Instituto Tecnológico de Ciudad Madero

División de Estudios de Posgrado e Investigación

TESIS

Método de procesamiento celular aplicado al problema de calendarización de tareas en sistemas de procesamiento paralelo

PARA OBTENER EL TÍTULO DE:

Maestro en Ciencias de la Computación

PRESENTA:

I.S.T.I. Miguel Ángel Ramiro Zúñiga

DIRECTOR DE TESIS:

Dr. Héctor Joaquín Fraire Huacuja

CO-DIRECTOR DE TESIS:

Dra. Guadalupe Castilla Valdez

Cd. Madero, Tamaulipas, México Mayo de 2014

"2014, Año de Octavio Paz"

Cd. Madero, Tamps; a 30 de Abril de 2014.

OFICIO No.: U5.078/14
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. MIGUEL ÁNGEL RAMIRO ZÚÑIGA
NO. DE CONTROL G12072008
P R E S E N T E

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

PRESIDENTE :	DRA. LAURA CRUZ REYES
SECRETARIO :	DR. ARTURO HERNÁNDEZ RAMÍREZ
VOCAL :	DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA
SUPLENTE	DRA. CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
DIRECTOR DE TESIS :	DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA
CO-DIRECTORA DE TESIS:	DRA. GUADALUPE CASTILLA VALDÉZ

Se acordó autorizar la impresión de su tesis titulada:

"MÉTODO DE PROCESAMIENTO CELULAR APLICADO AL PROBLEMA DE CALENDARIZACIÓN 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. MARÍA YOLANDA CHÁVEZ CINCO
JEFA DE LA DIVISIÓN

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

c.c.p.- Archivo
Minuta
MYCHC 'NLCO ' jar

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.

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

(*I.S.T.I. Miguel Ángel Ramiro Zúñiga*)

Agradecimientos

Quiero agradecer de todo corazón a mis padres, pues su apoyo incondicional en todas mis empresas ha sido invaluable a lo largo de mi vida, y esta no fue la excepción. Gracias por creer en mí.

A mi asesor de tesis, el Dr. Héctor Joaquín Fraire Huacuja, cuya visión y atinada dirección hicieron posibles este trabajo.

Al resto de mi comité tutorial: el Dr. Arturo Hernández Ramírez y las Dras. Guadalupe Castilla Valdez, Laura Cruz Reyes y Claudia Guadalupe Gómez Santillán; por sus contribuciones, revisiones y observaciones durante el desarrollo de este proyecto.

Al Dr. Jesús David Terán Villanueva, que nunca se cansó de alentarme a emprender esta tarea y que, en compañía del M.C. Aurelio Alejandro Santiago Pineda, siempre estuvieron prestos a brindarme apoyo cuando así lo requería.

A mis compañeros de generación: Oscar Manuel y Andrés, por haber compartido las angustias y alegrías vividas a lo largo de estos dos años, así como a mis compañeros tesistas que me brindaron su amistad durante este trayecto.

Al Instituto Tecnológico de Ciudad Madero, mi nueva Alma Mater, así como al Consejo Nacional de Ciencia y Tecnología, por haberme brindado la oportunidad y los recursos económicos necesarios para llevar a cabo mis estudios de maestría.

A todos mis amigos, por su apoyo y comprensión en este largo periodo, pero sobretodo, por darme ese ocasional y tan necesario espacio apartado de la academia.

Y por último, pero no menos importante, a Gabriela Franco, por todos estos días de aliento, cariño, comprensión y compañía. Gracias a ella, yo soy.

In memoriam de Tomás Ramiro Barrios.

Índice general

Capítulo 1

Introducción

La cada vez más creciente demanda de procesamiento computacional en la sociedad actual ha llevado a un requerimiento cada vez mayor de cantidad de procesamiento. Una de las soluciones más comunes a esta necesidad de cómputo intensivo es el uso de sistemas de cómputo distribuido, muchas veces compuestos por recursos heterogéneos. Sin embargo, el alto consumo energético de estos sistemas se ha vuelto un problema en términos económicos y ambientales.

El problema de calendarización de tareas consiste en minimizar el tiempo de ejecución y el consumo de energía de un programa paralelo ejecutado sobre un sistema de procesamiento distribuido heterogéneo. El tiempo de ejecución (*makespan*) de un programa paralelo tiene una relación inversamente proporcional con la energía que este consume. Esto se debe a que la reducción de energía se logra reduciendo el voltaje de operación suministrado a los procesadores, causando un decremento en el rendimiento de los mismos y un consecuente incremento en el tiempo de ejecución del programa. Bajo este principio opera la técnica denominada DVS (Dynamic Voltage Scaling) que permite a los procesadores ajustar dinamicamente el nivel de voltaje que les es suminstrado, con el objetivo de reducir su consumo energético a cambio de un pérdida de rendimiento.

1.1. Descripción del problema

1.1.1. Modelo de aplicación

El problema objeto de este estudio se presenta en un sistema de procesamiento paralelo heterogéneo, el cual consiste en un conjunto máquinas (o procesadores) M, completamente interconectadas entre si mediante enlaces bidireccionales. Cada procesador $m_j \in M$ tiene diferente capacidad de procesamiento en términos de MIPS (Millones de Instrucciones por Segundo). Asimismo, cada máquina $m_j \in M$ es «DVS-enabled», es decir, puede operar en un conjunto de pares DVS_j de voltajes y velocidad relativa. Cada par $sp_k \in DVS_j$ está conformado por un voltaje de operación v_k y una correspondiente velocidad relativa rs_k, esto determina el rendimiento obtenido por el

procesador m_j cuando es provisto con el voltaje v_k (ver Tabla 1.1).

Tabla 1.1: Configuración de los procesadores (pares DVS)

	Máquina 0		Máquina 1		Máquina 2	
Nivel	Voltaje v	Velocidad rs	Voltaje v	Velocidad rs	Voltaje v	Velocidad rs
0	1.75	1	1.50	1	2.20	1
1	1.40	0.80	1.40	0.90	1.90	0.85
2	1.20	0.60	1.30	0.80	1.60	0.65
3	0.90	0.40	1.20	0.70	1.30	0.50
4			1.10	0.60	1.00	0.35
5			1.00	0.50		
6			0.90	0.40		

Para simplificar el modelo sin pérdida de generalidad se asume lo siguiente: [15] [23]

- Cuando un procesador esta en inactividad, opera a su mínimo nivel de voltaje.
- Cada máquina tiene una conexión directa a cada una de las máquinas restantes.
- La comunicación entre dos máquinas cualesquiera 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.
- El costo de comunicación entre tareas dentro de la misma máquina es despreciado.
- El costo de pasar de un nivel de voltaje a otro en un procesador es despreciado.

Los últimos puntos corresponden al modelo delay [28]. Modelo el cual simplifica el problema como se menciona y sigue siendo utilizado en [23].

1.1.2. Modelo del programa paralelo

Un programa paralelo está representado por medio de una grafo dirigido acíclico (DAG) $G = (T, E)$, donde T es un conjunto finito de nodos y E es un conjunto finito de aristas (ver figura 1.1). Cada nodo $t_i \in T$ está asociado a una tarea del programa paralelo. Se considera estas tareas como unidades indivisibles de trabajo, es decir, que cada tarea debe ser procesada completamente en un solo procesador [8].

Para cada tarea t_i existe un valor asociado p_{ij} que representa el costo, en unidades de tiempo, de computar la tarea t_i en un procesador dado m_j corriendo a su máxima

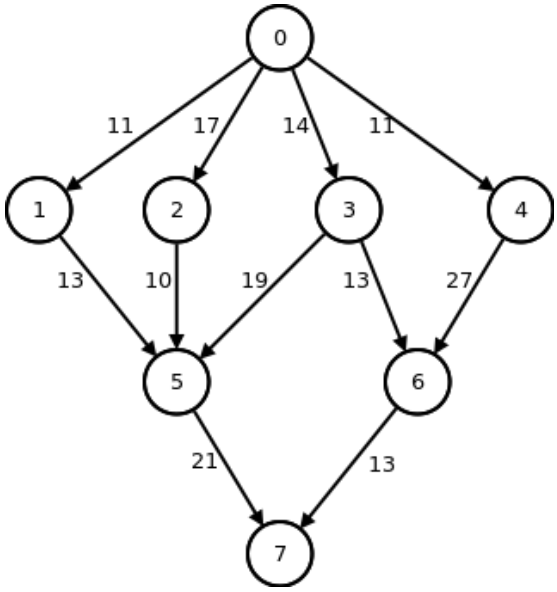

Figura 1.1: Grafo G de precedencia de tareas con costos de comunicación $c_{i,j}$

velocidad y voltaje (ver Tabla 1.2). Un costo de computación relativo p'_{ijk} para un tarea dada t_k, asignada al procesador m_j a una velocidad relativa rs_k puede ser definido como [32]:

$$p'_{ijk} = \frac{p_{ij}}{rs_k} \tag{1.1}$$

Tabla 1.2: Costos computacionales de las tareas p_{ij}

Tarea	m_0	m_1	m_2
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

Asimismo, cada arista $(t_i, t_j) \in E$ representa la restricción de precedencia de t_i sobre t_j, tal que t_j no puede ser procesada hasta que t_i finalice su ejecución. En otras palabras, para que una tarea inicie su procesamiento es necesario que todas sus tareas precedentes hayan concluido el suyo. Adicionalmente, el peso de cada arista $(t_i, t_j) \in E$ representa el costo c_{ij} de comunicar la información producida por la tarea t_i a la tarea t_j; en unidades de tiempo. Se incurre en este costo de comunicación si y sólo si t_i y t_j están asignadas a procesadores diferentes, de lo contrario $c_{ij} = 0$.

El C_{max}, o *makespan* (tiempo total de ejecución) del programa paralelo está dado por $tTermina_i$, que es el tiempo en que termina la tarea i.

$$C_{max} = MAX(tTermina_i), para \quad i = 0, ..., n. \tag{1.2}$$

Donde $n = |T|$.

1.1.3. 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 [23]. La disipación dinámica de energía en este modelo esta dado por:

$$P = ACV^2 f \tag{1.3}$$

Donde A es el factor de actividad, C es la carga total de capacitancia, V es el voltaje suministrado, y f es la frecuencia de operación.

El consumo directo de energía de un programa paralelo en este trabajo esta dado por la siguiente ecuación:

$$E_c = \sum_{i=1}^{|T|} ACv_i^2 f \cdot p_{ij}^* = \sum_{i=0}^{n} Kv_i^2 \cdot p_{ij}^* \tag{1.4}$$

Donde v_i es el voltaje suministrado al procesador en el cual la tarea t_i es ejecutada, p_{ij}^* es el costo de ejecución de la tarea t_i con la configuración de voltaje y máquina programada j y $K = ACf$.

Para simplificación del modelo y dado que no se tiene control sobre las variables que conforman la constante K, esta se desprecia, resultando la siguiente formulación para el cálculo de la energía: [31]

$$E_c = \sum_{i=1}^{|T|} v_{jk}^2 p_{ijk}^* \tag{1.5}$$

Donde v_{jk} es el voltaje suministrado al procesador m_j, y p_{ijk}^* es el costo de procesar la tarea t_i en m_j, corriendo este a la velocidad relativa rs_k asociada a dicho voltaje (ver ecuación 1.1).

Sin embargo, este modelo también considera la energía consumida durante los tiempos ociosos, esto es, el tiempo durante el cual un procesador m_j no está procesando tarea alguna. Durante el tiempo ocioso del procesador m_j se asume que m_j es automáticamente escalado al voltaje más bajo posible de su conjunto de pares DVS_j. Esto es definido como:

$$E_i = \sum_{j=1}^{m} \sum_{d_{jk} \in D_j} v_{min,j}^2 I_{jk} \tag{1.6}$$

Donde D_j es el conjunto de espacios de tiempo ocioso en el procesador m_j, $v_{min,j}$ es el voltaje más bajo disponible para m_j, y I_{jk} es la cantidad de tiempo ocioso para

el espacio de tiempo d_{jk}. Así, el consumo total de energía del programa está dado por [30]:

$$E_t = E_c + E_i \tag{1.7}$$

1.1.4. Problema de calendarización de tareas

Dado un programa paralelo $G = (T, E)$, el problema de calendarización de tareas consiste en distribuir el conjunto T de t tareas en un conjunto M de m procesadores DVS heterogéneos de tal manera que las restricciones de precedencia de las tareas, definidas por las aristas E, sean respetadas, y que el tiempo de respuesta C_{max} del programa y el consumo energético total del sistema E_t sea minimizado.

1.1.5. Instancia del problema

Una instancia del problema esta conformada por un grafo dirigido acíclico de precedencia de tareas $G = (T, E)$ (ver figura 1.1), una matriz de costos de ejecución de dichas tareas a máximo voltaje P en cada máquina $m_j \in M$ (ver Tabla 1.2), así como las configuraciones de pares DVS disponibles para cada máquina (Tabla 1.1).

La figura 1.1 y las Tablas 1.2 y 1.1 constituyen la instancia denominada *sample_3_8*, la cual se usa a manera de ejemplo en las secciones siguientes.

1.1.6. Solución candidata

Una solución candidata del problema esta formada por un orden de ejecución de tareas O sin violaciones de precedencia (véase Tabla 1.3), y una configuración de parejas *máquina/DVS* para cada tarea del grafo, tal como se muestra en la 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 *maquina/DVS* para cada tarea

Tarea	0	1	2	3	4	5	6	7
Máquina	0	2	1	0	0	2	0	2
Nivel de par DVS	1	4	2	0	0	0	0	0

1.1.7. Cálculo del Valor Objetivo *Makespan*

Para el cálculo del *makespan* es necesario calcular los costos computacionales relativos de cada tarea p'_{ij} (Tabla 1.5), los cuales se calculan con la ecuación 1.1, llevar un registro de los tiempos de terminación de la última tarea ejecutada en cada máquina

$TAEm_j$ (tiempo actual de ejecución en máquina j), así como de los costos de comunicación ajustados a la configuración actual c'_{ij}. Como se mencionó anteriormente, se incurre en estos costos sólo si las tareas en cuestión se ejecutan en diferentes máquinas, de tal manera que:

$$c'_{ij} = \begin{cases} c_{ij} & \text{si } t_i \text{ y } t_j \text{ se ejecutan en diferente máquina} \\ 0 & \text{si } t_i \text{ y } t_j \text{ se ejecutan en la misma máquina} \end{cases}$$

Tabla 1.5: Costo computacional con velocidad relativa p'_{ij}

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

A continuación se presenta el algoritmo del cálculo del valor objetivo *makespan*: Algoritmo 1 [33].

Algoritmo 1 Cálculo del valor 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 ajustados c'_{ij}.

Ensure: O no viola restricciones de precedencia de tareas.

for $x = 0$ to n **do**
  $t_{actual} = o_x$
  if $((u, actual) \in E) = null$ **then**
    $tEmpieza_{actual} = TAEm_j$
    $tTermina_{actual} = tEmpieza_{actual} + p'_{actual,j}$
  else
    $u^* = argmax_{u \in T | (u,actual) \in E} \{tTermina_u + c'_{u,actual}\}$

    $tEmpieza_{actual} = \text{MAX}(TAEm_j, tTermina_{u^*} + c'_{u^*,actual})$
    $tTermina_{actual} = tEmpieza_{actual} + p'_{actual,j}$
  end if
end for
$C_{max} = \text{MAX}(tTermina_x)$ for $x = 0, ..., n$

Dada la instancia *sample_3_8* (definida en la sección 1.1.5) y la solución candidata definida en la sección 1.1.6, la Tabla 1.5 muestra los valores de p'_{ij}, y la Tabla 1.6 muestra la ejecución del Algoritmo 1. Donde $TAEm_j$ es el tiempo en el que ejecutó su última tarea la máquina m_j, la columna *Empieza* es el tiempo en el que la tarea t_i está lista para ser ejecutada y 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

1.1.8. Cálculo del Valor Objetivo Energía

Para calcular la energía que requeriría una solución candidata es necesario conocer el valor del *makespan* C_{max}, los costos computacionales relativos p'_{ij}, el voltaje v_k asociado al par *DVS* seleccionado en cada tarea t_i con el procesador m_j asignado a esta, y el voltaje más bajo v_{min} para cada m_j. Véase Algoritmo 2 [33].

Algoritmo 2 Cálculo del valor objetivo Energía

Require: Tiempo total de ejecución C_{max}, costos computacionales relativos p'_{ij}, configuración *máquina/DVS* para cada tarea, voltaje mínimo de cada procesador v_{min}

```
energia = 0
for all t_i do
    energia+ = v_k^2 · P'ij
end for
for all m_j do
    idle = C_max
    for t_i ejecutadas en m_j do
        idle− = P'_{t_ij}
    end for
    energia+ = idle · v_min^2
end for
```

Dada la instancia *sample_3_8* y la solución candidata definida en la sección 1.1.6, la Tabla 1.7 muestra la ejecución del Algoritmo 2. En la primera columna se 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 los tiempos ociosos *idle* para cada maquina. La energía total es la suma de la energía directa más la energía indirecta, de acuerdo a la ecuación 1.5.

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. Objetivos

A continuación se presentan los objetivos definidos al inicio de este proyecto de investigación.

1.2.1. Objetivo general

Desarrollar un método de solución competitivo del problema de calendarización de tareas en sistemas de procesamiento paralelo basado en NSGAII y procesamiento celular.

1.2.2. Objetivos específicos

- Analizar los procesos críticos del algoritmo NSGAII.
- Diseñar las celdas del algoritmo de procesamiento celular.
- Diseñar las poblaciones.
- Diseñar el método de comunicación.

- Diseñar el método de detección del estancamiento.
- Implementar el método de solución utilizando el procesamiento celular.
- Evaluación experimental del método de solución propuesto.

1.3. Alcances y limitaciones

Las instancias que se emplearán en la evaluación experimental son un subconjunto de las presentadas en [14] y posteriormente usadas en [33].

Asímismo, los resultados de dicha experimentación se compararán contra los resultados de una implementación de NSGA2 para el problema en cuestión (sin procesamiento celular), tal como se describe en [33].

Para verificar la calidad de los frentes de Pareto arrojados por el algoritmo propuesto se prevee el uso de indicadores de calidad multiobjetivo unarios. Concretamente, para fines comparativos, este trabajo se enfoca en análisis de resultados usando los indicadores *hypervolume*, *generational distance* y *spread*

Capítulo 2

Marco teórico

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 [38].

Clase P

Es el conjunto de todos los problemas de decisión que pueden ser resueltos en tiempo polinomial por un algoritmo determinista [38].

Clase NP

La clase NP es el conjunto de todos los problemas de decisión para los que existe tanto un algoritmo no determinista de tiempo polinomial que genere una solución candidata del mismo, así como un algoritmo determinista de tiempo polinomial que decida dicha solución candidata [38].

Problema NP-Completo

Se dice que un problema B es o pertenece a la clase NP-Completo si satisface las siguientes dos condiciones [38]:

1. $B \in NP$
2. Todo problema $A \in NP$ es reducible en tiempo polinomial a B.

Problema NP-Duro

Un problema NP-Duro es un problema de optimización cuya versión de decisión es NP-Completo [12].

Problema de Optimización

En los problemas llamados problemas de optimización se busca la mejor solución posible entre un conjunto de posibles soluciones. Cuando un problema de optimización es NP-Duro, no existe un algoritmo de tiempo polinomial que encuentre la solución óptima de este, a menos que que $P = NP$ [38].

2.2. Métodos de Optimización

Métodos Exactos

Los algoritmos exactos resuelven problemas que pertenecen a la clase P de forma óptima y en tiempo aceptable (tiempo polinomial). Sin embargo, no pueden resolver problemas de la clase NP en tiempo polinomial; es decir, aunque exista un algoritmo que encuentra la solución exacta, este tomaría tanto tiempo en encontrarla que lo hace completamente inaplicable [1].

Ejemplos de este tipo de métodos son los algoritmos enumerativos exhaustivos, algoritmos de ramificación y acotamiento, algoritmos de plano de corte, etc.

Métodos Heurísticos

Las heurísticas utilizan reglas, estrategias o programas basados generalmente en conocimientos empíricos para guiar el conocimiento. La palabra heurística procede del griego y significa "hallar, inventar"[34]. Nicholson y Reeves proponen 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 [26].
- Una técnica que busca buenas soluciones con un tiempo de computación razonable sin garantizar la optimalidad [36].

Métodos Meta-heurísticos

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

- Los procedimientos meta-heurísticos 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. [9]
- Una meta-heurística 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. [19]

2.3. Optimización multiobjetivo

Un problema de optimización consiste en minimizar (o maximizar) uno o más funciones objetivo. Cuando el problema involucra un solo objetivo, el proceso de encontrar una solución óptima se llama optimización de un solo objetivo. Por otro lado, cuando el problema involucra más de un objetivo, entonces a este se le llama optimización multiobjetivo.

Formalmente, un problema de optimización multiobjetivo se puede definir como [27]: "...un vector de variables de decisión que satisface restricciones y optimiza una función vector cuyos elementos representan las funciones objetivos. Estas funciones forman una descripción matemática de criterios de desempeño, los cuales usualmente están en conflicto unos con otros. Luego, el término 'optimizar' significa encontrar una solución que de a todas las funciones objetivo valores aceptables por el Tomador de Decisiones."

En el lenguaje de la administración estos problemas son conocidos como toma de decisiones multi-criterio (multiple criterion decision-making, MCDM). Diferentes soluciones pueden producir escenarios conflictivos a través de diferentes objetivos. Para dos objetivos en conflicto cada objetivo corresponde a una diferente solución óptima [5].

2.3.1. Óptimo de Pareto

Generalizada por Vilfredo Pareto a finales del siglo XIX, la noción de optimalidad de Pareto es actualmente una de las más usadas en problemas de optimización multiobjetivo. Consiste en lo siguiente:

Se dice que un vector $\vec{x}^* \in \Omega$ es un óptimo de Pareto si para todo $\vec{x} \in \Omega e I = \{1, 2, 3, ..., k\}$ ya sea,

$$\forall i \in I f_i((\vec{x}) = f_i(\vec{x}^*)),$$

o existe al menos una $i \in I$ tal que

$$f_i(\vec{x}) > f_i(\vec{x}^*)$$

En otras palabras, se dice que $\vec{x}^*$ es un óptimo de Pareto si no existe un vector factible $\vec{x}$ que mejore un criterio del objetivo sin causar un empeoramiento en al menos otro criterio. [4]

2.3.2. 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 sólo si u es parcialmente menor que v. Ejemplo: $\forall i \in \{1, ..., k\}$, $u_i \leq$

$v_i \wedge \exists i \in \{1, ..., k\} : u_i < v_i$.

Es decir, para que una solución domine a otra, ésta debe ser necesariamente mejor en al menos un objetivo, y no peor en ninguno de ellos.

2.3.3. Conjunto de Óptimos de Pareto

Para un problema multiobjetivo dado $\vec{f}(x)$, el conjunto de óptimos de Pareto P^* se define como:

$$P^* := \{x \in \Omega | \neg \exists x^{'} \in \Omega \vec{f}(x^{'}) \leq \vec{f}(x)\}$$

2.3.4. 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:

$$PF^* := \{\vec{u} = \vec{f} = (f_1(x), ..., f_k(x)) | x \in P^*\}$$

Visto de otra manera, el frente de Pareto está conformado por el conjunto de óptimos de Pareto que no están dominados.

2.3.5. Conjunto de soluciones no dominadas

Un conjunto de vectores solución A es un conjunto de soluciones no dominadas si y sólo si:

$$\forall \vec{a} \in A \neg \exists \vec{a}^{'} \in A | \vec{a}^{'} \prec \vec{a}$$

2.4. Algoritmos genéticos

Los algoritmos genéticos son algoritmos de optimización estocásticos y poblacionales, cuyos métodos de búsqueda están inspirados en el proceso de selección natural y la herencia genética [24] [2].

El Algoritmo 3 describe la estructura general de un algoritmo genético [2].

Algoritmo 3 Algoritmo genético simple

1: Inicialización.
▷ Generar población inicial P aleatoriamente o con conocimiento previo.
2: Evaluación.
▷ Evaluar la aptitud de todos los individuos en P.
3: Selección.
▷ Seleccionar un conjunto de candidatos prometedores S de P.
4: Cruza.
▷ Aplicar cruzamiento al conjunto de parejas S para generar un conjunto de descendencia D.
5: Mutación.
▷ Aplicar mutación al conjunto de descendencia D para obtener su conjunto perturbado D'
6: Reemplazo.
▷ Reemplazar la población actual P con el conjunto de descendencia D'.
7: Terminación.
▷ Si el criterio de salida no se ha cumplido, ir al paso 2.

Los algoritmos genéticos usan un vocabulario tomado de la genética natural. Se habla de individuos (o genotipos, cromosomas) en una población. Asimismo, una algoritmo genético para un problema particular debe tener los siguientes cinco componentes [24]:

- Una representación para las soluciones candidatas al problema.
- Una forma de crear una población inicial de soluciones candidatas.
- Una función de evaluación que funge el papel del entorno, clasificando soluciones en términos de su "aptitud".
- Operadores genéticos que alteren la composición de los hijos.
- Valores para varios parámetros que el algoritmo genético usa (tamaño de la población, probabilidades de aplicar algoritmos genéticos, etc.)

Los principales operadores que los algoritmos genéticos usan son [1]:.

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

2.4.1. NSGAII

Originalmente propuesto en [6], el *Fast Elitist Non-Dominated Sorting Genetic Algorithm (NSGAII)*, es un algoritmo genético multiobjetivo de uso general.

Se caracteriza por el uso de un método de ordenamiento rápido con base en la construcción de múltiples frentes de Pareto y un método para asegurar la diversidad de soluciones basado en la posición de estas dentro de cada frente. El elitismo del método está dado por la conservación constante de las soluciones localizadas en el primer frente generado durante el ordenamiento, siendo estas aquellas que no están dominadas.

Una descripción a detalle de este algoritmo se presenta en la sección 4.2

Capítulo 3

Estado del arte

3.0.2. Métodos de solución

El problema de calendarización de tareas (en su versión mono-objetivo) a sido abordado en diversas ocasiones desde su aparición [16] [20] [18], siendo las últimas propuestas de solución las presentadas en [21] y [14].

Sin embargo, dada la relativa recencia de la versión actual del problema (la descrita en este trabajo), que incluye los niveles de voltaje de cada máquina, el cálculo de la energía, la energía consumida en tiempos muertos (idle) y el enfoque multiobjetivo, esta ha sido abordada en muy pocas ocasiones. Aquí se presentan las tres más importantes.

En [22], una continuación del trabajo presentado en [21], se propone un algoritmo (ECS+idle) de list-scheduling, es decir, que parte de un ordenamiento topológico generado a partir de una heurística [31], en este caso, la de ordenamiento por $B-levels$. Posteriormente, se hace uso de una función objetivo denominada $RelativeSuperiorityRS$, que es una razón entre el *makespan* y la energía de una solución. Así, el problema se reduce a minimizar RS. Para esto se hace uso de un método de búsqueda local llamado MCER, que prueba combinaciones de máquina/voltaje, buscando una que minimice RS sin incrementar el *makespan*.

Por otro lado, en [32] se propone una metaheurística GRASP multiobjetivo que incorpora el concepto de optimalidad de Pareto. Este método está compuesto de dos fases: una construcción voraz de un orden topológico a voltajes máximos y una búsqueda local dentro del vecindario de la solución geradada para construir el conjunto de soluciones no dominadas. El método de construcción de soluciones hace uso de un lista de prioridades de tareas generada a partir de la heurística de $B - levels$, y genera buenas soluciones factibles a voltajes máximos. En la fase de búsqueda local se escalan los voltajes para reducir la energía, actualizando el frente de Pareto con las nuevas soluciones no dominadas.

Por último, en [33] se propone un método de solución basado en el algoritmo genético multiobjetivo NSGA2 ([6]), así como una versión memética del mismo que añade una búsqueda local. Ambos métodos siguen un enfoque multiobjetivo de Frente de Pareto y parten de una población inicial aleatoria, misma que va evolucionando mediante

operadores de cruza y mutación (búsqueda local en el caso del memético), en cada generación se da preferencia a las soluciones no dominadas, así como a las que aporten mayor diversidad en cada generación. Esto último se logra mediante un método que ordena las soluciones en función de su no dominancia y otro más que asigna a cada una una medida de diversidad basada en su posición dentro del Frente de Pareto (ver sección 4.2). La mutación se basa en la asignación de una tarea a una máquina y un voltaje aleatorios; la búsqueda local es similar, pero en esta se busca hasta alcanzar un máximo número de ciclos sin mejora, usando un criterio *BEST*.

3.0.3. Algoritmos de procesamiento celular

Se han propuesto diversos algoritmos para la solución de problemas de alta complejidad basados en el concepto de procesamiento celular, como lo son los algoritmos genéticos celulares [10], o los algoritmos co-evolutivos [44]. Sin embargo, estos se tratan de algoritmos específicos, limitados en muchos aspectos por su misma estructura.

Con el objetivo de traer flexibilidad y generalización a la idea de procesamiento celular, en [40] se propone en nuevo tipo de algoritmos de procesamiento celular, diferente a los ya existentes. Este nuevo enfoque se caracteriza por el uso de múltiples células de procesamiento para explorar diferentes regiones el espacio de búsqueda, así como de la posibilidad de usar diversas heurísticas o metaheurísticas para cada célula de procesamiento.

Los componentes principales de un algoritmo de procesamiento celular son: las celdas de procesamiento, la comunicación entre celdas y los métodos de estancamiento local y global [40]. El algoritmo simula la ejecución paralela de varias celdas de procesamiento independientes situadas en diversos puntos del espacio de búsqueda. Cada celda de procesamiento implementa un proceso de búsqueda heurístico o metaheurístico hasta llegar a un punto de estancamiento, determinado por un criterio particular. El algoritmo principal continua su ejecución hasta que todas las celdas se han estancado. Las celdas pueden intercambiar información entre si en diversos puntos, ya sea durante su ejecución, o al terminar esta.

Se reporta que los algoritmos de procesamiento celular de este tipo obtienen mejoras de hasta un 3.6 % en calidad de la solución y hasta un 20 % de reducción de tiempo de ejecución comparados con el enfoque monolítico de solución, es decir, una metaheurística sin procesamiento celular[40].

Capítulo 4

Propuesta de solución

El método de solución propuesto sigue un enfoque semi-independiente: se parte del cálculo de un orden de ejecución de tareas factible, el cual se fija y no se altera en ningún momento, limitando así el espacio de búsqueda al problema de asignar tareas a procesadores y escalarlos a diferentes niveles de voltaje. Posteriormente, se usa un algoritmo de procesamiento celular para encontrar un conjunto de óptimos de Pareto de asignaciones de procesadores-voltajes de tal manera que tanto el *makespan* como la energía consumida sean optimizados en igual proporción. El orden de ejecución de tareas es calculado usando el algoritmo de Ordenamiento por *B-Levels* descrito en [32]. Dicho algoritmo fue seleccionado por su simplicidad, velocidad y resultados cercanos al óptimo.

4.1. Algoritmo de procesamiento celular

El enfoque de procesamiento celular utilizado en este trabajo es el propuesto en [40], el cual se caracteriza por el uso de un conjunto de celdas o células de procesamiento que se ejecutan de manera pseudo-paralela hasta llegar a un estancamiento: situación en la que una celda deja de producir buenas soluciones. Pudiendo compartir información entre si mientras se ejecutan (comunicación en línea) o al final de todo el proceso (comunicación fuera de línea).

Cada celda de procesamiento en un algoritmo de procesamiento celular está compuesta por una instancia de una heurística o meta-heurística particular. Si todas las celdas del algoritmo implementan la misma heurística se habla de celdas homogéneas, de lo contrario, estas son heterogéneas. La ejecución pseudo-paralela de las celdas permite explorar diferentes regiones del espacio de solución. Asimismo, la verificación constante de las condiciones de estancamiento evita gastar tiempo en labores innecesarias. [40]

El método de solución propuesto en este trabajo consiste en un algoritmo de procesamiento celular homogéneo (Algoritmo 15.) en el que cada celda implementa una versión del algoritmo genético multiobjetivo NSGAII.

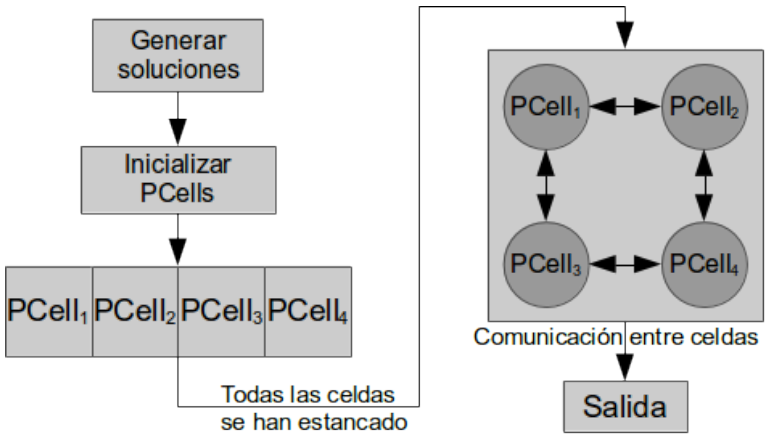

Figura 4.1: Estructura de un algoritmo celular con comunicación fuera de línea

4.2. El algoritmo NSGAII

Originalmente propuesto en [6], el *Fast Elitist Non-Dominated Sorting Genetic Algorithm II* (NSGAII), sucesor del algoritmo NSGA [39], es un algoritmo genético multiobjetivo de uso general, fundamentado en el concepto de optimalidad de Pareto.

Se caracteriza por el uso de un método de ordenamiento rápido con base en la construcción de múltiples frentes de Pareto (*Fast Non-Dominated Sort*) y un método para asegurar la diversidad de las soluciones basado en la posición de estas dentro de cada frente (*Crowding Distance Assignment*). Ambos definidos en [6]. El elitismo del algoritmo está dado por la conservación constante de las soluciones localizadas en el primer frente generado durante cada ordenamiento, siendo este el conjunto de soluciones no dominadas de la población actual.

Se eligió el NSGAII como base para las celdas del algoritmo de procesamiento celular debido a su uso previo para la solución del problema en cuestión en [33] y los buenos resultados que se presentan en el mismo al resolver el problema usando un enfoque de solución multiobjetivo.

4.2.1. Fast Non-dominated Sort

El algoritmo de ordenamiento basado en la *no-dominancia* de Pareto *Fast Non-dominated Sort* [6] es uno de los componentes esenciales del NSGAII. Toma por entrada un conjunto de soluciones P y las ordena en diversos grupos basándose en su nivel de dominancia F, cada grupo constituye un frente $F_i \in F$. El primer frente contiene las soluciones que no son dominadas por ninguna otra, el segundo las que son dominadas únicamente por las soluciones del primero, el tercero las que son dominadas por las soluciones del primero y el segundo, y así sucesivamente. Al dar preferencia a las soluciones localizadas en los primeros frentes durante las operaciones genéticas se provee de elitismo al algoritmo.

El *Fast Non-dominated Sort* (Algoritmo 4) toma como entrada un conjunto de

soluciones P. El ciclo de las Líneas 1-2 comprueba, para cada una de las soluciones $p \in P$, si existe otra $q \in P$ tal que $p \prec q$, de ser así, la Línea 4 agrega la solución q al subconjunto de soluciones dominadas por p, llamado S_p; en caso contrario, si $q \prec p$ se aumenta el contador de soluciones que dominan a p, llamado n_p. Si no se encontró ninguna solución q que domine a p ($n_p = 0$) entonces p pertenece al primer frente F_1, mismo al que se agrega en la Línea 12.

Después de conformar el primer frente F_1, el algoritmo continua formando el resto de los existentes. Se comienza con el primero frente F_1 (Línea 15) y en la Línea 16 se entra a un ciclo que recorre todos los frentes formados. Se inicializa el frente temporal H en vacío y en las Lineas 18-19 se comparan, para cada una de las soluciones $p \in F_i$, las soluciones que dominan a $q \in S_p$, por cada una de ellas se resta una unidad al contador de soluciones que dominan a q: n_q. Cuando $n_q = 0$, se agrega q al frente temporal H (Líneas 21-23). Al finalizar el ciclo de las Líneas 18-19 se actualiza $i \leftarrow i + 1$ 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 $F_i \in F$.

Tabla 4.1: Población desordenada

ID	Obj. 1	Obj. 2
1	13	25
2	10	18
3	15	20
4	18	17
5	9	23
6	12	18
7	14	23
8	8	15
9	12	15
10	8	19

Tabla 4.2: Población ordenada

Frente	ID	Obj. 1	Obj. 2
F_1	8	8	15
F_2	2	10	18
	9	12	15
	10	8	19
F_3	4	18	17
	6	12	18
	5	9	23
F_4	3	15	20
	1	13	25
	7	14	23

Las Tablas 4.1 y 4.2 ejemplifican el ordenamiento que realiza el Algoritmo 4 con una población de 10 soluciones, para un problema de minimización de dos objetivos. La Tabla 4.1 muestra la población antes de ser ordenada y la Tabla 4.2 la misma población después de ser ordenada por el algoritmo en cuestión. La columna *Frente* de la segunda tabla muestra el frente F_i al que se asignaron cada una de las soluciones.

Algoritmo 4 Fast Non-dominated Sort

Require: Conjunto de soluciones P.

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

La complejidad del Algoritmo 4 es de $O(mN^2)$ [6], donde m es el número de objetivos y N es el número de elementos en el conjunto de soluciones a ordenar. De ahí que, en comparación con su predecesor [39] (de complejidad $O(mN^2)$), este sea considerado como un ordenamiento rápido.

4.2.2. Asiganción de Crowding Distance

La *crowding distance* es un métrica que define la diversidad de una solución dentro de un frente. Se trata de una cantidad escalar referida como $i_{crowdist} \in F_x$ y es un estimado del cuboide más grande con centro en i que no incluye ningún otro miembro $j \in F_x$. [6].

El Algoritmo 5 muestra el cálculo de la *crowding distance*. En [6] se establece que los límites de las distancias (Líneas 7 y 8) deben ser valores infinitos. Sin embargo, en este trabajo se propone una variación del método propuesto en [33]. En dicho trabajo se añade una normalización (Línea 11) con los valores mínimo y máximo por objetivo m: MIN_m y MAX_m. Estos valores son dinámicos y se actualizan en cada nueva población Q_t con los mejores y peores valores objetivo encontrados [33]. Esto se hace a manera de asegurar la optimización de ambos objetivos en igual medida.

El Algoritmo 5 tiene como entrada un conjunto de soluciones I. La Línea 1 asigna a l el número de soluciones en I, en las Líneas 2 y 3 se inicializa la distancia de todos los individuos $i \in I$ con el valor 0. Posteriormente, en el ciclo principal de la Línea 5, se ordenan las soluciones $i \in I$ por cada uno de sus valores objetivos m. En las Líneas 7 y 8 se aumenta la distancia de los individuos en la posición $I[1]$ y $I[l]$ (los límites) en MAX_m. El ciclo de la Línea 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 $(MAX_m - MIN_m)$. Este último paso sólo se efectúa si $MAX_m \neq MIN_m$, es decir, se asegura que exista un rango entre los valores objetivo actuales.

Algoritmo 5 Normalized Crowding Distance Assignment

Require: Conjunto de soluciones I.

1: $l \leftarrow |I|$
2: **for all** $i \in I$ **do**
3: $I[i].crowdist \leftarrow 0$
4: **end for**
5: **for all** objective m **do**
6: $I \leftarrow Ordena(I, m)$
7: $I[1].crowdist \leftarrow I[1].crowdist + MAX_m$
8: $I[l].crowdist \leftarrow I[l].crowdist + MAX_m$
9: **for** $i = 2$ to $l - 1$ **do**
10: **if** $MAX_m \neq MIN_m$ **then**
11: $I[i].crowdist \leftarrow I[i].crowdist + ((I[i+1].m - I[i-1].m)/(MAX_m - MIN_m))$
12: **end if**
13: **end for**
14: **end for**

4.2.3. El operador de comparación *crowded*

El *crowded comparision operator* $(\geq_n)$ [6] es un operador comparativo que se usa para guiar el proceso evolutivo de selección hacia la construcción de un frente de Pareto uniforme y diverso. Hace uso de dos atributos propios de cada individuo i:

- Frente al que pertenece en el ordenamiento *Fast Non-dominated Sort* (i_{rank}) .

- Distancia obtenida por *Crowding Distance Assignment* ($i_{crowdist}$).

Para dos soluciones i y j el operador de comparación *crowded* $\geq_n$ se define como:
$i \geq_n j$ Si ($i_{rank} < j_{rank}$) o (($i_{rank} = j_{rank}$) & ($i_{crowdist} > j_{crowdist}$))

En otras palabras, se da preferencia a las soluciones encontradas en los primeros frentes encontrados por el *Fast Non-dominated Sort*, pero si ambas soluciones pertenecen al mismo frente se prefieren las soluciones con mayor *crowding distance*.

4.2.4. Ciclo principal de NSGAII

El ciclo principal de NSGAII descrito en [6] establece la secuencia que sigue cada generación t del algoritmo. Tratándose de un algoritmo genético, es necesario establecer un tamaño de población, que en este caso debe ser un entero par N. Esto porque la población para cada generación se conforma por la unión de dos conjuntos: el conjunto de soluciones padres P_t y el conjunto de soluciones hijos Q_t.

El Algoritmo 6, en la Línea 1, inicializa t en 1 y en la Línea 2 inicializa las poblaciones P_t y Q_t de la primera generación con individuos generados aleatoriamente. La Línea 4 une las poblaciones P_t y Q_t en R_t y en la Línea 5 el método *Fast Non-dominated Sort* hace uso de R_t para generar el conjunto de frentes encontrados F. La Línea 6 inicializa i en 1 para posicionarse en el primer frente F_i. Mientras $|P_{t+1}| + |F_i| \leq N$ se calcula la distancia a los individuos en F_i con el método *Crowding Distance Assignment* y se agregan a la población de padres P_{t+1}, incrementando i en cada ciclo, para posicionarse en el siguiente frente F_i (Líneas 7-11).

Posteriormente, se verifica si $|P_t| < N/2$ (Línea 12), de ser así, se completa P 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 Assignment* 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 Línea 20 se genera la nueva población de hijos Q_{t+1} a partir de la población de padres P_{t+1} usando un método de selección basado en el *crowded comparision operator* y aplicando operadores genéticos. Una vez formados los hijos, se prosigue a la siguiente generación. El ciclo principal se repite hasta que t alcanza el número máximo de generaciones MAX_GEN.

Algoritmo 6 NSGAII

Require: Población de padres P_t. Población de hijos Q_t

1: $t = 1$
2: Generar aleatoriamente y evaluar los individuos de P_t y Q_t.
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** $(|P_{t+1}| + |F_i|) \leq N/2$ **do**
8: $crowding\_distance\_assignment(F_i)$
9: $P_{t+1} \cup F_i$
10: $i++$
11: **end while**
12: **if** $|P_{t+1}| < N/2$ **then**
13: $crowding\_distance\_assignment(F_i)$
14: $Sort(F_i, distancia)$ ▷ Ordena F_i en orden descendente usando crowding distance
15: **repeat**
16: $x \leftarrow Pop(F_i)$
17: $P_{t+1} = P_{t+1} \cup \{x\}$
18: **until** $|P_{t+1}| = N$
19: **end if**
20: $Q_{t+1} = make\_new\_pop(P_{t+1})$ ▷ Utilizar selección,cruza y mutación para crear una nueva población Q_{t+1}
21: Evaluar nueva población
22: $t++$
23: **until** $t = MAX\_GEN$ **return** P_t

4.3. Celda de procesamiento

Como ya se mencionó anteriormente, cada celda de procesamiento del algoritmo propuesto consiste en una implementación de NSGAII basada en la versión descrita en [33]. El método *Fast Non-Dominated Sort* usado es el descrito en [6], mientras que el *Crowding Distance Assignment* es el propuesto en [33], el cual implementa una normalización de los valores objetivo. Todas las celdas usan la misma versión del NSGAII.

El proceso de inicialización de una celda $PCell_i \in PCells$ es similar al definido en [33] para la inicialización del algoritmo NSGAII principal: Se generan aleatoriamente dos conjuntos de soluciones $PCell_i.P$ y $PCell_i.Q$ (padres e hijos), ambos de igual tamaño. Posteriormente, estos se evaluan. La unión de los mismos conformará la población inicial de la celda.

El Algoritmo 7 describe un ciclo de procesamiento (generación) de una celda $PCell$. Este no difiere mucho de lo que constituye una generación en el algoritmo NSGAII original (ver Algoritmo 6). Se comienza por unir las poblaciones $PCell.P_t$ y $PCell.Q_t$ en R_t (Línea 2) y en la Línea 3 el método *Fast Non-Dominated Sort* hace uso de R_t para generar el conjunto de frentes encontrados F, ordenados de acuerdo a su dominancia. Posteriormente, en las Líneas 6-11 se llena el conjunto de padres $PCell.P_{t+1}$ con las soluciones presentes en F, en orden descendiente, dando preferencia a las soluciones con mayor dominancia sobre el resto (aquellas presentes en los frentes con índice menor). Las últimas soluciones en entrar a $PCell.P_{t+1}$ son las favorecidas por la métrica obtenida mediante el método *Crowding Distance Assignment*, lo que asegura diversidad en el conjunto (Líneas 12-18). Por último, en la Línea 20 se genera la nueva población de hijos $PCell.Q_{t+1}$ a partir de la población de padres $PCell.P_{t+1}$.

Algoritmo 7 Procesamiento de una celda $PCell_i$

Require: Celda a procesar $PCell$

1: $t \leftarrow PCell.cicloActual$
2: $R_t \leftarrow PCell.P_t \cup PCell.Q_t$
3: $N \leftarrow |R_t|$
4: $F \leftarrow FastNondominatedSort(R_t)$ ▷ Donde $F \leftarrow (F_1, F_2, ..., F_n)$: frentes encontrados en R_t
5: $PCell.PF \leftarrow F_1$
6: $i \leftarrow 1$
7: **while** $(|PCell.P_{t+1}| + |F_i|) \leq N/2$ **do**
8: $\quad CrowdingDistanceAssignment(F_i)$
9: $\quad PCell.P_{t+1} \leftarrow PCell.P_{t+1} \cup F_i$
10: $\quad i++$
11: **end while**
12: **if** $|PCell.P_{t+1}| < N/2$ **then**
13: $\quad CrowdingDistanceAssignment(F_i)$
14: $\quad OrdenaDesc(F_i, crowdist)$ ▷ Ordena en orden descendente F_i usando crowding distance
15: $\quad$ **repeat**
16: $\quad\quad x \leftarrow Pop(F_i)$ ▷ Remueve el primer elemento de F_i
17: $\quad\quad PCell.P_{t+1} \leftarrow PCell.P_{t+1} \cup x$
18: $\quad$ **until** $|PCell.P_{t+1}| = N/2$
19: **end if**
20: $PCell.Q_{t+1} \leftarrow NuevaPoblacion(PCell.P_{t+1})$ ▷ Usar selección, cruza y mutación para crear una nueva población Q_{t+1}
21: Evaluar nueva población
22: $PCell.cicloActual \leftarrow t + 1$

4.4. Operadores genéticos

4.4.1. Selección

El método de selección genética de padres se implementa mediante una selección por torneo (Algoritmo 8). Para seleccionar cada padre en una generación dada se toman dos individuos de la población de manera aleatoria y se comparan usando el *crowded comparision operator* $\geq_n$ (ver sección 4.2.3). El ganador del torneo será seleccionado. Se seleccionan $|PCell.P|$ padres. Posteriormente, los elementos seleccionados se tomarán por parejas para participar en el proceso de cruza.

Algoritmo 8 Selección genética por torneo

Require: Población de soluciones P

```
1: S ← ∅        ▷ S es el conjunto de padres seleccionados para participar en la cruza.
2: repeat
3:     i ← Seleccionar aleatoriamente un elemento de P
4:     j ← Seleccionar aleatoriamente un elemento de P (diferente de i).
5:     if i ≥_n j then
6:         Añadir i a S
7:     else
8:         if j ≥_n i then
9:             Añadir j a S
10:        else
11:            Seleccionar aleatoriamente i o j y añadir a S
12:        end if
13:    end if
14: until |S| = |P|
```

4.4.2. Cruza

El algoritmo NSGAII propuesto utiliza una cruza en un punto: se selecciona un número aleatorio entre 0 y $|T|$ (donde T es el conjunto de tareas del programa paralelo), el cual definirá el punto en que se dividirán los padres. El primer hijo se forma tomando del padre 1 los elementos que van desde la posición 0 hasta la posición del punto de cruza y colocándolos en las mismas posiciones dentro del hijo. El resto de los elementos del hijo son tomados de la parte que va de la posición del punto de cruza a la posición final en el padre 2, en el mismo orden en que aparecen. El hijo 2 se forma intercambiando el padre 2 por el 1 y repitiendo la operación previa. Todos los padres se cruzan. Las Tablas 4.3 a 4.6 (tomadas de [33]) ejemplifican el proceso de cruza entre dos soluciones

Tabla 4.3: Padre Uno

Tarea	0	1	2	3	4	5	6	7
Máquina	0	2	2	1	0	1	2	2
Par DVS	1	4	2	0	0	6	0	0

Tabla 4.4: Padre Dos

Tarea	0	1	2	3	4	5	6	7
Máquina	0	2	1	0	0	2	1	2
Par DVS	1	4	6	2	1	0	6	0

Tabla 4.5: Hijo Uno

Tarea	0	1	2	3	4	5	6	7
Máquina	0	2	2	1	0	2	1	2
Par DVS	1	4	2	0	0	0	6	0

Tabla 4.6: Hijo Dos

Tarea	0	1	2	3	4	5	6	7
Máquina	0	2	1	0	0	1	2	2
Par DVS	1	4	6	2	1	6	0	0

4.4.3. Mutación

Se implementaron dos métodos de mutación genética. El primero se trata de la mutación propuesta en [33], referido a partir de ahora como mutación uniforme. El segundo método es el propuesto en este trabajo y se denomina mutación sesgada.

Todos los hijos generados mediante la cruza entran al proceso de mutación.

Mutación uniforme

El primer método de mutación presentado es el propuesto en [33]. En esta mutación (Algoritmo 9), cada elemento (representando una tarea $t_i \in T$) de la solución a mutar tiene una probabilidad β de ser alterado. Esta alteración consiste en la asignación aleatoria de una máquina $m_j \in M$ de todas las disponibles, a dicho elemento. Así como en la posterior asignación aleatoria de un par DVS_j (de voltajes/velocidades), de los asociados con dicha m_j al mismo.

Algoritmo 9 Mutación uniforme

Require: Solución a mutar s. Probabilidad de mutación β

1: **for** $i = 1 \rightarrow |T|$ **do**
2: $r \leftarrow DecimalAleat(0, 1)$ ▷ Genera un número decimal aleatorio entre 0 y 1
3: **if** $r \leq \beta$ **then**
4: $j \leftarrow Aleat(0, |M|)$ ▷ Selecciona aleatoriamente una máquina
5: $k \leftarrow Aleat(0, |DVS_j|)$ ▷ Selecciona aleatoriamente un voltaje para la máquina $m_j \in M$
6: $s_i \leftarrow (j, k)$ ▷ Asigna al elemento s_i los valores j y k
7: **end if**
8: **end for**

En [33] se establece una $\beta = 0.05$. Dicha asignación se emplea de igual manera en este trabajo.

Mutación sesgada

El segundo método de mutación propuesto consiste en un método de mutación sesgada (Algoritmo 10). Esté método es similar a la mutación uniforme, pero difiere en la manera en que se selecciona el par DVS que se asigna al elemento que se mutará. Mientras que la mutación uniforme da igual preferencia a todos los pares DVS (de ahí su nombre), la mutación sesgada establece un criterio probabilístico durante la selección de los pares.

Se comienza de manera similar a la mutación uniforme, seleccionando aleatoriamente una máquina m_j, que se asignará al elemento a alterar (Línea 4). Pero una vez fija la máquina se procede a asignar pesos de manera incremental a los elementos contenidos en el par DVS_j asociado con m_j (Línea 5). Para esto se define un vector X, tal que:

$$X = \{x_1, x_2, x_3, ..., x_n | x_i = W(i, n)\} \tag{4.1}$$

donde $n = |DVS_j|$ para el par DVS asociado con la máquina m_j y $W(i, n)$ es la función propuesta definida como:

$$W(i,n) = \begin{cases} \frac{2i}{n^2+n} + W(i-1,n), & i > 0 \\ 0, & \text{en caso contrario} \end{cases} \tag{4.2}$$

Esto asigna a X una serie de valores fraccionarios que incrementan de acuerdo a un factor constante $\kappa = 2/(n^2 + n)$ dado por la longitud n del vector DVS_j. De tal manera que $X_1 = \kappa$ y $X_n = 1$. Este vector de pesos acumulados X establece la probabilidad acumulada con que se elegirán los elementos del par DVS_j (Línea 7), siendo los elementos con índice mayor aquellos con mayor probabilidad de ser elegidos.

Algoritmo 10 Mutación sesgada

Require: Solución a mutar s. Probabilidad de mutación β

1: **for** $i = 1 \rightarrow |T|$ **do**
2: $r \leftarrow DecimalAleat(0, 1)$ ▷ Genera un número decimal aleatorio entre 0 y 1
3: **if** $r \leq \beta$ **then**
4: $j \leftarrow Aleat(0, |M|)$ ▷ Selecciona aleatoriamente una máquina
5: $X \leftarrow AsignaPesos(DVS_j)$ ▷ Asigna pesos acumulados a los elementos de DVS_j
6: $r \leftarrow DecimalAleat(0, 1)$ ▷ Genera un número decimal aleatorio entre 0 y 1
7: $k \leftarrow \text{máx}_l\{\forall x_l \in X : r < x_l\}$ ▷ Selecciona la posición en X tal que $r < x_l$
8: $s_i \leftarrow (j, k)$ ▷ Asigna al elemento s_i los valores j y k
9: **end if**
10: **end for**

Mutación sesgada inversa

Siguiendo el mismo principio de la mutación sesgada, mediante una evidente modificación del Algoritmo 10 se define un método de mutación sesgada inversa, que da preferencia a los elementos con menor índice dentro del par DVS_j. Así, se puede guiar al proceso de mutación para que este genere soluciones con voltajes altos (mejorando así el valor objetivo *makespan*) o con voltajes bajos (que mejora el valor objetivo energía).

Tomando una columna de la Tabla 1.1, se presentan las Tablas 4.7 y 4.8. Estas muestran las probabilidades (no acumuladas) de ser seleccionados asignados a cada par DVS (de la Máquina 1) durante un ciclo de mutación sesgada y mutación sesgada inversa, respectivamente. Las probabilidades (columna p) son asignadas de acuerdo al factor κ, definido en la sección anterior (ver ecuación 4.2).

Tabla 4.7: Probabilidades generadas durante mutación sesgada

Máquina 1		
v_k	rs_k	p
1.50	1	0.036
1.40	0.90	0.071
1.30	0.80	0.107
1.20	0.70	0.143
1.10	0.60	0.179
1.00	0.50	0.214
0.90	0.40	0.250

Tabla 4.8: Probabilidades generadas durante mutación sesgada inversa

Máquina 1		
v_k	rs_k	p
1.50	1	0.250
1.40	0.90	0.214
1.30	0.80	0.179
1.20	0.70	0.143
1.10	0.60	0.107
1.00	0.50	0.071
0.90	0.40	0.036

4.5. Detección de estancamiento en las celdas

Se proponen dos métodos para la detección del estancamiento de las celdas: El primero, basado en las medias agrupadas de los valores objetivos de las soluciones contenidas en la población; el segundo, en un indicador de calidad multiobjetivo: ϵ.

4.5.1. Estancamiento por medias

El Algoritmo 11 muestra el proceso de detección de estancamiento por medias. El primer paso es tomar los individuos contenidos en el conjunto de padres $PCell.P$ y calcular el promedio de los valores objetivos de los mismos, tanto para el *makespan*, como para la energía (Líneas 4-5). Posteriormente, se comparan uno a uno estos promedios con los promedios previamente guardados para el conjunto P de la generación anterior (Línea 6). Si los promedios calculados no mejoran con respecto a los anteriores en al menos un objetivo, se establece la celda como estancada (Líneas 10-12).

Algoritmo 11 Estancamiento por medias

Require: Celda de procesamiento $PCell$.

1: $t \leftarrow PCell.cicloActual$
2: $k \leftarrow 0$
3: $Obj \leftarrow \{ObjetivoMakespan, ObjetivoEnergia\}$
4: **for all** $o_i \in Obj$ **do**
5: $\quad PCell.\mu_t.o_i \leftarrow MediaAritmetica(PCell.P_t, o_i)$
6: $\quad$ **if** $PCell.\mu_t.o_i \geq PCell.\mu_{t-1}.o_i$ **then**
7: $\quad\quad k \leftarrow k + 1$
8: $\quad$ **end if**
9: **end for**
10: **if** $k = |Obj|$ **then**
11: $\quad PCell.estancada \leftarrow True$
12: **end if**

4.5.2. Estancamiento ϵ

El indicador ϵ

La indicador ϵ (I_ϵ) es un indicador de calidad multiobjetivo binario propuesto en [45].

Sea $P \in \mathbb{R}^{+^n}$ un problema de minimización con n objetivos positivos, se dice que un vector $a = (a_1, a_2, ..., a_n) \in P$ ϵ-domina a otro vector $b = (b_1, b_2, ..., b_n) \in P$, denotado como $a \succeq_\epsilon b$, si y sólo si:

$$\forall i : 1 < i < n : a_i \leq \epsilon \cdot b_i, \tag{4.3}$$

para una $\epsilon > 0$ dada. El indicador I_ϵ se define como [45]:

$$I_\epsilon(A, B) = \inf_{\epsilon \in \mathbb{R}} \{\forall b \in B \exists a \in A : a \succeq_\epsilon b\} \tag{4.4}$$

En otras palabras, ϵ es el factor mínimo que hay que aplicar al conjunto A para que cada solución en B sea dominada por al menos una solución en A. Véase la figura 4.2.

Una $\epsilon = 1$ indica que A es equivalente a B; si $\epsilon < 1$, el conjunto B se puede considerar "superior" al conjunto A. Lo contrario aplica si $\epsilon > 1$.

Detección de estancamiento mediante I_ϵ

Se propone un método de detección de estancamiento basado en el indicador I_ϵ: estancamiento ϵ (Algoritmo 12). Para una generación t dada dentro de un celda $PCell$, el método extrae el conjunto de soluciones no dominadas $PCell.PF_t \subset PCell.P_t$ de la población de $PCell$ y calcula el indicador $I_\epsilon(PCell.PF_t, PCell.PF_{t-1})$ (Línea 2), donde $PCell.PF_{t-1}$ es el conjunto de soluciones no dominadas extraído de $PCell$ en su generación anterior. Si I_ϵ no determina mejoría de $PCell_t.PF$ con respecto a $PCell.PF_{t-1}$

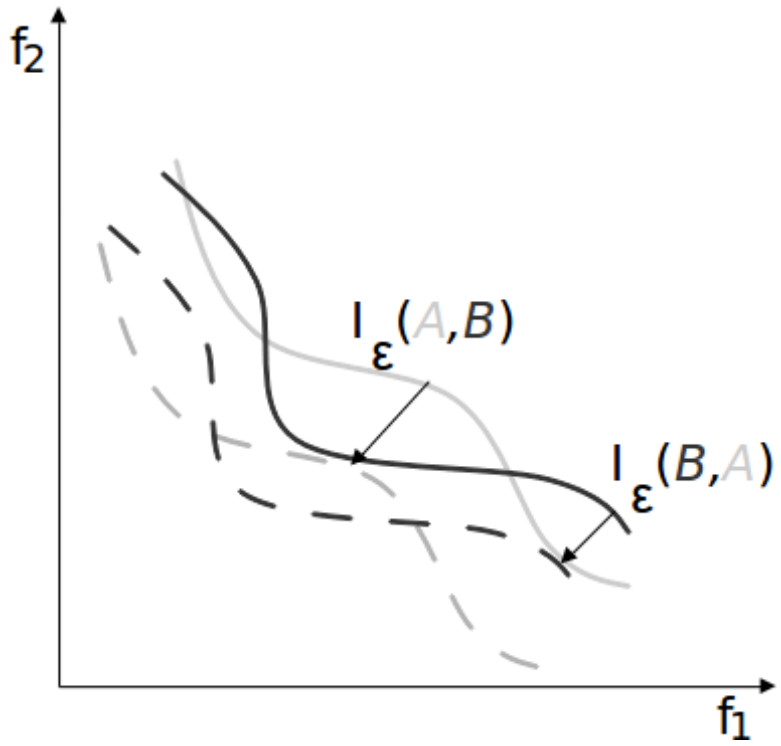

Figura 4.2: Indicador ϵ para un problema de minimización de dos objetivos

se incrementa un contador de ciclos sin mejora (Línea 3), en caso contrario, dicho contador se restablece a 0 (Línea 5). Una vez llegado a un número límite de generaciones sin mejora (LGSM) previamente establecido, se considera *PCell* como estancada (Líneas 7-9).

Algoritmo 12 Estancamiento ϵ

Require: Celda de procesamiento *PCell*

1: $t \leftarrow PCell.cicloActual$
2: **if** $I_\epsilon(PCell.PF_t, PCell.PF_{t-1}) \geq 1$ **then**
3: $\quad PCell.contadorEst \leftarrow PCell.contadorEst + 1$
4: **else**
5: $\quad PCell.contadorEst \leftarrow 0$
6: **end if**
7: **if** $PCell.contadorEst = LGSM$ **then**
8: $\quad PCell.estancada \leftarrow True$
9: **end if**

4.6. Comunicación entre celdas

Una vez que todas las celdas se han estancado y consecuentemente han dejado de ser procesadas, se efectúa un proceso de comunicación entre celdas a manera de que estas intercambien información y produzcan nuevas soluciones. Para entender el proceso

global de comunicacón entre todas las celdas del algoritmo de procesamiento celular propuesto, primero es necesario entender como se realiza la comunicación entre dos celdas.

4.6.1. Path Relinking entre celdas

El Algoritmo 13 describe el método propuesto para la generación de nuevas soluciones a partir de las poblaciones contenidas en dos celdas dadas. Este método implementa una variación de la técnica conocida como "Reencadenamiento de Trayectorias" (*Path Relinking* [13], por su nombre en inglés) pues busca aproximar una solución origen so a otra solución guía sg, devolviendo como resultado una solución situada en un punto intermedio (dado por un grado de similitud deseado α) de la trayectoria que sigue dicha aproximación.

El primer paso en este algoritmo (Algoritmo 13) consiste en seleccionar las soluciones so y sg. Estas son tomadas aleatoriamente de los conjuntos de soluciones no dominadas $PCell_i.PF \subseteq PCell_i.P$ de cada celda, asegurándose que la distancia de Hamming [17] entre ellas sea de al menos $n\alpha$ (Líneas 1-4), donde n es la longitud del cromosoma usado para representar las soluciones. Posteriormente, se entra a un ciclo (Línea 9) en el cual se procede a recorrer las soluciones seleccionadas en un orden aleatorio, previamente generado (Línea 5). En cada ciclo de este recorrido se toma el elemento actual de sg y se sitúa en la posición actual de so, siempre y cuando los elementos en cuestión no sean idénticos (Líneas 9 y 10). A continuación se evalua so con el cambio efectuado. Si la nueva solución intermedia so no es dominada por ninguna de las soluciones en $PCell_i.PF$, esta se copia dentro de un conjunto denominando S y se actualiza $PCell_i.PF$ (Líneas 14-16). Esto permite localizar nuevas soluciones no dominadas durante el trayecto hacia sg.

Se continúa el recorrido por las soluciones hasta que se llegue a un número máximo de cambios realizados, definido por α. Por último, se añade la solución final al conjunto S y este se devuelve como resultado.

Algoritmo 13 Path Relinking entre dos celdas

Require: $PCell_i$. $PCell_j$. α

1: **repeat**
2: $so \leftarrow$ Extraer individuo aleatoriamente de $PCell_i.PF$
3: $sg \leftarrow$ Extraer individuo aleatoriamente de $PCell_j.PF$
4: **until** $HammingDistance(so, sg) >= (n\alpha))$
5: $O \leftarrow GenerarOdenamientoAleatorio(1, n)$
6: $k \leftarrow 0$
7: $cambios \leftarrow 0$
8: $S \leftarrow \emptyset$
9: **while** $(cambios < (n\alpha))$ **do**
10: $t \leftarrow O_k$
11: **if** $so_t \neq sg_t$ **then**
12: $so_t \leftarrow sg_t$
13: $Evaluar(so)$
14: **if** $\forall x_l \in PCell_i.PF : so \preceq x_l$ **then**
15: $S \leftarrow S \cup so$
16: Actualizar $PCell_i.PF$ con el vector objetivo de so
17: **end if**
18: $cambios \leftarrow cambios + 1$
19: **end if**
20: $k \leftarrow k + 1$
21: **end while**
22: $S \leftarrow S \cup so$
23: **return** S

Path Relinking bidireccional

Es posible definir un *Path Relinking* "bidireccional", en el cual, un vez generada la solución final (Línea 22), se repite el proceso a partir de la Línea 9, invirtiendo *so* y *sg*. Para esto, es evidente la necesidad de almacenar una copia de *so* al momento de la selección de soluciones.

Esto permite obtener al menos dos nuevas soluciones por llamada: *so* aproximada a *sg* y *sg* aproximada a *so*. En contraste con el método "unidireccional" que regresa una solución (*so* aproximada a *sg*) como mínimo.

4.6.2. Proceso de comunicación global

Complementariamente, el Algoritmo 14 describe el proceso completo de comunicación entre todas las celdas. Como se mencionó anteriormente, este se realiza una vez que todas las celdas se han estancado. Dado el arreglo de celdas $PCells$, cada celda $PCell_i \in PCells$ se comunica con la celda consecuente a ella dentro del arreglo $PCell_{i+1}$

(Línea 2). Se considera que las celdas se encuentran en un arreglo circular, por lo que la última celda se comunica con la primera (Líneas 3-5). Una vez seleccionada la celda $PCell_{sig}$ con que la celda actual $PCell_i$ se comunicará, se generan nuevas soluciones a través del Path Relinking de $PCell_i$ a $PCell_{sig}$ y se almacenan en $PCell_i.Q$. Este proceso se repite hasta que $PCell.Q$ tenga la misma cantidad de soluciones que $PCell_i.P$ (Líneas 7-9)

Una vez efectuada la comunicación entre todas las celdas, se unen los conjuntos $PCell_i.P$ y $PCell_i.Q$ de cada celda en un solo conjunto P (Líneas 12-14), que constituye la nueva población global del algoritmo.

Algoritmo 14 Proceso de comunicación entre celdas

Require: Conjunto de celdas de procesamiento $PCells$

1: **for all** $PCell_i \in PCells$ **do**
2: $PCell_{sig} \leftarrow PCell_{i+1}$
3: **if** $i = |PCells|$ **then**
4: $PCell_{sig} \leftarrow Pcells_0$
5: **end if**
6: $Pcell_i.Q \leftarrow \emptyset$
7: **while** $|Pcell_i.Q| < |PCell_i.P|$ **do**
8: $PCell_i.Q \leftarrow PCell_i.Q \cup PathRelinking(PCell_i, PCell_{sig})$
9: **end while**
10: **end for**
11: $P \leftarrow \emptyset$
12: **for all** $PCell_i \in PCells$ **do**
13: $P \leftarrow U \cup PCell_i.P \cup PCell_i.Q$
14: **end for**
15: **return** P

4.7. Ciclo principal

El Algoritmo 15 define el ciclo principal del algoritmo de procesamiento celular propuesto. Se comienza por generar un número fijo de soluciones aleatorias, las cuales se reparten de manera equitativa entre las celdas disponibles para su inicialización (Líneas 1-5). Durante la inicialización de cada celda se pueden definir parámetros propios de la misma; como el tipo de mutación, de estancamiento, etc. Posteriormente, se comienza el ciclo de procesamiento pseudo-paralelo de las celdas, y el proceso de detección de estancamiento (Líneas 7-12). Cada celda $PCell_i \in PCells$ es procesada hasta que esta se estanca. Una vez que todas las celdas se han estancado concluye el procesamiento principal y se entra al proceso de comunicación entre celdas (Línea 14). La población devuelta por el proceso de comunicación es procesada para extraer el conjunto de soluciones no dominadas $PF \subseteq P$ o para ser refinada mediante la ejecución de un

nuevo proceso de NSGAII (Línea 15). Por último, el algoritmo de procesamiento celular devuelve el conjuto de soluciones no dominadas extraído de la población global.

Algoritmo 15 Algoritmo de procesamiento celular

Require: $PCells$ Conjunto de celdas de procesamiento

1: $P \leftarrow$ Generar N soluciones aleatorias
2: Dividir P en $|PCells|$ grupos de $N/|PCells|$ soluciones cada uno
3: **for** $i = 1 \rightarrow |PCells|$ **do**
4: $\quad Inicializar(PCell_i, P_i)$
5: **end for**
6: **while** Al menos una $PCell_i \in PCells$ no está estancada **do**
7: $\quad$ **for all** $PCell_i \in PCells$ **do**
8: $\quad\quad$ **if** $Pcell_i$ no está estancada **then**
9: $\quad\quad\quad ProcesaPCell(PCell_i)$
10: $\quad\quad\quad DetectaEstancamiento(PCell_i)$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end while**
14: $P \leftarrow ComunicaPCells()$
15: $PF \leftarrow PostProcesamiento(P)$
16: **return** PF

Capítulo 5

Experimentación y resultados

Se reportan resultado experimentales para el algoritmo propuesto en este trabajo y se comparan contra los obtenidos por la versión de NSGAII originalmente definida en [33]. Se denomina esta versión «monolítica», dado que no está basada en el principio de procesamiento celular, no cuenta con detección de estancamiento, ni método de comunicación.

5.1. Configuración de la experimentación

Se presentan cuatro configuraciones diferentes del algoritmo de procesamiento celular, comparándolas todas con la versión monolítica de NSGAII. Los parámetros de cada configuración, así como los del algoritmo de referencia, se muestran en la Tabla 5.1.

Tabla 5.1: Configuración de los algoritmos usados en la experimentación

	NSGAII	conf43	conf42	conf36	conf35
Número de celdas	1	2	2	2	2
Población global	200	200	200	200	200
Generaciones máximas	300	300	300	200	200
Criterio de estancamiento	N/A	Medias	Medias	ϵ (30 LGSM)	Medias
Mutación	Uniforme	Uniforme	Sesgada	Sesgada	Sesgada
Path Relinking	N/A	Bilateral	Bilateral	Unilateral	Bilateral
α	N/A	0.1	0.1	0.1	0.1
Post Procesamiento	N/A	Ninguno	Ninguno	100	100

El tamaño de la población global fijado para todos los algoritmos fue de 200 individuos. Con el fin de establecer una igualdad de condiciones, se estableció como criterio de paro un máximo de 300 generaciones para cada ejecución (200 para las configuraciones que añaden post procesamiento). En el caso del algoritmo celular, este número de generaciones máximas se establecen por celda, de manera complementaria al criterio

de detección de estancamiento.

Las configuraciones con mutación sesgada aplican mutación sesgada a las celdas impares y mutación sesgada inversa a la celdas pares. Las configuraciones que implementan estancamiento ϵ establecen un límite de generaciones sin mejora (LGSM) de 30. Por último, el post procesamiento consiste en la ejecución de NSGAII monolítico durante 100 generaciones, usando como población inicial la generada tras el proceso de comunicación entre celdas.

La experimentación se realizó en un sistema con procesador Intel Core i3 3220 a 3.3 Ghz y 8 GB de RAM ejecutando el núcleo Linux 3.8 en su versión de 64 bits. El lenguaje usado para la programación de los algoritmos es ANSI C, compilado con GCC 4.6. Se resolvió cada instancia 10 veces por algoritmo y se presentan las medias de dichas ejecuciones.

5.2. Instancias usadas

Las instancias utilizadas para la evaluación experimental son las previamente usadas en [33], siendo estas un subconjunto de 32 instancias tomadas de las anteriormente definidas en [14]. Dichas instancias son extraídas del conjunto de instancias STG[1] y constituyen la representación, a manera de DAG, de cuatro aplicaciones paralelas reales:

- El problema fpppp, extraído de los benchmarks SPEC: Instancias denominadas *fpppp*.
- Un porción del flujo de trabajo del proyecto Laser Interferometer Gravitational-Wave Observatory: Instancias de tipo *LIGO*.
- Una aplicación de control de un robot: Instancias *robot*.
- Una aplicación solucionadora de matrices disperas: Instancias *sparse*.

Asimismo, la notación de las instancias tiene la siguiente nomenclatura (ver figura 5.1):

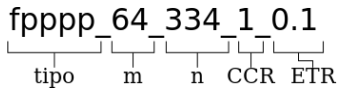

Figura 5.1: Nomenclatura de las instancias

El primer elemento en el nombre de la instancia indica el tipo (*fpppp, LIGO, robot* o *sparse*). El primer número indica el número de máquinas m, mientras que el segundo indica el número de tareas n, siendo estos valores los que determinan el tamaño de la instancia. El tercer valor indica el *Cociente de Comunicación a Computación* o CCR [14], el cual es la razón entre la media de los costos de ejecución $\bar{c}$ y la media de los costos

[1] `http://www.kasahara.elec.waseda.ac.jp/schedule/stgarc_e.html`

de comunicación $\bar{c}$ (ver sección 1.1.2). Por último, el cuarto valor indica el *Cociente de Aristas a Tareas ETR* [14], que es la razón entre el número de aristas $|E|$ en G y el número de tareas $|T|$.

Por último, todas las instancias hacen uso de una misma configuración de máquinas (o pares DVS) con seis máquinas diferentes, un mínimo de pares DVS de 3 y un máximo de 7. Para conseguir el número necesario de máquinas que precisa cada instancia los conjuntos de pares DVS se asignan de manera equitativa usando el principio de *round-robin* (ver apéndice A).

5.3. Indicadores de calidad

Dado el enfoque de solución multiobjetivo seguido en este trabajo, resulta inapropiado presentar un comparativo de las soluciones arrojadas por los algoritmo directamente, las cuales consisten en conjuntos de soluciones no dominadas. Por ello se hace uso de tres indicadores de calidad multiobjetivo.

En [45] se presenta la siguiente definición:

Un indicador de calidad *m-ario* I es una función $I : \Omega^m \to \mathbb{R}$, que asigna a cada vector $(A_1, A_2, ..., A_n)$ de los m conjuntos de referencia un valor real $I(A_1, ..., A_m)$, donde Ω es el conjunto de todos los conjuntos de referencia $A_1, ..., A_m$.

Los resultados de la experimentación realizada en este trabajo se evaluaron usando tres indicadores de calidad multiobjetivo unarios: *Hypervolume*, *Generational Distance* y *Spread*

5.3.1. Hypervolume

El *Hypervolume* [3] I_H o Métrica S [25] es un indicador de calidad ampliamente usado en optimización multiobjetivo. Se trata de un valor escalar que representa el volumen del espacio n-dimensional dominado por las soluciones en el conjunto de referencia A, donde n es el número de objetivos. De tal manera que un número mayor indica un mayor espacio de dominancia, cualidad altamente deseable en todo algoritmo multiobjetivo. Ver figura 5.2.

El *Hypervolume* I_H, para una conjunto de referencia (conjunto de soluciones no dominadas) A dado está definido como [25]:

$$I_H(A) = \lambda(\bigcup_{a \in A} \{x | a \prec x \prec x_{ref}\}), \tag{5.1}$$

donde λ denota la medida de Lebesgue y $x_{ref} = (r_1, ..., r_n)$ es un punto de referencia que es dominado por todas las soluciones factibles: el "anti-oṕtimo". Para problemas de maximización usualmente se elige el origen como punto de referencia [43]. Sin embargo, en problemas de minimización resulta común que se desconozca el anti-óptimo, por lo que en dado caso se sugiere conformar el punto de referencia identificando el peor valor posible para cada objetivo e incrementándolo en uno [25].

La figura 5.2 muestra la representación gráfica del *Hypervolume* para un problema de minimización de dos objetivos (f_1 y f_2). La línea remarcada representa el conjunto de referencia A, mientras que el espacio sombreado representa el *Hypervolume* para A, calculado con un punto de referencia r.

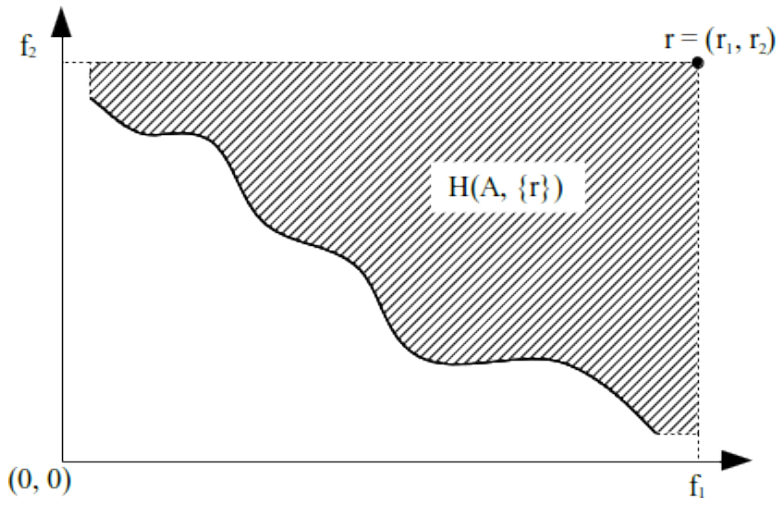

Figura 5.2: Hypervolume para un problema de minimización de dos objetivos

5.3.2. Generational Distance

La *Generational Distance* I_{GD} es una métrica de convergencia del algoritmo. Representa la distancia que existe entre el conjunto de referencia A encontrado y el verdadero frente de Pareto PF_{true} [41]. Una *Generational Distance* de 0 indica que A contiene elementos en el verdadero frente de Pareto.

$$I_{GD} = \frac{(\sum_{i=1}^{n} d_i^p)^{1/p}}{n} \tag{5.2}$$

Donde n es el número de vectores en A, $p = 2$, y d_i es la distancia euclidiana entre cada vector y el miembro más cercano de PF_{true}. Un resultado de 0 indica que $PF_{true} = A$. Cualquier valor mayor indica la medida en que A difiere de PF_{true}, por lo que se busca que esta métrica se minimice.

La figura 5.3 describe gráficamente la *Generational Distance* para un problema de minimización de dos objetivos. Los puntos sombreados representan las soluciones contenidas en el conjunto de referencia A, mientras que los puntos vacíos representan las soluciones presentes en el verdadero frente de Pareto PF_{true}. Las distancias euclidianas entre cada punto sombreado y el punto vacío más cercano al mismo forman el conjunto de valores que toma d_i en la ecuación 5.2.

5.3.3. Spread

La métrica *Spread* I_S, también llamada Δ[7], es un indicador de diversidad que representa la distancia que existe entre dos soluciones consecutivas, y sólo es aplicable

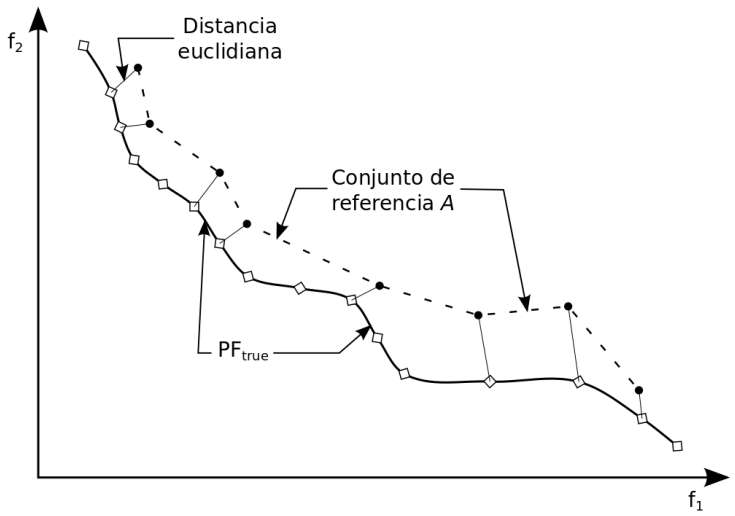

Figura 5.3: Generational distance para un problema de minimización de dos objetivos

con problemas bi-objetivo. La *Spread* para un conjunto de referencia dado A se define como:

$$I_S = \frac{d_f + d_l + \sum_{i=1}^{n-1} |d_i - \bar{d}|}{d_f + d_l + (n-1)\bar{d}} \tag{5.3}$$

Donde d_f y d_l son las distancias euclidianas entre las soluciones extremas del verdadero frente de Pareto PF_{true} y las soluciones extremas de A, $\bar{d}$ es el promedio de todas las distancias $d_i, i = 1, 2, ..., (n-1)$ y n es el número de soluciones en A. Una Δ de 0 indica una distribución uniforme de las soluciones en el frente de Pareto, por lo que se busca que está métrica tienda a 0.

La figura 5.4 es una gráfica que contiene los elementos necesarios para el cálculo de la *Spread* para un problema de minimización de dos objetivos. Los puntos sombreados representan las soluciones contenidas en el conjunto de referencia A y los puntos vacíos las contenidas en el verdadero frente de Pareto PF_{true}. Las distancias euclidianas entre cada solución contenida en A y su pariente más cercano (también en A) conforman los valores de d_i en la ecuación 5.3. Los valores de d_f y d_l son las distancias euclidianas entre cada punto extremo en PF_{true} y la solución más cercana a cada uno en A.

5.4. Resultados

Las Tablas 5.2, 5.3 y 5.4 muestra las medias de los indicadores de calidad multiobjetivo *Hypervolume* (I_H), *Generational Distance* (I_{GD}) y *Spread* (I_S), respectivamente, calculados para los algoritmos por separado. La primera columna muestra el nombre de la instancia, la segunda muestra los valores obtenidos por el algoritmo monolítico y el resto los valores obtenidos por el algoritmo de procesamiento celular en sus diversas

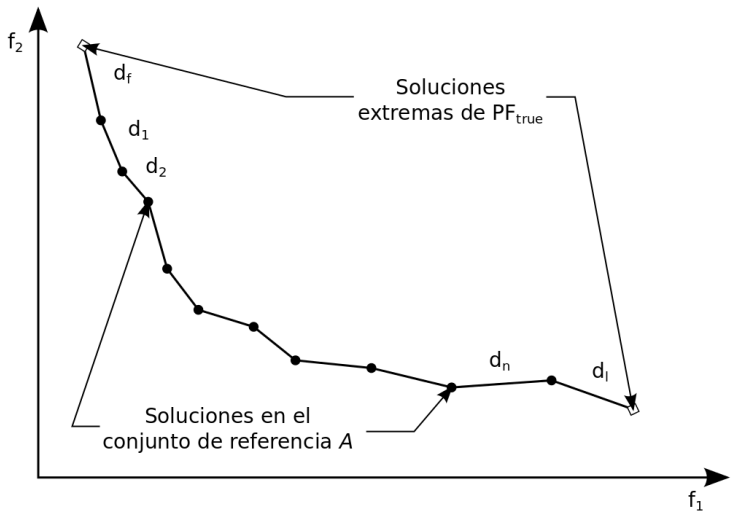

Figura 5.4: Spread para un problema de minimización de dos objetivos

configuraciones.

Asimismo, la Tabla 5.5 muestra los porcentajes de mejora promedio de cada configuración con respecto al algoritmo monolítico de referencia, para cada indicador de calidad. La primera columna muestra los indicadores, mientras que el resto de las columnas muestran el porcentaje de mejora promedio exhibido por cada uno de las configuraciones del algoritmo de procesamiento celular propuesto.

Como se puede apreciar, el algoritmo propuesto supera al algoritmo de referencia en casi un 10 % en cuanto a espacio de dominancia de las soluciones, así como en cercanía al verdadero frente de Pareto de las mismas (con una mejora porcentual promedio de más de 23 %). De igual manera, la dispersión de las soluciones presentes en los frentes de Pareto obtenidos por el algoritmo propuesto se mantiene estable, sin mejoras significativas.

Por otro lado, los mejores resultados están dados por la combinación de la mutación sesgada, estancamiento por medias, path relinking bilateral y post procesamiento. La combinación de estancamiento ϵ con path relinking unidireccional y post procesamiento también muestra resultados competitivos.

5.4.1. Pruebas estadísticas

Con el fin de validar los resultados, se llevaron a cabo una serie de pruebas estadísticas no paramétricas, siguiendo la metodología propuesta en [11].

La Tabla 5.6 muestra los resultados de la prueba de Friedman [11], para los tres indicadores de calidad. Se realizó esta prueba para las cuatro configuraciones y el algoritmo de referencia (columna 2), así como para las cuatro configuraciones únicamente (columna 3). Como se puede apreciar, para un nivel de significancia de 0.05, se rechaza la hipótesis nula H_0 de Friedman de las medias de las poblaciones son iguales en todos los casos. Esta afirmación es sustentada por los $p-values$ mostrados en la Tabla, puesto

Tabla 5.2: Resultados para Hypervolume

	NSGAII	conf43	conf42	conf36	conf35
fpppp_64_334_0.1_0.1	3044158	3261427	3674990	3669269	3562839
fpppp_64_334_0.1_10	705970915	720571926	712201408	751987223	759736667
fpppp_64_334_1_0.1	963429	1045636	1231551	1155464	1202587
fpppp_64_334_1_10	1605039872	1715902108	1692439097	1870578273	1879869601
fpppp_8_334_0.1_0.1	20606	19673	27580	27054	26908
fpppp_8_334_0.1_10	35232136	38357951	39940105	39358736	39200998
fpppp_8_334_1_0.1	79474	84469	92386	91266	92988
fpppp_8_334_1_10	355063	378108	397716	384506	380152
LIGO_64_76_0.1_0.1	5730346	5916130	6043191	6149091	6207959
LIGO_64_76_0.1_10	279705109	290379549	304895188	307306736	311924492
LIGO_64_76_1_0.1	12218801	12857727	13676348	13663738	13843150
LIGO_64_76_1_10	2534917	2756151	2615152	2950079	2841336
LIGO_8_76_0.1_0.1	3720760	3844358	3938343	4113789	4195755
LIGO_8_76_0.1_10	53702472	57159450	58198204	57309037	55948648
LIGO_8_76_1_0.1	699398	667213	716274	726320	724095
LIGO_8_76_1_10	90571603	97981648	96058848	94101296	95978211
robot_64_88_0.1_0.1	297984	315053	348492	339030	346571
robot_64_88_0.1_10	3841302704	3950971782	4093738612	4258842854	4281681723
robot_64_88_1_0.1	78031262	81646801	86885557	86607107	85614522
robot_64_88_1_10	3661377655	3858369960	3971740535	4203726907	3995521905
robot_8_88_0.1_0.1	14461697	15191173	15495176	16616073	16031213
robot_8_88_0.1_10	378654469	374309717	392234416	400823556	396589274
robot_8_88_1_0.1	1659816	1720106	1704427	1773528	1788187
robot_8_88_1_10	25674980	28088330	28231180	28475403	28411721
sparse_64_96_0.1_0.1	2009710	2048380	2066822	2063404	2044795
sparse_64_96_0.1_10	1330198	1344479	1529591	1516891	1712116
sparse_64_96_1_0.1	23938142	25075295	25819464	25639166	26147022
sparse_64_96_1_10	309082142	325819646	317631890	335297204	338078284
sparse_8_96_0.1_0.1	11791	11881	12932	12846	12508
sparse_8_96_0.1_10	22491244	23195003	23588028	23652788	23434553
sparse_8_96_1_0.1	3613706	3662963	3735735	3754208	3706561
sparse_8_96_1_10	35959286	36865795	36948012	37293603	36992508
PROMEDIO	349983933	364994371	373058039	393125201	387932808

Tabla 5.3: Resultados para Generational Distance

	NSGAII	conf43	conf42	conf36	conf35
fpppp_64_334_0.1_0.1	237.90	245.30	173.68	169.35	170.31
fpppp_64_334_0.1_10	514.92	478.15	349.25	402.24	414.78
fpppp_64_334_1_0.1	42.86	34.62	31.63	31.73	30.18
fpppp_64_334_1_10	30.56	27.21	24.62	24.98	26.33
fpppp_8_334_0.1_0.1	0.13	0.16	0.14	0.13	0.13
fpppp_8_334_0.1_10	9.36	7.31	5.61	4.82	6.71
fpppp_8_334_1_0.1	3.09	2.97	2.61	1.89	1.95
fpppp_8_334_1_10	5.84	5.95	5.34	4.88	4.55
LIGO_64_76_0.1_0.1	9.01	9.99	9.14	6.97	6.94
LIGO_64_76_0.1_10	130.05	117.84	115.44	115.03	109.77
LIGO_64_76_1_0.1	42.08	34.83	24.35	25.33	22.89
LIGO_64_76_1_10	12.55	11.20	11.69	9.62	11.33
LIGO_8_76_0.1_0.1	0.58	0.56	0.47	0.43	0.41
LIGO_8_76_0.1_10	42.83	37.34	32.09	31.38	29.23
LIGO_8_76_1_0.1	1.12	1.29	1.00	0.72	0.95
LIGO_8_76_1_10	25.15	20.68	21.14	22.25	17.84
robot_64_88_0.1_0.1	6.86	6.57	5.16	4.63	4.05
robot_64_88_0.1_10	22.53	19.71	17.99	16.93	17.31
robot_64_88_1_0.1	98.36	70.26	55.91	64.83	59.91
robot_64_88_1_10	1200.99	1164.86	1259.01	1131.74	1126.85
robot_8_88_0.1_0.1	3.03	3.33	2.43	2.10	2.36
robot_8_88_0.1_10	46.98	44.87	36.34	32.19	37.54
robot_8_88_1_0.1	1.83	1.71	1.46	1.19	1.16
robot_8_88_1_10	5.07	3.95	4.13	3.84	4.44
sparse_64_96_0.1_0.1	6.51	6.06	6.22	5.44	5.48
sparse_64_96_0.1_10	64.43	65.75	51.33	62.79	46.06
sparse_64_96_1_0.1	61.32	44.51	48.72	35.74	38.03
sparse_64_96_1_10	900967.51	951713.88	844066.37	830068.69	832579.37
sparse_8_96_0.1_0.1	0.17	0.13	0.13	0.13	0.15
sparse_8_96_0.1_10	2.83	2.57	2.40	2.03	1.75
sparse_8_96_1_0.1	0.67	0.53	0.57	0.51	0.50
sparse_8_96_1_10	10.35	11.76	10.57	10.25	10.30
PROMEDIO	28237.73	29818.62	26449.28	26009.21	26087.17

Tabla 5.4: Resultados para Spread

	NSGAII	conf43	conf42	conf36	conf35
fpppp_64_334_0.1_0.1	0.99	1.00	0.98	0.99	0.99
fpppp_64_334_0.1_10	1.00	1.00	0.99	1.00	1.00
fpppp_64_334_1_0.1	0.99	0.99	0.98	0.99	0.99
fpppp_64_334_1_10	1.00	1.00	0.99	1.00	1.00
fpppp_8_334_0.1_0.1	0.87	0.91	0.82	0.84	0.81
fpppp_8_334_0.1_10	0.98	0.98	1.01	0.97	0.97
fpppp_8_334_1_0.1	0.96	0.98	0.97	0.96	0.96
fpppp_8_334_1_10	0.97	0.97	0.97	0.95	0.96
LIGO_64_76_0.1_0.1	0.97	0.98	0.98	0.98	0.98
LIGO_64_76_0.1_10	1.00	1.00	1.00	1.00	1.00
LIGO_64_76_1_0.1	1.00	1.00	0.99	0.99	0.98
LIGO_64_76_1_10	1.00	1.00	1.00	1.00	1.00
LIGO_8_76_0.1_0.1	0.93	0.95	0.94	0.96	0.94
LIGO_8_76_0.1_10	1.00	1.00	0.99	0.99	0.99
LIGO_8_76_1_0.1	0.95	0.94	0.98	0.95	0.95
LIGO_8_76_1_10	0.99	0.99	1.00	0.99	0.99
robot_64_88_0.1_0.1	0.98	0.99	0.98	0.98	0.98
robot_64_88_0.1_10	1.00	1.00	1.00	1.00	1.00
robot_64_88_1_0.1	1.00	0.99	0.99	0.99	0.99
robot_64_88_1_10	1.00	1.00	1.00	1.00	1.00
robot_8_88_0.1_0.1	0.96	0.97	0.96	0.92	0.95
robot_8_88_0.1_10	1.00	1.00	1.00	1.00	1.00
robot_8_88_1_0.1	0.96	0.96	0.95	0.94	0.94
robot_8_88_1_10	0.99	0.99	0.99	0.99	0.99
sparse_64_96_0.1_0.1	0.98	0.98	0.99	0.99	0.97
sparse_64_96_0.1_10	1.00	1.00	1.00	1.00	1.00
sparse_64_96_1_0.1	0.99	0.98	0.98	0.97	0.97
sparse_64_96_1_10	1.00	1.00	1.00	1.00	1.00
sparse_8_96_0.1_0.1	0.88	0.88	0.85	0.89	0.88
sparse_8_96_0.1_10	0.97	0.97	0.97	0.95	0.97
sparse_8_96_1_0.1	0.94	0.95	0.96	0.93	0.91
sparse_8_96_1_10	0.98	0.98	0.98	0.99	0.98
PROMEDIO	0.98	0.98	0.98	0.97	0.97

Tabla 5.5: Porcentajes de mejora promedio

	conf43	conf42	conf36	conf35
I_H	3.87 %	8.15 %	9.43 %	9.52 %
I_{GD}	6.89 %	17.43 %	23.93 %	23.38 %
I_S	-0.31 %	0.06 %	0.40 %	0.63 %

que todos son menores a 0.05, siendo los correspondientes a los indicadores I_H e I_{GD} los que presentan mayor peso. Esto se repite (con menor intensidad) aún cuando no se considera el algoritmo de referencia (columna 3).

Tabla 5.6: P-Values obtenidos mediante la prueba de Friedman

	NSGAII,43,42,36,35	43,42,36,35
I_H	9.31×10^{-18}	3.81×10^{-8}
I_{GD}	6.56×10^{-17}	3.19×10^{-10}
I_S	0.0001041	0.0002006

Una vez determinado que existe evidencia para decir que el desempeño de las cuatro configuraciones es diferente entre si, así como con respecto al algoritmo de referencia, se realizaron pruebas para validar que configuración presentan las mayores diferencias. Para esto se realizaron una seria de pruebas de Wilcoxon, también conocidas como pruebas de *Signed-Rank* [42]. La prueba de *Signed-Rank* de Wilcoxon compara dos poblaciones entre si. La hipótesis nula H_0 de Wilcoxon es que no existen diferencias significativas entre las medias de dichas poblaciones.

La Tabla 5.7 muestra los $p - values$ obtenidos como resultado de las pruebas de Wilcoxon realizadas. Se compara cada configuración con respecto al algoritmo de referencia NSGAII, para los tres indicadores de calidad. Como se puede apreciar, para un nivel de significancia de 0.05, se rechaza H_0 en la mayoría de los casos, con excepción del indicador I_S en las configuraciones *conf43* y *conf42*. Sin embargo, la columna que muestra mayor evidencia para rechazar H_0 es *conf35*, la cual corresponde a la configuración que implementa mutación sesgada, estancamiento por medias, path relinking bilateral y post procesamiento. Esto lleva a afirmar que es esta configuración la que presenta diferencias más significativas con respecto al algoritmo de referencia.

Tabla 5.7: P-values obtenidos mediante la prueba de signed-rank de Wilcoxon

	NSGAII vs...			
	conf43	conf42	conf36	conf35
I_H	1.73×10^{-6}	4.66×10^{-10}	4.66×10^{-10}	4.66×10^{-10}
I_{GD}	0.00934	5.98×10^{-6}	4.66×10^{-10}	4.66×10^{-10}
I_S	0.2618	0.8034	0.03249	0.001667

Capítulo 6

Conclusiones y trabajo futuro

En este trabajo se abordó el problema de calendarización de tareas en sistemas de procesamiento paralelo usando un enfoque de solución multiobjetivo. Se propuso un algoritmo de procesamiento celular basado en NSGAII y se le evaluó experimentalmente comparándolo con un algoritmo "monolítico" tomado del estado del arte. Los resultados experimentales muestran que el algoritmo NSGAII basado en procesamiento celular supera al algoritmo NSGAII monolítico del estado del arte, cuando se implementa mutación sesgada, *path relinking* bidireccional, estancamiento por medias y post procesamiento (ver capítulo 5).

Los resultados presentados muestran que se cumplieron satisfactoriamente todos los objetivos comprometidos inicialmente.

En este capítulo se enumeran las principales aportaciones científicas realizadas con este trabajo y las publicaciones generadas gracias al mismo. Asimismo, se describen las oportunidades de investigación identificadas como trabajo futuro.

6.1. Contribuciones científicas

Las aportaciones más relevantes de este trabajo son las siguientes:

6.1.1. Algoritmo NSGAII de procesamiento celular

La principal aportación de este trabajo es un método de solución multiobjetivo para el problema de calendarización de tareas en sistemas de procesamiento paralelo. Dicho método consiste en un algoritmo de procesamiento celular homogéneo basado en NSGAII (ver capítulo 4).

Como aportaciones secundarias se encuentran los componentes del método de solución y sus variantes, entre los que destacan los siguientes:

Operadores de mutación

Se propone un operador de mutación para la representación de las soluciones del problema de calendarización de tareas, denominado *mutación sesgada*. Este operador,

y su variante la *mutación sesgada inversa*, se caracterizan por favorecer la generación de soluciones que optimicen el *makespan* o la energía, según sea el caso (ver sección 4.4.3).

Métodos de detección de estancamiento

Se proponen dos métodos para la detección de estancamiento en una población de soluciones multiobjetivo, ambos útiles tanto para su uso en celdas de procesamiento como en cualquier metaheurística poblacional multiobjetivo: estancamiento por medias y estancamiento ϵ (ver sección 4.5).

Métodos de comunicación

Se contribuye con un método de comunicación entre dos conjuntos de soluciones no dominadas basado en el principio de reencadenamiento de trayectorias (*path relinking*), usado en este caso para la comunicación entre celdas (ver sección 4.6).

6.1.2. Métodos de solución exacta

Adicionalmente, a manera de validar los resultados obtenidos durante la fase inicial del desarrollo de este proyecto, se diseñó e implementó un método de solución exacta basado en la técnica de ramificación y acotamiento para el problema de calendarización en su versión mono-objetivo (minimización del *makespan*). La evaluación experimental de este método muestra su superioridad en términos de tiempo de solución con respecto a un algoritmo de enumeración exhaustiva. Este trabajo se presenta en [37].

6.2. Publicaciones

Resultados preliminares de este trabajo de investigación se han presentado en congresos científicos nacionales e internacionales. A continuación se listan las publicaciones generadas para dichas presentaciones:

- Algoritmos exactos de calendarización de tareas para programas paralelos en sistemas de procesamiento heterogéneos. En el *VII Encuentro de Investigadores del Instituto Tecnológico de Ciudad Madero (ITCM)*, 2012. A. Alejandro Santiago Pineda, Miguel A. Ramiro Zúñiga y Héctor J. Fraire Huacuja [37].

- Método de procesamiento celular aplicado al problema de calendarización de tareas en sistemas de procesamiento paralelo. En el *VII Encuentro de Investigadores del Instituto Tecnológico de Ciudad Madero (ITCM)*, 2013. Miguel A. Ramiro Zúñiga, Héctor J. Fraire Huacuja y Guadalupe Castilla Valdez [35].

- Task scheduling in heterogeneous computing systems using a microGA. En la *Eighth International Conference P2P, Parallel, Grid, Cloud and Internet Compu-*

ting (3PGCIC), 2013. J.E. Pecero, P. Bouvry, H.J.F. Huacuja, J.D.T. Villanueva, M.A.R. Zúñiga y C.G.G. Santillán [29].

6.3. Trabajo Futuro

Durante el desarrollo del proyecto se identificaron las siguientes áreas de oportunidad para continuar con la investigación:

- Evaluar experimentalmente el uso de celdas de diversos tamaños, así como un número mayor de celdas.
- Evaluar el desempeño del algoritmo propuesto contra métodos de solución mono-objetivo.
- Implementar métodos de búsqueda local mono y multiobjetivo dentro de cada celda.
- Desarrollar un algoritmo de procesamiento celular completamente heterogéneo, es decir, que implemente diferentes metaheurísticas para cada celda.

Apéndice A

Configuración de máquinas

En este apartado se presenta la configuración de máquinas y sus respectivos pares DVS usados para resolver las instancias usadas en la experimentación.

Tabla A.1: Pares DVS usados en la experimentación

	Máquina 0		Máquina 1		Máquina 2		Máquina 3		Máquina 4		Máquina 5	
k	v_k	rs_k	v_k	rs_k	v_k	rs_k	v_k	rs_k	v_k	rs_k	v_k	rs_k
0	1.75	1	1.5	1	2.2	1	1.5	1	1.2	1	1.35	1
1	1.4	0.8	1.4	0.9	1.9	0.85	1.2	0.8	1.15	0.9	1.25	0.857
2	1.2	.6	1.3	.8	1.6	.65	.9	.5	1.1	.8	1.2	.715
3	.9	.4	1.2	.7	1.3	.5			1.05	.7	1.1	.571
4			1.1	.6	1	.35			1	.6	.9	.322
5			1	.5					.9	.5		
6			.9	.4								

Bibliografía

[1] Micael Gallego Carrillo Abraham Duarte Muñoz, Juan José Pantrigo Fernández. Metaheurísticas. Universidad Rey Juan Carlos, 2010.

[2] Chang Wook Ahn. Advances in Evolutionary Algorithms: Theory, Design and Practice (Studies in Computational Intelligence). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Anne Auger, Johannes Bader, Dimo Brockhoff, and Eckart Zitzler. Theory of the hypervolume indicator: Optimal μ-distributions and the choice of the reference point. In In Foundations of Genetic Algorithms (FOGA 2009). ACM, 2009.

[4] Carlos Artemio Coello Coello. An Empirical Study Of Evolutionary Techniques For Multiobjective Optimization In Engineering Design. PhD thesis, Tulane University, 1996.

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

[6] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. 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, September 2000. Springer.

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

[8] C.O. Diaz, M. Guzek, J.E. Pecero, G. Danoy, P. Bouvry, and S.U. Khan. Energy-aware fast scheduling heuristics in heterogeneous computing systems. In High Performance Computing and Simulation (HPCS), 2011 International Conference on, pages 478 –484, july 2011.

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

[10] G. Folino, C. Pizzuti, and G. Spezzano. Combining cellular genetic algorithms and local search for solving satisfiability problems. In Tools with Artificial Intelligence, 1998. Proceedings. Tenth IEEE International Conference on, pages 192 –198, nov 1998.

[11] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. Journal of Heuristics, 15:617–644, 2009.

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

[13] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. CONTROL AND CYBERNETICS, 39:653–684, 2000.

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

[15] Mateusz Guzek, Johnatan E. Pecero, Bernabé Dorronsoro, and Pascal Bouvry. A cellular genetic algorithm for scheduling applications and energy-aware communication optimization. In International Conference on High Performance Computing & Simulation (HPCS), pages 241–248. 2010.

[16] Min-You Wu Haluk Topcuoglu, Salim Hariri. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, pages 260–274, 2002.

[17] R. W. Hamming. Error detecting and error correcting codes. Bell System Technical Journal, 29(2):147–160, 1950.

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

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

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

[21] Young Choon Lee and Albert Y. Zomaya. 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, 2009. IEEE Computer Society.

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

[23] Young Choon Lee and Albert Y. Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst., 22:1374–1381, August 2011.

[24] Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.). Springer-Verlag, London, UK, UK, 1996.

[25] B. Naujoks, N. Beume, and M. Emmerich. Multi-objective optimisation using s-metric selection: application to three-dimensional solution spaces. In Evolutionary Computation, 2005. The 2005 IEEE Congress on, volume 2, pages 1282–1289 Vol. 2, Sept.

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

[27] Andrzej Osyczka. Multicriteria Optimization for Engineering Design. Academic Press, 1985.

[28] Christos Papadimitriou and Mihalis Yannakakis. 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, 1988. ACM.

[29] J.E. Pecero, P. Bouvry, H.J.F. Huacuja, J.D.T. Villanueva, M.A.R. Zuniga, and C.G.G. Santillan. Task scheduling in heterogeneous computing systems using a microga. In P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on, pages 618–623, Oct 2013.

[30] J.E. Pecero, H.J.F. Huacuja, P. Bouvry, A.A.S. Pineda, M.C.L. Loces, and J.J.G. Barbosa. 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, July.

[31] Johnatan E. Pecero, Pascal Bouvry, and Carlos J. Barrios. Low energy and high performance scheduling on scalable computing systems. In Latin-American Conference on High Performance Computing, pages 1–8, 2010.

[32] Johnatan E. Pecero, Pascal Bouvry, Hector J. Fraire Huacuja, and Samee U. Khan. 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, December 2011. IEEE CS Press. To appear.

[33] Aurelio Alejandro Santiago Pineda. Estrategias de búsqueda local para el problema de programación de tareas en sistemas de procesamiento paralelo. Master's thesis, Instituto Tecnológico de Ciudad Madero, 2013.

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

[35] Miguel Á. Ramiro Zuniga, Héctor J. Fraire Huacuja, J. David Terán Villanueva, and Guadalupe Castilla Valdez. Método de procesamiento celular aplicado al problema de calendarización de tareas en sistemas de procesamiento paralelo. In VII Encuentro de Investigadores del Instituto Tecnológico de Ciudad Madero (ITCM), Nov 2013.

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

[37] A. Alejandro Santiago Pineda, Miguel Á. Ramiro Zuniga, and Héctor J. Fraire Huacuja. Algoritmos exactos de calendarización de tareas para programas paralelos en sistemas de procesamiento heterogéneos. In VI Encuentro de Investigadores del Instituto Tecnológico de Ciudad Madero (ITCM), Nov 2012.

[38] Michael Sipser. Introduction to the Theory of Computation, 2nd Edition. Course Technology PTR, 2006.

[39] N. Srinivas and Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol. Comput., 2(3):221–248, September 1994.

[40] J.David Terán-Villanueva, HéctorJoaquínFraire Huacuja, JuanMartínCarpio Valadez, RodolfoA. Pazos Rangel, HéctorJoséPuga Soberanes, and JoséAntonio-Martínez Flores. Cellular processing algorithms. In Patricia Melin and Oscar Castillo, editors, Soft Computing Applications in Optimization, Control, and Recognition, volume 294 of Studies in Fuzziness and Soft Computing, pages 53–74. Springer Berlin Heidelberg, 2013.

[41] David A. Van Veldhuizen and G.B. Lamont. On measuring multiobjective evolutionary algorithm performance. In In 2000 Congress on Evolutionary Computation, pages 204–211, 2000.

[42] R. Walpole, R. Myers, S. Myers, and K. Ye. Probability and statistics for engineers and scientists (8th ed.). Prentice-Hall, NJ, USA, 2006.

[43] L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. Evolutionary Computation, IEEE Transactions on, 10(1):29–38, 2006.

[44] Rudolf Paul Wiegand. An analysis of cooperative coevolutionary algorithms. PhD thesis, George Mason University, Fairfax, VA, USA, 2004. AAI3108645.

[45] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. Evolutionary Computation, IEEE Transactions on, 7(2):117–132, 2003.