Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Un Algoritmo Evolutivo Multiobjetivo basado en Hipervolumen en GPUs

Tesis que presenta

Edgar Manoatl Lopez

para obtener el Grado de

Maestro en Ciencias

en Computación

Director de la Tesis

Dr. Carlos Artemio Coello Coello

México, D.F. **Noviembre 2013**

Resumen

El hipervolumen (o métrica $\mathcal{S}$) se ha vuelto extremadamente popular hoy en día en el área de optimización multi-objetivo. Debido a sus buenas propiedades matemáticas, se ha utilizado no sólo como indicador de desempeño para comparar los resultados finales de los algoritmos evolutivos multi-objetivo (AEMOs), sino también como criterio de selección de los optimizadores multi-objetivo. Sin embargo, calcular el valor exacto del hipervolumen es sumamente costoso (computacionalmente hablando); de hecho, es un problema NP-duro. Los mejores algoritmos conocidos para este fin, calculan el valor del hipervolumen en un tiempo de ejecución que es polinomial conforme al número de puntos, pero que crece exponencialmente con el número de objetivos. Por lo tanto, ya que el tiempo computacional es crucial para el redimiendo de los algoritmos evolutivos, el alto costo del hipevolumen limita su uso, especialmente cuando el número de objetivos de un problema es elevado. No obstante, una ventaja del hipervolumen es que la clasificación de soluciones que produce es invariante al escalamiento lineal de las funciones objetivo, por lo que su uso sería prometedor para problemas de optimización con muchos objetivos si su costo computacional no lo hiciese prohibitivo.

En esta tesis se propone un nuevo enfoque para utilizar el hipervolumen como criterio de selección mediante el uso de Unidades de Procesamiento Gráfico (GPUs por sus siglas en inglés), el cual calcula de una manera más rápida la contribución al hipervolumen de un punto específico. Llevamos a cabo una implementación altamente paralela de nuestro enfoque y demostramos su rendimiento cuando se utiliza con el *$\mathcal{S}$-Metric Selection Evolutionary Multi-Objective Algorithm* (SMS-EMOA). Los resultados muestran que el enfoque alcanza una aceleración de hasta 883x con respecto a su versión secuencial, haciendo posible el uso del hipervolumen exacto en problemas de hasta nueve objetivos.

Abstract

The hypervolume (or $\mathcal{S}$-metric) has become extremely popular nowadays within multi-objective optimization. Because of its good mathematical properties, it has been used not only as a performance indicator to compare the final results obtained by multi-objective evolutionary algorithms (MOEAs), but also as a selection criterion in multi-objective optimizers. However, computing the exact hypervolume value is very expensive (computationally speaking); in fact, it is an NP-hard problem. The best algorithms known today for this task compute the hypervolume value in an execution time that is polynomial with respect to the number of points, but that grows exponentially with the number of objectives. Therefore, and since the computational time is crucial to the performance of evolutionary algorithms, the high cost of the hypervolume limits its use, particularly when the number of objectives of a problem is high. Nevertheless, an advantage of the hypervolume is that the classification of solutions that it produces, is invariant to the linear scaling of the objective functions, which would make it an attractive choice for dealing with optimization problems having many objectives, if its computational cost did not turn it prohibitive.

In this thesis, we propose a new approach for the use of the hypervolume as a selection criterion, based on the use of Graphical Processing Units (GPUs). The proposed approach computes in a faster manner the hypervolume contribution of a specific point. We also provide a highly parallel implementation of our approach and show its performance using the *$\mathcal{S}$-Metric Selection Evolutionary Multi-Objective Algorithm* (SMS-EMOA). Our results indicate that our proposed approach can reach a speedup of up to 883x with respect to its sequential version, which makes possible the use of exact hypervolume contributions for problems having up to nine objectives.

Agradecimientos

La realización de esta tesis marca un objetivo más en mi lista de metas, la cual no habría sido posible sin la presencia de perseverancia, responsabilidad, constancia y todos esos valores que sirven como guía para lograr un objetivo.

Me gustaría agradecer a mis padres por el amor y apoyo incondicional que me han brindado durante toda mi vida. También a todos mis seres queridos, por todo lo que han compartido conmigo y por ser el motor que me impulsa cada día. De igual forma, quiero agradecer a todos los que se han cruzado en mi camino y me han brindado su cariño en algún momento de mi vida.

También quiero agradecer a todas aquellas personas que de algún modo han sido mis guías, pues sin sus enseñanzas y consejos ninguno de mis logros hubiese sido posible. A todos los Drs. que me impartieron clases durante esta etapa de mi vida por compartir sus conocimientos.

Quiero agradecer a mi asesor el Dr. Carlos A. Coello Coello por haberme dado el honor de trabajar bajo su guía y lidiar conmigo, por transmitirme su pasión por la investigación y con ello sembrar en mi un gran deseo de aprender, por todas sus enseñanzas, por su apoyo y por compartir sus experiencias de vida de forma chusca. Finalmente quiero agradecer al CINVESTAV por permitirme formar parte de esta gran institución y al CONACyT por brindarnos un apoyo económico, permitiendo que muchas personas como yo, podamos concluir los estudios de maestría.

Índice general

Índice de figuras

Lista de Algoritmos

Índice de tablas

Capítulo 1

Introducción

En el mundo real, a menudo se presentan problemas de optimización numérica y combinatoria. Ejemplos comunes pueden encontrarse en áreas tales como: finanzas, predicción, robótica y teoría de juegos, entre otras. Estos problemas de optimización del mundo real suelen requerir la optimización de más de una función objetivo al mismo tiempo, y dichas funciones suelen estar en conflicto entre sí. Dichos problemas se denominan Problemas de Optimización Multiobjetivo (POMs) y resolverlos consiste en encontrar un conjunto de soluciones que representen los mejores compromisos posibles entre las funciones objetivo que están siendo optimizadas. Estas soluciones se denominan *conjunto de óptimos de Pareto*, y su respectiva proyección en el espacio de los objetivos se denomina *frente de Pareto*.

Los POMs son comúnmente resueltos por técnicas clásicas de programación matemática como el método del criterio global, la programación por metas, el método de satisfacción de metas, la optimización min-max y combinaciones lineales de pesos, entre otros [1]. A pesar de la considerable cantidad de técnicas de programación matemática existentes para atacar este tipo de problemas, éstas requieren de alguna información adicional del problema (p.ej., que las funciones objetivo sean diferenciables, que el frente de Pareto sea continuo, entre otras). El hecho es que, los problemas del mundo real tienden a ser no lineales y con funciones objetivo las cuales resultan muy costosas para evaluar, lo que conlleva a buscar otras alternativas, tales como las metaheurísticas. Entre las metaheurísticas disponibles en la actualidad, los algoritmos evolutivos (AEs) son unos de los métodos más utilizados en la literatura especializada. Los algoritmos evolutivos multiobjetivo (AEMO) tienen la ventaja de generar varios elementos del conjunto de óptimos de Pareto en una sola ejecución, en contraste con las técnicas de programación matemática que usualmente sólo producen una solución por cada ejecución. Además, los AEMOs son menos sensibles ante aspectos como la forma y continuidad del frente de Pareto, mientras que dichas características usualmente causan serias dificultades a las técnicas de programación matemática a la hora de resolver estos problemas.

La principal motivación de este trabajo es que muchos problemas multiobjetivo del mundo real tienen un gran número de funciones objetivo que se encuentran en conflicto entre sí, por lo cual la escalabilidad es un tema de importancia en los algoritmos

de optimización. Esto, sin embargo, contrasta con los modelos evolutivos existentes para optimización multiobjetivo los cuales suelen validarse con problemas de no más de 5 funciones objetivo. El diseñar nuevos métodos que sean más eficientes a la hora de abordar problemas con un gran número de funciones objetivo es un tema de importancia, y el puede involucrar la realización de cálculos mucho más costosos que los requeridos en problemas de baja dimensionalidad. El uso de esquemas de selección de las soluciones basados en optimalidad de Pareto presenta diversas limitantes, ya que se necesita información extra del problema para poder resolverlo y en ocasiones no se logra discernir la diferencia entre las soluciones. En años recientes, se han planteado diferentes mecanismos de selección para algoritmos evolutivos multiobjetivo que no se basan en la optimalidad de Pareto. Una de las alternativas más prometedoras es el uso de indicadores. La métrica $\mathcal{S}$ conocida comúnmente como el indicador hipervolumen, es sin duda la más popular debido a sus propiedades matemáticas ya que, se ha podido demostrar que maximizar el hipervolumen conduce a la convergencia hacia el verdadero frente del Pareto de un problema [2]. Sin embargo, el uso de dicho indicador es sumamente costoso (computacionalmente hablando) conforme se incrementa el número de funciones objetivo, por lo que es deseable minimizar dicho tiempo de ejecución, sin sacrificar dichas propiedades del indicador. Esto nos lleva a proponer un algoritmo que pueda lidiar con el cálculo de la contribución al hipervolumen para usarse como criterio de selección de un algoritmo evolutivo multi-objetivo haciendo uso de la tecnología de las unidades de Procesamiento Gráfico (GPUs) para buscar disminuir los tiempos de ejecución del algoritmo evolutivo multi-objetivo.

Las GPUs tienen un alto desempeño ya que cuentan con un gran número de procesadores embebidos en una misma tarjeta. Las GPUs fueron diseñadas con el objetivo de realizar procesamiento gráfico y pueden ser programadas con ayuda de Interfaces de Programación Aplicada (APIs) tales como: DirectX y OpenGL. Hoy en día, las GPUs son procesadores de propósito general y diseñados para trabajar con: CUDA y OpenGL. Existen muchas aplicaciones en las cuales el uso de GPUs presenta un gran beneficio (disminución de tiempo de ejecución) con respecto a implementaciones en CPU. CUDA es acrónimo de Compute Unified Device Architecture y consiste en una plataforma de software y un modelo de programación creada por NVIDIA, que a su vez, son diseñadores de tarjetas gráficas de alto rendimiento y de propósito general. CUDA es una extensión de C, C++ y Fortran. Esto nos permite crear prototipos de funciones, las cuales son llamadas *kernels* y son ejecutadas en paralelo por hilos de CUDA.

1.1. Objetivos de la tesis

En esta sección se describen los objetivos de esta tesis, abarcando tanto el general como los particulares.

Objetivo General

Proponer un algoritmo que pueda lidiar con el cálculo de la contribución al hipervolumen exacto para usarse como criterio de selección de un algoritmo evolutivo multi-objetivo con ayuda de las Unidades de Procesamiento Gráfico.

Objetivos Particulares

- Diseñar un modelo algorítmico paralelo para GPUs que permita atacar de mejor manera el cálculo de la contribución al hipervolumen en alta dimensionalidad para usarse como criterio de selección en los algoritmos evolutivos multi-objetivo.
- Realizar una implementación del modelo propuesto.
- Realizar comparativas de desempeño con respecto al algoritmo representativo del estado del arte con el fin de observar las fortalezas y debilidades del algoritmo propuesto.

1.2. Organización de la tesis

Esta tesis está organizada de la siguiente manera. Los cuatro primeros capítulos describen los antecedentes y los conceptos básicos requeridos para comprender el trabajo y descripción de esta tesis. Los últimos tres capítulos están dedicados a describir la propuesta desarrollada, su implementación y los resultados obtenidos con ella. El contenido de cada capítulo se describe brevemente a continuación.

El capítulo 2 presenta una breve introducción a la optimización multiobjetivo y a la computación evolutiva. Al inicio del capítulo se proporciona una descripción del problema que se aborda en la tesis, proporcionándose además definiciones y conceptos básicos de optimización multiobjetivo. Posteriormente, se describe la computación evolutiva incluyendo los componentes básicos de un algoritmo evolutivo. También se describen los algoritmos evolutivos multiobjetivo, sus componentes básicos y sus criterios de selección, incluyendo el uso de indicadores de desempeño. En el capítulo 3 se habla acerca de la programación paralela describiendose los modelos de programación paralela, los tipos de paralelismo y el paralelismo de grano fino y grano grueso. También se presenta una breve introducción a las Unidades de Procesamiento Gráfico (GPUs) incluyendo sus conceptos básicos, arquitectura, interacción entre CPU y GPU, y ejecución concurrente, entre otros.

El capítulo 4 está orientado a describir el hipervolumen y la contribución al hipervolumen, proporcionándose su definición formal, los algoritmos más representativos de la literatura para el cálculo de dicha métrica tales como: Inclusión-Exclusión, lebMeasure y FPL. Así mismo, se proporciona la definición de un KMP (*Klee Measure Problem*), así como la relación que existe entre el hipervolumen y el KMP.

En el capítulo 5 se describe nuestra propuesta. También se mencionan cuáles fueron los criterios de diseño que se tomaron en cuenta y se describe la forma de

atacar el problema de nuestro interés en dos y más de dos dimensiones. Así mismo, se detalla el funcionamiento general del algoritmo propuesto y de las técnicas en él utilizadas. Finalmente, se describen los detalles de la implementación.

El capítulo 6 se dedica a presentar los resultados obtenidos por el algoritmo. Se comienza con una descripción de los problemas de prueba y de las métricas utilizadas. El capítulo finaliza con la evaluación de los resultados realizando un análisis del desempeño de los algoritmos comparados para cada problema de prueba.

En el capítulo 7 se presentan las conclusiones generales de este trabajo de tesis y una posible línea de trabajo futuro. También se presenta un breve análisis de las principales fortalezas y debilidades de la propuesta desarrollada en este trabajo de tesis.

Capítulo 2

Optimización Multi-objetivo

En este capítulo se da una introducción a la optimización multi-objetivo. Así mismo, se propone el planteamiento del problema general de optimización. También, se introducen conceptos básicos de Computación Evolutiva, incluyendo una descripción breve de sus paradigmas principales, enfatizando ventajas y desventajas del uso de un algoritmo evolutivo (AE).

2.1. Problemas de optimización multi-objetivo

La *optimización* es el proceso de seleccionar a la mejor solución candidata de un conjunto posible [3].

Algunos problemas del mundo real pueden ser tratados como problemas mono-objetivo, aunque, por lo general es complicado hacerlo, ya que requerimos definir todos los aspectos o características del problema en un solo objetivo. Definiendo múltiples objetivos, normalmente podemos darnos una mejor idea del problema. La optimización multi-objetivo también es conocida como optimización multicriterio.

En la optimización mono-objetivo, el espacio de búsqueda a menudo se encuentra bien definido y aspiramos a encontrar una solución única (la mejor posible). Sin embargo, cuando tenemos varios objetivos en conflicto entre sí, tendremos varias soluciones, que representan los mejores compromisos posibles entre los objetivos (es decir, no podemos mejorar un objetivo sin empeorar otro).

Un problema de optimización multi-objetivo (POM) consiste en encontrar un vector de variables de decisión que satisfagan un cierto conjunto de restricciones y optimice un conjunto de funciones objetivo [4]. Estas funciones forman una descripción matemática de los criterios de desempeño que suelen estar en conflicto unos con otros y que se suelen medir en unidades diferentes. El término *optimizar* en este caso significa encontrar soluciones que den valores a las funciones objetivo que sean aceptables para los tomadores de decisiones.

Un POM se puede escribir de la siguiente manera:

Encontrar un vector

$$\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T \tag{2.1}$$

Tal que se requiere minimizar la función vectorial

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{2.2}$$

Sujeto a m restricciones de desigualdad y p restricciones de igualdad:

$$g_i(\vec{x}) \leq 0 \qquad\qquad i = 1, 2, \ldots, m \tag{2.3}$$

$$h_i(\vec{x}) = 0 \qquad\qquad i = 1, 2, \ldots, p \tag{2.4}$$

donde

$$\vec{x} = [x_1, x_2, \ldots, x_n]^T \tag{2.5}$$

es el vector de variables de decisión que definen el problema.

En otras palabras, se pretende determinar un conjunto $[x_1, x_2, \ldots, x_n]^T$ tal que satisfaga las ecuaciones (2.3) y (2.4) y que produzcan valores óptimos para todas las funciones objetivo. Las restricciones descritas anteriormente representan limitantes respecto al tipo de soluciones que debemos generar (p.ej., restricciones de tiempo, limitaciones físicas o de estado, entre otras.).

La noción de *óptimo* que se suele adoptar en optimización multi-objetivo es la propuesta originalmente por Francis Ysidro Edgeworth y que más tarde fue generalizada por Vilfredo Pareto (en 1896), la cual se conoce como óptimo de Pareto [5]. Koopmans fue uno de los primeros en emplear el concepto de óptimo de Pareto en 1951. A continuación, se presentan de manera más formal algunas definiciones.

Definición **2.1** ***Optimalidad de Pareto:*** *Decimos que un vector de variables de decisión $\vec{x}^* \in \Omega$ (donde Ω es la zona factible) es un óptimo de Pareto si no existe otro $\vec{x} \in \Omega$ tal que $f_i(\vec{x}) \leq f_i(\vec{x}^*)$ para toda $i = 1, \ldots, k$ y $f_j(\vec{x}) < f_j(\vec{x}^*)$ para al menos una j.*

Definición **2.2** ***Dominancia de Pareto:*** *Se dice que un vector $\vec{u} = [u_1, \ldots, u_k]^T$ domina a otro vector $\vec{v} = [v_1, \ldots, v_k]^T$ (denotado por $\vec{u} \preceq \vec{v}$) si y sólo si $\vec{u}$ es parcialmente menor que $\vec{v}$, es decir:*

$$\forall i \in \{1, \ldots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i \tag{2.6}$$

Definición **2.3** ***Conjunto de óptimos de Pareto:*** *Para un problema multi-objetivo dado $\vec{f}(\vec{x})$, el conjunto de óptimo s de Pareto P^* se define como:*

$$P^* := \{\vec{x} \in \Omega | \nexists \vec{x'} \in \Omega, \vec{f}(\vec{x'}) \preceq \vec{f}(\vec{x})\} \tag{2.7}$$

Definición **2.4** ***Frente de Pareto:*** *Para un problema multi-objetivo dado $\vec{f}(\vec{x})$ y su conjunto de óptimos de Pareto P^*, el frente de Pareto se define como:*

$$PF^* := \{\vec{f}(\vec{x}), \vec{x} \in P^*\} \tag{2.8}$$

Algunos autores [6] consideran la dominancia débil y fuerte de Pareto como variantes de la optimalidad de Pareto. La relación entre estas definiciones es que el conjunto de óptimos de Pareto fuerte es un subconjunto del conjunto de Pareto, el cual es un subconjunto del conjunto de óptimos de Pareto débil.

La obtención del frente de Pareto de un problema de optimización multi-objetivo es la meta principal de la optimización multi-objetivo. Es deseable que la aproximación obtenida esté tan cerca como sea posible del verdadero frente de Pareto y que se distribuya a lo largo del frente de Pareto de la manera más uniforme posible [5].

2.2. Vectores de referencia

En esta sección, se definen algunos vectores especiales los cuales son usados como puntos de referencia en algoritmos de optimización multi-objetivo. El primero de ellos denota los límites inferiores del conjunto de óptimos de Pareto y se expresa de la siguiente forma:

Definición **2.5** ***Vector objetivo ideal:*** *Cada elemento del vector ideal $\vec{z}^* = (z_1^*, ..., z_m^*)$ minimiza cada función objetivo de manera independiente. El i-ésimo componente se define como $z_i^* = \text{mín}_{\vec{x} \in S} f_i(\vec{x})$.*

Es obvio que este vector corresponde a una solución inexistente porque existen conflictos entre los objetivos. En general, las soluciones más cercanas al vector ideal son mejores. Este punto es utilizado en algunos algoritmos como NBI (*Normal Boundary Intersection*) [7]. Por otra parte, algunos algoritmos pueden requerir una solución que tiene un valor objetivo estrictamente mejor que todas las soluciones en la región factible. Por esta razón, el siguiente punto se define como:

Definición **2.6** ***Vector objetivo utópico:*** *El vector objetivo utópico es un vector infactible, el cual está compuesto por $z^{**} = z^* - \epsilon_i$ para toda $i = \{1, ..., m\}$, donde ϵ_i es un escalar, con un valor pequeño.*

Cabe mencionar que este vector domina estrictamente cada solución óptima de Pareto. Por último el vector opuesto al vector ideal, el cual denota los límites superiores del conjunto de óptimos de Pareto, se define de la siguiente manera.

Definición **2.7** ***Vector objetivo de nadir:*** *El vector objetivo de nadir* $\vec{z}^{nad} = (z_1^{nad}, ..., z_m^{nad})$ *contiene en cada uno de sus componentes al máximo de cada función objetivo, obtenido de manera independiente. En este caso, el i-ésimo componente se define como* $z_i^{nad} = \max_{\vec{x} \in S} f_i(\vec{x})$.

El vector objetivo de nadir podría ser o no un punto factible. En la práctica, es difícil obtenerlo para problemas con más de dos objetivos. Estos vectores objetivo son útiles para la normalización de los valores objetivos en una escala común.

2.3. Métodos tradicionales para resolver problemas multi-objetivo

La programación matemática envuelve varios métodos de búsqueda. Estos métodos son agrupados respecto a la complejidad de los POMs. La programación lineal está diseñada para resolver problemas, en los cuales la función objetivo y todas la restricciones son lineales. En contraste, las técnicas de programación no lineal resuelven algunos POMs no cumpliendo con estas restricciones, pero requieren que las restricciones sean convexas. Finalmente, la programación estocástica es usada cuando los parámetros son de valor aleatorio y las funciones objetivo están sujetas a perturbaciones estadísticas. Dependiendo del tipo de características usadas en el problema, existen distintas variantes de estos métodos.

Una de las taxonomías más usadas para la programación no lineal es la propuesta por Hwang y Masud en 1979 [6], donde los métodos son catalogados dependiendo de la participación del tomador de decisiones en el proceso de la búsqueda. La taxonomía propuesta por Hwang y Masud es la siguiente :

- Métodos sin preferencias,
- Métodos a posteriori,
- Métodos a priori,
- Métodos interactivos.

En las siguientes subsecciones se presenta una breve descripción de cada una de estas categorías.

2.3.1. Métodos sin preferencias

En los métodos sin preferencias se incluyen todos aquellos métodos que no toman en consideración las opiniones de preferencias de quien toma las decisiones. El POM se resuelve utilizando algún método relativamente simple y la solución obtenida es presentada al encargado de tomar las decisiones, el cual puede aceptar o rechazar la solución. Estos métodos son generalmente utilizados cuando el tomador de decisiones no espera una solución especial y está satisfecho con cualquier solución óptima. Un ejemplo de esta categoría es el método de criterio global [8].

2.3.2. Métodos a posteriori

Estos métodos generan un conjunto de óptimos de Pareto, con el inconveniente de que el proceso de generación es usualmente muy costoso computacionalmente y en la mayoría de los problemas es sumamente difícil de lograr. Por otro lado, es difícil para el tomador de decisiones seleccionar de un conjunto grande de soluciones. Una cuestión importante es cómo mostrar las alternativas para la toma de decisiones de una manera efectiva. Ejemplos de métodos a posteriori:

- Método de sumas ponderadas [9],
- Método de restricciones ϵ [10].

2.3.3. Métodos a priori

La especificación de estos métodos se realiza antes del proceso de búsqueda y estas especificaciones pueden ser preferencias u opiniones. La principal dificultad de estos métodos radica en que no siempre se saben de antemano las características deseables que deben tener las soluciones del POM. Algunos métodos de esta categoría son:

- Programación por metas [11],
- Método lexicográfico [12],
- Método de la función de valor [13].

2.3.4. Métodos Interactivos

Suponiendo que el tomador de decisiones tiene el tiempo y las habilidades necesarias, estos métodos pueden producir mejores resultados. Es decir, sólo una parte del frente de Pareto tiene que ser generado y evaluado, y el tomador de decisiones puede especificar y corregir sus selecciones, y continuar con el proceso. Esto significa que no es necesario tener conocimiento previo acerca de la estructura del problema. Sin embargo, la información debe ser significativa y fácil de entender. A continuación se nombran algunos de los métodos pertenecientes a esta clase:

- Método de satisfacción de compromisos,
- Método de Geoffrion-Dyer-Feinberg [14],
- Método de Tchebycheff [15],
- Método del punto de referencia [16],
- Método de NIMBUS (*Nondifferentiable Interactive Multi-objective Bundle-based optimization System*)[17].

Estos métodos clásicos resuelven los problemas multi-objetivo usando técnicas de programación matemática. Sin embargo, algunos de ellos requieren información sobre el gradiente de las funciones objetivo y, por lo general generan una sola solución. De hecho, muchos de estos métodos transforman el problema multi-objetivo en un problema mono-objetivo usando restricciones, jerarquizando las funciones objetivo o usando funciones agregativas.

2.4. Algoritmos evolutivos

A diferencia de los métodos tradicionales, existen otras alternativas de resolver los POMs, las cuales no se basan en las propiedades matemáticas del problema; sino en simular el proceso evolutivo en una computadora; estos métodos son conocidos como algoritmos evolutivos multi-objetivo (AEMO). Los algoritmos evolutivos tienen como objetivo principal evolucionar individuos, los cuales, en este caso, representan soluciones a un cierto problema de optimización multi-objetivo. De esta manera, los individuos se irán actualizando al paso de las generaciones bajo el principio de supervivencia del más apto, por lo que cada nueva generación debe poseer características mejores que las anteriores, acercándonos con cada iteración a las mejores soluciones posibles al problema. En otras palabras, al paso de cada generación estamos mejorando las soluciones que tenemos, aproximándonos así, a las soluciones óptimas.

Antes de hablar de los algoritmos evolutivos multi-objetivo (AEMO), repasaremos algunos conceptos básicos de computación evolutiva. Es necesario tener en cuenta de algunos conceptos básicos, los cuales son útiles para lograr a entender los algoritmos evolutivos multi-objetivo.

2.4.1. Computación evolutiva

La Computación Evolutiva es una rama de la inteligencia artificial inspirada en los procesos de la evolución natural. Ésta concibe a la evolución como un proceso de optimización el cual es simulado para resolver problemas. La idea principal de la computación evolutiva es evolucionar una población de posibles soluciones a un problema, utilizando operadores inspirados en la variación genética y en la selección natural (reproducción, mutación, competencia y selección). La evolución opera como un método de búsqueda en un espacio de soluciones posibles. El conjunto de opciones de búsqueda consiste en las posibles secuencias genéticas, y las soluciones deseables son los organismos más aptos para sobrevivir y reproducirse en su ambiente. Las técnicas que conforman la computación evolutiva son procedimientos inspirados en el principio natural de la evolución de las especies, las cuales modelan la evolución como un proceso de optimización.

Conceptos básicos de computación evolutiva

En las técnicas de la computación evolutiva existen individuos (entidades) los cuales tienen la capacidad de reproducirse con una determinada probabilidad y que representan posibles soluciones al problema que se desea resolver. Al reproducirse los individuos, se realiza un proceso de exploración en el espacio de búsqueda para mejorar cada vez más las posibles soluciones y con esto acercarnos a la solución óptima del problema. A cada individuo se le asocia una aptitud, la cual describe la habilidad o capacidad del individuo; con dicha aptitud se evalúa la supervivencia de un individuo en cada generación. Esto se relaciona con el valor de la función objetivo para la correspondiente interpretación de éste en términos de soluciones del problema. Al conjunto de individuos se le denomina población, los cuales se renuevan generación tras generación bajo el principio de la supervivencia del más apto. La población cuenta con diversidad entre sus individuos, con lo cual se busca obtener un muestreo representativo del espacio de búsqueda del problema, es decir, contar con un mecanismo de exploración. Por esta razón, todos los algoritmos evolutivos tienen tres características importantes que los distinguen de otros algoritmos de búsqueda. En primer lugar, son algoritmos poblacionales. En segundo lugar, hay comunicación e intercambio de información entre todos los individuos de la población, mediante la selección o la recombinación de los individuos. Por último, cuentan con un factor muy importante: son estocásticos.

Entre los problemas apropiados para resolver con estas técnicas se encuentran aquellos en los que la región factible no se puede modelar de manera sencilla, así como aquellos en que no se cumplen condiciones de convexidad o conectividad, o cuando no se cuenta con conocimiento previo de las características del espacio de búsqueda. A continuación se definen algunos conceptos básicos de la computación evolutiva.

Población: es el conjunto de individuos que representan las posibles soluciones a un problema. Estos individuos están compuestos de un cromosoma o genoma cuya codificación es llamada genotipo. De manera opuesta se encuentra la decodificación de dicho cromosoma a la cual se denomina fenotipo.

Operadores: son los procedimientos que manipulan la información genética de la población para crear nuevos individuos a partir de la población original. Los operadores genéticos más comunes en los algoritmos evolutivos (AEs) son la cruza y la mutación.

Mutación: es la perturbación sobre uno o más elementos (también denominados como alelos) del cromosoma de un individuo. Nos permite alcanzar una región del espacio de búsqueda diferente a la del individuo original, lo que ayuda a que la búsqueda no se estanque en un óptimo local.

Cruza: tiene la finalidad de heredar la información genética de dos o más padres a la siguiente generación. En la computación evolutiva, la cruza entre cromosomas se simula intercambiando segmentos de cadenas lineales de longitud fija. Los

genes de los padres se combinan para formar las cadenas cromosómicas de sus descendientes, buscando preservar los buenos genes y obtener mejores soluciones al problema. El principio detrás de la cruza es combinar dos o más individuos con características deseadas para producir uno o más individuos que combinen tales cualidades.

Selección: es el proceso mediante el cual se determina qué individuos pasarán a formar parte de la siguiente generación. Cada uno de los individuos cuenta con un valor de aptitud determinado, el cual indica la diferencia entre los individuos de la población. A la determinación de dichos valores se le conoce como evaluación de la función de aptitud y dado que el proceso de selección toma en cuenta esta función, este procedimiento está basado en el principio de supervivencia del más apto.

2.4.2. Principales paradigmas de la computación evolutiva

La computación evolutiva se agrupa en tres principales paradigmas: los algoritmos genéticos (AGs), la programación evolutiva (PE) y las estrategias evolutivas (EE). Dichos algoritmos tienen en común el uso de operadores que simulan la reproducción, la mutación, la competencia, y la selección de individuos dentro de una población. A continuación se da una explicación breve acerca del funcionamiento de cada uno de estos paradigmas [18].

Algoritmos genéticos

Los algoritmos genéticos (AG) fueron propuestos por John H. Holland [19] y originalmente fueron denominados *planes reproductivos*, John H. Holland fue motivado por su interés en resolver problemas de aprendizaje de máquina. Con los AGs se introducen conceptos existentes en la naturaleza, como el genotipo y fenotipo. El genotipo representa la información genética de un individuo, que ha heredado de sus antecesores y que a su vez transmite a sus descendientes. El fenotipo son las características visibles de cada individuo. También dentro de la computación evolutiva, el fenotipo es la representación decodificada de una posible solución al problema y el genotipo es la cadena cromosómica que codifica dicha solución. El algoritmo genético opera a nivel de genotipo de las soluciones y enfatiza la importancia de la cruza sexual sobre la mutación, por lo que la cruza es el operador principal de este paradigma el cual utiliza una selección probabilística. La cadena binaria es la representación tradicional de los algoritmos genéticos, y es llamada *cromosoma*. A cada posición dentro del cromosoma correspondiente a una variable de decisión se le denomina *gene* y cada uno de sus elementos específicos es llamado *alelo*. De tal forma, para poder aplicar el algoritmo genético se requiere de los siguientes componentes básicos:

- Una representación de las soluciones al problema.

- Un procedimiento para crear una población de posibles soluciones de manera aleatoria.
- Una función de evaluación que simule el ambiente, clasificando los individuos en términos de su *aptitud*.
- Operadores genéticos que alteren la composición de los descendientes que se producirán para las siguientes generaciones.
- Los parámetros utilizados por el algoritmo genético tales como: el tamaño de población y la probabilidad de cruza, probabilidad de mutación y número máximo de generaciones, entre otros.

Este es el algoritmo general de este paradigma.

Algoritmo 1: Algoritmo genético simple.

Entrada: Tamaño de población (*tampob*), número máximo de generaciones (*Gmax*)
Salida: *ConjuntoDeSoluciones*

```
1 Generar aleatoriamente la población inicial, G(0);
2 t := 0;
3 while Cierta condición de paro no se satisfaga o t < Gmax do
4 |   sea t := t + 1;
5 |   calcular la aptitud de cada individuo de G(t);
6 |   seleccionar G_1(t) con base en la aptitud de G(t);
7 |   aplicar los operadores genéticos a G_1(t) para generar G(t + 1);
8 return ConjuntoDeSoluciones;
```

Estrategias evolutivas

Las estrategias evolutivas (EE) fueron propuestas por varios investigadores alemanes encabezados por Ingo Rechenberg en 1964 [20]. Son un método de ajustes discretos aleatorios inspirado en el mecanismo de mutación que ocurre en la naturaleza. Por esta razón el operador principal es la mutación y la cruza es un operador secundario en las EEs. La versión original de la EE, denotada por (1 + 1)-EE, usa un solo padre y produce en cada iteración un solo hijo usando mutación. El mejor de entre los dos (el padre y el hijo) pasará a formar parte de la siguiente generación. En la (1 + 1)- EE, a partir de un padre $x^t = (x_1^*, x_2^*, \ldots, x_n^*)^T$ se genera un hijo mediante:

$$x_i^{t+1} = x_i^t + N_i(0, \sigma_i^t) \tag{2.9}$$

donde t es la generación en la que se encuentra el algoritmo y $N_i(0, \sigma_i^t)$ es un vector de números Gaussianos independientes con media cero y desviación estándar σ_i. De tal forma, la mutación opera como una perturbación entre los individuos.

Más tarde, el concepto de población fue propuesto por Rechenberg [21] en las EEs, al crear una estrategia evolutiva llamada $(\mu + 1)$-EE, la cual está compuesta de μ padres, creándose un solo hijo a partir de éstos, el cual puede reemplazar al peor padre de la población en una selección extintiva.

Finalmente, Hans-Paul Schwefel introdujo a las estrategias el uso de múltiples hijos en las denominadas $(\mu+\lambda)$-EE y (μ, λ)-EE, donde μ son los padres y λ los hijos. La primera estrategia selecciona μ individuos de la unión de padres e hijos, mientras que la segunda selecciona μ individuos de la población de hijos.

Las EEs han sido empleadas para resolver problemas en áreas tales como: ingeniería, bioquímica, óptica y redes, entre muchas otras [22].

Programación evolutiva

La programación evolutiva (PE) fue propuesta por Lawrence J. Fogel [23]. En este paradigma la inteligencia se ve como un comportamiento adaptativo. Está inspirada en el principio de la evolución al nivel de las especies. Utiliza una selección probabilística y no requiere de la cruza u otro tipo de recombinación, puesto que, en la naturaleza, una especie no puede mezclarse con otra y este algoritmo evolutivo simula este principio. Enfatiza los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos, como lo hacen los algoritmos genéticos.

Algoritmo 2: Algoritmo básico de la programación evolutiva

Entrada: Tamaño de población (*tampob*)
Salida: *ConjuntoDeSoluciones*

```
1 Generar aleatoriamente la población inicial;
2 while Cierta condición se satisfaga do
3 |   aplicar Mutación;
4 |   calcular la aptitud de cada hijo;
5 |   seleccionar las soluciones que se retendrán;
6 |   remplazar la población actual con la seleccionada;
7 return ConjuntoDeSoluciones ;
```

La PE ha sido utilizada en diferentes áreas tales como: reconocimiento de patrones, predicción, identificación, control automático y juegos, entre otras [23, 24]. En la tabla 2.1 se pueden observar las principales características, similitudes y diferencias de cada uno de los paradigmas, que conforman la computación evolutiva.

2.5. Algoritmos evolutivos multi-objetivo

Desde que *Rosenberg* [25] hiciese alusión al potencial de los algoritmos evolutivos para resolver problemas multi-objetivo, el interés de esta área por parte de la comunidad de computación evolutiva ha crecido enormemente, hasta llegar a afianzarse

Tabla 2.1: Tabla comparativa entre cada uno de los paradigmas principales que conforman la computación evolutiva.

Características	**Estrategias Evolutivas**	**Programación Evolutiva**	**Algoritmo Genético**
Representación	Real	Real	Binaria
Función de Aptitud	Valor de la función objetivo	Valor de la función objetivo ajustada	Valor de la función objetivo ajustada
Auto-Adaptación	Desviaciones Estándar y ángulos de rotación	Ninguna, Varianzas (PE- Estándar), Coeficientes de correlación (meta-PE)	Ninguna
Mutación	Gaussiana, operador principal	Gaussiana, operador único	Inversión de bits, operador secundario
Recombinación	Discreta e intermedia, sexual y panmítica, importante para la auto-adaptación	Ninguna	Cruza de 2- puntos, cruza uniforme, únicamente sexual, operador principal
Selección	Determinística, extintiva o basada en la preservación	Probabilística, extintiva	Probabilística, basada en la preservación
Restricciones	Retricciones arbitrarias de desigualdad	Ninguna	Límites simples mediante el mecanismo de codificación
Teoría	Velocidad de Convergencia para casos especiales, (1+1)-EE, (1+λ)-EE, Convergencia global para $(\mu + \lambda)$-EE	Velocidad de Convergencia para casos especiales, (1+1)-PE, Convergencia global para meta-PE	Teoría de los Esquemas, Convergencia global para la versión elitista

como una línea de investigación propia, llamada *optimización evolutiva multi-objetivo* [26]. Existen varias ventajas al utilizar las técnicas evolutivas. Una de ellas es que la computación evolutiva involucra métodos que son poblacionales, lo que significa que trabajan con varias soluciones a la vez, y no con una, como lo hacen la mayoría de las técnicas de programación matemática. Esto permite que los algoritmos evolutivos multi-objetivo (AEMOs) generen varios elementos del conjunto de óptimos de Pareto en una sola ejecución. Ademas, los AEMOs no requieren que el problema a resolverse cumpla condiciones de continuidad y/o diferenciabilidad en sus funciones objetivo, ni que éstas estén dadas en forma algebraica.

En general, un AEMO mantiene un conjunto de soluciones potenciales, el cual es sometido a un proceso de selección y es manipulado por operadores genéticos (generalmente la recombinación y la mutación [27]). La selección, sin embargo, suele hacerse con base en la optimalidad de Pareto y no con base en aptitud. Así mismo, los AEMOs requieren un estimador de densidad para mantener varias soluciones diferentes en la misma población.

En una reseña de AEMOs Fonseca y Fleming [28] presentan las principales estrategias para llevar a cabo la asignación de aptitud de los individuos. Estas estrategias se muestran a continuación:

- **Métodos de agregación:** mapean los objetivos en una función escalar que es utilizada para calcular la aptitud un individuo. Producen una sola solución por ejecución. Por otro lado, definir una función objetivo con estas características requiere de un gran conocimiento del problema.

- **Métodos basados en población:** son capaces de evolucionar múltiples soluciones no dominadas en una sola ejecución. Cambian el criterio de selección durante la fase de reproducción y la búsqueda es guiada en varias direcciones al mismo tiempo. A menudo, se seleccionan fracciones del conjunto de individuos para ser recombinados (*mating pool*) de acuerdo a uno de los n objetivos del problema.

- **Métodos basados en jerarquización de Pareto:** estos métodos asignan la aptitud con base en la dominancia de Pareto. Utilizan la dominancia de Pareto para determinar la probabilidad de reproducción de cada individuo. Hoy en día, los métodos basados en la jerarquización de Pareto son los más populares en el área de optimización multi-objetivo.

Existen además otras clasificaciones de los AEMOs, como la propuesta por Coello [5]:

- Enfoques simples:
  - Funciones agregativas.
  - Programación por metas.
  - Método de restricciones ϵ.
- Enfoques que no incorporan la optimalidad de Pareto en su mecanismo de selección.
  - Ordenamiento lexicográfico.
  - Uso de pesos y elitismo generados aleatoriamente.
  - Métodos basados en población.
- Enfoques que incorporan la optimalidad de Pareto en su mecanismo de selección.

- MOGA.
- NSGA y NSGA-II.
- SPEA y SPEA2.

Elementos clave de los AEMOs

De las definiciones previamente proporcionadas, podemos distinguir dos características principales que debemos alcanzar en la aproximación del conjunto de óptimos de Pareto: una es minimizar la distancia al verdadero frente de Pareto y la segunda es maximizar la diversidad dentro del conjunto de óptimos de Pareto. La mayoría de los AEMOs más populares utilizan la jerarquización de individuos con respecto a la optimalidad de Pareto, para la selección y adoptan algún estimador de densidad como nichos [29] y *crowding* [30] para producir varias soluciones distintas en una sola ejecución. En esta sección se describen algunos conceptos importantes de los AEMOs, que se toman en consideración para su diseño, a fin de lograr cubrir estas características.

Elitismo A diferencia de la optimización global, en la cual se retiene a la mejore solución de la población de una generación a otra sin alteración, la implementación del elitismo en los AEMOs es más compleja. En lugar de un mejor individuo, hay un conjunto de élite de un tamaño que puede ser mayor que el de la población. En la actualidad, hay dos enfoques principales para llevar a cabo el mecanismo de elitismo. Uno de ellos es combinar la población de padres y la de hijos, para posteriormente retener la mejor mitad. Otra alternativa es mantener una población secundaria llamada archivo histórico, en el cual se retienen las soluciones no dominadas encontradas en el transcurso del proceso de búsqueda [27].

Diversidad poblacional En un AE, el mecanismo de selección toma las soluciones con aptitud alta y descarta aquellas con aptitud baja, lo cual permite que la población converja a una solución única. Sin embargo, en los AEMOs la meta es encontrar un conjunto de soluciones bien distribuidas a lo largo del frente de Pareto en una sola ejecución. Esto requiere de estimadores de densidad que eviten convergencia a una solución única (*fitness sharing* [31], *clustering* [32], nichos [29], *crowding* [30], entropía [33], etc.).

2.5.1. Indicadores multi-objetivo

Los AEMOs producen un conjunto de soluciones que representan compromisos en los objetivos. Las soluciones producidas por AEMOs son difíciles de comparar entre sí. Esto ha motivado el desarrollo de indicadores de desempeño que permitan comparar los resultados obtenidos por dos o más AEMOs diferentes.

Indicadores como medida de desempeño

Los indicadores se utilizan comúnmente para comparar la calidad de las soluciones producidas en un AEMO. A lo largo del uso de estas medidas, se puede obtener más información acerca del funcionamiento de los AEMOs, incluyendo el efecto de modificar sus parámetros. Los indicadores también se utilizan para comparar el desempeño entre diferentes AEMOs.

Indicadores como criterio de selección

Recordemos que un AE, sigue un proceso iterativo que emula las selección natural, la adaptación y la herencia genética. El factor clave para los AEMOs, es la selección para realizar el apareamiento. Los AEMOs intentan encontrar el mejor conjunto de posibles soluciones al frente de Pareto. En los AEMOs, como se mencionó anteriormente, esto ocurre a través de un proceso de selección y reproducción. En optimización mono-objetivo, la selección es relativamente simple, puesto que clasificar las soluciones involucra un solo objetivo. En optimización multi-objetivo es más complicado, ya que entre mayor sea el número de objetivos es más difícil discernir la deferencia de cada una de las soluciones, si usamos el concepto de optimalidad de Pareto. Por desgracia, las poblaciones de los AEMOs normalmente están limitadas en tamaño y, como tal, no siempre es posible mantener todo un rango de soluciones entre las generaciones. Por lo tanto, el optimizador debe elegir en subconjunto de estas soluciones, las cuales son incomparables, y usarlas para la próxima generación. Para garantizar la diversidad en la población, es importante que las soluciones elegidas se encuentren bien distribuidas en todo el espacio de las funciones objetivo. Cabe mencionar que conforme se incrementa el número de objetivos del problema, el número de posibles soluciones contenidas en cada rango puede aumentar exponencialmente [34, 35]. Por esta razón, se requieren mejores métodos de selección para un mejor desempeño de los AEMOs en la presencia de muchas funciones objetivo. Uno de los indicadores más populares para los AEMOs es el hipervolumen, también es conocido como métrica $\mathcal{S}$ o la medida de Lebesgue [36]. El uso del hipervolumen se ha limitado en gran medida por el costo computacional que existen al calcular dicha métrica. Sin embargo, una de las características importantes del hipervolumen es que está comprobado que el maximizar el hipervolumen es equivalente a lograr convergencia al verdadero frente de Pareto [37] (en el capítulo 4, en la página 41 se describe a mayor detalle).

2.5.2. Algoritmos evolutivos multi-objetivo basados en indicadores

En esta sección se describen brevemente algunos de los AEMOs basados en indicadores que se encuentran en los más representativos del área:

Indicator Based Evolutionay Algorithm (IBEA)

El algoritmo IBEA fue propuesto por Ziztler y Künzli [38], el cual permite el uso de distintos indicadores de desempeño como criterio de selección del AEMO. Los autores lo probaron con el hipervolumen y el indicador ϵ. La aptitud de un solución esta basada en la suma de los valores de los indicadores resultantes de las comparaciones por pares a todas las otras soluciones. Existen distintas variantes del IBEA [39, 40], las cuales, el uso del hipervolumen como indicador ha sido utilizado casi siempre. Wagner presento muy buenos resultados por los IBEAs para problemas de muchos objetivos, ya que los IBEAs no usan la dominancia de Pareto. El problema principal de los IBEAs para problemas con muchos objetivos es el costo computacional que se tiene al calcular el valor del hipervolumen. Por esta razón, Ishibuchi propuso una versión iterativa de los IBEAs para reducir el costo computacional [41].

S-Metric Selection Evolutionary Multi-Objetive Algorithm (SMS-EMOA)

Originalmente el algorítmico SMS-EMOA fue propuesto por Emmerich [42], el cual está compuesto por dos partes fundamentales: (1) utiliza un ordenamiento no-dominado para jerarquizar cada uno de los frentes del conjunto de puntos y (2) el indicador de hipervolumen es utilizado en su proceso de selección del AEMO. En el SMS-EMOA sólo una solución es creada cada vez y es agregada a la población actual, para entonces ejecutar la selección de individuos que pasarán a la siguiente generación. Posteriormente, para cada solución en la población extendida, se calcula su contribución a la métrica de hipervolumen (se da una descripción más detallada en el capítulo 4). La contribución al hipervolumen es asignada como aptitud a cada individuo (solución) en la población. Al inicio del proceso evolutivo, muchas soluciones dentro de la población actual pueden ser dominadas y, por lo tanto, no contribuyen a la métrica del hipervolumen de la aproximación del frente de Pareto. SMS-EMOA requiere jerarquizar cada uno de los frentes como la que utiliza NSGA-II y la contribución al hipervolumen es calculada en el ultimo frente de soluciones no dominadas. En consecuencia, se descarta la solución con menor contribución al hipervolumen. Debido a que el costo computacional que se requiere para calcular la contribución al hipervolumen es elevado para más de tres objetivos, difícilmente el SMS-EMOA es aplicable a problemas de dimensiones superiores. En 2007, Beume propuso una nueva versión del SMS-EMOA en la cual no se usa la contribución al hipervolumen cuando, en la jerarquización de Pareto obtenemos más de un frente. En este caso, proponen usar el número de soluciones que dominan a una solución. Los autores indican que la motivación para usar el hipervolumen es mejorar la distribución de las soluciones en el frente de Pareto, por lo tanto no es necesario calcular la contribución al hipervolumen en frentes distintos del verdadero frente de Pareto.

Algoritmo 3: Algoritmo del SMS-EMOA

Entrada: Un problema multi-objetivo, criterio de paro y un punto de referencia $\mathcal{R}$

Salida: Aproximación del conjunto de óptimos de Pareto

```
1  Generar aleatoriamente la población inicial P;
2  Evaluar la población P;
3  while Cierta condición de paro no se satisfaga do
4      Generar un individuo hijo p del conjunto P usando operadores genéticos;
5      Evaluar el individuo p ;
6      P ← P ⋃ {p};
7      {R_1, ..., R_h} ← Ordenamiento no_dominado(P);
8      ∀x ∈ R_h : r(x) ← C_HV(R_h, R);
9      x* ← arg mín r(x);
10     P ← P \ {x*};
11 return P ;
```

Delta p-EMOA

Recientemente, un nuevo indicador de desempeño llamado Δ_p fue propuesto en [43]. Este indicador puede ser visto como *la distancia media de Hausdorff* entre el conjunto de soluciones y el conjunto de óptimos de Pareto. Este indicador está compuesto de ligeras modificaciones de dos indicadores de desempeño: distancia generacional (GD, ver [44]) y la distancia generacional invertida (IGD, ver [45]). Los autores linearizaron el frente no dominado e incluyeron este mecanismo en el delta p-EMOA, el cual es usado para resolver problemas de optimización de dos objetivos. Este algoritmo está inspirado en el SMS-EMOA, el cual agrega una población secundaria. Por lo que, Delta p-EMOA realiza una mejora al NSGA-II, consumiendo un menor número de de evaluaciones de función, respecto al NSGA-II, pero simplemente resuelve POMs con dos funciones objetivo. Existen una extensión de este modelo para problemas con tres funciones objetivo [46]. En este caso, el algoritmo requiere de algunos pasos previos. Esta nueva versión alcanza una mejor distribución de las soluciones que el MOEA/D, SMS-EMOA y NSGA-II. Sin embargo, este AEMO requiere de algunos parámetros adicionales y tiene un alto costo computacional para un mayor número de objetivos.

Delta p-DDE

El Delta p-DDE es un AEMO basado en el indicador Δ_p [45], el cual utiliza una forma escalonada de los individuos no dominados para construir una aproximación al conjunto de óptimos de Pareto. El indicador Δ_p es usado como mecanismo de selección. Delta p-DDE provee resultados muy competitivos con respecto al SMS-EMOA. A diferencia del SMS-EMOA, el costo computacional de esta propuesta es menor con-

forme se incrementa el número de objetivos de los problemas. La principal limitación de este enfoque es que produce una mala distribución de las soluciones en espacios de altas dimensiones. Ademas, tiene dificultades para aproximar soluciones al conjunto de óptimos de Pareto que sean discontinuos.

Existen otras propuestas de algoritmos evolutivos multi-objetivo tal como: el HyPE (*Hypervolume estimation algorithm for multi-objetive optimization*) [47]. En este caso, se propone un algoritmo rápido de búsqueda que usa muestreos basados en Monte Carlo para aproximar los valores del hipervolumen. La idea del método es jerarquizar las soluciones en base a la aproximación al hipervolumen. Aunque la idea detrás de este método es fascinante, en la práctica, su desempeño resulta muy pobre con respecto al de algoritmos que usan el valor exacto de la contribución al hipervolumen. En la optimización multi-objetivo, los indicadores de desempeño pueden adaptarse a la selección de los AEMOs durante el proceso evolutivo, tal como el hipervolumen. Después de todo, este indicador gracias a sus propiedades matemáticas garantiza distribución y convergencia al frente de Pareto. Sin embargo, su problema principal es que su elevado costo computacional hace prohibitivo el uso del hipervolumen exacto en el mecanismo de selección de un AEMO. Es por eso que en este trabajo de tesis, abordamos la implementación de este cálculo del hipervolumen en GPUs, a fin de disminuir su costo computacional de manera significativa.

Capítulo 3

Computación Paralela

Para algunos problemas de cómputo, el uso de la CPU (del inglés: *Central Processing Unit*) no es suficiente, ya que la respuesta del método para resolver el problema no se da en un tiempo razonable. Además, estos problemas pueden llegar a ser muy costosos y ni siquiera el uso de la CPU-multicore es suficiente para disminuir el tiempo de ejecución a un valor razonable. Estos problemas se pueden encontrar en distintas áreas de la ciencia y tecnología, tales como: Ciencias Naturales (Física, Química, Biología, etc.), tecnologías de la información, sistemas de información goespacial y en las ciencias de la computación. Hoy en día, muchos de estos problemas pueden ser resueltos más rápida y eficazmente, usando procesadores masivamente paralelos. La historia de cómo surgieron los procesadores masivamente paralelos es única en su clase. Ya que combinan dos áreas que no se encontraban relacionadas en lo absoluto: las ciencias de la computación y la industria de los videojuegos. En las ciencias de la computación, existe una necesidad para resolver problemas con un costo computacional elevado con la La industria de los videojuegos está interesada en lograr en tiempo real gráficos foto-realistas, con algoritmos embebidos en hardware. La necesidad de videojuegos más realistas llevó a la creación de un acelerador de gráficos, un procesador paralelo que maneja muchos cálculos gráficos usando funciones implementadas en hardware. Las dos necesidades, han dado origen a una de las herramientas más importante para la computación paralela: la *Graphical Processing Unit* (GPU) o Unidad de Procesamiento Gráfico.

En este capítulo se describen algunos conceptos básicos acerca de la computación paralela. Así mismo, se describen los modelos existentes para cómputo paralelo. Además, se introducen los tipos de paralelismo y los modelos de programación disponibles actualmente. Para finalizar, se da una breve introducción a las GPUs, describiendo su arquitectura y su modelo de programación, así como algunos ejemplos de la interacción entre la CPU y la GPU.

3.1. Conceptos básicos

Los términos *concurrencia* y *paralelismo* han sido debatidos en la comunidad de la computación paralela y en ocasiones no es muy clara la diferencia entre ellos. Ambos términos son usados frecuentemente en la computación de alto rendimiento. En [48] se proporcionan definiciones para concurrencia y paralelismo, las cuales son consideradas consistentes y correctas, y, por lo tanto, se incluyen a continuación:

Definición **3.1** ***Concurrencia*** *es una propiedad de un programa (a nivel de diseño) donde dos o más tareas pueden estar en progreso simultáneamente.*

Definición **3.2** ***Paralelismo*** *es una propiedad del tiempo de ejecución donde dos o más tareas se están ejecutando simultáneamente.*

Existe una diferencia entre *estar en progreso* y *estar ejecutando* ya que el primero no implica estar en ejecución. Sean C y P concurrencia y paralelismo, respectivamente, entonces $P \subset C$. En otras palabras, el paralelismo requiere de concurrencia, pero la concurrencia no requiere del paralelismo. Un buen ejemplo de ambos conceptos es un Sistema Operativo (SO), que es concurrente por diseño, ya que un SO realiza multiples tareas o están en progreso en un momento dado y, dependiendo al número de procesadores físicos, estas tareas pueden ejecutarse en paralelo o no.

Definición **3.3** ***Computación Paralela*** *es el acto de resolver un problema de tamaño n dividiendo su dominio en $l \geq 2$ (con $l \in \mathbb{N}$) partes y resolviéndolo con p procesadores físicos simultáneamente.*

Un algoritmo paralelo se define de la siguiente forma. Sea P_D un problema con dominio D. Si P_D es paralelizable, entonces D puede descomponerse en l sub-problemas:

$$D = d_1 + d_2 + ... + d_l = \sum_{i=1}^{l} d_i \tag{3.1}$$

Si P_D es un problema de paralelismo a nivel de datos si D se compone de datos y resolver el problema requiere la aplicación de una sola función $f(...)$ para todo el dominio D. En este caso:

$$f(D) = f(d_1) + f(d_2) + ... + f(d_l) = \sum_{i=1}^{l} f(d_i) \tag{3.2}$$

En otras palabras, el paralelismo a nivel de datos trabaja con una misma instrucción o función la cual opera con distintos datos. Por esta razón, este tipo de paralelismo es un buen candidato para implementarse en los GPUs. Por otra parte, existe otro tipo de problema el cual se define como:

P_D es un problema de paralelismo a nivel de tarea si D se compone de funciones y resolver el problema requiere la aplicación de cada función a un flujo común de datos S.

$$D(S) = d_1(S) + d_2(S) + ... + d_l(S) = \sum_{i=1}^{l} d_i(S) \tag{3.3}$$

Este tipo de paralelismo se puede implementar en CPU-multicore ya que las tareas se pueden ejecutar en cada unidad de procesamiento físico. Es de gran importancia identificar el tipo del problema, con la finalidad de alcanzar un alto desempeño y reducir el tiempo de ejecución de un algoritmo.

3.2. Modelos de cómputo paralelo

Los modelos de cómputo son máquinas computacionales abstractas que permiten el análisis teórico de los algoritmos. Estos modelos reducen todo el campo computacional a un pequeño conjunto de parámetros que definen el tiempo de acceso a memoria o el costo que se tendrá con una operación matemática cuando sea ejecutada en el procesador. Este análisis teórico es fundamental para el proceso de la investigación de nuevos algoritmos, ya que permite determinar qué algoritmo es mejor. En el caso de la computación paralela, hay varios modelos disponibles, los cuales se centran en distintos aspectos. Los modelos más representativosde cómputo paralelo son:

Máquina de acceso aleatorio paralelo: La máquina de acceso aleatorio paralelo o MAAP fue propuesta por Fortune y Wyllie en 1978 [49], y se inspiró en la máquina de acceso aleatorio. Éste ha sido uno de los modelos más usados para diseño o análisis de algoritmos paralelos. El modelo MAAP consta de p procesadores los cuales operan de manera sincronizada a través de una memoria limitada completamente visible para cada procesador. El parámetro p no necesita ser un valor constante. También se puede definir como una función de un problema de tamaño n, donde cada operación lectura/escritura tiene un costo de $\mathcal{O}(1)$. Existen diferentes variantes de este modelo para hacer más realista el modelado de nuevos algoritmos. Estas variantes especifican la forma de acceso a la memoria, ya sea al mismo tiempo o de forma exclusiva (ver la figura 3.1).

Jerarquía de memoria paralela (JMP): Fue propuesta en 1993 por Alpern [50]. Este modelo fue propuesto para afrontar los problemas de la MAAP con respecto a las operaciones en la memoria, las cuales se deben realizar en tiempo constante. Actualmente, las CPUs están compuestas de memorias en forma jerárquica, tales como; registros, cachés L1, cachés L2, y cachés L3. Este modelo está definido por un árbol jerárquico o módulos de memoria, donde las hojas del árbol corresponden a los procesadores y los nodos representan los módulos de memoria. Los módulos más cercanos a los procesadores tienen un acceso más

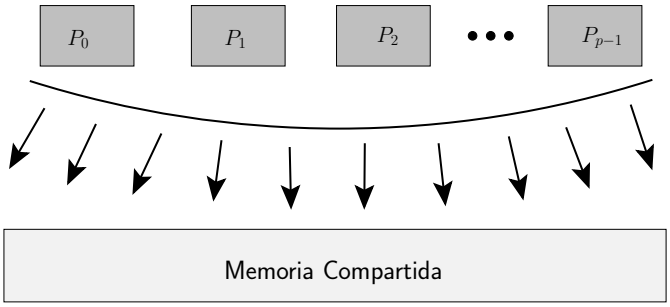


Figura 3.1: Modelo MAAP

rápido, pero son pequeños, y los módulos más lejanos a los procesadores tienen un acceso a la memoria más lento, pero cuentan con una mayor capacidad de almacenamiento. En la figura 3.2 se muestra este modelo, donde el i-ésimo nivel del modelo, define los siguientes parámetros: s_i es el número de elementos por bloque (tamaño del bloque), n_i el número de bloques y l_i la latencia entre cada nivel.

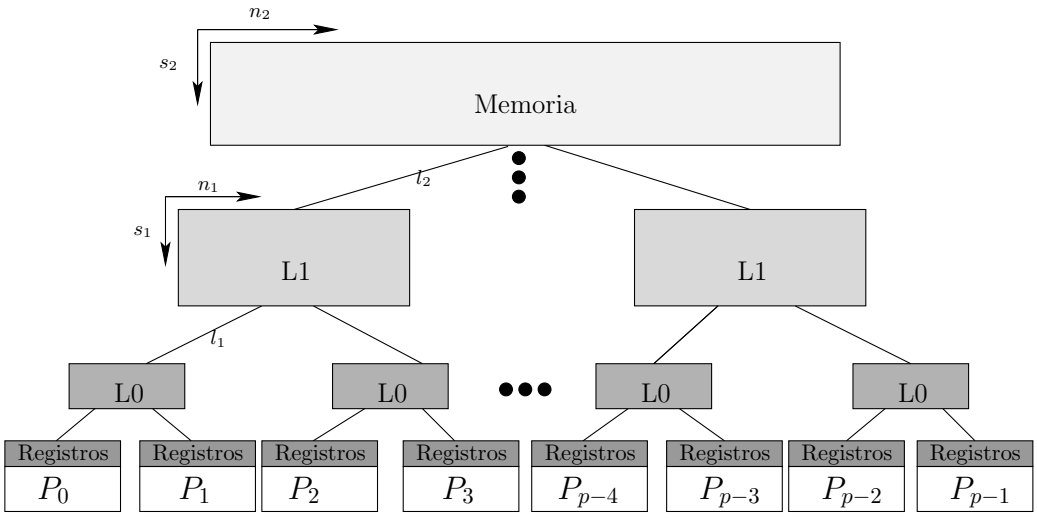


Figura 3.2: Modelo de jerarquía de memoria paralela

Comunicación Síncrona Paralela (CSP): Este modelo está enfocado en la comunicación, y fue publicado por Leslie Valiant en 1990 [51]. El modelo consiste de un número de procesadores con una memoria local de rápido acceso, conectada a través de la red, la cual, a su vez, es capaz de enviar y recibir mensajes hacia cualquier otro procesador (ver la figura 3.3). Cualquier algoritmo que se base en este modelo requiere de un bloque paralelo, el cual está compuesto de tres pasos:

- Cómputo local: p procesadores realizan hasta L cálculos locales.
- Comunicación global: Los procesadores pueden enviar y recibir datos entre ellos.
- Sincronización: Deberá existir una barrera para esperar a que todos los procesadores terminen sus cálculos.

El costo c del bloque, suponiendo que se usan p procesadores se define como:

$$c = \max_{\forall p}(w_i) + g \max_{\forall p}(h_i) + l \tag{3.4}$$

donde w_i es el tiempo de ejecución del i-ésimo procesador, h_i el número de mensajes usados por el procesador p_i, g es la capacidad de la red y l es el costo de la sincronización. Para un algoritmo que esté compuesto de S bloques, el costo final se define como la suma de cada uno los bloques:

$$C = \sum_{i=1}^{S} c_i \tag{3.5}$$

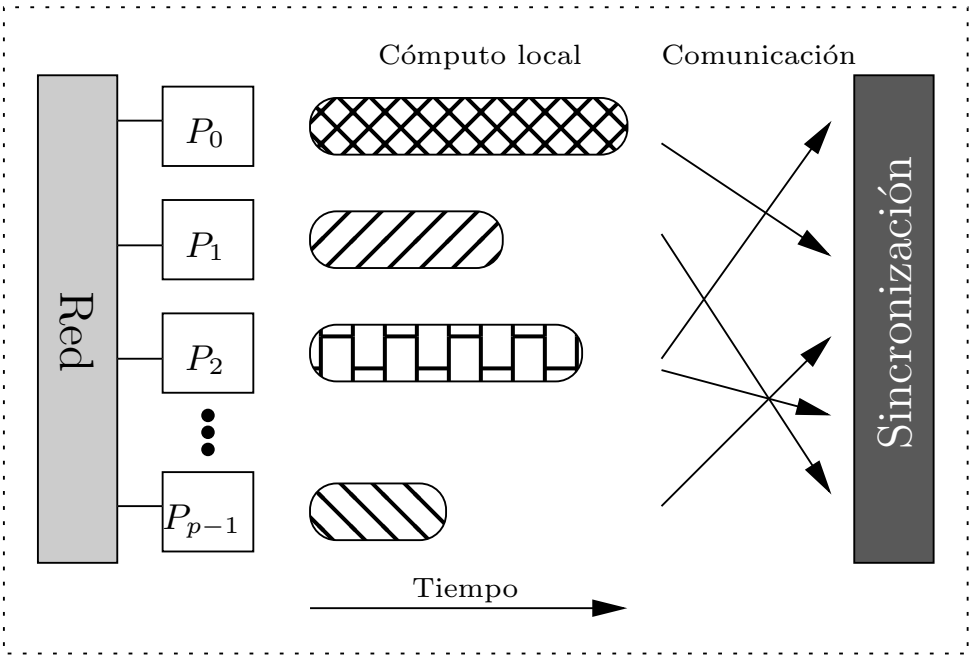


Figura 3.3: Una representación de los pasos del modelo CSP

LogP: El modelo LogP fue propuesto por Culler en 1993 [52]. Este modelo es similar a CSP, el cual se enfoca en modelar el costo de comunicación de un conjunto distribuido de procesadores (como una red de computadoras). En el modelo, el

costo de las operaciones locales está definido por un intervalo de tiempo, pero la red tiene *latencia y overhead*. Este modelo requiere de algunos parámetros los cuales se definen como:

Procesadores (p) : Define el número de procesadores a utilizar.

Latencia (L) : Es la cantidad de tiempo que existe en la comunicación de un mensaje que contenga una palabra (o un pequeño número de palabras) desde su recurso hasta el procesador objetivo. Puede decirse que es el retraso temporal que existe en la comunicación.

Overhead (o) : Es la cantidad de tiempo que un procesador invierte en la comunicación (enviar o recibir mensajes). Durante este tiempo, el procesador no puede realizar otras operaciones.

Gap (g) : Es la cantidad mínima de tiempo entre el paso sucesivo de mensajes en un procesador determinado.

Todos los parámetros, excepto por la cantidad de procesadores p, son medidos en ciclos. En la figura 3.4 se muestra el modelo con un ejemplo de comunicación con mensajes de una sola palabra. La diferencia que existe entre CSP y LogP, es que para el caso de CSP se usa una barrera de sincronización, mientras que LogP sincroniza por pares de procesadores. Éste modelo considera también el *overhead* al enviar o recibir mensajes.

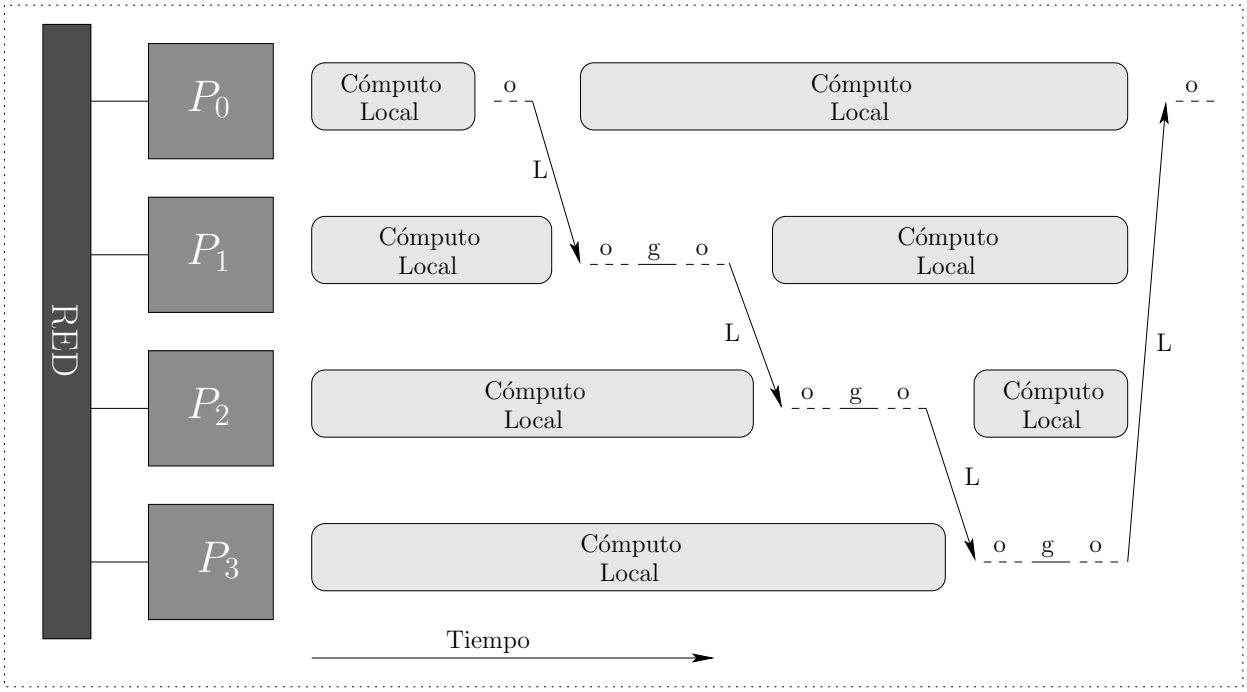


Figura 3.4: Ejemplo de comunicación usando el modelo LogP

Los modelos de computación paralela son útiles para analizar el tiempo de ejecución de un algoritmo paralelo, el cual se puede visualizar como una función que depende de n, p y otros parámetros específicos dependiendo el tipo de modelo que elijamos. Existen también otros aspectos importantes que debemos considerar en relación con los estilos de programación paralela. En la siguiente sección se mencionan los aspectos principales a tomarse en cuenta en la programación paralela.

3.3. Modelos de programación paralela

En la computación paralela también se debe analizar por qué los procesadores se comunican y cómo se programan. Por ejemplo, el modelo MAAP y jerarquía de memoria paralela usan el concepto de *memoria compartida*, mientras que LogP y CSP usan el modelo *paso de mensajes*. Estos dos modelos son muy conocidos en la programación paralela. Un modelo de programación paralela es una abstracción de los aspectos programables de un modelo de computación [53]. Los modelos de programación paralela son útiles para la implementación de un algoritmo. En las siguientes subsecciones se muestran los modelos más importantes de programación paralela.

3.3.1. Modelo implícito

Este modelo se refiere al paralelismo implícito haciendo uso de compiladores o herramientas de alto nivel, las cuales son capaces de lograr un grado de paralelismo automáticamente a partir de una secuencia de código fuente. La ventaja del paralelismo implícito es que todo el trabajo duro está hecho por el compilador, y casi logramos un beneficio similar al paralelismo explícito. La desventaja es que sólo funciona para problemas sencillos como ciclos o iteraciones independientes. En [54] se describe la estructura de un compilador capaz de realizar paralelismo implícito y explícito. *Map-Reduce* es conocida como una herramienta de programación paralela y ha sido usada en marcos de trabajo tales como; Hadoop, la cual ha dado muy buenos resultados en el procesamiento de datos a través de los sistemas distribuidos. Al igual que Hadoop, los lenguajes funcionales tal como Haskell or Racket se han beneficiado del uso de *Map-Reduce*.

También, OpenMP (del acrónimo *Open Multiprocessing*) se puede considerar como una API que realiza un semi-paralelismo implícito. El paralelismo que ofrece esta herramienta se basa en las recomendaciones dadas por el programador. OpenMP define un modelo portátil, escalable para el desarrollo de paralelismo, yendo desde un simple procesador multicore hasta super-computadoras.

La paralelización implícita es muy difícil de lograr para los algoritmos que no están basados en ciclos y se ha convertido en un tema de investigación muy activo en años recientes.

3.3.2. Modelo de memoria compartida

El modelo de memoria compartida pertenece al paralelismo explícito, donde los *hilos* pueden leer y escribir de forma asíncrona en un espacio común de memoria. Este modelo de programación funciona de forma natural con el MAAP y es utilizado cuando el problema se puede mapear a multiples procesadores, tales como; multicore o GPU. OpenMP es una herramienta muy conocida para el desarrollo de este modelo al igual que el modelo anterior, el cual está basado en los *pthreads* de Unix. Para el caso de las GPUs, existen herramientas como OpenCL y CUDA. Muchas de las veces, en un algoritmo paralelo basado en memoria compartida se requiere tener un buen manejo de los hilos, ya que puede ocasionar pérdida de rendimiento. Cuando un conjunto de hilos se están ejecutando de forma concurrente y escriben alguna información en un mismo segmento de memoria ocasiona un *deadlock*. En estos casos se debe suministrar una sincronización explicita o un mecanismo de control como por ejemplo; monitores, semáforos, operaciones atómicas y cerrojos, sólo por mencionar algunos. El uso de estos mecanismos permiten que los hilos trabajen de forma correcta, cada uno sin interferir con los demás. Estos modelos definen reglas y estrategias para mantener la consistencia en la memoria compartida.

3.3.3. Modelo paso de mensajes

La programación por paso de mensajes también es conocida como modelo distribuido y está basada en la comunicación de forma síncrona o asíncrona entre cada uno de los procesadores mediante el envío y la recepción de mensajes. Este modelo enfatiza la comunicación y la sincronización, por lo que realiza un cómputo distribuido. Dijkstra introdujo nuevas ideas para la consistencia de los mensajes en los sistemas distribuidos basados en el mecanismo de exclusión [55]. Este modelo trabaja de manera natural con CSP y LogP, los cuales fueron diseñados con el mismo paradigma.

Una herramienta estándar para el paso de mensajes es OpenMPI (del acrónimo *Open Message Passing Interface*), la cual se utiliza para el manejo de comunicación en la CPU a nivel multicore o a nivel distribuido. En ocasiones, se utiliza esta herramienta para distribuir el trabajo cuando se utilizan multiples GPUs.

3.4. Arquitecturas

Las arquitecturas de las computadoras definen la forma de comunicación u organización de cada uno de los procesadores y también especifican el manejo de memoria. Normalmente, una arquitectura de computadora está descrita por uno o dos modelos de computación. Es importante mencionar que el objetivo de las arquitecturas no es implementar un modelo. De hecho, es lo contrario, los modelos de computación tratan de modelar las arquitecturas de las computadoras. El principal objetivo de una arquitectura de computadora es que los programas, al ser ejecutados, sean más eficientes y

el tiempo de ejecución sea menor. En los últimos años, se han creado nuevos diseños, los cuales han conseguido un mayor desempeño y esto se debe a que la industria de hardware ha crecido exponencialmente. Hoy en día, las arquitecturas más importantes son multi-núcleos y muchos núcleos para CPU y GPU, respectivamente.

Desafortunadamente, las implementaciones no se ejecutarán más rápidamente con sólo comprar mejor hardware. Todos los métodos o implementaciones deben ser rediseñados con un enfoque paralelo, de tal manera que se pueda escalar en más procesadores. Los aspectos como: tipo de paralelismo (ver figura 3.5), la organización de la memoria y la comunicación entre cada procesador, ayudan a lograr una mejor implementación.

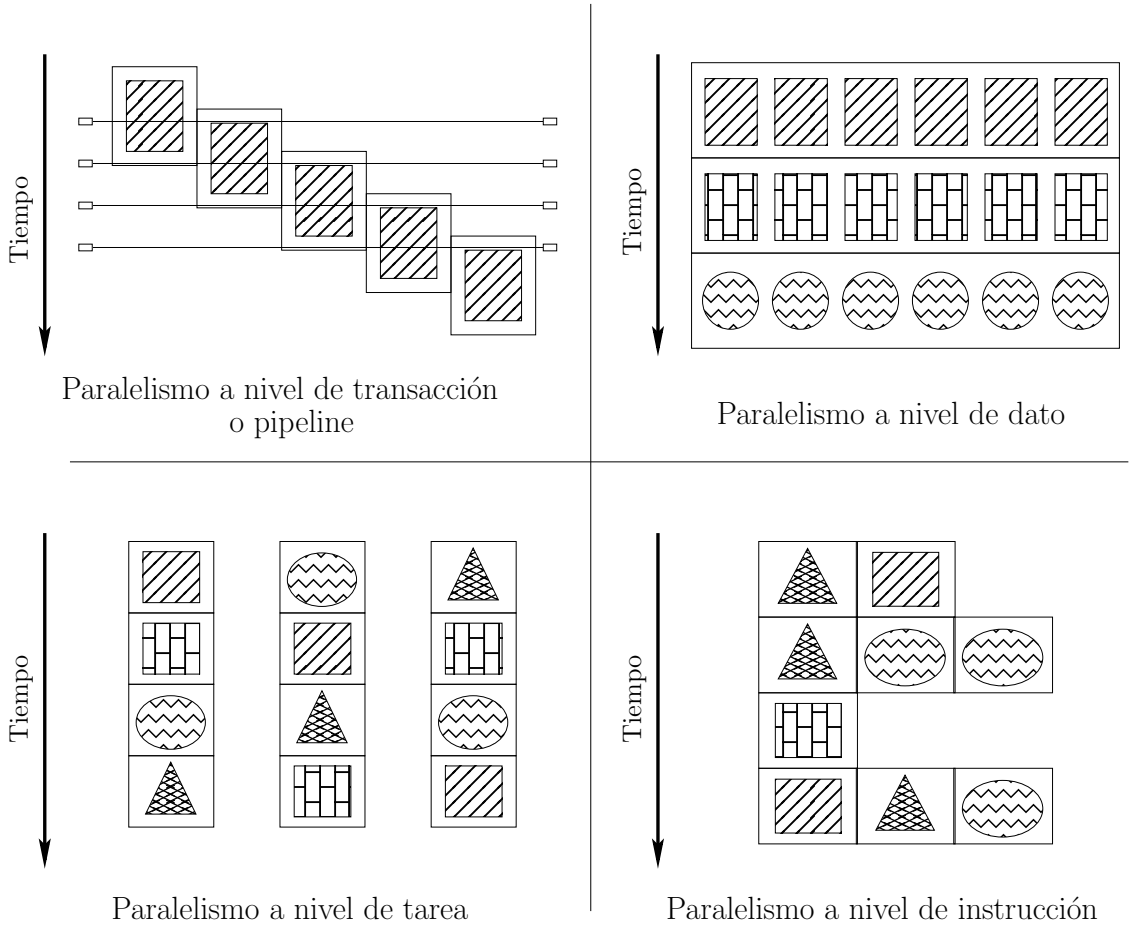


Figura 3.5: Tipos de paralelismo

3.5. Taxonomía de Flynn

Las arquitecturas se pueden clasificar usando la taxonomía de Flynn [56]. Flynn se dio cuenta que todas las arquitecturas se pueden clasificar en cuatro categorías.

Esta clasificación depende de dos aspectos: el número de instrucciones y el número de datos que se pueden manejar de forma paralela. La taxonomía de Flynn es la siguiente:

SISD (*Single Instruction Single Data*): En esta categoría solamente se puede realizar una instrucción para un flujo de datos al mismo tiempo. Aquí no existe paralelismo. Las CPUs antiguas de los años setentas y ochentas (como intel 8086 y 80486), estaban basadas en la arquitectura original de Von Neumann, las cuales todas eran del tipo SISD.

SIMD (*Single Instruction Multiple Data*): Pueden manejar una sola instrucción a múltiples datos, simultáneamente. Estas arquitecturas permiten el paralelismo a nivel de datos, el cual es útil para la aplicación de modelos matemáticos. Las computadoras vectoriales fueron las primeras en los años 70s y 80s en aplicar esta arquitectura. Hoy en día, las GPUs son clasificadas en esta categoría, ya que trabajan con muchos lotes de SIMD simultáneamente.

MISD (*Multiple Instruction Single Data*): En esta categoría se puede manejar diferentes tareas en un mismo dato. Esta arquitectura no es tan común y a menudo terminan siendo implementaciones específicas, las cuales llegan a ser redundantes. Estas implementaciones sirven para verificar los fallos en un sistema.

MIMD (*Multiple Instruction Multiple Data*): Es la arquitectura más flexible, la cual puede manejar una instrucción diferente para múltiples datos y con ella se puede lograr cualquier tipo de paralelismo. Sin embargo, la complejidad de la implementación es alta y muchas veces el *overhead* involucrado en el manejo de diferentes tareas y flujos de datos se convierte en un problema cuando tratamos de escalar el número de procesadores.

3.6. Unidad de procesamiento gráfico

Las unidades de procesamiento gráfico son procesadores de múltiples núcleos de alto rendimiento capaces de realizar un gran procesamiento de datos. Las GPUs fueron diseñadas con el objetivo de realizar procesamiento gráfico y pueden ser programadas con ayuda de Interfaces de Programación Aplicada (APIs) tales como DirectX y OpenGL. Hoy en día, las GPUs son procesadores de propósito general y diseñadas para trabajar con CUDA y OpenGL.

Hoy en día, las GPUs son utilizadas en una amplia gama de aplicaciones [57], tales como videojuegos, minería de datos, bioinformática, química y finanzas, sólo por mencionar algunas. En estas aplicaciones las GPUs suelen ser más rápidas que las versiones tradicionales, lográndose con ellas una aceleración de 10x. Las GPUs originalmente fueron diseñadas para la industria de los videojuegos. A pesar de esto, las GPUs han sido utilizadas para cálculos numéricos en la ciencia y la ingeniería, y cualquier aplicación en la que se requiera procesar grandes cantidades de operaciones

enteras y punto flotante, esto con el fin de simular fenómenos físicos de la manera más realista posible. Las aplicaciones numéricas han sido desarrolladas en lenguajes de alto nivel tales como Fortran y C. Dichas aplicaciones han sido implementadas en clusters compuestos de nodos multi-core con la interacción de una GPU. Estas aplicaciones usan OpenMP y MPI, esto con la finalidad de sacar provecho a todo el poder de cómputo disponible.

3.6.1. Diferencias entre CPU y GPU

Antes de adentrarnos en la arquitectura de las GPUs, es necesario visualizar cuales son las características de cada una de estas tecnologías. La principal diferencia entre la CPU y la GPU es que la GPU está especializada en cómputo intensivo y para problemas altamente paralelos. Es importante mencionar que no todos los problemas pueden ser resueltos con ayuda de una GPU. La GPU es un multiprocesador de propósito general, pero no sustituye a la CPU, por esta razón, las GPUs están diseñadas con más transistores, los cuales se dedican al procesamiento de datos en lugar de almacenamiento de datos en cache y un flujo de control (ver figura 3.6).

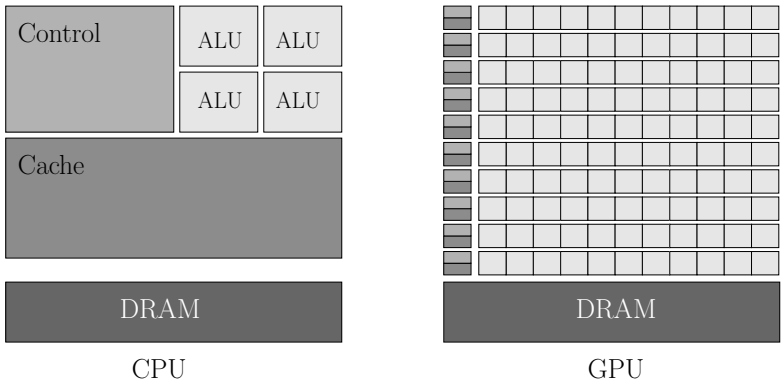


Figura 3.6: La arquitectura de la GPU difiere de la CPU ya que está diseñada con muchos núcleos pequeños, dando poco espacio para el control y almacenamiento de datos en la memoria cache.

Más específicamente, la GPU está diseñada para hacer frente a los problemas que se pueden expresar como SIMD. En otras palabras, un mismo programa es ejecutado en muchos datos de forma paralela. Debido a que un mismo programa ejecuta la misma operación en todos los datos, hay un menor requerimiento del flujo de control y la latencia que hay en acceso a memoria es menor. Es por ello que a las GPUs son conocidas como procesadores vectoriales. Muchas aplicaciones que procesan grandes volúmenes de datos pueden utilizar un paralelismo a nivel de datos para acelerar sus cálculos.

Adicionalmente, la GPU es más restrictiva que la CPU, ya que para trabajar con ellas es necesario cambiar nuestro modelo de programación. Las arquitecturas más recientes de las GPUs, tales como Fermi y Kepler de Nvidia, han añadido un

importante grado de flexibilidad al incorporar una memoria cache L2. Sin embargo, esta flexibilidad aún está lejos de la CPU. Cuando el problema no se logra solucionar con el paralelismo a nivel de datos, en ocasiones se puede solucionar con una CPU multi-core. Las CPUs luchan por mantener un equilibrio entre el poder de cómputo de propósito general y la funcionalidad, mientras que las GPUs apuntan a cálculos aritméticos altamente paralelos, introduciendo muchas restricciones.

3.6.2. CUDA

CUDA es acrónimo de *Compute Unified Device Architecture* y consiste en una plataforma de software y un modelo de programación creada por NVIDIA en 2006. CUDA viene con un entorno de software que permite a los desarrolladores el uso de C como lenguaje de programación de alto nivel. También incluye bibliotecas con una amplia variedad de aplicaciones, las cuales están basadas en STL (del acrónimo *Standard Template Library*). No solamente está disponible para el lenguaje C sino que también existen extensiones compatibles para Fortran, DirectCompute [58] y OpenACC [59]. CUDA es compatible con distintos niveles de computación [1] y cada nivel de computación depende de la arquitectura de la tarjeta. CUDA trabaja con todas las tarjetas de NVIDIA. Una de las ventajas al usar CUDA, es que los programas desarrollados para una GPU en particular también trabajan con otras GPUs, siempre y cuando el nivel de computación sea igual o mejor.

3.6.3. Modelo de programación

El modelo de programación CUDA para C es conocido como CUDA C. En este capítulo llamaremos CUDA al CUDA C, a la CPU le llamaremos *host* y a la GPU le llamaremos *device*, de ahora en adelante. En esencia hay tres abstracciones clave: la jerarquía de los grupos de hilos, el manejo de memoria y la sincronización. Estas abstracciones proporcionan paralelismo de datos de grano fino a nivel de hilos y paralelismo de datos de grano grueso a nivel tarea.

En un programa de CUDA, la ejecución inicia con el host. Cuando una función *kernel* se invoca o se pone en marcha, la ejecución del programa se realiza en el *device*, donde un conjunto de hilos se crean para aprovechar el paralelismo a nivel de datos. Todos los hilos son creados por el kernel durante la invocación de la función. Esta invocación es llamada *grid* (ver figura 3.7). El host ejecuta código serial, incluyendo el manejo de memoria y la gestión de las funciones. Tanto el host como el device mantienen su propia memoria, la memoria del host y la memoria en el device, respectivamente.

[1] El nivel de computación se refiere a las propiedades y características de cada GPU. Esto se determina dependiendo de la versión de la tarjeta (desde la 1.0 hasta 3.5). Cada versión tiene distintas funciones.

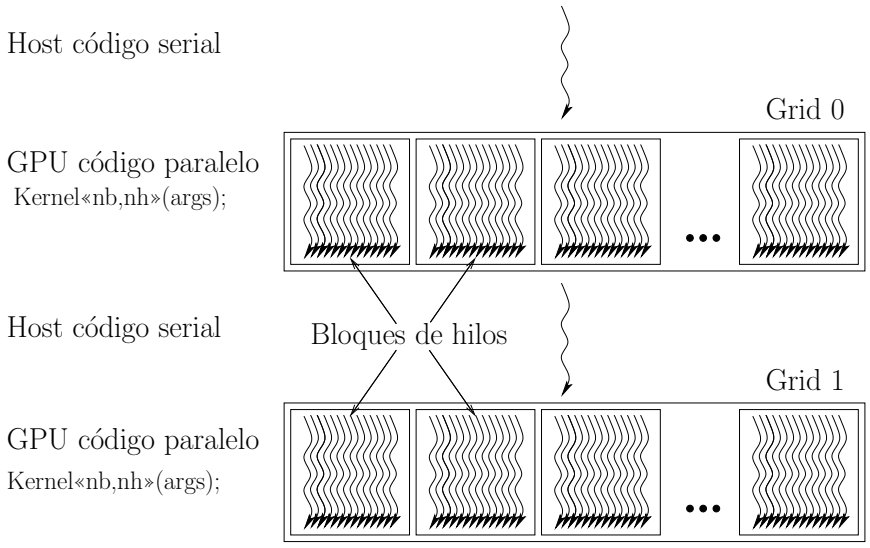


Figura 3.7: Ejecución de un programa en CUDA

Kernel

CUDA es una extensión de C con sus propias funciones y palabras reservadas. Esto permite la definición de funciones llamadas kernel, las cuales son ejecutadas en el device por hilos de CUDA.

La definición del kernel se efectúa usando la palabra reservada `__global__` antes del tipo de la función. Por ejemplo, el siguiente código realiza la suma de dos vectores A y B, y el resultado lo almacena en el vector C (ver código 3.1).

```
//Suma de dos vectores A + B de tamaño N
__global__ void SumaVect(float* A, float* B, float* C, int N)
{
        int i;
        i = threadIdx.x;
        if(i < N)
        {
                C[i] = A[i] + B[i];
        }
}
```

Código 3.1: Definición de un kernel

A cada hilo que se ejecuta se le asigna un identificador único que es accesible dentro del kernel mediante la variable `threadIdx.x`. Para que el host invoque el kernel, es necesario especificar el número de hilos a crear, para ser ejecutados dentro del device usando <<< ... >>>. Por ejemplo, la llamada del kernel `SumaVect` es de la siguiente forma:

```
//Suma de dos vectores A + B
int main(int argc,char* argv)
{
        ...
        SumaVect<<< 1 , N >>>( A , B , C , N );
        ...
        return 0;
}
```

Código 3.2: Invocación del kernel

En el ejemplo 3.2, invocamos N hilos con identificadores globales desde 0 hasta $N-1$. El número 1 en el interior de <<< ... >>> especifica la cantidad de *bloques* que serán creados dentro del kernel.

Jerarquía de hilos

Un conjunto de hilos forman un bloque y un conjunto de bloques forman un *grid* (en la figura 3.7 se observa esta jerarquía). Para asignar el trabajo a cada hilo y controlar su ejecución, los hilos están identificados con índices, los cuales determinan su posición dentro del bloque. Un hilo puede tener distintos índices, dependiendo de la forma del bloque: una dimensión, dos dimensiones y tres dimensiones.

Los bloques tienen sus propios índices para ser identificados dentro de un grid. Al igual que los bloques, el grid puede tener un máximo de tres dimensiones. Por lo tanto, los índices de los bloques pueden alcanzar hasta tres valores (x, y, z). Para identificar cada hilo y cada bloque al ejecutarse en el kernel, CUDA proporciona algunas palabras reservadas:

- **threadIdx** es el identificador del hilo dentro de un bloque.
- **blockIdx** es el identificador del bloque dentro de un grid.
- **blockDim** es el tamaño o el número de hilos por bloque.
- **gridDim** es el tamaño o el número de bloques por grid.

En el código 3.3, se muestra el uso de estos identificadores en la suma de dos matrices A y B de tamaño NxN.

```
// Definición del kernel
__global__ void SumaMat(float A[][], float B[][], float C[][],
    int N)
{
        int i,j;
        i = threadIdx.x;
        i = threadIdx.y;
        C[i][j] = A[i][j] + B[i][j];
```

```
}

//Suma de matrices A + B de tamaño NxN
int main(int argc,char* argv)
{
        ...
        int numbloques = 1;
        dim3 numhilos ( N , N );
        SumaMat<<< numBloques , numhilos >>>( A , B , C , N );
        ...

        return 0;
}
```

Código 3.3: Suma de matrices

Hay un límite en el número de hilos por bloque, ya que todos los hilos son creados en la misma GPU y deben compartir la memoria. Hoy en día, existen GPUs capaces de crear hasta 1024 hilos por bloque. El kernel puede ser ejecutado también con múltiples bloques, de modo que el número total de hilos es la suma de cada bloque. Por ejemplo, ejecutamos 32 bloques con 7 hilos cada uno. Esto nos da un total de 224 hilos. Si ejecutamos 7 bloques con 32 hilos, la cantidad de hilos es la misma, pero el desempeño del paralelismo es distinto. Esto se debe a la arquitectura de la GPU y la asignación que se realice.

La GPU tiene una cantidad de *cores* físicos, los cuales determinan la cantidad de SMs (del acrónimo *Streaming Multiprocesor*) físicos en la tarjeta. Éstos determinan el desempeño del programa de CUDA (ver figura 3.8). La GPU asigna automáticamente los bloques de hilos a cada SM disponible. Esto permite a las GPUs ejecutar *kernels* de acuerdo a sus capacidades. Esta política de planificación debe considerarse al determinar la cantidad de bloques. Si el número de bloques es menor a la cantidad de SMs disponibles, el cómputo no será explotado.

Manejo de memoria

Los hilos de CUDA tienen acceso a diferentes tipos de memoria. Cada hilo tiene una memoria local (registros). Esta memoria es privada para cada hilo, y puede almacenar todas las variables locales. Existen diferentes políticas de acceso para cada hilo. Por ejemplo, un hilo puede acceder a todas sus variables locales, pero no puede acceder a variables locales de otros hilos. Todos los hilos que pertenecen a un mismo bloque, pueden acceder a la memoria compartida de ese bloque, pero hilos de distintos bloques no pueden acceder a la misma memoria compartida. La única forma de comunicar hilos de diferentes bloques es con ayuda de la memoria global.

Memoria global : Es el medio de comunicación entre el host y el device. Por lo general, el host transfiere datos a la memoria global para ser procesados en la GPU y para obtener el resultado del procesamiento se procede de la misma

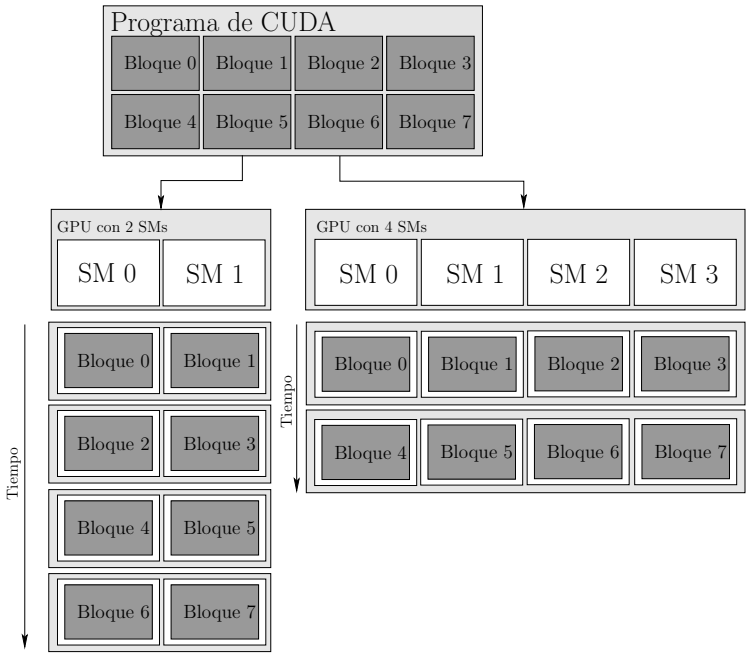


Figura 3.8: Asignación de los bloques a cada SM

forma. CUDA proporciona palabras reservadas como `cudaMalloc`, la cual sirve para asignar un espacio de memoria en el device.

Memoria compartida : Es el medio de comunicación entre cada hilo que pertenece a un bloque específico. Esta memoria es mucho más rápida que la memoria global. Si un elemento de la memoria global tiene que ser leído o escrito más de una vez, es una buena idea transferirlo a registros o memoria compartida. La memoria compartida es declarada en el kernel usando la palabra reservada `__shared__` antes del tipo de dato.

Regitros : Son segmentos de memoria para cada hilo. Esta memoria es más rápida que la memoria compartida, pero la capacidad de almacenamiento es menor con respecto a la memoria compartida y la memoria global. Aquí se almacenan las variables locales del hilo.

Existen otros temas de suma importancia acerca de las GPUs, de entre los que destacan la ejecución de operaciones concurrentes de una GPU. A estas operaciones, CUDA las llama *streams*. Los *streams* sirven para ejecutar un paralelismo a nivel de transacción, esto significa que podemos realizar operaciones asíncronas en el intercambio de información del host al device y viceversa. No todas las GPUs tiene esta capacidad, ya que esto depende del nivel de computabilidad de cada tarjeta. Por otra parte, los eventos de CUDA proporcionan los mecanismos de sincronización para complementar la ejecución paralela que habilitan los *streams*. Cuando se lanza un *kernel* en la GPU, el intercambio de información entre el host y el device es de forma

síncrona, ya que solo utiliza un solo *Stream*. Sin embargo, un conjunto de *kernels* se ejecutan de manera asíncrona, el objetivo de realizar estas operaciones es reducir el *overhead* y utilizar todo el recurso de hardware disponible con la finalidad de aumentar la aceleración del algoritmo paralelo.

En resumen, las Unidades de Procesamiento Gráfico son procesadores de múltiples núcleos de alto rendimiento, las cuales tienen la capacidad de realizar operaciones de forma vectorial. Existen distintas plataformas para programar las GPUs, una de las cuales es CUDA. CUDA es una extensión de lenguaje C. Esta plataforma tiene un modelo de programación muy flexible y su curva de aprendizaje es baja. CUDA permite definiciones de funciones (llamadas *kernel*), las cuales son ejecutadas directamente en la GPU. Cada *kernel* se puede ejecutar con múltiples hilos de forma paralela. Estos hilos son organizados en bloques y, a su vez, un conjunto de bloques forman, un grid. Esta jerarquía es de gran ayuda para diseñar nuevos algoritmos paralelos. Necesitamos entender todos estos conceptos para desarrollar un mejor algoritmo paralelo.

Capítulo 4

Hipervolumen

Dado que el hipervolumen es el único indicador de desempeño unario que es compatible con la dominancia de Pareto y a que se ha podido demostrar que su maximización es equivalente a lograr convergencia al verdadero frente de Pareto, su uso ha cobrado gran importancia en años recientes. Esto ha motivado a varios investigadores a buscar métodos más eficientes para calcularlo. Este capítulo presenta la definición formal y las características principales del hipervolumen, así como la definición de la contribución al hipervolumen. Para finalizar se muestran los algoritmos representativos del estado del arte que se han diseñado para calcularlo de manera exacta.

4.1. Indicador de hipervolumen

El hipervolumen es un indicador muy común que se utiliza para medir y comprar la calidad de las solcuiones finales producidas por un algoritmo evolutivo. La definición formal de la métrica del hipervolumen fue propuesta originalmente por Zitzler y Thiele en 1998 [60] , quienes lo definen como *el tamaño del espacio cubierto o dominado.*

Definición **4.1** ***El indicador de hipervolumen*** *Dado un conjunto de puntos* $\mathcal{P} \in \mathbb{R}^m$ *y un punto de referencia* y_{ref}*, donde* Λ *denota la métrica de Lebesgue o métrica* $\mathcal{S}$*, el hipervolumen es definido como:*

$$\mathcal{I}_h(\mathcal{P}) = \mathcal{S}(\mathcal{P}, y_{ref}) = \Lambda\left(\bigcup_{y\in\mathcal{P}}\{y'|y \prec y' \prec y_{ref}\}\right) \tag{4.1}$$

El hipervolumen se define como la medida de Lebesgue de la unión de los hipercubos que están limitados por un punto de referencia y_{ref}. Por ejemplo, para hacer el cálculo del hipervolumen para dos funciones objetivo, se define como el área de cobertura del $\mathcal{FP}_{conocido}$ en relación con el espacio objetivo de dichas funciones [61] . Esto equivale a la suma de todas las áreas rectangulares, delimitadas por un punto de referencia. En la figura 4.1 se muestra de forma gráfica la interpretación para dos y tres dimensiones.

El principal inconveniente de este indicador, es que no se conoce ningún algoritmo que determine el valor del indicador en tiempo polinomial, lo cual, limita un tanto su uso en la práctica. Sin embargo, existe una gran variedad de algoritmos para determinar de forma exacta el valor de este indicador, si bien éstos requieren un elevado costo computacional.

Bringmann y Friedrich demostraron que el calcular el hipervolumen como indicador en alta dimensionalidad es considerado NP-duro [62], lo cual implica que un algoritmo tendrá un tiempo de ejecución muy grande y costoso.

4.2. Contribución al hipervolumen

Como vimos antes, los AEMOS intentan encontrar la mejor aproximación al frente de Pareto y una de las formas de hacer esto es mediante el uso de indicadores como criterio de selección. De hecho este enfoque ha sido muy popular en años recientes particularmente usando la contribución al hipervolumen. El principal objetivo es discernir la diferencia entre cada solución generada por un AEMO. La contribución al hipervolumen de una solución refleja la influencia de un solo punto del conjunto de aproximación (ver figura 4.1). La contribución al hipervolumen de un individuo $a \in \mathcal{P}$ se define de la siguiente forma:

$$\mathcal{C}_a = \mathcal{S}(\mathcal{P}, y_{ref}) - \mathcal{S}(\mathcal{P} \backslash \{a\}, y_{ref}) \tag{4.2}$$

El algoritmo 4 muestra la estimación de la contribución de cada individuo del conjunto $\mathcal{P}$ usando el indicador del hipervolumen.

Algoritmo 4: Cálculo de la contribución al hipervolumen de un conjunto $\mathcal{P}$

Entrada: Una población $\mathcal{P}$, donde $\mathcal{P}_i = (y_{i,1}, ..., y_{i,n})$ y un punto de referencia $\mathcal{R} = (r_1, ..., r_n)$

Salida: Contribución al hipervolumen $C_{\mathcal{P}}$

```
1 C_P ← 0;
2 for cada elemento de P do
3 |   a ← P_i;
4 |   C_{P_i} ← S(P, y_ref) − S(P\{a}, y_ref);
5 return C_P;
```

Existen distintos algoritmos para calcular la contribución al hipervolumen los cuales son presentados en [42], sin embargo son algoritmos para dos y tres dimensiones, los cuales no son escalables para una mayor dimensión. Por lo tanto, para realizar el cálculo de la contribución al hipervolumen para más de tres dimensiones se utiliza la ecuación (4.2). Debido a que el cálculo de la contribución al hipervolumen requiere el valor exacto del hipervolumen, el costo computacional es elevado para más de tres dimensiones. El uso de la contribución al hipervolumen en un algoritmo evolutivo

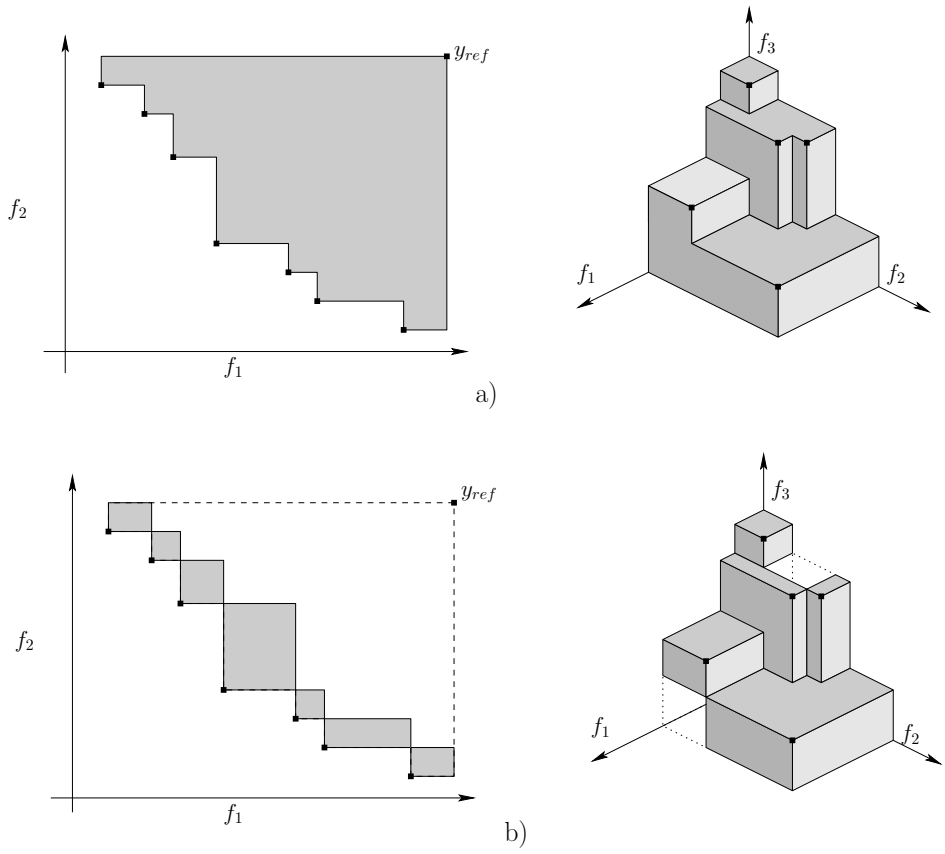


Figura 4.1: *a*) Interpretación gráfica del indicador de hipervolumen $\mathcal{I}_h(\mathcal{P})$ para dos y tres dimensiones, *b*) Interpretación gráfica de la contribución al hipervolumen de cada punto que pertenece a $\mathcal{P}$ para dos y tres dimensiones.

multi-objetivo genera grandes beneficios en la búsqueda de las aproximaciones a los óptimos de Pareto, pero es prohibitivo debido a su elevado costo computacional.

4.3. Características del indicador de hipervolumen

El hipervolumen es el único indicador de desempeño unario conocido a la fecha, que garantiza la monotonicidad estricta con respecto a la relación de dominancia de Pareto. Este indicador tiene dos ventajas importantes, descritas por Zitzler en [63]:

- Es sensible a cualquier tipo de mejora, es decir, cuando se calcula el hipervolumen para un conjunto $\mathcal{A}$ de soluciones, las cuales dominan a otro conjunto $\mathcal{B}$ de soluciones, por lo tanto el aporte del hipervolumen será de mayor calidad para el primer conjunto que para el segundo.
- Como resultado de la primera propiedad, el hipervolumen garantiza que para cualquier aproximación al conjunto $\mathcal{A}$ de soluciones que alcance el valor máximo

posible para un problema en particular, todo el conjunto de soluciones pertenezca al conjunto de óptimos de Pareto. En pocas palabras, el maximizar el valor del hipervolumen garantiza convergencia al conjunto de óptimos de Pareto.

Este indicador no es invariante a la escala, y es sensible a la elección del punto referencia. La elección del punto de referencia puede afectar el resultado del hipervolumen. En un AEMO, ya sea maximizar o minimizar el valor del hipervolumen, el punto de referencia debe ser escogido de acuerdo a las siguientes reglas:

- El punto referencia debe ser menor o igual que el vector objetivo ideal para la minimización del hipervolumen.
- El punto de referencia debe ser mayor o igual que el vector objetivo nadir para la maximización del hipervolumen.

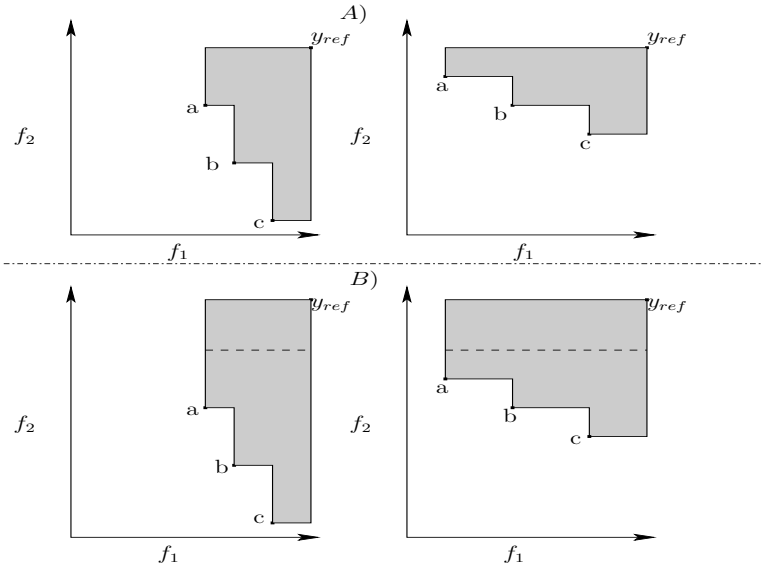


Figura 4.2: El valor relativo del hipervolumen

En la figura 4.2 se muestra que el valor del hipervolumen es sensible al punto de referencia. Por ejemplo, sean dos conjuntos de puntos no dominados $\mathcal{A}$ y $\mathcal{B}$ con dos funciones objetivo en conflicto, en la imagen (A) se cumple $\mathcal{I}_h(\mathcal{A}) > \mathcal{I}_h(\mathcal{B})$, mientras que en (B) $\mathcal{I}_h(\mathcal{A}) < \mathcal{I}_h(\mathcal{B})$.

4.4. Algoritmos para el cálculo del hipervolumen

Hoy en día existen muchos algoritmos para el cálculo del hipervolumen y cada uno tiene distinta complejidad. En esta sección se presenta los principales algoritmos propuestos a la fecha para determinar el valor del hipervolumen.

4.4.1. Hipervolumen mediante inclusión-exclusión

Este algoritmo es el más obvio y fácil de comprender para el cálculo del hipervolumen. Este método esta basado en el principio de matemática combinaria propuesto por Abraham de Moivre [64]. Este principio ha sido utilizado en matemáticas discretas y probabilidad. La forma en cómo trabaja este algoritmo es sumar todos los volúmenes obtenidos al intersectar un conjunto de volúmenes n-dimensionales dominados cuya cardinalidad sea impar y restar los volúmenes obtenidos al intersectar un conjunto de volúmenes n-dimesionales dominados cuya cardinalidad sea par.

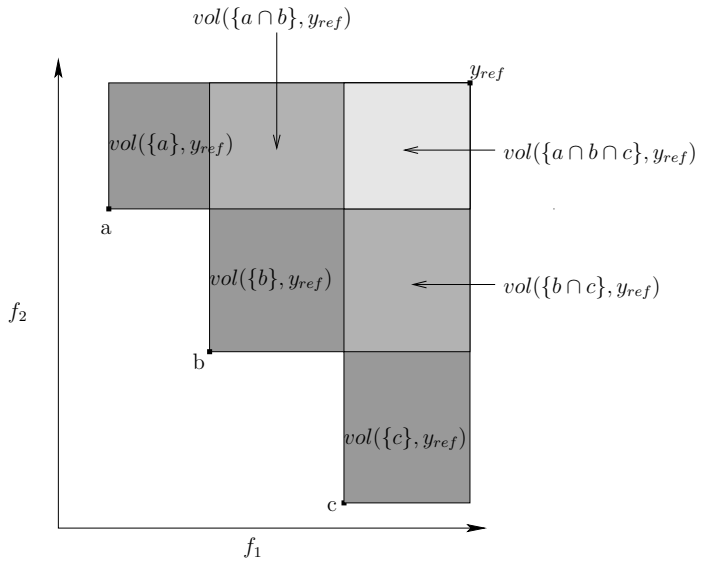


Figura 4.3: Representación gráfica del hipervolumen mediante inclusión-exclusión en dos dimensiones

Por ejemplo, dado un conjunto $\mathcal{P} = \{a, b, c\}$, el valor del hipervolumen $\mathcal{I}_h(\mathcal{P})$ se define como:

$$\mathcal{I}_h(\mathcal{P}) = \text{vol}(a) + \text{vol}(b) + \text{vol}(c) - \text{vol}(\{a \cap b\}) - \text{vol}(\{b \cap c\}) + \text{vol}(\{a \cap b \cap c\}) \tag{4.3}$$

En la figura 4.3 muestra la intersección de los volúmenes dominados para dos dimensiones; este método se puede generalizar para cualquier número de dimensiones.

La forma más general se presenta en el algoritmo 5 cuya complejidad es $\mathcal{O}(n2^m)$ (donde n es la dimensión y m es la caridad de puntos), siendo exponencial con respecto al número de puntos del conjunto $\mathcal{P}$.

Algoritmo 5: Algoritmo para el cálculo del hipervolumen mediante inclusión-exclusión

Entrada: Un conjunto de puntos no dominados $\mathcal{P}$ cuya carnalidad es m y la dimensión es n, donde $\mathcal{P}_i = (y_{i,1}, ..., y_{i,n})$ y un punto de referencia $\mathcal{R} = (r_1, ..., r_n)$

Salida: $\mathcal{I}_h(\mathcal{P})$

```
1 Vol ← 0;
2 for cada subconjunto s de P do
3     Vol ← Vol+ intersección(s) * (−1)^(‖s‖+1);
4 return Vol;
```

4.4.2. Hipervolumen mediante la medida de Lebesgue

Este algoritmo fue propuesto por Fleischer [65], quien observó que para cualquier espacio cubierto por un conjunto de puntos no dominados se puede identificar un politopo[1] rectangular que no intersecta con alguna otra región; en otras palabras esta región n-dimensional rectangular es disjunta. La principal idea es identificar el politopo disjunto del espacio formado por un conjunto de puntos no dominados y determinar su volumen. Esto se realiza con cada espacio disjunto, y luego se procede a acumular el valor del volumen y a remover el politopo rectangular. Este proceso se repite hasta que no se tenga ningún politopo disjunto.

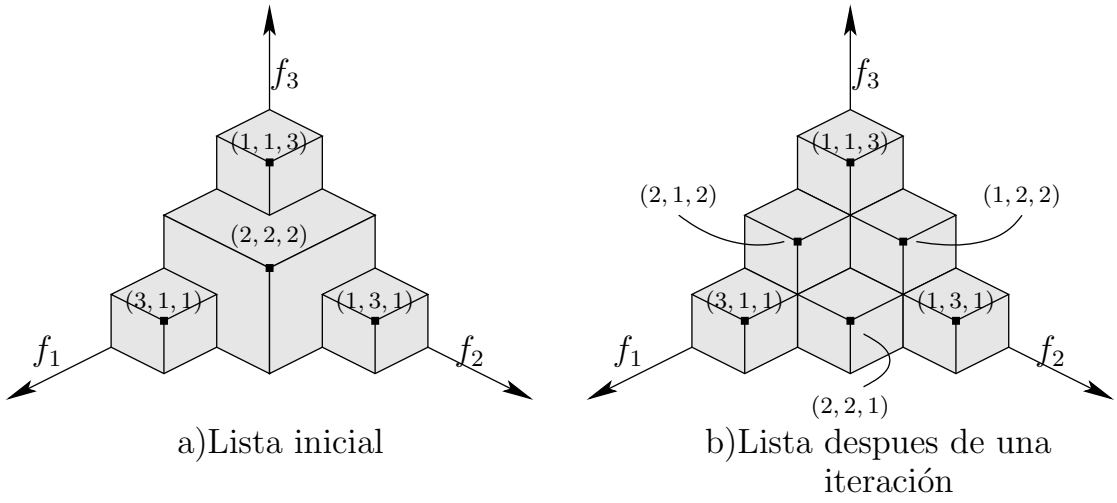


Figura 4.4: Iteración del algoritmo que usa la medida de lebesgue

En la figura 4.4 se muestra una iteración del algoritmo, donde se encuentra el politopo disjunto y se remueve del conjunto no dominado. Este algoritmo fue pensado

[1]Un politopo es la generalización a cualquier dimensión de un polígono. Este termino fue creado por Alicia Boole Stott.

para tener una complejidad polinomial. Sin embargo más tarde se demostró empíricamente por While [66] que la complejidad es exponencial con respecto al número de objetivos. La complejidad para el peor caso de este algoritmo es $\mathcal{O}(2^{n-1})$ (ver algoritmo 6).

Algoritmo 6: $HLebMeasure(\mathcal{P}, \mathcal{R}, n, m)$

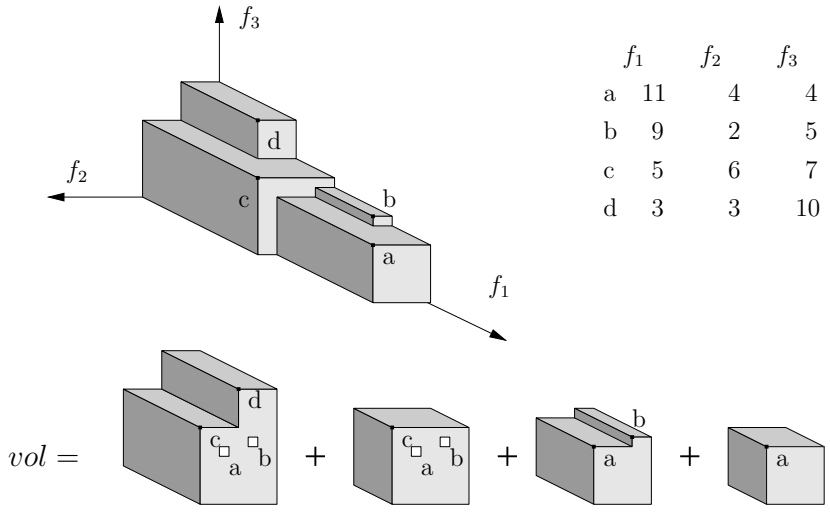
Entrada: Un conjunto de puntos no-dominados $\mathcal{P}$ cuya cardinalidad es m y la dimensión es n, donde $\mathcal{P}_i = (y_{i,1}, ..., y_{i,n})$ y un punto de referencia $\mathcal{R} = (r_1, ..., r_n)$

Salida: $\mathcal{I}_h(\mathcal{P})$

```
1  hipervol ← 0;
2  Inicializar la lista pl con los puntos no dominados ordenados con respecto a un
   objetivo;
3  while pl ≠ ⊘ do
4      Obtenemos el primer elemento de la lista pl y lo asignamos a p;
5      Asignamos a pl el resto de la lista;
6      Determinamos el punto a que sea el límite para formar el politopo
       rectangular disjunto, el cual es dominado solamente por p;
7      hipervol ← hipervol + vol(p, a);
8      Asignamos a ql los puntos generados al crear el politopo rectangular (p, a)
       de p;
9      if ql ≠ ⊘ then
10         Agregamos el conjunto de puntos ql a pl;
11 return hipervol;
```

4.4.3. Algoritmo HSO

Del acrónimo *Hypervolume by Slicing Objectives* (HSO) se desprende que este algoritmo procesa objetivos en lugar de puntos. A pesar de este enfoque, la complejidad de HSO es exponencial con respecto al número de objetivos en el peor de los casos, pero se ejecuta en un tiempo menor, aproximadamente de dos a tres órdenes de magnitud que dicha complejidad. Este algoritmo fue considerado tanto para medir la calidad de las soluciones de un AEMO, como para usarse como criterio de selección de un algoritmo evolutivo. El algoritmo HSO trabaja con un conjunto de puntos no dominados previamente ordenados con respecto al primer objetivo. Estos puntos se utilizan para crear secciones transversales a lo largo de este objetivo. El recorrido se realiza a lo largo de las secciones transversales. Cada una de las secciones contiene una lista de puntos de $n-1$ objetivos, cada una con la profundidad de la sección correspondiente al primer objetivo. Después, a cada una de las listas se expande en un conjunto de listas en $n-2$ objetivos. Estos conjuntos de listas se combinan de nuevo para hacer un solo conjunto. Esta expansión se realiza $n-1$ veces, con la finalidad de tener listas de un solo objetivo. Los volúmenes presentados por estas listas se suman para dar el valor total del hipervolumen.



	f_1	f_2	f_3
a	11	4	4
b	9	2	5
c	5	6	7
d	3	3	10

Figura 4.5: Ejemplo de la creación de las rebanadas en un espacio de tres dimensiones; cada rebanada tiene su propia lista de puntos

While [67] demostró que la complejidad de este algoritmo es de $\mathcal{O}(m^{n-1})$, el cual realiza una mejora en comparación con el algoritmo basado en la medida de Lebesgue. Sin embargo, HSO no logra resolver el problema para alta dimensionalidad.

Algoritmo 7: $HSO(\mathcal{P}, \mathcal{R}, n, m)$

Entrada: Un conjunto de puntos no dominados $\mathcal{P}$ cuya cardinaldad es m y la dimensión es n, donde $\mathcal{P}_i = (y_{i,1}, ..., y_{i,n})$ y un punto de referencia $\mathcal{R} = (r_1, ..., r_n)$

Salida: $\mathcal{I}_h(\mathcal{P})$

```
1  Ordenar el conjunto de puntos P con respecto al primer objetivo y asignarlo a
   pl;
2  s ← {(1, pl)};
3  k ← 1;
4  while k < n − 1 do
5      s' ← {};
6      for cada elemento (x, ql) de la lista en s do
7          for cada elemento (x', ql') de la lista slice(ql, k) do
8              Agregar el elemento (x' ∗ x, ql') a la lista s';
9      s ← s';
10     k + +;
11 vol ← 0;
12 for cada elemento (x, ql) de la lista s do
13     vol ← vol + x ∗ ‖head(ql)[n] − R[n]‖;
14 return vol;
```

4.4.4. Algoritmo FPL

Este algoritmo fue propuesto por Fonseca, Paquete y López Ibáñez (FPL) [68] y constituye una mejora al algoritmo HSO. El análisis del algoritmo HSO en [67] ayudó a crear este algoritmo, el cual consta de tres principales mejoras:

1. Agregan un algoritmo más eficiente para el caso de tres dimensiones el cual es utilizado como caso base para un mayor número de objetivos. Este algoritmo fue propuesto por Paquete en [68]. Este algoritmo permite trabajar a un nivel menor y evita cortes que no son necesarios.

2. Otra mejora fue el adaptar una estructura de datos para almacenar los puntos no dominados, esto con la finalidad de evitar cortes repetidos. La estructura de datos que se considera es una lista circular doblemente ligada, la cual permite que las operaciones se realicen en $\mathcal{O}(nm \log m)$, donde m es la cantidad de puntos del conjunto y n la dimensión.

3. Por último, esta estructura de datos se extiende para almacenar el estado actual de cada punto, así como los valores intermedios del hipervolumen, permitiendo que el árbol de recursión reutilice los resultados previos.

El algoritmo está organizado en dos ciclos consecutivos (ver algoritmo 8). En el primer ciclo, durante la dimensión actual i se recorre de forma descendente y todos los puntos encontrados de las listas con dimensiones menores al actual son eliminados. Esto se realiza hasta alcanzar el valor mínimo. Al igual que HSO, de forma recursiva el algoritmo calcula el valor del hipervolumen del politopo de una dimension menor con los puntos restantes.

En el segundo ciclo, sombre la misma dimensión i se recorre de forma ascendente para volver a insertar un punto en las listas de menor dimensión. Como en HSO, se multiplica la distancia de la rebanada por el valor del volumen. Este procedimiento se mantiene en ambos recorridos (ascendente y descendente). Con estas mejoras, este algoritmo logra reducir el tiempo de ejecución a $\mathcal{O}(m^{n-2} \log m)$ en comparativa a la propuesta original.

4.4.5. Cálculo del hipervolumen mediante la medida de Klee

El problema de la medida de Klee (del acrónimo Klee Measure Problem o KMP) ha sido estudiado en el área de geometría computacional, este problema fue propuesto por Victor Klee en 1977. Virtor Klee se cuestionó si es posible determinar de forma eficiente la unión de rangos rectangulares que pertenecen a un espacio n-dimensional. Esto puede ser definido como el producto cartesiano de n intervalos de números reales, donde n es considerado un valor constante. En la figura 4.6 se muestra un ejemplo de este problema para dos dimensiones, este problema se puede solucionar en tiempo $\mathcal{O}(m \log m)$, este algoritmo fue propuesto por Victor Klee [69], el cual está basado en un simple ordenamiento.

Algoritmo 8: $H(i, L_i, \mathcal{R}, len)$

Entrada: La dimensión i, una lista doblemente ligada L con todos los puntos, un punto de referencia $\mathcal{R} = (r_1, ..., r_n)$ y el tamaño de lista len

Salida: $\mathcal{I}_h(\mathcal{P})$

```
1  if i = 3 then
2      hvol ← HvP(L, R, len) //Caso base de tres dimensiones//;
3  else if i > 3 then
4      hvol ← 0;
5      p ← nil(L_i);
6      while len > 1 do
7          p ← prev_i(p);
8          eliminar(p, i);
9          len ← len − 1;
10     q ← prev_i(p);
11     H ← H(i − 1, L_{i−1}, R, len);
12     while p ≠ nil(L_i) do
13         hvol ← +H ∗ (p_i − q_i);
14         reinsertar(p, i);
15         len ← len + 1;
16         q ← p;
17         p ← sig_i(p);
18         H ← H(i − 1, L_{i−1}, R, len) ;
19     hvol ← hvol + H ∗ (r_i − q_i);
20 return hvol;
```

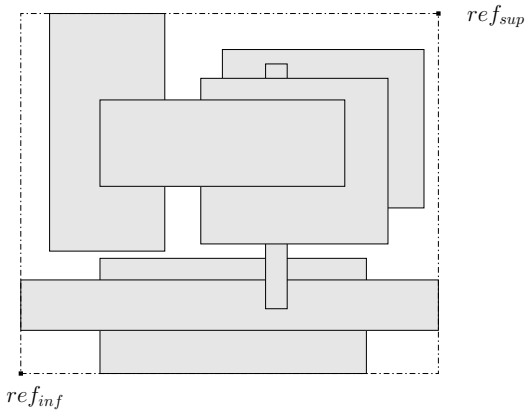


Figura 4.6: Representación gráfica de un KMP en dos dimensiones

En este campo existen distintos algoritmos para el cálculo de esta medida, pero el algoritmo más eficiente es el propuesto por Overmars y Yap [70]. Existe una gran relación entre un KMP y el hipervolumen, Karl Bringmann logra demostrar que el hipervolumen es un subconjunto de un caso especial del KMP (Unitcube-KMP [2]) [71]. Bringmann toma una instancia del hipervolumen y extiende el valor de la longitud de cada uno de los rectángulos del hipervolumen a un valor Δ. Con este proceso, logra inferir que el nuevo volumen generado es un Unitcube-KMP, el cual proviene de una instancia del hipervolumen (ver figura 4.7).

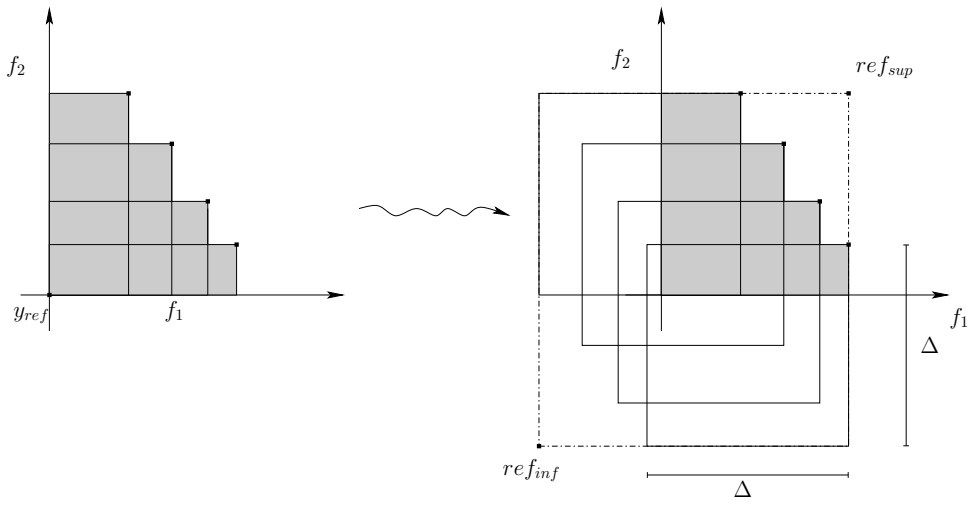


Figura 4.7: Extensión del hipervolumen a un KMP

Beume y Rudolph adaptaron el algoritmo de Overmars y Yap para determinar el

[2]El Unitcube-KMP es un caso especial del KMP, en el cual cada lado del politopo que pertenece a la región tiene la misma longitud.

valor del hipervolumen [72], el cual fue llamado HOY. Este algoritmo está basado en la propuesta original de Overmars y Yap, donde utiliza un árbol kd[3] para dividir el espacio cubierto por un politopo formado por cada punto no dominado.

HOY inicia con el politopo rectangular formado con todos los puntos no dominados, este politopo está acotado por una región, la cual es definida por dos puntos de referencia. Esta región es dividida en cada llamada recursiva, la cual forma el árbol kd (ver figura 4.8). El nodo hoja se genera cuando la región contiene un *enrejado*. El volumen del enrejado se determina mediante el principio de inclusión-exclusión.

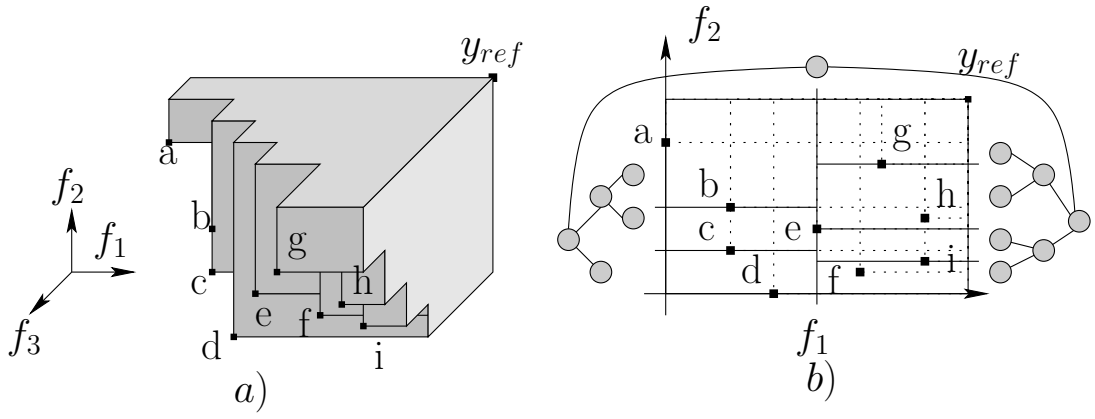


Figura 4.8: División de un conjunto de puntos no dominados en tres dimensiones mediante un árbol kd, donde la figura *a*) es el volumen original, la cual es proyectada en la figura *b*) en dos dimensiones con ayuda de la partición ortogonal.

HOY se divide en dos partes fundamentales: primero, verifica si el politopo cubre toda la región y si es un enrejado. En caso de ser un enrejado, se determina el valor del volumen. En cualquier otro caso, se divide la región en dos nodos (nodo izquierdo y nodo derecho) y se genera un nivel más al árbol kd. La división se realiza dependiendo de la dimensión candidata a formar un enrejado. Para cada nodo se realiza un filtro de puntos, donde se mantienen los puntos que cubran la región (izquierda o derecha). Este proceso se realiza de forma recursiva hasta encontrar un enrejado. La suma del volumen de cada enrejado forma el valor total del hipervolumen (ver algoritmo 9). La complejidad de este algoritmo es $\mathcal{O}(m \log m + m^{n/2} \log m)$, para m puntos de $n \geq 3$ dimensiones, donde establece una cota superior de $\Omega(m \log m + m^{n/2})$.

En resumen, el hipervolumen es uno los indicadores más populares en el de área optimización multiobjetivo. Este indicador tiene diversas ventajas con respecto a otros indicadores. Sin embargo tiene el inconveniente de que no existe ningún algoritmo que determine el valor exacto del indicador en tiempo polinomial, lo cual, limita un tanto

[3]El árbol kd es una estructura de datos, la cual se utiliza para el particionado ortogonal de un espacio, el cual organiza los puntos de un espacio en k dimensiones, es una generalización de un árbol binario.

Algoritmo 9: $HOY(\mathcal{P}, region, split)$

Entrada: Conjunto de puntos $\mathcal{P}$, región definida por dos puntos de referencia ($region$) y división del conjunto $\mathcal{P}$ ($split$)
Salida: $\mathcal{I}_h(\mathcal{P})$

```
for cada p ∈ P do
    if la región es completamente cubierta por p en n − 1 objetivos then
        Sumar el valor del volumen de la región a vol;
        Eliminar los puntos de p a partir de P;
if P = ⊘ then
    return vol;
if los puntos en P forman un enrejado then
    Sumar el valor del volumen del enrejado a vol;
else
    if al menos un elemento de P intersecta la región en la dimensión split
    then
        Dividir la región en dos partes en base a la media de la lista de puntos
        de P;
        HOY(P_1, region1, split);
        HOY(P_2, region2, split);
    else if el número de elementos que no intersectan a la región es > √||P|| 
    then
        Dividir la región en dos partes con base en la media de la lista de
        puntos que no intersectan a la región de P;
        HOY(P_1, region1, split);
        HOY(P_2, region2, split);
    else
        HOY(P, region, split + 1);
return hvol;
```

su uso en la práctica. En la comunidad de computación evolutiva, se han desarrollado distintas estrategias para calcular el indicador de hipervolumen. Los algoritmos más representativos son FPL y HOY, los cuales tiene una menor complejidad en comparación a los propuestos originalmente. A pesar que el algoritmo HOY tiene una menor complejidad, los estudios realizados en [73] muestran que el algoritmo FPL tiene una mejor implementación con respecto al algoritmo HOY. Aunque estos algoritmos sean los mejores, es complicado utilizarlos con un mayor número de objetivos. Hoy en día, los investigadores han diseñado nuevas estrategias para solucionar el problema. Una de ellas es realizar un muestreo para aproximar las contribuciones al hipervolumen de un conjunto de soluciones $\mathcal{P}$. Como se menciono anteriormente este método fue nombrado HypE [47], y está basado en el método de Monte Carlo. La desventaja de este algoritmo, es que es muy sensible al incremento en el número de dimensiones de un problema multi-objetivo. En esta tesis se aborda el problema del cálculo de la contribución al hipervolumen de manera exacta para usarse como criterio de selección de un AEMO con ayuda de las Unidades de Procesamiento Gráfico.

Capítulo 5

Modelo Propuesto

El cálculo de la contribución al hipervolumen de forma exacta se vuelve sumamente costoso (computacionalmente hablando) conforme se incrementa el número de objetivos y limita un tanto su uso en la práctica en un optimizador multi-objetivo. Por esta razón, el objetivo principal de nuestra propuesta es lidiar con el problema del cálculo de la contribución al hipervolumen de forma exacta para un optimizador multi-objetivo. La principal idea de nuestro modelo es utilizar todos los recursos de hardware disponibles para calcular la contribución al hipervolumen de una manera más eficiente. Por ese motivo, nuestro modelo está basado en dos distintas estrategias de paralelismo: paralelismo a nivel de datos y paralelismo a nivel de transacción.

En este capítulo se describe el modelo que es capaz de resolver, de una mejor manera, el cálculo de la contribución al hipervolumen para un optimizador multi-objetivo, el cual logra reducir el tiempo de ejecución de un algoritmo evolutivo multi-objetivo basado en el indicador de hipervolumen de forma exacta. Nuestra propuesta es un algoritmo con un enfoque paralelo para las GPUs. Se muestra la forma en que se realiza el paralelismo a nivel de datos, así como el paralelismo a nivel de transacción. Adicionalmente, se muestra la interacción entre la CPU y la GPU, la cual involucra el envío, el procesamiento y la recepción de los datos. Esto, con la finalidad de efectuar ambas estrategias de paralelismo.

5.1. Descripción del modelo

Antes de adentrarnos a describir nuestra propuesta, iniciamos por definir la notación que usaremos para describir nuestra propuesta.

Sea $d \in \mathbb{N}$ la dimensión del conjunto de puntos $\mathcal{P} \subseteq \mathbb{R}^d$, definimos a una *caja* como el conjunto $[x_1, y_1] \times \cdots \times [x_d, y_d] \subseteq \mathbb{R}^d$, donde $x_i \leq y_i$, se cumple para $\forall i$. Para un conjunto $\mathcal{P} \subseteq \mathbb{R}^d$ definimos $Vol(\mathcal{P})$ (o volumen) como la métrica de Lebesgue de $\mathcal{P}$. Por otra parte, para una *caja* C definimos $Vol_C(\mathcal{P})$ como el volumen de $\mathcal{P}$ dentro de C. Adicionalmente, denominamos a $\mathcal{C}_{\mathcal{P}_i}$ como la contribución del hipervolumen del i-ésimo elemento del conjunto $\mathcal{P}$ y denominamos a $\mathcal{C}_{\mathcal{P}}$ como el conjunto de contribuciones al hipervolumen de $\mathcal{P}$.

Este modelo está dividido en dos estrategias diferentes, las cuales dependen del número de objetivos (o dimensiones) a optimizar de un problema multi-objetivo. En primera instancia, se resuelve el cálculo de $\mathcal{C}_{\mathcal{P}}$ para dos dimensiones en el espacio de los objetivos de un problema multi-objetivo, donde cada hilo de la GPU determina la contribución al hipervolumen de manera independiente. La segunda estrategia resuelve el problema para más de dos dimensiones de un problema multi-objetivo, para lo cual recorta el volumen generado por $\mathcal{P}$ ($Vol(\mathcal{P})$), con el objetivo de eliminar volúmenes que no aportan información a $\mathcal{C}_{\mathcal{P}_i}$.

5.1.1. Solución para dos funciones objetivo en conflicto

El caso base de un problema multi-objetivo es el manejo de dos dimensiones (o dos funciones objetivo en conflicto). Para este caso el cálculo de la contribución al hipervolumen no es costoso, puesto que determinar la contribución al hipervolumen tiene un costo computacional $\mathcal{O}(n \log n)$ (donde n es la carnalidad de $\mathcal{P}$) [42]. Por está razón, se considera una estrategia diferente para determinar $\mathcal{C}_{\mathcal{P}}$. La contribución al hipervolumen para un punto en particular $\mathcal{P}_i$ en dos dimensiones, se considera como el área que se encuentra disjunta del volumen generado por $\mathcal{P}\backslash\{\mathcal{P}_i\}$ (ver figura 5.1).

En la figura 5.1 se observa que para calcular la contribución al hipervolumen de un punto no dominado $\mathcal{P}_i$ en particular, necesitamos conocer los puntos no dominados que colindan al punto $\mathcal{P}_i$. Con este análisis, observamos que no es necesario determinar $\mathcal{S}(\mathcal{P}, y_{ref}) - \mathcal{S}(\mathcal{P}/\{\mathcal{P}_i\}, y_{ref})$ (ver capítulo 4) para cada punto $\mathcal{P}_i \in \mathcal{P}$ en dos dimensiones. Tomando en cuenta que el conjunto de puntos generados por el optimizador multi-objetivo son no dominados (ver capítulo 2), se cumple lo siguiente:

Definición **5.1** *Sea un conjunto de puntos no dominados entre sí, denotado por* $\mathcal{P} \subseteq \mathbb{R}^2$ *con* $|\mathcal{P}| = k$*, donde* $\mathcal{P}_i = [p_{i,1}, p_{i_2}] \in \mathcal{P}$*, cuyos componentes corresponden a cada función objetivo (o cada dimensión). Si ordenamos el conjunto de puntos no dominados con respecto al primer objetivo:*

$$p_{1,1} \leq p_{2,1} \leq p_{3,1} \leq \cdots \leq p_{i,1} \leq \cdots \leq p_{k,1} \tag{5.1}$$

Implica que:

$$p_{1,2} \geq p_{2,2} \geq p_{3,2} \geq \cdots \geq p_{i,2} \geq \cdots \geq p_{k,2} \tag{5.2}$$

Se dice que un punto $\vec{u} = [u_1, ..., v_k]$ no domina a otro vector $\vec{v} = [v_1, ..., v_k]$ (denotado por $\vec{u} \npreceq \vec{v}$), si al menos una de estas dos ecuaciones se cumple:

$$\forall i \in \{1, \ldots, k\}, u_i > v_i \wedge \exists i \in \{1, \ldots, k\} : u_i \geq v_i \tag{5.3}$$

$$\forall i \in \{1, \ldots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i \geq v_i \tag{5.4}$$

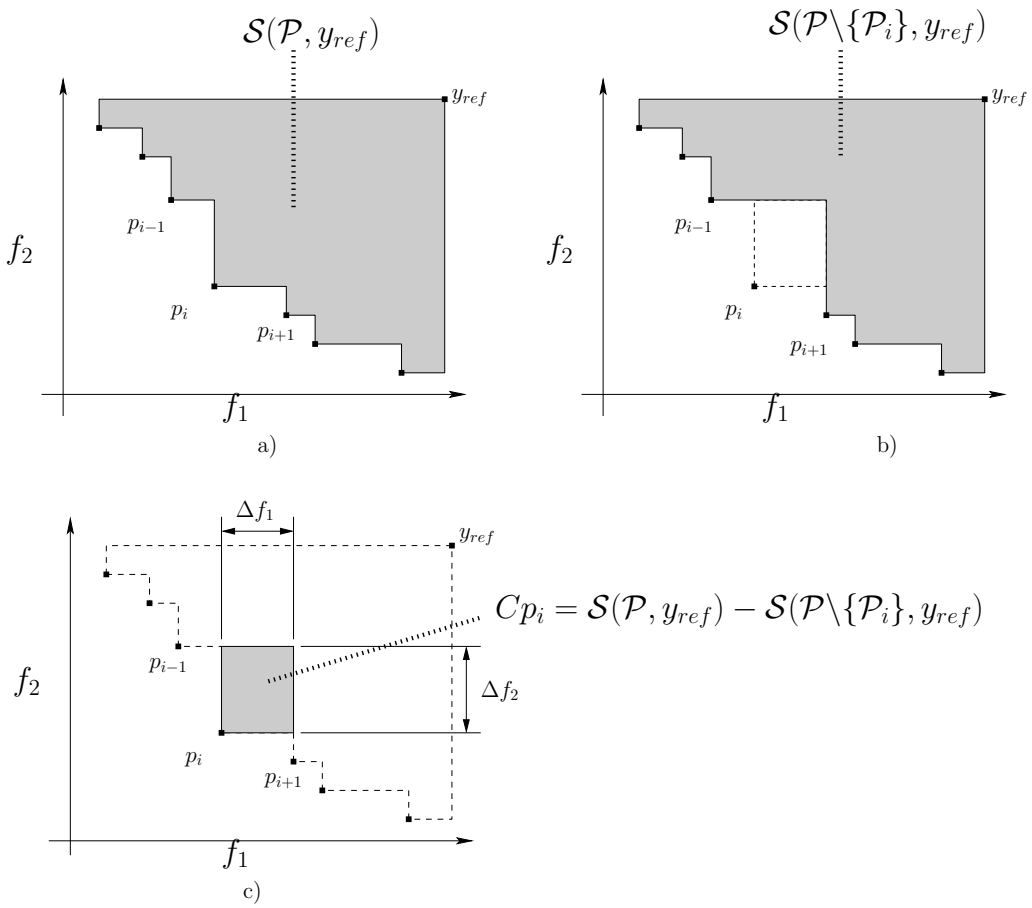


Figura 5.1: Interpretación gráfica del procedimiento del cálculo de $C_{\mathcal{P}_i}$

La ecuación (5.3) denota $\vec{v} \preceq \vec{u}$, ya que $\vec{u}$ no domina a $\vec{v}$. Sin embargo, ambos puntos deben ser no dominados entre sí, lo cual se denota como $\vec{v} \npreceq \vec{u} \wedge \vec{u} \npreceq \vec{v}$. Por esta razón, se descarta la ecuación (5.3). Si todo el conjunto de puntos son no dominados entre sí, entonces para cualquier $i, j \in k$ se cumple $p_i \npreceq p_j$, donde $p_i, p_j \in \mathcal{P}$. Además, si ordenamos el conjunto de puntos con respecto al primer objetivo, se cumple lo siguiente:

$$\text{Si } p_{1,1} \leq p_{2,1} \text{ se cumple } p_{1,2} \geq p_{2,2}, \text{ si } p_{2,1} \leq p_{3,1} \text{ se cumple } p_{2,2} \geq p_{3,2}, \text{ ... , y si } p_{k-1,1} \leq p_{k,1} \text{ se cumple } p_{k-1,2} \geq p_{k,2}.$$

Esto implica:

$$p_{1,2} \geq p_{2,2} \geq p_{3,2} \geq \cdots \geq p_{i,2} \geq \cdots \geq p_{k,2}$$

Para determinar la contribución al hipervolumen en dos dimensiones, requerimos tener el conjunto de puntos no dominados ordenados en ambos objetivos y con la propiedad anterior nos ahorramos un ordenamiento. Para este caso en especial la contribución al hipervolumen de un punto en específico se define como:

$$C_{\mathcal{P}_i} = \Delta f_1 \Delta f_2 = (p_{i+1,1} - p_{i,1})(p_{i,2} - p_{i-1,2}) \tag{5.5}$$

La fórmula anterior garantiza que cada hilo de la Unidad de Procesamiento Gráfico (GPU) trabaje de forma independiente con distintos datos (o distintos segmentos de memoria). Por lo tanto, el i-ésimo hilo de la GPU se encarga de calcular C_{p_i}. Esto implica que un conjunto de hilos $\mathcal{H}$ determina $C_{\mathcal{P}}$. Adicionalmente, obtenemos paralelismo a nivel de datos, bajo la arquitectura SIMD. El pseudo-código se da en el algoritmo 10, el cual define la estructura del *kernel* para la GPU. Éste consta de dos partes fundamentales. Primero se realiza un ordenamiento con respecto al primer objetivo y en segunda instancia, calculamos $C_{\mathcal{P}_i}$.

Algoritmo 10: Cálculo de la contribución al hipervolumen de un conjunto $\mathcal{P}$ para dos dimensiones en una GPU

Entrada: Un conjunto de puntos no dominados $\mathcal{P}$ con $\|\mathcal{P}\| = k$, donde $\mathcal{P}_i = (p_{i,1}, p_{i,2})$ y un punto de referencia $\mathcal{R} = (r_1, r_2)$
Salida: Contribución al hipervolumen $C_{\mathcal{P}}$

```
1  Asignar a Id el identificador del hilo;
2  Asignar a Dimblock la cantidad de hilos creados en la GPU;
3  C_P ← 0;
4  if Id = 0 then
5      Agregar el punto de referencia R a P;
6  k ← k + 1;
7  Ordenar ascendentemente de forma paralela el conjunto P con respecto al
   primer objetivo;
8  i ← Id + 1;
9  while i < k do
10     C_{P_i} ← (p_{i+1,1} − p_{i,1}) * (p_{i,2} − p_{i−1,2});
11     i ← i + Dimblock;
12 return C_P;
```

El algoritmo consta de un ciclo interno, el cual considera si el número de hilos creados en la GPU es menor a la cantidad de puntos no dominados en $\mathcal{P}$. Por ejemplo, supongamos que el número total de hilos en la GPU es de 64 y requerimos calcular la contribución al hipervolumen de un conjunto de puntos no dominados $\mathcal{P}$, cuya cardinalidad es 128. Por lo tanto, ejecutamos dos iteraciones del ciclo para calcular $C_{\mathcal{P}}$. Es importante mencionar que la cantidad de hilos creados en la GPU depende de la arquitectura de la tarjeta. La comunicación que existe entre la CPU y la GPU, para este método en particular, es síncrona, ya que necesitamos enviar el conjunto de puntos no dominados a la memoria global de la GPU, antes de realizar el cálculo, para después, regresar el resultado (el conjunto $C_{\mathcal{P}}$) a la CPU.

5.1.2. Solución para más de dos funciones objetivo en conflicto

En esta sección, proponemos una nueva estrategia para determinar $\mathcal{C}_{\mathcal{P}}$ en problemas que cuenten con más de dos objetivos. Dicha estrategia está basada en omitir el cálculo de los volúmenes que no proporcionan información a la contribución al hipervolumen, permitiendo con ello no tener que realizar el cálculo del hipervolumen para todo el conjunto de puntos no dominados $\mathcal{P}$.

Además, incluimos el paralelismo a nivel de datos y el paralelismo a nivel de transacción, a fin de reducir el tiempo de ejecución de un optimizador evolutivo multi-objetivo.

Suponemos que un optimizador evolutivo multi-objetivo consta de μ individuos, y que en cada generación calcula la contribución al hipervolumen en el último frente no dominado a fin de identificar al individuo menos apto (ver sección 2.5.2). Por lo tanto, para el peor de los casos, tenemos un conjunto de puntos no dominados $\mathcal{P}$, cuya cardinalidad es μ. En este caso:

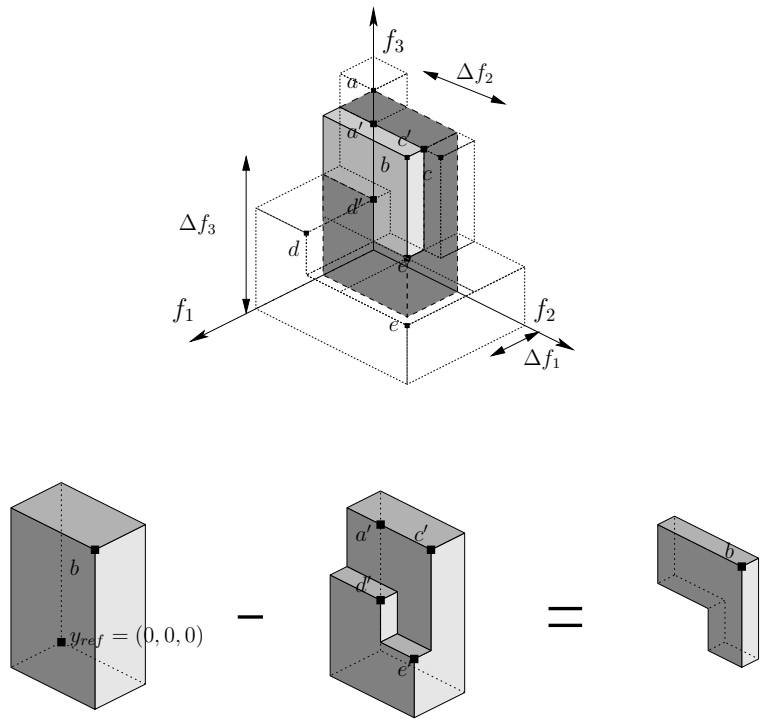
$$\forall \mathcal{P}_i \in \mathcal{P}, \text{ determinamos } \mathcal{C}_{\mathcal{P}_i} = \mathcal{S}(\mathcal{P}, y_{ref}) - \mathcal{S}(\mathcal{P} \backslash \{\mathcal{P}_i\}, y_{ref})$$

Lo anterior implica que ejecutemos $\mu + 1$ veces el cálculo de la métrica $\mathcal{S}$ en una generación. Por lo tanto, si utilizamos el mejor algoritmo existente (ver capítulo 4.4.5), esto nos tomaría $\mathcal{O}\left((\mu \log \mu + \mu^{d/2} + \mu((\mu-1)\log(\mu-1) + (\mu-1)^{d/2})\right)$, donde d es la dimensión del conjunto $\mathcal{P}$.

La principal característica de nuestra propuesta es eliminar los volumenes que no aportan información a la contribución al hipervolumen para un punto especifico. En este caso, podemos determinar $\mathcal{C}_{\mathcal{P}_i}$ como:

$$\prod_{k=0}^{n} (|\mathcal{P}_i[k] - y_{ref}[k]|) - Vol\left(\bigcup_{y \in \mathcal{P}'} \{y' | y \prec y' \prec y_{ref}\}\right) \tag{5.6}$$

En este caso, $\mathcal{P}'$ es un conjunto de puntos, los cuales están delimitados por el punto $\mathcal{P}_i$. En otras palabras, el minuendo (o productorio) define el $Vol(C)$ (el volumen de la caja delimitada por $\mathcal{P}_i$ y y_{ref}), mientras que, el sustraendo define el $Vol_C(\mathcal{P}')$ (el volumen de $\mathcal{P}'$ dentro de la caja C). En la figura 5.2 se muestra un ejemplo, donde el conjunto de puntos no dominados $\mathcal{P}$ está dado por $\{a, b, c, d, e\}$, y $\mathcal{P}' = \{a', c', d', e'\}$. Por lo tanto, el conjunto $\mathcal{P}'$ contiene los puntos delimitados por el punto b, esto implica $Vol(\mathcal{P}') \leq Vol(\mathcal{P})$, ya que $\mathcal{P}' \subset \mathcal{P}$. En pocas palabras, recortamos el volumen generado por $\mathcal{P}$ y y_{ref}. Esto nos lleva a que, determinemos μ veces el valor del $Vol(\mathcal{P}')$, el cual está delimitado por el punto de referencia y_{ref} y el punto no dominado $\mathcal{P}_i \in \mathcal{P}$ del cual se desea calcular su contribución al hipervolumen. Con este enfoque logramos reducir el cálculo, ya que en lugar de calcular $\mu + 1$ veces la métrica S del conjunto $\mathcal{P}$ para determinar $\mathcal{C}_{\mathcal{P}}$, se requiere calcular μ veces el volumen generado por $\mathcal{P}'$(este volumen está delimitado por la caja C).



$$\prod_{k=0}^{3} (b[k] - y_{ref}[k]) - vol\,(\mathcal{P}', y_{ref}) = C_b$$

Figura 5.2: Interpretación gráfica para determinar $\mathcal{C}_{\mathcal{P}_i}$ mediante la caja C_i

El conjunto $\mathcal{P}'$ creado por un punto no dominado de $\mathcal{P}$ puede contener tanto puntos dominados como no dominados entre sí. Por esta razón, necesitamos eliminar los puntos que se encuentren cubiertos completamente por algún otro punto de $\mathcal{P}'$, para después, determinar el $Vol(\mathcal{P}')$. Para determinar $\mathcal{C}_{\mathcal{P}}$ se integra el uso de hilos en una GPU, donde un conjunto de hilos $\mathcal{H}$ crea el conjunto $\mathcal{P}'$, y a su vez, realiza el filtrado de los puntos dominados del conjunto $\mathcal{P}'$. Una vez hecho esto, procedemos a calcular el $Vol(\mathcal{P}')$ y el $Vol(C)$. El pseudocódigo se proporciona en el algoritmo 11. Este algoritmo contiene ambos enfoques, paralelismo a nivel de transacción y paralelismo a nivel de datos.

En primera instancia, se crea un conjunto de *streams* dentro de la CPU para realizar las operaciones de manera concurrente con la GPU. Después, se envía el conjunto $\mathcal{P}$ y el punto de referencia y_{ref} a la GPU. El conjunto $\mathcal{P}$, es común para generar cualquier caja que describa al conjunto $\mathcal{P}'$, por lo cual, se requiere enviar a la GPU una sola vez. En cada *stream* se lanza la ejecución del *kernel* en la GPU, la recepción del conjunto $\mathcal{P}'$ de la GPU a la CPU y el cálculo de $\mathcal{C}_{\mathcal{P}_i}$, todo de forma paralela. Es de suma importancia mencionar que la cantidad de *streams* depende de la arquitectura de la GPU (número de SMs disponibles en la tarjeta). La política de

asignación de tareas (asignar cada $\mathcal{C}_{\mathcal{P}_i}$ a cada *stream*) que se lleva a cabo durante el proceso de este algoritmo, es de forma dinámica. Es por ello que, el *overhead* generado en este algoritmo es mínimo. En la figura 5.3 se muestra un ejemplo de la política de asignación, donde la asignación inicial se realiza con respecto al identificador del *stream*, para después, asignar la tarea al primer stream que se encuentre disponible. Esto se repite hasta tener cubierto todos los puntos del conjunto $\mathcal{P}$. Por lo tanto, la variable `IdSig` es utilizada como mecanismo de control, para la asignación de las tareas a cada *stream*. La comunicación que hay entre la CPU y la GPU es de forma asíncrona. Por ejemplo, mientras el *stream* 2 determina $\mathcal{C}_{\mathcal{P}_2}$, el *stream* 1 lanza el *kernel* del *stream* 1 y el *stream* 0 descarga los datos de la GPU. El pseudocódigo de la implementación del *kernel*, se muestra en el algoritmo 12. Este algoritmo crea el conjunto de puntos $\mathcal{P}'$, eliminando cada punto dominado, y a su vez, realiza un ordenamiento paralelo del conjunto $\mathcal{P}'$ con respecto al primer objetivo. El número de hilos a ejecutar se especifica en el argumento de invocación del *kernel* dentro de la CPU.

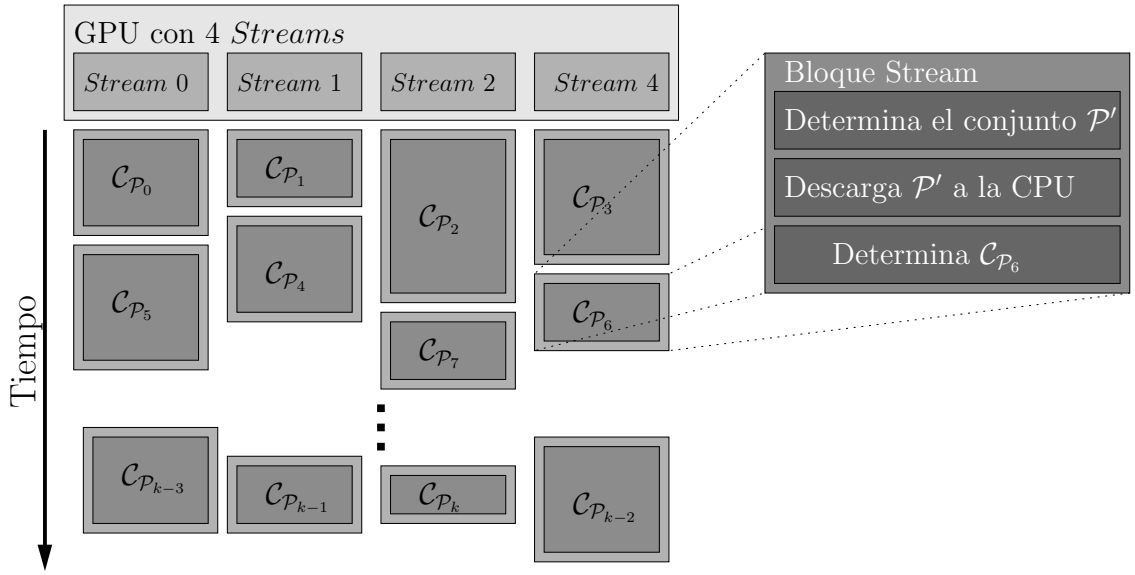


Figura 5.3: Interpretación gráfica de la asignación de tareas a cada *Stream*

Algoritmo 11: Pseudocódigo CH_RC_GPU

Entrada: Un conjunto de puntos no dominados $\mathcal{P} \subseteq \mathbb{R}^d$ con $\|\mathcal{P}\| = k$, donde $\mathcal{P}_i = (p_{i,1}, \cdots, p_{i,k})$ y un punto de referencia $\mathcal{R} = (r_1, \cdots, r_k)$

Salida: Contribución al hipervolumen $\mathcal{C}_\mathcal{P}$

```
1  Crear s streams para el manejo asíncrono de los datos;
2  Asignar a cada stream de la GPU un identificador único;
3  C_P ← 0;
4  if IdStream = 0 then
5  |  Enviar el conjunto P a la GPU;
6  |  Enviar el punto de referencia R a la GPU ;
7  Esperar hasta que todos los datos se terminen de enviar a la GPU;
8  IdSig ← 0;        // IdSig es un recurso compartido para cada stream
   // Inicio del paralelismos a nivel de transacción
9  for cada stream de la GPU de manera concurrente do
10 |  while IdSig < k do
11 |  |  i ← IdSig;
12 |  |  if i < NumStreams then
13 |  |  |  i ← IdStream;
   |  |  // Inicio del paralelismo a nivel de datos
14 |  |  Ejecutar el kernel RecortaCaja <<<
   |  |  NumSMs/NumStreams, TamBloque >>> (P, i, k, d) ; // Se crea y
   |  |  se filtra el conjunto P'
15 |  |  Copiar el conjunto P' de la GPU a la CPU;
16 |  |  Establecer la caja C definida por P_i y R;
17 |  |  Calcular el Vol(C);
18 |  |  Calcular el Vol(P');
19 |  |  C_{P_i} ← Vol(C) − Vol(P');
   |  // Esperamos a que el recurso compartido esté disponible
20 |  while no se incremente la variable IdStream do
21 |  |  if El recurso IdSig está disponible then
22 |  |  |  IdSig ← IdSig + 1
23 return C_P;
```

Algoritmo 12: Pseudocódigo del *kernel* RecortaCaja

Entrada: Un conjunto de puntos no dominados $\mathcal{P} \subseteq \mathbb{R}^d$ con $\|\mathcal{P}\| = k$, donde $\mathcal{P}_i = (p_{i,1}, \cdots, p_{i,k})$ e índice actual I para el punto $\mathcal{P}_i$

Salida: Conjunto $\mathcal{P}'$

```
1  Asignar a Id el identificador del hilo;
2  Asignar a Dimblock la cantidad de hilos creados en la GPU;
3  tam ← d * k;
4  i ← I;
   // Limita los puntos a la Caja C_I
5  while i < tam do
6      l ← i/d;
7      m ← i %d;
8      if P[I][m] > P[l][m] then
9          Asignar a P'[l][m] el punto P[l][m] que se encuentre en la caja;
10     i ← i + Dimblock;
   // Remover los puntos que se encuentren cubiertos por otro punto
      del conjunto P'
11 i ← Id;
12 for i < k do
13     if ∃P'_j ∈ P' | P'_j ⊄ P'_i then
14         Remover el punto P'_i de P';
15     i ← i + Dimblock;
16 Ordenar ascendentemente de forma paralela el conjunto P' con respecto al
   primer objetivo;
17 return P';
```

5.1.3. Ordenamiento paralelo

El algoritmo de ordenamiento paralelo utilizado en nuestra implementación de nuestro algoritmo es el ordenamiento bitónico, el cual fue propuesto por Batcher en 1968 [74]. Batcher observó que, para alcanzar un alto rendimiento en las computadoras, se requería realizar varias operaciones de forma simultánea. También, se percato que, el problema principal en el diseño de sistemas de cómputo, es la forma de conectar las distintas partes del sistema (es decir, dispositivos de entrada/salida, memorias, unidades de procesamiento, etc.), de tal manera que la trasferencia de los datos entre cada componente tenga un orden predeterminado. Es por ello que surge el ordenamiento bitónico. El objetivo de este algoritmo es alcanzar una secuencia bitónica, la cual está definida de la siguiente manera:

$$a_0 \leq \cdots \leq a_k \geq \cdots \geq a_{n-1}, \text{ donde } 0 \leq k \leq n \in \mathbb{N} \tag{5.7}$$

En la figura 5.4, se muestra un ejemplo del ordenamiento bitónico, donde en primera instancia se crea la secuencia bitónica para después realizar el ordenamiento

de la secuencia. Por ejemplo, x y y forman la secuencia bitónica a, al igual que x' y y' forman a'. Por lo tanto, las secuencias bitónicas son ordenadas y así forman la secuencia bitónica e, la cual está formada por la secuencia d y d'. Por está razón cada secuencia bitónica creada se mezcla para tener una secuencia ordenada. Esto se realiza hasta tener una secuencia total completamente ordenada.

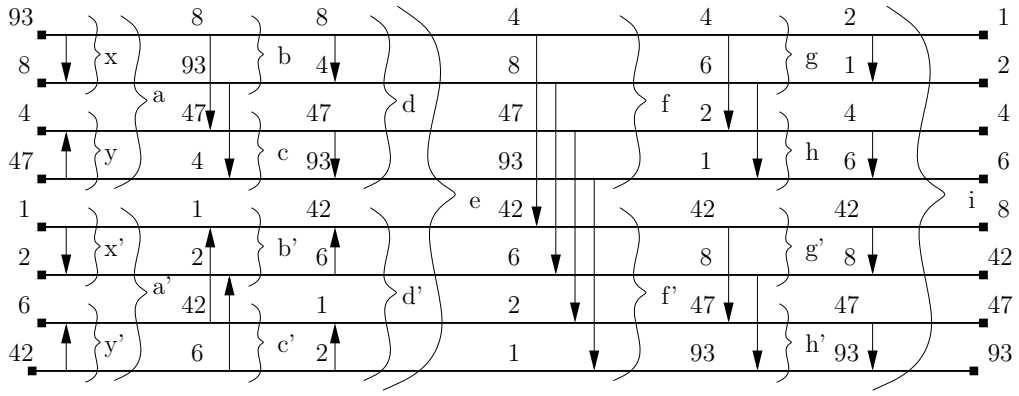


Figura 5.4: Ejemplo del ordenamiento bitónico

La complejidad de este algoritmo es $\mathcal{O}(n\log(n)^2)$, donde n es el tamaño de la secuencia. A pesar de que este algoritmo fue propuesto para ordenar memorias, redes de conmutación y estructuras de direccionamiento del microprocesador, también es utilizado en la computación paralela como método de ordenamiento.

En resumen, nuestra propuesta tiene otro enfoque para calcular la contribución al hipervolumen, la cual tiene como objetivo calcular volúmenes que realmente son necesarios para $\mathcal{C}_{\mathcal{P}_i}$, permitiendo descartar volumenes que no aportan información a $\mathcal{C}_{\mathcal{P}_i}$. Adicionalmente, integramos a nuestra propuesta un enfoque paralelo, de tal forma que logramos dos tipos de paralelismo, paralelismo a nivel de datos y paralelismo a nivel de transacción. Todo esto se logró sin la necesidad de penalizar las propiedades matemáticas que el hipervolumen ofrece para los optimizadores evolutivos multi-objetivo. Es por ello que logramos explotar todo el poder de cómputo disponible para obtener un beneficio en el tiempo de ejecución para los optimizadores evolutivos multi-objetivo.

Capítulo 6

Experimentos

En este capítulo se presentan los resultados de los estudios experimentales que se realizaron sobre nuestra propuesta. Para validar el desempeño del algoritmo propuesto, se adoptó el algoritmo evolutivo SMS-EMOA, el cual utiliza el indicador hipervolumen como criterio de selección. También se utilizaron problemas multi-objetivo de dos distintos conjuntos de problemas de prueba DTLZ [75] y WFG [76]. La razón por la que se adoptaron estos problemas multi-objetivo es que son escalables conforme al número de objetivos. Además, cada problema de prueba incluye distintas características del frente de Pareto, lo cual es deseable para observar el desempeño de nuestra propuesta ante diferentes características. Todos los experimentos fueron ejecutados en la misma computadora, la cual cuenta con las siguientes características:

- Procesador.- Intel(R) Core(TM) i7-3930k CPU @ 3.20 GHz.
- 8G de memoria RAM.
- Una Unidad de Procesamiento Gráfico .- Geforce GTX 680.
- Sistema Operativo.- Fedora 18 de 64 bits.

Se comparó el desempeño de la versión original de SMS-EMOA con el algoritmo FPL para el cálculo de la contribución al hipervolumen (ver capítulo 4) y el SMS-EMOA con nuestro algoritmo propuesto para el cálculo de la contribución al hipervolumen mediante el uso de la GPU, la cual la llamamos CH_RC_GPU (contribución al hipervolumen mediante el recorte de caja en una GPU). Por lo que, nuestra propuesta está implementada bajo en el modelo de programación CUDA para C.

6.1. Problemas de Prueba

Se tomaron cuatro problemas del conjunto de prueba DTLZ, los cuales son: DTLZ1, DTLZ2, DTLZ3 y DTLZ4. Las principales características de estos problemas de prueba se describen a continuación.

- DTLZ1 .- Tiene un frente de Pareto lineal, separable y multimodal.
- DTLZ2 .- Tiene un frente de Pareto cóncavo y es unimodal.
- DTLZ3 .- Tiene un frente de Pareto cóncavo y es multimodal con $(3^k - 1)$ frentes de Pareto locales y un frente de Pareto global.
- DTLZ4 .- Tiene un frente de Pareto cóncavo, separable y unimodal.

En el conjunto de problemas DTLZ, el número total de variables está dado por $n = m + k - 1$, donde m es el número de objetivos y k está definido de la siguiente forma: 5 para DTLZ1 y 10 para DTLZ2-4.

Además, se consideraron los tres primeros problemas de prueba (WFG1-WFG3) del conjunto de prueba WFG. Estos problemas agregan características como el sesgo, la multimodalidad y la no separabilidad mediante un conjunto de transformaciones. Las principales características de estos problemas de prueba son las siguientes:

- WFG1 .- Este problema es separable y unimodal.
- WFG2 .- Este problema es no separable y multimodal. Su frente de Pareto está desconectado.
- WFG3 .- Este problema es no separable pero unimodal y su frente de Pareto es lineal.

En el conjunto de problemas WFG, el número total de variables está dado por $n = k + l$, con $k = 4(m - 1)$ y $l = 2$, donde m es el número de objetivos, k describe la posición de los parámetros subyacentes del vector de variables y l describe la distancia de los parámetros subyacentes del vector de variables. En el apéndice A se muestra una descripción más detallada de cada uno de los problemas de prueba utilizados.

6.2. Metodología

El principal objetivo de este trabajo es evaluar el comportamiento y el tiempo de ejecución de un AEMO usando nuestra propuesta como criterio de selección al momento de resolver problemas de optimización multi-objetivo con muchos objetivos. Para ello se decidió analizar la tasa de convergencia que se tuvo en el AEMO (SMS-EMOA) utilizando nuestra propuesta contra el mismo AEMO usando el algoritmo FPL.

6.2.1. Tasa de convergencia

Para calcular dicha tasa de convergencia, se adoptó el indicador de hipervolumen. Como se mencionó en el capítulo 4, el hipervolumen es la unión de todos los hipercubos encontrados, los cuales están delimitados por un punto de referencia. Usamos

Problema	**No. objetivos**	**Tamaño de población**	**Generaciones**	**No. de evaluaciones de función**
DTLZ1	2 a 4 y 8	100	250	25000
	5 a 7	95	263	24985
	9	90	277	24930
DTLZ2	2 a 8	100	150	15000
	9	90	166	14940
DTLZ3, WFG1 y WFG3.	2 a 6	100	750	75000
	7 y 8	95	789	74955
	9	90	830	74700
DTLZ4 y WFG2	2 a 7	100	500	50000
	8	110	454	49940
	9	90	555	49950

Tabla 6.1: Parámetros

Problema	**Punto de referencia**
DTLZ1	$(1, 1, 1, \ldots, 1)$
DTLZ2-4	$(2, 2, 2, \ldots, 2)$
WFG1-3	$(3, 5, 7, \ldots, 2m + 1)$

Tabla 6.2: Puntos de referencia

el algoritmo propuesto por Fonseca, Paquete y Luis Ibañez (ver sección 4.4.4) para calcular la tasa de convergencia del AEMO. Se decidió establecer una condición de paro para el AEMO, la cual fue un número máximo de evaluaciones de las funciones objetivo tal y como se definen en la tabla 6.1. Debido a que era posible que el AEMO no pudiera terminar la ejecución en un tiempo considerable, decidimos usar otra condición de paro alternativa: el tiempo máximo de una ejecución es 8000 min. Los puntos de referencia utilizados para el cálculo del hipervolumen para cada problema se presentan en la tabla 6.2.

6.2.2. Espaciamiento

Es uno de los indicadores más populares para observar la diversidad entre las soluciones no dominadas generadas por un AEMO. Esta métrica fue propuesta por Schott [77], y calcula la distancia relativa entre las soluciones consecutivas. Por lo tanto se define de la siguiente forma:

$$S = \sqrt{\frac{1}{|Q|}\sum_{i=1}^{|Q|}(d_i - \overline{d})^2} \tag{6.1}$$

donde $d_i = \text{mín}_{k \in Q \bigwedge k \neq i} \sum_{j=1}^{m} |f_j^i - f_j^k|$ y $\overline{d}$ es el valor promedio de todas las distancias y se define como: $\overline{d} = \sum_{i=1}^{|Q|} \frac{d_i}{|Q|}$. Cuando las soluciones se encuentran esparcidas uniformemente entre sí, la distancia tendrá un valor pequeño. Por lo tanto, el algoritmo

que encuentre una aproximación al conjunto de óptimos de Pareto teniendo un menor valor de espaciado es mejor. El mejor valor posible para este indicador es cero.

6.2.3. Aceleración

Una de las métricas más importantes del cómputo paralelo es medir la rapidez de nuestro algoritmo paralelo con respecto a la versión secuencial del algoritmo. Esta métrica es conocida como Aceleración. Para un problema de tamaño n, la aceleración se determina de la siguiente forma:

$$S_p = \frac{T_s(n,1)}{T(n,p)} \tag{6.2}$$

Donde $T_s(n,1)$ es el tiempo obtenido del algoritmo secuencial y $T(n,p)$ es el tiempo del algoritmo paralelo con p procesadores, ambos resolviendo el mismo problema.

6.2.4. Eficiencia

La eficiencia de un algoritmo se denota por E_p usando p procesadores. Alcanzar la máxima eficiencia en una implementación es muy difícil por lo que esta medida nos ayuda a observar el comportamiento de nuestro algoritmo paralelo. La eficiencia se define de la siguiente forma:

$$E_p = \frac{S_p}{p} = \frac{T_s(n,1)}{pT(n,p)} \tag{6.3}$$

6.3. Parametrización

Los parámetros utilizados en la ejecución del AEMO fueron catalogados con la finalidad de tener una buena aproximación al conjunto de óptimos de Pareto. Por lo tanto, los índices de distribución para los operadores SBX y *polynomial-based mutation* [30], fueron establecidos como $\eta_c = 20$ y $\eta_m = 20$, respectivamente. La probabilidad de cruza es $p_c = 0.9$ y la probabilidad de mutación $p_m = 1/n$, donde n es el número de variables de decisión en el POM. El tamaño de población y el número de generaciones del AEMO se muestran en la tabla 6.1. La cantidad de hilos creados en la GPU para ejecutar el AEMO que utiliza nuestra propuesta son los siguientes: para dos objetivos de un POM se usaron 1024 hilos con 1 *Stream* y para más de dos objetivos de un POM se utilizaron 1024 hilos con 7 *Streams*.

6.4. Resultados

En nuestros resultados, se reporta la media del tiempo que le tomó a cada algoritmo (SMS-EMOA y SMS-EMOA-CH-RC-GPU) resolver cada problema, medido sobre 25 ejecuciones independientes con el objetivo de observar la aceleración y la eficiencia

que se obtuvó con respecto al algoritmo secuencial. También se reporta la media de los valores de los indicadores de desempeño de un conjunto de 25 ejecuciones independientes de cada algoritmo. En algunos casos, la ejecución del algoritmo secuencial excede el tiempo establecido como criterio de paro. En estos casos, evidentemente no se pudieron determinar los valores de los indicadores. En las figuras 6.2, 6.4, 6.6, 6.8, 6.10 y 6.12 se muestran gráficas comparativas del tiempo de ejecución de cada algoritmo, las cuales están en una escala logarítmica con respecto al tiempo. Cabe mencionar que, los tiempos obtenidos por nuestro algoritmo propuesto para resolver el caso base (para dos objetivos de un POM) no se incluye en las gráficas comparativas de tiempo, ya que el algoritmo usado para dos objetivos en la GPU utiliza una cantidad diferente de hilos con respecto a la otra versión (para mas de dos objetivos), por lo que esta comparación no tendría sentido. En las figuras 6.1, 6.3, 6.5, 6.7, 6.9 y 6.11 se muestran las aproximaciones finales del AEMO de cada problema de prueba para tres funciones objetivo, las cuales fueron obtenidas de la mediana de la tasa de convergencia de las 25 ejecuciones independientes.

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	**0.873238**	**0.001283**	0.873162	0.001457
3	**0.973716**	**0.007665**	0.97325	0.009791
4	0.992412	0.017341	**0.994162**	**0.016266**
5	**0.998515**	**0.026013**	0.99851	0.02788
6	0.999553	0.038846	**0.99958**	**0.034769**
7	0.999841	0.050285	**0.999876**	**0.041381**
8	–	–	**0.99993**	**0.059126**
9	–	–	**0.99997**	**0.056958**

Tabla 6.3: Resultados correspondientes de los indicadores de desempeño para el problema DTLZ1

El problema DTLZ1 es lineal, pero multimodal por lo cual a los AEMOs les resulta difícil alcanzar una buena aproximación al conjunto de óptimos de Pareto. Sin embargo, como se observa en la tabla 6.3, ambas propuestas obtienen una buena aproximación del frente de Pareto con una buena distribución de ellas a lo largo del frente de Pareto (ver figura 6.1). La diferencia entre ambas propuestas es mínima. Nuestra propuesta ayuda al AEMO a obtener una buena aproximación al conjunto de óptimos de Pareto en un menor tiempo de ejecución (ver tabla 6.4). En esa tabla podemos observar que hay un incremento cada vez mayor de la aceleración conforme se incrementa el número de objetivos del POM, alcanzándose una aceleración máxima de 486.30x. Esto se debe a la cantidad de información que se envía a la GPU, reduciendo el *overhead* que existe entre cada generación del AEMO.

A diferencia del problema DTLZ1, los problemas DTLZ2, DTLZ3 y DTLZ4 tienen un frente de Pareto cóncavo, por lo que agregan una nueva dificultad en la aproxi-

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	0.1784	**0.1354**	1.3186	0.001288
3	0.6921	**0.5578**	1.2409	0.000174
4	9.927	**0.9542**	10.404	0.001452
5	146.5268	**1.9638**	74.6154	0.01041
6	1954.4977	**9.4954**	205.8373	0.028717
7	25632.8153	**52.7094**	486.3049	0.067844
8	–	**437.9792**	–	–
9	–	**1673.3501**	–	–

Tabla 6.4: Comparación de los tiempos de ejecución para el problema DTLZ1

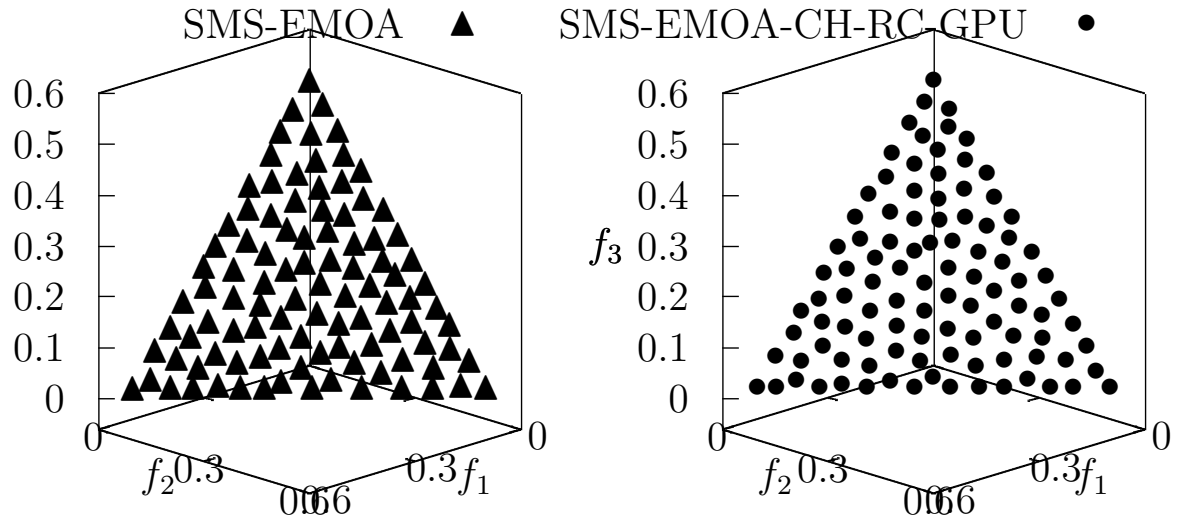


Figura 6.1: Frentes de Pareto obtenidos por ambos algoritmos para el problema DTLZ1 con tres funciones objetivo

mación al conjunto de óptimos de Pareto. En las tablas 6.5, 6.7 y 6.9 se observa que nuestra propuesta no penaliza las propiedades matemáticas que el hipervolumen ofrece para aproximar el conjunto de óptimos de Pareto. Sin embargo, nuestra propuesta ayuda a ejecutar el AEMO en un menor tiempo (ver tablas 6.6, 6.8 y 6.10). En las figuras 6.2, 6.4, 6.6 y 6.8 se muestra una comparativa del tiempo promedio de ambas implementaciones y se observa que nuestra propuesta acelera significativamente al

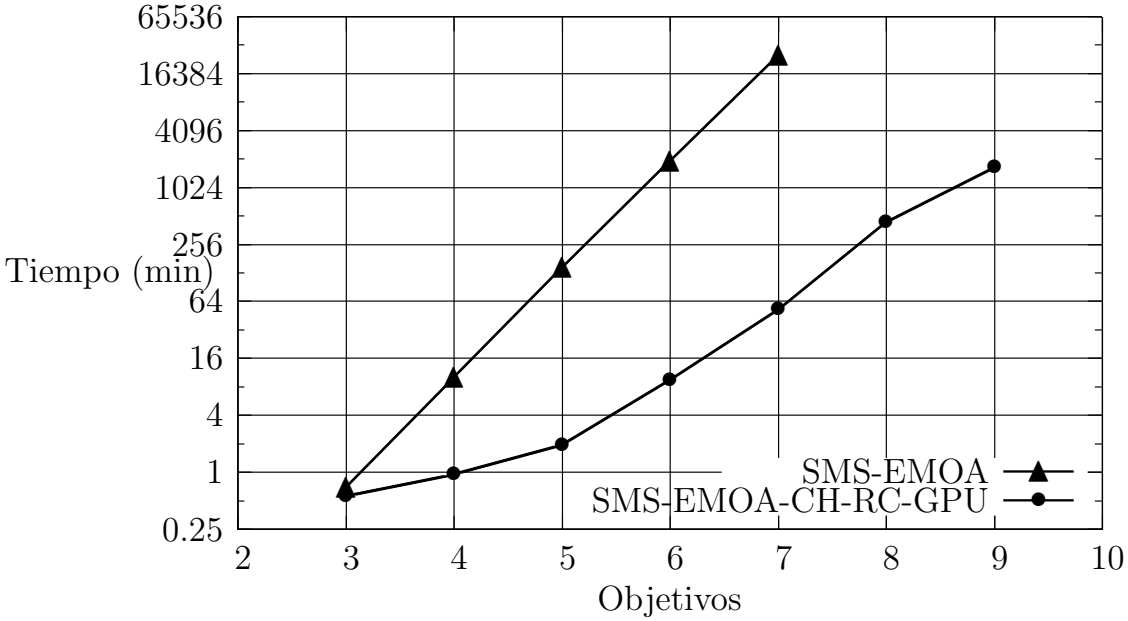


Figura 6.2: Comparación gráfica de los tiempos de ejecución para el problema DTLZ1

AEMO conforme se incrementa el número de objetivos.

Es importante resaltar que para todos los POMs con dos funciones objetivo en conflicto se obtuvó una mayor aceleración, que para tres objetivos. Esto se debe a que la solución para esta cantidad de objetivos está basada en la arquitectura SIMD con una comunicación síncrona, por lo que es una versión más eficiente para este caso en particular.

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	**3.210879**	0.007702	3.210868	**0.007602**
3	**7.426076**	0.041653	7.426069	**0.040691**
4	15.5753	0.067581	**15.575445**	**0.06661**
5	31.677782	**0.084248**	**31.678164**	0.086913
6	63.75333	**0.133544**	**63.753512**	0.135595
7	–	–	**127.79839**	**0.181875**
8	–	–	**255.837758**	**0.192845**
9	–	–	**511.846819**	**0.168345**

Tabla 6.5: Resultados correspondientes de los indicadores de desempeño para el problema DTLZ2

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	0.2784	**0.1339**	2.0806	0.002032
3	1.6382	**0.2926**	5.5995	0.000782
4	11.1038	**0.673**	16.5008	0.002302
5	168.761	**1.3908**	121.3476	0.01693
6	2420.9844	**10.4016**	232.7513	0.032471
7	–	**49.7891**	–	–
8	–	**525.0347**	–	–
9	–	**1713.133**	–	–

Tabla 6.6: Comparación de los tiempos de ejecución para el problema DTLZ2

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	3.208826	**0.007444**	**3.20903**	0.0076
3	7.420476	**0.041679**	**7.421543**	0.041999
4	**15.574479**	**0.06765**	15.57325	0.067938
5	**31.67833**	0.088963	31.678284	**0.085466**
6	63.740603	**0.146409**	**63.74412**	0.161621
7	–	–	**127.788183**	**0.223037**
8	–	–	**255.815976**	**0.192676**
9	–	–	**511.840859**	**0.180436**

Tabla 6.7: Resultados correspondientes de los indicadores de desempeño para el problema DTLZ3

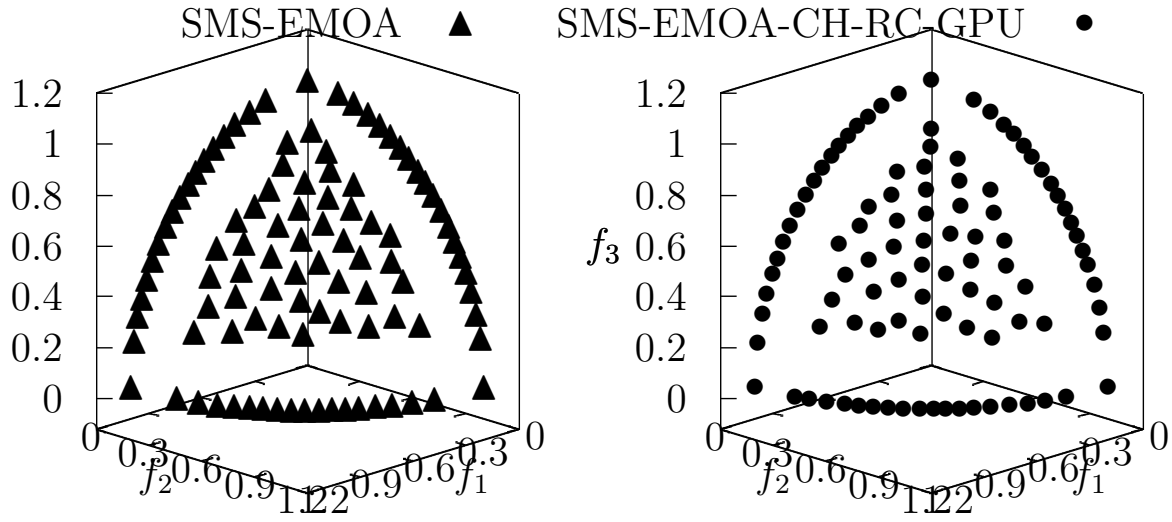


Figura 6.3: Frentes de Pareto obtenidos por ambos algoritmos para el problema DTLZ2 con tres funciones objetivo

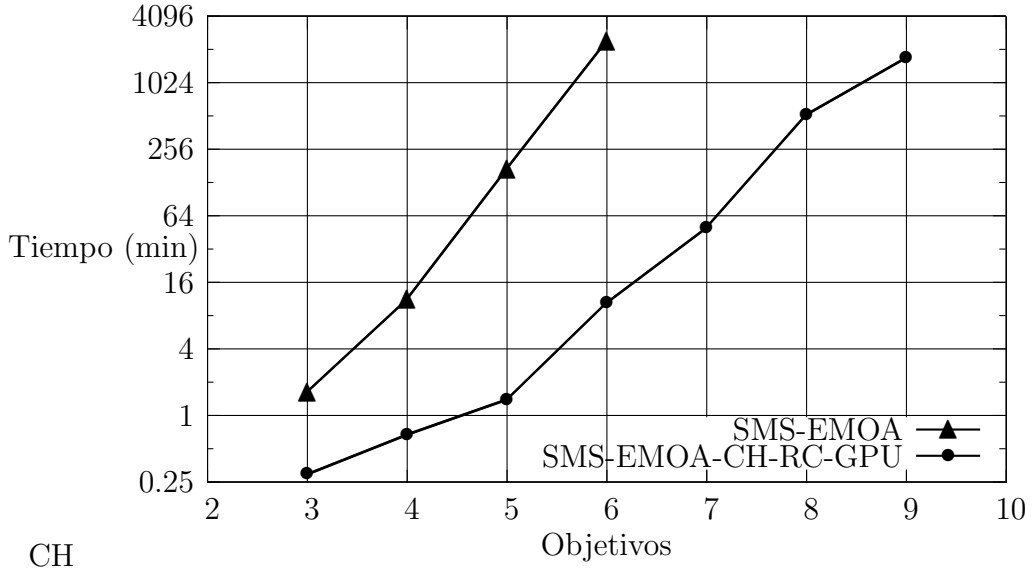


Figura 6.4: Comparación gráfica de los tiempos de ejecución para el problema DTLZ2

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	0.6189	**0.4559**	1.3577	0.001326
3	1.8316	**1.4572**	1.257	0.000176
4	27.1425	**2.1452**	12.6527	0.001766
5	434.3908	**4.2416**	102.4135	0.014288
6	6040.9012	**23.9932**	251.7766	0.035126
7	–	**133.1065**	–	–
8	–	**1021.2014**	–	–
9	–	**4437.1247**	–	–

Tabla 6.8: Comparación de los tiempos de ejecución para el problema DTLZ3

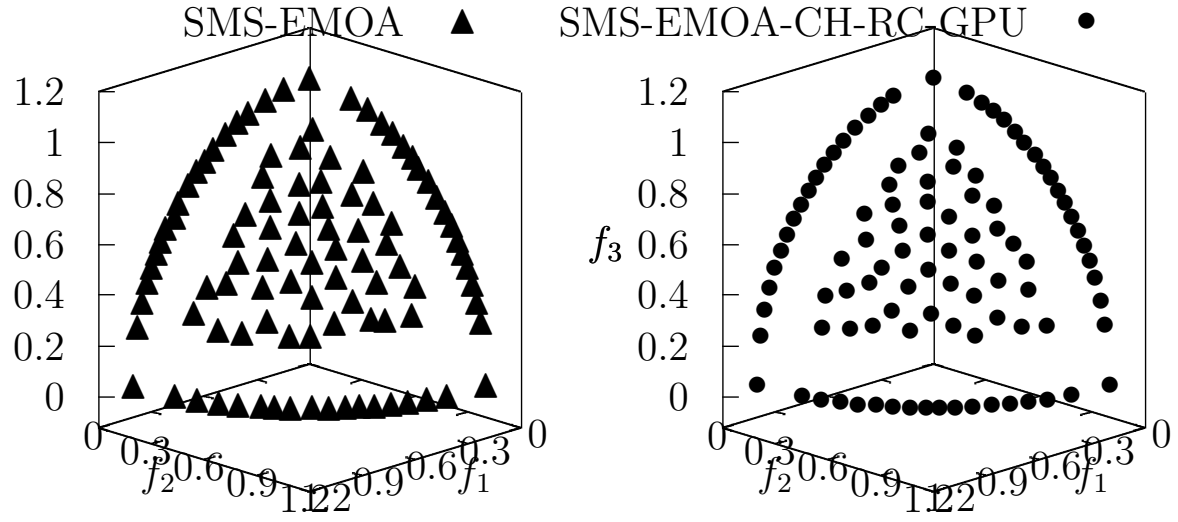


Figura 6.5: Frentes de Pareto obtenidos por ambos algoritmos para el problema DTLZ3 con tres funciones objetivo

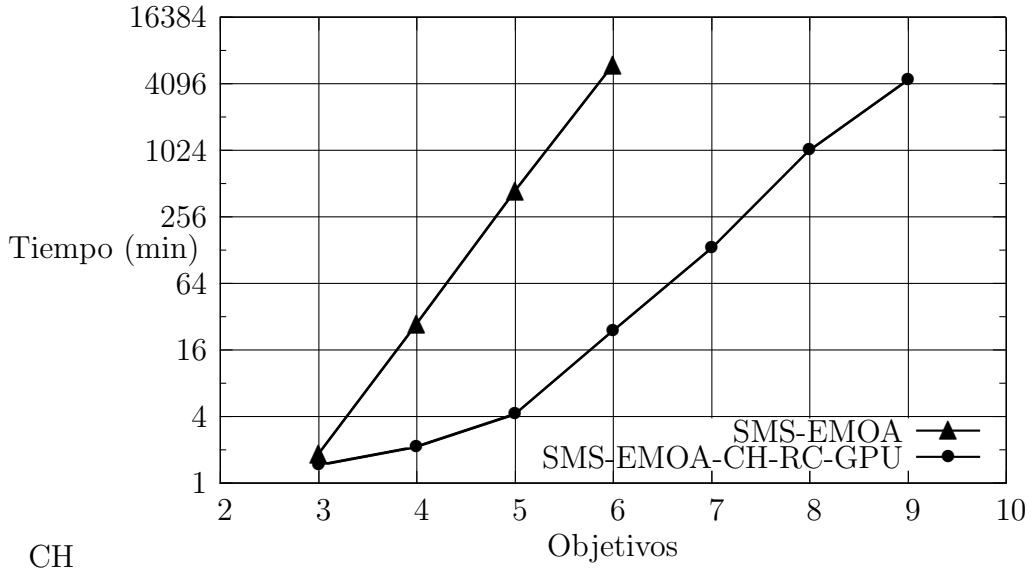


Figura 6.6: Comparación gráfica de los tiempos de ejecución para el problema DTLZ3

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	2.87192	**0.005499**	**3.065692**	0.006722
3	6.887928	0.026107	**7.145613**	0.032301
4	14.99058	**0.049317**	**15.406114**	0.057918
5	30.189868	**0.050782**	**30.787056**	0.062325
6	62.447761	**0.07819**	**63.368461**	0.098488
7	–	–	**127.726866**	**0.177763**
8	–	–	**255.731938**	**0.194006**
9	–	–	**511.694486**	**0.184917**

Tabla 6.9: Resultados correspondientes de los indicadores de desempeño para el problema DTLZ4

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	0.5385	**0.3511**	1.5339	0.001498
3	3.1228	**2.5011**	1.2486	0.000175
4	35.218	**3.4136**	10.3172	0.00144
5	520.502	**5.6853**	91.5528	0.012773
6	7446.6796	**26.3014**	283.129	0.0395
7	–	**125.5394**	–	–
8	–	**1629.4346**	–	–
9	–	**4199.4646**	–	–

Tabla 6.10: Comparación de los tiempos de ejecución para el problema DTLZ4

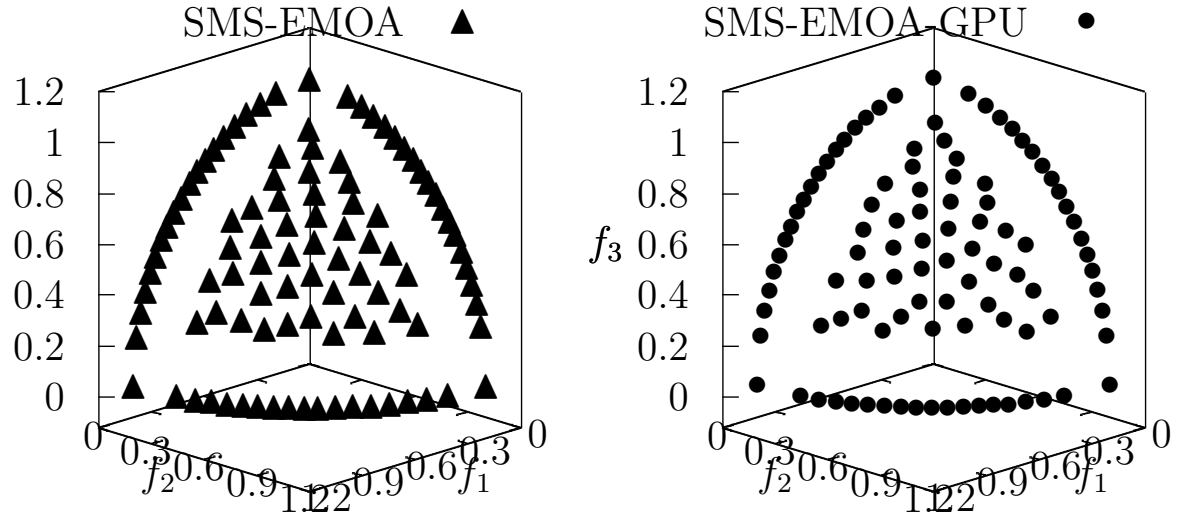


Figura 6.7: Frentes de Pareto obtenidos por ambos algoritmos para el problema DTLZ4 con tres funciones objetivo

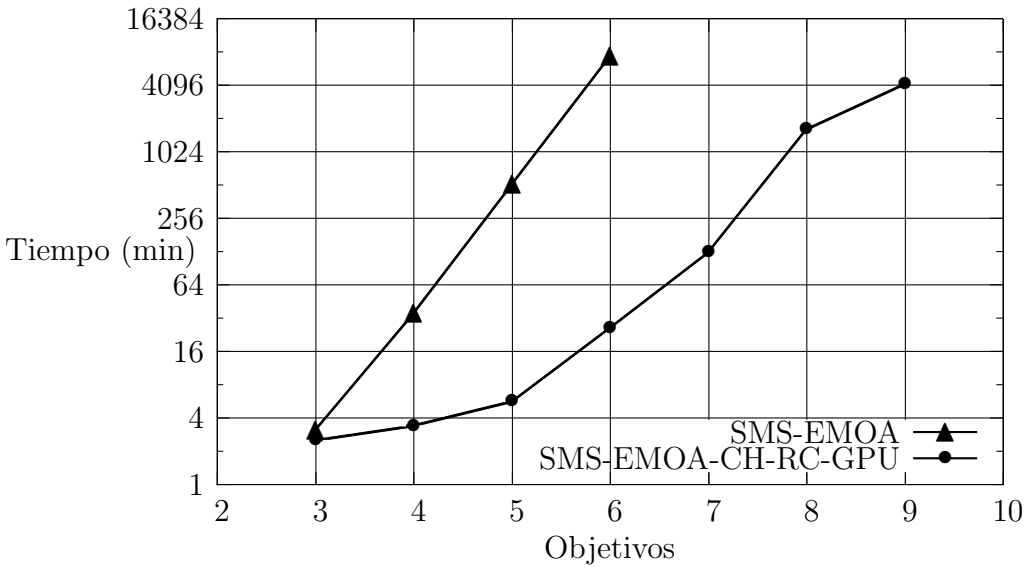


Figura 6.8: Comparación gráfica de los tiempos de ejecución para el problema DTLZ4

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	**6.735278**	**0.006432**	6.611425	0.009437
3	**66.595412**	**0.047712**	64.496605	0.049191
4	550.509191	0.077903	**635.368636**	**0.070628**
5	5811.563198	0.077009	**5981.306146**	**0.075655**
6	–	–	**69980.43159**	**0.072144**
7	–	–	**975295.6509**	**0.074721**
8	–	–	**16252223**	**0.071669**
9	–	–	**260018586.1**	**0.082601**

Tabla 6.11: Resultados correspondientes de los indicadores de desempeño para el problema WFG1

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	1.6154	**0.6304**	2.5626	0.002503
3	7.3843	**4.2319**	1.745	0.000244
4	108.9946	**5.0127**	21.7439	0.003034
5	2047.7594	**6.6857**	306.2917	0.042731
6	–	**28.7109**	–	–
7	–	**154.8892**	–	–
8	–	**992.7482**	–	–
9	–	**5850.1223**	–	–

Tabla 6.12: Comparación de los tiempos de ejecución para el problema WFG1

El conjunto de problemas WFG agregan un mayor grado de dificultad, por lo que la ejecución del AEMO requiere un mayor número de evaluaciones de función. Por esta razón, en este caso se incrementa el número de llamadas del cálculo de la contribución al hipervolumen y, consecuentemente, se incrementa el tiempo de ejecución del AEMO. Para el problema WFG1, la forma del frente de Pareto es plana y polinomial, por lo que, las aproximaciones al frente de Pareto que entrega el AEMO se quedan concentradas en una sola región. Esta difícultad afecta a ambas implementaciones (ver figura 6.9). Sin embargo, nuestra propuesta logra reducir el tiempo de ejecución del AEMO con las mismas características que el algoritmo FPL (ver tabla 6.12). En tanto, en el problema WFG2, la aproximación que realizó el AEMO con ambos algoritmos es muy buena. En general, nuestra propuesta muestra muy buenos resultados con un menor tiempo de ejecución (ver tabla 6.13 y 6.14). La aceleración obtenida fue de 883x, lo cual permitió resolver problemas cuyo costo computacional resulta prohibitivo con la versión secuencial.

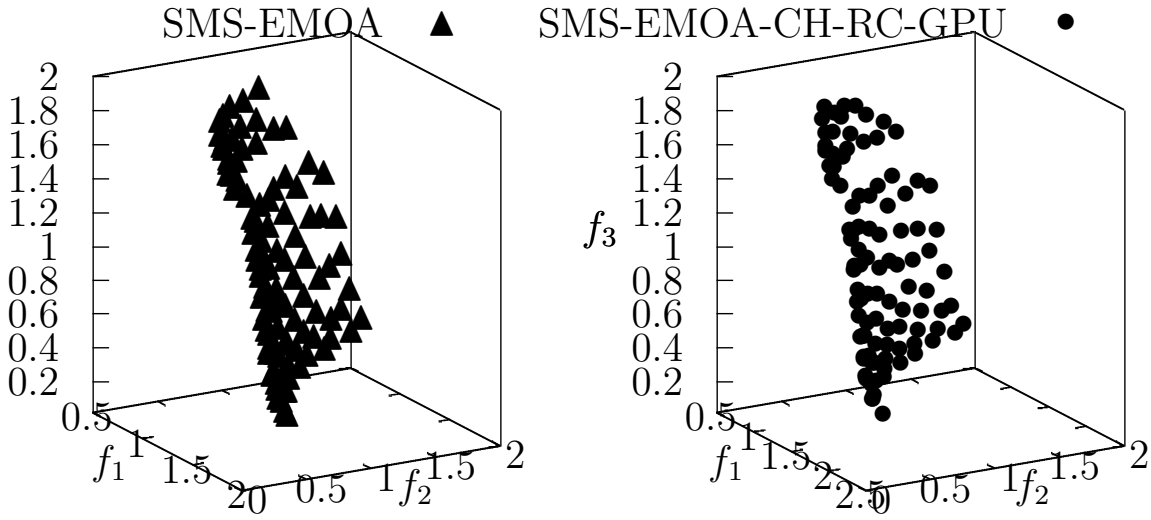


Figura 6.9: Frentes de Pareto obtenidos por ambos algoritmos para el problema WFG1 con tres funciones objetivo

El problema WFG3 tiene un frente de Pareto lineal, por lo que determinar el valor de la contribución al hipervolumen toma un menor tiempo. Por lo tanto, el algoritmo FPL se ejecuta más rápido para este problema en particular. A nuestra propuesta, le afecta que el frente de Pareto sea lineal, ya que la creación del nuevo conjunto de puntos $\mathcal{P}'$ se inhibe. En otras palabras, el recorte del $vol(\mathcal{P})$ es mínimo (ver capítulo 5). Por esta razón, no se logra reducir el tiempo de ejecución significativamente. Sin embargo, nuestra propuesta aún con esta pequeña deficiencia logra reducir el tiempo de ejecución del AEMO (ver tabla 6.16).

En los resultados mostrados anteriormente, observamos que la eficiencia de nuestro algoritmo difiere de la forma del frente de Pareto del problema multi-objetivo. La única deficiencia se observa cuando el frente de Pareto es lineal. Sin embargo, en cualquier otro caso, el algoritmo propuesto logra lidiar significativamente con el tiempo de ejecución del AEMO, obteniendo resultados que no era posible alcanzar con la versión secuencial. También, se observa que nuestra propuesta no penaliza las características que el hipervolumen ofrece para la aproximación del conjunto de óptimos de Pareto en un AEMO, por lo que obtiene una buena aproximación al frente de Pareto en un tiempo significativamente menor.

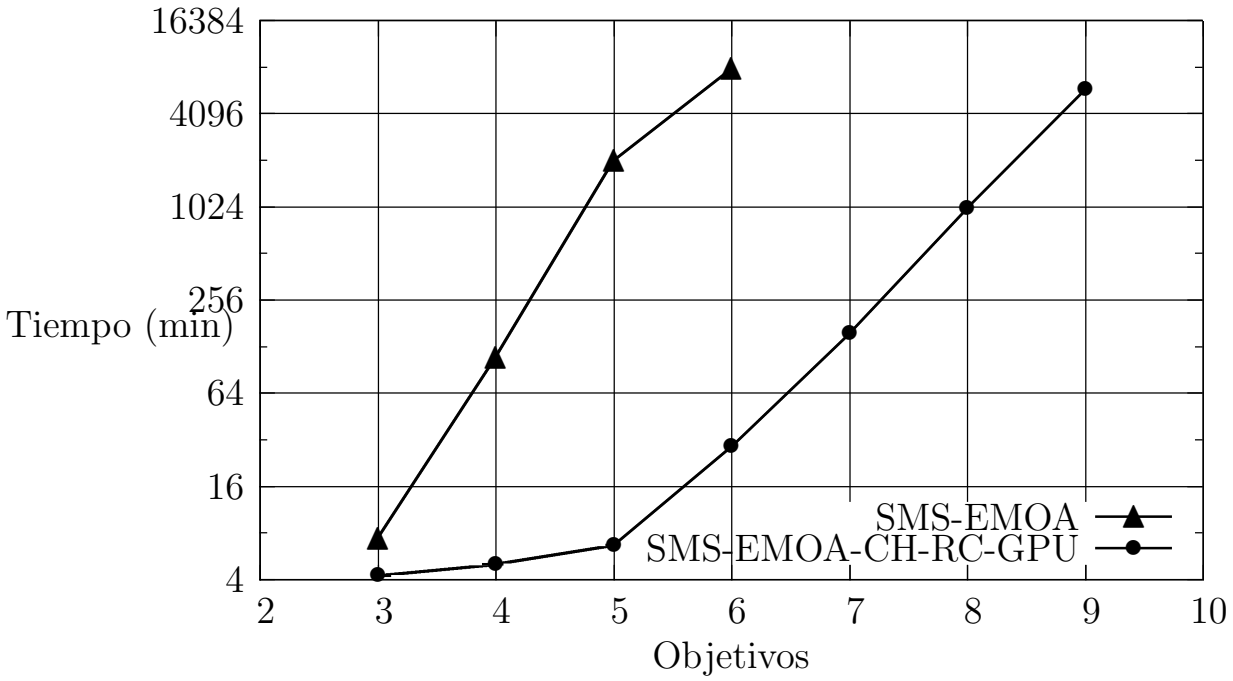


Figura 6.10: Comparación gráfica de los tiempos de ejecución para el problema WFG1

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	**10.655908**	0.008643	10.569597	**0.008441**
3	86.215292	**0.04886**	**86.827259**	0.052103
4	786.873748	**0.096958**	**793.125358**	0.101117
5	**8860.244206**	0.082606	8566.72997	**0.07059**
6	110240.8108	**0.100501**	**112357.7782**	0.108096
7	–	–	**1646787.124**	0.178076
8	–	–	**27973328.67**	0.240691

Tabla 6.13: Resultados correspondientes de los indicadores de desempeño para el problema WFG2

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	0.6955	**0.3571**	1.9476	0.001902
3	3.7612	**2.4849**	1.5137	0.000212
4	74.153	**3.3485**	22.1456	0.00309
5	1458.6645	**4.5121**	323.2813	0.045101
6	17867.2959	**20.2233**	883.5017	0.123257
7	–	**134.0804**	–	–
8	–	**1288.5644**	–	–

Tabla 6.14: Comparación de los tiempos de ejecución para el problema WFG2

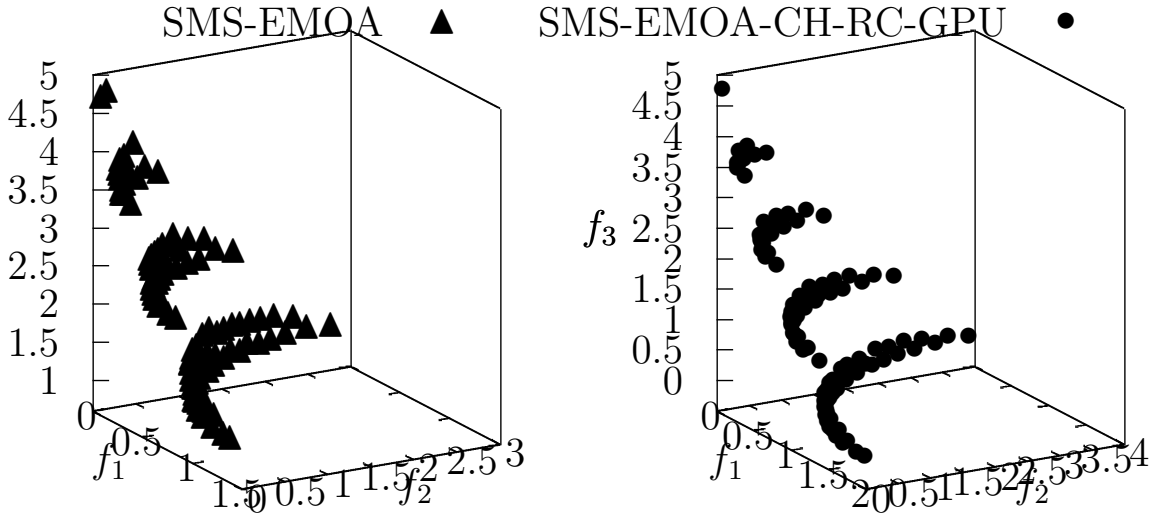


Figura 6.11: Frentes de Pareto obtenidos por ambos algoritmos para el problema WFG2 con tres funciones objetivo

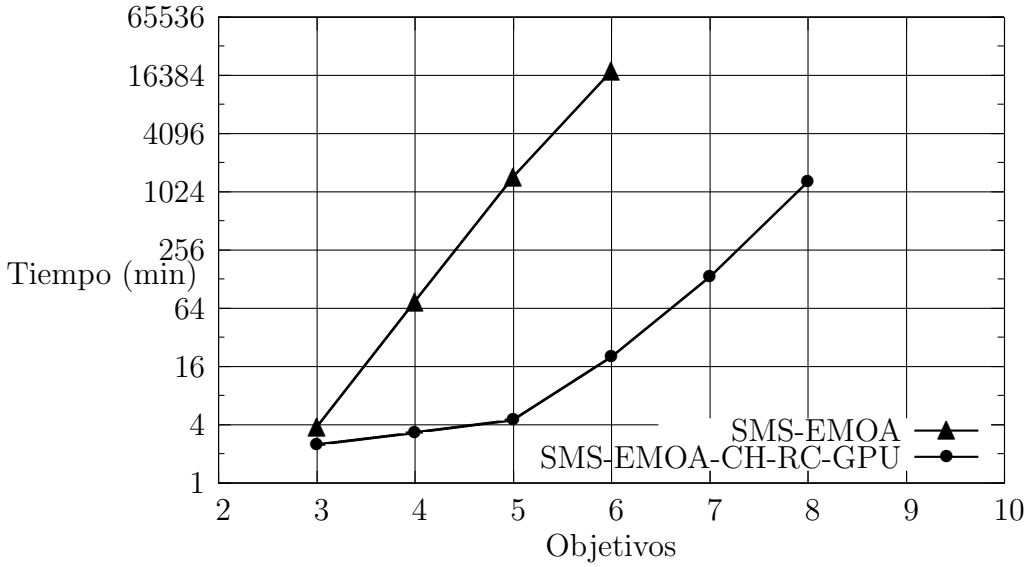


Figura 6.12: Comparación gráfica de los tiempos de ejecución para el problema WFG2

	SMS-EMOA		**SMS-EMOA-CH-RC-GPU**	
Objetivos	Hipervolumen	Espaciamiento	Hipervolumen	Espaciamiento
2	**10.954985**	0.004739	10.954594	**0.00423**
3	76.42553	0.132885	**76.426786**	**0.132637**
4	683.216698	0.442542	**683.589267**	**0.439745**
5	7474.344326	**0.704298**	**7484.96526**	0.706353
6	96821.66284	0.939095	**96950.48865**	**0.905224**
7	–	–	**1365800.643**	**1.030017**
8	–	–	**22781648.93**	**1.332752**

Tabla 6.15: Resultados correspondientes de los indicadores de desempeño para el problema WFG3

	SMS-EMOA	**SMS-EMOA-CH-RC-GPU**		
Objetivos	Tiempo (min)	Tiempo (min)	Aceleración	Eficiencia
2	1.648	**0.6439**	2.5597	0.0025
3	6.3563	**4.4548**	1.4269	0.0002
4	38.2058	**7.8809**	4.848	0.000677
5	188.9923	**35.5841**	5.3112	0.000741
6	880.7624	**212.272**	4.1493	0.000579
7	–	**779.205**	–	–
8	–	**5323.5093**	–	–

Tabla 6.16: Comparación de los tiempos de ejecución para el problema WFG3

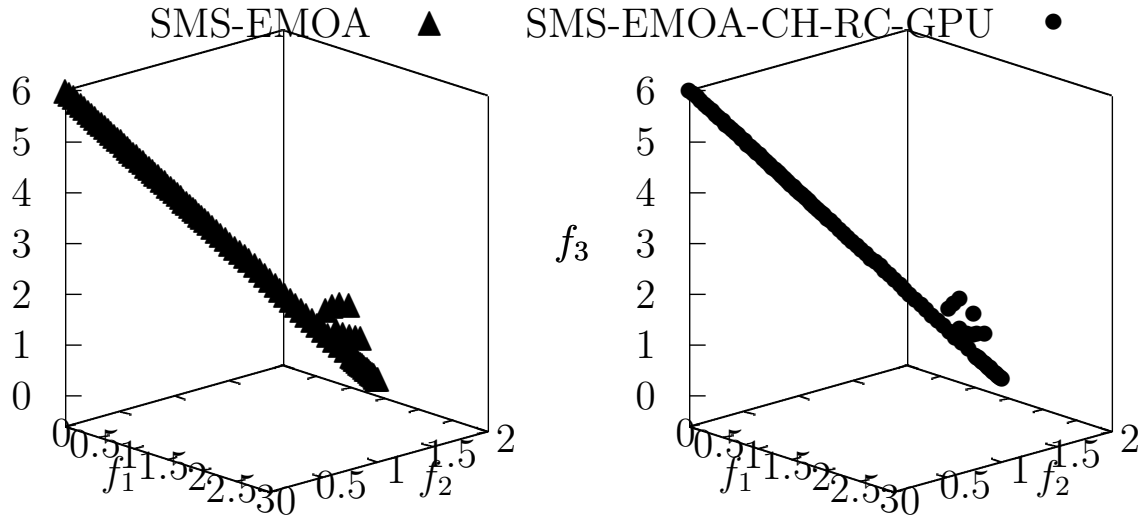


Figura 6.13: Frentes de Pareto obtenidos por ambos algoritmos para el problema WFG3 con tres funciones objetivo

Capítulo 7

Conclusiones y Trabajo futuro

En esta tesis se abordó el problema del cálculo del hipervolumen exacto para usarse como criterio de selección de un algoritmo evolutivo multi-objetivo (AEMO). Dicho cálculo se torna muy costoso conforme se incrementa el número de objetivos. Por esta razón, el objetivo principal de este trabajo de tesis fue proponer un algoritmo capaz de realizar el cálculo de la contribución al hipervolumen de forma exacta de un AEMO con ayuda de las Unidades de Procesamiento Gráfico (GPUs por sus siglas en inglés), sin la necesidad de penalizar las propiedades matemáticas que el hipervolumen garantiza.

Para llevar a cabo este objetivo se realizó un estudio de los algoritmos más representativos del estado del arte para el cálculo del hipervolumen. Posteriormente, se analizaron las estrategias y modelos de paralelismo disponibles para llevar a cabo el diseño del algoritmo. Aunado a esto, se estudió la arquitectura y el modelo de programación para las GPUs. Como producto de este análisis, se propuso un algoritmo que fuera capaz de realizar el cálculo de la contribución al hipervolumen de forma exacta usando GPUs, el cual fue nombrado CHV_RC_GPU (Contribución al hipervolumen mediante el recorte de caja en una GPU). Para realizar el estudio experimental se adoptó un AEMO, denominado $\mathcal{S}$-*Metric Selection Evolutionary Multi-Objective Algorithm* (SMS-EMOA). Este AEMO utiliza el hipervolumen como criterio de selección, por lo que, el algoritmo propuesto fue integrado al SMS-EMOA para realizar una comparación con el mejor algoritmo del estado del arte para el cálculo del hipervolumen de forma exacta. La evaluación se realizó utilizando un conjunto de problemas multi-objetivo que reúnen diferentes características que causan dificultades a un AEMO.

Se realizaron estudios para evaluar el desempeño del SMS-EMOA con CHV_RC_GPU y del SMS-EMOA con FPL. Para ello, se utilizaron distintas métricas (tasa de convergencia y espaciamiento). Para tal fin, se estableció un número máximo de evaluaciones de las funciones objetivo para realizar la aproximación al conjunto de óptimos de Pareto, o en su defecto, se estableció no exceder de los 8000 minutos (5 días, 13 horas, 20 minutos) para la ejecución del AEMO.

El algoritmo propuesto (CHV_RC_GPU) mostró ser capaz de reducir el tiempo de ejecución del AEMO significativamente sin penalizar las propiedades matemáticas que

el hipervolumen ofrece. Los resultados experimentales confirman que CHV_RC_GPU es muy eficiente y efectivo al resolver problemas multi-objetivo, produciendo una buena aproximación al conjunto de óptimos de Pareto. Claramente, se muestra que nuestra implementación reduce el tiempo de ejecución significativamente, alcanzando una aceleración máxima de 883x, garantizando un buena distribución de las soluciones en el frente de Pareto. También se observó que nuestra propuesta resulta ser más eficiente conforme se incrementa la cantidad de datos manejados en la GPU. Esto se debe a la reducción del *overhead* que existe entre la CPU y la GPU, con lo que se explota mejor la cantidad de hilos creados en la GPU.

Como parte del trabajo futuro, sería interesante adaptar CHV_RC_GPU para generar vectores de pesos, los cuales son utilizados en algunos AEMOs para describir una superficie simplex. Estos vectores deben estar distribuidos uniformemente a lo largo del espacio objetivo para obtener diferentes soluciones del frente Pareto. En este sentido, una posibilidad para realizar esta aproximación es adoptar la contribución al hipervolumen utilizando nuestra propuesta.

Apéndice A

Funciones de prueba

En este apéndice se describen los dos conjuntos de prueba que suelen usarse más frecuentemente en optimización evolutiva multi-objetivo. Estos problemas de prueba examinan la capacidad y desempeño de los optimizadores evolutivos multi-objetivo, ya que proporcionan distintas fuentes de dificultad para alcanzar el verdadero frente de Pareto. De tal forma, estos problemas permiten evaluar la capacidad de un algoritmo evolutivo multi-objetivo para lidiar con distintas características que un problema multi-objetivo puede presentar (p.ej., multi-frontalidad, discontinuidades, concavidades, etc.).

A.1. Conjunto de problemas Deb-Thiele-Laumanns-Zitzler

El conjunto de prueba Deb-Thiele-Laumanns-Zitzler (DTLZ) [75] incluye nueve problemas para comparar el desempeño de los optimizadores evolutivos multi-objetivo. Este conjunto de problemas es escalable a cualquier número de variables de decisión y número de objetivos. La mayoría de estos problemas son separables, incluyendo frentes de Pareto multimodales. A continuación se presentan 4 problemas sin restricciones del conjunto de problemas DTLZ, donde la cantidad total de variables está definida por n, el número de objetivos por m y la cantidad de variables para la función g está definida por k. La función g define la dificultad del problema.

A.1.1. DTLZ1

Este problema tiene un frente de Pareto lineal, separable y multimodal. Está definido de la siguiente forma:

$$\begin{aligned}
&\text{Dado } \vec{x} = \{x_1, \ldots, x_{m-1}, x_m, \ldots, x_n\} \\
&\text{Minimizar } f_1(\vec{x}) = 0.5(1 + g(\vec{y})) \prod_{i=1}^{m-1} x_i \\
&f_{j=2:m-1}(\vec{x}) = 0.5(1 + g(\vec{y}))(1 - x_{m-j+1}) \prod_{i=1}^{m-j} x_i \\
&f_m(\vec{x}) = 0.5(1 + g(\vec{y}))(1 - x_1) \\
&\text{Donde } y_{i=1:k} = \{x_m, x_{m+1}, \ldots, x_n\} \\
&g(\vec{y}) = 100 \left[k + \sum_{i=1}^{k} (y_i - 0.5)^2 - \cos(20\pi(y_i - 05)) \right] \\
&\text{Sujeto a} \quad 0 \leq x_i \leq 1, \text{ para } i = 1, 2, \ldots, n.
\end{aligned} \tag{A.1}$$

La cantidad de variables se define como $n = m+k-1$, donde se sugiere una $k = 5$. Todos los valores de las funciones objetivo están en el hiper-plano $\sum_{i=1}^{m} f_i = 0.5$. Una solución es un óptimo de Pareto cuando $\vec{y} = (0, 0, \ldots)^T$. La dificultad de este problema es su multi-frontalidad. El espacio de búsqueda contiene $(11^k - 1)$ frentes de Pareto locales (ver figura A.1).

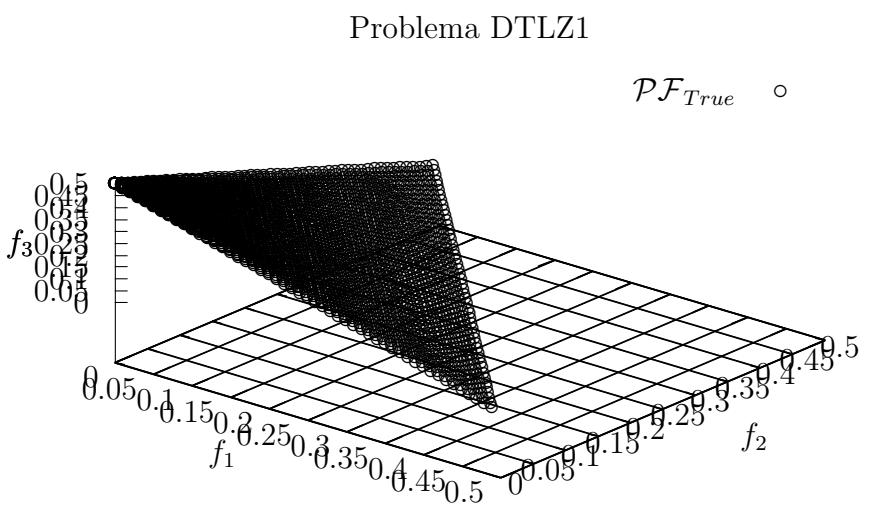


Figura A.1: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema DTLZ1

A.1.2. DTLZ2

Este problema tiene un frente de Pareto cóncavo y es definido como:

$$\begin{aligned}
\text{Dado } \vec{x} &= \{x_1, \ldots, x_{m-1}, x_m, \ldots, x_n\} \\
\text{Minimizar } f_1(\vec{x}) &= (1 + g(\vec{y})) \prod_{i=1}^{m-1} \cos\left(\frac{x_i \pi}{2}\right) \\
f_{j=2:m-1}(\vec{x}) &= (1 + g(\vec{y})) \left(\prod_{i=1}^{m-j} \cos\left(\frac{x_i \pi}{2}\right)\right) \operatorname{sen}\left(\frac{x_{m-j+1} \pi}{2}\right) \\
f_m(\vec{x}) &= (1 + g(\vec{y})) \operatorname{sen}\left(\frac{x_1 \pi}{2}\right) \\
\text{Donde } y_{i=1:k} &= \{x_m, x_{m+1}, \ldots, x_n\} \\
g(\vec{y}) &= \sum_{i=1}^{k} (y_i - 0.5)^2 \\
\text{Sujeto a} \quad & 0 \leq x_i \leq 1, \text{ para } i = 1, 2, \ldots, n.
\end{aligned} \tag{A.2}$$

La cantidad de variables se define como $n = m + k - 1$ (se sugiere un valor de $k = 10$). Una solución es un óptimo de Pareto cuando $\vec{y} = (0.5, 0.5, \ldots)^T$ y todos los valores de las funciones objetivo satisfacen $\sum_{i=1}^{m} (f_i)^2 = 1$. En la figura A.2 se muestra el verdadero frente de Pareto.

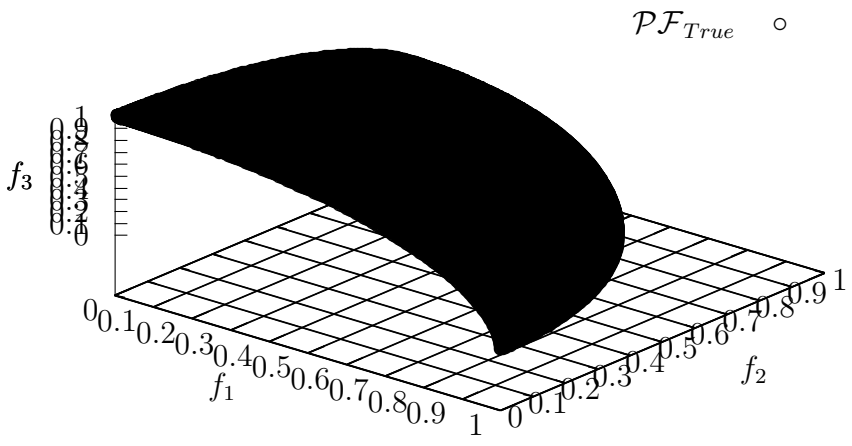


Figura A.2: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema DTLZ2

A.1.3. DTLZ3

Este problema es el mismo que DTLZ2 excepto por la función g, la cual es multimodal (ver figura A.3). Por lo tanto se define como:

$$\begin{aligned}
\text{Dado } \vec{x} &= \{x_1, \ldots, x_{m-1}, x_m, \ldots, x_n\} \\
\text{Minimizar } f_1(\vec{x}) &= (1 + g(\vec{y})) \prod_{i=1}^{m-1} \cos\left(\frac{x_i \pi}{2}\right) \\
f_{j=2:m-1}(\vec{x}) &= (1 + g(\vec{y})) \left(\prod_{i=1}^{m-j} \cos\left(\frac{x_i \pi}{2}\right)\right) \operatorname{sen}\left(\frac{x_{m-j+1} \pi}{2}\right) \\
f_m(\vec{x}) &= (1 + g(\vec{y})) \operatorname{sen}\left(\frac{x_1 \pi}{2}\right) \\
\text{Donde } y_{i=1:k} &= \{x_m, x_{m+1}, \ldots, x_n\} \\
g(\vec{y}) &= 100 \left[k + \sum_{i=1}^{k} (y_i - 0.5)^2 - \cos(20\pi(y_i - 05))\right] \\
\text{Sujeto a} \quad & 0 \leq x_i \leq 1, \text{ para } i = 1, 2, \ldots, n.
\end{aligned} \tag{A.3}$$

Al igual que DTLZ2 el valor sugerido de k es 10. La función g introduce $(3^k - 1)$ frentes de Pareto locales y un frente de Pareto global. Todos los frentes de Pareto locales son paralelos al frente de Pareto global. Por lo que, este problema es de gran ayuda para medir el desempeño de un optimizador multi-objetivo. Una solución pertenece al conjunto de óptimos de Pareto si $\vec{y} = (0.5, 0.5, \ldots)^T$.

A.1.4. DTLZ4

Este problema tiene un frente de Pareto que es cóncavo, separable y unimodal, y se define como:

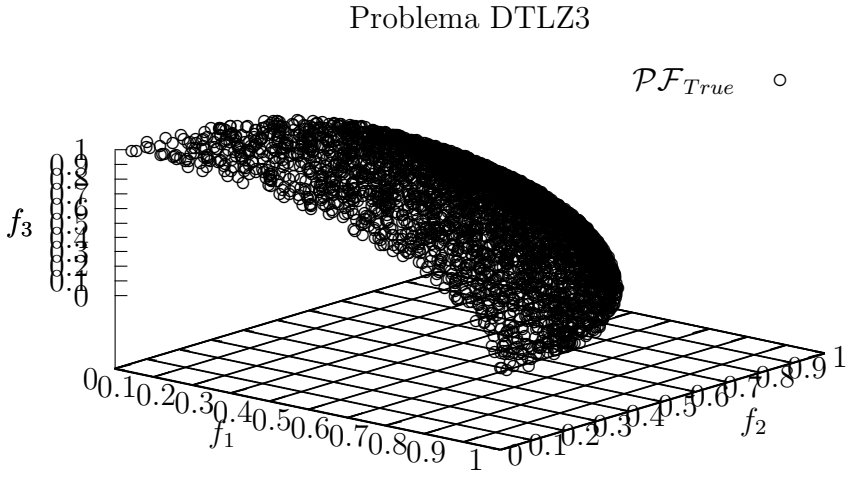


Figura A.3: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema DTLZ3

Dado

$$\vec{x} = \{x_1, \ldots, x_{m-1}, x_m, \ldots, x_n\}$$

Minimizar

$$\begin{aligned} f_1(\vec{x}) &= (1 + g(\vec{y})) \prod_{i=1}^{m-1} \cos\left(\frac{x_i^\alpha \pi}{2}\right) \\ f_{j=2:m-1}(\vec{x}) &= (1 + g(\vec{y})) \left(\prod_{i=1}^{m-j} \cos\left(\frac{x_i^\alpha \pi}{2}\right)\right) \operatorname{sen}\left(\frac{x_{m-j+1}^\alpha \pi}{2}\right) \\ f_m(\vec{x}) &= (1 + g(\vec{y})) \operatorname{sen}\left(\frac{x_1^\alpha \pi}{2}\right) \end{aligned}$$

Donde

$$\begin{aligned} y_{i=1:k} &= \{x_m, x_{m+1}, \ldots, x_n\} \\ g(\vec{y}) &= 100 \left[k + \sum_{i=1}^{k} (y_i - 0.5)^2 - \cos(20\pi(y_i - 05))\right] \end{aligned}$$

Sujeto a $\quad 0 \leq x_i \leq 1, \text{ para } i = 1, 2, \ldots, n.$

(A.4)

Se sugieren los siguientes parámetros: $\alpha = 100$ y $k = 10$. Este problema pone a prueba la capacidad de un optimizador multi-objetivo para producir una buena distribución de las soluciones. Estas modificaciones permiten un descenso del conjunto de soluciones muy cerca del plano $f_m - f_1$ (ver figura A.4).

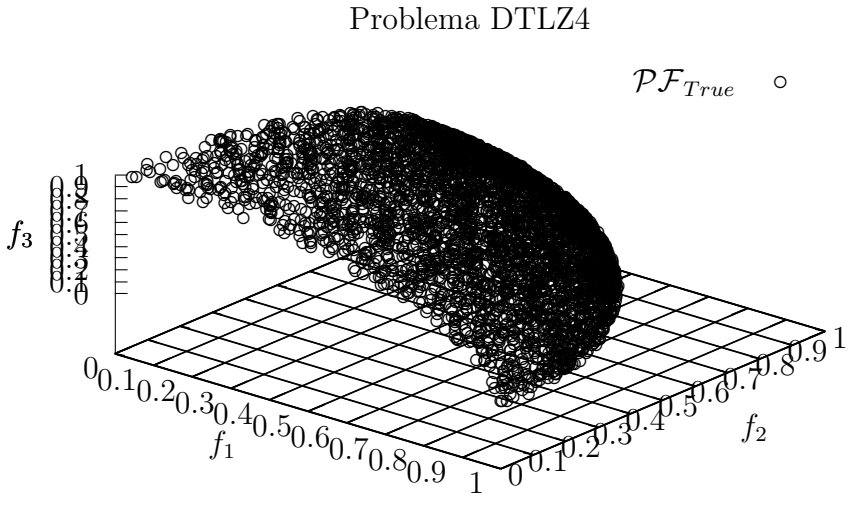


Figura A.4: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema DTLZ4

A.2. Conjunto de problemas *Walking-Fish-Group* (WFG)

El conjunto de problemas WFG fue publicado en el 2005 por Huband [76], quien propuso nueve problemas de prueba (WFG1-WFG9). Estos problemas de pruebas son escalables con respecto al número de variables y al número de objetivos. Estos problemas tiene una amplia variedad de formas geométricas para los frentes de Pareto. Sin embargo, las características como el sesgo, la multi-modalidad y la no separabilidad están definidas por un conjunto de transformaciones. A continuación se muestran las definiciones de tres problemas del conjunto WFG, donde m representa el número de objetivos, y cada problema es definido en términos de un vector de parámetros $\vec{x} \in \mathbb{R}^m$, el cual corresponde al espacio de las variables de decisión. Para toda $x_i \in \vec{x}$ su dominio es $[0, 1]$. La variable x_m es conocida como el parámetro de distancia subyacente y $x_{1:m-1}$ son los parámetros de posición subyacente. Por lo tanto, el vector $\vec{x}$ se obtiene al paso de varias transiciones a partir del vector de variables $\vec{z} \in \mathbb{R}^n$. El dominio del vector de variables $\vec{z}$ se define como $z_i \in [0, 2i]$. Cabe mencionar que se debe cumplir $n \geq m$ y $n = k + l$, donde $k \in \{m - 1, 2(m - 1), 3(m - 1), \dots\}$,

y describe la posición de los parámetros subyacentes. Por último, $l \in \{1, 2, 3, \dots\}$ describe la distancia de los parámetros subyacentes.

A.2.1. WFG1

Este problema de prueba es separable y unimodal, pero tiene una región plana y polinomial (ver figura A.5), y se define de la siguiente forma:

$$
\begin{aligned}
\text{Dado } \vec{z} &= \{z_1, \dots, z_k, z_{k+1}, \dots, z_n\} \\
\text{Minimizar } f_1(\vec{x}) &= x_m + 2 \prod_{i=1}^{m-1} \left(1 - \cos\left(\frac{x_i \pi}{2}\right)\right) \\
f_{j=2:m-1} &= x_m + 2j \left(\prod_{i=1}^{m-j} \left(1 - \cos\left(\frac{x_i \pi}{2}\right)\right)\right) \left(1 - \operatorname{sen}\left(\frac{x_{m-j+1}\pi}{2}\right)\right) \\
f_m(\vec{x}) &= x_m + 2m \left(1 - x_1 - \frac{\cos\left(\frac{10\pi x_1}{2}\right)}{10\pi}\right)
\end{aligned}
$$

Donde

$$
\begin{aligned}
x_{i=1:m-1} &= \text{r\_sum}\left(\{y_{(i-1)k/(m-1)+1}, \dots, y_{ik/(m-1)}\}, \left\{\frac{2(i-1)k}{(m-1)} + 1, \dots, \frac{2ik}{(m-1)}\right\}\right) \\
x_m &= \text{r\_sum}\left(\{y_{k+1}, \dots, y_n\}, \{2(k+1), \dots, 2n\}\right) \\
y_{i=1:n} &= \text{b\_poly}(y'_i, 0.02) \\
y'_{i=1:k} &= y''_i \\
y'_{i=k+1:n} &= \text{b\_flat}(y''_i, 0.8, 0.75, 0.85) \\
y''_{i=1:k} &= \frac{z_i}{2i} \\
y''_{i=k+1:n} &= \text{s\_linear}\left(\frac{z_i}{2i}, 0.35\right)
\end{aligned}
\tag{A.5}
$$

A.2.2. WFG2

Este problema es no separable y multimodal. El frente de Pareto es desconectado (ver figura A.6), y se define de la siguiente forma:

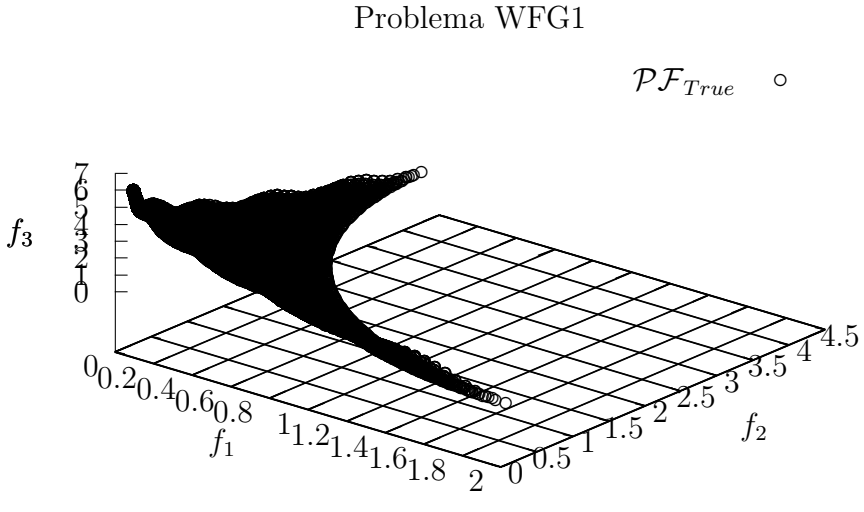


Figura A.5: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema WFG1

$$
\begin{aligned}
\text{Dado } \vec{z} &= \{z_1, \ldots, z_k, z_{k+1}, \ldots, z_n\} \\
\text{Minimizar } f_1(\vec{x}) &= x_m + 2 \prod_{i=1}^{m-1} \left(1 - \cos\left(\frac{x_i \pi}{2}\right)\right) \\
f_{j=2:m-1} &= x_m + 2j \left(\prod_{i=1}^{m-j} \left(1 - \cos\left(\frac{x_i \pi}{2}\right)\right)\right) \left(1 - \text{sen}\left(\frac{x_{m-j+1}\pi}{2}\right)\right) \\
f_m(\vec{x}) &= x_m + 2m \left(1 - x_1 \cos^2(5x_1\pi)\right)
\end{aligned}
$$

Donde

$$
\begin{aligned}
x_{i=1:m-1} &= \text{r\_sum}\left(\{y_{(i-1)k/(m-1)+1}, \ldots, y_{ik/(m-1)}\}, \{1, \ldots, 1\}\right) \\
x_m &= \text{r\_sum}\left(\{y_{k+1}, \ldots, y_{k+l/2}\}, \{1, \ldots, 1\}\right) \\
y'_{i=1:k} &= y'_i \\
y'_{i=k+1:k+l/2} &= \text{r\_nonsep}(\{y'_{k+2(i-k)-1}, y'_{k+2(i-k)}\}, 2) \\
y'_{i=1:k} &= \frac{z_i}{2i} \\
y'_{i=k+1:n} &= \text{s\_linear}\left(\frac{z_i}{2i}, 0.35\right)
\end{aligned}
\tag{A.6}
$$

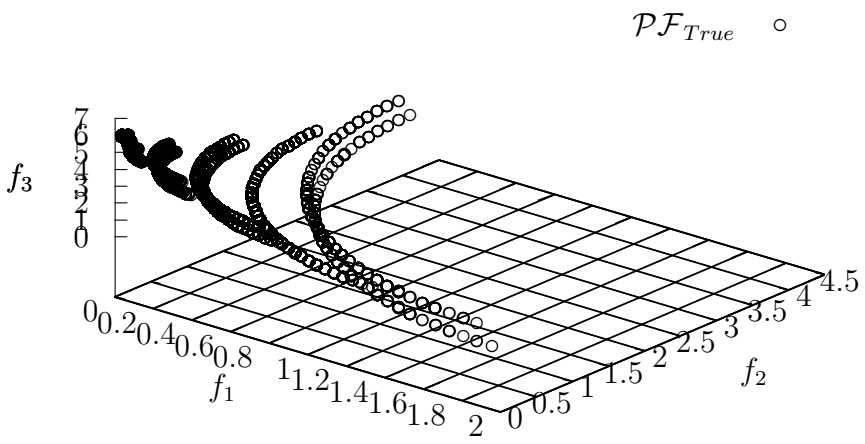


Figura A.6: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema WFG2

A.2.3. WFG3

Este problema es no separable pero es unimodal. El frente de Pareto es lineal (ver figura A.7), y está definido de la siguiente forma:

$$
\begin{aligned}
\text{Dado } \vec{z} &= \{z_1, \ldots, z_k, z_{k+1}, \ldots, z_n\} \\
\text{Minimizar } f_1(\vec{x}) &= x_m + 2 \prod_{i=1}^{m-1} (x_i) \\
f_{j=2:m-1} &= x_m + 2j \left(\prod_{i=1}^{m-j} x_i \right) (1 - x_{m-j+1}) \\
f_m(\vec{x}) &= x_m + 2m(1 - x_1) \\
\text{Donde } x_{i=1} &= u_i \\
x_{i=2:m-1} &= x_m(u_i - 0.5) + 0.5 \\
x_m &= \text{r\_sum}\left(\{y_{k+1}, \ldots, y_{k+l/2}\}, \{1, \ldots, 1\}\right) \\
u_i &= \text{r\_sum}\left(\{\ y_{(i-1)k/(m-1)+1}, \ldots, y_{ik/(m-1)}\}, \{1, \ldots, 1\}\right) \\
y'_{i=1:k} &= y'_i \\
y'_{i=k+1:k+l/2} &= \text{r\_nonsep}(\{y'_{k+2(i-k)-1}, y'_{k+2(i-k)}\}, 2) \\
y'_{i=1:k} &= \frac{z_i}{2i} \\
y'_{i=k+1:n} &= \text{s\_linear}\left(\frac{z_i}{2i}, 0.35\right)
\end{aligned}
\tag{A.7}
$$

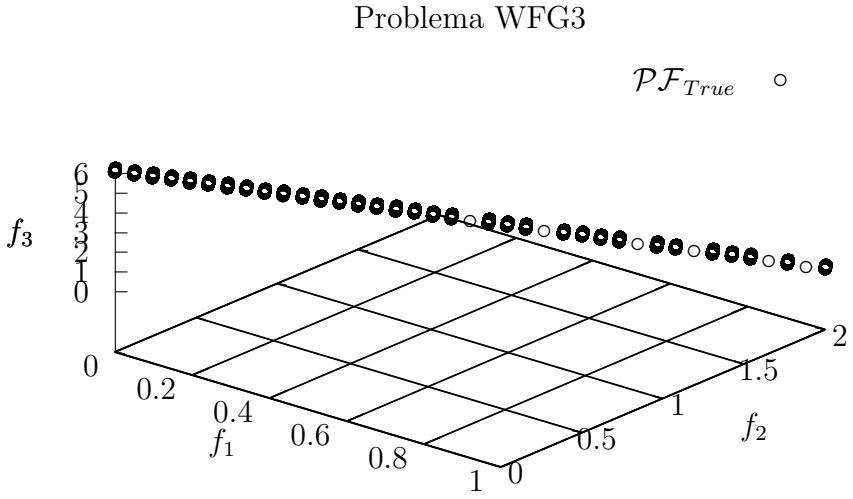


Figura A.7: Representación gráfica del verdadero frente de Pareto en tres dimensiones del problema WFG3

Bibliografía

[1] W.J. Cook. *Mathematical Programming Computation*. Springer, mathematical optimization society edition, September 2009.

[2] J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 711–716, May 2002.

[3] Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[4] J.J. Durillo, A.J. Nebro, C.A. Coello Coello, J. Garcia-Nieto, F. Luna, and E. Alba. A Study of Multiobjective Metaheuristics When Solving Parameter Scalable Problems. *IEEE Transactions on Evolutionary Computation*, 14(4):618–635, August 2010.

[5] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[6] K. Miettinen. *Nonlinear multiobjective optimization*. Kluwer Academic Publishers, Boston, 1999.

[7] Indraneel Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM J. on Optimization*, 8(3):631–657, March 1998.

[8] P. L. Yu. A class of solutions for group decision problems. *Management Science*, 19(8):936–946, 1973.

[9] L. A. Zadeh. Optimality and Non-Scalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control*, 8:59–60, 1963.

[10] L.S. Lasdon Y. Y. Haimes and D. A. Wisner. On a bicriterion formulation of the problems of integrated system identification and system optimization. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-1(3):296–297, July 1971.

[11] Shamik Chaudhuri and Kalyanmoy Deb. An interactive evolutionary multi-objective optimization and decision making procedure. *Applied Soft Computing*, 10(2):496–511, March 2010.

[12] Juan Castro-Gutierrez, Dario Landa-Silva, and José Moreno Pérez. Improved Dynamic Lexicographic Ordering for Multi-Objective Optimisation. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature–PPSN XI, 11th International Conference, Proceedings, Part II*, pages 31–40. Springer, Lecture Notes in Computer Science Vol. 6239, Kraków, Poland, September 2010.

[13] A. Charnes, W. W. Cooper, and R. O. Ferguson. Optimal estimation of executive compensation by linear programming. *Management Science*, 1(2):138–151, 1955.

[14] A. M. Geoffrion, J. S. Dyer, and A. Feinberg. An interactive approach for multi-criterion optimization, with an application to the operation of an academic department. *Management Science*, 19(4-Part-1):357–368, 1972.

[15] Carlos A. Coello Coello and Carlos E. Mariano Romero. Evolutionary Algorithms and Multiple Objective Optimization. In Matthias Ehrgott and Xavier Gandibleux, editors, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*, pages 277–331. Kluwer Academic Publishers, Boston, 2002.

[16] AndrzejP. Wierzbicki. The use of reference objectives in multiobjective optimization. In GÃ$\frac{1}{4}$nter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 468–486. Springer Berlin Heidelberg, 1980.

[17] K. Miettinen and M.M. Mäkelä. Interactive bundle-based method for nondifferentiable multiobjeective optimization: nimbus. *Optimization*, 34(3):231–246, 1995.

[18] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[19] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Bradford Books. MIT Press, 1992.

[20] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[21] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Alemania, 1973.

[22] Schwefel Hans Paul. *Numerical Optimization of Computer Models*. John Wiley & Sons Ltd, New York, USA, 1981.

[23] David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Enginers, New York, NY, USA, 1998.

[24] Carlos M. Fonseca and Peter J. Fleming. Multiobjective Optimization. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, volume 1, pages C4.5:1–C4.5:9. Institute of Physics Publishing and Oxford University Press, 1997.

[25] Richard S. Rosenberg. Simulation of genetic populations with biochemical properties: Ii. selection of crossover probabilities. *Mathematical Biosciences*, 8(12):1 – 37, 1970.

[26] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.

[27] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, December 1999.

[28] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evol. Comput.*, 3(1):1–16, March 1995.

[29] Jianjun Hu and Erik Goodman. Robust and Efficient Genetic Algorithms with Hierarchical Niching and a Sustainable Evolutionary Computation Model. In Kalyanmoy Deb et al., editor, *Genetic and Evolutionary Computation–GECCO 2004. Proceedings of the Genetic and Evolutionary Computation Conference. Part I*, pages 1220–1232, Seattle, Washington, USA, June 2004. Springer-Verlag, Lecture Notes in Computer Science Vol. 3102.

[30] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[31] David Edward Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In John. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, Hillsdale, Nueva Jersey, EE. UU., 1987. Lawrence Erlbaum.

[32] Thomas E. Koch and Andreas Zell. MOCS: Multi-Objective Clustering Selection Evolutionary Algorithm. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 423–430, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[33] Ali Farhang-Mehr and Shapour Azarm. Entropy-based multi-objective genetic algorithm for design optimization. *Structural and Multidisciplinary Optimization*, 24(5):351–361, November 2002.

[34] Tobias Wagner, Nicola Beume, and Boris Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Proceedings of the 4th International Conference on Evolutionary Multi-criterion Optimization*, EMO'07, pages 742–756, Berlin, Heidelberg, 2007. Springer-Verlag.

[35] M. Farina and P. Amato. On the optimal solution definition for many-criteria optimization problems. In *Fuzzy Information Processing Society, 2002. Proceedings. NAFIPS. 2002 Annual Meeting of the North American*, pages 233–238, 2002.

[36] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. A unified model for multi-objective evolutionary algorithms with elitism. In *In Congress on Evolutionary Computation (CEC 2000*, pages 46–53. IEEE Press, 2000.

[37] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata, editors, *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 862–876. Springer Berlin Heidelberg, 2007.

[38] Eckart Zitzler and Simon Künzli. Indicator-based Selection in Multiobjective Search. In Xin Yao et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag. Lecture Notes in Computer Science Vol. 3242.

[39] Kalyanmoy Deb, Kaisa Miettinen, and Deepak Sharma. A hybrid integrated multi-objective optimization procedure for estimating nadir point. In Matthias Ehrgott, Carlos M. Fonseca, Xavier Gandibleux, Jin-Kao Hao, and Marc Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, volume 5467 of *Lecture Notes in Computer Science*, pages 569–583. Springer Berlin Heidelberg, 2009.

[40] Stanley Zionts. Decision making: Some experiences, myths and observations. In Ganter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making*,

volume 448 of *Lecture Notes in Economics and Mathematical Systems*, pages 233–241. Springer Berlin Heidelberg, 1997.

[41] Raquel Hernandez and Carlos Coello. Mombi: A new metaheuristic for many-objective optimization based on the r2 indicator. In *2013 IEEE Conference on Evolutionary Computation*, volume 1, pages 2488–2495, June 20-23 2013.

[42] Nicola Beume, Boris Naujoks, and Michael Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, September 2007.

[43] O. Schütze, X. Esquivel, A Lara, and Carlos A Coello Coello. Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 16(4):504–522, Aug 2012.

[44] David Allen Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, USA, 1999. AAI9928483.

[45] Carlos A.Coello Coello and Nareli Cruz Cortés. Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 6(2):163–190, 2005.

[46] Heike Trautmann, Günter Rudolph, Christian Dominguez-Medina, and Oliver Schütze. Finding evenly spaced pareto fronts for three-objective optimization problems. In Oliver Schütze, Carlos A. Coello Coello, Alexandru-Adrian Tantar, Emilia Tantar, Pascal Bouvry, Pierre Del Moral, and Pierrick Legrand, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, volume 175 of *Advances in Intelligent Systems and Computing*, pages 89–105. Springer Berlin Heidelberg, 2013.

[47] Johannes Bader and Eckart Zitzler. HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evolutionary Computation*, 19(1):45–76, Spring, 2011.

[48] Clay Breshears. *The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications*. O'Reilly Media, Inc., 2009.

[49] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 114–118, New York, NY, USA, 1978. ACM.

[50] B. Alpern, L. Carter, and J. Ferrante. Modeling parallel computers as memory hierarchies. In *Programming Models for Massively Parallel Computers, 1993. Proceedings*, pages 116–123, Sep 1993.

[51] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[52] David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken. Logp: A practical model of parallel computation. *Commun. ACM*, 39(11):78–85, November 1996.

[53] Cristobal Navarro, Nancy Hitschfeld, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics (CiCP)*, 15:285–329, February 2014.

[54] Seon Wook Kim and Rudolf Eigenmann. The structure of a compiler for explicit and implicit parallelism. In *Proceedings of the 14th International Conference on Languages and Compilers for Parallel Computing*, LCPC'01, pages 336–351, Berlin, Heidelberg, 2003. Springer-Verlag.

[55] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569–, September 1965.

[56] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[57] NVIDIA Corporation. Gpu applications transforming computational research and engineering, 2013.

[58] NVIDIA Corporation. Directcompute para nvidia, 2013.

[59] OpenACC. Openacc directives for accelerators, 2013.

[60] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms: A comparative case study. In AgostonE. Eiben, Thomas Back, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–301. Springer Berlin Heidelberg, November 1998.

[61] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[62] Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Comput. Geom. Theory Appl.*, 43(6-7):601–610, August 2010.

[63] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *Proceedings of the 4th International Conference on Evolutionary*

Multi-criterion Optimization, EMO'07, pages 862–876, Berlin, Heidelberg, 2007. Springer-Verlag.

[64] Brualdi Richard A. *Introductory Combinatorics*. Prentice Hall, China, 5th edition, 2010.

[65] M. Fleischer. The measure of pareto optima applications to multi-objective metaheuristics. In *Proceedings of the 2Nd International Conference on Evolutionary Multi-criterion Optimization*, EMO'03, pages 519–533, Berlin, Heidelberg, 2003. Springer-Verlag.

[66] Lyndon While. A new analysis of the lebmeasure algorithm for calculating hypervolume. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, EMO'05, pages 326–340, Berlin, Heidelberg, 2005. Springer-Verlag.

[67] L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. *Evolutionary Computation, IEEE Transactions on*, 10(1):29–38, February 2006.

[68] C.M. Fonseca, L. Paquete, and M. Lopez-Ibanez. An improved dimension-sweep algorithm for the hypervolume indicator. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1157–1163, July 2006.

[69] Victor Klee. Can the measure of $\bigcup_1^n$ [ai, bi] be computed in less than $\mathcal{O}(n \log n)$ steps? In *The American Mathematical Monthly*, volume 84, No. 4, pages 284–285. Mathematical Association of America, April 1977.

[70] M.H. Overmars and Chee-Kang Yap. New upper bounds in klee's measure problem. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 550–556, Oct 1988.

[71] Karl Bringmann. Bringing order to special cases of klee's measure problem. In Krishnendu Chatterjee and Sgall, editors, *Mathematical Foundations of Computer Science 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin Heidelberg, 2013.

[72] Nicola Beume. S-metric calculation by considering dominated hypervolume as klee's measure problem. *Evol. Comput.*, 17(4):477–492, December 2009.

[73] Lucas Bradstreet. *The Hypervolume Indicator for Multi-objective Optimisation: Calculation and Use*. University of Western Australia, 2011.

[74] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[75] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. In Ajith Abraham, Lakhmi Jain, and Robert Goldberg, editors, *Evolutionary Multiobjective Optimization*, Advanced Information and Knowledge Processing, pages 105–145. Springer London, 2005.

[76] Simon Huband, Luigi Barone, Lyndon While, and Phil Hingston. A scalable multi-objective test problem toolkit. In Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, pages 280–295. Springer Berlin Heidelberg, 2005.

[77] Kalyanmoy Deb and Deb Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.