Metaheurísticas Multinivel para la Resolución del Set Covering Problem

Claudio Valenzuela and Broderick Crawford

Pontificia Universidad Católica de Valparaíso, Chile. FirstName.Name@ucv.cl

Abstract. Las Metaheurísticas son métodos de solución que combinan procedimientos de mejora local y estrategias de alto nivel para resolver problemas combinatoriales y de optimización no lineal. En general, las metaheurísticas requieren una cantidad importante de esfuerzo enfocado en el *setting* de parámetros a fin de mejorar su rendimiento. Se propone un enfoque multinivel de tal manera que *Scatter Search* y *Ant Colony Optimization* actúan como metaheurísticas de "bajo nivel" cuyos parámetros son ajustados por un Algoritmo Genético de "alto nivel" durante la ejecución, buscando mejorar el rendimiento y reducir el mantenimiento. El problema de *Set Covering* se toma como referencia dado que es uno de los problemas de optimización más importantes, el cual sirve como base para problemas de ubicación de instalaciones, cronograma de tripulaciones de líneas aéreas, turnos de enfermería y ubicación de recursos.

Keywords. Optimización Combinatorial, Metaheurísticas, Genetic Algorithm, Scatter Search, Ant Colony Optimization, Set Covering Problem.

1 Introducción

The Set Covering Problem (SCP) es un problema clásico en el área de las ciencias de la computación y es uno de los problemas de optimización discretos más importantes, puesto que puede modelar convenientemente problemas del mundo real. Algunos de estos problemas — los cuales pueden ser modelados como un problema de set covering — incluyen problemas de ubicación de instalaciones, cronograma de tripulaciones de líneas aéreas, turnos de enfermería, repartición de recursos, balanceo de línea de ensamblaje, enrutamiento de vehículos, entre otros [11].

Existen varios estudios que han implementado soluciones para SCP usando metaheurísticas [2, 1, 7, 8]. Dependiendo de los algoritmos utilizados, la calidad de la solución esperada y, por supuesto, la complejidad del SCP elegido, se define la cantidad de esfuerzo en personalización requerido. Convenientemente, este trabajo propone transferir parte de estos esfuerzos de personalización hacia otra metaheurística (metaheurística de "alto nivel"), la cual puede manejar las tareas de ajuste de parámetros sobre una metaheurística de bajo nivel. Se considera que este enfoque es de multinivel, puesto que existen dos metaheurísticas — no

necesariamente del mismo tipo — cubriendo tareas de *setting* de parámetros, para la primera, y resolución del problema, para la última [6].

El diseño principal de la implementación propuesta considera tener un Algoritmo Genético, en inglés Genetic Algorithm (GA) [13], a cargo del *setting* de parámetros online (Control) y offline (Tuning) para una metaheurística de bajo nivel (Ant Colony Optimization o Scatter Search, cada uno en distintas instancias), usando un enfoque de Búsqueda Reactiva y de Tuning de Parámetros Automático. En la Búsqueda Reactiva, los mecanísmos de feedback son capaces de modificar los parámetros de búsqueda de acuerdo a la eficiencia del proceso de búsqueda, i.e. el balance entre intensificación y diversificación, el cual se puede automatizar explotando el pasado reciente del proceso de búsqueda a través de técnicas de aprendizaje dedicadas [15]. El Tuning de Parámetros Automático se lleva a cabo mediante un algoritmo externo, el cual busca los mejores parámetros dentro de un espacio de parámetros, a fin de ajustar el *solver* automáticamente.

Las técnicas Ant Colony Optimization (ACO) y Scatter Search (SS) [9] han mostrado interesantes resultados al resolver SCP [11] y problemas similares [12]. Para el propósito de este trabajo, el primero es elegido por su enfoque constructivo al generar soluciones junto con sus operadores basados en la estocástica. El último es considerado como un algoritmo evolutivo (basado en poblaciones) que usa, esencialmente, operadores determinísticos. Ambos demuestran ser buenas metaheurísticas de referencia en cuanto a sus fundamentos, sus enfoques al solucionar problemas, la madurez de sus diseños, y con respecto a cuán diferente es uno del otro, lo que los hace adecuados para el desarrollo de este trabajo.

2 Set Covering Problem

Un modelo matemático general para el Set Covering Problem puede ser formulado de la siguiente manera:

$$(1)\ \ Minimizar\, Z = \sum_{j=1}^{n} c_j x_j \qquad j = \{1, 2, 3, ..., n\}$$

Sujeto a:

$$(2)\ \ \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = \{1, 2, 3, ..., m\}$$

$$(3)\ \ x_j = \{0, 1\}$$

La ecuación (1) es la función objetivo del problema de set covering, donde c_j se refiere a los pesos o costos de cubrir la j-ésima columna, y x_j es la variable de decisión. La ecuación (2) es una restricción que asegura que cada fila es cubierta por al menos una columna, donde a_{ij} es una matriz de coeficientes de restricción de tamaño m x n cuyos elementos se componen de tanto "1" ó "0". Finalmente, la ecuación (3) es la restricción de valores enteros, en la cual el valor x_j puede ser "1" si la columna j está activada (seleccionada) o "0" de otro modo.

Diferentes métodos han sido propuestos en la literatura para resolver el problema de set covering. Existen ejemplos usando métodos exactos [4], programación lineal y métodos heurísticos [5], y métodos metaheurísticos [2]. Se ha señalado que, una de las aplicaciones de SCP más relevantes está relacionada con problemas de turnos de tripulaciones en empresas de transporte masivo, donde un conjunto definido de viajes debe ser cubierto po un conjunto de "asociaciones" de costo mínimo, siendo una asociación una secuencia de viajes que pueden ser ejecutados por una sola tripulación.

3 Enfoque Multinivel

3.1 Diseño GA-SS

El diseño propuesto para la implementación multinivel se basa en los estándares generalmente propuestos para cada metaheurística. Ambas metaheurísticas búscan ser lo más cercano posible a sus orígenes. Obviamente, una implementación multinivel con control de parámetros adaptativo fuerza algunos cambios en la metaheurística de "alto nivel", en este caso, AG. Este caso será similar en ambas combinaciones; AG-SS y AG-ACO. Por lo tanto, el diseño de AG se acerca bastante a su diseño básico.

[*Size of P , BestSet , DiverseSet , EnhanceTrials , MaxSol*]

Fig. 1. Representación de cromosoma de AG para SS.

En la representación arriba, el primer gen, en este caso "*Size of P*" es el número de soluciones iniciales de SS, los cuales tomarán valores entre 1 y $n/2$ donde n es el número de variables. De la misma manera, el segundo gen "*BestSet*" es el tamaño del Conjunto de las Mejores Soluciones. Similarmente, "*DiverseSet*" es el tamaño del Conjunto de Soluciones más Diversas. "*EnhanceTrials*" representa el número e de de intentos para el Método de Mejora, para intentar mejorar una solución, donde $e = \{1...n\}$. Finalmente, "*MaxSol*" es el límite de soluciones generadas por una llamada del algoritmo Scatter Search.

3.2 Componentes AG

A continuación se muestra una descripción de los componentes de AG, los cuales están incluidos en el diseño de AG-SS:

- Función de Inicialización: Inicializa los valores de los genes dentro de los límites para cada variable. Además, inicializa (a cero) todos los valores de fitness para cada miembro de la población. Toma los límites superior e inferior para cada variable desde los parámetros definidos por el usuario, y genera valores aleatorios dentro de estos límites, para cada gen dentro de cada genotipo en la población.

- Función de Evaluación: La metaheurística de nivel superior dispone de dos criterios; (1) la misma función de evaluación de la metaheurística de nivel inferior, la cual es la *función objetivo* correspondiente del problema de set covering, y (2) una *Función Objetivo* del problema de set covering penalizada por el esfuerzo en procesamiento. Este último enfoque parece ser el más justo, puesto que "valores más grandes" para cada parámetro de SS permitirán, obviamente, obtener mejores resultados (puesto que un mayor número de soluciones es revisado).
- Función "Guarda el Mejor": Esta función guarda el rastro del mejor miembro de la población. Se hace notar que la última entrada en el arreglo Población mantiene una copia del mejor individuo.
- Función Elitista: El mejor miembro de la generación previa es almacenado como el último en el arreglo. Si el mejor miembro de la generación actual es peor que el mejor miembro de la generación previa, el último deberá reemplazar el peor miembro de la generación actual.
- Función de Selección: Una selección proporcional estándar para problemas de maximización/minimización incorporando un modelo elitista — asegura que el mejor miembro sobreviva.
- Selección de Crossover (Cruzamiento): Elige los dos padres que participarán del cruzamiento. Implementa un cruzamiento en un solo punto.
- Mutación: Corresponde a mutación uniforme aleatoria. Una variable seleccionada para mutación es reemplazada por un valor aleatorio entre los límites superior e inferior de esta variable.

3.3 Componentes de SS

Se presenta una descripción de los componentes de SS, los cuales están incluidos en el diseo AG-SS, a continuación:

Método de Generación de Diversificación: Este método está diseñado basándose en la propuesta en [10] para generar soluciones iniciales binarias diversas. Empieza con una solución arbitraria de solo ceros, para luego añadir "1" en cada posición con un salto de k bits, donde $k = \{1...n\}$ y $n < (number\_of\_vars) - 1$. Por ejemplo, para $k = 1$ generar un vector de solución $[1, 0, 1, 0, 1, 0, 1, 0...]$. Se debe notar que este método siempre comienza ubicando un bit "1" en la primera posición. Junto con la solución previamente generada, el método genera el complemento de aquella solución, el cual sería $[0, 1, 0, 1, 0, 1, 0, 1, 0, 1...]$. La cantidad de soluciones generadas por este método es un parámetro controlado por el AG.

Método de Mejora: transforma una solución tentativa en una solución tentativa mejorada. Si la solución tentativa no es factible, debe ser reparada hasta que sea factible. Las soluciones de entrada no requieren ser factibles. Si la solución tentativa de entrada no es mejorada como resultado de la aplicación de este método, la solución "mejorada" se considera como la misma solución de entrada. El límite de los intentos de mejora (no los intentos de reparación) son controlados por el AG.

Las tareas de reparación y mejora son repotenciadas puesto que un vector de ratios es calculado (el largo del vector es el número de variables, o *columnas*).

Este vector es ordenado de menor a mayor ratio, y cada miembro representa una variable en la función objetivo. Luego, al intentar reparar una solución, este vector debe ser accedido en-orden, puesto que el primer item representará la columna la cual cubre más filas en la matriz de incidencia. Si la posición es "0" en la solución, debe ser cambiada a "1", puesto que el método busca añadir "1" para cubrir filas y luego llevar la solución a la factibilidad. Este proceso continúa hasta que la solución es factible.

Al intentar mejorar una solución, el vector de rations debe ser accedido en post-orden, intentando convertir tantos "1" a "0" como sea posible mientras la solución se mantenga factible. El método intentará mejorar hasta que el límite de l intentos es alcanzado, donde l es un parámetro controlado por el AG.

Método de Actualización del Conjunto de Referencia: construye y mantiene dos conjuntos de referencia, los cuales consisten en b "mejores" soluciones y d soluciones "*más diversas*" encontradas (donde el valor de b y d son definidos por el AG). Los conjuntos se organizan para proveer un acceso eficiente a las otras partes del proceso de solución. El criterio para agregar las "mejores" soluciones a su conjunto correspondiente, es el costo de la función objetivo. Adicionalmente, el criterio para agregar soluciones al conjunto de las "más diversas" es la distancia de Hamming a todas las soluciones dentro del conjunto de las "mejores" y de las "más diversas".

Método de Generación de Subconjuntos: opera en el conjunto de referencia para producir un subconjunto de soluciones como base para crear soluciones combinadas. Se usará el método más común de generación de subconjuntos, el cual genera pares de todas las soluciones de referencia (i.e. todos los subconjuntos de tamaño 2) desde el conjunto de "mejores" y de las "más diversas" al mismo tiempo.

Método de Combinación de Soluciones: transforma un subconjunto dado de soluciones, producido por el Método de Generación de Subconjuntos, en una solución combinada. El método de combinación está basado en el ratio *costo en la función objetivo / filas cubiertas*, el cual es calculado por columna. Al combinar dos soluciones, el siguiente procedimiento es usado:

$Sol_1[x] = Sol_2[x]$ $newSol[x] = Sol_1[x]$
$Ratio[x] < median$ $newSol[x] = 1$
$newSol[x] = 0$

donde x es la posición del bit siendo evaluado con $x = \{1...maxColumns\}$, y $median$ es la mediana del vector de ratios para una instancia determinada de SCP.

3.4 Diseño AG-ACO

Para esta implementación, la misma estructura que para AG presentada en 3.2 será usada. Adicionalmente, la implementación de ACO para SCP es bastante directa. A continuación, los componentes especiales usados por ACO serán revisados:

Feromona. Denotada por $t = \tau_i$, la matriz de feromonas será usada para consolidar la información obtenida por cada hormiga, i.e. la cantidad de feromona almacenada en cada columna i. $\tau_i(t)$ especifica la intensidad de la feromona en la columna i en tiempo t, siendo actualizada de manera local (de acuerdo al recorrido de cada hormiga), y de manera global (la feromona se evapora en cada columna de la matriz).

La matriz es inicializada con un valor t_0 el cual será 10^{-6} para esta implementación.

Transición. Para realizar una transición entre dos columnas se debe hacer de acuerdo a la lista de candidatos. La lista de candidatos para una columna contiene las c columnas más atractivas, las cuales son ordenadas de manera secuencial. La transición se lleva a cabo de acuerdo a:

Si existe al menos una columna $j \in$ lista de candidatos, luego, elegir la siguiente columna j donde $j \in J_i^k$ entre las c columnas en la lista de candidatos con:

$$j = \begin{cases} max_u \in J_i^k[\tau_u(t)][\eta_u]^\beta & \text{if } q \leq q_0 \\ J & \text{if } q > q_0 \end{cases}$$

donde η_u representa la información heurística, y $j \in J_i^k$ es elegido con la probabilidad:

$$p_i^k(t) = \frac{[\tau_i(t)][\eta_i]^\beta}{\sum_{l \in N^k}[\tau_l(t)][\eta_l]^\beta}$$

donde N^k es un posible vecino si la hormiga k está compuesta por todas aquellas columnas las cuales cubren al menos una fila no cubierta,

sino, elige $j \in J_i^k$ con el menor valor.

Después de cada elección de cada columna i ocurre una modificación local del nivel de feromona de aquella columna, dada por la ecuación:

$$\tau_i(t) = (1 - \rho)\tau_i(t) + \rho\tau_0$$

Esta evaporación es hecha para que la columna visitada no sea interesante para las hormigas que le siguen, estimulando de esta forma la exploración de soluciones. En la ecuación de arriba, $\rho\tau_0$ es un factor de estabilización de la modificación de feromona, usada con la intención de no tener columnas poco atractivas tan pronto, permitiendo la exploración de un número mayor de soluciones.

Cuando cada hormiga termina una iteración (una solución se ha encontrado), el nivel de feromona de la mejor solución es actualizado globalmente, procediendo a cargar feromona a cada columna de acuerdo a:

$$\tau_i(t) = (1 - \rho)\tau_i(t) + \rho\triangle\tau_i(t)$$

donde $\triangle\tau_i(t)$ es la variación de feromona dejada en la columna i por la mejor hormiga. Esta variación es calculada como la frecuencia de la columna i en el recorrido de cada hormiga, i.e. el número de veces donde la columna i está en las soluciones encontradas por las hormigas.

Para operar, los siguientes parámetros controlados por AG serán definidos:

- "Number of Ants" (número de hormigas)
- ρ: factor de evaporación
- β: importancia en la elección de la próxima columna
- *Length of the List (largo de la lista)*: la cual es usada para limitar el número de columnas que pueden ser visitadas a continuación
- Q_0: parámetro que indica si la exploración es soportada al momento de la elección de la próxima columna
- *MaxIter*: número máximo de iteraciones

En general, el procedimiento ACO en seudocódigo es llamado de la misma forma que el procedimiento de SS.

3.5 Función de Penalización

Una ayuda a la función de setting de parámetros es necesaria para que el Algoritmo Genético pueda manejar un trade-off entre continuar mejorando una solución — ergo más recursos son aplicados — o dejar la solución y e intentar otra búsqueda. La función de penalización es una muy buena solución a este problema puesto que la comparación directa es entre la versión con Tuning de Parámetros y la versión con Control de Parámetros. Como función de penalización, se propone lo siguiente:

$$fitness(x_1) = ObjFuncValue(x_1) + TimeTaken(x_1) * FCT$$

donde x_1 es una solución y $ObjFuncValue$ es el valor — o costo — de la solución generada evaluada en la Función Objetivo de la instancia correspondiente de SCP. Adicionalmente, $TimeTaken$ es la cantidad de tiempo que tomó el generar esa solución.

3.6 Consideración a la Implementación con Control de Parámetros

En este punto, algunas diferencias claves entre las implementaciones deben ser tomadas en cuenta. Puesto que dos tipos de procedimientos de setting de parámetros son considerados para cada configuración, se entregará una explicación más en detalle al respecto, en esta sección. Los diseños presentados en la sección 3.1 y 3.4 fueron programados directamente — sin grandes cambios hechos — en la versión de Tuning de Parámetros. Para obtener

Control de Parámetros en AG-SS Para este caso, el AG fue modificado para permitir una "memoria intermedia" de resultados. En las siguientes transiciones de SS, el AG recibe feedback del desempeño de la metaheurística de bajo nivel:

- Después de cada $P/2$ llamadas a la función de mejora durante la ejecución del procedimiento DivGen. Exclusivamente los parámetros de SS: K, J y *EnhanceTrials*, son considerados para ser mutados por el operador correspondiente de AG dependiendo en el desempeño en curso.
- Después de cada $BestSet*DiverseSet/2$ llamadas a la función de mejora durante la ejecución del Método de Combinación. Exclusivamente los parámetros de SS: *BestSet* y *DiverseSet*, son considerados para ser mutados por el operador correspondiente de AG dependiendo en el desempeño en curso.

Control de Parámetros en AG-ACO Habilitar la configuración de AG-ACO para el Control de Parámetros es una tarea más directa que en el caso de AG-SS. Los siguientes puntos detallan los cambios hechos:

- Después de cada iteración completada en la heurística de bajo nivel (ACO) — se ha enviado *"Number of Ants (Número de Hormigas)"* — los parámetros ρ, β, *"Lenght of the List (Largo de Lista)"*, Q_0 y *"Number of Ants"* son mutados dependiendo en el rendimiento de la mejor hormiga de la iteración.

3.7 Criterio de Evaluación para la Implementación

Puesto que el análisis del rendimiento de las metaheurísiticas es un aspecto clave en este trabajo — y tal como se presentó anteriormente — varias instancias de código serán probadas, alejándose de las propuestas puramente teóricas.

De acuerdo a [14], los siguientes puntos deben ser completados para asegurar un análisis de rendimiento riguroso:

Diseño Experimental Como primer paso, los objetivos de los experimentos, las instancias y los factores elegidos, deben ser definidos. Los objetivos para esta experimentación buscan evaluar el rendimiento — como un todo — entre las versiones de Control de Parámetros y Tuning de Parámetros de cada configuración (AG-SS y AG-ACO). Los parámetros principales a modificar serán los parámetros del AG: Probabilidad de Cruzamiento y Probabilidad de Mutación. Está fuera del propósito de este trabajo, el probar un espectro más amplio de cambios principalmente debido a que las metaheurísticas multinivel buscan ajustarse así mismas con la mínima intervención humana. Por lo tanto, el impacto del cambio de esos parámetros será comparado y expuesto.

Algunas instancias "preparadas" serán usadas para prubas, las cuales son librerías estándar bien conocidas, provistas en OR-Library. Se explica más al respecto en la sección 3.7. No se usaron instancias para calibrar los parámetros de ninguna forma puesto que no hay necesidad de ello, nuevamente, el tuning de parámetros de AG fue mínimo y neutral.

Mediciones Como segundo paso, las medidas a calcular son elegidas. Después de ejecutar los distintos experimentos, el análisis estadístico es aplicado a los resultados obtenidos. El análisis de rendimiento debe ser hecho con algoritmos de optimización estado-del-arte, dedicados para el problema. Las medidas de rendimiento serán:

- Mejor Solución Encontrada (valor en la Función Objetivo de SCP)
- Tiempo Requerido para encontrar la Mejor Solución (mejor representada por "Llamadas a la Función Objetivo hechas antes de encontrar la mejor solución")
- Llamadas Totales a la Función Objetivo
- Fitness Promedio (penalizado) y su Desviación Estándar

La calidad de las soluciones es tomada midiendo su cercanía al valor óptimo de una instancia determinada (archivo SCP de OR-Library). El fitness promedio es también una ayuda al evaluar la calidad en combinación con su desviación estándar.

El esfuerzo computacional es medido por: el número de llamadas a la función objetivo necesarias para generar la mejor solución, y las llamadas totales a la función objetivo a través del proceso completo. Como se hizo notar anteriormente, las comparaciones son fundamentalmente "internas".

Benchmarks de Referencia Tercero y final, los resultados son presentados en una manera comprensiva, y un análisis es llevado a cabo siguiendo los objetivos definidos. Otro tema principal acá es el asegurar la *reproducibilidad* de los experimentos computacionales.

Cada implementación — i.e. AG-SS y AG-ACO — será probada usando los benchmarks provistos por [3] los cuales son ampliamente usados por los Investigadores del campo de las Operaciones y la Optimización. Los archivos usados son las instancias de Set Covering Problem denotadas por las series SCP4x.txt, SCP5x.txt y SCP6x.txt. Para obtener más resultados experimentales — y comparar el rendimiento de cada implementación — tales benchmarks serán probados contra ambas versiones de configuraciones de setting de parámetros.

Los benchmarks presentados son problemas de minimización, donde la idea es encontrar el menor costo en la Función Objetivo. Cada benchmark se basa en una matriz de 200 filas y 1000 columnas, lo cual los convierte, con cierta justicia, en un problema de tamaño grande.

4 Resultados

4.1 Entorno

Todos los algoritmos necesarios para testins serán programados bajo el lenguaje C estándar y compilados con GNU GCC. La IDE Xcode 3.2 fue elegida para las tareas de depuración de código, y como computador de referencia se uso

un CPU Intel Core 2 Duo con 4Gb de RAM DDR3. Se hace notar que la el código implementado no hace uso de núcleos múltiples de CPU aunque esto sea soportado por el hardware.

4.2 Robustez basada en los parámetros de AG

La primera tabla de esta sección muestra los resultados para AG-SS con Tuning de Parámetros. Estos resultados son importantes puesto que cumplen dos objetivos: permitir evaluar las medidas con esta configuración de setting de parámetros offline, y representar la ejecución de varias metaheurísticas únicas (o de un solo nivel). Como se ha explicado anteriormente, una configuración de Tuning de Parámetros ejecuta una instancia completa de la metaheurística SS con parámetros definidos — por AG — antes de su ejecución. Cuando la ejecución de SS termina, los mejores resultados son entregados como feedback al AG, el cual puede continuar el proceso de experimentación con otros parámetros de SS. Por lo tanto, los resultados obtenidos consideran una configuración de parámetros neutral — una de tipo aleatorio — representando un testing de tipo prueba y error hecha por humanos, donde los mejores resultados pueden ser tan buenos como los encontrados con AG-SS, los cuales seguramente requerirán más trabajo humano en términos de definir aquellos parámetros y experimentar con ellos.

PXOVER	PMUT	BestFound	CallsOF	Average	Fit-StDev
0.25	0.25	1009	5778578	3017	433
0.5	0.5	1009	4698886	2915	675
0.75	0.75	1007	5750974	3104	571
0.25	0.75	1008	4545205	2803	458
0.75	0.25	1009	4698886	2644	578

Table 1. Distintas configuraciones de parámetros de AG-SS, usando Tuning de Parámetros. Benchmark: SCP41.

Los primeros resultados (Tabla 1) muestran una variabilidad considerable. El aspecto general representa lo que un humano puede esperar al experimentar — sin considerar la buena suerte. Como se verá en las siguientes tablas, *PXOVER* y *PMUT* tienden a perder su impacto, siendo esta instancia la más inesperada.

En esta segunda tabla (Tabla 2) los mejores resultados encontrados no muestran variación, principalmente debido a la Función de Mejora de SS. Las otras medidas muestran una baja desviación estándar, especialmente en las llamadas totales a la Función Objetivo. Como es reportado en la solución promedio, los resultados son los que presentan la mayor variación, entonces, existe cierta lógica al pensar que existe un buen nivel de exploración entre espacios solución.

En la próxima tabla (Tabla 3) el rendimiento de ACO es evaluado usando Tuning de Parámetros. Aparte de los mejores resultados obtenidos con ACO —

PXOVER	PMUT	BestFound	CallsOF	Average	Fit-StDev
0.25	0.25	509	3446206	2235	1075
0.5	0.5	509	3462702	2329	1078
0.75	0.75	509	3481599	2002	954
0.25	0.75	509	3502315	1735	797
0.75	0.25	509	3308283	1980	890

Table 2. Distintas configuraciones de parámetros de AG-SS, usando Control de Parámetros. Benchmark: SCP41.

PXOVER	PMUT	BestFound	CallsOF	Average	Fit-StDev
0.25	0.25	449	1477894	781	236
0.5	0.5	434	1095816	768	240
0.75	0.75	449	1659306	896	274
0.25	0.75	449	1599972	862	260
0.75	0.25	449	1435961	762	220

Table 3. Distintas configuraciones de parámetros de AG-ACO, usando Tuning de Parámetros. Benchmark: SCP41.

y su mejor manejo de recursos — la misma figura que en AG-SS es observada: variación muy baja en cada medida.

Usando un criterio de setting de parámetros online (Tabla 4) muestra que no hay mayores variaciones en los resultados; en realidad, una variación bastante baja es obtenida. Es claro que los resultados bajo está configuración son los mejores de todos. Lo anterior será revisado en la siguiente muestra de experimentos.

PXOVER	PMUT	BestFound	CallsOF	Average	Fit-StDev
0.25	0.25	436	673804	442	7
0.5	0.5	436	691756	442	8
0.75	0.75	436	743647	444	8
0.25	0.75	436	671342	444	8
0.75	0.25	436	703318	441	7

Table 4. Distintas configuraciones de parámetros de AG-ACO, usando Control de Parámetros. Benchmark: SCP41.

4.3 Convergencia a la mejor solución

Estos experimentos fueron elegidos entre los resultados más interesantes: los cuales presentaron una convergencia notoria a través del tiempo. A primera vista, la tentación es hacia comparar lo obvio; AG-ACO se desempeña mejor que sus contrapartes. Después de un análisis minucioso, entendemos que la comparación es injusta fundamentalmente porque ACO usa una manera más “inteligente” de

construir sus soluciones, donde SS es más ciego. Una conclusión preliminar es que el proceso constructivo de ACO es mejor que el evolutivo de SS. De cualquier manera, el propósito de las gráficas es permitir observar el cómo cada versión converge en el tiempo hacia una mejor solución.

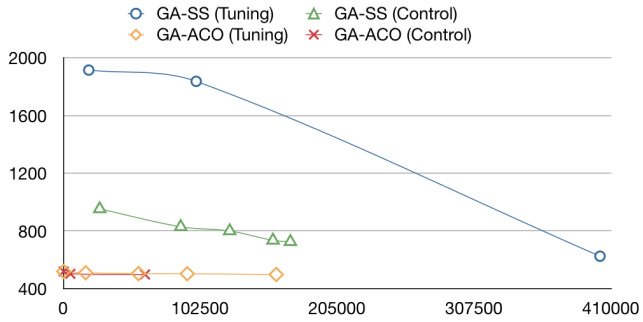

Fig. 2. Análisis de convergencia a una mejor solución. Benchmark: SCP48

El primer gráfico se basa en el benchmark *scp48* y la idea más clara es que el Control de Parámetros permite obtener mejores soluciones en menor tiempo, lo cual se debe principalmente a la intensificación de la explotación de ciertos espacios solución donde los resultados con "mejor aspecto" son encontrados, y a la vez, porque se ahorra tiempo al cancelar la exploración de resultados "de mal aspecto". Además, un item en común en cada gráfico es que AG-SS (Tuning) tiende a obtener mejores resultados que AG-SS (Control), cuando grandes cantidad de soluciones son exploradas.

El segundo y tercer gráfico confirman lo que fue visto en el primer gráfico, Control de Parámetros en SS realmente produce una diferencia en el rendimiento, permitiendo acercarse al rendimiento de ACO, lo cual es un gran salto.

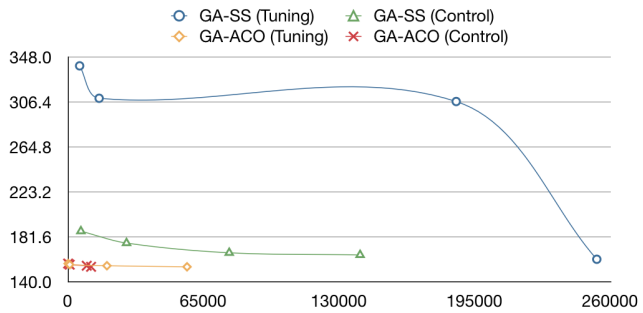

Fig. 3. Análisis de convergencia a una mejor solución. Benchmark: SCP62

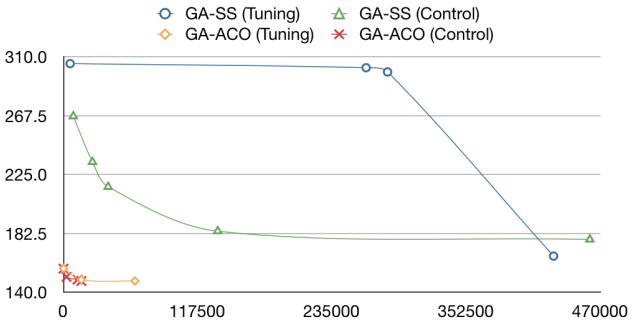

Fig. 4. Análisis de convergencia a una mejor solución. Benchmark: SCP63

4.4 Comparación amplia de los mejores resultados encontrados

La siguiente tabla presenta los mejores resultados encontrados a través de las distintas instancias ejecutadas. La convergencia más rápida de las versiones con Control de Parámetros ha sido considerada en las secciones previas — lo cual no será representado en esta tabla. Los mismos resultados obtenidos en ambas versiones de ACO ocultan evidentemente el factor de eficiencia. Cada instancia ejecutada para obtener los resultados, se llevo a cabo usando $PXOVER = 0.5$ y $PMUT = 0.5$ el cual parece un trade-off justo entre cruzamiento y mutación en AG. Bajo estas circunstancias, otro criterio de robustez es manejado: desempeñarse bien en una variedad de instancias. Quien se desempeño peor a través de todos los experimentos fue AG-SS con Tuning de Parámetros, pero, incluso a pesar de ésto, no fue un desempeño tan malo.

	scp41	scp42	scp48	scp61	scp62	scp63
Optimum	429	512	492	138	146	145
GA-SS-Offline	1007	981	561	154	155	166
GA-SS-Online	509	603	642	164	165	176
GA-ACS-Offline	434	529	497	142	154	148
GA-ACS-Online	434	529	497	142	154	148

Table 5. Table of best results.

4.5 Comparación contra metaheurísticas únicas

En *Tuning Before Solving*, y para los practicantes buscando una comparación directa, parece bastante lógico considerar que la mejor representación del rendimiento de una metaheurística es lograda al trabajar con Tuning de Parámetros, por ejemplo:

– Un practicante puede empezar con un conocimiento específico muy bajo de qué parámetros pudieran ser mejores al intentar resolver una instancia de un problema, ergo, probablemente se obtendrán resultados pobres en la ejecución de la metaheurística. Esto quedará representado por parámetros de bajo rendimiento elegidos aleatoriamente por la metaheurística de alto nivel.
– Como ser humano inteligente, el practicante podría notar que algunos parámetros han mostrado mejores resultados, lo cual queda representado cuando la metaheurística de alto nivel intenta mantener los parámetros de "buen aspecto" a lo largo de las iteraciones.
– Eventualmente, el practicante puede adquirir suficiente conocimiento para ajustar los mejores parámetros para una metaheurística en una determinada instancia de un problema. Lo anterior, es representado por una metaheurística de alto nivel y su mejor conjunto de parámetros encontrados — los cuales en el caso de AG es un cromosoma siendo mantenido a través de las generaciones.

5 Conclusiones

Los enfoques tradicionales de metaheurísticas requieren un control de parámetros adecuado. En este trabajo se presentó un enfoque multinivel basado en dos metaheurísticas, donde una (bajo nivel) resuelve el Set Covering Problem y otra (alto nivel) se encarga de ajustar los parámetros de la primera. Aunque los enfoques multinivel son muchas veces menos sensibles a los cambios en estos parámetros, no existe garantía en que una implementación multinivel funcionará igualmente bien en distintos problemas usando el mismo setting de parámetros. Se reitera que uno de los principales objetivos de un enfoque multinivel es proveer un método de resolución no atendido, a fin de producir, rápidamente, soluciones de buena calidad para distintas instancias del Set Covering Problem.

Los benchmarks han mostrado interesantes resultados en términos de robustez de cada enfoque, siendo el Control de Parámetros el más robusto en términos de buen rendimiento frente a varias instancias usando los mismos parámetros, y en términos de resultados estables frente a pequeñas variaciones en los valores de los parámetros. Además, el Control de Parámetros muestra una convergencia más rápida hacia mejores resultados que el Tuning de Parámetros, pero, cuando ejecuciones prolongadas son llevadas a cabo, ambos parecen obtener resultados similares.

Considerando el desempeño particular del enfoque constructivista y del evolutivo, para este tipo de problemas (SCP) parece más efectivo el "construir" inteligentemente una solución en vez de obtener una "a ciegas" y luego evolucionarla. Un hecho interesante es que SS con Control de Parámetros puede obtener un desempeño similar a ACO.

Finalmente, se considera que la implementación general de metaheurística multinivel es bastante directa; no se encontraron mayores problemas y el Algoritmo Genético — al usar cromosomas con codificación de números reales —

no fue, en ningún momento, un obtáculo para los experimentos. Además, como se hizo uso de números reales, se logra una representación más directa de qué parámetros un humano usaría para ejecutar la heurística de bajo nivel, logrando una implementación que satisface los requerimientos del proyecto.

References

1. U. Aickelin. An indirect genetic algorithm for set covering problems. *Journal of the Operational Research Society*, 53:1118–1126, 2002.
2. U. Aickelin. An indirect genetic algorithm for set covering problems. *CoRR*, abs/0803.2965, 2008.
3. J. E. Beasley. Or library, 2010.
4. J. E. Beasley and K. Jornsten. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58(2):293–300, April 1992.
5. A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:2000, 1998.
6. C. Cotta, M. Sevaux, and K. Sörensen, editors. *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*. Springer, 2008.
7. B. Crawford and C. Castro. Integrating lookahead and post processing procedures with aco for solving set partitioning and covering problems. In *ICAISC*, pages 1082–1090, 2006.
8. B. Crawford, C. Lagos, C. Castro, and F. Paredes. A evolutionary approach to solve set covering. In J. Cardoso, J. Cordeiro, and J. Filipe, editors, *ICEIS (2)*, pages 356–363, 2007.
9. G. A. K. Fred Glover. *Handbook of metaheuristics*. Springer, 2003.
10. F. Glover. A template for scatter search and path relinking. In *Artificial Evolution*, pages 1–51, 1997.
11. D. Gouwanda and S. G. Ponnambalam. Evolutionary search techniques to solve set covering problems. *World Academy of Science, Engineering and Technology*, 39:20–25, 2008.
12. R. Martí and M. Laguna. Scatter search: Diseño básico y estrategias avanzadas. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 19:123–130, 2003.
13. Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.
14. E.-G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.
15. F. S. Youssef Hamadi, Eric Monfroy. What is autonomous search?, msr-tr-2008-80, 2008.