Una Heurística Cooperativa para resolver el Problema de Scheduling en un Cluster con Procesadores Homogéneos

Cristian Morales[a] Mauricio Solar[bc] Raul Monge[d]

Departamento de Informática
Universidad Técnica Federico Santa María (UTFSM)
Santiago, Chile

Victor Parada[e]

Departamento de Ingeniería Informática
Universidad de Santiago de Chile (USACH)
Santiago, Chile

Abstract

This article presents a cooperative heuristic based on evolutionary algorithms to solve the scheduling of a parallel application running on a cluster with homogeneous processors. The aim is to find optimal solutions of the scheduling in a prudent time. In order to achieve this objective we present the results obtained with a parallel implementation of this cooperative genetic algorithm. The results in terms of execution time of this parallel genetic algorithm are excellent when considering that the scheduling problem is a NP-hard problem.

Keywords: Scheduling, Cluster, Algoritmo Genético, Modelo Paralelo, Modelo Cooperativo.

1. Introducción

Para obtener la programación óptima en la asignación de recursos a una serie de consumidores, como en cualquier problema de optimización combinatoria, existen básicamente las búsquedas completas y las incompletas. Las búsquedas completas garantizan encontrar el óptimo por medio de la revisión sistemática y completa del espacio de búsqueda, analizando todas y cada una de las posibilidades en búsqueda

[a] Email: cristian.morales@inf.utfsm.cl
[b] Email: msolar@inf.utfsm.cl
[c] Thanks to project DGIP 24.11.42
[d] Email: rmonge@inf.utfsm.cl
[e] Email: victor.parada@usach.cl

de la mejor. Algunos mecanismos de técnicas completas se basan en formulaciones matemáticas como programación lineal (Pinedo, 2008). Éstas revisan cada uno de los caminos y evalúa cada potencial solución, garantizando la obtención del óptimo global, pero a un costo alto en términos de esfuerzo computacional. Lamentablemente, el problema es NP-completo y sufre de explosión combinatoria (Ullman, 1975; Papadimitriou 1994). Debido a esto, las técnicas completas en la práctica no pueden resolver problemas reales de gran tamaño en un tiempo razonable. A modo de ejemplo, las aplicaciones científicas de gran tamaño son normalmente particionadas en un gran número de tareas (del orden de 10^4 tareas) para ser ejecutadas en un *cluster* con un gran número de procesadores (del orden de 10^2 procesadores) (Kacsuk et al., 2007). Tomando como número de tareas $v = 10^4$ y como número de procesadores $p = 10^2$ se puede estimar que el tamaño del espacio de búsqueda es $(v + p - 1)! = (10099)! \approx 4{,}66 \times 10^{36055}$. El tiempo que tomaría revisar todas esas potenciales soluciones hace que utilizar una técnica completa sea inviable (Schutten, 1998).

Cuando no es necesario contar con el óptimo global y se puede conformar con un valor optimal (cercano al óptimo) se suelen utilizar técnicas de búsqueda incompleta (McKay, Pinedo y Webster, 2002). Las técnicas incompletas, sacrifican el objetivo de buscar el óptimo a cambio de obtener buenos tiempos de ejecución. Esto se logra a través de una búsqueda parcial y dirigida en el espacio de búsqueda. Así se obtienen soluciones optimales de forma rápida y eficiente. Los mecanismos más usados son: Reglas de Prioridad (Panwalker y Iskander, 1977), Técnicas de satisfacción de restricciones (Cheng y Smith, 1997), Técnicas de Inteligencia Artificial (Agarwal et al., 2003), Multi-agentes (Rojas, 2011), Métodos de Búsqueda Local y Metaheurísticas. Dentro de esta última categoría se encuentran métodos como: *Simulated Annealing* (Perregaard, 1995), Búsqueda Tabu (Geyik y Cedimoglu, 2004), y Algoritmos Genéticos (Solar et al., 2002).

El problema que se quiere resolver es encontrar un scheduling cercano al óptimo en la asignación de las v tareas a ejecutar, sobre un conjunto de p procesadores homogéneos disponibles en un *cluster*. Como resultado, se espera obtener una cola de tareas para cada procesador del *cluster* que minimice el tiempo de ejecución global (ejecución de todas las tareas).

El objetivo de este trabajo es diseñar una solución basada en un algoritmo evolutivo para obtener una planificación optimal de forma eficiente (en un tiempo prudente). Para lograr este objetivo se presenta un Algoritmo Genético (AG) para ser ejecutado en un ambiente paralelo y distribuido con el fin de disminuir los tiempos de ejecución y obtener soluciones de buena calidad para problemas de gran tamaño. Para evaluar el desempeño del AG paralelo (AGP) y considerarlo como una alternativa viable para crear planificaciones en la realidad, se hacen pruebas con problemas de hasta $v = 10000$ tareas, y se analiza el tiempo en obtener las soluciones.

En la sección 2 se explican los conceptos asociados al problema de scheduling y al modelamiento a través de grafos para representar las tareas de una aplicación que se quieren ejecutar en los procesadores de un *cluster*. Luego, en la sección 3 se muestra el diseño del AGP propuesto para resolver el problema de scheduling en *clusters* de computadores. En la sección 4 se muestra la implementación del modelo, cuyos resultados y análisis se presentan en la sección 5 para verificar el comportamiento del AGP y medir su desempeño. Finalmente, se entregan las conclusiones obtenidas en el trabajo.

2. Marco Teórico del Problema de Scheduling

El problema de planeación del orden de ejecución de las tareas dentro de un sistema computacional se define como: existe un número limitado v de tareas numeradas de modo que t_i es el tiempo que toma la i-ésima tarea en ser ejecutada, las cuales deben ser ejecutas en p procesadores (donde p_j es el j-ésimo procesador). El objetivo es buscar la programación de ejecución óptima para estos conjuntos de tareas y procesadores (Pinedo, 2008).

El problema de scheduling tiene diferentes instancias, que dependen de los siguientes parámetros:

Número de procesadores (p): El número de unidades de procesamiento que hay disponibles para asignar. La calidad de las soluciones en términos de tiempo de computación es dependiente de este parámetro ya que restringe la paralelización de las tareas. El número de unidades de procesamiento puede ser: (*i*) Limitado, porque se dispone de un conjunto finito de procesadores para la planificación; o (*ii*) Ilimitado (teórico).

Homogeneidad de la capacidad de cómputo: Si dos unidades de procesamiento p_j y p_k con diferentes capacidades de cálculo ejecutan una misma tarea t_i, el tiempo requerido en cada procesador es diferente. Las unidades de procesamiento pueden ser: (*i*) Idénticas (homogéneas); o (*ii*) Heterogéneas.

Duración de las tareas (t_i): Es el tiempo que necesita una tarea t_i en ser ejecutada en el procesador p_j (procesadores homogéneos). La duración de las tareas es: (*i*) Igual para todas las tareas ($t_i = t_j \ \forall i, j$); o (*ii*) Arbitrario, y por ende cada tarea tiene una duración diferente.

Dependencia entre tareas: Una tarea t_i puede requerir resultados de las tareas t_1, t_2, ... t_k (con $k < i$) para comenzar su ejecución. Esta relación de dependencia define condiciones para el problema. Existen dos posibilidades: (*i*) Las tareas pueden depender de otras tareas. (*ii*) No existe dependencia entre las tareas.

Costos de comunicación (c_{ij}): Para comenzar la ejecución de la tarea t_k se deben transferir o copiar los resultados obtenidos de las tareas de las cuales t_k depende. Se representa el tiempo de comunicación entre las tareas respectivas, como c_{ij} donde i representa la tarea de origen y j la tarea de destino. Existen tres posibilidades: (*i*) Se ignoran los costos de comunicación asociados o no se consideran. (*ii*) Existen costos de comunicación y son todos iguales. (*iii*) Existen costos arbitrarios de comunicación entre las tareas.

Tipo de comunicación: Consiste de dos variables que dependen de la configuración del sistema en el cual está construido: La forma de transferencia de datos; y cuándo se realiza la comunicación. La forma de transferencia se refiere a cómo son transferidos los datos desde un nodo a otro y puede tener una de las siguientes formas: (*i*) Secuencial, es decir, se realizan las transferencias una tras otra. Este método toma más tiempo ya que el costo de la comunicación corresponde a la suma de los costos de las dependencias de cada tarea. (*ii*) Paralela, es decir, se realizan todas las transferencias a la vez. El costo que debe asumir la tarea corresponde al máximo costo de transferencia ya que los demás se realizarán simultáneamente y duran menos.

Por otro lado, el comienzo de la comunicación se refiere a cuando se inicia la transferencia de datos y las posibilidades son: (*i*) Inmediato, es decir, cuando termina la ejecución de la dependencia, ésta transfiere los resultados a las tareas que los requieren. (*ii*) Diferido, es decir, cuando se inicia la ejecución de una tarea que requiere resultados hace una *petición* de los datos y se realiza la transferencia.

El problema de scheduling en un *cluster* corresponde a una instancia particular

de los problemas de scheduling, donde las condiciones del ambiente en que se ejecutan las tareas y la programación resultante la definen. Los *clusters* permiten garantizar que:

- Se dispone de un número limitado de p unidades homogéneas de procesamiento, que se comunican a través de una red de interconexión con un costo arbitrario de comunicación.
- La transmisión de datos es diferida y secuencial, cuando la tarea t_i comienza su ejecución, hace una petición de los datos que necesita, y espera a que todas las transferencias se completen de forma secuencial (tiempo equivalente a $\sum c_{ij}$ sin considerar las tareas que son ejecutadas en el mismo procesador).
- Las aplicaciones científicas que corren en un *cluster* tienen típicamente tareas de duración arbitraria, con cierto grado de dependencia entre ellas.

2.1. DAGs para representar el Problema de Scheduling

Los grafos dirigidos acíclicos o DAG (por su sigla en inglés) se utilizan en la planeación de asignación ya que permite representar las dependencias entre elementos de modo simple y directo. En el caso del scheduling se puede asociar los tiempos de cada tareas t_i a cada vértice v_i y los costos de comunicación c_{ij} a las aristas e_{ij} del DAG.

Inspirado en la idea de agrupación de tareas según sus dependencias de Abido y Elazouni (2009), se creó una medida de profundidad llamada *PROF*, la cual agrupa las tareas según sus dependencias. Las resume en un solo número que define un orden dentro del DAG (Morales, 2010). Así, una tarea con *PROF* 4 no puede ser ejecutada antes que una tarea con *PROF* 2 debido a que el grupo *PROF* 4 depende del grupo *PROF* 2.

El cálculo de *PROF* se basa en un algoritmo iterativo que va marcando las tareas del DAG como procesadas. El algoritmo en cada iteración busca las tareas que tienen sus dependencias cumplidas y que están "listas" para ser ejecutadas. Una vez localizadas les asocia el *PROF* correspondiente a la iteración actual y las marca como procesadas, repitiendo el proceso hasta marcar todas las tareas.

En la primera iteración de la Figura 1, la tarea t_1 no tiene dependencias y es marcada como lista asociándole un *PROF*=1. En la siguiente iteración (*PROF*=2), las tareas t_2, t_3 y t_4 podrían ser procesadas, ya que su única dependencia (t_1) está marcada. Éstas son marcadas como listas, asociándole un *PROF*=2. Así sucesivamente hasta que todas las tareas han sido marcadas.

Los valores de *PROF* obtenidos se asocian a dos arreglos: El primero mantiene ordenadas las tareas según su *PROF*; el segundo mantiene el *PROF* asociado a cada tarea. Si se accede al arreglo con el índice i se obtiene el valor del *PROF* de la tarea t_i en tiempo constante, acelerando el proceso de búsqueda ya que no requiere revisar las dependencias.

La *Ruta Crítica* (*CP* de sus siglas en inglés) de un DAG determina el camino de dependencias de mayor duración dentro del DAG. Esta ruta define el tiempo mínimo que se puede lograr en una planificación de tareas. En el cálculo de *CP* se ignora los costos de comunicación ya que todas las tareas en *CP* son asignadas sobre el mismo procesador.

Para obtener *CP* se calcula el valor de cada camino hasta cada tarea, sumando los tiempos de las tareas que componen el camino de dependencia. Una vez calculados los caminos posibles hasta una tarea, se toma el mayor y se asigna a la tarea como el tiempo crítico sumando su tiempo respectivo.

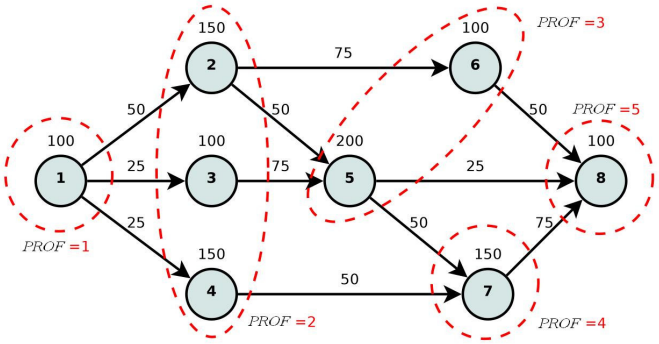

Figura 1: Caracterización de tareas mediante *PROF*

En el caso de un DAG con una gran cantidad de tareas para asignar en pocos procesadores, la *CP* es una medida optimista difícil de lograr. Debido a esto, se utiliza el *Tiempo Paralelo Ideal* (*TPI*), que es el tiempo promedio que usaría cada procesador incluyendo los costos de comunicación. Se define *TPI* como la suma de los tiempos de todas las tareas (t_i) y los costos de comunicación c_{ij} de las dependencias entre tareas, divididas por el número p de procesadores de modo de tener un tiempo estimado de la paralelización perfecta del DAG (Ec. 1).

$$TPI = \frac{\sum_{i=1}^{v} t_i + \sum_{i=1}^{v}\sum_{j=1}^{v} c_{ij}(1 - 1/p)}{p} \qquad \text{(Ec.1)}$$

TPI no necesariamente corresponde al óptimo real, ya que la topología del DAG puede cambiar la solución sin importar de cuántos procesadores se disponga. Es útil en condiciones ideales, i.e., tareas con tiempos iguales, número de tareas múltiplo del número de procesadores disponibles, etc.

El cálculo de *PROF*, *CP* y *TPI* se realiza en el momento en que se carga el DAG del problema en memoria, y se computa una única vez, pero sus valores son utilizados permanentemente para obtener el scheduling optimal.

Considerando que los indicadores *CP* y *TPI* no tienen una correlación entre ellos, se utiliza el mayor de ellos, ya que ambos son cota inferior para el tiempo total de ejecución y su valor depende de diversos parámetros (topología del DAG, procesadores disponibles, duración de las tareas, costos de comunicación, etc.)

3. Solución Propuesta: Algoritmo Genético Paralelo (AGP)

Un algoritmo evolutivo es un proceso estocástico iterativo que opera sobre una población en la que cada individuo (cromosoma) representa una posible solución al problema objetivo. Mediante una función de *Fitness* se evalúa cada individuo para determinar cuantitativamente cuán apto es para resolver el problema en cuestión (Sivanandam y Deepa, 2008). Dentro de los algoritmos evolutivos, los Algoritmos Genéticos (AG) utilizan operadores de Selección, Cruzamiento y Mutación, creando generaciones de mejores individuos hasta alcanzar un criterio de detención (Vose, 1999).

Un AG estándar representa sus individuos en forma de *string* binario

(cromosoma) y sus operadores son genéricos, por lo que podría resolver cualquier problema representable en forma binaria. Sin embargo, el desempeño se puede ver afectado ya que dependiendo del problema, la cantidad de soluciones no factibles puede ser demasiado alta. Por este motivo, es común crear implementaciones de AGs que utilizan una representación específica al problema y movimientos que optimizan el comportamiento de modo de mejorar el desempeño y obtener mejores soluciones más rápidamente.

Para crear un AG, se requiere definir una representación del problema (cromosoma), los operadores genéticos de Selección, Cruzamiento y Mutación; y una función de evaluación de la aptitud (*Fitness*).

La convergencia del AG depende de sus operadores genéticos. Por ejemplo, en el problema de scheduling se tiene un gran número de restricciones variables que dependen de la instancia del problema, por lo que generar en forma aleatoria un gran número de soluciones puede resultar en una pérdida de recursos al generar soluciones no factibles. La solución propuesta es un AG con operadores adaptados al problema para guiar la búsqueda de forma que en cada generación la mejor solución converja al óptimo (de modo de no perder tiempo de cómputo en el cálculo de soluciones no factibles) (Abido y Elazouni, 2009).

3.1. Paralelización del Algoritmo Genético (AGP)

Los Algoritmos Genéticos Paralelos (AGP) son clasificados por Cantu-Paz (1995) en 4 categorías dependiendo de las características en el método utilizado para la paralelización como de la granularidad resultante. El AGP propuesto es una variante del modelo de Esquema de Concurrencia Síncrona de Redes descrito en Solar et al. (2002). En este AGP propuesto existen N nodos que cumplen el rol de AGs calculadores. De éstos, $N-1$ nodos son AGs esclavos que esperan instrucciones del nodo AG maestro que coordina todo el proceso. Las comunicaciones están restringidas a Maestro → Esclavo y Esclavo → Maestro.

Se define el concepto de tamaño de la población global como P_G, la cual se divide en N poblaciones locales de tamaño P_L de modo que $P_G = \sum P_L$. En cada una de las particiones de la población global (poblaciones locales P_L) hay un AG que trabaja de forma aislada e independientemente. Un operador de migración se encarga de mantener la cooperación entre nodos y así mantener la población global conectada. Este operador introduce parámetros como la tasa de migración y la frecuencia de migración. Según esos parámetros se define el nivel de cooperación que existe entre las poblaciones locales y cuán distante está el comportamiento de las soluciones que tendrían N AG ejecutados secuencialmente.

Representación

Dado que para cada paso de la ejecución del AG, el cromosoma debe ser decodificado para su utilización, se busca una representación que permita utilizar los operadores sin perder tiempo de cómputo.

La representación de una solución para el problema de scheduling es un arreglo, que al recorrerlo de izquierda a derecha, un número negativo representa el comienzo de la cola de ejecución del procesador correspondiente al valor absoluto de ese número y un número positivo representa la tarea i dentro de la cola de ejecución del procesador en el orden que aparece en la cola.

En la Figura 2(a) se muestra un ejemplo con $v = 8$ tareas (del 1 al 8) y $p = 4$ procesadores (-1 al -4). Cada cola en la Figura 2(b) corresponde al orden de tareas

a ejecutar en el procesador respectivo. La ejecución se realiza en cuanto es posible (ya sea se desocupe el procesador o se cumplan las dependencias de la tarea).

-1	2	3	6	-2	5	1	-3	4	8	-4	7

(a) Representación

Cola 1: 2 3 6
Cola 2: 5 1
Cola 3: 4 8
Cola 4: 7

(b) Solución

Figura 2: Ejemplo de la representación de una solución

Esta representación admite cualquier movimiento. Sin importar que modificación se haga, siempre será una solución válida (a excepción del movimiento del procesador 1, ya que no se puede comenzar el arreglo con un valor mayor que cero).

La representación no es dependiente del tiempo, por lo que los operadores permiten ahorrar tiempo de cómputo y destinarlo a la búsqueda en sí.

La operación de "Inserción de tareas en una cola" es una operación específica de la solución propuesta y que se utiliza constantemente en los operadores genéticos. El procedimiento determina la zona donde se puede insertar la tarea sin violar las dependencias definidas en el DAG. Esta zona está definida según el *PROF* de la tarea que será insertada. Antes de insertar una tarea en la cola de un procesador, se revisa las dependencias (los *PROF*) de modo de no crear un *deadlock*. Este procedimiento de revisión permite determinar las posiciones posibles en las cuales la tarea puede ser insertada.

Una vez determinadas las posibles posiciones se elije una de forma aleatoria (se podría buscar la posición donde se minimice la espera, sin embargo, esta evaluación agregaría complejidad al algoritmo y no necesariamente daría mejores resultados) y se realiza la inserción dentro del cromosoma (Figura 3).

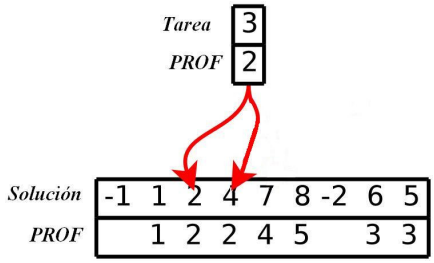

Figura 3: Búsqueda de posibles posiciones

Operadores Genéticos

Los operadores de un AG definen su comportamiento en términos de exploración o de explotación del espacio de búsqueda y por ende definen su desempeño. Para lograr un buen desempeño en el AG se requiere que los operadores cumplan determinadas características, como son: rápida ejecución, transmisión de alguna característica de los padres en caso del cruzamiento, bajo impacto en la mutación, etc.

Debido a que se buscan operadores que no generen soluciones inválidas y que se ejecuten rápidamente, los operadores son especialmente diseñados para mantener las precondiciones de cada tarea.

Selección de individuos

La selección de individuos como un paso importante para la convergencia del AG, debe dar énfasis en la supervivencia de los individuos mejor evaluados (y así garantizar la convergencia hacia las soluciones con mejor *Fitness*). A pesar de esto, no se debe anular la posibilidad de que individuos mal evaluados sobrevivan, ya que la diversidad en el material genético es lo que permite la mejor adaptación al ambiente (facilita la búsqueda del óptimo global y evita estancarse en óptimos locales generando saltos a otras áreas del espacio de búsqueda).

Debido a lo anterior, no se puede seleccionar sólo la parte superior de la población locales (i.e., los individuos mejor evaluados), si no que, hay que dar una posibilidad de sobrevivir a los individuos de "peor calidad". Para lograr este objetivo, cada individuo pasa por una prueba aleatoria donde se determina si sobrevive o no a esa generación. Esta prueba aleatoria asigna a cada individuo de la población local una probabilidad de sobrevivir según la Ecuación 2 (Figura 4), que trabaja sobre una lista ordenada de individuos (de mejor a peor), donde x corresponde a la posición del individuo en la lista.

$$y = \frac{P_1 - P_0}{P_L + 1} x + P_0 \qquad \text{(Ec. 2)}$$

Los valores de P_0 y P_1 son calculados según el porcentaje deseado de supervivencia de la población (E) y se calculan de acuerdo a la Ecuación 3.

$$P_0 = \begin{cases} 2E & si \quad E \le 0{,}5 \\ 1 & si \quad E > 0{,}5 \end{cases}, P_1 = \begin{cases} 0 & si \quad E \le 0{,}5 \\ 2E-1 & si \quad E > 0{,}5 \end{cases} \qquad \text{(Ec.3)}$$

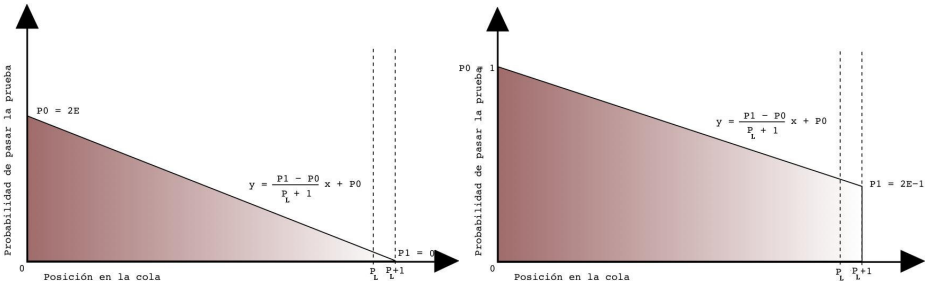

Figura 4: Probabilidad de sobrevivir de cada individuo en una población local

Operador de cruzamiento

El operador de cruzamiento retiene la característica de la asignación de una tarea t_i en un procesador p_j. La selección de la procedencia de cada gen es determinada aleatoriamente con igual probabilidad para cada padre. Cada gen seleccionado es puesto en el nuevo cromosoma en el procesador que tenía asignado inicialmente en su padre, una vez determinada la cola de destino se inserta de forma aleatoria en el rango de posiciones posibles para su inserción.

Operador de mutación

El operador de mutación evita que el algoritmo se atasque en óptimos locales, ya que al mutar un individuo se introducen características que pueden no estar presentes en la población, aumentando la diversidad de material genético en etapas posteriores del proceso de evolución. Para ello, el operador realiza un cambio de tareas tomando dos tareas al azar (de procesadores diferentes) y realiza un

intercambio en la solución. Esta operación puede producir un *deadlock* en la solución y por lo tanto debe revisar las dependencias de las tareas movidas utilizando *PROF*. El proceso se resume a: seleccionar dos tareas, extraerlas del cromosoma recordando su procesador de origen e insertar cada una en el procesador de la otra.

Operador de migración

El operador de migración determina la forma de cooperación entre las poblaciones locales. De no existir, el funcionamiento del AGP sería igual a repetir *N* veces un AG secuencial, eliminando así los beneficios del trabajo distribuido y cooperativo.

El operador de migración definido envía al término de cada generación, el mejor individuo al nodo maestro. Este reúne todos los mejores locales y determina el mejor global. Ese mejor global es enviado a todos los AG esclavos (y a sí mismo) para que lo incorporen en sus poblaciones locales. Cada AG toma su peor individuo y lo reemplaza por el mejor global recibido. Una vez realizada esta operación, el algoritmo continúa con su siguiente generación.

El operador de migración introduce la idea de topología y consecuentemente el concepto de vecindad, donde las comunicaciones se realizan entre vecinos. En el modelo AGP propuesto el maestro es vecino de todos los AGs esclavos, sin embargo éstos no son vecinos entre sí. De esta forma el AGP no depende del número de nodos, lo que resulta en escalabilidad al momento de añadir nodos al proceso de búsqueda (Solar et al., 2002).

Este operador produce una barrera al hacer esperar que todos los AGs envíen sus mejores locales. Al ejecutar todos los nodos el mismo AG sobre poblaciones de igual tamaño, el tiempo perdido no degrada el desempeño (Pinacho et al., 2004).

Función de evaluación (*Fitness*)

La función de evaluación calcula el tiempo estimado que tomaría la solución en ser ejecutada. En caso de determinar que la solución es no válida, la función entrega el valor máximo. Sin embargo, dada las características de los operadores genéticos ninguna solución no viable debería llegar a ser evaluada.

El cálculo del *Fitness* es un proceso iterativo que calcula el momento en que cada tarea termina su ejecución. La tarea con el mayor tiempo es el *Fitness*. El proceso comienza ordenando las tareas por *PROF* en cada cola de forma de que cada tarea es calculada cuando sus dependencias (incluyendo su predecesora en la cola del procesador) están listas. Después se calcula para cada tarea en la cola ordenada, el tiempo de inicio, revisando las dependencias, buscando entre ellas el mayor tiempo de término. En caso de no haber dependencias, el tiempo de inicio es cero.

Al tiempo de inicio se suma la duración de la tarea y la suma de los costos de sus dependencias y es asignado al tiempo de término de la tarea. Una vez calculados todos los tiempos de término se busca la que tiene mayor valor, y se asigna al *Fitness* del individuo, que es equivalente al término de ejecución global.

Para saber si una solución es de buena *Calidad* se compara el *Fitness* con el valor óptimo ideal. Se considera como valor óptimo ideal la función *max*(*CP*, *TPI*), obteniendo la cota inferior para el tiempo de una solución y se calcula una razón de *Calidad* = *Fitness*/*max*(*CP*, *TPI*), para determinar si una solución es buena o no.

4. Implementación

La implementación del AGP se realizó en el *cluster* de la Universidad Técnica Federico Santa María (UTFSM) que es parte de la Grid EELA (*E-science grid facility for Europe and Latin America*). Este *cluster* tiene $n = 13$ nodos con 52 procesadores disponibles para cómputo. De éstos, $n = 3$ nodos están disponibles para utilizarlos con *Message Passing Interface* (MPI), con $ppn = 8$ procesadores disponibles por nodo se tiene en total un máximo de $N = 24$ procesadores para ejecutar los esclavos del AGP y realizar cálculos de forma distribuida.

El *cluster* trabaja con *Linux*, utiliza *Torque PBS* como *resource manager* y *Maui scheduler* como scheduler de tareas.

Todos los nodos esclavos del modelo AGP ejecutan el mismo AG, siendo que el maestro tiene rol de nodo de cómputo y controlador. La función del maestro es:

1. **Enviar semillas** al motor de números seudo-aleatorios de cada AG esclavo, eliminando la dependencia del tiempo de inicio (dos nodos con la misma semilla se comportan idénticamente).
2. **Iniciar proceso de evolución** en sí mismo (maestro) y enviar señal de control a los AG esclavos para iniciar proceso de evolución en la población local (P_L).
3. **Recopilar mejores locales** al término de cada generación desde cada uno de los nodos AG esclavos (inclusive de sí mismo).
4. **Buscar mejor global** dentro de individuos recibidos comparando sus *Fitness*.
5. **Enviar mejor global** a cada nodo AG esclavo para que lo agreguen a sus poblaciones locales.
6. **Finalizar ejecución** de nodos AG esclavos al cumplir condición de término.
7. **Concluir e informar** la solución encontrada en el proceso cooperativo.

Las funciones 1, 6 y 7 se ejecutan sólo una vez, las funciones 2, 3, 4 y 5 se ejecutan repetitivamente dando como resultado el cuerpo del AG.

Los nodos AG esclavos ejecutan las siguientes operaciones coordinadas por el nodo maestro:

1. **Evolución**: Lleva la población local a la generación siguiente.
2. **Sumariar**: Completa el sumario local con los datos que tiene hasta el momento y lo envía al nodo maestro.
3. **Finalizar**: Finaliza el proceso del nodo y libera todos los recursos tomados.

Con estas tres operaciones, el nodo maestro controla el funcionamiento de la aplicación global y dirige a los nodos esclavos según los resultados que se obtengan. Estas operaciones son las que definen el comportamiento del AGP, haciendo converger los procesos aleatorios hacia la solución óptima.

El mismo código del proceso de **Evolución** se ejecuta en todos los nodos (maestro y esclavos) para pasar de la generación G a la generación $G + 1$ (con P_L como tamaño de la población local):

1. Aplica **operador de selección** sobre P_L con la probabilidad de supervivencia de las Ecs. 2 y 3, y almacena en una lista *padres* los individuos que sobrevivieron.
2. Aplica **operador de cruzamiento** con probabilidad p_c sobre la lista *padres* para obtener los nuevos individuos que se almacenan en una lista de *descendientes*.
3. Aplica **operador de mutación** con probabilidad p_m sobre la lista *descendientes* que son mutados y mantenidos en la lista *descendientes*.
4. Une de forma ordenada las listas *padres* y *descendientes* en una única lista que pasa a ser la **nueva población** local.
5. Si es nodo esclavo envía el **mejor individuo** de esa P_L al maestro y espera que éste le envíe el mejor global. Si es el nodo maestro recibe los individuos, selecciona el mejor y lo envía a los esclavos como el mejor global.

Afinamiento de parámetros
El proceso de afinamiento consiste en realizar pruebas variando los parámetros para comparar y buscar la configuración que da los mejores resultados con un conjunto de problemas de características similares. Las pruebas se realizaron con un conjunto de DAGs de v = 1000 tareas. Los resultados obtenidos son:

La población local (P_L) del AG debe moverse entre 50 y 100 individuos.

El tamaño de la población global (P_G) es fijado en 800 individuos. El tamaño máximo de P_L es 100 con N = 8 AG (7 esclavos + 1 maestro).

El **número máximo de Generaciones** (G) se deja en G = 1000, dado que en todas las ejecuciones se encontró la solución optimal antes de 1000 generaciones.

El **número máximo de repeticiones** es 100 (10% de G), valor suficientemente alto que asegura una baja probabilidad de encontrar una mejor solución.

La **probabilidad de cruzamiento** (p_c) se recomienda que varíe entre 35% y 65%. El valor 50% demostró mantener suficientes individuos en la población para mantener diversidad de material genético y a la vez una tasa de renovación alta, obteniendo tanto buena calidad de la solución como velocidad en la búsqueda.

La **probabilidad de mutación** (p_m) se eligió el máximo recomendado en la bibliografía (entre 1% y 5%), debido que al tener N = 24 AG, la P_L disminuye de tamaño estancando al algoritmo en óptimos locales.

El parámetro ***Walltime*** fue fijado en 24 horas, debido a que el máximo registrado en las pruebas fue de casi 23 horas con N = 8 AG al asignar un DAG con v = 10000 nodos sobre p = 64 procesadores, y un máximo de repeticiones de 10%.

5. Resultados y Análisis

Se utilizaron 15 DAGs que reúnen las características de 15 problemas diferentes. Estos 15 DAGs son procesados para generar 45 pruebas a ser ejecutadas en el *cluster*. Estas 45 pruebas toman alrededor de 7 días en ser ejecutadas para obtener los resultados. En la Tabla 1 se presenta el detalle de los experimentos ejecutados, donde se muestra en sus columnas los siguientes datos:

- ***Exp***: Identificador del experimento.
- ***N*:** Número de AG paralelos con N = 8, 16 y 24 procesadores (un maestro y el resto AG esclavos). En un *cluster* este parámetro está compuesto por dos valores: n que corresponde al número de nodos reservados del *cluster* y *ppn* que corresponde al número de núcleos reservados por cada nodo. El número de procesadores (N) disponibles para ejecutar el AGP es el producto de estos valores. Las configuraciones elegidas son el uso de 1, 2 y 3 nodos completos (***N* = 8** = 1n 8*ppn*; ***N* = 16** = 2n 8*ppn* y ***N* = 24** = 3n 8*ppn*).
- ***P*$_L$:** Tamaño de la población local para cada uno de los casos. El tamaño de la población global se obtiene como: P_G = (P_L=100) (N=8) = 800; P_G = (P_L=50) (N=16) = 800; P_G = (P_L=34) (N=24) = 816.
- ***p*:** Número de procesadores sobre los cuales se asignan las v tareas de los DAGs de la Tabla 1. Las tareas de cada uno de los 15 DAG iniciales son asignadas a p=16, 32 y 64 procesadores, dando origen a los 45 experimentos.
- ***v*:** Número de tareas del DAG con valores v = 1000, 2500, 5000, 7500 y 10000, que son representativos de un problema real.
- ***t*$_{STotal}$**: Tiempo total de computación de las v tareas en un único procesador.
- ***t*$_{comm}$**: Tiempo total de comunicación entre tareas que se obtiene sumando todos los costos en las aristas del DAG.
- ***TPI, CP***: *Tiempo Paralelo Ideal* (Ec. 1) y ruta crítica del DAG.

Tabla 1: Descripción de los 45 experimentos realizados

	Parámetros del AGP		Características del Problema					
Exp	***N***	***P_L***	***p***	***v***	***t_{STotal}***	***t_{comm}***	***TPI***	***CP***
1	8	100	16	1000	502578	460007	58364	12670
2	16	50	16	1000	502578	460007	58364	12670
3	24	34	16	1000	502578	460007	58364	12670
4	8	100	32	1000	499136	473702	29938	13273
5	16	50	32	1000	499136	473702	29938	13273
6	24	34	32	1000	499136	473702	29938	13273
7	8	100	64	1000	501043	454723	14822	13092
8	16	50	64	1000	501043	454723	14822	13092
9	24	34	64	1000	501043	454723	14822	13092
10	8	100	16	2500	1245112	3007986	254068	30565
11	16	50	16	2500	1245112	3007986	254068	30565
12	24	34	16	2500	1245112	3007986	254068	30565
13	8	100	32	2500	1259988	3021095	130833	28995
14	16	50	32	2500	1259988	3021095	130833	28995
15	24	34	32	2500	1259988	3021095	130833	28995
16	8	100	64	2500	1248550	2978842	65325	27739
17	16	50	64	2500	1248550	2978842	65325	27739
18	24	34	64	2500	1248550	2978842	65325	27739
19	8	100	16	5000	2511201	12207849	872253	57576
20	16	50	16	5000	2511201	12207849	872253	57576
21	24	34	16	5000	2511201	12207849	872253	57576
22	8	100	32	5000	2508238	12333144	451749	61090
23	16	50	32	5000	2508238	12333144	451749	61090
24	24	34	32	5000	2508238	12333144	451749	61090
25	8	100	64	5000	2495348	12240308	227256	61806
26	16	50	64	5000	2495348	12240308	227256	61806
27	24	34	64	5000	2495348	12240308	227256	61806
28	8	100	16	7500	3745150	27766498	1861015	85193
29	16	50	16	7500	3745150	27766498	1861015	85193
30	24	34	16	7500	3745150	27766498	1861015	85193
31	8	100	32	7500	3747227	27889554	961413	84550
32	16	50	32	7500	3747227	27889554	961413	84550
33	24	34	32	7500	3747227	27889554	961413	84550
34	8	100	64	7500	3740687	27707279	484610	83991
35	16	50	64	7500	3740687	27707279	484610	83991
36	24	34	64	7500	3740687	27707279	484610	83991
37	8	100	16	10000	5013751	49512957	3214509	114411
38	16	50	16	10000	5013751	49512957	3214509	114411
39	24	34	16	10000	5013751	49512957	3214509	114411
40	8	100	32	10000	4995420	49686065	1660274	112936
41	16	50	32	10000	4995420	49686065	1660274	112936
42	24	34	32	10000	4995420	49686065	1660274	112936
43	8	100	64	10000	4998774	49503017	839504	112883
44	16	50	64	10000	4998774	49503017	839504	112883
45	24	34	64	10000	4998774	49503017	839504	112883

Los resultados obtenidos de la ejecución de los experimentos se resumen en dos grupos: los datos de la solución del problema de scheduling del DAG y los datos de la ejecución del AGP. Los datos de la solución del problema, detallados en las columnas de la Tabla 2, son:

- ***Fitness***: Valor de evaluación de la solución del DAG.
- ***$Calidad_{Ideal}$***: Razón *Fitness*/*OptimoIdeal*, con *OptimoIdeal* = *max*(*TPI*, *CP*).
- ***$Calidad_{Obtenida}$***: Razón entre solución obtenida y t_{STotal} (*Fitness*/t_{STotal}).
- ***N_{comm}***: Valor compuesto (número y porcentaje) de las comunicaciones anuladas en la solución (tareas asignadas al mismo procesador).
- ***T_{comm}***: Valor compuesto (tiempo y porcentaje) de las comunicaciones anuladas.

Tabla 2: Resultados de la solución al problema de scheduling

	Fitness			**Comunicaciones**			
		Calidad		N_{comm}		T_{comm}	
Exp	***Fitness***	***Ideal***	***Obtenida***	***Núm.***	***%***	***tiempo***	***%***
1	63534	1,09	12,64%	269	5,86%	27102	5,89%
2	64246	1,1	12,78%	336	7,32%	33470	7,28%
3	66383	1,14	13,21%	300	6,53%	30258	6,58%
4	36811	1,23	7,37%	165	3,52%	16241	3,43%
5	39986	1,34	8,01%	132	2,81%	13334	2,81%
6	40203	1,34	8,05%	147	3,13%	15033	3,17%
7	24175	1,63	4,82%	76	1,66%	7610	1,67%
8	27256	1,84	5,44%	85	1,86%	8785	1,93%
9	28435	1,92	5,68%	60	1,31%	6495	1,43%
10	275699	1,09	22,14%	1924	6,42%	193096	6,42%
11	289166	1,14	23,22%	1919	6,40%	194664	6,47%
12	302361	1,19	24,28%	1901	6,34%	192365	6,40%
13	160311	1,23	12,72%	927	3,07%	92634	3,07%
14	174835	1,34	13,88%	944	3,13%	93915	3,11%
15	184836	1,41	14,67%	936	3,10%	93908	3,11%
16	105553	1,62	8,45%	481	1,62%	49022	1,65%
17	120748	1,85	9,67%	442	1,49%	44173	1,48%
18	122702	1,88	9,83%	486	1,64%	48203	1,62%
19	981736	1,13	39,09%	7664	6,28%	767767	6,29%
20	1032011	1,18	41,10%	7708	6,32%	769274	6,30%
21	1043555	1,2	41,56%	7659	6,28%	766488	6,28%
22	610027	1,35	24,32%	3914	3,17%	393763	3,19%
23	692692	1,53	27,62%	3856	3,12%	388862	3,15%
24	696078	1,54	27,75%	3893	3,15%	391307	3,17%
25	395199	1,74	15,84%	1995	1,63%	199670	1,63%
26	445683	1,96	17,86%	1967	1,61%	196460	1,61%
27	464540	2,04	18,62%	1866	1,52%	186158	1,52%
28	2147407	1,15	57,34%	17486	6,29%	1745503	6,29%
29	2228115	1,2	59,49%	17555	6,32%	1757840	6,33%
30	2347443	1,26	62,68%	17413	6,27%	1735385	6,25%
31	1309840	1,36	34,95%	8820	3,16%	882048	3,16%
32	1385008	1,44	36,96%	8725	3,13%	870766	3,12%
33	1429930	1,49	38,16%	8776	3,15%	881214	3,16%
34	891799	1,84	23,84%	4367	1,58%	435315	1,57%
35	970320	2	25,94%	4363	1,57%	438271	1,58%
36	1021737	2,11	27,31%	4336	1,57%	435639	1,57%
37	3706804	1,15	73,93%	31273	6,32%	3127126	6,32%
38	3986749	1,24	79,52%	31320	6,33%	3124862	6,31%
39	4022584	1,25	80,23%	30898	6,24%	3096635	6,25%
40	2385823	1,44	47,76%	15621	3,14%	1558460	3,14%
41	2463221	1,48	49,31%	15867	3,19%	1579333	3,18%
42	2533065	1,53	50,71%	15616	3,14%	1558835	3,14%
43	1520878	1,81	30,43%	7719	1,56%	777219	1,57%
44	1678228	2	33,57%	7563	1,53%	762914	1,54%
45	1764653	2,1	35,30%	7857	1,59%	780782	1,58%

La $Calidad_{Ideal}$ varía entre 1,09 y 2,11, siendo equivalente a 9% y 111% sobre el *ÓptimoIdeal*. Al contrastar la solución con la ejecución en un solo procesador ($Calidad_{Obtenida}$), éstas van de 5% a 80%. Las dos comparaciones son de buena calidad ya que, al no considerar la topología del DAG, tanto *CP* como *TPI* son predicciones optimistas. Por ejemplo, el experimento 36 (Tabla 2) tiene un valor de

2,11, que al compararlo con el tiempo secuencial tiene un valor de 27%, lo que valida la afirmación de que la solución es de buena calidad a pesar de que el valor 2,11 parece alto.

Los datos de la ejecución del AGP con las mediciones realizadas se muestran en las columnas de la Tabla 3.

Tabla 3: Resultados de la ejecución del modelo AGP en el *cluster*

	Generaciones		**Tiempo**				**Soluciones**	
Exp	G_E	G	t_{wall}	t_{CPU}	$\frac{t_{CPU}}{t_{wall}}$	t_G	***Sol***	t_{Sol}
1	485	586	616	281,35	45,67%	0,48	230668	102,48
2	527	628	355	147,88	41,66%	0,24	241764	102,18
3	164	265	104	40,44	38,88%	0,15	99472	102,49
4	859	960	1034	469,24	45,38%	0,49	376788	100,37
5	731	832	481	200,47	41,68%	0,24	320378	99,88
6	210	311	125	48,72	38,98%	0,16	116530	99,66
7	479	580	655	298,1	45,51%	0,51	227819	95,53
8	392	493	298	124,75	41,86%	0,25	189954	95,17
9	246	347	146	56,93	38,99%	0,16	129848	95,03
10	491	592	3535	1705,06	48,23%	2,88	232611	17,05
11	382	483	1499	683,28	45,58%	1,41	185879	17
12	105	206	445	190,36	42,78%	0,92	77406	16,94
13	751	852	5103	2461,27	48,23%	2,89	334164	16,97
14	710	811	2531	1151,16	45,48%	1,42	312050	16,94
15	532	633	1358	582,71	42,91%	0,92	235818	16,86
16	755	856	5233	2526,08	48,27%	2,95	336380	16,65
17	151	252	798	364,74	45,71%	1,45	97092	16,64
18	157	258	561	243,21	43,35%	0,94	96741	16,57
19	861	962	22702	11001,75	48,46%	11,44	377952	4,29
20	594	695	8457	3906,98	46,20%	5,62	268006	4,29
21	232	333	2709	1212,24	44,75%	3,64	124545	4,28
22	639	740	17477	8477,35	48,51%	11,46	290830	4,29
23	627	728	8845	4080,34	46,13%	5,6	280396	4,29
24	120	221	1795	802,55	44,71%	3,63	82690	4,29
25	991	1001	23709	11490,79	48,47%	11,48	392758	4,27
26	519	620	7599	3496,92	46,02%	5,64	238451	4,26
27	388	489	4004	1788,91	44,68%	3,66	182925	4,26
28	500	601	31843	15411,57	48,40%	25,64	235555	1,91
29	405	506	13688	6370,06	46,54%	12,59	194107	1,9
30	284	385	6989	3151,14	45,09%	8,18	143996	1,9
31	618	719	38000	18385,6	48,38%	25,57	282480	1,92
32	270	371	10051	4676,87	46,53%	12,61	143057	1,91
33	487	588	10677	4799,24	44,95%	8,16	220271	1,91
34	940	1001	53022	25705,35	48,48%	25,68	392928	1,91
35	414	515	13974	6498,54	46,50%	12,62	198254	1,91
36	199	300	5478	2458,48	44,88%	8,19	112381	1,9
37	745	846	79563	38561,11	48,47%	45,58	332587	1,08
38	323	424	20408	9515,04	46,62%	22,44	163554	1,07
39	451	552	17877	8026,28	44,90%	14,54	206776	1,07
40	737	838	78466	38002,76	48,43%	45,35	329407	1,08
41	409	510	24467	11369,43	46,47%	22,29	196414	1,08
42	446	547	17597	7910,72	44,95%	14,46	204771	1,08
43	779	880	82281	39861,44	48,45%	45,3	344922	1,08
44	440	541	25932	12074,67	46,56%	22,32	208101	1,08
45	349	450	14488	6533,25	45,09%	14,52	168984	1,08

Los datos presentados en la Tabla 3 son:

- G_E: Número de la generación en la cual se encontró la solución final.
- G: Número total de generaciones ejecutadas.
- t_{wall}: Tiempo de reloj que duró la ejecución en segundos.
- t_{CPU}: Tiempo de ejecución real en segundos (el procesador no estuvo en espera).
- t_G: Tiempo promedio en segundos que tomó cada generación en la ejecución.
- Sol: Número total de soluciones evaluadas.
- t_{Sol}: Promedio del número de soluciones por segundo evaluadas por procesador.

Se observa que la razón de utilización de procesador t_{CPU}/t_{wall} es cercana al 45%, y tiende a disminuir cuando se aumenta el número de AG esclavos del AGP. La relación t_{Sol} es lineal con respecto al número de AG, mostrando un comportamiento del tipo $100\times(1000/v)^2$.

Era esperable que al aumentar el número de procesadores disponibles (p = 16, 32 y 64) para asignar las v tareas de un DAG de mayor tamaño, mejoren las soluciones en calidad considerando los tiempos y no la razón de $Calidad_{Ideal}$.

La media de generaciones utilizadas para encontrar la solución final es G_E=486, con una desviación estándar de 231 por lo que estuvo bien afinado el límite máximo de generaciones (todas las ejecuciones terminaron antes de G = 1000).

6. Conclusiones

En general la utilización de un AG dirigido limita el espacio de búsqueda permitiendo encontrar soluciones en un tiempo prudente. En particular, un AGP cooperativo reduce notablemente el tiempo de ejecución para encontrar una solución optimal. Estas soluciones se encuentran rápidamente, recorriendo una porción reducida del espacio de búsqueda y se acercan al óptimo ideal comprobando su calidad absoluta.

Se comprueba la proporcionalidad inversa entre la razón t_{CPU}/t_{wall} y el número de AG cooperando en la búsqueda, debido al incremento de las comunicaciones al agregar nodos. Sin embargo, los tiempos asociados a la sincronización debido al operador de migración son despreciables en relación a los tiempos de ejecución. La sincronización y la pérdida de tiempo asociada a las comunicaciones no explican completamente las diferencias entre t_{wall} y t_{CPU} ya que la sincronización solo representa cerca de un 1% y la razón de cambio de t_{CPU}/t_{wall} no explica esa cantidad de tiempo perdido.

El número de soluciones que cada AG es capaz de evaluar por segundo se ajusta a $100\times(1000/v)^2$ es específico de este modelo AGP sobre el *cluster* de la UTFSM. Sin embargo, considerando el comportamiento estable del AGP, se podría llegar a una conclusión similar en implementaciones sobre otros *clusters* lo que permite predecir el tiempo máximo que una búsqueda puede durar (en función del número de generaciones máximas y el número de AG cooperativos).

Los *Fitness* obtenidos en la ejecución de un mismo DAG se mantienen en un rango estrecho, lo que permite asegurar que son valores optimales de buena calidad que se encuentran en torno a la configuración óptima para el DAG. Sin embargo, en la medida que se aumenta el número de AG se percibe una ligera tendencia a la baja en la calidad de la solución encontrada (menor *Fitness*). Esto se debe a que al aumentar el número de AG cooperando en la búsqueda, la población local de cada AG se reduce. Con esto, dado que la fórmula de probabilidad de la prueba de supervivencia en la selección es lineal, ésta puede estar asignando probabilidades altas a los mejores individuos disminuyendo la variedad de material genético. Esto

no quiere decir que el tiempo no disminuya o que las soluciones sean de mala calidad, ya que se ve una clara tendencia a mantener un valor de *Fitness* cercano (a las otras ejecuciones) y a disminuir el tiempo t_{CPU} (y consecuentemente el t_{wall}) al aumentar el número de AG.

Con los resultados obtenidos se puede afirmar que el modelo AGP ejecutado en el *cluster* de la UTFSM disminuye los tiempos de ejecución y obtiene soluciones optimales, por lo que puede considerarse como una alternativa viable para crear planificaciones de problemas reales.

El trabajo futuro que se propone es comparar los resultados con otros algoritmos paralelos. Se propone realizar comparaciones de desempeño y de calidad de soluciones con otros métodos de búsqueda incompleta (por ejemplo, Búsqueda Tabú) como fue realizado en Pinacho et al. (2002).

Otro estudio interesante es cuantificar el impacto de la probabilidad lineal del operador de selección usado. Este operador anula de cierta forma la diversidad de material genético en la población y en poblaciones pequeñas. Se propone estudiar este impacto y proponer alternativas que permitan mitigar o anular este problema.

Bibliografía

Abido M.A. y Elazouni A.: Improved Crossover and Mutation Operators for Genetic Algorithm Project Scheduling, CEC'09 *Proc. of the 11th Conf. on Evolutionary Computation*, IEEE CS Press, 2009.

Agarwal A, Pirkul H. y Jacob V.: Augmented neural networks for task scheduling. *European Journal of Operational Research.* 151, pp. 481–502, 2003.

Cantu-Paz E.: A Summary of Research on Parallel Genetic Algorithms, Computer Science Department and The Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, 1995.

Cheng Ch-Ch. y Smith S.: Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research* 70, 327-357, 1997

Geyik F. y Cedimoglu I.: The strategies and parameters of tabu search for job-shop scheduling. *Journal of Intelligent Manufacturing*, 15, 439-448, 2004.

Kacsuk P., Fahringer T. y Németh Z.: *Distributed and Parallel Systems. From Clustering to Grid computing*, NY, USA, Springer, 233p, 2007.

Morales, C.: Algoritmo de Scheduling para un cluster. Trabajo de Memoria para Ingeniería Civil en Informática, Universidad Técnica Federico Santa María, 2010.

McKay K., Pinedo M. y Webster S.: Practice-focused reasearch issues for Scheduling Systems. *Production and Operations Management*. 11(2), 2002.

Panwalker S.S. y Iskander W.A.: A Survey of Scheduling rules. *Operations Research*. 25(1), 64-76, 1977.

Papadimitriou C.: *Computational complexity*, Addison Wesley Pub. Co., USA, 523 p., 1994.

Perregaard M.: Branch-and-bound methods for the Processor Job Shop and Flow Shop scheduling problems, Tesis de Magister, DIKU, November, 1995.

Pinacho P., Solar M., Inostroza M. y Muñoz R.: Using genetic algorithms and tabu search parallel models to solve the scheduling problem, *IFIP International Conference on Artificial Intelligence*, 2004.

Pinedo M.: *Scheduling Theory, Algorithms and Systems*, 3a ed., NY, USA, Springer, 671p, 2008.

Rojas J.: Algoritmo de Scheduling para Grid basado en Multiagentes. Trabajo de Memoria para Ingeniería Civil en Informática, Universidad Técnica Federico Santa María, 2011.

Schutten J.: Practical job shop scheduling, *Annals of Operations Research* 83, pp. 161-177, 1998.

Sivanandam S.N. y Deepa S.N.: *Introduction to Genetic Algorithms*, 1a ed, Berlin, Alemania, Springer, 400p, 2008.

Solar M., Parada V. y Urrutia R.: A parallel genetic algorithm to solve the set-covering problem, *Journal on Computers and Operations Research*, 29, pp. 1221-1235, 2002.

Ullman J.D.: NP-complete scheduling problems. *Journal of Computer and System Sciences*. 10(3), Jun, 1975.

Vose M.D.: *The Simple Genetic Algorithm: Foundations and Theory Complex Adaptive Systems*, USA, MIT Press, 1999.