SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Tecnológico Nacional de México
Instituto Tecnológico de Ciudad Madero

División de Estudios de Posgrado e Investigación

Optimización del problema de programación de tareas independientes en sistemas de procesamiento paralelo

TESIS

Para obtener el grado de:
Maestro en Ciencias de la Computación

PRESENTA:
I.S.C. José Carlos Soto Monterrubio

DIRECTOR DE TESIS:
Dr. Héctor Joaquín Fraire Huacuja

Cd. Madero, Tamaulipas, México

Marzo 2015

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

"2015, Año del Generalísimo José María Morelos y Pavón"

Cd. Madero, Tamps; a 10 de Marzo de 2015.

OFICIO No.: U5.049/15
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. JOSÉ CARLOS SOTO MONTERRUBIO
NO. DE CONTROL G08070650
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias de la Computación, el cual está integrado por los siguientes catedráticos:

PRESIDENTE :	DR. JUAN JAVIER GONZÁLEZ BARBOSA
SECRETARIO :	DR. RODOLFO ABRAHAM PAZOS RANGEL
VOCAL :	DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA
SUPLENTE	DR. ARTURO HERNÁNDEZ RAMÍREZ
DIRECTOR DE TESIS :	DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA

Se acordó autorizar la impresión de su tesis titulada:

"OPTIMIZACIÓN DEL PROBLEMA DE PROGRAMACIÓN DE TAREAS INDEPENDIENTES EN SISTEMAS DE PROCESAMIENTO PARALELO"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta.

Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
"POR MI PATRIA Y POR MI BIEN"®

M. P. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

S.E.P.
DIVISIÓN DE ESTUDIOS
DE POSGRADO E
INVESTIGACIÓN
ITCM

c.c.p.- Archivo
Minuta

MYCHC 'NLCO ' jar

Ave. 1° de Mayo y Sor Juana I. de la Cruz, Col. Los Mangos, CP. 89440 Cd. Madero, Tam.
Tel. (833) 357 48 20, Fax, Ext. 1002, e-mail: itcm@itcm.edu.mx
www.itcm.edu.mx

ISO 14001
CERTIFIED COMPANY

Declaración de originalidad

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y relevo de ésta a mi director y codirectores de tesis, así como al Instituto Tecnológico de Ciudad Madero y sus autoridades.

Marzo de 2015, Cd. Madero, Tamaulipas

I.S.C. José Carlos Soto Monterrubio

Resumen

En esta tesis el problema de investigación que se aborda es el problema de asignación de tareas independientes en sistemas de procesamiento paralelo. Este problema nace debido al crecimiento de los grandes centros de datos o clusters, en los cuales se tiene que buscar una manera optima de asignar las tareas en las maquinas que serán ejecutadas. Esta asignación tiene que aprovechar la energía consumida y, ademas que el tiempo en que se terminan de ejecutar todas las tareas sea mínimo. Por lo tanto el objetivo de este problema es minimizar el *makespan* y el consumo de energía. Para el control de la energía utilizada en una máquina se usa la técnica de escalamiento dinámico de voltaje y frecuencia. En este trabajo se realizó un estudio comparativo entre dos métodos exactos y dos metaheurísticas.

El primer método exacto es un modelo de Programación Lineal Entera Mixta (MILP, por sus siglas en inglés) propuesto en este trabajo y el segundo es un *Branch and Bound*. Ambos métodos están implementados en dos fases, la primer fase consiste en minimizar el *makespan* y la segunda fase en minimizar la energía consumida. Después se realizó una comparación con los resultados obtenidos de ambos métodos. Los resultados experimentales muestran que el mejor método exacto es el MILP.

La siguiente experimentación realizada es entre dos metaheurísticas multiobjetivo el NSGA–II y el MOEA/D. En el algoritmo NSGA–II se implementaron diversas técnicas de cruza y mutación de los algoritmos evolutivos. Aplicando los indicadores de calidad se selecciono la mejor técnica. Para el algoritmo MOEA/D se implementaron nueve funciones de descomposición y se selecciono la mejor función de acuerdo a los indicadores de calidad.

Con las mejores técnicas de cada metaheurística se realizo una experimentación comparativa entre el NSGA–II y el MOEA/D. Los resultados muestran que la mejor cruza y mutación para el NSGA–II es una cruza uniforme y una mutación propuesta denominada mutación de balanceo de cargas. La mejor función para el MOEA/D es la función Augmented Tchebycheff Kaliszewski. Los resultados experimentales entre ambas metaheurística con sus mejores técnicas muestran que el mejor fue el MOEA/D.

Summary

In this thesis the research problem is Scheduling in unrelated parallel machines. This problem appears due to the rapid growth of data centers or clusters, in which the tasks needs to be optimally scheduled. This schedule needs to exploit the energy consumed and, the time in which all the tasks completed is the minimum. Therefore the problem objective is minimize the makespan and the energy consumption. To control the energy in each machine is used the technique called Dynamic Voltage and Frequency Scaling. In this works we made a comparative study between two exact methods and two meta-heuristics.

The first exact method proposed in this work is a Mixed Integer Linear Programming (MILP) model. The second method is a Branch and bound. Both methods are implemented in two phases, the first phase minimizes the makespan and the second phase minimize the energy consumed. With the results from these algorithms we made a comparative study. The experimental results shows that the best exact method is the MILP.

The following comparison is performed between two multiobjective meta-heuristic the NSGA–II and MOEA/D. In the NSGA–II different crossovers and mutations techniques from evolutive algorithms are implemented. The best techniques were selected by applying quality indicators. In the MOEA/D nine decomposition functions were implemented and the best function was picked according to quality indicators.

Using the best techniques in each meta-heuristic a comparative experiment was performed. The results shows that the best crossover for the NSGA–II is the uniform crossover and the best mutation is the proposed in this work wichs is called load balancing mutation. The best function for the MOEA/D is the Augmented Tchebycheff Kaliszewski. The experiment between both meta-heuristic, each one with theirs best techniques, shows that the best is the MOEA/D.

Índice general

1. Introducción **8**
- 1.1. Descripción del problema . . . 9
  - 1.1.1. Modelo de *makespan* . . . 10
  - 1.1.2. Modelo de energía . . . 10
  - 1.1.3. Definición del problema . . . 11
  - 1.1.4. Instancias del problema . . . 11
  - 1.1.5. Solución candidata . . . 13
  - 1.1.6. Cálculo de la función objetivo *makespan* . . . 13
  - 1.1.7. Cálculo de la función objetivo energía . . . 14
- 1.2. Complejidad del problema . . . 14
- 1.3. Objetivos . . . 14
  - 1.3.1. Objetivo general . . . 14
  - 1.3.2. Objetivos específicos . . . 14
- 1.4. Justificación y beneficios . . . 15
- 1.5. Alcances y limitaciones . . . 15

2. Marco Teórico **16**
- 2.1. Complejidad computacional . . . 16
  - 2.1.1. Clase P . . . 16
  - 2.1.2. Clase NP . . . 16
  - 2.1.3. Problema NP-Completo . . . 16
  - 2.1.4. Problema NP-Hard . . . 17
- 2.2. Métodos de optimización . . . 17
  - 2.2.1. Métodos exactos . . . 17
- 2.3. Heurísticas . . . 18
- 2.4. Metaheurísticas . . . 18
- 2.5. Algoritmos evolutivos . . . 19
  - 2.5.1. Algoritmos genéticos . . . 19
- 2.6. Optimización multiobjetivo . . . 20
  - 2.6.1. Indicadores de calidad . . . 21
- 2.7. Algoritmos Multiobjectivo . . . 23
  - 2.7.1. Non-Dominated sorting genetic algorithm II . . . 23
  - 2.7.2. Algoritmos basados en descomposición . . . 25

3. Estado del arte **27**
- 3.1. Algoritmos heurísticos . . . 27
- 3.2. Algoritmos metaheurísticos . . . 28

4. Métodos exactos propuestos **30**
4.1. Modelo de programación lineal entera mixta 30
4.1.1. Descripción del modelo para *makespan* (Fase I) 30
4.1.2. Descripción del modelo para energía (Fase II) 31
4.2. Descripción del *Branch and Bound* 31

5. Métodos propuestos **33**
5.1. Representación del problema 33
5.2. Implementación genética 34
5.2.1. Selección 34
5.2.2. Cruza 34
5.2.3. Mutación 36

6. Resultados experimentales **39**
6.1. Equipo utilizado 39
6.2. Instancias 39
6.3. Experimentación de métodos exactos. 40
6.4. Experimentación de métodos aproximados 42
6.4.1. Experimentación de cruzas y mutaciones implementadas en el NSGA–II 42
6.4.2. Experimentación de funciones de descomposición en MOEA/D . 45
6.4.3. Experimentación comparativa entre NSGAII y MOEA/D 45

7. Conclusiones y trabajos futuros **47**
7.1. Conclusiones 47
7.2. Productos científicos derivados de la tesis 47
7.3. Publicaciones 47
7.4. Trabajos futuros 48

Capítulo 1

Introducción

En 1992 con la creación del programa *Energy Star* inició la concientización del consumo eficiente de energía, en dispositivos móviles, inalámbricos y dispositivos dedicados. Estos están limitados a la energía que les proveen sus baterías, las cuales deben de ser cambiadas frecuentemente. Actualmente este problema se ha convertido en uno de los principales temas en los dispositivos conectados en red. Como se resume en [Liu and Zhu, 2010], en el 2006 los servidores de E.U.A. y centros de datos consumieron alrededor de 61 billónes de kilowatt-hora (kWh) a un costo de 4.5 billónes de dólares, esto es el 1.5 % del consumo total eléctrico de E.U.A el equivalente a lo generado por 15 plantas eléctricas. Se ha estimado que las Tecnologías de Información y Comunicación (TIC) generan el 2 % de emisión de CO_2 mundial, un número equivalente a las emisiones de aviación [Pettey, 2007].

Actualmente existe una gran demanda de las TIC en velocidad de procesamiento, número de usuarios y ejecución de tareas computacionales cada vez más complejas. Esto ha creado la necesidad de usar centros de datos con sistemas de alto rendimiento computacional conocidos como (*High Performance Computing*, HPC), pero un alto desempeño de computo conlleva a un alto consumo de energía. La Tabla 1.1 muestra el consumo actual de los sistemas HPC [TOP500, 2013]. Como se puede apreciar en la tabla se ha incrementado el desempeño de los HPC e igualmente ha aumentado el consumo de energía. El sistema HPC que tiene un mayor consumo de energía es el Tianhe-2 (Primer lugar en junio 2013) con un total de 17.80 MW y con un máximo de 33,862.7 Tera Flops.

Tabla 1.1: Consumo de energía en sistemas HPC

Consumo	Nov 2008	Nov2010	Nov2012	Jun2013
Consumo promedio de energía en el TOP10 de [TOP500, 2013].	2.84 MW	3.2 MW	5.17 MW	6.67MW
Promedio de Tera Flops en el TOP10 de [TOP500, 2013].	515 TFlops	1,241 TFlops	6,809 TFlops	10,766 TFlops
Consumo promedio de energía en el [TOP500, 2013].	358 kW	447 kW	684 kW	833 kW
Promedio de Tera Flops en el [TOP500, 2013].	34.74 TFlops	87.57 TFlops	324.27 TFlops	447.30 TFlops
Total de sistemas que usan más de 1 MW [TOP500, 2013].	14	25	39	43

Para reducir el consumo de energía en estos sistemas los investigadores se han

enfocado en las siguientes estrategias:

- Diseño de equipos de consumo eficiente.
- Técnica de escalamiento dinámico de voltaje y frecuencia, DVFS (*Dynamic Voltage and Frequency Scaling*).
- Apagado de componentes cuando baja el uso del sistema.
- Limitación de energía.
- Administración térmica.

Por otra parte, debido al rápido incremento del poder de procesamiento de computadoras de bajo costo y el desarrollo de tecnologías en redes de alta velocidad se ha acelerado la difusión de ambientes de cómputo distribuidos para resolver problemas complejos. La expresión *grid computing* (computo en red o procesamiento en red) se ha vuelto popular. Esta tecnología esta denotada por un conjunto de técnicas de cómputo distribuido que trabaja sobre una supercomputadora virtual que permite un acoplamiento flexible. El *grid computing* está construido con diferentes componentes heterogéneos de diferentes características y poder de computo. Este tipo de infraestructura permite proveer mecanismos fuertes y con accesibilidad de costo – efectividad a colecciones de computadoras distribuidas para resolver problemas que requieren un gran poder de computo [Foster and Kesselman, 1998]. Los sistemas de Computo Heterogéneos (*Heterogeneous Computing*, HC) son sistemas que contienen un conjunto de elementos de procesamiento que están coordinados, además de estar conectados a través de una red de trabajo.

El principal problema cuando se usan ambientes de cómputo distribuido consiste en encontrar una estrategia de planeación para un conjunto de tareas que serán ejecutadas. Esta planeación tiene el objetivo de asignarles un recurso de cómputo de una manera inteligente cubriendo algunos criterios de eficiencia. Este problema es conocido como Problema de Calendarización de Computo Heterogéneo (*Heterogeneous Computing Scheduling Problem*, HCSP) [CECAL, 1993].

1.1. Descripción del problema

El problema de investigación en este trabajo es el HCSP tiene como objetivo minimizar el tiempo máximo de terminación, conocido como *makespan*, y el consumo de energía. El HCSP consta de un conjunto de M procesadores/máquinas, cada máquina $m_j \in M$ es $DVFS$, lo que permite que cada m_j opere con un conjunto de voltajes suministrados V, a una velocidad relativa asociada a cada $v_i \in V$. Esta técnica ofrece la ventaja de tener una fácil implementación y buen rendimiento [Pecero et al., 2011, Pineda et al., 2013].

En este trabajo de investigación se considerará que:

- La tarea no puede ser interrumpida una vez que ha sido asignada.
- Cada tarea es una unidad trabajo que será asignada a un procesador.
- Las tareas no pueden ser divididas en sub-tareas.
- El tiempo de ejecución de una tarea varía de una máquina a otra.
- Todas las tareas son independientes.

Antes de mostrar la definición formal del problema se abordan elementos importantes para entender el problema, estos elementos son el *makespan* y energía.

1.1.1. Modelo de *makespan*

Se tiene un conjunto de tareas $T = \{t_1, t_2, \ldots, t_n\}$ y máquinas $M = \{m_1, m_2, ..., m_k\}$. Cada tarea t_i tiene asociado un costo computacional $P_{i,j}$ en una máquina m_j a una máxima velocidad y voltaje. Cuando el máximo voltaje es seleccionado entonces la *velocidad* es igual a 1 (el tiempo de ejecución normal en P), con un voltaje menor seleccionado la velocidad realtiva decrece por ejemplo cuando la *velocidad* es igual a 0.5 (50 % de la velocidad normal), entonces existe un tiempo de ejecución relativo $P'_{i,j}$ el cual es obtenido mediante la ecuación siguiente [Pecero et al., 2011, Pecero et al., 2010].

$$P'_{ij} = \frac{P_{ij}}{Velocidad} \tag{1.1}$$

Donde P_{ij} corresponde al tiempo que requiere la tarea t_i para ejecutarse en la máquina m_j, y *Velocidad* esta asociada al nivel de configuración del procesador. El *makespan* es el tiempo en que finaliza la ejecución de la última tarea. El *makespan* mínimo es producido por la asignación de máquinas y tareas que minimice:d

$$makespan = MAX_{j=1}^{k}(\sum P_{i,j} \forall t_i \in m_j) \tag{1.2}$$

1.1.2. Modelo de energía

El modelo de energía usado en este trabajo es derivado del modelo de consumo de energía CMOS (*Complementary Metal-Oxide Semiconductor*) el cual se encuentra definido en [Lee and Zomaya, 2011]. La disipación dinámica de energía P en este modelo está dado por:

$$P = \alpha C V^2 f. \tag{1.3}$$

Donde α es el factor de actividad, C es la carga total de capacitancia, V es el voltaje suministrado, y f es la frecuencia de operación. La ecuación anterior claramente indica que el voltaje suministrado es un factor dominante y su reducción será muy influyente para disminuir el consumo de energía. El consumo de un programa paralelo G en este trabajo está dado por:

$$E_c = \sum_{i=0}^{n} \alpha C V_i^2 f P'_{i,j} = \sum_{i=0}^{n} K V_i^2 f P'_{i,j}. \tag{1.4}$$

Donde V_i es el voltaje suministrado al procesador en el cual la tarea t_i es ejecutada, y P'_{ij} es el costo computacional de la tarea t_i con la configuración de voltaje y máquina programada. En el presente trabajo la constante K se desprecia y la frecuencia de operación es igual a 1. Usando para el cálculo de la energía la siguiente sumatoria [Pecero et al., 2010].

$$E_c = \sum_{i=0}^{n} V_i^2 P'_{i,j}. \tag{1.5}$$

1.1.3. Definición del problema

Enseguida se presenta la definición formal del problema así como conceptos relevantes relacionados con el mismo.

Considerando un conjunto de n tareas $T = \{t_1, t_2, \ldots, t_n\}$ que tienen que ser procesadas en k máquinas $M = \{m_1, m_2, \ldots, m_k\}$, también se tienen los tiempos de ejecución de cada tarea en cada máquina $P = \{p_{1,1}, p_{1,2}, \ldots, p_{n,k}\}$, suponiendo además, que en todo momento cada máquina puede procesar una sola tarea a la vez y cada tarea puede ser procesada en una sola máquina a la vez [Mokotoff and Jimeno, 2002]. El HCSP tiene como objetivo encontrar una apropiada asignación de tareas del conjunto T, en procesadores de las máquinas del conjunto M, tal que optimice el criterio de rendimiento. Donde nuestro criterio de interés es minimizar el *makespan* [Graham et al., 1979].

En [CECAL, 1993] se presenta la siguiente formulación matemática del problema.

- Dado un sistema HC compuesto de un conjunto de tareas T, recursos de computo (máquinas) M y los tiempos de ejecución P.
- El objetivo del HCSP está dirigida a minimizar el *makespan* al encontrar una asignación de tareas a máquinas la cual minimice:

$$MAX_{j=1}^{k} \left(\sum P_{i,j} \forall t_i \in m_j \right) \tag{1.6}$$

Debido a que las máquinas son DVFS y heterogéneas entonces existen diferentes niveles de voltaje para cada máquina m_j con una velocidad relativa asociada. Cuando el voltaje más alto es seleccionado la velocidad es igual a 1 que es el tiempo de ejecución normal, cuando un voltaje más bajo es seleccionado la velocidad disminuye, por ejemplo una velocidad igual a 0.5 es el 50 % de la velicada normal. Por lo tanto también se tienen los siguientes elementos.

- Un conjunto de voltajes para cada máquina $V_j = \{v_{i,1}, v_{i,2}, ..., v_{j,l}\} \forall m_j \in M$ de diferentes tamaños l.
- Generando como salida las asignaciones de energía para las n tareas y una asignación de las n tareas en los k procesadores tal que minimice.

$$\sum_{j=1}^{k} V_{j,p}^2 P'_{i,j} \forall t_i \in m_j \tag{1.7}$$

  Donde p es el índice del voltaje seleccionado en V_j.

El objetivo de este problema consiste en encontrar la asignación de máquina, voltaje y tarea que minimice 1.6 y 1.7. Los dos objetivos de este problema se encuentran en conflicto, debido a que al minimizar el *makespan* produce un aumento en el consumo de energía y minimizar el consumo de energía produce un aumento del *makespan*. Lo cual lo convierte en un problema de optimización multiobjetivo (MOP).

1.1.4. Instancias del problema

A pesar de la gran comunidad de investigadores que han abordado este problema, aún no existe un conjunto de instancias estandarizadas (*benchmark*). Sin embargo las más comúnmente utilizadas y documentadas se presentan en [Braunt et al., 2001, Ali et al., 2000].

El conjunto de instancias diseñadas en [Braunt et al., 2001] contiene un total de doce instancias, todas tienen una dimensión de 512 tareas y 16 máquinas. Las características utilizadas en este conjunto son la heterogeneidad de las máquinas y tareas así como la consistencia de los tiempos de ejecución. La variación de los tiempos de ejecución de las tareas en una máquina determinada se define como heterogeneidad de las tareas. La variación de lo tiempos de ejecución de una determinada tarea en todas las máquinas se conoce como heterogeneidad de las máquinas. Los tiempos de ejecución son consistentes cuando una máquina m_j ejecuta cualquier tarea t_i más rápido que cualquier máquina m_k, entonces la máquina m_j ejecuta todas las tareas más rápido que la máquina m_k.

Este conjunto de instancias están etiquetadas siguiendo el patrón d_c_THMH.0. Donde d indica el tipo de distribución de probabilidad que se usó para generar los valores de heterogeneidad, se utiliza u para indicar una distribución uniforme. La c indica el tipo de consistencia, los valores posibles son c para consistente, i para inconsistente y s para semi-consistente. Los siguientes caracteres TH y MH indican el nivel de heterogeneidad para las tareas y máquinas respectivamente. Los valores posibles son lo y hi para indicar una alta o baja heterogeneidad, respectivamente. El número al final representa el número de prueba.

En [Braunt et al., 2001] se generaron diversas clases de instancias pero únicamente los de la clase 0 ganaron popularidad. Esta clase de instancias contienen los tiempos estimados de ejecución (P_{ij}) de cada tarea en cada una de las diferentes máquinas, bajo el siguiente formato:

Tabla 1.2: Formato de las instancias [Braunt et al., 2001]

$P(t_1, m_1)$
$P(t_1, m_2)$
...
$P(t_1, m_j)$
$P(t_2, m_1)$
$P(t_2, m_2)$
...
$P(t_2, m_j)$
...
$P(t_i, m_j)$

También se utilizó otro conjunto de cuarenta instancias proporcionadas por el Dr. Jonathan Pecero Sanchez de la universidad de Luxemburgo.

En las instancias no se proporcionan las configuraciones de velocidad ni de voltaje. Por lo tanto se utilizaron las configuraciones que se han aplicado en los trabajos relacionados [Pineda et al., 2013, Pecero et al., 2011]. Estas configuraciones consisten en un conjunto de voltajes/velocidades para tres máquinas. Cuando se requieran más de tres máquinas se aplicará el método de *round-robín* por lo cual se repiten las mismas configuraciones para las nuevas máquinas.

Las configuraciones utilizadas se muestran en la Tabla 1.3.

Tabla 1.3: Configuración de procesadores con sus respectivos voltajes y velocidades

Nivel	Máquina 1		Máquina 2		Máquina 3	
	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad
1	1.75	1.00	1.50	1.00	2.20	1.00
2	1.40	0.80	1.40	0.90	1.90	0.85
3	1.20	0.60	1.30	0.80	1.60	0.65
4	0.90	0.40	1.20	0.70	1.30	0.50
5	–	–	1.10	0.60	1.00	0.35
6	–	–	1.00	0.50	–	–
7	–	–	0.90	0.40	–	–

1.1.5. Solución candidata

En esta sección se muestra un ejemplo de una solución candidata utilizando los datos de una instancia de tres máquinas y ocho tareas, como se muestra en la siguiente tabla.

Tabla 1.4: Costos computacionales de las tareas a voltaje máximo P_{ij}

Tarea	M1	M2	M3
1	15	14	12
2	12	10	7
3	10	15	11
4	9	18	12
5	13	9	15
6	7	12	9
7	14	11	15
8	10	18	15

La Tabla 4 muestra una solución candidata con las asignaciones de tarea – máquina y nivel de voltaje seleccionado. Esta representación quiere decir que la Tarea 1 se ejecuta en la máquina 2 con un nivel de voltaje 2, la tarea 2 se ejecuta en la máquina 3 con un nivel de voltaje 2, y así sucesivamente para las demás tareas.

Tabla 1.5: Solución candidata

Tarea	1	2	3	4	5	6	7	8
Máquina	2	3	2	1	1	2	3	1
voltaje	2	2	2	1	2	1	1	2

1.1.6. Cálculo de la función objetivo *makespan*

El cálculo del *makespan*, para la solución candidata es el siguiente. Utilizando la ecuación 1.1 se obtiene el tiempo de ejecución de cada asignación, la tabla 1.6 muestra los resultados obtenidos.

Tabla 1.6: Costo computacional de cada tarea con su respectiva configuración de máquina y voltaje

Tarea	1	2	3	4	5	6	7	8
Máquina	2	3	2	1	1	2	3	1
$P'_{i,j}$	15.56	8.24	16.67	9.00	16.25	12.00	15.00	12.50

La siguiente tabla muestra los tiempos de ejecución en cada máquina que se obtienen con la ecuación 1.6. La primer columna de la tabla contiene las máquinas. La segunda columna los tiempos de ejecución acumulados de todas las tareas asignadas a dicha máquina. La última fila contiene el *makespan*, el cual es el mayor de los tiempos acumulados.

Tabla 1.7: Tiempo de ejecución en cada máquina y *makespan* obtenido

Máquina	Tiempo total
M_1	37.75
M_2	44.22
M_3	23.24
Makespan	44.22

1.1.7. Cálculo de la función objetivo energía

Para el cálculo de la energía se utiliza la ecuación 1.5 la cual requiere los costos computacionales P, la máquina asignada m_j y el voltaje seleccionado v_p.

1.2. Complejidad del problema

En [Garey and Johnson, 1979] se demuestra que el problema es NP-Hard incluso para máquinas idénticas donde el tiempo de procesamiento de cada tarea no depende de la máquina asignada. El problema de la selección discreta de voltajes es un problema NP-Duro el cual se prueba por restricción al problema *discrete time-cost trade-off* ($DTCT$) el cual es NP-Duro. Entonces restringiendo el problema a *discrete voltage selection problem without overheads* ($DNOH$) en el cual todas las tareas requieren la unidad como tiempo de ejecución, se vuelve idéntico al problema $DTCT$ entonces, $DTCT \in DNOH$ se llega a la conclusión que $DNOH \in NP$ [Andrei et al., 2007].

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar métodos de optimización multiobjetivo exactos y metaheurísticos para el HCSP con DVFS. Estos métodos deben de aprovechar las características de la representación del problema para lograr un mejor desempeño.

1.3.2. Objetivos específicos

- Desarrollar un MILP.
- Desarrollar un Branch and bound.
- Desarrollar un método multiobjetivo de solución basado en algoritmos evolutivos.
- Una publicación en revista.

1.4. Justificación y beneficios

Como se mostró en la Tabla 1.1, la capacidad de procesamiento de los HPC crecen constantemente así como su consumo de energía. Por lo que es necesario la implementación de algoritmos que sean capaces de asignar las tareas de una manera óptima, para disminuir la energía consumida sin aumentar el tiempo de procesamiento. A demás la mejora de estos algoritmos benefician al medio ambiente al disminuir la emisión de CO_2 al requerir menos energía para ejecutar las tareas.

A pesar de que es un problema que desde hace tiempo fue formulado, recientemente ha empezado a recibir una gran atención debido al incremento del uso de los centros de datos y los sistemas de cómputo heterogéneo. También debido a la preocupación del área científica por el medio ambiente.

Hasta el momento no se ha reportado ningún modelo de programación lineal para el problema tratado que minimice ambos objetivos, por lo tanto se considera conveniente aportar cuando menos un modelo exacto al estado del arte.

1.5. Alcances y limitaciones

Limitaciones

- No existe un conjunto de instancias estándar.
- No existe un modelo de programación lineal contra cual compararse.
- La evaluación de los algoritmos se sujetarán a las instancias de prueba [Braunt et al., 2001].
- Se utilizará la biblioteca de CPLEX 12.5.

Alcances

- Se utilizarán el conjunto de instancias de [Braunt et al., 2001].
- Las tareas son independientes.
- No se considera la duplicación de tareas, ni el uso de algoritmos con programación paralela.

Capítulo 2

Marco Teórico

2.1. Complejidad computacional

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia los problemas de decisión, así como los recursos que requieren los algoritmos para resolverlos. Los problemas pueden clasificarse como P, NP, NP-completos y NP-duros.

2.1.1. Clase P

Es el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial por un algoritmo determinista. A los problemas que pertenecen a esta clase se les denomina tratables.

2.1.2. Clase NP

La clase NP es el conjunto de todos los problemas de decisión que se pueden verificar en tiempo polinomial con un algoritmo no determinista. Por verificar se entiende que se pueda generar en tiempo polinomial una solución candidata con un algoritmo no determinista y que se pueda verificar su factibilidad en tiempo polinomial con un algoritmo determinista.

Para probar que un problema de decisión pertenece a la clase NP se debe de realizar lo siguiente:

- Definir una estructura de datos para representar las soluciones candidatas.
- Construir un algoritmo aleatorio para generar una solución candidata.
- Construir un algoritmo determinista para verificar que una solución candidata cumple las condiciones especificadas en el problema.

2.1.3. Problema NP-Completo

Un problema B es NP-Completo si satisface dos condiciones. Si B pertenece a la clase NP y cada problema A en NP es reducible en tiempo polinomial a B [Sipser, 2006].

2.1.4. Problema NP-Hard

Numerosos problemas derivados de aplicaciones prácticas pertenecen a la clase NP-Hard de acuerdo a la teoría de la complejidad desarrollada por Garey [Garey and Johnson, 1979] que incluye en esta clase a los problemas para los que no existe actualmente algún algoritmo de tiempo polinomial capaz de resolverlos hasta la optimalidad. La dificultad para resolver estos problemas mediante algoritmos de tiempo polinomial ha impulsado el desarrollo de métodos de solución alternativos tales como los métodos heurísticos, los cuales ofrecen soluciones aproximadas en tiempos razonables con bajos requerimientos computacionales.

2.2. Métodos de optimización

Los métodos de optimización es una rama de las matemáticas que consiste en el uso de modelos matemáticos, estadísticos y algorítmicos con el objetivo de realizar un proceso de toma de decisiones. Frecuentemente se utilizan en el estudio de complejos sistemas reales, con la finalidad de mejorar u optimizar su funcionamiento. Estos métodos permiten el análisis de la toma de decisiones teniendo en cuenta la escasez de recursos, para determinar cómo se puede optimizar un objetivo definido, por ejemplo la maximización de los beneficios o la minimización de costos.

2.2.1. Métodos exactos

Los métodos exactos garantizan encontrar la mejor solución o solución óptima de todo el espacio de soluciones del problema.

Definición de un modelo de programación lineal (LP)

La definición de un modelo de programación lineal es la siguiente: El propósito general de la programación lineal (PL) consiste en la búsqueda del óptimo (en nuestro caso del mínimo) de una función de p variables x_j $(j = 1, 2, ..., p)$ ligadas por relaciones (igualdades o desigualdades) lineales llamadas condiciones.

$$min \quad z = \sum_{j=1}^{p} c_j x_j \tag{2.1}$$

Sujeto a:

$$\sum_{j=1}^{p} a_{ij} x_j \leq d_i \quad i = 1, 2, ..., m. \tag{2.2}$$

$$x_j \geq 0 \quad j = 1, 2, ..., p. \tag{2.3}$$

Donde c_j, a_{ij} y d_i son constantes conocidas para toda i y j; y x_j son variables de decisión no negativas.

Algoritmo branch & bound

Branch and Bound (B&B) es una técnica que se basa en la exploración de grafos.

El funcionamiento general del algoritmo B&B consiste en reducir el espacio de búsqueda del problema podando las zonas del árbol aún no exploradas que no pueden crear mejores soluciones que la solución en curso. Por lo tanto, el método de ramificación

y poda B&B busca siempre la mejor solución o conocido también como óptimo global y, no solamente una buena solución.

Explora un árbol comenzando a partir de un problema raíz y su región factible. Aplica funciones de acotación al problema raíz, para el que establece cotas inferiores y/o superiores. Si las cotas cumplen las condiciones que se han establecido, habremos encontrado la solución óptima del problema y la búsqueda termina. Si se encuentra una solución óptima para un sub-problema concreto, ésta será una solución factible para el problema completo, pero no necesariamente su óptimo global. Cuando en un nodo (sub-problema), su cota local es peor que el mejor valor conocido en la región, no puede existir un óptimo global en el sub-espacio de la región factible asociada a ese nodo y, por lo tanto, ese nodo puede ser eliminado ("podado"). En B&B, la búsqueda prosigue hasta que se examinan o "podan" todos los nodos, o bien se cumple algún criterio pre-establecido sobre el mejor valor encontrado y las cotas locales de los sub-problemas aún no resueltos.

2.3. Heurísticas

El uso de heurísticas es aplicable cuando el problema de optimización es NP-Duro. Dentro de este tipo de métodos se encuentran los algoritmos heurísticos y metaheurísticos. Los algoritmos heurísticos son utilizados para garantizar una solución de calidad en un tiempo razonable.

La palabra Heurística proviene del término griego *heuriskein*, que significa *"Hallar, inventar"*, según la Real Academia Española.

En [Zanakis, 1981] se presenta una de las definiciones más aceptada de heurística, *"procedimientos simples, a menudo basados en el sentido común, que se supone obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido."*

Los algoritmos heurísticos se caracterizan porque utilizan alguna propiedad de la estructura del problema para encontrar rápidamente una solución aproximada. Pero estos algoritmos no garantizan que la solución encontrada sea la óptima. El principal defecto de los algoritmos heurísticos es que no cuentan con mecanismos que les permitan escapar de los óptimos locales. Para resolver este problema se desarrollaron otro tipo de algoritmos de búsqueda inteligente a los cuales se les conoce como metaheurísticas, estos algoritmos guían a los métodos heurísticos para evitar el estancamiento en óptimos locales.

2.4. Metaheurísticas

El término de metaheurística está definido en [Glover, 1986] como *"un procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá del óptimo local"*. Una definición más completa es la descrita en [Kelly and Osman, 1996], la cual dice, *"las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos"*.

Algunos ejemplos de metaheurísticas son: optimización por colonia de hormigas (ACO), algoritmos evolutivos (EA), dentro de estos se incluyen los algoritmos genéticos

(GA) y algoritmos meméticos (MA), búsqueda local iterativa (ILS), búsqueda dispersa (SS), búsqueda tabú (TS), entre otros.

2.5. Algoritmos evolutivos

Los algoritmos evolutivos son métodos de búsqueda que se basan en la teoría neodarwiniana de la evolución de las especies, en donde la selección natural y la supervivencia de los individuos más fuertes o más aptos son los que logran tener descendencia.

Estos algoritmos son metaheurísticas de tipo poblacional debido a que emplean un conjunto de soluciones denominado población. La eficiencia de este tipo de metaheurísticas depende en gran medida de cómo se manipula la población, en este caso operadores que se inspiran en la teoría de la evolución.

Estos algoritmos fueron propuestos a finales de los años 50 en diversas publicaciones para resolver problemas de optimización combinatoria. En la década de los 60 L. Fogel [Fogel et al., 1966] establece las bases de la programación genética. En los años 70 Rechemberg [Rechemberg, 1973] introduce las estrategias evolutivas.

Los algoritmos evolutivos [Duarte et al., 2007] son metaheureurísticas guiadas por una búsqueda poblacional (conjunto de soluciones), en la que los individuos de la población más aptos (mejores soluciones encontradas) tienen mayor posibilidad de supervivencia así como de generar descendencia (soluciones derivadas).

Cada iteración del algoritmo evolutivo implica una selección competitiva que mantiene a las soluciones mas aptas. Las soluciones con una alta aptitud se recombinan con otras soluciones por medio del intercambio de partes entre ellos. La recombinación y mutación se utilizan para generar nuevas soluciones que exploran regiones del espacio de búsqueda en donde se pueden encontrar buenas soluciones.

2.5.1. Algoritmos genéticos

Los algoritmos genéticos son algoritmos evolutivos y heredan sus características. Involucran una estrategia adaptativa y una técnica de optimización global. Se identifican por elementos como la población inicial, una representación de las soluciones, una función para la evaluación de la calidad de los individuos y operadores genéticos como selección, cruza y mutación. Los algoritmos genéticos fueron introducidos por J. Holland [Duarte et al., 2007].

En los algoritmos genéticos la solución se representa mediante una cadena binaria a la cual se le denomina "cromosoma". A cada posición de la cadena se le denomina "gen" y al valor dentro de esta posición se le llama "alelo".

Los elementos que caracterizan a un algoritmo genético son los siguientes:

- **Población inicial.** Es un conjunto de soluciones generalmente aleatorias.
- **Representación.** Es el mapeo del conjunto de variables de decisión (fenotipo) en una cadena binaria (genotipo). En la actualidad el cromosoma es representado por una estructura de datos en particular.
- **Función de evaluación.** Determina la aptitud de los individuos de la población. Normalmente es una función que asigna un valor que representa la calidad del individuo. Cuanto mayor la aptitud mejor es la solución del individuo.
- **Operadores genéticos.** Son mecanismo probabilísticos que permiten obtener nuevos individuos y suelen ser independientes de la representación del problema. Los operadores más comunes son los siguientes.

- Selección: Mecanismo probabilista que favorece a los individuos con mayor aptitud para tener descendencia.
- Cruza: Intercambio de genes entre los padres para formar un nuevo individuo. Existen diferentes tipos de cruza.
- Mutación: Cambio aleatorio de los genes de un individuo. Permite diversificación y la exploración del espacio de soluciones.

A continuación se muestra el algoritmo genético general. Donde N es el tamaño de la población, P_{cruza} es la probabilidad de que los padres se crucen y $P_{mutación}$ es la probabilidad de que un individuo sea mutado.

Algorithm 1 Algoritmo Genético general

```
1: procedure GENETIC ALGORITHM(N, P_cruza, P_mutación)
2:     Población ← generarPoblaciónInicial(N)
3:     calcularAptitud(Población)
4:     while condiciónDeParo do
5:         Padres ← seleccionarPadres(Población)
6:         Hijos ← 0
7:         for Padre_1, Padre_2 ∈ Padres do
8:             Hijo_1, Hijo_2 ← Cruza(Padre_1, Padre_2, P_cruza)
9:             Hijos ← Mutación(Hijo_1, P_mutación)
10:            Hijos ← Mutación(Hijo_2, P_mutación)
11:        end for
12:        calcularAptitud(Hijos)
13:        S_mejor ← obtenerMejorSolución(Hijos)
14:        Población ← Reemplazar(Población, Hijos)
15:    end while
16:    return S_mejor
17: end procedure
```

2.6. Optimización multiobjetivo

El concepto de óptimo de pareto fue formulado por Vilfredo Pareto en 1896. Muchos problemas de la vida real involucran simultáneamente la optimización de diversos objetivos inmensurables y competitivos. Con frecuencia, no hay una única solución óptima, en cambio existe un conjunto de soluciones alternativas. Estas soluciones son óptimas en el amplio sentido de que ninguna solución en el espacio de búsqueda es superior a ellas cuando todos los objetivos son considerados. Estas son conocidas como las soluciones óptimas de Pareto.

En forma general un problema de optimización multiobjetivo (MOP, del inglés Multi-objective Optimization Problem) puede ser descrito como un vector de funciones f que mapea una tupla de m parámetros (variables de decisión) a una tupla de n objetivos.

$$min/max \quad y = f(x) = (f_1(x), f_2(x), ..., f_n(x)) \tag{2.4}$$

Sujeto a:

$$x = (x_1, x_2, ..., x_m) \in X \tag{2.5}$$

$$y = (y_1, y_2, ..., y_n) \in Y \tag{2.6}$$

Donde x es el vector de decisión, X es el espacio de parámetros, y es el vector de objetivos y Y es el espacio de objetivos.

El conjunto de soluciones de un problema de optimización multiobjetivo consiste de todos los vectores de decisión para los cuales los vectores de valores objetivos no pueden ser mejorados sin que se degrade el conjunto de soluciones, estos vectores son conocidos como el óptimo de Pareto. Matemáticamente, el concepto del óptimo de Pareto es el siguiente: Suponiendo un problema de minimización y considerando dos vectores de decisión $a, b \in X$, a domina $b (a \prec b)$, si y solo si

$$\forall i \in \{1, 2, ..., n\} : f_i(a) \leq f_i(b) \wedge \exists j \{1, 2, ..., n\} : f_j(a) \leq f_j(b). \tag{2.7}$$

Todos los vectores de decisión que no son dominados por ningún otro vector de decisión de un conjunto dado se dice que son no-dominados respecto a este conjunto. El vector de decisión que es no-dominado dentro del espacio de búsqueda se denota como el óptimo de Pareto y constituye el llamado Pareto-óptimo o frente óptimo de Pareto.

En [Pineda et al., 2014] encontramos las siguientes definiciones de óptimo de Pareto.

- **Definición 1.** Problema de Optimización Multiobjetivo.

  Dado un vector función $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_k(\vec{x})]$ y un espacio factible de soluciones Ω, el MOP consiste en encontrar un vector $\vec{x} \in \Omega$ que optimice el vector función $\vec{f}(\vec{x})$. Sin perdida de generalidad asumimos únicamente funciones de minimización.

- **Definición 2.** Dominancia de pareto. Un vector $\vec{x}$ domina a $\vec{x'}$ (denotado por $\vec{x} \prec \vec{x'}$) si $f_i(\vec{x}) \leq f_i(\vec{x'})$ para toda función i en $\vec{f}$ y existe al menos una i tal que $f_i\vec{x} < f_i\vec{x'}$.

- **Definición 3.** Óptimo de Pareto. Un vector $\vec{x^*}$ es óptimo de Pareto si no existe $\vec{x'} \in \Omega$ tal que $\vec{x'} \prec \vec{x^*}$.

- **Definición 4.** Conjunto de óptimo de Pareto. Dado un MOP, el conjunto de óptimo de Pareto es definido como $P^* = \left\{\vec{x^*} \in \Omega\right\}$.

- **Definición 5.** Frente de Pareto. Dado un MOP y su conjunto de óptimo de Pareto P^*, el frente de Pareto se define como $PF^* = \left\{\vec{f}(\vec{x}) | \vec{x} \in P^*\right\}$.

Cuando obtenemos soluciones del frente óptimo de pareto es difícil preferir una sobre otra si no tenemos información alguna sobre el problema. Por lo tanto, todas las soluciones del óptimo de Pareto son igualmente importantes. En un problema es importante encontrar tantas soluciones en el óptimo de Pareto como sea posible. En consecuencia obtenemos dos objetivos en optimización multiobjetivo:

- Encontrar un conjunto de soluciones que sean lo más cercanas al frente óptimo de Pareto.

- Encontrar un conjunto de soluciones tan diversas como sea posible.

2.6.1. Indicadores de calidad

El resultado de un algoritmo de optimización multiobjetivo es un conjunto de soluciones $A_1 \in \Omega$ de cardinalidad variable, si comparamos contra otro algoritmo de optimización que produce un conjunto de soluciones $A_2 \in \Omega$ surgen las siguientes interrogantes:

- ¿Cuándo podemos decir que A_1 es mejor que $A_2(A_1 \triangleleft A_2)$?
- Si A_1 es mejor ¿Qué tan mejor es respecto a A_2?
- Si $A_1 \not\triangleleft A_2$ y $A_1 \not\triangleright A_2$ ¿'En qué aspectos se diferencian y en qué cantidad?

De acuerdo a la relación de dominancia en conjuntos de aproximación [Zitzler et al., 2002] $(A_1 \triangleleft A_2)$ si y solo si toda las soluciones de A_2 son dominadas por alguno en A_1 y A_1 contiene por lo menos una solución que no es dominada por ninguna en A_2. Cuando esta condición no se cumple y $A_1 \not\triangleright A_2$ no se puede decir que ninguno de los conjuntos sea mejor que el otro, pero se pueden observar propiedades deseables como la diversidad de las soluciones o la cardinalidad del conjunto, para facilitar la comparación de conjuntos de soluciones se hace uso de los indicadores de calidad multiobjetivo.

Los indicadores de calidad nos permiten comparar el desempeño de algoritmos multiobjetivo. Los más populares son los indicadores de calidad unarios, también existen los binarios. Los indicadores miden diferentes características deseables de los algoritmos multiobjetivo.

Los indicadores de calidad unarios se pueden aplicar a un conjunto de aproximación sin depender de otro para su comparación. Tienen la desventaja de que requieren conocer previamente el espacio de soluciones o estimarlo, por ejemplo el frente de Pareto PF_{true} o sus límites superiores e inferiores. Los indicadores binarios requieren comparar dos conjuntos por ejemplo dado dos conjuntos de soluciones A_1 y A_2 queremos saber cuántas soluciones de A_1 son dominadas por A_2 y viceversa, la desventaja radica en que la comparación solamente puede ser entre dos conjuntos.

A continuación se describen dos indicadores de calidad del tipo unarios.

Hypervolumen

El Hypervolumen [Auger et al., 2009, Yang and Ding, 2007] también conocido como S-metric [Naujoks and Beume, 2005] es uno de los indicadores de calidad más usados en multiobjetivo, representa el volumen del espacio n–dimensional dominado por las soluciones en el conjunto de referencia A esto es un valor único conocido como Hypervolume y es una propiedad deseable que este se maximice.

El Hypervolumen del frente A es medido de acuerdo a un punto de referencia usualmente el anti óptimo global es decir una solución x que es dominada por todas las soluciones. Se aconseja que este punto de referencia sea el peor valor conocido en cada objetivo y agregarle la unidad [Naujoks and Beume, 2005].

Hay diferentes formas de calcular el Hypervolumen como se muestra en [Auger et al., 2009, Yang and Ding, 2007, Naujoks and Beume, 2005]. Está probado que el cálculo del Hypervolume es NP-Hard [Bringmann and Friedrich, 2009].

Generational distance

El indicador de *Generational Distance* (GD) nos permite medir la convergencia del algoritmo, representa la distancia entre el frente encontrado A y el frente verdadero PF_{true} y se define en [Veldhuizen and Lamont, 2000]como:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{2.8}$$

Donde n es el número de puntos en el frente encontrado en la generación actual y d_i es la distancia euclidiana del punto i al punto más cercano del PF_{true} una GD de 0 nos indica que el algoritmo ha encontrado puntos en el verdadero frente de Pareto.

Generalized spread

Generalized Spread (GS) es una extensión de la métrica de diversidad *Spread* de [Deb et al., 2002]. Esta métrica calcula la distancia entre soluciones consecutivas, pero solamente funciona para problemas con dos objetivos. En [Zhou et al., 2006] se propuso una extensión de esta métrica la cual calcula la distancia de un punto a su vecino mas cercano.

2.7. Algoritmos Multiobjectivo

En esta sección se presentan dos de los principales algoritmos multi-objetivo.

2.7.1. Non-Dominated sorting genetic algorithm II

El NSGA-II es un algoritmo de optimizacioń multiobjetivo propuesto por [Deb et al., 2000] como una mejora del NSGA [Deb et al., 2000], utiliza la estructura de los algoritmos genéticos y sus principales ideas son:

- Los mejores individuos jamás desaparecen de la población.
- En la selección de encontrarse dos soluciones no dominadas entre sí se prefiere la más diversa.

El NSGAII fue propuesto por Deb et al en [Deb et al., 2000] como una mejora al NSGAII agregando dos algoritmos que atacan la falta de elitismo, la complejidad de ordenamiento de frentes y la falta de diversidad: *fast_nondominated_sort* y *crowding_distance_assignament*.

El algoritmo general del NSGAII es el siguiente, en la línea 2 las soluciones son ordenadas por frentes en $\mathcal{F}$ y después en la línea 4 se calcula el *crowding distance* del frente $\mathcal{F}_i$. En la línea 5 se incluye el frente no dominado en la población. En la línea 7 ordena la población de acuerdo al operador de comparación *crowded*. En la siguiente línea se seleccionan los primeros N individuos de la población. El método *make–new–population* de la línea 9 aplica los operadores genéticos de selección, cruza y mutación. Finalmente pasa a la siguiente generación.

fast_nondominated_sort

Este algoritmo es un componente esencial del NSGAII, se encarga de ordenar el conjunto de soluciones P en diversos conjuntos de soluciones no dominadas F. Para esto cada solución debe ser comparada contra las demás soluciones en la población para saber si esta es dominada. Es un algoritmo que tiene un comportamiento elitista ya que ordena los frentes por conjuntos de soluciones F_i que dominan a los conjuntos $F_{i+1}, F_{i+2}, ..., F_{i+n}$.

A continuación se muestra el algoritmo *fast_nondominated_sort*.

Algorithm 2 Non Sorted Genetic Algorithm II

```
1: procedure NSGAII()
2:     F = fast–nondominated–sort(R_t)
3:     while |P_{t+1}| < N do
4:         crowding–distance–assignment(F_i)
5:         P_{t+1} = P_{t+1} ∪ F_i
6:     end while
7:     sort(P_{t+1}, ≥_n)
8:     P_{t+1} = P_{t+1}[0 : N]
9:     Q_{t+1} = make–new–population(P_{t+1})
10:    t = t + 1
11: end procedure
```

Algorithm 3 Fast nondominated sort algorithm

```
1: procedure FAST–NONDOMINATED–SORT(P)
2:     for ∀ ∈ P do
3:         for ∀q ∈ P do
4:             if (p ≺ q) then
5:                 S_p = S_p ∪ {q}
6:             else if q ≺ p then
7:                 n_p = n_p + 1
8:             end if
9:         end for
10:        if n_p = 0 then
11:            F_1 = F ∪ {p}
12:        end if
13:    end for
14:    i = 1
15:    while F_i ≠ 0 do
16:        H = 0
17:        for ∀p ∈ F_i do
18:            for ∀q ∈ S_p do
19:                n_q = n_q − 1
20:                if n_q = 0 then
21:                    H = H ∪ {q}
22:                end if
23:            end for
24:        end for
25:    end while
26: end procedure
```

Crowding_distance_assignment

Este algoritmo permite asignar un valor que mide la densidad de soluciones que se encuentran alrededor de una solución en particular. Este valor es un estimado del hypercubo más grande encerrando a la solución actual sin incluir otro miembro del frente encontrado.

A continuación se muestra el algoritmo *crowding_distance_assignment*.

Algorithm 4 Crowding distance assignment algorithm

```
1: procedure CROWDING_DISTANCE_ASSIGNMENT(𝒥)
2:     l = |𝒥|
3:     for ∀i in l do
4:         𝒥[i]_distance = 0
5:     end for
6:     for ∀objective in m do
7:         𝒥 = sort(𝒥, m)
8:         𝒥[1]_distance = 𝒥[l]_distance = ∞
9:         for i = 2 to l − 1 do
10:            𝒥[i]_distance = 𝒥_distance + 𝒥[i + 1].m − 𝒥[i − 1].m
11:        end for
12:    end for
13: end procedure
```

2.7.2. Algoritmos basados en descomposición

Los métodos de optimización tradicionales están basados en Dominancia de Pareto o en formas relajadas de dominancia para lograr la representación del frente de Pareto. Sin embargo, el desempeño de los métodos de optimización multi-objetivo tradicionales disminuyen en problemas con más de tres objetivos. La descomposición de un problema multi-objetivo es un acercamiento que transforma un problema multi-objetivo en muchos problemas de optimización de un solo objetivo, evitando usar la dominancia.

La descomposición de problemas de optimización es una antigua idea que aparece en los siguientes trabajos [Jin et al., 2001, Hughes, 2003, Zhang and Li, 2007]. Este enfoque transforma el problema original en subproblemas y los resuelve por separado. Los algoritmos de descomposición multi-objetivo están basados en algoritmos de búsquedas poblacionales como algoritmos evolutivos, colonia de hormigas y enjambres de partículas.

Esta estructura requiere que cada vector $\vec{w}_i$ similar se encuentre agrupado en un conjunto W. Esta estructura funciona en cuatro pasos. El primer paso asigna una solución inicial $\vec{x}_i$ a cada vector $\vec{w}_i$. El segundo paso selecciona un vector $\vec{w}_i$ del conjunto W y lo optimiza, con cualquier método de optimización, con la mejor solución $\vec{x}_i$ conocida para ese vector. El tercer paso utiliza la nueva solución encontrada $\vec{x}'$ para actualizar la mejor solución encontrada en el agrupamiento de vectores de peso con w_i. El cuarto paso verifica si el criterio de parada se ha alcanzado, si no, regresa al segundo paso.

A continuación se muestra la estructura general del algoritmo de optimización multi-objetivo basado en descomposición [Pineda et al., 2014].

Uno de los principales algoritmos de descomposición es el *Multi-Objective Evolutionary Algorithm Based on Decomposition (MOEA/D)*, fue propuesto originalmente en [Zhang and Li, 2007] y modificado en [Zhang and Li, 2009] para aplicar diferentes acercamientos en los problemas mono objetivo con prioridad en los más difíciles, en la última versión este algoritmo utiliza la evolución diferencial [Zhang et al., 2009].

Algorithm 5 Multiobjective Evolutionary Algorithm Based on Decomposition

1: **procedure** MOEAD(Vectores $\vec{w}_i$ que estén en C_z)
2: **for** $\forall \vec{w}_i \in W$ **do**
3: $\vec{x}'_i \leftarrow$ una solución inicial para el vector $\vec{w}_i$
4: **end for**
5: **repeat**
6: Seleccionar un vector $\vec{w}_i \in W$
7: $\vec{x}' \leftarrow$ Optimizar($\vec{x}_i$)
8: **for** $\forall \vec{w}_j \in C_z$ **do**
9: **if** $\vec{x}' < \vec{x}_j$ **then**
10: $\vec{x}_j \leftarrow \vec{x}'$
11: **end if**
12: **end for**
13: **until** Criterio de parada
14: **end procedure**

Capítulo 3

Estado del arte

En esta sección se describen los trabajos relacionados. Actualmente se han aplicado diversas metaheurísticas para resolver este problema y variantes del mismo. A continuación se describen algunos trabajos.

3.1. Algoritmos heurísticos

"New Heuristic for Scheduling of Independent Tasks in Computational Grid", (2011). Anand K. Chaturvedi y Rajendra Sahu. Solamente minimiza el *makespan* y no considera la energía. Utiliza las instancias de Braunt.

Implementa una heurística de dos fases. En la primera fase asigna las tareas a las máquinas dependiendo de los tiempos de ejecución mínimos. En la segunda fase las máquinas sobrecargadas pasan tareas a aquellas máquinas que no lo están. La compara contra las siguientes heurísticas:

- *Opportunistic Load Balancing (OLB)*
- *Minimum Execution Time (MET)*
- *Minimum Completion Time (MCT)*
- *Switching Algorithm (SA)*
- *k–Percent Best (kPB)*
- *Work Queue (WQ)*
- *Min–min*
- *Max–min*
- *Largest Job in Faster Resource – Shorter Job on Faster Resource (LJFR – SJFR)*
- *Suffrage*

3.2. Algoritmos metaheurísticos

"Independent Task Scheduling in Grid Computing Based on Queen-Bee Algorithm", Zahra Pooranian, Mohammad Shojafar y Bahman Javadi, (2012). Solamente considera el *makespan*. Utiliza instancias que tienen 50, 100, 300, 500 tareas, y 10, 20 y 30 máquinas. Implementa el algoritmo *Queen–Bee*, este algoritmo se base en los GA. Tiene dos diferencias principales, en la selección de los padres solamente se selecciona uno que se conoce como *Queen Bee*. La segunda diferencia es que mantiene individuos malos dentro de la población.

La implementación del algoritmo *Queen–Bee* lo compara contra las siguientes metaheurísticas:

- *Genetic Algorithm*
- *Simulated Annealing*
- *GSA Genetic + Simulated Annealing*
- *PSO-SA*

"New Hybrid Algorithm for Task Scheduling in Grid Computing to Decrease Missed Task", Z. Pooranian, A. Harounabadi y M. Shojafar, (2011). Solamente considera el *makespan*. Utiliza instancias que tienen entre 20 y 60 tareas, 20 máquinas. En el algoritmo propuesto utilizan una nueva función para calcular la aptitud de un cromosoma.

$$Fitness(ch_i) = \frac{1}{makespan(che_i)} + \frac{1}{missTask * MD} + \frac{1}{LB_i} \tag{3.1}$$

Donde $missTask$ es el número de tareas que no se encuentran en el cromosoma. Donde MD es el tiempo límite de las tareas. Y LB_i es el balance de la carga total en las máquinas. Implementa las siguientes metaheurísticas:

- *Genetic Algorithm (GA).*
- *Gravitational Emulation Local Search (GELS).*
- *Genetic–Simulated Annealing (GSA).*
- *GA–GELS.*

"Heuristics Based on Partial Enumeration for the Unrelated Parallel Processor Scheduling Problem", E. Mokotoff y J. L. Jimeno, (2002). En este artículo se presenta un MILP que resuelve el problema de *makespan*. Presenta tres heurísticas que combinan el MILP con un B&B.

En la siguiente tabla se muestra un resumen de los trabajos relacionados.

Tabla 3.1: Resumen de los trabajos relacionados

Propuesta	Tipo de Tarea	Makespan	Energía	Exacto	Instancias
New Heuristic for Scheduling of Independent Tasks in Computational Grid (2011).	Independientes con *deadline* sin IDLE	✓			Braunt
Independent Task Scheduling in Grid Computing Based on Queen-Bee Algorithm (2012).	Independientes con *deadline* sin IDLE	✓			Propias
New Hybrid Algorithm for Task Scheduling in Grid Computing to Decrease Missed Task (2011).	Independientes con *deadline* sin IDLE	✓			Propias
Heuristics Based on Partial Enumeration for the Unrelated Parallel Processor Scheduling Problem (2002).	Indpendientes sin IDLE	✓		✓	Propias
Energy-Aware Grid Scheduling of Independent Tasks and Highly Distributed Data (2013).	Independientes con IDLE	✓	✓		Propias
Estrategias de Búsqueda Local para el Problema de Programación de Tareas en Sistemas de Procesamiento Paralelo (2013).	Dependientes con IDLE	✓	✓	✓	Mateusz

Como se puede apreciar en la tabla anterior existen pocas propuestas que minimizan la energía consumida y tampoco existe un conjunto de instancias estándar.

Capítulo 4

Métodos exactos propuestos

Los métodos exactos que se trabajan en este proyecto son los modelos de programación lineal entera mixta y *Branch and Bound* (B&B), ambos con un enfoque jerárquico.

Como primera parte de la propuesta de este proyecto se presenta primero el modelo de programación lineal entera mixta.

4.1. Modelo de programación lineal entera mixta

En esta sección se presenta la descripción de los modelos de programación lineal entera mixta para el problema tratado. Este modelo es de dos fases, dando prioridad al cálculo del *makespan*, después de obtener el *makespan* se calcula la energía mínima. A continuación se describe el modelo ($M1$) de la fase I tomado de [Mokotoff and Jimeno, 2002]. Después se presenta el modelo implementado ($M2$) de la fase II.

4.1.1. Descripción del modelo para *makespan* (Fase I)

En el modelo $M1$ se define una variable binaria x_{ij}. Esta variable toma el valor de 1 si la tarea i se ejecuta en la máquina j. La variable y almacena el *makespan*. Donde P_{ij} es el tiempo de procesamiento de la tarea i en la máquina j.

$$\min y \tag{4.1}$$

Sujeto a:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, ..., n. \tag{4.2}$$

$$\sum_{i=1}^{n} P_{ij} x_{ij} \leq y, \quad j = 1, ..., m. \tag{4.3}$$

$$x_{ij} \in \{0, 1\} \tag{4.4}$$

$$y \in \left\{\mathbb{R}^{+}\right\} \tag{4.5}$$

En donde 4.1 representa la función objetivo a minimizar. La restricción 4.2 relacionada con la asignación permite seleccionar una sola máquina para ejecutar una sola

tarea. La restricción 4.3 asocia al *makespan* el máximo del total de carga asignada a la máquina j. Las restricciones 4.4 y 4.5 indican el rango de las variables.

4.1.2. Descripción del modelo para energía (Fase II)

El objetivo del segundo modelo es obtener una asignación de voltajes que minimice el consumo de energía. El *makespan* obtenido en $M1$ se establece como límite en la restricción 4.9(13) de $M2$. Se agrega la restricción que permite calcular el tiempo de cada máquina para evitar que sobrepase el *makespan* establecido en $M1$. En este modelo se define una variable binaria x_{ijk}, la cual toma el valor de 1 si la tarea i se ejecuta en la máquina j con la configuración k.

$$\text{mín} \sum_{j=1}^{m}\sum_{i=1}^{n}\sum_{k=1}^{K}\frac{P_{ij}}{v_{kj}}V_{kj}^2 x_{ijk} \tag{4.6}$$

Sujeto a:

$$\sum_{j=1}^{m}\sum_{k=1}^{K} x_{ijk} = 1, \quad i = 1, ..., n. \tag{4.7}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{K}\frac{P_{ij}}{v_{kj}}x_{ijk} \leq y, \quad j = 1, ..., m. \tag{4.8}$$

$$y \leq makespan. \tag{4.9}$$

$$x_{ijk} \in \{0, 1\} \tag{4.10}$$

$$y \in \left\{\mathbb{R}^+\right\} \tag{4.11}$$

La función objetivo representada por 4.6 usa la ecuación 1.5 para el calculo de la energía consumida. La restricción 4.7 de asignación, permite seleccionar la configuración del voltaje en una máquina cuando ejecuta una tarea. La restricción 4.8 obtiene el tiempo total de cómputo de la máquina j. Las restricciones 4.10 y 4.10 indican el rango de las variables.

4.2. Descripción del *Branch and Bound*

El *Branch and bound* esta implementado en dos fases. Esto es, en la primer fase obtiene la asignación de tareas/máquinas que minimiza el *makespan* y en la segunda fase encuentra la mejor asignación de voltajes que minimice la energía a las asignaciones de máquinas establecidas.

En el *branch and bound* de *makespan* cada nivel del árbol es una tarea y cada nodo del árbol contiene la máquina que ejecutara dicha tarea. Por lo tanto el primer nivel árbol (nodo raíz) es la primer tarea, el segundo nivel del árbol corresponde a la segunda tarea y así sucesivamente. Igualmente en el branch and bound de energía cada nivel del árbol representa una tarea, pero cada nodo del árbol contiene la asignación del nivel de voltaje asignado para ejecutar la tarea con la máquina previamente seleccionada.

El *branch and bound* implementado funciona de la siguiente manera. Se inicia con una solución en la cual todas las tareas se ejecutan en la primer máquina y se establece como mejor solución, y el valor del *makespan* de la solución se establece como límite superior del árbol. Después se inicia la exploración del árbol asignando en cada nivel una máquina y calculando parcialmente el *makespan* de la solución hasta el nivel alcanzado.

Si la solución mejora o es menor que el límite superior entonces continua explorando por la rama actual hasta llegar al último nivel en cual se guarda la solución actual como la mejor solución encontrada y se actualiza el límite superior. Una vez que se llega al último nivel se inicia el proceso de retro propagación en el cual se retrocede un nivel para seguir explorando las demás ramas que no han sido recortadas. El *branch and bound* termina hasta que todas las ramas hallan sido exploradas o recortadas. La mejor solución encontrada es enviada al *brand and bound* de energía.

El *branch and bound* de energía establece como límite superior la energía consumida al aplicar el máximo nivel de configuración de voltaje a cada tarea. Realiza el mismo procedimiento que el *branch and bound* de *makespan* pero buscando las asignaciones de voltajes y calculando parcialmente la energía y el *makespan*. Si la rama actual disminuye la energía pero aumenta el *makespan* entonces es recortada, solamente explora aquellas ramas en las que la energía disminuye y en las que el *makespan* se mantenga igual o sea menor que el encontrado por la fase anterior.

La Figura siguiente muestra una ejecución del *branch and bound* de *makespan* en donde se van generando todas las posibles asignaciones de máquinas para cada tarea y se van recortando aquellas ramas que van empeorando la solución. Los nodos con lineas gruesas representan la mejor solución encontrada.

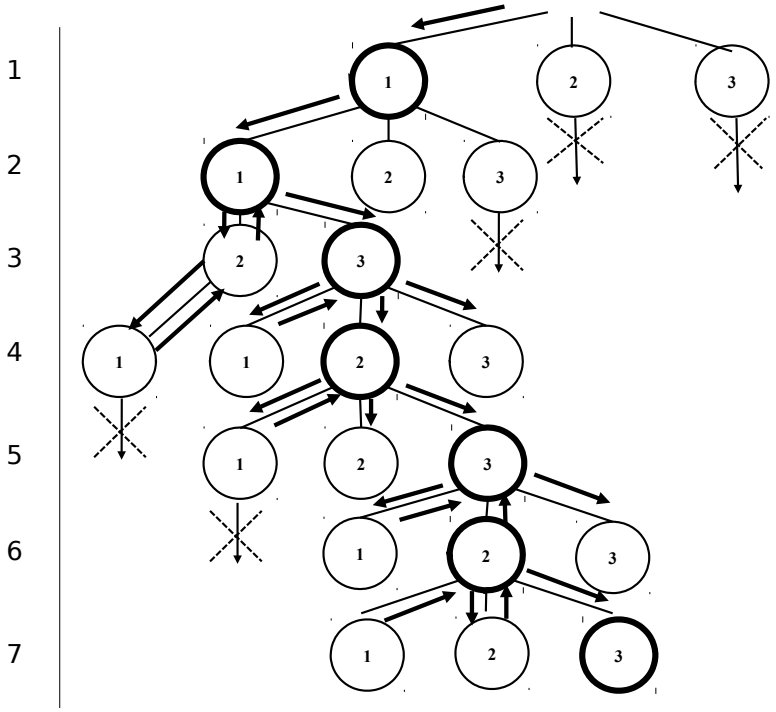

Figura 4.1: Representación gráfica del *Branch and bound* de *makespan*.

Capítulo 5

Métodos propuestos

Las aportaciones más relevantes de este proyecto consisten en el desarrollo de métodos de solución exacta y aproximada para el problema HCSP con DVFS. En esta sección se describe la metodología general para el desarrollo de estos métodos.

Para el desarrollo de algoritmos metaheurísticos se utilizo un enfoque genético y la estructura del *Non–Dominated Sorting Genetic Algorithm II* (NSGA–II) el cual ya se ha aplicado para resolver un problema de la misma familia obteniendo buenos resultados.

Además se aplico la metaheurística *Multiobjective Evolutionary Algorithm Based on Decomposition* (MOEAD/D) que se encuentra en la biblioteca de jmetal 4.5.

A continuación se da una descripción general de cada uno de los elementos que se han desarrollado.

5.1. Representación del problema

En los algoritmos genéticos al vector de las variables de decisión se le conoce como cromosoma. El cromosoma mapea el conjunto de variables de decisión (fenotipo) en una cadena binaria (genotipo). En la actualidad el cromosoma es representado por una estructura de datos en particular.

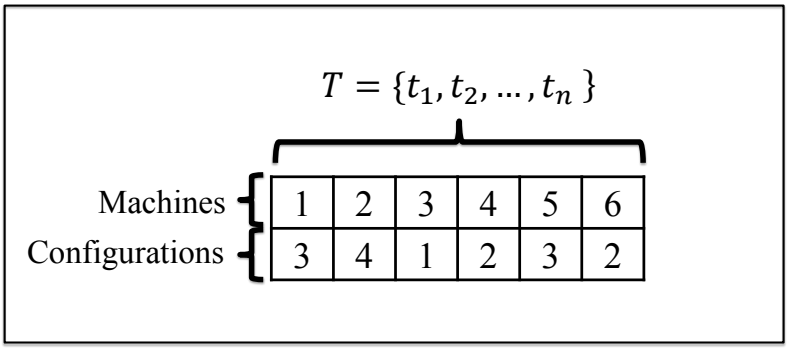

Figura 5.1: Representación del cromosoma.

La representación del cromosoma en la Fig. 5.1 asigna las máquinas y voltajes a cada tarea en T. El genotipo esta formado por el gen de máquinas y el gen de voltajes. Esta representación es la que se utiliza en los algoritmos implementados.

5.2. Implementación genética

Los operadores genéticos que a continuación se describen son implementados en el algoritmo propuesto utilizando la estructura del NSGA–II.

5.2.1. Selección

Para el operador de selección se seleccionaron del frente cero los individuos de mayor diversidad, si el número de padres no alcanza entonces se toman del siguiente frente hasta que se obtiene el número de padres deseados. Después se seleccionan aleatoriamente dos padres para realizar la cruza y generar dos hijos.

5.2.2. Cruza

Los operadores descritos a continuación son los implementados en el algoritmo del NSGA–II, los cuales modifican los operadores genéticos.

- C1: Cruza uniforme.
- C2: Cruza en punto medio.
- C3: Cruza en dos puntos.
- C4: Cruza en multipuntos aleatorios.

Cruza uniforme (C1)

En la cruza uniforme cada gen del padre tiene el 50 % de probabilidad de pasar al hijo.

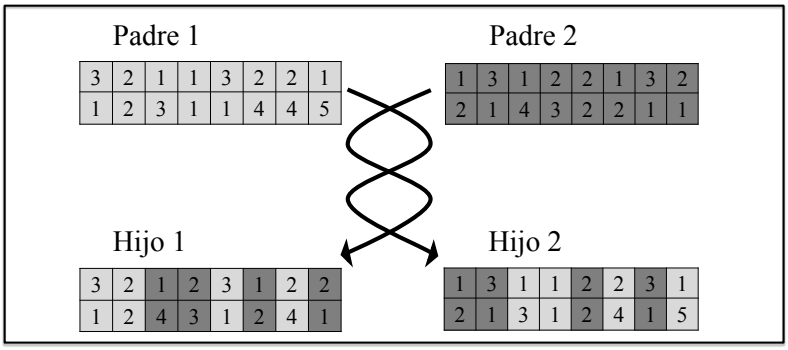

Figura 5.2: Representación de la cruza uniforme.

Cruza en punto medio (C2)

Esta cruza selecciona el punto medio del cromosoma y pasa cada mitad del cromosoma a cada hijo.

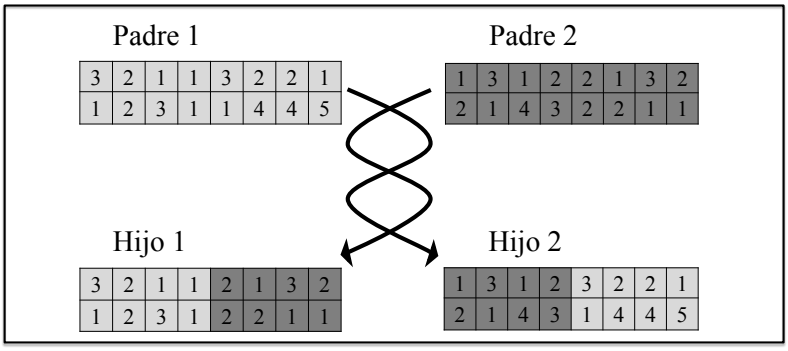

Figura 5.3: Representación de la cruza en el punto medio.

La Fig. 5.3 muestra que el primer segmento del Padre 1 pasa al Hijo 1 y el segundo segmento pasa al Hijo 2. El Primer segmento del Padre 2 pasa al Hijo 2 y el segundo segmento pasa al Hijo 1 completando ambos hijos.

Cruza en dos puntos (C3)

Divide el cromosoma de cada padre en tres segmentos y cada segmento pasa a cada uno de los hijos.

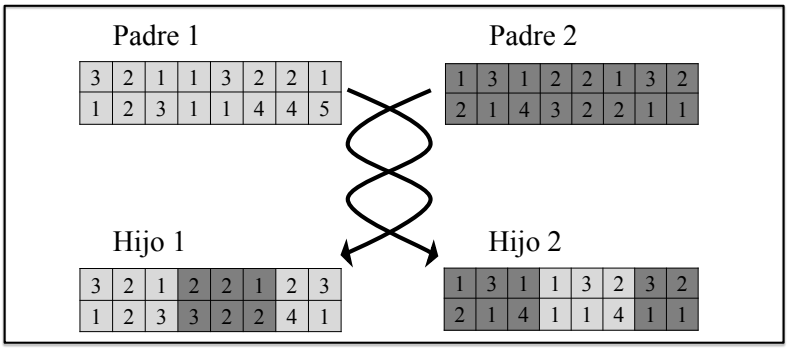

Figura 5.4: Representación de la cruza en dos puntos.

En la Fig. 5.4 cada Padre se divide en tres segmentos. El primer y tercer segmento del padre 1 pasan al Hijo 1 y el segundo segmento pasa al Hijo 2. Del Padre 2 el segundo segmento pasa al Hijo 1, el primer y tercer segmento pasan al Hijo 2, con esto los dos hijos quedan con sus genes completos.

Cruza en multipuntos aleatorios (C4)

En esta cruza se selecciona aleatoriamente un número el cual indica en cuantos segmentos se dividirá el cromosoma.

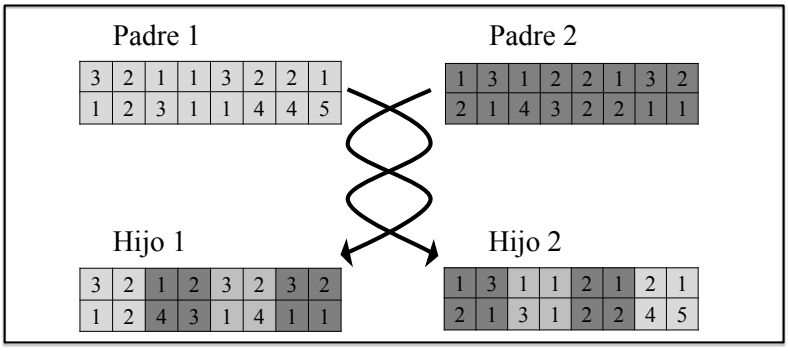

Figura 5.5: Representación de la cruza múltiples puntos aleatorios.

La Fig. 5.5 muestra que cada padre se dividió en cuatro segmentos y cada segmento se va intercalando entre el Hijo 1 y el Hijo 2.

5.2.3. Mutación

Se implementaron diferentes tipos mutaciones.

M1: Mutación del 10 %

Esta mutación modifica aleatoriamente el 10 % del cromosoma.

M2: Mutación del 5 %

Cada gen tiene una probabilidad del 5 % de ser modificado.

M3: Mutación de carga

Esta mutación esta basada en la idea del balanceo de cargas, esto quiere decir que las máquinas ejecuten la misma cantidad de carga de trabajo. En este caso solamente mueve una tarea que se encuentra en una máquina que genera el *makespan* a otra máquina, con la intención de reducir la carga de la máquina y así reducir el *makespan*.

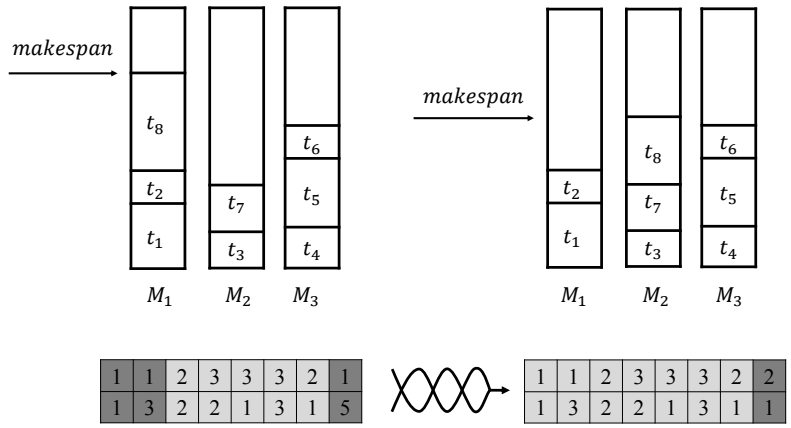

Figura 5.6: Representación del balanceo de cargas.

La Fig 5.6 muestra la máquina M_1 que esta generando *makespan* por lo tanto se selecciona aleatoriamente una de las tareas que esta ejecutando t_1, t_2 y t_8. En este caso se selecciona la tarea t_8 y es cambiada aleatoriamente a otra máquina (m_2). Aleatoriamente se cambia la configuración de voltaje quedando en 1, lo que disminuye el *makespan* pero aumentando la energía consumida.

En el algoritmo NSGAII se modifico el método de $make - new - population$ en el cual se agregaron los operadores genéticos implementados.

Algorithm 6 Non Sorted Genetic Algorithm II

1: **procedure** NSGAII()
2: $\mathcal{F}$ = fast–nondominated–sort($\mathcal{R}_t$)
3: **while** $|P_{t+1}| < N$ **do**
4: crowding–distance–assignment($\mathcal{F}_i$)
5: $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
6: **end while**
7: sort($P_{t+1}, \geq_n$)
8: $P_{t+1} = P_{t+1}\,[0 : N]$
9: Q_{t+1} = make–new–population(P_{t+1})
10: t = t + 1
11: **end procedure**

El MOEA/D utiliza una función de descomposición para el cálculo de la aptitud de un individuo. En la siguiente tabla se muestran las nueve funciones implementadas las cuales se encuentran en la literatura, además de una función propuesta (B).

Tabla 5.1: Funciones de descomposición

Indice	Nombre	Función
A	Tchebycheff	$U = \max_i \{w_i[F_i(x) - F_i^o]\}$
B	ITCMLog	$U = \sum_{i=1}^{k} \frac{log_2(w_i f_i(\vec{x}))}{log_2 3}$
C	Weighted sum	$U = \sum_{i=1}^{k} w_i f_i(\vec{x})$
D	Weighted exponential	$U = \sum_{i=1}^{k} (e^{pwi} - 1)e^{pf_i(x)}$
E	Weighted min–max	$U = \max_i \{w_i[F_i(x) - F_i^o]\}$
F	Weighted product	$U = \prod_{i=1}^{k} [F_i(x)]^{w_i}$
G	Nash	$U = \prod_{i=1}^{k} [s_i - F_i(x)]$
H	Augmented Tchebycheff	$U = max\{w_i[f_i(\vec{x} - f_i^o)]\} + \rho \sum_{j=1}^{k} (f_j(\vec{x}) - f_j^o)$
I	Augmented Tchebycheff Kaliszewski	$U = max\left\{w_i[f_i(\vec{x} - f_i^o) + \rho \sum_{j=1}^{k} (f_j(\vec{x}) - f_j^o)]\right\}$

Capítulo 6

Resultados experimentales

En esta sección se detallan las condiciones de experimentación y se reportan los resultados experimentales obtenidos por las implementaciones realizadas. La experimentación se divide en dos secciones, en la primera se comparan los métodos exactos y en la segunda los métodos aproximados.

6.1. Equipo utilizado

La evaluación experimental fue realizada en una computadora con arquitectura x86-x64, procesador Core i7 a 2.40 GHz y 6 GB en RAM. El sistema operativo utilizado fue Windows 8.1. El lenguaje utilizado para implementar los algoritmos fue Java utilizando el jdk 8. Se utilizó la biblioteca de jmetal 4.5 que está desarrollada en Java para calcular los indicadores de calidad y para el algoritmo MOEA/D. Para la implementación del modelo de programación lineal se utilizó la Biblioteca de CPLEX 12.1.

6.2. Instancias

La tabla 6.1 muestra el conjunto de instancias que fueron proporcionadas por el Dr. Johnatan Pecero Sanchez de la Universidad de Luxemburgo, este conjunto cuenta con 40 instancias [Pineda et al., 2013], a las cuales les fueron removidas las precedencias. La tabla 6.2 muestra el conjunto de 12 instancias proporcionadas en [Braunt et al., 2001].

Tabla 6.1: Conjunto de cuarenta instancias

Ahmad_3_9_28	heteroparjorgebarbosa2	Linshan_4_9_38	sample_3_8_100
Bittencourt_3_9_184	Ijaz_3_10_133	Liu_2_8_364	sample_4_11_25
Cao_3_10_536	Ilavarasan_3_10_77	Mohammad_2_11_64	sample_8_3_100
Ching_3_10_84	Ilavarasan_3_11_27	Munir_3_10_76	Tao_3_10
Demiroz_3_7_47	Ilavarasan_3_15_114	Rahmani_3_7	TOPCUOGLU_3_10_80
Eswari_2_11_61	IlavarasanIJCSIT_3_10	SahA_3_11_131	Topcuoglu_3_8_51
Gulzar_3_10_124	Kang_3_10_76	SahB_3_6_76	Xu_3_8_66
Hamid_3_10	Kang_3_10_84	Samantha_5_11_31	YCLee_3_8_80
Hernandez_3_10_70	Kuan_3_10_28	sample_3_10	Yu_4_10
heteroparjorgebarbosa1-3	Liang_3_10_80	sample_3_13	Zhao_3_10_143

Tabla 6.2: Conjunto de 12 instancias de [Braunt et al., 2001]

u_c_hihi.0	u_i_lohi.0
u_c_hilo.0	u_i_lolo.0
u_c_lohi.0	u_s_hihi.0
u_c_lolo.0	u_s_hilo.0
u_i_hihi.0	u_s_lohi.0
u_i_hilo.0	u_s_lolo.0

6.3. Experimentación de métodos exactos.

Actualmente no se han proporcionado instancias con valores óptimos para el problema multi-objetivo que se aborda en este proyecto. Por lo que para las pruebas de los métodos exactos se compararon los resultados que se generaron con ambas implementaciones. En esta experimentación se utilizó el conjunto de las cuarenta instancias.

En la siguiente tabla la primer columna contiene el nombre de la instancia. La segunda columna muestra el *makespan* (M) obtenido por el MILP. La tercer columna contiene la energía (E) obtenida por el MILP. La cuarta columna muestra el tiempo (T) que tardo el MILP en resolver la instancia, el tiempo se muestra en segundos. La quinta, sexta y séptima columna muestran la misma información para el *branch and bound*.

Tabla 6.3: Resultados experimentales de las algoritmos exactos

Instancia	MILP			B&B		
	M	E	T	M	E	T
Ahmad_3_9_28	13	79.57619	0.337	13	123.7144444	0.031
Bittencourt_3_9_184	89	454.775	0.061	89	806.7544118	0.016
Cao_3_10_536	379	1952.1	0.088	379	3768.310833	0
Ching_3_10_84	36	218.842857	0.058	36	334.7062092	0.032
Demiroz_3_7_47	19	217.4375	0.088	19	189.835	0
Eswari_2_11_61	55.5	186.3	0.043	55.5	293.3125	0.015
Gulzar_3_10_124	63	670.6875	0.067	63	622.93	0.031
Hamid_3_10	58	661.5	0.094	58	581.3425	0.016
Hernandez_3_10_70	33	339.9375	0.075	33	335.0325	0.016
heteroparjorgebarbosa1-3	27	581.875	0.076	27	338.1330882	0.015
heteroparjorgebarbosa2	35	283.642857	0.08	35	450.6722222	0.094
Ijaz_3_10_133	46	486.9375	0.083	46	455.415	0.015
Ilavasaran_3_10_77	36	388.9375	0.08	36	337.8007143	0
Ilavasaran_3_11_27	14	84.514286	0.047	14	142.135	0.016
Ilavasaran_3_15_114	79	771.75	0.08	79	802.0475	0.359
IlavasaranIJCSIT_3_10	36	388.9375	0.083	36	337.8007143	0
Kang_3_10_76	36	388.9375	0.101	36	337.8007143	0
Kang_3_10_84	36	388.9375	0.083	36	337.8007143	0.015
Kuan_3_10_28	11	134.75	0.088	11	103.8727941	0
Lian_3_10_80	36	388.9375	0.074	36	337.8007143	0
Linshan_4_9_38	14	229.6875	0.073	14	167.5239706	0.016
Liu_2_8_364	199	1592.5	0.065	199	1045.5875	0
Mohamad_2_11_64	55.5	186.3	0.034	55.5	293.3125	0.016
Munir_3_10_76	36	388.9375	0.076	36	356.9344444	0
Rahmani_3_7	912	5179.039286	0.067	912	9096.633464	0
SahA_3_11_131	48	462.4375	0.089	48	478.4575	0.172
SahB_3_6_76	22	136.080357	0.044	22	180.1763575	0
Samantha_5_11_31	10	90.95	0.06	10	149.05	0
Sample_3_10	63	655.375	0.064	63	608.5541667	0.015
Sample_3_13	49	627.8125	0.085	49	473.46	0.078
Sample_3_8_100	31	407.3125	0.05	31	284.715	0
Sample_4_11_25	11	80.875	0.068	11	134.9233333	0.016
Sample_8_3_100	31	407.3125	0.051	31	284.715	0
Tao_3_10	72	713.5625	0.074	72	730.98	0.016
TOPCUOGLU_3_10_80	36	388.9375	0.068	36	337.8007143	0
TOPCUOGLU_3_8_51	20	121.235714	0.046	20	199.9875	0
Xu_3_8_66	23.31	130.442571	0.053	23.31	201.145565	0.015
YCLee_3_8_80	29	161.835714	0.05	29	247.364134	0
Yu_4_10	28	388.9375	0.079	28	350.6547222	0
Zhao_3_10_143	91	937.125	0.111	91	823.5044118	0.016
Promedio	72.95775	558.9002458	0.077325	72.95775	687.0674465	0.025775

De la tabla anterior podemos apreciar que el Modelo y el B&B obtienen el mismo promedio de *makespan* pero el modelo genera menor energía. En cuanto al tiempo de ejecución el B&B es ligeramente más rápido. El *branch and bound* genera una menor energía ya que la estructura no permite cambiar la asignación de tareas/máquinas que genera el *branch and bound* de *makespan*. En cambio el modelo de programación lineal entera puede realizar estos cambios si encuentra una asignación diferente de tareas/máquinas/configuración que genere el mismo *makespan* pero con un menor consumo de energía.

6.4. Experimentación de métodos aproximados

En las experimentaciones de las metaheurísticas consistió en diez corridas independientes por instancia. Esto es el algoritmo NSGAII se ejecuto diez veces con cada combinación de cruza y mutación. Para el algoritmo MOEA/D se ejecuto diez veces por cada función de descomposición. Con los resultados obtenidos de cada instancia se juntaron para generar un frente de referencia el cual es utilizado en los indicadores de calidad de *Generational Distance* y *Generalized Spread*. Además se guardaron los peores valores encontrados en cada objetivo de todas las corridas para ser utilizadas como punto de referencia en el indicador de *Hypervolumen*, esto es debido a que el *Hypervolumen* utiliza un punto de referencia que contenga los valores extremos de cada objetivo.

A las diez salidas que generó cada instancia se les aplicaron los indicadores de calidad, después se calculo el promedio por instancia. Este promedio es utilizado en las comparaciones experimentales que se muestran en la siguientes secciones.

6.4.1. Experimentación de cruzas y mutaciones implementadas en el NSGA–II

La experimentación consistió en ejecutar diez veces cada metaheurística con cada instancia. Después se aplicó el indicador de calidad de Hypervolumen a cada frente de Pareto y se obtuvo el promedio de los resultados correspondientes a cada instancia.

En esta experimentación se midió el rendimiento de cuatro cruzas y tres mutaciones. Se probó la combinación de cada cruza con cada mutación. Para la experimentación cada instancia se resolvió diez veces y a los frentes obtenidos se les aplicaron los indicadores de calidad para medir el rendimiento de cada combinación de mutación y cruza.

En la tabla se muestran las combinaciones de cruzas y mutaciones que se utilizaron para las experimentaciones.

Tabla 6.4: Combinación de cruzas y mutaciones.

Cruza/Mutación	M1: Mutación del 10 %	M2: Mutación del 10 %	M3: Mutación de carga
C1: Cruza uniforme	C1M1	C1M2	C1M3
C2: Cruza punto medio	C2M1	C2M2	C2M3
C3: Cruza en dos puntos	C3M1	C3M2	C3M3
C4: Cruza en multipunto aleatorio	C4M1	C4M2	C4M3

A continuación se muestran las gráficas de los resultados obtenidos para *Dominated Hypervolumen, Generational Distance* y *Generalized Spread*. En las gr´aficas el eje Y representa valor del indicador de calidad, mientras que el eje X representa el número de instancia utilizada. Abajo del eje X cada gráfica tiene etiquetas de colores que representan la combinación de cruza y mutación las cuales siguen el formato *CxMy* donde la x representa el número de la cruza y y el número de mutación utilizada.

En la figura 6.1 se muestra la gráfica de los resultados del *Hypervolumen*. En donde las lineas superiores corresponden a las combinaciones de cruza y mutación C1M1, C1M2, C1M3, C4M1, C4M2 y C4M3. Las lineas inferiores corresponden a las combinaciones de cruza y mutación restantes. Por lo que al tratarse del indicador de hypervolume, en el cual se busca maximizar, las lineas superiores representan a las mejores

combinaciones de mutación y cruza.

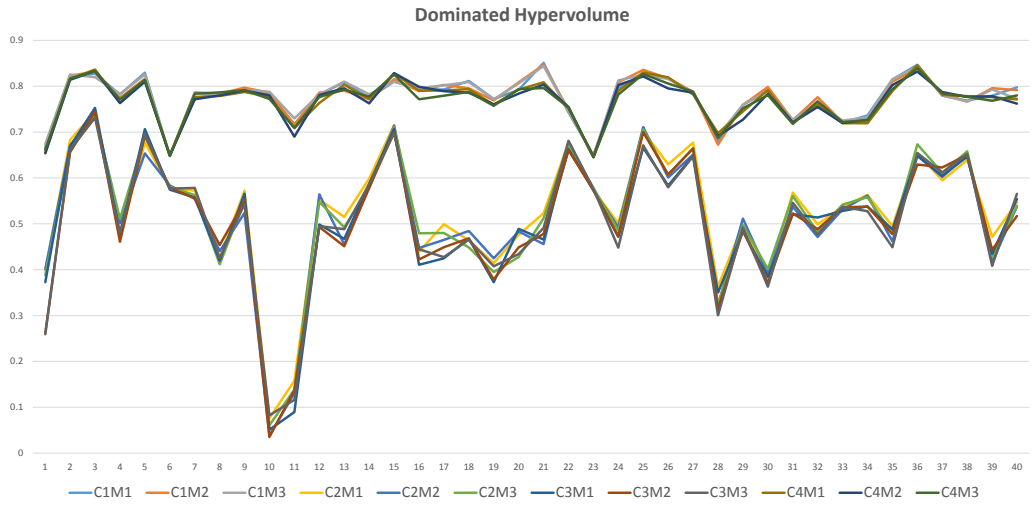

Figura 6.1: Resultados de hypervolume.

En la figura 6.2 se muestra la gráfica que representa los resultados obtenidos del indicador *Generational Distance*. En donde las lineas inferiores corresponden a las combinaciones de cruza y mutación C1M1, C1M2, C1M3, C4M1, C4M2 y C4M3. Las lineas superiores corresponden a las combinaciones restantes. En este indicador se busca minimizar, por lo tanto los mejores valores las contienen las combinaciones de cruzas y mutaciones que se encuentran en las lineas inferiores.

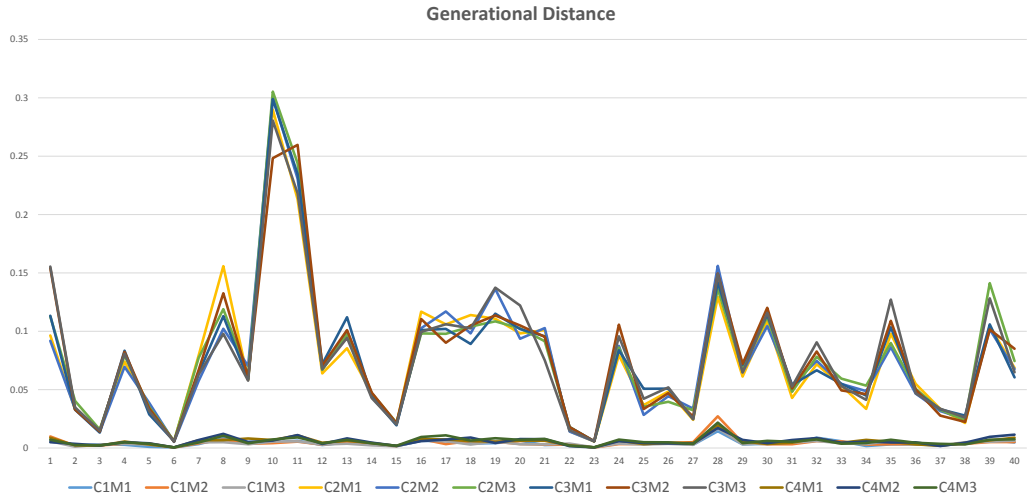

Figura 6.2: Resultados de Distancia generacional.

En la figura 6.3 se muestran los resultados de *Generalized Spread* de los cuales

sobresalen ligeramente la C1M1, C1M2, C1M3. Como se podrá apreciar mas adelante en la tabla 6.5. En este indicador se busca maximizar el valor del indicador de calidad.

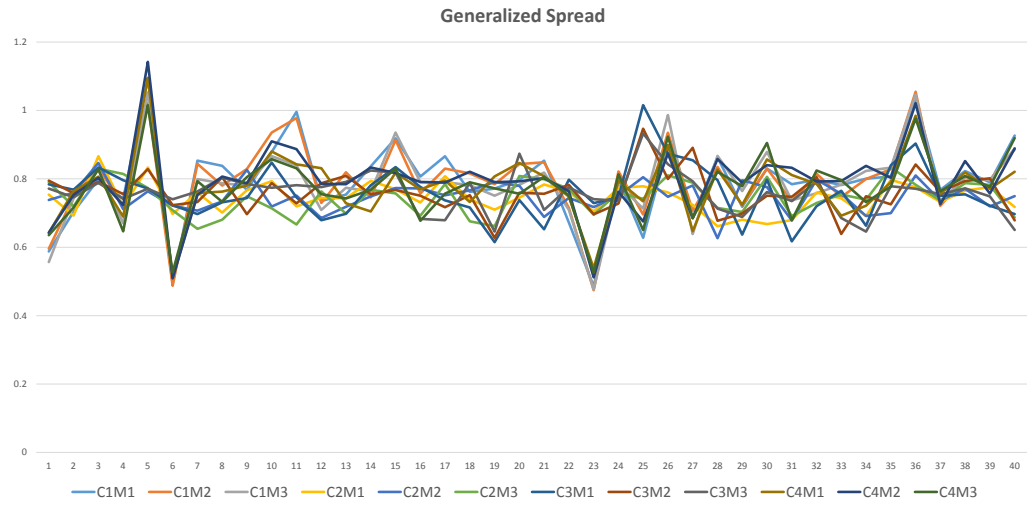

Figura 6.3: Resultados de Generalized Spread.

La siguiente tabla muestra el promedio de los resultados obtenidos para cada indicador de calidad. La primera, tercera y quinta columna de la tabla muestran el nombre que representa la combinación de la cruza y mutación. En la segunda columna se encuentran los resultados de *Dominated Hypervolume* (DH) que están ordenados de mayor a menor. En la cuarta columna los resultados de *Generational Distance* (GD) ordenados de menor a mayor. En la sexta columna los valores de *Generalized Spread* (GS) los cuales están ordenados de mayor a menor.

Tabla 6.5: Promedios de los resultados experimentales de cruzas y mutaciones.

Cruza-Mutación	DH	Cruza-Mutación	GD	Cruza-Mutación	GS
C1M3	0.77638996	C1M3	0.004160182	C3M3	0.794631196
C1M2	0.775394121	C1M1	0.004252087	C1M1	0.79460645
C1M1	0.774733877	C1M2	0.004576777	C1M3	0.793532997
C4M1	0.769458986	C4M1	0.005542124	C1M2	0.783733013
C4M3	0.768544196	C4M3	0.005807846	C2M3	0.77725219
C4M2	0.767315216	C4M2	0.005810714	C3M2	0.773652399
C2M1	0.528142194	C2M1	0.075285201	C2M2	0.759892876
C2M3	0.520751598	C2M2	0.075713006	C3M1	0.759231456
C2M2	0.516561711	C3M1	0.076518545	C2M1	0.758623316
C3M1	0.510699641	C2M3	0.077972721	C4M3	0.747649094
C3M3	0.50692644	C3M3	0.078445799	C4M2	0.745545496
C3M2	0.506755806	C3M2	0.078885777	C4M1	0.745476286

De la tabla anterior se puede concluir que la mejor combinación de cruza y mutación es la 1 con la 3 las cuales corresponden a la cruza uniforme y a la mutación de balanceo. Esta combinación de cruza y mutación se usara en la experimentación comparativa entre el NSGA–II y el MOEA/D

6.4.2. Experimentación de funciones de descomposición en MOEA/D

Esta experimentación consistió en probar nueve funciones de descomposición para ver cuál tiene un mejor desempeño al resolver el problema tratado. Se aplicaron dos indicadores de calidad el *Hypervolumen*(HV) y distancia generacional (GD).

La siguiente tabla muestra un resumen de los resultados obtenidos. La primera columna representa la función evaluada, la segunda columna el promedio del volumen que genero la función en el cálculo del HV, la tercera columna el promedio que genero la función en el cálculo de GD.

Tabla 6.6: Resultado experimentales de funciones de descomposición

Función	HV	GD
A	1.96E+16	6.44E-16
B	1.56E+16	1.13E-16
C	1.77E+16	8.13E-16
D	1.86E+16	7.82E-16
E	1.74E+16	6.81E-16
F	1.58E+16	1.08E-16
G	3.05E+15	1.71E-16
H	1.95E+16	6.50E-16
I	1.96E+16	6.41E-16

De la tabla anterior podemos apreciar que las funciones que tienen mejor desempeño son la I (*Augmented Tchebycheff Kaliszewski* en *Hypervolumen* y la G ($NASH$) en distancia generacional.

6.4.3. Experimentación comparativa entre NSGAII y MOEA/D

En esta experimentación cada algoritmo se ejecuto diez veces. El algoritmo NSGA–II utilizo la cruza uniforme y la mutación de balanceo de cargas. El algoritmo MOEA/D utilizó la función de descomposición de *Augmented Tchebycheff Kaliszewski*.

En la siguiente tabla se muestra una comparación de los promedios obtenidos del NSGAII y el MOEA/D. La primera columna es el nombre de la instancia, la segunda y tercer columna el promedio de hypervolumen obtenido para el NSGAII y MOEA/D respectivamente.

Tabla 6.7: Resultados experimentales de los indicadores de calidad con los algoritmos NSGAII y MOEA/D

Instancia	NSGAII	MOEA/D	Instancia	NSGAII	MOEA/D
Ahmad_3_9_28	1.86E+17	3.53E+25	Linshan_4_9_38	5.09E+25	4.14E+25
Bittencourt_3_9_184	1.91E+13	5.44E+26	Liu_2_8_364	2.72E+26	6.60E+26
Cao_3_10_536	2.04E+14	5.57E+27	Mohammad_2_11_64	2.32E+26	1.94E+25
Ching_3_10_84	2.21E+10	6.57E+25	Munir_3_10_76	1.18E+26	8.05E+25
Demiroz_3_7_47	1.39E+17	2.46E+25	Rahmani_3_7	6.34E+25	9.26E+28
Eswari_2_11_61	1.49E+13	1.37E+25	SahA_3_11_131	7.14E+24	1.05E+26
Gulzar_3_10_124	1.52E+14	2.15E+26	SahB_3_6_76	1.81E+26	3.62E+25
Hamid_3_10	1.65E+10	2.24E+26	Samantha_5_11_31	6.39E+25	1.54E+25
Hernandez_3_10_70	1.60E+17	5.09E+25	sample_3_10	7.67E+25	1.84E+26
heteroparjorgebarbosa1-3	1.75E+13	2.72E+26	sample_3_13	8.56E+25	1.98E+26
heteroparjorgebarbosa2	1.60E+14	2.32E+26	sample_3_8_100	9.60E+24	1.01E+26
Ijaz_3_10_133	2.16E+10	1.18E+26	sample_4_11_25	6.86E+25	1.18E+25
Ilavarasan_3_10_77	3.53E+25	6.34E+25	sample_8_3_100	4.14E+25	1.08E+26
Ilavarasan_3_11_27	5.44E+26	7.14E+24	Tao_3_10	6.60E+26	2.26E+26
Ilavarasan_3_15_114	5.57E+27	1.81E+26	TOPCUOGLU_3_10_80	1.94E+25	8.02E+25
IlavarasanIJCSIT_3_10	6.57E+25	6.39E+25	Topcuoglu_3_8_51	8.05E+25	2.64E+25
Kang_3_10_76	2.46E+25	7.67E+25	Xu_3_8_66	9.26E+28	4.09E+25
Kang_3_10_84	1.37E+25	8.56E+25	YCLee_3_8_80	1.05E+26	4.03E+25
Kuan_3_10_28	2.15E+26	9.60E+24	Yu_4_10	3.62E+25	1.39E+26
Liang_3_10_80	2.24E+26	6.86E+25	Zhao_3_10_143	1.54E+25	3.76E+26
Promedio				2.54E+27	2.58E+27

De la tabla anterior se puede apreciar que en general el MOEA/D genera un poco mas de volumen que el NSGAII para este conjunto de instancias. A continuación se muestra un resumen de los resultados obtenidos para las instancias de Braun. La primer columna contiene el nombre de los indicadores aplicados, la segunda columna contiene los promedios obtenidos por el NSGA-II y la tercer columna los promedios obtenidos por el MOEA/D.

Tabla 6.8: Resultados experimentales de los indicadores de calidad de los algoritmos NSGAII y MOEA/D para las instancias de Braunt

Indicador	NSGA-II	MOEA/D
HV	4.05E+14	6.85E+14
GD	0.008144914	11.71517311
GS	0.919328596	0.988951194

De la tabla anterior podemos observar que el MOEA/D generó más volumen y obtuvo una mejor distribución del frente, pero con menor diversidad. En cambio el NSGA-II obtuvo un menor volumen pero mayor diversidad y menor distribución del frente. Por lo tanto el MOEA/D dio mejores resultados para este conjunto de instancias en cuanto a calidad y distribución del frente.

Capítulo 7

Conclusiones y trabajos futuros

En esta sección se resumen las aportaciones de estre proyecto de tesis así como las conclusiones.

7.1. Conclusiones

Los resultados de este proyecto de investigación cumplieron con los objetivos que se plantearon inicialmente. Aún cuando en los objetivos solamente se consideraba la implementación de una metaheurística se abordo la implementaciíon de funciones de descomposición para el algoritmo MOEA/D y el estudio del framework de jMetal.

7.2. Productos científicos derivados de la tesis

- Un MILP de dos fases para obtener el *makespan* y la energía mínima.
- Un *Branch and Bound* de dos fases para obtener *makespan* y la energía mínima.
- Experimentación de cuatro cruzas y tres mutaciones para el NSGA-II.
- Experimentación comparativa entre NSGA-II y MOEA/D.
- Escritura de un artículo para ISCI de Tijuana.

7.3. Publicaciones

A continuación se presentan las publicaciones que se realizaron durante el proyecto.

- Monterrubio, J. C. S., Huacuja, H. J. F., and Valdez, G. C. (2013). *Modelo de programación lineal entera mixta de dos fases para el problema de asignación de tareas independientes con escalamiento dinámico de voltaje y frecuencia.* VII Encuentro de investigadores en el Instituto Tecnológico de Ciudad Madero (ITCM) [Monterrubio et al., 2013].
- Santiago, A., Huacuja, H. J. F., Dorronsoro, B., Pecero, J. E., Santillan, C. G., Barbosa, J. J. G., and Monterrubio, J. C. S. (2014). *A survey of decomposition methods for multi-objective optimization.* In Castillo, O., Melin, P., Pedrycz, W., and Kacprzyk, J., editors, Recent Advances on Hybrid Approaches for Designing

Intelligent Systems, volume 547. Springer International Publish [Pineda et al., 2014].

7.4. Trabajos futuros

Algunos de los trabajos futuros para continuar con esta investigación son las siguientes.

- Explorar nuevas formas de resolver el problema.
- Generar un benchmark de instancias para este problema.
- Obtener los óptimos de las instancias de Braunt.

Bibliografía

[Ali et al., 2000] Ali, S., Siegel, H. J., Maheswaran, M., Hensgen, D., and Ali, S. (2000). Task execution time modeling for heterogeneous computing systems. In Proceedings of the 9th Heterogeneous Computing, pages 170–185. IEEE Computer Society.

[Andrei et al., 2007] Andrei, A., Eles, P., Peng, Z., Schmitz, M., and Al-Hashimi, B. M. (2007). Designing embedded processors. Springer.

[Auger et al., 2009] Auger, A., Bader, J., Brockhoff, D., and Zitzler, E. (2009). Theory of the hypervolume indicator: Optimal distributions and the choice of the reference point. Foundations of Genetic Algorithms.

[Braunt et al., 2001] Braunt, T. D., Siegel, H. J., Beck, N., Boloni, L. L., and Maheswarans, M. (2001). A comparison study of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of parallel and distributed computing, 22:810–837.

[Bringmann and Friedrich, 2009] Bringmann, K. and Friedrich, T. (2009). Approximating the least hypervolume contributor: Np-hard in general, but fast in practice. Evolutionary Multi-criterion Optimization, 5467:6–20.

[CECAL, 1993] CECAL (1993). Hcsp - heterogeneous computing scheduling problem.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, 1917.

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. Trans. Evol. Comp, 6(2):182–197.

[Duarte et al., 2007] Duarte, A., Fernande, J. J. P., and Carrillo, M. G. (2007). Metaheurísticas. Ciencias Experimentales y Tecnolog¡a. Editorial Dykinson, S.L.

[Fogel et al., 1966] Fogel, L., Owens, A., and Walsh, M. (1966). Artificial Intelligence Through Simulated Evolution. John Wiley & Sons.

[Foster and Kesselman, 1998] Foster, I. and Kesselman, C. (1998). Computational Grids. Elsevier.

[Garey and Johnson, 1979] Garey, M. A. and Johnson, S. D. (1979). Computers and intractability: A guide to the theory of np-completeness. Freeman.

[Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res., 13(5):533–549.

[Graham et al., 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. Technical report, North-Holland Publishing Company.

[Hughes, 2003] Hughes, E. J. (2003). Multiple single objective pareto sampling. Proc. Congr. Evol. Comput., pages 2678–2684.

[Jin et al., 2001] Jin, Y., Okabe, T., and Sendhoff, B. (2001). Adapting weighted aggregation for multiobjective evolutionary strategies. Evolutionary Multicriterion Optimization, 1993:96–110.

[Kelly and Osman, 1996] Kelly, J. P. and Osman, I. H. (1996). Meta-Heuristic: Theory and Applications. Kluwer Academic Publisher.

[Lee and Zomaya, 2011] Lee, Y. C. and Zomaya, A. Y. (2011). Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst, 22:1374–1381.

[Liu and Zhu, 2010] Liu, Y. and Zhu, H. (2010). A survey of the research on power management techniques for high-performance systems. Software: Practice and Experience, pages 943–964.

[Mokotoff and Jimeno, 2002] Mokotoff, E. and Jimeno, J. L. (2002). Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. In Annals of Operations Research, pages 133–150. Kluwer Academic Publishers.

[Monterrubio et al., 2013] Monterrubio, J. C. S., Huacuja, H. J. F., and Valdez, G. C. (2013). Modelo de programación lineal entera mixta de dos fases para el problema de asignación de tareas independientes con escalamiento dinámico de voltaje y frecuencia. VII Encuentro de investigadores en el Instituto Tecnológico de Ciudad Madero (ITCM).

[Naujoks and Beume, 2005] Naujoks, B. and Beume, N. (2005). Multi-objective optimization using s-metric selection: Application to three-dimensional solution spaces. CEC, pages 1282–1289.

[Pecero et al., 2010] Pecero, J. E., Bouvry, P., and Barrios, C. J. (2010). Low energy and high performance scheduling on scalable computing systems. Latin-American Conference on High Performance Computing, pages 1–8.

[Pecero et al., 2011] Pecero, J. E., Bouvry, P., Huacuja, H. J. F., and Khan, S. U. (2011). A multi-objective grasp algorithm for joint optimization of energy consumption and schedule length of precedence-constrained applications. Cloud and Green Computing, pages 943–964.

[Pettey, 2007] Pettey, C. (2007). Gartner estimates ict industry accounts for 2 percent of global co2 emissions.

[Pineda et al., 2013] Pineda, A. A. S., Huacuja, H. J. F., and Sanchez, J. E. P. (2013). Estrategias de búsqueda local para el problema de programación de tareas en sistemas de procesamiento paralelo. Master's thesis, Instituto Tecnológico de Ciudad Medero, División de Estudios de Posgrado e Investigaciń.

[Pineda et al., 2014] Pineda, A. S., Huacuja, H. J. F., Dorronsoro, B., Pecero, J. E., Santillan, C. G., Barbosa, J. J. G., and Monterrubio, J. C. S. (2014). A survey of decomposition methods for multi-objective optimization. In Castillo, O., Melin, P., Pedrycz, W., and Kacprzyk, J., editors, Recent Advances on Hybrid Approaches for Designing Intelligent Systems, volume 547. Springer International Publish.

[Rechemberg, 1973] Rechemberg, I. (1973). Evolutions strategie: Optimizierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog.

[Sipser, 2006] Sipser, M. (2006). Introduction to the Theory of Computation. Course Technology PTR, 2nd edition.

[TOP500, 2013] TOP500 (2013). Top 500 list.

[Veldhuizen and Lamont, 2000] Veldhuizen, D. A. V. and Lamont, G. (2000). On measuring computation and convergence to a pareto front. 2000 Congress on Evolutionary Computation, pages 204–211.

[Yang and Ding, 2007] Yang, Q. and Ding, S. (2007). Novel algorithm to calculate hypervolume indicator of pareto approximation set. CoRR.

[Zanakis, 1981] Zanakis, S. (1981). A Large-Scale Integer Goal Programming Method with an Application to a Facility Location-Allocation Problem, pages 490–498. Springer-Verlag.

[Zhang and Li, 2007] Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on descomposition. IEEE Transactions on Evolutionary Computation, 11:712–731.

[Zhang and Li, 2009] Zhang, Q. and Li, H. (2009). Multiobjective optimization problems with complicated pareto sets, moea/d and nsgaii. IEEE Transactions on evolutionary computation, pages 229–242.

[Zhang et al., 2009] Zhang, Q., Liu, W., and Li, H. (2009). The performance of a new version of moea/d on cec09 unconstrained mop test instances. Evolutionary Computation, CEC'09. IEEE Congress, pages 203–208.

[Zhou et al., 2006] Zhou, A., Jin, Y., Zhang, Q., Sendhoff, B., and Tsang, E. (2006). Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. Proceedings of the Congress on Evolutionary Computation, pages 3234–3241.

[Zitzler et al., 2002] Zitzler, E., Laumanns, M., Thiele, L., Fonseca, M. C., and Fonseca, G. V. (2002). Why quality assessment of multiobjective optimizers is difficult. GECCO, pages 666–674.