Optimización Multi-Objetivo y Computación Grid

Antonio J. Nebro, Enrique Alba y Francisco Luna

Resumen— **Este trabajo analiza algunos aspectos técnicos y prácticos relativos al uso de sistemas paralelos para resolver problemas de optimización multi-objetivo mediante búsqueda enumerativa. Esta técnica constituye una estrategia de exploración conceptualmente simple, basada en la evaluación de cada posible solución dado un espacio de búsqueda finito. La obtención de resultados mediante enumeración para problemas complejos es inviable para la mayoría de las plataformas de computación actuales. No obstante, estos resultados son de gran interés ya que se pueden utilizar para ser comparados con aquellos obtenidos mediante técnicas heurísticas. Nuestro estudio consiste en analizar el uso de un sistema de computación Grid que subsane en cierta medida los límites impuestos por la búsqueda enumerativa. Una vez evaluado el rendimiento del algoritmo secuencial, presentamos, primero, un algoritmo paralelo diseñado para sistemas multiprocesador. En segundo lugar, hemos desarrollado una versión distribuida preparada para ejecutar sobre una federación de computadoras geográficamente distribuidas, lo que se conoce como red computacional (*computational grid*). La conclusión alcanzada es que este tipo de sistemas puede proporcionar a la comunidad científica un gran conjunto de frentes de Pareto muy precisos que, de otra forma, no serían conocidos.**

Palabras clave— **Optimización multi-objetivo, Búsqueda enumerativa, Computación paralela, Computación Grid**

I. Introducción

UN problema de optimización multi-objetivo se puede definir como el problema de encontrar un vector de variables de decisión que satisfaga ciertas restricciones y optimice un vector de funciones cuyos elementos representan las funciones objetivo [1]. Estas funciones forman una descripción matemática de criterios de rendimiento que están normalmente en conflicto entre ellos. Por tanto, el término "optimización" se refiere a la búsqueda de una solución tal que contenga valores aceptables para todas las funciones objetivo.

En general, la optimización multi-objetivo no se restringe a la búsqueda de una única solución, sino de un conjunto de soluciones llamadas *soluciones no-dominadas*. Cada solución de este conjunto se dice que es un *óptimo de Pareto* y, al representarlas en el espacio de los valores de las funciones objetivo, conforman lo que se conoce como *frente de Pareto*. Dado un problema concreto, la obtención del frente de Pareto es la principal finalidad de la optimización multi-objetivo.

Las diversas técnicas existentes para obtener el frente de Pareto se pueden clasificar en 3 categorías: enumerativas, deterministas y estocásticas [2]. En los últimos años, los métodos estocásticos han sido ampliamente estudiados; en particular, un gran número de autores han trabajado en la investigación de algoritmos evolutivos [3], [4], [5]. Estos métodos no garantizan la obtención de la solución óptima, pero ofrecen soluciones aceptables para un amplio rango de problemas de optimización en los que los métodos deterministas encuentran dificultades. La búsqueda enumerativa, que es un método determinista en el que no se emplea heurística alguna, constituye una estrategia conceptualmente simple de optimización, y está basada en la evaluación de cada posible solución dado un espacio de búsqueda finito. El inconveniente de esta técnica es su inherente ineficiencia, ya que puede ser computacionalmente costosa e incluso prohibitiva a medida que el espacio de búsqueda crece.

A pesar de sus desventajas, los resultados que se obtienen mediante enumeración son de gran interés para la comunidad de investigación en optimización multi-objetivo, ya que pueden ser utilizados para compararlos con aquellos obtenidos mediante el uso de algoritmos heurísticos. En consecuencia, podemos medir de forma objetiva la calidad de estas soluciones. Por tanto, se pueden alcanzar importantes avances científicos con la comparación y evaluación de nuevos algoritmos, algo que contrasta con la situación actual en la que se realizan análisis no estandarizados.

En este trabajo se aborda, en primer lugar, la resolución de un conjunto de problemas de optimización multi-objetivo sobre un sistema monoprocesador utilizando una estrategia enumerativa. La aplicación que se encarga de ello está escrita en C++ y ha sido diseñada específicamente para facilitar la inclusión de nuevos problemas. En segundo lugar, e intentando mejorar la eficiencia de la búsqueda enumerativa, se han estudiado distintos enfoques paralelos del proble-

Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga. E.T.S. Ingeniería Informática. Campus de Teatinos, 29071. E-mail: {antonio,eat,flv}@lcc.uma.es.

ma. El análisis se ha centrado principalmente en el multiprocesamiento y el uso de redes computacionales (*computational grids*) [6], [7], [8]. Las redes computacionales son sistemas distribuidos compuestos, potencialmente, por miles de ordenadores que aprovechan la infraestructura creada por Internet para proporcionar un enorme supercomputador virtual. Estos sistemas permiten, por tanto, abordar problemas considerados como intratables hasta hace unos años. Finalmente, hemos desarrollado un algoritmo enumerativo distribuido ejecutado sobre Condor [9]. Con este algoritmo, se han resuelto algunos problemas de optimización multi-objetivo que se podrían considerar como intratables para un sistema monoprocesador.

Las contribuciones de este trabajo se pueden resumir como sigue:

- La resolución de un conjunto de problemas multi-objetivo sobre un sistema monoprocesador equipado con una CPU moderna ha permitido determinar los tiempos que han servido como base para nuestro estudio. A partir de aquí hemos determinado la complejidad de cada problema cuando se resuelve tanto en un sistema secuencial como paralelo.
- Hemos estudiado distintos enfoques, desde el punto de vista paralelo, que se pueden aplicar a la resolución mediante enumeración de problemas complejos. Con estas implementaciones paralelas estamos mejorando el tiempo de respuesta de la búsqueda enumerativa. Además, el uso de algoritmos enumerativos distribuidos basados en redes computacionales nos muestra una nueva perspectiva sobre los límites existentes en la actualidad a la hora de determinar qué problemas pueden resolverse y cuáles no.
- La comunidad investigadora puede aprovechar tanto el software desarrollado como los resultados obtenidos (ambos están disponibles en `http://neo.lcc.uma.es/Software/ESaM`). Entre estos resultados se incluyen tanto el conjunto de óptimos de Pareto como los valores de las variables de decisión del problema. Así, cabe la posibilidad de crear una base de datos de libre disposición de problemas multi-objetivo y sus correspondientes soluciones que sirva como base para futuras comparaciones con otras técnicas.

El resto del trabajo está organizado como sigue. En la siguiente sección se presentan los trabajos relacionados con la aplicación de técnicas de computación paralela a la optimización de problemas multi-objetivo. La Sección III incluye una descripción del algoritmo secuencial, así como su rendimiento al resolver un banco de pruebas de problemas multi-objetivo. En la Sección IV se describen las implementaciones paralelas del algoritmo. Finalmente, las conclusiones y el trabajo futuro aparecen en la Sección V.

II. Computación Paralela y Optimización Multi-Objetivo

Los sistemas de computación paralelos han sido utilizados ampliamente en el campo de la optimización mono-objetivo [10], [11], [12]. En el caso de las técnicas deterministas, un ejemplo típico consiste en la resolución de problemas de optimización mediante algoritmos paralelos de Ramificación y Poda [13]. Se trata, en general, de resolver problemas en menos tiempo o bien resolver problemas más complejos. En el contexto de los métodos heurísticos, el paralelismo no sólo significa resolver los problemas de forma más rápida, sino que además se obtienen modelos de búsqueda más eficientes: un algoritmo heurístico paralelo puede ser más efectivo que uno secuencial, aún ejecutándose en un solo procesador. Sirvan como ejemplo los estudios sobre algoritmos evolutivos presentados en [14], [15].

Las ventajas que ofrece la programación paralela a la optimización mono-objetivo también se mantienen en la optimización multi-objetivo. Algunos estudios referentes a técnicas evolutivas son [2], [16], [17]. Sin embargo, como se afirma en [2], los trabajos sobre implementaciones paralelas han sido escasos en este campo.

En general, la mayoría de los trabajos sobre computación paralela y optimización están centrados en dos tipos de sistemas paralelos: los sistemas multiprocesadores de memoria compartida y los sistemas distribuidos, estos últimos basados en clusters o redes de área local. En el primer caso, los servicios ofrecidos por el sistema operativo son suficientes para desarrollar programas paralelos: se pueden usar lenguajes secuenciales con bibliotecas de hebras o llamadas al sistema para escribir aplicaciones multi-hebra o multi-proceso que utilicen los distintos procesadores del sistema. Otra opción consiste en usar lenguajes paralelos como Java.

En el caso de los sistemas distribuidos, existe una cantidad enorme de aspectos a considerar, como la red de conexión (Ethernet, Myrinet, ATM), las distintas topologías (anillo, árbol, hipercubo, malla), los modelos de programación (paso de mensajes, RPC, memoria compartida distribuida), las bibliotecas de comunicación (MPI, PVM, sockets), lenguajes paralelos (Java, C#), middleware (CORBA, DCOM, RMI) e incluso tecnologías de Internet (XML, SOAP, Servicios Web).

La última generación de sistemas distribuidos está basada precisamente en la popularidad de Internet y la disponibilidad de una gran cantidad de recursos computacionales que están repartidos geográficamente. Estos recursos pueden agruparse para ofrecer un recurso computacional único y unificado. Este proceso ha desembocado en lo que se denomina como *Grid computing* [8], aunque también se usan términos como *metacomputing*, *Internet computing* o *Web computing*.

La idea de utilizar cientos o miles de procesadores es muy atractiva, ya que puede permitir la resolución de problemas de optimización considerados como intratables. Por ejemplo, en [18], se resuelve, en una semana, una instancia del problema de asignación cuadrática (QAP, *Quadratic Assignment Problem*) que habría necesitado 7 años de tiempo de cómputo en una estación de trabajo, utilizando para ello una red computacional con unos 2500 procesadores. El problema se resolvió con un algoritmo distribuido de Ramificación y Poda. Con excepción de este trabajo, las referencias relacionadas con optimización y computación Grid que se encuentran son escasas [19], [20], [21]. En lo que a nuestro conocimiento se refiere, no existe trabajo alguno relacionado con optimización multi-objetivo y computación Grid; por tanto, este trabajo supone la primera tentativa en este sentido.

Respecto a trabajos relacionados con búsqueda enumerativa y paralelismo en el contexto de la optimización multi-objectivo, en Coello et al. [2] (pág. 144-145) se detalla una implementación distribuida usando MPI. Nuestro trabajo supone un avance respecto a éste en el sentido de que el uso de MPI impide sacar partido a las redes computacionales, lo que limita las posibilidades de resolver problemas de elevada complejidad.

III. Búsqueda Enumerativa Secuencial

En esta sección presentamos el algoritmo enumerativo secuencial que ha sido diseñado para este trabajo, que constituye, además, la base de las implementaciones paralelas.

El algoritmo es similar al descrito en [22] (pág. 36-38). Las variables de decisión, que se asumen continuas, se discretizan con cierta granularidad y, para cada combinación de valores de las variables, las funciones objetivo son evaluadas; los vectores resultantes son comparados entre sí mediante un test de dominancia de Pareto, obteniendo así un conjunto de soluciones no-dominadas. Obviamente, a medida que la granularidad se hace más fina, mayor es el coste computacional del algoritmo. Además, si el problema de optimización tiene restricciones, es necesario determinar si éstas se satisfacen antes de realizar la evaluación y los test de dominancia. La Fig. 1 contiene el pseudocódigo del algoritmo.

```
F[M] = {F1, F2, ..., FM} // Funciones Objetivo
R[C] = {R1, R2, ..., RC} // Restricciones
x[N] = {x1, x2, ..., xN} // Vbles. de decisión
f[M] = {f1, f2, ..., fM} // Valores de función
P = ∅ // Conjunto de soluciones no-dominadas

Fijar la granularidad G de las vbles. de decisión
Para cada vector x[i]
  Si x[i] satisface las restricciones R[C]
    f[j] = evaluación de x[i] usando F[M]
    Comprobar la dominancia de f[j] en P
    Si f[j] es una solución no-dominada
      Añadir f[j] a P
    Eliminar de P las soluciones dominadas por f[j]
```

Fig. 1. Pseudocódigo del algoritmo secuencial.

El número de iteraciones que ejecuta el algoritmo depende, por un lado, del número de variables de decisión N y, por otro, de la granularidad deseada G, ya que el tamaño del espacio de búsqueda es G^N. Sin embargo, la complejidad del algoritmo está bastante influenciada por el chequeo de restricciones y la evaluación de las funciones objetivo. Consideremos, además, que el paso de evaluación sólo se realiza cuando se satisfacen las restricciones, por lo que cuanto más restrictivas sean éstas, menor será el número de evaluaciones a computar.

El algoritmo, implementado en C++, ha sido diseñado con dos objetivos: primero, la inclusión de nuevos problemas ha de ser una tarea fácil y, segundo, ha de servir como base para las implementaciones paralelas. El primer objetivo se logra con el mecanismo de herencia; cada clase que representa a un problema ha de heredar de una clase padre, de forma que todas tienen la misma estructura. Para conseguir el segundo objetivo, la clase principal del programa incluye, además del problema, parámetros adicionales que permiten explorar la totalidad del espacio de búsqueda, o bien sólo una parte del mismo. El primer caso se utiliza en el algoritmo secuencial, mientras que el segundo se aplica en las versiones paralelas. En ambos casos, el programa calcula el frente de Pareto (o un subfrente, si el espacio de búsqueda no es totalmente explorado) del problema multi-objetivo. Cuando la computación acaba, los valores de las variables de decisión y de las funciones objetivo se escriben en sendos ficheros.

A. Banco de Pruebas de Problemas Multi-Objetivo

Para medir el rendimiento de la implementación secuencial hemos seleccionado diversos problemas multi-objetivo procedentes de la literatura especializada; en concreto, los problemas han

TABLA I

Resultados de los problemas con el algoritmos enumerativo secuencial.

Problema	Var.	Func.	Rest.	Tiempo (100)	Puntos (100)	Tiempo (1000)	Puntos (1000)
Poloni	2	2	0	< 1	75	126	1102
Kursawe	2	3	0	8	42	118074	874
Deb	2	2	0	< 1	28	27	262
Viennet4	3	2	3	< 1	447	609	39664
Tanaka	2	2	2	< 1	15	1	152
Osyczka2	6	2	6	90.2 días	570	N/A	N/A
Golinski	7	2	11	>600 días	N/A	N/A	N/A

sido elegidos del libro de Coello et al. [2]. Los nombres de cada uno de ellos siguen la terminología empleada en dicho libro.

Los problemas seleccionados son: Poloni, Kursawe, Deb, Viennet4, Tanaka y Osyczka2. El trabajo también incluye el problema de Golinski [23]. Este banco de pruebas comprende una serie de problemas con y sin restricciones que son ampliamente conocidos y han servido de base para comparar un gran número de algoritmos.

B. Resultados

Hemos utilizado un PC con un procesador Pentium 4 a 2.4 GHz y 512 MB de RAM. El sistema operativo es SuSE Linux 8.1 (Kernel 2.4.20). La compilación de los programas se ha realizado con la herramienta `gcc v.3.2`, incluyendo la opción `-O3`. La Tabla I incluye los nombres de los problemas, así como sus principales características: número de funciones objetivo, número de variables de decisión y número de restricciones. Las cuatro últimas columnas contienen los tiempos de resolución de los problemas y el número de puntos del frente cuando usamos 100 y 1000 particiones por variable, respectivamente.

Si no se tienen en cuenta los resultados de los problemas Osyzcka2 y Golinski, para los que los tiempos sólo se pueden estimar debido a su dificultad, en la Tabla I se puede observar que el tiempo necesario para resolver los problemas con 100 particiones por variable es de escasos segundos. Pero cuando abordamos su solución con 1000 particiones, algunos requieren varias horas de cómputo, siendo el problema de Kursawe el más lento (tarda unas 33 horas). A la vista de estos resultados, podemos concluir que todos pueden resolverse en un tiempo razonable con un único procesador.

Los frentes de Pareto obtenidos para cada problema aparecen en la Fig. 3 (Apéndice A). Es importante observar la diferencia entre los frentes de un mismo problema cuando se considera un mayor número de particiones por variable. De hecho, con una mayor precisión en la búsqueda (1000 particiones), los frentes contienen un gran número de puntos (ver Tabla I). Por tanto, se genera una mayor cantidad de información que puede ser utilizada para realizar análisis objetivos y precisos de algoritmos multi-objetivo heurísticos.

Pasemos ahora a examinar los resultados de los dos últimos problemas, que son los más complejos de todos los seleccionados ya que tienen un gran número de variables de decisión y de restricciones. Como consecuencia, la cantidad de tiempo que consumen es considerable, incluso con 100 particiones por variable. En la Tabla I aparecen los tiempos de CPU estimados para resolverlos (en la Subseción IV-B se mostraran más detalles sobre estas estimaciones). Aunque son estimaciones, estos tiempos dan una idea de la complejidad que supone la resolución de estos problemas utilizando un sistema monoprocesador.

Estos resultados monoprocesador justifican claramente el interés en el trabajo sobre, primero, el uso de sistemas paralelos para resolver problemas complejos y, segundo, el estudio de heurísticas eficientes cuando la búsqueda enumerativa no es viable.

IV. Aplicación de Técnicas Paralelas

Actualmente existen dos tipos principales de arquitecturas paralelas: los multiprocesadores de memoria compartida y los sistemas distribuidos. En esta sección se detallan dos implementaciones paralelas del algoritmo de búsqueda enumerativa presentado en la sección anterior, una por cada tipo de sistema paralelo.

A. Implementación para Multiprocesadores de Memoria Compartida

El algoritmo paralelo que se considera está basado en la ejecución de varios procesos en paralelo, cada uno de los cuales ejecuta el algoritmo secuencial, pero explorando partes distintas del espacio de búsqueda. Todo proceso, al acabar, escribe dos ficheros con los resultados obtenidos (como se explicó en la Sección III). Cuando todos finalizan, un nuevo proceso lee esos ficheros y combina su contenido aplicando tests de dominancia, para obtener así el frente de Pareto. Este

algoritmo paralelo es simple ya que no es necesaria una comunicación entre procesos y existe sólo un punto de sincronización.

Para implementar el algoritmo en C++ tenemos dos opciones: hebras y procesos. En los siguientes apartados se describen cada una de ellas.

A.1 Hebras

La utilización de hebras frente a procesos tiene, a priori, dos ventajas. En primer lugar, la creación y la gestión de hebras es menos costosa que la de los procesos; y, en segundo, la comunicación entre hebras es más simple y eficaz que entre procesos.

Con el uso de las hebras, el algoritmo paralelo sería aún más simple, ya que los ficheros que almacenan los resultados no serían necesarios; en su lugar, los subfrentes se pueden guardar en la memoria compartida, donde se puede acceder directamente para obtener el frente de Pareto.

A.2 Procesos

Pese a que la gestión de hebras es menos costosa y su uso es más simple, se ha optado por implementar el algoritmo paralelo mediante procesos por dos motivos: primero, una implementación basada en hebras no supone beneficios substanciales debido a los escasos requisitos de comunicación y sincronización del algoritmo propuesto; en segundo lugar, la implementación multiproceso se puede utilizar sin cambio alguno en un entorno distribuido, tal y como se presentará en la subsección siguiente.

Para evaluar su eficiencia, nuestra implementación multiproceso se ha ejecutado en un multiprocesador Sun Ultra Enterprise 450, con 4 procesadores UltraSPARC II a 450 MHz y 4 GB de memoria RAM. Su sistema operativo es Solaris 2.8 y hemos usado el compilador `g++ v.2.95`. Como ejemplo se ha resuelto el problema de Kursawe (3 variables de decisión) con 200 particiones por variable. El programa secuencial tarda 783 segundos, mientras que el paralelo necesita 196, lo que permite obtener una ganancia de velocidad casi perfecta de 3.99.

La conclusión a la que se llega con estos resultados es que las implementaciones paralelas basadas en memoria compartida son simples y proporcionan un buen rendimiento. Sin embargo, para la resolución de problemas complejos se necesitan sistemas con un mayor número de procesadores, y este número está limitado incluso para los supercomputadores. Por lo tanto, si es necesaria la utilización de cientos de procesadores, la mejor elección no es el uso de un multiprocesador, sino de un sistema distribuido.

B. Implementación Distribuida

Tal y como se expuso en la Sección II, existe un gran número de aspectos que han de ser considerados en el desarrollo de aplicaciones orientadas a sistemas distribuidos. Sin embargo, muchos de ellos fallan cuando se intenta utilizar una gran cantidad de máquinas que pertenecen a propietarios u organizaciones distintas, ya que no satisfacen adecuadamente algunos de los requisitos de este tipo de sistemas. Entre estos requisitos se pueden considerar la heterogeneidad, el uso poco intensivo de las comunicaciones (el ancho de banda y la latencia están limitadas, ya que se conectan máquinas que pueden estar a grandes distancias), la falta de fiabilidad y la disponibilidad dinámica de máquinas. A modo de ejemplo, las bibliotecas de comunicación más conocidas como PVM o MPI no satisfacen adecuadamente los dos últimos requisitos, y bastantes programas diseñados para usarlas fallan con el segundo. La computación Grid está dedicada a tratar con estos y otros aspectos relacionados, siendo un área de investigación muy activa en los últimos años [6]. Así, sistemas como Globus [24], Legion [25] o Condor [9] se están utilizando para ejecutar programas sobre redes computacionales compuestas por miles de máquinas.

B.1 Implementación con Condor

La implementación de nuestro algoritmo distribuido de búsqueda enumerativa ha utilizado Condor. En comparación con otras plataformas de computación Grid, es fácil de instalar y administrar y, además, los programas ya existentes no necesitan ser re-compilados para su ejecución en Condor (sólo han de ser re-enlazados con sus bibliotecas). Condor está diseñado para gestionar colecciones distribuidas de procesadores que están repartidos por distintos campus u otro tipo de organizaciones [9], donde cada máquina tiene un propietario. Una de las características de Condor es que permite al propietario de cada máquina especificar las condiciones bajo las que los trabajos de Condor pasan a ejecutarse; por defecto, un trabajo que ejecuta Condor se detiene cuando el propietario de la máquina comienza a utilizarla. Por tanto, estos trabajos utilizan aquellos ciclos de procesador que, de otra forma, serían desperdiciados. Así se anima a los usuarios para que no sean reacios a donar sus máquinas para poder construir colecciones enormes de máquinas.

La utilización de Condor para desarrollar la versión distribuida de nuestro algoritmo enumerativo ha sido una tarea fácil, ya que se han reutilizado los mismos programas de la versión multiproceso propuesta para multiprocesadores de me-

moria compartida (razón para no usar hebras). Lo único que se ha tenido que hacer es obtener un ejecutable por cada plataforma que compone nuestro sistema distribuido, además de escribir un fichero de configuración. Este fichero contiene, entre otra información, el nombre del ejecutable, el número de instancias que se crearán y los parámetros que se le pasarán a cada una de ellas. Después, simplemente hay que trasladar a Condor los trabajos especificados en el fichero de configuración. Tal y como ocurría en el caso del código multiproceso, cuando todas las tareas acaban, se procede a la recolección de información almacenada en los ficheros de resultados para obtener el frente de Pareto.

B.2 Resultados

Nuestro sistema Condor está compuesto por PCs, estaciones de trabajo y servidores de varios laboratorios del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga. Las máquinas utilizadas son las siguientes:

- Un cluster de 4 Digital AlphaServer 4100, cada una con 4 procesadores Alpha a 300 MHz y con 256 MB de memoria. El sistema operativo instalado es Digital Unix 4.0D.
- Un cluster de 22 estaciones de trabajo Sun Ultra 1. Los procesadores que llevan son UltraSPARC II a 400 MHz, con 256 MB de RAM. Ejecutan Solaris 2.8.
- Dos servidores Sun Ultra Enterprise, equipados con 4 procesadores UltraSPARC II a 450 MHz y 4 GB de RAM. Su sistema operativo también es Solaris 2.8.
- Un cluster de 16 PCs, cada uno con un procesador Pentium 4 a 2.4 GHz y 512 MB de memoria. El sistema operativo que tienen instalado es SuSE Linux 8.1 (Kernel 2.4.20).
- Un cluster con 22 PCs que llevan un procesador AMD Athlon XP a 1.2 GHz y 256 MB de RAM. Su sistema operativo es Debian Linux 3.0 (Kernel 2.2).
- Un cluster con 20 PCs, cada uno con un procesador Pentium III a 600 MHz y 128 MB de memoria. También llevan instalado SuSE 8.1 (Kernel 2.4.20).
- Otros 6 PCs que ejecutan distintas versiones de Linux.

En total hemos podido utilizar más de 110 procesadores: 16 Alpha, 30 UltraSPARC, 16 Pentium 4, 26 Pentium III y 22 Athlon.

Este sistema Condor se ha utilizado para resolver los problemas Osyzcka2 y Golinski con 100 particiones por variable. Comencemos con los resultados obtenidos para el problema Osyzcka2 (su frente de Pareto se muestra en la Fig. 2). Este problema se ha resuelto en menos de dos días reales, mientras que el tiempo total de CPU que Condor ha necesitado para resolver la tarea ha sido de 90.25 días. Dado que el sistema está compuesto por máquinas con capacidades de cómputo distintas, el tiempo que requeriría el algoritmo secuencial con el procesador más rápido sería menor que esta cantidad de tiempo, aunque en cualquier caso no bajaría de varias decenas de días. Durante la redacción de este documento, el problema de Golinski se está resolviendo en nuestro sistema Condor. Estimamos que el cómputo se completará en unos 10 días, si bien el tiempo total de CPU que proporcionará Condor será del orden de los 600 días (en estos momentos ha completado la mitad de las tareas y dicho tiempo es superior a los 300 días).

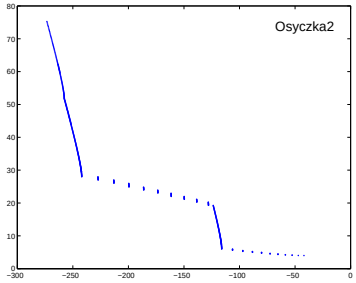

Fig. 2. Frente del problema Osyzcka2.

Estos resultados muestran los beneficios de la utilización de tecnologías Grid para resolver problemas cuya computación no es viable es sistemas monoprocesador. Nuestro sistema distribuido está compuesto por un número pequeño de máquinas, sin embargo, Condor permite combinar sistemas de organizaciones diferentes. Por tanto, la construcción de un computador Grid con miles de procesadores es bastante factible actualmente. No obstante, incluso utilizando un sistema con estas características, la resolución de problemas como Osyzcka2 y Golinski con 1000 particiones por variable seguiría siendo intratable. En este sentido, la utilización de técnicas heurísticas pasa a ser la única opción posible.

V. Conclusiones y Trabajo Futuro

En este trabajo hemos desarrollado dos versiones paralelas de un algoritmo de búsqueda enumerativa que resuelve problemas de optimización multi-objetivo: una destinada para su uso en sistemas multiprocesador de memoria compartida y otra diseñada para ejecutarse con Condor, un sistema que permite crear redes computacionales. El principal objetivo ha sido analizar cómo estos sistemas paralelos se pueden usar para obtener

por enumeración el frente de Pareto de problemas en los que utilizar un sistema monoprocesador no sería viable.

Los experimentos con la versión multiproceso muestran que se pueden obtener ganancias lineales al usar multiprocesadores, algo esperado puesto que los procesos son independientes y no interaccionan entre ellos.

Nuestras experiencias utilizando Condor para ejecutar un algoritmo enumerativo distribuido en una red de más de 100 procesadores ponen de manifiesto que es posible resolver problemas en pocos días que de otra forma habrían necesitado cientos de ellos para resolverse en un sistema monoprocesador.

Se puede concluir, entonces, que la comunidad investigadora en optimización multi-objetivo puede aprovechar las redes computacionales para resolver problemas complejos y obtener frentes de Pareto reales, permitiendo así la utilización de métricas justas y significativas, como la distancia al frente de Pareto calculado con un computador Grid.

Como planes de futuro, intentaremos aplicar las experiencias obtenidas en este trabajo a la paralelización de técnicas heurísticas para optimización multi-objetivo en el contexto de las tecnologías Grid. Además, se pretende desarrollar una nueva versión de la aplicación paralela en la que exista interacción entre particiones, ya que el algoritmo actual trabaja con procesos independientes.

Agradecimientos

Este trabajo ha sido realizado con el apoyo del Ministerio de Ciencia y Tecnología (MCYT) y el Fondo Europeo de Desarrollo Regional (FEDER) mediante los contratos TIC2002-04498-C05-02 (el proyecto TRACER) y TIC2002-04309-C02-02.

Referencias

[1] A. Osyczka, *Multicriteria Optimization for Engineering Design*, Academic Press, 1985.

[2] C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.

[3] C.A. Coello, "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," *Knowledge and Information Systems. An International Journal*, vol. 1, no. 3, pp. 269–308, 1999.

[4] C. M. Fonseca and P. J. Flemming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.

[5] D.A. Van Veldhuizen and G.B. Lamont, "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art," *Evolutionary Computation*, vol. 8, no. 2, pp. 125–147, 2000.

[6] M. Baker, R. Buyya, and D. Laforenza, "Grids and Grid Technologies for Wide Area Distributed Computing," *Software: Practice and Experience*, vol. 32, pp. 1437–1466, 2002.

[7] F. Berman, G.C. Fox, and A.J.G. Hey, *Grid Computing. Making the Global Infrastructure a Reality*, Communications Networking and Distributed Systems. Wiley, 2003.

[8] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.

[9] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. John Wiley & Sons Inc., December 2002.

[10] A. Grama and V. Kumar, "State of the Art in Parallel Search Techniques for Discrete Optimization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 28–35, 1999.

[11] UEA CALMA Group, "CALMA Project Report 2.4: Parallelism in Combinatorial Optimisation," Tech. Rep., School of Information Systems, University of East Anglia, Norwich, UK, September 18 1995.

[12] A. Migdalas, P. M. Pardalos, and S. Story, *Parallel Computing in Optimization (Applied Optimization, Vol. 7)*, Kluwer Academic Publishers, 1997.

[13] B. Gendron and T. G. Crainic, "Parallel Branch and Bound Algorithms: Survey and Synthesis," *Operations Research*, vol. 42, 1994.

[14] E. Alba and M. Tomassini, "Parallelism and Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[15] E. Alba and J.M. Troya, "A Survey of Parallel Distributed Genetic Algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.

[16] K. Deb, P. Zope, and A. Jain, "Distributed Computing of Pareto-Optimal Solutions Using Multi-Objective Evolutionary Algorithms," Tech. Rep. 2002008, KanGAL, September 2002.

[17] D.A. Van Veldhuizen, J.B. Zydallis, and G.B. Lamont, "Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 87, no. 2, pp. 144 –173, April 2003.

[18] K. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth, "Solving Large Quadratic Assignment Problems on Computational Grids," *Mathematical Programming*, vol. 91, pp. 563–588, 2002.

[19] M.O. Neary and P. Cappello, "Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing," in *Proc. ACM Java Grande/ISCOPE Conference*, November 2002, pp. 56–65.

[20] Y. Tanimura, T. Hiroyasu, M. Miki, and K. Aoi, "The System for Evolutionary Computing on the Computational Grid," in *Parallel and Distributed Computing and Systems (PDCS 2002)*, November 2002.

[21] S.J. Wright, "Solving Optimization Problems on Computational Grids," Tech. Rep., Mathematics and Computer Science Division, Argonne National Latoratory, Argonne, Ill., November 2000.

[22] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, 2001.

[23] A. Kurpati, S. Azarm, and J. Wu, "Constraint Handling Improvements for Multi-Objective Genetic Algorithms," *Structural and Multidisciplinary Optimization*, vol. 23, no. 3, pp. 204–213, April 2002.

[24] I. Foster and C. Kesselman, "Globus: a Metacomputing Infraestructure Toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[25] A. Grimshaw and W. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40, no. 1, 1997.

Appendix

I. Frentes de Pareto Obtenidos

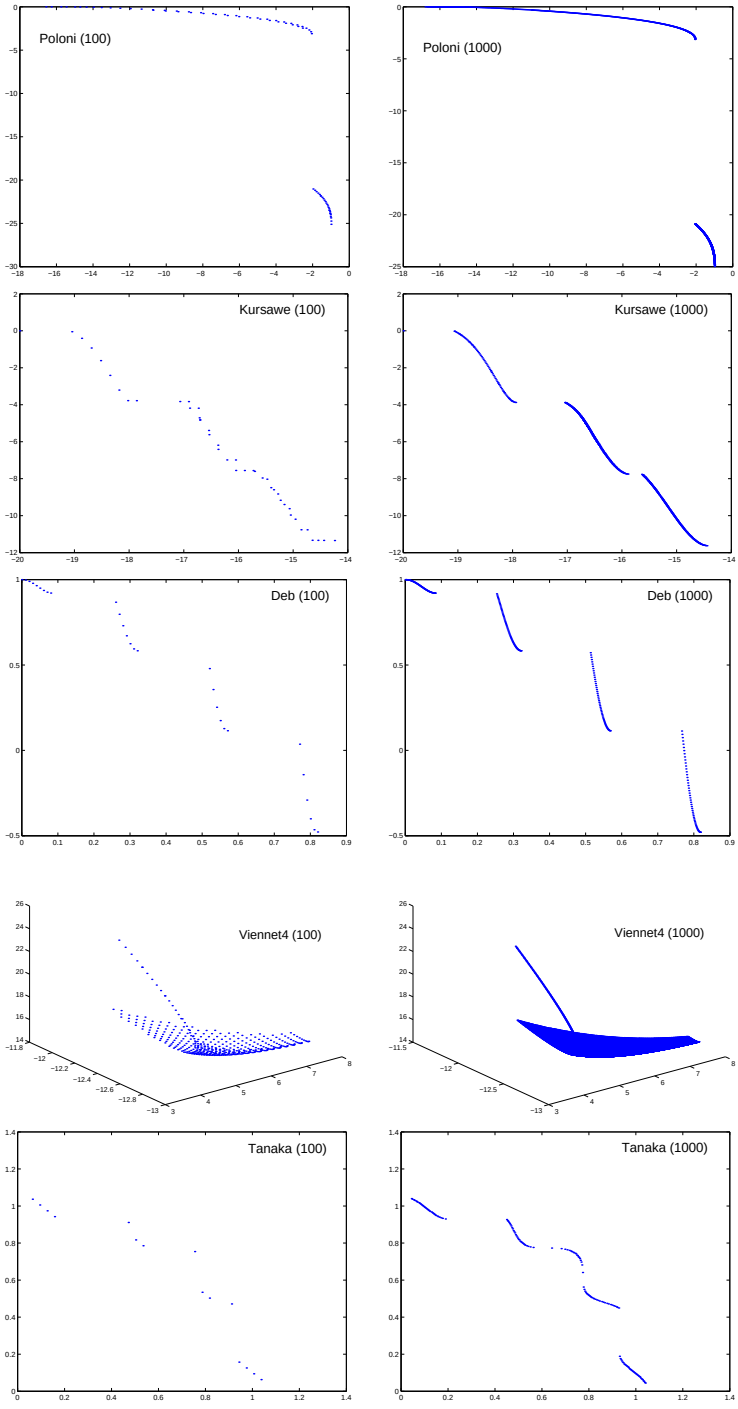

Fig. 3. Frentes de Pareto de los problemas resueltos (el número entre paréntesis indica la precisión).