Asociación Argentina de Mecánica Computacional
Mecánica Computacional Vol XXVI, pp.441-454
Sergio A. Elaskar, Elvio A. Pilotta, Germán A. Torres (Eds.)
Córdoba, Argentina, Octubre 2007

UN SOFTWARE PARALELO PARA PROBLEMAS DE OPTIMIZACIÓN DE GRAN TAMAÑO

Juan I. Ardenghi[a], Gustavo E. Vazquez[a] and Nélida B. Brignole[a,b]

[a]*Laboratorio de Investigación y Desarrollo en Computación Científica (LIDeCC)*
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur - Av. Alem 1253 – 8000 - Bahía Blanca – ARGENTINA
e-mail: ardenghi@criba.edu.ar, gvazquez@criba.edu.ar, dybrigno@criba.edu.ar

[b]*Planta Piloto de Ingeniería Química (UNS-CONICET)*
Complejo CRIBABB Cno. La Carrindanga km 7 - 8000 - Bahía Blanca – ARGENTINA
Tel.: 486 1700 - Fax: 486 1600

Palabras Clave: procesamiento paralelo, optimización, programación no lineal, NLP, gradiente espectral

Resumen. El método del gradiente espectral proyectado (SPG) es un método de optimización global no monótono para problemas de programación no lineal de características diferentes a los métodos clásicos de gradiente proyectado. La no monotonicidad y una elección particular de la longitud del paso permiten aprovechar situaciones especiales que se presentan en los problemas, acelerando la convergencia con mínimos costos de almacenamiento de datos.
En diversas publicaciones se analiza el rendimiento del SPG aplicado a diferentes ramas de la ingeniería. Entre sus características más atractivas aparece su bajo costo en operaciones: SPG no calcula Hessianos ni resuelve sistemas lineales, sólo utiliza productos matriz vector y una estrategia de búsqueda lineal no monótona para garantizar convergencia global. Su punto débil se encuentra en la necesidad de muchas búsquedas lineales para obtener un nuevo iterado, y en la necesidad de una buena aproximación del gradiente cuando éste no está disponible. En este caso, el auge del desarrollo en la programación en paralelo hace que este paradigma se presente como un recurso que ofrece una gran oportunidad para superar estos inconvenientes.
Este método, una vez paralelizado, se torna una herramienta muy importante y prometedora para resolver problemas de optimización de gran tamaño. En este trabajo se presenta una nueva versión del SPG que implementa la búsqueda lineal y la evaluación del gradiente en paralelo. El algoritmo presentado implementa una distribución dinámica de las tareas, funcionando eficientemente sobre sistemas distribuidos heterogéneos. La implementación fue realizada apuntando a resolver problemas de diseño y condiciones operativas de plantas de procesos industriales. Se reportan resultados numéricos.

1 INTRODUCCION

Los problemas de optimización representan una parte muy importante dentro de lo que se refiere a programación matemática y computación científica. El desarrollo de métodos numéricos para resolver problemas con y sin restricciones se ha incrementado en forma considerable dada su aplicabilidad en gran cantidad de disciplinas científicas. Una de estas áreas es la de sistemas de procesos, en la que los métodos numéricos de optimización han avanzado de tal manera que han pasado a ser considerados más como una tecnología que como una metodología de interés académico.

Por otro lado, junto a las nuevas técnicas en métodos numéricos, han aparecido problemas de aplicación más complejos. Entonces, aquellos métodos que son globalmente convergentes y sus tiempos de ejecución son reducidos son los que se han tornado más populares. Una de las técnicas más importantes para poder reducir tiempos de ejecución es el procesamiento paralelo. Este ha sido facilitado por el desarrollo de procesadores paralelos masivos (MPPs) y el uso de computación distribuida. Luego estos procesos en paralelo pueden ser implementados en una única máquina paralela o en una configuración distribuida de *workstations* homogénea o heterogénea.

En este trabajo se introduce el paralelismo en una clase de algoritmos de optimización de bajo costo llamado gradiente espectral proyectado (SPG). Estos métodos aplicados a problemas sin restricciones y con restricciones convexas han ganado cierta popularidad debido a su convergencia global con requerimientos computacionales mínimos.

Los métodos SPG, introducidos en Raydan (1997) utilizan direcciones de descenso, y la forma de escoger la longitud del paso requiere menos trabajo computacional y acelera notablemente la velocidad de convergencia. Esta variante combinada con el esquema de búsqueda lineal no monótona desarrollada por Grippo et al (1986) aseguran la convergencia global. La comparación con otras técnicas de optimización numérica de bajo costo ha favorecido a SPG en gran cantidad de problemas de gran escala (Birgin et.al. 2000; Birgin et. al. 2001; Bello and Raydan 2004; Luengo and Raydan 2003). Estos métodos también han sido aplicados exitosamente en diferentes problemas industriales (Birgin et.al 1999a; Birgin et. al 1999b; Birgin and Evtushenko 1998; Castillo et.al. 2000; Mulato et. al 2000; Glunt et.al. 1993), y se ha demostrado que aparentan ser superiores en su desempeño a los métodos de optimización existentes en los paquetes clásicos utilizados en ingeniería química (Domancich et. al, 2004). Una revisión de estos métodos mostrando la importancia del uso de una búsqueda lineal no monótona se puede encontrar en Fletcher (2001).

Una versión de SPG fue desarrollada en Molina et.al (2000) para correr sobre máquinas paralelas. En dichos trabajos se desarrolla una paralelización de las operaciones de SPG (productos escalares y productos matriz-vector) aplicando un precondicionador, utilizando versiones de grano fino y grueso. En los problemas testeados se cuenta con el gradiente analítico y la sucesión de iterados por las versiones paralelas son los mismos que los de la versión secuencial, sólo que fueron calculados en forma paralela. Este trabajo está orientado con otro objetivo. En los problemas de diseño y condiciones operativas de plantas de procesos industriales en general no se cuenta con el gradiente en forma analítica. Luego, dado que SPG tiene mínimos requerimientos de memoria y de operaciones aritméticas, el tiempo total de cada iteración depende de dos puntos clave: la búsqueda lineal no monótona y la aproximación del gradiente dado que este no está disponible. Estos puntos son claramente factibles de paralelizar. La versión paralela que vamos a presentar busca impactar sobre los iterados de SPG, es decir, que la distribución de tareas genere una sucesión de iterados

alternativa a la que genera la versión secuencial buscando hacerlo aun más competitivo, pero manteniendo la robustez de convergencia que posee el método SPG secuencial. En este trabajo se presenta una versión paralela de grano grueso del algoritmo SPG para correr sobre sistemas distribuidos homogéneos o heterogéneos. Para el análisis de desempeño se utilizaron 28 versiones de un problema académico. En esta clase de problemas los gradientes analíticos y la función objetivo son conocidos. Dado que la implementación futura de este algoritmo es sobre problemas donde no se cuenta con expresiones analíticas de la función objetivo y/o de los gradientes, lo que se busca analizar es el desempeño del algoritmo paralelo en condiciones de desconocimiento absoluto de esta información. Luego las expresiones analíticas conocidas se utilizan sólo como referencia en la comparación cuando se examina el posible impacto del procesamiento paralelo en los tiempos de ejecución.

2 SOFTWARE PARALELO PARA SISTEMAS DISTRIBUIDOS HETEROGÉNEOS

La computación en paralelo permite resolver un problema de gran tamaño resolviendo varias instancias más pequeñas simultáneamente. Esto permite un ahorro importante en lo que se refiere a tiempos de ejecución. Para el desarrollo de software paralelo es necesario un software de pasaje de mensajes que permita establecer comunicaciones entre los diferentes procesadores. Entre los programas más populares de pasaje de mensajes para realizar este tipo de implementaciones son MPI (Message Passing Interface) y PVM (Parallel Virtual Machine)

MPI tiene muy buen rendimiento en máquinas multiprocesadores grandes, contiene un conjunto muy rico de funciones para comunicaciones, y alta performance en estas (MPI fórum 1994). Pero cuando las aplicaciones deben ser ejecutadas sobre estaciones de trabajo heterogéneas PVM aparece como más conveniente. Tiene muy buena interoperabilidad entre diferentes tipos de huéspedes y permite el desarrollo de aplicaciones tolerantes a fallas que pueden continuar a pesar de errores en algún huésped o tarea (Geist et.al. 1994). El concepto de máquina virtual que implementa PVM provee una base para la heterogeneidad y portabilidad. Comparaciones más detalladas entre PVM y MPI pueden encontrarse en Geist et. al (1996) y Gropp and Lusk (2002). El algoritmo paralelo presentado en este trabajo fue desarrollado para correr bajo entornos distribuidos tanto homogéneos como heterogéneos, luego fue implementado en PVM 3.4. En la sección 4 se describen los detalles de este desarrollo.

3 EL ALGORITMO SPG

El algoritmo del Gradiente Espectral Proyectado (SPG) fue introducido por Raydan (1993, 1997) y se aplica a problemas de la forma

$$Min\ f(x)\ \ sujeto\ a\ x \in S,$$

donde S es un conjunto convexo y cerrado en $\mathbb{R}^n$.

Este método utiliza direcciones de descenso, pero, en contraste con el clásico método de Cauchy (1947) la elección del paso se realiza de una manera diferente. La forma de escoger la longitud del paso, originalmente presentada por Barzilai and Borwein (1988) y descripta por Raydan, (1993), requiere menos trabajo computacional y acelera notablemente la velocidad de convergencia. Esta variante combinada con el esquema de búsqueda lineal no monótona desarrollada por Grippo et al (1986) aseguran la convergencia global del SPG, es decir, la convergencia a un punto estacionario desde cualquier punto inicial.

El algoritmo se describe en el siguiente esquema:

1- k=0
2- Chequear convergencia
3- Determinar una dirección de descenso.
4- Determinar la longitud del paso (paso espectral).
5- Búsqueda lineal no monótona
6- k = k+1, ir a 2

Paso 2- Si se define P(z) como la proyección ortogonal de *z* en *S*, el algoritmo cicla mientras $\|P(x_k + \lambda_k d_k) - x_k\| > tolerancia$. $\lambda_0 = 1$ y $d_0 = -\nabla f(x_0)$.

Paso 3- Como dirección de descenso se utiliza $d_k = -g_k$ donde $g_k = \nabla f(x_k)$. Si este gradiente no se conoce explícitamente se aproxima por algún método.

Paso 4- El paso espectral se calcula $\lambda_k = \frac{s_{k-1}^t s_{k-1}}{s_{k-1}^t y_{k-1}}$ donde $s_k = x_{k+1} - x_k$ e $y_{k-1} = g_k - g_{k-1}$. Esta elección del paso busca aproximar los autovalores de la matriz Hessiana generando iteraciones no monótonas.

Paso 5- La condición de búsqueda lineal que debe satisfacerse es

$$f(x_k + \lambda_k d_k) \leq \max_{0 \leq j \leq \min(k,M)} \{f_{k-j}\} - \gamma \lambda_k g_k^T g_k \qquad (1)$$

donde *M* y γ son parámetros del algoritmo. Esta búsqueda lineal es no monótona y garantiza la convergencia global sin desaprovechar las bondades del paso espectral. Sobre la convergencia global del método puede consultarse en Raydan, (1997).

Este algoritmo tiene requerimientos mínimos de memoria y no exige decrecimiento en la función objetivo, lo que permite explotar al máximo sus características de convergencia local. Cuando no se realizan búsquedas lineales cada iteración requiere solo O(n) operaciones de punto flotante y una evaluación del gradiente (Raydan 1997). En los problemas de aplicación que nos interesan, el gradiente no se conoce en forma explícita luego es necesaria una buena aproximación. El tiempo total de cada iteración queda determinado por la suma de tiempo que insumen esta aproximación del gradiente y la búsqueda lineal no monótona.

4 EL ALGORITMO SPG EN PARALELO

Los problemas de optimización que se busca resolver con este algoritmo son problemas cuyo gradiente no siempre se conoce en forma explícita. Luego es necesario aproximar este gradiente de alguna manera. Obtener una buena aproximación representa un consumo extra de tiempo de cómputo. Por otro lado, el tiempo que insume cada búsqueda lineal en cada iteración es variable e impredecible, dependiendo de las características de cada problema y del punto que se está analizando. El tiempo total de cada iteración queda entonces determinado por la suma de tiempo que insumen esta aproximación del gradiente y la búsqueda lineal no monótona. Como se explica en la sección siguiente, estos dos puntos son claramente paralelizables. Esto da lugar a un diseño alternativo de SPG en paralelo.

4.1 Evaluación del gradiente

Existen varias formas de aproximar las derivadas parciales. Una de ellas es la fórmula de diferencias finitas centradas:

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i} + O(h_i^2), \; i = 1, \ldots, n \qquad (2)$$

donde h_i es un valor pequeño, e_i es el vector canónico en la dirección *i*, *n* es la dimensión del vector *x* y el último término representa el error en la aproximación, que es del orden de h_i^2.

Para más detalles ver Dennis and Schnabel (1983)

Para cada derivada parcial es necesario calcular la función en los puntos$(x + h_i e_i)$ y $(x - h_i e_i)$ por lo que calcular la derivada con esta fórmula insume *2n* evaluaciones de función. Comparando con el esquema de aproximación de diferencias finitas hacia adelante, esta fórmula tiene la desventaja de duplicar el número de evaluaciones de función, pero da una aproximación más precisa para un h_i dado. Debido a que SPG usa fuertemente la dirección del gradiente, nosotros escogemos esta aproximación para garantizar exactitud a pesar de *n*-1 evaluaciones de funciones adicionales.

Las evaluaciones involucradas en el cálculo de la derivada parcial *i* son independientes de las necesarias para calcular la derivada parcial *j*, luego es posible calcularlas en forma simultánea. En este trabajo se implementa esta evaluación simultánea de las derivadas parciales para agilizar la obtención de la aproximación del gradiente. Se utiliza un esquema MASTER/SLAVE donde a cada nodo SLAVE el nodo que hace de MASTER le asigna una componente para que calcule la derivada parcial. Cuando el SLAVE termina, envía el valor al MASTER y este le asigna una nueva componente a calcular, hasta completar las *n* derivadas parciales. Esta distribución dinámica permite evitar que queden procesadores ociosos cuando aún haya cálculos por hacer.

4.2 Búsqueda lineal

Como se expuso más arriba, la cantidad de búsquedas lineales necesarias en la iteración k dependerá de cómo es la función involucrada en un entorno del punto x_k. La situación ideal es que el paso espectral original sea aceptado. En caso de no ser así, dicho paso se reduce hasta que la búsqueda lineal se satisface. Para pasos suficientemente chicos esta búsqueda lineal se satisface (Ver Raydan, 1997), pero para alcanzar dicho paso pueden llegar a ser necesarias muchas iteraciones de búsqueda. Por otro lado, pasos muy chicos afectan la velocidad de convergencia del método. En función de esta dificultad se implementa una variante.

Siguiendo la idea básica establecida en Vázquez y Brignole (1999) y Vazquez et.al. (2000), si p es la cantidad de procesadores disponibles, se puede evaluar simultáneamente $f(x_k + \frac{n}{p+1}\lambda_k d_k)$ con $n = 1...p$, es decir, se toman p puntos (igualmente espaciados en este caso) del intervalo $[x_k, x_k + \lambda_k d_k]$ y se evalúa f en cada uno de esos puntos. Si ninguno satisface la condición de búsqueda lineal., entonces se subdivide el intervalo $\left[x_k, x_k + \frac{\lambda_k}{p+1} d_k\right]$ en p puntos y se vuelve a aplicar el procedimiento. Si aún no es satisfecha la búsqueda lineal, se repite el procedimiento en el intervalo $\left[x_k, x_k + \frac{\lambda_k}{(p+1)^2} d_k\right]$ y así sucesivamente hasta alcanzar un punto exitoso.

Es necesario establecer un criterio para determinar en qué momento se interrumpe esta búsqueda y se sigue adelante con las iteraciones de SPG.

Tiempo mínimo

Un primer criterio es el de quedarse con el primer punto que satisface la búsqueda lineal. Esto minimiza el tiempo en que se realiza esta búsqueda. Al primer éxito se sigue adelante. La búsqueda lineal en paralelo utilizando este criterio la notamos LS1. La desventaja que este criterio presenta es que, teniendo un punto que satisface esta búsqueda lineal, ¿No podría haber otro que también sea exitoso pero que su longitud sea mayor? Si fuera así solo bastaría

esperar un poquito más a que lleguen las respuestas de los restantes procesadores y elegir el punto exitoso de mayor longitud. De hecho, cuanto más chica es la longitud del paso, más posibilidades tiene de satisfacer la condición (1).

Máxima longitud

Esperar todas las respuestas, por más que ya se cuente con algunas exitosas, representa un segundo criterio de continuación. Este criterio permite asegurarse la longitud de paso más larga, pero podría presentar cierta ineficiencia dependiendo de cómo se implemente. Supongamos que los procesadores se numeran según la longitud de paso a evaluar. Entonces al procesador 1 le toca el paso de mayor longitud, al procesador 2 el que sigue en longitud, y así hasta el procesador p. Si se recibe un éxito en el procesador 4 y fracasos en los procesadores 1, 2 y 3, entonces no tiene sentido esperar la llegada de todas las respuestas ya que no es posible obtener un éxito de longitud mayor. Una estructura que lleve un registro de lo recibido determinaría si se sigue esperando o se continúa con la búsqueda lineal. La implementación de esta versión en paralelo la notamos como LS2.

4.3 Convergencia del algoritmo paralelo

La convergencia global del algoritmo SPG está garantizada por la búsqueda lineal no monótona. Esta viene dada por el siguiente resultado demostrado en Grippo et al.(1986):

TEOREMA 1: *Sea $\{x_k\}$ una sucesión definida por*

$$x_{k+1} = x_k + \alpha_k d_k, \qquad d_k \neq 0$$

Sea $a>0$, $\sigma\in(0,1)$, $\gamma\in(0,1)$ y sea M un entero no negativo. Se asume que

i) $\Omega_0=\{ x: f(x) \leq f(x_0)\}$ *es un conjunto compacto*

ii) existen dos números positivos c_1 y c_2 tales que :

$$g_k^T d_k \leq -c_1 ||g_k||^2$$
$$||d_k|| \leq c_2 ||g_k||$$

iii) $\alpha_k = \sigma^{h_k}$ *a donde h_k es un entero no negativo para el cual*

$$f(x_k + \sigma^{h_i} a d_k) \leq F_{max} + \gamma \sigma^{h_i} a g_k^T g_k$$

siendo $F_{max}=max\{ f(x_k), f(x_{k-1}), ..., f(x_{k-M})\}$

Entonces

i) $\lim_{k\to\infty} \|g_k\| = 0$,

ii) ningún punto límite de $\{x_k\}$ es un máximo local de f,

iii) si el número de puntos estacionarios de f en Ω_0 es finito, entonces la sucesión $\{x_k\}$ converge.

La existencia de este resultado permite establecer un teorema similar para la convergencia del algoritmo SPG. El siguiente resultado fue establecido en Raydan (1997):

TEOREMA 2: *Sea $\Omega_0=\{ x: f(x) \leq f(x_0)\}$ un conjunto compacto. Sea $f: \Re^n \to \Re$ una función continuamente diferenciable en un entorno N de Ω_0. Sea $\{x_k\}$ la sucesión generada por el algoritmo SPG. Entonces o $g(x_j)=0$ para algún índice j, o valen las siguientes propiedades:*

i) $\lim_{k\to\infty} \|g_k\| = 0$,

ii) ningún punto límite de $\{x_k\}$es un máximo local de f,

iii) si el número de puntos estacionarios de f en Ω_0 es finito, entonces la sucesión $\{x_k\}$ converge.

A partir de estos teoremas hemos podido establecer la convergencia de nuestro algoritmo paralelo. En otras palabras, bajo las condiciones del teorema 1, el algoritmo SPG con las búsquedas lineales en paralelo convergerá a un punto estacionario.
En base a este concepto, nosotros hemos podido definir el siguiente corolario:

COROLARIO: *Sea $\Omega_0=\{ x: f(x) \leq f(x_0)\}$ un conjunto compacto. Sea f: $\Re^n \rightarrow \Re$ una función continuamente diferenciable en un entorno N de Ω_0. Sea $\{x_k\}$ la sucesión generada por la versión paralela del algoritmo SPG. Entonces o $g(x_j)=0$ para algún índice j, o valen las siguientes propiedades:*

i) $\lim_{k\to\infty} \|g_k\| = 0$,

ii) ningún punto límite de $\{x_k\}$ es un máximo local de f,

iii) si el número de puntos estacionarios de f en Ω_0 es finito, entonces la sucesión $\{x_k\}$ converge.

Demostración: El algoritmo utiliza como dirección $d_k = -\tilde{g}_k$ donde $\tilde{g}_k$ es la aproximación del gradiente con diferencias finitas centradas. De lo descripto en la sección 4.1 se desprende que para $c_1 \geq 0$ es

$$\|g_k - \tilde{g}_k\| \leq c_1 h^2$$

Luego, sin perder generalidad, se puede elegir *h* suficientemente chico tal que

$$\|g_k - \tilde{g}_k\| \leq \frac{1}{2}\|g_k\|$$

Considerando que

$$-\tilde{g}_k^T g_k + \|g_k\|^2 = (-\tilde{g}_k + g_k)^T g_k \leq \|-\tilde{g}_k + g_k\|\|g_k\| \leq \frac{1}{2}\|g_k\|^2$$

Entonces

$$d_k^T g_k \leq \frac{1}{2}\|g_k\|^2 - \|g_k\|^2 \quad \Rightarrow \quad d_k^T g_k \leq -\frac{1}{2}\|g_k\|^2$$

Por otro lado

$$\|d_k\| = \|\tilde{g}_k\| \leq \|\tilde{g}_k - g_k\| + \|g_k\| \leq \frac{1}{2}\|g_k\| + \|g_k\| \leq \frac{3}{2}\|g_k\|$$

Luego se satisface la hipótesis ii) del teorema 1.

Por otro lado, en la búsqueda lineal no monótona del algoritmo en paralelo se obtienen iterados que satisfacen

$$f(x_{k+1}) \leq f_{max} - \tilde{\gamma}\sigma^{h_i}\lambda_k \tilde{g}_k^T \tilde{g}_k$$

Eligiendo adecuadamente el parámetro $\tilde{\gamma}$, los iterados obtenidos por el algoritmo paralelo satisfacen la hipótesis iii) del teorema 1. En efecto, si tomamos $\tilde{\gamma} = \gamma \frac{g_k^T \tilde{g}_k}{\tilde{g}_k^T \tilde{g}_k}$, para un h suficientemente chico es $\frac{g_k^T \tilde{g}_k}{\tilde{g}_k^T \tilde{g}_k} \leq 1 + \epsilon$ con $0 < \epsilon \ll 1$. Luego $\tilde{\gamma} < 1$ y

$$f(x_{k+1}) \leq f_{max} - \tilde{\gamma}\sigma^{h_i}\lambda_k \tilde{g}_k^T \tilde{g}_k = f_{max} - \gamma\sigma^{h_i}\lambda_k g_k^T \tilde{g}_k = f_{max} + \gamma\sigma^{h_i}\lambda_k g_k^T d_k$$

Al cumplirse las condiciones del teorema 1, éste se aplica, asegurando la convergencia global de SPG en paralelo. ■

5 ANALISIS DE DESEMPEÑO

Se analizaron 28 problemas de locación basados en aquellos propuestos en Birgin et al (2000) para testear el comportamiento de SPG secuencial. Dependiendo de los parámetros, este puede ser un problema con gran número de variables y restricciones. La descripción es la siguiente: Se considera una malla rectangular de puntos en $\Re^2$. Esta malla es el espacio en que un conjunto de plantas de producción (representado por polígonos) serán emplazadas. De antemano, se define un área rectangular reservada, donde, en principio, nada puede ser construido. Este área reservada contendrá una planta de generación de energía para suministrar a las plantas de producción. En cada punto restante de la malla (excluyendo el región central) una de estas plantas (representado por un polígono) será edificada con una probabilidad definida de antemano. Para transmitir energía desde la central de energía a las plantas de producción, una torre en cada planta y una torre en el interior de la región central deber ser construido. El objetivo del problema es determinar la colocación de estas torres de forma tal que se minimice la suma de las distancias desde las torres de cada una de las plantas a la torre central.

Entonces, dado un conjunto de *npol* polígonos disjuntos $P_1, P_2, \ldots, P_{npol}$ en $\Re^2$, queremos encontrar el punto $z^1 \in P_1$ que minimiza la distancia a los restantes polígonos. El problema a resolver es

$$\min_{z^i,\, i=1\ldots npol} \sum_{i=2}^{npol} \left\| z^i - z^1 \right\|_2$$

$$\text{sujeto a } z^i \in P_i,\ i = 1,\ldots, npol$$

El problema tiene 2×*npol* variables y el número de restricciones es $\sum_{i=1}^{npol} v_i$, donde v_i es el número de vértices del polígono P_i. Una descripción más detallada del problema se puede encontrar en el trabajo de Birgin et.al. (2000).

Como fue descripto en la sección 4, el diseño de este algoritmo fue realizado con una distribución dinámica de las tareas haciéndolo eficiente para sistemas distribuidos heterogéneos. De todos modos, por razones prácticas, las experiencias numéricas fueron realizadas en un cluster homogéneo. Este estaba conformado por 8 PC Pentium 4 de 3Ghz conectadas por una red Ethernet de 1 Gb. Las *workstations* utilizaron GNU/LINUX con distribución *Gentoo* y se utlizó el compilador g77 de LINUX.

Como se menciona en la sección 3, el algoritmo SPG utiliza fuertemente el gradiente de la función objetivo, pero en los problemas de ingeniería que se buscan resolver no se cuenta con dicho gradiente en forma explícita.

La tabla 1 compara el desempeño de SPG en 28 problemas utilizando el gradiente exacto y una aproximación de este por diferencias finitas centradas como se describe en (4.1)

			Valor óptimo de f(x)		Nº de iteraciones		Tiempo CPU (seg)	
Prob Nº	Poligonos	Vértices	Dif. Finitas	G. Exacto	Dif. Finitas	G. Exacto	Dif. Finitas	G. Exacto
1	98	399	1,906590085	1,906590085	16	15	0,05	0
2	95	778	1,817209646	1,817209646	15	14	0,03	0
3	93	1152	1,910986659	1,910986659	15	14	0,03	0
4	173	705	1,970172556	1,970172556	27	23	0,17	0,02
5	178	1483	1,954646798	1,954646798	21	20	0,16	0,02
6	174	2219	1,984684449	1,984684449	13	12	0,14	0,02
7	259	1057	1,964057056	1,964057056	13	12	0,17	0,02
8	262	2161	2,003345163	2,003345163	22	21	0,36	0,02
9	256	3210	2,022382107	2,022382107	22	21	0,36	0,02
10	364	1448	1,973318845	1,973318845	27	23	0,72	0,02
11	369	2965	1,944828076	1,944828076	19	18	0,61	0,02
12	362	4358	1,972744433	1,972744433	22	21	0,67	0,03
13	457	1821	1,956266279	1,956266279	14	13	0,66	0,02
14	481	3852	2,001664609	2,001664609	18	17	0,95	0,03
15	456	5468	2,01745534	2,01745534	15	14	0,72	0,03
16	939	3787	1,330872584	1,330872584	19	18	4,95	0,03
17	928	7529	1,281374181	1,281374181	25	23	4,69	0,06
18	941	11405	1,27050022	1,27050022	23	21	4,45	0,09
19	1873	7528	1,28602419	1,28602419	34	33	26,83	0,09
20	1851	14978	1,31147726	1,31147726	27	26	20,97	0,16
21	1848	22327	1,329438743	1,329438743	24	23	18,53	0,17
22	2851	11459	1,324575329	1,324575329	53	41	117,75	0,27
23	2821	22805	1,339371482	1,339371482	47	40	73,39	0,34
24	2854	34319	1,319195315	1,319195315	51	38	71,58	0,44
25	3830	15350	1,308253163	1,308253163	30	29	98,73	0,17
26	3794	30654	1,321296006	1,327486712	31	27	90,47	0,31
27	3853	46454	1,32748672	1,331917699	47	38	130,3	0,58
28	4746	18987	1,301917699	1,308752708	30	26	136,92	0,22

Tabla 1: Comparación de SPG con gradiente exacto y con diferencias finitas

Notando F_{DF} y F_{GE} los valores óptimos de F(x) con diferencias finitas y gradiente exacto, e It_{DF} e It_{GE} las iteraciones respectivas, la siguiente tabla muestra algunos valores promedio.

$\lvert F_{DF} - F_{GE} \rvert$	$\lvert It_{DF} - It_{GE} \rvert$	T. de CPU con G.E (Seg)	T. de CPU con Dif. Fin. (Seg)
0.0006	2.8214	0.11428	28.76285

Tabla 2: Valores promedio.

Como se puede observar, los valores óptimos de SPG con diferencias finitas así como las iteraciones requeridas no difieren en gran medida con los de SPG con gradiente exacto. Esto es un indicador del buen desempeño de la aproximación utilizada. En contraste se puede observar que se produce un incremento notable en los tiempos de ejecución. Este es el costo

de obtener una buena aproximación del gradiente, fundamental para el buen desempeño del algoritmo principal.

Como fue mencionado en la sección 4, el tiempo de CPU puede ser reducido a través de las versiones paralelas diseñadas. De este modo se obtienen buenas aproximaciones del gradiente en tiempos de ejecución más razonables.

Para medir el impacto del paralelismo se utiliza el factor de *speed up*. Este se define como

Speed-up = tiempo secuencial / tiempo en paralelo

para un sistema distribuido homogéneo. El algoritmo paralelo también puede trabajar sobre un sistema distribuido heterogéneo, pero el speed-up debería medirse utilizando una fórmula ponderada (Vazquez et. al 2000)

El porcentaje de eficiencia es

Eficiencia = (speed-up / número de procesadores)*100

Las siguientes tablas muestran los valores promedio del algoritmo sobre un sistema distribuido con ambas versiones de LS.

Nº Proc.	Tiempo Promedio (Seg)	Speed-up	Eficiencia
1	28.7746	-----	-----
2	19.759	1,4562	72.81 %
4	11.200	2.5689	64.22 %
8	6.0524	4.7542	59.42 %

Tabla 3: Promedio de tiempos de SPG en paralelo con LS1

Nº Proc.	Tiempo Promedio (Seg)	Speed -up	Eficiencia
1	28.7746	----	----
2	14.6362	1.5121	75.60 %
4	7.3649	2.6469	66.17 %
8	3.7319	4.8237	60.29 %

Tabla 4: Promedio de tiempos de SPG en paralelo con LS2

Como se puede observar en las tablas 3 y 4, la eficiencia es alta en ambas versiones, mostrándose una leve superioridad en la versión con LS2. La probabilidad de edificación citada en la descripción del problema introduce valores aleatorios. Luego cada problema representa una clase de problemas, y en cada una de ellas se pueden observar distintos desempeños del algoritmo.

La tabla 5 muestra el detalle de la performance del algoritmo con la búsqueda lineal LS2. La búsqueda lineal en paralelo disminuye la cantidad de búsquedas por llamadas, pero no reduce la cantidad de llamadas. Tomemos como ejemplo el problema 22. La versión secuencial hace dos llamadas a LS y realiza 5 búsquedas en la primera llamada y 6 en la segunda. Si T_B es el tiempo de una búsqueda entonces el tiempo acumulado T_A en las dos llamadas es $T_A = 11*T_B$, como se muestra en la figura 1.

Resolviendo el mismo problema en paralelo con 4 procesadores se aprecia en la tabla que las llamadas son dos, pero el esquema de búsqueda es el que se muestra en la figura 2.

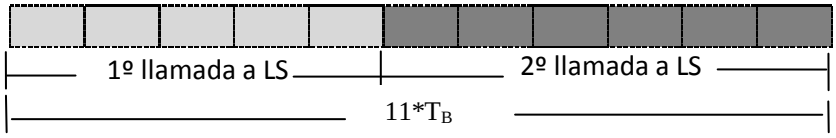

Figura 1

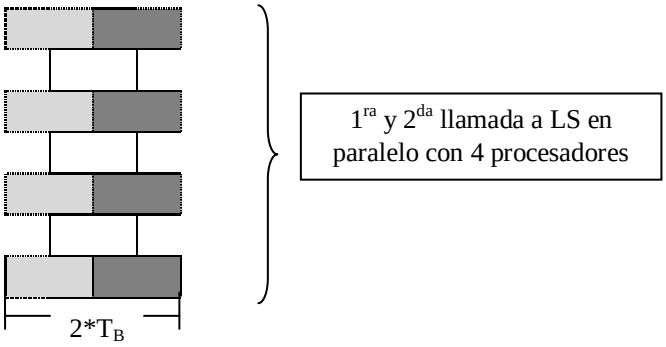

Figura 2

	Tiempo CPU (segundos)				% de eficiencia			# de búsquedas lineales			
Prob Nº	1 proc.	2 proc	4 proc.	8 proc.	2 proc	4 proc.	8 proc.	1 proc.	2 proc	4 proc.	8 proc.
1	0,0552741	0,05174	0,057113	0,056222	53,419	24,195	12,28924	0	0	0	0
2	0,0364686	0,03574	0,037813	0,036729	51,0238	24,111	12,41122	0	0	0	0
3	0,0397974	0,03218	0,032308	0,032704	61,831	30,796	15,2112	0	0	0	0
4	0,1728222	0,12098	0,114927	0,108878	71,4286	37,594	19,84127	3	1	1	1
5	0,1600538	0,16353	0,167283	0,165952	48,9369	23,92	12,05573	0	0	0	0
6	0,143413	0,14881	0,14607	0,146181	48,1873	24,545	12,2633	0	0	0	0
7	0,1735694	0,15621	0,138855	0,253117	55,5556	31,25	8,571601	0	0	0	0
8	0,3667017	0,23465	0,12628	0,081	78,138	72,597	56,58976	0	0	0	0
9	0,3613521	0,24327	0,12614	0,081	74,2698	71,617	55,76421	0	0	0	0
10	0,7279151	0,64925	0,580116	0,547397	56,0584	31,369	16,62218	4	3	3	3
11	0,6101846	0,48848	0,2295	0,10675	62,4575	66,469	71,45019	0	0	0	0
12	0,6788229	0,53648	0,2514	0,11725	63,2664	67,504	72,36918	0	0	0	0
13	0,6644056	0,52864	0,24765	0,1155	62,841	67,071	71,90536	0	0	0	0
14	0,9594916	0,86354	0,767594	0,671644	55,5556	31,25	17,85714	0	0	0	0
15	0,7260771	0,65347	0,580864	0,508254	55,5556	31,25	17,85714	1	1	1	1
16	4,9655123	2,96775	1,486303	0,745458	83,6578	83,521	83,26276	9	4	3	3
17	4,712604	2,81611	1,411264	0,70842	83,6721	83,482	83,15348	6	4	4	4
18	4,470767	2,67107	1,336544	0,668741	83,6887	83,626	83,56686	1	1	1	1
19	26,836885	13,3975	6,704126	3,357324	100,156	100,08	99,91919	16	5	3	3
20	20,982605	12,5824	6,296334	3,153141	83,381	83,313	83,18136	6	4	3	3
21	18,534271	11,1201	5,562999	2,784386	83,3368	83,293	83,20627	3	2	2	2
22	117,79718	58,8393	29,42986	14,7252	100,101	100,07	99,99627	11	5	2	2
23	73,397326	36,6663	18,34638	9,182683	100,088	100,02	99,9127	13	5	3	2
24	71,587296	35,7591	17,90156	8,972528	100,097	99,974	99,73122	24	9	7	7
25	98,77476	49,3237	24,68451	12,36457	100,129	100,04	99,85664	27	9	7	7
26	90,4796	45,1879	22,6177	11,33255	100,115	100,01	99,80061	32	9	8	8
27	130,34967	65,1209	32,59018	16,32022	100,083	99,992	99,83752	17	8	6	5
28	136,92513	68,4533	34,24812	17,14203	100,014	99,951	99,84605	10	6	5	4

Tabla 5: SPG en paralelo con LS2

Al encontrar entre los cuatro puntos que se evalúan en paralelo uno que satisface la búsqueda lineal entonces la búsqueda finaliza. Si llamamos T_C el tiempo de comunicación entre procesadores, entonces el tiempo acumulado de búsqueda lineal en paralelo para el problema 22 es

$$T_A = 2*T_B + T_C$$

La eficiencia que introduce la búsqueda en paralelo dependerá de la cantidad de búsquedas que se realizan por llamada. Como contraste al problema nº 22 podemos ver el problema nº 14 donde no hay llamadas a LS, o el problema nº 10 donde el tiempo acumulado de LS con un procesador es $T_A = 4*T_B$ y en paralelo es $T_A = 3*T_B + T_C$. En estos casos la eficiencia es la que aporta el paralelismo en el cálculo del gradiente.

6 CONCLUSIONES

El método SPG fue adaptado para correr eficientemente bajo cualquier sistema distribuido, respondiendo a las demandas que presentan los problemas de gran tamaño donde no se conocen los gradientes en forma analítica. Como muestran los resultados numéricos, la aproximación del gradiente utilizada no hace mella en los valores de la aproximación obtenida, pero multiplica los tiempos de ejecución. Estos tiempos tienen una reducción notable en las versiones paralelas. El rendimiento de estas versiones muestra un alto promedio de eficiencia con ambas versiones de búsqueda lineal, notándose una tendencia a aumentar en la medida que aumenta el número de variables Como fue desarrollado en la sección anterior, la diferencia de eficiencia se debe a las diferentes demandas de búsqueda lineal propia de cada problema. En aquellos donde no hay gran demanda el paralelismo solo aporta en la evaluación del gradiente que está presente en todos los problemas y en cada iteración. El tiempo de búsqueda lineal se reduce pero no en una proporción constante como lo es la de la aproximación del gradiente, dado que la búsqueda lineal en paralelo no es una paralelización de la búsqueda secuencial sino que se realiza en forma diferente. Luego aquellos problemas en los que la versión secuencial realizaba muchas búsquedas lineales se ven más afectados por las versiones paralelas que aquellos en los cuales había pocas búsquedas, donde la reducción de tiempos se da prácticamente sólo en la aproximación del gradiente.

Los problemas de optimización considerados en este trabajo incluyeron casos de estudio de mediano y gran tamaño donde la función objetivo es conocida y la proyección sobre la región factible es simple de realizar. En tal sentido, se proyecta ampliar el espectro de problemas a analizar abordando ejemplos con restricciones de características menos tratables y con desconocimiento de la expresión analítica de la función objetivo, dado que estas características aparecen con frecuencia en la formulación de numerosos problemas del campo de la ingeniería de procesos.

REFERENCIAS

Bello L. and Raydan M.. Convex-Constrained Optimization for the Seismic Reflection Tomography Problem,*Technical report 2003-12, Centro de cálculo científico y Tecnológico, Facultad de Ciencias, Universidad Central de Venezuel,* 2003.

Birgin E.G., Biloti R., Tygel M. and Santos L.T. Restricted oprimization: a clue to a fast and

accurate implementation of the common reflection surface method, *Journal of Applied Geophisycs*, 42:143-155, 1999a.
Birgin E.G, Chambouleyron I. and. Martinez J.M. Estimation of optical constants of thin films using unconstrained optimization, *Journal of Computation Physics*, 151:862-880, 1999b.
Birgin E.G. and Evtushenko Y.G. Automatic differentiation and spectral projected gradient methods for optimal control problems, *Optimization Methods and Software*, 10:125-146, 1998.
Birgin E., Martinez M. and Raydan M., Non-monotone spectral projected gradient method on convex sets, *SIAM Journal on Optimization*, 10: 1196-1212, 2000.
Birgin E., Martinez M. and Raydan M., "Algorithm 813: SPG-Software for convex constrained optimization", *ACM Transactions on Mathematical Software*, 27: 340-349, 2001.
Z.Castillo, Cores D. and Raydan M.. Low Cost Optimization Techniques for Solving the Nonlinear Seismic Reflection Tomography Problem, *Optimization and Engineering*, 1: 155-169, 2000
Dennis J.E. and Schnabel R.B. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, NJ, 1983.
Domancich A.O.,. Ardenghi J.I, Vazquez G. E. y Brignole N. B., Desempeño de un algoritmo espectral para optimización rigurosa de plantas de procesos industriales. *Mecánica Computacional XXIII*, 3047-3057, 2004.
Fletcher R. On the Barzilai and Borwein method, *Numerical Analysis Report NA/207*, 2001
Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R. and Sunderam V. *PVM: Parallel Virtual Machine*. MIT press, 1994.
Geist A., Kohl A.and Papadopoulos P. PVM and MPI: a Comparison of Features. *Calculateurs Paralleles*, 8(2), 1996.
Glunt W., Hayden T.L., and Raydan M.. Molecular conformations from distance matrices. *J. Comp. Chem.*, 14:114–120, 1993.
Grippo L., Lampariello F., and Lucidi S, A non-monotone line-search technique for Newton's method, *SIAM Journal on Numerical Analysis*, 23: 707-716, 1986.
Gropp W. and Lusk E. *Goals Guiding Design: PVM and MPI . Technical Report*, 2002
Luengo F.and Raydan M.. Gradient Method with Dynamical Retards for Large-Scale Optimization Problems, *Electronic Transactions on Numerical Analysis (ETNA)*, 16:186-193, 2003
Molina B., Petiton S.and Raydan M., Preconditioned Gradient Method with Retards for Parallel Computers, *Proceedings of the III National Meeting on Parallel and Distributed Computing, Merida, Venezuela*, 91-98, 1997
Molina B.and Raydan M. Preconditioned Barzilai–Borwein method for the numerical solution of partial differential equations, *Numerical Algorithms*, 13: 45-60, 1996.
MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8 (3/4):165-416, 1994
Mulato M., Chambouleyron I., Birgin E.G. and. Martinez J.M. Determination of thikness and optical constants of a-Si:H films from transmittance data, Applied Physics Letters, 77:2133-2135, 2000
Raydan M., On the Barzilai and Borwein choice of step-length for the gradient method. *IMA Journal on Numerical Analysis*, 13: 321-326, 1993.
Raydan M., The Barzilai and Borwein gradient method for the large-scale unconstrained minimization problem. *SIAM Journal on Optimization.*, **7**, 1: 26-33, 1997.
Vazquez, G.E. and Brignole N.B. Parallel NLP Strategies using PVM on Heterogeneous

Distributed Environments, *Lecture Notes in Computer Science Nº 1697: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Verlag,* XVII: 533-540, 1999.

Vazquez G.E., Rainoldi R. and Brignole N.B. Non-Linear Constrained GRG Optimization under Heterogeneous Parallel-distributed Computing Environments, *Computer-Aided Chemical Engineering, Elsevier*, 8: 127-132, 2000.