Centro de Investigación Científica y de Educación Superior de Ensenada

Algoritmos Genéticos para un Problema de Calendarización en un Grid Computacional con Múltiples Criterios usando un Método de Agregación

TESIS

MAESTRIA EN CIENCIAS

Yair Castro García

ENSENADA B. C. MEXICO ENERO DE 2010

TESIS DEFENDIDA POR

Yair Castro García

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Andrei Tchernykh
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. José Alberto Fernández Zepeda
Miembro del Comité

Dr. Victor Hugo Yaurima Basaldúa
Miembro del Comité

Dr. Joaquín Alvarez Gallegos
Miembro del Comité

Ana Isabel Martínez García
Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales
Director de Estudios de Posgrado

29 de enero de 2010.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACION

Algoritmos Genéticos para un Problema de Calendarización en un Grid Computacional con Múltiples Criterios usando un Método de Agregación

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de
MAESTRO EN CIENCIAS

Presenta:

Yair Castro García

Ensenada, Baja California, México, enero del 2010.

RESUMEN de la tesis de **Yair Castro García,** presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS en Ciencias de la Computación. Ensenada, Baja California. Enero del 2010.

Algoritmos Genéticos para un Problema de Calendarización en un Grid Computacional con Múltiples Criterios usando un Método de Agregación

Resumen aprobado por:

Dr. Andrei Tchernykh
Director de Tesis

El presente trabajo se enfoca en el problema de calendarización de trabajos paralelos en un sistema Grid jerárquico que consta de dos niveles y en el cual se consideran cuatro criterios de optimización. En el primer nivel se realiza una asignación de los trabajos a recursos por medio de un meta-calendarizador. Los trabajos se envían a las máquinas del Grid (segundo nivel) y aplican una estrategia de calendarización local en forma independiente.

En este ambiente, uno de los mayores retos es ofrecer una calendarización que permita mayor eficiencia en el uso de los recursos y satisfacción de los usuarios. Por lo general, los criterios que ayudan a lograr tales metas suelen estar en conflicto. Para este fin, se adopta el método de agregación de criterios, una función para generar los pesos que indican la relativa importancia de cada criterio y una función escalar para normalizar dichos criterios. Se propone un algoritmo genético como una estrategia de asignación de recursos a trabajos. Se presenta un análisis experimental con distintas combinaciones de operadores genéticos.

Se concluye que es apropiado aplicar algoritmos genéticos al problema de calendarización en un Grid computacional de dos niveles con múltiples criterios usando un método de agregación porque supera el desempeño de las estrategias clásicas de calendarización.

Palabras Clave: Calendarización en Grid, Agregación de múltiples criterios, Algoritmos Genéticos.

ABSTRACT of the thesis presented by **Yair Castro García** as a partial requirement to obtain the MASTER OF SCIENCE degree in Computer Science. Ensenada, Baja California, México. Enero 2010.

Genetic Algorithms for a Scheduling Problem in a Computational Grid with Multiple Criteria using an Aggregation Method

This work focuses on the problem of scheduling parallel jobs on a Grid system, that consists of two hierarchical levels and in which we consider four optimization criteria. In the first level, making an assignment of jobs to resources using a meta-scheduler. Jobs are submitted to the Grid machines (second level) and apply a local scheduling strategy independently.

In this environment, one of the biggest challenges is to provide a schedule to allow more efficient use of resources and user satisfaction. In general, the criteria that help achieve these goals are often in conflict. To this end, adopting the method of aggregation of criteria, a function to generate weights that indicate the relative importance of each criterion and a scalar function to standardize the criteria. A genetic algorithm is proposed as a strategy for allocating resources to work. We present an experimental analysis with different combinations of genetic operators.

This thesis concludes that it is appropriate to apply genetic algorithms to the scheduling problem in a two-level computational Grid with a method that uses multiple criteria aggregation using a method because it exceeds the performance of traditional scheduling strategies.

Keywords: Scheduling in the Grid, aggregation of multiple criteria, Genetic Algorithms.

Dedicatoria

A mi mamá

Agradecimientos

A mi asesor, Dr. Andrei Tchernykh y a los miembros del comité de tesis

Dr. Carlos Alberto Brizuela Rodríguez
Dr. José Alberto Fernández Zepeda
Dr. Victor Hugo Yaurima Basaldúa
Dr. Joaquín Álvarez Gallegos

Al

Consejo Nacional de Ciencia y Tecnología

CONTENIDO

Página

CONTENIDO (continuación)

Página

LISTA DE FIGURAS

Figura **Página**

LISTA DE TABLAS

Tabla **Página**

Capítulo I

Introducción

Por naturaleza, la humanidad tiene la capacidad tanto de conocer y entender el mundo que lo rodea como la necesidad de satisfacer sus necesidades básicas y secundarias. Como consecuencia surgen nuevas áreas de conocimiento y gran parte de ellas necesitan poder de cómputo.

La Ciencias de la Computación[1] atiende dichas necesidades con ayuda del Cómputo Paralelo (realizar varias instrucciones simultáneamente) (Sinnen, 2007). En su esmerado intento por colaborar en dicha área, Garey y Graham, (1975) presentaron un arreglo de computadoras llamado clúster o arreglo de múltiples procesadores. Probablemente, no imaginaron que estaban creando los predecesores de lo que hoy se conoce como Grid computacional (Foster y Kesselman, 1999), enfoque que se expone a continuación.

I.1. Enfoque Grid

Un Grid se define como una infraestructura (construcción para prestar algún servicio o realizar determinada actividad) a gran escala (cientos o miles) de recursos (máquinas, dispositivos de almacenamiento, instrumentos especializados, software, etc.) heterogéneos (diferente hardware y/o software) distribuidos geográficamente (a lo largo y ancho del mundo), compartidos entre un conjunto dinámico (dicho conjunto puede crecer o disminuir) de individuos e instituciones para resolver problemas (situación de difícil solución), suministrando flexibilidad (capacidad para ceder en una negociación o conflicto),

[1] Las ciencias de la computación son aquellas que abarcan el estudio de las bases teóricas de la información y la computación, así como su aplicación en sistemas computacionales (*Department of Computer and Information Science*).

seguridad (proteger datos de manipulación, pérdida, destrucción o acceso por personas no autorizadas) y coordinación de recursos (colaboración de los diferentes individuos, departamentos y organizaciones para lograr una meta común) (Foster y Kesselman, 1999). Esta descripción se refiere a que instituciones como universidades, empresas, centros de investigación y otros grupos, se ponen de acuerdo para coordinar de manera lógica sus recursos de cómputo formando una comunidad que le permite a cada usuario trabajar con mayores recursos de cómputo a un costo menor, contar con un sistema tolerante a fallas y tener al alcance la capacidad de distribuir (balanceo de carga) las aplicaciones a ejecutar (poner un programa en función) de forma automática (Ramírez *et al.*, 2007).

Los recursos distribuidos geográficamente pertenecen a diferentes propietarios (organizaciones y/o instituciones); cuentan con sus propias políticas de acceso, costos y restricciones (carencias, dificultades, escasez y limitaciones). Los prestadores de servicios (propietarios de los recursos y/o administradores de los recursos) y consumidores de servicios (usuarios de los recursos) tienen distintos patrones de oferta, demanda y objetivos (finalidad hacia la cual deben dirigirse los recursos, también llamados métricas o criterios y son las reglas básicas para evaluar el resultado del proceso) (Kurowski *et al.*, 2006). Por ello, es importante contar con estrategias (acciones que se llevan a cabo para lograr un determinado fin) que administren de forma eficiente (relación entre el resultado alcanzado y los recursos utilizados) los recursos, intentando usarlos de la mejor manera posible para que beneficien las relaciones entre administradores y consumidores del sistema (Zhu *et al.*, 2004). Mientras los administradores/proveedores de recursos utilizan estrategias con el fin de utilizar los recursos en lo mejor posible (maximizar el uso de los recursos), los consumidores de recursos adoptan estrategias con la intención de resolver sus problemas en tiempo y costo menor. También se debe tomar en cuenta que los usuarios están en competencia (para recibir servicios) con otros usuarios y los propietarios de recursos están en competencia (ofrecer servicios) con sus similares (Zhu *et al.*, 2004).

El Grid surge como un nuevo paradigma (modelo, enfoque, patrón, forma de pensar) para resolver problemas en diferentes áreas: ciencia, ingeniería, medicina y

comercio, por citar algunos (*The Grid Forum*). En general, los principales beneficios que se logran con este modelo son (Foster y Kesselman, 1999): procesar aplicaciones en paralelo (simultáneamente), incrementar ganancias, reducir costos de producción y tiempos de entrega de los productos.

El Grid se clasifica con base en los problemas que pueden resolver o aplicaciones que pueden ejecutar. Enseguida se describe dicha clasificación.

I.2. Clasificación de Grid

La clasificación del Grid ayuda a entender la orientación (a qué tipo de problemas se enfoca) y el tipo de aplicaciones que se pueden ejecutar sobre él. De acuerdo a Foster y Kesselman, (1999), se han identificado cinco clases:

1. *Supercómputo distribuido:* las aplicaciones de supercómputo distribuido utilizan al Grid para agregar (conectar de manera física y lógica) recursos con la finalidad de resolver problemas que no se pueden resolver en un solo sistema. Dicha agregación de recursos puede abarcar la mayoría de las supercomputadoras (computadora con capacidad de cálculo superior a las comúnmente disponibles) que pertenecen a una compañía, ciudad o de todo el mundo. Ya se han realizado proyectos de este tipo para Cosmología (Norman *et al.*, 1996), Química (Nieplocha y Harrison, 1996) y modelos climáticos (Mechoso *et al.*, 1993). Los desafíos para esta arquitectura de Grid incluyen la necesidad de planificación (calendarización, ver §I.5) de recursos, la escalabilidad (incrementar la facilidad para aceptar mayor número de recursos) de protocolos (reglas que especifican el intercambio de datos entre sistemas de cómputo) y algoritmos (conjunto de instrucciones que permite la resolución de un problema paso a paso) para cientos o miles de nodos (también llamados máquinas y son computadoras con varios procesadores), algoritmos tolerantes a latencia (suma de retardos temporales dentro de una red de computadoras), lograr y mantener niveles altos en los resultados (desempeño) de sistemas heterogéneos (Foster y

Kesselman, 1999). Esta clasificación también se conoce como Grid computacional (*C-Grid*).

2. *Cómputo de alto rendimiento:* en esta clasificación el Grid se usa para calendarizar (§I.5) un gran número de trabajos[2] (aplicaciones) independientes con la meta de poner a trabajar ciclos de procesador no utilizados (componente lógico de un sistema de computación que interpreta y ejecuta instrucciones de programas). El resultado puede ser similar al supercómputo distribuido, pero los recursos se enfocan sobre un solo problema. Como ejemplo, el sistema Condor (The University of Wisconsin Madison, 1988) de la Universidad de Wisconsin se utiliza para administrar lotes (conjuntos o cantidades definidas) compuestos por computadoras en universidades y laboratorios alrededor del mundo (Litzkow *et al.*, 1988); los recursos se utilizan para estudiar y realizar simulaciones (experimentación) de cristal líquido, estudios en la penetración de la radiación, y el diseño de motores a diesel. Los desafíos para esta arquitectura, son similares a los de la clasificación supercómputo distribuido (Foster y Kesselman, 1999).
3. *Cómputo bajo demanda:* esta clasificación se denomina así por la cantidad de bienes o servicios que los consumidores están dispuestos a adquirir dado un nivel determinado de precios. Las aplicaciones bajo demanda utilizan las capacidades del Grid para reunir recursos a corto plazo que pueden no ser localmente convenientes o rentables (obtener menos ganancias que pérdidas). Dichos recursos pueden ser computadoras, software, repositorios de datos, sensores especializados, entre otros (Foster y Kesselman, 1999). Contrario a las aplicaciones de supercómputo distribuido, éstas son aplicaciones que a menudo se manejan por la relación costo-desempeño en lugar del desempeño absoluto. Como ejemplo, una computadora (*Computer-Enhaced Machine, MRI*) (Potter *et al.*, 1996 y Potter *et al.*, 1994) y un microscopio (*Scanning Tunneling Microscope, STM*) (Potter *et al.*, 1996 y Potter *et al.*, 1994) desarrollados en el Centro Nacional para Aplicaciones de Supercómputo, usan computadoras para lograr el procesamiento de imágenes en tiempo real

[2] Un trabajo es un programa que se ejecuta sobre una computadora, también conocido como tarea (Cormen *et al.*, 2001).

(recibiendo datos, procesándolos y devolviéndolos con la suficiente rapidez como para influir en el ambiente prácticamente en el mismo momento). El resultado mejora significativamente la habilidad para entender lo que se está viendo, y en el caso del microscopio, para dirigir el instrumento. Los desafíos en las aplicaciones bajo demanda derivan de la naturaleza dinámica de los recursos y la gran población de usuarios. Dichos desafíos incluyen planificación (calendarización, ver §I.5), administración del código (conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar un programa), configuración, tolerancias a fallos, seguridad, mecanismos de pago y localización de recursos (Foster y Kesselman, 1999).

4. *Manejo de datos en forma intensa:* la clasificación para aplicaciones con manejo de datos en forma intensa se centra en la síntesis (unificación) de información que se mantiene en repositorios geográficamente distribuidos, bibliotecas digitales y bases de datos (Foster y Kesselman, 1999). Dicho proceso de síntesis a menudo requiere de cómputo intenso y comunicación. Por ejemplo, experimentos físicos futuros de alta energía generarán terabytes de datos por día, o alrededor de un petabyte (unidad de medida de la capacidad de memoria y de dispositivos de almacenamiento informático) por año (Marzullo *et al.*, 1996). Se necesitaran solicitudes complejas para acceder a una gran fracción de datos que contengan eventos interesantes. Los colaboradores científicos que accedan a dichos datos, estarán distribuidos sobre la faz de la tierra y como consecuencia, los sistemas en los que estén localizados los datos, también estarán distribuidos. Los desafíos en aplicaciones que manejan grandes cantidades de datos, son la calendarización (§I.5) y configuración de sistemas complejos, así como un flujo (movimiento o circulación) de datos alto por medio de una jerarquía (forma de organización, generalmente por etapas) compuesta por múltiples niveles (Foster y Kesselman, 1999).
5. *Cómputo colaborativo:* esta clasificación se denomina así porque resulta de un proceso donde se involucra el trabajo de varias personas en conjunto. Las aplicaciones colaborativas se concentran principalmente en permitir y mejorar las interacciones humano-humano. Dichas aplicaciones se estructuran en términos de

un espacio virtual (sistema o interfaz que genera entornos sintéticos en tiempo real) compartido (Foster y Kesselman, 1999). Muchas aplicaciones se interesan en compartir el uso de recursos computacionales, tales como archivos de datos y simulaciones; en este caso, también tienen características de las clases descritas. Un ejemplo de dicha aplicación es el sistema BoilerMaker desarrollado en los Laboratorios Nacionales de Argonne (Diachin *et al.*, 1996). Este sistema permite a múltiples usuarios colaborar en el diseño de un sistema que controla emisiones en la industria incineradora. Los diferentes usuarios interactúan unos con otros y simultáneamente con un simulador de incineración. Los desafíos de aplicaciones colaborativas son los requerimientos de un sistema de tiempo real impuesto por las capacidades perceptivas de los humanos y una gran variedad de interacciones que pueden surgir (Foster y Kesselman, 1999).

Aunado a los desafíos que se describen para cada clasificación, todas ellas comparten problemas; descritos a continuación.

I.3. Problemas en Grid

Mientras el enfoque Grid se torna común, la calidad (característica benéfica que se espera de un servicio) en la utilización de sus recursos está más lejos porque surgen varios problemas por resolver:

1. *Múltiples capas de calendarizadores* (Kurowski *et al.*, 2006). La gestión (organización y administración) de los recursos del Grid involucran diferentes capas (el Grid puede estar organizado en niveles múltiples que obedecen a una jerarquía, ver §I.6) de calendarizadores (el término se refiere a planificar, ver §I.5). Los calendarizadores del Grid en el nivel más alto pueden tener una visión más general de los recursos, pero están muy alejados de los recursos que necesitan los usuarios para ejecutar sus trabajos. En el nivel más bajo, el sistema gestor (administrador)

controla los recursos específicos o conjunto de recursos locales (pertenecientes a una institución o a una máquina).

2. *Falta de control sobre los recursos* (Kurowski *et al.*, 2006). Los calendarizadores (planificadores) del Grid generalmente no tienen control sobre los recursos. Es decir, en repetidas ocasiones los trabajos se envían de un calendario de alto nivel a un conjunto local de recursos omitiendo (no toman en cuenta las restricciones para acceder el sistema) los permisos que el usuario posee.
3. *Uso compartido y variabilidad de recursos* (Kurowski *et al.*, 2006). En el ambiente Grid varios recursos se comparten entre varios proyectos de usuarios. El compartir tiene un alto grado de variabilidad de los recursos (pueden estar disponibles o no, sin control durante un periodo de tiempo) y es difícil predecir la disponibilidad de los recursos válidos a utilizar.
4. *Conflicto de objetivos* (Kurowski *et al.*, 2006). A menudo los proveedores de recursos y usuarios tienen diferentes objetivos (finalidad hacia la cual deben dirigirse los recursos y esfuerzos para cumplir los propósitos que se propone cumplir en un lapso de tiempo definido); optimizar (lograr el mejor resultado posible, ver §II.2.2) el desempeño de una aplicación para un costo específico hasta obtener un mejor sistema que minimice el tiempo de respuesta.

I.4. Grid jerárquico

Un modelo propuesto para calendarización en Grid consta de dos niveles (Figura 1) (Tchernykh *et al.*, 2006). En el primer nivel un meta-calendarizador (sistema de administración de los recursos, también llamado broker o sistema administrador de recursos) de Grid descubre los recursos por medio de alguna estrategia (plan para alcanzar las metas, ver §I.6.1) y los asigna a los trabajos, después, cada máquina utiliza su propia estrategia (calendarizador local, §I.6.2) para ejecutarlos. Una responsabilidad del meta-calendarizador es proveer acceso a los recursos distribuidos actuando como un mediador entre los usuarios y los recursos por medio de servicios intermedios (*middleware*), su intención es presentar los recursos del Grid al usuario como un recurso unificado

(Tchernykh *et al.*, 2006), es decir, que el usuario no perciba los diferentes componentes del Grid, arquitectura o estructura.

En la Figura 1 se ilustra un Grid computacional jerárquico de dos niveles; se observa la lista o cola conformada por n trabajos paralelos (el procesamiento de cada trabajo se lleva a cabo sobre varios procesadores simultáneamente). El meta-calendarizador toma los trabajos con base en alguna estrategia (dicha estrategia puede estar basada en el número de procesadores solicitados por cada trabajo, sus características, orden en que llegan, etc.), Posteriormente, el meta-calendarizador usa alguna estrategia de selección de recursos (también denominada como estrategia de asignación, §I.6.1) de recursos para asignarlos a los n trabajos. Para cada recurso (máquina que consta de varios procesadores) se forma una cola (lista) de trabajos en espera de ser procesados. Cada máquina utiliza alguna estrategia de calendarizador local (§I.6.2) para procesar los trabajos que le fueron asignados.

Uno de los mayores retos para este enfoque (sistemas descrito en el párrafo anterior), es ofrecer una calendarización eficiente de los trabajos sobre las máquinas (Lorpunmanee *et al.*, 2006), ya que para muchos problemas reales la calendarización de trabajos es un problema $NP - completo$ (ver Apéndice A). Para asimilar, la magnitud del reto, es necesario conocer el concepto de calendarización.

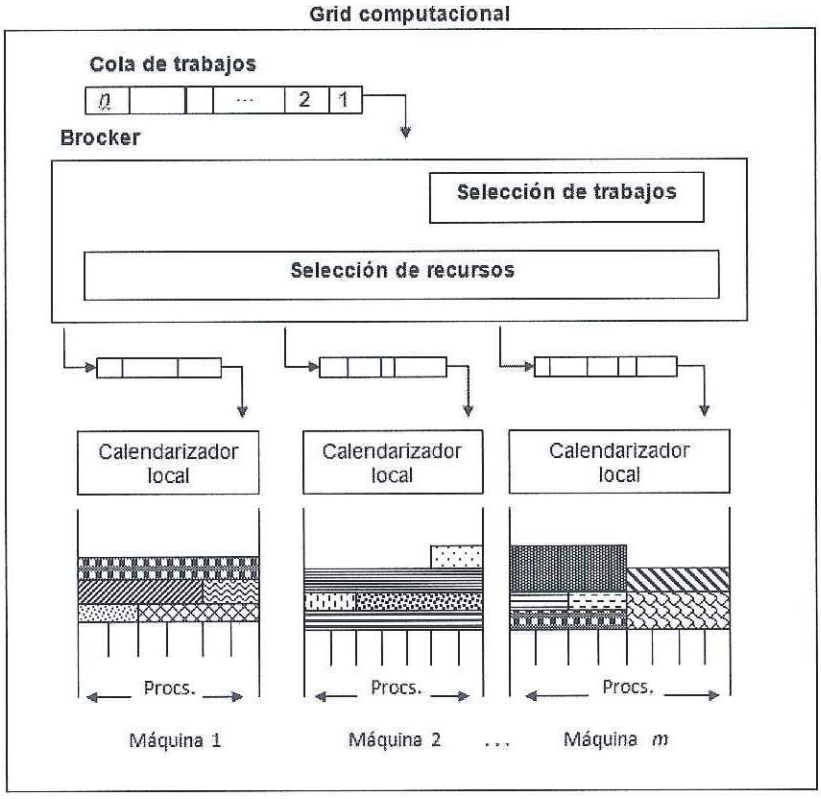


Figura 1. Grid computacional jerárquico con dos niveles.

I.5. Calendarización

Se sabe que al cambiar el orden de la lista de trabajos, decrementar su tiempo de ejecución, relajar sus relaciones de precedencia (característica de los trabajos que hace referencia a la facilidad para intercambiar lugares en la lista sin generar consecuencias graves) o cambiar el número de procesadores del sistema (en general, son características de lo que se denomina calendarización basada en lista), se presentan anomalías (irregularidades) como el aumento de la longitud del calendario (tiempo de finalización máximo de procesamiento de todos los trabajos) (Graham, 1969). Adicionalmente, los problemas de calendarización en Grid es compleja (Vadhiyar y Vadhiyar, 2002; Gehring y Streit, 2000; Hamscher *et al.*, 2000 y James *et al.*, 1999) porque involucra máquinas con diferente tamaño (cantidad distinta de procesadores por cada máquina) y distintas políticas

(estrategias) de calendarización (§I.6.1 y §I.6.2). Existen diversas definiciones de calendarización, aquí se citan las siguientes:

1) "Calendarización es pronosticar el procesamiento de un trabajo asignando recursos a trabajos y determinar sus tiempos de inicio. Los componentes de un problema de calendarización son los trabajos, las restricciones (carencias, dificultades, escasez y limitaciones), los recursos y la función objetivo (entidad que evalúa los resultados del procesamiento). Los trabajos se deben programar con la finalidad de optimizar (encontrar la mejor solución posible) un objetivo específico" (Carlier y Chrétienne, 1988).
2) "Calendarización, se refiere a la asignación de recursos limitados a trabajos en el tiempo. Es un proceso de toma de decisiones (seleccionar, bajo ciertos criterios, entre dos o más alternativas) que tiene como meta la optimización de uno o más objetivos" (Pinedo, 1995).

Generalmente, se considera que los recursos son de dos tipos (T'kindt y Billau, 2006): recursos renovables: se pueden utilizar nuevamente después de ser usados (máquina, archivo, procesador, personal, entre otros). Entre los materiales renovables se distinguen los recursos disjuntos, los cuales pueden desempeñar una operación en un solo instante y los recursos acumulativos, que procesan un número limitado de operaciones simultáneamente. Los recursos no renovables desaparecen inmediatamente después de utilizarse (dinero, material crudo, entre otros).

Un resultado en calendarización es la cota de competitividad (capacidad de un algoritmo para generar una solución con el menor costo) para trabajos secuenciales (procesamiento en el que cada trabajo precede a otro y sigue a otro, sin que nunca dos de ellos sean simultáneos) y calendarización determinística (cuando todas los trabajos están disponibles desde el inicio de la calendarización) (Graham, 1966):

$$\frac{C_{max}(A)}{C^*_{max}} \leq 2 - \frac{1}{m} \qquad (1)$$

donde $C_{max}(A)$[3] es la longitud del calendario generado por un algoritmo basado en lista, C^*_{max} es la longitud del calendario óptimo (el mejor de todos) y m es el número de procesadores paralelos. Esta cota de desempeño también se aplica para trabajos paralelos y calendarización determinística (Garey y Graham, 1975).

I.6. Calendarización en Grid de dos niveles

La calendarización se torna compleja al incluir múltiples máquinas paralelas (máquinas con varios procesadores capaz de ejecutar trabajos simultáneamente y/o procesar trabajos en varios procesadores) en el segundo nivel de un Grid. De acuerdo a Garey y Johnson, (1979) este problema pertenece a la clase de problemas *NP*-difícil (Apéndice A). La cota de competitividad (§I.5) para algoritmos de calendarización basados en lista no se puede aplicar a un Grid. Lo mismo sucede para los algoritmos de calendarización basados en lista porque no garantizan una cota de competitividad constante (Schwiegelshohn *et al.*, 2008). Esto se debe a la complejidad para balancear la carga de trabajo (distribuir los trabajos uniformemente en las máquinas), provocando que los trabajos paralelos grandes esperen ser procesados. No existe un algoritmo de tiempo polinomial (ver Apéndice A) que siempre forme calendarios que generen un cociente de competitividad menor que 2 para el problema de calendarización en Grid con máquinas paralelas al tratar de reducir el tiempo de ejecución máximo de los trabajos, a menos que $P = NP$ (Schwiegelshohn *et al.*, 2008) (ver Apéndice A).

El Grid jerárquico de dos niveles utiliza estrategias en ambos niveles para calendarizar los trabajos. Enseguida se listan las estrategias de asignación de recursos a los trabajos que corresponden al primer nivel.

---

[3] $\frac{C_{max}(A)}{C^*_{max}}$: es el máximo valor del cociente de un calendario ($C_{max}(A)$) con el calendario óptimo teórico (C^*_{max}), donde el óptimo teórico es el máximo de dos posibilidades, ya sea del tiempo de procesamiento más grande de un trabajo o el cociente de la suma de todos los tiempos de procesamiento de los trabajos con el número total de procesadores (Ramírez *et al.*, 2007).

I.6.1. Estrategias de asignación

Como se ha mencionado, las estrategias de asignación son las técnicas, métodos o conjunto de acciones que asignan recursos a los trabajos propuestos por los usuarios al Grid. Las estrategias de asignación están agrupadas por la manera en que trabajan. Las estrategias que no necesitan un calendario previo basan su decisión en los parámetros de los trabajos. Para las estrategias que requieren un calendario previo se efectúan varios pasos; un calendarizador envía el trabajo que se va a asignar a todas las máquinas y cada una de ellas construye un calendario tomando en cuenta el trabajo a calendarizar junto con los que ya tenga asignados. Cada máquina regresa la información generada al meta-calendarizador y éste toma la decisión de asignarlo a la máquina con base en la estrategia que está aplicando. Las estrategias que no necesitan crear un calendario previo son:

1. *Random* (Madrigal *et al.*, 2006): es la estrategia que se rige bajo un criterio simple, selecciona la máquina de manera aleatoria.
2. *Mínima carga por procesador* (*Min Load per-proc, Min_LP*) (Madrigal *et al.*, 2006): selecciona la máquina con la mínima carga por procesador (número de trabajos sobre el número de procesadores en la máquina).

$$min(n_i/m_i), \quad i = 1,2, \dots, M_m \tag{2}$$

donde n_i es el número de trabajos asignados a la máquina i y m_i es el tamaño de la misma máquina y M_m denota m máquinas que componen el sistema.

3. *Mínima carga paralela* (*Min Parallel Load, Min_PL*) (Madrigal *et al.*, 2006): selecciona la máquina con la mínima carga paralela por procesador (suma de los tamaños de los trabajos sobre el número de procesadores en la máquina).

$$min\left(\frac{1}{m_i}\sum_{j=1}^{n} size_j\right), i = 1,2, \dots, M_m \tag{3}$$

donde $size_j$ es el número de procesadores solicitados por el trabajo j, m_i denota el número de procesadores en la máquina i, M_m indica el número total de máquinas en el sistema.

4. *Menor cota inferior de tiempo de finalización* (*Min Lower Bound, Min_LB*) (Madrigal *et al.*, 2006): selecciona la máquina con la menor cota inferior de tiempo de terminación de los trabajos. En lugar del tiempo real de ejecución de un trabajo que no está disponible, se usa el valor proporcionado por el usuario con la que somete el trabajo al sistema.

$$min\left(\frac{1}{m_i}\sum_{j=1}^{n} w_j\right),\ i = 1, 2, \ldots, M_m \tag{4}$$

donde $w_j = size_j \cdot p_j$, es el trabajo compuesto por el tiempo de ejecución (p_j) sobre $size_j$ procesadores y M_m indica el número total de máquinas en el sistema.

Las estrategias que sí requieren datos de un calendario y que por lo tanto necesitan crearlo anticipadamente son:

5. *Menor tiempo de finalización* (*Min Completion Time, Min_CT*) (Madrigal *et al.*, 2006): se elige la máquina con el menor tiempo de terminación estimado de todos sus trabajos asignados, tomando en cuenta el trabajo por asignar.

$$min\left(C_{max}^{i}\right),\qquad i = 1, 2, \ldots, M_m \tag{5}$$

donde C_{max}^{i} es la longitud del calendario en la máquina i, M_m indica el número total de máquinas en el sistema.

6. *Menor tiempo de espera* (*Min Waiting Time, Min_WT*) (Madrigal *et al.*, 2006): estrategia que selecciona la máquina con el promedio menor de tiempo de espera de los trabajos asignados, incluyendo el trabajo por asignar.

$$min\left(\sum_{j=1}^{n}\frac{C_j^i - r_j - p_j}{n_i}\right),\ i = 1, 2, \ldots, M_m \tag{6}$$

donde C_j^i es el instante de finalización del trabajo j en la máquina i, r_j es el instante a partir del cual se encuentra disponible el trabajo j y p_j es su tiempo de ejecución, n_i es el número de trabajos en la máquina i, M_m indica el número total de máquinas en el sistema.

7. *Menor utilización* (*Min Utilization, Min_U*) (Madrigal *et al.*, 2006): elige la máquina con la utilización mínima.

$$min(R_i/C_i \cdot m_i), \qquad i = 1, 2, \ldots, M_m \tag{7}$$

donde R_i son los recursos consumidos en la máquina i, C_i denota la longitud del calendario en la máquina i, m_i indica el número de procesadores en la máquina i, M_m indica el número total de máquinas en el sistema.

8. *Menor tiempo de inicio* (*Min Start Time, Min_ST*) (Madrigal *et al.*, 2006): selecciona la máquina que proporciona el menor tiempo de inicio de ejecución para el trabajo que se va a asignar.

$$min\left(C_j^i - r_j\right), \qquad i = 1, 2, \ldots, M_m \tag{8}$$

donde C_j^i es el instante de finalización del trabajo j en la máquina i, r_j el instante a partir del cual se encuentra disponible el trabajo j y M_m indica el número de máquinas disponibles en el sistema.

9. *Menor tiempo de permanencia en el sistema* (*Min Turnaround, Min_TA*) (Madrigal *et al.*, 2006): selecciona la máquina que tiene el promedio menor de tiempo de permanencia en el sistema de los trabajos asignados incluyendo el trabajo por asignar.

$$min\left(\frac{1}{n_i}\sum_{j=1}^{n} C_j^i - r_j\right), \quad i = 1, 2, \ldots, M_m \tag{9}$$

donde C_j^i es el instante de finalización del trabajo j en la máquina i, r_j el instante a partir del cual se encuentra disponible el trabajo j y M_m indica el número de máquinas disponibles en el sistema.

Para efectos del presente trabajo, únicamente se aplican las siguientes estrategias (§IV.3): *Min_CT*, *Min_ST*, *Min_ST* y *Min_WT*. En el segundo nivel del Grid se emplea alguna estrategia de calendarización local. A continuación se describen algunas de ellas.

I.6.2. Estrategias de calendarización local

Las estrategias de calendarización local, son las técnicas que asignan procesadores a los trabajos para que se ejecuten. Cada estrategia tiene la finalidad de encontrar un calendario mejor de acuerdo al criterio (objetivo) que se desee mejorar. Se ha demostrado

que el rendimiento general del Grid se ve afectado por la elección de la estrategia de calendarización local (Shan *et al.*, 2003). Algunas estrategias de calendarización local son:

1. *Primero en llegar-primero en atenderse* (*First-Come-First-Serve, FCFS*): de todas las estrategias, es la que se considera más simple por la manera en que trabaja. El calendarizador comienza a asignar procesadores a los trabajos de acuerdo al orden en que se encuentran en la cola local (lista de trabajos en espera de procesadores libres). Si no hay suficientes recursos disponibles, el calendarizador espera hasta que el trabajo se pueda asignar y procesar. Mientras, los demás trabajos están detenidos en cola.
2. *Primero el más grande/chico* (*Largest/Smallest-Size-First, LSF*): estrategia en la que los trabajos de la cola local se ordenan por tamaños de manera ascendente o descendente para posteriormente calendarizarse con *FCFS*. Es importante notar que se puede aplicar únicamente cuando se tiene a disposición un conjunto de trabajos para ejecutarse inmediatamente (lote de trabajos, también conocido como *batch*).
3. *Primero el consumidor de recursos más grande/chico* (*Largest/Smallest Resource Consumed First, L/S-RCF*): con dicha estrategia, los trabajos en la cola local se ordenan de acuerdo a la cantidad de recursos consumidos de forma decreciente o creciente y posteriormente se calendarizan con *FCFS*. Al igual que *LSF* se aplica solamente cuando está disponible un lote de trabajos.
4. *Backfilling-EASY-FirstFit* (*Backfilling EASY First Fit, BEFF, EASY*: *Extensible Argonne Scheduling sYstem* (Etsion y Tsafrir, 2005)): es una mejora del algoritmo *FCFS* y ha mostrado ser una buena opción para mejorar el rendimiento tanto para el usuario como para el sistema (Wang *et al.*, 2006; Hamscher *et al.*, 2000 y Ramírez *et al.*, 2007). Dicha mejora se logra al permitir que algunos trabajos puedan iniciar su ejecución antes del tiempo que les correspondería según su posición en la cola de espera. Realiza una búsqueda en la cola de trabajos disponibles para ejecución inmediata cuando el trabajo de la cabeza de la cola no puede iniciar su ejecución debido a que no existen suficientes recurso para él inmediatamente. Antes de realizar la búsqueda de trabajos más pequeños, se calcula el tiempo en que habrá

recursos suficientes para el trabajo en la cabeza de la cola. La única limitante para iniciar los otros trabajos es que no retarden el inicio de la ejecución del trabajo que se encuentra al inicio de la cola (Lifka, 1998).

En el presente trabajo se aplica únicamente la estrategia de calendarización local *Backfilling-EASY-FirstFit*.

I.7. Planteamiento del problema

En general, los problemas de calendarización en Grid se pueden ver, como problemas de asignación de recursos que consiste en encontrar una calendarización sub-óptima (que se encuentre cerca de la mejor solución) que mejore algún criterio (Ramírez *et al.*, 2007), donde calendarización se refiere a la asignación en el tiempo de recursos limitados a trabajos. Los trabajos tienen sus propias características: número de procesadores requeridos, tiempo de llegada al sistema, tiempo de procesamiento estimado (propuesto por el usuario), tiempo de procesamiento real (tiempo en que el trabajo se termina de procesar realmente), etc.; ello provoca que por cada permutación (posibles ordenaciones de los trabajos) se generen distintos calendarios. Por otro lado, puede ser de interés que los trabajos terminen de procesarse lo más rápido posible desde el primero hasta el último, es decir, se intenta reducir el tiempo máximo de finalización de todos los trabajos que se procesan sobre el sistema (§II.1); también, que cada uno de los trabajos permanezcan el menor tiempo posible en el sistema, es decir, reducir (minimizar) el tiempo promedio de permanencia de los trabajos (§II.1), y/o que el sistema responda al usuario lo más pronto posible acerca del trabajo propuesto, es decir, minimizar el cociente de respuesta promedio (§II.1). En general, el minimizar el tiempo de finalización máximo de todos los trabajos procesados sobre el sistema no implica minimizar el tiempo promedio de permanencia de los trabajos (§II.2). De hecho, cuando se tienen dos o más objetivos como éstos que se encuentran en conflicto (§II.2), y es importante que todos se optimicen al mismo tiempo (que se encuentren buenas soluciones para los criterios), entonces el problema se vuelve un problema múlti-criterio (§II.2).

Adicionalmente, se deben tomar en cuenta algunas suposiciones y características: en la administración de los recursos se involucran diferentes grupos de interés, tales como usuarios, proveedores de recursos y administradores. Los grupos interesados pueden tener diferentes restricciones, criterios y preferencias (inclinación favorable). En general, se distinguen dos formas de conocer las preferencias (Kurowski *et al.*, 2006): (1) los grupos interesados proporcionan explícitamente las preferencias, es decir, cada parte interesada expresa la importancia de cada criterio o la importancia relativa entre los criterios (importancia de un criterio con respecto a otro), lo cual se puede llevar a cabo definiendo explícitamente la importancia de los criterios o por el orden en que se encuentran (ordenándolos del menos al más importante). (2) las preferencias de los grupos interesados se descubren con base en decisiones previas, es decir, por los patrones que se conocen basados en su historial (eventos anteriores que sirven como indicio).

También se deben considerar las diversas configuraciones de máquinas:

1. *Máquinas idénticas* (P): una operación O_{ij} (la operación j del trabajo J_i) tiene el mismo tiempo de procesamiento en todas las máquinas M_k, donde k denota el índice de la máquina M.
2. *Máquinas uniformes* (Q): el tiempo de procesamiento de una operación O_{ij} en la máquina M_k es igual a $p_{ijk} = q_{ij}/v_k$, donde q_{ij} es un número de componentes en la operación O_{ij} a ser procesada, y v_k es el número de componentes que la máquina M_k puede procesar por unidad de tiempo.
3. *Máquinas no relacionadas o independientes* (R): el tiempo de procesamiento de la operación O_{ij} en la máquina M_k es igual a p_{ijk} y es un dato del problema. No se conoce la asignación de O_{ij}.

En el enfoque con múltiples criterios aquí tratado se hacen las siguientes suposiciones: el meta-calendarizador del Grid es el punto principal de decisión en el proceso de calendarización. Este toma en cuenta los requisitos (necesidades, condiciones,

exigencias) y preferencias de todos los grupos interesados (usuarios, proveedores de recursos y administradores del Grid) y asigna recursos a los trabajos. Los grupos interesados expresan sus requisitos y preferencias considerando varios parámetros y criterios de los procesos de calendarización. También expresan restricciones fuertes (*hard constraints*), es decir, requisitos que deben cumplirse antes de que el meta-calendariador del Grid ponga en marcha los procesos de asignación. En la práctica, las restricciones fuertes determinan el conjunto de soluciones apropiadas (Kurowski *et al.*, 2008). Las preferencias de los grupos interesados se expresan por medio de restricciones flexibles (*soft constraints*) (por ejemplo, calidad de servicio), las cuales describen las preferencias de los criterios. Las restricciones fuertes se deben cumplir de manera estricta, en cambio, las restricciones flexibles pueden no cumplirse en su totalidad, sin embargo, se deben considerar para mejorar la calendarización porque dirigen el proceso para seleccionar las mejores soluciones (Kurowski *et al.*, 2008). Después de asignar los recursos a los trabajos, los recursos aplican su estrategia local que permite asignar procesadores a los trabajos de las colas locales. Por naturaleza, en Grid existen conflictos de interés, límite de tiempo para procesar los trabajos, límite de tiempo para tomar decisiones, entre otros. Aplicar los algoritmos que trabajan con frentes de soluciones (generan soluciones múltiples y las organizan con base en su comparación) suelen ser costosos en tiempo, operaciones y espacio en memoria (Coello *et al.*, 2002). Otra manera de resolver los problemas con las características que se mencionan, es aplicar la metodología de agregación (§III.2.1) de objetivos (toma varios criterios y los convierte en uno sólo por medio de un operador) para obtener una solución que quizás no es la óptima, pero si aceptable (Coello *et al.*, 2002).

I.8. Metodología

En Ramírez *et al.*, (2007) se presenta una comparación de varias estrategias de calendarización considerando el caso determinístico. Una estrategia se basa en asignar trabajos a la máquina que ofrece el menor tiempo de finalización (*Min_CT*, §I.6.1), la otra estrategia asigna los trabajos a la máquina que ofrece el mejor tiempo de inicio (*Min_ST*, §I.6.1), estadísticamente son las estrategias que presentan los mejores resultados. El estudio

se enfoca principalmente en estimar los tiempos de ejecución basándose en los tiempos solicitados por el usuario, su intención es mejorar los resultados. Un punto importante es que no se presenta un análisis de la carga de trabajo, por lo que los resultados no están completamente sustentados. Por otra parte, las estrategias que aplican se enfocan en encontrar la solución para un criterio en especial, es decir, no abordan el problema con múltiples criterios (§II.2). Es de esperarse que los trabajos que contemplan el problema de calendarización de trabajos en Grid se enfoquen en generar buenas soluciones para un criterio en específico ya que las funciones de muchos sistemas actuales que administran y calendarizan los sistemas Grid actuales lo hacen. Por ejemplo, los sistemas Legion, Condor, AppLeS PST, Net-Solve, PUNCH, XtremeWeb, por mencionar algunos, adoptan una estrategia convencional, donde un calendarizador selecciona los trabajos para ser ejecutados con base en funciones de costo que controlan los sistemas centralizados. Estos calendarizadores tienden a optimizar un solo objetivo: procesar la mayor cantidad de trabajos por unidad de tiempo, maximizar la utilización de los recursos o terminar de procesar los trabajos lo más pronto posible, y no se ocupan en mejorar las estrategias de calendarización cuando se contemplan varios objetivos simultáneamente (Zhu *et al.*, 2004). Tal hecho no es casualidad ya que Garey y Graham, (1975) expresan que considerando únicamente el tiempo máximo de finalización de los trabajos (longitud del calendario, C_{max} o *Makespan*), la calendarización posee problemas de optimización complejos y pertenecen a la clase $NP-difícil$ ($NP-hard$, ver Apéndice A). Sin embargo, la comunidad científica realiza varios esfuerzos para aplicar técnicas de cómputo evolutivo (§III.1) en afán por resolver tales problemas. De acuerdo a Esquivel *et al.*, (2002) los algoritmos evolutivos (*Evolutionary Algorithms, EA*) engloban una serie de técnicas inspiradas biológicamente y suelen ser una herramienta eficiente para encarar los problemas de calendarización. Los *EA* parecen ser apropiados (se pueden aplicar) para resolver Problemas Multi-Objetivo (*Multi-Objective Problems, MOPs,* (Chipperfield y Fleming, 1996) porque pueden tratar con un conjunto de posibles soluciones simultáneamente. Ello, permite encontrar varias soluciones al problema en una sola ejecución del algoritmo, en lugar de realizar una serie de ejecuciones separadas como en el caso de las técnicas matemáticas tradicionales (Coello, 1999). Los algoritmos genéticos (§III.6) forman parte de

los algoritmos evolutivos y han mostrado tener buen desempeño al momento de optimizar (§II.2.2) problemas de calendarización con varios criterios (Coello *et al.*, 2002; Eiben y Smith, 2003); en cambio, las técnicas existentes para calendarización que se aplican a Grid, están orientadas y restringidas a optimizar un objetivo en específico (Ramírez *et al.*, 2007).

La idea del presente trabajo es aplicar algoritmos genéticos (§III.6) como estrategia de asignación (§III.6.1) para encontrar soluciones (calendarios) considerando múltiples criterios usando un método de agregación (§III.2.1) y que las soluciones superen a estrategias de asignación clásicas (§I.6.1), las cuales asignan recursos a los trabajos para mejorar un criterio en particular. El interés del presente trabajo surge de conjuntar ideas obtenidas por la revisión de literatura: en el trabajo de (Tchernykh *et al.*, 2006) presentan varias políticas de calendarización para un Grid de dos niveles: en el primer nivel, el calenarizador asigna recursos a los trabajos de acuerdo a criterios de selección que toman en cuenta parámetros de los trabajos y disponibilidad de las máquinas. En el segundo nivel, cada máquina genera calendarios bajo su propio calendarizador local. Presentan algoritmos de calendarización basados en la combinación de estrategias de selección (asignación) de recursos y calendarización local. Consideran un escenario con trabajos propuestos al calendarizador provenientes de un ambiente descentralizado (los datos enviados por un recurso a otro por una conexión son recibidos por todos los demás recursos) y que pueden ser procesados dentro de un mismo lote (cantidad de trabajos disponibles para ser procesados). Su principal objetivo es comparar diferentes estrategias de calendarización y estimar su eficiencia. Como algoritmo de calendarización local utilizan el llamado *LSF* (*Largest Size First*) y como estrategias de selección: *Min-Load (ML)*, *Min-Load-admissible (ML-a)*, *Min-Parallel-Load (MPL)*, *Min-Parallel-Load-admissible (MPL-a)*, *Min-Lower-Bound (MLB)*, *Min-Lower-Bound-admissible (MLB-a)*, *Min-Completion-Time (MCT)*, and *Min-Completion-Time-admissible (MCT-a)*. La comparación de las estrategias *MLB-a* y *MCT-a-LSF* muestran que *MLB-a-LSF* tienen la misma cota en el peor caso como *MCT-a-LSF*, *MCT* se basa en calendarios existentes y requiere mayor esfuerzo computacional que *MLB*, la cual solamente se basa en parámetros de los trabajos. Con *MLB-a-LSF* el calendarizador puede seleccionar apropiadamente las máquinas. En el trabajo de Kurowski

et al., (2006) aplican el método de agregación (agregan, convierten varios criterios por medio de un operador en uno sólo, ver §III.2.1) al problema de calendarización trabajos en un Grid considerando múltiples criterios. Presentan dos modelos de problemas de administración de recursos en Grid: (i) problemas de calendarización en Grid con falta de información referente al tiempo de procesamiento de los trabajos, y (ii) calendarización de trabajos en donde se conocen algunas características de tiempo y que son descubiertas con ayuda de alguna técnica de predicción, así como de mecanismos de reserva de recursos. Se enfocan en demostrar, cómo estos dos escenarios (los cuales son ejemplos importantes de ambientes Grid) pueden ser modelados como problemas con múltiples criterios usando el método de agregación. En su trabajo utilizan el método de agregación para modelar las preferencias de las partes interesadas. Realizan un análisis de la calidad de las soluciones que se obtienen con este método. En el trabajo de Kurowski *et al.*, (2008) abordan el problema de calendarización con múltiples criterios en un sistema Grid computacional. Consideran un Grid jerárquico con dos niveles. En el primer nivel un meta-calendarizador asigna recursos (nodos) a los trabajos y posteriormente, cada nodo calendariza los trabajos que le son asignados utilizando un calendarizador local. En su enfoque, toman en cuenta las preferencias de los grupos interesados en la calendarización de Grid (usuarios finales, administradores del Grid y proveedores de recursos) y suponen que no conocen los tiempos de los trabajos. Estudian el impacto del tamaño de los conjuntos de trabajos (lotes de trabajos) sobre la eficiencia (relación entre el resultado alcanzado y los recursos utilizados) de calendarización. Para efectuar sus experimentos utilizan cargas compuestas por distintas fuentes (cargas sintéticas) y un algoritmo del tipo voraz (*greedy*). Concluyen que el desempeño de las estrategias depende de los parámetros de la carga de trabajo (conjunto de trabajos).

En la literatura ya se ha aplicado cómputo evolutivo a los problemas de calendarización, por mencionar un ejemplo, en Esquivel *et al.*, (2002) resuelven un problema de *Job Shop*[4] para uno y varios objetivos. Dichos autores aplican el método de

[4] Generalmente, los problemas de asignación y calendarización corresponden a una de las siguientes configuraciones (T'kindt y Billau, 2006): *Máquinas paralelas* (*Parallel Machines, P/Q/R, ver §I.7*): existe

agregación (§III.2.1) para múltiples criterios. Toman en cuenta la multi-recombinación, donde multiples soluciones se combinan varias veces para dar lugar a otras soluciones (*Multi Crossover per Couple, MCMP*) lo cual permite aplicar varios operadores (métodos para generar soluciones apropiadas, es decir que cumplan con las restricciones del problema) a un par de soluciones seleccionadas, y su extensión se genera aplicando diversos operadores a múltiples soluciones (*Multiple Crossovers on Multiple Parents, MCMP*), es decir, aplicar varios operadores de cruzamiento a varias parejas seleccionadas. Consideran tres criterios para minimizar el C_{max}, el retraso global (cuando los trabajos tienen fechas límite para ser entregados) y C_{max} ponderado (multiplicado por algún parámetro que le adhiera cierta importancia). Para los problemas de calendarización en Grid utilizando algoritmos genéticos se citan los siguientes trabajos: en Carretero y Xhafa, (2006) presentan un algoritmo genético para calendarizar trabajos en un Grid computacional de gran escala (cientos o miles de recursos) optimizando el C_{max} y el tiempo promedio de permanencia de los trabajos en el sistema (*flowtime*). El trabajo no contempla el caso multi-criterio. El objetivo que se plantea es lograr un calendarizador eficiente capaz de asignar una gran cantidad de trabajos en los recursos del Grid. Examinan varios operadores de algoritmos genéticos para identificar cuál brinda la mejor solución para el problema. Realizan extensos experimentos e identifican la configuración de operadores y parámetros que superan las ya existentes en la literatura. Sus resultados experimentales muestran que la implementación es robusta (que no se altera por la variación de los factores secundarios), mejora el rendimiento de los casos en comparación con los de la literatura; finalmente, el calendarizador que proponen se torna interesante debido a una rápida reducción del C_{max}. En el trabajo de Lorpunmanee *et al.*, (2006) proponen un modelo para calendarización de trabajos en el ambiente Grid considerando múltiples criterios. Presentan estrategias de asignación de trabajos hacia los diferentes nodos. Utilizan un algoritmo genético para generar calendarios. Realizan pruebas preliminares y muestran cómo las soluciones encontradas pueden minimizar el tiempo de

---

Solamente una etapa y los trabajos constan de una operación. *Taller de flujo de trabajos híbrido* (*Hybrid Flowshop, HF*): todos los trabajos tiene la misma ruta de producción, además, usan las etapas en el mismo orden. *Taller de trabajos general* (*General Jobshop, GJ*): cada trabajo tiene su propia ruta. *Taller de trabajos general* (*General Openshop, GO*): los trabajos no tienen una ruta fija.

espera promedio y el C_{max} en el Grid. Finalmente, afirman bajo sus resultados experimentales que la calendarización considerando varios criterios puede realizarse de manera eficiente y efectiva aplicando algoritmos genéticos. Posteriormente, en Carretero y Xhafa, (2007) presentan un trabajo de calendarización basado en algoritmos genéticos (ver §III.6) para realizar una asignación eficiente de los recursos a los trabajos en un sistema Grid. Presentan un extenso estudio sobre el uso de los algoritmos genéticos para diseñar calendarizadores eficientes (que se usan de la mejor manera) cuando se requiere minimizar el C_{max} y la suma de los tiempos de finalización de todos los trabajos. Consideran dos esquemas de codificación e implementan diversos operadores genéticos. Con base en un extenso estudio muestran que el desempeño del calendarizador basado en algoritmos genéticos supera las implementaciones ya existentes en la literatura. Identifican a las versiones de algoritmos genéticos que trabajan mejor bajo ciertas características del Grid, lo cual tiene un verdadero impacto ya que se presentan comúnmente en Grids reales. En Lim *et al.*, (2007) proponen la estructura (*framework*) de un algoritmo genético paralelo sobre un Grid jerárquico (*Grid-Enabled Hierarchical Parallel Genetic Algorithm, GE-HPGA*). La estructura ofrece una solución para problemas con funciones de aptitud costosa porque oculta la complejidad del ambiente Grid. Para evaluar la estructura realizan análisis teórico sobre la máxima aceleración (*speed-up*) del *GE-HPGA*. En el primer nivel del Grid se encuentra el modelo de islas en donde evolucionan las sub-poblaciones y en el segundo nivel denominado Amo-Esclavo se evalúan los cromosomas. Finalmente, en el trabajo de Grimme *et al.*, (2008) introducen una metodología para aproximarse a las soluciones óptimas en el problema de asignación de recursos en un Grid. Proponen utilizar NSGA-II (*Non-dominated Sorting Genetic Algorithm,* ver §III.4) como método de optimización multi-objetivo (§III.4). La idea es intercambiar trabajos entre sitios heterogéneos del Grid y para ello consideran el problema de encontrar una fracción óptima de toda la carga de trabajo.

Para resolver el problema de calendarización considerando múltiples criterios (§II.1) se aplica algoritmos genéticos (§III.6) como estrategia de asignación (§I.6.1) y el método de agregación de criterios (§III.2.1).

I.8.1. Objetivo principal

Implementar, comparar y analizar algoritmos genéticos aplicados al problema de calendarización con múltiples criterios en un Grid computacional usando un método de agregación.

I.8.2. Objetivos específicos

1. Determinar los operadores de los algoritmos genéticos y los operadores del método de agregación.
2. Proponer la representación de una solución para el algoritmo genético.
3. Adaptar los operadores genéticos encontrados en la literatura para la representación.
4. Analizar y seleccionar las cargas de trabajo.
5. Realizar experimentos preliminares que ayuden a determinar la configuración de los experimentos finales.
6. Analizar y comparar los resultados finales.

I.9. Organización del trabajo

El contenido de la tesis se encuentra organizado de la siguiente manera: el Capítulo 2 (Problema de calendarización de trabajos multi-criterio) presenta la definición formal del problema, conceptos básicos de problemas multi-objetivo y los métodos que existen en la literatura para resolverlos. En el Capítulo 3 se realiza una descripción general de los algoritmos evolutivos, así como los algoritmos genéticos que se aplican para resolver el problema que se define en el Capitulo 2; también se muestran los operadores de cruzamiento y mutación. En el Capítulo 4 se efectúa el análisis de la carga que se usa para realizar los experimentos, se presenta la configuración de los experimentos, los resultados y el análisis. Finalmente, en el Capítulo 5 se enumeran las conclusiones del trabajo y se plantean las posibles líneas a seguir.

Capítulo II

Problema de calendarización multi-criterio

II.1. Formulación del problema

El problema de calendarización de trabajos en el Grid se plantea de la siguiente manera: se deben calendarizar n trabajos paralelos $J_1, J_2, \ldots, J_n$ independientes en m máquinas del Grid computacional $M_1, M_2, \ldots, M_m$; m_i denota el número de procesadores idénticos en la máquina M_i. Se realiza una indización de las máquinas paralelas en orden ascendente de acuerdo a su tamaño (grado de paralelismo en función del número de procesadores) $m_1 \leq m_2 \leq \cdots \leq m_m$. Cada trabajo J_j se describe por una tripleta $(r_j, size_j, p_j)$, donde el tiempo a partir del cual el trabajo J_j está disponible para ser procesado es $r_j = 0$, su tamaño (número de procesadores sobre los que se ejecuta) $size_j$ está en el intervalo $1 \leq size_j \leq m_m$, lo cual indica su grado de paralelismo; el tiempo de ejecución se denota por p_j y se procesa exclusivamente sobre $size_j$ procesadores. Se supone que no hay dependencia entre los trabajos y se pueden ejecutar en algún momento, en algún orden y sobre alguna máquina. También se supone una calendarización en modo de espacio compartido (los trabajos se asignan a una máquina con grado de paralelismo mayor a uno), donde para cada trabajo J_j, el área de trabajo se denota como $w_j = p_j \cdot size_j$. Todos los trabajos están disponibles al instante cero ($r_j = 0$) y se procesan en un mismo lote, ello implica que un nuevo conjunto de trabajos disponibles se ejecuta hasta que se termine de procesar el lote que está en curso. El tiempo de ejecución no se conoce hasta que el trabajo finaliza su procesamiento. Se supone que el trabajo J_j puede ejecutarse en la máquina M_i si se cumple la restricción $size_j \leq m_i$. Un trabajo paralelo J_j se ejecuta exactamente sobre $size_j$ procesadores sin interrupciones (el procesamiento del trabajo no se detiene temporalmente en un punto determinado por alguna instrucción).

Ahora se establecen las funciones objetivo a minimizar. La primera función objetivo que se requiere minimizar es el tiempo de finalización máximo de todos los trabajos procesados sobre el sistema (C_{max}, *makespan* o longitud del calendario):

$$f_1 = C_{max} = max\ (C_{M_i})\ i = 1,2,3,\ldots,M_m \tag{10}$$

donde C_{M_i} es el tiempo de finalización máximo en la máquina M_i, M_m indica el número de máquinas disponibles en el sistema. Es importante minimizar esta función desde el punto de vista del administrador de recursos porque ello implica maximizar la utilización de los procesadores.

La segunda función que se considera para minimizar es el tiempo promedio de permanencia de los trabajos en el sistema (*Mean turnaround* o *Mean flow time*):

$$f_2 = TA = \frac{1}{n}\sum_{j=1}^{n}(C_j - r_j) \tag{11}$$

donde C_j es el instante de finalización del trabajo j y r_j es el instante en que está disponible. El tiempo inicia a contar desde el instante en que el trabajo llega al sistema y el conteo se detiene en el instante en que se termina de procesar dicho trabajo. Esta función beneficia al usuario ya que indica el tiempo promedio que tarda en recibir una respuesta por parte del sistema.

La tercera función para minimizar es el cociente de respuesta promedio (*Mean response ratio o Mean Slowdown*):

$$f_3 = MRR = \frac{1}{n}\sum_{j=1}^{n}\frac{C_j - r_j}{p_j} \tag{12}$$

donde C_j es el instante de finalización del trabajo j y r_j es el instante en que está disponible (tiempo de respuesta), p_j es el tiempo de procesamiento del trabajo j.

Finalmente, la cuarta función que se desea minimizar es el tiempo que en promedio esperan los trabajos para ser ejecutados (*Mean waiting time*):

$$f_4 = MWT = \frac{1}{n}\sum_{j=1}^{n}(C_j - r_j - p_j) \tag{13}$$

donde C_j es el instante de finalización del trabajo j y r_j es el instante en que está disponible (tiempo de respuesta), p_j es el tiempo de procesamiento del trabajo j.

El problema de calendarización en Grid consiste en asignar trabajos a máquinas para procesarlas respetando las restricciones impuestas. En la notación[5] corta de tres campos se describe como $GP_m|r_j = 0, size_j, p_j|OWA$. Donde OWA es el operador de agregación (§III.6.3): $OWA = (w_1 C_{max} + w_2 TA + w_3 + MRR + w_4 MWT)$, w_i es la combinación lineal de pesos e indica la relativa importancia de cada objetivo (§III.2.4). Con la notación *MPS* (*Multi-Parallel Stage*) se hace referencia a los dos niveles de calendarización en Grid con (Graham *et al.*, 1979): $MPS = MPS_{Alloc} + PS$. En el primer nivel (MPS_{Alloc}), se asigna una máquina disponible para cada trabajo utilizando alguna estrategia de selección de recursos (§I.6.1). En el segundo nivel, se aplica un algoritmo *PS* (§I.6.2) a cada máquina con el conjunto de trabajos que le fueron asignados a partir del primer nivel. El problema de calendarización en el segundo nivel se denota con ($P_m|r_j = 0, size_j, p_j|OWA$) (Tchernykh *et al.*, 2008).

En el presente trabajo, se utiliza el cociente de competitividad (*Competitive ratio*) para analizar los algoritmos porque indica la aproximación (cuántas veces se aleja del óptimo) de un algoritmo a una mejor cota mínima del tiempo de finalización (longitud del calendario); se define como el cociente de la longitud del calendario obtenido por alguna estrategia de calendarización con la longitud del calendario óptimo. Debido a que no se cuenta con el resultado calendario óptimo, se calcula la cota mínima. Por lo que el cociente de competitividad se obtiene dividiendo el resultado del calendario generado por la estrategia y la cota mínima de finalización del calendario: Sea C^*_{max} y $C_{max}(A)$ las longitudes de los calendarios óptimo y el determinado por el algoritmo A, respectivamente.

---

[5] La notación corta de 3 campos describe el modelo de Grid que se utiliza. La notación $\alpha|\beta|\gamma$ es un esquema que clasifica a los problemas de calendarización. Cada uno de los tres campos especifica un aspecto de los problemas. El campo α especifica el ambiente de los procesadores; el campo β las características de los trabajos y el campo γ el criterio a optimizar (Sinnen, 2007).

El cociente de competitividad del algoritmo A se define como $\rho_A = max\left(\frac{C_{max}(A)}{C^*_{max}}\right)$ para todas las instancias del problema.

Entonces, el problema de calendarización en Grid considerando múltiples criterios usando un método de agregación, es minimizar (f_1, f_2, f_3, f_4). De acuerdo al enfoque de agregación (§III.2.1) con que se trata el problema multi-criterio del presente trabajo, se proporciona una solución única aplicando el operador *OWA* (§III.6.3).

II.2. Problemas multi-objetivo

Existen tres posibles situaciones al tratar de optimizar un problema multi-objetivo (*Multi-Objective Problem, MOP*):

1. Maximizar todos los objetivos
2. Minimizar todos los objetivos
3. Minimizar algunos objetivos y maximizar otros.

Por razones de simplicidad, normalmente todas las funciones se convierten a la forma de maximización o minimización. Por ejemplo, se puede utilizar la siguiente identidad para convertir todas las funciones que se desean maximizar a minimizar:

$$max f_i(x) = -min(-f_i(x)) \tag{14}$$

Similarmente, las restricciones de la forma:

$$g_i(x) \leq 0 \quad i = 1, 2, \ldots, z \tag{15}$$

donde z indica el número de restricciones, se pueden convertir a la forma:

$$q_i(x) \geq 0 \quad i = 1, 2, \ldots, z \tag{16}$$

multiplicando por -1 y cambiando el signo de la desigualdad, entonces, la ecuación es equivalente a

$$-g_i(x) \geq 0 \quad i = 1, 2, \ldots, z \tag{17}$$

Un problema multi-objetivo tiene varios componentes:

Variables de decisión (Zitzler *et al.*, 2004 y Coello, 2001). Son las componentes de un vector $x = [x_1, x_2, ..., x_n]^T$ cuyos valores se deben escoger para la optimización.

Restricciones (Zitzler *et al.*, 2004 y Coello, 2001). Son parte inherente del problema, pueden ser físicas, de tiempo o de otro tipo. Estas restricciones se deben satisfacer para que la solución sea apropiada. Pueden ser restricciones de desigualdad o desigualdad:

$$g_i(x) \geq 0 \quad i = 1, 2, ..., z \tag{18}$$

$$h_i(x) = 0 \quad i = 1, 2, ..., p \tag{19}$$

donde z y p indican el número de restricciones.

Funciones objetivo (Zitzler *et al.*, 2004 y Coello, 2001). Son las funciones que se deben optimizar (maximizar o minimizar). Se puede ver como un vector de la forma:

$$\mathbf{f}(x) = [f_1(\mathrm{x}), f_2(\mathrm{x}), ..., f_u(\mathrm{x})]^T$$

Las variables de decisión y las funciones objetivo definen dos espacios a considerar (Zitzler *et al.*, 2004, Coello, 2001):

1. El espacio de las variables de decisión **x** (decisión space), el cual es de dimensión a. Cada punto en este espacio corresponde a un vector **x**.
2. El espacio de criterios $\mathbf{f(x)}$ (criteria space), el cual es de dimensión b. Cada punto es la correspondiente imagen, a por medio de $\mathbf{f}(\cdot)$, de una solución **x** al problema.

Definición 1. (Zitzler *et al.*, 2004 y Coello, 2001). El *Punto Ideal* o *Vector Ideal* $\mathbf{z}^*$ que está compuesto por los mejores valores objetivo que se pueden obtener. Esto es, $\boldsymbol{z_j^* = max/min\{f_j(\mathrm{x}) | \mathrm{x} \in A\}}$, $\boldsymbol{A}$ es el conjunto de todos los posibles valores de **x**, $\boldsymbol{j = 1, 2, ..., m}$, donde $\boldsymbol{m > 1}$ es el número de funciones objetivo.

Definición 2. (Aceves, 2003) *Dos funciones objetivo están en conflicto* si la distancia Euclidiana desde el punto ideal al conjunto de los mejores valores de las soluciones apropiadas es diferente de cero.

Definición 3. (Aceves, 2003) Tres o más funciones objetivo están en conflicto si están en conflicto por pares.

Al estar en conflicto las funciones objetivo hace que se tengan varias soluciones para el mismo problema multi-objetivo (Steur, 1986). Es decir, mientras un problema con un solo objetivo se resuelve encontrando el mejor valor para una función dada; un problema multi-objetivo se resuelve encontrando un conjunto de vectores que no se pueden comparar entre sí (Veldhuize y Lamont, 2000; Knowles y Corne, 2002; Zitzler *et al.*, 2002). En la Figura 2 se presenta un ejemplo sencillo (con cinco trabajos) del conflicto entre el tiempo máximo de finalización de todos los trabajos (C_{max}, ver §II.1) y el tiempo promedio de permanencia de los trabajos en el sistema (TA, ver §II.1): se desea optimizar ambas funciones simultáneamente en una máquina que consta de dos procesadores; al optimizar C_{max} (Figura 2 izquierda) se obtiene un calendario con 11 unidades de tiempo (no se puede reducir más), bajo el mismo calendario se obtiene $TA = 5.2$. Al minimizar TA (existe la manera de hacerlo) se logra un valor de 4.8, pero C_{max} se incrementa a 12 unidades de tiempo (Figura 2 derecha).

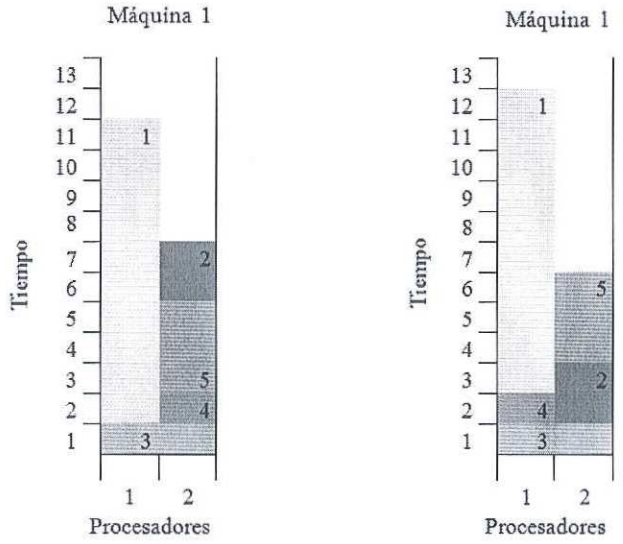


Figura 2: Conflicto entre dos funciones objetivo. Al optimizar la longitud del calendario para cinco trabajos sobre una máquina con dos procesadores, el tiempo promedio de permanencia de los trabajos se incrementa (izquierda). Si se minimiza el tiempo promedio de permanencia de los trabajos, entonces se incrementa la longitud del calendario.

Cuando las funciones objetivo se encuentran en conflicto se tienen varias soluciones para el mismo problema multi-objetivo, es decir, dicho problema se resuelve encontrando un conjunto de vectores que se pueden comparar entre sí. Enseguida se presentan las relaciones entre dos vectores (cuyos valores se deben escoger para la optimización).

II.2.1. Relación entre vectores

Para dos vectores cualesquiera $v = [v_1, v_2, \dots, v_u]^T$ y $w = [w_1, w_2, \dots, w_u]^T$ existen cuatro relaciones entre ellos. Estas relaciones son de utilidad, ya que cada solución de un problema mult-objetivo se puede ver como un vector de u componentes, donde u es el número de funciones objetivo. Cada componente del vector es el valor que toma la función objetivo respectiva. Enseguida se describen tales relaciones ejemplificando para el caso de minimización, sin pérdida de generalidad se puede aplicar a maximización:

1. El vector $\boldsymbol{v}$ *domina* al vector $\boldsymbol{w}$ ($\boldsymbol{v} \prec \boldsymbol{w}$) (Zitzler *et al.*, 2004). Para cada componente de $\boldsymbol{w}$ los componentes de $\boldsymbol{v}$ son iguales o mejores ($v_i \leq w_i$ para todo $i = 1, 2, \dots, u$; y $v_i \neq w_i$ para al menos un i).
2. El vector $\boldsymbol{v}$ *es dominado* por el vector $\boldsymbol{w}$ ($\boldsymbol{v} \succ \boldsymbol{w}$) (Zitzler *et al.*, 2004). Para cada componente de $\boldsymbol{v}$ los componentes de $\boldsymbol{w}$ son iguales o mejores ($v_i \geq w_i$ para todo $i = 1, 2, \dots, u$; y $v_i \neq w_i$ para al menos un i).
3. El vector $\boldsymbol{v}$ *es igual* al vector $\boldsymbol{w}$ ($\boldsymbol{v} = \boldsymbol{w}$) (Zitzler *et al.*, 2004). En cada uno de los componentes de $\boldsymbol{v}$, los componentes de $\boldsymbol{w}$ son los mismos ($v_i = w_i$ para todo $i = 1, 2, \dots, u$).
4. El vector $\boldsymbol{v}$ *no el comparable* con el vector $\boldsymbol{w}$ ($\boldsymbol{v} \succ\!\prec \boldsymbol{w}$) (Zitzler *et al.*, 2004). En este caso se dice que ambos vectores son igual de buenos o igual de malos ($v_i \leq w_i$ y $v_j \geq w_j$para algun $i, j = 1, 2, \dots, u$ y $v \neq w$).

A diferencia de un problema de un solo objetivo en donde se desea encontrar una solución, en un problema multi-objetivo se pretende encontrar un conjunto de soluciones que son causadas por los objetivos en conflicto (Coello *et al.*, 2002). El conjunto que se quiere encontrar se denomina frente Pareto y se describe a continuación.

II.2.2. Frente Pareto

Definición 4. (Coello *et al.*, 2002) Para un problema de optimización multi-objetivo, el conjunto óptimo de Pareto ($\mathbf{P}_{true}$) es

$$P_{true} := \{x \in A | \nexists\, y \in A\ f(y) \prec f(x)\}$$

A es el conjunto de soluciones.

Definición 5. (Coello *et al.*, 2002) Para un problema de optimización multi-objetivo y un conjunto óptimo de Pareto ($\mathbf{P}_{true}$), el *frente Pareto* se define como:

$$PF_{true} := \{u = f(x) = [f_1(x), f_2(x), \ldots, f_m(x)]^T | x \in P_{true}\}$$

Entonces, un problema multi-objetivo se define de la siguiente manera:

Definición 6. Coello *et al.*, 2002. *Problema multi-objetivo*. Encontrar los vectores $\mathbf{x} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$ que satisfacen las z y p restricciones de desigualdad correspondientemente

$$g_i(x) \geq 0 \quad i = 1, 2, \ldots, z \tag{20}$$

$$h_i(x) = 0 \quad i = 1, 2, \ldots, p \tag{21}$$

y optimizan el vector compuesto por u funciones

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

donde optimizar $\mathbf{f}(\mathbf{x})$ significa encontrar el conjunto óptimo de Pareto.

La solución de un problema multi-objetivo es un conjunto de soluciones que tienen el mejor valor posible en todas las funciones objetivo, y que no pueden ser mejores entre ellas. Este conjunto se llama Frente Pareto (soluciones no dominadas) (Steur, 1986). El conjunto óptimo de Pareto se encuentra en el espacio de decisión y el frente Pareto se encuentra en el espacio de criterios (Aceves, 2003).

Hasta el momento se cuenta con la definición del problema, se sabe que las funciones están en conflicto, entonces, lo que se busca es dar solución al el problema de calendarización de trabajos multi-objetivo en Grid. Por lo tanto, el tema siguiente a tratar es la metodología (computo evolutivo).

Capítulo III

Algoritmos evolutivos

De acuerdo a Wright, (1932), existen dos candidatos poderosos para resolver los problemas: el cerebro humano (creó la rueda, ciudades, guerras, entre otros) y el proceso evolutivo (creador del cerebro humano). Esta última respuesta es el fundamento de los algoritmos evolutivos.

El término algoritmos evolutivos (*Evolutionary Algorithm, EA*) es un método de optimización estocástica (Coello *et al.*, 2002) que simula el proceso de evolución natural de los seres vivos. Estos métodos no garantizan identificar las soluciones óptimas, pero tratan de encontrar una buena aproximación, *i.e.*, un conjunto de soluciones cuyos vectores objetivo (§II.2) no están muy lejos de los vectores con soluciones óptimas (Zitzler *et al.*, 2004). El conjunto de soluciones se inicia bajo algún método o simplemente al azar. Cuentan con un proceso de selección en donde se puede distinguir el ambiente de apareamiento (conjunto de soluciones que se recombinan para formar nuevas soluciones) y el ambiente de selección (selección de soluciones). El ambiente de apareamiento tiene como objetivo detectar soluciones prometedoras para mantener la diversidad de soluciones. En contraste, el ambiente de selección determina cuál de las soluciones almacenadas previamente y de las nuevas se guardan. Durante el proceso evolutivo se debe mantener diversidad entre las soluciones (soluciones que no sean iguales o parecidas), para ello se toma un conjunto de soluciones y se modifica de manera sistemática o al azar con la idea de encontrar mejores soluciones potenciales. En resumen, un ciclo del proceso evolutivo (simulado) incluye: selección para el apareamiento, diversidad y selección; este ciclo se puede repetir hasta que se cumpla cierto criterio de paro (criterio para decidir el momento o circunstancias bajo las que se detienen los ciclos).

Los algoritmos evolutivos se han aplicado a problemas de optimización con uno o varios objetivos que previamente ya se han tratado de resolver con simulaciones numéricas. El interés por parte de los investigadores y el rápido crecimiento de los algoritmos evolutivos han hecho que se forme una nueva área enfocada a problemas que tratan múltiples objetivos simultáneamente, dicha área se denomina algoritmos evolutivos multi-objetivo (*Multiobjective Evolutionary Algorithms, MOEA*) (Chipperfield y Fleming, 1996; Coello, 1999; Coello, 2001; Deb, 2001; Büche *et al.*, 2002 y Esquivel *et al.*, 2002). En el trabajo de Zitzler *et al.*, (2004) brindan una revisión general acerca de algoritmos evolutivos multi-objetivo poniendo mayor énfasis en la teoría y métodos. Otro trabajo en el que realizan una revisión de los conceptos básicos relacionados con los algoritmos evolutivos que manipulan múltiples objetivos en la función es el de Coello, (2001); en su trabajo describen y critican los algoritmos evolutivos más usados en la literatura, además de sus aplicaciones.

III.1. Ventajas de los algoritmos evolutivos

Los procesos evolutivos se pueden simular en una computadora, donde se pueden realizar millones de ejecuciones en horas o días y repetidos bajo las mismas y/o distintas circunstancias. Naturalmente, la interpretación de dichas simulaciones se debe efectuar con sumo cuidado, pues los modelos no representan la realidad biológica con suficiente fidelidad. Los algoritmos evolutivos parecen ser apropiados para resolver problemas de optimización multi-criterio, porque pueden tratar con un conjunto de posibles soluciones simultáneamente. Ello, permite encontrar varios miembros del conjunto ideal en una sola ejecución del algoritmo, en lugar de realizar una serie de ejecuciones separadas como en el caso de las técnicas matemáticas tradicionales (Coello, 2001). Adicionalmente, los algoritmos evolutivos son menos susceptibles a la forma o continuidad del frente Pareto (*e.g.*, pueden tratar fácilmente con frentes Pareto continuos o convexos, es decir, puede encontrar las soluciones posibles dentro de un intervalo). A pesar de que los mecanismos de selección y recombinación básicos son simples, estos algoritmos han probado ser robustos y herramientas poderosas para búsqueda de soluciones (Zitzler *et al.*, 2004). Poseen

características que son deseables para resolver problemas que incluyen: criterios múltiples que pueden estar en conflicto y espacios de búsqueda complejos.

III.2. Clasificación de los algoritmos evolutivos multi-objetivo

Existen diversas maneras de clasificar a los MOEAs (Coello *et al.*, 2002) y una forma sencilla de hacerlo y que utiliza un enfoque basado en el tipo de selección, es la siguiente:

1. Funciones de agregación lineal (*Linear Aggregating Functions*).
2. Enfoque basado en la población (*Population-Based Approaches*).
3. Enfoque basado en el Pareto (*Pareto-based Approaches*).

III.2.1. Funciones de agregación lineal

Un algoritmo genético (III.6.3) depende de una función de aptitud para guiar la búsqueda de soluciones. Por tanto, el enfoque más intuitivo (sencillo) puede ser, tratar con objetivos múltiples que se combinan en uno sólo, es decir, combinar dichos objetivos en un valor único escalar. Dentro de los MOEA's, a esta técnica se le conoce como "agregación de funciones" (Coello y Lamont, 2004), aunque este método sale a la luz con Das y Dennis, (1997). Típicamente, las funciones de agregación lineal o de utilidad tiene la forma de suma ponderada:

$$min \sum_{i=1}^{k} w_i f_i(x) \tag{21}$$

donde los coeficientes de peso $w_i \geq 0$ (combinación lineal de pesos) representan la importancia relativa de los k objetivos del problema. Por lo general, se supone que

$$\sum_{i=1}^{k} w_i = 1 \tag{22}$$

Los resultados que se obtienen al tratar de resolver un problema de optimización utilizando esta función pueden variar significativamente de acuerdo a los pesos, por lo que, una posibilidad es resolver el mismo problema con diversos valores de w_i.

III.2.2. Ventajas de las funciones de agregación lineal

1. La sumatoria ponderada es fácil de entender y de implementar.
2. Este método es aceptable para encontrar una solución atractiva en problemas con múltiples criterios si el dominio del problema es "*fácil*" o si el tiempo disponible para la búsqueda es corto.
3. Tomando en cuenta, estas características, para alguna función de aptitud y algún conjunto de pesos positivos, el óptimo global que se obtiene, siempre es un elemento del conjunto Pareto óptimo.
4. Este enfoque no requiere cambios para aplicarlo con algoritmos genéticos, es decir, es simple, fácil para implementar y eficiente.
5. El mecanismo puede trabajar apropiadamente con problemas de optimización multi-criterio con pocas funciones objetivo y espacios de búsqueda convexos.
6. Se puede aplicar a problemas reales de investigación e ingeniería en los que se puede incorporar algunas variaciones a la forma de la función, es decir, el método es flexible.

III.2.3. Desventajas de las funciones de agregación lineal

1. Un problema que salta rápido a la vista, es generar un conjunto de pesos apropiado para el problema.
2. Presenta limitaciones ante problemas multi-criterio que no son convexos ya que no se puede encontrar la solución óptima a pesar de los pesos que se utilicen.
3. A pesar de que el método de agregación fue popular, ahora es menos común que los enfoques basados en Pareto.

III.2.4. Combinación lineal de pesos

Las soluciones no inferiores (las mejores soluciones) se pueden encontrar resolviendo un problema de optimización en el que la función objetivo es una sumatoria ponderada del vector evaluado (Zadh, 1963). Los pesos son estrictamente positivos para todos los objetivos. El conjunto no inferior de soluciones se puede generar al variar los

pesos en la función objetivo. Tal hecho lo demostró Gass y Saaty, (1955) para un problema con dos objetivos.

Es importante notar que los pesos no reflejan proporcionalmente (relación entre la magnitud del peso y la magnitud de la importancia) la importancia relativa de los objetivos, pero son factores que al variar, localizan puntos en el conjunto Pareto óptimo.

III.3. Enfoque basado en la población

La intención de este enfoque es diversificar la búsqueda, pero no encierra el concepto de Pareto dominado, *i.e.*, no se incluye directamente en el proceso de selección. Para entenderlo, comúnmente se toma como ejemplo el algoritmo genético vector de evaluación (*VEGA*). Consiste básicamente de un algoritmo genético (§III.6) con una modificación en la selección. En cada generación, una parte de la población se genera por desempeño proporcional de acuerdo a cada función objetivo que se esté evaluando. Es decir, para un problema con k objetivos, se generan k sub-poblaciones de tamaño P_T/k en cada generación. Estas sub-poblaciones se mezclan para obtener una población de tamaño P_T, después, se aplican los operadores de cruzamiento (§III.6.6) y mutación (§III.6.7).

III.4. Enfoque basado en Pareto

Esta categoría engloba los algoritmos evolutivos multi-objetivo que cuentan con el concepto de Pareto óptimo dentro de su mecanismo de selección. En los últimos años se han propuesto diversos MOEAs, a continuación se presentan algunos:

1. *Categorias de Pareto de Goldberg* (*Goldberg's Pareto Ranking*) (Coello *et al.*, 2002). La idea de Goldberg consiste en mover la población hacia el Pareto óptimo usando un mecanismo de selección que favorezca las soluciones que son no dominadas con respecto a la población en curso. También sugiere que la diversidad se mantenga con base en un mecanismo de aptitud compartida.

2. *Algoritmo Genético Multi-Objetivo* (*Multi-Objective Genetic Algorithm, MOGA*) (Coello *et al.*, 2002). Realiza una aproximación por clasificación. Cada individuo de la población se clasifica basándose en la cantidad de puntos (soluciones) que lo dominan. Todos los individuos no dominados en la población, se asignan a la misma clasificación y obtienen la misma aptitud, por ende, todos ellos tienen la misma posibilidad de seleccionarse.
3. *Algoritmo Genético con Clasificación no Dominada* (*The Nondominated Sorting Genetic Algorithm, NSGA*) (Coello *et al.*, 2002). La población se clasifica con base en su no-dominancia momentos previos a realizar la selección. Todos los individuos no dominados se clasifican en una categoría con un valor inicial de aptitud (valor postizo), el cual es proporcional al tamaño de la población, ello, con la idea de generar igualdad a la hora de la reproducción para los individuos. Para mantener la diversidad, los individuos clasificados comparten su valor postizo de aptitud. Después, este grupo de individuos clasificados se ignora y se considera otra capa (frente) de individuos no considerados previamente. Los individuos en el primer frente tienen la máxima aptitud, siempre obtienen más copias que el resto de la población. Una variante de este enfoque es el NSGA-II que usa elitismo y varias comparaciones para clasificar la población basada en el frente de Pareto y la región de densidad. Este operador de comparación hace que el NSGA-II sea considerablemente más rápido que su predecesor y genera buenos resultados.
4. *Algoritmo Genético con Nicho de Pareto* (*Niched Pareto Genetic Algorithm, NPGA*) (Coello *et al.*, 2002). Usa un método llamado torneo de Pareto dominado. Se seleccionan aleatoriamente dos miembros de la población y cada uno se compara con un subconjunto de la población. Si alguno es dominado y el otro no, entonces el miembro no dominado se acepta. Si ambos son dominados o no dominados, entonces los resultados del torneo se deciden con base en la aptitud.
5. *Algoritmo Evolutivo con fortaleza de Pareto* (*Strength Pareto Evolutionary Algorithm, SPA*) (Coello *et al.*, 2002). Este método integra diferentes algoritmos evolutivos multi-objetivo. El algoritmo usa un valor de fortaleza que se calcula de manera similar al sistema de clasificación MOGA. A cada miembro de la población

se le asigna un valor de aptitud de acuerdo a su fuerza tomando en cuenta las soluciones no dominadas.

III.5. Complejidad de los algoritmos evolutivos multi-objetivo

Al hablar de complejidad computacional se hace referencia a los recursos que requiere un algoritmo durante la resolución de un problema (Coello *et al.*, 2002). El estudio de la complejidad es tan amplio que existe una rama de la teoría de la computación llamada "*Teoría de la Complejidad Computacional*". La palabra recursos indica comúnmente el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema), también al espacio requerido (memoria) para resolver un problema, número de procesadores que se necesitan para resolver el problema en paralelo, aunque se pueden medir otros parámetros. Como ejemplo, existen problemas con tiempo polinomial, es decir, para una entrada de tamaño a, el peor caso de su tiempo de ejecución es $O(a^c)$ para alguna constante c (Cormen *et al.*, 2001); los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño, *i.e.* $c = 1$.

Cuando se discute la complejidad de diferentes MOEAs se debe enfocar principalmente en el número de evaluaciones que realizan; también se comparan las soluciones y se consideran algunos cálculos adicionales. Se sabe que al aplicar MOEAs a problemas reales, las funciones de evaluación consumen demasiado tiempo de ejecución (Coello *et al.*, 2002). Generalmente la complejidad de los MOEAs es más grande que la de los algoritmos genéticos. Después que un algoritmo genético realiza las evaluaciones los resultados se almacenan en memoria y normalmente no se requieren más cálculos para conocer la aptitud; en cambio un MOEA en ocasiones combina y/o compara esos valores almacenados, los cuales adjuntan complejidad al algoritmo (Coello *et al.*, 2002).

Se sabe que un algoritmo es eficiente cuando el éxito de tal algoritmo no depende en ningún caso de la velocidad ni del sistema en que se ejecuta. En ese caso, un algoritmo

es mejor que otro, aunque el segundo se ejecute en un sistema completamente superior. El costo computacional mide la eficiencia de los algoritmos y se conoce estableciendo un sistema simple dándole un tiempo de ejecución a cada operación elemental y contando el número de operaciones elementales que realiza un algoritmo en concreto. Con base estas ideas, el componente más costoso en un algoritmo evolutivo es la función de evaluación. Como todos los algoritmos deben terminar el número de evaluaciones, a menudo se calcula los recursos que se gastan en la búsqueda. Suponiendo que las soluciones no se evalúan más de una vez, se exploran un total de P_T puntos en el espacio de búsqueda. Ahora considérese una función con k-objetivos. Las k funciones de evaluación se emplean para cada solución posible (una por cada objetivo). Suponiendo que los recursos son limitados a e funciones de evaluación y que cada objetivo es igual de costoso, solamente se exploran $\left\lfloor \frac{P_T}{k} \right\rfloor$ puntos en el espacio de búsqueda.

Estos resultados implican que un MOEA puede requerir gran tiempo de ejecución para obtener buen desempeño aunque tengan un solo objetivo. No se garantiza que la respuesta sea la óptima pero es deseable explorar el mayor espacio en el tiempo permitido. Esto incrementa la confianza para encontrar el óptimo global y no uno local (Coello *et al.*, 2002).

III.6. Algoritmos genéticos

Los Algoritmos Genéticos (AGs) (Holland, 1975) son métodos que se utilizan para resolver problemas de optimización y se basan en el proceso evolutivo de los organismos. Dentro de los AGs, a cada solución candidata se le denomina *individuo* o *cromosoma* y a un conjunto de esas soluciones se le llama *población* (P_T). Cuentan con un proceso de *selección* para el apareamiento (proceso en el que dos o más individuos se combinan para formar un nuevo individuo) que consiste de dos etapas: asignación de aptitud y muestreo. En la primera etapa, se realiza la *evaluación* de los individuos y se le asigna un valor escalar llamado *aptitud*, que refleja su calidad. Posteriormente, se crea el grupo de apareamiento (*padres*) copiando a éste conjunto aquellas soluciones con la mejor aptitud

(etapa de muestreo) hasta que se completa la población. Después, al conjunto de apareamiento se le aplica operadores para mantener la diversidad (soluciones lo más diferentes en la medida de lo posible). Usualmente se conocen dos operadores: cruzamiento (recombinación) y mutación (cambio ligero en un individuo); con base en la probabilidad (P_c y P_m, respectivamente), el operador de cruzamiento toma un número de padres (soluciones) y crea un número de *descendientes* o *hijos* (nuevas soluciones) formados a partir de dichos padres. Bajo el esquema de probabilidad, algunos individuos pueden no ser afectados por tales operadores. Enseguida se presenta la secuencia para el AG estándar que se aplica en el presente trabajo y que fue propuesto por Holland, (1975):

1. Iniciar aleatoriamente la población, generando individuos apropiado.
2. Evaluar cada uno de los individuos con ayuda de la función de aptitud.
3. Aplicar selección para identificar a los individuos que se cruzarán.
4. Si se cumple el criterio de paro, terminar el proceso evolutivo y entregar el mejor individuo encontrado, si no, regresar al paso 2.
5. Aplicar un operador de cruzamiento.
6. Aplicar un operador de mutación.
7. Realizar reemplazo de la población anterior con la nueva población obtenida.

Los AGs tienen un problema nato, encontrar los parámetros de control con los que obtienen mejores resultados para el problema que se están aplicando, así como decidir cuáles operadores aplicar y cómo aplicarlos; cada uno de ellos tiene críticas tanto positivas como negativas en la literatura; aún no se logra un consenso para estandarizarlos, pues cada uno de ellos implica un análisis amplio y una dependencia fuerte del problema (Eiben y Smith, 2003).

En resumen, los AGs requieren de cinco componentes: representación (§III.6.1), población inicial (§III.6.2), función de evaluación (§III.6.3), operadores para evolucionar (§III.6.6 y §III.6.7) y parámetros (§IV.3).

En el trabajo de Carretero y Xhafa, (2006) aplican un algoritmo genético para calendarizar trabajos en un Grid. Aplican 3 criterios para generar la población inicial: el trabajo más grande/chico se asigna al recurso más rápido (*Longest/Shortest Job to Faste Resourse, LJFR-SJFR*), el segundo se basa en la asignación de trabajos a los recursos que ofrecen el tiempo mínimo de finalización de los trabajos (*Minimum Completion Time, MCT*) y el tercero con base en la función *Random* (al azar).

III.6.1. Representación

Una definición apropiada de un problema genera una buena formulación de las soluciones codificándolas en cromosomas (representación de una solución, también denominada individuo). Inicialmente, en el trabajo de Holland, (1975) se propone una representación binaria (vector unidimensional con elementos 0's y 1's), sin embargo el uso de dicha representación tiene varias desventajas para problemas reales (cadenas muy largas, el valor de un bit puede suprimir las contribuciones de aptitud de otros bits en el grupo, puede producir soluciones ilegales constantemente). Utilizar una representación adecuada permite simplificar el proceso de búsqueda de un AG. Por lo tanto, se requieren representaciones más poderosas (que tengan la capacidad de generar todas las soluciones posibles para un problema y que no generen soluciones erróneas) y conocer las variantes que surgen, como la representación con base en el conjunto de los números enteros. El uso adecuado de la representación permite ahorrar trabajo innecesario y concentrarse en la aplicación del AG en lugar de enfocar esfuerzo en el diseño de operadores genéticos especiales; entonces, la representación de un cromosoma debe ser simple porque permite la aplicación de operadores rápidos, con los cuales se puede lograr un buen desempeño del algoritmo.

En la calendarización de trabajos, un cromosoma representa una solución al problema, en otras palabras, representa un calendario apropiado. En el presente trabajo se aplica la representación entera; donde cada número hace referencia al índice de un trabajo. Los índices se estructuran en un vector de dos dimensiones, tal estructura se basa en la del modelo Grid jerárquico de dos niveles. Se utiliza la representación entera debido a que la

solución del problema es una permutación de números enteros. Cabe señalar que en la literatura referente a Grid no se encuentran trabajos que apliquen la estructura bidimensional. En la Figura 3 se representa un Grid donde se aprecia la cola global de trabajos (lote de tamaño 10) a los cuales se les asigna recursos aplicando algoritmos genéticos. Como consecuencia, en cada máquina se forma una cola local de trabajos de tamaño menor que la cola global. Se puede dar el caso en el que todas las colas locales sean de igual tamaño; en el peor caso, la cola de una máquina cuenta con los mismos trabajos de la cola global y el resto de las máquinas no tienen ningún trabajo. En general, la representación bidimensional trata evitar el último caso. Cada máquina con tamaño diferente calendariza los trabajos (recuadros achurados) que se le han asignado bajo alguna estrategia (§I.6.2). En la Figura 4 se representa el individuo que contiene las colas locales después efectuar la asignación de los trabajos con algoritmos genéticos. El conjunto de tres máquinas está representado por los renglones en el intervalo [1, 3] y las columnas representan el conjunto de trabajos [1, 10].

En el trabajo de Carretero y Xhafa, (2006) efectúan la codificación de las soluciones (representación) de longitud igual al número de tareas que desean calendarizar, donde *scheduler[i]* indica la máquina en donde el trabajo *i* se asigna.

Una vez que ha quedado comprendida la representación de las soluciones en un AG, el siguiente paso es enfocarse en el segundo elemento importante del proceso evolutivo, la población inicial (Eiben y Smith, 2003).

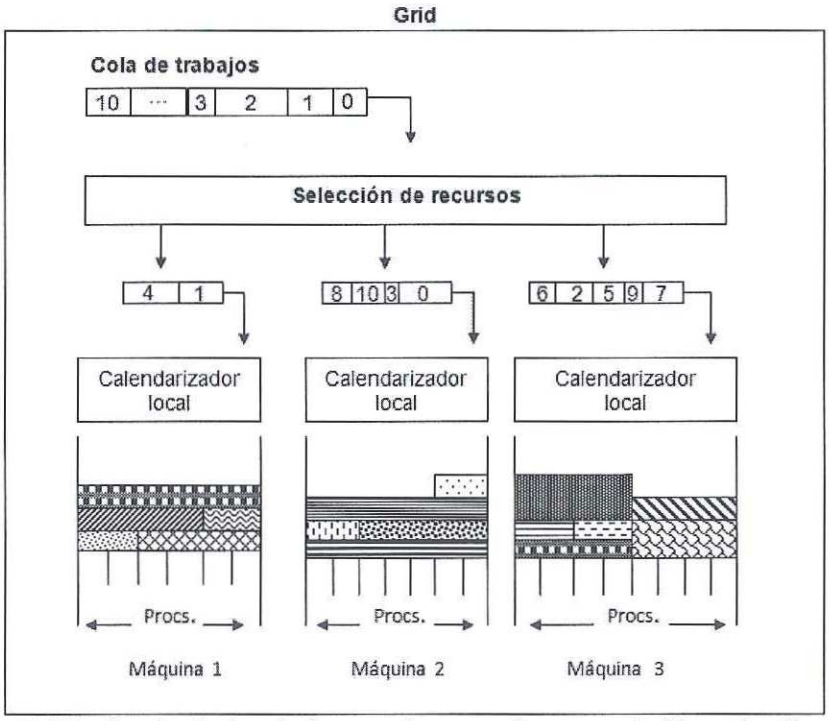


Figura 3. Grid jerárquico de dos niveles como base para la representación en dos dimensiones.

Figura 4. Representación de un individuo (cromosoma) en dos dimensiones con 11 trabajos.

III.6.2. Población inicial

Generalmente, en calendarización los AG inician la población inicial de manera aleatoria (en la representación binaria, el cero y el uno tienen la misma probabilidad de ocurrir). Bajo la representación entera es importante tener cuidado en generar soluciones apropiadas porque si se repite el índice de un trabajo, significa que le mismo trabajo se

calendariza dos veces, lo cual no es posible. Típicamente, la población es de tamaño P_T. Es común complementar una parte de la población inicial generada al azar con otra parte generada bajo algún criterio; la intención de enriquecer la población inicial y proporcionarle cierta velocidad al AG (encontrar mejores soluciones más rápido) y mejorar la calidad del resultado. La desventaja, es que se puede guiar la búsqueda hacia un mínimo local, por lo tanto, los criterios se deben utilizar con sumo cuidado y en pequeñas fracciones (Sinnen, 2007).

Tomando en cuenta las características del problema (§II.1) y la representación entera y la estructura bidimensional propuesta (§III.6.1), se utiliza el siguiente método para generar la población inicial: para cada trabajo se identifica su tamaño (número de procesadores solicitados); se selecciona una máquina aleatoriamente del intervalo de máquinas que cumplen con la solicitud. Para buscar una máquina, se supone que el conjunto de ellas está en un arreglo ordenado de menor a mayor tamaño (número de procesadores), se realiza una búsqueda secuencial hasta encontrar la primera máquina que cumpla con la solicitud, por ende, la última máquina es el fin del arreglo de máquinas. Este procedimiento se realiza hasta terminar de asignar los trabajos de la cola global (tal asignación forma un individuo). Se repite el proceso hasta completar el tamaño de la población total.

En el trabajo de Carretero y Xhafa, (2006) forman la población inicial bajo las propiedades de las soluciones. Una solucione factible para el problema que plantean está conformada por *compretion[m]* que es un vector de longitud m (máquinas) e indica el tiempo en el que se termina de procesar el trabajo asignado previamente.

III.6.3. Asignación de aptitud

El proceso de selección para el apareamiento (seleccionar soluciones para que a partir de ellas se nuevas), consiste de dos etapas: asignación de aptitud y muestreo. En la primera etapa, se evalúa a los individuos de la población actual y se le asigna un valor escalar, la aptitud que refleja su calidad. Posteriormente, con ayuda de la aptitud se realiza

un muestreo sobre los individuos y se crea la población de apareamiento. En calendarización de trabajos, la aptitud de un individuo (cromosoma) se relaciona directamente con el calendario (Sinnen, 2007), entonces, el objetivo es encontrar el calendario más corto posible. En contraste a la optimización de un objetivo, donde la función objetivo y de aptitud son la misma, en optimización con múltiples criterios la función de aptitud se debe aplicar para varios objetivos (Zitzler *et al.*, 2004).

En el presente trabajo se ha seleccionado el método de agregación (§III.2.1) para tratar el problema con múltiples criterios (ver §II.1). Este método se utiliza ampliamente en diversos campos por su simplicidad y facilidad de implementación. Con dicha técnica se puede manipular varios criterios. Se aplica el operador promedio ponderado de orden (*Ordered Weighted Averaging*, *OWA*) utilizado en Kurowski *et al.*, (2008); tal operador es apto para obtener un mejor desempeño global del sistema.

$$OWA(x_1, x_2, \dots x_n) = \sum_{j=1}^{k} w_j s(x)_{\sigma(j)} \tag{23}$$

donde x_j, $j = 1, \dots, k$ es un valor asociado a la satisfacción del *j-ésimo* objetivo (función objetivo). Se realiza una permutación (σ) que ordena los valores: $s(x)_{\sigma(1)} \leq s(x)_{\sigma(2)} \dots \leq s(x)_{\sigma(k)}$. Los pesos ($w_j$) son no negativos ($w_j \geq 0$) y la suma de todos ellos es igual a 1 $\sum_{j=1}^{k} w_j = 1$. Los pesos se generan a partir de (32).

El operador de agregación es útil porque es versátil comparado con los operadores básicos tales como la suma ponderada y la norma de Chebyshev que aplican en Kurowski *et al.*, (2008), es decir, el operador *OWA* es más general porque se puede transformar a minimizar, maximizar o promedio aritmético. Al poner un peso w_i (importancia de un criterio) en 1 y el resto de los pesos en 0, el operador *OWA* ayuda a minimizar. En el análisis multi-criterio, ello indica que se están buscando soluciones que tengan buenos valores del peor criterio, es decir, buenos valores para todos los criterios. Si a todos los pesos se les asigna el mismo valor, el operador *OWA* se comporta como el promedio aritmético. En este caso, los valores altos de algún criterio compensan los valores bajos de

los otros criterios. El peso más grande es w_1 y los subsecuentes se van decrementando pero nunca son 0. Esto significa que se toman en cuenta los casos anteriores, el caso del peor criterio y el caso promedio. Ello representa la posibilidad de evaluar calendarios que toman en cuenta varios criterios. La idea es encontrar un esquema de pesos que proporcionen el mejor valor promedio para todas las funciones de evaluación y el valor más alto para la peor función de evaluación. Para lograrlo, el peso w_1 debe ser relativamente grande, mientras el peso w_k debe ser chico. El resto de los pesos intermedios deben decrementarse Kurowski *et al.*, 2008 de acuerdo a la función:

$$(w_j) = \begin{cases} \dfrac{3}{2k} & , \quad i = 1 \\ \dfrac{3k - 2i - 1}{2n(k-1)} & , \quad 1 < j \leq k \end{cases} \tag{24}$$

Los criterios no se pueden comparar en su forma original debido a que están expresados en diferentes unidades y escalas. Los métodos escalares se utilizan para asegurarse que las funciones realmente reflejen la importancia (peso) de cada criterio Kurowski *et al.*, 2008. En el presente trabajo se utiliza la función escalar $s(x)$ porque ayuda a normalizar los criterios en el intervalo (0,1).

$$s(x) = \frac{s(x)}{max_{s(x)}} \tag{25}$$

III.6.4. Selección

El proceso de selección de candidatos (soluciones) a reproducirse (combinarse) es una parte fundamental de un AG. Por lo general, este proceso se realiza de forma estocástica (al azar), con este método, los individuos menos aptos también tienen probabilidad de sobrevivir (soluciones que no necesariamente son las mejores), a diferencia de las selecciones extintivas en donde los menos aptos no se consideran en definitiva.

Existen diferentes métodos para realizar la selección de individuos y clasificados en tres grupos de acuerdo a sus características (Eiben y Smith, 2003): selección proporcional: ruleta, sobrante estocástico, universal estocástica, muestreo determinístico, escalamiento sigma, selección jerárquica, selección de Boltzmann; selección mediante torneo:

determinística, probabilística; Selección de estado uniforme. A continuación se describe la selección mediante torneo determinístico porque es el método que se aplica en el algoritmo genético usado en el presente trabajo. El torneo realiza comparaciones directas entre los individuos y selecciona al mejor bajo los pasos siguientes:

1. Revolver los individuos de la población.
2. Escoger un número r de individuos (comúnmente 2), se seleccionan los que tienen mayor aptitud.
3. El ganador del torneo es el individuo con mayor aptitud.
4. Debe revolverse la población un total de r veces para seleccionar P_T (tamaño de la población) padres.

El algoritmo de selección por torneo probabilístico (no se aplica en el presente trabajo) es similar al de selección determinística, solamente cambia en el paso en donde se selecciona al ganador. En lugar de seleccionar siempre al que tiene mayor aptitud, se aplica una probabilidad, si el resultado es favorable, se selecciona al más apto, de lo contrario se selecciona al menos apto. El valor de la probabilidad permanece fijo en todo momento y debe permanecer en el intervalo $0.5 <$ probabilidad ≤ 1.

Mediante la selección por torneo (en la versión determinística) se garantiza que el mejor individuo será seleccionado r veces (tamaño del torneo). La técnica es eficiente y fácil de implementar, usa comparaciones directas, por lo que no requiere aplicar una función escalar. En la versión determinística, se puede introducir una presión de selección alta, donde los individuos menos aptos no pueden sobrevivir. Puede regularse la presión de selección al variar el tamaño del torneo.

En el trabajo de Carretero y Xhafa, (2006) aplican diversos operadores de selección: selección al azar (selecciona los individuos bajo una función aleatoria uniforme), mejor selección (selecciona al individuo con mejor aptitud) y selección proporcional (cada individuo se selecciona con base en la probabilidad proporcional).

III.6.5. Elitismo

El elitismo se enfoca en el problema de pérdida de buenas soluciones durante el proceso de optimización provocado por los efectos del azar. Aunque ya existen muchos operadores de selección (§III.6.4), no garantizan la supervivencia del mejor cromosoma durante todas las iteraciones del AG. El operador de elitismo es una extensión del proceso de selección, este operador garantiza la supervivencia del cromosoma más apto. Dicho operador es simple de aplicar (Eiben y Smith, 2003), ya que solamente se guarda y copia el mejor cromosoma a la siguiente generación (método que se aplica en el presente trabajo). En términos de calendarización, significa que la longitud del calendario generado por el cromosoma con mayor aptitud es de menor longitud o igual a la longitud del calendario a partir de los cromosomas iníciales (Sinnen, 2007).

III.6.6. Cruzamiento

Los operadores de cruzamiento y mutación son las herramientas con las cuales el AG explora el espacio de búsqueda (todas las posibles soluciones validas) del problema. Con dichos operadores se crean cromosomas nuevos a partir de los ya existentes, al igual que los cromosomas de donde descienden, deben ser apropiados (soluciones válidas) o de lo contrario, el espacio de búsqueda se extiende fuera de la región apropiada (soluciones no válidas) del problema. El operador de cruza es el más significativo porque sigue el principio de evolución y por su aportación al explorar con mayor capacidad el espacio de búsqueda. Este operador da lugar a que se generen nuevos cromosomas (descendientes) por recombinación de otros cromosomas (padres) ya existentes y que se toman al azar (utilizando probabilidades altas de 0.8 o más), con ello, los cromosomas descendientes heredan el material genético (parte de las soluciones) de sus ancestros (Eiben y Smith, 2003). En un inicio, las técnicas de cruza se adaptaron a la representación de cadenas binarias (Holland, 1975), pero en la actualidad existen diferentes técnicas que se generalizan a alfabetos de cardinalidad (conjuntos de símbolos) mayor tomando en cuenta algunos casos que requieren ciertas modificaciones de acuerdo a las circunstancias del problema. Existen varios operadores de cruzamiento:

1. Cruzamientos básicos: generalmente se aplica a la representación binaria; sin embargo, se puede generalizar a alfabetos de mayor cardinalidad, aunque en ciertos casos requieren modificaciones: Cruza de un Punto (Goldberg, 1991), Cruza de dos Puntos (Goldberg, 1991), Cruza Uniforme (Goldberg, 1991), Cruza Acentuada (Eiben y Smith, 2003).
2. Cruzamiento para permutaciones: consiste básicamente en usar cadenas de enteros para realizar permutaciones: *Order Crossover* (*OX*) (Abraham *et al.*, 2000), *Partially Mapped Crossover* (*PMX*) (Abraham *et al.*, 2000), *Position Based Crossover* (*PBX*) (Eiben y Smith, 2003), *Order Based Crossover* (*OBX*) (Eiben y Smith, 2003), *Cycle Crossover* (*CX*) (Abraham *et al.*, 2000).
3. En Eiben y Smith, (2003) se describen los operadores de cruzamiento para representación real: Cruza Simple, Cruza de dos puntos, Cruza Uniforme, Cruza Intermedia, Cruza Aritmética Simple, Cruza Aritmética Total, *Simulated Binary Crossover* (*SBX*).

En el trabajo de Aceves, (2003) se presenta un estudio de operadores genéticos para un problema de calendarización multi-objetivo. Realizan un análisis experimental de las combinaciones de cuatro operadores de cruzamiento (*OBX* (*Order Based Crossover*), *PPX* (*Precedence Preservative Crossover*), *OSX* (*One Segment Crossover*) y *Twopoint* (*Two Point Crossover*)) y tres de mutación (*Insert*, *Swap* y *Switch*, (§III.6.7) con el fin de encontrar la combinación que genere mejores soluciones. Los análisis se hacen sobre tres algoritmos genéticos distintos (Algoritmo Genético Multi-Objetivo (*Multi-Objective Genetic Algorithm*), Algoritmo Genético de Ordenamiento No-dominado (*Non-dominated Sorting Genetic Algorithm*) y Algoritmo Evolutivo con fortaleza de Pareto (*Strength Pareto Evolutionary Algorithm*), ver §III.4) distintos a fin de demostrar que la supremacía de la combinación ganadora de operadores genéticos sea independiente del algoritmo. Adicionalmente, toman dos casos de tamaño distinto para evaluar la forma en que el tamaño del problema afecta el comportamiento de los operadores genéticos. Concluyen con base en los experimentos que la combinación del operador de cruzamiento OBX con

mutación Insert genera el mayor promedio relativo de soluciones no dominadas. Los operadores Switch, Osx y TwoPoint son los operadores con peores resultados.

Por razones de uso en el presente trabajo, a continuación se describe el operador OBX (*Order Based Crossover*): como su nombre lo indica, este operador cuida de conservar el orden de los bloques que se van formando durante las iteraciones del AG (Gen y Cheng, 2000). Comúnmente (en vectores unidimensionales) este operador trabaja de la siguiente manera: con ayuda de una máscara de ceros y unos generados al azar, se eligen un conjunto de posiciones no necesariamente consecutivas (ceros o unos), enseguida se genera un *Hijo 1* borrando del *Padre 1* todos los valores, excepto aquellos que se hayan elegido en el paso anterior, enseguida se borran del *Padre 2* los valores seleccionados, ya que la secuencia resultante de valores se usará para completar al *Hijo 1*. Como último paso, se colocan en el *Hijo 1* los valores faltantes de izquierda a derecha de acuerdo a la secuencia en el *Padre 2*. En el trabajo de Carretero y Xhafa, (2006) aplican los operadores: *Partially Matched Crossover* (*PMX*), *Cycle Crossover* (*CX*) y *order Crossover* (*OX*).

Aplicar el operador OBX a la representación de dos dimensiones no requiere de conocimientos adicionales, se aplica tal cual. A continuación se explica el método: se genera una máscara de ceros y unos, se eligen al azar un conjunto de columnas (no necesariamente consecutivas) señaladas por la máscara con ceros o unos (Figura 5-a). Se genera un *Hijo 1* (Figura 5-c) que hereda las columnas del *Padre 1* (Figura 5-a), excepto aquellas que no se han elegido en el paso anterior, en seguida, se borran del *Padre 2* (Figura 5-b) los valores que se encuentran en el *Hijo 1*. Finalmente, se colocan en el *Hijo 1* los valores faltantes de izquierda a derecha de acuerdo a la secuencia en el *Padre 2*. La misma estrategia se puede aplicar por filas.

Al aplicar el operador OBX a los AGs del presente trabajo: por cada dos padres se generan dos hijos. Antes de iniciar la operación de cruzamiento se revuelve la población de padres con la intención de que cada padre tenga la misma oportunidad de ser recombinado con otro padre, una vez realizado esto, se toman dos padres contiguos para generar dos

descendientes. Cada nuevo descendiente se genera a partir de una máscara distinta, ello con la intención de abarcar un mayor espacio en la búsqueda de soluciones.

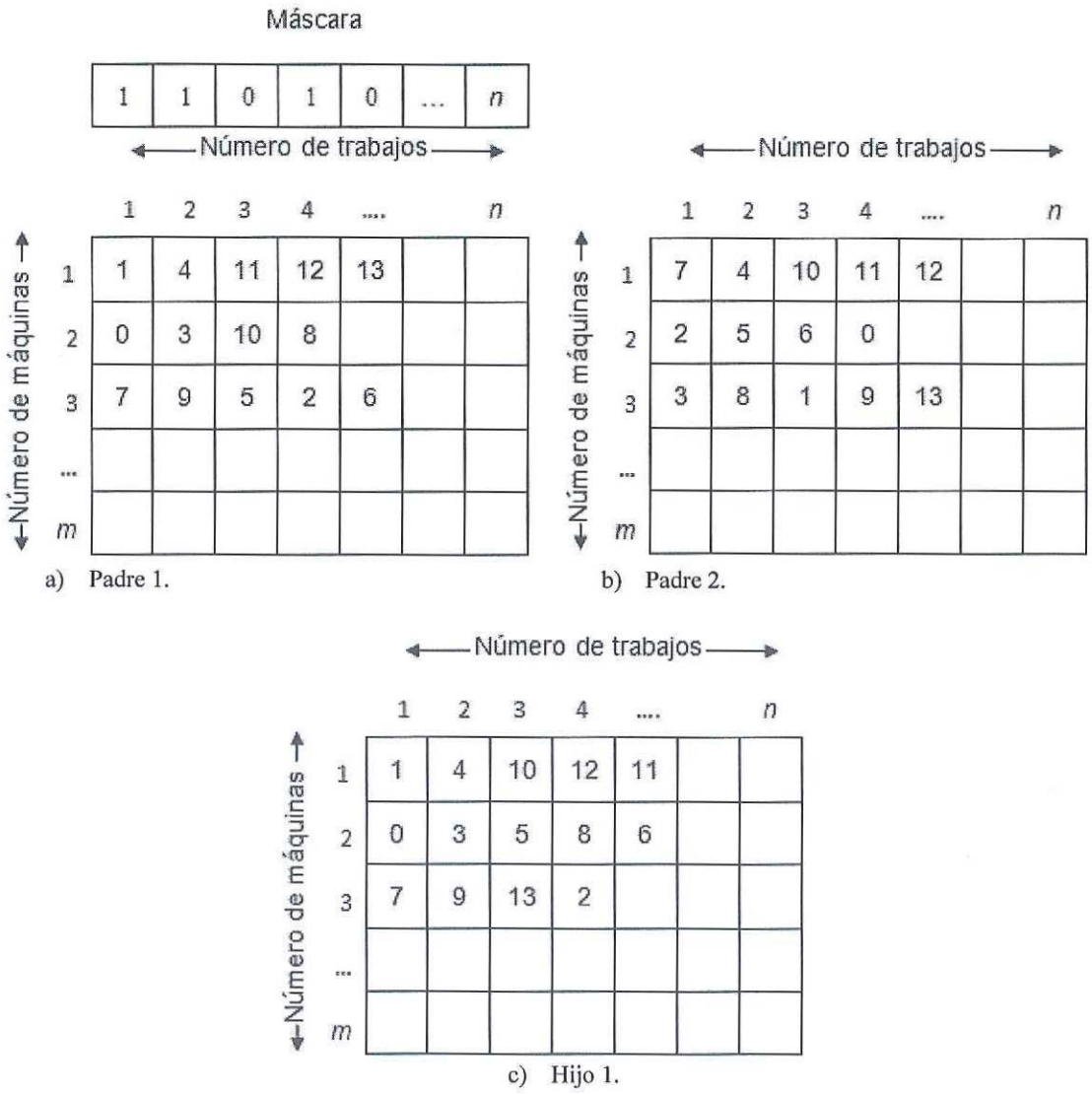


Figura 5. Operador de cruzamiento basado en el orden (OBX). En a) se genera una máscara binaria al azar para identificar a los elementos (columnas) del Padre 1 que se heredan al Hijo 1. En b) se identifican los elementos que aun no se encuentran en el Hijo 1 y se copian a este ultimo respetando el orden. Finalmente, en c) el Hijo 1 cuenta con elementos heredados de ambos padres y sin elementos repetidos.

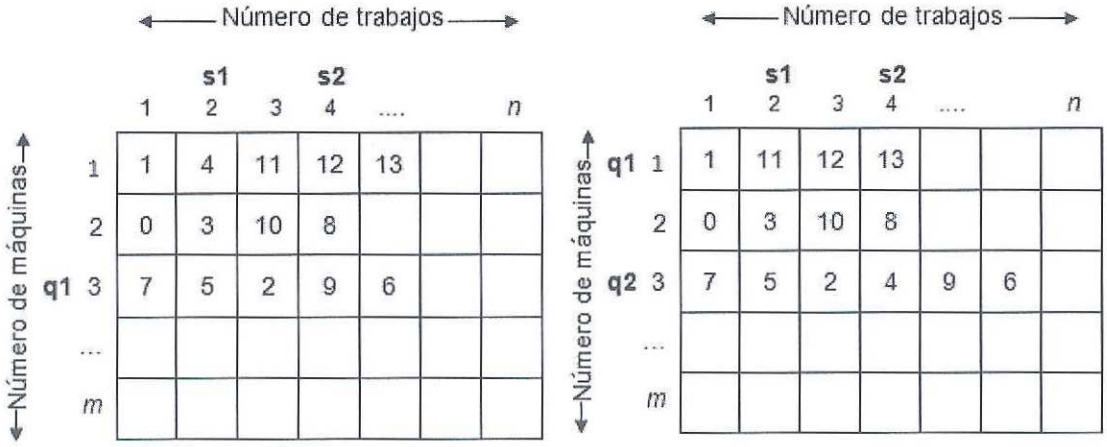
III.6.7. Mutación

La mutación es un nombre genérico dado a los operadores que generan pequeños cambios en un hijo. La forma que toman depende del tipo de la representación que se utilice. El operador de mutación se considera como un operador secundario (Coello, 2001) y como consecuencia, se aplica con una probabilidad baja (0.2 o menor) en comparación con la probabilidad del operador de cruzamiento. Su principal objetivo es evitar la convergencia prematura a un óptimo local. Al igual que los operadores de cruzamiento, existen diversas maneras de llevar a cabo este operador. En el presente trabajo se aplican los operadores estándar para vectores, todos ellos se explican en Gen y Cheng, (1997):

1. *Insert* (*Shift*): se seleccionan dos posiciones al azar $s1$ y $s2$, si $s1 < s2$, entonces el elemento correspondiente a $s1$ se coloca en la posición $s2$ y todos los elementos desde $s1 + 1$ hasta $s2$ se recorren una posición hacia $s1$, pero si $s1 > s2$, entonces la operación de corrimiento se realiza hacia $s2$.
2. *Swap* (*arbitrary two-job change*): se seleccionan dos posiciones al azar y sus elementos se intercambian.
3. *Switch* (*Adjacent two-job change*): se selecciona una posición $s1$ al azar para intercambiarlo con el elemento de su sucesor inmediato a la derecha. Cuando se llega a seleccionar la posición n, entonces se intercambia su elemento con el de la posición 1.

Para el caso de dos dimensiones también se pueden aplicar los operadores de mutación mencionados en la sección anterior. Debido a la doble dimensión (con base en la Figura 5-a), existen dos posibilidades para cada operador:

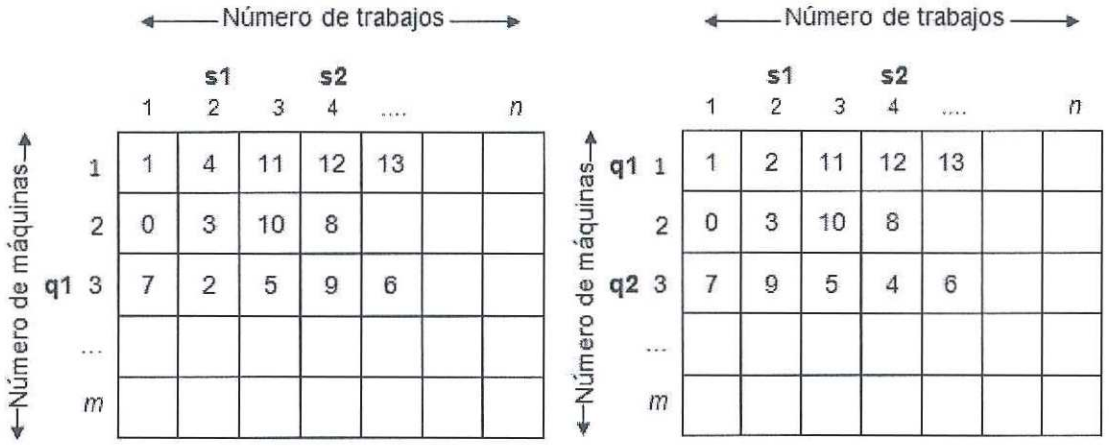
1. *Insert*: a) Seleccionar un renglón $q1$ al azar y aplicar el operador Insert sobre $q1$ (Figura 6-a). b) seleccionar dos renglones $q1$ y $q2$ al azar, seleccionar una posición $s1$ de $q1$. Insertar el elemento de $s1$ en una posición $s2$ seleccionada al azar en $q2$ (Figura 6-b).
2. *Swap*: a) seleccionar un renglón $q1$ al azar y aplicar el operador Swap sobre $q1$ (Figura 7-a). b) seleccionar dos renglones $q1$ y $q2$ al azar, seleccionar una posición $s1$ de $q1$ y $s2$ de $q2$ e intercambiar los elementos de $s1$ y $s2$ (Figura 7-b).
3. *Switch*: a) seleccionar un renglón $q1$ al azar y aplicar el operador Switch sobre $q1$ (Figura 8-a). b) seleccionar dos renglones $q1$ y $q2$ al azar, seleccionar una posición $s1$ (misma posición para ambos), intercambiar los elementos de $q1$ y $q2$ que se encuentran en la posición $s1$ (Figura 8-b).



a) Insert estándar. b) Insert entre filas.

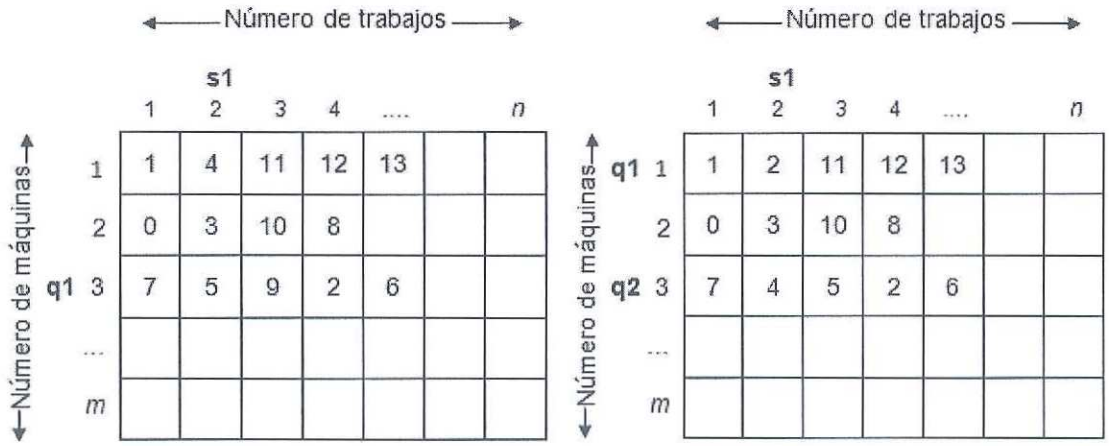
Figura 6. Operador de mutación Insert. En a) se aplica el operador Insert a una fila seleccionada (q1), similar al descrito en la literatura. En b) se realiza una expansión del operador Insert seleccionando dos filas (q1 y q2) que hacen las veces de las colas locales.

En el trabajo de Carretero y Xhafa, (2006) aplican los operadores: *Move* (mueve un trabajo de un recurso diferente) y *Swap* (se realiza un intercambio de trabajos entre máquinas, pero también toman en cuenta el intercambio entre dos trabajos asignados a una máquina).



a) Swap estándar. b) Swap entre filas.

Figura 7. Operador de mutación Swap. En a) se aplica el operador Swap a una fila seleccionada (q1), s1 y s2 indican los elementos a intercambiar. En b) La expansión del operador Swap se realiza seleccionando dos filas (q1 y q2) que hacen las veces de las colas locales; s1 y s2 indican los elementos a intercambiar.



a) Switch estándar. b) Switch entre filas.

Figura 8. Operador de mutación Switch. En a) se aplica el operador Switch a una fila seleccionada (q1), s1 indica el elemento a intercambiar con su adyacente. En b) La expansión del operador Swap se realiza seleccionando dos filas (q1 y q2) que hacen las veces de las colas locales; s1 indica los elementos a intercambiar.

Capítulo IV

Experimentos y resultados

Los experimentos finales se diseñaron reuniendo la mayor cantidad de información del problema que se investiga (§II.1). El mejor diseño de ellos asegura que los resultados ayuden a realizar un análisis eficiente y objetivo dando lugar a deducciones válidas y robustas.

Se puede tener combinaciones múltiples entre el problema y el método para resolverlo. Con la idea de aprovechar de forma eficiente el tiempo, utilizar los recursos al máximo y evitar repetir experimentos por descuido, se han simplificado en lo posible. Bajo la metodología que se usa, los experimentos pueden repetirse con los mismos datos y condiciones.

IV.1. Carga de trabajo utilizada

La carga de trabajo es el conjunto de trabajos que se utiliza en el presente trabajo. Cada carga de trabajo tiene diferentes características por el tipo de trabajos que contiene (distintos centros de cómputo, zona horaria, tipo de aplicaciones, número de procesadores que solicitan los trabajos, entre otros). El análisis estadístico de la carga de trabajo es importante para interpretar los resultados de las simulaciones y para sustentarlos. Algunas estrategias de calendarización generan buenos resultados con trabajos grandes y otros con trabajos pequeños. Las características de la carga de trabajo afectan el desempeño del Grid; esto significa que si no se conocen las características de la carga de trabajo utilizada, no se puede decir si los resultados de las simulaciones son de calidad (si al comparar dichos resultados son mejores en relación a otros).

Los experimentos se basan en una carga de trabajo semi-sintética (se llama así porque contiene trabajos de cinco registros reales de centros de cómputo de alto desempeño en distintos continentes). Cuatro cargas pertenecen a los siguientes centros (Feitelson, 2005): Cornell Theory Center, High Performance Computing Center North, Swedish Royal Institute of Technology y Los Alamos National Lab. Una proviene del *Grid Workload Archive* (*GWA*), (Anoep *et al.*, 2007): The Advanced School for Computing and Imaging. Estos registros se incluyeron en la carga de trabajo debido a que contienen información importante acerca de los trabajos como: número de usuario, tiempo de ejecución, tiempo solicitado, número de procesadores asignados, tiempo de llegada, tipo de trabajos, entre otros; en total son 18 datos que proporcionan información para cada trabajo. Adicionalmente incluye información relacionada al centro de cómputo de origen como: máquina, número de procesadores, zona horaria, periodo del registro, entre otros datos.

La carga se encuentra almacenada en un archivo que sigue el estándar de cargas de trabajo (*Standard Workload Format*, *SWF*) (Feitelson, 2005), contiene 30,000 trabajos (equivalente a 7 días de trabajos propuestos) y con ellos se pueden realizar 30 experimentos con lotes de 1,000 trabajos cada uno. La zona horaria después de la normalización de los registros originales es GMT-7, la mínima encontrada. El filtro que se aplica a la carga permite trabajos con tamaño (número de procesadores que solicita) mayor o igual que 1 y menor o igual que 32.

IV.1.1. Estadística de la carga de trabajo

El análisis estadístico de la carga de trabajo ayuda a sustentar e interpretar los resultados de los experimentos simulados. Conocer los trabajos sometidos por hora, semana, tamaño y los recursos consumidos, contribuye a decidir si los resultados obtenidos de las simulaciones son de calidad baja o alta. Por ejemplo, hay estrategias (§I.6.1 y §I.6.2) enfocadas (especializadas) a mejorar algún criterio bajo ciertas características de los trabajos, algunas son buenas trabajando con trabajos de tamaño grande y otras no lo son. Como consecuencia, el desempeño de un Grid depende de las características de los trabajos, estrategias y parámetros de los algoritmos.

Respecto a la carga que se utiliza en el presente trabajo, se observa en la Figura 9 que no existe una hora durante el día en donde el arribo de los trabajos sea intenso. Ello se debe a que se toman en cuenta registros provenientes de diferentes zonas horarias y al combinarlos, la distribución de llegadas se torna uniforme.

También es interesante observar el número promedio de trabajos sometidos por cada día de la semana (Figura 10). Una gran parte de los trabajos se someten el día cinco (viernes), el resto de los días se someten casi de manera uniforme (1: lunes,..., 7: domingo).

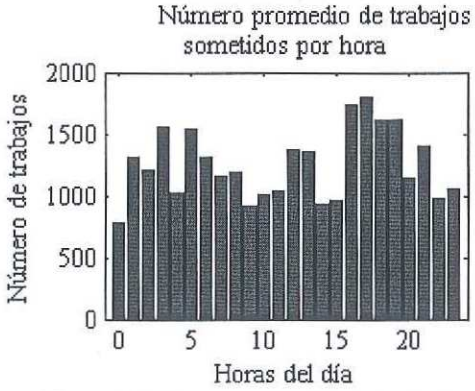


Figura 9. Número promedio de trabajos sometidos por cada hora durante el día.

Figura 10. Número promedio de trabajos sometidos por cada día durante la semana.

Otra característica importante para analizar acerca de los trabajos, es la cantidad de recursos que solicitan. En la Figura 11 se observa que la distribución de recursos consumidos por hora del día es distinta a la distribución de trabajos sometidos por hora del día. Obsérvese en la hora 20 de la Figura 9 que llega una cantidad de trabajos por encima del promedio y los recursos que consumen se encuentran por debajo del promedio (Figura 11). Con ello se puede afirmar que la cantidad de trabajos no necesariamente define la cantidad de recursos consumidos. El momento en que solicitan recursos con mayor intensidad corresponde a las 10 horas. En la Figura 12 y Figura 10 se observa el mismo patrón de las figuras anteriores. La cantidad promedio de recursos consumidos por cada día

de la semana no proporciona idea acerca de los trabajos sometidos. Sin embargo, se puede ver que semanalmente se conserva una distribución más uniforme (Figura 12) que por día.

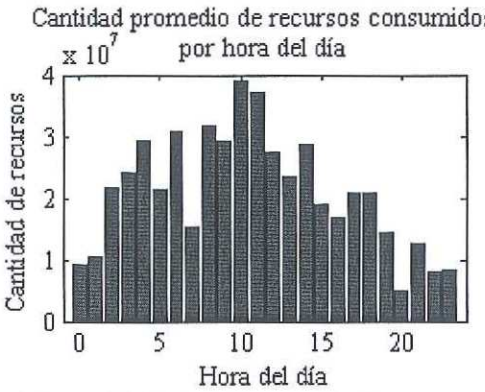


Figura 11. Cantidad promedio de recursos consumidos por cada hora del día.

Figura 12. Cantidad promedio de recursos consumidos por cada día de la semana.

En la Figura 13 se observa la irregularidad de los trabajos sometidos por cada usuario. Los usuarios con identificadores 600s y 800s muestran mayor actividad con hasta 14,000 trabajos propuestos por usuario en promedio; el resto de los usuarios envían en menor proporción (menor a 1,000) y con uniformidad (Figura 14). La Figura 15 muestra los mismos trabajos en forma ordenada; la mayoría de los usuarios someten 100 o menos trabajos.

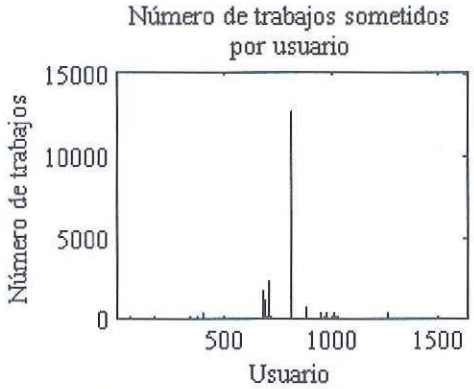


Figura 13. Número de trabajos sometidos por usuario.

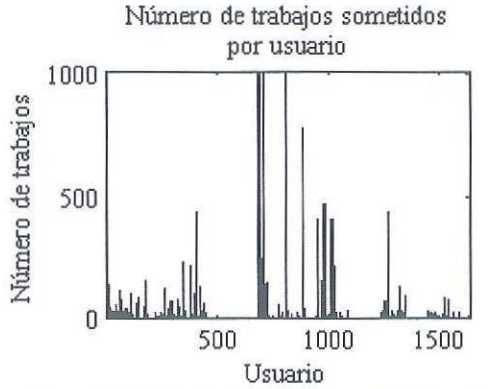


Figura 14. Número de trabajos sometidos por usuario. Usuarios con menos de 1,000 trabajos.

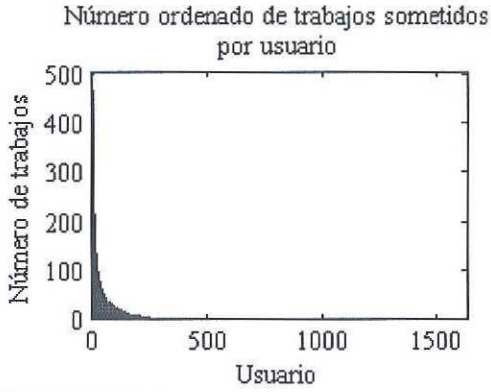


Figura 15. Número ordenado de trabajos sometidos por usuario.

Con respecto al tamaño de los trabajos sometidos, se observa que la mayoría son menor a 16, siendo los de tamaño 1 (secuenciales) y 2 los que predominan (Figura 16). Es interesante ver en la Figura 17 que los tamaños siguen el patrón con base en la potencia de dos, es decir, destacan los tamaños 1, 2, 4, 8, 16 y 32. Es curioso notar que no se someten trabajos con tamaño 30, pues no es común o no guarda alguna relación con la potencia de 2.

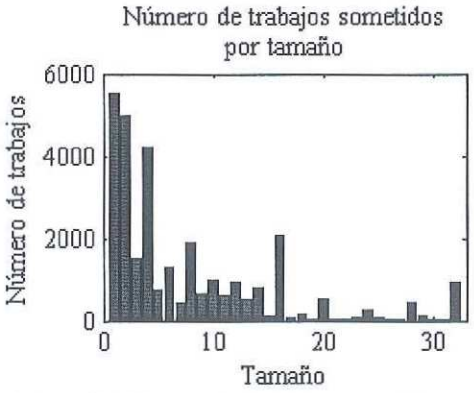


Figura 16. Número de trabajos sometidos por tamaño.

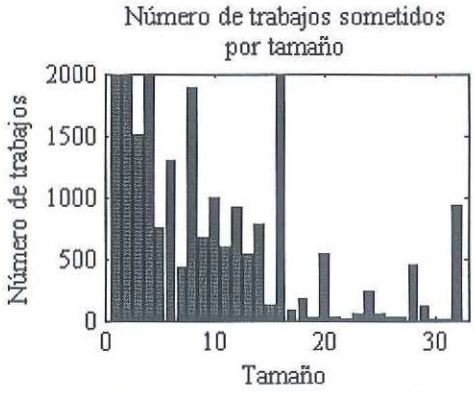


Figura 17. Número de trabajos sometidos por tamaño, vista detallada de los tamaños con menos de 50,000 trabajos.

En la Figura 18 se observa que la mayoría de los trabajos sometidos tienen un tiempo de ejecución menor o igual a una hora. Posteriormente, una vista detallada de los trabajos para el mismo caso (Figura 19) permite visualizar un patrón; conforme incrementa el tiempo de ejecución, disminuye el número de trabajos sometidos. No se debe pasar por

alto el incremento de trabajos que utilizan los recursos entre 10 y 12 horas. Finalmente, se observa que muy pocos trabajos tienen un tiempo alto de ejecución en el intervalo de 25 a 125 horas.

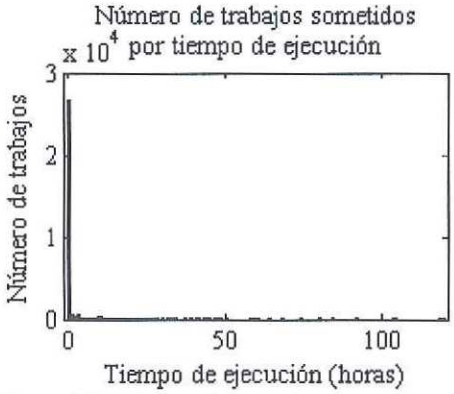


Figura 18. Número de trabajos sometidos por tiempo de ejecución. Vista que contempla todos los trabajos.

Figura 19. Número de trabajos sometidos por tiempo de ejecución. Vista que contempla cantidad de trabajos menor a 1000.

El tipo común de trabajos que predomina, es aquel que cuenta con un tiempo de procesamiento de hasta 20 horas, aunque los trabajos pequeños tienden a acumularse en los tiempos de menor ejecución (Figura 20). Nuevamente, es interesante notar que los trabajos con potencia de dos son los que cuentan con mayor tiempo de ejecución. Aún más, los trabajos que prácticamente son secuenciales (tamaño 1) son los que utilizan más horas de ejecución (por arriba de 100).

Únicamente los trabajos con tamaños de 1 y 4 son las que tienen tiempos de procesamiento mayores a 100 horas. En cambio, los trabajos con tamaños 1, 2, 4, 6, 8, 16 y 24 tienen tiempos solicitados de más de 100 horas. También se puede apreciar el hecho que los usuarios solicitan más tiempo del necesario para sus trabajos; los trabajos se encuentran más dispersos (ver Figura 20).

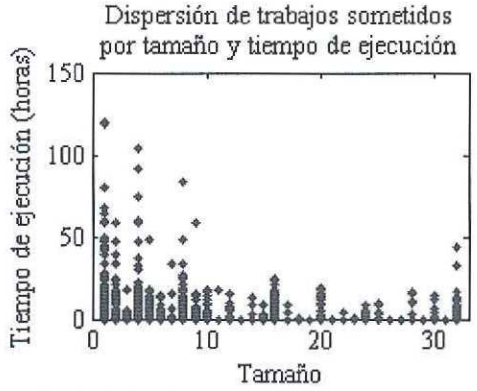


Figura 20. Dispersión de trabajos sometidos por tamaño y tiempo de ejecución.

Por lo regular, los usuarios tienden a solicitar tiempo de procesamiento mayor al requerido por su aplicación (trabajo) de lo necesario, pues cuidan o se aseguran que no sea suspendido por falta de tiempo. Similarmente, los recursos solicitados también son mayores.

La Figura 21 ayuda a comprender la diferencia entre los recursos solicitados y los verdaderamente usados; en ella se observa relaciones próximas a 1500 veces, es decir, los usuarios quieren asegurarse (quizás por falta de conocimiento) que su trabajo termine de ejecutarse sin interrupciones y para ello solicitan casi 1500 veces más recursos de los necesarios. Se puede notar una relación curiosa entre el tamaño del trabajo (Figura 16) con la relación de recursos solicitados y consumidos; los trabajos de mayor tamaño presentan mayor cantidad de malas estimaciones. También, la mayor imprecisión se nota en los trabajos con tamaño que es potencia de dos, con gran certidumbre, se debe a que los usuarios tengan conocimientos preliminares acerca de las configuraciones de sistemas que suelen tener arreglos de procesadores con dicha potencia.

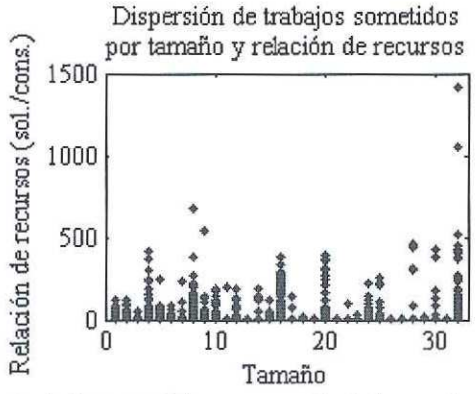


Figura 21. Dispersión de trabajos sometidos por tamaño (número de procesadores) y relación de recursos (solicitados/consumidos).

La Figura 22 presenta una relación entre el tiempo de ejecución por los trabajos sometidos y los recursos consumidos; los trabajos con menor tiempo de ejecución requieren mayor cantidad de recursos. Contrario a lo que se puede pensar en primera instancia, aunque los trabajos con tiempo de ejecución menor a una hora sean los más comunes, no son los que consumen la mayor cantidad de recursos. Similarmente se aplica la idea para los trabajos con dos horas de tiempo de ejecución.

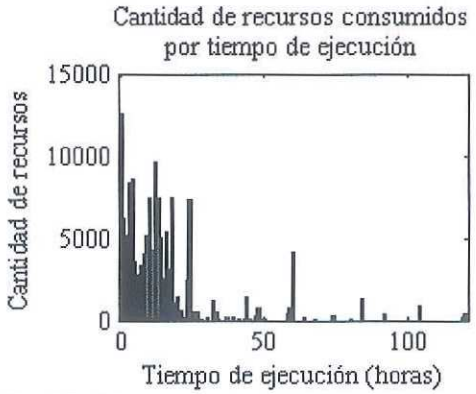


Figura 22. Cantidad de recursos consumidos por tiempo de ejecución.

Como se observa en el transcurso del presente análisis estadístico de la carga, los trabajos más frecuentes son de tamaño uno y con potencia de dos. Como consecuencia, en la Figura 23 se aprecia que dichos trabajos son los que más consumen recursos.

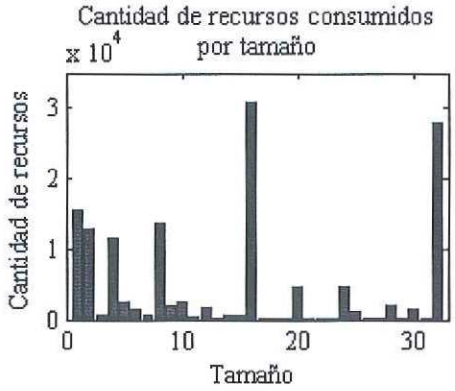


Figura 23. Cantidad de recursos consumidos por tamaño.

Es importante tomar en cuenta los trabajos que no terminan su ejecución por alguna razón (por lo general, algún error), pero que consumen recursos; tales trabajos se encuentran en la carga sintética porque no se filtran. El tipo de error que se muestra en la Figura 24 es 0, es decir, alrededor de 14,000 trabajos no terminan su procesamiento por fallo y pertenecen a distintos usuarios como lo muestra la Figura 25. También se aprecia en esta última que existen usuarios (con identificadores 800 - 1200) que cuentan con una proporción casi imperceptible de errores por fallo en relación a los trabajos que someten.

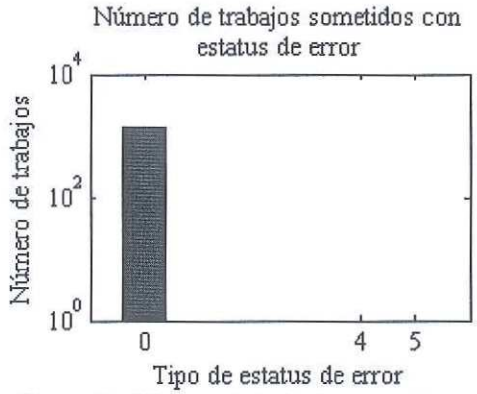


Figura 24. Número de trabajos sometidos con estatus de error por tipo.

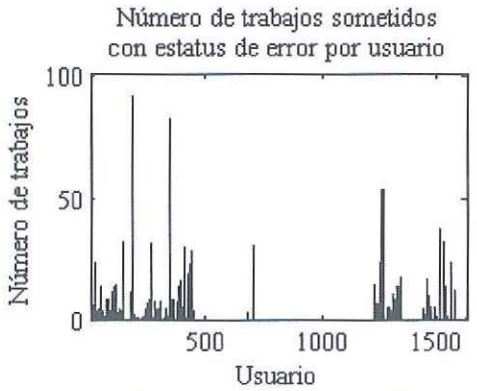


Figura 25. Número de trabajos sometidos con estatus de error por usuario.

IV.2. Configuración de un Grid

Una parte que se debe configurar, son los parámetros propios del experimento; en la (Tabla I). Se efectúan 30 experimentos con 1,000 trabajos cada uno. El promedio de los 30 experimentos ayuda a tener mayor confianza en los resultados obtenidos. Los 30,000 trabajos representan aproximadamente una semana de trabajos propuestos al Grid. Se utiliza una estructura de nivel jerárquico con dos niveles, por lo tanto se requieren estrategias de asignación y de calendarización local. Los trabajos se toman de la cola de espera global de acuerdo al orden en que van llegando (*FIFO*) y como se desconocen los tiempos de ejecución, se toma el tiempo solicitado por el usuario.

Por lo general, las máquinas contienen en su interior una configuración de procesadores con potencia de dos, se puede constatar con el análisis de carga, por tal razón se decide tomar la configuración que se muestra en la Tabla II. El orden de procesadores en cada máquina se toma con base en la carga sintética a la cual se le ha aplicado un filtro que acepta trabajos con tamaño mayor que uno y menor que 32; por ende, se permite procesar un trabajo con tamaño máximo 32. No se toman los arreglos de máquinas que proponen las cinco cargas de trabajo originales ya que son muy grandes. Con la presente configuración de Grid se pretende simular la situación actual de los centros de cómputo donde existen problemas reales con grandes colas de espera por cada máquina. No existiría mayor problema al utilizar una configuración de Grid con una carga de trabajo que no genere colas de espera. Si se diera esto último, sería trivial encontrar la mejor calendarización con alguna estrategia clásica de calendarización, pues siempre habría recursos disponibles para todos los trabajos.

Tabla I. Parámetros de los experimentos.

Parámetros de los experimentos	
Número de experimentos	30
Número de trabajos por experimento	1,000
Niveles de calendarización	2
Selección de trabajos de la cola global de espera	FIFO
Estrategia de asignación	Ver Tabla V
Calendarización local	Backfilling-EASY-FirstFit
Tiempo de ejecución por cada trabajo	Tiempo solicitado por el usuario.

Tabla II. Configuración de un Grid para efectuar los experimentos.

Configuración del Grid	
Tamaño máximo de los trabajos permitido	32
Número de máquinas	11
Tamaño de la máquina 0	4
Tamaño de la máquina 1	4
Tamaño de la máquina 2	4
Tamaño de la máquina 3	4
Tamaño de la máquina 4	8
Tamaño de la máquina 5	8
Tamaño de la máquina 6	8
Tamaño de la máquina 7	16
Tamaño de la máquina 8	16
Tamaño de la máquina 9	32
Tamaño de la máquina 10	32

IV.3. Parámetros de los algoritmos

Para evaluar los resultados entre los operadores genéticos de cruzamiento y de mutación (§III.6.6 y §III.6.7) se crean tres combinaciones distintas. Son tres combinaciones porque se toma el operador de cruzamiento OBX con cada uno de los operadores de mutación Insert, Swap y Switch. En la Tabla III se muestran los nombres de los algoritmos genéticos (C1, C2 y C3) que consideran múltiples criterios y que aplican el método de agregación bajo el operador *OWA* descrito en §III.6.3.

Tabla III. Nombre de los algoritmos que consideran varios criterios.

Nombre	Significado
C1	Cruzamiento OBX con mutación Q-Insert
C2	Cruzamiento OBX con mutación Q-Swap
C3	Cruzamiento OBX con mutación Q-Switch

En la Tabla IV se muestran los parámetros tomados Aceves, (2003) para todas las combinaciones de operadores genéticos.

Tabla IV. Parámetros de los algoritmos genéticos.

Nombre	Significado
Tamaño de la población	100
Probabilidad de cruzamiento	0.9
Probabilidad de mutación	0.01
Criterio de paro	10 veces sin cambio en el resultado de la función objetivo.
Selección	Torneo binario determinístico
Reemplazo	Generacional

Para evaluar los resultados generados por los operadores genéticos cuando se contempla el caso multi-criterio usando el método de agregación se realizan simulaciones con otras estrategias (*Min_CT, Min_ST, Min_TA y Min_WT*, definidos en §I.6.1) especializadas en optimizar un solo criterio; en la Tabla V se nombran como A1, A2, A3 y A4. Similarmente, se efectúan simulaciones con AGs que consideran un solo objetivo nombrados como B1, B2, B3 y B4 (en la Tabla III se muestra la configuración con los operadores y los parámetros en la Tabla IV), enfocados en mejorar cada criterio. Finalmente, la Tabla V muestra el nombre de todos los algoritmos generados (once en total) y los criterios en que se enfoca cada uno. Para cada simulación se utiliza la misma carga de trabajo, con el mismo lote de trabajos. Los AGs que consideran un solo objetivo y los que consideran múltiples criterios parten de la misma población inicial, también usan los mismos parámetros.

Aunque en §III.6.7 se describen seis operadores de mutación para la representación en dos dimensiones, solamente se realizan experimentos finales con el operador Q-Insert en los algoritmos genéticos que consideran un solo objetivo. Ello se debe a que en experimentos preliminares (no detallados en el presente trabajo), el resto de los operadores mostraron desempeño menor en promedio al operador Q-Insert. Es importante mencionar que la diferencia de desempeño entre el operador Q-Insert y el resto de los operadores no fue de gran magnitud, sin embargo, los resultados basados en experimentos preliminares coinciden con la conclusión de Aceves, (2003), en donde se afirma que la mejor combinación del operador de cruzamiento OBX (§III.6.6) y el operador de mutación Insert (§III.6.7) resulta ser la mejor. A pesar de ello, en los experimentos finales para el caso multi-criterio usando el método de agregación se toman en cuenta los operadores Insert,

Swap y Switch con el fin de notar la diferencia en los resultados bajo la representación de dos dimensiones (§III.6.1).

Tabla V. Nombre de los once algoritmos para realizar los experimentos.

Nombre	Significado				
	Estrategia de asignación	Estrategia de Calendarización	Operador de cruzamiento	Operador de mutación	Criterios contemplados
A1	Min_CT	BEFF			Cmax
A2	Min_ST	BEFF			Slowdown
A3	Min_TA	BEFF			Tournaround
A4	Min_WT	BEFF			Wait time
B1	AG	BEFF	OBX	Q-Insert	Cmax
B2	AG	BEFF	OBX	Q-Insert	Slowdown
B3	AG	BEFF	OBX	Q-Insert	Tournaround
B4	AG	BEFF	OBX	Q-Insert	Wait time
C1	AG-MO	BEFF	OBX	Q-Insert	TODOS
C2	AG-MO	BEFF	OBX	Q-Swap	TODOS
C3	AG-MO	BEFF	OBX	Q-Switch	TODOS

La función de los AGs es asignar recursos a trabajos, es decir, su función es similar a la que realizan las estrategias de asignación bien definidas en la literatura para calendarización (*Min_CT, Min_ST, Min_TA y Min_WT*, ver §I.6.1). La ventaja que presentan los AGs en relación a dichas estrategias, es que tienen la capacidad de efectuar el intercambio de trabajos entre máquinas durante su ejecución, en cambio, una vez que las otras estrategias asigna recursos a los trabajos ya no los pueden reasignar. La diferencia entre el AG que contempla el caso multi-criterio y el que considera un solo objetivo (ambos asignan recursos, aplican los mismos operadores y parámetros), es que el primero encuentra soluciones considerando varios criterios simultáneamente que están en conflicto aplicando el método de agregación (§III.2.1) por medio del operador *OWA* (§III.6.3).

IV.4. Resultados experimentales

El desempeño alude a los resultados obtenidos por los algoritmos con que se efectúan experimentos y la calidad de dichos resultados. La calidad es el valor obtenido por cada algoritmo y que se compara con los valores de los otros algoritmos. En algún sentido, *"los algoritmos compiten entre sí por tener mejores resultados"*.

En la Figura 26 se ilustra el desempeño de los Algoritmos C1, C2 y C3 durante los 30 experimentos. Debido a que la meta es minimizar los criterios (métricas) agregados por el operador OWA; los criterios se normalizan de acuerdo a la Ecuación (25) descrita en §III.6.3, la calidad de los resultados por cada algoritmo oscila en el intervalo [0, 1]. Entre más próxima se encuentra un valor al 0, la calidad es mayor y se está más próxima de lograr una mejor solución (calendarización más próxima a la óptima de los trabajos). Por ejemplo, en el experimento 10 el Algoritmo C3 muestra mayor desempeño que los Algoritmos C2 y C3 porque su solución es de mayor calidad. Por otra parte, en la misma figura se observa que ante los mismos parámetros (§IV.3) y mismo lote de trabajos (§IV.1), todos los algoritmos tienen un comportamiento equivalente y no existe un marcado dominio por alguno, es decir, en promedio tienen un desempeño similar, lo cual se corrobora en la Figura 27; donde se observa que el Algoritmo C3 logra mejor promedio aunque es prácticamente irrelevante la diferencia en relación a los Algoritmos C1 y C2, lo mismo sucede con la desviación estándar (*DS*) correspondientemente.

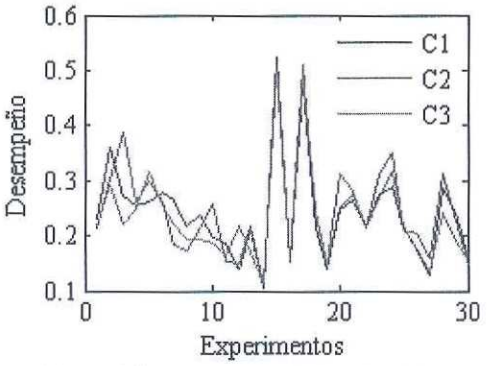


Figura 26. Desempeño de los algoritmos con enfoque multi-criterio usando el método de agregación. No hay diferencia significativa de calidad entre los Algoritmos C1, C2 y C3 en cada uno de los 30 experimentos.

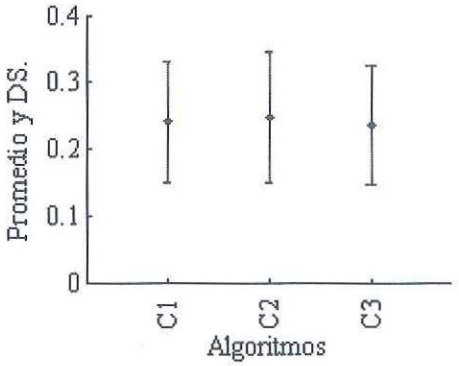


Figura 27. Desviación estandar de los calendarios generados con los algoritmos multi-criterio.

Con el análisis anterior no es posible darse cuenta de qué tan buenos o qué tan malas son las soluciones generadas por los Algoritmos CI, C2 y C3. Para emitir un juicio

al respecto, es necesario comparar los resultados de dichos algoritmos con los resultados generados por las estrategias especializadas (Algoritmos A1, A2, A3 y A4) en optimizar cada criterio; lo mismo para cada algoritmos genético (B1, B2, B3 y B4). La Tabla VI muestra un resumen de esos resultados. En el trabajo de Kurowski *et al.*, (2008) se considera que los AGs pueden superar los resultados obtenidos con estrategias clásicas para calendarización porque cuentan con métodos que les ayudan a encontrar mejores soluciones (esto se confirma con los resultados a lo largo de la presente sección). Por tal razón, los resultados de los Algoritmos A1, A2, A3 y A4 se toman como 100% y los resultados de los Algoritmos B1, B2, B3, B4 CI, C2 y C3 se toman como mejora en relación a los primeros (ver Tabla VI). Para obtener el porcentaje, primero se efectúa el promedio de los 30 resultados experimentales con cada algoritmo. La información detallada de los experimentos se presenta enseguida.

Tabla VI. Porcentaje (%) de los algoritmos evolutivos con respecto a los algoritmos para calendarizacion.

Métrica	Algoritmo de referencia 100%	Porcentaje de cada criterio con algoritmos evolutivos						
		B1	B2	B3	B4	C1	C2	C3
Longitud del calendario	A1	**18.16**	-29.36	-8.74	-1.10	11.38	12.78	12.02
Cociente de competitividad	1.38	1.11	1.74	1.44	1.38	1.23	1.20	1.20
Cociente de respuesta promedio	A2	-7.70	**96.15**	89.68	87.81	93.81	91.86	93.78
Tiempo promedio de permanencia	A3	1.45	51.84	**62.41**	63.27	61.34	59.98	61.60
Tiempo promedio de espera	A4	4.26	58.94	70.40	**71.34**	69.25	67.78	69.53

El Algoritmo A1 toma en cuenta a la estrategia *Min_CT* que reduce el tiempo de terminación de los trabajos, por ello, el promedio de sus resultados experimentales se toma como punto de referencia para determinar el porcentaje de B2 que optimiza el mismo criterio, al igual que C1, C2 y C3. En la Figura 28 se observa que el Algoritmo B1 supera al Algoritmo A1 con una diferencia del 18.16%. Similarmente, una diferencia de 11.38%, 12.78% y 12.02% a favor de los Algoritmos C1, C2 y C3 respectivamente. El Algoritmo A1 presenta la mayor desviación estándar y B1 la menor, aunque la diferencia en magnitud

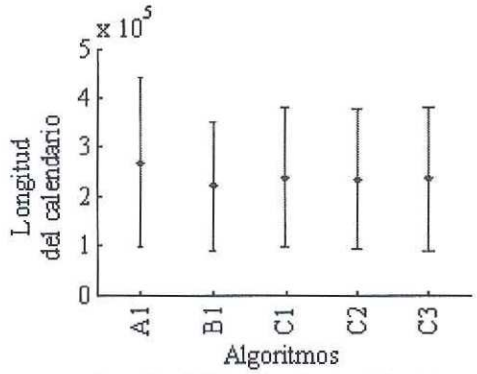


Figura 28. Longitud del calendario obtenida por los algoritmos que consideran un criterio y los algoritmos multi-criterio que usan el método de agregación.

Tabla VII. Cociente de competitividad de los algoritmos con y sin enfoque multi-criterio.

Algoritmos	Cociente de competitividad	Desviación estándar
A1	1.38	0.33
B1	1.11	0.16
C1	1.23	0.25
C2	1.15	0.21
C3	1.20	0.20

La estrategia *Min_ST* tiene como objetivo reducir la longitud del calendario al asignar el trabajo donde el inicio de su ejecución es más próximo y como consecuencia de ello, puede mejorar el cociente de respuesta promedio; como en el presente trabajo no se tiene una estrategia enfocada en optimizar el cociente de respuesta, dicha estrategia forma parte del algoritmo A2. El Algoritmo B2 también tiene la meta de minimizar el cociente de respuesta promedio. En la Tabla VIII se observa que el algoritmo A2 presenta el peor cociente de respuesta promedio; es de esperarse ya que no se especializa en minimizar dicho criterio. Por tal motivo, el desempeño de los otros algoritmos es notorio, 96.16% para B2, 93.83%, 91.88% y 93.80% para C1, C2 y C3 respectivamente. Tomando en cuenta únicamente a los algoritmos que consideran múltiples criterios, se aprecia que no existe un marcado dominio por alguno sobre sus similares (C1, C2 o C3) aunque, sí se pretende ser estricto, C2 que contempla al operador Swap presenta el peor promedio y la mayor desviación estándar. Como se puede inferir, el algoritmo evolutivo que contempla solamente un objetivo (B2) muestra el mejor promedio y la menor desviación estándar. En la Tabla VI se observa que el resultado (calendario) que minimiza el cociente de respuesta promedio no minimiza la longitud del calendario ya que el Algoritmo B2 no supera al Algoritmo A2 (el Algoritmo B2 presenta una diferencia negativa de -29.36 en relación al Algoritmo A2). Con respecto al tiempo promedio de permanencia y tiempo promedio de espera, el Algoritmo B2 supera con 51.84% y 58.94% respectivamente al Algoritmo A2.

Sin embargo, los Algoritmos C1, C2 y C3 superan al Algoritmo B2 en ambos criterios (Tabla VI).

El algoritmo A3 incluye la estrategia *Min_TA* porque tiene como objetivo minimizar el tiempo promedio de permanencia de los trabajos; el Algoritmo B3 también tiene el mismo objetivo. En la Figura 29 se observa que los Algoritmos B3, C1, C2, y C3 superan al Algoritmo A3 en una diferencia de 62.41%, 61.34%, 59.98% y 61.60% respectivamente (Tabla VI).

Tabla VIII. Cociente de respuesta promedio obtenido a partir de los algoritmos que consideran un objetivo y los que consideran múltiples criterios usando el método de agregación.

Algoritmos	Cociente de respuesta promedio	Desviación estándar
A2	1183.69	2064.87
B2	45.57	34.71
C1	73.25	58.66
C2	96.38	123.12
C3	73.57	66.46

Sin embargo, no hay una diferencia notable entre los algoritmos que consideran múltiples criterios (C1, C2, y C3), tampoco entre estos últimos en relación al algoritmo que considera un objetivo (B3). El mismo comportamiento se observa en la desviación estándar. Con el calendario que genera el Algoritmo B3 para minimizar el tiempo promedio de permanencia no se logra minimizar la longitud del calendario satisfactoriamente ya que se muestra peor desempeño que el calendario generado por el Algoritmo A3 en un -8.74%. En cambio, los Algoritmos C1, C2 y C3 muestran desempeño de 11.38%, 12.78% y 12.02% (Tabla VI). En el cociente de respuesta promedio el Algoritmo B3 supera al Algoritmo A3 en 89.68% y para el tiempo promedio de espera en 70.40%. Los Algoritmos C1, C2 y C3 superan al Algoritmo B3 en el criterio cociente de respuesta promedio, lo contrario sucede con el criterio tiempo promedio de respuesta (ver Tabla VI).

El algoritmo A4 contempla la estrategia *Min_WT* porque tiene como meta minimizar el tiempo de espera de los trabajos y el Algoritmo B4 tiene la misma meta. Los

resultados experimentales generados por los Algoritmos B4, C1, C2 y C3 muestran un desempeño similar (71.34%, 69.25%, 67.78% y 69.53% respectivamente). El algoritmo A4 presenta el peor promedio (Tabla IX). La desviación estándar entre todos los algoritmos tiene el mismo comportamiento que los valores porcentuales. El Algoritmo B4 muestra un desempeño menor al desempeño del Algoritmo A4 con respecto a la longitud del calendario (bajo el calendario generado para minimizar el tiempo promedio de espera), en cambio, los Algoritmos C1, C2 y C3 superan al Algoritmo A4 en el mismo criterio (Tabla VI). Con respecto al cociente de respuesta promedio los Algoritmos B4, C1, C2 y C3 superan al Algoritmo A4, a su vez, los Algoritmos C1, C2 y C3 superan al Algoritmo B4 (Tabla VI). En el caso del tiempo promedio de espera, los Algoritmos B4, C1, C2 y C3 también superan el desempeño del Algoritmo A4, pero el Algoritmo B4 supera (no en gran magnitud) a los Algoritmos C1, C2 y C3 (Tabla VI).

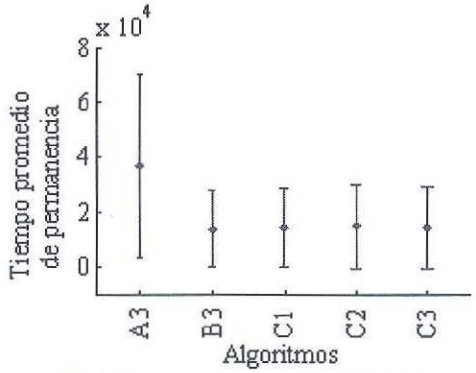


Figura 29. Tiempo promedio de permanencia de los trabajos en el sistema con ambos enfoques.

Tabla IX. Promedio y desviación estándar de los calendarios generados por los algoritmos.

Algoritmos	Tiempo promedio de espera	Desviación estándar
A4	33883.82	30796.37
B4	9710.27	10009.74
C1	10419.38	10542.38
C2	109187.72	11577.95
C3	10325.41	11330.53

Después de comparar el desempeño superior de los algoritmos evolutivos respecto a los algoritmos de calendarización, es necesario conocer el costo que ello implica, en algún sentido es "*el precio que se que se debe pagar por tener un beneficio*"; el número de iteraciones que realiza cada algoritmo forma parte de ese costo. Mientras que las estrategias bien definidas para calendarización (A1, A2, A3 y A4) obtienen resultados en una sola iteración y en unos cuantos segundos, los algoritmos evolutivos (B1, B1, B3, B4, C1, C2 y C3) realizan una serie de iteraciones (Figura 30). Nótese que minimizar la longitud del

calendario resulta "*fácil*" para el Algoritmo B1 pues realiza alrededor de 40 iteraciones en promedio con una desviación estándar relativamente pequeña (comparado con los otros algoritmos). El resto de los algoritmos realizan entre 100 y 150 iteraciones en promedio para minimizar el objetivo; recuérdese que el objetivo de los Algoritmos C1, C2 y C3 implica cuatro criterios que se minimizan simultáneamente por medio del método de agregación (§III.2.2). La Figura 31 muestra los tiempos promedio (en minutos) en que se llevan a cabo las iteraciones de los algoritmos evolutivos. El Algoritmo B1 realiza las iteraciones en un promedio de cinco minutos y en un máximo de diez (solamente optimiza la longitud del calendario). El Algoritmo B2 realiza mayor número de iteraciones y consume mayor tiempo (optimiza el cociente de respuesta promedio), 22 minutos en promedio con una desviación estándar que se extiende por encima de los 33 minutos. Nuevamente, se observa que no hay una diferencia significativa entre los tiempos promedios que necesitan los algoritmos evolutivos que consideran un solo criterio y los que consideran criterios múltiples aunque estos últimos realizan mayor cantidad de operaciones.

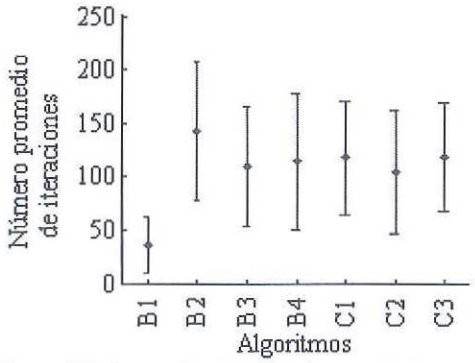


Figura 30. Promedio de iteraciones efectuadas por los algoritmos evolutivos.

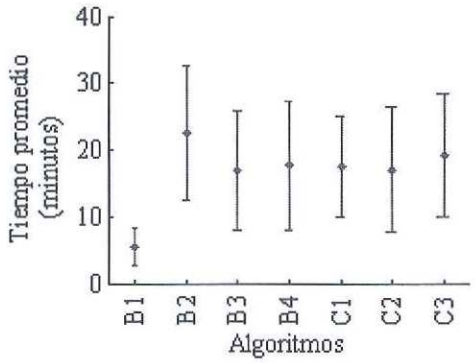


Figura 31. Tiempo promedio consumido por los algoritmos evolutivos durante la simulación.

IV.5. Análisis de los resultados

Bajo las mismas condiciones todos los algoritmos que consideran criterios múltiples y que usan el método de agregación (C1, C2 y C3), muestran desempeño similar ente ellos;

aunque en algunos resultados experimentales un algoritmo supere a los otros dos, todos obtienen un promedio y desviación estándar similar.

Se observa que el mejor desempeño siempre lo obtienen los algoritmos que consideran un solo objetivo (B1, B2, B3 y B4) con respecto a las estrategias de calendarización (A1, A2, A3 y A4). Los algoritmos que aplican el método de agregación (C1, C2 y C3) también superan el desempeño de las estrategias de calendarización en el criterio en común.

Es importante notar que los algoritmos que aplican el método de agregación para criterios múltiples (C1, C2 y C3) efectúan en promedio el mismo número de iteraciones como los algoritmos que consideran un solo objetivo (B2, B3 y B4), lo mismo sucede con las desviaciones estándar respectivas. En el caso del tiempo de simulación es similar, a pesar de que los algoritmos con criterios múltiples, realizan mayor cantidad de operaciones que los algoritmos que consideran un solo criterio.

Capítulo V

Conclusiones y trabajo futuro

V.1. Resumen

En este trabajo se aborda el problema de calendarización de trabajos en un Grid jerárquico de dos niveles con criterios múltiples usando el método de agregación. La idea básica es calendarizar un conjunto de trabajos simultáneamente (de los cuales no se conoce el tiempo de ejecución) sobre un conjunto de recursos disponibles, tomando en cuenta las preferencias de los usuarios (importancia de cada criterio). Para este fin se aplica un algoritmo genético estándar que asigna recursos a los trabajos y utiliza al operador de agregación (OWA) como función de evaluación. Se realizan 30 experimentos con una carga de trabajo sintética; cada experimento consta de 1000 trabajos. Se presentan resultados experimentales del operador genético OBX (operador de cruzamiento) en combinación con los operadores de mutación Insert, Swap y Switch.

Para conocer al algoritmo que genera mejores soluciones, se aplicó el procedimiento para algoritmos genéticos que contemplan un solo objetivo; similarmente se efectúa el procedimiento con los algoritmos que incluyen estrategias de calendarización. Se comparan los resultados experimentales generados por los algoritmos y se observa que los algoritmos genéticos que contemplan varios criterios y que aplican el método de agregación superan a los algoritmos que consideran a las estrategias de calendarización.

V.2. Conclusiones finales

Para el problema tratado en el presente trabajo se obtienen las siguientes conclusiones:

1. Todas las combinaciones de operadores genéticos muestran relativamente la misma calidad de soluciones promedio.
2. Con base en la conclusión anterior, se puede aplicar el operador de cruzamiento OBX con operador de mutación Insert, Swap o Switch ya que no existe diferencia promedio.
3. Bajo la codificación de dos dimensiones usada, los algoritmos genéticos que consideran varios objetivos a la vez y que aplican el método de agregación requieren en promedio el mismo tiempo de simulación que los algoritmos genéticos que consideran un solo objetivo.
4. Similarmente, los algoritmos genéticos con múltiples criterios que aplican el método de agregación efectúan en promedio el mismo número de iteraciones que los algoritmos genéticos que contemplan un solo objetivo.
5. La función usada para generar pesos evita generar pesos al azar, que a la vez reduce esfuerzo innecesario, además de contribuir a la generación de soluciones con calidad.
6. Si se considera una función de utilidad que pondera a todo los criterios relativamente con los mismos pesos, entonces el método de agregación obtiene los mejores resultados en promedio.

Después de implementar, comparar y analizar algoritmos evolutivos aplicados al problema de calendarización considerando varios criterios y usando el método de agregación en un Grid computacional, se puede concluir que es apropiado aplicar algoritmos genéticos al problema de calendarización planteado usando el método de agregación en Grid computacional de dos niveles porque supera el desempeño de las estrategias clásicas de calendarización. Con ello se cubre el objetivo principal de esta tesis.

V.3. Trabajo futuro

El presente trabajo provee las bases para estudios futuros:

1. Utilizar cargas de trabajos con mayor diversidad, tratando de aproximarse aun más a las reales y analizar el comportamiento de los algoritmos aquí propuestos.
2. Los experimentos aquí simulados equivalen a los trabajos propuestos en una semana, sin embargo, se sabe que es una corta aproximación a la realidad, por eso es necesario incrementar el tamaño de cada lote.
3. Similarmente, el número de métricas que se consideran en casos reales es mayor al aquí tratado, entonces se deben realizar experimentos incrementando el número de ellas.
4. Aunado a los puntos anteriores, se propone usar diferentes configuraciones de Grid para probar la robustez de los operadores genéticos.
5. Encontrar los operadores y parámetros genéticos que aportan mayor calidad en los resultados con técnicas como ANOVA.
6. Implementar más operadores de cruzamiento y tomar en cuenta los puntos anteriores.
7. Pensando en el mismo problema, implementar y analizar los resultados experimentales con algoritmos evolutivos que consideran el enfoque frente de Pareto.
8. Si bien en la literatura proponen a los algoritmos genéticos como herramienta para abordar el problema aquí tratado, no se debe subestimar heurísticas tales como búsqueda tabú o recocido simulado.
9. Extender el problema al caso con trabajos en línea y validar los puntos arriba mencionados.

Referencias

Abraham, A. R. Buyya y B. Nath. 2000. Nature's heuristics for scheduling jobs on computational grids. En: The 8th IEEE Int. Conference on Advanced Computing and Communications. Cochin, India. 45-52 p.

Aceves, R. 2003. *Estudio de Operadores Genéticos para un Problema de Calendarización Multi-Objetivo.* En: Tesis de maestria, CICESE, Ensenada. B. C. Mex. 88 pp.

Anoep, S., C. Dumitrescu, D. Epema, A. Iosup, M. Jan, H. Li y L. Wolters. 6 de septiembre de 2007. *The Grid Workload Format.* Technische Universiteit Delft.. http://gwa.ewi.tudelft.nl/TheGridWorkloadFormat_v001.pdf (último acceso: 22 de diciembre de 2008)

Brizuela, C., N. Sannomiya y Y. Zhao. 2001. *Multi-objective Flow-Shop: Preliminary Results.* Lecture Notes in Computer Science. vol. 1993: 443-457 p.

Brown, M. y E. Smith. 2005. *Directed Multi-Objective Optimization.* International Journal of Computers, Systems and Signals. 6(1): 3-17 p.

Brucker, P. 1998. *Scheduling Algorithms.* Springer-Verlag. 2nd edition. New York. 354 pp.

Büche, D., P. Stoll, R. Dornberger y P. Koumoutsakos. 2002. *Preprint: Multi-objective Evolutionary Algorithm.* IEEE Transactions on Systems, Man and Cybernetics. 32(4).

Carlier, J. y P. Chrétienne. 1988. *Problem d'ordonnancement:modelisation / compexite /algorithmes.* In french Masson. Paris.

Carretero, J. y F. Xhafa. 2007. *Genetic Algorithm Based Schedulers for Grid Computing Systems.* International Journal of Innovative Computing, Information and Control. 3(5): 1-19 p.

Carretero, J.y Xhafa, F. 2006. *Using genetic algorithms for scheduling jobs in large scale grid applications*. Journal of Technological and Economic Development—A Research Journal of Vilnius Gediminas Technical University 12(1): 11–17 p.

Chipperfield, A. J. y P. J. Fleming. 1996. *Multiobjective Gas Turbine Engine Controller Design Using Genetic Algorithms.* IEEE Transactions on Industrial Electronics. 43(5): 583-587 p.

Coello, C. A. 1999. *A comprehensive Survey of Evolutionary-Based Multiobjective Optimzation techniques.* Knowledge and Information Systems. An international Journal. 1(3): 269-308 p.

Coello, C. A. 2001. *A Short Tutorial on Evolutionary Multiobjective Optimization.* Lecture Notes in Computer Science. 1993(2001): 21-40 p.

Coello, C. A., D. A. Van Veldhuizen y G. B. Lamont. 2002. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer. 1 edicion. New York. 1-97 p.

Cormen, T. H., C. E. Leiserson, R. L. Rivest y C. Stein. 2001. *Introduction to algorithms.* McGraw-Hill. second edition. USA. 1216 pp.

Das, I. y J. Dennis. 1997. *A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems.* Structural Optimization. 14(1): 63-69 p.

Deb, K., S. Agrawal, P. Pratap y T. Meyarivan. 2000. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II.* In M. Schoenauer Schoenauer, Springer. vol.1917: 849-858 p.

Deb, K. 2001. *Multi-objective optimization using evolutionary algorithms.* Wiley, Chichester. UK. 497 pp.

Diachin, D., L. Freitag, D. Heath, J. Herzog, W. Michels y W. Plassmann. 1996. *Remote engineering tools for the design of pollution control system for commercial boilers.* International Journal of Supercomputer Applications. 10(2): 208-218 p.

Eiben, A. E. y J. E. Smith. 2003. *Introduction to Evolutionary Computing.* Springer. First Edition. Verlag. 300 pp.

Esquivel, S., S. Ferrero, R. Gallard, C. Salto, H. Alfonso y M. Schütz. 2002. *Enhanced evolutionary algorithms for single and multiobjective optimization in the job shop scheduling problem.* Knowledge-Based Systems. 15(1-2): 13-25 p.

Etsion, Y. y D. Tsafrir. 2005. *A short survey of commercial cluster batch schedulers.* School of Computer Science and Engineering, the Hebrew University. Jerusalem, Israel. Reporte técnico 2005-13.

Feitelson, D. G., L. Rudolph y U. Schwiegelshohn. 2005. *Parallel Job Scheduling – A Status Report.* Lecture Notes in Computer Science (LNCS), Proceedings of Job Scheduling Strategies for Parallel Processing . vol. 3277: 1–16 p.

Feitelson, Dror G. 8 de diciembre de 2005. *Parallel Workloads Archive.* The Hebrew University, Jerusalem, Israel. http://www.cs.huji.ac.il/labs/parallel/workload/ (último acceso: 22 de diciembre de 2008).

Feitelson, Dror G. 8 de diciembre de 2005. *The Standard Workload Format.* The Hebrew University, Jerusalem, Israel. http://www.cs.huji.ac.il/labs/parallel/workload/swf.html (último acceso: 22 de diciembre de 2008)

Fonseca, C. M. y P. J. Fleming. 1993. *Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization.* In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann. San Mateo, California. vol. 1917: 849-858 p.

Foster, I., C. Kesselman y S Tuecke. 2001. *The anatomy of the grid.* International Journal of Supercomputer Applications. 15(3): 200-222 p.

Foster, I. y C. Kesselman. 1998. *The Grid - Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers. San Fransisco. Edicion 2. 748 pp.

Foster, I. y C. Kesselman. 2004. *The Grid in a nutshell.* En: Foster y Kesselman (eds.). *Grid resource management: state of the art and future trends.* Kluwer Academic Publishers. First. Norwell, MA, USA. 3 - 13 p.

Foster, I. y C. Kesselman. 2003. *The Grid In A Nutshell in Grid Resource Management State of the Art and Future Trends.* Edited by Jarek Nabrzyski. 1 edition. 3 -13 p.

Foster, I. y C. Kesselman. 1999. *The Grid: Blueprint for a future computing infrastructure.* Morgan Kaufmann. San Fransisco. Edicion 2. 748 pp.

Garey, M. R. y R. L. Graham. 1975. *Bounds for Multiprocessor Scheduling with Resource Constraints.* SIAM Journal on Applied Mathematics. 4(2): 187-200 p.

Garey, M. y R. Graham. 1975. *Bounds for multiprocessor schedulingwith resource constraints.* SIAM Journal on Computing. 4(2): 187-200 p.

Garey, M. R. y D. D. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company. Primera edición. San Francisco. 340 pp.

Gass, S. y T. L. Saaty. 1955. *The computational algorithm for the parametric objective function.* Naval Research Logistics Quarterly. 2(1): 39-45 p.

Gehring, J. y A. Streit. 2000. *Robust Resource Management for Metacomputers.* In Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC-9 '00), 105 pp.

Gen, M. y R. Cheng. 1997. *Genetic algorithms & engineering design.* John Wiley & Sons. . New Yor. 342 pp.

Gen, M. y R. Cheng. 2000. *Genetic algorithms & engineering optimization.* John Wiley & Sons. New York. 512 pp.

Goldberg, D. E. 1991. A *comparative analysis of selection schemes used in genetic algorithms.* In G. J. Rawlins (Ed.) *Foundations of Genetic Algorithms. San Mateo:* Morgan Kauffman. 69-93 p.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley. First edition. Massachusetts. 95 -99 p.

Graham, R. L. 1966. *Bounds on Multiprocessing Timing Anomalies.* Bell System Technical Journal. 45(9): 1563-1581 p.

Graham, R. L. 1969. *Bounds on Multiprocessing Timing Anomalies.* SIAM Journal on Applied Mathematics. 17(2): 416-429 p.

Graham, R. L., E. L. Lawler, J. K. Lenstra y A. H. G. Rinnooy Kan. 1979. *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.* Annals of Discrete Mathematics. 5(1979): 287-326 p.

Grimme, C., J. Lepping y A. Papaspyrou. 2008. *Discovering Performance Bounds for Grid Scheduling by using Evolutionary Multiobjective Optimization.* In Prococeedings of the Genetic and Evolutionary Computation Conference (GECCO 2008). ACM Press. Atlanta, Georgia, USA. 1491-1498 p.

Hamscher, V., U. Schwiegelshohn, A. Streit y R. Yahyapour. 2000. *Evaluation of Job-Scheduling Strategies for Grid Computing.* Proceedings of the First IEEE/ACM International Workshop on Grid Computing. vol. 1971: 191-202 p.

Holland, J. H. 1975. *Adaptation in natural and artificial systems.* University of Michigan Press. Ann Arbor, MI.

Ishibuchi, H. y T. Murata. 1996. *Multi-objective genetic local search algorithm.* In Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC'96). 119-124 p.

James, A., K. A. Hawick y P. D. Coddington. 1999. *Scheduling Independent Tasks on Metacomputing Systems.* In Proc. Conf. on Parallel and Distributed Systems, Fort Lauderdale, FL.

Knowles, J. y D. Corne. 2002. *On metrics for comparing nondominated sets.* Proceedings of the 2002 Congress on Evolutionary Computation, Hawaii, USA. IEEE Neural Network Council (NNC), Institution of Electrical Engineers (IEE), Evolutionary Programming Society (EPS), IEEE Presss. vol. 1: 711-716 p.

Kurowski, J., J. Nabrzyski, A. Oleksiak y J. Weglarz. 2006. *Scheduling jobs on the grid--multicriteria approach.* In Computational methods in science and technology. 12(2): 123-138 p.

Kurowski, K., J. Nabrzyski, A. Oleksiak y J. Węglarz. 2008. *A multicriteria approach to two-level hierarchy scheduling in grids.* Journal of Scheduling. 11(5): 371 - 379 p.

Kursawe, F. 1991. *A variant of evolution strategies for vector optimization.* En: H.-P. Schwefel and R. M¨anner (eds.). *Parallel Problem Solving from Nature.* Springer-Verlag. Berlin, Germany. 193–197 p.

Lifka, D. A. 1998. *An Extensible Job Scheduling System For Massively Parallel Processor Architectures.* College of the Illinois Institute of Technology Chicago. Illinois. Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the Illinois Institute of Technology Chicago .

Lim, D., Y. Ong, Y. Jin, B. Sendhoff y B. Lee. 2007. *Efficient Hierarchical Parallel Genetic Algorithms using Grid Computing.* Future Generation Computer Systems. 23(4): 658-670 p.

Litzkow, J., M. Livny y W. Mutka. 1988. *Condor- a hunter of idle workstations.* In proccedings of the 8th International Conference of Distributed Computing Systems. San Jose, CA. IEEE CS Press, USA. pp. 104-111.

Lorpunmanee, S., M. Noor, A. Hanan y S. Srinoy. 2006. *Genetic algorithm in Grid scheduling with multiple objectives.* Proceedings of the 5th WSEAS int. Conf. on Artificial Intelligence. knowledge Engineering and Data Bases. Madrid, Spain 429-435 p.

Madrigal, D., L. M. Galaviz, J. M. Ramírez, R. Sánchez, A. Tchernykh y J. A. Verduzco. 2006. *Estrategias de Calendarización para un Grid Computacional.* CiComp'06, Primer Congreso Internacional de Ciencias Computacionales. 6 - 8 de noviembre. Ensenada, Baja California, México.

Marzullo, K., M. Ogg, A. Ricciardi, A. Amoroso, F. Calkins y E. Rothfus. 1996. Nile: *Wide-area computing for high energy physics.* Proceedings of the 1996 SIGOPS Conference.

Mechoso, C., C.-C. Maria, J Farrara, J. Spahr y C. Moore. 1993. *Parallelization and distribution of a coupled atmosphere-ocean general circulation model.* Mon. Wea. Rev. vol. 121:2062-2076 p.

Nieplocha, j. y R. Harrison. 1996. *Shared memory NUMA programming on the I-WAY.* In Proc. 5th IEE Symp. On High Performance Distributed Computing. IEEE Computer Society Press. New York, USA. 432-441pp.

Norman, M., P. Beckman, G. Bryan, J. Dubinski, D. Gannon, K. Hernquist, K. Keahey, J. Ostriker, J. Shalf, J. Welling y S. Yang. 1996. *Galaxies collide on the I-WAY: An example of heterogeneous wide-area collaborative supercomputing.*. International Journal of Supercomputer Applications. 10(2): 131-140 p.

Osyczka, A. 1985. *Multicriteria optimization for ingeenering design. In Gero, J. S., editor, Desing Optimization.* Academic Press. 193 - 227 pp.

Pinedo, M. 1995. *Scheduling - theory, algorithms, and systems.* Prentice Hall. Englewood Cliffs. 586 pp.

Potter, C., R. Brady, P. Moran, C. Gregory, N. Kisseberth, J. Lyding y J. Lindquist. 1996. *EVAC: A virtual environment for control of remote imaging instrumentation.* IEEE Computer Graphics and Applications. 6(4): 62-66 p.

Potter, C., Z-P. Liang, C. Gregory, H. Morris y P. Lauterbur. 1994. *Toward a neuroscope: A real-time system for the evaluation of brain function.* In Proc. First IEEE Int. Conf. on Image Processing. vol. 3: 25-29 p.

Ramírez, J., A. Rodríguez, A. Tchernykh y J. Verduzco. 2007. *Rendimiento de Estrategias de Calendarización Considerando Fluctuación de Tiempo de Ejecución de Tareas en un Grid Computacional.* Conferencia Latinoamericana de Computación de Alto Rendimiento, CLCAR. Santa Marta, Colombia 13-18 agosto. 273-281 p.

Rodríguez, J., A. Medaglia y C. Coello. 2009. *Design of a motorcycle frame using neuroacceleration strategies in MOEAs* . Journal of Heuristics. 15(2): 177-196 p.

Schaffer, J. D. 1985. *Multiple objective optimization with vector evaluated genetic algorithms.* Proceedings of an International Conference on Genetic Algorithms and Their Applications. 93-100 p.

Schwiegelshohn, U., A. Tchernykh y R. Yahyapour. 2008. *Online Scheduling in Grids.* IPDPS 2008 Conference. 14-18 de abril. Miami, Florida, EUA.

Shan, H., L. Oliker y R. Biswas. 2003. *Job Superscheduler Architecture and Performance in Computational Grid Environments.* ACM/IEEE Conference on Supercomputing. . Phoenix, AZ, USA.

Sinnen, O. 2007. *Task scheduling for parallel systems.* Wiley Series. New Jersey. 296 pp.

Srinivas, N. y K. Deb. 1994. *Multiobjective optimization using nondominated sorting in genetic algorithms.* Evolutionary Computation. 2(3): 221-248 p.

Systems. September 20-24 (in conjunction with ENC'04 Mexican International Conference in Computer Science), IEEE Computer Society Press. Colima, Mexico.

Zitzler, E., M. Laumanns y S. Bleuler. 2004. *A Tutorial on Evolutionary Multiobjective Optimization.* Lecture Notes in Economics and Mathematical Systems. vol. 535: 3-37 p.

Zitzler, E., M. Laumanns, L. Thiele, C. M. Fonseca y G. Fonseca. 2002. *Why quality assessment of multiobjective optimizers is difficult.* En: Langdon,W. B., E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, y N. Jonoska, editores, GECCO 2002. Proceedings of the Genetic and Evolutionary Computation Conference. San francisco, CA. Morgan Kaufman Publishers. 666-673 p.

Apéndice A

Clase de Problemas

A continuación se describen las clases de problemas de acuerdo a Cormen *et al.*, 2001:

1. La clase P consiste de aquellos problemas que se pueden resolver en tiempo polinomial. Es decir, son problemas que se pueden reslver en tiempo $O(n^k)$ para alguna constante k y una entrada de tamaño n.
2. La clase NP consiste de aquellos problemas que son *verificables* en tiempo polinomial. Es decir, si se proporciona un *certificado* de una solución, entonces, se puede verificar que el certificado (solución) es correcto en tiempo polinomial en función del tamaño de la entrada del problema.
3. Un problema C se encuentra en la clase $NP - competo$ si: C está en NP y todo problema en NP se puede reducir en forma polinomial en C.
4. La clase de problemas $NP - difícil$ $(NP - hard)$ es el conjunto de problemas de decisión que contiene los problemas H tales que todo problema L en NP se puede transformar de manera polinomial en H.

La relación entre las clases de complejidad P y NP es una pregunta que aún no ha respondido la teoría de la computación. En esencia, la pregunta ¿es $P = NP$? significa: ¿si es posible "verificar" rápidamente soluciones positivas a un problema del tipo SI/NO (donde "rápidamente" significa "en tiempo polinomial"), es que entonces también se pueden "obtener" las respuestas rápidamente?.

Los problemas NP-difícil no son todos *NP*: los problemas *NP-completo* significa problemas que son *completos* en *NP*, es decir, los más difíciles de resolver en *NP*; *NP-difícil* quiere decir *al menos* tan complejo como *NP* (pero no necesariamente en *NP*).

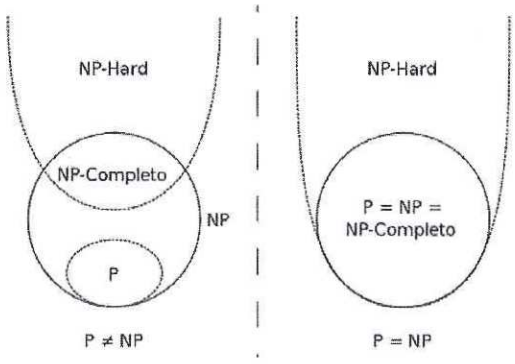


Figura 32. Clase de problemas. Representación de los problemas *P*, *NP*, *NP-completo*, y *NP-difícil* (*NP-hard*) tomando en cuenta que los problemas *P* son diferentes de *NP* o que son iguales (Figura tomada de: http://es.wikipedia.org/wiki/NP-hard).