Mejora de arquitecturas *cloud* utilizando *Metamorphic Testing* y técnicas de optimización multiobjetivo

Improving cloud architectures using *Metamorphic Testing* and multi-objective optimization techniques

MIGUEL PÉREZ DE LA RUBIA

MÁSTER EN INGENIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería Informática

Fecha
Madrid, 5 de septiembre de 2023

Director/es y/o colaborador:

Alberto Núñez Covarrubias
Pablo Cerro Cañizares

Convocatoria: Septiembre de 2023. Calificación: 9

Resumen

Actualmente, debido a la evolución tecnológica existente, se produce una gran cantidad de datos que han de ser gestionados de forma eficiente. Para ello, los sistemas *cloud* ofrecen una alternativa consistente para reducir el tiempo de procesado. Sin embargo, esto se consigue aumentando los recursos computacionales drásticamente, lo cual conlleva un aumento significativo en el consumo energético del sistema. Teniendo esto en cuenta, además de las crisis climáticas y energéticas vividas en los últimos años, es necesario un nuevo enfoque que trate de minimizar el consumo energético de los sistemas *cloud* sin reducir – en la medida de lo posible – potencia computacional.

Con el fin de afrontar este problema, en este trabajo se han combinado técnicas de optimización multiobjetivo, junto con *Metamorphic Testing* (MT) y herramientas de simulación, para optimizar sistemas *cloud* teniendo en cuenta su rendimiento y el consumo energético. Para ello, se han integrado varios algoritmos genéticos multiobjetivo (MOGAs) en el *framework* MT-EA4Cloud, el cual, en su versión actual, aplica un único algoritmo evolutivo de un sólo objetivo con MT.

Además, se ha realizado un estudio empírico para analizar el comportamiento de los distintos MOGAs incluidos en el *framework*. Se han definido distintos conjuntos de test que representan distintas arquitecturas *cloud* y dos cargas de trabajo distintas inspiradas en las operaciones realizadas en el análisis de *big data*. En particular, las trazas empleadas representan la infraestructura de PlanetLab. Los resultados obtenidos muestran claramente que los MOGAs pueden combinarse con MT para optimizar sistemas *cloud* teniendo en cuenta varios objetivos, en este caso, rendimiento y consumo energético. Tras analizar estos resultados, podemos concluir que utilizar probabilidades altas en los operadores de mutación, proporciona los mejores resultados. Además, en términos generales, el algoritmo NSGAII es el que mejor resultados ha aportado en los experimentos realizados en este estudio.

Palabras clave

Algoritmos Genéticos Multiobjetivo, Metamorphic Testing, Sistemas Cloud y Simulación

Abstract

Currently, due to the existing technological evolution, a large amount of data is being generated that needs to be efficiently managed. For this purpose, cloud systems offer a consistent alternative to reduce processing time. However, this is achieved by drastically increasing computational resources, which leads to a significant increase in the system's energy consumption. Taking this into account, along with the climate and energy crises experienced in recent years, a new approach is needed to minimize the energy consumption of cloud systems without reducing computational power as much as possible.

In order to address this problem, this work combines multi-objective optimization techniques with Metamorphic Testing (MT) and simulation tools to optimize cloud systems while considering their performance and energy consumption. To achieve this, several multi-objective genetic algorithms (MOGAs) have been integrated into the MT-EA4Cloud framework, which in its current version applied a single-objective evolutionary algorithm with MT.

Furthermore, an empirical study has been conducted to analyze the behavior of the different MOGAs included in the framework. In this study, various test sets have been defined to represent different cloud architectures and two different workloads inspired by operations performed in big data analysis. In particular, the traces used represent the infrastructure of PlanetLab.

The results obtained clearly show that MOGAs can be combined with MT to optimize cloud systems while considering multiple objectives, in this case, performance and energy consumption. After analyzing these results, it can be concluded that using high probabilities in mutation operators provides the best results. Furthermore, in general terms, the NSGAII algorithm has yielded the best results in the experiments conducted in this study.

Keywords

Multi-Objective Genetic Algorithms, Metamorphic Testing, Cloud Systems, and Simulation

Índice general

Índice de Algoritmos

Índice de Figuras

Índice de Tablas

Agradecimientos

Agradecer a Pablo Cerro y Alberto Núñez la labor realizada como directores de este trabajo. La orientación aportada y el esfuerzo implicado por su parte ha sido crucial para poder sacar este proyecto adelante.

Dedicatoria

A todos los que me han ayudado en las noches sin dormir y me han acompañado en este camino. En especial a Mig, Pepe y Alex, quienes han ayudado a aclarar mis ideas en los momentos finales y han hecho una gran labor de apoyo.

Capítulo 1

Introducción

Este capítulo presenta el contexto global sobre el cual se ha desarrollado el presente trabajo. Las contribuciones principales del trabajo, así como la motivación, se presentan en la Sección 1.1. Los retos encontrados durante el desarrollo se presentan en la Sección 1.2. Los objetivos principales y el plan de trabajo se introducen en la Sección 1.3. Finalmente, la estructura del documento está resumida en la Sección 1.4.

1.1. Motivación

En la actualidad, estamos inmersos en una sociedad con clara tendencia a la evolución tecnológica. Un reflejo de ello son los servicios en línea, enmarcados en un entorno altamente competitivo, con cada vez más alternativas y donde se encuentra en auge la creación de nuevos servicios enfocados a liderar sus sectores. La transmisión de vídeo en tiempo real, redes sociales, y otros servicios bajo demanda son un claro ejemplo. Para extender su uso y afianzar su liderazgo, estos servicios deben dar cabida a millones de usuarios, conectados concurrentemente, proporcionando una calidad de servicio apropiada. Esto genera una cantidad masiva de datos, los cuales deben ser gestionados de manera que se proporcione al usuario sensación de inmediatez al realizar sus operaciones[1]. La gestión y procesamiento de esta gran cantidad de datos se ha convertido en un desafío que demanda un considerable espacio de almacenamiento y una gran potencia de cálculo. Por ello, se requiere la utilización de un elevado número de recursos computacionales, como centros de datos, redes de comunicaciones de baja latencia y grandes cantidades de almacenamiento.

Los sistemas *cloud*, y en general, la Computación de Alto Rendimiento (HPC, por sus siglas en inglés) son considerados como una potencial solución para abordar estos retos[2]. Estos sistemas explotan las ventajas de la paralelización del procesamiento de los datos para proporcionar una mejora significativa de rendimiento. Sin embargo, la mejora de tiempos de ejecución obtenida mediante la explotación del paralelismo conlleva un coste energético extremadamente alto[3]. A modo de ejemplo, según el estudio

más reciente (en junio de 2023) de los 500 superordenadores más rápidos del mundo, el superordenador Frontier alcanza un rendimiento de 1.194 Petaflops/s con 8.699.904 núcleos[4]. Este sistema requiere 22.703 kW de potencia por hora, lo que representa un coste de 5.448,72€ por hora si asumimos un costo de 0,24€ por kW.

Aunque las técnicas de HPC han demostrado poseer un enfoque efectivo, las crisis energéticas que hemos experimentado en los últimos tiempos han provocado una tendencia hacia el *Green Computing*. Estos esfuerzos han desembocado en nuevas aproximaciones para la mejora energética de los sistemas *cloud*[5], maximizando la eficiencia energética de los sistemas computacionales, tratando de reducir el consumo energético. Así pues, en las últimas décadas, las empresas del sector tecnológico han invertido sus esfuerzos en reducir el coste energético.

Un punto clave para mantener la competitividad de este tipo de sistemas, es encontrar un delicado equilibrio entre el rendimiento y su consumo energético. Si bien la potencia de procesamiento es esencial para abordar la creciente demanda de análisis de datos, es igualmente esencial minimizar la energía consumida, para así minimizar el impacto medioambiental y reducir los costes asociados a un volumen elevado de operaciones computacionales. Es por esto que, un aspecto esencial en este proceso es la creación de técnicas que permitan encontrar este equilibrio.

Debido a la complejidad de estos sistemas – los cuales pueden estar formados por cientos de miles de elementos computacionales – encontrar este equilibrio es una tarea compleja. Para ello, es necesario introducir cambios en la configuración y realizar modificaciones tanto en la topología, como en los múltiples elementos computacionales que componen estos sistemas. Por lo tanto, realizar estas modificaciones de forma manual puede considerarse una tarea costosa y propensa a fallos. Afortunadamente, a lo largo de las últimas décadas, los algoritmos genéticos han demostrado su utilidad en la exploración de espacios de búsqueda complejos, permitiendo realizar modificaciones sobre sistemas bajo estudio para alcanzar objetivos. Este tipo de algoritmos se basa en la idea de evolución natural de individuos, permitiendo la supervivencia de aquellos considerados más adecuados, cuya solución es considerada más cercana a los objetivos seleccionados. Los algoritmos genéticos no sólo se han enfocado en la optimización de problemas con un sólo objetivo, sino que también han demostrado ser altamente efectivos en el manejo de problemas con múltiples objetivos. En estos casos, el objetivo no consiste en encontrar la solución óptima, sino un conjunto de soluciones que representen un equilibrio entre los múltiples objetivos. Los algoritmos genéticos multiobjetivo utilizan técnicas de selección y clasificación para identificar y mantener un conjunto de soluciones que aborden diferentes compromisos entre los objetivos. Este enfoque se adecúa a las características requeridas para el proyecto, en el que se busca encontrar el equilibrio entre la energía consumida por el sistema sin perder capacidad computacional.

1.2. Retos

Pese a la alta aplicabilidad de los algoritmos genéticos para evolucionar sistemas cloud hacia unos objetivos determinados, aplicar estas técnicas sobre sistemas reales tiene asociadas una serie de desafíos inherentes: i) *Modificación y configuración de las infraestructuras subyacentes*. Tomemos como referencia el sistema MareNostrum 4 en Barcelona, un superordenador de cómputo compuesto por 48 racks[1] y 3.456 nodos con dos chips de 24 procesadores cada uno, además de una memoria central de 390 Terabytes. Realizar cambios en la arquitectura de este sistema podría ser una tarea costosa y compleja debido a la gran cantidad de componentes y su distribución en distintos espacios. Es importante destacar que en algunas situaciones, las modificaciones pueden llegar a no ser realizables, pues los usuarios finales pueden carecer de permisos para el acceso a la infraestructura subyacente del sistema; ii) *Costes asociados*. La naturaleza de los sistemas distribuidos demanda el acceso a una considerable cantidad de nodos y máquinas que puedan soportar cargas de trabajo prolongadas, consiguiendo de esta forma llevar a cabo el análisis de su consumo y tiempos de ejecución individuales. Sin embargo, esta experimentación acarrea una serie de costes económicos elevados relacionados con el acceso a dichas infraestructuras a través de proveedores de servicio; y iii) *Desafíos en la reproducibilidad de pruebas*. Las variaciones existentes en las medidas realizadas sobre los sistemas distribuidos, generadas por factores como alta concurrencia, latencia de red, y especialmente en sistemas *cloud*, la virtualización, presentan un obstáculo en la replicación precisa de pruebas. En los sistemas *cloud*, la tendencia a la virtualización maximiza los recursos, permitiendo la ejecución de tareas de múltiples usuarios simultáneamente, aunque esto puede influir en el rendimiento de las aplicaciones.

Estos desafíos mencionados pueden limitar la viabilidad de realizar experimentación en sistemas reales. Por esta razón, la opción de recurrir a herramientas de simulación resulta una solución viable para superar problemas de accesibilidad al sistema bajo estudio, reduciendo los costes de acceso, y posibilitando la reproducción de escenarios en un entorno controlado y repetible. En la actualidad, existe una gran cantidad de herramientas de simulación disponibles, que abarcan desde elementos individuales, como discos de almacenamiento o CPUs[6], hasta abarcar la representación de centros de datos complejos[7], que permiten modelar y representar el comportamiento de sistemas reales a nivel de arquitectura y ejecutar aplicaciones para entornos distribuidos.

Sin embargo, emplear herramientas de simulación provoca la aparición de nuevos desafíos. Entre ellos, uno de los más destacables radica en la dificultad de asegurar que el modelo simulado refleje el comportamiento deseado, es decir, que cumpla con las expectativas de su diseño. Por ello, la validación de

[1] Estructuras de montaje que albergan y organizan diversos componentes

estos modelos resulta un aspecto crucial para minimizar la aparición de fallos potencialmente críticos y garantizar su correcto funcionamiento. Habitualmente, se utilizan conjuntos de pruebas diseñados manualmente para comprobar la corrección del modelo. Esto genera una exploración poco exhaustiva del sistema, favoreciendo la existencia de potenciales casos de fallo que puedan afectar a su funcionamiento. Pese a ser una de las técnicas más extendidas para validar la corrección de sistemas. Las técnicas de *testing* tradicionales no suelen aplicarse en sistemas complejos debido a los dos problemas fundamentales: *el problema del oráculo* y *el problema del conjunto de tests confiable*. El primero hace referencia a ausencia de un mecanismo – ya sea por su alto coste computacional o por no haber sido implementado – que asegure que el sistema funcione con respecto a su especificación. El segundo concierne a la dificultad de generar un conjunto de test adecuado para comprobar la corrección de un sistema en un tiempo computacionalmente aceptable.

Metamorphic testing (MT) es una técnica de testing ampliamente reconocida por aliviar los dos problemas fundamentales del testing. Específicamente, en MT se utiliza conocimiento implícito del sistema bajo estudio en forma de reglas metamórficas (MRs, por sus siglas en inglés). Estas reglas son utilizadas para generar nuevos casos de prueba así como para construir un oráculo de manera sencilla. Debido a sus características, se considera que las MRs pueden ser integradas dentro del proceso de los algoritmos genéticos multiobjetivo (MOGA, por sus siglas en inglés), permitiendo una búsqueda guiada y controlada dentro del proceso evolutivo. Para ello, se introducen cambios a los individuos de manera que se sigan manteniendo las restricciones incluidas en las MRs. Por ello, en este trabajo se propone la integración de tres campos ortogonales: los MOGAs, MT y simulación. Para la optimización multiobjetivo de sistemas cloud.

1.3. Objetivos y Plan de trabajo

El objetivo principal de este trabajo es **evaluar la viabilidad del uso de algoritmos de optimización multiobjetivo para optimizar sistemas *cloud***. Para alcanzar este objetivo, se propone la integración de técnicas de simulación, que permiten modelar y simular sistemas distribuidos, junto con técnicas de *metamorphic testing*, que puedan aplicarse para optimizar la búsqueda en el espacio de soluciones, facilitando la evolución controlada de los individuos a través de conocimiento especifico del sistema. Para la consecución de este objetivo se ha partido de la base del trabajo presentado por Cañizares et al.[8], donde los autores proponen un *framework*, llamado MT-EA4Cloud, el cual combina un algoritmo evolutivo de un sólo objetivo con *metamorphic testing*.

Seguidamente, se muestra el plan de trabajo seguido, donde cada punto refleja un objetivo secundario

del trabajo:

- Estudio y análisis. Durante esta fase se ha estudiado el marco de trabajos previos más relevantes sobre los algoritmos genéticos multiobjetivos más relevantes. El objetivo es comprender en profundidad estas técnicas para realizar una exploración exhaustiva de la literatura y seleccionar los algoritmos más relevantes del estado del arte. Además, se ha realizado un estudio comparativo entre todas las propuestas seleccionadas, determinando sus características principales, así como las similitudes y diferencias con otras propuestas.

- Implementación. En la fase de implementación, tras la selección de los algoritmos, nos enfocamos en comprender la estructura de programación del trabajo previo de Cañizares et al.[8] para expandir eficientemente el *framework*. La expansión de este *framework* está enfocada en añadir los algoritmos genéticos estudiados en la fase anterior y seguir garantizado una arquitectura flexible para incorporar nuevos algoritmos y objetivos de manera sencilla.

- Depuración. Una vez implementados los algoritmos, la fase de depuración examina el código en detalle con el objetivo de detectar posibles errores durante la implementación y asegurar la validez de la implementación de los algoritmos añadidos al *framework*.

- Test y análisis de resultados. Esta fase propone la realización de un estudio empírico enfocado en el análisis de la adecuación y eficacia tanto del *framework* como de los algoritmos añadidos. Para ello, hemos diseñado una serie de pruebas en las que evaluamos diversos modelos de sistemas *cloud* y cargas computacionales con cada uno de los algoritmos multiobjetivo. Posteriormente, analizamos los resultados obtenidos para identificar cuáles de estos algoritmos se ajustan mejor a este tipo de sistemas.

1.4. Estructura del documento

El resto del documento está organizado en los siguientes capítulos:

- Capítulo 3, Preliminares, introduce los conceptos preliminares necesarios para la comprensión de las técnicas utilizadas durante el desarrollo del trabajo.

- Capítulo 4, Estado del arte, presenta un análisis de los trabajos más relevantes en los campos de problemas de optimización con algoritmos multiobjetivo, simulación de sistemas distribuidos, *metamorphic testing* y aplicación de algoritmos genéticos sobre sistemas Cloud computing y HPC.

- Capítulo 5, *Framework* propuesto para la optimización multiobjetivo de sistemas *cloud*, presenta y describe en detalle cómo se ha llevado a cabo la integración en un *framework* multiobjetivo las diferentes técnicas utilizadas. Además, se describen los algoritmos utilizados, así como los detalles de implementación de cada uno de ellos.

- Capítulo 6, Estudio empírico, se presentan los experimentos desarrollados durante esta fase de experimentación. Estos experimentos tratan de comprobar mediante los resultados obtenidos la idoneidad de la aplicación de los algoritmos genéticos multiobjetivo frente a un algoritmo genético tradicional.

- Capítulo 7, Conclusiones y trabajo futuro, este capítulo concluye el trabajo con las enseñanzas adquiridas y algunas líneas de trabajo futuro.

Capítulo 2

Introduction

The main objective of this chapter is to provide an overview of the global context in which the present work has been developed. The main contributions of the work, as well as the motivation, are presented in Section 2.1. The challenges faced during the development are discussed in Section 2.2. The main objectives and the work plan are introduced in Section 2.3. Finally, the document's structure is summarized in Section 2.4.

2.1. Motivation

Currently, we find ourselves immersed in a society characterized by a clear tendency toward technological evolution. One illustration of this trend can be seen in the proliferation of online services in an increasingly competitive environment, where the range of alternatives continues to expand. An exemplification of this trend is the proliferation of online services within a highly competitive environment, with an ever-increasing array of alternatives. Furthermore, there is a notable surge in the creation of new services with the aspiration to assume leadership roles within their respective sectors. Real-time video streaming, social networks, and other on-demand services serve as prominent examples. In order to expand their user base and solidify their leadership, these services must accommodate millions of concurrently connected users while providing an appropriate quality of service.

This fact leads to the generation of a massive volume of data, requiring management practices that provide users with a sense of immediacy in their operations[1]. The management and processing of such vast quantities of data have become a challenge, demanding a large storage capacity and an extensive computing power. Consequently, the utilization of a considerable number of computational resources, including data centers, low-latency communication networks, and huge storage facilities, is imperative.

Cloud systems and, in general, High-Performance Computing (HPC) are considered potential solutions to address these challenges[2]. These systems leverage the advantages of data processing parallelization to provide a significant performance improvement. However, the improvement in execution

times achieved through parallelism comes with an extremely high energy cost[3]. For example, according to the most recent study (as of June 2023) of the world's 500 fastest supercomputers, the Frontier supercomputer achieves a performance of 1,194 Petaflops/s with 8,699,904 cores[4]. This system requires 22,703 kW of power per hour, which translates to a cost of €5,448.72 per hour assuming a cost of €0.24 per kW.

While HPC techniques have proven to be effective, recent energy crises have led to a trend toward Green Computing. These efforts have resulted in new approaches to improve the energy efficiency of cloud systems[5], maximizing the energy efficiency of computational systems to reduce energy consumption. Consequently, in recent decades, technology sector companies have directed their efforts toward reducing energy costs.

A key point in maintaining the competitiveness of such systems is to reach a balance between execution time and energy consumption. While processing power is essential to meet the growing demand for data analysis, it is equally crucial to minimize energy consumption to reduce environmental impact and the costs associated with a high volume of computational operations. Hence, an essential aspect of this process is the development of techniques that enable finding this balance.

Due to the complexity of these systems, which may consist of hundreds of thousands of computational elements, achieving this balance is a challenging task. For this, it is neccessary to introduce changes in configuration and modifications in both the topology and the multiple computational elements that compose these systems. Therefore, performing these modifications manually can be considered a costly and error-prone task. Fortunately, over the past decades, genetic algorithms have proven their utility in exploring complex search spaces, enabling to evolve the systems under study to achieve desired objectives. These algorithms are based on the concept of the natural evolution of individuals, allowing the survival of those considered fittest, whose solutions are deemed closer to the selected objectives. Genetic algorithms have not only focused on optimizing single-objective problems but have also proven to be highly effective in handling multi-objective problems. In these cases, the goal is not to find the optimal solution but a set of solutions representing a balance among multiple objectives. Multi-objective genetic algorithms use selection and ranking techniques to identify and maintain a diverse set of solutions that address different trade-offs among the objectives. This genetic algorithm approach aligns with the requirements of the project, which aims to find a balance between the energy consumed by the system without sacrificing computational capacity.

2.2. Challenges

Despite the high applicability of genetic algorithms for evolving cloud systems toward specific objectives, applying these techniques to real systems presents a series of inherent challenges: i) *Modification and Configuration of Underlying Infrastructures*: Consider, for example, the MareNostrum 4 in Barcelona, a supercomputer that consists of 48 racks[1] and 3,456 nodes with two chips of 24 processors each, along with a central memory of 390 Terabytes. Introducing changes to the architecture of such a system could be a costly and complex task due to the large number of components and their distribution across various spaces. It is important to note that in some situations, modifications may not be feasible as end users may lack permissions to access the underlying infrastructure of the system; ii) *Associated Costs*: Distributed systems inherently require access to a considerable number of nodes and machines capable of sustaining extended workloads, which involves analyzing their individual consumption and execution times. However, this experimentation incurs high economic costs associated with accessing such infrastructures through service providers; and iii) *Challenges in Test Reproducibility*: Variations in measurements taken on distributed systems, influenced by factors such as high concurrency, network latency, and especially virtualization in cloud systems, pose a challenge in precisely replicating tests. In cloud systems, the trend toward virtualization maximizes resource utilization, allowing for the execution of tasks from multiple users simultaneously, which can influence application performance.

These challenges mentioned can limit the feasibility of conducting experiments on real systems. Therefore, simulation techniques becomes a viable solution to overcome issues related to system accessibility, reduce access costs, and enable the reproduction of scenarios in a controlled and repeatable environment.

Currently, multiple simulation tools are available in the literature. These tools range from simulating individual elements such as storage disks or CPUs[6] to representing complex data centers[7]. They allow for modeling and representing the behavior of real systems at the architectural level and executing applications for distributed environments.

However, employing simulation tools introduces new challenges. One of the most notable challenges is the difficulty of ensuring that the simulated model accurately reflects the desired behavior, i.e., fulfills the expectations of its design. Therefore, model validation is a crucial aspect to minimize the appearance of potentially critical failures and ensure proper functioning. Typically, manually designed test sets are used to validate the correctness of the model. This approach results in a less exhaustive exploration of the system, increasing the likelihood of potential failure cases that could affect its operation.

[1]Mounting structures housing and organizing various components

Despite being one of the most widely adopted techniques for validating system correctness, traditional testing techniques are not commonly applied to complex systems due to two fundamental problems: the oracle problem and the reliable test set problem. The oracle problem refers to the absence of a mechanism—either due to its high computational cost or lack of implementation—that ensures that the system behave according to its specification. The reliable test set problem concerns the difficulty of generating an adequate test set to check the correctness of a system within computationally acceptable timeframes.

Metamorphic testing (MT) is a widely recognized testing technique that effectively addresses the two fundamental problems of testing. Specifically, MT utilizes implicit knowledge of the system under study in the form of metamorphic rules (MRs). These rules are employed to generate new test cases and construct an oracle in a straightforward manner.

Due to their characteristics, it is considered that MRs can be integrated into the process of Multi-Objective Genetic Algorithms (MOGAs), allowing for a guided and controlled search within the evolutionary process. To achieve this, changes are introduced to individuals while still adhering to the constraints included in the MRs. Thus, this work proposes the integration of three orthogonal fields: MOGAs, MT, and simulation, for the multi-objective optimization of cloud systems.

2.3. Objectives

The main objective of this work is to **evaluate the feasibility of using multi-objective optimization algorithms to optimize cloud systems**. To achieve this goal, we propose the integration of simulation techniques, which allow modeling and simulating distributed systems, along with metamorphic testing techniques that can be applied to optimize the search in the solution space, facilitating the controlled evolution of individuals through specific system knowledge. To accomplish this objective, we build upon the work presented by Cañizares et al.[8], where the authors propose a framework called MT-EA4Cloud, which combines a single-objective evolutionary algorithm with metamorphic testing.

Next, we present the work plan, where each point reflects a secondary objective of the project:

- **Study and Analysis**: During this phase, we have studied the most relevant previous work on multi-objective genetic algorithms. The goal is to gain a deep understanding of these techniques to conduct a comprehensive literature review and select the most relevant algorithms from the state of the art. Furthermore, we have conducted a comparative study of all the selected proposals, defining their main characteristics, as well as similarities and differences with other approaches.

- **Implementation**: In the implementation phase, following the selected algorithms, we focus on

understanding the programming structure of the previous work by Cañizares et al.[8] to efficiently extend the framework. The expansion of this framework is focused on adding the genetic algorithms studied in the previous phase while ensuring a flexible architecture for the straightforward incorporation of new algorithms and objectives.

- **Debugging**: Once the algorithms are implemented, the debugging phase examines the code in detail with the aim of detecting possible errors during implementation and ensuring the validity of the implementation of the algorithms added to the framework.

- **Testing and Results Analysis**: This phase proposes conducting an empirical study focused on the analysis of the suitability and effectiveness of both the framework and the added algorithms. For this, we have designed several set of tests in which we evaluate various models of cloud systems and computational workloads with each of the multi-objective algorithms. Subsequently, we analyze the results obtained to identify which of these algorithms best fit this type of systems.

2.4. Document Organization

The rest of the document is organized into the following chapters:

- Chapter 3, Preliminaries, introduces the preliminary concepts for understanding the techniques used during the development of the work.

- Chapter 4, State of the Art, presents an analysis of the most relevant works in the fields of multi-objective optimization problems, simulation of distributed systems, metamorphic testing, and the application of genetic algorithms to Cloud computing and HPC systems.

- Chapter 5, Proposed Framework for Multi-Objective Optimization of Cloud Systems, presents and describes in detail how the integration of various techniques into a multi-objective framework has been carried out. It also describes the algorithms used, as well as the implementation details of each one.

- Chapter 6, Empirical Study, presents the experiments conducted during this experimentation phase. These experiments aim to validate, through the results obtained, the suitability of applying multi-objective genetic algorithms compared to a traditional genetic algorithm.

- Chapter 7, Conclusions and Future Work, concludes the work with the lessons learned and some future lines of research.

Capítulo 3

Preliminares

En este capítulo se realiza una introducción a algunos de los conceptos principales que se usarán en este trabajo: computación en sistemas *cloud* eficientes energéticamente (Sección 3.1), algoritmos de optimización (Sección 3.2) y *metamorphic testing* (Sección 3.3).

3.1. Computación en sistemas *cloud* eficientes energéticamente

En la actualidad, el consumo energético en los sistemas *cloud* es uno de los principales factores a tener en cuenta[9][10]. En general, un alto coste energético puede resultar una deficiencia en el sistema, ya que disminuye la rentabilidad de la inversión realizada y aumenta el coste total de las infraestructuras. Un elevado coste económico tiene un impacto negativo en los usuarios finales, afectando tanto a aquellos que disponen de su propia infraestructura como a aquellos que necesitan emplear la infraestructura de terceros[11]. Además, el aumento del consumo energético ha producido cambios significativos sobre el medio ambiente a nivel global, lo que repercute negativamente en todo el mundo[12]. Este problema energético de los sistemas *cloud* viene dado principalmente por dos aspectos: i) los centros de datos operan constantemente por debajo de su capacidad máxima, provocando que haya máquinas ociosas que es más rentable apagar pero que cuando sean necesarias haya que activar, siendo este el momento en el que más energía consumen[13]; y ii) las aplicaciones que se ejecutan sobre estos centros de datos, generalmente no tienen en cuenta el consumo energético[14][15].

Existen distintos factores que motivan el interés por detectar y reducir las principales causas del consumo energético desmedido, tales como la reducción de la huella de carbono[16], el ahorro en las facturas de electricidad de tecnologías de la información[17] y el aumento del tiempo de vida de los algunos dispositivos[18]. De esta forma, la informática sostenible, o *Green Computing*[19], se ha convertido en modelo de inspiración de otras iniciativas como es Green Grid[20]. Esta iniciativa es un consorcio industrial sin ánimo de lucro de usuarios finales, legisladores, proveedores de tecnología, arquitectos de instalaciones y empresas de servicios públicos que colaboran para mejorar la eficiencia de los recursos

de los centros de datos. Esta tendencia hacia la informática sostenible también se ve reflejada en la lista Green500[21], una lista que recoge los 500 supercomputadores más eficientes energéticamente del mundo.

3.2. Algoritmos de Optimización

Los algoritmos de optimización se pueden clasificar principalmente en dos categorías: *algoritmos deterministas* y *algoritmos estocásticos*[22]. Los algoritmos deterministas son aquellos que están puramente determinados por sus entradas y donde no hay aleatoriedad involucrada en el modelo, por lo que siempre obtendrán el mismo resultado dadas las mismas entradas. Un ejemplo de algoritmo determinista es la búsqueda del máximo en un vector de números, pues si siempre proporcionamos el mismo vector de números la salida no sufrirá variaciones. Por otro lado, los algoritmos estocásticos poseen aleatoriedad, es decir, pueden obtener distintas soluciones dada la misma entrada. Por ejemplo, los primeros GPS utilizaban algoritmos no deterministas para planificar las rutas y esto daba como resultado que proporcionando los mismos puntos de inicio y fin, se obtuvieran rutas distintas en cada consulta. Debido a esta característica, los algoritmos estocásticos se convirtieron en una herramienta importante para una amplia gama de problemas de optimización. A su vez, los algoritmos estocásticos se clasifican además en dos tipos: algoritmos heurísticos y metaheurísticos[23]. Los algoritmos heurísticos siguen una estrategia "ensayo y error" que no garantiza hallar el óptimo global[24]. Por otro lado, los algoritmos metaheurísticos son el desarrollo posterior de los algoritmos heurísticos que combinan principalmente la inteligencia de la búsqueda local y el procedimiento de aleatoriedad[25], siendo capaces de hallar el óptimo global de forma más eficiente que los heurísticos. Los algoritmos metaheurísticos se subdividen siguiendo la analogía de procesos naturales como la biología, la química, la física o el medio ambiente. Esta taxonomía se puede observar en la Figura 3.1.

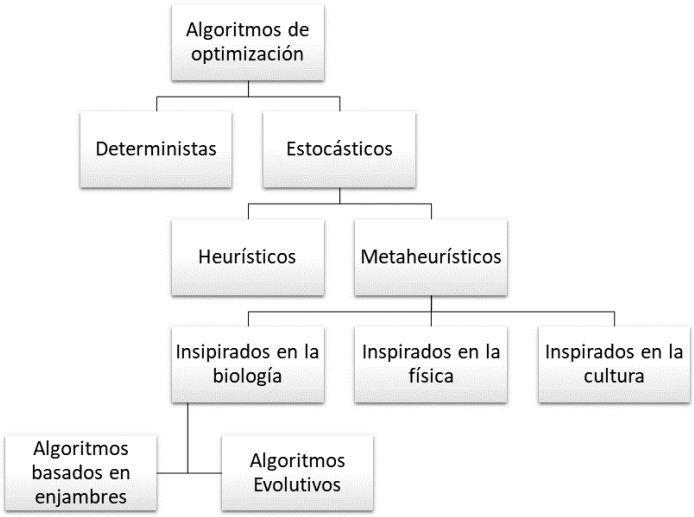


Figura 3.1: Taxonomía de los algoritmos de optimización

En este trabajo nos centramos en los algoritmos genéticos, pertenecientes a los algoritmos evolutivos, pues presentan las siguientes ventajas: no necesitan conocimientos específicos sobre el problema que intentan resolver; operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales; y utilizados para problemas de optimización resultan menos afectados por los máximos y mínimos locales que las técnicas tradicionales.

3.2.1. Algoritmos Genéticos

En 1975 Jhon Holland sentó las bases para lo que hoy en día conocemos como algoritmos genéticos (GAs, por sus siglas en inglés)[26]. Un GA se puede definir como un método de búsqueda cuyo objetivo es imitar la evolución biológica de Darwin para la resolución de problemas. Esto se consigue partiendo de una población inicial de la cual se seleccionan los individuos más capacitados, para luego reproducirlos y mutarlos para obtener, de manera sucesiva, generaciones de individuos mejor adaptados al problema que las generaciones previas. A continuación, se describen las distintas fases que lleva a cabo un GA, ilustrado en la Figura 3.2.

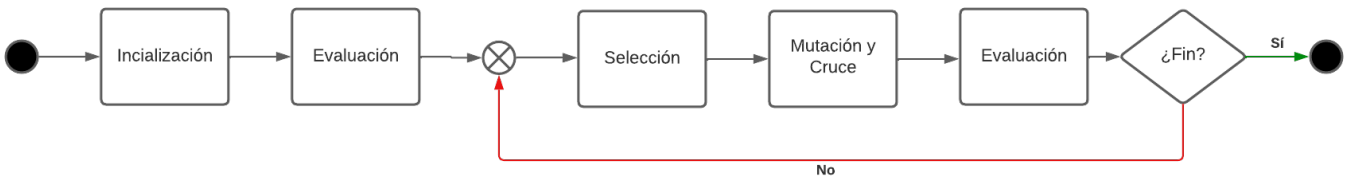


Figura 3.2: Fases de un algoritmo genético

Inicialización: El algoritmo toma como entrada un conjunto de individuos llamado población inicial, el cual suele ser proporcionado por el usuario. Cada individuo es una solución al problema a resolver por el algoritmo y está representado por un conjunto de variables conocidas como genes. Estos genes están unidos en forma de cadena, es decir, dispuestos de forma consecutiva uno tras otro, para así formar un cromosoma: la solución. Por norma general los genes están representados en formato binario por lo que cada gen se encuentra codificado en cada cromosoma. La Figura 3.3 ilustra los distintos componentes de la población de un algoritmo genético: el gen, mostrado en la zona azul; el conjunto de genes, el cromosoma, enmarcado en amarillo; la población es el conjunto de cromosomas, enmarcado en rojo.

Evaluación: Un aspecto clave en los GAs es la función de evaluación de una solución, que determinará la calidad de esta. Para ello, la función de aptitud se define para el problema en concreto, y se encargará de asignar un valor numérico para representar la calidad de la solución hallada.

Selección: Durante esta fase se seleccionan los individuos que mejor valor han obtenido en la función de aptitud para que estos puedan preservarse en la siguiente generación. Tras este proceso de selección,

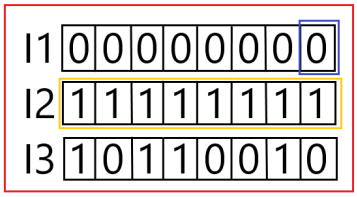


Figura 3.3: Individuos y población de un algoritmo genético

se escogen pares de individuos en función de sus valores para que estos actúen como progenitores, es decir, serán los encargados de propagar sus genes a las siguientes generaciones mediante mutación y cruce.

Mutación y cruce: La mutación y el cruce de individuos es la parte más relevante dentro de un algoritmo genético. Esencialmente, la mutación consiste en cambiar el valor de algunos genes de los nuevos individuos para añadir aleatoriedad al proceso de generación de los mismos. Este proceso se realiza con una probabilidad de mutación variable, definida por el usuario, que ayuda a mantener la diversidad en la población y previene de convergencias prematuras. El proceso de mutación se muestra en la Figura 3.4, de tal forma que el cromosoma I4 (ver Figura 3.4.a) es el cromosoma inicial, y el cromosoma I6 (ver Figura 3.4.b) es el resultado de aplicar el proceso de mutación sobre el I4.

Por otro lado, la función de cruce escogerá aleatoriamente un punto de cruce, de forma que el punto de cruce divide a cada individuo en dos partes. A continuación, se intercambia una de las partes entre los individuos, sin modificar el valor de los genes ni su posición en la cadena. En la Figura 3.5 queda reflejado de manera gráfica este proceso. Seleccionando aleatoriamente dos individuos de la población (ver Figura 3.5.a), se selecciona un punto de cruce de forma aleatoria. Una vez que este punto de cruce ha sido seleccionado, se intercambia una de las partes de cada individuo desde el punto de cruce de ambos (ver Figura 3.5.b), de tal forma que, la parte derecha del individuo I1 del ejemplo pasa a formar un nuevo cromosoma con la parte izquierda del punto de cruce del individuo I2, y viceversa.

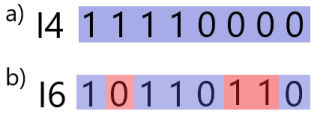


Figura 3.4: Proceso de mutación de un cromosoma

3.3. Metamorphic Testing

Los métodos tradicionales de *testing* requieren comprobar la conformidad entre las entradas y salidas del sistema bajo estudio, es decir, si para determinadas entradas, las salidas correspondientes son las

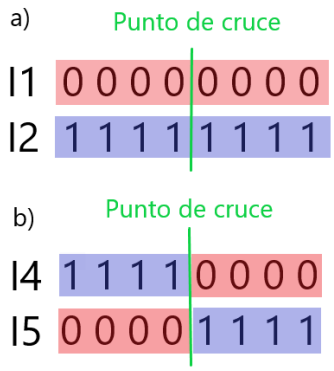


Figura 3.5: Proceso de cruce de dos cromosomas

esperadas. Sin embargo, esto supone dos problemas fundamentales. El primero de ellos es conocido como *el problema del oráculo* (*oracle problem*)[27]. En *testing*, un oráculo es un mecanismo utilizado para distinguir entre comportamientos correctos e incorrectos del sistema bajo estudio. Desafortunadamente, en muchas situaciones no está disponible o su aplicación es demasiado costosa computacionalmente, por lo que deben utilizarse enfoques alternativos. El segundo problema que afronta el *testing* convencional es el *problema del conjunto de tests fiable* (*reliable test set problem*)[28], que consiste en proporcionar un conjunto de pruebas apropiado para determinar la validez de un sistema. Normalmente no es viable ejecutar todos los casos de prueba posibles por el coste computacional que ello conlleva, por lo que se debe seleccionar un subconjunto de tests que sea capaz de analizar adecuadamente la corrección del sistema. Es en la selección de este subconjunto donde yace el problema, ya que si un conjunto de pruebas no es lo suficientemente adecuado, puede no contemplar ciertos casos que generen comportamientos incorrectos del sistema.

Para aliviar estos problemas, en este trabajo se ha decidido utilizar *metamorphic testing* (MT)[29][30]. El principio de MT consiste en modelar las propiedades del sistema bajo prueba como relaciones metamórficas (MRs, de sus siglas en inglés). Una MR se define como una formula $i(MR) \rightarrow o(MR)$, donde $i(MR)$ representa a la relación entre los parámetros de entrada de dos casos de prueba, y $o(MR)$ muestra a la relación que debe cumplir la salida obtenida en los tests.

Cabe destacar que para llevar a cabo el proceso de MT, el usuario debe proporcionar un conjunto de pruebas inicial (en inglés, *source test cases*). A partir de este conjunto, se deben generar casos de prueba que cumplan las MRs, llamados *follow-up test cases*. Así, $i(MR)$ y $o(MR)$ relacionan las entradas y salidas de un *source test case* y un *follow-up test case*, respectivamente.

Para ejemplificar, se define E1 y E2 como las entradas proporcionadas al sistema, y S1 y S2 como las salidas producidas al ejecutar el test con las entradas E1 y E2 respectivamente. La idea esencial es que en lugar de verificar la salida S1 producida al ejecutar el test con E1, se verifica con una segunda entrada E2 (*follow-up test case*), y verifica que S1 y S2 cumplan la relación según lo especificado por la

MR. Una ventaja qdue ofrece MT, al contrario que otras muchas técnicas, es que permite ser aplicado tanto para la verificación de los resultados como para la generación de casos de prueba, resolviendo así los dos problemas presentados previamente[31 32 33].

Un ejemplo de aplicación de MT es la función trigonométrica *coseno*. Cualquier implementación de esta función es una aproximación, donde el proceso de verificación de una salida esperada para una entrada dada puede ser complejo y propenso a errores. En este caso, se define una MR que describe una propiedad que debe cumplir la función *coseno*, como por ejemplo *cos(x) = cos(-x)*. Una vez definida la MR se proporciona un *source test case* para que ejecute la función coseno. Para comprobar esta MR, después de probar $f_1 = cos(45)$, también se comprueba $f_2 = cos(-45)$ y se comprueba la propiedad anterior. Si la propiedad no se cumple, esto es indicativo de que el sistema contiene un error.

Capítulo 4

Estado del arte

En este capítulo se presenta un análisis de las técnicas y estudios realizados en los campos de *optimización con algoritmos multiobjetivo* (Sección 4.1), simulación de *sistemas distribuidos* (Sección 4.2), *metamorphic testing* (Sección 4.3) y GAs en arquitecturas *cloud* y HPC (Sección 4.4).

4.1. Problemas de optimización con algoritmos multiobjetivo

Durante las últimas décadas, los problemas de optimización han sido ampliamente estudiados tanto en la comunidad científica como en la industria, los cuales se enfocan en encontrar la mejor solución posible dentro de un conjunto de soluciones. Para ello, tratan de optimizar variables de decisión a través de funciones objetivo. En función del número de objetivos a optimizar, este tipo de problemas pueden clasificarse en dos categorías: problema de optimización de un solo objetivo y problema de optimización multiobjetivo (MOP por sus siglas en inglés). La principal dificultad que afrontan los MOP es que se deben utilizar simultáneamente dos o más funciones objetivo. Como ejemplo ilustrativo supongamos que queremos llegar del punto A al punto B de una ciudad. Un problema de un solo objetivo consistiría en encontrar la ruta más corta entre ambos puntos, mientras que un MOP consistiría encontrar la ruta más corta y que, además, pase por menos semáforos. Sumándose a esta dificultad, estas funciones objetivo tienden a ser contradictorias entre sí, es decir, una solución óptima para una función puede obtener una puntuación baja en otra de las funciones. Por lo tanto, en términos generales, es difícil para un MOP encontrar una solución que satisfaga todas las funciones objetivo. En esta situación, existe un conjunto de soluciones factibles que son óptimas para una de las funciones, e iguales o mejores que el resto de soluciones para el resto de funciones a optimizar.

Otra cuestión a afrontar en los MOP es cómo definir las soluciones. Desde el punto de vista matemático, no existe una única solución para los problemas multiobjetivo, sino que existe un conjunto de soluciones. En 1951, Koopmans[34] presentó por primera vez el concepto de conjunto de soluciones eficientes de Pareto, que describía la solución bajo la relación del orden parcial pero no del orden total.

Para afrontar las soluciones de los problemas multiobjetivo hay que definir los siguientes conceptos: variable de decisión de dominación, dominación de la función objetivo, solución óptima de Pareto, frente óptimo de Pareto y solución no dominada.

Se define como *variable de decisión de dominación*, sobre dos vectores en el espacio de variables de decisión a y b, aquella que para todos los elementos de b existe una solución mejor o igual en a y existe, al menos, una solución en a estrictamente mejor que en b. Encontramos además la *dominación de la función objetivo*, la cual, para dos vectores en el espacio de las funciones objetivo h y g, todos los elementos en g son mejores o iguales que en h y existe, al menos, un elemento estrictamente mejor en g, entonces decimos que g domina a h. La *solución óptima de Pareto* se da si el vector x cumple la condición de que no existe un v que pertenezca al espacio de soluciones tal que x sea dominado por v. Entonces se denomina a x solución óptima de Pareto global, o solución óptima. Todas las soluciones óptimas de Pareto globales constituyen un conjunto óptimo de Pareto global, expresado como PS^*. A su vez, el *frente óptimo de Pareto* es el conjunto óptimo de Pareto presente en el espacio de funciones objetivo representado como $PF^* = \{f(x^*)|x^* \in PS^*\}$. Por último, la *solución no dominada* es la solución óptima en cada generación de población de evolución.

Los métodos de optimización multiobjetivo se enfocan en dos aspectos fundamentales: convergencia hacia soluciones que estén lo más cerca posible del frente óptimo de Pareto y diversidad de soluciones sobre el espacio de las mismas. Después de encontrar el conjunto óptimo de Pareto, los responsables de la toma de decisiones deben seleccionar entre estas soluciones, la solución final de acuerdo con los problemas de optimización concretos. En la Sección 4.1.1 se describen distintos algoritmos enfocados en la resolución de problemas de optimización multiobjetivo, o algoritmos multiobjetivo, y en la Sección 4.1.2 se presentan los algoritmos genéticos multiobjetivo para la solución de este tipo de problemas.

4.1.1. Algoritmos multiobjetivo

Los algoritmos multiobjetivo son aquellos enfocados a la resolución de problemas en los que se tiene en cuenta más de un parámetro para llegar a la solución. Un enfoque para afrontar estos problemas es determinar un conjunto óptimo de Pareto, es decir, un conjunto de soluciones que no son dominadas entre sí. De esta forma, al pasar de una solución de Pareto a otra siempre hay una cierta cantidad de sacrificio en uno de los objetivos para lograr una ganancia en los otros.

Sobre los métodos de optimización encontramos otra manera de clasificarlos a la expuesta en la Figura 3.1. Los métodos de optimización se pueden clasificar como: *métodos analíticos*[35] y *métodos numéricos*[36], tal y como se muestra en la Figura 4.1. El método analítico representa soluciones basadas en fórmulas matemáticas, en las que se definen variables de entrada para el cálculo de una o más variables

de salida. Sin embargo, este método también es estricto con las características del problema, las cuales muchos problemas realistas no podrían cumplir. Por ejemplo, las ecuaciones sobre hechos relacionados con la física generalmente se plantean sobre escenarios relativamente idealizados comparados a lo que se observa en la realidad. Por ello, los resultados obtenidos por estas ecuaciones pueden presentar una desviación significativa en relación al resultado real y no proporcionar un método de comprensión detallado del comportamiento del objeto bajo estudio.

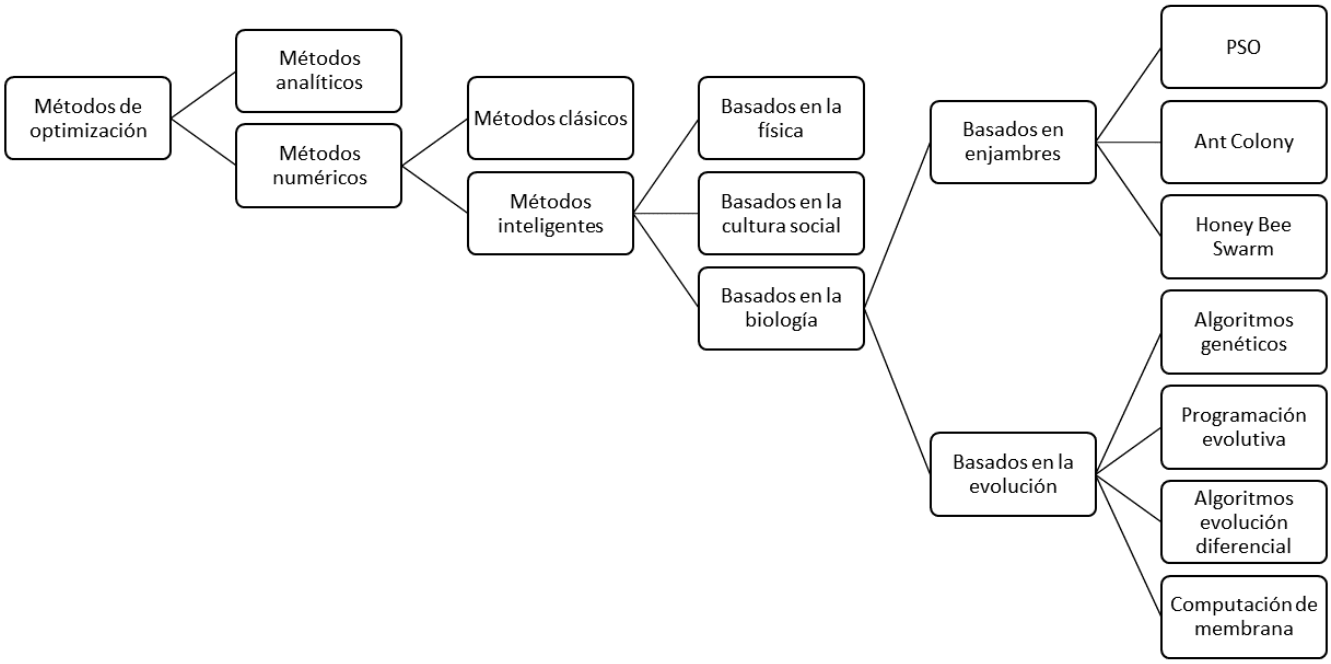


Figura 4.1: Métodos de optimización

Por otra parte, el método numérico está diseñado con fórmulas de iteración aplicadas para obtener la solución aproximada. Este método únicamente necesita variables de decisión definidas y variables objetivo con retroalimentación de problemas de optimización. Entre los métodos numéricos, se encuentran los *métodos clásicos*[37] y los *métodos inteligentes*[38]. Los métodos clásicos están orientados a problemas de optimización de un sólo objetivo. Estos métodos son muy eficientes a la hora de llevar a cabo la búsqueda y poseen una alta velocidad de convergencia, ya que generalmente comienzan con un punto inicial dado y calculan el siguiente punto de la iteración a partir de la información obtenida en previas iteraciones y el gradiente calculado de la función a optimizar. Sin embargo, estos métodos encuentran dificultades para resolver problemas cuyo gradiente no se puede calcular o tiene un coste computacional muy alto. Cuando se aplican para resolver MOP, los métodos numéricos clásicos encuentran dificultades con los óptimos locales, agravándose este problema debido a la facilidad que tienen para encontrar uno. Otro problema añadido, es que estos métodos solo encuentran una solución en cada iteración, y la preci-

sión de esta variará en función de los valores iniciales. Sin embargo, los métodos numéricos inteligentes, tratados como algoritmos de búsqueda, están inspirados en distintos comportamientos, reacciones y mecanismos de comunicación que se encuentran en la naturaleza. Estos algoritmos se dividen en tres categorías[39]: algoritmos inspirados en la física[40], algoritmos inspirados en la cultura social[41] y algoritmos inspirados en la biología[42]. A su vez, los algoritmos inspirados en la biología están subdivididos en dos grandes familias: algoritmos basados en evolución[43], o algoritmos evolutivos, y algoritmos basados en enjambres[44].

Los algoritmos basados en la evolución son métodos de búsqueda que pretenden imitar la supervivencia del individuo más apto de los ecosistemas naturales. Este tipo de algoritmos tiene una gran adaptabilidad y auto-organización, y en ellos se encuentran incluidos otros campos ampliamente conocidos como son la programación evolutiva[45], las estrategias evolutivas[46], los algoritmos genéticos[47], algoritmos de evolución diferencial[48], algoritmos de búsqueda armónica[49], la computación de membrana[50], etc. Por otro lado, los algoritmos basados en enjambres se inspiran en la naturaleza social[51] y modelan el comportamiento colectivo de las poblaciones, como abejas melíferas[52], colonias de hormigas[53], etc. Estos agentes (individuos del enjambre), cooperan entre sí para buscar alimento, necesario para su supervivencia, y también mantenerse a salvo de otros agentes.

A continuación se referencian distintos trabajos y autores que utilizan los algoritmos mencionados previamente.

Xue et al.[54] presentan el primer estudio sobre el algoritmo PSO[55], *Particle Swarm Optimization*, un algoritmo basado en enjambres, para la selección de características, que consiste en generar un frente de Pareto de soluciones no dominadas. En el estudio analizan dos algoritmos de selección de características multiobjetivo basados en PSO. El primer algoritmo introduce la idea de ordenación no dominante en PSO para abordar los problemas de selección de características. El segundo algoritmo aplica las ideas de aglomeración, mutación y dominación a PSO para buscar las soluciones del frente de Pareto. Tras la comparación de los algoritmos, los resultados experimentales muestran que ambos pueden hacer evolucionar automáticamente un conjunto de soluciones no dominadas.

Dorigo et al.[56] presentan el algoritmo ACO, *Ant Colony Optimization*, un algoritmo basado en el comportamiento de colonias de hormigas. Una hormiga es un agente computacional simple que busca soluciones a un problema de optimización dado. Para aplicar un algoritmo ACO, es necesario convertir el problema de optimización en el problema de encontrar el camino más corto en un grafo ponderado. En el primer paso de cada iteración, cada hormiga construye una solución, es decir, el orden en el que deben seguirse las aristas del grafo. En el segundo paso, se comparan los caminos encontrados por las distintas hormigas. El último paso consiste en actualizar los niveles de feromona en cada arista que

indican el número de hormigas que han pasado por una arista, y por lo tanto lo buena que es esta arista para la solución.

Shi et al.[57] proponen un nuevo método de optimización multiobjetivo de asignación dinámica de recursos para la distribución estable de máquinas virtuales. Combinando el estado actual y los datos previstos para el futuro de cada aplicación, el coste de la reubicación de la máquina virtual y la estabilidad del nuevo estado de la máquina virtual se consideran integralmente.

Hosseinzadeh et al.[58] presentan un artículo que se centra en el contexto de optimización metaheurística de múltiples objetivos y presentan un estudio completo y una descripción general de los enfoques de programación multiobjetivo diseñados para varios entornos de computación en sistemas *cloud*. Además, proporcionan una comparación de los esquemas de programación multiobjetivo, presentando las direcciones de investigación futuras.

Usando un algoritmo basado en PSO, Yassa et al.[59] presentan un estudio en el que proponen un nuevo enfoque para la planificación del flujo de trabajo multiobjetivo en sistemas *cloud* y presentan el algoritmo PSO híbrido para optimizar el rendimiento de este flujo. La propuesta se basa en la técnica Dynamic Voltage and Frequency Scaling (DVFS) para minimizar el consumo de energía. Esta técnica permite que los procesadores operen en diferentes niveles de voltaje con el inconveniente de perder frecuencia de reloj. Este voltaje múltiple implica un compromiso entre los tiempos de ejecución y la energía. Los resultados de la simulación con aplicaciones científicas sintéticas y aplicaciones ejecutadas sobre entornos de producción destacan el sólido rendimiento del enfoque propuesto.

En la literatura se encuentran más variaciones de los distintos algoritmos y técnicas presentados anteriormente. Sin embargo, para este trabajo, además de las ventajas expuestas al final de la Sección 3.2, se ha decidido emplear los algoritmos genéticos multiobjetivo (MOGAs por sus siglas en inglés), mostrados en la Sección 4.1.2, debido a que son paralelizables, proporcionan un mayor conjunto de espacio de soluciones y requieren menos información previa.

4.1.2. Algoritmos Genéticos Multiobjetivo

La principal dificultad de los GAs de un solo objetivo es la falta de aplicabilidad a situaciones reales dentro de la comunidad científica y el mundo industrial, ya que la aplicación de una optimización dependiente de un sólo parámetro puede considerarse residual. Para paliar estas dificultades surgen los MOGAs. En 1985 David Schaffer propone el algoritmo llamado *Vector Evaluated genetic algorithm* (o VEGA)[60]. Schaffer presenta una extensión lógica de un algoritmo genético clásico a un enfoque multiobjetivo. Desde entonces, se han implementado a lo largo de los años distintos algoritmos mediante distintos enfoques y aplicaciones. Entre estos algoritmos cabe mencionar Weight Based Gene-

tic Algorithm (WBGA)[61], Multi-objective Optimization Genetic Algorithm (MOGA)[62], Nondominated Sorted Genetic Algorithm (NSGA)[63], Niched Pareto Genetic Algorithm (NPGA)[64], Random Weight Genetic Algorithm (RWGA)[65], Strength Pareto Evolutionary Algorithm (SPEA)[66], Pareto Envelope-based Selection Algoritm (PESA)[67], Pareto Archived Evolution Strategy (PAES)[68][69], Rank-Density Based Genetic Algorithm (RDGA)[70] y Dynamic Multi-objective Evolutionary Algorithm (DMOEA)[71]. A partir de este punto los algoritmos propuestos aparecen como mejoras y adaptaciones a propuestas anteriores, de los que encontramos PESA-II Region-Based[72], NSGA-II[73], SPEA2[74] y NSGA-III[75][76]. Hay que tener en cuenta que en la literatura se encuentran muchas variaciones de los distintos MOGAs, y por lo tanto estos GAs citados son algoritmos bien conocidos, han sido utilizados en multitud de aplicaciones, y cuyo rendimiento ha sido probado en varios estudios comparativos[77][78][66][79][80][81].

La Tabla 4.1 resume las principales características de cada uno de los algoritmos presentados anteriormente. En la tabla se muestran los algoritmos ordenados por el año de propuesta, indicado en la primera columna. A continuación, las tres siguientes columnas, etiquetadas como "AG", "Elitista" y "Comp." indican el nombre del algoritmo, si se engloba dentro de los algoritmos elitistas, es decir, si el algoritmo usa algún mecanismo para propagar únicamente las características de las mejores soluciones, y el orden de complejidad computacional, respectivamente. Las tres siguientes columnas, etiquetadas como "PE", "Método aptitud" y "Método diversidad" indican si el algoritmo hace uso de una población externa, es decir, un fichero externo de individuos que se utilizan para introducir cambios a la población de la generación, el método de aptitud que utiliza el algoritmo y el mecanismo de diversidad, respectivamente. Las dos últimas columnas indican, respectivamente, las ventajas y desventajas de cada uno de los algoritmos recogidos en la tabla.

El primer MOGA propuesto, como vemos en la tabla 4.1, fue VEGA[60]. VEGA es un algoritmo de optimización multiobjetivo que divide una población en subpoblaciones basadas en objetivos específicos, asignando valores de aptitud según la evaluación de cada objetivo. A través de un proceso de selección, similar al de un GA tradicional, las soluciones se eligen para llevar a cabo cruces y mutaciones, generando nuevas soluciones en cada iteración. El enfoque apunta a obtener soluciones no dominadas que logren un equilibrio entre diferentes objetivos. Sobre este algoritmo existe una variante que emplea la alternancia de objetivos y se centra en seleccionar una función objetivo en cada iteración, lo que puede llevar a soluciones unilaterales en objetivos individuales.

WBGA[61] asigna a cada solución x_i un peso para cada función objetivo a optimizar para así formar un vector de pesos diferentes $w_i = w_1, w_2, ..., w_k$ en el cálculo de la función objetivo. El vector de peso w_i está integrado dentro del cromosoma de la solución x_i. Por lo tanto, se pueden buscar múltiples soluciones simultáneamente en una sóla ejecución. Además, los vectores de peso se pueden ajustar para

Año	AG	Elitista	Comp.	PE	Método aptitud	Método diversidad	Ventajas	Desventajas
1985	VEGA	✗		✗	Cada subpoblación se evalúa con respecto a un objetivo diferente a cumplir.	No tiene.	Primer MOGA implementado, implementación directa, ampliamente estudiado y testeado.	Tiende a converger hacia los extremos de cada objetivo.
1992	WBGA	✗		✗	Peso medio de los objetivos normalizados.	Niching mediante pesos predefinidos.	Extensión simplificada de un algoritmo genético simple.	Encuentra dificultades a la hora de objetivos en espacios de función no convexas.
1993	MOGA	✗	$O(G_m N^2)$	✗	Ranking de pareto.	Da más peso a las soluciones que se encuentran en zona escasamente pobladas.	Extensión simplificada de un algoritmo genético simple.	Lento en converger, además tiene problemas asociados con el parámetro del tamaño del nicho.
1994	NSGA	✗	$O(MN^3)$	✗	Ranking basado en el orden de no-dominación.	Da más peso a las soluciones que se encuentran en zona escasamente pobladas.	Rápido en converger.	Problemas con el tamaño del nicho.
1995	NPGA	✗	$O(G_m N^2)$	✗	Ranking basado en el orden de no-dominación.	Torneo de dominación de Pareto para ver qué solución es mejor. En caso de empate mediante Fitness sharing.	Selección simple mediante torneo.	Problemas con el tamaño del nicho, necesita un parámetro extra para el torneo.
1999	RWGA	✓		✓	Peso medio de los objetivos normalizados.	Valores aleatorios en función del número de elementos en la población.	Eficiente y fácil de implementar.	Encuentra dificultades a la hora de objetivos en espacios de función no convexas.
2000	SPEA	✓	$O(G_m N^2)$	✓	Ranking basado en el archivo de soluciones no dominadas.	Se agrupa la población en conjuntos homogéneos, con más de 1 elemento por conjunto, a continuación se van uniendo conjuntos hasta llegar al tamaño de población requerido para empezar el cómputo del conjunto de soluciones reducido.	Muy testado y no necesita parámetros a la hora de agrupar.	Tiene un algoritmo para agrupar las soluciones complejo.
2000	PESA	puro	$O(GNlog^m - 1AloglogA)$	✓	No tiene.	Formando una rejilla que divide el espacio de solución, cada solución es asignada a una casilla. Se seleccionan dos rejillas al azar y se favorece a la que tenga menos población para facilitar nuevas soluciones.	Fácil de implementar y eficiente.	El rendimiento depende del tamaño de las celdas que este depende del problema. Se necesita información previa del espacio objetivo.
2001	PAES	✓	$O(GNlog^m - 1AloglogA)$	✓	Dominancia de Pareto para reemplazar a los padres si la salida domina.	Se evalúa cada cromosoma para determinar si es una solución válida. Si la solución actual domina al candidato, se conserva. De lo contrario, se compara el candidato con la población de referencia. En caso de empate, se elige la solución en la región menos poblada del espacio de soluciones.	Estrategia de escalada de la colina con mutaciones aleatorias, fácil de implementar y computacionalmente eficiente.	No tiene un enfoque basado en población y el rendimiento depende del tamaño de celda utilizado.
2001	PESA-II	✓	$O(GmNA)$	✓	Similar a PESA	se realiza de forma similar a PESA pero se lleva a cabo sobre una hiperbox, en lugar de un individuo y una vez se selecciona un hiperbox, y el individuo resultante elegido para las operaciones genéticas se selecciona aleatoriamente.	Mantiene la diversidad en la población y al trabajar con hyperboxes las soluciones se seleccionan aleatoriamente para los procesos de cruce y mutación evitando la convergencia a puntos fuera del frente de Pareto.	Encontrar hyperboxes no es trivial, y dependiendo de la plataforma y de las estructuras de datos utilizadas podrían no actualizarse bien las hyperboxes de cada generación.
2002	NSGA-II	✓	$O(Nlog(N)^{(M-2)})$	✗	Similar a NSGA pero con un fichero con las soluciones no-dominadas hasta ese momento.	Se escoge la solución con menos rango, si el este es igual entonces se escoje la solución con menos soluciones dentro del cuboide.	Un solo parámetro (n) tamaño población. Bien testado y eficiente.	La distancia de agrupamiento sólo funciona en el la función objetivo.
2002	SPEA2	✓	$O(G_m(N+A)^2)$	✓	Similar a SPEA pero ahora tiene en cuenta las tantos las non-dominated como las dominated junto con el factor de los k-esimos más cercanos.	La solución que tienes más soluciones próximas es eliminada del archivo, en caso de empate se busca vecinos en segundo orden y así sucesivamente.	Mejora SPEA y se asegura de conservar los puntos extremos.	Es computacionalmente costoso calculando la aptitud y la densidad.
2003	RDGA	✓	$O(N^3)$	✓	Reduce los problemas MO a dos campos teniendo en cuenta la aptitud de la población así como la densidad de esta.	Basado en la densidad de las celda, añaden también la "forbbiden region"para añadir otro sesgo que evite la descendencia con más rango al mutar la población hacia una menos densidad.	Las celdas se van actualizando dinámicamente, es robusto con respecto al número de objetivos.	Difícil de implementar con respecto a otros.
2003	DMOEA	✓	$O(N)$	✓	En función de cada una de las celdas formadas les asigna un ranking mediante el rango y la densidad.	Reduce la población manteniendo la diversidad basándose en el rango y la densidad de cada población en cada iteración.	Incluye técnicas eficientes para actualizar la densidad de las celdas, tiene enfoques adaptadas a parámetros de algoritmos genéticos.	Difícil de implementar con respecto a otros.
2014	NSGA-III	✓	$O(N^2 log^{M-2} N)$	✓	Evalua cuánto contribuye una solución a la expansión del espacio de soluciones no dominadas	Mantiene la diversidad mediante *benchmarks* basados que se encuentran en la intersección entre el óptimo de Pareto con cada línea de referencia.	Bueno manteniendo la diversidad.	Convergencia pobre en muchas ocasiones, difícil de implementar.

Tabla 4.1: Principales algoritmos genéticos estudiados

promover la diversidad de la población.

MOGA[62] es el primer algoritmo genético multiobjetivo en utilizar explícitamente las técnicas de clasificación y *niching*[1], basadas en la técnica de Pareto, para facilitar la búsqueda hacia el frente de Pareto donde se encuentran las mejores soluciones posibles dentro del espacio de búsqueda, mientras se mantiene la diversidad en la población. Por lo tanto, es un buen ejemplo para demostrar cómo la clasificación basada en frentes de Pareto y la función de aptitud pueden integrarse en un algoritmo genético multiobjetivo.

El algoritmo NSGA posee tres versiones diferentes. En la primera propuesta[63], NSGA, se caracteriza por dos aspectos esenciales: la clasificación de la población se realiza en función de frentes no dominados, es decir, no se tiene en cuenta el comportamiento para cada una de las distintas funciones objetivo; y la diversidad en la población se mantiene mediante una distribución de los puntos no dominados a través de *niching*, el valor de niching se calcula contando el número de puntos de la población situados a una cierta distancia de un individuo, seleccionándose el individuo con menor valor. En su segunda versión[73], NSGA-II, mejora el la propuesta original añadiendo un enfoque elitista al algoritmo e introduce el operador de selección basado en torneos, además mejora la diversidad de las soluciones y aborda los problemas de convergencia prematura. Otra mejora que incluye es un selector que crea una nueva población mezclando soluciones progenitoras e hijas, seleccionando las mejores en función de su valor de aptitud y propagándolas hasta conseguir una población de tamaño N. Por último, en su tercera versión[75][76], NSGA-III, el algoritmo pasa a estar enfocado en problemas de tres o mas objetivos. El algoritmo ahora está basado en puntos de referencia[82], es decir, se generan una serie de puntos de en el espacio objetivo que poseen buenas prestaciones en convergencia y distribución en función de la población actual, sustituyendo de esta forma el criterio de selección de individuos de las versiones previas, basado en la distancia entre soluciones no dominadas. A continuación, para discriminar entre las soluciones no dominadas se utiliza una función de aptitud, cuyo valor indica la relevancia de una solución para aproximar un punto de referencia.

NPGA[64] define la dominancia de una solución con respecto a la violación de restricciones de dos soluciones. Se dice que la solución inviable x restringe o domina a la solución y, si x tiene las mismas o menos violaciones que la solución y para cada restricción del problema, y estrictamente menos infracciones para al menos una restricción. Un empate entre dos soluciones no dominadas por las restricciones se resuelve mediante la violación total de las restricciones de las soluciones.

RWGA[65] asigna aleatoriamente un peso normalizado para cada uno de los individuos de la población. El procedimiento de selección de individuos del algoritmo escoge los individuos para el cruce basado

[1]Clase de técnicas que intentan converger a más de una solución durante una sola ejecución.

en una suma ponderada de los pesos de múltiples funciones objetivo, donde los pesos asignados a las distintas funciones objetivo son proporcionados aleatoriamente para cada selección. Además, RWGA es un algoritmo elitista, es decir, pretende asegurar que los individuos más aptos de la población actual sobrevivan y continúen participando en el proceso evolutivo, pasando a la siguiente generación de manera intacta, sin cruzarse ni mutarse. La estrategia elitista en el algoritmo consiste en seleccionar un cierto número de individuos de un conjunto de posibles soluciones óptimas, no dominadas, y que se propagan a la próxima generación como individuos de élite.

SPEA cuenta con dos implementaciones diferentes. En su primera propuesta[66] utiliza un procedimiento de clasificación para asignar valores de aptitud elevados a soluciones no dominadas en regiones con menor número de individuos del espacio objetivo, consiguiendo así mantener la diversidad y evitar la convergencia en una región del espacio. En esta primera versión, una lista externa de un tamaño fijo almacena soluciones no dominadas que han sido estudiadas hasta el momento durante la búsqueda, de esta forma mantiene una población elitista para ser usada en la siguiente generación. En su segunda versión[74], SPEA2 realiza una mejora en la asignación de aptitud de cada individuo, una implementación nueva para la técnica de estimación de densidad y un método mejorado de cruce de archivos. A la hora de asignar la aptitud de cada individuo, se le tendrán en cuenta las soluciones dominadas y no dominadas para evitar valores idénticos entre individuos. La técnica de estimación de densidad utilizada en SPEA2 es una adaptación del método del k ésimo vecino más cercano. La densidad es una función que para cada individuo i calcula las distancias a todos los individuos j, que se encuentran tanto en el archivo como en la población, y el resultado se almacena en un archivo externo para luego llevar a cabo el cruce de individuos. La operación de actualización del archivo en SPEA2 difiere de la de SPEA en dos aspectos: el número de individuos contenidos en el archivo es constante a lo largo del tiempo, y el método de cruce evita que se eliminen las soluciones en los límites del espacio.

PESA se encuentra en la literatura con dos versiones. La primera versión de PESA[67] propone, el uso de la misma técnica para mantener la diversidad en la población y para el proceso de selección, simplificando así la implementación del algoritmo. Para ello, divide el espacio objetivo en celdas homogéneas de tal forma que cada solución se hallará en una de ellas. Durante la selección de individuos, se toman dos cromosomas al azar de las soluciones calculadas anteriormente, las cuales se encuentran en el archivo de soluciones, y se elige la solución que posee una densidad menor, es decir, se selecciona aquella que tenga menos individuos en su celda. En caso de empate, la selección se deberá llevar a cabo de manera aleatoria. Al realizar la selección de individuos mediante esta técnica, el esfuerzo de la búsqueda se centra en zonas del frente de Pareto emergente que poseen poca representación en la población. Por otra parte, el archivo de soluciones tiene un tamaño limitado, por lo que al añadir un individuo se puede

superar el tamaño máximo de este, y por lo tanto hay que eliminar una solución de que ya se encuentre añadida. La elección se lleva a cabo calculando primero la densidad de celda máxima en la población, y eliminando un cromosoma arbitrario correspondiente. En su segunda versión, PESA-II Region-Based[72], se proporciona un nuevo método de selección basada en *hyperboxes*, cuadrados en el espacio objetivo. Este método de selección es más sensible a los cambios, asegurando una buena distribución de soluciones a lo largo del frente de Pareto. Para esto, en lugar de asignar una selección de aptitud a cada individuo, se aplica a cada una de estas *hyperboxes* que están compuestas por al menos un objetivo cercano al frente de Pareto. Una vez que se selecciona una celda, teniendo más posibilidades aquellas que se encuentran menos pobladas, las soluciones dentro de la celda se seleccionan aleatoriamente para participar en los procesos de cruce y mutación.

PAES[68] sigue una estrategia evolutiva $(1 + 1)$, lo que significa que un único padre genera una única descendencia mediante mutación, sin utilizar el cruce. La descendencia será descartada si es dominada por el padre o por cualquier solución del archivo de soluciones. En cambio, si el descendiente domina al padre, el hijo lo sustituirá en la población y se insertará en el archivo. Si el archivo se encuentra lleno, el padre y el hijo deberán compararse utilizando un algoritmo de cuadrícula adaptativa, que mide el número de soluciones que se encuentran dentro de una celda con tantas dimensiones como objetivos tenga el problema, y queda descartado aquel que tenga una mayor densidad de celda. La cuadrícula se adapta recursivamente para separar las soluciones sin necesidad de un parámetro de *niching*. La solución más diversa se mantendrá en la población y se introducirá en el archivo.

RDGA[70] introduce un nuevo método de clasificación denominado estrategia de clasificación automática acumulada, y un concepto de *región prohibida*. Esta región incluye todas las celdas que son dominadas por el padre seleccionado, para que no se propaguen a la siguiente generación, evitando así la pérdida de velocidad evolutiva y su influencia en la dirección de evolución de las soluciones. Por lo tanto, este algoritmo utiliza un enfoque de densidad basado en celdas, de manera que convierte un problema general con K-objetivos en un problema de optimización biobjetivo que trata de minimizar el valor de rango individual y la densidad de la población. La principal ventaja del enfoque de densidad basado en celdas es que se obtiene un mapa de densidad global del espacio de función objetivo como resultado del cálculo de densidad. Se puede orientar la búsqueda hacia regiones escasamente habitadas del espacio de la función objetivo en base a este mapa. Además, utiliza un método basado en el mapa de densidad global para mover soluciones desde áreas de alta densidad, hacia áreas de baja densidad. Otra ventaja es su eficiencia computacional en comparación con las técnicas de densidad basadas *niching* o en vecindades.

DMOEA[71] propone, de manera similar a RDGA, enfocar los MOP como problemas biobjetivos, de tal

forma que divide el espacio objetivo en celdas, tratando de minimizar el valor del rango de cada individuo y asignando un valor de densidad a cada individuo. Esta propuesta se caracteriza principalmente por conseguir una complejidad computacional de $O(N)$ al usar un esquema de estimación basado en el rango y la densidad de las celdas generadas. Además, mediante una compresión del espacio objetivo, es decir, reducir el espacio de soluciones a explorar para ajustarlo lo máximo posible al frente de Pareto, consigue reducir los valores de rango y densidad de los individuos. Otra característica es la capacidad de adaptar dinámicamente el tamaño de la población según el rango y la densidad de la población actual. Por último, cabe destacar la convergencia que lleva a cabo para alcanzar un tamaño de población deseado mediante la exploración de todas las hiperáreas no dominadas con un número fijo de puntos de Pareto.

Tras esta revisión podemos concluir que los algoritmos VEGA y WBGA cuentan con la ventaja de ser fáciles de implementar además de estar muy testeados, aunque por ser de los primeros no proporcionan las mismas herramientas de convergencia y por lo tanto, pueden no obtener las mejores soluciones. También, MOGA ha sido uno de los algoritmos más ampliamente utilizado, pues fue el primero que propuso el uso del frente de Pareto como técnica para resolver MOP. Dentro de los algoritmos que utilizan el frente de Pareto, podemos remarcar las siguientes características: PESA consigue unificar los métodos de selección y diversidad del algoritmo; NSGA-II y NSGA-III con su estrategia de ordenación de soluciones en frentes no dominados consiguen una alta eficacia, siendo NSGA-II mejor para MOPs con dos objetivos y NSGA-III para MOPs con tres o más objetivos[83]; DMOEA consigue un buen rendimiento por su complejidad lineal y una buena convergencia al transformar un MOP en un problema biobjetivo.

En términos de eficiencia se encuentra que DMOEA es el que menor complejidad computacional tiene pero tiene el inconveniente de ser difícil de implementar respecto a otros y no se encuentran estudios que lo utilicen con aplicaciones en entornos de producción. Sin embargo, NSGA-II que pese a ser más costoso computacionalmente se encuentra que ha sido utilizado para múltiples estudios[84][85][86][87] proporcionando soluciones óptimas y presentando un rendimiento óptimo en los problemas a los que se le ha sometido. Así pues, su siguiente versión, NSGA-III, presenta que para MOPs con más de tres objetivos posee un mayor rendimiento con un coste computacional similar.

Para este trabajo se han decidido implementar cuatro de estos algoritmos presentados. Debido a su extendido uso, y por ser el primero en emplear la clasificación en frentes de Pareto, se ha seleccionado el algoritmo MOGA para su implementación. Por otro lado, el algoritmo PAES es un algoritmo puramente elitista y aporta una implementación basada en celdas que puede ser interesante comprobar su desempeño. A su vez, el algoritmo NSGA-II ha demostrado un buen rendimiento a la hora de trabajar con MOPs de dos objetivos y es de los algoritmos más extendidos actualmente, por ello se ha escogido para este trabajo. Por último, el algoritmo SPEA2 es otro de los algoritmos más extendidos en la resolución

de MOPs en la actualidad, por ello es interesante comprobar su comportamiento para la optimización de sistemas *cloud*.

4.2. Simulación de Sistemas distribuidos

Tomando en consideración que el objetivo del trabajo está orientado a la optimización energética de sistemas *cloud* y computación de altas prestaciones (HPC, por sus siglas en inglés), se propone la utilización de técnicas de simulación de sistemas distribuidos para tratar de alcanzar este objetivo.

En las últimas décadas se ha despertado un gran interés por el ámbito de la simulación en diversos campos de aplicación tales como la mecánica cuántica[88], la física[89] o medicina[90]. Siguiendo la definición de Robert E. Shannon[91] el concepto de simulación se define como "el proceso de diseño y realización de un modelo de un sistema real con el fin de comprender el comportamiento del sistema y/o evaluar diversas estrategias para el funcionamiento del mismo".

Entre los motivos por los que la simulación se ha abierto hueco a la hora de representar el comportamiento de sistemas hay que destacar los siguientes:

- Ahorro de costes. La mayor parte de los simuladores poseen una licencia GPL[92], permitiendo que no sea necesaria una inversión para el uso de las herramientas requeridas. Además, las simulaciones se pueden ejecutar en ordenadores personales o, en caso de disponer de ellos, en pequeños grupos de ordenadores que permiten mejorar el rendimiento.

- No requieren hardware especializado. Por lo tanto, no es necesaria la arquitectura específica del sistema a simular, pudiendo utilizar cualquier sistema informático para ejecutar las simulaciones.

- La simulación permite un alto nivel de flexibilidad sobre los experimentos. Un sistema puede ser modelado modificando los parámetros de simulación. En general, esto se consigue mediante ficheros de texto o configurando los parámetros en una GUI. Aún así, a la hora de llevar a cabo experimentos sobre sistemas distribuidos reales puede requerir cambios en el hardware, implicando así más tiempo y esfuerzo.

- Los experimentos pueden ser fácilmente controlados y repetidos en entornos de simulación debido a que los parámetros de cada simulación son conocidos y están recogidos en los ficheros de configuración.

- Los modelos pueden ser escalados con mayor facilidad. Configurar parámetros para hacer que el sistema crezca es más sencillo y sin costes adicionales, al contrario que modificar una arquitectura real, que conlleva un gasto elevado tanto de tiempo como de recursos.

- Los modelos y los simuladores pueden ser compartidos con otros investigadores fácilmente, pues sólo es necesario compartir los ficheros de configuración los modelos pueden ser replicados.

Pese a las ventajas expuestas anteriormente, hay que tener en cuenta que también se presentan ciertos inconvenientes. Diseñar sistemas *cloud* es una tarea compleja debido a la gran cantidad de componentes y subsistemas que forman parte de ellos. Es por esta gran cantidad de componentes que conforman el sistema que no existe hoy en día una propuesta que represente todos ellos de manera fidedigna. Debido a las ventajas previamente descritas y, teniendo en cuenta los inconvenientes, en este trabajo se propone la utilización de técnicas de optimización en entornos simulados.

En el campo de simulación de sistemas distribuidos hay múltiples herramientas, cada una enfocada a un componente, como las redes de comunicación, la CPU o la memoria, o aspecto del sistema, como el rendimiento, el consumo o el coste. Puesto que el objetivo de este trabajo es optimizar sistemas distribuidos complejos, es necesario tomar en consideración estos distintos aspectos a la hora de seleccionar el simulador con el que se llevarán a cabo las pruebas. Por ello, en esta sección se hará una revisión de los simuladores de sistemas distribuidos más relevantes que se encuentran actualmente.

En el ámbito de simulación enfocado en las redes de comunicación encontramos NS-2[93] y OMNeT++[94]. Estas herramientas se encargan en reproducir en detalle aspectos importantes de las redes de comunicación como son los protocolos de red, la latencia o la fragmentación. El inconveniente de estos simuladores es que carecen de los métodos necesarios para simular sistemas complejos de E/S, cómputo o virtualización.

Para cubrir estos inconvenientes, surgen herramientas para el modelado y la simulación de sistemas distribuidos. Basada en OMNeT++, se encuentra la plataforma SIMCAN[95]. SIMCAN es una plataforma de simulación, destacada por su alto grado de definición, cuyo objetivo es el modelado de arquitecturas HPC. Este simulador nos permite diseñar y modelar tanto herramientas como arquitecturas distribuidas con un rango de configuraciones amplio. Otra de las características de SIMCAN es el repositorio que posee que permite modelar distintos componentes con diferentes configuraciones, por ejemplo, las CPUs pueden modelarse con un número determinado de núcleos, así como seleccionar su velocidad. De esta forma, los usuarios pueden modelar sistemas distribuidos a gran escala, como son los clústeres HPC y los centros de datos utilizados para dar soporte a sistemas *cloud*.

Además de estas herramientas descritas previamente, se pueden encontrar simuladores enfocados en sistemas *grid*. Los sistemas *grid*[96] han sido ampliamente utilizados por la comunidad científica a principio de siglo. Para llevar a cabo distintas investigaciones en esta área, se desarrollaron un conjunto de simuladores entre los que se encuentran SimGrid[97], GangSim[98] y GridSim[99].

Como siguiente paso en el desarrollo de los simuladores de sistemas distribuidos, y como evolución

a los sistemas *grid*, surgen los simuladores de sistemas *cloud*. Algunos de los simuladores de sistemas *grid* previos, tales como SimGrid y GridSim, se utilizan como base para modelar y diseñar sistemas *cloud*, mientras que otros son desarrollados completamente de cero para este propósito. Los entornos de simulación que mejor se adaptan a los requisitos inherentes al modelado y la simulación de sistemas de *cloud computing* son CloudSim[100], EMUSIM[101], SimGrid[97], GDCSim[102], GroudSim[103], SimIC[104], MDCSim[105], iCanCloud[106] y GreenCloud[107].

CloudSim[100] es una herramienta de simulación utilizada para modelar centros de datos, máquinas virtuales, recursos informáticos y aplicaciones como el aprovisionamiento de recursos[108], así como el diseño[109] y ejecución de flujos de trabajo[110]. Una de las características clave de CloudSim es la posibilidad de incluir funcionalidades adicionales mediante extensiones como CloudSimStorage, una extensión que permite modelar el consumo de energía del sistema de almacenamiento[111]. Sin embargo, CloudSim tiene una serie de limitaciones. Por ejemplo, no permite modelar centros de datos complejos con varios modelos de aplicaciones y recursos heterogéneos. Por otro lado, tampoco permite modelar el uso compartido del ancho de banda en los enlaces de red. Además, al no contar con una GUI, para modelar entornos, los usuarios deben realizar el modelado y la configuración del sistema mediante programación, lo que resulta tedioso y propenso a fallos.

Basado en CloudSim y con el objetivo de solucionar alguna de las limitaciones que este presenta, surge EMUSIM. EMUSIM es un entorno integrado de emulación y simulación que se encarga evaluar el rendimiento de las aplicaciones de sistemas *cloud*. Para llevar a cabo la simulación se basa en CloudSim, mientras que para la parte de emulación se basa en AEF (*Automatic Emulation Framework*)[109]. EMUSIM aprovecha la información del comportamiento de la aplicación a través de la emulación para, a continuación, utilizarla para producir un modelo de simulación. Una ventaja que proporciona esta herramienta es que realiza la evaluación utilizando la información a la que tienen acceso los clientes de los proveedores generales de IaaS, sin necesidad de información adicional como el número de máquinas virtuales o sus ubicaciones. Sin embargo, presenta problemas de escalabilidad debido a las limitaciones de emulación de sistemas y el hardware necesario de estas. Por lo tanto, no es muy adecuado para generar cargas de trabajo grandes y similar a entornos de desarrollo o investigación.

Por otro lado, SimGrid[97] proporciona funcionalidades básicas a la hora de simular algoritmos y aplicaciones distribuidas, tales como *workstations* y entornos *grid*. Los recursos se modelan teniendo en cuenta distintos aspectos como latencia o tasa de servicio. Sin embargo, en las simulaciones no se incluyen detalles hardware tales como acceso a memoria principal o memoria caché, que pueden afectar al consumo de energía.

GDCSim[102] es una herramienta para analizar un centro de datos ecológico, así como las técnicas de

gestión de los recursos del sistema. El objetivo de GDCSim es proporcionar una herramienta de simulación de infraestructuras de centros de datos para aumentar el nivel de concienciación medioambiental de los centros de datos que operan en todo el mundo. Por otro lado, GDCSim posee el inconveniente de que no tiene en cuenta los experimentos paralelos, por lo que los resultados de los experimentos se ven afectados por la evaluación de aplicaciones a gran escala, ya que no puede ejecutar los experimentos a través de varias máquinas. Otro aspecto negativo es que no considera los aspectos de seguridad de la plataforma en la nube, por lo que no se puede diseñar un esquema de seguridad en términos de autenticación y autorización.

GroudSim[103] es un simulador basado en eventos que ofrece la posibilidad de registrar errores dentro de ciertos intervalos de tiempo, permitiendo así un mayor ajuste a escenarios reales. Este simulador posee un buen rendimiento sobre aplicaciones en entornos de investigación científica en *clouds* y *grids* y, gracias a esto, los desarrolladores pueden estudiar las propuestas en ambos entornos con un solo hilo de simulación. Además, utiliza el entorno ASKALON[112] para que los desarrolladores puedan importar experimentos que muestren aplicaciones que estén en uso fuera de la investigación sobre ese entorno. Sin embargo, GroudSim presenta solo conceptos básicos sobre las redes de comunicación, por lo que los desarrolladores no pueden configurar una red realista en función de la topología, el tamaño de los paquetes y la jerarquía. Además, sufre un decremento del rendimiento en aplicaciones a gran escala, por lo que no se pueden llevar a cabo experimentos con cientos de nodos.

SimIC[104] es un conjunto de herramientas de simulación basado en la plataforma de simulación de SimJava, que permite replicar un entorno en el que varios sistemas *cloud* colaboran entre sí para distribuir las solicitudes de servicio. SimIC logra interoperabilidad, flexibilidad y elasticidad del servicio a la vez que introduce heterogeneidad en la configuración de los distintos sistemas *cloud* que participan durante la simulación. Un inconveniente que tiene esta herramienta es que se centra menos en el consumo de energía del sistema, por lo que no permite estudiar las dificultades energéticas que enfrentan los sistemas *cloud*. Sumado a este inconveniente, no ofrece un modelo de red completo dificultando así la investigación de los enfoques de control de tráfico y congestión en la sistema bajo estudio.

MDCSim[105] es una herramienta de simulación de centros de datos multicapa[113] para diseñar y analizar sistemas a gran escala y, por lo tanto, permite el estudio el rendimiento de la aplicación en una plataforma escalable bajo diferentes cargas de red con diferentes configuraciones de nivel. Además, permite la simulación del consumo energético por lo que se pueden comparar políticas de eficiencia energética para entornos *cloud*. Una característica adicional de esta herramienta es que tiene una sobrecarga de simulación baja por lo que se pueden evaluar aplicaciones a gran escala variando la configuración para cada una de las capas que pueden conformar el sistema. Como inconvenientes sobre esta herramienta,

aparece que no es compatible con la política para la gestión de aplicaciones en sistemas *clouds* heterogéneos con varios dominios. Además, otro inconveniente que posee es que no es capaz de representar un modelo de red completo, por lo que no se pueden explorar todas las características de sistemas realistas, ni estudiar posibles soluciones a problemas que puedan encontrarse.

iCanCloud[106] es una plataforma de modelado y simulación de sistemas *cloud* construida a partir de la plataforma de OMNeT++[94]. Proporciona un conjunto de módulos diseñados para reproducir fielmente el comportamiento de distintos componentes del sistema. Una ventaja que ofrece iCanCloud es que puede ser utilizada en conjunto al *framework* de $E-mc^2$[7], incluyendo de esta forma mecanismos para realizar mediciones de consumo energético. Esta herramienta presenta dos inconvenientes: no proporciona un modelo completo de utilización de energía, por lo tanto no permite investigar de manera eficiente las técnicas de administración de energía; y no se centra en los factores de seguridad, por lo que no se puede estudiar un conjunto de mecanismos para la protección de datos y aplicaciones.

GreenCloud[107] es una extensión del simulador de redes NS-2[93], centrado en simular las comunicaciones entre procesos ejecutándose en un sistema *cloud* a nivel de paquete. Su principal enfoque estaba en la evaluación del rendimiento energético y la eficiencia de los centros de datos y las redes. De la misma forma que NS-2, GreenCloud está escrito en C++ y OTcl, siendo esto una desventaja, pues el uso de dos lenguajes diferentes dificulta la implementación de los experimentos. Además, presenta un inconveniente de escalabilidad ya que requiere un tiempo de simulación muy elevado, así como una gran cantidad de memoria. Junto a esto, otro inconveniente es que no proporciona un modelo de agregación de tráfico completo, por lo que no permite estudiar estrategias de control de congestión en los centros de datos de sistemas *cloud*.

En la Tabla 4.2 se recogen las características de estos simuladores. En la primera columna se encuentra el nombre del simulador y, a continuación, se muestra el lenguaje en el que ha sido desarrollado. Las dos siguientes columnas, etiquetadas como "Plataforma" y "GUI" respectivamente, hacen referencia a la plataforma que se ha usado para desarrollar el simulador, indica si el simulador posee interfaz gráfica, respectivamente. La columna etiquetada como "Redes" determina si el simulador es capaz de simular las redes de comunicación. En este caso, los simuladores SIMCAN, CloudSim, iCanCloud y GreenCloud están marcados como "completo" pues son capaces de modelar una red de comunicación completa, implementando también protocolos de red. Por otro lado, EMUSIM, SimGrid, SimIC y MDCSim se encuentran limitados en este aspecto, mientras que GDCSim y GroudSim no tienen implementada la simulación de redes de comunicación. Por último, las dos últimas columnas, etiquetadas "Ener." y "Coste", recogen dos de los aspectos en los que se enfoca este trabajo, indican si el simulador es capaz de simular el coste energético del sistema y si es capaz de medir el coste de producción del sistema,

respectivamente.

Debido a que este trabajo tiene como objetivo reducir el consumo energético de un sistema *cloud* los simuladores SIMCAN, GroudSim, SimIC, y iCanCloud quedan descartados por no poseer herramientas para el cálculo de este parámetro. De los simuladores restantes, EMUSIM, SimGrid, GDCSim, MDCSim tienen limitaciones para la simulación de los sistemas de red de los sistemas, pudiendo afectar esto a los resultados sobre el consumo energético. De esta forma, los simuladores que más se adecúan a este trabajo son CloudSim, CloudSimStorage y GreenCloud. Finalmente, para este trabajo se emplea el simulador CloudSimStorage, pues está basado en CloudSim que es uno de los simuladores más empleados en la investigación y permite calcular el consumo energético del los sistemas de almacenamiento.

Simulador	**Lenguaje**	**Plataforma**	**GUI**	**Redes**	**Ener.**	**Coste**
SIMCAN	C++	OMNeT++	✓	✓	✗	✗
CloudSim	Java	SimJava	✗	✓	✓	✓
CloudSimStorage	Java	CloudSim	✗	✓	✓	✓
EMUSIM	Java	CloudSim, AEF	✗	Limitado	✓	✓
SimGrid	C/C++	GPL	✗	Limitado	✓	✗
GDCSim	C, C++, Shell	Bluetoll	✗	✗	✓	✗
GroudSim	Java	-	✗	✗	✗	✓
SimIC	Java	SimJava	✗	Limitado	Limitado	✓
MDCSim	Java, C++	CSim	✗	Limitado	✓	✗
iCanCloud	C++	OMNeT++,SIMCAN	✓	✓	✗	✓
GreenCloud	C++, OTcl	NS-2	Limitada	✓	✓	✗

Tabla 4.2: Características de los simuladores

4.3. Metamorphic Testing

Metamorphic testing (MT) es una técnica de testing propuesta por Chen et al.[31] en 1998.Esta técnica en lugar de centrarse en cada resultado individual, analiza múltiples ejecuciones del programa y comprueba si las entradas y salidas de estas ejecuciones múltiples satisfacen ciertas relaciones metamórficas (MRs, por sus siglas en inglés), que son propiedades necesarias de la funcionalidad del programa previsto. Gracias a esto, consigue solucionar dos problemas del testing tradicional: el *oracle problem*[27] y el *reliable test set problem*[28], expuestos en la Sección 3.3.

MT tiene la ventaja de poder ser aplicado tanto para la generación de casos de prueba como para la verificación de resultados, consiguiendo así resolver los problemas asociados al testing expuestos anteriormente. Además, pese a que la identificación de MR puede ser una tarea complicada, el éxito de una MR que contemple correctamente una propiedad del sistema es alto[114]. Otros aspectos a tener en cuenta son la facilidad de automatización del proceso de testing dadas las MRs, pues los *source test cases* se pueden generar a través de los métodos de generación de prueba existentes, mientras

que los *follow-up test cases* se pueden construir a través de transformaciones de acuerdo con las MRs previamente definidas. A su vez, cabe destacar su bajo coste, debido a que los *follow-up test cases* se generan fácilmente a través de transformaciones de acuerdo con las MRs, aunque la verificación de los resultados de las pruebas implica comparar los resultados con las MRs, los gastos generales asociados son relativamente bajos en comparación al coste de la verificación de los resultados cuando existe el problema del oráculo.

Pese a haber sido propuesta hace más de dos décadas, no ha sido hasta los últimos años cuando MT ha empezado a cobrar protagonismo. En la literatura, se encuentra el estudio de Segura et al. [29] en el que llevan a cabo una revisión sistemática del estado del arte de MT, donde se reflejan los distintos campos de aplicación de esta técnica. Entre ellos pueden encontrarse propuestas donde los autores aplican MT en servicios web y aplicaciones[115], simulación y modelado[116], computación gráfica[117], sistemas empotrados[118], machine learning[119], variabilidad y decisión de soporte[120], bioinformática[121], componentes[122], programas numéricos[123] y compiladores[124]. Pese a la aplicación en campos tan dispares, a día de hoy aún está por explorar el potencial de esta técnica[125], y sobre todo en uno de los enfoques de este trabajo, los sistemas *cloud* y HPC.

Núñez y Hierons[126], en su primera propuesta, proponen una metodología semiautomática para probar y validar sistemas *cloud* mediante la integración te técnicas de simulación y MT. Dado un modelo *cloud* de entrada, los autores proponen un catálogo de MRs enfocadas en el rendimiento del sistema, en la funcionalidad de este y en el consumo energético del sistema. Para comprobar la adecuación de la metodología propuesta, los autores realizan una fase de experimentación donde dado un modelo *cloud* de entrada, se lleva a cabo la simulación del mismo y se analizan los resultados con el catálogo de reglas propuestas.

Posteriormente, Núñez et al. mejoran su anterior propuesta para el testing de *cloud* proponiendo TEA-Cloud[127]. El objetivo de este trabajo es proporcionar una metodología completa para ayudar a los usuarios a modelar las partes de software y hardware de los sistemas en *cloud* y probar automáticamente la validez de estos utilizando un enfoque rentable. Para ello, la utilización de MT ayuda a la verificación del comportamiento del sistema durante las pruebas sin necesidad de utilizar un oráculo, con las ventajas que esto conlleva. Se definen tres tipos de MR, enfocadas en aspectos como el rendimiento, la gestión de recursos y el coste. TEA-Cloud se evaluó a través de un estudio empírico donde se utilizaron técnicas basadas en testing de mutación y diez MRs para probar diferentes configuraciones de *cloud*.

En esta misma línea de investigación, Nuñez et al. proponen el uso de MT para detectar errores en sistemas distribuidos simulados utilizando aplicaciones HPC[116]. En esta metodología se utilizan las MRs como forma para representar las propiedades más relevantes del sistema. De esta forma, los resultados

obtenidos se contrastan con las MR definidas para determinar la fiabilidad de cada uno de ellos. Para mostrar la aplicabilidad de este enfoque, se modelan diferentes arquitecturas de sistemas distribuidos utilizando SIMCAN[95] y una aplicación de alto rendimiento que se ejecuta sobre los modelos. Una de las mayores ventajas que ofrece esta metodología es que es capaz de comprobar la corrección de una parte específica del sistema.

Relacionado con los entornos *cloud* y la validación de sistemas HPC, Cañizares et al.[128] proponen la validación de redes de comunicación en sistemas de computación de altas prestaciones. Para ello, proporcionan catálogo de MRs basadas en aspectos relacionados con las redes de comunicación para comprobar su corrección. Además, realizan un estudio experimental para analizar la red de comunicación de un sistema HPC. Los resultados muestran que MT es apropiado para comprobar la corrección de las redes de comunicación soportadas por topologías complejas en sistemas HPC.

En el ámbito la validación de software, Olsen y Raunak[129] proponen un enfoque donde utilizan MT para validar modelos de simulación ejecutables. Para ello, establecen pseudo-oráculos basados en MRs para un modelo ejecutable y así crear un enfoque metódico para la validación de los modelos de simulación. La fase de experimentación la realizan a través de tres casos de estudio diferentes: dos orientados a modelos de simulación basados en agentes y uno orientado a simulación de eventos discretos. Los resultados muestran la aplicación satisfactoria de MT para validar estos modelos de simulación diferentes.

En relación con los algoritmos genéticos, Rounds y Kanewala[130] presentan un enfoque en el cual se identifican 17 MRs para probar un algoritmo genético y mostrar, mediante MT, que estas relaciones son más efectivas para encontrar fallos que las pruebas unitarias tradicionales basadas en comparar los resultados obtenidos con resultados esperados. Durante el estudio, se encontraron ciertas MRs, enfocadas en analizar algoritmos genéticos, que son generalizables a diferentes tipos de algoritmos evolutivos que posean porcentajes de mutación similares. Cabe remarcar que esta es la primera vez que se aplican pruebas metamórficas para probar algoritmos genéticos.

Por último, Segura et al. presentan el concepto Performance MT[131 132]. En estos dos trabajos presentan las relaciones metamórficas de rendimiento (PMR por sus siglas en inglés) como las relaciones sobre las medidas de rendimiento del sistema bajo prueba a lo largo de diferentes ejecuciones del mismo. La hipótesis que proponen es que estas relaciones se pueden convertir en afirmaciones para la detección automática de errores de rendimiento, lo que elimina la necesidad de establecer *benchmarks* para medir el rendimiento del sistema así como la ayuda de expertos en el campo de aplicación.

4.4. Aplicación de GAs en sistemas Cloud computing y HPC

Cloud computing es un paradigma de computación que ha cobrado gran protagonismo en los últimos años[133][134]. Según la definición que ofrecen Boss et al.[135]: "*Cloud computing* es un término utilizado para describir tanto una plataforma como un tipo de aplicación. Como plataforma de computación en la nube suministra y configura servidores, los cuales pueden ser máquinas físicas o virtuales, dinámicamente bajo demanda de los usuarios. A su vez, *cloud computing* también define aplicaciones que se amplían para ser accesibles de manera remota. Estas aplicaciones en nube utilizan centros de datos y servidores que alojan aplicaciones y servicios web". Esta definición es solo una de las muchas que se pueden encontrar[136][137][134][138] en la literatura.

La computación en la nube es una estrategia que proporciona beneficios tanto para el proveedor de servicios como para el usuario. Quian et al.[134] resume las ventajas en tres importantes beneficios: satisfacción de los requisitos comerciales bajo demanda, es decir, se puede modificar el número de máquinas utilizadas para cada aplicación para cumplir con los requisitos de cada cliente; menor coste y mayor ahorro de energía, pues al hacer uso de máquinas diseñadas con un hardware optimizado para el bajo consumo de energía y optimizadas para la virtualización de servidores, tanto el coste de inversión como el coste de operación se reducen; y mejoran la gestión de recursos, a través de la planificación dinámica de los recursos del sistema se consigue optimizar la utilización de los mismos.

Pese a estas ventajas, el autor también presenta ciertos desafíos que han de ser considerados a la hora de utilizar *cloud computing*. El primero de ellos es la seguridad, antes de utilizar un servicio de computación ha de tenerse en cuenta que los datos estarán alojados en un sistema que será accedido desde múltiples localizaciones simultáneamente y una mala protección de los datos puede ser perjudicial para los usuarios. El segundo problema que se afronta es la migración de los servicios *cloud* pues no existe una estandarización de estos entre las distintas organizaciones proveedoras de este servicio y un cambio de proveedor puede significar un alto coste para la adaptación de las necesidades del usuario al nuevo *cloud*. Por último, existe un problema con la estabilidad del servicio, es decir, existen distintas circunstancias que pueden provocar que el servicio quede suspendido durante un periodo indeterminado de tiempo, como pueden ser problemas de conexión a la red, cortes de energía, interrupción del servicio y errores del sistema.

Por otro lado, la computación de altas prestaciones (HPC por sus siglas en ingles) según la definición que se encuentra en Oracle[139] se define como: "La computación de alto rendimiento se refiere a la práctica de agregar potencia de cálculo de forma que se obtenga una potencia mucho mayor que la de los ordenadores y servidores tradicionales."

Pese a la gran ventaja que supone esta capacidad de cálculo, el consumo energético es algo que también hay que tener en cuenta a la hora de diseñar un nuevo centro de datos para sistemas *cloud* y HPC[140]. Para disminuir este consumo, es trabajo de los diseñadores del sistema el encontrar la mejor configuración para reducir el consumo del sistema y que sea viable el uso de este. Sobre este aspecto se encuentran en la literatura multitud de propuestas[141].

A continuación se exponen los trabajos en los que se utilizan algoritmos evolutivos para mejorar la eficiencia energética de los sistemas *cloud*.

MT-EA4Cloud[8] es una metodología propuesta por Cañizares et al. la cual usa simultáneamente algoritmos evolutivos y MT para el desarrollo de un sistema eficiente desde un punto de vista energético. MT-EA4Cloud combina MT, un algoritmo evolutivo y simulación para un enfoque más práctico de la validación del consumo de energía. La aplicación del MT permiten modelar formalmente la infraestructura del sistema subyacente en forma de relaciones metamórficas. MT-EA4Cloud utiliza algoritmos evolutivos para la generación de nuevos individuos, mientras que utilizando un catálogo de MRs, definido por los autores, se guía el proceso de evolución de los individuos. De esta forma, se guían las nuevas generaciones de individuos de manera para la búsqueda de la optimización del consumo energético de los sistemas *cloud*, empleando diferentes simuladores *cloud*.

Parvizi et al.[142] presentan un trabajo en el que utilizan el algoritmo NSGA-III para determinar el despliegue de máquinas virtuales. El objetivo principal es minimizar la pérdida general de recursos y, al mismo tiempo, minimizar el consumo de energía y disminuir la cantidad de máquinas físicas activas. Para ello se diseña un problema de optimización multiobjetivo, y tras introducir una solución de optimización no lineal convexa, lo resuelven mediante NSGA-III. Con el objetivo de analizar esta propuesta, los autores utilizaron la plataforma de simulación CloudSim[100] para simular el algoritmo. Los resultados confirman la superioridad del método propuesto sobre los métodos básicos y los enfoques matemáticos en términos de criterios significativos, como el tiempo de ejecución, la utilización o el desperdicio de recursos y el consumo de energía.

Vila et al.[143] proponen un enfoque que utiliza algoritmos genéticos multiobjetivo para determinar la asignación óptima de *cloudlets*[144] a máquinas virtuales en entornos *cloud*. Buscan minimizar tanto el tiempo de ejecución de tareas como el consumo de energía, evitando asignaciones sistemáticas predefinidas y en su lugar utilizando algoritmos genéticos adaptables. Los resultados de su investigación muestran que su enfoque logra tiempos de ejecución más cortos y un menor consumo de energía en comparación con las soluciones tradicionales.

Ding et al.[145] proponen un enfoque para paliar el alto coste computacional de los algoritmos genéticos. Para ello los autores diseñan una estructura de datos que reduzca la complejidad del cálculo de

la función de aptitud para el problema de ubicación de máquinas virtuales en el sistema. Además, se propone una función de aptitud alternativa para reducir significativamente el número de instrucciones, disminuyendo así el tiempo de ejecución del algoritmo genético utilizado. Los estudios experimentales que presentan, muestran que este enfoque logra acelerar la computación del algoritmo genético para la colocación de máquinas virtuales con el objetivo de reducir consumo en centros de datos.

Athavale et al.[146] proponen un marco de trabajo, basado en un algoritmo genético, para minimizar el consumo energético durante la refrigeración de los centros de datos. Para ello, se lleva a cabo una optimización de puntos clave para el enfriamiento, asegurando que se cumplan los criterios de gestión térmica. En el trabajo incluyen tres aspectos claves: i) un modelo basado en una red neuronal artificial para la predicción rápida de la temperatura[147][148]; ii) un modelo termodinámico para la estimación de la energía de enfriamiento; y iii) un proceso de optimización basado en un algoritmo genético. Este trabajo presenta un problema en el modelo de predicción presentado, pues este es específico para la configuración considerada en el estudio y no es aplicable a otros escenarios, pero es posible generalizarlo.

Gabaldon et al.[149] presentan un MOGA para planificar de forma eficiente distintos procesos, teniendo en cuenta no solo la optimización del intervalo global de producción, sino también la reducción del consumo de energía. Esta propuesta utiliza una técnica basada en una *lista negra* de asignación de recursos que tiene la capacidad de prohibir el acceso a algunos nodos computacionales, proporcionando un mecanismo de reserva para aquellos trabajos con requisitos computacionales que pueden ofrecer mayores beneficios. Los experimentos fueron llevados a cabo mediante simulación, utilizando trazas de cargas de trabajo reales en un entorno de un clúster heterogéneo. Los resultados demostraron que el algoritmo presentado puede obtener mejores resultados que otras propuestas de la literatura, al tiempo que se optimizan tanto los objetivos como el ancho de banda y la eficiencia energética. Además, las soluciones proporcionadas tuvieron resultados de menor dispersión, lo que permite concluir que la robustez de un MOGA no depende de la naturaleza de las cargas de trabajo.

Vasudevan et al.[150] proponen un Algoritmo Genético de Reparación (RGA, por sus siglas en inglés) para abordar el desafío de equilibrar la eficiencia energética y la utilización de recursos en la asignación de aplicaciones a máquinas virtuales en centros de datos y servicios de nube. Este enfoque optimiza la asignación de aplicaciones como un problema de optimización bajo restricciones y mejora el algoritmo genético estándar al incorporar el concepto del procesador más rápido de la nube más larga (LCFP) y un procedimiento de reparación de soluciones inviables (ISRP). Los experimentos muestran que el RGA logra reducir el consumo energético y aumentar la utilización de los recursos en comparación a otros algoritmos genéticos.

Tseng et al.[151] proponen un MOGA para pronosticar dinámicamente el uso de recursos y el consumo

de energía en un centro de datos. Para ello, formulan un problema de optimización multiobjetivo de asignación de recursos, que considera la utilización de CPU, de memoria tanto de la máquina virtual como de la máquina física, y el consumo de energía. El algoritmo propuesto pronostica el requisito de recursos del próximo intervalo de tiempo de acuerdo con los datos históricos en los intervalos de tiempo anteriores. Además, proponen un algoritmo de ubicación de máquinas virtuales para asignar en el próximo intervalo de tiempo en función de los resultados de predicción del algoritmo.

Keshanchi et al.[152] proponen un algoritmo genético para la planificación de tareas en entornos *cloud* utilizando colas de prioridad[153]. Para optimizar la programación de tareas, el algoritmo genético utiliza las colas de prioridad como fichero elitista de tal forma que se hallará en la primera posición de la cola la tarea que se haya detectado como más prioritaria en el algoritmo. Los resultados estadísticos revelaron que el algoritmo disminuye los tiempos de ejecución de los otros algoritmos utilizados para la comparativa.

Li et al.[154] proponen un MOGA para la optimización del diseño térmico de un centro de datos. Primero, se construye un modelo del clúster bajo estudio mediante la Descomposición Ortogonal Propia (POD por sus siglas en inglés)[155]. Una vez obtenido este modelo, se utiliza para formar las funciones objetivo y de restricción de un modelo de optimización. A continuación, este modelo de optimización se integra con el nuevo MOGA. El algoritmo utiliza una operación guiada de *kriging*[156], un método que permite estimar los valores de una variable en lugares no muestreados, además de las operaciones de algoritmos genéticos convencionales para la búsqueda de soluciones óptimas. Los autores muestran que al optimizar el problema del clúster del centro de datos, el algoritmo propuesto mejora a uno convencional a la hora de estimar el frente de Pareto usando un 50 % menos de llamadas de simulación.

Rekha et al.[157] proponen un algoritmo genético para la optimización en la asignación de tareas de para así lograr reducir el tiempo de finalización de tareas. Este algoritmo propuesto se ha simulado utilizando el kit de herramientas de CloudSim[100]. El rendimiento del algoritmo se evalúa comparándolo con métodos de asignación de tareas tanto simples como voraces. teniendo un conjunto de parámetros como el rendimiento y el rendimiento para la programación de tareas. Los resultados de la evaluación han mostrado un mejor rendimiento con el enfoque propuesto.

Pese a los múltiples estudios enfocados en la optimización energética de los centros de datos orientados a *cloud computing* y HPC en los que combinan distintas tecnologías, en la literatura no se encuentran soluciones que afronten el problema desde la creación del centro de datos, sino que se afronta a nivel de planificación de procesos. Es por ello, que esta propuesta pretende optimizar el consumo energético de un centro de datos mediante el uso de simulación y algoritmos genéticos, junto con la validación de los resultados obtenidos mediante *metamorphic testing*, para llevarlo a cabo.

Capítulo 5

Framework propuesto para la optimización multiobjetivo de sistemas *cloud*

Uno de los objetivos del trabajo es la creación de un *framework* que integre algoritmos genéticos multiobjetivo, *metamorphic testing* (MT) y simulación. En este capítulo se presenta la arquitectura del *framework* desarrollado, así como los elementos que lo conforman (Sección 5.1). A su vez, para ilustrar su funcionamiento se detalla una interacción completa de la propuesta (Sección 5.2).

5.1. Arquitectura

En esta sección se presenta la arquitectura del *framework* propuesto. Este *framework* ha sido desarrollado a partir de un prototipo diseñado en el trabajo de Cañizares et al.[8], en el cual se emplean técnicas inspiradas en GAs tradicionales junto con MT y simulación. Cabe destacar que uno de los objetivos del trabajo es el estudio de MOGAs, por lo que las novedades incluidas en el *framework* base han sido realizadas en el módulo de selección, en el cual se han añadido 4 nuevos algoritmos. Por otro lado, las técnicas de MT han sido integradas en los módulos cruce y mutación, mientras que las técnicas de simulación han sido incluidas en el módulo evaluación.

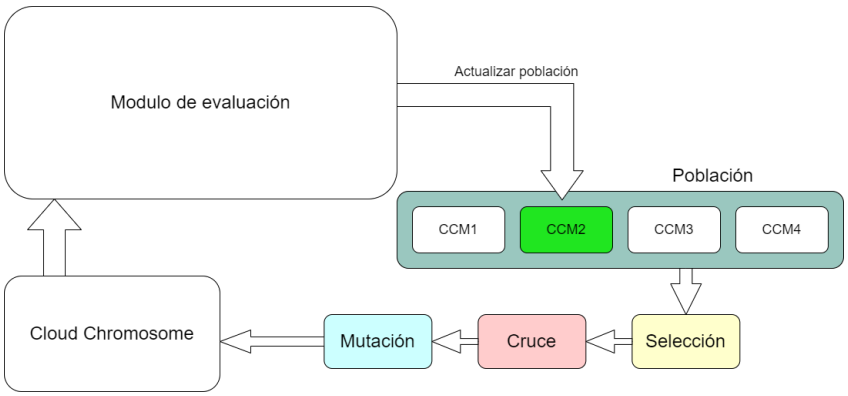


Figura 5.1: Arquitectura general del framework propuesto

La Figura 5.1 ilustra la arquitectura del *framework*, el cual está formado por seis módulos: *cloud*

```
# Host                      | # Storage
host.quantity = 2048        | sto.type = hdd
host.ram = 4000             | sto.capacity = 1000000
host.ramspeed = 1600        | sto.maxTransferRate = 100
host.bw = 2000000           | sto.latency = 5
host.sto = 1000000          |
host.mips = 1000            |
host.pes = 2                | # Network
                            | net.bandwidth = 500000
# Storage traces            | net.latency = 38
work.path = ...             | net.switches = 0
work.name = mix_vm          |
work.volume = 10            | #Cloud
work.numTraces = 10         | cloud.model = <ruta dir topología>
                            |
```

Figura 5.2: Fichero de configuración de un modelo *cloud*

chromosome, población, selección, cruce, mutación y evaluación.

El módulo *cloud chromosome* (CCM, por sus siglas en inglés) es el encargado de representar un individuo, el cual contiene la información necesaria para explorar el espacio de búsqueda de un dominio específico con unos objetivos determinados. Este módulo está compuesto por estructuras de datos que representan la información de un sistema *cloud*, a su vez, contiene los mecanismos necesarios para transformarlos en una codificación binaria a la hora de abordar las fases de cruce o mutación. Específicamente, un CCM está formado por tres elementos principales: un caso de prueba, el consumo de energía y el tiempo de ejecución. A su vez, un caso de prueba (TC, por sus siglas en inglés) consiste en un modelo *cloud* y una carga de trabajo. El modelo *cloud* define todos los componentes necesarios para modelar la arquitectura subyacente de un sistema *cloud*, tales como sus recursos computacionales (*racks*, nodos, CPUs, etc) y la topología de red. La Figura 5.2 muestra un ejemplo de un fichero de configuración de un modelo *cloud*, el cual contiene 2048 máquinas con una memoria RAM de 4000 MB. Además, posee una capacidad de 1 TB de almacenamiento junto con una red de comunicaciones de 500000 Mbps. Finalmente, cabe destacar que a través del parámetro `cloud.model`, se proporciona un mecanismo que permite detallar topologías complejas, en caso que sean soportadas por la plataforma de simulación en cuestión. Por otro lado, la carga de trabajo contiene las operaciones que debe procesar el modelo *cloud* durante la simulación. En la Figura 5.3 se muestra un ejemplo de la carga de trabajo, formada por 512 máquinas virtuales que poseen una RAM de 128 MB y una velocidad de CPU de 250 MIPS.

El módulo población se encarga de agrupar los distintos individuos que han sido generados y seleccionados durante las iteraciones previas. La población está formada por elementos del módulo CCM.

```
# VM
vm.quantity = 512
vm.mips = 250
vm.pes = 1
vm.ram = 128
vm.bw = 100000
vm.size = 2500
vm.priority = 1
vm.schedulingInterval = 300
```

Figura 5.3: Fichero de configuración de workload

Además, la población puede experimentar cambios a lo largo de las iteraciones por lo tanto este módulo se encarga de añadir, eliminar o intercambiar los individuos, así como de ordenarlos con un criterio definido, cuando sea necesario.

El módulo de selección es el encargado de escoger a los individuos más aptos de la población y es un proceso que debe implementar cada algoritmo de manera individual. El módulo de cruce es el encargado de generar descendencia mediante el cruce genético de dos individuos de la población, los progenitores. El módulo de mutación es el encargado de añadir modificaciones aleatorias a los individuos resultantes de la fase de cruce. Estos tres módulos están explicados en detalle en las secciones 5.1.1, 5.1.2 y 5.1.3, respectivamente.

Por último, el módulo de evaluación, representa la función fitness de los algoritmos genéticos tradicionales. Este módulo se encarga de simular y evaluar el rendimiento de los individuos de la población, es decir, los sistemas *cloud*. Para ello, toma los CCM generados tras el proceso de mutación, que aún no tienen valores de aptitud, los simula, mide el consumo de energía y tiempo de ejecución, y los añade a la población. Específicamente, las tareas realizadas por este módulo se detallan a continuación:

- Extracción de información: Recibe un CCM producido después de realizarse el proceso de mutación y extrae la información del modelo *cloud* contenido en el TC. Esta información se almacena en una estructura genérica, independiente de la plataforma de simulación y almacena los datos necesarios para configurar un caso de prueba.

- Generación de Archivos: Se generan los archivos de configuración propios del simulador, necesarios para ejecutar el TC.

- Simulación del *cloud*: El simulador recibe la información del modelo *cloud* y el TC y lleva a cabo la simulación del modelo.

- Extracción de Resultados: Una vez finalizada la simulación, se procesan los resultados obtenidos,

extrayéndose el consumo de energía y el tiempo de ejecución.

Una vez finalizado el proceso de evaluación, el CCM contiene toda la información en sus estructuras, por lo que este nuevo CCM se añade a la población.

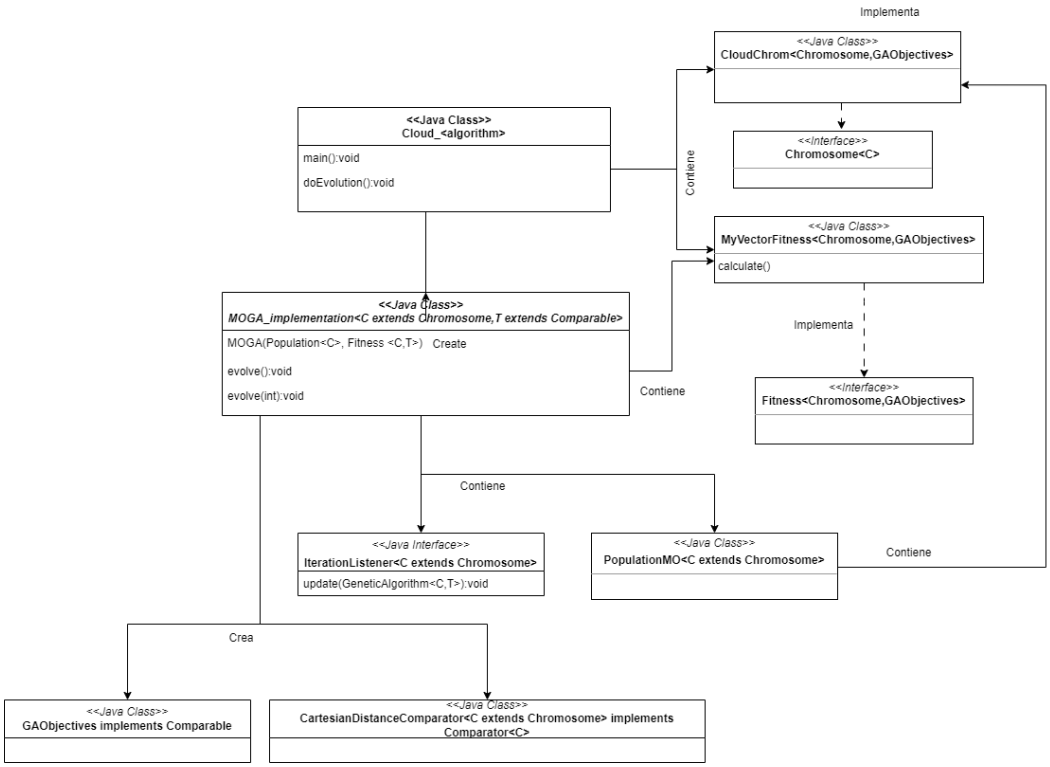


Figura 5.4: Diagrama de clases principales para la implementación de un algoritmo

Teniendo esta arquitectura en cuenta, la distribución de clases llevada a cabo durante la implementación se ha orientado en encapsular cada módulo en su propia clase. De esta forma, se establece la siguiente estructura de clases y módulos:

- Módulo CCM: Contenido en la clase `CloudChrom`, la cual hereda de la interfaz `Chromosome` y define el comportamiento de un individuo para llevar a cabo el algoritmo, es decir, se encargará de almacenar los valores de aptitud, almacenar los TC en las estructuras correspondientes o de comprobar si los resultados de la simulación son válidos.

- Módulo población: Contenido en la clase `PopulationMO` es la encargada de implementar el módulo población. Esta clase contiene una lista de individuos que tendrá que ir modificando según se sucedan los pasos del algoritmo y las iteraciones. Además, es la encargada de seleccionar aleatoriamente los cromosomas para el proceso de selección, así como de añadir individuos al terminar el módulo de evaluación, y eliminar o intercambiar los individuos una vez terminado el proceso de selección.

- Módulo selección: Contenido en la clase `MOGA_implementation`[1]. Esta clase es la encargada de orquestar los distintos pasos del algoritmo, haciendo las llamadas a los módulos correspondientes en cada paso de la iteración. Debe recibir como parámetro un objeto que pertenezca a la clase `PopulationMO` para realizar el proceso de selección de individuos, para continuar con los procesos de cruce y mutación que deben llevar a cabo los individuos seleccionados.

- Módulos cruce y mutación: Debido a las características de la implementación, estos módulos están contenidos dentro de la clase `CloudChrom`. Esta clase recibe la información del porcentaje de cruce y la probabilidad de mutación de la clase `MOGA_implementation` por ser la encargada de gestionar el algoritmo. Así pues, esta clase se encarga de comprobar que los individuos obtenidos tras el cruce y la mutación son correctos.

- Módulo de evaluación: Contenido dentro de la clase `MyVectorFitness` la cual, a través del método `calculate` definido se encarga de extraer el TC del CCM, generar los archivos necesarios y dar la orden de ejecución al simulador para al acabar recibir los resultados y actualizar la información del CCM escogido.

Adicionalmente a las clases expuestas, se encuentra la clase `Cloud_MOGA`. Esta clase se encarga de analizar los datos de entrada. Sigue una estructura similar en todos los algoritmos y únicamente habría que ajustar la invocación al algoritmo añadido.

En el repositorio **https://github.com/techmike17/ImprovingCloudArchitectures** se puede encontrar el código desarrollado durante este proyecto.

5.1.1. Selección

El proceso de selección es el mecanismo que emplean los GAs y MOGAs para escoger los individuos más aptos de cada generación permitiendo su propagación, mejorando así la población en cada iteración. Los algoritmos implementados en este trabajo son los siguientes: MOGA, PAES, NSGAII y SPEA2. A continuación, se detallan las características de sus procesos de selección:

MOGA: El proceso de selección de MOGA emplea un enfoque basado rangos: el algoritmo ordena la población en frentes de dominación y asigna a cada individuo un rango, es decir, el frente en el que se encuentra, cuanto más bajo sea el rango más dominante es el individuo. Una vez asignados los rangos, este se utiliza para elegir a los individuos que van a sobrevivir, y por lo tanto, pasar la siguiente generación. Los individuos en los frentes de Pareto con rangos más bajos tienen una mayor probabilidad de ser seleccionados. Esto fomenta la diversidad de soluciones en la población, ya que se

[1]Este nombre de clase es genérico y para la implementación se emplea el nombre del algoritmo que desarrolla.

busca mantener representantes de diferentes frentes de Pareto, aunque se da preferencia a las mejores soluciones. A continuación los individuos seleccionados son los que pasarán a la fase de cruce y mutación. Los detalles del proceso de selección de MOGA se describen en el Algoritmo 1. Inicialmente, comienza la iteración creando una nueva población en la que se van a copiar los elementos de la solución actual para ser utilizados posteriormente desde esta nueva población (líneas 2–4). A continuación, con cada individuo de la nueva población se generan nuevos individuos mediante cruce y mutación (líneas 5–20). Para llevar a cabo los operadores genéticos se realizan los siguientes pasos: i) *Extraer individuo*, se extrae el individuo de la población con el que se va a operar (línea 6); ii) *Mutación*: se crea nuevo individuo aplicando un operador de mutación (línea 7); iii) *Cruce*: otro individuo de la población es seleccionado aleatoriamente y se realiza el proceso de cruce con los individuos seleccionados, generando dos nuevos individuos (líneas 8–9). Después de generar estos nuevos individuos, se comprueba que los individuos obtenidos hayan cumplido satisfactoriamente las MRs definidas. A continuación, estos individuos se añaden a la nueva población (líneas 11–16). Este proceso se repite con todos los elementos de la población inicial y, una vez terminado, se calculan los valores de aptitud de los nuevos individuos y se ordenan los individuos en función del rango que se les ha asignado (línea 21). Una vez terminado este proceso comprueba nuevamente la aptitud de los cromosomas en la nueva población, analizando que siguen cumpliendo las MRs (líneas 22–28). Una vez que se han eliminado los individuos que no cumplen con alguna MR, y con los individuos ordenados de por rango, se ajusta el tamaño de la población al tamaño definido (línea 31). Finalmente, la nueva población sobre escribe la población actual (línea 32). Este ciclo se repite hasta que alcanzan el número de generaciones indicado.

Para la ordenación en distintos frentes se muestra el código empleado en el Algoritmo 2. Este algoritmo primero inicializa: una lista de listas, `dominationFronts`, para almacenar los frentes de Pareto a formar, una lista con las soluciones del frente de ese momento y los índices para recorrer las distintas listas (líneas 2-5). Estas listas contienen objetos `MOSolution`, que contienen la información del modelo *cloud*, el valor de aptitud en los objetivos y las estructuras y variables para poder asignarles un rango y una lista de ids de las soluciones dominadas. A continuación, se añaden los individuos de la lista *chromosomes* en la lista de soluciones para facilitar el proceso (líneas 6-10). Una vez inicializada esta lista se realiza un bucle doble sobre la lista de cromosomas (líneas 12-38). Este bucle doble se emplea para comprobar a qué individuos domina cada solución. En el caso en el que el individuo $c1$ domina al individuo $c2$, el individuo $c2$ se incluye como dominado en la lista de dominación de $c1$. En caso contrario, se incrementa en uno el número de soluciones que dominan a $c1$ (líneas 15-22). Una vez comprobada la dominación de $c1$ con el resto de soluciones, se comprueba si $c1$ ha sido dominado alguna vez, y en caso de no haberlo sido se añade al primer frente de dominación de `dominationFronts` (líneas 26-35). Una

vez que se han comprobado todos los individuos, se recorren las listas en las que se han almacenado los individuos para comprobar en qué frente de dominación han de ir localizados (líneas 41-51), teniendo en cuenta que el primer frente de dominación, con `rank=1` ya está completo. Por ello, se añaden al frente de dominación los individuos que al decrementarles en uno el número de veces que han sido dominados sea igual a cero, pues esto indica que los individuos que los dominan ya están en un frente anterior. Al terminar este proceso, se recorren los distintos frentes formados en `dominationFronts` y se aplanan para añadirlos a la lista `chromosomes` por orden de rangos (líneas 57-61).

Algorithm 1 Pseudocódigo MOGA

```
 1: procedure EVOLVE
 2:     parentPopulationSize ← tamaño de la población
 3:     newPopulation ← nueva población
 4:     COPY(newPopulation, population)
 5:     for i ← [0, parentPopulationSize) do
 6:         chromosome ← GETCHROMOSOMEBYINDEX(i, population)
 7:         mutated ← MUTATE(chromosome)
 8:         otherChromosome ← GETRANDOMCHROMOSOME(population)
 9:         crossovered ← CROSSOVER(chromosome, otherChromosome)
10:         if mutated ≠ NULL then
11:             ADDCHROMOSOME(newPopulation, mutated)
12:         end if
13:         if crossovered ≠ NULL then
14:             for c en crossovered do
15:                 if c ≠ NULL then
16:                     ADDCHROMOSOME(newPopulation, c)
17:                 end if
18:             end for
19:         end if
20:     end for
21:     SORTPOPULATIONBYFITNESS(newPopulation, chromosomesComparator)
22:     for i ← 0 hasta newPopulation.getSize() do
23:         chrom ← GETCHROMOSOMEBYINDEX(i, newPopulation)
24:         if no es válida la aptitud de chrom then
25:             DELETECHROMOSOME(newPopulation, chrom)
26:             Decrementar i en 1
27:         end if
28:     end for
29:     newPopulation ← ISFITNESSVALID(newPopulation)
30:     SAVEFORNORMALIZATION(newPopulaton)
31:     population ← newPopulation.trim(POPULATION_MAX_SIZE)
32:     population ← newPopulation
33: end procedure
```

PAES: El proceso de selección de PAES se basa en la no dominación entre individuos y no existe un proceso de cruce. Este algoritmo genera, en cada iteración, un nuevo individuo y una mutación de este. Estos dos nuevos individuos son analizados para comprobar su dominancia, y si uno es dominado por el otro, se desecha el dominado. El individuo restante pasará a ser comparado a las soluciones que se encuentran en el archivo de soluciones. Si este individuo no es dominado por ninguna solución, se eliminan las soluciones dominadas por el nuevo individuo y se añade. Si por el contrario, no domina a ningún individuo que pertenezca al archivo, el individuo se desecha. Si los dos individuos generados en

Algorithm 2 Ordenación cromosomas en frentes de dominación

```
1: function SORT(chromosomes)
2:     dominationFronts ← LinkedList < LinkedList < MOSolution < C, T ⋙
3:     solutionList ← LinkedList < MOSolution < C, T ≫
4:     moSolIndex1, moSolIndex2 ← MOSolution < C, T >
5:     nIndex1, nIndex2 ← 0
6:     for c1 ∈ chromosomes do
7:         moSolIndex1 ← MOSolution<C, T>(c1, fit(c1))
8:         solutionList.add(nIndex1, moSolIndex1)
9:         nIndex1 ← nIndex1 + 1
10:    end for
11:    nIndex1, nIndex2 ← 0
12:    for c1 ∈ chromosomes do
13:        moSolIndex1 ← solutionList.get(nIndex1)
14:        for c2 ∈ chromosomes do
15:            if nIndex1 ≠ nIndex2 then
16:                if dominates(c1, c2) then
17:                    moSolIndex2 ← solutionList.get(nIndex2)
18:                    moSolIndex1.insertDominatedSolutions(moSolIndex2)
19:                else if dominates(c2, c1) then
20:                    moSolIndex1.incrementDominations()
21:                end if
22:            end if
23:            nIndex2 ← nIndex2 + 1
24:        end for
25:        nIndex2 ← 0
26:        if moSolIndex1.getDominations() == 0 then
27:            moSolIndex1.setRank(1)
28:            if dominationFronts.isEmpty() then
29:                firstDominationFront ← LinkedList < MOSolution(C, T) > ()
30:                firstDominationFront.add(moSolIndex1)
31:                dominationFronts.add(firstDominationFront)
32:            else
33:                firstDominationFront ← dominationFronts.getFirst()
34:                firstDominationFront.add(moSolIndex1)
35:            end if
36:        end if
37:        nIndex1 ← nIndex1 + 1
38:    end for
39:    chromosomes.clear()
40:    i ← 1
41:    while dominationFronts.size() == i do
42:        nextDominationFront ← LinkedList < MOSolution < C, T ≫ ()
43:        for individualP ∈ dominationFronts.get(i − 1) do
44:            for individualQ ∈ individualP.getDominatedIndividuals() do
45:                individualQ.decrementDominations()
46:                if individualQ.getDominations() == 0 then
47:                    individualQ.setRank(i + 1)
48:                    nextDominationFront.add(individualQ)
49:                end if
50:            end for
51:        end for
52:        i ← i + 1
53:        if !nextDominationFront.isEmpty() then
54:            dominationFronts.add(nextDominationFront)
55:        end if
56:    end while
57:    for domFrontI ∈ dominationFronts do
58:        for domIndiv ∈ domFrontI do
59:            chromosomes.add(domIndiv.getIndividual())
60:        end for
61:    end for
62: end function
```

la iteración son candidatos a entrar en el archivo, quiere decir que no se dominan entre sí, por lo que ambos realizan el proceso para entrar al archivo.

El algoritmo PAES, presentado en pseudocódigo en el Algoritmo 3, siguiendo una estrategia (1+1) extrae un cromosoma en cada iteración, lo añade al archivo y seguidamente se obtiene otro individuo mediante la mutación (líneas 3-6). Si la mutación no es nula y cumple las MRs, se evalúa su dominancia en comparación con el cromosoma original (líneas 8-10), si no se generan individuos hasta que se consigue un individuo mutado válido. Si el cromosoma mutado está dominado por el original, el individuo original permanece en el archivo y el mutado se descarta (línea 11). Si el individuo mutado domina al original, se descarta la primera solución y se añade el individuo mutado al archivo y el otro se descarta (líneas 14-16). Si no se dominan mutuamente, es decir, cada uno es mejor que el otro en un objetivo distinto, ambos cromosomas se añaden al archivo (línea 18). El método *addToArchive* administra la adición de cromosomas al archivo teniendo en cuenta que el número de individuos en el archivo no sea mayor al tamaño definido (línea 26) y la densidad de la celda en la que se encuentra el individuo (línea 34). Este método contribuye a la estrategia de PAES para mantener un conjunto de soluciones no dominadas mientras explora el espacio de búsqueda en problemas de optimización multiobjetivo, descrito como estrategia de escalada de la colina en la Tabla 4.1.

NSGAII: El proceso de selección de NSGAII emplea la clasificación en frentes de no dominación para dividir las soluciones en diferentes niveles de dominancia. A continuación, se seleccionan las soluciones de los niveles no dominados de manera sucesiva, priorizando las soluciones con menor índice de dominación. Para mantener la diversidad, se utiliza un operador de selección en cada nivel de dominación el cuál escoge las soluciones en función de la densidad de población de cada solución. Esto permite que las soluciones no dominadas, pertenecientes al primer frente de dominación, y con menor índice de densidad pasen a las siguientes generaciones.

El algoritmo NSGAII, presentado en pseudocódigo en el Algoritmo 4, sigue una implementación basada en frentes de dominación de forma similar al algoritmo MOGA (ver Algoritmo 2), pero este nuevo algoritmo emplea una técnica conocida como "Fast Non-Dominated Sorting" (Ordenación no dominada rápida), que es más eficiente en términos de tiempo de ejecución. Su proceso de selección está basado en torneos, donde se seleccionan dos individuos al azar y se compara su dominancia (líneas 15-17). Una vez establecidos los dos padres, se llevan a cabo los procesos de mutación y cruce con los individuos seleccionados (líneas 18-21). Una vez realizado este proceso tantas veces como elementos haya en la población, se lleva a cabo la ordenación mediante frentes de dominación (línea 37). Finalmente, se descartan los individuos que no pertenezcan a los dos primeros frentes de dominación y se ajusta el tamaño de la población resultante para cumplir con el tamaño de población definido.

Algorithm 3 Pseudocódigo PAES

```
1: procedure EVOLVE
2:     mutated ← nulo
3:     chrom ← basepopulation.GETCHROMOSOMEBYINDEX(iteration)
4:     repeat
5:         mutated ← MUTATE(chrom)
6:         if mutated ≠ NULL then
7:             CALCULATESINGLECHROMOSOME(mutated)
8:             if mutated.isValid() then
9:                 dominated ← ISDOMINATED(mutated, chrom)
10:                if dominated < 0 then                ▷ El nuevo cromosoma está dominado por su padre
11:                    ADDTOARCHIVE(chrom, archive)
12:                end if
13:                if dominated > 0 then                ▷ El nuevo cromosoma domina a su padre
14:                    REPLACECHROMOSOME(chrom, mutated, archive)
15:                    ADDTOARCHIVE(mutated, archive)
16:                    DELETECHROMOSOME(chrom, archive)
17:                else                                 ▷ No se dominan entre ellos, entran al archivo
18:                    ADDTOARCHIVE(chrom, mutated)
19:                end if
20:            end if
21:        end if
22:    until mutated ≠ NULL
23: end procedure
24:
25: procedure ADDTOARCHIVE(chrom, mutated)
26:    if GETSIZE(archive) <archiveSize then
27:        if ADD(mutated, archive) then
28:            if GETDENSITY(mutated, archive) <GETDENSITY(chrom, archive) then
29:                DELETECHROMOSOME(chrom, basePopulation)
30:                ADDCHROMOSOME(mutated, basePopulation)
31:            end if
32:        end if
33:    else
34:        if ISINLESSCROWDEDREGION(chrom, mutated, archive) then
35:            if ADD(mutated, archive) then
36:                if GETDENSITY(mutated, archive) <GETDENSITY(chrom, archive) then
37:                    DELETECHROMOSOME(chrom, basePopulation)
38:                    ADDCHROMOSOME(mutated, basePopulation)
39:                end if
40:            end if
41:        else
42:            if GETDENSITY(mutated, archive) <GETDENSITY(chrom, archive) then
43:                DELETECHROMOSOME(chrom, basePopulation)
44:                ADDCHROMOSOME(mutated, basePopulation)
45:            end if
46:        end if
47:    end if
48: end procedure
```

Además, incorpora un operador llamado *crowding distance* (distancia de congestión), presentado en el Algoritmo 5, que se utiliza para mantener la diversidad en la población. Este operador ayuda a garantizar que las soluciones en cada frente de dominación estén bien distribuidas en el espacio de búsqueda, este valor se calcula en el proceso de ordenación y es una medida que evalúa cuán congestionada, o cercana, está una región del espacio de búsqueda en términos de soluciones cercanas en el espacio objetivo. Para el cálculo de este parámetro, primeramente se declara un array en el que se almacenarán las soluciones del frente de dominación (línea 2) y a continuación se fija la distancia de congestión de cada individuo a cero y se añade al array (línea 3-5). Para el cálculo de la *crowding distance* comienza un bucle doble el cuál se encarga de ordenar las soluciones de cada frente en función de un objetivo (línea 10) y fijar las distancias en los valores extremos del frente a infinito (líneas 11-12). En el caso de que ambas soluciones sean distintas, se calcula la *crowding distance* para la solución actual (líneas 14-19). La *crowding distance* se calcula como la diferencia entre los valores de objetivo de las soluciones siguientes y anteriores en la lista, normalizada por la diferencia entre los valores de objetivo de la última y la primera solución en la lista (línea 16). Una vez calculado, esto se añade a la *crowding distance* actual (líneas 17-18).

SPEA2: El proceso de selección de SPEA2 evalúa la aptitud de las soluciones en función de cuántas soluciones son dominadas por cada individuo. A su vez, para cada solución calcula una medida de densidad de población en el archivo. Estos dos valores obtenidos se emplean para asignar valores de adecuación a las soluciones, para a continuación, utilizando un operador de selección basado en torneos, elegir las soluciones más apropiadas para llevar a cabo el cruce de individuos. La selección favorece a las soluciones más fuertes, con mayor cantidad de soluciones dominadas, y menos densas, lo que guía la búsqueda hacia soluciones no dominadas y bien dispersas.

El algoritmo SPEA2, presentado en pseudocódigo en el Algoritmo 6, al principio de la iteración crea una nueva población temporal, a la cuál se copian los individuos que se encuentran en este momento en la población (líneas 2-4). A continuación, se obtienen los elementos almacenados en el archivo elitista "archive" (línea 5). Para cada cromosoma, se realiza una mutación y un cruce con otro cromosoma aleatorio de este mismo archivo (línea 6-9). Si los cromosomas generados tras el cruce y la mutación no son nulos se añaden a la nueva población (líneas 10-15). Una vez generados los nuevos individuos se ordena la nueva población en función de la aptitud de los cromosomas (línea 17). Esta aptitud se calcula en base al número de soluciones que domina cada solución y a su densidad de población, calculada mediante la distancia Manhattan de cada individuo al resto (líneas 39-46). Para este algoritmo, una solución que domine a un gran número de individuos y su densidad de población sea baja tendrá un valor de aptitud alto. Una vez obtenidos los valores de aptitud para cada solución de la nueva población

Algorithm 4 Pseudocódigo para NSGAII

```
1: procedure EVOLVE
2:     parentPopulationSize ← tamaño de la población
3:     newPopulation ← nueva población
4:     p1 ← Lista vacía de cromosomas
5:     for i ← (0, (mín parentPopulationSize, parentChromosomesSurviveCount)) do
6:         p1[i] ← population[i]
7:     end for
8:     for i ← (0 to parentPopulationSize) do
9:         randomIndex ← RANDOMNUMB([0, longitud(p1))
10:        if randomIndex ≠ −1 then
11:            SWAP(p1[i], p1[randomIndex1])
12:        end if
13:    end for
14:    for i ← (0 to parentPopulationSize) do
15:        randomNumbers ← TWORANDOMINDEX(0, longitud(p1))
16:        parent1 ← BINARYTOURNAMENT(p1[i], p1[randomNumbers[0]])
17:        parent2 ← BINARYTOURNAMENT(p1[i], p1[randomNumbers[1]])
18:        child1 ← DUP(parent1)
19:        child2 ← DUP(parent2)
20:        mutate1 ← MUTATE(child1)
21:        mutate2 ← MUTATE(child2)
22:        if mutate1 ≠ NULL then
23:            newPopulation.add(mutate1)
24:        end if
25:        if mutate2 ≠ NULL then
26:            newPopulation.add(mutate2)
27:        end if
28:        crossovered ← CROSSOVER(child1, child2)
29:        if crossovered ≠ NULL then
30:            for c en crossovered do
31:                if c ≠ NULL then
32:                    newPopulation.add(c)
33:                end if
34:            end for
35:        end if
36:    end for
37:    SORTBYFITNESS(newPopulation, chromosomesComparator)
38:    newPopulation ← ISFITNESSVALID(newPopulation)
39:    SAVEFORNORMALIZATION(newPopulaton)
40:    population ← newPopulation.trim(POPULATION_MAX_SIZE)
41: end procedure
```

Algorithm 5 Pseudocódigo para Calculate Crowding Distance

```
1: procedure CalculateCrowdingDistance(dominationFront)
2:     sortedSols ← MOSolution[ ]
3:     for k = 0 to dominationFront.size() − 1 do
4:         dominationFront[k].setCrowdingDistance(0)
5:         sortedSols[k] = dominationFront[k]
6:     end for
7:     Declare numobjectives as the number of EGAObjectives
8:     for m = 0 to numobjectives − 1 do
9:         objective ← EGAObjectives[m]
10:        sortedSols.sort(getObjective(objective))
11:        sortedSols[0].setCrowdingDistance(∞)
12:        sortedSols[sortedSols.length − 1].setCrowdingDistance(∞)
13:        if sortedSols[0].getObjective(objective) ≠ sortedSols[sortedSols.length − 1].getObjective(objective) then
14:            for i = 1 to sortedSols.length − 2 do
15:                newCrowdingDistance = sortedSols[i].getCrowdingDistance()
16:                aux = (sortedSols[i+1].getObjective(objective) − sortedSols[i−1].getObjective(objective)) / (sortedSols[sortedSols.length−1].getObjective(objective) − sortedSols[0].getObjective(objective))
17:                newCrowdingDistance+ = aux
18:                sortedSols[i].setCrowdingDistance(newCrowdingDistance)
19:            end for
20:        end if
21:    end for
22: end procedure
```

se actualiza el archivo de soluciones con aquellas que poseen mejor valor de aptitud. A continuación, se calcula la distancia cartesiana para las soluciones del archivo y se ordenan en función de este nuevo parámetro (línea 25). De esta forma, en la primera posición se encuentra la solución que más cerca se queda del centro del eje de coordenadas. Finalmente, se ajusta el tamaño de la población nueva para que no exceda un límite máximo y se actualiza la población actual con la nueva población generada (líneas 26-27).

Para potenciar y facilitar el estudio de los MOGAs, además de incluir los algoritmos presentados, se ha dotado al *framework* con una arquitectura flexible que permite la inclusión de nuevos algoritmos de manera sencilla. Para añadir un nuevo algoritmo es necesario implementar cinco clases sobre la arquitectura base mostrada en la Figura 5.4.

- `MOGA_implementation`: Esta clase se encarga de orquestar el algoritmo, realizando llamadas a los módulos encargados de implementar la funcionalidad de los distintos pasos. Recibe como parámetros una clase que herede de `Chromosome` y otra de `Comparable` para llevar a cabo los operadores genéticos.

- `Cloud_MOGA`: Esta clase se encarga de analizar los datos de entrada. Sigue una estructura similar en todos los algoritmos y sólo habría que ajustar la invocación al algoritmo añadido.

- `CloudChromosome`: Esta clase hereda de la interfaz `Chromosome` y define el comportamiento de un individuo. Por ejemplo, se encargará de almacenar los valores de aptitud o de comprobar si los

Algorithm 6 Pseudocódigo SPEA2

```
1: procedure EVOLVE
2:     parentPopulationSize ← tamaño de la población
3:     newPopulation ← nueva población
4:     COPY(newPopulation, population)
5:     for i ← 0 hasta archive.getSize() do
6:         chromosome ← GETCHROMOSOMEBYINDEX(i, archive)
7:         mutated ← MUTATE(chromosome)
8:         otherChromosome ← GETRANDOMCHROMOSOME(archive)
9:         crossovered ← CROSSOVER(chromosome, otherChromosome)
10:        if mutated ≠ NULL then
11:            ADDCHROMOSOME(newPopulation, mutated)
12:        end if
13:        if crossovered ≠ NULL then
14:            ADDCHROMOSOME(newPopulation, crossovered)
15:        end if
16:    end for
17:    SORTPOPULATIONBYFITNESS(newPopulation, chromosomesComparator)
18:    for i ← 0 hasta newPopulation.getSize() do
19:        chrom ← GETCHROMOSOMEBYINDEX(i, newPopulation)
20:        if !chrom.isNotValid() then
21:            DELETECHROMOSOME(newPopulation, chrom)
22:        end if
23:    end for
24:    UPDATEARCHIVE(newPopulation)
25:    SAVEFORNORMALIZATION(newPopulaton)
26:    population ← newPopulation.trim(POPULATION_MAX_SIZE)
27:    population ← newPopulation
28: end procedure
29: procedure UPDATEARCHIVE(newPop)
30:    for cada chrom en newPop do
31:        if chrom.getNumDom() = 0,0 then
32:            ADDCHROMOSOME(archive, chrom)
33:        end if
34:    end for
35:    SORTPOPULATIONBYFITNESS(archive, fc)
36:    TRIM(archive, POPULATION_MAX_SIZE)
37: end procedure
38:
39: function CALCULATEMANHATTAN(o1, o2)
40:    dist ← 0,0
41:    for obj ∈ EGAObjectives.values() do
42:        dValue1 ← o1.getObjective(obj)
43:        dValue2 ← o2.getObjective(obj)
44:        dist ← dist + |dValue1 − dValue2|
45:    end for
46:    return dist
47: end function
```

resultados de la simulación son válidos.

- `VectorFitness`: Esta clase hereda de la interfaz *Fitness* y se encarga de calcular los valores de aptitud para los objetivos.

- `IterationListenerMOGA`: Esta clase almacena en estructuras la información relativa a cada iteración.

Por ejemplo, para añadir el algoritmo NSGAII al *framework* se han seguido los siguientes pasos:

- Implementar clase parser: Primero se crea la clase `Cloud_NSGAII`, encargada de analizar los datos de entrada y crear las estructuras correspondientes para el individuo y la población inicial. Esta clase es la encargada de instanciar la clase del algoritmo y ejecutarlo (mediante el método `evolve(iterations)`) y de implementar dos clases internas, `CloudChromosome` y `VectorFitness`, que implementen las interfaces `Chromosome` y `Fitness`, respectivamente.

- Implementar clase `NSGAII`: Esta clase implementa los métodos necesarios para el funcionamiento del algoritmo. En este caso se implementa un comparador para que ordene la población en frentes de no dominación, así como la selección de los individuos para llevar a cabo el cruce y la mutación, de la cual se encargan los propios individuos de la clase `Chromosome`.

- Implementar `IterationListenerNSGAII`: La implementación de esta interfaz es la encargada de guardar el estado de la población y del algoritmo en cada iteración del algoritmo.

5.1.2. Cruce

Una vez finalizado el proceso de selección, en el que se han escogido los mejores individuos de cada generación, se lleva a cabo el proceso de cruce con estos individuos. Durante la fase de cruce, los individuos seleccionados, se combinan siguiendo los principios de la reproducción biológica, como se ha explicado en la Sección 3.2.1. En este *framework*, se han implementado dos tipos de cruce distintos con el objetivo de evitar la convergencia prematura y mejorar el rendimiento del algoritmo. Debido a la naturaleza de este tipo de algoritmos, únicamente se realiza un cruce entre cada par de individuos. Por tanto, en este caso, es necesario seleccionar una única técnica de cruce, por lo que en cada iteración del algoritmo se selecciona una técnica al azar para ser aplicada. Los operadores de cruce desarrollados se explican a continuación:

- **Cruce mixto**: En este cruce, se combina la información de dos máquinas físicas para generar una nueva descendencia. Para cada par de *clouds* de la población actual se generan dos descendientes

para la siguiente población. Este cruce se lleva a cabo representando en un vector binario cada uno de los componentes que conforman el conjunto de componentes de cada *cloud*. Estos vectores binarios se combinan empleando los pasos del cruce presentado en la Sección 3.2.1 e ilustrado en la Figura 3.5, estableciendo un punto de cruce y combinando el vector de componentes. Para ilustrar este cruce, la Figura 5.5 representa el cruce entre dos CPUs, CPU1 (verde) y CPU2 (rosa), se combinan para generar dos nuevas CPUs, cada una con información de los dos padres. Inicialmente, el usuario debe proporcionar el porcentaje de individuos involucrados en el cruce. Por lo tanto, una parte de la *cloud* permanecerá sin modificaciones, a menos que se indique un 100 % de cruce entre los individuos.

- **Cruce de intercambio**: En este cruce, el operador combina componentes de los padres para generar la descendencia, pero sin alterar sus características. Por lo tanto, la información binaria de cada componente no se modifica, sino que intercambia con la del otro individuo. De esta forma, las máquinas de la nueva descendencia son una combinación de componentes de ambos padres. Por ejemplo, dadas dos máquinas seleccionadas al azar, una de las máquinas posee 12 núcleos en la CPU y 4 GB de memoria RAM y la otra máquina posee 4 núcleos en la CPU y 12 GB de memoria RAM. Tras llevar a cabo este cruce, la descendencia generada será una máquina de 12 núcleos en la CPU y 12 GB de memoria RAM y otra máquina con 4 núcleos en la CPU y 4 GB de memoria RAM.

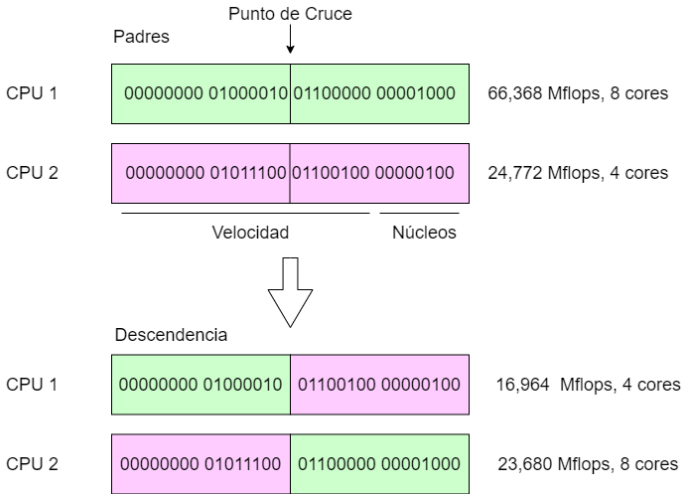


Figura 5.5: Cruce mixto

5.1.3. Mutación

Tras realizar el proceso de cruce, se lleva a cabo el proceso de mutación para añadir cambios aleatorios en los individuos generados en el módulo anterior. El objetivo del proceso de mutación es introducir

cambios en los individuos de la población de tal forma que se modifiquen sus componentes para facilitar la evolución de la población, mitigando así potenciales problemas de convergencia. Para el proceso de mutación se han definido ocho operadores de mutación diferentes:

- **Operador 1**: Inserta mutaciones en todos los componentes de un *rack*. Inicialmente, dado un *cloud*, se selecciona aleatoriamente un *rack* donde todos tienen la misma probabilidad de ser seleccionados. Una vez seleccionado, el operador introduce cambios en todos los componentes del *rack* (nodos, disco, CPUs, ...).
- **Operador 2**: Disminuye el ancho de banda de un enlace de red seleccionado aleatoriamente de la red de comunicaciones del *cloud*.
- **Operador 3**: Aumenta la latencia de un enlace de red seleccionado al de manera aleatoria de la red de comunicaciones del *cloud*.
- **Operador 4**: Elimina un enlace de la red, esto sólo se aplica si el grafo resultante es conexo.
- **Operador 5**: Crea un nuevo enlace que conecta dos *racks*, seleccionados al azar, que no están directamente conectados. Los valores de ancho de banda y latencia del nuevo enlace de red se establecen como el promedio de los enlaces existentes en la red.
- **Operador 6**: Reemplaza el origen/destino de un enlace seleccionado aleatoriamente.
- **Operador 7**: Elimina un *rack* seleccionado aleatoriamente y todos sus enlaces conectados. Si el grafo resultante no está conectado, se crean nuevos enlaces hasta que el grafo sea conexo. Los nuevos enlaces generados toman los valores de ancho de banda y latencia de los enlaces eliminados. Los nodos que conforman los enlaces se seleccionan al azar siguiendo el Operador 6. Para ello, los enlaces de conexión son modificados para evitar grafos no conectados.
- **Operador 8**: Divide un *rack* en dos, ambos con la mitad de las capacidades del original, y donde los enlaces se duplican.

El problema que presenta la mutación es que puede desembocar en la generación de individuos incorrectos. Para controlar estos casos, se definen una serie de MRs que los nuevos individuos deben cumplir con el objetivo de guiar el proceso de mutación hacia un espacio de soluciones reducido y adecuado. Las MRs definidas son las siguientes:

- **MR1**: Si la CPU del *cloud* m tiene un mejor rendimiento que la CPU del *cloud* m', y la carga de trabajo es la misma para ambos, entonces la cantidad de energía requerida para ejecutar esta

carga de trabajo en m debería ser menor o igual a la requerida para ejecutarla en m' y el tiempo necesario para ejecutar la carga en m sería menor o igual al necesario para ejecutarla en m'.

- **MR2**: Si el *cloud* m contiene más máquinas que el *cloud* m', y la carga de trabajo es la misma para ambos, entonces la proporción entre el número de máquinas de m y m' debería ser mayor o igual a la proporción entre el consumo de energía necesario para ejecutar la carga en m y en m' y menor o igual a la proporción entre el tiempo necesario para ejecutar la carga en m y al necesario para ejecutarla en m'.

- **MR3**: Si la proporción entre el número de máquinas del *cloud* m y el *cloud* m' es mayor o igual a la proporción entre el rendimiento de la CPU de m y el rendimiento de la CPU de m', y la carga de trabajo es la misma para ambos, entonces la proporción entre el consumo de energía necesario para ejecutar la carga de trabajo en m y el necesario para ejecutarla en m' debería ser menor o igual a la proporción entre el número de máquinas de m y m' y mayor o igual respecto a la proporción entre el tiempo necesario para ejecutar la carga de trabajo en m y el necesario para ejecutarla m'.

- **MR4**: Si el rendimiento de E/S de m es mejor que el rendimiento de E/S de m', y la carga de trabajo es la misma para ambos, entonces el consumo de energía necesario para ejecutar esta carga en m debería ser menor o igual al necesario para ejecutarla en m' y el tiempo de ejecución necesario para ejecutar la carga en m debería ser menor o igual al necesario para ejecutarla en m'.

- **MR5**: Si el rendimiento de red del *cloud* m es mejor que el rendimiento de red del *cloud* m', y la carga de trabajo es la misma para ambos, entonces el consumo de energía necesario para ejecutar esta carga de trabajo en m debería ser menor o igual al necesario para ejecutarla en m' y el tiempo de ejecución necesario para ejecutar la carga en m debería ser menor o igual al necesario para ejecutarla en m'.

- **MR6**: Si el rendimiento de la memoria RAM del *cloud* m es mejor que el rendimiento de la memoria RAM del *cloud* m', y la carga de trabajo es la misma para ambos, entonces el consumo de energía necesario para ejecutar esta carga de trabajo en m debería ser menor o igual al necesario para ejecutarla en m' y el tiempo de ejecución necesario para ejecutar la carga en m debería ser menor o igual al necesario para ejecutarla en m'.

- **MR7**: Si el número de máquinas utilizadas en dos *clouds*, m y m', es igual, el número de máquinas virtuales generadas en m es mayor que el número generado en m', y la carga de trabajo es la misma

para ambos, entonces el consumo de energía necesario para ejecutar esta carga de trabajo en m debería ser mayor o igual al necesario para ejecutarla en m' y el tiempo de ejecución necesario para ejecutar la carga en m debería ser mayor o igual al necesario para ejecutarla en m'..

- **MR8**: Si los *clouds* m y m' son iguales y la carga de trabajo ω es una subtraza de la carga de trabajo ω', entonces la energía necesaria para ejecutar ω en m debería ser menor o igual a la necesaria para ejecutar ω' en m' y el tiempo de ejecución necesario para ejecutar ω en m debería ser menor o igual al necesario para ejecutarω' en m'.

Como se puede apreciar, los operadores de mutación están basados en las MRs definidas. Esto se lleva a cabo de esta forma para que cada vez que se aplica un operador, haya una MR que compruebe el individuo obtenido de esta mutación.

5.2. Ciclo completo de ejecución

Una vez presentada la arquitectura del *framework* desarrollado, en esta sección se describen los pasos necesarios para la ejecución de un TC concreto en el sistema. Estos pasos se recogen en la figura 5.6 y se detallan a continuación.

Creación TC inicial. El usuario debe modelar el sistema *cloud* a partir del cual se creará la población inicial. Este modelo es almacenado en un fichero y da valor a los componentes que conforman el *cloud* diseñado, de forma similar al módulo CCM. Además, el usuario debe generar tres ficheros de configuración del TC: metaInfo, input, output.

- MetaInfo: Contiene el archivo donde se define la ruta en la que se encuentra el fichero de entrada (input) y salida (output) del simulador, respectivamente.
- Input: Representa un modelo *cloud*, proporcionando valores a los distintos componentes del mismo, tal y como se ha mostrado en la Figura 5.2, así como la ruta al *workload*.
- Output: Contiene los resultados de la simulación del *cloud* y es generado por el simulador tras finalizar su ejecución.

Especificación parámetros de entrada. Para ejecutar el TC generado en el paso anterior, el sistema requiere determinados parámetros de entrada. Los parámetros de entrada que espera recibir el *framework* son los siguientes:

- *Algoritmo*: Especifica el algoritmo que se desea emplear para esta ejecución. Este parámetro es opcional y en caso de no aparecer se ejecutarán todos los algoritmos integrados en el *framework*.

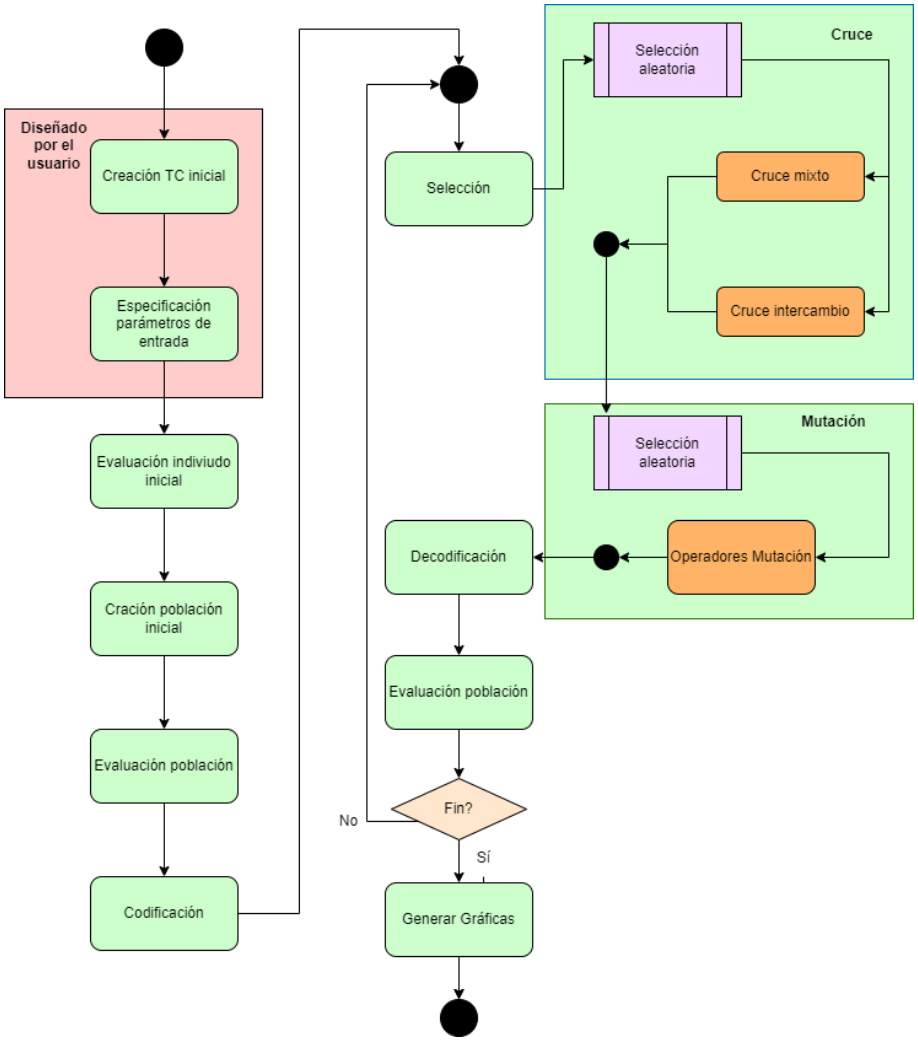


Figura 5.6: Iteración completa del framework

- *Simulador*: Simulador sobre el que se van a ejecutar los casos de prueba.
- *Ruta_tests*: Ruta al directorio donde se han definido los ficheros de configuración.
- *Evoluciones*: Número de iteraciones que ha de realizar el algoritmo.
- *Porcentaje1*: Número entero que define el porcentaje de individuos involucrados en el cruce.
- *Probabilidad2*: Número entero que define la probabilidad de mutación.
- *Ruta_simulador*: Ruta al ejecutable del simulador encargado de ejecutar las pruebas.

En la Figura 5.7 se muestra un ejemplo de configuración de los parámetros de entrada. Se puede observar que el algoritmo seleccionado es NSGAII, con el simulador CloudSimStorage, el directorio donde se encuentran los TCs es */localSpace/cloudEnergy/cloudsimStorage/test_generic_fw*, durante una iteración en la que se van a involucrar el 100 % de las máquinas en el cruce y con un 10 % de probabilidades de mutación, con el simulador alojado en el directorio */localSpace/cloudEnergy/cloudsimStorage/evolutionary*.

```
nsgaii cloudsimstorage /localSpace/cloudEnergy/cloudsimStorage/test_generic_fw 1 100 10
/localSpace/cloudEnergy/cloudsimStorage/evolutionary
```

Figura 5.7: Ejemplo de parámetros de entrada del *framework*

Evaluación individuo inicial. Una vez el usuario ha diseñado el TC inicial, ha de ser evaluado para calcular su consumo y el tiempo de ejecución. Esta evaluación consiste en simular el modelo *cloud* diseñado en pasos anteriores, para así obtener los valores de aptitud de los objetivos.

Creación población inicial. Una vez obtenidos los valores iniciales del individuo es necesario generar la población inicial. Esta población se genera obteniendo nuevos individuos aplicando mutaciones en el individuo inicial.

Evaluación población. Una vez obtenida la población inicial, se simula cada individuo generado para así obtener los valores de aptitud de cada objetivo para los individuos de la población.

Codificación. Este paso se encarga de de representar los individuos de la población de forma que su modificación sea sencilla y evite modelos incorrectos. Esto lo consigue empleando una configuración híbrida empleando grafos, para representar las topologías, y números enteros, para representar los componentes. Además, se añaden los mecanismos necesarios para poder representar el sistema de forma binaria para los casos en los que sea necesario. Esto facilita la gestión de grandes estructuras, como puede ser un sistema *cloud*, y de la aplicación de los operadores genéticos.

Selección. El proceso de selección se encarga de escoger los individuos más aptos de cada generación, con el objetivo de propagar sus características al resto de generaciones. Este proceso difiere entre los distintos MOGAs integrados en el *framework* y por lo tanto los detalles específicos están sujetos a cada uno de ellos (ver Sección 5.1.2).

Cruce y mutación. Este proceso está explicado en detalle en las secciones 5.1.2 y 5.1.3. Por un lado, el proceso de cruce se lleva a cabo seleccionando de manera aleatoria entre dos métodos de cruce. Por otro lado, el proceso de mutación dirigida por MRs se realiza de forma similar a la mutación de los GAs tradicionales, a diferencia de que los individuos deben cumplir las distintas MRs propuestas.

Decodificación. Este paso se encarga de comprobar que los individuos sigan satisfaciendo las MRs tras los múltiples cambios a los que se han sometido y realizar el proceso de codificación a la inversa, es decir, su paso a estructuras de nuevo. En caso de detectar un modelo incorrecto este se sustituye por otro que satisfaga las MRs.

Tras evaluar de nuevo la población generada, se comprueba si se cumple la condición de parada del algoritmo, es decir, se han realizado todas las iteraciones. En caso de que no se satisfaga la condición se vuelve a realizar el proceso desde el paso de selección. En caso contrario el algoritmo finaliza y comienza

el siguiente y último paso.

Generación Gráficas. En este paso se recoge la información almacenada por el *listener* durante la ejecución y se generan los ficheros que proporcionan información sobre el comportamiento del algoritmo a lo largo de las iteraciones. Esta información se escribe en tres ficheros distintos. El primero de los ficheros generados almacena los valores de energía consumida y tiempo de ejecución de todos los individuos generados, el segundo genera una lista de los individuos que han formado parte de cada una de las iteraciones, y por último, el tercer fichero genera una lista que contiene los tres mejores individuos de cada iteración. Una vez generados estos ficheros de salida, se generan las gráficas que permiten la evaluación, de manera clara y comprensible, de cada uno de los algoritmos en función de los resultados obtenidos.

Capítulo 6

Estudio empírico

En este capítulo se describe el estudio empírico, llevado a cabo con el *framework* propuesto, enfocado en la evaluación de sistemas *cloud* teniendo en cuenta el consumo energético y el tiempo de ejecución. Para ello se han implementado cuatro MOGAs, descritos en la Sección 5.1.1 y se han incluido en el *framework*. Además se han diseñado manualmente dos sistemas *cloud* (uno de bajas prestaciones y otro con altas capacidades de cómputo y almacenamiento). El objetivo de este estudio es analizar la idoneidad de los algoritmos implementados para optimizar las dos arquitecturas *cloud* propuestas. En la Sección 6.1 se describen los *clouds* diseñados para este estudio. Seguidamente, en la Sección 6.2, se presenta el ajuste de parámetros de los algoritmos para la ejecución del proceso de optimización. Por último, se exponen y analizan los resultados obtenidos en la Sección 6.3.

6.1. Sistemas *cloud* utilizados en el estudio

Los sistemas *cloud* son arquitecturas distribuidas formadas por una amplia variedad de componentes, tales como CPUs, memorias, redes, etc, los cuales pueden afectar al rendimiento del sistema en función de sus características. Para afrontar la fase experimental del trabajo, se han configurado manualmente dos sistemas *clouds* diferentes, *CloudA* y *CloudB*. Las características de ambos sistemas se muestran en la Tabla 6.1.

Por un lado, *CloudA* posee un perfil de prestaciones bajo, proporcionando CPUs con una potencia de cómputo limitada y un número reducido de núcleos de CPU, memorias RAM de tamaño reducido y discos de almacenamiento con prestaciones limitadas, esto es, con una tasa de Mbps baja, junto con una red de comunicación con bajo ancho de banda. Por otro lado, *CloudB* presenta un perfil de rendimiento alto, utilizando CPUs de mayor potencia y aumentando las prestaciones de los discos duros y de la memoria RAM, junto con una red de comunicación con mayor ancho de banda.

Parámetro	*CloudA*	*CloudB*
Hosts	512	512
RAM (MBytes)	1024	16,384
CPU speed (MIPS)	1k	90k
CPU cores	2	8
HDD size (TBytes)	1	1
HDD speed (Mbps)	20	350
Net bw (Mbps)	500	10,000
Net lat (us)	10	10

Tabla 6.1: Configuraciones de los clouds diseñados

6.2. Configuración de los algoritmos utilizados en el estudio

La eficiencia y efectividad, tanto de los GAs, como de los MOGAs, depende en gran parte de los parámetros de configuración. A lo largo de esta sección se describe el ajuste de parámetros para llevar a cabo la fase experimental.

Los algoritmos utilizados en este estudio dependen principalmente de cuatro parámetros diferenciados: número de generaciones (o iteraciones), tamaño de la población, porcentaje de cruce y probabilidad mutación. El número de generaciones define las veces que debe aplicarse el algoritmo sobre una generación de individuos, es decir, el número de iteraciones que tiene que ejecutar el ciclo de selección, cruce, mutación y evaluación. El tamaño de la población define el número de elementos máximos que tendrá el algoritmo en cada iteración, de tal forma que al terminar una iteración, en caso de superar el límite, el proceso de selección escogerá como mucho este número de individuos para la población. El porcentaje de cruce define el porcentaje de los individuos que van a formar parte en el proceso de cruce. La probabilidad de mutación define la probabilidad que tiene cada individuo de ser sometido al proceso de mutación.

En el trabajo de Cañizares et al.[8] se presenta un estudio sobre el impacto de estos parámetros en el algoritmo utilizado para la optimización de sistemas *cloud*. Para los MOGAs utilizados en este estudio, se utilizan los mismos parámetros que los utilizados por Cañizares et al. en su propuesta. Así, se establece el número de iteraciones en 100, el tamaño de población en 10 individuos y, por último, la probabilidad de cruce será del 50 %. Para definir la probabilidad de mutación se emplean 3 configuraciones distintas, las cuales se presentan en la Tabla 6.2. En esta tabla, cada columna refleja la probabilidad de aplicar un operador de mutación específico (ver Sección 5.1.3), mientras que cada fila representa una configuración de mutación (*com*). Para cada operador, la suma de las probabilidades debe ser igual o menor al 100 %. En aquellas situaciones donde la suma es menor, se utiliza la diferencia para indicar el operador nulo, es decir, aquel que no aplica ninguno de los operadores de mutación. Por ejemplo, para configuración

low hay una probabilidad del 1.5 % de aplicar el operador *oper2* sobre cada individuo, mientras que esta configuración tiene una probabilidad del $100-(1{,}5+1{,}5+1+0{,}5)=95{,}5\,\%$ de aplicar el operador nulo.

Config	**oper1**	**oper2**	**oper3**	**oper4**	**oper5**	**oper6**	**oper7**	**oper8**
low	1.5 %	1.5 %	1 %	—	—	—	0.5 %	—
mid	15 %	10 %	5 %	—	—	—	1 %	—
high	25 %	25 %	25 %	—	—	—	25 %	—

Tabla 6.2: Valores mutación para los operadores

Una vez configurados los parámetros de los algoritmos, es necesario definir la carga de trabajo que deben procesar las arquitecturas *cloud* mostradas en la Tabla 6.1. Para definir la carga de trabajo se ha optado por generar dos modelos inspirados en las operaciones realizadas en el análisis de *big data*. En particular, las trazas empleadas representan la infraestructura de PlanetLab [158] y tienen dos configuraciones. La primera, de mayor tamaño, se identifica como ω^l, mientras que la segunda, de menor tamaño, se representa con ω^s.

Adicionalmente, se ha añadido un mecanismo de ordenación para establecer un orden entre individuos no dominados entre sí. Los individuos no dominados son estrictamente mejores en un objetivo, y mejores o iguales en el resto. Aquellos individuos no dominados se ordenan en función de la distancia normalizada al centro del eje de coordenadas, de tal forma que se consideran mejores soluciones aquellas que estén más cerca del centro y serán peores los individuos que se encuentren en los extremos del espacio de soluciones. Este mecanismo ha sido añadido para poder ofrecer, adicionalmente, la solución más adecuada en cada iteración, representada en los resultados.

6.3. Evaluación de los resultados obtenidos

En esta sección se exponen y discuten los resultados obtenidos durante el proceso de experimentación. La idea es optimizar dos sistemas *cloud* utilizando varios MOGAs para la generación de nuevos individuos, junto con MT para comprobar la validez de éstos. Además, estos nuevos individuos se comparan con los obtenidos en el trabajo original de Cañizares et al. (utilizando el algoritmo GA [8]).

En cada experimento realizado utilizamos la notación $\omega_{ga}^{trace}, com = value$, donde *ga* indica el algoritmo utilizado para optimizar el *cloud*, *trace* representa la traza a procesar por el *cloud* y *com* es la configuración del operador de mutación empleado que puede tomar los valores *high*, *mid* o *low* (ver Tabla 6.2).

La Tabla 6.3 muestra el consumo y el tiempo de ejecución requerido por cada sistema *cloud* bajo estudio para procesar las distintas cargas de trabajo.

Workload	*CloudA*	
	Energía (kWh)	Tiempo Ejecución (s)
ω^l	18.7733	2770.10
ω^s	15.4567	2770.10

Workload	*CloudB*	
	Energía (kWh)	Tiempo Ejecución (s)
ω^l	17.0880	2770.10
ω^s	4.6219	970.10

Tabla 6.3: Resultados de los individuos iniciales (*CloudA* y *CloudB*) al procesar ω^l y ω^s

La Figura 6.1 muestra los resultados donde se utiliza *CloudA* para procesar ω^l. Esta figura está formada por 18 gráficas distintas que muestran el consumo de energía y el tiempo de ejecución del mejor individuo de cada iteración. En estas gráficas se emplean tres ejes para la representación de los individuos: el eje y (situado a la izquierda de la gráfica) es la energía consumida, representada por una línea morada; el eje y (situado a la derecha de la gráfica) es el tiempo de ejecución, representado por una línea verde; y el eje x representa la iteración en la que se ha generado el individuo. Cada fila de gráficas utiliza el mismo algoritmo de optimización con los tres parámetros de configuración distintos ($high, mid$ y low), mientras que cada columna muestra la misma configuración de mutación para los distintos algoritmos.

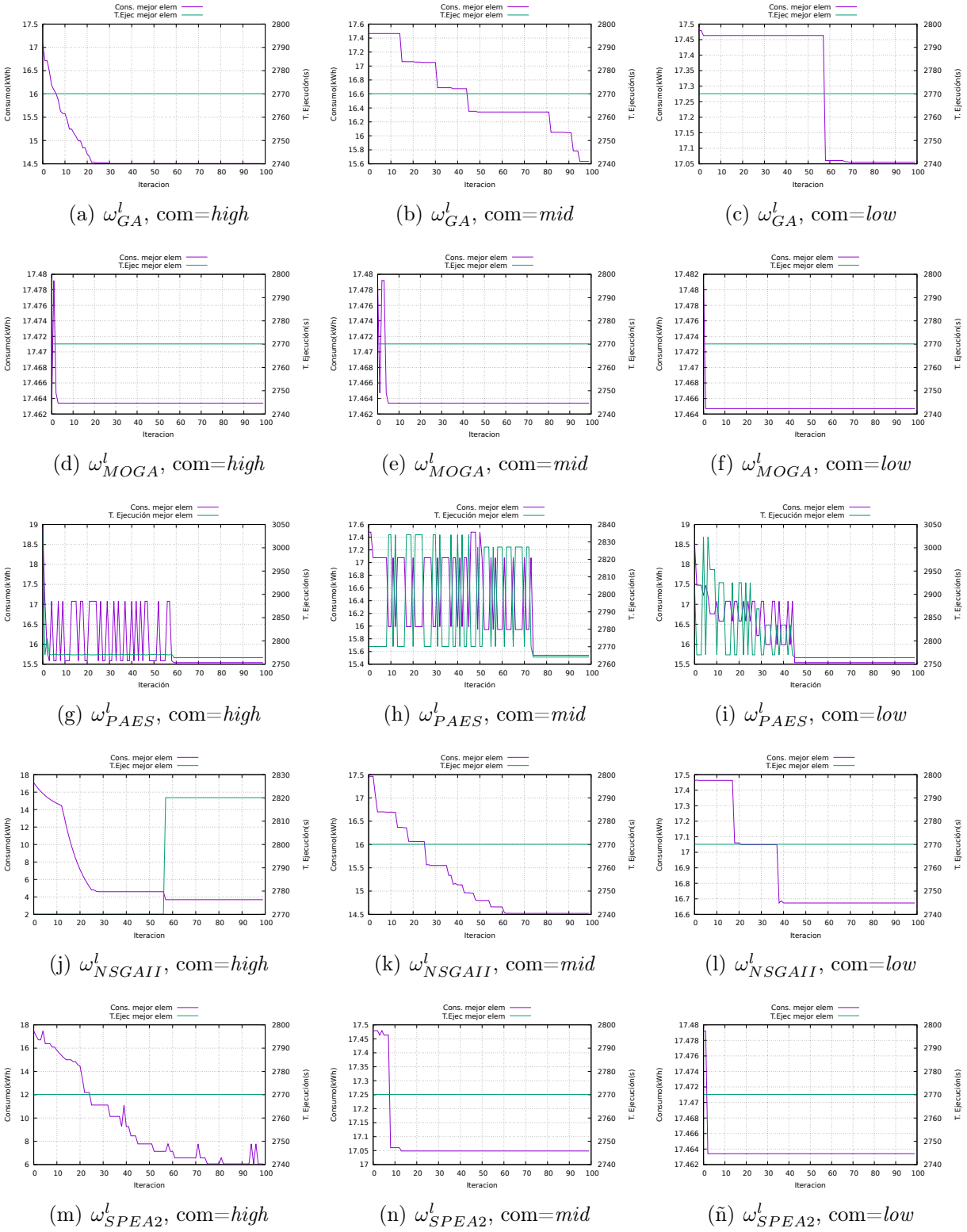
En este experimento todos los algoritmos mejoran el individuo inicial, en este caso, *CloudA* (ver Tabla 6.3). Así pues, observando las gráficas se aprecian dos patrones de comportamiento durante la evolución. En el primero de ellos, algunos algoritmos mantienen el tiempo de ejecución constante mientras disminuyen el consumo. Si analizamos los resultados obtenidos vemos que los algoritmos que presentan este comportamiento son GA, MOGA y SPEA2. Este comportamiento refleja que los algoritmos encuentran con mayor facilidad soluciones que mejoran el consumo energético. Si nos enfocamos en la evolución de GA para (*com=high*) (Figura 6.1.a), se aprecia que durante las primeras iteraciones es cuando sufre una mayor mutación de su mejor individuo hacia un menor consumo para, posteriormente, mantener este individuo hasta el final de las iteraciones. Por parte del algoritmo SPEA2, los resultados muestran unas importantes mejoras en su individuo final con la configuración de mutación más alta (Figura 6.1.m) obteniendo un consumo energético notablemente mejor que el individuo inicial. Para el caso de la configuración de mutación intermedia ($com = mid$) y baja ($com = low$) el GA (figuras 6.1.b y 6.1.c) el mejor individuo se obtiene a partir de la iteración número 50, mientras que en los casos donde se utiliza MOGA (figuras 6.1.e y 6.1.f) y SPEA2 (figuras 6.1.n y 6.1.ñ) se encuentran las principales mejoras de la población en las primeras iteraciones para a continuación no encontrar otras configuraciones que minimicen alguno de los objetivos.

El otro patrón de comportamiento encontrado en la figura muestra variaciones significativas tanto en el consumo como en el tiempo de ejecución. Este suceso se observa en los algoritmos PAES y NSGAII en su configuración de mutación alta. Observando el algoritmo PAES (figuras 6.1.g, 6.1.h y 6.1.i), comprobamos que en las tres configuraciones de mutación establece una alternancia entre mejoras de objetivos, es decir, disminuye el consumo energético a costa de aumentar el tiempo de ejecución y viceversa. Este proceso se repite hasta que finalmente halla un individuo el cual se mantiene como mejor solución durante el resto de la ejecución. En este algoritmo se ve claramente que con la configuración de mutación baja, se llega antes a la mejor solución encontrada. En el caso del algoritmo NSGAII, en su configuración de mutación alta se encuentra un individuo final que aumenta el tiempo de ejecución con respecto al individuo inicial, pero que mejora significativamente el consumo energético de este, reduciéndolo hasta los 3,6831 kW (Figura 6.1.j).

Con estos resultados podemos concluir que la configuración de mutación que mejor resultado obtiene es la configuración de mutación alta. Para esta configuración cabe destacar tres algoritmos que generan individuos finales no dominados entre sí: PAES, NSGAII y SPEA2. Para la configuración de mutación intermedia destacan dos algoritmos que obtienen individuos no dominados entre sí, PAES y NSGAII. Por último, para la configuración de mutación baja el mejor individuo final lo obtiene PAES, dominando al resto de soluciones al mejorar los valores de ambos objetivos.

La Figura 6.2, compuesta por seis gráficas, muestra la evolución del frente de Pareto. Para estas gráficas se emplean 2 ejes en su representación, siendo el eje y el tiempo de ejecución y el eje x el consumo de energía. En este caso se muestran únicamente dos algoritmos, pues en general los algoritmos sólo alcanzan un individuo en su frente de Pareto, mostrando un comportamiento similar al MOGA en esta figura (figuras 6.2.a, 6.2.b y 6.2.c) en el cual al encontrarse únicamente un individuo en el frente de Pareto, simplemente forma una línea de puntos. Así pues, los resultados relevantes se encuentran en el algoritmo PAES, el cual muestra que llega a tener dos individuos no dominados entre sí en el frente de Pareto que genera. Los individuos que obtiene están representados por puntos morados, mientras que cada línea verde representa un frente de Pareto formado. En esta figura se puede apreciar que con la configuración de mutación baja (Figura 6.2.f) se genera un gran número de individuos que disminuyen el consumo energético progresivamente, los individuos de la zona superior, pero cuyo tiempo de ejecución es peor que los individuos más cercanos al eje x. Los individuos aislados, que no están unidos a ninguna línea, quiere decir que dominaron al resto de individuos en esa iteración y, por lo tanto, se mantiene únicamente el nuevo individuo como solución.

En la Tabla 6.4 se presentan los valores del último individuo obtenido en cada uno de los algoritmos al utilizar el *CloudA* para procesar ω^l. Esta tabla sigue la misma distribución que las gráficas, de tal forma



(a) ω^l_{GA}, com=*high* (b) ω^l_{GA}, com=*mid* (c) ω^l_{GA}, com=*low*

(d) ω^l_{MOGA}, com=*high* (e) ω^l_{MOGA}, com=*mid* (f) ω^l_{MOGA}, com=*low*

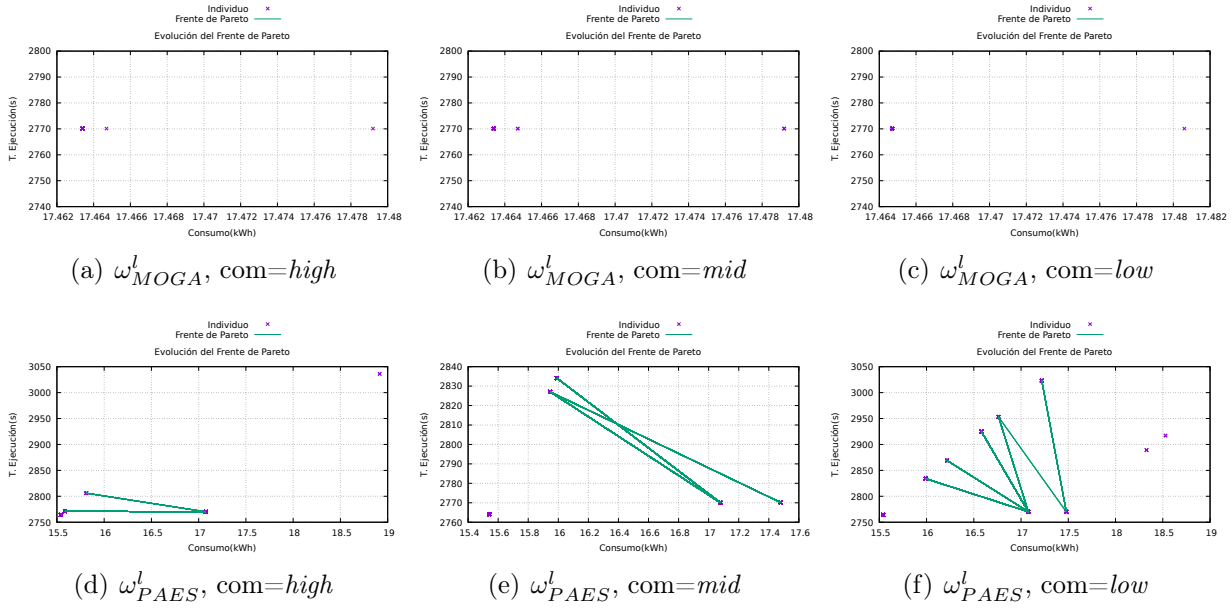
(g) ω^l_{PAES}, com=*high* (h) ω^l_{PAES}, com=*mid* (i) ω^l_{PAES}, com=*low*

(j) ω^l_{NSGAII}, com=*high* (k) ω^l_{NSGAII}, com=*mid* (l) ω^l_{NSGAII}, com=*low*

(m) ω^l_{SPEA2}, com=*high* (n) ω^l_{SPEA2}, com=*mid* (ñ) ω^l_{SPEA2}, com=*low*

Figura 6.1: Evolución de los mejores individuos por iteración para el *CloudA* al procesar ω^l

que en cada columna encontramos la configuración de mutación en la que se ha logrado el individuo, y en cada fila el algoritmo. Adicionalmente, en negrita, se marcan los valores de los individuos que no se dominan entre sí de esta última iteración de los distintos algoritmos. Para este caso, podemos ver cómo los algoritmos llegan a soluciones más dispares en el caso de la configuración *high*, en la que



(a) ω^l_{MOGA}, com=*high* (b) ω^l_{MOGA}, com=*mid* (c) ω^l_{MOGA}, com=*low*

(d) ω^l_{PAES}, com=*high* (e) ω^l_{PAES}, com=*mid* (f) ω^l_{PAES}, com=*low*

Figura 6.2: Evolución del frente de Pareto formado para el *CloudA* al procesar ω^l

encontramos tres individuos que presentan soluciones no dominadas. Por un lado, el algoritmo PAES encuentra una solución que optimiza el tiempo de ejecución sobre el consumo energético, obteniendo un consumo energético inferior al GA y MOGA, que en este caso son los que presentan los peores resultados, y adicionalmente con menor tiempo de ejecución. Por otra parte, el algoritmo NSGAII halla una solución que reduce significativamente el consumo energético, con el coste de aumentar el tiempo de ejecución significativamente sobre el resto de soluciones. Por último, el algoritmo SPEA2 encuentra un individuo entre PAES y NSGAII, el cual, obtiene mejor consumo energético que el GA, MOGA y PAES, pero peor tiempo de ejecución que PAES, y obtiene mejor tiempo de ejecución que NSGAII, pero peor consumo energético. Es por ello, que los individuos finales de PAES, NSGAII y SPEA2 son no dominados entre sí. Por otro lado, para la configuración *mid* encontramos que los individuos no dominados son los obtenidos por PAES y NSGAII. El individuo del algoritmo PAES mejora el tiempo de ejecución frente al de NSGAII, mientras que el individuo de NSGAII mejora el consumo energético respecto al de PAES. Por último, el algoritmo PAES en la configuración *low* obtiene un individuo que domina sobre el resto de individuos, obteniendo tanto mejor consumo energético como mejor tiempo de ejecución.

La Figura 6.3 muestra los resultados de la simulación donde se utiliza el *CloudB*, de mejores prestaciones, para procesar ω^l. En general, se observa que en este experimento el individuo inicial es mejorado en todos los algoritmos (ver Tabla 6.3) y la mejora es significativamente mayor en comparación a la Figura 6.1, indicando de esta forma el impacto que tiene la arquitectura inicial en la ejecución de la traza ω^l. Adicionalmente, esta figura presenta un nuevo patrón de comportamiento, junto a los dos observados

Algoritmo	**Configuración de mutación**					
	high		*mid*		*low*	
	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)
GA	14.5062	2770.10	15.6332	2770.10	17.0553	2770.10
MOGA	17.4634	2770.10	17.4634	2770.10	17.4647	2770.10
PAES	**15.5405**	**2764.10**	**15.5405**	**2764.10**	**15.5405**	**2764.10**
NSGAII	**3.6831**	**2820.10**	**14.5219**	**2770.10**	16.6734	2770.10
SPEA2	**6.0583**	**2770.10**	17.0487	2770.10	17.4634	2770.10

Tabla 6.4: Valores individuos finales para *CloudA* al procesar ω^l

en la Figura 6.1. En esta figura se puede observar que los patrones de comportamiento varían en mayor medida entre configuraciones de mutación, mientras que en la Figura 6.1 dependía en mayormente del algoritmo.

Así pues, el nuevo patrón lo observamos en los casos de la configuración de mutación baja de GA (Figura 6.3.c) y SPEA2 (Figura 6.3.ñ). En este patrón el algoritmo implementado mejora al individuo inicial pues en su población inicial se encontraba un individuo que reducía el consumo energético con respecto a este, pero que en las posteriores iteraciones no mejora en ninguno de los dos objetivos, resultando así en la línea recta generada en las gráficas.

El patrón de comportamiento que mantiene el tiempo de ejecución constante, se encuentra en distintos algoritmos dependiendo de su configuración de mutación. En este caso lo encontramos en el GA, para la configuración de mutación media (Figura 6.3.b), en MOGA, para todas las configuraciones de mutación (figuras 6.3.d, 6.3.e y 6.3.f), en NSGAII, para las configuraciones de mutación media y baja (figuras 6.3.k y 6.3l), y en SPEA2, esta vez en las configuraciones de mutación alta y media (figuras 6.3.m, 6.3.n). Para la configuración de mutación alta, SPEA2 presenta el mejor comportamiento de este patrón, optimizando sustancialmente el individuo inicial logrando una notable disminución del consumo energético, obteniendo un inviduo con un consumo energético de 2,5562 kWh. Sin embargo, en la configuración de mutación media es el GA el que consigue obtener el individuo final con menor consumo energético, con un valor de 13,1786 kWh. Para la configuración de mutación baja, es el algoritmo NSGAII el que presenta el mejor individuo final, disminuyendo la energía hasta los 15,7519 kWh. Por último, MOGA mantiene una convergencia temprana en las tres configuraciones de mutación de forma que rápidamente converge en un individuo que no es mejorado por ningún individuo posterior.

En el caso del patrón que varía significativamente tanto en el consumo como en el tiempo de ejecución encontramos que el GA, en su configuración de mutación más alta (Figura 6.3.a) reduce significativamente el consumo energético respecto al individuo inicial a 2,9653 kWh, aunque esto lo hace a costa de aumentar el tiempo de ejecución. Por otro lado, el algoritmo PAES mantiene su comportamiento

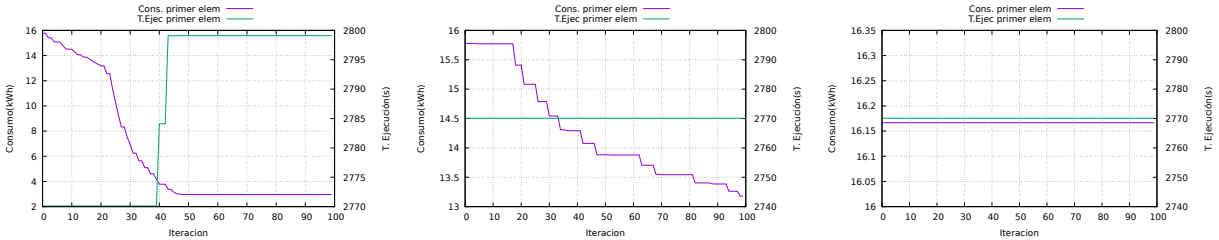
sobre la alternancia de mejoras en los objetivos en las tres configuraciones de mutación (figuras 6.3.g, 6.3.h y 6.3.i). En el caso de la configuración alta la alternancia dura hasta el final de las iteraciones, aunque esta vez aumenta el tiempo de ejecución del individuo final respecto al inicial para disminuir su consumo. En las otras dos configuraciones de mutación converge en un individuo antes de finalizar su ejecución. El último algoritmo que muestra este patrón de comportamiento es NSGAII, el cual muestra una mejora del tiempo de ejecución al principio de sus iteraciones y, a continuación, reduce el consumo energético de forma temprana a un individuo que no variará en iteraciones posteriores.

Así pues, la configuración de mutación que mejores soluciones aporta es *high*, en la que el algoritmo SPEA2 encuentra el mejor individuo, ya que domina a cualquier otra solución hallada, obteniendo el individuo final con menor consumo energético, 2,5562 kWh e iguala o mejora el resto de tiempos de ejecución con 2770 s. Para la configuración intermedia el GA y PAES presentan los mejores individuos, no habiendo una dominación entre los individuos finales generados. Para la configuración de mutación *low* el algoritmo PAES es el que presenta la mejor solución, obteniendo el mínimo tiempo de ejecución 2764 s y un consumo de 14,4083 kWh.

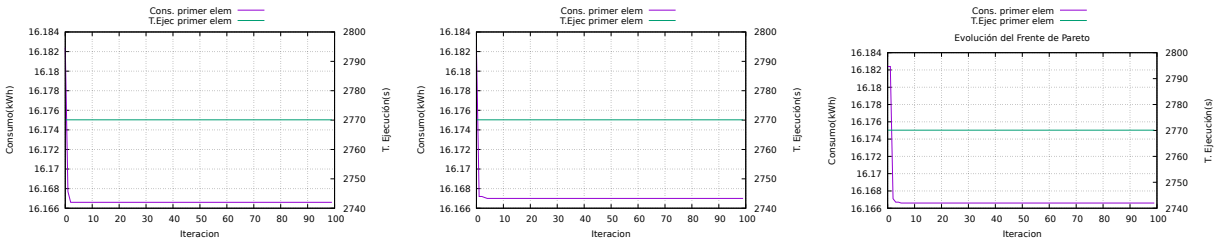
La Figura 6.4 muestra únicamente el algoritmo PAES por simplicidad. Una característica que se presenta en esta figura es que en la configuración de mutación *high* (Figura 6.4.a), esta vez el algoritmo termina con dos individuos como solución, pues no se encuentra ningún punto aislado cerca del centro del eje de coordenadas. Así pues, en la configuración de mutación *mid* es en la que muestra una mayor variación del frente de Pareto, aunque termina obteniendo un individuo que domina las soluciones encontradas en iteraciones previas.

En la Tabla 6.5 se presentan los valores del último individuo obtenido en cada uno de los algoritmos al utilizar el *CloudB* para procesar ω^l. En la configuración de mutación *high* el individuo del algoritmo SPEA2 es el único individuo no dominado, pues presenta un tiempo de ejecución igual o mejor que el resto de individuos y reduce considerablemente el consumo energético respecto al resto de individuos. En la configuración de mutación *mid* GA y PAES son los algoritmos que presentan el mejor rendimiento. GA mantiene el tiempo de ejecución similar al resto de individuos pero reduce el consumo energético respecto al resto. El algoritmo PAES no presenta una mejora significativa en cuanto al consumo energético pero sí que reduce el tiempo de ejecución con respecto al resto de resultados. Por último, en la configuración de mutación *low* encontramos que, de forma similar a la Tabla 6.4, el algoritmo PAES es el que halla el mejor individuo, tanto en consumo energético como en tiempo de ejecución haciendo que el individuo obtenido domine sobre el resto.

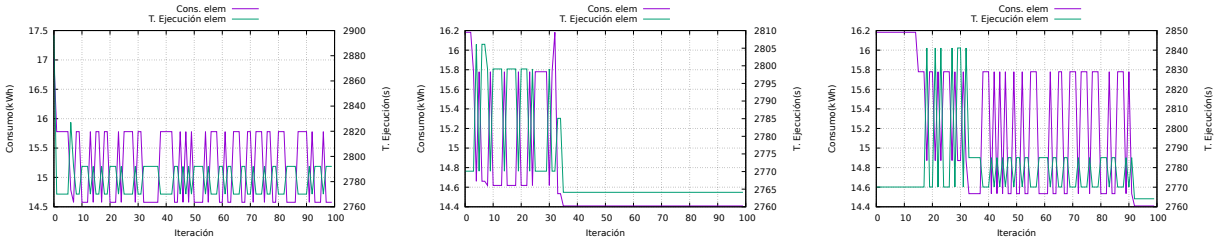
La Figura 6.5 muestra los resultados donde se utiliza *CloudA* para procesar ω^s. En este caso vemos un comportamiento similar al de la Figura 6.1. En esta figura se vuelve a observar el patrón de



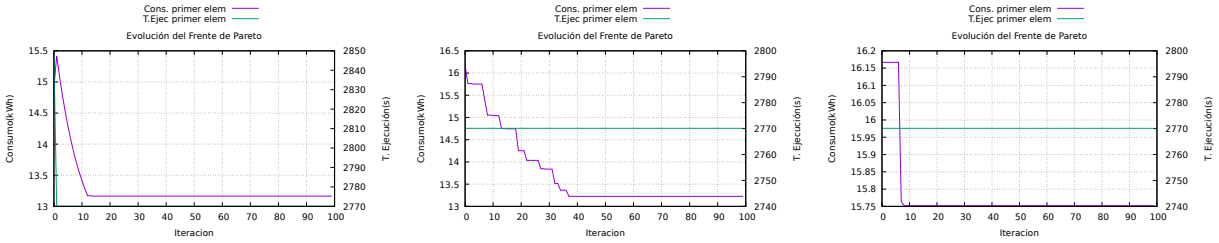
(a) ω^l_{GA}, com=*high* (b) ω^l_{GA}, com=*mid* (c) ω^l_{GA}, com=*low*



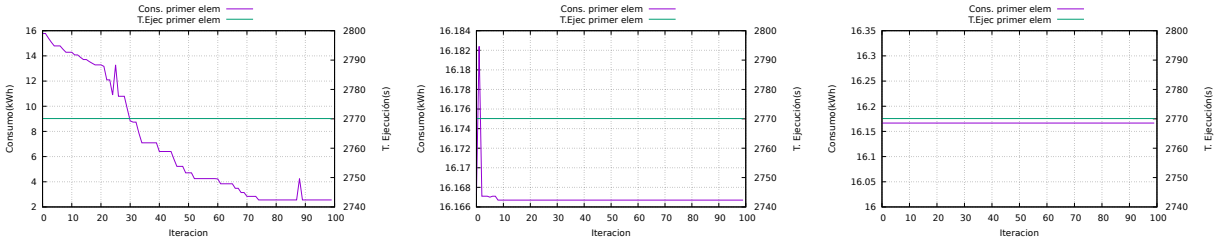
(d) ω^l_{MOGA}, com=*high* (e) ω^l_{MOGA}, com=*mid* (f) ω^l_{MOGA}, com=*low*



(g) ω^l_{PAES}, com=*high* (h) ω^l_{PAES}, com=*mid* (i) ω^l_{PAES}, com=*low*



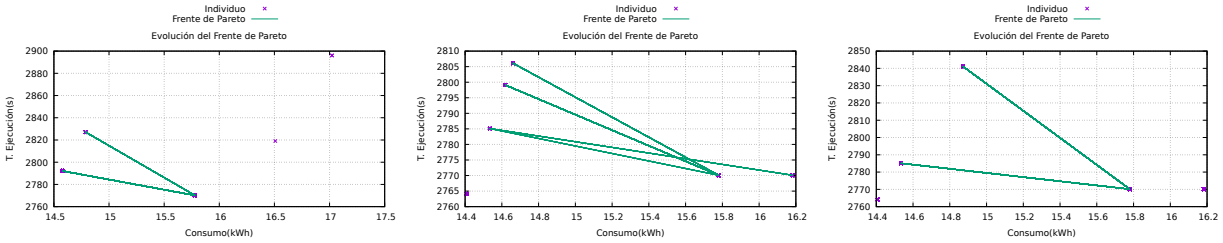
(j) ω^l_{NSGAII}, com=*high* (k) ω^l_{NSGAII}, com=*mid* (l) ω^l_{NSGAII}, com=*low*



(m) ω^l_{SPEA2}, com=*high* (n) ω^l_{SPEA2}, com=*mid* (ñ) ω^l_{SPEA2}, com=*low*

Figura 6.3: Evolución de los mejores individuos por iteración para el *CloudB* al procesar ω^l

comportamiento donde se mejora el consumo energético manteniendo constante el tiempo. Este comportamiento lo podemos ver en los algoritmos GA (figuras 6.5.a, 6.5.b y 6.5.c), NSGAII (figuras 6.5.j, 6.5.k y 6.5.l) y SPEA2 (figuras 6.5.m, 6.5.n y 6.5.ñ). En estos algoritmos el consumo energético alcanza mejoras significativas con respecto a los experimentos anteriores, ya que para la configuración de mu-



(a) ω^l_{PAES}, com=*high* (b) ω^l_{PAES}, com=*mid* (c) ω^l_{PAES}, com=*low*

Figura 6.4: Evolución del frente de Pareto formado para el *CloudB* al procesar ω^l

Algoritmo	**Configuración de mutación**					
	high		*mid*		*low*	
	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)
GA	2.9653	2799.10	**13.1786**	**2770.10**	16.1666	2770.10
MOGA	16.1666	2770.10	16.1670	2770.10	16.1666	2770.10
PAES	14.5765	2792.10	**14.4083**	**2764.10**	**14.4083**	**2764.10**
NSGAII	13.1645	2770.10	13.2260	2770.10	15.7519	2770.10
SPEA2	**2.5562**	**2770.10**	16.1667	2770.10	16.1666	2770.10

Tabla 6.5: Valores individuos finales para *CloudB* al procesar ω^l

tación *high* de estos algoritmos (figuras 6.5.a, 6.5.j y 6.5.m) el consumo se disminuye por debajo de los 8 kWh, reduciendo así en más de un 200 % el consumo del individuo inicial (ver Tabla 6.3).

Por otro lado, el segundo comportamiento que vemos en la figura es aquel que muestra variaciones tanto en el tiempo de ejecución como en el consumo energético. Los algoritmos que encontramos en este caso es el MOGA, para su configuración de mutación *low* (Figura 6.5.f) en la que se aprecia que las mejoras en uno de los dos objetivos se llevan a cabo empeorando el otro, por lo que el individuo final termina mejorando ligeramente el individuo inicial pero con una variación mínima, obteniendo en este caso un consumo de 15,4502 kWh. En el resto de configuraciones, el algoritmo MOGA sigue manteniendo una convergencia temprana con pobres mejoras en el consumo de energía respecto al individuo inicial. El algoritmo PAES mantiene su comportamiento de alternancia entre mejoras, salvo en su configuración de mutación *high* (Figura 6.5.g). En esta gráfica vemos que el individuo obtenido en las primeras iteraciones aumenta el tiempo de simulación notablemente con respecto al individuo inicial, sin embargo, rápidamente mejora la solución actual, reduciendo el consumo energético y el tiempo de ejecución respecto del individuo inicial.

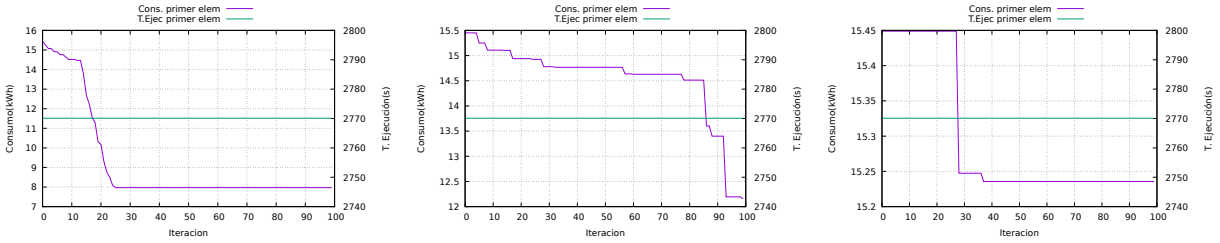
La configuración de mutación *high* se mantiene como la mejor opción en este experimento, presentando los mejores resultados hasta el momento. En esta figura, encontramos dos individuos finales no dominados entre sí. Uno de ellos es el hallado por el algoritmo PAES que posee un consumo energético de 13,5885 kWh y un tiempo de ejecución de 2764 s. El otro individuo dominante con respecto al resto

de algoritmos es el obtenido por SPEA2 y obtiene un individuo con un consumo energético de 4,8785 kWh y un tiempo de ejecución de 2770 s. Para la configuración *mid* los individuos finales no dominados se encuentran en los algoritmos GA y PAES. En el caso de GA mantiene el tiempo del individuo inicial, pero reduce su consumo energético en más de 3 kWh. En el caso del algoritmo PAES el individuo obtenido reduce los valores de aptitud del individuo inicial, aunque en este caso, el consumo energético presenta una menor disminución que el algoritmo GA.

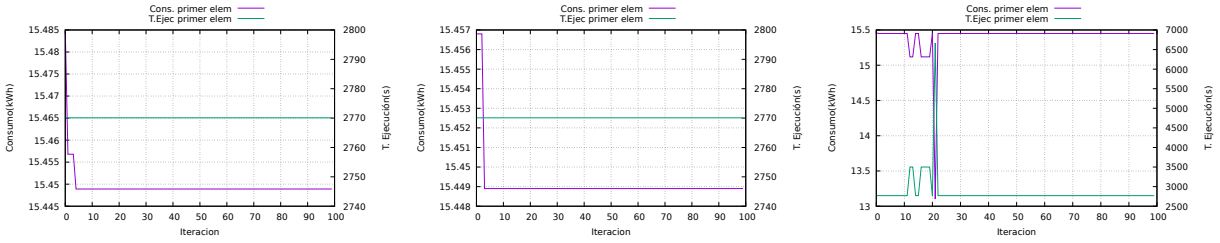
En la Figura 6.6 se muestra la evolución del frente de Pareto en los algoritmos que presentan algún comportamiento singular. En este caso vemos que el algoritmo MOGA contiene dos individuos en el frente de Pareto en el caso de la configuración de mutación *low*, aunque como se ha mostrado en la Figura 6.5 el individuo final es el que en este caso se encuentra en la esquina inferior izquierda. En el caso del algoritmo PAES, las configuraciones de mutación *high* (Figura 6.6.d) y *mid* (Figura 6.6.e) presentan un comportamiento entre individuos similar, aunque en el caso de la configuración de mutación *high* el número de evoluciones es menor. Así pues, vemos que en la configuración *mid* el frente de Pareto termina las iteraciones con dos individuos en él, los cuales poseen un tiempo de ejecución muy similar, como se puede ver por la línea casi paralela al eje x, aunque una diferencia mayor en el consumo energético.

En la Tabla 6.6 se presentan los valores del último individuo obtenido en cada uno de los algoritmos al utilizar el *CloudA* para procesar ω^s. Para este TC el algoritmo que peor resultados presenta en la configuración *high* y *mid* es MOGA pues tanto el consumo como el tiempo de ejecución son mayores o iguales que el resto de individuos. Los algoritmos que mejor resultados presentan con la configuración *high* son PAES y SPEA2. PAES reduce el tiempo de ejecución respecto al resto de individuos, aunque las mejoras sobre el consumo energético no son tan significativas, sin embargo, el algoritmo SPEA2 reduce significativamente el consumo energético respecto al resto de individuos, manteniendo el tiempo de ejecución igual al resto. Para la configuración de mutación *mid* se encuentra únicamente un individuo no dominado, perteneciente al GA original, el cuál reduce el consumo energético en 1 kWh respecto al siguiente individuo pero 3 kWh respecto al individuo inicial. Para la configuración *low* es el algoritmo PAES el que vuelve a encontrar el mejor individuo de los algoritmos. Este individuo mejora al resto de individuos tanto en el consumo energético como en el tiempo de ejecución.

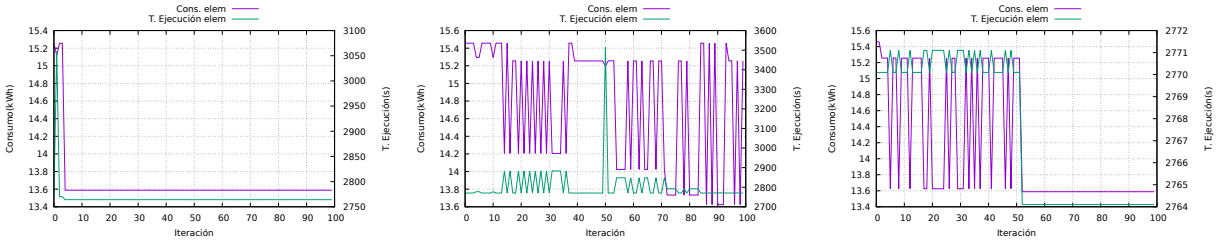
La Figura 6.7 muestra los resultados de la simulación donde se utiliza el *CloudB* para procesar ω^s. Esta figura muestra un comportamiento más unificado en los algoritmos, los cuales muestran en general mejoras sobre el consumo energético sin variaciones en el tiempo de ejecución. El único algoritmo que no sigue este patrón es PAES que sí que muestra más variaciones en el tiempo de ejecución. Aún así, una característica que presenta esta figura es que todos los algoritmos mejoran tanto el tiempo de ejecución como el consumo energético respecto al individuo inicial (ver Tabla 6.3). Esto significa que



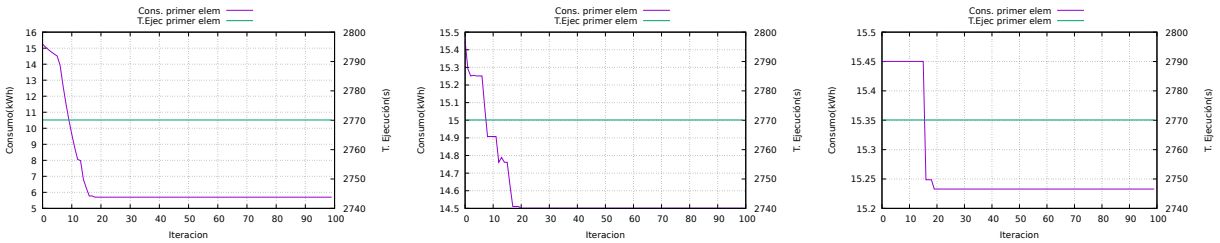
(a) ω^s_{GA}, com=*high* (b) ω^s_{GA}, com=*mid* (c) ω^s_{GA}, com=*low*



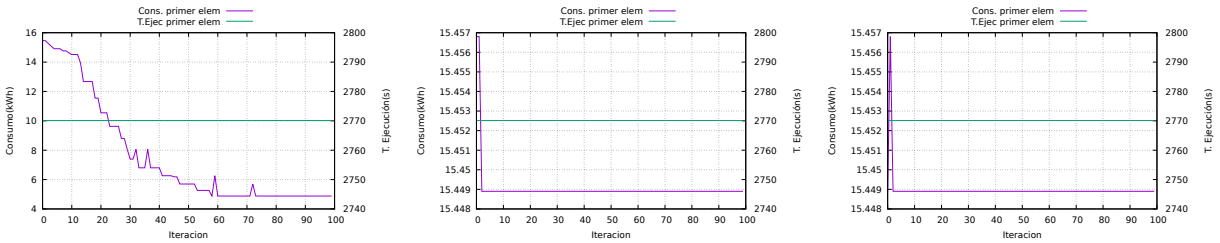
(d) ω^s_{MOGA}, com=*high* (e) ω^s_{MOGA}, com=*mid* (f) ω^s_{MOGA}, com=*low*



(g) ω^s_{PAES}, com=*high* (h) ω^s_{PAES}, com=*mid* (i) ω^s_{PAES}, com=*low*



(j) ω^s_{NSGAII}, com=*high* (k) ω^s_{NSGAII}, com=*mid* (l) ω^s_{NSGAII}, com=*low*

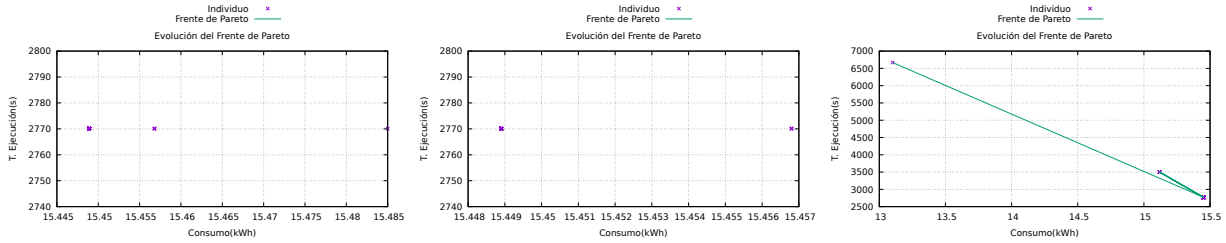


(m) ω^s_{SPEA2}, com=*high* (n) ω^s_{SPEA2}, com=*mid* (ñ) ω^s_{SPEA2}, com=*low*

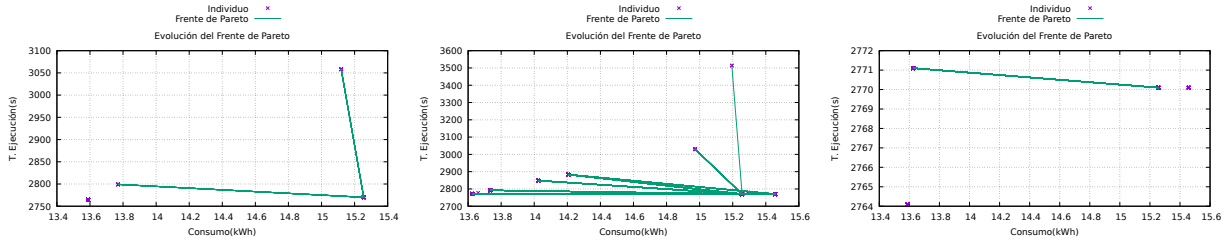
Figura 6.5: Evolución de los mejores individuos por iteración para el *CloudA* al procesar ω^s

una arquitectura inicial de mejores prestaciones, junto con una carga de trabajo menos densa posee un margen de mejora amplio en los objetivos analizados, siendo posible disminuirlos ambos.

Si estudiamos el comportamiento del algoritmo PAES observamos que en este caso, en la configuración *high* (Figura 6.7.g) la variación de tiempo entre los individuos es mínima y por eso hacia el final de



(a) ω^s_{MOGA}, com=*high* (b) ω^s_{MOGA}, com=*mid* (c) ω^s_{MOGA}, com=*low*



(d) ω^s_{PAES}, com=*high* (e) ω^s_{PAES}, com=*mid* (f) ω^s_{PAES}, com=*low*

Figura 6.6: Evolución del frente de Pareto formado para el *CloudA* al procesar ω^s

Algoritmo	**Configuración de mutación**					
	high		*mid*		*low*	
	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)
GA	7.9685	2770.10	**12.1550**	**2770.10**	17.0553	2770.10
MOGA	15.4489	2770.10	15.4489	2770.10	15.4502	2770.10
PAES	**13.5885**	**2764.10**	13.6248	2771.10	**13.5885**	**2764.10**
NSGAII	5.6992	2770.10	14.5009	2770.10	15.2329	2770.10
SPEA2	**4.8785**	**2770.10**	15.4489	2770.10	15.4489	2770.10

Tabla 6.6: Valores individuos finales para *CloudA* al procesar ω^s

las iteraciones presenta una línea casi recta mientras el consumo energético si muestra más variaciones entre los individuos. Así pues, en el caso de las configuraciones *mid* y *low* (figuras 6.7.h y 6.7.i) la característica de la alternancia de mejoras entre objetivos no se cumple, alcanzando una convergencia temprana que consigue estabilizar los valores de los objetivos tras reducirlos con respecto al individuo inicial. Los individuos obtenidos en las configuraciones de mutación *high* y *mid* son idénticos, y poseen un consumo de 1,4832 kWh y un tiempo de ejecución de 691 s. El individuo obtenido en la configuración de mutación *low*, posee un consumo energético ligeramente mayor, de 1,6464 kWh, mientras que el tiempo de ejecución es mínimamente inferior, siendo este de 690 s.

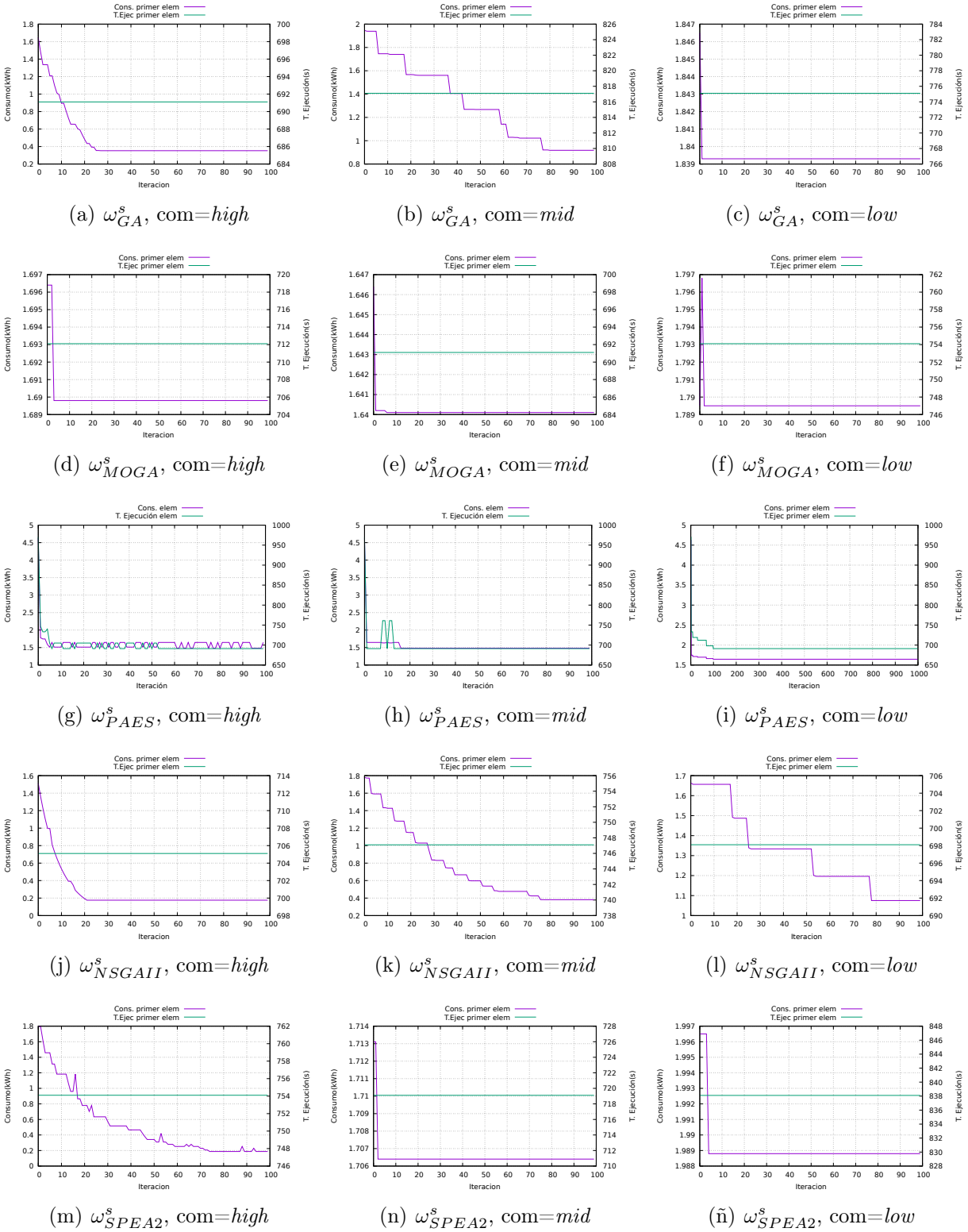
Por otro lado, si nos enfocamos en el comportamiento del resto de algoritmos comprobamos que el tiempo de ejecución, aunque se mantiene constante a lo largo de las iteraciones, este varía entre los algoritmos GA, MOGA, NSGAII y SPEA2, variando también entre las distintas configuraciones de mutación. Así pues, los individuos finales, que mejoran el tiempo de ejecución del individuo inicial, presentan las principales variaciones en el consumo energético, siendo este reducido significativamente

respecto del individuo inicial.

En general, los resultados obtenidos para la configuración de mutación *high* vemos que es la que presenta en general una mayor disminución de la energía consumida, estableciéndose como individuos no dominados el obtenido por el GA (Figura 6.7.a), con un consumo energético de 0,3528 kWh y un tiempo de ejecución de 691 s, y el obtenido por NSGAII (Figura 6.7.j), con un consumo energético de 0,1757 kWh y un tiempo de ejecución de 705 s. Así pues, para la configuración de mutación *mid* encontramos que los algoritmos que generan individuos no dominados por el resto de algoritmos son los obtenidos por PAES (Figura 6.7.h) y NSGAII (Figura 6.7.k), los cuales obtienen dos individuos finales cuyos valores de aptitud son 1,4832 kWh y 691 s, en el caso de PAES, y 0,3816 kWh y 747 s, en el caso de NSGAII. Por último, para la configuración de mutación *low* vuelven a ser estos dos algoritmos los que presentan los mejores individuos finales, con un consumo de 1,6464 kWh y un tiempo de ejecución de 690 s para el individuo obtenido por PAES, y un consumo de 1,0751 kWh y 698 s de tiempo de ejecución para el individuo del NSGAII.

En la Figura 6.8 se muestra la evolución del frente de Pareto en el algoritmo PAES. En este caso, vemos que los individuos iniciales, localizados en la esquina superior derecha de las gráficas, son similares y únicos en el frente de Pareto y a medida que avanzan las iteraciones aparecen individuos que disminuyen los valores de aptitud de los objetivos. En este caso vemos que los individuos se encuentran concentrados en la esquina inferior izquierda, cerca del centro del eje de coordenadas. En los casos de las configuraciones de mutación *high* y *mid* los individuos del frente de Pareto muestran valores de aptitud similares, por ello se encuentran más cerca en la gráfica, y forman frentes de Pareto más compactos en el espacio. Sin embargo, en la configuración de mutación *low* se observa que no se llega a formar un frente de Pareto con más de un individuo, lo cual significa que, en este caso, en las soluciones encontradas por PAES durante su ejecución, sólo ha contenido, como mucho, un elemento a lo largo de las iteraciones.

En la Tabla 6.7 se presentan los valores del último individuo obtenido en cada uno de los algoritmos al utilizar el *CloudB* para procesar ω^s. La tabla muestra en este caso la disparidad de comportamiento que hay entre algoritmos. Aunque de forma general el tiempo de ejecución no varía a lo largo de las iteraciones, como se muestra en la Figura 6.7, este sí que cambia entre algoritmos dependiendo de la población inicial seleccionada. Por ello, se puede ver que entre las distintas configuraciones de mutación ambos valores de aptitud varían notablemente. Además, en esta configuración es en el que más varía el tiempo de ejecución, pues en los anteriores TC el tiempo de ejecución apenas variaba. Esto indica que una inversión en la arquitectura del sistema *cloud* da lugar a mejores optimizaciones en bajas cargas de trabajo. Así pues, en la configuración *high* se encuentran dos individuos no dominados, generados por el algoritmo GA y NSGAII. En este caso, el GA obtiene un individuo cuyo tiempo de ejecución



(a) ω^s_{GA}, com=*high* (b) ω^s_{GA}, com=*mid* (c) ω^s_{GA}, com=*low*

(d) ω^s_{MOGA}, com=*high* (e) ω^s_{MOGA}, com=*mid* (f) ω^s_{MOGA}, com=*low*

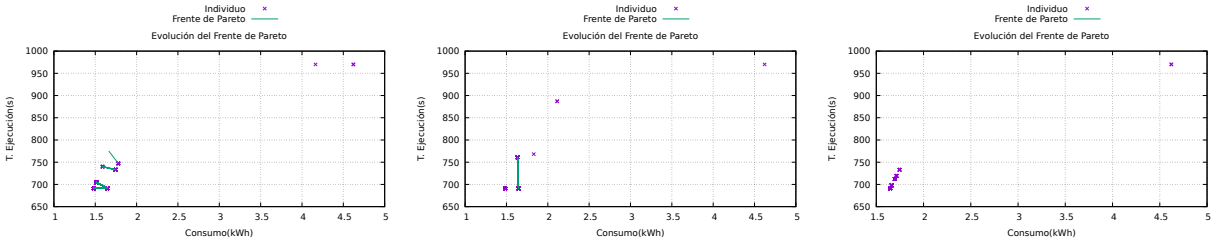
(g) ω^s_{PAES}, com=*high* (h) ω^s_{PAES}, com=*mid* (i) ω^s_{PAES}, com=*low*

(j) ω^s_{NSGAII}, com=*high* (k) ω^s_{NSGAII}, com=*mid* (l) ω^s_{NSGAII}, com=*low*

(m) ω^s_{SPEA2}, com=*high* (n) ω^s_{SPEA2}, com=*mid* (ñ) ω^s_{SPEA2}, com=*low*

Figura 6.7: Evolución de los mejores individuos por iteración para el *CloudB* al procesar ω^s

es menor que el de NSGAII, pero el algoritmo NSGAII reduce notablemente el consumo energético respecto al individuo del GA. En los resultados de la configuración *mid* los individuos que dominan al resto son los obtenidos por PAES y NSGAII. El algoritmo PAES obtiene el mismo individuo que en la configuración *high*, en el cual obtiene el un tiempo de ejecución de 691.1 segundos pero con un consumo



(a) ω^s_{PAES}, com=*high* (b) ω^s_{PAES}, com=*mid* (c) ω^s_{PAES}, com=*low*

Figura 6.8: Evolución del frente de Pareto formado para el *CloudB* al procesar ω^s

Algoritmo	**Configuración de mutación**					
	high		*mid*		*low*	
	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)	Consumo Energético (kWh)	Tiempo ejecución (s)
GA	**0.3528**	**691.10**	0.9179	817.10	1.8393	775.10
MOGA	1.6898	712.10	1.6401	691.10	1.7895	754.10
PAES	1.4832	691.10	**1.4832**	**691.10**	**1.6464**	**690.10**
NSGAII	**0.1757**	**705.10**	**0.3816**	**747.10**	**1.0751**	**698.10**
SPEA2	0.1879	754.10	1.7064	719.10	1.9888	838.10

Tabla 6.7: Valores individuos finales para *CloudB* al procesar ω^s

energético de 1,4832 kWh. Sin embargo, el algoritmo NSGAII, obtiene un consumo energético muy inferior al de PAES a cambio de elevar notablemente el tiempo de ejecución respecto al otro individuo no dominado. Por último, en la configuración *low* los algoritmos que obtienen los individuos dominantes vuelven a ser obtenidos por los algoritmos PAES y NSGAII. En el caso de PAES, mejora en un segundo el individuo de la configuración *mid* a costa de aumentar el consumo energético del individuo. Por otro lado, el algoritmo NSGAII disminuye el tiempo de ejecución de su individuo respecto a la configuración anterior pero casi triplica el consumo energético de este.

En el **https://github.com/techmike17/ImprovingCloudArchitectures**, empleado para el desarroyo de este proyecto, se encuentra el acceso al framework y a los casos de prueba empleados.

Capítulo 7

Conclusiones y trabajo futuro

En este trabajo se ha desarrollado un *framework* que integra algoritmos genéticos multiobjetivo, *metamorphic testing* (MT), y entornos de simulación, para optimizar el sistemas *cloud* teniendo en cuenta el consumo energético y el tiempo de ejecución de la carga de trabajo.

Para lograr este objetivo se ha realizado un estudio previo del trabajo MT-EA4Cloud[8] y de las técnicas de optimización multiobjetivo y MT. Una vez estudiado el problema que se presenta en ese trabajo, y analizadas las ventajas de las técnicas, el reto principal consistía en integrar distintos MOGAs en el *framework* desarrollado para MT-EA4Cloud. Cabe remarcar que las principales dificultades de este proyecto han surgido al combinar los MOGAs con MT y la simulación de sistemas *cloud* en el *framework* propuesto.

El estudio empírico de este proyecto se ha llevado a cabo utilizando dos sistemas *clouds* – diseñados manualmente – con prestaciones distintas y diferentes cargas de trabajo. Para realizar este estudio se ha utilizado el *framework* propuesto con un GA tradicional y cuatro MOGAs diferentes. Todos los algoritmos utilizados se apoyan en técnicas de MT, durante su fase de mutación, para generar individuos apropiados dentro del espacio de soluciones.

Una vez analizados los resultados obtenidos en el estudio empírico, podemos concluir que éstos son satisfactorios. Por un lado, observamos que los algoritmos utilizados se pueden aplicar para optimizar sistemas *cloud* teniendo en cuenta varios objetivos, en este caso, rendimiento y consumo energético. Es importante remarcar que estos resultados muestran que el rendimiento de los algoritmos empleados tiene una fuerte dependencia con la probabilidad de mutación aplicada en cada caso. Los mejores resultados se han obtenido – en la mayor parte de los casos – en las ejecuciones que utilizan la probabilidad de mutación más alta. El algoritmo que mejor comportamiento ha presentado es NSGAII, obteniendo – generalmente – mejores soluciones que el GA y mostrando un comportamiento consistente a lo largo de las iteraciones. Por otro lado, el algoritmo MOGA ha mostrado un peor rendimiento, obteniendo una solución en las iteraciones iniciales que no mejoraba en el resto de la ejecución. Sin embargo,

podemos concluir que combinar algoritmos genéticos multiobjetivo y MT es apropiado para optimizar arquitecturas *cloud*, mejorando de esta forma el proceso de búsqueda de posibles soluciones.

Como trabajo futuro, y para estudiar en mayor profundidad la aplicabilidad y usabilidad del *framework* propuesto, proponemos utilizar distintos simuladores de sistemas *cloud* para llevar a cabo el proceso experimental. Con ello podremos observar un mayor número de características de los sistemas bajo estudio, tales como, por ejemplo, el uso de memoria o el rendimiento del sistema de almacenamiento. Además, para ampliar la funcionalidad del *framework* propuesto, proponemos añadir nuevas MRs, de forma que nos permita expresar de una forma más precisa las propiedades de los sistemas *cloud*, lo cual se verá reflejado en el rendimiento del proceso de testeo.

Capítulo 8

Conclusions and future work

In this work, a framework has been developed unifying multi-objective genetic algorithms (MOGAs), metamorphic testing (MT), and simulation environments to optimize cloud systems, taking into account energy consumption and workload execution time.

To achieve this goal, a preliminary study of the MT-EA4Cloud work[8] and multiobjective optimization techniques and MT was conducted. Once the problem presented in that work was studied, and the advantages of the techniques were analyzed, the main challenge was to integrate different MOGAs into the framework developed for MT-EA4Cloud. It is worth noting that the main difficulties of this project arose when combining MOGAs with MT and simulating cloud systems in the proposed framework.

The empirical study of this project was carried out using two manually designed cloud systems with different performance characteristics and different workloads. To conduct this study, the proposed framework was used with a traditional GA and four different MOGAs. All the algorithms used rely on MT techniques during their mutation phase to generate suitable individuals within the solution space.

After analyzing the results obtained in the empirical study, we can conclude that they are satisfactory. On one hand, we observe that the algorithms used can be applied to optimize cloud systems considering multiple objectives, in this case, performance and energy consumption. It is important to note that these results show that the performance of the algorithms used has a strong dependence on the mutation probability applied in each case. The best results were obtained – in most cases – in the executions that used the highest mutation probability. The algorithm that showed the best performance is NSGAII, generally obtaining better solutions than GA and demonstrating consistent behavior throughout the iterations. On the other hand, the MOGA algorithm showed poorer performance, obtaining a solution in the initial iterations that did not improve in the rest of the execution. However, we can conclude that combining multi-objective genetic algorithms and MT is appropriate for optimizing cloud architectures, thereby improving the search process for possible solutions.

As future work, in order to study the applicability and usability of the proposed framework in greater

depth, we propose using different cloud simulators to conduct the experimental process. This will allow us to observe a greater number of characteristics of the systems under study, such as memory usage or storage system performance. Additionally, to expand the functionality of the proposed framework, we propose adding new metamorphic relations (MRs) so that it can express the properties of cloud systems more precisely, which will be reflected in the performance of the testing process.

Bibliografía

[1] Jai Prakash Verma, Smita Agrawal, Bankim Patel, and Atul Patel. Big data analytics: Challenges and applications for text, audio, video, and social media data. International Journal on Soft Computing, Artificial Intelligence and Applications (IJSCAI), 5(1):41–51, 2016.

[2] HamidReza Asaadi, Dounia Khaldi, and Barbara Chapman. A comparative survey of the hpc and big data paradigms: Analysis and experiments. In 2016 IEEE International Conference on Cluster Computing (CLUSTER), pages 423–432. IEEE, 2016.

[3] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11. Association for Computing Machinery, 2011.

[4] Top500. `https://www.top500.org/lists/top500/2023/06/`.

[5] Avita Katal, Susheela Dahiya, and Tanupriya Choudhury. Energy efficiency in cloud computing data centers: a survey on software technologies. Cluster Computing, pages 1–31, 2022.

[6] Weining Liu and Tao Fan. Live migration of virtual machine based on recovering system and cpu scheduling. In 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, volume 1, pages 303–307, 2011.

[7] Gabriel G. Castañé, Alberto Núñez, Pablo Llopis, and Jesús Carretero. E-mc2: A formal framework for energy modelling in cloud computing. Simulation Modelling Practice and Theory, 39:56–75, 2013.

[8] Pablo C. Cañizares, Alberto Núñez, Juan de Lara, and Luis Llana. MT-EA4Cloud: A Methodology For testing and optimising energy-aware cloud systems. Journal of Systems and Software, 163:110522, 2020.

[9] Ali Vafamehr and Mohammad E Khodayar. Energy-aware cloud computing. The Electricity Journal, 31(2):40–49, 2018.

[10] Nisha Chaurasia, Mohit Kumar, Rashmi Chaudhry, and Om Prakash Verma. Comprehensive survey on energy-aware server consolidation techniques in cloud computing. The Journal of Supercomputing, 77:11682–11737, 2021.

[11] Abhishek Gupta, Laxmikant V. Kale, Filippo Gioachin, Verdi March, Chun Hui Suen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, and Dejan Milojicic. The Who, What, Why, and How of High Performance Computing in the Cloud. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, pages 306–314. IEEE, 2013.

[12] Ashwin Rode, Tamma Carleton, Michael Delgado, Michael Greenstone, Trevor Houser, Solomon Hsiang, Andrew Hultgren, Amir Jina, Robert E Kopp, Kelly E McCusker, et al. Estimating a social cost of carbon for global energy consumption. Nature, 598(7880):308–314, 2021.

[13] Erol Gelenbe and Yves Caseau. The impact of information technology on energy consumption and carbon emissions. Ubiquity, 2015(June):1–15.

[14] Gustavo Pinto and Fernando Castor. Energy efficiency: A new concern for application software developers. Commun. ACM, 60(12):68–75, 2017.

[15] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. An empirical study of practitioners' perspectives on green software engineering. In Proceedings of the 38th International Conference on Software Engineering, ICSE '16, page 237–248. Association for Computing Machinery, 2016.

[16] Atefeh Khosravi and Rajkumar Buyya. Energy and carbon footprint-aware management of geo-distributed cloud data centers: A taxonomy, state of the art, and future directions. Sustainable Development: Concepts, Methodologies, Tools, and Applications, pages 1456–1475, 2018.

[17] Álvaro López García, Enol Fernández-del Castillo, Pablo Orviz Fernandez, Isabel Campos Plasencia, and Jesus Marco de Lucas. Resource provisioning in science clouds: Requirements and challenges. Software: Practice and Experience, 48(3):486–498, 2018.

[18] Karthik Rao, Jun Wang, Sudhakar Yalamanchili, Yorai Wardi, and Handong Ye. Application-specific performance-aware energy optimization on android mobile devices. In 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pages 169–180. IEEE, 2017.

[19] Patrick Kurp. Green computing. Commun. ACM, 51(10):11–13, 2008.

[20] The Green Grid. https://www.thegreengrid.org/.

[21] Green500. https://www.top500.org/lists/green500/.

[22] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. IEEE Transactions on Automatic Control, 31(9):803–812, 1986.

[23] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. Int. J. Comput. Eng. Res. Trends, 351(5):296–304, 2015.

[24] Natallia Kokash. An introduction to heuristic algorithms. Department of Informatics and Telecommunications, pages 1–8, 2005.

[25] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. Computers & Industrial Engineering, 137, 2019.

[26] John H. (John Henry) Holland, 1929. Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence. Complex adaptive systems. MIT Press, 4th printing. edition, 1995. Section: xiv, 211 p. : il. ; 24 cm.

[27] E. J. Weyuker. On Testing Non-Testable Programs. The Computer Journal, 25(4):465–470, 1982.

[28] Mark Harman, Yue Jia, and Yuanyuan Zhang. Achievements, open problems and challenges for search based software testing. In 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pages 1–12, 2015.

[29] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes. A Survey on Metamorphic Testing. IIEEE Trans. Software Eng., 42(9):805–824, 2016.

[30] Sergio Segura, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic Testing: Testing the Untestable. IEEE Softw., 37(3):46–53, 2020.

[31] T. Chen, Shing-Chi Cheung, and Sm Yiu. Metamorphic testing: A new approach for generating next test cases. ArXiv, abs/2002.12543:11, 2020.

[32] Junhua Ding, Dongmei Zhang, and Xin-Hua Hu. An application of metamorphic testing for testing scientific software. In Proceedings of the 1st International Workshop on Metamorphic Testing, MET '16, page 37–43. Association for Computing Machinery, 2016.

[33] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. How effectively does metamorphic testing alleviate the oracle problem? IEEE Transactions on Software Engineering, 40(1):4–22, 2014.

[34] Tjalling C Koopmans. An analysis of production as an efficient combination of activities. Activity analysis of production and allocation, pages 33–97, 1951.

[35] Derek F Lawden. Analytical Methods of Optimization. Courier Corporation, 2006.

[36] B Xu. Research and application of multi-objective optimization algorithms base on differential evolution. East China University of Science and Technology, Shanghai, 2013.

[37] Richard Hamming. Numerical methods for scientists and engineers. Courier Corporation, 2012.

[38] George A Anastassiou and Ioannis K Argyros. Intelligent numerical methods: applications to fractional calculus. Springer, 2016.

[39] Sasmita Behera, Subhrajit Sahoo, and B. B. Pati. A review on optimization algorithms and application to wind energy integration to grid. Renewable and Sustainable Energy Reviews, 48:214–227, 2015.

[40] Anupam Biswas, KK Mishra, Shailesh Tiwari, and AK Misra. Physics-inspired optimization algorithms: a survey. Journal of Optimization, 2013:16, 2013.

[41] Meeta Kumar and Anand J Kulkarni. Socio-inspired optimization metaheuristics: a review. Socio-cultural inspired metaheuristics, pages 241–265, 2019.

[42] Xin-She Yang. Biology-derived algorithms in engineering optimization. Handbook of Bioinspired Algorithms and Applications, page 12, 2010.

[43] Carlos A Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information systems, 1(3):269–308, 1999.

[44] Mohd Nadhir Ab Wahab, Samia Nefti-Meziani, and Adham Atyabi. A comprehensive review of swarm optimization algorithms. PloS one, 10(5), 2015.

[45] V William Porto. Evolutionary programming. In Evolutionary Computation 1, pages 127–140. CRC Press, 2018.

[46] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. Evolution Strategies, pages 871–898. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[47] John H. Holland. Genetic algorithms. Scientific American, 267(1):66–73, 1992.

[48] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4):341–359, 1997.

[49] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new heuristic optimization algorithm: harmony search. simulation, 76(2):60–68, 2001.

[50] Gheorghe Păun. Introduction to membrane computing. In Applications of Membrane Computing, pages 1–42. Springer, 2006.

[51] J. Kennedy and R. Eberhart. Particle swarm optimization. In Proceedings of ICNN'95 - International Conference on Neural Networks, volume 4, pages 1942–1948 vol.4, 1995.

[52] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization, technical report - tr06. Technical Report, Erciyes University, 01 2005.

[53] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. Artificial Life, 5(2):137–172, 1999.

[54] Bing Xue, Mengjie Zhang, and Will N. Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. IEEE Transactions on Cybernetics, 43(6):1656–1671, 2013.

[55] James Kennedy and Russell Eberhart. Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks, volume 4, pages 1942–1948. IEEE, 1995.

[56] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. IEEE Computational Intelligence Magazine, 1(4):28–39, 2006.

[57] Feng Shi and Jingna Lin. Virtual Machine Resource Allocation Optimization in Cloud Computing Based on Multiobjective Genetic Algorithm. Computational Intelligence and Neuroscience, 2022:e7873131, 2022. Publisher: Hindawi.

[58] Mehdi Hosseinzadeh, Marwan Yassin Ghafour, Hawkar Kamaran Hama, Bay Vo, and Afsane Khoshnevis. Multi-Objective Task and Workflow Scheduling Approaches in Cloud Computing: a Comprehensive Review. J Grid Computing, 18(3):327–356, 2020.

[59] Sonia Yassa, Rachid Chelouah, Hubert Kadima, and Bertrand Granado. Multi-Objective Approach for Energy-Aware Workflow Scheduling in Cloud Computing Environments. The Scientific World Journal, 2013:350934, 2013. Publisher: Hindawi Publishing Corporation.

[60] J. Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In Proceedings of the First Int. Conference on Genetic Algortihms, Ed. G.J.E Grefensette, J.J. Lawrence Erlbraum, pages 93–100, 1985.

[61] P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. Structural Optimization, 4(2):99–107, 1992.

[62] C.M. Fonseca and P.J. Fleming. Multiobjective genetic algorithms. In IEE Colloquium on Genetic Algorithms for Control Systems Engineering, pages 6/1–6/5, 1993.

[63] N. Srinivas and Kalyanmoy Deb. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation, 2(3):221–248, 1994.

[64] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, pages 82–87. IEEE, 1994.

[65] T. Murata and H. Ishibuchi. MOGA: multi-objective genetic algorithms. In Proceedings of 1995 IEEE International Conference on Evolutionary Computation, volume 1, page 289. IEEE, 1995.

[66] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Computat., 3(4):257–271, 1999.

[67] David W. Corne, Joshua D. Knowles, and Martin J. Oates. The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization. In Parallel Problem Solving from Nature PPSN VI, volume 1917, pages 839–848. Springer Berlin Heidelberg, 2000. Series Title: Lecture Notes in Computer Science.

[68] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation, 8(2):149–172, 2000.

[69] J. Knowles and D. Corne. The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), pages 98–105. IEEE, 1999.

[70] Haiming Lu and G.G. Yen. Rank-density-based multiobjective genetic algorithm and benchmark test function study. IEEE Trans. Evol. Computat., 7(4):325–343, 2003.

[71] G.G. Yen and Haiming Lu. Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. IEEE Trans. Evol. Computat., 7(3):253–274, 2003.

[72] David W Corne, Nick R Jerram, and Joshua D Knowles. PESA-II: Region-based SeOlepcttiimoniziantiEonvolutionary Multiobjective. page 8.

[73] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In Parallel Problem Solving from Nature PPSN VI, volume 1917, pages 849–858. Springer Berlin Heidelberg, 2000. Series Title: Lecture Notes in Computer Science.

[74] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical report, ETH Zurich, 2001. Artwork Size: 21 p. Medium: application/pdf.

[75] Himanshu Jain and Kalyanmoy Deb. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. IEEE Trans. Evol. Computat., 18(4):602–622, 2014.

[76] Kalyanmoy Deb and Himanshu Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. IEEE Trans. Evol. Computat., 18(4):577–601, 2014.

[77] Laura Diosan and Mihai Oltean. Who's better? PESA or NSGA II? In Proceedings of Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007), pages 869–874. IEEE, 2007.

[78] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8(2):173–195, 2000.

[79] M.T. Jensen. Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms. IEEE Transactions on Evolutionary Computation, 7(5):503–515, 2003.

[80] Abdullah Konak, David W. Coit, and Alice E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety, 91(9):992–1007, 2006.

[81] Yunfei Cui, Zhiqiang Geng, Qunxiong Zhu, and Yongming Han. Review: Multi-objective optimization methods and application in energy saving. Energy, 125:681–704, 2017.

[82] Yiping Liu, Dunwei Gong, Xiaoyan Sun, and Yong Zhang. A reference points-based evolutionary algorithm for many-objective optimization. In Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO Comp '14, page 1053–1056. Association for Computing Machinery, 2014.

[83] Manlin Wang, Yu Zhang, Yan Lu, Xinyu Wan, Bin Xu, and Lei Yu. Comparison of multi-objective genetic algorithms for optimization of cascade reservoir systems. Journal of Water and Climate Change, pages 4069–4086, 2022.

[84] Mingjie Song and Dongmei Chen. An improved knowledge-informed NSGA-II for multi-objective land allocation (MOLA). Geo-spatial Information Science, 21(4):273–287, 2018.

[85] Sergio Nesmachnow. Una Versión Paralela del Algoritmo Evolutivo para Optimización Multiobjetivo NSGA-II. page 12.

[86] Upaka Rathnayake. Review of binary tournament constraint handling technique in NSGA II for optimal control of combined sewer systems. Journal of Information and Optimization Sciences, 37(1):37–49, 2016.

[87] A. Sathya Sofia and P. GaneshKumar. Multi-objective Task Scheduling to Minimize Energy Consumption and Makespan of Cloud Computing Using NSGA-II. J Netw Syst Manage, 26(2):463–485, 2018.

[88] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. Reviews of Modern Physics, 86(1):153, 2014.

[89] Richard P Feynman. Simulating physics with computers. In Feynman and computation, pages 133–153. CRC Press, 2018.

[90] Andrea Burton, Douglas G Altman, Patrick Royston, and Roger L Holder. The design of simulation studies in medical statistics. Statistics in medicine, 25(24):4279–4292, 2006.

[91] Robert E Shannon. Introduction to simulation. In Proceedings of the 24th conference on Winter simulation, pages 65–73, 1992.

[92] James Byrne, Sergej Svorobej, Konstantinos M. Giannoutakis, Dimitrios Tzovaras, Peter J. Byrne, Per-Olov Östberg, Anna Gourinovitch, and Theo Lynn. A review of cloud computing simulation platforms and related environments. In International Conference on Cloud Computing and Services Science, 2017.

[93] Network Simulator NS-2. `https://nsnam.sourceforge.net/wiki/index.php/Main`$_{Page}$.

[94] OMNeT++ Discrete Event Simulator. `https://omnetpp.org/`.

[95] Alberto Núñez, Javier Fernández, Rosa Filgueira, Félix García, and Jesús Carretero. SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications. Simulation Modelling Practice and Theory, 20(1):12–32, 2012.

[96] Heinz Stockinger. Defining the grid: a snapshot on the current view. The Journal of Supercomputing, 42:3–17, 2007.

[97] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 430–437, 2001.

[98] Catalin L Dumitrescu and Ian Foster. Gangsim: a simulator for grid scheduling studies. In CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005., volume 2, pages 1151–1158. IEEE, 2005.

[99] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice and experience, 14(13-15):1175–1220, 2002.

[100] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1):23–50.

[101] Rodrigo N. Calheiros, Marco A.S. Netto, César A.F. De Rose, and Rajkumar Buyya. EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications: EMUSIM: AN INTEGRATED EMULATION AND SIMULATION ENVIRONMENT FOR CLOUDS. Softw. Pract. Exper., 43(5):595–612, 2013.

[102] Sandeep K.S. Gupta, Rose Robin Gilbert, Ayan Banerjee, Zahra Abbasi, Tridib Mukherjee, and Georgios Varsamopoulos. GDCSim: A tool for analyzing Green Data Center design and resource

management techniques. In 2011 International Green Computing Conference and Workshops, pages 1–8. IEEE, 2011.

[103] Simon Ostermann, Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. GroudSim: An Event-Based Simulation Framework for Computational Grids and Clouds. In Euro-Par 2010 Parallel Processing Workshops, volume 6586, pages 305–313. Springer Berlin Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[104] S. Sotiriadis, N. Bessis, N. Antonopoulos, and A. Anjum. SimIC: Designing a New Inter-cloud Simulation Platform for Integrating Large-Scale Resource Management. In 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pages 90–97. IEEE, 2013.

[105] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim, and Chita R. Das. Mdcsim: A multi-tier data center simulation, platform. 2009 IEEE International Conference on Cluster Computing and Workshops, pages 1–9, 2009.

[106] Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. J Grid Computing, 10(1):185–209, 2012.

[107] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. page 21.

[108] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science, pages 1–6, 2009.

[109] Rodrigo N Calheiros, Rajkumar Buyya, and César AF De Rose. Building an automated and self-configurable emulation testbed for grid applications. Software: Practice and Experience, 40(5):405–429, 2010.

[110] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In 2009 international conference on high performance computing & simulation, pages 1–11. IEEE, 2009.

[111] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. Integrating i/os in cloudsim for performance and energy estimation. SIGOPS Oper. Syst. Rev., 50(2):27–36, 2017.

[112] T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, and M. Wieczorek. Askalon: a grid application development and computing environment. In The 6th IEEE/ACM International Workshop on Grid Computing, 2005., pages 10 pp.–, 2005.

[113] Seung-Hwan Lim, Bikash Sharma, Byung Chul Tak, and Chita R Das. A dynamic energy management scheme for multi-tier data centers. In (IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software, pages 257–266. IEEE, 2011.

[114] Vu Le, Mehrdad Afshari, and Zhendong Su. Compiler validation via equivalence modulo inputs. ACM Sigplan Notices, 49(6):216–226, 2014.

[115] WK Chan, Shing Chi Cheung, and Karl RPH Leung. Towards a metamorphic testing methodology for service-oriented software applications. In Fifth International Conference on Quality Software (QSIC'05), pages 470–476. IEEE, 2005.

[116] Alberto Núñez, Pablo C Cañizares, Pablo Gómez-Abajo, Esther Guerra, and Juan de Lara. Analyzing the reliability of simulated distributed systems using Metamorphic Testing. page 8, 2022.

[117] Johannes Mayer and Ralph Guderlei. On random testing of image processing applications. In 2006 Sixth International Conference on Quality Software (QSIC'06), pages 85–92. IEEE, 2006.

[118] TH Tse and Stephen S Yau. Testing context-sensitive middleware-based software applications. In Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004., pages 458–466. IEEE, 2004.

[119] Christian Murphy, Gail E Kaiser, and Lifeng Hu. Properties of machine learning applications for use in metamorphic testing. 2008.

[120] Sergio Segura Rueda, Robert M Hierons, David Felipe Benavides Cuevas, and Antonio Ruiz Cortés. Automated metamorphic testing on the analyses of feature models. Information and Software Technology, 53 (3), 245-258., 2011.

[121] Tsong Yueh Chen, Joshua WK Ho, Huai Liu, and Xiaoyuan Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. BMC bioinformatics, 10(1):1–12, 2009.

[122] Sami Beydeda. Self-metamorphic-testing components. In 30th Annual International Computer Software and Applications Conference (COMPSAC'06), volume 2, pages 265–272. IEEE, 2006.

[123] Tsong Yueh Chen, Jianqiang Feng, and TH Tse. Metamorphic testing of programs on partial differential equations: a case study. In Proceedings 26th Annual International Computer Software and Applications, pages 327–333. IEEE, 2002.

[124] Qiuming Tao, Wei Wu, Chen Zhao, and Wuwei Shen. An automatic testing approach for compiler based on metamorphic testing technique. In 2010 Asia Pacific Software Engineering Conference, pages 270–279. IEEE, 2010.

[125] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. Metamorphic Testing: A Review of Challenges and Opportunities. ACM Comput. Surv., 51(1):1–27, 2019.

[126] Alberto Núñez and Robert M. Hierons. A methodology for validating cloud models using metamorphic testing. Ann. Telecommun., 70(3-4):127–135, 2015.

[127] Alberto Núñez, Pablo C. Cañizares, Manuel Núñez, and Robert M. Hierons. TEA- *Cloud* : A Formal Framework for Testing Cloud Computing Systems. IEEE Trans. Rel., 70(1):261–284, 2021.

[128] Pablo C. Cañizares, Miguel Pérez, Alberto Núñez, and Rosa Filgueira. Validating communication network configurations in cloud and HPC systems using Metamorphic Testing. In 2022 4th International Electronics Communication Conference (IECC), pages 55–62. ACM, 2022.

[129] Megan Olsen and Mohammad Raunak. Increasing Validity of Simulation Models Through Metamorphic Testing. IEEE Transactions on Reliability, 68(1):91–108, 2019. Conference Name: IEEE Transactions on Reliability.

[130] Janette Rounds and Upulee Kanewala. Systematic Testing of Genetic Algorithms: A Metamorphic Testing based Approach, 2018. arXiv:1808.01033 [cs].

[131] Sergio Segura, Javier Troya, Amador Duran, and Antonio Ruiz-Cortes. Performance Metamorphic Testing: Motivation and Challenges. In 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER), pages 7–10. IEEE, 2017.

[132] Sergio Segura, Javier Troya, Amador Durán, and Antonio Ruiz-Cortés. Performance metamorphic testing: A Proof of concept. Information and Software Technology, 98:1–4, 2018.

[133] Gang Lu and Wen Hua Zeng. Cloud Computing Survey. AMM, 530-531:650–661, 2014.

[134] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud Computing: An Overview. page 6.

[135] Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni, and Harold Hall. Cloud computing. IBM white paper, 321:224–231, 2007.

[136] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica, et al. Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California . . . , 2009.

[137] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6):599–616, 2009.

[138] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In 2008 grid computing environments workshop, pages 1–10. Ieee, 2008.

[139] Oracle. `https://www.oracle.com/cloud/hpc/what-is-hpc/`.

[140] Awada Uchechukwu, Keqiu Li, Yanming Shen, et al. Energy consumption in cloud computing data centers. International Journal of Cloud Computing and Services Science (IJ-CLOSER), 3(3):31–48, 2014.

[141] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing: Survey on energy efficiency. ACM Comput. Surv., 47(2), 2014.

[142] Elnaz Parvizi and Mohammad Hossein Rezvani. Utilization-aware energy-efficient virtual machine placement in cloud networks using NSGA-III meta-heuristic approach. Cluster Comput, 23(4):2945–2967, 2020.

[143] Sergi Vila, Fernando Guirado, Josep L. Lerida, and Fernando Cores. Energy-saving scheduling on IaaS HPC cloud environments based on a multi-objective genetic algorithm. J Supercomput, 75(3):1483–1495, 2019.

[144] Aleksandar Bahtovski and Marjan Gusev. Cloudlet challenges. Procedia Engineering, 69:704–711, 2014.

[145] Zhe Ding, Yu-Chu Tian, You-Gan Wang, Wei-Zhe Zhang, and Zu-Guo Yu. Accelerated computation of the genetic algorithm for energy-efficient virtual machine placement in data centers. Neural Comput & Applic, 2022.

[146] Jayati Athavale, Minami Yoda, and Yogendra Joshi. Genetic algorithm based cooling energy optimization of data centers. HFF, 31(10):3148–3168, 2021.

[147] Jayati Athavale, Yogendra Joshi, and Minami Yoda. Artificial neural network based prediction of temperature and flow profile in data centers. In 2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), pages 871–880. IEEE, 2018.

[148] Jayati Athavale, Minami Yoda, and Yogendra Joshi. Comparison of data driven modeling approaches for temperature prediction in data centers. International Journal of Heat and Mass Transfer, 135:1039–1052, 2019.

[149] Eloi Gabaldon, Josep Lluis Lerida, Fernando Guirado, and Jordi Planes. Blacklist muti-objective genetic algorithm for energy saving in heterogeneous environments. J Supercomput, 73(1):354–369, 2017.

[150] Meera Vasudevan, Yu-Chu Tian, Maolin Tang, Erhan Kozan, and Xueying Zhang. Energy-efficient application assignment in profile-based data center management through a Repairing Genetic Algorithm. Applied Soft Computing, 67:399–408, 2018.

[151] Fan-Hsun Tseng, Xiaofei Wang, Li-Der Chou, Han-Chieh Chao, and Victor C. M. Leung. Dynamic Resource Prediction and Allocation for Cloud Data Center Using the Multiobjective Genetic Algorithm. IEEE Systems Journal, 12(2):1688–1699, 2018.

[152] Bahman Keshanchi, Alireza Souri, and Nima Jafari Navimipour. An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing. Journal of Systems and Software, 124:1–21, 2017.

[153] Robert Rönngren and Rassul Ayani. A comparative study of parallel and sequential priority queue algorithms. ACM Trans. Model. Comput. Simul., 7(2):157–209, 1997.

[154] G. Li, M. Li, S. Azarm, J. Rambo, and Y. Joshi. Optimizing thermal design of data center cabinets with a new multi-objective genetic algorithm. Distrib Parallel Databases, 21(2-3):167–192, 2007.

[155] Anindya Chatterjee. An introduction to the proper orthogonal decomposition. Current science, pages 808–817, 2000.

[156] Margaret A Oliver and Richard Webster. Kriging: a method of interpolation for geographical information systems. International Journal of Geographical Information System, 4(3):313–332, 1990.

[157] PM Rekha and M Dakshayini. Efficient task allocation approach using genetic algorithm for cloud environment. Cluster Computing, 22(4):1241–1251, 2019.

[158] KyoungSoo Park and Vivek S. Pai. Comon: A mostly-scalable monitoring system for planetlab. SIGOPS Oper. Syst. Rev., 40(1):65–74, 2006.