Facultad de Ciencias

Estrategias evolutivas para el análisis y resolución de problemas de optimización MaOP

(Evolutionary strategies for analysis and resolution of MaOP optimization problems)

Trabajo de Fin de Máster
para acceder al

MÁSTER UNIVERSITARIO EN MATEMÁTICAS Y COMPUTACIÓN

Autor: Giusseppe Benito Bervis Quintero

Director: Ángel Cobo Ortega Ph.D.

Junio - 2022

Resumen: La optimización es un área muy importante dentro del quehacer humano, tanto por sus implicaciones teóricas, como por sus aplicaciones. Un tipo particular de problemas de esta rama de la ciencia son los problemas de optimización multiobjetivos (MOOP, por sus siglas en inglés); estos consisten en optimizar más de una función a la vez. Así, en este Trabajo de Fin de Máster se pretende, en primer lugar, hacer una revisión de las principales técnicas de resolución de los MOOP; para posteriormente ejemplificar una de las principales dificultades de estos problemas: la conflictividad entre objetivos. En los MOOP los objetivos casi siempre entran en conflictos; es decir si se mejora el resultado en uno, se empeora en otro u otros. Cabe destacar que esta dificultad aumenta cuando se tienen más de tres objetivos, a estos se les llama *«Many-Objetives Problems»* (MaOP). Ante tal situación, una respuesta a dicha problemática son las estrategias de reducción de objetivos, estas buscan generar un problema con un subconjunto propio del conjunto de objetivos original (menor conflictividad) que sea más sencillo de resolver, pero que aporte luces respecto de la solución del problema inicial. Por otro lado, una vez que se cuenta con un número razonable de objetivos, es necesario disponer de metodologías de aproximación del conjunto de soluciones eficientes, soluciones que no son dominadas por ninguna otra y que, por tanto, todas ellas pueden resultar válidas como solución al problema propuesto. En este trabajo se analiza, también, el uso de técnicas metaheurísticas poblacionales para aproximar dichas soluciones.

Palabras claves: Optimización Multiobjetivos, Problemas Manyobjetivos, Metaheurísticas, Reducción de objetivos.

Abstract: Optimization is a very important area of human endeavor, both for its theoretical implications and its applications. A particular type of problems in this branch of science are Multi-Objective Optimization Problems (MOOP); these consist of optimizing more than one function at a time. Thus, in this Master's Thesis we intend, first of all, to review the main techniques for solving MOOPs, and then to exemplify one of the main difficulties of these problems: the conflict between objectives. In MOOPs, the objectives almost always conflict; that is, if the result is improved in one, it worsens in another or others. It should be noted that this difficulty increases when there are more than three objectives; these are called "Many-Objective Problems" (MaOP). In this situation, one response to this problem is the objective reduction strategies, which seek to generate a problem with a subset of the original set of objectives (less conflict) that is simpler to solve, but that provides information regarding the solution of the initial problem. On the other hand, once a reasonable number of objectives is available, it is necessary to have approximation methodologies of the set of efficient solutions, solutions that are not dominated by any other and that, therefore, all of them can be valid as a solution to the proposed problem. This thesis also analyzes the use of population metaheuristic techniques to approximate such solutions.

Palabras claves: Multi-objetive Optimization, Many-objetives Problems, Metaheurístics, Objective Reduction.

Contenido

Capítulo 1

Introducción a la Optimización Multiobjetivo

En la vida cotidiana, sobre todo en el sector industrial, muchas veces se presentan situaciones en las que se quiere optimizar algún factor decisivo en determinado proceso; por ejemplo, minimizar costos, horas de trabajo de máquinas, número de trabajadores o maximizar beneficios, entre otros. A este tipo de problemas se les conoce como **problemas de optimización**. En la resolución de estos problemas, la matemática juega un papel importante; ya que estos pueden ser modelados con objetos matemáticos como funciones, sistemas de desigualdades, entre otros.

En la mayoría de los casos de problemas de optimización se busca optimizar (maximizar o minimizar) una única función **(optimización mono-objetivo)**. Sin embargo, en la vida real muchas veces se presenta la necesidad de optimizar varias funciones simultáneamente **(optimización multiobjetivo, «OMO»)**. La meta, entonces, en la OMO es encontrar los valores adecuados para que las variables de decisión optimicen todas las funciones objetivos a la vez y satisfagan las restricciones dadas (si las hay).

El principal inconveniente en este tipo de problemas es que las funciones, casi siempre, entran en conflicto; es decir, si se mejora el resultado en una función, pueda que se empeore en otra (u otras); esto conlleva que no se puedan alcanzar conjuntamente los óptimos de todas las funciones **(situación de conflicto)**. Debido a lo anterior, lo que se busca es lograr los mejores compromisos entre los resultados de las funciones objetivos.

Para ilustrar lo expuesto en el párrafo anterior, se considerará el proceso de decisión que se puede hacer cuando se va a comprar un ordenador.

Ejemplo 1.1. *Considérense 5 ordenadores: A, B, C, D, E; dispuestos, respecto al precio, de menor a mayor. Para simplificar, se tomarán en consideración dos características para realizar la compra; a saber, el precio y capacidad de la memoria RAM. El problema anterior, con valores concretos, se puede ver en la Figura 1.1.*

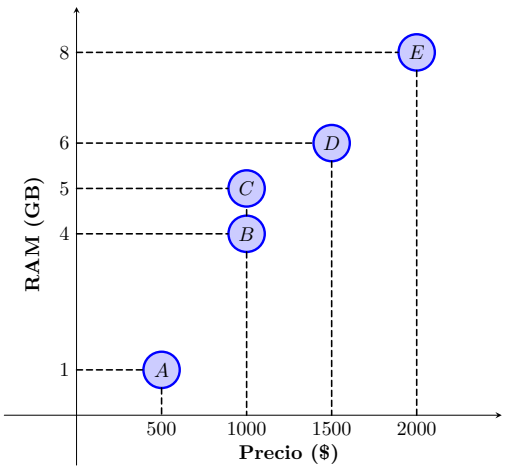


Figura 1.1: Relación precio - memoria de los ordenadores.

Lo que se desearía, lógicamente, es conseguir la mayor cantidad de memoria RAM al menor precio posible

(maximizar memoria y minimizar costo). Evidentemente que si se toma en cuenta solo el precio, la mejor opción sería el ordenador A (el menor precio); sin embargo, pueda que para algunos el dinero no sea problema y su mejor opción sea el ordenador E (el de mayor memoria). Entre estas dos soluciones extremas, tenemos otras tres que tienen distintos niveles para las dos características que se están evaluando. Asimismo, nótese que si nos tocara decidir entre la opción B y la C, la opción a elegir sería la C, ya que posee el mismo precio que B, pero la memoria es mayor. Entre las distintas soluciones hay que notar que la mejora de un objetivo implica cierto sacrificio (empeoramiento) en el otro.

A pesar de la simplicidad y pequeñez del ejemplo anterior, nos puede dar un atisbo de la utilidad que tiene esta rama de las matemáticas y de la dificultad que se presenta en los MOOP.

1.1. Importancia de la Optimización Multiobjetivo

Como ya se aseveró, los problemas con los que comúnmente se tienen que lidiar en la vida real implican la optimización de múltiples funciones. Resolver estos problemas, de manera eficiente, implica el mejor aprovechamiento de los recursos que se tengan a disposición y, como es evidente, menor desperdicio de estos. Una característica notoria de este tipo de problemas, también comentada en líneas anteriores, es la conflictividad que se presenta, muchas veces, entre los múltiples objetivos; cuando se mejora en uno, pueda que se empeore en otros. En la situación actual, de pandemia, se presentan situaciones en las que hay factores que entran en conflictos; la economía y el decreto total y estricto de cuarentena indefinida, por ejemplo. Ante tal escenario, tenemos que: si se cesa la actividad económico laboral, se podría cumplir el régimen de encierro, pero las implicaciones económicas de tales medidas, a mediano y largo plazo, serían catastróficas para cualquier nación; por otro lado, si se hiciera caso omiso a las medidas sanitarias, dando prioridad a la economía, el impacto en la población sería la pérdida de vidas humanas. En este contexto, entonces, es necesario el desarrollo e implementación de modelos multiobjetivos, que nos ayuden a determinar posibles soluciones, frente a escenarios tan conflictivos como el actual.

El contexto actual nos enfrenta a la necesidad de valernos de la OMO, sin embargo esta es solo una de las posibles situaciones en donde esta herramienta puede ser aplicada. A continuación se presentarán, sin ánimos de ser exhaustivos en cada una de ellas, algunas, pocas, aplicaciones de esta área de estudio.

- **Aplicaciones en economía:** Dentro el ámbito económico hay situaciones en las que se desea cumplir con ciertos objetivos; como maximizar beneficios, minimizar costos y número de trabajadores, por ejemplo. Así, la OMO es una excelente herramienta para tratar de lograr tales propósitos.
    - **Cadenas de suministro de petróleo y gas**: Un factor importante dentro de la economía global es la producción de hidrocarburos, ya que de estos dependen buena parte del quehacer humano; el transporte, la energía eléctrica y la industria son algunos ejemplos que dependen en gran medida de este rubro. Así pues, la optimización en este sector puede ser gran interés para los productores. Un modelo que pretende ayudar a la optimización de este recurso lo encontramos en (Attia et al., 2019). El modelo presentado en dicho artículo pretende minimizar la tasa de consumo, así como el costo total y maximizar los ingresos, satisfaciendo las restricciones de la demanda.
    - **Minimización de las emisiones de CO_2**: La globalización ha causado que los modos en que las naciones realizaban transacciones comerciales haya cambiado. Las importaciones y exportaciones, hoy, se realizan a escalas macro. Esto ha tenido un impacto negativo sobre el ecosistema mundial, el resultado: «calentamiento global». En este contexto se encuentran dos factores que entran en conflicto, el cumplimiento de la demanda y el impacto ambiental. Se podría satisfacer toda la demanda, pero el impacto medioambiental sería el máximo; y si se quisiese reducir a cero el impacto medioambiental, se tendría que detener la producción y el comercio. Así, ante el evidente escenario de conflicto, en (Pascual-González et al., 2016) se desarrolla un estudio en el cual emplean varias herramientas, entre ellas la OMO, con el fin de determinar estrategias que tengan como meta primordial minimizar las emisiones de CO_2, pero a la vez, buscando maximizar la satisfacción de la demanda.

Otros estudios en esta misma línea son: (Cui et al., 2017), (Mousavie-Avval et al., 2017)

- **Aplicaciones en medicina:** Un ámbito importante para cualquier población es la salud. El desarrollo de la OMO ha logrado crear aplicaciones muy útiles en este sector, obteniendo importantes resultados para que la labor médica logre mejores resultados en el cuido y preservación de la vida humana.
    - **Determinando la dosis de Pretomanida:** Según la Organización Panamericana de la Salud (OPS), diariamente 4,000 personas mueren a causa de la tuberculosis. Esta se considera una enfermedad prevenible y curable, pero hay factores que intervienen en la dosificación del tratamiento de esta enfermedad, entre ellos encontramos: la posible resistencia a medicamentos cuando se aplica un solo tipo de fármaco, límites de dosificación en la combinación de distintos fármacos, y técnicas limitadas para la selección de las dosis pertinentes de múltiples medicamentos. Un estudio pretende dar solución al último problema, a saber (Lyons, 2021). Usando un enfoque de OMO, se identificaron regímenes de dosificación que producen compromisos óptimos entre los múltiples objetivos terapéuticos en conflicto. En este se probaron dos medicamentos, la Pretomanida y el Nitriomidazol, recomendados para casos de tuberculosis pulmonar con alta resistencia a los medicamentos. Lo que se logró fue crear un modelo de optimización que minimiza mejor la carga bacilar del esputo y la probabilidad de eventos adversos ante la medicación.
    - **Problemas de listas de esperas quirúrgicas:** Otro ejemplo interesante en esta área es la optimización de las listas de esperas de pacientes que aguardan por una cirugía. Un ejemplo, aunque con datos no tan recientes, lo encontramos en (Carballo Mato, 2019). En este se tiene un hospital con listas de espera muy grandes para 4 procedimientos quirúrgicos en el año 1998. Se plantea la necesidad de que los pacientes sean atendidos en no más de 9 meses, 6 meses en el caso de que entren en la lista después de julio. El número de quirófanos es limitado, aunque hay suficientes camas y personal. Entonces, se plantea un modelo de OMO que ayuda a resolver este problema particular, teniendo como objetivos: minimizar la lista de espera al final del año (la menor cantidad de pacientes que tengan que esperar a ser operados el año siguiente) y minimizar los costes.
- **Aplicaciones en redes:** Otras de las aplicaciones que pueden resultar interesantes son las siguientes.
    - **Rehabilitación de redes de alcantarillado:** Un ejemplo de aplicación de la OMO a redes lo encontramos en (Yazdi et al., 2015). En este se hace un estudio comparativo entre diferentes algoritmos, con el fin de crear planes de asignación óptimos para la rehabilitación de redes de alcantarillado. En este caso se pretendía, por un lado, minimizar el costo de la rehabilitación y, por otro, maximizar el volumen de fluidos que puede pasar por la red (minimizar la sobre carga de la red). Claramente puede verse que estos dos factores entran en conflicto, ya que a mayor inversión se minimiza la sobrecarga de la red, y viceversa.
    - **Mejorando la robustez de una red:** No hay un consenso respecto a la definición de robustez de una red. Pero, de manera general, podría decirse que la robustez tiene que ver con la capacidad de una red para soportar fallas o ataques. En este sentido se han introducido varias métricas para poder capturar esa idea, algunas propiedades deseables para la robustez son:
        - **Conectividad:** se desea que una red permanezca conectada.
        - **Distancias entre nodos:** se espera que en una red robusta, la distancia (mínima) entre los nodos no varíe mucho.
        - **Propiedades de la red:** se desea que la distribución de los grados de los nodos, así como la distribución de las distancias cambien muy poco.

      Lo anterior, se espera cuando algunos de los nodos fallen. Así, en (Gunasekara et al., 2018) se analizan tres medidas de robustez incorreladas, a saber *Size of the largest connected component* (LCC), *Spectral gap* (SG), *Normalized effective resistance* (nER). Lo que se pretende, entonces, es, dado un número de nodos, encontrar el conjunto de k aristas, de modo que se logre maximizar las tres medidas de robustez, lo más que se pueda.

Así, con estos ejemplos, podemos ver la utilidad de la OMO, que no agota su aplicabilidad en los ejercicios ideales de los libros, sino que su alcance llega hasta la realidad, donde ayuda a los tomadores de decisiones a desarrollar planes que permitan alcanzar los mejores resultados, cuando los objetivos entran en conflicto. Para más ejemplos, ver: (Mandal et al., 2018), (Branke et al., 2008)

1.2. El problema de Optimización Multiobjetivo

Para mejor manejo de las definiciones, si no se indica lo contrario, se considerarán problemas de optimización multiobjetivo de minimización (minimizar todas las funciones objetivos). Así, la definición formal del **problema de optimización multiobjetivo (MOOP)** es:

Definición 1.2.1 (**Problema de optimización multiobjetivo (MOOP)**[1]).

$$\begin{aligned} \textit{minimizar} \quad & f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix} \\ \textit{sujeto a} \quad & x \in \mathcal{S} \end{aligned}$$

donde: $x \in \mathbb{R}^n$, es un vector cuyas componentes son las «variables de decisión»; $f : \mathbb{R}^n \to \mathbb{R}^m$ es el vector de funciones objetivos, y $\mathcal{S} = \{x : g_j(x) \geq 0, h_k(x) = 0, x_i^{(L)} \leq x_i \leq x_i^{(U)}\}$ es la región factible.

Puede verse que $x_i^{(L)}$ y $x_i^{(U)}$ representan cotas para las variables de decisión, inferior y superior, respectivamente. Generalmente, la cota inferior se toma como igual a 0. También cabe mencionar que existen otro tipo de restricciones que limitan las variables a cierto tipo de valores; por ejemplo, que estas sean discretas ($x_i \in \mathbb{N}$) o binarias ($x_i \in \{0, 1\}$).

Definición 1.2.2 (**Espacio de decisión**). *El espacio de decisión, $\mathcal{D}$, es el conjunto de soluciones x que cumplen $x_i^{(L)} \leq x_i \leq x_i^{(U)}$, donde x_i es la componente i-ésima de x.*

Definición 1.2.3 (**Solución factible**). *Sea $x \in \mathcal{D}$, donde $\mathcal{D}$ es el espacio de decisión asociado a un MOOP. Diremos que x es una solución factible si cumple, además, con las restricciones del MOOP asociado. De lo contrario se dice **«no factible»**.*

Definición 1.2.4 (**Región factible**). *El conjunto de todas las soluciones factibles se conoce como región factible, $\mathcal{S}$. También se le conoce como **espacio de solución**.*

Hay que notar que en la OMO, las funciones objetivos forman un espacio multidimensional, conocido como espacio de objetivos. Esto marca una clara diferencia entre la optimización multiobjetivo y la mono-objetivo, en esta última solo se tiene el espacio de decisión.

Definición 1.2.5 (**Espacio objetivo, $\mathcal{Z}$**). *Para cada $x \in \mathcal{S}$ existe $f(x) = (z_1, z_2, \ldots, z_m)' = z$. Al conjunto de todos los puntos z se le conoce como espacio objetivo.*

1.3. Dominancia y soluciones de Pareto

Cuando se tiene solo un objetivo, es posible determinar la «calidad» de las soluciones factibles comparando las evaluaciones de estas en la función objetivo. Lo anterior no se puede hacer en los MOOP, ya que, en cierto sentido, este tipo de problemas se encuentran «mal definidos» (Cf. Miettinen, 2004, p. 11), porque no existe una relación natural en el espacio objetivo que nos permita determinar un orden para cada par de puntos de dicho espacio. Es por eso, que la «calidad» de una solución en este tipo de problemas queda determinada por la dominancia.

Definición 1.3.1 (**Dominancia**). (Goldberg, 1989, p. 198) Sean x, $y \in \mathbb{R}^m$, tales que $x \neq y$. Diremos que x es parcialmente menor que y o que x **domina a** y si se cumplen las siguientes condiciones:

- $x_i \leq y_i \quad \forall i$ (x no es peor que y).
- $\exists i / x_i < y_i$ (x es estrictamente menor que y en al menos una componente).

[1] Observése que en un MOOP se desea optimizar un vector, donde las componentes son las funciones objetivo; justamente por esto a la optimización multiobjetivo se le conoce, también, como **optimización vectorial**.

Definición 1.3.2 (**Dominancia débil**). *Sean x, $y \in \mathbb{R}^m$, tales que $x \neq y$. Diremos que x* ***domina débilmente*** *a y, simbolicamente $x \preceq y$, si se cumple:*

$$x_i \leq y_i, \quad \forall i = 1, 2, \ldots, n$$

Definición 1.3.3 (**Dominancia fuerte**). *Sean x, $y \in \mathbb{R}^m$, tales que $x \neq y$. Diremos que x* ***domina fuertemente*** *a y, simbólicamente $x \prec y$, si se cumple:*

$$x_i < y_i, \quad \forall i = 1, 2, \ldots, n$$

Ejemplo 1.3.1. *Consideremos los siguientes puntos: $x_1 = (1.5, 2), x_2 = (1, 4), x_3 = (2.5, 1)$, $x_4 = (2, 3), x_5 = (3.5, 2)$, los cuales pueden verse en la Figura 1.2.*

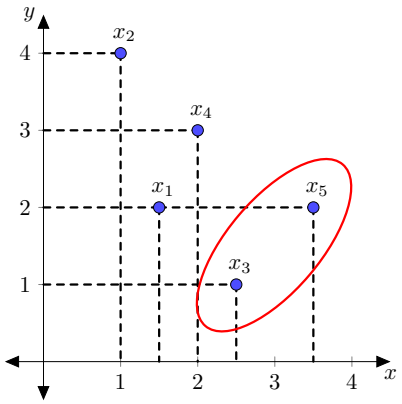


Figura 1.2: Ejemplo de soluciones no dominadas

Si consideramos como mejores aquellos puntos que maximizan la primera coordenada y minimizan la segunda, podemos ver que las dos mejores opciones son x_3 y x_5. Está claro, por ejemplo, que x_3 domina fuertemente a x_1, ya que $x_{3_1} > x_{1_1}$ y $x_{3_2} < x_{1_2}$. También, puede verse que, según el criterio, no puede considerarse, entre x_3 y x_5, uno de estos puntos como mejor opción, ya que $x_{3_1} < x_{5_1}$ y $x_{3_2} < x_{5_2}$, es decir, ambas representan buenas soluciones, y puede verificarse que no hay mejores puntos que esos dos.

De lo anterior podemos inferir que, en un MOOP, una solución es óptima si no existe otra solución factible que mejore el resultado de todas las funciones objetivos. A este tipo de soluciones óptimas, de la OMO, se les conoce como «soluciones de Pareto», «Pareto-eficiente» o «Pareto-óptima».

Definición 1.3.4 (**Solución Pareto-eficiente**). *Sean $f : \mathbb{R}^n \to \mathbb{R}^m$ y $x \in \mathcal{S}$. Para el problema de optimización*

$$\begin{gathered} \textit{minimizar } f(x) \\ \textit{sujeto a } x \in \mathcal{S} \end{gathered}$$

un punto $x^* \in \mathcal{S}$ se llama Pareto-eficiente o ***solución no dominada*** *si no existe $x \in \mathcal{S}$ tal que para $i = 1, 2, \ldots, m$*

$$f_i(x) \leq f_i(x^*)$$

y para al menos un i,

$$f_i(x) < f_i(x^*)$$

Definición 1.3.5 (**Frente de Pareto**). *El conjunto de todos los puntos Pareto-eficiente se conoce como el frente de Pareto.*

Ejemplo 1.3.2 (**Ejemplos de frentes de Pareto**). *Óbservese, en las gráficas de la Figura 1.3, que el frente de Pareto queda determinado por el tipo de optimización que se esté realizando.*

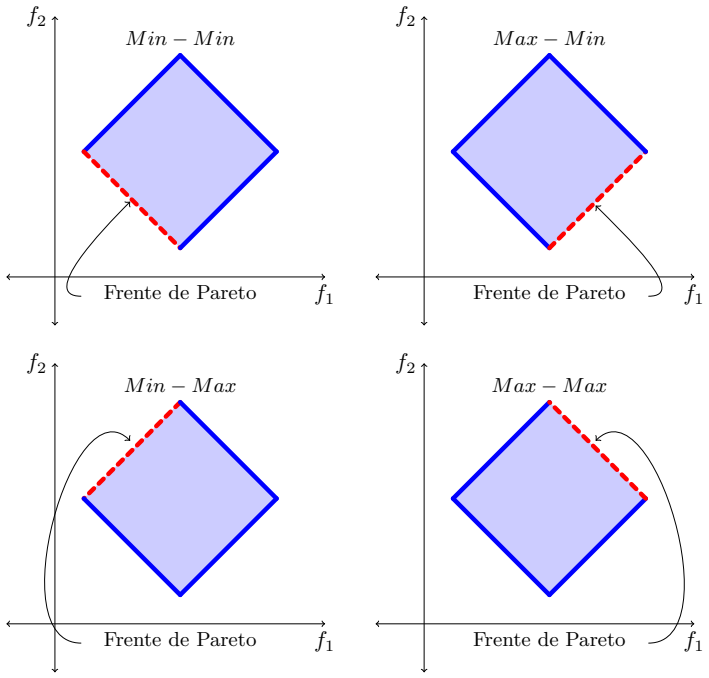


Figura 1.3: Ejemplos de frente de Pareto

Ejemplo 1.3.3. *Considere el siguiente MOOP:*

$$\begin{aligned} \textit{minimizar} \quad & \begin{bmatrix} 0.5x_1 + x_2 \\ 1.5x_1 + x_2 \end{bmatrix} \\ \textit{sujeto a} \quad & x_1 + x_2 \geq 1 \\ & 0 \leq x_i \leq 1, i \in \{1, 2\} \end{aligned}$$

En la Figura 1.4 se puede apreciar, en la gráfica de la izquierda: la región factible, $\mathcal{S}$, las soluciones que producen el frente de Pareto están marcados en naranja; en la gráfica de la derecha: el espacio objetivo, $\mathcal{Z}$, los puntos que forman el frente de Pareto están marcados en rojo.

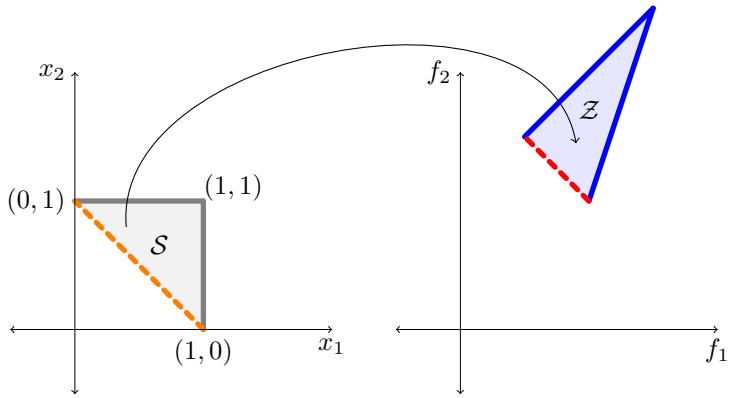


Figura 1.4: Ejemplos de región factible (izquierda) y espacio de objetivos (derecha), pertenecientes al Ejemplo 1.3.3.

Capítulo 2

Métodos de la Optimización Multiobjetivo

Existe una extensa lista de métodos para resolver MOOP, cada uno posee características propias que pueden hacerlos útiles para la resolución de ciertos tipos, específicos, de problemas. Debido a la gran variedad de estos, no se puede decir que haya un método superior a los demás, en todos los sentidos. La elección de un método implica una serie de condiciones que hay que tomar en cuenta, como las características propias del problema que se quiere resolver y las preferencias del modelador, la variabilidad de la eficiencia de los métodos para problemas puntuales, entre otros; por esto, elegir un método implica ya un MOOP. Para ver más sobre el problema de la selección de métodos revisar (Miettinen, 2004, Part III, 1.3) y (Yann y Siarry, 2003, p. 223).

2.1. Tipos de métodos

Para resolver problemas de optimización, en general, como una primera clasificación, se dispone de dos perspectivas, a saber:

- **Métodos *determinísticos*:** Estos tienen su base fundamental en el cálculo, trabajan con una solución y usan información obtenida del gradiente de la función objetivo y las restricciones para generar un nuevo candidato.
- **Métodos *no determinísticos*:** Estos, por su parte, se basan en analogías con la naturaleza; trabajan con una población de candidatos y usan diversas estrategias para modificarlos y así generar una nueva población.

Una diferencia notoria entre estos dos tipos de métodos, a parte del número de soluciones con las que trabajan, es que en los métodos no determinísticos se usan números pseudoaleatorios para la generación y modificación de soluciones, cosa que no ocurre en los métodos determinísticos.

La clasificación anterior no es la única, existen distintas clasificaciones de los métodos de resolución de problemas de optimización; cada cual toma en cuenta diferentes criterios para agruparlos. Así, otra clasificación la encontramos en (Miettinen, 2004), aquí se clasifican en cuatro familias de métodos:

- **Métodos de no preferencia:** En estos, las opiniones del tomador de decisiones no son tomadas en consideración, ya que probablemente este no tiene ninguna expectativa respecto a la solución del problema, solo necesita una solución óptima.
- **Métodos *a priori*:** Los compromisos a ser aplicados son definidos antes de ejecutar el método de optimización.
- **Métodos interactivos o progresivos:** En los que el tomador de decisiones va alterando los compromisos a medida que se ejecuta el método de resolución.
- **Métodos *a posteriori*:** En este tipo de métodos se elige una solución después de examinar las soluciones generadas por el método de optimización.

Hay que destacar que existen métodos que no encajan exclusivamente en ninguna de las tipologías anteriores, ya que son una mezcla de algunos de dichos tipos.

A su vez, en (Yann y Siarry, 2003) se propone otra clasificación, que los divide en 5 grupos, a saber:

1. Métodos escalarizados.
2. Métodos interactivos.
3. Métodos fuzzy.
4. Métodos que usan metaheurística.
5. Métodos de ayuda de decisión.

En las siguientes secciones, con el objetivo de ahondar en la resolución de los MOOP, se tratarán los métodos 1. y 4. de esta última clasificación, así como algunos métodos particulares de los mismos. Nos centraremos en estos dos tipos, porque los métodos escalarizados se presentan como una herramienta inicial (básica) para resolver MOOP y los métodos que usan metaheurísticas son una herramienta apropiada para dar solución a la misma clase de problemas; estos últimos serán implementados en páginas posteriores.

2.2. Métodos escalarizados o agregativos

Aquí se presentan los métodos agregativos; se ha puesto especial énfasis en estos, ya que representan un primer acercamiento a la resolución y, quizás, el modo más sencillo de tratar los MOOP.

El propósito en estos métodos es combinar los distintos objetivos, a modo de convertir el MOOP original en un problema equivalente de optimización mono-objetivo. Esto se hace con el fin de usar las técnicas de este último tipo para darles solución.

Cabe destacar que este tipo de métodos da pie a otra clasificación de los métodos de resolución, en **métodos agregativos** y **métodos no agregativos** (Yann y Siarry, 2003), en donde, como la intuición lo indica, en estos últimos no se pretende hacer la conversión del MOOP en un problema mono-objetivo.
A continuación, se presentan algunos de los métodos agregativos más usados.

1. **Método de las sumas ponderadas:** Al ser un método escalarizado, se parte de un problema como el de la Definición 1.2.1, y se transforma en un problema mono-objetivo donde se desea optimizar una combinación lineal ponderada de los objetivos originales. Es decir, se obtiene:

$$\begin{aligned} &\text{minimizar} \quad f(x) = \sum_{i=1}^{m} w_i \cdot f_i(x) \\ &\text{sujeto a} \quad x \in \mathcal{S} \end{aligned}$$

   Donde los w_i representan los pesos o grados de importancia que se le dan a cada uno de los objetivos. Estos cumplen que $w_i \geq 0$ para toda $i \in \{1, 2, \ldots, m\}$, y usualmente se toma $\sum_{i=1}^{m} w_i = 1$, es decir la combinación lineal es convexa.

2. **Método híbrido:** Propuesto primeramente por R.E. Wendell y D.N. Lee en 1977, según (Miettinen, 2004, p. 96), y posteriormente por H.W. Corley en 1980, (íbid.), aunque de formas particulares. Este parte de la combinación del método de las sumas ponderadas y el método de la $\varepsilon-$restricción[1]. Este se plantea de la siguiente manera:

$$\begin{aligned} &\text{minimizar} \quad \sum_{i=1}^{m} w_i \cdot f_i(x) \\ &\text{sujeto a} \quad f_j(x) \leq \varepsilon_j \ \ \forall j = 1, 2, \ldots, m, \\ &\qquad\qquad x \in \mathcal{S} \end{aligned}$$

   donde, tenemos que $w_i > 0$ para cualquier $i = 1, 2, \ldots, m$.

[1] **Método de la $\varepsilon-$restricción:** En (Haimes et al., 1971) se sugiere reformular el MOOP, tomando como único objetivo una función del conjunto de objetivos del problema original, supongamos f_l, y las demás se introducen como restricciones, asignándoles una cota superior. El modelo queda:

$$\begin{aligned} &\text{minimizar} \quad f_l(x) \\ &\text{sujeto a} \quad f_j(x) \leq \varepsilon_j \quad \forall j = 1, 2, \ldots, m \text{ con } j \neq l, \\ &\qquad\qquad x \in \mathcal{S} \end{aligned}$$

Los parámetros ε_j representan una cota superior para cada función distinta de la que se quiere optimizar, estos no necesitan ser valores pequeños cercanos a cero. Cabe destacar que este método supera las dificultades que tiene el método de las sumas ponderadas para resolver problemas no convexos.

3. **Programación por metas:** Este método fue introducido por Charnes et al. (1955), citado por (Deb, 2001, p. 67), aplicándolo a un problema lineal mono-objetivo. En este, el tomador de decisiones debe asignar metas para uno o más objetivos. La idea acá es minimizar la distancia entre la evaluación de los puntos en cada objetivo y las metas que se desean alcanzar. Matemáticamente, este método está dado por:

$$\begin{aligned} \text{minimizar} \quad & \sum_{i=1}^{m} |f_i(x) - T_i| \\ \text{sujeto a} \quad & x \in \mathcal{S} \end{aligned}$$

donde T_i denota la meta que se desea alcanzar para el i-ésimo objetivo. Está claro que cuando no se le asigna una meta a un objetivo i, entonces $T_i = 0$.

4. **Método de las métricas ponderadas:** Asumimos que $w_i \geq 0$ para toda $i = 1, 2, \ldots, m$ y que $\sum_{i=1}^{m} w_i = 1$. También se asume un punto objetivo ideal $T \in \mathbb{R}^m$, elegido por el tomador de decisión, al que la solución encontrada por el método debe acercarse lo máximo posible.

   - **Problema ponderado L_p:** En este se usan las métricas L_p para minimizar la distancia entre la solución encontrada x y la solución ideal T. Así, tenemos que el problema original debe llevarse a uno de la siguiente forma.

$$\begin{aligned} \text{minimizar} \quad & \left(\sum_{i=1}^{m} w_i |f_i(x) - T_i|^p \right)^{1/p} \\ \text{sujeto a} \quad & x \in \mathcal{S} \end{aligned}$$

     para $1 \leq p < \infty$.

   - **Problema ponderado de Chebyschev:** Este también es conocido como el método min-max y tiene el siguiente esquema.

$$\begin{aligned} \text{minimizar} \quad & \max_{i=1,\ldots,m} [w_i |f_i(x) - T_i|] \\ \text{sujeto a} \quad & x \in \mathcal{S} \end{aligned}$$

     Está claro que alterando los valores de los w_i obtenemos diferentes resultados para los problemas usando métricas L_p o la métrica de Chebyschev.

Una ventaja evidente en este tipo de métodos es el hecho de que, al convertir el MOOP en un problema de optimización mono-objetivo, hay una serie de estrategias y algoritmos bastante eficientes para resolverlos. Entre las desventajas encontramos que no hay estrategias que permitan determinar con certeza los parámetros necesarios a usar en los mismos; pesos o metas, por ejemplo. Esto último puede llevar a que la solución encontrada no se adapte a las características deseadas por el tomador de decisiones. A este respecto, también, tenemos el hecho de que en los MOOP casi siempre se generan compromisos entre los objetivos, dando pie al surgimiento de un frente de Pareto. Así, obtener una única solución en este tipo de problemas puede ser una ventaja y una desventaja.

2.3. Métodos que usan metaheurística

El término «*metaheurísticas*» procede de la conjunción de dos terminos griegos, a saber: $\mu\varepsilon\tau\alpha$ (más allá) y $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ (descubrir); y esta designa a un grupo de métodos no determinísticos. Estos aparecieron por la necesidad de superar las limitaciones que presentan los métodos determinísticos, ya que muchos de estos necesitan condiciones como linealidad, convexidad o continuidad de las funciones objetivos para poder ser aplicados. Pero es evidente que no todos los problemas son lineales, convexos o continuos; en el caso de los MOOP aplicar estos métodos es poco práctico o imposible, ya que, recuérdese, que las funciones a optimizar entran, casi siempre, en conflicto.

Cabe destacar, también, que existen algunos problemas combinatorios para los cuales no se tienen algoritmos determinísticos eficientes y problemas de variables continuas para los que no se tienen ningún algoritmo que encuentre un óptimo global en una cantidad finita de iteraciones, en este contexto los métodos no determinísticos son una herramienta importante para tratar de dar soluciones a los problemas que son difíciles de solucionar con los métodos clásicos, incluidos los MOOP.

Estos métodos poseen algunas características en común:

- Son estocásticos, al menos parcialmente.
- No poseen como base principal herramientas del cálculo.
- Están inspirados por analogías:
  - en la física (enfriamiento simulado, difusión simulada, entre otras).
  - en la etología (colonias de hormigas y enjambres de partículas, entre otros).
  - en la biología (búsqueda tabú, algoritmos genéticos, entre otros).
- Trabajan con poblaciones de candidatos y usan estrategias para la modificación de los mismos, con la intención de actualizar (mejorar) la población.

Estos no son mutuamente excluyentes y cada vez más se apunta a la hibridación de dichos métodos, combinándolos para obtener métodos más eficientes para problemas específicos.

Algunas **ventajas** de este tipo de métodos son:

- No requieren ninguna información respecto a las derivadas de las funciones objetivos o las restricciones.
- Admite el tratamiento no solo de variables reales, sino también enteras y binarias.
- Pueden escapar de mínimos locales.
- Relativamente fáciles de implementar.
- Versátiles y con un amplio rango de aplicación.
- No necesitan de condiciones como convexidad, linealidad, continuidad o diferenciabilidad de las funciones que intervienen en el problema.

Las **desventajas** más relevantes de los métodos no determínisticos son:

- Tienen costos computacionales altos en cada ejecución, en comparación con los métodos determinísticos.
- Los resultados encontrados en cada ejecución dependen de la semilla de los generadores de números pseudoaleatorios que intervienen en el algoritmo.
- Poseen parámetros libres, los cuales son difíciles de calibrar.

Así, por sus características, los métodos que usan metaheurísticas se presentan especialmente apropiados para resolver los MOOP; ya que, como trabajan con poblaciones de soluciones, permiten aproximar el Frente de Pareto del MOOP que se esté resolviendo. Asimismo, dado que no requieren condiciones especiales, como la diferenciabilidad de los objetivos o convexidad del espacio objetivo, son útiles para diversos tipos de MOOP que no cumplen con las condiciones antes mencionadas y que serían realmente difíciles (o imposibles) de resolver si no se contará con estos métodos. De este tipo de métodos nos centraremos en dos específicos; los algoritmos genéticos y los inspirados en enjambres de partículas. Estos se desarrollarán a continuación.

2.3.1. Algoritmos genéticos (AG)

Estos son un tipo de algoritmos evolutivos, que están inspirados en la genética clásica. También vale decir que toman el vocabulario propio de esta ciencia para nombrar a las entidades que intervienen en estos algoritmos, y que se definen a continuación:

- **Individuo:** es una potencial candidato a ser el óptimo de un problema de optimización.
- **Genotipo o cromosoma:** corresponde a la codificación, como un gen, del individuo. Una posible representación de un cromosoma, es: [00101001], que es una representación clásica binaria de longitud 8. Aunque vale decir que esta representación puede extenderse a otro tipo de valores, como reales o enteros. Elegir dicha representación puede representar un obstáculo a vencer.
- **Gen:** Es cada componente que forma al cromosoma. En el caso binario, un gen puede tomar el valor de 0 o 1.
- **Fenotipo:** Representa el valor que toma un individuo en los objetivos. En el caso de los MOOP, este representa también un cromosoma, donde cada gen es el valor del individuo evaluado en cada objetivo.

- **Cruce:** Es un operador que permite la combinación de dos o más soluciones, con el fin de crear «descendientes». Existen diversas estrategias, como el cruce de *n-puntos* o el cruce *aritmético.*
- **Mutación:** Es un operador que realiza cambios aleatorios en los individuos.
- ***«Fitness»*:** Es la función que mide la calidad de las soluciones generadas por el algoritmo genético; en el caso MOOP, las funciones de *«fitness»* son los objetivos.
- **Selección:** Para garantizar la convergencia hacia el punto óptimo, las mejores soluciones son seleccionadas para ser «progenitores». Esta selección está basada en los valores de «*fitness*» de la población y garantiza que las soluciones con valores bajos de aptitud vayan desapareciendo.

La estructura general de los AG, en pseudocódigo, se presenta en Algoritmo 1 (Yann y Siarry, 2003, p. 122):

Algoritmo 1: Algoritmo genético

```
Inicialización de la población
Evaluación de la función objetivo
Cálculo de la eficiencia
while no se cumpla la condición de parada do
    Selección aleatoria
    Selección proporcional a la eficiencia
    Cruce
    Mutación
    Evaluación de la función objetivo
    Cálculo de la eficiencia
end
```

En el Algoritmo 1, la eficiencia corresponde al desempeño, «performance», de un individuo en la resolución del problema con el que se esté tratando. Asimismo, la condición de parada determina cuándo debe detenerse el AG, algunas de estas son:

- **Número de generaciones:** Se le indica el número máximo de generaciones que debe realizar el AG.
- **Número de evaluaciones:** El AG puede tener como condición de parada el número de evaluaciones que se realizan sobre la población.
- **Número de generaciones sin cambios:** Cuando las soluciones se aproximan al óptimo, el progreso del «*fitness*» decrece significativamente, por lo que las mejoras de este van siendo cada vez más pequeñas; así, una estrategia de parada puede ser que el algoritmo se detenga cuando no hayan cambios significativos en las soluciones después de determinada cantidad de generaciones.

Así, por sus bondades, los algoritmos genéticos han sido adaptados para lidiar con los MOOP. La estructura general, en pseudocódigo, de un AG para resolver MOOP, lo encontramos en el Algoritmo 2.

Algoritmo 2: Algoritmo genético para MOOP

```
Inicialización de la población
Cálculo de la eficiencia de los individuos de la población
Transformación de la solución en eficiencia
while no se cumpla la condición de parada do
    Cruce
    Mutación
    Cálculo de la eficiencia de los individuos de la población
    Transformación de la solución en eficiencia
    Selección
end
```

A continuación se presentan algunos de los algoritmos genéticos aplicados a la OMO.

1. **Vector Evaluated Genetic Algorithm (VEGA)**

   Propuesto por Schaffer en 1984, según (Deb, 2001), y es el primer algoritmo genético de OMO para encontrar soluciones no dominadas.

   En este algoritmo se trabaja con una población de N individuos, los cuales se dividen en k grupos con N/k individuos cada uno, donde k es el número de objetivos del problema y N es un múltiplo de k. En

cada grupo se evalúa una función objetivo, esta permite determinar la eficiencia de los individuos en el grupo. Después, los individuos son mezclados y se ejecuta el cruce con respecto a la eficiencia de cada individuo. Este se describe en el Algoritmo 3, (Yann y Siarry, 2003, p. 124).

Algoritmo 3: VEGA

Inicialización de una población de tamaño N
while *no se cumpla la condición de parada* **do**
  Cálculo de la eficiencia de los individuos de la población
  Creación de k grupos (subpoblación)
  Mezcla de los individuos
  Se aplica el genético algoritmo clásico (cruce - mutación - selección)
end

Una desventaja de este método es que los individuos tienden a valores promedios respecto a todos los objetivos, los cuales, de manera general, no representarían una buena aproximación del frente de Pareto; agregado a esto, se tiene que este algoritmo no es útil para encontrar soluciones que están en secciones cóncavas de dicho frente.

2. Multiple Objective Genetic Algorithm (MOGA)

Este método está basado en la relación de dominancia de Pareto. Aquí, se considera el «rango» (*rank* en inglés) y este está determinado por el número de individuos que dominan al individuo considerado. A los individuos no dominados se les asigna el valor 1. El algoritmo, en pseudocódigo, se presenta en Algoritmo 4, (Yann y Siarry, 2003, p. 126). Cabe mencionar que este algoritmo, en algunos casos, no permite obtener gran diversidad en las soluciones para aproximar, de mejor manera, el frente de Pareto.

Algoritmo 4: Algoritmo MOGA:

Inicialización de la población
Cálculo de los valores de los objetivos
Asignación del rango a cada individuo, usando dominancia
Asignación de eficiencia, usando el rango
while *no se cumpla la condición de parada* **do**
  Selección aleatoria en proporción a la eficiencia
  Cruce
  Mutación
  Evaluación de las funciones objetivos
  Asignación del rango, usando dominancia
  Asignación de la eficiencia, usando el rango
end

3. Nondominated sorting genetic algorithm (NSGA)

Este método considera distintos niveles de individuos. En un primer paso, antes de la selección, se asigna un rango a cada individuo de la población. Todos los individuos no dominados son agrupados en la misma categoría. Su algoritmo, en pseudocódigo, está en Algoritmo 5, (Srinivas y Deb, 1994).

A partir del NSGA se han desarrollado otros algoritmos, como el NPGA y el NSGA-II; este último se utilizará para realizar algunas implementaciones en el siguiente capítulo.

4. Niched Pareto Genetic Algorithm (NPGA)

Este es un método basado en el método NSGA, la diferencia se presenta en el proceso de selección. Su algoritmo, en pseudocódigo, está en Algoritmo 6, (Cf. Yann y Siarry, 2003, p. 131).

Algoritmo 5: NSGA:

```
Inicialización de la población
while no se cumpla la condición de parada do
    Frente ← 1
    while no hayan sido clasificados todos los individuos de la población do
        Identificar los individuos no dominados
        Asignar la eficiencia
        Asociar los individuos al frente de Pareto
        Frente ← Frente +1
    end
    Reproducir de acuerdo a la eficiencia
    Cruce
    Mutación
end
```

Algoritmo 6: NPGA:

```
Inicialización de la población
Cálculo de los valores de los objetivos
while no se cumpla la condición de parada do
    Torneo de selección entre dos individuos
      Solo el candidato 1 es dominado: Seleccionar el candidato 2
      Solo el candidato 2 es dominado: Seleccionar el candidato 1
      Ambos candidatos son dominados o no dominados
        Ejecutar la eficiencia compartida
        Seleccionar con el menor número de vecinos
    Cruce
    Mutación
    Evaluación de las funciones objetivos
end
```

2.3.2. Enjambre de partículas

Este es un tipo de método no determinístico; inspirados en la inteligencia social de las bandadas de pájaros en la búsqueda de comida, fue presentado por Ederhart y Kennedy en 1995, según (Tang et al., 2012). Estos dan origen a la optimización por enjambres de partículas («PSO», por sus siglas en inglés), usan un simple, pero efectivo, mecanismo en el cual cada partícula posee tres factores: la velocidad (v_i), su mejor experiencia previa ($pBest_i$) y la mejor experiencia en el enjambre ($gBest$). Estos han demostrado ser una herramienta muy útil en un amplio rango de problemas de optimización, incluidos los MOOP.

Parecido a los algoritmos genéticos en la generación de la población inicial, en cuanto que inicia con un enjambre aleatorio de partículas; en cada iteración todos los individuos tratan de imitar la mejor partícula del enjambre o bien dirigirse a su mejor experiencia previa. Un algoritmo general de la estrategia PSO se presenta en el Algoritmo 7, (Mouayad y Bestoun, 2016).

A continuación se comentarán dos de los algoritmos PSO aplicados a la optimización multiobjetivo.

1. **Adaptative Weighted PSO (AWPSO)**

   Fue propuesto en (Mahfouf et al., 2004), en este se incluye un término que sirve de aceleración y modifica la velocidad, este aumenta a medida que incrementa el número de iteraciones. Esta estrategia ayuda al algoritmo a escapar de óptimos locales. Se propone usar pesos dinámicos, con la intención de generar soluciones no dominadas. Cuando la población está perdiendo diversidad, se aplica un operador de mutación a determinadas partículas y se retienen las mejores de ellas. Finalmente, los autores incluyen un algoritmo de clasificación de soluciones no dominadas para seleccionar las partículas que pasarán de una iteración a otra.

Algoritmo 7: Algoritmo básico PSO

```
Inicialización de la población
Inicializar velocidades
while No se cumpla la condición de parada do
    for cada partícula p do
        Cálculo de los valores de los objetivos («fitness(p)»)
        if el fitness(p) es mejor que su pBest_i then
            pBest_{i+1} ← fitness(p)
    end
    Escoger la partícula con mejor fitness de todos como gBest
    for cada partícula do
        Calcular la velocidad
        Actualizar la posición de la partícula
    end
end
```

2. **Vector Evaluated PSO (VEPSO)**

   Propuesto en (Parsopoulos y Vrahatis, 2002), este es un método inspirado en el Vector Evaluated Genetic Algorithm (VEGA), antes mencionado. La idea detrás de este algoritmo es evolucionar varios enjambres usando solo uno de los objetivos del problema, y la información del desarrollo es transmitido a los otros enjambres a través del intercambio de las mejores experiencias ($gBest$). Esta estrategia permite alcanzar el frente de Pareto.

En el siguiente capítulo se presentará el algoritmo «Multi-Objective PSO», el cual es una herramienta importante, desde las estrategias PSO, para la resolución de los MOOP.

2.3.3. Tratamiento de soluciones no factibles en métodos metaheurísticos

Cuando se trata con problemas de optimización, muchas veces se requiere que las variables estén restringidas por cotas superiores o inferiores, así como que cumplan ciertas restricciones; esto da pie a la conformación del espacio factible y a la clasificación de soluciones en «factibles» y «no factibles». Lo deseable es que todas las restricciones se cumplan, es decir que todas las soluciones encontradas sean factibles.

En los métodos no determinísticos, los operadores pueden generar soluciones no factibles. Por ende, uno de los principales problemas al usar este tipo de métodos es crear estrategias para preservar la factibilidad de las soluciones; esto es una tarea de suma importancia para la solución de problemas de optimización con restricción. A continuación se presentan algunas estrategias para abordar las soluciones no factibles que surgen en la ejecución de algunos de estos algoritmos.

1. **Pena de muerte:** Un acercamiento sencillo al tratamiento de las soluciones no factibles es el de la pena de muerte. El proceso que sigue este procedimiento se expone en el pseudocódigo del Algoritmo 8, (Cf. Kramer, 2017, p. 41):

   Algoritmo 8: Pena de muerte

   ```
   while la solución es no factible do
       Cruce
       Mutación
       Verificar restricciones
   end
   ```

   Es un mecanismo bastante sencillo para lidiar con las restricciones; aunque, como puede verse, no es un algoritmo eficiente, ya que puede tomar muchas iteraciones generar soluciones factibles.

2. **Métodos de penalización:** Este método consiste en asignar una penalización en la función objetivo a aquellas soluciones que no son factibles, esto les reduce el «fitness», en comparación con las soluciones factibles. Así, podríamos considerar las funciones objetivos penalizadas: $f_p(x) = f(x) + p(x)$. Como, desde el inicio, asumimos tratar problemas de minimización, se tiene que esta penalización, $p(x)$, es

positiva y tiene que ser un número conveniente y cuidadosamente elegido para que la estrategia sea útil, no hay recomendaciones generales para elegirlo. Algunas estrategias son: que la penalización sea un valor estático en todas las iteraciones (penalización estática), otra es que este vaya variando a medida que evoluciona el algoritmo (penalización dinámica). Para más métodos de asignación del valor de penalización, ver (Dreo et al., 2006) y (Yu y Mitsuo, 2010).

Este método puede ser útil, no solo para tratar las restricciones de las variables, sino también para tratar restricciones del tipo: $g_j(x) \geq 0, h_k(x) = 0$, donde si la solución no cumple con estas, entonces se le agrega una penalización.

3. **Superioridad de los individuos factibles:** La idea central de estos métodos es trabajar bajo el supuesto de que una solución factible es mejor que una infactible. El primero de este tipo fue propuesto en (Deb, 2000), y en estos se propone una especie de torneo entre las distintas soluciones. Así, se seleccionan dos soluciones y se tienen las siguientes posibilidades:
   - que ambas soluciones sean factibles, donde la que tenga mejor «fitness» gana;
   - si una es factible y la otra infactible, la factible gana;
   - si ambas son infactibles, la que viole menos restricciones gana.

   Este método presenta la ventaja de que no es necesario calcular las funciones objetivos en individuos que están fuera de la región factible, y es de gran utilidad para problemas en los que alguna de las funciones a optimizar no está definida fuera de dicha región.

4. **Métodos de reparación:** En estos métodos la intención principal es convertir las soluciones infactibles en factibles (o que por lo menos pertenezcan al espacio de decisión). Considerando x como un individuo de la población utilizada en un algoritmo evolutivo de optimización, se desea que $x_i^{(L)} \leq x_i \leq x_i^{(U)}$ para que esta sea una solución factible, donde $x_i^{(L)}$ y $\leq x_i^{(U)}$ son las cotas inferiores y superiores, respectivamente, de la componente i-ésima, x_i, de x. Un algoritmo de reparación lo encontramos en el Algoritmo 9.

Algoritmo 9: Algoritmo de reparación - I

```
for i = 1 hasta m do
    if x_i < x_i^(L) o x_i > x_i^(U) then
        x_i ← máx(mín(x_i, x_i^(U)), x_i^(L))
    end
end
```

Donde m representa el número de componentes de cada individuo. Puede verse que la reparación implica llevar aquellas componentes que no están en los intervalos permitidos, a la cota de dicho intervalo que esté más cercana.

Ejemplo 2.3.1. *Considérese un MOOP que tiene la siguiente restricción sobre las variables* $x_i \in [0, 1]$ *para* $i = 1, 2, \ldots, 5$. *Ahora considérese* $x = [0.1, 2.1, 0.56, 0.8, -1.5]$, *es evidente que esta solución es infactible. Aplicando el Algoritmo 9, tenemos la reparación de la solución, de la cual se obtiene que:* $x = [0.1, 1, 0.56, 0.8, 0]$

A partir del Algoritmo 9, también se puede considerar:

Algoritmo 10: Algoritmo de reparación - II

```
for i = 1 hasta m do
    if x_i < x_i^(L) o x_i > x_i^(U) then
        x_i ← RUnif(x_i^(U), x_i^(L))
    end
end
```

Donde, en lugar de considerar los extremos, se considera un valor pseudoaleatorio generado con distribución uniforme en el intervalo permitido para la componente, esto es $\text{RUnif}(x_i^{(U)}, x_i^{(L)})$.

Cabe destacar que dichas estrategias serán de mucha utilidad, sobre todo, en problemas de optimización donde las únicas restricciones son las cotas de las variables.

5. **Operadores de variación para satisfacer las restricciones de las variables:** Es posible usar operadores (cruce y mutación) que sean capaces de generar únicamente individuos dentro de la región factible, lo que garantizaría que no se generen soluciones no factibles en la ejecución del algoritmo. Este presenta la ventaja de que no se tienen que calcular los valores de las funciones objetivos en dichas soluciones no factibles. El problema de estos es que dependen en gran medida de la representación que se esté usando para los individuos, diseñar dichos operadores para casos específicos puede ser muy difícil.

Capítulo 3

Estrategias evolutivas para la generación de soluciones de Pareto

Existe una diversidad de algoritmos para resolver los MOOP y, como ya se ha mencionado, no hay algoritmos absolutamente eficientes para la resolución de todos los problemas de este tipo. A pesar de lo anterior, hay algunos algoritmos que han sobresalido por su efectividad y eficiencia. Algunos de los algoritmos más usados son los algoritmos: «PAES», «NSGA-II» y «MOPSO», estos se han seleccionado para realizar las implementaciones que se trabajarán en este capítulo. Se eligieron por su eficiencia y por tratarse de enfoques distintos, esto último podría aportar diversidad en la obtención de las soluciones. Asimismo, se presentarán algunos problemas y su resolución usando un paquete del lenguaje Python, el paquete «EMO»; este toma como base los métodos del paquete *«Inspyred»* para resolver MOOP, con la posibilidad de implementar alguno de los tres algoritmos antes mencionados.

3.1. Pareto Archived Evolutionary Strategy

Este algoritmo fue propuesto en (Knowles y Corne, 2000), se le conoce por sus siglas como PAES. Es un algoritmo de búsqueda local que usa una estrategia evolutiva «(1+1)-ES» y en el que las mejores soluciones se van guardando en un archivo, de ahí que en su nombre se lea *«archived»*.

Según los autores este comprende tres partes, a saber:

1. el generador de la solución candidata,

2. la función de aceptación de las soluciones,

3. y el archivo de soluciones no dominadas.

Cabe destacar que el generador de las soluciones es parecido a una mutación aleatoria de escalada; el cual toma una solución y, por mutación aleatoria, produce una nueva solución en cada iteración. A esto último se le conoce como «(1+1)-ES». El pseudocódigo de este se presenta en Algoritmo 11.

Una parte integral de PAES es el uso de un procedimiento de agrupación basado en la división recursiva del espacio objetivo, generando una rejilla. Evidentemente cada solución pertenece a una celda de dicha rejilla, la cual puede ser determinada bisecando repetidamente el rango en cada objetivo y verificando en qué mitad está la solución. La rejilla será de gran importancia, porque ayudará a determinar niveles de concentración de soluciones, elemento decisivo para la función de aceptación *«test»*. Recuérdese que esta permitirá decidir cuando una solución mutada (hijo) es aceptada o no, así como decidir si se archiva o no. En Algoritmo 12 encontramos el pseudocódigo presentado por los autores para dicha función.

- **Ventajas**
  - El algoritmo tiene un control directo sobre la diversidad de soluciones que se puede alcanzar en el frente de Pareto.
  - La función *«test»* asegura que las soluciones que estén en regiones menos pobladas sobrevivan, de modo que se asegura la diversidad.
  - El tamaño de las celdas (hipercubos) en las que será dividido el espacio objetivo puede ser manipulado; debe tomarse en cuenta que si el tamaño de dichas celdas crece, el número de celdas

Algoritmo 11: (1+1)-PAES:

Generar la solución inicial aleatoria c y agregarla al archivo
* Mutar c para producir m y evaluar m
  if *c domina m* **then**
    Descartar m
  else if *m domina c* **then**
    Reemplazar c con m, y agregarla al archivo
  else if *m es dominada por algún miembro del archivo* **then**
    Descartar m
  else
    Aplicar test(c, m, archivo) para determinar cuál será la nueva solución actual y si se debe agregar m al archivo
  end
Hasta que el criterio de parada sea cumplido, regresar a *

Algoritmo 12: Pseudocódigo de la función *test(c, m,* archivo*)*

if *el archivo no está lleno* **then**
  añadir m al archivo
  if *m está en una región con menos densidad de soluciones del archivo que c* **then**
    aceptar m como la nueva solución actual
  else
    mantener c como la solución actual
  end
else
  if *m está en una región menos agrupada del archivo que x para algún miembro x del archivo* **then**
    agregar m al archivo, y remover un miembro del archivo de la región más agrupada
    if *m está en una región menos agrupada del archivo que c* **then**
      aceptar m como la nueva solución actual
    else
      mantener c como la solución actual
    end
  else
    if *m está en una región menos agrupada del archivo que c* **then**
      aceptar m como la nueva solución actual
    else
      mantener c como la solución actual
    end
  end
end

decrece; por lo que esto afecta directamente la diversidad de las soluciones. Así mismo, si se toman tamaños de celdas muy pequeños el coste computacional podría crecer, ya que el número de celdas aumenta.
- En comparación con otros algoritmos, este funciona muy bien con problemas cuyo frente de Pareto no se haya distribuido uniformemente en el espacio objetivo.

- **Desventajas**
  - No está muy claro la cantidad de soluciones que debe contener el archivo, para que el frente de Pareto quede bien representado.
  - El tamaño de las celdas de la rejilla puede ser un parámetro de difícil elección, tomando en cuenta que la variación de este puede hacer que la cantidad de celdas crezca exponencialmente. Algo está claro, las soluciones finales, dependerán de esta elección.

3.2. Non-dominates Sorting Genetic Algorithm II

Por sus siglas se le conoce como «NSGA-II» y es una estrategia de optimización evolutiva, propuesta en (Deb et al., 2002); está basada en varias etapas de clasificación de los individuos. La población es clasificada usando el concepto de dominancia. Los individuos no dominados pueden permanecer o ser eliminados de la población en cada iteración. Este representa una mejora del NSGA, a secas, en el cual la clasificación de las soluciones se debe realizar varias veces. Usar el concepto de dominancia y una distancia de agrupamiento, durante el proceso de selección, parece ser una forma eficiente para seleccionar soluciones no dominadas, esto reduce el costo computacional y logra obtener una buena combinación de los mejores padres con los mejores hijos.

El pseudocódigo del NSGA-II puede verse en el Algoritmo 13, (Coello Coello et al., 2007, p. 93). Una carencia a destacar de (Deb et al., 2002), documento original de este algoritmo, es que no se proporciona información sobre qué estrategia usar para el cruce y mutación de las soluciones, así como las probabilidades de las mismas.

Algoritmo 13: NSGA-II

Generar aleatoriamente la población de tamaño N
Evaluar las funciones objetivos
Asignar rangos (niveles) basados en la dominancia
Generar la población de hijos
    Selección por torneo binario
    Recombinación y mutación
while *el criterio de parada no sea cumplido* **do**
    for *cada Padre e Hijo en la población* **do**
        Asignar rangos (niveles) basados en la dominancia
        Generar conjuntos de soluciones no dominadas
        Determinar la distancia de agrupamiento (crowding)
        Ciclo (interno) que agregue soluciones a la siguiente generación empezando por el primer frente hasta obtener los N individuos
    end
    Seleccionar puntos en el frente más bajo con mayor distancia de agrupamiento
    Crear siguiente generación
        Selección por torneo binario
        Recombinación y mutación
end

- **Ventajas**
    - Al usar una técnica de ordenación de soluciones no dominadas este provee soluciones lo más cercanas posibles al frente de Pareto.
    - El uso de una técnica elitista preserva las mejores soluciones durante todas las generaciones de la ejecución, lo que impide que las soluciones que pertenecen al frente de Pareto sean eliminadas entre iteraciones.
    - La distancia de agrupamiento asegura la diversidad en las soluciones, lo que podría cubrir con mayor amplitud el frente de Pareto.
- **Desventajas**
    - Es muy eficiente con una cantidad de objetivos pequeños, esta eficiencia se aminora a medida que se aumenta el número de las funciones que se desean optimizar. (Coello Coello, 2021)
    - La comparación de agrupamiento entre rejillas puede afectar la convergencia.

3.3. Multi-Objective Particle Swarm Optimization

Este fue propuesto en (Coello Coello y Salazar Lechuga, 2002), y corresponde al nombre de «MOPSO», por sus siglas. En este se presenta la idea de extender la técnica del enjambre de partículas para tratar de resolver problemas de OMO. Una idea general del algoritmo se plantea en Algoritmo 14, (Hsieh y Hu, 2014).

Algoritmo 14: MOPSO

```
Inicializar enjambre, velocidades y la mejor posición
Inicializar archivo externo (usualmente vacío)
while el criterio de parada no sea satisfecho do
    for cada partícula do
        Seleccionar un miembro para el archivo externo (si es necesario)
        Actualizar la velocidad y posición
        Evaluar la nueva posición
        Actualizar la mejor posición y el archivo externo
    end
end
```

- **Ventajas**
  - Asegura la convergencia hacia el óptimo global, así como la rapidez en encontrarlo.
  - El algoritmo es menos dependiente de los puntos en los que inicia.
  - Posee gran desempeño en problemas donde el frente de Pareto es no diferenciable.
- **Desventajas**
  - La elección efectiva de los parámetros puede ser una tarea difícil de realizar, esto influye en la diversidad de las soluciones.
  - Debido a que toma en cuenta la mejor solución global, las soluciones tienden a concentrarse en áreas pequeñas, inclusive en un solo punto, lo cual no siempre logra determinar la amplitud total del frente de Pareto.

3.4. Implementación en Python: desarrollo de un paquete basado en el paquete Inspyred

Existen diversos paquetes, en diferentes lenguajes, para la implementación de algoritmos evolutivos; por esto se optó por no partir de cero y se escogió, de entre una vasta variedad de paquetes, utilizar el paquete *«Inspyred»*, del lenguaje Python. Se escogió este paquete por ser uno de los paquetes más completos, en cuanto a cantidad de algoritmos, posibilidad de modificación de parámetros, entre otras prestaciones del mismo; así como por las ventajas que ofrece Python.

Inspyred es un paquete realizado por Aaron Garret, del Wofford College en Carolina del Sur, E.E.U.U. Este es un modulo de código abierto gratuito para desarrollar algoritmos bioinspirados en el lenguaje Python, entre estos se incluyen computación evolutiva, inteligencia de enjambre e inmunocomputación. Para más información del modulo Inspyred, ver (Tonda, 2020) y `https://pythonhosted.org/inspyred/`

A partir del paquete «*Inspyred*» se desarrolló un paquete al que se ha llamado *«EMO»* (Evolutionary Multi-Objective Optimization). El paquete «*Inspyred*» tiene la desventaja de que la definición de ciertos parámetros que intervienen en los algoritmos es demasiado compleja; por lo anterior se creó *EMO* con la intención de hacer más fácil el uso de los algoritmos, incluso por personas que tienen poco conocimiento en programación. Así, en este paquete se generó una clase llamada «*MOOP*», la cual cuenta con los métodos de «*Inspyred*», pero agregando otras funciones a los métodos; por ejemplo:

- Para hacer más fácil la construcción y resolución de los problemas, el paquete cuenta con tres «*solver*» (los algoritmos ya mencionados); los cuales, junto con la clase, poseen algunos parámetros predefinidos, pero se permite que el usuario los modifique si desea. Esto se hizo para evitar la definición de todos los parámetros de «*Inspyred*», proceso que resultaba muy engorroso.

- Con EMO se pueden graficar algunas o todas las iteraciones de la ejecución; también es posible mostrar la evolución las soluciones sobre espacio el espacio objetivo. Esto será de gran utilidad para observar el desarrollo de los algoritmos en las implementaciones.

A continuación se presentarán los parámetros de la clase MOOP y de los «*solver*»:

Clase MOOP: Esta recoge las características propias del MOOP que se desea resolver, es decir los parámetros necesarios que permitirán definir el problema y acceder a los métodos de dicha clase. Estos son:

- `functions:` *Nombres* de las funciones objetivos, previamente definidas, entre corchetes y separadas por comas.
- `num_variables:` Número de variables que intervienen en el problema. Se debe tener cuidado con la definición de las funciones, para que concuerden con este parámetro.
- `intervals:` Intervalos factibles para cada variable; se presentan dos posibilidades:
  - **Intervalos diferentes:** Si las cotas para cada una de las variables son distintas, es necesario definir las cotas inferiores y superiores, en ese orden. Así, por ejemplo, para 4 variables: `intervals=([-1,1,0,4],[1,2,3,5])`. Donde los elementos del primer corchete representan las cotas inferiores de cada variable, y las del segundo las cotas superiores. Se debe verificar que los valores del primer grupo sean menores, en correspondencia, con los elementos del segundo grupo.
  - **Intervalos iguales:** La definición sería de este modo, a primera impresión, `intervals = ([0,0,0,0],[1,1,1,1])`, suponiendo que todas las variables tienen las mismas cotas. Lo anterior se simplificaría así: `intervals = (0,1)`
- `seed` **(opcional):** Permite establecer una semilla para la generación de los números pseudoaleatorios que intervienen en los algoritmos. Por defecto toma la semilla del reloj, es decir varia cada segundo, lo que permite variabilidad en los números pseudoaleatorios.

«Solvers»: El paquete cuenta con tres métodos de resolución de MOOP, a saber:
`solvePAES`, `solveNSGA2`, `solveMOPSO`; en los cuales se hace referencia al algoritmo utilizado. Estos cuentan con algunos parámetros en común, a saber:

- `population_size:` Tamaño de la población de soluciones que usará el algoritmo, es un valor que se mantiene estático en todas las iteraciones. Por defecto es 100.
- `maximize:` Variable booleana que permite cambiar el tipo de optimización; por defecto es `False`, es decir que se minimizan todas las funciones objetivos.
- `num_generations:` Número máximo de generaciones que debe realizar el algoritmo. Por defecto es 100.
- `mutation_rate:` Tasa de mutación. Tiene, por defecto, el valor de 0.25.
- `plot_iter:` Por defecto toma el valor de `False` (no graficar). Para graficar iteraciones, este puede tomar alguno de los siguientes valores:
  - **Graficar varias iteraciones:** Es posible graficar varias iteraciones de los algoritmos, para esto se le debe asignar el valor `plot_iter = True`. De esto, también es posible asignar un «paso» y una cota inferior a estas iteraciones. Con los siguientes parámetros:
    - `iteration_step:` Determina el paso que se debe dar para graficar las iteraciones; por defecto tiene el valor de 1, es decir grafica todas las iteraciones, desde 0 hasta `num_generations`.
    - `greater_than:` Permite establecer a partir de qué iteración empieza a graficar, por defecto es igual a 0.
  - **Graficar la última iteración:** También se puede graficar únicamente la última iteración, para esto se debe escribir `plot_iter = 'last'`.
- `objective_space:` Con esta variable es posible ver el avance de las soluciones sobre el espacio objetivo; para esto habrá que generar el espacio objetivo e incluirlo en este parámetro; por defecto este parámetro es vacío (`[ ]`).

A parte de los parámetros ya mencionados, dos de los «*solvers*» tienen parámetros específicos, propios del algoritmo que ejecutan, estos se presentan a continuación para cada uno de ellos:

- `solvePAES`
  - `num_grid:` Número de divisiones en la rejilla.
- `solveMOPSO`
  - `neighborhoods:` La topología del vecindario. Por defecto tiene el valor de 3.

- `inertia`: La constante de inercia a ser usada en la actualización de la posición de la partícula. Por defecto es 0.1.
- `cognitive`: Tasa cognitiva, que es la tasa en que la propia partícula influye sobre su propio movimiento. Por defecto tiene asignado el valor de 0.2.
- `social`: La tasa en la que las partículas vecinas influyen en el movimiento de la partícula. Es 0.15, por defecto.

3.5. Ejemplos de resolución de MOOP con EMO

A continuación se presentará la resolución de distintos MOOP usando el paquete EMO, pero antes se comentará una herramienta útil para los problemas MOOP: la generación del espacio factible. Generar el espacio factible será de utilidad para visualizar el espacio de objetivos, y así saber qué tan cerca estamos, en determinada iteración, del frente de Pareto.

3.5.1. Generación del espacio factible

La generación del espacio factible puede ser un factor importante, pero no decisivo, para visualizar el espacio de objetivos y así verificar si se ha alcanzado el frente de Pareto; esto porque hay algoritmos, como el NSGA-II, que trabajan con conjuntos de soluciones no dominadas, por lo que estas, como conjunto, se van pareciendo al frente de Pareto, pero sin llegar a serlo del todo. Más adelante se presentará gráficamente este problema.

Para la generación y visualización del espacio factible se usarán dos módulos, a saber: *numpy* y *matplotlib*. La llamada de dichos módulos se hace de la siguiente manera:

```
import numpy as np
import matplotlib.pyplot as plt
```

Podemos citar dos estrategias para generar el espacio factible: generando una rejilla con puntos igualmente espaciados, o bien, generando puntos al azar; estas se presentan a continuación.

1. **Generación de la rejilla:** Para generar el espacio completo podría considerarse una rejilla (*«mesh-grid»*) con puntos equiespaciados por cada componente. Para el caso de 2 variables, considerando el intervalo [0,1] y generando 25 puntos, en cada variable, tenemos el siguiente código para generar la rejilla:

   ```
   #Cantidad de puntos
   n = 25

   #Definiendo los puntos en las variables
   V1, V2 = np.meshgrid(np.linspace(0,1,n), np.linspace(0,1,n))
   V1 = np.repeat(V1, 1)
   V2 = np.repeat(V2, 1)

   #Graficar los puntos
   plt.plot(V1,V2,'o')
   plt.show()
   ```

   Lo que genera:

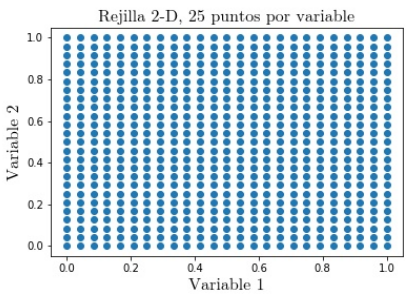


Figura 3.1: Rejilla en 2 dimensiones, 25 puntos por variable.

Puede verse que con 25 puntos en cada variable se logra cubrir muy bien el espacio factible. Considerando esta cantidad vemos que se generan $25^2 = 625$ puntos. Para el caso 3-dimensional, con la misma cantidad de puntos por coordenada, tendríamos $25^3 = 15625$ puntos, que es un número grande, pero todavía razonable.

2. **Generación de puntos al azar:** A medida que se aumenta el número de variables, realizar todas las permutaciones posibles entre los puntos generados puede ser muy costoso computacionalmente. Consideremos el caso de 30 coordenadas; si se tomasen 3 puntos del intervalo [0,1], a saber: 0, 0.5, 1, para cada variable, necesitaríamos $3^{30} = 205891132094649$ puntos para poder representar las distintas permutaciones de dichos valores y poder generar la rejilla. La cantidad de puntos por coordenada, evidentemente, es muy pequeña (3 puntos) para considerar una rejilla n-dimensional que pueda representar con certeza la región factible, y, por ende, el espacio de objetivos. Si con 3 puntos por cada coordenada se genera un número de 15 cifras, aumentando el número de puntos en el intervalo, el número de posibilidades crece exponencialmente.

   Así, para dar solución al inconveniente anterior, otra posible estrategia es generar los puntos usando números pseudoaleatorios. En este se usará la función `random` de `random` en *Numpy*. Se generarán 625 puntos (lo mismo que el ejemplo anterior) usando una distribución uniforme en el intervalo $[0, 1]$. Para generar valores en otros intervalos, con la misma distribución, puede usarse la función `uniform`.

```
#Cantidad de puntos
n = 625

#Definiendo los puntos en las variables
V1 = np.random.random(n)
V2 = np.random.random(n)

#Graficar los puntos
plt.plot(V1,V2,'o')
plt.show()
```

Este código produce:

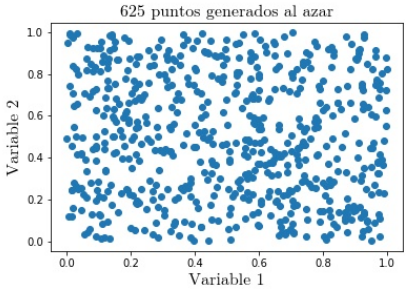


Figura 3.2: 625 puntos pseudoaleatorios, en 2 dimensiones.

Aquí se tomó en cuenta la misma cantidad de puntos que en el primer método, pero puede notarse que hay espacios con poca densidad de puntos, por lo que quizás sea necesario aumentar la cantidad de puntos. Véase con 1000 puntos.

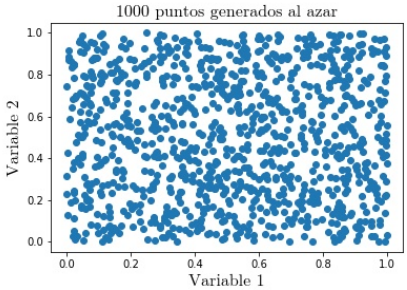


Figura 3.3: 1000 puntos pseudoaleatorios, en 2 dimensiones.

Y, aunque no se soluciona del todo el problema, puede verse que el plano queda mejor cubierto. Este ejemplo nos deja ver que para este segundo método se necesitan más valores a ser generados, pero puede ser útil para generar la región factible de un MOOP cuando este tiene más de 3 variables. Otras consideraciones respecto a desventajas de este método serán comentadas en secciones siguientes.

Generar los valores de la región factible, en los MOOP, es de gran utilidad, porque nos permitirá observar los espacios objetivos; los puntos que pertenecen a dichos espacios, en las implementaciones que se realizarán, tendrán siempre el color azul, para no confundirlos con los puntos generados por los algoritmos a tratar, que tendrán color rojo. Tómese en cuenta, también, que la visualización de los espacios objetivos es una representación aproximada, y no la representación real de los mismos.

3.5.2. Ejemplo de uso del paquete EMO

El uso del paquete EMO se presenta a continuación, considerando el siguiente problema, ya presentado en el Capítulo 1:

$$\begin{aligned} \text{minimizar} \quad & \begin{bmatrix} 0.5x_1 + x_2 \\ 1.5x_1 + x_2 \end{bmatrix} \\ \text{sujeto a} \quad & x_1 + x_2 \geq 1 \\ & 0 \leq x_i \leq 1, i \in \{1, 2\} \end{aligned}$$

Resolver dicho problema, constará de dos pasos, a saber:

1. **Definición del problema:** La definición del problema consta, también, de dos partes: la definición de las funciones objetivos (FO) y la definición del MOOP en EMO.

   - **Definición de las FO:** Lo primero que haremos será definir las funciones objetivos, nótese que el problema cuenta con restricciones sobre las variables, así como una restricción del tipo $g(x) \geq 0$, por esto se agregará una penalización sobre las soluciones no factibles.

     ```
     #Definiendo 1er. F.O.
     def f1(x):
         if x[0]+x[1] >=1:
             return 0.5*x[0] + x[1]
         else:
             return 3

     #Definiendo 2da. F.O.
     def f2(x):
         if x[0]+x[1] >=1:
             return 1.5*x[0] + x[1]
         else:
             return 3
     ```

     Algo de resaltar es el manejo de los índices en *Python*, estos inician en 0, por eso el cambio en los índices de las variables en la definición de las funciones. También, véase que se ha elegido el valor de 3, como penalización, porque el máximo valor que puede tomar $x_1 + x_2$ es 2.

   - **Definiendo el MOOP:** Una vez definidas las FO, es menester crear el problema usando la clase MOOP de EMO. Así, nuestro problema quedaría definido de la siguiente manera:

     ```
     problem = MOOP(functions = [f1,f2], num_variables = 2,
                    intervals = (0,1))
     ```

2. **Solución del problema:** Posterior a la definición del problema se procede a la solución del mismo. De secciones anteriores se sabe que se cuenta con 3 solucionadores. Así, una posible solución del problema puede realizarse con el algoritmo PAES, con 10 iteraciones y graficando la iteración final:

   ```
   problem.solvePAES(num_generations = 10,plot_iter = 'last')
   ```

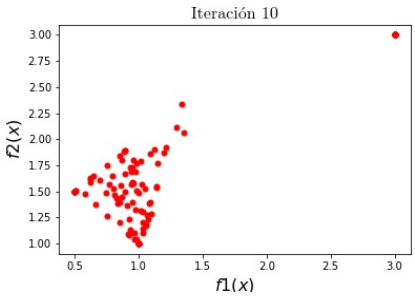


Figura 3.4: Resultado del código, algoritmo PAES en la iteración 10.

El resultado del código anterior puede verse en la Figura 3.4. Asimismo, puede observarse que las 10 iteraciones realizadas con este algoritmo son insuficientes para que las soluciones aproximen el frente de Pareto, por lo que se necesitaría realizar más iteraciones.

Es posible también generar el espacio de objetivos para ver la evolución del algoritmo y verificar si se aproxima al frente de Pareto. Para la generación de este espacio, lo que se debe hacer es generar el espacio factible y evaluar los puntos generados en las FO. En este caso, en la Figura 3.5, los puntos azules, que forman un triángulo, corresponde con la representación del espacio de objetivos del problema tratado. Asimismo, puede verse que en las primeras iteraciones hay puntos que quedan fuera del espacio de objetivos, esto se debe a que en la generación de la población no se toma en cuenta las restricciones del tipo $g(x) \geq 0$, por lo que al inicio se producen algunas soluciones no factibles que pueden estar presentes durante varias generaciones. También, es notable que los puntos del algoritmo, a medida que avanza la ejecución del mismo, se van acercando al frente de Pareto. Al final de las 1000 iteraciones pareciera que hay menos puntos, pero esto se debe a que hay puntos superpuestos (la cantidad de puntos es estática), también se aprecia que no todos los puntos son soluciones no dominadas.

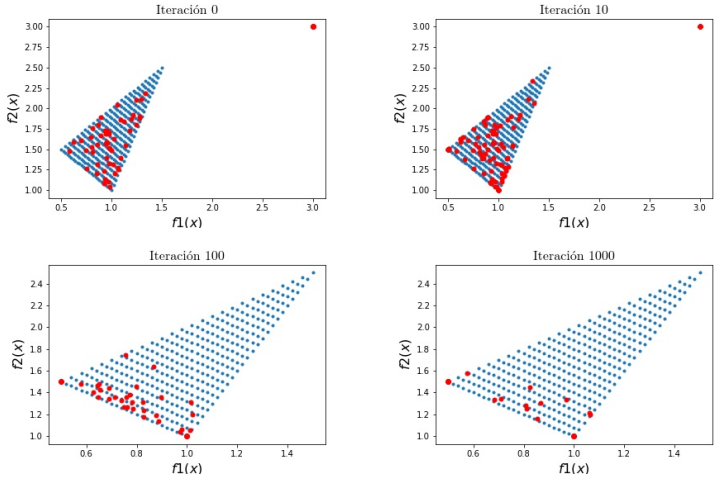


Figura 3.5: Implementación del algoritmo PAES en el problema, con el espacio objetivo.

Resolución con todos los algoritmos: Con el objetivo de realizar una comparación, se resolverá el problema anterior haciendo uso de los 3 algoritmos del paquete EMO. Así, al mismo tiempo, se analizarán los tiempos de ejecución para cada una de las iteraciones. A continuación de muestran los códigos que generan las gráficas de la Figura 3.6, respectivamente.

```
problem.solvePAES(num_generations=100,plot_iter='last')
problem.solveNSGA2(num_generations=100,plot_iter='last')
problem.solveMOPSO(num_generations=100,plot_iter='last')
```

En la Figura 3.6 pueden verse los resultados de las primeras 100 iteraciones de los algoritmos, está claro que no todas las soluciones de los algoritmos PAES y MOPSO han alcanzado el frente de Pareto, como lo han hecho las del NSGA-II. En este punto, pueda que al realizar más iteraciones, se mejoren los resultados; se incrementará a 300 iteraciones.

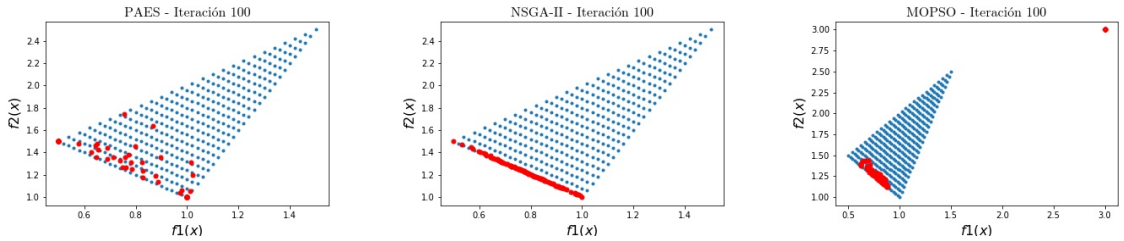


Figura 3.6: Implementaciones de los algoritmos, 100 iteraciones.

En la Figura 3.7 se ve que el algoritmo PAES, con 300 iteraciones, no logra alcanzar completamente el frente de Pareto y que las soluciones del algoritmo MOPSO se están concentrando en un área pequeña. Aunque el algoritmo NSGA-II presenta, en este caso, mejores resultados se debe tomar en cuenta que en los algoritmos PAES y MOPSO hay parámetros que deben ser calibrados, por lo que no se debe generalizar las bondades del NSGA-II y considerársele una panacea para los MOOP.

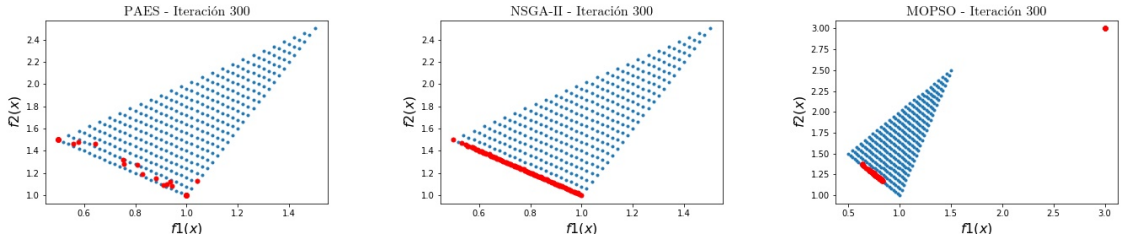


Figura 3.7: Implementaciones de los algoritmos, 300 iteraciones.

En la Figura 3.8 se pueden observar los tiempos de ejecución de los tres algoritmos respecto al número de iteraciones realizadas. Aunque MOPSO es el que logra los mejores tiempos, en esta implementación no se cambiaron los valores por defectos de los parámetros específicos del algoritmo (componente social con mayor peso que la inercia), debido a esto no logra una buena cobertura del frente de Pareto. Así, la calibración de los parámetros de los algoritmos se presenta como una dificultad a ser vencida, aunque hay que decir que para realizar dicha elección no hay estrategias universales totalmente efectivas.

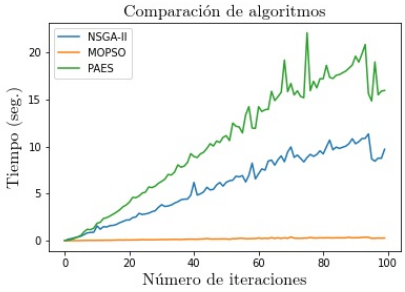


Figura 3.8: Comparación en tiempos de resolución de los 3 algoritmos.

Con el objetivo de ejemplificar el problema de la calibración de los parámetros, a continuación se presentan implementaciones del algoritmo PAES y MOPSO con distintos parámetros particulares.

Con 100 iteraciones, variando el valor del `num_grid`, se tiene que en la Figura 3.9 se presentan mejores resultados para este algoritmo que las presentadas en la Figura 3.6. La desventaja al incrementar el número de celdas de la rejilla es que también incrementa el tiempo de ejecución, unido a esto está el hecho de que no se puede determinar *a priori* un número de celdas que genere resultados aceptables.

Así mismo el algoritmo MOPSO presenta una mayor dificultad, ya que unido al problema de calibrar los 3 parámetros del algoritmo, se une, como obstáculo, determinar la cantidad de iteraciones del algoritmo. Esto se deja ver en las Figuras 3.10 y 3.11.

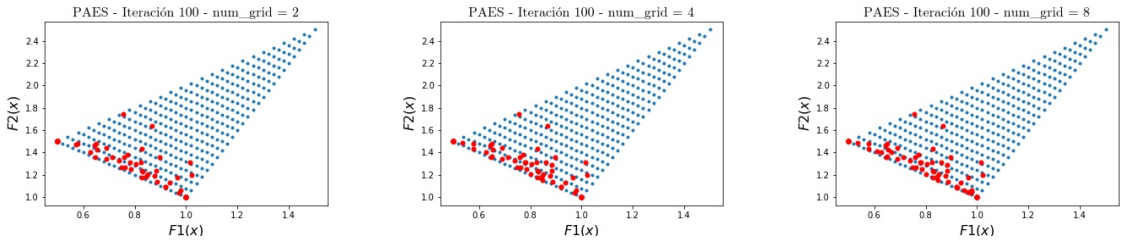


Figura 3.9: PAES, 100 iteraciones y diferentes valores de `num_grid`.

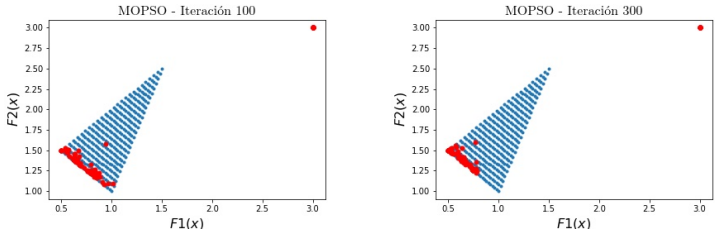


Figura 3.10: Implementaciones del algoritmo MOPSO, valores de `inertia = 0.65`, `cognitive = 0.3`, `social = 0.35`.

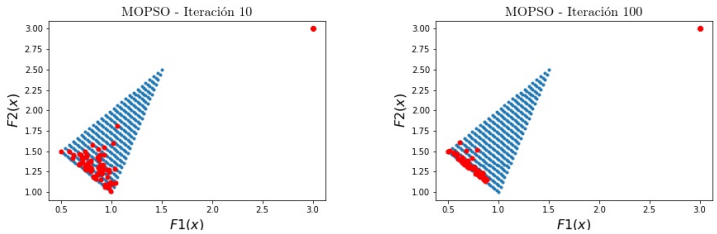


Figura 3.11: Implementaciones del algoritmo MOPSO, valores de `inertia = 0.64`, `cognitive = 0.3`, `social = 0.35`.

Nótese que, a pesar de haber cambiado los valores de los parámetros, con 100 iteraciones no se erradican las infactibilidades. La diferencia entre las Figuras 3.10 y 3.11, a parte del número de iteraciones, radica en la constante de `inertia`, variando un decimal. Y puede verse que en la Figura 3.11, con 10 iteraciones, se logra un resultado parecido al que se obtuvo en la Figura 3.10 con 100 iteraciones. Así mismo, cuando se hace crecer el número de iteraciones, las soluciones tienden a congregarse en torno un punto.

3.6. Ejemplos ilustrativos

En esta sección se presentará la resolución de algunos MOOP, iniciando con algunas funciones de prueba clásicas dentro de la literatura de la OMO, reconocidas especialmente por las dificultades que presentan, bien por la naturaleza del frente de Pareto o por la cantidad de objetivos o variables que poseen; para al final presentar un ejemplo en el cuál se verifica el problema principal de los MOOP cuando el número de objetivos es mayor que 3. Asimismo, se harán algunas comparaciones entre los algoritmos que se han elegido para las implementaciones.

Ejemplo 3.6.1 (**Función de Kursawe**). *El siguiente ejemplo se puede encontrar en (Kursawe, 1990), y es un problema que consta de 2 funciones objetivo con 3 variables de decisión. Cabe mencionar, también, que*

el espacio objetivo es no convexo.

$$\textit{minimizar} \quad \begin{bmatrix} f_1(x) = \sum_{i=1}^{2}[-10\, e^{-0.2\sqrt{x_i^2+x_{i+1}^2}}] \\ f_2(x) = \sum_{i=1}^{3}[|x_i|^{0.8} + 5\sin(x_i^3)] \end{bmatrix}$$

$$\textit{sujeto a} \quad -5 \leq x_i \leq 5, i \in \{1,2,3\}$$

En la Figura 3.12 puede verse que el que logra mejores resultados, por lo menos con la configuración actual (por defecto), sigue siendo el NSGA-II. Incluso si se realizan 1000 iteraciones, Figura 3.13, PAES y MOPSO no logran alcanzar completamente al frente de Pareto. Nótese que, con la configuración actual, MOPSO tiende a concentrarse en un solo punto.

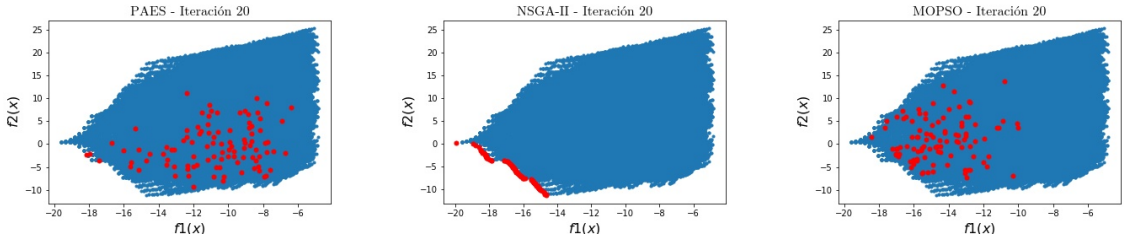


Figura 3.12: Implementaciones de los algoritmos, 20 iteraciones.

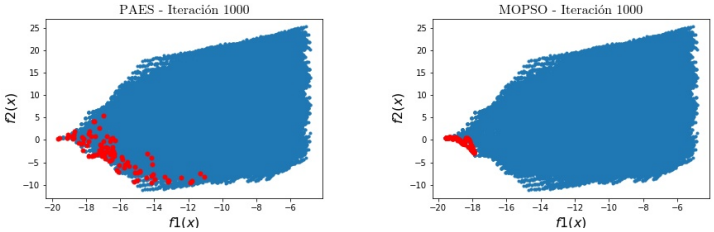


Figura 3.13: Implementaciones de los algoritmos PAES y MOPSO, 1000 iteraciones.

De igual modo, puede verse que quien logra mejores resultados, en tiempo, es el algoritmo MOPSO.

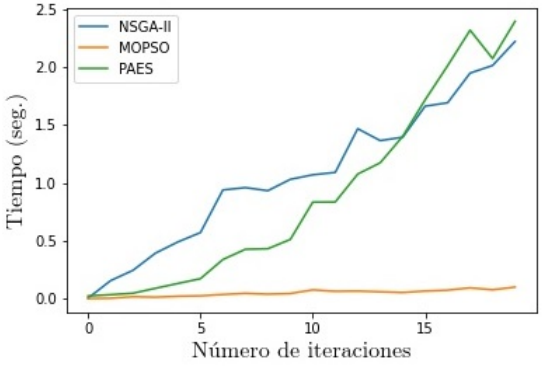


Figura 3.14: Comparación en tiempos de resolución de los 3 algoritmos.

Ejemplo 3.6.2 (Zitzler–Deb–Thiele's function N.3). *Esta función se encuentra en (Deb et al., 2002), y es una función de prueba que consta de 4 funciones objetivos y 30 variables de decisión.*

$$\begin{aligned} &\textit{minimizar} \quad \begin{bmatrix} f_1(x) = x_1 \\ f_2(x) = f_3(x) f_4(f_1(x), f_3(x)) \\ f_3(x) = 1 + \frac{9}{29} \sum_{i=2}^{30} x_i \\ f_4(f_1(x), f_3(x)) = 1 - \sqrt{\frac{f_1(x)}{f_3(x)}} - \left(\frac{f_1(x)}{f_3(x)}\right) \sin\left(10\pi f_1(x)\right) \end{bmatrix} \\ &\textit{sujeto a} \quad 0 \leq x_i \leq 1 \ \textit{donde} \ 1 \leq i \leq 30 \end{aligned}$$

Como ya se hizo ver en el apartado 3.5.1, este tipo de problemas (con gran cantidad de variables) presenta una dificultad para verificar si se ha alcanzado o no el frente de Pareto. En este caso, como se cuenta con 4 funciones objetivos, se optará por graficar las funciones confrontando dos funciones por gráfica, para darnos una idea de la evolución del algoritmo.

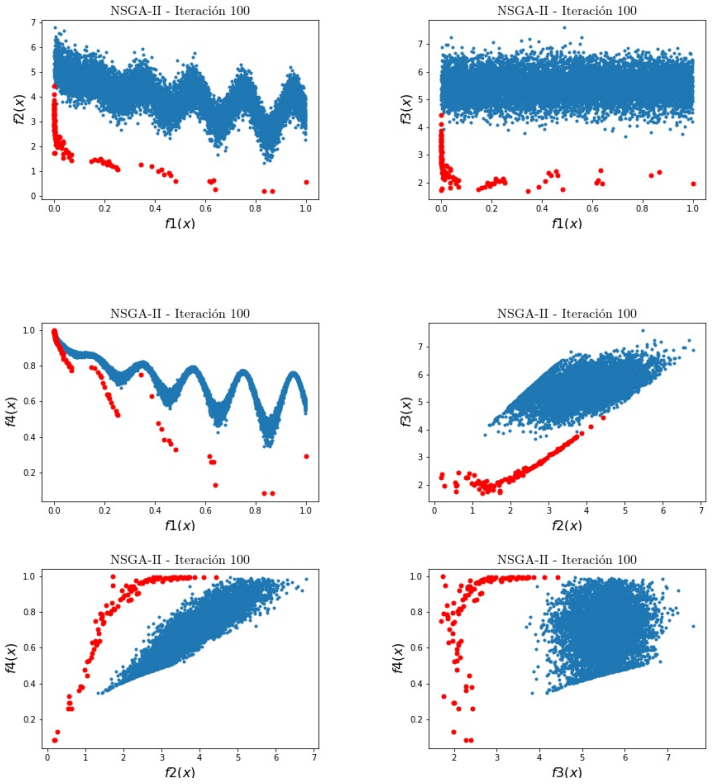
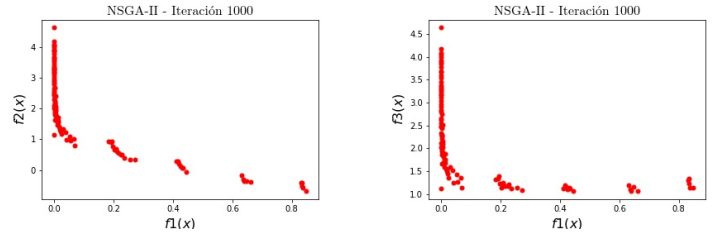


Figura 3.15: Implementaciones de NSGA-II en ZDT, 100 iteraciones.

Aunque pareciese que los puntos generados por el algoritmo se «salen» del espacio de objetivos, lo cierto es que, como ya se habrá hecho ver, la generación del espacio objetivo, a priori, es incapaz de abarcar el espacio objetivo en su extensión total. Sin embargo, podemos graficar las soluciones sin necesidad del espacio objetivo.



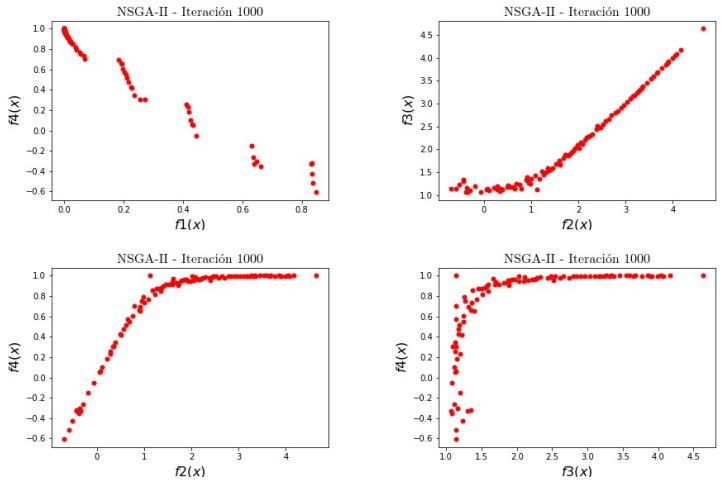


Figura 3.16: Implementaciones de NSGA-II en ZDT sin espacio objetivo, 1000 iteraciones.

Cuando se tiene una gran cantidad de FO, es posible que no se alcance el verdadero frente de Pareto, esta es una dificultad propia de los MOOP. Para ejemplificar mejor esto, se presentará el siguiente ejemplo.

Ejemplo 3.6.3. *Se considerará el siguiente MOOP con 4 funciones objetivos de una sola variable, cada una:*

$$\begin{aligned} & \textit{minimizar} \quad \begin{bmatrix} f_1(x) = 4x^2 - 4x + 1 \\ f_2(x) = x^3 \\ f_3(x) = -x + 1 \\ f_4(x) = \begin{cases} -4x + 1, & si\ 0 \leq x \leq 0.25 \\ 4x - 1, & si\ 0.25 < x \leq 0.5 \\ -4x + 3, & si\ 0.5 < x \leq 0.75 \\ 4x - 3, & si\ 0.75 < x \leq 1 \end{cases} \end{bmatrix} \\ & \textit{sujeto a} \quad 0 \leq x \leq 1 \end{aligned}$$

Nótese, a partir de la siguiente figura, que las funciones, a pesar de estar definidas en el mismo intervalo para ambas variables, no tienen sus mínimos en puntos iguales.

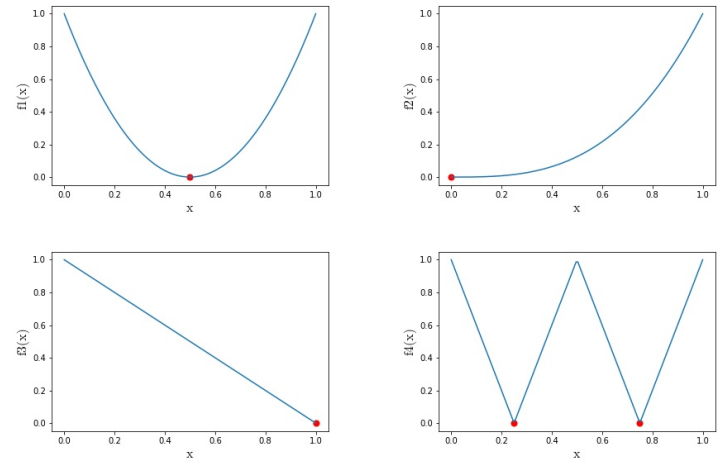


Figura 3.17: Gráfica de las funciones objetivos con sus mínimos, marcados en rojo.

Con 1000 iteraciones, puede verse en las gráficas de la Figura 3.18, que en la mayoría, las soluciones proporcionadas por el algoritmo no han podido alcanzar el frente de Pareto y que casi no hubo modificación en las soluciones, respecto a la iteración 0.

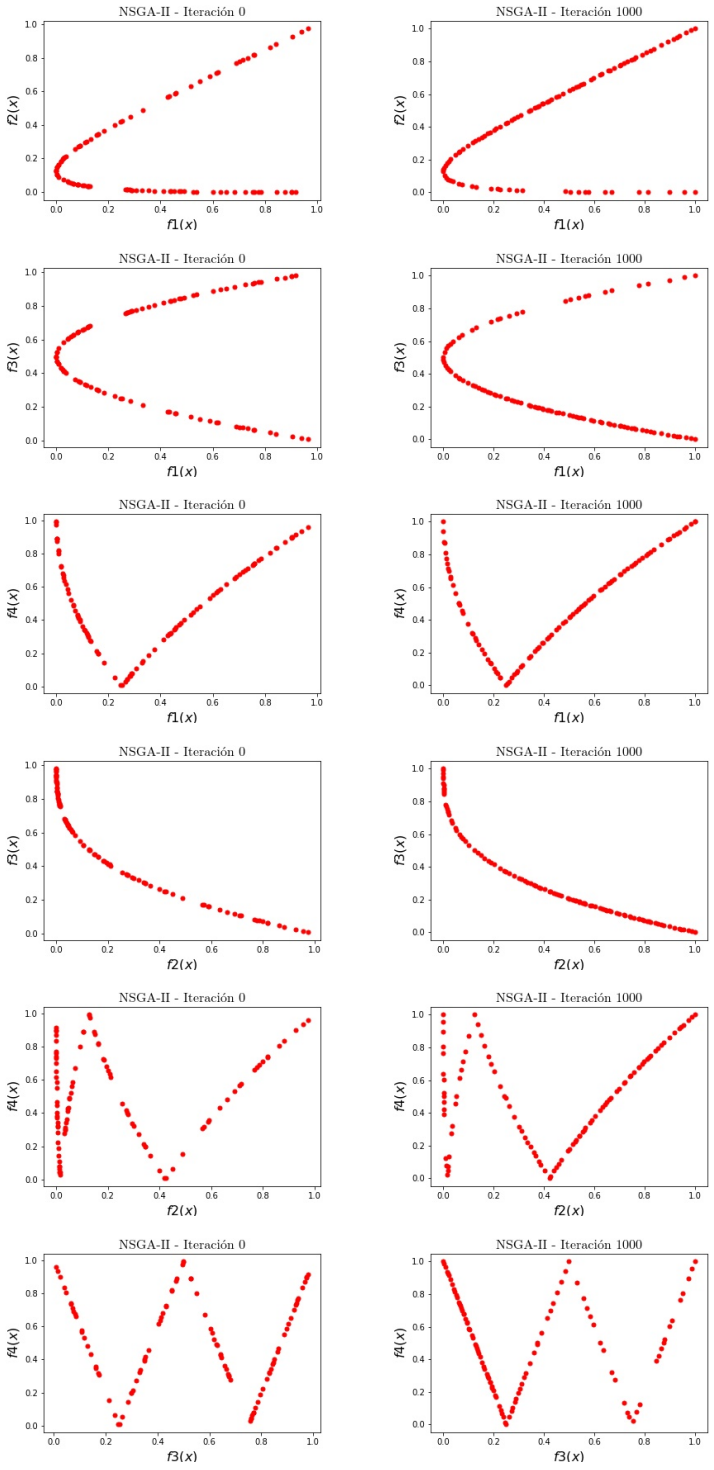


Figura 3.18: Comparación del algoritmo NSGA-II, entre la primera iteración y la iteración 1000.

La evaluación de las soluciones encontradas (puntos rojos), en cada una de las funciones objetivos, se muestra la Figura 3.19. En las gráficas de esta figura, puede notarse que la mayoría de las soluciones (x) se concentra cerca de $x = 1$ (más densidad de color rojo), que corresponde con el valor que produce el mínimo de la función $f_3(x)$. Dada la conflictividad entre las funciones, no se han minimizado simultáneamente todas las funciones; ya que es evidente que las soluciones que minimizan cada una de las funciones son valores distintos (ver Figura 3.17).

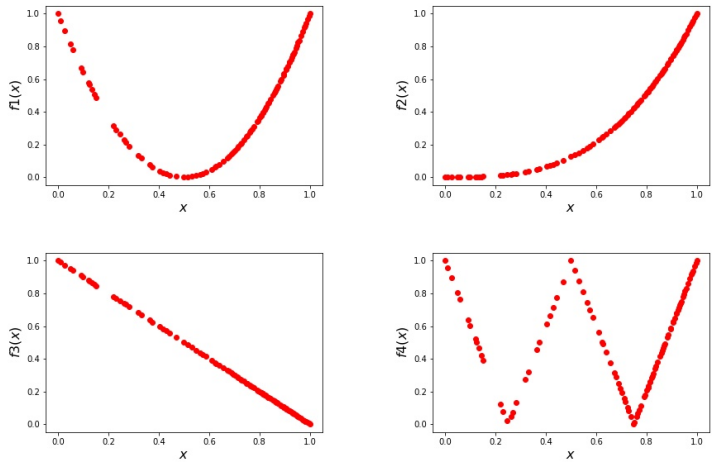


Figura 3.19: Soluciones del algoritmo evaluadas en las funciones objetivos.

Este ejemplo sirve para mostrar una de las dificultades de los MOOP, como es la de lograr optimizar todas las funciones simultáneamente cuando se tienen más de 3 objetivos; una de las estrategias para tratar de superar esta dificultad es reducir el número de objetivos. En el siguiente capítulo se abordará un poco más esta problemática y la estrategia de reducción mencionada.

Capítulo 4

Estrategias de reducción de objetivos

Los MOOP, por su naturaleza, demuestran tener varias dificultades cuando se intenta darles solución; por esto, se iniciará presentando algunos de estos obstáculos para, posteriormente, centrarnos en uno, la cantidad de objetivos y, se presentará, una de las estrategias para lidiar con esta dificultad cuando el número de objetivos es mayor que 3.

4.1. Dificultades de los MOOP

La mayoría de los algoritmos desarrollados para el tratamiento de los MOOP trabajan muy bien con problemas que tienen 2 o 3 objetivos, pero cuando se incrementa el número de estos (siempre que sean objetivos en conflicto entre sí), la capacidad de alcance del frente de Pareto se ve seriamente afectada. Los MOOP que tienen más de 3 objetivos usualmente se les asigna el nombre de problemas de optimización «*Many-Objetivos*» (MaOP). Algunas de las dificultades que se presentan cuando se desea resolver MaOP son, (Cf. Ishibuchi et al., 2008):

1. A medida que se incrementa el número de objetivos, la cantidad de soluciones que se necesitan para representar todo el frente de Pareto crece exponencialmente, por lo que en ciertas situaciones pueden necesitarse miles de soluciones para esto.

2. Al tratarse de más de 3 objetivos, la visualización de las soluciones se torna un inconveniente, esto dificulta la elección de una única solución entre las soluciones que se encuentran en el frente de Pareto.

3. El incremento del número de objetivos implica que casi todas las soluciones en la población tiendan a ser no dominadas, lo que afecta directamente a los algoritmos y su capacidad de búsqueda.

La última dificultad, en la que nos centraremos, se verifica en la Figura 4.1, donde se tomó como referencia 200 puntos (vectores) generados al azar en el hipercubo unidad $[0, 1]^m$, y se aprecia la evolución del porcentaje de vectores no dominados al incrementar el número de objetivos (m). Asimismo, esta dificultad ha sido tratada en varios estudios como: (Khare et al., 2003), (Purshouse y Fleming, 2003), (Ishibuchi et al., 2006).

Un muestra de esta dificultad lo tenemos en el Ejemplo 3.6.3, donde se ejemplifica el inconveniente que

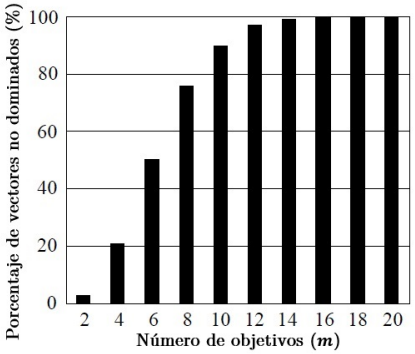


Figura 4.1: Porcentaje de soluciones no dominadas, respecto al número de objetivos. (Ishibuchi et al., 2008)

presentan los MaOP. Un acercamiento a la resolución de este tipo de problemas se puede generar a partir de la pregunta: ¿son necesarios todos los objetivos o se puede prescindir de alguno, o algunos? Así, a partir de los conflictos generados en los MaOP, y con mente en la pregunta formulada, en el siguiente capítulo se pretende hacer una revisión de las estrategias para la reducción de objetivos.

4.2. Estrategia de Brockhoff y Zitzler

La estrategia de los autores Brockhoff y Zitzler recopilada en (Brockhoff y Zitzler, 2006), nos presenta un método en el cual sea usa una medida que permite determinar los cambios en la estructura de dominancia, con el objetivo de medir la calidad del conjunto de soluciones no dominadas encontrada. Asimismo, se presenta un tipo de problemas llamado *problema del subconjunto mínimo de objetivos* (MOSS, por sus siglas en inglés), el cuál pretende mostrar qué funciones objetivos son esenciales para la resolución del MOOP original.

4.2.1. Medida para los cambios en la estructura de dominancia

Se asume que la estructura de dominancia subyacente está definida por la dominancia débil, esta se define a continuación:

Definición 4.2.1 (**Relación de dominancia**).

$$\preceq_{\mathcal{F}'} := \{(x,y) | x,y \in \mathbb{R}^n \wedge \forall f_i \in \mathcal{F}' : f_i(x) \leq f_i(y)\}$$

donde $\mathcal{F}$ es el conjunto original de objetivos, de modo que $\mathcal{F}' \subset \mathcal{F} := \{f_1, f_2, \ldots, f_m\}$.

Se dirá que x domina débilmente a y con respecto al conjunto de objetivos $\mathcal{F}'$ si $(x,y) \in \preceq_{\mathcal{F}'}$. La idea de la relación de dominancia es buscar un conjunto $\mathcal{F}' \subset \mathcal{F} := \{f_1, f_2, \ldots, f_m\}$ que preserve las relaciones de dominancia entre las soluciones. Es decir, $x \preceq_{\mathcal{F}'} y \Leftrightarrow x \preceq_{\mathcal{F}} y, \forall x, y \in \mathbb{R}^n$.

Definición 4.2.2 (**Relación ε-dominancia aditiva**).

$$\preceq^{\varepsilon}_{\mathcal{F}'} := \{(x,y) | x,y \in \mathbb{R}^n \wedge \forall f_i \in \mathcal{F}' \subset \mathcal{F} : f_i(x) - \varepsilon \leq f_i(y)\}$$

El valor de ε es una medida del error que se comete al tomar un subconjunto propio de los objetivos, cuando este error es el mayor posible se denota δ.

Ejemplo 4.2.1. *En este ejemplo, en (Brockhoff y Zitzler, 2006, p. 535), se toman tres soluciones, a saber: x_1 (línea sólida), x_2 (línea discontinua) y x_3 (línea punteada), son soluciones no dominadas.*

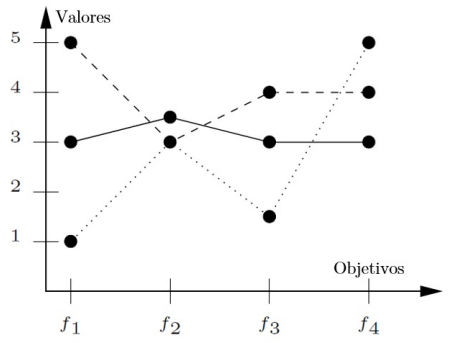


Figura 4.2: Coordenadas paralelas graficadas para 3 soluciones y 4 objetivos. (Brockhoff y Zitzler, 2006)

- ***Dominancia:*** *Uno de los acercamientos que se puede realizar para la reducción de los objetivos es verificar aquellos objetivos en los que la relación de dominancia sea redundante. Por ejemplo, puede verse que $\preceq_{f_1}$ y $\preceq_{f_3}$ son la misma: $x_3 \preceq_{f_1} x_1 \preceq_{f_1} x_2$ así como $x_3 \preceq_{f_3} x_1 \preceq_{f_3} x_2$. Así mismo, se puede verificar que $\{f_1, f_2, f_4\}$ y $\{f_2, f_3, f_4\}$ son subconjuntos mínimos de objetivos que preservan la estructura de dominancia entre las soluciones, ya que cuando se toma alguno de estos subconjuntos, estas continúan siendo no dominadas.*

- ***Error:*** *Cuando se realiza una reducción de objetivos, puede que la dominancia cambie, así considérese:* $\mathcal{F}' := \{f_3, f_4\}$*, de esto puede verse que* $x_1 \preceq_{\mathcal{F}'} x_2$*, pero sabemos que* $x_1 \not\preceq_{\mathcal{F}} x_2$*. La reducción del conjunto original de objetivos, en este caso, conlleva un error; para que la estructura se preserve y sigan siendo no dominadas a* x_1 *se le debe restar* $\delta = 0.5$*, para que se cumpla que* $x_1 \not\preceq_{\mathcal{F}'} x_2$*.*

A continuación se presentan algunas de las definiciones referente al error y conflictividad entre conjuntos de objetivos.

Definición 4.2.3. *Sean* $\mathcal{F}_1$ *y* $\mathcal{F}_2$ *dos conjuntos de objetivos. Se define* $\mathcal{F}_1 \sqsubseteq^{\delta} \mathcal{F}_2 :\Leftrightarrow \preceq_{\mathcal{F}_1} \subseteq \preceq^{\delta}_{\mathcal{F}_2}$*.*

Definición 4.2.4. *Sean* $\mathcal{F}_1$ *y* $\mathcal{F}_2$ *dos conjuntos de objetivos. Se nombran:*

- $\mathcal{F}_1$ *es* δ*-no conflictivo con* $\mathcal{F}_2$ *si y solo si* $(\mathcal{F}_1 \sqsubseteq^{\delta} \mathcal{F}_2) \wedge (\mathcal{F}_2 \sqsubseteq^{\delta} \mathcal{F}_1)$*;*
- $\mathcal{F}_1$ *es* δ*-conflictivo con* $\mathcal{F}_2$ *si y solo si* $\neg(\mathcal{F}_1$ *es* δ*-no conflictivo con* $\mathcal{F}_2)$*;*

Basados en las definiciones anteriores de conflictividad entre conjuntos de objetivos, se pueden definir los conjuntos δ-minimal y δ-mínimo.

Definición 4.2.5. *Sea* $\mathcal{F}$ *un conjunto de objetivos y* $\delta \in \mathbb{R}$*. Un conjunto de objetivos* $\mathcal{F}' \subseteq \mathcal{F}$ *es denotado como:*

- δ***-minimal*** *con respecto a* $\mathcal{F}$ *si y solo si:*
  1. $\mathcal{F}'$ *es* δ*-no conflictivo con* $\mathcal{F}$*,*
  2. $\mathcal{F}'$ *es* δ'*-conflictivo con* $\mathcal{F}$ *para todo* $\delta' < \delta$*,*
  3. *no existe* $\mathcal{F}'' \subset \mathcal{F}'$ *que sea* δ*-no conflictivo con* $\mathcal{F}$*.*
- δ***-mínimo*** *con respecto a* $\mathcal{F}$ *si y solo si:*
  1. $\mathcal{F}'$ *es* δ*-minimal con respecto a* $\mathcal{F}$*, y*
  2. *no existe* $\mathcal{F}'' \subset \mathcal{F}'$ *con* $|\mathcal{F}''| < |\mathcal{F}'|$ *que sea* δ*-minimal con respecto a* $\mathcal{F}$*.*

Definición 4.2.6. *Un conjunto* $\mathcal{F}$*, de objetivos, se dice* δ*-redundante si y solo si existe* $\mathcal{F}' \subset \mathcal{F}$ *que es* δ*-minimal con respecto a* $\mathcal{F}$*.*

4.2.2. Conjuntos mínimos de objetivos

A continuación se presentan dos tipos de problemas relacionados al **subconjunto mínimo de objetivos** (MOSS).

Definición 4.2.7 (Subconjunto mínimo de objetivos (MOSS))**.** *Dado un MOOP, se desea computar un conjunto de funciones* $I \subseteq \{f_1, f_2, \ldots, f_m\}$ *de cardinalidad mínima, de modo que* $\cap_{f_i \in I} \preceq_{f_i} = \preceq_{\mathcal{F}}$*.*

De esto puede verse, que se busca determinar un conjunto de objetivos cuya cardinalidad sea mínima y que preserve la relación de dominancia del conjunto inicial de funciones objetivo. Un algoritmo que soluciona este problema lo encontramos en Algoritmo 15. Cabe destacar que este será empleado en las implementaciones de este capítulo.

En el Algoritmo 15, X es el conjunto de soluciones; $\sqcup$ es la unión de dos subconjuntos de objetivos definida como: $S_1 \sqcup S_2 := \{s_1 \cup s_2 | s_1 \in S_1 \wedge s_2 \in S_2 \wedge (\nexists p_1 \in S_1, p_2 \in S_2 : p_1 \cup p_2 \subset s_1 \cup s_2)\}$

Teniendo como base el problema MOSS, se plantean los siguientes:

Definición 4.2.8 (δ-mínimo subconjunto de objetivos (δ-MOSS))**.** *Dado un MOOP y los vectores objetivos* $f(x_1), f(x_2), \ldots, f(x_m) \in \mathbb{R}^m$ *de las soluciones* $x_1, x_2, \ldots, x_m \in A \subseteq \mathcal{S}$ *y un* $\delta \in \mathbb{R}$*, el objetivo es computar un subconjunto de objetivos* $\mathcal{F}' \subseteq \mathcal{F}$ *que sea* δ*-mínimo con respecto* $\mathcal{F}$*.*

Definición 4.2.9 (Subconjunto mínimo de objetivos de cardinal k con error mínimo (k-EMOSS))**.** *Dado un MOOP, y los vectores objetivos* $f(x_1), f(x_2), \ldots, f(x_m) \in \mathbb{R}^m$ *de las soluciones* $x_1, x_2, \ldots, x_m \in A \subseteq \mathcal{S}$ *y un* $k \in \mathbb{R}$*, el objetivo es calcular un subconjunto de objetivos* $\mathcal{F}' \subseteq \mathcal{F}$ *de modo que* $|\mathcal{F}'| \leq k$ *y sea* δ*-no conflictivo con* $\mathcal{F}$ *con el minimal posible* δ*.*

Algoritmo 15: Un algoritmo exacto para MOSS

Iniciar: $S := \emptyset$
for *cada par de soluciones* $x, y \in X$ **do**
  $S_x := \{\{i\} | i \in \{1, \ldots, k\} \wedge x \preceq_i y \wedge y \not\preceq_i x\}$
  $S_y := \{\{i\} | i \in \{1, \ldots, k\} \wedge y \preceq_i x \wedge x \not\preceq_i y\}$
  $S_{xy} := S_x \sqcup S_y$
  if $S_{xy} = \emptyset$ **then**
    $S_{xy} := \{1, \ldots, k\}$
  end
end
Output: el conjunto más pequeño s_{min} en S

4.3. Estrategia de Deb y Saxena

En (Deb y Saxena, 2005), estos autores proponen una estrategia de reducción de objetivos, la meta es determinar la dimensión más baja que permita definir el verdadero frente de Pareto; para esto usan la estrategia de Análisis de Componentes Principales (PCA, por sus siglas en inglés) unido al algoritmo NSGA-II para hacer que las soluciones se vayan acercando, iterativamente, al frente de Pareto de más baja dimensión.

4.3.1. Análisis de componentes principales (PCA)

Es una herramienta del análisis multivariado, que permite reducir la dimensión de un conjunto de datos con número grande de variables interrelacionadas. Esta estrategia permite recoger la mayor cantidad posible de variación de los datos originales en un conjunto de variables nuevas incorreladas y ordenadas, las cuales reciben el nombre de *componentes principales*.

El conjunto inicial de datos es representado en la forma de una matriz

$$D = (D_1, D_2, \ldots, D_M)^T,$$

donde D_i representa el i-ésimo dato. El número de datos (M) es la dimensión del conjunto de datos, de donde se tiene que las columnas de la matriz D es un vector del espacio $M-$dimensional. Para realizar el PCA, estos datos deben estar estandarizados, lo que quiere decir que el centroide de todo el conjunto de datos debe ser cero; es decir, debe generarse una nueva matriz, a saber $\mathrm{X} = (X_1, X_2, \ldots, X_M)^T$ con los datos estandarizados. Una manera de identificar los objetivos redundantes puede ser calcular las matrices de covarianza (V) o la de correlación (R):

$$\text{Matriz de covarianza (V): } V_{ij} = \frac{X_i X_j^T}{M-1}$$

$$\text{Matriz de correlación (R): } R_{ij} = \frac{V_{ij}}{\sqrt{V_{ii} * V_{jj}}} \tag{4.1}$$

Aquí, las componentes principales no son más que los vectores propios de cualquiera de estas dos matrices, aunque, según los autores, usar la matriz de correlación presenta mayores ventajas. El vector propio correspondiente al máximo valor propio recibe el nombre de primera componente principal, y así sucesivamente.

4.3.2. PCA y MOOP

A partir de un MOOP de m objetivos que se resuelve, con algún algoritmo, usando una población de n individuos, se puede construir una matriz inicial de datos de dimensión $m \times n$, la cual se convierte a su forma estandarizada y se calculan su matriz de correlación y sus valores propios, procurando que estos aparezcan en orden descendente. Los coeficientes de cada componente principal expresan la contribución relativa de cada objetivo. Un valor positivo muestra un incremento en el valor del objetivo al desplazarse a través de esa componente principal y un decremento si el valor es negativo. Entonces, si se consideran los objetivos que correspondan al mayor de los positivos y al menor de los negativos, tendríamos elementos que tienen la

máxima contribución para un incremento o un decremento en la componente principal, obteniendo los dos objetivos más conflictivos de la componente principal.

Un factor importante a definir en este método es el *umbral de corte* (TC), este es un valor «máximo» para la varianza explicada y cuando la contribución acumulada sobrepase el TC no seguimos analizando más componentes principales. Así, por ejemplo si se tiene un MOOP y se elige un $TC = 95\,\%$, y se determina que la primera componente principal contribuye un $96\,\%$ de la variabilidad, entonces nos detenemos.

Una dificultad que se añade a este método es la elección de TC, ya que elegir un TC muy bajo podría dejar por fuera algunos objetivos que son importantes en el problema; así, también, si se escoge un valor muy alto (cercano a $100\,\%$) se pueden escoger objetivos redundantes que entorpezcan la solución del problema. Sin embargo, (Deb y Saxena, 2005), nos recomienda usar un TC cercano al $95\,\%$, ya que este permite una menor posibilidad de elección de objetivos redundantes y de pérdida de objetivos importantes.

Algoritmo 16: Elección de objetivos no redundantes - Deb y Saxena

Para la primera componente principal, tomar los objetivos con el mayor valor positivo y el menor negativo en la misma.

Para la demás componentes principales:

- **if** *el correspondiente valor propio es menor o igual que* 0.1 **then**
  tomar el mayor de los valores absolutos del vector propio.
- **if** *el correspondiente valor propio es mayor que* 0.1 *y la contribución acumulada es menor que TC:* **then**

```
if todos los valores son positivos then
    tomar el objetivo con el mayor valor.
else if todos los valores son negativos then
    tomar todos los objetivos.
else
    Considerar p al mayor de los valores positivos y a n el menor de los negativos.
    if p < |n|: then
        if p ≥ 0.9|n| then
            tomar los dos objetivos correspondientes a p y a n.
        else
            tomar el objetivo correspondiente a n
        end
    else if p > |n|: then
        if p ≥ 0.8|n| then
            tomar los objetivos correspondientes a p y a n.
        else
            tomar el objetivo correspondiente a p.
        end
end
```

A continuación se presentan los procedimientos que se definen en (Deb y Saxena, 2005), y que son útiles para reducir la cantidad de objetivos.

- **Elección de objetivos no redundantes:** El Algoritmo 16 nos permite determinar los objetivos no redundantes, a partir de las evaluaciones de las funciones de *«fitness»* de determinada población. Así, este nos permitirá determinar los objetivos que presentan mayor conflictividad entre si, y con base en esto determinar qué objetivos son necesarios para la resolución del MOOP.

- **Reducción final:** Con suerte, el procedimiento anterior identifica la mayoría de los objetivos redundantes. Para considerar si es posible una mayor reducción en el número de objetivos, consideramos la matriz de correlación reducida (solo columnas y filas correspondientes a objetivos no redundantes) e investigamos si todavía existe un conjunto de objetivos con coeficientes de correlación positivos o negativos idénticos con otros objetivos y teniendo una correlación positiva entre ellos. Esto sugeriría que cualquier miembro de dicho grupo sería suficiente para establecer las relaciones conflictivas con los objetivos restantes. En tal caso, retenemos el que fue elegido con anterioridad (correspondiente al valor propio más grande) por el análisis de PCA. Otros objetivos del conjunto no se consideran más.

- **PCA-NSGA-II:** El procedimiento descrito en el Algoritmo 17 inicia con m objetivos e iterativamente encuentra un conjunto de objetivos no redundantes, a partir del análisis de soluciones generadas por un algoritmo de resolución de MOOP. Cuando no es posible reducir más la cantidad de objetivos, el procedimiento se detiene y declara que ha encontrado el conjunto final de objetivos y el correspondiente frente de Pareto.

  Así mismo, los autores del procedimiento advierten que este no tiene mucho sentido cuando los MOOP presentan un frente de Pareto m-dimensional, ya que en este caso el algoritmo determinará que todos los objetivos son importantes; pero a pesar de esto, puede ser de utilidad para el tomador de decisiones, dado que el algoritmo presenta un orden relativo de importancia entre los objetivos.

Algoritmo 17: Procedimiento general PCA-NSGA-II

Paso 1: Establecer un contador de iteracion $t = 0$ y el conjunto inicial de objetivos, $\mathbb{I}_0 = \{1, 2, \ldots, M\}$

Paso 2: Inicializar una población aleatoria para todos los objetivos en el conjunto $\mathbb{I}_t$, ejecutar un algoritmo para resolver MOOP, y obtener una población P_t

Paso 3: Realizar PCA sobre la población P_t para escoger un conjunto de objetivos $\mathbb{I}_{t+1}$ usando el TC predefinido. Los pasos para el PCA son los siguientes:

1. Calcular la matriz de correlación usando la Ecuación 4.1
2. Calcular los valores y vectores propios y escoger los objetivos no redundantes usando el Algoritmo 16
3. Reducir aún más el número de objetivos, si es posible, utilizando los coeficientes de correlación de los objetivos no redundantes que se encuentran en el inciso anterior, 2., y en líneas anteriores

Paso 4:

if $\mathbb{I}_{t+1}$ **then**
detenerse y declarar que se ha obtenido el frente de Pareto.
else
definir $t = t + 1$ e ir al **Paso 2**
end

4.4. Aplicaciones de las estrategias

En esta sección se presentarán las características y sintaxis del paquete «EMO», al cuál se le integraron los algoritmos de las estrategias de Brockhoff & Zitzler y la de Deb & Saxena. También se presentarán algunas experimentaciones en las que se hizo uso del paquete para reducir la dimensión de unos MOOP particulares.

4.4.1. Características del paquete «EMO»

- **Estrategia de Brockhoff & Zitzler:** Aunque el algoritmo de los autores no lo menciona, aquí consideramos usar un algoritmo de solución (NSGA-II) para generar un conjunto de soluciones de Pareto, a partir del cuál se tomará una «n» cantidad de individuos (se pueden tomar todos); sobre estos individuos se implementará el Algoritmo 15. A continuación se presenta la sintaxis de la implementación de esta estrategia en el paquete:

  ```
  MOSS_Exact(population, functions, n)
  ```

  Donde:

  - `population:` Es el conjunto de soluciones de Pareto.
  - `functions:` Son las funciones objetivos del MOOP, es el conjunto que se quiere reducir.
  - `n:` Es la cantidad de soluciones que se tomarán, aleatoriamente, de la población para ejecutar el algoritmo. Si este valor no se especifica, se toman todas las soluciones de `population`.

  Con inspiración en el problema $\delta-$MOSS, se creó la función «`Compute_Delta_Pop`», la cuál calcula el error, desde la $\varepsilon-$conflictividad, para todos los posibles subconjuntos del conjunto original de FO. Su sintaxis es: `Compute_Delta_Pop(population)`

- **Estrategia de Deb** & **Saxena:** Como lo propone el Algoritmo 17, una vez generada la población de individuos no dominados usando el NSGA-II, se implementa PCA y se obtienen el conjunto de objetivos no redundantes. La sintaxis para la implementación de este algoritmo en EMO es:

```
Obj_NonRed(population, functions, TC)
```

Donde:

  - `population:` Es el conjunto de soluciones de Pareto generadas con NSGA-II.
  - `functions:` Son las funciones del MOOP cuyo conjunto de FO se quiere reducir.
  - `TC:` Es el umbral de corte mencionado en la estrategia.

Como un método auxiliar, que puede ser de interés, es la función `«PCAs»`, con esta se obtiene el resultado del PCA aplicado a la población, su sintaxis es: `PCAs(population)`. Esta nos retorna: vectores y valores propios, varianza explicada y matriz de covarianza.

4.4.2. Experimentaciones

A continuación se presentarán algunas experimentaciones, en las cuales utilizaremos las estrategias descritas en este capítulo y que fueron añadidas al paquete «EMO». Cabe mencionar, que en estas se fijó la probabilidad de cruce en 0.9, y la probabilidad de mutación en 0.1. Asimismo, sobre las poblaciones que se usarán, para la estrategia de Brockhoff & Zitzler, se aplicará el algoritmo *«Simple Cull»* (Yukish, 2004), para asegurarnos de tener una población de soluciones no dominadas.

Ejemplo 4.4.1. *Consideremos el MOOP presentado en el capítulo anterior en el* ***Ejemplo 3.6.3****.*

Para este problema se usó una población de 100 individuos, los cuales se generaron usando el algoritmo NSGA-II con 100 iteraciones sobre el MOOP en cuestión.

- ***Estrategia de Deb*** & ***Saxena:*** *Aplicando PCA a la población obtenida en la iteración final, tenemos los siguientes datos:*

Matriz de correlación			
1.	*0.66523457*	*-0.34247732*	*0.40687077*
0.66523457	*1.*	*-0.92705033*	*0.26924361*
-0.34247732	*-0.92705033*	*1.*	*-0.11721421*
0.40687077	*0.26924361*	*-0.11721421*	*1.*
Valores propios			
6.02779281e+00	*1.04106164e+00*	*2.73192504e-01*	*3.38980748e-06*
Varianza explicada			
82.09958428	*96.27902451*	*99.99995383*	*100.*
Eigenvectores			
0.49200518	*0.32692132*	*-0.75314631*	*0.28952373*
0.62025524	*-0.23064076*	*0.01694162*	*-0.74953403*
-0.53474267	*0.46842723*	*-0.37477369*	*-0.59512258*
0.29540586	*0.78772027*	*0.54039643*	*0.01427796*

En la estrategia se recomienda usar un umbral de corte del $95\,\%$*, pero en este usaremos un umbral de corte del* $97\,\%$*, para ampliar nuestro rango de elección de objetivos. Así, de la primera componente principal (eigenvector) tomamos el mayor valor y el menor valor (esta componente explica* $82.09\,\%$ *de la variabilidad), dichos valores son:* 0.62025524 *y* -0.53474267*, lo que indica que* $f_2(x)$ *y* $f_3(x)$*, son las funciones con la mayor conflictividad entre ellas. De igual modo, pasamos a la segunda componente, ya que no hemos alcanzado el umbral de corte; en esta se toman el mayor valor positivo* (p) *y el menor valor negativo* (n)*, si* $p \geq 0.8|n|$ *entonces tomamos los objetivos relacionados con* p *y* n*, en caso contrario solo se toma* p*. Así, los valores máximo y mínimo de la segunda componente son:* 0.78772027 *y* -0.23064076*, respectivamente, lo que indica que* $f_2(x)$ *y* $f_4(x)$ *son las dos siguientes funciones con la mayor conflictividad. Así, tenemos que el conjunto de funciones no redundantes (con* $TC = 97\,\%$*) son:* $\{f_2(x), f_3(x), f_4(x)\}$

De lo anterior, se tiene que las tres funciones encontradas son funciones no redundantes; en donde, según esta estrategia, no podemos prescindir de ninguna de ellas para poder resolver el MOOP en cuestión.

Para verificar la elección del algoritmo se determinarán los objetivos no redundantes de este problema usando el paquete «EMO», el cual trae integrado Algoritmo 17 que nos permite encontrar el conjunto funciones objetivos no redundantes. Prescindiremos del código de las funciones para ahorrar espacio.

```
# Importar el algoritmo PCA-NSGA-II
from EMO.MOOP import Obj_NonRed

# Se definen las funciones: f1,f2,f3,f4

# Input: Individuos, lista de funciones y el TC a elegir
Obj_NonRed(population, ['f1','f2','f3','f4'],97)
```

El resultado del código anterior es el conjunto de objetivos no redundantes, en este caso particular: `{'f2', 'f3', 'f4'}`.

Cabe destacar que la elección del TC en este ejercicio se hizo con la intención de que el algoritmo eligiera 3 objetivos; sin embargo, tomando el valor propuesto para el umbral de corte, $TC = 95\,\%$, se tiene que los objetivos no redundantes son: `{'f2','f3'}`.

- ***Estrategia de Brockhoff** & **Zitzler:** Usando la misma población (final, generada con NSGA-II) que se usó en la estrategia anterior:*

```
# Importar el algoritmo MOSS_Exact
from EMO.MOOP import MOSS_Exact

# Se definen las funciones: f1,f2,f3,f4

# Input: Individuos y lista de funciones.
MOSS_Exact(population, ['f1','f2','f3','f4'])
```

El algoritmo da como resultado el subconjunto mínimo de las funciones objetivos donde se preserva la estructura de dominancia, en este caso que todas las soluciones sigan siendo soluciones de Pareto. El resultado de dicha estrategia es: `{'f2','f3'}`

A partir de este ejemplo, puede verse que, en la estrategia de Deb & Saxena, el umbral de corte (TC) es un parámetro importante, ya que si este es muy alto podrían incluirse objetivos que son redundantes. A su vez, podemos observar que esta estrategia no toma en cuenta la estructura de dominancia, por lo que una posible estrategia para reducir la cantidad de objetivos es:

1. Aplicar la estrategia de Brockhoff & Zitzler sobre una población de individuos no dominados.
2. Aplicar la estrategia de Deb & Saxena con $TC = 95\,\%$ a la misma población.
3. Del resultado del paso 1 tomar aquellos subconjuntos de objetivos en los que estén presentes los objetivos que resulten del paso 2.

Esta estrategia nos asegurará dos cosas, que la estructura de dominancia se preserve y que entre los objetivos considerados se encuentren los objetivos que tienen mayor conflictividad.

Ejemplo 4.4.2. *Aquí se considera un problema con 5 funciones objetivos y 3 variables de decisión. La función $f_1(x)$ fue tomada de (Deb et al., 2002, p. 110); se puede notar que esta es la única función del problema, ya que las demás son múltiplos de esta. Es un ejemplo sencillo, en el que para resolver dicho problema se debe optimizar una única función del conjunto de funciones del problema.*

$$\begin{aligned} \textit{minimizar} \quad & \begin{bmatrix} f_1(x) = 1 - \exp\Big(-\sum_{i=1}^{3}\big(x_i - \frac{1}{\sqrt{3}}\big)^2\Big) \\ f_2(x) = 3f_1(x) \\ f_3(x) = 2f_1(x) \\ f_4(x) = 5f_1(x) \\ f_5(x) = f_3(x) + f_4(x) \end{bmatrix} \\ \textit{sujeto a} \quad & -4 \leq x_i \leq 4, \forall i \in \{1,2,3\}. \end{aligned}$$

Para hacer el análisis de las estrategias se usó una población de 100 individuos, generados a partir de 100 iteraciones del algoritmo NSGA-II sobre el MOOP en cuestión.

- ***Estrategia de Deb*** & ***Saxena:*** *A continuación se presenta el resultado de la aplicación del PCA sobre la población utilizada.*

Matriz de correlación				
1.	*1.*	*1.*	*1.*	*1.*
1.	*1.*	*1.*	*1.*	*1.*
1.	*1.*	*1.*	*1.*	*1.*
1.	*1.*	*1.*	*1.*	*1.*
1.	*1.*	*1.*	*1.*	*1.*
Valores propios				
2.50e+01	*3.94e-31*	*2.59e-33*	*0.00e+00*	*-7.40e-16*
Varianza explicada				
100.	*100.*	*100.*	*100.*	*100.*
Eigenvectores				
-0.4472136	*-0.37080945*	*-0.89349044*	*-0.89442719*	*0.2236068*
-0.4472136	*0.72905581*	*0.22478718*	*0.2236068*	*0.2236068*
-0.4472136	*-0.23605021*	*0.25475246*	*0.2236068*	*0.2236068*
-0.4472136	*0.30482018*	*0.22384413*	*0.2236068*	*-0.89442719*
-0.4472136	*-0.42701632*	*0.19010666*	*0.2236068*	*0.2236068*

Aunque al aplicar esta estrategia da como resultado: `{'f1'}`*. Se puede observar que en la primera componente en todas las filas se tiene el mismo valor, el algoritmo programado en Python, ante este tipo de situaciones, siempre tomará la primera función de la lista (f_1 en este caso).*

- ***Estrategia de Brockhoff*** & ***Zitzler:*** *Esta estrategia nos genera el siguiente resultado:* `{'f1'}, {'f2'}, {'f3'}, {'f4'}, {'f5'}`*.*

En este caso podemos ver que con las dos estrategias se llega a la conclusión de que podemos resolver el problema optimizando una única función objetivo; aunque en la primera estrategia nos dice que tomemos la función f_1, se puede constatar que para resolver el problema se puede optimizar cualquiera de las funciones objetivos, como nos asegura la estrategia de Brockhoff & Zitzler, ya que es evidente que cada una de las funciones f_2, f_3, f_4, f_5 son la función f_1 multiplicada por una constante.

Ejemplo 4.4.3. *Consideraremos el siguiente MOOP, cabe mencionar que las funciones $f_1(x)$ y $f_2(x)$ corresponden a un problema de optimización multiobjetivo presentado en (Deb et al., 2002, p. 110), llamado problema «FON».*

$$\begin{aligned} \textit{minimizar} \quad & \begin{bmatrix} f_1(x) = 1 - \exp\Big(-\sum_{i=1}^{3}\big(x_i - \frac{1}{\sqrt{3}}\big)^2\Big) \\ f_2(x) = 1 - \exp\Big(-\sum_{i=1}^{3}\big(x_i + \frac{1}{\sqrt{3}}\big)^2\Big) \\ f_3(x) = 2f_2(x) \\ f_4(x) = 5f_1(x) \\ f_5(x) = f_3(x) + f_4(x) \end{bmatrix} \\ \textit{sujeto a} \quad & -4 \leq x_i \leq 4, \forall i \in \{1,2,3\}. \end{aligned}$$

Como puede verse, las funciones $f_3(x), f_4(x)$ y $f_5(x)$ son funciones que dependen de f_1 y f_2; esperaríamos que las estrategias elijan a estas últimas como las únicas funciones necesarias para resolver el MOOP.

Para generar la población de soluciones de Pareto, se utilizó el algoritmo NSGA-II con distintas cantidades de generaciones, estas se especificarán en cada estrategia.

- ***Estrategia de Deb*** & ***Saxena:*** *Se usó una población de 100 individuos generadas con 50 iteraciones, se aplicó PCA sobre estas y se obtuvieron los siguientes resultados.*

Varianza explicada				
99.42070532	*100.*	*100.*	*100.*	*100.*
Eigenvectores				
0.46357897909137613	*0.06877893*	*-0.46527532*	*-0.88338213*	*-0.08978524*
-0.44844479768446005	*0.4307593*	*-0.33514164*	*-0.20179536*	*0.33832306*
-0.4484447976844599	*0.4307593*	*-0.33558202*	*-0.20179536*	*-0.7287711*
0.46357897909137635	*0.06877893*	*-0.58961198*	*0.24863081*	*-0.52429579*
0.40985801	*0.78703988*	*0.45928356*	*0.27636208*	*0.26736252*

Siguiendo esta estrategia, tenemos que la varianza explicada alcanza el 99.42% en la primera componente, por lo que no se deben analizar más componentes; aquí debemos tomar el máximo y el mínimo de los valores en dicha componente (valores de las filas 2 y 4); estos corresponde con el subconjunto de funciones objetivos: $\{f_2, f_4\}$. Cabe resaltar que se realizaron varias experimentaciones en las cuales se varió el tamaño de la población; en todas se obtuvo el mismo subconjunto de objetivos.

- ***Estrategia de Brockhoff** & **Zitzler:** En este ejemplo se observará que la población sobre la cuál se aplicará la estrategia es decisiva para determinar el subconjunto mínimo de objetivos no redundantes.*
  - *Se usó una población de 100 individuos generados con 20 iteraciones de NSGA-II; se realizaron varias ejecuciones de esta estrategia sobre esta población, en cada una de las cuales se tomaron 10 individuos aleatoriamente y se aplicó esta estrategia sobre estos. En las sucesivas ejecuciones se obtenía como resultado alguno de los siguientes subconjuntos de objetivos:*
    - $\{f_1, f_2\}, \{f_1, f_3\}, \{f_2, f_4\}, \{f_3, f_4\}$
    - $\{f_1, f_2\}, \{f_1, f_3\}, \{f_2, f_4\}, \{f_3, f_4\}, \{f_2, f_5\}, \{f_3, f_5\}$

    Cabe destacar al aplicar esta estrategia sobre 100 individuos generados con 50 iteraciones de NSGA-II, se produjeron los mismos resultados.
  - *Ahora, se considerará una población de 10 individuos generados usando NSGA-II con 50 iteraciones, se aplicó la estrategia sobre toda la población. Se obtuvo como resultado: $\{f_1, f_2\}$, $\{f_1, f_3\}$, $\{f_2, f_4\}$, $\{f_3, f_4\}$*

  De lo anterior se deduce que en algoritmo de esta estrategia, la población y el tamaño de la misma son elementos fundamentales para que obtengamos los verdaderos subconjuntos de objetivos no redundantes.

Para confirmar los resultados se usó la función `Compute_Delta_Pop`; esta nos da información del error (δ) que se cometería al resolver el MOOP con determinados subconjuntos de funciones objetivos.

δ	Objetivos	δ	Objetivos	δ	Objetivos
0	$\{f_1, f_2, f_3, f_4, f_5\}$	0.45	$\{f_2, f_3, f_5\}$	0.22	$\{f_2, f_5\}$
0	Con 4 objetivos	0	$\{f_2, f_4, f_5\}$	0	$\{f_3, f_4\}$
0	$\{f_1, f_2, f_3\}$	0	$\{f_3, f_4, f_5\}$	0.45	$\{f_3, f_5\}$
0	$\{f_1, f_2, f_4\}$	0	$\{f_1, f_2\}$	2.51	$\{f_4, f_5\}$
0	$\{f_1, f_2, f_5\}$	0	$\{f_1, f_3\}$	0.84	$\{f_1\}$
0	$\{f_1, f_3, f_4\}$	2.51	$\{f_1, f_4\}$	0.70	$\{f_2\}$
0	$\{f_1, f_3, f_5\}$	1.19	$\{f_1, f_5\}$	1.40	$\{f_3\}$
2.51	$\{f_1, f_4, f_5\}$	1.39	$\{f_2, f_3\}$	2.51	$\{f_4\}$
0	$\{f_2, f_3, f_4\}$	0	$\{f_2, f_4\}$	1.19	$\{f_5\}$

En la tabla anterior, aquellos subconjuntos donde $\delta = 0$ la estructura de dominancia se preserva (siguen siendo soluciones no dominadas). Algo de notar es que los subconjuntos $\{f_2, f_5\}$ y $\{f_3, f_5\}$, generados como subconjuntos no redundantes en la estrategia de Brockhoff & Zitzler (en algunas ejecuciones) tienen un error distinto de cero, por lo que si resolviéramos el MOOP usando alguno de estos subconjuntos no nos aseguraríamos que la estructura de dominancia se preserve.

Capítulo 5

Conclusiones

La optimización es un área de las matemáticas de gran importancia, por sus resultados teóricos y prácticos. En esta se desarrollan estrategias para resolver problemas de optimización; muchos sectores del quehacer humano hacen más eficientes sus procesos resolviendo casos particulares de este tipo de problemas, usando herramientas matemáticas desarrolladas en optimización. He ahí la importancia de esta área de las ciencias, ya que cada vez más se incrementa la necesidad de métodos que resuelvan eficientemente dichos problemas.

A grandes rasgos, para dar solución a los problemas de optimización tenemos dos enfoques: los métodos determinísticos y las metaheurísticas (no determinísticos). En el primer tipo de métodos se usan herramientas del cálculo para encontrar las soluciones óptimas, esto hace que se requiera que los problema a tratar con estos cumplan condiciones especiales como convexidad, linealidad, entre otras. Para superar estas limitaciones se cuenta con las metaheurísticas; estos son métodos en los que, a diferencia de los anteriores, intervienen factores aleatorios (pseudoaleatorios) para obtener las soluciones óptimas del problema. En contraste con los métodos determinísticos, las metaheurísticas no nos aseguran encontrar las soluciones exactas, en muchas ocasiones se encontrarán aproximaciones de estas.

Un tipo particular de problema de optimización se presenta cuando se quiere optimizar, simultáneamente, varias funciones; este tipo de problemas recibe el nombre de **problemas optimización multiobjetivo (MOOP, por sus siglas en inglés)**; cuando el número de objetivos es mayor a 3 se les conoce como **«Many-Objectives problems» (MaOP)**. Una característica que hace especialmente difícil de resolver los MaOP es que, muchas veces, las funciones objetivos entran en conflicto, por lo que no es posible alcanzar simultáneamente todas las soluciones óptimas de cada objetivo; ante tal situación no es posible encontrar una solución única, por lo que se obtiene como resultado un conjunto de soluciones que, al compararlas entre sí, ninguna es mejor que las otras al evaluarlas en las funciones objetivos; a este conjunto se le conoce como «frente de Pareto». Debido a esta conflictividad entre objetivos no existen métodos de validez general, sin embargo las metaheurísticas, como trabajan con poblaciones de soluciones, dan buenos resultados para aproximar dicho frente.

Como ya se aseveró, para la resolución de los MOOP, se requiere de las metaheurísticas; estas, a su vez, implican el uso de computadores para la ejecución de sus algoritmos y, de este modo, obtener resultados en tiempos bastante razonables. Es por esto, que el desarrollo de algoritmos de resolución de los MOOP es fundamental para la OMO; lo que se desea son algoritmos cada vez más eficientes (en este caso, que en menor tiempo se logre mayor proximidad al frente de Pareto y que, al mismo tiempo, se logré representar la amplitud del mismo). Debido a la amplia gama de MOOP que se pueden presentar, es necesario contar con herramientas que tengan a disposición diversos algoritmos (recuérdese que hay algoritmos más eficientes para ciertos tipos de MOOP), cuya ejecución no sea difícil de realizar. Así, partiendo de paquetes realizados en el lenguaje Python (a saber, *«Inspyred»*), se desarrolló el paquete *EMO*. Este cuenta con tres algoritmos de resolución de MOOP (PAES, MOPSO y NSGA-II), así como dos estrategias de reducción de objetivos (Deb & Saxena y Brockhoff & Zitzler). Una de las características del paquete creado es que permite la ejecución de los distintos algoritmos de una forma bastante sencilla, útil incluso para aquellas personas que no cuentan con mucho conocimiento de programación. Es por esto, que este paquete jugó un papel importante en la implementación de los algoritmos, tanto de resolución como de reducción de objetivos, de los MOOP tratados en este trabajo.

Existen problemas que, por sus características, afectan el desempeño de los algoritmos evolutivos para la

generación de las soluciones del frente de Pareto, obteniendo falsos frentes de Pareto. Una de las estrategias para abordar esta problemática es la reducción de la cantidad original de funciones objetivos, para quedarnos con aquellas que determinen el verdadero frente de Pareto. Respecto a esto último, se consideraron dos estrategias, cuyas conclusiones se presentan a continuación:

- **Estrategia de Deb & Saxena (Deb y Saxena, 2005):**
  - En esta se determina la conflictividad entre los objetivos usando el análisis de componentes principales (PCA).
  - El algoritmo de esta estrategia debe aplicarse sobre una población de soluciones, las cuales son generadas con el algoritmo NSGA-II. Respecto a esto, no se menciona una estrategia para determinar la cantidad de soluciones que se deben usar y la cantidad de iteraciones a ejecutar en el algoritmo.
  - Un parámetro importante en esta estrategia es el «umbral de corte» (TC), este determinará la cantidad de variabilidad que deseamos contabilizar con PCA. Los autores recomiendan usar el 95 %, aunque quizás dicho valor no sea apropiado para todos los MOOP que se quieran tratar.
  - Es una estrategia que no toma en cuenta que la estructura de dominancia entre las soluciones se preserve.
- **Estrategia de Brockhoff & Zitzler (Brockhoff y Zitzler, 2006):**
  - Para medir la conflictividad entre los objetivos se usa la dominancia entre las soluciones.
  - Se propone el problema subconjunto mínimo de objetivos (MOSS), en este se busca un subconjunto mínimo de objetivos que preserve la estructura de dominancia. Es decir, partiendo de un conjunto de soluciones no dominadas, se busca un subconjunto de objetivos (el más pequeño) en el cual las soluciones sigan siendo no dominadas.
  - A partir del problema anterior se propone el problema δ-MOSS, aquí se considera una medida de error, cuando los objetivos seleccionados no preservan la estructura de dominancia. Es decir, se puede resolver el problema con un subconjunto de objetivos que no preserven la estructura de dominancia, pero estaríamos cometiendo un error, δ.
  - No se específica cómo generar la población sobre la que se va a aplicar el algoritmo de esta estrategia. Esto es un obstáculo a vencer, ya que el error δ varía respecto a la población utilizada; esto no permite determinar con exactitud el error que se comete al tomar subconjuntos de objetivos donde el error sea distinto de 0.

Las estrategias de reducción de objetivo buscan reducir la cantidad de objetivos sin afectar la precisión de las soluciones a encontrar, es decir que estas buscan un subconjunto propio de objetivos, del conjunto original, que represente el frente de Pareto del MOOP con el que se está tratando. Estas representan, quizás, un primer enfoque para resolver los MOOP (omitiendo los métodos agregativos). Y se puede afirmar, que su importancia radica en que reducir la cantidad de objetivos, en diversos problemas, mejorará el desempeño de los algoritmos empleados, esto logrará que se obtengan resultados en menor tiempo y con mayor precisión.

Existen casos en el que todos los objetivos son «importantes», por ende no se puede prescindir de ninguno de ellos. En estos casos también se puede realizar reducción de objetivos, a costa de disminuir la precisión de búsqueda; ya que introducimos un error. En este punto sería útil contar con más medidas que permitan determinar el error que se comete al quitar ciertos objetivos (importantes); una de estas medidas la encontramos en δ-MOSS, sin embargo esta medida varía dependiendo de la población que se use para estimarla. Por otro lado, si no se quiere afectar la precisión de las soluciones a encontrar, al quitar objetivos importantes, se deberá buscar otra estrategia de resolución.

Como ya se mencionó, las estrategias presentadas se deben aplicar sobre una población de soluciones, uno de los inconvenientes es determinar el tamaño de la misma, de esto dependerá si se hace o no una correcta reducción de objetivos; pueda que si se considera una población muy pequeña o demasiado grande, al final de la ejecución de los algoritmos, se incluyan objetivos redundantes. Determinar el tamaño idóneo de la población es un parámetro a estimar, ya que el conjunto de soluciones sobre la que se ejecutan los algoritmos de las estrategias de reducción condicionan los resultados.

Bibliografía

[1] Attia, A., Ghaithan, A. y Duffuaa, S. *A multi-objective optimization model for tactical planning of upstream oil & gas supply chains. Computers & Chemical Engineering*, 128(2): pp.216–227, 2019. doi: https://doi.org/10.1016/j.compchemeng.2019.06.016.

[2] Branke, J., Deb, K., Miettinen, K. y Slowinski, R. *Multiobjective Optimization. Interactive and Evolutionary Approaches.* Springer-Verlag Berlin Heidelberg, 2008.

[3] Brockhoff, D. y Zitzler, E. *Are All Objectives Necessary? On Dimensionality Reduction in Evolutionary Multiobjective Optimization. Parallel Problem Solving from Nature - PPSN IX. Lecture Notes in Computer Science*, 4193: pp.533–542, 2006. doi: https://doi.org/10.1007/11844297_54.

[4] Carballo Mato, J. *Programación Lineal Multiobjetivo y Aplicaciones.* Trabajo de fin de máster, Universidad Santiago de Compostela & Universidad da Coruña & Universidad de Vigo, abril 2019.

[5] Chong, E. y Zak, S. *An introduction to optimization.* Wiley & Sons, inc., 4ta. edición, 2013.

[6] Coello Coello, C. A. *Introducción a la Computación Evolutiva (Notas de Curso)*, 2021. URL `http://delta.cs.cinvestav.mx/~ccoello/compevol/apuntes.pdf`.

[7] Coello Coello, C. A. y Salazar Lechuga, M. *MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 2: pp.1051–1056, 2002. doi: 10.1109/CEC.2002.1004388.

[8] Coello Coello, C. A., Lamont, G. B. y Van Veldhuizen, D. A. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer Science+Business Media, LLC, 2007.

[9] Cui, Y., Geng, Z., Zhu, Q. y Han, Y. *Multi-objective optimization methods and application in energy saving. Energy*, 125: pp.681–704, 2017. doi: https://doi.org/10.1016/j.energy.2017.02.174.

[10] Deb, K. *An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering*, 186(2-4): pp.311–338, 2000. doi: https://doi.org/10.1016/S0045-7825(99)00389-8.

[11] Deb, K. *Multi-Objective Optimization using Evolutionary Algorithm.* Wiley & Sons, Ltd., 2001.

[12] Deb, K. y Saxena, D. K. *On finding Pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems.* Technical report, 2005.

[13] Deb, K., Pratap, A., Agarwal, S. y Meyarivan, T. *A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput.*, 6(2): pp.182–197, 2002. doi: 10.1109/4235.996017.

[14] Deb, K., Thiele, L., Laumanns, M. y Zitzler, E. *Scalable multi-objective optimization test problems. Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 1: pp.825–830, 2002. doi: https://doi.org/10.1109/CEC.2002.1007032.

[15] Dreo, J., Pétrowski, A., Siarry, P. y Taillard, E. *Metaheuristics for Hard Optimization. Methods and Case Studies.* Springer-Verlag Berlin Heidelberg, 2006.

[16] Garrett, A. Inspyred, 2015. URL `https://pypi.org/project/inspyred/`.

[17] Goldberg, D. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley publishing company, Inc., 1989.

[18] Guenin, B., Könemann, J. y Tunçel, L. *A Gentle Introduction to Optimization.* Cambridge University Press, 2014.

[19] Gunasekara, R. C., Mohan, C. y Mehrotra, K. *Multi-objective Optimization to Improve Robustness in Networks.* En Mandal, J., Mukhopadhyay, S. y Dutta, P., editores, *Multi-Objective Optimization*, pp.266–290. Springer, Singapore, 2018. doi: https://doi.org/10.1007/978-981-13-1471-1_5.

[20] Haimes, Y. Y., Lasdon, L. S. y Wismer, D. A. *On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Transactions on Systems, Man, and Cybernetics*, 1(3): pp.296–297, 1971. doi: https://ieeexplore.ieee.org/document/4308298.

[21] Hsieh, C.-T. y Hu, C.-S. *Fingerprint Recognition by Multi-objective Optimization PSO Hybrid with SVM. Journal of Applied Research and Technology*, 12, 2014. doi: 10.1016/S1665-6423(14)71662-1.

[22] Ishibuchi, H., Nojima, Y. y Doi, T. *Comparison between single-objective and multi-objective genetic algorithms: Performance maesures. 2006 IEEE International Conference on Evolutionary Computation*, pp.1143–1150, 2006. doi: doi:10.1109/CEC.2006.1688438.

[23] Ishibuchi, H., Tsukamoto, N. y Nojima, Y. *Evolutionary many-objective optimization: A short review. 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp.2419–2426, 2008. doi: 10.1109/CEC.2008.4631121.

[24] Keller, A. *Multi-Objective Optimization In Theory and Practice I: Classical Methods.* Bentham Science Publishers, 2017.

[25] Khare, V., Yao, X. y Deb, K. *Performance scaling of multi-objective evolutionary algorithms. Evolutionary Multi-Criterion Optimization. EMO 2003. Lecture Notes in Computer Science,*, 2632: pp.367–390, 2003. doi: https://doi.org/10.1007/3-540-36970-8_27.

[26] Knowles, J. y Corne, D. W. *Approximating the non-dominated front using the Pareto archived evolution strategy. Evolutionary Computation Journal*, 8(2): pp.149–172, 2000. doi: 10.1162/106365600568167.

[27] Kramer, O. *Genetic Algorithm Essentials.* Springer International Publishing, 2017.

[28] Kursawe, J. *A variant of evolution strategies for vector optimization.* En Schwefel, H. y Männer, R., editores, *Parallel Problem Solving from Nature. PPSN 1990. Lecture Notes in Computer Science, vol 496.*, pp.193–197. Springer, Berlin, Heidelberg., 1990. doi: https://doi.org/10.1007/BFb0029752.

[29] Lyons, M. *Pretomanid dose selection for pulmonary tuberculosis: An application of multi-objective optimization to dosage regimen design. CPT: Pharmacometrics Syst Pharmacol*, 10(3): pp.211–219, 2021. doi: https://doi.org/10.1002/psp4.12591.

[30] Mahfouf, M., Chen, M. Y. y Linkens, D. A. *Adaptive weighted particle swarm optimisation for multi-objective optimal design of alloy steels. Lecture notes in computer science. Springer*, 3242: pp.762–771, 2004. doi: https://doi.org/10.1007/978-3-540-30217-9_77.

[31] Mandal, J., Mukhopadhyay, S. y Dutta, P. *Multi-Objective Optimization Evolutionary to Hybrid Framework.* Springer, 2018.

[32] Miettinen, K. *Nonlinear multiobjective optimization.* Springer Science+Business Media, 2004.

[33] Mouayad, A. S. y Bestoun, S. A. *A new multiobjective performance criterion used in PID tuning optimization algorithms. Journal of Advanced Research*, 7(1): pp.125–134, 2016. doi: https://doi.org/10.1016/j.jare.2015.03.004.

[34] Mousavie-Avval, S., Rafiee, S., Sharifi, M., Hosseinpour, S., Notarnicola, B., Tassielli, G. y Renzulli, P. A. *Application of multi-objective genetic algorithms for optimization of energy, economics and environmental life cycle assessment in oilseed production. Journal of Cleaner Production*, 140(2): pp.804–815, 2017. doi: https://doi.org/10.1016/j.jclepro.2016.03.075.

[35] Parsopoulos, K. E. y Vrahatis, M. N. *Recent approaches to global optimization problems through particle swarm optimization. Natural computing*, 1(2-3): pp.235–306, 2002. doi: https://doi.org/10.1023/A:1016568309421.

[36] Pascual-González, J., Jímenez-Esteller, L., Guillén-Gosálbez, G., Siirola, J. J. y Grossmann, I. E. *Macro-Economic Multi-Objective Input–Output Model for Minimizing CO_2 Emissions: Application to the U.S. Economy. AIChE*, 62(10): pp.3639–3656, 2016. doi: https://doi.org/10.1002/aic.15376.

[37] Purshouse, R. C. y Fleming, P. J. *Evolutionary many-objective optimization: An exploratory analysis. The 2003 IEEE Congress on Evolutionary Computation*, 3(CEC 03): pp.2066–2073, 2003. doi: 10.1109/CEC.2003.1299927.

[38] Sarker, R. y Newton, C. *Optimization Modelling: A Practical Approach.* CRC Press, 2008.

[39] Srinivas, N. y Deb, K. *Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation*, 2(3): pp.221–248, 1994. doi: https://doi.org/10.1162/evco.1994.2.3.221.

[40] Tang, K., Chan, T., Yin, R. y Man, K. *Multiobjective optimization methodology. A jumping gene approach.* CRC Press, 2012.

[41] The Luc, D. *Multiobjective Linear Programming. An Introduction.* Springer, 2016.

[42] Tonda, A. *Inspyred: Bio-inspired algorithms in Python. Genet Program Evolvable Mach*, 21: pp.269–272, 2020. doi: https://doi.org/10.1007/s10710-019-09367-z.

[43] Yann, C. y Siarry, P. *Multiobjective Linear Programming. Principles and Cases Studies.* Springer-Verlag Berlin Heidelberg, 2003.

[44] Yazdi, J., Sadollah, A., Lee, E., Yoo, D. y Kim, J. *Application of multi-objective evolutionary algorithms for the rehabilitation of storm sewer pipe networks. Journal of Flood Risk Management*, 10(3): pp.326–338, 2015. doi: https://doi.org/10.1111/jfr3.12143.

[45] Yukish, M. *Algorithms to Identify Pareto Points in Multi-Dimensional Data Sets.* Ph.D. dissertation, Universidad del estado de Pensilvania, 2004.

[46] Yu, X. y Mitsuo, G. *Introduction to Evolutionary Algorithms.* Springer-Verlag Lonon Limites, 2010.