

**k - Árbol generador de peso mínimo**  
**Algoritmo del Artillero Innovador**

**Seminario de Ciencias de la Computación B**

Canek Peláez Valdés

Universidad Nacional Autónoma de México

Sánchez Correa Diego Sebastián

# 1. Introducción

Dada la dificultad de los problemas NP (y dada la misma dificultad que poseen estos para reducirse a problemas resolubles en tiempo polinomial) se ha acudido a técnicas que pretenden, sin bases que garanticen su buen funcionamiento, resolver a partir de aproximaciones aquellos problemas cuya solución no es obtenible más que probando todas las posibles combinaciones.

Estas técnicas suelen, en la mayor parte de los casos, conseguir soluciones a partir de simulaciones de fenómenos del mundo real: el comportamiento de animales, fenómenos físicos y sociales, entre otros. Estas técnicas son denominadas heurísticas.

El algoritmo del artillero innovador es una heurística que pretende encontrar soluciones simulando el comportamiento (a grandes rasgos, y abstrayendo en demasía el proceso) de un artillero que tiene como objetivo alcanzar los blancos que, en nuestro contexto, serán soluciones del problema.

En este caso, se pretenderá encontrar aproximaciones que resuelvan el problema del  $k$ -árbol generador de peso mínimo.

## 1.1. El problema

*... requerimos de un árbol de peso mínimo que abarque al menos  $k$  nodos en una gráfica ponderada. [3]*

Este problema fue formulado por Lozovanu y Zelikovsky, en 1933; además, en 1966, en [3] se hizo énfasis en una concepción construida a partir de puntos en el plano euclidiano. Es claro que el valor de  $k$  debe ser menor al número de nodos, de otra manera se reduce a un problema trivial que puede ser resoluble en tiempo polinomial por el algoritmo de Kruskal<sup>1</sup> y/o Prim<sup>2</sup>.

## 1.2. La heurística

El algoritmo del artillero innovador, como ya se ha mencionado, pretende simular el comportamiento de un artillero *innovador* que pretende acertar a ciertos blancos a base de una metodología de prueba y error que progresa modificando el ángulo de disparo.

---

<sup>1</sup>El algoritmo de Kruskal un bosque de peso mínimo de una gráfica ponderada no dirigida. Si la gráfica es conectada, encuentra un árbol generador de peso mínimo [4]. Complejidad:  $O(E \log E)$

<sup>2</sup>Algoritmo "greedy" que encuentra un árbol generador de peso mínimo es una gráfica ponderada no dirigida [5]. Complejidad:  $O(|V|^2)$

*La innovación del Algoritmo del Innovador radica en el hecho de que en cada paso del proceso iterativo, la solución previa es corregida a partir de factores multiplicativos especialmente seleccionados. [2]*

La innovación que describe es comparada con técnicas de modificación aditiva; entre estas heurísticas se encuentran:

- PSO (Eberhart and Kennedy, 1995):

$$x_l^{(i+1)} = x_l^{(i)} + v_l^{(i)}$$

- SA (Yang, 2010):

$$x^{(i+1)} = x^{(i)} + \varepsilon$$

- CS (Yang and Deb 2009):

$$x_l^{(i+1)} = x_l^{(i)} + \alpha \cdot \text{Lévy}(\lambda)$$

- GSA (Rashedi, Nezamabadi-pour, and Saryazdi, 2009):

$$x_l^{(i+1)} = x_l^{(i)} + v_l^{(i+1)}$$

Esto se justifica argumentando que se pretende reemplazar la dependencia intuitiva, con dependencia:<sup>3</sup>

$$x_l^{(i+1)} = x_l^{(i)} \cdot f_l(\xi)$$

### 1.2.1. Factores Multiplicativos

La selección de los factores multiplicativos (descritos como una fórmula aplicada a ángulos de corrección generados aleatoriamente) es crucial, se describen como funciones que deben cumplir:

1. Para valores de los ángulos de corrección  $\alpha$  y  $\beta$  cercanos a cero, los coeficientes de corrección deberían tener un valor cercano a uno.
2. Para  $\alpha > 0$ ,  $g(\alpha) > 1$
3. Para  $\alpha < 0$ ,  $g(\alpha) < 1$
4. Para ángulos generados aleatoriamente cercanos a  $\alpha_{máx}$  y  $-\alpha_{máx}$ , los coeficientes deberían ser claramente diferentes del valor 1 ( $g(\alpha) \gg 1, g(\alpha) \ll 1$ ), para asegurar una corrección lo suficientemente larga a la variable de decisión  $\delta$

Se concluye, entonces, que la función más adecuada, dadas las especificaciones necesarias, se trata de:

$$g(\alpha) = \csc(\alpha) = (\cos(\alpha))^{-1} \text{ para } \alpha > 0$$

$$g(\alpha) = \cos(\alpha) \text{ para } \alpha \leq 0$$

---

<sup>3</sup>Esto está, por lo tanto, en contra de la concepción de la solución como una búsqueda local; inclinándose más a algoritmos genéticos donde el resultado de soluciones previas está, en la mayor parte de los casos, totalmente alejada de las soluciones *padres*.

Así, los pasos de la ejecución del algoritmo se describen como sigue:

1. Seleccionar aleatoriamente los valores componentes del vector de decisión <sup>4</sup>

$$x_1^0 = [x_{11}^0, x_{12}^0, \dots, x_{1n}^0] \quad (\text{Vector inicial 1})$$

$$x_m^0 = [x_{m1}^0, x_{m2}^0, \dots, x_{mn}^0] \quad (\text{Vector inicial m})$$

2. Determinar el valor de la función objetiva para el vector inicial  $F_{obj}(x_1^0) \dots F_{obj}(x_m^0)$ ,  $F_{obj.best} = \min[F_{obj}(x_1^0) \dots F_{obj}(x_m^0)]$ .
3. Inicializar el contador de iteraciones a  $i = 1$ ; sustitución de  $x_1^i = x_1^0 \dots x_m^i = x_m^0$ .
4. Determinar el rango de ángulos de corrección  $\alpha_{máx}^i$  y  $\beta_{máx}^i$ . <sup>5</sup>
5. Ajustar los valores componentes del vector decisión con base en los factores de corrección como sigue:

Solución número 1: elegir ángulos de corrección  $\alpha_{11}^i \dots \alpha_{1n}^i$  y  $\beta_{11}^i \dots \beta_{1n}^i$  de acuerdo a la distribución de probabilidad definida <sup>6</sup>; corregir los componentes del vector de decisión:

$$x_{1,1}^{(i+1)} = x_{1,1}^{(i)} \cdot g(\alpha_{1,1}^i) \cdot g(\beta_{1,1}^i) \dots x_{1,n}^{(i+1)} = x_{1,n}^{(i)} \cdot g(\alpha_{1,n}^i) \cdot g(\beta_{1,n}^i)$$

Solución número m: elegir ángulos de corrección  $\alpha_{m1}^i \dots \alpha_{mn}^i$  y  $\beta_{m1}^i \dots \beta_{mn}^i$  de acuerdo a la distribución de probabilidad definida; corregir los componentes del vector de decisión:

$$x_{m,1}^{(i+1)} = x_{m,1}^{(i)} \cdot g(\alpha_{m,1}^i) \cdot g(\beta_{m,1}^i) \dots x_{m,n}^{(i+1)} = x_{m,n}^{(i)} \cdot g(\alpha_{m,n}^i) \cdot g(\beta_{m,n}^i)$$

6. Determinar el valor de la función objetivo para las soluciones  $I-m$ , es decir,

$$F_{obj}(x_1^{i+1}) \dots F_{obj}(x_m^{i+1})$$

Identificar solución  $q$  para la cual

$$F_{obj}(x_q^{i+1}) = \min[F_{obj}(x_1^{i+1}) \dots F_{obj}(x_m^{i+1})]$$

7. Si, para la solución  $q$ ,  $F_{obj}(x_q^{i+1}) < F_{obj.best}$  se cumple, entonces se sustituirá  $F_{obj.best} = F_{obj}(x_q^{i+1})$  y  $x_{best} = x_q^{i+1}$ .
8. Verificar el criterio de finalización del algoritmo ( $i + 1 = i_{máx}$ ); continuar al final (Paso 9) o regresar al Paso 4 ( $i = i + 1$ ).
9. El fin del algoritmo; terminamos con los valores  $x_{best}$  y  $F_{obj.best}$ .

<sup>4</sup>La heurística pretende resolver problemas modelados por una función lineal, la adaptación al problema se describe en las secciones de Diseño e Implementación

<sup>5</sup>La elección de dos factores multiplicativos no cuenta con justificación, sin embargo, se describe como una buena elección a partir de la experimentación realizada.

<sup>6</sup>Distribución normal, con un valor promedio de cero y una desviación estándar que cumple la dependencia  $3\sigma = \alpha_{máx} = \beta_{máx}$ , asumiendo que, por ejemplo,  $\alpha_{máx} = \beta_{máx} = (\pi/2)$ .

## 2. Diseño

El diseño siguió una estructura orientada a objetos, dentro de los límites de la abstracción proporcionada por C (conceptos como herencia o polimorfismo son inexistentes sin una implementación previa).

### 2.1. Analizador Sintáctico

Este objeto se encarga de proveer los datos de entrada a partir de un archivo con el formato

$$\begin{array}{c} x_1, y_1 \\ x_2, y_2 \\ \vdots \\ x_n, y_n \end{array}$$

*Donde  $x$  y  $y$  son las coordenadas  $x$  y  $y$  de los puntos.*

Este hará dos procesos lineales de lectura, acción esencial para calcular el número de puntos antes de asignar el espacio en el heap para la creación del arreglo de puntos. Puede pensarse que no ha sido la mejor decisión, haciendo notar que la primera iteración sería innecesaria si se usaran listas; pero consideré más importante el acceso en tiempo constante que el aumento en uno de un proceso lineal (que, dado que son procesos constantes lineales, sigue siendo, en general, un proceso lineal).

#### 2.1.1. Puntos

Los puntos se abstrayeron en una clase `point` que, además, de proveer las coordenadas  $x$  y  $y$  en una sola estructura, proporciona comportamiento como el cálculo de la distancia euclidiana o el punto medio. Se trata de la estructura esencial del programa.

### 2.2. El problema

El problema del  $k$  - árbol generador de peso mínimo se abstraigo en una clase `kmst`, cuya función es recibir los puntos obtenidos por el analizador sintáctico, el valor de  $k$  y la semilla de una ejecución.

Su propósito radica en la organización de los datos de entrada como una sola entidad que abstrae el problema y sirve como entrada a la heurística.

### 2.2.1. El árbol

La clase `tree` abstrae la idea de la creación de un árbol a partir de un conjunto de puntos, donde la trayectoria se crea a partir del algoritmo de Kruskal.

## 2.3. Algoritmo del Artillero Innovador

Esta clase reúne los parámetros necesarios para la ejecución del algoritmo del artillero innovador (los ángulos de corrección y el parámetro  $\varepsilon$ , encargado de fungir como cota que determina el final del algoritmo), además, recibe el problema `kmst` como parámetro; la ejecución del algoritmo se reduce a solo una invocación de una función.

### 2.3.1. Círculo

La clase `circle` abstrae un círculo en el plano cartesiano que proporciona comportamiento como la devolución de los puntos que yacen dentro del mismo o la creación de su cuadrado circunscrito.

### 2.3.2. Cuadrado

La clase `square` proporciona comportamiento para encontrar la cantidad de puntos contenidos en él, así como la lista de ellos o la computación de las celdas en las que se puede dividir el mismo.

## 3. Implementación

La orientación a objetos se ha simulado a partir de la asociación de funciones, en conjunto con una estructura que es parámetro de la primera; esta abstracción permite, al menos de manera primitiva, simular una orientación a objetos básica que facilita el proceso creativo y de depuración del programa.

### 3.1. Generador de números pseudoaleatorios

Dado que [1] describe a la función `rand_r` como un *generador de números pseudoaleatorios débil*, ya que la semilla que recibe es del tipo `unsigned int` (esta proporciona una cantidad reducida de simulaciones diferentes), he decidido usar `drand48_r` y `lrand48_r`, los cuales trabajan una semilla de tipo `long int` (que, por lo tanto, induce una mayor cantidad de estados posibles). Ambas se encuentran disponibles dentro de la biblioteca `stdlib.h`.

### 3.2. Adaptación

Como se mencionó en la descripción de la heurística, el algoritmo del artillero innovador fue diseñado en términos de la optimización de un problema que puede reducirse a la minimización de una función lineal (función objetivo) a partir de la modificación *aleatoria*<sup>1</sup> de las variables de las que depende la función.

Inicialmente, la concebí como la minimización de la suma de las aristas contenidas en el árbol (una función lineal intuitiva, pero errónea), pero esta carecía de una variable libre que pudiera modificar con los factores multiplicativos que permitiera, además, llegar a una mejoría sin modificar la intención original de los autores; no era posible usar las aristas (o coordenadas de los vértices) como valor modificado ya que los puntos resultantes podrían no encontrarse dentro de la gráfica original; considero, incluso, que, dejando a un lado este problema, resolviéndolo, digamos, con una función que asegure que los puntos resultantes se tratasen de solo aquellos contenidos en la gráfica, la intención de los autores (y, por lo tanto, la experimentación que se llevó a cabo para llegar a tales conclusiones) quedaría apartada. Es por ello que decidí implementar un algoritmo que involucrara las ideas expuestas en [3].

En este artículo se se abstrae la resolución del problema como un conjunto contenido en el plano cartesiano. Presenta un algoritmo que pretende aproximar soluciones formando figuras geométricas que abarquen al menos una cantidad  $k$  de puntos, y, con

---

<sup>1</sup>No se trata de un proceso totalmente aleatorio, técnicamente, el hecho de simular un fenómeno actúa como guía y permite un poco más de control y teoría en su ejecución y diseño.

ellos, formar un árbol generador de peso mínimo. Esta idea funciona por el simple hecho de tratar de juntar en un árbol aquellas zonas más densas de la gráfica, es decir, con menor distancia entre sus puntos.

Es aquí donde me surgió la idea de definir la variable que reduciría mi función objetivo como el diámetro de un círculo que crecería o disminuiría según un proceso iterativo ya definido. Entonces el diámetro sería la variable que sería expuesta a la modificación de los dos factores multiplicativos descritos por la heurística. Pero, eventualmente, me daría cuenta de un problema primordial: la búsqueda local había sido ignorada; el concepto de vecinos no tiene tanto sentido al concebir como unidad fundamental y de cambio las medidas de una figura geométrica; la heurística tornaba más en un algoritmo de búsqueda global.

### 3.3. Problemas

C tiene como principio fundamental la libertad de operación del programador; solo el compilador "guía" de cierta manera (sintácticamente, al menos), pero todas las vicisitudes son cargadas a la experiencia y creatividad del programador.

Solo considero relevante destacar la problemática que involucra tres alternativas, ninguna de las cuales, me parece clara de seguir.

#### 3.3.1. Encapsulamiento

El encapsulamiento es un concepto clave en la orientación a objetos. En C, puede simularse si no se incluye la definición de la estructura dentro del encabezado de la clase. De esta manera, la clase tendrá que proporcionar el comportamiento que permita crear objetos instancia de la misma, ya que el operador `sizeof` (usado al crear dinámicamente un objeto) no posee la información del tamaño de la estructura. En caso de necesitar estructuras compuestas con estos objetos (como arreglos), será necesario crear apuntadores a apuntadores de esos objetos, de esta manera se podrá guardar un objeto de esa clase invocando su función constructora, además, será posible recorrer el arreglo ya que no se debe conocer el tamaño de la estructura para hacerlo, sino solo el de un apuntador (ya estandarizado en el lenguaje).

Puede parecer un poco rebuscada esta solución ya que implica bastantes niveles de referencias; pero seguimos cumpliendo el encapsulamiento de las clases.

Por otro lado, se pueden usar arreglos convencionales (como un apuntador al tipo de esa clase), pero se deberá proporcionar comportamiento para recorrerlo, modificarlo y acceder a sus datos. Lo que resulta en una tarea muy tediosa, dadas las funciones requeridas.

Por último, tratándose tal vez de la alternativa menos tediosa, se encuentra la inclusión de la definición de la estructura dentro del encabezado de la clase. Esta técnica no sigue los patrones de orientación a objetos, ya que ahora será posible crear, con la función `malloc` objetos de la clase, sin necesidad de contar con una función constructora; los atributos de la clase, ya no serán privados; en general, se perderá la modularización y el encapsulamiento del paradigma orientado a objetos.



## 4. Experimentación

Dada la técnica de búsqueda de la heurística (evitando por completo las búsquedas locales), la prueba de su efectividad no ha sido del todo certera, esto yace de la naturaleza de los ejemplos artificiales que, la mayor parte del tiempo, suelen agrupar el árbol generador de peso mínimo en un lugar apartado de los demás puntos contenidos. Dado que mi implementación genera círculos y que estos pueden abarcar enteramente esta sección excluida, algunas veces se daba el caso en el que el círculo abarcaba completa y únicamente los puntos que poseían el árbol de peso mínimo.

Sin embargo, esos solo han sido pocos ejemplos que no evalúna en gran medida su desempepeño.

## 5. Conclusiones

Considero que esta implementación no se ha acercado a la concepción que poseían los autores del algoritmo del artillero innovador. No posee varias características fundamentales como el uso de semillas o la búsqueda local.

Definitivamente, el tiempo invertido no ha sido el adecuado para integrar, de manera correcta, el problema y la heurística.

# Bibliografía

- [1] Linux man page. rand\_r - linux man page. [https://linux.die.net/man/3/rand\\_r](https://linux.die.net/man/3/rand_r).
- [2] Pawel Pijarski and Piotr Kacejko. A new metaheuristic method: the algorithm of the innovative gunner (aig). Engineering Optimization, 2019, 2018.
- [3] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spannig trees short or small. SIAM Journal on Discrete Mathematics, 1996.
- [4] Wikipedia. Kruskal's algorithm. [https://en.wikipedia.org/wiki/Kruskal %27s\\_algorithm](https://en.wikipedia.org/wiki/Kruskal_%27s_algorithm).
- [5] Wikipedia. Prim's algorithm. [https://en.wikipedia.org/wiki/Prim %27s\\_algorithm](https://en.wikipedia.org/wiki/Prim_%27s_algorithm).