matemática: Una publicación del ICM – ESPOL
2011, Vol. 9, No. 2

SCATTER SEARCH Y LA OPTIMIZACIÓN FUNCIONAL

Martín Carlos[1]

Resumen. El objetivo principal de este documento es el de mostrar cómo trabaja la metaheurística "scatter search", en cada una de sus etapas, cuando encara problemas de optimización. Para ello, el problema a resolver con "scatter search" es el de la optimización de funciones de variable real. Además, para la implementación de "scatter search" se ha construido un software con el lenguaje de programación C#.NET.

Palabras Clave: scatter search, metaheurística, heurística, algoritmo evolutivo.

Abstract. The main objective of this paper is to show how it works metaheuristic called "scatter search", in each of its stages, when faced with optimization problems. For this, the problem to be solved with "scatter search" is the optimization of functions of real variable. Also, for the implementation of "scatter search" has built a software with the programming language C #. NET

Key Words: scatter search, metaheuristic, heuristic, evolutionary algorithm

Recibido: Agosto 2011
Aceptado: Septiembre 2011

1. INTRODUCCIÓN

Existen muchas metaheurísticas, entre ellas: el algoritmo genético, la búsqueda tabú, el recocido simulado, GRASP, entre otras, que pueden utilizarse, junto con heurísticas, para resolver problemas de optimización. Scatter search es también un método metaheurístico que se puede emplear para resolver problemas de optimización y forma parte de la familia de los algoritmos evolutivos. Tiene sus orígenes en los años setenta, pero es en la última década cuando ha sido probado en muchos problemas con un alto grado de dificultad y con excelentes resultados [1], [2].
El objetivo del presente documento entonces es el de explicar cada una de las fases de "scatter search", es decir, indicar en detalle cómo trabaja internamente el algoritmo. Para tal efecto, se ha construido un software que optimiza funciones continuas de una variable real sobre un conjunto compacto utilizando la metaheurística de "scatter search".

2. EL ALGORITMO Y EL SISTEMA

El problema de optimización que se pretende resolver es el de maximizar o minimizar una función objetivo en general no lineal con regla de correspondencia $f(x)$ donde la única restricción es que la variable independiente x pertenezca a un intervalo cerrado $[a,b]$. El algoritmo trabaja con cualquier función, sin embargo, el software sólo permite escoger una función de entre diez, las cuales detallo a continuación:

$[1]\ f(x) = x\ sen(10\,\pi\,x) + 1$
$[2]\ f(x) = 1/x$
$[3]\ f(x) = x\ sen(1/x)$
$[4]\ f(x) = x^4 + x^3 - 3x^2 + 1$
$[5]\ f(x) = \dfrac{6}{x^2+3}$
$[6]\ f(x) = (x^2-4)^3$
$[7]\ f(x) = x^3 + e^{-x}$
$[8]\ f(x) = 3 - |1 + 2x|$
$[9]\ f(x) = |\ln(x-2)| + 2$
$[10]\ f(x) = x^2 - 5x + 6$

Los valores de a y de b que determinan el intervalo cerrado son fijados por el usuario del software, junto con otros parámetros adicionales (se habla de ellos en la sección 2.1).
Este documento tiene como objetivo mostrar la metodología y la filosofía de trabajo del scatter search. Se podrá apreciar que la mezcla de calidad y diversidad que utiliza el algoritmo es fascinante.
A continuación se procede a explicar cómo configurar el sistema para poder utilizarlo y cómo ingresar adecuadamente los parámetros para que trabaje correctamente el algoritmo de scatter search.

2.1 CONFIGURACIÓN Y PARÁMETROS DEL SISTEMA

Para que el software funcione correctamente el usuario del sistema debe configurar los siguientes parámetros: la función con la que desea trabajar y definir un intervalo cerrado $[a,b]$ sobre el cual la

[1] *Martín Carlos, M.Sc., Profesor de la Escuela Superior Politécnica del Litoral (ESPOL).*
(e_mail: cmmartin@espol.edu.ec).

función f sea continua (si no lo es, el software lo identifica y envía al usuario el mensaje correspondiente). A través de la interfaz gráfica el usuario también escoge si desea maximizar o minimizar la función. Parámetros puntuales del scatter search que debe fijar el usuario para el correcto funcionamiento del algoritmo son: el tamaño del conjunto R, es decir, su cardinalidad, también el número de subintervalos en el cual se va a dividir el intervalo cerrado antes indicado (la finalidad de este parámetro es asegurar la diversidad de los elementos) y, finalmente, el número de iteraciones que desea dentro del algoritmo de scatter search.

El software realiza todas las validaciones del caso que aseguran su normal comportamiento, según los parámetros configurados por el usuario para cada corrida. Una característica importante del sistema construido es que tiene opciones gráficas, es decir, en cualquier momento el usuario puede graficar la función f sobre el intervalo $[a,b]$. Una vez que el algoritmo termina su ejecución, la mejor solución encontrada se muestra en una pequeña ventana de salida y, adicionalmente, se añade el punto óptimo en el gráfico previo de la función. Para que este utilitario gráfico funcione se necesita tener instalado en el ordenador, lo siguiente: El software R con su paquete rscproxy y el R_Scilab_DCOM3.0-1B5. Además, debido a que el software ha sido desarrollado con C#.NET es necesario tener también instalado el Microsoft Framework.NET. A continuación, detalles internos de cómo opera el algoritmo implementado.

3. OPERACIÓN DEL ALGORITMO

No es un objetivo de este documento presentar el código fuente del sistema, ni hablar de la codificación en C#.NET, lo que se va a hacer es explicar, con todo el detalle necesario, los aspectos internos de la implementación.

Generación del Conjunto P

Lo primero que debe hacerse al resolver un problema de optimización con "scatter search" es construir un primer conjunto al que llamaremos P. El sistema pide entonces que el usuario defina el tamaño del conjunto R (internamente se almacena este valor en una variable $RSize$), luego la cardinalidad del conjunto P se define como: $PSize = MAX\{100,\ 5\,RSize\}$ y así aseguramos que el conjunto P tenga al menos 100 elementos.

El número de subintervalos de igual tamaño en el intervalo $[a,b]$ se va a representar con la variable n, el cual es también un parámetro del sistema que lo fija el usuario a través de la interfaz gráfica del sistema. De esta manera, la longitud de cada subintervalo (todos los subintervalos tienen la misma longitud) es:

$$\Delta x = (b-a)/n$$

Ahora, se van a generar $PSize$ elementos con una mezcla de diversidad y calidad. Primero hacemos uso de la diversidad, ¿cómo trabaja? Así: Se selecciona aleatoriamente un subintervalo (se genera un número entero aleatorio entre 1 y n), luego se genera un número aleatorio de punto flotante dentro del subintervalo seleccionado anteriormente. Finalmente, se aplica un método de mejora al número de punto flotante anterior pero sin salirse del subintervalo, es decir, se busca dentro del subintervalo, a través de una "local search", si existe "cerca" de la solución generada otra solución que mejora el comportamiento de la función objetivo (ya sea si se busca maximizar o minimizar). Posteriormente se explicará la forma de trabajar de la búsqueda local. Pero, para asegurar la diversificación, la probabilidad de elegir un subintervalo es inversamente proporcional al número de soluciones generadas en dicho subintervalo. De esta manera se trata de "balancear" el número de soluciones a lo largo de todos los subintervalos en los que se ha dividido el intervalo $[a,b]$. Para esto se hace uso de una memoria por frecuencia, es decir, a medida que se van generando soluciones, además de almacenar en el conjunto P cada solución generada (considerando que no hayan repetidas), se almacena la cantidad de soluciones generadas en cada uno de los n subintervalos (frecuencia). Algo más, el cociente $PSize/n$ debe ser un entero ya que este valor define, en promedio, el número de soluciones que se debe generar por cada subintervalo. Como podemos apreciar, se tiene ahora un conjunto P con una mezcla muy buena de calidad y diversidad.

Búsqueda Local.

La "local search" o búsqueda local implementada recibe como entrada la solución a mejorar, un límite inferior y un límite superior (los extremos del subintervalo) que definen lo "local" de la búsqueda y un parámetro adicional que fija el número de iteraciones o intentos para buscar si existe una mejor solución alrededor de la solución dada inicialmente. ¿Cómo trabaja? Se genera un número aleatorio que se le suma o se le resta (lo cual también es aleatorio) a la solución vieja para obtener un solución nueva (aquí se debe tener todo el cuidado de que la nueva solución "no se salga" de los límites definidos). Apenas se encuentra una solución mejor se detiene la búsqueda. Si se ha llegado al número de intentos tope de búsqueda definido a través del

argumento "número de iteraciones" y no se ha encontrado una mejor solución, el método devuelve la primera solución que se pasó como argumento, es decir, se indica así que "nos quedamos igualitos", no se encontró nada mejor.

Construcción del Conjunto R

En el método de "scatter search" existe un segundo conjunto importante, el conjunto R, el cual es construido a partir del conjunto P. Para construir el conjunto R nos preocupamos que la variable $RSize$ sea par, ya que la mitad de los elementos de R serán escogidos por calidad mientras que la otra mitad por diversificación, haciendo uso del conjunto P. Aquí es importante indicar que se pudo haber parametrizado esto, es decir, que b_1 elementos sean escogidos por calidad y b_2 por diversidad teniendo presente que $b_1 + b_2 = RSize$. Retomando, se toman entonces los $RSize/2$ "primeros en calidad" elementos del conjunto P (es decir, los mejores elementos de dicho conjunto) y se incorporan al conjunto R.

De los elementos que quedan de P se deben tomar $RSize/2$ elementos más para completar el conjunto R, ¿cómo se lo hace para garantizar diversidad? Se calcula la distancia de cada elemento de P (ya no se consideran aquellos que se llevan a R) al conjunto R con la fórmula: Sea $x \in P$, luego $d(x,R) = \underset{y \in R}{MIN} \mid x - y \mid$ y el elemento que se almacena en R es aquel elemento de P que dista más de R, es decir, el más lejano al conjunto R.

En este momento se tiene ya construido totalmente el conjunto R con una mezcla muy buena y adecuada de calidad y diversidad.

Actualización y Mantenimiento del Conjunto R

Durante la búsqueda del óptimo el conjunto R debe actualizarse y eso es lo que se va a explicar. Una vez construido el conjunto R se procede a crear subconjuntos de R. En esta implementación el tamaño de cada subconjunto es de dos elementos de R (pero en otras implementaciones con scatter search la cardinalidad de estos subconjuntos puede ser de más elementos), es decir, usando R se generan todas las parejas posibles de elementos sin repetir, porque se les va a aplicar un método de combinación. ¿Cómo trabaja el método de combinación? Eso se explica unas líneas más adelante. Al aplicar un método de combinación a cada subconjunto, se generan cuatro elementos y, a cada uno de estos nuevos elementos (llamados soluciones prueba) se les aplica un método de mejora, es decir, se emplea la operación de búsqueda local explicada anteriormente para estudiar la vecindad de estas "soluciones prueba" en busca de unas mejores soluciones. Entonces, por cada subconjunto se tienen cuatro "soluciones mejoradas" que se añaden a un conjunto denominado $Pool$. Es decir que finalmente en $Pool$ se tiene el resultado de aplicar métodos de "combinación" y "mejora" a cada uno de los subconjuntos construidos a partir del gran conjunto R.

Una vez que el conjunto $Pool$ está lleno se construye un conjunto de "vida corta" $Temp$ de forma tal que ahí se almacenen todos los elementos de R y de $Pool$, es decir $Temp = R \cup Pool$. Este conjunto temporal, tiene una cardinalidad igual a la cardinalidad de R más la cardinalidad de $Pool$. Los $RSize$ mejores elementos son los que pasan finalmente al conjunto R. ¿Cuántas veces ocurre esto? Hay un parámetro llamado "Número de Iteraciones" que el usuario del sistema fija antes de correr el algoritmo de scatter search, pues es este mismo parámetro el que sirve para configurar el número de veces que el conjunto R se va a actualizar.

La opción de reconstruir el conjunto R se debe indicar que no se ha incorporado en la implementación. Es decir, si el conjunto R permanece sin cambios después de una actualización (debido a que el "combinar y mejorar" no produce nada nuevo) se recomienda una reconstrucción donde participa nuevamente el conjunto P y el conjunto R se actualiza nuevamente por calidad y diversidad. No se implementó esta etapa debido a que no se lo consideró necesario porque en las pruebas que se realizaron el algoritmo siempre encontraba el óptimo, es decir, el software encontraba correctamente el óptimo en todas las variantes en las que se lo puso a prueba, por lo que consideró innecesario agregar una capa más de implementación.

Método de Combinación

El método de combinación implementado es muy sencillo y opera así: se generan cuatro números aleatorios de punto flotante en el intervalo $(0,1)$ con la idea de aplicar combinaciones lineales "convexas". Es decir, el método de combinación lo único que recibe como parámetro es el subconjunto de R que tiene dos soluciones $SolA$ y $SolB$, y devuelve como salida cuatro combinaciones lineales convexas distintas. Es importante decir que esta salida no pasa directamente a R ya que antes se aplica el método de mejora a cada una de las cuatro

soluciones obtenidas por combinación, y éstas sí pasan ya al conjunto R.

Método Principal del Algoritmo

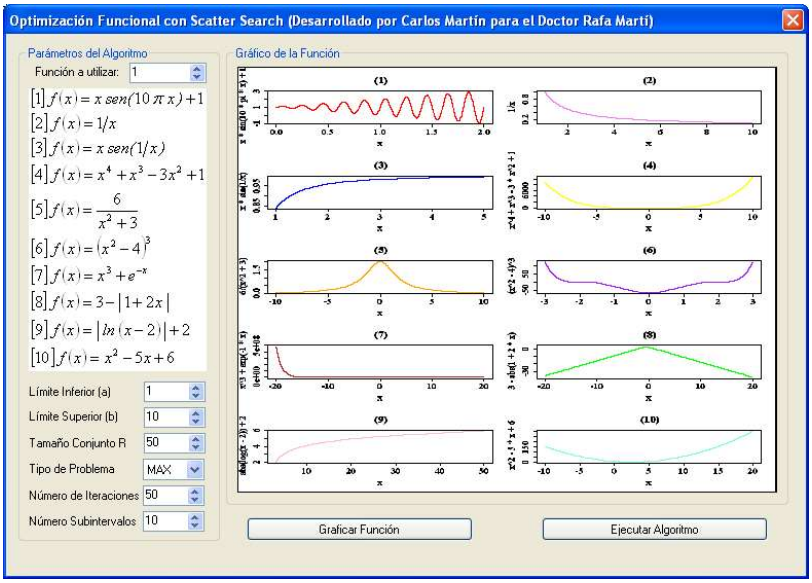
A continuación se va a mostrar el método "main" que gobierna el comportamiento y la operación general del algoritmo de scatter search que se implementó. Todas las etapas o fases anteriores se coordinan y se enlazan aquí. A continuación el método:

```
public double Main(out double Fitness)
{
   double[ ] P = new double[this.PSize];
   ArrayList Pool = new ArrayList();
   this.GenerarP(P);
   double[ ] R = new double[this.RSize];
   this.ConstruirR(P, R);
   ArrayList SubSets = new ArrayList();
   int NumIteraciones = 0;
   while (true)
   {
      this.GenerarSubSets(SubSets, R);
      while (SubSets.Count != 0)
      {
         double[ ] SubSet =
               (double[ ])SubSets[0];
         double[ ] TrialSol =
this.Combinar(SubSet);
         double MejoraUno =
this.LS(TrialSol[0], a, b);
               Pool.Add(MejoraUno);
               double MejoraDos =
this.LS(TrialSol[1], a, b);
               Pool.Add(MejoraDos);
               double MejoraTres =
this.LS(TrialSol[2], a, b);
               Pool.Add(MejoraTres);
               double MejoraCuatro =
this.LS(TrialSol[3], a, b);
               Pool.Add(MejoraCuatro);
               SubSets.RemoveAt(0);
            }
            this.ActualizarR(R, Pool);
NumIteraciones++;
            if (NumIteraciones ==
this.NumIteraciones) break;
         }
         Fitness = EvalR[0];
         return R[0];
      }
```

4. EJECUCIÓN DEL ALGORITMO DE SCATTER SEARCH

A continuación se presenta una corrida del software. Lo primero que se debe hacer es ejecutar el archivo OFSS.exe (OFSS quiere decir "Optimización Funcional con Scatter Search") y luego se verá la siguiente ventana:

FIGURA 1
Scatter search y la optimización funcional
Ventana Principal del Software



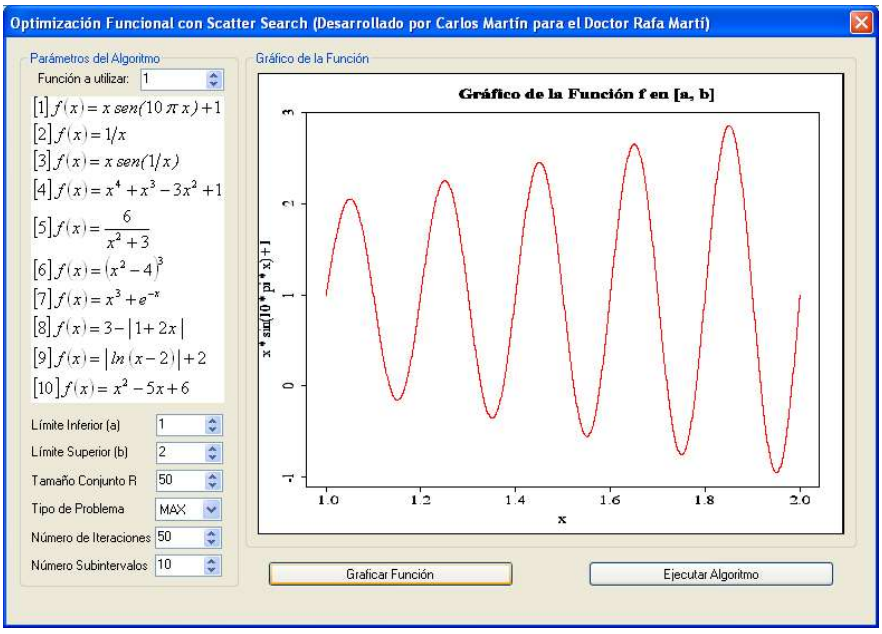
Podemos apreciar en la gráfica que el sistema asigna valores por "default" a los parámetros del software antes mencionados en 2.1, pero como también se dijo, se pueden cambiar antes de cada

corrida del algoritmo de scatter search. Por ejemplo, de las 10 funciones (en la ventana se muestran sus gráficas en algún intervalo cerrado), está configurada la primera función. Como ejemplo, se presenta la gráfica de la función $f(x) = x\,sen(10\,\pi\,x) + 1$ en el intervalo cerrado $[1, 2]$. Para graficar una de las 10 funciones en cualquier intervalo cerrado se debe ingresar un número entre 1 y 10, incluidos, donde dice "Función a utilizar" para escoger la función. Luego se debe ingresar en "Límite Inferior (a)" el punto frontera izquierdo del intervalo cerrado y en "Límite Superior (b)" el punto frontera derecho del intervalo cerrado. Por supuesto se hace la validación $b > a$ y se chequea que la función sea continua. Ahora se procede a hacer un "click" en el botón etiquetado "Graficar Función". Con esto se verá la gráfica de la función. El objetivo de esto es que el usuario pueda VER un gráfico de cualquiera de las funciones en cualquier intervalo cerrado donde la función sea continua y, de esta manera, que el usuario pueda apreciar, desde un punto de vista gráfico, lo que ocurre, y pueda "sospechar" dónde se encuentra el óptimo del problema.

FIGURA 2

Scatter search y la optimización funcional

Gráfica de la función [1] en el intervalo [1,2]

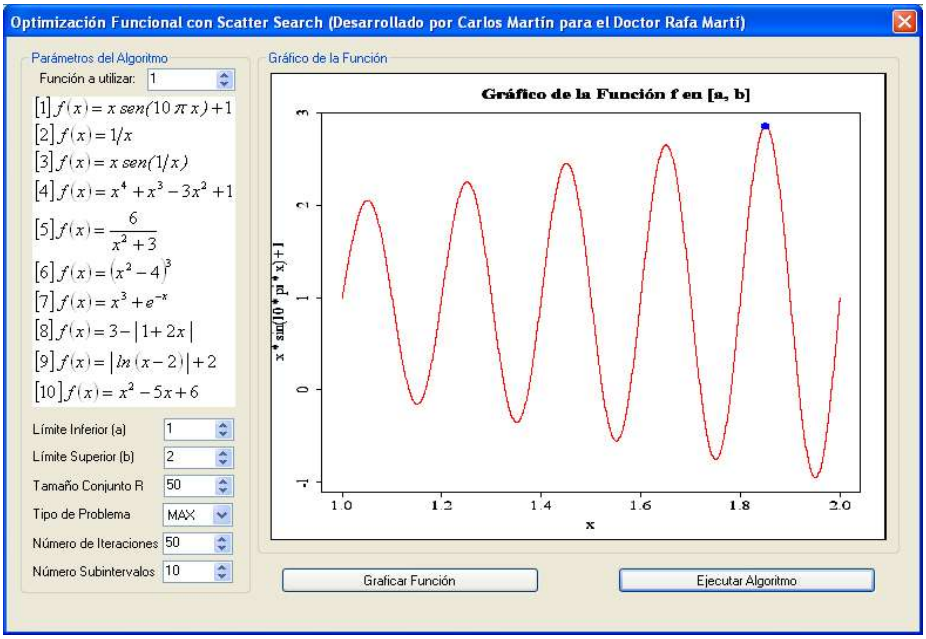


Podemos ver entonces la gráfica de la función $f(x) = x\,sen(10\,\pi\,x) + 1$ en el intervalo cerrado $[1, 2]$. La pregunta es ¿por qué dentro de las 10 funciones se escogió la primera como la función por default para trabajar? Se puede apreciar que, ya sea si se desea maximizar o minimizar, existe con certeza un óptimo global, pero también tenemos muchos óptimos locales, y si se expande el intervalo, el número de óptimos locales es aún mayor. La intención era probar si la filosofía de diversidad y calidad del algoritmo scatter search le permitía, por lo menos al resolver el problema de la "optimización funcional no lineal", no quedar atrapado en óptimos locales. Se va a mostrar sólo una ejecución del software construido, pero se debe decir que probó el sistema, sin exagerar, algunas decenas de veces, con todas las 10 funciones y con muchos intervalos cerrados. En ninguna ejecución el algoritmo se quedo atrapado en un óptimo local, siempre presentó como solución el óptimo global, es decir, la mejor solución encontrada fue siempre el óptimo del problema. Finalmente, una vez que se grafica la función no lineal f en un intervalo $[a, b]$ donde se tiene continuidad, el usuario debe hacer "click" en el botón "Ejecutar Algoritmo" para que se ejecute el scatter search. A continuación se muestra la ventana del final de la ejecución del software.

FIGURA 3
Scatter search y la optimización funcional
Salida en valores del algoritmo (MAX)

Esta ventana indica la mejor solución encontrada y, también, el valor que se obtiene al evaluar a la función no lineal en la mejor solución encontrada y, además, resalta el hecho de que se puede observar, en el gráfico, esa mejor solución encontrada con un punto bien pintado de color azul. Se muestra a continuación el gráfico de la función incluida la mejor solución encontrada "pintada de azul". En este caso se ha escogido maximizar la función no lineal.

FIGURA 4
Scatter search y la optimización funcional
Resultado gráfico del algoritmo (MAX)

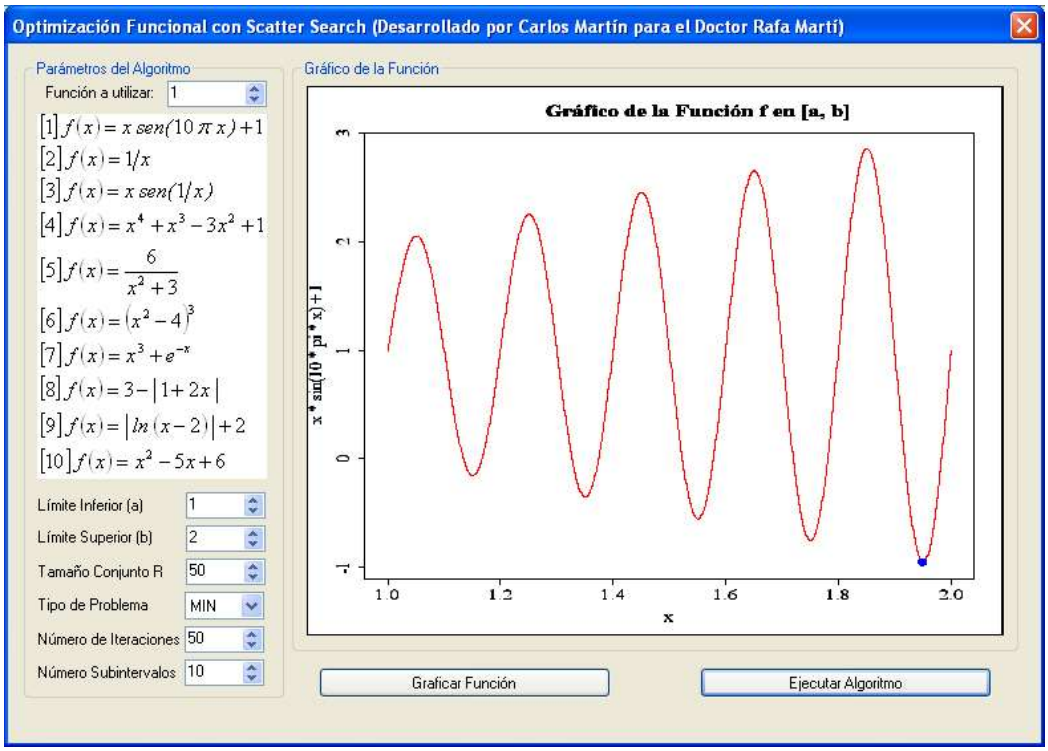


Ahora se va a escoger ***minimizar*** la función en el problema y, de la misma manera que antes, se mostrará la mejor solución encontrada, primero de manera analítica en una ventana pequeña y luego gráficamente pintada de color azul junto con el gráfico de la función que es de color rojo.

FIGURA 5
Scatter search y la optimización funcional
Salida en valores del algoritmo (MIN)

Finalmente se muestra la mejor solución encontrada, ahora gráficamente.

FIGURA 6
Scatter search y la optimización funcional
Resultado gráfico del algoritmo (MIN)



Se puede observar el punto pintado de color azul sobre la gráfica de la función de color rojo. Dicho punto azul representa la mejor solución encontrada. Cabe resaltar lo flexible del software al ejecutar el algoritmo de scatter search, ya que podemos tratar con cualquiera de las 10 funciones dadas, en cualquier intervalo cerrado donde la función no lineal sea continua, podemos "jugar" con todas las gráficas posibles y, al final, cuando el usuario se ha decidido respecto de la función y el intervalo cerrado en el que desea trabajar, simplemente escoge si desea maximizar o minimizar para finalmente hacer "click" en el botón etiquetado "Ejecutar Algoritmo".

5. CONCLUSIONES Y RECOMENDACIONES

La recomendación principal para futuros trabajos es la de extender a más variables independientes la implementación realizada, es decir, el poder optimizar funciones no lineales de varias variables reales. Otra recomendación sería la de hacer más flexible al software en el sentido de permitir que el usuario ingrese, a través de la interfaz gráfica, la función con la que desea trabajar, y que el sistema no tenga como restricción el poder maximizar o minimizar sólo una cuantas. Se puede apreciar también que "scatter search" no hace un uso intensivo de la aleatoriedad, es decir, no abusa de ella como sí lo hacen, por ejemplo, los algoritmos genéticos. La mezcla de diversidad y calidad del algoritmo muestran lo inteligente de su forma de trabajar.

REFERENCIAS BIBLIOGRÁFICAS Y ELECTRÓNICAS

[1]. **LAGUNA Y MARTÍ** (2003). *"Scatter Search. Methodology and Implementations in C"*, Kluwer Academic Publishers, Norwell Massachusetts

[2]. **MARTÍ Y LAGUNA** (2003). "*Scatter Search: Diseño Básico y Estrategias Avanzadas*", Universidad de Valencia, España.

[3]. **Microsoft Developer Network (MSDN) ONLINE.** http://msdn.microsoft.com