Trabajo Practico

Emily Mac Carthy, Guido Salem y Serena

Sean (x1, y1), . . ., (xn, yn) ∈ R2 una serie n puntos, ordenados de forma que x1 < x2 < ・ ・ ・ < xn, provenientes de una función g(t) desconocida que buscamos aproximar. Consideramos también una grilla correspondiente a la discretización del área [x1, xn] × [y1, yn] ⊂ R2. El objetivo del trabajo practico consiste tomar como input los valores (xi, yi), i = 1, . . ., n, una discretización (ti, zj) con i = 1, . . ., m1, j = 1, . . ., m2, un valor K para la cantidad de breakpoints y calcular una función continua PWL f(t): [x1, xn] −→ R con K *breakpoints* (K −1 piezas f1(t), . . ., fK−1(t)) pertenecientes a la discretización (ti, zj) y que minimicen el error total

1. Fuerza Bruta

En fuerza bruta se nos pide implementar un algoritmo que para encontrar las rectas que minimicen el error según los puntos datos debe pasar por todos los puntos de la grilla posible a través de todas las combinaciones posibles que de la grilla mxn, calcular el mínimo error para así comparar con todas las líneas previamente calculadas. El problema inicial viene cuando los k breakpoints no son iguales al numero n, al tener menor breakpoints esto genera que los puntos no estén espaciados equidistantemente. Debido a esto nuestro enfoque a la solución se basa en eso, primero deberíamos encontrar todas las subgrillas posibles cuando el k es menor al n ya que dos de los breakpoints deben estar en la primer y última columna de la grilla. Esto nos deja con combinaciones posibles de como separar la grilla. Al correr una función auxiliar que nos da todas las combinaciones posibles de la grilla corremos la función fuerzaBrutaRecursiva que recursivamente intentara de encontrar el mínimo error. Esto lo hace a través ESCRIBIR COMO FUNCIONA EL PASO RECURSIVO, una vez que llega al caso base donde la lista de puntos agregados a la lista auxiliar coincide con la longitud de la grilla x. Esto después vuelve a la función principal y devuelve el error de esta subgrilla si el error dado es mas chico que el mejor error esto lo remplaza y vuelve a hacer lo mismo con la próxima subgrilla, así probando todas las posibilidades de grilla con todas las combinaciones de líneas resultando en el error mínimo de estas.

Una de las dificultades que tuvimos principalmente fue la redacción del TP, al no tener en claro que la cantidad de breakpoints era diferente a la cantidad de n columnas programamos el TP en función de eso y una vez que la terminamos nos enteramos de que no era así. Consecuentemente tuvimos que reescribir la función teniendo esta consideración en cuenta, también una de las razones por las que decidimos crear las subgrillas ya que corre con la función recursiva que habíamos hecho solo que con más casos.

Una consideración que tuvimos en este ejercicio fue ignorar la posibilidad de las rectas que no son factibles ya que nos impenitentes al problema ya que, si las recta no es factible, es decir el principio de una recta no se conecta con el final de otra recta nunca resolvería el problema aclarado con el TP.

MAYBE EXPPLICAR PORUEE ESTO SUCEDERIA TODAS LAS VECES

* 1. Experimentación

RESUMEN DE LOS RESULTADOS DE LA EXPERIMENTACION

La calidad de la aproximación mejora a medida que se agranda la discretización y la cantidad de piezas seleccionadas porque asegura tener un camino posible a través de los puntos. Los parámetros que más afectan la performance de los algoritmos son los m y los k cuando se eligen de alguna manera donde k es mas grande que 2 pero mucho mas chico que m2. Esto genera una gran cantidad de subconjuntos posibles por lo tanto el algoritmo tiene que recorrer muchas subgrillas y a su vez una gran cantidad de arboles recursivos para cada subgrilla si el m1 es grande.

FALTA DIFERNECIA EN C CON PYTHON. Y PROPUESTA

* 1. Conclusiones

Algunas posibles mejores que se le puede hacer a este algoritmo y una poda por optimalidad, back tracking, ya que este algoritmo construye la rama entera del árbol para calcular el error sin considerar que si el error actual es mas grande que el mejor error ya debería desechar dicha rama.

1. Back tracking

Intro al problema y la decisión

Descripción de los algoritmos

El problema es muy parecido al de fuerza bruta con la diferencia que queremos crear podas del árbol recursivo una vez que verifiquemos que el error ya es mas grande que la solución optima que ya tenemos guardada. Por un lado, existe la poda de factibilidad, descartar las ramas de los arboles que no son factibles ya que las rectas no se conectan en los mismos breakpoitnts. Esto ya lo hicimos en fuerza bruta ya que el algoritmo hace la recursión desde el último punto calculado, generando un descarte de las ramas no factibles. La otra poda posible es la poda por optimalidad, esta se genera descartando caminos que ya sabemos que no van a ser solución. La forma que decidimos encararlo fue podar las ramas las que tenían un resultado mayor al mejor resultado actual y saltar a la siguiente.

Consideraciones generales que deberían tener

Dificultade encontradas

* 1. Experimentación

Resumen de resultados

*Calidad*: Como impacta la granularidad de la discretización y la cantidad de

piezas seleccionadas en la calidad de las aproximaciones?

*Performance*: Cuales son los parámetros de las instancias que mayor impacto

tienen en la performance de los algoritmos? Como afecta la elección del lenguaje

de programación y la implementación elegida?

*Propuesta*: En base a los análisis previos, cuál sería la sugerencia del grupo

respecto al algoritmo, implementación y lenguaje de programación?

* 1. Conclusiones

Conclusiones allegadas

Posibles mejores> programación dinámica

Observaciones adicionales que consideran penitentes

1. programación Dinámica

Intro al problema y la decisión

Descripción de los algoritmos

Consideraciones generales que deberían tener

Dificultade encontradas

* 1. Experimentación

Resumen de resultados

*Calidad*: Como impacta la granularidad de la discretización y la cantidad de

piezas seleccionadas en la calidad de las aproximaciones?

*Performance*: Cuales son los parámetros de las instancias que mayor impacto

tienen en la performance de los algoritmos? Como afecta la elección del lenguaje

de programación y la implementación elegida?

*Propuesta*: En base a los análisis previos, cuál sería la sugerencia del grupo

respecto al algoritmo, implementación y lenguaje de programación?

* 1. Conclusiones

Conclusiones llegadas

Posibles mejores> programación dinámica

Observaciones adicionales que consideran penitentes

5. **(3.5 puntos) Algoritmos, informe, presentación de resultados y delibera del Código.**

El modelo, la descripción de los algoritmos, los operadores, las decisiones de diseño, la

implementación, el testigo realizado, la presentación de resultados, instrucciones de

compilación y ejecución.

**Modalidad de entrega**

Se pide presentar el modelo y la experimentación en un informe de máximo 15 páginas que

contenga:

• introducción al problema y la decisión,

• descripción de los algoritmos propuestos e implementados, según corresponda,

• consideraciones generales respecto a la implementación del modelo, incluyendo dificultades

que hayan encontrado,

• resumen de resultados obtenidos en la experimentación,

• conclusiones, posibles mejoras y observaciones adicionales que consideren pertinentes.

Junto con el informe debe entregarse el Código con la implementación del modelo. El mismo

debe ser entendible, incluyendo comentarios que faciliten su corrección y ejecución.