Instituto Tecnológico de Ciudad Madero

División de Estudios de Posgrado e Investigación

Estrategias de Búsqueda Local para el problema de programación de tareas en sistemas de procesamiento paralelo

TESIS

PARA OBTENER EL TÍTULO DE:

Maestro en Ciencias en Ciencias de la Computación

PRESENTA:

Ing. Aurelio Alejandro Santiago Pineda

DIRECTOR DE TESIS:

Dr. Hector Joaquin Fraire Huacuja

CO-DIRECTOR DE TESIS:

Dr. Johnatan Eliabeth Pecero Sanchez

Enero 2013

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

SEP Institutos Tecnológicos

Subsecretaría de Educación Superior
Dirección General de Educación Superior Tecnológica
Instituto Tecnológico de Ciudad Madero

Cd. Madero, Tamps; a 08 de Febrero de 2013.

OFICIO No.: U5.070/13
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN
DE TESIS

C. ING. AURELIO ALEJANDRO SANTIAGO PINEDA
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 en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ESTRATEGIAS DE BÚSQUEDA LOCAL PARA EL PROBLEMA DE PROGRAMACIÓN DE TAREAS 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. MARIA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

ESTADOS UNIDOS MEXICANOS
S.E.P.
DIVISION DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
I T C M

c.c.p.- Archivo
Minuta

MYCHC 'NECO ' 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 9001 CERTIFIED COMPANY
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.

(*Ing. Aurelio Alejandro Santiago Pineda*)

In memoriam de Aurelio Pineda Macias

Índice general

Capítulo 1

Introducción

Actualmente las Tecnologías de la Información (TI) se han vuelto pieza clave del desarrollo de la humanidad, éstas son ampliamente utilizadas en instituciones gubernamentales, bancarias, transnacionales, científicas, de negocio electrónico, así como de esparcimiento social y de entretenimiento.

La alta demanda de las TI con mayor cantidad de procesamiento, mayor cantidad de usuarios y tareas computacionales cada vez más complejas, ha creado la necesidad de granjas de servidores o cluster (usualmente un conjunto de servidores interconectados por red).

En una infraestructura de TI como lo es un centro de datos (lugar que contiene los cluster) el consumo energético se incrementa dependiendo de la necesidad de procesamiento de datos, puesto que cada servidor en el centro de datos requiere un consumo energético.

Es aquí donde la GreenIT (Tecnologías de la información verdes) y las Ciencias Computacionales (estudio de las bases teóricas de la computación e información) se unen para encontrar un ahorro de consumo energético y optimizar el tiempo de ejecución de tareas computacionales.

El problema tratado en el presente documento es minimizar el tiempo de ejecución y el consumo de energía en sistemas de procesamiento paralelo heterogéneos (es decir cluster con servidores de diferentes características en hardware y/o software).

1.1. Descripción del problema

El sistema estudiado en este trabajo es un sistema computacional consistente de un conjunto M de procesadores/máquinas heterogéneas que están completamente interconectadas. Los procesadores tienen diferente capacidad de procesamiento, cada procesador $m_j \in M$ es $DVFS-enabled$ (Dynamic Voltage and Frecuency Scaling) es decir, que opera en un conjunto de voltajes suministrados V, a una velocidad relativa asociada a cada $v_i \in V$. Ejemplos de procesadores que utilizan esta tecnología se pueden encontrar en AMD e Intel.

Para simplificar sin ninguna perdida de generalidad se asume lo siguiente [Guzek et al., 2010] y [Lee and Zomaya, 2011]:

- Cuando un procesador esta en inactividad su voltaje más bajo es suministrado.
- Cada máquina tiene una conexión directa a otra máquina.
- La comunicación entre 2 máquinas no bloquea la comunicación de otras en la red.
- Los enlaces de comunicación no tienen conflictos.
- Todos los enlaces de comunicación funcionan a la misma velocidad.
- Una máquina puede enviar información a otra mientras ejecuta una tarea.
- Una máquina puede recibir información de otra mientras ejecuta una tarea.
- La comunicación entre tareas en la misma máquina es despreciada.

Los últimos puntos corresponden al modelo delay [Papadimitriou and Yannakakis, 1988]. Modelo el cual simplifica el problema como se menciona y sigue siendo utilizado en [Lee and Zomaya, 2011].

1.1.1. Modelo del programa paralelo

Un programa paralelo representado por un grafo dirigido acíclico $G = (T, E)$. Consiste de un conjunto T de vértices, y un conjunto E de aristas. Donde cada vértice t_i representa una tarea de un programa paralelo, cada arista $(t_i, t_j) \in E$ representan la(s) tarea(s) precedente(s) tal que t_j no puede iniciar hasta que la(s) tarea(s) t_i finalicen su ejecución.

Cada arista $(t_i, t_j) \in E$ tiene asociado un costo de comunicación entre tareas $C_{i,j}$, es decir que la salida de ti tiene que ser transmitida a t_j, para que t_j comience su ejecución, sin embargo, este costo de comunicación sólo es requerido cuando las tareas se ejecutan en diferente máquina.

Cada tarea t_i, tiene asociado un valor representando su costo computacional $P_{i,j}$ en una máquina m_j a una máxima velocidad y voltaje. El costo relativo en términos del tiempo de ejecución $P'_{i,j}$ es obtenido por la relación de transformación siguiente [Pecero et al., 2010] y [Pecero et al., 2011].

$$P'_{i,j} = \frac{P_{i,j}}{Velocidad} \tag{1.1}$$

donde P_{ij} corresponde a la tarea t_i y a la máquina m_j en la que se ejecuta, y $Velocidad$ esta asociada al nivel configurado en el procesador.

El *makespan* es el tiempo total de ejecución de tareas en un programa paralelo, dado por:

$$makespan = MAX(tTermina_i) for \quad i = 0, ..., n. \tag{1.2}$$

Donde $tTermina_i$ es el tiempo en que termina la tarea i.

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) definido en [Lee and Zomaya, 2011]. La disipación dinámica de energía P en este modelo esta 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 esta dado por:

$$Ec = \sum_{i=0}^{n} \alpha C V_i^2 f \cdot P^*_{i,j} = \sum_{i=0}^{n} K V_i^2 \cdot P^*_{i,j} \tag{1.4}$$

Donde V_i es el voltaje suministrado al procesador en el cual la tarea t_i es ejecutada, y $P^*_{i,j}$ es el costo computacional de la tarea t_i con la configuración de voltaje y máquina programada. Donde para el presente trabajo la constante K se desprecia usando para el calculo de la energía la siguiente sumatoria [Pecero et al., 2010].

$$Ec = \sum_{i=0}^{n} V_i^2 \cdot P^*_{i,j} \tag{1.5}$$

En este trabajo se considera la energía consumida durante el tiempo ocioso E_i, es decir cuando la máquina no esta ejecutando ninguna tarea su procesador es automáticamente escalado al voltaje más bajo lo cual esta definido por:

$$Ei = \sum_{m_j=0}^{n} (makespan - P'_{i,j} \quad \forall \quad ti \in m_j) V_n^2 \tag{1.6}$$

Donde $makespan$ es el costo computacional del programa paralelo, $P'_{i,j}$ es el costo computacional relativo de la tarea t_i en la máquina m_j y V_n es el voltaje mínimo en la máquina m_j.

Por último el costo total de consumo de energía esta dado por:

$$Et = Ec + Ei \tag{1.7}$$

1.1.3. Problema de calendarización de tareas

En este trabajo de tesis la calendarización de tareas esta definida como :
El proceso de asignar a un conjunto T de tareas un conjunto M de procesadores (sin violar condiciones de precedencia), con una selección discreta de voltajes. Con los objetivos de minimizar el *makespan* de la ecuación 1.2 y el consumo de energía de la ecuación 1.7. No se consideran tiempos límite para las tareas y éstas no pueden ser duplicadas.

1.1.4. Instancia del problema

Una instancia del problema está conformado por un grafo dirigido acíclico (ver Figura 1.1) $G = (T, E)$ de precedencia de tareas, los costos computacionales de dichas tareas a máximo voltaje P_{ij} en cada máquina m_j (ver Tabla 1.1), así como las configuraciones de procesador para cada máquina ver (Tabla 1.2).

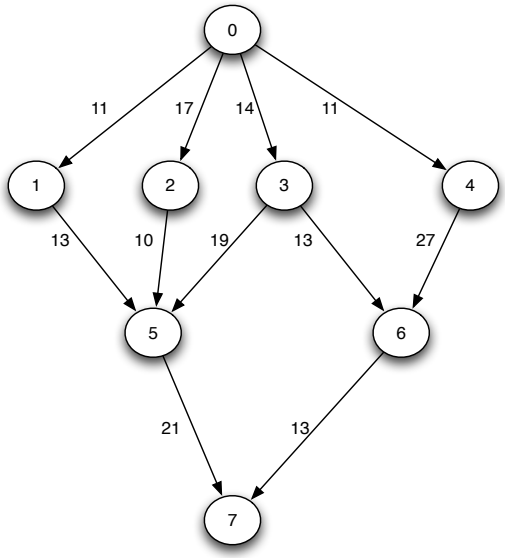


Figura 1.1: Grafo G de precedencia de tareas con costos de comunicación C_{ij}

Tabla 1.1: Costos Computacionales de las tareas a voltaje máximo P_{ij}

Tarea	m0	m1	m2
0	11	13	9
1	10	15	11
2	9	12	14
3	12	16	10
4	15	11	19
5	13	9	5
6	11	15	13
7	11	15	10

Tabla 1.2: Configuración de Procesadores

	Máquina 0		Máquina 1		Máquina 2	
Nivel	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad
0	1.75	1	1.50	1	2.20	1
1	1.40	.80	1.40	.90	1.90	.85
2	1.20	.60	1.30	.80	1.60	.65
3	0.90	.40	1.20	.70	1.30	.50
4			1.10	.60	1.00	.35
5			1.00	.50		
6			0.90	.40		

1.1.5. Solución candidata

Una solución candidata del problema esta formada por un orden de ejecución de tareas O sin violaciones de precedencia ver Tabla 1.3, y una configuración de parejas máquina/voltaje para cada tarea de el grafo como se muestra en el Tabla 1.4.

Tabla 1.3: Orden de ejecución de tareas sin violación de precedencia O

Tarea	0	4	3	1	2	5	6	7

Tabla 1.4: configuración máquina/voltaje

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2
voltaje	1	4	2	0	0	0	0	0

1.1.6. Cálculo de la Función Objetivo Makespan

Para el cálculo del *makespan* es necesario calcular los costos computacionales relativos de cada tarea $P'_{i,j}$ Tabla 1.5, éstos se calculan con la ecuación 1.1, y llevar un contador con el tiempo de finalización de la última tarea ejecutada en cada máquina $TAEm_j$ (tiempo actual de ejecución en máquina).

Tabla 1.5: Costo computacional con velocidad relativa $P'_{i,j}$

Tarea	0	1	2	3	4	5	6	7
Tiempo	13.75	31.42	15	12	15	5	11	10

El Algoritmo 1 muestra el cálculo de la función objetivo requiere el grafo G representando al programa paralelo, un orden de ejecución de tareas O sin violaciones de precedencia y los costos computacionales de las tareas $P'_{i,j}$. El algoritmo toma cada tarea del orden O y la asigna a la variable de tarea actual t_{actual}, cuando t_{actual} no tiene precedentes su tiempo de ejecución inicial $tEmpieza_{actual}$ es el valor del contador $TAEm_j$, en caso contrario $tEmpieza_{actual}$ tendrá el valor en el que la última tarea entre los precedentes termine la comunicación.

Algorithm 1 Función objetivo Makespan

Require: Grafo $G = (T, E)$, orden de ejecución de tareas $O = \{o_i, \ldots, o_n\}$, costos computacionales con velocidad relativa P'_{ij}, costos de comunicación entre tareas C_{ij}.
Ensure: O no viola precedencia de tareas.
  for $x = 0$ to n **do**
    $t_{actual} = o_x$
    if $((u, actual) \in E) = null$ **then**
      $tEmpieza_{actual} = TAEm_j$
      $tTermina_{actual} = tEmpieza_{actual} + P'ij_{actual}$
    else
      $u^* = argmax_{u \in V | (u,actual) \in E}\{tTermina_u + C'_{u \quad actual}\}$
      $tEmpieza_{actual} = \text{MAX}(tTermina_{u^*} + C'_{u^* \quad actual}, TAEm_j)$
      $tTermina_{actual} = tEmpieza_{actual} + P'ij_{actual}$
    end if
  end for
  makespan = MAX($tTermina_x$) for $x = 0, ..., n$

En la Tabla 1.6 se muestra un ejemplo de ejecución del Algoritmo 1 donde $TAEm_j$ es el tiempo en el que ejecuto su última tarea la máquina m_j, la columna empieza es el tiempo en el que la tarea $t_i$esta lista para ser ejecutada, la columna Termina es el tiempo en el que la tarea t_i termina su ejecución.

Tabla 1.6: Corrida de algoritmo 1

Configuración	Precedentes+Comunicación	$TAEm_j$	Empieza	Termina
t_0 en m_0 con v_1		0	0	13.75
t_4 en m_0 con v_0	$t_{0termina}$+ ~~11~~=13.75	13.75	13.75	28.75
t_3 en m_0 con v_0	$t_{0termina}$+ ~~14~~=13.75	28.75	28.75	40.75
t_1 en m_2 con v_4	$t_{0termina}$+11=24.75	0	24.75	56.17
t_2 en m_1 con v_2	$t_{0termina}$+17=30.75	0	30.75	45.75
t_5 en m_2 con v_0	$t_{1termina}$+ ~~13~~=56.17 $t_{2termina}$+10=55.75 $t_{3termina}$+19=59.75	56.17	59.75	64.75
t_6 en m_0 con v_0	$t_{3termina+}$ ~~13~~=40.75 $t_{4termina+}$ ~~27~~=28.75	40.75	40.75	51.75
t_7 en m_2 con v_0	$t_{5termina+}$ ~~21~~=64.75 $t_{6termina+}$13=64.75	64.75	64.75	74.75

1.1.7. Cálculo de la Función Objetivo Energía

Para calcular la energía requerida es necesario conocer el valor de *makespan*, costos computacionales relativos P'_{ij}, voltaje seleccionado v_k en cada tarea t_i con su procesador m_j asignado y v_n el voltaje más bajo para cada m_j.

El Algoritmo 2 muestra el cálculo de la función objetivo de energía el primera ciclo for corresponde a la Ecuación 1.5 de energía directa por tarea, mientras que el segundo corresponde a la Ecuación 1.6 consumo de energía en tiempo ocioso.

Algorithm 2 Función objetivo Energía

```
energia = 0
for all t_i do
    energia+ = v_k^2 · P'_{i,j}
end for
for all m_j do
    idle = makespan
    for all t_i ejecutadas en m_j do
        idle− = P'_{i,j}
    end for
    energia+ = idle · v_n^2
end for
```

La Tabla 1.7 es un ejemplo de ejecución del Algoritmo 2, la primera columna muestra la configuración de energía a calcular, y en la segunda columna la energía acumulada, primero la acumulación de la energía directa de la tarea t_0 hasta la tarea t_7, después la energía indirecta en tiempos muertos *idle* para cada máquina dándonos la energía total consumida.

Tabla 1.7: Corrida de algoritmo 2

	$energia = 0$
	$makespan = 74.75$
t_0 en m_0 con v_1	$energia+ = (1.40)^2 \cdot 13.75$
t_1 en m_2 con v_4	$energia+ = (1.00)^2 \cdot 31.42$
t_2 en m_1 con v_2	$energia+ = (1.30)^2 \cdot 15$
t_3 en m_0 con v_0	$energia+ = (1.75)^2 \cdot 12$
t_4 en m_0 con v_0	$energia+ = (1.75)^2 \cdot 15$
t_5 en m_2 con v_0	$energia+ = (2.20)^2 \cdot 5$
t_6 en m_0 con v_0	$energia+ = (1.75)^2 \cdot 11$
t_7 en m_2 con v_0	$energia+ = (2.20)^2 \cdot 10$
	$energia = 272.695$
m_0	$idle_{m0} = 23$
	$energia = 272.695 + 23 \cdot .90^2$
m_1	$idle_{m1} = 59.75$
	$energia = 291.325 + 59.75 \cdot .90^2$
m_2	$idle_{m2} = 28.33$
	$energia = 339.7225 + 28.33 \cdot 1^2$
	$energia = 368.0525$

1.2. Complejidad del problema

El problema de la asignación de tareas con y sin costos de comunicación con un número limitado de procesadores heterogéneos es NP-Hard; incluso un grafo extremadamente simple con procesadores homogeneos [Ullman, 1975] formado por cadenas (una cadena de un grafo G es una secuencia alternante de vértices y aristas $G_{i,j}$ empezando y terminando con vértices en los cuales cada aristas es incidente con 2 o más vértices inmediatamente precediendo y siguiéndola) con la unidad como tiempo de ejecución sin costos de comunicación no puede ser resuelto óptimamente en tiempo polinomial. Aun cuando el número de procesadores esta limitado a 2 el problema es NP-Hard [Sinnen, 2007].

El problema de la selección discreta de voltajes es un problema NP-Hard el cual se prueba por restricción al problema discrete time-cost trade-off (DTCT) conocido como NP-Hard. Entonces restringiendo al problema al discrete voltage selection problem without overheads (DNOH) con todas sus tareas requiriendo la unidad como tiempo de ejecución, se vuelve idéntico al problema DTCT entonces, DTCT $\in$ DNOH se llega ala conclusión que DNOH $\in$ NP [Andrei et al., 2007].

Ambos objetivos por separado son NP-Hard y la unión de ambos aumenta la dificultad del problema.

1.3. Objetivos General y Específicos

1.3.1. Objetivo General

Diseñar y evaluar estrategias de Búsqueda Local multi-objetivo para el problema de calendarización de tareas en sistemas de procesamiento paralelo.

1.3.2. Objetivos Específicos

- Analizar el problema.
- Analizar y rediseñar las Funciones Objetivo.
- Diseñar e implementar estrategias de Búsqueda Local.
- Evaluar experimentalmente las diferentes estrategias implementadas.

1.4. Justificación

La asignación de tareas con y sin costos de comunicación es NP-Hard con un número limitado de procesadores heterogéneos [Sinnen, 2007]. La selección discreta de voltajes es un problema NP-Hard.[Andrei et al., 2007]. Esto implica que no se conoce un algoritmo que los resuelva en tiempo polinomial, rázon por la cual se justifica el uso de métodos heurísticos para resolverlo.

Sin embargo estos dos objetivos se encuentran en conflicto, minimizar el *makespan* produce un aumento en el consumo de energía, minimizar el consumo de energía utilizando DVFS produce un aumento del *makespan*. Lo cual lo convierte en un problema multiobjetivo [Deb, 2001].

La firma de análisis de TI IDC estima que en todo el mundo las empresas gastaron en administración de energía cerca de 40 billones de dólares en el 2009. El equipo computacional en los Estados Unidos se estima que consume más de 20 millones de Gigajoules de energía por año, el equivalente a 4 millones de toneladas de emisiones de dióxido de carbono hacia la atmósfera [Ranganathan, 2010].

1.5. Alcances y Limitaciones

- El presente trabajo de investigación tiene como propósito la implementación y evaluación de mecanismos de búsqueda local para minimizar el tiempo de ejecución y el consumo de energía en sistemas de procesamiento paralelo.

- Los conjuntos de instancias empleadas en el estudio experimental son una abstracción de aplicaciones paralelas reales : FPPP, LIGO, ROBOT, SPARSE [Mateusz et al., 2010].

- No se consideran la duplicación de tareas, ni el uso de algoritmos con programación paralela.

Capítulo 2

Marco Teórico

En este capítulo de tesis se abordan los conceptos necesarios para resolver problemas de optimización mono objetivo y multi-objetivo, así como el diseño de búsquedas locales y metaheurísticas.

2.1. Complejidad Computacional

Problema de decisión

Los problemas de optimización tienen su versión en problema de decisión en el que la respuesta es si o no [Sipser, 2006].

Clase P

Es el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial por un algoritmo determinista [Sipser, 2006].

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 [Sipser, 2006].

Problema NP-Completo

Un problema B es NP-completo si satisface 2 condiciones:
Si B pertenece ala clase NP y cada problema A en NP es reducible en tiempo polinomial a B [Sipser, 2006].

Problema NP-Hard

Un problema NP-Hard es un problema de optimización cuya versión de decisión es NP-Completo [Garey et al., 1979].

Problema de Optimización

En los problemas llamados problema de optimización se busca la mejor solución posible de un conjunto de posibles soluciones. Por ejemplo, nosotros podemos buscar encontrar el clique [Garey et al., 1979] más grande de un grafo, el recubrimiento de vértices [Garey et al., 1979] más pequeño, o el camino más corto entre dos nodos [Sipser, 2006].

2.2. Métodos de Optmización

Métodos Exactos

Los métodos exactos de resolución de problemas como: enumerativo, algoritmos de ramificación y poda, backtracking, etc; Resuelven problemas que pertenecen ala clase P de forma óptima y en tiempo razonable. Pero no pueden resolver problemas de la clase NP en tiempo polinomial; es decir, aunque existe un algoritmo que encuentra la solución exacta tardaría tanto tiempo en encontrarla que lo hace completamente inaplicable [Abraham Duarte Muñoz, 2010].

Métodos Heurísticos

Las heurísticas utilizan reglas, estrategias o programas basados generalmente en conocimientos empíricos para guiar el conocimiento. La palabra heuriskein procede del griego y significa «descubrir, encontrar, inventar» (etimología que comparte con eureka) [Polya, 1971]. Nicholson y Reeves ofrecen las siguientes definiciones de heurística.

- Un procedimiento para resolver problemas por medio de un método intuitivo en el que la estructura del problema puede interpretarse y explotarse inteligentemente para obtener una solución razonable [Nicholson, 2007]
- Una técnica que busca buenas soluciones con un tiempo de computación razonable sin garantizar la optimalidad [Reeves, 1993]

Métodos Metaheurísticos

Las siguientes definiciones aportadas por diferentes investigadores describen sus principales propiedades.

- Los procedimientos Metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son eficientes. [Duarte et al., 2006]
- Una metaheurística es un proceso iterativo que combina conceptos derivados de las ciencias naturales, la biología, la matemáticas, la física y la inteligencia artificial, para guiar de forma inteligente una heurística subordinada durante la exploración y explotación del espacio de búsqueda con el fin de encontrar soluciones cercanas al óptimo de forma eficiente. [Kelly and Osman, 1996]

2.3. Optimización Multiobjetivo

Cuando un problema de optimización involucra una sola función objetivo la tarea de encontrar una solución óptima es llamada optimización de un solo objetivo. Cuando un problema de optimización involucra más de una función objetivo la tarea de encontrar una o más soluciones óptimas se conoce como optimización multiobjetivo.

En el lenguaje de la administración son conocidos como multi-criterio (multiple criterion decision-making MCDM). Diferentes soluciones pueden producir escenarios conflictivos atraves de diferentes objetivos. Para dos objetivos en conflicto cada objetivo corresponde a una diferente solución óptima [Deb, 2001].

Soluciones Óptimas de Pareto

El teorema del contacto definido en [Coello, 1996], es uno de los principales teoremas en la optimización multiobjetivo, nos asegura que nuestra solución sea pareto óptima. Defina el cono negativo como:

$$C^{-} = \{f \in R^{k} | f <= 0\} \tag{2.1}$$

Entonces el teorema de contacto nos dice que: un vector f^{*} es una solución óptima de pareto para un problema multiobjetivo en general si y solo si

$$(C^{-} + f^{*}) \cap F = f^{*} \tag{2.2}$$

Gráficamente se puede ver de la siguiente forma tomada de [Osyczka, 1984]:

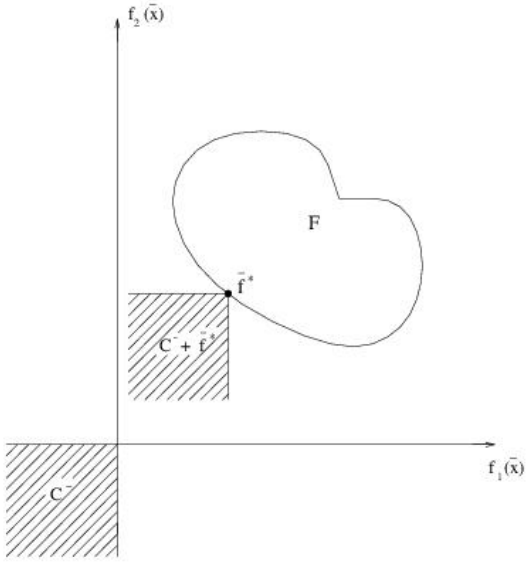


Figura 2.1: Ilustración gráfica del teorema de contacto

Óptimo de Pareto

Decimos que un punto $\vec{x}^* \in \Omega$ es un óptimo de Pareto si para todo $\vec{x}^{'} \in \Omega eI = \{1, 2, 3, ..., k\}$ si $\forall i \in I \ f_i(\vec{x}^{'}) = f_i(\vec{x}^*)$, o existe almenos una $i \in I$ tal que $f_i(\vec{x}^{'}) > f_i(\vec{x}^*)$

Dominancia de Pareto

Un vector $\vec{u} = (u_1, ..., u_k)$ domina a otro $\vec{v} = (v_1, ..., v_k)$ denotado mediante $(\vec{u} \prec \vec{v})$ si y solo si u es parcialmente menor que v $\forall i \in \{1, ..., k\}$, $u_i \leq v_i \wedge \exists i \in \{1, ..., k\} : u_i < v_i$.

Conjunto de Óptimos de Pareto

Para un problema multiobjetivo dado $\vec{f}(x)$, el conjunto de óptimos de pareto P^* se define como : $P^* := \{x \in \Omega | \neg \exists x^{'} \in \Omega \vec{f}(x^{'}) \leq \vec{f}(x)\}$

Frente de Pareto

Para un problema multiobjetivo dado $\vec{f}(x)$ y un conjunto de óptimos de pareto P^*, el frente de Pareto (PF^*) se define como :
$PF^* := \{\vec{u} = \vec{f} = (f_1(x), ..., f_k(x)) | x \in P^*\}$

Conjunto de soluciones no dominadas (Random Non-Dominated point set)

Un conjunto U es un conjunto de soluciones no dominadas si y solo si $\forall \vec{u} \in U$ no existe $\vec{u}' \in U | \vec{u}' \prec \vec{u}$

2.4. Búsquedas Locales

2.4.1. Vecindario

En optimización dada una solución $x \in \Omega$ el vecindario $N(x)$ de x es un conjunto de soluciones en el espacio de soluciones Ω que se encuentran "próximas"[Abraham Duarte Muñoz, 2010]

a un "movimiento"(aplicación del operador de vecindario).

2.4.2. Operador de vecindario

Un operador de vecindario transforma una solución inicial x y permite alcanzar todas sus soluciones en $N(x)$ mediante movimientos (aplicaciónes del operador) usualmente swaps o inserciones.

2.4.3. Óptimo Local

Un óptimo local es una solución $x|\neg\exists x' \in N(x)|f(x) > f(x')$ [Abraham Duarte Muñoz, 2010]; En optimización multiobjetivo la noción de óptimo local tiene que ser definida en términos de óptimo de pareto. Una solución $\vec{u} \in \Omega$ se dice que es un óptimo local o localmente eficiente con respecto a un vecindario $N(\vec{u})$ si $\neg\exists\vec{u}' \in N(\vec{u})$ tal que $\vec{u}' \prec \vec{u}$ [Liefooghe et al., 2011].

2.4.4. Búsqueda Local

Una Búsqueda Local es un método de mejora para una solución x, el cual aplicando un operador de vecindario actualiza x con mejoras en $N(x)$ iterativamente, se detiene al no encontrar mejoras en $N(x)$.

2.5. Algoritmos Evolutivos

Los Algoritmos Evolutivos [Abraham Duarte Muñoz, 2010] son metaheurísticas 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).

Algoritmo Genético

Los algoritmos genéticos son algoritmos evolutivos, fueron introducidos por J. Holland utilizan los operadores genéticos de selección, cruza y mutación [Abraham Duarte Muñoz, 2010].

- Selección: mecanismo probabilista que favorece a los individuos más aptos para tener descendencia.
- Cruza: intercambio de secciones entre 2 individuos para formar uno nuevo.
- Mutación: cambio aleatorio de una sección del individuo.

Non-Dominated Sorting Genetic Algorithm II

El NSGA-II es un algoritmo de optimización multi-objetivo 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 jamas desaparecen de la población.
- En la selección de encontrarse 2 soluciones no dominadas entre si se prefiere la más diversa.

Capítulo 3

Estado del Arte

3.1. Algoritmos de clustering

Los algoritmos de clustering buscan agrupar tareas que están fuertemente relacionadas entre si para minimizar tiempos de ejecución.

[Kumar and Manimaran, 2005]Common Hot Path (CHP) la idea básica es agrupar todos los hot-path (secuencias de tareas frecuentemente utilizadas) y combinarlas en un path común, el cual representa un hot-path virtual, cuya duración debe ser semejante ala mayoría de los hot-path, cada bloque es asignado a un procesador a una frecuencia determinada.

La Tabla 3.1 resume las características del algoritmo de cluster analizado, utiliza la tecnica $DVFS$ pero no toma en cuenta la energía en idle.

Tabla 3.1: Algoritmos de clustering

Autor	Algoritmo	DVFS	Energía	Energía-idle	Multiobjetivo
[Kumar and Manimaran, 2005]	CHP	SI	SI	NO	NO

3.2. Algoritmos de lista scheduling

Los algoritmos de lista scheduling se basan en heurísticas como el b-level [Pecero et al., 2010] entre otras dando un ordenamiento topológico, a partir del cual trabajan en asignar las tareas a los procesadores.

[Haluk Topcuoglu, 2002]Heterogeneous Earliest Finish Time (HEFT) dado un ranking de prioridad de tareas prueba acomodar la tarea en cada uno de los procesadores y esta es asignada al procesador que minimize su EFT (Earliest Finish Time) que en algunos casos es igual al AFT (Actual Finish Time) dependiendo de la implementación del algoritmo.

[Haluk Topcuoglu, 2002] Critical Path On a Processor (CPOP) en la primera fase el ranking se realiza recorriendo el grafo de arriba hacia abajo y se penaliza su prioridad recorriendo el grafo de abajo hacia arriba, en la segunda fase la tarea con mayor prioridad es seleccionada para su ejecución si la tarea se encuentra dentro del Critical Path (Camino más largo) su Critical Processor es aquel que minimize los costos acumulados en el critical path, de otro modo se le asigna a un procesador que minimize si EFT.

[Baskiyar and Palli, 2006] Low power HEFT (LPHEFT) es una versión del HEFT que nos da el mismo makespan y consume considerablemente menos energía que el HEFT, en este algoritmo la frecuencia del procesador es escalada durante el tiempo *idle*. El tiempo *idle* es eliminado decrementando la velocidad del procesador incrementando el tiempo que toma ejecutar una tarea.

[Wang et al., 2010]LPHM la idea es básicamente parecida a LPHEFT solo que si la tarea se encuentra en el camino critico su tiempo no es reducido, si el usuario tolera un aumento en el tiempo ejecución se reducen los voltajes de todas las tareas siempre y cuando estén en tiempo *idle* o de comunicación.

[Lee and Zomaya, 2009a]Energy Concious Scheduling (ECS) en este trabajo se propone una función objetivo llamada Relative Superiority (RS) la cual es una relación entre el *makespan* y la *energia*. y se computan las combinaciones de procesador/voltaje para cada tarea y el que minimize es seleccionado.

[Lee and Zomaya, 2009b] ECS+idle en este trabajo se retoma la función objetivo RS en [Lee and Zomaya, 2009a] la cual fue modificada para tomar en cuenta el *idle*. La Tabla 3.2 resume las características de los algoritmos de ordenación de lista scheduling analizados, el unico que tiene todas las caracteristicas tratadas en este trabajo es el ECS+IDLE [Lee and Zomaya, 2009b].

Tabla 3.2: list scheduling algorithms

Autor	Algoritmo	DVFS	Energía	Energía-idle	Multiobjetivo
[Haluk Topcuoglu, 2002]	HEFT	NO	NO	SI	NO
[Baskiyar and Palli, 2006]	LPHEFT	SI	SI	SI	NO
[Wang et al., 2010]	LPHM	SI	SI	SI	NO
[Haluk Topcuoglu, 2002]	CPOP	NO	NO	NO	NO
[Lee and Zomaya, 2009a]	ECS	SI	SI	NO	SI
[Lee and Zomaya, 2009b]	ECS+IDLE	SI	SI	SI	SI

3.3. Algoritmos de ordenación de nivel

[Ilavarasan and Thambidurai, 2007]Levelized min time (LMT) en la primera fase agrupa las tareas que pueden ser ejecutadas en paralelo, nivel por nivel, la segunda fase es voraz y asigna la tarea al procesador más rápido disponible, si el número de tareas en un nivel es mayor que el número de procesadores disponibles las tareas se fusionan en tareas virtuales que son ejecutadas por un procesador.

[Ilavarasan and Thambidurai, 2007]Performance Effective Task Scheduling (PETS) Este algoritmo consta de tres fases, ordenamiento por nivel, priorización de tareas, selección de procesador. Se recorre el grafo de arriba hacia abajo para ordenar en cada nivel las tareas que son independientes una de otra. Se computa la prioridad de las tareas con el Average Computation Cost (ACC),Data Transfer Cost(DTC) y Rank of Predecesor (RPT). Se selecciona el procesador que minimiza el EFT.

La Tabla 3.3 resume las características de los algoritmos de ordenación de nivel analizados, ninguno optimiza el objetivo de energía y *makespan* al mismo tiempo.

Tabla 3.3: level sorting algorithms

Autor	Algoritmo	DVFS	Energía	Energía-idle	Multiobjetivo
[Ilavarasan and Thambidurai, 2007]	PETS	NO	NO	SI	NO
[Ilavarasan and Thambidurai, 2007]	LMT	NO	NO	NO	NO

3.4. Búsquedas Locales

[Pecero et al., 2010] En este trabajo se utiliza el b-level para el ranking de prioridad de tareas, se crea posteriormente la asignación de tareas a procesador usando la heurística HEFT con política de inserción de tareas en tiempo *idle*, y se realiza una Búsqueda Local Aleatoria para minimizar la energía sin aumentar el *makespan*.

[Kim et al., 2007]Push Pull Emplean una búsqueda determinista, que se le aplica a una solución previamente creada por un algoritmo de scheduling como HEFT o HCPT, la idea básica en la parte Push es empujar las tareas hacia otro procesador y ver si la solución mejora, de ser así se toma. En la parte Pull la idea es jalar las tareas hacia un mismo procesador y si mejora la solución es tomada.

[Kang et al., 2011]Iterated local search se parte de una solución inicial cualquiera factible se le aplica Búsqueda Local, en la siguiente fase de destrucción algunos elementos son removidos aleatoriamente de la solución para obtener una solución parcial, después se vuelve a construir

una solución candidata completa usando una función greedy, y se le vuelve aplicar Búsqueda Local iterativamente, este proceso se repite.

[Wu et al., 2001]Efficient Local Search for DAG, Búsqueda Local basada en ordenamiento topológico la idea básica es minimizar la suma de b-level y t-level de cada tarea y propagar las modificaciones a las tareas hijo, después de minimizada la suma de b-level+t-level de cada tarea se evalúa si se encontró una mejor solución.

La Tabla 3.4 resume las características de las búsquedas locales analizadas, la unica que maneja el objetivo de energía es [Pecero et al., 2010].

Tabla 3.4: Búsquedas Locales

Autor	Algoritmo	DVFS	Energía	Energía-idle	Multiobjetivo
[Wu et al., 2001]	Local Search for DAG	NO	NO	NO	NO
[Kang et al., 2011]	iterated greedy algorithm	NO	NO	NO	NO
[Kim et al., 2007]	Push Pull	NO	NO	NO	NO
[Pecero et al., 2010]	Random Local Search	SI	SI	NO	SI

3.5. Metaheurísticas

[Mezmaz et al., 2010] Usando un algoritmo genético bi-objetivo híbrido el cual mejora las soluciones con la primera fase de la heurística ECS [Lee and Zomaya, 2009a] usando el enfoque de el frente de soluciones Pareto. Este enfoque es un híbrido entre AG Multiobjetivo y ECS. La heurística ECS es llamada cuando una solución es modificada por operadores genéticos (mutación y cruza). ECS corrige los errores de precedencia de tareas.

[Pecero et al., 2011] GRASP el algoritmo consiste en 2 fases la primera construye una solución factible usando una función greedy a partir de una lista de candidatos restrictiva, en la segunda fase el algoritmo usa Búsqueda Local para mejorar la calidad de la solución de la primera fase. las soluciones son generadas a máximo voltaje y después escaladas para reducir el consumo de energía.

[Guzek, 2010] Multi-objective Evolutionary Algorithm, provee un algoritmo que produce un conjunto de soluciones Pareto-Optimal,implementado en JMetal.

La Tabla 3.5 resume las características de las metaheurísticas analizadas, solo el GRASP utiliza energía en tiempo oscioso[Pecero et al., 2011].

Tabla 3.5: Metaheurísticas

Autor	Algoritmo	DVFS	Energía	Energía-idle	Multiobjetivo
[Mezmaz et al., 2010]	Hybrid bi-objective	SI	SI	NO	SI
[Pecero et al., 2011]	GRASP	SI	SI	SI	SI
[Guzek, 2010]Mateuz	MOEA	SI	SI	NO	SI

Capítulo 4

Heurísticas constructivas y ordenamientos topológicos

En este capítulo se describen los algoritmos utilizados para generar un orden de ejecución de tareas sin violaciones de precedencia, conocidos en la literatura como algoritmos de ordenamiento topológico; Así como algoritmos de construcción inicial de soluciones que son utilizados en este trabajo de tesis.

4.1. Algoritmos de ordenamiento topológico

Un ordenamiento topológico es necesario para no violar la precedencia de tareas, en la Figura 1.1 se muestra el grafo para la instancia sample8_3.txt y a continuación algunos de los resultados de los algoritmos implementados.

4.1.1. B-Level

El ordenamiento por b-level [Pecero et al., 2010] genera un orden de ejecución de tareas factible, el b-level de una tarea t_i es el camino más largo hasta la última tarea en su camino tomando en cuenta el promedio de los costos de ejecución (ver $\overline{pi}$ en Tabla 4.1), como el tiempo que toma cada tarea en ser realizada y los costos de comunicación en el grafo, por ejemplo para la instancia sample8_3.txt su b-level por tarea es el siguiente:

Tabla 4.1: Costos computaciones instancia sample8_3.txt con promedio

Tarea	m0	m1	m2	$\overline{pi}$
0	11	13	9	11.00
1	10	15	11	12.00
2	9	12	14	11.66
3	11	16	10	12.33
4	15	11	19	15.00
5	12	9	5	8.66
6	10	14	13	12.33
7	11	15	10	12.00

El Algoritmo 3 calcula el b-level de cada tarea recursivamente. Recibe como parámetro de entrada la tarea a calcular el b-level t_{actual} y el grafo G representando al programa paralelo; La Linea 1 inicializa el acumulador *sum* en el promedio de ejecución de la tarea actual $\overline{pi}[t_{actual}]$, después la Linea 2 hace la primer llamada al método recursivo *blevel* el cual recibe la t_{actual} y *sum* dentro del método *blevel* asignamos a la variable *max* el valor de *sum* en la Linea 3. La Linea 4 verifica si t_{actual} tiene descendentes en caso de tenerlos *max* toma el valor máximo producido por el método recursivo blevel que recibe como parámetros la tarea descendente u

y la suma de los valores del acumulador sum, el promedio de ejecución de $\overline{pi}[u]$, y el costo de comunicación (t_{actual}, u). Por último el algoritmo regresa el valor de max.

Algorithm 3 Recursivo B-Level

Require: t_{actual}, sum, $G = (T, E)$
1: $sum = \overline{pi}[t_{actual}]$
2: $blevel(t_{actual}, sum)$
3: $max = sum$
4: $max = \text{argmax } \forall(t_{actual}, u) \in E\{blevel(u, (sum + (t_{actual}, u) + \overline{pi}[u]))\}$
5: **return** max

Tabla 4.2: Ejemplo de calculo de b-level para las tarea 7 y 6

Tarea	Suma	Resultado
7	12	12
6	12.33 + 13 + 12	37.33

Tabla 4.3: El b-level de las tareas en la instancia sample8_3.txt

Tarea	b-level
0	101.33
1	66.66
2	63.33
3	73,00
4	79.33
5	41.66
6	37.33
7	12.00

Ya que se tienen calculados los b-level de cada tarea se ordenan de mayor a menor produciendo el orden de ejecución.

Tabla 4.4: Orden de ejecución generado por b-level para la instancia sample8_3.txt

Orden	0	4	3	1	2	5	6	7

4.1.2. Orden Aleatorio Factible

El ordenamiento aleatorio implementado genera un orden de ejecución de tareas factible, con una lista de tareas ejecutadas se revisan las tareas que todos sus precedentes ya han sido ejecutados y no han sido ejecutadas, dichas tareas están listas para ser ejecutadas, de entre las cuales se escoge aleatoriamente la siguiente en el orden.

El Algoritmo 4 genera un orden de ejecución de tareas factible de manera estocástica, requiere como parámetros de entrada un arreglo de tareas ejecutadas R y el grafo G representando al programa paralelo. Inicialmente el arreglo de tareas ejecutadas se inicializan en 0 representando su no ejecución de la Linea 1 a la 3, después la Linea 4 inicializa el contador *ordenadas* en 0 representando que ninguna tarea ha sido ordenada. De la Linea 5 a 17 el ciclo principal es ejecutado, la Linea 6 prepara una lista vacía L de las tareas listas para su ejecución, de la Linea 7 a la 12 se buscan las tareas que no han sido ejecutadas y que todos sus precedentes han sido ejecutados para agregarlas ala Lista L. Al finalizar el proceso de búsqueda la Linea 13 agrega una tarea aleatoria de las pertenecientes a L y en la Linea 14 se aumenta el contador *ordenadas*; El proceso principal se repite hasta que el contador *ordenadas* alcanza la cardinalidad del conjunto de tareas T.

Algorithm 4 Orden aleatorio

Require: Arreglo de tareas ejecutadas $R = \{r_i, \ldots, r_{|T|-1}\}$
Require: $G = (T, E)$
for $x = 0$ to $|T| - 1$ **do**
  $r_x = 0$
end for
$ordenadas = 0$
while $ordenadas < |T|$ **do**
  vacía la lista L
  for $x = 0$ to n **do**
    $t_{actual} = x$
    if $R[t_{actual}] = 0 \wedge \forall$ (u,actual) $\in$ E $\{R[t_u] = 1\}$ **then**
      Agrega a L la tarea t_{actual}
    end if
  end for
  Agrega a orden una tarea aleatoria de L y marcar en R como ejecutada
  $ordenadas = ordenadas + 1$
end while
return orden

Tabla 4.5: Ejemplo de arreglo de tareas ejecutadas R

Tarea	0	1	2	3	4	5	6	7
Ejecutada	SI	SI	SI	SI	SI	NO	NO	NO
Lista	NO	NO	NO	NO	NO	SI	SI	NO

Tabla 4.6: Orden de ejecución generado aleatoriamente para la instancia sample8_3.txt

Orden	0	4	2	3	1	5	6	7

4.2. Algoritmo Constructivos

Un algoritmo constructivo construye una solución inicial de cierta calidad utilizando heurísticas, evitando empezar con soluciones aleatorias que pueden estar muy retiradas de un óptimo local o global.

4.2.1. HEFT con evaluación de Actual Finish Time

Como primer constructivo se implementó el algoritmo HEFT [Haluk Topcuoglu, 2002] con evaluación de Actual Finish Time (tiempo actual de terminación), sin la política basada en inserción. Los resultados para el grupo de instancias que se conoce el óptimo global de *makespan* utilizando b-level y ordenes aleatorios se muestran en la Tabla 4.7 y 4.8.

El Algoritmo 5 construye una solución inicial apartir de un orden de ejecución de tareas O para cada tarea $t_i \in O$ se asigna el procesador $m_j^* \in M$ que minimiza el AFT de la tarea t_i.

Algorithm 5 HEFT con evaluación de AFT

Require: Orden de ejecución de tareas sin violaciones de precedencia $O = \{o_i, \ldots, o_{|T|-1}\}$
for $x = 0$ to $|T| - 1$ **do**
  $t_{actual} = o_x$
  $m_j^* = argmin\{AFT(t_{actual}, j)\} \forall m_j \in M$
end for

El algoritmo 6 calcula el AFT de una tarea especifica t^*, es una versión recortada de la función objetivo de *makespan* que retorna el tiempo en que termina la tarea t^*, la diferencia con la función objetivo es que esta regresa el tiempo de terminación de la tarea t^*.

Algorithm 6 Actual Finish Time

Require: Orden de ejecución de tareas sin violaciones de precedencia $O = \{o_i, \ldots, o_{|T|-1}\}$
Require: indice i en orden de tarea t^* a evaluar su AFT
Require: Asignación parcial de máquinas hasta la tarea t^*

for $x = 0$ to i **do**
  $t_{actual} = o_x$
  if $((u, actual) \in E) = null$ **then**
    $tEmpieza_{actual} = TAEm$
    $tTermina_{actual} = tEmpieza_{actual} + P'_{ij_{actual}}$
  else
    $u^* = argmax_{u \in V|(u,actual) \in E}\{tTermina_u + C'_{u,actual}\}$
    $tEmpieza_{actual} = \text{MAX}(tTermina_{u^*} + C'_{u^* \quad actual}, TAEm)$
    $tTermina_{actual} = tEmpieza_{actual} + P'_{ij_{actual}}$
  end if
end for
return $t^*_{termina}$

Las Tablas a continuación se muestran los resultados de utilizar el constructor HEFT con el ordenamiento blevel Tabla 4.7 y utilizarlo con 10 ordenamientos generados aleatoriamente Tabla4.8, la mejora es notable explorando diferentes ordenes por que se explora un mayor espacio de soluciones.

Tabla 4.7: Resultados utilizando orden de blevel

Instancia	*Makespan* óptimo	Blevel+HEFT	Error
sample10_2.txt	56	76.5	36 %
sample_8_3.txt	66	88	33 %
sample10_3GA.txt	143	153	6.9 %
sample10_3.txt	143	156	9 %
sample10_0.dag	73	76	4 %
sample11_3.txt	27	29	7.4 %
sample13_3102.txt	112	127	13.39 %

Tabla 4.8: Resultados utilizando 10 orden de aleatorio

Instancia	*Makespan* óptimo	Aleatorio+HEFT	Error
sample10_2.txt	56	63	12.5 %
sample_8_3.txt	66	80	21.2 %
sample10_3GA.txt	143	143	0 %
sample10_3.txt	143	147	2.7 %
sample10_0.dag	73	76	4.1 %
sample11_3.txt	27	27	0 %
sample13_3102.txt	112	112	0 %

4.2.2. Menor costo por tarea + CCR

Este constructivo selecciona el procesador que le cuesta menos tiempo ejecutar cada tarea, después se localizan las tareas sin predecesores y sus decendientes con un alto valor de CCR (communication ratio) se asigna el mismo procesador que su padre de tener un CCR mayor a 1, en el caso de tener varios hijos se selecciona el de mayor CCR.

El Algoritmo 7 asigna a cada tarea $t_i \in T$ la máquina m_j que la ejecuta en el menor tiempo posible, después agrega a una lista L todas las tareas que no tienen precedentes y ejecuta el algoritmo 8 sobre cada $t_i \in L$.

Algorithm 7 Menor costo por tarea + CCR

Require: lista vacía L de tareas
1: **for** $x = 0$ to $|T| - 1$ **do**
2: $t_{actual} = x$
3: $m_j^* = argmin\{P_{ij}\} \forall m_j \in M$
4: **end for**
5: **for** $x = 0$ to $|T| - 1$ **do**
6: $t_{actual} = x$
7: **if** $((u, actual) \in E) = null$ **then**
8: agrega a L t_{actual}
9: **end if**
10: **end for**
11: CCR(t_i) $\forall$ $t_i \in L$

El algoritmo 8 asigna recursivamente a las tareas hijo con mayor CCR la misma máquina que el padre si el CCR es mayor a 1

Algorithm 8 Cálculo CCR Recursivo

Require: $tarea_{actual}$
1: CCR(t_{actual})
2: **if** $((u, actual) \in E) = null$ **then**
3: **return**
4: **end if**
5: u^* = argmax $\forall$ (u,actual) $\in$ E $\{(u, actual)/Pu_j + Pt_{actual\,j}\}$
6: **if** ((u,actual) $\in$ E $\{(u, actual)/Pu_j + Pt_{actual\,j}\})> 1$ **then**
7: Asigna a t_{actual} la misma maquina que t_u
8: **end if**
9: CCR(t_{u^*})

Los resultados del constructor CCR utilizando el blevel Tabla 4.9 y utilizando ordenes generados aleatoriamente Tabla 4.10 no producen diferencias en la calidad.

Tabla 4.9: Resultados utilizando orden de blevel

Instancia	*Makespan* óptimo	Blevel+CCR	Error
sample10_2.txt	56	92	64.2 %
sample_8_3.txt	66	91	37.87 %
sample10_3GA.txt	143	179	23.0 %
sample10_3.txt	143	151	5.5 %
sample10_0.dag	73	80	9.5 %
sample11_3.txt	27	39	44.4 %
sample13_3102.txt	112	129	15.1 %

Tabla 4.10: Resultados utilizando 10 orden de aleatorio

Instancia	*Makespan* óptimo	Aleatorio+CCR	Error
sample10_2.txt	56	92	64.2 %
sample_8_3.txt	66	91	37.87 %
sample10_3GA.txt	143	191	33.5 %
sample10_3.txt	143	151	5.5 %
sample10_0.dag	73	80	9.5 %
sample11_3.txt	27	39	44.4 %
sample13_3102.txt	112	129	15.1 %

Capítulo 5

Makespan y su relación con la energía

En este capítulo de tesis se analiza la relación entre el *makespan* y la energía con el fin de identificar si realmente un menor *makespan* conlleva siempre un ahorro en energía.

5.1. Makespan Exacto

Para calcular el óptimo global de *makespan* se implementó (Algoritmo 9) exacto de enumeración lexicográfica de permutaciones sin repetición.

Algorithm 9 Permutaciones sin repetición $O(|T|^2)$

Require: Permutación $A = \{a_i, \ldots, a_{|T|}\}$.

for $x = 0$ to $|T|$ **do**
  $a_i = x$
end for
$i = 1$
while i $\neq 0$ **do**
  Imprimir $\{a_1 - 1, \ldots, a_{|T|} - 1\}$
  $i = |T| - 1$
  while $a_i > a_{i+1}$ **do**
    $i = i - 1$
  end while
  $j = |T|$
  while $a_i > a_j$ **do**
    $j = j - 1$
  end while
  Intercambio(a_i, a_j)
  $r = |T|$
  $s = i + 1$
  while $r > s$ **do**
    Intercambio(a_r, a_s)
    $r = r - 1$
    $s = s + 1$
  end while
end while

Con el algoritmo anterior obtenemos todas los posibles órdenes de ejecución de tarea y con el Algoritmo 10 verificamos su factibilidad.

Algorithm 10 Verifica Factibilidad orden de ejecución de tareas $O(|T|)$

Require: Permutación $A = \{a_i, \ldots, a_n\}$
Require: Arreglo de tareas ejecutadas $B = \{b_i, \ldots, b_n\}$.
Require: Grafo $G = (T, E)$ representando al programa paralelo.
1: **for** $x = 0$ to $|T| - 1$ **do**
2: $b_x = 0$
3: **end for**
4: **for** $x = 0$ to $|T| - 1$ **do**
5: $t_{actual} = a_i$
6: **if** $((u, actual) \in E)$ **then**
7: **if** $A[u] = 0$ **then**
8: **return false**
9: **end if**
10: **end if**
11: $B[t_{actual}] = 1$
12: **end for**
13: **return true**

El algoritmo exacto de *makespan* esta formado por el Algoritmo 9, 10, y 11 el Algoritmo 11 genera las permutaciones con repetición de tamaño $|T|$ (número de tareas), con $|M|$ elementos (número de máquinas) como se ilustra en el Algoritmo 11.

Algorithm 11 Permutaciones con repetición $O(|T||M|)$

Require: Permutación $A = \{a_0, \ldots, a_{|T|}\}$
1: **for** $x = 0$ to $|T|$ **do**
2: $a_x = 1$
3: **end for**
4: **while** $a_0 = 1$ **do**
5: Imprimir $\{a_1 - 1, \ldots, a_{|T|} - 1\}$
6: $j = |T|$
7: **while** $a_j = |M|$ **do**
8: $a_j = 1$
9: $j = j - 1$
10: **end while**
11: $a_j = a_j + 1$
12: **end while**

Por último para tener una solución candidata completa de nuestro problema hace falta la selección de niveles de voltaje, pero para el caso del cálculo de *makespan* óptimo se asume que cada máquina trabaja al máximo de su capacidad.

Con estos algoritmos implementados se logró encontrar los óptimos globales de *makespan* de algunas instancias mostrados en la Tabla 5.1, la columna de órdenes factibles representa la cantidad de órdenes que no violan la precedencia de tareas.

Tabla 5.1: Resultados Exacto Makespan

Instancia	Máquinas	Tareas	órdenes factibles	órdenes con óptimo	óptimo global
sample10_2.txt	2	11	23454	816	56
sample_8_3.txt	3	8	70	40	66
sample10_3GA.txt	3	10	336	336	143
sample10_3.txt	3	10	1680	343	143
sample10_0.dag	3	10	1680	708	73
sample11_3.txt	3	11	990	990	27
sample13_3102.txt	3	13	3360	1110	112

Figura 5.1: Órdenes de ejecución instancia sample_8_3.txt

De la experimentación anterior podemos concluir lo siguiente:

- Los órdenes de ejecución de tareas factibles en las instancias probadas no representan ni el 1 % del total de permutaciones sin repetición
- No todos los órdenes de ejecución de tareas factibles contienen el óptimo de *makespan*

Por lo tanto las heurísticas de este problema deben:

- Evitar explorar órdenes no factibles ya que la región de no factibles es muy grande
- Diversificar en órdenes de ejecución evitando caer en un orden sin óptimo de *makespan*

5.2. Relación energía makespan

Observando una solución óptima de *makespan* en Tabla 5.2 se observa que la instancia de sample_8_3.txt tiene 3 máquinas y para alcanzar el óptimo de *makespan* la solución solo hace uso de 2 máquinas.

Tabla 5.2: Una de las soluciones óptimas $Makespan$ Instancia prueba

Tarea	0	1	2	3	4	5	6	7
Orden	0	1	2	3	4	6	5	7
Máquina	2	0	0	2	2	2	2	2
Voltaje	0	0	0	0	0	0	0	0

lo cual conlleva a las siguientes implicaciones :

- No es siempre necesario usar todas las máquinas disponibles para alcanzar el óptimo de *makespan*
- Las máquinas que no se usen en la solución candidata producirán un 100 % del *makespan* en energía *idle*

Como el modelo de problema no considera el apagado de máquinas estas consumirán un 100 % del tiempo de *makespan* en voltaje mínimo referido en este trabajo como energía en *idle*. Por lo tanto es deseable un *makespan* pequeño con el fin de evitar gasto de energía en *idle*.

5.2.1. Búsquedas Locales para $Makespan$

A continuación se investiga haciendo uso de Búsquedas Locales si existe siempre una relación de mejora directa en el objetivo de energía mediante la reducción del *makespan*.

Búsqueda Local aleatoria

La forma de generación de una solución vecina aleatoria esta dada por una máquina $m_j \in M$ y una tarea $t_i \in T$.

Algorithm 12 Aleatoria

Require: Solución candidata S

repeat
  for $i = 0 < MAXSTEPS$ **do**
    $m_j = rand()M$
    $t_i = rand()T$
    $neighbor =$ Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j
    if $neighbor.C_{max} < S.C_{max}$ **then**
      $S \leftarrow neighbor$
      $i \leftarrow 0$
    end if
  end for
until $sinmejora = MAX\_LOCAL\_STEPS$

Tabla 5.3: configuración máquina/voltaje

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2
voltaje	1	4	2	0	0	0	0	0

Ejemplo con $t_i = 4$ y $m_j = 2$ obtenemos:

Tabla 5.4: configuración máquina/voltaje

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	2	2	0	2
voltaje	1	4	2	0	0	0	0	0

La búsqueda local procede iterativamente generando soluciones aleatorias hasta el criterio de paro, en este caso MAX_LOCAL_STEPS soluciones vecinas visitadas continuas sin mejora.

Búsqueda Local First

La forma de exploración de soluciones vecinas esta dada por una máquina $m_j \in M$, y una exploración secuencial del orden de ejecución de tareas O.

Algorithm 13 First Izquierda a Derecha

Require: Solución candidata S
Require: orden de ejecución de tareas $orden$

```
repeat
  maquina = rand()M
  sinmejoras++
  for i = 0 < |T| do
    t_i = orden[i]
    neighbor = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j
    if neighbor.C_max < S.C_max then
      S ← neighbor
      sinmejora ← 0
      break
    end if
  end for
until sinmejora = MAX_LOCAL_STEPS
```

La exploración en la Búsqueda Local $first$ de el vecindario de una solución candidata puede ser de izquierda a derecha, o de derecha a izquierda, el Algoritmo 13 empieza a explorar desde la tarea 0 hasta la tarea $|T|$ denominado de izquierda a derecha ver Tabla 5.5

Izquierda a derecha Ejemplo con $m_j = 0$ las soluciones vecinas visitadas pueden ser:

Tabla 5.5: posibles vecinas visitadas máquina/tarea

a)

Tarea	**0**	1	2	3	4	5	6	7
máquina	**0**	2	1	0	0	2	0	2

b)

Tarea	0	**1**	2	3	4	5	6	7
máquina	0	**0**	1	0	0	2	0	2

c)

Tarea	0	1	**2**	3	4	5	6	7
máquina	0	2	**0**	0	0	2	0	2

d)

Tarea	0	1	2	**3**	4	5	6	7
máquina	0	2	1	**0**	0	2	0	2

e)

Tarea	0	1	2	3	**4**	5	6	7
máquina	0	2	1	0	**0**	2	0	2

f)

Tarea	0	1	2	3	4	**5**	6	7
máquina	0	2	1	0	0	**0**	0	2

g)

Tarea	0	1	2	3	4	5	**6**	7
máquina	0	2	1	0	0	2	**0**	2

h)

Tarea	0	1	2	3	4	5	6	**7**
máquina	0	2	1	0	0	2	0	**0**

Al encontrar la primera mejora en la Tabla 5.5 se toma esa solución como la actual, se vuelve a generar m_j aleatoriamente y repite el proceso iterativamente, hasta MAX_LOCAL_STEPS recorridos continuos del orden O sin mejora.

Derecha a Izquierda Una variante de $first$ explora el orden de ejecución de tareas de derecha a izquierda, es decir de la tarea $|T| - 1$ a la 0.

Búsqueda Local Best

La forma de exploración de soluciones vecinas esta dada por una máquina $m_j \in M$ seleccionada aleatoriamente, y sus $|T|$ soluciones vecinas.

Algorithm 14 Best

Require: Solución S
Require: orden de ejecución de tareas $orden$
1: $bestsol = S$
2: **repeat**
3: $m_j = rand()M$
4: $sinmejora++$
5: **for** $\forall t_i \in T$ **do**
6: $tarea = orden[i]$
7: $neighbor$ = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j
8: **if** $neighbor.C_{max} < bestsol.C_{max}$ **then**
9: $bestsol \leftarrow neighbor$
10: $sinmejora \leftarrow 0$
11: **end if**
12: **end for**
13: $S = bestsol$
14: **until** $sinmejora = MAX\_LOCAL\_STEPS$

La exploración del vecindario en la Búsqueda Local *best* en una solución candidata es indiferente de el orden en que se visitan los vecinos, siempre se genera todo el vecindario y la solución actual se actualiza a la mejor encontrada ver Tabla 5.6.

Ejemplo con $m_j = 0$ las soluciones vecinas visitadas son:

Tabla 5.6: vecinas visitadas máquina/tarea

a)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2

b)

Tarea	0	1	2	3	4	5	6	7
máquina	0	0	1	0	0	2	0	2

c)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	0	0	0	2	0	2

d)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2

e)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2

f)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	0	0	2

g)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2

h)

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	0

A diferencia de la búsqueda local $first$ la *best* evalúa todos los posibles movimientos en la Tabla 5.6 y toma como solución actual la mejora más significativa, la búsqueda *best* implementada no permite empeorar.

Búsqueda Local Improve

La forma de exploración de soluciones vecinas estada dada por una máquina $m_j \in M$, y una exploración secuencial del orden de ejecución de tareas O. A diferencia de la $first$ que al encontrar una mejora vuelve a empezar el proceso desde la generación de m_j aleatoriamente, $Improve$ toma la mejora como la solución actual y las siguientes vecinas en el orden se generan a partir de la solución actual.

Algorithm 15 Improve Izquierda a Derecha

Require: Solución candidata S
Require: orden de ejecución de tareas *orden*

```
repeat
    m_j = rand()M
    sinmejora++
    for i = 0 < |T| do
        t_i = orden[i]
        neighbor = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j
        if neighbor.C_max < S.C_max then
            S ← neighbor
            sinmejora ← 0
        end if
    end for
until sinmejora = MAX_LOCAL_STEPS
```

La Tabla 5.7 muestra una hipotetica mejora encontrada con $m_j = 1$ y la Tabla 5.8 muestra las siguientes vecinas visitadas hasta terminar la exploración con $m_j = 1$, como se observa el proceso de exploración de vecinos no se reinicializa al encontrar una mejora y se terminan de explorar hasta llegar a la última tarea apartir de la solución actual.

Izquierda a derecha Ejemplo con $m_j = 1$:

Tabla 5.7: Hipotetica vecina visitada con primera mejora máquina/tarea

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	0	2	0	2

Tabla 5.8: Soluciones vecinas visitadas sin mejora máquina/tarea

a)

Tarea	0	1	2	3	4	5	6	7
máquina	1	1	1	0	0	2	0	2

b)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	0	2	0	2

c)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	1	0	2	0	2

d)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	1	2	0	2

e)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	0	1	0	2

f)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	0	2	1	2

g)

Tarea	0	1	2	3	4	5	6	7
máquina	1	2	1	0	0	2	0	1

Derecha a Izquierda Una variante de *Improve* explora el orden de ejecución de tareas de derecha a izquierda, es decir de $|T| - 1$ a 0

Resultados

La Tabla 5.9 muestra los resultados de la combinación del ordenamiento blevel, el constructor HEFT y diferentes búsquedas locales mientras que la Tabla 5.10 muestra los resultados de la combinación del ordenamiento blevel, constructor CCR y diferentes búsquedas locales. Como se puede observar la combinación con el constructor HEFT genera menos error relativo al óptimo de *makespan*.

Tabla 5.9: Blevel + Resultados heft +BL

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	60	69.5	72.5	76.5	60	72.5
sample_8_3.txt	88	68	80	88	76	88
sample10_3GA.txt	153	153	153	153	153	153
sample10_3.txt	156	156	156	156	156	156
sample10_0.dag	76	79	76	80	79	76
sample11_3.txt	27	29	29	29	27	27
sample13_3102.txt	126	127	123	127	126	123
Suma	686	681.5	689.5	709.5	677	695.5
Error	10.6 %	9.9 %	11.2 %	14.4 %	9.1 %	12.1 %

Tabla 5.10: Blevel + CCR +BL

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	60	63.5	87	81.5	63.5	92
sample_8_3.txt	84	81	69	69	76	81
sample10_3GA.txt	152	162	163	154	162	163
sample10_3.txt	151	151	151	151	151	151
sample10_0.dag	80	80	80	80	80	80
sample11_3.txt	30	29	34	29	29	34
sample13_3102.txt	120	126	120	126	126	120
Suma	677	692.5	704	690.5	687.5	721
Error	9.1 %	11.69 %	13.5 %	11.3 %	10.8 %	16.2 %

La Tabla 5.11 muestra los resultados de la combinación de 10 ordenamientos generados aleatoriamente, el constructor HEFT y diferentes búsquedas locales mientras que la Tabla 5.12 muestra los resultados de la combinación de 10 ordenamientos generados aleatoriamente, constructor CCR y diferentes búsquedas locales.

Tabla 5.11: 10 orden aleatorio + heft +BL

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	63.5	58.5	58.5	61.5	58.5	58.5
sample_8_3.txt	66	66	69	66	66	69
sample10_3GA.txt	143	143	143	143	143	143
sample10_3.txt	151	151	147	147	151	147
sample10_0.dag	76	73	73	73	73	73
sample11_3.txt	27	27	27	27	27	27
sample13_3102.txt	121	119	121	122	119	122
Suma	647.5	637.5	638.5	639.5	637.5	639.5
Error	4.43 %	2.8 %	2.9 %	3.1 %	2.8 %	3.1 %

Tabla 5.12: 10 orden aleatorio + CCR +BL

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	62.5	63.5	60	60	60.5	60
sample_8_3.txt	66	66	66	66	66	66
sample10_3GA.txt	143	151	145	145	145	145
sample10_3.txt	151	146	151	151	151	151
sample10_0.dag	80	76	76	76	76	76
sample11_3.txt	27	27	27	27	28	27
sample13_3102.txt	120	119	116	121	121	116
Suma	649.5	648.5	641	646	647.5	641
Error	4.7 %	4.5 %	3.3 %	4.1 %	4.4 %	3.3 %

Se observa los mejores resultados utilizando diferentes órdenes aleatorios en combinación con el constructor HEFT, las combinaciones con búsqueda local first e improve derecha son las mejores en el conjunto de prueba.

5.2.2. DVFS

Se utiliza la técnica $DVFS$ para optimizar el consumo de energía siempre y cuando esta no aumente el *makespan* final. El Algoritmo 16 configura cada una de las tareas a su voltaje minimo posible partiendo de su máxima configuración (voltaje más bajo), al encontrar la minima posible sin aumentar el *makespan* la asigna y continua explorando en la siguiente tarea.

Algorithm 16 técnica $DVFS$ para optimizar energía

Require: Solución candidata S
Require: Orden de ejecución de tareas *orden*
for $i = 0$ to $T - 1$ **do**
  $t_i = orden[i]$
  $m_j \leftarrow$ máquina asignada a t_i en *candidate*
  for $j = MAX(v_k \in m_j)$ to $MIN(v_k \in m_j)$ **do**
    neighbor = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
    if $neighbor.C_{max} = candidata.C_{max}$ **then**
      $S \leftarrow neighbor$
      break {Menor voltaje posible encontrado rompe ciclo for}
    end if
  end for
end for

La Tabla 5.13 muestra la combinación de el orden blevel, el constructor HEFT y la técnica de optimización de energía dvfs mientras que la Tabla 5.14 muestra la combinación de el orden blevel con el constructor CCR y la técnica de optimizaicón de energía dvfs. Se observa que la aplicación de la técnica dvfs obtiene un ahorro de energía similar para ambas combinaciones de algoritmos.

Tabla 5.13: BLEVEL + HEFT + Optimización de energia

Instancia	makespan	Energía sin optimizar	Energía optimizada	Ahorro
sample10_2.txt	76.5	328.3	307.4	6.7 %
sample_8_3.txt	88	518.3	414.3	25.1 %
sample10_3GA.txt	153	833	788	5.71 %
sample10_3.txt	156	1157	1091	6 %
sample10_0.dag	80	500	465	7.5 %
sample11_3.txt	29	171.6	153	12.1 %
sample13_3102.txt	127	717	669	7.1 %
suma	709.5	4225.2	3887.7	8.68 %

Tabla 5.14: BLEVEL + CCR + Optimización de energia

Instancia	makespan	Energía sin optimizar	Energía optimizada	Ahorro
sample10_2.txt	92	356	356	0 %
sample_8_3.txt	91	450	362	24.3 %
sample10_3GA.txt	179	819	747	9.6 %
sample10_3.txt	151	1073	1021	5 %
sample10_0.dag	80	437	395	10.6 %
sample11_3.txt	39	197	190	3.68 %
sample13_3102.txt	129	646	590	9.49 %
suma	761	3978	3661	8.65 %

Como resultado de la observación de las tablas anteriores se concluye lo siguiente:

- Un solución con menor makespan no garantiza un menor consumo energético.
- Existen diferentes configuraciones con el mismo valor de makespan pero diferencias importantes en energía (instancia sample10_0.dag).
- Se deben implementar búsquedas locales multiobjetivo que optimicen los 2 objetivos al mismo tiempo.

5.2.3. Búsqueda Local Iterada

Derivado de la experimentación con búsquedas locales para *makespan* se propone una Iterated Local Search (ILS) Algoritmo 17, utilizando diferentes órdenes de ejecución aleatorios, el constructivo heft genera una solución inicial y se le aplica búsqueda local y perturbación hasta alcanzar el contador de iteraciones sin mejora MAX_STEPS, al alcanzar el estancamiento se repite el proceso con un nuevo orden generado aleatoriamente hasta el número máximo de órdenes $MAXORDERS$, por último se optimiza la energía de la mejor solución encontrada *bestknown* utilizando la técnica dvfs.

Algorithm 17 ILS

```
1:  for i = 0 to MAXORDERS do
2:     sinmejora = 0
3:     orden = GeneraOrdenAleatorio()
4:     solactual = HEFT(orden)
5:     valor = makespan(solactual)
6:     if bestknown > valor then
7:        bestknown = valor
8:     end if
9:     repeat
10:       sinmejora++
11:       solactual = LS(solactual)
12:       if bestknown > valor then
13:          bestknown = valor
14:          sinmejora = 0
15:       end if
16:       solactual = Perturbacion(solactual)
17:    until sinmejora = MAX_STEPS
18: end for
19: DVFS(solactual)
```

La Tabla 5.15 muestra los resultados del ILS implementado para el objetivo *makespan*, meintras que la combinación de órdenes aleatorios, constructor HEFT y búsqueda local en la Tabla 5.11 tiene un valor cercano al 10 % de error, el ILS reduce este margen de error a un valor cercano al 3 %.

Tabla 5.15: ILS con 3 órdenes de ejecución

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	60.5	58.5	61.5	63.5	56	63.5
sample_8_3.txt	66	66	66	68	66	66
sample10_3GA.txt	143	143	143	143	143	143
sample10_3.txt	147	147	147	146	148	148
sample10_0.dag	76	73	76	76	73	76
sample11_3.txt	27	27	28	27	27	27
sample13_3102.txt	112	114	112	112	114	120
Suma	631.5	628.5	633.5	635.5	627	643.5
Error	1.8 %	1.3 %	2.1 %	2.5 %	1.1 %	3.7 %

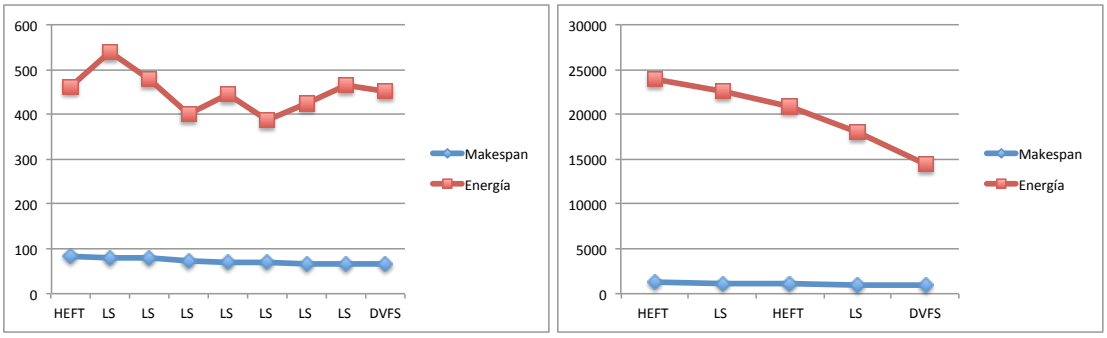
La Tabla 5.16 muestra la energía obtenida depués de la aplicación de la técnica dvfs al *makespan* obtenido por el ILS, se obtuvo un mayor ahorro de energía cuando se utilizo la búsqueda local best.

Tabla 5.16: Energía

Instancia	Aleatoria	first derecha	first izquierda	best	improve derecha	improve izquierda
sample10_2.txt	293.2	287.8	301.8	288.0	292.5	278.3
sample_8_3.txt	454.9	450.0	454.9	426.9	450.0	450.0
sample10_3GA.txt	738.0	738.0	685.1	726.9	738.0	792.7
sample10_3.txt	1091.2	1091.2	1091.2	1117.1	1090.7	1090.7
sample10_0.dag	461.4	415.5	426.7	369.7	413.3	369.7
sample11_3.txt	157.8	157.8	183.1	157.8	157.8	157.8
sample13_3102.txt	603.9	595.6	580.6	580.6	595.6	619.3
Suma	3800.4	3735.9	3723.4	3667	3737.9	3695.5

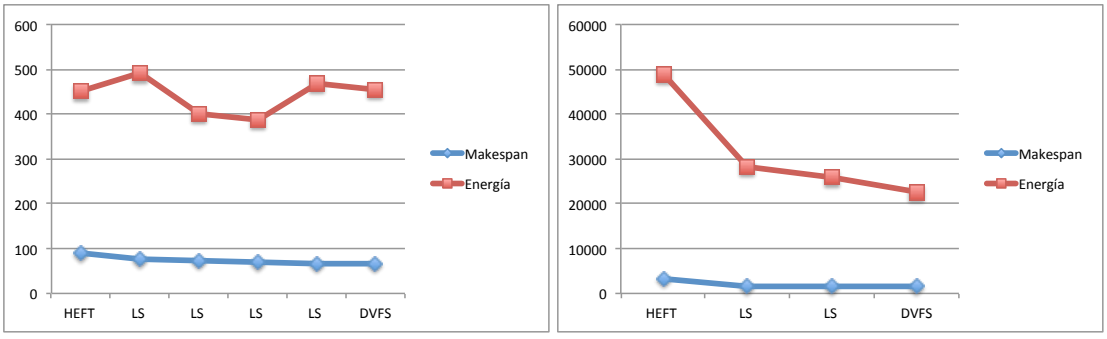
5.2.4. Gráficas de comportamiento

Las siguientes gráficas muestran los cambios en la mejora global del *makespan* e incluyen el comportamiento del valor objetivo de la energía en cada mejora del makespan, indicando que modulo del ILS produce la mejora, el constructor (HEFT), la búsqueda local (LS), el último punto corresponde a la técnica de optimizacion de enería (DVFS).



(a) sample8_3 (b) fpppp_16_334_1_10

Figura 5.2: Graficas de Mejora Global ILS con HEFT



(a) sample8_3 (b) fpppp_16_334_1_10

Figura 5.3: Graficas de Mejora Global ILS con CCR

Como se observa en las gráficas anteriores no siempre que se encuentran mejoras en *makespan* la energía disminuye, en algunas ocaciones el consumo de energía aumenta como se muestra en las figuras de la instancia sample8_3.

Capítulo 6

Estrategias de Búsqueda Local

6.1. Operador de vecindario *Boundary*

El operador de vecindario usado en esta tesis es el operador *Boundary* (en español limite), en los algoritmos genéticos este operador reemplaza un gen en el cromosoma con un valor aleatorio entre su limite inferior y superior valido.

En la representación de la solución generamos soluciones vecinas a partir de una solución como se muestra en la Tabla 6.1 en la cual la tarea 6 esta configurada en la máquina 0 con configuración de voltaje 0, utilizando el operador *Boundary* la tarea 6 puede ejecutarse en la máquina 0, 1, o 2 al aplicarse el cambio a la máquina 1 esta puede seleccionar cualquiera de sus niveles de voltaje validos ver Tabla 6.3 y producir una solución vecina como se muestra en la Tabla 6.2

Tabla 6.1: Solución antes de la aplicación del operador *Boundary*

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	0	2
voltaje	1	4	2	0	0	0	0	0

Tabla 6.2: Solución despues de la aplicación del operador *Boundary*

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	1	2
voltaje	1	4	2	0	0	0	6	0

Tabla 6.3: Configuración de Procesadores

	Máquina 0		Máquina 1		Máquina 2	
Nivel	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad	Voltaje v_k	Velocidad
0	1.75	1	1.50	1	2.20	1
1	1.40	.80	1.40	.90	1.90	.85
2	1.20	.60	1.30	.80	1.60	.65
3	0.90	.40	1.20	.70	1.30	.50
4			1.10	.60	1.00	.35
5			1.00	.50		
6			0.90	.40		

6.2. Bi Objetivo

Existen diversas formas de atacar un problema multi-objetivo. En esté trabajo se usa la calidad de servicio (*QoS* por sus siglas en inglés - *Quality of service*) *QoS* es comúnmente utilizado en las redes de datos, para garantizar la transmisión de cierta cantidad de datos en un tiempo determinado, también se hace uso de la prioridad de objetivos que busca preferentemente mejoras en un objetivo sobre otros.

Los algoritmos Bi Objetivo en esta tesis siguen un enfoque multiobjetivo lexicográfico, dando prioridad a la QoS. En este caso el problema consiste en minimizar el *makespan* (C_{max}) tal que el consumo de energía (E_t) sea lo más bajo posible. Se aplica una técnica basada en el mejor esfuerzo [Haluk Topcuoglu, 2002],[Pecero et al., 2010]. La idea principal es minimizar el *makespan* al máximo, para después utilizar el *makespan* como restricción y optimizar la energía. Esta decisión de diseño se toma por dos razones:

- El constructor $HEFT$ optimiza el makespan y no toma en cuenta la energía
- Una reducción de *makespan* es probable que lleve a una reducción en energía

Se propusieron dos Búsquedas Locales en [Pecero et al., 2012] usando el operador *Boundary*, la $BEST\_RT\_MVk$ la cual escoje una tarea aleatoria y prueba todas las posibles configuraciones máquina/voltaje para esa tarea en la solución candidata, y la $BEST\_RMVk\_T$ la cual escoje una configuración máquina/voltaje valida aleatoria y la prueba en todas las posibles tareas de la solución candidata. El criterio de mejora encontrada en la solucion vecina *neighbor* sobre la mejor solucion encontrada *bestsol* es ($neighbor.C_{max} \leq bestsol.C_{max}$ and $neighbor.E_t < bestsol.E_t$).

El Algoritmo 18 describe la Búsqueda Local Bi Objetivo BEST_RT_RMVk como parámetro de entrada recibe una solución candidata S, la Linea 1 inicializa el contador *searchstep* en 0, la Linea 2 asigna como la mejor solución encontrada *bestsol* la solución S, la Linea 3 es el comienzo del ciclo principal de búsqueda el cual es ejecutado mientras el contador *searchstep* sea menor que el número máximo de exploraciones sin mejora MAX_STEPS, la Linea 4 agrega un paso de exploración al contador *searchstep*, la Linea 5 selecciona aleatoriamente una tarea t_i después en la Linea 5 y Linea 6 prueba todas las posibles configuraciones de máquina $m_j \in M$ y para cada una de ellas sus voltajes válidos $v_k \in m_j$, la Linea 8 se genera una solución vecina *neighbor* apartir de S aplicando el operador *Boundary* asignando la tarea t_i en la máquina m_j con el voltaje v_k, si la solución *neighbor* es mejor que la solución actual mejor conocida *bestsol* actualizamos *bestsol* con la solución *neighbor* y el contador *searchstep* es reinicializado en 0, después que se probaron las posibles configuraciones en la tarea t_i actualizamos la solución candidata S con la mejor solución encontrada *bestsol*. Por último al terminar el proceso la solución candidata S esta garantizada en *makespan* y energía por una calidad igual o mayor ala obtenida con $HEFT$.

Algorithm 18 Búsqueda Local Bi Objetivo BEST_RT_MVk

Require: $S \leftarrow$ Solución creada con el constructor HEFT
1: $searchstep \leftarrow 0$
2: $bestsol \leftarrow S$
3: **repeat**
4: $searchstep++$
5: Selecciona una tarea aleatoria t_i
6: **for** $\forall m_j \in M$ **do**
7: **for** $\forall v_k \in m_j$ **do**
8: $neighbor$ = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
9: **if** $neighbor.C_{max} \leq bestsol.C_{max}$ and $neighbor.E_t < bestsol.E_t$ **then**
10: $bestsol = neighbor$
11: $searchstep \leftarrow 0$
12: **end if**
13: **end for**
14: **end for**
15: $S = bestsol$
16: **until** searchstep = MAX_STEPS

El Algoritmo 19 describe la Búsqueda Local Bi Objetivo BEST_RMVk_T como parámetro de entrada recibe una solución candidata S, la Linea 1 inicializa el contador *searchstep* en 0, la

Linea 2 asigna como la mejor solución encontrada *bestsol* la solución S, la Linea 3 es el comienzo del ciclo principal de búsqueda el cual es ejecutado mientras el contador *searchstep* sea menor que el número máximo de exploraciones sin mejora MAX_STEPS, la Linea 4 agrega un paso de exploración al contador *searchstep*, la Linea 5 selecciona aleatoriamente una máquina $m_j \in M$ y la Linea 6 selecciona aleatoriamente un voltaje $v_k \in m_j$, después se prueba con todas las tareas $t_i \in T$ generando una solución vecina *neighbor* apartir de S aplicando el operador *Boundary* asignando la tarea t_i en la máquina m_j con el voltaje v_k, si la solución *neighbor* es mejor que la solución actual mejor conocida *bestsol* actualizamos *bestsol* con la solución *neighbor* y el contador *searchstep* es reinicializado en 0, después que se probaron las posibles configuraciones en la tarea t_i actualizamos la solución candidata S con la mejor solución encontrada *bestsol*. Por último al terminar el proceso la solución candidata S esta garantizada en *makespan* y energía por una calidad igual o mayor ala obtenida con $HEFT$.

Algorithm 19 Búsqueda Local BI Objetivo BEST_RMVk_T

Require: $S \leftarrow$ Solución creada con el constructor HEFT
1: $searchstep \leftarrow 0$
2: $bestsol \leftarrow S$
3: **repeat**
4: $searchstep++$
5: Selecciona una máquina aleatoria $m_j \in M$
6: Selecciona un voltaje aleatorio $v_k \in m_j$
7: **for** $\forall t_i \in T$ **do**
8: $neighbor =$ Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
9: **if** $neighbor.C_{max} \leq bestsol.C_{max}$ and $neighbor.E_t < bestsol.E_t$ **then**
10: $bestsol = neighbor$
11: $searchstep \leftarrow 0$
12: **end if**
13: **end for**
14: $S = bestsol$
15: **until** searchstep = MAX_STEPS

6.2.1. Iterated Local Search BI Objetivo

El Algoritmo 20 describe el Iterated Local Search Bi Objetivo. Como parámetro de entrada requiere el orden generado por b-level y una cantidad de ordenes MAX_ORDERS generados aleatoriamente. En la Linea 1 asignamos a la solución candidata *candidate* la solución construida por $HEFT$ con b-level, en la Linea 2 construimos nuevas soluciones *newsol* a partir de $HEFT$ y los MAX_ORDERS guardados en *lista*, si alguna solución construida *newsol* es mejor que *candidate* actualizamos *candidate* con *newsol*. Enseguida, en la Linea 8 asignamos a la mejor solución conocida *bestsol* la solución *candidate*. La Linea 9 es el ciclo principal de nuestro ILS Bi Objetivo proceso que se repite mientras el contador *searchstep* sea menor que el número máximo de pasos MAX_SETPS, en la Linea 10 agregamos un paso de búsqueda al contador *searchstep*. Después en la Linea 11 aplicamos una Búsqueda Local Bi Objetivo BEST_RMVk_T o la BEST_RT_RMVk a *candidate*, si se encuentra una solución mejor conocida que *bestsol* actualizamos *bestsol* con *candidate* en la Linea 13 y reinicilizamos el contador *searchstep* en 0 en la Linea 14, si no se encontraron mejoras en *candidate* respecto a *bestsol* se perturba la solución *candidate*.

Algorithm 20 Iterated Local Search

Require: orden generado por b-level y lista de ordenes generados aleatorios
1: $candidate \leftarrow HEFT(b-level)$
2: **for** $i = 0 < MAX\_ORDERS$ **do**
3: $newsol = HEFT(lista(i))$ Construye nueva solución a partir de un orden aleatorio en lista
4: **if** $newsol.C_{max} \leq candidate.C_{max}$ and $newsol.E_t < candidate.E_t$ **then**
5: $candidate = newsol$
6: **end if**
7: **end for**
8: $bestsol = candidate$
9: **repeat**
10: $searchstep++$
11: $LocalSearch(candidate)$
12: **if** $candidate.C_{max} \leq bestsol.C_{max}$ and $candidate.E_t < bestsol.E_t$ **then**
13: $bestsol = candidate$
14: $searchstep$=0
15: **end if**
16: $candidate = Disturbance(candidate)$ Perturba la solución
17: **until** $searchstep = MAX\_STEPS$
18: **return** $bestsol$

En la perturbación se aplica el operador de vecindario *Boundary* a cada tarea de la solución candidata, *candidate* tiene un 5 % de probabilidad de ser cambiada como se muestra en el Algoritmo 21.

Algorithm 21 perturbación

Require: Solución candidata *candidate*
1: **for** $\forall t_i \in T$ **do**
2: $number = random(1:100)$ Genera un número aleatorio entre 1 y 100
3: **if** $number <= probability$ **then**
4: $Boundary(candidate)$ Aplica el operador boundary en la tarea t_i
5: **end if**
6: **end for**

6.3. Multi Objetivo

La principal diferencia entre la optimización multiobjetivo y la optimización mono objetivo viene dada por el concepto de óptimo. En la optimización mono objetivo se dice que el óptimo global es aquella solución x en el espacio de soluciones Ω tal que $\neg\exists x' \in \Omega$ que mejore su valor objetivo $f(x)$. En la optimización multiobjetivo se tiene, un vector de soluciones, al conjunto de soluciones óptimas globales se le conoce como el verdadero frente de Pareto.

Este conjunto viene dado por la dominancia de Pareto en el que para una solución $x \in \Omega$ no existe otra solución $x' \in \Omega$ la cual sea igual de buena en cada uno de sus objetivos y mejor en por lo menos uno de ellos, como se definió en el Capítulo 2.

6.3.1. Pareto Búsqueda Local

Proponemos dos Pareto Búsqueda Local [Basseur et al.,] basadas en [Pecero et al., 2012]. En problemas de un solo objetivo de optimización como se definió en el Capítulo 2 las búsquedas locales tradicionalmente se desplazan en el espacio de soluciones guiadas por una solución inicial visitando a sus soluciones vecinas en busca de mejoras, pero en la optimización multi objetivo una solución está relacionada con un conjunto de soluciones, siempre y cuando ninguna solución en el conjunto sea dominada por otra del mismo conjunto, debido a esto las mejoras deben

modificar el conjunto completo de soluciones no dominadas (Random Non Dominated Point Set RNDP) y actualizarlo.

PARETO_RT_MVk

El Algoritmo 22 muestra una Pareto Local Search (PLS) PARETO_BEST_RT_RMvk basada en [Pecero et al., 2012], la Linea 1 inicializa el contador *searchstep* en 0. En la la Linea 2 el bucle principal es ejecutado hasta que el contador *searchstep* alcanza el número máximo de pasos MAX_STEPS, la Linea 3 incrementa el contador *searchstep*, por cada vez que el bucle principal es ejecutado en la linea 4 cada solucion en el RNDP actual es copiada a una lista temporal *temp*. Después en la Linea 6 se remueve una solución x de la lista temporal *temp* y en la Linea 7 el contador *localstep* es iniciado en 0. Después otro ciclo es ejecutado en la Linea 8 por cada solución x removida hasta que el contador *localstep* alcanza un máximo número de pasos MAX_LOCAL_STEPS, La linea 9 incrementa el contador *localstep* y en la Linea 10 se selecciona aleatoriamente una tarea t_i. En las Lineas 11 12 y 13 se aplica el operador *Boundary* $\forall m_j \in M$ con sus respectivos voltajes válidos $v_k \in m_j$ y se genera una solución vecina *neighbor*, después en la Linea 14 si no existe ninguna solución $y \in RNDP$ que domine a la solución vecina *neighbor* reiniciamos el contador *localstep* en 0. En la Linea 16 removemos las soluciones $y \in RNDP$ que sean dominadas por *neighbor* y en la Linea 17 agregamos *neighbor* al $RNDP$ y este proceso se repite hasta que *localstep* alcanze el número de pasos MAX_LOCAL_STEPS.

Algorithm 22 Pareto Local Search PARETO_RT_MVk

Require: Lista (RNDP) de Random Non-Dominated Point Set y una lista temporal *temp*

1: $searchstep = 0$
2: **repeat**
3: $searchstep++$
4: copia cada solución en $RNDP$ a *temp*
5: **repeat**
6: remueve una solución x de *temp*
7: $localstep = 0$
8: **repeat**
9: $localstep++$
10: selecciona una tarea aleatoria t_i
11: **for** $\forall m_j \in M$ **do**
12: **for** $\forall (v_k, ms_k) \in DVS_{m_j}$ **do**
13: $neighbor$ = Genera una solución vecina a partir de x cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
14: **if** $\neg\exists\ y \in RNDP | neighbor \prec y$ **then**
15: $localstep = 0$
16: Remueve las soluciones $\in RNDP$ que son dominadas por *neighbor*
17: Agrega *neighbor* a $RNDP$
18: **end if**
19: **end for**
20: **end for**
21: **until** $localstep = MAX\_LOCAL\_STEPS$
22: **until** *temp* este vacio
23: **until** $searchstep = MAX\_STEPS$

PARETO_BEST_RMvk_T

El Algoritmo 23 muestra una Pareto Local Search (PLS) PARETO_BEST_RMVk_T basada en [Pecero et al., 2012]. La Linea 1 inicializa el contador *searchstep* en 0. En la Linea 2 el bucle principal es ejecutado hasta que el contador *searchstep* alcanza el número máximo de pasos MAX_STEPS. Después la Linea 3 incrementa el contador *searchstep*, por cada vez que el bucle principal es ejecutado en la Linea 4 cada solucion en el RNDP actual es copiada a una

lista temporal *temp*. En la Linea 6 se remueve una solución x de la lista temporal *temp* y en la Linea 7 el contador *localstep* es iniciado en 0, después otro ciclo es ejecutado en la Linea 8 por cada solución x removida hasta que el contador *localstep* alcanza un máximo número de pasos MAX_LOCAL_STEPS, La linea 9 incrementa el contador *localstep* y en la Linea 10 se selecciona aleatoriamente una máquina m_j, la Linea 11 selecciona aleatoriamente un voltaje $v_k \in m_j$, en las Lineas 12 13 se aplica el operador *Boundary* $\forall t_i \in T$ y se genera una solución vecina *neighbor*, después en la Linea 14 si no existe ninguna solución $y \in RNDP$ que domine a la solución vecina *neighbor* reiniciamos el contador *localstep* en 0 en la Linea 16 removemos las soluciones $y \in RNDP$ que sean dominadas por *neighbor* y en la Linea 17 agregamos *neighbor* al $RNDP$ y se repite hasta que *localstep* alcanze el número de pasos MAX_LOCAL_STEPS.

Algorithm 23 Pareto Local Search PARETO_BEST_RMVk_T

Require: Lista (RNDP) de Random Non-Dominated Point Set
Require: lista temporal de soluciones *temp*
1: $searchstep = 0$
2: **repeat**
3: $searchstep++$
4: copia cada solución en $RNDP$ a *temp*
5: **repeat**
6: remueve una solución x from *temp*
7: $localstep = 0$
8: **repeat**
9: $localstep++$
10: selecciona una máquina aleatoria m_j
11: selecciona un voltaje aleatorio $v_k \in m_j$
12: **for** $\forall t_j \in T$ **do**
13: $neighbor$ = Genera una solución vecina apartir de x cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
14: **if** $\neg\exists\ y \in RNDP | neighbor \prec y$ **then**
15: $localstep = 0$
16: Remueve las soluciones $\in RNDP$ que son dominadas por *neighbor*
17: Agrega *neighbor* a $RNDP$
18: **end if**
19: **end for**
20: **until** $localstep = MAX\_LOCAL\_STEPS$
21: **until** *temp* este vacio
22: **until** $searchstep = MAX\_STEPS$

6.3.2. NSGA-II

El algoritmo de optimización multiobjetivo Non-Dominated Sorted Genetic Algorithm 2 (NSGA-II) fue propuesto por [Deb et al., 2000] como una mejora al NSGA [Srinivas and Deb, 1995] con el fin de atacar 3 debilidades del NSGA, la complejidad de el método de ordenamiento de frentes en el NSGA es de $O(mN^3)$ y en el NSGA-II es de $O(mN^2)$, la falta de elitismo en el NSGA y la necesidad de un parámetro σ_{share} para asegurar la diversidad.

A continuación se describen los módulos en la implementación de esta tesis del NSGA-II y los operadores utilizados en la selección cruza y mutación.

fast_nondominated_sort

El algoritmo de fast_nondominated_sort [Deb et al., 2000] es uno de los componentes esenciales del NSGA-II ordena un conjunto de soluciones P en diversos conjuntos de soluciones no dominadas F, también conocidos como frentes, es el principal encargado de el elitismo en el NSGA-II ya que ordena los frentes por conjuntos de soluciones F_i que dominan los conjuntos $F_{i+1}, F_{i+2}, ...F_{i+n}$ en F.

El Algoritmo 24 muestra el fast_nondominated_sort el cual toma como entrada un conjunto de soluciones P, el ciclo en las Lineas 1 2 comprueba para cada una de las soluciones $p \in P$ si existe otra $q \in P$ tal que $p \prec q$ de cumplirse la Linea 4 agrega la solución q al subconjunto de soluciones dominadas por p llamado S_p, en caso comprueba si $q \prec p$ de cumplirse se aumenta el contador de soluciones que dominan a p llamado n_p, en el caso de no haberse encontrado ninguna solución q que domine a p el contador $n_p = 0$ de cumplirse esto significa que p pertenece al primer frente F_1 y se agrega a este frente en la Linea 12.

Después de encontrar el primer frente F_1 el algoritmo prosigue en encontrar los demás existentes, en la Linea 15 inicializamos $i = 1$ para ubicarnos en el primer frente F_1 y en la Linea 16 se ejecuta un ciclo while mientras el frente actual F_i no este vacío, inicializa el frente temporal H en vacío y en las Lineas 18 19 comprueba para cada una de las soluciones $p \in F_i$ las soluciones que domina $q \in S_p$ y por cada una de ellas le resta una unidad en su contador de soluciones que la dominan n_q, en el momento que $n_q = 0$ se agrega q al frente temporal H, al finalizar el ciclo de las Lineas 18 19 actualiza $i = i + 1$ para ubicarse en el siguiente frente y asignarle las soluciones guardadas en el frente temporal H. Este proceso se repite hasta encontrar todos los frentes F_i.

Algorithm 24 fast_nondominated_sort

Require: Conjunto de soluciones P
1: **for** $\forall p \in P$ **do**
2: **for** $\forall q \in P$ **do**
3: **if** $p \prec q$ **then**
4: $S_p = S_p \cup \{q\}$
5: **else**
6: **if** $q \prec p$ **then**
7: $n_p = n_p + 1$
8: **end if**
9: **end if**
10: **end for**
11: **if** $n_p = 0$ **then**
12: $F_1 = F_1 \cup p$
13: **end if**
14: **end for**
15: $i = 1$
16: **while** $F_i \neq 0$ **do**
17: $H = 0$
18: **for** $\forall p \in F_i$ **do**
19: **for** $\forall q \in S_p$ **do**
20: $n_q = n_q - 1$
21: **if** $n_q = 0$ **then**
22: $H = H \cup \{q\}$
23: **end if**
24: **end for**
25: **end for**
26: $i = i + 1$
27: $F_i = H$
28: **end while**

crowding_distance_assignament Normalizado

La crowding distance sirve como indicador de diversidad de una solución, esta cantidad es referida en los individuos i como $i_{distance}$ y es un estimado de el hypercubo más grande encerrando al individuo i sin incluir otro miembro en su población [Deb et al., 2000].

En el Algoritmo 25 muestra el calculo de la crowding distance, algunos autores sugieren normalizarla como se hace en [Pedersen and Goldberg, 2004] en la implementación de este trabajo se normaliza en la Linea 10 con los valores mínimo y máximo por objetivo m, MIN_m y

MAX_m, los cuales son dinámicos y actualizados en cada nueva población Q_t, en lugar de usar ∞ para los puntos limite (mayor y menor valor objetivo) como se hace en [Deb et al., 2000], se usa el máximo valor objetivo conocido MAX_m.

El Algoritmo 25 requiere un conjunto de soluciones I para calcularles su distancia, la linea 1 asigna a l el número de soluciones en I, en la Linea 2 3 inicializa la distancia de todos los individuos $i \in I$ con el valor 0, después el ciclo principal de la linea 5 ordena las soluciones $i \in I$ por cada uno de sus valores objetivos m, el método de ordenamiento usado en esta tesis es QuickSort, en las lineas 7 8 aumentamos la distancia de los individuos en la posición $I[1]$ y $I[l]$ en MAX_m, el ciclo de la linea 9 aumenta la distancia para el resto de las soluciones con la diferencia de los valores objetivo $(I[i+1]_m - I[i-1]_m)$ y el valor obtenido se normaliza dividiendo entre $(MAX_m - MIN_m)$.

Algorithm 25 Normalized crowding_distance_assignament

Require: Conjunto de soluciones I
1: $l = |I|$
2: **for** $\forall i \in I$ **do**
3: $I[i].distance = 0$
4: **end for**
5: **for** $\forall$ objective m **do**
6: $I = Sort(I, m)$
7: $I[1].distance = I[1].distance + MAX_m$
8: $I[l].distance = I[l].distance + MAX_m$
9: **for** $i = 2$ to $l - 1$ **do**
10: $I[i].distance = I[i].distance + ((I[i+1].m - I[i-1].m)/(MAX_m - MIN_m))$
11: **end for**
12: **end for**

El operador de comparación Crowded

El *crowded comparision operator* $(\geq_n)$ [Deb et al., 2000] es el encargado de guiar el proceso de selección a hacia un frente de Pareto uniforme y diverso, para su uso es necesario calcular a cada individuo i de la población 2 atributos en cada generación.

- Frente al que pertenece en el ordenamiento fast_nondominated_sort (i_{rank}) .
- Distancia obtenida por crowding_distance_assignament $(i_{distance})$.

Definiendo el orden parcial $\geq_n$ a continuación:
$i \geq_n j$ Si $(i_{rank} < j_{rank})$ o $((i_{rank} = j_{rank})$ & $(i_{distance} > j_{distance}))$

Es decir que se prefieren las soluciones encontradas en los primeros frentes por el fast_nondominated_sort, en caso de pertenecer al mismo frente se prefieren las soluciones con mayor crowding distance.

Ciclo Principal

El ciclo principal del NSGA-II descrito en [Deb et al., 2000] marca los pasos a seguir en cada generación t, como principio de diseño de los algoritmos genéticos el tamaño de la población debe ser un entero par N, el NSGA-II cuenta con 2 poblaciones por generación t la población de padres P_t y la población de hijos Q_t, y hace uso de una tercera población temporal R_t la cual esta formada por $P_t \cup Q_t$ y un conjunto de frentes encontrados por el método fast_nondominated_sort F.

El Algoritmo 28 en la Linea 1 inicializa t en 1 y en la Linea 2 inicializa las poblaciones P_t y Q_t de la primera generación con individuos generados aleatoriamente, el ciclo principal de la Linea 3 se repite hasta que t alcanza el número máximo de generaciones $GENERACIONES$, la Linea 4 une las poblaciones P_t y Q_t en R_t y en la Linea 5 el método fast_nondominated_sort hace uso de R_t para generar el conjunto de frentes encontrados F, la Linea 6 inicializa $i = 1$ para posicionarnos en el primer frente F_i, de la Linea 7 a la 11 mientras $|P_{t+1}| + |F_i| \leq N$ se

calcula la distancia a los individuos en F_i con el método crowding_distance_assignament y se agregan a la población de padres P_{t+1} al terminar incrementamos i en uno para ubicarnos en el siguiente frente F_i, después en la Linea 12 se verifica si $|P_t|$ es menor N de serlo se rellena con los individuos restantes en el frente actual F_i primero se calcula la distancia de los individuos en F_i en la Linea 8 con el método crowding_distance_assignament después en la Linea 14 se ordena F_i en orden descendente por distancia, de la Linea 15 a la 18 se remueve el primer elemento x de F_i y se agrega a la población de padres P_{t+1} este proceso se repite hasta alcanzar $|P_{t+1}| = N$.

Por último en la Linea 20 generamos la nueva población de hijos Q_{t+1} a partir de la población de padres P_{t+1} y en la Linea 21 aumentamos el contador t en uno.

Algorithm 26 NSGA2

Require: P_t Población de padres
Require: Q_t Población de hijos
1: $t = 1$
2: Genera los individuos de Pt y Qt aleatoriamente
3: **repeat**
4: $R_t = P_t \cup Q_t$
5: $F = fast\_nondominated\_sort(R_t)$ {Donde $F = (F_1, F_2, ...)$ frentes encontrados en R_t}
6: $i = 1$
7: **while** $(|P_{t+1}| + |F_i|) \leq N$ **do**
8: $crowding\_distance\_assignament(F_i)$
9: $P_{t+1} \cup F_i$
10: $i++$
11: **end while**
12: **if** $|P_{t+1}| < N$ **then**
13: $crowding\_distance\_assignament(F_i)$
14: $Sort(F_i, distancia)$ {Ordena F_i en orden decendente usando crowding distance}
15: **repeat**
16: remueve el primer elemento x de F_i
17: $P_{t+1} = P_{t+1} \cup \{x\}$
18: **until** $|P_{t+1}| = N$
19: **end if**
20: $Q_{t+1} = make\_new\_pop(P_{t+1})$ {Utilice selección,cruza y mutación para crear una nueva población Q_{t+1}}
21: $t++$
22: **until** $t = GENERACIONES$
23: **return** P_t

Selección

La selección empleada en este trabajo es mediante torneo de 2 individuos seleccionados aleatoriamente de cada generación de padres P_t se seleccionan $|P_t|$ padres para ser cruzados, el ganador del torneo es seleccionado mediante el *crowded comparision operator* $(>_n)$

Cruza

Se utilizo un solo punto de cruza aleatorio entre la primer y ultima tarea partiendo en 2 la configuración máquina/voltaje formando 2 hijos, el primer hijo se forma con la primera sección del padre 1 y la segunda sección del padre 2, el segundo hijo se forma con la primera sección del padre 2 y la segunda sección del padre 1.

Tabla 6.4: Padre Uno

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	2	1	0	1	2	2
voltaje	1	4	2	0	0	6	0	0

Tabla 6.5: Padre Dos

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	2	1	2
voltaje	1	4	6	2	1	0	6	0

Tabla 6.6: Hijo Uno

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	2	1	0	2	1	2
voltaje	1	4	2	0	0	0	6	0

Tabla 6.7: Hijo Dos

Tarea	0	1	2	3	4	5	6	7
máquina	0	2	1	0	0	1	2	2
voltaje	1	4	6	2	1	6	0	0

Mutación

En la mutación se aplica el operador de vecindario *Boundary*, cada individuo nuevo generado i es mutado, $\forall t_i \in i$ tienen un 5 % de probabilidad de ser mutadas como se muestra en el algoritmo a continuación.

Algorithm 27 Mutación

Require: Candidate solution
1: **for** $\forall t_i \in T$ **do**
2: $number = random(1:100)$ Genera un número aleatorio entre 1 y 100
3: **if** $number <= probability$ **then**
4: $Boundary(candidate)$ Aplica el operador boundary en la tarea t_i
5: **end if**
6: **end for**

6.3.3. NSGA-II MEMETICO

Los Algoritmos Memeticos son técnicas de optimización basadas en dos principios básicos las búsquedas basadas en población (Algoritmos Evolutivos) y la inclusión de mejoras locales (Búsquedas Locales), en este trabajo de tesis proponemos un NSGA-II Memetico haciendo uso del NSGA-II descrito en el Algoritmo 28 y una Búsqueda Local descrita en el Algoritmo 29. El NSGA-II Memetico remplaza la mutación del NSGA-II del Algoritmo 27 y aplica la Búsqueda Local del Algoritmo 29 a las nuevas poblaciones de padres P_{t+1} e hijos Q_{t+1}, el resto del ciclo principal del NSGA-II es identico al ya descrito en el Algoritmo 28.

Algorithm 28 NSGA2

Require: P_t Población de padres
Require: Q_t Población de hijos
1: $t = 1$
2: Genera los individuos de Pt y Qt aleatoriamente
3: **repeat**
4: $R_t = P_t \cup Q_t$
5: $F = fast\_nondominated\_sort(R_t)$ {Donde $F = (F_1, F_2, ...)$ frentes encontrados en R_t}
6: $i = 1$
7: **while** $(|P_{t+1}| + |F_i|) \leq N$ **do**
8: $crowding\_distance\_assignament(F_i)$
9: $P_{t+1} \cup F_i$
10: $i++$
11: **end while**
12: **if** $|P_{t+1}| < N$ **then**
13: $crowding\_distance\_assignament(F_i)$
14: $Sort(F_i, distancia)$ {Ordena F_i en orden decendente usando crowding distance}
15: **repeat**
16: remueve el primer elemento x de F_i
17: $P_{t+1} = P_{t+1} \cup \{x\}$
18: **until** $|P_{t+1}| = N$
19: **end if**
20: $Q_{t+1} = make\_new\_pop(P_{t+1})$ {Utilice selección y cruza para crear una nueva población Q_{t+1}}
21: $local\_search\_mutation(P_{t+1})$ {Aplique mutación de Búsqueda Local a P_{t+1}}
22: $local\_search\_mutation(Q_{t+1})$ {Aplique mutación de Búsqueda Local a Q_{t+1}}
23: $t++$
24: **until** $t = GENERACIONES$
25: **return** P_t

El Algoritmo 29 requiere una solución a mutar S, en la Linea 1 inicializa los pasos de búsqueda *searchstep* en 0 la Linea 2 inicializa la mejor solución encontrada *bestsol* con la solución a mutar S, la Linea 3 el ciclo principal es ejecutado hasta que el contador *searchstep* alcanze el número maximo de pasos MAX_STEP, la Linea 4 incrementa el contador *searchstep* en uno, la Linea 5 selecciona una máquina aleatoria m_j y la Linea 6 un voltaje aleatorio valido $v_k \in m_j$, después el ciclo de la Linea 7 se ejecuta para todas las tareas $t_i \in T$, la Linea 8 genera un vecino *neighbor* a partir de la solución S aplicando el operador *Boundary* a la tarea t_i con la configuración de máquina m_j y voltaje v_k, la Linea 9 verifica si el vecino domina a la mejor solución encontrada *bestol* de ser cierto la Linea 10 asigna a *bestol* la solución *neighbor* y la Linea 11 reinicializa el contador *searchstep* en 0, al terminar el ciclo de la Linea 7 se actualiza la solución S con la solución *bestsol*.

Algorithm 29 Mutación de Búsqueda Local BEST_RMVk_T

Require: Solución a mutar S

1: $searchstep \leftarrow 0$
2: $bestsol \leftarrow S$
3: **repeat**
4: $searchstep++$
5: Selecciona una máquina aleatoria m_j
6: Selecciona un voltaje aleatorio $v_k \in m_j$
7: **for** $\forall t_i \in T$ **do**
8: $neighbor$ = Genera una solución vecina a partir de S cambia la asignación original de t_i con la máquina m_j y el voltaje v_k
9: **if** $neighbor \prec bestsol$ **then**
10: $bestsol \leftarrow neighbor$
11: $searchstep \leftarrow 0$
12: **end if**
13: **end for**
14: $S \leftarrow bestsol$
15: **until** searchstep $<$ MAX_STEPS

Capítulo 7

Resultados Experimentales

A continuación se muestran los resultados experimentales para los algoritmos propuestos en el capítulo de estrategias de Búsqueda Local el cual esta dividido en 2 secciones, algoritmos que encuentran una sola solución optima (Bi Objetivo) e algoritmos que hacen uso de la dominancia de Pareto (Multi Objetivo).

La evaluación experimental fue realizada en una computadora con procesador de arquitectura x86 con 2 núcleos a 3.06Ghz y 4GB de ram a 1066Mhz, el lenguaje de programación utilizado ANSI C estandar.

Los criterios de paro en los algoritmos Bi Objetivo son un máximo de 10 segundos y estancamiento; en los algoritmos Multi Objetivo el criterio de paro es 100 generaciones y un máximo de 1 segundo por aplicación de Búsqueda Local.

Cada experimento fue replicado 10 veces por instancia, los valores utilizados en este análisis son el promedio de esas ejecuciones.

7.1. Bi Objetivo

Se evaluó los algoritmos Bi Objetivo respecto al mejor propuesto, considerando que se supero al estado del arte de Búsquedas Locales en[Pecero et al., 2012].

La Tabla 7.1 muestra los resultados de los algoritmos BEST_RT_MVk, BEST_RMVk_T, ILS Bi Objetivo, para el objetivo *Makespan*, se observa se tiene un error relativo en el algoritmo BEST_RMVK_T de 3.85 % y en el algoritmo BEST_RT_MVk de 6.59 % respecto al ILS Bi Objetivo.

Tabla 7.1: Resultados algoritmos Bi Objetivo en el objetivo Makespan

INSTANCIA	BEST_RMVk_T	BEST_RT_MVk	ILS
fpppp_64_334_0.1_0.1.dag	258.907334	258.9026338	258.8232652
fpppp_64_334_0.1_10.dag	2094.780559	2115.237868	2107.297423
fpppp_64_334_1_0.1.dag	81.526511	81.526511	81.526511
fpppp_64_334_1_10.dag	3954.809075	3993.283665	3947.606036
fpppp_8_334_0.1_0.1.dag	93.1705993	93.2021725	93.1453202
fpppp_8_334_0.1_10.dag	1630.320059	1649.228892	1633.606317
fpppp_8_334_1_0.1.dag	70.5278983	70.6535472	70.4580567
fpppp_8_334_1_10.dag	152.7880729	157.909902	152.9063588
LIGO_64_76_0.1_0.1.dag	602.4838231	602.8863543	598.9252289
LIGO_64_76_0.1_10.dag	1326.515772	1332.437172	1311.643423
LIGO_64_76_1_0.1.dag	295.5611826	297.9674711	295.0102634
LIGO_64_76_1_10.dag	150.8868626	149.5381309	121.9417522
LIGO_8_76_0.1_0.1.dag	566.826641	566.826641	566.826641
LIGO_8_76_0.1_10.dag	1652.366314	1702.229174	1352.167709
LIGO_8_76_1_0.1.dag	116.6286502	116.724039	116.4378726
LIGO_8_76_1_10.dag	1395.616791	1408.1295	1267.752834
robot_64_88_0.1_0.1.dag	215.047893	215.047893	213.4107116
robot_64_88_0.1_10.dag	3196.883093	3276.3829	3063.17576
robot_64_88_1_0.1.dag	842.2022715	841.7940408	840.0761224
robot_64_88_1_10.dag	3229.448192	3286.510518	2833.072625
robot_8_88_0.1_0.1.dag	1845.354866	1845.994857	1845.354866
robot_8_88_0.1_10.dag	2199.555589	2464.855523	2219.422202
robot_8_88_1_0.1.dag	249.9092647	249.9018739	249.9038281
robot_8_88_1_10.dag	821.9019078	835.833084	817.2562536
sparse_64_96_0.1_0.1.dag	137.362006	137.362006	136.6038647
sparse_64_96_0.1_10.dag	133.8692213	139.718266	134.1254782
sparse_64_96_1_0.1.dag	232.437822	232.437822	232.437822
sparse_64_96_1_10.dag	1600.82564	1521.977017	1465.01717
sparse_8_96_0.1_0.1.dag	23.879413	23.879413	23.879413
sparse_8_96_0.1_10.dag	745.1551848	858.590458	735.5863883
sparse_8_96_1_0.1.dag	219.8199277	219.8359594	219.8198248
sparse_8_96_1_10.dag	731.4449728	945.9714691	727.3184899
Total	30868.81341	31692.77677	29732.53583
Error	3.85 %	6.59 %	0

Como se observa en la tabla anterior el ILS Bi objetivo obtiene una ganancia en el objetivo *makespan* con un valor aproximado entre 3 % y 7 %, la ganancia se da debido a una mayor exploración del espacio de soluciones utilizando ordenes generados aleatoriamente.

La Tabla 7.2 muestra los resultados de los algoritmos BEST_RT_MVk, BEST_RMVk_T, ILS Bi Objetivo para el objetivo energía, se observa un error relativo en el Algoritmo BEST_RMVk_T de 3.64 % y en el algoritmo BEST_RT_MVk de 5.39 % respecto al algoritmo ILS Bi Objetivo. Al optimizar ambos objetivos al mismo tiempo es de esperarse que las mejores soluciones encontradas en el ILS Bi Objetivo obtengan un mejor ahorro en energía como se muestra en la tabla.

Tabla 7.2: Resultados algoritmos Bi Objetivo en el objetivo Energía

INSTANCIA	BEST_RMVk_T	BEST_RT_MVk	ILS
fpppp_64_334_0.1_0.1.dag	19500.36235	20588.48113	19469.36164
fpppp_64_334_0.1_10.dag	127662.488	131288.0352	128306.4417
fpppp_64_334_1_0.1.dag	5981.177444	6303.32556	5958.26502
fpppp_64_334_1_10.dag	230528.8343	236313.3153	230292.8242
fpppp_8_334_0.1_0.1.dag	1781.593143	1809.197617	1783.489802
fpppp_8_334_0.1_10.dag	21042.34169	21227.12026	21049.60493
fpppp_8_334_1_0.1.dag	1365.29966	1369.14978	1368.087086
fpppp_8_334_1_10.dag	1681.590137	1625.157129	1687.595047
LIGO_64_76_0.1_0.1.dag	34476.9521	34754.73012	34194.95591
LIGO_64_76_0.1_10.dag	73959.83022	74143.80129	73070.28248
LIGO_64_76_1_0.1.dag	16795.88741	17048.39527	16759.82088
LIGO_64_76_1_10.dag	8296.13217	8220.994474	6728.512231
LIGO_8_76_0.1_0.1.dag	8138.72693	8348.779588	8037.171743
LIGO_8_76_0.1_10.dag	11932.85599	12216.25525	10178.88643
LIGO_8_76_1_0.1.dag	1497.682739	1567.650429	1516.002641
LIGO_8_76_1_10.dag	10679.53256	10957.19176	9707.696516
robot_64_88_0.1_0.1.dag	12209.56572	12220.511	12033.20036
robot_64_88_0.1_10.dag	174804.2285	178877.0498	167425.9379
robot_64_88_1_0.1.dag	46937.01855	47019.01589	46793.14423
robot_64_88_1_10.dag	176790.0771	179619.2702	155763.8109
robot_8_88_0.1_0.1.dag	18666.48524	19109.26896	18488.47792
robot_8_88_0.1_10.dag	16648.80578	17964.95758	16971.1191
robot_8_88_1_0.1.dag	2602.135397	2683.02528	2581.776975
robot_8_88_1_10.dag	6410.56933	6626.710983	6391.416002
sparse_64_96_0.1_0.1.dag	7708.056393	7714.071378	7679.284078
sparse_64_96_0.1_10.dag	7406.09116	7709.800394	7420.112485
sparse_64_96_1_0.1.dag	13024.91525	13126.52711	13001.04038
sparse_64_96_1_10.dag	86827.23082	82734.33787	79845.72172
sparse_8_96_0.1_0.1.dag	500.448148	500.448148	500.448148
sparse_8_96_0.1_10.dag	6565.920637	6993.819888	6600.99684
sparse_8_96_1_0.1.dag	4326.779331	4333.077497	4319.200483
sparse_8_96_1_10.dag	5631.967523	6917.341427	5524.231074
Total	1162381.582	1181930.813	1121448.917
Error	3.64 %	5.39 %	0

Después de obtener los resultados se utilizo la prueba estadística no paramétrica de Friedman [Corder and Foreman, 2009] utilizando la herramienta descrita en [Garcia and Herrera, 2008] para validar los resultados con un nivel de significancia $\alpha = 0.05$

Tabla 7.3: Ranking de los algoritmos en el objetivo *Makespan*

Algoritmo	Ranking
BEST_RMVk_T	1.984375
BEST_RT_MVk	1.34375
ILS	2.671875

P-value computado por la prueba: 7.39591756282465E-7.

Tabla 7.4: Ranking de los algoritmos en el objetivo Enérgia

Algorithm	Ranking
BEST_RMVk_T	2.1875
BEST_RT_MVk	1.15625
ILS	2.65625

P-value computado por la prueba: 6.581389211390842E-9.

Ambos valores de p-value están por debajo del nivel de significancia 0.05 por lo que se concluye que existen diferencias significativas entre los algoritmos, también se observa como el mejor algoritmo en el ranking es el ILS obteniendo el ranking más alto en ambos objetivos.

7.2. Multi Objetivo

El resultado de un algoritmo optimización multiobjetivo es un conjunto de soluciones $A_1 \in \Omega$ de cardinalidad variable usualmente un Random Non-Dominated Point Set (conjunto de soluciones no dominadas entre si) si comparamos con otro algoritmo de optimización que produce un conjunto de soluciones $A_2 \in \Omega$ surgen las siguientes interrogantes:

- ¿Cuando podemos decir que A_1 es mejor que A_2 ($A_1 \lhd A_2$)?
- Si A_1 es mejor ¿que tan mejor es respecto a A_2?
- Si $A_1 \not\lhd A_2$ y $A_1 \not\rhd A_2$ ¿en que aspectos se diferencian y en que cantidad?

De acuerdo ala relación de dominancia en conjuntos de aproximación en [Zitzler et al., 2002] ($A_1 \lhd A_2$) Si y solo si todas las soluciones en A_2 son dominadas por alguna 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 $A_1 \not\rhd A_2$ 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 conjuntos de soluciones se hace uso de los indicadores de calidad multiobjetivo.

7.2.1. Indicadores de Calidad

Los indicadores de calidad multiobjetivo son aceptados por la comunidad científica como referencia del desempeño de un algoritmo frente a otro, sin embargo existen muchos los cuales cuantifican diferentes propiedades deseables en los algoritmos multiobjetivo y su elección e uso no es trivial.

Existen dos ramas principales de indicadores de calidad multiobjetivo los unarios y los binarios, los unarios son indicadores de calidad que se le pueden aplicar a un conjunto de aproximación sin depender de otro para su comparación tienen la desventaja de que requieren previamente conocer el espacio de soluciones o estimarlo, detalles como el verdadero frente de Pareto PF_{true} o sus limites superiores e inferiores deben ser previamente conocidos. Los binarios requieren la comparación de 2 conjuntos por ejemplo cuantas soluciones del conjunto A_1 son dominadas por el conjunto A_2 y viceversa, su desventaja radica en que la comparación solo puede ser entre 2 conjuntos y no múltiples al mismo tiempo.

Los indicadores de calidad utilizados en este trabajo de tesis son unarios, se utilizo el paquete estadístico libre R con la librería mco.

Hypervolume

El Hypervolume [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 MO, representa el volumen del espacio n-dimensional dominado por las soluciones en el conjunto de referencia A esto en un valor único unario conocido como Hypervolume y es una propiedad deseable que este se maximize.

El hypervolume de A es medido de acuerdo a un punto de referencia ref usualmente el anti optimo global es decir una solución x tal que es dominada por todas las soluciones $\in \Omega$, la selección del punto de referencia sigue siendo un problema abierto por lo cual se aconseja tomar el peor valor conocido en cada objetivo como ref y agregarle la unidad [Naujoks and Beume, 2005] como se muestra en la Figura Tal 7.1.

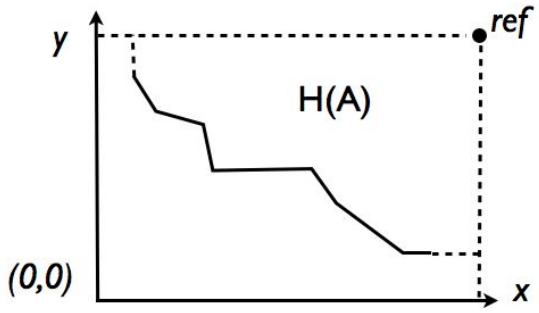


Figura 7.1: Hypervolume para problema de minimizacion de 2 objetivos

Existen diversos algoritmos para calcular el hypervolume como en [Auger et al., 2009], [Yang and Ding, 2007], [Naujoks and Beume, 2005] también esta probado que el calculo del hypervolume es NP-hard [Bringmann and Friedrich, 2009].

Generational Distance

La Generational Distance es un indicador de convergencia del algoritmo, representa la distancia entre el frente encontrado A y el PF_{true} y se define en [Veldhuizen and Lamont, 1998] como:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{7.1}$$

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

Generalized Spread

Generalized Spread es una extensión de la metrica de diversidad Spread en [Veldhuizen and Lamont, 2000] la cual calcula la distancia entre soluciones consecutivas, esto solo funciona para problemas con 2 objetivos, se propuso una extención de esta metrica en [Zhou et al., 2006] calculando la distancia de un punto a su vecino más cercano.

$$GS(PF_{current}, PF_{true}) = \frac{\sum_{i=1}^{m} d(e_i, PF_{current}) + \sum_{X \in PF_{true}} |d(X, PF_{current}) - \overline{d}|}{\sum_{i=1}^{m} d(e_i, PF_{current}) + |PF_{true}|\overline{d}} \tag{7.2}$$

Donde $\{e_1, e_2, ..., e_m\}$ son m soluciones extremas en PF_{true} y

$$d(X, PF_{current}) = \min_{Y \in PF_{current}, Y \neq X} ||F(X) - F(Y)||^2 \tag{7.3}$$

,

$$\overline{d} = \frac{1}{|S^*|} \sum_{X \in S^*} d(X, S) \tag{7.4}$$

Si las soluciones en $PF_{current}$ están bien distribuidas e incluyen las soluciones extremas entonces GS=0.

7.2.2. Resultados

La Tabla 7.5 concentra los resultados de los algoritmos multi-objetivo: NSGA2, NSGA2 Memetico, PLS_RMVk_T y PLS_RT_MVk; Evaluados con los indicadores de calidad multi-objetivo: Hypervolume (H), Generational Distance (GD), Generalized Spread (GS).

Tabla 7.5: Resultados algoritmos Multi Objetivo

Algoritmo	NSGA2			NSGA2 Memetico			PLS_RMVk_T	PLS_RT_MVk
Instancia	H	GD	GS	H	GD	GS	GD	GD
fpppp_64_334_0.1_0.1.dag	2520191	49.96981	0.8770378	3017193	41.21829	0.8549394	8.551745	145.2836
fpppp_64_334_0.1_10.dag	512743937	87.25834	0.9494811	833595428	45.54878	0.840615	114.7383	558.6226
fpppp_64_334_1_0.1.dag	824726.7	7.698271	0.8567601	1073444	5.593551	0.8083547	2.308437	28.89915
fpppp_64_334_1_10.dag	1050897495	5.683548	0.944855	2098059488	2.590658	0.8706549	12.37807	36.2823
fpppp_8_334_0.1_0.1.dag	15944.46	0.06001368	0.4944285	21624.87	0.02574273	0.402244	0.006863407	0.01802862
fpppp_8_334_0.1_10.dag	27583590	2.060232	0.803703	38368965	1.346028	0.7330271	1.188148	3.721112
fpppp_8_334_1_0.1.dag	65085.17	0.8140555	0.7050423	864267	0.6920687	0.6487886	0.1178855	0.3837147
fpppp_8_334_1_10.dag	298167.5	1.655607	0.7232072	326790.8	1.20515	0.6569078	2.036242	2.890934
LIGO_64_76_0.1_0.1.dag	4525549	3.309957	0.7897357	7584165	0.2895859	0.4751579	1.750477	1.187055
LIGO_64_76_1_0.1.dag	10073918	8.558273	0.9209722	17037380	0.4973722	0.5139701	6.18888	15.30091
LIGO_64_76_1_10.dag	1837473	1.947796	0.971842	5453863	0.1701185	0.2915115	2.649538	4.428898
LIGO_8_76_0.1_0.1.dag	3127633	0.207384	0.6494829	5053338	0.03299617	0.447436	0.01864485	0.02335708
LIGO_8_76_0.1_10.dag	38751321	6.89312	0.910353	92533676	0.7915659	0.5572578	17.82753	17.4657
LIGO_8_76_1_0.1.dag	589310.3	0.3980243	0.685116	890971.4	0.02457954	0.4167028	0.06911202	0.0645969
LIGO_8_76_1_10.dag	76506732	5.555945	0.8800716	121137522	0.5769582	0.464782	10.82421	16.25224
robot_64_88_0.1_0.1.dag	234161.7	1.766015	0.8467936	410010.7	0.08870764	0.5186532	1.312092	0.9940087
robot_64_88_0.1_10.dag	6459176328	3.250243	0.9709356	3065082541	0.3890992	0.4650507	10.88686	10.81001
robot_64_88_1_0.1.dag	64860126	21.41969	0.8825936	102226642	0.7071975	0.3019082	4.724204	12.48699
robot_64_88_1_10.dag	2641806829	214.7757	0.9762598	6629239243	17.36316	0.6985106	483.2761	574.3122
robot_8_88_0.1_0.1.dag	10641717	1.258667	0.6929935	22218006	0.2119495	0.3114544	0.116376	0.1000777
robot_8_88_0.1_10.dag	297502306	8.820762	0.9016601	552233473	0.8491864	0.5640044	37.41045	32.0723
robot_8_88_1_0.1.dag	1380864	0.6118564	0.7040222	2266655	0.04452885	0.326791	0.07418506	0.1022148
robot_8_88_1_10.dag	21377497	1.037166	0.8450778	41225246	0.0512869	0.1969596	1.261525	1.62694
sparse_64_96_0.1_0.1.dag	1855093	1.699186	0.8440444	2295985	0.1506781	0.6119155	1.175118	2.44423
sparse_64_96_0.1_10.dag	893157	12.32844	0.9621416	2153744	0.5221583	0.5781105	48.62627	41.80632
sparse_64_96_1_0.1.dag	21172132	12.17249	0.8309727	30440770	0.7651316	0.1700536	1.71803	5.61863
sparse_8_96_0.1_0.1.dag	11135.82	0.06120706	0.4664569	12990.51	0.01048455	0.3963027	0.006961813	0.009515877
sparse_8_96_0.1_10.dag	20665428	0.706737	0.748781	28293083	0.1225503	0.3012799	1.231414	0.7910213
sparse_8_96_1_0.1.dag	3323293	0.1930477	0.575063	4540103	0.03409112	0.3427468	0.04811587	0.0622516
sparse_8_96_1_10.dag	30682876	3.012998	0.8328835	44873265	0.3643735	0.4382417	6.21639	4.912437
Total	11305944017	465.1845816	24.2427677	13752529873	122.2780283	15.2043324	778.7381745	1518.973343
Error respecto a Memetico	17.79 %	280.43 %	59.45 %				536.86 %	1142.23 %

Tabla 7.6: Resumen Resultados MO

Algoritmo	Indicador de Calidad	total	Error
NSGA2	Hypervolume	11305944017	17.79 %
	Generational Distance	465.1845816	280.43 %
	Generalized Spread	24.2427677	59.45 %
NSGA2 Memetico	Hypervolume	13752529873	
	Generational Distance	122.2780283	
	Generalized Spread	15.2043324	
PLS BEST_RMVk_T	Generational Distance	778.7381745	536.86 %
PLS BEST_RT_MVk	Generational Distance	1518.973343	1142.23 %

Las Pareto Local Search PLS_RMVk_T y PLS_RT_MVk no fueron medidas en hypervolume y generalized spread debido al bajo número de soluciones que manejan encontrando en muchas ocaciones una sola solución no dominada; debido a que inicializan con una sola solución candidata, mientras que el NSGA2 y su versión memetico utilizan una población fija.

Las Pareto Local Search son posiblemente sensibles a la utilización de una población de soluciones tan pequeña a consecuencia de su criterio de aceptación por dominancia, esta forma de navegación del espacio de soluciones afecta el desempeño de estos algoritmos produciendo el peor resultado.

La combinación del NSGA2 Memetico produce una mejora de 17.79 % en hypervolume, 280.43 % en generational distance, y 59.45 % en generalized spread respecto al NSGA2, estos resultados estan concentrados en la Tabla 7.6

Para validar estos resultados se utilizo la prueba estadisctica de Mann U Whitney también conocida como prueba de Wilcoxon [Oja, 2008], para un 95 % se rechaza la hypotesis nula H_o que las diferencias entre los algoritmos no son significativas cuando el p-value es menor o igual a 0.05.

Se Observa que ambas Pareto Local Search implementadas son equivalentes, mientras que el NSGA2 es significativamente diferente al NSGA2 Memetico en los indicadores de calidad de Generational Distance y Generalized Spread Ver Tabla 7.7, sin embargo en hypervolume no lo es; posiblemente por que ambos algoritmos utilizan la misma cantidad de individuos en la población afectando el area de dominancia que tienen en el espacio de soluciones.

Tabla 7.7: Resultados MO prueba estadistica de Wilcoxon

Algoritmos	Indicador de Calidad	Wilcoxon p-value	H_o rechazada
NSGA2 VS NSGA2 Memetico	Hypervolume	0.4404	No
	Generational Distance	0.0002269	Si
	Generalized Spread	1.727e-08	Si
PLS_BEST_RT_MVk VS PLS_BEST_RMVk_T	Generational Distance	0.4853	No

Capítulo 8

Conclusiones y trabajos futuros

8.1. Conclusiones

Los resultados de este este proyecto de investigación cumplieron suficientemente con los objetivos planteados inicialmente. Aun cuando los objetivo solo consideraban el desarrollo de Búsquedas Locales Mono y Bi Objetivo, ademas de estas se implementaron métodos de solución Metaheurística para evaluar dichas estrategias. En esta sección se describen las contribuciones científicas más relevantes, las publicaciones y los trabajos futuros identificados en el proyecto.

8.2. Contribuciones científicas

8.2.1. Búsquedas Locales Mono Objetivo

En este trabajo se proponen 3 Búsquedas Locales que permiten minimizar el makespan. Las estrategias de búsqueda evaluadas son: $first$ derecha e izquierda, $Improve$ derecha e izquierda y $best$. Las evaluaciones experimentales realizadas muestran que las dos primeras estrategias son las más eficientes. estas estrategias se evaluaron en el contexto de una Metaheurística de Búsqueda Local Iterada. (Ver capítulo 5).

8.2.2. Métodos de solución Mono-Objetivo

En este contexto se propone un método de solución basado en la Metaheurística de Búsqueda Local Iterada. Este método de solución se utilizó para evaluar las estrategias de Búsqueda Local Mono-Objetivo propuestas (Ver capítulo 5)

8.2.3. Búsquedas Locales Bi Objetivo

En este contexto se aportan 2 Búsquedas Locales Bi Objetivo: BEST_RMVk_T y BEST_RT_MVk. Estas estregias permiten optimizar el *makespan* y la energía, asignandole mayor prioridad al makespan. Actualmente se estan incorporando en un método de solución Metaheurístico como consecuencia de que su evaluación independiente muestra que tienen un alto desempeño. (Ver Capítulo 6)

8.2.4. Búsquedas Local de Pareto

En este contexto se contribuye con 2 Búsquedas Locales Multi-objetivo que permiten optimizar simultaneamente el *makespan* y la energía. Las estrategías se evaluaron de manera independiente, y muestra un desempeño altamente sensible a el tamaño de la población de soluciones. (Ver Capítulo 6)

8.2.5. Métodos de solución Multi-Objetivo

En este ambito se propone un método de solución basado en NSGA-II que incorpora Búsquedas Locales. Los resultados experimentales muestran que el desempeño de este algoritmo supera al algoritmo generado con la estrategia tradicional de NSGA-II. (Ver capítulo 6)

8.3. Publicaciones

Resultados parciales de este proyecto de investigación se han publicado en foros científicos. Una lista de las publicaciones generadas es la siguiente:

- *On the energy optimization for precedence constrained applications using local search algorithms* on the 2012 International Conference on High Performance Computing & Simulation in Madrid (HPCS 2012), Johnatan E. Pecero, Héctor Joaquín Fraire Huacuja, Pascal Bouvry, Aurelio Alejandro Santiago Pineda, Mario César López Locés, Juan Javier González Barbosa [Pecero et al., 2012].
- *Algoritmos exactos de calendarización de tareas para programas paralelos en sistemas de procesamiento heterogéneos* en el VI Encuentro de investigadores en el Instituto Tecnológico de Ciudad Madero (ITCM), Aurelio Alejandro Santiago Pineda, Miguel Ángel Ramiro Zuñiga, Héctor Joaquín Fraire Huacuja [Pineda et al.,]

8.4. Trabajos futuros

Algunos de los trabajos futuros con los que se puede continuar esta investigación son los siguientes:

- Evaluar experimentalmente el NSGA2 con crowding distance sin normalizar contra el propuesto.
- Comparar las Metaheurísticas implementadas contra otras para el problema sin idle.
- Evaluar experimentalmente el NSGA2 Memetico con Búsqueda Local solo en hijos contra el propuesto.

Bibliografía

[Abraham Duarte Muñoz, 2010] Abraham Duarte Muñoz, Juan José Pantrigo Fernández, M. G. C. (2010). Metaheurísticas. Universidad Rey Juan Carlos.

[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 μ-distributions and the choice of the reference point. In In Foundations of Genetic Algorithms (FOGA 2009). ACM.

[Baskiyar and Palli, 2006] Baskiyar, S. and Palli, K. (2006). Low power scheduling of dags to minimize finish times. In Robert, Y., Parashar, M., Badrinath, R., and Prasanna, V., editors, High Performance Computing - HiPC 2006, volume 4297 of Lecture Notes in Computer Science, pages 353–362. Springer Berlin / Heidelberg.

[Basseur et al.,] Basseur, M., Seynhaeve, F., and Talbi, E.-G. Path relinking in pareto multi-objective genetic algorithms.

[Bringmann and Friedrich, 2009] Bringmann, K. and Friedrich, T. (2009). Approximating the least hypervolume contributor: Np-hard in general, but fast in practice. In Ehrgott, M., Fonseca, C., Gandibleux, X., Hao, J.-K., and Sevaux, M., editors, Evolutionary Multi-Criterion Optimization, volume 5467 of Lecture Notes in Computer Science, pages 6–20. Springer Berlin / Heidelberg.

[Coello, 1996] Coello, C. A. C. (1996). An Empirical Study Of Evolutionary Techniques For Multiobjective Optimization In Engineering Design. PhD thesis, Tulane University.

[Corder and Foreman, 2009] Corder, G. W. and Foreman, D. I. (2009). Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach. New Jersey: Wiley.

[Deb, 2001] Deb, K. (2001). Multi-objective optimization using evolutionary algorithms. Wiley-Interscience series in systems and optimization. John Wiley & Sons.

[Deb et al., 2000] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, volume 1917 of Lecture Notes in Computer Science, Berlin, Heidelberg. Springer.

[Duarte et al., 2006] Duarte, A., Laguna, M., and Martí, R. (2006). Tabu search for the linear ordering problem with cumulative costs. Computational Optimization and Applications, pages 1–19.

[Garcia and Herrera, 2008] Garcia, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. Journal of Machine Learning Research, 9:2677–2694.

[Garey et al., 1979] Garey, M. R., Johnson, D. S., and Others (1979). Computers and Intractability: A Guide to the Theory of NP-completeness. WH freeman San Francisco.

[Guzek, 2010] Guzek, M. (2010). Performance and energy optimization by a multi-objective evolutionary algorithm in large-scale distributed systems. Master's thesis, University of Southampton, UK.

[Guzek et al., 2010] Guzek, M., Pecero, J. E., Dorronsoro, B., and Bouvry, P. (2010). A cellular genetic algorithm for scheduling applications and energy-aware communication optimization. In International Conference on High Performance Computing & Simulation (HPCS), pages 241–248.

[Haluk Topcuoglu, 2002] Haluk Topcuoglu, Salim Hariri, M.-Y. W. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, pages 260–274.

[Ilavarasan and Thambidurai, 2007] Ilavarasan, E. and Thambidurai, P. (2007). Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. Journal of Computer Sciences, 3(2):94–103.

[Kang et al., 2011] Kang, Q., He, H., and Song, H. (2011). Task assignment in heterogeneous computing systems using an effective iterated greedy allgorithm. The journal of Systems and Software, 84:985–992.

[Kelly and Osman, 1996] Kelly, J. P. and Osman, I. H. (1996). Meta-Heuristics: Theory and Applications. Kluwer Academic Publishers Norwell, MA, USA.

[Kim et al., 2007] Kim, S. C., Lee, S., and Hahm, J. (2007). Push-pull deterministic search-based dag scheduling for heterogeneous cluster systems. IEEE Transactions On Parallel And Distribuited Systems, 18(11):1489–1502.

[Kumar and Manimaran, 2005] Kumar, G. S. A. and Manimaran, G. (2005). An intra-task DVS algorithm exploiting path probabilities for real-time systems. Real-Time embedded Technology and Applications Symposium (RTAS 05), pages 1–4.

[Lee and Zomaya, 2009a] Lee, Y. C. and Zomaya, A. Y. (2009a). Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In Proc. of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '09, pages 92–99, Washington, DC, USA. IEEE Computer Society.

[Lee and Zomaya, 2009b] Lee, Y. C. and Zomaya, A. Y. (2009b). On effective slack reclamation in task scheduling for energy reduction. JIPS, 5(4):175–186.

[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.

[Liefooghe et al., 2011] Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., and Talbi, E.-G. (2011). On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems.

[Mateusz et al., 2010] Mateusz, G., E., P. J., Bernabe, D., Pascal, B., and U, K. S. (2010). High performance computing and simulation (hpcs), 2010 international conference on. 0:241–248.

[Mezmaz et al., 2010] Mezmaz, M., Lee, Y. C., Melab, N., Talbi, E.-G., and Zomaya, A. (2010). A bi-objective hybrid genetic algorithm to minimize energy consumption and makespan for precedence-constrained applications using dynamic voltage scaling. In Evolutionary Computation (CEC), 2010 IEEE Congress on, pages 1 –8, Barcelona, Spain. IEEE.

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

[Nicholson, 2007] Nicholson, T. A. J. (2007). Optimization in industry: Optimization techniques. Aldine.

[Oja, 2008] Oja, H. (2008). Nonparametric statistics with applications to science and engineering by paul h. kvam, brani vidakovic. International Statistical Review, 76(1):150–151.

[Osyczka, 1984] Osyczka, A. (1984). Multicriterion Optimization in Engineering with FORTRAN Programs. Ellis Horwood Ltd.

[Papadimitriou and Yannakakis, 1988] Papadimitriou, C. and Yannakakis, M. (1988). Towards an architecture-independent analysis of parallel algorithms. In Proceedings of the twentieth annual ACM symposium on Theory of computing, STOC '88, pages 510–513, New York, NY, USA. ACM.

[Pecero et al., 2012] Pecero, J., Huacuja, H., Bouvry, P., Pineda, A., Loces, M., and Barbosa, J. (2012). On the energy optimization for precedence constrained applications using local search algorithms. In High Performance Computing and Simulation (HPCS), 2012 International Conference on, pages 133 –139.

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

[Pecero et al., 2011] Pecero, J. E., Bouvry, P., Fraire Huacuja, H. J., and Khan, S. U. (2011). A multi-objective grasp algorithm for joint optimization of energy consumption and schedule length of precedence-constrained applications. In Cloud and Green Computing (CGC 2011), 2010 IEEE International Conference on, Sydney,Australia. IEEE CS Press. To appear.

[Pedersen and Goldberg, 2004] Pedersen, G. K. M. and Goldberg, D. E. (2004). Dynamic uniform scaling for multiobjective genetic algorithms.

[Pineda et al.,] Pineda, A. A. S., Ángel Ramiro Zúñiga, M., and Huacuja, H. J. F. Algoritmos exactos de calendarización de tareas para programas paralelos en sistemas de procesamiento heterogeneos.

[Polya, 1971] Polya, G. (1971). How to solve it: A new aspect of mathematical method. Princeton University Press Princeton, NJ.

[Ranganathan, 2010] Ranganathan, P. (2010). Recipe for Efficiency: Principles of Power-Aware Computing. Communications of the ACM, pages 1–17.

[Reeves, 1993] Reeves, C. R. (1993). Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc. New York, NY, USA.

[Sinnen, 2007] Sinnen, O. (2007). Task scheduling for parallel systems. Wiley-Interscience.

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

[Srinivas and Deb, 1995] Srinivas, N. and Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms. Evolutionary Computation.

[Ullman, 1975] Ullman, J. D. (1975). Np-complete scheduling problems. J. Comput. Syst. Sci., 10(3):384–393.

[Veldhuizen and Lamont, 2000] Veldhuizen, D. A. V. and Lamont, G. (2000). On measuring multiobjective evolutionary algorithm performance. In In 2000 Congress on Evolutionary Computation, pages 204–211. press.

[Veldhuizen and Lamont, 1998] Veldhuizen, D. A. V. and Lamont, G. B. (1998). Evolutionary computation and convergence to a pareto front. In Stanford University, California, pages 221–228. Morgan Kaufmann.

[Wang et al., 2010] Wang, L., von Laszewski, G., Dayal, J., and Wang, F. (2010). Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on, pages 368 –377.

[Wu et al., 2001] Wu, M.-Y., Shu, W., and Gu, J. (2001). Efficient local search for dag scheduling. IEEE Transactions On Parallel And Distribuited Systems, 12(6):617–627.

[Yang and Ding, 2007] Yang, Q. and Ding, S. (2007). Novel algorithm to calculate hypervolume indicator of pareto approximation set. CoRR, pages –1–1.

[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. In in Proceedings of the Congress on Evolutionary Computation (CEC, pages 3234–3241. IEEE Press.

[Zitzler et al., 2002] Zitzler, E., Laumanns, M., Thiele, L., Fonseca, C. M., and da Fonseca, V. G. (2002). Why quality assessment of multiobjective optimizers is difficult. In Langdon, W. B., Cantú-Paz, E., Mathias, K. E., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. K., and Jonoska, N., editors, GECCO, pages 666–674. Morgan Kaufmann.