

Algorithmic Design for the procedural optimization of the Zero Line Method in the Geometric Road Design

Diego Felipe Beltrán Cardona¹; Lorena Yinet Ortégón Cáceres²; Miguel Ángel Olivares Tenorio³

^{1,3} Universidad Militar Nueva Granada, Colombia, diego.beltran@unimilitar.edu.co, miguel.olivares@unimilitar.edu.co

² Universidad Militar Nueva Granada, Colombia, lorena.ortegon@unimilitar.edu.co

Abstract— *Algorithmic design and theories of spatial organization in video games find a common ground to solve one of the problems in the pre-feasibility stage of geometric road design: connecting two points on a terrain thru the most optimal and efficient route. This solution is a conjunction of multiple fields of knowledge, which come together to develop a tool that materializes these theories in a practical case of study. From this development, a first prototype of algorithmic design emerges, presenting the Preliminary Designs of the Zero-Line Method across nine routes that are analyzed and compared to identify the characteristics of those that approach the destination point thru algorithmic design. Future work aims to improve the approximation of these Preliminary Designs in order to continue with the subsequent pre-feasibility stages that will complete the solution to the optimization problem in the manual procedures of the Zero-Line Method.*

Keywords-- *Geometric Road Design, Terrain, Algorithmic Design, Zero Line, Optimization.*

Diseño algorítmico para la optimización procedimental del método de la Línea de Ceros en el Diseño Geométrico de Vías

Diego Felipe Beltrán Cardona¹; Lorena Yinet Ortega Cáceres²; Miguel Ángel Olivares Tenorio³

^{1,3} Universidad Militar Nueva Granada, Colombia, diego.beltran@unimilitar.edu.co, miguel.olivares@unimilitar.edu.co

² Universidad Militar Nueva Granada, Colombia, lorena.ortegon@unimilitar.edu.co

Resumen– El diseño algorítmico y las teorías de organización del espacio en los videojuegos encuentran un terreno común para resolver uno de los problemas en la etapa de pre-factibilidad del diseño geométrico de vías: conectar dos puntos sobre un terreno a través de la ruta más óptima y eficiente. Esta solución es una conjunción de múltiples campos del conocimiento, que se unen para desarrollar una herramienta que materializa tales teorías en un estudio de caso práctico. De este desarrollo, surge un primer prototipo de diseño algorítmico, que presenta el Trazado Preliminar basado en el Método de la Línea de Ceros bajo nueve rutas que son analizadas y comparadas, para identificar las características de aquellas que se acercan al punto destino a través del diseño algorítmico. El trabajo futuro tiene como objetivo mejorar la aproximación de estos Trazados Preliminares con el fin de continuar con las siguientes etapas de la pre-factibilidad que completen la solución al problema de optimización en los procedimientos manuales del Método de la Línea de Ceros.

Palabras clave– Diseño Geométrico de vías, Terreno, Diseño Algorítmico, Línea de Ceros, Optimización.

I. INTRODUCCIÓN

En el Diseño Geométrico de Vías el procedimiento de la Línea de Ceros es el primer trazo técnico que valida la conectividad entre el punto inicial con el punto final, manteniendo una pendiente constante y uniforme. Esta Línea de Ceros permite definir la ruta más eficiente a partir de los cálculos de pendientes, longitud del trazado, las condiciones topográficas y costo del proyecto. La línea de Ceros asegura una vía económicamente óptima, minimizando la cantidad de corte y terraplén. El procedimiento tiene ejecuciones manuales y semiautomatizadas basadas en herramientas digitales que generan variaciones en los resultados y altos tiempos en la ejecución. Estas dificultades en el procedimiento evidencian la necesidad de estudiar soluciones algorítmicas que permitan optimizar y aproximar el procedimiento a resultados más eficientes y precisos.

Un proyecto como “Herramientas avanzadas de análisis de datos de aplicación en ingeniería civil: Machine Learning using Python” [9] es una referencia que explica la importancia y las ventajas de abordar problemas de optimización, mediante modelos parametrizados e iterativos de aprendizaje automático en áreas de la ingeniería civil que impliquen Big Data, por medio del uso de librerías especializadas en análisis, estructuración y manipulación de datos. El desarrollo del software para Diseño Geométrico de

Vías llamado “VIAS” [8] también es un referente de valor en la automatización de cálculos topográficos, que a través de lenguajes de programación como Auto Lisp utiliza herramientas digitales accesibles para mejorar los flujos de trabajo.

Este artículo científico presenta los resultados del desarrollo de un primer prototipo de herramienta digital, cuyo objetivo es optimizar el procedimiento y precisar los resultados del trazado de la Línea de Ceros en la fase de prefactibilidad del Diseño Geométrico de Vías. Bajo este contexto el presente artículo busca abordar la siguiente pregunta problema: ¿Qué diseño algorítmico puede identificarse como el más adecuado para optimizar el procedimiento y aproximar los resultados de la Línea de Ceros en el Diseño Geométrico de Vías?

II. DESCRIPCIÓN DEL PROCEDIMIENTO MANUAL PARA LA LÍNEA CEROS

El diseño de una ruta se desarrolla en etapas y cada etapa evalúa gradualmente la factibilidad del proyecto. En la primera fase de prefactibilidad se identifican los posibles corredores de ruta y se evalúa la viabilidad económica de los mismos, para ello se requiere como insumo esencial la topografía, la cual permite el análisis y la caracterización del terreno utilizado. El terreno inicialmente se visualiza en curvas de nivel representadas como líneas de contorno que indican las elevaciones del terreno y luego se procesan para convertirse en una nube de puntos, también conocido como modelo digital del terreno, manteniendo la representación de las curvas de nivel en puntos COGO (Coordinate Geometry). Cada curva de nivel está separada por una Equidistancia que es la diferencia de altitud entre dos curvas consecutivas. La Línea de Ceros es un trazado segmentado que pasa por las curvas de nivel y se define por la conexión de puntos específicos que marcan la ruta entre un punto de inicio y un punto final manteniendo la pendiente constante y uniforme. La Línea de Ceros es fundamental ya que esencialmente es el “esqueleto” por donde se apoyará el diseño definitivo y según la referencia [6] de Cárdenas es una línea que, al ir a ras del terreno natural, sigue la forma de éste, convirtiéndose en una línea de mínimo movimiento de tierra. Por lo tanto, cualquier eje vial de diseño que trate de seguirla lo más cerca posible, será un eje económico, desde este punto de vista.

Existen metodologías para el trazado de la línea de ceros que combinan diferentes criterios técnicos, económicos y específicos del terreno. La metodología utilizada en este caso de estudio se basa en la Pendiente Constante Uniforme, que como su nombre indica, mantiene un trazado con pendientes constantes de principio a fin. Para este procedimiento, se implementaron técnicas de trazado manual las cuales se dividen en cuatro procedimientos ejemplificados a continuación en los pasos de las Figura 1 (Fig. 1): El primer procedimiento, como se muestra en el paso 1, consiste en establecer los puntos de inicio y fin de la ruta, sobre las curvas de nivel. Luego se calcula la Pendiente Aproximada del terreno “ $Pendiente_0$ ” de la ecuación (1), la cual es la diferencia de las elevaciones “ Δz ” entre los puntos de inicio y fin, dividida por la distancia horizontal entre el Punto inicial y el Punto final llamada Longitud Aproximada “ L_{aprox_0} ”.

$$Pendiente_0 = \frac{\Delta z}{L_{aprox_0}} \quad (1)$$

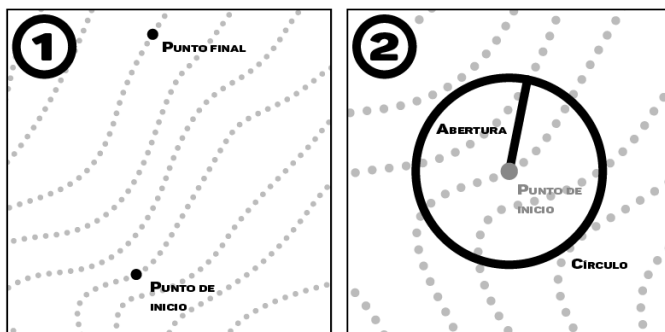


Fig. 1 Representación del paso uno y dos del procedimiento manual de la Línea de Ceros.

El segundo procedimiento convierte el porcentaje de la “ $Pendiente_0$ ” en una distancia horizontal, calculando la “ $Abertura_0$ ” con la ecuación (2) que se define como la división entre la Equidistancia “ E ” y la Pendiente Aproximada “ $Pendiente_0$ ”. El resultado de la “ $Abertura_0$ ” se utiliza para establecer la abertura de compás, representada en el radio de un círculo centrado en el Punto de inicio.

$$Abertura_0 = \frac{E}{Pendiente_0} \quad (2)$$

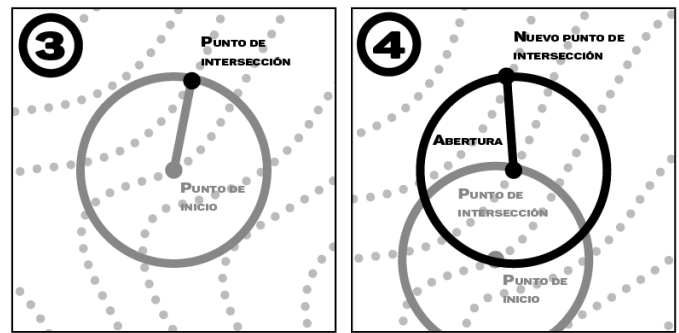


Fig. 2 Representación del paso tres y cuatro del procedimiento manual de la Línea de Ceros.

El tercer procedimiento, como se muestra en el paso 3 de la Figura 2 (Fig. 2), ubica el Punto de Intersección entre el círculo y la línea de contorno contigua a la que tiene el punto inicial y que esté en dirección hacia el Punto final, de esto resulta el siguiente punto en la trayectoria de la Línea de Ceros. Luego, como se muestra en el paso 4, se repite el segundo procedimiento con la abertura ya calculada, tomando un nuevo centro del círculo como el Punto de Intersección anterior y así sucesivamente hasta tener una Ruta Segmentada como se muestra en el paso 5 de la Figura 3 (Fig. 3).

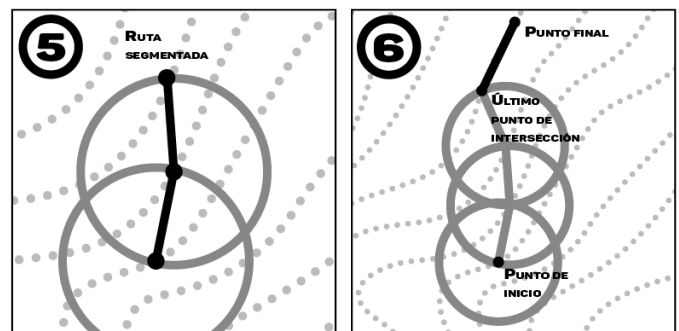


Fig. 3 Representación del paso cinco y seis del procedimiento manual de la Línea de Ceros.

Esta ruta tiende a alejarse del punto final debido a que el primer cálculo de la Línea de Ceros se realiza con una Pendiente y Longitud aproximadas. El cuarto procedimiento calcula la ruta de la Línea de Ceros con la mayor precisión, a través de un procedimiento iterativo donde se espera que la abertura converja en un valor específico, reduciendo entonces la diferencia entre estos dos puntos, y para resolverlo se suma la distancia entre el último Punto de Intersección y el Punto final de la ruta a la Distancia Longitudinal total, como se muestra en el paso 6 de la Figura 3. Este procedimiento debe probarse con diversos puntos de intersección, Pendientes y Longitudes Aproximadas tantas veces como sea necesario hasta lograr la aproximación más cercana entre el Último Punto de Intersección y el Punto final de la ruta

III. DESARROLLO METODOLÓGICO PARA LA OPTIMIZACIÓN PROCEDIMENTAL DE LA LÍNEA DE CEROS

A. Teorías de algoritmos de ruta y organización del espacio en videojuegos

En el presente desarrollo, se define el paradigma para la identificación de patrones a través del pensamiento algorítmico del referente [2], el cual utiliza las instrucciones como guía del descubrimiento en la optimización procedimental. El tratamiento de estos procedimientos a través de algoritmos de ruta está fundamentado por las teorías de grafos referenciados de Restrepo y Sánchez [1] y la organización del espacio en la teoría de videojuegos de Jesse Schell del referente [3], que abordan el problema de encontrar los caminos más cortos en las condiciones de terreno dadas. Según la referencia de Knuth [2], un conjunto finito y definido y efectivo de instrucciones que transforman datos de entrada en resultados de salida es lo que se entiende por un algoritmo. Siguiendo estas teorías y definiciones, este proyecto de investigación se apropia de las teorías de grafos y la organización espacial en la teoría de videojuegos para diseñar algoritmos específicos y lograr la optimización de los procedimientos en la Línea de Ceros.

Según la referencia [10], Un grafo es una representación matemática dada por nodos o vértices de información relacionados por aristas. Estas relaciones pueden representar redes de transporte o posibles rutas para llegar de un punto a otro. En estas relaciones la información tiene direcciones de desplazamiento, si tienen una única dirección es un grafo de tipo no orientado, de lo contrario es un grafo orientado, siendo para las rutas de este estudio los grafos dirigidos como la mejor opción, con el fin de encontrar, utilizando diferentes algoritmos, la mínima cantidad de vértices intermedios entre los nodos.

Los tres algoritmos de interés utilizados para encontrar caminos cortos que han sido estudiados en la teoría de grafos y los videojuegos son:

1) *Dijkstra*: Es un algoritmo conocido en la teoría de grafos que utiliza el criterio de peso o distancia de un nodo a otro; según la referencia [11] este algoritmo fue planteado en 1959 por E. W. Dijkstra, el cual explica el método óptimo para obtener la menor longitud entre dos nodos. Este algoritmo evalúa los posibles caminos a través de vecinos cercanos y luego compara la suma de las distancias entre puntos para determinar cuál es el camino más corto, como se muestra en el diagrama de flujo de la Figura 4 (Fig. 4), para cada vecino se calcula un nuevo costo. Su ecuación resultante se basa en un criterio de selección de puntos por descubrir “ $u \in Abierta$ ” donde ‘ $g(u)$ ’ es el costo referido a la distancia que existe entre un punto a otro, el cual se acumula en la suma de las distancias entre los puntos de la ruta para llegar a ‘ u_{min} ’ que es el punto siguiente resultante de la ecuación (3).

$$u_{min} = \arg \min_{u \in Abierta} g(u) \quad (3)$$

Con el algoritmo Dijkstra se logra integrar dos criterios del Diseño Geométrico de Vías para la selección de puntos de la ruta: el primer criterio es validar la curva a la que pertenece el candidato y determinar si la diferencia de elevación es equivalente o menor a la Equidistancia; y el segundo, en la distancia mínima para considerar un punto como candidato, este valor de distancia es igual a la Abertura inicial.

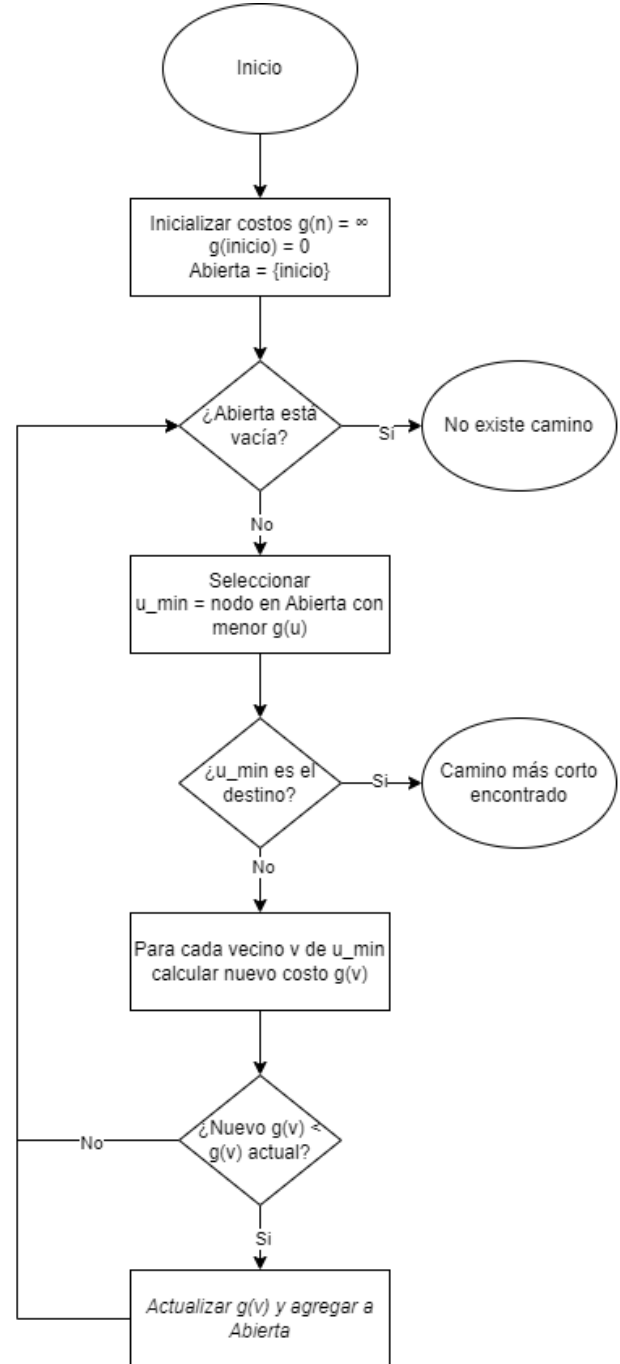


Fig. 4 Diagrama de flujo del pseudocódigo: algoritmo Dijkstra.

2) *A** o *A star*: Según la referencia [12], Este fue un algoritmo propuesto en 1968 por Peter Hart, Nils Nilsson y Bertram Raphael, el cual se rige por heurísticas de distancias al punto final. Una de sus aplicaciones es NextBot, un actor que persigue al jugador por el mapa teniendo en cuenta la geografía. *A** trabaja con la heurística " $h(n)$ ", que es la distancia entre el punto actual " n " y el final de la ruta " fin ", siendo " $d(n, fin)$ " una función que mide la distancia (4). La heurística " $g(n)$ " es la distancia desde el inicio hasta el punto actual de la ruta. Con la suma " $f(n)$ " que representa la sumatoria para cada camino posible (5), se determina el siguiente paso de la ruta calculando con la ecuación " $actual$ " (6) cuál de los caminos posibles tiene la menor distancia, entre la recorrida desde el punto de inicio al punto actual, o la distancia desde el punto actual al punto final. Como se muestra en el diagrama de flujo de la Figura 5 (Fig. 5), se selecciona la menor con la sumatoria de " $f(n)$ ", luego valida si el punto es el final y vuelve a revisar los posibles caminos calculando " $g(n)$ ", " $h(n)$ " y " $f(n)$ ", que en el caso de una nueva sumatoria menor a la actual, se actualiza y la agrega a la ruta.

$$h(n) = d(n, fin) \quad (4)$$

$$f(n) = g(n) + h(n) \quad (5)$$

$$actual = \arg \min_{n \in Abierta} [f(n)] \quad (6)$$

Con el algoritmo *A**, se pueden integrar valores máximos como la diferencia de elevaciones y mínimos como la Abertura para calcular el costo de un candidato, condicionando a que no se tomará este valor como su costo, sino como el error (la diferencia de la abertura y la distancia entre el punto anterior y el candidato). Luego se determina la heurística de este candidato, siendo la sumatoria de ambos valores el criterio de elección del siguiente punto. En el caso de no tener una pendiente constante entre vértices o que no llegar al punto destino, se puede aplicar una iteración para determinar un nuevo valor de Abertura, incrementando o disminuyendo este valor según sea el caso.

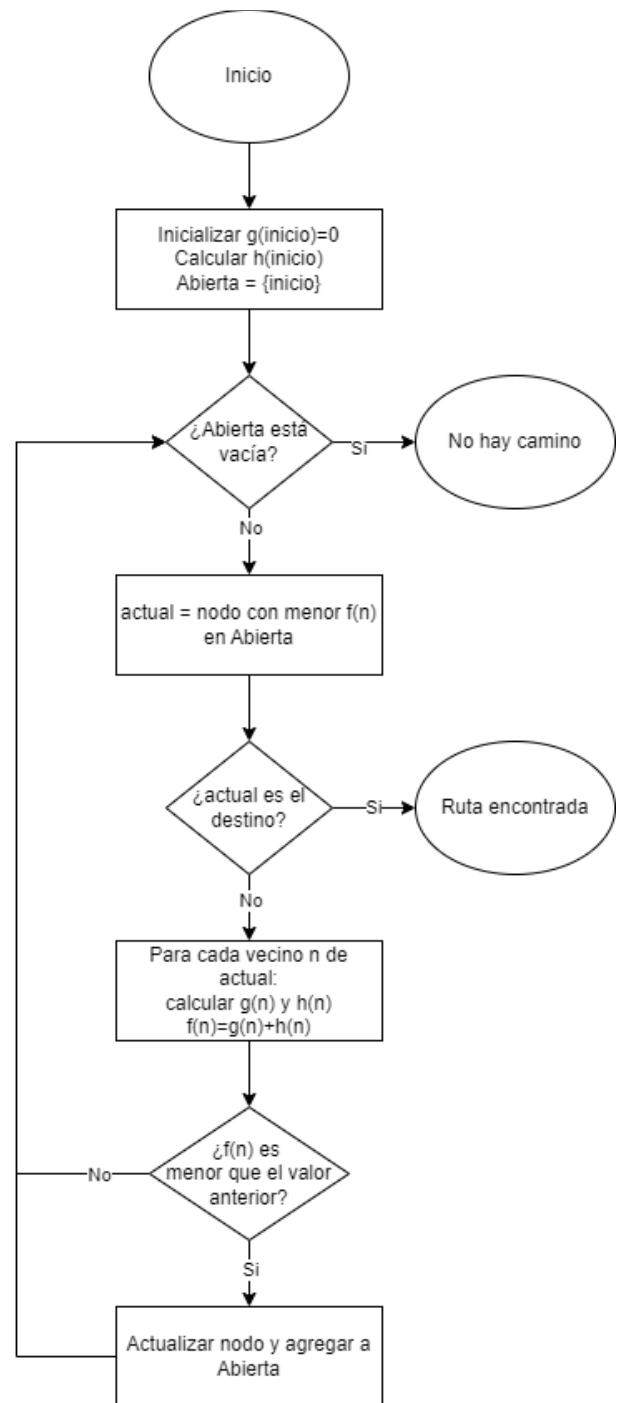


Fig. 5 Diagrama de flujo del pseudocódigo: algoritmo *A**.

3) *BSF* (*Búsqueda en amplitud*): Según la referencia [13], la publicación del algoritmo *BSF* se atribuye a Konrad Zuse en 1972. La finalidad del algoritmo es resolver problemas combinatorios (contar, ordenar y agrupar) en grafos no dirigidos. Es similar a Dijkstra porque tiene una distribución

por vecinos cercanos, pero funciona con capas de información donde los puntos más cercanos al inicio son la primera capa, los puntos cercanos a esta capa serán la segunda, este proceso se muestra en la ecuación “ $\text{camino}(u, v)$ ” (7), que crea las capas con “ u ” como punto actual y “ v ” como punto de comparación, y así sucesivamente hasta que una capa alcanza el destino, evaluando qué punto está más cerca para conectar una capa con la anterior. “ f ” es el punto final y “ p ” es padre o el punto anterior del punto actual, la ecuación “ ruta ” (8) construye una ruta con los padres del punto final, es decir que va del punto más cercano al final en la última capa, al punto más cercano de la capa anterior y así sucesivamente hasta llegar al punto de inicio, esta lógica se observa en el diagrama de flujo de la Figura 5 (Fig. 5) donde se empieza por crear la primera capa de puntos más cercanos al punto de inicio, continúa con la selección del nodo o punto más cercano de la capa actual y valida si es el punto final, que en caso de no serlo, se toman los puntos cercanos a la capa anterior para crear una nueva capa. La abertura es el parámetro para integrar en el algoritmo BSF, porque este el algoritmo crea las capas de los candidatos y restringe la diferencia de elevaciones como un criterio adicional, así se reducirá la cantidad de candidatos que podría tomar de cada curva.

$$\text{camino}(u, v) = \min_{\text{dist}(u, v)} |\text{dist}(u, v)| \quad (7)$$

$$\text{ruta} = [f, p[f], p[p[f]], \dots, \text{inicio}] \quad (8)$$

Las cuatro teorías de interés para organizar espacios en videojuegos definidas por Jesse Schell son:

- 1) *Organización Lineal*: El espacio está configurado de tal manera que el actor solo puede avanzar, y si quiere retroceder, será bajo ciertas condiciones.
- 2) *Organización en cuadrícula*: El espacio utiliza geometrías para su distribución.
- 3) *Organización en red*: El espacio marca puntos en el espacio y los conecta mediante caminos.
- 4) *Organización por espacios divididos*: Divide el espacio en secciones irregulares para replicar el terreno real.

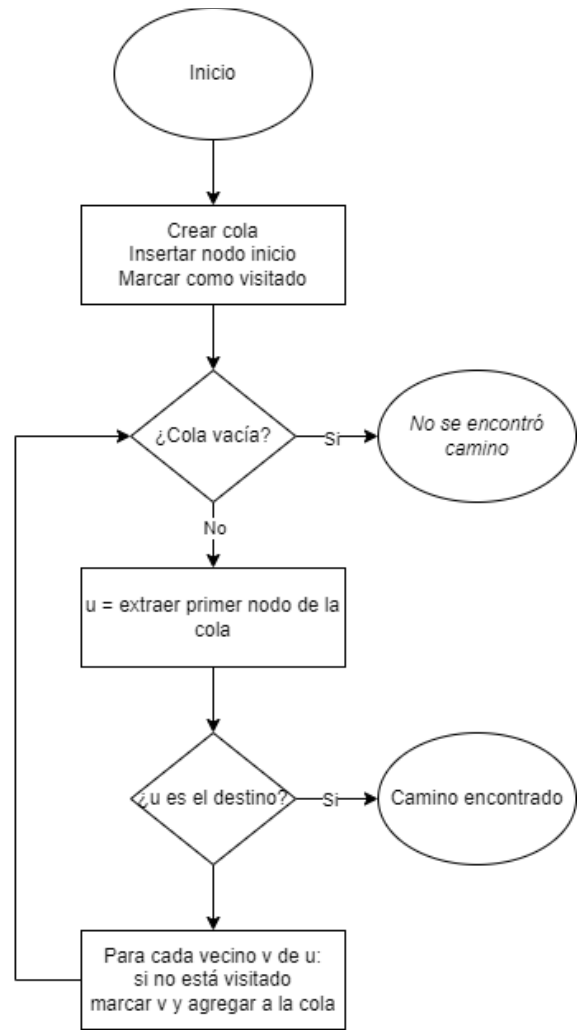


Fig. 6 Diagrama de flujo del pseudocódigo: algoritmo BSF.

IV. ANÁLISIS PROCEDIMENTAL DE LA LÍNEA DE CEROS Y SU DISEÑO ALGORÍTMICO.

A. Limpieza y Preprocesamiento de Datos

Después de entender el procedimiento manual y el método de optimización procedimental de la Línea de Ceros, se deben identificar las herramientas involucradas, el estado de los datos de entrada y la preparación de las condiciones óptimas para el desarrollo técnico del prototipo. Según la referencia [5], Dynamo es una herramienta de programación basada en bloques que permite la automatización de tareas, la generación de geometría y la personalización de flujos de trabajo para entornos de Autodesk como Civil 3D. Utiliza bloques relacionales compuestos por comandos y herramientas integradas entre Dynamo y Civil 3D, en los cuales el vértice, el punto y la línea son los elementos de estructurales fundamentales. Generalmente, la topografía del terreno comienza con una representación a través de vértices, la referencia [4] indica que Civil 3D utiliza herramientas para conectar estos vértices mediante polilíneas para representar las

líneas de contorno. También utiliza scripts en lenguaje LISP para la automatización versátil de procesos. Para limpiar los datos en las líneas de contorno se desarrollaron los siguientes tres scripts en lenguaje LISP: El primero elimina todo lo que está fuera de la selección actual de vértices, polilíneas y lo que no está en la misma capa de las líneas de contorno. El segundo comienza desde la selección de las polilíneas que limitan el área de interés y selecciona las polilíneas intermedias. Y el tercero es un script que convierte los vértices a puntos CAD basándose en la selección actual de las líneas de contorno y aplica una reducción porcentual al número de puntos, para obtener la nube de puntos CAD optimizada. Finalmente, para relacionar la información de los puntos CAD (nombre y coordenadas espaciales) con Dynamo, en Civil 3D los puntos CAD deben ser convertidos a puntos COGO para manipular los datos de la nube de puntos dentro de Dynamo, se utiliza el bloque de selección y las funciones que extraen las coordenadas X, Y y Z de los puntos COGO. Además, se desarrolla una función para devolver las coordenadas como un vector y agruparlas en una lista de salida para generar una matriz de vectores que Dynamo pueda leer.

B. Diseño de algoritmos

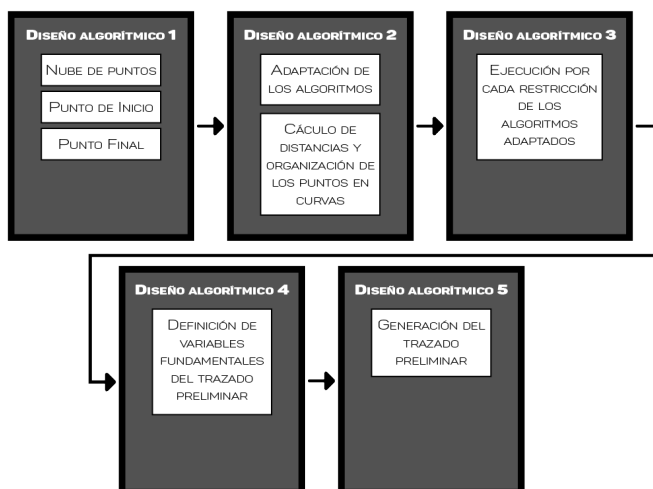


Fig. 7 Diagrama de los diseños algorítmicos.

Los cuatro procedimientos manuales descritos en el primer apartado se tradujeron en cinco diseños algorítmicos para detectar los patrones y replicarlos computacionalmente, como se pueden ver en la Figura 7 (Fig. 7). En el primer diseño algorítmico, los datos preprocesados se transfieren de Civil 3D a Dynamo a través de bloques de selección de objetos, que incluyen la nube de puntos y los puntos de inicio y fin. Estos datos se utilizan para extraer las coordenadas espaciales y organizarlas en una lista. Se aplica un filtro para eliminar entradas duplicadas, y los datos se pasan a un bloque de Python que asigna puntos a sus curvas correspondientes utilizando dos criterios: elevación y la distancia entre los ejes X y Y.

El segundo diseño algorítmico establece la estructura general común a todos los algoritmos, introduciendo variaciones lógicas según el algoritmo específico implementado. El proceso comienza con un bloque que recibe la nube de puntos y los puntos de inicio y fin designados, generando un diccionario que asigna cada punto a su curva correspondiente evitando los duplicados. Luego se ejecuta una función de distancia euclidiana para calcular la distancia entre los puntos, seguida de una función para la determinación del vecino más cercano que se adapta a cada algoritmo. Para Dijkstra y BFS, el punto y la curva actuales se comparan con todos los demás puntos y curvas para determinar las distancias mínimas y máximas basadas en el número predefinido de vecinos. En contraste, la implementación de A* identifica los vecinos más cercanos sin considerar inicialmente la curva, validando posteriormente la pertenencia a la curva para los puntos más cercanos. El procedimiento continúa con la ejecución de una función nombrada según el respectivo algoritmo. Finalmente, una función de temporización mide el tiempo de ejecución de cada estructura algorítmica. La salida de cada algoritmo consiste en un conjunto de los puntos procesados seleccionados y los datos correspondientes del tiempo de ejecución.

El tercer diseño algorítmico utiliza la primera ruta generada como un camino de referencia con el cual se pueden comparar los resultados subsecuentes. Esto produce un nuevo conjunto de puntos, denominado aquí como la ruta. Los algoritmos adaptados varían con las siguientes restricciones definidas según las teorías para organizar espacios en los videojuegos: (I) El punto uno no puede ser el mismo en ambas rutas; (II) El punto actual de la ruta recién generada se compara con los puntos anterior y siguiente de la ruta previamente generada; (III) El punto actual de la nueva ruta se compara con los puntos anterior y siguiente de las dos rutas generadas previamente. Estas restricciones están diseñadas para evaluar si mejoran o empeoran el resultado final de la ruta para cada algoritmo. Como resultado, se generan nueve rutas distintas, correspondientes a cada combinación de algoritmo y restricción.

El cuarto diseño algorítmico, define las siguientes variables fundamentales: La diferencia de alturas entre los puntos de inicio y fin, llamada Delta. Se obtienen las elevaciones de la nube de puntos que inicialmente ingresaron a Dynamo, estas elevaciones se procesan para extraer los valores únicos y se organizan de mayor a menor para luego validar las diferencias entre ellas y obtener el promedio para encontrar la Equidistancia. Para encontrar la Longitud Aproximada, se utilizan los puntos resultantes del tercer diseño algorítmico y se procesan en un bloque de Python que ejecuta las siguientes funciones: Obtener la distancia entre los puntos y luego los segmentos compuestos por dos puntos para cada ruta, que se suman en serie; Con el Delta de las alturas de los puntos de inicio y fin y la Longitud Aproximada para cada ruta, se calcula

la Pendiente Aproximada. Finalmente, con la Equidistancia y la Pendiente Aproximada, se calcula la Abertura para cada ruta.

El quinto diseño algorítmico, prepara los datos para hacer el primer Diseño Preliminar de la Línea de Ceros. Para esto, se llaman las nueve rutas resultantes del tercer diseño algorítmico, los puntos resultantes del primer diseño algorítmico y los puntos inicial y final. Para crear grupos de puntos por curva, se crea un diccionario con "Claves" que contiene las coordenadas espaciales del punto y su respectiva curva, las cuales luego se condicionan para evitar repeticiones. La primera iteración del Diseño Preliminar utiliza un bloque de Python que llama a los puntos resultantes del primer diseño algorítmico, las Aberturas de Corte, el arreglo con las curvas ordenadas y los puntos de inicio y fin. Con estos datos, se ejecutan tres funciones: La primera agrupa los puntos por cada curva en un diccionario. La segunda compara la distancia desde el punto actual hasta el punto final. Y la tercera compara la Abertura y las distancias con el punto actual.

V. RESULTADOS Y DISCUSIÓN

Los resultados presentados en las Figuras 8, 9 y 10 (Fig. 8, Fig. 9, Fig. 10), corresponden a las Rutas de Puntos generadas del tercer diseño algorítmico y a sus respectivos Diseños Preliminares de la Línea de Ceros. Las líneas de contorno en color gris proporcionan la referencia espacial de un terreno montañoso, con pendientes longitudinales entre el seis y el ocho por ciento (6% - 8%), equidistancia de 5 metros y sin obstáculos naturales de acuerdo con la caracterización del Manual Colombiano de Diseño Geométrico de Vías de INVIAS de la referencia [7]. La representación gráfica permite comparar directamente las tres restricciones definidas: restricción (I) representada en color púrpura, la segunda (II) en color verde y la tercera (III) en color naranja.

Para el algoritmo de Dijkstra (Fig. 8), las Rutas de Puntos obtenidas corresponden a una aproximación inicial punto a punto entre el inicio y el destino, y un Diseño Preliminar respectivo. La primera Ruta de Puntos, representada en púrpura, es la más larga ya que es la primera opción generada por el algoritmo; se observa que desciende siguiendo una línea de contorno, lo que indica que no considera la dirección en su camino.

Las Rutas de Puntos en verde y naranja comparten gran parte de su recorrido mediante puntos intermedios cercanos, aunque difieren en los saltos entre líneas de contorno, lo que evidencia el funcionamiento del criterio de diferenciación del algoritmo. En los Diseños Preliminares, se observa que el trazado intercepta cada línea de contorno sin omisiones. El Diseño Preliminar púrpura alcanza el punto de destino porque su último segmento es menor que la apertura definida; en contraste, los otros dos alcanzan la curva final, pero mantienen una distancia considerable respecto al destino.

Las rutas generadas por el algoritmo A* (Fig. 9), exhiben saltos entre curvas sin una distribución homogénea de puntos.

La excepción es la Ruta de Puntos Naranja, que sigue en su mayoría la misma curva y combina segmentos de otras dos. Los Diseños Preliminares, con menos intersecciones, muestran un mejor rendimiento: el morado llega al destino, aunque con un tramo final más corto. Los otros Diseños Preliminares terminan en la última curva a una corta distancia del destino, lo que valida el camino con una Pendiente constante. En este caso, la longitud de la Ruta de Puntos de la primera restricción (I) influye directamente en la Abertura.

Las rutas del algoritmo BFS (Fig. 10), son similares entre sí, pero con saltos abruptos entre puntos y curvas, especialmente en la parte superior de la ruta. Aunque esto sugiere posibles problemas en el enrutamiento, los Diseños Preliminares son adecuados ya que respetan el orden y el número de puntos por curva sin fijar el destino como el punto final. El Diseño Preliminar verde destaca, siendo el más cercano al destino y teniendo secciones de longitud constante.

La validación se basó en los siguientes criterios del Diseño Geométrico de Vías: proximidad al destino, consistencia de la Abertura y ausencia de retrocesos entre líneas de contorno. Bajo estos criterios, las rutas verde y naranja del algoritmo BFS fueron las más cercanas a un diseño adecuado, aunque no alcanzan el destino, debido a la similitud de sus Aberturas y a la convergencia de sus caminos.

Finalmente, para realizar una comparación técnica y cualitativa, en la TABLA I se comparan el tiempo de cálculo, el uso de RAM y el porcentaje de CPU de cada ruta para analizar su eficiencia computacional. El orden de las rutas corresponde a la secuencia de los algoritmos evaluados: Dijkstra, A* y BFS.

TABLA I
RESUMEN DE MÉTRICAS COMPUTACIONALES POR RUTA

Ruta	Nº puntos	Tiempo (s)	% CPU	RAM (MB)
1	199	52,606	93,56	671,75
2	173	60,836	88,29	988,38
3	171	63,872	96,99	969,73
4	208	64,938	97,1	1039,32
5	157	91,982	80,13	1011,16
6	171	42,099	88,35	1047,21
7	122	108,78	88,06	1543,21
8	109	93,99	88,44	1546,9
9	101	113,98	85,67	1520,7

Medición del tiempo de computación, consumo de RAM, porcentaje de uso de CPU y número de puntos para cada ruta generada por los algoritmos Dijkstra, A* y BFS

Los valores máximos de la TABLA I se usaron como referencia para calcular la eficiencia computacional relativa entre rutas. Los mayores registros fueron: ruta 9 en tiempo de cálculo con 114 s, ruta 4 en uso de CPU con 97%, y ruta 8 en consumo de RAM con 1457 MB.

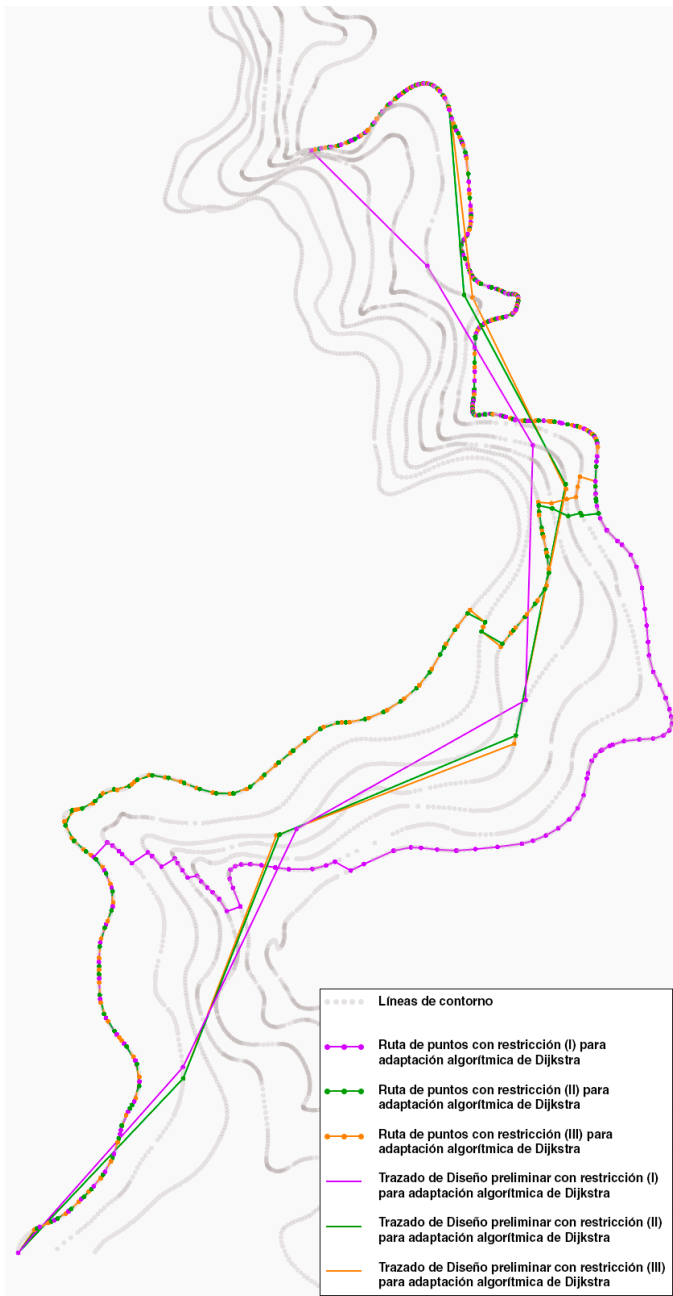


Fig. 8. Rutas de puntos generadas por el algoritmo de Dijkstra entre el punto inicial y el punto final, con las tres alternativas restringidas diferenciadas por color y sus respectivos Diseños Preliminares.

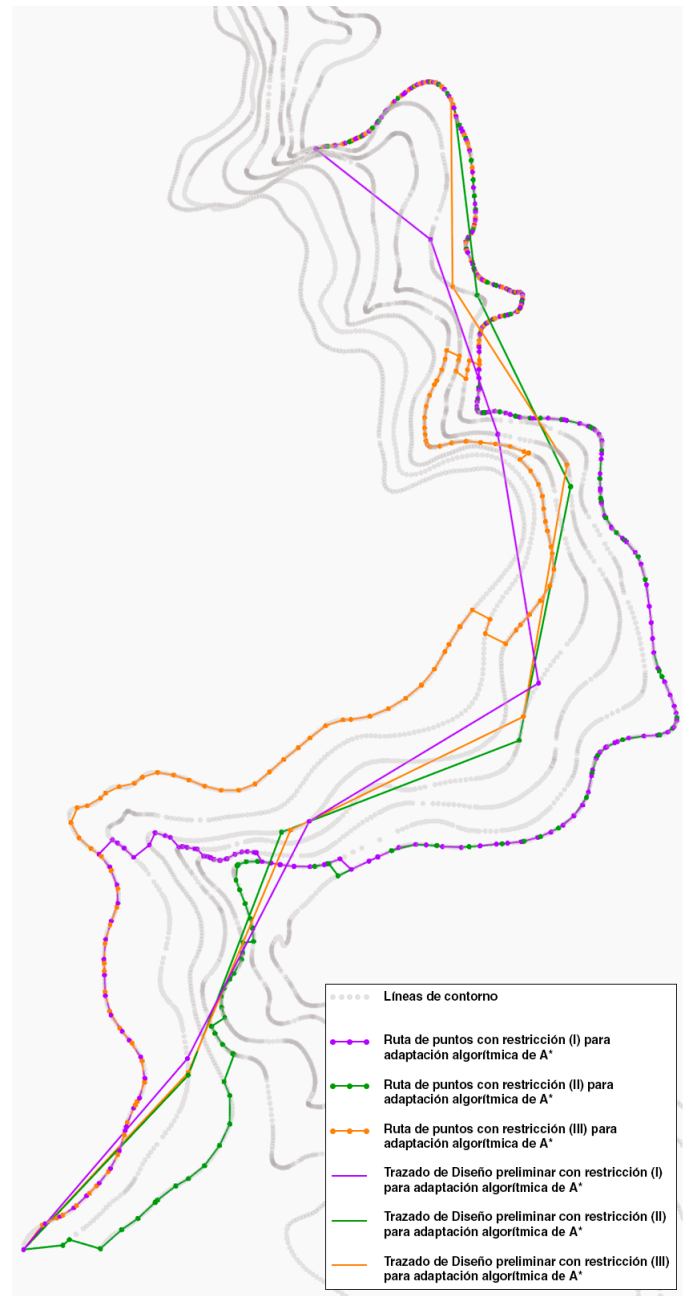


Fig. 9. Rutas de puntos generadas por el algoritmo de A* entre el punto inicial y el punto final, con las tres alternativas restringidas diferenciadas por color y sus respectivos Diseños Preliminares.

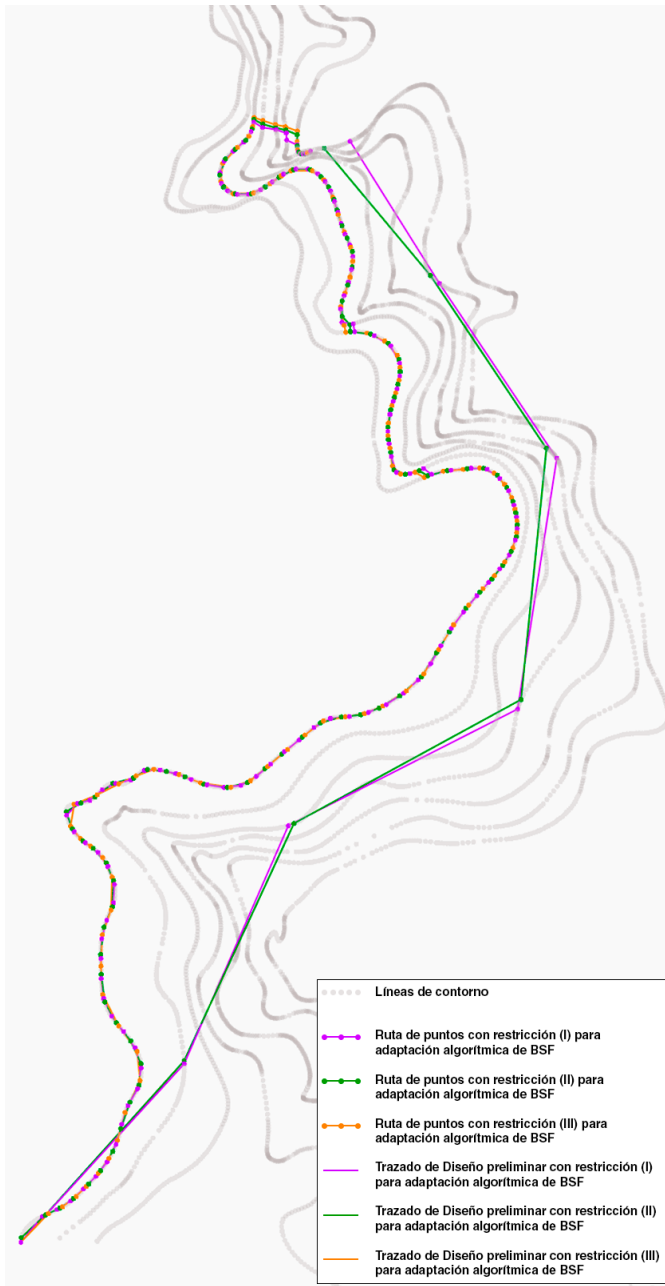


Fig. 10. Rutas de puntos generadas por el algoritmo de BSF entre el punto inicial y el punto final, con las tres alternativas restringidas diferenciadas por color y sus respectivos Diseños Preliminares.

TABLA II

RESUMEN DE EFICIENCIAS RELATIVA COMPUTACIONAL POR RUTA

Ruta	Eficiencia Tiempo	Eficiencia CPU	Eficiencia RAM
1	0,538	0,036	0,566
2	0,466	0,091	0,361
3	0,440	0,001	0,373
4	0,430	0	0,328
5	0,193	0,175	0,346
6	0,631	0,090	0,323
7	0,046	0,093	0,002
8	0,175	0,089	0
9	0	0,118	0,017

Eficiencia relativa computacional de cada medición por ruta

A partir de los valores de referencia de la TABLA I, los datos fueron normalizados respecto al valor máximo de cada métrica y expresados en una escala de eficiencia computacional entre 0 y 1, donde 1 representa el mejor desempeño. La TABLA II muestra que la ruta 6 obtuvo la mayor eficiencia en tiempo de cálculo con 42s, la ruta 5 el menor consumo de CPU con 80%, y la ruta 1 el mejor rendimiento de RAM con 671 MB. En esta primera fase, el prototipo trabajó con pendiente constante uniforme y equidistancia del modelo topográfico, para comparar los algoritmos bajo parámetros homogéneos. La Abertura se calculó internamente a partir de la Equidistancia y la Pendiente Aproximada, como valor de referencia para verificar la proporción constante entre curvas de nivel. Este enfoque aproxima el primer prototipo al procedimiento técnico base de la Línea de ceros, sin resolver todavía la totalidad del problema. Las métricas computacionales, como tiempo de cálculo, uso de RAM y porcentaje de CPU, se asumen como indicadores complementarios de desempeño y no como validación suficiente de la calidad ingenieril del trazado. Por ello, el modelo se plantea como una arquitectura extensible, capaz de incorporar variables como velocidad del diseño y tipo de vía, asociadas a una tabla paramétrica con valores mínimos de curvatura y Entretangencia mínima para contrastar el Diseño Preliminar con condiciones geométricas básicas.

VI. CONCLUSIONES

Los caminos preliminares generados por el algoritmo BFS muestran un avance progresivo y secuencial de las líneas de contorno, sin omitir elevaciones y alcanzando la curva final esperada. Aunque no alcanza el punto exacto, logra la elevación deseada, indicando un comportamiento consistente con el principio de pendiente constante y la dirección de la ruta. Esto asegura un diseño coherente entre la topografía y la lógica utilizada.

Los Diseños Preliminares resultantes no presentan retrocesos en las líneas de contorno, cambios abruptos de

dirección ni estancamiento en la misma línea de contorno. La orientación del trazo permanece constante sin reversos en la dirección, lo que demuestra que está directamente condicionada por el destino. Esto valida la efectividad de los criterios sobre la definición de restricciones utilizados para cada ruta generada.

Según los porcentajes de eficiencia presentados en la tabla II se puede concluir que el algoritmo más eficiente es el A*, ya que dos de sus rutas son las mejores en dos métricas, las ruta 6 y 5 destacan por ser el tiempo más corto con 42s y el menor consumo de CPU con 80%, respectivamente. BFS es el algoritmo que menor eficiencia presentó lo cual presume de mayor cómputo para los resultados.

La Abertura entre puntos en las trayectorias preliminares permanece proporcional excepto en las primeras rutas de los algoritmos Dijkstra y A*, ya que en estos dos casos el trazado llega al destino, pero el último segmento es más corto que el espaciado correspondiente, lo que indica que no hay una pendiente constante uniforme a lo largo de toda la ruta. Las otras rutas alcanzan la última línea de contorno, pero se desvían del destino. Es notable que el algoritmo BFS es el que más se acerca al destino, manteniendo una apertura constante y respetando el principio de pendiente constante.

Los Diseños Preliminares generados por el algoritmo BFS logran alcanzar una distancia efectiva, considerando la pendiente uniforme y constante del diseño. Lo cual sugiere que ajustar este camino requerirá menos iteraciones en comparación con otras rutas. Sin embargo, las rutas preliminares que alcanzan el punto de destino también pueden considerarse como un preámbulo de pocas iteraciones. Se requiere un ajuste de pendiente para todos los algoritmos con el fin de alcanzar no solo la elevación deseada, sino también el punto exacto requerido y no cortar más de una línea de contorno durante los trazados de las líneas segmentadas que formarán el Diseño Preliminar. Esto dicta que la parametrización y representación de los trazados preliminares deben ser modificados para que se puedan cumplir todos los requisitos técnicos de una Línea de Ceros y así reducir los movimientos de tierra.

Se recomienda que el prototipo final ajuste sus algoritmos integrando criterios de la fase de factibilidad y diseños definitivos, asegurando el cumplimiento de las especificaciones del perfil longitudinal, las secciones transversales y los requerimientos generales del Diseño Geométrico de Vías.

VII. AGRADECIMIENTOS

A la Universidad Militar Nueva Granada por financiar este proyecto de investigación titulado “Integración de herramientas digitales multimedia en el diseño geométrico de vías: Enfoque metodológico para la optimización de procesos” con código INV-ING-4158, perteneciente a la “Convocatoria interna para la financiación de proyectos de investigación - 2025-01”.

REFERENCIAS

- [1] J. H. Restrepo C. and J. J. Sanchez C., “Aplicación de la teoría de grafos y el algoritmo de Dijkstra para determinar las distancias y las rutas más cortas en una ciudad,” *Scientia et Technica*, vol. 10, no. 26, pp. 121–126, 2004. [Online]. Available: <http://www.redalyc.org/articulo.oa?id=84911640021>
- [2] D. E. Knuth, *The Art of Computer Programming*, vol. 1: Fundamental Algorithms, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.
- [3] J. Schell, *The Art of Game Design: A Book of Lenses*. San Francisco, CA, USA: Morgan Kaufmann, 2008.
- [4] Autodesk, Inc., *AutoCAD Civil 3D Developer’s Guide*. San Rafael, CA, USA: Autodesk, Inc., 2023.
- [5] Autodesk, Inc., *Dynamo Developer’s Guide*. [Online]. Available: <https://developer.autodesk.com> Accessed: Aug. 1, 2025.
- [6] J. Cárdenas Grisales, *Diseño Geométrico de Carreteras*, 2nd ed. Bogotá, Colombia: Ecoe Ediciones, 2013.
- [7] Instituto Nacional de Vías, *Manual de diseño geométrico de carreteras*. Bogotá, Colombia: Instituto Nacional de Vías, 2018.
- [8] J. J. Agudelo Ospina, “VÍAS: software libre para el diseño geométrico de vías, topografía y SIG,” *Revista Ingeniería de Construcción*, vol. 23, no. 1, pp. 52–59, 2008, doi: 10.4067/S0718-50732008000100006.
- [9] L. Moreno González-Páramo, *Herramientas avanzadas de análisis de datos de aplicación en ingeniería civil: Machine Learning using Python*. 2017.
- [10] D. Ramos Narváez, “El algoritmo de Dijkstra desde el Álgebra Tropical,” *Monografía de trabajo de grado, Proyecto Curricular de Matemáticas*, Bogotá D.C., Colombia, jun. 2025.
- [11] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
- [12] P. E. Hart, N. J. Nilsson and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” in *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968, doi: 10.1109/TSSC.1968.300136.
- [13] W.K. Gioli, “Plankalkül de Konrad Zuse”: el primer lenguaje de programación de alto nivel, ‘no von Neumann’, “*IEEE Annals of the History of Computing*”, vol 19, no. 2, pp. 17–24, Apr-Jun 1997, doi:10.1109/85.586068.