CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN

APARTADO POSTAL 14-740, AV. IPN 2508, MÉXICO, D.F. CP 07000

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
SECCIÓN COMPUTACIÓN

"REASIGNACIÓN DE TAREAS DE UN SISTEMA MULTIPROCESADOR"

Tesis que presenta el **Ing. Mario Farías Elinos** que para obtener el grado de **Maestro en Ciencias** dentro de la especialidad de **Ingeniería Eléctrica** con opción en **Computación**

Trabajo dirigido por:

Director: **Dr. Guillermo B. Morales Luna** (CINVESTAV, Departamento de Ingeniería Eléctrica, Sección de Computación)

MÉXICO, D.F., 1999

Becario del CONACYT: 72404

En memoria de la Dra. Araceli Sánchez de Corral,
Fundadora del Centro de Investigación de la
Universidad La Salle (CIULSA).

Dedicatorias

A Dios por darme la oportunidad de hacer algo en la vida.

A mi madre, la M. en C. Carmen Martha Elinos-Baez por su gran ejemplo que me ha brindado durante toda una vida.

A mi hermana, la Doctora Médica Luz Ma. Farías Elinos por su gran apoyo.

A mi hermana, la Q.F.B. Martha Farias Elinos y su esposo, el Doctor Médico Carlos Mauricio Laguna Izquierdo por el apoyo que me han brindado en todo momento.

A los pequeñines Mauricio y Martín, por ser como son.

A la Q.F.B. Guadalupe Vértiz Serrano por su apoyo incondicional en la terminación de este proyecto.

Agradecimientos

A CONACyT y al CINVESTAV por su apoyo brindado.

Al Dr. Guillermo Morales-Luna por darme ese conocimiento, y esa brújula para que no se pierda ese conocimiento.

Al M. en C. Hugo González-Hernández por darme la oportunidad de ser parte de él y de su familia.

Al M. en C. Eduardo Goméz-Ramírez por darme esos consejos de hermano en todo momento.

A la Sra. Sofía "sofi" Reza por su constante apoyo durante mi estancia en el CINVESTAV

A mis compañeros de la Maestría, por todos los ratos que pasamos juntos.

A LIDETEA por dejarme ser parte de ellos.

Resumen

En la presente investigación se muestra la forma de resolver el problema de asignación de tareas para sistemas distribuidos. Esto es, se busca la manera de obtener un sistema cuyos procesadores tengan la misma carga de trabajo, y por ende, obtener el mayor rendimiento del sistema. Para ello la obtención de la mejor asignación es un problema cuya complejidad es exponencial y se le se considera un problema NP completo (su solución se obtiene en un tiempo No Polonomial), característica que lo hace apto para ser resuelto por técnicas de optimización o de aproximación.

Para ello el autor utiliza tres algoritmos de optimización: recocido simulado, algoritmo genético y búsqueda local. Así como tres algoritmos heurísticos creados por el autor, todos ellos para buscar la mejor manera de asignar las tareas. Posteriormente se realizan algunas simulaciones y se comentan los resultados obtenidos. Además se adicionan perturbaciones a las simulaciones, tales como la eliminación de un procesador y la creación de tareas adicionales entre otras;. y la forma en que se resuelven estas eventualidades.

Palabras clave: Algoritmo genético, recocido simulado, heurístico, asignación de tareas, sistema distribuido, multiprocesador.

INDICE

Introducción

Uno de los problemas existentes en el ambiente computacional multiprocesador, sobre todo en los sistemas distribuidos, es la forma de obtener el máximo rendimiento del sistema, esto ha involucrado el estudio de los diferentes factores que, de una u otra manera, afectan el rendimiento, tales como: La velocidad de la red, el sistema operativo, los protocolos de comunicación, la velocidad del procesador, la arquitectura de la tarjeta principal, etc.

Esta investigación se ocupó del tratamiento del problema de asignación, es decir, como repartir las tareas involucradas de tal forma que se obtenga como resultado un sistema donde los procesadores tengan la misma carga de trabajo, y por ende un mayor rendimiento. Aunque este problema de asignación ya ha sido estudiado en forma exhaustiva, sólo se han encontrado métodos que se aproximan a la respuesta óptima, entre los que se pueden mencionar están: redes neuronales artificiales, teoría de grafos, Teoría de decisión de Markov, etc.

En este trabajo en particular se presentan variantes de algunos algoritmos y de la misma función objetivo. Estos son:

- Prueba y ensayo de diferentes funciones objetivo.
- Análisis y desarrollo de algoritmos de optimización, tales como: recocido simulado, algoritmo genético y búsqueda local.
- Análisis y desarrollo de algoritmos heurísticos basados en condiciones de equilibrio de carga.
- Perturbaciones del problema consistentes en:
  - Eliminación de un procesador
  - Terminación de alguna tarea

- Adición de un procesador
- Creación de alguna tarea

El objetivo principal es comparar el comportamiento de las variantes de los algoritmos involucrados, principalmente en su rendimiento y el tiempo que consumen para la obtención del resultado, así como su forma de proceder ante las perturbaciones mencionadas anteriormente.

La formas en que se estructura esta investigación consiste de cuatro capítulos organizados de la siguiente forma: En el primer capítulo se presentan los antecedentes del trabajo, así como la forma en que fue creada la función objetivo y sus variantes. En los siguientes dos capítulos se comenta la forma de trabajar de cada uno de los algoritmos utilizados durante el desarrollo de esta investigación, separándose los algoritmos de optimización en el segundo capítulo y los heurísticos en el tercer capítulo.

La idea de trabajar con estos algoritmos es analizar el comportamiento y la eficiencia de cada uno de ellos, tanto para optimizar el problema, como el tiempo ocupado para encontrar la mejor solución; ello se puede observar en el cuatro capítulo. Además, los métodos se comparan ante escenarios perturbados, tales como la adición de un procesador, la terminación de una tarea, etc.

1. Presentación del problema

Dentro de los sistemas multiprocesador, principalmente en los sistemas distribuidos, el problema de asignación consiste en buscar la manera de obtener un sistema cuyos procesadores tengan la misma carga de trabajo, y por ende, obtener el mayor rendimiento del sistema.

1.1 Antecedentes

El avance acelerado de la tecnología ha permitido que los sistemas computacionales distribuidos sean económicamente atractivos para muchas aplicaciones computacionales. Sin embargo, existen algunos problemas abiertos para el desarrollo de los sistemas distribuidos [19,6]. Algunos de los problemas que frenan el uso exhaustivo de los sistema distribuidos son:

- La degradación en el "throughput" (rendimiento real del sistema) causado por el efecto de saturación [7]. El efecto de saturación es causado por tráfico excesivo en la comunicación inducido por la transferencia de datos de una tarea a otra que residen en procesadores separados. El tráfico entre procesos es un factor muy costoso y poco confiable en la conexión de un sistema distribuido [19,11]. Como resultado del efecto de saturación, el incremento constante en el uso de los recursos computaciones podrían, realmente, decrementar el "throughput" del sistema.

- La dificultad en lograr el equilibrio entre los procesadores del sistema. El efecto de saturación puede ser encubierto por la carga de tareas con una alta demanda de comunicación dentro del mismo procesador. Por lo que se genera frecuentemente cargas desequilibradas y sus efectos se ven reflejados en un bajo "throughput" del sistema. Por lo tanto, es importante equilibrar estos dos factores.

- Una diferencia entre los requerimientos demandantes de las aplicaciones y la arquitectura existente de las redes [21]. Los requerimientos de las aplicaciones a menudo no pueden encajar en la arquitectura distribuida [21]. Así mismo, esta diferencia existe entre la arquitectura distribuida y los requerimientos del software y del sistema. De aquí, que un programa sea prácticamente necesario para la asignación de las tareas en un sistema distribuido. Por ejemplo, aunque actualmente el avance tecnológico por parte de las arquitecturas de redes ha sido muy acelerado, los nuevos requerimientos hacen que los sistemas distribuidos no sean muy grandes.

- La dificultad en la verificación de la asignación de cualquier modelo de asignación. Esto se debe a la escasez de datos reales, los resultados de investigación están limitados a los modelos teóricos y matemáticos [19,11] y que entre si son difíciles de comparar [4].

No todos los factores mencionados anteriormente son estudiados en esta tesis, sin embargo se hace una mención para indicar algunos de los problemas existentes actualmente dentro de los sistemas distribuidos.

1.2 Asignación

Uno de los primeros problemas encontrados en la operación de un sistema distribuido es la asignación de las tareas en los procesadores involucrados. Se han estudiado extensamente a los varios tipos de problemas de asignación [3,5]. El problema de asignación ha sido típicamente formulado como un problema de optimización. El modelo deberá describir los atributos del sistema tales como la capacidad de memoria o la velocidad del procesador, y el objetivo es usualmente la minimización de alguna función de costo que varía de acuerdo a una asignación en particular.

Los problemas de asignación se resuelven, por lo general, por un procedimiento de costo eficiente que encuentre la asignación óptima para instancias específicas del problema. Como una regla, los problemas de asignación tienden a ser computacionalmente intensos, es decir, muy tardados cuando se requiere de una respuesta rápida [7,18]. Esto ha generado una búsqueda de la solución al problema de asignación, entre los métodos para buscar la solución están los métodos generales de optimización de funciones o los provenientes de la teoría de gráficas y de la programación entera. La teoría de gráficas representa el problema como un gráfico y entonces utiliza técnicas comunes como el algoritmo de máximo flujo o de corte mínimo para obtener la mejor asignación. El método de la programación entera es el más utilizado para la solución de problemas de asignación. El problema puede formularse como un problema entero, y utilizando técnicas como enumeración implícita o "branch-and-bound" [7,19] puede encontrarse la solución.

Otros métodos utilizados para obtener la asignación óptima son teoría de decisión de Markov [5] y los basados en inteligencia computacional como Redes Neuronales Artificiales [17].

En la búsqueda de la optimización del problema de asignación el objetivo principal es tener un sistema con una carga computacionalmente equilibrada sobre los procesadores involucrados. Supóngase que las tareas han sido asignadas de tal manera que exista un desequilibrio en la utilización de los procesadores, es decir, un procesador está siendo significativamente más utilizado que los demás. Un postulado fundamental del sistema es que cualquier procesador es vulnerable a fallas, pues la posibilidad de que un procesador sobreutilizado experimente una falla representa en sí una debilidad del sistema. Algunos de los peligros asociados con el desequilibrio de cargas en los procesadores del sistema son:

- La falla de un procesador sobreutilizado hace necesario cargar, configurar y reinicializar en los otros procesadores las tareas originalmente asignadas al procesador fallido. El tiempo requerido para la reconfiguración varía aproximadamente con la utilización de reasignación de las tareas. De este modo,

la falla del procesador sobreutilizado pone en riesgo la robustez del sistema aumentando la probabilidad de no realizar la recuperación en un tiempo razonablemente favorable.

- El tiempo de ociosidad (tiempo cuando el procesador está sin realizar alguna tarea) de un procesador sobreutilizado es significativamente menor que en los otros procesadores. El efecto de la reducción del tiempo de ociosidad forza a repartir el poco tiempo disponible para la ejecución de programas de monitoreo del sistema. Esto incrementa la probabilidad de que el procesador sobreutilizado esté subdiagnosticado y, por lo tanto, más propenso a producir resultados erróneos; i.e., errores que pueden dañar seriamente las operaciones del sistema.

- Supongamos que se conocen con anticipación las características de las tareas (tiempo de ejecución, cantidad de información a manipular, etc.), estas características son un factor abierto por naturaleza, es decir, no es posible conocer por anticipado, ni durante la ejecución, estos valores debido a que se depende de otros factores, que para el proceso en sí son totalmente desconocidos, por ejemplo: el tiempo de ejecución de una tarea será influida por la dependencia de los datos, por las características del hardware (tales como la frecuencia del procesador, el tipo de procesador, tiempo de acceso a la memoria, etc.) y otros factores.

- En los casos donde los programas de diagnóstico no son ejecutados o solamente son ejecutados por una fracción del tiempo de ociosidad, es concebible que la posibilidad de falla del procesador se incrementará.

Todos estos puntos adquieren una importancia considerable si partimos del hecho de que las aplicaciones que se ejecutan son críticas. Los puntos anteriores sugieren que un sistema equilibrado tiene un mayor rendimiento que un sistema en desequilibrio.

1.3 Construcción de la expresión matemática

Supóngase que para cada tarea *j* se conoce el número de instrucciones (I_{ij}), cuando es ejecutada por el procesador *i*, y el periodo de iteración de la tarea (T_j). El periodo de iteración es el tiempo total en el cual la tarea deberá ser ejecutada por un procesador específico. El periodo de iteración está dictado por condiciones físicas, tal como la frecuencia en que un dispositivo debe ser atendido. Si la velocidad del procesador *i* es R_i instrucciones por segundo, entonces la tarea *j* requiere I_{ij}/R_i segundos para ser ejecutada en el procesador *i*. La expresión (1.1) representa la fracción del periodo de iteración necesarios en la ejecución de la tarea *j* (si está asignado al procesador *i*).

$$\frac{I_{ij}}{R_i T_j} \tag{1.1}$$

Como se comentó anteriormente, la carga que una tarea puede generar sobre un procesador es, actualmente, un problema abierto, para ello definiremos

$$u_{ij} \in \Re \tag{1.2}$$

como la utilización del procesador *i* por la tarea *j* (en porcentaje).

Para la obtención de la expresión que plantee el problema de asignación, considere el siguiente *problema de asignación*. Supóngase un sistema consistente de *m* procesadores. Existen *n* tareas que deberán ser ejecutadas. Cada tarea será asignada a cierto número de procesadores. Dada la matriz $\boldsymbol{V} = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ con entradas 0 ó 1 para cada *i* y cada *j*, se tiene $V_{ij}=1$ si la *j*-ésima tarea está asignada al *i*-ésimo procesador, $V_{ij}=0$ en otro caso; dicha matriz es la *matriz de asignación*. El número de procesadores para el cual cada tarea *j* es asignada es el factor de copia r_j de la tarea; es decir, si la tarea *j* está compuesta por r_j copias, y cada copia es ejecutada por distintos

procesadores. Cada tarea j deberá entonces ser asignada exactamente a r_j procesadores distintos.

Entonces podemos definir la expresión (1.3) que representa la utilización del procesador i bajo la asignación V_{ij}.

$$p_i = \sum_{j=1}^{n} u_{ij} V_{ij} \tag{1.3}$$

De acuerdo con [2], minimizar la suma de todos los $p_i^{\,2}$ también minimiza la varianza estadística de las p_i's, la cual es una medida del desequilibrio en la utilización de los procesadores. El objetivo es tener una carga equilibrada debido a que un desequilibrio decrementa potencialmente el rendimiento del sistema, por razones comentadas anteriormente.

Considerando que se tienen los parámetros r_j y u_{ij} donde $1 \le i \le m$ y $1 \le j \le n$. El problema básicamente se puede definir como la acción de:

Minimizar

$$\sum_{i=1}^{m} p_i^{\,2} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} u_{ij} V_{ij} \right)^2 \tag{1.4}$$

restringido a

$$\sum_{i=1}^{m} V_{ij} = r_j \quad \forall\, 1 \le j \le n \tag{1.5}$$

En otras palabras, se desea encontrar una matriz $V = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ con entradas *ceros* o *unos* que minimice a la expresión (1.4), que es la función de costo, y que satisfaga la restricción indicada por la expresión (1.5).

Como se puede observar, es espacio de búsqueda consiste de matrices del tipo $\{0,1\}^{m\times n}$, por lo que el número total de asignaciones es de $2^{m\times n}$.

Para definir el planteamiento básico del problema se deben aislar algunos aspectos centrales e investigarlos a fondo antes de considerar dificultades adicionales. Para este caso se toma en cuenta los siguientes puntos:

- Supóngase que el sistema es homogéneo, es decir, todos los procesadores tienen las mismas características.
- Existe el mismo número de copias por cada tarea existente.
- No se considera la arquitectura de la red existente en el sistema.
- No se toma en cuenta las características de los sistemas operativos existentes en el sistema.

Una vez considerados los puntos anteriores se supone que $u_{ij}=u_{kj}$ $1 \le i,k \le m$ y $1 \le j \le n$, con lo cual la utilización se representa como: u_j, y el número de copias como $r_j=r$. La expresión (1.4) se reescribe como:

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{n} u_j V_{ij}\right)^2 \qquad (1.6)$$

y la (1.5) como:

$$\sum_{i=1}^{m} V_{ij} = r \quad \forall\, 1 \le j \le n \tag{1.7}$$

Para poder utilizar la expresión (1.7) como restricción es necesario modificarla, de tal manera que se obtenga un mínimo en el caso de que el número de copias *r* se satisfaga en todo momento. La nueva expresión se puede escribir como:

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m} V_{ij} - r\right)^2 \tag{1.8}$$

El problema de reasignación de tareas puede plantearse mediante una expresión que es la suma de las expresiones (1.6) y (1.8), esto debido a que se debe de considerar tanto la minimización de la función de costo, así como el número de copias que deben de existir en el sistema. Obteniéndose la siguiente expresión:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m} V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n} u_j V_{ij}\right)^2 \tag{1.9}$$

En la expresión (1.9) el primer término de la función es un mínimo si cada tarea es ejecutada en *r* procesadores distintos en forma simultánea y el segundo término representa la función de costo.

Se puede observar, por la forma en que se ha presentado, que el espacio de trabajo para la expresión (1.9) es un espacio binario, es decir, sólo trabaja con valores de 0 y 1 ($\{0,1\}^{m\times n}$). Pero también es posible que esta expresión pueda trabajar con valores continuos en el intervalo 0 y 1, es decir con valores reales ($\Re^{m\times n}$), para esto último es

necesario modificar dicha expresión.

1.4 Planteamiento del problema en $\Re^{m\times n}$

Para una matriz $\boldsymbol{V} = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ cuyas entradas están dentro de un intervalo continuo entre cero y uno, es necesario adicionar la expresión (1.9) que obligue a la matriz a que tenga entradas con valores finales de unos y ceros.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} V_{ij}\left(1-V_{ij}\right) \tag{1.10}$$

Para completar el planteamiento dentro de $\Re^{m\times n}$ se agrega la expresión (1.10) como otro sumando de la expresión (1.9) debido a que todo los términos deberán ser considerados como una sola expresión para poder llevar a cabo la minimización, quedando la expresión como:

$$E^{*} = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m} V_{ij} - r\right)^{2} + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n} u_{j}V_{ij}\right)^{2} + c\sum_{i=1}^{m}\sum_{j=1}^{n} V_{ij}\left(1-V_{ij}\right) \tag{1.11}$$

También cabe mencionar que cada término de la expresión puede tener una mayor o menor importancia que los otros términos, es por ello que se les agrega a cada término un coeficiente (*a, b, c*) que determina el grado de importancia que tienen dentro de la expresión matemática.

1.5 Perturbaciones al problema

Otra forma de plantear el problema de asignación sería:

Tomando en cuenta la expresión (1.3) que representa la carga existente en el procesador *i* bajo la asignación V_{ij} se puede definir la expresión

$$w(V) = \sum_{i=1}^{m} p_i = \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij} V_{ij} \tag{1.12}$$

que representa la carga total del sistema, y basándonos en la expresión (1.7) donde una tarea es ejecutada exactamente por *r* procesadores, podemos definir la siguiente expresión:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} V_{ij} = r \cdot n \tag{1.13}$$

que indica el número de *1's* esperados en la matriz de asignación ***V***, tomando en cuenta esto, la carga total del sistema puede ser representado también como:

$$w_0 = ru_1 + ru_2 + \ldots + ru_n = r \sum_{j=1}^{n} u_j \tag{1.14}$$

Ahora, siguiendo con el criterio de equilibrio de carga entre todos los procesadores del sistema, tenemos la siguiente expresión que indica la carga promedio que deberá tener cada procesador involucrado.

$$\overline{w} = \frac{r}{m} \sum_{j=1}^{n} u_j \tag{1.15}$$

Como se ha venido comentando, el equilibrio de cargas en un sistema distribuido

permite tener un mayor rendimiento del sistema, partiendo de esta premisa, durante la operación de un ambiente distribuido existen factores que constantemente están perturbando al sistema, es decir, que de una u otra manera tratan de desequilibrar la carga de trabajo en el sistema, estos factores son:

- La creación de tareas
- La terminación de tareas existentes
- La inserción de procesadores
- La eliminación de procesadores

Estos factores, como se menciono anteriormente, están presentes en todo momento y son parte del funcionamiento normal de un sistema distribuido, sus efectos se reflejan directamente en el rendimiento del sistema, por ello es necesario que dicho sistema tenga la capacidad de recuperarse ante estas eventualidades.

Para analizar cada uno de los factores, definimos como condición inicial que el sistema se encuentra en equilibrio de cargas.

Creación de tareas

El efecto de la creación de una tarea se ve reflejado directamente en r procesadores, los cuales tendrán una mayor carga de trabajo que los demás procesadores involucrados. El efecto se observa en la reducción del tiempo de ociosidad en los r procesadores.

Recordando la expresión (1.14), donde se obtiene el valor de la carga total del sistema, se puede observar que este valor se incrementa por la creación de la nueva tarea,

quedando la expresión como:

$$w_1 = (ru_1 + ru_2 + \ldots + ru_n) + ru_{n+1} = r\sum_{j=1}^{n+1} u_j \tag{1.16}$$

Por lo que la nueva carga promedio que deberá tener cada procesador involucrado está definida por:

$$\frac{r}{m}\sum_{j=1}^{n+1} u_j \tag{1.17}$$

Terminación de tareas

La terminación de una tarea también afecta directamente el equilibrio del sistema. El impacto se observa en el momento en que finaliza la ejecución de una tarea, debido a que existen r procesadores que tienen una carga de trabajo menor con respecto a los demás procesadores. Esto también afecta el rendimiento del sistema, principalmente en los $m-r$ procesadores, ya que éstos tienen un menor tiempo de ociosidad, lo cual, como hemos visto en éste capitulo es uno de los peligros latentes en un sistema distribuido con desequilibrio de cargas.

Tomando en cuenta la expresión (1.14), que nos indica el valor de la carga total del sistema, se ve decrementada por la terminación de una tarea, por lo que dicha expresión puede rescribirse como:

$$w_1 = (ru_1 + ru_2 + \ldots + ru_n) - ru_h = r\left(\sum_{j=1}^{n} u_j - u_h\right) \tag{1.18}$$

donde h indica la tarea que ha finalizado, $1 \leq h \leq n$.

Inserción de un procesador

El hecho de que un procesador se agregue al sistema afecta directamente a la capacidad máxima por procesador definido por la expresión (1.15), con lo que los demás procesadores pueden reducir su carga de trabajo delegando algunas tareas al nuevo procesador, con lo que la expresión (1.14) puede reescribirse como:

$$\frac{r}{m+t}\sum_{j=1}^{n} u_j \tag{1.19}$$

donde t indica el número de procesadores que se agregan al sistema.

Eliminación de un procesador

El hecho de que un procesador sea eliminado puede deberse a diferentes razones, una de ellas puede ser que presente fallas, o bien, que simplemente esa computadora se haya apagado. Esto, además de afectar el número de copias existentes por tarea y tener que realizar un reacomodo de las tareas que se tenían en ése procesador para mantener el número de copias, también afecta en forma directa el equilibrio en las cargas de trabajo de los procesadores restantes. Recordando la expresión (1.15) que indica el valor máximo de carga que puede tener un procesador. Dicha expresión puede reescribirse como:

$$\frac{r}{m-t}\sum_{j=1}^{n} u_j \tag{1.20}$$

donde t indica el número de procesadores eliminados, $1 \leq t \leq m$.

Es evidente que el incremento del valor de t implica que los procesadores restantes tendrán una mayor carga de trabajo.

1.6 Otros planteamientos del problema

Tomando en cuenta las expresiones (1.9) donde se define el planteamiento del problema de asignación, podemos modificar el segundo término de dicha expresión adicionándole la expresión (1.15), quedando como:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m}V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n}u_j V_{ij} - \frac{r}{m}\sum_{j=1}^{n}u_j\right)^2 \qquad (1.21)$$

Para la expresión (1.11), que es el planteamiento dentro de $\Re^{m\times n}$, modificando el segundo término queda como:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m}V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n}u_j V_{ij} - \frac{r}{m}\sum_{j=1}^{n}u_j\right)^2 + c\sum_{i=1}^{m}\sum_{j=1}^{n}V_{ij}\left(1-V_{ij}\right) \qquad (1.22)$$

Con el segundo término de las expresiones (1.9) y (1.11) se tiene un valor mínimo, lo mínimo que se puede obtener es cero, únicamente cuando no existe ninguna tarea ejecutándose, lo que elevaría el primer término de la misma expresión. Es por ello que se definen las expresiones (1.21) y (1.22) con el que se logra que cada procesador tenga una carga equitativa, con esto se logra que el espacio de búsqueda no tenga tantas variantes, es decir, se evita tener una mayor cantidad de mínimos locales como es en el caso de las expresiones (1.9) y (1.11).

En este trabajo se utiliza las expresiones (1.9) y (1.11) debido a que son las que se han encontrado dentro de la literatura, las expresiones (1.21) y (1.22) son una variante de las expresiones (1.9) y (1.11) respectivamente. Una desventaja de estas últimas expresiones es que tiene una mayor complejidad, lo cual, para un sistema donde se requiere una respuesta rápida no podrían encajar estas expresiones. Sin embargo se podría generar una segunda variante para reducir esta complejidad, dicha variante puede quedar

como:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m}V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n}u_j V_{ij} - d\right)^2 \qquad (1.23)$$

para el dominio $\{0,1\}^{m\times n}$, y para el dominio $\Re^{m\times n}$ queda como:

$$E^* = a\sum_{j=1}^{n}\left(\sum_{i=1}^{m}V_{ij} - r\right)^2 + b\sum_{i=1}^{m}\left(\sum_{j=1}^{n}u_j V_{ij} - d\right)^2 + c\sum_{i=1}^{m}\sum_{j=1}^{n}V_{ij}\left(1 - V_{ij}\right) \qquad (1.24)$$

donde *d* es una constante, cuyo valor adecuado se obtendría por experimentación. Con estas expresiones, (1.23) y (1.24) se reduce la complejidad la variante en las expresiones (1.21) y (1.22).

2. El problema como un proceso de optimización

Uno de los aspectos más desafiantes del análisis multivariable es el encontrar el valor óptimo que maximice o minimice una respuesta. El término de "algoritmos de optimización" es el nombre genérico para las técnicas que son diseñadas para resolver estos problemas. El proceso de optimización puede ser definido matemáticamente como la búsqueda de los valores de las n variables de una función ($f(x_1,x_2,...,x_n)$) que optimicen $f(x)$. Este proceso puede ser un problema de maximización o un problema de minimización dependiendo del problema. Para fines de este trabajo, el objetivo es minimizar las expresiones (1.9) y (1.11), como se menciono en el capítulo anterior, cada una de estas expresiones fue creada dependiendo de los valores utilizados en el espacio de búsqueda. Durante el desarrollo de este capítulo se indicara cual expresión es utilizada, y en lo sucesivo se le denominara como *función objetivo*

Existen varios métodos de aproximación que son utilizados para resolver este tipo de problemas. En este capítulo se analizan técnicas clásicas como *búsqueda local* y *recocido simulado*, y técnicas de computación inteligente como *algoritmos genéticos*, así como las características de cada una de ellas y sus procedimientos.

Como se mencionó en el capítulo anterior, el dominio del problema de asignación es el "hipercubo" $\{0,1\}^{m\times n}$. Dado un espacio de búsqueda $\{0,1\}^{m\times n}$ que a su vez esta contenido en el cubo real del espacio euclidiano (mn)-dimensional.

$$\{0,1\}^{m\times n} \subset [0,1]^{m\times n} \subset \Re^{m\times n} \tag{2.1}$$

2.1 Búsquedas locales

En la búsqueda local para la optimización de una función objetivo f, dado un punto actual V se obtiene un valor $v = f(V)$, el cual es comparado con los valores obtenidos

por cada vecino de V. V es actualizado por el valor del vecino que minimice a f. Para ello es necesario definir el concepto de vecindad.

En $\Re^{m\times n}$, considere la norma de convergencia uniforme $N_\infty : V \mapsto Max_{ij}|v_{ij}|$. Dado una $\varepsilon > 0$ y un punto $V_0 \in \Re^{m\times n}$, los $(2mn, \varepsilon)$-vecinos de V_0 son los elementos del conjunto

$$V_\varepsilon^g(V_0) = \left\{V \in \Re^{m\times n} \mid V \neq V_0 \wedge \forall i,j\left(v_{ij} = v_{ij}^0 \vee \left|v_{ij} - v_{ij}^0\right| = \varepsilon\right)\right\} \tag{2.2}$$

consecuentemente

$$V \in V_\varepsilon^g(V_0) \Rightarrow N_\infty(V - V_0) = \varepsilon \tag{2.3}$$

Para cualquier punto, el número de vecinos para $(2mn, \varepsilon)$-vecinos está indicado por $2mn$.

$V_\varepsilon^g(V_0) = \left\{V \in \Re^{m\times n} \mid V \text{ difiere de } V_o \text{ en una sóla coordenada , con diferencia } \varepsilon\right\}$

o equivalentemente

$$V \in V_\varepsilon^g(V_0) \mid \exists i_0 \leq m, j_0 \leq n \forall i,j : \left|v_{ij} - v_{ij}^0\right| = \begin{cases} \varepsilon & \text{si } i = i_0 \wedge j = j_0 \\ 0 & \text{en otro caso} \end{cases} \tag{2.4}$$

Como una primera aproximación para definir la noción de vecindad en $[0,1]^{m\times n}$, para $\varepsilon > 0$ y $V_0 \in [0,1]^{m\times n}$ se puede decir que $V \in [0,1]^{m\times n}$ es un (k,ε)-vecino para V_0 en $[0,1]^{m\times n}$, para $k = 2mn$, si hay un (k,ε)-vecino $V' \in \Re^{m\times n}$ de V_0 en $\Re^{m\times n}$ tal que $V \equiv V' \bmod 1$.

Como una segunda aproximación, si V' es un (k,ε)-vecino $V' \in \Re^{m\times n}$ de V_0 en $\Re^{m\times n}$, con $V_0 \in [0,1]^{m\times n}$ entonces:

- Si $V' \in [0,1]^{m\times n}$ entonces V' es también un (k,ε)-vecino de V_0 en $[0,1]^{m\times n}$, y
- Si $V' \notin [0,1]^{m\times n}$ entonces sea t el número máximo en [0,1] tal que $V = tV' + (1-t)V_0$. Entonces V es un (k,ε)-vecino de V_0 en $[0,1]^{m\times n}$.

Ahora bien, el diámetro del hipercubo $\{0,1\}^{m\times n}$, con respecto a la norma N_∞ es exactamente 1, es decir, dado $\varepsilon = 1$, la noción de vecindad se define como sigue: para cualquier $V_0 \in \{0,1\}^{m\times n}$ se puede decir que $V \in \{0,1\}^{m\times n}$ es un vecino de V_0 en $\{0,1\}^{m\times n}$, si existen los índices $i_0 \le m$, $j_0 \le n$ tal que:

$$\forall i \le m, j \le n : v_{ij} = \begin{cases} \bar{v}_{ij}^{0} \text{ si } (i,j) = (i_0, j_0) \\ v_{ij}^{0} \text{ en otro caso} \end{cases} \tag{2.5}$$

Basándose en esta definición, se explicará la forma en que trabaja este método, para ello se tomará a la expresión (1.9) como la función objetivo.

El objetivo principal de este algoritmo es encontrar la matriz que minimice a la función objetivo, para ello

$$V = \min(f(V_h)) \quad \forall\, h = 1 \ldots n \cdot m \tag{2.6}$$

Procedimiento BL: Dado un punto inicial en el espacio de búsqueda con su respectivo valor obtenido por la función objetivo, a éste se le considera como punto *actual*. En cada iteración de este procedimiento, se busca entre los posibles vecinos del punto actual a un vecino cuyo valor en la función objetivo sea menor al del actual, si se

localiza a tal vecino, a éste se le considera como el punto actual con fines de una nueva iteración. Esto se repite en tanto no se arribe a condiciones terminales.

```
Entrada:    func        : Función objetivo
            Ainit       : Matriz inicial
Salida: V               : Matriz de asignación
Algoritmo:
    {
        Anew = Ainit;
        Vinit = func(Ainit);
        Anew = Mostvariation(Ainit);
        Vnew = func(Anew);
        while (Vnew < Vinit) {
            Vinit = Vnew;
            Anew = Mostvariation(Ainit);
            Vnew = func(Anew);
        }
    }
```

Esquema de la búsqueda por vecinos

2.2 Recocido simulado (RS)

Recocido simulado es una técnica de optimización basada en el principio de termodinámica [20,14]. El término "recocido" se refiere al proceso por el cual un sólido es inicialmente fundido y posteriormente es enfriado paulatinamente reduciendo su temperatura lentamente. Durante el proceso de enfriamiento las partículas del material intentan acomodarse en un estado de menor energía. Los estados colectivos de energía del ensamble de partículas puede ser considerado la "configuración" del material. La probabilidad de que una partícula esté en cualquier nivel de energía puede ser calculada utilizando la distribución de Boltzmann. Como la temperatura del material decrece, la distribución de Boltzmann tiende hacia una configuración donde las partículas tienen un nivel más considerablemente menor de energía

Cuando el sistema es perturbado (reducción de la temperatura) para producir una nueva configuración de las partículas el nivel de energía anterior a la perturbación (E_s) y el nivel de energía posterior a la perturbación (E_i) son comparadas. Si E_s es mayor que E_i (i.e., $E > 0$), el nuevo sistema perturbado es considerado como la nueva configuración de las partículas. Si $E < 0$, la probabilidad de aceptar al sistema perturbado sigue el criterio de Metropolis [16] mostrado en la expresión (2.7).

$$p = e^{-E/kT} \tag{2.7}$$

donde k es la constante de Boltzmann y T es la temperatura. Usando este criterio, el material alcanzará eventualmente su configuración de equilibrio.

El concepto básico puede ser aplicado a problemas de optimización numérica. Recocido simulado (RS) aplica el criterio de Metropolis a una serie de variables (configuración) para optimizar el sistema. El nuevo valor de la variable es obtenido perturbando la configuración actual y puede ser considerado como un paso o una serie de movimientos sobre la respuesta. Para la optimización numérica, la respuesta de la función objetivo, que se denotará por R (i.e. $R = f(x)$), reemplaza a los términos de energía. El concepto de temperatura es mantenido. De cualquier modo, esto es combinado con k y utilizado como un factor importante de control. La probabilidad de aceptar un paso perjudicial, i.e., R > 0 (suponiendo minimización) es guiado por la expresión (2.8).

$$p = e^{-R/T} \tag{2.8}$$

El valor obtenido, al momento de evaluar la expresión (2.8) es comparado con un número aleatorio P obtenido de una distribución aleatoria uniforme en el intervalo [0,1]. Si $P > p$, el paso perjudicial es rechazado y un nuevo movimiento es realizado a partir de la actual posición. Si $p < P$, el paso perjudicial es aceptado y una nueva configuración reemplaza a la anterior. Un nuevo paso es entonces tomado relativo a esta configuración.

Este criterio permite la posibilidad de que un nuevo valor sea aceptado como la nueva configuración, siempre y cuando exista una mala respuesta de la función objetivo respecto a la actual configuración. Los movimientos que no son muy buenos son menos idóneos para ser aceptados que los movimientos que son buenos. Este aspecto permite que el algoritmo salga de un local óptimo. Esto se repite hasta que algún criterio de terminación es alcanzado.

Análogo al proceso físico, T es lentamente reducido causando la probabilidad de aceptar movimientos no muy buenos. La función por el cual T es reducido es llamado *función de enfriamiento* y es crítico para el éxito del RS. Dos importantes parámetros dirigen a la función de enfriamiento, los cuales son: el factor de vecindad para la perturbación y T.

La principal ventaja de RS es la habilidad para desplazarse del óptimo local. Así, la habilidad para encontrar el óptimo global no está relacionado con las condiciones iniciales (i.e, el punto de inicio). RS es también muy simple de implementar. Las principales desventajas de RS son la naturaleza subjetiva de elegir la configuración de los parámetros de RS (e.g. T y el factor de vecindad) y que esto requiere típicamente más evaluaciones de la función objetivo que otros métodos de optimización. RS se define como un algoritmo de "caminata aleatoria sesgada (biased random walk)". A diferencia de otros métodos que intentan hacer movimientos inteligentes sobre la respuesta, el paso en RS es tomado aleatoriamente. Así, RS está clasificado como una búsqueda heurística débil.

Problema de asignación

Para nuestro problema de asignación de tareas, el nivel de energía del material está determinada por la respuesta de la expresión (1.10), la cual será la función objetivo para RS. La configuración de las partículas está dada por la matriz $V = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ cuyas entradas son valores continuos en el intervalo de *cero* y *uno*, definamos a

V^* como el conjunto de todas las configuraciones existentes. La obtención de una nueva configuración, $V_A \in V^*$, está determinada por el cálculo de un vecino cercano a V, la diferencia ente V y V_A es una variación en una de sus entradas, esta variación está dada por una constante definida *de vecindad* (ε). En nuestro problema, el hecho de calcular un vecino es interpretado como la perturbación al sistema.

La expresión (2.9) determina el cálculo del vecino.

$$V_{kl} = V_{kl} + (2k-3) \times \varepsilon \tag{2.9}$$

donde:

k	es la selección del vecino (izquierda o derecha)
ε	es la constante de la distancia entre vecinos
k,l	son la posición de una entrada específica

Durante la perturbación se puede generar alguna configuración indeseable, por lo que la probabilidad de aceptar una configuración indeseable está determinada por la siguiente expresión:

$$P = e^{\left(E_i - E_s / T\right)} \tag{2.10}$$

donde

T	es la temperatura del sistema
E_s	es el nivel de energía antes de la perturbación
E_i	es el nivel de energía posterior a la perturbación

La *función de enfriamiento* esta dada por la expresión

$$T = \frac{1}{t} \tag{2.11}$$

donde t es incrementado por cada vez que se perturba el sistema. Podemos ver que la temperatura se decrementa paulatinamente con respecto al tiempo.

El algoritmo se presenta a continuación:

```
Entradas:   func        : Función objetivo
            eps         : Distancia entre los vecinos
            coolSched   : Función de enfriamiento
            maxtime     : Número máximo de iteraciones
            n x m       : Orden de la matriz de asignación
Salida: V           : Matriz de asignación con 0's y 1's
Algoritmo:
    {
        Actual = Matriz(0.5,m,n);
        Costo = func(Curr);
        for time = 1 to time = maxtime {
            temp = coolSched(time);
            if (temp = 0) then time = maxtime;
            else {
                Vecina = com(Actual,eps);
                NuevoCosto = func(Vecina);
                if (NuevoCosto < Costo) then {
                    Actual = Vecina;
                    Costo = NuevoCosto;
                }
                else {
                    Cambio = Exp( (NuevoCosto - Costo) / temp )
                    tómese p∈ [0,1];
                    if (Cambio > p) then {
                        Actual = Vecina;
                        Costo = NuevoCosto;
                    }
                }
            }
        }
    }
```

La condición de terminación de este algoritmo depende bien de un valor máximo de iteraciones fijo de antemano, o bien de umbral tal que, habiendo alcanzado la función de enfriamiento un valor por debajo de él, el proceso se detiene.

Una de las características de este método es que trabaja con valores continuos, es decir, con valores reales dentro del intervalo [0,1].

Una desventaja de este algoritmo es el tiempo de cómputo requerido para alcanzar al valor óptimo de la función objetivo. Esto se debe a que en nuestro problema se requiere por lo menos $2^{m \times n}$ iteraciones para considerar la posible variación de todas y cada una de las entradas de la matriz de asignaciones. Por otro lado, el algoritmo trabaja con valores de punto flotante, lo cual también aumenta el número de instrucciones a ser ejecutadas por el algoritmo.

Técnicas de relajamiento

Debido a que la técnica de recocido simulado utiliza valores continuos, es necesario asegurar que la matriz $V = \left(V_{ij}\right)_{1 \le i \le m}^{1 \le j \le n}$ de orden $m \times n$ tenga entradas *ceros* o *unos*. Para ello se lleva a cabo un proceso de discretización siguiendo la regla

$$v_{ij} = \begin{cases} 1 & \text{si } v_{ij} \ge umbral \\ 0 & \text{en otro caso} \end{cases} \qquad 1 \le j \le n, 1 \le i \le m \tag{2.8}$$

Por la expresión que define al problema es indispensable que las entradas de las matrices de asignación sean valores 0 ó 1, pues éstos determinan unívocamente el asignar o no una tarea a un procesador. El procedimiento de recocido simulado trabaja en el "cubo continuo" $[0,1]^{m \times n}$, por lo cual los valores del punto óptimo han de ser redondeados para que queden en el conjunto de vértices $[0,1]^{m \times n}$ del "cubo continuo"

$[0,1]^{m\times n}$. Con el criterio de la ecuación (2.12), si el umbral fuese 1/2, dado un punto en el "cubo continuo" $[0,1]^{m\times n}$ se está obteniendo el vértice en $[0,1]^{m\times n}$ más cercano a él. Debido a que la función objetivo es continua, se espera que el valor en el vértice seleccionado esté cerca del correspondiente al óptimo encontrado. Al variar el umbral se espera que las variaciones entre el óptimo encontrado y el vértice seleccionado no sean demasiado bruscas.

Existen varios métodos para la obtención del umbral, una de ellas es el determinado por la expresión.

$$umbral = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} V_{ij} \tag{2.17}$$

con la que se obtiene un promedio de los valores existentes en la matriz V, la desventaja de este método es que la probabilidad de la existencia de variaciones muy bruscas es mayor.

Otra forma de obtener el umbral es a través de procedimientos experimentales y adecuado para la naturaleza del problema en particular que se esté tratando.

2.3 Algoritmo Genético

Un algoritmo genético (AG) es un modelo de aprendizaje que debe su comportamiento a una metáfora de algunos de los mecanismos de evolución en la naturaleza. Cada AG está dado como un conjunto de transformaciones que generan consecutivamente poblaciones de individuos, formados por cromosomas. Cada uno de estos cromosomas es una cadena de caracteres de manera similar a la forma en que un cromosoma en la naturaleza es una secuencia de DNA. Los individuos en la población

entonces sufren una sucesión de diversas transformaciones [22,12,13].

Como se mencionó anteriormente, cada AG es un modelo de aprendizaje que puede ser utilizado para resolver problemas de búsquedas u optimización. Está basado en los procesos genéticos que se observan en los organismos biológicos. A lo largo de muchas generaciones, las poblaciones de ciertos organismos evolucionan de acuerdo a los principios de la selección natural y supervivencia de los más aptos. Mediante la imitación de este proceso, los AGs son capaces de proporcionar soluciones para problemas reales, si es que éstos son codificados de forma adecuada.

Exactamente cuáles procesos biológicos son esenciales para la evolución y cuáles procesos juegan o no un papel importante dentro de ésta, es todavía una cuestión de debate; pero las bases son claras [22,12,10].

En la naturaleza, los individuos que forman parte de una población compiten entre ellos por recursos; tales como comida, agua, vivienda, etc.; también los miembros de una misma especie usualmente compiten por la obtención de una pareja. De forma que, los individuos que tengan un mayor éxito en la supervivencia y en la atracción de una pareja tendrán un mayor número de descendientes; por otro lado, los individuos que presenten un desempeño bajo en comparación con el resto de la población, tendrán una descendencia pequeña o nula. Lo anterior nos indica que los genes de los que tienen un índice de adaptación o desempeño alto provocará la aparición creciente de individuos más aptos dentro de cada siguiente generación. Ocasionalmente, la combinación de características buenas de diferentes ancestros puede producir descendientes cuyo desempeño es mayor en comparación con cualquier padre. De esta forma, las especies evolucionan y se encuentran más preparadas para sobrevivir dentro de su medio ambiente [22,12].

Los AGs utilizan una analogía directa con el comportamiento natural de las especies: trabajan con una población de individuos, donde cada uno de éstos representa una posible solución para un problema específico. A cada individuo se le asigna una

"medida" de su desempeño, dependiendo de qué tan buena solución represente para el problema en cuestión. Esto es equivalente a estimar qué tan efectivo es un organismo en la competencia para la obtención de recursos dentro de su medio ambiente.

Una nueva población de posibles soluciones puede ser producida mediante la selección de los mejores individuos de la generación actual, y la cruza de éstos produce un nuevo conjunto de individuos. Esta nueva generación contiene una proporción más alta de las características que poseían los miembros más aptos de la generación previa. De esta forma, a través de las generaciones, son mezcladas e intercambiadas con otras buenas características a lo largo del tiempo. Si se favorece la reproducción de los individuos más aptos, entonces son exploradas las áreas que puedan contener una solución adecuada dentro del espacio de búsqueda; de modo que si el AG fue diseñado correctamente, se aumenta la probabilidad de que la población converja hacia la solución óptima para el problema [12].

El poder de los AGs se debe a que la técnica es robusta y puede manejar exitosamente una gran cantidad de problemas, incluyendo aquellos que son de difícil solución mediante otros métodos. Los AGs no garantizan el encontrar una solución global óptima para un problema, pero generalmente son buenos para encontrar soluciones que pueden considerarse como "aceptables". Cabe señalar que en el caso de que existan técnicas especializadas para la solución de un problema específico, éstas tendrán mejor desempeño en comparación con el AG tanto en velocidad como en precisión. Entonces, el campo de trabajo para el AG es en áreas difíciles donde no existan técnicas o métodos para la solución de un problema. Actualmente, aún cuando ciertas técnicas trabajan adecuadamente, se les han hecho mejoras mediante la combinación de éstas con el AG [22,1,9].

Requerimientos

Antes de que el AG pueda implementarse sobre una aplicación específica, es necesario planear una codificación o representación adecuada para el problema a resolver, así como establecer una función objetivo, la cual asigna un valor de desempeño a cada solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para reproducirse y generar descendencia.

Codificación

Se supone que cada solución potencial para un problema puede ser representada por un conjunto de parámetros. Estos parámetros, conocidos como genes, se unen para formar una cadena con valores específicos, a la cual se le denomina cromosoma. Holland fue el primero en mostrar que lo ideal es usar un alfabeto binario para la generación de la cadena; aunque no es la única opción [12].

En términos genéticos, el conjunto de parámetros representados por un cromosoma en particular se denomina genotipo; éste contiene la información necesaria para la construcción de un organismo, el cual se denomina fenotipo. Los mismos términos son utilizados dentro de los AGs; por ejemplo, en la tarea del diseño de un puente, el conjunto de parámetros que especifican un diseño en particular representan el genotipo; mientras que el puente en si, es el fenotipo. Éste puede ser inferido a partir del genotipo, ya que es posible calcularlo mediante el empleo de la función objetivo [12].

Función objetivo

La función objetivo debe ser planteada para cada problema que se pretenda resolver. Dado un individuo, éste debe de regresar un valor numérico único que represente su medida de desempeño; la cual debe de reflejar la utilidad o habilidad del individuo representado por este cromosoma. Para una gran variedad de problemas, particularmente para la optimización de funciones, es obvio lo que la función debe medir: el valor de la

función a optimizar. Pero no siempre es así, ya que se puede presentar optimización combinatoria; donde hay diversas medidas de desempeño que se quieren optimizar al mismo tiempo [22,12,13,12,9,8].

Implementación

En la práctica, se puede implementar este modelo genético de computación mediante arreglos de bits o caracteres que representen los cromosomas. Operaciones simples de manipulación de bits permiten la implementación de las operaciones de *recombinación homóloga*, la *mutación* y otro proceso especial denominado *adición de padres*.

Cuando un AG es implementado, usualmente se hace de forma que involucre el siguiente ciclo (fig 2.1):

- Generación de la población inicial en forma aleatoria
- Evaluación del desempeño de cada uno de los individuos que conforman la población
- Creación de una nueva población mediante la ejecución de las operaciones, las cuales se explican más adelante.
- Eliminación de la población anterior e iterar con la nueva población hasta cumplir un criterio de terminación.

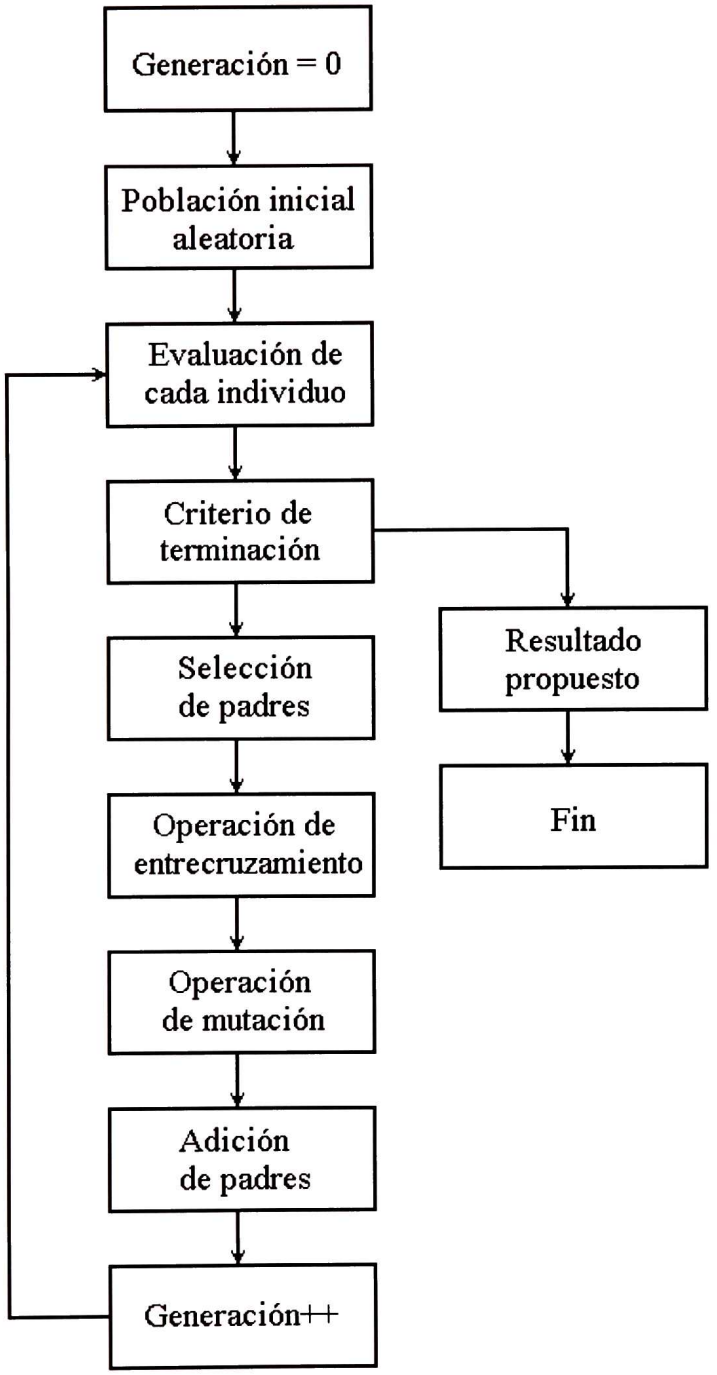

Fig. 2.1: Diagrama de flujo del Algoritmo Genético

Recombinación homóloga

Este proceso también se le conoce con el nombre de *entrecruzamiento* (*crossover*).

Dado $f:\{0,1\}^{m\times n}\rightarrow\Re$ una función objetivo, y el dominio $\{0,1\}^{m\times n}$ dentro de una cadena de bits de longitud $m\cdot n$.

Sea P un conjunto de elementos de la forma $\{0,1\}^{m\cdot n}$ la población actual en el dominio, y $p=card(P)$ el número de elementos en P. manipulando cada elemento de P como un vector, entonces la población de P puede ser representada como una matriz de tipo $\{0,1\}^{p\times m\cdot n}$.

Sea k un valor entero tal que $k\mid m\cdot n$, es decir, k divide al producto $m\cdot n$. Cada individuo $V\in P$ puede ser una yuxtaposición de k bloques de $\frac{m\cdot n}{k}$ bits. Se puede decir que $V=\rho_1(V)*\cdots\rho_k(V)$. De P se define una nueva población $Q\subset\{0,1\}^{m\cdot n}$ como sigue:

Sea $V_f\in P\ \forall f\leq p$, la creación de individuos será:

$$\left\{W_q=\rho_1(V_{f+d})*\cdots\rho_g(V_{f+d})\mid\forall f\leq p,g\leq k,d\leq p\right\}\in Q\ \ \forall q\leq p^k \quad (2.14)$$

El mapeo $P\mapsto Q$ es la operación de entrecruzamiento.

Por ejemplo: sea $P=\begin{Bmatrix}101110011\\111100010\end{Bmatrix}$, por lo tanto, $p=2$ es la cardinalidad o el número de elementos del conjunto P, la longitud de cada elemento de P es: $m\cdot n=9$, los cuales serán dividido en $k=3$ bloques cada uno. el resultado del entrecruzamiento es:

$$Q = \begin{Bmatrix} 101110011 \\ 101110010 \\ 101100011 \\ 101100010 \\ 111110011 \\ 111110010 \\ 111100011 \\ 111100010 \end{Bmatrix}$$ y el número de elementos es $q = 8$.

Mutación

Sea $x \in [0,1]$ el porcentaje de los genes de la población total a ser mutado, sean Z un vector de la forma $(i,j) | \forall i \leq m \cdot n \wedge j \leq q$ donde los valores de i y j se obtienen de manera aleatoria, y el número de entradas de Z es $\lfloor x \cdot 100 \rfloor$. Se define el punto tal que:

$$\mu p(i_0, j, Z) = (t_i)_{i \leq m \cdot n} \text{ donde } \forall i \leq m \cdot n : t_i = \begin{cases} \bar{v}_i^{i_0} & \text{si } i = i_0 \\ v_i^{i_0} & \text{en otro caso} \end{cases} \quad (2.15)$$

$\mu p(i_0, j_o, Z)$ es la operación de mutación.

Siguiendo con el ejemplo anterior se define una probabilidad de mutación del 16%, obteniendo como resultado de la operación $\mu p(i_0, j_o, z) = \begin{Bmatrix} 100110011 \\ 101110110 \\ 101111111 \\ 101101010 \\ 101110011 \\ 111010010 \\ 101101111 \\ 111100010 \end{Bmatrix}$

Adición de padres

En esta parte se adicional los padres, que generaron la nueva población, al resultado del proceso de mutación. Este proceso se realiza para asegurar la convergencia del algoritmo.

$$Q = Q \cup P \tag{2.16}$$

La población del ejemplo quedará como:

$$Q = \begin{Bmatrix} 100110011 \\ 101110110 \\ 101111111 \\ 101101010 \\ 101110011 \\ 111010010 \\ 101101111 \\ 111100010 \\ 101110011 \\ 111100010 \end{Bmatrix}$$

Proceso de selección

Dada la población $\boldsymbol{Q}$ constituida por q individuos, ordenados en forma creciente con respecto a los valores obtenidos por la función objetivo:

$$Q = (V_i)_{i \le q} \text{ con : } \forall i_1, i_2 \left(i_1 \le i_2 \Rightarrow f(V_{i_1}) \le f(V_{i_2}) \right) \tag{2.17}$$

entonces, se consideran los primeros p individuos de la población $\boldsymbol{Q}$, los cuales formarán la nueva población de P.

Para nuestro ejemplo, se tomarán únicamente a los dos primeros elementos que den un mejor resultado, en este caso se supone que fueron el tercer y cuarto elementos de

conjunto Q, estos elementos serán los nuevos padres de la siguiente población, es decir:

$$P = \begin{Bmatrix} 101111111 \\ 101101010 \end{Bmatrix}$$

Problema de asignación

Para nuestro problema de asignación cada individuo de la población está representado por una matriz $V = \left(V_{ij}\right)_{1\le i\le m}^{1\le j\le n}$ de orden $m \times n$ con entradas *ceros* o *unos*, lo que implica que cada individuo consta de $m \cdot n$ bits. Cada individuo está conformado por k cromosomas. La forma en que se programó el algoritmo para éste trabajo se esquematiza a continuación

Entrada:	func	: Función objetivo
	probmut	: Probabilidad de mutación
	n x m	: Orden de la matriz de asignación
	ng	: Número máximo de generaciones
	np	: Número de padres
	nt	: Número de entrecruzamientos
Salida:	V	: Matriz de asignación

Algoritmo:

$ns = np^{nt}$
done = *False*;
makeinit(sons);
done = evalpob(func,sons,V);
makeparents(parents,sons,noparents);
while ((gen < ng) && (**not** done)) **do**
    makeson(sons,parents);
    mutation(sons,probmut);
    addparents(sons,parents);
    done = evalpob(func,sons,V);
    makeparents(parents,sons,noparents);
    gen = gen + 1;

Esquema del Algoritmo Genético

3. Métodos heurísticos

Uno de los métodos generalmente utilizados por su velocidad en la obtención de la solución son los algoritmos heurísticos, dichos métodos se basan en la experiencia, lo cual permite su facilidad tanto de programación como de ejecución. Considere los siguientes datos:

Los métodos programados en este parte del trabajo se basan en la expresión (1.15), con la cual se pretende obtener una matriz de asignación que dé como resultado un sistema equilibrado.

Para la realización de estos métodos, tómese un vector $\boldsymbol{X} = (X_j)_{1 \le j \le n}$ de orden n cuyas entradas tienen como valor inicial el número de copias por tarea (r), es decir, $X_j = r \quad \forall\, 1 \le j \le n$. Este vector se utiliza para asegurar que cada tarea sea asignada a r procesadores diferentes, con el fin de asegurar el número de copias de cada tarea en el sistema, para ello este vector se decrementará en la posición correspondiente a la tarea, cuando dicha tarea sea asignada a algún procesador.

Para facilidad de cada algoritmo, el vector de carga de las tareas, u_j, está ordenado en forma ascendente.

3.1 Asignación por tarea-procesador-maxmin (TPMM)

Este algoritmo consiste en asignar, primero las máximas tareas que puede realizar un procesador antes de pasar al siguiente procesador, para este proceso, primero toma la tarea más pesada disponible y posteriormente la tarea menos pesada disponible. La capacidad máxima que debe tener un procesador, para mantener el equilibrio de cargas

del sistema, está determinada a la expresión (1.15), es decir, se seguirán asignando tareas a un procesador mientras se cumpla la siguiente condición:

$$g_i < \frac{r}{m} \sum_{j=1}^{n} u_j \tag{3.1}$$

donde:

$$g_i = \sum_{j=1}^{n} V_{ij} u_j \tag{3.2}$$

Esto se repite por cada procesador existente en el sistema.

Procedimiento TPMM: Dado un vector inicial X de longitud n cuyo valor inicial en cada entrada es igual al número de copias por tarea, y un vector u ordenado en forma ascendente, posteriormente para cada procesador existente se toma la tarea más pesada disponible y se decrementa el vector X en la posición correspondiente. Dentro del vector de asignación se asigna un *uno* a la posición correspondiente al de la tarea y del procesador. Esto se repite hasta terminar con todos los procesadores y no se sobrepase el valor promedio de la carga de trabajo de cada tarea. Posteriormente se ordenan los procesadores en forma ascendente considerando la carga de trabajo que tienen y se toma el de menor carga. Se le asigna la tarea más pesada que no haya sido asignada. Esta última parte se repite hasta que el vector X tenga todas sus entradas con valor de *cero*. En la figura 3.1 se muestra un esquema de este algoritmo.

```
Entrada:    func        : Función objetivo
            n x m       : Orden de la matriz de asignación
            r           : Número de copias
Salida:     V           : Matriz de asignación
Algoritmo:
    {
        Maxr = r;
        maxmin = true;
        lowindex = 1;
        highindex = n;
        i = highindex;
        j = 1;
        while (tareas por asignar  y procesadores por ser visitados) {
            while (true) {
                if (u[j] <= a la capacidad del procesador)
                    sizeproc = sizeproc – u[j];
                    V[j,i] = 1;
                    Maxr[i] = Maxr[i] – 1;
                else break;
                if (maxmin)
                    i = lowindex;
                    lowindex = lowindex + 1;
                else
                    i = highindex;
                    highindex = highindex - 1;
                maxmin = not(maxindex)
            }
            j = j + 1;
        }
        sortbyprocs;
        while (fulltasklist) {
            V[getlowproc,i] = 1;
            Maxr[i] = Maxr[i] – 1;
            sortbyprocs;
        }
    }
```

Fig. 3.1: Esquema de "Tarea-Procesador-MaxMin".

Considere un sistema compuesto de 4 procesadores y 5 tareas, cada tarea consta de 2 copias en total, el vector de utilización es: $u = \{0.20,0.30,0.60,0.68,0.79\}$. Evaluando los valores anteriores en la expresión (1.14) se obtiene el valor total de la carga del

sistema, el cual es: $w_0 = 8.74$, y la carga promedio por procesador se obtiene a través de la expresión (1.15) obteniendo $\overline{w} = 2.185$, y el vector $X = \{2,2,2,2,2\}$. A continuación, en las figura 3.2, se muestra una ejecución con este algoritmo.

No. Paso	Matriz de asignación	No. Paso	Matriz de asignación
1	$V = \begin{Bmatrix} 0,0,0,0,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{2,2,2,2,1\}$	6	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 1,0,0,0,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{0,1,2,1,0\}$
2	$V = \begin{Bmatrix} 1,0,0,0,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{1,2,2,2,1\}$	7	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 1,0,0,1,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{0,1,2,0,0\}$
3	$V = \begin{Bmatrix} 1,0,0,1,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{1,2,2,1,1\}$	8	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 1,1,0,1,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{0,0,2,0,0\}$
4	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{1,1,2,1,1\}$	9	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 1,1,0,1,1 \\ 0,0,1,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{0,0,1,0,0\}$
5	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 0,0,0,0,1 \\ 0,0,0,0,0 \\ 0,0,0,0,0 \end{Bmatrix}$ $X = \{1,1,2,1,0\}$	10	$V = \begin{Bmatrix} 1,1,0,1,1 \\ 1,1,0,1,1 \\ 0,0,1,0,0 \\ 0,0,1,0,0 \end{Bmatrix}$ $X = \{0,0,0,0,0\}$

Fig. 3.2: Resultados de una ejecución del algoritmo *TPMM*

3.2 Asignación por tarea-procesador-máximo (TPM)

Este algoritmo se basa en asignar todas las tareas posibles a un procesador. Dicha asignación la hace tomando la primera tarea más pesada, y luego la siguiente tarea más pesada que no rebase la carga de trabajo permitido para el procesador, y continúa con cada una de las tareas existentes mientras se cumpla la restricción dada por la expresión (3.1) donde la carga existente en el procesador se calcula con la expresión (3.2). El procedimiento es el siguiente:

Procedimiento TPM: Dado un vector inicial de longitud n cuyo valor inicial en cada entrada es igual al número de copias por tarea, a este vector le denominaremos X, y un vector u ordenado en forma ascendente. Con el primer procesador se toma la primera tarea de mayor peso, posteriormente se toma la segunda tarea más pesada y se asigna al procesador. Esto se repite hasta alcanzar el valor máximo permitido por procesador. Cuando se alcance dicho valor se continúa el proceso con el siguiente procesador. Esto se repite en cada uno de los procesadores del sistema. En la figura 3.3 se muestra un esquema del algoritmo programado.

```
Entrada:    func        : Función objetivo
            n x m       : Orden de la matriz de asignación
            r           : Número de copias
Salida:     V           : Matriz de asignación
Algoritmo:
    {
        Maxr = r;
        i = n;
        while (fulltasklist && j < m) {
            while (sizeproc >= 0.0) {
                if (u[j] <= sizeproc) {
                    sizeproc = sizeproc – u[j];
                    V[j,i] = 1;
                    Maxr[i] = Maxr[i] – 1;
                }
                else break;
                i = i - 1;
            }
            j = j + 1;
        }
        sortbyprocs;
        while (fulltasklist) {
            V[getlowproc,i] = 1;
            Maxr[i] = Maxr[i] – 1;
            sortbyprocs;
        }
    }
```

Fig. 3.3: Esquema de "Tarea-Procesador-Máximo"

Considere un sistema compuesto de 4 procesadores y 5 tareas, cada tarea consta de 2 copias en total, el vector de utilización es: $u = \{0.20,0.30,0.60,0.68,0.79\}$. Evaluando los valores anteriores en la expresión (1.14) se obtiene el valor total de la carga del sistema, el cual es: $w_0 = 8.74$, y la carga promedio por procesador se obtiene a través de la expresión (1.15) obteniendo $\overline{w} = 2.185$, y el vector $X = \{2,2,2,2,2\}$. A continuación, en la figura 3.4, se muestra una ejecución con este algoritmo.

No. Paso	Matriz de asignación	No. Paso	Matriz de asignación
1	$V=\begin{Bmatrix}0,0,0,0,1\\0,0,0,0,0\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,2,2,1\}$	6	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,1,1,1\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,0,0,0\}$
2	$V=\begin{Bmatrix}0,0,0,1,1\\0,0,0,0,0\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,2,1,1\}$	7	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,1,1,1\\0,1,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,1,0,0,0\}$
3	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,0,0,0\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,1,1,1\}$	8	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,1,1,1\\1,1,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{1,1,0,0,0\}$
4	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,0,0,1\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,1,1,0\}$	9	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,1,1,1\\1,1,0,0,0\\0,1,0,0,0\end{Bmatrix}$ $X=\{1,0,0,0,0\}$
5	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,0,1,1\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,1,0,0\}$	10	$V=\begin{Bmatrix}0,0,1,1,1\\0,0,1,1,1\\1,1,0,0,0\\1,1,0,0,0\end{Bmatrix}$ $X=\{0,0,0,0,0\}$

Fig. 3.4: Resultados de una ejecución del algoritmo *TPM*

3.3 Asignación por copia-procesador (RP)

Este algoritmo se basa en asignar a la primera tarea en r procesadores diferentes, posteriormente sigue con la segunda tarea, y esto se realiza con todas las tareas mientras se cumpla con la condición de la expresión (3.1) que está restringida por la expresión (3.2).

Procedimiento RP: Dado un vector inicial de longitud n cuyo valor inicial en cada entrada es igual al número de copias por tarea, a este vector le denominaremos X, y un vector u ordenado en forma ascendente. Se toma la primera tarea más pesada y se asigna a cada procesador diferente hasta alcanzar el número de copias existente para la tarea en cuestión. Posteriormente se toma la siguiente tarea menos pesada y se repite la operación con los siguientes procesadores disponibles. Este proceso se repite hasta que no exista alguna tarea sin asignar. En la figura 3.5 se muestra un pequeño esquema del algoritmo.

```
Entrada:    func        : Función objetivo
            n x m       : Orden de la matriz de asignación
            r           : Número de copias
Salida:     V           : Matriz de asignación
Algoritmo:
    {
        Maxr = r;
        i = n;
        while (fulltasklist && i < n) {
            while (sizeproc >= 0.0) {
                if (u[j] <= sizeproc) {
                    sizeproc = sizeproc – u[j];
                    V[j,i] = 1;
                    Maxr[i] = Maxr[i] – 1;
                }
                else break;
                i = i - 1;
            }
        }
        sortbyprocs;
        while (fulltasklist) {
            V[getlowproc,i] = 1;
            Maxr[i] = Maxr[i] – 1;
            sortbyprocs;
        }
    }
```

Fig. 3.5: Esquema de "Copia-Procesador"

Considere un sistema compuesto de 4 procesadores y 5 tareas, cada tarea consta de 2 copias en total, el vector de utilización es: $u = \{0.20,0.30,0.60,0.68,0.79\}$. Evaluando los valores anteriores en la expresión (1.14) se obtiene el valor total de la carga del sistema, el cual es: $w_0 = 8.74$, y la carga promedio por procesador se obtiene a través de la expresión (1.15) obteniendo $\overline{w} = 2.185$, y el vector $X = \{2,2,2,2,2\}$. A continuación, en la figura 3.6, se muestra una ejecución con este algoritmo.

No. Paso	Matriz de asignación	No. Paso	Matriz de asignación
1	$V=\begin{Bmatrix}0,0,0,0,1\\0,0,0,0,0\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,2,2,1\}$	6	$V=\begin{Bmatrix}0,0,1,0,1\\0,0,1,0,1\\0,0,0,1,0\\0,0,0,1,0\end{Bmatrix}$ $X=\{2,2,0,0,0\}$
2	$V=\begin{Bmatrix}0,0,0,0,1\\0,0,0,0,1\\0,0,0,0,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,2,2,0\}$	7	$V=\begin{Bmatrix}0,0,1,0,1\\0,0,1,0,1\\0,1,0,1,0\\0,0,0,1,0\end{Bmatrix}$ $X=\{2,1,0,0,0\}$
3	$V=\begin{Bmatrix}0,0,0,0,1\\0,0,0,0,1\\0,0,0,1,0\\0,0,0,0,0\end{Bmatrix}$ $X=\{2,2,2,1,0\}$	8	$V=\begin{Bmatrix}0,0,1,0,1\\0,0,1,0,1\\0,1,0,1,0\\0,1,0,1,0\end{Bmatrix}$ $X=\{2,0,0,0,0\}$
4	$V=\begin{Bmatrix}0,0,0,0,1\\0,0,0,0,1\\0,0,0,1,0\\0,0,0,1,0\end{Bmatrix}$ $X=\{2,2,2,0,0\}$	9	$V=\begin{Bmatrix}1,0,1,0,1\\0,0,1,0,1\\0,1,0,1,0\\0,1,0,1,0\end{Bmatrix}$ $X=\{1,0,0,0,0\}$
5	$V=\begin{Bmatrix}0,0,1,0,1\\0,0,0,0,1\\0,0,0,1,0\\0,0,0,1,0\end{Bmatrix}$ $X=\{2,2,1,0,0\}$	10	$V=\begin{Bmatrix}1,0,1,0,1\\1,0,1,0,1\\0,1,0,1,0\\0,1,0,1,0\end{Bmatrix}$ $X=\{0,0,0,0,0\}$

Fig. 3.6: Resultados de una ejecución del algoritmo *RP*

4. Pruebas experimentales

Se desarrollaron los métodos de optimización presentados en el capítulo anterior en lenguaje C y ejecutados en una misma plataforma para poder comparar su desempeño.

Para poder mencionar a los algoritmos de una manera más sencilla, se renombrará a cada método de optimización de la siguiente manera: como AG al algoritmo genético, como RS al recocido simulado y por último, a la búsqueda local como BL. En los heurísticos definiremos a H1 para tarea-procesador-maxmin, H2 se utilizará para tarea-procesador-máximo y finalmente H3 para copia-procesador.

Parámetros utilizados

Los parámetros utilizados para la simulación son:

Número de tareas involucradas (n) = 7
Número de procesadores existentes (m) = 14
Número de copias por tarea (r) = 3
Ponderaciones: a = 1200, b = 200 , c = 700
Distancia de la vecindad (ε) = 0.13
Vecindad (k) = 3
El vector de pesos de las tareas (u_j) está dada por las siguientes expresiones:

$$u_j = 1t \tag{4.1}$$

$$u_j = 1 - \left(\frac{j}{n}\right)^2 \tag{4.2}$$

$$u_j = \left(\frac{j}{n}\right)^2 \tag{4.3}$$

$$u_j = \left(1 - \frac{j}{n}\right)^2 \tag{4.4}$$

$$u_j = 1 - \left(1 - \frac{j}{n}\right)^2 \tag{4.5}$$

$$u_j = 1 - \frac{j}{n} \tag{4.6}$$

$$u_j = \frac{j}{n} \tag{4.7}$$

$$u_j = 0.02\left(j - \frac{n}{2}\right)^2 \tag{4.8}$$

$$u_j = 1 - 0.02\left(j - \frac{n}{2}\right)^2 \tag{4.9}$$

$$u_j = \frac{\tan\left(\frac{\pi(j-1)}{2(n-1)}\right)}{1 + \tan\left(\frac{\pi(j-1)}{2(n-1)}\right)} \tag{4.10}$$

$$u_j = 1 - \frac{\tan\left(\frac{\pi(j-1)}{2(n-1)}\right)}{1 + \tan\left(\frac{\pi(j-1)}{2(n-1)}\right)} \tag{4.11}$$

donde $j \in [0, n]$ y $t = 0.35$

Cada una de estas expresiones generan diferentes vectores de carga de trabajo, por ejemplo: en la expresión(4.3) se tiene que la mayoría de las entradas del vector

representan una carga mayor del 80%; mientras que en la expresión (4.2) se representa el caso contrario. Para la expresión (4.10), en la mayor parte de las entradas representan un promedio del 50% de la carga para el sistema.

Cada vector de tareas corresponde a una simulación diferente, que es ejecutada por cada método (RS, AG, BL y heurísticos).

Como se puede apreciar, existen varios vectores de tareas (u's) con los cuales se desea analizar el comportamiento de los algoritmos involucrados. En este caso tendremos 11 simulaciones diferentes por cada método de solución.

4.1 Comparación de los algoritmos RS, genético y heurístico

En las siguientes figuras, de la 4-1 a la 4-4, se pueden observar los resultados y las comparaciones de los algoritmos AG, RS y BL; así mismo, de la 4-5 a la 4-8 se observan los algoritmos heurísticos.

En cada una de estas figuras, en el eje de las abscisas se presenta el resultado o la comparación de cada simulación utilizando cada una de las expresiones indicadas anteriormente, es decir, donde *S00* corresponde al resultado con el vector de tareas calculado por la expresión (4.1), el *S01* con la expresión (4.2) y así sucesivamente.

Comparación de los algoritmos BL, RS y AG

En la figura 4-1 se muestra el resultado obtenido de la evaluación de la función objetivo en cada una de las simulaciones; la expresión (1.9) para los algoritmos AG y BL, y la expresión (1.11) para RS. En la ordenada se muestra la escala de valores que se puede obtener al evaluar a la función objetivo, en esta gráfica dichos valores son adimensionales. Recordando que el objetivo es minimizar la función objetivo, podemos observar que el

algoritmo BL es el que mejor cumple con esta meta, y que el algoritmo que menos la cumple es RS. Aunque en algunos casos, como se observa en *S04*, *S05* y *S09* el peor algoritmo es AG. De lo que se deduce que el mejor algoritmo es BL y que el peor es RS.

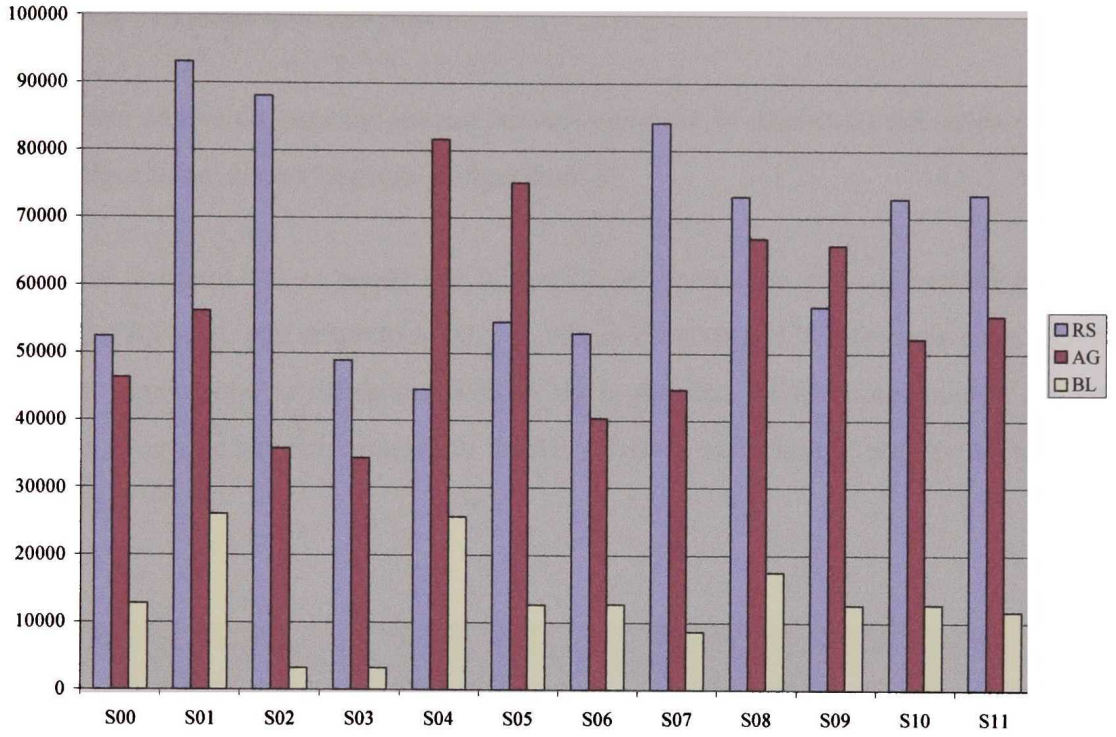

Fig. 4-1: Comparación en el resultado de la función objetivo por cada algoritmo (BL, RS y AG)

Cabe mencionar que esto es sólo dentro de este problema y con los valores mencionados anteriormente, en ningún momento se puede generalizar este comportamiento. Muy posiblemente con otros valores puede variar el comportamiento de cada uno de los algoritmos.

Como se puede observar en la figura 4-1, de los algoritmos utilizados se obtienen valores totalmente diferentes; por lo que se realiza una comparación siguiendo la expresión (4.12) como criterio y mostrando los resultados en la figura 4-2, para AG, BL y RS; y en la figura 4-6 para H1, H2 y H3.

$$C_B(A_1, A_2) = \left| \frac{f(u_{A_1}) - f(u_{A_2})}{f(u_{A_2})} \right| \tag{4.12}$$

donde: A_1 : Cualquier algoritmo menos el genético

A_2 : Algoritmo genético

Esta expresión permite obtener el porcentaje de la diferencia del valor obtenido por los algoritmos A_1 con respecto al algoritmo A_2.

En la figura 4-2 se puede ver la gráfica del porcentaje de la diferencia entre los algoritmos RS y BL con respecto al AG, en ella se observa que la diferencia entre AG-RS es mucho menor que la diferencia AG-BL en la mayoría de las simulaciones. De esto podemos decir que el comportamiento de RS y AG es casi similar, aunque no se puede generalizar.

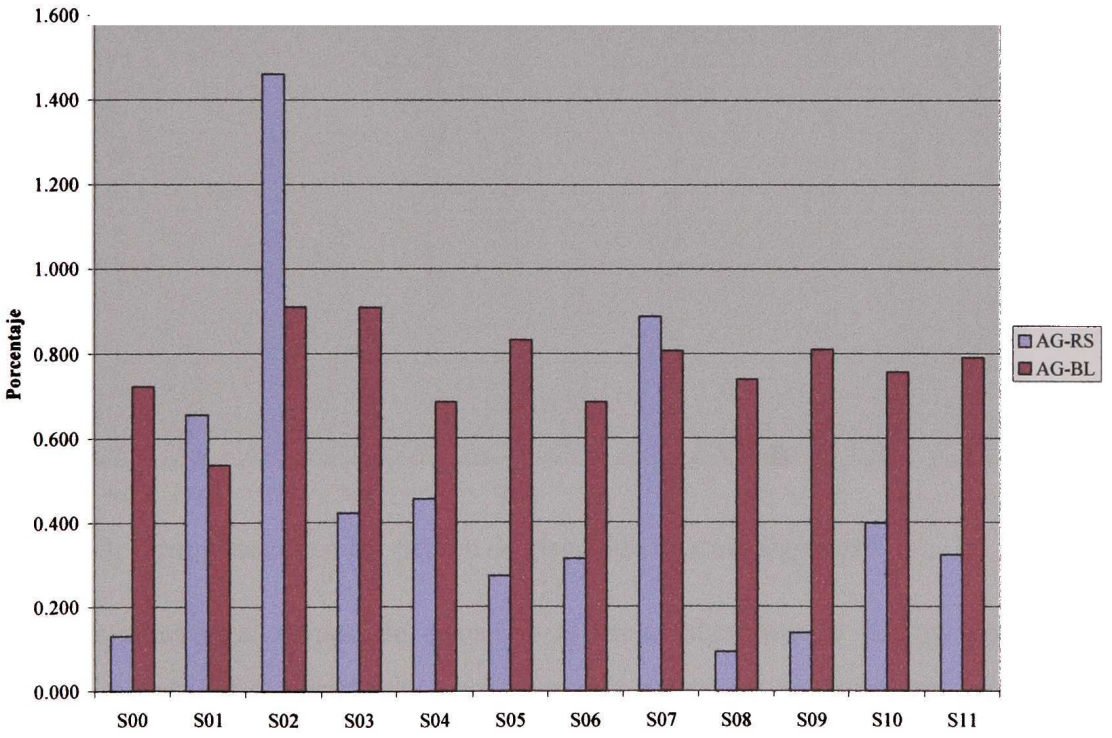

Fig. 4-2: Comparación utilizando la expresión (4.12)

Otro factor que se utiliza para la comparación es el tiempo de ejecución de cada algoritmo, el cual se representa en la figura 4-3. En dicha figura RS tiene el mismo tiempo de ejecución en cada una de las simulaciones, al igual que BL; sin embargo AG tiene diferentes tiempos de ejecución. En el caso de BL, el criterio de terminación es que no exista alguna variación en el resultado de la función objetivo. Para RS los criterios son: el número de iteraciones o que la función de enfriamiento pase cierto umbral; para estas 11 simulaciones siempre se cumplió el número de iteraciones. Pero en AG los criterios de terminación son: que no exista una población que minimice la función objetivo, o que se de el máximo número de generaciones permitidas; dentro de estas 11 simulaciones se dio el primer criterio, es por esto que el tiempo varía en este algoritmo.

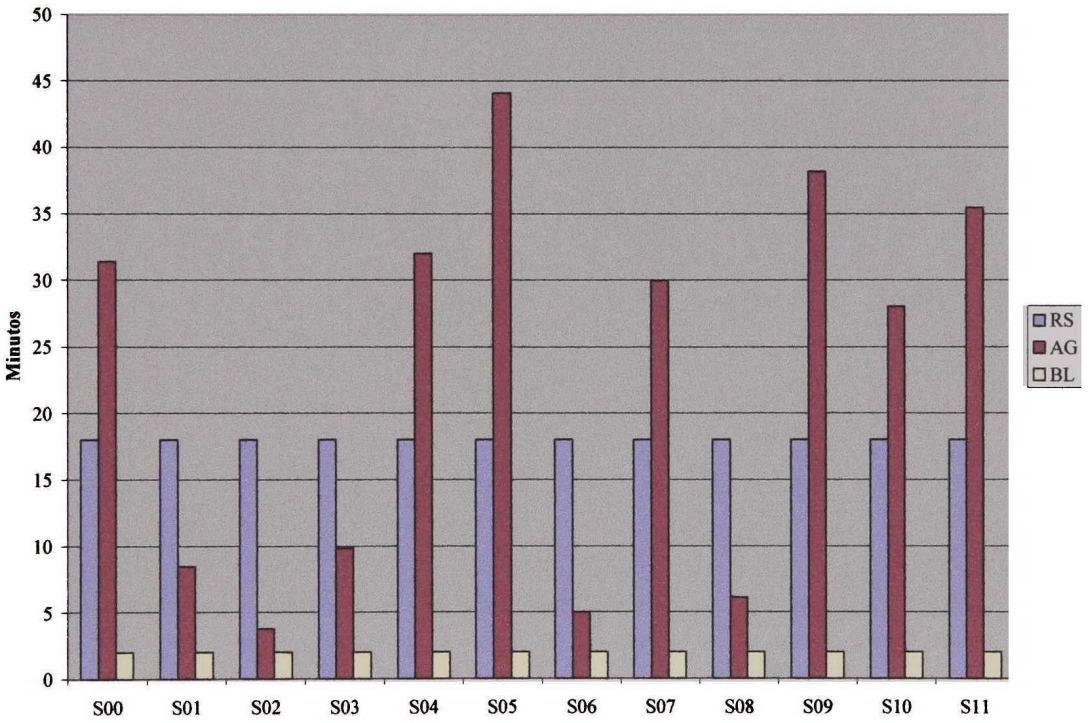

Fig. 4-3: Comparación en los tiempo de ejecución de cada algoritmo (BL, RS y AG)

De acuerdo al tiempo de ejecución el mejor algoritmo es BL, ya que sólo se necesitan menos de 5 minutos para obtener una respuesta; mientras que en 7 de las simulaciones el que requiere de mayor tiempo es AG, y en las 5 simulaciones restantes es RS quien demanda de mayor tiempo. De aquí podemos decir que el algoritmo más rápido

es BL y el más lento es AG. Además, este último tiene un consumo de tiempo no tan fácil de predecir, a diferencia de BL y RS.

En la figura 4-4 se muestra la aceleración de los algoritmos, es decir, que tan rápidos son los algoritmos con respecto a uno de ellos. Es este caso se analiza la velocidad de RS y BL con respecto a la de AG. El resultado se muestra en la figura 4-4, en la que se puede ver que BL es casi 23 veces más rápido que AG en el mejor caso y menos de 5 veces en el peor caso. También se observar que RS es a lo más 3 veces más rápido que AG.

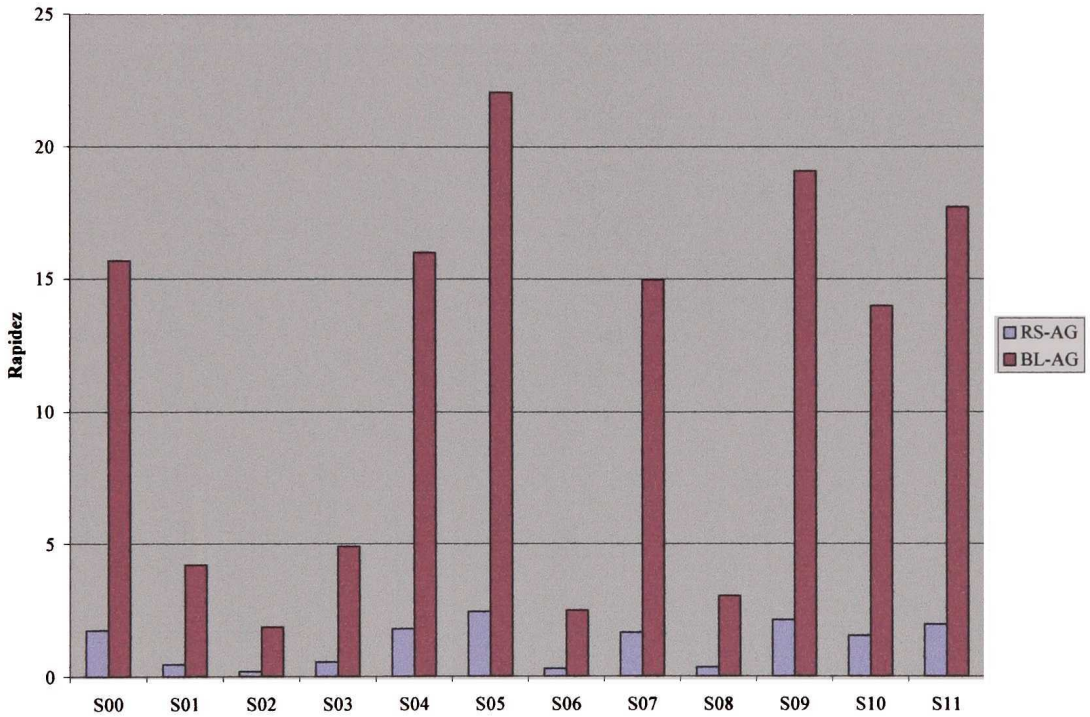

Fig. 4-4: Comparación de la aceleración de los algoritmos

De las comparaciones realizadas, el algoritmo BL tiene mejor rendimiento que AG y RS, tanto por su velocidad como por el hecho de minimizar la función objetivo.

Comparación de los algoritmos heurísticos (H1, H2 y H3) y AG

En la figura 4-5 se muestra el resultado obtenido de la evaluación de la función objetivo en cada una de las simulaciones, expresión (1.9) para los algoritmos AG, H1, H2 y H3. La ordenada representa la escala de valores que se pueden obtener de la función objetivo; en esta gráfica dichos valores son adimensionales. Recordando que el objetivo es minimizar la función objetivo, el algoritmo H3 es el que mejor cumple con esta meta, y el algoritmo que menos se acerca es H2. Por lo que el mejor algoritmo es H3 y que el peor es H2. La finalidad de incluir en la gráfica a AG es para poder comparar los heurísticos con los otros algoritmos.

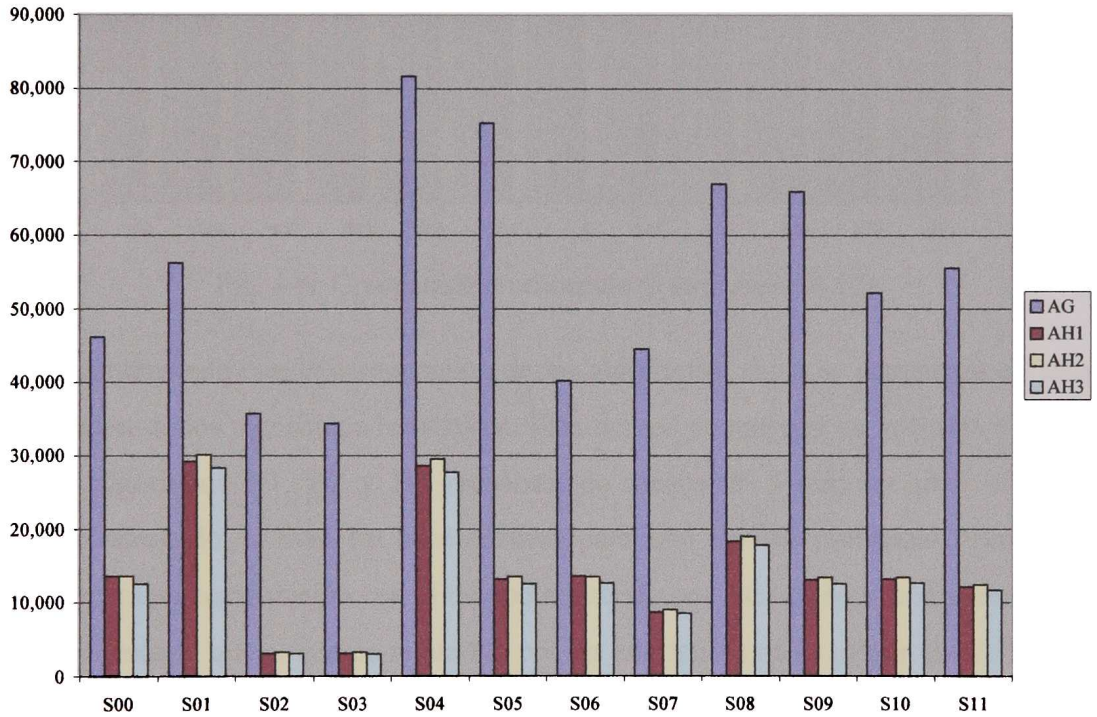

Fig. 4-5: Comparación en el resultado de la función objetivo por cada algoritmo (AG, H1, H2 y H3)

La figura 4-6 representa la gráfica del porcentaje de la diferencia entre los algoritmos H1, H2 y H3 con respecto al AG., en la cual se observa que la diferencia entre los algoritmos H1, H2 y H3 es casi mínima, aunque se puede apreciar perfectamente que el algoritmo H3 tiene los valores mas pequeños y que el H2 es el que tiene los valores más

grandes. Por lo tanto, H1, H2 y H3, tienen el mismo comportamiento.

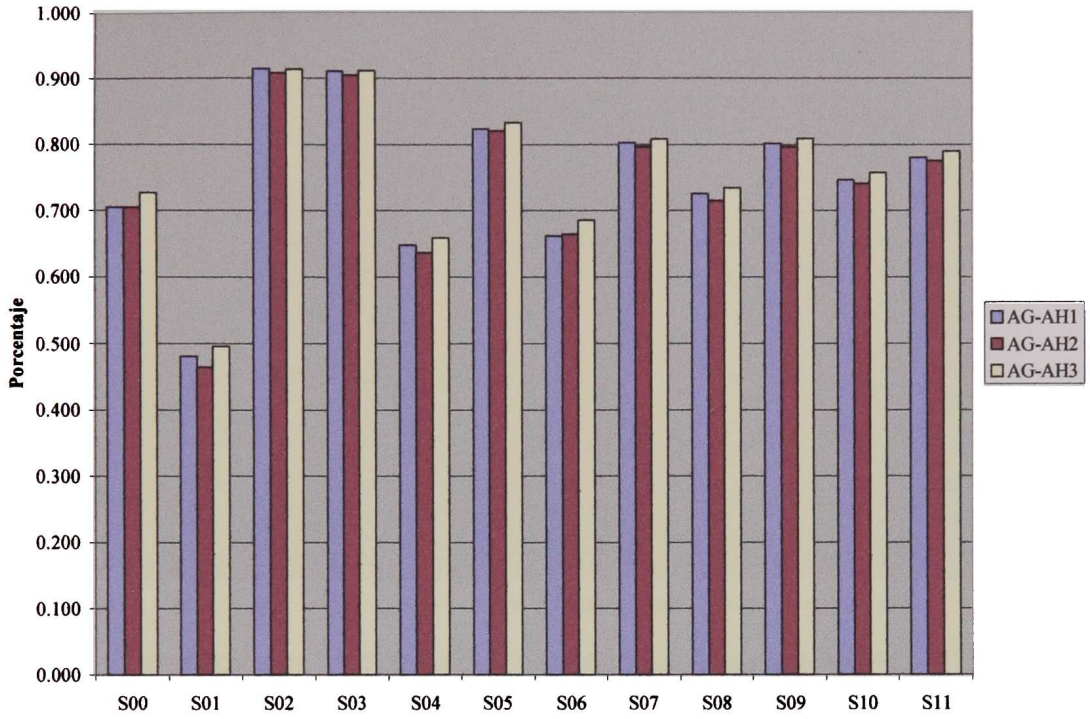

Fig. 4-6: Comparación utilizando la expresión (4.12)

Continuando con la comparación de los algoritmos, ahora se comentará el tiempo de ejecución de los algoritmos heurísticos. Este tiempo se muestra en la figura 4-7, en la cual los algoritmos H1, H2 y H3 requieren de menos de 3 minutos para obtener el resultado adecuado, lo cual los hace idóneos para este tipo de problemas. También se puede hacer una comparación con el tiempo requerido por AG; de estas comparaciones es relevante destacar que el tiempo requerido por los heurísticos no es mayor que AG.

De acuerdo al tiempo de ejecución los tres métodos heurísticos son muy rápidos para encontrar la solución del problema.

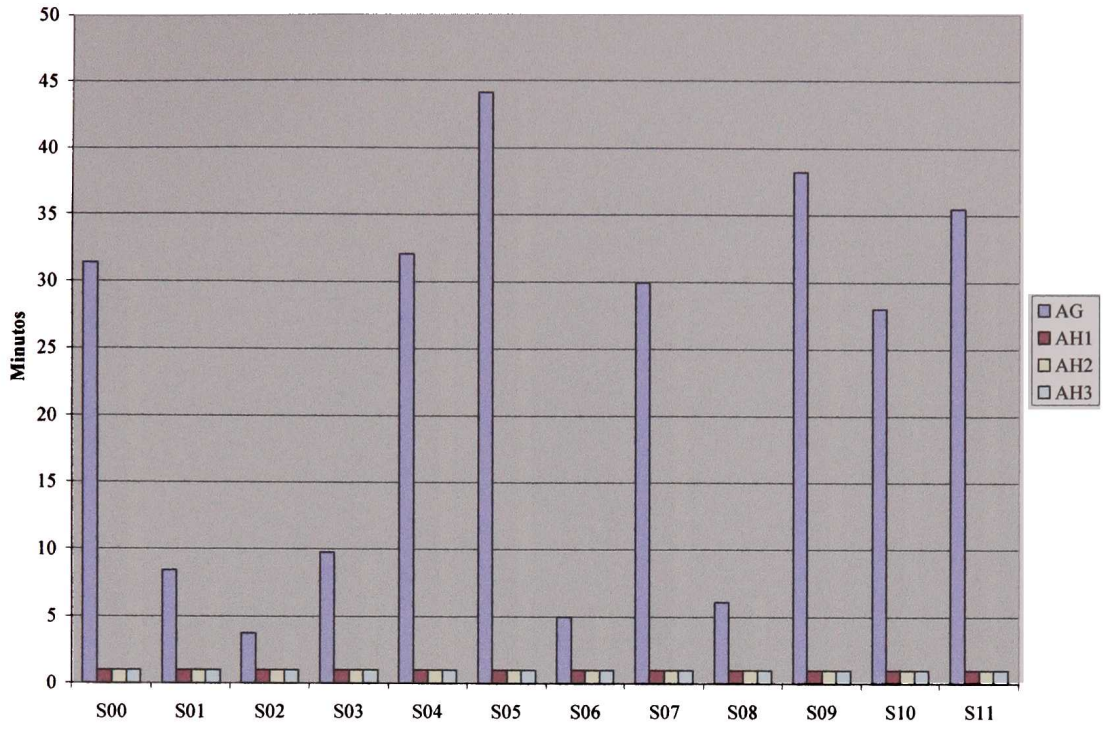

Fig. 4-7: Comparación en los tiempo de ejecución de cada algoritmo (AG, H1, H2 y H3)

La figura 4-8 muestra la aceleración de los algoritmos, es decir, que tan rápidos son los algoritmos heurísticos con respecto a AG. Los heurísticos, con respecto a AG, son hasta 44 veces más rápidos en el mejor de los casos y en la peor situación sólo 4 veces. Como se puede vislumbrar en la mayoría de las simulaciones, los algoritmos heurísticos son por lo menos 25 veces más rápidos en promedio que AG.

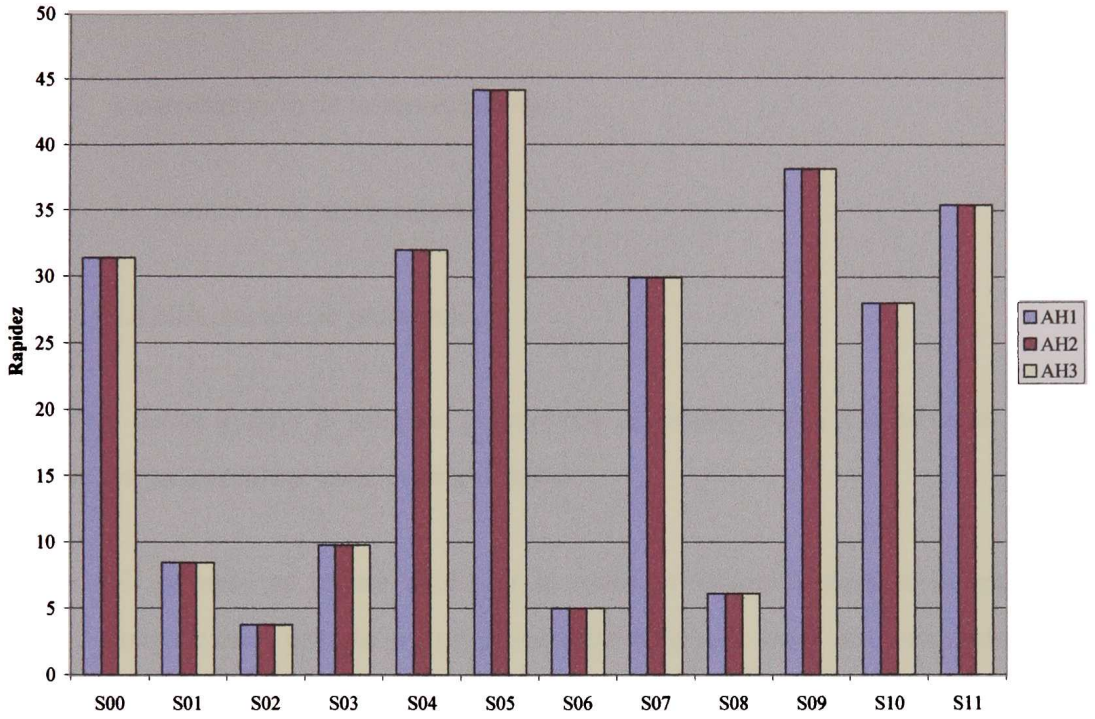

Fig. 4-8 Comparación de la aceleración de los algoritmos

De acuerdo con lo anterior los heurísticos son algoritmos que tienen un buen rendimiento por su velocidad, ya que los tres algoritmos consumen el mismo tiempo en su ejecución. En lo que se refiere a minimizar la función objetivo, es claro que el algoritmo H3 es el que mejor cumple con esta meta y que el H1 es el que peor la realiza; aunque el resultado de los dos algoritmos es aceptable.

4.2 Tratamiento de perturbaciones a los problemas

Dentro del ambiente de cómputo distribuido existen factores que perturban un sistema distribuido, afectando directamente su rendimiento, para ello es necesario que el sistema tenga la capacidad de recuperarse ante estas eventualidades. En este capítulo se analizará el resultado de los factores mencionados en el primer capítulo:

- La creación de tareas
- La terminación de tareas existentes
- La inserción de procesadores
- La eliminación de procesadores

Para llevar a cabo la solución y preservar el rendimiento del sistema se sugieren dos enfoques para resolver estas eventualidades:

- El primero se refiere a utilizar la matriz V obtenida durante el proceso de asignación y pasarla por un proceso de búsqueda local, una vez perturbado el escenario.
- El segundo enfoque se basa en la obtención la matriz de asignación V sin depender del resultado previo obtenido de cada algoritmo.

Aquí se comparan los dos enfoques, tanto en los resultados obtenidos por la función objetivo, como en su tiempo de ejecución y en la aceleración. En lo que se refiere a la aceleración la comparación se realiza entre el primer y el segundo enfoque de solución.

Para el primer enfoque se utilizan los resultados obtenidos por los algoritmos estudiados (AG, RS, BL, H1, H2 y H3). Para el segundo enfoque se emplea el algoritmo H3, ya que presenta el mejor comportamiento tiene para minimizar la función objetivo. Para llevar a cabo las comparaciones, se utilizan los resultados obtenidos por el algoritmo H3.

Falla de un procesador

La figura 4-9 se muestran los resultados obtenidos utilizando el primer enfoque en cada uno de los algoritmos: AG, RS, BL, H1, H2 y H3. El comportamiento del primer enfoque es casi igual para todos los algoritmos, pero con una ligera inclinación hacia los heurísticos. En el eje de las ordenadas se representa el valor obtenido de la evaluación de la función objetivo una vez que se ha pasado por el primer enfoque, este valor es adimensional.

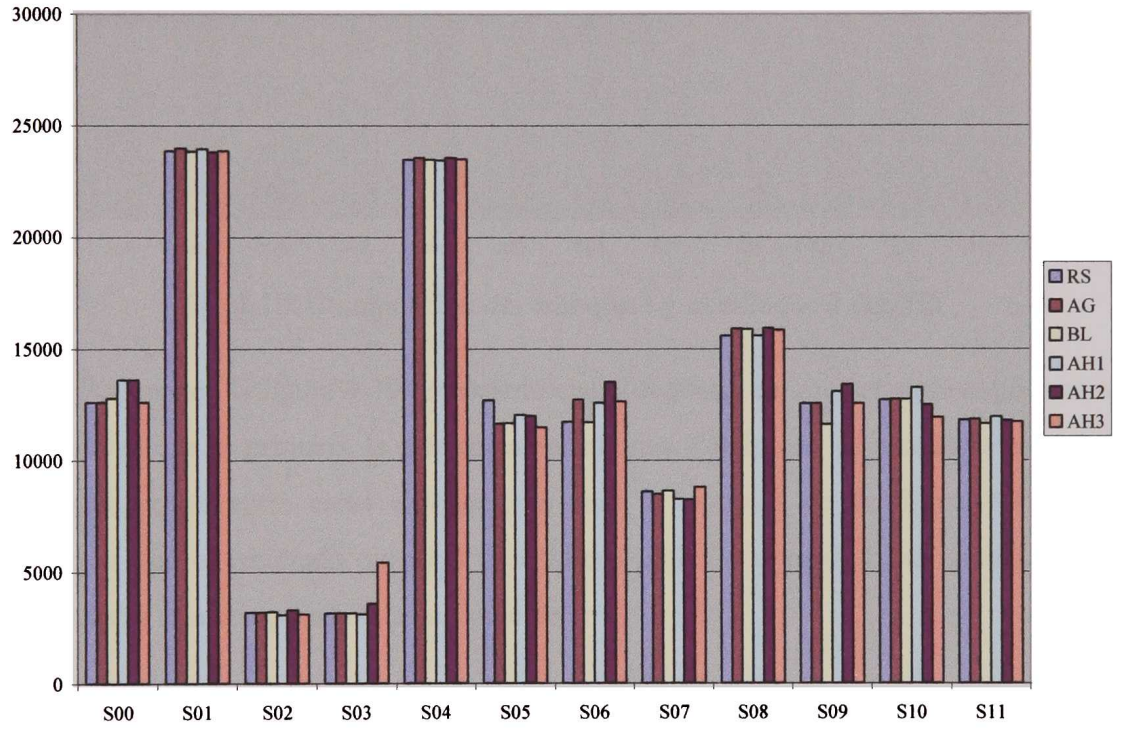

Fig. 4-9: Comparación de la función objetivo por algoritmo utilizando el primer enfoque

En la figura 4-10 se presentan los resultados obtenidos por el algoritmo H3, tanto por el primer enfoque como por el segundo. El segundo enfoque es mucho mejor que el primero. Nuevamente el eje de las ordenadas es adimensional.

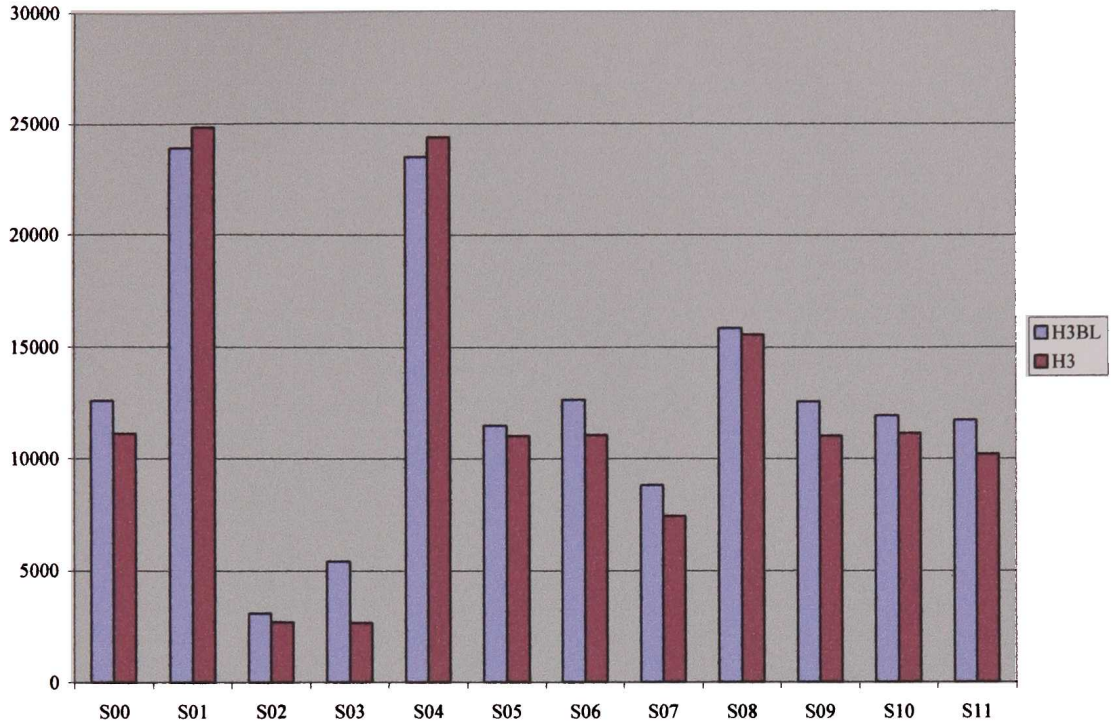

Fig. 4-10: Comparación del enfoque 1 y el enfoque 2 con H3

Aunque en la figura 4-10 se advierte que el segundo enfoque cumple mejor con las expectativas que el primero, la diferencia es mínima. Por lo que es necesario obtener la diferencia exacta entre estos enfoques, la cual se presenta en la figura 4.11. Cabe mencionar que el resultado obtenido es adimensional. La expresión que se utiliza para llevar a cabo esta comparación es la siguiente:

$$\partial(A_1, A_2) = f(V_{A_1}) - f(V_{A_2}) \tag{4.13}$$

donde: A_1 : El primer enfoque

A_2 : El segundo enfoque

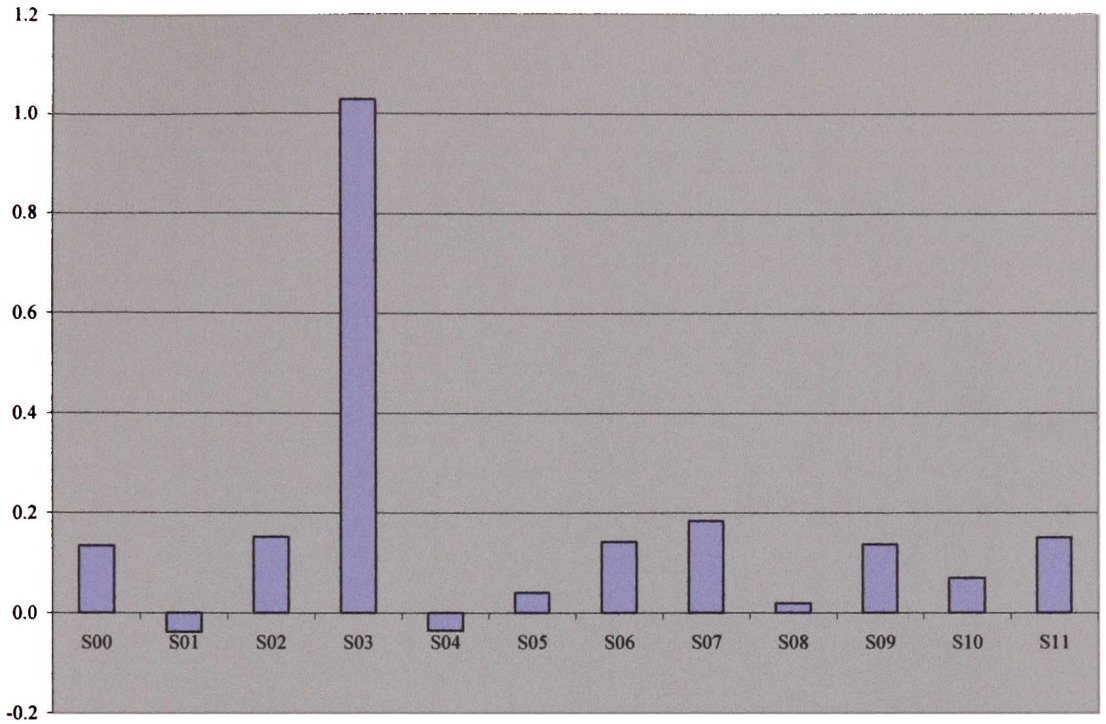

Fig. 4-11: Comparación del error relativo entre los enfoques utilizando H3

Siguiendo con las comparaciones, el tiempo utilizado por el primer enfoque sólo necesita de 1 minuto para obtener el resultado, y el segundo enfoque requiere del tiempo que consume el algoritmo utilizado, en este caso que se compara con el algoritmo H3 que también requiere de un minuto para obtener el resultado, la aceleración es 1, tal como se presenta en la figura 4.12.

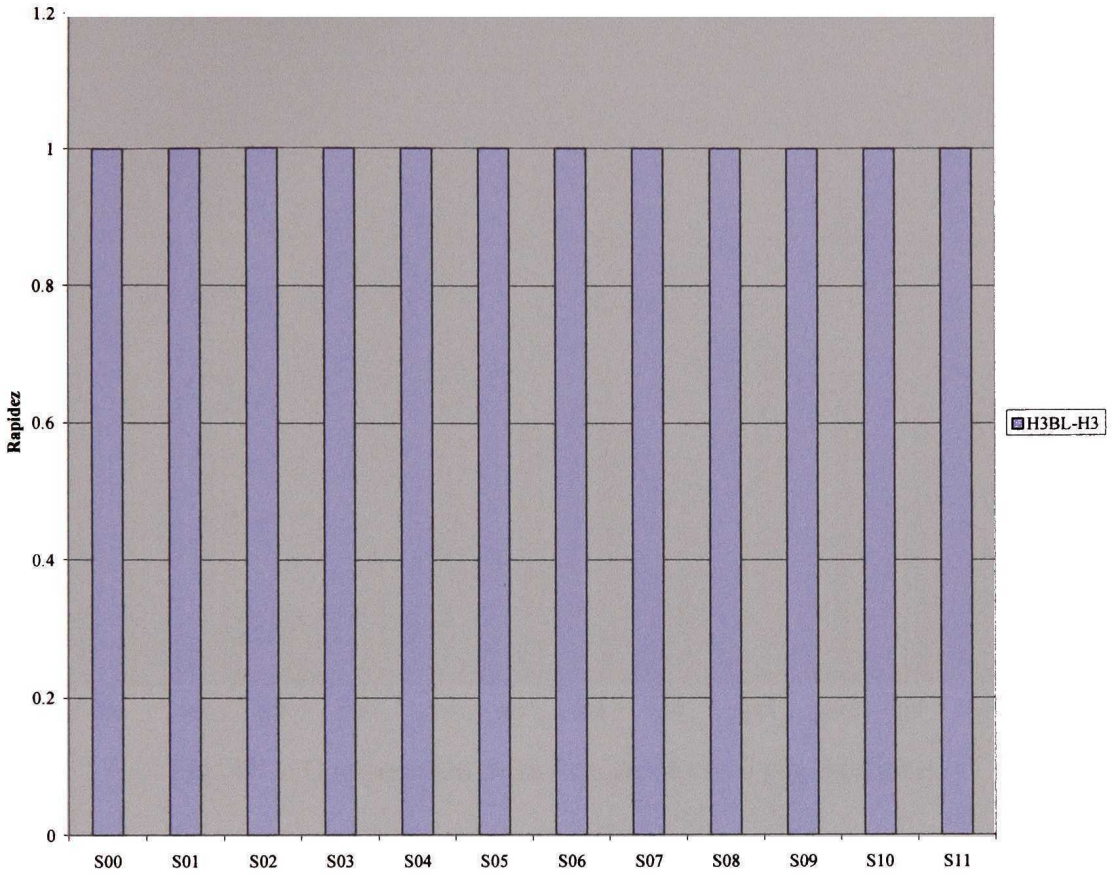

Fig. 4-12: Comparación de aceleración entre los dos enfoques

Terminación de una tarea

La figura 4-13 muestra los resultados obtenidos utilizando el primer enfoque en cada uno de los algoritmos, AG, RS, BL, H1, H2 y H3. Se puede observar que el comportamiento del primer enfoque es casi igual para todo los algoritmos, pero con una ligera inclinación hacia los heurísticos. En el eje de las ordenadas tenemos el valor obtenido de la evaluación de la función objetivo una vez pasado por el primer enfoque, este valor es adimensional.

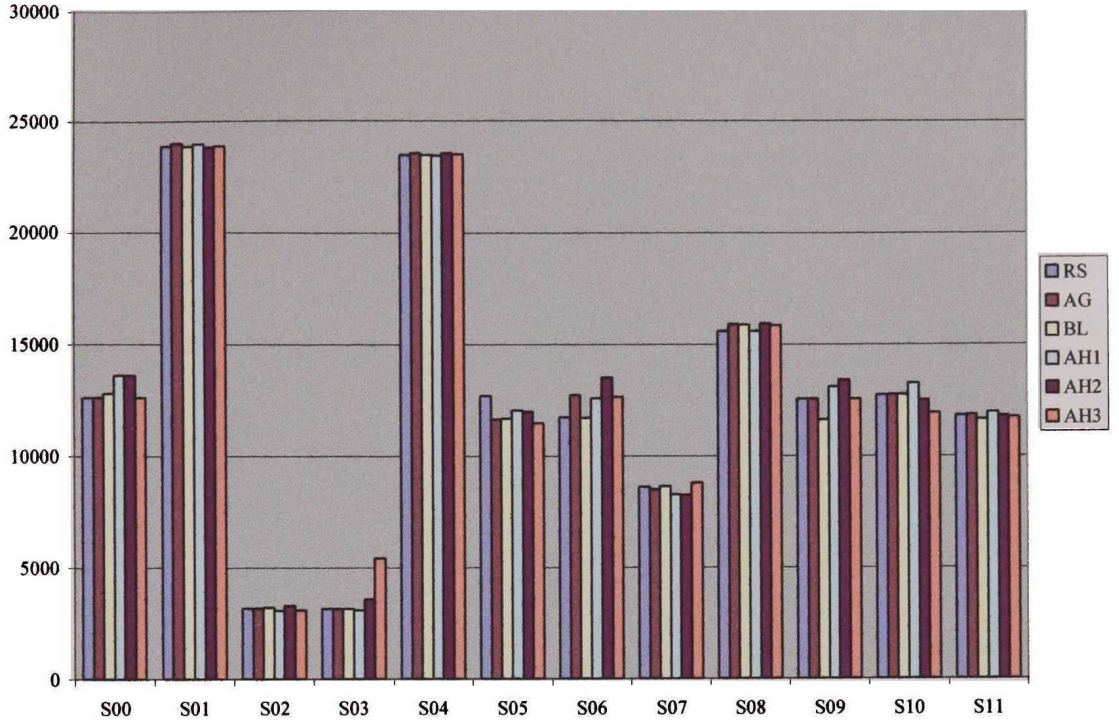

Fig. 4-13: Comparación de la función objetivo por algoritmo

En la figura 4-14 se presentan los resultados obtenidos por el algoritmo H3, tanto por el primer enfoque como por el segundo. El segundo enfoque es mucho mejor que el primero. Nuevamente el eje de las ordenadas es adimensional.

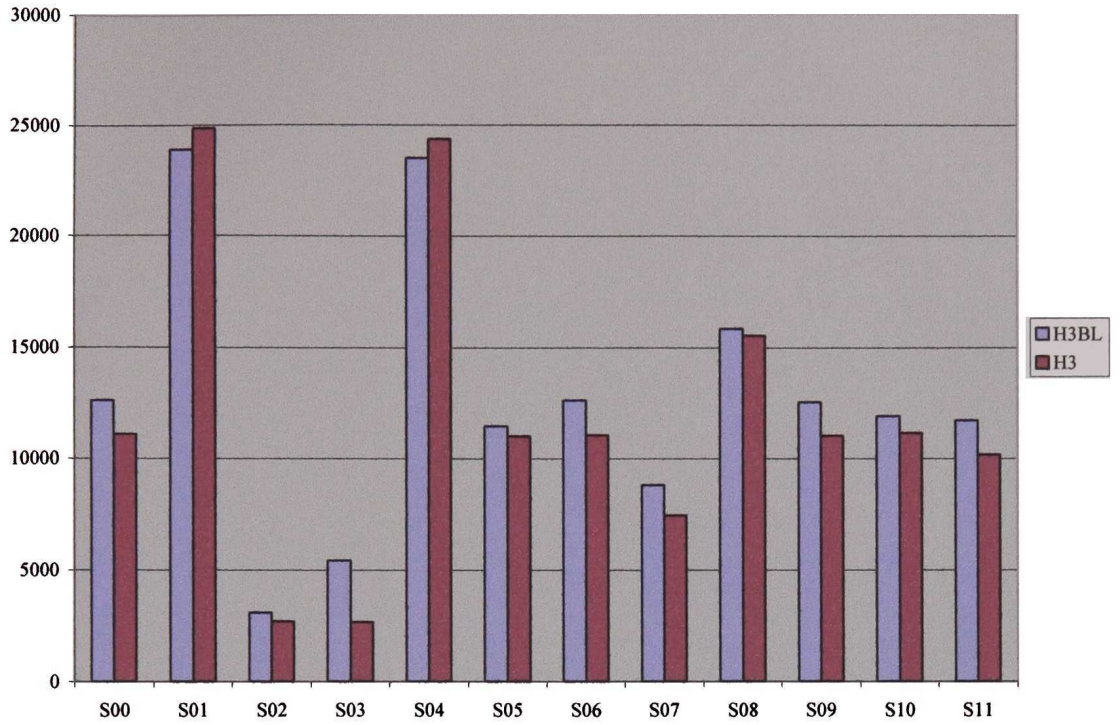

Fig. 4-14: Comparación del enfoque 1 y el enfoque 2 con H3

Aunque en la figura 4-14 se advierte que el segundo enfoque cumple mejor que las expectativas que el primero, la diferencia es mínima. Por lo que es necesario obtener la diferencia exacta entre estos enfoques, la cual se presenta en la figura 4.15. Cabe mencionar que el resultado obtenido es adimensional. Se utiliza la expresión (4.13) para llevar a cabo esta comparación.

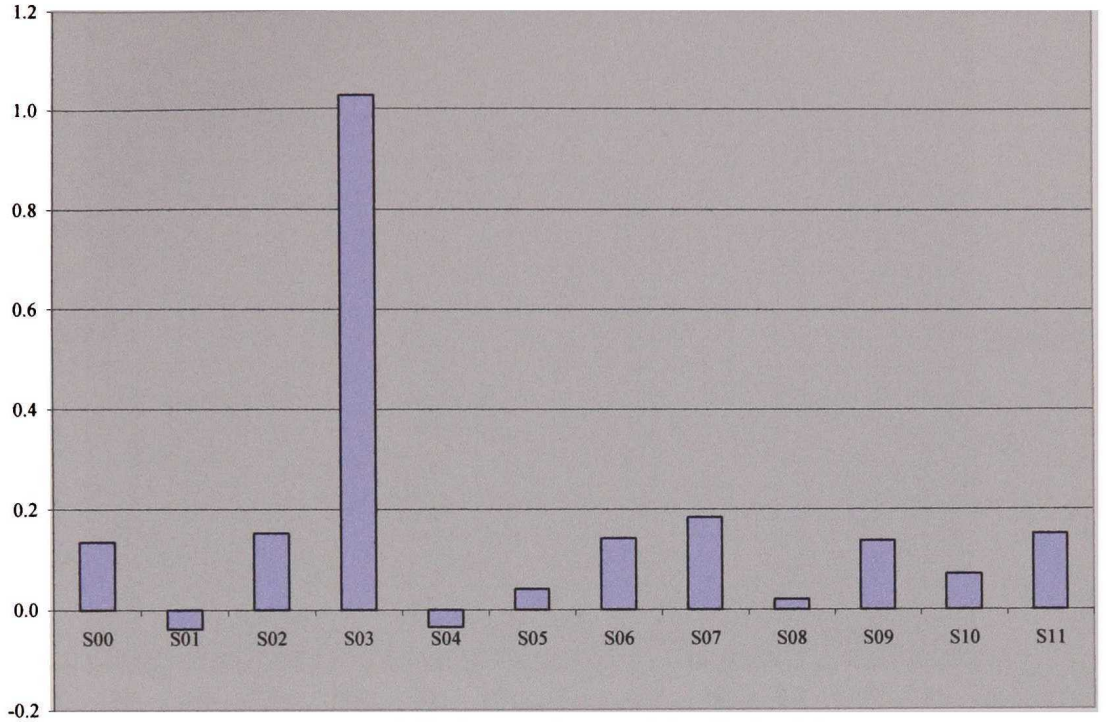

Fig. 4-15: Comparación del error relativo entre los enfoques utilizando H3

Siguiendo con las comparaciones, el tiempo utilizado por el primer enfoque sólo necesita de 1 minuto para obtener el resultado, y el segundo enfoque requiere del tiempo que consume el algoritmo utilizado, en este caso que se compara con el algoritmo H3 que también requiere de un minuto para obtener el resultado, la aceleración es 1, tal como se observa en la figura 4.16.

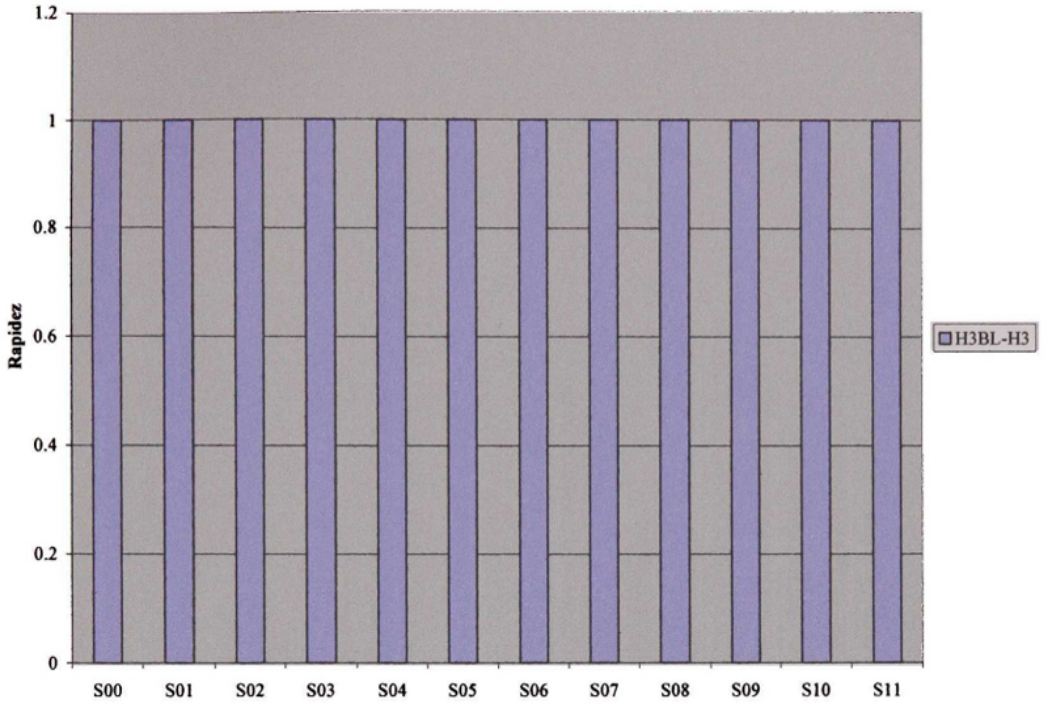

Fig. 4-16: Comparación de aceleración entre los dos enfoques

Inserción de una tarea

La figura 4-17 muestra los resultados obtenidos utilizando el primer enfoque en cada uno de los algoritmos, AG, RS, BL, H1, H2 y H3. Se puede observar que el comportamiento del primer enfoque es casi igual para todo los algoritmos, pero con una ligera inclinación hacia los heurísticos. En el eje de las ordenadas tenemos el valor obtenido de la evaluación de la función objetivo una vez pasado por el primer enfoque, este valor es adimensional.

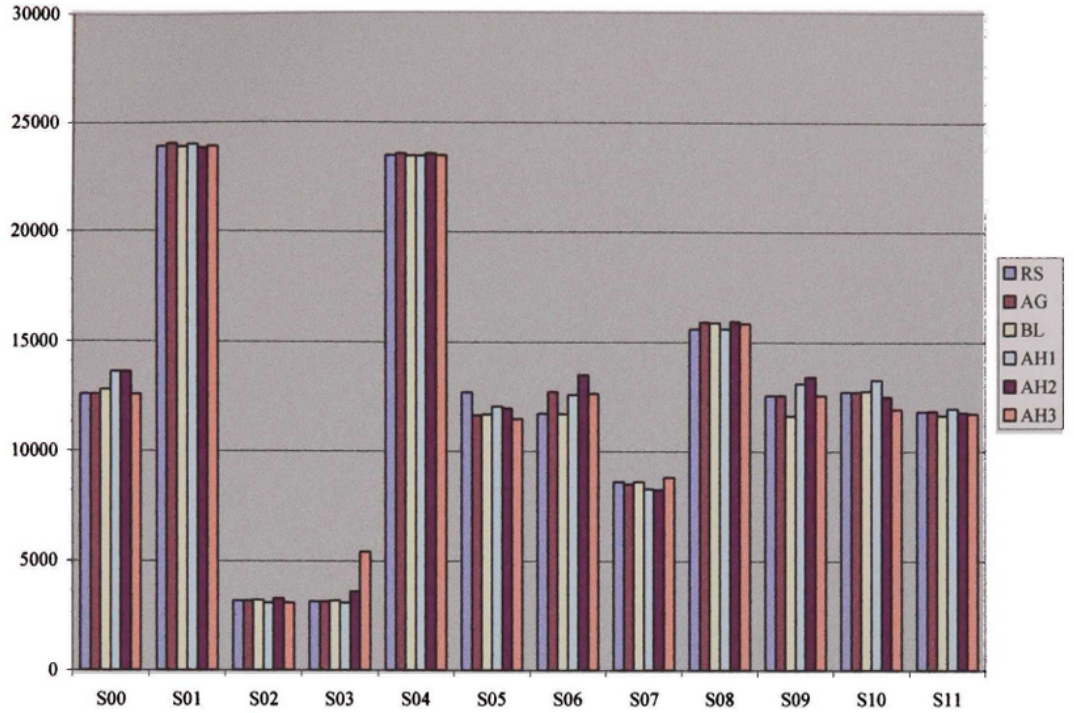

Fig. 4-17: Comparación de la función objetivo por algoritmo

En la figura 4-18 se presentan los resultados obtenidos por el algoritmo H3, tanto por el primer enfoque como por el segundo. El segundo enfoque es mucho mejor que el primero. Nuevamente el eje de las ordenadas es adimensional.

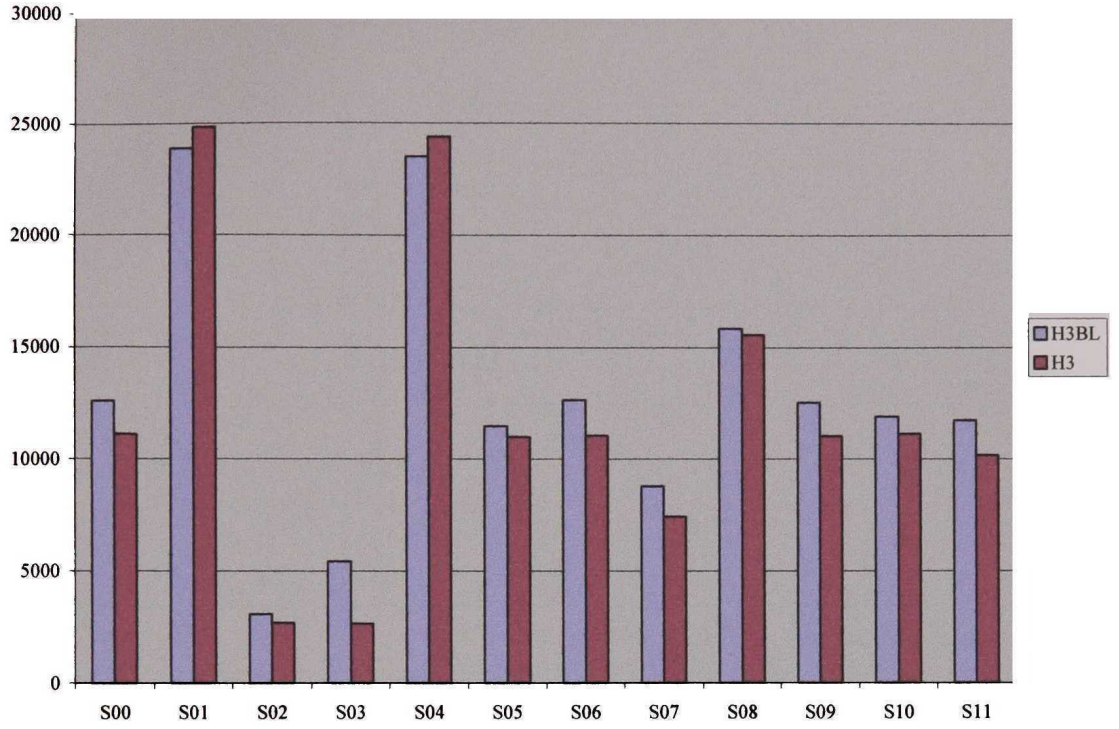

Fig. 4-18: Comparación del enfoque 1 y el enfoque 2 con H3

Aunque en la figura 4-18 se advierte que el segundo enfoque cumple mejor que las expectativas que el primero, la diferencia es mínima. Por lo que es necesario obtener la diferencia exacta entre estos enfoques, la cual se presenta en la figura 4.19. Cabe mencionar que el resultado obtenido es adimensional. Se utiliza la expresión (4.13) para llevar a cabo esta comparación.

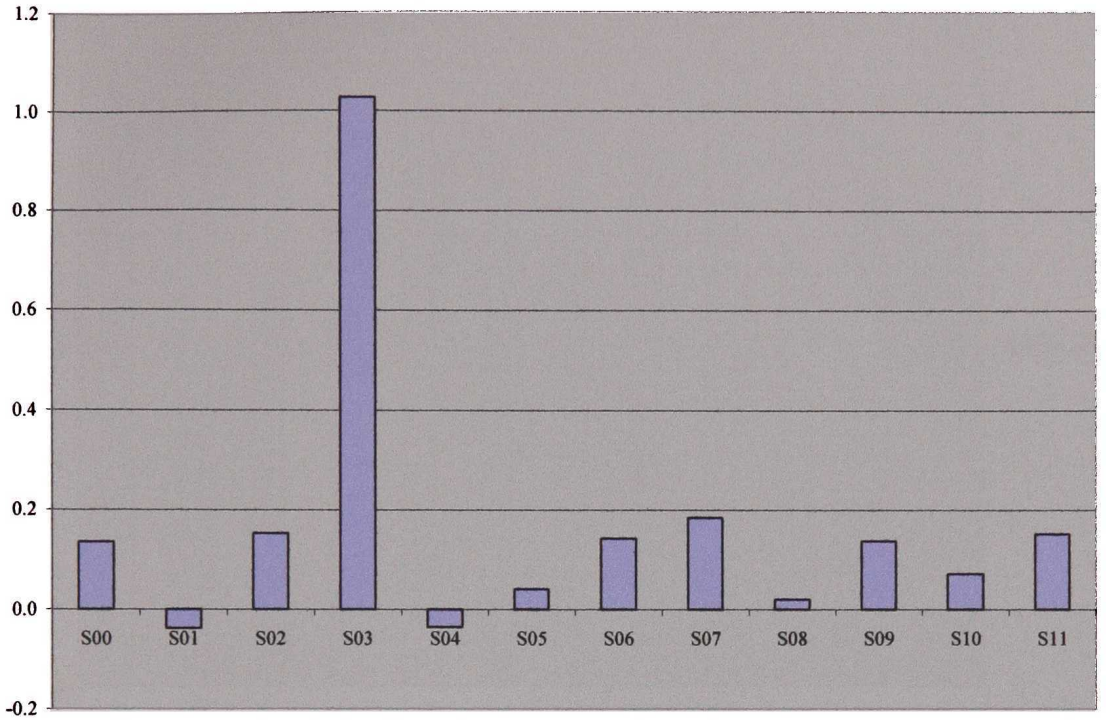

Fig. 4-19: Comparación del error relativo entre los enfoques utilizando H3

Siguiendo con las comparaciones, el tiempo utilizado por el primer enfoque sólo necesita de 1 minuto para obtener el resultado, y el segundo enfoque requiere del tiempo que consume el algoritmo utilizado, en este caso que se compara con el algoritmo H3 que también requiere de un minuto para obtener el resultado, la aceleración es 1, tal como se observa en la figura 4.20.

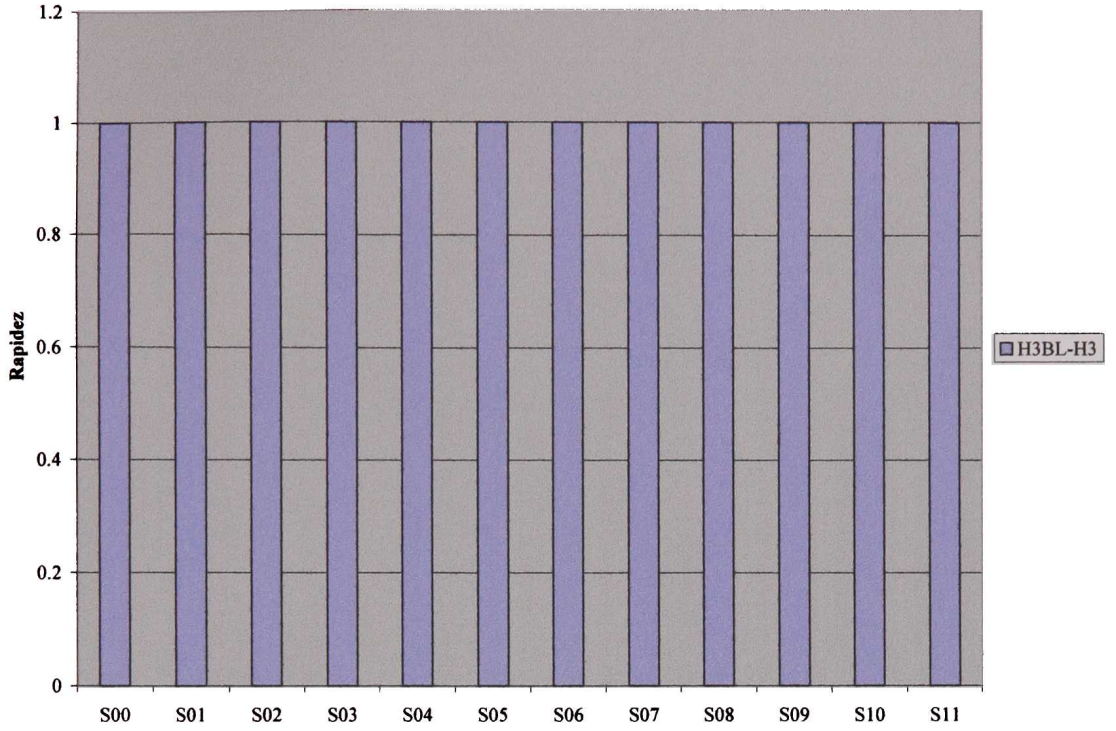

Fig. 4-20: Comparación de aceleración entre los dos enfoques

Inserción de un procesador

La figura 4-21 se muestran los resultados obtenidos utilizando el primer enfoque en cada uno de los algoritmos, AG, RS, BL, H1, H2 y H3. Se puede observar que el comportamiento del primer enfoque es casi igual para todo los algoritmos, pero con una ligera inclinación hacia los heurísticos. En el eje de las ordenadas tenemos el valor obtenido de la evaluación de la función objetivo una vez pasado por el primer enfoque, este valor es adimensional.

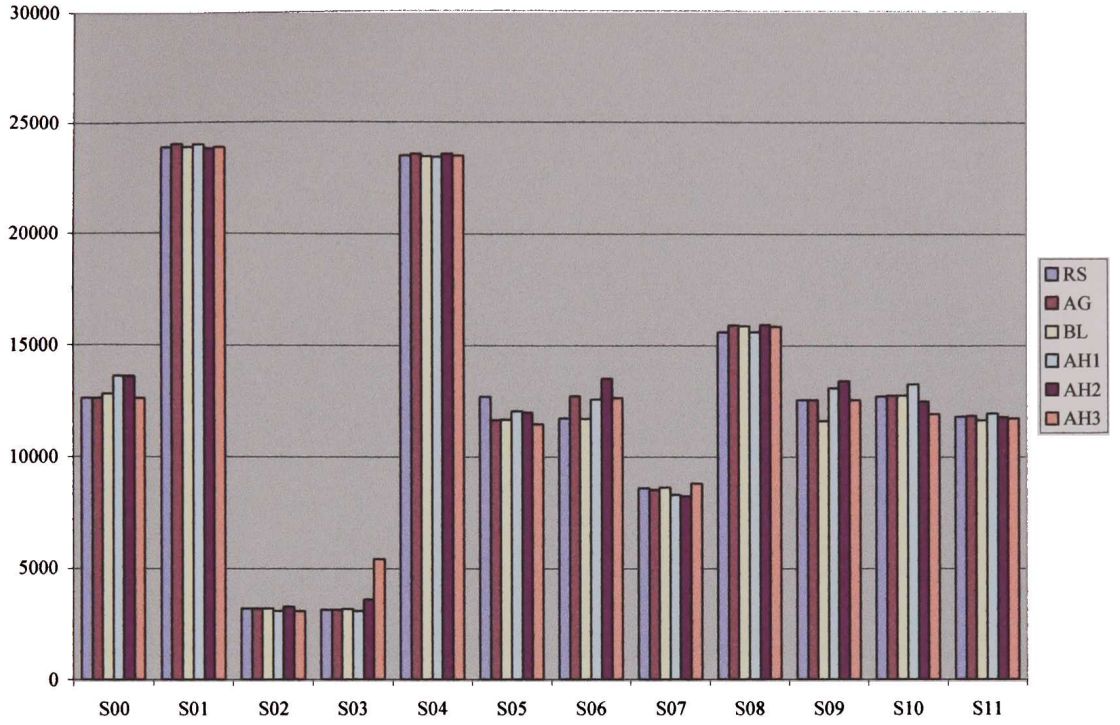

Fig. 4-21: Comparación de la función objetivo por algoritmo

En la figura 4-22 se presentan los resultados obtenidos por el algoritmo H3, tanto por el primer enfoque como por el segundo. El segundo enfoque es mucho mejor que el primero. Nuevamente el eje de las ordenadas es adimensional.

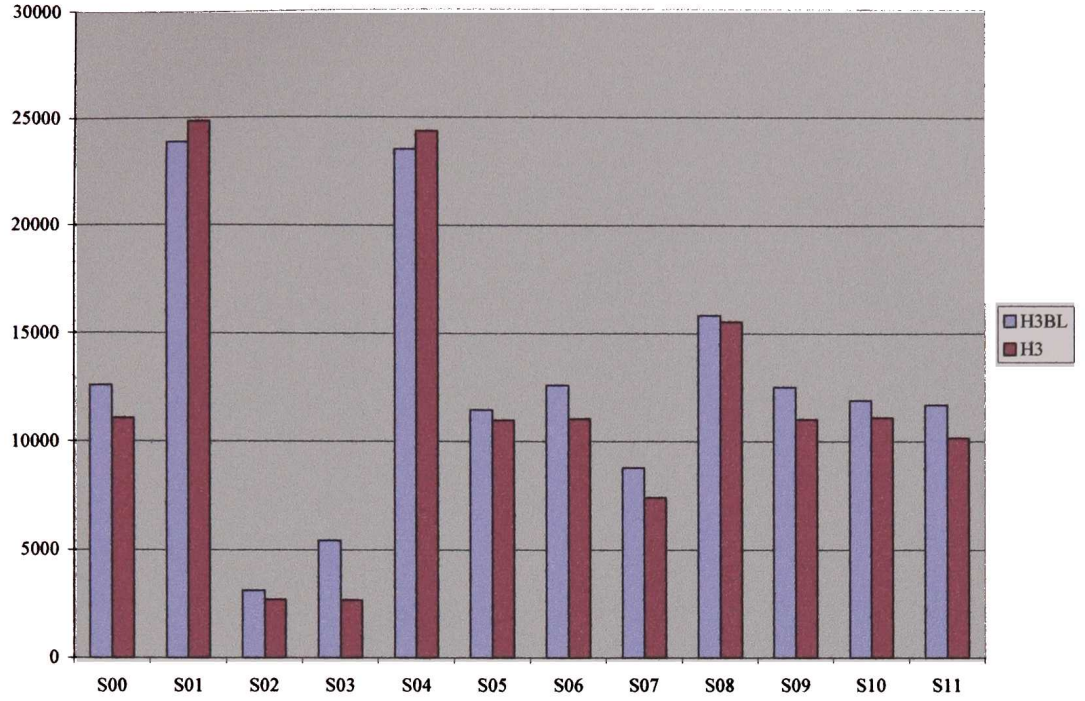

Fig. 4-22: Comparación del enfoque 1 y el enfoque 2 con H3

Aunque en la figura 4-22 se advierte que el segundo enfoque cumple mejor que las expectativas que el primero, la diferencia es mínima. Por lo que es necesario obtener la diferencia exacta entre estos enfoques, la cual se presenta en la figura 4.23. Cabe mencionar que el resultado obtenido es adimensional. Se utiliza la expresión (4.13) para llevar a cabo esta comparación.

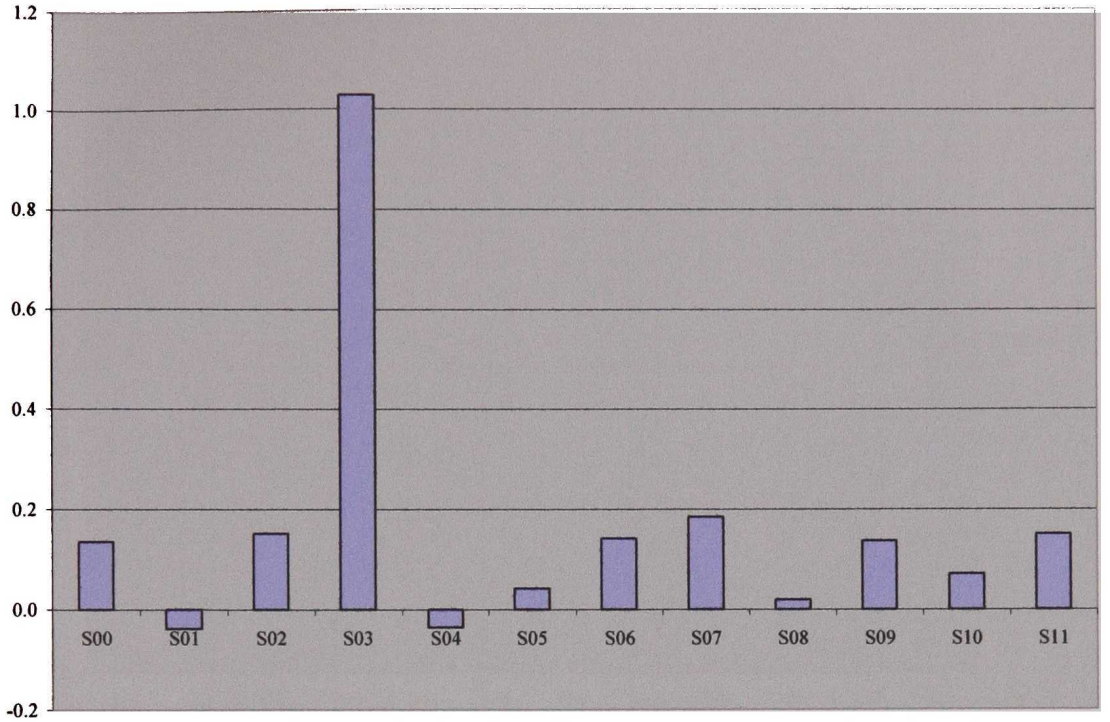

Fig. 4-23: Comparación del error relativo entre los enfoques utilizando H3

Siguiendo con las comparaciones, el tiempo utilizado por el primer enfoque sólo necesita de 1 minuto para obtener el resultado, y el segundo enfoque requiere del tiempo que consume el algoritmo utilizado, en este caso que se compara con el algoritmo H3 que también requiere de un minuto para obtener el resultado, la aceleración es 1, tal como se observa en la figura 4.24.

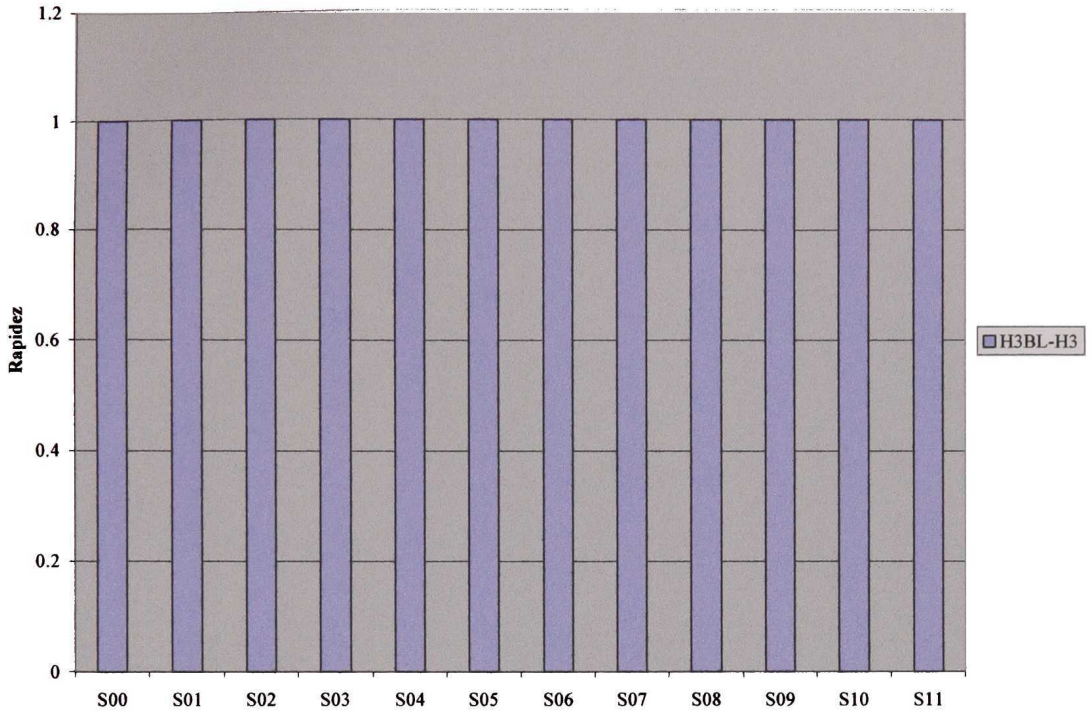

Fig. 4-24: Comparación de aceleración entre los dos enfoques

4.3 Comparación de otros planteamientos

Continuando con las comparaciones, se analizaran los resultados obtenidos con AG utilizando las diferentes funciones objetivos propuestas en el primer capítulo. Por un lado la expresión (1.9) con la cual se utilizo durante el desarrollo de este trabajo, y la expresión (1.21) que es una variante de la primera expresión. Los resultados obtenidos por AG utilizando estas funciones objetivos, expresiones (1.9) y (1.21), son comparados bajo una misma función objetivo.

En la figura 4-25 se muestra la comparación bajo la expresión (1.9), V01 es evaluando la asignación obtenida con la expresión (1.9) y V02 es la obtenida por la expresión (1.21), y ambas evaluadas el la expresión (1.9) y presentadas a continuación.

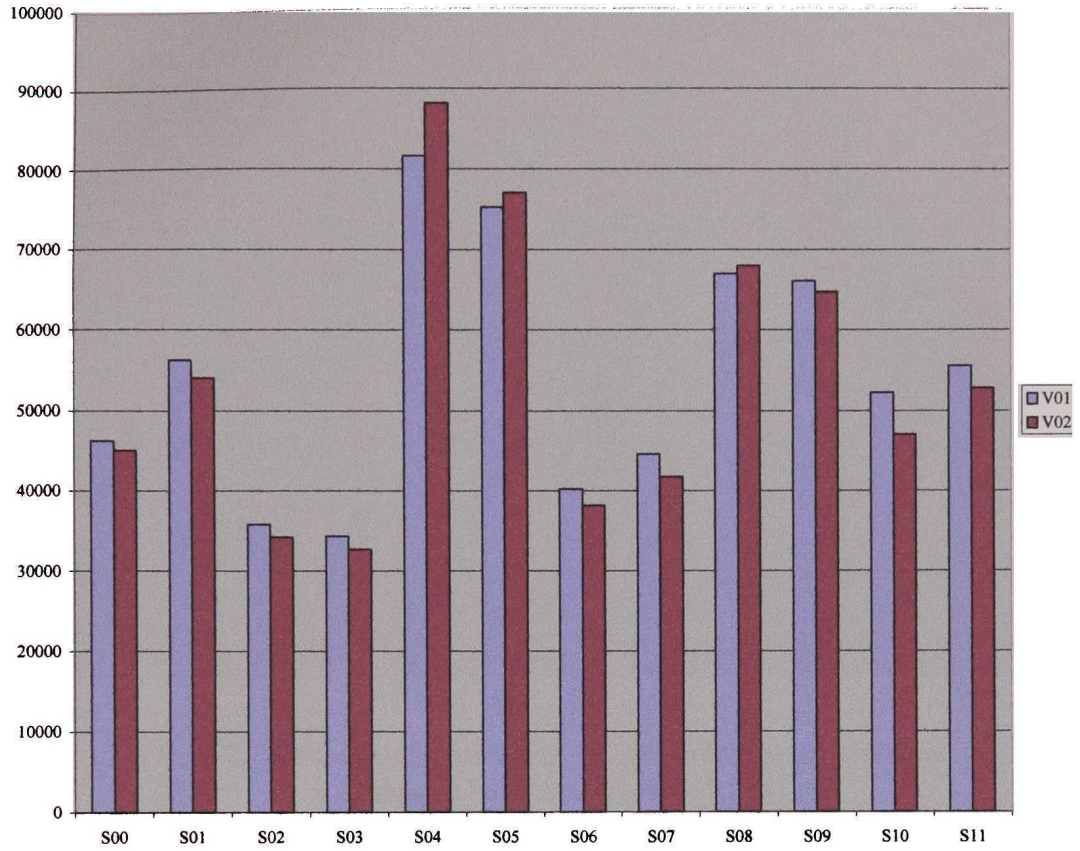

Fig. 4-25: Vectores obtenidos por diferentes funciones objetivos y evaluadas bajo la misma expresión (1.9).

Nuevamente se recuerda que el valor de las ordenadas es adimensional. A continuación se muestra la misma comparación en la figura 4-26, con la diferencia de que son evaluadas en la expresión (1.21). Se puede observar que en ambos casos la expresión (1.21) hace que AG tenga un mejor desempeño que con la expresión (1.9), pero a su vez la expresión (1.21) tiene mayor complejidad.

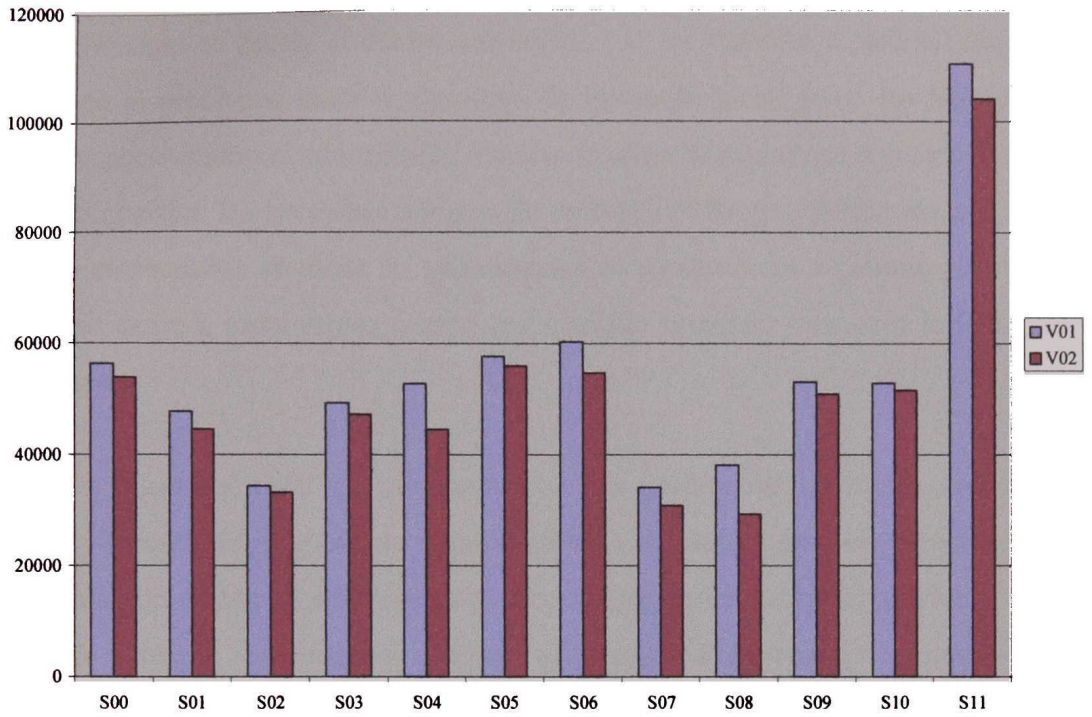

Fig. 4-26: Vectores obtenidos por diferentes funciones objetivos y evaluadas bajo la expresión (1.21).

4.4 Discusión de resultados

Desde el punto de vista de minimizar la función objetivo, los métodos de optimización se observa que el algoritmo de búsqueda local tiene mejor comportamiento y que en algunos casos algoritmo genético y en otros recocido simulado tiene el peor. En los algoritmos heurísticos, el denominado *copia-procesador* tiene un mejor desempeño, y que el algoritmo *tarea-procesador-máximo* es el caso contrario. Comparando los seis algoritmos se observar que los resultados obtenidos por los algoritmos heurísticos son satisfactorios en los ejemplos mostrados, con respecto a los métodos de optimización. Cabe mencionar que con estos resultados no se puede generalizar el hecho de que los algoritmos heurísticos sean mejores que los métodos de optimización.

Tomando en cuenta el tiempo consumido por los algoritmos para la obtención de la solución al problema, tanto el algoritmo de búsqueda local, como los heurísticos les toma muy poco tiempo el minimizar la función objetivo. Mientras que recocido simulado y algoritmo genético les toma más tiempo, sin embargo el tiempo consumido por recocido simulado es constante en todas las simulaciones, al contrario del algoritmo genético, que en algunos casos le toma menos tiempo que recocido simulado y en otros le toma mucho más tiempo.

En cuanto a nivel de programación. Los parámetros utilizados en algoritmo genético, tales como: número de padres, número de puntos para el entrecruzamiento, probabilidad de mutación, etc. dependen directamente de los datos del problema, la única forma de obtener los valores ideales de estos parámetros es a través de la experimentación. Cabe mencionar que este algoritmo presenta un buen comportamiento, sin embargo hay ejemplos donde se muestra que es el peor método de optimización , tanto en el hecho de minimizar como en el tiempo consumido, esto se debe a la dependencia que hay entre los parámetros del algoritmo y los datos del problema.

Dentro del algoritmo de recocido simulado, tanto la función de enfriamiento como la vecindad y el valor de umbral juegan un papel muy importante dentro de este método. Al igual que el algoritmo genético, la función de enfriamiento y la vecindad son parámetros cuyos valores se obtienen a través de la experimentación. Pero a diferencia del algoritmo genético, estos parámetros no dependen tan fuertemente de los datos del problema.

Los algoritmos heurísticos son los más fáciles de programar, debido a que se basan en la experiencia obtenida en forma empírica, sin embargo esto no garantiza que sean los mejores métodos para optimizar una función objetivo y que consuman el menor tiempo posible en realizar la optimización.

Dentro de las perturbaciones al sistema, se observa que el hecho de volver a

calcular la asignación de tareas (segundo enfoque) resulta mejor que partir de una asignación obtenida previamente (primer enfoque), esto se observa tanto en el hecho de minimizar a la función objetivo, como en el tiempo consumido para realizar la optimización. Esto se muestra en las cuatro formas de perturbación que se realizan en este trabajo.

En cuanto a los diferentes planteamientos del problema, existen algunas diferencias, como por ejemplo, las expresiones (1.21) y (1.22) presentan mayor complejidad que las expresiones (1.9) y (1.11), sin embargo dichas expresiones, (1.21) y (1.22), hacen que los algoritmos utilizados presenten un mejor rendimiento al momento de buscar la respuesta, pero con la desventaja de que los algoritmos consuman mayor tiempo en la solución del problema.

Conclusiones y trabajo a futuro

Como se puede observar los resultados obtenidos por los algoritmos heurísticos en este trabajo, son satisfactorios en los ejemplos mostrados, los cuales pueden ser típicos de los encontrados en las aplicaciones, con respecto a otros métodos de optimización, debido a que consumen menor tiempo de ejecución y el resultado esperado por los algoritmos es aceptable, en algunos casos mejor que los métodos de optimización.

Como es de esperarse entre los métodos de optimización, el método de *búsqueda local* tiene mejor comportamiento, debido al poco tiempo que toma para obtener el resultado. En particular en los ejemplos vistos, es el que cumple mejor con la tarea de minimizar a la función objetivo.

En el caso del algoritmo genético, se requiere que éste tenga un ajuste fino de los parámetros, tales como el número de padres, el número de cromosomas, etc. Esto se debe a que los algoritmos genéticos dependen del problema, y cada problema es diferente en esencia.

Sería mejor una combinación de los métodos propuestos, por ejemplo, el utilizar un AG para acercase al óptimo global y posteriormente una búsqueda local para llegar al óptimo local. Otro ejemplo puede ser el utilizar un AG para calcular los parámetros de operación (tales como: número de padres, puntos de entrecruzamiento, probabilidad de mutación, etc.) y posteriormente otro AG que se encargue de buscar al óptimo global del problema.

Cabe mencionar que la combinación de métodos tiene la desventaja de consumir mayor tiempo para obtener el resultado, y su ventaja es acercarse o encontrar el óptimo global, por esta ventaja vale la pena este consumo de tiempo.

Dentro de los métodos heurísticos, aunque todos ellos consumen el mismo tiempo

de ejecución para encontrar la respuesta, el método *copia-procesador* es el que tiene mejor rendimiento, esto se debe a que obtiene una minimización de mejor calidad en la función objetivo que los otros métodos.

Para la solución a problemas perturbados, donde se presentan dos enfoques, el de volver a calcular a la matriz de asignación tiene un mejor rendimiento, ya que se logra obtener una mayor minimización de la función objetivo. Además que el tiempo utilizado para solucionar la eventualidad es similar al enfoque de calcular la asignación a partir del resultado obtenido previamente por los métodos de optimización y los heurísticos.

Entre los diferentes planteamientos existentes del problema, se presentan diferentes comportamientos, es decir, algunas expresiones son menos complejas, sin embargo hacen que el algoritmo utilizado tenga menor rendimiento. Mientras que hay un planteamiento con mayor complejidad, pero con la ventaja de hacer que el algoritmo tenga un mayor rendimiento.

Como trabajo a futuro se recomienda:

Incorporar en respectivas funciones "objetivo" otros elementos que aumenten la dificultad del problema de asignación, tales como:

- Extender los métodos heurísticos a una clase mayor de problemas.
- Analizar variaciones de mínimos respecto a perturbaciones.
- Adicionar otros factores, tales como: la memoria utilizada, la prioridad en las tareas, la reducción del tráfico en la comunicación entre procesadores, etc.

Referencias

[1] Bäck, T., Hammel, U., Scwefel, H., Evolutionary Computation: Commants on the History an Current State, IEEE Transactions on Evolutionary Computation, April 1997, Vol. 1, Num. 1.

[2] Bannister, J.A. & Trivedi K.S., " Task allocation in fault-tolerant distributed systems in hard Real-Time systems" (Tutorial), J.A. Stankovic & K. Ramamritham, Eds. IEEE Computer society, 1988, pp. 256-272, 1988.

[3] Bokhari. S.H.: Dual Processor Scheduling with Dynamic Reassignment. IEEE Trans. Software Engrg. SE-5. 341-349 (1979).

[4] Chen, P & Akuka J. "Optimal design of distributed information system", IEEE Tans. Comput. Vol C-29, Dec. 1980.

[5] Chou, T.C.K., Abraham, J.A.: Load Balancing in Distributed Systems. IEEE Trans Software Engrg. SE-8, 401-412 (1982).

[6] Chu, W.W. "Introduction in the Special issue on distributed processing systems", IEEE Trans. Comp. Vol C-29. Dec. 1980.

[7] Chu, W.W., Holloway, L.J., Lam, M.-T. Efe, K.: Task Allocation in Distributed Data Processing. IEEE Comput. 13, 57-69 (1980).

[8] Eiben, A.E., Arts, E.H.L., & Van Hee K.M., Global Convergence of Genetic Algorithms: A Markov Chain Analysis in Parallel Problem Solving from Nature, Springer, 1991.

[9] Goldberg, G.E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

[10] Günter, R., Convergence Analysis of Canonical Genetic Algoriths, IEEE Transactions on Neural Networks, Jan. 1994, Vol. 5, Num. 1.

[11] H.S. Stone: "Multiprocessor scheduling with the aid of network flow algorithm", IEEE Trans. Software Eng., Vol. SE-3, pp. 85-93, Jan. 1977.

[12] Holland, J.H., Adaptation in Natural and Artificial Systems, MIT Press/Bradford Books Edition, 1992.

[13] Koza, J.R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press/Bradford Books Edition, 1994.

[14] Krikpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Science 1983, 220, 671-680.

[15] Metropolis, N.; Rosenbluth, A.; Rosenbuth, M.; Teller, A.; Teller, E.J. Chem Phys. 1953, 21, 1087-1092.

[16] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of State Calculations by Fast Computing Machines, J. of Chem. Phys., Vol. 21, No. 6, pp. 1087-1092, 1953.

[17] Protzel, Peter W. "Artificial Neural Network for Real-Time task allocation in Fault-tolerant, distributed processing system", Parallel Processing in Neural systems and computers, p.p. 307-310, 1990.

[18] Stone, H.S., Bokhari, S.H.: Control of Distributed Processes. Computer 11, 97-106 (1978).

[19] V.B. Gylys & J.A. Edwards:Optimal partitioning of workload for distributed systems", in Dig. COMPCON Fall 1976, pp. 353-357, 1976.

[20] Van Laarhoven, P.J.M.; Aarts, E.H.L. Simulated annealing: Theory and Applications; D. Reidel: Dordrecht, 1987.

[21] Vick, C.R. "A next generation of supercompoters: From mainframes to micros" presented at Euromicro 1980, London, England, Sept-1980.

[22] M. Zbigniew, Genetic Algoriths + Data Structures = Evolution Programs, Springer-Verlag, 1994.

Los abajo firmantes, integrantes de jurado para el examen de grado que sustentará el **Ing. Mario Farías Elinos,** declaramos que hemos revisado la tesis titulada:

Reasignación de tareas de un sistema multiprocesador

y consideramos que cumple con los requisitos para obtener el grado de Maestro en Ciencias, con especialidad en Ingeniería Eléctrica.

Atentamente

Dr. Guillermo B. Morales Luna

Dr. Arturo Díaz Pérez

Dr. Héctor Ruíz Barradas