BRAVO SERNA DAVID ALVARO

GALLO MUÑOZ JUAN CARLOS

SORIA MUÑOZ DANNY PAUL

TORRES VERA DANILO ADRIAN

Estrategias de diseño: Programación Dinámica

Fuerza Bruta

Este algoritmo compararía cada curva con todas las demás en un conjunto de N curvas.

Ventajas:

- Es fácil de implementar
- Garantiza que se encuentre una solución exacta, ya que examina todas las posibles combinaciones

Desventajas:

- Es extremadamente ineficiente para conjuntos grandes, con una complejidad temporal de O(N^2)
- Realiza muchas comparaciones redundantes

Dividir y Conquistar

El conjunto de curvas se divide en subconjuntos más pequeños, y se comparan los subconjuntos entre sí antes de combinar los resultados.

Ventajas:

- Reduce el número de comparaciones necesarias al dividir el problema en subproblemas más pequeños y manejables
- Suele ser más rápido que la fuerza bruta, dependiendo de cómo se divida el conjunto

Desventajas:

- La división y combinación de resultados puede complicar la implementación
- No garantiza una reducción significativa en el tiempo de computación en comparación con la fuerza bruta, especialmente si las curvas no se dividen de manera eficiente

Programación Dinámica

Este algoritmo utiliza un enfoque de "memorización" para almacenar los resultados de comparaciones anteriores y reutilizarlos, evitando el trabajo redundante.

Ventajas:

- Evita el recálculo innecesario al almacenar y reutilizar resultados
- Puede ser significativamente más rápido que los otros enfoques si hay muchas redundancias en las comparaciones
- Es especialmente útil si ciertas comparaciones de curvas o cálculos de curvatura se repiten con frecuencia.

Desventajas:

- Requiere un almacenamiento adicional para guardar los resultados intermedios
- La implementación puede ser más compleja en comparación con la fuerza bruta

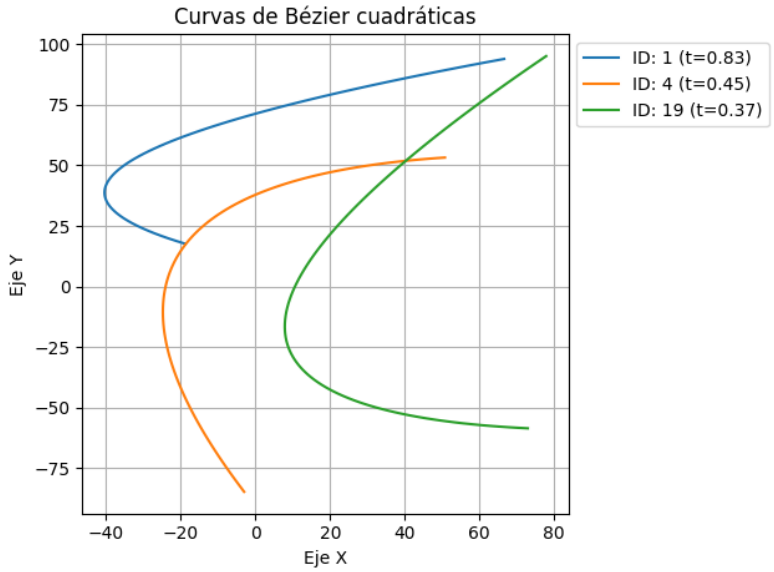
Justificación

La programación dinámica es la estrategia más adecuada para este problema porque las curvas de Bézier pueden tener características comunes que se repiten. Al almacenar los resultados de las comparaciones anteriores, el algoritmo puede evitar recálculos redundantes y acelerar el proceso general de comparación. Aunque se necesita un almacenamiento adicional, el beneficio en la reducción del tiempo de computación justifica este costo. La fuerza bruta sería demasiado lenta y divide y vencerás no garantiza la eficiencia que proporciona la programación dinámica en este contexto.

Descripción de la solución

Nuestro problema es cómo comparar una cantidad N de curvas cuadráticas de Bezier, para esto, utilizando el principio de Reductibilidad [1] , podemos definir que nuestro problema A es comparar una cantidad N de curvas cuadráticas de Bezier y el problema B o la reducción de este problema, sería cómo extrapolar las características de la forma de una curva cuadrática de Bezier a través del tiempo t con el fin de obtener una métrica de similitud para estas curvas.

Nos enfocamos en solucionar B, con el fin de poder solucionar A; para esto nuestra solución del problema B se basa en el método Dynamic Time Warping (DTW) para comparar y clasificar conjuntos de curvas utilizando un umbral de similitud. El Dynamic Time Warping (DTW) es una técnica que se utiliza para calcular la distancia óptima entre dos curvas “permitiendo” deformaciones en el tiempo, estas deformaciones facilitan la alineación de dos curvas para que puedan tener variaciones temporales entre sí. [2]



Esto lo podemos apreciar en esta imagen, estas tres curvas tienen características similares en su curvatura y en su forma, aunque no tengan la misma longitud de arco, al final su forma es similar.

Este algoritmo es útil para comparar series temporales que tengan patrones similares, pero estos patrones no ocurren exactamente en los mismos intervalos de tiempo. En lugar de comparar el primer punto de una curva con el primer punto de la otra, el segundo punto con el segundo, y así sucesivamente, el DTW permite "flexibilidad" en las comparaciones para tener en cuenta posibles desplazamientos temporales o distorsiones en la secuencia.

Para explicar esto mejor, piensa en dos secuencias temporales que representan la misma palabra pronunciada por dos personas diferentes. Una persona podría pronunciar la palabra más lentamente que la otra, lo que significa que los picos y valles de la representación de la onda sonora no se alinearán perfectamente si las comparas punto por punto. Sin embargo, la forma general de las ondas sonoras será muy similar.

Para realizar nuestra solución, utilizaremos el algoritmo DTW como base para ajustarlo a las características de nuestro problema, utilizando un enfoque de preferencias que vimos en clase, es decir, utilizaremos una curva cuadrática de Bézier como curva “base” de nuestras preferencias y en base a esta curva, se realizarán las respectivas comparaciones en base a un threshold, con las N curvas utilizando el algoritmo DTW modificado. Como resultado obtendremos un archivo output.txt con dos listas de identificadores con las curvas similares a la curva y curvas diferentes a la curva base.

De esta forma, al obtener una solución para el problema B, podemos resolver el problema A en base al principio de la reductibilidad.

Algoritmo: Dynamic Time Warping Modificado

Entrada

El algoritmo comienza leyendo un archivo que contiene las coordenadas de los puntos A, B y C de N curvas de Bézier cuadráticas, así como su identificador (id) y un valor t.

Pseudocódigo

```
Algoritmo: Clasificación de curvas usando Dynamic Time Warping (DTW) *MOD

Funciones:

- dynamic_time_warping(curve1, curve2) -> número:
    n <- longitud(curve1)
    m <- longitud(curve2)
    crear matriz dp de tamaño (n + 1) x (m + 1) inicializada con ceros

    para i desde 1 hasta n:
        dp[i][0] <- dp[i-1][0] + distancia(curve1[i-1], curve2[0])

    para j desde 1 hasta m:
        dp[0][j] <- dp[0][j-1] + distancia(curve1[0], curve2[j-1])

    para i desde 1 hasta n:
        para j desde 1 hasta m:
            coste <- distancia(curve1[i-1], curve2[j-1])
            dp[i][j] <- coste + mínimo(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])

    devolver dp[n][m]

- compare_curves_to_reference(reference_curve, curve, threshold=250) -> booleano:
    dtw_distance <- dynamic_time_warping(reference_curve, curve)
    devolver dtw_distance <= threshold
```

```
- classify_curves_with_reference(reference_curve, curves, threshold=250) -> lista, lista:
    lista similar_curves vacía
    lista different_curves vacía

    para cada curve en curves:
        si compare_curves_to_reference(reference_curve, curve[1], threshold) es verdadero:
            agregar curve[0] a similar_curves
        sino:
            agregar curve[0] a different_curves

    devolver similar_curves, different_curves

Inicio:

// Código principal
referencia <- curva de referencia (especificada por el usuario o el sistema)
curvas <- lista de curvas para comparar (por ejemplo, leídas desde un archivo)

curvas_similares, curvas_diferentes <- classify_curves_with_reference(referencia, curvas)
mostrar(curvas_similares)
mostrar(curvas_diferentes)

Fin
```

Funcionamiento

La función dynamic_time_warping inicialmente, crea una matriz dp de tamaño (n+1) x (m+1), donde n y m son las longitudes de curve1 y curve2 respectivamente. Luego inicializa la primera columna y la primera fila de dp con las distancias acumuladas entre los puntos de una curva y el primer punto de la otra curva.

Posteriormente, se llena la matriz dp usando un proceso iterativo. Cada entrada dp[i][j] representa la distancia mínima entre los primeros i puntos de curve1 y los primeros j puntos de curve2. Se calcula sumando la distancia entre curve1[i-1] y curve2[j-1] al mínimo de las tres entradas adyacentes: dp[i-1][j], dp[i][j-1] y dp[i-1][j-1]. Al final, dp[n][m] contiene la distancia DTW entre las dos curvas completas.

La función compare_curves_to_reference compara una curva dada con una curva de referencia usando DTW y devuelve True si la distancia DTW entre las dos curvas es menor o igual a un umbral (por defecto 150, valor que nos ha dado buenos resultados) o False en caso contrario.

Por último, la función classify_curves_with_reference clasifica un conjunto de curvas en dos grupos:

- Aquellas que son similares a la curva de referencia (es decir, tienen una distancia DTW con la curva de referencia menor o igual al umbral).
- Aquellas que son diferentes de la curva de referencia.

Para hacer esto, itera sobre cada curva en la lista curves y usa la función compare_curves_to_reference para determinar si es similar o no a la curva de referencia. Las curvas se añaden a las listas similar_curves o different_curves según corresponda. Al final, la función devuelve las dos listas: similar_curves y different_curves.

Salida

El algoritmo proporciona una clasificación de un conjunto de curvas basándose en su similitud con una curva de referencia específica. Utiliza el método de Dynamic Time Warping (DTW) para comparar las curvas y determinar la "distancia" entre ellas. Esta distancia no es simplemente una medida directa entre los puntos, sino que tiene en cuenta posibles desplazamientos temporales y distorsiones entre las

curvas, haciendo que la comparación sea más robusta en contextos donde las secuencias pueden variar en velocidad o estar desfasadas.

El algoritmo retorna dos listas:

- Curvas Similares (similar_curves): Esta lista contiene las curvas que tienen una distancia DTW con la curva de referencia que es menor o igual a un umbral predeterminado (por defecto es 250). Es decir, son curvas que, desde la perspectiva del DTW, son consideradas "similares" o "cercanas" a la curva de referencia.
- Curvas Diferentes (different_curves): Esta lista contiene las curvas cuya distancia DTW con la curva de referencia supera el umbral predeterminado. Estas son curvas que se consideran "diferentes" o "lejanas" de la curva de referencia.

A través de estas listas, el usuario o sistema puede identificar rápidamente qué curvas se alinean estrechamente con la curva de referencia y cuáles no. Esta clasificación puede ser valiosa en numerosas aplicaciones, como el reconocimiento de patrones, análisis de series temporales, y más.

Análisis

Complejidad Temporal

- Función dynamic_time_warping: O(n*m)
    - La creación de la matriz *dp* es O(n*m)
    - El primer bucle for recorre desde 1 hasta n y en cada iteración se realiza una operación que toma tiempo constante, entonces este bucle tiene una complejidad de O(n)
    - El segundo bucle for recorre desde 1 hasta m y al igual que el primer bucle, en cada iteración se realiza una operación que toma tiempo constante. Por lo tanto, este bucle tiene una complejidad de O(m)
    - El bucle anidado recorre desde 1 hasta n y desde 1 hasta m. Por lo que, en cada iteración del bucle anidado, se realizan operaciones que toman tiempo constante, al final este bucle tiene una complejidad de O(n*m)
- Función classify_curves_with_reference: O(N*n*M)
    - Este método itera sobre la lista de curvas y llama a *compare_curves_to_reference* en cada iteración. Si asumimos que hay N curvas y que cada curva tiene, en promedio, una longitud M, entonces la complejidad temporal de esta función será O(N*n*M) tanto espacial como temporal
- Compare_curves_to_reference: O(n*m)
    - Esta llama a dynamic_time_warping y compara el resultado con un umbral o threshold, por lo que hereda la complejidad de dynamic_time_warping de O(n*m)

Complejidad Espacial

- Función dynamic_time_warping: O(n*m)
    - La matriz dp tiene un tamaño de (n + 1) por (m + 1), por lo que toma un espacio de O(n*m)
    - Las demás variables utilizadas en la función toman un espacio constante.
- Función classify_curves_with_reference: Este análisis se realizó en la complejidad temporal y es O(N*n*M)

- Compare_curves_to_reference: La complejidad especial total es O(N + n*M), aunque podríamos considerar que solo es O(N) ya que n*M es un término intermedio que no se acumula con el número de curvas.
    - Las listas similar_curves y different_curves tienen en el peor de los casos, una longitud de N. Por lo tanto, la complejidad espacial es de O(N)
    - La complejidad espacial de la llamada interna a compare_curves_to_reference es O(n*M), pero este espacio se libera después de que se ha procesado cada curva individual.

Comentarios

Juan Gallo: El diseño del algoritmo aplicando el DTW es apropiado para abordar la similitud de las curvas teniendo en cuenta su forma y longitud y aplicando en conjunto con la programación dinámica para no volver a hacer cálculos redundantes

Paul Soria: Otra posible forma interesante para abordar el problema sería usando la métrica de Frechet, la distancia de Hausdorff, el área entre las curvas y la dispersión de la distancia entre las curvas que son métricas acertadas para comparar curvas cuadráticas por su forma para abordar las formas de las curvas en un espacio métrico y así compararlas.

Danilo Torres: La elección del algoritmo DTW es acertada para evaluar similitudes entre curvas considerando forma y longitud, respaldada por el enfoque en programación dinámica para evitar cálculos innecesarios. Esta estrategia garantiza una comparación precisa y eficiente al aprovechar la estructura de las curvas y al almacenar resultados previos.

David Bravo: Otra forma posible para calcular la similitud entre las curvas cuadráticas sería considerar el desplazamiento o “desarrollo” de la curva cuadrática en base al tiempo t, ya que pueden tener t diferentes y, por lo tanto, desarrollos diferentes. Lo que implican longitudes de arco diferentes, causando grandes cambios en el análisis y comparación de las curvas

Bibliografía formato IEEE

Se utilizaron los conceptos de reductibilidad del libro guía, además de conceptos y código implementado del algoritmo DTW en la web Towards Data Science en su blog web que luego fue previamente modificado para la realización de este trabajo.

[1] C. E. L. R. L. R. C. S. Thomas H. Cormen, "NP-completeness and reducibility," in *Introduction to Algorithms*, Cambridge, Masssachusetts London, England, The MIT Press, 2009, pp. 1067-1069.

[2] J. Zhang, "Dynamic Time Warping: Explanation and Code Implementation," 1 Feb 2020. [Online]. Available: https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd. [Accessed 28 Aug 2023].