*, pages 702–711.

Springer. Lecture Notes in Computer Science Vol. 5199, Dortmund, Germany, September 2008.

[202] Michael Pilegaard Hansen and Andrzej Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Technical University of Denmark, March 1998.

[203] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, June 2001.

[204] Emma Hart and Kevin Sim. A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling. *Evolutionary Computation*, 24(4):609–635, Winter 2016.

[205] Arturo Hernández Aguirre, Salvador Botello Rionda, Carlos A. Coello Coello, Giovanni Lizárraga Lizárraga, and Efrén Mezura Montes. Handling Constraints using Multiobjective Optimization Concepts. *International Journal for Numerical Methods in Engineering*, 59(15):1989–2017, April 2004.

[206] Raquel Hernández Gómez and Carlos A. Coello Coello. MOMBI: A New Metaheuristic for Many-Objective Optimization Based on the R2 Indicator. In *2013 IEEE Congress on Evolutionary Computation (CEC'2013)*, pages 2488–2495, Cancún, México, 20-23 June 2013. IEEE Press. ISBN 978-1-4799-0454-9.

[207] Jesus Alejandro Hernandez Mejia, Oliver Schutze, Oliver Cuate, Adriana Lara, and Kalyanmoy Deb. RDS-NSGA-II: A Memetic Algorithm for Reference Point Based Multi-Objective Optimization. *Engineering Optimization*, 49(5):828–845, 2017.

[208] Leticia Hernando, Alexander Mendiburu, and Jose A. Lozano. An Evaluation of Methods for Estimating the Number of Local Optima in Combinatorial Optimization Problems . *Evolutionary Computation*, 21(4):625–658, Winter 2013.

[209] Francisco Herrera and Manuel Lozano. Adaptation of genetic algorithm parameters based on fuzzy logic controllers. *Genetic Algorithms and Soft Computing*, 8:95–125, 1996.

[210] Francisco Herrera and Manuel Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, 2000.

[211] Francisco Herrera and Manuel Lozano. Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. *Soft Computing*, 7(8):545–562, 2003.

[212] W. D. Hillis. Co-evolving Parasites Improve Simulated Evolution As an Optimization Procedure. *Physica D: Nonlinear Phenomena*, 42(1–3):228–234, 1990.

[213] J. H. Holland. Outline for a Logical Theory of Adaptive Systems. *Journal of the ACM*, 9(3):297–314, July 1962.

[214] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[215] Simon Huband, Luigi Barone, Lyndon While, and Phil Hingston. A Scalable Multi-objective Test Problem Toolkit. In *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 280–295, Guanajuato, México, March 2005. Springer. Lecture Notes in Computer Science Vol. 3410.

[216] Evan J. Hughes. Evolutionary Many-Objective Optimisation: Many Once or One Many? In *2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, volume 1, pages 222–227, Edinburgh, Scotland, September 2005. IEEE Service Center.

[217] Christian Igel, Nikolaus Hansen, and Stefan Roth. Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary Computation*, 15(1):1–28, Spring 2007.

[218] Hisao Ishibuchi, Ryo Imada, Yu Setoguchi, and Yusuke Nojima. Reference Point Specification in Inverted Generational Distance for Triangular Linear Pareto Front. *IEEE Transactions on Evolutionary Computation*, 22(6):961–975, December 2018.

[219] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified Distance Calculation in Generational Distance and Inverted Generational Distance. In *Evolutionary Multi-Criterion Optimization, 8th International Conference, EMO 2015*, pages 110–125. Springer. Lecture Notes in Computer Science Vol. 9019, Guimarães, Portugal, March 29 - April 1 2015.

[220] S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan. An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):482–500, 2012.

[221] A. Jain and D.B. Fogel. Case studies in applying fitness distributions in evolutionary algorithms. II. Comparing the improvements from crossover and Gaussian mutation on simple neural networks. In

2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, pages 91–97, San Antonio, Texas, USA, 11-13 May 2000. IEEE Press. ISBN 0-7803-6572-0.

[222] Himanshu Jain and Kalyanmoy Deb. Parent to Mean-Centric Self-Adaptation in SBX Operator for Real-Parameter Optimization. In Bijaya Ketan Panigrahi, Ponnuthurai Nagaratnam Suganthan, Swagatam Das, and Suresh Chandra Satapathy, editors, *Swarm, Evolutionary, and Memetic Computing, Second International Conference, SEMCCO 2011*, pages 299–306. Springer. Lecture Notes in Computer Science Vol. 7076, Visakhapatnam, Andhra Pradesh, India, December 19-21 2011. ISBN 978-3-642-27171-7.

[223] Hugo Jair Escalante, Maribel Marin-Castro, Alicia Morales-Reyes, Mario Graff, Alejandro Rosales-Perez, Manuel Montes y Gomez, Carlos A. Reyes, and Jesus A. Gonzalez. MOPG: A Multi-Objective Evolutionary Algorithm for Prototype Generation. *Pattern Analysis and Applications*, 20(1):33–47, February 2017.

[224] Thomas Jansen. On the Analysis of Dynamic Restart Strategies for Evolutionary Algorithms. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, Hans-Paul Schwefel, and José-Luis Fernández-Villacañas, editors, *Parallel Problem Solving from Nature – PPSN VII, 7th International Conference*, pages 33–43. Springer. Lecture Notes in Computer Science Vol. 2439, Granada, Spain, September 7-11 2002.

[225] Andrzej Jaszkiewicz. Improved quick hypervolume algorithm. *Computers & Operations Research*, 90:72–83, February 2018.

[226] Mikkel T Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2004.

[227] Yaochu Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[228] Julio Juárez and Carlos A. Brizuela. A Multi-Objective Formulation of the Team Formation Problem in Social Networks: Preliminary Results. In *2018 Genetic and Evolutionary Computation Conference (GECCO'2018)*, pages 261–268, Kyoto, Japan, July 15–19 2018. ACM Press. ISBN: 978-1-4503-5618-3.

[229] Julio Juárez, Carlos A. Brizuela, and Israel M. Martínez-Pérez. An evolutionary multi-objective optimization algorithm for the routing of droplets in digital microfluidic biochips. *Information Sciences*, 429:130–146, March 2018.

[230] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, USA, 2001.

[231] V. Khare, X. Yao, and K. Deb. Performance Scaling of Multi-objective Evolutionary Algorithms. In *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 376–390, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[232] Joshua Knowles and David Corne. On Metrics for Comparing Nondominated Sets. In *Congress on Evolutionary Computation*

(CEC'2002), volume 1, pages 711–716, Piscataway, New Jersey, May 2002. IEEE Service Center.

[233] Joshua Knowles and David Corne. Quantifying the Effects of Objective Space Dimension in Evolutionary Multiobjective Optimization. In *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007*, pages 757–771, Matshushima, Japan, March 2007. Springer. Lecture Notes in Computer Science Vol. 4403.

[234] Joshua D. Knowles. *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, The University of Reading, Department of Computer Science, Reading, UK, January 2002.

[235] Joshua D. Knowles and David W. Corne. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In *1999 Congress on Evolutionary Computation*, pages 98–105, Washington, D.C., July 1999. IEEE Service Center.

[236] Joshua D. Knowles, Richard A. Watson, and David W. Corne. Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 268–282. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[237] Abdullah Konak, David W Coit, and Alice E Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[238] Vlasis K Koumousis and Christos P Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitial-

ization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28, 2006.

[239] J. R. Koza. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In N. S. Sridharan, editor, *IJCAI'89*, pages 768–774. Morgan Kauffman, 1989.

[240] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, Massachusetts, USA, 1992.

[241] Krzysztof Krawiec. Semantic Genetic Programming. In *Behavioral Program Synthesis with Genetic Programming*, pages 55–66. Springer, Cham, 2016.

[242] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[243] Saku Kukkonen and Carlos A. Coello Coello. Applying Exponential Weighting Moving Average Control Parameter Adaptation Technique with Generalized Differential Evolution. In *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 4755–4762, Vancouver, British Columbia, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0624-3.

[244] Abhishek Kumar, Rakesh Kumar Misra, and Devender Singh. Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 1835–1842. IEEE, 2017.

[245] Renaud Lacour, Kathrin Klamroth, and Carlos M. Fonseca. A Box Decomposition Algorithm to Compute the Hypervolume Indicator. *Computers & Operations Research*, 79:347–360, March 2017.

[246] Eric-Wubbo Lameijer, Thomas Bäck, Joost N Kok, and Ad P Ijzerman. Evolutionary algorithms in drug design. *Natural Computing*, 4(3):177–243, 2005.

[247] J. Lampinen and I. Zelinka. On stagnation of the differential evolution algorithm. In *Proceedings of 6th Int. Mendel Conference on Soft Computing*, pages 76–83, 2000.

[248] R. Landa Becerra and C. A. Coello Coello. Solving hard multiobjective optimization problems using ε-constraint with cultured differential evolution. In *Parallel Problem Solving from Nature-PPSN IX*, pages 543–552. Springer, 2006.

[249] Adriana Lara, Gustavo Sanchez, Carlos A. Coello Coello, and Oliver Schütze. HCS: A New Local Search Strategy for Memetic Multi-Objective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 14(1):112–132, February 2010.

[250] Joel Lehman and Kenneth O. Stanley. Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, 19(2):189–223, Summer 2011.

[251] M. Leon and N. Xiong. Adapting differential evolution algorithms for continuous optimization via greedy adjustment of control parameters. *Journal of Artificial Intelligence and Soft Computing Research*, 6(2):103–118, 2016.

[252] Martin Letras, Alicia Morales-Reyes, and Rene Cumplido. A scalable and customizable processor array for implementing cellular genetic algorithms. *Neurocomputing*, 175, Part B:899–910, January 2016.

[253] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-Objective Evolutionary Algorithms: A Survey. *ACM Computing Surveys*, 48(1), September 2015.

[254] Hui Li and Qingfu Zhang. A Multiobjective Differential Evolution Based on Decomposition for Multiobjective Optimization with Variable Linkages. In *Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 583–592. Springer. Lecture Notes in Computer Science Vol. 4193, Reykjavik, Iceland, September 2006.

[255] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. *IEEE Transactions on Evolutionary Computation*, 13(2):284–302, April 2009.

[256] Jingpeng Li, Edmund K. Burke, and Rong Qu. Integrating Neural Networks and Logistic Regression to Underpin Hyper-Heuristic Search. *Knowledge-Based Systems*, 24(2):322–330, March 2011.

[257] Kaiwen Li, Rui Wang, Tao Zhang, and Hisao Ishibuchi. Evolutionary Many-Objective Optimization: A Comparative Study of the State-of-the-Art. *IEEE Access*, 6:26194–26214, 7 May 2018.

[258] X. Li and M. Yin. An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. *Advances in Engineering Software*, 55:10–31, 2013.

[259] Jian Lin, Dike Luo, Xiaodong Li, Kaizhou Gao, and Yanan Liu. Differential evolution based hyper-heuristic for the flexible job-shop

scheduling problem with fuzzy processing time. In Yuhui Shi, Kay Chen Tan, Mengjie Zhang, Ke Tang, Xiaodong Li, Qingfu Zhang, Ying Tan, Martin Middendorf, and Yaochu Jin, editors, *Simulated Evolution and Learning*, pages 75–86, Cham, 2017. Springer International Publishing.

[260] Yung-Chien Lin, Kao-Shing Hwang, and Feng-Sheng Wang. Hybrid differential evolution with multiplier updating method for nonlinear constrained optimization problems. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 872–877, Honolulu, Hawaii, USA, 12-17 May 2002. IEEE Press. ISBN 0-7803-7282-4.

[261] J. Liu and J. Lampinen. On Setting the Control Parameter of the Differential Evolution Method. *Proc. of the 8th Int. Mendel Conference on Soft Computing*, pages 11–18, 2002.

[262] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.

[263] Shih-Hsi Liu, Marjan Mernik, and Barrett R Bryant. To explore or to exploit: An entropy-driven approach for evolutionary algorithms. *International Journal of Knowledge-based and Intelligent Engineering Systems*, 13(3-4):185–206, 2009.

[264] Giovanni Lizárraga Lizárraga, Arturo Hernández Aguirre, and Salvador Botello Rionda. G-Metric: an M-ary Quality Indicator for the Evaluation of Non-dominated Sets. In *2008 Genetic and Evolutionary Computation Conference (GECCO'2008)*, pages 665–672, Atlanta, USA, July 2008. ACM Press. ISBN 978-1-60558-131-6.

[265] FJ Lobo, Cláudio F Lima, and Zbigniew Michalewicz. *Parameter setting in evolutionary algorithms*, volume 54. Springer Science & Business Media, 2007.

[266] Edgar Galván López, Riccardo Poli, and Carlos A. Coello Coello. Reusing Code in Genetic Programming. In *European Conference on Genetic Programming EuroGP 2004*, pages 359–368. Springer, Berlin, Heidelberg, 2004.

[267] Jesus R. Lopez, Luis C. Gonzalez, Johan Wahlstrom, Manuel Montes y Gomez, Leonardo Trujillo, and Graciela Ramirez-Alonso. A Genetic Programming Approach for Driving Score Calculation in the Context of Intelligent Transportation Systems. *IEEE Sensors Journal*, 18(17):7183–7192, 9 2018.

[268] Roberto López, Luis González Gurrola, Leonardo Trujillo, Olanda Prieto, Graciela Ramírez, Antonio Posada, Perla Juárez-Smith, and Leticia Méndez. How Am I Driving? Using Genetic Programming to Generate Scoring Functions for Urban Driving Behavior. *Mathematical and Computational Applications*, 23(2):19, 4 2018.

[269] Eunice López-Camacho, Hugo Terashima-Marin, Peter Ross, and Gabriela Ochoa. A unified hyper-heuristic framework for solving bin packing problems. *Expert Syst. Appl.*, 41(15):6876–6889, November 2014.

[270] Antonio López-Jaimes and Carlos Coello Coello. MRMOGA: Parallel Evolutionary Multiobjective Optimization using Multiple Resolutions. In *2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, volume 3, pages 2294–2301, Edinburgh, Scotland, September 2005. IEEE Service Center.

[271] Antonio López Jaimes and Carlos A. Coello Coello. MRMOGA: A New Parallel Multi-Objective Evolutionary Algorithm Based on the Use of Multiple Resolutions. *Concurrency and Computation: Practice and Experience*, 19(4):397–441, March 2007.

[272] Antonio López-Jaimes and Carlos A. Coello Coello. Including preferences into a multiobjective evolutionary algorithm to deal with many-objective engineering optimization problems. *Information Sciences*, 227:1–20, September 1 2014.

[273] Antonio López-Jaimes, Alfredo Arias-Monta no, and Carlos A. Coello Coello. Preference Incorporation to Solve Many-Objective Airfoil Design Problems. In *2011 IEEE Congress on Evolutionary Computation (CEC'2011)*, pages 1605–1612, New Orleans, Louisiana, USA, 5-8 June 2011. IEEE Service Center.

[274] Manuel Lozano, Francisco Herrera, and José Ramón Cano. Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Information Sciences*, 178(23):4421–4433, 2008.

[275] Dinh The Luc. *Multiobjective linear programming: an introduction.* Springer International Publishing, 2016.

[276] Francisco Luna and Enrique Alba. Parallel Multiobjective Evolutionary Algorithms. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, chapter 50, pages 1017–1031. Springer-Verlag, Berlin, Germany, 2015. ISBN 978-3-662-43504-5.

[277] Gabriel Luque and Enrique Alba. *Parallel Genetic Algorithms. Theory and Real World Application.* Springer-Verlag, Berlin, Germany, 2011. ISBN 978-3-642-22083-8.

[278] M. R. Lyle. An Investigation Into Scoring Techniques in Evolutionary Programming. Master's thesis, New Mexico State University, Las Cruces, New Mexico, USA, 1972.

[279] Nateri K. Madavan. Multiobjective Optimization Using a Pareto Differential Evolution Approach. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1145–1150, Piscataway, New Jersey, May 2002. IEEE Service Center.

[280] Samir W Mahfoud. Crowding and preselection revisited. *Urbana*, 51:61801, 1992.

[281] Yuri Malitsky. Evolving instance-specific algorithm configuration. In *Instance-Specific Algorithm Configuration*, pages 93–105. Springer International Publishing, 2014.

[282] Yuri Malitsky and Meinolf Sellmann. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. In Nicolas Beldiceanu, Narendra Jussien, and Éric Pinson, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*, pages 244–259, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[283] R. Mallipeddi and P. N. Suganthan. Empirical study on the effect of population size on Differential Evolution Algorithm. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3663–3670, June 2008.

[284] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, and M.F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011.

[285] Edgar Manoatl Lopez and Carlos A. Coello Coello. IGD$^+$-EMOA: A Multi-Objective Evolutionary Algorithm based on IGD$^+$. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 999–1006, Vancouver, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0623-9.

[286] André L Maravilha, Jaime A Ramírez, and Felipe Campelo. A new algorithm based on differential evolution for combinatorial optimization. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 60–66. IEEE, 2013.

[287] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, and Uriel López. A comparison of fitness-case sampling methods for genetic programming. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(6):1203–1224, 11 2017.

[288] Yuliana Martínez, Leonardo Trujillo, Pierrick Legrand, and Edgar Galván-López. Prediction of expected performance for a genetic programming classifier. *Genetic Programming and Evolvable Machines*, 17(4):409–449, 12 2016.

[289] María-Guadalupe Martínez-Peñaloza and Efrén Mezura-Montes. Immune generalized differential evolution for dynamic multi-objective environments: An empirical study. *Knowledge-Based Systems*, 142:192–219, 2018.

[290] Mirjam Sepesy Maučec, Janez Brest, Borko Bošković, and Zdravko Kačič. Improved Differential Evolution for Large-Scale Black-Box Optimization. *IEEE Access*, 6:29516–29531, 2018.

[291] Adriana Menchaca-Mendez and Carlos A. Coello Coello. Solving Multi-Objective Optimization Problems using Differential Evolution and a Maximin Selection Criterion. In *2012 IEEE Congress on Evolutionary Computation (CEC'2012)*, pages 3143–3150, Brisbane, Australia, June 10-15 2012. IEEE Press.

[292] Adriana Menchaca-Mendez, Carlos Hernández, and Carlos A. Coello Coello. Δ_p-MOEA: A New Multi-Objective Evolutionary Algorithm Based on the Δ_p Indicator. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 3753–3760, Vancouver, Canada, 24-29 July 2016. IEEE Press. ISBN 978-1-5090-0623-9.

[293] H. D. Menéndez and D. Camacho. GANY: A genetic spectral-based clustering algorithm for Large Data Analysis. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 640–647, May 2015.

[294] Ole J Mengshoel, Severino F Galán, and Antonio De Dios. Adaptive generalized crowding for genetic algorithms. *Information Sciences*, 258:140–159, 2014.

[295] E. Mezura-Montes, C. A. Coello Coello, and E. I. Tun-Morales. Simple feasibility rules and differential evolution for constrained optimization. In *Mexican International Conference on Artificial Intelligence*, pages 707–716. Springer, 2004.

[296] E. Mezura-Montes, M. E. Miranda-Varela, and R. del C. Gómez-Ramón. Differential evolution in constrained numerical optimization: an empirical study. *Information Sciences*, 180(22):4223–4262, 2010.

[297] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A Comparative Study of Differential Evolution Variants for Global Optimization. In *Proc. of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 485–492, New York, NY, USA, 2006. ACM.

[298] Efrén Mezura-Montes and Carlos A. Coello Coello. A Simple Multimembered Evolution Strategy to Solve Constrained Optimization Problems. *IEEE Transactions on Evolutionary Computation*, 9(1):1–17, February 2005.

[299] Efrén Mezura-Montes, Margarita Reyes-Sierra, and Carlos A. Coello Coello. Multi-Objective Optimization using Differential Evolution: A Survey of the State-of-the-Art. In *Advances in Differential Evolution*, pages 173–196. Springer, Berlin, 2008. ISBN 978-3-540-68827-3.

[300] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*, volume 12. Springer Science & Business Media, 1998.

[301] Alan Tan Wei Min, Yew-Soon Ong, Abhishek Gupta, and Chi-Keong Goh. Multi-problem surrogates: Transfer evolutionary multiobjective optimization of computationally expensive problems. *IEEE Transactions on Evolutionary Computation*, 2017.

[302] L. Mingyong and C. Erbao. An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and

deliveries and time windows. *Engineering Applications of Artificial Intelligence*, 23(2):188–195, 2010.

[303] Octavio Raymundo Miramontes Vidal. Algunos Aspectos de la Teoría de Autómatas Celulares y sus Aplicaciones en Biofísica. Tesis de Licenciatura (Física), 1988.

[304] Pedro Eduardo Miramontes Vidal. *Un Esquema de Autómata Celular como Modelo Matemático de la Evolución de los Ácidos Nucleicos.* PhD thesis, Doctorado en Ciencias (Matemáticas), 1992.

[305] Péricles B.C. Miranda, Ricardo B.C. Prudêncio, and Gisele L. Pappa. H3ad: A hybrid hyper-heuristic for algorithm design. *Information Sciences*, 414:340–354, 2017.

[306] A. W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi. Lshade with semi-parameter adaptation hybrid with cma-es for solving cec 2017 benchmark problems. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 145–152, June 2017.

[307] A. W. Mohamed and A. K. Mohamed. Adaptive guided differential evolution algorithm with novel mutation for numerical optimization. *International Journal of Machine Learning and Cybernetics*, Aug 2017.

[308] Daniel Molina, Francisco Moreno-García, and Francisco Herrera. Analysis among winners of different ieee cec competitions on real-parameters optimization: Is there always improvement? In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 805–812. IEEE, 2017.

[309] James Montgomery. Differential evolution: Difference vectors and movement in solution space. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2833–2840. IEEE, 2009.

[310] James Montgomery and Stephen Chen. An analysis of the operation of differential evolution at high and low crossover rates. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[311] James Montgomery and Stephen Chen. A simple strategy for maintaining diversity and reducing crowding in differential evolution. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[312] A. Morales-reyes and A. T. Erdogan. A structure-based coarse-fine approach for diversity tuning in cellular gas. *Advances in Electrical and Computer Engineering*, 12(3):39–46, 2012.

[313] A. Morales-Reyes and A.T. Erdogan. Internal lattice reconfiguration for diversity tuning in cellular genetic algorithms. In *PLoS ONE*, page 7(7): e41279. PloS ONE, 2012.

[314] A. Morales-Reyes, A.T. Erdogan, and T. Arslan. Lattice reconfiguration vs. local selection criteria for diversity tuning in cellular gas. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 1 – 8. IEEE, 2010.

[315] Alicia Morales-Reyes, Hugo Jair Escalante, Martin Letras, and Rene Cumplido. An empirical analysis on dimensionality in cellular genetic algorithms. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 895–902, New York, NY, USA, 2015. ACM.

[316] David E. Moriarty and Risto Miikkulainen. Evolutionary neural networks for value ordering in constraint satisfaction problems. Technical Report AI94-218, Department of Computer Sciences, The University of Texas at Austin, 1994.

[317] Jean-Baptiste Mouret. Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*, pages 139–154. Springer, 2011.

[318] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. In H. M. Voigt et al., editors, *PPSN IV*, volume 1141 of *LNCS*, pages 178–187. Springer, 1996.

[319] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i. continuous parameter optimization. *Evolutionary computation*, 1(1):25–49, 1993.

[320] Anirban Mukhopadhyay, Ujjwal Maulik, and Sanghamitra Bandyopadhyay. A Survey of Multiobjective Evolutionary Clustering. *ACM Computing Surveys*, 47(4), July 2015. Article Number: 61.

[321] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos A. Coello Coello. A Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part I. *IEEE Transactions on Evolutionary Computation*, 18(1):4–19, February 2014.

[322] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos A. Coello Coello. Survey of Multiobjective Evolutionary Algorithms for Data Mining: Part II. *IEEE Transactions on Evolutionary Computation*, 18(1):20–35, February 2014.

[323] Leandro N. de Castro and Jonathan Timmis. *An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm.* Springer, London, 2002. ISBN 1-85233-594-7.

[324] Vinod Nair and Geoffrey E Hinton. 3d object recognition with deep belief nets. In *Advances in neural information processing systems*, pages 1339–1347, 2009.

[325] Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, Sara Silva, and Luis Muñoz. Evolving genetic programming classifiers with novelty search. *Information Sciences*, 369:347–367, 11 2016.

[326] Enrique Naredo, Paulo Urbano, and Leonardo Trujillo. The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Computing*, 21(15):4399–4416, 8 2017.

[327] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1-2):61–106, 2010.

[328] Networking and Emerging Optimization. *NEO Research group.* 2014.

[329] Su Nguyen, Mengjie Zhang, and Mark Johnston. A Genetic Programming Based Hyper-heuristic Approach for Combinatorial Optimisation. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1299–1306, New York, NY, USA, 2011. ACM.

[330] Christos A Nicolaou and Nathan Brown. Multi-objective optimization methods in drug design. *Drug Discovery Today: Technologies*, 10(3):e427–e435, 2013.

[331] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, and E.K. Burke. Hyflex: A benchmark framework for cross-domain heuristic search. In J.-K. Hao and M. Middendorf, editors, *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012)*, volume 7245 of *LNCS*, pages 136–147, Heidelberg, 2012. Springer.

[332] Gustavo Olague, Eddie Clemente, León Dozal, and Daniel E. Hernández. Evolving an Artificial Visual Cortex for Object Recognition with Brain Programming. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation III*, pages 97–119. Springer, Heidelberg, 2014.

[333] Gustavo Olague and Leonardo Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image and Vision Computing*, 29(7):484–498, 6 2011.

[334] Gustavo Olague and Leonardo Trujillo. Interest point detection through multiobjective genetic programming. *Applied Soft Computing*, 12(8):2566–2582, 8 2012.

[335] M.G.H. Omran, A. Salman, and A.P. Engelbrecht. Self-adaptive Differential Evolution. In Y. Hao et al., editors, *Computational Intelligence and Security*, pages 192–199, Berlin, Heidelberg, 2005. Springer.

[336] Isao Ono, Hajime Kita, and Shigenobu Kobayashi. A Real-coded Genetic Algorithm Using the Unimodal Normal Distribution Crossover. In Ashish Ghosh and Shigeyoshi Tsutsui, editors, *Advances in Evolu-*

tionary Computing, pages 213–237. Springer-Verlag, New York, NY, USA, 2003. ISBN 3-540-43330-9.

[337] G. C. Onwubolu. *Differential Evolution for the Flow Shop Scheduling Problem*, pages 585–611. Springer, 2004.

[338] Godfrey C. Onwubolu and Donald Davendra. *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[339] Karol R. Opara and Jarosław Arabas. Differential evolution: A survey of theoretical analyses. *Swarm and Evolutionary Computation*, 2018.

[340] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín. A genetic programming hyper-heuristic: Turning features into heuristics for constraint satisfaction. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, pages 183–190, September 2013.

[341] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín. A genetic programming hyper-heuristic: Turning features into heuristics for constraint satisfaction. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, pages 183–190, Sept 2013.

[342] Jose Carlos Ortiz-Bayliss, Ender Ozcan, Andrew J. Parkes, and Hugo Terashima-Marin. A genetic programming hyper-heuristic: Turning features into heuristics for constraint satisfaction. In *2013 13th UK Workshop on Computational Intelligence (UKCI)*, pages 183–190. IEEE, 9 2013.

[343] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos. Neural networks to guide the selection of

heuristics within constraint satisfaction problems. In José Francisco Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa, Cherif Ben-Youssef Brants, and Edwin Robert Hancock, editors, *Pattern Recognition*, pages 250–259", Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[344] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos. Learning vector quantization for variable ordering in constraint satisfaction problems. *Pattern Recognition Letters*, 34(4):423–432, 2013.

[345] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos. Combine and conquer: an evolutionary hyper-heuristic approach for solving constraint satisfaction problems. *Artificial Intelligence Review*, 46(3):327–349, Oct 2016.

[346] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, and Santiago Enrique Conant-Pablos. A neuro-evolutionary hyper-heuristic approach for constraint satisfaction problems. *Cognitive Computation*, 8(3):429–441, Jun 2016.

[347] José Carlos Ortiz-Bayliss, Hugo Terashima-Marín, Peter Ross, and Santiago Enrique Conant-Pablos. Evolution of neural networks topologies and learning parameters to produce hyper-heuristics for constraint satisfaction problems. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, pages 261–262, New York, NY, USA, 2011. ACM.

[348] José Carlos Ortiz-Bayliss, Hugo Terashima-Marin, Peter Ross, Jorge Iván Fuentes-Rosado, and Manuel Valenzuela-Rendón. A neuro-evolutionary approach to produce general hyper-heuristics for the dy-

namic variable ordering in hard binary constraint satisfaction problems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1811–1812, New York, NY, USA, 2009. ACM.

[349] Jose Ortiz-Bejar, Vladimir Salgado, Mario Graff, Daniela Moctezuma, Sabino Miranda-Jimenez, and Eric Sadit Tellez. INGEOTEC at IberEval 2018 Task HaHa: microTC and EvoMSA to Detect and Score Humor in Texts. *CEUR Workshop Proceedings*, 2150:195–202, 2018.

[350] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.*, 12(1):3–23, jan 2008.

[351] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish conference on artificial intelligence and cognitive science*, pages 210–216, 2008.

[352] K. Pal, C. Saha, and S. Das. Differential evolution and offspring repair method based dynamic constrained optimization. In *International Conference on Swarm, Evolutionary, and Memetic Computing*, pages 298–309. Springer, 2013.

[353] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. A comparative review of approaches to prevent premature convergence in ga. *Applied Soft Computing*, 24:1047–1077, 2014.

[354] M. A. Panduro, C. A. Brizuela, L. I. Balderas, and D. A. Acosta. A comparison of genetic algorithms, particle swarm optimization and

the differential evolution method for the design of scannable circular antenna arrays. *Progress In Electromagnetics Research B*, 13:171–186, 2009.

[355] Marco A. Panduro, David H. Covarrubias, Carlos A. Brizuela, and Francisco R. Marante. A multi-objective approach in the linear antenna array design. *AEU - International Journal of Electronics and Communications*, 59(4):205 – 212, 2005.

[356] Panos M Pardalos, Antanas Žilinskas, and Julius Žilinskas. *Non-convex multi-objective optimization*. Springer, 2017.

[357] C. Lee S. Park and J. Kim. Topology and migration policy of fine-grained parallel evolutionary algorithms for numerical optimization. In *IEEE Congress on Evolutionary Computation*, pages 70–76, 2000.

[358] Andrew J Parkes, Ender Özcan, and Daniel Karapetyan. A software interface for supporting the application of data science to optimisation. In *International Conference on Learning and Intelligent Optimization*, pages 306–311. Springer, 2015.

[359] SI Valdez Peña, S Botello Rionda, and A Hernández Aguirre. Multi-objective topological optimization of structures. In *Numerical Modeling of Coupled Phenomena in Science and Engineering*, pages 77–86. CRC Press, 2008.

[360] Hernan Peraza-Vázquez, Aidé M. Torres-Huerta, and Abelardo Flores-Vela. Self-adaptive differential evolution hyper-heuristic with applications in process design. *Computación y Sistemas*, 20(2), 2016.

[361] Sanja Petrovic and Rong Qu. Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems. In *Pro-*

ceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Applied Technologies (KES'02), volume 82, pages 336–340, 2002.

[362] Alain Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 798–803. IEEE, 1996.

[363] M Pilát. Evolutionary multiobjective optimization: A short survey of the state-of-the-art. *Proceedings of the Contributed Papers Part I-Mathematics and Computer Sciences, WDS, Prague, Czech*, pages 1–4, 2010.

[364] N. Pillay and D. Beckedahl. Evohyp - a java toolkit for evolutionary algorithm hyper-heuristics. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2706–2713, June 2017.

[365] Nelishia Pillay and Rong Qu. *Hyper-heuristics: Theory and Applications*. Springer, 2019. ISBN 978-3-319-96513-0.

[366] Jan Poland and Andreas Zell. Main Vector Adaptation: A CMA Variant with Linear Time and Space Complexity. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1050–1055, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[367] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Published via http://lulu.com

and freely available at http://www.gp-field-guide.org.uk, 2008. ISBN 978-1-4092-0073-4.

[368] Riccardo Poli, Jonathan E. Rowe, Christopher R. Stephens, and Alden H. Wright. Allele Diffusion in Linear Genetic Programming and Variable-Length Genetic Algorithms with Subtree Crossover. In *European Conference on Genetic Programming EuroGP 2002*, pages 212–227. Springer, Berlin, Heidelberg, 2002.

[369] Riccardo Poli and Christopher R. Stephens. Taming the Complexity of Natural and Artificial Evolutionary Dynamics. In *Evolution, Complexity and Artificial Life*, pages 19–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[370] V.W. Porto and D.B. Fogel. Alternative neural network training methods [active sonar processing]. *IEEE Expert*, 10(3):16–22, June 1995.

[371] M. A. Potter and K. A. DeJong. A Cooperative Coevolutionary Approach to Function Optimization. In Y. Davidor et al., editors, *PPSN III*, volume 866 of *LNCS*, pages 249–257. Springer, 1994.

[372] R. S. Prado, R. C. P. Silva, F. G. Guimarães, and O. M. Neto. Using differential evolution for combinatorial optimization: A general approach. In *Proc. of the Int. Conf. on Systems, Man and Cybernetics*, pages 11–18. IEEE, 2010.

[373] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization.* Springer, Berlin, Germany, 2006. ISBN 978-3-540-20950-8.

[374] Cesar Puente, Gustavo Olague, Stephen V. Smith, Stephen H. Bullock, Alejandro Hinojosa-Corona, and Miguel A. González-Botello. A Genetic Programming Approach to Estimate Vegetation Cover in the Context of Soil Erosion Assessment. *Photogrammetric Engineering & Remote Sensing*, 77(4):363–376, 4 2011.

[375] B. Qian, L. Wang, R. Hu, W.L. Wang, D.X. Huang, and X. Wang. A hybrid differential evolution method for permutation flow-shop scheduling. *The Int. Journal of Advanced Manufacturing Technology*, 38(7):757–777, 2008.

[376] A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791, 2005.

[377] M. Rebaudengo and M. S. Reorda. An experimental analysis of effects of migration in paralell genetic algorithms. In *IEEE/Euromicro Workshop on Parallel and Distributed Processing*, pages 232–238, 1993.

[378] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical Report Library Translation 1122, Royal Air Force Establishment, 1965.

[379] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stutgart, 1973.

[380] I. Rechenberg. Evolutionsstrategin. In B. Schneider and U. Ranft, editors, *Simulationsmethoden in der Medizin und Biologie*, pages 83–114, Berlin, Germany, 1978. Springer-Verlag.

[381] Jon Reed, Robert Toombs, and Nils Aall Barricelli. Simulation of biological evolution and machine learning: I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing. *Journal of Theoretical Biology*, 17(3):319–342, 1967.

[382] R. G. Reynolds and W. Sverdlik. Problem solving using cultural algorithms. In *Proc. of the First IEEE Conf. on Evolutionary Computation*, volume 2, pages 645–650, 1994.

[383] John R. Rice. The algorithm selection problem. In Morris Rubinoff and Marshall C. Yovits, editors, *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[384] H. Richter. Fitness landscapes: From evolutionary biology to evolutionary computation. In H. Richter and A. Engelbrecht, editors, *Recent Advances in the Theory and Application of Fitness Landscapes, Emergence, Complexity and Computation*, pages 3–31. Springer, Berlin, Germany, 2014.

[385] J. Neal Richter, Alden Wright, and John Paxton. Ignoble Trails - Where Crossover Is Provably Harmful. In Günter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature – PPSN X*, Lecture Notes in Computer Science Vol. 5199, pages 92–101. Springer Berlin Heidelberg, 2008.

[386] Rafael Rivera-Lopez and Juana Canul-Reich. Construction of Near-Optimal Axis-Parallel Decision Trees Using a Differential-Evolution-Based Approach. *IEEE Access*, 6:5548–5563, 1 January 2018.

[387] Katya Rodriguez and Rosalva Mendoza. A Matlab Genetic Programming Approach to Topographic Mesh Surface Generation. In *Engineering Education and Research Using MATLAB*. InTech, 10 2011.

[388] Lino Rodriguez-Coayahuitl, Alicia Morales-Reyes, and Hugo Jair Escalante. Structurally Layered Representation Learning: Towards Deep Learning Through Genetic Programming. In *European Conference on Genetic Programming EuroGP 2018*, pages 271–288. Springer, Cham, 2018.

[389] K. Rodriguez-Vazquez, M. L. Arganis-Juarez, C. Cruickshank-Villanueva, and R. Domínguez-Mora. Rainfall–runoff modelling using genetic programming. *Journal of Hydroinformatics*, 14(1):108–121, 1 2012.

[390] K. Rodriguez-Vazquez and P.J. Fleming. Evolution of mathematical models of chaotic systems based on multiobjective genetic programming. *Knowledge and Information Systems*, 8(2):235–256, August 2005.

[391] K. Rodriguez-Vazquez, C.M. Fonseca, and P.J. Fleming. Identifying the Structure of NonLinear Dynamic Systems Using Multiobjective Genetic Programming. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34(4):531–545, 7 2004.

[392] Katya Rodríguez-Vázquez. Sunspots modelling: comparison of GP approaches. In *GECCO'13 Companion - Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1745–1746, Amsterdam, The Netherlands, July 6-10 2013. ACM Press. ISBN 978-1-4503-1964-5.

[393] Cynthia A. Rodríguez Villalobos and Carlos A. Coello Coello. A New Multi-Objective Evolutionary Algorithm Based on a Performance Assessment Indicator. In *2012 Genetic and Evolutionary Computation Conference (GECCO'2012)*, pages 505–512, Philadelphia, USA, July 2012. ACM Press. ISBN: 978-1-4503-1177-9.

[394] Alejandro Rosales-Pérez, Jesus A. Gonzalez, Carlos A. Coello Coello, Carlos A. Reyes-Garcia, and Hugo Jair Escalante. EMOPG+FS: Evolutionary multi-objective prototype generation and feature selection. *Intelligent Data Analysis*, 20(s1):S37–S51, 2016.

[395] Peter Ross, Emma Hart, and Dave Corne. Some observations about ga-based exam timetabling. In Edmund Burke and Michael Carter, editors, *Practice and Theory of Automated Timetabling II*, pages 115–129, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[396] A.E. Ruano, P.J. Fleming, C. Teixeira, K. Rodriguez-Vazquez, and C.M. Fonseca. Nonlinear identification of aircraft gas-turbine dynamics. *Neurocomputing*, 55(3-4):551–579, 10 2003.

[397] Günter Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Transactions on Neural Networks*, 5:96–101, January 1994.

[398] C. Ryan, J. J. Collins, and M. O. Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf et al., editors, *EuroGP'98*, volume 1291 of *LNCS*, pages 83–96. Springer, 1998.

[399] Ângela AR Sá, Adriano O Andrade, Alcimar B Soares, and Slawomir J Nasuto. Exploration vs. exploitation in differential evolution. In *AISB*

2008 Convention Communication, Interaction and Social Intelligence, volume 1, page 57, 2008.

[400] R. A. Sarker, H. A. Abbass, and C. S. Newton. Solving two multi-objective optimization problems using evolutionary algorithm. In *Computational Intelligence in Control*, pages 218–233. IGI Global, 2003.

[401] R. A. Sarker and M. F. A. Kazi. Population size, search space and quality of solution: an experimental study. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 3, pages 2011–2018, Dec 2003.

[402] J. Sarma and K. De Jong. An analysis of the effects of neighbourhood size and shape on local selection algorithms. In *Parallel Problem Solving from Nature*, pages 236–244. Springer, 1996.

[403] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[404] Jason R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, May 1995.

[405] Oliver Schütze, Xavier Esquivel, Adriana Lara, and Carlos A. Coello Coello. Using the Averaged Hausdorff Distance as a Performance Measure in Evolutionary Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 16(4):504–522, August 2012.

[406] Hans-Paul Schwefel. *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*. Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger–Institute for Hydrodynamics, March 1965.

[407] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[408] Eduardo Segredo, Carlos Segura, and Coromoto León. A Multiobjectivised Memetic Algorithm for the Frequency Assignment Problem. In *2011 IEEE Congress on Evolutionary Computation (CEC'2011)*, pages 1132–1139, New Orleans, Louisiana, USA, 5–8 June 2011. IEEE Service Center.

[409] Carlos Segura, Carlos A. Coello Coello, Gara Miranda, and Coromoto Leon. Using Multi-objective Evolutionary Algorithms for Single-objective Optimization. *4OR-A Quarterly Journal of Operations Research*, 11(3):201–228, September 2013.

[410] Carlos Segura, Carlos A. Coello Coello, Gara Miranda, and Coromoto Leon. Using Multi-Objective Evolutionary Algorithms for Single-Objective Constrained and Unconstrained Optimization. *Annals of Operations Research*, 240(1):217–250, May 2016.

[411] Carlos Segura, Carlos A. Coello Coello, Eduardo Segredo, and Arturo Hernández Aguirre. A Novel Diversity-Based Replacement Strategy for Evolutionary Algorithms. *IEEE Transactions on Cybernetics*, 46(12):3233–3246, December 2016.

[412] Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances*

and historical development of vector optimization, pages 222–232. Springer, 1987.

[413] Ramiro Serrato Paniagua, Juan J. Flores Romero, and Carlos A. Coello Coello. A Genetic Representation for Dynamic System Qualitative Models on Genetic Programming: A Gene Expression Programming Approach. In *MICAI 2007: Advances in Artificial Intelligence*, pages 30–40. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[414] Mohammad Javad Shafiee, Akshaya Mishra, and Alexander Wong. Deep learning with darwin: evolutionary synthesis of deep neural networks. *Neural Processing Letters*, 48(1):603–613, 2018.

[415] Alexei A. Sharov. Logistic model @ONLINE, June 2010.

[416] Ye shi Jiang, Ying Lin, Jing-Jing Li, Zheng jia Dai, Jun Zhang, and Xinglin Zhang. A novel genetic algorithm for constructing uniform test forms of cognitive diagnostic models. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 5195–5200, Vancouver, British Columbia, Canada, 24-29 July 2016. ISBN 978-1-5090-0624-3.

[417] Kevin Sim and Emma Hart. A combined generative and selective hyper-heuristic for the vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 1093–1100, New York, NY, USA, 2016. ACM.

[418] D. Simoncini, S. Verel, P. Collard, and M. Clergue. Anisotropic selection in cellular genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06)*, pages 559–566, 2006.

[419] D. Simoncini, S. Verel, P. Collard, and M. Clergue. Centric selection: a way to tune the exploration/exploitation trade-off. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 891–898, 2009.

[420] J. A. Soria-Alcaraz, A. Espinal, and M. A. Sotelo-Figueroa. Evolvability metric estimation by a parallel perceptron for on-line selection hyper-heuristics. *IEEE Access*, 5:7055–7063, 2017.

[421] Jorge A. Soria-Alcaraz, J. Martin Carpio-Valadez, and Hugo Terashima-Marin. Academic Timetabling Design Using Hyper-Heuristics. In *Soft Computing for Intelligent Control and Mobile Robotics*, pages 43–56. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-15534-5.

[422] Jorge A. Soria-Alcaraz, Gabriela Ochoa, Marco A. Sotelo-Figeroa, and Edmund K. Burke. A methodology for determining an effective subset of heuristics in selection hyper-heuristics. *European Journal of Operational Research*, 260(3):972–983, 2017.

[423] Jorge A. Soria-Alcaraz, Gabriela Ochoa, Marco A. Sotelo-Figueroa, Martín Carpio, and Hector Puga. *Iterated VND Versus Hyper-heuristics: Effective and General Approaches to Course Timetabling*, pages 687–700. Springer International Publishing, 2017.

[424] Jorge A. Soria-Alcaraz, Ender Özcan, Jerry Swan, Graham Kendall, and Martin Carpio. Iterated local search using an add and delete hyper-heuristic for university course timetabling. *Applied Soft Computing*, 40:581–593, 2016.

[425] Alejandro Sosa-Ascencio, Gabriela Ochoa, Hugo Terashima-Marin, and Santiago Enrique Conant-Pablos. Grammar-based generation of variable-selection heuristics for constraint satisfaction problems. *Genetic Programming and Evolvable Machines*, 17(2):119–144, 6 2016.

[426] Alejandro Sosa-Ascencio, Gabriela Ochoa, Hugo Terashima-Marin, and Santiago Enrique Conant-Pablos. Grammar-based generation of variable-selection heuristics for constraint satisfaction problems. *Genetic Programming and Evolvable Machines*, 17(2):119–144, Jun 2016.

[427] Alejandro Sosa-Ascencio, Hugo Terashima-Marin, and Manuel Valenzuela-Rendon. Grammar-based Genetic Programming for evolving variable ordering heuristics. In *2013 IEEE Congress on Evolutionary Computation*, pages 1154–1161. IEEE, 6 2013.

[428] M. A. Sotelo-Figueroa, Arturo Hernández-Aguirre, Andrés Espinal, J. A. Soria-Alcaraz, and Janet Ortiz-López. Symbolic Regression by Means of Grammatical Evolution with Estimation Distribution Algorithms as Search Engine. In *Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications*, pages 169–177. Springer, Cham, 2018.

[429] M. A. Sotelo-Figueroa, H. J. Puga Soberanes, J. Martín Carpio, H. J. Fraire Huacuja, L. C. Reyes, and J. A. Soria-Alcaraz. Evolving and reusing bin packing heuristic through grammatical differential evolution. In *World Congress on Nature and Biologically Inspired Computing*, pages 92–98, Aug 2013.

[430] Marco Aurelio Sotelo-Figueroa, Héctor José Puga Soberanes, Juan Martín Carpio, Héctor J. Fraire Huacuja, Laura Cruz Reyes, and Jorge Alberto Soria-Alcaraz. Improving the bin packing heuristic

through grammatical evolution based on swarm intelligence. *Mathematical Problems in Engineering*, 2014, 2014.

[431] P. Spiessens and B. Manderick. A massively parallel genetic algorithm. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 279–286. Morgan Kaufmann, 1991.

[432] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[433] Christopher R. Stephens and Riccardo Poli. EC Theory — "in Theory". In *Frontiers of Evolutionary Computation*, pages 129–155. Kluwer Academic Publishers, Boston, 2004.

[434] Ralph E Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application.* Krieger Publishing Company, Melbourne, Florida, 1986.

[435] R. Storn and K. Price. Differential Evolution – A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, 1995.

[436] Rainer Storn and Kenneth Price. Differential Evolution–A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[437] Dirk Sudholt. Parallel evolutionary algorithms. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, chapter 46, pages 931–957. Springer-Verlag Berlin Heidelberg, Switzerland, 2015.

[438] Jerry Swan, Ender Özcan, and Graham Kendall. Hyperion – a recursive hyper-heuristic framework. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 616–630, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[439] Aleksandra Swiercz. Hyper-heuristics and metaheuristics for selected bio-inspired combinatorial optimization problems. In Javier Del Ser Lorente, editor, *Heuristics and Hyper-Heuristics*, chapter 1. IntechOpen, Rijeka, 2017.

[440] E.G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009.

[441] Kay Chen Tan, Eik Fun Khor, and Tong Heng Lee. *Multiobjective evolutionary algorithms and applications*. Springer Science & Business Media, 2006.

[442] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for Differential Evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78, 2013.

[443] L. Tang, Y. Dong, and J. Liu. Differential Evolution With an Individual-Dependent Mechanism. *IEEE Transactions on Evolutionary Computation*, 19(4):560–574, Aug 2015.

[444] M.F. Tasgetiren, M. Sevkli, Y.C. Liang, and G. Gencyilmaz. Particle swarm optimization algorithm for single machine total weighted tardiness problem. In *Proceedings of the Congress on Evolutionary Computation (CEC2004)*, pages 1412–1419, Portland, OR, USA, 2004. IEEE.

[445] M.F. Tasgetiren, P.N. Suganthan, and Q.K. Pan. An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem. *Applied Mathematics and Computation*, 215(9):3356–3368, 2010.

[446] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing*, 10(8):673–686, Jun 2006.

[447] H. Terashima-Marín, C. J. Farías Zárate, P. Ross, and M. Valenzuela-Rendón. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 591–598, New York, NY, USA, 2006. ACM.

[448] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 571–578, New York, NY, USA, 2008. ACM.

[449] H. Terashima-Marín, P. Ross, C. J. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179(1):369–392, Sep 2010.

[450] Hugo Terashima-Marín, Peter Ross, and Manuel Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, GECCO'99, pages 635–642, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[451] M. Tomassini. *Spatially Structured Evolutionary Algorithms, Artificial Evolution in Space and Time*. Springer. Series: Natural Computing Series, 2005.

[452] Marco Tomassini. Parallel and distributed evolutionary algorithms: A review, 1999.

[453] Heike Trautmann, Günter Rudolph, Kathrin Klamroth, Oliver Schütze, Margaret Wiecek, Yaochu Jin, and Christian Grimme, editors. *Evolutionary Multi-Criterion Optimization, 9th International Conference, EMO 2017*. Springer. Lecture Notes in Computer Science Vol. 10173, Münster, Germany, March 19-22 2017. ISBN 978-3-319-54156-3.

[454] Leonardo Trujillo, Pierrick Legrand, Gustavo Olague, and Jacques Lévy-Véhel. Evolving estimators of the pointwise Hölder exponent with Genetic Programming. *Information Sciences*, 209:61–79, 11 2012.

[455] Leonardo Trujillo, Luis Muñoz, Edgar Galván-López, and Sara Silva. neat Genetic Programming: Controlling bloat naturally. *Information Sciences*, 333:21–43, 3 2016.

[456] Leonardo Trujillo and Gustavo Olague. Automated Design of Image Operators that Detect Interest Points. *Evolutionary Computation*, 16(4):483–507, 12 2008.

[457] A. M. Turing. Intelligent machinery. Technical report, National Physical Laboratory, Teddington, UK, 1948.

[458] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[459] Tea Tušar and Bogdan Filipič. Differential Evolution Versus Genetic Algorithms in Multiobjective Optimization. In *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007*, pages 257–271, Matshushima, Japan, March 2007. Springer. Lecture Notes in Computer Science Vol. 4403.

[460] R. Tyasnurita, E. Özcan, and R. John. Learning heuristic selection using a time delay neural network for open vehicle routing. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1474–1481, June 2017.

[461] Raras Tyasnurita, Ender Özcan, Shahriar Asta, and Robert John. Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing. In *Proceedings of the 15th Annual Workshop on Computational Intelligence (UKCI)*, 2015.

[462] José María Valencia-Ramírez, Mario Graff, Hugo Jair Escalante, and Jaime Cerda-Jacobo. An iterative genetic programming approach to prototype generation. *Genetic Programming and Evolvable Machines*, 18(2):123–147, 6 2017.

[463] David A. van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

[464] Leonardo Vanneschi. An Introduction to Geometric Semantic Genetic Programming. In Oliver Schütze, Leonardo Trujillo, Pierrick Legrand, and Yazmin Maldonado, editors, *NEO 2015*, pages 3–42. Springer, Cham, 2017.

[465] Leonardo Vanneschi, Illya Bakurov, and Mauro Castelli. An initialization technique for geometric semantic GP based on demes evolution and despeciation. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 113–120. IEEE, 6 2017.

[466] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214, 6 2014.

[467] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, 45(3):35:1–35:33, July 2013.

[468] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Computing Surveys*, 45(3):35:1–35:33, July 2013.

[469] R. V. Veiga, J. M. de Freitas, H. S. Bernardino, H. J. C. Barbosa, and N. M. Alcântara-Neves. Using grammar-based genetic programming to determine characteristics of multiple infections and environmental factors in the development of allergies and asthma. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1604–1611, May 2015.

[470] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithm Test Suites. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 351–357, San Antonio, Texas, USA, 1999. ACM Press.

[471] David A. Van Veldhuizen and Gary B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *2000 IEEE*

Congress on Evolutionary Computation, volume 1, pages 204–211, Piscataway, New Jersey, July 2000. IEEE Service Center.

[472] J. Vesterstrøm and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Congress on Evolutionary Computation, 2004. CEC2004*, volume 2, pages 1980–1987. IEEE, 2004.

[473] Pablo Vidal and Enrique Alba. Cellular Genetic Algorithm on Graphic Processing Units. In Juan R. González, David Alejandro Pelta, Carlos Cruz, Germán Terrazas, and Natalio Krasnogor, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, pages 223–232. Springer, Berlin, Germany, 2010.

[474] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489, 2013.

[475] Miguel G. Villarreal-Cervantes, Efrén Mezura-Montes, and José-Yair Guzmán-Gaspar. Differential evolution based adaptation for the direct current motor velocity control parameters. *Mathematics and Computers in Simulation*, 150:122–141, 2018.

[476] Juan Villegas-Cortez, Gustavo Olague, Humberto Sossa, and Carlos Avilés. Evolutionary Associative Memories through Genetic Programming. In *Parallel Architectures and Bioinspired Algorithms*, pages 171–188. Springer, Berlin, Heidelberg, 2012.

[477] José Carlos Villela Tinoco and Carlos A. Coello Coello. *hyp*DE: A hyper-Heuristic Based on Differential Evolution for Solving Constrained Optimization Problems. In Oliver Schütze, Carlos A. Coello Coello, Alexandru-Adrian Tantar, Emilia Tantar, Pascal Bouvry, Pierre Del Moral, and Pierrick Legrand, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, pages 267–282. Springer, Advances in Intelligent Systems and Computing Vol. 175, Berlin, Germany, 2012. ISBN 978-3-642-31519-0.

[478] H-M Voigt, Heinz Mühlenbein, and Dragan Cvetkovic. Fuzzy recombination for the breeder genetic algorithm. In *Proc. Sixth Int. Conf. on Genetic Algorithms*. Citeseer, 1995.

[479] Christian von Lücken, Benjamin Baran, and Carlos Brizuela. A survey on multi-objective evolutionary algorithms for many-objective problems. *Computational Optimization and Applications*, 58(3):707–756, July 2014.

[480] Christian von Lücken, Carlos A. Brizuela, and Benjamín Barán. An overview on evolutionary algorithms for many-objective optimization problems. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(1), January/February 2019.

[481] Ivan Voutchkov and Andy Keane. Multi-Objective Optimization Using Surrogates. In Yoel Tenne and Chi-Keong Goh, editors, *Computational Intelligence in Optimization. Applications and Implementations*, pages 155–175. Springer, Berlin, Germany, 2010. ISBN 978-3-642-12774-8.

[482] María Vázquez-Ojeda, Juan Gabriel Segovia-Hernández, Salvador Hernández, Arturo Hernández-Aguirre, and Anton Alexandra Kiss. Optimization of an Ethanol Dehydration Process Using Differential Evolution Algorithm. *Computer Aided Chemical Engineering*, 32:217–222, 2013.

[483] Handing Wang, Yaochu Jin, Chaoli Sun, and John Doherty. Offline Data-Driven Evolutionary Optimization Using Selective Surrogate Ensembles. *IEEE Transactions on Evolutionary Computation*, 23(2):203–216, April 2019.

[484] Jianrong Wang, Tianbiao Yu, and Wanshan Wang. Integrating Analytic Hierarchy Process and Genetic Algorithm for Aircraft Engine Maintenance Scheduling Problem. In George Q. Huang, K.L. Mak, and Paul G. Maropoulos, editors, *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, pages 897–915. Springer, Berlin, Germany, 2010.

[485] S. Wang, Y. Li, H. Yang, and H. Liu. Self-adaptive differential evolution algorithm with improved mutation strategy. *Soft Computing*, 22(10):3433–3447, May 2018.

[486] Y. Wang, Z. Cai, and Q. Zhang. Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, Feb 2011.

[487] Shinya Watanabe and Kazutoshi Sakakibara. A Multiobjectivization Approach for Vehicle Routing Problems. In *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007*, pages 660–672, Matshushima, Japan, March 2007. Springer. Lecture Notes in Computer Science Vol. 4403.

[488] Roger Weinberg. *Computer Simulation of a Living Cell*. PhD thesis, University of Michigan, Ann Arbor, Michigan, USA, 1970.

[489] Shimon Whiteson and Peter Stone. On-Line Evolutionary Computation for Reinforcement Learning in Stochastic Domains. In *2006 Genetic and Evolutionary Computation Conference (GECCO'2006)*, pages 1577–1584, Seattle, Washington, USA, July 2006. ACM Press. ISBN 1-59593-186-4.

[490] Darrell Whitley and Joan Kauth. GENITOR: A different genetic algorithm. Technical report, Colorado State University, Department of Computer Science, 1988.

[491] Paweł Widera, Jonathan M Garibaldi, and Natalio Krasnogor. GP challenge: evolving energy function for protein structure prediction. *Genetic Programming and Evolvable Machines*, 11(1):61–88, March 2010.

[492] D.H. Wolpert and W.G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[493] S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *VI International Congress of Genetics*, volume 1, pages 356–366, 1932.

[494] Yi Xiang and Yuren Zhou. A dynamic multi-colony artificial bee colony algorithm for multi-objective optimization. *Applied Soft Computing*, 35:766 – 785, 2015.

[495] Jing Xie, Yi Mei, Andreas T. Ernst, Xiaodong Li, and Andy Song. A genetic programming-based hyper-heuristic approach for storage

location assignment problem. In *2014 IEEE Congress on Evolutionary Computation (CEC'2014)*, pages 3000–3007, Beijing, China, 6-11 July 2014. IEEE Press. ISBN 978-1-4799-1488-3.

[496] Bing Xue, Mengjie Zhang, Will N Browne, and Xin Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, 2016.

[497] Yukiko Yamamoto, Setsuo Tsuruta, Takayuki Muranushi, Yuko Hada-Muranushi, Syoji Kobashi, Yoshiyuki Mizuno, and Rainer Knauf. Solar flare prediction by SVM integrated GA. In *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*, pages 4127–4134, Vancouver, British Columbia, Canada, 24-29 July 2016. ISBN 978-1-5090-0624-3.

[498] Ming Yang, Changhe Li, Zhihua Cai, and Jing Guan. Differential Evolution with Auto-Enhanced Population Diversity. *IEEE Transactions on Cybernetics*, 45(2):302–315, February 2015.

[499] J. Yao, W. Wang, Z. Li, Y. Lei, and Q. Li. Tactics Exploration Framework based on Genetic Programming. *International Journal of Computational Intelligence Systems*, 10(1):804–814, 2017.

[500] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transasctions on Evolutionary Computation*, 3(2):82–102, July 1999.

[501] Wei-Jie Yu, Meie Shen, Wei-Neng Chen, Zhi-Hui Zhan, Yue-Jiao Gong, Ying Lin, Ou Liu, and Jun Zhang. Differential evolution with two-level parameter adaptation. *IEEE Transactions on Cybernetics*, 44(7):1080–1099, 2014.

[502] Xiaobing Yu, Yiqun Lu, Xuming Wang, Xiang Luo, and Mei Cai. An effective improved differential evolution algorithm to solve constrained optimization problems. *Soft Computing*, 23(7):2409–2427, April 2019.

[503] Shiu Yin Yuen and Chi Kin Chow. A Genetic Algorithm That Adaptively Mutates and Never Revisits. *IEEE Transactions on Evolutionary Computation*, 13(2):454–472, April 2009.

[504] Emigdio Z-Flores, Mohamed Abatal, Ali Bassam, Leonardo Trujillo, Perla Juárez-Smith, and Youness El Hamzaoui. Modeling the adsorption of phenols and nitrophenols by activated carbon using genetic programming. *Journal of Cleaner Production*, 161:860–870, 10 September 2017.

[505] Lofti A. Zadeh. Optimality and Nonscalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control*, AC-8(1):59–60, 1963.

[506] Daniela Zaharie. Control of population diversity and adaptation in differential evolution algorithms. In *Proceedings of the 9th International Mendel Conference on Soft Computing*, volume 9, pages 41–46, 2003.

[507] Saul Zapotecas-Martinez, Antonio Lopez-Jaimes, and Abel Garcia-Najera. LIBEA: A Lebesgue Indicator-Based Evolutionary Algorithm for Multi-Objective Optimization. *Swarm and Evolutionary Computation*, 44:404–419, February 2019.

[508] Milan Zeleny. *Linear Multiobjective Programming*. Springer. Lecture Notes in Economics and Mathematical Systems, Berlin, Germany, 1974. ISBN 978-3-540-06639-2.

[509] Jingqiao Zhang and Arthur C. Sanderson. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In *2007 IEEE Congress on Evolutionary Computation (CEC'2007)*, pages 2251–2258, Singapore, 25-28 September 2007. IEEE Press. ISBN 978-1-4244-1339-3.

[510] Jun Zhang, Zhi-hui Zhan, Ying Lin, Ni Chen, Yue-jiao Gong, Jing-hui Zhong, Henry SH Chung, Yun Li, and Yu-hui Shi. Evolutionary computation meets machine learning: A survey. *IEEE Computational Intelligence Magazine*, 6(4):68–75, 2011.

[511] M. Zhang, W. Luo, and X. Wang. Differential evolution with dynamic stochastic selection for constrained optimization. *Information Sciences*, 178(15):3043–3074, 2008.

[512] Qingfu Zhang and Hui Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, December 2007.

[513] Qingfu Zhang, Wudong Liu, and Hui Li. The Performance of a New Version of MOEA/D on CEC09 Unconstrained MOP Test Instances. In *2009 IEEE Congress on Evolutionary Computation (CEC'2009)*, pages 203–208, Trondheim, Norway, May 2009. IEEE Press.

[514] Li Zhao, Chao jiao Sun, Xian-chi, and Bing xu Zhou. Differential Evolution with strategy of improved population diversity. In *2016 35th Chinese Control Conference (CCC)*, pages 2784–2787, Chengdu, China, July 27-29 2016. IEEE Press. ISBN 978-1-5090-0910-7.

[515] Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Swagatam Das. Self-adaptive differential evolution with multi-trajectory

search for large-scale optimization. *Soft Computing*, 15(11):2175–2185, November 2011.

[516] Jun Zheng, Chao Lu, and Liang Gao. Multi-objective cellular particle swarm optimization for wellbore trajectory design. *Applied Soft Computing*, 77:106–117, April 2019.

[517] Qingling Zhu, Qiuzhen Lin, Zhihua Du, Zhengping Liang, Wenjun Wang, Zexuan Zhu, Jianyong Chen, Peizhi Huang, and Zhong Ming. A novel adaptive hybrid crossover operator for multiobjective evolutionary algorithm. *Information Sciences*, 345:177–198, June 1 2016.

[518] Karin Zielinski and Rainer Laur. Stopping Criteria for Differential Evolution in Constrained Single-Objective Optimization. In Uday K. Chakraborty, editor, *Advances in differential evolution*, pages 111–138. Springer, Berlin, Germany, 2008. ISBN 978-3-540-68827-3.

[519] Karin Zielinski, Petra Weitkemper, Rainer Laur, and Karl-Dirk Kammeyer. Parameter Study for Differential Evolution Using a Power Allocation Problem Including Interference Cancellation. In *2006 IEEE International Conference on Evolutionary Computation (CEC'2016)*, pages 1857–1864, Vancouver, British Columbia, Canada, July 16-21 2006. IEEE Press. ISBN 0-7803-9487-9.

[520] Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos Coello Coello, and David Corne, editors. *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001.* Springer. Lecture Notes in Computer Science Vol. 1993, Zürich, Switzerland, March 7-9 2001. ISBN 978-3-540-41745-3.

[521] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality Assessment of Pareto Set Approximations. In *Multiobjective Optimization. Interactive and Evolutionary Approaches*, pages 373–404. Springer. Lecture Notes in Computer Science Vol. 5252, Berlin, Germany, 2008.

[522] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In *Metaheuristics for Multiobjective Optimisation*, pages 3–37, Berlin, 2004. Springer. Lecture Notes in Economics and Mathematical Systems Vol. 535.

[523] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2001.

[524] Eckart Zitzler and Lothar Thiele. Multiobjective Optimization Using Evolutionary Algorithms—A Comparative Study. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature V*, pages 292–301, Amsterdam, The Netherlands, September 1998. Springer-Verlag. Lecture Notes in Computer Science No. 1498.

[525] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

Computación Evolutiva
se terminó el 30 de septiembre de 2019.
A partir del 1 de diciembre de 2019 está disponible en formato PDF
en la página web de la Academia Mexicana de Computación:
http://www.amexcomp.mx