Un algoritmo multithreading para el problema del árbol de Steiner.

Gerardo Ares Meneces
CeCal, Facultad de Ingeniería.
Universidad de la República, Uruguay
gares@fing.edu.uy

Sergio Nesmachnow
CeCal, Facultad de Ingeniería.
Universidad de la República, Uruguay
sergion@fing.edu.uy

Resumen

Este artículo presenta una implementación paralela de la metaheurística SN [13] utilizando una técnica de programación multithreading y su aplicación a la resolución del problema del árbol de Steiner. Se describen las decisiones de diseño del algoritmo y se presentan experimentos realizados sobre un conjunto de problemas de prueba estándar, analizando la calidad de resultados obtenidos y la eficiencia computacional de la versión paralela del algoritmo.

Palabras clave: Problema del árbol de Steiner, STP, multithreading, metaheurística SN.

Destinado al Workshop de Agentes y Sistemas Inteligentes.

1. Introducción

El continuo crecimiento en el tamaño de los problemas de telecomunicaciones ha conducido a los investigadores a proponer alternativas a los enfoques exactos tradicionales para resolver problemas complejos. En este contexto, varias heurísticas se han aplicado para la resolución de los complejos problemas de optimización subyacentes a los problemas de diseño de redes de comunicaciones.

Considerando una red de comunicaciones con nodos distinguidos denominados terminales, el problema del árbol de Steiner (Steiner Tree Problem – STP) refiere al diseño de una subred de mínimo costo que garantice la conectividad entre pares de nodos terminales. Este problema modela el diseño de redes de comunicaciones que requieren una configuración que asegure la existencia de un camino entre pares de nodos terminales, garantizando un valor mínimo de confiabilidad en la comunicación de datos entre nodos emisores/receptores de información. El STP forma parte de los problemas clásicos de optimización combinatoria y diversos algoritmos han sido propuestos para su resolución. Cuando las instancias del problema crecen en tamaño y complejidad, los algoritmos exactos tradicionales dejan de ser aplicables en la práctica, y las técnicas heurísticas de resolución emergen como alternativas viables para obtener resultados aproximados de calidad aceptable en tiempos de ejecución moderados.

La metaheurística SN [13] propone utilizar una técnica de subdivisión de un problema de optimización en varios problemas de decisión que se resuelven mediante técnicas heurísticas para las dos alternativas posibles, correspondientes a los resultados "SI" y "NO" del problema de decisión. De acuerdo a su formulación, la metaheurística SN es altamente paralelizable, siendo posible atacar concurrentemente los diversos subproblemas generados utilizando varios elementos de procesamiento para resolver cooperativamente el problema de optimización original.

Este trabajo describe una implementación paralela genérica de la metaheurística SN desarrollada sobre una arquitectura de memoria compartida utilizando técnicas de programación multithreading. Complementariamente, se presentan los resultados preliminares obtenidos en la resolución del problema del árbol de Steiner sobre un conjunto estándar de problemas de prueba, analizando la calidad de resultados obtenidos y la eficiencia computacional del algoritmo paralelo.

El resto del artículo se organiza del modo que se describe a continuación. La sección 2 brinda una presentación del problema del árbol de Steiner. La sección 3 presenta los conceptos relacionados con la metaheurística SN. La aplicación de la metaheurística SN para la resolución del problema del árbol de Steiner se presenta en la sección 4. Los detalles de diseño e implementación del algoritmo paralelo multithreading se ofrecen en la sección 5. La sección 6 presenta los experimentos realizados y el análisis de calidad de los resultados obtenidos en la resolución del STP y el estudio de la eficiencia computacional del algoritmo paralelo. Por último, la sección 7 ofrece las conclusiones del trabajo y líneas de investigación actual y futura.

2. El problema del árbol de Steiner

Dado un grafo conexo no dirigido, con aristas de costos no negativos y un subconjunto distinguido de nodos, denominados *nodos terminales*, el problema del árbol de Steiner plantea hallar un árbol de costo mínimo que incluya a todos los nodos terminales. La Figura 1 presenta una formulación del problema, basada en la presentada en el compendio de Kahn y Crescenzi [2].

Instancia del problema:

- Un grafo no dirigido $G = (V,E)$, siendo V el conjunto de nodos y E el conjunto de aristas.
- Una función de costos C asociados las aristas de G.
- Un subconjunto fijo del conjunto de nodos $T \subseteq V$, llamados nodos *terminales,* de cardinalidad $n_T = |T|$, tal que $2 \leq n_T \leq n$, siendo $n = |V|$ la cardinalidad del conjunto de nodos V.

Solución:

- Un subgrafo de cubrimiento del conjunto de nodos terminales, que sea de costo mínimo.

Figura 1. Formulación del STP.

Sobre los nodos no pertenecientes al conjunto de nodos terminales no se plantean requisitos de conectividad. Estos nodos, llamados *nodos de Steiner*, pueden formar parte de la solución o no, dependiendo de la conveniencia de utilizarlos. En el contexto de problemas de redes de comunicaciones, los nodos terminales representan a los agentes emisores/receptores de información, los nodos de Steiner a posibles entidades ruteadoras, mientras que las aristas representan a los enlaces bidireccionales de comunicación entre nodos.

En 1973 Karp [5] demostró que el problema del árbol de Steiner es NP-Completo. Desde entonces, diferentes metaheurísticas han sido aplicadas para obtener soluciones aproximadas de calidad aceptable para el problema, en tiempos de ejecución razonables. Entre los trabajos pioneros se pueden mencionar los de Kapsalis et al. [4] y Esbensen [3] quienes propusieron resolver el problema utilizando algoritmos genéticos. Otros enfoques fueron propuestos por Martins et al. [7] y Ribeiro et al. [11], abordando el problema utilizando GRASP, y Ribeiro y De Souza [10] aplicando Tabu Search. En uno de los trabajos más recientes, Lo Presti et al. [6] resolvieron el problema mediante un algoritmo genético paralelo ejecutado en un entorno de área global utilizando tecnología GRID.

3. Metaheurística SN

La metaheurística SN, propuesta por Urrutia y Loiseau [13], plantea dividir un problema de optimización en varios subproblemas de decisión de resolución más simple que el problema original. Utilizando una heurística específica, se obtiene una solución (s_i) y un grado de confianza (gc_i) para cada uno de los subproblemas. De este modo, el problema original se transforma en un problema equivalente tomando en cuenta la decisión del subproblema cuya solución tuvo el mayor grado de confianza. El procedimiento que define la metaheurística se aplica recursivamente a este nuevo problema, mientras sea posible realizar la división en subproblemas.

El esquema general de la metaheurística SN se presenta en la Figura 2.

```
Mientras pueda dividirse el problema en n subproblemas equivalentes hacer
     Para cada i desde 1 hasta n hacer
           Resolver heurísticamente el problema i obteniendo s_i y gc_i
     Fin Para
     Modificar el problema utilizando la información del subproblema
    que obtuvo mayor grado de confianza
Fin Mientras
```

Figura 2. Esquema genérico de la metaheurística SN.

Dado un elemento del problema, existirán dos subproblemas de decisión complementarios que lo involucren en cada paso. Estos corresponderán a casos "SI" y "NO" para los cuales se propone un esquema constructivo para la metaheurística resumido en dos casos disjuntos:

- Obtener una solución constructiva al subproblema asumiendo que la decisión es SI.
- Obtener una solución constructiva al subproblema asumiendo que la decisión es NO.

Evaluando la magnitud a optimizar, se toma como solución para el subproblema *i* la mejor de las dos soluciones constructivas de acuerdo al criterio de optimización utilizado. El grado de confianza se computa como la diferencia entre los valores de las soluciones obtenidas para cada caso.

Sobre las transformaciones del problema que se realizan al final de cada iteración, se exigen tres condiciones:

- La transformación tiene que generar un problema que pueda ser dividido en menos subproblemas que en la instancia anterior, de modo de reducir la complejidad de los problemas generados a medida que se aplica la etapa de división de la metaheurística.
- La solución óptima del problema transformado sea igual o peor que la del problema original.
- Si la solución para el subproblema *i* (s_i) es correcta, entonces la solución óptima para el problema transformado debe ser igual a la del problema original.

4. Aplicación de SN al problema STP

Los propios creadores de la metaheurística SN han propuesto aplicarla a la resolución del problema del árbol de Steiner [13], tomando en cuenta la posibilidad de dividir este clásico problema de optimización combinatoria en subproblemas independientes.

Para el problema del árbol de Steiner, la decisión a tomar en cada iteración de la metaheurística corresponde a determinar si un nodo no terminal pertenecerá o no al árbol solución del problema. De este modo, se divide el problema original en subproblemas, cada uno de ellos considerará un nodo no terminal y buscará dos soluciones complementarias: una que contenga el nodo terminal ("caso SI") y otra en que no lo contenga ("caso NO"). El grado de confianza para cada subproblema estará dado por la diferencia en valor absoluto de los costos de los árboles solución para cada caso.

Una vez evaluados todos los subproblemas se escoge el nodo que generó el mayor grado de confianza y se transforma el problema según la decisión que produjo la mejor solución. Si la decisión corresponde a un "caso SI", se agrega el nodo no terminal considerado al conjunto de terminales del grafo. Si por el contrario la mejor solución fue para el "caso NO", entonces se elimina del grafo el nodo en cuestión.

La aplicación iterativa del procedimiento llegará a su fin cuando se haya procesado la totalidad de los nodos no terminales del problema.

El algoritmo utiliza la heurística MCP propuesta por Takahashi y Matsuyama [12] para la generación de las soluciones en los casos "SI" y "NO", basándose en la construcción de un árbol de cubrimiento para los terminales partiendo de un nodo terminal elegido al azar. En cada paso se procesa el nodo más cercano al árbol de cubrimiento, si este nodo resulta ser terminal o resulta ser el nodo no terminal de un subproblema "caso SI", se agregan al árbol todos los nodos (terminales y no terminales) pertenecientes al camino que une el nodo encontrado con el árbol.

En el "caso SI", el nodo del subproblema es considerado como si fuera un nodo terminal, incluyéndolo al árbol que genera la búsqueda. En el caso NO, se utiliza la misma heurística constructiva pero sobre el grafo resultado de eliminar el nodo del subproblema, generando una solución en donde no participará el nodo en cuestión.

La fase iterativa del algoritmo llegará a su fin cuando haya procesado la totalidad de los nodos no terminales. Se obtiene entonces un subgrafo compuesto por los nodos terminales y aquellos no terminales que se decidió incluir en las etapas iterativas del algoritmo. Sobre este subgrafo se halla el árbol de cubrimiento de costo mínimo utilizando el algoritmo de Prim [8] de acuerdo a la mejora propuesta por Rayward-Smith y Clare [9].

La instancia de la metaheurística SN utilizada para resolver el problema STP se presenta en la Figura 4.

```
Mientras existan nodos terminales a procesar hacer
    Para cada nodo no terminal hacer
          Resolver heurísticamente el problema para el "caso SI" y
          para el "caso NO" obteniendo s_i y gc_i
    Fin Para
    Obtener el mayor gc_i y transformar el problema según la mejor
    solución entre el "caso SI" y "caso NO".
Fin Mientras
```

Figura 4. Metaheurística SN aplicada al STP.

Ciertos casos particulares han sido contemplados en la etapa de diseño para mejorar la eficiencia del algoritmo:

- El caso en que no se encuentre solución al evaluar algún subproblema. Como ejemplo, en un subproblema "caso NO" el grafo podría quedar desconexo si el nodo en estudio no se incluye en la solución. Ante tal situación se considera al nodo como parte de los nodos terminales, ya que necesariamente deberá formar parte de la solución.
- El caso en que el problema de decisión resultante sea tal que divida al grafo en dos componentes conexas, una compuesta por terminales y otra por no terminales. El algoritmo toma ventaja de esta situación descartando la componente que no contiene nodos terminales.

5. Algoritmo multithreading para el problema STP

El esquema de la metaheurística SN es paralelizable en un alto grado, tomando en cuenta que los subproblemas a ser resueltos en cada iteración deben ser independientes entre sí. Una implementación paralela reducirá los tiempos de ejecución total del algoritmo utilizando los recursos computacionales disponibles y permitirá atacar instancias de mayor dimensión y complejidad del problema del árbol de Steiner.

La propuesta consiste en aplicar el criterio de descomposición de dominio, asignando diferentes secciones del espacio de búsqueda a diferentes procesadores, tal como se describe en el diagrama de la Figura 3. El diseño del algoritmo corresponde al modelo de programación maestro-esclavo, en donde un proceso maestro se encarga de la división de tareas entre procesos esclavos, quienes llevan a cabo la exploración del espacio de soluciones de los diferentes subproblemas. Una vez finalizada la resolución de los subproblemas, el proceso maestro centraliza la búsqueda del mejor grado de confianza entre los valores reportados por los procesos esclavos.

Se propuso trabajar sobre una arquitectura de memoria compartida, implementando los procesos esclavos mediante threads que trabajan sincrónicamente. Los detalles de la implementación paralela se ofrecen a continuación.

Se diseñó una implementación genérica de la metaheurística que permitiera su aplicación a diferentes problemas de optimización combinatoria. Se utilizó el lenguaje de programación C++ y el paralelismo se implementó utilizando la biblioteca *pthread* para programación multithreading. La implementación genérica luego fue instanciada para la resolución específica del STP.

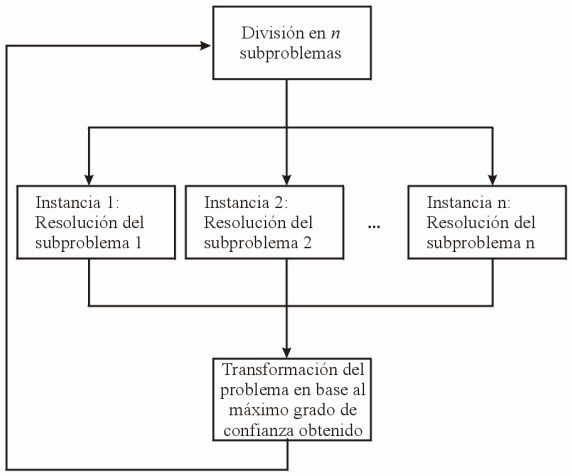

Figura 3. Esquema de paralelismo aplicado a la metaheurística SN.

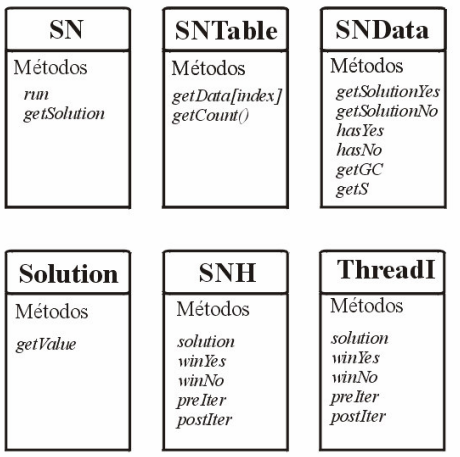

Figura 4. Clases de la implementación multithreading para la metaheurística SN.

Las clases que implementan el esqueleto genérico de la metaheurística se presentan en la Figura 4 y se describen a continuación:

- **SN:** Contiene la iteración general de la metaheurística.
- **SNTable:** Tabla que almacena la información generada en la resolución de los subproblemas. Cada elemento de esta tabla es un objeto que implementa la clase abstracta SNData.
- **SNData:** Clase abstracta en donde se encuentra la información básica generada en la resolución de los subproblemas (gc_i y s_i).
- **ThreadI:** Pool de threads especializados en resolver la heurística constructiva.
- **SNH:** Clase abstracta. Su utilidad es definir una interfaz que debe cumplir una instancia de la heurística constructiva.
- **Solution:** Clase abstracta. Define una interfaz que debe cumplir la solución al problema SN.

La clase *SN* provee un método de nombre *run,* el cual tiene la estructura de iteración general presentada en la Figura 2. Para llevar a cabo la resolución de la metaheurística la clase *SN* define un pool de threads (clase *ThreadI*), una tabla en donde se almacena la información que genera la resolución de los subproblemas en cada iteración (clase *SNTable*), un objeto que contiene la implementación de la heurística constructiva a aplicar en cada subproblema "caso SI" y "caso NO" (clase *SNHInstance*), y la mejor solución encontrada hasta el momento (clase *Solution*).

La clase *ThreadI* representa al pool de threads, que se especializan en resolver únicamente los subproblemas. Son creados al comienzo de algoritmo y quedan a la espera de una señal que activará al máximo número de threads que deben ejecutar en paralelo. Este valor es determinado como el menor entre un número configurable (como ejemplo, la cantidad de procesadores disponibles) y la cantidad de nodos no terminales que tenga el problema.

Cada thread invoca a los métodos *solutionYes* y *solutionNo* de la instancia *SNHInstance* pasando como parámetro la entrada en la tabla *SNTable* correspondiente al subproblema a resolver. Los métodos *solutionYes* y *solutionNo* cargan la entrada de la tabla con la información obtenida por la heurística constructiva para el "caso SI" y "caso NO" respectivamente. Posteriormente se determina la mejor resolución y se definen los valores gc_i y s_i para la entrada. Si existen más subproblemas a resolver, el thread selecciona uno y lo resuelve. Si no, queda a la espera para ser invocado en una próxima iteración de la metaheurística.

Finalizada la ejecución de todos los threads activados, el control pasa nuevamente al método *run,* quien busca el mejor grado de confianza obtenido en la iteración. Una vez hallado, se transforma el problema en un subproblema equivalente, aplicando los métodos *winYes* o *winNo* de la clase *SNHInstance* según corresponda al caso "SI" o "NO".

Las clases que instancian la metaheurística para resolver el problema del árbol de Steiner se incluyen en el diagrama de la Figura 5, y se describen a continuación:

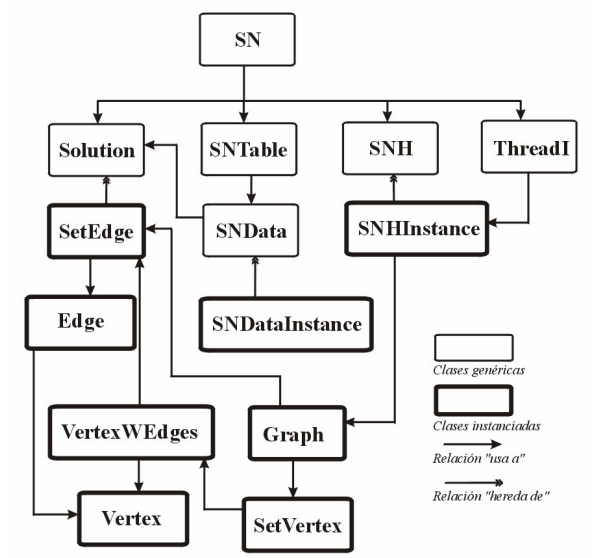

Figura 5. Diagrama de clases de la implementación multithreading para la metaheurística SN

- **SNDataInstance:** Extiende la clase SNData, incorporando las soluciones para el "caso SI" y el "caso NO" del nodo correspondiente a cada subproblema.
- **SNHInstance:** Extiende la clase SNH, implementando la heurística constructiva (MCP). Contiene el grafo e implementa las transformaciones del problema.
- **Edge:** Clase que representa una arista del grafo.
- **Vertex:** Clase que representa un nodo del grafo.
- **SetEdges:** Extiende la clase Solution, representando a un conjunto de aristas. La solución del STP constituye un ejemplo de un objeto de esta clase.
- **SetVertex:** Representa a un conjunto de nodos.
- **Graph:** Conjunto de nodos, aristas y nodos terminales que definen el problema.

Para reducir los tiempos de ejecución de la metaheurística aplicada a la resolución del STP Urrutia y Loiseau [13] propusieron una serie de mejoras, que se describen a continuación:

- Utilizar una base de datos que almacene soluciones del STP halladas mediante la heurística MCP. De esta forma, se trata de evitar la ejecución redundante la heurística para subproblemas que ya fueron resueltos en alguna iteración anterior.
- Evaluar un número reducido de subproblemas en cada iteración. Se evaluaran solamente los subproblemas que han logrado mejor grado de confianza hasta el momento.
- Debido a la evaluación selectiva se pierde calidad en las soluciones, por lo que se introducen búsquedas aleatorias de soluciones utilizando la heurística MCP, al final de cada iteración.

De los puntos anteriores surgen los siguientes parámetros de ejecución de la metaheurística SN:

- Cantidad de soluciones que se buscan para inicializar la base de datos.
- Cantidad de evaluaciones calificadas por iteración.
- Probabilidad de búsqueda de soluciones al final de cada iteración.

Estas mejoras fueron incorporadas al algoritmo paralelo diseñado.

6. Resultados obtenidos

La correctitud de la implementación multithreading de la metaheurística SN ha sido verificada sobre un conjunto de problemas de prueba extraídos del repositorio OR-Library [1], sobre los cuales fueron aplicadas las reducciones presentadas por Esbensen [3] que simplifican los problemas. Las características de las instancias de prueba se presentan en las Tablas 1 y 2, indicando número de nodos, aristas y terminales para las instancias originales y reducidas de cada grafo.

Grafo	*Originales*			*Reducidos*		
	|V|	*|E|*	*|T|*	*|V|*	*|E|*	*|T|*
c01	500	625	5	145	263	5
c02	500	625	10	130	239	8
c03	500	625	83	120	232	35
c04	500	625	125	106	217	36
c05	500	625	250	37	92	17
c06	500	1000	5	369	847	5
c07	500	1000	10	382	869	9
c08	500	1000	83	334	815	52
c09	500	1000	125	346	829	75
c10	500	1000	250	202	591	65
c11	500	2500	5	499	2184	5
c12	500	2500	10	498	2236	9
c13	500	2500	83	455	2061	57
c14	500	2500	125	413	1874	67
c15	500	2500	250	282	1303	77
c16	500	12500	5	500	4740	5
c17	500	12500	10	499	4696	9
c18	500	12500	83	486	4666	70
c19	500	12500	125	457	4320	82
c20	500	12500	250	343	3235	100

Tabla 1: Características de grafos clase c.

Grafo	*Originales*			*Reducidos*		
	|V|	*|E|*	*|T|*	*|V|*	*|E|*	*|T|*
d01	1000	1250	5	274	510	5
d02	1000	1250	10	285	523	10
d03	1000	1250	167	224	441	67
d04	1000	1250	250	153	332	59
d05	1000	1250	500	95	229	47
d06	1000	2000	5	761	1741	5
d07	1000	2000	10	754	1735	10
d08	1000	2000	167	730	1707	123
d09	1000	2000	250	650	1610	151
d10	1000	2000	500	399	1263	128
d11	1000	5000	5	993	4674	5
d12	1000	5000	10	999	4669	9
d13	1000	5000	167	915	4387	115
d14	1000	5000	250	838	4100	145
d15	1000	5000	500	515	2613	128
d16	1000	25000	5	1000	10595	5
d17	1000	25000	10	1000	10542	10
d18	1000	25000	167	961	9956	128
d19	1000	25000	250	924	9565	179
d20	1000	25000	500	708	7541	218

Tabla 2: Características de grafos clase d.

En la etapa de evaluación de resultados se realizaron treinta ejecuciones independientes de la metaheurística para cada una de las instancias estudiadas sobre un equipo multiprocesador Sun SPARCcenter 2000E con 16 procesadores SuperSPARC II de 85 MHz. La configuración de parámetros empleada se presenta en la Tabla 3, diferenciándose en la utilizada en el modelo serial de Urrutia y Loiseau [13] en el número de soluciones iniciales en la base de datos (un valor constante en [13] y dependiente del número de terminales en nuestra propuesta) y la cantidad de soluciones buscadas al final de cada iteración (solamente una en el modelo serial, y hasta 16, una por cada proceso utilizado, en la implementación multithreading).

Parámetro	*Valor*
Cantidad de evaluaciones por iteración	50
Cantidad de soluciones iniciales en la base de datos	|T|
Soluciones aleatorias buscadas al final de cada iteración.	1 solución por procesador disponible
Probabilidad de búsqueda de soluciones al final de cada iteración	0, 0.10 y 0.25

Tabla 3: Configuración de parámetros en el algoritmo multithreading.

Las Tablas 4 y 5 presentan los resultados obtenidos para los problemas estudiados, ofreciendo valores de costo promedio y su desviación estándar, así como el mejor valor obtenido y el promedio de tiempos de ejecución del algoritmo utilizando 16 procesadores (en segundos). Para la comparación de resultados se toma como referencia el valor óptimo de cada problema, calculado por Beasley utilizando un algoritmo exacto de *branch and cut* [1].

Grafo	*Óptimo*	*Prom. SN*	*σ*	*Mejor SN*	*T (s)*
c01	85	85,0	0,0	85	3,4
c02	144	144,0	0,0	144	3,1
c03	754	755,1	1,2	754	2,8
c04	1079	1080,1	0,6	1079	2,7
c05	1579	1579,0	0,0	1579	0,8
c06	55	55,3	0,5	55	15,4
c07	102	102,0	0,0	102	17,1
c08	509	509,4	1,1	509	27,1
c09	707	710,4	2,2	707	39,9
c10	1093	1093,7	1,0	1093	13,0
c11	32	32,3	0,5	32	41,7
c12	46	46,1	0,3	46	33,1
c13	258	261,5	1,9	259	66,6
c14	323	324,9	1,2	323	52,0
c15	556	556,6	0,6	556	29,4
c16	11	11,3	0,5	11	40,1
c17	18	18,2	0,4	18	37,0
c18	113	118,9	1,8	116	83,4
c19	146	150,1	1,3	148	74,1
c20	267	267,3	0,5	267	22,3

Tabla 4: Resultados obtenidos en grafos clase c.

Grafo	*Óptimo*	*Prom. SN*	*σ*	*Mejor SN*	*T (s)*
d01	106	106,0	0,0	106	9,7
d02	220	220,5	1,0	220	11,0
d03	1565	1571,8	2,7	1567	13,6
d04	1935	1936,3	1,1	1935	6,8
d05	3250	3252,9	1,3	3251	1,8
d06	67	67,3	0,5	67	98,6
d07	103	103,2	0,4	103	89,0
d08	1072	1084,5	3,1	1078	513,5
d09	1448	1456,8	2,5	1452	404,0
d10	2110	2113,3	1,6	2110	72,0
d11	29	29,4	0,5	29	210,7
d12	42	42,0	0,0	42	177,2
d13	500	506,9	2,1	503	783,8
d14	667	671,9	3,0	669	798,2
d15	1116	1118,8	2,0	1116	161,4
d16	13	13,3	0,5	13	406,4
d17	23	23,0	0,0	23	272,9
d18	223	234,4	2,8	229	1049,8
d19	310	323,7	5,4	316	1093,2
d20	537	538,7	1,2	537	259,1

Tabla 5: Resultados obtenidos en grafos clase d.

Las Figuras 1 y 2 presentan los resultados comparativos para las clases *c* y *d* respectivamente, graficando los valores de GAP respecto a los óptimos entre promedios y mejores valores obtenidos para cada instancia de prueba estudiada.

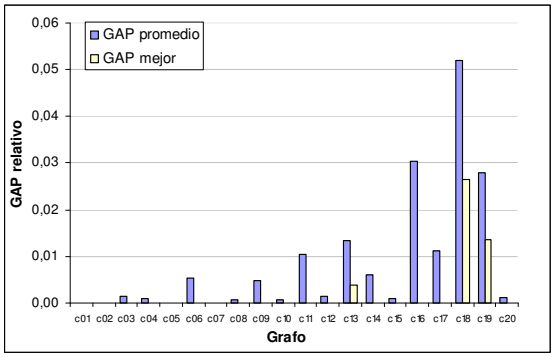

Figura 1 : GAPs obtenidos en grafos clase c.

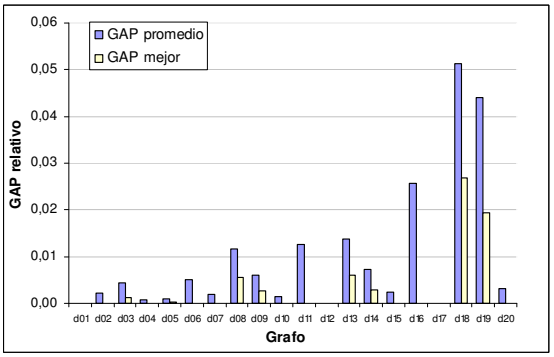

Figura 2: GAPs obtenidos en grafos clase d.

La Tabla 6 compara los resultados obtenidos por la versión paralela de la metaheurística SN con los resultados alcanzados por otras técnicas heurísticas aplicadas a la resolución del problema STP.

Autor	***Heurística***	***Porcentaje de alcance del óptimo***	***Resultados al 99% (GAP<1%)***
Kapsalis et al.	Alg. Genéticos	70.0 %	No disponibles.
Esbensen	Alg. Genéticos	90.0 %	92.5 %
Martins et al.	GRASP Paralelo	78.3 %	91.7 %
Lo Presti et al.	Alg. Genéticos Paralelos	80.0 %	90.0 %
Urrutia y Loiseau	SN serial	57.5 %	90.0 %
Este trabajo	SN paralelo	72.5 %	90.0 %

Tabla 6: Resultados comparativos en grafos clase c y d.

Los resultados obtenidos por la implementación multithreading de la metaheurística SN han mostrado estar a la par de los resultados obtenidos en los trabajos citados, alcanzando valores con GAP menor al 1% con respecto al óptimo en 90% de los casos. Respecto al número de veces en que se alcanza el óptimo, los resultados de la versión paralela de la metaheurística SN son significativamente inferiores a los obtenidos por Esbensen, pero son competitivos respecto a las propuestas de Kapsalis et al., Martins et al. y Lo Presti et al. Asimismo, se mejoran los resultados de la versión serial de Urrutia y Loiseau, como consecuencia de haber utilizado una configuración diferente en los parámetros de ejecución de la metaheurística.

La Tabla 7 presenta el análisis de la eficiencia computacional del algoritmo multithreading al trabajar con diferentes números de procesadores, para resolver casos representativos de la clase D, que requieren alta demanda de requerimientos computacionales. Se ofrecen valores de speedup algorítmico y eficiencia definidos por las Ecuaciones 1 y 2, donde T_M corresponde al tiempo de ejecución del algoritmo utilizando M procesadores.

$$S_M = \frac{T_1}{T_M}$$

Ecuación 1: Definición de speedup algorítmico.

$$E_M = \frac{S_M}{M}$$

Ecuación 2: Definición de eficiencia.

Proc.	*d3*		*d4*		*d10*		*d12*		*d17*		*d20*	
	S_i	E_i	S_i	E_i	S_i	E_i	S_i	E_i	S_i	E_i	S_i	E_i
1	1.0	1.00	1.0	1.00	1.0	1.00	1.0	1.00	1.0	1.00	1.0	1.00
2	1.8	0.90	1.8	0.89	1.9	0.96	1.6	0.79	1.5	0.75	1.9	0.95
3	2.4	0.80	2.4	0.79	2.7	0.91	1.9	0.63	1.6	0.54	2.9	0.95
4	3.0	0.75	2.8	0.70	3.4	0.86	2.1	0.52	2.0	0.49	3.7	0.92
5	3.3	0.66	3.1	0.61	4.1	0.83	2.1	0.42	2.0	0.41	4.6	0.93
6	3.6	0.60	3.2	0.54	4.7	0.78	2.3	0.38	2.2	0.36	5.4	0.90
7	3.9	0.56	3.3	0.47	5.3	0.76	2.4	0.35	2.1	0.30	6.1	0.88
8	4.1	0.51	3.4	0.43	5.6	0.71	2.5	0.31	2.3	0.28	6.8	0.86
9	4.3	0.48	3.4	0.38	5.8	0.65	2.5	0.27	2.1	0.23	7.6	0.85
10	4.4	0.44	3.4	0.34	6.5	0.65	2.5	0.25	2.4	0.24	8.5	0.85
11	4.3	0.39	3.3	0.30	6.7	0.61	2.6	0.24	2.4	0.22	9.0	0.82
12	4.3	0.36	3.3	0.27	7.1	0.59	2.6	0.22	2.4	0.20	9.4	0.78
13	4.4	0.34	3.1	0.24	7.4	0.57	2.6	0.20	2.0	0.15	10.4	0.80
14	4.4	0.32	3.0	0.21	7.5	0.53	2.7	0.19	2.4	0.17	10.7	0.77
15	4.4	0.29	2.6	0.18	7.6	0.50	2.7	0.18	2.4	0.16	10.6	0.71
16	4.1	0.26	2.7	0.17	7.8	0.49	2.7	0.17	2.3	0.14	10.8	0.68

Tabla 7: Speedup y eficiencia en grafos clase d.

El algoritmo presenta un comportamiento de speedup sub-lineal para todos los casos. Los valores de eficiencia se apartan significativamente del valor ideal (correspondiente a speedup lineal) en los grafos con pocos nodos terminales, pero presentan valores aceptables para los grafos con mayor cantidad de nodos terminales (d10 y d20). Para el caso del grafo d20, el análisis de los valores de eficiencia permite comprobar que se obtienen valores aceptables, superiores a 0.85, hasta con 10 procesadores, degradándose paulatinamente al aumentar el número de recursos computacionales.

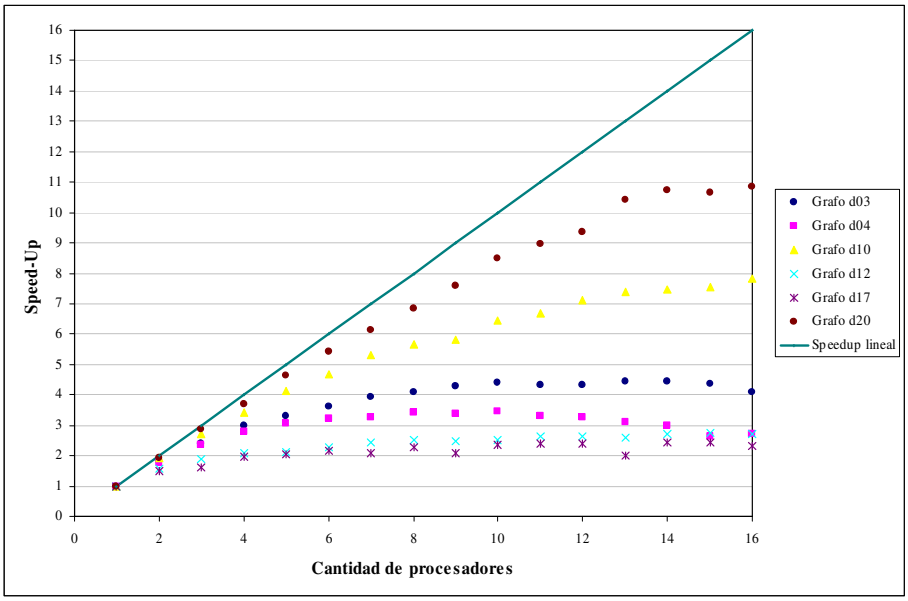

Figura 3: Análisis de la eficiencia computacional.

7. Conclusiones y trabajo futuro

El trabajo presenta el diseño e implementación de un algoritmo multithreading para la resolución del problema del árbol de Steiner aplicando la metaheurística SN. Las pruebas de verificación del algoritmo han mostrado su correctitud y su capacidad de alcanzar resultados de buena calidad para el conjunto de problemas de prueba presentado. La versión paralela alcanza los óptimos en el 72.5% de los casos estudiados, superando los valores obtenidos por la versión serial de la metaheurística.

La implementación paralela presentó un comportamiento de speedup sub-lineal al trabajar con 16 procesadores. El algoritmo mostró valores bajos de eficiencia computacional en problemas donde el conjunto de nodos terminales es reducido, pero mejoró significativamente al resolver problemas de mayor porte, alcanzando valores aceptables en el entorno de 0,85 al trabajar con 10 procesadores.

A lo largo del trabajo, numerosas líneas de investigación han reportado interesantes resultados o han dejado aspectos abiertos que plantean líneas de trabajo futuro. En la actualidad se trabaja en la determinación de la influencia de los múltiples parámetros de configuración de la implementación paralela de la metaheurística en la calidad de resultados obtenidos en la resolución del STP.

Complementariamente, se plantea extender el conjunto de problemas de prueba incluyendo grafos de mayor dimensión (clase *e* de OR-Library), de modo de determinar la escalabilidad del algoritmo propuesto al aumentar el tamaño y la complejidad de los problemas.

Asimismo, se plantea la posibilidad de abordar otros problemas de optimización combinatoria NP-difíciles, tomando en cuenta la generalidad de la implementación paralela diseñada para la metaheurística SN.

8. Referencias bibliográficas

[1] Beasley, J. *OR-Library: Distributing test problems by electronic mail*. Journal of the Operational Research Society **41**, pp. 1069–1072, 1990.

[2] Crescenzi P, Kahn V. *A compendium of NP optimization problems*. Disponible online: http://www.nada.kth.se/~viggo/problemlist/. Consultada junio 2004.

[3] Esbensen H. *Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm*. Ph. D. Tesis, Department of Computer Science, University of Aarhus, Dinamarca. Resumen publicado en Networks **26**, pp. 173–185, 1995.

[4] Kapsalis, A., Rayward-Smith, V., Smith, G. *Solving the graphical Steiner tree problem using genetic algorithms*. Journal of the Operational Research Society **44**, pp. 397–406, 1993.

[5] Karp, R. *Reducibility among combinatorial problems*, Complexity of Computer Computations, R.E. Miller and J.W.Tatcher, eds., Plenum Press, New York, 1972.

[6] Lo Presti, G., Lo Re, G., Storniolo, P., Urso, A. *A Grid Enabled Parallel Hybrid Genetic Algorithm for SPN*. Proceedings of the International Conference on Computational Science, ICCS'04, pp. 156–163, 2004.

[7] Martins, S., Resende, M., Ribeiro, C., Pardalos, P. *A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy*. Journal of Global Optimization **17**, pp. 267–283, 2000.

[8] Prim, R. *Shortest Connection Networks and Some Generaliazations*, Bell System. Technical Journal **36**, pp. 1389–1401, 1957.

[9] Rayward-Smith, V., Clare, A., *On finding Steiner vertices*, Networks **16**, pp.283– 294, 1986.

[10] Ribeiro, C., De Souza, M. *Tabu Search For The Steiner Problem In Graphs*. Networks **36**, pp. 138–146, 2000.

[11] Ribeiro, C., Uchoa, E., Wernek R. *A hybrid GRASP with perturbations for the Steiner problem in graphs*. INFORMS Journal on Computing, **14** (3), pp. 228–246, 2002.

[12] Takahashi, H., Matsuyama, A. *An approximate solution for the Steiner problem in graphs*. Math. Japonica **24**, pp. 573–577, 1980.

[13] Urrutia, S., Loiseau I. *A new metaheuristic and its application to the Steiner Problems in graph*. Proceedings of the XXI International Conference of the Chilean Computer Science Society (SCCC'01), 2001.