"Resolución del Problema de Set-Covering utilizando un Algoritmo Genético"

Pablo Itaim Ananias

Valparaíso, 20 de Junio del 2005

Resumen

El *Set Covering Problem* (SCP)es un problema clásico, que consiste en encontrar un conjunto de soluciones que permitan cubrir un conjunto de necesidades al menor costo posible. Existen muchas aplicaciones de este tipo de problemas, siendo las principales la localización de servicios, selección de archivos en un banco de datos, simplificación de expresiones booleanas, balanceo de líneas de producción, entre otros. En el presente trabajo se muestra el estado del arte del problema de Set-Covering y se presenta un Algoritmo Genético para su resolución.

1. Introducción

El *Set Covering Problem* (SCP), el *Set Packing Problem* (SPP) y el *Set Partitioning Problem* (SPaP), son unos problemas clásicos, que pertenecen a la clase NP-Completo [1] y en donde la entrada esta dada por varios conjuntos de elementos o datos que tienen algún elemento en común. En general, estos problemas consisten en encontrar el número mínimo de conjuntos que contengan un cierto número de elementos de todos los conjuntos. En otras palabras, consiste en encontrar un conjunto de soluciones que permitan cubrir un conjunto de necesidades al menor costo posible. Un conjunto de necesidades corresponde a las filas, y el conjunto solución es la selección de columnas que cubren en forma óptima al conjunto de filas. Estos tres problemas tienen la misma función objetivo y se diferencian en las restricciones que deben cumplir.

En este trabajo nos vamos a enfocar en el Set-Covering Problem, poniendo énfasis en el estado del arte del problema y en la descripción de un algoritmo genético diseñado para resolverlo. Para ello, el trabajo ha sido organizado de la siguiente manera: en el punto 2 se da una definición formal del problema, en el punto 3 se presenta una visión del estado del arte respecto a los algoritmos y técnicas incompletas utilizadas para la resolución de problemas del tipo Set-Covering. Aquí se plantean las relajaciones más comunes del modelo y los algoritmos basados en heurísticas y exactos más difundidos en la literatura. El punto 4, presenta el modelo matemático planteado para la resolución de un problema específico. En el punto 5 se discute respecto a la representación utilizada para el modelamiento del problema, para luego pasar al punto 6 en donde se describe el algoritmo genético desarrollado. El punto 7 indica los experimentos que se efectuaron principalmente en la etapa de sintonización de los parámetros del algoritmo y el punto 8 muestra los resultados obtenidos al ejecutar la aplicación con problemas de test. Finalmente, en el punto 9 se presentan las principales conclusiones del trabajo realizado.

2. Definición del Problema

Como dijimos en el punto anterior, el problema de Set-Covering consiste en encontrar el número mínimo de conjuntos que contengan todos los elementos de todos los conjuntos dados. Existen muchas aplicaciones de este tipo de problemas, siendo las principales la localización de servicios, selección de archivos en un banco de datos, simplificación de expresiones booleanas, balanceo de líneas de producción, entre otros. [2].

Una definición más formal es la siguiente.

Sea:

$$\begin{aligned} A &= (a_{ij}) \text{ una matriz binaria } (0,1) \text{ de dimensiones } m \text{ x } n \\ C &= (c_j) \text{ un vector } n \text{ dimensional de enteros} \\ M &= \{1, \ldots, m\} \\ N &= \{1, \ldots, n\} \end{aligned}$$

El valor $c_j (j \in N)$ representa el costo de la columna j, y podemos asumir, sin pérdida de generalidad, $c_j > 0$ para $j \in N$. Así, decimos que una columna $j \in N$ *cubre* una fila $i \in M$ si $a_{ij} = 1$. El SCP busca un subconjunto de columnas de costo mínimo $S \subseteq N$, tal que cada fila $i \in M$ esta cubierta por al menos una columna $j \in S$.

Así, el problema de Set-Covering se puede plantear de la siguiente manera:

$$\text{mín}(z) = \sum_{j=1}^{n} c_j x_j \tag{1}$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad \forall i \in M \tag{2}$$

$$x_j = \begin{cases} 1 & \text{si } j \in S \\ 0 & \text{si no} \end{cases} \qquad \forall j \in N \tag{3}$$

Para entender mejor el planteamiento, supongamos el ejemplo de la Figura 1. Se tiene una ciudad dividida en 20 sectores y 10 lugares en donde pueden ser instalados ciertos servicios, como por ejemplo Cuarteles de Bomberos, Hospitales, etc. Cada lugar puede dar servicio a las comunas adyacentes. Se pide determinar donde ubicar los servicios de tal forma de minimizar los costos, pero asegurar que se cubran todos los sectores.

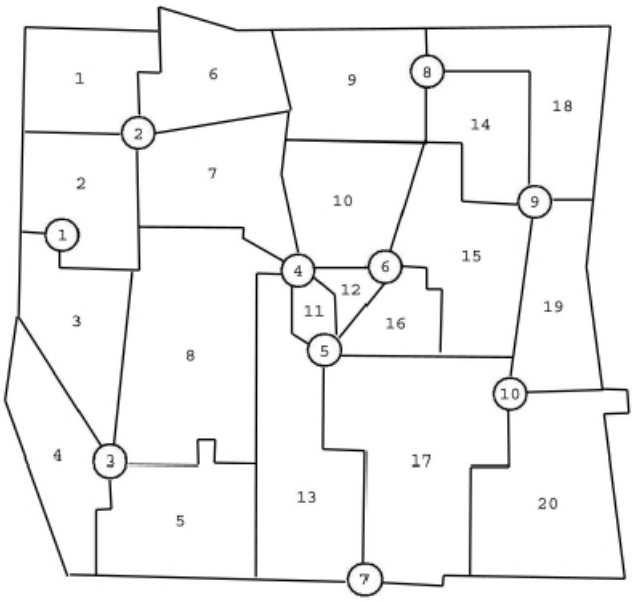

Figura 1: Sectores de una Ciudad y posibles lugares de instalación de servicios

Tenemos el vector de costos $C = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$, $M = \{1, \ldots, 20\}$ y $N = \{1, \ldots, 10\}$. De la Figura 1, podemos construir la Matriz de cobertura A. Así, un $a_{ij} = 1$ indica que la columna j cubre a la fila i. Por ejemplo, la columna $j = 2$ cubre a las filas $i = 1$, $i = 2$, $i = 6$ e $i = 7$ mientras que la columna $j = 10$ cubre a las filas $i = 17$, $i = 19$ e $i = 20$.

Ahora, habría que encontrar el vector binario $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\}$ tal que se cumpla lo siguiente:

$$\begin{aligned} \text{mín}(z) &= \sum_{j=1}^{10} c_j x_j \\ \text{Sujeto a:} & \\ \sum_{j=1}^{10} a_{ij} x_j &\geq 1 \qquad \forall i \in M \\ x_j &= \begin{cases} 1 & \text{si se instala el servicio en la ubicación } j \\ 0 & \text{si no} \end{cases} \qquad \forall j \in N \end{aligned}$$

$$A = \begin{pmatrix} 0100000000 \\ 1100000000 \\ 1010000000 \\ 0010000000 \\ 0010000000 \\ 0100000000 \\ 0101000000 \\ 0011000000 \\ 0000000100 \\ 0001010000 \\ 0001100000 \\ 0001110000 \\ 0001101000 \\ 0000000110 \\ 0000010010 \\ 0000110000 \\ 0000101001 \\ 0000001100 \\ 0000000011 \\ 0000000001 \end{pmatrix}$$

La Solución de este problema nos indicará donde debemos ubicar los servicios para que cubran todos los sectores a un costo mínimo.

3. Estado del Arte

Dada la dificultad de determinar la solución óptima (es un problema NP-difícil) y el gran tamaño de los problemas reales, es que se han ideado varios algoritmos de solución óptima y algoritmos basados en Heurística. Aunque se ha probado que la existencia de un algoritmo polinomial que se aproxime a la solución óptima en un valor cercano a $\frac{1}{4} \log m$, implica que P=NP [3], en la práctica la mayoría de las heurísticas que aparecen en la literatura se acercan muy eficientemente a las soluciones óptimas [4]. En general, dada la estructura del problema en los casos del mundo real y al considerable esfuerzo en desarrollar algoritmos que rindan más y mejor en esos casos, el actual estado del arte del SCP es que los casos con unos cientos de filas y unos miles de columnas pueden resolverse obteniendo resultados óptimos y los casos con unos miles de filas y unos millones de columnas pueden resolverse obteniendo un valor dentro del 1 % alrededor del óptimo en un tiempo de ejecución computacional razonable [5].

Fisher y Kedia [6] presentaron un algoritmo de solución óptima basado en una heurística dual y aplicado a SCP con hasta 200 filas y 2000 columnas. Beasley [7] combina heurística Lagrangiana con una estrategia de Branch mejorada para aumentar las capacidades de su algoritmo anterior [8] y resolver problemas de hasta 400 filas y 4000 columnas. Estos algoritmos de solución cercana a la óptima están basados en procedimientos de árboles de búsqueda.

Entre los métodos basados en heurísticas, Beasley [9] presentó un algoritmo heurístico Lagrangiano e informó que su heurística había dado resultados de mejor calidad que otras heurísticas [10] [11] para problemas

que involucraran hasta 1000 filas y 10000 columnas. Por otra parte, Jacobs y Brusco [12] desarrollaron una heurística basada en Simulated Annealing e informaron un éxito considerable en problemas de hasta 1000 filas y 10000 columnas. Sen [13] investigó el rendimiento de un algoritmo de simulated annealing y de un algoritmo genético simple en el SCP de costo mínimo, pero dio pocos resultados computacionales. En otro estudio, Beasley [14] desarrolló una heurística para SCP sin costo único basado en algoritmos genéticos. Los resultados computacionales indicaron que dicha heurística era capaz de generar soluciones óptimas para problemas de tamaño pequeño y soluciones de alta calidad para problemas de gran tamaño.

Adicionalmente, en la literatura se ilustran varios procedimientos que reducen el tamaño de un SCP mediante la remoción de filas o columnas redundantes. Los procedimientos más comúnmente usados consideran:

- La eliminación de una columna j tal que existe una columna $k \neq j$ para la cual $I_j \subseteq I_k$ y $c_j \geq c_k$
- La eliminación de una columna j tal que $c_j \geq \sum_{i \in I_j} \text{mín}\{c_k : k \in J_i\}$
- La eliminación de una fila i tal que existe una fila $h \neq i$ para la cual $J_h \subseteq J_i$
- La inclusión en la solución de la columna j cuando $J_i = \{j\}$

Estas reglas se deben aplicar en forma muy cuidadosa, ya que la implementación directa de los correspondientes chequeos puede consumir muchísimo tiempo en casos de gran tamaño. Por otro lado, estos chequeos pueden reducir considerablemente el tamaño del caso [5].

3.1. Relajaciones del Modelo

En general, muchas de las heurísticas más efectivas y de las aproximaciones exactas a SCP están basadas en la solución a la relajación de *Programación Lineal* (PL) del SCP, definida como:

$$\begin{aligned} \text{mín}(z) &= \sum_{j=1}^{n} c_j x_j \\ \text{Sujeto a:} & \\ \sum_{j=1}^{n} a_{ij} x_j &\geq 1 \qquad \forall i \in M \\ 0 \leq x_j &\leq 1 \qquad \forall j \in N \end{aligned}$$

Sin embargo, como la solución exacta de esta relajación por códigos PL de propósito general es típicamente alta en consumo de tiempo, muchos algoritmos SCP recurren a la relajación Lagrangiana combinada con una optimización sub-gradiente en orden de determinar soluciones cercanas al óptimo u del dual de la relajación PL [1], dado por:

$$\text{máx}\left\{ \sum_{i \in M} u_i : \sum_{i \in I_j} u_i \leq c_j (j \in N), u_i \geq 0 (i \in M) \right\}$$

La Relajación Lagrangiana del modelo SCP original se define como: $\forall$ vector $u \in R_+^m$ de multiplicadores Lagrangianos asociados a la restricción del modelo, el sub problema Lagrangiano queda:

$$\begin{aligned} L(u) &= \text{mín} \sum_{j \in N} c_j(u) x_j + \sum_{i \in M} u_i \\ \text{Sujeto a:} & \\ x_j &\in \{0,1\} \qquad j \in N \end{aligned}$$

donde $c_j(u) = c_j - \sum_{i \in I_j} u_i$ es el costo Lagrangiano asociado con la columna $j \in N$. Ahora, una solución óptima al modelo planteado estaría dada por $x_j(u) = 1$ si $c_j(u) < 0$, $x_j(u) = 0$ si $c_j(u) > 0$ y $x_j(u) \in \{0,1\}$ cuando $c_j(u) = 0$. Además, dado u, se puede determinar $x(u)$ en un tiempo $O(q)$. El problema Lagrangiano dual asociado con este modelo consiste en encontrar un vector multiplicador Lagrangiano $u^* \in R_+^m$ que maximice el corte inferior $L(u)$. Como el modelo tiene la propiedad de integralidad, cualquier solución óptima u^* del dual de la relajación PL del SCP es también una solución óptima del problema Lagrangiano dual [6].

[1] con cada modelo PL, existe otro modelo PL dual que tiene el mismo óptimo [15]

Pero este no es el único procedimiento, existiendo en la literatura algunas variaciones al procedimiento anterior. Así, Balas y Carrera [16], en vez de relajar las restricciones correspondientes a todas las filas en M, mantiene en forma explícita en el problema relajado las filas en un subconjunto $\overline{M}$ tal que $J_i \bigcap J_h = 0$, $\forall i,h \in \overline{M}$, $i \neq h$, y relajan de manera Lagrangiana las filas en $M \setminus \overline{M}$. Este problema relajado puede resolverse en un tiempo $O(q)$ y el mejor valor obtenido por esta relajación es igual al valor obtenido a través del problema PL relajado.

Una alternativa a la relajación Lagrangiana es la *relajación sustituta* (surrogate relaxation), utilizada por Lorena en [17]. Para un vector de multiplicadores sustitutos $v \in R_+^m$, el problema de relajación sustituta de un SCP tiene la siguiente forma:

$$\begin{aligned} S(u) &= \text{mín} \sum_{j \in N} c_j x_j \\ \text{Sujeto a:} & \\ \sum_{j \in N} \left(\sum_{i \in I_j} v_i \right) x_j &\geq \sum_{i \in M} v_i \\ x_j &\in \{0,1\} \qquad j \in N \end{aligned}$$

Además, Lorena [17] relaja la restricción $x_j \in \{0,1\}$ y resuelve en un tiempo $O(n)$ la relajación continua del problema. Aunque esta última relajación sigue siendo equivalente a la relajación PL del SCP, los resultados experimentales indicados por Lorena, sugieren que el uso de relajación sustituta en vez de la Lagrangiana dentro de la optimización de sub gradiente, permite obtener multiplicadores cercanos al óptimo en tiempos de computación más pequeños.

Muchos de los métodos presentados hasta ahora son utilizados dentro de los algoritmos basados en heurística. Así, su objetivo principal no es el encontrar el valor más cercano al óptimo, sino dirigir la búsqueda de soluciones cercanas al óptimo en los problemas SCP.

3.2. Algoritmos basados en Heurísticas

Muchos Algoritmos basados en Heurísticas utilizan un enfoque Greedy, ya que les permite ser más simples y más eficientes en términos del tiempo computacional. La idea básica consiste en inicializar una solución S como vacía e igualar el conjunto de filas no cubiertas M' con M. Después, en forma iterativa, la columna j con el mejor valor de un puntaje σ_j es sumado a S y M'se actualiza. El puntaje σ_j es una función del costo original c_j, del número de filas en M' cubiertas por la columna j y por un multiplicador asociado con esas filas. Al final del procedimiento, el conjunto S contiene un conjunto R de columnas redundantes. La eliminación óptima de las columnas redundantes implica la resolución de un SCP definido por las columnas en R y las filas en $M \setminus (U_{j \in S \setminus R} I_j)$.

En otras palabras, se construye una solución comenzando de $S = 0$, incluyendo en S, en cada iteración, el mejor de los subconjuntos restantes. Existen varios criterios para determinar el mejor subconjunto. Estos criterios caracterizan a los diferentes algoritmos.

El esquema general del algoritmo Greedy es el siguiente:

```
procedimiento greedy;
begin
    S := 0;
    while S no cubra a I do
    begin
        seleccionar y eliminar I_j de F;
        S := S ⋃ {I_j};
        reducir(F)
    end
end
```

El procedimiento **reducir**(F) elimina de F aquellos subconjuntos dominados por subconjuntos previamente seleccionados en la solución. La selección de los subconjuntos se puede hacer de acuerdo a los criterios propuestos por Chvátal [18]:

$$I_j = \arg \text{máx} \left\{ \frac{|I_k|}{c_k} : I_k \in F \right\}$$

Otros criterios atribuyen valores diferentes al numerador o al denominador en la formula anterior, dándole más fuerza a la cardinalidad de un subconjunto o a su costo. Para más detalles se puede revisar [10].

Otro Algoritmo basado en heurística es el propuesto por Beasley [9], el cual entrega soluciones cuyos valores están muy cercanos al óptimo. Es una especie de algoritmo Primal-Dual. La idea es considerar el dual Lagrangiano del problema original, donde se relajan las restricciones $\sum_{j=1}^{n} a_{ij}x_j \geq 1$. El problema consiste en:

$$\text{máx}\{\varphi(u) : u \geq 0\}$$

y la función Lagrangiana es

$$\text{mín}\left\{\sum_{j=1}^{n}\left(c_j - \sum_{i=1}^{m} a_{ij}u_i\right)x_j : x_j \in \{0,1\}, j = 1,\ldots,n\right\}$$

El Dual Lagrangiano es aproximado mediante un método de sub-gradiente simple y estándar. En cada iteración, se fijan los multiplicadores u_i en un valor tentativo y se calcula la función Lagrangiana $\varphi(u)$. Después, se actualiza el valor de u y el procedimiento itera hasta que se cumple alguna condición de término. En cada iteración del algoritmo de sub gradiente, se mantiene a mano una solución dual u. Las variables duales u son utilizadas para calcular el *costo reducido* asociado a cada subconjunto de F. La reducción de costos del subconjunto j esta dado por la siguiente expresión:

$$c_j - \sum_{i=1}^{m} a_{ij}u_i$$

Al seleccionar todos los subconjuntos que no tengan reducción de costos positivos, se puede obtener una cobertura posible y si aún hay elementos de I sin cubrir, la solución se completa seleccionando los subconjuntos remanentes de una manera similar al greedy. Finalmente, el algoritmo minimiza la solución, eliminando de la posible solución los subconjuntos redundantes.

El Algoritmo de Lorena [17], por su parte, utiliza la relajación sustituta descrita en la sección anterior e incluye, dentro del procedimiento de optimización del sub-gradiente, una heurística greedy idéntica a la propuesta por Beasley en [9].

Aunque la mayoría de las aproximaciones heurísticas exitosas para el problema SCP están basadas en la relajación Lagrangiana, existen algunas excepciones dignas de mencionar. Entre las más destacadas se encuentran [14] de Beasley y Chu, [19] de Jacobs y Brusco y [20] de Brusco, Jacobs y Thompson. El primero sigue una aproximación genética para solucionar el problema mientras que los dos últimos se basan en simulated annealing.

El algoritmo de Beasley y Chu [14], mantiene una *población* de soluciones del SCP codificadas como vectores x, con valores $0, 1$. Inicialmente, se genera una población aleatoria de p soluciones. Después, en cada iteración, se seleccionan dos soluciones, x^1 y x^2, en forma aleatoria desde la población actual y se combinan en una tercera solución x^3 tal que $x_j^3 = x_j^1$ cuando $x_j^1 = x_j^2$, y $x_j^3 = x_j^1$ con probabilidad $c(x^2)/(c(x^1)+c(x^2))$ si $x_j^1 \neq x_j^2$, donde $c(x)$ es el costo correspondiente a x. Después de la definición de x^3 de x^1 y x^2, se cambian un cierto número de componentes de x^3 seleccionados en forma aleatoria. Después, x^3 es transformado en una solución mínima factible a través de un greedy, en forma análoga al utilizado por Beasley en [9]. Finalmente, la nueva solución x^3 es sumada a la población actual en lugar de la solución previa elegida aleatoriamente. El proceso se detiene luego de un número determinado de iteraciones.

Por otro lado, en [19], se genera una solución inicial S por medio de un algoritmo greedy el cual, en cada iteración, seleccione aleatoriamente una fila no cubierta y agrega a la solución la columna con el menor índice que cubre esa fila. Después de esta adición, se remueven todas las posibles columnas redundantes de la solución y el proceso itera hasta que se determina la solución mínima posible. Posteriormente, se ejecutan un número de iteraciones en las cuales se eliminan de la solución actual S, d columnas elegidas aleatoriamente. A su vez, la solución actual S se completa hasta obtener una nueva solución mínima S' a través de una proceso greedy, en forma análoga al utilizado por Beasley en [9]. S' se vuelve solución actual si es mejor que S, de lo contrario S es reemplazado por S' con una probabilidad que decrece exponencialmente con la diferencia entre los valores de S y S' y con el número de iteraciones realizadas.

Por su parte, en [2], Parada, Calderón, Flores y Pradenas, verificaron el comportamiento de la *búsqueda Tabu* en Problemas de Optimización Combinatoria. Con respecto al SCP, los resultados no fueron muy alentadores. Si bien el método consigue acercarse a la solución óptima conocida, no logra superar la eficiencia de otras estrategias empleadas en este tipo de problemas.

3.3. Algoritmos exactos

Las aproximaciones exactas más efectivas para SCP se basan en algoritmos del tipo Branch-and-Bound [5], en los cuales los cortes inferiores se calculan resolviendo la relajación PL del SCP, posiblemente de una forma de heurística dual como se vio en la sección anterior. La razón principal del éxito de estas aproximaciones es el hecho de que, a pesar que el corte inferior PL no es siempre fuerte para estas instancias, aparentemente es muy difícil lograr cortes inferiores significativamente fuertes por métodos alternativos que son computacionalmente más caros.

Dado que las técnicas completas, como Branch-and-Bound, escapan a los objetivos del presente trabajo, no serán tratadas aquí. Sin embargo, si el lector desea obtener más información respecto a este tipo de algoritmos, puede consultar las siguientes publicaciones: [8], [7] y [16].

4. Modelo matemático

Se tiene el siguiente problema:

> Se dispone de las encuestas sobre los hábitos de lectura de M personas, referidos a N revistas de circulación nacional (se sabe qué revistas leen las personas). Y también se conoce el costo de publicar un aviso en cada una de las revistas. Una nueva empresa, que ha definido como su público objetivo a estas personas, desea optimizar sus costos de publicidad en una o más revistas. Se requiere un modelo Set Covering que les permita resolver las siguientes preguntas:
>
> - Encontrar el máximo número de personas a las que puede llegar un anuncio publicitario dado cierto presupuesto.
> - ¿Cuál es el costo mínimo de llegar a todo el público?

Cada pregunta plantea un modelo distinto, por lo que el problema se divide en dos. Así, es posible formular dos modelos que resolverán cada una de las preguntas.

4.1. Modelo 1

La primera pregunta plantea la incógnita de cuánto es el máximo número de personas a las cuales puedo llegar con mi publicidad dado un cierto presupuesto. Para determinar el modelo, debemos definir lo siguiente:

Datos: Los datos conocidos son:

- Número de Personas : M={1,. . .,m}
- Número de Revistas : N={1,. . .,n}
- Presupuesto disponible: P
- Costo de publicar un aviso en la revista j : $c_j \qquad j \in N$
- Información respecto a qué revistas leen las personas: $A_{ij} = \begin{cases} 1 & \text{si la persona } i \text{ lee la revista } j \\ 0 & \text{si no} \end{cases}$

Variables: Las variables necesarias para modelar el problema son:

- $x_j = \begin{cases} 1 & \text{si se publica en la revista } j \\ 0 & \text{si no} \end{cases}$
- $y_i = \begin{cases} 1 & \text{si } \mathbf{no} \text{ se considera a la persona } i \\ 0 & \text{si no} \end{cases}$

Así, podemos plantear la función objetivo como:

$$\text{mín}(z) = \sum_{i=1}^{m} y_i$$

Sujeto a:

$$\sum_{j=1}^{n} c_j x_j \leq P$$

$$\sum_{j=1}^{n} a_{ij} x_j + y_i \geq 1 \qquad \forall i \in M$$

4.2. Modelo 2

La segunda pregunta plantea la incógnita de cuánto sería el costo mínimo que debo pagar para poder llegar a todo el público. Para determinar el modelo, debemos definir lo siguiente:

Datos: Los datos conocidos son los mismos del punto anterior.

Variables: Las variables necesarias para modelar el problema son:

- $x_j = \begin{cases} 1 & \text{si se publica en la revista } j \\ 0 & \text{si no} \end{cases}$

Así, podemos plantear la función objetivo como:

$$\text{mín}(z) = \sum_{j=1}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad \forall i \in M$$

Para los efectos de este trabajo, resolveremos el Modelo 2 utilizando un algoritmo genético (AG). Para comprobar su efectividad, trataremos de solucionar algunos de los problemas de la OR-Library de J. E. Beasley, que es una Colección de benchmarks (test data sets) que se encuentran en: http://people.brunel.ac.uk/ mastjjb/ jeb/orlib/scpinfo.html.

5. Representación

Al tratar de resolver un problema, lo primero que hay que definir es la representación a utilizar. Por ejemplo, en los modelos del punto anterior se utilizó una representación binaria (el dominio de todas las variables era {0,1}). Así, tenemos un vector de n bits (con n el número de columnas del problema) en donde un 1 en el bit i indica que esa columna (i) es parte de la solución.

Por definición, los AG utilizan representación binaria, sin embargo, esta representación tiene el problema que al aplicarle los operadores genéticos a los individuos, los resultados no siempre serán soluciones factibles. Esto obliga a aplicar una función de penalización en la función de evaluación para que quede representado el grado de no satisfacción de la solución. Otra forma de resolver este problema es aplicando una heurística que "arregle" la solución no factible, convirtiéndola en una factible.

Pero esta no es la única representación que se puede utilizar. Si definimos un vector de largo igual al número de filas y en cada posición guardamos el número de la columna que cubre dicha fila, tendremos una representación que resuelve el problema de crear soluciones no factibles por la utilización de los operadores genéticos. Sin embargo, como una misma columna puede cubrir más de una fila, se debe crear un método que elimine esta redundancia para la evaluación de su fitness[2]. Es decir, al momento de aplicar la función de evaluación, solo se cuenta una vez cada columna. Lamentablemente, la evaluación del fitness se vuelve ambigua ya que la misma solución puede ser representada de diferentes formas y cada forma puede dar un valor distinto de fitness dependiendo de cómo fue definido el vector.

[2]Fitness: valor obtenido de la función de evaluación. Mide que tan buena es la solución analizada. En ocasiones, coincide con el valor de la función objetivo.

Beasley [14] efectuó pruebas que indicaron que el rendimiento de un Algoritmo Genético utilizando una representación no binaria fue más bajo que el mismo utilizando una representación binaria. Por tal motivo, en nuestra implementación se utilizó la representación binaria junto con una función que toma en consideración las soluciones no factibles. Dicha función se explicará más adelante.

6. Descripción del Algoritmo

Una vez definida la representación, se puede comenzar el desarrollo del algoritmo que resolverá el problema. Para entender mejor el trabajo realizado, primero se explicará, en forma breve, las características principales de un AG, para luego explicar cómo se aplican dichas características al problema en cuestión.

6.1. El Algoritmo Genético estándar

Los Algoritmos Genéticos (AG) son atribuidos a los estudios efectuados por Holland [21] y son Meta-Heurísticas que imitan el proceso de evolución de la naturaleza. Los AG estándar, por su parte, no fueron desarrollados para solucionar problemas de optimización y por lo tanto no consideran las restricciones que pueda tener un problema determinado. Sin embargo, con pequeñas modificaciones, se pueden obtener muy buenos resultados en problemas de optimización.

Los AG copian el proceso evolucionario mediante el procesamiento simultaneo de una población de soluciones. Por lo general, se comienza con una población generada de manera aleatoria, cuyos individuos se van recombinando de acuerdo al valor que tengan en la función de evaluación. Así, mientras mejor sea el valor obtenido por el individuo, más posibilidades tiene de ser seleccionado para generar nuevas soluciones. De esta forma, las nuevas generaciones de soluciones conservan lo bueno de las generaciones anteriores permitiendo, después de un proceso iterativo, llegar a un valor que eventualmente podría ser el óptimo.

Para hacer las combinaciones, el AG estándar posee dos operadores genéticos que son: el cruzamiento y la mutación. El primero toma dos individuos y forma un tercero mediante la combinación de los componentes de los individuos padres, permitiendo de esta forma que el algoritmo efectúe una explotación del espacio de búsqueda. Por otra parte, la mutación toma un individuo y por medio de un proceso genera otro individuo para la nueva población. Esto permite que el algoritmo efectúe una exploración del espacio de búsqueda.

En líneas generales, un AG estándar se puede ver como un proceso que efectúa lo siguiente:

Algoritmo Genético
begin
    Generar una población inicial;
    Evaluar el fitness de los individuos de la población;
    repetir:
        Seleccionar individuos padres desde la población;
        Aplicar los operadores genéticos para generar nuevos individuos;
        Evaluar el fitness de los individuos hijos;
        Reemplazar algunos o todos los individuos de la población por los hijos;
    hasta condición de término
end

Para implementar un AG es necesario definir los siguientes puntos:

Representación:Este es el punto de partida en la definición de un AG. De esta definición va a depender gran parte de la estructura final del AG. Como comentamos anteriormente, los AG utilizan una representación binaria, pero existen variaciones como los Algoritmos Evolutivos que permiten otros tipos de representaciones.

Tratamiento de los individuos no factibles:Dado que los AG no están diseñados para trabajar con restricciones, es necesario definir un criterio o un mecanismo para tratar los individuos no factibles.

Inicialización:Se refiere a cómo se va a generar la población inicial. Generalmente, dicha población se genera de manera aleatoria.

Condición de término:Se debe establecer hasta cuando va a correr el algoritmo. Generalmente se utiliza un número determinado de generaciones (iteraciones) o una medida de convergencia.

Funciones de Evaluación o Fitness:Es la función que va a determinar que tan buena es una solución. La función de evaluación puede ser igual o diferente a la función objetivo.

Operadores Genéticos:Los operadores genéticos son los encargados de efectuar la reproducción de la población, siendo los más comunes el cruzamiento y la mutación.

Selección:Se refiere a como se van a seleccionar los individuos para aplicarles los operadores genéticos. La selección debe dirigir el proceso de búsqueda en favor de los miembros más aptos.

Reemplazo:Este punto se refiere a cómo se va a reemplazar la población con los individuos generados, lo cuál puede diferir del cómo se seleccionan. Es decir, los criterios de selección y reemplazo no tienen por qué ser iguales.

Parámetros de funcionamiento:Un AG necesita que se le proporcionen ciertos parámetros de funcionamiento, tales como el tamaño de la población, la probabilidad de aplicación de los operadores genéticos y la cantidad de generaciones, entre otros.

Una vez definidos los puntos anteriores, se tiene una estructura adecuada para comenzar el desarrollo y la implementación del AG.

6.2. Algoritmo propuesto

Basándonos en lo anteriormente descrito y en los trabajos de Beasley [14] y Cormier [22], se desarrolló un Algoritmo Genético para resolver el problema de Set Covering teniendo los datos de los archivos de la OR-Library de J.E.Beasley como entrada. Para efectos del algoritmo, se definieron las siguientes variables:

$$
\begin{aligned}
I &= \text{el conjunto de todas las filas} \\
J &= \text{el conjunto de todas las columnas} \\
a_i &= \text{el conjunto de columnas que cubren la fila } i,\ i \in I \\
b_j &= \text{el conjunto de filas cubiertas por la columna } j,\ j \in J \\
S &= \text{el conjunto de columnas en una solución} \\
U &= \text{el conjunto de filas no cubiertas} \\
w_i &= \text{el número de columnas que cubren la fila } i,\ i \in I \text{ en } S \\
c_j &= \text{costo de la columna } j
\end{aligned}
$$

A continuación se describe el cómo se desarrollaron los distintos puntos que definen un AG.

6.2.1. Representación

Como se indicó anteriormente, lo primero que se debe definir es la representación que se va a utilizar. En nuestro caso utilizaremos la representación binaria ya que es la representación más natural para el problema de set covering. Además, del trabajo de Beasley se desprende que la representación binaria es la más adecuada para este problema. Así, tendremos un vector de igual largo que la cantidad de filas a ser cubiertas. La Tabla 1 muestra un ejemplo de dicha representación.

columna (gen)	1	2	3	4	5	$\cdots$	$n-1$	n
string de bits	0	1	0	0	1	$\cdots$	1	0

Tabla 1: Representación binaria de un individuo

6.2.2. Proceso previo

Una vez definida la representación y dada la estructura de los archivos de la OR-Library, se debió programar una rutina que tomara los datos de un archivo y transformara dicha información a una representación binaria. La función *inivar()* es la encargada de tomar los datos desde un archivo determinado y crear las matrices a_i, b_j y el vector de costos c, lo que permite que el algoritmo trabaje con esos datos. El nombre del archivo de entrada es indicado al programa a través del teclado.

6.2.3. Tratamiento de los individuos no factibles

Las soluciones modificadas por los operadores genéticos no siempre se van a mantener factibles, por lo que hay que aplicar algún método o función que trabaje con las soluciones no factibles. Para considerar lo anterior, se probaron dos estrategias. La primera esta basada en el trabajo de [14] y consiste en aplicar una heurística que arregla una solución no factible y la convierte en una factible. Dicha heurística se implementa en la función *arregla()* y provee, además, un paso de optimización local con el fin de hacer al AG más eficiente. Los pasos requeridos para hacer factible una solución pasa por determinar cuales filas aún no han sido cubiertas y escoger las columnas necesarias para su cobertura. La búsqueda de dichas columnas se basa en la proporción:

$$\frac{\text{costo de una columna}}{\text{cantidad de filas } \textit{no cubiertas} \text{ que cubre}}$$

Una vez que la solución se ha vuelto factible, se aplica un paso de optimización que elimina aquellas columnas redundantes. Una columna redundante es aquella que si se quita, la solución sigue siendo factible. La heurística efectúa, a grandes rasgos, lo siguiente:

procedimiento arregla();
begin
Inicializar $w_i := |S \cap a_i|, \forall i \in I$;
Inicializar $U := \{i | w_i = 0, \forall i \in I\}$;
Para cada fila i en U (en orden creciente de i):
encontrar la primera columna j (en orden creciente de j) en a_i, que minimice $\frac{c_j}{U \cap b_j}$;
incluir j en S y hacer $w_i := w_i + 1, \forall i \in b_j$. Hacer $U := U - b_j$;
Para cada columna j en S (en orden decreciente de j), si $w_i \geq 2, \forall i \in b_j$, hacer $S := S - j$ y $w_i := w_i - 1, \forall i \in b_j$;
Ahora S es una solución factible del problema que no contiene columnas redundantes.
end

La segunda estrategia utilizada, fue el aplicar una función de penalización en la función de evaluación, la cual incrementa el valor de esta última de acuerdo a la cantidad de filas que no sean cubiertas. El problema principal de esta técnica es que es difícil encontrar un valor adecuado para el factor de penalización.

6.2.4. Inicialización

El tamaño de la población, así como la población inicial son elegidos de manera tal que el dominio de las soluciones asociadas con la población este cubierto adecuadamente. El tamaño de la población depende de cómo se genere la población inicial. En nuestro caso, la población inicial se generó aleatoriamente utilizando un método que genera una solución factible y sin columnas redundantes. Dicho método lo implementa la función *inicia()* de acuerdo a lo siguiente:

procedimiento inicia();
begin
Inicializar $w_i := 0, \forall i \in I$;
Inicializar $S := 0$;
Para cada fila i en I:
seleccionar en forma aleatoria una columna j desde a_i;
incluir j en S y hacer $w_i := w_i + 1, \forall i \in b_j$;
Hacer $T := S$;
repetir
Seleccionar, al azar, una columna j, con $j \in T$, y hacer $T := T - j$;
Si $w_i \geq 2, \forall i \in b_j$, hacer $S := S - j$ y $w_i := w_i - 1$, $\forall i \in b_j$;
hasta que $T = 0$.
end

El tamaño de la población se determinó en el proceso de sintonización del AG y será explicado más adelante.

6.2.5. Condición de término

Los AG poseen básicamente dos condiciones de término: número máximo de generaciones y convergencia en solución. En el primer caso, el algoritmo itera un número determinado de veces, luego se detiene y entrega el mejor valor que tenga hasta ese momento. En el segundo caso, el algoritmo itera hasta que los individuos de la población converjan en un único valor.

En nuestro caso, se eligió el número máximo de generaciones como condición de término.

6.2.6. Función de Evaluación

Como se dijo anteriormente, para trabajar con soluciones no factibles se utilizaron dos estrategias. En la primera, utilizando una heurística que repara las soluciones, las funciones de evaluación y objetivo son iguales y son implementadas a través de la función *evalúa()*. Así, la función de evaluación queda:

$$\sum_{j=1}^{n} c_j * x_j$$

De la misma manera, en la segunda estrategia, donde se utiliza una función de penalización, a la función objetivo se le agrega un factor proporcional a la cantidad de filas no cubiertas. De esta forma, la función de evaluación queda:

$$\sum_{j=1}^{n} (c_j * x_j + (k * \text{factor de penalización}))$$

Dicho factor de penalización debe tener un valor adecuado de forma tal que afecte al resultado final. Es decir, si el valor de dicho factor es muy pequeño y el problema es de minimización, no habrá diferencias notables en la función de evaluación respecto a una solución válida. Lo anterior es implementado por la función *eval()*.

6.2.7. Operadores Genéticos

Los operadores genéticos utilizados en el desarrollo del algoritmo fueron el cruzamiento y la mutación. En ambos casos se probaron dos estrategias distintas para comprobar el funcionamiento del algoritmo.

En el caso del cruzamiento, primero se implementó el operador de fusión propuesto por Beasley [14] en la función *cruzam()*, la cual toma dos individuos padres y genera un solo individuo hijo. Su funcionamiento es el siguiente: dadas las soluciones P_1 y P_2, con sus respectivos valores de fitness f_{P_1} y f_{P_2} y el individuo hijo C, se tiene $\forall i = 1, \ldots, n$:

begin
Si $P_1[i] = P_2[i]$, entonces $C[i] := P_1[i] = P_2[i]$;
Si $P_1[i] \neq P_2[i]$, entonces:
$C[i] := P_1[i]$ con probabilidad $p = \frac{p_2}{p_1+p_2}$;
$C[i] := P_2[i]$ con probabilidad $1 - p$;
end

Lo destacable de este operador es que considera los valores de fitness para determinar que elementos se copian en el hijo, logrando pasar información de mayor valor a las descendencias.

Por otro lado, se implementó un operador de cruzamiento en un punto en la función *cruzamiento()*. Esta función toma dos individuos de la población, elige un punto de cruce al azar, intercambia los elementos iniciales entre los individuos y genera dos individuos hijos.

Por su parte, el operador de mutación se implementó por la función *mutación()* con una probabilidad fija y con una variable. Esta ultima va aumentando de acuerdo a la cantidad de generaciones que van pasando y posee un valor máximo definido por un coeficiente. Esto le permite al algoritmo efectuar más exploración a medida que avanza en generaciones.

En ambos casos, la función de mutación toma un individuo, y de acuerdo con la probabilidad de mutación cambia sus componentes: si hay un 0 lo cambia por un 1 y si hay un 1, lo cambia por un 0.

6.2.8. Selección

Los operadores genéticos necesitan que se les entregue uno o dos individuos para procesarlos. Esto hace necesario que se defina un criterio de selección de los individuos. En nuestro caso, la selección se efectuó de acuerdo al método de la ruleta, en donde los individuos son seleccionados de acuerdo a una probabilidad determinada a partir de la función de fitness. Así, un individuo con mayor valor de fitness tendrá más posibilidades de ser seleccionado. Esta idea fue implementada en la función *selección()*. Por su parte, dado que el problema de set covering es de minimización, la probabilidad que un individuo sea seleccionado esta determinado por:

$$p_i = \frac{\frac{1}{f_i}}{\sum_{i=1}^{m} \frac{1}{f_i}}$$

Donde f_i es el valor obtenido por la solución i en la función de evaluación.

6.2.9. Reemplazo

Una vez generada la nueva población con los individuos hijos, se evalúan y pasa completa como población de padres a la nueva generación.

6.2.10. Parámetros de funcionamiento

El AG posee varios parámetros de funcionamiento entre los que se encuentran:

- Tamaño de la población (TAMPOB)
- Número máximo de generaciones (MAXGEN)
- Probabilidad de aplicación del operador de cruzamiento (PCRUZA)
- Probabilidad de aplicación del operador de mutación (PMUTA)
- Factor de penalización (PENA)

Además de los anteriores, se pueden variar la forma de evaluación y los operadores de cruzamiento y mutación.

Una vez codificado el algoritmo, los parámetros de funcionamiento fueron determinados por medio de la sintonización del AG. Es decir, se ejecutó el algoritmo varias veces y de acuerdo a los resultados obtenidos se fueron modificando los valores de los parámetros.

6.2.11. Funcionamiento general del Algoritmo

En líneas generales el AG desarrollado tiene el siguiente funcionamiento:

procedimiento AG;
begin
  Generar una población inicial de N soluciones aleatorias. Hacer $t = 0$;
  repetir
    Seleccionar 2 soluciones P_1 y P_2 de la población utilizando el método de la ruleta;
    Combinar P_1 y P_2 para formar una nueva solución C utilizando el operador de cruzamiento;
    Mutar k columnas seleccionadas de C en forma aleatoria, donde k esta determinado
    por la probabilidad de aplicación del operador de mutación;
    Hacer C válida y remover las columnas redundantes en C aplicando el operador de heurística;
    Hacer $t = t + 1$;
  hasta que $t = M$ **soluciones se hayan generado**. La mejor solución encontrada es la con
  el menor fitness en la población.
end

El algoritmo entrega el archivo *salida.txt* con los datos de columnas incluidas en la solución y valor de la solución.

7. Experimentos

Las primeras ejecuciones del AG tuvieron por finalidad sintonizar el algoritmo. La Tabla 2 muestra las principales ejecuciones realizadas con este fin con los respectivos valores de salida registrados. El archivo de entrada utilizado para la sintonización fue el **scp41.txt**.

Ejecución	MAXGEN	TAMPOB	PCRUZA	PMUTA	f()evaluación	PENA	salida
1	10	10	0.95	0.10	arregla	-	4395
2	10	20	0.90	0.10	penaliza	5000	4064
3	10	30	0.87	0.15	arregla	-	3051
4	20	30	0.85	0.15	penaliza	10000	2955
5	20	40	0.80	0.20	penaliza	10000	2848
6	20	50	0.75	0.20	arregla	-	2940
7	30	50	0.70	0.25	penaliza	10000	2885
8	30	60	0.75	var	arregla	-	2536
9	30	70	0.80	var	penaliza	10000	2467
10	40	70	0.85	var	arregla	-	2591
11	40	80	0.87	var	arregla	-	2341
12	40	90	0.87	var	arregla	-	2466
13	50	90	0.87	0.15	penaliza	10000	2034
14	50	100	0.87	0.15	arregla	-	1964
15	50	110	0.87	0.15	penaliza	10000	1625
16	50	100	0.87	var	arregla	-	930
17	50	110	0.87	var	arregla	-	441
18	1000	10000	0.87	var	arregla	-	430

Tabla 2: Principales ejecuciones realizadas para sintonizar el AG

Adicionalmente se probó el algoritmo con parámetros intermedios entre el 17 y el 18, pero la eficiencia no mejoró lo suficiente como para que valiese la pena utilizar dichos valores. Lo anterior, dado los altos tiempos de ejecución necesarios. Por ejemplo, la ejecución del algoritmo con los parámetros del punto 18 duro 20 horas y 36 minutos, contra los 23 minutos que demoró en correr con los parámetros del punto 17.

Finalizada la sintonización, se definieron los parámetros de funcionamiento del algoritmo de acuerdo a lo siguiente:

- Tamaño de la población (TAMPOB): 110
- Número máximo de generaciones (MAXGEN): 50
- Probabilidad de aplicación del operador de cruzamiento (PCRUZA): 0.87
- Probabilidad de aplicación del operador de mutación (PMUTA): variable
- Tipo de evaluación: con arreglo de soluciones no factibles.
- Tipo de operador de Cruzamiento: con operador de fusión.

8. Resultados

Con los parámetros anteriormente descritos, se ejecutó el AG para los sets de problemas de la OR-Library 4, 5 y 6. Se efectuaron 10 ejecuciones por cada problema y se guardó el mejor valor obtenido. La Tabla 3 muestra un listado de los problemas, con sus valores óptimos conocidos y los mejores resultados obtenidos mediante el AG.

Problema	Valor Óptimo	Mejor solución
4.1	429	430
4.2	512	547
4.3	516	538
4.4	494	517
4.5	512	586
4.6	560	615
4.7	430	435
4.8	492	501
4.9	641	734
4.10	514	519
5.1	253	276
5.2	302	325
5.3	226	268
5.4	242	253
5.5	211	223
5.6	213	227
5.7	293	302
5.8	288	299
5.9	279	281
5.10	265	279
6.1	138	158
6.2	146	148
6.3	145	145
6.4	131	145
6.5	161	163

Tabla 3: Resultados obtenidos para los diferentes problemas

9. Conclusiones

- Se puede observar que, si bien sólo se alcanzó el óptimo en el problema 6.3, los valores entregados por el AG se acercan bastante a él en la mayoría de los problemas.
- En general, los tiempos de ejecución no son menores, por lo que queda pendiente el efectuar una revisión de la implementación con el objeto de mejorar la eficiencia y exactitud del AG.
- El AG desarrollado permite variar una serie de parámetros, entre los que destacan la función de evaluación y el operador de cruzamiento. Lo anterior, permite ajustar de mejor manera el funcionamiento del algoritmo de acuerdo a los distintos tipos de problemas que se deseen resolver.
- A pesar de los largos tiempos de ejecución, el AG propuesto es una alternativa válida para la resolución de problemas de Set covering.

Referencias

[1] M.R. Garey and D.S.Johnson. Computers and intractability: A guide to the theory of np-completeness. *W.H. Freeman and Co., San Francisco*, 1979.

[2] M. Flores L. Pradenas V. Parada, M. Calderon. Resolución de problemas combinatoriales mediante busqueda tabu. Technical report, Depatamento de Ingenieria Informatica, Universidad de Santiago de Chile.

[3] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *33rd IEEE Symposium on Foundations of Computer Science.*, 1992.

[4] M. Fischetti A. Caprara and P. Toth. A heuristic method for the set covering problem. *Proc. of the Fifth IPCO Conference, Lecture Notes on Computer Science.*, 1084:72–84, 1995.

[5] M. Fischetti A. Caprara and P. Toth. Algorithms for the set covering problem. *Annals of Operations Research 98*, pages 353–371, 2000.

[6] M.L. Fisher and P. Kedia. Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science*, 36:674–688, 1990.

[7] J.E. Beasley and K. Jornsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research.*, 58:293–300, 1992.

[8] J.E. Beasley. An algorithm for set covering problems. *European Journal of Operational Research.*, 31:85–93, 1987.

[9] J. E. Beasley. A lagrangean heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.

[10] E. Balas and A. Ho. Set covering algoritms using cutting planes, heuristics and subgradient optimization. a computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[11] F.J. Vasko and G.R. Wilson. An efficient heuristic for large set covering problems. *Naval Research Logistics Quarterly*, 31:163–171, 1984.

[12] L.W. Jacobs and M.J. Brusco. A simulated annealing-based heuristic for the set covering problem. *Working paper, Operations Management and Information Systems Department, Dekalb, IL 60115, USA*, 1993.

[13] S. Sen. Minimal cost set covering using probabilistic methods. *ACM/SIGAPP Symposium on Applied computing*, pages 157–164, 1993.

[14] J. E. Beasley and P.C. Chu. A genetic algoritm for the set covering problem. 1994.

[15] J.J.Franken. Algorithmic modeling and complexity. Lecture Notes., 2003.

[16] E. Balas and M.C. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44:875–890, 1996.

[17] L.A.N Lorena and F.B. Lopez. A surrogate heuristic for set covering problems. *European Journal of Operational Research*, 79:138–150, 1994.

[18] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[19] L.W. Jacobs and M.J. Brusco. A local search heuristic for large set-covering problems. *Naval Research Logistics*, 52:1129–1140, 1995.

[20] L.W.Jacobs M.J.Brusco and G.M.Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated weighted set-covering problems. *Working Papers*, 1996.

[21] J. Holland. Adaptation in natural and artificial systems. *Ann Arbor, University of Michigan Press.*, 1976.

[22] D. Cormier and S.S.Raghavan. A very simple real-coded genetic algorithm. *North Carolina State University and University of North Carolina at Charlotte.*