An Ant Colony Optimization Algorithm for solving the Generalized Steiner Problem

Martín Pedemonte y Héctor Cancela

{mpedemon,cancela}@fing.edu.uy INCO, Facultad de Ingeniería, UdelaR

Abstract. The Generalized Steiner Problem models the design of high reliable communications networks, demanding a variable number of independent paths between pairs of terminals. GSP solutions are built using intermediate nodes to ensure redundancy between paths, while trying to minimize the network's cost. The GSP is an NP-hard problem that has previously been tackled with some metaheuristics, though not covering the entire range of possible proposals with this approach. In this work, we present the evaluation of an Ant Colony Optimization algorithm for solving several instances of the GSP. The comparative study shows that the proposed technique is able to obtain superior results to those so far obtained for the studied instances.

1. Introducción

El auge de Internet, así como el desarrollo de la infraestructura de redes ha renovado el interés en los problemas de diseño de redes de comunicaciones. Dentro de los problemas de diseño de redes de comunicaciones, uno de los que concita mayor interés consiste en obtener una topología de conexión de los nodos de una red cuyas propiedades aseguren la comunicación confiable de datos. Como el crecimiento del tamaño de las redes es continuo, las instancias subyacentes de los problemas de optimización que surgen frecuentemente superan la capacidad de los algoritmos exactos. Es este contexto, las técnicas metaheurísticas se han aplicado con éxito al diseño de redes de comunicaciones confiables para resolver instancias de dimensiones reales en tiempos razonables.

Este artículo presenta la aplicación de Ant Colony Optimization (ACO) a la resolución de los problemas de diseño de redes de comunicaciones confiables que pueden ser modelados por el Problema de Steiner Generalizado (GSP). Dada una red con algunos nodos distinguidos, el GSP consiste en diseñar una subred de costo mínimo que verifique requisitos prefijados de conexión entre pares de los nodos distinguidos. En general, son intereses contrapuestos la reducción del costo y el aumento en la confiabilidad de la red. Por ejemplo, un modelo que solamente minimice el costo de la red, sin incorporar redundancia entre los caminos conduce a topologías incapaces de soportar fallos en los nodos o enlaces. El GSP incorpora requisitos de conectividad adicionales que aumentan la confiabilidad de la red demandada al considerar escenarios realistas. El problema ha sido poco abordado con técnicas metaheurísticas; exisitiendo enfoques exitosos con algoritmos evolutivos. Este trabajo presenta un análisis empírico de la resolución de

instancias del GSP de un conjunto de casos de prueba usado previamente en la literatura, representativo de redes de tamaño mediano.

La estructura del artículo es la siguiente. La sección 2 presenta la metaheurística ACO. La sección 3 describe el GSP y sus variantes, su formulación y trabajos relacionados con su resolución. El algoritmo ACO propuesto para la resolución del GSP se comenta en la sección 4. La sección 5 presenta los casos de prueba considerados, así como los resultados obtenidos y su análisis. Finalmente, se comentan las conclusiones de este trabajo y las líneas de trabajo futuro.

2. Ant Colony Optimization

Ant Colony Optimization [3] es una metaheurística poblacional basada en la utilización de hormigas artificiales para construir soluciones mediante el agregado de componentes a una solución parcial. Estos componentes se seleccionan a partir de considerar información heurística del problema resuelto y rastros de feromona que reflejan la experiencia adquirida durante la búsqueda.

El Algoritmo 1 presenta el esquema de un algoritmo ACO aplicado a un problema estático de optimización combinatoria. En primer lugar, se inicializan los valores de los rastros de feromona. Luego, se itera hasta cumplir un cierto criterio de parada. Cada paso de la iteración está dividido en cuatro etapas. En la primera etapa cada hormiga construye una solución en forma concurrente, independiente y asíncrona. La construcción se realiza seleccionando componentes mediante una regla probabilística que considera la experiencia adquirida durante la búsqueda (a través del rastro de feromona) e información heurística de los componentes considerados (conocida como visibilidad). La segunda etapa es opcional y consiste en la aplicación de un algoritmo de búsqueda local para mejorar las soluciones. En la tercera etapa, los valores de los rastros de feromona son decrementados por la evaporación e incrementados por el depósito de feromona en las componentes usadas en la construcción de soluciones; el cambio neto depende de la contribución de estos dos procesos de actualización. En la última etapa, si es necesario se actualiza la mejor solución obtenida hasta el momento.

La comunidad científica ha propuesto múltiples variantes que instancian el esquema presentado en el Algoritmo 1. En este trabajo usamos la variante Hyper-Cube Framework for ACO (HCF-ACO) combinada con $\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS). HCF-ACO [1] es un mecanismo general para el manejo de la actualización de los rastros de feromona que puede aplicarse a cualquier variante de ACO. Se trabaja con valores escalados de los rastros de forma que permanezcan en el intervalo [0,1]. En particular, elegimos aplicar HCF-ACO a $\mathcal{MM}$AS [12] ya que es una de las variantes que presenta los mejores resultados.

HCF-$\mathcal{MM}$AS utiliza la regla proporcional aleatoria para agregar componentes a la solución parcial s^p. Esta regla es presentada en la ecuación 1, donde c_{i,x_i} es un componente de una solución que corresponde a combinar una variable con uno de los valores de su dominio y $J(s^p)$ es el conjunto de componentes que se pueden agregar a la solución parcial. Los parámetros α y β ajustan la influencia relativa de los rastros de feromona τ_{i,x_i} y de la visibilidad η_{i,x_i}.

Algoritmo 1 ACO para un problema estático de optimización combinatoria

```
T = initializePheromoneTrails()
s_best = s | f(s)= +∞
while not stopCriteria() do
   pop = constructAntsSolutions(T)
   pop'= applyLocalSearch(pop)   % opcional
   T = updatePheromones(T,pop')
   s = selectBestOfPopulation(pop')
   if f(s) < f(s_best) then
      s_best = s
   end if
end while
return s_best
```

$$p(c_{i,x_i}|s^p) = \begin{cases} \frac{[\tau_{i,x_i}]^\alpha [\eta_{i,x_i}]^\beta}{\sum_{c_{j,x_j} \in J(s^p)} [\tau_{j,x_j}]^\alpha [\eta_{j,x_j}]^\beta} \text{ si } c_{i,x_i} \in J(s^p) \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{en otro caso} \end{cases} \tag{1}$$

$\mathcal{MM}$AS y por tanto HCF-$\mathcal{MM}$AS tiene limites explícitos τ_{min} y τ_{max} para la cantidad de feromona en un componente, asegurando que se cumple $\tau_{min} \leq \tau_{i,x_i} \leq \tau_{max}$. El rastro de feromona se actualiza de acuerdo a la ecuación 2, donde s_{best} es la mejor solución (de la iteración o hasta el momento) y ρ la tasa de evaporación de feromona ($0 \leq \rho \leq 1$).

$$\tau_{i,x_i} \leftarrow (1-\rho) * \tau_{i,x_i} + \rho \Delta\tau_{i,x_i}^{s_{best}} \; \forall \tau_{i,x_i} \in T, \; \Delta\tau_{i,x_i}^{s_{best}} = \begin{cases} 1 \text{ si } c_{i,x_i} \in s_{best} \\ 0 \text{ en otro caso} \end{cases} \tag{2}$$

HCF-$\mathcal{MM}$AS no requiere el recalculo de τ_{min} y τ_{max} cuando se encuentra una solución mejor, ya que sus valores se conocen a priori (0 y 1, respectivamente). Blum y Dorigo [1] sugieren inicializar los rastros de feromona en 0.5 porque da la misma importancia a incluir o no los componentes en la solución.

3. Problema de Steiner Generalizado

En esta sección se presenta una descripción del GSP y otras variantes de la clase de problemas de Steiner. La sección también incluye los trabajos relacionados a la resolución del GSP mediante metaheurísticas.

3.1. Formulación del GSP

La formulación del GSP considera los siguientes elementos [2]:

- un grafo no dirigido $G = (V, E)$, donde V es el conjunto de nodos y E es el conjunto de aristas,

- una matriz de costos C asociados a las aristas del grafo G,
- un subconjunto T de nodos distinguidos, llamado conjunto de nodos terminales (o simplemente terminales), tal que $2 \leq |T| \leq |V|$,
- una matriz simétrica $R = \{r_{ij} | i, j \in T\}$ de dimensiones $|T| \times |T|$, cuyos elementos $r_{ij} \in \mathbb{Z}^+$ indican el número de caminos disjuntos requeridos entre cada par de terminales i y j.

El GSP consiste en encontrar un subgrafo de costo mínimo G_T de G, tal que para todo par de terminales $i, j \in T$ existan r_{ij} caminos arista-disjuntos entre los nodos i y j en G_T. No hay requisitos específicos de conexión para los nodos que no pertenecen al conjunto de terminales. Estos nodos conocidos como *nodos de Steiner* pueden formar parte o no de la solución óptima. También existe una variante del problema en la que los caminos disjuntos no deben compartir nodos.

El GSP pertenece a la clase de problemas de Steiner. Esta clase incluye otros problemas que simplifican las condiciones de conectividad impuestas entre los pares de terminales. En particular, los problemas de k-conexión requieren una cantidad común fija k de caminos disjuntos entre pares de terminales. Un caso especial de problema de k-conexión es el problema del árbol de Steiner (STP). El STP requiere solamente la existencia de un único camino entre pares de terminales, y su solución toma forma de un árbol que cubre el subconjunto de terminales al menor costo posible.

El GSP [2] e incluso el STP [5] pertenecen a la clase de problemas NP-difíciles. La complejidad de los problemas de Steiner los vuelve intratables con algoritmos exactos al incrementarse el tamaño de las instancias consideradas, limitando su aplicabilidad. Como consecuencia, los problemas de diseño de redes de la vida real no pueden ser afrontados con algoritmos exactos. Por el contrario, las metaheurísticas son ideales para este tipo de escenarios ya que permiten encontrar soluciones casi óptimas en tiempos razonables.

3.2. Trabajos relacionados

Robledo [11] propuso el primer ACO para el GSP. En su trabajo, para cada requisito de conexión de cada par de terminales se obtenía el camino más corto usando un ACO independiente, a partir de las aristas disponibles, privilegiando los caminos que atraviesan una mayor cantidad de terminales ante costos similares. Los resultados obtenidos fueron limitados y la escalabilidad al considerar instancias de tamaño creciente fue pobre.

Nesmachnow et al. [8] estudiaron los algoritmos Genetic Algorithms, Simulated Annealing, CHC (Cross generational elitist selection, Heterogeneous recombination and Cataclism mutation) y un híbrido GA+SA para la resolución del GSP. Los algoritmos propuestos usan representación binaria, operadores que no incorporan conocimiento específico del problema y descartan las soluciones no factibles. Los resultados muestran que el CHC obtuvo las mejores soluciones.

Recientemente, Nesmachnow y Pedemonte [9] evaluaron los algoritmos Particle Swarm Optimization (PSO) y ACO para el GSP. El PSO propuesto seguía

un enfoque similar para la representación y los operadores al trabajo de Nesmachow et al. [8]. En el ACO propuesto, cada hormiga iterativamente incorporaba aristas a su solución construyendo los caminos necesarios para los pares de terminales. Los resultados obtenidos fueron buenos pero de menor calidad que los obtenidos por el algoritmo CHC antes comentado.

4. Algoritmo ACO para el GSP

Esta sección presenta un algoritmo ACO para la resolución del GSP. En primer lugar, se describe el mecanismo de construcción de las soluciones usado por las hormigas. Luego, se comentan los detalles de la construcción de caminos para las soluciones. Finalmente, se presenta un método de búsqueda local diseñado para mejorar las soluciones construidas.

4.1. Mecanismo de construcción de las soluciones

Cada hormiga construye su solución procesando iterativamente los requisitos de conectividad. Cuando se procesa un par de terminales correspondientes a un requerimiento de conectividad, se intenta reutilizar las aristas que forman parte de la solución parcial para evitar incrementar el costo de la solución. El Algoritmo 2 presenta el esquema del proceso de construcción de soluciones.

En el Algoritmo 2, s es la solución construida por la hormiga, M es la matriz con los requisitos de conectividad, E es el conjuntos de aristas, E_a es el conjunto de aristas disponibles para construir caminos entre el par de terminales, E_u es el conjunto de aristas usado en la construcción de un nuevo camino por la subrutina `buildPath` o en los caminos que ya están presentes en la solución para el par de terminales y E_p es el conjunto de aristas *privilegiadas*. Las aristas *privilegiadas* son aquellas que forman parte de la solución pero que no han sido usadas para construir caminos entre el par de terminales considerados.

Algoritmo 2 Proceso de construcción de una solución

```
s = empty()
M = getRequiments()
for all i, j ∈ T|i ≠ j do
   M(i,j) = M(i,j)-inducedDisjointPaths(s,i,j)
   E_u = edgesInducedDisjointPaths(s,i,j)
   E_a = removeEdges(E,E_u)
   E_p = removeEdges(s,E_u)
   while M(i,j) >0 do
      E_u = buildPath(i,j,E_a,E_p)
      E_a = removeEdges(E_a,E_u)
      M(i,j) = M(i,j) - 1
      s = addEdges(s,E_u)
   end while
end for
```

Cuando una hormiga procesa un nuevo par de terminales, calcula cuantos caminos disjuntos existen en la solución parcial mediante `inducedDisjointPaths`. Si el número de caminos es mayor o igual que el número de requerimientos de conectividad, la solución ya los satisface; en otro caso, se determinan las aristas usadas mediante `edgesInducedDisjointPaths`. Para determinar el número de requerimientos satisfechos y las aristas usadas se utiliza el algoritmo de Ford-Fulkerson [4]. Si se considera un terminal como origen y el otro como destino y se asume capacidad unitaria en cada arista, el flujo máximo entre el origen y el destino determina el número máximo de caminos disjuntos entre esos nodos.

Si existen requerimientos insatisfechos para el par de terminales, las aristas obtenidas con la subrutina `edgesInducedDisjointPaths` se marcan como usadas (E_u) y no son consideras para la construcción de caminos disjuntos para este par. Si hay aristas que no fueron usadas para el par de terminales considerado pero que forman parte de la solución parcial, se marcan como *privilegiadas* (E_p) porque su uso en la construcción del resto de los caminos es conveniente, ya que no incrementan el costo de la solución. Luego, la hormiga construye un camino entre el par de terminales usando las aristas disponibles (E_a) y *privilegiadas* a través de la subrutina *buildPath*. Para asegurar que los caminos construidos entre los terminales sean disjuntos, las aristas de cada camino construido por la hormiga son marcadas como usadas y eliminadas de las disponibles.

El esquema de construcción descrito no requiere controles de factibilidad pero el uso del algoritmo de Ford-Fulkerson incrementa la complejidad computacional. Sin embargo, el algoritmo propuesto explota información presente en la solución parcial que está siendo construida, considerando las aristas *privilegiadas*, evitando resolver requerimientos que ya están satisfechos y minimizando el número de caminos disjuntos que cada hormiga construye para cada par de terminales.

4.2. La subrutina `buildPath`

El mecanismo para construir caminos entre pares de terminales es una generalización de una idea usada por Luyet et al. [6] para la resolución del STP. Cada hormiga encuentra los K caminos más cortos sin ciclos entre los pares de terminales considerados y elige entre ellos usando el esquema de trabajo de *HCF-$\mathcal{MM}AS$*. Para encontrar los K caminos más cortos se usa el algoritmo clásico de Yen [13]. En caso de empate en el K-ésimo camino más corto todos los caminos computados con el mismo costo son retornados.

La visibilidad es inversamente proporcional al costo agregado a la solución por el camino (resta del costo de las aristas *privilegiadas* al costo del camino). La ecuación 3 presenta la función de visibilidad, donde p_i es el camino considerado, *cost* es el costo del camino o la arista y E_p es el conjunto de aristas *privilegiadas*.

$$\eta_{p_i} = \frac{1}{cost(p_i)},\ cost(p_i) = \sum_{c_{i,x_i} \in p_i} cost(c_{i,x_i}) - \sum_{c_{i,x_i} \in E_p} cost(c_{i,x_i}) \qquad (3)$$

El valor asociado a la experiencia adquirida durante la búsqueda se calcula como el promedio de los rastros de feromona en las aristas que no forman parte

de la solución (las que pertenecen al camino y que no pertenecen al conjunto de aristas *privilegiadas*). La ecuación 4 presenta la función para calcular el valor de la experiencia adquirida, donde p_i es el camino considerado y E_p es el conjunto de aristas *privilegiadas*.

$$\tau_{p_i} = \frac{\sum_{c_{i,x_i} \in p_i \text{ y } c_{i,x_i} \notin E_p} \tau_{c_{i,x_i}}}{|c_{i,x_i} \in p_i \text{ y } c_{i,x_i} \notin E_p|} \quad (4)$$

4.3. Búsqueda local

El enfoque seguido para resolver el GSP intenta evitar la incorporación de aristas superfluas que generen más caminos de los necesarios. Sin embargo, el mecanismo de construcción de las soluciones no garantiza esta propiedad y la topología construida puede contener estrictamente un subgrafo que sea solución al problema. En ese caso, es claro que la solución encontrada no será un óptimo, por contener otra de menor costo. Con el objetivo de subsanar este inconveniente, se diseñó un método de búsqueda local para mejorar las soluciones construidas por las hormigas. El método de búsqueda local consiste en aplicar el mecanismo de construcción a la solución construida, en lugar de aplicarlo al grafo original. La vecindad de una solución son todos los subgrafos de la misma que son solución del problema original. El método de búsqueda local es aplicado iterativamente hasta que no se produzcan mejoras. La búsqueda local propuesta restringe la búsqueda y se concentra en un conjunto de caminos potencialmente distinto a los considerados originalmente.

5. Análisis empírico

Esta sección introduce el conjunto de casos de prueba del GSP usados para evaluar los algoritmos implementados. Luego, se comenta la evaluación experimental del algoritmo propuesto, incluyendo el análisis de los resultados obtenidos.

5.1. Conjunto de casos de prueba

El conjunto de casos de prueba usado es el propuesto por Nesmachnow et al. [8]. El conjunto está formado por tres grafos representativos de redes de tamaño mediano con una cantidad variable de requisitos de conectividad y fueron construidos usando un generador de instancias aleatorias. La tabla 1 presenta las instancias indicando el número de nodos, aristas y terminales, la cantidad total de requerimientos de conectividad, el grado de conectividad (radio entre el número de aristas en el grafo y el número de aristas en el grafo completo), costo total y mejor solución conocida. El nombre de la instancia indica el número de nodos y terminales del grafo. Las instancias descritas y el generador aleatorio de grafos están públicamente disponibles [7].

La tabla 2 presenta los resultados reportados por Nesmachnow et al. [8] sobre el conjunto de casos de prueba del GSP. Se indica el costo promedio, la desviación

Tabla 1. Conjunto de casos de prueba

	Grafo 100-10	**Grafo 75-25**	**Grafo 50-15**
Nodos	100	75	50
Terminales	10	25	15
Aristas	500	360	249
Requerimientos	138	362	214
Grado de conectividad	0.1	0.13	0.2
Costo total	4925	6294.93	10949.98
Mejor solución conocida	291	773.09	1353.49

Tabla 2. Resultados del conjunto de casos de prueba

		Grafo 100-10	**Grafo 75-25**	**Grafo 50-15**
GA+SA	*Promedio*	433.5	931.04	1576.66
	Desv. Est.	36.7	39.8	70.7
	Mejor	376	865.59	1475.72
	Tiempo Prom.	520.0	1229.4	462.6
GA	*Promedio*	455.5	986.71	1633.16
	Desv. Est.	28.2	50.2	85.8
	Mejor	394	888.27	1493.95
	Tiempo Prom.	122.8	310.2	125.4
CHC	*Promedio*	323.0	815.98	1435.82
	Desv. Est.	15.1	21.2	43.2
	Mejor	291	773.09	1353.49
	Tiempo Prom.	59.4	275.4	51.0

estándar, la mejor solución y el tiempo promedio de ejecución medido en minutos sobre 30 ejecuciones independientes. La plataforma de ejecución usada fue una computadora Intel Pentium 4 de 2.4 Ghz con 512 Mb RAM. Actualmente, CHC es el algoritmo que obtiene los mejores resultados para el GSP.

5.2. Resultados obtenidos

Para el presente trabajo, los algoritmos fueron implementados en C++. La plataforma de ejecución usada fue una computadora AMD Athlon 64 Processor 3000+ a 2.0 Ghz con 1 GB RAM, usando el sistema operativo SuSE Linux 10.0.

Los valores de los parámetros de ACO usados en este trabajo son $\alpha = 5$, $\beta = 5$, $\rho = 0.1$, $\tau_{init} = 0.5$ y el número de caminos considerados para satisfacer un requerimiento de conexión es 10; un análisis detallado de la calibración de los parámetros puede encontrarse en [10] . Los algoritmos de este estudio usaron un criterio de parada de esfuerzo prefijado de 500 iteraciones. Se realizaron 15 ejecuciones independientes de cada algoritmo sobre el conjunto de instancias de prueba. A pesar de que la plataforma de ejecución de esta prueba y la usada por Nesmachnow et al. [8] no son exactamente las mismas, los procesadores utilizados guardan cierta similitud en su desempeño, por lo que se tomarán los tiempos allí reportados como referencia.

Tabla 3. Resultados con una población de 60 hormigas

		Grafo 100-10	**Grafo 75-25**	**Grafo 50-15**
HCF-$\mathcal{MM}$AS	*Promedio*	300.13	998.94	1671.73
	Desv. Est.	2.29	10.38	19.93
	Mejor	296	981.84	1626.87
	Tiempo Prom.	640.39	518.78	216.02
HCF-$\mathcal{MM}$AS+BL	*Promedio*	281.40	796.36	1389.24
	Desv. Est.	0.63	7.22	5.74
	Mejor	280	770.53	1383.49
	Tiempo Prom.	1216.15	1763.25	647.99

Tabla 4. Resultados con una población de 10 hormigas

		Grafo 100-10	**Grafo 75-25**	**Grafo 50-15**
HCF-$\mathcal{MM}$AS	*Promedio*	307.20	1022.90	1712.04
	Desv. Est.	4.80	15.83	37.66
	Mejor	299	985.67	1643.61
	Tiempo Prom.	97.77	85.91	33.02
HCF-$\mathcal{MM}$AS+BL	*Promedio*	282.47	808.24	1430.16
	Desv. Est.	0.64	6.08	15.72
	Mejor	282	796.51	1394.97
	Tiempo Prom.	204.25	301.30	109.32

En la tabla 3 se presentan los resultados obtenidos al trabajar con una población de 60 hormigas, detallándose el costo promedio, la desviación estándar, la mejor solución obtenida y el tiempo promedio de las ejecuciones en minutos. Se aprecia que los resultados obtenidos por *HCF-$\mathcal{MM}$AS* son buenos, siendo superiores a los obtenidos por CHC para la instancia *Grafo 100-10* y similares a los obtenidos por GA+SA y GA. Sin embargo, el tiempo de ejecución promedio es considerablemente mayor al de CHC. La incorporación del método de búsqueda local provoca una importante mejora en los resultados obtenidos aunque a costa de un incremento sustancial en el tiempo de ejecución promedio. La variante *HCF-$\mathcal{MM}$AS+BL* permitió mejorar las mejores soluciones conocidas para las instancias *Grafo 100-10* y *Grafo 75-25*. Estas soluciones son actualmente las mejores soluciones conocidas para esas instancias del GSP. El costo promedio de las soluciones obtenidas por *HCF-$\mathcal{MM}$AS+BL* es inferior al obtenido por CHC.

En la tabla 4 se presentan los resultados obtenidos con una población de 10 hormigas, detallándose el costo promedio, la desviación estándar, la mejor solución obtenida y el tiempo promedio de las ejecuciones en minutos. Se comprueba que también al trabajar con una población de 10 hormigas, los resultados obtenidos por *HCF-$\mathcal{MM}$AS* son buenos, pero están por debajo del nivel de CHC (con excepción de la instancia *Grafo 100-10*). En cambio, la variante *HCF-$\mathcal{MM}$AS+BL* obtiene resultados de costo promedio mejor a los obtenidos por CHC, e incluso supera a la mejor solución conocida para el *Grafo 100-10*. La reducción en el tamaño de la población muestra que es posible obtener soluciones precisas en un tiempo de ejecución aceptable, si bien es mayor que el de CHC.

6. Conclusiones y trabajo futuro

En este trabajo se estudio la aplicación de un ACO para la resolución del GSP sobre un conjunto de instancias representativas de redes de tamaño mediano.

El ACO propuesto obtuvo buenos resultados pero inferiores a los obtenidos por CHC. La incorporación del método de búsqueda local estudiado provocó importantes mejoras en la calidad de las soluciones obtenidas, a costa de un incremento sustancial en el tiempo promedio de ejecución. Los resultados obtenidos al incorporar la búsqueda local son excelentes, incluso superando las mejores soluciones conocidas para las instancias *Grafo 100-10* y *Grafo 75-25*. La reducción del tamaño de la población permitió obtener resultados competitivos con CHC, utilizando un tiempo de ejecución que si bien es mayor, se considera aceptable.

A partir de los excelentes resultados obtenidos es posible establecer dos líneas de trabajo futuro. El método de búsqueda local puede ser fácilmente incorporado a otras metaheurísticas, por lo que la primera línea de trabajo consiste en estudiar sus propiedades, y adaptarlo para usarlo en el algoritmo CHC. La segunda línea de trabajo consiste en el estudio de alternativas para la reducción del tiempo de ejecución, entre las que se encuentra la paralelización del algoritmo.

Referencias

1. Blum, C. y Dorigo, M. (2004) *The Hyper-Cube Framework for Ant Colony Optimization. IEEE Trans. on Systems, Man, and Cybernetics - Part B*, Vol. 34, No. 2, pp.1161–1172.
2. Crescenzi, P. y Kann, V. (2009) *A Compendium of NP Optimization Problems Website*, `http://www.nada.kth.se/~viggo/problemlist/` (accedida en Mayo 2009).
3. Dorigo, M. and Stützle, T. (2004) *Ant Colony Optimization*, MIT Press.
4. Ford, L. y Fulkerson, D. (1962) *Flows in Networks*, Princeton University Press.
5. Karp, R. (1972) *Complexity of Computer Computations*. Miller, R. y Thatcher, J. (editores) *Reducibility Among Combinatorial Problems*, Plenum Press, pp.85–103.
6. Luyet, L., Varone, S. y Zufferey, N. (2007) *An Ant Algorithm for the Steiner Tree Problem in Graphs*. Proc. of EvoWorkshops 2007, LNCS, Vol. 4448, pp.42–51.
7. Nesmachnow, S. (2009) *Generalized Steiner Problem Website*, Disponible en: `http://www.fing.edu.uy/inco/grupos/cecal/hpc/gsp/` (accedida en Mayo 2009).
8. Nesmachnow, S., Cancela, H. y Alba, E. (2007) *Evolutionary Algorithms Applied to Reliable Communication Network Design. Engineering Optimization*, Vol. 39, No. 7, pp.831–855.
9. Nesmachnow, S. y Pedemonte, M. (2009) *Metaheurísticas basadas en adaptación social para el Problema de Steiner Generalizado*. Actas MAEB 2009, pp.107–114.
10. Pedemonte, M. (2009) *Ant Colony Optimization para la resolución del Problema de Steiner Generalizado*. M.Sc. Tesis, PEDECIBA Informática, UDELAR, Uruguay.
11. Robledo, F. (2001) *Diseño Topológico de Redes, Casos de Estudio: The Generalized Steiner Problem y The Steiner 2-Edge-Connected Subgraph Problem*. M.Sc. Tesis, PEDECIBA Informática, UDELAR, Uruguay.
12. Stützle, T. y Hoos, H. (2000) $\mathcal{MAX} - \mathcal{MIN}$ *Ant System*, *Future Generation Computer Systems*, Vol. 16, No. 8, pp.889–914.
13. Yen, J. (1971) *Finding the K Shortest Loopless Paths in a Network. Journal of Computer Science and Technology*, Vol. 17, pp.712–716.