Técnicas evolutivas aplicadas al diseño de redes de comunicaciones confiables

Sergio Nesmachnow * Héctor Cancela Enrique Alba

Resumen - Este trabajo presenta la aplicación de diversos algoritmos evolutivos a un problema propuesto para modelar el diseño de redes de comunicaciones de alta confiabilidad topológica. El Problema de Steiner Generalizado (GSP) exige la existencia de un número variable de caminos disjuntos entre cada par de nodos terminales de comunicación. La solución del GSP se construye utilizando nodos intermedios para asegurar la redundancia de caminos, y tratando de minimizar el costo total. Se trata de un problema NP-difícil, para el que existen pocos algoritmos propuestos. Este trabajo presenta la resolución del GSP mediante algoritmos evolutivos, codificados sobre MALLBA una biblioteca de propósito general para optimización combinatoria. Se compara el comportamiento de los algoritmos sobre un conjunto de instancias de prueba, reportándose resultados eficientes para los modelos seriales y distribuidos implementados.

Palabras clave - Redes de Comunicaciones Confiables, Problema de Steiner Generalizado, Algoritmos Evolutivos.

I. INTRODUCCIÓN

Al diseñar redes de comunicaciones, un problema importante consiste en hallar una topología de conexión de sus nodos cuyas propiedades aseguren la comunicación confiable de datos. En los últimos años, el rápido desarrollo de la infraestructura de redes, del software y de los servicios de Internet han renovado el interés por los problemas de diseño de redes de comunicaciones. Como consecuencia del continuo crecimiento en el tamaño de las redes, los problemas de optimización subyacentes sobrepasan con frecuencia la capacidad de los algoritmos exactos tradicionales. En este contexto, varias heurísticas se han aplicado al diseño de redes de comunicaciones confiables cuando se desea resolver problemas de dimensiones reales en tiempos razonables. Entre ellas, las técnicas de programación evolutiva se han manifestado como métodos flexibles y robustos para la solución de los complejos problemas de optimización relacionados con el diseño de redes confiables.

Este trabajo presenta la aplicación de diversos Algoritmos Evolutivos (EAs) a la clase de problemas de diseño de redes de comunicaciones confiables modelados bajo el denominado Problema de Steiner Generalizado.

Considerando una red de comunicaciones con nodos distinguidos denominados *terminales*, el GSP consiste en diseñar de una subred de mínimo costo que verifique requisitos prefijados de conexión entre pares de nodos terminales. Usualmente la minimización del costo de las conexiones se contrapone con el objetivo de maximizar las propiedades de confiabilidad de la red. Como ejemplo, un modelo que no agregue un mínimo nivel de redundancia de caminos conducirá a una topología de árbol para la red, poco útil en escenarios reales, ya que no es capaz de soportar fallas en sus componentes. El GSP incorpora requisitos adicionales de conectividad para garantizar la alta confiabilidad en las comunicaciones que demandan los escenarios reales. El GSP ha sido poco estudiado en el pasado; nuestra contribución consiste en la aplicación de algoritmos evolutivos puros e híbridos, en sus modelos secuenciales y paralelos, en un intento por resolverlo con alta eficiencia numérica.

El artículo se organiza del modo que se describe a continuación. La sección II presenta el Problema de Steiner Generalizado, sus variantes y aplicaciones de técnicas evolutivas a variantes simples del GSP. La sección III describe los algoritmos considerados en el estudio. La sección IV presenta los detalles de implementación de los algoritmos y una descripción de la biblioteca sobre la cual fueron desarrollados. En la sección V se discuten los experimentos y se analizan los resultados obtenidos. La sección VI formula las conclusiones y líneas de trabajo futuro.

II. EL PROBLEMA DE STEINER GENERALIZADO

Esta sección ofrece la formulación del Problema de Steiner Generalizado, presenta variantes simplificadas del problema y resume antecedentes de aplicación de técnicas evolutivas a las variantes simples del GSP.

A. Formulación del GSP

La siguiente formulación del GSP se basa en el compendio de problemas de optimización NP de Kahn y Crescenzi [16].

Considérense los siguientes elementos :

- Un grafo no dirigido $G = (V,E)$, siendo V el conjunto de nodos y E el conjunto de aristas que representan a los enlaces bidireccionales de comunicación.

* Instituto de Computación, Universidad de la República, Herrera y Reissig 565, Montevideo, Uruguay, E-mail: sergion@fing.edu.uy

- Una matriz 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 vértices V.
- Una matriz n_T x n_T simétrica $R = r_{ij}$ con $i,j \in T$, cuyos elementos son enteros positivos que indican los requerimientos de conectividad –cantidad de caminos disjuntos– entre todo par de nodos terminales i y j.

El GSP plantea encontrar un subgrafo $G_T \subseteq G$ de costo mínimo, tal que todo par de nodos $i,j \in T$, sean r_{ij} arista-conexos en G_T, es decir que existan r_{ij} *caminos disjuntos*, que no comparten aristas, entre los nodos i y j en G_T. Sobre los nodos no terminales no se plantean requisitos de conectividad. Éstos, llamados *nodos de Steiner*, pueden formar parte o no de la solución óptima, de acuerdo a la conveniencia de utilizarlos.

B. Complejidad y variantes de problemas de Steiner

La complejidad del problema de Steiner obedece a la generalidad de su planteo, al exigir requisitos variables de conectividad entre pares de nodos terminales. Ciertas variantes simplifican estos requerimientos, la subclase de *Problemas de k-conexión* exigen un número fijo k de caminos disjuntos entre pares de nodos terminales. El caso más simple de problema de Steiner exige sólo un camino entre pares de nodos; la solución a este problema tiene topología de árbol y por ello se conoce como *Problema del Árbol de Steiner*.

El GSP pertenece a la clase de problemas NP difíciles [16]. El propio Problema del Árbol de Steiner, que plantea las restricciones más simples, es NP-completo [18]. La complejidad de los problemas de Steiner hace difícil su resolución mediante algoritmos exactos al aumentar el tamaño del problema. Por este motivo, se buscan alternativas utilizando heurísticas que permitan abordar instancias complejas y encontrar buenas soluciones en tiempos razonables.

C. Técnicas evolutivas aplicadas a la resolución de los problemas de Steiner

Si bien las técnicas evolutivas han sido usadas para abordar variantes simples del problema de Steiner, fuera de nuestro entorno de trabajo no existen antecedentes de aplicación al problema generalizado

Hesser et al. [12] realizaron la primer propuesta de optimización de árboles de Steiner utilizando un AG, utilizando una codificación basada en información espacial de los nodos de Steiner y una función de fitness que evalúa el costo del árbol asociado. Los resultados no fueron concluyentes, al no encontrar los autores diferencias significativas al comparar el AG con una heurística especializada para el problema.

Kapsalis et al. [17] presentaron un AG para el Problema del Árbol de Steiner, utilizando una codificación basada en representar los nodos e incluyendo soluciones no factibles en la población. Su AG integraba una función de penalización en la evaluación del fitness, para presionar a soluciones no factibles a ingresar a la región de factibilidad. Este método se reportó como exitoso para hallar soluciones al problema en grafos dispersos.

Julstrom estudió el Problema de Steiner Rectilíneo, donde los nodos se asumen ubicados en el plano y las conexiones deben ser horizontales o verticales. Este problema tiene aplicación en el diseño de redes, sistemas mecánicos y paquetes VLSI. Julstrom [14] propuso una codificación mixta (binaria y no binaria) para árboles de cubrimiento e introdujo un operador de cruzamiento para evitar pérdida de información. Años después, propuso un algoritmo híbrido que incorporó una heurística específica de inicialización, reportando mejores resultados y mejoras en la performance [15].

En su tesis de doctorado, Esbensen [6] extendió las ideas de Kapsalis et al [17] para el Problema del Árbol de Steiner. Impuso el trabajo con soluciones factibles evitando términos de penalización en la función de fitness. Sobre esta idea, propuso un AG aplicado al diseño de circuitos VLSI, obteniendo soluciones de calidad superior y performance competitiva respecto a dos heurísticas bien conocidas para el problema [7].

Diversos artículos recientes han abordado problemas sobre redes de comunicaciones utilizando AG para resolver problemas de Steiner.

Zhu et al. [25] propusieron un algoritmo híbrido para hallar árboles de Steiner, combinando la idea de Esbensen y un AG para optimizar requerimientos de conexiones multipunto. Hwang et al. [13] y Zhou et al. [24] propusieron AG aplicados al problema de ruteo multicasting. Ljubic et al. [21] presentó un AG híbrido aplicado al problema de biconectividad sobre redes. Galiasso y Wainwright [11] diseñaron un AG híbrido para hallar porcentajes de ancho de banda y un orden óptimo para procesar requisitos en un problema de ruteo multipunto. Wakabashi [23] presentó un AG para hallar árboles de Steiner rectilíneos, construidos usando un método basado en el algoritmo de Kruskal.

Ninguno de los trabajos mencionados abordó el caso generalizado, en su lugar todos trabajaron sobre variantes más simples como el Problema del Árbol de Steiner o los Problemas de k-conexión.

Un trabajo reciente por parte de uno de los autores [3] presentó un AG para resolver el Problema de Steiner Generalizado, basado en la idea de mantener la codificación y los operadores tan simples como fuera posible. En este artículo utilizaremos las mismas ideas al diseñar otros algoritmos evolutivos para el GSP.

III. ALGORITMOS CONSIDERADOS EN EL ESTUDIO

Esta sección presenta los algoritmos evolutivos estudiados: algoritmos genéticos en su formulación clásica y en su variante CHC, Simulated Annealing (SA), dos técnicas híbridas que combinan AG y SA y los conceptos de sus implementaciones paralelas.

A. Algoritmo Genético

La formulación clásica de un AG puede encontrarse en Goldberg [10]. Basado en el esquema de un algoritmo evolutivo de la Figura 1, el algoritmo genético define operadores de selección, cruzamiento y mutación para aplicar a la población en cada generación. Los algoritmos genéticos están ampliamente difundidos por su versatilidad para la resolución de problemas de optimización. El AG diseñado se basa en el presentado en el trabajo previo de uno de los autores [3].

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
mientras (no CriterioParada) hacer
  Padres = Seleccion(P(generacion))
  Hijos = Operadores de Reproduccion(Padres)
  NuevaPop = Reemplazar(Hijos,P(generacion))
  generacion ++
  P(generacion) = NuevaPop
retornar Mejor Solucion Hallada
```

Figura 1: Esquema de un algoritmo evolutivo.

B. Algoritmo CHC

El algoritmo CHC es una variante del AG clásico propuesta por Eshelman [8]. Utiliza una estrategia de selección elitista y el operador de cruzamiento uniforme (HUX), que intercambia exactamente la mitad de los bits diferentes entre dos individuos padre. Solo se permite el cruce entre individuos que difieran bit a bit una cierta distancia, inicializada en 1/4 del largo del cromosoma utilizado y disminuida en 1 en cada generación en que no se generan descendientes. La diversidad no se introduce mediante un operador de mutación, sino por un mecanismo de reinicialización, que se aplica al detectar la convergencia. La Figura 2 presenta un esquema del algoritmo CHC.

```
Inicializar(P(0))
generacion = 0
distancia = LargoCromosoma/4
Evaluar(P(0))
mientras (no CriterioParada) hacer
  Padres = Seleccion(P(generacion))
  Hijos = HUX(Padres)
  Evaluar(Hijos))
  NuevaPop = Reemplazar(Hijos,P(generacion))
  Si (NuevaPop == P(generacion)) entonces
      distancia --
  generacion ++
  P(generacion) = NuevaPop
  Si (distancia == 0) entonces
      Reinicializacion(P(generacion))
      distancia = LargoCromosoma/4
retornar Mejor Solucion Hallada
```

Figura 2: Esquema del algoritmo CHC

C. Simulated Annealing

Simulated Annealing es un método de búsqueda local basado en la simulación Monte Carlo para hallar la configuración más estable de un sistema físico de n partículas. La generalización del método a problemas de optimización fue propuesta por Kirkpatrick et al. [19].

SA no es una técnica basada en población, ya que mantiene una única solución candidata para el problema (análoga al estado actual de un sistema) con una función objetivo asociada (análoga a la función de energía) para la que se quiere hallar el mínimo global (análogo al estado más estable).

El espacio de búsqueda se explora mediante un operador de transición y para controlar la probabilidad de aceptar soluciones peores que la actual, SA utiliza un parámetro temperatura *T* que no tiene un análogo en el problema de optimización, por lo cual definir un *esquema de enfriamiento* que permita evitar mínimos locales es un arte. También los restantes parámetros del método (el valor inicial de *T*, el número de iteraciones realizadas en cada paso y el largo de la cadena de Markov utilizada) son usualmente determinados empíricamente. La Figura 3 presenta un esquema para el algoritmo SA.

```
Inicializar(T)
paso = 0
Sol = SolucionInicial()
Valor = Evaluar(Sol)
repetir
   repetir
      paso ++
      NuevaSol = Generar(Sol,T)
//Transición
      NuevoValor = Evaluar(NuevaSol)
      si Aceptar(valor,NuevoValor,T)
          sol = NuevaSol
          valor = NuevoValor
   hasta ((paso mod LargoCadenaMarkov)==0)
   T = Actualizar(T)
hasta (CriterioParada)
retornar Sol
```

Figura 3: Esquema del algoritmo Simulated Annealing

D. Algoritmos Híbridos

Las técnicas de hibridación refieren a la inclusión en un método de búsqueda de conocimiento dependiente del problema para mejorar el mecanismo de búsqueda [5]. Puede instrumentarse a nivel de la codificación y/o en operadores específicos (*híbridos fuertes*) o combinando diferentes algoritmos (*híbridos débiles*)

En este trabajo hemos utilizado esta última técnica, combinando los algoritmos AG y SA. En la primer propuesta (GASA1) el AG utiliza en cada generación al SA como un operador evolutivo. En la segunda propuesta (GASA2) se ejecuta el AG hasta su finalización y luego se seleccionan mediante torneo estocástico individuos de la población final, que se toman como soluciones iniciales para aplicarles el algoritmo SA.

E. Algoritmos Paralelos

Las implementaciones paralelas se han popularizado como un mecanismo para mejorar la eficiencia de los algoritmos evolutivos. Dividiendo la población entre varios elementos de procesamiento, los Algoritmos Evolutivos Paralelos (PEAs) permiten hallar resultados de alta calidad en tiempos razonables para problemas difíciles de resolver o de grandes dimensiones.

Para los algoritmos basados en población, nuestros PEAs se clasifican en el modelo de subpoblaciones distribuidas [2]. La población original se divide en varias subpoblaciones (*demes*), cada uno de los cuales ejecuta un EA serial donde los individuos solo son capaces de interactuar con otros en su mismo *deme*. Se define un operador adicional de *Migración* que permite el intercambio ocasional de individuos entre demes, introduciendo una nueva fuente de diversidad.

La Figura 4 presenta un esquema paralelo para un EA basado en población. *Emigrantes* denota al conjunto de individuos a intercambiar con otro deme, seleccionados de acuerdo a una política determinada por *SeleccMigracion*. El operador *Migración* intercambia los individuos entre demes de acuerdo a un grafo de conectividad definido entre ellos, generalmente un anillo unidireccional. *CondicionMigracion* determina cuándo se lleva a cabo el intercambio.

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
mientras (no CriterioParada) hacer
  Padres = Seleccion(P(generacion))
  Hijos = Operadores de Reproduccion(Padres)
  NuevaPop = Reemplazar(Hijos, P(generacion))
  generacion ++
  P(gener) = NuevaPop
  Si (CondicionMigracion)
    Emigrantes = SeleccMigracion(P(generacion))
    Inmigrantes = Migracion(Emigrantes)
    Insertar(Inmigrantes, P(generacion))
retornar Mejor Solucion Hallada
```

Figura 4: Esquema de un algoritmo evolutivo paralelo

Por otra parte, varios enfoques han sido propuestos para paralelizar el algoritmo SA [20]. Nuestra versión ejecuta varios SA seriales a partir de distintas soluciones iniciales, que cooperan esporádicamente intercambiando las mejores soluciones obtenidas.

IV. IMPLEMENTACIÓN

Esta sección explica la codificación del problema GSP y la función objetivo utilizada para su resolución mediante los EAs presentados en la sección previa. Luego se brindan detalles sobre los parámetros de configuración de los algoritmos. Por último, se presenta la biblioteca utilizada para la implementación de los algoritmos y se ofrecen los detalles de la plataforma de ejecución utilizada.

A. Codificación del problema

Se utilizó una codificación binaria simple basada en aristas para representar grafos que constituyen soluciones factibles del problema GSP. Una solución factible se representa como un arreglo de bits (indexado entre 0 y $|E|$-1); cada bit en la representación indica la presencia o ausencia de una arista en el grafo original. La figura 5 presenta un grafo de ejemplo y su codificación en la representación binaria propuesta.

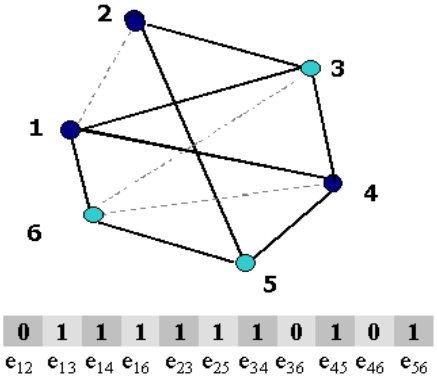

Figura 5: Codificación binaria basada en aristas.

Utilizando esta representación binaria los operadores de evolución resultan sencillos de implementar, pero presentan el inconveniente de que pueden generar soluciones no factibles del GSP. Hemos adoptado la propuesta de Esbensen en sus trabajos sobre el Problema del Árbol de Steiner [6,7], descartando las soluciones no factibles. Esta decisión simplifica el algoritmo, evitando la tarea de cuantificar cuán lejos de una solución factible se encuentra y de introducir un modelo de penalización para el fitness.

El chequeo de factibilidad tiene dos componentes. Una sencilla heurística descarta soluciones si los grados de cada nodo terminal son menores que el máximo requerimiento de conexión para ese nodo. Si los grados son compatibles con los requerimientos de conexión, se utiliza una variante del algoritmo de Ford–Fulkerson [9] para hallar los caminos entre cada par de nodos terminales, considerando uno como fuente y el otro como pozo. Asumiendo la capacidad de las aristas unitaria, el flujo máximo entre fuente y pozo coincide con el número máximo de caminos disjuntos entre los nodos terminales. Si éste es menor que el valor del requerimiento correspondiente, la solución es no factible.

La población es inicializada mediante un procedimiento que elimina aleatoriamente hasta un 5% de las aristas del grafo original. Luego se aplica el chequeo de factibilidad para eliminar de la población inicial soluciones no factibles y si es necesario se aplica nuevamente el proceso para hallar una nueva solución inicial.

B. Función de fitness

La función de fitness evalúa el costo del grafo representado. El problema de minimización se mapea a uno de maximización restándose el costo del grafo del valor constante C_{ORIG}, el máximo valor de costo para el grafo considerado.

$$[fitness] \quad f = C_{ORIG} - \sum_{0}^{|E|-1} \left[EDGE(i) * C(i) \right]$$

Ecuación 1: función de fitness para el GSP.

En la Ecuación 1, la función $C : N \rightarrow R$ devuelve el costo de una arista y la función $EDGE : N \rightarrow [0,1]$ retorna el valor binario correspondiente a la arista que ocupa la posición *i*-ésima en la representación.

C. Configuración de parámetros

Los operadores utilizados por los EA implementados corresponden a:

- AG : Selección proporcional, cruzamiento de dos puntos, mutación de inversión.
- SA : Transición que invierte los valores de cinco aristas, esquema proporcional de decaimiento para la temperatura ($T_k = \alpha.T_{k-1}$ con $\alpha = 0.99$).
- CHC : Cruzamiento uniforme, selección elitista, reinicialización basada en transición de SA.

No se realizaron experimentos para configurar los parámetros para cada EA. Se decidió trabajar con valores del trabajo utilizado como referencia [3], donde se hallaron valores óptimos para el tamaño de población y probabilidades de cruzamiento y mutación para un AG que utilizó la misma codificación del problema.

Se definió un esfuerzo prefijado de 2000 generaciones como criterio de parada para los EAs. Se descartó el uso de un criterio basado en detectar la convergencia porque no conduciría a una comparación justa de los tiempos de ejecución de los algoritmos.

Dado que no se conocían referencias sobre la aplicación de SA al GSP, luego de algunos experimentos iniciales se decidió usar 10000 iteraciones y una cadena de Markov de 250 pasos.

En los EAs basados en población se utilizó un conjunto de 120 individuos. La probabilidad de cruzamiento en el AG y las dos variantes de GASA se fijó en 0.9 y la probabilidad de mutación en 0.01. Para CHC, se fijó la probabilidad de aplicación de HUX en 0.8, mientras que el proceso de reinicialización involucró el 35% de la población.

En ambas variantes de GASA se aplica el operador SA con probabilidad 0.01. Para GASA1 se utilizó una cadena de Markov corta (de largo 10) y 20 iteraciones, tratando de reducir el costo computacional de aplicar SA como operador interno al AG. Para GASA2, donde se aplica SA luego que finalizado el AG, se usó una cadena de Markov de largo 10 y 100 iteraciones.

D. La biblioteca MALLBA

El proyecto MALLBA [1] constituye un esfuerzo para desarrollar una biblioteca de algoritmos para optimización capaz de tratar eficientemente con el paralelismo, manteniendo una interfaz amigable para el usuario. Los algoritmos presentados en este trabajo están implementados en la biblioteca MALLBA como *esqueletos de software*, que deben ser instanciados con las características del problema por parte del usuario. Los detalles de los métodos de resolución, la interacción con el problema y el paralelismo se implementan mediante clases C++ *requeridas* y *provistas* que abstraen a las entidades involucradas en cada método de resolución:

- *Clases provistas*: implementan aspectos internos del algoritmo, de un modo independiente al problema. Las principales clases provistas son `Solver` (el algoritmo) y `SetUpParams` (para fijar parámetros).
- *Clases requeridas*: especifican la información relevante del problema a resolver. Cada esqueleto incluye las clases requeridas `Problem` y `Solution` que encapsulan las entidades dependientes del problema necesarias para los métodos de resolución.

La infraestructura del proyecto MALLBA se compone de clusters de computadores localizados en Málaga, La Laguna y Barcelona en España, conectados por redes de comunicaciones de alta velocidad. La biblioteca MALLBA está disponible en `http://neo.lcc.uma.es/mallba/easy-mallba.`

E. Plataforma de ejecución

Se utilizó un cluster de ocho computadores Intel Pentium III a 2.4 GHz, cada uno con 512 Mb RAM y sistema operativo SuSE Linux 8.0, conectados por una LAN Fast Ethernet a 100Mb/sec.

V. RESULTADOS

Esta sección introduce el conjunto de problemas diseñado para evaluar los algoritmos estudiados y se presentan y analizan los resultados de los modelos seriales y paralelos, desde el punto de vista de su calidad de resultados y de su eficiencia.

A. Problemas de prueba

Dado la escasa literatura existente sobre la aplicación de técnicas heurísticas para la resolución del GSP, no existen conjuntos de problemas de prueba estándares.

Para evaluar los algoritmos se diseñaron tres instancias, seleccionando aleatoriamente topologías de conexión, costos y requerimientos (elegidos al azar entre 0 y 4). La tabla 1 resume las características de los grafos de prueba, indicando el número total de nodos, de terminales, de aristas y el grado de conectividad promedio (número de aristas sobre el número de aristas del grafo completo).

	Nodos	*Terminales*	*Aristas*	*Grado Conectividad Promedio*
grafo 100-10	100	10	500	0.1
grafo 75-25	75	25	360	0.13
grafo 50-15	50	15	249	0.2

Tabla 1: Características de los grafos de prueba

Los problemas diseñados pueden considerarse "representativos" para redes de comunicaciones de mediano tamaño con requerimientos de conexión variables. Los grafos se designan por un nombre que refiere a la cantidad total de nodos y a la de terminales, por ejemplo *grafo 50-15* para el grafo que tiene 50 nodos de los cuales 15 son terminales. Los problemas de prueba y el generador aleatorio de grafos se encuentran disponibles públicamente en `http://www.fing.edu.uy/cecal/hpc/gsp`.

Se dispone de los resultados para el *grafo 100-10* del trabajo previo [3], que usaremos como base para evaluar la calidad de los resultados de los algoritmos

Se diseñaron dos variantes del algoritmo CHC. En CHC1 la reinicialización descarta soluciones no factibles. En la variante mejorada CHC2 la reinicialización se aplica iterativamente hasta lograr una solución factible.

B. Resultados de los algoritmos seriales

Los resultados de los algoritmos seriales se resumen en la Tabla 2. Se presentan los promedios, los mejores valores de fitness y la desviación estándar para cada algoritmo sobre las 30 ejecuciones independientes realizadas.

SA muestra los peores resultados, notoriamente inferiores a los de las técnicas basadas en población. Al menos para la implementación propuesta, SA parece incapaz de manejar la complejidad del GSP. Se realizaron experimentos laterales tratando de mejorar el mecanismo de búsqueda del algoritmo, utilizando operadores de transición que introducen mayor diversidad (modificando un número mayor de aristas) pero las mejoras no fueron significativas, aún incrementando el número de iteraciones.

El AG clásico obtuvo buenos resultados para las tres instancias consideradas, pero no fue capaz de superar el mejor valor conocido para el *grafo100-10* (4561 obtenido en [3]). El promedio de fitness alcanzado se encuentra lejano al mejor valor mencionado, sugiriendo que aún resta trabajo por realizar para calibrar adecuadamente el AG.

GASA1 superó en resultados al AG clásico, mientras que GASA2 no lo superó significativamente. Esto sugiere que la aplicación de SA al finalizar el AG no mejora la búsqueda, pero sí es efectiva al aplicarse como operador interno, ofreciendo una nueva fuente de diversidad que ayuda a alcanzar mejores resultados. El promedio y el mejor fitness de GASA1 superan a los del AG, pero como la mejora es inferior a la desviación estándar de los resultados, no puede considerarse significativa estadísticamente.

El algoritmo CHC logró buenos resultados La variante CHC1 obtuvo resultados similares a los del AG para el *grafo100-10*, pero mostró problemas de convergencia prematura sobre los otros casos de prueba, debido a su impreciso operador de reinicialización. La variante CHC2, con su operador mejorado de reinicialización, obtuvo consistentemente los mejores resultados, superando significativamente los valores del AG y de los híbridos implementados. Este resultado llevó a conjeturar que el uso del operador HUX era una idea promisoria a profundizar. En el análisis de operadores de cruzamiento realizado posteriormente por Pedemonte y Nesmachnow [22], la familia de operadores de cruzamiento uniforme permitió alcanzar mejores resultados para el problema que los operadores de cruzamiento multipunto.

	GASA1	*GASA2*	*CHC1*	*CHC2*	*GA*	*SA*
			Grafo100-10			
Promedio	4491.5	4471.5	4449.0	4602.0	4469.5	4186.0
Mejor	4549.0	4507.0	4567.0	4634.0	4531.0	4302.0
Desv. Est.	36.7	30.2	42.6	15.1	28.2	48.6
			Grafo75-25			
Promedio	5363.9	5335.4	3657.0	5479.0	5308.2	5171.7
Mejor	5429.3	5427.1	4574.9	5521.8	5406.7	5249.0
Desv. Est.	39.8	49.2	372.6	21.2	50.2	51.91
			Grafo50-10			
Promedio	9373.3	9330.6	7646.3	9514.2	9316.8	9177.2
Mejor	9474.3	9538.3	9143.0	9596.5	9456.0	9321.9
Desv. Est.	70.7	85.3	726.3	43.2	85.8	114.6

Tabla 2: Resultados de los algoritmos seriales.

C. Resultados de los algoritmos paralelos

La Tabla 3 resume los resultados de los algoritmos paralelos utilizando ocho demes, cada uno ejecutando sobre un equipo del cluster. Se ofrecen los promedios de valores de fitness y la desviación estándar para 30 ejecuciones independientes de cada algoritmo. Debido al alto número de combinaciones de algoritmos e instancias de prueba, sólo se realizaron ejecuciones paralelas para los algoritmos cuyo comportamiento fue promisorio en los experimentos seriales.

	GASA1	***CHC2***	***GA***	***SA***
		grafo 50-15		
Promedio	4529.0	4436.5	4489.0	4170.5
Mejor	4608.0	4566.0	4537.0	4289.0
Desv. Est.	32.5	51.3	24.5	48.2
		grafo 75-25		
Promedio	5415.5	5353.2	5379.4	5174.2
Mejor	5440.9	5408.2	5447.2	5290.9
Desv. Est.	28.4	34.5	38.8	41.4
		grafo 50-15		
Promedio	9473.1	9418.2	9393.3	9266.3
Mejor	9569.7	9596.5	9535.3	9441.0
Desv. Est.	34.0	70.7	86.7	81.6

Tabla 3: Resultados de los algoritmos paralelos.

El objetivo de los experimentos consistió en evaluar si los modelos paralelos implementados eran capaces de mejorar la calidad de los resultados seriales, tomando en cuenta que utilizan un mecanismo diferente de exploración del espacio de soluciones [4].

Asimismo, se buscó determinar si los PEAs implementados sobre la biblioteca MALLBA mantenían un *speedup* casi lineal, como el observado en el AG específico [3].

En la Tabla 3 se observa que los PEAs mejoraron la calidad de resultados respecto a los modelos seriales, excepto para la variante del algoritmo CHC estudiada. Trabajando con demes de pocos individuos se reduce la diversidad y el CHC distribuido sufre los problemas detectados para la variante serial (CHC1), aún trabajando con el operador mejorado de reinicialización.

Los resultados comparativos entre los modelos seriales y paralelos se presentan en la Tabla 4, que muestra valores promedio y mejores valores de fitness. Los factores de mejora (FM) para GA y GASA1 no son elevados dado que los algoritmos seriales son muy precisos, al utilizar un alto número de generaciones como criterio de parada. Sin embargo, el factor de mejora superó a la desviación estándar en todos los casos, indicando que los modelos paralelos realmente producen resultados más precisos que los seriales al resolver el GSP. Para los algoritmos donde se detectaron mejoras, se realizó el test de Kruskal-Wallis para estudiar los promedios. Para GASA1, los p-values son inferiores a 0.01, y puede considerarse que las diferencias entre promedios de los modelos paralelo y serial son significativas al nivel de 1%.

	GASA1	***CHC2***	***GA***	***SA***
	grafo100-10			
Promedio Serial	4491.5	4602.0	4469.5	4186.0
Promedio Paralelo	4529.0	4436.5	4489.0	4170.5
FM Promedio	1.008	0.964	1.004	0.996
p-value	$1.8\ 10^{-6}$	n/a	0.035	n/a
Mejor Serial	4549.0	4634.0	4531.0	4302.0
Mejor Paralelo	4608.0	4566.0	4537.0	4289.0
FM Mejor	1.013	0.986	1.001	0.997
	grafo 75-25			
Promedio Serial	5363.9	5479.0	5308.2	5171.8
Promedio Paralelo	5415.5	5353.2	5379.4	5174.2
FM Promedio	1.010	0.977	1.014	1.001
p-value	$5.8\ 10^{-6}$	n/a	$1.9\ 10^{-6}$	n/a
Mejor Serial	5429.3	5521.8	5406.7	5249.0
Mejor Paralelo	5440.9	5408.2	5447.2	5290.9
FM Mejor	1.002	0.980	1.008	1.008
	grafo50-15			
Promedio Serial	9373.3	9514.2	9316.8	9177.2
Promedio Paralelo	9473.1	9418.3	9393.3	9266.3
FM Promedio	1.011	0.990	1.008	1.010
p-value	$8.9.10^{-9}$	n/a	$5.1\ 10^{-4}$	n/a
Mejor Serial	9474.3	9596.5	9456.0	9321.9
Mejor Paralelo	9569.7	9596.5	9535.3	9441.0
FM Mejor	1.010	1.000	1.008	1.013

Tabla 4: Resultados comparativos.

Aunque la mejora de performance no fue un objetivo del trabajo, se analizaron los tiempos de ejecución de los algoritmos seriales y paralelos. La Tabla 5 presenta los tiempos promedio de ejecución y un valor calculado de *Eficiencia = Speedup/#Equipos*.
Si bien los PEAs muestran un *speedup* sublineal, GASA1 y GA tienen un buen valor de eficiencia, mientras que CHC2 mostró una eficiencia muy pobre, mostrando nuevamente un comportamiento diferente al de los otros EAs basados en población.

	GASA1	***GA***	***CHC2***	***SA***	***GASA2***	***CHC1***
	grafo 100-10					
Serial	520.0	122.8	59.4	18.6	139.2	52.6
Paralelo	81.8	24.2	28.7	24	*	*
Eficiencia	0.79	0.63	0.26	*	*	*
	grafo 75-25					
Serial	462.6	125.4	51.0	9.0	126	11.4
Paralelo	81.0	24.0	20.4	14.4	*	*
Eficiencia	0.71	0.65	0.31	*	*	*
	grafo 50-15					
Serial	1229.4	310.2	275.4	31.8	319.8	31.2
Paralelo	219.6	59.4	74.4	28.8	*	*
Eficiencia	0.70	0.65	0.46	*	*	*

Tabla 5: Tiempos de ejecución promedio (minutos)

La Figura 6 ofrece la evolución de valores de fitness en las generaciones para CHC2, GASA1 y GA durante una ejecución representativa sobre el *grafo100-10*. El análisis permitió a GASA1 y ambas variantes de CHC mostrar una gran ventaja, al obtener valores relativamente altos de fitness en pocas generaciones.

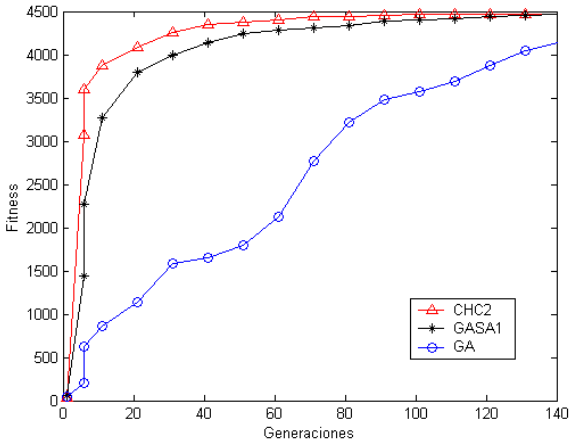

Figura 6: Evolución del fitness.

VI. Conclusiones y Trabajo Futuro

En este trabajo hemos presentado varias técnicas evolutivas aplicadas a la resolución de un problema que modela el diseño de redes de comunicaciones confiables. Analizando los resultados para el conjunto de problemas de prueba estudiado es posible extraer conclusiones sobre la aplicabilidad de los algoritmos implementados para la resolución del GSP.

El método SA no produjo resultados precisos utilizando el sencillo operador de transición propuesto, aun permitiendo un elevado número de iteraciones.

El AG reportó buenos resultados sobre las instancias de prueba consideradas, pero no logró superar el mejor valor conocido anteriormente.

La técnica de hibridación que incorpora a la transición de SA dentro del AG permitió mejorar la calidad de resultados y acelerar la búsqueda, aunque la mejora es poco significativa en el largo plazo. Por su parte, aplicar el SA luego de finalizado el AG no conduce a mejoras significativas en los resultados.

El algoritmo CHC se presentó como una técnica muy promisoria para la resolución del GSP. Tomando ventaja de su mecanismo especial de cruzamiento, logró los mejores resultados y alcanzó muy buenos individuos en pocas generaciones. La primer versión diseñada presentó problemas de convergencia prematura, pero luego de mejorar el mecanismo de reinicialización se obtuvo un método potente y robusto para la resolución del GSP.

Se confirmó que el modelo de EAs de poblaciones distribuidas es capaz de obtener mejores resultados que el modelo serial para el GSP. Aunque los algoritmos presentaron un *speedup* sublineal, tanto el AG como GASA1 mostraron buenos valores de eficiencia al utilizar 8 demes.

El estudio ha dejado algunos aspectos inconclusos que plantean líneas de trabajo futuro. Respecto al comportamiento de los algoritmos, debe estudiarse la contribución de los modelos de hibridación, tomando en cuenta el costo computacional extra que demandan. Asimismo, debe analizarse la influencia del operador de reinicialización en los resultados del algoritmo CHC y su adaptación a poblaciones distribuidas. Diseñar un eficiente operador de reinicialización que provea la diversidad necesaria no es una tarea trivial, y plantea un desafío a afrontar mediante estudios teóricos y experimentales. El aspecto referente a la eficiencia de los modelos seriales y paralelas no fue afrontado en este trabajo. En este sentido, las implementaciones deben optimizarse para ser capaces de abordar instancias más complejas del GSP. Relacionado con este aspecto, debe investigarse la escalabilidad de los modelos paralelos, y su posible utilidad para resolver instancias complejas empleando el poder computacional de grandes clusters.

VII. Referencias

[1] E. Alba, C. Cotta, *Optimización en entornos geográficamente distribuidos. Proyecto MALLBA*, Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados, pp. 38–45, 2002.

[2] E. Alba, M. Tomassini, *Parallelism and Genetic Algorithms*, IEEE Transactions on Evolutionary Computation 6, 5, pp. 443-462, 2002.

[3] M. Aroztegui, S. Arraga, S. Nesmachnow, *Resolución del Problema de Steiner Generalizado utilizando un Algoritmo Genético Paralelo*, Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, pp 387-394, 2003.

[4] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic, 2001.

[5] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[6] H. Esbensen, *Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm*. Networks 26, pp. 173-185, 1995.

[7] H. Esbensen, P. Mazumder, *A Genetic Algorithm for the Steiner Problem in a Graph*, European Design and Test Conference, pp. 402-406. 1994.

[8] L. Eshelman, *The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination*, Foundations of Genetic Algorithms,. pp. 265-283, 1991.

[9] L. Ford, D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, 1962

[10] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.1989.

[11] P. Galiasso, R. Wainwright, *A hybrid genetic algorithm for the point to multipoint routing problem with single split paths.* Proceedings of the ACM Symposium on Applied Computing 2001, pp. 327-332. 2001.

[12] J. Hesser, R. Männer, O. Stucky, *On Steiner Trees and Genetic Algorithms*. Lecture Notes in Computer Science 565, pp. 509-525, 1991.

[13] R. Hwang, W. Do, S. Yang, *Multicast Routing Based on Genetic Algorithms*. Journal of Information Science and Engineering, 16, 6, pp. 885-901, 2000.

[14] B. Julstrom, *A genetic algorithm for the rectilinear Steiner problem.* Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 474-480, 1993.

[15] B. Julstrom, *Encoding rectilinear Steiner trees as lists of edges.* Proceedings of the ACM Symposium on Applied Computing, pp. 356-360, 2001.

[16] V. Kahn, P. Crescenzi, *A compendium of NP optimization problems*, Disponible en línea http://www.nada.kth.se/theory/problemlist.html. Consultada setiembre 2003

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

[18] R. Karp, *Reducibility among combinatorial problems*. Complexity of Computer Communications, pp. 85-103, Plenum Press, 1972.

[19] S. Kirkpatrik, C. Gelatt, M. Vecchi, *Optimization by simulated annealing*, Science, pp. 671–680, 1983.

[20] G. Kliewer, A General Software Library for Parallel Simulated Annealing, Proceedings of EURO Winter Institute on Metaheuristics in Combinatorial Optimisation, 2000.

[21] I. Ljubic, J. Kratica. *A genetic algorithm for biconnectivity augmentation problem*. Proceedings of the IEEE Congress on Evolutionary Computation, pp. 89-96, 2000.

[22] M. Pedemonte, S. Nesmachnow. *Estudio Empírico de Operadores de Cruzamiento en un Algoritmo Genético Aplicado al Problema de Steiner Generalizado*, a publicarse en Actas del Congreso Argentino de Ciencias de Computación, CACIC 2003.

[23] S. Wakabayashi, *A Genetic Algorithm for the Rectilinear Steiner Tree Problem in VLSI Interconnect Layout*. Journal of the Information Processing Society of Japan, Vol. 43, 05, 019, 2002.

[24] X. Zhou, C. Chen, G. Zhu, *A Genetic Algorithm for Multicasting Routing Problem*. Proceedings of the IEEE International Conference on Communication Technology, Vols. I & II, pp. 1248-1253, 2000.

[25] L. Zhu, D. Schoenefeld, R. Wainwright, *A Genetic Algorithm for the Point to Multipoint Routing Problem with Varying Number of Requests* Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 171-176, 1998.