Revista Iberoamericana de Inteligencia Artificial

Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
Asociación Española para la Inteligencia Artificial
revista@aepia.org
ISSN (Versión impresa): 1137-3601
ISSN (Versión en línea): 1988-3064
ESPAÑA

2003
Armando Blanco / David A. Pelta / José L. Verdegay
FANS:UNA HEURÍSTICA BASADA EN CONJUNTOS DIFUSOS PARA PROBLEMAS DE OPTIMIZACIÓN
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, año/vol. 7, número 019
Asociación Española para la Inteligencia Artificial
Valencia, España

FANS: A fuzzy adaptive neighborhood search method for optimization problems

Armando Blanco, David A. Pelta, José L. Verdegay

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada
18071 GRANADA, Spain

e-mail: armando@ugr.es, dpelta@ugr.es, verdegay@ugr.es

The mix between basic ideas from the Fuzzy Set and Systems area with simple optimization methods give raise to a fuzzy sets-based heuristic called FANS, by Fuzzy Adaptive Neighborhood Search.

A very remarkable feature of the method is its ability to capture the qualitative behavior of other neighborhood search methods, what is achieved through appropriated handling of the membership function defining the so called *fuzzy valuation*. Here, we propose to review the main concepts of FANS and to describe four applications of the algorithm to problems of quite different domains.

Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.19 (2003),pp. 103-122
ISSN: 1137-3601. © AEPIA (http://www.aepia.org/revista).

FANS: una Heurística basada en Conjuntos Difusos para Problemas de Optimización

Armando Blanco
Depto. de C.C. e I.A
Universidad de Granada
18071 Granada, Spain
armando@ugr.es

David A. Pelta
Depto. de C.C. e I.A
Universidad de Granada
18071 Granada, Spain
dpelta@ugr.es

José L. Verdegay
Depto. de C.C. e I.A
Universidad de Granada
18071 Granada, Spain
verdegay@ugr.es

Resumen

La combinación entre ideas básicas del área de conjuntos y sistemas difusos, y métodos simples de optimización, dan lugar a una nueva heurística basada en conjuntos difusos, llamada FANS (siglas en inglés de Fuzzy Adaptive Neighborhood Search). Un elemento destacable del método es su capacidad de reflejar el comportamiento cualitativo de otros métodos de búsqueda por entornos, lo cual se consigue utilizando definiciones adecuadas para la denominada "valoración difusa", uno de los componentes claves de FANS. En este trabajo nos proponemos revisar los conceptos esenciales del método y reseñar su aplicación a cuatro problemas de dominios diferentes.

1. Introducción

El diseño, la construcción y la búsqueda de algoritmos que permitan resolver problemas que surgen en la vida real, son objetivos esenciales de las Ciencias de la Computación. A pesar de la alta complejidad que suelen presentar dichos problemas, deben ser resueltos pues son de notable importancia. Ambos aspectos, dificultad e importancia, han potenciado el desarrollo de técnicas heurísticas que, aunque pueden llevar a la obtención de soluciones sub-óptimas, son capaces de resolver el problema en términos de "satisfacción" del usuario. En general, estos métodos obtienen soluciones "buenas" en términos de cierta función de costo, pero también es posible que, además, el usuario considere otras características que en ocasiones pueden ser de naturaleza subjetiva y por lo tanto, modelizables mediante conjuntos difusos.

En las últimas décadas se produjo un flujo de información importante desde áreas clásicas, como la Investigación Operativa o la Teoría de Control, hacia el área de los Conjuntos y Sistemas Difusos y que permitió obtener resultados verdaderamente fructíferos. Hoy en día, términos como "Control Difuso", "Programación Matemática Difusa", "Sistemas basados en reglas difusas", etc, resultan familiares a cualquier investigador. Para ver ejemplos de esta interacción, el lector interesado puede referirse a [18, 13] y también a la recopilación bibliográfica de Cordón et al. [10], donde se presenta una buena muestra de la combinación de conjuntos y lógica difusa con algoritmos genéticos. Además en [28] se describen aplicaciones exitosas de heurísticas para optimización basadas en conjuntos difusos.

Siguiendo en esta dirección, mostraremos como un método clásico de optimización combinatoria se puede combinar con elementos básicos del área de los conjuntos y sistemas difusos para dar lugar a una herramienta de optimización adaptativa y robusta que permita

abordar la resolución de problemas complejos de forma satisfactoria. Para ello, se plantea como objetivo la presentación de las ideas esenciales de un método de búsqueda por entornos, adaptativo y difuso, denominado *FANS* por las iniciales de su nombre en inglés: *Fuzzy Adaptive Neighborhood Search*. *FANS*, incorpora como elementos destacables la utilización de una valoración difusa o subjetiva para evaluar las soluciones, y el uso de varios operadores para explorar el espacio de búsqueda.

La motivación para la utilización de la valoración subjetiva, parte de la idea que en ocasiones se suele hablar de soluciones *Diferentes, Similares, Razonables*, etc, siendo estas características de naturaleza subjetiva o vaga. Por lo general, las reglas de transición entre soluciones en los métodos clásicos de búsqueda local son de naturaleza crisp, es decir, consideran a una solución como "mejor" o no, "diferente" o no, etc. En el contexto de *FANS*, las soluciones son vistas como elementos de un conjunto difuso[1] cuya función de pertenencia es dicha valoración difusa, la cual representa cierta propiedad denominada genéricamente P. De esta forma, las soluciones tienen asignado un grado de pertenencia a dicho conjunto (por ejemplo, el conjunto de soluciones P = *aceptables*) lo cual permite definir subconjuntos de soluciones del tipo *muy* P, *algo* P, *no* P, etc. A través de esta valoración difusa, el comportamiento de *FANS* puede ser modificado de forma tal que se logra reflejar el comportamiento cualitativo de otros métodos de búsqueda local. De esta manera, veremos que *FANS* es en realidad un "framework" o molde de métodos de búsqueda local.

La utilización de varios operadores en el proceso de búsqueda es un área relativamente poco explorada, pero con una justificación muy simple: que una solución sea localmente óptima bajo cierto vecindario (inducido por un operador), no implica que dicha condición de optimalidad se verifique en otro vecindario[2]. Esta idea de variación del vecindario a explorar, también se utiliza en una variante muy poco investigada de *variable neighborhood search* (VNS)[15] denominada *variable neighborhood descent* [16].

Siendo además un método de búsqueda por entornos, resulta simple de comprender, implementar y mejorar, y por lo tanto resulta atractivo como herramienta para realizar las primeras aproximaciones para la resolución de un problema dado.

En este artículo se detallan las tareas realizadas para alcanzar el objetivo descrito, utilizando la siguiente estructura: en la Sección 2 se describen los elementos básicos de *FANS* y se presenta el esquema general del algoritmo. Posteriormente en la Sección 3 se describe la motivación y utilidad de la valoración difusa y se muestra como este componente puede ser utilizado para variar el comportamiento del algoritmo, dando lugar a un "framework" de métodos de búsqueda por entornos. La Sección 4 esta dedicada a verificar la utilidad del segundo elemento novedoso de *FANS*: el uso de varios operadores en el proceso de búsqueda. Luego, en la Sección 5, se describen la aplicación y resultados obtenidos mediante *FANS* sobre cuatro problemas de optimización. Finalmente, se presentan las conclusiones en la Sección 6.

2. Presentación de *FANS*

FANS se define como un método de búsqueda por entornos, adaptativo y difuso. Es un método de búsqueda por entornos porque el algoritmo realiza transiciones desde una solución de referencia a otra de su entorno, produciendo "trayectorias" o caminos. Es adaptativo porque su comportamiento varía en función del estado de la búsqueda y, finalmente, es considerado difuso porque las soluciones son evaluadas, además de por la función objetivo, mediante una *valoración difusa* que representa algún concepto subjetivo o abstracto.

FANS está basado en cuatro componentes principales: un operador, para construir soluciones; una valoración difusa, para cualificarlas; un administrador de operación, para adaptar el comportamiento o características del operador; y un administrador de vecindario, para generar y seleccionar una nueva solución.

[1] Los elementos básicos de conjuntos difusos y variables lingüísticas no serán descritos aquí. El lector interesado puede referirse a [11, 30].

[2] Salvo que esta solución sea el óptimo global.

En esta sección se describen con más detalle los elementos principales del algoritmo para lo cual se utilizarán las siguientes convenciones: $s_i \in \mathcal{S}$ es una solución del espacio de búsqueda, $\mathcal{O}_i \in \mathcal{M}$ es un operador de Modificación o de Movimiento del espacio de operadores; $\mathcal{P}$ representa el espacio de parámetros (tuplas de valores) y $\mathcal{F}$ representa el espacio de los conjuntos difusos cuyos elementos se denotan como $\mu_i()$.

Con estas definiciones, *FANS* queda especificado con la 7-tupla siguiente:

$$Fans(\mathcal{NS}, \mathcal{O}, \mathcal{OS}, \mu(), Pars, (cond, accion))$$

donde $\mathcal{NS}$ es el *Administrador de Vecindario*, $\mathcal{O}$ es el operador utilizado para construir soluciones, $\mathcal{OS}$ es el *Administrador de Operación* y $\mu()$ es una valoración difusa.

Además, $Pars$ representa un conjunto de parámetros y el par $(cond, accion)$ es una regla de tipo $IF\ cond\ THEN\ accion$ que se utiliza para detectar y actuar en consecuencia cuando la búsqueda se ha estancado.

A continuación se describen las características principales de cada componente y posteriormente se presenta el esquema del algoritmo.

2.1. Operador de Modificación

Dada una solución de referencia $s \in \mathcal{S}$, el operador de modificación $\mathcal{O}_i \in \mathcal{M}$, con $\mathcal{O}_i : \mathcal{S} \times \mathcal{P} \rightarrow \mathcal{S}$, construye nuevas soluciones s_i a partir de s.

Cada aplicación del operador sobre la misma solución s debe devolver una solución diferente. Es decir, debe existir algún elemento de aleatoriedad en su definición. Un requerimiento adicional es que el operador provea algún elemento o parámetro adaptable $t \in \mathcal{P}$ para controlar su comportamiento. Utilizaremos $\mathcal{O}^t$ para referirnos al operador con ciertos parámetros t.

2.2. Valoración Difusa

En el contexto de *FANS* las soluciones son evaluadas (además de la función objetivo) en términos de la *valoración difusa*. La motivación de este aspecto parte de la idea que en ocasiones se suele hablar de soluciones *Diferentes, Similares, Razonables*, etc, siendo estas características de naturaleza subjetiva o vaga.

Una forma de modelizar adecuadamente este tipo de conceptos vagos se logra representando la valoración difusa mediante un conjunto difuso $\mu() \in \mathcal{F}, \text{con } \mu : \mathcal{R} \rightarrow [0, 1]$. Como sinónimo de valoración difusa, son aceptables términos como *concepto difuso* o *propiedad difusa*, lo que permite hablar de soluciones que verifican dicha propiedad en cierto grado.

Por lo tanto, la valoración difusa nos permite obtener el grado de pertenencia de la solución al conjunto difuso representado por $\mu()$. Por ejemplo, teniendo el conjunto difuso de las soluciones "buenas", consideraremos el grado de bondad de la solución de interés. Así, dadas dos soluciones $a, b \in \mathcal{S}$ podemos pensar en cuan *Similar* es a con b, o cuan *Cerca* están, o también cuan *Diferente* es b de a. *Similar, Cerca, Diferente* serán conjuntos difusos representados por funciones de pertenencia $\mu()$ apropiadas.

2.3. Administrador de Operación

FANS utiliza varios operadores de modificación en el proceso de búsqueda y el Administrador de Operación $\mathcal{OS}$ es el responsable de definir el esquema de adaptación u orden de aplicación de los diferentes operadores utilizados.

El cambio de operador, o lo que es lo mismo, la ejecución del administrador, puede realizarse en cada iteración, o cuando el estado de la búsqueda así lo requiera. Esta adaptación puede estar basada en estadísticas del algoritmo tales como cantidad de evaluaciones de la función de costo realizadas, o teniendo en cuenta medidas particulares del proceso de búsqueda, como por ejemplo el número de iteraciones sin cambios en la mejor solución hallada, etc.

Básicamente se sugieren dos posibilidades para

el Administrador. En la primera, utilizada en este trabajo, el administrador $\mathcal{OS}$ ajustará los parámetros del operador y en consecuencia, devolverá un operador cuyo comportamiento será diferente: $\mathcal{OS}$ $(\mathcal{O}^{ti}) \Rightarrow \mathcal{O}^{tj}$.

La segunda opción, consiste en disponer de una familia de operadores de modificación $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k\}$. Bajo esta situación, $\mathcal{OS}$ podría definir el orden de aplicación de los mismos.

Dado que cada operador implica un vecindario diferente, se obtendrá un esquema que recuerda al de VNS [15]. Sin embargo, nuestro esquema es diferente al de VNS ya que *FANS* no utiliza un método explícito de búsqueda local ni requiere una métrica de distancia entre soluciones como la necesaria en VNS.

2.4. Administrador de Vecindario

Este componente es el responsable de generar y seleccionar una nueva solución del vecindario. Podemos verlo como una función que dadas una solución, una valoración difusa, un operador y un conjunto de parámetros, retorna una solución. Es decir:

$$\mathcal{NS} : \mathcal{S} \times \mathcal{F} \times \mathcal{M} \times \mathcal{P} \Rightarrow \mathcal{S}$$

En *FANS* se utilizan dos tipos de vecindarios: el *operativo* y el *semántico*, definidos respecto a cierta solución de referencia s.

Dados el operador $\mathcal{O}$ y la solución actual s, se define el vecindario operativo como:

$$\mathcal{N}(s) = \{\hat{s}_i \mid \hat{s}_i = \mathcal{O}_i(s)\} \qquad (1)$$

donde $\mathcal{O}_i(s)$ indica la i-ésima aplicación de $\mathcal{O}$ sobre s.

Para obtener el *vecindario semántico* de s se utiliza la valoración difusa $\mu()$, dando lugar a la siguiente definición:

$$\hat{\mathcal{N}}(s) = \{\hat{s}_i \in \mathcal{N}(s) \mid \mu(\hat{s}_i) \geq \lambda\} \qquad (2)$$

Es decir, $\hat{\mathcal{N}}(s)$ representa el λ-corte del conjunto difuso de soluciones representado por $\mu()$. En otras palabras, las soluciones de interés serán aquellas que satisfagan la valoración con, al menos, cierto grado λ.

La operación del administrador es simple: primero se ejecuta un *generador* para obtener soluciones del vecindario semántico a partir de varias aplicaciones del operador de modificación $\mathcal{O}$. Posteriormente, el procedimiento *selector* debe decidir cuál de estas soluciones retornar teniendo en cuenta: los grados de pertenencia de dichas soluciones, su costo o una combinación de ambos valores.

Por ejemplo, si estuviéramos utilizando una valoración difusa de "Similaridad" con respecto a la solución actual, el selector podría utilizar reglas de selección como las siguientes:

- *Mejor:* Devolver la solución más similar disponible,
- *Primera:* Devolver la primera solución suficientemente similar.

Naturalmente, también podría utilizarse el costo de esas soluciones similares para obtener reglas de selección como:

- *MaxMax:* De las soluciones similares, retornar la de mayor costo,
- *MaxMin:* De las soluciones similares, retornar la de menor costo.

2.5. Parámetros Globales

FANS mantiene un conjunto de parámetros globales $Pars \in \mathcal{P}$ para llevar el registro del estado de la búsqueda.Este conjunto es utilizado por varios componentes para decidir las acciones a tomar.

2.6. Manejo de óptimos locales

Los métodos de búsqueda local presentan como principal inconveniente, el quedar atrapados en óptimos locales. En *FANS* se proveen dos mecanismos para tratar con este problema.

El primero está contenido en la variación del operador en las etapas de búsqueda y

```
Procedimiento FANS:
Begin
  /* O : el operador de modificacion */
  /* μ() : la valoracion difusa */
  /* S_cur, S_new : soluciones */
  /* NS: el administrador de vecindario */
  /* OS: el administrador de operacion */
  Inicializar Variables();
  While ( not-fin ) Do
    /* Ejecutar Admin. de Vecindario NS */
    S_new = NS(S_cur, μ(), O);
    If (S_new es ''buena'' en base a μ()) Then
      S_cur := S_new;
      adaptar-Valoracion-Difusa(μ(), S_cur);
    Else
      /* Fue imposible hallar una S_new ''buena''
       con el operador O :. Sera modificado */
      O := OS-ModificarOperador(O);
    endIf
    If (HayEstancamiento?()) Then
      Escape();
    endIf
  endDo
End.
```

Figura 1: Esquema de *FANS*

se describirá en la sección 4. El segundo mecanismo de escape esta basado en el par $(cond, accion)$ donde $cond$, llamada $HayEstancamiento?() : \mathcal{P} \rightarrow [True, False]$, se utiliza para determinar cuando hay suficiente evidencia de que la búsqueda esta definitivamente estancada. Cuando $cond$ se verifique, entonces se ejecutará la acción $accion = Escape()$. Por ejemplo, se podría reiniciar el algoritmo desde una nueva solución inicial, reiniciar desde una solución obtenida a partir de una modificación especial de la actual, o cualquier otra opción que se considere adecuada.

2.7. El Algoritmo

El esquema de *FANS* se muestra en la Fig. 1. Cada iteración comienza con una llamada al administrador de vecindario $\mathcal{NS}$ con los siguientes parámetros: la solución actual S_{cur}, la valoración difusa $\mu()$ y el operador de modificación $\mathcal{O}$. Como resultado de la ejecución de $\mathcal{NS}$ pueden ocurrir 2 cosas: se pudo encontrar una solución vecina aceptable S_{new} (en términos de $\mu()$), o no se pudo.

En el primer caso, S_{new} pasa a ser la solución actual y los parámetros de $\mu()$ son adaptados. Por ejemplo, si $\mu()$ representa la similaridad respecto a la solución actual, entonces la valoración difusa debe ser adaptada para reflejar este cambio en la solución de referencia.

Si $\mathcal{NS}$ no pudo retornar una solución aceptable, es decir, en el vecindario inducido por el operador no se pudo encontrar una solución suficientemente buena, entonces se aplica el nuevo mecanismo de escape: se ejecuta el administrador de operación OS el cual retornara una versión modificada del operador $\mathcal{O}$. La próxima vez que $\mathcal{NS}$ se ejecute, dispondrá de un operador modificado para buscar soluciones, y por lo tanto es posible que el resultado sea diferente.

La condición $HayEstancamiento?()$ será verdadera cuando (por ejemplo) se hayan realizado $Tope$ llamadas a OS sin haber obtenido mejoras en la solución actual. Es decir, se probaron varios operadores y no se obtuvieron mejoras. En este caso, se ejecutará el procedimiento $Escape()$, se evaluará el costo de la nueva solución y se adaptará la valoración difusa $\mu()$. Posteriormente, se reiniciará la búsqueda.

Debe quedar claro que lo que varía en cada iteración son los parámetros utilizados en las llamadas a $\mathcal{NS}$. El algoritmo comienza con $\mathcal{NS}\ (s_0, \mathcal{O}^{t_0}, \mu_0)$. Si $\mathcal{NS}$ puede retornar una solución aceptable, entonces en la siguiente iteración la llamada será $\mathcal{NS}\ (s_1, \mathcal{O}^{t_0}, \mu_1)$; es decir cambia la solución actual y por lo tanto se modifica la valoración difusa. Si en cierta iteración l, $\mathcal{NS}$ no puede retornar una solución aceptable, entonces se ejecutará el administrador de operación el cual devolverá una versión modificada del operador. La iteración siguiente, $\mathcal{NS}$ será llamado con $\mathcal{NS}\ (s_l, \mathcal{O}^{t_1}, \mu_l)$.

Durante la ejecución del algoritmo pueden ocurrir dos situaciones problemáticas: primero, se realizaron varias llamadas $\mathcal{NS}(s_j, \mathcal{O}^{t_i}, \mu_j)$ con $i \in [1, 2, \ldots, k]$, lo cual significa que se probaron k formas diferentes de obtener una solución aceptable y ninguna tuvo éxito; o segundo, la búsqueda se está moviendo entre soluciones aceptables pero en las ultimas m llamadas no se consiguió mejorar la mejor solución de todas las visitadas hasta el momento.

Cuando cualquiera de las dos situaciones ocurra, se ejecutará el procedimiento $Escape()$ para el cual se plantean las dos posibilidades siguientes: $\mathcal{NS}\ (\hat{s}_0, \mathcal{O}^{t_j}, \hat{\mu}_0)$ o $\mathcal{NS}\ (\hat{s}_0, \mathcal{O}^{t_0}, \hat{\mu}_0)$, donde $\hat{s}_0$ es una nueva solucion inicial (generada aleatoriamente, por ejemplo), y $\hat{\mu}_0$ es la valoración difusa obtenida en función de $\hat{s}_0$. Respecto al operador, se puede mantener con los mismos parámetros ($\mathcal{O}^{t_j}$) o también se lo puede "reinicializar" ($\mathcal{O}^{t_0}$).

El algoritmo finaliza cuando el número de evaluaciones de la función de costo alcanza cierto límite o cuando se realizó un número predeterminado de evaluaciones.

3. Sobre la Utilidad de la Valoración Difusa

El comportamiento global de *FANS* depende de la definición de sus componentes y de la interacción entre ellos. A través de la utilización de diferentes definiciones y parámetros para la valoración difusa se consigue que el algoritmo se comporte de diferente manera, lo que da lugar, en consecuencia, a variaciones en los resultados que se puedan obtener.

Antes de seguir adelante, es necesario proveer una definición para el administrador de vecindario. Por simplicidad, utilizaremos un esquema denominado $First$ que funciona de la siguiente manera: dada una solución s, y un operador $\mathcal{O}$, el procedimiento *generador* del Administrador obtendrá soluciones del vecindario operativo, las cuales serán analizadas por el *selector* utilizando la valoración difusa $\mu()$, y un nivel mínimo de calidad requerido λ.

La regla de selección empleada simplemente establece que se devolverá la primera solución analizada $\hat{s}$ que verifique $\mu(s, \hat{s}) \geq \lambda$. A lo sumo se utilizarán cierto número de intentos y este valor debe especificarse en cada caso.

La valoración difusa representará cierta noción de "Aceptabilidad" y asumiremos un problema de minimización.

Teniendo en mente el esquema de *FANS* mostrado en la Fig. 1, discutiremos en esta sección como el algoritmo puede ser adaptado para reflejar el comportamiento cualitativo de otros métodos clásicos de búsqueda local [5].

Ejemplo 1: Caminos Aleatorios

Para obtener este comportamiento, todo lo que se necesita es considerar cualquier solución del vecindario como aceptable. Por lo tanto, utilizando una valoración difusa con $\mu(s, \hat{}) = 1\ \forall\ \hat{s} \in \mathcal{N}(s)$, cualquier solución del vecindario operativo tendrá la opción de ser seleccionada. De esta manera, *FANS* se comportará como un método que produce caminos aleatorios.

Ejemplo 2: Hill Climbing

En este caso, para obtener el comportamiento deseado es necesario utilizar una valoración difusa tal que $\mu(s, \hat{s}) = 1$ si $f(\hat{s}) < f(s)$ y asignar $\lambda = 1$. De esta manera, únicamente serán consideradas como aceptables aquellas soluciones que mejoren el costo actual. El método de Hill Climbing termina cuando se alcanza un óptimo local. En el contexto de *FANS*, la obtención de un método multi-arranque es directo.

Ejemplo 3: Recocido Simulado

En el método de Recocido Simulado, la aceptación de soluciones esta gobernada por un parámetro denominado "Temperatura". Las soluciones que mejoran el costo actual siempre son aceptadas. Aquellas que lo empeoran también pueden ser aceptadas con cierta probabilidad, que será alta al inicio de la ejecución. A medida que la búsqueda progresa, la temperatura disminuye, decrementando entonces la probabilidad de aceptación de soluciones peores. Hacia el final de la ejecución, solo las soluciones mejores son aceptadas.

Para reflejar este comportamiento mediante la valoración difusa, se puede utilizar la siguiente definición de "Aceptabilidad":

$$\mu(\hat{s}, s) = \begin{cases} 0 & \text{si } f(\hat{s}) > \beta \\ \frac{\beta - f(\hat{s})}{\beta - f(s)} & \text{si } f(s) \leq f(\hat{s}) \leq \beta \\ 1 & \text{si } f(\hat{s}) < f(s) \end{cases}$$

donde f es la función objetivo, s es la solución actual y $\hat{s}$ es una solución del vecindario operativo.

El valor β representa el límite para lo que se considera aceptable. Este valor es clave ya que en última instancia determina que soluciones pertenecen o no al vecindario semántico. Si se define β como una función $h(s, \hat{s}, t)$ donde t es un parámetro que represente por ejemplo el número actual de iteraciones, el límite de deterioro aceptable puede ser reducido a medida que la búsqueda progresa. Hacia el final de la ejecución, solo aquellas soluciones mejores que la actual serán tenidas en cuenta. Un ejemplo para la función h es:

$$h = f(s) * (1 + \frac{1}{1 + e^{(f(s)-f(\hat{s}))/t}}) \qquad (3)$$

donde el segundo término de la suma representa la distribución de probabilidad de Boltzmann.

En términos más generales, β representan el umbral de aceptabilidad y por lo tanto, su variación está reflejando una clase de algoritmos más amplia, los algoritmos de umbral.

FANS como heurística de propósito general

En los párrafos anteriores mostramos como *FANS* puede ser ajustado para reflejar el comportamiento cualitativo de otras técnicas, principalmente a través de la manipulación de la valoración difusa. En consecuencia, y como se había planteado, podemos decir que *FANS* representa un "framework" para métodos simples de búsqueda local.

Sin embargo, *FANS* puede plantearse como una heurística en sí misma capaz de mostrar una variedad de comportamientos muy amplia. Este potencial queda claro si consideramos que la valoración difusa representa el criterio de un decisor buscando soluciones para un problema.

Por ejemplo, en la Fig. 2 aparecen tres criterios de aceptabilidad. Las definiciones se basan en el costo de cierta solución actual, indicado en los gráficos por una línea de puntos.

La definición triangular (a), representa un decisor buscando soluciones similares en costo

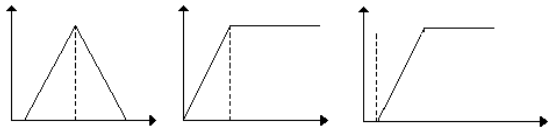


Figura 2: Ejemplos de Valoraciones Difusas.

pero diferentes en estructura. Este tipo de comportamiento puede aparecer cuando las soluciones presentan características difíciles de cuantificar (por ejemplo, aspectos estéticos) con lo cual el decisor desea obtener un conjunto de opciones de costo similar para luego hacer su elección en términos de otros criterios diferentes al costo. La definición central (b) representa un decisor "insatisfecho" con la solución actual. Cualquier opción que la mejore, tendrá un valor máximo de aceptabilidad. Finalmente la tercera opción (c), representa un decisor "conservador" en el sentido que para cambiar la solución actual, la mejora obtenida debe ser considerable.

Otra forma de conseguir comportamientos diferentes, es mediante la aplicación de modificadores lingüísticos como *muy* o *algo* sobre una definición dada de la valoración difusa.

En síntesis, *FANS* brinda la posibilidad de variar su comportamiento y en última instancia las soluciones que se pueden obtener, de forma simple y comprensible. Si consideramos que cada comportamiento permite obtener resultados cuantitativamente diferentes, y que para cada problema un algoritmo puede resultar mejor que otro [3], la potencialidad de *FANS* como herramienta genérica de optimización resulta clara.

4. Sobre el Uso de Múltiples Operadores en la Búsqueda

Para concluir la presentación de *FANS*, en esta sección se analiza el otro elemento novedoso que incorpora el método: la utilización de varios operadores en el proceso de búsqueda

[3] Aunque a priori es imposible saberlo

lo cual sirve a dos propósitos: primero, como mecanismo de escape de óptimos locales, y segundo como mecanismo para potenciar la búsqueda local.

Para el primer aspecto se presentará una justificación intuitiva pero muy clara sobre el efecto que produce la utilización de varios operadores en el espacio de búsqueda. Posteriormente, se realizarán una serie de experimentos sobre instancias del problema de la mochila, para mostrar que un algoritmo que utiliza varios operadores en el proceso de búsqueda, permite obtener mejores resultados que otro que utiliza un único operador.

4.1. Beneficios en Escape de Óptimos Locales

En el contexto de *FANS*, si bien la definición del vecindario operativo es fija, la idea de cambiar el operador de modificación $\mathcal{O}$ es inducir un cambio en las soluciones que se pueden obtener a partir de cierta solución de referencia s dada.

Si pensamos en un problema con soluciones binarias, una solución actual s, un operador $\mathcal{A}$ que modifique k variables, y un operador $\mathcal{B}$ que modifique $k+1$ variables, entonces resulta que $\mathcal{N}_{\mathcal{A}}(s) \neq \mathcal{N}_{\mathcal{B}}(s)$. Un ejemplo trivial es el siguiente: supongamos que $s = 0000$, $\mathcal{A}$ modifica una variable y $\mathcal{B}$ modifica dos, entonces los vecindarios inducidos son: $\mathcal{N}_{\mathcal{A}}(s) = \{1000, 0100, 0010, 0001\}$ y $\mathcal{N}_{\mathcal{B}}(s) = \{1100, 1010, 1001, 0110, 0101, 0011\}$

Si imaginamos el espacio de búsqueda como un grafo, como se propone en [20], la situación se representa en la Fig. 3, donde cada nodo de ambos grafos representa un string binario de cuatro elementos. Ahora siendo el nodo $s = 0000$ y $\mathcal{A}$, $\mathcal{B}$, operaciones de modificación como las definidas anteriormente, el grafo de la izquierda representa las soluciones accesibles desde s mediante la operación $\mathcal{A}$. La accesibilidad queda reflejada por la existencia de aristas. Si cambiamos la operación por $\mathcal{B}$, obtenemos el grafo de la derecha, donde se han eliminado las aristas previas y se han dibujado las nuevas que conectan s con aquellos nodos en $\mathcal{N}_{\mathcal{B}}(s)$.

Por lo tanto, se observa que un cambio de operación modifica las aristas del grafo, o espacio de búsqueda, modificando el conjunto de soluciones accesibles. La consecuencia de este enfoque es clara: que una solución sea óptimo local bajo un operador, no implica que lo siga siendo cuando el operador se cambia, y es ésta la razón que justifica la utilización de varios operadores en el proceso de búsqueda.

4.2. Beneficios en la Optimización

En esta sección se describen los experimentos realizados para mostrar que un algoritmo que utiliza varios operadores en el proceso de búsqueda, permite obtener mejores resultados que otro que utiliza un único operador [4].

Para los experimentos se utilizó *FANS* como una búsqueda local simple con multi-arranque. Las pruebas se realizaron sobre 10 instancias del problema de la mochila: 5 instancias aleatorias de la versión con una única restricción (instancias ST) y 5 instancias de la versión con múltiples restricciones (instancias MR).

Se utilizó un administrador de vecindario $First$, el cual retorna la primera solución que mejore el costo de la actual, utilizando un máximo de intentos fijado en $maxTrials = n$, donde n es el tamaño de la instancia. La solución inicial se genera aleatoriamente con un único valor en 1.

Como operador de modificación se utilizó el k-BitFlip, el cual complementa el valor de k bits seleccionados al azar. Se implementaron dos administradores de operación $\mathcal{OS}$. El primero, denominado *Decremento*, hace $k_t = k_{t-1} - 1$ cada vez que el Administrador de Vecindario no pueda obtener una solución mejor que la actual utilizando un operador que modificaba k_{t-1} bits. Es decir, suponiendo $maxK = 3$, entonces el algoritmo comienza a progresar con 3-BitFlip. Cuando se estanca, sigue con 2-BitFlip y luego con 1-BitFlip. El segundo administrador, denominado *Incremento*, aumenta el valor de k: $k_t = k_{t-1} + 1$. En este caso, siempre se comienza con 1-BitFlip, se sigue con 2-BitFlip y así hasta $maxK$-BitFlip. Cuando el administrador de vecindario no pueda obtener una solución mejor con $k = 1$

[4] Naturalmente, esta afirmación no puede ser extrapolada directamente a otras situaciones diferentes de las planteadas en este experimento

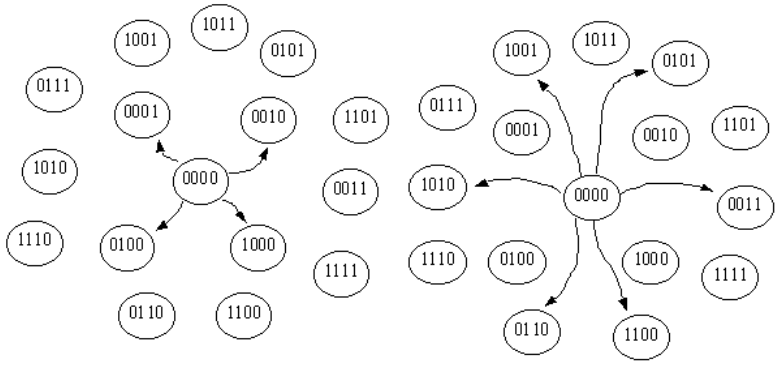


Figura 3: Efecto sobre el espacio de búsqueda ante un cambio de operador

o $k = maxK$, según se hagan decrementos o incrementos respectivamente, se ejecuta el procedimiento $Escape$, el cual genera una nueva solucion aleatoria. Posteriormente, el algoritmo se reinicia desde esta nueva solución.

El objetivo del experimento es mostrar que utilizar un valor de $maxK > 1$ resulta mejor que utilizar $maxK = 1$. La base de comparación son los resultados obtenidos con $k = 1$.

Por cada instancia, valor de $maxK = \{1, 2, 3, 4\}$, y administrador de operación $\mathcal{OS} = \{incremento, decremento\}$, se realizaron 30 ejecuciones del algoritmo, finalizando cada una cuando se agotaron las 15000 evaluaciones disponibles de la función de costo o se generaron 45000 soluciones. A continuación se presentan los resultados obtenidos.

4.2.1. Análisis de Resultados

Los resultados se analizaron en términos de la media de los errores agrupados por tipo de instancias y en forma global, para cada valor utilizado de $maxK$ y $\mathcal{OS}$. El error se calcula como:

$$error = 100 * \frac{ValRef - Valor\,Obtenido}{ValRef} \quad (4)$$

donde $ValRef$ es el óptimo de la instancia para los problemas de la mochila con múltiples restricciones (MR); y es la cota de Dantzig (mirar por ej. [24]) para la versión clásica (ST).

En la Tabla 1 se muestra la media del error para cada valor de $maxK$ y cada $\mathcal{OS}$, discriminado para las instancias ST y MR.

Se puede observar que tanto en forma global, como para cada tipo de instancia en particular, las versiones del algoritmo que usan $maxK > 1$ obtienen mejores resultados que los obtenidos con $maxK = 1$. Esto ocurre para los dos esquemas de adaptación de $maxK$ propuestos. También se verifica que para ambos $\mathcal{OS}$'s, un incremento en el valor de $maxK$ permite reducir el error global (indicado en la columna $Total$). Para el caso de decremento, resulta beneficioso comenzar las mejoras con un operador que cambie 4 bits. Cuando con 4 ya no se obtengan mejoras, se reduce a 3, luego a 2 y finalmente se realiza un "ajuste fino" con $k = 1$. La adaptación de k en sentido contrario también resulta beneficiosa, aunque para las instancias MR solo aparecen mejoras respecto a $k = 1$ y no entre los valores obtenidos cuando se utiliza $maxK > 1$.

Es interesante analizar cuanto contribuye cada operador a la obtención del resultado final. En la Figura 4 se muestran la cantidad de transiciones a soluciones aceptables (en promedio) que permitió obtener cada operador para cada valor de $maxK$. Los valores obtenidos se escalaron en el rango $[1, 100]$ para mejorar la interpretabilidad. Naturalmente cuando $maxK = 1$, todas las mejoras se obtuvieron con $k = 1$. Es a partir de $maxK = 2$ donde el análisis se vuelve interesante.

En los resultados correspondientes al $\mathcal{OS}$ que decrementa k, se observa que cuando $maxK = 2$, prácticamente un 95 % de la mejora se obtuvo

	Base	*Decremento maxK*			*Incremento maxK*		
Instancia	1	2	3	4	2	3	4
MR	9.01	2.19	1.95	1.92	3.98	3.85	3.90
ST	6.86	4.58	3.87	3.58	4.73	4.28	4.11
Total	7.94	3.39	2.91	2.75	4.36	4.06	4.01

Tabla 1: Medias del Error en función de $maxK$ y $\mathcal{OS}$ para instancias con una restricción (ST) y con múltiples restricciones (MR).

con 2-BitFlip y el restante porcentaje con $k = 1$. Cuando $maxK = 3$, el 80 % del progreso se debe al uso de 3-BitFlip. Del restante 20 %, casi toda la mejora se realiza con 2-BitFlip. Finalmente cuando $maxK = 4$, el operador 4-BitFlip es el responsable del 70 % de los pasos de mejora. La utilización de 3 y 2-BitFlip completa prácticamente la optimización aunque aparecen aproximadamente un 5 % de mejoras correspondientes a 1-BitFlip.

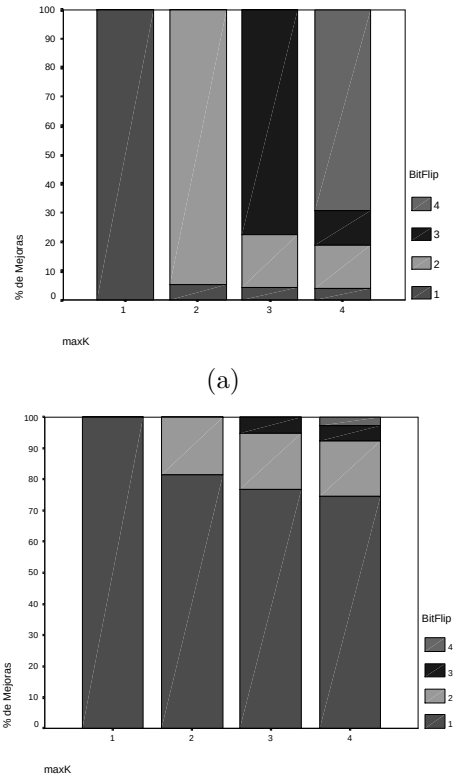
Para el caso de incrementos en k, se observa que para un valor de $maxK = 2$, hasta un 20 % de mejora se puede conseguir con 2-BitFlip una vez que la búsqueda se estancó con 1-BitFlip. Para $maxK = 3$ y 4, los gráficos muestran que hasta un 75 % de la mejora corresponde a 1-BitFlip, pero luego los operadores subsiguientes pueden mejorar los óptimos locales encontrados.

También se analizó la media de la cantidad de evaluaciones realizadas para encontrar el mejor valor, discriminada en función de $maxK$ y por tipo de instancia (resultados no mostrados).

Cuando se utiliza un esquema de decremento, para las instancias con múltiples restricciones se observa una reducción de los valores a medida que aumenta $maxK$. Para las instancias del problema clásico, esta tendencia no se verifica. En este caso, el valor más alto de evaluaciones se alcanza con $maxK = 2$. Luego aparecen los asociados a $maxK = \{1, 4\}$ mientras que el menor valor corresponde a $maxK = 3$.

Para el esquema de adaptación con incrementos de k, los resultados para MR indican que un aumento de $maxK$ permite, no sólo obtener mejores resultados, sino también de forma más rápida. Para las instancias ST, el valor de $e2b$ (media de la cantidad de evaluaciones realizadas para obtener la mejor solución) para $maxK = 1$ es el menor de todos, seguido por $maxK = 3$ y 4.

Naturalmente, esta medida del "esfuerzo" no se puede analizar en forma aislada sino teniendo en mente los resultados obtenidos en términos del error. Por lo tanto, creemos que las diferencias en la media del error pueden compensar un posible aumento en las evaluaciones necesarias.



(a)

(b)

Figura 4: Contribución promedio de cada operador BitFlip para cada valor de $maxK$. En (a) resultados con administrador de operación decremento y en (b) con incremento.

5. Ejemplos de Aplicación

En esta sección, revisaremos brevemente los principales resultados obtenidos con *FANS* sobre un conjunto de problemas de optimización, entre los cuales se encuentran el problema de la mochila simple y con múltiples restricciones, la minimización de funciones complejas y el problema de predicción de estructura de proteínas.

Para cada problema, describiremos brevemente como se definieron los componentes de *FANS* y resumiremos los principales resultados, indicando naturalmente las referencias donde se pueden encontrar las descripciones completas.

5.1. Mochila Clásica

El problema de la mochila se formula de la siguiente manera:

$$\text{Max.} \sum_{j=1}^{n} p_j * x_j \tag{5}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j * x_j \leq C,\ x_j \in \{0,1\}, j = 1, \ldots, n$$

donde n es el número de items, x_j indica si el ítem j está incluido o no en la mochila, p_j es el beneficio asociado al ítem j, $w_j \in [0, \ldots, r]$ es el peso del ítem j, y C es la capacidad de la mochila. Además, se asume que cada ítem cabe en la mochila ($w_j < C$, $\forall j$), y que el conjunto completo no cabe ($\sum_{j=1}^{n} w_j > C$).

A continuación se describen resultados presentados en [5].

5.1.1. Implementación de *FANS*

La aplicación de *FANS* a este problema es relativamente directa y simple. Las soluciones se representan mediante vectores binarios, lo cual permite utilizar un operador de modificación k-BitFlip, que simplemente cambia el valor de k posiciones seleccionados al azar.

La valoración difusa representa la noción de "Aceptabilidad" con la siguiente idea: aquellas soluciones que mejoran el costo actual tienen un grado de aceptabilidad mayor que aquellas que lo empeoran. Aquellas soluciones cuyo costo sea peor que cierto umbral, no se consideran como aceptables.

Siendo f la función objetivo, s la solución actual, $\hat{s} = \mathcal{O}(s)$ una nueva solución, y $\beta < f(s)$ el límite para lo que se considera aceptable, la función de pertenencia siguiente modeliza el comportamiento deseado:

$$\mu(\hat{s}, s) = \begin{cases} 0 & \text{si } f(\hat{s}) < \beta \\ \frac{(f(\hat{s})-\beta)}{(f(s)-\beta)} & \text{si } \beta \leq f(\hat{s}) \leq f(s) \\ 1 & \text{si } f(\hat{s}) > f(s) \end{cases}$$

En dicho trabajo se utilizó $\beta = f(s) * 0{,}95$.

Para adaptar el operador de modificación, el administrador de operación cambia el valor del parámetro k. Cada vez que se ejecuta el administrador, el valor de k se reemplaza por un nuevo valor entero $\hat{k}$, elegido aleatoriamente en el rango $[1, 2*k]$. Además, si $\hat{k} > n/10$, donde n es el tamaño de la instancia, entonces $\hat{k} = n/10$.

Para el administrador de vecindario, se utilizó un *Esquema de Agrupamiento basado en la Calidad* [7], el cual tiene en cuenta el grado de aceptabilidad de las soluciones para decidir cual debe ser devuelta. Los parámetros utilizados fueron $R = 5|S = 3|T = 1$ y $maxTrials = 25$

El administrador intenta generar R soluciones "Aceptables" mediante el operador $\mathcal{O}$ utilizando a lo sumo un número máximo de intentos $maxTrials$. Luego, las soluciones obtenidas se agrupan en S conjuntos difusos teniendo en cuenta los grados de aceptabilidad, y finalmente se devuelve 1 solución. El lector puede considerar el segundo paso como un proceso de clustering sencillo.

Los $S = 3$ conjuntos difusos o clusters se representan mediante funciones de pertenencia triangulares solapadas, cuyos extremos se ajustan al rango $[\lambda, 1{,}0]$, siendo $\lambda = 0{,}98$ el mínimo nivel que necesita una solución para ser considerada como aceptable. Los conjuntos representan los términos *Baja, Media, Alta* para la variable lingüística *Calidad*. Al final del

proceso se devuelve una única solución ($T = 1$). La regla de selección es:

Retornar alguna solución de las de máxima calidad disponible.

Si no existe ninguna solución con calidad suficiente, se generara una condición de excepción ya que no fue posible encontrar ningún vecino aceptable con los elementos disponibles.

5.1.2. Experimentos y Resultados

La calidad de *FANS* se comparó de forma empírica frente a dos implementaciones de algoritmos genéticos (AG) y a una de recocido simulado (SA).

Los AG's diferían en el operador de cruce: para uno se utilizó cruce de un punto ($GAop$) y para el otro, cruce uniforme ($GAux$). Como operador de mutación se utilizó el operador de modificación de *FANS*, donde el valor del parámetro k se elige aleatoriamente.

Los algoritmos se compararon sobre 45 problemas aleatorios, con $n = 100$. Se consideraron tres tipos de correlación entre el peso y el beneficio de los items, dando lugar a tres grupos de instancias: no correlacionadas (NC), con correlación débil (DC), y con correlación fuerte (FC). Cada grupo contiene 15 instancias.

Todos los algoritmos utilizaban operadores simples y cantidad limitada e igual de "recursos" para todos los algoritmos. Por recursos deben entenderse evaluaciones de la función de costo y cantidad máxima de soluciones a generar.

Los resultados se midieron en términos del error respecto a la cota de Dantzig y se muestran en la Tabla 2. Cada valor representa la media y la varianza de los errores sobre las 30 ejecuciones realizadas por cada uno de los 15 problemas en cada clase.

Los resultados de la Tabla 2, junto a resultados de t-test, mostraron que los errores obtenidos por *FANS* fueron significativamente menores que los de ambos AG en los tres tipos de instancias. También fueron mejores que los de SA para las instancias NC y FC, y similares para las DC.

Ambos AG obtuvieron resultados similares sobre las instancias DC y FC, pero $GAux$ fue mejor en el caso NC. Los errores obtenidos por SA resultaron menores que los de ambos AG's para las instancias NC y DC, aunque para las instancias FC, los tres algoritmos resultaron equivalentes.

El análisis de los resultados sobre el conjunto de 45 instancias de prueba, permitió verificar que *FANS* alcanzaba errores medios significativamente menores que SA, $GAop$ y $GAux$. Los siguientes algoritmos en el "ranking" fueron SA, que superó a ambos AG, seguido por $GAux$ que superó a $GAop$, principalmente por los resultados en las instancias NC.

Además, se pudo constatar que tanto *FANS*, SA y $GAop$, alcanzaron los errores medios más bajos en las instancias FC, contradiciendo la hipótesis que relaciona mayor correlación entre peso y beneficio, con mayor dificultad.

5.2. Mochila con Restricciones Múltiples

Esta es la clase más general de los problemas de mochila. Básicamente es un problema de programación lineal entera con la siguiente formulación:

$$\text{Max} \sum_{i=1}^{n} p_i * x_i \tag{6}$$

$$\text{s.t} \sum_{i=1}^{n} w_{ij} * x_i \leq C_j \text{ con } j = 1, \ldots, m$$

donde m es el número de restricciones, w_{ij} el costo del ítem j para la restricción i y los demás elementos son como en el problema clásico.

A continuación, se presenta la aplicación de *FANS* a este problema. La descripción completa puede verse en [8].

	NC		*DC*		*FC*	
Algoritmo	Media	Var.	Media	Var.	Media	Var.
FANS	1.08	0.55	1.75	1.08	0.84	1.06
SA	1.61	3.17	1.80	1.37	1.32	1.25
GAop	2.06	0.80	2.74	1.18	1.32	1.26
GAux	1.32	0.77	2.63	1.25	1.37	1.25

Tabla 2: Media y varianza de los errores por tipo de instancia calculados sobre 30 ejecuciones por cada uno de los 15 problemas de prueba.

5.2.1. Implementación de *FANS*

Las características de la implementación utilizada son prácticamente idénticas a las descriptas en la sección anterior. La diferencia reside en el administrador de vecindario utilizado. Además del esquema $R|S|T$, para este caso, se empleó el administrador $First$: dada una solución actual s y un nivel de aceptabilidad deseado,λ, el administrador intenta obtener una solución $\hat{s} \in \mathcal{N}(s)$ tal que $\mu(\hat{s}, s) \geq \lambda$. $First$ dispone de un número máximo de intentos para obtener $\hat{s}$.

5.2.2. Experimentos y Resultados

En esta aplicación se compararon los siguientes 5 algoritmos: *FANS* con el administrador $R|S|T$ (F_{rst}); *FANS* con el administrador $First$ (F_{ff}); un AG con cruce uniforme ($GAux$); un AG con cruce de un punto ($GAop$); y una implementación de recocido simulado (SA).

Los algoritmos se compararon bajo la hipótesis de nulo o mínimo conocimiento del problema y cantidad limitada e igual de recursos para todos los métodos.

Los experimentos se realizaron sobre 9 instancias, disponibles en [3], identificadas como *pb5, pb7, Weing1, Weing3, Weing7, Weish10, Weish14, Weish18, Weish27*. El número de items varía entre 20 y 105 conteniendo entre 2 y 30 restricciones.

Para cada instancia se ejecutó 30 veces cada algoritmo. Cada ejecución finalizó cuando se agotaron las $maxEvals = 15000$ evaluaciones de la función de costo disponibles o cuando se generaron $maxEvals * 4$ soluciones. Este límite se estableció porque solamente se evaluaban las soluciones factibles.

Como en el caso anterior, los resultados se analizaron en términos del error respecto al óptimo en cada problema y en forma global sobre todo el conjunto.

La Tabla 3 (a) muestra la media del error sobre las 30 ejecuciones en cada problema. Los resultados indican que ambas versiones de *FANS* tuvieron errores menores que los demás algoritmos en todos los casos, salvo Weing3, Weing7 y Weish27. *SA* alcanzó el mejor valor en Weing7 y *GAux* en los dos restantes.

En la Tabla 3 (b), se indica para cada algoritmo (mediante una x) aquellos problemas que resolvió de forma óptima en alguna de las 30 ejecuciones. Se observa que F_{rst}, F_{ff} y *GAux* obtuvieron el óptimo en 6 de 9 casos, mientras que *GAop* en 5 de 9 y *SA* solamente alcanzó el óptimo de un solo problema (este problema fue resuelto por todos los algoritmos).

El análisis de la media y la varianza de los errores sobre el conjunto de los problemas (resultados no mostrados), revelaron que globalmente ambas versiones de *FANS* alcanzaron los valores de error más bajos, seguidos por *GAux*. La media del error para *GAux* y *GAop* fue casi el doble que la obtenida por F_{rst} y F_{ff}, mientras que los valores de *SA* fueron 5 veces mayores.

Para confirmar si existían diferencias significativas en las medias del error se realizaron los correspondientes tests, los cuales confirmaron que ambas versiones de *FANS* resultaron mejores que *GAux*, *GAop* y *SA*. Ambos AG superaron a *SA*, y no se detectó diferencia significativa entre ambos.

	F_{rst}	F_{ff}	SA	$GAop$	$GAux$
pb5	1.04	0.92	6.52	3.37	3.07
pb7	0.94	1.19	4.13	3.83	4.23
weing1	0.20	0.19	8.07	0.92	1.37
weing3	1.54	1.32	22.04	1.85	0.91
weing7	0.50	0.51	0.48	1.13	0.93
weish10	0.27	0.14	1.22	1.34	1.18
weish14	0.85	0.78	1.93	1.85	0.91
weish18	0.73	0.71	1.39	0.95	0.89
weish27	3.02	2.89	2.85	3.21	1.18

(a)

	F_{rst}	F_{ff}	SA	$GAop$	$GAux$
pb5	x	x			
pb7	x	x			
weing1	x	x		x	x
weing3				x	x
weing7					
weish10	x	x	x	x	x
weish14	x	x		x	x
weish18	x	x		x	x
weish27					x

(b)

Tabla 3: En (a), media de los errores obtenidos por cada algoritmo sobre cada problema (30 ejecuciones). En (b), una x indica que el algoritmo de la columna obtuvo el óptimo del problema de la fila en algún intento.

5.3. Minimización de Funciones

En esta sección describiremos los resultados presentados en [6] donde se aplicó *FANS* en la minimización de funciones $f : \mathcal{R}^n \rightarrow \mathcal{R}$. El conjunto de pruebas utilizado presentaba casos de multi modalidad, separabilidad, etc. [2, 25].

5.3.1. Implementación de *FANS*

En este caso, las soluciones se representan como vectores de números reales, afectando por lo tanto la definición del operador de modificación y el correspondiente administrador de operación. Se mantienen la valoración difusa de "Aceptabilidad" y el administrador de vecindario $R|S|T$, aunque con diferentes parámetros.

El Operador de Modificación $Move$ producirá variaciones en algunas variables. Siendo s la solución actual con n variables $x_1 \dots x_n$, el operador $Move$ selecciona al azar $n/2$ variables y les aplica una perturbación aleatoria positiva o negativa. Para cada variable a modificar $x_i \in [min, max]$, se calcula un "intervalo de modificación" $[a_i, b_i]$ cuya amplitud es función de min, max (constantes) y dos parámetros m y $Tope$. Luego, el valor de la variable se reemplaza por un nuevo valor aleatoriamente en $[a_i, b_i]$.

El administrador de operación es el responsable de adaptar los parámetro m y $Tope$. La estrategia utilizada produce fases de expansión y contracción de los intervalos de modificación. A medida que la ejecución avanza, la expansión permitida es cada vez menor lo que provoca una concentración de la búsqueda alrededor de los valores actuales de las variables.

5.3.2. Experimentos y Resultados

FANS se comparó frente a un algoritmo genético de dominio público (AGB) [1] y a una implementación de SA sobre las las funciones de prueba de la Tabla 4, donde además de la definición, se indica el rango para las variables $x_i \in [min, max]$. Todas las funciones alcanzan el óptimo en $x^* = (0, \dots, 0)$ con $f(x^*) = 0$ y se utilizan $n = 25$ variables. Para ef_{10}, se empleó $n = 10$.

Los resultados obtenidos se muestran en la Tabla 5, donde AE es el número promedio de evaluaciones de la función de costo realizadas para alcanzar el mejor valor; AB es el promedio de los mejores valores hallados en cada una de las 25 ejecuciones; SD es la desviación estándar de los mejores valores encontrados; BF es el mejor de los valores encontrados en el conjunto de ejecuciones. Si aparece un valor porcentual, este indica el porcentaje sobre las 25 ejecuciones donde el óptimo fue alcanzado.

La Tabla 5 muestra que *FANS* superó claramente al AGB en todas las funciones. Este hecho fue confirmado mediante t-test con un nivel de significación del 95 %. Los test también confirmaron que *FANS* es mejor que SA en ef_{10}, f_{sph} y f_{sch}. No se encontró diferencia significativa en la media

$f_{sph}(\hat{x}) = \sum_{i=1}^{n} x_i^2$ [-5.12, 5.12]
$f_{ras}(\hat{x}) = 10.n + \sum_{i=1}^{n}[x_i^2 - 10 * cos(2\pi x_i)]$ [-5.12, 5.12]
$f_{ros}(\hat{x}) = \sum_{i=1}^{n-1}(100 * (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ [-5.12, 5.12]
$f_{gri}(\hat{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$ [-600,600]
$f_{sch}(\hat{x}) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$ [-65.5, 65.5]
$ef_{10}(\hat{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{i-1}, x_i) + \ldots + f_{10}(x_n, x_1)$
donde $f_{10} = (x^2 + y^2)^{0,25} * [sin^2(50 * (x^2 + y^2)^{0,1}) + 1]$ [-100.0, 100.0]

Tabla 4: Funciones de prueba utilizadas

	Sphere				Rosen			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	2.6E-02	2.1E-02	6.7E-03	3.0E+05	1.1E+02	3.8E+01	2.6E+01
CHC	*	2.0E-32	9.0E-32	5.0E-32	*	2.0E+01	7.0E-01	2.0E+01
FANS	3.0E+05	2.6E-26	2.7E-26	3.3E-27	2.9E+05	8.7E+00	1.9E+01	1.8E-04
SA	3.0E+05	2.0E-08	5.4E-09	9.1E-09	3.0E+05	3.0E+00	4.6E+00	4.0E-03

	Rastringin				Schaffer			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	1.6E+01	4.4E+00	7.2E+00	3.0E+05	1.0E+03	3.2E+02	4.1E+02
CHC	*	0.0E+00	0.0E+00	100%	*	1.0E-01	3.0E-01	7.0E-12
FANS	1.3E+05	9.6E-01	9.8E-01	40%	3.0E+05	3.5E-05	4.8E-05	7.9E-07
SA	3.0E+05	2.4E-08	5.3E-09	1.6E-08	3.0E+05	8.7E-05	9.1E-05	4.3E-06

	Griewank				EF10			
	AE	*AB*	*SD*	*BF*	*AE*	*AB*	*SD*	*BF*
BCGA	3.0E+05	1.1E+00	1.1E-01	8.9E-01	3.0E+05	1.2E+00	3.6E-01	7.1E-01
CHC	*	0.0E+00	0.0E+00	100%	*	1.0E-07	2.0E-08	9.0E-08
FANS	2.0E+05	1.5E-02	1.6E-02	5.4E-20	3.0E+05	2.4E-11	1.7E-11	9.1E-12
SA	3.0E+05	2.1E-02	2.1E-02	1.9E-08	3.0E+05	1.0E-07	2.5E-08	5.9E-08

Tabla 5: Resultados obtenidos por *FANS*, *AGB* y *SA*.

de los errores para las funciones f_{ros} y f_{gri}, aunque los valores de BF fueron mejores en *FANS*.

Para f_{gri}, *FANS* no fue capaz de escapar de la gran cantidad de mínimos locales existentes. Solamente en 7 de 25 ejecuciones, *FANS* obtuvo valores cercanos a $1.e^{-20}$. Mientras que los t-test indicaron que *SA* resultó mejor que *FANS* en f_{ras}, *SA* nunca encontró el óptimo de la función mientras que *FANS* lo hizo en el 40 % de las ejecuciones. Incluso dicho valor se mejoró hasta un 90 % en f_{ras} variando los parámetros del administrador de vecindario.

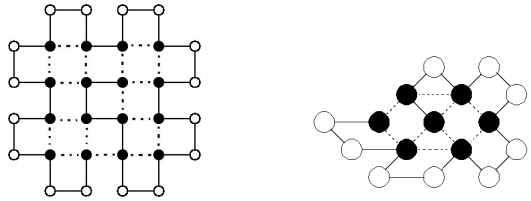
Finalmente, en la Tabla 5 se incluyen algunos resultados obtenidos mediante el algoritmo CHC reportados en [17]. El CHC es un algoritmo genético con codificación real especialmente diseñado para evitar los problemas de convergencia prematura y se lo suele utilizar como medida de referencia.

Los resultados mostraron que la versión de *FANS* utilizada permitió obtener mejores resultados que CHC en f_{sch}, f_{ros} y ef_{10} en términos de AB. Para f_{sph}, se consideró que el nivel de precisión alcanzado por *FANS* (10^{-26}) es más que suficiente para cualquier aplicación práctica. Los resultados del CHC en f_{ras} y f_{gri} son excelentes ya que siempre alcanzó el óptimo correspondiente. En f_{gri}, los valores de *FANS* resultaron regulares, aunque similares a los obtenidos mediante AG's distribuidos también presentados en [17].

5.4. El Problema de Predicción de Estructura de Proteínas

El Problema de Predicción de Estructura de Proteínas, *PSP*, es uno de los problemas abiertos más importantes a los que la que biología se enfrenta. En términos muy simples, se puede formular como: dada una secuencia de aminoácidos, cual es la estructura tridimensional asociada de mínima energía libre?.

PSP es un problema importante ya que la estructura 3D de una proteína determina



(a) (b)

Figura 5: Secuencias del modelo HP en reticulado cuadrado (a) y triangular (b).

la funcionalidad biológica de la misma. La obtención de la secuencia de aminoácidos de una proteína puede considerarse un problema resuelto; sin embargo, la determinación de la estructura mediante técnicas experimentales como la resonancia magnética nuclear o la cristalografía de rayos X, sigue siendo una tarea costosa, lenta y compleja.

Dada la complejidad de las proteínas, se debe recurrir a modelos simplificados para obtener información respecto al proceso de plegamiento. Dentro de los modelos simplificados, surge el modelo HP de K. Dill [14] donde una secuencia se representa como un string $s \in \{H, P\}^+$, donde H representa un aminoácido hidrofóbico y P un hidrofílico.

La Figura 5 muestra secuencias ubicadas en los reticulados cuadrado y triangular. Los contactos entre H's se muestran resaltados con líneas de puntos. La conformación de la Figura 5(a) tiene un puntaje de 17 mientras que la estructura en 5(b) tiene 9 (nueve contactos).

PSP en este modelo es encontrar la estructura de máximo puntaje y se probó que era un problema NP-Hard para el reticulado cuadrado [12] y el cúbico [4]. Entre los enfoques que se aplicaron para intentar resolverlo se pueden citar algoritmos genéticos, recocido simulado, GRASP y autómatas celulares[21, 22, 26, 29].

Para representar las estructuras se puede utilizar un esquema basado en *coordenadas internas*, donde surgen dos tipos: absoluta y relativa. En la codificación absoluta, las estructuras se representan como una lista de movimientos absolutos en el espacio correspondiente. Por ejemplo, si se utiliza un reticulado cuadrado en 2D, entonces una estructura e se codifica como un string $s = \{$**A**rriba, A**b**ajo, **I**zquierda, **D**erecha$\}^+$.

Bajo la codificación relativa, cada movimiento es interpreta en términos del anterior: la estructura se codifica como un string en el alfabeto $e = \{$**A**delante, Girar**D**erecha, Girar**I**zquierda$\}^+$.

Recientemente se realizó una comparación directa entre ambas codificaciones en el marco de algoritmos evolutivos [21]. En esta sección, describiremos dos trabajos [9, 27] sobre la aplicación de *FANS* a *PSP* cuyos objetivos fueron: primero, analizar la influencia de la codificación en los resultados que obtiene *FANS*; y segundo, evitar un problema teórico que se da en la aplicación de *AG*'s a *PSP* en este modelo.

5.4.1. Implementación de FANS

La aplicación de *FANS* a este problema difiere de las anteriores en el operador de modificación utilizado y su correspondiente administrador. Como valoración difusa se utiliza "aceptabilidad" y como administrador de vecindario, la estrategia *First*.

El operador $\mathcal{O}$ utiliza un parámetro k que representa el número de posiciones a cambiar en la solución (estructura) dada. Estas k posiciones pueden ser aleatorias o consecutivas, dando lugar a los modos *Flip* y *Segmento* respectivamente.

Cada vez que se ejecuta el administrador de operación, este modifica el parámetro k decrementando su valor en uno. De esta forma, se realizan modificaciones grandes al comienzo de la búsqueda (correspondientes a una etapa de exploración), las cuales se van afinando a medida que la ejecución progresa. El valor del parámetro es $k = n/4$, donde n indica la longitud de la secuencia. Cuando se alcanza $k = 0$, se ejecuta el procedimiento *Recomenzar*(), el cual genera una nueva solución aleatoria $\hat{s}$. Luego se asigna $k = n/4$ y *FANS* se reinicia desde $\hat{s}$.

5.4.2. Experimentos y Resultados para la Comparación de Codificación

Para estos experimentos se seleccionaron tres valores de $\lambda = \{0{,}0, 0{,}9, 1{,}0\}$, los cuales inducen

3 comportamientos para *FANS* denominados aquí λ_0, $\lambda_{,9}$, y λ_1. Cuando se utiliza λ_0, *FANS* se comporta como un procedimiento que produce caminos aleatorios: cualquier solución del entorno puede ser seleccionada. Para el esquema $\lambda_{,9}$, *FANS* puede realizar transiciones a soluciones que empeoren el costo, mientras que cuando se utiliza λ_1, se obtiene un comportamiento tipo hillclimbing: únicamente las soluciones que mejoran el costo son tenidas en cuenta.

Para los experimentos se utilizaron instancias en los reticulados cuadrado, cúbico y triangular. Para cada versión de *FANS*, codificación (absoluta y relativa) e instancia, se realizaron 30 ejecuciones del algoritmo.

Se realizaron T-test para detectar si la media de los valores obtenidos por los algoritmos bajo ambas codificaciones era diferente o no. Cuando se utilizó un esquema λ_0, la codificación relativa permitió obtener mejores resultados en los tres reticulados. Para el caso de $\lambda_{,9}$, la codificación relativa resultó mejor que la absoluta en el reticulado cuadrado y ambas permitieron obtener resultados similares en los otros dos reticulados. Finalmente, cuando *FANS* utilizó $\lambda = 1$, la codificación absoluta resultó mejor que la relativa en el reticulado triangular, no se detectaron diferencias en el cúbico, y la codificación relativa resultó superior en el reticulado cuadrado.

Estos resultados están en consonancia con los presentados en [21], donde se realizó la misma clase de experimentos pero utilizando algoritmos genéticos. Por lo tanto, si se pretenden aplicar otras heurísticas sobre este modelo del problema *PSP*, ambos resultados deben tenerse en cuenta al momento de decidir la codificación a utilizar.

5.4.3. Experimentos y Resultados: Evitando los problemas de *AG*

Es sabido que desde un punto de vista práctico, los *AG*'s son capaces de obtener buenos resultados para *PSP*. Sin embargo, existe un problema teórico derivado del uso de operadores de cruce estandar y una representación basada en coordenadas internas. En [23] se aplicó un *AG* a *PSP* y se mostró que el operador de cruce no transfería información entre individuos de la población. En otros términos, la *idea* del cruce no estaba funcionando. Para verificar este hecho, se utilizó el *headless chicken test* [19], el cual compara los resultados de un *AG* que utiliza cruce estandar frente a otro que utiliza cruce aleatorio (uno de los individuos a cruzar se genera aleatoriamente). Los resultados mostraron que para los casos de prueba, el *AG* con cruce aleatorio obtuvo resultados iguales e incluso mejores que el *AG* con cruce estandar. Por lo tanto, dado que no se produce transferencia de información, se argumentó que en dicha situación no es necesario el uso de una población de individuos y que *potencialmente* los mismos resultados podrían obtenerse mediante optimización sobre un único individuo.

A continuación se describen los experimentos realizados para verificar si *FANS* es capaz de obtener tan buenos resultados como un *AG*.

Teniendo en cuenta las conclusiones de la sección anterior, se eligió la codificación relativa para representar las soluciones y la versión de *FANS* denominada $\lambda_{,9}$. Para el *AG* implementado se empleó la misma codificación y como operador de mutación se utilizó el operador de modificación de *FANS*. El valor del parámetro k se asigna aleatoriamente en cada aplicación. El operador de cruce es de dos puntos y se realizó selección por torneos. Se utilizaron 7 instancias en el reticulado cuadrado y por cada una de ellas se realizaron 30 ejecuciones de cada algoritmo.

En términos de la media de los mejores valores obtenidos, *FANS* consiguió mejores resultados en 5 instancias sobre 7. En general también se verificó que la desviación estandar fue menor en *FANS* y ambos algoritmos fueron capaces de alcanzar el óptimo en todas las instancias salvo una, donde solamente el *AG* pudo alcanzarlo.

En resumen, se pudo concluir que *FANS* obtuvo los mismos resultados que un *AG* utilizando un esquema simple de optimización, dando evidencia para confirmar la hipótesis que la utilización de una población no es necesaria si el *AG* emplea un operador de cruce estandar y representación en coordenadas internas.

6. Conclusiones

En este trabajo se presentó *FANS*, una heurística basada en conjuntos difusos para problemas de optimización. Se describieron las características principales de sus componentes y el esquema del algoritmo. Posteriormente, se mostró la utilidad de sus dos elementos novedosos: la valoración difusa, y el uso de varios operadores durante la búsqueda. Según nuestro conocimiento, este último punto también se utiliza en una variante de VNS.

Sobre la valoración difusa, se mostró que permite reflejar el comportamiento cualitativo de otros métodos de búsqueda local, lo cual permite plantear que *FANS* puede considerarse como un framework de heurísticas basadas en búsqueda local. Es decir, mediante una idea relativamente simple, hemos sido capaces de capturar bajo un mismo esquema, un conjunto de métodos que resultan adecuados para resolver aceptablemente una amplia gama de problemas de optimización. También se destacó que *FANS* es en sí misma, una nueva heurística ya que permite obtener comportamientos diferentes a aquellos presentados por otros métodos clásicos.

Respecto a la utilización de varios operadores, se mostró su utilidad en dos sentidos: como mecanismo de escape de óptimos locales y como mecanismo para potenciar la búsqueda.

Finalmente, se describieron aplicaciones de *FANS* a cuatro problemas relevantes de optimización y que sirven de muestra sobre la potencia y versatilidad de esta nueva herramienta de resolución de problemas.

Agradecimientos

Este trabajo contó con apoyo de los Proyectos TIC2002-04242-CO3-02 y TIC 2002-00407. David Pelta es becario del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina.

Referencias

[1] T. Bäck. Genesys 1.0 user guide. Technical report, Univ. of Dortmund, Germany, 1992.

[2] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, 1996.

[3] J. Beasley. The or-library: a collection of test data sets. Technical report, Management School, Imperial College, London SW7 2AZ., 1997. http://mscmga.ms.ic.ac.uk/info.html.

[4] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[5] A. Blanco, D. Pelta, and J. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.

[6] A. Blanco, D. Pelta, and J. L. Verdegay. A fuzzy adaptive neighborhood search for function optimization. In *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies, KES 2000*, volume 2, pages 594–597, 2000.

[7] A. Blanco, D. Pelta, and J. L. Verdegay. Introducing fans: a fuzzy adaptive neighborhood search. In *Eigth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems, IPMU 2000*, volume 3, pages 1349–1355, 2000.

[8] A. Blanco, D. Pelta, and J. L. Verdegay. Learning about fans behaviour: Knapsack problems as a test case. In *International Conference in Fuzzy Logic and Technology, EUSFLAT'2001*, volume 1, pages 17–21, 2001.

[9] A. Blanco, D. Pelta, and J. L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *International Journal of Intelligent Systems*, 17(7):629–643, 2002.

[10] O. Cordón, F. Herrera, and M. Lozano. A classified review on the combination fuzzy

logic-genetic algorithms bibliography. Technical Report DECSAI-95129, Dept. of Computer Science and A.I., University of Granada, 1995. Ultima Actualización en Diciembre de 1996, 544 referencias, http://decsai.ugr.es/~herrera/fl-ga.html.

[11] E. Cox. *The Fuzzy Systems Handbook: A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems*. Academic Press, Boston, 1994.

[12] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):409–422, 1998.

[13] M. Delgado, J. Kacprzyk, J. Verdegay, and M. Vila, editors. *Fuzzy Optimization. Recent Advances*. Physica Verlag, 1994.

[14] K. A. Dill. Dominant forces in protein folding. *Biochemistry*, 24:1501, 1985.

[15] P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization*, pages 433–458. Kluwer, 1999.

[16] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[17] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, 2000.

[18] *8th International Fuzzy Systems Associations World Conference, IFSA99*, Taipei, Taiwan, 1999.

[19] T. Jones. Crossover, macromutation, and population based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kauffman, 1995.

[20] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, 1995.

[21] N. Krasnogor, W. Hart, J. Smith, and D. Pelta. Protein structure prediction with evolutionary algorithms. In W. B. et.al., editor, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1596–1601. Morgan Kaufman, 1999.

[22] N. Krasnogor, D. H. Marcos, D. Pelta, and W. A. Risi. Protein structure prediction as a complex adaptive system. In C. Janikow, editor, *Frontiers in Evolutionary Algorithms (FEA98)*, pages 441–447, 1998.

[23] N. Krasnogor, D. Pelta, P. M. Lopez, P. Mocciola, and E. de la Canal. Genetic algorithms for the protein folding problem: A critical view. In C. F. E. Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems, EIS'98*, pages 353–360. ICSC Academic Press, 1998.

[24] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[25] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1999.

[26] J. T. Pedersen and J. Moult. Genetic algorithms for protein structure prediction. *Current Opinion in Structural Biology*, 6:227–231, 1996.

[27] D. Pelta, N. Krasnogor, A. Blanco, and J. L. Verdegay. F.a.n.s. for the protein folding problem: Comparing encodings and search modes. In *Fourth International Metaheuristics Conference, MIC 2001*, pages 327–331, 2001.

[28] J. L. Verdegay, editor. *Fuzzy Sets based Heuristics for Optimization*. Studies in Fuzziness and Soft Computing. Springer, 2003.

[29] K. Yue, K. M. Fiebig, P. D. Thomas, C. H. Sun, E. I. Shakhnovich, and K. A. Dill. A test of lattice protein folding algorithms. *Proc. Natl. Acad. Sci. USA*, 92:325–329, 1995.

[30] H. J. Zimmermann. *Fuzzy Sets Theory and Its Applications*. Kluwer Academic, 1996.