UNIVERSIDAD NACIONAL DEL SUR

Tesis de Doctor en Ciencias de la Computación

"El Método del Gradiente Espectral Proyectado Acelerado mediante Paralelismo: Aplicaciones a Ingeniería de Procesos"

Juan Ignacio Ardenghi

Bahía Blanca Argentina

2014

A Liz, mi gran amor.

A todos aquellos que pusieron un granito de arena en este trabajo de tesis,
desde los que la leyeron y dieron su opinión,
hasta los que me enseñaron a usar distintos tipos de software.
Pero una dedicación especial a aquellas personas que,
ligadas o no a las ciencias afines a este trabajo,
me transmitieron el valor del esfuerzo,
de la ética del trabajo y de la bonhomía de bien.

Juan Ignacio Ardenghi

Prefacio

Esta Tésis se presenta como parte de los requisitos para optar al grado Académico de Doctor en Ciencias de la Computación, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el ámbito del Departamento de Ciencias e Ingeniería de la Computación durante el período comprendido entre el 2004 y el 2013, bajo la dirección de la Dra. Nélida Beatriz Brignole y el Dr. Gustavo Esteban Vazquez.

Juan Ignacio Ardenghi

ardenghi@criba.edu.ar

Departamento de Ciencias e Ingeniería de la Computación

Universidad Nacional del Sur

Bahía Blanca, 20 de diciembre de 2013.

UNIVERSIDAD NACIONAL DEL SUR

Secretaría General de Posgrado y Educación Continua

La presente tesis ha sido aprobada el ..../..../....., mereciendo la calificación de .....(.....)

Resumen: En el área de Ingeniería de Procesos abundan los problemas de optimización no lineales. En busca de formulaciones más realistas ha aumentado la exigencia de un modelado riguroso. Como complejidades incorporadas, al aumento de la cantidad de variables continuas y restricciones no lineales se le suman la presencia de variables binarias. En muchos casos los problemas se resuelven mediante la relajación de variables y condiciones, así generando subproblemas no lineales cuya resolución se realiza a través de aproximaciones lineales y cuadráticas. La pregunta formulada en esta tesis es la siguiente ¿Podemos lograr eficiencia sin tener que relajar el problema? Es decir ¿podemos conseguir soluciones del modelo original en tiempos razonables? En esta tesis proponemos explotar el Método del Gradiente Espectral Proyectado (SPG) mediante su refundación a partir del paradigma paralelo.

El SPG es un método de optimización global no monótono para problemas de programación no lineal, con características diferentes a las exhibidas por 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 muchos problemas, acelerando la convergencia con mínimos costos de almacenamiento de datos. Entre sus características más atractivas aparece su bajo costo en operaciones: SPG no calcula matrices hessianas ni resuelve sistemas lineales. SPG sólo utiliza productos matriz vector y una estrategia de búsqueda lineal no monótona para garantizar convergencia global. Combinado con un esquema de Lagrangiano Aumentado, el método se muestra como una herramienta muy prometedora para el abordaje de problemas muy exigentes en cuanto a esfuerzo computacional y eficiencia. Sus puntos débiles se encuentran en el requerimiento 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 forma analítica. En problemas de aplicaciones industriales estos dos aspectos pueden devenir en verdaderos cuellos de botella del algoritmo. En consecuencia, el bajo costo aritmético por iteración no se ve reflejado en el tiempo total de resolución.

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. El objetivo de esta tesis fue el desarrollo y análisis del desempeño de una versión eficiente del algoritmo SPG programado en paralelo, asumiendo desconocimiento de expresiones analíticas de la función objetivo o de los gradientes. Este escenario a menudo se presenta en los problemas de optimización en ingeniería de procesos con gran cantidad de variables y restricciones no lineales. La nueva versión del algoritmo SPG genera una sucesión de iterados que es alternativa a la que genera la versión secuencial lo que lo hace más competitivo, pero manteniendo la robustez de convergencia que posee el método SPG original.

Se desarrollaron e implementaron dos versiones del algoritmo paralelo: una fue concebida para ejecutarse eficientemente sobre una arquitectura distribuida mediante pasaje de mensajes sobre una red de área local estándar, y la otra fue diseñada para ejecutarse sobre una arquitectura de memoria local compartida. La experimentación numérica se realizó sobre un cluster de 8 procesadores y en una computadora multicore de 12 núcleos. Se demostró en forma teórica la eficiencia esperada. Además, hemos contrastado estos desarrollos teóricos con resultados empíricos obtenidos en algunos problemas de diseño relacionados a plantas de procesos industriales, ubicando así a este resolvedor paralelo como una herramienta competitiva frente a los resolvedores clásicos de paquetes comerciales.

Abstract: There are many nonlinear optimization problems in the area of Process Engineering. In the search of more realistic formulations the need of more rigorous modeling has grown. The presence of binary variables, the increasing amount of continuous variables and nonlinear constraints count among the incorporated complexities. In many cases the problems are solved by relaxing variables and conditions, thus generating nonlinear subproblems whose resolution is carried out through linear and quadratic approximations. The question posed in this thesis is the following: Can we achieve efficiency without having to relax the problem? I mean: Can we get the original model solutions in reasonable time? In this thesis we propose to exploit the Spectral Projected Gradient method (SPG) by its relaunching from the parallel paradigm.

SPG is a non-monotone global optimization method for nonlinear programming problems, its features being different from those exhibited by the classical projected-gradient methods. The non-monotonicity and a particular choice of the step length allow to exploit special situations that arise in many problems, accelerating the convergence with minimal data-storage costs. Its low operating cost features among its most attractive attributes SPG neither calculates Hessian matrices nor solves linear systems. SPG just performs matrix vector products and a non-monotone line-search strategy in order to ensure global convergence. When combined with an Augmented Lagrangian scheme, the method looks like a promising tool for addressing demanding problems in terms of computational effort and efficiency. Its weaknesses lie in the requirement of too many line-searches for a new iterate, and in the need for a good approximation of the gradient when it is not available in analytical form. In industrial application these two mentioned aspects may become real bottlenecks in the algorithm. In consequence, the low arithmetic cost per iteration is not reflected in the total elapsed time of resolution.

The boom development in parallel programming presents this paradigm as a resource that provides a great opportunity to overcome these drawbacks. The goal of this thesis was the development and analysis of the performance of an efficient version of the SPG algorithm programmed in parallel, assuming lack of knowledge about the analytical expressions of the objective function or gradients. This scenario often appears in process

engineering optimization problems with many variables and non-linear constraints. The new version of the SPG algorithm generates a sequence of iterates that is alternative to the one generated by the sequential version. In this way, the proposed version becomes more competitive, while maintaining the robustness of the original method.

Two versions of the parallel algorithm were developed and implemented: one of them was conceived to run efficiently on a distributed architecture by using message passing on a standard local area network, and another one was designed to run on a shared local-memory architecture. The numerical experiments were performed on a cluster of 8 processors and a 12-core multicore computer. We have proved the expected efficiency theoretically. Besides, we have contrasted these theoretical developments with empirical results in some design problems related to industrial plants processes. thus placing this parallel solver as a competitive tool against classical commercial packages.

Contenido

Capítulo 1

Introducción

"La computación científica se mueve
entre la matemática y las ciencias de la computación
para desarrollar las mejores formas de utilizar los sistemas de cómputo
para resolver problemas de ciencias e ingenierías".
G. Golub

La optimización ha evolucionado en poco tiempo dejando de ser una metodología de interés puramente académico para convertirse en una herramienta tecnológica con fuerte impacto en la industria [Biegler & Grossmann, 2004]. Las computadoras con nuevas capacidades tecnológicas y las técnicas de resolución innovadoras que surgen día a día compiten con los cada vez mas complejos problemas que se plantean. La búsqueda de mayor realismo en los modelos hace que estos se vuelvan más y más complicados y por ende difíciles de resolver en tiempos cortos.

Uno de los campos en los cuales el impacto de estas nuevas tecnologías computacionales ha sido mayor es el campo de la optimización global dado el gran tamaño y la naturaleza combinatoria de muchos de sus problemas. La investigación en este campo se puede abordar desde distintas aristas o niveles. El nivel denominado de computación científica es solo una parte de las *ciencias computacionales*, y concierne al desarrollo, la implementación y el uso de software para algoritmos de optimización numérica

[Heath, 1997]. La investigación en este nivel está fuertemente enfocada tanto en las propiedades matemáticas como en la puesta en práctica del método de optimización, buscando un uso eficiente y 'práctico' incluyendo entre los temas de investigación el rendimiento y la complejidad computacional [Biegler, 2010].

En particular, en ingeniería de procesos abundan los problemas muy exigentes en cuanto a esfuerzo computacional y eficiencia. Al abordar un estudio a este nivel de computación científica se deben tener en cuenta los conceptos de eficacia y eficiencia, que son dos requisitos metodológicos para un algoritmo en sistemas de procesos. Siguiendo a Ponzoni et al., estos dos conceptos pueden definirse como :

1. **Eficacia**: capacidad de encontrar una solución fiable para un determinado problema

2. **Eficiencia**: el potencial para encontrar esta solución en tiempos de ejecución razonablemente bajos, incluso para los problemas industriales de gran tamaño [Ponzoni *et al.*, 1999].

Entonces un algoritmo no puede ser eficiente si no es eficaz, pero a un algoritmo eficaz se lo puede hacer eficiente.

En muchos casos, la optimización en ingeniería química implica la solución de un problema de programación no lineal[1] (NLP[2])[Himmelblau & Edgar, 1988]. El nivel de realismo y detalle introducido en la formulación matemática da lugar a modelos de complejidades y tamaños diferentes. Una forma generalizada de encontrar soluciones de un problema complejo es la relajación de algunas restricciones, dándole al problema la estructura adecuada para ser resuelto por los paquetes de optimización comercial. Este enfoque simplificado puede resultar en una solución inexacta o no factible [Abdel-Jabbar *et al.*, 1998, Egea *et al.*, 2007]. Por otro lado, buscar una solución

[1]El uso del término *programación* en optimización no refiere a la programación de computadoras sino al planeamiento de actividades en el sentido de la investigación operativa o de la ciencia de la administración [Heath, 1997].

[2]Nonlinear Programming

manteniendo la complejidad original del problema original puede producir un costo computacional muy alto [Alattas *et al.*, 2011].

En términos generales, en ingeniería química hay una amplia variedad de intentos para reducir tiempo de computación, tales como reformulación de modelos [Kraemer *et al.*, 2009, You *et al.*, 2011], control de profundidad de convergencia [Wang *et al.*, 2007] y enfoques basados en sustitutos (surrogate-based approach) [Egea *et al.*, 2007].

Al mismo tiempo, en la informática una de las alternativas más potentes que ha surgido para reducir tiempos de cómputo es la programación en paralelo, siendo esta uno de los puntales tecnológicos para el abordaje de problemas NLP [Alba, 2005]. Hacer diseños para computación en paralelo puede dar a lugar a programas muy eficientes y prácticos, pero al mismo tiempo alcanzar estos diseños óptimos representa un importante desafío para la comunidad de programadores. Por otra parte, la computación en paralelo es útil para complementar otros enfoques, como son las reformulaciones de modelos. Entonces, vale la pena abordar a través del paralelismo problemas especiales en el área de ingeniería química, ya que éste puede contribuir a inspirar a otros en la búsqueda de nuevos y más eficientes enfoques relacionados con las nuevas tecnologías.

1.1 Motivación

En los problemas de optimización aplicados a ingeniería de procesos uno de los cálculos más costosos es el de evaluar la función objetivo. Cada una de estas evaluaciones suele requerir la simulación entera de la planta de procesos y cuanto más complejo es un sistema, mas dramáticamente se incrementa el tiempo que requiere su simulación [High & Laroche, 1995]. Este tipo de problemas son los que, mas allá de contar con una solucion algorítmica y la cantidad de memoria requerida, el tiempo de ejecución es crucial (*Time-consuming problems*).

Hoy en día, la forma de abordar estos problemas exigentes en lo que respecta al esfuerzo computacional es, sin duda, un tema de interés y preocupación para los

ingenieros químicos. Kossack et al. propusieron un método de programación sucesiva para problemas de programación no lineal mixta entera (MINLP[3]) relajados, con el fin de reducir tanto los tiempos de cálculo como para mejorar la robustez del algoritmo de optimización basado en el diseño de una columna de destilación [Kossack *et al.*, 2006].

Kraemer et al. abordaron el exigente problema de optimización rigurosa de un complejo proceso de destilación a gran escala. Los autores pudieron reformular el modelo como un problema de optimización puramente continua para buscar un óptimo local de mejor calidad con una reducción del tiempo de cálculo requerido [Kraemer *et al.*, 2009].

En caso de problemas de optimización relacionados con las cuestiones de planificación, You et al. también se abocaron a la reducción del tiempo de cálculo por medio de reformulaciones. Con el fin de resolver en gran escala instancias eficazmente, se han propuesto las siguientes estrategias computacionales: I. una estrategia de solución de dos niveles y II. un método de aproximación continua. Sus enfoques condujeron a las mismas soluciones óptimas, pero con tiempos de CPU diferentes. En la sección A.5 titulada 'Complejidad Computacional', los autores señalaron que en la medida que el problema aumente de tamaño, la resolución del modelo agregado puede llegar a ser intratable debido a su complejidad combinatoria [You *et al.*, 2011].

Estos enfoques se realizaron partiendo desde un punto de vista secuencial, es decir sin explotar cualquier oportunidad de paralelismo,

El pujante desarrollo de arquitecturas de computación paralela permite el progreso de un paradigma diferente en el desarrollo de algoritmos numéricos. De hecho, muchos de los nuevos algoritmos han sido inspirados en la disponibilidad de las nuevas posiblidades de paralelismo generando una importante simbiosis entre la optimización numérica y las tecnologías computacionales [Migdalas *et al.*, 2003]. Algunas características ventajosas de estas transformaciones son reconocidos con distintas perspectivas en la literatura actual.

Cheimarios et al. introdujeron el esquema de paralelismo de maestro-trabajador con el fin de reducir el tiempo de cómputo de un método iterativo aplicado a una

[3]Mixed Integer Nonlinear Programming

metodología de acoplamiento empleada para modelar procesos de deposición química de vapor (CVD). El análisis del rendimiento de su aplicación paralela lo informaron minuciosamente. Le dedicaron una sección completa a este debate, y afirmaron que sus cálculos fueron 'eficientemente acelerados' mediante el uso de una técnica paralela [Cheimarios *et al.*, 2013].

Para el proceso de análisis de flexibilidad, Moon et al. propusieron un nuevo algoritmo híbrido que es adecuado para la computación en paralelo [Moon *et al.*, 2008]. A su vez, Chen et al. aceleraron el cálculo de un modelo riguroso de la distribución de peso molecular con un método de cálculo paralelo [Chen *et al.*, 2011]. Laird et al. presentaron una estrategia de descomposición aplicable a problemas de optimización con restricciones con ecuaciones diferenciales-algebraicas con el objeto de distribuir las operaciones del álgebra lineal en múltiples procesadores [Laird *et al.*, 2011].

Según Buzzi-Ferraris y Manenti, la programación paralela puede significativamente modificar el análisis numérico como se ha concebido hasta ahora. Estos autores afirman que esta programación sin duda afectará la forma en que se aplican los métodos y algoritmos numéricos. Por otra parte, han señalado que la computación paralela en los ordenadores personales constituye una revolución silenciosa que implica directamente a Ingeniería de Sistemas de Procesos y las comunidades de diseñadores de procesos asistidos por computadora [Buzzi-Ferraris & Manenti, 2010]. El paso de la programación secuencial a la programación en paralelo representa un paso muy grande ya que se descartan las propiedades más importantes y atractivas de la computación secuencial: claridad, previsibilidad, y determinismo [Lee, 2006].

Asimismo, aquellos algoritmos originalmente secuenciales, pueden ser 'recreados' a partir de sus posibilidades de paralelismo, aumentando considerablemente sus prestaciones [Buzzi-Ferraris & Manenti, 2010]. Este es el caso del método del gradiente espectral proyectado.

El método del gradiente espectral proyectado (SPG[4]) es un método de optimización global no monótono para problemas de programación no lineal de características

[4]Spectral Projected Gradient

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. Entre sus características más atractivas aparece su bajo costo en operaciones: SPG no calcula matrices Hessianas ni resuelve sistemas lineales, sólo utiliza productos matriz vector y una estrategia de búsqueda lineal no monótona para garantizar convergencia global.

Muchos estudios se han realizado sobre este método y la comparación con otras técnicas de optimización numérica de bajo costo ha favorecido a SPG en gran cantidad de problemas de gran escala. Entre los principales estudios se encuentran los de Birgin et al. [Birgin *et al.*, 2001a, Birgin *et al.*, 2000, Birgin *et al.*, 2009] y la revisión general de Fletcher [Fletcher, 2001]. Estos métodos también han sido aplicados exitosamente en diferentes áreas como son la geofísica [Bello & Raydan, 2004, Birgin *et al.*, 1999b, Castillo *et al.*, 2000, Cores *et al.*, 2000], la física [Birgin *et al.*, 1999a, Mulato *et al.*, 2000], la química [Glunt *et al.*, 1991, Glunt *et al.*, 1993a, Glunt *et al.*, 1993b, Glunt *et al.*, 1994, Wells *et al.*, 1994] y la teoría de control [Birgin & Evtushenko, 1998]. También es parte de algoritmos para resolver sistemas de ecuaciones no lineales [Brezinski & Chehab, 1999, La-Cruz & Raydan, 2003], ecuaciones diferenciales parciales [Molina & Raydan, 1996, Raydan, 2002] y otros problemas de programación no lineal [Birgin & Martínez, 2002, Diniz-Ehrhardt *et al.*, 2004, Molina & Raydan, 2002, Luengo *et al.*, 2002].

Una versión de SPG para minimizar funciones cuadráticas sin restricciones fue desarrollada en el trabajo de Molina et al. para correr sobre máquinas paralelas [Molina *et al.*, 2000]. En dicho trabajo la paralelización se implementó en la distribución de datos y en las operaciones de álgebra lineal involucradas en el algoritmo, logrando una eficiencia fuertemente dependiente de la estructura cuadrática de la función objetivo. Esto representa una limitación en la aplicación del algoritmo en problemas NLP donde la función objetivo, además de no ser cuadrática, está ligada a restricciones no lineales de igualdad y desigualdad.

SPG ofrece otras posibilidades de paralelismo y mantiene sus propiedades de convergencia, aún cuando el gradiente no es exacto [Ardenghi *et al.*, 2009]. Al mismo tiempo, la combinación de SPG con un esquema de lagrangiano aumentado para problemas con restricciones puede resultar una herramienta alternativa muy interesante para resolver problemas NLP como los que se presentan en ingeniería de procesos.

1.2 Optimización aplicada a ingeniería de sistemas de procesos

El desarrollo de algoritmos para problemas de optimización no lineal generalizados ha tenido un importante crecimiento en las últimas décadas dada su aplicabilidad a diferentes y diversificadas áreas científicas. Ingeniería de Procesos es uno de estos campos, donde ha habido una proliferación importante de este tipo de desarrollos debido al aumento requerido en la rigurosidad de modelado buscando formulaciones más realistas. Como consecuencia aparece un aumento en el tamaño y la complejidad de los problemas tratados. En muchos casos es inevitable la presencia de restricciones no lineales de igualdad y desigualdad agregadas a una función objetivo también no lineal.

Desde el punto de vista teórico, los problemas en este campo suelen tener muchos óptimos locales representando rendimientos variados y complejos, por lo que a menudo no es fácil identificar la solución óptima a través del razonamiento intuitivo. La economía del sistema planteado muestra a menudo que la solución óptima se traduce en grandes ahorros, del mismo modo que aferrarse a una solución subóptima puede representar una sanción económica grave. Por lo tanto, la optimización se ha convertido en un aporte importante que ayuda a la industria química para mantener su competitividad [Biegler, 2010].

Los problemas de optimización que surgen en ingeniería de sistemas de procesos son muy variados. Cuestiones de scheduling y de planeamiento originan problemas de programación lineal (LP) y mezcla entera LP (MILP), mientras que diseñar procesos

tiende a generar problemas de tipo NLP y de mezcla entera no lineales (MINLP). Esto se debe principalmente a que en las cuestiones de diseño se tiende a darle un peso más importante a las predicciones del modelo de procesos, que es no lineal, mientras que en scheduling y planeamiento las predicciones físicas tienden a ser comparativamente menos relevantes ya que la mayor parte de las operaciones son descriptas a través de requerimientos de tiempo y actividades. Estos problemas MINLP se resuelven mediante la relajación de las variables enteras generando subproblemas no lineales cuya resolución se logra a través de aproximaciones lineales y cuadráticas. La inquietud que surge en este punto es la siguiente: ¿podemos conseguir mejores soluciones mediante un algoritmo que resuelva de 'otra forma' el subproblema no lineal?

Entre las técnicas generales de resolución de los problemas NLP pueden distinguirse dos grandes categorías: I. técnicas que no emplean información asociada a las derivadas y II. técnicas que sí requieren esa información.

Los métodos de tipo I están basados en estrategias de búsqueda bien definidas que requieren sólo evaluaciones de la función objetivo. Entre ellos se destaca el de direcciones conjugadas de Powell [Powell, 1964], el cual fue desarrollado en la década del 70 y aún se aplica en problemas de ingeniería química [Biegler & Grossmann, 2004]. Los algoritmos metaheurísticos son otras alternativas, cuya popularidad está creciendo.

Las técnicas metaheurísticas, también llamadas sistemas inteligentes, son algoritmos aproximados de optimización y búsqueda de propósito general y provienen del área computacional conocida como Inteligencia Artificial. Se pueden definir como procedimientos iterativos que guían una heurística subordinada combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda [Herrera, 2006]. Estos algoritmos surgieron a comienzos de la década del 80, cuando se empezó a disponer de un potencial de cálculo más grande. Entonces, fue posible empezar a transformar algunas ideas provenientes de otras ciencias en algoritmos útiles. A partir de ese momento y hasta el día de hoy, se intenta aprovechar el creciente desarrollo de las ciencias computacionales y por ende, se han propuesto diversas ideas interesantes para diseñar algoritmos heurísticos. Una heurística es una técnica que busca

buenas soluciones (es decir, cuasióptimas) a un costo computacional razonable, pero sin contar con criterios formales que garanticen la optimalidad o factibilidad de la solución obtenida, o incluso que puedan indicar cuan cerca de la optimalidad se encuentra una solución factible [Biegler & Grossmann, 2004, Carballido *et al.*, 2004]. El aspecto más relevante de su popularidad en entornos científicos, tecnológicos, ingenieriles o empresariales es el éxito asociado a la eficiencia y efectividad. Estos algoritmos paralelos derivados de las metaheurísticas se destacan en la solución de casos de gran tamaño y aplicaciones reales [Alba, 2005]. A pesar de que los métodos heurísticos pueden estar basados en diferentes filosofías, sin duda ofrecen mecanismos eficaces para investigar efectivamente un espacio de búsqueda con el fin de resolver problemas duros (NP-hard) de optimización donde los métodos tradicionales han fallado, o son prácticamente inaplicables [Borraz-Sánchez, 2010].

Entre las técnicas metaheurísticas más populares se encuentran búsqueda tabú o *Tabu Search* [Glover & Laguna, 1997], algoritmos genéticos o *Genetic Algorithms* [Holland, 1975], enfriamiento simulado o *Simulated Annealing* [Kirkpatrick *et al.*, 1983], redes neuronales o *Neural Networks* [Hopfield & Tank, 1985], optimización basada en colonias de hormigas o *Ant Colony Optimization* [Dorigo, 1992], búsqueda variable por entornos Variable o *Neighborhood Search* [Mladenović & Hansen, 1997], y GRASP o *Greedy Randomized Adaptive Search Procedures* [Feo & Resende, 1995].

El escaso desarrollo de las bases teóricas de estas técnicas puede ser considerado como una debilidad. Estos métodos son inexactos y altamente probabilísticos (no determinísticos) [Herrera, 2006, Biegler & Grossmann, 2004]. Los intentos por organizar este campo son numerosos, pero los conceptos principales son raramente definidos con precisión y hay todavía muy pocos teoremas significativos. Ninguna estructura ha conseguido una aceptación general. De hecho, cada grupo de investigación inspirador de una metaheurística tiene su propio punto de vista y habilidad para explicar muchas heurísticas en su propio vocabulario, así como para absorber ideas de todo el campo (generalmente bajo la forma de híbridos), lo que Moreno Pérez llama el 'caracter babélico' de las investigaciones en este campo [Pérez, 2004].

Con respecto a los métodos de tipo II, es decir, que utilizan derivadas, los de tipo gradiente se basan en la búsqueda de direcciones de descenso, mientras que los de tipo SQP (Sequential Quadratic Programming) emplean aproximaciones cuadráticas de ciertas funciones asociadas al problema [Luenberger, 1984]. Los métodos basados en SQP son los más populares resolvedores de NLPs en los que se han focalizado los investigadores en ingeniería de sistemas de procesos. El uso de técnicas de tipo Newton o quasi Newton aplicadas a estrategias de programación cuadrática sucesiva tienden a requerir menos iteraciones, por ende, menos evaluaciones de función, que otros algoritmos. Como fue descripto en la introducción, en optimización de procesos uno de los cálculos mas costosos es el de la evaluación de la función objetivo, que puede llegar a requerir la simulación completa de una planta de procesos.

Estos métodos SQP tienen dos modos básicos de ser aplicados cuando el número de variables es grande: usando técnicas de descomposición o explotando la estructura natural del problema específico a resolver. En el área de optimización de procesos químicos han sido experimentados ambos modos. Las técnicas de descomposición permiten reducir el tamaño de los subproblemas QP que deben ser resueltos en cada iteración, pero dificultan el uso de información de segundo orden analíticas, y el subproblema QP asociado puede ser grande y denso cuando el problema general tiene muchos grados de libertad [Biegler & Grossmann, 2004].

Por otro lado, también se han reportado buenos resultados al explotar la estructura natural que adquieren ciertos problemas al aplicarse una estrategia SQP para resolverlo. En general, aplicados a un amplio rango de problemas de ingeniería de procesos con variadas estructuras, han demostrado excelente rendimiento en la práctica por utilizar relativamente poca cantidad de evaluaciones de funciones.

Los métodos tipo gradiente también han sido desarrollados y adaptados para el tratamiento de problemas de gran escala. El más conocido es el del gradiente reducido generalizado (GRG), que particiona el vector de variables, clasificándolas en básicas e independientes, para luego evaluar el gradiente reducido de estas últimas. En comparación con SQP, estos métodos son más costosos [Abadie & Carpentier, 1969].

Con respecto a problemas MINLP sólo vamos a mencionar una de las técnicas más populares que es la de tipo Branch and Bound. En ésta un árbol de búsqueda abarca todo el espacio de soluciones del problema y el algoritmo intenta ubicar la solución óptima mediante una selección inteligente de las ramas a explorar. En este punto hay que destacar que cada una de estas exploraciones involucra la resolución de un subproblema asociado de tipo NLP [Leyffer, 2001, Borchers & Mitchell, 1985].

Muchas investigaciones han sido dedicado a desarrollar métodos eficientes de propósito más general para resolver problemas de programación no lineal de gran escala. Estos métodos han sido estudiados y refinados por más de una década, muchos usuarios los utilizan a través de plataformas de modelado como GAMS (General Algebraic Modeling System), AMPL(A Mathematical Programming Language) y AIMMS (Advanced Interactive Multidimensional Modeling System), y estos resolvedores están a su vez integrados a muchas otras aplicaciones [Biegler & Grossmann, 2004].

La plataforma GAMS merece un párrafo aparte dado que ha devenido la herramienta comercial orientada a ecuaciones para modelado más utilizada en simulación y optimización de procesos [Biegler, 2010]. Sobre esta plataforma pueden ser utilizados resolvedores de distintas características entre los que se encuentran

- MINOS5 [Murtagh & Saunders, 1987] y SNOPT [Gill *et al.*, 2002], que utilizan un método de Lagrangiano aumentado como su algoritmo central. Este método resulta útil para resolver problemas NLP porque no produce el efecto de *Maratos* , i.e. no rechaza pasos favorables [Nocedal & Wright, 1999]. Esta es la razón por la que muchos paquetes comerciales lo eligen. Este algoritmo funciona bien cuando las derivadas segundas exactas están disponibles [Biegler & Grossmann, 2004]. Para problemas no lineales SNOPT utiliza un algoritmo de programación cuadrática sucesiva para problemas sparse con un método quasi-Newton de memoria limitada para aproximar el hessiano del lagrangiano. MINOS resuelve iterativamente subproblemas con con restricciones linealizadas junto con una función objetivo ensamblada como un lagrangiano aumentado [GAMS, 2013].

- PATHNLP [Ferris & Munson, 1999] resuelve el problema NLP cosntruyendo internamente el sistema asociado a las condiciones de primer orden de optimalidad de Karush-Kuhn-Tucker (KKT). La solución de este sistema se encuentra con el resolvedor para problemas de complementaridad PATH.

- LINDO [Lindo, n.d.], que es un procedimiento de optimización global que emplea un método *Branch-and-cut* para descomponer un modelo NLP en una lista de subproblemas.

- CONOPT [Drud, 1985], que utiliza el ya mencionado algoritmo GRG con un diseño especial para problemas NLP de gran tamaño.

- KNITRO [Byrd *et al.*, 1999], que es una implementación de un método de punto interior junto con gradientes conjugados.

- BARON [Sahinidis, 1996], que es un resolvedor que puede ir conmutando resolvedores durante una búsqueda, basándose en las características del problema y de las performance que van teniendo los resolvedores elegidos.

Otras variantes, que eliminan variables de estado y linealizan ecuaciones, son conocidas como *reduced space* SQP, o métodos rSQP [Jager & Sachs, 1997].

Los resolvedores de sistemas lineales involucrados en NLP también han tenido una evolución importante en los últimos 20 años. Este desarrollo avanza desde resolvedores con matrices densas (como las factorizaciones QR en NPSOL [Gill *et al.*, 1998]) a factorizaciones directas en matrices sparse en códigos como SOCS [Betts, 2001], IPOPT [Wachter & Biegler, 2006] y LOQO [Vanderbei & Shanno, 1997].

En este punto cabe preguntarse por qué la necesidad de continuar desarrollando nuevos resolvedores de NLP con tantas implementaciones disponibles tan diferentes y refinadas. Es claro que la motivación para desarrollar software específico de optimización es aprovechar mejor las propiedades particulares de cierta clase de problemas NLP. Bartlet y Biegler afirman que en general, los resolvedores de NLP actuales pueden explotar la estructura sparse del problema NLP, aunque los resolvedores lineales

que utilizan requieren las entradas no nulas explícitas de la matrices Jacobiana y Hessiana [Bartlett & Biegler, 2003]. Estos resolvedores, en general, no sacan ventaja de las oportunidades de aplicar técnicas avanzadas de álgebra lineal, o de los puntos paralelizables. Como ejemplo de estas aplicaciones aparecen los problemas de optimización con restricciones de tipo diferencial-algebraica (DAE) y de ecuaciones en derivadas parciales (PDE) [Biegler *et al.*, 2001]. Códigos de simulación para problemas grandes ha sido desarrollados para resolver sistemas DAEs no lineales y PDEs con millones (o eventualmente billones) de variables de estado discretizadas. Esto ha sido posible gracias a los avances logrados en el desarrollo de resolvedores paralelos masivos y computadoras con multi procesadores [Womble *et al.*, 1999]. Sin embargo los códigos de optimización actuales no aprovechan las ventajas de estos avances [Bartlett & Biegler, 2003].

Al mismo tiempo, dado que muchos métodos han sido desarrollados para para su ejecución en computadoras convencionales, en general no explotan el poder de cálculo de las máquinas paralelas [Coon & Stadtherr, 1995].

Este punto evidencia un tema de investigación abierto que es el desarrollo de algoritmos eficientes para optimización no lineal a partir de la integración de los métodos más modernos con los grandes avances que han tenido lugar (y se siguen produciendo) a nivel de hardware.

1.3 Objetivos

En base a las oportunidades de paralelismo que ofrece el método SPG, y dado que no existe una implementación de éste para problemas donde no se cuenta con expresiones analíticas de la función objetivo o de los gradientes, el objetivo de esta tesis es el desarrollo y análisis del desempeño de una versión eficiente del algoritmo SPG programado en paralelo, en condiciones de desconocimiento de estas mencionadas expresiones, escenario que se presenta en los problemas de optimización en ingeniería de procesos con gran cantidad de variables y restricciones no lineales. La nueva versión del

algoritmo deberá generar 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 original.

Como resultado, en este trabajo de tesis hemos desarrollado e implementado el algoritmo SPG para ejecutarse eficientemente sobre una arquitectura distribuida mediante mensajes sobre una red de área local estándar, y sobre una arquitectura de memoria local compartida. La ya alta performance de SPG secuencial se ve mejorada por una distribución de tareas dinámica y eficiente que permite abordar problemas de gran porte. La eficiencia para resolver problemas de ingeniería de procesos permite contar con una herramienta alternativa a las que actualmente se cuentan en el campo de la ingeniería de procesos (HYSYS, GAMS, etc.) Por otro lado, estos problemas también pueden ser el punto de partida para futuros desarrollos en otras aplicaciones.

1.4 El problema general de optimización

Dada una función $f(x) : \mathbb{R}^n \to \mathbb{R}$ que se asume diferenciable, el problema general de optimización planteado en este trabajo es el de encontrar $x \in \mathbb{R}^n$ que minimiza esta función dentro de un conjunto definido por cotas en las componentes de x, ecuaciones e inecuaciones no lineales. Si notamos $c_i(x) = 0,\ i = 1 \cdots n_i$ a las restricciones de igualdad, $c_j \geq 0,\ j = 1 \cdots n_j$ a las restricciones de desigualdad, y $l_i \leq x \leq u_i$ las cotas que limitan las variables, entonces el conjunto de restricciones $\Omega \in \mathbb{R}^n$ definido por

$$\Omega = \{x \in \mathbb{R}^n : \mathrm{c_i(x) = 0, i = 1 \cdots n_i},\ \mathrm{c_j \geq 0, j = 1 \cdots n_j},\ \mathrm{l_i \leq x \leq u_i}\} \tag{1.1}$$

determina la *región factible* del problema de optimización. Denominando *función objetivo* a la función f a minimizar, el problema general se puede enunciar entonces como

$$\begin{cases} \text{Mínimo de } f(x) \\ \text{sujeto a } x \in \Omega \end{cases} \tag{1.2}$$

o en su forma algebraica, a través del sistema de ecuaciones (1.3)

$$\begin{cases} \min\limits_{x} f(x) & \\ \text{s. a. } c_i(x) = 0 & i = 1 \cdots n_i \\ c_j(x) \geq 0 & j = 1 \cdots n_j \\ l_i \leq x_i \leq u_i & \end{cases} \tag{1.3}$$

La condición que se impone sobre el conjunto Ω es que sea cerrado y convexo, es decir, cualquier segmento que una dos de sus puntos esté completamente contenido en el conjunto.

Al ser la *función objetivo* y las restricciones expresiones no lineales, el modelo del problema es un modelo NLP.

En este trabajo no se abordan problemas de programación entera como son los modelos MINLP. Si bien en el capítulo 8.3 se analiza la superestructura de un problema MINLP, en esta tesis el análisis en la búsqueda de una solución se basa en el desempeño de la resolución de los subproblemas NLP intrínsecos al problema general.

1.5 Estructura de esta tesis

El trabajo está organizado en diez capítulos. Este primer capítulo de introducción presenta la motivación y los objetivos generales de esta tesis, expone el estado del arte en el área de optimización en ingeniería de procesos y enuncia el planteo de nuestro problema de optimización. En el capítulo 2 se describe el método SPG. El análisis de este algoritmo se describe en el capítulo 3. En el capítulo 4 se analiza el desempeño del algoritmo SPG en problemas de ingeniería. El capítulo 5 se refiere al paradigma paralelo. En el capítulo 6 se describe nuestra versión paralela de SPG. El capítulo 7 se concentra en las diferentes implementaciones. En el capítulo 8 se presentan los resultados obtenidos en aplicaciones sobre problemas industriales y se aborda el problema de minimizar la superestructura MINLP que se genera en ciertos problemas industriales. Finalmente,

en el capítulo 9 se exponen las principales conclusiones de este trabajo y se enuncian los lineamientos y sugerencias para futuras implementaciones.

Capítulo 2

El método del gradiente espectral proyectado (SPG)

"Es sorprendente que una herramienta tan simple [SPG] pueda ser competitiva con algoritmos tan elaborados que utilizan subrutinas y procedimientos numéricos ampliamente probados..."

M. Raydan

El método del gradiente espectral proyectado (SPG) pertenece a una clase de algoritmos de requerimientos mínimos de almacenaje y juega un rol muy importante en la resolución numérica de problemas de optimización de gran escala. Este método, aplicado a problemas con y sin restricciones ha ganado su importancia dado que converge globalmente con bajos requerimientos de trabajo computacional [Birgin *et al.*, 2000, Raydan, 1993, Raydan, 1997].

Consideremos el problema definido en 1.3

$$\begin{cases} \text{Mínimo de } f(x) \\ \text{sujeto a } x \in \Omega \end{cases}$$

donde $\Omega \in \mathbb{R}^n$ es el conjunto que contiene todas las restricciones y cotas, definido en 1.1, y $f(x) : \mathbb{R}^n \to \mathbb{R}$ es una función continuamente diferenciable en un conjunto abierto que contiene a Ω.

A lo largo de esta tesis vamos a usar la siguiente notación: x_k es el iterado k-ésimo, $g_k = \nabla f(x_k)$, el vector gradiente de f en x_k, y $H_k = \nabla^2 f(x_k)$, la matriz Hessiana de f en x_k.

Definición 2.1 *Un método iterativo para minimizar una función* $f(x) : \mathbb{R}^n \to \mathbb{R}$ *se dice monótono si para dos iterados consecutivos* x_k *y* x_{k+1} *es* $f(x_{k+1}) < f(x_k)$.

El método SPG tiene como principal característica la no monotonicidad, es decir, no se exige a la función objetivo decrecer de iterado a iterado. Esta característica permite mantener ciertas propiedades que se perderían si se demandaran iterados estrictamente decrecientes.

La figura 2.1 muestra un esquema general del algoritmo SPG.

- **Paso 1.** Inicializar
- **Paso 2.** Chequear convergencia
- **Paso 3.** Calcular la dirección d_k y la longitud de paso s_k
- **Paso 4.** Búsqueda lineal no monótona.
- **Paso 5.** Actualización.

Figura 2.1: Esquema general del algoritmo SPG

- **Paso 1**: Se setean los parámetros iniciales, se proyecta el punto inicial y se calcula el gradiente. Considerando que la función podría no tener una expresión explícita, es necesario obtener una buena aproximacón del gradiente.

- **Paso 2**: Se evalúa si la norma de la proyección $\|P(x_\star - g(x_\star)) - x_\star\|_\infty$ [1] es menor que una tolerancia especificada. Esta proyección sobre la región factible depende mucho de las características geométricas de esta última. Esto se va a ver

[1] Se utiliza la norma *infinito* pero a los efectos de la convergencia puede ser usada cualquier norma-p [Diniz-Ehrhardt *et al.*, 2004]

simplificado a partir de la introducción de un esquema de lagrangiano aumentado, como se describirá en el capítulo 3.

- **Paso 3:** La dirección d_k es la dirección de máximo descenso

$$d_k = -g_k$$

y la longitud del paso es

$$\sigma_k = \frac{d_{k-1}^T d_{k-1}}{d_{k-1}^T y_{k-1}} \tag{2.1}$$

donde $y_{k-1} = g_k - g_{k-1}$. Luego, el paso de un iterado al siguiente es

$$s_k = \sigma_k d_k$$

- **Paso 4:** Búsqueda lineal no monótona.
Si se considera F_{max} el máximo valor de función en los últimos m iterados, y $0 < \gamma < 1$, entonces, dados x_k y el paso s_k, la búsqueda lineal no monótona consiste en encontrar un valor $t \in (0, 1)$ que verifique:

$$f(x_k + ts_k) \leq F_{max} + \gamma t s_k^T g_k \tag{2.2}$$

- **Paso 5:** : Se actualizan x_k, g_k y F_{Max}.

La primera versión de este algoritmo fue presentada por Barzilai y Borwein para funciones cuadráticas [Barzilai & Borwein, 1988]. Para dicho caso los autores probaron el método resulta ser R-linealmente convergente [Raydan, 1993, Dai & Liao, 2002]. Molina y Raydan también han propuesto una versión precondicionada [Molina & Raydan, 1996]. Para el caso general, está demostrado que el método converge globalmente a un punto estacionario incorporándole una búsqueda lineal no monótona como estrategia de globalización [Raydan, 1997]. La velocidad de convergencia viene dada por la siguiente definición

Definición 2.2 *Se dice que la sucesión $\{x_k\}$ converge a $x_\star$ con R-orden de convergencia p si existe una constante $c \geq 0$ i un índice $K > 0$ tal que para todo $k \geq K$ se cumple*

$$\|x_{k+1} - x_\star\| \leq c\|x_k - x_\star\|^p$$

Si $p = 1$ la convergencia se dice R-lineal [Dennis & Schnabel, 1983].

Dai y Liao mostraron que, bajo ciertas condiciones, el método es localmente R-lineal convergente para funciones en general [2] [Dai & Liao, 2002].

2.1 Longitud del paso

El análisis del modelo cuadrático ofrece una visión clara del funcionamiento del método elegido. Consideremos la función

$$f(x) = \frac{1}{2}x^T Ax + b^T x + c \tag{2.3}$$

el método del gradiente espectral define el paso en el iterado k

$$d_k = -\frac{1}{\alpha_k}g_k$$

donde

$$\alpha_k = \frac{s_{k-1}^T y_{k-1}}{d_{k-1}^T d_{k-1}}$$

siendo $y_{k-1} = g_k - g_{k-1}$ ($\alpha_k = \frac{1}{\sigma_k}$ siguiendo la definición dada en 2.1)

Esta expresión para el caso cuadrático equivale a

$$\alpha_k = \frac{g_{k-1}^T A g_{k-1}}{g_{k-1}^T g_{k-1}}$$

Sin perder generalidad se puede asumir que existe una transformación ortogonal que transforma la matriz A en una matriz diagonal con sus autovalores en la diagonal [3]

[2] Barzilai y Borwein probaron convergencia Superlineal pero sólo para el caso cuadrático de dimensión 2 [Barzilai & Borwein, 1988].

[3] En un entorno de la solución, la función objetivo puede ser aproximada por una función cuadrática cuya matriz A es simétrica y definida positiva. Entonces todos sus autovalores son reales y positivos, y existe una transformación ortogonal para diagonalizar la matriz A siendo sus autovalores los elementos de la diagonal [Dennis & Schnabel, 1983].

Si los valores λ_i, $i = 1, \ldots, n$ son los n autovalores de la matriz A en 2.3 , entonces la componente i-ésima del gradiente viene dada por

$$g_{k+1}^{(i)} = \left(1 - \frac{\lambda_i}{\alpha_k}\right) g_k^{(i)}$$

En efecto

$$\begin{aligned} g_{k+1} &= Ax_{k+1} + b \\ &= A\left(x_k - \frac{1}{\alpha_k} g_k\right) + b \\ &= Ax_k + b - \frac{1}{\alpha_k} Ag_k \\ &= g_k - \frac{1}{\alpha_k} Ag_k \end{aligned} \tag{2.4}$$

Considerando que $A = diag(\lambda_i)$, tomando la expresión 2.4 componente a componente

$$\begin{aligned} g_{k+1}^{(i)} &= g_k^{(i)} - \frac{\lambda_i}{\alpha_k} g_k^{(i)} \\ &= \left(1 - \frac{\lambda_i}{\alpha_k}\right) g_k^{(i)} \end{aligned}$$

Es claro que si $g_k^{(i)} = 0$ para algún i en la iteración $\bar{k}$ entonces esto se mantendrá para $k > \bar{k}$. Es decir, cada vez que α_k iguale a un autovalor de A λ_i, la componente i-ésima del gradiente se anulará y ya no volverá a modificar su valor.

La extensión al caso no cuadrático es a través del *cociente de Raileigh* [Golub & Van Loan, 1989].

Definición 2.3 *Dada una matriz cuadrada A y un vector v, se denomina cociente de Raileigh del vector v respecto a la matriz A al cociente*

$$r(x) = \frac{v^T A v}{v^T v}$$

.

Este *cociente de Raileigh* $r(x)$ tiene la propiedad

$$\lambda_{min} \leq r(x) \leq \lambda_{max}$$

donde λ_{min} y λ_{max} son el mínimo y máximo autovalor respectivamente de la matriz A.

Para el caso no cuadrático, el valor α_{k+1} es un *cociente de Raileigh* del paso anterior d_k respecto a la matriz Hessiana promedio H

$$H = \int_0^1 \nabla^2 f(x_k + t.d_k)\, \partial t \tag{2.5}$$

En efecto:

$$\int_0^1 \nabla^2 f(x_k + t.d_k) d_k \partial t = g(x_{k+1}) - g(x_k) = y_k,$$

luego

$$\begin{aligned} \alpha_{k+1} &= \frac{d_k^T \left(\int_0^1 \nabla^2 f(x_k + t.d_k) d_k \partial t \right)}{d_k^t d_k} \\ &= \frac{d_k^T (g_{k+1} - g_k)}{d_k^T d_k} \\ &= \frac{d_k^T y_k}{d_k^T d_k} \end{aligned}$$

Con esta elección del paso se busca que el gradiente tienda a ser nulo 'persiguiendo' los autovalores de la matriz Hessiana en la solución.

Friedlander et al. observaron que es importante detectar la presencia de direcciones de descenso que se aproximen a autovectores de la matriz Hessiana [Friedlander *et al.*, 1999]. Glunt et al. ya habían mostrado que en el caso de SPG esta aproximación ocurre frecuentemente [Glunt *et al.*, 1993a]. La longitud del paso 2.1 se asemeja a la longitud del paso del método de *Cauchy* pero aplicada con retardo, es decir, en la iteración k se aplica α_{k-1}. Luengo y Raydan no desechan el uso de esta longitud de paso de *Cauchy* y la incorporaron para establecer una variante que robustece al método. Si la dirección del gradiente es un autovector de la matriz Hessiana, estos autores afirman que la mejor elección es la longitud de paso determinada por *Cauchy*, es decir, sin aplicar retardo [Luengo & Raydan, 2003]. El fundamento es que en el caso cuadrático esto llevaría a la solución en la iteración siguiente. Para detectar esta cercanía entre el gradiente y un autovector se utiliza el *coseno* del ángulo entre la aproximacón H multiplicada por d_k y g_k.

De la ecuación (2.5) tenemos que

$$Hd_k \approx y_k$$

Luego Hd_k será paralelo a g_k si el ángulo δ_k entre los vectores y_k y g_k es cero. Ó bien,

$$cos\delta_k = \frac{|g_k^T y_k|}{\|g_k\|\|y_k\|} \approx 1$$

Esto motivó a incluir la siguiente estrategia para elegir la longitud del paso:

- Si x_k esta 'cerca de la solución' y

  $$\frac{|g_k^T y_k|}{\|g_k\|\|y_k\|} \approx 1$$

  entonces $\alpha_k = \dfrac{d_k^T y_k}{d_k^T d_k}$ (máximo descenso)

  Siguendo el diseño de Luengo y Raydan, un punto x_k se considera 'cerca de la solución' si satisface

  $$\|g_k\|_2 \leq 10^{-2}(1 + |f(x_k)|)$$

  [Luengo & Raydan, 2003].

- Si $cos\delta_k < cos\delta_{k-1}$, cuanto más grande es el coseno más cercano están el gradiente de un autovector y α_k de un autovalor. Entonces se aplica un doble retardo, es decir

  $$\alpha_k = \frac{d_{k-2}^T y_{k-2}}{d_{k-2}^T d_{k-2}}.$$

  Esta elección favorece pasos más largos y mantiene al valor α_k cerca de un autovalor.

- Si ninguna de las situaciones anteriores se da, se elige el paso espectral dado por la ecuación (2.1)

Esta elección de longitud de paso (paso espectral) tiene una semejanza a la longitud del paso determinada en el método de máximo descenso (método de *Cauchy*), pero

como fue expuesto en esta sección, el mecanismo que se produce en la búsqueda del optimizador es completamente diferente. El método naturalmente es no monótono, es decir, ni la función objetivo ni el gradiente decrecen sistemáticamente. De hecho, forzar este decrecimiento echa a perder las propiedades del método. Para asegurar su convergencia global Raydan lo combinó con una búsqueda lineal no monótona [Raydan, 1997].

2.2 Búsqueda lineal no monótona

Los métodos de gradiente proyectado son efectivos para la resolución de problemas de minimización con restricciones de gran tamaño porque requieren poco almacenamiento de datos y pocas operaciones aritméticas [Nocedal & Wright, 1999]. Estos buscan direcciones de descenso pero proyectando sobre la zona de factibilidad. Para asegurar la convergencia de este tipo de algoritmos, Armijo y Goldstein establecieron una condición que se debe cumplir entre dos iterados consecutivos:

$$f(x_k + ts_k) \leq f(x_k) + \gamma t s_k^T g_k$$

Esta condición fuerza a la función a decrecer en cada iterado una cierta proporción de la tasa de decrecimiento inicial. Asegura la convergencia pero arruina la propiedades del paso espectral.

Raydan propone implementar la búsqueda lineal no monótona que habían comenzado a desarrollar Grippo et al.:

$$\left(f(x_+) < F_{max} + \gamma s_k^T g_k\right)$$

donde $F_{max} = \max_{0 \leq j \leq \min\{k, M-1\}} \{F(x_{k-j})\}$ [Grippo *et al.*, 1986]. Es decir, de acuerdo con la definición 2.1, que la razón de decrecimiento se mida en función del máximo valor de $f(x)$ en los últimos M iterados .

El esquema de la subrutina de búsqueda es el que muestra el algoritmo 1

Algoritmo 1 Búsqueda lineal no monótona

Entrada: x_k, $g(x_k)$, σ_k y F_{max}

Salida: x_{k+1}

1: $x_+ = x_k - \sigma_k g(x_k)$

2: **Mientras** $\Big(f(x_+) > F_{max} + \gamma s_k^T g(x_k)\Big)$ **hacer**

3: Definir $\sigma_k = t.\sigma_k$, $t \in (0,1)$

4: $x_+ = P(x_k - \sigma_k g(x_k))$

5: **Fin** {Mientras}

La implementación de esta búsqueda permite asegurar la convergencia del método espectral para funciones en general. Esto queda establecido en los teoremas enunciados en la sección 2.3.

2.3 Convergencia del algoritmo

La prueba de convergencia del algoritmo SPG a un punto estacionario fue realizada por Raydan en [Raydan, 1997], basada en el siguiente teorema de Grippo et al.

Teorema 2.1 *[Grippo* et al.*, 1986] Sea* $\{x_k\}$ *una sucesión definida por*

$$x_{k+1} = x_k + \sigma_k d_k, \;\; d_k \neq 0$$

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

i) el conjunto de nivel $\Omega_0 = \{x : f(x) \leq f(x_0)\}$ *es compacto;*

ii) existen número positivos c_1, c_2 *tales que :*

$$g_k^T d_k \;\; \leq \;\; -c_1 \|g_k\|^2, \tag{2.6}$$

$$\|d_k\| \;\; \leq \;\; c_2 \|g_k\|, \tag{2.7}$$

iii) $\sigma_k = \theta^{h_k} a$, *donde* h_k *es el primer entero no negativo* h *para el cual:*

$$f(x_k + \theta^h a d_k) \leq \max_{0 \leq j \leq m(k)} [f(x_{k-j})] + \gamma \theta^h a g_k^T d_k, \tag{2.8}$$

donde $m(k) = \min[k, M]$, $k \geq 1$.

Entonces:

a) La sucesión $\{x_k\}$ *está contenida* Ω_0 *y todo punto límite* $\bar{x}$ *satisface* $g(\bar{x}) = 0$,

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

c) si el número de puntos estacionarios de f *en* Ω_0 *es finito, 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 el trabajo de Raydan:

Teorema 2.2 *[Raydan, 1997] Sea* $\Omega_0 = \{x : f(x) \leq f(x_0)\}$ *un conjunto compacto. Sea* $f : \mathbb{R}^n \to \mathbb{R}$ *una función continuamente diferenciable en un entorno* N *de* Ω_0*. Sea* $\{x_k\}$ *la sucesión generada por el algoritmo SPG. Entonces o* $g(x_j) = \nabla f(x_j) = 0$ *para algún índice* j*, o valen las siguientes propiedades:*

a) $\lim_{k\to\infty} \|g(x_k)\| = 0$,

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

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

La eficiencia del algoritmo viene dada por su bajo costo en operaciones aritméticas. El análisis se realiza en el capítulo 3 de esta tesis.

Capítulo 3

Análisis del algoritmo SPG

"Antes las distancias eran mayores
porque el espacio se mide por el tiempo".
J.L. Borges

3.1 Acerca de la complejidad

Los problemas de optimización numérica que se plantean en aplicaciones reales se caracterizan por su intratabilidad. La solución de muchos de ellos puede ser aproximada en tiempos razonables gracias a la velocidad de las computadoras actuales, pero no por la existencia de algoritmos realmente eficientes [Vavasis, 1991]. No todos los modelos tienen el mismo grado de complejidad a la hora de ser abordados. Analizando los problemas de optimización como problemas de decisión, la teoría de complejidad computacional nos brinda sus clases de complejidad para clasificar cada problema según su grado de tratabilidad.

Los problemas de optimización con restricciones más simples de tratar son los problemas de programación lineal (LP). Estos son de la forma:

$$\begin{cases} \text{Min } c^T x \\ \text{Sujeto a } Ax = b \\ x \geq 0 \end{cases} \tag{3.1}$$

donde x y c son vectores n-dimensionales, $A \in \mathbb{R}^{m \times n}$, y b es un vector m- dimensional. La desigualdad $x \geq 0$ significa que cada componente de x es no negativa.

En 1979 Khachiyan demostró que el problema de programación lineal puede ser resuelto en tiempo polinomial, es decir, está en la clase P [Khachiyan, 1979].

Cuando la función objetivo en 3.1 es una función cuadrática (en el lugar de $c^T x$ aparece $x^T H x + c^T x$, con H una matriz de $n \times n$), estamos ante lo que se denomina un problema de programación cuadrática. Este tipo de problemas pertenecen a la clase NP [Vavasis, 1990].

Los problemas generales de optimización con restricciones (no lineales ni cuadráticos) son más intratables, en general son NP-Hard [Pardalos & Schnitger, 1988]. Inclusive los problemas cuadráticos con un autovalor negativo son NP-Hard [Pardalos & Vavasis, 1991]

Las medidas de complejidad se expresan a menudo en función del tamaño n del problema a resolver. De manera informal se define como el número de entradas para el cálculo (por ejemplo, en el problema de resolver un sistema de ecuaciones lineales, n es del número de coeficientes del sistema). En el análisis de nuestro algoritmo esta va a ser la unidad de medida que adoptaremos, es decir, la medida de complejidad será en función de n siendo este la dimensión del optimizador que se busca encontrar. [1]

3.2 Costo computacional del algoritmo

Vamos a definir la notación que se utlizará a lo largo de esta tesis. Sea A un subconjunto de $\mathbb{R}$ y sean $f : A \to \mathbb{R}$ y $g : A \to \mathbb{R}$ dos funciones. La notación $f(x) = \mathcal{O}(g(x))$ significa que existe una constante positiva c y un valor x_0 tal que para todo $x \in A$ que satisface $x \geq x_0$ se tiene $|f(x)| \leq cg(x)$. La notación $f(x) = \Omega(g(x))$ significa que existe una

[1] Debe aclararse cuál será la medida de complejidad dado que el tamaÃśo de las entradas podrÃıa medirse en bits, y el tiempo de ejecución en operaciones sobre bits. Muchos problemas computacionales teóricos solo tienen uno o dos número como entrada. (test de primalidad, divisor común mayor), y la dificultad del problema depende de la magnitud de dichos números. No es el caso de problemas NLP donde las clases de complejidad vienen dadas por la dimensión de los problemas y no por la magnitud de los valores involucrados [Vavasis, 1991].

constante positiva c y un valor x_0 tal que para todo $x \in A$ que satisface $x \geq x_0$ se tiene $|f(x)| \geq cg(x)$. La notación $f(x) = \Theta(g(x))$ significa que se verifican $f(x) = \mathcal{O}(g(x))$ y $f(x) = \Omega(g(x))$.

Para establecer el orden de complejidad computacional vamos a tomar la iteración principal del SPG dejando abierto el costo computacional para aquellas subrutinas que van a variar en función del problema a resolver. Entonces el orden de la rutina $evalf$ es $\mathcal{O}(f)$, el de la rutina *Proyeccion* es $\mathcal{O}(P)$ y el de la búsqueda lineal no monótona es $\mathcal{O}(BL)$.

La aproximación del gradiente se describe en la subsección siguiente.

3.2.1 Aproximación del gradiente

Considerando que la función podría no tener una expresión explícita, y que el gradiente negativo es la dirección de descenso del método, la aproximación del gradiente se realiza mediante diferencias finitas centradas:

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i},\ i = 1, \cdots, n \tag{3.2}$$

donde h_i se calcula como $max\{\sqrt{\epsilon_M}, \sqrt{\epsilon_M}|x_i|\}$ siendo ϵ_M el *epsilon machine* de la computadora[2], e_i es el vector unitario i-ésimo y n es la dimensión del vector x. El error de esta aproximación es $\mathcal{O}(h_i^2)$ [Dennis & Schnabel, 1983]. Esta formulación requiere la evaluación de la función objetivo en los puntos $(x+h_i e_i)$ y $(x-h_i e_i)$ para $i = 1..n$. Luego, para un problema $n-$dimensional la evaluación del gradiente involucra $2n$ evaluaciones de función y el orden es $2n\mathcal{O}(f)$.

Esquema general de una iteración

En la descripción del algoritmo SPG (algoritmo 2) se agrega el orden de operaciones aritméticas en cada instancia.

[2] ϵ_M = mínimo $\epsilon > 0$ tal que $1 + \epsilon > 1$ en aritmética de punto flotante

Algoritmo 2 Gradiente espectral proyectado (SPG).

Entrada: $x_0 \in \mathbb{R}^n$ (Punto inicial), M (parámetro de no monotonicidad), σ_{min}, σ_{max} (cotas) y tol (tolerancia). Rutinas evalf, evalg y Proyección.

Salida: x_k tal que $\|g_k(x_k)\| < tol$.

1: $x_k = Proyeccion(x_0)$ $\mathcal{O}(P)$

% Evaluar el gradiente

2: $g_k = evalg(x_k)$ $2n\mathcal{O}(f)$

3: $s_k = -g_k$ $\mathcal{O}(cte)$

% Evaluar la funcion

4: $f_k = evalf(x_k)$; $\mathcal{O}(f)$

% Bucle principal

5: **Mientras** $\|s_k\| > tol$ $\mathcal{O}(n)$

6: $d_k = Proyeccion(x_k - \sigma * g_k) - x_k$; $\mathcal{O}(P)$

7: `Búsqueda lineal no monótona` $\mathcal{O}(BL)$

% Rectificación de la longitud del paso

8: $s_k = t * d_k$ (t es el valor dado por la búsqueda lineal) $\mathcal{O}(n)$

9: $x_k = x_k + s_k$ $\mathcal{O}(n)$

10: $g_{new} = evalg(x_k)$ $2n\mathcal{O}(f)$

11: $y_k = g_{new} - g_k$ $\mathcal{O}(n)$

%Actualización

12: $g_k = g_{new}$ $\mathcal{O}(n)$

13: $skTyk = s_k^t y_k$ $\mathcal{O}(n)$

% Cálculo de la longitud del paso para la iteración siguiente

14: **Si** $skTyk \leq 0$ **entonces**

$\sigma = \sigma_{max}$

15: **sino**

$\sigma = max(min(\frac{s_k^t s_k}{skTyk}, \sigma_{max}), \sigma_{min})$; $\mathcal{O}(n)$

16: **Fin**{Si}

17: **Fin** {Mientras}

El orden de una iteración viene dado por $\max\{\mathcal{O}(n),\ \mathcal{O}(f),\ \mathcal{O}(P),\ \mathcal{O}(BL),\ n\mathcal{O}(f)\}$. Asimismo cada iteración interna de la búsqueda lineal involucra una evaluación de función, luego $\mathcal{O}(BL) = k.\mathcal{O}(f),\ k \in \mathbb{N}$

Como se verá en la sección siguiente, el posible mal condicionamiento y la complejidad de la proyección sobre la región factible se simplifican mediante la combinación con un esquema de lagrangiano aumentado.

3.3 El lagrangiano aumentado

Uno de los inconvenientes más importantes en estos métodos de gradiente proyectado es la sobrecarga de cálculo en cada iteración que puede representar la proyección sobre la región factible [Bertsekas & Tsitsiklis, 1995]. Esto es fuertemente dependiente de las características de esta región (el número de ecuaciones que la describen, o el grado de no linealidad de dichas ecuaciones). A partir de esta situación, Diniz-Ehrhardt et al. proponen combinar el método SPG con un lagrangiano aumentado [Diniz-Ehrhardt *et al.*, 2004].

Consideremos el problema

$$\begin{cases} \text{Mínimo de } f(x) & \\ \quad \text{sujeto a} & \\ \quad c_i(x) = 0 \qquad & i = 1 \cdots n_i \\ \quad c_j \geq 0 \qquad & j = 1 \cdots n_j \\ \quad l_i \leq x_i \leq u_i & \end{cases} \tag{3.3}$$

Si se introducen en las restricciones de desigualdad variables de holgura [3], estas restricciones devienen en restricciones de igualdad: $c_j(x) - s_j = 0$. La factibilidad de región donde se busca el óptimo se mantendrá siempre que $s_j \geq 0$. La introducción de estas variables aumenta la dimensión del problema, pero simplifica la proyección sobre la región factible. En efecto $Proy(s_j) = max\{0, s_j\}$

[3] También llamadas con su denominación original del inglés variables *slack*.

El problema ahora solo tiene restricciones de igualdad y cotas en las variables

$$\begin{cases} \text{Mínimo de } f(x) \\ \quad \text{sujeto a} \\ \quad c_i(x) = 0 \qquad i = 1 \cdots n_i \\ c_j(x) - s_j = 0 \quad j = 1 \cdots n_j \\ \quad l_i \leq x_i \leq u_i \\ \quad s_j \geq 0 \end{cases} \tag{3.4}$$

Para problemas de optimización con restricciones, Lagrange demostró que existe un vector[4] $\lambda \in R^m$ que verifica que el optimizador buscado es al mismo tiempo optimizador de la función sin restricciones

$$l(x, \lambda) = f(x) + C(x)^T \lambda \tag{3.5}$$

donde $C(x)$ es la función vectorial que reune, en este caso, todas las restricciones de igualdad

$$C(x) = \{c_i(x)\} \cup \{c_j(x) - s_j\}$$

El método del lagrangiano aumentado consiste en agregar a esta función de Lagrange un término que penaliza la no factibilidad del punto con el que se está iterando. A partir de una estimación inicial de los multiplicadores de Lagrange se minimiza según x la función

$$L(x, \lambda, \rho) = f(x) + C(x)^T \lambda + (\rho/2).\|C(x)\|^2 \tag{3.6}$$

donde $\lambda \in \Re^m$ es la estimación del vector de multiplicadores de Lagrange, $\rho > 0$ es el parámetro de penalización y $\|.\|$ es la norma euclídea. Es decir, se busca resolver

$$\begin{cases} \min\limits_x L(x, \lambda, \rho) \\ \quad \text{sujeto a} \\ \quad l_i \leq x_i \leq u_i \\ \quad s_j \geq 0 \end{cases} \tag{3.7}$$

[4]Los llamados multiplicadores de Lagrange. Publicado en 1758 en *Miscellanea Taurinensia.*

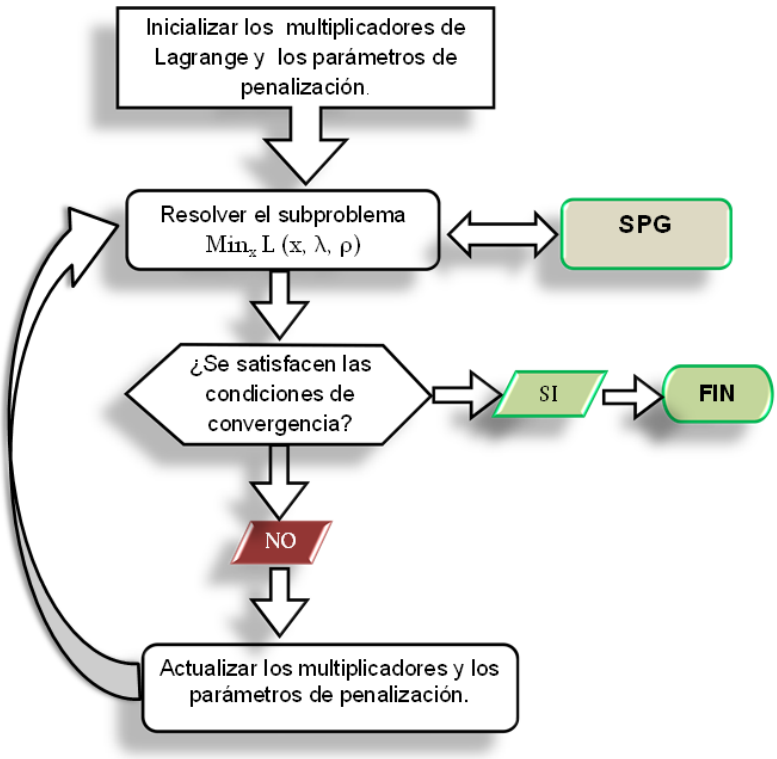


Figura 3.1: Diagrama de flujo del algoritmo ALSPG

La solución parcial obtenida permite ajustar la estimación de los multiplicadores. A partir de este ajuste y con una posible modificación en el parámetro de penalización, se vuelve a resolver el problema 3.7, que pasa a ser un subproblema dentro de una estructura mayor. Diniz-Ehrhardt et al. implementaron el método SPG para obtener las sucesivas soluciones de este subproblema, a este algoritmo general lo denominaron ALSPG por la sigla para Augmented Lagrangian and Spectral Projected Gradient [Diniz-Ehrhardt *et al.*, 2004]. La descripción de ALSPG viene dada por el algoritmo 3 cuya estructura se esquematiza en la figura 3.1.

Esta función lagrangiano aumentado, introducida por Fletcher , reduce la posiblidad de generar, en el paso 1, subproblemas mal condicionados [Fletcher, 1987]. Esto se da a través de la estimación explícita de los multiplicadores de Lagrange [Nocedal & Wright, 1999].

Teniendo en cuenta las cotas sobre las variables en las que se busca el óptimo, la

Algoritmo 3 Lagrangiano aumentado con SPG (ALSPG).

Entrada: $x_0 \in \mathbb{R}^n$ (Punto inicial)

$\Lambda > 0$ (Cota para el vector de multiplicadores),

$\lambda_0 \in \mathbb{R}^m$ tal que $\|\lambda_0\| \leq \Lambda$ (Valor incial de los multiplicadores),

$\rho_0 > 0$ (parámetro de penalización inicial),

$\epsilon_0 > 0$ (tolerancia inicial respecto a la optmalidad),

$\epsilon_\star > 0$ (tolerancia final requerida de optimalidad)

$\nu_0 > 0$ (tolerancia inicial respecto a la factibilidad),

$\nu_\star > 0$ (tolerancia final requerida de factibilidad)

Salida: Solución factible, es decir $x_\star$ tal que $\|\nabla L(x_\star)\| < \epsilon_\star$ y $\|C(x_\star)\| < \nu_\star$

1: Resolver el subproblema 3.7 mediante el `Algoritmo SPG` con una tolerancia ϵ_k siendo $\bar{w}$ la solución obtenida.

% Chequear el criterio de parada.

2: $x_{k+1} = \bar{w}$

3: **Si** $\|C(x_{k+1})\| \leq \nu_\star$ y $\epsilon_k \leq \epsilon_\star$ **entonces**

TERMINAR siendo x_{k+1}la solución de 3.3.

4: **Fin** {Si}

% Actualizar los multiplicadores

5: $\lambda_{k+1} = \lambda_k + \eta_k C(x_{x+1})$.

6: **Si** $\lambda_{k+1} \geq \Lambda$ **entonces**

$\lambda_{k+1} = \lambda_k$

7: **Fin** {Si}

% Actualizar el parámetro de penalización

8: **Si** $\|C(x_{k+1})\|_\infty \leq 0.1\|C(x_k)\|_\infty$ o $\|C(x_{k+1})\|_\infty \leq \eta_k$ **entonces**

$\eta_{k+1} = \eta_k$

9: **Si no**

$\eta_{k+1} = 10\eta_k$

10: **Fin** {Si}

% Actualizar las tolerancias.

11: Mediante alguna estrategia heurística determinar ϵ_k y η_k de forma tal que

$\epsilon_\star \leq \epsilon_{k+1} < \epsilon_k$ y $\eta_\star \leq \eta_{k+1} < \eta_k$.

12: $k = k + 1$. Volver al paso 1.

proyección sobre la región factible es

$$Proy(x,s) = \begin{cases} max\{min\{u_i, x_i\}, l_i\},\ i = 1 \cdots n \\ max\{0, s_j\},\ j = 1 \cdots n_j \end{cases}$$

que es un procedimiento de orden $\mathcal{O}(n + n_j)$.

Finalmente, la relación entre la solución arrojada por el algoritmo y la solución del problema 3.3 viene dada por la siguiente proposición:

Proposición 3.1 *El punto $(x_\star, \lambda_\star)$ es un punto estacionario de $l(x,\lambda)$ si y sólo si es un punto estacionario de $L(x,\lambda,\rho)$*

Demostración: Para la función de Lagrange tenemos

$$\nabla_x l(x,\lambda) = \nabla f(x) + [\nabla C(x)]\lambda$$

$$\nabla_\lambda l(x,\lambda) = C(x)$$

y para el lagrangiano aumentado resulta

$$\nabla_x L(x,\lambda,\rho) = \nabla_x l(x,\lambda) + \frac{1}{\rho}[\nabla C(x)]C(x)$$

$$\nabla_\lambda L(x,\lambda,\rho) = C(x)$$

Si $\nabla_x l(x_\star,\lambda_\star) = 0$ y $\nabla_\lambda l(x_\star,\lambda_\star) = 0$ resulta que $\nabla_x L(x_\star,\lambda_\star,\rho) = 0$ y $\nabla_\lambda L(x_\star,\lambda_\star,\rho) = 0$. Recíprocamente, si $\nabla_x L(x_\star,\lambda_\star,\rho) = 0$ y $\nabla_\lambda L(x_\star,\lambda_\star,\rho) = 0$ se tiene que $\nabla_x l(x_\star,\lambda_\star) = 0$ y $\nabla_\lambda l(x_\star,\lambda_\star) = 0$ ∎

3.4 Costo aritmético de funciones habituales

Como se puede ver en la descripción del algoritmo 2, el costo aritmético de la función general a optimizar ocupa un lugar preponderante en la performance del método. En la medida en que evaluar estas funciones no sea un problema 'duro', es decir, su orden de complejidad no supere el orden polinomial, entonces este orden será el del método SPG. En esta sección se describe el costo de algunas funciones que habitualmente se presentan en los problemas de optimización en ingeniería de procesos.

Operaciones de álgebra lineal

Si consideramos $c \in \mathbb{R}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$ y $A \in R^{n \times n}$, la siguiente tabla muestra el costo aritmético de algunas operaciones elementales del álgebra lineal.

Operación	Sumas	Productos	Orden
cx	$n-1$	n	$\mathcal{O}(n)$
$x^T b$	$n-1$	n	$\mathcal{O}(n)$
Ax	$n(n-1)$	n^2	$\mathcal{O}(n^2)$
$x^T Ax$	$(n-1)(n+1)$	n^2+n	$\mathcal{O}(n^2)$

Entonces una expresión lineal $l(x) = Ax + b$ es de orden $\mathcal{O}(n^2)$.

Una expresión cuadrática $q(x) = \frac{1}{2}x^T Ax + b^T x + c$ tiene orden de complejidad $\mathcal{O}(n^2)$.

Polinomio de grado n

Utilizando el método de evaluación anidada[5], el polinomio de grado n

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_n x^n$$

se puede escribir como

$$p(x) = a_0 + x(a_1 + x(a_2 + x(\ldots + a_n x^n))\ldots)$$

De esta forma la evaluación insume n sumas y n productos, siendo el procedimiento de evaluación de orden $\mathcal{O}(n)$

La importancia de la evaluación de polinomios en forma eficiente radica en que toda función real continua en un intervalo cerrado y acotado, puede aproximarse por un polinomio de coeficientes reales. A este resultado se lo conoce como como el *teorema de aproximación de Weierstrass.*

Teorema 3.1 *Si $f : R \to R$ es una función continua sobre un intervalo $[a, b] \subset R$, y*

[5]LLamada regla de Horner

$\epsilon > 0$, *existe un polinomio de coeficientes reales,* $p(x)$, *tal que* $\forall x \in [a, b]$ *es*

$$|f(x) - p(x)| < \epsilon$$

Expresión A.sen(x) + B.cos (x)

Dado $x \in \mathbb{R}$ un ángulo en radianes ($0 \leq x \leq 2\pi$), el desarrollo en serie de McLaurin del $sen(x)$ es

$$sen\ (x) = \sum_{i=0}^{k} (-1)^i \frac{x^{2i+1}}{(2i+1)!} + R_k \tag{3.8}$$

donde R_k representa el el error que se comete al truncar la serie infinita.

Se puede observar que para dos términos consecutivos de 3.8 se tienen las siguientes expresiones:

$$u_{k-1} = (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!} \qquad u_k = (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

Se cumple entonces la siguiente relación de recurrencia

$$u_k = -\frac{x^2}{2k(2k+1)} u_{k-1}$$

Esto significa que a partir del término u_{k-1} solo son necesarias 6 operaciones para obtener u_k (x^2 se calcula al principio por única vez).

Por otro lado la precisión con que se efectúa el desarrollo viene dada por el término R_k, que se denomina *error residual*:

$$R_k = \frac{cos(\xi) 3.15^{2k+3}}{(2k+3)!}$$

Acotando este error residual y forzándolo a ser menor que una cierta tolerancia se puede determinar el número k de términos necesarios en la aproximación:

$$|R_k| < \frac{(3.15)^{2k+3}}{(2k+3)!} < tolerancia$$

Considerando que $|R_{12}| < 2.6 \times 10^{-15}$ y $|R_{13}| < 3.1 \times 10^{-17}$ entonces para obtener una aproximación con un error menor a 10^{-16} (el *epsilon machine* estándar de las PC) son necesarios 13 términos en la serie de McLaurin, lo que da un total de 78 operaciones

para aproximar $sen(x)$. Análogamente se calcula el $cos(x)$ con 78 operaciones. Luego la expresión $Asen(x) + Bcos(x)$ insume $2 \times 78 + 3$ operaciones, siendo la expresión de orden $\mathcal{O}(cte)$.

La importancia de la evaluación de este tipo de expresiones viene dada por el *teorema de aproximación de Weierstrass de funciones periódicas.*

Teorema 3.2 *Si $f : R \to R$ es una función periódica continua de período 2π, entonces dado $\epsilon > 0$, existe un $n \in N$ y una suma trigonométrica*

$$S_n(x) = a_0 + \sum_{k=1}^{n} a_k cos(kx) + b_k sen(kx)$$

tal que $\forall x \in [a, b]$ es

$$|f(x) - S_n(x)| < \epsilon.$$

Capítulo 4

El método SPG para optimización rigurosa de plantas de procesos industriales

"Podría parecer que hemos llegado a los límites alcanzables por la tecnología informática, aunque uno debe ser prudente con estas afirmaciones, pues tienden a sonar bastante tontas en cinco años."

J. Von Neumann

Hemos aplicado el algoritmo SPG descripto en el Capítulo 2 para la optimización de varios casos académicos e industriales, con el objeto de verificar su desempeño en problemas cuyos modelos de simulación exhiben variados niveles de complejidad como son la cantidad de variables, el número de restricciones y su grado de no linealidad (cuadrática, polinomial o general).

Para analizar el desempeño del Algoritmo 2 se seleccionó el conjunto de problemas de prueba que se describe a continuación, el cual incluye ejemplos académicos e industriales.

Los problemas académicos considerados son 15 casos de estudio extraídos de la batería de problemas diseñada por Hock y Schittkowski [Hock & Schittkowski, 1981]. Todos los problemas elegidos poseen restricciones de igualdad y sus características

generales se resumen en la Tabla 4.1. Los casos industriales analizados incluyen la optimización económica de una planta de producción de benceno por hidrodealquilación de tolueno (problema HDA), el diseño óptimo de las características geométricas de intercambiadores de calor con el objeto de maximizar el intercambio de energía (Problema RED) y la maximización de las ganancias para una planta de producción de cloruro de etilo (Problema CLE). La descripción de estos problemas se puede consultar en el Anexo B de esta tesis. Estos problemas tienen un número mayor de variables en comparación con los casos académicos analizados en este capítulo y figuran con su sigla en la Tabla 4.1. La primera columna de esta tabla corresponde al código de problema asignado en el texto de referencia; la segunda contiene la cantidad de variables de optimización del problema; la tercera es su número de restricciones y las dos siguientes contienen, respectivamente, las características de la función objetivo y de las restricciones. La sexta columna indica si la solución se conoce en forma exacta (problema teórico) o si, por el contrario, sólo se cuenta con soluciones numéricas (problema práctico). La séptima columna indica el número de cotas sobre las variables y finalmente, la última columna muestra el número de condición $\kappa(H)$ de la matriz Hessiana en el punto óptimo (un cero en esta columna indica que la matriz Hessiana es singular).

Para aplicar el algoritmo SPG se explicitaron las restricciones de forma tal que las proyecciones en los pasos 1 y 6 del Algoritmo 2 se realizaron sólo sobre el hipercubo que representan las cotas. Para ilustrar esta reformulación se muestra como se explicitó el problema que figura con el número 53 .

Ejemplo 4.1 *El problema 53 está planteado de la siguiente forma:*

$$\min(x_1 - x_2)^2 + (x_2 + x_3 - 2)^2 + (x_4 - 1)^2 + (x_5 - 1)^2 \tag{4.1}$$

$$\textit{sujeto a } x_1 + 3x_2 = 0 \tag{4.2}$$

$$x_3 + x_4 - 2x_5 = 0 \tag{4.3}$$

$$x_2 - x_5 = 0 \tag{4.4}$$

$$-10 \leq x_i \leq 10 \qquad i = 1, \ldots, 5 \tag{4.5}$$

Problema	Dim.	N^o de Restr.	F. Objetivo	Restr.	Sol.	Cotas	$\kappa(H)$
26	3	1	Polinomial	Polinomial	Teórica	0	4
27	3	1	Polinomial	Cuadráticas	Teórica	0	25
28	3	1	Cuadrática	Lineales	Teórica	0	6.5
46	5	2	Polinomial	General	Teórica	0	4×10^{8}
47	5	3	Polinomial	Polinomial	Teórica	0	3.9
48	5	2	General	Lineales	Teórica	0	2.7
49	5	2	Polinomial	Lineales	Teórica	0	5.7×10^{10}
50	5	3	Polinomial	Lineales	Teórica	0	3.6
51	5	3	Cuadrática	Lineales	Teórica	0	1.8
52	5	3	Cuadrática	Lineales	Teórica	0	13.5
53	5	3	Cuadrática	Lineales	Teórica	10	1.8
55	6	6	Cuadrática	Lineales	Teórica	8	0
60	3	1	Polinomial	Polinomial	Práctica	10	2.8
77	5	2	Polinomial	General	Práctica	0	5.2
79	5	3	Polinomial	Polinomial	Práctica	0	2.9
HDA	19	16	General	General	Práctica	6	23
RED	60	86	General	General	Práctica	52	6.4
CLE	33	33	General	General	Práctica	4	3.9

Tabla 4.1: Lista condensada de los ejemplos considerados.

Entonces, de (4.4) se tiene

$$x_5 = x_2,$$

de (4.3) obtenemos

$$x_4 = 2x_2 - x_3,$$

y de (4.2) se puede escribir

$$x_1 = -3x_2.$$

De esta forma el problema se convierte en

$$\min 16x_2^2 + (x_2 + x_3 - 2)^2 + (2x_2 - x_3 - 1)^2 + (x_2 - 1)^2$$

$$\textit{sujeto a } -\frac{10}{3} \leq x_i \leq \frac{10}{3} \quad i = 2, 3$$

Aplicar este procedimiento no siempre es posible. Debe cumplirse que

i- exista expresión explícita de alguna variable en función de las otras en las restricciones (siempre es posible en las restricciones lineales),

ii- el problema no sea demasiado grande.

Los problemas de optimización fueron resueltos con el algoritmo SPG y con los métodos SNOPT, MINOS5, PATHNLP, LINDO, CONOPT, KNITRO y BARON provistos por la plataforma de modelado GAMS [GAMS, 2013]. Se utilizó una PC Pentium MMX Intel de 233Mhz y 64 Mb, utilizando una tolerancia de 10^{-9} como criterio de cero para la norma de la función objetivo. La Tabla 4 muestra el desempeño de los algoritmos testeados en términos de la cantidad de iteraciones y el tiempo de CPU requeridos por cada uno de los métodos para alcanzar convergencia. Cada problema figura con la numeración original con la que fue publicado [Hock & Schittkowski, 1981], o con la sigla indicada en esta sección. En las columnas correspondientes a cada resolvedor se reportan los valores de tiempo de cómputo en segundos, seguido entre paréntesis del número de iteraciones efectuadas por problema commpleto. Se destaca en negrita el menor tiempo insumido en la resolución de cada problema. La abreviatura *nf* indica que el método no alcanza una solución factible, *Maxit* indica que el método alcanzó el tope de iteraciones, y *ns* indica que el método no pudo hallar una solución debido a que la formulación no se adapta a la estructura del resolvedor.

Como fué descripto en el Capítulo 2, el número de condición de la matriz Hessiana en el punto óptimo tiene influencia directa en la cantidad de iteraciones y, por ende, en la velocidad de convergencia de SPG. Los casos testigos son los problemas n^o 46 y 49, cuyos números de condición son del orden de 10^8 y 10^{11}, respectivamente, y el problema n^o 55, cuyo Hessiano es singular. En los tres casos el número de iteraciones requeridas por SPG superó las 300.

Para analizar más globalmente los resultados de la Tabla 4 es necesario un índice de referencia para medir y comparar de alguna manera la competitividad de un algoritmo frente a otros.

Prob. nº	SPG.	CONOPT	MINOS	BARON	IPOPT	LINDO	PATHNLP	SNOPT	KNITRO
26	0.27 (83)	0.22 (17)	0.263 (79)	0.27 (2)	**0.032 (25)**	2 (15286)	0.07 (17)	0.235 (25)	0.045 (19)
27	0.53 (41)	0.83 (25)	0.293 (75)	0.24 (2)	nf	**0.032 (26)**	ns	Maxit	nf
28	0.82 (9)	0.11 (7)	0.28 (3)	0.53 (2)	**0.015 (1)**	0.02 (3)	0.05 (2)	**0.015 (6)**	0.107 (2)
46	1.12 (475)	0.38 (42)	0.229 (95)	ns	0.93 (26)	0.2 (44)	**0.07 (17)**	0.219 (3)	0.11 (18)
47	0.28 (150)	0.285 (28)	0.28 (39)	0.58 (13)	0.063 (19)	Maxit	**0.06 (20)**	1.75 (24)	0.156 (17)
48	0.32 (13)	0.28 (8)	0.295 (7)	0.15 (2)	**0.046 (2)**	0.01 (4)	0.06 (2)	0.73 (10)	0.093 (2)
49	2.23 (708)	0.28 (22)	0.28 (21)	0.15 (2)	0.062 (19)	0.124 (30)	0.07 (15)	**0.03 (14)**	0.051 (16)
50	0.71 (28)	0.11 (7)	0.269 (5)	0.21 (2)	**0.031 (9)**	0.219 (22)	0.07 (8)	0.187 (9)	0.113 (8)
51	0.15 (10)	**0.06 (7)**	0.27 (5)	0.15 (2)	0.25 (2)	0.78 (5)	**0.06 (2)**	0.11 (4)	0.32 (2)
52	0.16 (9)	0.11 (7)	0.28 (5)	0.1 (2)	0.32 (2)	0.25 (4)	**0.05 (2)**	0.1 (5)	0.12 (2)
53	0.16 (13)	0.17 (9)	0.276 (5)	0.2 (2)	0.23 (6)	0.219 (6)	**0.05 (2)**	0.09 (6)	0.125 (3)
55	1.31 (350)	0.29 (7)	0.272 (2)	**0.06 (2)**	0.359 (2)	1.26 (27)	0.08 (7)	0.1 (3)	0.34 (4)
60	0.17 (20)	nf	nf	0.1 (2)	0.32 (6)	0.688 (98)	0.07 (5)	**0.062 (12)**	0.086 (7)
77	0.26 (190)	0.30 (20)	nf	ns	0.766 (11)	0.562 (61)	**0.06 (10)**	0.109 (23)	0.312 (8)
79	0.2 (148)	0.28 (8)	nf	0.1 (2)	0.743 (4)	0.437 (69)	**0.06 (3)**	0.204 (13)	0.125 (4)
HDA	0.88 (23)	0.28 (59)	0.36 (280)	**0.25 (3)**	ns	2.45 (53)	ns	0.54 (65)	ns
RED	0.39 (2)	**0.16 (6)**	0.31 (22)	1.92 (19)	2.07 (131)	0.547 (58)	0.33 (260)	0.39 (75)	nf
CLE	0.56 (159)	0.3 (19)	**0.29 (14)**	ns	nf	ns	0.42 (341)	nf	nf

Tabla 4.2: Desempeño de SPG frente a los algoritmos de la plataforma GAMS

4.1 Evaluación comparativa del software de optimización

La interpretación y análisis de los datos generados en un proceso de evaluación comparativa pueden ser muy variados. La mayoría de los esfuerzos en este campo implican le exhibición de tablas donde se muestra el rendimiento de cada resolvedor en cada problema para un conjunto de métricas tales como el tiempo de CPU y el número de iteraciones. No mostrar estas tablas sería una omisión importante, pero al mismo tiempo, cuando hay gran cantidad de problemas y resolvedores a prueba, la cantidad de datos tiende a ser abrumadora, y la interpretación de los resultados de estas tablas es a menudo una fuente de desacuerdo.

Dolan y More [Dolan & More, 2002] diseñaron un perfil de desempeño como herramienta para evaluar y comparar el rendimiento de diferentes procedimientos de optimización. Este perfil de rendimiento para un resolvedor es la función de distribución acumulada para una métrica de rendimiento escogida. En este trabajo utilizamos el tiempo de cómputo como medida de performance, pero las ideas desarrolladas pueden ser usadas con otras medidas (por ejemplo evaluaciones de función, iteraciones, etc)

Definición 4.1 *Sea $\mathcal{P}$ un conjunto de problemas y $\mathcal{S}$ un conjunto de resolvedores. Para cada problema $p \in \mathcal{P}$ y cada resolvedor $s \in \mathcal{S}$ se define $t_{p,s}$ como el tiempo requerido para resolver el problema p por el resolvedor s.*

Comparamos la performance del resolvedor s sobre el problema p a través de la mejor performance obtenida con todos los resolvedores de $\mathcal{S}$, esto es, se utiliza la razón de performance dada por la ecuación 4.6

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}} \tag{4.6}$$

Se debe fijar un parámetro de penalización $r_M > r_{p,s}$ que se asignará cuando un resolvedor falla. En otras palabras, para todo p y s elegidos, $r_{p,s} = r_M$ si y solo si el resolvedor s no pudo resolver el problema p. La elección de este parámetro depende

del análisis que se quiere hacer, es decir, prestando atención a aspectos tales como practicidad, celeridad, sensibilidad, etc.

Definición 4.2 *La función*

$$\rho_s(\tau) = \frac{1}{n_p} Cantidad\{p \in \mathcal{P} : r_{p,s} < \tau\}, \tag{4.7}$$

donde n_p es el número de problemas que se está considerando, es la probabilidad de que la razón de performance $r_{p,s}$ del resolvedor $s \in \mathcal{S}$ sea un factor τ de la mejor razón de performance posible. La función ρ_s es la función de distribución acumulada para la razón de performance.

A partir de los resultados expuestos en la Tabla 4, utilizando como parámetro de penalización $r_M = 100$, obtuvimos los perfiles de desempeño que muestra la Figura 4.1.

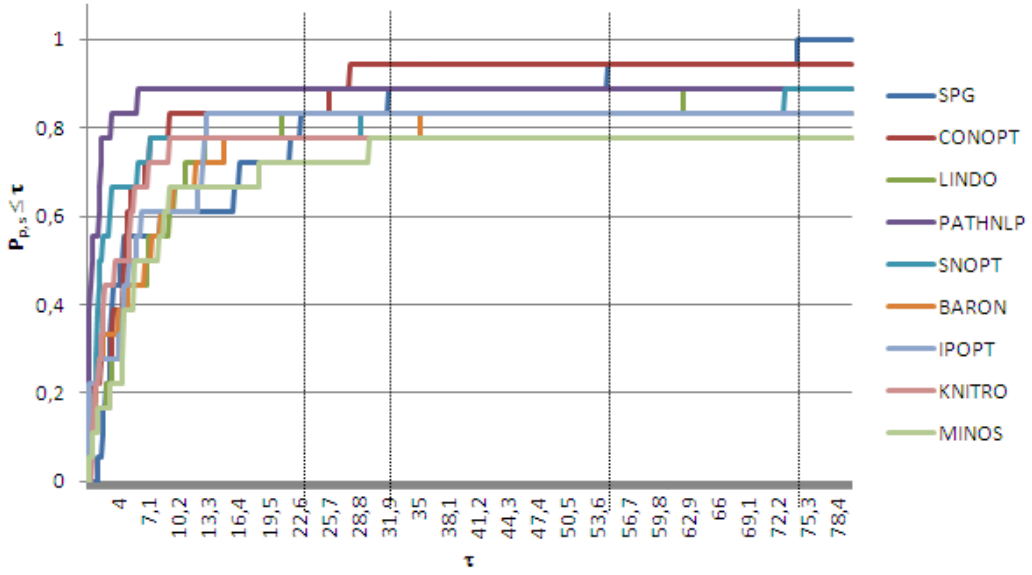


Figura 4.1: Perfiles de desempeño

Los marcadores indican cómo se categorizan los resolvedores a partir de sus probabilidades de éxito.

En la obtención de estos perfiles fue fundamental la elección del parámetro de penalización r_M. Una forma de penalizar un problema no resuelto es asignarle el peor

$\rho_s(23)$		$\rho_s(32)$		$\rho_s(55)$		$\rho_s(75)$	
PATHNLP	0.88	CONOPT	0.94	CONOPT	0.94	SPG	1
SPG	0.83	SPG	0.88	SPG		CONOPT	0.94
CONOPT		PATHNLP		PATHNLP	0.88	LINDO	0.88
LINDO		LINDO	0.83	LINDO	0.83	PATHNLP	
IPOPT		SNOPT		SNOPT		SNOPT	
SNOPT	0.77	IPOPT		BARON		BARON	0.83
BARON		BARON	0.77	IPOPT		IPOPT	
KNITRO		KNITRO		KNITRO	0.77	KNITRO	0.77
MINOS	0.72	MINOS		MINOS		MINOS	

Tabla 4.3: Categorización de resolvedores con $r_M = 100$

tiempo de ejecución apuntado para ese problema. Esto hace equivalente un problema no resuelto a la peor performance registrada. Este es un análisis en el que un resolvedor puede compensar un problema no resuelto con muchos resueltos en forma eficiente. Esta especie de 'promedio' de performance es una forma de medir y comparar, y puede ubicar al resolvedor en un lugar más alto en la categorización. De hecho, utilizando los mismos marcadores que se usaron en la Tabla 4.1 pero aplicando $r_{M_p} = \max\{r_{p,s} : s \in \mathcal{S}\}$ la ubicación de los resolvedores se modifica, como se puede ver en la Tabla 4.1. De hecho aparecen resolvedores que alcanzan valor de probabilidad ρ_s igual a 1 sin haber resuelto todos los problemas planteados.

Al elegir $r_M = 100$, la sanción es marcadamente alta y no pueden ser compensados los problemas que no son resueltos. Este es el criterio de elección que hemos asumido en este trabajo de tesis ya que le da preponderancia a la eficacia por sobre la eficiencia. Luego los algoritmos que ocupen los primeros puestos serán indefectiblemente los más eficaces, y luego se ordenarán a partir de su eficiencia. Los resultados expuestos en la Tabla 4.1 evidencian entonces uno de los argumentos de esta tesis: el algoritmo SPG en ninguno de los casos obtuvo el menor tiempo de ejecución ni la menor cantidad de iteraciones, pero alcanzó soluciones satisfactorias en todos los casos de estudio, lo que

$\rho_s(23)$		$\rho_s(32)$		$\rho_s(55)$		$\rho_s(75)$	
MINOS	0.94	CONOPT	1	CONOPT	1	SPG	1
IPOPT		LINDO	0.94	PATHNLP		CONOPT	
SNOPT		PATHNLP		SPG	0.94	PATHNLP	
PATHNLP		SNOPT		LINDO		BARON	
CONOPT	0.88	BARON		SNOPT		SNOPT	0.94
LINDO		IPOPT		BARON		LINDO	
BARON		MINOS		IPOPT	0.77	KNITRO	
SPG	0.83	SPG	0.88	MINOS		IPOPT	
KNITRO		KNITRO		KNITRO		MINOS	

Tabla 4.4: Categorización de resolvedores con $r_{M_p} = \max\{r_{p,s} : s \in \mathcal{S}\}$

lo ubica, a partir de un criterio de eficacia, en los primeros puestos de este ranking.

Por otro lado, si bien SPG realiza muchas iteraciones, estas son de muy bajo costo. Esto se puede apreciar en la Tabla 4.5 donde se muestra la razón entre el número de iteraciones y el tiempo de CPU requeridos para lograr convergencia. Se destaca en negrita el valor más alto registrado por cada problema. SPG obtiene el mayor valor en el 44.4% de los problemas lo cual implica que el costo por iteración de este procedimiento tiende a ser más bajo.

El análisis que hemos realizado acerca del rendimiento de SPG aplicado a problemas de ingeniería mostró que el algoritmo resultó adecuado y competitivo para problemas de diseño y condiciones operativas de plantas de procesos industriales por sus mínimos requerimientos de memoria y de operaciones aritméticas.

Un método es paralelizable cuando es posible dividir los pasos del proceso de solución en varias etapas que pueden ser efectuadas simultáneamente. En general, los algoritmos paralelos constituyen una alternativa atractiva y viable cuando existe un método paralelizable que consume mucho tiempo para ofrecer una solución óptima, o bien cuando se desea agilizar los pasos intermedios para obtener una solución inicial previa a la aplicación de otra técnica.

Prob N^o	SPG	CONOPT	MINOS	BARON	IPOPT	LINDO	PATHNLP	SNOPT	KNITRO
26	**307.41**	77.27	300.38	7.41	781.25	7643	242.86	106.38	422.22
27	77.36	30.12	8.12	8.33	–	**812.5**	–	–	–
28	10.98	63.64	10.71	3.77	66.67	150	40	**400**	18.69
46	**424.11**	110.53	413.04	–	27.96	220	242.86	13.7	163.64
47	**535.71**	100	139.29	22.41	301.59	–	333.33	13.71	108.97
48	40.63	28.57	23.33	13.33	43.48	**400**	33.33	13.7	21.51
49	317.49	78.57	75	13.33	306.45	241.94	214.29	**466.67**	313.73
50	39.44	63.64	18.52	9.52	**290.32**	100.46	114.29	48.13	70.8
51	66.67	**116.67**	18.52	13.33	8	6.41	33.33	36.36	6.25
52	56.25	**63.64**	17.86	20	6.25	16	40	50	16.67
53	**81.25**	52.94	17.86	10	26.09	27.4	40	66.67	24
55	**267.18**	24.14	7.14	33.33	5.57	21.43	87.5	30	11.76
60	117.65	–	–	20	18.75	**142.44**	71.43	193.55	81.4
77	**730.77**	66.67	–	–	14.36	108.54	166.67	211.01	25.64
79	**740**	28.57	–	20	5.38	157.89	50	63.73	32
HAD	26.14	210.71	**777.78**	12	–	21.63	–	120.37	–
RED	5.13	37.5	70.97	9.9	63.29	106.03	**787.88**	192.31	–
CLE	**283.93**	63.33	48.28	–	–	–	811.9	–	–

Tabla 4.5: Razón número de iteraciones/tiempo

La rapidez del método SPG para realizar cada una de sus iteraciones es una de sus características más atractivas para resolver problemas de gran escala. Sin embargo, debería subsanarse el problema de que, para alcanzar el óptimo, SPG comparativamente requiere un gran número de iteraciones. Dado que el número de evaluaciones de función en este algoritmo es proporcional al número de iteraciones realizadas, el método podría volverse ineficiente en problemas donde evaluar la función objetivo o las restricciones fuera muy costoso. La exigencia de muchas búsquedas lineales para obtener un nuevo iterado, o la necesidad de una buena aproximación del gradiente cuando éste no está disponible pueden volverse verdaderos cuellos de botella del método. Dado que estos

dos sectores del algoritmo son paralelizables[1], los tiempos de ejecución ligados al gran tamaño de los problemas pueden ser mejorados notablemente, ganando en eficiencia pero conservando la gran eficacia exhibida. 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 lograr este objetivo.

[1]Este concepto se desarrolla más ampliamente en el capítulo 6

Capítulo 5

El paradigma paralelo

"La disponibilidad de potentes computadoras paralelas
está generando interés en nuevos tipos de problemas
que no se habían tratado en el pasado".
D. Bertsekas

La resolución de modelos numéricos y simulación de problemas en ciencia e ingeniería demandan una velocidad computacional cada vez más importante. Los enfoques rigurosos para el tratamiento de problemas complejos y el requerimiento de cálculos repetitivos en gran cantidad de datos para lograr resultados válidos, conducen a formulaciones cuyo procesamiento eficiente requiere mayor poder de cómputo del que una computadora con un único procesador podría proveer. Una forma de superar esta limitación es mejorar la velocidad de los procesadores y componentes de manera que puedan ofrecer la capacidad de cómputo requerida. Si bien esto es posible, las mejoras alcanzables en el diseño de hardware están restringidas por la velocidad de la luz, las leyes termodinámicas y un gran costo de inversión asociado al diseño y proceso de fabricación de nuevos procesadores [El-Rewini & Lewis, 1998].

Una forma de incrementar el poder computacional es utilizar múltiples procesadores operando juntos en un único problema. El problema 'general' es dividido en partes, cada una de la cual es tratada por un procesador separado en paralelo. Escribir programas

de esta forma es lo que se denomina *programacion paralela.*

No hay ningún sistema de programación paralela universal. Realizar programas de estas características sigue siendo difícil y requiere lenguajes especiales de programación. Existe una gran variedad de estos lenguajes y herramientas lo que permite, no con poco esfuerzo, construir sistemas paralelos altamente distribuidos, pero aún queda mucho camino por recorrer. El-Rewini y Lewis han señalado las dificultades que implica la programación paralela. Estos autores afirman lo siguiente: "La programación paralela involucra todos los desafíos de la programación en serie agregando retos adicionales como son la partición de datos, la partición de tareas, la programación de tareas simultáneas, la depuración en paralelo y la sincronización. A diferencia de los sistemas de procesadores individuales, el ancho de banda de interconexión y la latencia de los mensajes juegan un papel dominante en el desempeño de estos sistemas distribuidos y paralelos. No hay una manera clara de predecir el rendimiento de estos nuevos sistemas, por lo tanto es difícil anticipar sus posibles beneficios previamente a cualquier inversión de tiempo y esfuerzo." [El-Rewini & Lewis, 1998]. Otras declaraciones refuerzan este punto de vista. Por ejemplo, Hennessy et al. han señalado que "los programas de ordenador en paralelo son más difíciles de escribir que los secuenciales porque la concurrencia introduce nuevas clases de posibles errores de software. La comunicación y sincronización entre las diferentes subtareas suelen ser algunos de los mayores obstáculos para conseguir un buen rendimiento de los programas paralelos." [Hennessy *et al.*, 1999] Esto deja a las claras que los pasos de paralelización que puedan proponerse no serán adiciones fútiles. El Profesor Alba, al dar una conferencia sobre "Algoritmos evolutivos paralelos" [Alba & Brignole, 2010], mostró la siguiente lista de los grandes desafíos que se presentan para la programación en paralelo:

1. Usted debe saber sobre programación concurrente y paralela.

2. Usted debe conocer acerca de las herramientas adecuadas para este tipo de programación.

3. Usted debe conocer las prestaciones del hardware con el que cuenta para

programación en paralelo.

4. Usted debe saber sobre las redes y protocolos de comunicación.

5. La tasa de error de programación en paralelo es mayor que en secuencial.

6. El algoritmo paralelo resultante es más difícil de analizar.

7. Al final del proceso de diseño, implementación y ejecución puede no evidenciarse ninguno de los beneficios esperados.

8. El sistema paralelo es más difícil de usar por otras personas.

El citado profesor ha señalado que paralelizar está lejos de ser trivial, pero que realmente vale la pena dado que muchos problemas cuya resolución era un tema pendiente, ahora son resolubles mediante el uso de algoritmos paralelos.

5.1 Computadoras paralelas

Una *computadora paralela* podría ser un sistema único especialmente diseñado conteniendo múltiples procesadores, o computadoras diferentes interconectadas de alguna manera. Este enfoque debería proveer un incremento sustancial en la performance. La teoría dice lo que realiza una computadora en un tiempo T, n computadoras lo realizan en un tiempo $\frac{T}{n}$. Por supuesto, esto es una situación ideal, situación que pocas veces se alcanza en la práctica. Los problemas no siempre pueden dividirse en partes completamente independientes, y dichas partes deben interoperar transfiriéndose datos y sicronizándose. Sin embargo sustanciales mejoras pueden lograrse, dependiendo de las características y del grado de 'paralelismo' que posee el problema.

El advenimiento de nuevas computadoras paralelas en las últimas décadas ha presentado nuevas oportunidades (y desafíos) en la comunidad de investigadores en análisis numérico, ya que estos se han propuesto aprovechar al máximo las

nuevas arquitecturas de computadoras y el gran poder computacional disponible [de Leone *et al.*, 1998, Schnabel, 1995, Migdalas *et al.*, 2003]. Los mayores esfuerzos fueron fueron realizados en la comunidad del álgebra lineal numérica desarrollando nuevos paquetes de álgebra lineal en paralelo, y algoritmos para problemas densos y ralos. El primer impacto de la computación en paralelo para resolver problemas de optimización numérica fue una directa consecuencia de esta evolución del álgebra en paralelo, al incorporar en sus códigos software de álgebra lineal en paralelo encapsulado [D'Apuzzo *et al.*, 2000]. Esto es todavía un asunto crítico ya que paralelizar los núcleos de álgebra lineal en algoritmos de optimización no es una tarea trivial. Algunas características específicas y peculiares del álgebra lineal que incumbe al software de optimización (por ejemplo, la actualización de una matriz factorizada en un metodo quasi Newton) que están basadas en operaciones vector-vector tienen muy mala performance en máquinas paralelas [Migdalas *et al.*, 2003].

El desempeño del sistema no alcanza la eficiencia deseada con la incorporación de más procesadores debido a dos aspectos principales:

- limitaciones propias del algoritmo,
- limitaciones en la implementación

Las limitaciones del algoritmo están asociadas a su grado de paralelizabilidad. No todos los algoritmos permiten ser paralelizados pues el paralelismo implica la ejecución simultánea de tareas, cuya factibilidad depende de las depedencias existentes entre la información y los resultados intermedios. Así, un algoritmo inherentemente secuencial impone una restricción insalvable para su realización paralela. Por otra parte, cada algoritmo tiene sus propias características que hacen que su paralelización sea más o menos beneficiosa.

Existen limitaciones de implementación cuando un procesador no puede dedicar tiempo a realizar cómputo provechoso. Estas penalidades surgen cuando el procesador ejecuta una operación de comunicación o cuando queda ocioso:

- **Necesidad de comunicación:** esta es una penalidad propia al modelo paralelo si se considera que los procesadores tienen necesidad de comunicarse para resolver un problema cooperativamente. Dos medidas que revisten gran importancia en esta penalidad son el *ancho de banda* y la *latencia*. El *ancho de banda* se refiere a la cantidad de información que puede transmitirse por unidad de tiempo, mientras que la *latencia* mide la cantidad de tiempo que necesita un dato para ser transmitido desde un nodo de trabajo hacia otro. Enviar mensajes pequenños puede hacer que el tiempo de *latencia* sea el tiempo dominante en el total de tiempo insumido en todas las comunicaciones. Suele ser más eficiente empaquetar varios mensajes pequeños en uno mayor, aprovechando así el máximo de *ancho de banda* disponible.

- **Tiempo ocioso:** ocurre cuando un procesador no puede realizar cómputo productivo, ya sea porque no tiene más tareas para ejecutar, porque su tarea actual se encuentra suspendida esperando una sincronización, o porque los datos que necesita para continuar con su operación no han arribado. Cuando una tarea T se divide en m subtareas T_m que pueden realizarse en paralelo, el tiempo total de esta tarea T se mide desde que comienzan a ejecutarse simultáneamente las m subtareas hasta que finaliza la más lenta. Luego el tiempo total que insume ejecutar T es $t_{total} = max\{t_1, t_2, ..., t_m\}$ donde t_i es el tiempo de realización de cada tarea T_i. Cada procesador que concluye su tarea en un tiempo $t_i < t_{total}$ queda desocupado hasta que terminen los otros restantes, como se muestra en la figura 5.1.

  La estrategia de utilización plena de todos los procesadores, es decir, la distribución de cantidades iguales de trabajo entre tareas de manera que se reduzca al mínimo el tiempo de inactividad es lo que se denomina *balance de carga*.

Si bien estas limitaciones no pueden ser eliminadas en su totalidad, diversas estrategias tales como el correcto balanceo de carga de los procesadores o el uso de protocolos no bloqueantes para las comunicaciones, permiten mejorarlas para ofrecer un

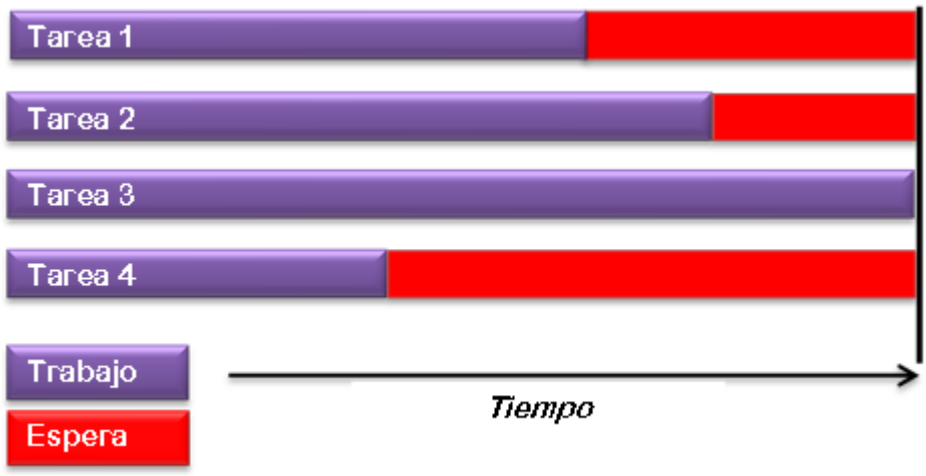


Figura 5.1: Esquema de cargas desbalanceadas

mayor desempeño del sistema paralelo [El-Rewini & Lewis, 1998, Grama *et al.*, 2003].

5.2 Arquitecturas de cómputo paralelo

Varios sistemas de clasificación para las computadoras paralelas han sido definidos a lo largo de los últimos años. La taxonomía más comúnmente utilizada es la propuesta que Flynn hizo en 1972 [Dowd & Severance, 2005], que está basada en el número de instrucciones concurrentes y en los flujos de datos disponibles en la arquitectura:

- *Una instrucción, un dato* (SISD[1]). Estas máquinas poseen un único flujo de instrucciones y un único flujo de datos. Se trata de los monoprocesadores tradicionales como las PC o los antiguos *mainframe*.

- *Una instrucción, múltiples datos* (SIMD[2]). La misma instrucción es ejecutada por varios procesadores empleando datos diferentes. Cada procesador tiene su propia memoria de datos, pero la memoria de instrucciones es única, así como el procesador de control, que recoge y organiza la ejecución de las instrucciones. Los mencionados procesadores que ejecutan las instrucciones son típicamente

[1]Single Instruction stream, Single Data stream
[2]Single Instruction stream, Multiple Data streams

específicos para llevar a cabo una cierta tarea, ya que no es necesario que sean procesadores de propósito general.

- *Múltiples instrucciones, un dato* (MISD[3]). Este tipo de máquinas contendrían varios flujos de instrucciones que acceden a un único flujo de datos y podrían ser usados en situaciones de paralelismo redundante, como por ejemplo en navegación aérea, donde se necesitan varios sistemas de respaldo en caso de que uno falle. Aún no se ha implementado ninguna máquina comercial de estas características.

- *Múltiples instrucciones, múltiples datos* (MIMD[4]). Contienen varios procesadores, cada uno de los cuales ejecuta sus propias instrucciones y opera con sus propios datos. Dichos procesadores son a menudo comerciales y de propósito general.

Aunque la taxonomía de Flynn se ha convertido en un modelo estándar, hoy en día no es lo suficientemente precisa para describir todas las arquitecturas paralelas posibles. En consecuencia, otras clasificaciones se han presentado, aunque muchas de ellas son extensiones de la clasificación de Flynn. Una de estas extensiones es la realizada por Alba y se muestra en la Figura 5.2 [Alba, 2005].

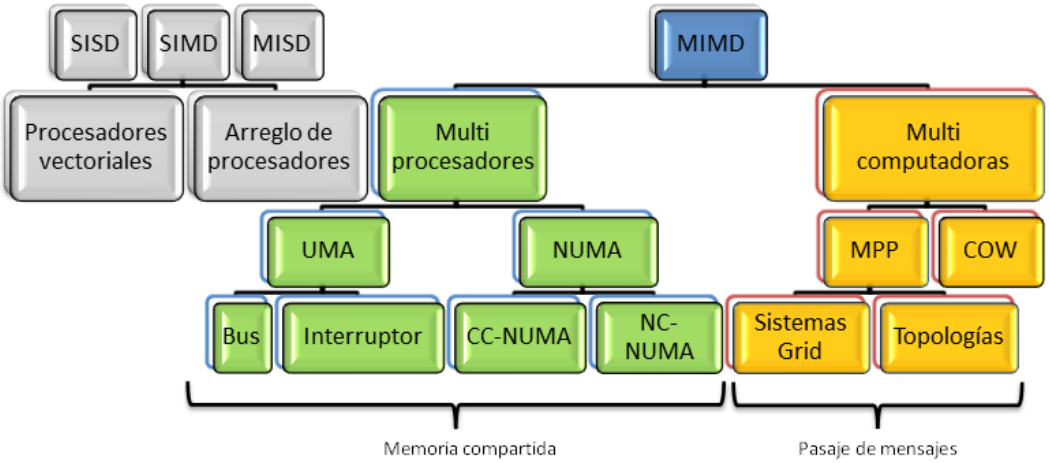


Figura 5.2: Extensión de la taxonomía de Flynn

[3]Multiple Instruction streams, Single Data stream
[4]Multiple Instruction streams, Multiple Data streams

Aquí, los sistemas MIMD se subdividen en multiprocesadores y multicomputadoras. En los sistemas multiprocesadores todos los procesadores tienen acceso directo a toda la memoria y en los sistemas multicomputadoras (también llamado sistemas distribuidos) cada procesador tiene su propia memoria local y el acceso a módulos de memoria remotos requiere el uso de un mecanismo de pasaje de mensajes.

Los multiprocesadores se clasifican en función de si el tiempo de acceso a toda dirección de memoria es constante (acceso a memoria uniforme, o UMA[5]) o no lo es (acceso a memoria no uniforme o NUMA[6]). En el primer caso, la interconexión entre los procesadores puede ser un bus o un interruptor, en el segundo caso hay una diferencia si los cachés se mantienen coherentes (coherent-cache o CC-NUMA) o no (NC-NUMA). Aunque los multiprocesadores son ampliamente utilizado, tienen el inconveniente de estar limitados en términos del máximo número de procesadores que pueden formar parte de ellos, y su precio tiende a aumentar exponencialmente con el número de procesadores.

Los sistemas distribuidos se componen de colecciones de ordenadores interconectados, cada uno con su propio procesador y memoria, y un adaptador de red. En comparación con los multiprocesadores, estos tienen un número de ventajas significativas: son más fáciles de construir y extender, tienen mejor relación precio/rendimiento, poseen mayor escalabilidad y flexibilidad, y son la única opción para ejecutar aplicaciones inherentemente distribuidas [Tanenbaum, 1995]. Un típico sistema distribuido es un conjunto de estaciones de trabajo (Cluster of Workstations, COW), integrado por PCs o estaciones de trabajo interconectadas por una red de comunicación, tales como Fast Ethernet (de propósito general, de bajo costo) o Myrinet (de alto rendimiento, de costo medio). El número de equipos en un cluster se limita a unos pocos cientos debido a las limitaciones en la tecnología de redes. Los sistemas que pertenecen al modelo MPP (Procesador Masivamente Paralelo) se componen de miles de procesadores. Si el sistema MPP está fuertemente acoplado (es decir, se trata de un equipo único) entonces tenemos los sistemas MIMD basados en topologías (hipercubo, árbol, toro).

[5]Uniform Memory Access

[6]Non-uniform Memory Access

Por otro lado, un MPP puede estar compuesta de máquinas pertenecientes a múltiples organizaciones y dominios administrativos, lo que lleva a los llamados sistemas de red o *Grid Systems*, que se construyen en torno a la infraestructura que proporciona Internet [Berman *et al.*, 2003].

5.3 Procesamiento paralelo sobre sistemas distribuidos

El uso de múltiples procesadores interconectados permiten coordinar sus esfuerzos computacionales para resolver un mismo problema. Este concepto constituye la base de los mecanismos del procesamiento paralelo. Naturalmente, para obtener beneficios concretos es necesario asimismo organizar las tareas en forma eficiente, es decir efectuar un diseño de software adecuado.

La programación en paralelo resultó prometedora en su inicio para la realización de cómputo intensivo pero su principal obstáculo era lo costoso de las máquinas con esta capacidad. Es por eso que desde los comienzos de la década del 90 comenzó a crecer la implementación de *clusters* conformados por unidades de cómputo de propósito general. Un *cluster* es un conjunto de estaciones de trabajo cualesquiera que se comunican entre sí mediante una red de datos local (Local Area Network - LAN). La programación sobre ambientes distribuidos permite aprovechar las bondades de la repartición de tareas que permite el paralelismo, pero trabajando en redes de máquinas comunes, que pueden tener distintas arquitecturas y por ende, distinta performance.

La configuración distribuida tiene éxito debido a la creciente disponibilidad tanto de estaciones de trabajo como de redes locales de comunicación de datos de alto desempeño. Los clusters resultan muy viables por su bajo costo de inversión inicial y la fácil escalabilidad de los sistemas. Otro factor importante que contribuye a facilitar el empleo de este enfoque es la estandarización de muchas de las herramientas usadas por las aplicaciones paralelas, lográndose así un entorno de programación más

práctico, independiente de la arquitectura de cada estación de trabajo o de la red de comunicación de datos. Ejemplos de estos estándares son las librerías de pasaje de mensajes PVM [Geist *et al.*, 1994], MPI [MPI Forum, 1994] y el lenguaje HPF (High Performance Fortran) [Koelbel & Zosel, 1993].

5.3.1 Ventajas y desventajas del procesamiento paralelo distribuido

Los clusters de estaciones de trabajo ofrecen las siguientes ventajas:

- Son más fáciles de integrar en redes locales existentes que las computadoras paralelas.
- Pueden usarse nodos de cómputo no dedicados.
- Las redes de estaciones de trabajo son más económicas y altamente disponibles comparadas Con las plataformas de supercomputadoras.
- Los clusters pueden crecer fácilmente; las capacidades de un nodo pueden ser incrementadas agregando nuevos periféricos o reemplazando su procesador.

Sin embargo, disponer de nodos de cómputo veloces en un cluster no asegura que la ejecución de un algoritmo paralelo sea eficiente [Vazquez, 2002]. Resolver un problema de manera cooperativa implica que esos nodos deben comunicarse, tanto para transmitir resultados intermedios como para realizar tareas de coordinación. Por lo tanto, la red de transmisión de datos también influye en la eficiencia de un cluster. En este sentido el *ancho de banda* en las comunicaciones y la *latencia* en la transmisión de la información son fundamentales para establecer la calidad del cluster.

Afortunadamente las redes de comunicación de datos han evolucionado sostenidamente en los últimos años mejorando ostensiblemente sus prestaciones (originalmente, su función era proveer a los usuarios servicios de red básicos como compartir impresoras,

archivos de datos, etc.). Esto favoreció el uso efectivo de los cluster para resolver problemas de cómputo intensivo dado que en estos sistemas paralelos, como fue mencionado en la introducción de este capítulo, la latencia en los mensajes es una penalidad de magnitud significativa pudiendo llegar a provocar que el tiempo de comunicación domine al tiempo total de ejecución.

5.3.2 Parallel Virtual Machine

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 se encuentran 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 Forum, 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]. Comparaciones más detalladas entre PVM y MPI pueden encontrarse en [Geist *et al.*, 1996, Gropp & Lusk, 2002].

El concepto de máquina virtual que implementa PVM provee una base para la heterogeneidad y portabilidad. La heterogeneidad ha adquirido mucha importancia para computación de alta performance dado que los investigadores con demandas computacionales importantes se han orientado hacia clusters de multiprocesadores más pequeños comunicados por interconexiones de alta velocidad. Muchas organizaciones ya están utilizando diferentes sistemas de computación en al forma de PCs o estaciones de trabajo (workstations) en los escritorios de sus empleados. Este tipo de máquinas de escritorio integradas pueden representar un poder computacional muy interesante. Los sistemas de software paralelo necesitan entonces adaptar su ejecución a diferentes

plataformas.

PVM es un sistema que hace posible desarrollar aplicaciones en un conjunto de computadoras heterogéneo conectados por una red, que funciona para el usuario como una única máquina paralela.

Máquina Virtual

PVM está basado en el concepto de máquina virtual, que es una colección de recursos computacionales (potencialmente heterogéneos) manejados como una única máquina paralela. El concepto de máquina virtual es fundamental en PVM, proveyendo una base para la heterogeneidad, portabilidad y encapsulamiento de las funciones que constituyen PVM. Es el concepto de máquina virtual lo que revolucionó los sistemas distribuidos heterogéneos, vinculando diferentes workstations, computadoras personales y computadoras paralelas masivas para formar una única maquinaria computacional integrada.

En términos de manejo de recursos, PVM es inherentemente dinámico. Los recursos computacionales, o huéspedes, pueden ser agregados o eliminados, desde un sistema de consola o desde la misma aplicación del usuario. Permitir a las aplicaciones interactuar y manipular su entorno provee un paradigma poderoso para realizar balance de carga, migración de tareas y tolerancia a las fallas. La máquina virtual provee una estructura para determinar qué tareas se están ejecutando y soporta un servicio de nombres de manera tal que tareas iniciadas en forma independiente unas de otras pueden localizarse y cooperar. Otro aspecto de la dinámica de la máquina virtual es la eficiencia. Las aplicaciones de los usuarios pueden exhibir variaciones en la demanda de trabajo computacional a lo largo de su ejecución. Luego una infraestructura de pasaje de mensajes debería proveer un control flexible de la cantidad de poder computacional a utilizar.

Tolerancia a las fallas

La tolerancia a las fallas es un asunto crítico en aplicaciones computacionales científicas de gran escala. Simulaciones que pueden tomar días, o hasta semanas en ejecutarse, deben tener alguna forma de tratar las fallas en el sistema o en la tarea que se está aplicando. Sin detección de fallas y recuperación es difícil que tales aplicaciones se completen. Por ejemplo, consideremos una simulación grande con docenas de workstations trabajando. Si una de estas workstations fallara, entonces ciertas tareas críticas para la aplicación desaparecerían. Adicionalmente, esta falla podría no ser inmediatamente obvia para el usuario. Muchas horas podrían desperdiciarse hasta que se descubriese que algo anduvo mal. PVM soporta un esquema básico de notificación de fallas. Bajo el control del usuario, las tareas se pueden registrar con PVM para ser notificadas cuando el estatus de la máquina virtual cambia, o cuando una tarea falla. Esta notificación se realiza mediante un mensaje especial que contiene información sobre el evento particular.

5.4 Procesamiento paralelo sobre arquitecturas de memoria compartida

Por otro lado, los avances en la tecnología han permitido el surgimiento de otras alternativas en el procesado paralelo. En sistemas operativos, un hilo de ejecución, hebra o subproceso (thread) es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente [Eggers *et al.*, 1997]. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. Un

proceso multihilo es cuando un proceso tiene múltiples hilos de ejecución los cuales realizan actividades distintas, que pueden o no ser cooperativas entre sí. Las nuevas arquitecturas de procesadores permiten la aplicación de varios hilos en paralelo en un único procesador mediante el intercalado de éstos, aprovechando los tiempos de latencia de memoria. Esto es conocido como el multi-hilado simultáneo[7]. La aparicición de los procesadores con múltiples núcleos, unido a la extensión de los lenguajes de programación, que permiten implementar paralelismo mediante la manipulación de hilos, hizo posible el paralelismo real de estos hilos simultáneos (multi-procesador a nivel del chip)[8] dándole una relevancia fundamental a este tipo de arquitecturas [Tullsen *et al.*, 1998, Marr *et al.*, 2002, INTEL, 2013].

El paradigma de memoria compartida es completamente distinto al de memoria distribuida. La idea básica se centra en ejecutar en una serie de hilos o *threads*, las tareas concurrentes que expresa el algoritmo paralelo, dentro del mismo espacio de memoria. El algoritmo paralelo debe controlar el acceso de lectura y escritura a la zona de variables compartidas para evitar los problemas que originan las condiciones de carrera y las dependencias de los datos, evitando, por tanto, incorrecciones en la ejecución. En 1995, IEEE especifica el estándar API (Application Programming Interface) [9] POSIX, también comúnmente conocido como PThreads [Lewis & Berg, 1997]. POSIX ha emergido como la API estándar para la programación multi-hilo. Sin embargo, la programación paralela con esta librería se hace a un muy bajo nivel, y por tanto es más compleja de llevar a cabo y de poner a punto. Es por ello que su uso se ha restringido principalmente para la programación de sistemas operativos, en lugar de aplicaciones.

5.4.1 Librería OpenMP

La librería OpenMP [Dagum & Menon, 1998, Quinn, 2003] es un API usado para multihilado directo explícito y paralelismo de memoria compartida. Con OpenMP se

[7]SMT, Simultaneous Multithreading.

[8]CMP, chip multiprocessor.

[9]API 1003.1c-1995

puede programar a un nivel superior si lo comparamos con los PThreads, pues con las directivas no debe preocuparse por la inicialización de los atributos de los objetos, por la configuración de algunos argumentos de los hilos, y por la partición del espacio de datos. Consta de tres componentes API principales: directivas del compilador, rutinas de librería de tiempo de ejecución y variables de entorno. Es una librería portable, pues está especificado para los lenguajes C/C++ y Fortran y se ha implementado en múltiples plataformas incluyendo la mayoría de Unix y Windows. Es además muy sencilla de usar, ya que con tan sólo 3 o 4 directivas se puede implementar un paralelismo significativo, proporcionando la capacidad para paralelizar de forma incremental un programa secuencial.

5.5 Esquemas paralelos híbridos

Un algoritmo paralelo concebido para una arquitectura de memoria distribuida puede, bajo ciertas condiciones, ejecutarse en un sistema que posea una arquitectura de memoria compartida, si se proveen los mecanismos necesarios para emular las comunicaciones. De manera que cada proceso se traduciría en un hilo o thread y los mecanismos de transmisión o recepción serían los provistos para tal efecto. Con la reciente aparición de las arquitecturas de cluster de multicores surge el modelo de programación híbrido, es decir, una colección de procesadores multicore interconectados mediante una red. Los clusters de multicore permiten combinar las características más distintivas de los clusters y de los multicores: la comunicación entre procesos que pertenecen al mismo procesador físico puede realizarse utilizando memoria compartida, mientras que la comunicación entre procesadores físicos puede ejecutarse utilizando pasaje de mensajes.

Actualmente, entre los lenguajes que se utilizan para programación híbrida aparecen OpenMP para memoria compartida y PVM o MPI para pasaje de mensajes

En este trabajo de tesis hemos diseñado un algoritmo paralelo para sistemas distribuidos que fue ejecutado en el cluster del Departamento de Ciencias e Ingeniería

de la Computación de la UNS. La técnica de diseño utilizada nos permitió adaptar el algoritmo para hacerlo plausible de ser ejecutado sobre una arquitectura multicore. Esta experimentación numérica fue realizada en una computadora con estas características ubicada en la Planta Piloto de Ingeniería Química y con un servidor del Departamento de Ciencias e Ingeniería de la Computación de la UNS. El estudio de los posibles beneficios de una implementación sobre plataformas híbridas corresponderá a un trabajo futuro.

5.6 Métricas de rendimiento para algoritmos paralelos

Para analizar el rendimiento de programas paralelos se tienen en cuenta ciertos índices particulares que se utlizan como referencia.

5.6.1 Tiempo de ejecución

En el caso de un programa secuencial, consiste en el tiempo transcurrido desde que se lanza su ejecución hasta que finaliza. En el caso de un programa paralelo, el tiempo de ejecución es el tiempo que transcurre desde el comienzo de la ejecución del programa en el sistema paralelo hasta que el último procesador culmina su ejecución [Grama *et al.*, 2003]. Para sistemas paralelos con memoria distribuida el tiempo paralelo con p procesadores, T_p , se puede determinar de modo aproximado mediante la fórmula

$$T_p \approx T_{Cmpt} + T_{Com}$$

donde T_{Cmpt} es el tiempo que tarda el sistema multiprocesador en hacer las operaciones aritméticas y T_{Com} es el tiempo de comunicación, o sea, el tiempo que tarda el sistema multiprocesador en ejecutar transferencias de datos.

El tiempo aritmético se expresa en cantidad de FLOPs[10], donde el FLOP es una medida que indica la velocidad que tarda el procesador en realizar una operación aritmética en punto flotante.

[10]FLOP: Floating point operation

5.6.2 Ganancia en velocidad *Speed-up*

El Speed-up para p procesadores, S_p, es la razón entre el tiempo de ejecución de un programa secuencial, T_S, y el tiempo de ejecución de una versión paralela de dicho programa en p procesadores, T_P. Este índice indica la ganancia de velocidad que se ha obtenido con la ejecución en paralelo.

$$S_p = \frac{T_S}{T_P} \tag{5.1}$$

Alba distingue entre varias definiciones de Speed-up dependiendo de cómo se interpreten los tiempos T_S y T_P [Alba, 2002]. Speed-up *fuerte* es el que se obtiene cuando se compara el tiempo de ejecución en paralelo contra la más rápida de las versiones secuenciales. Barr y Hickman lo llaman Speed-up *absoluto* [Barr & Hickman, 1993]. Esta es la definición más exacta de aumento de velocidad, pero debido a la dificultad de determinar cuál es la mejor versión secuencial es poco usada por los diseñadores de programas [Bertsekas & Tsitsiklis, 1997]. El Speed-up *débil* es el que se obtiene al comparar el tiempo del algoritmo paralelo con la versión serial del mismo algoritmo. Para esta definición Alba propone dos tipos de Speed-up *débil*: de tipo 1 es cuando se compara la versión paralela del algoritmo con la versión secuencial canónica del mismo. De tipo 2 es la razón del tiempo de ejecución en serie de un código paralelo en un procesador con respecto al tiempo de ejecución del mismo código en p procesadores. Esta razón Barr y Hickman la llaman Speed-up *relativo* [Barr & Hickman, 1993].

A partir de la definición de Speed-up se puede distinguir entre Speed-up *sublineal* ($S_p < p$), *lineal* ($S_P = p$) y *superlineal* ($S_P > p$)

5.6.3 Eficiencia

La eficiencia es el cociente entre el Speed-up y el número de procesadores, multiplicado por 100. Significa el porcentaje de aprovechamiento de los procesadores para la resolución del problema.

$$E_p = \frac{S_p}{p} * 100$$

5.6.4 Granularidad de los procesos

Para lograr una mejora en la velocidad de procesamiento mediante el paralelismo, es necesario dividir el cómputo en subtareas que puedan ser ejecutadas en forma simultánea, no importa que clase de sistema de cómputo paralelo se disponga. El tamaño de estas tareas puede ser descripto mediante su granularidad. Con granularidad gruesa, cada tarea contiene gran cantidad de código que se ejecuta secuencialmente y consume un tiempo de cómputo sustancial. De la misma manera, granularidad fina implica que el código consiste de pocas instrucciones o, eventualmente, una sola. En general se busca incrementar la granularidad de las tareas en la implementación paralela para compensar el costo de creación de los procesos asociados y el de comunicación de los datos.

La siguiente fórmula:

$$\frac{\text{Tiempo total de cómputo}}{\text{Tiempo total de comunicación}} = \frac{T_{Cmpt}}{T_{Com}} \tag{5.2}$$

puede ser usada como una métrica para la granularidad. Se observa claramente que es importante maximizar la relación cómputo/comunicación mientras se mantiene el paralelismo. Además, la granularidad óptima en algunas ocasiones está relacionada con la cantidad de procesadores disponibles en el sistema. Por ejemplo, cuando la metodología de resolución de un problema implica una descomposición de su dominio, a mayor cantidad de nodos de cómputo menor sería el tamaño de las tareas y mayor la cantidad de comunicaciones (debido a la necesidad de sincronización y/o reportes de resultados parciales). Así, el valor de la relación 5.2 baja al aumentar el número de procesadores. Esto muestra que en los sistemas paralelos existe un punto óptimo de operación, y cuando se lo alcanza, la adición de nuevos nodos de cómputo disminuye la eficiencia.

La relación entre la entre la granularidad y la eficiencia es de proporcionalidad directa. En efecto

$$\begin{aligned} E = \frac{T_s}{pT_p} &= \frac{T_{Cmpt}}{p(\frac{T_{Cmpt}}{p} + T_{Com})} \\ &= \frac{T_{Cmpt}}{T_{Cmpt} + pT_{Com}} \end{aligned}$$

$$= \frac{1}{1 + p\frac{T_{Com}}{T_{Cmpt}}} \quad \text{(Dividiendo ambos miembros por } T_{Cmpt}) \qquad (5.3)$$

El segundo término del denominador en 5.3 es el inverso de la medida de granularidad definida en 5.2. Luego, en la medida que el cociente 5.2 es grande, también lo es la eficiencia del sistema paralelo.

Estos problemas de granularidad pueden surgir de los pequeños subproblemas que a menudo deben ser resueltos en los problemas de optimización de gran escala. Luego lograr una implementación paralela de alta performance y de propósito general para resolver problema de optimización se ve obstaculizado por el carácter intrínsecamente secuencial de ciertas operaciones de álgebra lineal.

5.7 Paralelismo en optimización no lineal

El área de la optimización global es aquel en el cual el impacto de las nuevas tecnologías computacionales ha sido mayor. Dado el gran tamaño y la naturaleza combinatoria de algunos de estos problemas, solo pueden ser abordados por las mas poderosas máquinas. De hecho muchos de los nuevos algoritmos han sido inspirados en la disponibilidad de las nuevas máquinas paralelas, y la simbiosis entre la optimización numerica y las tecnologías computacionales es hoy mas grande que nunca.

Siguiendo a Schnabel, la introducción de la computación paralela en optimización no lineal puede ser realizada en tres niveles:

1) paralelización de la evaluación de función y/o las derivadas,

2) paralelización del los núcleos de álgebra lineal,

3) modificación de los algoritmos básicos incrementando el grado de paralelismo intrínseco.

Como se dijo anteriormente, los niveles 1 y 2 de paralelización han demandado una gran cantidad de trabajo, buscando principalmente generar software paralelo de propósito general, con un alto nivel de robustez, portabilidad y confiabilidad. También

se busca que el grado de escalabilidad sea lo más alto posible (al menos en máquinas paralelas MIMD). En consecuencia, un gran número de algoritmos altamente sofisticados han sido desarrollados en las últimas dos décadas. El tercer nivel en la clasificación de Schnabel conduce al diseño de nuevos algoritmos y ha sido una linea de investigación muy importante en los últimos años [Schnabel, 1995].

Como en muchas areas del análisis numérico, diversos algoritmos de optimización numérica han sido analizados y reformulados en base a consideraciones de paralelismo. Como un ejemplo, el algoritmo BFGS inverso se ha encontrado, desde el punto de vista del paralelismo, mucho más atractivo que el directo [Byrd *et al.*, 1988]. Similarmente, las estrategias de punto interior han ganado en preferencia a los algoritmos basados en conjuntos activos por su posibilidad directa de implementación en paralelo. Esto ha estimulado una grand cantidad de investigaciones para entender y solucionar las bondades y debilidades de estas estrategias en términos de velocidad de convergencia y robustez [D'Apuzzo & Marino, 2003, Gupta *et al.*, 1997]. En el trabajo de Ghatas y Orozco se implementó un método de SQP (programación cuadrática secuencial) del Hessiano reducido en paralelo para optimización de problemas sin restricciones. En particular, el paralelismo se utilizó para obtener la solución de dos sistemas de ecuaciones lineales. La construcción y solución de cada uno de estos dos sistemas se realizó en paralelo, como también cálculos de sensibilidad asociadas con variables de estado. Se utilizó un método para resolver PDE (ecuaciones diferenciales parciales) en paralelo, el cual fue extendido con cálculos de gradiente y los multiplicadores de Lagrange [Ghattas & Orozco, 1997]. Durazzi y Ruggiero propusieron utilizar resolvedores iterativos de métodos de punto interior paralelos para problemas de programación lineal y cuadrática [Durazzi & V.Ruggiero, 2003]. Ferris y Mangasarian mostraron una solución para resolver problemas de optimización en el cual las variables están distribuidas entre varios procesadores. Cada procesador actualiza su propio bloque de variables en paralelo, mientras permite que las variables restantes cambien en una manera restringida. Luego de la paralelización le sigue una sincronización entre los procesadores, que optimiza sobre los puntos obtenidos

por éstos [Ferris & Mangasarian, 1994]. Luego Mangasarian propondría una versión paralela para un teorema fundamental de optimización sin restricciones secuencial. En dicha versión cada uno de los procesadores utiliza simultáneamente un algoritmo diferente, y puede realizar uno o varios pasos de un algoritmo secuencial sobre una porción del gradiente de la función objetivo, en forma totalmente independiente [Mangasarian, 1995]. Mittelmann presentó un método en el cual se reemplaza un problema no lineal de gran escala por un conjunto de subproblemas menores, cada uno de los cuales puede ser resuelto local e independientemente en paralelo [Mittelmann, 1996]. Nagaiah y Kunisch desarrollaron e implementaron una técnica numérica eficiente para resolver un problema de control óptimo originado en un sistema de reacción-difusión que surge en la electrofisiología cardiaca [Nagaiah & Kunisch, 2011].

5.8 Paralelismo en problemas de ingeniería de procesos

Mejorar el tiempo de cómputo que insume la convergencia a la solución de los algoritmos para problemas de optimización de sistemas de procesos es de gran importancia en la actualidad. En comparación con las formas tradicionales, algunos autores han propuesto alternativas algorítmicas eficientes con el fin de reducir el costo computacional que presentan algunos problemas muy exigentes. Keawhom reportó un sistema de manejo de las restricciones que exhibió un costo computacional considerablemente más bajo que el costo requerido por la función de penalización tradicional [Kheawhom, 2010]. A su vez, respecto a la solución de un problema MINLP, Kraemer et al. proponen una reformulación que requiere significativamente menos tiempo de cálculo con el fin de identificar óptimos locales de mejor calidad [Kraemer *et al.*, 2009]. En el diseño de procesos Wang et al. mostraron resultados numéricos ventajosas mediante un control profundo de los criterios de convergencia aplicado a un método de programación cuadrática sucesiva con Hessiano reducido [Wang *et al.*, 2007]. En control de procesos,

Jabbar et al. diseñaron un algoritmo paralelo que garantiza la estabilidad y el rendimiento óptimo del observador de variables de estado [Abdel-Jabbar *et al.*, 1998]. Más tarde, para el problema de optimización de diseño y control integrados en plantas de proceso, Egea et al. propusieron un método basados en sustitutos que compite con las estrategias de control convencionales [Egea *et al.*, 2007].

Claramente una de las alternativas en la búsqueda de reducción de este costo computacional es el uso de múltiples procesadores interconectados que permitan coordinar sus esfuerzos para resolver un mismo problema, concepto que es base de los mecanismos del procesamiento paralelo. Naturalmente, para obtener beneficios concretos es necesario asimismo organizar las tareas en forma eficiente, es decir efectuar un diseño de software adecuado.

Como fue mencionado anteriormente los métodos SQP son los más populares resolvedores de NLPs en ingeniería de sistemas de procesos, luego se han realizado desarrollos sobre estos métodos para aprovechar sus puntos paralelizables. Existen versiones de SQP para ser ejecutadas en computadoras paralelas [High & Laroche, 1995] y sobre clusters [Vazquez & Brignole, 1999]. También se han desarrollado versiones paralelas de GRG [Vazquez *et al.*, 2000]. Diversos ingenieros químicos han intentado desarrollar técnicas para dividir la evaluación de la función en pequeñas partes independientes unas de las otras y asi ser evaluadas en paralelo [Chen *et al.*, 2011].

La dificultad encontrada es que estas técnicas son altamente dependientes del problema específico a resolver, luego no pueden ser incluidas como técnicas generales para optimización en paralelo. Las investigaciones deberían entonces apuntar al desarrollo de dichas técnicas generales que permitan abordar cualquier problema NLP del área de Ingeniería de Procesos y resolverlo satisfactoriamente en tiempos razonables.

El uso de clusters es un recurso con gran reconocimiento en la actualidad para la computación de alto rendimiento [Chai *et al.*, 2007, Abbasi & Younis, 2007, Isard & Yu, 2009, Zaharia *et al.*, 2010]. Más aún, en los últimos tiempos estos han servido como herramienta para resolver problemas de matemática aplicada de manera eficiente [Franco & Gómez, 2011][Liu *et al.*, 2006][Shang & He, 2010]. No obstante,

en el campo de la ingeniería química hay pocas contribuciones acerca del uso de este recurso de programación [Cheimarios *et al.*, 2013, Kumar & Mazumder, 2010, Mahmoudi & Mazaheri, 2011]. Luego, el uso de clusters aparece como un recurso muy prometedor que debe ser explotado para aplicaciones de ingeniería de procesos.

Para implementar la próxima generación de software de optimización es necesario especializar los algoritmo para explotar las características particulares de cada clase de problemas. Pero también diseñarlos para aprovechar los avances logrados, no solo en computadoras multi procesadores, sino también en el desarrollo alcanzado en materia de sistemas distribuidos, donde es posible simular una máquina multiprocesador en un cluster heterogéneo, logrando así una mayor capacidad de cálculo. La adaptación del software a estas nuevas posibilidades debería incluir estructuras de datos específicas, resolvedores particulares y principalmente paralelismo. Esto puede requerir hasta inclusive modificar la lógica de los algoritmos generando tal vez, nuevas familias de algoritmos de optimización.

Al mismo tiempo, si a un usuario le es permitido especializar componentes en un algoritmo, el software de optimización debería incluir run time test optativos, pero eficientes, para validar los distintos cálculos realizados. Esto incluye validar las evaluaciones de función y gradientes, resolvedores intermedios (si los incluye), y cualquier cálculo importante que determine claramente condiciones a las tareas a realizar posteriormente por el algoritmo [Biegler & Grossmann, 2004].

Una de las posibilidades de paralelismo en optimización no lineal que pueden ser aprovechadas para el estudio de diseño de procesos químicos es el método SPG. Su desempeño en problemas específicos de diseño de plantas industriales, analizado en el capítulo 4, y sus grandes posibilidades de paralelismo[11] lo ubican como una herramienta potencialmente muy importante para resolver eficientemente problemas en este área.

[11]Este item se desarrolla más ampliamente en el capítulo 6.

Capítulo 6

La versión paralela del gradiente espectral proyectado

"En un minuto hay muchos días".

W. Shakespeare

La descripción del algoritmo SPG (algoritmo 2) muestra el papel central que tienen la aproximación del gradiente y la búsqueda lineal no monótona. Obtener una buena aproximación del gradiente representa una inversión extra de tiempo de cómputo. Y 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.

La forma de gestionar el trabajo en paralelo es mediante el modelo *maestro-trabajador*. En este modelo un procesador ocupa el lugar de *maestro* llevando el hilo principal del algoritmo, generando y asignando las tareas a los restantes procesadores (*trabajadores*).

6.1 Evaluación del gradiente en paralelo

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 sincrónica de las derivadas parciales para agilizar la obtención de la aproximación del gradiente. Se utiliza un modelo de bolsa de trabajo donde a cada nodo *trabajador* el nodo que hace de *Maestro* le asigna una componente para que calcule la derivada parcial. Cuando el *Trabajador* termina, envía el valor al *Maestro* 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.

Algoritmo 4 *Trabajador* para evaluar el gradiente en paralelo

Entrada: Recibir del *Maestro* el vector x_k y el índice i

Salida: Componente i-ésima del vector gradiente

1: Calcular $h_i = max\{\sqrt{\epsilon_M}, \sqrt{\epsilon_M}|x_{k_i}|\}$

2: Calcular $g(x_k)_i = \dfrac{f(x_k + h_i e_i) - f(x_k - h_i e_i)}{2h_i}$

3: Enviar al *Maestro* el valor $g(x_k)_i$.

4: Esperar por nuevas instrucciones

La cantidad de veces que el bloque de p rutinas en paralelo se ejecutan está vinculado a la relación de divisibilidad de n con p.

Lema 6.1 *Dada la rutina 4 para evaluar el gradiente.*

El bloque de p rutinas 4 en paralelo se ejecuta

$$\begin{cases} \frac{n}{p} \text{ veces} & \text{si } n \text{ modulo } p = 0 \\ \frac{n-r}{p} + 1 \text{ veces} & \text{si } n \text{ modulo } p \neq 0 \end{cases}$$

entonces el mejor caso se da cuando n modulo p = 0.

Demostración: Si tomamos el algoritmo de la división entera, al dividir n con p se obtiene $n = pq + r$ con $0 \leq r \leq p - 1$. Si $n \ mod \ p \neq 0$ entonces es necesaria una ejecución adicional de al rutina.

Por otro lado, al ser $r \leq p-1$ entonces

$$\frac{n-r}{p} \geq \frac{n+1-p}{p}$$
$$\frac{n-r}{p} + 1 \geq \frac{n}{p} + \frac{1}{p} > \frac{n}{p}$$

por ser $p \geq 1$. ■

Si bien $n \; mod \; p \neq 0$ nos ubica fuera del mejor caso, el peor caso se puede considerar cuando $n \; mod \; p = 1$ dado que es el que deja más procesadores inactivos $(p-1)$.

Para analizar la eficiencia del procedimiento es necesario analizar el tiempo insumido en comunicaciones. Retomando la notación de la sección 5.6, T_{Cmpt} es el tiempo insumido en cómputo y T_{Com} el tiempo insumido en las comunicaciones.

Entonces el tiempo de cálculo de la rutina en paralelo viene dado por:

$$T_p = T_{Cmpt} + T_{Com}$$

Dado que el paso 1 de la rutina es de orden constante, el tiempo de cómputo es el tiempo necesario para las dos evaluaciones de función. Si notamos T_f el tiempo de cómputo de la función entonces

$$T_{Cmpt} = \begin{cases} 2.T_f \frac{n}{p} & \text{si } n \; modulo \; p = 0 \\ 2.T_f.(\frac{n-r}{p} + 1) & \text{si } n \; modulo \; p \neq 0 \end{cases}$$

El tiempo de comunicaciones se puede establecer como:

$$T_{Com} = \alpha + \beta l$$

donde l es la longitud del mensaje, α es la latencia (el tiempo necesario para iniciar el mensaje) y $\beta = \frac{1}{\theta}$ es el tiempo de una comunicación (θ es el ancho de banda).

Sin perder generalidad, un vector de n componentes tiene un tiempo de comunicación $\alpha + \beta n$.

Para el análisis de los problemas de comunicación, es útil para ver el sistema de computación distribuida como una red de procesadores conectados por enlaces de comunicación en una determinada topología.

Definición 6.1 *Se define como topología de la red de comunicación al número, naturaleza y ubicación de los enlaces de comunicación.*

La topología de una red se puede representar con un grafo $G = (N, A)$ donde cada nodo $N_i \in N$ representa un procesador y la presencia de una arista $A_{i,j} \in A$ indica que hay una enlace de comunicación directa entre el procesador i y el procesador j.

Definición 6.2 *Una topología de bus es una topología de red en la que un conjunto de clientes se conectan a través de una línea de comunicaciones común llamado bus.*

Existen otras topologías físicas como muestra la figura 6.1. En este trabajo nos concentraremos en la topología de bus ya que es la topología de red con la que realizamos nuestros cálculos[1].

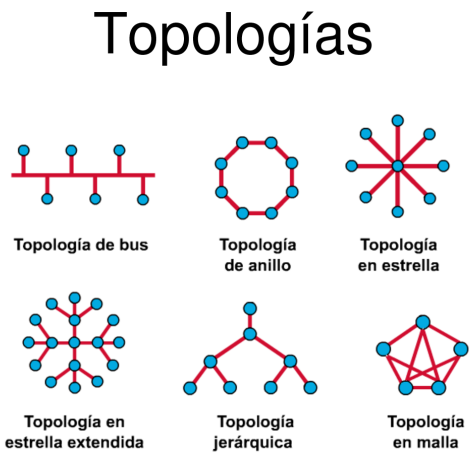


Figura 6.1: Tipos de topologías físicas

En una red basada en bus la distancia entre dos nodos en la red es constante ($\mathcal{O}(1)$). Dado que el medio de transmisión es compartido, hay poca sobrecarga asociada con la transmisión en comparación con la transferencia de mensajes de punto a punto.

[1]Las características de las plataformas utilizadas se describen en el capítulo 7.

Sin embargo, el ancho de banda limitado del bus puede representar limitaciones al rendimiento global de la red cuando el número de nodos aumenta [Grama *et al.*, 2003].

Con esta topología, la comunicación entre el *maestro* y los p *trabajadores* requiere

$$T_{Com} = \underbrace{(\alpha + \beta n)p + (n-p)(\alpha+\beta)}_{Maestro \rightarrow Trabajadores} + \underbrace{(\alpha+\beta)(n+1)n}_{Trabajadores \rightarrow Maestro} \tag{6.1}$$

Entonces el *speed-up* esperado es

$$S_p = \begin{cases} \dfrac{2nT_f}{2.T_f.(\frac{n-r}{p}+1) + (\alpha+\beta n)p + (n-p)(\alpha+\beta) + (\alpha+\beta)(n+1)n} & \text{en el peor caso} \\ \dfrac{2nT_f}{2.T_f\frac{n}{p} + (\alpha+\beta n)p + (n-p)(\alpha+\beta) + (\alpha+\beta)(n+1)n} & \text{en el mejor caso} \end{cases}$$

El límite $\lim_{n\to\infty} Sp$ se puede escribir

$$\lim_{n\to\infty} \frac{2nT_f}{n[\frac{2T_f}{p} - \frac{r}{np} + \frac{1}{n} + (\frac{\alpha}{n}+\beta)p + (1-\frac{p}{n})(\alpha+\beta) + (\alpha+\beta)(n+1)]} \tag{6.2}$$

Si $T_f = \Omega(n^2)$, dividiendo ambos miembros por nT_f se tiene $\lim_{n\to\infty} S_p = p$. Esto representa una eficiencia que tiende al 100% cuando n es grande.

6.2 Búsqueda lineal en paralelo

Un ejemplo de algoritmos de optimización inspirados en los nuevos ambientes computacionales está dado por las estrategias de búsqueda lineal multidireccionales en los cuales un alto nivel de paralelismo intrínseco es alcanzado. Este enfoque, a pesar de ser cuestionable desde el punto de vista teórico, exhibe poca interacción entre procesadores, es altamente escalable y se muestra como un muy buen ejemplo del uso de los nuevos paradigmas en el desarrollo de nuevas estrategias computacionales. La búsqueda lineal no monótona implementada en paralelo en esta sección es implementada a través del cálculo especulativo de la función en distitntos puntos.

La cantidad de búsquedas lineales necesarias en la iteración k del algoritmo 2 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. Esto ocurre para pasos suficientemente chicos, pero para alcanzar dicho paso pueden llegar a ser necesarias muchas iteraciones de búsqueda [Raydan, 1997]. Por otro lado, pasos muy chicos afectan la velocidad de convergencia del método. En función de esta dificultad se implementa una variante.

Supongamos que tenemos p procesadores disponibles $\{P_1, P_2, \cdots P_p\}$. Siguiendo la idea básica establecida por Lewis et al. [Lewis & Torczon, 2000], en el iterado k se subdivide el intervalo $[x_k, x_k + \lambda_k d_k]$ en p subintervalos de igual longitud $[x_{i-1}, x_i]$, donde $x_0 = x_k$, y $x_p = x_k + \lambda_k d_k$, $i = 1, \cdots, p$. El procesador i-ésimo evalúa $f(x_{+_i})$ donde $x_{+_i} = x_k + \frac{i}{p+1}\lambda_k d_k$ y chequea si se cumple la condición de búsqueda lineal

$$\beta_k^{(i)} \leq 0 \qquad i = 1 \ldots p \tag{6.3}$$

$$\text{siendo } \beta_k^{(i)} = f(x_{+_i}) - F_{max} - \gamma \frac{i}{p+1} d_k^T g(x_k) \tag{6.4}$$

esto es, p puntos igualmente espaciados son tomados del intervalo a explorar y la función objetivo es evaluada simultáneamente en cada uno de dichos puntos. Si ninguno de los puntos satisface la condición 6.4, es el intervalo $[x_k, x_k + \dfrac{\lambda_k}{p+1} d_k]$ el que es dividido en p puntos y el procedimiento es aplicado nuevamente. Si el criterio de búsqueda lineal aún no es satisfecho, entonces el procedimiento se aplica una vez más en el intervalo $[x_k, x_k + \dfrac{\lambda_k}{(p+1)^2} d_k]$ y así sucesivamente hasta obtener un punto exitoso. En la implementación se introduce un parámetro *Nblp* (Número de búsquedas lineales en paralelo) para limitar el número de repeticiones del procedimiento. Si más de un número fijo de búsquedas se realizan sin alcanzar un punto exitoso, el algoritmo se declara atascado, luego se interrumpe.

Continuando con la implementación del esquema *Maestro- Trabajador*, la evaluación es llevada a cabo por los *Trabajadores*, y estos informan al *Maestro* si sus puntos satisfacen o no la condición de búsqueda.

El Algoritmo 5 describe el **Algoritmo Trabajador** para la búsqueda lineal no monótona.

Algoritmo 5 *Trabajador* para búsqueda lineal no monótona

Entrada: del *Maestro* x_k, índice i, longitud del paso θ_i, f_{max}, g_k y dirección d_k

Salida: **Verdadero** si encontró un punto y el índice de dicho punto, y **Falso** en caso contrario.

1: Recibir del *Maestro* x_k, índice i, longitud del paso θ_i, f_{max}, g_k y dirección d_k.
2: Setear $x_+ = x_k + \theta_i d_k$
3: Calcular $f(x_+)$
4: Calcular $\beta_k = f(x_+) - F_{max} - \gamma\theta_i g_k^t d_k$
5: **Si** No hay mensaje de INTERRUMPIR (recibido del *Maestro*) **entonces**
6: **Si** $\beta_k \leq 0$ **entonces**
7: Enviar al *Maestro* el índice i, $f(x_+)$ y un mensaje de éxito.
8: **Si no**
9: Enviar al *Maestro* un mensaje de fracaso
10: **Fin** {Si}
11: **Fin** {Si}
12: Esperar por nuevas instrucciones

La figura 6.2 muestra el grafo acíclico dirigido (DAG[2]) de la búsqueda lineal no monótona secuencial y el de la búsqueda en paralelo.

Es de esperar que más de un trabajador encuentre un punto exitoso, entonces es necesario establecer un criterio para determinar en cál de los puntos será el nuevo iterado.

Un posible criterio podriía ser 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 desventaja que este criterio presentaría 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 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 6.4.

Esperar todas las respuestas, por más que ya se cuente con algunas exitosas, representa un segundo criterio de continuación. Este criterio permite asegurarse la

[2]Direct Acyclic Graph: Grafo cuyas aristas indican un sentido y no tiene ciclos, es decir, para cada vértice v, no hay un camino directo que empiece y termine en v.

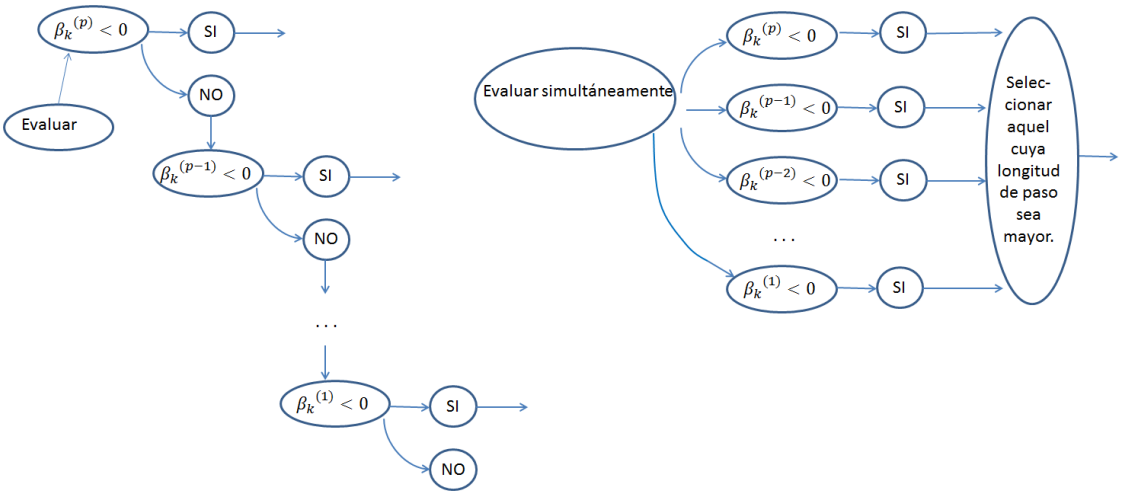


Figura 6.2: DAG para los procedimientos de búsqueda lineal. A la izquierda: procedimiento secuencial. A la derecha: procedimiento paralelo.

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. El *Maestro* contiene una lista con dos índices: T (apunta a lo que tengo) y Q (apunta a lo que quiero). Inicialmente $T = p + 1$ y $Q = 1$.

Este mecanismo se ilustra en el ejemplo siguiente:

Ejemplo 6.1 *Se inicia una búsqueda con 6 procesadores. La figura 6.3 muestra en el paso 1 como se ubican los índices T y Q. En el paso 2 se recibe un fracaso en el procesador nº 5, los índices no modifican su ubicación. En el paso 3 se recibe un fracaso en el procesador nº 2, los índices no se modifican pero el enlace del procesador 1 ahora apunta al 3. En el paso 4 se recibe un éxito del procesador nº 4, el índice T ahora apunta a 4 y este éxito* ya garantiza *el éxito de la búsqueda lineal. En el paso 5 se recibe un*

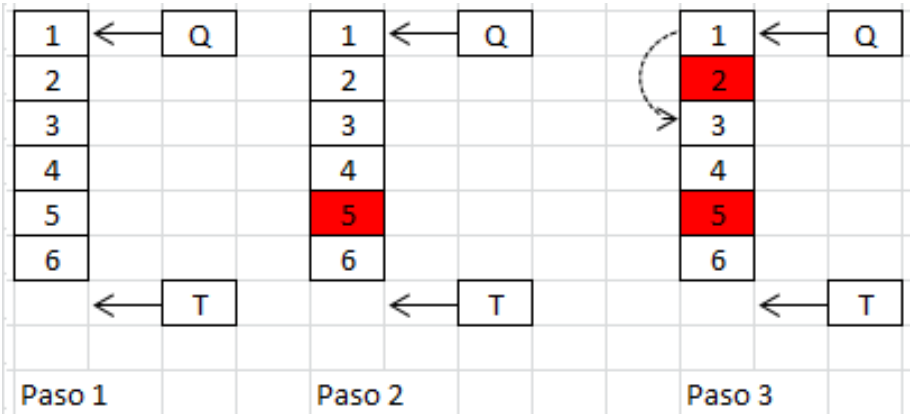


Figura 6.3: Pasos 1, 2 y 3 del ejemplo 6.1.

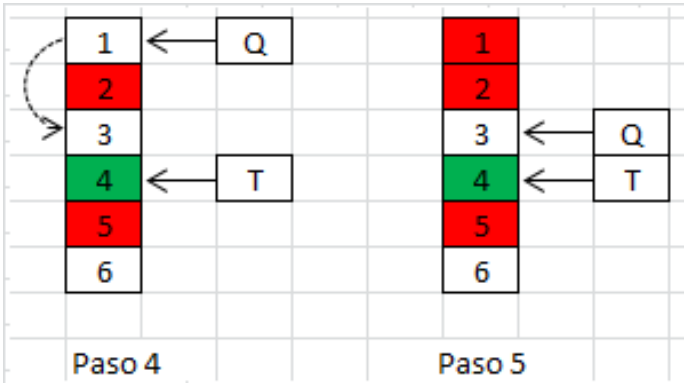


Figura 6.4: Pasos 4 y 5 del ejemplo 6.1.

fracaso en el procesador nº 1, luego Q apunta al nº 3 y el resultado final dependerá del éxito o fracaso en el procesador nº 3. Si hay éxito en el procesador nº 3, ahi coincidirán los índices Q y T, sino coincidirán en el resultado del procesador nº 4.

El esquema del procedimiento *Maestro* para la búsqueda lineal viene dado por el algoritmo 6.

La eficiencia de la búsqueda lineal el paralelo viene dada por los siguientes resultados:

Teorema 6.1 *Dado un punto $x_k \in \mathbb{R}^n$ y $d_k = -\sigma g_k$, existe al menos un valor $\bar{t} \in (0,1)$ para el cual $x_k + \bar{t}s_k$ satisface la condición de búsqueda lineal.*

Demostración: Dado que d_k es una dirección de descenso, es $g_k^T d_k < 0$. Al mismo

Algoritmo 6 *Maestro* para búsqueda lineal no monótona

Entrada: x_k, longitud del paso θ_i, F_{max}, g_k y la dirección d_k.

Salida: x_{k_+} que satisface la condición 6.4.

```
1: para i:1 hasta N_t        %N_t: Número de trabajadores. hacer
2:    Enviar al trabajador(i) x_k, índice i, longitud del paso θ_i, f_max, g_k y la dirección d_k.
3:    Lista[i]=1             % 0- Fracaso, 1- Vacío
4: Fin para
5: Terminar=FALSO
6: Repetir
7:    Recibir un mensaje de un trabajador
8:    contador = contador + 1
9:    Si índice >= T entonces
10:      Si Mensaje == Exito entonces
11:         Si índice == Q entonces
12:            Terminar=VERDADERO
13:            Enviar mensaje a todos los trabajadores de INTERRUMPIR
14:         Si no
15:            T=índice
16:         Si no, Si índice = Q  entonces
17:            Mover Q a la próxima celda > 0
18:            Si Q=T entonces
19:               Terminar= VERDADERO
20:               Enviar mensaje a todos los trabajadores de INTERRUMPIR
21:            Si no
22:               Lista[índice]=0
23:            Fin {Si}
24:         Fin {Si}
25:      Fin {Si}
26:   Fin {Si}
27: hasta que contador = p O Terminar
```

tiempo, si $0 < \gamma < 1$ entonces

$$d_k^T g_k < \gamma d_k^T g_k \tag{6.5}$$

Supongamos por el absurdo que no existe $t \in (0,1)$ para el que $x_k + td_k$ satisface la búsqueda lineal. Luego existe una sucesión $\{t_j\}$ que verifica $\lim_{j\to\infty} t_j = 0$ y tal que

$$f(x_k + t_j d_k) > F_{max} + \gamma t_j d_k^T g_k \ , \ \ \forall t_j \tag{6.6}$$

$$f(x_k + t_j d_k) - f(x_k) > F_{max} - f(x_k) + \gamma t_j d_k^T g_k \tag{6.7}$$

Al ser F_{max} el máximo valor de función de los últimos k iterados, entonces es $F_{max} - f(x_k) \geq 0$. Pero para j suficientemente grande, como $t_j \to 0$ entonces, siguiendo la ecuación 6.7, es $0 \geq F_{max} - f(x_k)$, lo cual es una contradicción salvo que $F_{max} - f(x_k) = 0$ En ese caso, dividiendo ambos miembros por t_j

$$\frac{f(x_k + t_j d_k) - f(x_k)}{t_j} > \gamma d_k^T g_k \tag{6.8}$$

y pasando al límite

$$\lim_{j\to\infty} \frac{f(x_k + t_j d_k) - f(x_k)}{t_j} > \lim_{j\to\infty} \gamma d_k^T g_k \tag{6.9}$$

$$g_k^T d_k \geq \gamma d_k^T g_k \tag{6.10}$$

lo cual contradice 6.5. Luego debe existir al menos un valor $\bar{t} \in (0,1)$ para el cual $x_k + \bar{t} s_k$ satisface la condición de búsqueda lineal. ∎

El siguiente teorema determina la eficiencia de la búsqueda lineal en paralelo.

Teorema 6.2 *Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función continuamente diferenciable en un conjunto D abierto y convexo. Existe $\bar{t} \in (0,1)$ tal que* ***para todo*** *$t \in (0, \bar{t}]$ la condición de búsqueda lineal se cumple.*

Demostración: Dado $x_{k+1} = x_k + d_k \in D$, al ser D convexo entonces $x_k + td_k \in D \ \forall t \in (0,1)$. Recordemos la notación g_k abrevia $g(x_k) = \nabla(x_k)$.

Dado que

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

entonces $f(x_k) \leq F_{Max}$.

Por la continuidad de la derivada entonces la función en la dirección d_k se puede aproximar por una recta tangente.

Sean $\phi(t) = f(x_k + t\sigma d_k)$,

$T(t) = f(x_k) + tg_k^T d_k$ (la aproximación tangente), y

$l(t) = F_{Max} + t\gamma g_k^T d_k$

Debo probar que $\exists \bar{t} : \forall t \in (0, \bar{t}]$ es $\phi(t) \leq l(t)$.

Dado que $\gamma < 1$ y que d_k es una dirección de descenso (i.e. $d_k^T g_k < 0$), entonces $\gamma g_k^T d_k > g_k^T d_k$, lo que es equivalente a que $\exists \epsilon > 0$ tal que

$$\gamma g_k^T d_k = g_k^T d_k + \epsilon \tag{6.11}$$

Entonces para el valor ϵ de 6.11 existe $0 < \bar{t} \leq 1$: $\forall t \in (0, \bar{t}]$ es

$$|\phi(t) - T(t)| < \epsilon$$

Este valor de $\bar{t}$ es el que demuestra el teorema. En efecto, si $t \in (0, \bar{t}]$ entonces $t < 1$, luego $\epsilon t < \epsilon$ y

$$\begin{aligned}
\phi(t) &< T(t) + \epsilon \\
&\leq T(t) + \epsilon t \\
&= f(x_k) + tg_k^T d_k + \epsilon t \\
&\leq F_{max} + t(g_k^T d_k + \epsilon) \\
&= F_{max} + t\gamma g_k^T d_k \\
&= l(t)
\end{aligned}$$

■

El corolario de este teorema es el que da la potencial eficiencia de la búsqueda lineal en paralelo.

Corolario 6.1 *En las condiciones del teorema 6.2, si $\bar{t} \geq \frac{1}{p+1}$ entonces el procedimiento de búsqueda lineal en paralelo encuentra un punto que satisface 6.4 en una sola llamada.*

Demostración: En efecto, el punto $x_k + \frac{1}{p+1}\sigma d_k$ es el de menor longitud de paso entre los puntos inspeccionados en la primer llamada de la búsqueda en paralelo, y al menos este satisface la condición buscada.

Lo descripto esta esquematizado en la figura 6.5. ∎

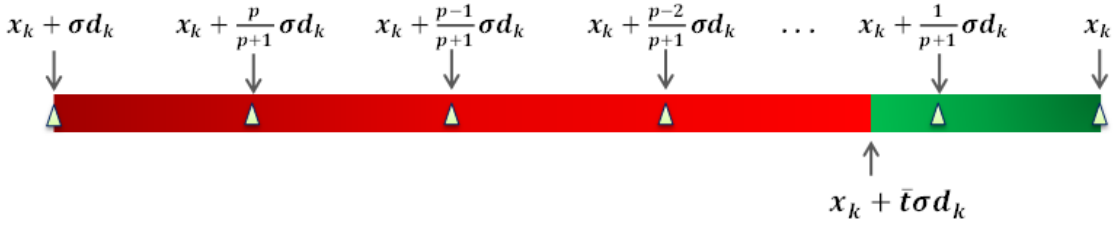


Figura 6.5: Esquema del corolario 6.1

Este enfoque tiene como objetivo llevar a cabo la búsqueda de forma diferente. El objetivo que se persigue es el de hacer menos búsquedas y no la misma cantidad pero en paralelo. Cuando esta reducción es lograda, esto tiene un impacto directo en el speed-up general del algoritmo, como se demuestra en el siguiente teorema:

Teorema 6.3 *Si el algoritmo SPG en paralelo puede reducir el número de llamadas al procedimiento de búsqueda lineal no monótona, entonces incrementa su factor de speed-up.*

Demostración: Si llamamos T_S el tiempo necesario para realizar una búsqueda y N_S el número de búsquedas realizadas, entonces el tiempo total T_{Tot} insumido en búsquedas lineales por la versión secuencial es

$$T_{Tot} = N_S * T_S \tag{6.12}$$

Por otro lado, si llamamos NC_{LS} al número de llamadas que inician el procedimiento búsqueda paralela, $Nproc$ el número de procesadores en paralelo y T_C el tiempo empleado en la comunicación entre el maestro y un trabajador, el tiempo total de búsquedas en paralelo es

$$T^p_{Tot} = NC_{LS} * T_S + NC_{LS} * Nproc * T_C \tag{6.13}$$

Luego, la ecuación 6.14 define el factor de *Speed up* S_p para el procedimiento de búsqueda lineal.

$$S_p = \frac{N_S * T_S}{NC_{LS} * (T_S + Nproc * T_C)} \tag{6.14}$$

Como $Nproc \geq 1$ y $T_C > 0$, $\frac{T_S}{T_S + Nproc * T_C} < 1$ entonces $S > 1$ cuando $\frac{N_S}{NC_{LS}} > 1$.

Luego, una reducción a las llamadas al procedimiento de búsqueda en paralelo es necesario para lograr factor de speed-up mayor a 1 ∎

Ejemplo 6.2 *Sea la función* $f(x) = (x_1 - 4)^4 + (x_2 - 3)^2 + (x_3 - 2)^4 + (x_4 - 1)^2 + 5$ *con el punto* $x_k = (5, 5, 5, 5)$. *El valor del vector gradiente es* $g(x_k) = (-4, -4, -108, -8)$. *Tomando como* $F_{max} = 107$, $\gamma = 0.1$, *longitud del paso 1 entonces la dirección de búsqueda es* $d = (4, 4, 108, 8)$.

Si $\beta = f(x_0) - F_{max} - \gamma d^T g$ *entonces la condición de búsqueda lineal no monótona se cumple cuando* $\beta < 0$. *Con el punto* x_k *dado esta condición no se cumple, luego se realiza un backtracking hasta cumplirla. Indicando con c la proporción en la que se reduce el paso, la subsucesión generada es*

x_k	$f(x_k)$	c	$g^T d$	β
$(5, 5, 5, 5)$	107	-	1.176×10^3	> 0
$(3, 3, -49, 1)$	6765207	0.5	-588	> 0
$(4, 4, -22, 3)$	331786	0.25	-294	> 0
$(4.5, 4.5, -8.5, 4)$	1.21×10^4	0.125	-147	> 0
$(4.75, 4.75, -1.75, 4.5)$	218.3828	0.0625	-73.5	> 0
$(4.875, 4.875, 1.625, 4.75)$	23.1841	0.03125	-36.75	$\mathbf{< 0}$

Para esta función y el punto dado, la condición se cumple con $c = 0.16$, *con 4 procesadores el paso más corto es* $0.2 \times d$ *en la primer búsqueda paralela y* $0.04 \times d$ *en la segunda, luego las cinco búsquedas del procedimiento secuencial se reducen a dos. Con 6 procesadores el paso más corto en la primer búsqueda es* $0.14 \times d$ *lo que resuelve el problema en esa única iteración.*

6.2.1 Análisis del Peor/Mejor caso

Con una topología definida en 6.2, la comunicación de p mensajes utilizando un algoritmo de broadcasting *one to all* para la búsqueda lineal requiere

$$t_{Com} = \underbrace{(\alpha + \beta n)p}_{Maestro \to Trabajadores} + \underbrace{2p}_{Trabajadores \to Maestro} \tag{6.15}$$

Peor caso Este se da cuando se despliegan los hilos trabajadores y el punto óptimo es el primero en la búsqueda, es decir, el punto $x_k + \frac{p}{p+1}s_k$ satisface la condición de búsqueda lineal 2.2.

Entonces el *speed-up* esperado es

$$S_p = \frac{T_f}{T_f + (\alpha + \beta n)p + 2p}$$

Si $T_f = \Omega(n^2)$ entonces $\lim_{n\to\infty} S_p = 1$. Esto representa una eficiencia de $\frac{1}{p}\%$, es decir, la eficiencia decrece con el agregado de procesadores.

Mejor caso Este caso se da cuando el primer punto que satisface la búsqueda es el último en el orden tal como muestra la figura 6.1, es decir, el punto $x_k + \frac{1}{p+1}s_k$.

Entonces el *speed-up* esperado es

$$S_p = \frac{pT_f}{T_f + (\alpha + \beta n)p + 2p}$$

Igual que en el caso anterior, si $T_f = \Omega(n^2)$ y $n > log_2 p$ entonces $\lim_{n\to\infty} S_p = p$. Esto representa una eficiencia que tiende al 100% cuando n es grande.

Tanto el mejor como el peor caso son casos extremos, el caso medio lo vamos a determinar a partir de la estadística de búsquedas en los casos de estudio.

6.3 Análisis del balance de carga

La implementación del paralelismo en el cálculo del gradiente es de distribución dinámica por demanda. Las p primeras derivadas parciales se mandan a calcular en forma

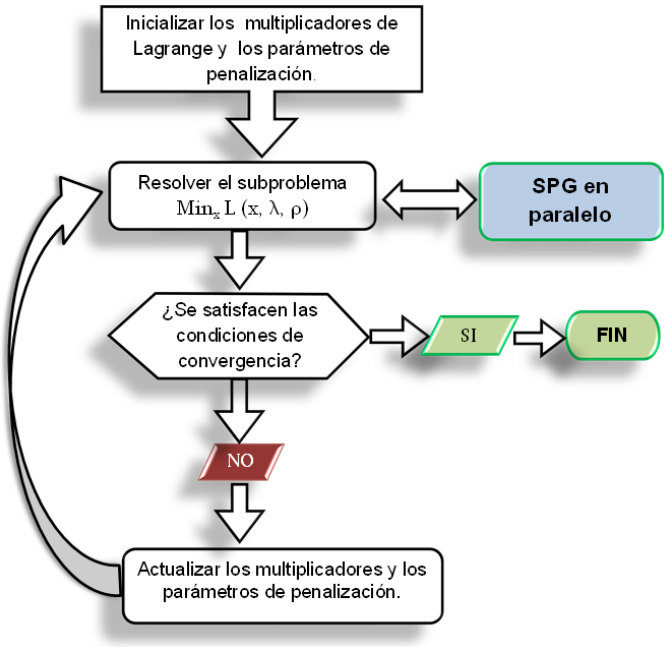


Figura 6.6: Lagrangiano Aumentado con SPG en paralelo (pALSPG).

predeterminada y las $n - p$ restantes serán calculadas por aquellos *trabajadores* que estén disponibles primero. Este enfoque es el de bolsa de trabajo [Grama *et al.*, 2003].

La búsqueda lineal no monótona tiene una distribución estática directa [Giusti *et al.*, 2005], donde las p evaluaciones de función se reparten en los p procesadores de la arquitectura disponibles en forma homogénea. El desbalance es mínimo en la medida que el tiempo de cómputo de la función en el punto determinado sea uniforme entre los p procesadores.

6.4 Convergencia del algoritmo paralelo

De lo descripto en las dos secciones anteriores, el esquema del SPG en paralelo queda conformado como se indica en el algoritmo 7. La figura 7 describe como queda el esquema general del Lagrangiano Aumentado con el método SPG en paralelo (pALSPG)

A partir de los teoremas 2.1 y 2.2 hemos podido establecer la convergencia de nuestro algoritmo paralelo. En otras palabras, bajo las condiciones del teorema 2.1, el algoritmo SPG paralelo convergerá a un punto estacionario.

Algoritmo 7 Gradiente espectral proyectado en paralelo (pSPG).

Entrada: $x_0 \in \mathbb{R}^n$ (Punto inicial), M (parámetro de no monotonicidad), σ_{min}, σ_{max} (cotas) y tol (tolerancia). Rutinas evalf, evalg y Proyección.

Salida: x_k tal que $\|g_k(x_k)\| < tol$.

1: $x_k = Proyeccion(x_0)$

% `Evaluar el gradiente` *`en paralelo`*

2: $g_k =$ `Gradiente en Paralelo`(x_k)

3: $s_k = -g_k$ % Evaluar la funcion

4: $f_k = evalf(x_k)$ % Bucle principal

5: **Mientras** $\|s_k\| > tol$ **hacer**

6: $d_k = Proyeccion(x_k - \sigma * g_k) - x_k$

7: `Búsqueda lineal en paralelo`

% Rectificación de la longitud del paso

8: $s_k = t * d_k$ (t es el valor dado por la búsqueda lineal)

9: $x_k = x_k + s_k$

10: g_{new}=`Gradiente en Paralelo`(x_k)

11: $y_k = g_{new} - g_k$

%Actualización

12: $g_k = g_{new}$

13: $skTyk = s_k^t y_k$ % Cálculo de la longitud del paso para la iteración siguiente

14: **Si** $skTyk \leq 0$ **entonces**

15: $\sigma = \sigma_{max}$

16: **Si no**

$\sigma = max(min(\frac{s_k^t s_k}{skTyk}, \sigma_{max}), \sigma_{min})$

17: **Fin** {Si}

18: **Fin** {Mientras}

Teorema 6.4 [3] *Sea* $\Omega_0 = \{x : f(x) \leq f(x_0)\}$ *un conjunto compacto. Sea* $f : \mathbb{R}^n \to \mathbb{R}$ *una función continuamente diferenciable en un entorno* N *de* Ω_0*. Sea* x_k *la sucesión generada por el algoritmo SPG en paralelo (Algoritmo 7). Entonces o* $g(x_j) = \nabla f(x_j) = 0$ *para algún índice* j*, o valen las siguientes propiedades:*

a) La sucesión $\{x_k\}$ *está contenida* Ω_0 *y todo punto límite* $\bar{x}$ *satisface* $g(\bar{x}) = 0$*,*

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

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

Demostración: La prueba consiste en demostrar que se satisfacen las hipótesis del teorema (2.1), luego vale su tésis. El algoritmo pSPG 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 (3.2) se desprende que para $c_1 > 0$ es

$$\|g_k - \tilde{g}_k\| \leq c_1 h^2 \tag{6.16}$$

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

$$\|g_k - \tilde{g}_k\| \leq \frac{1}{2}\|g_k\| \tag{6.17}$$

Considerando que

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

Siguiendo (6.17) y utilizando la desigualdad de Cauchy-Schwarz[4] se tiene

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

$$\text{de esta forma} \quad d_k^T g_k \leq \frac{1}{2}\|g_k\|^2 - \|g_k\|^2 \quad \Rightarrow \quad d_k^T g_k \leq -\frac{1}{2}\|g_k\|^2 \tag{6.18}$$

[3] Publicado en [Ardenghi *et al.*, 2007]

[4] La desigualdad de Cauchy-Schwarz establece que para todo par de vectores x e y de un espacio vectorial con producto escalar se cumple $|x^T y| \leq \|x\| \cdot \|y\|$

Al mismo tiempo

$$\|d_k\| = \|\tilde{g}_k\| \leq \|\tilde{g}_k - g_k\| + \|g_k\| \leq \frac{1}{2}\|g_k\| + \|g_k\| \leq \frac{3}{2}\|g_k\| \tag{6.19}$$

Luego se satisface la hipótesis ii) del teorema 2.1.

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 2.1. 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 $0 < \epsilon << 1$. Luego $\tilde{\gamma} < 1$ y

$$\begin{aligned} f(x_{k+1}) &\leq f_{max} + \tilde{\gamma}\sigma^{h_i}\lambda_k \tilde{g}_k^T \tilde{g}_k = \\ &= f_{max} + \gamma\sigma^{h_i}\lambda_k g_k^T \tilde{g}_k = f_{max} + \gamma\sigma^{h_i}\lambda_k g_k^T d_k \end{aligned} \tag{6.20}$$

De 6.18, 6.19, 6.20 y el teorema 2.1 se cumplen las afirmaciones a), b) y c). ∎

Capítulo 7

Implementaciones

"Es posible escribir como nuevos muchos algoritmos
que no fueron considerados para la resolución de problemas numéricos
debido a sus reducidas prestaciones en su forma secuencial".
G. Buzzi-Ferraris

En este capítulo se describe cómo se implementó el algoritmo paralelo diseñado en el capítulo 6 utilizando dos plataformas con arquitecturas diferentes:

- **Sistema Distribuido (Plataforma 1)** Cluster conformado por 8 PC Pentium 4 de 3Ghz conectadas por una red Ethernet de 1 Gb, sistema operativo GNU/LINUX con distribución *Gentoo*.
- **Computadora Multicore (Plataforma 2)** Servidor AMD Opteron, 12 cores, 50GB RAM, Debian GNU/Linux 6.0 64 bits, kernel 3.8.3.

7.1 Implementación sobre sistemas distribuidos

La implementación sobre sistemas distribuidos se realizó utilizando el modelo maestro-trabajador y PVM como protocolo de pasaje de mensajes. Como muestra la figura **??**, cada paquete de mensajes tiene en el encabezado un código que representa cinco tipos de mensaje y dos acciones. Los tipos de mensaje `Interrumpir` y aquellos que comienzan

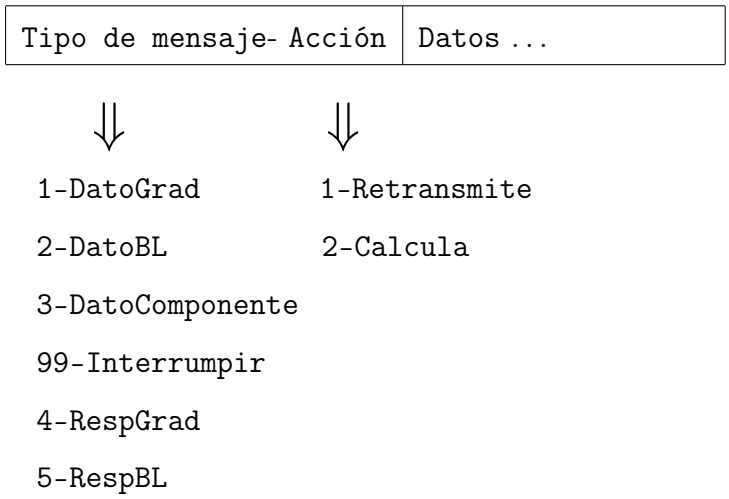


Figura 7.1: Encabezado de los mensajes

con el prefijo `Dato` son mensajes que van del proceso maestro a los trabajadores; los tipos de mensaje que comienzan con `Resp` son mensajes que fluyen de los trabajadores al maestro. Por ejemplo un mensaje con código 21 contiene datos para una búsqueda lineal que deben ser retransmitidos. Los trabajadores sólo se comunican entre sí cuando se hace una distribución de datos de tipo broadcasting *one to all* (ver capítulo 6) en donde algunos trabajadores son intermediarios entre el maestro y el destinatario final.

Como se puede ver en el Algoritmo 7, en los pasos 2 y 10 se evalúa el gradiente en paralelo y en el paso 7se realiza una búsqueda lineal no monótona en paralelo. Estos tramos en paralelo no se solapan, es decir, cuando se evalúa un gradiente no se hace búsqueda lineal y vice versa, entonces esto nos permite realizar un diseño con un control del tipo de mensajes que fluirán entre el maestro y los trabajadores, eliminando posibles mensajes rezagados.

Los mensajes se examinan a través de la subrutina `pvmfprobe()` con la que se puede verificar el tipo de mensaje sin desempaquetarlo.

El protocolo PVM garantiza que el orden de los mensajes se preserva. Si la tarea 1 le manda a la tarea 2 un mensaje A y luego un mensaje B, la tarea 2 recibirá primero el mensaje A y luego el B. Además si ambos mensajes llegan antes que la tarea 2 los haya recibido, entonces una recepción cuyo tipo de mensaje sea del modo comodín

Algoritmo 8 Cálculo del gradiente en un sistema distribuido

```
1: tipoMensaje = DatoGrad
2: compo=1
   { Mando a cada nodo el vector x_k y una componente}
3: Empaquetar {n, x_k, componente_i}
4: Enviar cada paquete a los p − 1 procesadores
5: tipoMensaje = RespGrad !Respuesta del trabajador
6: calculadas=0
7: enviadas = nproc
8: Mientras (calculadas < n) hacer
9:    Examinar si llegó un mensaje y verificar su tipo (valor de la variable
      tipoMsgRec)
10:   Si (tipoMsgRec ≠ tipoMensaje) entonces
11:      Borrar el mensaje
12:   Si no
13:      Desempaquetar {derparc, compo y tid } {La derivada parcial calculada, a qué
         componente corresponde y el identificador de tarea}
14:      grad(compo) = derParc
15:      calculadas = calculadas + 1
16:      Si (enviadas≠ n) entonces
17:         tipoMensaje = DatoComponente
18:         enviadas = enviadas + 1
19:         Empaquetar {enviadas} {Al trabajador con identificador tid le mando a
            calcular otra componente}
20:      Fin {Si}
21:      tipoMensaje = RespGrad
22:   Fin {Si}
23: Fin {Mientras}
```

(msgtype=-1) siempre va a retornar mensaje A [Geist *et al.*, 1994].

Para la búsqueda lineal no monótona el proceso *maestro* gestiona la distribución de estas tareas y, eventualmente, puede ejecutar también evaluaciones de función. Este modelo aumenta su eficiencia en proporción directa al aumento de la complejidad de la función objetivo, dado que la sobrecarga de comunicación puede volverse insignificante en relación al tiempo de cómputo que puede requerir la evaluación de dicha función [**?**].

Como muestra el algoritmo 6, en la búsqueda lineal se realizan cálculos especulativos cuya información puede quedar sin uso. Cuando el proceso maestro ya tiene lo que busca manda un mensaje general de interrupción. Si el mensaje de interrupción le llega al trabajador con posterioridad al envío de la respuesta, ese mensaje de respuesta llega al buffer del maestro y allí es borrado.

7.2 Implementación sobre máquinas paralelas

7.2.1 OpenMP

OpenMP puede verse como una notación que puede ser agregada a un programa secuencial en Fortran, C o C++, para describir la manera en que el trabajo se comparte entre los hilos que se ejecutarán en diferentes procesadores o núcleos, y para ordenar los accesos a los datos compartidos cuando son necesarios. La inserción apropiada de características OpenMP en un programa secuencial permitirá que varias, o quizás muchas, aplicaciones se beneficien de las arquitecturas paralelas de memoria compartida. En la práctica, muchas aplicaciones poseen considerable paralelismo que puede ser explotado

OpenMP se basa en el modelo de memoria compartida; por lo tanto, por defecto, los datos son compartidos entre los hilos y son visibles a todos ellos. Sin embargo, muchas veces es necesario contar con variables que posean valores específicos del hilo. Cuando cada hilo posee su propia copia de una variable, ésta potencialmente podría tener un valor diferente para cada hilo, se trata de una variable privada. Los datos pueden ser

declarados como compartidos o privados con respecto a una región paralela o constructor de trabajo compartido

El uso de variables privadas puede ser beneficioso de varias maneras. Pueden reducir la frecuencia de actualizaciones a la memoria compartida. Luego, pueden ayudar a evitar hot spots, o competencia por acceso a ciertas locaciones de memoria, las cuales pueden resultar costosas si están involucrados una gran cantidad de hilos. Por defecto, OpenMP hace que los hilos esperen al final de un constructor de trabajo compartido o región paralela, hasta que todos los hilos en el equipo ejecutándola hayan finalizado sus porciones de trabajo. Sólo luego podrán continuar. La sincronización de las acciones de un subconjunto de hilos es difícil de lograr en OpenMP, y requiere cuidado en la programación debido a que no existe un soporte explícito para ello. Muchas veces un programador puede necesitar asegurar que sólo un hilo a la vez trabaje sobre una pieza de código. OpenMP posee varios mecanismos que soportan esta clase de sincronización.

El gradiente en paralelo incluye la siguiente linea que indica qué variables serán privadas (locales) en cada hilo (`PRIVATE`) y cuáles estarán disponibles para todos los hilos (`SHARED`)

```
!$OMP PARALLEL DO DEFAULT(NONE), SHARED(x, n, grad), PRIVATE(compo, h,
fmas, fmenos, derParc, informf)
```

Las variables locales comienzan indefinidas en cada hilo, salvo que se las indique como `FIRSTPRIVATE`, lo que le asigna a estas variables el valor que tenían antes de iniciar la región paralela. En la búsqueda lineal no monótona un conjunto de variables son designadas de esta manera.

```
!$OMP PARALLEL DEFAULT(NONE), SHARED(exitos, longitudes, fnews), PRIVATE(
indice, longPaso, i, xnew, fdelp, inform), FIRSTPRIVATE(nproc, k, n, x, d,
fmax, gtd)
```

Como fue descripto en la sección 6.2, en esta búsqueda lineal el proceso *maestro* no necesariamente tiene que esperar los resultados de todos los *trabajadores*. La sincronización implícita de OpenMP se desactiva mediante el comando `NOWAIT`. La directiva `!$OMP FLUSH` actualiza los valores de las variables compartidas [Openmp, 2008].

7.2.2 MATLAB: Parallel Toolbox

El *Parallel Computing Toolbox* de MATLAB permite al usuario ejecutar un trabajo en paralelo usando 4 'labs' o *trabajadores* (copias adicionales de MATLAB) que asisten a la copia principal (cliente). Si la máquina tiene múltiples procesadores, los 'labs' se pueden activar a partir del comando `matlabpool open`. Este tipo de computación en MATLAB es muy similar al paradigma de memoria compartida habilitada con OpenMP, pero en este caso desarrollado de forma más sencilla e intuitiva. En el modo paralelo (`pmode`), con el comando `Drange` la distribución de tareas es controlada por el indice del bucle. Las variables o vectores predefinidas en el cliente son accesibles como locales en los *trabajadores*.

Capítulo 8

Aplicaciones sobre problemas industriales

8.1 Los problemas asociados a columnas de separación

Los procesos de separación constituyen una parte importante de la inversión total de capital y gastos de funcionamiento de una planta de procesos. El desarrollo de enfoques sistemáticos a fin de seleccionar la secuencia óptima de columnas de separación deriva en problemas altamente no lineales con gran cantidad de restricciones de igualdad y desigualdad. Por consiguiente, la solución de estos problemas consiste en la exploración de espacios de búsqueda muy complejos [Kheawhom, 2010]. Los problemas NLP que se generan representan los escenarios de un problema general MINLP. En esta tesis, el abordaje de los problemas NLP es en términos de computación paralela a través del algoritmo pALSPG desarrollado en el capítulo 6.

Los problemas clásicos tratados en este trabajo pueden establecerse de la siguiente manera. Una corriente de alimentación multicomponente, cuyos parámetros (velocidad de flujo, composición, temperatura y presión) son conocidos, tiene que ser separada en un cierto número de productos multicomponentes, cuyas composiciones se especifican. Entonces, el problema consiste en la sintetización de una secuencia de destilación óptima que debe satisfacer el criterio de ofrecer un costo anual total mínimo. Esto da lugar

a varios escenarios posibles formulados como modelos NLP que implican una función objetivo no lineal (costo total anual) sometida a un conjunto no lineal de restricciones. Estos problemas suelen abordarse mediante la fijación del valor numérico de las llamadas las variables 'complicadas' (complicating variables), convirtiendose así el problema general en uno más simple de programación lineal [Aggarwal & Floudas, 1990].

8.1.1 Casos de estudio

Los casos de estudio cuyo análisis presentamos en esta tesis son los siguientes:
Caso 1: Síntesis de un sistema de separación basado en Destilación . Este problema involucra una mezcla de alimentación de tres componentes que tiene que ser separado en dos productos de múltiples componentes. El costo de cada separador depende linealmente de la velocidad de flujo a través de este, y las limitaciones corresponden a los balances de masa alrededor de los divisores, separadores y diversos mezcladores. Este problema NLP es parte de una superestructura MINLP. Para realizar este test utilizamos la misma configuración establecida por Floudas en la descripción del problema [Floudas & Pardalos, 1990].

Caso 2: Separación Nonsharp de Propano, Isobutano y n-Butano. Este caso consiste en una mezcla de alimentación de tres componentes que tiene que ser separado en dos productos de tres componentes. Para evitar la distribución de componentes no clave los porcentajes de recuperación de los componentes clave se establecieron para ser mayor que 0.85. Este problema tiene la misma superestructura del caso 1, pero la composición del producto deseado es diferente [Aggarwal & Floudas, 1990].

Caso 3: Separación de Propano, Isobutano, n-Butano e Isopentano. En este problema una mezcla de cuatro componentes tiene que ser separado en dos productos de múltiples componentes. El número de columnas NC es 3 [Aggarwal & Floudas, 1990].

Estos casos de estudio fueron formulados en el trabajo de Floudas y Pardalos [Floudas & Pardalos, 1990]. En este texto el caso de prueba 1 corresponde a la sección 5.4, mientras que la formulación general del problema para los casos 2 y 3 viene dada

en las secciones 5.5 y 5.6 respectivamente. Las variables binarias b_j, $j = 1 \cdots NC$ están incluidas en la función objetivo. Para los casos de estudio 2 y 3 fueron todas seteadas en 1 para denotar la existencia de 2 y 3 columnas respectivamente.

El tipo de problemas de los que pueden obtenerse beneficios de la programación paralela se pueden agrupar bajo el título de 'problemas difíciles de manejar', ya que insumen un tiempo de cómputo significativo. Este se debe a diversas razones, como son la incorporación de modelos termodinámicos, o gran cantidad de restricciones. En particular, la destilación reactiva muestra el comportamiento típico de un problema computacionalmente exigente. Este modelo de ecuaciones habitualmente se resuelven por medio de un método de relajación combinada con el método de Newton para el cálculo de las variables en cada punto del tiempo. Sin embargo, Taylor y Krishna han señalado que los métodos de relajación no se utilizan a menudo porque se considera que son demasiado exigentes en tiempo de cálculo [Taylor & Krishna, 2000]. Otro enfoque consiste en reescribir las ecuaciones mediante la definición de variables transformadas [Barbosa & Doherty, 1987]. De esta manera, el orden se reduce con un método de colocación a fin de ahorrar tiempo de cálculo. En vista de estas desafiantes características, el siguiente caso de estudio fue incorporado en este análisis.

Caso 4: Columna reactiva [Domancich *et al.*, 2009]. El modelo para una columna de destilación reactiva consiste en etapas de separación puras combinadas con un sector de destilación reactiva, en donde tanto la reacción y la separación se llevan a cabo al mismo tiempo. En este ejemplo, cantidades equimolares de metanol e i-$C_4^=$ son alimentados en proporción estequiométrica, y un producto con una alta pureza de éter terc-butil-metil-se obtiene (MTBE). Un modelado riguroso de la columna se emplea, con un modelo de equilibrio para el sector reactivo. La columna tiene 15 bandejas, siendo las etapas reactivas para el caso base las etapas de la 2 a la 9. Este es un problema grande, computacionalmente exigente porque más del 60 % del conjunto total de ecuaciones en este modelo se compone de engorrosas funciones de equilibrio termodinámico.

Como herramienta de modelado se utilizó la plataforma GAMS. Estos modelos se

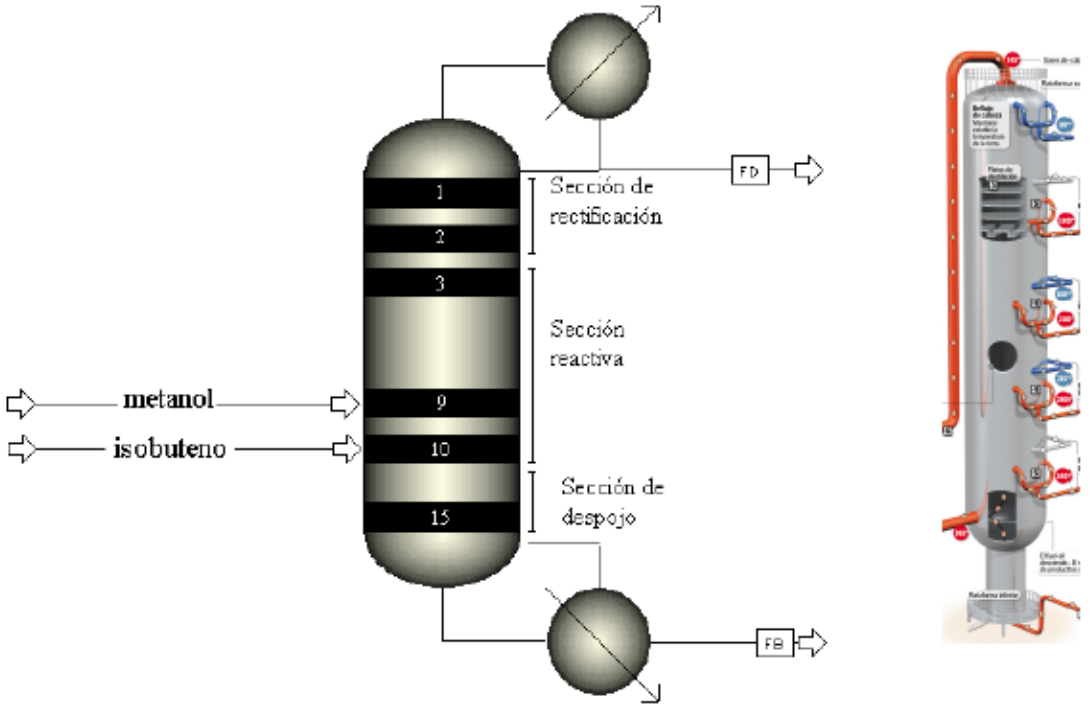


Figura 8.1: Representación esquemática de una columna reactiva

convirtieron a modelos escalares a través del módulo convertidor `CONVERT.gms`[1] que es provisto por la misma plataforma GAMS.

La dimensión de los casos de prueba se presentan en la Tabla 8.1. La resolución de estos problemas fue abordada con ocho resolvedores clásicos proporcionadas por GAMS y con nuestro resolvedor en paralelo sobre las plataformas descriptas en el Capítulo 7. Cada uno de estos problemas representa una etapa particular de un problema más general. La etapa es un problema NLP derivado de una configuración particular de variables binarias en un problema general MINLP. Por lo tanto, es de vital importancia no sólo para resolver las etapas, sino también ser capaz de hacerlo de una manera eficiente.

[1]CONVERT.gms es una utilidad que transforma una instancia de modelo GAMS en un modelo escalar donde toda la información confidencial es eliminada o convertida a formatos utilizados por otros sistemas de modelado y solución provisto por GAMS.

Caso de estudio	1	2	3	4
Número de variables continuas	38	48	86	1173
Número de restricciones lineales	17	13	22	227
Número de restricciones no lineales	15	25	46	930

Tabla 8.1: Estadística de de los modelos para cada caso de prueba.

8.1.2 Acerca de la eficacia de los resolvedores

Para fines comparativos, la batería de pruebas elegida contiene los solucionadores ofrecidos por GAMS. El modelo matemático es no convexo en los casos de prueba 1, 2, 3 y 4. Para los tres primeros, el rendimiento se resume en la tabla 8.2, donde los resultados producidos por un conjunto de resolvedores adecuados de alto rendimiento se informa junto con las dados por la propuesta descrita en este trabajo (pALSPG) [GAMS, 2013]. F_{obj} indica el valor final de la función objetivo y el nivel de factibilidad se calcula como la norma del vector dado por la evaluación de las restricciones: $\|C(x_\star)\|$. El tiempo de ejecución de cada resolvedor se reporta en la sección 8.1.4.

El resolvedor BARON pudo encontrar una solución para la casos de estudio 1 y 3, pero el nivel de factibilidad informado no resultó satisfactorio. Como es habitual en la literatura, se estableció la tolerancia final para la factibilidad $\eta_\star = 10^{-4}$ [Diniz-Ehrhardt *et al.*, 2004, Birgin *et al.*, 2010]. Para el caso de prueba 2 BARON informó 'Cotas demasiado amplias - Estado del modelo incierto' (Bounds too wide - model status uncertain) y no devuelve solución. En contraste, IPOPT encontró soluciones satisfactorias para los casos de prueba 1 y 2, pero superó el límite de iteraciones (1000) para el caso de prueba 3.

MINOS5 iterrumpió los casos de prueba 1 y 2 reportando que no pudo encontrar un punto factible, mientras que en el caso de prueba 3 el reporte del resolvedor fue que el problema no está acotado (o está mal escalado). Por otro lado, SNOPT encontró soluciones factibles para los tres casos de prueba.

Para los problemas elegidos, PATHNLP y LINDO mostraron resultados opuestos.

	Caso de estudio 1		Caso de estudio 2		Caso de estudio 3	
	F_{obj}	$\|C(x_\star)\|$	F_{obj}	$\|C(x_\star)\|$	F_{obj}	$\|C(x_\star)\|$
pALSPG	1.861	1.1×10^{-11}	0.998	1.2×10^{-8}	1.645	4.6×10^{-8}
BARON	1.861	1.1×10^{2}	Cotas demasiado amplias		1.668	$2.9\ 10^{2}$
IPOPT	1.861	$1.7\ 10^{-10}$	0.997	$9.5\ 10^{-5}$	Máximo de iteraciones	
LINDO	1.861	$3\ 10^{-6}$	Localmente no factible		Localmente no factible	
PATHNLP	Localmente no factible		0,997	$1.6\ 10^{-8}$	1,688	$2.9\ 10^{-7}$
SNOPT	1.861	$8.3\ 10^{-1}$	0.997	9.410^{-1}	1.687	$3.1\ 10^{-7}$
CONOPT	1.861	9.110^{-1}	Solución no factible		1.687	1.410^{1}
MINOS5	No factible		No factible		Problema no acotado	
KNITRO	1.861	2.210^{-7}	Máximo de iteraciones		Máximo de iteraciones	

Tabla 8.2: Valor final de la función objetivo y norma de las restricciones en el punto solución

Mientras PATHNLP encontró buenas soluciones para los casos de prueba 2 y 3 y devolvió que el modelo es inviable a nivel local para el caso de prueba 3, LINDO produjo el mismo estatus de modelo para los casos de prueba 1 y 2, pero obtuvo una solución factible para el caso de prueba 3.

CONOPT encontró una solución factible en el caso de estudio 1, declaró como no factible el caso 2, y encontró una solución en el caso de prueba 3 no satisfactoria con respecto a su factibilidad.

Por último, KNITRO llegó a una solución factible en el caso de estudio 1, pero interrumpió la búsqueda en los casos 2 y 3 debido a que en ambos casos se superó el límite de iteraciones (1000).

pALSPG exhibió la atractiva ventaja de ser robusto, ya que alcanzó una solución factible en los tres casos de estudio. De hecho, la tabla 8.2 muestra que sólo pALSPG y SNOPT mostraron esta performance. Para el caso de prueba 1 el valor de la función objetivo coincide con el encontrado por BARON, SNOPT, CONOPT, LINDO, IPOPT

y KNITRO, pero sólo con los últimos tres resolvedores coincidió con un valor de norma del vector de las restricciones significativamente bajo. En el caso de prueba 2 pALSPG fue tan eficaz con IPOPT, PATHNLP y SNOPT. En este caso la calidad de la solución dada por SNOPT es ligeramente inferior. Por último, en el caso de estudio 3 pALSPG, PATHNLP y SNOPT encuentran las mejores soluciones desde el punto de vista de factibilidad.

8.1.3 Acerca de la aplicación del paralelismo

El paralelismo es un recurso computacional que sirve para reducir tiempos de ejecución. El caso de prueba 4, que es el ejemplo de mayor tamaño entre los elegidos aquí, muestra la utilidad de paralelización. Si el problema es grande, podría ser no sólo difícil de manejar, sino también económicamente inviable con respecto a las licencias requeridas por las herramientas de optimización para problemas de ese porte. Entonces, para el caso de prueba 4 sólo los solucionadores de licencia libre que soportan el tamaño del problema fueron incluidos en la evaluación.

MINOS5, IPOPT y BARON fueron adecuados para resolver este problema. Sin embargo, la mejor solución fue obtenida por CONOPT ($F_{obj} = 2244.7$; $\|C(x_\star)\| = 2.0 \times 10^{-2}$). ALSGP fue efectivo ya que reportó valores similares ($F_{obj} = 2245.6$; $\|C(x_\star)\| = 3.1 \times 10^{-2}$). La estrategia algorítmica del ALSPG secuencial se centra básicamente en la eficacia, es decir, en la búsqueda de una solución satisfactoria. Si es necesario, la eficiencia de ALSPG bien puede ser lograda por medio del paralelismo. Mediante el uso de pALSGP nos podemos permitir el abordaje de problemas NLP muy grandes y conseguir una solución satisfactoria en tiempos de ejecución razonables.

8.1.4 Acerca de la eficiencia del pALSPG

A partir de la eficacia del algoritmo ALSPG en los casos de estudio, nuestro interés se centra en la reducción del tiempo de ejecución, es decir el logro de un mayor rendimiento mediante el algoritmo paralelo.

	ALSPG			
	Caso 1	**Caso 2**	**Caso 3**	**Caso 4**
	65.74	122.46	52.58	606.87
N^o proc.	pALSPG			
2	46.92	85.04	39.23	332.92
3	31.12	62.3	29.34	250.55
4	20.54	38.75	18.06	195.98
5	14.67	32.44	15.78	133.4
6	12.11	28.78	12.3	111.6
7	12.02	22.3	11.5	98.5
8	11.82	21.26	10.12	98.1

Tabla 8.3: Tiempo de CPU (segundos) sobre cluster

Las implementaciones descriptas en el Capítulo 7 arrojan los tiempos de ejecución que se muestran en las tablas 8.3 y C.3.

Las tablas C.2 y 8.6 muestran la eficiencia de cada implementación paralela.

Para un problema de tamaño fijo, a medida que el número de elementos de procesamiento aumenta, la eficiencia global del sistema paralelo tiende a disminuir. Este fenómeno es común a todos los sistemas paralelos [Grama *et al.*, 2003]. Si bien disminuye la eficiencia, la curva de *speed-up* se mantiene creciente como muestra la Figura 8.5.

El porcentaje de reducción absoluta del tiempo se calcula a través de la fórmula 8.1.

$$Red_{abs} = 100\left(1 - \frac{1}{S_p}\right) \tag{8.1}$$

Las reducciones de tiempo son cercanas al 90%. Esto se muestra en las figura 8.7. En el caso de la plataforma multicore se da un repunte en la eficiencia al utilizar 9 hilos. Esto se debe al efecto de la búsqueda lineal descripto en el Teorema 6.3.

	ALSPG			
	Caso 1	**Caso 2**	**Caso 3**	**Caso 4**
	53.22	98.44	43.66	488.77
N^o hilos	pALSPG			
2	30.82	61.1	23.81	283.05
3	27.73	55.98	21.75	254.65
4	24.74	49.12	19.3	245.51
5	21.82	43.31	17.18	204.55
6	18.71	37.25	15.11	183.43
7	15.83	31.37	12.93	155.06
8	12.74	25.71	10.77	117.02
9	9.71	20.2	8.62	92.2
10	6.55	12.32	5.01	88.39
11	6.61	12.36	5.06	57.56
12	6.22	12.74	5.04	62.37

Tabla 8.4: Tiempo de CPU (segundos) sobre una arquitectura multicore

N^o de proc.	**Caso 1**	**Caso 2**	**Caso 3**	**Caso 4**
2	70.06	72	67.02	91.14
3	70.42	65.52	59.74	80.74
4	80.01	79.01	72.79	77.41
5	89.63	75.5	66.64	90.99
6	90.48	70.92	71.25	90.63
7	78.13	78.45	65.32	88.02
8	69.52	72	64.95	77.33

Tabla 8.5: Eficiencias obtenidas sobre un cluster

N^o de hilos	**Caso 1**	**Caso 2**	**Caso 3**	**Caso 4**
2	86.34	80.56	91.67	82.34
3	63.98	58.6	66.91	61.98
4	53.77	50.1	56.56	51.77
5	48.79	45.46	50.83	44.79
6	47.41	44.05	48.15	47.41
7	48.03	44.83	48.22	44.03
8	52.21	47.87	50.65	48.21
9	60.9	54.14	56.26	55.9
10	81.2	79.88	87.3	78.3
11	77.5	72.4	85.2	77.2
12	71.3	64.4	74.2	65.3

Tabla 8.6: Eficiencias obtenidas sobre una arquitectura multicore

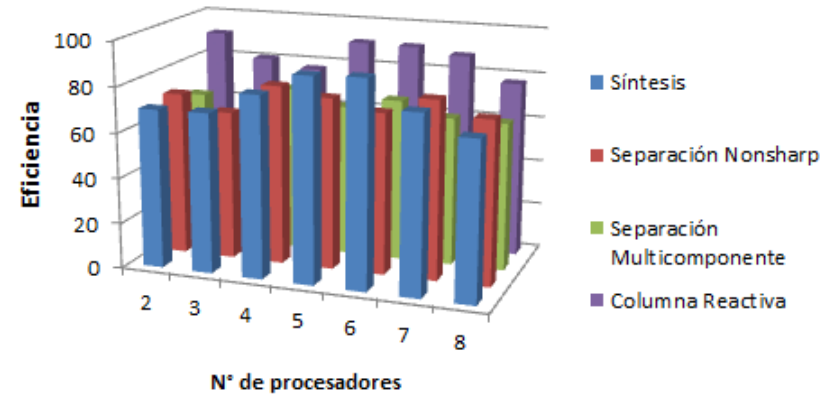


Figura 8.2: Eficiencia para los casos de estudio 1, 2, 3 y 4 sobre un cluster

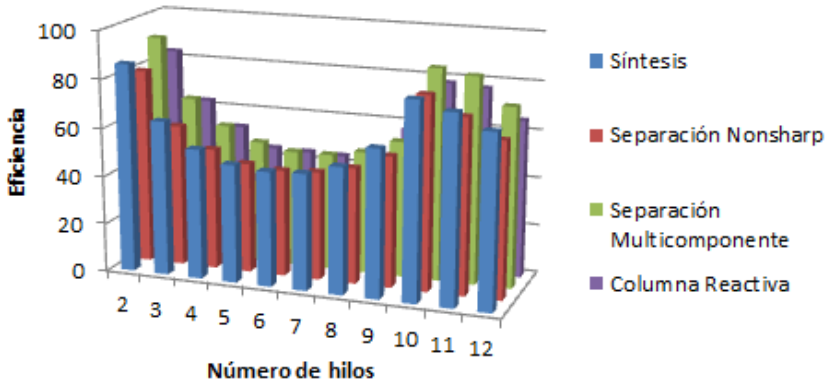


Figura 8.3: Eficiencia para los casos de estudio 1, 2, 3 y 4 sobre una plataforma multicore

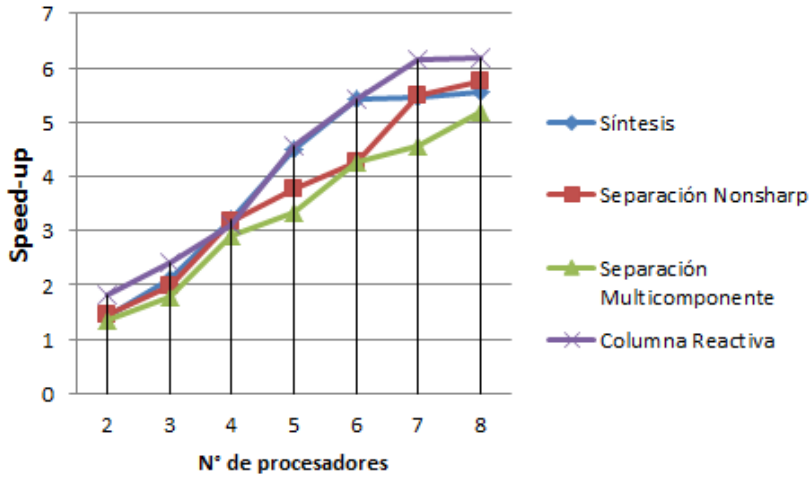


Figura 8.4: Curvas de Speed-up para los casos de estudio 1, 2, 3 y 4 sobre un cluster

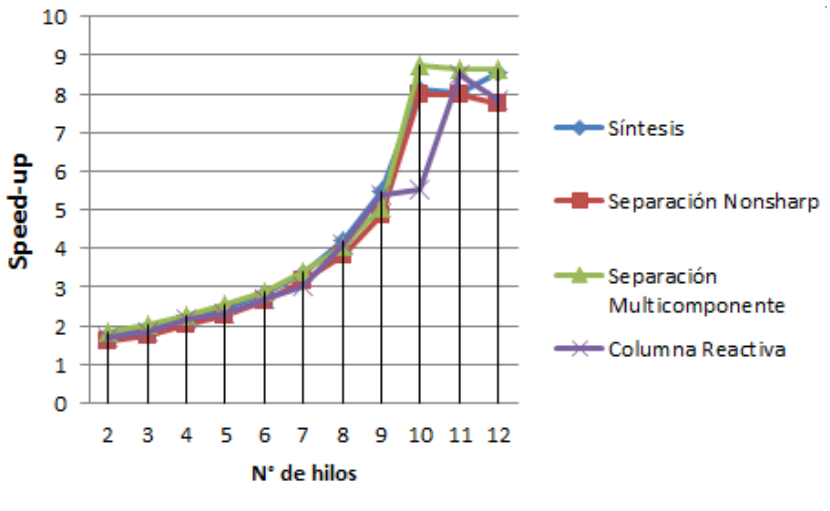


Figura 8.5: Curvas de Speed-up para los casos de estudio 1, 2, 3 y 4 sobre una plataforma multicore

	Promedios	
N° proc.	Eficiencia	Reducción
2	75,05	32,43
3	69,1	51,18
4	77,3	67,62
5	80,69	74,8
6	80,82	79,07
7	77,48	81,35
8	70,95	82,31

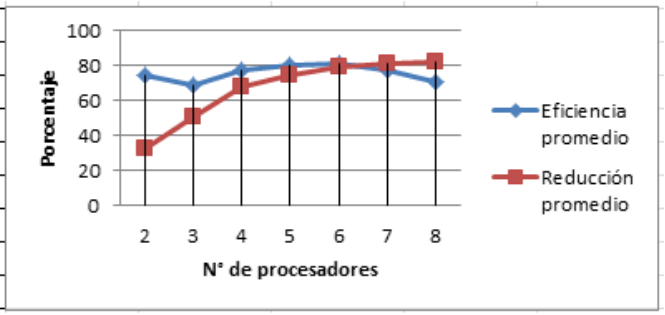


Figura 8.6: Promedio de eficiencia y de reducción absoluta de tiempo sobre un cluster

	Promedios	
N° hilos	Eficiencia	Reducción
2	86,23	41,72
3	62,87	46,63
4	53,55	52,78
5	47,22	58,67
6	45,51	63,5
7	47,28	69,6
8	50,24	74,83
9	57,55	80,57
10	75,92	86,41
11	75,3	87,91
12	68,3	87,77

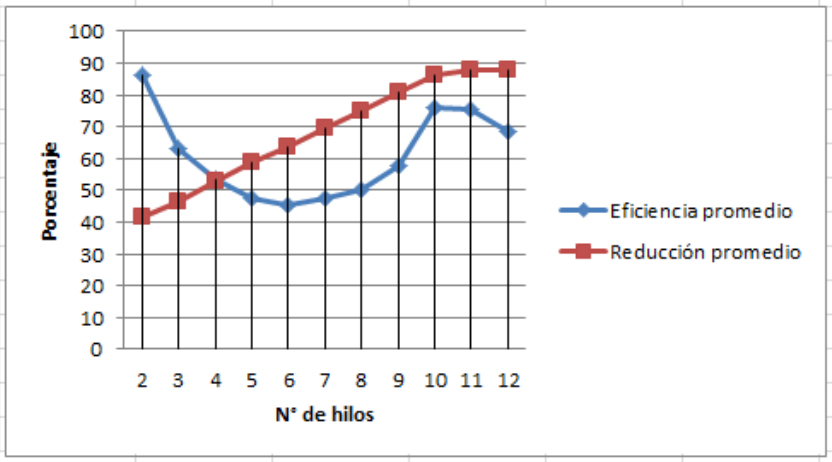


Figura 8.7: Promedio de eficiencia y de reducción absoluta de tiempo sobre una plataforma multicore

8.1.5 Evaluación comparativa de desempeño

La figura 8.8 muestra los tiempos de CPU para los resolvedores provistos por GAMS elegidos previamente junto con los de pALSPG en sus dos versiones paralelas, para los casos de estudio 1, 2 y 3. En el gráfico las barras planas, como las de MINOS5, indican que el resolvedor no tuvo éxito en la búsqueda de una solución, lo que muestra la eficacia de pALSPG. En contraste, los tiempos insumidos por pALSPG son mayores, GAMS demostró ser más rápido en los casos en que logró obtener una solución al problema.

En contraste, pALSPG es a veces más lento, siendo menos eficiente, pero realmente eficaz. Aunque algunos casos de estudio pueden requerir una cantidad significativa de tiempo de cálculo, este problema de eficiencia en tiempo se anula a través de la programación en paralelo como se ha afirmado en el capítulo 5.

La figura 8.9 muestra el perfil de performance descripto en el Capítulo 4 para estos cuatro casos de estudio.

El perfil de rendimiento permite clasificar a pALSPG respecto a los otros resolvedores. La figura 8.9 muestra a pALSPG y SNOPT como los solucionadores más competitivos para los casos 1, 2 y 3, ya que tienen la mayor probabilidad de ser el solucionador óptimo. La probabilidad de pALSPG está cerca de 1 a partir de $m = 18, 8$.

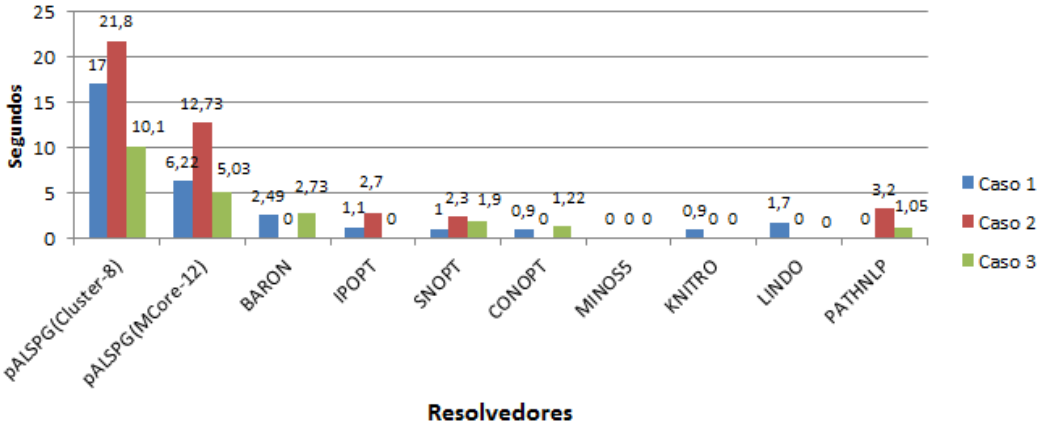


Figura 8.8: Comparación de tiempos de CPU entre resolvedores.

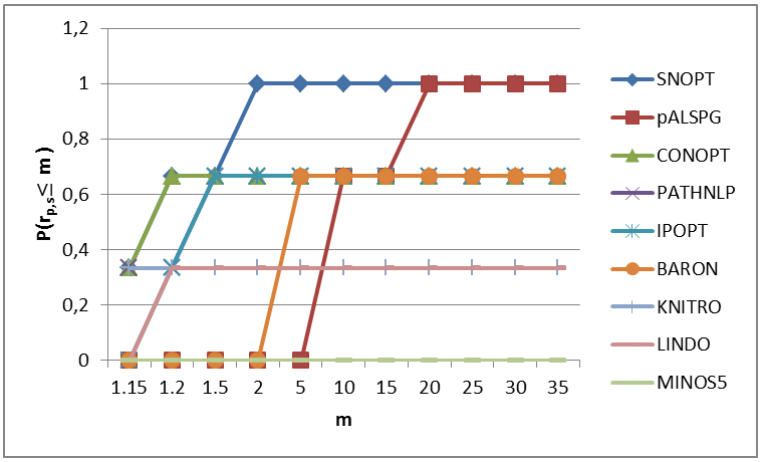


Figura 8.9: Perfil de performance para un conjunto de resolvedores

Sin embargo, su rendimiento sería considerado como menos competitivo si limitamos nuestra m de interés a 10.

8.2 El problema de asignación de servicios

Este problema tiene aplicación directa en la decisión económica involucrada en la ubicación de determinadas instalaciones para responder a ciertas demandas en forma eficiente. Ejemplos de esto son la ubicación de fábricas, depósitos, almacenes, bibliotecas, estaciones de bomberos u hospitales. En ingeniería de procesos hay una relación directa con el problema óptimo de logística de transporte. Los modelos matemáticos derivados son de tamaño medio o chico. Birgin et al. aplicaron el algoritmo SPG para estos problemas obteniendo resultados muy satisfactorios [Birgin *et al.*, 2001a]. Pero cuando este modelo se aplica para determinar la ubicación de estaciones de base para los servicios inalámbricos (como difusión de TV o de servicios de telefonía móvil), servicios cuyo auge resultó posterior al desarrollo es estos modelos, aparece un conjunto significativamente mayor de puntos de demanda (en este caso representando a los usuarios y clientes) [Vygen, 2005]. La resolución de este problema, ahora de gran escala, exige tiempos de cálculo extremadamente altos.

8.2.1 Descripción

El problema de la asignación de instalaciones considerado en esta sección fue descripto por Birgin et al. de la siguiente forma: Se considera una malla rectangular de puntos en $\mathbb{R}^2$. 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 la 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 ponderadas desde las torres de cada una de las plantas a la torre central. En otras palabras, el problema consiste en encontrar la ubicación de un punto central que reduzca al mínimo la suma de las distancias a un conjunto de puntos en una región de demanda. Esta es una variación del problema de localización de Fermat-Weber clásica ya que el conjunto de puntos de demanda es variable [Brimberg *et al.*, 1998]. Este problema, al poder modelarse geométricamente mediante el uso de polígonos, ha estimulado el desarrollo de algoritmos dedicados a la geometría computacional [Abbasi & Younis, 2007].

La formulación matemática del problema es la siguiente: dado un conjunto de *npol* polígonos disjuntos $P_1, P_2, \cdots, P_{npol}$, si P_1 representa el área central donde el punto c será ubicado, entonces el objetivo es determinar las coordenadas de dicho punto c y de $x_i \in P_i$, $i = 2, \cdots, npol$ para minimizar la suma de los costos que insume conectar cada región con la región central. Como el costo no es igual en los diferentes puntos, se incluyen pesos en cada conexión. Luego, dado un conjunto de números positivos w_i, $i = 2, \cdots, npol$, el problema se puede plantear como

$$\min_{x_i, i=2,\cdots npol} \quad \sum_{i=2}^{npol} w_i \|x_i - c\| \tag{8.2}$$

$$\text{sujeto a} \quad c \in P_1 \tag{8.3}$$

$$x_i \in P_i,\ i = 2, \cdots, npol \tag{8.4}$$

El problema tiene $2.npol$ variables y $\sum_{i=1}^{npol} v_i$ restricciones, donde v_i es el número de vértices del polígono P_i.

8.2.2 Experimentos numéricos

Con el fin de evaluar el rendimiento de pALSPG se seleccionaron setenta y tres problemas con diferentes tamaños de retícula y número de polígonos. Los problemas fueron

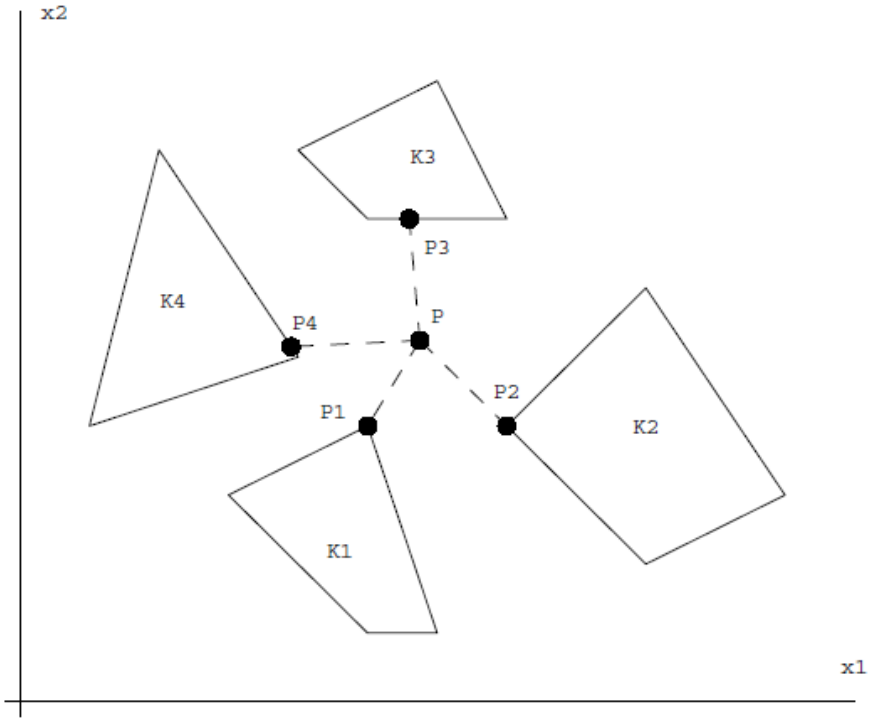


Figura 8.10: Posible configuración óptima del problema de asignación

creados con el generador de polígonos de Birgin, disponible en *TOMS* (Transactions on Software Matemático) [Birgin *et al.*, 2001b]. Un conjunto de parámetros determina un identificador del problema, el número de puntos horizontales y verticales en la red, la probabilidad de tener un polígono en un punto de la rejilla, y el número mínimo y máximo de vértices de los polígonos, respectivamente.

Para medir el rendimiento del algoritmo paralelo se utilizaron las dos plataformas paralelas descriptas en el Capítulo 7.

Resultados sobre el cluster

El conjunto de problemas numerados del 1 al 30 tienen un tamaño medio y sus detalles se exponen en la tabla 8.7. En dicha tabla `Prob Nº` provee una referencia a cada problema, `Polig` es el número de polígonos, `Vert`es el número de vértices y `tiempo` indica el tiempo de CPU en segundos insumido por el algoritmo ALSPG para resolverlo. Notamos con

`Grilla` el tamaño de la grilla de puntos para ese conjunto de problemas.

La figura 8.11 muestra el promedio de eficiencia y reducción absoluta de tiempo obtenidos. Las tablas con los detalles de cada problema se pueden ver en el anexo C.

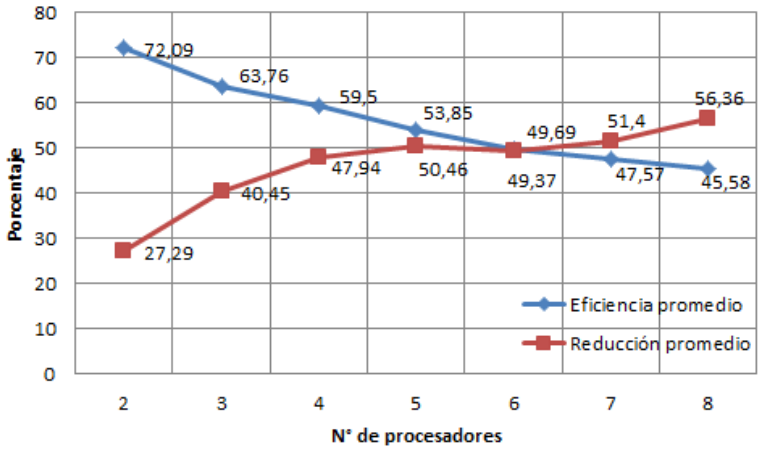


Figura 8.11: Porcentaje de eficiencia y reducción absoluta de tiempo (promedio)

La eficiencia del algoritmo paralelo mejora con los problemas de mayor tamaño, especialmente en aquellos en los que la versión secuencial demandó un gran número de búsquedas en línea. En aquellos problemas donde esta demanda no es tal, el algoritmo paralelo no alcanza una eficiencia relevante. La reducción de tiempo no es sustancial en aquellos problemas en los que el tiempo secuencial es muy bajo. El estudio de problemas más grandes sobre la plataforma multicore revela el efecto de la búsqueda lineal en paralelo en relación a la cantidad de búsquedas qeu cada problema realiza por iteración.

Resultados sobre la plataforma multicore

Sobre la plataforma 2 se resolvieron otros 45 problemas con grillas de tamaño medio y grande. Las tablas 8.8 y 8.9 muestran las características de cada problema con una descripción ampliada respecto a la realizada en la sección 8.2.2. Se agregan los datos `nIt`: número de iteraciones, `nBl`: número de búsquedas lineales realizadas, `Prom.`: promedio de búsquedas por iteración y `MaxBl`: pico máximo de búsquedas lineales en una iteración.

Prob. n^o	Grilla	Polig.	Vert.	Tiempo(seg.)
1	100	98	399	0.06
2	x	95	778	0.04
3	100	93	1152	0.04
4		173	705	0.18
5		178	1483	0.17
6		174	2219	0.15
7		259	1057	0.18
8		262	2161	0.37
9		256	3210	0.37
10		364	1448	0.73
11		369	2965	0.62
12		362	4358	0.68
13		457	1821	0.67
14		481	3852	0.96
15		456	5468	0.73
16	100	939	3787	4.97
17	x	928	7529	4.72
18	1000	941	11405	4.48
19		1873	7528	26.84
20		1851	14978	20.99
21		1848	22327	18.54
22		2851	11459	117.8
23		2821	22805	73.4
24		2854	34319	71.59
25		3830	15350	98.78
26		3794	30654	90.48
27		3853	46454	130.35
28		4746	18987	136.93
29		4767	38479	148.36
30		4814	57787	213.89

Tabla 8.7: Dimensión de los problemas y tiempo de ejecución de ALSPG

Este reporte muestra que en aquellos problemas con iteraciones que concentran muchas búsquedas lineales, el procedimiento de búsqueda en paralelo aumenta la eficiencia en las plataformas con un número mayor de procesadores. Al mismo tiempo, el aporte de este procedimiento en paralelo es menor en aquellos problemas en el que el número de búsquedas lineales por iteración es más equilibrado.

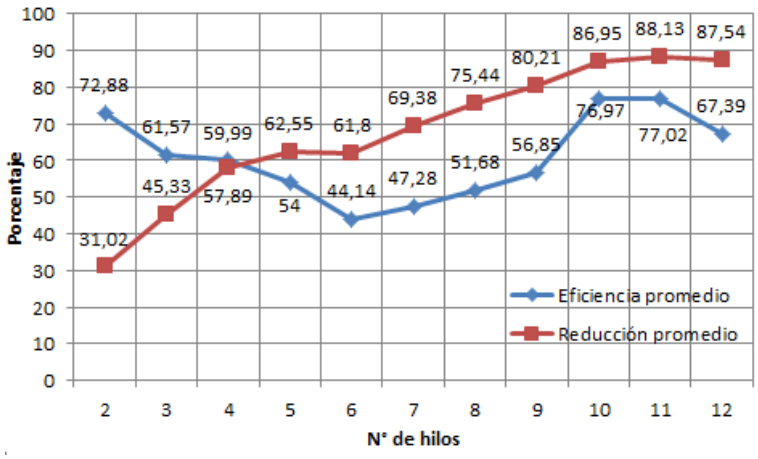


Figura 8.12: Porcentaje de eficiencia y reducción absoluta de tiempo (promedio)

El detalle de las eficiencias obtenidas por problema se puede ver en la tabla C.

Prob N^o	Grilla	Polig.	Vert.	Tiempo(seg)	nIt	nBl	Prom	MaxBl
1	1000	10014	30039	8,63	12211	25822	2,11	26
2	X	10081	30240	8,98	12222	26959	2,21	30
3	1000	10124	30369	7,54	12229	17450	1,43	35
4		20046	60135	10,71	13318	29477	2,21	28
5		20126	60375	16,79	13324	25746	1,93	16
6		20273	60816	11,49	13335	18636	1,4	6
7		29949	89844	19,14	13941	22005	1,58	7
8		30004	90009	17,36	13944	25752	1,85	7
9		30097	90288	15,38	13948	31487	2,26	17
10		39992	119973	4,81	14385	32580	2,26	30
11		40014	120039	35,51	14386	20969	1,46	4
12		40179	120534	42,17	14392	32678	2,27	32
13		49610	148827	24,33	14715	33214	2,26	29
14		49760	149277	23,52	14719	26278	1,79	7
15		49763	149286	36,68	14719	30914	2,1	20
16	1000	99412	298233	105,06	15770	22738	1,44	16
17	X	99780	299337	118,79	15775	27160	1,72	23
18	10000	99843	299526	166,54	15776	34955	2,22	24
19		200081	600240	182,6	16825	35201	2,09	29
20		200106	600315	242,16	16826	38018	2,26	33
21		200413	601236	312,84	16828	32911	1,96	24
22		299931	899790	229,52	17435	23288	1,34	6
23		299959	899874	364,84	17435	37470	2,15	4
24		300273	900816	145,46	17436	38951	2,23	35
25		399104	1197309	475,09	17864	35348	1,98	27
26		399562	1198683	202,63	17866	36763	2,06	7
27		400704	1202109	204,31	17870	36510	2,04	17
28		499629	1498884	538,58	18202	30801	1,69	29
29		500110	1500327	496,09	18203	35595	1,96	19
30		500424	1501269	165,11	18204	26781	1,47	18

Tabla 8.8: Reporte de resolución de ALSPG en problemas de tamaño medio.

Prob N°	Grilla	Polig.	Vert.	Tiempo(seg)	nIt	nBl	Prom	MaxBl
31	10000	998355	2995062	2283,99	19241	38598	2,01	31
32	X	1000662	3001983	2362,07	19245	25631	1,33	26
33	10000	1002115	3006342	2410,37	19247	30351	1,58	7
34		1997548	5992641	2699,1	20282	27303	1,35	23
35		1998762	5996283	2764,76	20283	28338	1,4	16
36		2001254	6003759	3097,23	20285	43074	2,12	24
37		2998443	8995326	3483,03	20892	41675	1,99	30
38		2999726	8999175	2979,59	20892	33784	1,62	24
39		3000512	9001533	3727,7	20893	47013	2,25	21
40		3999022	11997063	2489,36	21324	28455	1,33	33
41		4002619	12007854	4298,89	21325	37078	1,74	15
42		4003864	12011589	2794,92	21326	35860	1,68	18
43		4996620	14989857	2799,5	21658	44734	2,07	35
44		4998624	14995869	4595,52	21659	45379	2,1	27
45		4998876	14996625	4355,36	21659	32204	1,49	26

Tabla 8.9: Reporte de resolución de ALSPG en problemas problemas de tamaño grande.

	N^o de procesadores										
Prob N^o	2	3	4	5	6	7	8	9	10	11	12
1	76.17	53.58	59.37	50.78	47.31	53.85	57.03	47.4	75.02	**84.47**	66.2
2	66.01	72.58	66.38	46.75	43.65	46.93	49.02	53.34	83.29	**86.2**	75.32
3	71.36	57.35	54.73	44.65	50.14	50.77	49.74	63.93	69.72	**82.1**	74.8
4	72.99	70.71	62.04	55.03	47.5	47.75	55.38	65.01	80.74	**82.23**	61.09
5	75.72	63.58	53.63	56.79	49	43.29	58.38	57.98	79.02	**82.59**	63.06
6	79.1	71.13	65.39	53.64	40.04	52.38	50.57	56.56	**81.48**	76.36	62.76
7	73.75	59.15	59.61	48.97	44.95	51.61	47.3	50.4	**80.96**	79.3	71.97
8	67.93	71.93	54.83	58.52	52.78	47.05	42.04	59.27	**74.06**	71.48	72.65
9	74.03	56.95	67.93	49.92	39.71	50.66	60.98	64.25	80.07	**81.13**	75.73
10	77.98	66.36	57.62	62.32	35.37	48.6	43.08	57.33	**86.2**	75.71	63.61
11	**81**	65.45	66.65	46.82	52.98	40.55	45.56	64.71	72.85	79.93	76.55
12	71.41	60.29	66.55	61.31	44.85	43.78	42.77	49.93	**84.03**	79.41	68.58
13	62.01	69.34	65.73	52.24	48.29	43.12	58.82	60.31	82.11	**85.92**	64.51
14	**78.16**	65.09	59.07	58.63	44.81	55.88	58.89	62.77	73.7	67.61	69.34
15	68.77	70.46	68.92	55.22	40.56	53.76	46.14	59.67	74.26	**79.51**	64.74
16	73	53.3	55.39	47.03	51.82	37.5	54.34	60.67	**83.89**	71.53	69.6
17	75.26	65.18	66.15	43.31	49.6	49.07	46.05	47.53	73.87	**83.92**	76.03
18	66.22	54.38	49.13	52	54.89	41.34	42.72	47.17	**78.56**	73	68.33
19	72.58	66.27	59.4	54.03	40.38	47.72	57.78	65.29	75.59	**79.06**	75.41
20	66.78	67.93	50.4	59.72	41.13	46.5	57.82	62.91	74.42	**82.13**	76.36
21	69.58	55.21	66.55	56.73	39.39	39.18	48.62	62.65	**83.03**	74.7	70.76
22	**80.26**	60.2	64.86	56.9	47.85	44.34	41.63	52.01	74.49	70.1	65.77
23	**81.83**	59.33	64.04	57	51.03	53.08	53.1	51.84	68.48	70.43	61.2
24	69.65	63.39	58.43	50.04	41.9	50.45	58.76	48.56	**77.29**	71.19	71.9
25	67.48	54.27	52.69	61.86	43.11	40.72	44.96	63.92	**68.68**	68.21	68.45
26	**81.56**	56.43	65.74	48.19	42.13	55.12	50.12	63.24	71.48	79.43	58.97
27	77.13	56.78	52.04	54.33	39.64	56.75	48.69	55.23	**84.94**	67.95	71.72
28	**78.49**	53.33	65.61	53.8	45.29	43.59	41.63	65.43	77.56	68.46	72.92
29	76.7	59.16	62.32	48.88	35.94	50.04	45.06	64.63	**87.87**	70.66	60.16
30	62.88	54.42	56.53	61.57	46.2	52.22	55.25	50.08	**74.55**	67.79	74.49
31	69.71	54.52	51.59	62.77	40.59	39.36	59.41	58.63	80.26	**82.2**	69.7
32	74.74	62.92	50.54	43.64	36.35	50.34	48.44	56.58	71.98	**80.37**	60.03
33	**78.28**	67.11	68.11	55.54	38.68	48.51	43.32	50.33	71.45	73.73	69.46
34	68.14	61.48	49.64	59.94	45.21	41.46	59.98	60.45	68.59	**70.53**	62.2
35	75.01	72.1	53.12	60.48	41.23	45.39	54.38	47.94	**81.34**	75.97	59.2
36	66.82	55.42	55.33	58.11	45.35	50.99	51.87	66.11	72.96	67.28	**74.71**
37	69.68	55.98	56.33	58.73	41.72	45.14	49.12	50.08	74.2	**83.6**	58.32
38	72.99	68.85	58.98	51.28	46.74	45.73	43.76	62.13	**80.83**	76.8	60.73
39	72.34	70.05	63.54	45.47	36.24	54.67	60.14	53.63	**87.23**	77.37	64.86
40	77.08	62.98	55.84	46.72	42.7	50.9	56.49	49.49	69.86	**78.91**	62.55
41	76.91	57.53	68.84	54.3	41.36	55.87	51.95	50.67	79.25	**84.69**	64.12
42	72.91	56.05	65.67	62.7	48.38	42.01	60.53	62.77	**73.85**	73.85	68.82
43	65.46	53.55	55.65	58.91	40.48	37.9	60.36	50.84	71.42	**77.79**	57.59
44	63.36	56.95	61.74	51	44.7	39.24	54.73	48.61	77.9	**84.22**	67.01
45	80.24	61.83	66.79	53.37	44.4	42.3	58.84	55.91	70.36	**86.18**	60.17

Tabla 8.10: Eficiencias obtenidas sobre la paltaforma multicore. Se destaca en negrita la máxima eficiencia obtenida por problema.

8.3 Resolución de la superestructura MINLP

La superestructura MINLP surge en el modelado de numerosos casos industriales [Aggarwal & Floudas, 1990, Yiqing *et al.*, 2007, Kheawhom, 2010]. En este tipo de problemas el rendimiento computacional es fundamental dado que diversos problemas NLP deben ser visitados para alcanzar un óptimo en la superestructura. El algoritmo paralelo desarrollado e implementado aqui es utilizado para resolver cada uno de los escenarios que presentan las configuraciones de las variables binarias en una superestructura que contiene cada problema NLP. Los experimentos diseñados para esta sección nos sirven para realizar un estudio comparativo que nos permita determinar si es mejor aplicar los recursos de paralelismo con los que se cuentan, a nivel del resolvedor o a nivel de las variables binarias. Al mismo tiempo se analiza la importancia de de la factibilidad del punto inicial, es decir, si vale la pena invertir tiempo de cómputo en lograr un punto factible como punto inicial del optimizador. Este punto factible se obtiene resolviendo el problema

$$\begin{cases} \min\limits_{x} \|C(x)\| \\ \text{s. a. } l_i \leq x_i \leq u_i \end{cases} \tag{8.5}$$

Paralelismo a nivel del resolvedor del problema NLP

Cada problema no lineal que determina cada una de las configuraciones binarias que son exploradas es resuelto con la versión paralela del pALSPG (esquema I de la figura 8.13).

Paralelismo a nivel de las variables binarias

Se resuelven en simultáneo cada problema no lineal derivado de cada combinación de ceros y unos en las variables binarias (Esquema II de la figura 8.13).

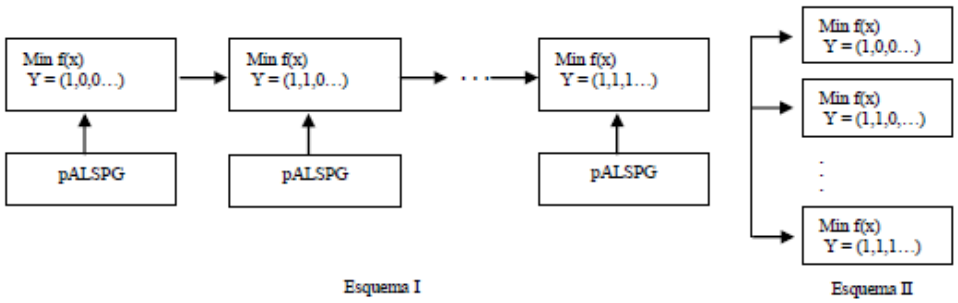


Figura 8.13: I - Paralelismo a nivel del resolvedor. II - Paralelismo a nivel de las variables binarias

8.4 Análisis de resultados

Los casos de estudio aquí analizados son cuatro problemas de escala y complejidad creciente:

- Problema 1: Separación no-estricta entre: Propano, Isobutano, n-Butano [Floudas & Pardalos, 1990].
- Problema 2: Separación de: Propano, Isobutano, n-Butano, Isopentano [Floudas & Pardalos, 1990].
- Problema 3: Planeamiento de un caso pequeño: con 3 variables binarias [Kocis & Grossmann, 1987].
- Problema 4: Optimización de una red de corrientes de agua: 5 variables binarias [Yiqing *et al.*, 2007].

La dimensión de estos problemas se reporta en la Tabla 8.11.

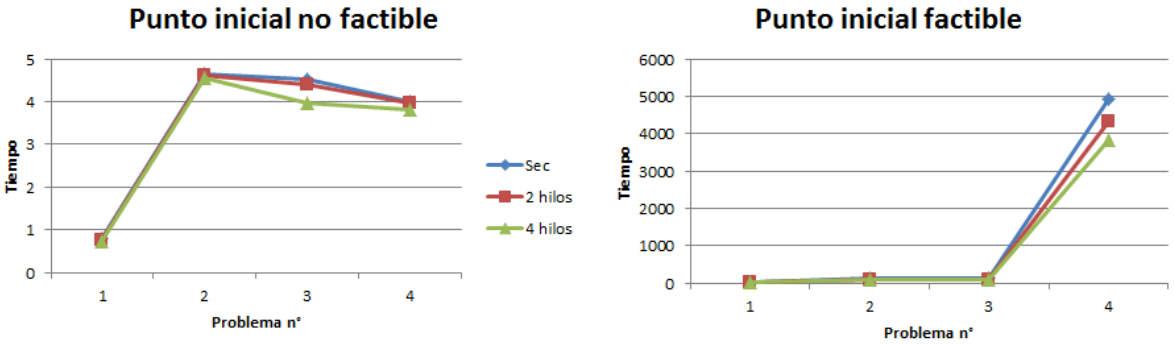
Para estos casos de estudio, se evaluó el desempeño de los algoritmos descriptos en las secciones 3.1 y 3.2. Ambos algoritmos fueron ejecutados en una computadora IBM con procesador AMD Phenom Quad Core Serie 9000 (Quad 9850), Memoria RAM 8GB, DDR3 1333MHZ, corriendo una versión secuencial, y otras dos versiones en paralelo con 2 y 4 hilos de ejecución respectivamente. Los tiempos de cálculo se reportan en las tablas

Problema n^o	1	2	3	4
Número de variables	38	50	89	24
Número de restricciones lineales	17	13	22	4
Número de restricciones no lineales	15	25	46	32
Número de variables binarias	1	2	3	5

Tabla 8.11: Características de los casos de estudio.

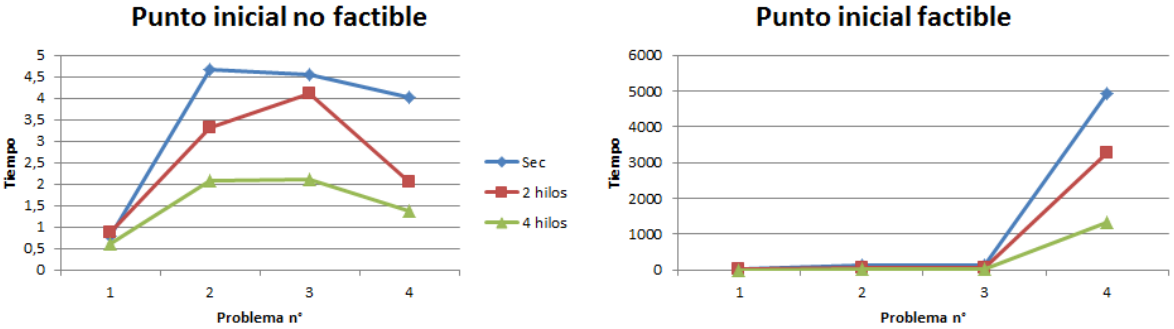
2 y 3. Prob. indica la referencia al número de problema, Sec. es el tiempo de ejecución de la versión secuencial, Efic 2 y Efic 4 indican la eficiencia de la versión paralela con 2 y 4 hilos de ejecución respectivamente. Al mismo tiempo, cada problema fue ejecutado dos veces: una iniciando el optimizador con un punto factible, y la otra con un punto no factible.

La eficiencia del paralelismo a nivel del resolvedor es creciente en proporción al número de variables del problema. Los bajos porcentajes de eficiencia en la tabla 2 están directamente relacionados con los tiempos de la resolución secuencial que son muy reducidos. El número de variables resulta ser bajo en relación al costo computacional que introduce la distribución de tareas en los diferentes núcleos, superando éste la ganancia obtenida a través del paralelismo. Esto se traslada a porcentajes más altos en el paralelismo a nivel de las variables binarias. Dado que cada proceso independiente resuelve todo el subproblema no lineal, el problema general es atacado desde el orden de complejidad que introducen las variables binarias. Si b es el número de variables binarias, el problema general tiene $2^b - 1$ subproblemas no lineales (dado que al menos un equipo está activo se excluye la combinación de todos ceros). Por otro lado el resolvedor muestra una gran sensibilidad respecto a la factibilidad del punto inicial. El algoritmo tiene un doble objetivo que debe mantener en equilibrio: disminuir la función objetivo y al mismo tiempo lograr la factibilidad. Obtener un punto factible antes de iniciar el optimizador y minimizar la función objetivo simplemente manteniendo la factibilidad produce una reducción notable en el tiempo de resolución (Ver tablas 8.14 y 8.15).



Punto inicial	Prob	Sec	2 hilos	4 hilos	Efic. 2	Efic. 4
Factible	1	0,78	0.77	0.72	50,64%	27,08%
	2	4,66	4,63	4,56	50,32%	25,54%
	3	4,54	4,41	3,98	51,47%	28,51%
	4	4,01	3,98	3,82	50,37%	26,24%
No factible	1	22,53	20,22	17,40	55,71%	32,37%
	2	119,56	112,34	109,3	53,21%	27,34%
	3	119,02	110,4	105,4	53,90%	28,23%
	4	4928,9	4342,5	3831,4	56,75%	32,16%

Figura 8.14: Reporte del algoritmo paralelo a nivel del resolvedor



Punto inicial	Prob	Sec	2 hilos	4 hilos	Efic. 2	Efic. 4
Factible	1	0,78	0,87	0,62	44,82%	31,45%
	2	4,66	3,32	2,07	70,18%	56,28%
	3	4,54	4,1	2,1	54,76%	53,82%
	4	4,01	2,05	1,39	97,80%	72,12%
No factible	1	22,53	15,18	8,34	73,82%	67,47%
	2	119,56	65,77	34,49	90,89%	86,64%
	3	119,02	70,51	36,97	84,39%	80,58%
	4	4928,9	3285,3	1345,6	75,01%	91,15%

Figura 8.15: Reporte del algoritmo paralelo a nivel de las variables binarias

Los resultados obtenidos muestran la conveniencia de introducir el paralelismo a nivel de las variables binarias dado que el algoritmo tiene muy buena respuesta para cada problema NLP. Al mismo tiempo, obtener un punto factible antes de iniciar el optimizador y minimizar la función objetivo simplemente manteniendo la factibilidad, produce una reducción notable en el tiempo de resolución. Este efecto da cuenta de que el tiempo utilizado para el cálculo del punto inicial factible deviene en una ganancia posterior en el algoritmo general, lo que convierte ese consumo inicial de tiempo de cómputo en una gran inversión.

Capítulo 9

Conclusiones y trabajo futuro

La demanda de desarrollos e investigación en técnicas ágiles para resolver problemas complejos en tiempos computacionales cortos es un tema en plena vigencia. Al mismo tiempo, el avance constante que se sigue produciendo en plataformas de computación paralela proporciona una poderosa alternativa para lograr este propósito, opción que es complementaria a otros enfoques tales como las reformulaciones de modelos. Se necesita mucho más que el último modelo de procesador, estación de trabajo o PC para hacer un desarrollo de software de alta calidad. Las herramientas de ingeniería de software asistida por computadora son más importantes que el hardware para lograr buena calidad y productividad; sin embargo, algunos autores sostienen que la mayoría de los desarrolladores de software todavía no las utilizan de manera efectiva.

Las fuertes bases teóricas de los enfoques basados en derivadas, sumado al desempeño exhibido por el método SPG en problemas específicos de ingeniería de procesos, nos llevaron a elegir este enfoque para la resolución de los problemas NLP provenientes de este área.

En este trabajo de tesis hemos desarrollado una nueva metodología llamada pALSPG, que es un enfoque paralelo del Método del Gradiente Espectral Proyectado combinado con una técnica clásica de Lagrangiano Aumentado. pALSPG demostró ser eficaz y competente en problemas relacionados con la optimización rigurosa de plantas de procesos industriales y con el diseño óptimo de columnas de destilación. Estas

características se evidenciaron a través de la comparación de desempeño con un conjunto de resolvedores provistos por la plataforma GAMS. Esta plataforma es una herramienta comercial orientada a ecuaciones para modelado muy utilizada en la actualidad para simulación y optimización de procesos.

pALSPG permite el uso eficiente de los recursos existentes no sólo para aumentar la velocidad de cómputo, sino también para dar cabida a problemas más grandes en un entorno de memoria distribuida. Los resultados obtenidos demuestran claramente la viabilidad de la utilización de este enfoque con el fin de resolver los problemas de secuencias óptimas de columnas de destilación. Es importante señalar que el crecimiento general que se sigue produciendo a nivel de equipos es una dorada oportunidad para renovar muchos algoritmos existentes con el fin de aprovechar la disponibilidad de una potencia de cálculo cada vez mayor. En cuanto a los datos numéricos, cabe destacar que pALSPG ha tenido un desempeño exitoso frente a problemas en los que los resolvedores de GAMS arrojaron resultados poco satisfactorios.

En resumen, la propuesta de pALSPG resultó ser muy importante ya que

- implementa una técnica robusta que todavía no se ha explotado para problemas difíciles de manejar,
- fue eficaz en algunos escenarios complejos, donde resolvedores clásicos ya probados no dieron respuestas satisfactorias respecto a la calidad de la solución pretendida,
- nos permite manejar un problema general sin necesidad de relajarlo,
- su diseño en paralelo es adaptable tanto a sistemas distribuidos como a arquitecturas de memoria compartida, y
- este diseño en paralelo es fácilmente escalable a plataformas con un número mayor de procesadores.

9.1 Trabajo futuro

A futuro se vislumbran interesantes perspectivas para continuar las investigaciones realizadas en esta tesis. Entre los temas que quedan abiertos a partir de este trabajo podemos mencionar:

- Estudiar la posibilidad de una implementación utilizando las GPU (Graphic Processing Units) como co-procesadores paralelos.
- Utilizar el paralelismo para desarrollar un resolvedor múltiple con la plataforma GAMS, que permita, entre otras cosas, aproximar un punto factible para inicialización de algoritmos de optimización.
- Estudiar la factibilidad de precondicionar los problemas para modificar su número de condición, característica del problema que tiene fuerte influencia en el desempeño de pALSPG.
- Considerar un posible rediseño del algoritmo paralelo para el aprovechamiento de los clusters multicore, arquitectura que está comenzando a tener un auge importante en la aplicación de la programación en paralelo.

Nomenclatura

$ALSPG$: lagrangiano aumentado con gradiente espectral proyectado

b_i: variables binarias

$C(x)$: conjunto de restricciones luego de introducir las variables de holgura ($C(x) = \{c_i(x)\} \cup \{c_j(x) - s_j\}$)

$c_i(x)$: restricciones de igualdad ($i = 1 \cdots n_i$)

$c_j(x)$: restricciones de desigualdad ($j = 1 \cdots n_j$)

d_k: dirección de búsqueda en la iteración k-ésima

E: eficiencia del algoritmo

F_{obj}: valor de la funci'on objetivo en el minimizador obtenido

g_k: gradiente evaluado en x_k

GC: condición de crecimiento ($L_{new} > \beta_k$)

k: contador de iteraciones

$L(x, \lambda, \rho)$: función Lagrangiano aumentado

L_{max}: máximo valor de función en los últimos M iterados

L_{new}: valor de función en x_{new}

l_i: cota inferior en la componente i de la variable x

M: entero que controla el grado de no monotonicidad

m: cota superior sobre la mejor proporción de performance

N_{pLS}: número de búsquedas lineales en paralelo

n_i: número de restricciones de igualdad

n_j: número de restricciones de desigualdad

N_S: número de búsquedas lineales

NC: número de columnas

NC_{LS}: número de llamadas al procedimiento de búsqueda lineal en paralelo

$Nproc$: número de procesadores en paralelo

$\mathcal{P}$: conjunto de problemas

$pALSPG$: lagrangiano aumentado con gradiente espectral proyectado en paralelo

$pSPG$: gradiente espectral proyectado en paralelo

$r_{p,s}$: proporción de desempe´ no del problema p ejecutado con el resolvedor s

$\mathcal{S}$: conjunto de resolvedores

S_p: speed-up

s_j: variables de holgura (slack)$(j = 1 \cdots n_j)$

SPG= gradiente espectral proyectado

T_{Tot}: tiempo total de búsquedas lineales

T^p_{Tot}: tiempo total de b'usquedas lineales en paralelo

T_C: tiempo de comunicación entre el Maestro y un trabajador

T_S: tiempo insumido en una búsqueda lineal

u_i: cota superior en la componente i de la variable x

x_i: componente i-th de la variable continua a optimizar

$x_\star$: solución obtenida por el algoritmo

x_{new}: posible nuevo iterado

x_0: punto inicial

x_k: valor actual de la variable de optimización

y_k: diferencia entre gradientes $(g_k - g_{k-1})$

α_{max}: cota superior para el parámetro α_k

α_{min}: cota inferior para el parámetro α_k

α_k: longitud del paso en la iteración k-ésima

β_k: cota superior para la condición de creciemiento $(L_{max} + \gamma * g_k^T d_k)$

δ_k: ángulo entre g_k y d_k $(cos\,\delta_k = \frac{|g_k^T d_k|}{\|g_k\|\|y_k\|})$

ϵ: tolerancia de optimalidad

η: tolerancia de factibilidad

γ: parámetro de decrecimiento suficiente en el procedimiento de búsqueda lineal

λ: estimación del vector de multiplicadores de Lagrange

Ω_0: conjunto de nivel $\{x : f(x) \leq f(x_0)\}$

ρ: parámetro de penalización de la función Lagrangiano aumentado

σ: factor de reducción de la longitud del paso

Apéndice A

Contribuciones científicas

- Domancich, A., Ardenghi, J.I., Vazquez, G.E. y Brignole, N.B. **2004**. Desempeño de un algoritmo espectral para optimización rigurosa de plantas de procesos industriales. Mecánica Computacional, **23**, 3047-3057.

- Ardenghi, J.I., Vazquez, G.E. y Brignole, N.B. **2007**. Un software paralelo para problemas de optimización de gran tamaño. Mecánica Computacional, **24**, 441-454.

- Ardenghi, J.I., Gibelli, T.I. and Maciel, M.C. **2009**. The spectral gradient method for unconstrained optimal control problems. IMA Journal of Numerical Analysis, **29**(2), 315-331.

- Ardenghi, J. I. y Brignole, N. B. **2013**. Optimización en Paralelo para Equipos de Procesos. En: Matemática Aplicada, Computacional e Industrial, vol. 4, 239-242.

- Ardenghi, J.I., Vazquez, G.E. y Brignole, N.B. **2013**. The Spectral Projected Gradient Method for Parallel Optimization. Applied Numerical Mathematics. Enviado. (En revisión)

- Ardenghi, J.I., Vazquez, G.E. y Brignole, N.B. **2014**. A Parallel Spectral-Projected-Gradient Method for Optimization in Process Engineering. Proceedings

of the 8th International Conference on Foundations of Computer-Aided Process Design, FOCAPD 2014 Elsevier.

Apéndice B

Aplicaciones industriales

En este apéndice se describen los casos industriales analizados en la sección 4

B.1 Producción de benceno por hidrodealquilación de tolueno (Problema HDA)

En este ejemplo se consideró un modelo de un proceso de hidroalquilación de tolueno para producir benceno. Las reacciones de interés son:

$$\text{Tolueno} + H_2 \Longrightarrow \text{Benceno} + CH_4$$
$$2\ \text{Benceno} \Longrightarrow \text{Difenilo} + H_2$$

Ambas reacciones se llevan a cabo a una presión de 500 psia, en un rango de temperaturas que va desde 1150F hasta 1300F. Estas cotas se imponen por cuestiones operativas ya que por debajo de 1150F la velocidad de reacción es muy lenta, mientras que por encima de 1300F se produce una significativa cantidad de hidrocracking. Asimismo se requiere un exceso de hidrógeno (en relación 5 a 1) para prevenir la formación de coque, y los gases efluentes del reactor deben ser rápidamente llevados a 1150F para que no ocurra el mismo efecto en el intercambiador de calor aguas abajo del reactor. En la figura B.1 se muestra un esquema simplificado del proceso. Las dos corrientes de alimentación, que contienen principalmente tolueno e hidrógeno se

calientan y mezclan con dos corrientes de reciclo, antes de ser alimentadas al reactor. La corriente de producto que abandona el reactor contiene hidrógeno (H_2), metano (CH_4), benceno, tolueno y difenilo. Luego, mediante un sistema de separación se logra obtener el benceno (producto principal) y el difenilo (producto secundario). Es importante resaltar que se debe realizar una purga de gases antes de que estos ingresen nuevamente al reactor como reciclo, ya que la acumulación de inertes (CH_4) complicaría el correcto funcionamiento del reactor.

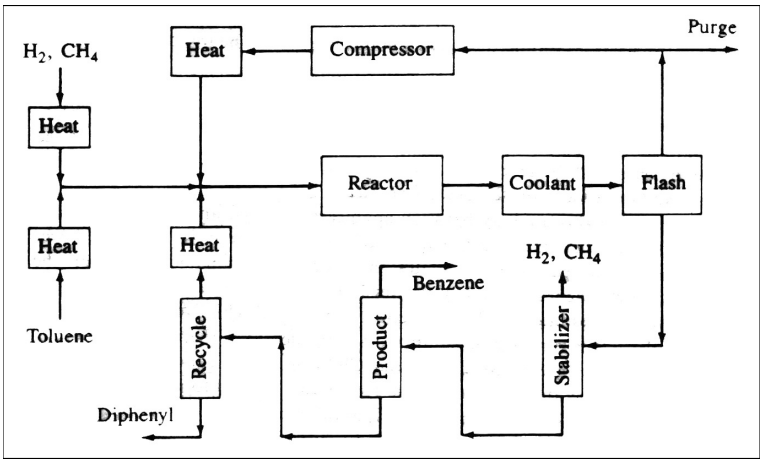


Figura B.1: Esquema simplificado de una planta de producción de benceno

Para este problema de optimización se fijó como función objetivo el potencial económico obtenido de la venta del benceno, el difenilo y los gases purgados (combustible). Las variables de optimización son la conversión que se obtiene en el reactor y la fracción de hidrógeno en la purga. El valor más conveniente para cada una de estas variables es calculado por el algoritmo de optimización. El sistema resultante es de 17 ecuaciones, las cuales provienen de realizar los balances de masa globales y por componentes en los mezcladores, el reactor y el separador (suponiendo separación perfecta). [Domancich *et al.*, 2004]

B.2 Red de intercambiadores de calor (Problema RED)

Este caso representa una red de intercambiadores de calor, en los que una corriente principal compuesta por metano, etano, propano y butano, es enfriada de 20F a Ű24F. Para lograr esto se utiliza una red de intercambiadores de calor con servicios externos, dividida en dos sectores en paralelo: uno que contiene un sólo intercambiador y otro que contiene dos. La alimentación principal se divide entonces en dos, para juntarse luego a la temperatura deseada. El diagrama del proceso se muestra en la figura B.2.

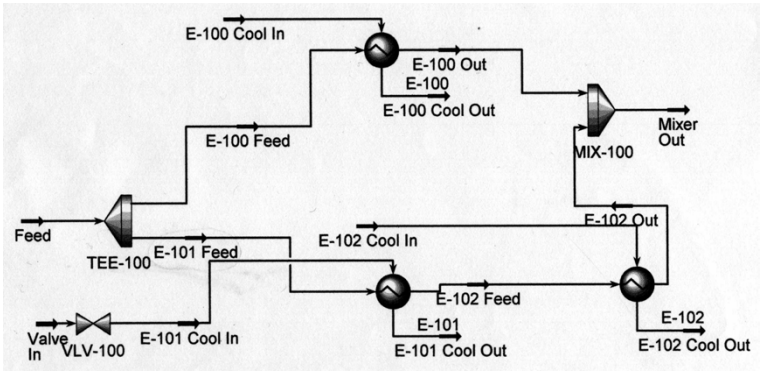


Figura B.2: Diagrama de una red de intercambiadores de calor

Este problema es relativamente más complejo que el anterior, ya que posee un número mayor de ecuaciones, las cuales resultan de realizar los balances de masa en los mezcladores y divisores, y los balances de masa y energía en cada uno de los tres intercambiadores de calor. El número total de ecuaciones del sistema es de 61. La función objetivo es la sumatoria de las áreas efectivas de transferencia de los equipos de intercambio de calor (UA), la cual debe ser minimizada para reducir al máximo los costos de instalación de los intercambiadores. La variable de optimización es el caudal denominado E-101 Feed en la figura B.2 que circula por la serie de dos intercambiadores, cuyo valor más conveniente se calcula mediante la optimización del proceso. [Domancich *et al.*, 2004]

B.3 Producción de cloruro de etilo (Problema CLE)

Uno de los caminos para producir cloruro de etilo (C_2H_5Cl) es mediante la siguiente reacción de cloruro de hidrógeno (HC_l) en fase gas con etileno (C_2H_4), sobre un catalizador de cobre en sílica:

$$C_2H_4 + HC_l \Longrightarrow C_2H_5C_l$$

En el proceso mostrado en la Figura B.3, la corriente de alimentación está compuesta por 50% molar de HCl, 48% molar de C2H4 y 2% molar de N2 circulando a razón de 100 kmol/hr. y 1 atm. de presión. Como la reacción alcanza solo un 90% de conversión, el cloruro de etilo producido es separado de los reactivos, los cuales son luego reciclados. La separación se logra mediante una columna de destilación, asumiendo que se logra alcanzar una separación perfecta. Para prevenir la acumulación de inertes en el sistema se realiza una purga de los gases que luego se reciclan al reactor. El efecto que produce la magnitud de la purga en el reciclo y en la composición en la alimentación al reactor es de especial interés.

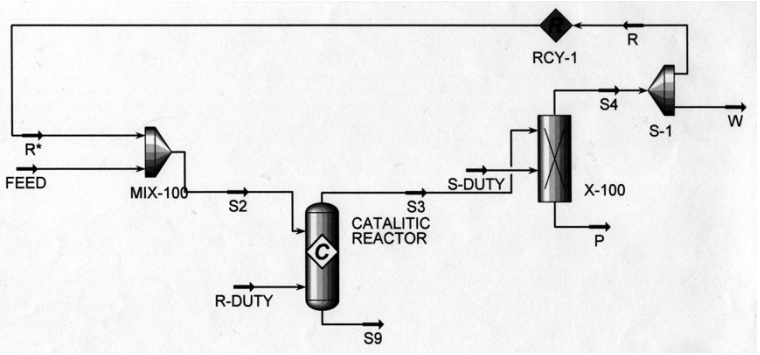


Figura B.3: Esquema simplificado de una planta de cloruro de etilo

El sistema resultante de plantear los balances de masa global y por componentes para cada uno de los equipos en estado estacionario es de 32 ecuaciones algebraicas. A pesar de poseer menos ecuaciones que el modelo del problema RED, este sistema es particularmente complicado de simular con HYSYS por el hecho de que contiene un

reciclo, lo que obliga al ingeniero a proveer al algoritmo de resolución algunos valores de puntos iniciales razonables, de manera tal de que este pueda llegar al resultado óptimo deseado. La función objetivo a optimizar es el potencial económico resultante de la venta del producto principal, teniendo en cuenta los costos de materia prima y de instalación del reactor (el cual varía de acuerdo al tamaño del equipo). La variable de optimización del proceso es el caudal de purga (W). [Domancich *et al.*, 2004]

Apéndice C

Tablas de resultados del capítulo 8

	N^o de procesadores							
Prob N^o	1	2	3	4	5	6	7	8
1	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
2	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
4	0.18	0.13	0.11	0.12	0.14	0.15	0.14	0.11
5	0.17	0.17	0.17	0.17	0.17	0.17	0.17	0.17
6	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
7	0.18	0.16	0.15	0.14	0.15	0.17	0.2	0.26
8	0.37	0.24	0.17	0.13	0.12	0.12	0.12	0.09
9	0.37	0.25	0.17	0.13	0.11	0.1	0.1	0.09
10	0.73	0.65	0.61	0.59	0.59	0.59	0.58	0.55
11	0.62	0.49	0.36	0.23	0.13	0.07	0.05	0.11
12	0.68	0.54	0.4	0.26	0.15	0.08	0.07	0.12
13	0.67	0.53	0.38	0.25	0.14	0.07	0.06	0.12
14	0.96	0.87	0.81	0.77	0.75	0.73	0.71	0.68
15	0.73	0.66	0.62	0.59	0.57	0.56	0.54	0.51
16	4.97	2.97	1.9	1.49	1.45	1.5	1.36	0.75
17	4.72	2.82	1.81	1.42	1.38	1.43	1.3	0.71
18	4.48	2.68	1.72	1.34	1.3	1.34	1.21	0.67
19	26.84	15.5	10.2	9.78	8.41	10.1	9.26	7.8
20	20.99	12.59	8.08	6.3	6.08	6.26	5.68	3.16
21	18.54	11.13	7.15	5.57	5.37	5.52	5.01	2.79
22	117.8	65.66	50.34	43.33	36.84	44.24	40.56	32.54
23	73.4	42.56	33.56	26.58	22.97	27.58	25.28	16.8
24	71.59	40.76	24.86	22.91	22.43	26.93	24.69	17.98
25	98.78	55.33	42.72	33.69	30.92	37.14	34.05	24.37
26	90.48	49.19	38.42	24.63	28.33	34.02	31.19	22.32
27	130.35	81.35	60.62	43.65	40.8	48.99	44.91	37.33
28	136.93	83.46	58.51	40.25	42.82	51.39	43.11	30.15
29	148.36	89.54	73.34	57.62	53.34	54.33	45.44	35.44
30	213.89	124.68	98.32	81.22	64.88	58.44	57.34	56.38

Tabla C.1: Tiempos (en segundos) obtenidos sobre un cluster

	N^o de procesadores						
Prob N^o	2	3	4	5	6	7	8
1	50	33.33	25	20	16.67	14.29	12.5
2	50	33.33	25	20	16.67	14.29	12.5
3	50	33.33	25	20	16.67	14.29	12.5
4	69.23	52.55	37.5	26.67	20.45	18.05	20.45
5	50	33.33	25	20	16.67	14.29	12.5
6	50	33.33	25	20	16.67	14.29	12.5
7	56.25	41.18	32.14	24.71	18.1	12.68	8.65
8	77.08	74.75	71.15	61.67	51.39	45.96	51.39
9	74	71.15	71.15	67.27	59.68	52.86	51.39
10	56.15	40.08	30.93	24.87	20.72	18.02	16.59
11	63.27	58.18	67.39	96.44	158.39	163.16	70.45
12	62.96	57.38	65.38	90.67	141.67	149.45	70.83
13	63.21	58.03	67	95.23	154.28	158.58	69.79
14	55.17	39.55	31.17	25.77	22.02	19.38	17.65
15	55.3	39.51	30.93	25.42	21.69	19.24	17.89
16	83.67	87.03	83.39	68.62	55.27	52.23	82.83
17	83.69	86.95	83.1	68.23	54.94	52	83.1
18	83.58	86.97	83.58	69.04	55.74	52.72	83.58
19	86.58	87.71	68.61	63.85	44.29	41.41	43.01
20	83.36	86.58	83.29	69.03	55.87	52.83	83.03
21	83.29	86.45	83.21	69.05	55.94	52.92	83.06
22	89.7	78	67.97	63.95	44.38	41.49	45.25
23	86.23	72.9	69.04	63.91	44.36	41.47	54.61
24	87.82	95.99	78.12	63.84	44.3	41.42	49.77
25	89.26	77.08	73.3	63.89	44.33	41.44	50.67
26	91.97	78.5	91.84	63.88	44.33	41.44	50.67
27	80.12	71.68	74.66	63.9	44.35	41.46	43.65
28	82.03	78.01	85.05	63.95	44.41	45.38	56.77
29	82.85	67.43	64.37	55.63	45.51	46.64	52.33
30	85.78	72.51	65.84	65.93	61	53.29	47.42

Tabla C.2: Eficiencias obtenidas sobre un cluster

	Nº de procesadores										
Prob Nº	2	3	4	5	6	7	8	9	10	11	12
1	5.66	5.37	3.63	3.4	3.04	2.29	1.89	2.02	1.15	0.93	1.09
2	6.8	4.12	3.38	3.84	3.43	2.73	2.29	1.87	1.08	0.95	0.99
3	5.28	4.38	3.44	3.38	2.51	2.12	1.89	1.31	1.08	0.83	0.84
4	7.34	5.05	4.32	3.89	3.76	3.2	2.42	1.83	1.33	1.18	1.46
5	11.09	8.8	7.83	5.91	5.71	5.54	3.59	3.22	2.12	1.85	2.22
6	7.26	5.38	4.39	4.28	4.78	3.13	2.84	2.26	1.41	1.37	1.53
7	12.98	10.79	8.03	7.82	7.1	5.3	5.06	4.22	2.36	2.19	2.22
8	12.78	8.04	7.92	5.93	5.48	5.27	5.16	3.25	2.34	2.21	1.99
9	10.39	9	5.66	6.16	6.46	4.34	3.15	2.66	1.92	1.72	1.69
10	3.08	2.42	2.09	1.54	2.27	1.41	1.4	0.93	0.56	0.58	0.63
11	21.92	18.09	13.32	15.17	11.17	12.51	9.74	6.1	4.87	4.04	3.87
12	29.53	23.32	15.84	13.76	15.67	13.76	12.32	9.38	5.02	4.83	5.12
13	19.62	11.7	9.25	9.31	8.4	8.06	5.17	4.48	2.96	2.57	3.14
14	15.05	12.04	9.95	8.02	8.75	6.01	4.99	4.16	3.19	3.16	2.83
15	26.67	17.35	13.31	13.29	15.07	9.75	9.94	6.83	4.94	4.19	4.72
16	71.96	65.7	47.42	44.68	33.79	40.02	24.17	19.24	12.52	13.35	12.58
17	78.92	60.75	44.89	54.86	39.92	34.58	32.24	27.77	16.08	12.87	13.02
18	125.75	102.08	84.74	64.05	50.57	57.55	48.73	39.23	21.2	20.74	20.31
19	125.79	91.85	76.85	67.59	75.37	54.66	39.5	31.08	24.16	21	20.18
20	181.31	118.83	120.12	81.1	98.13	74.4	52.35	42.77	32.54	26.8	26.43
21	224.81	188.88	117.52	110.29	132.37	114.07	80.43	55.48	37.68	38.07	36.84
22	142.99	127.09	88.47	80.67	79.94	73.95	68.92	49.03	30.81	29.77	29.08
23	222.93	204.98	142.43	128.01	119.16	98.19	85.89	78.2	53.28	47.09	49.68
24	104.42	76.49	62.24	58.14	57.86	41.19	30.94	33.28	18.82	18.58	16.86
25	352.02	291.81	225.42	153.6	183.67	166.67	132.09	82.58	69.17	63.32	57.84
26	124.22	119.69	77.06	84.1	80.16	52.52	50.54	35.6	28.35	23.19	28.63
27	132.45	119.94	98.15	75.21	85.9	51.43	52.45	41.1	24.05	27.33	23.74
28	343.09	336.63	205.22	200.22	198.2	176.51	161.72	91.46	69.44	71.52	61.55
29	323.4	279.52	199.01	202.98	230.05	141.63	137.62	85.29	56.46	63.83	68.72
30	131.29	101.13	73.02	53.63	59.56	45.17	37.36	36.63	22.15	22.14	18.47
31	1638.21	1396.42	1106.8	727.73	937.83	828.97	480.56	432.84	284.57	252.6	273.07
32	1580.19	1251.36	1168.42	1082.53	1083.02	670.32	609.53	463.86	328.16	267.18	327.9
33	1539.58	1197.22	884.73	867.98	1038.59	709.83	695.51	532.13	337.35	297.2	289.18
34	1980.55	1463.4	1359.34	900.6	995.02	930.02	562.5	496.11	393.51	347.9	361.62
35	1842.93	1278.21	1301.19	914.27	1117.62	870.16	635.52	640.79	339.9	330.84	389.18
36	2317.59	1862.88	1399.44	1065.99	1138.27	867.74	746.39	520.55	424.51	418.5	345.47
37	2499.3	2073.97	1545.81	1186.12	1391.43	1102.29	886.36	772.77	469.41	378.75	497.69
38	2041.09	1442.55	1262.97	1162.09	1062.47	930.8	851.12	532.86	368.62	352.7	408.86
39	2576.51	1773.83	1466.67	1639.63	1714.36	974.08	774.8	772.31	427.34	438	478.94
40	1614.79	1317.54	1114.51	1065.65	971.65	698.67	550.84	558.89	356.34	286.79	331.65
41	2794.75	2490.81	1561.19	1583.38	1732.31	1099.21	1034.38	942.68	542.45	461.46	558.7
42	1916.69	1662.16	1064	891.52	962.84	950.43	577.18	494.74	378.46	344.05	338.43
43	2138.33	1742.61	1257.64	950.43	1152.63	1055.22	579.75	611.83	391.98	327.16	405.09
44	3626.52	2689.8	1860.84	1802.16	1713.47	1673.04	1049.59	1050.43	589.93	496.05	571.5
45	2713.96	2348.03	1630.24	1632.14	1634.89	1470.91	925.25	865.55	619.01	459.44	603.2

Tabla C.3: Tiempos (en segundos) obtenidos sobre multicore

Lista de Figuras

Lista de algoritmos

Bibliografía

[Abadie & Carpentier, 1969] Abadie, J., & Carpentier, J. 1969. *Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints*. New York: Academic Press.

[Abbasi & Younis, 2007] Abbasi, A. A., & Younis, M. 2007. A survey on clustering algorithms for wireless sensor networks. *Journal Computer Communications*, **30**, 2826–2841.

[Abdel-Jabbar *et al.*, 1998] Abdel-Jabbar, N, Kravaris, C., & Carnahan, B. 1998. A partially decentralized State observer and its parallel computer implementation. *Ind. Eng. Chem. Res.*, **37**, 2741–2760.

[Aggarwal & Floudas, 1990] Aggarwal, A., & Floudas, C.A. 1990. Synthesis of General Separation Sequences - Nonsharp Separations. *Computers and Chemical Engineering*, **14**(6), 631–653.

[Alattas *et al.*, 2011] Alattas, A.M., Grossmann, I., & Palou-Rivera, I. 2011. Integration of nonliear crude distillation unit models in refinery planning optimization. *Ind. Eng. Chem. Res.*, **50**, 6860–6870.

[Alba, 2002] Alba, E. 2002. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 7–13.

[Alba, 2005] Alba, E. 2005. *Parallel Metaheuristics: a New Class of Algorithms*. John Wiley and Sons.

[Alba & Brignole, 2010] Alba, E., & Brignole, N.B. 2010. *Aplicaciones de Computación Científica*. Graduate course in Universidad Nacional del Sur, Bahía Blanca, Argentina.

[Ardenghi *et al.*, 2007] Ardenghi, J.I., Vazquez, G.E., & Brignole, N.B. 2007. Un software paralelo para problemas de optimización de gran tamaño. *Mecánica Computacional*, **24**, 441–454.

[Ardenghi *et al.*, 2009] Ardenghi, J.I., Gibelli, T.I., & Maciel, M.C. 2009. The spectral gradient method for unconstrained optimal control problems. *IMA Journal of Numerical Analysis*, **29**(2), 315–331.

[Barbosa & Doherty, 1987] Barbosa, D., & Doherty, M.F. 1987. A new set of composition variables for the representation of reactive phase diagrams. *Proceedings of the Royal Society London A*, **413**, 459–464.

[Barr & Hickman, 1993] Barr, R.S., & Hickman, B.L. 1993. Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Expert's Opinions. *ORSA Journal on Computing*, **5**(1), 2–18.

[Bartlett & Biegler, 2003] Bartlett, R. A., & Biegler, L. T. 2003. rSQP++ : An Object-Oriented Framework for Successive Quadratic Programming. *In: Lecture Notes in Computational Science and Engineering*. Large-scale PDE-Constrained Optimization. Springer Verlag, Berlin.

[Barzilai & Borwein, 1988] Barzilai, J., & Borwein, J.M. 1988. Two point step size gradient methods. *IMA Journal on Numerical Analysis*, **8**, 141–148.

[Bello & Raydan, 2004] Bello, L., & Raydan, M. 2004. *Convex-constrained optimization for the seismic reflection tomography problem*. Tech. rept. 2003-12. Centro de Cálculo Científico y Tecnológico, Facultad de Ciencias, Universidad Central de Venezuela, Caracas, Venezuela.

[Berman *et al.*, 2003] Berman, F., Fox, G. C., & Hey, A. J. G. 2003. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley.

[Bertsekas & Tsitsiklis, 1995] Bertsekas, D. P., & Tsitsiklis, J. N. 1995. *Nonlinear Programming*. Belmont, Massachussets: Athena Scientific.

[Bertsekas & Tsitsiklis, 1997] Bertsekas, D. P., & Tsitsiklis, J. N. 1997. *Parallel and Distributed Computations: Numerical Methods*. Prentice Hall.

[Betts, 2001] Betts, J. T. 2001. Practical methods for optimal control using nonlinear programming. *In: Advances in design and control*, vol. 3. Philadelphia, USA: SIAM.

[Biegler & Grossmann, 2004] Biegler, L., & Grossmann, I. 2004. Retrospective on Optimization. *Computers and Chemical Engineering*, **28**, 1169–1192.

[Biegler *et al.*, 2001] Biegler, L., Cervantes, A., & Waechter, A. 2001. *Advances in Simultaineous Strategies for Dynamic Optimization*. B-01-01. Carniege Mellon University, Department of Chemical Engineering.

[Biegler, 2010] Biegler, L.T. 2010. *Nonlinear Programming: Concepts, Algorithms and Applications to Chemical Engineering*. SIAM.

[Birgin & Evtushenko, 1998] Birgin, E., & Evtushenko, Y. 1998. Automatic diferentiation and spectral projected gradient methods for optimal control problems. *Optimization Methods and Software*, **10**, 125–146.

[Birgin & Martínez, 2002] Birgin, E., & Martínez, J.M. 2002. Large-scale active-set box-constrained optimization method with spectral projected gradients. *Computational Optimization and Applications*, **23**, 101–125.

[Birgin *et al.*, 2000] Birgin, E., Martínez, J.M., & Raydan, M. 2000. Nonmonotone spectral projected gradient methods on convex sets. *SIAM Journal on Optimization*, **10**(4), 1196–1211.

[Birgin *et al.*, 2001a] Birgin, E., Martínez, J.M., & Raydan, M. 2001a. Algorithm 813: SPG - Software for Convex-Constrained Optimization. *ACM Transactions on Mathematical Software (TOMS)*, **27**(3), 340–349.

[Birgin *et al.*, 2001b] Birgin, E., Martínez, J.M., & Raydan, M. 2001b. Algorithm 813: SPG - Software for Convex-Constrained Optimization. *ACM Transactions on Mathematical Software (TOMS)*, **27**(3), 340–349.

[Birgin *et al.*, 1999a] Birgin, E.G., Chambouleyron, I., & Martinez, J.M. 1999a. Estimation of the optical constants and thikness of thin films using unconstrained optimization. *Journal of Computational Physics*, **151**, 862–880.

[Birgin *et al.*, 1999b] Birgin, E.G., Biloti, R., Tygel, M., & Santos, L.T. 1999b. Restricted optimization: a clue to fast and accurate implementation of the common reflection surface method. *Journal of applied Geophysics*, **42**, 143–155.

[Birgin *et al.*, 2009] Birgin, E.G., Martínez, J.M., & Raydan, M. 2009. Spectral Projected Gradient Methods. *In:* Floudas, C. A., & Pardalos, P. M. (eds), *Encyclopedia of Optimization*, 2 edn. Springer.

[Birgin *et al.*, 2010] Birgin, E.G., Floudas, C.A., & Martínez, J.M. 2010. Global minimization using an Augmented Lagrangian method with variable lower-level constraints. *Math. Program.*, **125**, 139–162.

[Borchers & Mitchell, 1985] Borchers, B., & Mitchell, J.E. 1985. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, **31**(12), 1533–1546.

[Borraz-Sánchez, 2010] Borraz-Sánchez, C. 2010. *Optimization Methods for Pipeline Transportation of Natural Gas.* Ph.D. thesis, University of Bergen, Norway.

[Brezinski & Chehab, 1999] Brezinski, C., & Chehab, J.P. 1999. Multiparameter iterative schemes for the solution of systems of linear and nonlinear equations. *SIAM Journal on Scientific Computing*, **20**, 2140–2159.

[Brimberg *et al.*, 1998] Brimberg, J., Chen, R., & Chen, D. 1998. Accelerating Convergence in the Fermat-Weber Location Problem. *Operations Research Letters*, 151–157.

[Buzzi-Ferraris & Manenti, 2010] Buzzi-Ferraris, G., & Manenti, F. 2010. A Combination of Parallel Computing and Object-Oriented Programming to Improve Optimizer Robustness and Efficiency. *Computer Aided Process Engineering*, **28**, 337–342.

[Byrd *et al.*, 1999] Byrd, R. H., Hribar, M. E., & Nocedal, J. 1999. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization.*

[Byrd *et al.*, 1988] Byrd, R.H., Schnabel, R.B., & Schultz, G.A. 1988. Parallel quasi-Newton methods for unconstrained optimization. *Mathematical Programming*, **42**, 273–306.

[Carballido *et al.*, 2004] Carballido, J.A., Ponzoni, I., & Brignole, N.B. 2004. An Effective MOGA Initialization for Industrial-Plant Instrumentation Design Algorithms. *In: SBIA'04: XVII Brazilian Symposium on Artificial Intelligence.*

[Castillo *et al.*, 2000] Castillo, Z., Cores, D., & Raydan, M. 2000. Low cost optimization techniques for solving the nonlinear seismic reflection tomography problem. *Optimization and Engineering*, **1**, 155–169.

[Chai *et al.*, 2007] Chai, L., Gao, Q., & Panda, D.K. 2007. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. *Proceeding CCGRID*, 471–478.

[Cheimarios *et al.*, 2013] Cheimarios, N., G.Kokkoris, & Boudouvis, A.G. 2013. An effcient parallel iteration method for multiscale analysis of chemical vapor deposition processes. *Applied Numerical Mathematics*, **67**, 78–88.

[Chen *et al.*, 2011] Chen, Z., Chen, X., Yao, Z., & Shao, Z. 2011. GPU-Based parallel calculation method for molecular weight distribution of batch free radical polymerization. *Computer Aided Process Engineering*, **29**, 1583–1587.

[Coon & Stadtherr, 1995] Coon, A. B., & Stadtherr, M. A. 1995. Generalized block-tridiagonal matrix ordering for parallel computation in process flowsheeting. *Comput. Chem. Eng.*, **19**(6), 787–805.

[Cores *et al.*, 2000] Cores, D., Fung, G., & Michelena, R. 2000. A fast and globasl two point low storage optimization technique fro tracing rays in 2D and 3D isotropic media. *Journal of applied Geophysics*, **45**, 273–287.

[Dagum & Menon, 1998] Dagum, L., & Menon, R. 1998. OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, **5**, 46–55.

[Dai & Liao, 2002] Dai, Y. H., & Liao, L. Z. 2002. R-linear convergence of the Barzilai and Borwein gradient method. *IMA Journal on Numerical Analysis*, **22**, 1–10.

[D'Apuzzo & Marino, 2003] D'Apuzzo, M., & Marino, M. 2003. Parallel computational issues of an interior point method for solving large bound-constrained quadratic programming problems. *Parallel Computing*, **29**, 467–483.

[D'Apuzzo *et al.*, 2000] D'Apuzzo, M., Marino, M., Pardalos, P.M., & Toraldo, G. 2000. A parallel implementation of a potential reduction algorithm for box-constrained quadratic programming. *Lecture Notes in Computer Science-Euro-Par 2000*, 839–848.

[de Leone *et al.*, 1998] de Leone, R., Murli, A., Pardalos, P. M., & Toraldo, G. 1998. *High performance algorithms and software in nonlinear optimzation.* Kluwer academic.

[Dennis & Schnabel, 1983] Dennis, J.E., & Schnabel, R.B. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Systems.* Englewood Cliffs, New Jersey: Prentice-Hall.

[Diniz-Ehrhardt *et al.*, 2004] Diniz-Ehrhardt, M.A., Gomes-Ruggiero, M.A., Martinez, J.M., & Santos, S.A. 2004. Augmented lagrangian algorithms based on the spectral

projected gradient method for solving nonlinear programming problems. *Journal of Optimization Theory and Applications*, **123**, 497–517.

[Dolan & More, 2002] Dolan, E. D., & More, J. J. 2002. Benchmarking optimization software with performance profiles. *Math. Program.*, **Ser. A**(91), 201–213.

[Domancich *et al.*, 2004] Domancich, A. O., Ardenghi, J.I., Vazquez, G.E., & Brignole, N.B. 2004. Desempeño de un algoritmo espectral para optimización rigurosa de plantas de procesos industriales. *Mecánica Computacional*, **23**, 3047–3057.

[Domancich *et al.*, 2009] Domancich, A. O., Brignole, N.B., & Hoch, P.M. 2009. Structural Analysis of Reactive Distillation. *Virtual Pro*, **84**(18).

[Dorigo, 1992] Dorigo, M. 1992. *Optimization, Learning and Natural Algorithms*. Ph.D. thesis, Politecnico di Milano, Italy.

[Dowd & Severance, 2005] Dowd, K., & Severance, C. 2005. *High Performance Computing*. O' Reilly.

[Drud, 1985] Drud, A. 1985. CONOPT: A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems. *Mathematical Programming*, **31**(2), 153–191.

[Durazzi & V.Ruggiero, 2003] Durazzi, C., & V.Ruggiero. 2003. Numerical Solution of a Special Linear and Quadratic Program Via a Parallel Interior-Point Method. *Parallel Computing*, **29**, 485–503.

[Egea *et al.*, 2007] Egea, J. A., Vries, D., Alonso, A. A., & Banga, J. R. 2007. Glogal optimization for integrated design and control of computationally expensive process models. *Ind. Eng. Chem. Res.*, **46**, 9148–9157.

[Eggers *et al.*, 1997] Eggers, S., Emer, J., Levy, H., Lo, J., Stamm, R., & Tullsen, D. 1997. Simultaneous Multithreading: A Platform for Next-Generation Processors. *IEEE Micro*, 12–19.

[El-Rewini & Lewis, 1998] El-Rewini, H., & Lewis, T. G. 1998. *Distributed and Parallel Computing*. Manning.

[Feo & Resende, 1995] Feo, T.A., & Resende, M.G.C. 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, **6**, 109–133.

[Ferris & Munson, 1999] Ferris, M. C., & Munson, T. S. 1999. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 207–227.

[Ferris & Mangasarian, 1994] Ferris, M.C., & Mangasarian, O.L. 1994. Parallel Variable Distribution. *SIAM Journal of Optimization*, **4**, 815–832.

[Fletcher, 1987] Fletcher, R. 1987. *Practical methods of optimization*. New York: J. Wiley and Sons.

[Fletcher, 2001] Fletcher, R. 2001. *On the Barzilai and Borwein method*. Tech. rept. Numerical Analysis Report NA/207. Department of Mathematics, University of Dundee, Dundee, Scotland.

[Floudas & Pardalos, 1990] Floudas, C.A., & Pardalos, P.M. 1990. A Collection of Test Problems for Constrained Global Optimization Algorithms. *In: Lectures Notes in Computer Science*, vol. 455. Berlin: Springer-Verlag.

[Franco & Gómez, 2011] Franco, J.M., & Gómez, I. 2011. A family of explicit parallel Runge-Kutta-Nyström methods. *Applied Mathematics and Computation*, **218**, 4177–4191.

[Friedlander *et al.*, 1999] Friedlander, A., Martinez, J., Molina, B., & Raydan, M. 1999. Gradient method with retards and generalizations. *SIAM J. Numer. Anal.*, 275–289.

[GAMS, 2013] GAMS, Development Corporation. 2013. *GAMS, The Solver Manuals*. accessed July.

[Geist *et al.*, 1994] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., & Sunderam, V. 1994. *PVM: Parallel Virtual Machine*. MIT press.

[Geist *et al.*, 1996] Geist, A., Kohl, A., & Papadopolous, P. 1996. PVM and MPI: a comparison of features. *Calculateurs paralleles*, **8**(2).

[Ghattas & Orozco, 1997] Ghattas, O., & Orozco, C. E. 1997. *A parallel reduced hessian sqp method for shape optimization. Multidisciplinary Design Optimization: State of the Art*. SIAM.

[Gill *et al.*, 1998] Gill, P. E., Murray, W., Saunders, M. A., & Wright, M. H. 1998. *User's guide for NPSOL 5.0: A Fortran package for nonlinear programming*. Technical Report SOL 86-2. Systems Optimization Laboratory, Department of Operations Research, Stanford University.

[Gill *et al.*, 2002] Gill, P. E., Murray, W., & Saunders, M. A. 2002. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. Optim.*, 979–1006.

[Giusti *et al.*, 2005] Giusti, A. E. De, Naiouf, M. R., Giusti, L. C. De, & Chichizola, F. 2005. Dynamic Load Balancing in Parallel Processing on Non-Homogeneous Clusters. *Computer Science and Technology (JCST)*, **5**(4), 272–278.

[Glover & Laguna, 1997] Glover, F., & Laguna, M. 1997. *Tabu Search*. Kluwer Academic Publishers.

[Glunt *et al.*, 1991] Glunt, W., Hayden, T.L., & Liu, W. 1991. The embedding problem for predistance matrices. *Bulletin of Mathematical Biology*, **53**, 769–796.

[Glunt *et al.*, 1993a] Glunt, W., Hayden, T.L., & Raydan, M. 1993a. Molecular conformations from distance matrices. *J. Comput. Chem.*, **14**, 114–120.

[Glunt *et al.*, 1993b] Glunt, W., Hayden, T.L., & Raydan, M. 1993b. Preconditioners for distance matrix algorithms. *J. Comput. Chem.*, **15**, 227–232.

[Glunt *et al.*, 1994] Glunt, W., Hayden, T.L., C.Wells, Shelling, J.G., & Ward, D.J. 1994. Applications of weghting adn chirality strategies for distance geometry algorithms to an enterotoxin peptide analog. *Journal of Mathematical Chemistry*, **15**, 353–366.

[Golub & Van Loan, 1989] Golub, G.H., & Van Loan, Ch.F. 1989. *Matrix Computations*. Baltimore, Maryland: Johns Hopkins University Press.

[Grama *et al.*, 2003] Grama, A., Gupta, A., Karypis, G., & Kumar, V. 2003. *Introduction to Parallel Computing*. Addison Wesley.

[Grippo *et al.*, 1986] Grippo, L., Lampariello, F., & Lucidi, S. 1986. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numercial Analysis*, **23**(4), 707–716.

[Gropp & Lusk, 2002] Gropp, W., & Lusk, E. 2002. *Goals Guiding Design: PVM and MPI*. Tech. rept.

[Gupta *et al.*, 1997] Gupta, A., Karypis, G., & Kumar, V. 1997. A highly scalable parallel algorithm for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, **8**(5), 502–520.

[Heath, 1997] Heath, M.T. 1997. *Scientific Computing, An Introductory Survey*. McGraw-Hill.

[Hennessy *et al.*, 1999] Hennessy, J.L., Patterson, D. A., & Larus, J.R. 1999. *Computer organization and design: the hardware/software interface*. 2 edn. San Francisco: Kaufmann.

[Herrera, 2006] Herrera, F. 2006. *Introducción a los Algoritmos Metaheurísticos*. Tech. rept. Soft Computing and Intelligent Information Systems, Dpto. Ciencias de la Computación e I.A., Universidad de Granada.

[High & Laroche, 1995] High, K.A., & Laroche, R.D. 1995. Parallel Nonlinear Optimization techniques for Chemical Process Design Problems. *Computers Chem. Eng.*, **19**, 807–825.

[Himmelblau & Edgar, 1988] Himmelblau, D. M., & Edgar, T. F. 1988. *Optimization of Chemical Processes*. 1 edn. University of Texas: McGraw-Hill.

[Hock & Schittkowski, 1981] Hock, W., & Schittkowski, K. 1981. *Test examples for nonlinear programming codes*. Lecture Notes in Economics and Mathematical Systems. Berlin: Springer-Verlag.

[Holland, 1975] Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor. MI: University of Michigan Press.

[Hopfield & Tank, 1985] Hopfield, J.J., & Tank, D.W. 1985. Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, **52**, 141–152.

[INTEL, 2013] INTEL, Corporation. 2013. http://www.intel.com.

[Isard & Yu, 2009] Isard, M., & Yu, Y. 2009. Distributed data-parallel computing using a high-level programming language. *Proceeding SIGMOD '09 Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 987–994.

[Jager & Sachs, 1997] Jager, H., & Sachs, E.W. 1997. Global convergence of inexact reduced sqp methods. *Optimization methods and software*, **7**, 83–110.

[Khachiyan, 1979] Khachiyan, L.G. 1979. A Polynomial Algorithm in Linear Programming. *Soviet Math. Dokl.*, **20**, 191–194.

[Kheawhom, 2010] Kheawhom, S. 2010. Efficient constraint handling scheme for differential evolutionary algorithm in solving chemical engineering optimization problem. *Journal of Industrial and Engineering Chemistry*, **16**, 620–628.

[Kirkpatrick *et al.*, 1983] Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. 1983. Optimization by Simulated Annealing. *Science*, **220**(4598), 671–680.

[Kocis & Grossmann, 1987] Kocis, G., & Grossmann, I. 1987. Relaxation Strategy for the Structural Optimization of Process Flowsheets. *Ind. Eng. Chem. Res.*, **26**, 1869.

[Koelbel & Zosel, 1993] Koelbel, C.H., & Zosel, M.F. 1993. *The high performance fortran handbook*. MIT press.

[Kossack *et al.*, 2006] Kossack, S., Kraemer, K., & Marquardt, W. 2006. Efficient Optimization-Based Design of Distillation Columns for Homogenous Azeotropic Mixtures. *Ind. Eng. Chem. Res.*, **45**, 8492–8502.

[Kraemer *et al.*, 2009] Kraemer, K., Kossack, S., & Marquardt, W. 2009. Efficient Optimization-Based Design of Distillation Processes for Homogeneous Azeotropic Mixtures. *Ind. Eng. Chem. Res.*, **48**, 6749–6764.

[Kumar & Mazumder, 2010] Kumar, A., & Mazumder, S. 2010. Toward simulation of full-scale monolithic catalytic converters with complex heterogeneous chemistry. *Computers and Chemical Engineering*, **34**, 135–145.

[La-Cruz & Raydan, 2003] La-Cruz, W., & Raydan, M. 2003. Nonmonotone spectral methods for large-scale nonlinear systems. *Optimization Methods and software*, **18**, 583–599.

[Laird *et al.*, 2011] Laird, C. D., Wong, A. V., & Akesson, J. 2011. Parallel solution of large-scale dynamic optimization problems. *Computer Aided Process Engineering*, **29**, 813–817.

[Lee, 2006] Lee, E.A. 2006. *The Problem with Threads*. Tech. rept. UCB/EECS-2006. Electrical Engineering and Computer Sciences University of California, Berkeley.

[Lewis & Berg, 1997] Lewis, B., & Berg, D. J. 1997. *Multithreaded Programming With PThreads*. Prentice Hall.

[Lewis & Torczon, 2000] Lewis, R.M., & Torczon, V.J. 2000. Pattern search methods for linearly constrained minimization. *SIAM Journal of Optimization*, 917–941.

[Leyffer, 2001] Leyffer, S. 2001. Integrating SQP and branch and bound for mixed-integer noninear programming. *Computational Optimization and Applications*, **18**, 295–309.

[Lindo, n.d.] Lindo, Systems Inc. *LINDO*.

[Liu *et al.*, 2006] Liu, Ch., Woo, Ch., & Leung, D. Y.C. 2006. Performance analysis of a parallel finite element solution to the direct numerical simulation of fluid turbulence on Linux PC clusters. *Applied Mathematics and Computation*, **172**, 731–743.

[Luenberger, 1984] Luenberger, D.G. 1984. *Linear and Nonlinear programming*. Menlo Park, CA: Addison-Wesley.

[Luengo & Raydan, 2003] Luengo, F., & Raydan, M. 2003. Gradient Method with dynamical retards for large-scale optimization problems. *Electronic Transactions on Numerical Analysis.*, **16**, 186–193.

[Luengo *et al.*, 2002] Luengo, F., Glunt, W., Hayden, T.L., & Raydan, M. 2002. Preconditioned spectral gradient method. *Numerical Algorihms*, **30**, 241–258.

[Mahmoudi & Mazaheri, 2011] Mahmoudi, Y., & Mazaheri, K. 2011. High resolution numerical simulation of the structure of 2-D gaseous detonations. *Proceedings of the Combustion Institute*, **33**, 2187–2194.

[Mangasarian, 1995] Mangasarian, O.L. 1995. Parallel Gradient Distribution in Unconstrained Optimization. *SIAM Journal of on Control and Optimization*, **33**(6), 1916–1925.

[Marr *et al.*, 2002] Marr, D. T., Binns, F., Hill, D. L., Hinton, Glenn, Koufaty, D. A., Miller, A. J., & Upton, M. 2002. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, **6**(1).

[Migdalas *et al.*, 2003] Migdalas, A., Toraldo, G., & Kumar, V. 2003. Nonlinear Optimization and Parallel Computing. *Parallel Computing*, **29**, 375–391.

[Mittelmann, 1996] Mittelmann, H. 1996. Parallel Multispliting for Constrained Optimization. *Parallel Algor. Appl.*, **9**, 91–99.

[Mladenović & Hansen, 1997] Mladenović, N., & Hansen, P. 1997. Variable neighborhood search. *Computers Operations Research*, **24**(11), 1097–1100.

[Molina & Raydan, 1996] Molina, B., & Raydan, M. 1996. Preconditioned Barzilai-Borwein method for the numerical solution of partial differential equations. *Numerical Algorihms*, **13**, 45–60.

[Molina & Raydan, 2002] Molina, B., & Raydan, M. 2002. Spectral variant of Krylov subspaces methods. *Numerical Algorihms*, **29**, 197–208.

[Molina *et al.*, 2000] Molina, B., Petiton, S., & M, M. Raydan. 2000. An assessment of the preconditioned gradient method with retards for for parallel computers. *Comput. Appl. Math.*, **19**(3), 323–337.

[Moon *et al.*, 2008] Moon, J., Kulkarni, K., Zhang, L., & Linninger, A. A. 2008. Parallel Hybrid Algorithm for Process Flexibility Analysis. *Ind. Eng. Chem. Res.*, **47**, 8324–8336.

[MPI Forum, 1994] MPI Forum. 1994. MPI: A message-passing interface standard. *International journal of supercomputer applications*, **8**(3/4), 165–416.

[Mulato *et al.*, 2000] Mulato, M., Chambouleyron, I., Birgin, E.G., & Martinez, J.M. 2000. Determination of thikness and optical constants of a-Si:H films from transmittance data. *Applied Physics Letters*, **77**, 2133–2135.

[Murtagh & Saunders, 1987] Murtagh, B. A., & Saunders, M. A. 1987. *MINOS 5.1 User's Guide*. Tech. rept. SOL 83-20R. Department of Operations Research, Stanford University.

[Nagaiah & Kunisch, 2011] Nagaiah, C., & Kunisch, K. 2011. Higher order optimization and adaptive numerical solution for optimal control of monodomain equations in cardiac electrophysiology. *Applied Numerical Mathematics*, **61**(1), 53 – 65.

[Nocedal & Wright, 1999] Nocedal, J., & Wright, S. 1999. *Numerical Optimization*. Springer-Verlag.

[Openmp, 2008] Openmp. 2008. *OpenMP Architecture Review Board*. version 3.0 edn. application program interface.

[Pardalos & Vavasis, 1991] Pardalos, P. M., & Vavasis, S. A. 1991. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, **1**, 15–22.

[Pardalos & Schnitger, 1988] Pardalos, P.M., & Schnitger, G. 1988. Checking Local Optimality in Constrained Quadratic Programming is NP-hard. *Operations Research Letters*, 33–35.

[Pérez, 2004] Pérez, J. A. Moreno. 2004. *Metaheurísticas: Concepto y Propiedades*. Tech. rept. Departamento de Estadística, I.O. y Computación, Universidad de La Laguna.

[Ponzoni *et al.*, 1999] Ponzoni, I., Sánchez, M. C., & Brignole, N. B. 1999. A New Structural Algorithm for Observability Classification. *Ind. Eng. Chem. Res.*, **38**, 3027–3035.

[Powell, 1964] Powell, M.J.D. 1964. An Efficient Method for Finding the Minimum of a Function of Several Variables without Calculating Derivatives. *Computer J.*, **7**(6), 155–162.

[Quinn, 2003] Quinn, M. 2003. *Parallel Programming in C with MPI and OpenMP*. 1 edn. McGraw-Hill Science/Engineering/Math.

[Raydan, 1993] Raydan, M. 1993. On the Barzilai and Borwein choice of the steplength for the gradient method. *IMA Jounal Numerical Analysis*, **13**, 321–326.

[Raydan, 1997] Raydan, M. 1997. The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem. *SIAM Journal on Optimization*, **7**, 26–33.

[Raydan, 2002] Raydan, M. 2002. Preconditioned spectral gradient for solving nonlinear Poisson-type equations. *Pages 47–55 of: Proceedings CIMENICS 2002*, vol. M. Cerrolaza C. Muller-Karger, M. Lentini editor. Caracas, Venezuela: Editorial USB.

[Sahinidis, 1996] Sahinidis, N. V. 1996. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 201–205.

[Schnabel, 1995] Schnabel, R. B. 1995. A view of the limitations, opportunities and challenges in parallel nonlinear optimziation. *Parallel computing*, **21**(6), 875–905.

[Shang & He, 2010] Shang, Y., & He, Y. 2010. Parallel iterative finite element algorithms based on full domain partition for the stationary Navier Stokes equations. *Applied Numerical Mathematics*, **60**, 719–737.

[Tanenbaum, 1995] Tanenbaum, A. S. 1995. *Distributed Operating Systems*. Prentice-Hall.

[Taylor & Krishna, 2000] Taylor, R., & Krishna, R. 2000. Modelling reactive distillation. *Chemical Engineering Science*, **55**(22), 5183–5529.

[Tullsen *et al.*, 1998] Tullsen, D. M., Eggers, S.J., & Levy, H.M. 1998. Simultaneous multithreading: maximizing on-chip parallelism. *Pages 533–544 of: 25 years of the international symposia on Computer architecture (selected papers)*. ISCA '98. New York, NY, USA: ACM.

[Vanderbei & Shanno, 1997] Vanderbei, R. J., & Shanno, D. F. 1997. *An interior point algorithm for non-convex nonlinear programming.* Tech. rept. SOR-97-21. Princeton University.

[Vavasis, 1990] Vavasis, S. A. 1990. Quadratic programming is in NP. *Information Processing Letters*, 73–77.

[Vavasis, 1991] Vavasis, S. A. 1991. *Nonlinear Optimization: Complexity Issues.* New York: Oxford Science.

[Vazquez, 2002] Vazquez, G. E. 2002. *Procesamiento paralelo distribuido heterogéneo aplicado a ingeniería de procesos.* Ph.D. thesis, Universidad Nacional del Sur.

[Vazquez & Brignole, 1999] Vazquez, G. E., & Brignole, N. B. 1999. Parallel NLP Strategies using PVM on Heterogeneous Distributed Environments. *Lecture Notes in Computer Science: Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer Verlag*, **XVII**(1697), 533–540.

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

[Vygen, 2005] Vygen, Jens. 2005. *Approximation Algorithms for Facility Location Problems.* Tech. rept. Research Institute for Discrete Mathematics, University of Bonn.

[Wachter & Biegler, 2006] Wachter, A., & Biegler, L. T. 2006. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, **106**(1), 25–57.

[Wang *et al.*, 2007] Wang, K., Shao, Z., Zhang, Z., Chen, Z., Fang, X., Zhou, Z., Chen, X., & Qian, J. 2007. Convergence Depth Control for Process System Optimization. *Ind. Eng. Chem. Res.*, **46**, 7729–7738.

[Wells *et al.*, 1994] Wells, C., Glunt, W., & Hayden, T.L. 1994. Searching conformational space with the spectral distance geometry algorithm. *Journal of molecular structure (Theochem)*, **308**, 263–271.

[Womble *et al.*, 1999] Womble, D.E., Dosanja, S.S., Hendrickson, B., Heroux, M.A., Plimpton, S.J., Tomkins, J.L., & Greenberg, D.S. 1999. Massively Parallel Computing: A Sandia Perspective. *Parallel Computing*, **25**, 1853–1876.

[Yiqing *et al.*, 2007] Yiqing, L., Xigang, Y., & Yongjian, L. 2007. An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints. *Computers and Chemical Engineering*, 153–162.

[You *et al.*, 2011] You, F., Pinto, J.M., Capona, E., Grossmann, I.E., Arora, N., & Megan, L. 2011. Optimal Distribution-Inventory Planning of Industrial Gases. I. Fast Computational Strategies for Large-Scale Problems. *Ind. Eng. Chem. Res.*, **50**, 2910–2927.

[Zaharia *et al.*, 2010] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. 2010. Spark: cluster computing with working sets. *Pages 10–10 of: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. Proceeding HotCloud'10.