PARTICIÓN DE SISTEMAS ELÉCTRICOS PARA SU RESOLUCIÓN EN UN AMBIENTE COMPUTACIONAL DISTRIBUIDO

Rodrigo Andrés Ramos Galeano

Trabajo de investigación presentado como parte de los requisitos necesarios para la obtención del título de Ingeniero Electromecánico

FACULTAD DE INGENIERÍA

UNIVERSIDAD NACIONAL DE ASUNCIÓN

San Lorenzo - Paraguay
Agosto de 1996

Este trabajo está dedicado a mis padres y hermana. Espero poder retribuirles en alguna medida todo el apoyo, la comprensión y sobre todo, el amor que he recibido de ellos desde siempre.

Agradecimientos

Al Profesor Benjamín Barán, excelente profesional y por sobre todo, excelente persona. Sin su constante apoyo y orientación este proyecto no hubiera llegado a buen puerto. Su excepcional capacidad de superación frente a los problemas ha inyectado vida y dinamismo al quehacer diario de las personas que tenemos la gran suerte de pertenecer a su grupo de investigación.

A la Lic. Diana Benítez, gran compañera y mejor amiga, quien supo compartir a lo largo de este tiempo todos los sinsabores y todos los triunfos con la mejor predisposición de ánimo y un gran profesionalismo. Sin su apoyo y camaradería este trabajo no habría podido ser realizado.

Quisiera nombrar de una manera especial a la Lic. Blanca Troche de Trevisan, Directora del Centro Nacional de Computación, por ser la persona que posibilitó que en dicha institución puedan ser llevados a cabo trabajos de investigación de esta naturaleza. Con personas como ella estoy seguro que nuestro querido país tendrá un futuro luminoso, de pleno desarrollo y madurez.

Quiero agradecer también a los queridos compañeros del Centro Nacional de Computación, en la persona del Ing. Freddy Cardozo, por el apoyo que me han brindado, inclusive muchas veces sacrificando sus propias prioridades.

Hay personas que dejan en nuestra vida un recuerdo indeleble de cariño y amistad. En este contexto quiero especialmente agradecer a la familia Benítez Cáceres por abrirme las puertas de su hogar como a un miembro más durante el último año y medio de trabajo. Este trabajo es tan suyo como mío.

A los excelentes jóvenes de los Centros Universitarios Ybaté, Puntarrieles y especialmente a los de la Residencia Universitaria Ycuá, en la persona del Lic. Rafael Medina, por haberme recibido entre ellos durante los primeros años de la carrera y por continuar ofreciéndome su amistad y consejo. Su constante aliento y ejemplo me acompañará por el resto de mi vida, como señal clara de que la juventud paraguaya se encuentra entre las mejores del mundo.

A todos mis profesores, compañeros y amigos de la Facultad de Ingeniería, con quienes he compartido estos años de formación. Mi deseo es que las herramientas que hemos adquirido a través de nuestro paso esta querida facultad nos sirvan para forjarnos una vida digna, honesta y plena, al servicio de nuestras familias y nuestro país.

Finalmente, quiero agradecer al Padre Creador por todas las gracias con las que me ha colmado a lo largo de mi camino. Siempre es pequeño el reconocimiento comparado con el bien recibido. Gracias Señor!.

TABLA DE CONTENIDO

Lista de Figuras

Lista de Tablas

CAPITULO 1: INTRODUCCION

1.1 Consideraciones iniciales

Para que un sistema de energía eléctrica cumpla su finalidad de atender a la demanda siguiendo criterios de calidad, seguridad y economía, un amplio espectro de actividades debe ser considerado, envolviendo etapas de estudios de planeamiento, operación (incluyendo supervisión y control en tiempo real) y análisis de post-operación. Gran parte de las tareas asociadas a estas actividades está relacionada a simulaciones computacionales del comportamiento estático y dinámico del sistema en cuestión. Tales simulaciones exigen cada vez más un mayor esfuerzo computacional, debido a diversas razones, entre ellas, las elevadas dimensiones de los sistemas, la exigencia de utilización de modelos más detallados para los componentes de la red eléctrica y la necesidad de simulaciones en tiempo real. Como campo de aplicación de dichos estudios podemos citar, a nivel local, la futura interconexión de los sistemas eléctricos de los países miembros del Mercosur, la cual generará un sistema global eléctrico de grandes dimensiones con características muy peculiares debido a las particularidades de los sistemas a ser interconectados. A medida que aumentan las exigencias a que el sistema eléctrico se ve sometido, diversas situaciones pueden presentarse, entre ellas, por ejemplo, el Colapso de Tensión. En estas circunstancias sería necesario un estudio más detallado del comportamiento de sistema, lo que conllevaría a un uso cada vez más exigente de las herramientas computacionales.

Es este sentido, es ya reconocido el increíble nivel alcanzado en lo que se refiere a la velocidad de trabajo de los procesadores. Es así que hoy en día se está llegando al límite teórico de velocidad de los procesadores construidos en base a la tecnología de los semiconductores, pudiéndose concluir de esto que el aumento en la velocidad de procesamiento de estos procesadores no podrá continuar

sostenidamente en los próximos años, como muestra la figura 1.1, donde se puede observar el avance histórico de la velocidad de procesamiento en lo que concierne a los microprocesadores fabricados por INTEL [14].

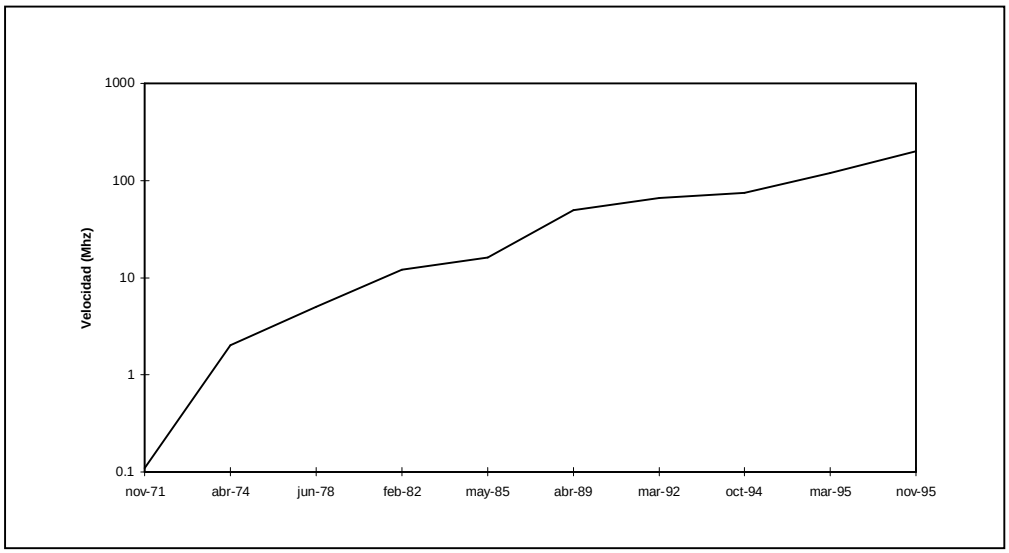

Figura 1.1 : Evolución de la velocidad de procesamiento

Por otro lado, la velocidad de comunicación con que operan las modernas redes de computadoras está aumentando a un ritmo sostenido, como puede apreciarse en la figura 1.2, donde se muestra la evolución de la misma a través de los últimos 18 años [10]. De acuerdo a las tendencias apreciadas en estas 2 figuras y para un mismo problema, la relación tiempo de procesamiento/tiempo de comunicación será cada vez menor, disminuyendo el costo computacional del envío de datos a través de una red de comunicaciones, surgiendo así la computación paralela/distribuida como la alternativa más efectiva para la resolución de problemas de dimensiones y complejidad creciente.

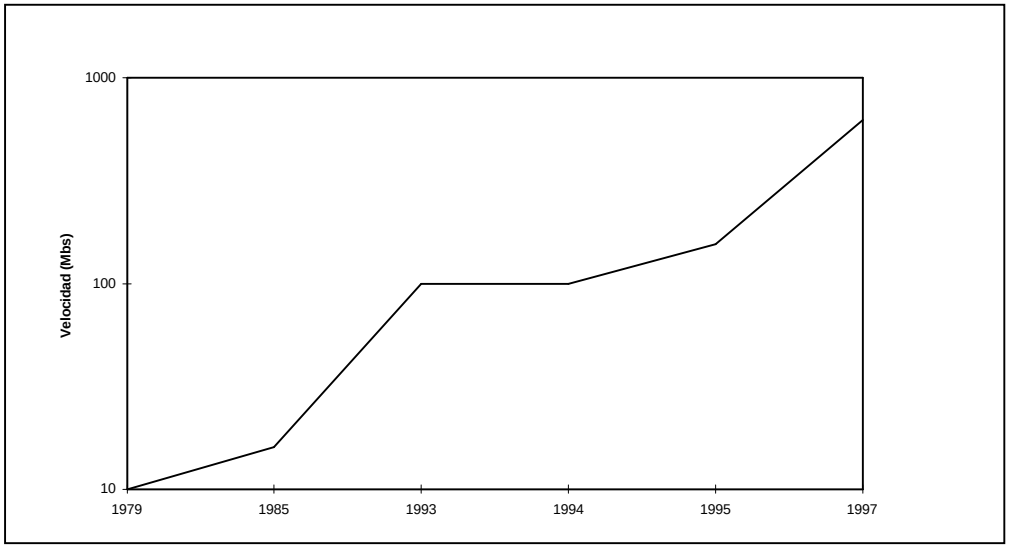

Figura 1.2 : Evolución de la velocidad de comunicación

1.2.- Procesamiento paralelo.

El procesamiento paralelo consiste en resolver *simultáneamente* varias tareas de un problema de gran dimensión o complejidad. Esta tecnología ha emergido como una importante opción en la computación moderna de alta *performance*, siendo los últimos años testigos de una aceptación y adopción creciente del procesamiento paralelo, tanto para computación científica de alto desempeño como para propósitos más generales como la administración y consulta de grandes bases de datos. En consecuencia a estas proyecciones y tendencias, estas plataformas de trabajo están captando cada vez más el interés del mercado y la atención de los investigadores como una interesante opción que permite el procesamiento de grandes volúmenes de información y la resolución de problemas computacionales de gran complejidad, que de ser resueltos en un solo procesador exigirían mucho más tiempo.

La plataforma computacional necesaria para la aplicación del procesamiento paralelo consiste en un sistema computacional distribuido/paralelo, es decir, un

conjunto de procesadores interconectados entre sí por una red de comunicación. Pudiendo utilizarse para este fin inclusive microprocesadores de muy bajo costo, es evidente las ventajas económicas que acarrea esta implementación, teniendo en cuenta el gran parque de computadoras personales (PC) disponibles en la actualidad.

Las técnicas de procesamiento paralelo han sido aplicadas con éxito en diversas áreas de la ciencia y la ingeniería, como [21]:

- Hidrodinámica computacional
- Programas de Sismografía
- Óptica Física
- Ingeniería Genética
- Medicina: Tomografía Cerebral Computada
- Diseño de motores
- Mecánica cuántica

La mayoría de estas aplicaciones, así como también las aplicaciones relacionadas con la Ingeniería Eléctrica, utilizan modelos matemáticos que implican el procesamiento de gran cantidad de datos. En este contexto, la aplicación del procesamiento paralelo a la resolución de problemas tales como el Flujo de Potencia y el Punto de Colapso presenta excelentes perspectivas, como puede apreciarse en trabajos recientes de investigación realizados sobre el tema [4,6,7,25].

1.3.- Partición de problemas

Para que un problema particular sea resuelto utilizando procesamiento paralelo, es necesario que dicho problema sea “paralelizable”, es decir, que las características de los métodos implementados en la resolución del mismo permitan la distribución de diversas tareas a los procesadores componentes del sistema distribuido, de manera tal a resolver el problema en conjunto.

De los numerosos problemas paralelizables que se conocen, nos ocuparemos preferentemente de los problemas de ingeniería que se plantean como un sistema (no necesariamente lineal) de ecuaciones, que por lo general son resueltos utilizando métodos iterativos. Para paralelizar estos métodos, nacen los métodos bloque-iterativos, que consisten en asignar a los procesadores del sistema distribuido distintos grupos de ecuaciones a ser resueltos localmente. Los resultados obtenidos localmente por cada uno de ellos son transmitidos a los demás a través de la red de comunicación, avanzando el conjunto hacia la solución global del sistema de una forma típicamente iterativa.

La utilización de los métodos de solución bloque-iterativos aplicados a los problemas eléctricos más arriba citados en un ambiente computacional paralelo merece especial atención, ya que ha presentado resultados promisorios. Para que la implementación de estos métodos sea computacionalmente eficiente, es importante descomponer la red eléctrica de forma a promover el adecuado “mapeamiento” del problema para la arquitectura paralela; esto es, debe descomponerse la red eléctrica en subredes de forma tal que la implementación del algoritmo de resolución sea computacionalmente eficiente.

La literatura presenta diversos trabajos sobre métodos de partición que descomponen la red eléctrica utilizando diversos criterios. Desafortunadamente, la mayoría de estos métodos no poseen la eficiencia requerida (mismo aquellos exclusivamente aplicados a los sistemas eléctricos) con relación al desempeño de los métodos de solución que resuelven la red descompuesta. Surge así la motivación de

este trabajo, que consiste en desarrollar un método de descomposición de redes eléctricas, que permita su resolución eficiente utilizando procesamiento paralelo en un sistema distribuido heterogéneo. Una resolución eficiente es aquella que utiliza un menor tiempo computacional en llegar a la solución, buscando balancear de manera adecuada la carga computacional a ser repartida a los diversos procesadores y minimizar la cantidad de envíos de mensajes entre procesos paralelos.

De esto último se puede concluir que, considerando la posibilidad de operar en un ambiente computacional compuesto por procesadores de diversas *performances*, interconectados por un sistema de comunicación, la descomposición de la red estará condicionada por dos factores:

a) El *balanceamiento de carga computacional*: Este factor determina las dimensiones de las subredes asignadas a cada procesador, que deberán estar en proporción con la capacidad relativa de procesamiento de cada uno de los procesadores;
b) El *grado de acoplamiento* que puedan tener las subredes entre sí, lo que determina la dependencia entre las variables de las barras y por consiguiente, influye en la convergencia del algoritmo [11].

En conclusión, el problema que se plantea, dado un sistema distribuido con cierto número de procesadores, consiste en descomponer una red eléctrica en varias subredes menores, de forma tal que a cada procesador se le asigne una subred de dimensión proporcional a su performance, y que la dependencia entre las variables actualizadas por cada uno de los procesadores facilite la convergencia del algoritmo a ser utilizado.

1.4.- Revisión bibliográfica

La partición de redes eléctricas no es un tema de introducción reciente, encontrándose los primeros trabajos al respecto a finales de la década del 60, donde Carré [11] ya afirmó que el acoplamiento entre las distintas subredes influye en la convergencia de los algoritmos iterativos. Desde entonces se sucedieron diversas publicaciones , las cuales presentan características variadas de acuerdo a las diferentes aplicaciones a las que fueron destinadas. De estos trabajos, varios buscan la eficiencia de técnicas de resolución bien específicas como, por ejemplo, la técnica diacóptica [29,16], eliminación de Gauss [20] o la programación no lineal [24] en la resolución del flujo de potencia; otros buscan la identificación de “clusters” [23] o la descomposición de sistemas eléctricos para posibilitar la solución de redes de gran porte [1], sin tener en cuenta la existencia de una red heterogénea. Puede verse así que los métodos precedentes no están directamente relacionados con la aplicación del procesamiento paralelo, aún cuando utilicen redes eléctricas descompuestas.

Aun así, se dispone en la actualidad de algunos métodos volcados a la aplicación de la computación paralela, presentando algunos de ellos características muy interesantes [18, 15, 22, 33]. Pero, a pesar de todo, no se encuentra en la mayoría de ellos un análisis de los aspectos básicos que favorecen el desempeño de los métodos de solución que irán a utilizar la red descompuesta, principalmente en lo que se refiere a la resolución de redes eléctricas. Entre los trabajos que incluyen dicho enfoque resaltan de manera especial dos: La *Descomposición* ε [26,27,28] y el *método de la semilla* [31,32]. El primero de ellos, aún cuando es el más implementado en la actualidad, posee la limitación de no poder ejercer un control efectivo sobre el número de subredes en el cual la red es descompuesta ni sobre la dimensión de las mismas. El segundo método, el *Método de la Semilla*, fue introducido en 1992 por Vale et al. [31]. Este método se basa en la identificación de los centros de aglutinamiento de barras, buscando separar la red en los lugares donde el acoplamiento entre barras es menor. Pero nuevamente se presenta la imposibilidad de ejercer dominio sobre las dimensiones de las subredes, al suponerse en dicho

trabajo que el sistema sería resuelto utilizando un sistema distribuido homogéneo, es decir, las subredes resultan siempre de igual dimensión. Inspirado en este último trabajo, surge la presente propuesta que propone una nueva metodología de partición que permite generar subredes con las dimensiones requeridas por el usuario.

1.5.- Objetivos, Metodología y Organización del presente trabajo

Teniendo en cuenta la finalidad principal del presente trabajo, los objetivos generales del mismo se establecieron como sigue:

- Desarrollar un método computacional automático, con la posibilidad de interacción con el usuario, para descomponer un sistema eléctrico de forma tal que su resolución utilizando procesamiento paralelo en un sistema distribuido heterogéneo sea eficiente.
- Demostrar experimentalmente las ventajas del método propuesto con respecto a los métodos de partición disponibles en la actualidad.

En la prosecución de los objetivos citados, la metodología de trabajo se basó en los siguientes puntos:

- Análisis de los métodos de partición disponibles.
- Evaluación experimental de dichos métodos.
- Proposición de un método alternativo.
- Evaluación experimental del método propuesto, utilizando los sistemas eléctricos paradigmas de la IEEE.

El trabajo fue dividido en ocho capítulos. La formulación matemática del problema de la partición de sistemas de ecuaciones se encuentra en el Capítulo 2,

con un apartado dedicado al repaso de las características de los métodos bloque-iterativos.

El Capítulo 3 contiene un breve estudio sobre el problema del Flujo de Potencia Eléctrica, la formulación matemática del mismo y su resolución utilizando una variante del método de Newton-Raphson.

En el Capítulo 4 se exponen los principales métodos de partición de sistemas eléctricos, disponibles en la actualidad: La Descomposición ε y el método de la semilla.

La exposición pormenorizada del método de partición propuesto se encuentra en el Capítulo 5. Se detalla cada una de las etapas de que consta el método propuesto y los parámetros que influyen en cada etapa.

A manera de ejemplo ilustrativo, el Capítulo 6 procede a la aplicación del método propuesto a un problema ejemplo diseñado para explicar el método propuesto. Las distintas etapas del método, así también como las distintas peculiaridades que pueden presentarse en su aplicación son analizadas en forma detallada.

Los estudios experimentales realizados con el fin de verificar la calidad del método propuesto se presentan en el Capítulo 7. Se utilizaron 2 sistemas eléctricos tipos proveídos por le IEEE, particionándolos y resolviendo en forma distribuida el problema del Flujo de Potencia para la red descompuesta. Se describe además en forma detallada la plataforma computacional distribuida utilizada para realizar los estudios.

Finalmente, las conclusiones que han podido ser obtenidas del presente trabajo se presentan en el Capítulo 8.

Los códigos de los programas computacionales utilizados tanto en la aplicación del método como en la obtención de los resultados experimentales se presentan en los apéndices 1 y 2. El apéndice 3 contiene el artículo publicado en el marco de la XXII Conferencia Latinoamericana de Informática. En dicho artículo, se demuestra la eficacia del método en la partición de cualquier sistema lineal o no lineal de ecuaciones para su resolución en un ambiente distribuido.

CAPITULO 2: FORMULACION MATEMATICA DEL PROBLEMA DE LA PARTICION

En este capítulo se presenta el problema de la partición de un sistema eléctrico desde el punto de vista matemático, siendo esta partición un caso particular de la partición de un sistema no lineal de ecuaciones. Se analizará primero el caso general, concentrándonos luego en el caso más específico del Flujo de Potencia Eléctrica. Una breve recapitulación sobre los métodos de resolución bloque iterativos se presenta previamente, con el fin de formalizar el contexto matemático del capítulo.

2.1.- Características de los métodos bloque iterativos

Los métodos bloque iterativos resuelven de forma iterativa los diversos subproblemas de un dado problema global. Para ilustrar mejor este concepto se muestra como ejemplo al método bloque-Jacobi para la resolución de sistemas de ecuaciones lineales.

Sea un sistema lineal y no singular de ecuaciones:

$$\mathbf{Ax} = \mathbf{b}, \qquad \mathbf{A} \in \Re^{n \times n}, \qquad \mathbf{x},\ \mathbf{b} \in \Re^{n} \tag{2.1}$$

El sistema puede ser descompuesto como :

$$\begin{bmatrix} \mathbf{A}_{11} \cdots \mathbf{A}_{1p} \\ \vdots \ \ddots \ \vdots \\ \mathbf{A}_{p1} \cdots \mathbf{A}_{pp} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_p \end{bmatrix}, \mathbf{A}_{ij} \in \Re^{n_i \times n_j}, \qquad i, j \in \{1, \ldots, p\} \tag{2.2}$$

donde:

$$\mathbf{x}_i, \mathbf{b}_i \in \Re^{n_i} \quad ; \quad n = \sum_{i=1}^{p} n_i$$

Desdoblando la matriz **A** según $\mathbf{A} = \mathbf{S} - \mathbf{T}$, de forma tal que **S** sea bloque diagonal e invertible por bloques, se obtiene

$$\mathbf{Sx} - \mathbf{Tx} = \mathbf{b} \tag{2.3}$$

$$\mathbf{x} = \mathbf{S}^{-1}(\mathbf{b} + \mathbf{Tx}) \tag{2.4}$$

$$\mathbf{x}(k+1) = \mathbf{S}^{-1}(\mathbf{b} + \mathbf{Tx}(k)) = \begin{bmatrix} \mathbf{S}_{11} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{S}_{pp} \end{bmatrix}^{-1} \left\{ \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_p \end{bmatrix} + \begin{bmatrix} \mathbf{T}_{11} & \cdots & \mathbf{T}_{1p} \\ \vdots & \ddots & \vdots \\ \mathbf{T}_{p1} & \cdots & \mathbf{T}_{pp} \end{bmatrix} \mathbf{x}(k) \right\} \tag{2.5}$$

lo que puede resolverse haciendo que cada procesador *i* actualice $\mathbf{x}_i$ conforme a:

$$\mathbf{x}_i(k+1) = \mathbf{S}_{ii}^{-1} \left(\sum_{j=1}^{p} \mathbf{T}_{ij}\, \mathbf{x}_j(k) + \mathbf{b}_i \right), \quad \forall i \in \{1, \ldots, p\} \tag{2.6}$$

De esta forma, el problema (2.1) puede ser resuelto con *p* procesadores de forma tal que en cada iteración, cada procesador *i* solo actualice su incógnita local x_i conforme a (2.6), comunicando el valor actualizado a los demás procesadores del sistema distribuido.

El objetivo del presente trabajo es encontrar la partición de la matriz **A** en las correspondientes submatrices $\mathbf{A}_{ij}$ de forma a asignar a cada procesador un subproblema proporcional a su capacidad de procesamiento (performance relativa), además de buscar obtener la mejor convergencia posible, es decir, que el método de resolución a ser utilizado emplee un número reducido de iteraciones hasta llegar a la solución global.

2.2.- Sincronismo y Asincronismo

En una resolución bloque-iterativa, cada procesador actualiza su incógnita local $\mathbf{x}_i$ utilizando los resultados obtenidos en la iteración anterior por todos los procesadores del sistema, que les fueron comunicados a través de la red de comunicación. Este proceso continúa hasta que todo el conjunto llegue a la solución global del problema.

La comunicación de los resultados parciales puede ser realizada de forma síncrona o asíncrona:

Comunicación síncrona: ocurre cuando cada procesador espera hasta disponer de todos los resultados parciales en la iteración anterior calculados por los demás procesadores del sistema antes de calcular su siguiente iteración. Al tiempo transcurrido entre la terminación de su iteración y el inicio de la siguiente iteración se denomina “tiempo de sincronización”.

Comunicación asíncrona: en este caso, cada procesador utiliza los resultados parciales más recientes que ha recibido de los demás procesadores al momento de iniciar una iteración. La comunicación asíncrona contempla la posibilidad de que un procesador más rápido que otros trabaje a su propia velocidad, sin depender de las velocidades relativas de los demás procesadores. Esto conlleva a que se requieran diferentes números de iteraciones por procesador, dependiendo de la capacidad del mismo y de la dimensión de su problema local.

Si bien las comunicación asíncrona “ahorra tiempo”, se resalta la dificultad de lograr un control estricto de las actividades realizadas por cada uno de los procesadores en el dicho contexto, debido a la complejidad del proceso de comunicación implicado. Por esto, es más difícil obtener la convergencia de los

métodos de resolución bloque-iterativos en la resolución asíncrona que en la síncrona, aunque una vez obtenida, el asincronismo alcanza la solución en menos tiempo [5].

2.3.- Un problema general

Considerando un sistema de *n* ecuaciones algebraicas con *n* incógnitas dado por:

$$\underset{\sim}{\Phi}(\mathbf{x}) = \mathbf{0}, \qquad \mathbf{x} \in D \subset \Re^n, \qquad \underset{\sim}{\Phi}: D \to \Re^n \tag{2.7}$$

definido en un dominio $D \subset \Re^n$; se desea resolver (2.7) utilizando un sistema distribuido con *p* procesadores, donde $n \geq p$. Para esto, se debe partir el problema de manera tal que cada procesador "resuelva" solamente una parte (o subproblema) del sistema completo, comunicando los resultados parciales a los otros procesadores para finalmente resolver en conjunto el problema global. Se introducen a continuación la notación y algunas definiciones a ser utilizadas.

Sea la Descomposición Cartesiana de $\Re^n$ dada por:

$$\Re^n = \Re^{n_1} \times \cdots \times \Re^{n_p}, \qquad n_1 + \cdots + n_p = n \tag{2.8}$$

con $D \subset \Re^n$ un dominio tal que

$$D = D_1 \times \cdots \times D_p, \qquad D_i \subset \Re^{n_i}, \qquad \forall i \in \{1, \ldots, p\} \tag{2.9}$$

El vector incógnita **x** puede ser particionado conforme a:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_p \end{bmatrix}, \qquad \mathbf{x}_i \in D_i, \qquad \forall\, i \in \{1, \ldots, p\} \tag{2.10}$$

Análogamente, $\Phi(\mathbf{x})$ puede descomponerse como:

$$\Phi(\mathbf{x}) = \begin{bmatrix} \Phi_1(\mathbf{x}) \\ \vdots \\ \Phi_p(\mathbf{x}) \end{bmatrix}, \qquad \Phi_i : D \to \Re^{n_i} \tag{2.11}$$

Consecuentemente, la ecuación a ser resuelta en cada procesador *i* (problema local), estará dada por:

$$\Phi_i(\mathbf{x}) = \mathbf{0}, \qquad \forall i \in \{1,\ldots,p\} \tag{2.12}$$

NOTA 1: *La partición adecuada de la función* $\Phi(\mathbf{x})$ *en p funciones* $\Phi_i(\mathbf{x})$ *es el objetivo central del presente trabajo.*

Para resolver el problema será utilizado un algoritmo iterativo de la forma

$$\mathbf{x}^{k+1} = \mathbf{G}(\mathbf{x}^k) \tag{2.13}$$

donde *k* representa la iteración. Por lo tanto, en forma análoga a (2.10) y (2.11):

$$\mathbf{G}(\mathbf{x}) = \begin{bmatrix} \mathbf{G}_1(\mathbf{x}) \\ \vdots \\ \mathbf{G}_p(\mathbf{x}) \end{bmatrix}, \qquad \mathbf{G}_i(\mathbf{x}) : D \to D_i\ , \qquad \forall i \in \{1,\ldots,p\} \tag{2.14}$$

La implementación síncrona del algoritmo (2.13) para el procesador *i* será entonces:

$$\mathbf{x}_i^{k+1} = \mathbf{G}_i(\mathbf{x}^k) \tag{2.15}$$

Utilizando la notación dada en [5] podemos escribir:

$$\mathbf{x}^{k+1} = \mathbf{H}(\mathbf{x}^k, k)\,\mathbf{x}^k \tag{2.16}$$

donde la *función iteración* $\mathbf{H}(\mathbf{x}^k, k)$ puede ser no lineal y variante en el tiempo.

A continuación, se harán las siguientes hipótesis:

Hipótesis 1 (Unicidad de la solución)

Dado el sistema de ecuaciones (2.7) que puede ser reescrito como (2.11) , definido en un dominio cerrado $D \subset \mathbb{R}^n$ *que satisface (2.9), existe un operador* **G** *de la forma (2.13), tal que* $\mathbf{G}(D) \subset D$ *y adicionalmente D contiene un y solamente un punto fijo*

$$\mathbf{x}^* = \begin{bmatrix} \mathbf{x}_1^* \\ \vdots \\ \mathbf{x}_p^* \end{bmatrix}, \qquad \mathbf{x}_i^* \in D_i , \qquad \forall\, i \in \{1,\ldots,p\} \tag{2.17}$$

del operador **G**. *El punto fijo* $\mathbf{x}^*$ *es a su vez solución de (2.7).*

Consecuentemente podemos escribir que:

$$\mathbf{x}_i^* = \mathbf{G}_i(\mathbf{x}^*) , \qquad \forall\, i \in \{1,\ldots,p\} \tag{2.18}$$

y

$$\mathbf{F}_i(\mathbf{x}^*) = \mathbf{0} , \qquad \forall\, i \in \{1,\ldots,p\} \tag{2.19}$$

Considerando un contexto asíncrono en el cual los procesadores trabajan sin barreras de sincronización, transmitiendo los valores calculados y recibiendo los valores que le van llegando, se denota como $\mathbf{x}_j(d_j^i(k))$ al valor de $\mathbf{x}_j$ enviado desde el procesador *j*, y disponible en el procesador *i* al iniciar su iteración local *k*.

Hipótesis 2 (Asincronismo parcial)

$$\exists\, d \in \mathbb{N} , \qquad \forall\, k \in \mathbb{N} , \qquad \forall\, i \in \{1,\ldots,p\} , \tag{2.20}$$

tal que, $d_j^i(k) \in \{k, k-1, \ldots, k-d\}$, *donde* $(k-d)$ *es el atraso máximo, determinado por la medida de asincronismo d.*

Denotamos como $\mathbf{x}^i(k)$ al vector **x** disponible en el procesador *i* en el momento de la iteración *k*, esto es:

$$\mathbf{x}^i(k) = \begin{bmatrix} \mathbf{x}_1(d_1^i(k)) \\ \vdots \\ \mathbf{x}_p(d_p^i(k)) \end{bmatrix}, \qquad \forall\, i \in \{1, \ldots, p\} \tag{2.21}$$

Usando esta notación podemos escribir el algoritmo asíncrono basado en (2.15) en la forma:

$$\mathbf{x}_i(k+1) = \mathbf{G}_i(\mathbf{x}^i(k)), \qquad \forall i \in \{1, \ldots, p\} \tag{2.22}$$

Esto es, cada procesador *i* actualiza $\mathbf{x}_i$ utilizando el valor disponible más actualizado de **x** al momento de iniciar la siguiente iteración.

Hipótesis 3 (Bloque-Lipschitz)

Cada operador $\mathbf{G}_i(\mathbf{x})$ *de la ecuación (2.14) es Bloque-Lipschitz continuo en D, esto es:*

$$\| \mathbf{G}_i(\mathbf{x}) - \mathbf{G}_i(\mathbf{y}) \| \leq \sum_{j=1}^{p} l_{ij} \| \mathbf{x}_j - \mathbf{y}_j \|, \qquad \forall \mathbf{x}, \mathbf{y} \in D \tag{2.23}$$

Con el objetivo de derivar una condición suficiente de convergencia para el algoritmo asíncrono (2.15), en [6] se define el vector error $\mathbf{e} = \left[\mathbf{e}_1^T, \ldots, \mathbf{e}_p^T\right]^T \in \Re^n$, $\mathbf{e}_i \in \Re^{n_i}$ como:

$$\forall i, j \in \{1,..,p\}, \qquad \begin{cases} \mathbf{e}_i(k+1) = \mathbf{x}_i(k+1) - \mathbf{x}_i^* \\ \mathbf{e}_j(d_j^i(k)) = \mathbf{x}_j(d_j^i(k)) - \mathbf{x}_j^* \end{cases} \tag{2.24}$$

y el vector error reducido $\mathbf{z} = \left[\mathbf{z}_i,...,\mathbf{z}_p\right]^T \in \Re^p$, $\mathbf{z}_i \in \Re$ como:

$$\forall i, j \in \{1,..,p\}, \qquad \begin{cases} \mathbf{z}_i(k+1) = \| \mathbf{e}_i(k+1) \| \\ \mathbf{z}_j(d_j^i(k)) = \| \mathbf{e}_j(d_j^i(k)) \| \end{cases} \tag{2.25}$$

Substrayendo (2.18) de (2.22) obtenemos $\forall i, j \in \{1,..,p\}$:

$$\mathbf{e}_i(k+1) = \mathbf{x}_i(k+1) - \mathbf{x}_i^* = \mathbf{G}_i(\mathbf{x}^i(k)) - \mathbf{G}_i(\mathbf{x}^*) \tag{2.26}$$

Tomando normas y usando la hipótesis 3:

$$\| \mathbf{e}_i(k+1) \| = \| \mathbf{G}_i(\mathbf{x}^i(k)) - \mathbf{G}_i(\mathbf{x}^*) \| \le \sum_{j=1}^{p} l_{ij} \| \mathbf{x}_j(d_j^i(k)) - \mathbf{x}_j^* \| \tag{2.27}$$

así:

$$\| \mathbf{e}_i(k+1) \| \le \sum_{j=1}^{p} l_{ij} \| \mathbf{e}_j(d_j^i(k)) \|, \qquad \forall i \in \{1,..., p\} \tag{2.28}$$

que según (2.25) puede ser escrito como:

$$\mathbf{z}_i(k+1) \le \sum_{j=1}^{p} l_{ij}\, \mathbf{z}_j(d_j^i(k)), \qquad \forall i \in \{1,... p\} \tag{2.29}$$

Para demostrar la convergencia del algoritmo asíncrono (2.22), es suficiente demostrar que el vector error reducido **z** que satisface la desigualdad (2.29) tiende a cero cuando $k \to \infty$. Para esto, usaremos el siguiente lema dado en [5]:

Lema 1 (Lema asíncrono de comparación)

Bajo la hipótesis 1 y 2, dada una matriz no negativa $\mathbf{H} = (h_{ij}) \geq 0$ *y un vector variante en el tiempo y no negativo* $\mathbf{z} \geq 0$*, satisfaciendo la inecuación*

$$\mathbf{z}_i(k+1) \leq \sum_{j=1}^{p} h_{ij}\, \mathbf{z}_j(d_j^i(k)), \qquad \forall i \in \{1,\dots p\} \tag{2.30}$$

entonces, $\rho(\mathbf{H}) < 1$ *es una condición suficiente para que* $\mathbf{z}(k)$ *tienda exponencialmente a cero cuando* $k \to \infty$*, esto es*

si $\rho(\mathbf{H}) < 1$ entonces $\lim_{k\to\infty} \mathbf{z}(k) = \mathbf{0}$

donde, $\rho(\mathbf{H})$ *denota el radio espectral de la matriz* $\mathbf{H} = \{h_{ij}\} \in \Re^{p\times p}$.

El resultado inmediato de la aplicación del Lema 1 en la ecuación (2.29), tiene como consecuencia el teorema [6] que establece una condición suficiente de convergencia para el algoritmo asíncrono dado por (2.22).

Teorema 1 (Condición suficiente de convergencia)

Bajo las hipótesis 1 a 3, el algoritmo asíncrono representado por

$$\mathbf{x}_i(k+1) = \mathbf{G}_i(\mathbf{x}^i(k)), \qquad \forall i \in \{1,\dots,p\}$$

converge a un único punto fijo $\mathbf{x}^*$ *en D si*

$$\rho(\mathbf{H}) < 1$$

donde $\mathbf{H}$ *está dada por:*

$$\mathbf{H} = (h_{ij}) \in \Re^{pxp}, \qquad \text{con } h_{ij} = l_{ij}$$

NOTA 2: *La matriz de Comparación H es un límite superior (upper bound) de la función iteración* $\mathbf{H}(\mathbf{x}^k, k)$ [17].

En resumen, el radio espectral de la matriz de comparación es un parámetro que nos permitiría asegurar a priori que el algoritmo converge, y por consiguiente podría ser utilizado para escoger buenas particiones.

2.4.- Un método de resolución: Newton- Raphson

El método de Newton-Raphson es un método iterativo de resolución utilizado con sistemas no lineales de *n* ecuaciones con *n* incógnitas. Posee un desempeño muy satisfactorio, y además, la convergencia por el proporcionada es muy buena [3].

Considérese un sistema unidimensional del tipo

$$f(x) = 0, \qquad f: \mathbb{R} \to \mathbb{R}, \qquad x, 0 \in \mathbb{R} \tag{2.31}$$

Se buscará hallar la solución del sistema, es decir, el punto donde la función cruza el eje *x*. Para resolver este problema utilizando el método de Newton-Raphson deben seguirse los siguientes pasos [19]:

■

1. Inicializar las iteraciones haciendo $k=0$ y escoger una estimativa inicial para $k=0$ que se denota $x(0)$.
2. Evaluar la función $f(x)$ en el punto $x = x(k)$
3. Comparar el valor calculado $f(x(k))$ con la tolerancia especificada ε; si se verifica que $|f(x(k))| \leq \varepsilon$, entonces la solución buscada será $x = x(k)$ por encontrarse dentro

de los límites establecidos de tolerancia; si $|f(x(k))| > \varepsilon$, debe realizarse otra iteración.

4. Linealizar la función $f(x)$ en torno del punto $(\, x(k); f(x(k)) \,)$ por medio de la serie de Taylor (ver figura 2.1):

$$f(\; x(k) + \Delta x(k) \;) \cong f(\; x(k) \;) + f\,'(\, x(k) \,) \Delta x(k) \qquad (2.32)$$

siendo $f\,'(x) = \dfrac{df}{dx}$

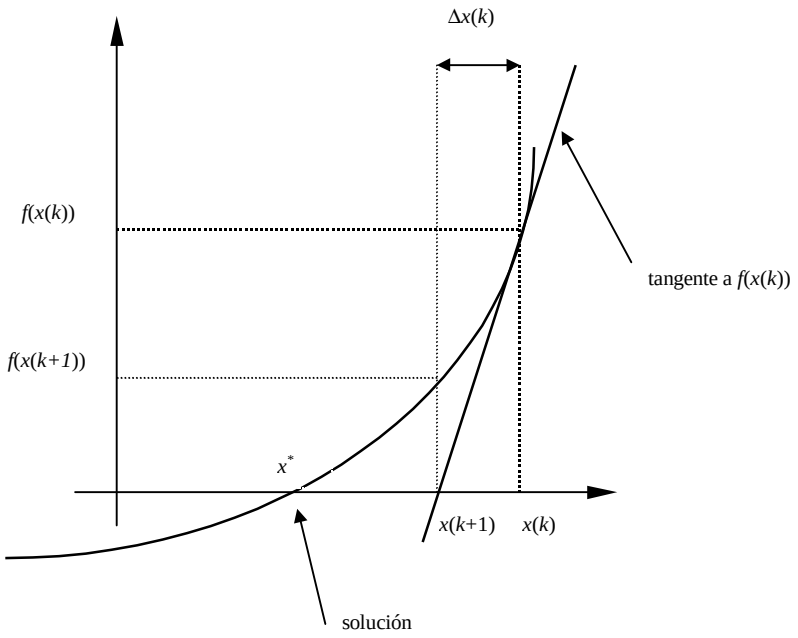

Figura 2.1: Método de Newton-Raphson

5.Resolver el problema linealizado, es decir, encontrar Δx tal que:

$$f(x(k)) + f'(x(k))\Delta x(k) = 0 \tag{2.33}$$

El nuevo valor calculado de x será entonces

$$x(k+1) = x(k) + \Delta x(k) \tag{2.34}$$

donde

$$\Delta x(k) = -\frac{f(x(k))}{f'(x(k))} \tag{2.35}$$

6.Hacer $k=k+1$ y regresar al paso 2.

■

Generalizando la ecuación (2.31) conforme a la notación de (2.7) a (2.11) para el caso multivariable, se tiene el siguiente sistema no lineal de ecuaciones:

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix} = \mathbf{0}, \qquad \mathbf{F}:\Re^n \to \Re^n, \qquad \mathbf{x},\mathbf{0} \in \Re^n \tag{2.36}$$

De acuerdo a lo mostrado más arriba, la resolución de este sistema de ecuaciones consistirá de los siguientes pasos:

●

1.Hacer $k = 0$ y escoger $\mathbf{x}(0)$

2.Evaluar $\mathbf{F}(\mathbf{x}(k))$

3.Verificar si $|\mathbf{F}(\mathbf{x}(k))| \leq \varepsilon$

4.Linealizar $\mathbf{F}(\mathbf{x})$ en torno al punto $(\mathbf{x}(k); (\mathbf{F}(\mathbf{x}(k))))$ según

$$\mathbf{F}(\mathbf{x}(k) + \Delta\mathbf{x}(k)) \cong \mathbf{F}(\mathbf{x}(k)) + \mathbf{J}(k)\Delta\mathbf{x}(k) \tag{2.37}$$

5. Actualizar **x** según $\mathbf{x}(k+1) = \mathbf{x}(k) + \Delta\mathbf{x}(k) = \mathbf{x}(k) - \mathbf{J}^{-1}(k)\mathbf{F}(\mathbf{x}(k))$ (2.38)

donde

$$\mathbf{J} = \left[\frac{\partial f_i}{\partial x_j}\right]$$

●

Cuando se aplica el método a un conjunto de *n* ecuaciones con *n* incógnitas, el jacobiano **J** es una matriz cuadrada cuyos elementos son iguales a las derivadas parciales de primer orden de cada función f_i con respecto a cada uno de los elementos del vector **x**. Para el sistema indicado en (2.36), se tendría entonces que:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \qquad (2.39)$$

En consecuencia, si queremos resolver (2.36) utilizando el método de Newton-Raphson en un sistema de *p* procesadores, debemos:

1º) Descomponer $\mathbf{F}(\mathbf{x})$ en *p* funciones $\mathbf{F}_i(\mathbf{x})$ conforme (2.11), <u>objetivo</u> de este trabajo.

2º) Resolver $\mathbf{F}_i(\mathbf{x}) = \mathbf{0}$ en el procesador *i* conforme

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) - \mathbf{J}_i^{-1}(k)\mathbf{F}_i(\mathbf{x}) \qquad (2.40)$$

donde, por ejemplo, para *i*=1

$$\Phi_1(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_{n_1}(\mathbf{x}) \end{bmatrix} \qquad \text{y} \qquad \mathbf{J}_1 = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{n_1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_1}}{\partial x_1} & \cdots & \frac{\partial f_{n_1}}{\partial x_{n_1}} \end{bmatrix}$$

2.5.- Solapamiento parcial

Las características particulares del sistema de ecuaciones a ser particionado pueden ocasionar que para ciertas ecuaciones (llamadas en el marco de este trabajo ecuaciones “criticas”) sea difícil realizar una decisión acertada en lo que respecta al procesador al cual serán asignadas.

Una manera de salvar esta dificultad es asignar dicha ecuación a más de un procesador, realizando el llamado *solapamiento parcial* [13], buscando de esta manera mejorar la convergencia del método de resolución. Para explicar mejor el concepto de solapamiento parcial, se presenta a continuación un ejemplo ilustrativo propuesto en [5]:

- Ejemplo 2.5.1

Resolver el siguiente sistema lineal de ecuaciones, utilizando el método bloque-Jacobi en un sistema distribuido de dos procesadores.

$$\begin{bmatrix} 2 & 10 & 0 \\ 0.08 & 1 & 0.08 \\ 0 & 10 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.41}$$

Este sistema de ecuaciones puede ser representado por el siguiente grafo:

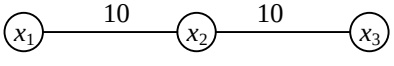

Figura 2.2: Grafo del sistema ejemplo

donde las ramas representan los acoplamientos entre las incógnitas, obtenidos a partir de los coeficientes del sistema de ecuaciones. De este modo, la incógnita x_2 se encuentra muy fuertemente acoplada con las incógnitas x_1 y x_3, las cuales no se encuentran acopladas entre sí. La influencia de estos acoplamientos en el comportamiento del método iterativo se verificará en los algoritmos de actualización de incógnitas que serán presentados en este ejemplo.

El sistema puede ser descompuesto de tres formas posibles:

$\{x_1, x_2\}$ y $\{x_3\}$
$\{x_1, x_3\}$ y $\{x_2\}$
$\{x_2, x_3\}$ y $\{x_2\}$

A continuación, y de manera a considerar el criterio de convergencia presentado en la sección 2.3, se calculará el radio espectral de la matriz de comparación para una de ellas, en la que a un procesador le son asignadas las dos primeras ecuaciones del sistema, mientras que al otro procesador se le asigna la tercera ecuación.

Así, y de acuerdo a lo mostrado en la sección 2.1, el algoritmo iterativo síncrono será:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 2 & 10 \\ 0.08 & 1 \end{bmatrix}^{-1} \left\{ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} [x_3(k)] \right\} \qquad \text{para el procesador 1} \qquad (2.42)$$

$$[x_3(k+1)] = 2^{-1} \left\{ b_3 - [0 \quad 10] \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \right\} \qquad \text{para el procesador 2} \qquad (2.43)$$

Se puede apreciar que el elemento de valor 10 en la ecuación (2.43) refleja el grado de dependencia (acoplamiento) que tiene la incógnita x_3 con respecto a la incógnita x_2, actualizada por el otro procesador.

Para obtener la matriz de comparación , se impone primero conocer la matriz de iteraciones **M**, que de acuerdo a (2.41) y (2.42) será:

$$\mathbf{M} = \begin{bmatrix} 0 & \begin{bmatrix} 2 & 10 \\ 0.08 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0.08 \end{bmatrix} \\ 2^{-1}\begin{bmatrix} 0 & 10 \end{bmatrix} & 0 \end{bmatrix} \tag{2.44}$$

Los elementos de la matriz de comparación **H** son las normas de las matrices bloques que componen la matriz de iteraciones. Nótese que la matriz de iteraciones en este caso tiene la dimensión del problema, esto es: 3x3, mientras que la matriz de comparación H tendrá la dimensión del número de procesadores, esto es: 2x2. Entonces:

$$\mathbf{H} = \begin{bmatrix} 0 & 0.66 \\ 5 & 0 \end{bmatrix} \tag{2.45}$$

Dado que el radio espectral ρ de una matriz se define como el máximo de sus autovalores, para este ejemplo se tiene:

$$\rho(\mathbf{H}) = \sqrt{0.66 \times 5} = 1.8 > 1 \tag{2.46}$$

Podemos decir así que la partición utilizada no presenta buenas perspectivas de convergencia, por no satisfacer la condición establecida en el Teorema 1. Las otras dos particiones posibles tampoco cumplen con la condición deseada de que $\rho(\mathbf{H}) < 1$, por lo que concluimos que no es posible resolver el problema (2.41) de forma efectiva utilizando las particiones disponibles. En consecuencia, utilizaremos la técnica de solapamiento parcial para resolver (2.41).

Para esto, se partirá el problema anterior de la siguiente manera: la segunda ecuación del sistema (ecuación crítica) se replicará en ambos procesadores, esto es, el procesador 1 resolverá las ecuaciones 1 y 2 del sistema, mientras el procesador 2 resolverá las ecuaciones 2 y 3. En este caso, el nuevo sistema de ecuaciones a ser resuelto, denominado *sistema expandido* [13], tiene una dimensión expandida n`=4, y puede ser representado por:

$$\begin{bmatrix} 2 & 10 & 0 & 0 \\ 0.08 & 1 & 0 & 0.08 \\ 0.08 & 0 & 1 & 0.08 \\ 0 & 0 & 10 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_{21} \\ x_{22} \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.47}$$

Note que ahora tenemos 2 versiones diferentes de la variable x_2, estas son: x_{21} y x_{22}. En consecuencia, cada procesador implementará el algoritmo de Jacobi según se muestra a continuación:

$$\begin{bmatrix} x_1(k+1) \\ x_{21}(k+1) \end{bmatrix} = \begin{bmatrix} 2 & 10 \\ 0.08 & 1 \end{bmatrix} \left\{ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0.08 \end{bmatrix} \begin{bmatrix} x_{22}(k) \\ x_3(k) \end{bmatrix} \right\} \quad \text{para el procesador 1} \tag{2.48}$$

$$\begin{bmatrix} x_{22}(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.08 \\ 10 & 2 \end{bmatrix} \left\{ \begin{bmatrix} b_2 \\ b_3 \end{bmatrix} - \begin{bmatrix} 0.08 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_{21}(k) \end{bmatrix} \right\} \quad \text{para el procesador 2} \tag{2.49}$$

En forma análoga al caso anterior, se construye la matriz de iteraciones:

$$\mathbf{MI} = \begin{bmatrix} 0 & \begin{bmatrix} 2 & 10 \\ 0.08 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ 0 & 0.08 \end{bmatrix} \\ \begin{bmatrix} 1 & 0.08 \\ 10 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0.08 & 0 \\ 0 & 0 \end{bmatrix} & 0 \end{bmatrix} \tag{2.50}$$

y la correspondiente matriz de comparación:

$$\mathbf{H} = \begin{bmatrix} 0 & 0.66 \\ 0.66 & 0 \end{bmatrix} \tag{2.51}$$

cuyo radio espectral $\rho(\mathbf{H})$ será:

$$\rho(\mathbf{H}) = \sqrt{0.66 \times 0.66} = 0.66 < 1$$

que por el Teorema 1 tiene asegurada su convergencia aún en un contexto computacional asíncrono.

En conclusión, el solapamiento parcial permite resolver problemas que no podrían ser resueltos con una simple partición de las ecuaciones de un problema.

CAPITULO 3: EL FLUJO DE POTENCIA ELECTRICA

3.1.- La red eléctrica: modelo matemático

La formulación de un modelo matemático adecuado a las características de la red y al tipo de estudio a ser realizado es el paso inicial para el análisis y resolución de un sistema eléctrico. El modelo matemático que aquí se presenta formulará las ecuaciones necesarias para proceder a la partición del sistema, asignando a los distintos procesadores del sistema distribuido las correspondientes ecuaciones a ser resueltas.

En la abundante bibliografía disponible sobre este punto particular, se destacan sobre otras, dos formas de representar una red eléctrica: ecuaciones de nudos y ecuaciones de lazos o mallas. En lo que sigue de la sección se utilizará la primera de ellas.

En este caso, las variables del sistema son las tensiones complejas (módulo y fase) en los nudos y las corrientes nodales. Se llama corriente nodal a aquella que entra o sale de la red por un nudo determinado; es entrante (considerada positiva) cuando proviene de una fuente, y es saliente (considerada negativa) si se dirige a una carga. La corriente nodal inyectada es la suma algebraica de las dos.

El conjunto completo de ecuaciones de nodos que definen una red se puede expresar en forma matricial como:

$$\begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1n} \\ Y_{21} & Y_{22} & \cdots & Y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{n1} & Y_{n2} & \cdots & Y_{nn} \end{bmatrix} \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ En \end{bmatrix} - \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} \qquad (3.1)$$

donde

$Y_{ii} = \sum_{\substack{m=0 \\ m \neq i}}^{n} y_{im}$ es la admitancia propia de la barra *i*.

$Y_{im} = -y_{im}$ es la admitancia mutua entre las barras *i* y *m*.

La ecuación matricial (3.1) puede ser expresada en la forma:

$$\mathbf{YE} = \mathbf{I}(\mathbf{E}) \tag{3.2}$$

donde

$\mathbf{Y} \in C^{n \times n}$ es la matriz admitancia

$\mathbf{I} \in C^{n}$ es el vector de corrientes inyectadas

$\mathbf{E} \in C^{n}$ es el vector de tensiones (módulo y fase)

La matriz **Y** es compleja, extremadamente esparza, simétrica, no posee estructura definida y sus elementos proporcionan información sobre las ligaciones entre los nudos del sistema eléctrico.

3.2.- Flujo de Potencia en un sistema eléctrico

Para poder definir con propiedad el problema del Flujo de Potencia, se deben identificar cuatro variables en cada barra *i* del sistema:

P_i = Potencia real o activa

Q_i = Potencia reactiva o de cuadratura

V_i = Módulo de la tensión E_i

θ_i = Fase de la tensión E_i

Inicialmente solo se conocen dos de las cuatro variables para cada barra *i*, y el objetivo de la resolución del Flujo de Potencia es calcular las otras dos variables implicadas en el problema.

En un sistema eléctrico, tres tipos diferentes de barras (nudos) pueden ser diferenciadas [3]:

1. Barras PV o barras de tensión controlada: la potencia activa total inyectada está especificada, y el módulo del voltaje es mantenido a un valor determinado por medio de la inyección de potencia reactiva. Este tipo de barra por lo general corresponde a un generador donde P_i está predeterminada por los controladores del flujo de la turbina, mientras V_i esta fijado por reguladores automáticos de voltaje que actúan sobre la excitación de la máquina. También puede fijarse el valor del módulo de la tensión en la barra al inyectar potencia reactiva por medio de capacitores en paralelo o compensadores rotativos síncronos. Las subestaciones eléctricas podrían encontrarse entre este tipo de barras.
2. Barras PQ o de tensión no controlada: en este tipo de barras, la potencia total inyectada P_i+jQ_i está especificada. Este tipo de barra se corresponde físicamente con un centro de consumo, como por ejemplo, una ciudad o centro industrial. Se supone que tanto P_i como Q_i no son afectadas por pequeñas variaciones en la tensión de la barra.
3. Barra Slack u oscilante: debido a que las pérdidas del sistema no son conocidas, se requiere la existencia de una barra de este tipo, en la cual se desconocen tanto la potencia activa P_i como la reactiva Q_i. Usualmente se asigna alguna de las barras de voltaje controlado como slack, suponiendo desconocido el valor de su potencia activa P_i. El voltaje de la barra slack es comúnmente usado como referencia de fase, por lo cual el mismo debe estar especificado. La analogía en

sistemas eléctricos prácticos es la estación generadora de energía, la cual carga con la tarea de regular la frecuencia del sistema.

De acuerdo a la ecuación (3.1), la corriente nodal inyectada en una barra *i* puede ser expresada como [3]

$$I_i = \sum_{m \in i} Y_{im} E_m \tag{3.3}$$

La potencia inyectada en una barra *i* esta dada por:

$$S_i = P_i + jQ_i = E_i I_i^*$$

$$= E_i \sum_{m \in i} Y_{im}^* E_m^* \tag{3.4}$$

Realmente, las ecuaciones complejas del Flujo de Potencia no son analíticas, y no pueden ser diferenciadas en su forma compleja. Para que el método de Newton-Raphson pueda ser aplicado, el problema es separado en ecuaciones y variables reales. Utilizando coordenadas polares en (3.4) se obtienen las siguientes ecuaciones por barra:

$$P_i = \sum_{m \in i} V_i V_m (G_{im} \cos_{\theta im} + B_{im} \operatorname{sen}_{\theta im}) \tag{3.5}$$

$$Q_i = \sum_{m \in i} V_i V_m (G_{im} \operatorname{sen}_{\theta im} - B_{im} \cos_{\theta im}) \tag{3.6}$$

con $\theta_{im} = \theta_i - \theta_m$

Relaciones lineales son obtenidas para pequeñas variaciones en las variables $\boldsymbol{\theta}$ y **V** al formar los diferenciales totales, siendo las ecuaciones resultantes las siguientes:

$$\Delta P_i = \sum_{m \in i} \frac{\partial P_i}{\partial \theta_m} \Delta \theta_m + \sum_{m \in i} \frac{\partial P_i}{V_i} \Delta V_m \tag{3.7}$$

y

$$\Delta Q_i = \sum_{m \in i} \frac{\partial Q_i}{\partial \theta_m} \Delta \theta_m + \sum_{m \in i} \frac{\partial Q_i}{V_i} \Delta V_m \tag{3.8}$$

Para una barra PV, se conocen las potencias activas de generación y carga, respectivamente, así como también el valor en módulo de la tensión local; las incógnitas a ser calculadas son el ángulo de fase de la tensión local y la potencia reactiva inyectada.

En una barra PQ, se conocen las potencias activas y reactivas de carga y generación, debiéndose calcular el módulo y la fase de la tensión local.

A partir del método de Newton-Raphson, el algoritmo (2.35) puede ser escrito en su forma matricial de la siguiente manera [3]:

$$\begin{bmatrix} \Delta \mathbf{P}(k) \\ \Delta \mathbf{Q}(k) \end{bmatrix} = \begin{bmatrix} \mathbf{H}(k) & \mathbf{N}(k) \\ \mathbf{J}(k) & \mathbf{L}(k) \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{\theta}(k) \\ \dfrac{\Delta \mathbf{V}(k)}{\mathbf{V}(k)} \end{bmatrix} \tag{3.9}$$

donde:

$\Delta\mathbf{P}$ vector de errores de P para todas las barras PQ y PV

$\Delta\mathbf{Q}$ vector de errores de Q para todas la barras PQ

$\Delta\boldsymbol{\theta}$ vector de correcciones de $\boldsymbol{\theta}$ para todas la barras PQ y PV

$\Delta\mathbf{V}$ vector de correcciones de $\mathbf{V}$ para todas las barras PQ

Las submatrices $\mathbf{H} = \{h_{im}\} \in \Re^{a_1 \times a_1}$, $\mathbf{N} = \{n_{im}\} \in \Re^{a_1 \times a_2}$, $\mathbf{J} = \{j_{im}\} \in \Re^{a_2 \times a_1}$ y $\mathbf{L} = \{l_{im}\} \in \Re^{a_2 \times a_2}$, componentes del jacobiano, poseen los siguientes elementos:

Para $m \neq i$

$$H_{im} = \frac{\partial P_i}{\partial \theta_m} = V_i V_m (G_{im} \, sen_{\theta_{im}} - B_{im} \cos_{\theta_{im}}) \tag{3.10}$$

$$N_{im} = \frac{\partial P_i}{\partial V_m} = V_i V_m (G_{im} \cos_{\theta_{im}} + B_{im} \, sen_{\theta_{im}}) \tag{3.11}$$

$$J_{im} = \frac{\partial Q_i}{\partial \theta_m} = -V_i V_m (G_{im} \cos_{\theta_{im}} + B_{im} \, sen_{\theta_{im}}) \tag{3.12}$$

$$L_{im} = V_m \frac{\partial Q_i}{\partial V_m} = V_i V_m (G_{im} \, sen_{\theta_{im}} - B_{im} \cos_{\theta_{im}}) \tag{3.13}$$

y para $m = i$

$$H_{ii} = \frac{\partial P_i}{\partial \theta_i} = -Q_i - B_{ii} V_i^2 \tag{3.14}$$

$$N_{ii} = V_i \frac{\partial P_i}{\partial V_i} = P_i + G_{ii} V_i^2 \tag{3.15}$$

$$J_{ii} = \frac{\partial Q_i}{\partial \theta_i} = P_i - G_{ii} V_i^2 \tag{3.16}$$

$$L_{ii} = V_i \frac{\partial Q_i}{\partial V_i} = Q_i - B_{ii} V_i^2 \tag{3.17}$$

En la sección 2.4 se han expuesto con detalle los pasos a seguir para aplicar el método de Newton-Raphson. A cada iteración, los valores de los módulos y las fases de las tensiones se actualizan al sumar los incrementos calculados a partir de (3.9) y se verifica si el método ya llegó a la solución comparando las potencias calculadas con las potencias que son datos del sistema.

3.3.- Resolución distribuida del Flujo de Potencia

En el contexto del procesamiento paralelo donde varios procesadores resuelven su subproblema local, es necesario asignar a cada procesador un grupo determinado de barras. Una vez que esto es realizado, cada procesador resolverá de forma local el problema del Flujo de Potencia para su subred, asumiendo conocidos los valores de tensión de las demás barras del sistema eléctrico.

En concordancia con lo expuesto en la formulación matemática del Capítulo 2, el sistema global de ecuaciones a ser resuelto tendrá la forma

$$\underset{\sim}{\Phi}(\mathbf{x}) = \mathbf{S} - \mathbf{EI}^{*} = \mathbf{0} \tag{3.18}$$

donde

$$\mathbf{x} = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_n \\ V_1 \\ \vdots \\ V_n \end{bmatrix}, \qquad \rho_i, V_i \in \Re \quad ; \tag{3.19}$$

$$\mathbf{S} = \begin{bmatrix} S_1 \\ \vdots \\ S_n \end{bmatrix}, \qquad S_i \in C \tag{3.19}$$

y

$$\mathbf{EI}^{*} = \begin{bmatrix} E_1 \sum_{m \in 1} Y_{1m}^{*} E_m^{*} \\ \vdots \\ E_n \sum_{m \in n} Y_{nm}^{*} E_m^{*} \end{bmatrix}, \qquad E_i, Y_{im} \in C \tag{3.20}$$

Una vez que un subsistema haya sido asignado a cada procesador *i*, el problema local a ser resuelto por dicho procesador *i* será, conforme (2.11) y (2.12):

$$\Phi_i(\mathbf{x}) = \mathbf{S}_i - \mathbf{E}_i \mathbf{I}_i^* = \mathbf{0} \tag{3.21}$$

con $S_i , E_i , I_i^* \in C^{n_i}$, donde n_i es el número de barras asignadas al procesador *i*.

Tenemos así que cada procesador *i* actualiza las variables correspondientes al subsistema eléctrico a él asignado, utilizando el algoritmo (2.34), para lo cual se necesitan los valores de tensión calculados por los otros procesadores, pues estos son necesarios en el cálculo de $\Delta\mathbf{P}$ y $\Delta\mathbf{Q}$.

Como puede verificarse en las ecuaciones (3.5) y (3.6), las incógnitas calculadas por los demás procesadores en la iteración anterior son utilizadas localmente en el cálculo del *mismatch* (error) de potencias.

Entonces, en cada iteración, y a partir de los incrementos calculados, cada procesador *i* actualiza sus incógnitas locales θ_i y $\mathbf{V}_i$, comunicando sus resultados parciales a los demás procesadores del sistema distribuido, avanzando así hacia la solución global del problema.

CAPITULO 4: METODOS DE DESCOMPOSICION

Como se había indicado en la revisión bibliográfica del Capítulo 1, hoy en día se dispone de varios métodos que son utilizados en la partición de sistemas eléctricos de gran porte, presentando algunos de ellos características que los hacen muy interesantes en el contexto del procesamiento paralelo.

En el presente capítulo se expondrán los principios básicos de los dos métodos de partición más utilizados de los hasta ahora publicados, que han sido implementados con éxito en diversas aplicaciones de ingeniería y la ciencia en general Dichos métodos son la *Descomposición* ε y el *método de la semilla*. Este último método es de especial importancia en el marco del presente trabajo, por servir de base e inspiración al mismo.

4.1.- La Descomposición ε

Desde finales de la década del 60, con la publicación del trabajo de Carré [11], se puso de manifiesto que el acoplamiento entre las subredes en que el sistema es descompuesto influye de manera decisiva en la eficiencia de los métodos de resolución, aún cuando originalmente dichos métodos hayan sido aplicados en un ambiente secuencial y no paralelo.

Basándose en este hecho, la Descomposición ε [26, 27, 28] introduce un método de partición de gran simplicidad. Se obtienen en esta aplicación diversas descomposiciones que presentan una gran variedad en las dimensiones de los subsistemas generados, así como también en la fuerza de sus acoplamientos mutuos. El objetivo básico de este método es separar la red en los puntos en los que el acoplamiento entre barras sea lo más débil posible, a través de la eliminación de las ramas cuyo módulo de admitancia sea menor que un límite inferior dado ε. Esta

eliminación de ramas se repite utilizando valores de ε cada vez mayores, hasta que el sistema quede efectivamente descompuesto en varios subsistemas. Como se verá más adelante, las dimensiones de los subsistemas no pueden ser controladas de una manera efectiva, al depender éstas exclusivamente de las características físicas de cada sistema.

Para ilustrar la aplicación de la Descomposición ε, considérese el sistema eléctrico representado por el grafo que se muestra en la figura 4.1.

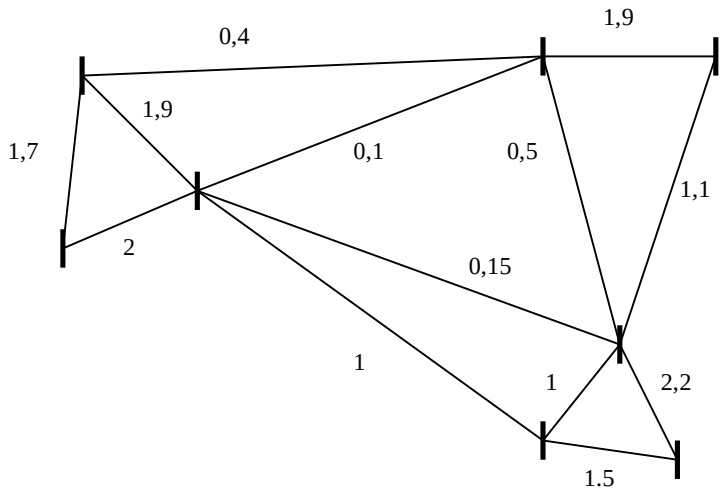

Figura 4.1 : Descomposición ε: Sistema eléctrico ejemplo

Los nodos del grafo representan las barras del sistema eléctrico, mientras que las ramas representan las líneas que las unen. Los números adosados a las ramas representan los valores en p.u del módulo de las susceptancias de las ramas.

A continuación se determina un límite inferior ε que será utilizado como referencia para seleccionar las ramas que serán eliminadas. Para el ejemplo, se utilizará inicialmente ε=0.6 , eliminándose del grafo las ramas cuyos valores de módulo de susceptancia sean menores que 0.6. Así, el sistema quedará como se muestra en la figura 4.2

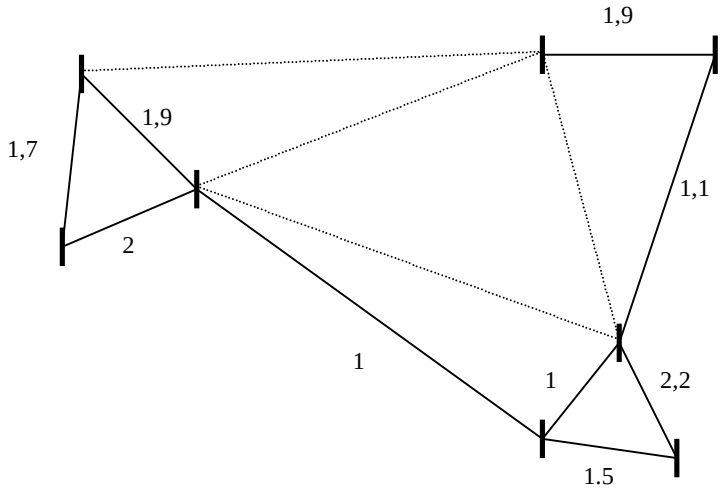

Figura 4.2 : Aplicación de la Descomposición ε: situación 1

Como aún no se ha logrado particionar el sistema, el paso anterior se repite tomando un valor de ε mayor, por ejemplo ε=1.2. Eliminando las barras con valores menores que 1.2 , el sistema quedará separado en 3 subredes menores según se observa en la figura 4.3.

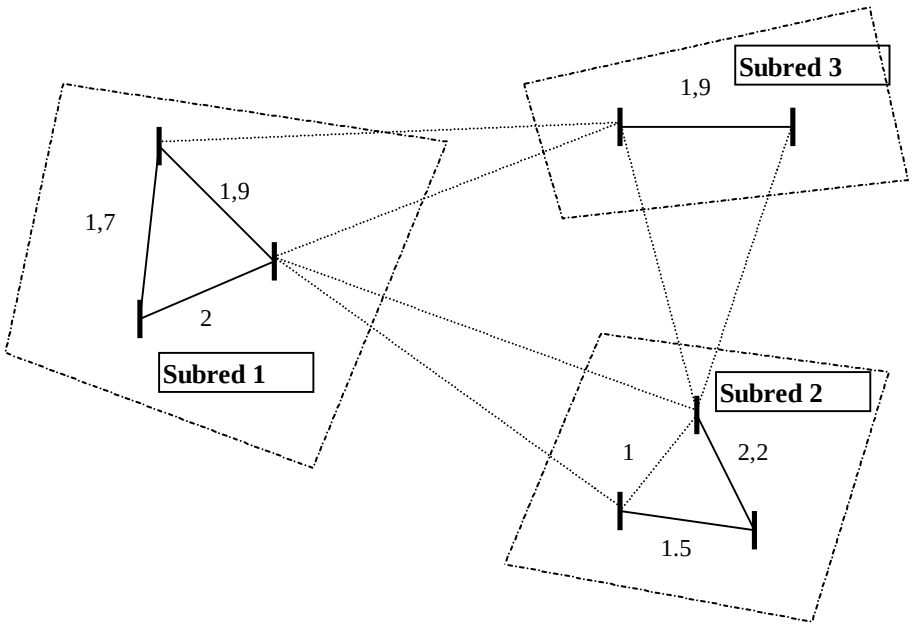

Figura 4.3 : Aplicación de la Descomposición ε: situación 2

Puede verse que las subredes generadas por el método expuesto están débilmente acopladas, que era lo que el método buscaba, pero las dimensiones relativas y el número de subredes obtenidas dependieron exclusivamente de las características particulares del ejemplo.

4.2.- El Método de la Semilla

Este método, presentado por M.H.Vale, D.M. Falcão y E. Kaszkurewicz [31], es una metodología de partición basada en la identificación de grupos de barras fuertemente acopladas, para descomponer así el sistema en subredes que contengan a cada uno de dichos centros de agrupamiento.

El principio básico de esta metodología es la formación de subredes a partir de un dado conjunto de barras *semillas*. Las subredes se forman agregando barras una a una a estas barras semillas, utilizándose como criterio de anexación un ranking de pesos previamente asignados a todas las barras del sistema. En cada paso del proceso, la siguiente barra a ser agregada a una subred en formación es aquella barra adyacente (unida por alguna rama a la subred) de mayor peso y que aún no haya sido asociada por las demás subredes en formación.

El ranking de pesos w_i de las barras se realiza en base a la fórmula

$$w_i = \sum_{l \in \Omega_i} B_l^{B_l / M}, \qquad \text{con} \qquad M = \sum_{l \in \Omega_T} \frac{B_l}{b} \tag{4.1}$$

donde:

Ω_i: Conjunto de todas las ramas conectadas a la barra i.

Ω_T: Conjunto de todas las ramas del sistema eléctrico.

B_l: Susceptancia de la rama l.

b: Número de ramas del sistema eléctrico.

La selección de las barras semillas se realiza en base a la determinación de un conjunto de barras candidatas a semillas. Este conjunto contiene a todas las barras que poseen un valor de peso superior a un determinado valor *vlim*.

A partir de cada barra del conjunto así determinado, e independientemente de las otras barras del conjunto, se agrupan las barras más pesadas de su entorno, pretendiéndose con esto identificar a las barras se rodean con acoplamientos más fuertes. No es preciso agrupar a todas las barras de la red en torno a cada barra candidata, sino sólo a *nagrup* barras.

Una vez realizado esto, se calculan los valores de las sumatorias de los pesos de las barras agrupadas alrededor de cada candidata a semillas seleccionándose las barras semillas de entre aquellas que posean los mayores valores de sumatoria. Para evitar que sean seleccionadas barras muy próximas, con fuertes acoplamientos entre sí, se impone que una barra semilla no se encuentre entre las *nvec* primeras barras del agrupamiento de las barras semillas seleccionadas anteriormente.

A continuación, y a manera de ilustración, se aplicará el método de la semillas en la descomposición del sistema ejemplo utilizado en la sección 4.1, buscando particionar la red de manera a resolverla en dos procesadores. Los valores de los parámetros a ser utilizados serán:

- *vlim* = 5.5
- *nagrup* = 2
- *nvec* = 2

Estos valores fueron determinados de manera a ofrecer una explicación lo más clara posible del método aquí expuesto. Para problemas reales, dichos valores son establecidos conforme a la experiencia y conocimiento que el operador posea del sistema eléctrico analizado.

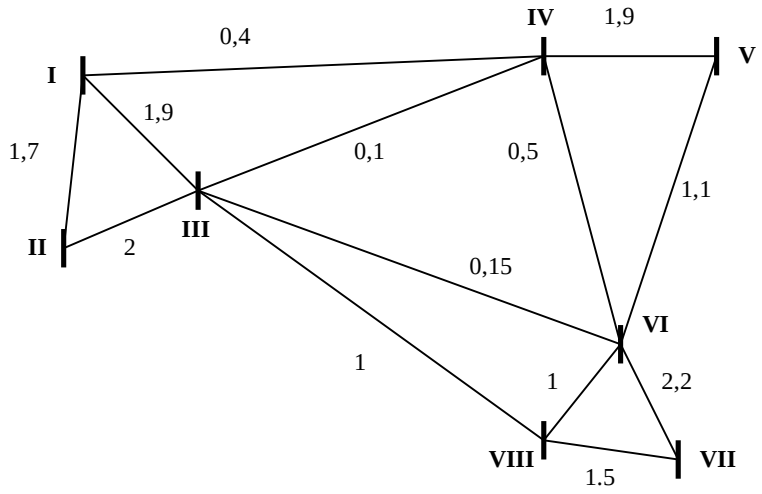

Figura 4.4 : Método de la semilla: Sistema eléctrico ejemplo

Como primer paso, se calcularon los pesos de las barras conforme a (4.1), resultando los siguientes valores:

$w_{I} = 5.65$

$w_{II} = 5.34$

$w_{III} = 8.6$

$w_{IV} = 5.08$

$w_{V} = 3.88$

$w_{VI} = 7.91$

$w_{VII} = 5.96$

$w_{VIII} = 3.66$

Tomando *vlim*=5.5, se determina el conjunto de barras candidatas a semillas, cuyos elementos serán las barras **III**, **VI**, **VII** y **I**, así ordenadas por peso (ver tabla 4.1). A continuación, alrededor de estas 4 barras se agrupan las *nagrup* barras más pesadas , calculándose luego la sumatoria de los pesos de las barras agrupadas. Para la selección realizada con *nagrup*=2, se tienen los siguientes agrupamientos:

Candidata	Barras agrupadas	Σw_i
III	**VI, VII**	22.47
VI	**III, VII**	22.47
VII	**VI, III**	22.47
I	**III, VI**	22.16

Tabla 4.1: Agrupamientos de barras

A pesar de existir coincidencia en las sumatorias de pesos, la primera barra seleccionada como semilla es la barra **III**, por poseer ésta individualmente el mayor valor de peso. La segunda barra con el mayor valor de sumatoria es la barra **VI**, pero antes de seleccionarla como semilla debe verificarse que no se encuentre entre las *nvec* primeras barras agrupadas alrededor de la barra **III**. Tomando *nvec*=2, se puede ver que la barra **VI** no puede ser seleccionada como semilla, por ser la primera barra del agrupamiento de la primera semilla seleccionada. La barra **VII**, que es la siguiente candidata a semilla con la mayor sumatoria de pesos, tampoco puede ser seleccionada, por encontrarse entre las *nvec* primeras barras del agrupamiento de la barra **III**. Se puede ver que la barra **I**, que es la última barra candidata, no se encuentra siquiera en el agrupamiento de la barra **III**, por lo que se la selecciona como segunda semilla.

De este modo las barras **I** y **III** quedan seleccionadas como semillas y ya se dispone de la cantidad de semillas necesarias de forma a particionar el sistema en las dos subredes deseadas. Es de esperar que las semillas seleccionadas sean centros de agrupamientos de barras, de manera a dar origen a subredes poco acopladas entre sí.

Una vez determinadas las barras semillas, se inicia el proceso de descomposición. Para esto, cada semilla selecciona a la más pesada de entre sus

adyacentes, y la incluye en su subred en formación. Tenemos así que, para nuestro ejemplo, la semilla **III** incluirá a la barra **VI**, mientras que la semilla **I** hace lo propio con la barra **II**. Las subredes en formación quedarán entonces como se muestra en la figura 4.5.

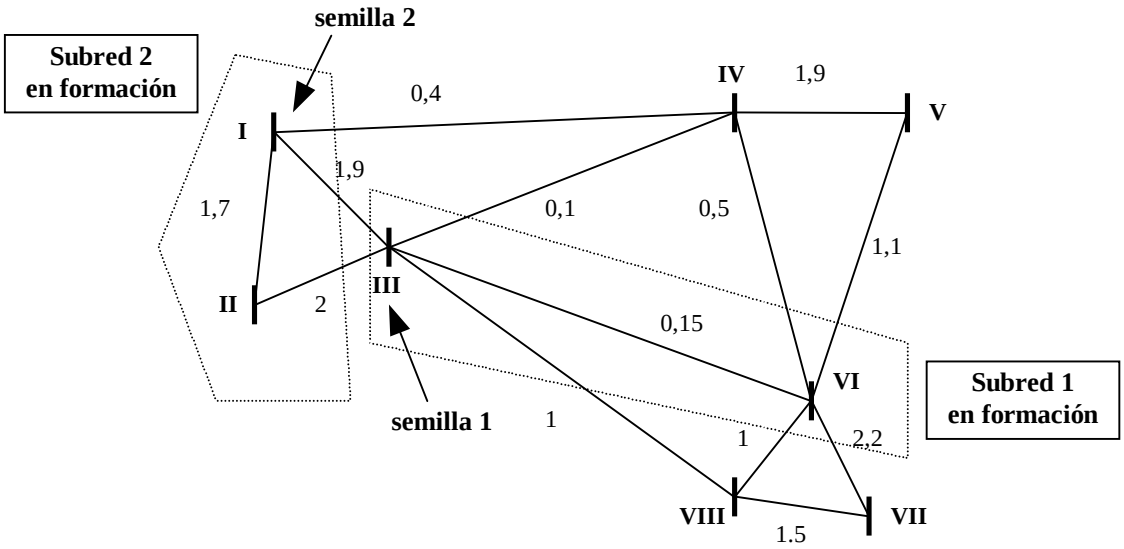

Figura 4.5 : Aplicación del método de la semilla

El proceso continúa hasta que no hayan más barras por ser anexadas, quedando así la red descompuesta en tantas subredes como semillas haya. Para nuestro ejemplo, al finalizar el proceso de partición el sistema quedará descompuesto como se muestra en la figura 4.6, donde se puede ver que el método de la semilla solamente genera particiones balanceadas, es decir, constituidas por subproblemas de dimensiones similares.

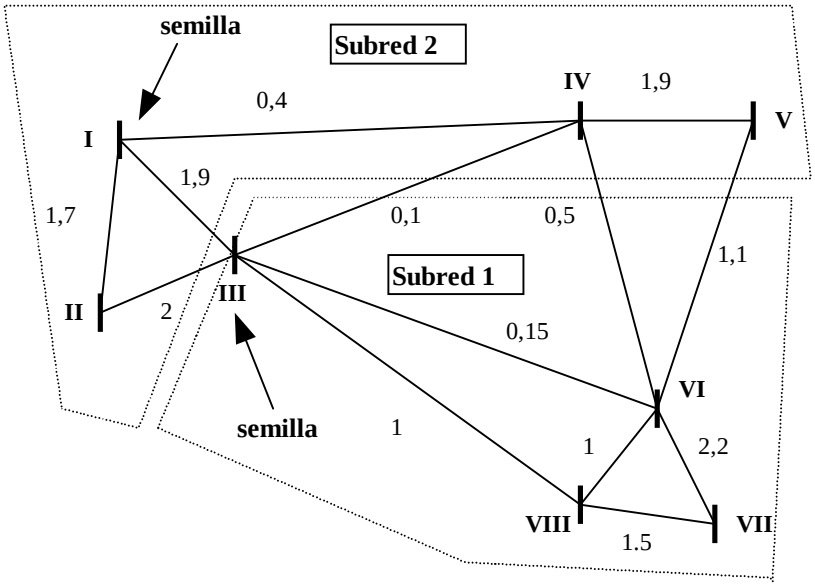

Figura 4.6 : Método de la semilla: partición generada

De este modo, el presente método genera particiones constituidas por subredes poco acopladas entre sí, lo que resulta muy beneficioso para el desempeño de los métodos de resolución bloque-iterativos. Con todo, el método no posee dominio sobre las dimensiones relativas de las subredes generadas, ni considera la posibilidad de realizar solapamiento parcial.

Basado en el método de la semilla, el método propuesto en el presente trabajo salva las limitaciones citadas, permitiendo así una mejor distribución de la carga computacional asignada a cada procesador, a la vez de ofrecer la posibilidad de realizar solapamiento parcial con miras a mejorar la convergencia del método de resolución empleado, conforme se ilustra en el próximo capítulo.

CAPITULO 5: METODO DE DESCOMPOSICION PROPUESTO

5.1.- Consideraciones Iniciales

Una buena descomposición del sistema eléctrico es sumamente importante, dado que sin ella no sería posible obtener resultados satisfactorios al utilizar los métodos bloque-iterativos en un ambiente de procesamiento paralelo para la resolución de los diversos problemas de ingeniería.

Se pudo observar que la manera en la cual un sistema es descompuesto tiene un efecto significativo en la convergencia de los métodos bloque-iterativos. Para obtener buenos resultados en términos de convergencia se debe descomponer el sistema de tal forma que el acoplamiento entre subsistemas sea lo más débil posible [9]. El acoplamiento entre dos barras esta expresado por el valor de las admitancias de la rama que las une (en p.u.). Como dicha admitancia posee por lo general una componente imaginaria (susceptancia) cuyo módulo es mucho mayor que su componente real (conductancia), en el marco del presente trabajo se consideró al valor de la susceptancia entre dos barras como una medida efectiva del acoplamiento entre ambas. Sin embargo, no habría ninguna dificultad en utilizar la admitancia $|y|$ en su lugar.

También ha podido constatarse que los métodos de partición descritos en el Capítulo 4 no se ocupan de manera especial en generar una descomposición de la red acorde con las capacidades de procesamiento relativas de los procesadores que constituyen el sistema distribuido, esto es, consideran sistemas distribuidos homogéneos, es decir, constituidos por procesadores de igual desempeño.

Esto último es de singular importancia, porque en muchos casos, especialmente en nuestro medio, es sumamente difícil conseguir varios ordenadores

de características similares. Es mas común el disponer de varias máquinas de capacidad y desempeño diferentes, constituyendo así un ambiente computacional heterogéneo, dado que los procesadores de la red tienen capacidades de procesamiento generalmente diferentes.

Considerando la posibilidad de operar en un ambiente de dichas características, formado por procesadores de diversas performances, interconectados por un sistema de comunicación, la descomposición también estará condicionada por la capacidad de procesamiento relativo de cada máquina del sistema distribuido. Este factor determina las dimensiones de los subproblemas asignados a cada procesador.

La técnica de descomposición debe por lo tanto perseguir los siguientes objetivos de interés:

- Convergencia del método de solución: en este sentido, debe buscarse un acoplamiento débil y el menor número posible de interligaciones entre las subredes interconectadas. Esto último es con el fin de minimizar la comunicación entre procesadores y mejorar la convergencia del conjunto.
- Eficiencia computacional del método de solución en términos del procesamiento paralelo: debe buscarse para esto el balanceamiento de la carga computacional entre los distintos procesadores que componen el sistema distribuido. El parámetro a ser utilizado como criterio de evaluación de la eficiencia computacional es el tiempo total de ejecución del método de solución en la resolución del problema eléctrico dado.

Una vez identificados los objetivos perseguidos, la formulación del problema de la descomposición podría entenderse como: "*una técnica que debe descomponer un sistema eléctrico en un número dado de subredes interconectadas entre sí de tal forma que su resolución en un sistema distribuido heterogéneo utilizando métodos bloque-iterativos sea obtenida en el menor tiempo de procesamiento posible*".

Atendiendo a esto, el método de descomposición propuesto utiliza la matriz de admitancias **Y** como parámetro de acoplamiento. Dicha matriz tiene las siguientes características: es cuadrada, de dimensión $n_x n$, simétrica ($y_{ij}=y_{ji}$), es extremadamente esparza (aproximadamente 4 elementos no nulos por fila/columna en promedio) y posee elementos diagonales no nulos ($y_{ij} \neq 0$). El elemento y_{ij} representa el acoplamiento existente entre las barras B_i y B_j, por lo tanto, decimos que las barras B_i y B_j no son adyacentes si $y_{ij} = 0$; caso contrario B_i y B_j son adyacentes.

Si y_{ij} es pequeño (en el caso límite $y_{ij} \rightarrow 0$) el acoplamiento existente entre las barras B_i y B_j es débil, con poca dependencia entre ellas, pudiendo por lo tanto estas formar parte de subredes diferentes. En consecuencia, el método buscará separar el sistema eléctrico en los puntos donde el acoplamiento entre las barras sea lo más débil posible.

Suponiendo que el sistema eléctrico será resuelto utilizando *p* procesadores, el principio básico del método consistirá en la formación de las *p* subredes a partir de *p* barras iniciales, llamadas *semillas* y que cumplen con la función de ser centros de acoplamientos de barras.

Una vez determinadas estas semillas para cada subred, de acuerdo al algoritmo de selección de semillas que será presentado en esta sección, es necesario adicionar las demás barras a la subred adecuada hasta tener el sistema totalmente particionado en *p* subsistemas. Para realizar esto se asocia la barra adyacente de mayor peso a cada subred en formación. El proceso de agrupamiento de barras continua hasta que todas ellas hayan sido asignadas a alguna subred. En caso de obtenerse más de una descomposición para un mismo sistema, un criterio de selección de descomposiciones será presentado con vista a elegir la partición más conveniente.

Basados en el principio descrito, se propone un método compuesto por cuatro etapas, esquematizado en a la figura 5.1.

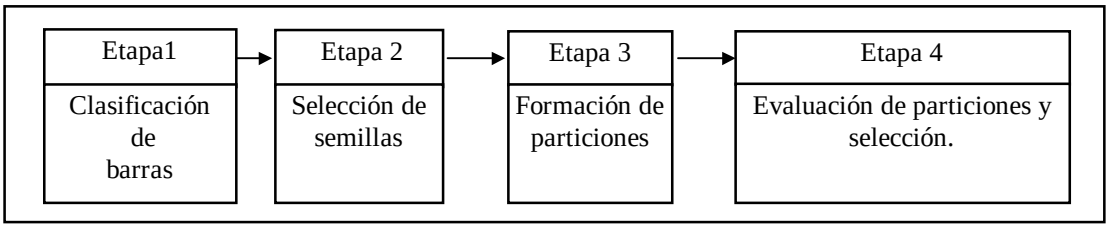

Figura 5.1: Etapas del método de partición propuesto.

5.2.- Etapa 1: Clasificación de barras

Como se ha resaltado con anterioridad, uno de los objetivos principales a ser perseguido por un buen método de partición es realizar cortes en los puntos donde el acoplamiento entre barras sea lo mas débil posible. Para esto, se impone conocer que tan acoplada se encuentra una barra con las demás barras de la red. Este grado de acoplamiento será medido en el marco de este trabajo por el valor de un cierto "peso" calculado para cada barra.

Sea $nc = número\ de\ y_{ij} \neq 0$

Se denota como *D* a la media aritmética de los valores absolutos de los elementos no diagonales de *Y* excluyendo a los elementos nulos; esto es:

$$D = \frac{1}{nc} \sum_{i=1}^{n} \sum_{j=1,\, j\neq i}^{n} \left| y_{ij} \right| \geq 0 \qquad (5.1)$$

y sea

$y'_{ij} = \left| y_{ij} \right| / \left| y_{ii} \right| \geq 0$ el valor normalizado de y_{ij} con respecto a y_{ii}

$z_{ij} = y_{ij}^{'} / D \geq 0$ el valor normalizado de y_{ij}' respecto a la media D.

Se define el peso P_i de una barra B_i como:

$$P_i = \sum_{j=1 \; j \neq i}^{n} (y_{ij}^{'})^{z_{ij}} \qquad \forall i \in \{1,...,n\}, \quad \forall y_{ij} \neq 0$$

(5.2)

Como en la mayoría de las redes eléctricas de gran porte se verifica que la influencia de la resistencia de la línea es mucho menor que la influencia de la inductancia (o capacitancia) de la misma, para mayor practicidad se puede reemplazar el valor del módulo de las distintas impedancias en las fórmulas anteriores por el valor absoluto de sus componentes imaginarias ($Im\{y_{ij}\} = b_{ij}$), por lo tanto, la formula de peso a ser utilizada en este trabajo será:

$$P_i = \sum_{j=1 \; j \neq i}^{n} (b_{ij}^{'})^{z_{ij}} \qquad \forall i \in \{1,...,n\}, \quad \forall b_{ij} \neq 0$$

(5.3)

con

$$b_{ij}^{'} = |b_{ij}| / |b_{ii}| \geq 0$$

$$z_{ij} = b_{ij}^{'} / D \geq 0$$

$$D = \frac{1}{nc} \sum_{i=1}^{n} \sum_{j=1, \, j \neq i}^{n} |b_{ij}| \geq 0$$

La clasificación de las barras es hecha calculando el peso de cada barra y estableciendo un *ranking* de las mismas conforme sus pesos P_i, según el algoritmo mostrado a continuación:

Entrada: matriz Y
DESDE i = 1 hasta n /* para cada una de las n barras.*/ Calcular P_i según la ecuación (5.2); Incluir a P_i en el ranking ordenado de barras; FIN_DESDE
Salida: Pesos P_i de las n barras ordenadas según un ranking de pesos

Figura 5.2: Algoritmo de clasificación de barras

Es importante observar que el valor del peso P_i busca representar si la barra en cuestión posee interligaciones fuertes (valor elevado de $|b_{ij}|$) o interligaciones débiles (valor pequeño de $|b_{ij}|$) con las otras barras de la red. El exponente $z_{ij} = |b_{ij}| / (D \times |b_{ii}|)$ aparece en la expresión para que ramas con elevados valores de $|b_{ij}|$ contribuyan más significativamente al valor del peso, buscando diferenciar barras con varias interligaciones de bajos valores de $|b_{ij}|$ de otras barras con pocas interligaciones pero con elevados valores de $|b_{ij}|$. Note que otras definiciones de pesos pueden ser dadas, pero la utilizada aquí es la que resultó experimentalmente mas conveniente.

5.3.- Etapa 2: Selección de semillas

La elección de las barras que harán el papel de semillas representa una importante etapa en la metodología propuesta. Además de determinar el número de subredes en que se irá a descomponer la red, dependiendo de la elección realizada pueden ser generadas diferentes descomposiciones. Consecuentemente, se obtendrán diferentes resultados en términos de eficiencia computacional del método de solución.

Se verificó en los resultados experimentales que los agrupamientos de barras fuertemente acopladas (elevado valor de $|b_{ij}|$), no deben ser separados en diferentes subredes, con vistas a atender los objetivos anteriormente citados. Así, se optó por seleccionar como semillas aquellas barras que sean centro de agrupamiento de barras y que no están fuertemente acopladas entre sí conforme al algoritmo ilustrado en la figura 5.3.

Para entender dicho algoritmo de selección de semillas, es necesario definir los parámetros: *vlim*, *nagrup* y *nvec*, así como los conjuntos ***K***, ***S***, ***I*** y ***CIA***.

- Valor límite: *vlim*

  Como las "semillas" deben ser centros de aglutinamiento de barras, es de esperar que las de mayor peso (las más fuertemente acopladas al resto de las barras) sean las mejores candidatas a semilla. Para esto, se define un parámetro *vlim* tal que toda barra B_i que satisfaga $P_i \geq vlim$ sea una candidata a semilla. En principio, todas las barras podrían ser evaluadas como candidatas a semillas haciendo *vlim* = 0, pero esto implicaría un alto costo computacional a causa de la gran cantidad de particiones que tendrían que ser evaluadas en la etapa 4. Como consecuencia de esto, se adoptan como *vlim* valores que limiten el número de candidatas a semillas a una cantidad razonable [25]. En el estudio de los sistemas eléctricos paradigmas de la IEEE, se observó una cierta tendencia al agrupamiento en los valores de los pesos de las barras; es decir, existen por lo

general grupos de barras que poseen pesos de valores aproximadamente iguales. Esto ofrece una referencia importante a la hora de decidir sobre el valor del parámetro *vlim*.

- Número de agrupamiento: *nagrup*

  nagrup es un parámetro que indica cuantas barras deben ser agrupadas alrededor de cada candidata a semilla para luego, realizando una sumatoria de sus pesos, decidir cuales serán las *p* semillas a ser utilizadas.

- Radio mínimo de vecindad: *nvec*

  nvec es un parámetro que se utiliza para evitar que barras fuertemente acopladas entre sí sean semillas al mismo tiempo.

El valor de estos tres parámetros puede ser determinado por el usuario según el conocimiento que tenga del sistema en estudio. En todo caso, pueden ser utilizados criterios estadísticos para determinar estos parámetros. Por ejemplo, Vale et al.[25] recomienda los siguientes valores:

vlim: dos valores de peso tales que un gran número de los pesos calculados se encuentre cerca de ellos.

nagrup: 1%,2%,5%,10% y 15% de la relación *n/p* entre el número de barras *n* del sistema y el número de procesadores *p*.

nvec: 1% y 10% de n/p.

En nuestra experiencia con respecto a los conjuntos utilizados por el algoritmo de la figura 5.3, se puede mencionar que dependiendo de la dimensión del sistema a ser descompuesto y de la capacidad del procesador en el cual el método de partición es implementado, se tomaron con éxito valores de *vlim* tales que sean consideradas

como candidatas a semillas hasta 11 barras, con valores de *nagrup* y *nvec* del orden del 8% de la relación *n/p*.

- Conjunto ***K***:

  Es el conjunto de barras candidatas a ser semillas; sus elementos son las barras B_i con un peso $P_i \geq vlim$.

- Conjunto ***S***:

  Es el conjunto de barras que serán semillas de una partición. Al iniciar el algoritmo de selección de semillas el conjunto ***S*** está vacío y al finalizar contiene las *p* barras semillas. Si se desea, se podrá tener tantos conjuntos ***S*** como ternas *vlim*, *nagrup* y *nvec* se tengan.

- Conjunto ***B***:

  Por cada barra B_i candidata a ser semilla existe un conjunto $\boldsymbol{B}_i$, que tiene como primer elemento la misma barra B_i candidata a ser semilla, a la cual se irán anexando *nagrup* barras que definirán su "región de influencia". Esta región de influencia define al conjunto de barras que de forma directa o indirecta (a través de otras barras) están fuertemente acopladas entre sí.

- Conjunto ***CBA***:

  Es el **C**onjunto de **B**arras **A**dyacentes a las barras del conjunto $\boldsymbol{B_i}$. , e indica cuáles barras están disponibles para ser incluidas en este último conjunto.

El principio básico del método de selección de semillas es el siguiente:

1º) Se determina el conjunto de barras que serían buenas candidatas a semillas (conjunto ***K***) individualizando a las barras cuyo valor de peso sea igual o mayor a *vlim*.

2º) A partir de cada una de las barras B_i del conjunto ***K***, e independientemente de las otras barras de este conjunto, se asocian a ella sucesivamente (se incluyen en su conjunto $\boldsymbol{B_i}$) las barras más pesadas de entre las adyacentes a la agrupación (pertenecientes al conjunto $\boldsymbol{CBA_i}$). A medida que tal agrupamiento es hecho, se calcula la sumatoria de los pesos de todas las barras agrupadas en $\boldsymbol{B_i}$. Se pretende con esto identificar las barras de ***K*** que se rodean con las mayores ligaciones. La agrupación continúa hasta que la barra candidata haya agrupado *nagrup* barras.

3º) Una vez calculados los valores de las sumatorias relativas a cada barra de ***K***, las barras semillas son escogidas de entre aquellas que poseen los mayores valores de sumatoria. Para evitar que sean escogidas barras muy próximas, con fuertes ligaciones entre ellas, se impone que una barra semilla no se encuentre entre las *nvec* primeras barras asociadas en el agrupamiento de las semillas previamente seleccionadas.

El pseudo-código del algoritmo implementado para seleccionar las semillas se muestra en la figura 5.3.

5.4.- Etapa 3: Generación de la partición

Una vez determinadas las semillas para cada subsistema de acuerdo al algoritmo de selección de semillas, es necesario adicionar las demás barras a las subredes más adecuadas. Cada una de las *p* subredes que formarán la partición tiene como barra inicial a su semilla, y a ella se irán anexando otras barras, teniendo en cuenta para ello los pesos P_i , los valores de acoplamiento $\left|b_{ij}\right|$ y el hecho de optar por realizar solapamiento parcial o no, en caso de empates.

Un aspecto importante en la formación de la partición es la necesidad de asignar a cada procesador un subproblema de dimensión proporcional a su

performance relativa con vistas a lograr el equilibrio de la carga computacional. Para ello, se definen a continuación los vectores ***w, C y Q***

```
Entradas:   ternas vlim, nagrup y nvec
            matriz de admitancias Y
            pesos de las barras Pi
            número de procesadores p
```

```
PARA cada terna seleccionada
    Inicializar el conjunto K como vacío;        /* Conjunto de barras candidatas a semillas */
    PARA cada una de las barras Bi
        SI peso Pi ≥ vlim ENTONCES
            Incluir la barra Bi en K ;
        FIN_SI
    FIN_PARA
    PARA cada una de las barras Bi en K
        Inicializar un conjunto Bi ; como vacío;        /* Conjunto de barras agrupadas
                                                        alrededor de las candidatas a semillas */
        Incluir la barra Bi en Bi ;
        Inicializar un conjunto CBAi como vacío;        /* Conjunto de Barras
        Adyacentes
                                                        al conjunto B*/
        Incluir en CBAi las barras adyacentes a la barra Bi;
        DESDE 1 hasta nagrup
            Incluir la barra de mayor peso del CBAi en Bi;
            Eliminar esta barra del CBAi;
            Incluir en CBAi las barras adyacentes a la recientemente incorporada en Bi, que no
                                                        /* se encuentren en CBAi ni en B; */
        FIN_DESDE
    FIN_PARA
    PARA cada conjunto B
            Calcular la sumatoria de los pesos de todas sus barras ;
    FIN_PARA
    Seleccionar de K la barra que posea mayor sumatoria de peso calculada a partir de los
                                valores del conjunto B y elegirla como primera semilla;
    Inicializar el conjunto S como vacío;        /* Conjunto de semillas de una partición */
    Incluir esta barra en S;
    Eliminar esta barra de K;
    MIENTRAS el número de semillas ≤ p
        Seleccionar en K la barra que posea la mayor sumatoria de pesos;
        SI la barra no está entre las nvec primeras barras de los conjuntos B de las semillas ya
        seleccionadas ENTONCES
            Elegir esta barra como semilla;
            Incluir esta barra en S;
            Eliminar esta barra de K;
        DE_LO_CONTRARIO
            Eliminar esta barra de K;
        FIN_SI
    FIN_MIENTRAS
FIN_PARA
```

Salida: Para cada terna se tendrá un conjunto ***S*** de *p* semillas

Figura 5.3: Algoritmo de selección de semillas

- *Performance Relativa entre Procesadores*: ***w***

  Se define $\boldsymbol{w} \in \mathrm{N}^{\mathrm{p}}$ como el vector de *Performance Relativa entre Procesadores*. Dicha performance relativa es la relación que existe entre las capacidades de procesamiento de los procesadores. Por ejemplo, $\boldsymbol{w} = \begin{bmatrix} 1 & 2 \end{bmatrix}^{\mathrm{T}}$ implica que el procesador 2 puede procesar el doble de ecuaciones que el procesador 1 en el mismo periodo de tiempo.

  En este trabajo se pretende que, dados el sistema eléctrico y un sistema distribuido con performance relativa ***w*** conocida, hallar una partición que resuelva el problema en estudio de forma eficiente.

- *Balanceamiento de carga*: ***C***

  Se define al vector Balanceamiento de carga $\boldsymbol{C} \in N^{p}$ como un vector de estado de la forma $\boldsymbol{C} = \begin{bmatrix} c_1 & \cdots & c_p \end{bmatrix}^{T}$, donde los c_i representan la cantidad de barras agrupadas en el subconjunto $\boldsymbol{B}_i$ (definido en la sección anterior) en un momento dado. Al finalizar el proceso de partición, el vector ***C*** indicará cuantas barras serán asignadas a cada procesador. El método buscará que los vectores ***w*** y ***C*** sean paralelos; esto es, se encuentren en siguiente relación:

  $$\boldsymbol{C} = \alpha \boldsymbol{w}$$

  donde $\alpha \geq 0$ es una constante escalar.

- *Cupo Parcial*: ***Q***

  Se define al vector Cupo Parcial $\boldsymbol{Q} \in \Re^{p}$ como un vector de estado de la forma $\boldsymbol{Q} = \begin{bmatrix} q_1 & \cdots & q_p \end{bmatrix}^{T}$, donde los q_i se obtienen de acuerdo a la relación $q_i = c_i/w_i$. Idealmente, al terminar la computación tendría que verificarse que

$$\|\boldsymbol{Q}\|_{\infty} = q_i = \alpha \qquad \forall i \in \{1,\ldots,p\}$$

(5.1)

aunque esto puede no verificarse por no ser los valores de las componentes del vector $\boldsymbol{w}$ divisores exactos del número de barras del sistema a descomponer.

- *Límite de Solapamiento Parcial*: *Lim_over*

  Se define *Lim_over* como el mínimo valor requerido del peso de una barra para realizar solapamiento parcial. Si el peso P_i de un una barra es mayor o igual que *Lim_over,* entonces se realiza el solapamiento parcial de esa barra en caso de empate. Este valor puede ser determinado empíricamente utilizando valores estadísticos. Por ejemplo, una opción apropiada para este parámetro sería tomar los valores de *vlim* que se han utilizado, al ser estos referencias válidas de la posición relativa de una barra con respecto a las demás en el ranking de pesos.

En el proceso de generación de las particiones, cada semilla selecciona entre sus adyacentes a la barra que posea mayor peso y que a su vez no sea parte de otro subsistema. Si no existen coincidencias entre las candidatas de las distintas semillas, dichas barras candidatas son asociadas a las semillas que la escogieron, constituyéndose así en subredes en formación. El proceso continúa de forma análoga: cada subred selecciona la barra más pesada y aun no agrupada de entre sus adyacentes. Si no existen coincidencias de candidatas, se procede a la inclusión de dichas candidatas en las subredes correspondientes. Debe tenerse en cuenta que, para obtener el balanceo de carga deseado, solamente seleccionarán candidatas aquellas subredes cuyo valor correspondiente de q_i sea menor a la norma infinita del vector $\boldsymbol{Q}$. Con esto se busca que la partición vaya siendo generada con el desbalanceamiento requerido.

En el caso de existir coincidencias entre candidatas, se analiza el peso de la barra en disputa. Si el mismo es menor que el valor requerido (*Lim_over*) para realizar solapamiento, se asigna la barra a aquella subred con la que tenga una ligación más fuerte. En caso que el peso de la barra sea mayor que el limite de solapamiento, se asigna dicha barra a todas las subredes que la reclamen como candidata.

En la figura 5.4 se presenta el algoritmo de formación de particiones.

5.5.- Etapa 4: Evaluación de particiones y selección

Hay diversas formas de generar descomposiciones de un sistema eléctrico, sea en forma intuitiva, distribución geográfica o utilizando algún criterio automático. En esta sección se presentará un *criterio selectivo de descomposiciones*, que tiene como objetivo indicar, entre un conjunto dado de decomposiciones del sistema, aquellas que presenten las mejores características en términos del desempeño computacional de los métodos de resolución bloque-iterativos.

En efecto, si se utilizan distintas ternas de parámetros *vlim*, *nagrup* y *nvec* pueden ser determinados varios conjuntos diferentes de semillas que a su vez servirán de pie para generar diferentes particiones. Debido a la necesidad de poseer una referencia previa sobre la calidad de la partición de forma a evitar la necesidad de resolver cada sistema para conocer su desempeño, es sumamente útil disponer de un parámetro que refleje la calidad de la partición.

En [32] se proponen varios parámetros posibles de selección, pudiéndose citar entre estos a:

- El número total de interligaciones que unen las distintas subredes en que la red global es descompuesta.
- El máximo de los números de interligaciones entre dos subredes cualesquiera.
- La sumatoria total del módulo de las admitancias de las ramas que son seccionadas por la partición.

Entradas: Conjuntos ***S*** de semillas $\{s_1,...,s_p\}$
Lim_over
matriz de admitancias Y
Pesos de las barras

Para cada grupo ***S***
Inicializar los vectores ***C*** y ***Q*** ;
PARA cada semillas $s_i \_ S$
Inicializar un conjunto ***B*** como vacío;
Incluir en B_i la semilla s_i;
Inicializar un conjunto CBA_i como vacío;
Incluir en CBA_i las barras adyacentes a la semilla s_i;
FIN_PARA
Calcular los vectores ***C*** y ***Q***
MIENTRAS existan barras no agrupadas
PARA todos los conjuntos B_i tales que $Q_i < \| Q \|_\infty$ ó $(Q_i = Q_j \ \forall i, j \in \{1,\ldots,p\})$;
Seleccionar como barra candidata a ser incluida a la de mayor peso en el ***CBA*** correspondiente;
SI no existen coincidencias en las barras candidatas a ser incluidas
Incluir la barra del primer conjunto ***B*** ;
Eliminar esta barra de todos los ***CBA***s;
FIN_SI
SI existen coincidencias en las barras candidatas
Si no se tiene opción de solapamiento /* si el usuario no desea solapamiento parcial*/
Seleccionar el valor del mayor acoplamiento entre las barras ya incluidas en B_i y la barra candidata a ser incluida;
SI no existen coincidencias de valor de mayor acoplamiento
Incluir la barra candidata en el conjunto ***B*** con el cual posea mayor acoplamiento
FIN_SI
SI existen coincidencias de valor de mayor acoplamiento
Incluir la barra candidata en el primer conjunto ***B*** que pelea por ella;
FIN_SI
FIN_SI
Si se tiene opción de solapamiento
SI P_i de la barra *i* es $\geq$ *Lim_over*
Incluir la barra en todos los subconjuntos ***B*** coincidentes;
FIN_SI
SI P_i de la barra *i* es $<$ *Lim_over*
Incluir la barra candidata en el primer conjunto ***B*** que pelea por ella;
FIN_SI
FIN_SI
FIN_SI
Eliminar la barra incluida de todos los ***CBA***s;
FIN_PARA
Incluir en los ***CBA***s correspondientes las barras adyacentes a las incluidas;
Actualizar ***C*** y ϕ ;
FIN_MIENTRAS

Salida: partición del sistema

Figura 5.4: Algoritmo de partición

- La mayor diferencia entre los números de ramas internas de dos subredes cualquiera.
- La menor sumatoria de valores absolutos de las admitancias de las ramas internas a las subredes.

En el marco de dicho trabajo, fue sugerido como el mejor parámetro a la sumatoria de todos los valores absolutos de las ligaciones entre barras separadas por la partición, el cual llamaremos Parámetro de Acoplamiento “Par_A”; esto es,

Par_A=Σ $|y_{ij}|$ para todo par de barras (B_i, B_j) que se encuentren en subredes diferentes.

Conforme a lo expuesto en el Capítulo 2 sección 2.3, el radio espectral $\rho(\mathbf{H})$ de la matriz de comparación sería un buen parámetro de selección por el hecho de proporcionar información sobre la tasa de convergencia del algoritmo de resolución, y en consecuencia, sobre el desempeño computacional de la partición analizada. Por lo tanto, en este trabajo se propone como parámetro de selección de descomposiciones al radio espectral de la matriz de comparación **H**, siendo elegida como óptima aquella partición que presente el menor valor de $\rho(\mathbf{H})$. La efectividad de este parámetro se analizará en los resultados experimentales presentados en el Capítulo 7.

Conviene acotar aquí que el radio espectral de la matriz de comparación **H** es un límite superior (*Upper Bound*) del radio espectral de la matriz de iteración, que en el caso lineal es, rigurosamente hablando, el mejor parámetro para indicar con seguridad si el algoritmo de resolución convergerá a la solución o no. Sin embargo y a pesar de sus ventajas, el cálculo de este último parámetro reviste una dificultad computacional mayor que la de resolver el sistema eléctrico en su totalidad, por lo

que no tiene sentido su utilización. En efecto, al tener la matriz de iteración dimensión $n_x n$, el cálculo de su radio espectral es exponencialmente más complejo que el cálculo del radio espectral de la matriz de comparación, cuya dimensión es solo de $p_x p$.

Para consolidar el método presentado, el siguiente capítulo presenta un ejemplo ilustrativo.

Entrada: las diversas particiones generadas por el algoritmo de particiones
PARA cada partición Calcular ρ(**H**) Incluir el valor de ρ(**H**) en el ranking Elegir como mejor partición la de menor valor de ρ(**H**).
Salidas: la mejor partición generada por el método propuesto, ranking de particiones.

Figura 5.5: Algoritmo de selección de particiones

CAPITULO 6: UN EJEMPLO ILUSTRATIVO

Se presenta en este capítulo una aplicación sencilla de la metodología propuesta. El ejemplo a ser utilizado fue especialmente concebido para permitir una explicación bien práctica del proceso de partición propuesto en el Capítulo 5.

6.1.- Presentación del problema

Considérese un pequeño sistema eléctrico constituido por 8 barras. Se buscará descomponer la red en 2 subredes lo más débilmente acopladas posible y con las dimensiones relativas deseadas. El sistema distribuido a ser utilizado consistirá en dos procesadores con performance relativa

$$w = [2,\ 1]$$

esto es, el 1er procesador es “el doble” de rápido que el segundo.

La matriz de admitancias del sistema eléctrico considerado como ejemplo de aplicación del método propuesto será:

$$\mathbf{Y} = \begin{bmatrix} 0.05 - j0.5 & -0.01 + j0.1 & 0 & -0.02 + j0.2 & -0.02 + j0.2 & 0 & 0 & 0 \\ -0.01 + j0.1 & .05 - j0.5 & -0.01 + j0.1 & 0 & 0 & 0 & 0 & -0.03 + j0.3 \\ 0 & -0.01 + j0.1 & 0.02 - j0.2 & 0 & 0 & 0 & 0 & -0.01 + j0.1 \\ -0.02 + j0.2 & 0 & 0 & 0.04 - j0.4 & -0.02 + j0.2 & 0 & 0 & 0 \\ -0.02 + j0.2 & 0 & 0 & -0.02 + j0.2 & 0.065 - j0.65 & -0.015 + j0.15 & -0.01 + j0.1 & 0 \\ 0 & 0 & 0 & 0 & -0.015 + j0.15 & 0.025 - j0.25 & -0.01 + j0.1 & 0 \\ 0 & 0 & 0 & 0 & -0.01 + j0.1 & -0.01 + j0.1 & 0.04 - j0.4 & -0.02 + j0.2 \\ 0 & -0.03 + j0.3 & -0.01 + j0.1 & 0 & 0 & 0 & -0.02 + j0.2 & 0.05 - j0.5 \end{bmatrix}$$

Esta matriz, dada en p.u, no representa necesariamente un sistema eléctrico existente y se lo usa solo a modo de ejemplo para ofrecer la mayor cantidad de

situaciones peculiares posibles, de forma tal a realizar una explicación lo más completa posible.

Como fuera mencionado, la matriz **Y** tiene elementos complejos, es muy esparza y simétrica. Además, los valores absolutos de las componentes imaginarias de los elementos y_{ij} son mucho mayores que los de las componentes reales. En consecuencia, en el marco de este trabajo al grado de acoplamiento entre las barras se lo representará por el módulo de las susceptancias ($|b_{ij}|$) y no por el módulo de las admitancias ($|y_{ij}|$).

El grafo que se muestra a continuación representa al sistema eléctrico del ejemplo. Los nodos del grafo representan las barras, los lazos representan las ramas de la red, y los números adjuntos a las ramas muestran el valor del acoplamiento ($|b_{ij}|$) entre barras. Por ejemplo, en la matriz **Y** podemos notar que las componentes imaginarias de los elementos y_{56} e y_{65} tienen como valor absoluto 0.15 p.u. En el grafo, encontramos dicho valor sobre la rama que une los nodos B_5 y B_6. Cualquier partición que separe a las barras B_5 y B_6 en subredes diferentes, necesitará necesariamente "*cortar*" ese acoplamiento.

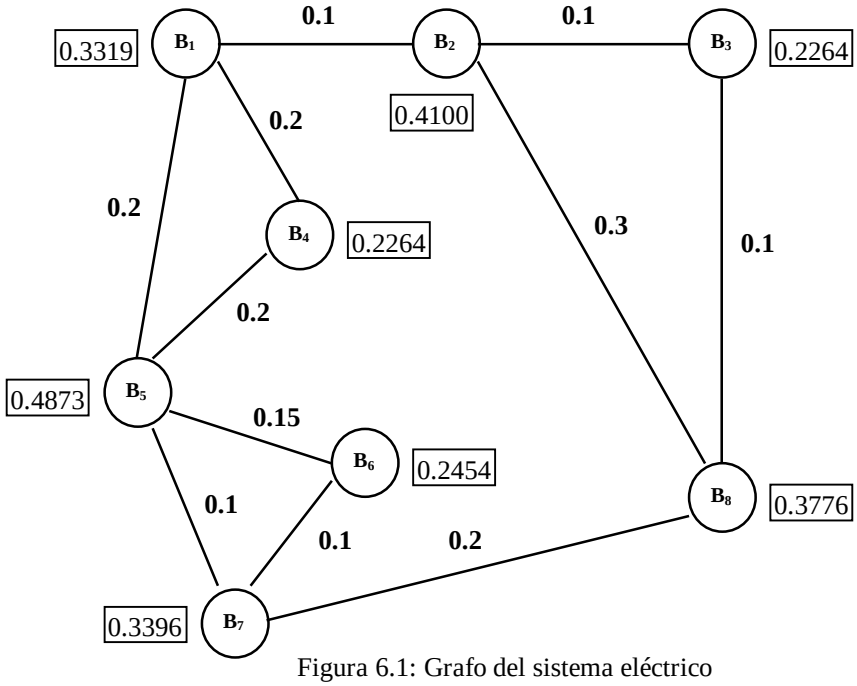

Figura 6.1: Grafo del sistema eléctrico

6.2.- Clasificación de las barras

Para el Sistema dado se calcularon los pesos de las 8 barras conforme a la ecuación (5.3). Los valores de los pesos se encuentran en los recuadros que aparecen junto a cada nodo del grafo, en la figura 6.1.

Con base en los valores calculados de pesos, se puede realizar ahora la clasificación de las barras en función a sus pesos. Vemos así, en la tabla 6.1 y la figura 6.2, las barras ordenadas de mayor a menor peso.

Barra	Peso
B_5	0.4873
B_2	0.4100
B_8	0.3776
B_7	0.3396
B_1	0.3319
B_6	0.2454
B_3	0.2264
B_4	0.2264

Tabla 6.1 : Clasificación de las barras

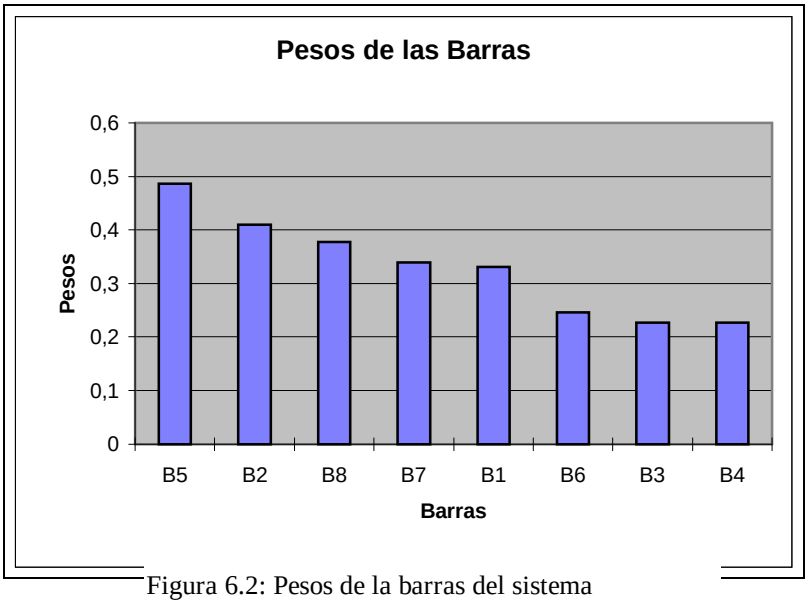

Figura 6.2: Pesos de la barras del sistema

6.3.- Selección de semillas

Combinando distintos valores de *vlim*, *nagrup* y *nvec* podrán ser seleccionados varios conjuntos diferentes de semillas. A continuación se determinarán los valores a ser utilizados, siguiendo criterios que fueron encontrados válidos para facilitar el entendimiento del lector. Otros criterios más prácticos ya fueron presentados en la sección 5.3.

- ***vlim***: Observando la Figura 6.2, se identifican dos agrupaciones bien diferenciadas de peso, separadas por la ordenada 0.3. Por lo tanto, este último valor puede ser una opción valida para el parámetro *vlim*. Note que tomando valores menores, la gran mayoría de las barras, si no la totalidad, serían seleccionadas como candidatas a semillas, lo que aumentaría de manera significativa el trabajo computacional a ser realizado. Por otro lado, si se toman valores de *vlim* muy elevados, se corre el riesgo de disponer de insuficientes barras candidatas como para poder realizar una selección adecuada.
- ***nagrup*:** Debido a la pequeña dimensión del problema, no sería práctico agrupar alrededor de cada barra candidata a semilla un número muy elevado de barras, pues ello resultaría en agrupamientos semejantes, impidiendo así obtener el número requerido de semillas. Se agruparán en este caso solo 2 barras además de la barra candidata, esto es, *nagrup=2*.
- ***nvec*:** Para evitar que dos semillas formen parte del mismo agrupamiento de barras, el parámetro *nvec* deberá asegurar un distanciamiento razonable entre ellas. En este caso, y debido nuevamente a la pequeña dimensión del problema, *nvec* =1 proporcionaría un *radio de vecindad* mínimo aceptable.

Con base en el valor de *vlim* adoptado, se puede determinar ahora los elementos del conjunto ***K***, que contendrá a las barras candidatas a semillas. Se incluirán en él a cada barra cuyo peso sea mayor o igual a 0.3. Tenemos así:

$$\boldsymbol{K} = \{ B_5, B_2, B_8 \}$$

A continuación se realizará el proceso de agrupamiento de barras alrededor de cada una de las 3 candidatas. Para el efecto, la tabla 6.2 muestra como ejemplo el proceso para la barra B_5. La primera columna de dicha tabla (***B***) contendrá a las barras que forman parte del agrupamiento, mientras que la segunda columna (***CBA***) esta destinada a contener a todas las barras adyacentes al agrupamiento en formación. Se puede ver que inicialmente en la primera columna solo se encuentra la barra candidata (en este caso B_5) y en la segunda sus barras adyacentes, conforme a la figura 5.1.

B	***CBA***
B_5	**B_1,B_4,B_6,B_7**

Tabla 6.2: Cuadro de agrupamiento de barras

Para continuar con el proceso, se selecciona seguidamente la barra de mayor peso de entre las adyacentes a B_5 y se la incluye entre las barras del agrupamiento en formación. En consecuencia, la barra B_7 pasa a integrar la columna ***B*** conforme se muestra en la tabla 6.3. Nótese que en la columna ***CBA*** será eliminada B_7 por no encontrarse ya disponible para un futuro anexamiento, mientras que las barras adyacentes a B_7 y que aun no se encontraban en ***CBA*** son incluidas ahora en dicha columna. La barra eliminada de ***CBA***, por haber pasado a la columna ***B***, está señalada por una tachadura en la tabla 6.3.

B	***CBA***
$\mathbf{B_5}$	$\mathbf{B_1,B_4,B_6,\cancel{B_7}}$
$\mathbf{B_7}$	$\mathbf{B_8}$

Tabla 6.3: Cuadro de agrupamiento de barras

El proceso anterior continúa hasta que *nagrup* barras hayan sido asociadas a la barra candidata a semilla en la columna ***B***. La siguiente y última barra a ser incluida en el agrupamiento es la barra B_8, ya que *nagrup*=2 y B_8 es la más pesada de entre las adyacentes a las barras B_5 y B_7. Ya no es necesario aquí incluir a las nuevas barras adyacentes en ***CBA***, ya que no se seleccionarán nuevas barras. La tabla quedará entonces como se muestra a continuación:

nagrup=2

B	***CBA***
$\mathbf{B_5}$	$\mathbf{B_1,B_4,B_6,\cancel{B_7}}$
$\mathbf{B_7}$	$\mathbf{\cancel{B_8}}$
$\mathbf{B_8}$	

Tabla 6.4: Cuadro de agrupamiento de barras

Este mismo procedimiento se aplica a cada una de las barras del conjunto ***K***.

Se muestra a continuación la tabla obtenida una vez finalizado este proceso. Se han agregado columnas donde se colocaron los pesos de las barras agrupadas, así como casillas donde se encuentran las sumatorias de los pesos de las barras agrupadas alrededor de cada candidata a semilla.

B_I	CBA_I	Pesos	B_{II}	CBA_{II}	Pesos	B_{III}	CBA_{III}	Pesos
B_5	B_1,B_4,B_6,~~B_7~~	0.4873	B_2	B_1,B_3,~~B_8~~	0.4100	B_8	B_3,~~B_7~~,~~B_2~~	0.3776
B_7	~~B_8~~	0.3396	B_8	~~B_7~~	0.3776	B_2	B_1	0.4100
B_8		0.3776	B_7		0.3396	B_7		0.3396
	$\sum$Pesos=	1.2045		$\sum$Pesos=	1.1272		$\sum$Pesos=	1.1272

Tabla 6.5: Cuadros de agrupamiento de barras

La primera barra a ser seleccionada como semilla debe ser aquella cuyo agrupamiento posea la mayor sumatoria de pesos. Sin embargo, se observa en la tabla 6.5 que los agrupamientos de las barras B_2 y B_8 tienen igual sumatoria de pesos, por lo que el peso de cada barra candidata es usado como criterio de desempate. En este ejemplo se seleccionará a la barra B_2 como primera semilla por tener mayor peso que la barra B_8. Así, la barra B_2 es incluida en el conjunto de semillas ***S***.

Como las condiciones del problema especificaban que éste debe ser particionado en dos subredes, se necesitarán por lo tanto dos semillas. La segunda semilla deberá ser aquella candidata (aún no seleccionada) cuya sumatoria de pesos sea la mayor. En este caso, la barra B_8 es la de mayor sumatoria, pero no puede ser la segunda semilla porque no cumple con la condición de radio de vecindad mínimo determinada por el parámetro *nvec*, que estipula que una semilla no debe encontrarse entre las *nvec* primeras barras asociadas del agrupamiento de barras de las semillas previamente seleccionadas. En el agrupamiento de la semilla B_2 previamente escogida, se encuentra a la barra B_8 como primera asociada, lo que la imposibilita de ser seleccionada como segunda semilla. La candidata restante, B_5, no se encuentra siquiera en el agrupamiento de B_2, por lo que se la selecciona como segunda semilla incluyéndola en el conjunto ***S***. De este modo, ya se dispone de un conjunto de semillas a partir del cual generar la descomposición de la red, esto es

$$\boldsymbol{S} = \{ B_5 , B_2 \}$$

6.4.- Generación de la partición

El algoritmo de generación de la partición posee alguna similitud con el algoritmo de selección de semillas. La diferencia principal consiste en que en la selección de semillas, la asociación de barras se lleva a cabo en forma independiente, sin existir interacción entre los distintos agrupamientos, mientras que en la generación de la partición todos los agrupamientos alrededor de las distintas semillas se realizan en forma simultánea, interactuando entre sí.

La Tabla 6.6 que se muestra a continuación es bastante similar a las utilizadas en la sección 6.3. La columna ***B***, que en el proceso de selección de semillas contenía al agrupamiento de barras, ahora contendrá a la subred eléctrica en formación. Se llamará subred 1 a la formada en torno a la semilla B_2, y subred 2 a la correspondiente a la semilla B_5. Debe observarse que en las columnas ***CBA*** que contienen a las barras adyacentes a la subred en formación no se encuentran las demás semillas, por más que estas se encuentren entre las adyacentes. Esto se debe a que en la citada columna solo se indicarán las barras adyacentes y disponibles de ser asociadas, es decir, las barras aun no asociadas por ninguna subred.

Subred 1		**Subred 2**	
$\boldsymbol{B_I}$	$\boldsymbol{CBA_I}$	$\boldsymbol{B_{II}}$	$\boldsymbol{CBA_{II}}$
$\mathbf{B_2}$	$\mathbf{B_1, B_3, B_8}$	$\mathbf{B_5}$	$\mathbf{B_1, B_4, B_6, B_7}$

Tabla 6.6: Cuadro de generación de la partición

Se impone aquí el cálculo del vector ***Q*** que se utilizará para determinar cuáles subredes en formación anexarán barras en un instante dado. De acuerdo a lo expuesto en la sección 5.4, las componentes del vector ***Q*** se calculan dividiendo las componentes del vector ***C*** (cantidad de barras agrupadas en la subred en un momento dado) por las componentes del vector ***w*** (vector de *performances* relativas). El vector ***Q*** será entonces:

$$\mathbf{Q} = [q_1, q_2] = \left[\frac{c_1}{w_1}, \frac{c_2}{w_2}\right] = \left[\frac{1}{2}, \frac{1}{1}\right] = [0.5, 1]$$

Teniendo en cuenta las dimensiones que se desean para las subredes, lo ideal sería que todas las componentes del vector ***Q*** sean iguales entre sí. Se ha detectado así que la subred 1 se encuentra en desventaja con respecto a la otra en lo que se refiere a la relación que debe existir entre sus dimensiones. En consecuencia, solamente la subred 1 seleccionará la barra mas pesada de entre sus adyacentes a fin de anexarla, en este caso la barra B_8. Como no existe coincidencia de candidatas con la subred 2, por no haber esta seleccionado a ninguna, B_8 pasa a formar parte automáticamente de la subred 1. La nueva situación puede verse en la tabla 6.7, que muestra a la subred 1 con 2 barras (B_2 y B_8) y a la subred 2 con 1 barra (B_5).

Subred 1		**Subred 2**	
$\boldsymbol{B_I}$	$\boldsymbol{CBA_I}$	$\boldsymbol{B_{II}}$	$\boldsymbol{CBA_{II}}$
$\mathbf{B_2}$	$\mathbf{B_1, B_3, \cancel{B_8}}$	$\mathbf{B_5}$	$\mathbf{B_1, B_4, B_6, \cancel{B_7}}$
$\mathbf{B_8}$	$\mathbf{B_7}$		

Tabla 6.7: Cuadro de generación de la partición

La nueva barra B_8 de la subred 2 ha sido eliminada de todos los ***CBA*** que la contaban como adyacente disponible a ser incluida. Se observa también que en

CBA₁ se ha incluido la barra B_7, que es la nueva barra adyacente a la subred 1 a través de la barra B_8 (por ser $b_{78} \neq 0$).

El vector ***Q*** será ahora:

$$\mathbf{Q} = [q_1, q_2] - \left[\frac{c_1}{w_1}, \frac{c_2}{w_2}\right] - \left[\frac{2}{2}, \frac{1}{1}\right] = [1, 1]$$

Dado que ambas subredes se encuentran con las dimensiones relativas deseadas, cada una de ellas procede a seleccionar a la barra más pesada de su entorno. En este caso, se verifica una coincidencia de candidatas sobre la barra B_7. Se impone aquí por lo tanto una decisión sobre el hecho de realizar o no solapamiento parcial, que consiste en que una misma barra sea asignada a dos o más procesadores para su resolución. Si no se desea realizar solapamiento parcial, se asigna la barra a aquella subred con la cual posea el mayor acoplamiento.

Observando el grafo del sistema, se verifica que B_7 se encuentra relacionada a la subred 1 por una acoplamiento de valor 0.2, mientras que una rama de valor 0.1 la relaciona con la subred 2. En consecuencia, la barra B_7 pasará a formar parte de la subred 1, como se muestra en la tabla 6.8:

Subred 1		**Subred 2**	
$\boldsymbol{B_I}$	$\boldsymbol{CBA_I}$	$\boldsymbol{B_{II}}$	$\boldsymbol{CBA_{II}}$
$\mathbf{B_2}$	$\mathbf{B_1, B_3, \cancel{B_8}}$	$\mathbf{B_5}$	$\mathbf{B_1, B_4, B_6, \cancel{B_7}}$
$\mathbf{B_8}$	$\mathbf{\cancel{B_7}}$		
$\mathbf{B_7}$	$\mathbf{B_6}$		

Tabla 6.8: Cuadro de generación de la partición

Si se desea realizar solapamiento parcial, el peso de la barra B_7 se analiza de manera a verificar si se encuentra por encima del valor de peso mínimo requerido

para el efecto. Siendo $P_7 = 0.33 > Lim\_over = 0.3$, se realizará solapamiento por medio de la asignación de la barra en cuestión a ambas subredes (ver la tabla 6.9)

Subred 1		**Subred 2**	
$\boldsymbol{B_I}$	$\boldsymbol{CBA_I}$	$\boldsymbol{B_{II}}$	$\boldsymbol{CBA_{II}}$
$\mathbf{B_2}$	$\mathbf{B_1, B_3, \cancel{B_8}}$	$\mathbf{B_5}$	$\mathbf{B_1, B_4, B_6, \cancel{B_7}}$
$\mathbf{B_8}$	$\mathbf{\cancel{B_7}}$	$\mathbf{B_7}$	
$\mathbf{B_7}$	$\mathbf{B_6}$		

Tabla 6.9: Cuadro de generación de la partición

El proceso continúa de esta manera hasta que no existan barras aún no asociadas a alguna subred, quedando por lo tanto la subred global efectivamente particionada.

Para el caso en el cual no se realiza solapamiento parcial, el problema ejemplo quedará particionado como se muestra en la tabla 6.10 y en la figura 6.3:

Subred 1		**Subred 2**	
$\boldsymbol{B_I}$	$\boldsymbol{CBA_I}$	$\boldsymbol{B_{II}}$	$\boldsymbol{CBA_{II}}$
$\mathbf{B_2}$	$\mathbf{\cancel{B_1}, \cancel{B_3}, \cancel{B_8}}$	$\mathbf{B_5}$	$\mathbf{\cancel{B_1}, \cancel{B_4}, \cancel{B_6}, \cancel{B_7}}$
$\mathbf{B_8}$	$\mathbf{\cancel{B_7}}$	$\mathbf{B_1}$	
$\mathbf{B_7}$	$\mathbf{\cancel{B_6}}$	$\mathbf{B_4}$	
$\mathbf{B_6}$			
$\mathbf{B_3}$			

Tabla 6.10: Partición obtenida sin solapamiento parcial

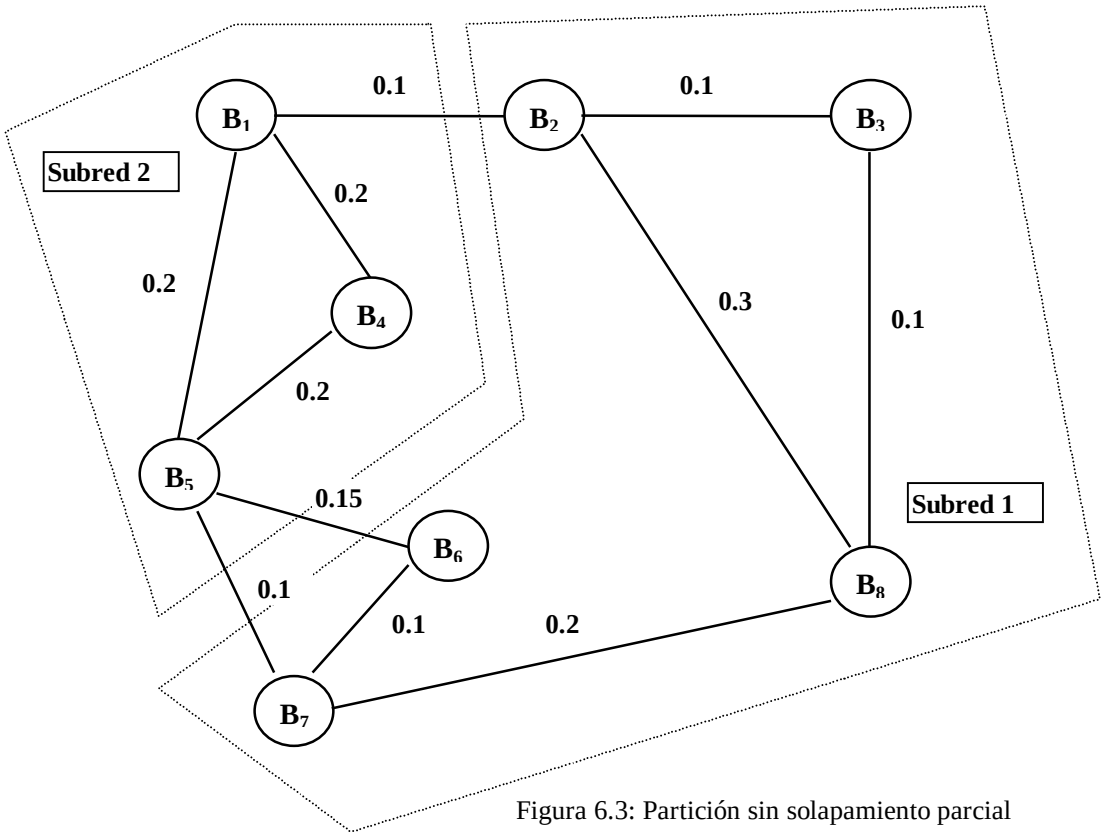

Figura 6.3: Partición sin solapamiento parcial

Si se realiza solapamiento, la partición quedará como se muestra en la tabla 6.11 y figura 6.4 que siguen.

Subred 1		**Subred 2**	
B_I	**CBA_I**	**B_{II}**	**CBA_{II}**
B_2	**~~B_1~~, ~~B_3~~, ~~B_8~~**	**B_5**	**~~B_1~~, ~~B_4~~, ~~B_6~~, ~~B_7~~**
B_8	**~~B_7~~**	**B_7**	
B_7	**~~B_6~~**	**B_6**	
B_1	**~~B_4~~**		
B_3			
B_4			

Tabla 6.11: Partición obtenida con solapamiento parcial

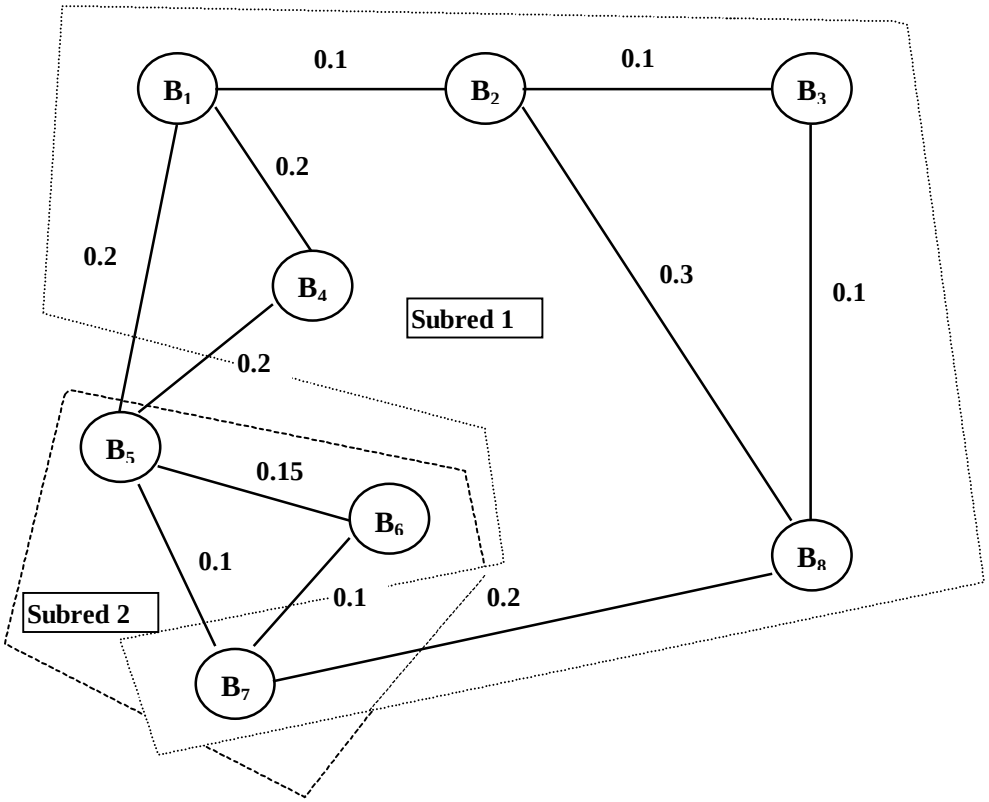

Figura 6.4: Partición con solapamiento parcial

6.5.- Evaluación de las descomposiciones

El parámetro de selección propuesto en este trabajo es el *radio espectral* de la matriz de comparación. Esta matriz, de singular importancia en el estudio de los métodos bloque iterativos, se forma en función del método utilizado en la resolución del problema. Aún cuando el método utilizado en el marco de este trabajo para resolver sistemas eléctricos es una variante del método de Newton-Raphson, por simplicidad en la explicación de este ejemplo se supondrá aquí que el sistema se resolverá aplicando el método de Jacobi [32] conforme a la ecuación

YV _ I

(6.1)

Como se había mostrado en la sección 2.1 , el algoritmo iterativo de Jacobi para la resolución de sistemas lineales de ecuaciones puede ser expresado como:

$$\mathbf{V}(k+1) = \mathbf{S}^{-1}\mathbf{T}\,\mathbf{V}(k) = \begin{bmatrix} \mathbf{S}_{11} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{S}_{pp} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{T}_{11} & \cdots & \mathbf{T}_{1p} \\ \vdots & \ddots & \vdots \\ \mathbf{T}_{p1} & \cdots & \mathbf{T}_{pp} \end{bmatrix} \mathbf{V}(k)$$

(6.2)

donde

$$\mathbf{Y} = \mathbf{S}\ \ \mathbf{T}$$

(6.3)

Los elementos de la matriz de comparación **H** se calculan tomando normas de los bloques de la matriz de iteración $\mathbf{S}^{-1}\mathbf{T}$. Para el ejemplo considerado, y teniendo en cuenta la partición obtenida sin realizar solapamiento, se tendrá entonces que

$$\mathbf{H} = \begin{bmatrix} \left\|\mathbf{S}_{11}^{1}\mathbf{T}_{11}\right\| & \left\|\mathbf{S}_{11}^{1}\mathbf{T}_{12}\right\| \\ \left\|\mathbf{S}_{22}^{1}\mathbf{T}_{21}\right\| & \left\|\mathbf{S}_{22}^{1}\mathbf{T}_{22}\right\| \end{bmatrix}, \qquad \mathbf{H} \in \mathbb{R}^{2\times 2}$$

(6.4)

y considerando para mayor simplicidad que $\mathbf{T}_{11}=\mathbf{T}_{22}=\mathbf{0}$

$$\mathbf{H} = \begin{bmatrix} 0 & \left\|\mathbf{S}_{11}^{1}\mathbf{T}_{12}\right\| \\ \left\|\mathbf{S}_{22}^{1}\mathbf{T}_{21}\right\| & 0 \end{bmatrix} = \begin{bmatrix} 0 & h_{12} \\ h_{21} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2.49 \\ 2.1 & 0 \end{bmatrix}$$

(6.5)

entonces

$$\rho(\mathbf{H}) = \sqrt{h_{12}\cdot h_{21}} = 2.2$$

(6.6)

Así, el cálculo del radio espectral de la matriz **H** se redujo en este caso a calcular la raíz cuadrada del producto de los elementos no diagonales de la matriz.

El cálculo arriba ejemplificado se debería repetir con cada una de las particiones que se desea comparar, de forma a obtener el radio espectral $\rho(\mathbf{H})$ para cada una de las particiones obtenidas. De este modo, se considerará como mejor partición aquella que posea el menor valor del radio espectral de la matriz de comparación, de entre todas las generadas por el método partiendo de distintos grupos de semillas y considerando o no la posibilidad de solapamiento parcial.

CAPITULO 7: ESTUDIOS EXPERIMENTALES

En este capítulo serán presentados los estudios experimentales realizados en base a problemas tipo de la IEEE (*The Institute of Electrical and Electronical Engineers*). Para esto, se resolvió el problema del Flujo de Potencia eléctrica para los referidos paradigmas de la IEEE.

Se analiza aquí el comportamiento de las particiones generadas por el método propuesto en este trabajo, a la vez de realizar un estudio comparativo con respecto a las particiones generadas por otros métodos y con otros criterios.

7.1.- Ambiente computacional

El sistema distribuido utilizado consistió en una red Ethernet a 10 Mbps constituida por las siguientes estaciones de trabajo:

- Una workstation DEC 3000 modelo 300 con procesador ALPHA de 150 MHz, con 32 MB de memoria RAM y operando bajo el sistema operativo OSF/1 Versión 2.0.
- Una workstation SUN SPARC Station 5 con procesador SUN de 66 MHz y memoria RAM de 32 MB, operando bajo el sistema operativo Solaris 5.3.
- 10 Computadoras Personales PREMIO con procesadores Pentium de 75 MHz, 8 MB de memoria RAM y operando bajo el sistema operativo LINUX. Las máquinas fueron nombradas como: Linux1, Linux2,...,Linux10.

El programa generador de particiones (ver Apéndice 1) fue inicialmente codificado en Lenguaje ANSI C. Una versión mejorada fue elaborada en un ambiente Windows utilizando el software Borland C++ Versión 4, que dispone de recursos gráficos que mejoran la interfaz hombre-máquina. Los programas de

resolución del Flujo de Potencia eléctrica fueron implementados en lenguaje ANSI C (ver Apéndice 2), con la biblioteca PVM (Parallel Virtual Machine) Versión 3.10 en su extensión para lenguaje C.

7.2.- Problemas de prueba

Para el levantamiento de los datos experimentales se utilizaron dos sistemas eléctricos tipos, proveídos por la IEEE: el sistema IEEE de 14 barras (IEEE-14) y el sistema IEEE de 118 barras (IEEE-118). El primero de ellos fue seleccionado con vistas a poder realizar un estudio exhaustivo de manera a evaluar todas las posibles formas en que el sistema podría ser particionado. Dicho estudio sería imposible para sistemas de mayor dimensión debido a que el número de posibles particiones del sistema crece en forma factorial con la dimensión del problema.

7.3.- Resolución del Sistema IEEE-14

La figura que se muestra a continuación representa al Sistema Eléctrico IEEE-14. Se indica a la barra slack, la cual no será incluida en ninguna subred dado que su tensión es conocida para la resolución del problema del Flujo de Potencia.

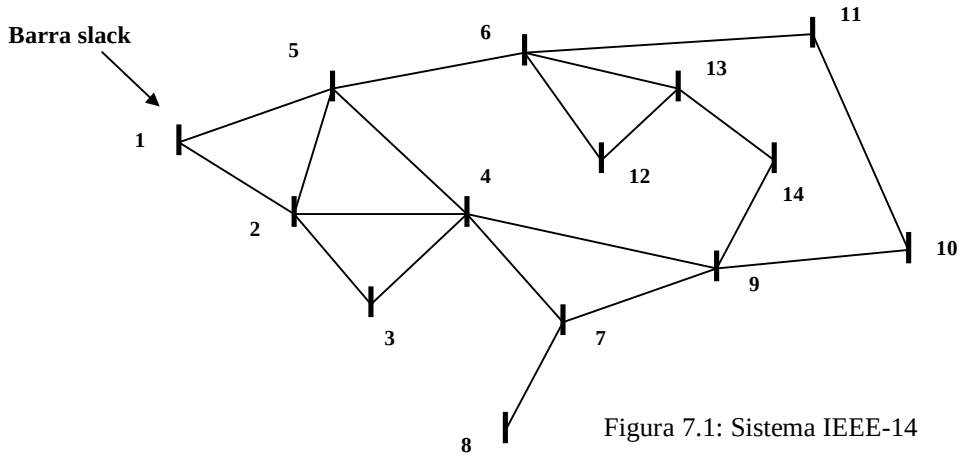

Figura 7.1: Sistema IEEE-14

El sistema IEEE-14 se descompuso de todas las formas posibles de manera a resolverlo utilizando las dos workstations descritas con anterioridad. Considerando que la workstation DEC tiene una capacidad de resolver problemas unas 3 veces mayores que la worstation SUN, en el mismo periodo de tiempo, el sistema fue particionado de forma tal que el procesador DEC resuelva 10 barras y el procesador SUN 3 barras. De esta forma, el número de particiones a ser analizado es:

$$\text{Nro de particiones posibles} = C_{10}^{13} = \frac{13!}{10! \times (13-10)!} = 286$$

Las particiones resultantes fueron numeradas de 1 a 286. Para cada una de las 286 particiones, el problema de flujo de potencia fue resuelto en forma paralela síncrona y asíncrona. La tolerancia fue verificada en la máquina DEC. Las magnitudes medidas y registradas en la resolución de cada partición fueron las siguientes:

- *Tiempo físico*: es el tiempo de reloj utilizado por el sistema hasta llegar a la solución buscada, medido en el procesador más rápido.
- *Tiempo de CPU de la DEC*: es el tiempo efectivamente utilizado por el procesador de la máquina DEC hasta la solución global del problema, sin considerar el tiempo que dicha máquina ha utilizado en atender otros procesos concurrentes con la resolución del Flujo de Potencia eléctrica.
- *Número de iteraciones:* es el número de veces que se actualizaron las variables locales de la workstation DEC.

Los valores experimentales así medidos fueron utilizados para elaborar las tablas y gráficos que se muestran en las secciones siguientes.

7.3.1.- Particiones generadas

El método de partición fue aplicado al Sistema IEEE-14. Utilizando diversas ternas de parámetros *vlim*, *nagrup* y *nvec* en la selección automática de semillas, fueron obtenidas dos particiones: la numero 56 y la número 59, cuya numeración se corresponde con el orden establecido al identificar las 286 posibles particiones. Tomando como criterio de selección el parámetro propuesto en este trabajo, es decir, el radio espectral $\rho(\boldsymbol{H})$ de la matriz de comparación, la partición 59 es la seleccionada como óptima.

Por su parte, el método de descomposición más utilizado en la actualidad, la Descomposición ε, no pudo organizar dos subredes con las dimensiones deseadas, por lo cual no se pudo comparar su desempeño frente al método propuesto.

En las figuras 7.2 y 7.3 se muestran las dos particiones generadas aplicando la metodología presentada en este trabajo.

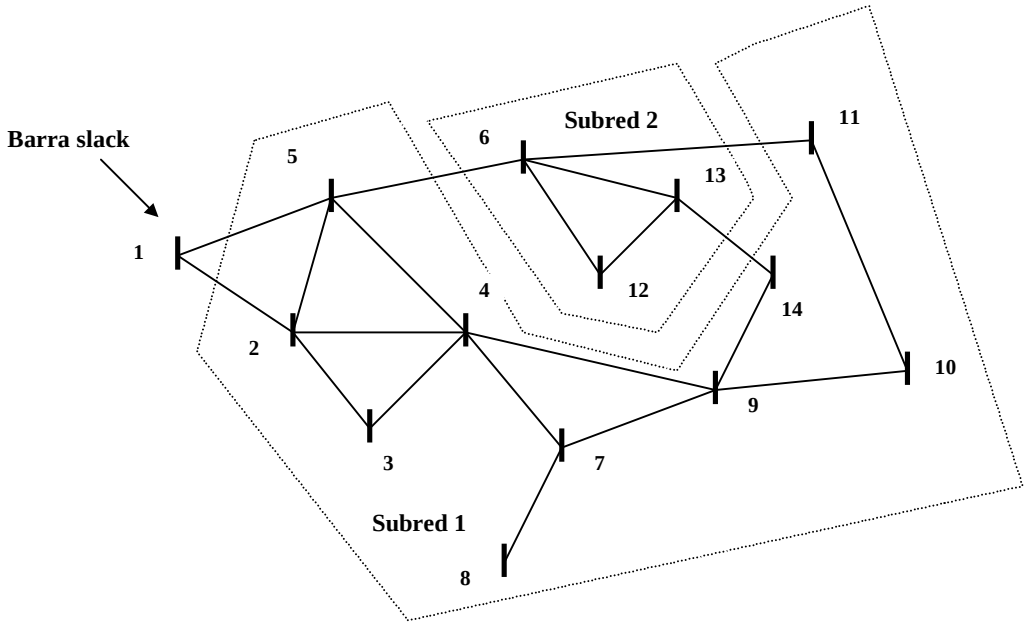

Figura 7.2: Mejor partición generada por el método propuesto.
(partición 59)

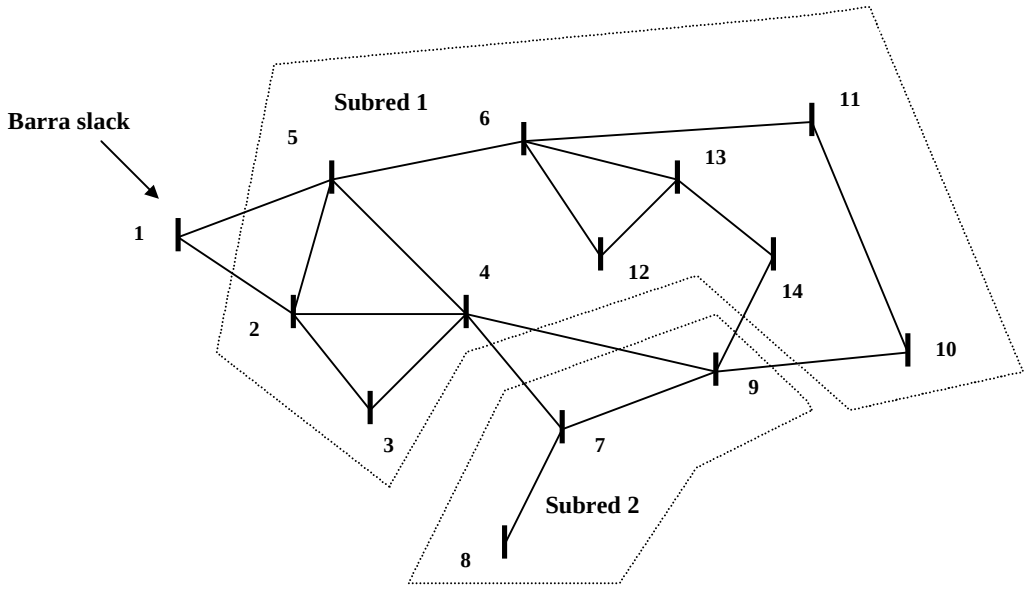

Figura 7.3: Segunda partición generada el método propuesto.
(partición 56)

7.3.2.- Resolución síncrona

En la tabla 7.1 se muestra el posicionamiento relativo de las particiones identificadas más arriba respecto a la totalidad de particiones posibles.

Resolución síncrona del sistema IEEE-14						
Identificador de Partición	Ubicación[+] respecto a:					
	N° de iteraciones	% sup.	Tiempo Real	% sup.	Tiempo de CPU	% sup.
59	3ra posición	1.04	4ta posición	1.39	3ra posición	1.04
56	47ª posición	16.43	34ª posición	11.88	43ª posición	15.03

[+] Existen 286 posibles particiones

Tabla 7.1: Posición de las particiones generadas en el ranking

En las gráficas 7.4, 7.5 y 7.6 se pueden observar el comportamiento de las diferentes particiones con respecto a los valores medidos. Se resaltan especialmente las particiones generadas por el método propuesto:

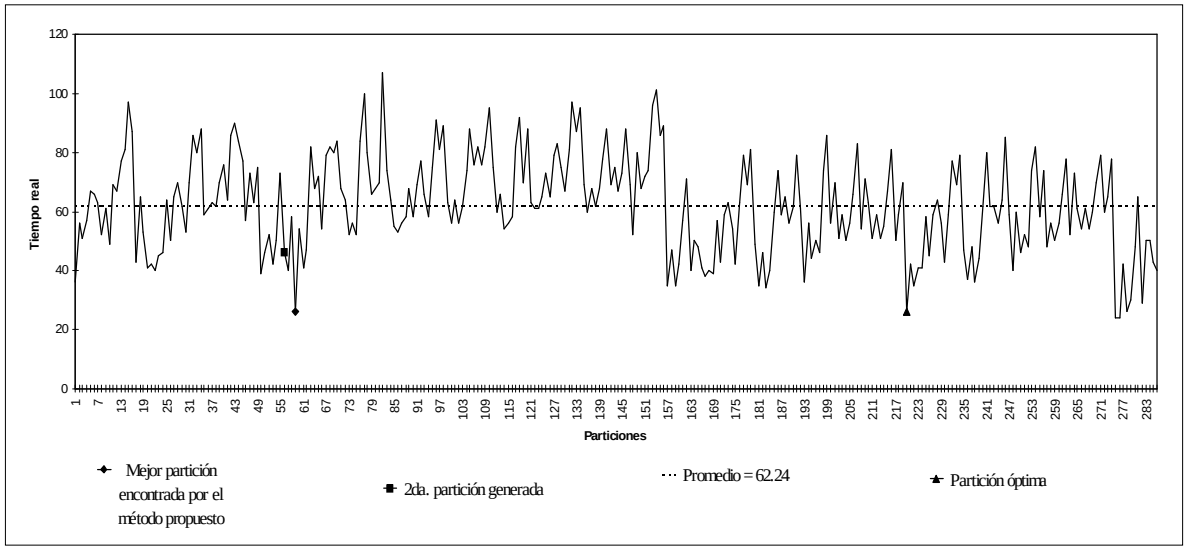

Figura 7.4: Tiempo real. Resolución síncrona del sistema IEEE-14

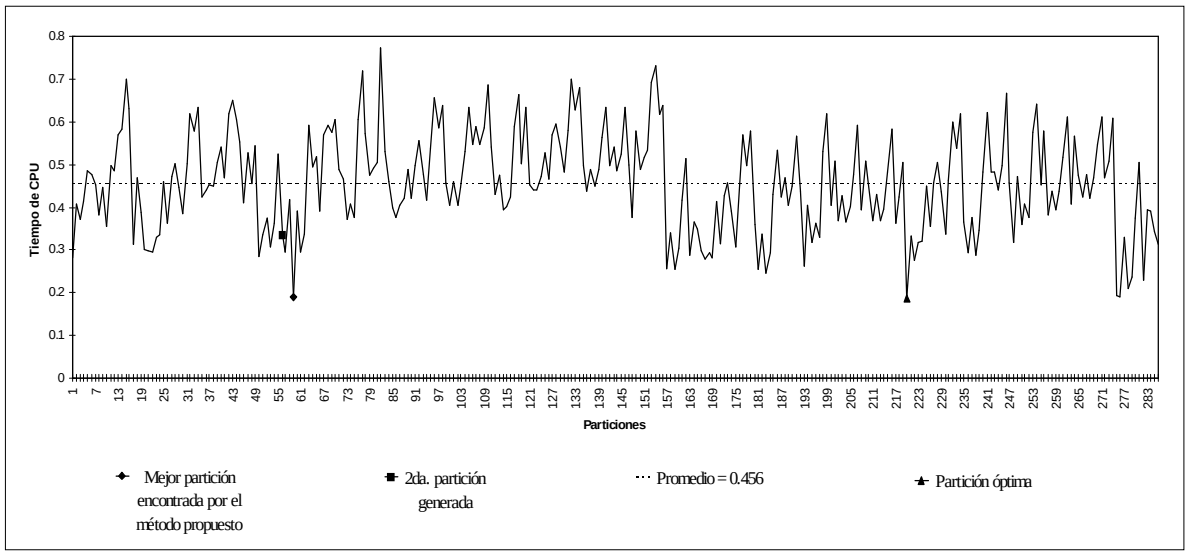

Figura 7.5: Tiempo de CPU. Resolución síncrona del sistema IEEE-14

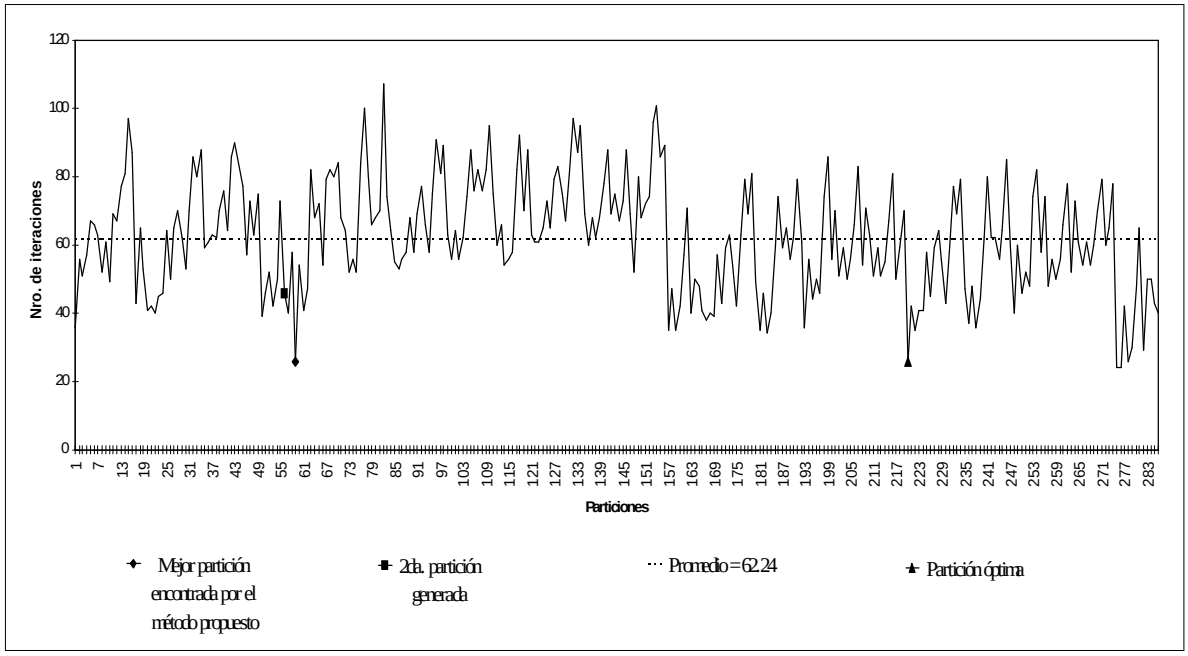

Figura 7.6: Iteraciones. Resolución síncrona del sistema IEEE-14

Puede observarse fácilmente que las dos particiones generadas por el método propuesto mostraron un comportamiento altamente satisfactorio, presentando un desempeño muy superior al promedio. La partición indicada como la mejor disponible según el criterio de selección propuesto se encontró por lo general dentro del 1.40 % superior con respecto a todas las magnitudes medidas. Es importante resaltar que ni la Descomposición ε (sección 4.1) ni el método de Vale et al. (sección 4.2) consiguieron descomponer el sistema IEEE-14 en la proporción deseada.

7.3.3.- Resolución asíncrona

Se resolvieron en forma asíncrona las 286 posibles particiones, es decir, de forma tal que los procesadores hagan uso del valor más actualizado que disponían para cada variable calculada por los otros procesadores del sistema distribuido sin

implementar barreras de sincronización (secc. 2.2). Los resultados experimentales arrojados son ligeramente diferentes a los obtenidos en la resolución síncrona. Si bien con respecto al número de iteraciones el desempeño de las particiones generadas se mostró inferior que antes, hubo un repunte en lo que respeta a los tiempos reales y de CPU, encontrándose inclusive que la mejor partición seleccionada por el método es efectivamente la óptima. Esto puede apreciarse mejor en la tabla 6.2:

Resolución asíncrona del sistema IEEE-14						
Número de Partición	Ubicación[+] respecto a:					
	N° de iteraciones	% sup.	Tiempo Real	% sup.	Tiempo de CPU	% sup.
59	7ª posición	2.44	1ra posición	0.349	1ra posición	0.349
56	82ª posición	28.67	45ª posición	15.73	56ª posición	19.58

[+] Existen 286 posibles particiones

Tabla 7.2: Posición de las particiones generadas en el ranking

Se muestran a continuación las diversas gráficas donde se aprecia el desempeño de todas las particiones estudiadas en este análisis exhaustivo.

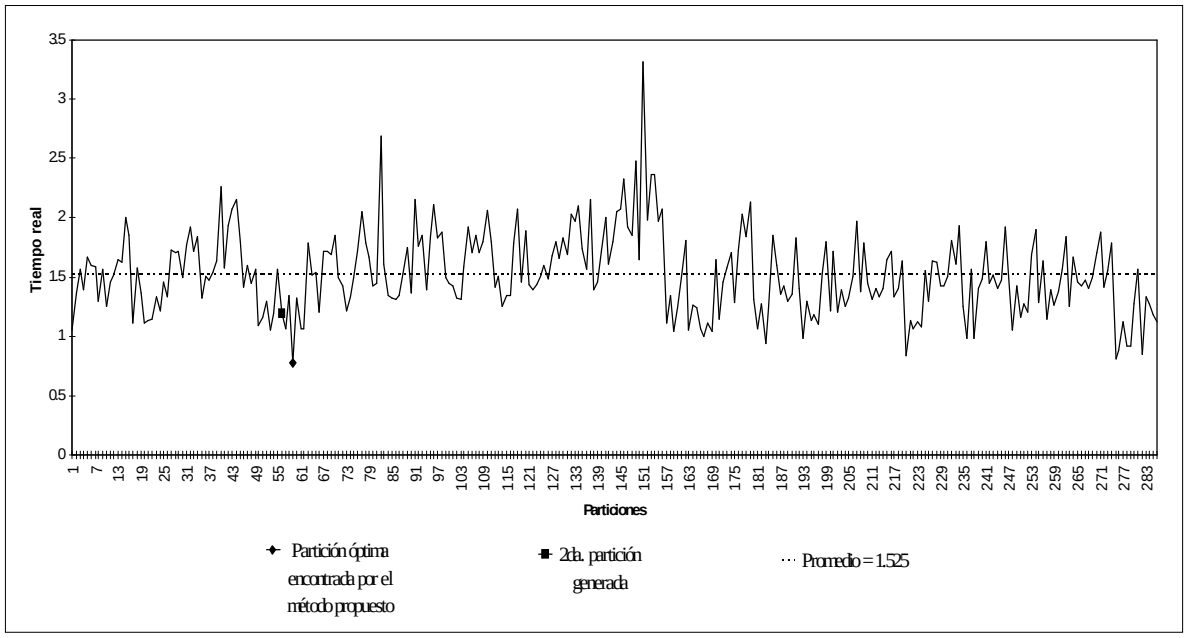

Figura 7.7: Tiempo real. Resolución asíncrona del sistema IEEE-14

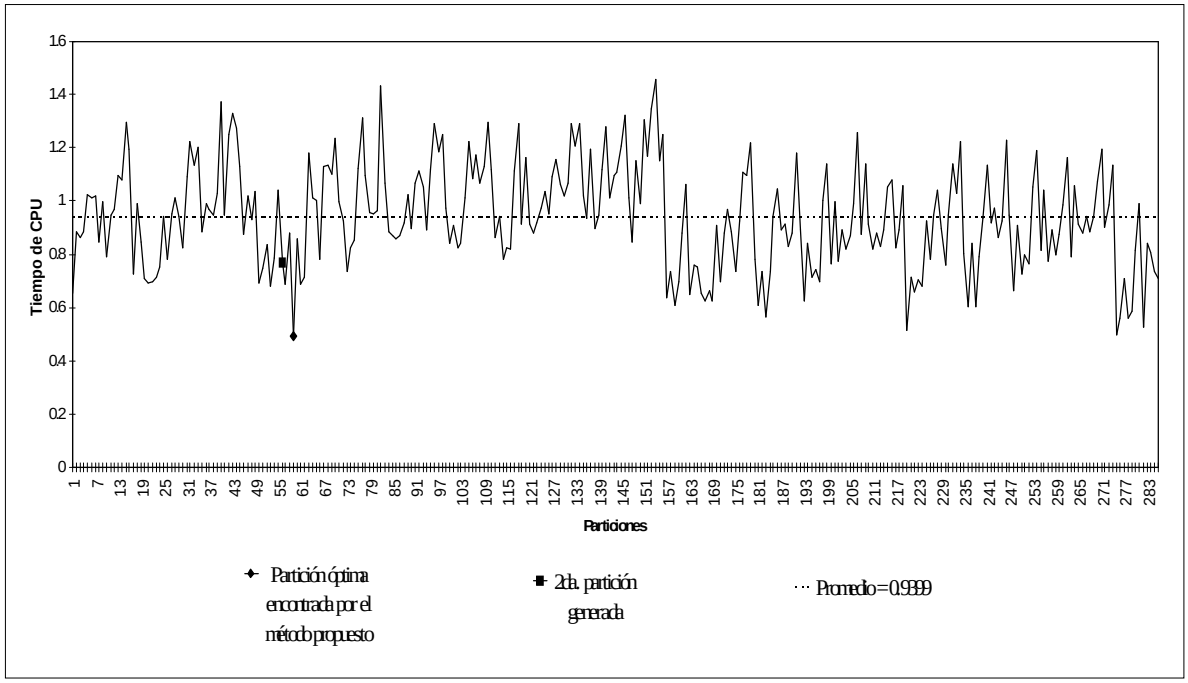

Figura 7.8: Tiempo de CPU. Resolución asíncrona del sistema IEEE-14

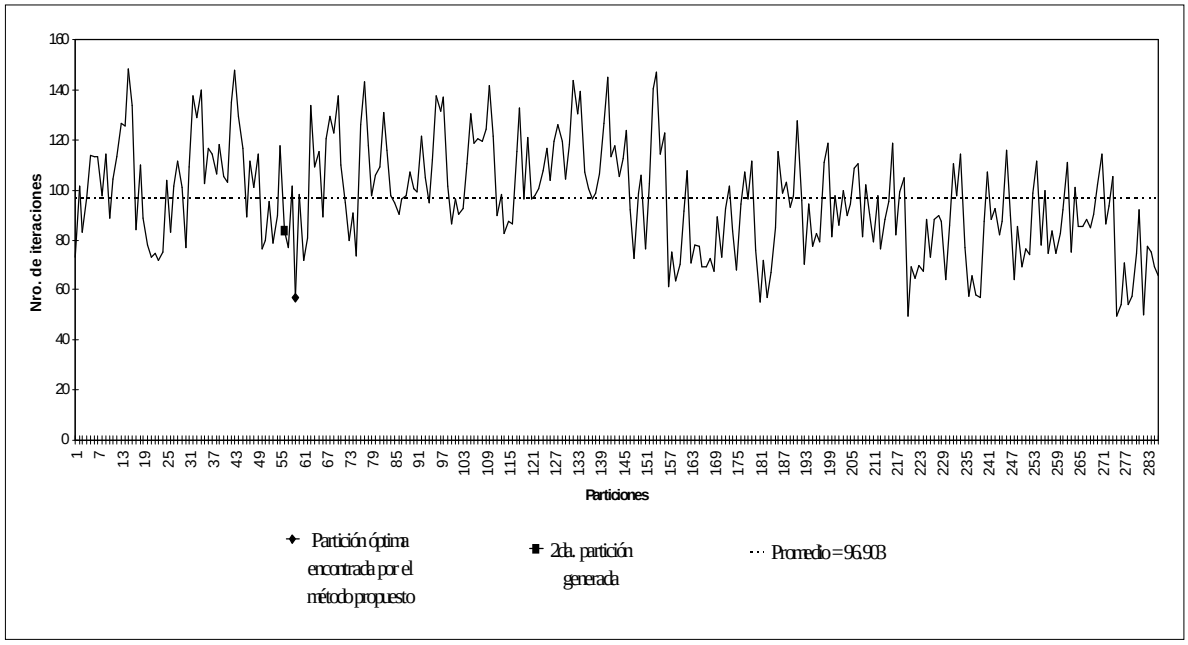

Figura 7.9: Iteraciones. Resolución asíncrona del sistema IEEE-14

En conclusión, el método propuesto demostró un excelente desempeño en la solución asíncrona del sistema IEEE-14, detectando a la partición que consigue resolver el problema del flujo de potencia con el menor tiempo de procesamiento. Es importante recordar que es justamente la implementación asíncrona la más interesante computacionalmente, por aprovechar mejor los recursos computacionales al no perder tiempo en barreras de sincronización.

7.3.4.- Comportamiento del parámetro de selección propuesto

Una de las motivaciones principales del presente trabajo fue la de proponer un parámetro válido que pudiera servir como una clara referencia de la calidad de una partición en términos del procesamiento paralelo. En las tablas y gráficos que siguen, se analiza el comportamiento del radio espectral de la matriz de comparación como parámetro de selección, comparándolo con el parámetro más utilizado hasta el momento, propuesto por Vale et al.[32], que consiste en la sumatoria de los valores absolutos de todas las ligaciones cortadas por la partición, o parámetro “Par_A” (sección 5.5).

La tabla 7.3 indica las correlaciones entre las diversas magnitudes medidas y los parámetros de selección, tanto en el caso síncrono como en el asíncrono. Se verificó en ambos casos que el radio espectral $\rho(\mathbf{H})$ presenta una mejor correlación que el parámetro de selección Par_A con respecto a los tiempos de procesamiento y número de iteraciones.

Correlaciones	Iteraciones		Tiempo Real		Tiempo de CPU	
	sinc.	asinc.	sinc.	asinc.	sinc.	asinc.
$\rho(H)$	0.414	0.87	0.378	0.343	0.41	0.392
Par_A	0.0377	-0.05	0.0104	0.089	0.0137	0.0137

Tabla 7.3: Correlaciones: sistema IEEE-14

Se muestran además en las figuras 7.10 a 7.13 las gráficas comparativas entre los valores normalizados de los parámetros de selección analizados y los tiempos reales medidos

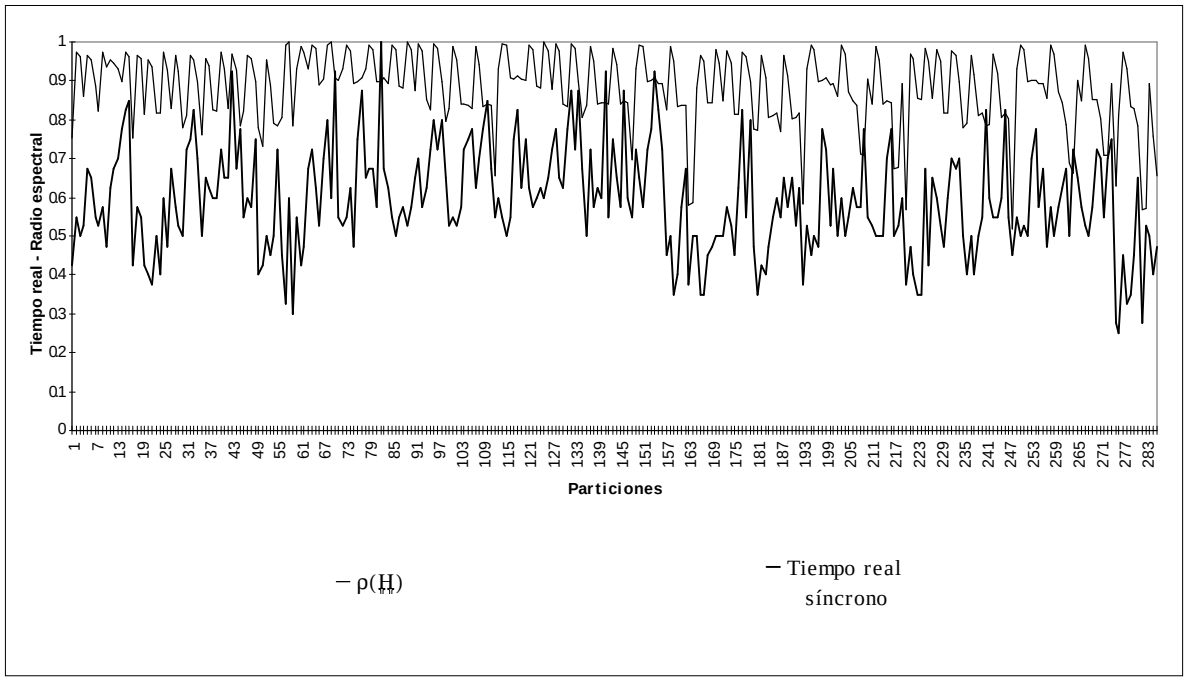

Figura 7.10: Gráfica normalizada: Tiempo real síncrono - ρ(**H**). Sistema IEEE-14

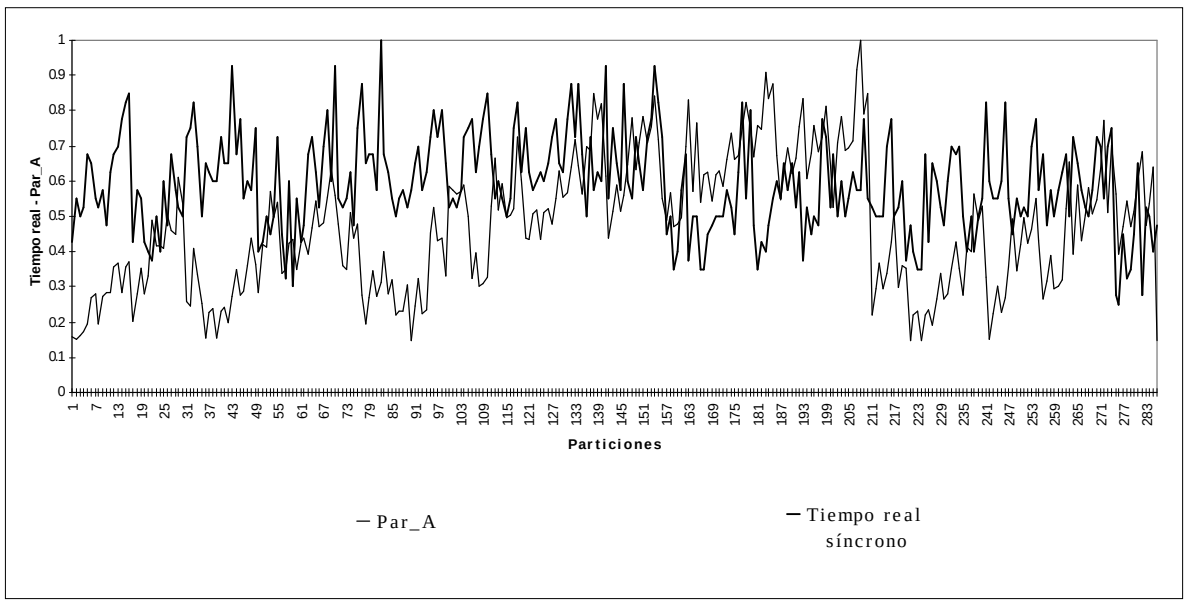

Figura 7.11: Gráfica normalizada: Tiempo real síncrono - Par_A. Sistema IEEE-14

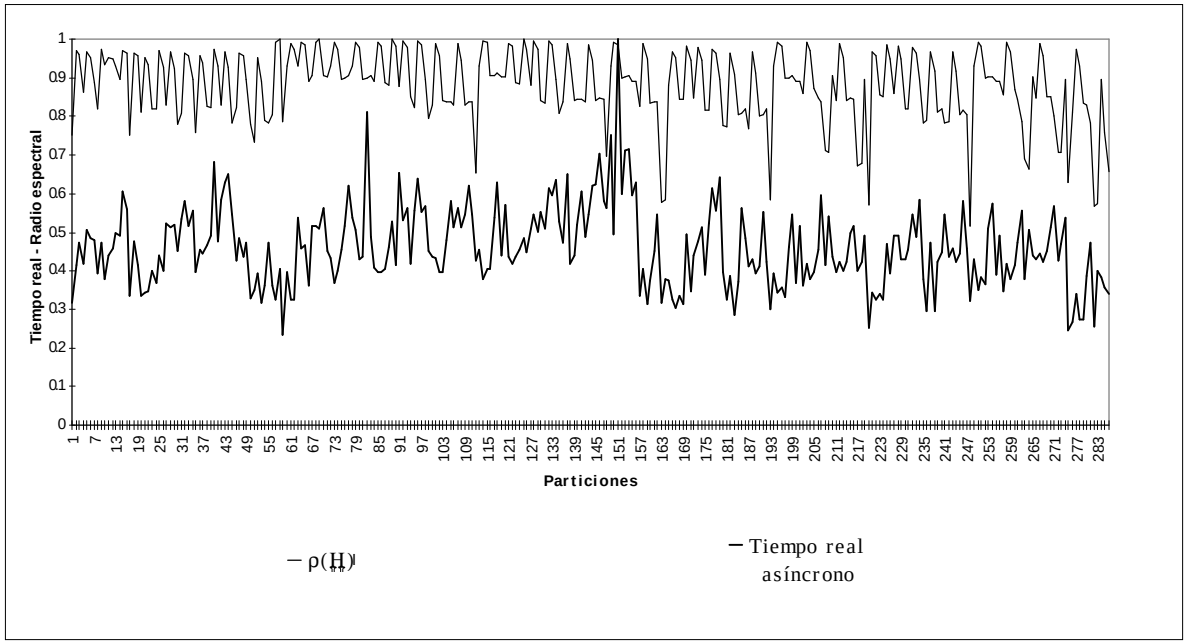

Figura 7.12: Gráfica normalizada: Tiempo real asíncrono - ρ(**H**). Sistema IEEE-14

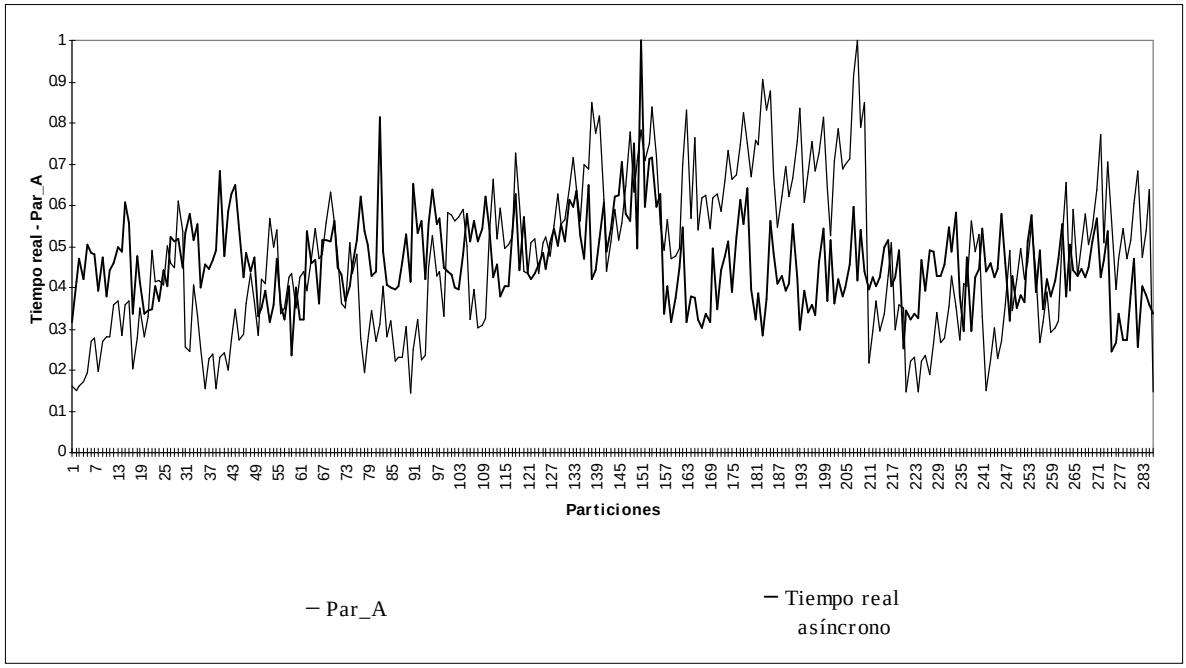

Figura 7.13: Gráfica normalizada: Tiempo real asíncrono - Par_A. Sistema IEEE-14

Conforme a estos resultados, el parámetro de selección $\rho(\mathbf{H})$ propuesto en este trabajo es la mejor opción como referencia para determinar la calidad de una dada partición frente a otra. Sin embargo, el precio que se paga por esto es un mayor costo computacional, pues el cálculo del radio espectral de una matriz es una operación cuya complejidad aumenta de forma exponencial con la dimensión de la matriz.

7.4.- Resolución del sistema IEEE-118

Con la intención de verificar la eficiencia del método propuesto utilizando sistemas eléctricos de mayor tamaño, se seleccionó al sistema IEEE-118. Es evidente que un estudio exhaustivo sobre todas las formas posibles de partir este sistema es imposible, debido a la dimensión del problema. Si, por ejemplo, se deseara partirlo simplemente en dos subredes iguales, tendríamos la siguiente cantidad de particiones:

Nro de particiones posibles = $C_{58}^{117} = \frac{117!}{58! \times (117-58)!} = 1.2157 \times 10^{34}$

Evaluar dicho número de particiones es del todo impráctico, por la cantidad de trabajo computacional implicado, lo que resulta imposible de experimentar aún durante toda la vida útil de una persona, utilizando las computadoras disponibles.

A raíz de esta imposibilidad práctica, en las experimentaciones realizadas sobre este sistema eléctrico se analizaron las siguientes particiones:

- Las particiones generadas por el método propuesto. Dentro de ellas se tendrán en cuenta tanto las generadas utilizando semillas seleccionadas automáticamente como las generadas utilizando semillas asignadas manualmente.
- La partición generada por la Descomposición ε.

- Las particiones realizadas manualmente sobre el grafo del sistema. Estas particiones fueron organizadas tomando en cuenta criterios empíricos y la experiencia del operador sobre el problema.
- Particiones generadas de forma aleatoria. Se analizaron un centenar de estas particiones.

Nuevamente, el sistema se resolvió con implementaciones síncronas y asíncronas, pero esta vez con 4 procesadores de similar performance, por lo que el problema fue descompuesto en 4 subredes de aproximadamente igual dimensión. En este caso solamente se midió el tiempo real utilizado por el sistema distribuido en llegar a la solución.

La representación del sistema eléctrico se muestra en la figura 7.14.

7.4.1.- Particiones generadas

Para aplicar el método propuesto a la descomposición del Sistema IEEE - 118, se hizo necesario un análisis mas detenido de los parámetros utilizados en la selección de semillas, tomándose finalmente valores dictados por criterios empíricos válidos en el marco de esta experimentación.

Fueron adoptados los siguientes valores de los parámetros:

- ***vlim*** = un valor tal que sean analizadas el 10% del total de barras (para el sistema en estudio *vlim*=4.9 produce 11 candidatas a semillas).
- ***nagrup*** = 15, pues valores mayores requerirían de un procesador más potente que el INTEL 486 utilizado.
- ***nvec*** = 10, pues es el número máximo para el cual aún se obtienen las 4 semillas requeridas.

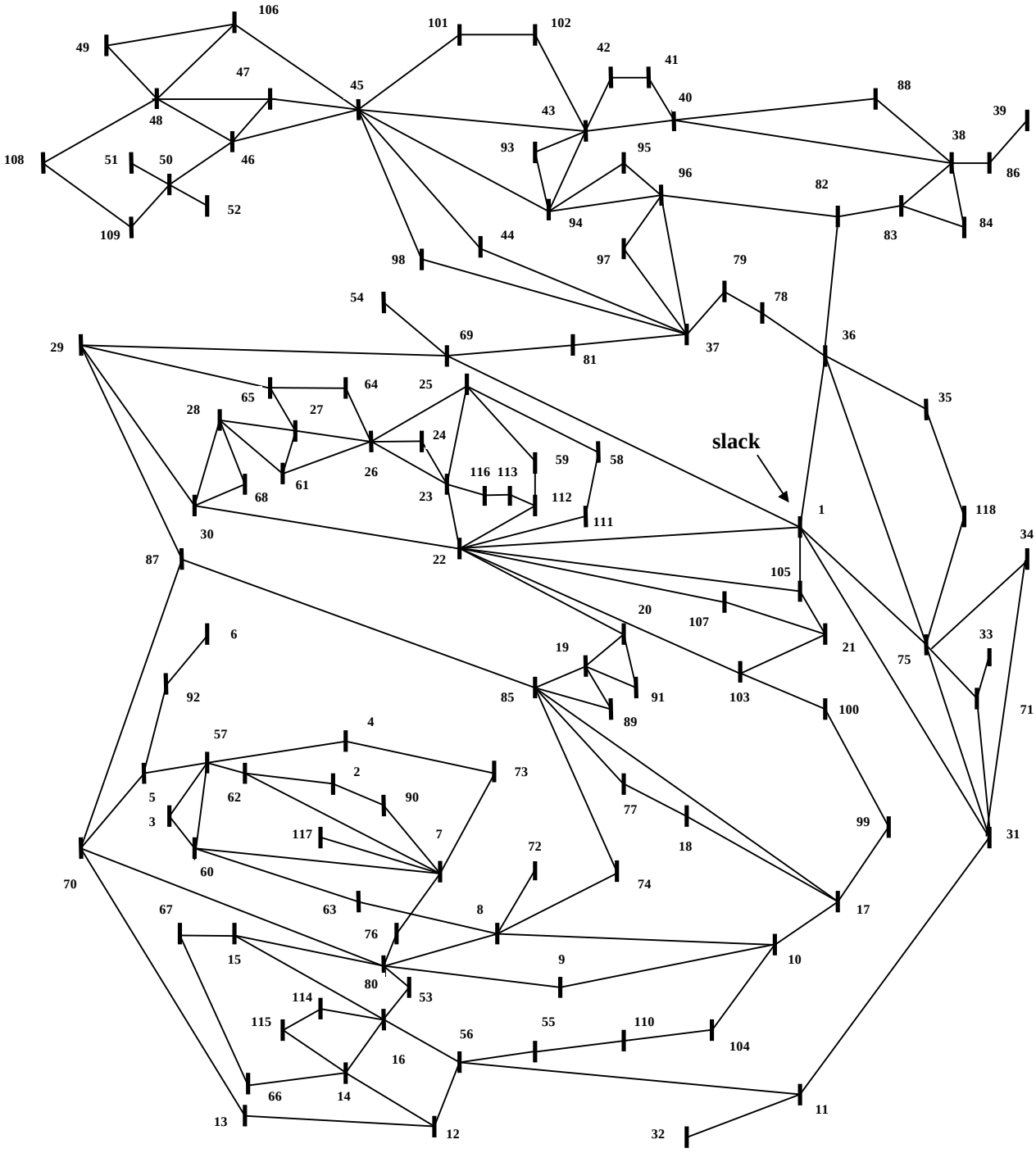

Figura 7.14: Sistema IEEE - 118

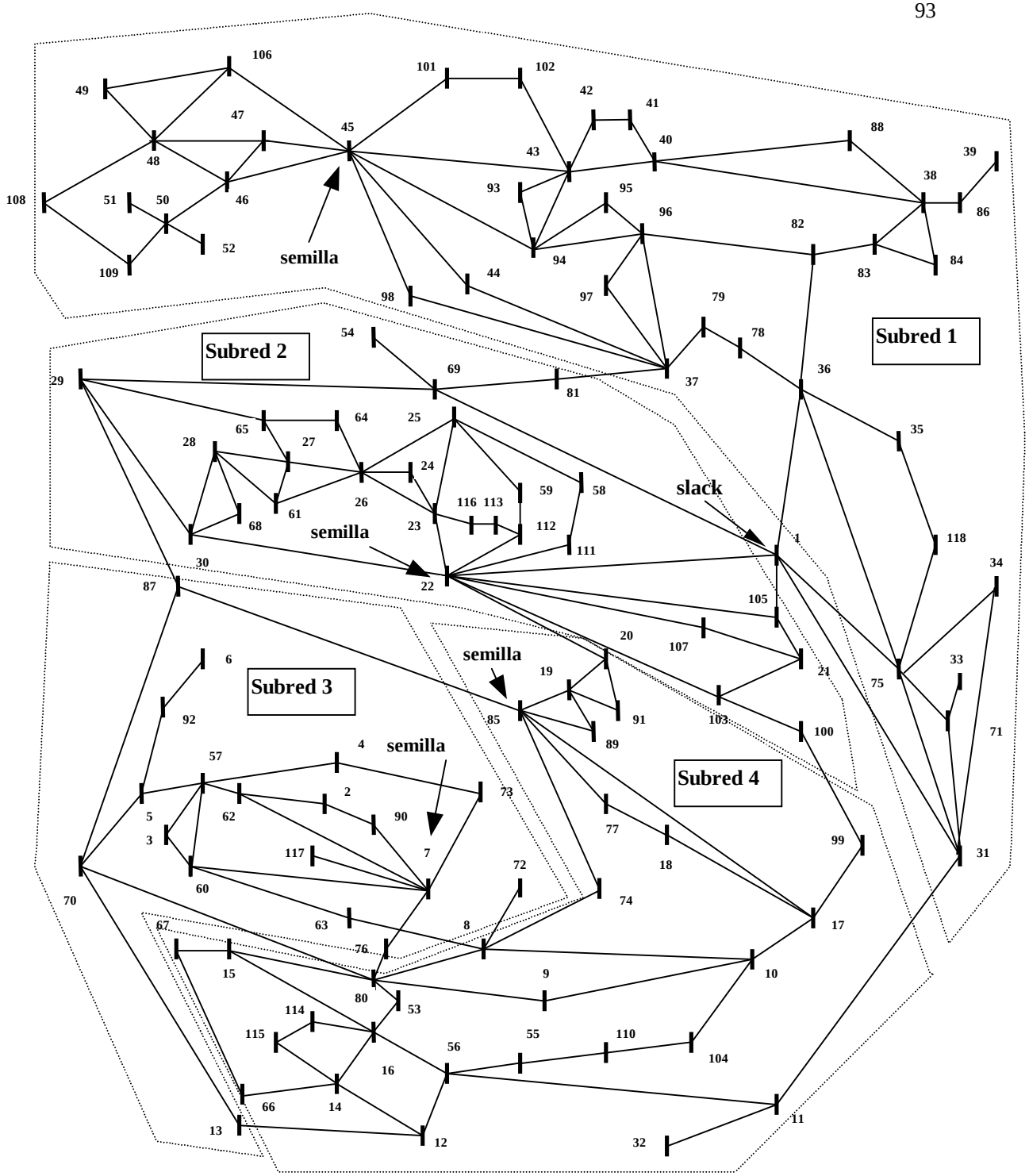

Figura 7.15: Sistema IEEE - 118: Partición generada por el método propuesto

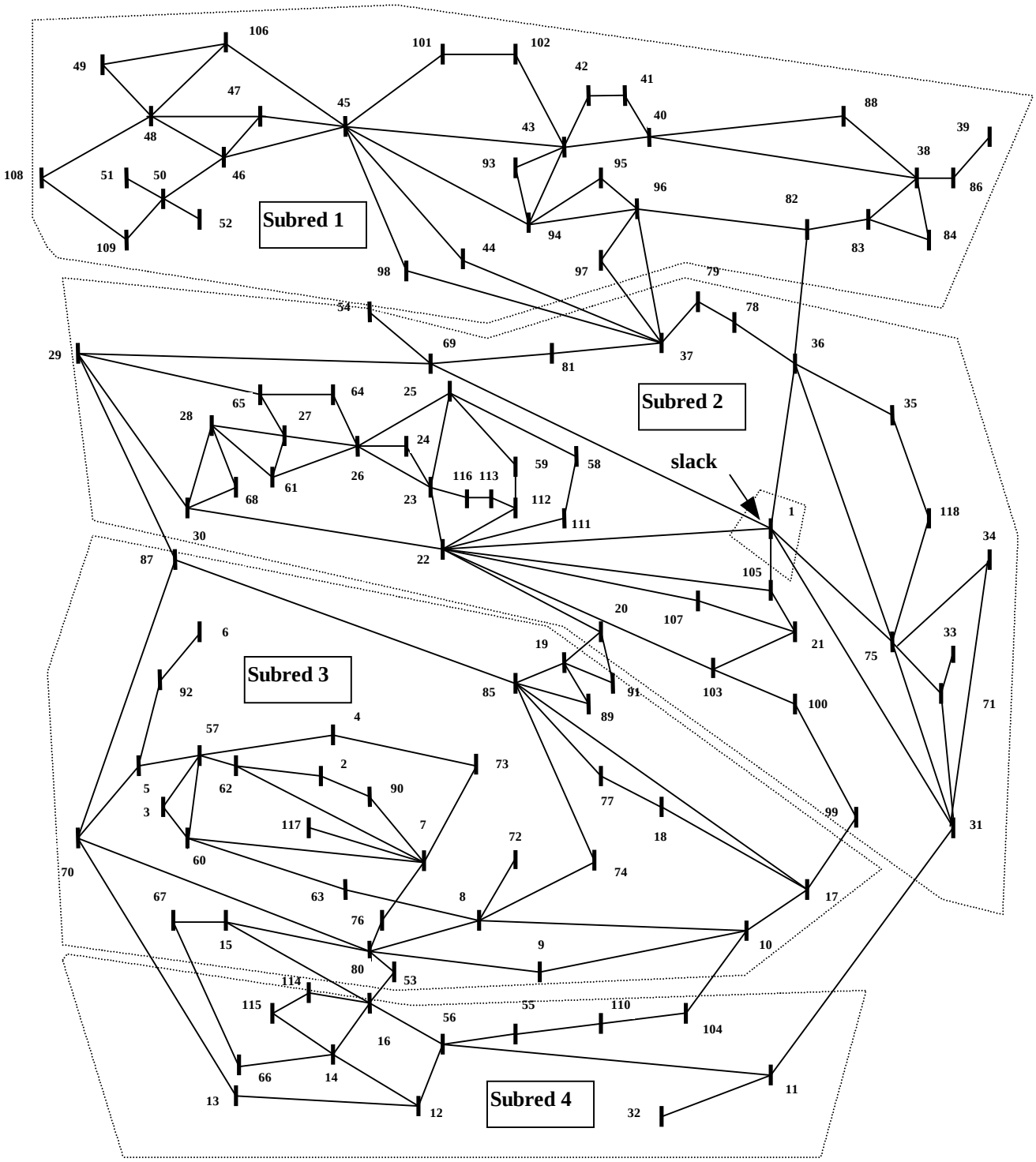

Figura 7.16: Sistema IEEE - 118: Partición generada por la Descomposición ε

Las semillas seleccionadas en forma automática por el método fueron las barras B_{45}, B_{7}, B_{22} y B_{85}. Puede observarse fácilmente en la figura 7.14 que dichas barras semillas constituyen efectivamente centros de agrupamientos de barras, lo que las convierte en candidatas idóneas para una buena partición.

En la figura 7.15 se muestra la partición generada por el método propuesto, mientras que la figura 7.16 muestra la partición generada aplicando la Descomposición ε.

6.4.2.- Resolución síncrona

El problema del Flujo de Potencia para el sistema IEEE-118 fue resuelto utilizando una amplia variedad de particiones, citadas en la sección 7.4.

A la hora de analizar las particiones generadas por el método propuesto, conviene resaltar aquí que dicho método consta principalmente de 4 procesos: clasificación de barras, selección de semillas, generación de la partición y evaluación de particiones. En la etapa de generación de la partición, puede ser obtenida una descomposición a partir de un grupo cualquiera de semillas, pudiendo estas ser seleccionadas de manera automática o manual. Con el objeto de analizar la conveniencia o no de seleccionar las semillas automáticamente, se estudiaron también 4 particiones obtenidas a partir de semillas asignadas de forma manual, utilizando criterios heurísticos basados en la experiencia del operador. Se analizaron además 3 particiones realizadas con criterios geográficos, utilizando la experiencia de expertos con respecto al sistema eléctrico en estudio.

En la tabla 7.4 se muestra el desempeño de las mejores particiones obtenidas, siendo el tiempo real utilizado por el sistema hasta la resolución del problema la magnitud medida en este caso.

Posición	Tipo de partición	Selecc. de semillas	Tiempo (seg.)
1ª posición	Generada por el método propuesto	manual	57
2ª posición	Generada por el método propuesto	manual	58
3ª posición	Generada por el método propuesto	manual	62
3ª posición	Generada por la Descomposición ε	---	62
4ª posición	Generada por el método propuesto	manual	87
5ª posición	Generada por el método propuesto	automática	95
6ª posición	Partición manual (geográfica)	---	139
7ª posición	Partición manual (geográfica)	---	149
8ª posición	Partición manual (geográfica)	---	289

Tabla 7.4: Desempeño de las particiones estudiadas en la resolución síncrona: Sistema IEEE-118.

Puede verificarse que casi la totalidad de las particiones que se encontraron en los mejores puestos fueron generadas por el método propuesto, aunque utilizando selección manual de semillas. La partición generada utilizando la selección automática de semillas no tuvo un desempeño tan satisfactorio como estas últimas, ubicándose sin embargo en una mejor posición que las particiones obtenidas manualmente por un especialista.

La partición generada aplicando la Descomposición ε tuvo un comportamiento bastante satisfactorio, aunque inferior por lo general al presentado por las particiones generadas por el método propuesto.

Adicionalmente a las particiones analizadas más arriba, se resolvió el sistema utilizando 100 particiones generadas en forma aleatoria. Es un hecho notable el que ninguna de esas 100 particiones pudo converger a la solución, demostrando así una vez más que la forma en que la red es descompuesta influye de manera decisiva en el comportamiento de los métodos de resolución.

7.4.3.- Resolución asíncrona

Al utilizar el asincronismo en la resolución de problemas, los procesadores son utilizados de una manera más eficiente, ya que no existen “tiempos muertos”, es decir, intervalos de tiempo en los cuales los procesadores se encuentran detenidos esperando resultados de los demás procesadores. La dificultad consiste en encontrar una partición lo suficientemente buena como para asegurar la convergencia y aprovechar los menores tiempos de procesamiento, siendo que el comportamiento de los métodos de resolución bloque iterativos presenta características que hacen que la convergencia de los mismos sea difícil de obtener en un contexto asíncrono [6, 10].

De lo anterior concluimos que los resultados obtenidos resolviendo en forma asíncrona el Sistema IEEE-118 son más importantes en el contexto de la computación distribuida que los obtenidos en la resolución síncrona. En la tabla 7.5 se muestra el comportamiento de las diversas particiones estudiadas.

Posición	Tipo de partición	Selecc. de semillas	Tiempo (seg.)
1ra posición	Generada por el método propuesto	automática	30
2da posición	Generada por el método propuesto	manual	31
3ra posición	Generada por el método propuesto	manual	43
4ta posición	Generada por el método propuesto	manual	44
5ta posición	Generada por la Descomposición $_{\varepsilon}$	---	189
6ta posición	Generada por el método propuesto	manual	N.C.
6ta posición	Partición manual (geográfica)	---	N.C.
6ta posición	Partición manual (geográfica)	---	N.C.
7a posición	Partición manual (geográfica)	---	N.C.

Tabla 7.5 Desempeño de las particiones estudiadas en la resolución asíncrona: Sistema IEEE-118.

Es evidente aquí que el método propuesto fue capaz de generar las particiones más idóneas para la resolución asíncrona. La selección automática de semillas

demostró en esta oportunidad ser la mejor de las opciones encontradas experimentalmente, recordando la imposibilidad de hacer un estudio exhaustivo de todas las alternativas.

La partición obtenida aplicando la Descomposición ε tuvo un desempeño muy inferior a la mayoría de las particiones generadas por el método propuesto, necesitando un tiempo de resolución 6 veces mayor que la partición encontrada por el método propuesto.

Nótese finalmente que las particiones generadas por el especialista no convergen, por lo que resulta evidente la necesidad/utilidad de los métodos automáticos.

CAPITULO 8: CONCLUSIONES

Como parte de nuestra realidad regional en materia de producción y distribución de energía eléctrica, se puede ver la tendencia de ir interconectando los diferentes subsistemas eléctricos de los países miembros del Mercosur en un gran sistema global.

Dada la necesidad de estudiar sistemas eléctricos cada vez más grandes y complejos, resulta evidente la importancia de poseer herramientas computacionales capaces de estudiar las mejores soluciones y las implicancias de cada decisión tecnológica.

Ante esta situación, la computación distribuida emerge como una herramienta altamente viable en la realidad de los países del Mercosur por su capacidad de ir resolviendo problemas cada vez mayores aprovechando la capacidad computacional existente en forma de redes de computadoras, las cuales podrán ir creciendo conforme con las nuevas exigencias.

Sin embargo, el óptimo aprovechamiento de los sistemas distribuidos se encuentra limitado por la capacidad de descomponer los sistemas a ser estudiados en subsistemas menores capaces de ser resueltos eficientemente con los sistemas distribuidos existentes. Este es el punto crucial de los estudios presentados en este trabajo.

En efecto, se presentó un método capaz de partir un sistema eléctrico en subsistemas menores de proporciones deseadas y con características matemáticas aceptables para su resolución en un contexto distribuido. Es más, resultados experimentales con el sistema IEEE-118 demostraron que el método propuesto es superior a otros similares ya publicados hasta la fecha, cuando implementado en un contexto asíncrono (tabla 7.5, sección 7.4.3).

Entre las características que hacen al método propuesto una excelente alternativa frente a otros métodos de descomposición utilizados hasta la fecha, podemos resaltar las que siguen:

- Permite obtener descomposiciones con tamaños relativos deseados por el operador, conforme sea el sistema distribuido a ser utilizado en la resolución del problema correspondiente. En este sentido, se recuerda que la Descomposición ε no tiene control sobre el tamaño relativo de los subsistemas generados, mientras que el método de la semilla solo genera particiones de igual dimensión. Se enfatiza la importancia de esta característica para el óptimo aprovechamiento de las redes de computadoras de diferentes performances o redes heterogéneas.

- La robustez del método en la selección de los diversos parámetros de partición, como por ejemplo, las semillas utilizadas. En efecto, en los estudios experimentales para la descomposición del sistema IEEE-118, quedó claramente demostrado que aun cambiando las semillas se generan las mismas buenas particiones debido a que, aunque las semillas en sí no sean los principales centros de agrupamientos de barras, las mejores candidatas a semillas son anexadas entre las primeras barras agrupadas de la descomposición en formación, pasando a dominar la dinámica de anexación de barras, y generando por consiguiente la misma partición que la generada con una buena selección de semillas.

- La capacidad de elegir o de seleccionar la mejor de varias particiones posibles utilizando un parámetro de selección matemáticamente fundamentado como lo es $_b(\mathbf{H})$ en lugar de parámetros meramente intuitivos presentados en [32]. En efecto, en la sección 7.3.4 quedó demostrado que el parámetro de selección aquí propuesto tiene una mejor correlación con las magnitudes cuya optimización se desea, y de hecho, en el ejemplo IEEE-14 el parámetro Par_A no hubiera elegido la mejor partición, mientras que $_b(\mathbf{H})$ si lo hace.

- El método propuesto es lo suficientemente general como para poder utilizarlo en la resolución de diversos problemas de ingeniería u otras áreas que conlleven la resolución de sistema de ecuaciones de gran porte (ver Apéndice 3).

En cuanto a los resultados experimentales, se pudo comprobar que el método propuesto genera mejores particiones que las generadas en promedio con un criterio aleatorio, siendo en general una muy buena partición e inclusive la optima teórica en algunos casos, como fuera presentado en la sección 7.3.3 para el sistema IEEE-14. Al mismo tiempo pudo comprobarse experimentalmente que las particiones generadas resultaron superiores a las generadas por la Descomposición ε.

En resumen, el presente trabajo brinda una solución superior a las hoy existentes para la descomposición de redes eléctricas en subsistemas menores y suficientemente desacoplados como para permitir su resolución utilizando sistemas distribuidos heterogéneos, inclusive en un contexto asíncrono, lo que permitiría que países como el nuestro aprovechen su capacidad computacional instalada en el estudio y búsqueda de solución de los grandes problemas regionales del sector eléctrico.

Finalmente, se debe mencionar que la metodología propuesta e implementada en un computador personal podría ser migrada a plataformas computacionales mas poderosas de forma a ampliar su capacidad de análisis. Al mismo tiempo, resulta sumamente importante para la consolidación de la metodología propuesta que esta sea efectivamente utilizada en problemas mas complejos del sector eléctrico que el problema del flujo de potencia eléctrico utilizado a modo de ejemplo en el presente trabajo. Por ejemplo, la metodología propuesta ya se ha utilizado con éxito en la partición de sistemas eléctricos para el estudio de su Punto de Colapso [7].

APÉNDICE 1

Código en lenguaje ANSI C para el método de partición propuesto.

PARTICION DE REDES ELÉCTRICAS PARA SU RESOLUCIÓN UTILIZANDO UN SISTEMA DISTRIBUIDO VERSIÓN 3.0/96

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>
#include "c:\part_aut\inversor.c"
```

Declaración de variables globales

char	**nom[30];**	Nombre del archivo donde se encuentra la matriz de acoplamientos
float	***Mcoef,**	Matriz de acoplamientos
	***Peso,**	Vector de pesos de las barras
	***Tem1;**	Vector auxiliar de pesos de las barras
int	**NN,**	Dimensión del sistema a ser partido
	op1,	Opción de tipo de formula de peso a ser utilizada
	op2,	Opción de tipo de selección de semillas
	op4,	Valor de *nagrup* a ser utilizado
	op5,	Valor de *nvec* a ser utilizado
	op6,	Tipo de parámetro de selección a ser utilizado
	op7,	Numero de subredes a ser generadas
	op9,	Opción de solapamiento
	***W,**	Vector de performances relativas
	***Sem,**	Vector que contiene a las semillas a ser utilizadas
	***Liber,**	Vector que indica la disponibilidad de las barras
	***Posi;**	Vector que contiene el orden decreciente de pesos
float	**op3,**	Valor de *vlim* a ser utilizado
	parametro,	Parámetro de selección calculado en la Etapa4
	op10;	Valor del parámetro Lim_over a ser utilizado

Declaración de funciones

void Opciones(void);	Función que lee los parámetros a ser utilizados por el programa
void Etapa1(char *arg1);	Función que clasifica a las barras
void Etapa2(int op, int np, float psup, int nagrup, int nvec);	Función que selecciona automáticamente a las semillas
int Pesomax(int *mat, int p, int n);	Función que halla la barra mas pesada entre las adyacentes a un grupo de barras dado
int Vernvec(int *Agrup,int *Sem,int nsem,int nvec,int pos,int nk);	Función que verifica la condición de *nvec*
void Etapa3(int NP, int ops, float Lim_over);	Función que realiza la partición de la red utilizando las semillas seleccionadas
float verigual(int dim, float *vector);	Función que verifica la existencia de coincidencia de barras candidatas a ser anexadas
float maxacop(int *mat, int p, int ninc , int incog);	Función que devuelve el mayor acoplamiento de una barra dada con un agrupamiento de barras dado
int cantad(int *mat, int p, int ninc);	Función que devuelve la cantidad de barras adyacentes a un agrupamiento de barras dado
float Etapa4(char *config, int opselecc);	Función que calcula el parámetro de selección de particiones
void blockd(float *matcoef, int *matgrup, int CI, float *ad, int p); **void blocknd(float *matcoef, int *matgrup, int CI, int CJ, float *ad, int f, int c);** **float mult_norm(float *mii,float *mij,int CI,int CJ);**	Funciones utilizadas para construir la matriz de comparación de la partición

```
main (void)
{
```

int	**i;**	Variable de ciclo

Opciones(); Lectura de los parámetros a ser utilizados

Asignación dinámica de memoria

```
if((Tem1=malloc(sizeof(float)*(NN+1)))==NULL)
        {printf("error de asignación de Tem1\n");}
if((Liber=malloc(sizeof(int)*(NN+1)))==NULL)
        {printf("error de asignación de Liber\n");}
if((Posi=malloc(sizeof(int)*(NN+1)))==NULL)
        {printf("error de asignación de Posi\n");}
if((Mcoef=malloc(sizeof(float)*(NN+1)*(NN+1)))==NULL)
        {printf("error de asignación de Mcoef\n");}
if((Peso=malloc(sizeof(float)*(NN+1)))==NULL)
```

```
        {printf("error de asignación de Peso\n");}
if((Sem=malloc(sizeof(int)*(op7+1)))==NULL)
        {printf("error de asignación de Sem\n");}
if((W=malloc(sizeof(int)*(op7+1)))==NULL)
        {printf("error de asignación de W\n");}

Etapa1(nom);                        Calculo de los pesos de las barras
Etapa2(op2,op7,op3,op4,op5);        Selección de semillas
Etapa3(op7,op9,op10);               Generación de la partición
parametro=Etapa4("parti.dat",op6);  Calculo de parámetro de selección
if(op6==1 && op7<=2 )               Impresión de los resultados
        printf("Radio espectral=%f\n",parametro);
if(op6==2 )
        printf("Par_A=%f\n",parametro);
```

Liberación de la memoria asignada dinámicamente

```
free(W);
free(Tem1);
free(Liber);
free(Posi);
free(Mcoef);
free(Peso);
free(Sem);

}
```

FUNCIONES

Opciones:
Función que contiene las asignaciones a las variables
asociadas con las distintas opciones del programa

```
void Opciones(void)
{
int             i;          Variable de ciclo
FILE            *opcion;    Nombre del archivo donde se encuentra la matriz de acoplamiento
char            cade[100];  Variable auxiliar de lectura

if((opcion=fopen("opciones.dat","r"))==NULL)    Apertura del archivo a ser leído
        {printf("no se puede abrir el archivo de opciones\n");
        exit(1);}

fscanf(opcion,"%s",&cade);          Lectura del nombre del archivo de acoplamiento
fscanf(opcion,"%s",&nom);
fscanf(opcion,"%s",&cade);          Lectura de la dimensión del sistema
fscanf(opcion,"%d",&NN);
fscanf(opcion,"%s",&cade);          Lectura del tipo de formula de peso a ser utilizada
fscanf(opcion,"%d",&op1);
fscanf(opcion,"%s",&cade);          Lectura del tipo de selección de semillas
fscanf(opcion,"%d",&op2);
fscanf(opcion,"%s",&cade);          Lectura del número de subredes a ser generadas
fscanf(opcion,"%d",&op7);
for(i=1;i<=op7;i++)                 Para cada una de las subredes que serán generadas:
{fscanf(opcion,"%s",&cade);                 Lectura de la dimensión relativa requerida
fscanf(opcion,"%d",(W+i));}
fscanf(opcion,"%s",&cade);                  Lectura del valor de vlim a ser utilizado
fscanf(opcion,"%f",&op3);
fscanf(opcion,"%s",&cade);          Lectura del valore de nagrup a ser utilizado
fscanf(opcion,"%d",&op4);
fscanf(opcion,"%s",&cade);          Lectura del valor de nvec a ser utilizado
fscanf(opcion,"%d",&op5);
fscanf(opcion,"%s",&cade);          Lectura del la opción de solapamiento
fscanf(opcion,"%d",&op9);
if(op9==1){ fscanf(opcion,"%s",&cade);  Si se admitirá solapamiento:
        fscanf(opcion,"%f",&op10);}         Lectura del valor del parámetro Lim_over
fscanf(opcion,"%s",&cade);          Lectura del tipo de parámetro de selección a ser utilizado
fscanf(opcion,"%d",&op6);
fclose(opcion);                     Cerrar el archivo de opciones
}
```

Etapa1

Función que lee la matriz de coeficientes de un archivo, la carga en la matriz Mcoef()() y calcula el valor delos pesos de las barras

```
void Etapa1(char *arg1)
{
FILE            *arch1;          Nombre del archivo donde se encuentra la matriz de acoplamientos
int             i,j;             Variables de ciclo
float           NA,              Variable auxiliar
                max,             Variable auxiliar en el ordenamiento de pesos
                bs,              Variable auxiliar
                ep,              Variable auxiliar
                M;               Sumatoria de los valores absolutos de las ligaciones

int              nl,             Contador del numero de ligaciones
                cont,            Contador  en el proceso de ordenamiento decreciente de pesos
                pos;             Variable auxiliar en el ordenamiento de los pesos

if((arch1=fopen(arg1,"r"))==NULL)      Apertura del archivo de coeficientes
        {printf("no se puede abrir el archivo\n");
        exit(1);}
fscanf(arch1,"%f",&NA);                Lectura de la dimensión del problema
NN=(int)NA;
nl=0;                                  Inicialización del contador del numero de ligaciones
M=0;                                   Inicialización de la sumatoria del valor absoluto de las ligaciones
for(i=1;i<=NN;i++)                     Cargar la matriz de acoplamientos Mcoef
        for(j=1;j<=NN;j++){
                fscanf(arch1,"%f",(Mcoef+(NN+1)*i+j));
                if(i!=j && *(Mcoef+(NN+1)*i+j)!=0){     Si el acoplamiento es diferente de 0
                        nl++;                                   Se incrementa el contador de numero de ligaciones
                        M=M+fabs(*(Mcoef+(NN+1)*i+j)); }}       Actualizar la sumatoria
M=M/nl;
for(i=1;i<=NN;i++)                     Inicialización a 0 de los pesos
        *(Peso+i)=0;
for(i=1;i<=NN;i++){                    Cálculo de los pesos aplicando la fórmula
        for(j=1; j<=NN; j++){
                if(i==j) continue;              Saltar el elemento diagonal
                if(*(Mcoef+(NN+1)*i+j)==0) continue;    No contar los elementos nulos
                if(op1==1){                     Si se utilizaran valores normalizados
                        bs=*(Mcoef+(NN+1)*i+j)/(*(Mcoef+(NN+1)*i+i));   Aplicar la fórmula correspondiente
                        ep=*(Mcoef+(NN+1)*i+j)/(*(Mcoef+(NN+1)*i+i)*M);
                        *(Peso+i)=*(Peso+i)+pow((double)fabs(bs),(double)fabs(ep));}
                if(op1==2){                     Si no se utilizaran valores normalizados
                        bs=*(Mcoef+(NN+1)*i+j);                 Aplicar la fórmula correspondiente
                        ep=*(Mcoef+(NN+1)*i+j)/M;
                        *(Peso+i)=*(Peso+i)+pow((double)fabs(bs),(double)fabs(ep));}}
        *(Tem1+i)=*(Peso+i); }
cont=0;                                Inicializar el contador de barras ordenadas
while(cont!=NN){                       Mientras haya barras que ordenar
        max=0;                         Ordenar las barras
        for(i=1;i<=NN;i++){
                if(*(Tem1+i)>max){
                        max=*(Tem1+i);
                        pos=i;}}
        cont++;
        *(Tem1+pos)=0;
        *(Posi+cont)=pos;}
fclose(arch1);}
```

Etapa2

Función que selecciona las semillas a ser utilizadas en el algoritmo de partición

```
void Etapa2(int op, int np, float psup, int nagrup, int nvec) {

float           *Sum;            Sumatoria de las barras agrupadas
int             *K,              Conjunto que contendrá las barras a ser evaluadas
                *Agrup;          Matriz que contiene las agrupaciones de barras */

float           maxs;            Variable auxiliar utilizada para calcular la máxima sumatoria
```

```
int             nk,         Número de barras candidatas a semillas
                i,j,        Variables de ciclo
                nsem,       Número de semillas seleccionadas en un momento dado
                pos,        Variable auxiliar para calcular la máxima sumatoria de pesos
                ninc;       Cantidad de barras agrupadas en un momento dado

if(op==1) {                                     Si la selección de semillas será automática
        nk=(int)(NN*psup/100);                  Se calcula el numero de elementos del conjunto K
        Sum=malloc(sizeof(float)*(nk+1));       Asignación dinámica de memoria de las variables locales de la función
        K=malloc(sizeof(int)*(nk+1));
        Agrup=malloc(sizeof(int)*(nk+1)*(NN+1));
        for(i=1;i<=nk;i++)                      Se incluyen en K las nk barras mas pesadas
                *(K+i)=*(Posi+i);
        for(i=1;i<=nk;i++){                     Para cada elemento de K
                for(j=1;j<=NN;j++)
                        *(Liber+j)=0;                   Inicializa el vector de disponibilidad de barras
                *(Agrup+(NN+1)*i+1)=*(K+i);     Incluye al elemento de K en la agrupación
                *(Liber+*(K+i))=1;              Lo elimina como disponible
                ninc=1;                         Hay una barra en la subred
                while(ninc<ngrup) {             Mientras se hayan agrupado menos de nagrup barras
                        *(Agrup+(NN+1)*i+ninc+1)=Pesomax(Agrup,i,ninc);     Selecciona la barra mas pesada entre
                                                                            las adyacentes
                        *(Liber+*(Agrup+(NN+1)*i+ninc+1))=1;                La elimina como disponible
                        ninc++;}}
        for(i=1;i<=nk;i++)                      Calculo de las sumatorias de las barras agrupadas
                for(j=1;j<=ngrup;j++)                   Sumar las nagrup barras agrupadas
                        (Sum+i)=*(Sum+i)+*(Peso+*(Agrup+(NN+1)*i+j));
        nsem=0;
        while(nsem<np) {                        Mientras no se seleccionen la cantidad de semillas deseada
                maxs=0;
                for(i=1;i<=nk;i++)              Identificar la barra con mayor valor de sumatoria de pesos
                        if(*(Sum+i)>maxs) {
                                maxs=*(Sum+i);
                                pos=i; }
                if(Vernvec(Agrup,Sem,nsem,nvec,*(Agrup+(NN+1)*pos+1),nk)) {     Si la condición de nvec se cumple
                        nsem++;                         Se incrementa en 1 el numero de semillas seleccionadas
                        *(Sem+nsem)=*(Agrup+(NN+1)*pos+1);  Se incluye a la seleccionada en el vector de semillas
                        *(Sum+pos)=0; }                 Se elimina a la seleccionada de entre las candidatas
                else    Si la condición de nvec no se verifica
                        *(Sum+pos)=0; }     Se elimina a la seleccionada de entre las candidatas
        free(Sum);                                  Liberación de las variables asignadas dinámicamente
        free(K);
        free(Agrup); }
if(op==2) {                                     Si la selección de semillas será manual */
        for(i=1;i<=np;i++){                     Introducción por teclado de las semillas a ser utilizadas
                printf("Semilla # %d = ",i);
                scanf("%d",(Sem+i)); }}}
```

Pesomax

Función que determina la barra con mayor peso de entre las adyacentes a un conjunto dado

```
int Pesomax(int *mat, int p, int n) {

float           maxp;       Peso de mayor valor
int             selec,      Identificación de barra
                i,j;        Variables de ciclo

maxp=0;
for(i=1;i<=n;i++){                      Para cada elemento de la subred
        for(j=1;j<=NN;j++){             Se analiza cada barra del sistema
                if(Liber[j]==1) continue;       Si la barra ya fue incluida, se la salta
                if((*(Mcoef+(NN+1)*(*(mat+(NN+1)*p+i))+j)!=0 || *(Mcoef+(NN+1)*j+*(mat+(NN+1)*p+i))!=0) &&
                (*(Peso+j)>maxp)) {     Si la barra es adyacente y su peso es mayor al máximo del momento
                        maxp=*(Peso+j);         Se la selecciona como la de máximo peso
                        selec=j; }}}            Se guarda que barra es
return(selec); }
```

Vernvec
Función que verifica la condición de *nvec*

```
int Vernvec(int *Agrup,int *Sem,int nsem,int nvec,int pos,int nk) {

int             resp,           Indica si la condición de nvec se cumple o no
                i,j,k;          Variables de ciclo

resp=1;
for(i=1;i<=nsem;i++){                   Para cada una de las semillas seleccionadas previamente
        for(j=1;j<=nk;j++){             Se detecta la posición de la semilla anterior
                if(*(Agrup+(NN+1)*j+1)==*(Sem+i)) {     Ubicada la semilla anterior en la posición j
                        for(k=2;k<=nviz+1;k++){         Para cada una de las barras agrupadas
                                if(pos==*(Agrup+(NN+1)*j+k))            Verificar si coinciden las barras
                                resp=0; }}}}
return(resp); }
```

Etapa 3
Etapa que genera la descomposición

```
void Etapa3(int NP, int ops, float Lim_over) {

int             i,j,            Variables de ciclo
                cantproc,       Contador auxiliar
                ganador,        Subred ganadora
                nagrup,         Barras agrupadas en un momento dado
                *Procpe,        Procesadores que pelean por una candidata determinada
                *Nprocpe,       Numero de subredes que pelean por una barra
                *Macop,         Vector de máximos acoplamientos
                *Cand,          Vector de barras candidatas de las subredes
                *Nady,          Vector de numero de adyacencias
                *Pelea,         Vector indicador de participación en la pelea
                *Magrup,        Matriz de agrupación
                *C;             Vector de cantidad de barras asociadas a las subredes en formación

char            sal[10];        Variable auxiliar para getchar

FILE            *salida;        Archivo que contendrá a la partición

float           *Q,             Vector de cupo parcial
                maxcop,         Máximo acoplamiento
                igual;          El mayor cupo parcial

                                Asignaciones dinámicas de memoria
if((Magrup=malloc(sizeof(int)*(NN+1)*(op7+1))  )==NULL)   Matriz de agrupación
        {printf("error de asignacion de Magrup\n");}
if((C=malloc(sizeof(int)*(NP+1))  )==NULL)                Vector de cantidad de barras asociadas
        {printf("error de asignacion de C\n");}
if((Q=malloc(sizeof(float)*(NP+1))  )==NULL)              Vector de cupo parcial
        {printf("error de asignacion de Q\n");}
if((Nady=malloc(sizeof(int)*(NP+1))  )==NULL)             Vector de numero de adyacencias
        {printf("error de asignacion de Nady\n");}
if((Pelea=malloc(sizeof(int)*(NP+1))  )==NULL)            Vector indicador de participación en la pelea
        {printf("error de asignacion de Pelea\n");}
if((Cand=malloc(sizeof(int)*(NP+1))  )==NULL)             Vector de barras candidatas de las subredes
        {printf("error de asignacion de Pelea\n");}
if((Macop=malloc(sizeof(float)*(NP+1))  )==NULL)          Vector de máximos acoplamientos
        {printf("error de asignacion de Pelea\n");}
if((Procpe=malloc(sizeof(int)*(NP+1)*(NP+1))  )==NULL)    Procesadores que pelean por una barra determinada
        {printf("error de asignacion de Procpe\n");}
if((Nprocpe=malloc(sizeof(int)*(NP+1))  )==NULL)          Número de subredes que pelean por una barra
        {printf("error de asignacion de Nprocpe\n");}
for(i=1;i<=NN;i++)                                        Inicialización del vector de disponibilidad Liber
        *(Liber+i)=0;                                             Todas las barras están disponibles
for(i=1;i<=NP;i++)                                        Cargar las semillas en Magrup
        *(Magrup+(NN+1)*i+1)=*(Sem+i);
for(i=1;i<=NP;i++){                                       Para cada semilla / partición
        *(Liber+*(Sem+i))=1;                                      Eliminarla como disponible
        *(Peso+*(Sem+i))=0;
```

```
        *(C+i)=1;                                   Cada subred tiene una barra: la semilla
        *(Q+i)=(float)(*(C+i))/((float)*(W+i)); }   Valor inicial de cupo parcial
for(i=1;i<=NP;i++)                                  Calculo de la cantidad de adyacentes de cada subred
        *(Nady+i)=cantad(Magrup,i,*(C+i));
nagrup=NP;                                          Las semillas son las primeras barras agrupadas
while(nagrup<NN) {                                  Mientras haya barras por agrupar
        for(i=1;i<=NP;i++)                          Actualización del valor de cupo parcial
                *(Q+i)=(float)(*(C+i))/((float)*(W+i));   Valor de cupo parcial
        for(i=1;i<=NP;i++){                         Calculo de la cantidad de adyacentes de cada subred
                *(Nady+i)=cantad(Magrup,i,*(C+i));
                if(*(Nady+i)==0)
                        *(Q+i)=-1; }
        igual=verigual(NP,Q);                       Determinación de cuales subredes pelean
                                                    Se verifica si todos los cupos parciales son iguales
        if(igual<0) {                               Si todos los cupos parciales son iguales
                for(i=1;i<=NP;i++)                      Para cada subred en formación
                        if(*(Nady+i)>0)                 Si tiene barras adyacentes
                                *(Pelea+i)=1;                   La subred pelea
                        else *(Pelea+i)=0; }            En caso contrario no pelea
        else{                                       En caso contrario
                for(i=1;i<=NP;i++)                      Para cada subred en formación
                        if(*(Q+i)<igual && *(Q+i)!=-1 )  Si su cupo parcial. es menor al máximo y si tiene
                                                        aun barras adyacentes
                                *(Pelea+i)=1;           La subred pelea
                        else                            En caso contrario
                                *(Pelea+i)=0; }         La subred no pelea
                        Selección de las barras candidatas de cada subred en formación que pelea
        for(i=1;i<=NP;i++)                          Para cada subred en formación
                if(*(Pelea+i)==0)                       Si la subred no pelea
                        *(Cand+i)=-1;                       No tiene candidato
                else                                    De lo contrario
                        *(Cand+i)=Pesomax(Magrup,i,*(C+i));     Se selecciona candidata
        for(i=1;i<=NP;i++)                          Calculo de los máximos acoplamientos con las candidatas
                *(Macop+i)=maxacop( Magrup,i,*(C+i),*(Cand+i));
                                                    Se carga la matriz Procpe y el vector Nprocpe
        for(i=1;i<=NP;i++){                         Para cada barra candidata
                cantproc=0;                             Se inicializa el contador del numero de subredes
                for(j=1;j<=NP;j++)                      Para cada barra
                        if(*(Cand+j)==*(Cand+i)) {
                                cantproc++;
                                *(Procpe+(NP+1)*i+cantproc)=j; }
                *(Nprocpe+i)=cantproc; }                Se asigna un valor a la componente de Nprocpe
        for(i=1;i<=NP;i++){                         Para cada barra candidata */
                if(*(Cand+i)==-1) continue;             Si la subred no pelea, se salta al siguiente ciclo
                        Verificación de la igualdad de máximos acoplamientos
                igual=1;                Se inicializa igual
                for(j=2;j<=*(Nprocpe+i);j++){           Se verifica si todas los acoplamientos son iguales
                        if(*(Macop+*(Procpe+(NP+1)*i+j))!=*(Macop+*(Procpe+(NP+1)*i+1))) Si hay alguna desigualdad
                                igual=1; }                                  No son todos los
                                                                            acoplamientos iguales
                if(igual==1) {                          Si no todos los acoplamientos  son iguales
                        ganador=1;
                        maxcop=0;
                        for(j=1; j<=*(Nprocpe+i); j++){     Se halla con cual subred se halla mas acoplada
                                if(*(Macop+*(Procpe+(NP+1)*i+j))>maxcop) {   Si el acoplamiento es mayor al
                                                                             máximo
                                        maxcop=*(Macop+*(Procpe+(NP+1)*i+j));
                                        ganador=j; }}                        Se guarda cual es el ganador
                        *(Magrup+(NN+1)*(*(Procpe+(NP+1)*i+ganador))+   Se incluye a la barra en la subred
                        *(C+*(Procpe+(NP+1)*i+ganador))+1)=*(Cand+i);    ganadora
                        nagrup++;
                        *(C+*(Procpe+(NP+1)*i+ganador))=*(C+*(Procpe+(NP+1)*i+ganador))+1;   La subred
                                                                                            ganadora tiene
                                                                                            una barra mas
                        *(Liber+*(Cand+i))=1;               Se elimina a la barra como disponible
                        *(Peso+*(Cand+i))=0;
                        for(j=1;j<=*(Nprocpe+i);j++){
                                *(Cand+*(Procpe+(NP+1)*i+j))=1; }}   Se sacan de combate a las demás subredes
                                                                     perdedoras
                else{                                   Si todos los acoplamientos son iguales
```

```
                    if( (*(Peso+*(Cand+i))>Lim_over)          Si el peso de la candidata en disputa es mayor que
                     && (ops==1)) {                           Lim_over y si se realizara solapamiento
                            for(j=1;j<=*(Nprocpe+i);j++){     Para cada procesador que la selecciono como
                                                              candidata
                                    *(Magrup+(NN+1)*(*(Procpe+(NP+1)*i+j))+   Se la incluye en todas las
                                    *(C+*(Procpe+(NP+1)*i+j))+1)=*(Cand+i);   subredes
                                    *(C+*(Procpe+(NP+1)*i+j))=*(C+*(Procpe+(NP+1)*i+j))+1;
                                                              Se incrementa en 1 el numero de barras
                                                              agrupadas en cada subred
                                    *(Liber+*(Cand+i))=1;     Se elimina a la barra como disponible
                                    *(Peso+*(Cand+i))=0;
                                    *(Cand+i)=-1; }           Se elimina a la barra como candidata
                            nagrup++;}                        Se incrementa en 1 el numero de barras agrupadas
                    else{                                     Si no se realizara solapamiento
                            *(Magrup + (NN+1)*( *(Procpe+(NP+1)*i+1)) +   Se asocia la barra a la primera subred
                            *(C+*(Procpe+(NP+1)*i+1))+1)=*(Cand+i);       que la selecciono
                            *(C+*(Procpe+(NP+1)*i+1))=*(C+*(Procpe+(NP+1)*i+1))+1;
                                                              Se incrementa en 1 el numero de barras agrupadas
                                                              en cada subred
                            *(Liber+*(Cand+i))=1;             Se elimina a la barra como disponible
                            *(Peso+*(Cand+i))=0;
                            *(Cand+i)=-1;                     Se elimina a la barra como candidata
                            nagrup++;}}}}                     Se incrementa en 1 el numero de barras agrupadas
free(Procpe);                       Liberación de la memoria asignada dinámicamente
free(Nprocpe);
free(Macop);
free(Cand);
free(Nady);
free(Pelea);
free(Q);
if((salida=fopen("parti.dat","w"))==NULL) {          Impresión en archivo de los resultados
        printf("no se puede abrir el archivo de salida \n");
        exit(1); }
fprintf(salida,"%d procesadores\n",NP);
for(i=1;i<=NP;i++){
        fprintf(salida," Procesador %d  con %d incognitas \n",i,*(C+i));
        fflush(salida);
        for(j=1;j<=*(C+i);j++){
                fprintf(salida," %d ",*(Magrup+(NN+1)*i+j));
                fflush(salida); }
        fprintf(salida,"\n");
        fflush(salida); }
fclose(salida);
free(C); }
```

Verigual

Función que verifica si todas las componentes de un vector son iguales

```
float verigual( int dim, float *vector ) {
int             i,cont;
float           max;

cont=1;                             Inicialización del contador de componentes
max=*(vector+1);                    Inicialización del indicador de la máxima componente
for(i=2;i<=dim;i++){                Para cada una de las componentes del vector
        if(*(vector+i)==1) {
                cont++;
                continue; }                          Ignorar las componentes de valor -1
        if(*(vector+i)>max) {
                max=*(vector+i); }
        if(*(vector+i)==max)
        cont++;}
if(cont==dim)
        return(-1);                          Retorna -1 si todas las componentes son iguales
else return (max); }                Retorna el valor de la componente mayor
```

Maxacop

Función que retorna el valor del mayor acoplamiento de una subred con una barra dada

```
float maxacop(int *mat, int p, int ninc , int incog ) {
int             i,pos;
float           max,val1,val2;

max=0;
for(i=1;i<=ninc;i++){                                   Para cada uno de los elementos de la sured
        if(*(Mcoef+(NN+1)*incog+*(mat+(NN+1)*p+i))>max) {        Hallar el valor del mayor acoplamiento
                max=*(Mcoef+(NN+1)*incog+*(mat+(NN+1)*p+i));    Se verifica por filas
                pos=i; }                                        Se guarda que barra es
        if(*(Mcoef+(NN+1)*(*(mat+(NN+1)*p+i))+incog)>max) {
                max=*(Mcoef+(NN+1)*(*(mat+(NN+1)*p+i))+incog);  Se verifica por columnas
                pos=i; }}                                       Se guarda que barra es
val1=*(Mcoef+(NN+1)*incog+*(mat+(NN+1)*p+pos));
val2=*(Mcoef+(NN+1)*(*(mat+(NN+1)*p+pos))+incog);
if(incog==-1)
        return 0;
else{
        if(val1>val2) return(val1);
        else return(val2); }}
```

Cantad

Función que retorna la cantidad de barras adyacentes a una subred dada

```
int cantad( int *mat, int p, int ninc ) {
int             *Free,
                nady,           Numero de adyacencias
                i,j;

nady=0;                                         Inicialización del numero de adyacencias
if((Free=malloc(sizeof(int)*(NN+1)))==NULL)     Vector auxiliar de disponibilidad
        {printf("error de asignacion de Free en la funcion cantad()\n");}
for(i=1;i<=NN;i++){                             Inicialización de Free a 1
        *(Free+i)=1;
        if(*(Liber+i)==1)                               Si la barra no se encuentra disponible
                *(Free+i)=0; }
for(i=1;i<=ninc;i++)                            Para cada barra de la subred
        for(j=1;j<=NN;j++)                              Para cada barra del sistema
                if((*(Free+j)==1) && (*(mat+(NN+1)*p+i)!=j) &&          Si no fue aún contada
                ((*(Mcoef+(NN+1)*(*(mat+(NN+1)*p+i))+j)!=0)|            Tiene acoplamiento por filas
                (*(Mcoef+(NN+1)*j+*(mat+(NN+1)*p+i))!=0))) {            o por columnas
                        *(Free+j)=0;
                        nady++;}
free(Free);                                     Liberación de la memoria asignada a Free
return(nady); }
```

Etapa 4

Calculo del parametro de selección de la partición

```
float Etapa4(char *config, int opselecc) {

FILE            *arch_cf,       Archivo donde se encuentra la partición
                *mah;           Archivo donde se guardara la matriz de comparación
int             i,j,k,l,auxi;
int             np,             Numero de subredes
                *C,             Numero de barras en cada subred
                *magrup;        Matriz que contiene la partición
float           *aii,*aij,*h,par_A, spec;
char            cade[10];

if((arch_cf=fopen(config,"r"))==NULL) {Lectura del archivo de partición
        printf("no se puede abrir el archivo de la particion de Etapa4\n");
        exit(1); }
fscanf(arch_cf,"%d",&np);               Se lee el numero de subredes
fscanf(arch_cf,"%s",&cade);
if((magrup=malloc(sizeof(int)*(NN+1)*(np+1))  )==NULL)          Asignación dinámica de memoria
        {printf("error de asignacion de magrup en Etapa4\n");}
if((C=malloc(sizeof(int)*(np+1))  )==NULL)
```

```
        {printf("error de asignacion de C en Etapa4\n");}
if((h=malloc(sizeof(float)*(np+1)*(np+1))  )==NULL)
        {printf("error de asignacion de C en Etapa4\n");}
for(i=1;i<=np;i++){                                     Para cada subred
        fscanf(arch_cf,"%s",&cade);
        fscanf(arch_cf,"%d",&auxi);
        fscanf(arch_cf,"%s",&cade);
        fscanf(arch_cf,"%d",(C+i));
        fscanf(arch_cf,"%s",&cade);
        for(j=1;j<=*(C+i);j++){
                fscanf(arch_cf,"%d",(magrup+(NN+1)*i+j)); }}
fclose(arch_cf);
if(opselecc==1) {                                       Si el parámetro de selección sera el Radio espectral
                                                        Calculo de la matriz de comparación H
        for(i=1;i<=np;i++){                                     Para cada una de las filas-bloque de la matriz
                if( (aii=malloc(sizeof(float)*(C[i]+1)*(C[i]+1)) )==NULL)
                        {printf("error de asignacion de aii en Etapa4\n");}
                blockd(Mcoef,magrup,*(C+i),aii,i);              Organiza el bloque diagonal
                inva(*(C+i),aii);                               Invierte la matriz diagonal
                for(j=1;j<=np;j++){                             Para cada bloque
                        if(i==j) {                                      El elemento hii = 0
                                *(h+(np+1)*i+j)=0;
                                continue; }
                        if( (aij=malloc(sizeof(float)* (*(C+i)+1)*(*(C+j)+1)))==NULL)
                                {printf("error de asignacion de aii en Etapa4\n");}
                        blocknd(Mcoef,magrup,*(C+i),*(C+j),aij,i,j);
                        *(h+(np+1)*i+j)=mult_norm(aii,aij,*(C+i),*(C+j));
                        free(aij); }
                free(aii); }
        if((mah=fopen("Mat_h.dat","w"))==NULL) {        Apertura del archivo donde se guardara la matriz de comparación
                printf("no se puede abrir el archivo Mat_h.dat en Etapa4\n");
                exit(1); }
        fprintf(mah,"Matriz de comparacion H \n\n");            Se escribe en el la matriz H
        for(i=1;i<=np;i++)
                for(j=1;j<=np;j++)
                fprintf(mah,"H[%d][%d]=%f\n",i,j,*(h+(np+1)*i+j));
                                                        Calculo del radio espectral para las matrices de comparación dedimensión 2
                if(np==2) {                                     Si la dimensión de H es igual a 2
                        spec=pow((double)*(h+(np+1)*1+2) * (double)(*(h+(np+1)*2+1)),(double)0.5);
                        fprintf(mah,"Radio espectral=%f\n",spec);
                        return spec; }
                if(np>2) {              Si la dimensión de H es mayor que 2
                        printf("Radio espectral no disponible para NP > 2\n\n");
                        fprintf(mah,"Radio espectral no disponible para NP > 2\n\n");
                        printf("La matriz de comparacion se encuentra en el archivo Mat_h.dat\n");}
                fclose(mah); }
if(opselecc==2) {                                       Si el parámetro sera el Par_A
        par_A=0;                                        Inicialización del parámetro a cero
        for(i=1;i<=np;i++){                             Para cada subred
                for(j=1;j<=*(C+i);j++){                         Para cada una de las barras de la subred
                for(k=1;k<=np;k++){                                     Para cada una de las demás subredes
                        if(k==i) continue;                              No se considera a si misma como subred
                        for(l=1;l<=*(C+k);l++){                         Para cada una de las barras de la subred
                                par_A=par_A+fabs(*(Mcoef+ (NN+1)*(*(magrup+(NN+1)*i+j)) +
                                *(magrup+(NN+1)*k+l)));}}}}              Se suman las ramas seccionadas
        return par_A;}
return 0;}
```

Blockd

Función que organiza el bloque diagonal Aii

```
void blockd(float *matcoef, int *matgrup, int CI, float *ad, int p) {

int             i,j;

for(i=1;i<=CI;i++)
        for(j=1;j<=CI;j++){
                *(ad+(CI+1)*i+j)=*(matcoef+(NN+1)*(*(matgrup+(NN+1)*p+i))+*(matgrup+(NN+1)*p+j));}}
```

Blocknd

Función que organiza el bloque no diagonal Aij

```
void blocknd(float *matcoef, int *matgrup, int CI, int CJ, float *ad, int f, int c) {

int             i,j;

for(i=1;i<=CI;i++)
        for(j=1;j<=CJ;j++){
                if(*(matgrup+(NN+1)*f+i)==*(matgrup+(NN+1)*c+j)){
                        *(ad+(CJ+1)*i+j)=0;
                        continue; }
                *(ad+(CJ+1)*i+j)=*(matcoef+(NN+1)*(*(matgrup+(NN+1)*f+i))+*(matgrup+(NN+1)*c+j));}}
```

Mult_norm

Función que multiplica dos matrices y calcula la norma infinita de la matriz producto

```
float mult_norm( float *mii, float *mij, int CI, int CJ ) {

int             i,j,k;          Variables de ciclo
float           *sum,           Vector de sumatorias por filas
                max;            Variable auxiliar para el calculo de la norma infinita

if( (sum=malloc(sizeof(float)*CI))==NULL)                       Asignación dinámica de memoria para el vector sum
        {printf("error de asignacion de sum en funcion mult_norm de Etapa4\n");}
for(i=1;i<=CI;i++)                                      Inicialización de sum a cero
        *(sum+i)=0;
for(i=1;i<=CI;i++)                                      Para cada fila de mii
        for(j=1;j<=CJ;j++)                                      Para cada columna de mij
                for(k=1;k<=CI;k++){
                        *(sum+i)=*(sum+i)+fabs(*(mii+(CI+1)*i+k)*(*(mij+(CJ+1)*k+j)));}
                                Calculo de la norma infinita del vector sum
max=0;
for(i=1;i<=CI;i++){
        if( fabs(*(sum+i)) > max)
        max=fabs(*(sum+i));}
free(sum);
return max;}
```

APÉNDICE 2

Código PVM para resolver el problema del Flujo de Potencia en forma distribuida

Manager 1
Programa Administrador para resolver en forma ASINCRONA el flujo de potencia de un sistema eléctrico, utilizando el método de Newton-Raphson
Utiliza programas lfslave1.c como esclavos

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include "../const.h"
# include "../pvm3.h"
# include<time.h>

#define ECUACION "../lfslave1"                    Nombre de los programas esclavos
#define ENVIAR(tid, msg) pvm_send(tid, msg)N      Función de envío de resultados

externfloat     CONV,                             Tolerancia para el error
                V[CNK],                           Vector tensión en barras
                TH[CNK],                          Vector ángulo en barras
                YSR[CNK],                         Vector admitancia real
                YSI[CNK],                         Vector admitancia imaginaria
                BK[CNK],
                PG[CNK],                          Vector potencia activa generada
                QG[CNK],                          Vector potencia reactiva generada
                PL[CNK],                          Vector potencia activa de carga
                QL[CNK],                          Vector potencia reactiva de carga
                SUS[CNK],
                PMM[CNK],
                QMM[CNK],
                YR[CNL],
                YI[CNL],
                GL[CNL];

extern int      IFLAG,                            Bandera
                NK,                               Numero de barras del sistema
                NL,                               Numero de líneas del sistema
                NR,
                NODE[CNK],                        Barras ordenadas
                BARRA[CNK];                       Orden original de barras
                NPV;                              Numero de barras tipo PV

int             NP,                               Numero de procesadores
                NPQ,                              Numero de barras PQ
                IL[CNL],
                KL[CNL],
                LIM_PV,                           Ultima barra PV
                *PART,                            Matriz que contiene la partición del sistema
                ORDR[CNK],ORD[CNK],dimhi,
                NBP[10],                          Numero de barras por procesador
                NOPQ[10],                         Numero de barras PQ que recibe un procesador
                NOPV[10];                         Numero de barras PV que recibe un procesador

float           X[2*CNK],                         Vector incógnitas
                VORIG[CNK],
                ERROR[CNK],
                error,max, errorp, errorhs, errorhl,
                THR[CNK],VR[CNK],
                G[CNK][CNK],                      Matriz que contiene las conductancias del sistema
                B[CNK][CNK];                      Matriz que contiene las susceptancias del sistema
                P[CNK],                           Potencias activas calculadas
                Q[CNK],                           Potencias reactivas calculadas
                delta[2*CNK];                     Incremento de las tensiones y fases en barras

void FORMA_GB(void);
void MULTIPLICACION (float vec2[cdim], float vec3[cdim], int dim);
float MAXIMO(float vec[cdim], int dim);
float f[CNK*2],fl[CNK*2],a[cdim][cdim];

int RECIBIR(int tid, int msg ) {                  Función de recepción asíncrona de resultados
int bufid;
int cont=0;
```

```
bufid=0;
bufid=pvm_nrecv(tid, msg);
while(pvm_probe(tid,msg)>0) {
        bufid=pvm_nrecv(tid,msg);}
return(bufid);}

void main(void) {

int             i,j,h,                  Variables de ciclos
                iter1,                  Contador de iteraciones
                cont1, conti,contj,contin,  Contadores auxiliares
                verover,                Variable verificadora de solapamiento
                int TID[20];            Vector que contiene a los números de identificación de las tareas esclavas
                int info;               Variable auxiliar de verificación de procesos
                int mytid;              Número de identificación de la tarea administradora
                int msg,                Etiqueta de envío de mensajes
                bufid, byte, source;    Variables auxiliares

time_t          first,second;           Variables de medición de tiempo real
clock_t         tiempo;

char*   host[] = {"","Linux1.cnc.una.py",   Vector donde se indican las máquinas donde las tareas esclavas estarán
                "Linux2.cnc.una.py",        residentes
                "Linux3.cnc.una.py",
                "Linux4.cnc.una.py",
                "Linux5.cnc.una.py",
                "Linux6.cnc.una.py",
                "Linux7.cnc.una.py",
                "Linux8.cnc.una.py",
                "Linux9.cnc.una.py",
                "Linux10.cnc.una.py"}

FILE            *arq_cf;                Archivo que contiene la partición

mytid = pvm_mytid();                    La tarea administradora inicia su sesión en PVM
iter1=0;                                Inicialización del número de iteraciones a cero
LFINP();                                Lectura de los datos del sistema a ser resuelto
NPQ=NK-1-NPV;                           Cálculo del número de barras tipo PQ
for (i=1; i<=NPV+1; i++){               Se crea el vector VORIG
        VORIG[i]=1.0;}
for (i=NPV+2; i<=NK; i++){
        VORIG[i]=0.0;}
```

Lectura del archivoCONF.dat, obteniendo el vector ORD[] , que contiene el orden en que será modificada la matriz Ybarra
Se organiza el vector NOPV[], que contiene el numero de barras PV que recibe un procesador
Se organiza el vector NBP[], que contiene el numero de barras por procesador

```
if ((arq_cf=fopen("conf.dat","r")) == NULL){
        printf("Archivo CONF.DAT no encontrado. Verifique.\n");
        exit(1);}
fscanf(arq_cf,"%d",&NP);                Lectura del numero de procesadores
printf("Nro de Procesadores = %d\n",NP);    Impresión del número de procesadores
printf("Numero de barras = %d\n",NK);       Impresión del número de barras del sistema
printf("Tolerancia = %f\n",CONV);           Impresión de la tolerancia

if((PART=malloc(sizeof(int)*(NP+1)*(NK+1)))==NULL){     Asignación dinámica del la matriz PART, que contiene que
        printf("Error en asignacion dinamica de PART\n");   barras serán asignadas a cada procesador
        exit(1);}
cont1=2;
ORD[1]=BARRA[1];                        ORD[1] es la barra slack
for(i=1; i<=NP; i++){                   Para cada procesador
        fscanf(arq_cf,"%d",&NBP[i]);            Lectura del número de barras
        for(j=1; j<=NBP[i]; j++){               Para cada barra
                fscanf(arq_cf,"%d",&ORD[cont1]);        Leer que barra es
                if(VORIG[ORD[cont1]]>0)
                        NOPV[i]=NOPV[i]+1;              Verificar que tipo de barra es
                cont1++;}
        NOPQ[i]=NBP[i]-NOPV[i];}                Calcular la cantidad de barras de cada tipo
```

Llenado de la matriz PART

```
cont1=1;                                        Posicionador en el vector ORD
for(i=1;i<=NP;i++)
        for(j=1;j<=NBP[i];j++){
                cont1++;
                *(PART+(NK+1)*i+j)=ORD[cont1];}

FORMA_GB();                                     Construcción de las matrices de conductancias y susceptancias

for(i=NPV+2;i<=NK;i++)                          Inicialización del vector V
        V[i]=1;
for(i=1;i<=NP;i++){                             Activación de los procesos esclavos
        info=pvm_spawn(ECUACION,(char**)0,1,host[i],1,&TID[i]);
        printf("Activacion de hijo %d en %s = %d \n", i,host[i], info);
        printf("TID %d = %d \n",i,TID[i]);}
for(h=1;h<=NP;h++){                             Envío de los datos iniciales a los procesos esclavos
        pvm_initsend(PvmDataDefault);           Limpiar el buffer de envío
        pvm_pkint(&NP,1,1);                     Empaquetar el número de procesadores
        pvm_pkint(TID,NP+1,1);                  Empaquetar el vector de identificación de los procesos esclavos
        pvm_pkint(&h,1,1);                      Empaquetar el orden del esclavo que recibirá los datos
        pvm_pkint(&NK,1,1);                     Empaquetar el número de barras del sistema
        pvm_pkint(&NBP[h],1,1);                 Empaquetar la cantidad de barras del esclavo
        pvm_pkint(&NOPV[h],1,1);                Empaquetar la cantidad de barras tipo PV del esclavo
        for(j=1;j<=NBP[h];j++)                  Cargar en ORD las barras a ser resultas por el esclavo
                ORD[j]=*(PART+(NK+1)*h+j);
        pvm_pkint(ORD,NBP[h]+1,1);              Empaquetar el vector ORD
        pvm_pkfloat(V,NK+1,1);                  Empaquetar el valor inicial de las tensiones en barras
        pvm_pkfloat(TH,NK+1,1);                 Empaquetar el valor inicial de las fases de las tensiones en barras
        for(i=1;i<=NBP[h];i++){                 Cargar las potencias necesarias para el esclavo
                PGE[i]=PG[ORD[i]];
                QGE[i]=QG[ORD[i]];
                PLE[i]=PL[ORD[i]];
                QLE[i]=QL[ORD[i]];}
        pvm_pkfloat(PGE,NBP[h]+1,1);            Empaquetar las potencias necesarias para el esclavo
        pvm_pkfloat(QGE,NBP[h]+1,1);
        pvm_pkfloat(PLE,NBP[h]+1,1);
        pvm_pkfloat(QLE,NBP[h]+1,1);
        for (i=1; i<=NBP[h]; i++)               Empaquetar los elementos de la matriz de admitancias
                for (j=1; j<=NK; j++){          necesitados por el esclavo
                        pvm_pkfloat(&G[ORD[i]][j],1,1);
                        pvm_pkfloat(&B[ORD[i]][j],1,1);}
        msg=100;                                Asignar un valor a la etiqueta de tipo de mensaje
        ENVIAR(TID[h], msg);}                   Enviar los datos empaquetados
time(&first);                                   Se comienza la medición del tiempo
for(i=1;i<=NP;i++)                              Inicialización del error
        ERROR[i]=2*CONV;
contin=1;                                       Inicialización de la etiqueta de continuación
while(contin){                                  Mientras no se llegue a un error máximo menor o igual a la tolerancia
        for(i=1;i<=NP;i++){                             Se verifica para cada proceso esclavo
                if(RECIBIR(TID[i],-1)){                         Si se ha recibido resultados del esclavo
                        pvm_upkint(&dimhi,1,1);                         Desempaquetar la dimensión del problema local
                        pvm_upkint(ORDR,dimhi+1,1);                     Desempaquetar a cuales barras corresponden los
                                                                        resultados
                        pvm_upkfloat(VR,dimhi+1,1);                     Desempaquetar las tensiones calculadas
                        pvm_upkfloat(THR,dimhi+1,1);                    Desempaquetar las fases calculadas
                        pvm_upkfloat(&ERROR[i],1,1);                    Desempaquetar el error del proceso esclavo
                        for(j=1;j<=dimhi;j++){                          Cargar las fases y las tensiones recibidas en los
                                V[ORDR[j]]=VR[j];                       vectores globales
                                TH[ORDR[j]]=THR[j];}}}
        max=0;                                  Inicialización del valor del máximo error a cero
        for(i=1;i<=NP;i++)                      Para cada procesador
        if(fabs(ERROR[i])>max)                  Si el error local es mayor al máximo
                max=fabs(ERROR[i]);                     El máximo es el error local
        if(max<= CONV) {                        Si el máximo error es menor o igual que la tolerancia
                for (i=1; i<=NK; i++){                  Se realizará el cálculo del error global
                        P[i]=0;
                        Q[i]=0;
                        for (j=1; j<=NK; j++){
                                P[i]=P[i]+V[i]*V[j]*(G[i][j]*cos(TH[i]-TH[j])+B[i][j]*sin(TH[i]-TH[j]));
```

```
                                        Q[i]=Q[i]+V[i]*V[j]*(G[i][j]*sin(TH[i]-TH[j])-B[i][j]*cos(TH[i]-TH[j]));}}
                conti=1;
                for (i=2; i<=NK; i++){
                        f[conti]=P[i]+PL[i]-PG[i];
                        conti++;}
                for (i=NPV+2; i<=NK; i++){
                        f[conti]=Q[i]+QL[i]-QG[i];
                        conti++;}
                error=MAXIMO(f,conti-1);
                if(error<=CONV)          Si el error global es menor o igual a la tolerancia
                        contin=0; }}             La bandera indica que se llegó a la solución

time(&second);                           Se termina la medición del tiempo
for(i=1; i<=NK; i++)                     Impresión de resultados
        printf("V(%d)=%5.6f\n",i,V[i]);
for(i=1; i<=NK; i++)
        printf("TH(%d)=%5.6f\n",i,TH[i]);
printf("El error=%2.6f\n",error);        `       Impresión del error en la solución
printf("\ntiempo=%5.2f\n",difftime(second,first));   Impresión del tiempo utilizado en la resolución
printf(" %d  iteraciones del Administrador\n",iter1);   Impresión del número de iteraciones realizadas por el proceso
                                                        administrador
for(i=1;i<=NP;i++)                       Eliminar a los programas esclavos
        pvm_kill(TID[i]);
pvm_exit();}                             Terminación de la sesión de PVM
```

FUNCIONES

FORMA_GB

Función que construye las matrices de conductancias y susceptancias

```
void FORMA_GB(void)
{
int             i,j;

A continuacion se forma la matriz GB, representante de Ybarra
for (i=1; i<=NK; i++)
        for (j=1; j<=NK; j++){
        G[i][j]=0.0;
        B[i][j]=0.0;
        if(i==j){
                G[i][j]=YSR[i];
                B[i][j]=YSI[i];}}
for (i=1; i<=NL; i++){
        G[IL[i]][KL[i]]=YR[i];
        G[KL[i]][IL[i]]=YR[i];
        B[IL[i]][KL[i]]=YI[i];
        B[KL[i]][IL[i]=YI[i];}}
```

MULTIPLICACION

Función que multiplica la matriz a por el vec2[] y coloca el resultado en vec3[]

```
void MULTIPLICACION (float vec2[cdim], float vec3[cdim], int dim){

int             i,j;
for (i=1; i<=dim; i++){
        vec3[i]=0.0;
        for (j=1; j<=dim; j++)
        vec3[i]=vec3[i]+a[i][j]*vec2[j];}}
```

MAXIMO
Función que devuelve la norma infinita de un vector

```
float MAXIMO(float vec[cdim], int dim){

int                     i;
                        float aux;
aux=0;
for (i=1; i<=dim; i++)
        if (fabs(vec[i])>aux) aux=fabs(vec[i]);
return (aux);}
```

lfslave1.c
Programa esclavo para el cálculo asíncrono del flujo de potencia de una red eléctrica

```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include "const.h"
# include "/usr/local/pvm3/include/pvm3.h"

#define ENVIAR(tid, msg) pvm_send(tid, msg)        Función de envío de resultados en forma asíncrona

int     mytid;          Números de identificación de la tarea propia
        master,         Número de identificación de la tarea administradora
        msgtype,        Tipo de mensaje
        msg,            Etiqueta de envío de mensajes
        bufid,          Variable auxiliar de información
        info;           Variable de verificación de procesos

void MULTIPLICACION (float vec2[cdim], float vec3[cdim], int dim);   Función que multiplica una matriz por un vector
float H(int i, int j);                  Función que calcula un elemento de la submatriz H del jacobiano
float N(int i, int j);                  Función que calcula un elemento de la submatriz N del jacobiano
float J(int i, int j);                  Función que calcula un elemento de la submatriz J del jacobiano
float L(int i, int j);                  Función que calcula un elemento de la submatriz L del jacobiano

float   G[CNK][CNK],            Matriz de conductancias del problema
        B[CNK][CNK],            Matriz de susceptancias del problema
        V[CNK],                 Tensiones en barras
        TH[CNK],                Ángulos de las tensiones
        THR[CNK],               Vector auxiliar donde se guardan las tensiones calculadas  por los otros esclavos
        VR[CNK],                Vector auxiliar donde se guardan las fases calculadas  por los otros esclavos
        VE[CNK],                Vector de tensiones calculadas a ser enviadas
        THE[CNK],               Vector de fases calculadas a ser enviadas
        PG[CNK],                Potencia activa generada
        QG[CNK],                Potencia reactiva generada
        PL[CNK],                Potencia activa de carga
        QL[CNK];                Potencia reactiva de carga

int     ORD[CNK],ORDR[CNK],     Barras a ser resueltas, en orden de PV a PQ */
        NBP,                    Cantidad de barras a ser resueltas
        NOPQ,                   Cantidad de barras PQ a ser resueltas
        NOPV,                   Cantidad de barras PV a ser resueltas
        contt,dimhi,contit,
        NT,
        ID,
        NK,                     Cantidad total de barras
        TID[20],TIDA[20];       Vector de números de identificación de todos los procesos esclavos

float   errorh,errH,            Error
        a[cdim][cdim],          Matriz jacobiano
        fl[CNK*2],              Vector de errores de las potencias calculadas
        delta[CNK*2],           Variación de tensiones y ángulos
        P[CNK],                 Potencia activa calculada
        Q[CNK];                 Potencia reactiva calculada

int RECIBIR(int tid,int msg){                   Función de recepción asíncrona de resultados
        int bufid;
        int cont=0;
        bufid=0;
        bufid=pvm_nrecv(tid,msg);
        while(pvm_probe(tid,msg)>0){
                bufid=pvm_nrecv(tid,msg);}
        return(bufid);}

void main (void){

int     i,j,            Variables de ciclos
        conti,contj,    Contadores
        verover;        Variable verificadora de solapamiento

mytid=pvm_mytid();      El programa inicia la sesión en PVM
```

```
master=pvm_parent();                    Se obtiene el número de identificación del proceso Administrador

Recibir del Administrador los datos iniciales

bufid=pvm_recv( master, 100 );          Recepción de los datos en el buffer de recepción
pvm_upkint(&NT,1,1);                    Desempaquetar orden del proceso propio
pvm_upkint(TID,NT+1,1);                 Desempaquetar el vector de numero de identificación de los demás esclavos
pvm_upkint(&ID,1,1);                    Desempaquetar el número de identificación propio
pvm_upkint(&NK,1,1);                    Desempaquetar el número de barras de la red global
pvm_upkint(&NBP,1,1);                   Desempaquetar el número de barras a ser procesadas localmente
pvm_upkint(&NOPV,1,1);                  Desempaquetar el número de barras PV a ser resueltas localmente
pvm_upkint(ORD,NBP+1,1);                Desempaquetar el vector que contiene a las barras a ser resueltas localmente
pvm_upkfloat(V,NK+1,1);                 Desempaquetar el vector de tensiones iniciales
pvm_upkfloat(TH,NK+1,1);                Desempaquetar el vector de fases iniciales
pvm_upkfloat(PG,NBP+1,1);               Desempaquetar el vector de potencias activas generadas
pvm_upkfloat(QG,NBP+1,1);               Desempaquetar el vector de potencias reactivas generadas
pvm_upkfloat(PL,NBP+1,1);               Desempaquetar el vector de potencias activas de carga
pvm_upkfloat(QL,NBP+1,1);               Desempaquetar el vector de potencias reactivas de carga
for (i=1; i<=NBP; i++)                  Desempaquetar las matrices de conductancias y susceptancias de la red local
        for (j=1; j<=NK; j++){
                pvm_upkfloat(&G[i][j],1,1);
                pvm_upkfloat(&B[i][j],1,1);}
NOPQ=NBP-NOPV;                          Se calcula el número de barras PQ
for(i=1;i<=NK;i++)                      Inicialización de las fases a cero
        TH[i]=0.0;

for (i=1; i<=NBP; i++){                 Se calculan las potencias activa y reactiva iniciales en cada barra
        P[i]=0;
        Q[i]=0;
        for (j=1; j<=NK; j++){
                P[i]=P[i]+V[ORD[i]]*V[j]*(G[i][j]*cos(TH[ORD[i]]-TH[j])+B[i][j]*sin(TH[ORD[i]]-TH[j]));
                Q[i]=Q[i]+V[ORD[i]]*V[j]*(G[i][j]*sin(TH[ORD[i]]-TH[j])-B[i][j]*cos(TH[ORD[i]]-TH[j]));}}

                Calculo del mismatch local inicial
                Mismatch de potencias activas
conti=1;
for(i=1;i<=NBP;i++){
        fl[conti]=P[i]+PL[i]-PG[i];
        conti++;}

                Mismatch de potencias reactivas
for(i=NOPV+1;i<=NBP;i++){
        fl[conti]=Q[i]+QL[i]-QG[i];
        conti++;}

contit=0;
while(1) {                      Ciclo infinito

        Formación de la matriz jacobiano a[][]
        for (i=1; i<=NBP; i++){         Formación de los elementos de H y de N
                contj=1;
                for (j=1; j<=NBP; j++){
                        a[i][j]=H(i,j);
                        contj++;}
                for(j=NOPV+1;j<=NOPV+NOPQ;j++){
                        a[i][contj]=N(i,j);
                        contj++;}}
        conti=NOPV+1;
        for(i=NBP+1; i<=NBP+NOPQ; i++){         Formación de los elementos de J y de L
                contj=1;
                for (j=1; j<=NBP; j++){
                        a[i][j]=J(conti,j);
                        contj++;}
                for(j=NOPV+1;j<=NOPV+NOPQ;j++){
                        a[i][contj]=L(conti,j);
                        contj++;}
                conti++;}
        inva(NBP+NOPQ,a);                       Inversión del jacobiano
```

```
        Multiplicación de la matriz jacobiano invertido por el vector mismatch, obteniendose las variación de de V y TH en el
        vector delta
        MULTIPLICACION(fl, delta, NBP+NOPQ );

        Actualización de los vectores V[] y TH[] locales
        conti=1;
        for(i=1;i<=NBP;i++){
                TH[ORD[i]]=TH[ORD[i]]-delta[conti];
                conti++;}
        for(i=NOPV+1; i<=NBP; i++){
                V[ORD[i]]=V[ORD[i]]-delta[conti];
                conti++;}

        Recepción de resultados de los demás procesos esclavos
for(i=1;i<=NT;i++){
        if(RECIBIR(TID[i],-1)){                 Si existen mensajes del esclavo dado en el buffer de recepción
                pvm_upkint(&dimhi,1,1);
                pvm_upkint(ORDR,dimhi+1,1);
                pvm_upkfloat(VR,dimhi+1,1);
                pvm_upkfloat(THR,dimhi+1,1);
                pvm_upkfloat(&errH,1,1);
                for(j=1;j<=dimhi;j++){
                        V[ORDR[j]]=VR[j];
                        TH[ORDR[j]]=THR[j];}}}

        for (i=1; i<=NBP; i++){                 Calculo de las potencias activa y reactiva en cada barra
                P[i]=0;
                Q[i]=0;
                for (j=1; j<=NK; j++){
                        P[i]=P[i]+V[ORD[i]]*V[j]*(G[i][j]*cos(TH[ORD[i]]-TH[j])+B[i][j]*sin(TH[ORD[i]]-TH[j]));
                        Q[i]=Q[i]+V[ORD[i]]*V[j]*(G[i][j]*sin(TH[ORD[i]]-TH[j])-B[i][j]*cos(TH[ORD[i]]-TH[j]));}}

        Formación del mismatch local
        conti=1;
        for (i=1; i<=NBP; i++){                 Mismatch de potencias activas
                fl[conti]=P[i]+PL[i]-PG[i];
                conti++;}
        for (i=NOPV+1; i<=NBP; i++){            Mismatch de potencias reactivas
                fl[conti]=Q[i]+QL[i]-QG[i];
                conti++;}

Calculo del máximo error
errorh=0;
for(i=1; i<=NBP+NOPQ; i++)
        if(fabs(fl[i])>errorh)
                errorh=fabs(fl[i]);

enviar los vectores V[], TH[] y el error calculado a todos los esclavos y al Administrador
for(j=1;j<=NBP;j++){                    Cargar los vectores de envío
        VE[j]=V[ORD[j]];
        THE[j]=TH[ORD[j]];}
pvm_initsend(PvmDataDefault);           Inicializar el buffer de envío
pvm_pkint(&NBP,1,1);                    Empaquetar el número de barras resueltas localmente
pvm_pkint(ORD,NBP+1,1);                 Empaquetar el vector de barras calculadas localmente
pvm_pkfloat(VE,NBP+1,1);                Empaquetar el vector de tensiones a ser enviado
pvm_pkfloat(THE,NBP+1,1);               Empaquetar el vector de fases a ser enviado
pvm_pkfloat(&errorh,1,1);               Empaquetar el error
msg=1;                                  Asignar un valor a la etiqueta de mensaje
info=pvm_send(master,ID);               Enviar al Administrador los datos empaquetados
for(i=1;i<=NT;i++){                     Enviar a los demás procesos esclavos los datos empaquetados
        if(i==ID) continue;
        info=pvm_send(TID[i],ID);}}}
```

FUNCIONES

MULTIPLICACION
Función que multiplica la matriz a por el vec2[] y coloca el resultado en vec3[]

```
void MULTIPLICACION (float vec2[cdim], float vec3[cdim], int dim){

int                     i,j;

for (i=1; i<=dim; i++){
        vec3[i]=0.0;
        for (j=1; j<=dim; j++)
                vec3[i]=vec3[i]+a[i][j]*vec2[j];}}
```

H
Función que calcula un elemento de H

```
float H(int i, int j){

float h;

if(i==j)
        h=-Q[i]-B[i][ORD[i]]*V[ORD[i]]*V[ORD[i]];
else
        h=V[ORD[i]]*V[ORD[j]]*(G[i][ORD[j]]*sin(TH[ORD[i]]-TH[ORD[j]])-B[i][ORD[j]]*cos(TH[ORD[i]]-TH[ORD[j]]));
return(h); }
```

N
Función que calcula un elemento de N

```
float N(int i, int j){

float n;

if(i==j)
        n=P[i]/V[ORD[i]]+G[i][ORD[i]]*V[ORD[i]];
else
        n=V[ORD[i]]*(G[i][ORD[j]]*cos(TH[ORD[i]]-TH[ORD[j]])+B[i][ORD[j]]*sin(TH[ORD[i]]-TH[ORD[j]]));
return(n);}
```

J
Función que calcula un elemento de J

```
float J(int i, int j){

float ja;

if(i==j)
        ja=P[i]-G[i][ORD[i]]*V[ORD[i]]*V[ORD[i]];
else
        ja=-V[ORD[i]]*V[ORD[j]]*(G[i][ORD[j]]*cos(TH[ORD[i]]-TH[ORD[j]])+B[i][ORD[j]]*sin(TH[ORD[i]]-
TH[ORD[j]]));
return(ja);}
```

L
Función que calcula un elemento de L

```
float L(int i, int j){

float l;

if(i==j)
        l=Q[i]/V[ORD[i]]-B[i][ORD[i]]*V[ORD[i]];
else
        l=V[ORD[i]]*(G[i][ORD[j]]*sin(TH[ORD[i]]-TH[ORD[j]])-B[i][ORD[j]]*cos(TH[ORD[i]]-TH[ORD[j]]));
return(l);}
```

Referencias Bibliográficas

[1] Aboytes, F., and Sasson, A. M., "A power Systems Decomposition Algorithm",*Proceedings of the IEEE Power Industries computer Application Conference"*, pp. 448-452, 1971.

[2] Alvarado, F. L., "On the Implementation of Parallel Computation by Network Diakoptics", *Special Report of the Electrical Power Research Institute,* EPRI EL 566-SR, pp. 209-221, octubre 1977.

[3] Arrillaga J., Arnold C.P., Harker B.J., *Computer modelling of Electrical Power Systems*, John Wiley & Sons, U.K. 1983.

[4] Barán B., Kaszkurewicz E. y Falcão D.M., "Team Algorithm in Distributed Load Flow Computations", *IEE Proceeding on Generation, Transmission and Distribution*, vol. 142, no. 6, pp. 583-588, noviembre 1995.

[5] Barán B., *Estudio de Algoritmos Combinados Paralelos Asíncronos*. Tesis Doctoral COPPE/UFRJ. Río de Janeiro, Brasil. Octubre 1993.

[6] Barán B., Kaszkurewicz E. y Bhaya A., "Parallel Asynchronous Team Algorithms: Convergence and Performance Analysis".*IEEE Transactions on Parallel & Distributed Systems*. Julio de 1996.

[7] Barán B., Cardozo F., Atlasovich J. y Schaerer C., "Solving the point of Collapse Problem using a Heterogeneus Computer Network". International Conference on Information Systems Analysis and Synthesis. Orlando, EE.UU. Julio de 1996.

[8] Baudet G.M. (1978). Asynchronous iterative methods for multiprocessors, J. ACM, 25(2), pp. 226-244.

[9] Bhaya M., Kaszkurewicz E. y Mota F., "Asynchronous Block-Iterative Methods for Almost Linear Equations". *Linear Algebra and Its Applications*, vol. 155, pp. 487-508, 1991.

[10] Cabling Business Magazine, vol. 6 no. 6, junio 1996.

[11] Carré B.A., "Solution of Load-Flow by Partitioning Systems into Trees", *IEEE Transactions on Power Apparatus and Systems,* vol. PAS-88, pp. 1931-1938, noviembre 1968.

[12] Chazan D. y Miranker W. (1969). Chaotic relaxation. *Linear Algebra Appl. 2, pp. 199-222.*

[13] Ikeda M. y Šiljak D.D., Overlapping decomposition, expansions and contractions of dynamic systems. *Large Scale System 1*, North-Holland Publishing Co., pp.29-38, 1980.

[14] INTEL Corporation, "Intel Microprocessor Quick Reference Guide". Documento disponible en World Wide Web: http :// www. intel. com /pressroom /no_frame/ quickref.htm

[15] Irving M.R. y Sterling M.J.H., "Optical Network Tearing Using Simulated Annealing", *IEEE Proceedings,* vol. 137, no. 1, pp. 69-72, enero 1990.

[16] Jain M.K. y Rao N.D., "A Power System Networks Decomposition for Network Solutions", *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-92, no. 2, pp. 619-625, 1973.

[17] Kaskurewicz E., Bhaya A. y Šiljak D. D. "On the convergence of parallel asynchronous block-iterative computations", *Linear Algebra Appl.*, 131, pp. 139-160, 1990.

[18] Mickle M.H., Vogt W.G. y Colclaser R.G., "Paralel Processing and Optimal Network Decomposition Applied to Load Flow Analysis and Related Problems", *Special Report of the Electrical Power Research Institute*, EPRI EL-566-SR, pp. 171-182, 1977.

[19] Monticelli A. *Fluxo de carga em redes de energía elétrica*. Editora Edgard Blucher Ltda. 1983.

[20] Ogbuobiri, E., Tinney, W.F., and Walker, J.W., "Sparsity Oriented Decomposition for Gaussian Elimination on Matrices", *IEEE Transactions on power Apparatus ans Systems*, vol. PAS-89, no. 1, pp. 141-150, enero 1970

[21] PVM: Parallel Virtual Machine. Documento disponible en World Wide Web: http: //www.epm.ornl.gov/pvm/

[22] Saheh A.O.M. y. Laughton M.A,"Cluster Analysis of Power System Networks for Array Processing Solutions", *IEEE Proceedings*, vol. 132, no. 4, pp. 172-178, july 1985.

[23] Sangiovanni-Vincentelli, A., Chen, L. K., and Chua, L. O., "An Efficient Heuristic Cluster algorthm for Tearing Large-Scale Networks", *IEEE Transactions on Circuits and Systems"*, vol. CAS-89, no. 12, pp. 709-717, diciembre 1977

[24] Sasson A.M., "Decomposition Tecnique Applied to the Nonlinear Programming Load-Flow Method", *IEEE Transactions on Power Apparatus nd Systems*, vol. PAS-89, no. 1, pp. 78-82, enero 1970.

[25] Schaerer C., Atlasovich J., *Flujo de Potencia Eléctrica en torno al Punto Crítico*, Tesis de grado, Facultad de Ingeniería de la Universidad Nacional de Asunción. Mayo 1995.

[26] Sezer M. y Šiljak D.D., "Nested epsilon decompositions of complex systems. IFAC" 9th *World Congress*, Budapest, Hungría.

[27] Sezer M. y Šiljak D.D., "Nested epsilon decompositions and clustering of complex systems", *Automática*, vol. 22, no. 3, pp. 69-72, 1986.

[28] Sezer M. y Šiljak D.D., "Nested epsilon decompositions of linear systems: Weakly coupled and overlapping blocks", *SIAM Journal of Matrix Analysis and Aplications*, 12, pp. 521-533, 1991.

[29] Undrill J.M. y Happ H.H., "Automatic Sectionalization of Power System Networks for Network Solution", *IEEE, Transactions on Power Aparatus and Systems*, vol. PAS-90, no.1, pp. 43-53, enero / febrero 1971

[30] Varga, R. (1962). *Matriz Iterative Analysis*. Prentice Hall, Englewoods Cliffs, N.J.

[31] Vale M.H., Falcão D.M. y Kaszkurewicz E., "Electrical Power Network Decomposition for Parallel Computations". *IEEE Iternational Symposium on Circuits and Systems*-ISCAS 92. San Diego, California, 1992.

[32] Vale, M.H. *Descomposicao de Redes Elétricas para Processamento Paralelo.* Tesis Doctoral COPPE/UFRJ. Río de Janeiro, Brasil. 1994.

[33] Zecevic A.Y. y Siljak D.D., "Balanced Decompositions of Sparse Systems for Parallel Processing", *IEEE Trasactions on Circuits and Systems*, vol. 41, no. 3, pp. 220-233, marzo 1994.