ALGORITMO EVOLUTIVO PARA EL PROBLEMA DE ÁRBOL DE EXPANSIÓN MÍNIMA (MST)

Rosmeri Mayta H.*

INTRODUCCIÓN

El MST tiene una historia venerable en la optimización combinatoria. Fue formulado inicialmente por Boruvka en 1926 quien se dice tuvo que aprender de éste durante la electrificación del sur de Moraria donde él proporcionó una solución para hallar la distribución más económica a través de una red de una línea de energía.

Desde entonces la formulación del MST ha sido aplicada en numerosos problemas combinatorios tales como: problemas de transporte, diseño de redes de telecomunicaciones, sistemas distribuidos y otros. Al mismo tiempo algunos algoritmos de tiempo polinomial fueron desarrollados para su resolución por Prim, Kruskal, Dijkstra y Sollin. Los problemas que son extensiones del MST son generalmente problemas NP-Hard para los cuales no existen algoritmos de orden polinomial que los resuelvan. Debido a su complejidad se han venido empleando algoritmos evolutivos para su solución.

Este problema es importante en dos aspectos, por el lado teórico, el MST pertenece al conjunto de problemas definidos como NP-Completos [5], por lo que dado su complejidad es importante encontrar técnicas estocásticas que encuentren soluciones aceptables en tiempos adecuados, ya que las técnicas determinísticas tienen un comportamiento exponencial y en las técnicas heurísticas conocidas se degrada la calidad de la solución conforme crece el tamaño del problema.

Por el lado práctico, problemas como encontrar la configuración óptima de redes de cómputo, redes de carreteras, o en general problemas de flujo son equivalentes al problema del MST [5].

La promesa de calidad, eficiencia y robustez de los «algoritmos genéticos» (AG) es una idea que atrae a mucha personas para trabajar con ellos, sobre todo porque los métodos tradicionales son buenos para ciertas clases de problemas específicos, pero cuando un problema no se adapta a tales métodos, encontrar la solución al problema puede ser frustrante.

Las técnicas estocásticas, al recorrer el espacio de búsqueda del problema, deben de tener un balance adecuado entre las partes de exploración y explotación que se hace sobre el espacio de soluciones posibles. La exploración nos permite que el algoritmo se mueva sobre todo el espacio de soluciones, con lo que evitamos los mínimos locales. Por otro lado, la explotación nos permite, al encontrar una posible solución óptima, moverse en el espacio circundante a ésta para encontrar la que más se acerque el óptimo en dicho vecindario.

De estas técnicas, los algoritmos genéticos cuentan con un buen balance entre la exploración y explotación, ya que aunque el AG esté cercano al punto de convergencia, el operador permite explorar individuos -posibles soluciones- con diferentes características. Por otra parte, el operador de mutación permite al AG una exploración sobre un área determinada del espacio de soluciones.

FUNDAMENTOS DE MST

Considere un grafo conectado y no dirigido G = (V,E). Donde $V = (v_1, v_2, \ldots v_n)$ es un conjunto finito de vértices (nodos) que pueden representar terminales o estaciones de telecomunicación; y $E = \{ e_{ij} | e_{ij} = (v_i, v_j), v_i, v_j \in V \}$ en un conjunto finito de enlaces que representan la conexión entre los terminales o estaciones. Cada enlace tiene un número positivo real asociado denotado por $W = \{ w_{ij} | w_{ij} = w(v_i, v_j), w_{ij} > 0, v_i, v_j \in V \}$ representando distancia, costo, etc.

Un árbol de expansión es un mínimo conjunto de enlaces de E que conectan todos los nodos en V y por lo tanto al menos un árbol de expansión puede ser encontrado en un grafo G. El mínimo árbol de expansión denotado por T* es un árbol de expan-

* Ingeniero Industrial. Instituto de Investigación de la Facultad de Ingeniería Industrial. UNMSM.
E-mail: iifi@unmsm.edu.pe

sión cuyo peso total de todos los enlaces es mínimo. Es decir:

$$T^* = \min_{e_{ij} \in E} \sum W_{ij}$$

Donde T es un conjunto de árboles de expansión del grafo G. El grado de un nodo es el número de enlaces conectados a éste. Un nodo hoja tiene solamente un enlace conectado; de tal manera que el grado de un nodo hoja en un árbol en uno y el de los otros nodos más de uno.

Luego para que un árbol de expansión sea mínimo debe cumplir una de tales condiciones:

a. Ser un subgrafo de G conectado con n-1 enlaces.
b. Ser un subgrafo de G sin ciclos con n-1 enlaces.
c. La sumatoria de los pesos de todos los arcos asociados al subgrafo de G es el menor de todos los subgrafos asociados al grafo G que cumplen con las mismas restricciones.

¿QUÉ ES UN AG?

Son algoritmos de búsqueda basados en la mecánica de la selección natural y de la genética natural. Estos combinan la supervivencia de los individuos más aptos entre las cadenas de estructuras con un intercambio de información aleatorio para formar un algoritmo de búsqueda.

ESTRUCTURA DE UN AG

El AG simple o básico comprende cinco pasos principales [5]:

1. Crear población inicial de cromosomas, ya sea en forma aleatoria o mediante cierto conocimiento inherente al problema en sí. Es muy importante que la población inicial tenga diversidad para que el algoritmo pueda explorar todo el espacio de soluciones, lo que se espera se cumpla dadas las características de la inicialización. Además, si se tiene conocimiento del problema, éste puede ser insertado en la población inicial, con lo que se espera conseguir un mejor tiempo de convergencia del algoritmo.

2. Evaluación, consiste en calcular la aptitud mediante una función definida para el problema en particular. El objetivo de esta función es proveer una medida numérica de la calidad de un cromosoma dado.

3. Selección o explotación, se escogen de forma aleatoria los cromosomas con mejor función de aptitud una o varias veces y se insertan en un depósito de cruza.

4. Enseguida se realiza una exploración, lo que consiste en la aplicación de operadores de recombinación y mutación, con lo que se obtienen dos hijos los que según el tipo de algoritmo usado, reemplazarán a sus padres en la siguiente generación.

5. La población es llenada con los nuevos cromosomas creados. Se repiten estos pasos hasta que el criterio de paro se cumpla, v.g. un número de generaciones determinado [5].

CONVERGENCIA DEL ALGORITMO

Dado que el algoritmo genético opera con una población en cada iteración, se espera que el método converja de modo que al final del proceso la población sea muy similar, y en el infinito se reduzca a un sólo individuo.

Utilizar GAP generacional en la implementación del los algoritmos genéticos garantiza que no se perderán las mejores soluciones de cada generación y esto contribuye a que se preservaran los mejores individuos y estos podrán aportar su carga genética a futuras generaciones que deberían ir aumentando su aptitud, es decir, ir aproximándose mucho más rápido a la solución.

Utilizar Función Penalty en la implementación del los algoritmos genéticos garantiza que se puede explorar en las regiones factible e infactibles, debido a que algunas regiones infactibles pueden proporcionar mayor información acerca de la solución óptima que otras soluciones factibles, se debe mantener un balance entre *Preservación de la Información* (manteniendo soluciones infactibles) y *Presión Selectiva* (rechazar algunas soluciones infactibles) [4].

CONSIDERACIONES AL USAR AG

Como cualquier otro algoritmo, el uso de un AG para resolver un problema a ciegas puede resultar desafiante, por lo que resulta de vital importancia usar el conocimiento del sistema a optimizar para determinar si el AG tendrá un desempeño adecuado.

Es importante observar que aún para algunos tipos donde se sugiere que otras técnicas trabajarán mejor que los AG, con suficiente experimentos, se podrá obtener buen desempeño en la búsqueda usando AG. Por supuesto, tomar cualquier método de optimización para trabajar de una manera óptima requiere alguna experimentación con sus parámetros de configuración y selecciones inadecuadas de éstos conducirán a un bajo rendimiento en la búsqueda. Cuando no se conoce mucho acerca de la superficie de respuesta y calcular el gradiente es

computacionalmente intensivo o numéricamente inestable muchas personas prefieren usar métodos de optimización como AG, SA y Simplex los cuales no requieren información del gradiente [5].

Por el contrario, para las aplicaciones donde el cálculo del vector gradiente es numéricamente preciso y rápido, no se recomienda usar los AG, ya que éstos alcanzarán la región óptima mucho más lento que los métodos hill-climbing. Otro tipo de aplicaciones no recomendadas para los AG son aquellas que requieren encontrar el óptimo global exacto, dada la característica de los AG de ser buenos encontrando la región óptima global pero enfrentando problemas algunas veces para localizar el óptimo exacto.

Una de las dificultades más comúnmente mencionadas con los AG comparado con las técnicas hill-climbing es que generalmente los AG requieren más evaluaciones de la función de aptitud. Si la superficie de respuesta es bastante suave entonces un método hill-climbing, v.g. el método Simplex, superará al AG para un número de evaluaciones dado [5].

PROBLEMA

Elaborar un programa utilizando algoritmos evolutivos para hallar una solución al problema MST que se presenta a continuación (figura 1):

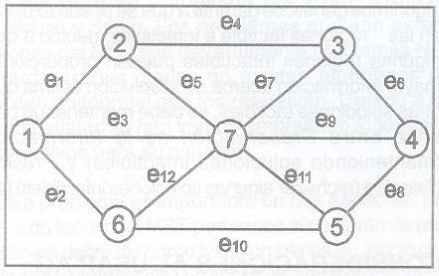

Figura 1. Grafo pesado no dirigido.

Las especificaciones y requerimientos de la implementación son los siguientes:

a. Representación genética. Se empleará la denominada Codificación Enlace en donde se asocia un índice k con cada enlace es decir, E = { ek}, k= 1,2,...,k en donde k es el número de enlaces en un nodo. De tal manera que una cadena de bits puede representar una solución candidata indicando cuales enlaces son usados en un árbol de expansión como se ilustra en la figura 2.

e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
0	1	0	1	1	0	1	0	1	0	0	1

Figura 2. Representación genética de los individuos.

b. Cruce. Un punto.

c. Mutación. Realizar mutación como una perturbación aleatoria dentro del rango permisible de enteros entre 1 y n (n = números de nodos en un grafo dado).

d. Selección. Rueda de la ruleta.

e. Parámetros del algoritmo (ver cuadro 1).

Cuadro 1. Parámetros del algoritmo.

GAP-Generacional	10%
Tamaño Población	50%
Probabilidad de Cruce	0.5
Probabilidad de Mutación	0,01
Generaciones	500
Numero de Corridas	30

f. Elaborar un gráfico en donde pueda apreciarse la mejor aptitud en cada corrida del algoritmo.

El algoritmo implementado recibirá un grafo como el mostrado en la figura 1 y determinará el árbol de mínima expansión sabiendo que los costos asociados son como se ilustra en el cuadro 2.

Cuadro 2. Costos asociados al MTS.

Nodo	Enlace						
	1	2	3	4	5	6	7
1	0	224	0	0	0	300	539
2	224	0	200	0	0	0	539
3	0	200	0	400	0	0	600
4	0	0	400	0	400	0	200
5	0	0	0	400	0	600	447
6	300	0	0	0	600	0	283
7	539	539	600	200	447	283	0

En donde cero (0) indica sin conexión directa. Asimismo, en la figura 3 se muestra el resultado de colocar los pesos asociados a cada arco del grafo de la figura 1.

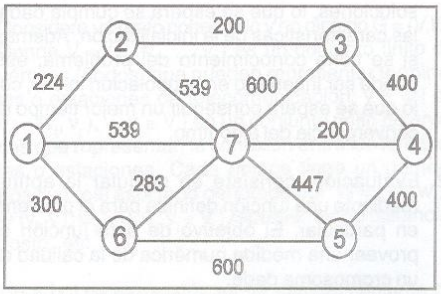

Figura 3. Grafo con pesos asignados a cada arco.

Debe ser implementado en C++ Builder, pues presenta una interfaz gráfica donde están predetermi-

nados los parámetros de entrada del algoritmo, estos pueden se cambiados a conveniencia. Será construido en una forma que permita el proceso de cualquier grafo dado suministrado.

CONCLUSIÓN

Por varias razones, los AG son muy atractivos como herramientas para varias tareas de optimización. Primero, presentan una abstracción del material y los operadores genéticos tal y como se presenta en la naturaleza. Segundo, se cree que los AG, como procedimientos de búsqueda multi-punto, pueden encontrar soluciones mucho mejores en tiempos más cortos que los procedimientos clásicos de búsqueda de un punto. Tercero, dado que los AG trabajan al nivel de la codificación, es difícil de que obtengan resultados engañosos aún cuando la función puede ser muy difícil para los esquemas tradicionales. Y cuarto, se piensa que los AG entregan un alto rendimiento, dado que muestrean los esquemas en los cromosomas de una manera inherentemente paralela.

BIBLIOGRAFÍA

1. **Herrera et al. 1994.** Algoritmos Genéticos: Fundamentos, extensiones y aplicaciones. Universidad de Granada. España.
2. **Fonseca, Carlos & Fleming, Peter. 1995.** Multiobjective optimization and multiple constraint handling with evolutionary algorithms. USA.
3. **Goldberg, David. 1997.** Genetic algorithms in search, Optimization & Machine Learning. Addison-Wesley Co. Inc. Reading. Massachusetts, USA.
4. **Michalewicz, Zbignew. 1994.** Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag. Alemania.
5. **García R., Agustín. 2000.** Un algoritmo genético para el Problema del Árbol de Expansión Mínima. Paper de Informática y Ciencias de la Computación. México.