PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# RESOLUCIÓN DEL SET COVERING PROBLEM UTILIZANDO AFSA

CRISTIAN MAURICIO DIAZ CANALES
ROMINA KARÍN RODRIGUEZ VEAS

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

**DICIEMBRE 2013**

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# RESOLUCIÓN DEL SET COVERING PROBLEM UTILIZANDO AFSA

CRISTIAN MAURICIO DIAZ CANALES
ROMINA KARÍN RODRIGUEZ VEAS

**Profesor Guía:** Ricardo Soto de Giorgis
**Profesor Co-referente:** Broderick Crawford

**Carrera:** Ingeniería de Ejecución en Informática.

**DICIEMBRE 2013**

*Dedicatoria*

*Para todos los que me apoyaron,*
*especialmente a mi familia y mi tía Smirna.*

*Dedicatoria*

*Para todos los que me apoyaron,*
*especialmente a mi familia y a mi novia Natalia Berrios.*

# Resumen

Artificial Fish Swarm Algorithm (AFSA) es una metaheurística moderna basada en los diferentes comportamientos observados en los cardúmenes de peces los cuales son: Prey Behavior, Swarming Behavior y Following Behavior. El problema computacional al cual se aplica esta metaheurística corresponde al Set Covering Problem (SCP), el cual busca minimizar la cantidad de elementos necesarios para cubrir la mayor cantidad de superficie posible de acuerdo a un conjunto de restricciones.



Palabras Claves: SCP, Solución Óptima, AFSA.



# Abstract

The Artificial Fish Swarm Algorithm metaheuristic (AFSA) is a modern metaheuristic based on the observed behaviors of school of fishes, which are: Prey Behavior, Following Behavior and Swarming Behavior. The computer problem to which we apply this metaheuristic correspond to the Set Covering Problem (SCP), which aims at minimizing the number of elements needed to cover as much surface amount as possible according to a set of constraints.



Key Words: SCP, Optimal Solution, AFSA.

# Índice

# Lista de Figuras

# Lista de Tablas

# 1 Introducción

La informática forma parte del estudio y desarrollo de los algoritmos necesarios para realizar procesos de optimización. Esta también nos entrega la formación necesaria para poder entender y comprender el cómo funcionan y su comportamiento a lo largo de su ejecución.

Estas técnicas de optimización consisten en resolver problemas donde se necesiten minimizar o maximizar en algunos casos, una función real o función objetivo, que represente algún problema del mundo real, tales como la cantidad de hospitales dentro de una comuna, la producción máxima de cajas para un contenedor,la distribución optima de alimentos en una comunidad, entre otras. Estas poseen datos de entrada que permiten el descubrimiento de los mejores valores de la función objetivo, dado un dominio definido incluyendo una serie de variables para su desarrollo.

Para el presente proyecto, se procederá a optimizar el problema de cobertura, más conocido como Set Covering Problem (SCP) el cual consiste en encontrar, dado un número de elementos, el menor conjunto de estos donde se puedan cubrir todos por igual. Para realizar esta tarea, se procederá a utilizar el algoritmo Artificial Fish Swarm Algorithm (AFSA), perteneciente a las técnicas de metaheurística de población, la cual permite encontrar una solución óptima en base al movimiento de las bancadas de peces en el mar, dependiendo de la visión, sus peces vecinos y de los comportamientos que decida ejecutar.

En primer lugar se procederá a plantear los objetivos generales y específicos del presente proyecto. También se presentará el estado del arte, donde se mostrará el estudio e investigación del problema a tratar en cuestión. Por último, se desarrollarán todos los tópicos necesarios para comprender la teoría y composición presente en el ámbito de estos problemas de optimización (Set Covering Problem, entre otros), como también la implementación de un prototipo que se llevó a cabo para resolver la problemática. A continuación se dan a conocer los objetivos referentes a este proyecto.

# 2 Definición de Objetivos

## 2.1 Objetivo General

- Implementar un algoritmo de resolución para el Set Covering Problem utilizando AFSA.

## 2.2 Objetivos Específicos

- Comprender el SCP.
- Implementar un algoritmo AFSA para SCP.
- Realizar experimentos con la implementación desarrollada.

# 3 Estado del Arte

A lo largo del tiempo, se han desarrollado muchas variaciones de las técnicas y métodos ya existentes, para resolver los distintos tipos de problemas relacionados a la optimización mediante la utilización de la programación con restricciones o de algún tipo de metaheurística (tales como las basadas en población o las basadas en trayectoria).

Se ha resuelto el SCP por Nehme Bilal y compañía, utilizando una nueva formulación de este, enfocando principalmente para ser utilizado con una metaheurística [3]. Este trabajo trata sobre eliminar las restricciones presentes en el problema SCP, eliminando así los problemas generados con las soluciones infactibles y problemas de redundancia.

Otro trabajo reciente sobre la resolución del Set Covering Problem, se puede apreciar en el trabajo de Ratnesh Rajan Saxena y compañía [4], el cual consiste en utilizar una técnica de enumeración para poder resolver el Set Covering Problem de característica difusa, linear y fraccional, utilizando una función de linearizacion para poder obtener una solución óptima al problema.

También existe un trabajo relacionado con el Set Covering Problem, desarrollado por Jordi Pereira e Igor Averbakh [5], el cual consiste en resolver el problema utilizando metaheurística genéticas y otros métodos, utilizando costos para el SCP de características inciertas, ya que estos valores son obtenidos desde un intervalo o conjuntos de valores finitos.

Se ha resuelto el Set Covering Problem, por B. Crawford y C. Castro [6], el cual utiliza la metaheurística ACO (Ant Colony Optimization) que utiliza procedimientos de lookahead para resolver tanto el Set Covering Problem, como el Set Partitioning Problem. Otro trabajo que también emplea ACO, corresponde al realizado por B. Crawford y C. Castro [7], el cual consiste en integrar la metaheurística ACO con procedimientos de lookahead y procedimientos de post-procesos para llegar a una solución. Incluso, se han presentado nuevas ideas para aplicar la metaheurística ACO para el Set Covering Problem, como lo es el trabajo realizado por Z.-G. Ren, Z.-R. Feng, L.-J. Ke y Z.-J. Zhang [9].

Se ha realizado también, un trabajo relacionado con el Set Covering Problem, por R. M. D. A. Silva y G. L. Ramalho [8], él hablaba sobre un sistema de hormigas (ant system) para resolver el Set Covering Problem.

Se ha llevado a cabo un trabajo realizado por L. Lessing, I. Dumitrescu, and T. Stutzle [10], el cual consiste en realizar una comparación entre dos algoritmos de la metaheurística ACO para resolver el Set Covering Problem. También se desarrolló un trabajo relacionado con el Set Covering Problem, por los autores M. Rahoual, R. Hadji, y V. Bachelet [11], el cual consistía en la realización de un sistema paralelo de la metaheurística ACO para resolver el Set Covering Problem. Finalmente tenemos el trabajo realizado por H. Mulati y A. A Constantino [12], el cual consistía en un algoritmo ACO orientado linealmente, para resolver el Set Covering Problem.

# 4 Set Covering Problem

El Set Covering Problem, es un problema clásico en combinatoria, ciencias de la computación y en teoría de la complejidad computacional, el cual consiste básicamente en identificar el menor número de conjuntos cuya unión contenga a todos los elementos del universo. Este problema se suele exponer gráficamente mediante la utilización de un mapa, que representa el universo, las regiones que representan los conjuntos como se aprecia en la figura 1. El universo se define como un conjunto de elementos del tipo $\{1,2\ldots n\}$, donde pueden existir $N$ de este tipo de conjuntos.

Por ejemplo, sea el universo $U = \{1,\ 2,\ 3,\ 4,\ 5\}$ y los conjuntos $S = \{\{1,\ 2,\ 3\},\ \{2, 4\},\ \{3, 4\}, \{4, 5\}\}$. Claramente, la unión de todos los conjuntos de $S$ contiene todos los elementos de $U$. Sin embargo, podemos cubrir todos los elementos con el siguiente conjunto de elementos, con menor número de elementos: $\{\{1, 2, 3\}, \{4, 5\}\}$.

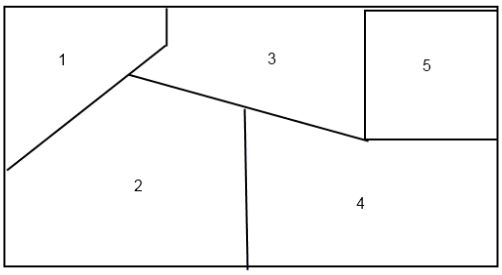


Figura 1: Mapa de ejemplo SCP divido en 5 regiones.

La definición formal del problema de cobertura SCP [13] se compone de la siguiente manera: Sea el universo $U$ y la familia $S$ de subconjuntos de $U$, una cobertura es una subfamilia $C \subseteq S$ de conjuntos, cuya unión es $U$.

En este caso las restricciones, la función objetivo y el dominio para el problema anterior serian de la siguiente manera:

- **Dominio**: Sea $x_i = \begin{cases} 1\ se\ contruye\ en\ la\ comuna. \\ 0\ no\ se\ construye\ en\ la\ comuna. \end{cases}$
- **Función Objetivo**: $Min \sum_{i=1}^{5} x_i$.
- **Restricciones**: las restricciones para este problema se definen de la siguiente manera:
    - $x_1 + x_2 + x_3 \geq 1$
    - $x_2 + x_1 + x_3 + x_4 \geq 1$

- $x_3 + x_1 + x_2 + x_4 + x_5 \geq 1$
- $x_4 + x_2 + x_3 + x_5 \geq 1$
- $x_5 + x_3 + x_4 \geq 1$

El problema del conjunto de cobertura para el presente proyecto será formalmente definido como [14]:

- Sea $A = (a_{ij})$ una m-fila, n-columna, matriz de ceros y unos.
- Se dice que la columna $j$ cubre una fila $i$ si $a_{ij} = 1$.
- Cada columna $j$ está asociada con un costo real no negativo $c_j$.
- Sea $I = \{1, \dots, m\}$el conjunto de filas y $j = \{1, \dots, n\}$ el conjunto de columnas.

El SCP requiere de un subconjunto de costo mínimo $S \subseteq J$, de tal modo que cada fila $i \in I$ es cubierta por al menos una columna $j \in S$. Un modelo matemático para la función objetivo es:

$$v(\text{SCP}) = \sum_{j \in J} c_j x_j$$

Sujeto a:

$$\sum_{j \in J} a_{ij} x_j \geq 1, \quad \forall\, i \in I,$$
$$x_j \in \{0, 1\}, \quad \forall\, j \in J$$

Donde $x_j = 1$ si $j \in S, x_j = 0$ de otra manera.

Las soluciones serán representadas mediante un arreglo binario para cada pez que representará las soluciones, un arreglo binario para los costos y de una matriz binaria, la cual cada una de sus filas representará las restricciones y variables del problema.

# 5 Metaheurística Artificial Fish Swarm Algorithm (AFSA)

La mayoría de las especies dentro del reino animal, presentan una serie de comportamientos que los diferencia del resto. Dentro de estos grupos separados de seres en la naturaleza, siempre son guiados por un líder, que se encarga de guiarlos a través del territorio, ya sea para buscar un lugar que les sirva como madriguera, o por el hecho de buscar alimentos para satisfacer las necesidades de la manada. Por otro lado, también están los grupos en los cuales, los símiles viven agrupados pero no poseen un líder que realice dicha tarea. Estos solamente actúan bajo las necesidades del momento y no poseen conocimiento en cuanto a las necesidades del resto o del entorno (aves, peces, ovejas, entre otras), sin embargo, estas pueden moverse en el entorno mediante el intercambio de información con el sujeto o miembro máscercano. Esta simple interacción es lo que hace que su comportamiento sea más sofisticado y pueda ser utilizado o replicado en otras ciencias para realizar la búsqueda de datos de una manera máseficiente. En base a lo anterior, un grupo de científicos ha desarrollado un método de optimización, basado en el comportamiento de los peces, llamado AFSA (Artificial Fish Swarm Algorithm) propuesta en el año 2002 [15] el cual en base a una serie de métodos que replican el comportamiento de estos, tales como el cazar el alimento, el seguir a los demás peces o el de nadar libremente por su entorno, permite encontrar resultados óptimos de una problemática especificada. Tal como en la naturaleza y al igual que en otras técnicas [1] [2] el pez puede encontrar el área que le otorgue mayor beneficio nutritivo de manera independiente o siguiendo al resto de los peces, por lo que el área que posea la mayor cantidad de estos, equivaldrá a el área que posee los mayores beneficios nutritivos para los peces en cuestión. Todos estos comportamientos asociados a lo anterior mencionado (cazar, seguir, o viajar en enjambre o cardumen) equivalen a encontrar el óptimo global de una problemática dada. Los métodos principales, por lo tanto, son *preying, following y swarming*; cuales se detallarán en profundidad.

## 5.1 Descripción de AFSA

En AFSA, el modelo del pez artificial basado en el comportamiento es construido en la arquitectura de múltiples vías paralelas, y este modelo encapsula el propio estado y el comportamiento del pez artificial. El proceso del algoritmo es el comportamiento auto adaptable del pez artificial. El algoritmo itera una vez a medida que el pez artificial se mueve.

Supongamos que el espacio de búsqueda es $D$ –dimension y $m$ un pez del cardumen. El estado del pez artificial puede ser expresado con el vector $X = (x_1, x_2, \dots, x_D)$, donde $x_i (i = 1, 2, \dots D)$ es la variable a ser buscada por el valor óptimo; la consistencia del alimento (fitness) en la posición actual del pez artificial puede ser representada por $Y = f(X)$, e $Y$ es la función objetivo; la distancia entre los peces artificiales puede ser expresada mediante el cálculo del código Hamming; $Visual$ representa la visión de la distancia; $Step$ es la longitud de paso máximo y δ es el factor multitud.

La consistencia del alimento en la posición actual del pez artificial puede ser representada por la ecuación:

$$\min \sum_{s \in S}^{n} c(S) * x_s$$

El entorno del pez artificial en el cual vive son el espacio de solución y los estados de los otros peces artificiales. El presente proyecto estará basado en la visión del pez [16].

El pez artificial realiza su percepción mediante su visión, como se muestra en la figura 2. X es el estado actual del pez artificial, $Visual$ es la distancia de la visual y $X_v$ es la posición visual en algún momento. Si el estado en la posición visual es mejor que el estado actual, va un paso adelante en esta dirección y alcanza el $X_{next}$ estado; de otra manera, continúa buscando dentro del rango de la visión. A mayor número de búsquedas de inspección que el pez artificial hace, más conocimiento acerca de los estados generales de la visiónes la que el pez artificial obtiene. Ciertamente, no es necesario viajar a través de estados complejos o infinitos, lo cual es útil para encontrar el óptimo global al permitir cierto óptimo local con cierta incertidumbre.

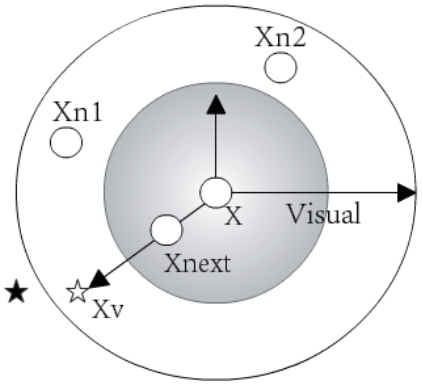


Figura 2: Concepto de visión del pez artificial.

Para calcular la distancia entre dos peces de características binarias, se emplea la distancia Hamming $H_d$ [17]. Este cálculo de distancia se utiliza entre dos bits de igual longitud y corresponde al número de posiciones en la que los bits correspondientes son diferentes. Este método se ve representado en la figura 3:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x^1 =$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $x^2 =$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

$$H_d(x^1, x^2) = 7$$

Figura 3: Distancia Hamming entre dos puntos binarios.

En donde $x^1$ representa el pez actual y $x^2$ representa el pez vecino. La distancia Hamming es calculada creando un nuevo vector comparando los dos vectores. Se agrega un 1 si los valores son distintos y un 0 si los valores son iguales, quedando como resultado la distancia Hamming igual a siete.

Al haber calculado la distancia Hamming entre todos los pares de peces en la población, los peces vecinos dentro del alcance visual de $x^i$son diferenciados como los puntos $x^j$ que satisfacen la condición $H_d(x^i, x^j) \leq v$ ( $v$ es la visual), para $j \in \{1, \dots, N\}, j \neq i$ , donde $v = \delta \, x \, n$ $(0 < \delta < 1)$. $n$, que corresponde al número de variables, representa la distancia Hamming máxima entre dos peces binarios.

### 5.1.1 Preying Behavior

Es un comportamiento biológico básico de los peces que tiende hacia el alimento. Los peces perciben la concentración de alimento para determinar el movimiento a través de la visión o el sentido para luego escoger la tendencia [16]. Suponiendo que el estado actual de un pez artificial es denotado como $X_i$, luego se selecciona aleatoriamente un nuevo estado denotado como $X_j$ que se encuentre dentro de su campo visual. Si $Y_i < Y_j$ en el problema en cuestion, se avanza en esa dirección. En otros casos, selecciona un estado $X_j$ de manera aleatoria nuevamente y se juzga si esta satisface la condición anterior. Si no se puede satisfacer después de varios intentos, denotados por la variable *try_numbers*, se mueve de posición aleatoriamente. Cuando el valor de la variable *try_number*s pequeño en el comportamiento *Prey_Behavior ()*, el pez artificial puede nadar aleatoriamente, lo que lo hace alejarse del lugar, que representa el valor extremo local que se busca en la función objetivo.

### 5.1.2 Swarming Behavior

Los peces se reúnen en grupo de forma natural durante el traslado en su hábitat, el cual es un tipo de hábito para garantizar la existencia de la colonia y evitar los peligros. Los peces obedecen 3 principios [18]:

- **Compartmentation**: Para evitar la congestión con otros peces.
- **Unification**: Para moverse aproximadamente en la dirección promedio de otros compañeros.
- **Cohesion**: Para moverse aproximadamente hacia el centro de los próximos compañeros.

Un pez artificial con el estado actual $X_i$ busca la cantidad de compañeros que se encuentran en el entorno actual, en las cercanías, donde se satisface la condición $d_{i,j} < Visible$; y se calcula la posición central de estos, denotada en la variable $X_c$ , $Y_c$ representa la consistencia de comida en la posición central y $n_f$ denota el número de compañeros de $X_i$ en las cercanías. Si $n_f \geq 1$, $X_j$ explora la posición central de sus compañeros. Si $\frac{Y_c}{n_f} > \sigma Y_i$, lo cual significa que la consistencia de comida en el centro del lugar donde están sus compañeros

es alta y en las partes colindantes no existe una aglomeración de individuos debido a la baja cantidad de alimentos en esta, $X_j$ cambia su posición hacia el centro donde se encuentran sus compañeros. En otros casos, el pez ejecuta el *Leap_Behavior ().* Si $n_f = 0$, se ejecuta el *Leap_Behavior ().*

### 5.1.3 Following Behavior

En el proceso de movimiento del cardumen, cuando uno o varios peces encuentran alimento, los demás peces del grupo lo rastrearán y alcanzarán el alimento rápidamente [16]. Un pez artificial con el estado actual $X_i$ busca la cantidad de compañeros que se encuentran en el entorno actual, en las cercanías, donde se satisface la condición $d_{i,j} < Visible$; y se busca la máxima posición, denotada en la variable $X_{min}$, $Y_{min}$ representa el máximo valor de los compañeros en los lugares cercanos y $n_f$ denota el número de compañeros de $X_{min}$ en las cercanías. Si $n_f \geq 1$ , $X_j$ explora la posición central de sus compañeros. Si $\frac{Y_c}{n_f} > \sigma Y_i$, lo cual significa que la consistencia de comida en el centro del lugar donde están sus compañeros es alta y en las partes colindantes no existe una aglomeración de individuos debido a la baja cantidad de alimentos en esta, $X_{min}$ cambia su posición hacia el centro, hacia $X_{min}$. En otros casos, el pez ejecuta el *Preying_Behavior ().*

### 5.1.4 Selection of the Behavior

El comportamiento de $X_i$ (iter) ($iter$ es el actual número iterativo) se definirá de acuerdo a un orden secuencial previamente establecido de los comportamientos. Si uno de los comportamientos no logra mejorar el óptimo, se ejecuta el siguiente comportamiento establecido en la secuencia. De no satisfacer ninguno, se queda con el mismo estado y pasa al pez siguiente $X_j$.

### 5.1.5 Bulletin

Es usado para registrar el estado óptimo y la consistencia del alimento en la posición actual del AF. Actualiza el Bulletin con el mejor estado de AF, el valor final del Bulletin es el valor óptimo del problema, el estado en el cual es la solución óptima del sistema [18].

### 5.1.6 Leap Behavior

Cuando el pez artificial no puede encontrar una solución que satisfaga el problema mediante los tres métodos principales anteriormente nombrados, procederá a ejecutar el comportamiento de salto (*Leap_Behavior ()*), el cual le permite escoger de manera randómica, un nuevo estado para salir del sector en el cual se encuentra y poder así encontrar una nueva solución o estado valido.

# 6 Implementación del problema

En el presente proyecto se implementó la metaheurística AFSA modificando ciertos comportamientos debido a la naturaleza binaria del Set Covering Problem (SCP). Los detalles de la implementación final se presentarán a continuación.

## 6.1 Parámetros Utilizados

Se implementó una interface en java, la cual contendrá los parámetros necesarios, que serán de carácter invariable (estáticos y finales en el caso de java), para que el problema funcione en sí. Estos datos corresponden a los siguientes:

- **max_inputs:** Corresponde al número máximo de iteraciones del problema.
- **max_particles:** Nº máximo de peces AFSA conocidas como m o población.
- **crowd_factor:** Factor de población. Su rango de valores esta entre 0 y 1.
- **factor_visual:** Factor de Visión. Su rango de valores va entre 0 y 1.
- **step_factor:** Corresponde a un número entero y sirve para generar soluciones.
- **try_numbers:** Corresponde al número de intentos en que el pez AFSA intentará buscar una mejor solución dentro del comportamiento *Prey_Behavior ()*.

Por otro lado, la población inicial AFSA, es representada mediante un array de largo m, donde m corresponde a la cantidad de variables presentes en el problema y el valor inicial de estas estructuras corresponde el llenado del array con valores 1. Esto representa la peor solución factible posible del problema.

## 6.2 Implementación del AFSA

- **Método Preying Behavior**: para el presente método se utilizó una iteración del tipo *For*, para representar la cantidad de veces (*try_numbers*) en la cual el pez intenta encontrar un mejor vector de solución para el problema. Para poder escoger un mejor estado, el pez debe buscar entre sus vecinos de manera aleatoria, los peces que se encuentren dentro de su rango visual, lo cual es representado como $Random\ (N(X_i, Visual))$, seleccionando uno de estos, el cual se representa mediante la notación $X_j$. Si el estado seleccionado, evaluado en la función objetivo, es mejor que la del pez actual, se reemplaza.

  La implementación se puede apreciar en la figura 4.

**Preying_Behavior( ){**

For i=0 to trynumber

$X_j$=Random($N(X_i, Visual)$);

If (Yj< Yi) then $X_i = X_j$ ;

end

end

}

Figura 4: Algoritmo Preying Behavior.

- **Método Swarming Behavior:** para el presente método, primero se guardan en una lista todos los peces artificiales que estén dentro de la visual, representado por la notación $nf = (N(X_i, Visual))$. Luego se verifica la condición $nf \neq 0\ y \frac{nf}{PecesTotales} < \delta$ y determinando la posición central de los peces vecinos anteriormente nombrados, la cual corresponde a determinar el pez que posee la menor distancia Hamming con respecto a todos sus vecinos. En caso de empate, se elige el pez con mejor fitness. Esta se representa media la terminología $X_c = Center\ (N\ (X_i, Visual))$. Se calcula la función objetivo del $X_c$. En caso de no cumplir con la condición, se ejecuta el *Leap_Behavior()*. Si el número de vecinos cercanos es igual a cero, ejecutara el *Leap_Behavior()* también. La implementación se puede apreciar en la figura 5.

**Swarming_Behavior( ){**

$n_f = N(X_i, Visual)$;

**If** $(nf \neq 0\ and\ \frac{nf}{PecesTotales} < \delta)$ **then**

$X_c = Center(N(X_i, Visual))$;

$Y_c = f(X_c)$;

**If** $Y_c < Y_i$ **then** $X_i = X_c$;

**else** Leap Behavior();

**else** Leap Behavior();

}

Figura 5: Algoritmo Swarming Behavior.

- **Método Following Behavior**: para el presente método se seleccionan todos los vecinos que se encuentren dentro de la visual, representado por la notación $N(X_i, Visual)$. Luego se evalúa el estado de los peces seleccionados en la función objetivo. Si se cumple que la cantidad de peces cercanos dividido en la población total es menor al factor de población $\delta$ y la función objetivo del pez que poseía el menor valor en la función objetivo es menor al valor obtenido evaluando el vector solución del pez actual en la función objetivo; entonces el estado del pez actual $X_i = X_j$. Si no se cumple con la condición, entonces se ejecuta el *Prey_Behavior()*. La implementación se puede apreciar en la figura 6.

**Following_Behavior( )**{

$Y_j = Min\left(f(X_j)\right), X_j \in N(X_i, Visual)$

$nf = (N(X_i, Visual))$;

If($\frac{nf}{PecesTotales} < \delta \; and \; Y_j < Y_i$)

Then $X_i = X_j$;

Else Prey Behavior();

end

}

Figura 6: Algoritmo Following Behavior.

- **Método Leap Behavior:** este método solo se ejecuta en caso de que en los comportamientos *Swarming_Behavior() y Following_Behavior()* no se encuentre un óptimo. Su función principal es escapar del lugar en el cual se encuentran. Para Esto selecciona un índice de manera aleatoria dentro del array de estado y cambia por 0 y 1 posición a posición con un límite máximo definido por el parámetro $step$. La implementación se nota en la figura 7.

**Leap_Behavior( )**{

$X_i = Random\ (0,1) * Random(índice) * Step$

}

Figura 7: Algoritmo Leap Behavior.

- **Método Calcular Centro:** para obtener el centro del cardumen, se recurre a calcular el pez vecino que posee la menor suma de distancias Hamming con respecto al vecindario del pez actual. Estas distancias se van sumando y guardando en un array. Si ocurre un empate, se opta por comparar el $fitness$ de los peces con menor distancia, de los cuales se escoge el que posea menor fitness. La implementación se puede apreciar en la figura 8.

```
Public  Artificial_Fish Calcular_centro(int cant_variables,ArrayList<Artificial_Fish> AF_cercanos)
{
      Artificial_Fish pez_actual;
       int suma_distancias_actual=0;
       int indice_menor=0;
       int suma_distancias_menor=99999999;
       int distancias []= new int[AF_cercanos.size()];
       ArrayList<int []>  menores= new ArrayList<>();// en caso de empate
       Artificial_Fish pez_mejor_costo= new Artificial_Fish();
       int costo_menor;
       int costo_empate=999999999;
       for(int j=0; j < AF_cercanos.size();j++){
            pez_actual= AF_cercanos.get(j);
            for(int i =0; i < AF_cercanos.size();i++){
                        if(AF_cercanos.get(i)!=pez_actual)
                        suma_distancias_actual=calcular_distancia(AF_cercanos.get(i),pez_actual.getEstado());
            }
            distancias[j]=suma_distancias_actual; }
       for(int l=0; l < distancias.length;l++){
       //buscar la distancia menor
             if(distancias[l]<suma_distancias_menor){
                        suma_distancias_menor=distancias[l];
            }
      }
       //sacar iguales   y calcular costo
       for(int k=0; k < distancias.length;k++){
            if(distancias[k]==suma_distancias_menor){
                        costo_menor=calcular_costo (AF_cercanos.get(k).getEstado(),costos, cant_variables);
                        if(costo_menor< costo_empate){
                                    pez_mejor_costo=AF_cercanos.get(k);
                                    costo_empate=costo_menor;
                        }
            }
       }
     return pez_mejor_costo;
}
```

Figura 8: Algoritmo Calcular Centro.

- **Método Calcular Distancia**: como se mencionó anteriormente, la distancia entre dos peces AFSA, se calcula mediante el código Hamming, el cual analiza las diferencias entre los 2 vectores posición a posición. Si los bits en la posición actual del vector son diferentes, el contador aumenta en 1. El código se puede apreciar en la figura 9.

```
public int calcular_distancia(Artificial_Fish
pez,Artificial_Fish pez_vecino)

{
 int cont_distancia=0;
 for(int j=0 ; j< pez.Estado.length;j++){

      if(pez_vecino.Estado[j]!=pez.Estado[j]){
         cont_distancia++;
      }
}

 return cont_distancia;
}
```

Figura 9: Algoritmo Calcular Distancia.

## 6.3 Implementación de AFSA para el Set Covering Problem

- **Mapa del Set Covering:** corresponde a una matriz de $nxm$, donde $n$ corresponde al número de restricciones presentes en el problema y $m$ la cantidad de regiones presentes en este. Estos datos y los valores de las casillas para rellenar el mapa son leídos de un archivo txt.
- **Vector de Costos:** corresponde a un vector de largo o tamaño $m$, que contendrá el costo que implica cada región presente en el mapa. Los valores de este vector son leídos desde un archivo txt.
- **Archivo de texto del problema:** corresponde a un archivo de texto plano que posee cierto formato para ser leído por el programa, de manera que permita crear las instancias necesarias para representar el Set Covering Problem. El formato de este archivo se muestra en la figura 10.

Nº_restricciones Nº_columnas

//Valores del Vector de Costos

Valor_col_1 ValorCol _2…….

Valorcol_n

//fin valores vector de costos

Numero_regiones_con_valor _1

Posiciones de las regiones que poseen el valor 1 en la región del mapa.

Figura 10: Formato del archivo txt del SCP.

## 6.4 Implementación AFSA+SCP

Para representar el problema de conjunto de cobertura, utilizando la metaheurística AFSA, se implementó mediante el lenguaje de programación java, por lo que todas las instancias necesarias para resolver el problema fueron representadas mediante clases. Las clases utilizadas se pueden observar en la figura 11.

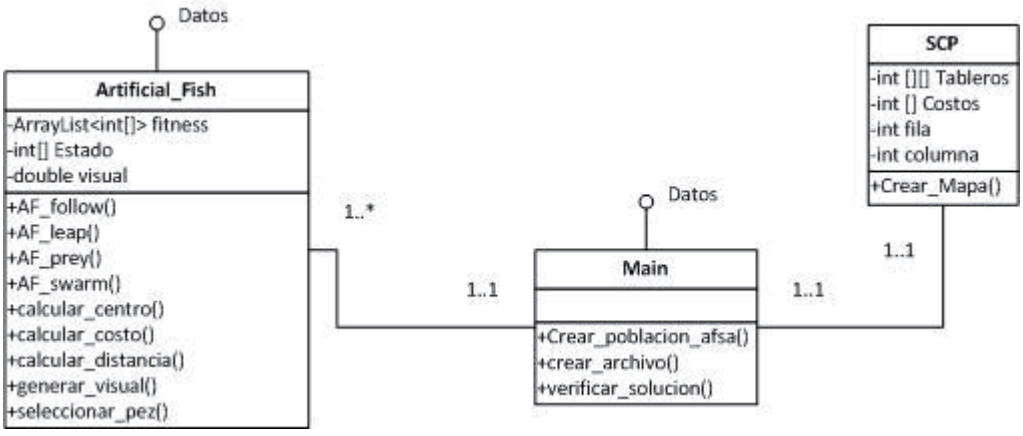


Figura 11: Diagrama de Clases del Programa.

## 6.5 Main del programa AFSA+SCP

```
Begin
Crear_Mapa(archivo);
Crear_Poblacion_inicial();
Desde i=0 hasta Numero_iteraciones Hacer
flag=0;
estado="Follow";
      Desde j=0 hasta Cantidad_Totales_Peces Hacer
            Si Estado=="follow"  && flag ==0 Hacer
                  pez_iteracion_actual ejecuta Follow();
                  Si Verificar == 1 Hacer
                  Si minimo_local < minimo_temporal Hacer
                        minimo_temporal=minimo_local;
                        flag=1;
                  fin_si
                fin_si
            Si Estado=="prey"  && flag ==0 Hacer
                  pez_iteracion_actual ejecuta Prey();
                  Si Verificar == 1 Hacer
                  Si minimo_local < minimo_temporal Hacer
                        minimo_temporal=minimo_local;
                        flag=1;
                  fin_si
            fin_si
            Si Estado=="swarm"  && flag ==0 Hacer
                  pez_iteracion_actual ejecuta Swarm();
                  Si Verificar == 1 Hacer
                  Si minimo_local < minimo_temporal Hacer
                        minimo_temporal=minimo_local;
                        flag=1;
                  fin_si
            fin_si
            Si Estado=="Leap"  && flag ==0 Hacer
                  pez_iteracion_actual ejecuta Leap();
                  Si Verificar == 1 Hacer
                 Si minimo_local < minimo_temporal Hacer
                        minimo_temporal=minimo_local;
                        flag=1;
                  fin_si
            fin_si
      fin_desde
      Si flag==1 Hacer
            bulletin=minimo_temporal;
            flag=0;
      fin_si
fin_desde
End
```

Figura 12: Main del AFSA+SCP.

## 6.6 Experimentos y Resultados

Para realizar los experimentos se utilizó el programa *Netbeans* y el lenguaje de programación Java. Con motivos de poder realizar un seguimiento a los resultados obtenidos por el programa creado, estos fueron almacenados en un archivo .txt con un formato apropiado para realizar las conclusiones pertinentes. Se utilizaron los siguientes parámetros de entrada para realizar las pruebas: ***max_inputs*** = 200000~400000, ***max_particles*** = 10~20, ***crowd_factor*** (***δ***) = 0.918, ***factor_visual*** = 0.7~0.9***, step_factor*** = 3, ***try_numbers*** = 10~20. Durante los cálculos se utilizó un notebook con procesador Intel Core i5® de 2,53 MHz, con 4 GB de memoria RAM, 2 de estas asignadas a la máquina virtual de java. El sistema operativo en el cual se realizaron todos estos procedimientos corresponden a Windows 7 Home Premium Edition® de 64 bits. Con motivos de evitar resultados basados en la suerte, se corrió el algoritmo 10 veces para cada instancia, para poder realizar un análisis a la convergencia de los resultados que se pretende alcanzar.

Las instancias probadas fueron obtenidas desde OR-Library [20], la cual posee muchos benchmarks probados del problema del Set Covering. Se probaron un total de 65 instancias, con los parámetros mencionados anteriormente, logrando obtener los resultados reflejados en la Tabla 1.

| Instancia SCP | Óptimo Conocido | Mejor Óptimo en 10 Ejecuciones. | Peor Óptimo en 10 Ejecuciones. | Promedio Óptimos en 10 Ejecuciones |
|---|---|---|---|---|
| 4.1 | 429 | 470 | 510 | 490 |
| 4.2 | 512 | 570 | 600 | 585 |
| 4.3 | 516 | 580 | 610 | 595 |
| 4.4 | 494 | 580 | 590 | 585 |
| 4.5 | 512 | 596 | 600 | 598 |
| 4.6 | 560 | 610 | 689 | 649,5 |
| 4.7 | 430 | 500 | 510 | 505 |
| 4.8 | 492 | 560 | 590 | 575 |
| 4.9 | 641 | 720 | 790 | 755 |
| 4.10 | 514 | 707 | 710 | 708,5 |
| 5.1 | 253 | 350 | 370 | 360 |
| 5.2 | 302 | 410 | 450 | 430 |
| 5.3 | 226 | 376 | 390 | 383 |
| 5.4 | 242 | 390 | 410 | 400 |
| 5.5 | 211 | 328 | 330 | 329 |
| 5.6 | 213 | 300 | 310 | 305 |
| 5.7 | 293 | 350 | 370 | 360 |
| 5.8 | 288 | 310 | 350 | 330 |
| 5.9 | 279 | 300 | 320 | 310 |
| 5.10 | 265 | 360 | 392 | 376 |
| 6.1 | 138 | 170 | 175 | 172,5 |
| 6.2 | 146 | 198 | 198 | 198 |
| 6.3 | 145 | 170 | 192 | 181 |
| 6.4 | 131 | 160 | 185 | 172,5 |
| 6.5 | 161 | 190 | 200 | 195 |
| A.1 | 253 | 476 | 476 | 476 |
| A.2 | 252 | 420 | 420 | 420 |
| A.3 | 232 | 444 | 456 | 450 |
| A.4 | 234 | 444 | 444 | 444 |
| A.5 | 236 | 407 | 407 | 407 |
| B.1 | 69 | 124 | 124 | 124 |
| B.2 | 76 | 121 | 121 | 121 |
| B.3 | 80 | 110 | 110 | 110 |
| B.4 | 79 | 135 | 135 | 135 |
| B.5 | 72 | 130 | 131 | 130,5 |
| C.1 | 227 | 329 | 329 | 329 |

| | | | | |
|---|---|---|---|---|
| C.2 | 219 | 320 | 340 | 330 |
| C.3 | 243 | 310 | 320 | 315 |
| C.4 | 219 | 305 | 310 | 307,5 |
| C.5 | 215 | 320 | 340 | 330 |
| D.1 | 60 | 120 | 120 | 120 |
| D.2 | 66 | 130 | 130 | 130 |
| D.3 | 72 | 155 | 155 | 155 |
| D.4 | 62 | 122 | 122 | 122 |
| D.5 | 61 | 148 | 148 | 148 |
| NRE.1 | 29 | 79 | 79 | 79 |
| NRE.2 | 30 | 86 | 86 | 86 |
| NRE.3 | 27 | 78 | 78 | 78 |
| NRE.4 | 28 | 80 | 80 | 80 |
| NRE.5 | 28 | 80 | 80 | 80 |
| NRF.1 | 14 | 42 | 42 | 42 |
| NRF.2 | 15 | 49 | 49 | 49 |
| NRF.3 | 14 | 45 | 45 | 45 |
| NRF.4 | 14 | 44 | 44 | 44 |
| NRF.5 | 13 | 46 | 46 | 46 |
| NRG.1 | 176 | 250 | 250 | 250 |
| NRG.2 | 154 | 260 | 260 | 260 |
| NRG.3 | 166 | 198 | 198 | 198 |
| NRG.4 | 168 | 201 | 201 | 201 |
| NRG.5 | 168 | 197 | 197 | 197 |
| NRH.1 | 63 | 110 | 110 | 110 |
| NRH.2 | 63 | 121 | 121 | 121 |
| NRH.3 | 59 | 111 | 111 | 111 |
| NRH.4 | 58 | 109 | 115 | 112 |
| NRH.5 | 55 | 105 | 107 | 106 |

Tabla 1: Resultados de pruebas AFSA+SCP.

Las pruebas de la Tabla 1 se ejecutaron con 200 mil iteraciones y en ninguna de las pruebas se logró alcanzar el óptimo conocido. Para las pruebas realizadas con 400 mil iteraciones, los resultados se pueden apreciar en el Anexo D.

## 6.7 Mejoras y pruebas anexas AFSA+SCP

1. **Visual Variable**: Se implementó mediante la siguiente ecuación:

$$Visual2 = visual * (1 - (iteracion_{actual}/cantidad\_Maxima_{iteraciones}))$$

2. **Reparación de estados Infactibles**: Se implementó una función que permite reparar un estado cuando no cumpla con alguna restricción del problema. La reparación se realiza mediante el análisis de cobertura v/s el costo que esta posea en la fila. Su función se puede apreciar en el Anexo A.

Con esta implementación se realizaron pruebas sobre 1 instancia de cada grupo (4.1, A.1, B.1, C.1, D.1, NRE.1, NRF.1, NRG.1, NRH.1).Los resultados de estas pruebas y el algoritmo principal del proyecto, se pueden apreciar en el Anexo B y C respectivamente.

# 7 Conclusiones

El investigar sobre el problema del conjunto de cobertura, ayudó a conocer la complejidad de este, los factores a tomar en cuenta para su resolución y la importancia que este problema posee, ya que se presenta a menudo en la vida diaria, por ejemplo en la selección de archivos en un banco de datos, localización de servicios, balanceo en líneas de producción, entre otros, y de cómo un programa puede facilitar a la resolución de este tipo de problemas.

Para la resolución del problema en este caso, se estudió, analizó y aplico la metaheurística Artificial Fish Swarm Algorithm, la cual se basa en el comportamiento de los peces para encontrar alimento por si solos o siguiendo a otros peces, para lo cual este algoritmo recrea los cuatro principales comportamientos observados que son el cazar, el moverse libremente, viajar en cardumen y seguir a otro pez. La idea principal es encontrar un óptimo (ya sea global o local) como solución al problema.

A lo largo de este informe, se pueden apreciar las pruebas realizadas con la metaheurística AFSA al problema de conjunto cobertura SCP, en la cual se ven reflejados los resultados obtenidos para las distintas instancias utilizadas como referencia. Si bien los resultados de estas instancias no alcanzaron el óptimo global con 200 mil y con 400 mil iteraciones, si se acercaron bastante. Aun así, Intentar aumentar la cantidad de iteraciones, o probar una combinación diferente de parámetros, no implicaría un gran cambio a la hora de analizar los resultados, ya que estos están configurados casi bordeando los límites permitidos y definidos en la teoría.

Para finalizar, con las modificaciones realizadas posteriormente al código, se pudo apreciar un gran cambio en cuanto a los resultados obtenidos, y esto se debe en parte a la función de reparación de soluciones, ya que permite reutilizar estados que poseían características que los convertían en estados Infactibles.

# 8 Referencias


[1] Z. Chuang-ye and M. Yuan-bin. **AFSA and PSO Hybrid Algorithm Based on Collaborative Evolution**. Journal of Guangxi University for Nationalities(Natural Science Edition) 2009-01. 2009.

[2] B. Crawford and C. Castro. **Ant Colonies using Arc Consistency Techniques for the Set Partitioning Problem**. IFIP PPAI 2006: 295-301. 2006.

[3] Nehme Bilal, Philippe Galinier, and Francois Guibault. "**A New Formulation of the Set Covering Problem for Metaheuristic Approaches**," ISRN Operations Research, vol. 2013, Article ID 203032, 10 pages, 2013. doi:10.1155/2013/203032.

[4] Ratnesh Rajan Saxena, Rashmi Gupta. **Enumeration Technique for Solving Linear Fractional Fuzzy Set Covering Problem**. International Journal of Pure and Applied Mathematics**,** vol. 84, issue 5, 2013.

[5] Jordi Pereira, Igor Averbakh. **The Robust Set Covering Problem with interval data**. Annals of Operations Research, vol. 207, issue 1, pp 217-235. 2013.

[6] B. Crawford and C. Castro, "**Aco with lookahead procedures for solving set partitioning and covering problems**," in Proceedings of Workshop on Combination of Metaheuristic and Local Search with Constraint Programming Techniques, Nantes, France, November 2005.

[7] B. Crawford and C. Castro, "**Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problem**," in Proceedings of the 8th International Conference on Artificial Intelligence and Soft Computing (ICAISC '06), L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, Eds., vol. 4029 of Lecture Notes in Computer Science, pp. 1082–1090, Springer, 2006.

[8] R. M. D. A. Silva and G. L. Ramalho, "**Ant system for the set covering problem**," in Proceedings of IEEE International Conference on Systems, Man and Cybernetics, vol. 5, pp. 3129–3133, October 2001.

[9] Z.-G. Ren, Z.-R. Feng, L.-J. Ke, and Z.-J. Zhang, "**New ideas for applying ant colony optimization to the set covering problem**," Computers and Industrial Engineering, vol. 58, no. 4, pp. 774–784, 2010.

[10] L. Lessing, I. Dumitrescu, and T. Stutzle, "**A comparison between aco algorithms for the set covering problem**," in Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS '04), pp. 1–12, 2004.

[11] M. Rahoual, R. Hadji, and V. Bachelet, "**Parallel ant system for the set covering problem**," in Ant Algorithms, pp. 262–267, 2002.

[12] M. H. Mulati and A. A. Constantino, "**Ant-line: a lineoriented aco algorithm for the set covering problem**," in Proceedings of the 30th International Conference of the Chilean Computer Science Society (SCCC '11), pp. 265–274, Curicó, Chile, November 2011.

[13] Problema del Conjunto de Cobertura, url: http://es.wikipedia.org/wikia/problema_del_conjunto. Revisado por última vez el 17 de abril de 2013.

[14] Z. Ren, Z. Feng, L. Ke and Z. Zhang. **New ideas for applying ant colony optimization to the set covering problem**. Computers& Industrial Engineering 58(4): 774-784 (2010). 2010.

[15] L. Xiao lei, S. Zhi-jiang and Q. Ji-xin. **An optimizing method based on autonomous animals: fish-swarm algorithm**. Systems Engineering Theory & Practice, vol. 22, pp. 32-38. 2002.

[16] M.Neshat, G.Sepidnam, M. Sargolzaei and A. NajaranToosi. **Artificial Fish Swarm Algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications**. Artificial Intelligence Review. 2012.

[17] Md. A. Kalam Azad, A. María A.C. Rocha and E. M.G.P. Fernandes. **Solving Multidimensional 0-1 Knapsack Problem with an Artificial Fish Swarm Algorithm**. ICCSA (3) 2012: 72-86. 2012.

[18] Y. Peng. **An improved artificial fish swarm algorithm for optimal operation of cascade reservoirs**. JCP 6(4): 740-746. 2011.

[19] Y. Cai. **Artificial Fish School Algorithm Applied in a Combinatorial Optimization Problem**. MECS, IJISA, vol. 2, n° 1,pp. 37-43. 2010.

[20] Benchmarks para pruebas, url: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/. Revisado por última vez el 21 de octubre de 2013.