Seminario de Ciencias de la Computación B

Heurísticas de Optimización Combinatoria

Problema del k-árbol generador de peso mínimo con Optimización de Lobos Grises

Profesor: Canek Peláez Valdés
Autor: Xin Wen Zhang Liu

Martes, 25 de Abril, 2023.

El problema del k árbol generador de peso mínimo

La entrada de este problema es una gráfica completa no dirigida, sobre la cual se debe encotrar una subconjutno de k vértices los cuáles generen un árbol de peso mínimo en la gráfica, entonces el resultado es un conjunto de k vértices y $k-1$ aristas dentro de la gráfica. Este problema es NP-duro, con una complejidad polinomial.

Optimización del Lobo Gris / Grey Wolf Optimization

Esta meta-heurística fue inspirar por el comportamiento depredatorio de los lobos grises, y planteada Mirjalili [2]. Propuesta como una nueva alternativa a la optimización por enjambre de partículas, este algoritmo simula el comportamiento depredatorio de los lobos gries , y su comportamiento jerárquico dentro de manadas.

Cada enjambre de lobos sigue una jerarquía social, como la que sigue

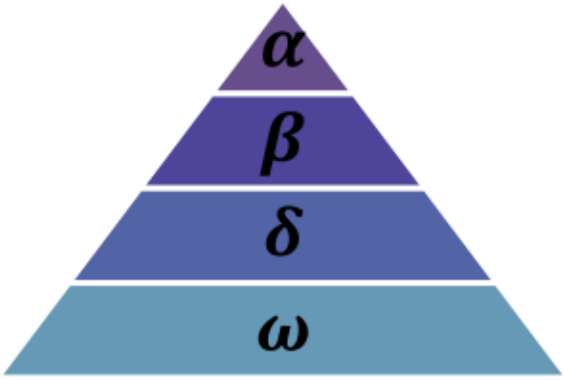

Fig. 1. Hierarchy of grey wolf (dominance decreases from top down).

donde Alpha, Beta y Delta son las 3 mejores soluciones respectivamente, y todas las demás son asignadas a los Omegas.

El modelo matemático que representa a este algoritmo contiene las siguientes estapas

1. Acorralar a la presa
2. Caza
3. Atacar a la presa

Los lobos acorrarlan y rodean a su presa cuando cazan, para modelar esto matemásticamente usamos la sigueinte ecuación

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)|$$

$$\vec{X}(t+1) = \vec{X}_p(t) = \vec{A} \cdot \vec{D}$$

donde t respresenta la iteración actual y $\vec{A}, \vec{C}$ son vectores coeficiente. $\vec{X}_p$ es la posición de la presa, de la cual no sabemos su paradero exacto, y es dependiente al función de costo que se quiera usar. $\vec{X}$ indica la posición del lobo en la iteración anterior.

Los vectores $\vec{A}$ y $\vec{C}$ son calculados de la siguente manera.

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a}$$

$$\vec{C} = 2 \cdot \vec{r}_2$$

donde $\vec{a}$ es un componente lineal que decrese de 2 a 0, a lo largo de las iteraciones , y r_1, r_2 son números aleatorios.

Los lobos son capaces de reconocer la unibación de su presa y rodearla. La caza es generalmente guiada por el alpha. Sin embargo en un espacio de búsqueda abstracto es complicado saaber la ubicacón de la presa (mínimo local). Para modelar este comportamiento de caza, asumimos que el alpha, beta y delta tiene el mayor conocimiento sobre de la ubicación de la presa. Por esot modificamos a los demás agentes a incluyendo a los omega a actualizar su ubicación en base a la de las 3 mejores soluciones. Para esto se utilizan las siguientes fórmulas.

$$\vec{D}_d = |\vec{C}_1 \cdot \vec{X}_d - \vec{X}|$$

$$\vec{X}_i = \vec{X}_d - \vec{A}_i \cdot \vec{D}_d$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}$$

Diseño

La gran parte de la heurística se encuentra dentro del struct, GWO el cual se encarga de generar conjuntos de vértices en cada iteración de la evolución de los lobos, para generar los árboles generadores de peso mínimo asociado. Para esto se implementó el algoritmo de Kruskal, y sus funciones asociadas dentro de del struct Tree, y el cual igual regresa el pero asociado al árbol el cuál es la evaluación de f̈itnessde cada lobo.

La ejecución del algoritmo está dividida en dos funciones, `run_gwo`, el cual se encarga de inicializar a la manada, y asignarles soluciones iniciales a cada uno. En cada iteración esta llama a `evolve` el cual se encarga de modificar cada agente hacia un vecino nuevo.

Dentro del crate Tree, igual se encuentran los structs Edge y Vertex, que representan las aristas de la gráfica y los vértices respectivamente. Cada Edge estás compuesto de dos vértices que delimitan cada extremo de la arista, además de el peso asociado a esta. Los vértices guardan el valor de las coordenas en el palno xy, además de un identificador representado por un entero.

El struct Reader es el encargado, leer los puntos de un archivo de texto y crear Vértices que guarden las coordenadas.

Implementación

El aplicar la heurística al problema del k árbol generador de peso mínimo fue el proceso más tardado. La mayoría del tiempo invertido en este proyecto fue hecho en la implementación de la heurística.

El primer paso fue investigar y comprender a fondo la idea general de ésta para despúes poder aplicarlo a nuestro problema. La modificación del modelo para que se acoplara a lo necesario para generar soluciones fue la parte más

Cada lobo dentro de la manada tiene asociado una solución del problema, y en cada iteración el conjunto de vértice asociada a la solución es modificada aleatoriamente, basado en la posición de los lobos Alpha, Beta y Delta con las mejores soluciones. Cada lobo Omega actualiza su nueva posición en base a estos, donde la posición es la que determina el nuevo vértice que va a ser agregado para obtener el vecino.

La implementación de kruskal siguió el algoritmo de union find [3] con path compression para una menor complejidad. Kruskal trabaja sobre una lista de aristas ordenaddas por su peso, y agarra en orden las aristas que no generen un ciclo dentro del árbol, esto hasta tener aritas que conecten a todos los vértices de la grfica. Uno de los principales problemas de esta parte de la implementación fue el refactorizar el código para que aceptara

valores flotantes. La primera implementación de este algoritmo fue usando HashMpas, lo que fue un error, ya que las coordenadas y pesos de tipo flotante, hacen que muchas de las operaciones sean necesariamente implementadas a mano.

Muchos de los problemas en rust surgen del tiempo de vida de vairables y de la posesión de estas. Esto lleva a que un código más simple requiera de muchas más copias de varibales para no tener problemas con lo mencionado anteriormente, esto llega a programas menos optimizados. [1]

Experimentación y resultados

Las soluciones generadas por la heurística tardan en converger y mejorar los anteriores mientras más baja la evaluación del resutado anterior, teniendo un estacamiento al estar generando los vecinos aleatorios en cada iteración.

La falta de experimentación llevó a un sistema menos predecible, y propenso a mínimos locales. Por la manera en que lso lobos toman nuevas posiciones depués de cada iteració hace que el rango de aleatoridad para ecoger nuevo vértices sea pequeño, lo que lleva a que los agentes converjan a un mínimo local y no haya una mayor exploración.

El promedio de la evaluación de los resultados encontrados por la heurística sobre los puntos pasados por correo caen en el rango de $50 - 70$.

El haber implementado la graficación de las soluciones ayudó mucho a la depuración del programa, pudiendo ver visualmente las soluciones generadas.

Después de observar varias gráficas generadas a lo largo de la ejecución, se notó que las gráficas de las primeras iteraciones siempre seguían una forma más esparcida en el espacio de búsqueda, sin embargo al continuar con la ejecucíón y al ir mejorando las soluciones, las gráficas terminaban teniendo una forma más compacta a las iniciales. Sin embargo, por la aleatoridad de la heurística no se podía asegurar que a través de las ejecuciones cada soluciones buscara su forma más compacta en un espacio de búsqueda local, lo que llevaba a no poder encontrar un mínimo local.

Para solucionar esto se implementó una manera diferente de seleccionar los vecinos en cada iteración. Primero se busca el centro geométrico de la gráafica y a partir de esto se compara con todos los puntos en la solucón anterior y se encuentra al más alejado, el cual va a ser el que sea modificado para encontrar una solución vecina. Después de esto se busca un vértice al azar el cual tenga una distancia al centro geométrico menor del vértice que se eliminó. De esta manera cada agente llega exaustivamente a un mínimo local.

Con esto se logró que las soluciones encontradas bajaran de un peso de $50 - 70$ para el conjunto original de puntos, a poder encontrar la solución óptima de 30. Además de que el tiempo de ejercución disminuyó un gran porcentage.

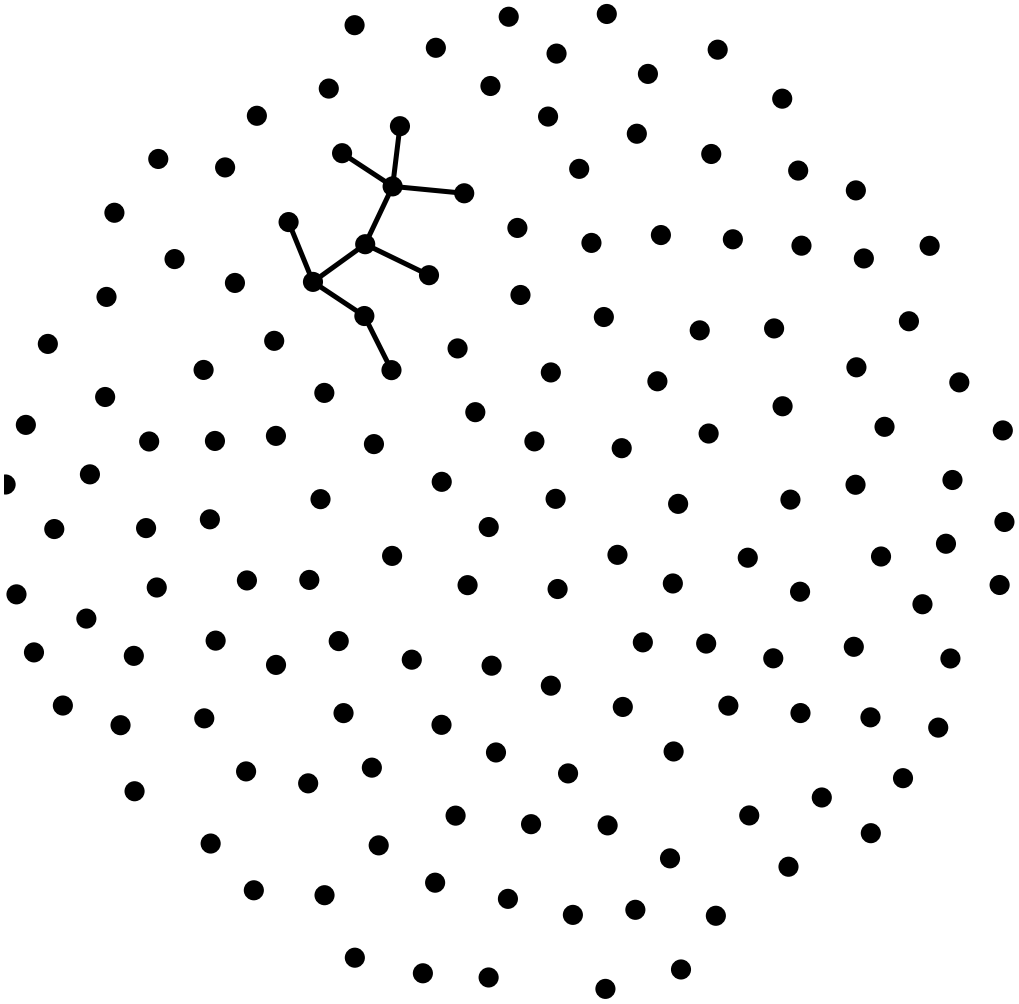

Figure 1: Grafica de solucón con los puntos de t3.txt, evaluación 57.64

Conclusiones

La planeación en este caso podría haberse visto avanzada con la implementación del programa, ya que crea dudas reales que aplican al código del programa y nos hace preguntarnos de la mejor manera de implementar lo requerido .
La experimentación no sólo conlleva a encontrar mejores resutlados sino igual a mejorar el código para que la calidad de los resultados sea cada vez más alta. Es por esto que
Por la falta de tiempo la optimización del sistema fue escaza, por lo que el experimentar fue más tardado de lo normal.
Aunque el programa ecuentre soluciones favorables, sigue teniendo una falta de exploración y cada agente cae en su mínimo local rápidamente. Por lo que las soluciones dependen de las soluciones aleatorias iniciales.
El haber modificado la manera de escoger aristas para nuevos vecinos, ayudó exponencialmente al tiempo de jecución y la calidad de soluciones encontradas. El filtrar condiciones para generar casos aleatorios enfonca al

algoritmo a tener mejores resultados.

Bibliography

[1] Rust Lang. The rust programming language. *doc.rust-lang.org*, 2017.

[2] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[3] Abhishek Rajput. Union find algorithm. *Geeks for Geeks*, 2022.