Scatter Search: Basic Design and Advanced Strategies

Rafael Martí (1), Manuel Laguna (2)

(1) Departamento de Estadística e Investigación Operativa
Facultad de Matemáticas
Universidad de Valencia
Dr. Moliner 50 46100 Valencia, Spain

(2) Leeds School of Business
University of Colorado
Boulder, CO 80309-0419, USA

e-mail: rmarti@uv.es, laguna@colorado.edu

Scatter search is an evolutionary method that has been successfully applied to hard optimization problems. The fundamental concepts and principles of the method were first proposed in the 1970s and were based on formulations, dating back to the 1960s, for combining decision rules and problem constraints. The combination strategy was devised with the belief that information could be exploited more effectively when integrated than when treated in isolation. In contrast to other evolutionary methods like genetic algorithms, scatter search is mostly based on systematic designs and methods with the purpose of creating new solutions. It uses strategies for search diversification and intensification that have proved effective in a variety of optimization problems. The Scatter Search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication. This paper's goal is to provide a grounding in the essential ideas of Scatter Search that will enable readers to create successful applications of their own. The paper also introduces an application of the method to solve the well-known knapsack problem, in order to illustrate some implementation details.

Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.19 (2003),pp. 123-130
ISSN: 1137-3601. © AEPIA (http://www.aepia.org/revista).

Scatter Search:
Diseño Básico y Estrategias Avanzadas

Rafael Martí * y Manuel Laguna[+]

*Departamento de Estadística e Investigación Operativa,
Universitat de València, 46100 Burjassot (Valencia), Spain.
rafael.marti@uv.es

[+]Leeds School of Business, University of Colorado,
Boulder, CO 80309-0419, USA.
laguna@colorado.edu

Resumen

Scatter Search es un método metaheurístico para resolver problemas de optimización. Aunque fue originalmente introducido en los años setenta, recientemente es cuando ha sido probado en numerosos problemas difíciles con gran éxito. Pertenece a la familia de los llamados Algoritmos Evolutivos, los cuales se distinguen por estar basados en la combinación de un conjunto de soluciones. Aunque presenta similitudes con los Algoritmos Genéticos, difiere de éstos en principios fundamentales, tales como el uso de estrategias sistemáticas en lugar de aleatorias. Scatter Search proporciona un marco flexible que permite el desarrollo de diferentes implementaciones con distintos grados de complejidad. En este trabajo se realiza una revisión del método desde sus orígenes hasta los aspectos más novedosos, que dan lugar a algoritmos más eficientes. Además, se ilustra la aplicación del método en la resolución del conocido problema de la mochila.

Palabras clave: MetaHeurísticos, Búsqueda Local, Optimización Combinatoria.

1. Introducción

Scatter search (SS), también conocido en castellano como Búsqueda Dispersa, es un procedimiento metaheurístico basado en formulaciones y estrategias introducidas en la década de los sesenta. Los conceptos y principios fundamentales del método, fueron propuestos a comienzo de la década de los setenta, basados en las estrategias para combinar reglas de decisión, especialmente en problemas de secuenciación, así como en la combinación de restricciones (como el conocido método de las restricciones subrogadas).

SS opera sobre un conjunto de soluciones, llamado conjunto de referencia, combinando éstas para crear nuevas soluciones de modo que mejoren a las que las originaron. En este sentido decimos que es un método evolutivo. Sin embargo, a diferencia de otros métodos evolutivos, como los algoritmos genéticos, SS no está fundamentado en la aleatorización sobre un conjunto relativamente grande de soluciones sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño. Como ilustración basta decir que los algoritmos genéticos suelen considerar una población de 100 soluciones mientras que en la búsqueda dispersa es habitual trabajar con un conjunto equivalente de tan sólo 10 soluciones. SS se basa en el principio de que la información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones puede ser utilizado mediante la combinación de éstas en lugar de aisladamente.

Una de las características más notables de SS es que se basa en integrar la combinación de soluciones con la búsqueda local. Aunque en diseños avanzados esta búsqueda local puede contener una estructura de memoria (tabu search), no es necesario que así sea, limitándose, en la mayoría de los casos a una búsqueda local convencional.

La primera descripción del método fue publicada en 1977 por Fred Glover [5], donde establece sus principios. En este primer artículo se determina que SS realiza una exploración sistemática sobre una serie de buenas soluciones llamadas conjunto de referencia. Los siguientes comentarios resumen los principales aspectos de este trabajo:

- El método se centra en combinar dos o más soluciones del conjunto de referencia. La combinación de más de dos soluciones tiene como objetivo el generar centroides.

- Generar soluciones en la línea que unen dos dadas se considera una forma reducida del método.
- Al combinar se deben seleccionar pesos apropiados y no tomar valores al azar.
- Se deben realizar combinaciones "convexas" y "no convexas" de las soluciones.
- La distribución de los puntos se considera importante y deben de tomarse dispersos.

En Glover [6] se introduce la combinación ponderada (*weighted combination*) como el mecanismo principal para generar nuevas soluciones. En esta versión se enfatizan las búsquedas lineales entre dos soluciones y el uso de pesos para muestrear en dicha línea. Asimismo, se introduce el concepto de combinar soluciones de calidad con soluciones diversas. Además, el método incluye una componente de intensificación que consiste en tomar una muestra mayor de la línea que ha producido mejores soluciones.

En este artículo el autor especifica que para trabajar con problemas con variables enteras, binarias o que forman una permutación, hay que diseñar métodos específicos de combinación (notar que no tiene sentido hablar de la combinación lineal de dos permutaciones). Para ello se introducen los mecanismos de combinación basados en votos. En estos se definen reglas mediante las cuales cada solución "vota" para que sus características aparezcan en la solución que se está construyendo. Estos métodos de votos han sido frecuentemente utilizados en las rutinas de combinación de los algoritmos de SS y parece que constituyen uno de las claves del éxito de estos métodos.

En 1998 Glover [7] publica una versión más específica del método en donde se recogen y simplifican muchas de las ideas expuestas en trabajos anteriores. Esta publicación tuvo un gran impacto en lo que a la difusión del método se refiere y ha quedado como la referencia *standard* de la búsqueda dispersa. Numerosos investigadores comenzaron a aplicar SS a la resolución de problemas de optimización obteniendo resultados de gran calidad. La siguiente sección describe esta versión del método, actualizada según implementaciones y desarrollos posteriores.

Glover, Laguna y Martí [9] estudian las implementaciones más recientes del método en la resolución de problemas de optimización combinatoria. Además, muestran las conexiones entre este método y el denominado Re-encadenamiento de Trayectorias ("*Path relinking*" que abreviaremos como PR). Así, desde un punto de vista espacial, el proceso de generar combinaciones lineales de un conjunto de referencia de soluciones, puede ser visto como el generar caminos entre, y más allá, de estas soluciones. Esto lleva a una concepción más amplia del significado de combinar que es la introducida en el PR.

PR se basa en el hecho de que entre dos soluciones se puede trazar un camino que las una, de modo que las soluciones en dicho camino contengan atributos de ellas. Las soluciones originales pueden haber sido generadas mediante un método basado en una búsqueda local y estar unidas por un camino, o haber sido generadas por otro método y no estar unidas de ningún modo; en cualquier caso, PR genera un nuevo camino que las une. Las características de dicho camino vendrán especificadas por los atributos que son añadidos o eliminados, o por los movimientos realizados para alcanzar una solución desde la otra. Esto constituye una extensión del concepto de combinación en tanto que se obtienen varias soluciones a partir de dos o más originales.

Consideremos en un problema de permutaciones dos soluciones $x = (1,3,4,2)$ e $y = (2,3,1,4)$. Si aplicamos un método de combinación podemos obtener una determinada solución $z = (1,3,2,4)$, mientras que si tratamos de llegar hasta y partiendo de x, podemos obtener la secuencia $z_1 = (2,1,3,4)$, $z_2 = (2,3,1,4)$. En este sentido decimos que PR es una generalización de los métodos de combinación.

Laguna y Martí [12] publican el primer libro monográfico sobre este método. Los autores realizan un estudio exhaustivo de SS, comenzando por unos tutoriales básicos, revisando los diseños más avanzados y llegando hasta las líneas de investigación actuales. El texto se acompaña de código fuente en C para aplicar las técnicas descritas.

2. Descripción del Método

Scatter Search se basa en combinar las soluciones que aparecen en el llamado conjunto de referencia. Este conjunto almacena las "buenas" soluciones que se han ido encontrando durante el proceso de búsqueda. Es importante destacar que el significado de buena no se restringe a la calidad de la solución, sino que también se considera la diversidad que ésta aporta al conjunto de referencia. SS consta básicamente de cinco elementos o métodos que describimos a continuación (mantenemos la acepción en ingles):

1. **Diversification Generation Method**. Un generador de soluciones diversas. El método se basa en generar un conjunto P de soluciones diversas (alrededor de 100), del que extraeremos un subconjunto pequeño (alrededor de 10) que denominamos conjunto de referencia *RefSet*.
2. **Improvement Method**. Un método de mejora. Típicamente se trata de un método de búsqueda

local para mejorar las soluciones, tanto del conjunto de referencia como las combinadas antes de estudiar su inclusión en el conjunto de referencia. Es importante destacar que en las implementaciones donde se manejen soluciones no factibles, este método ha de ser capaz de, a partir de una solución no factible, obtener una que sea factible y después intentar mejorar su valor. Si el método no logra mejorar a la solución inicial, se considera que el resultado es la propia solución inicial.

3. **Reference Set Update Method**. Un método para crear y actualizar el conjunto de referencia *RefSet*. A partir del conjunto de soluciones diversas *P* se extrae el conjunto de referencia según el criterio de contener soluciones de calidad y diferentes entre sí (Calidad y Diversidad). Las soluciones en este conjunto están ordenadas de mejor a peor respecto de su calidad.

   3.1. **Creación**. Iniciamos el conjunto de referencia con las *b*/2 (|*RefSet*|=*b*) mejores soluciones de *P*. Las *b*/2 restantes se extraen de *P* con el criterio de maximizar la mínima distancia con las ya incluidas en el conjunto de referencia. Para ello debemos de definir previamente una función de distancia en el problema.

   3.2. **Actualización**. Las soluciones fruto de las combinaciones pueden entrar en el conjunto de referencia y reemplazar a alguna de las ya incluidas, en caso de que las mejoren. Así pues, el conjunto de referencia mantiene un tamaño *b* constante, pero el valor de sus soluciones va mejorando a lo largo de la búsqueda. En implementaciones sencillas, la actualización de este conjunto se realiza únicamente por calidad, aunque se puede hacer también por diversidad.

4. **Subset Generation Method**. Un método para generar subconjuntos de *RefSet* a los que se aplicará el método de combinación. SS se basa en examinar de una forma bastante exhaustiva todas las combinaciones de *RefSet*. Este método especifica la forma en que se seleccionan los subconjuntos para aplicarles el método de combinación. Una implementación sencilla, utilizada a menudo, consiste en restringir la búsqueda a parejas de soluciones. Así el método considera todas las parejas que se pueden formar con los elementos de *RefSet* y a todas ellas le aplica el método de combinación.

5. **Solution Combination Method.** Un método de combinación. SS se basa en combinar todas las soluciones del conjunto de referencia. Para ello, se consideran los subconjuntos formados por el método del paso 4, y se les aplica el método de combinación. La solución o soluciones que se obtienen de esta combinación pueden ser inmediatamente introducidas en el conjunto de referencia (actualización dinámica) o almacenadas temporalmente en una lista hasta terminar de realizar todas las combinaciones y después ver qué soluciones entran en éste (actualización estática).

El siguiente esquema muestra cómo actúan los elementos descritos en un esquema básico del algoritmo para un problema de minimización.

***Algoritmo** Scatter Search*

1. *Comenzar con P = Ø. Utilizar el **método de generación** para construir una solución y el **método de mejora** para tratar de mejorarla; sea x la solución obtenida. Si $x \notin P$ entonces añadir x a P, en otro caso, rechazar x. Repetir esta etapa hasta que P tenga un tamaño prefijado.*
2. *Construir el **conjunto de referencia** RefSet = { x^1, ..., x^b } con las b/2 mejores soluciones de P y las b/2 soluciones de P más diversas a las ya incluidas.*
3. *Evaluar las soluciones en RefSet y ordenarlas de mejor a peor respecto a la función objetivo.*
4. *Hacer NuevaSolución = TRUE*

***Mientras** (NuevaSolución)*

5. *NuevaSolución = FALSE*
6. *Generar los subconjuntos de RefSet en los que haya al menos una nueva solución.*

   ***Mientras** (Queden subconjuntos sin examinar)*

   7. *Seleccionar un subconjunto y etiquetarlo como examinado.*
   8. *Aplicar el **método de combinación** a las soluciones del subconjunto.*
   9. *Aplicar el **método de mejora** a cada solución obtenida por combinación. Sea x la solución mejorada:*

      Si($f(x) < f(x^b)$ y x no está en RefSet)

      10. *Hacer x^b = x y reordenar Refset*
      11. *Hacer NuevaSolución = TRUE*

El paso 6 del algoritmo hace referencia a los subconjuntos de *RefSet* construidos con el "Subset Generation Method" ya que podemos combinar parejas, tríos o cualquier número de soluciones. Es usual el limitar las combinaciones a parejas, por lo que este punto equivaldría a decir: "Generar todas las parejas de soluciones de *RefSet* en las que al menos una de las dos sea nueva"; donde por nueva entenderemos que haya entrado al conjunto después de realizar la última combinación de todo *RefSet*.

La Figura 1 muestra un esquema del método básico descrito. Notar que el algoritmo se detiene cuando al

tratar de combinar vemos que no hay nuevos elementos en el conjunto de referencia (la variable NuevaSolución está en 0). Este algoritmo puede ser anidado en un esquema global que permita reconstruir el conjunto de referencia cuando éste ya ha sido utilizado. Una estrategia habitual es regenerar el conjunto de referencia dejando la mitad superior (*b*/2 mejores) y eliminando la mitad inferior. Después, se genera un conjunto *P* como al comienzo del algoritmo, del que se extraen únicamente las *b*/2 soluciones más diversas con las ya existentes en *RefSet*. De esta forma obtenemos un nuevo conjunto de referencia en el que mantenemos las soluciones de calidad y renovamos las debidas a diversidad. Ahora se vuelve a combinar como anteriormente sobre este conjunto de referencia (pasos 5 a 11 en el algoritmo). De este modo se obtiene un esquema cíclico indefinido al que hay que añadirle una variable de control para detenerlo. Esta variable suele estar en función del tiempo o del número de iteraciones (evaluaciones de la función objetivo).

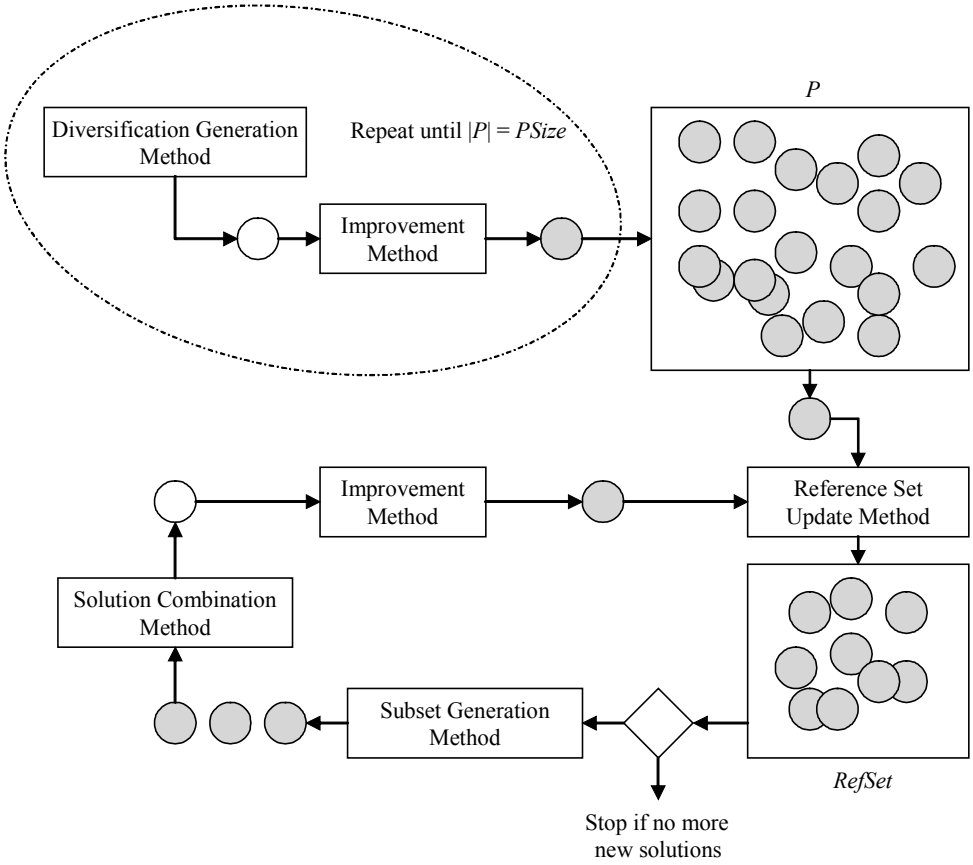

Figura 1. Esquema del Método

3. Estrategias Avanzadas

Scatter Search es un método cuyos principios fueron propuestos hace casi tres decadas pero que su desarrollo y aplicación es relativamente reciente. Durante los últimos años se han realizado nuevas contribuciones aplicando esta metodología en la resolución de conocidos problemas de optimización. Algunas de estas aplicaciones ([4] y [6]), y desarrollos teóricos [9], han abierto nuevos campos de estudio, ofreciendo alternativas a los diseños conocidos. Laguna y Martí [12] realizan una revisión de los aspectos clave del método desarrollados durante estos años. A continuación los resumimos de forma esquemática:

a. Considerar el **uso de memoria** basada en la frecuencia para desarrollar métodos de diversificación eficientes. En lugar de generar las soluciones al azar, se propone construir un método que, basado en el número de apariciones de los elementos significativos en la solución, evite la repetición de estructuras similares.

b. **El tamaño** "óptimo" **del conjunto de referencia** está por determinar. Es necesario

continuar con la experimentación para determinar si debemos mantener este número *b* relativamente pequeño, o en algunos problemas donde SS no ha proporcionado buenos resultados deberíamos aumentarlo y prolongar los tiempos de ejecución del algoritmo. También hemos de estudiar los diseños donde *b* sea variable en función del estado de la búsqueda. Inicializar el conjunto de referencia a partir de un gran conjunto de soluciones creado con el generador antes mencionado, proporcionaría un conjunto con mayor diversidad que podría generar mejores soluciones en un horizonte mayor.

c. **Aplicar la rutina de mejora de forma selectiva**. Las pruebas indican que el aplicar el método de mejora a todas las soluciones generadas y combinadas, no garantiza el obtener mejores resultados finales. Establecer umbrales de calidad para no aplicar la mejora a soluciones que difícilmente van a proporcionar la mejor solución, es un gasto innecesario de tiempo de computación (Ugray y otros [14]). Por otro lado, al aplicar el método de mejora a todas las soluciones se acelera la convergencia de éste, lo cual puede ser deseable si disponemos de poco tiempo de computación, pero debemos de evitarlo si queremos ejecutar el método en un horizonte largo para obtener soluciones de gran calidad.

d. Es necesario estudiar el porcentaje de tiempo que el método está generando soluciones y el tiempo que está combinando. En esencia ésta es la cuestión que se plantea en todos los métodos heurísticos: el **equilibrio entre la intensificación y la diversificación**.

e. Inicializar el conjunto de referencia con la mitad de soluciones por calidad y la otra mitad por diversidad. Se han realizado experimentos con distintos tamaños y parece ser que esta proporción es la que mejores resultados está dando.

f. La calidad es más importante que la diversidad al actualizar el conjunto de referencia. Notar que aunque el método comienza con un conjunto de referencia con soluciones de calidad y diversidad, al realizar las combinaciones, sólo entran al conjunto las soluciones por el criterio de calidad (hasta llegar a la etapa de regenerarlo). En Laguna y Martí [9] se han probado actualizaciones de este conjunto por diversidad, obteniendo resultados finales peores.

g. Comparar las **actualizaciones** del conjunto de referencia **estática y dinámica**. Notar que al combinar las soluciones podemos aplicar dos estrategias, introducirlas en el conjunto, si procede, nada más generarlas, o anotarlas en una "pila" y cuando terminemos de realizar todas las combinaciones, proceder a la actualización. La primera estrategia es dinámica y más agresiva, en tanto que las soluciones buenas entran rápidamente en el conjunto de referencia, pero dado que este es de tamaño constante, esto implica que hay soluciones que salen sin llegar a haber sido utilizadas para realizar combinaciones. Es necesario comparar ambas estrategias para ver cual proporciona mejores resultados finales.

h. La mayor parte de las soluciones de calidad proceden de combinaciones de dos soluciones. Asimismo, las buenas soluciones suelen proceder de combinar buenas soluciones. Campos et al. ([1] y [3]) realizan numerosos experimentos realizando un seguimiento a lo largo de la búsqueda de las soluciones de calidad en un problema concreto.

i. El uso de **múltiples métodos de combinación** ha de ser considerado. Campos, Laguna y Martí [2] realizan un análisis de diferentes métodos de combinación, algunos con elementos aleatorios y otros deterministas, de modo que el algoritmo selecciona el método de combinación probabilísticamente, de acuerdo con los éxitos obtenidos por éste. De la misma forma que los métodos de búsqueda local basados en distintos entornos (Variable Neighborhood Search) están proporcionando muy buenos resultados, hemos de considerar el definir varios "entornos" de combinación para realizar una búsqueda más completa del espacio de soluciones.

En el capítulo 8 de [12] podemos encontrar una revisión de las principales aplicaciones de *Scatter Search* y *Path Relinking* en la resolución de problemas de optimización. Tan sólo citaremos aquí el título y los autores de los trabajos originales.

- *The Graph Coloring Problem* (Hamiez y Hao)
- *Capacitated Multicommodity Network Design* (Ghamlouche, Crainic and Gendreau)
- *The Maximum Clique Problem* (Cavique, Rego y Themido)
- *Assigning Proctor to Exams* (Ramalhinho, Laguna and Martí)
- *Periodic Vehicle Loading* (Delgado, Laguna and Pacheco)
- *The Linear Ordering Problem* (Campos, Laguna y Martí)
- *The Bipartite Drawing Problem* (Laguna y Martí)

- *Job Shop Scheduling* (Nowicki y Smutnicki)
- *The Arc Routing Problem* (Greistorfer)
- *Multiple Criteria Scatter Search* (Beausoleil)
- *Meta-Heuristic Use of Scatter Search via OptQuest* (Hill)
- *Generation of Diverse MIP Solutions* (Glover, Løkketangen and Woodruff)
- *Path Relinking to Improve Iterated Start Procedures* (Ribeiro and Resende)

4. Una Aplicación del Método

En esta sección consideraremos el "problema de la mochila" para aplicar el método *Scatter Search*. Este problema ha sido muy estudiado y nuestro objetivo no es competir con los métodos de resolución actuales, sino ilustrar las técnicas descritas. El problema consiste en seleccionar, de un conjunto de elementos, el subconjunto que maximiza el valor de la función objetivo, sujeto a una restricción de capacidad.

$$\begin{aligned} &\textit{Maximize} && \sum_i c_i x_i \\ &\textit{Subject to} && \sum_i a_i x_i \le d \\ & && x_i \in \{0,1\} \qquad \forall i \end{aligned}$$

Este problema tiene muchas aplicaciones y debe su nombre a la situación en la que un montañero debe llenar su mochila, seleccionando algunos elementos de modo que no exceda de la capacidad de ésta (d). En la formulación anterior, cada elemento i tiene un valor c_i y un peso a_i. La variable x_i indica si el elemento ha sido seleccionado (x_i=1). En Martello y Toth [13] podemos encontrar una revisión de los principales métodos de resolución. Para ilustrar el método *Scatter Search* consideramos el siguiente ejemplo del problema de la mochila:

Maximizar

$11x_1 + 10x_2 + 9x_3 + 12x_4 + 10x_5 + 6x_6 + 7x_7 + 5x_8 + 3x_9 + 8x_{10}$

Sujeto a

$33x_1 + 27x_2 + 16x_3 + 14x_4 + 29x_5 + 30x_6 + 31x_7 + 33x_8 + 14x_9 + 18x_{10} \le 100$

$x_i \in \{0, 1\} \quad i = 1, \ldots, 10.$

En [7] se describe un generador de soluciones sistemático (es decir, no aleatorio sino determinista) para generar vectores 0-1. Utilizando dicho generador hemos construido el conjunto inicial P con 100 soluciones para nuestro problema. La tabla 1 muestra las 10 primeras. Notar que algunas de las soluciones son infactibles y por lo tanto, su valor ($f(x)$) puede ser mayor que el óptimo del problema (igual a 44).

Sol.	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	$f(x)$
1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	81
3	0	1	0	1	0	1	0	1	0	1	41
4	1	0	1	0	1	0	1	0	1	0	40
5	0	1	1	0	1	1	0	1	1	0	43
6	1	0	0	1	0	0	1	0	0	1	38
7	1	0	1	1	0	1	1	0	1	1	56
8	0	1	0	0	1	0	0	1	0	0	25
9	1	1	0	1	1	0	1	1	0	1	63
10	0	0	1	0	0	1	0	0	1	0	18

Tabla 1. Conjunto *P*

Dado que las soluciones que genera el método considerado pueden no ser factibles, el método de mejora debe incorporar un mecanismo para garantizar la factibilidad de la solución final. Vamos a considerar un procedimiento muy sencillo que, en la primera fase, hace factible la solución, y en la segunda, trata de mejorar su valor.

Fase de Factibilidad

Para hacer una solución factible, basta con cambiar el valor de algunas variables de 1 a 0 hasta satisfacer la restricción del problema. Consideraremos las variables en orden creciente respecto al cociente valor/peso, comenzando por la de menor ratio. Una vez la solución es factible (cumple la restricción) pasamos a la fase de mejora.

Fase de Mejora

Al contrario de la fase anterior, aquí cambiaremos el valor de algunas variables de 0 a 1 para incrementar el valor de la solución. Comenzaremos ahora por la variable con mayor cociente valor/peso y continuaremos en orden decreciente mientras la solución sea factible.

La Tabla 2 muestra las soluciones de la Tabla 1, una vez aplicada la fase de factibilidad y la de mejora. Notar que como las soluciones 1, 6, 8 y 10 ya eran factibles, no es necesario aplicarles la primera fase. Como era de esperar, las demás soluciones experimentan un deterioro en su valor al hacerlas factibles, y un incremento al aplicarles la fase de mejora.

Notar que en este ejemplo trivial, el método de mejora ha obtenido la solución óptima del problema (con valor 44) a partir de la solución 5.

Una vez aplicada la rutina de mejora a las soluciones de P, eliminamos las posibles duplicaciones (notar que soluciones iniciales diferentes pueden conducir a la misma solución final) y nos quedamos con las que son diferentes. Podemos seguir aplicando el generador de soluciones hasta obtener 100 soluciones mejoradas factibles que sean todas diferentes.

Sol.	Solución Factible	Valor	Solución Mejorada	Valor
1	(0,0,0,0,0,0,0,0,0,0)	0	(0,1,1,1,0,0,0,0,0,1)	39
2	(0,1,1,1,0,0,0,0,0,1)	39	(0,1,1,1,0,0,0,0,0,1)	39
3	(0,1,0,1,0,1,0,0,0,1)	36	(0,1,0,1,0,1,0,0,0,1)	36
4	(1,0,1,0,1,0,0,0,0,0)	30	(1,0,1,1,1,0,0,0,0,0)	42
5	(0,1,1,0,1,0,0,0,1,0)	32	(0,1,1,1,1,0,0,0,1,0)	44
6	(1,0,0,1,0,0,1,0,0,1)	38	(1,0,0,1,0,0,1,0,0,1)	38
7	(1,0,1,1,0,0,0,0,0,1)	40	(1,0,1,1,0,0,0,0,0,1)	40
8	(0,1,0,0,1,0,0,1,0,0)	25	(0,1,0,0,1,0,0,1,0,0)	25
9	(0,1,0,1,1,0,0,0,0,1)	40	(0,1,0,1,1,0,0,0,0,1)	40
10	(0,0,1,0,0,1,0,0,1,0)	18	(0,0,1,1,0,1,0,0,1,1)	38

Tabla 2. Soluciones Mejoradas

A partir del conjunto *P* construimos el conjunto de referencia *RefSet* incluyendo las 5 mejores de *P*. Para completar *RefSet* hasta un tamaño de 10 añadimos las 5 más diversas respecto de las consideradas. Aunque el cálculo de la máxima diversidad es un problema *NP-duro* en sí mismo [12], un método aproximado que se emplea habitualmente en las implementaciones de *Scatter Search* consiste en añadir una a una las solución de *P* con una distancia máxima a *RefSet*. Se considera la distancia de una solución al conjunto *RefSet* como la mínima de las distancias de dicha solución a todas las de *RefSet*. En este problema hemos considerado la distancia entre dos soluciones como la suma de las diferencias en valor absoluto entre sus variables:

$$d(x,y)=\sum_i abs(x_i-y_i)$$

Una vez construido *RefSet* pasamos a realizar las combinaciones de sus elementos. Consideraremos la implementación más sencilla del *"Subset Generation Method"* que consiste en realizar únicamente las combinaciones de dos elementos. Así pues, en cada fase de combinación, consideramos todas las parejas de soluciones de *RefSet* que no han sido combinadas anteriormente.

Para cada pareja de soluciones, el método de combinación considerado calcula una puntuación (*score*) para cada variable. El *score* para la variable *i* al combinar las soluciones *j* y *k* de *RefSet* (ordenado) se calcula según:

$$score(i)=\frac{f(j)x_i^j+f(k)x_i^k}{f(j)+f(k)}$$

donde *f*(*j*) es el valor de la función objetivo de la solución *j* y x_i^j es el valor de la *i-ésima* variable en la solución *j*. Estas puntuaciones se interpretan como probabilidades de que la variable tome el valor 1 ($P(x_i=1)=score(i)$). En términos de implementación, esto equivaldría a generar un número aleatorio *r* entre 0 y 1 para cada variable y hacer:

$$x_i=\begin{cases}1 & \text{si } r\le score(i)\\ 0 & \text{si } r> score(i)\end{cases}$$

Una variante determinista de este método consistiría en fijar un valor *r* en lugar de generarlo al azar. Para ilustrar el método consideremos que las soluciones cuarta y quinta de *RefSet* son las mostradas en la tabla 3 y que tienen un valor de *ObjVal*(4)=44 y *ObjVal*(5)=43. Si tomamos *r*=0,5 para todas las variables, obtenemos la nueva solución (0,1,1,1,1,0,0,0,1,0) con un valor de 44 y un peso total de 100.

Sol.	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
4	0	1	1	1	1	0	0	0	1	0
5	1	0	1	1	0	0	0	0	1	1
Score	0.49	0.51	1	1	0.51	0	0	0	1	0.49

Tabla 3. Combinación de Soluciones

Este mecanismo puede producir soluciones infactibles a partir de dos soluciones posibles (basta considerar r=0,4 en el ejemplo anterior para obtener una solución con todas las variables igual a 1). Esto no es en principio un problema, puesto que a todas las soluciones que resultan de la combinación se les aplica la rutina de mejora que, como hemos visto, las hace factibles.

Una vez realizadas todas las combinaciones estudiamos la inclusión de cada solución resultante en *RefSet* (actualización estática). Para ello comparamos cada solución con la peor de *RefSet*, y si la mejora entonces la reemplaza. Una vez realizadas las actualizaciones necesarias, volvemos a la fase de combinación, en la que consideramos todas las parejas de soluciones de *RefSet* no combinadas anteriormente; es decir, todas las parejas en las que al menos una solución ha entrado en *RefSet* en la última iteración. Este proceso de combinación / mejora / actualización de *RefSet* se repite hasta que al tratar de combinar, no encontramos ninguna pareja nueva (por no haber entrado ninguna solución en *RefSet* en la iteración anterior). En este momento el algoritmo finaliza su ejecución.

Una alternativa a este diseño básico consiste en no detener el algoritmo cuando se agotan las combinaciones, sino reconstruir *RefSet*. Una implementación habitual mantiene la mitad de las soluciones con mejor valor en *RefSet*, y reemplaza la mitad con peor valor por otras nuevas que introduzcan diversidad. Esto se puede hacer creando un nuevo conjunto *P* y aplicando el mismo mecanismo max-min sobre las distancias que el realizado en la inicialización del método. Una vez reconstruido *RefSet*, se procede de

nuevo a aplicar el mecanismo de combinación y actualización como se ha descrito anteriormente.

5. Conclusiones

Scatter Search es un método evolutivo que se encuentra en desarrollo. Sus orígenes se pueden situar en la década de los años 70 y, aunque es menos conocido que los algoritmos genéticos, se está aplicando con gran éxito en la resolución de numerosos problemas difíciles de optimización.

En la actualidad no existe un esquema único para aplicar *Scatter Search*. En este trabajo hemos tratado de introducir aquellos aspectos básicos del método que son ampliamente conocidos, así como revisar las últimas implementaciones realizadas y las cuestiones que son actualmente objeto de estudio. Además, hemos incluido un ejemplo para ilustrar las técnicas descritas y mostrar algunos de los detalles de implementación.

Referencias

[1] Campos V., Laguna M. y Martí R., *Scatter Search for the Linear Ordering Problem*, New Ideas in Optimisation, D. Corne, M. Dorigo and F. Glover (Eds.), McGraw-Hill. 331-341. 1999

[2] Campos V., Laguna M. y Martí R., *Context-Independent Scatter and Tabu Search for Permutation Problems*, Technical report TR03-2001, Departamento de Estadística e I.O., Universidad de Valencia. 2001

[3] Campos, V., F. Glover, M. Laguna y R. Martí, *An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem*, Journal of Global Optimization, 21, 397-414, 2001.

[4] Corberán A., E. Fernández, M. Laguna y R. Martí, *Heuristic Solutions to the Problem of Routing School Buses with Multiple Objectives*, Journal of the Operational Research Society**,** 53 (4), 427 - 435, 2002.

[5] Glover, F., *Heuristics for integer programming using surrogate constraints,* Decision Sciences 8, 156-166, 1977.

[6] Glover, F., *Tabu search for non-linear and parametric optimisation (with links to genetic algorithms),* Discrete Applied Mathematics 49, 231-255, 1994.

[7] Glover, F., *A Template for Scatter Search and Path Relinking*, in Artificial Evolution, Lecture Notes in Computer Science 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.), Springer-Verlag, 13-54, 1998.

[8] Glover, F., M. Laguna y R. Martí, *Scatter Search*, to appear in Theory and Applications of Evolutionary Computation: Recent Trends, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag. 1999.

[9] Glover, F., M. Laguna y R. Martí, *Fundamentals of Scatter Search and Path Relinking*, Control and Cybernetics, 29 (3), 653-684, 2000.

[10] Laguna M. y Martí R., *Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions*. Technical report TR11-2000, Departamento de Estadística e I.O., Universidad de Valencia, 2000.

[11] Laguna M. y Martí R., *The OptQuest Callable Library,* Optimization Software Class Libraries, Voss and Woodruff (Eds.), Kluwer Academic Publishers, Boston, 193-215, 2000.

[12] Laguna M. y Martí R., Scatter Search: Methodology and implementations in C, Kluwer Academic Publishers, Boston, 2003.

[13] Martello, S. y P. Toth, Knapsack Problems, John Wiley and Sons, Ltd., New York, 1989.

[14] Ugray Z., L. Lasdon, J. Plummer, F. Glover, J. Kelly y R. Martí, *A Multistart Scatter Search Heuristic for Smooth NLP and MINLP Problems*, Technical report, University of Texas at Austin, 2001