Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la Computación

Algoritmos de optimización para la provisión de máquinas virtuales en la nube elástica de Amazon para la ejecución de gemelos digitales de procesos industriales

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de
Maestro en Ciencias

Presenta:

Adrián Facio Medina

Ensenada, Baja California, México
2020

Tesis defendida por

Adrián Facio Medina

y aprobada por el siguiente Comité

Dr. Andrey Chernykh
Director de tesis

Dr. Israel Marck Martínez Pérez

Dr. Raúl Rivera Rodríguez

Dr. Israel Marck Martínez Pérez
Coordinador del Posgrado en Ciencias de la Computación

Dra. Rufina Hernández Martínez
Directora de Estudios de Posgrado

Resumen de la tesis que presenta **Adrián Facio Medina** como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Algoritmos de optimización para la provisión de máquinas virtuales en la nube elástica de Amazon para la ejecución de gemelos digitales de procesos industriales

Resumen aprobado por:

_____

Dr. Andrey Chernykh

Director de tesis

Un Gemelo Digital es un conjunto de modelos computacionales que representan objetos y procesos del mundo físico y reproduce su estado en el mundo digital en tiempo real. Conforme los principales proveedores de equipo industrial adoptan esta tecnología, investigadores han propuesto arquitecturas de gemelos digitales para Fábricas Inteligentes donde diversos gemelos digitales son publicados como microservicios que interactúan entre sí intercambiando información a través de una tecnología de transmisión de datos como Apache Kafka. En este sentido, el problema que enfrenta un tomador de decisiones que elige ejecutar esta carga de trabajo en los recursos computacionales de un proveedor de infraestructura como servicio, es dividir el conjunto gemelo digitales y seleccionar una máquina virtual para cada partición de tal manera que el costo total de renta es minimizado y la demanda de recursos computacionales es satisfecha. Esta tesis propone un conjunto de algoritmos que combinan heurísticas, metaheurísticas, y modelos programación entera para encontrar soluciones de bajo costo. El rendimiento de estos algoritmos es evaluado por medio de un estudio experimental que utiliza los tipos de máquinas virtuales de Amazon EC2 y un conjunto de gemelos digitales cuya demanda de procesador, memoria, y ancho de banda es generada aleatoriamente.

Palabras clave: Aprovisionamiento de máquinas virtuales, Gemelo Digital, Fábrica Inteligente, EC2, Programación Entera, Algoritmos Evolutivos, Metaheurísticas.

Abstract of the thesis presented by **Adrián Facio Medina** as a partial requirement to obtain the Master of Science degree in Computer Science.

Optimization algorithms for Amazon EC2 virtual machine provisioning to execute digital twins of industrial processes.

Abstract approved by:

_____

Dr. Andrey Chernykh
Thesis Director

A Digital Twin is a set of models that represents objects and processes of the physical world and replicates its state in the digital world in real-time. As major industrial equipment vendors adopt this technology, researchers have proposed Smart Factory Digital Twins architectures, where a set of digital twins published as micro-services interact with each other exchanging information by streaming technology such as Apache Kafka. In this sense, the problem faced by a decision-maker that chooses to execute this workload on IaaS provider computational resources is to simultaneously partition the set of digital twins and select a set of virtual machines instances in such a way that computational resources demand is satisfied, and the cost minimized. This thesis proposes a set of algorithms that combine Heuristics, Meta Heuristics, and Mixed Integer Programming to find low-cost solutions. The performance of these algorithms is evaluated with an experimental study of this problem using the Amazon EC2 instances and a set of digital twins with randomly generated bandwidth, memory, and processor requirements.

Keywords: Instance Provisioning, Digital Twin, Smart Factory, EC2, Mixed Integer Programming, Evolutionary Algorithms, Metaheuristics.

Dedicatoria

A Dios, quien me regaló el deseo de aprender y la capacidad para lograr esta meta.

A mi amada esposa Melina, quien ha sido mi compañía y ayuda en los días mas difíciles.

A mis padres, que con gran esfuerzo han salido adelante y con amor me han enseñado a ser un hombre de bien.

A mi hermano Abraham, por enseñarme las primeras líneas de código que despertaron en mi el gusto por la programación de computadoras.

Agradecimientos

Agradezco a Dios por darme salud, fuerzas, y el ánimo necesario para llevar a cabo esta tesis.

Gracias a mi esposa Melina, por su incansable apoyo, por acompañarme en las largas noches y llenar de alegría cada uno de mis dias. Igualmente le doy las gracias a mis padres y hermanos por su generosidad y preocuparse por mi bienestar.

Agradezco a mi director de tesis, el Dr. Andrey Chernykh por creer en mi para ser parte de su equipo de investigación y guiarme en el proceso de la elaboración de mi trabajo de tesis.

A los miembros del comité de tesis, el Dr. Israel Marck Martínez Pérez y el Dr. Raúl Rivera Rodríguez, por su valiosa orientación y retroalimentación que ayudaron a concluir con éxito este trabajo de tesis.

Gracias a mis compañeros de generación, quienes me dieron su amistad, consejos, y buenos momentos que jamás olvidaré. Gracias por siempre ayudarme cuando tuve dificultades y hacer de estos años una grata experiencia.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE) por darme la oportunidad de estudiar bajo la dirección de distinguidos investigadores y proveer un ambiente excelente de aprendizaje y crecimiento personal.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo económico que me permitió dedicarme de tiempo completo a mis estudios de maestría.

Tabla de contenido

Lista de figuras

Lista de tablas

Capítulo 1. Introducción

1.1 Industria 4.0

El mundo globalizado actual presenta el reto de satisfacer una creciente demanda de productos a través de procesos de producción eficientes y que promueven la sustentabilidad social, ambiental, y económica. Actualmente la creación de valor industrial está determinada por el desarrollo de la cuarta etapa de industrialización, llamada Industria 4.0 (Stock & Seliger, 2016).

La materialización de manufactura sustentable puede ser alcanzada por medio de la integración de Tecnologías de la Información y Comunicación (ICT), y dispositivos inteligentes que interactúan a través del Internet Industrial de las Cosas (IIoT) (Stock y Seliger, 2016). Según Radziwon et al., (2014) una Fábrica Inteligente es una solución de manufactura que provee procesos de producción, flexibles y adaptativos que solventan los problemas encontrados en una instalación de producción con condiciones de frontera dinámicas, y cambiantes en un mundo de complejidad creciente. Esta solución especial se relaciona con automatización, entendida como una combinación de software, hardware y/o dispositivos mecánicos, que conducen hacia la optimización del resultado de manufactura en la reducción de labor innecesaria y desperdicio de recursos. Por otro lado, puede ser interpretado bajo la perspectiva de colaboración entre diferentes participantes industriales y no industriales, donde la inteligencia se obtiene de formar una organización dinámica.

Stock y Seliger (2016) argumentan que la inteligencia de un sistema de manufactura se obtiene a través de la colaboración entre sistemas Ciber-Físicos (CPS). Un CPS está basado en componentes mecatrónicos, sistemas de sensores para recolección de datos, capacidad de procesamiento, y sistemas de actuadores para influencian los procesos físicos. Estos sistemas están conectados entre sí y comparten datos a través de redes virtuales como la nube. Con el uso de tecnologías ICT y IoT en la industria se abre paso al monitoreo y sincronización de actividades del mundo real al espacio virtual en tiempo real, gracias a la conexión física-virtual y la interconexión de sistemas ciber-físicos.

1.2 Gemelos Digitales

Un Gemelo Digital (DT) es la contraparte virtual y computarizada de un sistema físico que puede simularlo para varios propósitos y genera una sincronización en tiempo real de los datos obtenidos del mundo real por medio de sensores (Negri et al., 2017). Tao y Zhang (2017) definen al gemelo digital de un área de producción como el espejo digital del mundo físico y lo describen en cuatro niveles: geometría, física, comportamiento y reglas. Usando como ejemplo una máquina CNC, modelos 3D son creados para describir la estructura física del objeto. En segundo lugar, las propiedades físicas (torque, desgaste, y fuerza de corte) pueden ser modelados con Métodos de Elemento Finito. Redes neuronales, máquinas de estado finito, y redes de colaboración pueden ser usados para comportamiento de respuesta a estímulos externos. Por último, reglas de asociatividad, restricciones, y decisión son modeladas para describir el campo de conocimiento y hacer que los niveles antes mencionados colaboren para producir una representación fiel del objeto físico.

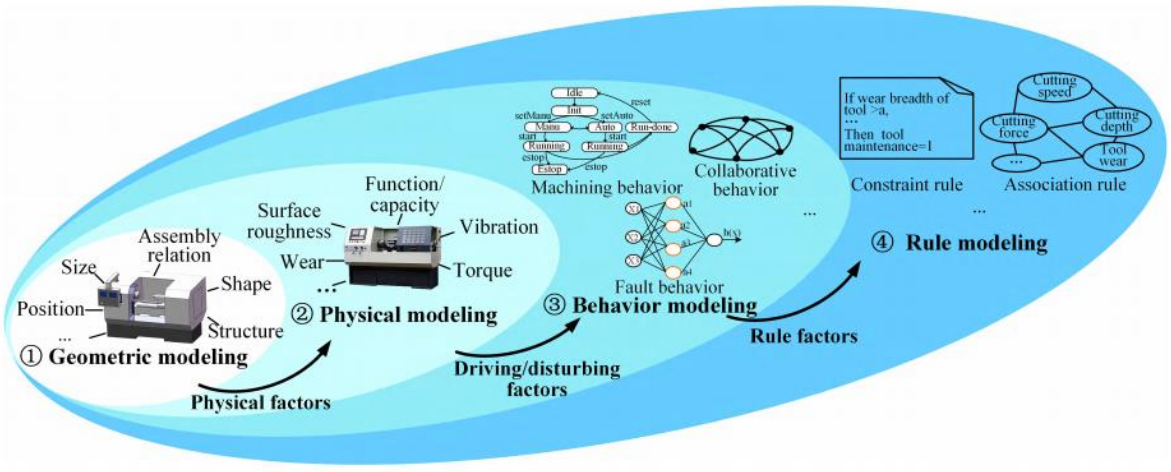

Figura 1. Cuatro niveles de modelado de Gemelo Digital de una máquina CNC. Capa 1: Modelos geométricos. Capa 2: Modelos físicos. Capa 3: Modelos de comportamiento. Capa 4: Modelos de colaboración. Tomado con permiso de "Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing", por Tao y Zhang (2017).

Para permitir a un gemelo digital proveer una sincronización en tiempo real entre el estado del proceso del mundo real, y su copia virtual, es necesario soportarlo con la habilidad de capturar, transferir y analizar los flujos de datos generados en tiempo real por los dispositivos IoT (Radchenko et al., 2019a). Bajo la premisa de que los sistemas ciber-físicos están reemplazando paulatinamente a los Controladores Lógicos Programables (PLC) en los procesos industriales, y que los proveedores de sistemas ciber-físicos dotan a

sus equipos de interfaz digitales y modelos de simulación, investigadores han propuesto que la simulación de un proceso industrial se puede realizar publicando los diferentes modelos de simulación como micro servicios y conectándolos a través de un software intermediario como Apache Kafka, de manera que la arquitectura se puede representar como un DAG donde los vértices representan los modelos y las aristas las dependencias de datos.

En 2015, Srirama y Ostovar proponen un modelo de flujo de trabajo para la implementación de una aplicación en la nube, donde una funcionalidad de usuario está compuesta por servicios que se ejecutan en un orden de dependencia. De manera similar, en la arquitectura propuesta por Tao y Zhang en 2017, el gemelo digital de un área de producción hace uso de servicios compuestos de diversos modelos y datos. Radchenko et al., (2019) plantean el concepto de MicroWorkflows que consiste en descomponer un flujo de trabajo monolítico de Kepler en flujos de trabajo menores con dependencia de datos entre ellos y comunicación a través de Apache Kafka.La Plataforma de Gemelos Digitales como Servicio (Radchenko et al., 2019b) y el Entorno de Simulación MAYA (Ciavotta et al., 2017) son propuestas como arquitecturas de ejecución de gemelos digitales en la nube. Ambas plataformas definen un nivel de IaaS donde se requiere la adquisición bajo demanda de recursos de cómputo en la nube.

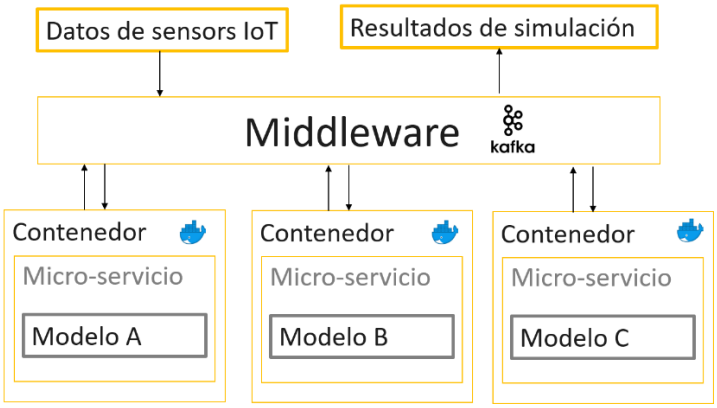

Figura 2. Arquitectura de ejecución de gemelos digitales en la nube. En esta ilustración, Kafka sirve como intermediario entre las entidades que publican y consumen información. Por ejemplo, podemos visualizar un procesamiento en secuencia donde el modelo A consume datos sensados para publicar un resultado. El modelo B a su vez consume el resultado del modelo A y otros datos sensados de su interés para publicar su resultado. Finalmente, el modelo C consume el resultado del modelo B para publicar el resultado final de simulación.

1.3 Cómputo en la nube

El paradigma de Cómputo en la Nube es una tecnología que permite el acceso flexible a recursos de cómputo disponibles. Opera de manera independiente de la ubicación física de los recursos, permitiendo su rápida asignación y reubicación de acuerdo a la demandas del usuario. Los recursos computacionales son generalmente virtualizados y abstraídos, lo cual facilita el desarrollo y publicación de servicios y soluciones, donde el costo de tales es proporcional a la cantidad de recursos utilizados (Dordević et al., 2014). Un ejemplo de este paradigma es La Nube de Cómputo Elástico de Amazon (EC2) que se define como un servicio de Infraestructura como Servicio (IaaS) que ofrece un amplia gama de máquinas virtuales con distinta configuración de memoria RAM, CPU, y sistema operativo, conocidas como tipos de instancias donde usuario puede rentar un conjunto de máquinas virtuales por el tiempo que desee, lo cual permite escalar o reducir la cantidad de acuerdo a la carga de trabajo de una aplicación. Radchenko et al. (2019b) afirman que las múltiples opciones de recursos que ofrece el Cómputo en la Nube permiten satisfacer requerimientos de cómputo heterogéneos de un gemelo digital.

1.4 Antecedentes en métodos de selección de máquinas virtuales

Bhise y Mali, establecen en 2013 que el problema de asignación de recursos que enfrenta un proveedor de servicios de aplicación (SaaS) es determinar el número mínimo de máquinas virtuales y la asignación optima de unidad de servicio de la aplicación en las máquinas virtuales de manera que los acuerdos de nivel de servicio (SLA) no sean violados.

En 2018, Soltani et al. proponen un método donde se elige una máquina virtual por cada unidad de servicio y utiliza medidas de similitud para consultar una base de datos histórica en busca de soluciones aproximadas a los requerimientos computacionales del componente. Si ninguna solución aproximada sobrepasa un porcentaje de similitud, entonces se clasifica el componente como parte de un grupo de la colección máquinas virtuales ofrecidas en la nube, y se elige al mejor vecino dentro de dicho grupo usando TOPSIS como técnica de solución a un modelo de decisión multicriterio. En el trabajo presentado por Srirama y Ostovar en 2015, se propone realizar diversas iteraciones de calendarización EDF (Earliest Deadline First) para asignar cada tarea de un flujo de trabajo a un tipo de máquina de virtual que permita satisfacer la fecha límite. De forma alterna, la selección de máquinas virtuales para la ejecución de un flujo de trabajo G con longitud de ruta crítica L, se puede obtener explorando soluciones con $n \in \{1, \dots, L\}$

máquinas virtuales, donde el tipo de cada máquina virtual es elegido de acuerdo con la distribución del costo-beneficio de las tareas (Convolbo y Chou, 2016).

Una vez elegido un conjunto de máquinas virtuales como infraestructura de cómputo, diversas unidades de servicio pueden ser consolidados en una misma instancia con el propósito de incrementar utilización, disminuir tráfico de red, y ahorrar costo de renta. Una representación utilizada para modelar la elección de unidades de servicio a consolidar es el problema de la mochila para el cual existen algoritmos voraces capaces de encontrar una solución sub-óptima de manera rápida (Soltani et al., 2018). Alternativamente, el sistema descrito en 2015 por Srirama y Ostovar, aprovecha las relaciones de dependencia entre componentes de un flujo de trabajo para agruparlos durante la fase de provisión, y en la etapa de monitoreo asigna las tareas en espera con mayor nivel de urgencia en la capacidad sobrante de las máquinas virtuales provisionadas. De manera similar, el algoritmo de Convolbo y Chou propuesto en 2016 busca agrupar las tareas de manera que minimicen el tiempo total de ejecución; partiendo de una asignación inicial máquinas virtuales la carga de trabajo es balanceada migrando las tareas de las máquinas virtuales con mayor tiempo de ejecución a las máquinas virtuales con menor tiempo de ejecución.

Debido a que el caso general de calendarización de grafos acíclicos dirigidos (DAG) es un problema NP-Difícil (Convolbo y Chou, 2016), múltiples enfoques se han utilizado para encontrar soluciones cercanas al óptimo en tiempo razonable. Las unidades de servicio o tareas son generalmente caracterizadas por el consumo de procesador y la carga de entrada/salida, mientras que los métodos empleados varían desde Programación de Restricciones, Metaheurísticas como Optimización por Enjambre de Partículas, Modelos de Decisión Multi Criterio, y Algoritmos de Aproximación. La tabla 1 muestra un listado representativo de las publicaciones relacionadas con este tema en esta década.

Tabla 1. Trabajos relacionados con minimización de costo en la selección de máquinas virtuales de un IaaS.

Artículo	**Enfoque**	**Criterio Adicional**	**Método**
(Alouane et al., 2018)	Escalabilidad de aplicación	SLA	Programación de restricciones con ILog
(Aldhalaan y Menasce, 2015)	Escalabilidad de cola de trabajo	Calidad de servicio	Búsqueda local
(Mao y Humphrey, 2011)	Escalabilidad de DAG	Violaciones de fecha límite	Programación entera
(Convolbo y Chou, 2016)	DAG	Restricciones de dependencias	Heurística
(Zhou et al., 2016)	DAG	Calidad de servicio probabilístico	A*

(Bhise y Mali, 2013b)	DAG	Violaciones de fecha límite	Heurística
(Bhise y Mali, 2013a)	Carga de trabajo fuera de línea	Restricciones de fecha límite	Calendarización heurística EDF
(Zhang et al., 2014)	MapReduce		Búsqueda en rejilla
(He et al., 2014)	Plataforma de streaming	Calidad de experiencia	Algoritmo de Aproximación para Programación Estocástica de Restricciones
(Valerio et al., 2013)	Carga de trabajo		Programación matemática con restricciones de equilibrio
(Chaisiri et al., 2011)	Cola de trabajo		Optimización robusta, Programación estocástica
(Kritikos y Horn, 2018)	Escalamiento de micro - servicios		Modelo invertido de programación de restricciones
(Soltani et al., 2018)	Escalabilidad de aplicación		Razonamiento de caso base, modelo de decisión multicriterio con TOPSIS
(Zhou et al., 2017)	DAG	Tiempo de respuesta	Programación de Restricciones con WLog usando GPU
(Stephanakis et al., 2013)	DAG en línea	Proporción de Longitud de Calendario	Optimización por enjambre de partículas
(Tian et al., 2015)	Streaming	Calidad de Experiencia	Optimización estocástica de restricciones con Lyapunov

1.5 Hipótesis

- Un gemelo digital construido con la arquitectura de micro servicios puede ser ejecutado en recursos virtuales de una nube minimizando el costo de renta y preservando la calidad de servicio

- Los algoritmos de optimización permiten obtener soluciones de alta calidad para el problema de asignación de máquinas virtuales de un gemelo digital en un tiempo razonable.

1.6 Planteamiento del problema

En el contexto de ejecución de gemelos digitales para la simulación tiempo real de un proceso industrial. El problema de asignación de recursos que enfrenta un proveedor de servicios de gemelos digitales es determinar el número de máquinas virtuales a adquirir de un proveedor de infraestructura como servicio (IaaS), el tipo de cada una de ellas, y la asignación de los componentes del gemelo digital en las máquinas virtuales de manera que minimice el costo de renta y preserve la calidad de servicio.

1.6.1 Definición formal

- Sea $J = \{J_1, \ldots, J_N\}$ un conjunto de N tareas computacionales que componen al gemelo digital donde:

  J_j = (p_j, m_j, b_j),

  p_j = requerimiento de cpu de J_j,

  m_j = requerimiento de memoria de J_j,

  b_j = requerimiento de ancho de banda de J_j.

- Sea $G(V, E)$ un grafo acíclico dirigido donde los vértices V representan las tareas computacionales J que componen al gemelo digital y las aristas E representan la dependencias de datos entre ellas.

- Sea $D_{j,w}$ el ancho de banda requerido para transferir mensajes desde J_j a J_w donde $D_{j,w} = 0$ significa que no existe dependencia de datos.

- Sea $T = \{T_1, \ldots, T_M\}$ el conjunto de tipos de máquinas virtuales donde:

T_i = (P_i, M_i, B_i, C_i),

P_i = capacidad de cpu de T_i,

M_i = capacidad de memoria of T_i,

B_i = capacidad de ancho de banda of T_i,

C_i = costo por hora de rentar una instancia de T_i,

G_i = grupo de tipo de máquina virtual de T_i,

F_i = factor de tamaño de T_i respecto a G_i.

- Sea $Q_i \in \mathbb{Z}$ una variable que indica la cantidad de máquinas virtuales de tipo T_i que serán rentadas.

- Sea $A_{j,i,k} \in \{0,1\}$ una variable que representa la asignación de tareas computacionales en las máquinas virtuales donde

$$A_{j,i,k} = \begin{cases} 1 & \text{si } J_j \text{ es asignado a la instancia } k \text{ del tipo de máquina virtual } T_i, \\ 0 & \text{de otro modo.} \end{cases}$$

  Y se cumple que $\sum_{i=1}^{M} \sum_{k=1}^{Q_i} A_{j,i,k} = 1 \ \ \forall j \in \{1, \ldots, N\}$.

- La sobrecarga de recursos de una instancia k del tipo de máquina virtual T_i, se definen de la siguiente manera:

$$\begin{aligned} O_p(i,k) &= \max\left\{\sum_{j=1}^{N} A_{j,i,k} \cdot p_j - P_i, 0\right\}, \\ O_m(i,k) &= \max\left\{\sum_{j=1}^{N} A_{j,i,k} \cdot m_j - M_i, 0\right\}, \\ O_b(i,k) &= \max\left\{\sum_{j=1}^{N} A_{j,i,k} \cdot b_j - B_i, 0\right\}. \end{aligned}$$

- Sea $SV = \frac{V_p + V_m + V_b}{3}$ la violación del acuerdo de nivel de servicio donde:

$$V_p = \frac{\sum_{i=1}^{M}\sum_{k=1}^{Q_i} O_p\,(i,k)}{\sum_{j=1}^{N} p_j},$$

$$V_m = \frac{\sum_{i=1}^{M}\sum_{k=1}^{Q_i} O_m\,(i,k)}{\sum_{j=1}^{N} m_j},$$

$$V_b = \frac{\sum_{i=1}^{M}\sum_{k=1}^{Q_i} O_b\,(i,k)}{\sum_{j=1}^{N} b_j},$$

- El objetivo del problema es:

$$\text{minimizar } Costo = \sum_{i=1}^{M}\sum_{k=1}^{Q_i}(C_i \cdot \prod_{j=1}^{N} A_{j,i,k})$$

$$\text{sujeto a } SV = 0.$$

1.7 Objetivos

1.7.1 Objetivo general

Diseñar, implementar, y analizar algoritmos de optimización con el fin de asignar las tareas de un gemelo digital a instancias virtuales rentadas en la nube, con el objetivo de disminuir costo y preservar la calidad de servicio.

1.7.2. Objetivos específicos

- Establecer cargas de trabajo de prueba, definiendo requerimientos computacionales de tareas y dependencia de datos entre ellas.

- Revisión bibliográfica.

- Modelar el problema de asignación de tareas de un gemelo digital a recursos virtuales de un

proveedor de IaaS.

- Estudiar estrategias y algoritmos de optimización para el problema de asignación de tareas de un gemelo digital a recursos virtuales de un proveedor de IaaS.
- Evaluar rendimiento de algoritmos estudiados.
- Evaluar la relación de compromiso entre costo de renta y calidad de servicio.

Capítulo 2. Marco teórico

En este capítulo se describe el funcionamiento general de los algoritmos genéticos con enfoque en los mecanismos evolutivos utilizados por los algoritmos de nuestro estudio. El contenido aquí presentado fue basado en el libro *Introduction to Evolutionary Computing* (Eiben y Smith, 2003).

2.1 Algoritmos genéticos

Un algoritmo genético simula un proceso evolutivo donde una población de individuos compiten entre sí para adaptarse al medio ambiente que habitan. La habilidad de un individuo de adaptarse y prosperar en su ambiente es mapeada a un valor numérico a través una función de aptitud. De manera similar a la experiencia natural, a través de las generaciones, nuevos individuos (hijos) son generados a través del intercambio de material genético entre individuos ya existentes (padres), y estos pueden poseer características novedosas causados por la mutación aleatoria de alguno de sus genes. Con el fin de simular la supervivencia de los individuos mas aptos, un mecanismo de selección de padres elige individuos con alto grado de aptitud para ser candidatos a reproducirse. Finalmente, el algoritmo limita el tamaño población empleando mecanimos que determinan cuales individuos serán parte de la siguiente generación.

El resultado esperado de repetir los procesos de cruce, mutación, y selección de padres por varias generaciones es un incremento en la aptitud de la población (Figura 3). En consecuencia, este procedimiento resulta útil para resolver problemas de optimización, donde una solución candidata es representada por un individuo y la función a optimizar determina la aptitud de dicho individuo.

2.1.1 Representación

El primer paso en diseñar un algoritmo genético es la traducción de las propiedades de un objeto que representa una posible solución en el contexto problema real (fenotipo) al espacio de soluciones donde toma lugar el proceso evolutivo (genotipo). Es decir, la representación de un individuo mapea su fenotipo a un genotipo. Por el ejemplo, en el problema de optimizar la función x^2 en el dominio $\{0,1,\dots,31\}$, el

fenotipo $x = 13$ puede ser codificado en forma de cadena binaria en el genotipo 01101. De este modo, los operadores de cruce y mutación operan sobre cadenas binarias para generar nuevos individuos.

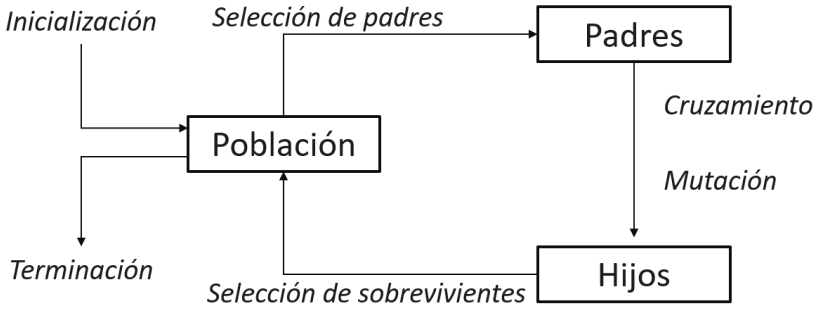

Figura 3. Diagrama de flujo de un algoritmo genético.

Representación binaria

Una cadena binaria es una codificación natural de una solución candidata que representa la elección de un conjunto de elementos. Por ejemplo, en el contexto del problema de maximizar el beneficio de un conjunto de servicios que se pueden contratar dado un presupuesto limitado, donde los servicios ofertados son $\{a, b, c, d, f, g\}$, un individuo $[0,1,0,0,1,0]$ representa la solución que elige los servicios $\{b, f\}$.

Representación entera

Las soluciones de algunos problemas son naturalmente representados como un vector de números enteros. Por ejemplo, en el problema de provisionar un almacén de ropa, el administrador debe elegir la cantidad que debe adquirir de cada producto de su catálogo con el fin de satisfacer la demanda pronósticada y maximizar la ganancia esperada. Suponiendo una lista de productos $\{a, b, c, d\}$, un individuo con la codificación $\{10,0,7,0\}$ representa la solución donde se obtienen 10 artículos de a y 7 de b. Otro tipo de problema que se puede beneficiar de este tipo de codificación, es aquel donde los elementos de la solución pueden tomar un valor de un conjunto finito que tienen interpretación cardinal. Por ejemplo, en el problema de optimizar un camino en una cuadrículo, cada paso puede tomar un valor de $\{arriba, abajo, izquierda, derecha\}$, lo cual puede mapeado con los valores $[1,2,3,4]$. De este modo, la solución $[arriba, arriba, derecha, derecha, abajo]$ es mapeada a la codificación $[1,1,4,4,3]$.

Representación de permutación

Muchos problemas toman la forma de decidir la secuencia en la que ocurre una serie de eventos. Las soluciones de tales problemas son fácilmente codificadas como una permutación de enteros. Por ejemplo, en el problema clásico del vendedor ambulante, una persona debe visitar n ciudades y regresar a su ciudad de origen, la desición que debe realizar es el orden en que visitara las ciudades de manera que minimice la distancia recorrida. En este contexto, la solución $[5,2,1,3,4]$ indica la secuencia de ciudades que debe visitar el vendedor. Finalmente, es importante hacer notar que la utilización exitosa de este tipo de representación depende de emplear operadores variacionales especializados que generan individuos que son permutaciones.

2.1.2 Cruce

El cruce es el proceso de crear un nuevo individuo por medio de la combinación de la información de genotipos existentes. Popularmente, los métodos de cruce emulan la reproducción biológica donde el nuevo individuo es generado a partir de dos progenitores.

Cruce de un punto

En un problema donde las soluciones son de longitud k. Este mecanismo elige un punto aleatorio $x \in \{1, \dots, k\}$ que divide los genotipos del par de individuos que serán reproducidos P_1 y P_2. Posteriormente, el genotipo de un nuevo individuo es formado tomando la parte izquierda del genotipo de P_1 y la parte derecha del genotipo de P_2. Adicionalmente, otro individuo es generado combinando la parte izquierda de P_2 con la parte derecha de P_1.

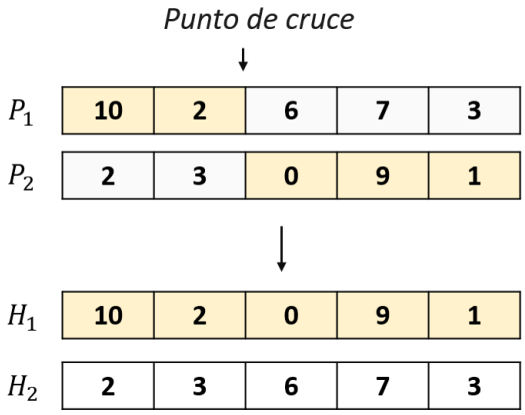

Figura 4. Cruce de un punto. El punto $x = 2$ divide a P_1 en los vectores $[10,2]$ y $[6,3,7]$, de igual manera divide a P_2 en los vectores $[2,3]$ y $[0,9,1]$. El hijo $H_1 = [10,2,0,9,1]$ se forma con el primer vector de P_1 y el segundo vector de P_2. El hijo $H_2 = [2,3,6,7,3]$ se forma con el primer vector de P_2 y el segundo vector de P_1.

Cruce de n puntos

Una generalización del mecanismo descrito anteriormente consiste en elegir n puntos de división de manera aleatoria. De este modo, cada individuo progenitor es representado como un secuencia de $n + 1$ segmentos. Entonces, un nuevo individuo H_1 se forma copiando del primer progenitor los segmentos cuya posición sea un número impar, y copiando del segundo progenitor los segmentos cuya posición sea un número par. De manera simétrica, un hijo H_2 se genera copiando del segundo progenitor los segmentos en posición impar, y copiando del primer progenitor los segmentos cuya posición sea par.

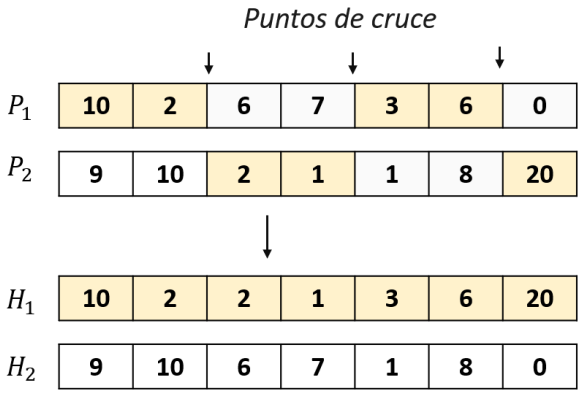

Figura 5. Cruce de $n = 3$ puntos. Los puntos aleatorios dividen a P_1 en los segmentos $[10,2], [6,7], [3,6], [0]$. De igual manera P_2 es divido en los segmentos $[9,10], [2,1], [1,8], [20]$. El hijo $H_1 = [10,2,2,1,3,6,20]$ se forma copiando los segmentos de P_1 en posición impar ($[10,2], [3,6]$), y copiando los segmentos de P_2 en posición par ($[2,1], [20]$). El hijo H_2 se genera de forma simétrica.

Cruce binario simulado (SBX) para enteros

El operador de cruce binario de un punto tiene la propiedad de que el valor promedio de los hijos es igual al valor promedio de los padres, $\frac{P_1+P_2}{2} = \frac{H_1+H_2}{2}$. El operador de cruce binario simulado simulado intenta reproducir este comportamiento con números reales utilizando una distribución de probabilidad similar a la exhibida por el cruce binario. Este operación genera el valor del genotipo de los hijos en la posición i del mezclando los valores del genotipo de los padres en la posición i. El procedimiento descrito por Deb en 2000 e implementado en la librería JMetal para este propósito es el siguiente:

1. Generar un valor aleatorio $s \in [0,1]$. Sea $p = 0.5$ la probabilidad de intercambiar valores. Si $s < p$, entonces se copia $P_{2,i}$ en $H_{1,i}$ y $P_{1,i}$ en $H_{2,i}$, y se concluye la operación. De otro modo, el proceso continúa en el paso 2.

2. Si los valores son muy cercanos entre sí ($|P_{1,i} - P_{2,i}| < 1e^{-14}$), entonces se copian los valores de $P_{1,i}$ en $H_{1,i}$ y $P_{2,i}$ en $H_{2,i}$, y se concluye la operación. De otro modo, el proceso continúa en el paso 3.

3. Sea l_i el valor mínimo que puede tomar el genotipo en la posicion i, y sean $x_i = \min(P_{1,i}, P_{2,i})$, $y_i = \max(P_{1,i}, P_{2,i})$. Se calcula la proporción de extensión $\beta = 1 + 2(x_i - l_i)/(y_i - x_i)$.

4. Calcular el factor de desplazamiento $\alpha = 2 - \beta^{-(\eta+1)}$, donde $\eta = 20$ controla la separación de un hijo con respecto a su padre. Un valor alto de η implica cercanía entre los hijos y los padres.

5. Generar $o \in [0,1]$ aletoriamente. Si $o < 1/\alpha$, entonces se calcula $\beta_q = (o\alpha)^{\frac{1}{\eta+1}}$ como un factor de contracción. De otro modo se define $\beta_q = \left(\frac{1}{2-o\alpha}\right)^{\frac{1}{\eta+1}}$ como un factor de expansión.

6. Sea $c = \frac{1}{2}(x_i + y_i - \beta_q(y_i - x_i))$. Si $c < l$ actualizar $c = l_i$. Sea u_i el límite superior para el gen i, si $c > u_i$, entonces actualizar $c = u_i$.De este modo, c es un valor que será copiado a uno de los hijos.

7. Almacenamos $c_1 = c$. Posteriormente, obtenemos c_2 definiendo $\beta = 1 + 2(u_i - y_i)/(y_i - x_i)$ y repitiendo los pasos 4 al 6.

8. Se elige $z \in [0,1]$ de manera aletoria para determinar la colocación de c_1 y c_2. Si $z \leq 0.5$, entonces c_1 se copia en el primer hijo, y c_2 se copia en el segundo hijo. De lo contrario, los valores son copiados de forma inversa.

Cruce parcialmente mapeado (PMX)

El cruce parcialmente mapedo está diseñado para combinar dos soluciones codificadas como permutaciones de enteros y producir una solución nueva que también es una permutación. Este procedimiento esta constituido por los siguientes pasos:

1. Elegir de manera aleatoria dos posiciones a, b dentro la longitud de la codificación.

2. Copiar en el hijo H_1 los genes del primer progenitor (P_1) contenidos entre las posiciones a y b.

3. Encontrar un elemento del segundo progenitor (P_2) contenido entre las posiciones a y b, que no ha sido copiado en H_1. Sea i la posición de dicho elemento en P_2, entonces el valor contenido en P_1 en la misma posición ($P_{1,i}$). Dado que P_{1_i} ya existe en H_1, entonces el espacio que ocupa dicho valor en P_2, puede estar vacante en el hijo. De aquí, buscamos la posición j tal que $P_{2,j} = P_{1,i}$, si tal posición esta vacante en el hijo, entonces copiamos ahí el valor inicial ($H_{1,j} = P_{2,i}$). Si resulta que esta posición no está vacante, entonces se repite el proceso con punto inicio j en el segundo progenitor.

4. Repetir el paso 3 hasta que dicha etapa no encuentre valores para copiar.

5. Copiar en H_1 cada elemento de P_2 que aún no ha sido reproducido conservando las posiciones de origen.

La creación del segundo hijo se realiza de manera análoga inviertiendo los roles de los progenitores.

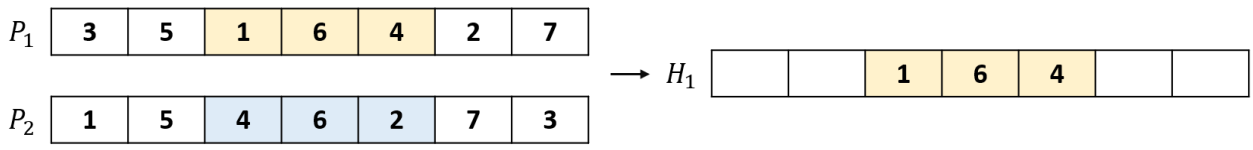

Figura 6. Primer paso de cruce PMX. Dos puntos aleatorios definen segmentos de cruce en ambos progenitores. Posteriormente, los genes del segmento de cruce de P_1 son copiados al hijo H_1.

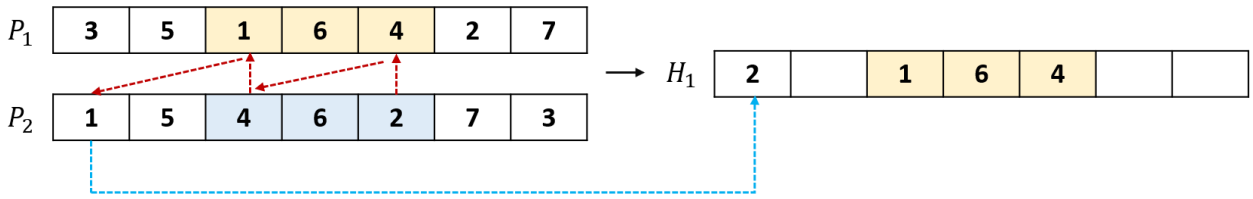

Figura 7. Segundo paso de cruce PMX. El quinto elemento del segundo progenitor ($P_{2,5} = 2$) no existe en H_1. El valor que ocupa esa misma posición en P_1 es 4, por lo que buscamos colocar el valor 2 en la posición ocupa el valor 4 en P_2, esto es $i = 3$. Esta posición ya esta ocupada en H_1, por lo que repetimos el proceso buscando la posición de P_2 que contiene el valor almacenado en la tercera posición del primer progenitor ($P_{1,3} = 1$), esto es $i = 1$. La posición 1 esta vacía en H_1 por lo que es posible asignar el valor 2 en esa posición ($H_{1,1} = 1$).

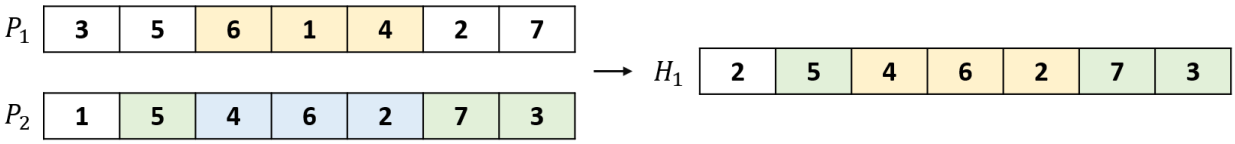

Figura 8. Tercer paso de cruce PMX. Los valores de P_2 que aún no han sido copiados, son reproducidos en H_2 conservando las mismas posiciones.

2.1.3 Mutación

Una mutación es un mecanismo de variación que opera sobre un genotipo para generar una solución con características distintas. Los cambios que se efectúan sobre el genotipo original son de naturaleza estocástica y tienen un rol importante en el nivel de diversidad de una población. Generalmente, las alteraciones se realizan en cada elemento del genotipo, donde dicho elemento es elegido de manera aleatoria con base en la probabilidad definida por una tasa de mutación. Es importante notar que la elección correcta de un mecanismo de mutación depende del tipo de codificación utilizado para representar un individuo.

Mutación polinomial para enteros

Sea $S \in \mathbb{N}^n$ un individuo codificado como un vector de n enteros. Esta operación de mutación altera el elemento $S_i \in [l, u]$ con la adición de un valor $\delta_q \cdot (l - u)$. Es decir, S_i se modifica con un valor proporcional a la cantidad de valores que puede contener. El método para obtener δ_q se describe a continuación. Sean $\delta_1 = \frac{S_i - l}{l - u}$, $\delta_2 = 1 - \delta_1$, y $x \in [0,1]$ un valor elegido aleatoriamente. Si $x \leq 0.5$, entonces $\delta_x = \delta_1$, de otro modo $\delta_x = \delta_2$. Posteriormente, se elige un factor de distribución $\eta = 21$ que

controla la distancia que puede existir entre el valor original de S_i y el valor mutado S_i'. De aquí, definimos $v = 2(1-x) + 2\left(x - \frac{1}{2}\right)\delta_x^{\eta}$, para finalmente obtener $\delta_q = 1 - v^{\frac{1}{\eta}}$.

Mutación de intercambio (SWAP)

Este mecanismo de mutación es adecuado para soluciones codificadas como permutaciones de enteros debido a que el resultado sigue siendo una permutación. Dada una permutación P, se elige un par $i, j \in \{1, \dots, |P|\}$ de manera aleatoria. Posteriormente, los valores de P_i y P_j son intercambiados. Es decir, en la nueva solución P', se tiene que $P_i' = P_j$ y $P_j' = P_i$.

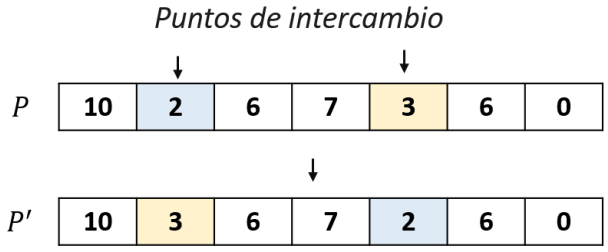

Figura 9. Mutación de intercambio. Los elementos $P_2 = 2$ y $P_5 = 3$ son intercambiados para generar una nueva solución P' donde $P_2' = 3$ y $P_5' = 2$.

2.1.4 Modelos de población

Un aspecto crucial de un proceso evolutivo es la elección de los individuos que sobreviven para competir por los recursos del ambiente y que podrán tener la oportunidad de continuar reproduciendose con base en su aptitud.

Generacional

En este modelo, una población inicia con μ individuos. Posteriormente, se realizan $\frac{\mu}{2}$ cruces para obtener $\lambda = \mu$ hijos. Después, toda la población original es reemplazada por lo hijos para formar la "siguiente generación". De esta manera, en cada generación todos los individuos son nuevos y existe la posibilidad de perder soluciones de alta calidad durante el proceso de reemplazo.

Estado estacionario

En este modelo, la población no es reemplazada en su totalidad. Una población de μ individuos genera $\lambda < \mu$ hijos, por lo que solo una fracción de la población será reemplazada con nuevos individuos. Al porcentaje de población reemplazado en cada generación (λ/μ) se le conoce como "brecha generacional". En consecuencia, el resultado espero de este modelo es un incremento lento de la diversidad de individuos que a su vez tiene mejor probabilidad de conservar soluciones de alta calidad a través de las generaciones.

2.1.5 Selección de progenitores

El mecanismo de selección tiene como propósito incentivar la reproducción de los individuos mas aptos sin eliminar la posibilidad de que los individuos de menor calidad puedan participar en el proceso reproductivo.

Selección con proporción de aptitud

En este esquema, la probabilidad de un individuo i de ser progenitor en un evento de cruce es $P_i = f_i / \sum_{k=1}^{\mu} f_k$. Es decir, esta probabilidad es la proporción del valor absoluto de la función de aptitud de i con respecto a la suma de la aptitud de toda la población. En este sentido, un individuo con mayor aptitud, tiene mayor probabilidad de ser reproducido. El primer paso para implementar este mecanismo en un algoritmo computacional es definir un intervalo $[l_i, u_i]$ para todo individuo i tal que $l_i = u_{i-1}, u_i = l_i + P_i$, donde $l_1 = 0$. Posteriormente, se elige un número aleatorio $x \in [0,1]$, y se elige como progenitor a aquel individuo i donde $l_i \leq x \leq u_i$.

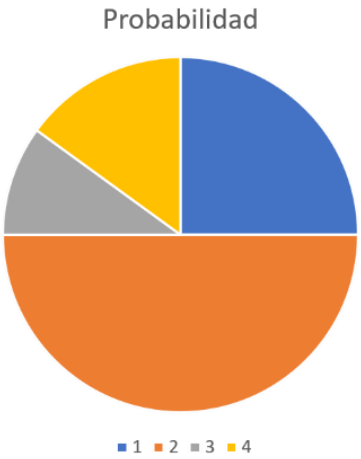

Figura 10. Selección con proporción de aptitud. El individuo 2 tiene probabilidad de 50% de ser elegido debido a que su valor de aptitud corresponde a 50% del total.

Selección con torneo binario

El método de selección por medio de torneos tuvo su origen en los algoritmos evolutivos donde la aptitud de individuo no se puede establecer numéricamente, como es el caso un de proceso evolutivo donde los individuos representan estrategias de juegos y solo se pueden comparar dos individuos por medio de la competencia entre sus estrategias. Adicionalmente, este mecanismo es popularmente usado en los problemas donde el tamaño de la población es tan grande que resulta costoso ejecutar el método de selección con probabilidad de aptitud.

Un torneo binario consiste de dos pasos. El primero es elegir aletoriamente dos individuos de la población. La probabilidad de competir en un torneo es uniforme, por lo que individuos de calidad mediana pueden convertirse en progenitores si participan en un torneo contra invididuos de menor calidad. El segundo paso consiste en determinar cual de los individuos es mejor por medio de la comparación de las funciones de aptitud, o a través de la compentencia entre las estrategias que representan.

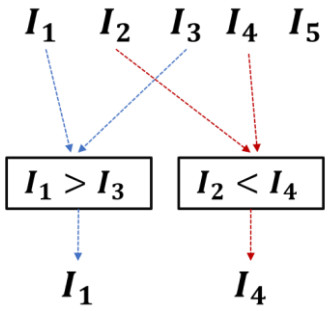

Figura 11. Selección con torneo binario. En el primer torneo compiten los individuos 1 y 3. En el segundo torneo participan los individos 2 y 4. Los ganadores de los torneos (1 y 4) son elegidos como progenitores para una operación de cruce.

Capítulo 3. Algoritmos

En este capítulo se describen diversas estrategias de asignación de tareas computacionales, métodos de consolidación de máquinas virtuales, y algoritmos de optimización diseñados para dar la solución al problema de nuestro estudio. La figura 12 muestra una clasificación de estos algoritmos, donde el primer grupo contiene los algoritmos básicos (sección 3.1 – 3.4) que son utilizados como componentes de algoritmos mas complejos para cumplir las siguientes funciones: 1. Decodificadores en algoritmos genéticos, 2. Etapas de heurísticas híbridas, 3. Generadores de soluciones parciales. El segundo grupo está constituido por algoritmos genéticos descritos en las secciones 3.7 – 3.10. Finalmente, las heurísticas híbridas de las secciones 3.5, 3.6, 3.11 resuelven el problema en etapas usando algoritmos de los grupos anteriores, mientras que el algoritmo de ramificación y acotamiento de la sección 3.12 hace uso de soluciones parciales generadas por una estrategia básica de empaquetado de máquinas virtuales.

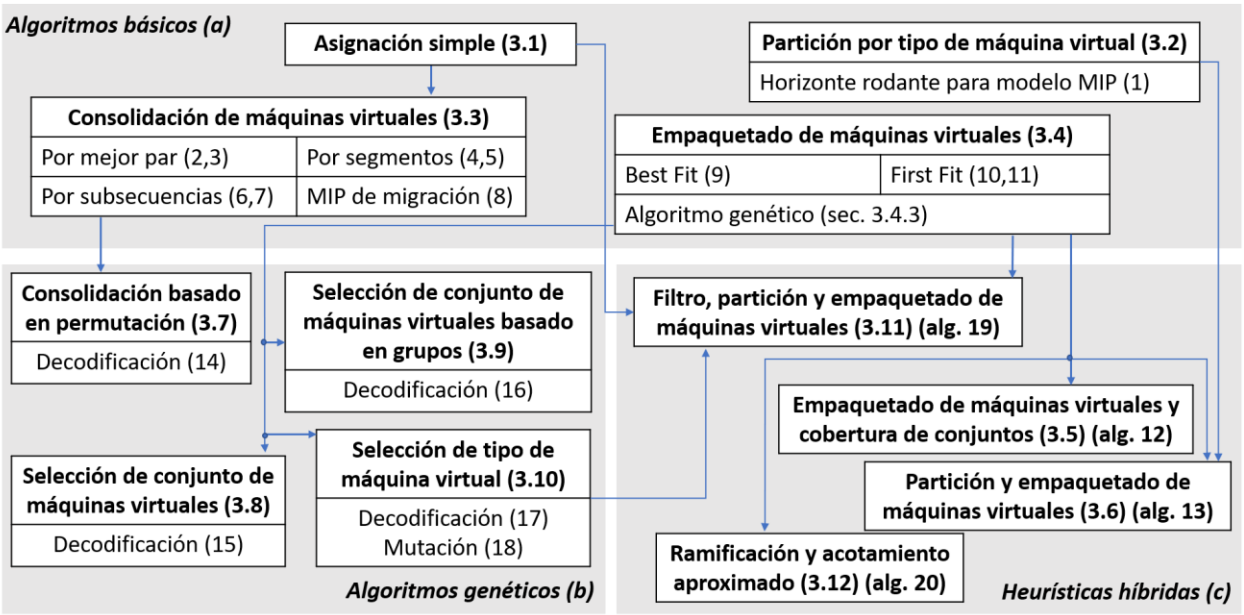

Figura 12. Clasificación de algoritmos. La primer subfigura (a) contiene las estrategias que sirven de componentes básicos para los algoritmos genéticos (subfigura b) y heurísticas híbridas (subfigura c) que dan solución al problema de nuestro estudio. Se puede observar que los algoritmos genéticos de consolidación (sección 3.7) utilizan las estrategias de la sección 3.3 como decodificadores, mientras que los algoritmos genéticos de selección de máquinas virtuales (secciones 3.8, 3.9, y 3.10) utilizan alguna estrategia de empaquetado de máquinas virtuales (sección 3.4) en su procedimiento de decodificación. De manera similar, las primeras tres heurísticas híbridas (3.5, 3.6, 3.11) utilizan algoritmos de los grupos anteriores para crear una solución en etapas, mientras que el algoritmo de ramificación y acotamiento aproximado utiliza empaquetado de máquinas virtuales (3.4) como generador de soluciones parciales.

3.1 Asignación simple

En el problema de nuestro interés, una solución directa es proveer una máquina virtual por cada tarea computacional, de tal manera que el tipo de la máquina virtual sea el de menor costo capaz de satisfacer la demanda de recursos que le fue asignada. La rápidez y simplicidad de esta estrategia le permite ser útil como punto de partida de algoritmos de búsqueda local. Adicionalmente, la solución de un problema a través del método más simple puede ser comúnmente utilizada como una referencia apropiada para medir el rendimiento de los resultados de algoritmos de mayor complejidad, ya que estos deben ser al menos tan buenos como la solución simple.

$$UB = \sum_{j=1}^{N} \min_{i \in \{1,..,M\}} \{C_i \mid p_j \leq P_i, m_j \leq M_i, b_j \leq B_i\}. \quad (1)$$

3.2 Cota inferior

Una cota inferior al costo puede ser obtenida por medio de una versión del problema con restricciones relajadas donde se permite que la demanda de recursos pueda ser distribuida libremente entre las máquinas virtuales provisionadas, y la solución puede ser encontrada fácilmente utilizando el modelo de programación entera mixta 1. La calidad de esta cota es baja ya que no considera si una tarea computacional completa puede ser realmente ejecutada en el conjunto de máquinas virtuales seleccionadas. Por consiguiente, una cota inferior mas cercana al valor óptimo real puede ser generado seleccionando un tipo de máquina virtual por cada tarea computacional, y calcular la cantidad de máquinas virtuales que son necesarias para satisfacer la demanda de recursos. El modelo de programación entera 2 provee la solución de costo mínimo para este planteamiento del problema.

La cantidad de tareas computacionales N establece la complejidad del modelo de programación entera 2 y condiciona la habilidad del software de optimización lineal para encontrar la solución óptima en tiempo razonable. Un enfoque común para obtener soluciones cercanas al óptimo en modelos de programación entera mixta con una cantidad grande de variables es el método del horizonte rodante. Este proceso resuelve una secuencia de submodelos relajados donde sistemáticamente se incrementa la cantidad de variables restringidas a ser enteras y se utilizan los valores enteros de la solución del submodelo previo como valores fijos del submodelo actual. El algoritmo 1 describe la utilización de esta técnica para encontrar soluciones al modelo de programación entera mixta 2.

Modelo 1. Distribución libre de demanda de recursos

$$\text{sea} \quad Copies_i \in \{0, \dots, N\} \qquad \forall\, i \in \{1, \dots, M\}.$$

$$\text{minimizar} \quad \sum_{i=1}^{M} Copies_i C_i,$$

$$\text{s.a} \quad \sum_{j=1}^{N} p_j \leq \sum_{i=1}^{M} P_i \cdot Copies_i\,,$$
$$\sum_{j=1}^{N} m_j \leq \sum_{i=1}^{M} M_i \cdot Copies_i\,,$$
$$\sum_{j=1}^{N} b_j \leq \sum_{i=1}^{M} B_i \cdot Copies_i\,.$$

Modelo 2. Elección de tipo de máquina virtual para cada tarea computacional

$$\text{sea} \quad Copies_i \in \{0, \dots, N\} \quad \forall\, i \in \{1, \dots, M\}\,,$$
$$X_{j,i} \in \{0,1\} \qquad \forall\, j, i \in \{1, \dots, N\} \times \{1, \dots, M\}\,.$$

$$\text{minimizar} \quad \sum_{i=1}^{M} Copies_i C_i\,,$$

$$\text{sujeto a} \quad \sum_{i=1}^{M} X_{j,i} = 1 \qquad \forall\, j \in \{1, \dots, N\}\,,$$
$$\sum_{j=1}^{N} p_j X_{j,i} \leq \sum_{i=1}^{M} P_i \cdot Copies_i \qquad \forall\, i \in \{1, \dots, M\}\,,$$
$$\sum_{j=1}^{N} m_j X_{j,i} \leq \sum_{i=1}^{M} M_i \cdot Copies_i \qquad \forall\, i \in \{1, \dots, M\}\,,$$
$$\sum_{j=1}^{N} b_j X_{j,i} \leq \sum_{i=1}^{M} B_i \cdot Copies_i \qquad \forall\, i \in \{1, \dots, M\}\,,$$
$$X_{j,i} = 0 \qquad \forall\, j, i \in \{1, \dots, N\} \times \{1, \dots, M\}\,,$$
$$P_i < p_j \; or \; M_i < m_j \; or \; B_i < b_j.$$

Algoritmo 1. Horizonte rodante para elección de tipo de máquina virtual

Entrada: Tamaño de horizonte $H \in \mathbb{Z}$

Resultado: Costo total del conjunto de máquinas virtuales seleccionadas

inicialización;

costo:= ∞;

horizonte := *H*;

completo := *falso*;

sea *SOLUCION* una matriz de $N \times M$;

while *completo==falso* **do:**

/* Soluciona submodelo */

prev := *horizonte* − *H*;

h := min{*horizonte,N*};

solve:

$$
\begin{aligned}
& Copias_i \in \{0, \dots, N\} && \forall\, i \in \{1, \dots, M\}, \\
& X_{j,i} \in \{0,1\} && \forall\, j,i \in \{1, \dots, h\} \times \{1, \dots, M\}, \\
& X_{j,i} \in [0,1] && \forall\, j,i \in \{h+1, \dots, N\} \times \{1, \dots, M\}.
\end{aligned}
$$

minimizar $\sum_{i=1}^{M} Copias_i C_i$,

sujeto a

$$
\begin{aligned}
\sum_{i=1}^{M} X_{j,i} &= 1 && \forall\, j \in \{1, \dots, N\}, \\
\sum_{j=1}^{N} p_j X_{j,i} &\le \sum_{i=1}^{M} P_i \cdot Copias_i && \forall\, i \in \{1, \dots, M\}, \\
\sum_{j=1}^{N} m_j X_{j,i} &\le \sum_{i=1}^{M} M_i \cdot Copias_i && \forall\, i \in \{1, \dots, M\}, \\
\sum_{j=1}^{N} b_j X_{j,i} &\le \sum_{i=1}^{M} B_i \cdot Copias_i && \forall\, i \in \{1, \dots, M\}, \\
X_{j,i} &= 0 && \forall\, j,i \in \{1, \dots, N\} \times \{1, \dots, M\}, \\
X_{j,i} &= SOLUCION_{j,i} && P_i < p_j \text{ or } M_i < m_j \text{ or } B_i < b_j, \\
& && \forall\, j,i \in \{1, \dots, prev\} \times \{1, \dots, M\}, \\
& && prev > 0.
\end{aligned}
$$

/* Almacena los valores de la solución encontrada */

$SOLUCION_{j,i} = X_{j,i} \quad \forall\, j,i \in \{1,\dots,h\} \times \{1,\dots,M\}$;

costo := valor de la función objetivo del submodelo;

/* Actualizar horizonte */

if *horizonte* ≥ *N* **then** *completo*:= *verdadero*;

horizonte := *horizonte* + *H*

3.3 Consolidación

Una solución a nuestro problema de asignación de recursos puede ser expresada como un conjunto de máquinas virtuales donde cada uno de ellas aloja un conjunto de tareas computacionales. De este modo, consolidar el contenido de un par de máquinas virtuales en uno sola mejora el costo de la asignación actual si existe un tipo de máquina virtual capaz de satisfacer la demanda computacional de las máquinas virtuales originales a un menor costo (Figura 13).

Sea x un máquina virtual de tipo $x_{vm} \in \{1, \dots, M\}$ con una asignación de tareas computacionales $x_{load} \subset \{1, \dots, N\}\}$, entonces se cumple el grupo de ecuaciones 2. De la misma manera, dado un par de máquinas virtuales x, y, entonces la ecuación 4 define el mejor tipo de máquina virtual para ejecutar la carga de trabajo consolidada $x_{load} \cup y_{load}$.

$$\begin{aligned} x_{cost} &= C_{x_{vm}}, \\ x_{cpu} &= \sum_{j \in x_{load}} p_j\,, \\ x_{memory} &= \sum_{j \in x_{load}} m_j\,, \\ x_{bandwidth} &= \sum_{j \in x_{load}} m_j\,. \end{aligned} \tag{2}$$

$$factible(x, y, z) = \begin{cases} 1 & \text{si} \quad (x_{cpu} + y_{cpu}) \le P_z, \\ & \quad (x_{memory} + y_{memory}) \le M_z, \\ & \quad (x_{bandwidth} + y_{bandwidth}) \le B_z \\ 0 & \text{de otro modo}\,. \end{cases} \tag{3}$$

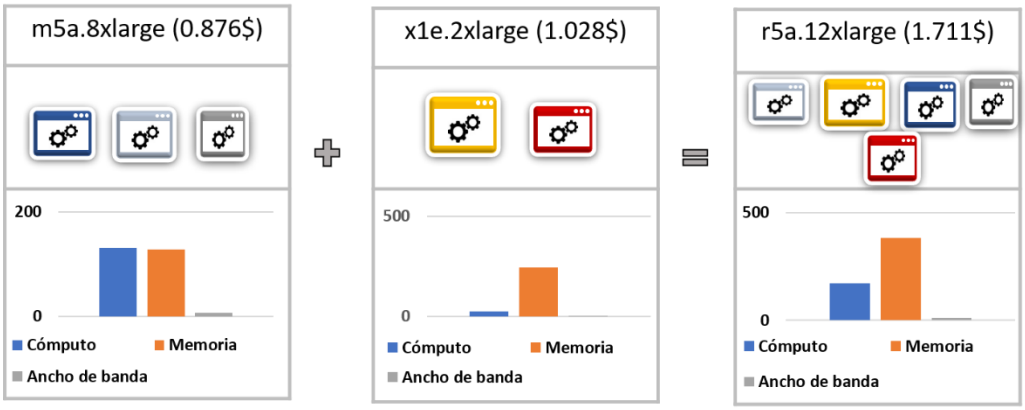

Figura 13. Consolidación de dos máquinas virtuales con carga de trabajo. Con un costo del 90% del total, la instancia r5a.12xlarge es capaz de satisfacer la demanda de trabajo asignada en las instancias m5a.8xlarge y x1e.2xlarge.

$$\arg\max_{z\in 1,\ldots,M}\{C_z - (x_{cost} + y_{cost}) \mid factible(x,y,z) = 1\}. \quad (4)$$

3.3.1 Consolidación del mejor par

Sea B una solución expresada como un conjunto de máquinas virtuales con tareas computacionales asignadas. Este algoritmo de búsqueda local emplea un vecindario que evalúa todos las pares de máquinas virtuales para realizar la consolidación de aquel par que genera la máxima reducción de costo (ecuación 5). La complejidad $O(M|B|^2)$ de una implementación sencilla de este vecindario puede ser reducido a $O(|B|^2)$ utilizando un índice que provee el tipo de máquina virtual de menor costo para una demanda de recursos computacionales específica. El apéndice A.2 muestra los detalles de construcción del índice utilizado. Adicionalmente, el algoritmo 3 implementa el concepto de divide y vencerás que resulta útil para proveer soluciones rápidas a instancias del problema con mas de 5000 tareas computacionales.

$$\arg\max_{x,y,z\in B\times B\times\{1,\ldots,M\},x\neq y} \quad R(x,y,z) = \{(x_{cost} + y_{cost}) - C_z \mid factible(x,y,z) = 1\}. \quad (5)$$

m5a.16xlarge	c4.4xlarge	x1e.2xlarge	m2.4xlarge	x1e.4xlarge	c5.large	m3.large
1.75	0.50	1.02	0.44	2.05	0.05	0.9

r5a.24xlarge	c4.4xlarge	x1e.2xlarge	m2.4xlarge		c5.large	m3.large
3.42	0.50	1.02	0.44		0.05	0.9

r5a.24xlarge	m5a.8xlarge	x1e.2xlarge			c5.large	m3.large
3.42	0.87	1.02			0.05	0.9

r5a.24xlarge	r5a.12xlarge				c5.large	m3.large
3.42	1.71				0.05	0.9

r5a.24xlarge	r5a.12xlarge				m5.xlarge	
3.42	1.71				0.12	

Figura 14. Consolidación del mejor par. En la primera iteración del algoritmo, las máquinas virtuales en la posición 1 y 5 de la lista generan el mayor beneficio al consolidarse en una sola. En la siguientes iteraciones los mejores pares son (2,4), (2,3), y (6,7). El procedimiento finaliza cuando no existen pares de máquinas virtuales que puedan producir reducción de costo al consolidarse.

Algoritmo 2. Consolidación del mejor par

Entrada: Arreglo de máquinas virtuales B, número máximo de iteraciones MAX.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.

$costo = \sum_{b \in B} b_{cost}$
for $i \leftarrow 1$ **to** MAX **do:**
  $candidatos \coloneqq x, y, z \in \{1, \dots, |B|\} \times \{1, \dots, |B|\} \times \{1, \dots, M\}, x \neq y, B_x \neq NIL, B_y \neq NIL;$
  $x, y, z := \arg\max_{x,y,z \in Candidatos} R(B_x, B_y, z) = \{(B_{x_{cost}} + B_{y_{cost}}) - C_z \mid factible(B_x, B_y, z) = 1\}$ **;**
  $costo' = costo - \hat{x}_{cost} - \hat{y}_{cost} + C_z;$
  if $costo' < costo$ **then:**
    b:= consolidación de B_x y B_y en máquina virtual de tipo T_z;
    $B_x \coloneqq b;$
    $B_y \coloneqq NIL;$
    $costo = \sum_{b \in B} b_{cost};$
  else:
    interrumpir iteraciones del ciclo for;
return $costo$;

Algoritmo 3. Divide y vencerás de consolidación del mejor par

Entrada: Lista de máquinas virtuales B,
número máximo de iteraciones MAX,
tamaño de lote de procesamiento TL.
Resultado: Lista de máquinas virtuales B con segmentos consolidados.

$costo = \sum_{b \in B} b_{cost};$
for $i \leftarrow 1$ **to** MAX **do:**
  $G =$ crear grupos de TL elementos de B;
  parallel for $g \in G$ **do**:
    $\hat{B}_g =$ consolidación del mejor par con entrada g; /* Algoritmo 2*/
  $B = \bigcup_{g \in G} B_g;$
  $costo = \sum_{b \in B} b_{cost};$
return $costo$;

3.3.2 Consolidación de segmentos

Sea B una solución expresada como un secuencia de máquinas virtuales que contienen tareas computacionales, el algoritmo 4 consolida segmentos de esta solución que producen una reducción de costo. Tomando como punto de partida alguna máquina virtual, este procedimiento consolidará todas las máquinas virtuales adyacentes posteriores que representan una reducción de costo, una vez que no se encuentran máquinas virtuales adyacentes útiles para consolidar se repite el procedimiento desde ese punto de partida para buscar nuevos segmentos. La complejidad de una implementación sencilla del algoritmo 4 es $O(M|B|)$, sin embargo esta puede ser reducida a $O(|B|)$ utilizando el índice descrito en el

apéndice A.2. Del mismo modo, el algoritmo 5 describe otro enfoque que ofrece reducción de tiempo de ejecución con un penalidad de costo menor, este consiste en considerar solo los tipos de las máquinas virtuales evaluadas, e incluir para cada uno de ellos el primer tipo de máquina virtual del mismo grupo que lo supera en factor de tamaño (ecuación 6).

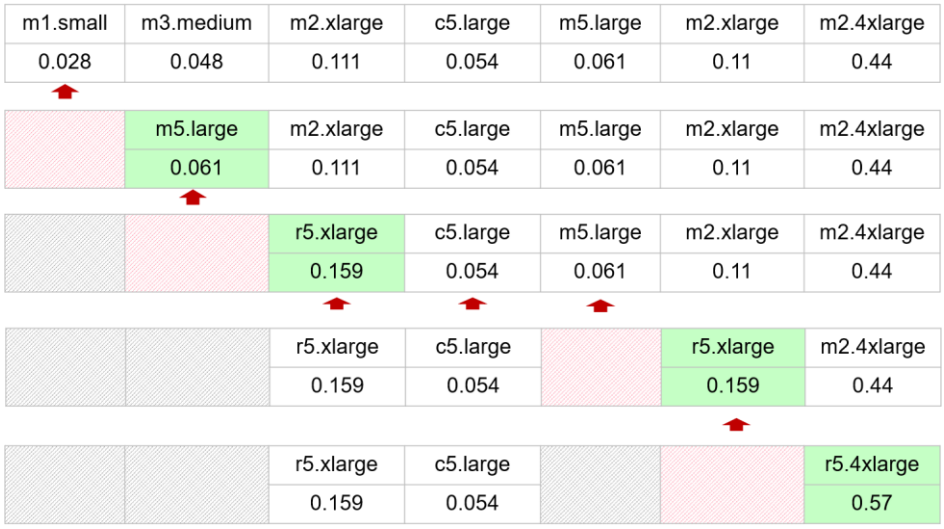

Figura 15. Consolidación de segmentos. Al momento de evaluar la máquina virtual i, el procedimiento determina si este puede ser consolidado con la máquina virtual $i-1$. El segmento de máquinas virtuales 1,2,3 se consolida en una máquina virtual tipo r5.xlarge, la máquina virtual 5 no puede consolidarse con la máquina virtual 4, y el segmento de máquinas virtuales 5,6,7 se consolida en una máquina virtual tipo r5.4xlarge. Al concluir, tres máquinas virtuales contienen todas las tareas computacionales a un costo menor que el original.

$$capacidadMayor(vm) = \underset{z \in \{1,..,M\}}{\operatorname{argmin}} \{F_z \mid G_z = G_{vm}, F_z > F_{vm}\}. \tag{6}$$

Algoritmo 4. Consolidación de segmentos

Entrada: Arreglo de máquinas virtuales B.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.
for $i \leftarrow 2$ **to** $longitud(B)$ **do:**
    $x \coloneqq B_i$;
    $y \coloneqq B_{i-1}$;
    $candidatos \coloneqq \{1, \dots, M\}$;
    $vm \coloneqq \underset{z \in candidatos}{\operatorname{argmin}} \{C_z \mid factible(x, y, z) = 1\}$;
    if $C_{vm} < (x_{cost} + y_{cost})$ **then:**
        $f \coloneqq$ consolida x e y usando en una máquina virtual de tipo vm;
        $B_i \coloneqq f$;
        $B_{i-1} \coloneqq NIL$;

Algoritmo 5. Consolidación de segmentos con búsqueda limitada

Entrada: Arreglo de máquinas virtuales B.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.
for $i \leftarrow 2$ **to** $longitud(B)$ **do:**
    $x \coloneqq B_i$;
    $y \coloneqq B_{i-1}$;
    $candidatos \coloneqq \{x_{vm}, capacidadMayor(x_{vm}), y_{vm}, capacidadMayor(y_{vm})\}$;
    $vm \coloneqq \underset{z \in candidatos}{\operatorname{argmin}} \{C_z \mid factible(x, y, z) = 1\}$;

    if $C_{vm} < (x_{cost} + y_{cost})$ **then:**
        $f \coloneqq$ consolida x e y en una máquina virtual de tipo vm;
        $B_i \coloneqq f$;
        $B_{i-1} \coloneqq NIL$;

3.3.3 Consolidación de subsecuencias

Sea B una solución expresada como un secuencia de máquinas virtuales, el algoritmo 6 consolida subsecuencias de B que producen una reducción de costos. El procedimiento definiendo el primer elemento de B como la máquina virtual acumuluadora A y se remueve de B, en los siguientes pasos se evalúa iterativamente el resto de máquinas virtuales de la secuencia para consolidar con la máquina virtual acumuladora A aquellas que implican disminución de costos y removerlas de B, al finalizar la máquina virtual acumuladora se guarda en una lista. En la siguientes iteraciones, la secuencia contiene máquinas virtuales que no fueron consolidadas, por lo que se repite el proceso anterior, iniciando con la primer máquina virtual disponible.

La complejidad de una implementación simple del algoritmo 6 es $O(M|B|^2)$, sin embargo esta puede ser reducida a $O(|B|^2)$ utilizando el índice descrito en el apéndice A.2. Del mismo modo, el algoritmo 7 describe una variante de este procedimiento donde la elección de tipos de máquinas virtuales para una operación de consolidación está limitada a aquellos configurados en el par de máquinas virtuales evaluadas junto con el sucesor de capacidad correspondiente de cada una de ellas definido en la ecuación 6.

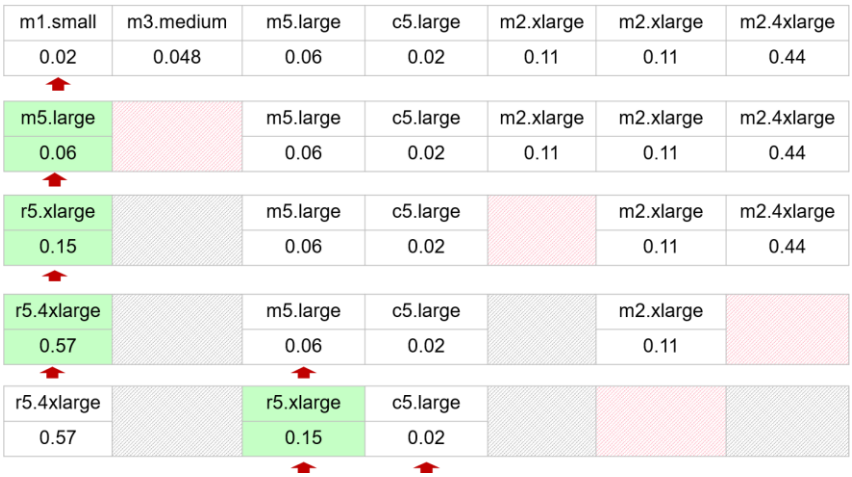

Figura 16. Consolidación de subsecuencias. La subsecuencia $\{1,2,5,7\}$ se consolida en una máquina virtual tipo r5.4xlarge. Posteriormente, la subsecuencia $\{3,6\}$ se consolida en una máquina virtual tipo r5.xlarge. Finalmente, la máquina virtual 4 queda colocada como la última de la secuencia y no se puede consolidar con otra máquina virtual.

Algoritmo 6. Consolidación de subsecuencias

Entrada: Arreglo de máquinas virtuales B.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.
for $i \leftarrow 1$ **to** $longitud(B)$ **do:**
  if $B_i = NIL$ **then:**
    continuar a siguiente iteración;
  for $j \leftarrow i + 1$ **to** $longitud(B)$ **do:**
    if $B_i = NIL$ **then:**
      continuar a siguiente iteración;
    $x \coloneqq B_i;\ y \coloneqq B_j;$
    $vm \coloneqq \underset{z \in \{1,\dots,M\}}{\operatorname{argmin}} \{C_z \mid factible(x, y, z) = 1\};$
    if $C_{vm} < (x_{cost} + y_{cost})$ **then:**
      $f \coloneqq$ consolidar x e y en una máquina virtual tipo vm;
      $B_i \coloneqq f; B_j \coloneqq NIL;$

Algoritmo 7. Consolidación de subsecuencias con búsqueda limitada

Entrada: Arreglo de máquinas virtuales B.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.
for $i \leftarrow 1$ **to** $longitud(B)$ **do:**
  if $B_i = NIL$ **then:**
    continuar a siguiente iteración;
  for $j \leftarrow i + 1$ **to** $longitud(B)$ **do:**
    if $B_i = NIL$ **then:**
      continuar a siguiente iteración;
    $x \coloneqq B_i;\quad y \coloneqq B_j;$
    $candidatos \coloneqq \{x_{vm}, capacidadMayor(x_{vm}), y_{vm}, capacidadMayor(y_{vm})\};$
    $vm \coloneqq \underset{z \in candidatos}{\operatorname{argmin}} \{C_z \mid factible(x, y, z) = 1\};$
    if $C_{vm} < (x_{cost} + y_{cost})$ **then:**
      $f \coloneqq$ consolidar x e y en una máquina virtual tipo vm;
      $B_i \coloneqq f; B_j \coloneqq NIL;$

3.3.4 Migración de carga de máquinas virtuales

Este algoritmo de búsqueda local emplea un vecindario donde cada opción k es el resultado de un modelo de programación entera mixta (modelo 3) que busca distribuir todas las tareas computacionales de una máquina virtual k en otras máquinas virtuales con incremento en factor de tamaño. La primera opción del vecindario que produzca una reducción de costo es elegida para actualizar la solución y continuar con el procedimiento de búsqueda local (algoritmo 8).

El modelo de distribución de carga computacional de una máquina virtual k define variables binarias $D_{j,i}$ que establecen si la tarea j de la máquina virtual k debe ser migrada a la máquina virtual i. El conjunto de máquinas virtuales receptoras de las tareas computacionales de k son modelados con las variables binarias X_i, y la tercera restricción impone la coherencia entre los valores de las variables X y D. Adicionalmente, las máquinas virtuales receptoras son definidas con un tipo de mayor capacidad (ecuación 6) que el establecido en la solución de entrada. La función objetivo del modelo es minimizar el costo de las máquinas virtuales receptoras, y la primera restricción se utiliza para considerar factible solo las soluciones cuyo costo es mejor que la solución original. El modelo debe asegurarse que las tareas computacionales no sean asignadas a la máquina virtual fuente k con la segunda restricción. Finalmente, las últimas tres restricciones establecen que cada máquina virtual seleccionada como receptor debe ser capaz de satisfacer la demanda de recursos de las tareas computacionales que tiene asignadas.

Algoritmo 8. Migración de carga de máquinas virtuales

Entrada: Arreglo de máquinas virtuales B, número máximo de iteraciones MAX.
Resultado: Arreglo de máquinas virtuales B con segmentos consolidados.

$costo = \sum_{b \in B} b_{cost}$;
for $i \leftarrow 1$ **to** MAX **do:**
    $mejorado = falso$;
    for $k \in \{1, \dots, |B|\}$ **do:**
        $B' =$ resultado de Modelo 3 con entrada B, k;
        $costo' = \sum_{b \in B'} b_{cost}$;
        if $costo' < costo$ **then**:
            $mejorado = verdadero$;
            $B = B'$;
            $costo = \sum_{b \in B} b_{cost}$;
            interrumpir iteraciones;
    if $mejorado = false$ **then:**
        interrumpir iteraciones;

Modelo 3. Distribución de carga computacional de una máquina virtual k

sea B un conjunto de máquinas virtuales,
k el índice de la máquina virtual cuya carga computacional se desea distribuir,
$z(i) = capacidadMayor\left(B_{i_{vm}}\right)\ \forall\, i \in \{1, \dots, |B|\}$ (ecuación 6),
$D_{j,i} \in \{0,1\}$ una variable que establece la asignación de la tarea j en la máquina virtual i, $\forall\, j, i \in B_{k_{load}} \times \{1, \dots, |B|\}$ tal que $z(i)$ está definida,
$X_i \in \{0,1\}$ una variable que indica si la máquina virtual i ha sido seleccionado como receptor de alguna tarea computacional de la máquina virtual k, $\forall\, i \in \{1, \dots, |B|\}$ tal que está $z(i)$ está definida.

minimizar $costo = \sum_{i \in \{1,\dots,|B|\}, z(i) \neq NIL} C_{z(i)} X_i,$

sujeto a

$$
\begin{aligned}
costo &\leq \sum_{i \in \{1,\dots,|B|\}} C_i X_i + B_{k_{cost}} - 0.1, \\
D_{j,k} &= 0 && \forall\, j \in \{1, \dots, N\}, \\
\sum_{j=1}^{N} D_{j,i} &\leq N X_i && \forall\, i \in \{1, \dots, |B|\}, z(i) \neq NIL, \\
\sum_{j=1}^{N} p_j D_{j,i} &\leq P_{z(i)} && \forall\, i \in \{1, \dots, |B|\}, z(i) \neq NIL, \\
\sum_{j=1}^{N} m_j D_{j,i} &\leq M_{z(i)} && \forall\, i \in \{1, \dots, |B|\}, z(i) \neq NIL, \\
\sum_{j=1}^{N} b_j D_{j,i} &\leq B_{z(i)} && \forall\, i \in \{1, \dots, |B|\}, z(i) \neq NIL.
\end{aligned}
$$

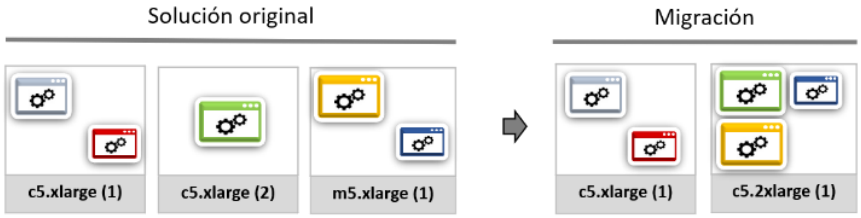

Figura 17. Migración de carga de máquinas virtuales. Las tareas de la tercer máquina virtual son enviadas a la segunda.

3.4 Empaquetado de máquinas virtuales

En el problema de empaquetado de contenedores clásico una colección de artículos ponderados debe ser agrupado en un conjunto finito de contenedores con capacidad limitada donde el objetivo de esta

asignación es minimizar el número total de contenedores utilizados. El problema de empaquetado de contenedores es NP-Difícil, y en la literatura se pueden encontrar heurísticas como Best Fit y First Fit que se han desarrollado para encontrar soluciones de buena calidad en una cantidad razonable de tiempo. En este sentido, en el problema de nuestro estudio podemos observar que el subproblema de asignar tareas computacionales en máquinas virtuales del mismo tipo puede ser traducido a una variante del problema de empaquetado de contenedores con tres dimensiones descrito en el modelo 4.

Modelo 4. Empaquetado de máquinas virtuales

sea
$\hat{J} \subset \{1, \dots, N\}$ el conjunto de tareas computacionales a ser empaquetadas,
$\widehat{N} = |\hat{J}|$,
$\hat{b} \in \{1, \dots, M\}$ el tipo de máquina virtual que puede ser utilizado,
$B = \{1, \dots, \widehat{N}\}$ el conjunto posible de máquinas virtuales,
$Y_i \in \{0,1\}$ una variable de decisión que establece si la máquina virtual i está en uso $\forall i \in B$,
$X_{j,i} \in \{0,1\}$ una variable de decisión que define si la tarea computacional j ha sido asignada a la máquina virtual i, $\forall j, i \in \hat{J} \times B$.

minimizar
$$\sum_{i \in B} Y_i,$$

sujeto a

$$\sum_{i \in B} X_{j,i} = 1 \qquad \forall j \in \hat{J},$$

$$\sum_{j \in \hat{J}} p_j \cdot X_{j,i} \leq P_{\hat{b}} \qquad \forall i \in B,$$

$$\sum_{j \in \hat{J}} m_j \cdot X_{j,i} \leq M_{\hat{b}} \qquad \forall i \in B,$$

$$\sum_{j \in \hat{J}} b_j \cdot X_{j,i} \leq B_{\hat{b}} \qquad \forall i \in B,$$

$$X_{j,i} \leq Y_i \qquad \forall j, i \in J \times B,$$

$$Y_{i+1} \leq Y_i \qquad \forall i \in \{1, \dots, \widehat{N} - 1\}.$$

3.4.1 Best Fit

El algoritmo 9 describe la estrategia Best Fit para el empaquetado de tareas computacionales en máquinas virtuales. El enfoque de esta estrategia consiste en asignar las tareas computacionales en secuencia. Al momento de considerar una tarea computacional t, el procedimiento debe buscar la máquina virtual con menor espacio disponible capaz de satisfacer la demanda de recursos de t y asignar ahí la tarea computacional, en caso de no encontrar una máquina virtual capaz de alojar a t entonces se debe crear una nueva máquina virtual donde se asignará t. Es claro que el orden de procesamiento de las tareas

influye en la calidad de la solución, por lo cual una variante de esta estrategia ordena las tareas computacionales de manera descendente con la función $volumen(j) = p_j * m_j * b_j$.

Algoritmo 9. Best Fit Packing

Entrada: $\hat{J} \subset \{1, \dots, N\}$ es el conjunto de tareas computacionales a empaquetar,
$\hat{b} \in \{1, \dots, M\}$ es el número del tipo de máquina virtual,
$O \in \{0,1\}$ indica si las tareas computacionales deben ser ordenadas.
Resultado: Conjunto de máquinas virtuales $Bins$.
sea $Bins$ una lista vacía;
if $O = 1$ **then**:
  ordenar $\hat{J}$ con la función $volumen(j) = p_j * m_j * b_j$;
for $t \in \hat{J}$ **do:**
  $bestBin \coloneqq \underset{b \in Bins}{\operatorname{argmin}}\{volumenDisponible(b) \mid t\ fits\ in\ b\}$;
  if $bestBin$ existe **then:**
    asigna tarea computacional t a $bestBin$;
    continuar en la siguiente iteración;
  crear una nueva máquina virtual x donde $x_{load} = \{t\}$ y $x_{vm} = \hat{b}$, y agrega x a $Bins$;
return $Bins$;

3.4.2 First Fit

El algoritmo 11 describe la estrategia First Fit. El concepto de este algoritmo es asignar las tareas computacionales en secuencia, una tarea computacional t se asigna a la primera máquina virtual provisionada capaz de satisfacer la demanda de recursos de t, en caso de no existir una máquina virtual capaz de alojar a t se crea una nueva máquina virtual. Una variante de esta estrategia consiste en ordenar las tareas computacionales de acuerdo con la demanda del recurso computacional para el cual la máquina virtual anfitriona ha sido optimizada (Algoritmo 10). Por ejemplo, si el tipo de máquina virtual es optimizado para cómputo entonces la secuencia de tareas se ordena de acuerdo con la demanda de CPU, de esta manera se pretende aprovechar la relación costo-beneficio del consumo de poder cómputo de una máquina virtual optimizada para este propósito.

Algoritmo 10. Empaquetado First Fit Personalizado

Entrada: Sea $\hat{J} \subset \{1, \dots, N\}$ el conjunto de tareas computacionales a empaquetar,
$\hat{b} \in \{1, \dots, M\}$ un tipo de máquina virtual.
Resultado: Conjunto de máquinas virtuales.
if $\hat{b}$ es optimizado para cómputo **then:**
  $S \coloneqq \hat{J}$ ordenado por requerimientos de cómputo;
  return $FirstFitPacking(S, \hat{b})$;
if $\hat{b}$ es optimizado en memoria **then:**
  $S \coloneqq \hat{J}$ ordenado por requerimiento de memoria;
  return $FirstFitPacking(S, \hat{b})$;
if $\hat{b}$ es optimizado para ancho de banda **then:**
  $S \coloneqq \hat{J}$ ordenado por requerimiento de ancho de banda;
  return $FirstFitPacking(S, \hat{b})$;

$S \coloneqq \hat{J}$ ordenado por volumen donde $volumen(\hat{J}_j) = p_j m_j b_j$;
return $FirstFitPacking(S, \hat{b})$;

Algoritmo 11. Empaquetado First Fit

Entrada: $\hat{J} \subset \{1, \dots, N\}$ es el conjunto de tareas computacionales para empaquetar,
$\hat{b} \in \{1, \dots, M\}$ es el número del tipo de máquina virtual a utilizar.
Resultado: Conjunto de máquinas virtuales $Bins$.
sea $Bins$ una lista vacía ;
for $t \in \hat{J}$ **do:**
  $firstBin \coloneqq \min\{b \in Bins \mid t \; cabe \; en \; b\}$;
  if $firstBin$ existe **then:**
    asigna tarea computacional t a $firstBin$;
    continuar en siguiente iteración;

  crear una nueva máquina virtual x donde $x_{load} = t$ y $x_{vm} = \hat{b}$;
  agrega x a $Bins$;
return $Bins$;

3.4.3 Empaquetado de máquinas virtuales con algoritmo genético

Este enfoque utiliza un algoritmo genético que codifica la solución como una permutación del conjunto de tareas computacionales a empaquetar y emplea la estrategia First Fit (Algoritmo 11) como procedimiento de decodificación. De esta manera, el algoritmo genético explora las diferentes variantes que puede producir la estrategia First Fit.

La tabla 2 presenta la configuración operadores genéticos utilizada por este algoritmo. El operador de recombinación PMX y el operador de mutación SWAP se eligieron debido al buen desempeño que exhiben

para problemas cuya solución es representada como una permutación (Corcoran y Wainwright, 1992). El procedimiento de selección con torneo binario fue elegido debido su popularidad, eficiencia computacional, y desempeño para proveer selección de presión (Yadav y Sohal, 2017). Posteriormente, se llevó a cabo un estudio de calibración para elegir los siguientes parámetros: probabilidad de mutación (0.1, 0.2, 0.3, 0.4), probabilidad de cruce (0.5, 0.6, 0.7, 0.8, 0.9) y tamaño de población (100, 200, 400, 500). Este estudio se llevó a cabo evaluando los resultados de cada combinación de parámetros al resolver 41 problemas en 30 ocasiones. De este modo, se eligieron aquellos parámetros que generan mejor costo normalizado promedio.

Tabla 2. Operadores genéticos para empaquetado de máquinas virtuales.

Codificación	Permutación
Decodificación	Heurística First Fit
Población	400
Reemplazo	Generacional
Selección	Torneo Binario
Cruce	PMX
Probabilidad de cruce	0.8
Mutación	SWAP
Probabilidad de mutación	0.1
Número de evaluaciones	100000

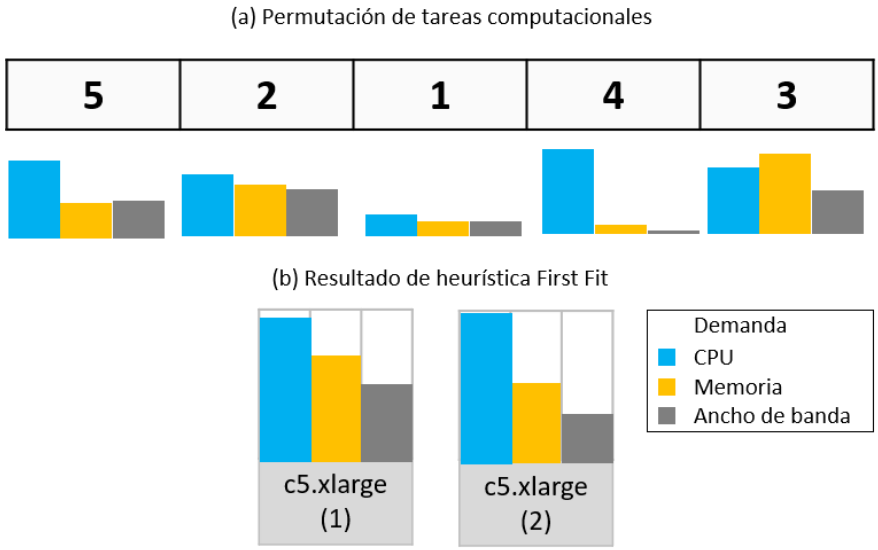

Figura 18. Codificación de solución para algoritmo genético de empaquetado de máquinas virtuales. Subfigura a: Ilustración de secuencia de cinco tareas computacionales con diferentes requerimientos de CPU, memoria, y ancho de banda. Subfigura b: La estrategia First Fit empaqueta las cinco tareas computacionales en dos máquinas virtuales tipo c5.xlarge.

3.5 Empaquetado de máquinas virtuales y cobertura de conjuntos

Dado un conjunto de elementos E, y n subconjuntos $e_1, \ldots, e_n$, donde $e_k \subset E$ está asociado a un costo $z_k \in \mathbb{R}_{>0}$. El problema de cobertura de conjuntos consiste en seleccionar la colección de $m \leq n$ subconjuntos cuya unión contenga todos los elementos de E y cuyo costo total sea el menor posible. Una forma intuitiva de generar subconjuntos razonables del conjunto de tareas computacionales es utilizando algún procedimiento de empaquetado cada tipo de máquina virtual. Posteriormente, el conjunto resultante de máquinas virtuales puede ser definido como la entrada del modelo de programación entera de cobertura de conjuntos (Modelo 5) con el objetivo de encontrar una asignación eficiente de las tareas computacionales en máquinas virtuales (Algoritmo 12).

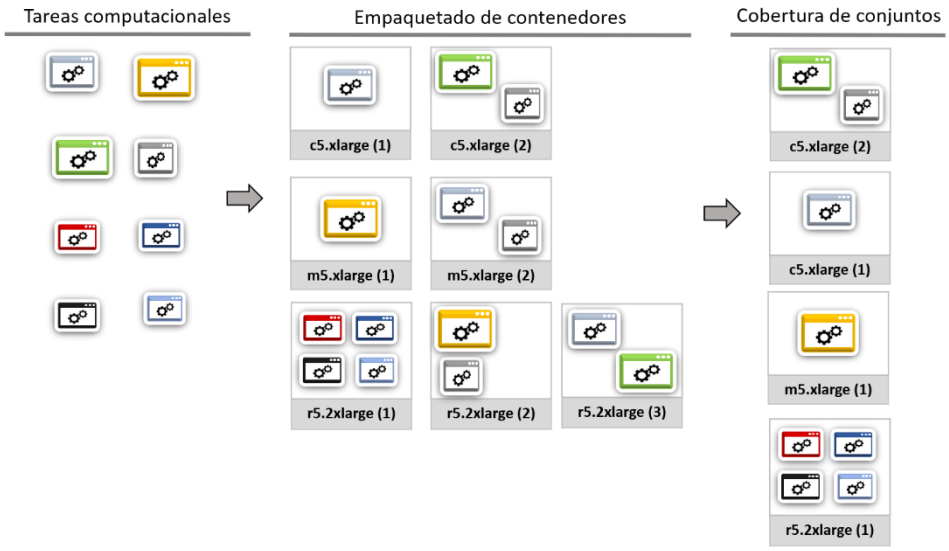

Figura 19. Empaquetado de máquinas virtuales y cobertura de conjuntos. Un algoritmo de empaquetado de máquinas virtuales es ejecutado para cada tipo de máquina virtual y las tareas computacionales que son compatibles. De la colección de máquinas virtuales se eligen aquellas que en conjunto contienen a todas las tareas computacionales y cuyo costo total es el menor posible.

Modelo 5. Cobertura de conjuntos

sea B un conjunto de máquinas virtuales,
$X_b \in \{0,1\}$ una variable de decisión que determina si la máquina virtual es seleccionada, $\forall b \in B$.

minimizar $$\sum_{b \in B} X_b \cdot b_{cost},$$

sujeto a $$\sum_{b \in B, j \in b_{load}} X_b \geq 1.$$

Algoritmo 12. Empaquetado de máquinas virtuales y cobertura de conjuntos

Entrada: Sea $J = \{J_1, \ldots, J_N\}$ un conjunto de N tareas computacionales, y sea $T = \{T_1, \ldots, T_M\}$ un conjunto de M tipos de máquinas virtuales.
Resultado: Costo de la solución
sea B una lista vacía;
for $vm \in T$ **do:**
  $J_{vm} \coloneqq \{j \in \{1, \ldots, N\} \mid J_j \; cabe \; en \; vm\}$;
  $bins \coloneqq empaquetadoDeMaquinasVirtuales(J_{vm}, vm)$;
  agregar $bins$ a lista B;
$cobertura \coloneqq coberturaDeMaquinasVirtuales(B)$;
$costo \coloneqq \sum_{b \in cobertura} b_{cost}$;

3.6 Partición y empaquetado de máquinas virtuales

La solución del modelo 2 producida por el algoritmo 1 de la sección 3.2 define a asociación razonable de las tareas computacionales a los tipos de máquinas virtuales utilizando una función de costo relajada. Por lo tanto, una consecuencia natural de este resultado puede ser utilizar un procedimiento de empaquetado de máquinas virtuales para definir la asignación de las tareas computacionales asociadas al mismo tipo de máquina virtual. El algoritmo 10 describe los pasos de esta estrategia y la figura 20 presenta una ilustración ejemplificada de las etapas este algoritmo.

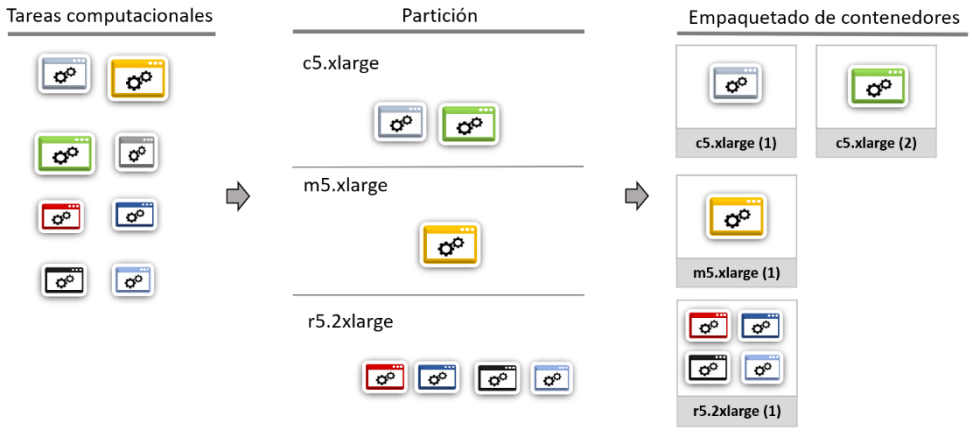

Figura 20. Partición y empaquetado de máquinas virtuales. El modelo de programación entera del Algoritmo 1 genera una partición del conjunto de tareas computacionales eligiendo el tipo de máquina virtual en que se alojará cada una. Posteriormente, cada subconjunto de tareas computacionales es procesado como un problema de empaquetado de máquinas virtuales.

Algoritmo 13. Partición y empaquetado de máquinas virtuales

Entrada: J es el conjunto de tareas computacionales a empaquetar,
T es el conjunto de tipos de máquinas virtuales a considerar
Resultado: Costo de la solución
sea B una lista vacía; sea LB la solución del **Algoritmo 1** para elección del tipo de máquina virtual de cada tarea;
for $vm \in T$ **do:**
  $jobs \coloneqq$ tareas computacionales en LB asignadas al tipo de máquina virtual vm;
  $bins \coloneqq empaquetadoDeMaquinasVirtuales(jobs, vm)$;
  agrega $bins$ a B;
$cost \coloneqq \sum_{b \in B} b_{cost}$;

3.7 Algoritmo genético de consolidación basado en permutación (DTP PLS GA)

Este algoritmo genético explora distintas formas de agrupar tareas computacionales en máquinas virtuales eficientes en costo usando heurísticas de consolidación basadas en una secuencia de máquinas virtuales tales como los algoritmos 4 y 6. En este enfoque una solución es codificada como una permutación de las tareas computacionales y la decodificación se realiza asignando cada tarea computacional a una máquina virtual factible de bajo costo para posteriormente aplicar una estrategia de consolidación. La tabla 4 enlista las variantes del procedimiento de decodificación presentado en el algoritmo 14 e ilustrado en la figura 21.

La configuración de operadores genéticos de este algoritmo (tabla 3) está basada en los resultados obtenidos del estudio de calibración del algoritmo genético empaquetado de máquinas virtuales. Esta decisión fue tomada debido a que ambos algoritmos codifican una solución como una permutación de tareas computacionales, además de considerar prioritaria la evaluación de los distintos procedimientos de decodificación en el período de experimentación. Finalmente, este algoritmo utiliza una población de 200 debido al elevado tiempo de ejecución observado durante las primeras ejecuciones con población de tamaño 400.

Tabla 3. Operadores genéticos para DTP PLS GA

Codificación	Permutación
Decodificación	Algoritmo de consolidación
Población	200
Reemplazo	Generacional
Selección	Torneo Binario
Cruce	PMX
Probabilidad de cruce	0.8
Mutación	SWAP
Probabilidad de mutación	0.15
Número máximo de evaluaciones	100000

Algoritmo 14. Decodificación DTP PLS GA

Entrada: Arreglo S de N enteros que representan la permutación de tareas computacionales
Resultado: Costo de la solución
sea B una lista vacía;
for $i \leftarrow 1$ **to** N **do:**
  $j \coloneqq S_i$;
  $t \coloneqq \underset{z \in \{1,\ldots,M\}}{\operatorname{argmin}} \{C_z \mid J_j \; cabe \; en \; T_z\}$;
  crear una nueva máquina virtual x donde $x_{load} = \{j\}$ y $x_{vm} = t$;
  agregar x a B;
$\hat{B} \coloneqq$ resultado del procedimiento de consolidación con las máquinas virtuales de B;
$cost \coloneqq \sum_{b \in \hat{B}} b_{cost}$;

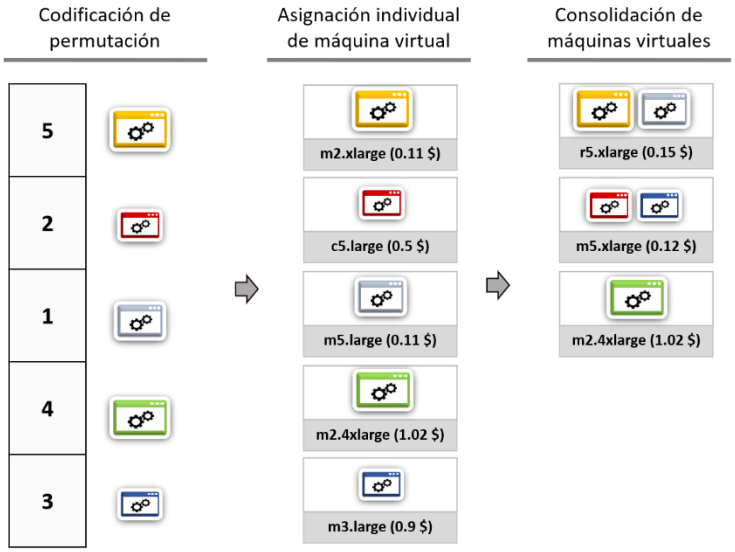

Figura 21. Procedimiento de decodificación del algoritmo genético de consolidación basado en permutación. El i-ésimo elemento de la codificación representa una tarea computacional. Posteriormente, cada tarea computacional es asignada una instancia del tipo de máquina virtual de menor costo capaz de alojarla. En la última etapa, el procedimiento de consolidación une la carga de trabajo de algunas máquinas virtuales para obtener una solución de menor costo.

Tabla 4. Variantes del procedimiento de decodificación para DTP PLS GA

Nombre	Algoritmo
LIMITED_SEARCH_LTR	Consolidación de subsecuencias con búsqueda limitada
INDEXED_FULL_SEARCH_LTR	Consolidación de subsecuencias
LIMITED_SEARCH_NEXT	Consolidación de segmentos con búsqueda limitada
INDEXED_FULL_SEARCH_NEXT	Consolidación de segmentos
LIMITED_SEARCH_NEXT_AND_BEST	Consolidación de segmentos con búsqueda limitada – consolidación del mejor par
INDEXED_FULL_SEARCH_NEXT_AND_BEST	Consolidación de segmentos – consolidación del mejor par

3.8 Algoritmo genético de selección de conjunto de máquinas virtuales (DTP SETUP GA)

El conjunto de máquinas virtuales utilizadas para alojar las tareas computacionales de un gemelo digital puede ser descrito como la cantidad de máquinas virtuales provisionadas de cada tipo. Este algoritmo explora el espacio de conjuntos de máquinas virtuales utilizando una codificación de M enteros donde el i-ésimo elemento representa la cantidad de máquinas rentadas del tipo T_i (Figura 22). Claramente, esta codificación no representa una solución completa del problema. Por lo tanto, es necesario que el procedimiento de decodificación utilice una estrategia de empaquetado de máquinas virtuales consciente de costos para asignar las tareas computacionales al conjunto de máquinas virtuales de la solución (Algoritmo 15).

El algoritmo 15 inicia haciendo una copia de la codificación llamada $Disponibles$ donde mantendrá el conteo de máquinas virtuales disponibles para utilizar durante el proceso de asignación de tareas computacionales. De manera similar, el arreglo $Usados$ es creado para registrar la cantidad de máquinas virtuales de cada tipo que son parte de la solución completa. Adicionalmente, la secuencia de asignación de tareas computacionales S está ordenada por la función $volumen(J_j) = p_j m_j b_j$ y la lista inicial de máquinas virtuales se encuentra vacía. Al momento de procesar una tarea computacional J_j, la primera opción del algoritmo es asignarla a la máquina virtual de menor precio en la lista capaz de alojarla. En caso de no existir tal máquina virtual, la segunda opción del algoritmo es alojar la tarea computacional en una nueva máquina virtual de tipo k tal que $Disponibles[k] > 0$, tenga la capacidad de satisfacer la tarea, y sea la opción más barata. Si tal tipo de máquina virtual existe entonces se disminuye la cantidad de disponibles y se incrementa la cantidad de usadas del tipo de máquina virtual k. La última opción del algoritmo es elegir el tipo de máquina virtual de menor costo capaz de alojar a J_j sin considerar si es parte del conjunto de máquinas virtuales definidos en la codificación. Al concluir la asignación de tareas computacionales, la codificación del individuo es corregida utilizando el arreglo $Usados$, y el costo de la solución es fácil de calcular. Finalmente, cabe mencionar que es posible reducir el tiempo de ejecución de la evaluación de la población del algoritmo genético calculando con antelación la secuencia de asignación de tareas computacionales S y creando una tabla de referencia que define la habilidad del tipo de máquina virtual T_i para alojar la tarea computacional J_j.

La tabla 5 muestra los operadores genéticos utilizados por el algoritmo para explorar el espacio de soluciones. La mayor parte de esta combinación de parámetros fue tomada de la configuración del

algoritmo genético de selección de tipo de máquina virtual para alojar tarea computacional (Tabla 7 de la sección 3.10). Esta decisión fue tomada con la intención de tomar ventaja del tiempo invertido en el estudio de calibración del segundo algoritmo debido a que ambos algoritmos codifican una solución como un arreglo de enteros. Adicionalmente, el tamaño de población fue reducido a 100 individuos con la motivación de aumentar la velocidad del algoritmo al procesar problemas de tamaño 5000.

Podemos observar que los resultados experimentales de la configuración descrita en la tabla 5 son favorables (Figura 51 de la sección 4.2.6). Sin embargo, con la intención de validar la efectividad de dicha combinación de parámetros, se realizó un estudio con 12 problemas representativos donde se probaron todas las combinaciones de los siguientes parámetros: operador de recombinación (SBX, cruce de un punto, cruce de dos puntos), probabilidad de cruce (0.6, 0.7, 0.8), mutación (polinomial, reasignación aleatoria), probabilidad de mutación (0.1, 0.2, 0.3), y tamaño de población (100, 200, 400). Los resultados indican que la combinación de parámetros con menor costo normalizado promedio es similar al propuesto, excepto la probabilidad de mutación y el tamaño de población, donde los valores recomendables son 0.6 y 400 respectivamente.

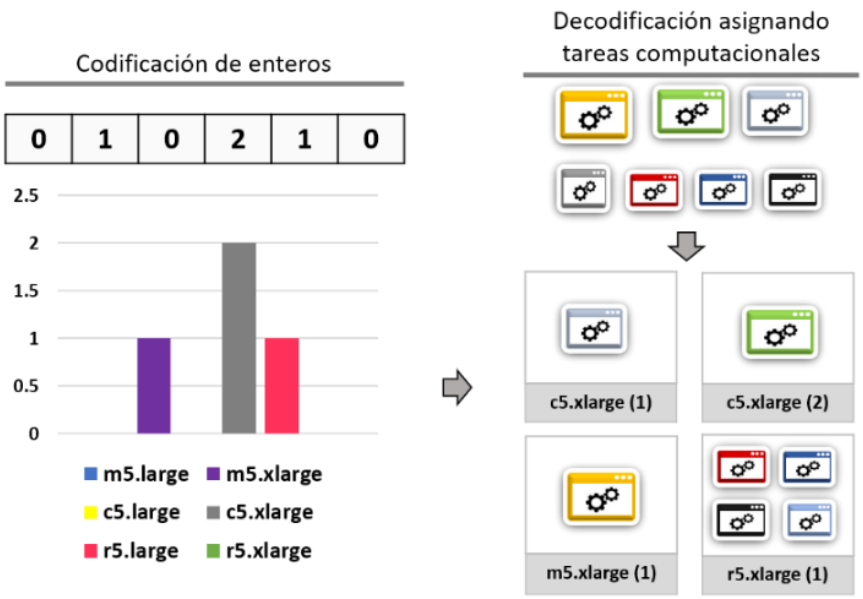

Figura 22. Algoritmo genético de selección del conjunto máquinas virtuales. El i-ésimo elemento de la codificación representa la cantidad de máquinas virtuales rentadas del tipo T_i. Posteriormente, el procedimiento de decodificación asigna las tareas computacionales en las máquinas virtuales provisionadas.

Tabla 5. Operadores genéticos para DTP SETUP GA

Codificación	Entera
Decodificación	Algoritmo 12
Población	100
Reemplazo	Generacional
Selección	Torneo Binario
Cruce	SBX
Probabilidad de cruce	0.8
Mutación	Polinomio de enteros
Probabilidad de mutación	$1/N$
Número máximo de evaluaciones	200000

Algoritmo 15. Decodificación DTP SETUP GA

Entrada: Arreglo Q de M enteros donde Q_i representa la cantidad de máquinas virtuales elegidas del tipo T_i .
Resultado: Costo de la solución
sea $Disponible$ una copia de Q;
sea $Usado$ un arreglo de enteros de tamaño M inicializado con ceros;
sea L una lista vacía ;
sea $S \coloneqq$ conjunto J ordenado por $volumen(J_j) = p_j m_j b_j$;

for $t \in S$ **do:**
  $mejorVM \coloneqq \underset{b \in L}{\operatorname{argmin}}\{b_{cost} \mid t\ cabe\ en\ b\}$;
  if $mejorVM$ existe **then:**
    añade t a $mejorVM$;
    continuar en siguiente iteración;

  $mejorTipo \coloneqq \underset{i \in \{1,\dots,M\}}{\operatorname{argmin}}\{C_i \mid t\ cabe\ en\ T_i, Disponible[i] > 0\}$;
  if $mejorTipo$ existe **then:**
    crear una nueva máquina virtual x donde $x_{load} = \{t\}$ y $x_{vm} = mejorTipo$;
    agrega x a L;
    disminuye 1 a $Disponible[mejorTipo]$;
    incrementa 1 a $Usado[mejorTipo]$;
    continuar en la siguiente iteración;

  $nuevoTipo \coloneqq \underset{i \in \{1,\dots,M\}}{\operatorname{argmin}}\{C_i \mid t\ cabe\ en\ T_i\}$;
  crear una nueva máquina virtual x donde $x_{load} = \{t\}$ y $x_{vm} = nuevoTipo$;
  agrega x a L;
  incrementa 1 a $Usado[nuevoTipo]$;
$costo \coloneqq \sum_{b \in L} b_{cost}$;
/* corrección de codificación de individuo*/
$Q \coloneqq Usado$;

3.9 Algoritmo genético de selección de conjunto de máquinas virtuales basado en grupos (DTP SETUP BY GROUP GA)

Cada tipo de máquina virtual de la colección de EC2 es denotada con un nombre de dos partes constituido por el nombre del grupo y la magnitud del tipo de máquina virtual. Por ejemplo, un tipo de máquina virtual c5.xlarge es el tipo de máquina virtual con la segunda mejor magnitud del grupo c5, y sus capacidad de recursos computacionales es aproximadamente equivalente a la suma de la capacidad de dos máquinas virtuales tipo c5.large. Dentro de un grupo de tipos de máquinas virtuales, EC2 denota la magnitud de cada tipo de máquina virtual con un número entero denominado factor de normalización, el cual es útil para comparar máquinas virtuales del mismo grupo y encontrar equivalencias como la mencionada anteriormente.

Sea GT la cantidad de grupos de tipos de máquinas virtuales existentes en T. El conjunto de máquinas virtuales definida para alojar la carga computacional puede ser modelado como el requerimiento de capacidad de recursos computacionales expresada en unidades del factor de normalización de cada grupo de tipos de máquinas virtuales. Este algoritmo explora el espacio de conjuntos de máquinas virtuales utilizando una codificación de GT enteros donde el i-ésimo elemento representa la cantidad de capacidad computacional requerida del grupo i (Figura 23).

El algoritmo 16 describe el procedimiento de decodificación que utiliza una estrategia de asignación de carga computacional consciente de costos. Este algoritmo inicia haciendo una copia de la codificación llamada $Disponible$ donde mantendrá el conteo de capacidad disponible de cada grupo para utilizar durante el proceso de asignación de tareas computacionales. De manera similar, el arreglo Uso es creado para registrar las unidades del factor de normalización de cada grupo que constituyen la solución final. Adicionalmente, la secuencia de asignación de tareas computacionales S está ordenada por la función $volumen(J_j) = p_j m_j b_j$ y se crea una lista vacía de máquinas virtuales. Al momento de procesar una tarea computacional J_j, la primera opción del algoritmo es asignarla a la máquina virtual de menor precio en la lista capaz de alojarla. En caso de no existir tal máquina virtual, el procedimiento define para cada grupo i, cuál es el tipo máquina virtual T_i perteneciente al grupo i con capacidad de alojar a J_j, y cuyo factor de normalización es el más cercano a $Disponible[i]$ en ese instante. Posteriormente, el procedimiento elige del conjunto de tipos de máquinas virtuales resultantes del paso anterior aquel cuyo precio es el menor. La última opción del algoritmo es elegir el tipo de máquina virtual de menor costo capaz de alojar a J_j sin

considerar si es parte del conjunto de máquinas virtuales modelados en la codificación. Al concluir la asignación de tareas computacionales, la codificación del individuo es corregida utilizando el arreglo Uso, y el costo de la solución es calculado sumando los precios de los elementos en $Contenedores$. Finalmente, cabe mencionar que es posible reducir el tiempo de ejecución de la evaluación de la población del algoritmo genético calculando con antelación la secuencia de asignación de tareas computacionales S y creando una tabla de referencia que define la habilidad del tipo de máquina virtual T_i para alojar la tarea computacional J_j.

La tabla 6 presenta la configuración operadores genéticos utilizada por este algoritmo, la cual fue tomada de la tabla 5 debido a que este algoritmo es una variante muy cercana algoritmo genético de selección de conjunto de máquinas virtuales (sección 3.8) la configuración de operadores genéticos. Los resultados experimentales observados con esta configuración son favorables (Figura 51 de la sección 4.2.6). Posteriormente, con la intención de validar la efectividad de dicha configuración, se realizó un estudio con 12 problemas representativos donde se probaron todas las combinaciones de los siguientes parámetros: operador de recombinación (SBX, cruce de un punto, cruce de dos puntos), probabilidad de cruce (0.6, 0.7, 0.8), mutación (polinomial, reasignación aleatoria), probabilidad de mutación (0.1, 0.2, 0.3), y tamaño de población (100, 200, 400). Los resultados indican que la combinación de parámetros con menor costo normalizado promedio es igual al propuesto en la tabla 6 , con la excepción de la probabilidad de mutación donde el valor es 0.6.

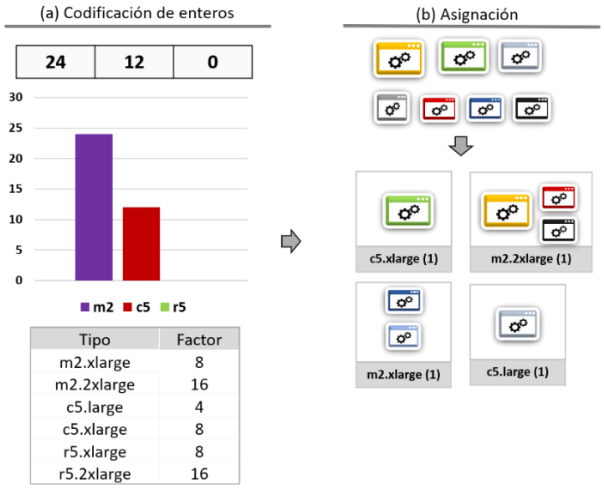

Figura 23. Algoritmo genético de selección del conjunto máquinas virtuales basado en grupos. Subfigura a: el i-ésimo elemento de la codificación representa la capacidad computacional provisionada del grupo i, expresada en unidades del factor de normalización. En esta ilustración el grupo m2 solicita 24 unidades que pueden ser satisfechas con 3 máquinas virtuales m2.xlarge ó con 1 máquina virtuales m2.2xlarge y una m2.xlarge. Subfigura b: el procedimiento de decodificación crea máquinas virtuales con base en el requerimiento de capacidad modelado por la codificación y asigna las tareas computacionales en ellos. En este ejemplo, el conjunto de máquinas virtuales m2.2xlarge, m2.xlarge,c5.xlarge, y c5.large alojan las tareas computacionales.

Tabla 6. Operadores genéticos para DTP SETUP BY GROUP GA

Codificación	Entera
Decodificación	Algoritmo 13
Población	100
Reemplazo	Generacional
Selección	Torneo Binario
Cruce	SBX
Probabilidad de cruce	0.8
Mutación	Polinomio de enteros
Probabilidad de mutación	$1/N$
Número máximo de evaluaciones	200000

Algoritmo 16. Decodificación DTP SETUP BY GROUP GA

Entrada: Arreglo Q de GT enteros que representan el poder de cómputo provisionado de cada grupo de tipos de máquinas virtuales.
Resultado: Costo de la solución
sea $Disponible$ una copia de Q;
sea Uso un arreglo de GT enteros inicializado con ceros;
sea L una lista vacía ;
sea $S \coloneqq J$ ordenado por $volumen(J_j) = p_j m_j b_j$;
for $t \in S$ **do:**
  $mejorVM \coloneqq \underset{b \in L}{\operatorname{argmin}}\{b_{cost} \mid t\ cabe\ en\ b\}$;
  if $mejorVM$ existe **then:**
    agrega t a $mejorVM$;
    continuar en siguiente iteración;
  for $g \in \{1, \dots, GT\}$ **do:**
    $candidato_g \coloneqq \underset{i \in \{1,\dots,M\}}{\operatorname{argmin}}\{\ |F_i - Disponible[g]|\ \mid\ t\ cabe\ en\ T_i,\ G_i = g\ \}$
  $candidatos \coloneqq \{candidato_g \mid g \in \{1, \dots, GT\},\ Disponible[g] > 0\ \}$;
  $mejorTipo \coloneqq \underset{i \in candidatos}{\operatorname{argmin}}\{C_i\}$;
  if $mejorTipo$ existe **then:**
    crear nueva máquina virtual x donde $x_{load} = \{t\}$ y $x_{vm} = mejorTipo$;
    agregar x a L ;
    $Disponible[mejorTipo]$:= $Disponible[mejorTipo]$- $F_{mejorTipo}$;
    $Uso[mejorTipo]$:= $Uso[mejorTipo] + F_{mejorTipo}$;
    continuar en siguiente iteración;

  $nuevoTipo \coloneqq \underset{i \in \{1,\dots,M\}}{\operatorname{argmin}}\{C_i \mid t\ cabe\ en\ T_i\}$;
  crear nueva máquina virtual x donde $x_{load} = \{t\}$ y $x_{vm} = nuevoTipo$;
  agrega x a L;
  $Uso[nuevoTipo] := Uso[nuevoTipo] + F_{nuevoTipo}$;
$costo \coloneqq \sum_{b \in L} b_{cost}$;
/* corrección de codificación */
$Q \coloneqq Uso$;

3.9.1 Postprocesamiento de consolidación de subsecuencias

El costo de la solución codificada por un individuo de la población puede ser obtenido por medio de la extensión del procedimiento de decodificación añadiendo la estrategia de consolidación de subsecuencias (Algoritmo 6) como paso final. El algoritmo genético que implementa este enfoque es denominado DTP SETUP BY GROUP GA (LTR) en este estudio, y la figura 24 presenta una ilustración de las etapas de decodificación.

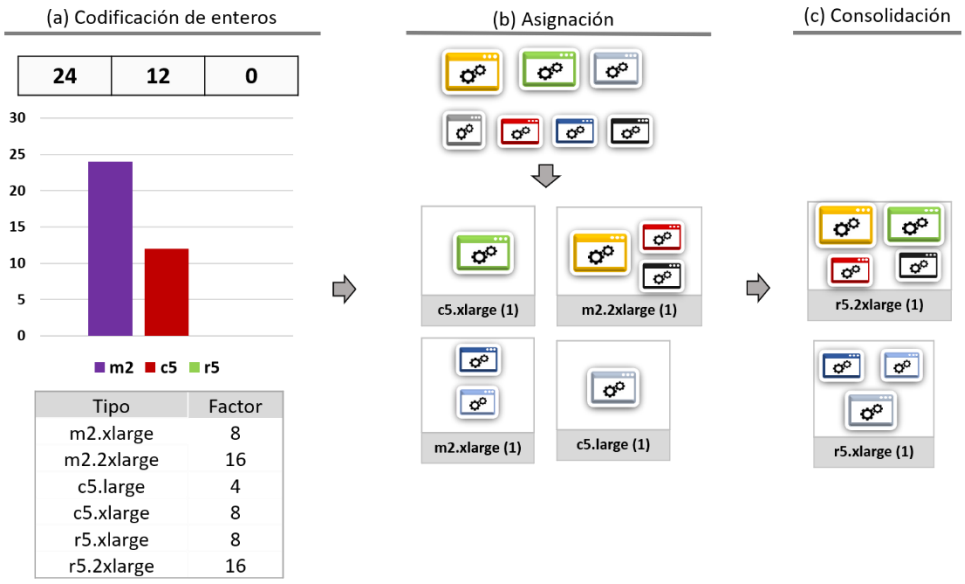

Figura 24. Algoritmo genético de selección del conjunto máquinas virtuales basado en grupos con postprocesamiento. El i-ésimo elemento de la codificación representa la capacidad computacional provisionada del grupo i, expresada en unidades del factor de normalización (subfigura a). Posteriormente, el procedimiento de decodificación crea máquinas virtuales con base en el requerimiento de capacidad modelado por la codificación y asigna las tareas computacionales en ellas (subfigura b). La última etapa del proceso de decodificación consiste utilizar la estrategia de consolidación de subsecuencias (Algoritmo 4) para mejorar el resultado del paso anterior (subfigura c).

3.10 Algoritmo genético de selección de tipo de máquina virtual para alojar tarea computacional (WTS GA)

Sea $FT_j = \{i \in \{1,\ldots,M\} \mid p_j <= P_i,\ m_j <= M_i, b_j <= B_i\}$ el conjunto tipos de máquinas virtuales factibles de alojar una tarea computacional J_j. Este algoritmo codifica un individuo como un arreglo Q de M enteros donde $Q_j \in \{1,\ldots,|FT_j|\}$ representa el tipo de máquina virtual factible seleccionado para alojar la tarea computacional J_j. Esta representación es decodificada por el algoritmo 14 utilizando una

heurística de empaquetado de máquinas virtuales para cada grupo de tareas computacionales asignadas al mismo tipo de máquina virtual (Figura 25).

Sea $K = 10,$ y sea KNN_j el conjunto de K tareas más cercanas a la tarea J_j utilizando la distancia definida en la ecuación 7. La mutación de vecino aleatorio (Algoritmo 18) modifica Q_j para asignar el valor que representa el tipo de máquina virtual mas cercano al seleccionado para alguna tarea aleatoria de KNN_j. La intención de este operador de mutación es obtener beneficio de la suposición que tareas computacionales similares pueden ser alojadas en máquinas virtuales del mismo tipo.

$$distancia(a, b) = \sqrt{(p_a - p_b)^2 + (m_a - m_b)^2 + (b_a - b_w)^2}. \quad (7)$$

La tabla 7 presenta la configuración operadores genéticos utilizada por este algoritmo. El procedimiento de selección con torneo binario fue elegido debido su popularidad, eficiencia computacional, y desempeño para proveer selección de presión (Yadav y Sohal, 2017). Posteriormente, se llevó a cabo un estudio de calibración para elegir los siguientes parámetros: operador de recombinación (SBX, cruce de un punto, cruce de dos puntos), probabilidad de cruce (0.5, 0.7, 0.8, 0.9), mutación (polinomial, reasignación aleatoria, vecino aleatorio), probabilidad de mutación (0.1, 0.2, 0.3, 0.4), y tamaño de población (100, 200, 300, 400). Este estudio se llevó a cabo evaluando los resultados de cada combinación de parámetros al resolver 41 problemas en 30 ocasiones. De este modo, se eligieron aquellos parámetros que generan mejor costo normalizado promedio.

Algoritmo 17. Decodificación WTS GA

Entrada: Arreglo Q de N enteros donde Q_j representa el número del tipo de máquina virtual $i \in \{1, \dots, M\}$ elegido para alojar la tarea J_j.
Resultado: Costo de la solución.

$VM_j \coloneqq FT_{j,Q_j} \quad \forall\, j \in \{1, \dots, N\}$;
sea B una lista vacía ;
for $i \in \{1, \dots, M\}$ **do:**
  $L_i \coloneqq \{j \mid j \in \{1, \dots, N\}, VM_j = i\}$;
  if $|L_i| = 0$ **then:**
    continuar en siguiente iteración;
  $\hat{B}_i =$ empaquetado de máquinas virtuales Best Fit (L_i, T_i); /* Algoritmo 9*/
  agregar elementos de $\hat{B}_i$ a B;
$costo \coloneqq \sum_{b \in B} b_{cost}$;

Algoritmo 18. Mutación de Vecino Aleatorio

Entrada: Arreglo Q de N enteros donde Q_j representa el número del tipo de máquina virtual $k \in \{1, \ldots, M\}$ elegido para alojar la tarea J_j. Entero j que indica el valor de Q que será mutado.

$n :=$ selecciona un elemento al azar del conjunto KNN_j;
$VM_n \coloneqq FT_{n,Q_n}$;
$nuevoQ_j := \min_{k \in \{1,\ldots,|FT_j|\}} distancia(VM_n, FT_{j,k})$;
$Q_j := nuevoQ_j$;

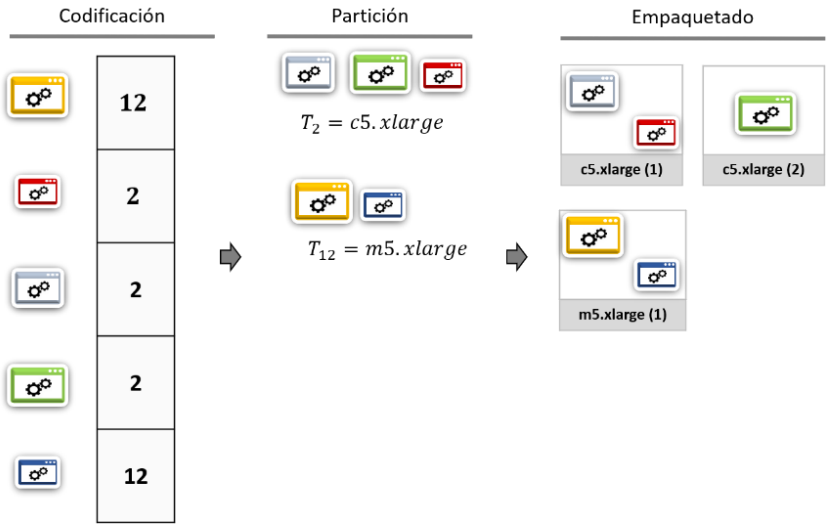

Figura 25. Algoritmo genético de selección de tipo de máquina virtual para tarea computacional. El i-ésimo elemento de la codificación representa el tipo de máquina virtual seleccionado para la tarea computacional J_i (subfigura a). Esta asignación induce una partición del conjunto de tareas computacionales (subfigura b). El procedimiento de decodificación (Algoritmo 17) utiliza una estrategia de empaquetado de máquinas virtuales por cada subconjunto de la partición (subfigura c).

Tabla 7. Operadores genéticos para WTS GA

Codificación	Entera
Decodificación	Algoritmo 17
Población	300
Reemplazo	Generacional
Selección	Torneo Binario
Cruce	SBX
Probabilidad de cruce	0.8
Mutación	Vecino Aleatorio
Probabilidad de mutación	0.2
Número máximo de evaluaciones	200000

3.11 Filtro, partición, y empaquetado de máquinas virtuales

Esta estrategia está compuesta por tres etapas. La primera etapa filtra la colección de tipos de máquinas virtuales, el resultado procedimiento de asignación simple de la sección 3.6 utilizado para conocer cuales grupos de tipos de máquinas virtuales son permitidos en la siguiente etapa. La segunda etapa hace uso del algoritmo genético de la sección 3.11 para encontrar una partición adecuada del conjunto de tareas computacionales y el tipo de máquina en el que se pueden asignar las tareas de cada subconjunto. Finalmente, cada subconjunto de la partición obtenida en el paso anterior es procesada como un problema de empaquetado de máquinas virtuales con el algoritmo genético de la sección 3.4.3.

Algoritmo 19. Filtro, Partición, y Empaquetado

Entrada: J es el conjunto de tareas computacionales a empaquetar,
T es el conjunto de tipos de máquinas virtuales a considerar.
Resultado: Costo de la solución.

/* Filtro */
sea $A = \{\}$;
for $i \in \{1, \dots, M\}$ **then**:
    $k = \min_{i \in \{1,\dots,M\}} \{C_i \mid p_j \leq P_i, m_j \leq M_i, b_j \leq B_i\}$;
    if G_k no existe en A **then:**
        agregar G_k en A;
$\hat{T} = \{T_i \mid i \in \{1, \dots, M\}, G_i \in A\}$;

/* Partición */
sea B la lista de máquinas virtuales resultante del algoritmo WTS GA (sección 3.10) con entrada $(J, \hat{T})$;
for $i \in \{1, \dots, M\}$**:**
    $L_i = \bigcup_{b \in B} \{b_{load} \mid b_{vm} = i\}$;

/* Empaquetado */
for $i \in \{1, \dots, M\}$**:**
    if $|L_i| > 0$ **then:**
        $\hat{B}_i =$ solución de empaquetado de máquinas virtuales con algoritmo genético (sección 3.4.3) con entrada (L_i, T_i);

/*Resultado*/
$$costo = \sum_{i \in \{1,\dots,M\}} \sum_{b \in \hat{B}_i} b_{cost};$$

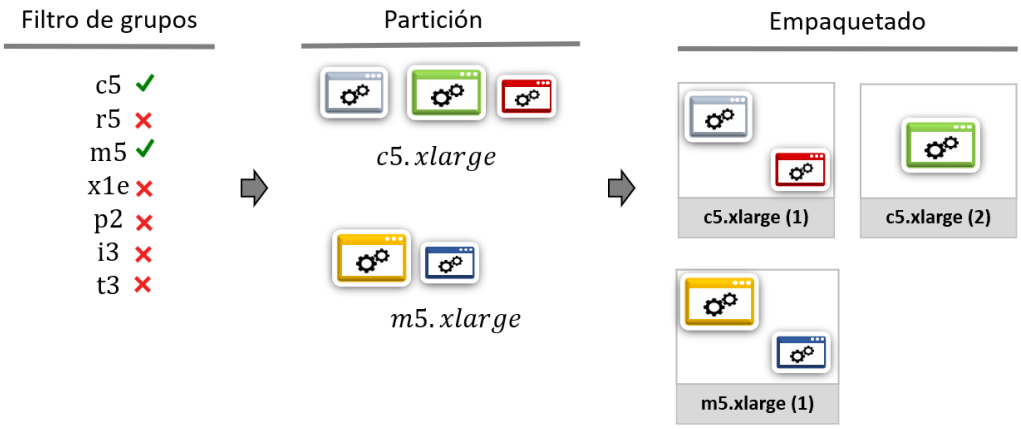

Figura 26. Filtro, partición, y empaquetado de máquinas virtuales. En la etapa uno se eligen los grupos c5 y m5. En la segunda etapa, el algoritmo WTS GA separa asigna las tareas computacionales en subconjuntos para los tipos c5.xlarge y m5.xlarge. Finalmente, el empaquetado de máquinas virtuales asigna las tareas en dos máquinas virtuales tipo c5.xlarge y una m5.xlarge.

3.12 Ramificación y acotamiento aproximado

Un algoritmo exacto de ramificación y acotamiento explora de manera implícita el espacio de soluciones de un problema combinatorio. Esta técnica utiliza una cola de prioridad para realizar una búsqueda en forma de árbol, donde el nodo raíz representa una solución vacía, los nodos del árbol contienen soluciones parciales construidas a partir de una serie de decisiones, y las hojas son soluciones completas. Adicionalmente, un componente crucial de cada nodo es la cota inferior, la cual se obtiene utilizando una versión relajada del problema y sirve para estimar cual es el mejor resultado al que puede aspirar la exploración de dicho nodo. Al momento de extraer un nodo de la cola de prioridad, este se compara con la mejor solución encontrada hasta el momento, y en caso de que la cota inferior sea mayor que la mejor solución entonces la exploración del nodo es ignorada, desechando todo el subárbol que podría obtenerse a partir de ahí. En caso contrario, el nodo es prometedor y se crean nuevos nodos hijos con diferentes opciones para el siguiente valor de la solución y se agregan a la cola de prioridad. En cada iteración de este algoritmo, un nodo de la cola de prioridad es procesado, y el procedimiento termina cuando no existen nodos en la cola de prioridad para explorar.

El algoritmo 20 es una variante aproximada del algoritmo de ramificación y acotamiento. En esta implementación, un nodo contiene una lista de tareas computacionales no asignadas, el valor de la cota inferior, y la solución parcial representada una lista de máquinas virtuales con alojamiento de tareas computacionales. Un nodo padre p, puede engendrar diferentes hijos x_k, donde un nodo hijo x_k representa la elección de continuar la construcción de la solución usando el tipo de máquina virtual como anfitriona de las tareas computacionales libres de p. De esta manera, la solución parcial de x_k se

constituye de solución parcial de p y el 15% de los máquinas virtuales resultantes de un procedimiento de empaquetado de las tareas computacionales libres de p en máquinas virtuales de tipo k. Consecuentemente, las tareas libres de x_k son actualizadas excluyendo aquellas que se encuentran en la solución parcial. Adicionalmente, la cota inferior de x_k se obtiene con la suma del costo de la solución parcial de x_k y el resultado de una versión relajada del modelo 2 donde las variables $Copies_i$ son definidas como tipo continuo, y la entrada son las tareas libres de x_k, y un subconjunto $T' \subseteq T$ de tipos de máquinas virtuales; donde T' es definida en base a los grupos de tipos de máquinas virtuales, donde cada grupo es representado por los dos tipos de máquina virtual cuya oferta de recursos computacionales se asemejan mejor a la demanda de recursos de las tareas libres de x_k.

El árbol de búsqueda del algoritmo 20 es generado por la extracción e inserción de nodos en pila de procesamiento donde el primer elemento en entrar es el último en salir. En una iteración del algoritmo, el nodo a procesar se extrae de la parte superior de la pila, y los hijos generados son insertados en la pila de manera ordenada por una función de prioridad que pondera la cota inferior de acuerdo con la cantidad de tareas computacionales que son parte de la solución parcial. De esta manera, el algoritmo generar un árbol de búsqueda en profundidad donde en cada nivel se intenta favorecer a los nodos con mejor potencial estimado por la cota inferior y el porcentaje de progreso de la solución parcial.

Algoritmo 20.1. Creación de nodo hijo

Entrada: x es el nodo padre,
k es el número del tipo de máquina virtual definido para asignar tareas computacionales.
Resultado: Nodo hijo x_k.
sea E_k el resultado del algoritmo de empaquetado first fit personalizado; /* Algoritmo 10 */
sea E'_k el subconjunto de las primeras máquinas virtuales de E_k que constituyen el 15%.
$l_k = \bigcup_{b \in E'_k} b_{load}$; /* tareas asignadas en máquinas virtuales */
sea x_k un nodo nuevo donde: $x_k.nivel \coloneqq x.nivel + 1$; $x_k.tareas \coloneqq x.tareas - l_k$;
$x_k.p \coloneqq \sum_{j \in x_k.tareas} p_j$; $x_k.m \coloneqq \sum_{j \in x_k.tareas} m_j$; $x_k.b \coloneqq \sum_{j \in x_k.tareas} b_j$;
$x_k.costo = x.costo + |E'_k|C_k$;
$x_k.costoNormalizado = x_k.costo \ raiz.cotaInferior$;
$x_k.solucion = x.solucion \cup E'_k$;
$T' = \widehat{M}_g \cup \widehat{M}'_g \cup M'_g$;
sea lb el resultado de una variante relajada del modelo *2* donde las tareas son $x_k.tareas$, los tipos de máquinas virtuales son T', y las variables $Copies_i$ son definidas como continuas;
$x_k.cotaInferior = x_k.costo + lb$;
$x_k.cotaInferiorNormalizada = x_k.cotaInferior \setminus raiz.cotaInferior$;
$x_k.cobertura = \left|\bigcup_{b \in x_k.solucion} b_{load}\right| \setminus M$;
return x_k;

Algoritmo 20. Ramificación y acotamiento aproximado

Entrada: J es el conjunto de tareas computacionales a empaquetar,
T es el conjunto de tipos de máquinas virtuales a considerar
Resultado: Costo de la solución
/* Elección de grupos permitidos */
sea A el conjunto de grupos existentes en la solución de asignación simple (sección 3.6);
/* inicialización de búsqueda */
sea *raiz* un nodo;
$raiz.nivel = 1$;
$raiz.tareas = \{1, \dots, M\}$;
$raiz.cotaInferior = modeloCotaInferior(raiz.tareas, A)$;
$raiz.cotaInferiorNormalizada = 1$;
$raiz.p := \sum_{j \in raiz.tareas} p_j$; $raiz.m := \sum_{j \in raiz.tareas} m_j$; $raiz.b := \sum_{j \in raiz.tareas} b_j$;
sea S una pila de procesamiento;
$S.\text{push}(raiz)$;
sea $mejor = NIL$;
while $|S| > 0$ **do:**
  $x = S.\text{pop}(\ )$;
  if $|x.tareas| = 0$ and $(mejor = NIL$ or $x.costo < mejor.costo)$ **then:**
    $mejor := x$;
    continuar en siguiente iteración del ciclo while; /*nodo ya no tiene tareas para asignar*/
  if $mejor \neq NIL$ and $mejor.costoNormalizado \leq x.cotaInferiorNormalizada)$ **then:**
    continuar en siguiente iteración del ciclo while; /*ignorar nodo por falta de potencial*/
  /*menor capaz de satisfacer demanda*/
  $\widehat{M}_g := \arg\min_{k \in \{1,\dots,M\}} \{F_k | G_k = g, x.p \leq P_k, x.m \leq M_k, x.b \leq B_k\} \ \ \forall\, g \in A$;
  /*mayor incapaz de satisfacer demanda*/
  $\widehat{M}'_g := \arg\max_{k \in \{1,\dots,M\}} \left\{F_k | G_k = g, F_k < F_{\widehat{M}_g}\right\} \ \ \forall\, g \in A$;
  /*mayor del grupo*/
  $M'_g := \arg\max_{k \in \{1,\dots,M\}} \{F_k | G_k = g\} \ \ \forall\, g \in A$;
  $\widehat{T} = \left\{ \arg\min_{k \in \widehat{M}_g \cup M'_g} \{F_k\} \,\middle|\, g \in A \right\}$;
  sea L una lista vacía;
  for $k \in \widehat{T}$ **then:**
    if $|E_k| > 0$ **then:**
      $x_k := nodoHijo(k, E_k)$; /*Algoritmo 20.1*/
      agregar x_k a L;
  ordenar L con $prioridad(l \in L) = l.cotaInferiorNormalizada \times (1 - l.cobertura)$;
  for $l \in L$ **then:**
    $S.\text{push}(l)$;
return $mejor$;

Capítulo 4. Análisis experimental

4.1 Configuración Experimental

Esta sección describe la colección de tipos de máquinas virtuales, instancias del problema, algoritmos, y ambiente de ejecución que conforman la configuración de los experimentos realizados.

4.1.1 Tipos de máquinas virtuales de EC2

Obtener la información de la colección de tipos de máquinas virtuales disponibles en EC2 directamente del sitio web https://aws.amazon.com/ec2/pricing/on-demand/es una labor manual intensiva. Una alternativa popular a este enfoque es usar el servicio Easy Amazon Instance Comparison que recopila periódicamente los precios de los tipos de máquinas virtuales publicadas en los servicios web de EC2 y los consolida en un archivo CSV fácil de consultar. Aunque el archivo antes mencionado contiene la mayor parte de la información necesaria para nuestro estudio, es necesario complementarla con la lista de factores de tamaño provista por Barr en 2017, además del reporte técnico del rendimiento de ancho de banda de los tipos de máquinas virtuales ampliables realizado por Gieniec en 2019. Este estudio experimental utiliza los tipos de máquinas virtuales Linux On-Demand Reserved del centro de datos US-East N.Virginia de EC2, y la información de costos fue obtenida de las fuentes antes mencionadas en agosto de 2019.

El costo de un tipo de máquina virtual de EC2 varía de acuerdo a los recursos computacionales que ofrece, tales como CPU, memoria, ancho de banda, almacenamiento, y unidades cómputo acelerado GPU. Dado que este estudio solo toma en consideración los primeros tres, el conjunto de 69 tipos de máquinas virtuales relevantes se obtiene por medio de Análisis de Frente Pareto sobre la colección total de 178 tipos de máquinas virtuales ofrecidas por EC2. Para este propósito, cada tipo de máquina virtual T_i es caracterizado como un vector $vm_i = [C_i, -P_i, -M_i, -B_i]$, utilizando el concepto de dominancia de pareto (ecuación 8) establecemos cuales tipos de máquinas $vm_i \in VM$ carecen de relevancia debido a que existen otros tipos de máquinas de virtuales que son mejores en al menos una característica y de igual valor en las demás. De esta manera, utilizar solamente el conjunto de tipos de máquinas virtuales que no dominados (ecuación 9) permite reducir el espacio de búsqueda al mismo tiempo que conserva las opciones que ofrecen mayor costo-beneficio. El apéndice A.1 contiene el listado resultante.

$$A \preccurlyeq B \Leftrightarrow \forall k \in \{1,2,3,4\}\ a_k \leq b_k, \text{ and } \exists k \in \{1,2,3,4\},\ a_k < b_k. \quad (8)$$

$$Front = \{vm_i \in VM \mid vm_j \not\preccurlyeq vm_i \quad \forall\ vm_j \in VM\}. \quad (9)$$

Finalmente, es necesario mencionar que este trabajo de investigación utiliza la Unidad de Cómputo Elástico (ECU por sus siglas en inglés) propuesta por EC2 como la medida que representa la capacidad de CPU asignada a tipo de máquina virtual en particular. De acuerdo a EC2, un ECU equivale a la capacidad de un procesador Intel Xeon del año 2017 con 1.2 Ghz de ciclo de reloj.

4.1.2 Instancias del problema

Una instancia del problema de nuestro estudio consiste de N tareas computacionales que se caracterizan por su demanda de CPU, memoria, y ancho de banda. Una tarea computacional puede ser clasificada como Intensiva en Cómputo (CI), Intensiva en Memoria (MI), e Intensiva en Ancho de Banda (NI). En este trabajo, cada una de estas clases es definida como un conjunto de intervalos cerrados (Tabla 9) que representan la proporción de CPU, memoria y ancho de banda que una tarea computacional consume con respecto a una máquina virtual específica. En este sentido, una tarea computacional aleatoria se genera eligiendo de manera aleatoria un tipo de máquina virtual de la colección y usando los intervalos de consumo para definir la demanda de recursos computacionales. El apéndice A.3 presenta detallamente el procedimiento de generación de tareas computacionales aleatorias para este estudio.

El conjunto de problemas de este estudio contiene 30 instancias generadas aleatoriamente para cada combinación de tres tamaños de problema (500, 1000, 5000) y cuatro tipos de cargas de trabajo (Intensiva en Cómputo, Intensiva en Memoria, Intensiva en Ancho de Banda, Mixta), que acumulan un total de 360 instancias (Tabla 8).

Tabla 8. Instancias del problema

Cantidad de tareas	Tipo de carga de trabajo	Instancias	Abreviación
500	Intensiva en Cómputo	30	CI500
500	Intensiva en Memoria	30	MI500
500	Intensiva en Ancho de Banda	30	NI500
500	Mixta	30	MX500
1000	Intensiva en Cómputo	30	CI1000
1000	Intensiva en Memoria	30	MI1000
1000	Intensiva en Ancho de Banda	30	NI1000
1000	Mixta	30	MX1000
5000	Intensiva en Cómputo	30	CI5000
5000	Intensiva en Memoria	30	MI5000
5000	Intensiva en Ancho de Banda	30	NI5000
5000	Mixta	30	MX5000

Tabla 9. Demanda de recursos de diferentes tipos de tareas computacionales

Tipo de Tarea	CPU	Memoria	Ancho de Banda
Intensiva en Cómputo	[0.5 ,0.95]	[0.03 , 0.15] de mínimo	[0.03 , 0.15] de mínimo
Intensiva en Memoria	[0.25 , 0.5]	[0.6 , 1.0]	[0.03 , 0.15] de mínimo
Intensiva en Ancho de Banda	[0.03 , 0.1]	[0.03 , 0.1]	[0.5 , 1.0]

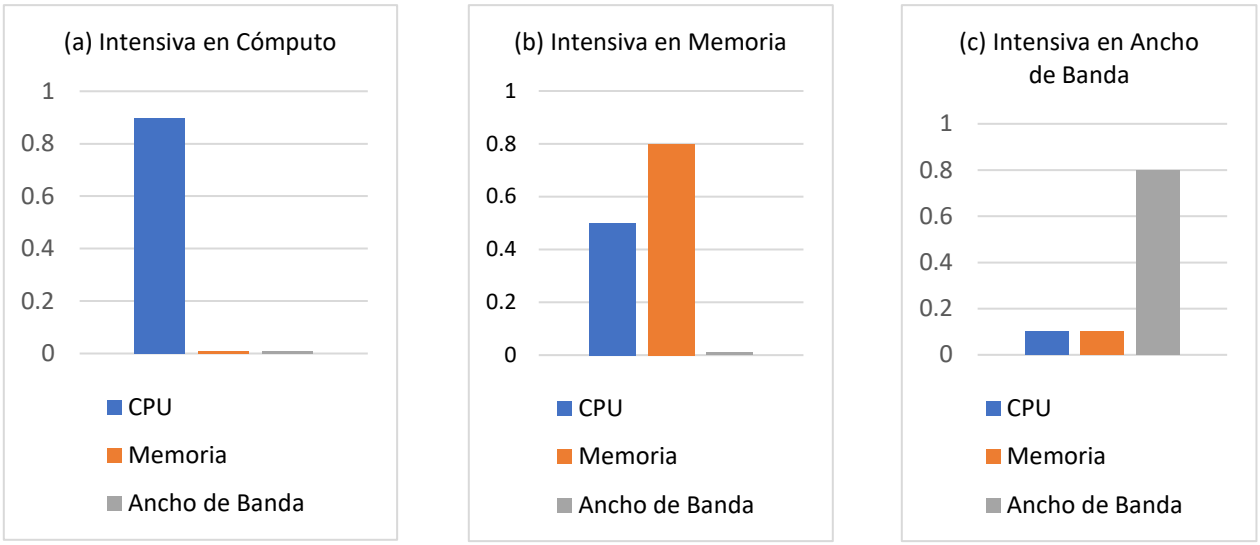

Figura 27. Ejemplos de tareas computacionales. (a): tarea intensiva en cómputo que consume 85% de la capacidad del procesador de la máquina virtual que la aloja. (b): tarea intensiva en memoria que consume 80% de memoria y 50% de la capacidad del procesador de la máquina virtual anfitriona. (c): Tarea intensiva en ancho de banda con consumo mínimo de procesador y memoria.

4.1.3 Implementación de algoritmos

En este trabajo de investigación lo algoritmos fueron codificados en el lenguaje Java bajo el ambiente JDK 11. JMetal 5.6 fue utilizado para implementar todos los algoritmos genéticos de este proyecto. Aquellos algoritmos que requieren resolver modelos MIP utilizan el optimizador Gurobi con licencia académica gratuita a través de la API de Java.

4.1.4 Entorno computacional

La computadora donde se realizaron los experimentos tiene las siguientes características:

Tabla 10. Características de computadora

Marca	MSI
Modelo	GL6595C
Tipo	Laptop
Memoria	8 GB
Procesador	Intel i5-9300H
Velocidad de reloj	2.4 GHz
Sistema operativo	Windows 10 Home (64 bit)

4.1.5 Algoritmos estudiados

La Tabla 11 muestra el listado de algoritmos que fueron parte de los experimentos e identifica con una letra a cada uno de ellos.

Tabla 11. Lista de algoritmos

Algoritmo	Id
Divide y vencerás BLMP	A
Consolidación de subsecuencias	B
Búsqueda local de consolidación del mejor par (BLMP)	C
BLMP: Filtro, partición, y empaquetado de máquinas virtuales	D
BLMP: Ramificación y acotamiento aproximado 480.0	E
BLMP: Empaquetado de máquinas virtuales (first fit personalizado) y cobertura de conjuntos	F
BLMP: Empaquetado de máquinas virtuales (algoritmo genético) y cobertura de conjuntos	G
BLMP: Empaquetado de máquinas virtuales (best fit ordenado) y cobertura de conjuntos	H
BLMP: DTP PLS GA FULL_SEARCH - LTR	I
BLMP: DTP PLS GA FULL_SEARCH - NEXT_AND_BEST	J
BLMP: DTP PLS GA INDEXED_FULL_SEARCH - LTR	K
BLMP: DTP PLS GA INDEXED_FULL_SEARCH - NEXT	L
BLMP: DTP PLS GA LIMITED_SEARCH - LTR	M
BLMP: DTP PLS GA LIMITED_SEARCH - LTR_AND_BEST	N
BLMP: DTP PLS GA LIMITED_SEARCH - NEXT	O
BLMP: DTP PLS GA LIMITED_SEARCH - NEXT_AND_BEST	P
BLMP: DTP SETUP GA	Q
BLMP: DTP SETUP_BY_GROUP GA	R
BLMP: DTP SETUP_BY_GROUP GA (LTR)	S
BLMP: WTS GA	T
BLMP: Partición y empaquetado de máquinas virtuales	U
Horizonte rodante para elección de tipo de máquina virtual	V
Asignación simple	W
Rebalanceo : BLMP	X

4.2 Resultados

Esta sección proporciona una interpretación de diferentes métricos de rendimiento mediante la experimentación de distintos algoritmos, tales como costo promedio de solución, tiempo de ejecución, y cantidad de veces que el algoritmo produjo la solución de mayor calidad. En este análisis, el costo de solución es normalizado con respecto a la cota inferior producida por el Algoritmo 1. Con el fin de simplificar la presentación visual, los resultados experimentales son mostrados utilizando la letra que identifica a cada algoritmo.

En los resultados expuestos en las siguientes secciones se puede observar que a medida que el tamaño de los problemas crece, la cantidad de algoritmos evaluados disminuye. Esta elección fue motivada principalmente por el tiempo disponible para efectuar experimentos que tuvieran potencial de generar resultados favorables.

Los experimentos con problemas de tamaño 500 son procesados con todos los algoritmos. En contraste, en los problemas de tamaño 1000, cuatro variantes del algoritmo de consolidación basado en permutación fueron descartados debido a su bajo desempeño computacional. El primer par excluido son los algoritmos I y J que utilizan búsqueda secuencial sobre el conjunto de tipos de máquinas virtuales (FULL SEARCH) en la evaluación de un individuo. El siguiente par excluido son los algoritmos N y P que utilizan una etapa de consolidación del mejor par con complejidad computacional $O(M^2)$ en el proceso de decodificación.

En los problemas de tamaño 5000 se eligieron aquellos algoritmos que mostraron buen rendimiento en los problemas de tamaño 500 y 1000. Adicionalmente, se evaluaron los algoritmos simples de gran velocidad con el objetivo de usar sus resultados como valores de referencia. De las variantes del algoritmo de consolidación basado en permutación (I-P) se ejecutaron los algoritmos L, M y O, excluyendo a K debido a que es muy similar al algoritmo M y ligeramente menos efectivo. Similarmente, otros algoritmos excluidos debido a su baja expectativa de éxito con base al desempeño observado en problemas de tamaño menor son U y X. Finalmente, los algoritmos H y F de la estrategia de empaquetado de máquinas virtuales y cobertura de conjuntos fueron seleccionados, el primero se ejecutó para todos los problemas de este tamaño, mientras que el segundo solo fue evaluado en los problemas CI5000 debido a que solo exhibió buenos resultados en CI500 y CI1000.

4.2.1 Tareas intensivas en cómputo

Los resultados experimentales indican que los algoritmos F, Q, y U son preferibles para problemas con este tipo de carga de trabajo. La figura 27 muestra que en problemas de tamaño 500, 1000 y 5000 el algoritmo F produce las mejores soluciones la mayoría del tiempo, mientras que el algoritmo Q aparece como el mejor algoritmo para algunas instancias del problema con tamaño 500 y 1000. Estos algoritmos pueden considerarse efectivos para esta carga de trabajo debido a que la tabla 12 y la figura 29 revelan que el 95 % del tiempo las soluciones producidas por estos algoritmos son a lo más 1.75% mas costosas que la cota inferior del costo proporcionada por el algoritmo V. Finalmente, la figura 32 confirma que estos algoritmos exhiben el comportamiento esperado de elegir una mayoría de máquina virtuales tipo C5 que pertenecen a la familia optimizada para cómputo de EC2.

Tabla 12. Rendimiento de los mejores algoritmos (CI)

	Ganador	Costo				Duración (Seg)
Id	%	Promedio	Desviación típica	Intervalo de confianza 95%		Promedio
CI500						
F	73.3333	1.0149	0.0007	1.0146	1.0151	3
Q	20.0000	1.0167	0.0021	1.0160	1.0175	14
U	6.6667	1.0202	0.0064	1.0179	1.0225	232
H	0.0000	1.0208	0.0013	1.0203	1.0213	10
C	0.0000	1.0234	0.0020	1.0227	1.0242	9
W	0.0000	1.2262	0.0088	1.2231	1.2293	0.0011
CI1000						
F	96.6667	1.0138	0.0005	1.0136	1.0139	85
Q	3.3333	1.0154	0.0013	1.0149	1.0158	92
H	0.0000	1.0204	0.0010	1.0200	1.0207	42
U	0.0000	1.0204	0.0052	1.0186	1.0223	292
C	0.0000	1.0228	0.0011	1.0224	1.0232	130
W	0.0000	1.2276	0.0063	1.2253	1.2298	0.0015
CI5000						
F	100.0000	1.0123	0.0002	1.0122	1.0124	411
Q	0.0000	1.0136	0.0007	1.0133	1.0139	1205
S	0.0000	1.0201	0.0003	1.0200	1.0202	1029
C	0.0000	1.0220	0.0003	1.0219	1.0221	138
W	0.0000	1.2280	0.0036	1.2267	1.2293	0.0045

El tiempo de ejecución de los algoritmos F, Q, y U es razonable para problemas de tamaño 500 y 1000, donde el algoritmo U es el mayor duración con un promedio de 262 segundos, mientras que los algoritmos F y Q toman menos de 90 segundos en terminar, como se puede apreciar en la figura 30 y la tabla 12. En el caso de problemas de tamaño 5000, los algoritmos Q y S requieren aproximadamente 1200 segundos

para concluir, mientras que el algoritmo F requiere alrededor de 400 segundos. Por último, cabe hacer notar que el algoritmo C basado en búsqueda local genera rápidamente soluciones con costo normalizado cercano a 1.024.

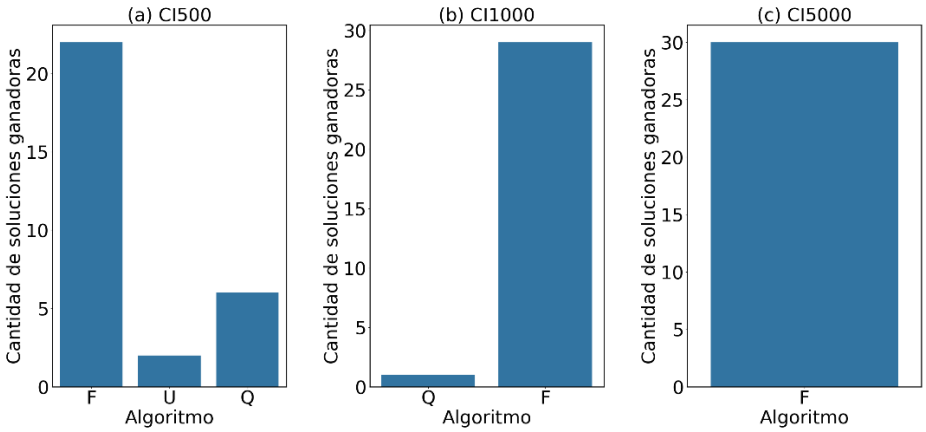

Figura 28. Algoritmos que producen las mejores soluciones para 30 experimentos con tareas computacionales CI. El algoritmo F genera la mayor parte de las soluciones con menor costo para CI500, CI1000, y CI5000. El algoritmo Q provee una porción menor para CI500 y CI1000.

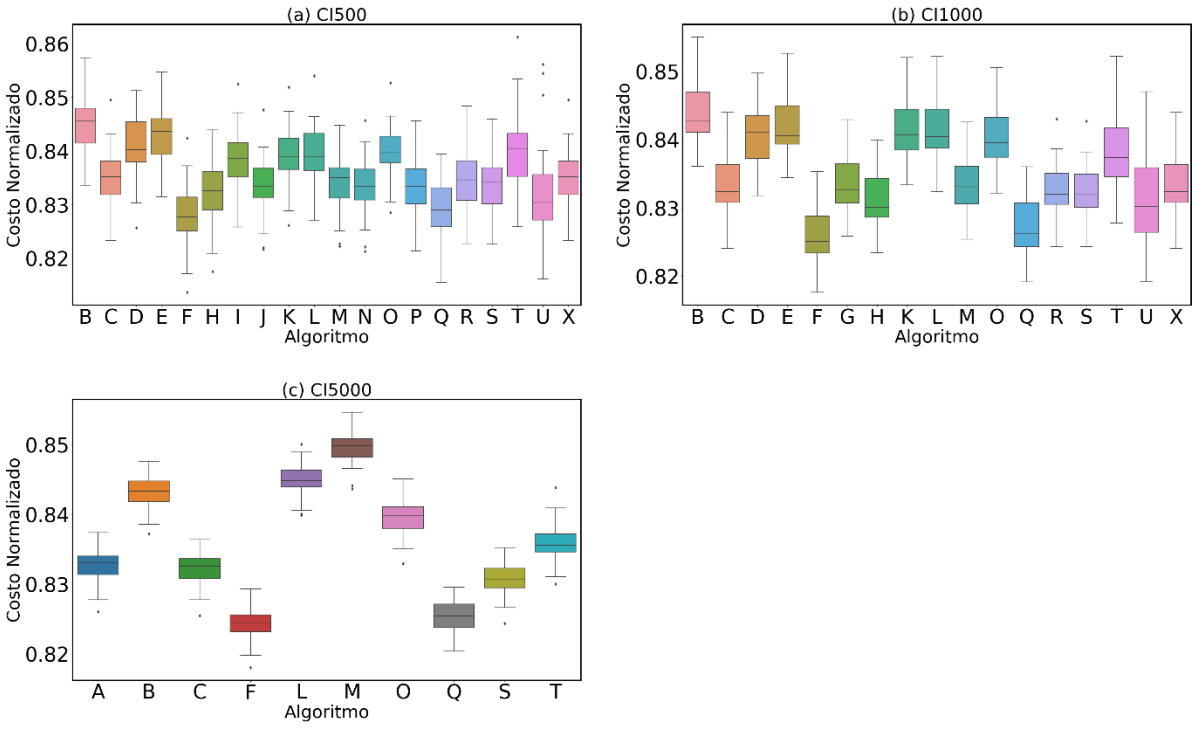

Figura 29. Costo promedio y desviación típica de algoritmos en cargas de trabajo CI. El algoritmo F presenta el mejor costo normalizado promedio.

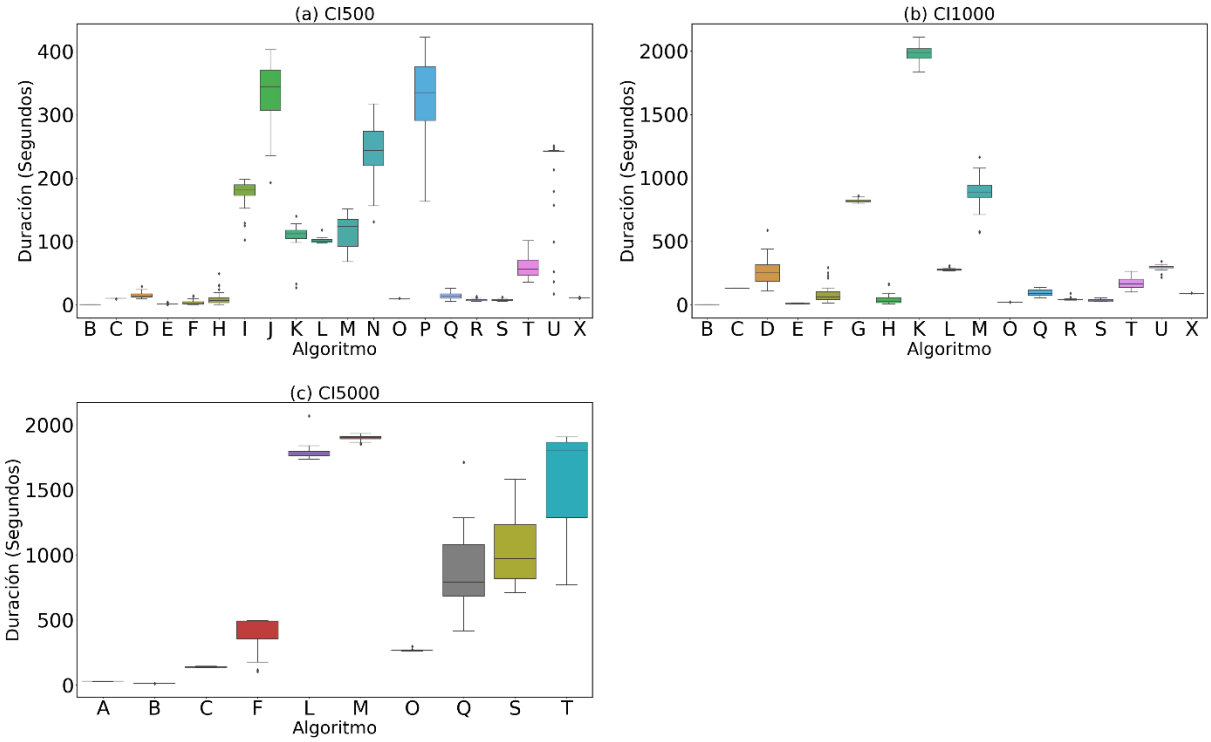

Figura 30. Tiempo de ejecución de algoritmos en cargas de trabajo CI. Los algoritmos F y Q se encuentran en la banda inferior de CI500, CI1000 y CI5000. Los algoritmos Q y S son mas rápidos que el resto de las metaheurísticas (M, L, T) en CI5000.

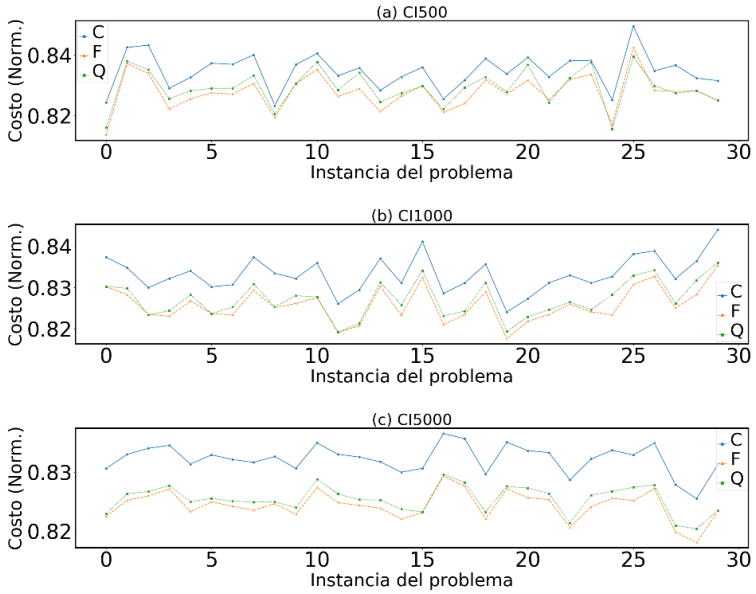

Figura 31. Comparación de costos de los mejores algoritmos en cargas de trabajo CI. En la subfigura a, el costo de las soluciones del algoritmo F es cercano a 1.015, mientras que el algoritmo Q produce soluciones cuyo costo ocasionalmente es mayor a 1.020. La subfigura b muestra que en CI1000, el algoritmo F generalmente soluciones ligeramente mas baratas que el algoritmo Q. En CI500, el algoritmo F genera mejores soluciones que Q.

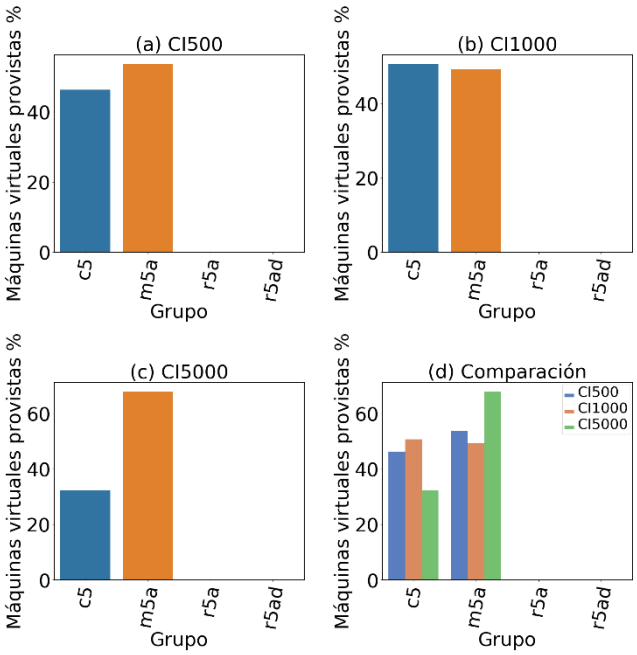

Figura 32. Elección de tipos de máquinas virtuales de los mejores algoritmos en cargas de trabajo CI. En CI500, las soluciones de mejor calidad eligieron 40% de máquinas virtuales tipo C5 y 60% de máquinas virtuales tipo m5a. De manera similar, las soluciones de CI1000 muestran una distribución balanceada entre estos tipos máquinas virtuales. Por último, para cargas de trabajo CI5000 los mejores algoritmos dan preferencia a las máquinas virtuales tipo m5a sobre las máquinas virtuales tipo C5.

4.2.2 Tareas intensivas en memoria

Los resultados indican que los algoritmos genéticos con codificación entera (S) o de permutación (I, J, K), y las heuristicas híbridas proven soluciones efectivas para este tipo de problemas. La figura 33 muestra que para problemas MI500 los algoritmos genéticos con codificación de permutación (I, K) generan las mejores soluciones la mayor parte del tiempo, mientras que en MI1000 y MI5000 los algoritmos genéticos con codificación entera (S, Q) y la estrategia de empaquetado de máquinas virtuales con etapa posterior de cobertura de conjuntos (H) producen las soluciones con mayor calidad.

Las cargas de trabajo de tareas intensivas en memoria parecen propiciar un rendimiento excelente de estos algoritmos. La tabla 13 y la figura 35 indican que generalmente el costo de las soluciones es entre 0.34% a 0.75% mas grande que la cota inferior producida por el algoritmo V. Es notable que el algoritmo de fusión B que es un componente clave de los algoritmos I,K,y S, genera por si mismo soluciones de alta calidad con un costo normalizado que oscila entre 1.008 y 1.011. Finalmente, la figura 34 provee confirmación al mérito de las mejores soluciones encontradas, demostrando que la mayor porción de los

tipos de máquinas virtuales elegidos (x1,x1e,r5,r5a) pertenecen a la familia optimizada para consumo de memoria.

La evaluación de los tiempos de ejecución presentados en la tabla 13 y la figura 36 muestra que el algoritmo K usualmente termina al exceder el tiempo límite establecido de 1800 segundos en MI1000, mientras que otros algoritmos necesitan menos tiempo para concluir. Del mismo modo, el mejor algoritmo para problemas MI5000 (H) require menos de 10 segundos para finalizar, y la duración promedio del segundo mejor algoritmo (S) es de aproximadamente 722 segundos.

Tabla 13. Rendimiento de los mejores algoritmos (MI)

	Ganador	Costo				Duración (Seg)
Id	%	Promedio	Desviación típica	Intervalo de confianza 95%		Promedio
MI500						
I	50.0000	1.0040	0.0016	1.0034	1.0046	191
K	46.6667	1.0040	0.0013	1.0035	1.0044	111
U	3.3333	1.0066	0.0026	1.0057	1.0075	1
J	3.3333	1.0068	0.0013	1.0063	1.0072	354
H	0.0000	1.0057	0.0016	1.0051	1.0062	0.01
B	0.0000	1.0108	0.0018	1.0101	1.0114	0.001
Q	0.0000	1.0124	0.0037	1.0110	1.0137	16
C	0.0000	1.0197	0.0024	1.0188	1.0206	11
W	0.0000	1.3558	0.0167	1.3499	1.3618	0.0003
MI1000						
K	46.6667	1.0047	0.0011	1.0043	1.0051	1851
H	46.6667	1.0048	0.0013	1.0043	1.0053	0.8
U	6.6667	1.0054	0.0016	1.0049	1.0060	10
B	0.0000	1.0085	0.0014	1.0080	1.0090	0.01
Q	0.0000	1.0109	0.0032	1.0097	1.0121	116
C	0.0000	1.0183	0.0018	1.0176	1.0189	152
W	0.0000	1.3566	0.0134	1.3518	1.3614	0.0006
MI5000						
H	66.66	1.0040	0.0006	1.0038	1.0042	5.2
S	33.33	1.0041	0.0006	1.0039	1.0043	722
B	0.0000	1.0067	0.0007	1.0064	1.0069	19
Q	0.0000	1.0082	0.0025	1.0073	1.0091	1255
C	0.0000	1.0177	0.0007	1.0174	1.0179	171
W	0.0000	1.3570	0.0050	1.3552	1.3588	0.0033

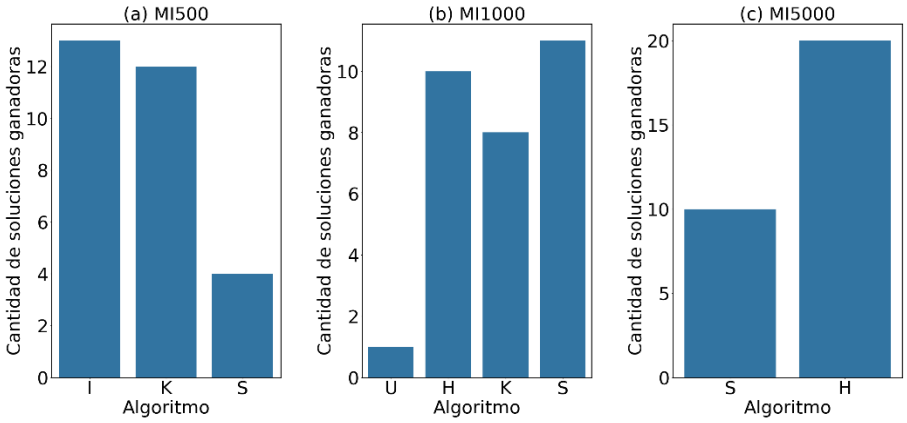

Figura 33. Algoritmos que producen las mejores soluciones para 30 experimentos con tareas computacionales MI. Los algoritmos genéticos con codificación de permutación I, J, y K producen las mejores soluciones en MI500. La heurística híbrida implementada por el algoritmo H es un fuerte competidor del algoritmo K en MI1000. El algoritmo H tiene mayor porcentaje de éxito que el algoritmo S en los problemas de tamaño 5000.

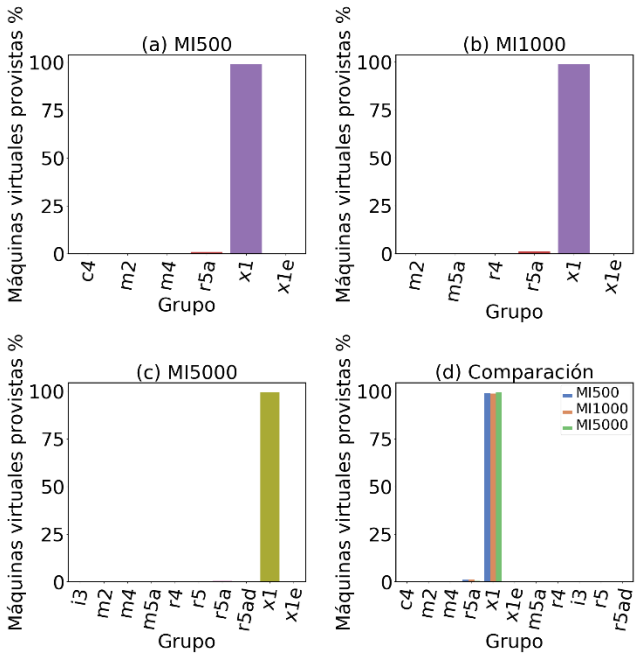

Figura 34. Elección de tipos de máquinas virtuales de los mejores algoritmos en cargas de trabajo MI. Las soluciones de mejor calidad están constituidas de manera predominante por máquinas virtuales optimizadas en memoria (x1).

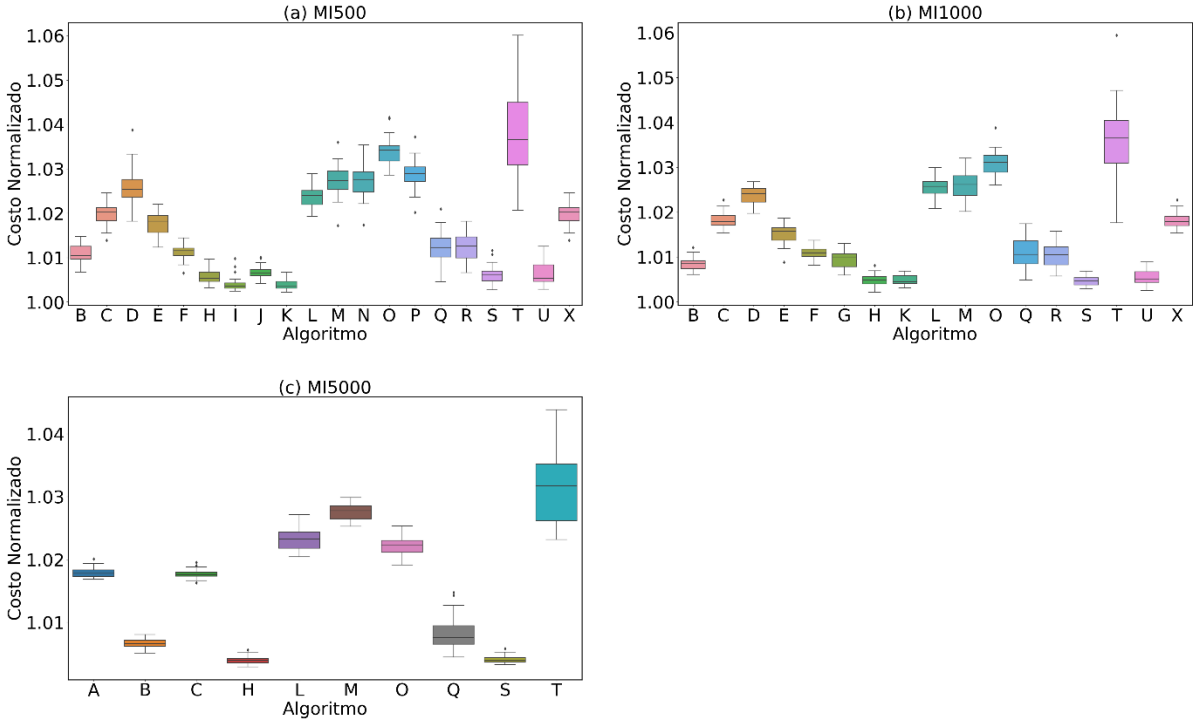

Figura 35. Costo promedio y desviación típica de algoritmos en cargas de trabajo MI. Los algoritmos genéticos I, y K superan al resto en MI500. De manera similar, el algoritmo K, y H proveen el mejor costo en MI1000. Finalmente, los algoritmos H y S producen soluciones con un costo normalizado promedio cercano a 1.004 para problemas MI5000.

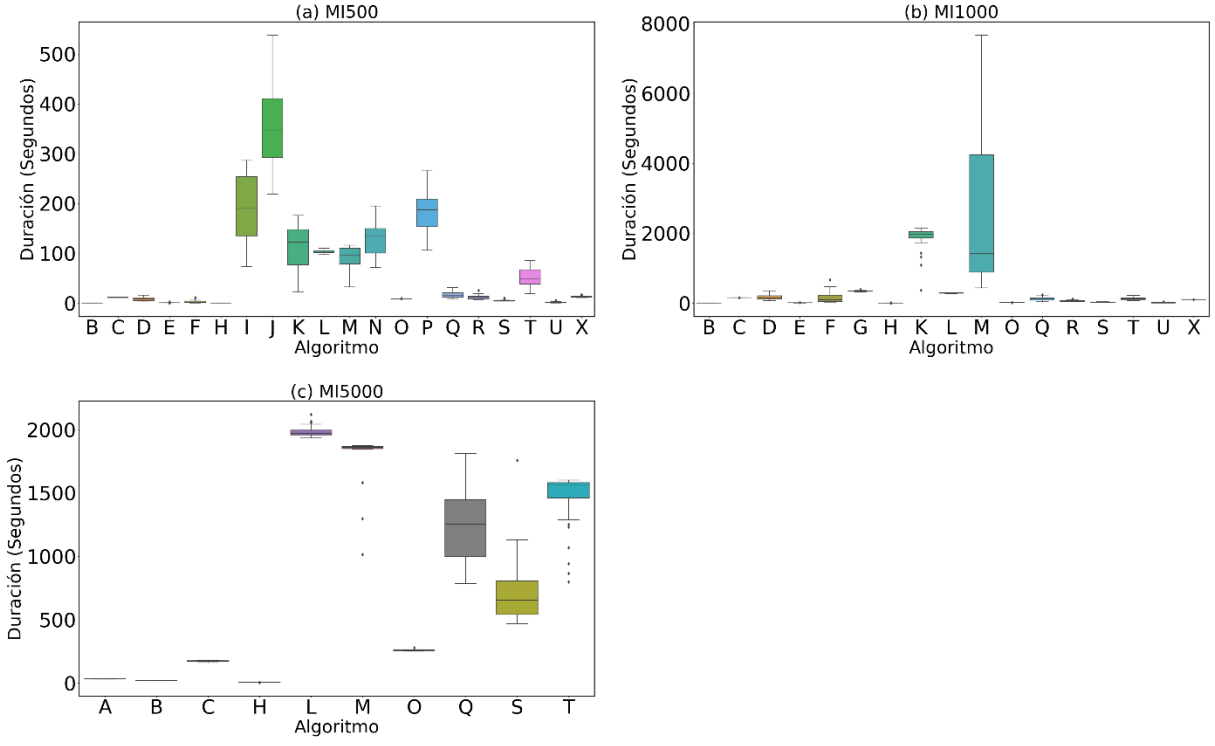

Figura 36. Tiempo de ejecución de algoritmos en cargas de trabajo MI. Los algoritmos H y U son considerablemente mas rápidos que el algoritmo K en MI500, y MI1000. De manera similar, el algoritmo H es más rápido que el algoritmo S en MI5000.

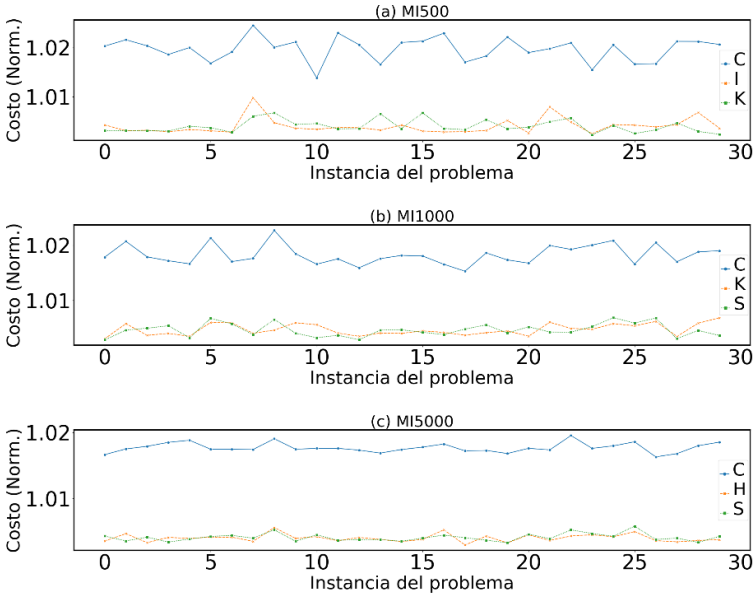

Figura 37. Comparación de costos de los mejores algoritmos en cargas de trabajo MI. En MI500, los algoritmos I, y K producen soluciones de costo similar y 30% de las veces convergieron a la misma solución. Los algoritmos H, y K producen soluciones cuya diferencia absoluta promedio es de 0.0009 en MI1000. El algoritmo S y H muestran un rendimiento muy cercano entre sí en MI5000.

4.2.3 Tareas intensivas en ancho de banda

En este escenario, la tabla 14 y la figura 38 revelan que la familia de algoritmos genéticos con codificación de permutación (P, J, I, N, K, M) siempre elaboran las mejores soluciones para problemas de tamaño 500, y proporcionan el 40% de las soluciones de primer nivel en problemas de 1000, lo cual contrasta con el rendimiento en NI5000 donde el mejor resultado del algoritmo M es de segundo nivel. De manera similar, es notable que las heurísticas híbridas H y G proveen una porción amplia de las soluciones de mejor calidad en problemas de tamaño mediano (1000) y grande (5000), mientras que los algoritmos genéticos con codificación entera (Q, S) muestran un comportamiento ligeramente inferior en los mismos problemas. La revisión de los resultados experimentales indican que las soluciones producidas por estos algoritmos consisten mayormente de máquinas virtuales optimizadas para consumo de ancho de banda (c5n y P3dn) como lo muestra la figura 41, y tienen costo cuya diferencia con respecto a la cota inferior (algoritmo V) está situada típicamente entre 2.1% y 4.5% (tabla 14 y figura 39). La figura 40 ilustra el tiempo de ejecución de los algoritmos examinados y evidencía que la familia de algoritmos genéticos con codificación de permutación (P, J, I, N, K, M) requieren el tiempo de cómputo mayor en los problemas de todos los tamaños. Adicionalmente, los mejores algoritmos genéticos para problemas de tamaño grande (Q y S)

necesitan de 800 a 1200 segundos para concluir, mientras que el algoritmo de búsqueda local H provee soluciones de gran calidad en menos de 7 segundos.

Tabla 14. Rendimiento de los mejores algoritmos (NI)

	Ganador	Costo				Duración (Seg)
Id	%	Promedio	Desviación típica	Intervalo de confianza 95%		Promedio
NI500						
P	46.6667	1.0254	0.0105	1.0217	1.0292	209
J	36.6667	1.0245	0.0097	1.0210	1.0280	323
N	10.0000	1.0360	0.0173	1.0299	1.0422	110
I	3.3333	1.0372	0.0157	1.0315	1.0428	168
K	3.3333	1.0385	0.0192	1.0316	1.0454	95
M	0.0000	1.0361	0.0178	1.0297	1.0424	56
Q	0.0000	1.0431	0.0164	1.0372	1.0489	9
H	0.0000	1.0478	0.0173	1.0416	1.0540	0.01
C	0.0000	1.0663	0.0176	1.0600	1.0726	11
W	0.0000	1.7821	0.0377	1.7686	1.7956	0.0003
NI1000						
M	40.0000	1.0320	0.0105	1.0283	1.0358	4351
Q	23.3333	1.0331	0.0102	1.0294	1.0368	62
H	23.3333	1.0335	0.0104	1.0297	1.0372	1
G	16.6667	1.0313	0.0097	1.0278	1.0348	618
K	0.0000	1.0401	0.0123	1.0357	1.0445	1252
C	0.0000	1.0504	0.0113	1.0464	1.0545	146
W	0.0000	1.7549	0.0295	1.7444	1.7655	0.0008
NI5000						
H	53.3333	1.0274	0.0053	1.0255	1.0294	7
S	30.0000	1.0289	0.0051	1.0270	1.0307	813
Q	13.3333	1.0296	0.0053	1.0277	1.0314	1227
T	3.3333	1.0453	0.0041	1.0438	1.0467	1846
C	0.0000	1.0491	0.0035	1.0478	1.0503	131
M	0.0000	1.0567	0.0051	1.0549	1.0586	1827

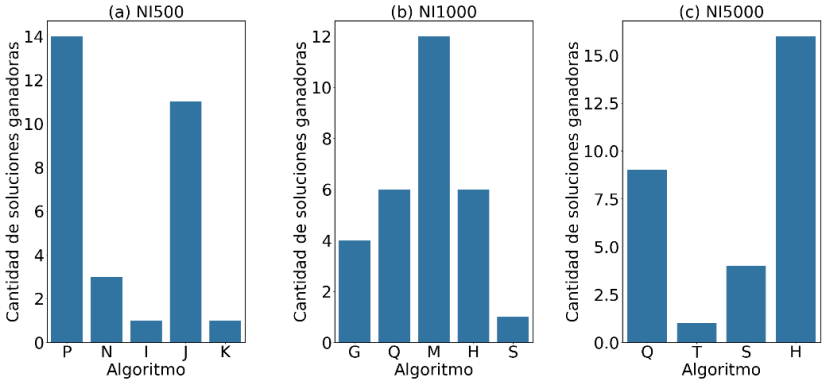

Figura 38. Algoritmos que producen las mejores soluciones para 30 experimentos con tareas computacionales NI. Los algoritmos DTP_PLS_GA con decodificación NEXT_AND_BEST (P y J) representa alrededor del 80% en NI500. DTP_PLS_GA con decodificación LTR (M) contribuye 40% en NI1000, mientras el enfoque Bin Packing And Set Cover (H y G) provee 23% en NI1000. En NI5000, el H genera el 53%, y los algoritmos genéticos de selección de máquinas virtuales (Q y S) producen el 43% de las mejores soluciones.

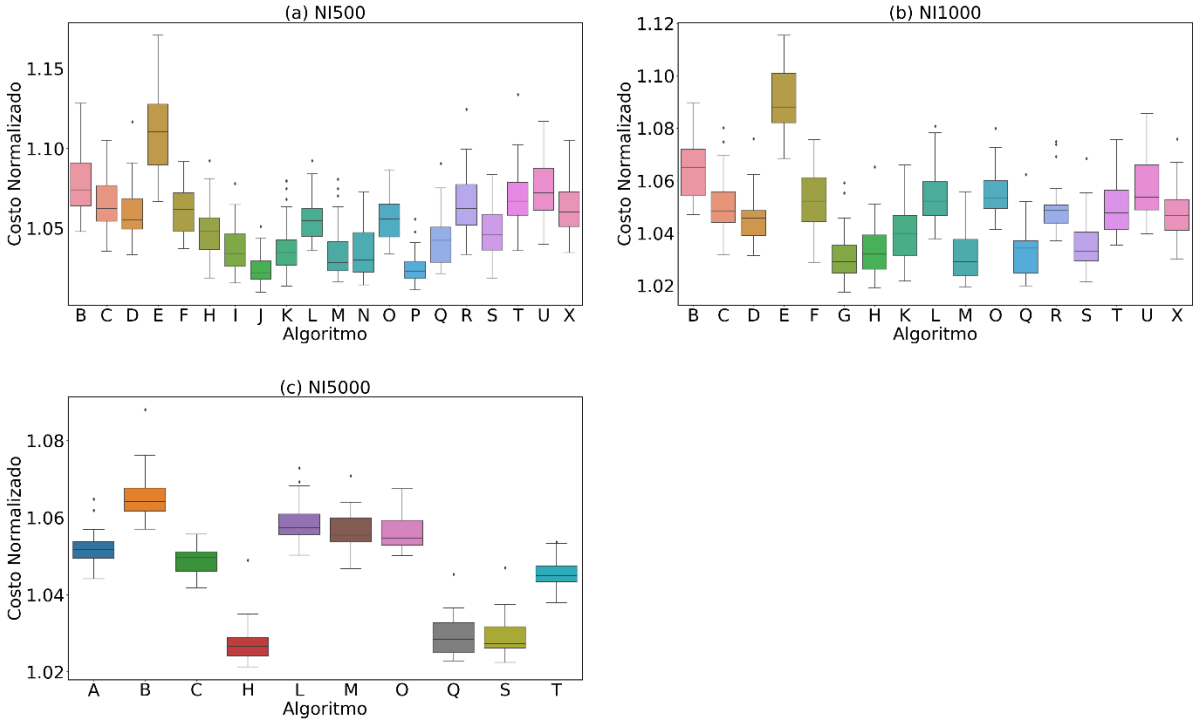

Figura 39. Costo promedio y desviación típica de algoritmos en cargas de trabajo NI. Los algoritmos P y J muestran el menor costo normalizado promedio en NI500. Los algoritmos G y M exhiben el mejor costo normalizado promedio en NI1000. Los algoritmos H,Q y S claramente superan al resto en NI5000.

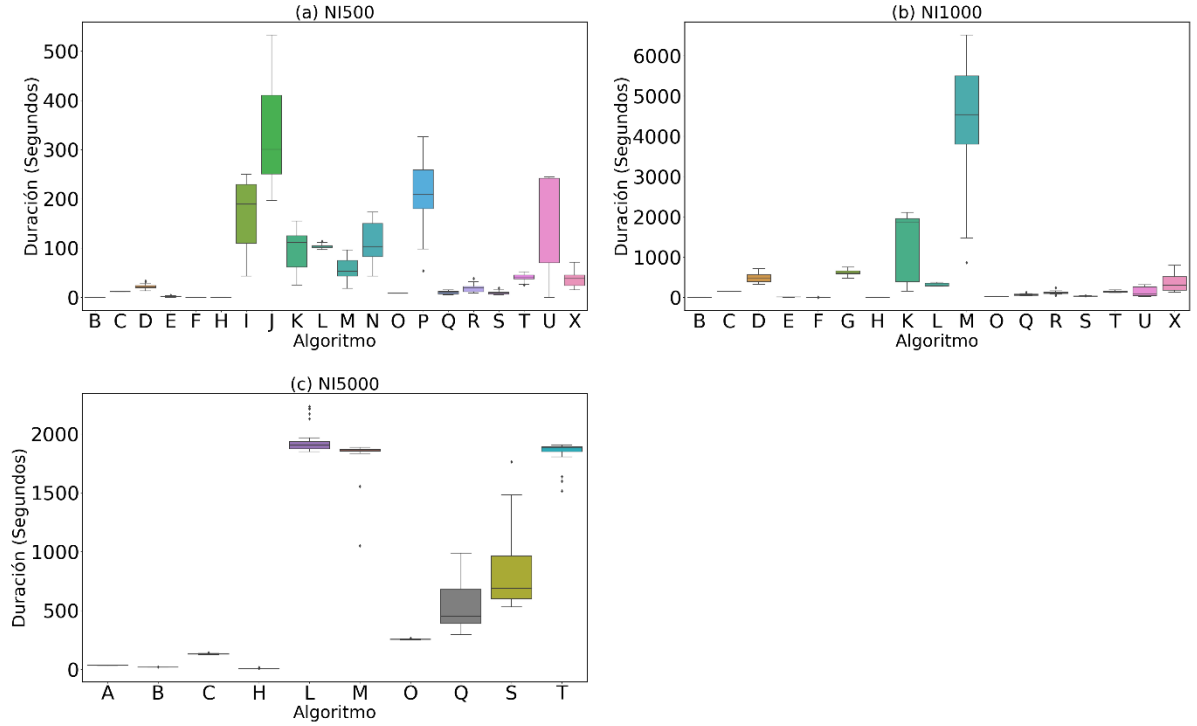

Figura 40. Tiempo de ejecución de algoritmos en cargas de trabajo NI. Los algoritmos Q y H exhiben duración corta en NI500, NI1000, y NI5000 superando claramente a las variantes de DTP_PLS_GA (I, J, K, L, N, M). En problemas NI5000, los algoritmos Q y S son más rápidos que el resto de las metaheurísticas (L, M, T).

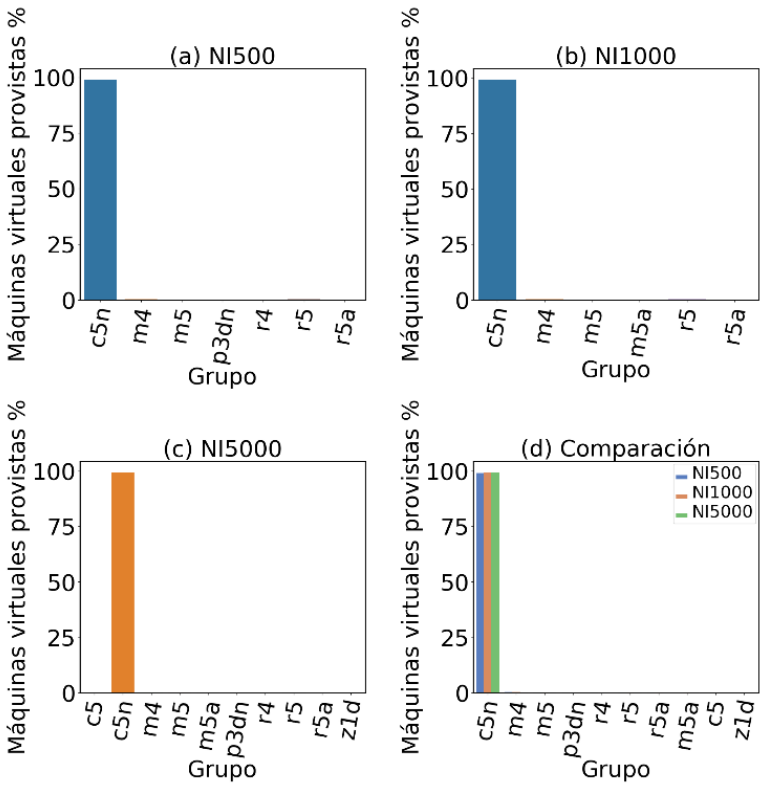

Figura 41.Elección de tipos de máquinas virtuales de los mejores algoritmos en cargas de trabajo NI. Los mejores algoritmos eligen casi exclusivamente máquinas virtuales c5n optimizadas para consumo de ancho de banda.

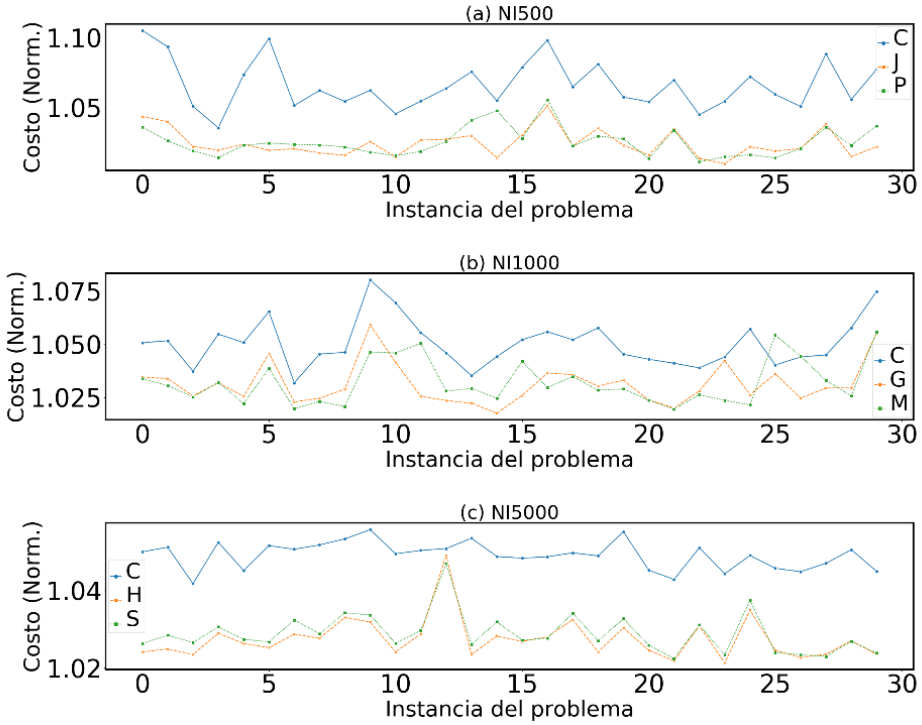

Figura 42. Comparación de costos de los mejores algoritmos en cargas de trabajo NI. En NI500 las soluciones de los algoritmos P y J difieren 0.006 en promedio. En NI1000, los algoritmos G y M convergen a soluciones con diferencia menor a 0.001 el 23% de las ocasiones, lo cual es notable debido a las diferencias conceptuales de ambos algoritmos. En NI5000, los algoritmos H y S tienen una diferencia promedio de 0.0014 mostrando una clara competencia entre sí.

4.2.4 Tareas mixtas

Los resultados exhibidos por la tabla 15 y la figura 43 muestran que el algoritmo genético DTP_PLS_GA con decodificación LTR (M) contribuye la mayor parte de las mejores soluciones en los problemas de todos los tamaños. Adicionalmente, otros algoritmos con la misma codificación (N y P) producen soluciones de primer nivel en problemas de tamaño 500, mientras que los algoritmos genéticos Q y S que utilizan codificación entera generan solamente una porción menor de soluciones en problemas de cada tamaño. Las mejores soluciones en problemas con este tipo de carga de trabajo utilizan máquinas virtuales de diferentes grupos y familias (figura 44) , y su costo es entre 4.7% y 7.9% mayor que la cota inferior producida por el algoritmo V (figura 45).

En términos de tiempo de ejecución , la figura 46 presenta evidencia que el algoritmo M requiere mas tiempo para concluir que el resto de los algoritmos en problemas de tamaño 1000 y 5000, mientras que los algoritmos Q y S requieren menos del 20% de dicha duración. Finalmente, es interesante hacer notar que el algoritmo de búsqueda local C también produce rápidamente soluciones de calidad aceptable.

Tabla 15. Rendimiento de los mejores algoritmos (MX)

	Ganador	Costo				Duración (Seg)
Id	%	Promedio	Desviación típica	Intervalo de confianza 95%		Promedio
MX500						
M	50.0000	1.0510	0.0205	1.0437	1.0583	35
N	40.0000	1.0517	0.0196	1.0447	1.0587	38
P	6.6667	1.0626	0.0179	1.0562	1.0690	134
Q	3.3333	1.0714	0.0223	1.0634	1.0793	5
K	0.0000	1.0763	0.0248	1.0674	1.0851	52
I	0.0000	1.0771	0.0259	1.0678	1.0864	75
C	0.0000	1.0892	0.0166	1.0833	1.0951	11
W	0.0000	1.8797	0.0513	1.8614	1.8981	0.0003
MX1000						
M	90.0000	1.0486	0.0128	1.0441	1.0532	2358
Q	10.0000	1.0639	0.0137	1.0590	1.0688	32
K	0.0000	1.0829	0.0179	1.0765	1.0893	228
C	0.0000	1.0918	0.0106	1.0881	1.0956	144
W	0.0000	1.8817	0.0319	1.8703	1.8931	0.0007
MX5000						
M	96.6667	1.0478	0.0064	1.0455	1.0501	1351
S	3.3333	1.0592	0.0098	1.0557	1.0627	276
Q	0.0000	1.0592	0.0082	1.0563	1.0622	298
C	0.0000	1.0763	0.0070	1.0738	1.0788	179
W	0.0000	1.8836	0.0118	1.8794	1.8879	0.0037

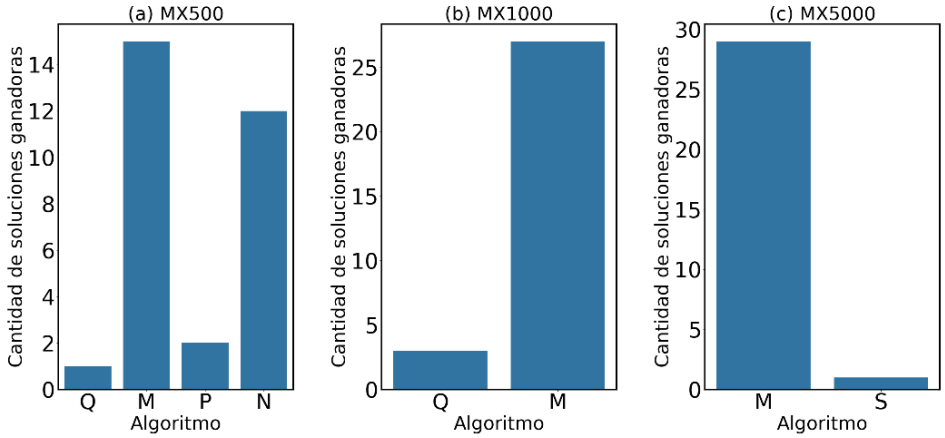

Figura 43. Algoritmos que producen las mejores soluciones para 30 experimentos con carga de trabajo MX. El algoritmo DTP_PLS_GA con decodificación LTR (M y N) producen la mayoría de las soluciones con costo mínimo. El algoritmo Q contribuye una pequeña porción de las soluciones ganadoras en MX500 y MX1000, y el algoritmo S provee una cantidad similar para MX5000.

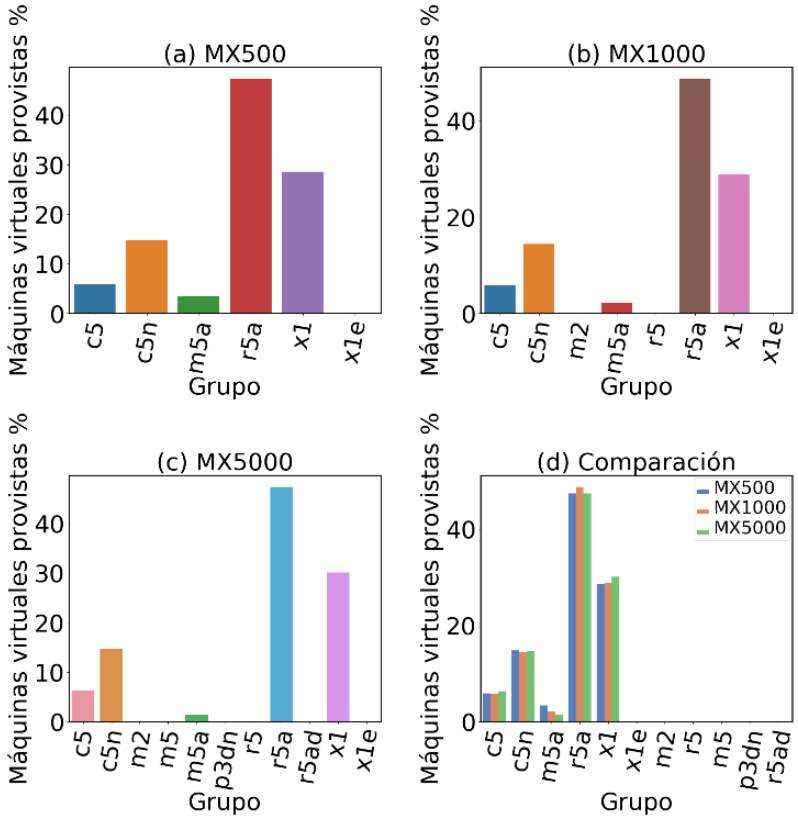

Figura 44. Elección de tipos de máquinas virtuales de los mejores algoritmos en cargas de trabajo MX. Las mejores soluciones de cada problema están constituidas por máquinas virtuales de distintas características: optimizadas para cómputo (c5), optimizadas para consumo de ancho de banda (c5n), de propósito general (m5a), y una porción predominante de optimizadas en memoria (r5a, x1).

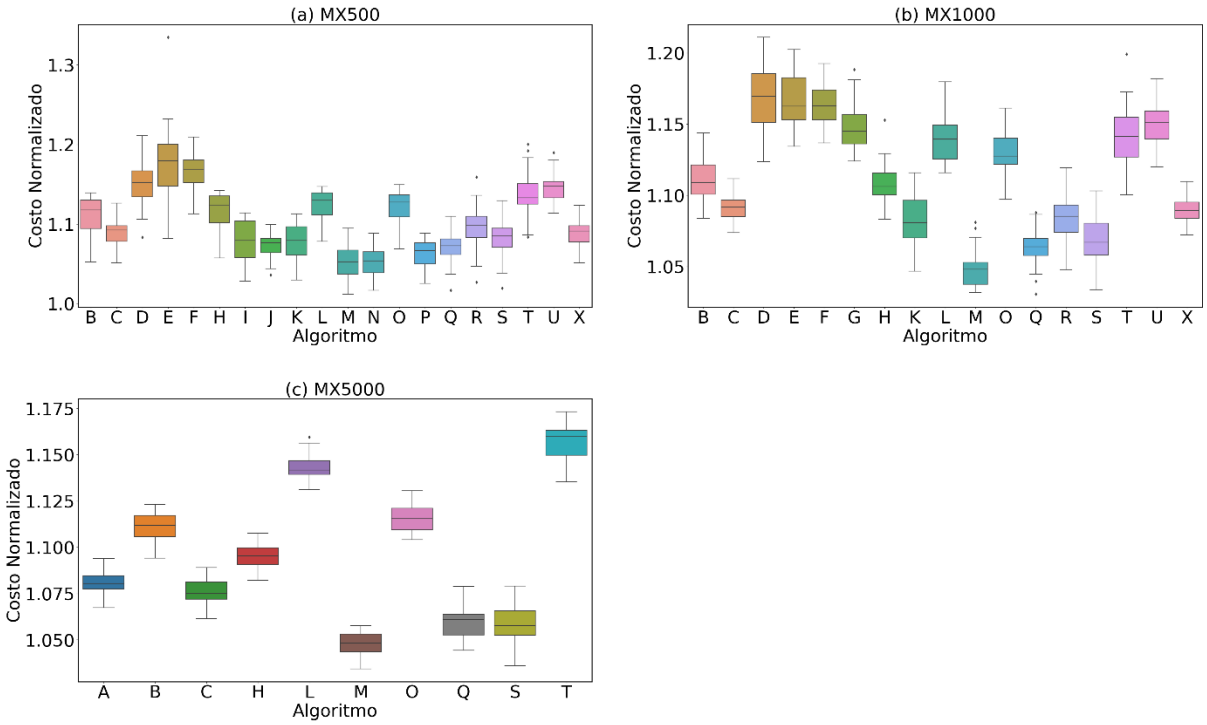

Figura 45. Costo promedio y desviación típica de algoritmos en cargas de trabajo MX. La superioridad del algoritmo M es clara en MX1000 y MX5000, mientras que en MX500 su ventaje es menos notable. El algoritmo Q se muestra como el segundo mejor para MX1000 y MX5000.

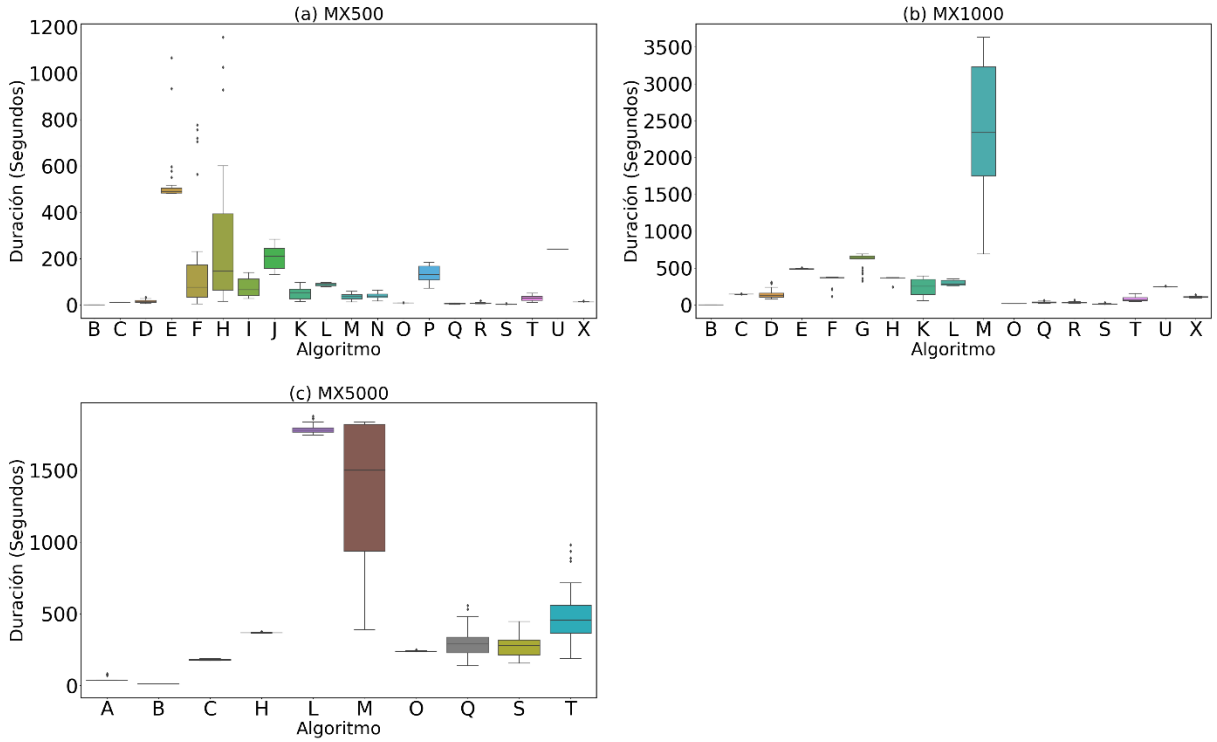

Figura 46. Tiempo de ejecución de algoritmos en cargas de trabajo MX. El algoritmo M es significativamente mas lento que el algoritmo Q y S en MX1000 y MX5000.

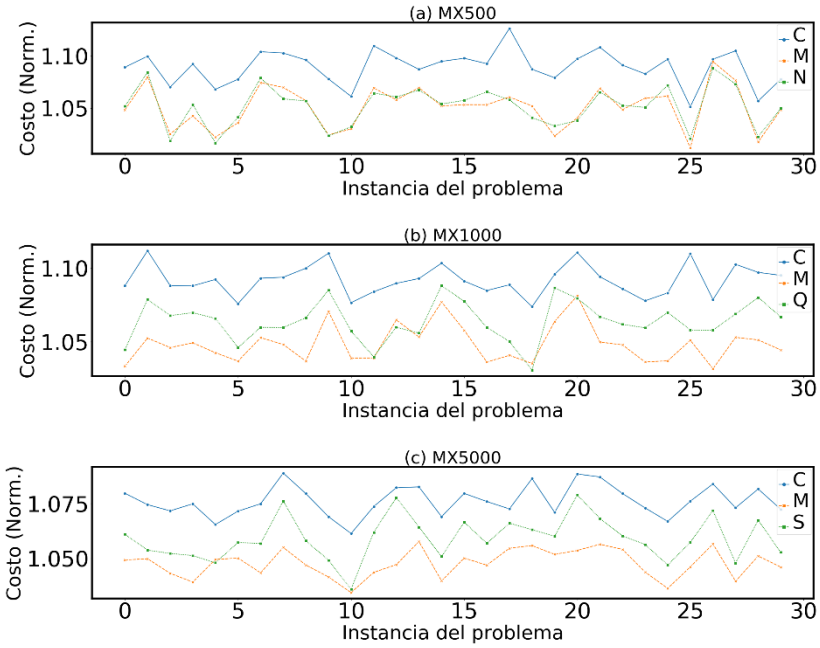

Figura 47. Comparación de costos de los mejores algoritmos en cargas de trabajo MX. En MX500, los algoritmos M y N producen soluciones cercanas entre sí, cuya diferencia de costo es en promedio 0.005. En contraste, las soluciones de MX1000 muestran una superioridad discernible del algoritmo M sobre el algoritmo Q. De manera similar, la ventaja del algoritmo S sobre el algoritmo M en MX5000, se puede apreciar claramente en la subfigura c.

4.2.5 Enfoque en tamaño de carga de trabajo

En problemas de tamaño 500, la familia de algoritmos genéticos con codificación de permutación (I, J, K, M, N, P) producen 70% de las mejores soluciones, mientras que los algoritmos genéticos con codificación entera (Q, S) aportan el 10%, y las heurísticas híbridas (F, U) contribuyen 20% (Tabla 16 y Figura 48). De manera similar, la evaluación del costo normalizado promedio en la figura 49 coloca al algoritmo J como el mejor con un valor de 1.031, seguido por los algoritmos M, N, y P con valor de 1.034. Adicionalmente, es posible observar bajo este concepto que el algoritmo Q con valor de 1.035 y desviación típica de 0.027 es preferible al algoritmo F con promedio 1.063 y desviación típica de 0.063. Por otra parte, en términos de tiempos de ejecución los algoritmos Q y S con duración promedio menor a 12 segundos superan a los algoritmos J, M y F que exhiben una duración promedio de 320, 75, y 47 segundos respectivamente (Figura 50).

Las mejores soluciones a los problemas de tamaño 1000 son producidas por algoritmos genéticos con codificación de permutación (M, K), heurísticas híbridas (H, G, U, F), y algoritmos genéticos con codificación entera (Q, S), donde cada grupo aporta 40%, 41.5%, y 18.5% respectivamente (Tabla 16 y Figura 48).

Igualmente, los algoritmos con mejor costo normalizado promedio son Q, M, y S con valores de 1.030, 1.032, y 1.034 respectivamente (Figura 49). Además, es importante notar que el algoritmo F muestra un costo normalizado promedio mayor al algoritmo de búsqueda local (C). Por último, la evaluación del tiempo de ejecución muestra que los algoritmos Q y S necesitan menos de 80 segundos y algoritmo F requiere alrededor de 150 segundos, mientras que la duración promedio del algoritmo M es mucho mayor (Figura 50).

Cuando los problemas son de tamaño 5000, las heurísticas híbridas (F, H) generan el 55% de las mejores soluciones, un algoritmo genético con codificación de permutación (M) produce el 25%, mientras que los algoritmos genéticos con codificación entera (Q, S) contribuyen con el 20% de soluciones ganadoras. Adicionalmente, la comparación del costo normalizado promedio (Tabla 16 y Figura 49) muestra que el algoritmo Q es el mejor con valor de 1.027, seguido del algoritmo S con valor de 1.028 (esta observación excluye al algoritmo F que solo fue evaluado en problemas de CI5000). Del mismo modo, la figura 50 revela que el tiempo promedio de ejecución del algoritmo a H es mejor que el resto de los algoritmos ganadores.

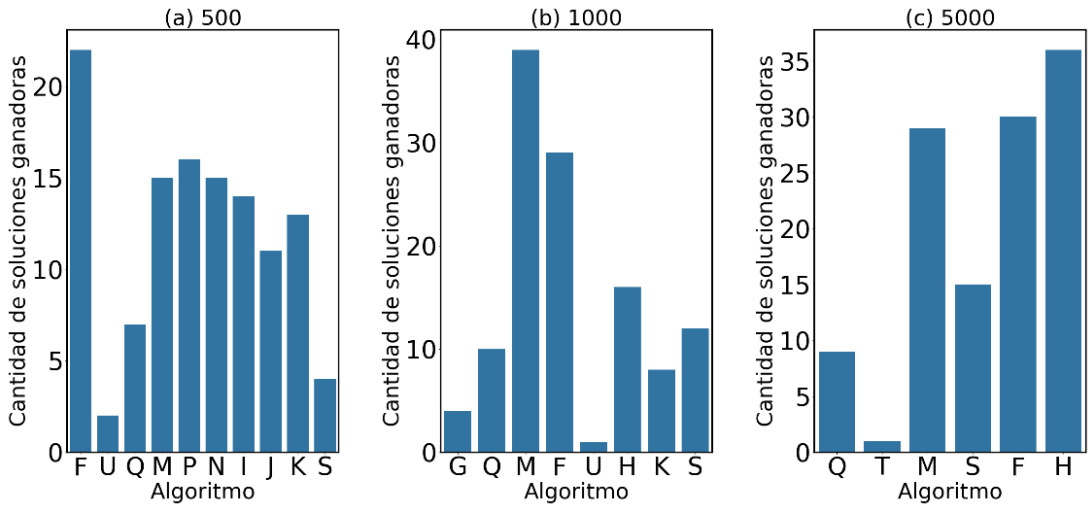

Figura 48. Algoritmos que producen las mejores soluciones para 120 experimentos con todos los tipos de cargas de trabajo. En problemas de tamaño 500, la heurística híbrida F provee la mayor parte de las soluciones de mejor calidad, seguida por el algoritmo genético de permutación P. Las mejores soluciones a los problemas de tamaño 1000 son producidas mayormente por el algoritmo genético de permutación M, seguido por la heurística híbrida F. En los problemas de tamaño 5000 los heurísticas híbridas F y H son los mejores contribuyentes, seguidos por el algoritmo M.

Tabla 16. Rendimiento de los mejores algoritmos

	Ganador	Costo				Duración (Seg)
Id	%	Promedio	Desviación típica	Intervalo de confianza 95%		Promedio
Tamaño: 500						
J	9.166667	1.031389	0.02646	1.026635	1.036143	304.8417
M	12.5	1.034116	0.01742	1.030986	1.037246	75.03333
N	12.5	1.034209	0.017273	1.031105	1.037312	130.3667
P	13.33333	1.034576	0.019465	1.031079	1.038073	213.5583
Q	5.833333	1.035877	0.027443	1.030946	1.040808	11.51667
I	11.66667	1.036581	0.030418	1.031116	1.042046	152.9083
K	10.83333	1.036767	0.030384	1.031308	1.042226	91.66667
S	3.333333	1.039175	0.031926	1.033438	1.044911	5.875
C	0	1.049658	0.031749	1.043954	1.055362	10.9
U	1.666667	1.061729	0.056978	1.051492	1.071967	157.9583
F	18.33333	1.063334	0.063981	1.051838	1.074829	47.05
W	0	1.560956	0.279068	1.510815	1.611097	0.0003
Tamaño: 1000						
Q	8.333333	1.030821	0.022617	1.026757	1.034884	75.95
M	32.5	1.032461	0.012927	1.030138	1.034784	2539.333
S	10	1.032603	0.025267	1.028063	1.037143	24.25
K	6.666667	1.040121	0.030099	1.034713	1.045529	1327.008
R	0	1.041445	0.029966	1.036061	1.046829	61.26667
H	13.33333	1.041655	0.040682	1.034345	1.048964	101.5167
X	0	1.044777	0.029425	1.03949	1.050064	162.0667
C	0	1.045845	0.030379	1.040387	1.051303	143.4417
G	3.333333	1.052922	0.056178	1.042828	1.063016	595.825
B	0	1.055228	0.039224	1.048181	1.062276	0
U	0.833333	1.058172	0.057057	1.04792	1.068423	174.1083
F	24.16667	1.060027	0.062694	1.048762	1.071291	147.5
W	0	1.5552	0.272857	1.506175	1.604225	0.0007
Tamaño: 5000						
F	25.00000	1.012308	0.000020	1.012234	1.012383	411.5666
Q	7.500000	1.027635	0.020551	1.023943	1.031328	746.4917
S	12.50000	1.028071	0.020861	1.024322	1.031819	710.4333
H	30.00000	1.041137	0.038977	1.032964	1.049327	121.44318
C	0	1.041259	0.023936	1.036959	1.04556	155.4667
M	24.16667	1.043832	0.011434	1.041778	1.045887	1720.475
B	0	1.054692	0.03921	1.047647	1.061737	15.73333
T	0.833333	1.065178	0.054333	1.055416	1.07494	1341.175
W	0	1.557458	0.273589	1.508301	1.606615	0.0037

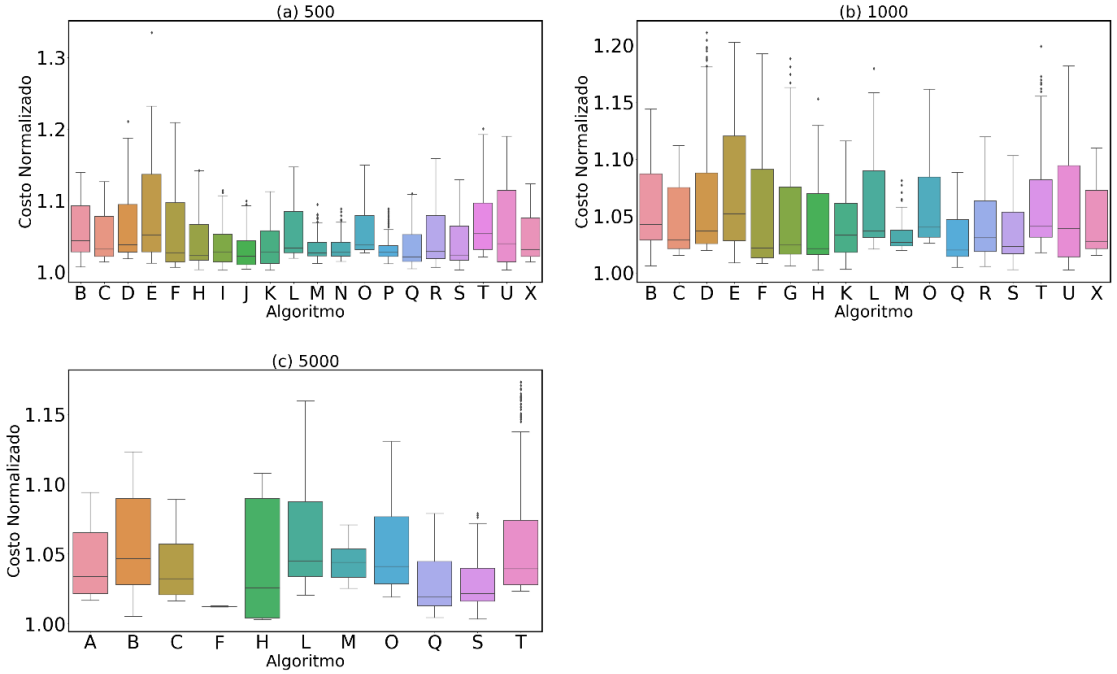

Figura 49. Costo promedio y desviación típica de algoritmos en todos los tipos de cargas de trabajo. En problemas de tamaño 500, el algoritmo J tiene el mejor costo normalizado promedio y el algoritmo M exhibe mejor comportamiento que el algoritmo F. En problemas de tamaño 1000 el algoritmo Q muestra el mejor costo normalizado promedio. En problemas de tamaño 5000, el algoritmo Q tiene un costo normalizado promedio similar a S con varianza mucho menor a la mostrada por el algoritmo H.

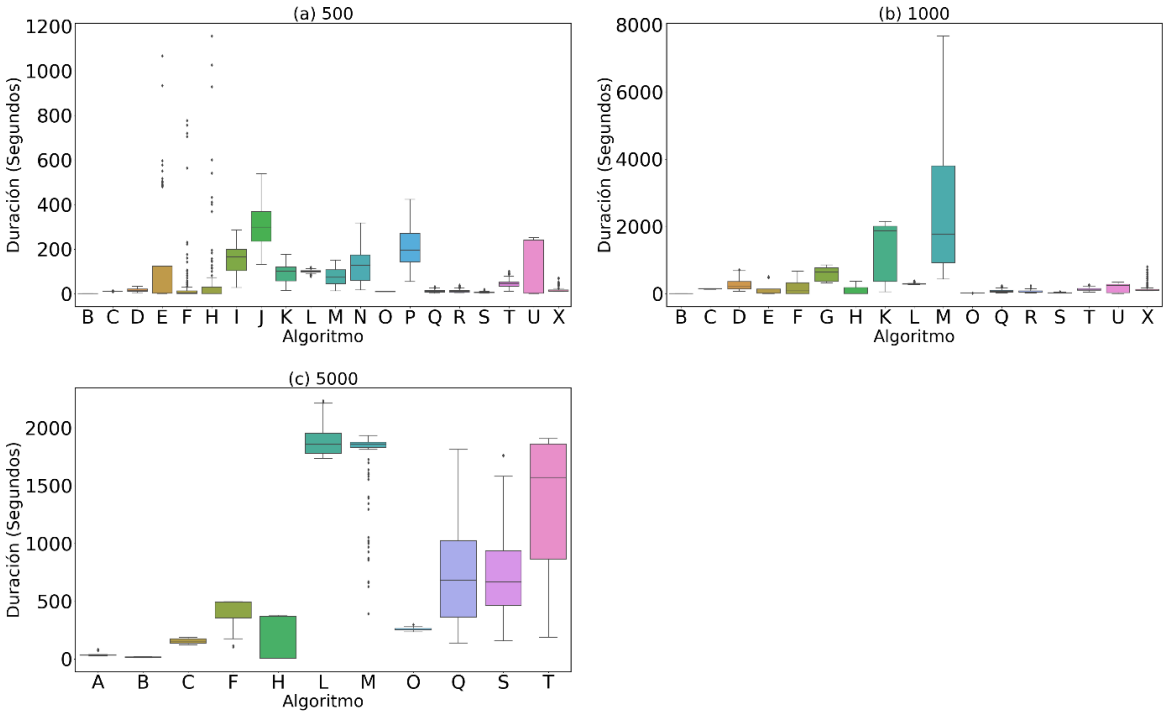

Figura 50. Tiempo de ejecución de algoritmos en todos los tipos de cargas de trabajo. En tamaño 500 y 1000, los algoritmos Q y S son mas rápidos que el algoritmo F, el cual a su vez supera al algoritmo M. En problemas de tamaño 5000, el algoritmo M necesita el doble de tiempo que los algoritmos Q y S, mientras los algoritmos F y H son mas rápidos.

4.2.6 Perfil de desempeño

El perfil de desempeño nos permite observar el porcentaje de soluciones que generó cuyo costo normalizado es menor a un valor dado. En la figura 51 podemos observar que el desempeño del algoritmo F no es apropiado, aunque el 40% de soluciones tienen costo menor a 1.02, solo el 60% de sus soluciones tiene costo menor a 1.07. De igual manera, esta figura nos permite exponer el hecho de que entre el 50% y 60% de las soluciones producidas por los algoritmos H,M,Q y S tienen costo menor a 1.03. Este mismo grupo de algoritmos generan entre el 70% y 80% de sus soluciones con costo menor a 1.05. A partir de este punto el rendimiento del algoritmo H es menor competitivo, y se puede notar que el desempeño del algoritmo comienza a superar a los demás a partir del costo 1.06 donde el porcentaje de soluciones que produce con costo menor es el 90%. Finalmente, el costo máximo de los algoritmos Q,M, y S converge en 1.11.

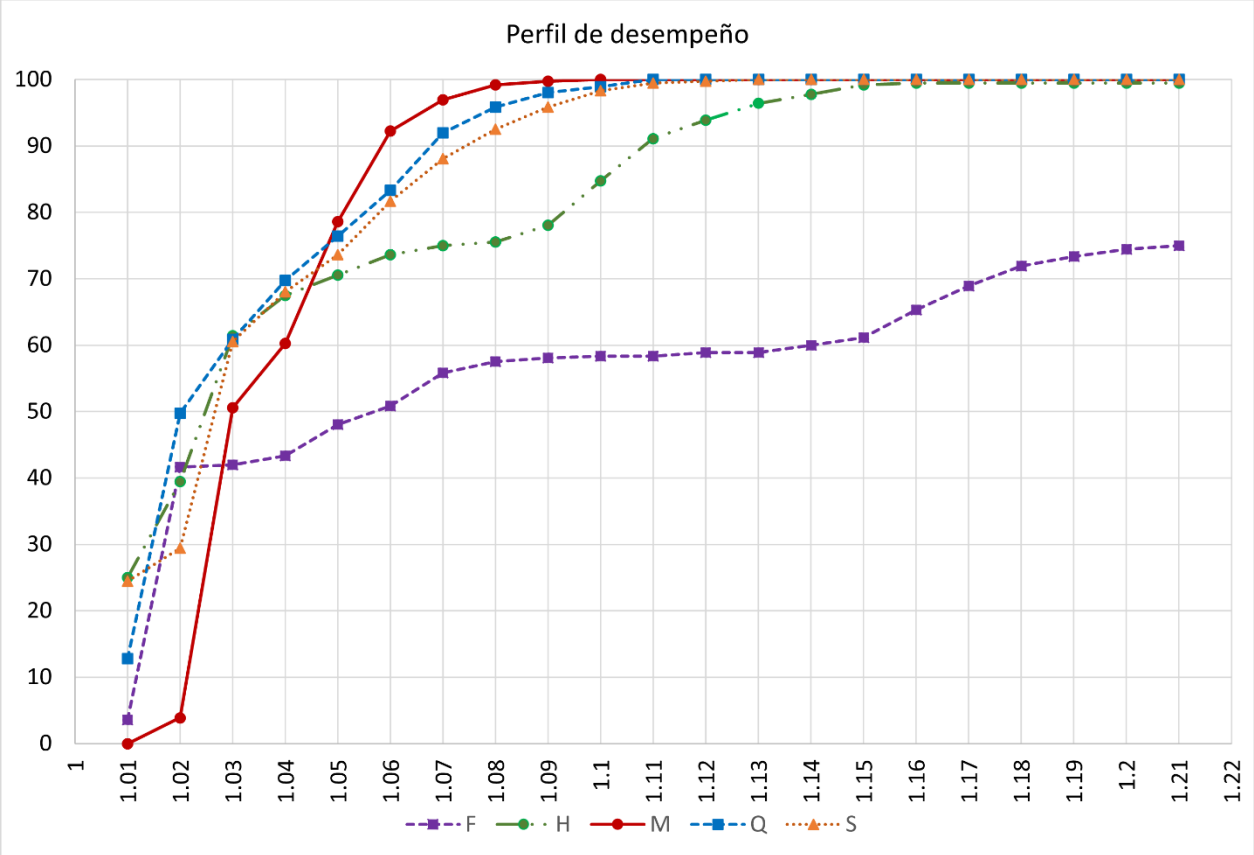

Figura 51. Perfil de desempeño de algoritmos destacados en todos los tipos de cargas de trabajo. El 90% de los soluciones del algoritmo M tienen costo normalizado menor 1.06. Los algorimos Q y S generan cerca del 90% soluciones con un costo normalizado menor a 1.07. El algoritmo H tienen un buen desempeño inicial, sin embargo el 90% de sus soluciones tienen un costo máximo de 1.11. Finalmente, el algoritmo F muestra un desempeño desfavorable.

4.2.7 Beneficio económico

Mejoras pequeñas en el costo total por hora de una solución representan ahorros significativos sobre un periodo largo de renta de las máquinas virtuales. La tabla 17 revela que la diferencia entre la mejor solución y la segunda mejor solución de un problema de tamaño grande (1000) implica ahorros en el orden de las decenas de miles de dólares estadounidenses durante un año. Claramente, la comparación entre la mejor solución y la solución de búsqueda local representan un beneficio económico de mayor impacto como lo muestra la tabla 18.

Tabla 17. Ahorro promedio del mejor algoritmo contra el segundo mejor algoritmo

Carga de trabajo	Ahorro promedio por hora	Ahorro anual (USD)
Tamaño: 500		
CI	0.47	4,114.86
MI	0.55	4,811.05
NI	0.68	5,916.32
MX	1.30	11,369.72
Tamaño: 1000		
CI	0.71	6,193.32
MI	0.82	7,164.22
NI	1.15	10,033.10
MX	7.92	69,397.42
Carga de trabajo	Ahorro promedio por hora	Ahorro anual (USD)
Tamaño: 5000		
CI	3.18	19,884.8
MI	2.85	17,840.57
NI	2.72	17,032.78
MX	30.79	192,144.99

Tabla 18. Ahorro promedio del mejor algoritmo contra búsqueda local

Carga de trabajo	Ahorro promedio por hora	Ahorro anual (USD)
Tamaño: 500		
CI	2.22	19,408.95
MI	11.14	97,611.47
NI	7.37	64,549.17
MX	11.09	97,162.71
Tamaño: 1000		
CI	4.48	39,248.60
MI	19.18	168,033.15
NI	8.11	71,077.25
MX	23.34	204,420.44
Tamaño: 5000		
CI	24.01	149,837.376
MI	93.74	584,977.74
NI	36.28	226,438.32
MX	76.52	477,524.73

Capítulo 5. Conclusiones y trabajo futuro

5.1 Conclusiones

Los resultados experimentales sugieren que la estrategia de empaquetado de máquinas virtuales y cobertura de conjuntos es apropiada para problemas con tareas computacionales donde la demanda de un tipo de recurso predomina sobre los demás, como es el caso de las tareas intensivas en cómputo (CI) y las tareas intensivas en memoria (MI). Adicionalmente, la variante con empaquetado best fit ordenado por volúmen (H) muestra un perfil de desempeño favorable donde el 90% de las soluciones generadas con este procedimiento tienen un costo normalizado menor a 1.1.

La operación de consolidación de máquinas virtuales demostró ser de gran utilidad. Esta operación es el fundamento de los algoritmos de búsqueda local que permiten encontrar una solución rápida con promedio de costo normalizado de 1.05. Tal eficiencia en tiempo de ejecución fue obtenida por medio de la utilización del índice invertido descrito en el anexo A.2 y el concepto de búsqueda limitada fundamentado en la ecuación 6. En consecuencia, la efectividad de los procedimientos de consolidación, motivaron el diseño de algoritmos genéticos que exploran la diversidad de secuencias de consolidación. Adicionalmente, los procedimientos de consolidación son utilizados exitosamente como una etapa adicional de optimización del resultado de una estrategia y como fase final del proceso de evaluación de un individuo de los algoritmos genéticos de selección de conjunto de máquinas virtuales.

En conjunto, las variantes del algoritmo genético de consolidación basado en permutación (DTP PLS GA) se establecen como el tipo de estrategia con mayor generación de soluciones mínimas con un total de 44.4%. De aquí, distinguimos el grupo de algoritmos que emplean el procedimiento de consolidación de subsecuencias (I, K, M, y N), y el grupo de algoritmos que utilizan la técnica de consolidación de segmentos en combinación con el método de consolidación del mejor par (J y P). Del primer grupo, destaca la variante de consolidación con búsqueda limitada (M) produciendo casi la totalidad de las soluciones de primera calidad en las cargas de trabajo con tareas mixtas (MX), y produciendo generalmente mejor costo normalizado promedio que las variantes con búsqueda indexada para el resto de las cargas de trabajo, además de exhibir el mejor porcentaje de soluciones con costo normalizado menor a 1.06. Por otra parte, el segundo grupo es competitivo solamente para problemas de tamaño 500 debido al elevado tiempo de ejecución requerido para evaluar la aptitud de un individuo, además exhibe un costo normalizado promedio ligeramente mayor que el algoritmo M. Por lo tanto, la evidencia indica que la elección del

algoritmo M para resolver problemas con hasta 5000 tareas computacionales mixtas es adecuada. Además, de acuerdo con los datos obtenidos en el contexto de problemas con menos de 1000 tareas, la ejecución en conjunto de las variantes competitivas de DTP PLS GA (I, J, K, M, N, y P) tiene 58% de probabilidad de generar una mejor solución que el resto de los algoritmos.

El algoritmo genético de selección de conjunto de máquinas virtuales (DTP SETUP GA en sección 3.8) y la variante basada en grupos con etapa de consolidación de subsecuencias (DTP SETUP BY GROUP GA – LTR) muestran rendimiento favorable generando soluciones mínimas en problemas de todos los tamaños. Adicionalmente, estos algoritmos exhiben perfiles de desempeño competitivos donde cerca del 90% de las soluciones producidas tiene un costo normalizado menor a 1.075. La velocidad de estos algoritmos es superior al resto de los algoritmos genéticos examinados debido a que la longitud de la codificación de los individuos es $O(M)$, lo cual permite que operaciones de cruce y mutación requieran el mismo tiempo de ejecución independientemente de la cantidad de tareas del problema $|J|$. Por otra parte, el proceso de empaquetado de máquinas virtuales utilizado en la evaluación de la función objetivo de un individuo $O(|J|)$, y la fase final de consolidación de subsecuencias recibe una entrada de longitud mucho menor a $|J|$, lo cual contribuye en la explicación del tiempo de ejecución observado. En consecuencia, estos algoritmos son una elección adecuada para problemas con gran cantidad de tareas computacionales mixtas donde se requiere obtener una solución de calidad en poco tiempo.

El análisis comparativo del costo de la mejor solución de cada instancia de problema con el costo de la segunda mejor solución, indica que una diferencia de costo de 0.5 USD por hora implica un ahorro cercano a 4500 USD anuales, y que las soluciones de problemas de tamaño 5000 pueden exhibir diferencias de entre 3 y 30 USD lo cual representa un beneficio económico de hasta 192,000 USD. Por lo tanto, con el objetivo de incrementar la probabilidad de encontrar mejores soluciones, se recomienda la ejecución en paralelo de diversos algoritmos de optimización en el tiempo disponible para tomar la decisión.

5.2 Trabajo futuro

Este trabajo no explora la reducción de consumo de ancho de banda que es producto de asignar tareas computacionales con dependencias de datos en la misma máquina virtual. De esta manera, el aprovechamiento de la estructura del grafo acíclico dirigido que modela comunicación entre tareas, ofrece el potencial de generar soluciones de menor costo y con mayor resistencia a los problemas de comunicación causados por la infraestructura de EC2.

El modelo de infraestructura computacional utilizado en este estudio supone que no hay límites en la cantidad de máquinas virtuales que pueden ser provisionadas para cada tipo. Sin embargo, los centros de datos tienen restricciones causados por la misma infraestructura y la carga computacional que aloja en determinado momento. En consecuencia, una organización que no se desea dividir su carga de trabajo en diferentes centros de datos debe considerar estas acotaciones, por lo que resulta necesario el desarrollo de un modelo de infraestructura con restricciones de provisionamiento.

Una simulación en tiempo real con alto grado de confianza debe estar libre de violaciones a la calidad del servicio. Sin embargo, existe la posibilidad de que una organización considere apropiado comprometer la calidad de servicio de un gemelo digital con el fin de economizar. Por lo tanto, una futura línea de investigación es la aplicación de métodos de optimización multiobjetivo para una variante de nuestro problema de estudio donde se pretende minimizar conjuntamente el costo y las violaciones a la calidad de servicio.

El procedimiento de consolidación del mejor par (sección 3.3.1) realiza $z = \frac{N^2}{2}$ evaluaciones cuyo resultados son independientes entre sí, exhibiendo el modelo de paralelismo SIMD (una instrucción, múltiples datos). De aquí, observamos que para $N = 5000$, la cantidad de comparaciones independientes ejecutadas son $z = 12{,}500{,}000$. De modo que la capacidad de paralelismo masivo que ofrecen las unidades de proceso de gráficos (GPU) de NVIDIA a través de la librería CUDA parece armonizar con el requerimiento computacional de nuestro procedimiento. Por lo tanto, la implementación del algoritmo 2 utilizando CUDA tiene el potencial de reducir el tiempo de ejecución consideramente, lo cual también puede impactar positivamente la velocidad de los algoritmos genéticos que lo utilizan como parte de la evaluación de la función objetivo.

Una de los principales incovenientes que presenta el algoritmo DTP PLS GA con consolidación de subsecuencias es la duración de la evaluación de la función objetivo, donde por cada individuo, el proceso de consolidación realiza $O(N^2)$ comparaciones. Dado que este algoritmo es uno de las efectivos de este trabajo, se considera relevante el desarrollo de un método que reduzca el tiempo de ejecución de la evaluación de nuevo individuo. Posiblemente, esto se pueda lograr almacenando información del conjunto de máquinas virtuales creadas por la función objetivo cada individuo, y aprovechando dicha información para evaluar a un nuevo individuo considerando las diferencias que tiene con sus padres.

De los algoritmos genéticos de selección de conjunto de máquinas virtuales, el algoritmo DTP SETUP BY GROUP GA con postprocesamiento de consolidación de subsecuencias es ligeramente superior al

algoritmo DTP SETUP GA. Esto es interesante debido a que todos los resultados del algoritmo DTP SETUP BY GROUP GA sin etapa de postprocesamiento son de menor calidad que los producidos por DTP SETUP GA. Por lo tanto, esta observación sugiere que la adición de una etapa de postprocesamiento de consolidación de subsecuencias al algoritmo DTP SETUP GA puede incrementar la calidad de las soluciones producidas.

Nuestro problema de estudio puede ser representado con un modelo de programación entera mixta que asigna las tareas computacionales en K grupos, y elige el tipo de máquina virtual para cada grupo. De este modo, el resultado de K máquinas virtuales producido por algún algoritmo, puede ser utilizado como solución inicial de este modelo con el objetivo de encontrar una solución de a lo más K máquinas virtuales con menor costo.

Literatura citada

Aldhalaan, A., Menasce, D. A. 2015. Near-Optimal Allocation of VMs from IaaS Providers by SaaS Providers. Proceedings - 2015 International Conference on Cloud and Autonomic Computing, ICCAC 2015, 228–231. doi:10.1109/ICCAC.2015.16

Alouane, N., Abouchabaka, J., Najat, R. 2018. Virtual machines online acquisition. IAENG International Journal of Computer Science, 45, 304–319.

Bhise, V. K., Mali, A. S. 2013a. Cloud resource provisioning for Amazon EC2. 2013 4th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2013. doi:10.1109/ICCCNT.2013.6726565

Bhise, V. K., Mali, A. S. 2013b. EC2 instance provisioning for cost optimization. Proceedings of the 2013 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2013, 1891–1895. doi:10.1109/ICACCI.2013.6637470

Chaisiri, S., Kaewpuang, R., Lee, B. S., Niyato, D. 2011. Cost minimization for provisioning virtual servers in amazon elastic compute cloud. IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings, 85–95. doi:10.1109/MASCOTS.2011.30

Ciavotta, M., Alge, M., Menato, S., Rovere, D., Pedrazzoli, P. 2017. A Microservice-based Middleware for the Digital Factory. Procedia Manufacturing, 11, 931–938. doi:10.1016/j.promfg.2017.07.197

Convolbo, M. W., Chou, J. 2016. Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources. Journal of Supercomputing, 72(3), 985–1012. doi:10.1007/s11227-016-1637-7

Corcoran, A. L., Wainwright, R. L. 1992. A Genetic Algorithm for Packing in Three Dimensions. En Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's, New York, NY, USA, 1992, Association for Computing Machinery, pp. 1021–1030. doi:10.1145/130069.130126

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

Dordević, B. S., Jovanović, S. P., Timčenko, V. V. 2014. Cloud Computing in Amazon and Microsoft Azure platforms: Performance and service comparison. 2014 22nd Telecommunications Forum, TELFOR 2014 - Proceedings of Papers, 931–934. doi:10.1109/TELFOR.2014.7034558

Eiben, A., Smith, J. 2003. Introduction To Evolutionary Computing. En Computing Reviews (Vol. 45). doi:10.1007/978-3-662-05094-1

He, J., Wen, Y., Huang, J., Wu, D. 2014. On the cost-qoe tradeoff for cloud-based video streaming under amazon EC2's pricing models. IEEE Transactions on Circuits and Systems for Video Technology, 24(4), 669–680. doi:10.1109/TCSVT.2013.2283430

Kritikos, K., Horn, G. 2018. IaaS Service Selection Revisited. En K. Kritikos, P. Plebani, & F. de Paoli (Eds.), Service-Oriented and Cloud Computing. Springer International Publishing: Cham. pp. 170–184.

Mao, M., Humphrey, M. 2011. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, 1–12. de http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6114435

Negri, E., Fumagalli, L., Macchi, M. 2017. A Review of the Roles of Digital Twin in CPS-based Production Systems. Procedia Manufacturing, 11(June), 939–948. doi:10.1016/j.promfg.2017.07.198

Radchenko, G., Alaasam, A., Tchernykh, A. 2019a. Micro-workflows: Kafka and kepler fusion to support digital twins of industrial processes. Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, (18), 83–88. doi:10.1109/UCC-Companion.2018.00039

Radchenko, G., Alaasam, A., Tchernykh, A. 2019b. Micro-workflows: Kafka and kepler fusion to support digital twins of industrial processes. En Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, 2019, pp. 83–88. doi:10.1109/UCC-Companion.2018.00039

Radziwon, A., Bilberg, A., Bogers, M., Madsen, E. S. 2014. The smart factory: Exploring adaptive and flexible manufacturing solutions. Procedia Engineering, 69, 1184–1190. doi:10.1016/j.proeng.2014.03.108

Soltani, S., Martin, P., Elgazzar, K. 2018. A hybrid approach to automatic IaaS service selection. Journal of Cloud Computing, 7(1). doi:10.1186/s13677-018-0113-8

Srirama, S. N., Ostovar, A. 2015. Optimal resource provisioning for scaling enterprise applications on the cloud. Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom, 2015-Febru(February), 262–271. doi:10.1109/CloudCom.2014.24

Stephanakis, I. M., Chochliouros, I. P., Caridakis, G., Kollias, S. 2013. A Particle Swarm Optimization (PSO) Model for Scheduling Nonlinear Multimedia Services in Multicommodity Fat-Tree Cloud Networks. En L. Iliadis, H. Papadopoulos, & C. Jayne (Eds.), Engineering Applications of Neural Networks. Springer Berlin Heidelberg: Berlin, Heidelberg. pp. 257–268.

Stock, T., Seliger, G. 2016. Opportunities of Sustainable Manufacturing in Industry 4.0. Procedia CIRP, 40(Icc), 536–541. doi:10.1016/j.procir.2016.01.129

Tao, F., Zhang, M. 2017. Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing. IEEE Access, 5, 20418–20427. doi:10.1109/ACCESS.2017.2756069

Tian, H., Wu, D., He, J., Xu, Y., Chen, M. 2015. On Achieving Cost-Effective Adaptive Cloud Gaming in Geo-Distributed Data Centers. IEEE Transactions on Circuits and Systems for Video Technology, 25(12), 2064–2077. doi:10.1109/TCSVT.2015.2416563

Valerio, V. Di, Cardellini, V., Presti, F. Lo. 2013. Optimal pricing and service provisioning strategies in cloud systems: A stackelberg game approach. IEEE International Conference on Cloud Computing, CLOUD, 115–122. doi:10.1109/CLOUD.2013.102

Yadav, S. L., Sohal, A. 2017. Comparative Study of Different Selection Techniques in Genetic Algorithm. International Journal of Engineering, Science and Mathematics, 6(3), 174–180. de http://www.indianjournals.com/ijor.aspx?target=ijor:ijesm&volume=7&issue=3&article=051

Zhang, Z., Cherkasova, L., Loo, B. 2014. Exploiting cloud heterogeneity for optimized cost/performance MapReduce processing. 2014. doi:10.1145/2592784.2592785

Zhou, A. C., He, B., Cheng, X., Lau, C. T. 2017. A Declarative Optimization Engine for Resource Provisioning of Scientific Workflows in Geo-Distributed Clouds. IEEE Transactions on Parallel and Distributed Systems, 28(3), 647–661. doi:10.1109/TPDS.2016.2599529

Zhou, A. C., He, B., Liu, C. 2016. Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. IEEE Transactions on Cloud Computing, 4(1), 34–48. doi:10.1109/TCC.2015.2404807

Anexos

A.1 Tipos de máquinas virtuales no dominadas de EC2

Tabla 19. Tipos de máquinas virtuales no dominadas de EC2

Tipo	Factor de tamaño	ECU	Memoria GB	Ancho de banda GBps
c4.4xlarge	32	62.0	30.0	4.96
c5.large	4	9.0	4.0	0.56
c5.xlarge	8	17.0	8.0	1.11
c5.2xlarge	16	34.0	16.0	2.22
c5.4xlarge	32	68.0	32.0	4.44
c5.9xlarge	72	141.0	72.0	10.0
c5.12xlarge	96	188.0	96.0	12.0
c5.18xlarge	144	281.0	144.0	25.0
c5.metal	192	375.0	192.0	25.0
c5.24xlarge	192	375.0	192.0	25.0
c5n.large	4	9.0	5.25	2.78
c5n.xlarge	8	17.0	10.5	5.56
c5n.2xlarge	16	34.0	21.0	11.11
c5n.4xlarge	32	68.0	42.0	22.22
c5n.9xlarge	72	141.0	96.0	50.0
c5n.18xlarge	144	281.0	192.0	100.0
i3.metal	128	208.0	512.0	25.0
m1.small	1	1.0	1.7	0.29
m2.xlarge	8	6.5	17.1	0.29
m2.2xlarge	16	13.0	34.2	0.6
m2.4xlarge	32	26.0	68.4	0.95
m3.medium	2	3.0	3.75	0.3
m3.large	4	6.5	7.5	0.69
m4.10xlarge	80	124.5	160.0	10.0
m4.16xlarge	128	188.0	256.0	25.0
m5.large	4	8.0	8.0	0.62
m5.xlarge	8	16.0	16.0	1.25
m5.2xlarge	16	31.0	32.0	2.5
m5.4xlarge	32	60.0	64.0	5.0
m5.8xlarge	64	131.0	128.0	10.0
m5.16xlarge	128	262.0	256.0	20.0
m5.24xlarge	192	345.0	384.0	25.0
m5.metal	192	345.0	384.0	25.0
m5a.large	4	8.0	8.0	0.42
m5a.xlarge	8	16.0	16.0	0.83
m5a.2xlarge	16	31.0	32.0	1.67
m5a.4xlarge	32	60.0	64.0	3.33

m5a.8xlarge	64	131.0	128.0	6.67
m5a.12xlarge	96	173.0	192.0	10.0
m5a.16xlarge	128	262.0	256.0	12.0
m5a.24xlarge	192	345.0	384.0	20.0
p3dn.24xlarge	192	345.0	768.0	100.0
r4.16xlarge	128	195.0	488.0	25.0
r5.large	4	9.0	16.0	0.62
r5.xlarge	8	19.0	32.0	1.25
r5.2xlarge	16	38.0	64.0	2.5
r5.4xlarge	32	71.0	128.0	5.0
r5.8xlarge	64	131.0	256.0	10.0
r5.16xlarge	128	262.0	512.0	20.0
r5.metal	192	347.0	768.0	25.0
r5.24xlarge	192	347.0	768.0	25.0
r5a.large	4	9.0	16.0	0.42
r5a.xlarge	8	19.0	32.0	0.83
r5a.2xlarge	16	38.0	64.0	1.67
r5a.4xlarge	32	71.0	128.0	3.33
r5a.8xlarge	64	131.0	256.0	6.67
r5a.12xlarge	96	173.0	384.0	10.0
r5a.16xlarge	128	262.0	512.0	12.0
r5a.24xlarge	192	347.0	768.0	20.0
r5ad.large	4	10.0	16.0	0.42
r5d.large	4	10.0	16.0	0.62
x1.16xlarge	128	174.5	976.0	10.0
x1.32xlarge	256	349.0	1952.0	25.0
x1e.xlarge	8	12.0	122.0	0.62
x1e.2xlarge	16	23.0	244.0	1.25
x1e.4xlarge	32	47.0	488.0	2.5
z1d.3xlarge	24	75.0	96.0	5.0
z1d.12xlarge	96	271.0	384.0	25.0
z1d.metal	96	271.0	384.0	25.0

A.2 Construcción de índice de tipo de máquina virtual de menor costo de acuerdo a demanda de recursos computacionales.

Sea $\alpha > 0$ la precisión del índice $I = (L, R)$. En este sentido, los valores de la demanda de un recurso computacional pueden ser mapeados a un múltiplo de α. Además, sea A_r el conjunto de valores distintos de capacidad ofertada de un recurso computacional $r \in \{cpu, memoria, anchoDeBanda\}$, y sea $A = A_{cpu} \times A_{memoria} \times A_{anchoDeBanda}$ todas las posibles configuraciones de tipos de máquina virtual utilizando estas capacidades. El algoritmo 21 construye un índice $L: A \mapsto T$ que define el tipo de máquina de menor costo que para cada combinación de CPU, memoria, y ancho de banda. Del mismo modo, construye una tabla auxiliar R que permita mapear un valor arbitrario de demanda de un recurso r al espacio de valores de A_r. La complejidad de este algoritmo es $O(M|A| + max\{A_{cpu} \cup A_{memoria} \cup A_{anchoDeBanda}\})$.

El Algoritmo 2 realiza una consulta arbitraria a un índice $I = (L, R)$ en $O(1)$ unidades de tiempo. El primer paso consiste en convertir los valores de la consulta en la escala definida por la presición α. Posteriormente, se utiliza la tabla R y los valores calculados en el paso anterior, para conocer las posiciones de la matriz tridimensional L donde se almanena el número del tipo de máquina virtual que satisface la demanda computacional de la consulta.

Algoritmo 21. Construcción de índice de tipo de máquina virtual de menor costo para demanda computacional

Resultado: Índice de tipo de máquina virtual de menor costo para demanda computacional $I = (L, R)$.

$Computo \coloneqq$ ordenar(distintos ($\{P_i | \ i \in \{1, \dots, M\}\}$));
$Memoria \coloneqq$ ordenar(distintos ($\{M_i | \ i \in \{1, \dots, M\}\}$));
$AnchoDeBanda \coloneqq$ ordenar(distintos ($\{B_i | \ i \in \{1, \dots, M\}\}$));
for $c \in \{1, \dots, |Computo|\}$ **do:**
  for $m \in \{1, \dots, |Memoria|\}$ **do:**
    for $b \in \{1, \dots, |AnchoDeBanda|\}$ **do:**
      $cv \coloneqq Computo_c$;
      $mv \coloneqq Memoria_m$;
      $bv \coloneqq AnchoDeBanda_b$;
      $L_{c,m,b} \coloneqq \underset{i \in \{1,\dots,M\}}{\operatorname{argmin}} \{C_i \mid cv \leq P_i, mv \leq M_i, bv \leq B_i\}$;
$mC = \max(Computo)$;
$mM = \max(Memoria)$;
$mB = \max(AnchoDeBanda)$;
$R_{cpu,v} \coloneqq \min\{r \in \{1, \dots, |Computo|\} \mid v\alpha \leq Computo_r\}$ $\quad \forall\, v \in \{0, \dots, \lfloor \frac{mC}{\alpha} \rfloor\}$;
$R_{memoria,v} \coloneqq \min\{r \in \{1, \dots, |Memory|\} \mid v\alpha \leq Memory_r\}$ $\quad \forall\, v \in \{0, \dots, \lfloor \frac{mM}{\alpha} \rfloor\}$;
$R_{anchoDeBanda,v} \coloneqq \min\{r \in \{1, \dots, |AnchoDeBanda|\} \mid v\alpha \leq AnchoDeBanda_r\}$ $\quad \forall\, v \in \{0, \dots, \lfloor \frac{mB}{\alpha} \rfloor\}$;

Algoritmo 22. Consultar tipo de máquina virtual de menor costo que satisface demanda computacional

Entrada: Índice de tipo de máquina virtual de menor costo para demanda computacional $I = (L, R)$.
Demanda de recursos computacionales $D = (Computo, Memoria, AnchoDeBanda)$.

Resultado: Número del tipo de máquina virtual de menor costo que satisfice la demanda computacional

sea $escala(x) = \left\lfloor \left\lceil x \frac{1}{\alpha} \right\rceil \right\rfloor$;

$sc := escala(Computo)$;

$sm := escala(Memoria)$;

$sb := escala(AnchoDeBanda)$;

if $sc > |R_{cpu}|$ **or** $sm > |R_{memoria}|$ **or** $sb > |R_{anchoDeBanda}|$ **then:**

    return NIL;

$c \coloneqq R_{cpu,sc}$;

$m \coloneqq R_{memoria,sm}$;

$b \coloneqq R_{anchoDeBanda,sb}$;

$i \coloneqq L_{c,m,b}$; /* $i = NIL$ si ningún tipo de máquina virtual satisface la demanda*/

return i;

A.3 Generación de tareas computacionales aleatorias

```
def GetCpuIntensiveTasks(Qty):
    RandEcuPortion = np.random.uniform(0.5,0.95,Qty);
    RandEcu = [ ]
    for e in RandEcuPortion:
        r = np.random.randint(0,len(ECU))
        RandEcu.append( ECU[r]*e  )


    RandMemory = np.random.uniform(0.03*min(Memory), 0.15*min(Memory) ,Qty)
    RandBandwidth = np.random.uniform(0.03*min(Bandwidth), 0.15*min(Bandwidth) ,Qty)

    Tasks=[]
    for i in range(Qty):
        Tasks.append( { "ECU":round(RandEcu[i],2),
                        "Bandwidth":round(RandBandwidth[i],2),
                        "Memory":round(RandMemory[i],2)}  )
    return Tasks
```

Figura 52. Código en Python para generar aleatoriamente N tareas computacionales intensivas en cómputo.

```
def GetMemoryIntensiveTasks(Qty):
    RandomEcuPortion = np.random.uniform(0.25,0.5,Qty);
    RandomMemoryPortion = np.random.uniform(0.6,1.5,Qty);
    RandEcu = [ ]
    RandMemory=[]

    MemoryOptions = [w["Memory"] for w in WorkerTypes if w["Family"]=="Memory Optimized"]
    ECUOptions = [w["ECU"] for w in WorkerTypes if w["Family"]=="Memory Optimized"]

    for i in range(Qty):
        r = np.random.randint(0,len(MemoryOptions))
        taskMemory = min( MemoryOptions[r]* RandomMemoryPortion[i] , max(MemoryOptions)*0.8)
        taskEcu = ECUOptions[ r ]* RandomEcuPortion[i]

        RandEcu.append( taskEcu )
        RandMemory.append(taskMemory  )

    RandBandwidth = np.random.uniform(0.03*min(Bandwidth), 0.15*min(Bandwidth) ,Qty)

    Tasks=[]
    for i in range(Qty):
        Tasks.append( { "ECU":round(RandEcu[i],2),
                        "Bandwidth":round(RandBandwidth[i],2),
                        "Memory":round(RandMemory[i],2)}  )
    return Tasks
```

Figura 53. Código en Python para generar aleatoriamente N tareas computacionales intensivas en memoria.

```
import random

Name="Mixed5000"
Load = [  ]
for i in range(30):
    Tasks=GetCpuIntensiveTasks(1667) + GetMemoryIntensiveTasks(1666) +GetNetworkIntensiveTasks(1667)
    random.shuffle(Tasks)
    Load.append(Tasks)
```

Figura 54. Código en Python para generar aleatoriamente N tareas computacionales mixtas.

```python
def GetNetworkIntensiveTasks(Qty):
    RandomEcuPortion = np.random.uniform(0.03,0.1,Qty);
    RandomMemoryPortion = np.random.uniform(0.03,0.1,Qty);
    RandomBandwidthPortion = np.random.uniform(0.5,1.15,Qty);

    Options = sorted(WorkerTypes,key=lambda w:w["Bandwidth"])

    Tasks=[]
    for i in range(Qty):
        r = np.random.randint( len(Options)//2.0   ,len(Options)) ## last 2 thirds
        taskBandwidth = min( Options[r]["Bandwidth"]* RandomBandwidthPortion[i] , max(Bandwidth)*0.9)

        taskEcu = Options[ r ]["ECU"]* RandomEcuPortion[i]
        taskMemory = Options[ r ]["Memory"]* RandomMemoryPortion[i]


        Tasks.append( { "ECU":round(taskEcu,2),
                        "Bandwidth":round(taskBandwidth,2),
                        "Memory":round(taskMemory,2)}  )



    return Tasks
```

Figura 55. Código en Python para generar aleatoriamente N tareas computacionales intensivas en ancho de banda.