

Consultes pel veí més proper en arbres k -dimensionals aleatoris

Anàlisi experimental del cost

GRAU A

Q1 CURS 2023-2024



Mateu Villaret Abio
Víctor Hernández Barragán
Houda El Fezzak Bekkouri
Adrián García Belmonte

Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

Índex

1	Introducció	2
1.1	Objectius i Metodologia	2
2	Disseny i implementació dels arbres	3
	<i>k</i> -dimensionals	3
2.1	Fonaments	3
2.1.1	<i>k-d trees</i> estàndard	3
2.1.2	<i>k-d trees</i> relaxats	3
2.1.3	<i>k-d trees</i> quadrats	3
2.1.3.1	Bounding Boxes	3
2.2	Algorismes	4
2.2.1	Correctesa	5
2.2.1.1	Actualització de la caixa delimitadora	5
2.2.1.2	Inserció	5
3	Algorisme del veí més proper	5
3.1	Correctesa	7
4	Experimentació	7
4.1	Generació de les mostres	7
4.2	Processament de les mostres	8
4.3	Estimació del paràmetre ζ	9
5	Conclusions	9
6	Descripció i valoració del procès d'autoaprenetatge	10
6.1	Metodologia	10
6.1.1	Construcció de proves	11
6.2	Autoaprenentatge	12
	Apèndixs	13
A	Pseudocodi de les funcions d'inserció	14
B	Figures	15
C	Gràfiques	16
	Bibliografia	19

Introducció

La cerca associativa de dades en aplicacions computacionals exerceix un paper essencial en el camp de la informàtica i l'anàlisi de dades. Aquest problema, tot i que aparentment senzill, es torna cada cop més complex a mesura que els conjunts de dades creixen i es diversifiquen, provocant que l'exploració de les estructures de dades que s'empren esdevingui pivotal. La selecció d'una eina de dades adequada pot impactar significativament l'eficiència dels algorismes per a la resolució d'aquests problemes computacionals.

Per abordar aquest problema, és crucial comprendre i avaluar les estructures de dades que poden donar suport de manera efectiva a les consultes associatives. La literatura acadèmica ha explorat aquesta problemàtica de diverses maneres. Investigacions prèvies han demostrat la rellevància de les estructures de dades multidimensionals en la resolució de consultes associatives. El treball pioner de Bentley i Friedman [1] va introduir els arbres k -dimensionals, que han estat àmpliament estudiats i aplicats en un ampli rang d'aplicacions, així com la mineria de dades, el processament d'imatges i la intel·ligència artificial [1].

Són indispensables en gràfics per computador per a tasques com la detecció de col·lisions i el traçat de raigs. En els sistemes d'informació geogràfica, els k - d trees faciliten la indexació espacial i la recuperació eficient de dades geoespaciales.

No obstant la importància d'aquestes estructures, encara hi ha preguntes obertes i reptes en l'avaluació del seu rendiment en consultes associatives específiques. En aquest sentit, la recerca de Bentley i Friedman [2] ha plantejat interrogants sobre el rendiment dels arbres k -dimensionals en el context de consultes per proximitat, la qual cosa ens conduceix a la motivació central d'aquest projecte.

El problema de les cerques pel veí més proper es defineix de la següent manera: Donat un conjunt d' n punts en un espai d -dimensional, $F \subseteq \mathbb{E}^d$, que representa un conjunt de registres amb d atributs o coordenades, i un *query point* $q \in \mathbb{E}^d$, l'objectiu és trobar el punt en F que minimitza la distància Euclidiana a q :

$$\min_{p \in F} \text{dist}(p, q) = \min_{p \in F} \sqrt{\sum_{i=1}^d (p_i - q_i)^2} \quad (1.1)$$

on p és un punt en F amb coordenades $\mathbf{p} = (p_1, p_2, \dots, p_d)$ i q és el *query point* amb coordenades $\mathbf{q} = (q_1, q_2, \dots, q_d)$. L'objectiu és determinar el punt p amb què aconseguim minimitzar la distància.

1.1 Objectius i Metodologia

Aquest projecte es centra en l'estudi experimental del cost mitjà de formulacions més eficients del problema de les cerques associatives, basades en el preprocessament dels punts en F i la construcció d'una estructura de dades que ens permeti realitzar la cerca en un temps millor que l'obtingut amb un algorisme de força bruta: $\Theta(n)$. En aquest projecte s'analitza el rendiment mitjà de les consultes pel veí més proper en arbres k -dimensionals per a diferents dimensions i variants de k - d trees.

$$C(n) = \Theta(n^\zeta + \log(n)) \quad (1.2)$$

Amb aquest objectiu, s'identificaran els registres de F amb les seves claus, representant-los com a punts en un espai k -dimensional amb coordenades $x = (x_0, x_1, \dots, x_{k-1})$. Es suposarà que cada coordenada x_i pertany a l'interval $D_i = [0, 1]$, de forma que D es converteix en un hipercub $[0, 1]^k$.

La metodologia emprada combina la recerca bibliogràfica, l'experimentació i l'avaluació crítica dels resultats per adquirir una millor comprensió d'aquestes estructures de dades i les seves variants: els k - d trees estàndard, relaxats, i quadrats; explorant així les diferències crítiques en el rendiment i comportament d'aquestes variants per a diferent dimensions.

Disseny i implementació dels arbres k -dimensionals

2.1 Fonaments

Els k -*d trees* resulten d'una generalització de l'arbre binari de cerca per a k dimensions. Consisteixen en estructures de dades jeràrquiques utilitzades per a la partició multidimensional de dades que ens permeten la cerca i recuperació eficients dels punts k -dimensionals que guarden.

Un k -*d tree* es construeix mitjançant la partició recursiva de les dades al llarg de dimensions alternatives. A cada nivell de l'arbre, es selecciona una dimensió de partició, que divideix les dades en dos subconjunts al llarg d'un hiperplà perpendicular a aquesta dimensió.

L'eficiència dels k -*d trees* es basa en aprofitar aquestes particions per eliminar regions senceres de l'espai de cerca durant el seu recorregut. Aquesta propietat les fa particularment adequades per a aplicacions que involucren dades de moltes dimensions, on els mètodes de cerca tradicionals esdevenen computacionalment costosos.

2.1.1 k -*d trees* estàndard

Es considera un conjunt de registres k -dimensionals. Cadascun dels nodes de l'arbre emmagatzema un dels registres i el seu corresponent discriminant.

Si $n = 0$, l'arbre és buit. En cas contrari, l'arrel de l'arbre emmagatzema un registre amb una clau x i el node corresponent està associat amb un discriminant i . Aleshores l'espai es divideix en dues regions respecte a $x[i]$. De forma que tota clau u amb $u[i] < x[i]$ s'emmagatzema al subarbre esquerre, i tota clau u tal que $u[i] > x[i]$ s'emmagatzema al subarbre dret. En el cas dels k -*d trees* estàndard el discriminant i es determina pel nivell de l'arrel mòdul k , tal que $0 \leq i < k$.

2.1.2 k -*d trees* relaxats

Els k -*d*-arbres relaxats representen una adaptació de l'estructura dels k -*d*-arbres estàndard. Introdueixen el concepte de l'equilibri relaxat, que fonamenten irregularitats en l'estructura de l'arbre. En un k -*d*-arbre relaxat, en lloc d'alternar estrictament les dimensions a cada nivell, la selecció del discriminant es realitza de forma arbitrària i uniforme entre les k possibilitats.

2.1.3 k -*d trees* quadrats

Els k -*d trees* quadrats, o *squarish* són una variant que utilitza les *bounding boxes* amb la finalitat d'abordar les limitacions dels k -*d trees* estàndard, especialment quan es tracten regions allargades dels rectangles que defineixen les particions en diversos plans que genera l'arbre k -dimensional.

2.1.3.1 Bounding Boxes

Com s'ha esmentat anteriorment, els arbres k -dimensionals divideixen l'espai \mathbb{E}^k en regions més petites mitjançant divisions recursives. Aquestes regions, també anomenades caixes delimitadores o *bounding boxes* representen una partició del domini D en $n - 1$ regions, on n és el nombre de nodes. En el nostre cas, tenim que tots els punts pertanyen a l'hipercub $D = [0, 1]^k$.

Cadascun d'aquests punts, a més de les coordenades que el constitueixen i el discriminant, també té associada una regió d'aquest espai k -dimensional. L'arrel de l'arbre r té associada la regió sencera $[0, 1]^k$, alhora que la divideix en dues capses delimitadores segons l'eix pel qual discrimina. Cada nivell de l'arbre divideix l'espai recursivament a través d'una de les k dimensions.

Per tant, cada node que no sigui una fulla genera implícitament un hiperpla perpendicular a la dimensió en què es discrimina, i tots els nodes del subarbre esquerre (amb $p[i] < r[i]$) cauen a l'esquerre de l'hiperpla, mentre que tots els nodes del subarbre dret (amb $p[i] > r[i]$) cauen a la dreta de l'hiperpla. Aquestes divisions continuen recursivament amb la inserció de cada node. A la figura B.1 es pot observar un exemple de partició de l'espai en el procés de construcció d'un *k-d tree* squarish.

Intuïtivament, les *bounding boxes* $B = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_k, b_k]$ son capses *k*-dimensionals de mesures mínimes de forma que contenen tots els punts del subarbre que té com a arrel el node respectiu.

En els *k-d trees* quadrats es modifica la forma en què s'escull el discriminant a cada node, de manera que es talla constantment el costat més llarg de la capsa delimitadora. Això garanteix que les particions resultants tinguin formes més quadrades, mitigant el problema d'allargament observat en els arbres *k*-dimensionals estàndard. Com s'observarà en seccions posteriors, la capacitat d'aquesta variant per crear particions més equilibrades contribueix significativament a millorar el rendiment de les consultes.

2.2 Algorismes

Les funcions *i_insert* i *insert* - de les quals es pot observar el pseudocodi en l'apèndix 2 i 3 - es fan servir, com el seu nom indica, per inserir un nou node a l'arbre, de manera que la distribució dels nodes es corresponga amb l'estruatura resultant de cada mètode de partició.

Primerament es crida a la funció *insert*, la qual rep com a entrada les coordenades del punt a inserir i el tipus d'arbre que s'està tractant. La funció inicialitza una *bounding box* que ocupa tot el pla i correspon a la caixa delimitadora del node arrel. També s'incrementa el número de nodes que té l'arbre i es crida a la funció *i_insert* amb el node arrel com a node actual i profunditat 0.

La funció *i_insert* té com a paràmetres d'entrada el node actual que s'està tractant (que es correspon amb l'arrel de l'arbre a la primera crida), el punt a inserir, la profunditat de l'arbre, el seu tipus i la *bounding box* del node pare de l'actual.

La idea de l'algorisme d'inserció és començar per l'arrel de l'arbre i seguir un recorregut descendent. Aquest recorregut és determinat pel valor de la coordenada del node a inserir en la dimensió amb què discrimina cada node no buit del recorregut fins a arribar a un node *null*.

És llavors quan es fa el càlcul del discriminant, es crea el nou node amb les coordenades que rep la funció com a paràmetre i amb els punters als seus fills inicialitzats a *null*.

Aquest càlcul del discriminant es realitza com ja s'ha explicat anteriorment: en cas que el tipus de l'arbre sigui estàndard, es pren la profunditat de l'arbre i se li aplica el mòdul *k*; si s'està treballant amb un arbre relaxat simplement es crida a *rand* perquè es determini de forma aleatòria; i en el cas que sigui de tipus quadrat el discriminant *j* es selecciona trobant l'eix més llarg de la *bounding box* del pare:

$$j = \max_{i=1}^k (b_i - a_i) \quad (2.1)$$

On:

j : Longitud màxima entre totes les dimensions

k : Nombre de dimensions

a_i : Element *i*-èssim del vector *Bb.minPoint*, que representa la coordenada mínima en la dimensió *i*

b_i : Element *i*-èssim del vector *Bb.maxPoint*, que representa la coordenada màxima en la dimensió *i*

La crida recursiva es fa tot comparant les coordenades del node a inserir amb les del node actual, de forma que es decideix si l'algorisme continua pel subarbre esquerre o el dret. En aquest procés també s'assigna la *bounding box* actualitzada al node corresponent.

2.2.1 Correctesa

2.2.1.1 Actualització de la caixa delimitadora

Quan $\text{info}[\text{curr}.discr]$ és menor que $\text{curr.x}[\text{curr}.discr]$, significa que el node actual s'ha de col·locar a l'esquerra. En aquest cas, la funció actualitza el valor màxim de la caixa delimitadora $Bb.\maxPoint$ per a la dimensió $\text{curr}.discr$ amb què discrimina el node curr . D'aquesta forma la caixa delimitadora cobreix tots els punts que, per correctesa de l'algorisme d'inserció, pertanyen al subarbre esquerre de curr , i per tant la *bounding box* que es passa en la crida recursiva amb el subarbre esquerre està correctament construïda.

D'altra banda, quan $\text{info}[\text{curr}.discr]$ és més gran o igual a $\text{curr.x}[\text{curr}.discr]$, indica que el node actual s'ha de col·locar a la dreta. En aquest cas, la funció actualitza el valor mínim de la caixa delimitadora $Bb.\minPoint$ per a la dimensió $\text{curr}.discr$. Aquesta actualització garanteix que la caixa delimitadora contingui tots els punts que pertanyen al subarbre dret que té com a arrel el node curr , de forma que al fer la crida recursiva amb el subarbre dret la *bounding box* està correctament actualitzada.

2.2.1.2 Inserció

Afirmació: L'algorisme presentat de la inserció insereix correctament els nodes.

Demostració per inducció:

- **Cas base:** Quan l'algorisme arriba a un node *null* significa que ha arribat al final del camí de l'arbre, de forma que el node *null* és la primera referència disponible per inserir el node. Un cop inserit, es calcula el discriminant i es retorna a la crida anterior, de manera que el pare pugui actualitzar les referències als seus fills.
- **Hipòtesi d'inducció:** Suposem que en un cert punt de l'execució de l'algorisme, el discriminant j del node curr està ben calculat i l'algorisme es situa al subarbre al que pertany el node a inserir.
- **Pas d'inducció:** A mesura que l'algorisme avança, descendeix pel camí de l'arbre de manera recursiva.
 - Si el valor $\text{info}[j]$ del node que es vol inserir és menor al valor del node actual, el node a inserir ha d'estar dins del subarbre esquerre. En cas contrari, ha d'estar dins del subarbre dret.
 - S'actualitza la *bounding box*.
 - Es crida recursivament i_insert amb el subarbre corresponent al qual pertany segons la coordenada j . D'aquesta forma, queda demostrat que el node s'insereix en el subarbre correcte.
- **Convergència:** Cada crida recursiva es fa amb un arbre més petit.
- **Finalització:** L'algorisme acaba quan arriba al fill d'una fulla. El nou node fulla del camí és el node inserit, i l'antic node ha actualitzat la referència al node fill que s'ha inserit.

Algorisme del veí més proper

La funció *nearestNeighbor* s'utilitza per trobar el veí més proper al *query point* donat en un arbre *k-d*. Rep com a entrada l'arbre *k-d*, representat pel seu node arrel, el punt de consulta, la profunditat actual de l'arbre (començant amb 0 per a l'arrel), el node més proper trobat fins ara i la millor distància, que correspon a la distància entre el *queryPoint* i el node actualment candidat a més proper. La funció realitza una cerca recursiva a través de l'arbre per trobar el veí més proper.

Si el node actual és *null*, de forma que no hi ha més nodes per explorar en aquesta branca de l'arbre, la funció retorna el millor node trobat fins ara. Aquest és el cas base de la recursió.

La funció guarda a la variable *axis* l'eix pel qual el node actual divideix l'espai. A continuació, decideix quina branca explorar segons l'emplaçament teòric de *queryPoint* si aquest s'estigués inserint en l'arbre, respecte al node actual. Si la coordenada del punt de consulta al llarg de l'eix de la divisió és menor que la coordenada del node actual, selecciona la branca esquerre com a *nextBranch* i la branca dreta com a *otherBranch*. En cas contrari, realitza la selecció inversa.

Algorithm 1 nearestNeighbor

Paràmetres: root, queryPoint, depth, bestNode, bestDistance

```

1: if root is NULL then
2:   return bestNode
3: end if
4: axis  $\leftarrow$  root $\rightarrow$ discr
5: nextBranch  $\leftarrow$  NULL
6: otherBranch  $\leftarrow$  NULL
7: if queryPoint[axis] < root $\rightarrow$ x[axis] then
8:   nextBranch  $\leftarrow$  root $\rightarrow$ left
9:   otherBranch  $\leftarrow$  root $\rightarrow$ right
10: else
11:   nextBranch  $\leftarrow$  root $\rightarrow$ right
12:   otherBranch  $\leftarrow$  root $\rightarrow$ left
13: end if
```

La funció calcula la distància euclidiana entre les coordenades del node actual $root \rightarrow x$ i el punt de consulta. Si aquesta distància és menor que la millor distància trobada fins ara, actualitza el *bestNode* i la *bestDistance* amb la informació del node actual, actualitzant així el node candidat a veí més proper.

A continuació, explora de manera recursiva la branca seleccionada com a *nextBranch* per buscar el veí més proper, mitjançant una crida recursiva a *nearestNeighbor* amb *nextBranch*, augmentant la profunditat en 1 i proporcionant la solució candidata fins al moment. Aquest pas explora el costat del pla de divisió on es troba el punt de consulta.

Un cop s'ha explorat *nextBranch*, la funció calcula la distància absoluta al llarg de la coordenada de divisió entre el punt de consulta i el node actual. Si aquesta distància és menor que la millor distància trobada fins ara, és possible que hi hagi un punt més proper a *queryPoint* a l'altre costat del pla de divisió. Per aquesta raó, la funció explora de manera recursiva, per exhaurir aquesta possibilitat, la branca *otherBranch* per comprovar si hi ha un veí més proper.

Finalment, la funció retorna el millor node trobat entre totes les branques explorades.

Aquesta funció recorre eficientment l'arbre *k-d* per trobar el veí més proper a un punt de consulta donat. Es basa en l'exploració recursiva i la poda basada en càlculs de distància per optimitzar el procés de cerca. El millor node i distància s'actualitzen a mesura que la funció recorre les branques, assegurant que finalment es trobi el veí més proper.

Algorithm 1 nearestNeighbor: Part 2

```

1: visitedNodes  $\leftarrow$  visitedNodes + 1
2: currentDistance  $\leftarrow$  euclideanDistance(root  $\rightarrow$  x, queryPoint)
3: if currentDistance < bestDistance then
4:   bestNode  $\leftarrow$  root
5:   bestDistance  $\leftarrow$  currentDistance
6: end if
7: bestNode  $\leftarrow$  nearestNeighbor(nextBranch, queryPoint, depth + 1, bestNode, bestDistance)
8: axisDistance  $\leftarrow$  abs(queryPoint[axis] - root $\rightarrow$ x[axis])
9: if axisDistance < bestDistance then
10:   bestNode  $\leftarrow$  nearestNeighbor(otherBranch, queryPoint, depth + 1, bestNode, bestDistance)
11: end if
12: return bestNode
```

3.1 Correctesa

Afirmació: L'algorisme presentat del veí més proper en un arbre k -d troba correctament el veí més proper.

Demostració per inducció:

- **Cas base:** Quan l'algorisme arriba a un node fulla, la pròxima crida recursiva es troba un subarbre *NULL*. Donat que en aquest punt no hi ha cap altre node per comparar, la funció retorna el millor node *bestNode* trobat en el camí des de l'arrel fins al node terminal.
- **Hipòtesi d'inducció:** Suposem que en un cert punt de l'execució de l'algoritme, les variables *bestNode* i *bestDistance* emmagatzemem correctament el veí més proper trobat fins ara, i la seva distància.
- **Pas d'inducció:** A mesura que l'algorisme avança, explora els nodes de l'arbre de manera recursiva.
 - L'algorisme considera la distància euclidiana entre el punt de consulta i les coordenades del node actual.
 - Si la distància és menor que *bestDistance*, l'algorisme actualitza *bestNode* i *bestDistance*. Aquest nou node passa a ser el candidat actual a veí més proper.
 - L'algorisme procedeix a explorar la següent branca basant-se en la ubicació del punt de consulta q respecte a la coordenada de divisió actual del node si s'estigués inserint q .
 - La branca alternativa *otherBranch*, que ens porta a l'altre costat del pla de divisió, només s'explora si existeix la possibilitat de trobar un veí més proper en aquesta branca inexplorada. Aquesta decisió es pren basant-se en la distància absoluta al llarg de la coordenada de divisió entre el punt de consulta i el node actual. Si aquesta distància és menor que *bestDistance*, significa que existeix la possibilitat de trobar un veí més proper a l'altre costat del pla de divisió. Com que s'explora el subarbre alternatiu sempre que és necessari per a cada nivell, l'algorisme sempre retorna el veí més proper.
- **Convergència:** Cada crida recursiva es fa amb un arbre més petit.
- **Finalització:** L'algorisme acaba quan s'han explorat totes les branques necessàries, tal que cadascuna retorna al seu respectiu pare el *bestNode* que ha trobat, fins a arribar al node arrel de l'arbre. En aquest s'han explorat totes les branques necessàries, i es retorna efectivament el veí més proper a q .

Experimentació

L'objectiu principal del projecte gira entorn de la investigació i predicció del valor de ζ en l'expressió $\Theta(n^\zeta + \log n)$, que representa el cost mitjà de la funció de veí més proper en els arbres k -d. Els experiments que es presentaran i s'analitzaran en les seccions posteriors tenen com a objectiu obtenir una comprensió més profunda del comportament de l'algorisme de cerca del veí més proper en aquests arbres i entendre com varia aquest cost en funció de la dimensionalitat de l'espai (k).

A continuació, es delinearà la metodologia experimental emprada, es descriuran els conjunts de dades utilitzats i es presentaran els resultats dels experiments, així com els respectius ànalisis i conclusions. Aquests experiments parteixen d'un conjunt mostra d'arbres k -dimensionals d' N nodes cadascun. Així, les gràfiques presentades en aquesta secció permetran una visualització i ànalisi de la relació entre el nombre de nodes n i el cost mitjà Θ .

4.1 Generació de les mostres

El conjunt mostra s'ha subdividit segons la dimensió k i el mètode d'elecció del discriminant, de forma que cada subconjunt s'ha construït mitjançant un conjunt des de 1.000 fins a 100.000 nodes en increments de 5.000, resultant en un total de 20 mostres per a cada conjunt definit pel parell (k , mètode).

Per cada mostra es generen un total de 100 arbres que s'anomenaran T .

Generant 100 arbres per a cada N , s'obté un conjunt de dades prou gran per a treure conclusions significatives sobre el rendiment mitjà, tot gestionant també la complexitat computacional involucrada, sobretot a mesura que creix el nombre de nodes.

Incrementar significativament el nombre d'arbres més enllà de 100 podria no proporcionar beneficis addicionals substancials en el context de l'estudi, ja que aquest està enfocat en el cas mitjà. La variabilitat en els resultats probablement seguirà un patró de rendiments decreixents, on el guany marginal en precisió disminueix amb cada arbre addicional.

El nombre de consultes Q es manté constant per a tots els conjunts i no varia en funció del nombre de nodes N o la dimensió k . Aquest enfocament permet una comparació més precisa entre els diferents casos d'estudi ja que totes les variables, llevat dels factors específics que s'investiguen, romanen constants.

Per a generar tant les consultes com els nodes dels k -*d trees* s'utilitza un generador de nombres pseudoaleatori per assignar els valors. Es parteix de la base que aquesta generació pseudoaleatòria de valors en genera un conjunt uniformement distribuït.

En les gràfiques que s'observen a la figura B.2, es poden examinar diferents distribucions de punts de consulta sobre el pla bidimensional que els punts de l'arbre conformen.

Una distribució uniforme dels *query points* emprats per a les consultes que es realitzen sobre la mostra és pivotal, ja que assegura que l'espai de mostra estigui cobert de manera uniforme. Això promou la representativitat estadística minimitzant el biaix de selecció, ja que cada part de l'espai de mostra té la mateixa probabilitat de ser inclosa en la mostra. Usar una distribució uniforme assegura que la mida de la mostra en cada regió de l'espai de mostra sigui suficient per a un anàlisi estadístic significatiu. En distribucions no uniformes, algunes regions podrien estar sobrerepresentades, mentre que d'altres n'estarien infrarepresentades, el que pot conduir a conclusions estadístiques no fiables i a una baixa significació estadística.

Per determinar el paràmetre Q , s'ha per les representacions dels arbres i la visualització de diferents nombres de consultes generades. Es pot observar que a la primera gràfica de la figura B.2, amb $Q = 10.000$, s'obté una distribució prou uniforme dels punts de consulta.

4.2 Processament de les mostres

Per dur a terme el processament, per a cada k i mètode, es llegeixen les dades i es calcula la mitjana que s'obté del cost de cadascun dels 100 arbres generats. Un cop es disposen d'aquests valors, se n'observa la tendència:

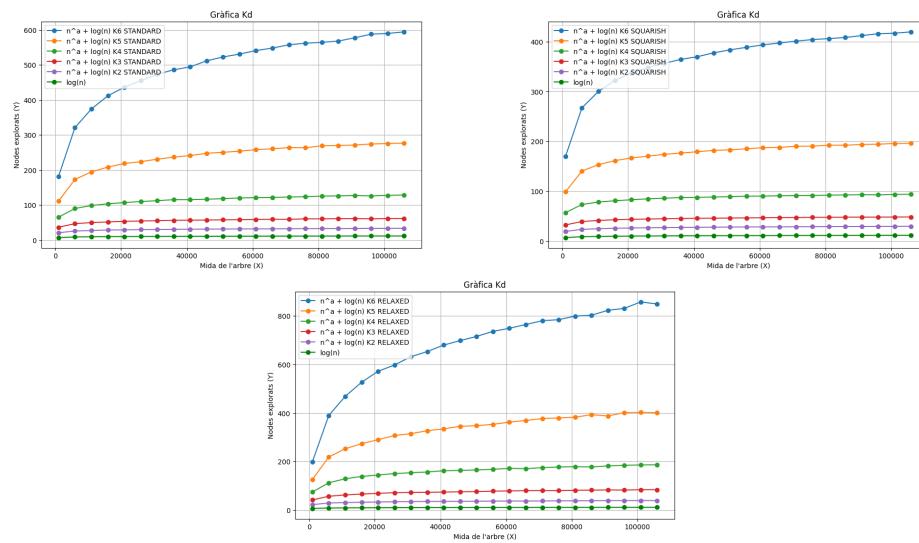


Figura 4.1: Cost mitja en funció d' n pels mètodes *Standard*, *Squarish* i *Relaxed* per a $k \in [2, 4, 5, 6]$.

L'observació inicial - i més rellevant - de les gràfiques obtingudes és el comportament logarítmic del cost (els nodes visitats) a mida que creix n , de forma que es pot trobar consistència entre aquesta tendència i la hipòtesi inicial del projecte, que n'indicava $\log n$ com el terme predominant de la complexitat mitjana en les cerques per veí més proper. Com es pot observar a la figura 4.1, els mètodes *standard* i *squarish* presenten tendències molt semblants, i al *relaxed* s'aprecien valors de ζ més alts, degut a la presència significativa del terme n^ζ , de forma que per a $k = 6$ es comença a apreciar una tendència sub-lineal.

4.3 Estimació del paràmetre ζ

Per a dur a terme la estimació experimental del valor ζ s'ha fet ús del mètode de mínims quadrats: una tècnica estadística que permet ajustar un model a un conjunt de dades. La idea és trobar la línia que minimitza la suma dels quadrats de les diferències entre les observacions reals i les observacions que prediu el model, llavors l'objectiu és minimitzar la següent fórmula:

$$\min_{\zeta, b, c, d} \sum_{i=1}^n \left(y_i - \left(cn_i^\zeta + b \log(n_i) + d \right) \right)^2 \quad (4.1)$$

on ζ és el paràmetre que s'està buscant, n_i és el número de nodes i y_i el cost real. Encara que els paràmetres b , c i d no es tenen en compte asymptòticament i són desconeguts, s'afegeix a la fórmula per millorar l'ajust.

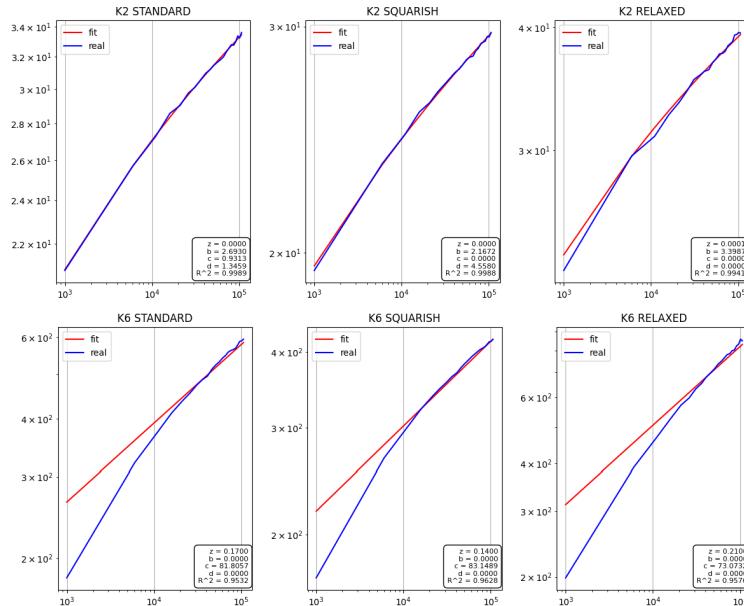


Figura 4.2: Estimació de ζ (al gràfic com a z) pels mètodes *Standard*, *Squarish* i *Relaxed* per a $k \in [2, 6]$.

Com es pot veure a la figura 4.2, el *squarish* és el mètode que té millor cost, seguit del *standard* i finalment del *relaxed*, per a dimensions $k < 6$ el *squarish* té un cost aproximat $\Theta(\log(n))$ en el cas mitjà, pel fet que ζ esdevé negligible. S'observa aquest patró amb el *k-d tree* estàndard per a $k < 5$ ja que $\zeta \approx 0$. El mètode de discriminació *relaxed* és el que dona pitjors resultats amb un cost sublineal n^ζ amb $\zeta \approx 0.21$.

El valor ζ obtingut és major als valors teòrics, això es pot atribuir a que la selecció dels paràmetres es fa respecte del coeficient de determinació R^2 , i al estar les mostres concentrades en un rang restringit de valors de n , poden no ser suficientment representatives de tot l'espai de mostres possibles, la qual cosa permetria un ajust més acurat de les dades i per tant una estimació més precisa de ζ .

Per abordar aquesta qüestió, en lloc de seleccionar el millor coeficient de determinació R^2 , es considera l'existència d'un error sistemàtic α . Això implica que enlloc de buscar maximitzar R^2 , es cerca el màxim $R^2 - \alpha$ sota la condició d'obtenir un valor ζ el més petit possible, on α representa l'error acceptable, que s'ha definit com $\alpha \leq 0.05$. En altres termes, s'estableix com a requisit que R^2 sigui com a mínim igual a 0.95. A la figura C.4 es pot veure el resultat per a totes les dimensions i mètodes i també es pot observar que totes les gràfiques satisfan un $R^2 \geq 0.95$.

Conclusions

L'algorisme de cerca del veí més proper sobre arbres k -dimensionals és un problema el cost del qual encara és objecte d'investigació experimental. En aquest context, s'han examinat tres tipus d'arbres: els arbres k -dimensionals *standard*, *relaxed* i *squarish*. Les conclusions més rellevants que es poden extreure dels experiments són les següents:

- L'algorithm de cerca del veí més proper sobre arbres k -dimensionals demostra empíricament ser significativament més eficient que l'opció de cerca lineal per a tots els mètodes estudiats. Això es manifesta mitjançant una reducció significativa del nombre de nodes visitats, la qual cosa es tradueix en un cost inferior. Aquesta millora es satisfà també en el cas pitjor, que es dona per als k - d trees *relaxed*, els quals permeten observar experimentalment la predominància del component sublineal del cost.
- En l'algorithm de cerca, el k - d arbre *standard* és més òptim que l'arbre k - d *relaxed*, i l'arbre k - d *squarish* presenta un valor de ζ més petit que el k - d arbre *standard*, la diferència entre aquests darrers s'observa en dimensions $k \geq 5$. Es teoritza que això és causat perquè, a mesura que augmenta la dimensionalitat, els punts de dades presenten més dispersió. L'arbre k - d *squarish* manté una major eficiència en dimensions més altes perquè aborda explícitament aquest repte donat que la seva construcció és guiada per la minimització del volum de les caixes delimitadores.

Es poden apreciar aquestes diferències en les gràfiques de la figura 4.1, on s'observa com les cotes de l'algorithm en el k - d arbre *squarish* són inferiors a les del k - d arbre *standard*. Es pot apreciar amb més detall a les figures C.1 C.2 C.3, on aquestes cotes són significativament inferiors a les del k - d arbre *relaxed*. A la figura 4.2, es veu com el valor z , la pendent de la recta, satisfà les relacions hipotetitzades entre els diferents tipus d'arbre. A la figura C.4 es mostren els diferents valors obtinguts per a ζ per a cada dimensió k i mètode.

- El conjunt d'anàlisis i resultats ens permet confirmar que la variable ζ té una forta relació amb la dimensió on es treballa i el tipus d'arbre que s'utilitza. A la figura 4.1 es pot observar com el cost augmenta en funció del valor de k .

Aquestes observacions reforcen la hipòtesi establerta per l'equació del cost [1.2]. Aquesta sosté que el cost de l'algorithm presenta un comportament sublineal i fins i tot es pot aproximar a un cost logarítmic en determinades circumstàncies.

A l'equació del cost es poden afegir unes constants b , c i d que ajuden amb l'aproximació del cost real. La importància d'afegir aquestes constants es pot veure reflectida a la figura 4.2, ja que la recta s'acosta més a la funció experimental que representa el cost. L'equació de la recta es pot veure implícita a l'equació 4.1.

Descripció i valoració del procès d'autoaprenetatge

6.1 Metodologia

El procediment emprat per a la realització d'aquest projecte es va basar principalment en la paral·leització del màxim de tasques possibles entre els membres del grup, per tal d'agilitzar l'elaboració de tasques atòmiques, conscient del pes en temps i complexitat del procés d'unificació dels diferents mòduls del projecte.

La duració total del projecte era de tres setmanes i, de la part tècnica, en poden anomenar tres subtasques distingides: la creació de la classe k - d tree, la implementació d'un algorisme per fer la cerca del veí més proper sobre k - d trees i l'experimentació. La segona part del projecte ha consistit en l'elaboració de la documentació de tot aquest procés.

Amb tal d'adaptar el procés de desenvolupament al temps disponible, es van establir dates límit per a cada part, cadascuna amb una duració d'una setmana. Cada setmana es realitzava una reunió presencial dels quatre membres amb tal de posar en comú tant els avenços com els dubtes, i basant-se en aquests repartir les tasques que prossegueixen. A partir d'aquest punt, només s'establien més reunions davant l'aparició de dificultats o problemes que impiden l'avenç del projecte en algun dels mòduls. També vam assistir a totes les sessions de dubtes amb tal d'obtenir una comprensió més rigorosa dels requisits del projecte, així com comprovar que l'enfocament pres fins al moment estava correctament orientat. A més de les reunions presencials, vam crear un servidor en una plataforma anomenada *Discord* amb tal de poder comentar dubtes i presentar resultats parciais del que anàvem avançant. Aquesta plataforma ens facilitava també fer trucades de veu quan necessitavem consultar quelcom amb els companys o quan havíem de treballar junts. A continuació descrivim l'aplicació del mètode proposat i la repartició de tasques.

En la fase inicial la tasca central va ser comprendre bé l'enunciat i identificar les tasques que genera. Aquesta identificació dels mòduls va ser primordial per a la generació de les tasques inicials que ens permetrien començar la

implementació. Vam decidir que utilitzaríem el llenguatge C++ per programar, a causa de la familiaritat de tots els membres del grup amb aquest i les optimitzacions que permet. A mesura que el projecte avançava, la repartició dels mòduls entre diferents components es tornava borrosa, degut a una col·laboració més conjunta en cada tasca. La divisió inicial va ser tal que un component es va encarregar d'implementar la classe *k-d tree*, un altre de programar un generador de *k-d trees* aleatoris donats els paràmetres k , N i T ; una tercera persona va investigar i portar a terme la implementació de l'algorisme del *nearest neighbour* i per últim, amb l'objectiu principal de poder implementar els *squarish k-d trees*, es va assignar la implementació de la definició i actualització de les *bounding boxes*.

6.1.1 Construcció de proves

La següent fase va consistir principalment en la verificació, tant en l'àmbit experimental mitjançant *tests*, com a nivell conceptual mitjançant demostracions, que els algorismes compleixen amb els seus objectius. Dos components es van centrar en la construcció de tests més complexos per a verificar el funcionament adequat de la generació d'arbres, la inserció de claus en aquests, i per últim el resultat de les cerques del veí més proper. La resta del grup va començar la documentació dels mòduls de codi i les demostracions de la correctesa dels algorismes codificats.

Els tests es van portar a terme mitjançant *Google Test*, un marc de proves de C++ que permet als desenvolupadors crear i executar proves unitàries pel seu codi.

En primer lloc, vam escriure les proves fent servir el marc de *Google Test*, on s'organitzen en conjunts de proves que poden compartir dades i subroutines. La forma de provar una classe és mitjançant la construcció d'affirmacions o assercions sobre el seu comportament. En les nostres proves hem decidit utilitzar les assercions *EXPECT_** donat que aquestes no avorten la funció actual davant una falla sinó que ens permeten informar més d'un cas d'error.

Amb tal d'assegurar la qualitat i la correcció del codi en el projecte allotjat a *GitHub*, vam integrar *Google Test* mitjançant la creació d'un flux de treball de les Accions de *GitHub* [3]. Aquest està definit en un fitxer YAML dins del directori *.github/workflows* del repositori. Aquest flux de treball descriu quan i com s'han d'executar les proves: en el nostre cas, en cada pujada de codi a la branca principal.

Mitjançant la integració de *Google Test* en una canalització de *CI/CD*, assegurem que les proves s'executen automàticament amb cada modificació del codi.

Per últim, la part d'experimentació i documentació es va repartir de forma que la meitat del grup es va encarregar de generar les gràfiques corresponents; i l'altra de documentar el projecte, amb ajuda d'un bon estudi bibliogràfic per tal de donar suport a la nostra comprensió i els coneixements adquirits sobre la temàtica, així com l'ús del pseudocodi per presentar els algorismes dissenyats.

L'etapa experimental es va enfocar en la creació del programa *generadorKdTree* per a la generació dels arbres amb què alimentarem la següent fase de construcció dels experiments. Aquest genera un arxiu que conté T arbres amb n nodes que emmagatzemen punts k -dimensionals. En aquesta fase també es generen tants *query points* com el nombre de *queries* Q que es passa com a paràmetre a la funció.

En la següent fase, el programa *queryKdTree* processa cadascun dels arbres i els punts de consulta generats segons el mètode (*Standard*, *Squarish* o *Relaxed*), indicat com a paràmetre del programa. Es construeixen els respectius *kdTrees*, s'insereix cada node i després s'executen les Q *queries* amb els punts de consulta que es generen a la primera fase.

S'han implementat dos programes addicionals en el llenguatge de programació Python amb l'objectiu de generar de manera massiva totes les mostres necessàries per al nostre experiment: *generadorArbres.py* i *generadorQueries.py*. El generador d'arbres està parametritzat de forma que s'especifica l'interval de nodes pel qual es generaran arbres, amb increments de 5.000 nodes. Per a cada valor d' n , crida al *generadorKdTree* amb el corresponent nombre de nodes, el nombre d'arbres T , la dimensió k i el nombre de consultes Q a executar sobre cadascun dels arbres. Aquestes dades s'emmagatzemen a una carpeta *inputs* de la qual el *generadorQueries.py* llegeix cada subdirector Ki amb $i \in [2, 3, 4, 5, 6]$ i genera els resultats a la carpeta *results*, cada subdirector de la qual correspon a una dimensió Ki amb $i \in [2, 3, 4, 5, 6]$, en cadascun trobem carpetes dedicades als resultats de cada mètode, i a l'interior hi figuren els resultats de les consultes pel veí més proper realitzades.

Un cop obtinguts els resultats, s'ha utilitzat l'eina de *Jupyter Notebook*, una aplicació que permet la creació de documents amb codi en viu, equacions, visualitzacions i text explicatiu. Dins d'aquest entorn, s'han emprat scripts en Python per carregar els resultats i realitzar el processament final.

Primer es carreguen els resultats i s'emmagatzemen a una estructura de dades utilitzant una llibreria de Python anomenada *Panda*, després es poden generar molt fàcilment gràfiques que permeten visualitzar les dades.

Finalment per a l'estimació de ζ s'utilitza la llibreria de *Scipy*, concretament el mètode *curve_fit* [4]. Aquest mètode és dissenyat per ajustar una funció a les dades experimentals i proporciona com a resultat els paràmetres que proporcionen un ajust òptim.

Ens vam trobar, com ja s'esmenta a l'apartat d'experimentació, que s'obtenen valors de ζ molt grans respecte als teòrics. Aquesta discrepància es pot atribuir a la limitació de la grandària de la mostra, la qual contribueix a una major variabilitat en les dades experimentals.

Per abordar aquesta situació, s'ha utilitzat el mètode *curve_fit*, que permet imposar una restricció al paràmetre ζ per evitar valors excessivament alts. Aquesta restricció ve a costa d'una disminució en el coeficient de determinació R^2 . Es parteix d'un valor $\zeta = 0$ i s'incrementa en centèssimes fins que s'obté un $R^2 \geq 0.95$.

Per últim, en paral·lel a la fase de desenvolupament dels experiments, tots els membres del grup ens vam dedicar a completar la documentació del projecte.

6.2 Autoaprenentatge

Aquest projecte ens ha permès assolir i consolidar diversos coneixements, doncs una gran majoria de les tasques a desenvolupar suposaven un component significatiu d'aprenentatge autònom. També, cal destacar que el grau de llibertat a la hora d'establir una metodologia de treball, i conseqüentment la presa de decisions sobre la planificació i distribució de tasques, ha estat fonamental per entendre com es relacionen els diferents mòduls del projecte i la seva dimensió tant temporal com conceptual. Considerem aquest punt important ja que es pot extrapolar i aplicar a una àmplia gamma de projectes similars.

Primerament, cal destacar l'aprenentatge derivat de la recerca i adquisició d'informació, per exemple per entendre el funcionament dels arbres relaxats i quadrats. Aquest últim en particular ens va permetre, i obligar, a consolidar les idees intuïties que n'havíem format sobre el funcionament intern dels *k-d trees*, de forma que mitjançant una formalització d'aquestes idees pogués esdevenir una implementació correcta dels algorismes implicats. No obstant això, l'aprenentatge més extens s'ha produït als punts de desenvolupament on els resultats obtinguts diferien dels esperats. En aquests moments, la necessitat de consultar diverses fonts, com articles científics i documentació externa, es va fer més evident i important.

Durant el transcurs del projecte, l'aspecte estadístic va guanyar progressivament una importància substancial.

El desenvolupament del projecte ha afavorit una revisió i aplicació de conceptes estadístics fonamentals. El més rellevant ha estat el *mètode de mínims quadrats* [5], que es presenta com una tècnica intrínsecament relacionada amb les línies de regressió i la construcció de models òptims per ajustar-se a les dades observades. Aquest mètode estadístic té com a objectiu determinar els paràmetres d'un model matemàtic de manera que es minimitzi la suma dels quadrats de les discrepàncies entre les observacions reals i les prediccions del model.

Aquest concepte, fonamental en l'anàlisi de regressió, ens ha permès trobar els paràmetres que defineixen una línia de regressió de manera que aquesta s'ajusti de la millor manera possible als punts de dades observats.

La col·laboració en equip ha representat sens dubte un aspecte fonamental d'aprenentatge, ja que hem treballat una competència personal de gran utilitat en l'àmbit professional: la capacitat de treballar de forma cooperativa amb altres individus. A més, aquesta experiència ha contribuït al desenvolupament de les competències relacionades amb el lideratge i la distribució de tasques, ja que nosaltres mateixos assumíem els rols de directors del projecte durant la seva elaboració.

També, en el context de la codificació en C++ dels algorismes, hem cobert una àmplia gamma de conceptes, des de qüestions bàsiques com la comprensió de les funcions *rand()* i *srand()* en la generació d'arbres fins a optimitzacions d'eficiència com l'ús de càculs de distàncies en quadrat en lloc d'arrels quadrades per evitar les crides a *sqrt*, o la utilització de *shared pointers*, que ofereixen una solució elegant als problemes de gestió de memòria. A diferència dels punters tradicionals, els *shared pointers* permeten una gestió automàtica de l'alliberament de memòria.

Per últim, va ser necessari familiaritzar-nos amb diverses llibreries de python per al desenvolupament dels experiments, així com la llibreria *gtest* de C++, per escriure les proves i poder assegurar i mantenir la correctesa de la construcció dels arbres i de la cerca.

Apèndixs

Pseudocodi de les funcions d'inserció

Algorithm 2 *i_insert*

Paràmetres: curr, info, depth, type, Bb

Require: depth = *depth(curr)* \wedge type \in (Standard, Relaxed, Squarish) \wedge Bb = *BoundingBox(curr)*

```

1: if curr = NULL then
2:   curr  $\leftarrow$  create a new Node with info and nullptr left and right pointers
3:   if type = Standard then
4:     disc_axis  $\leftarrow$  depth % k
5:   else if type = Relaxed then
6:     disc_axis  $\leftarrow$  rand() % k
7:   else if type = Squarish then
8:     dist  $\leftarrow$  0
9:     dnt  $\leftarrow$  0
10:    disc_axis  $\leftarrow$  0
11:    for i  $\leftarrow$  0 to k do
12:      dist  $\leftarrow$  Bb.maxPoint[i] - Bb.minPoint[i]
13:      if dist > dnt then
14:        dnt  $\leftarrow$  dist
15:        disc_axis  $\leftarrow$  i
16:      end if
17:    end for
18:  end if
19:  curr.dscr  $\leftarrow$  disc_axis
20:  return curr
21: end if
22:
23: if info[curr.dscr] < curr.x[curr.dscr] then
24:   Bb.maxPoint[curr.dscr]  $\leftarrow$  curr.x[curr.dscr]
25:   curr.left  $\leftarrow$  i_insert(curr.left, info, depth + 1, tipo, Bb)
26: else
27:   Bb.minPoint[curr.dscr]  $\leftarrow$  curr.x[curr.dscr]
28:   curr.right  $\leftarrow$  i_insert(curr.right, info, depth + 1, tipo, Bb)
29: end if
30: return curr

```

Algorithm 3 *insert*

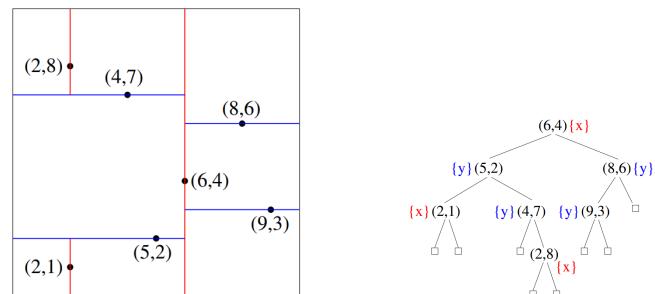
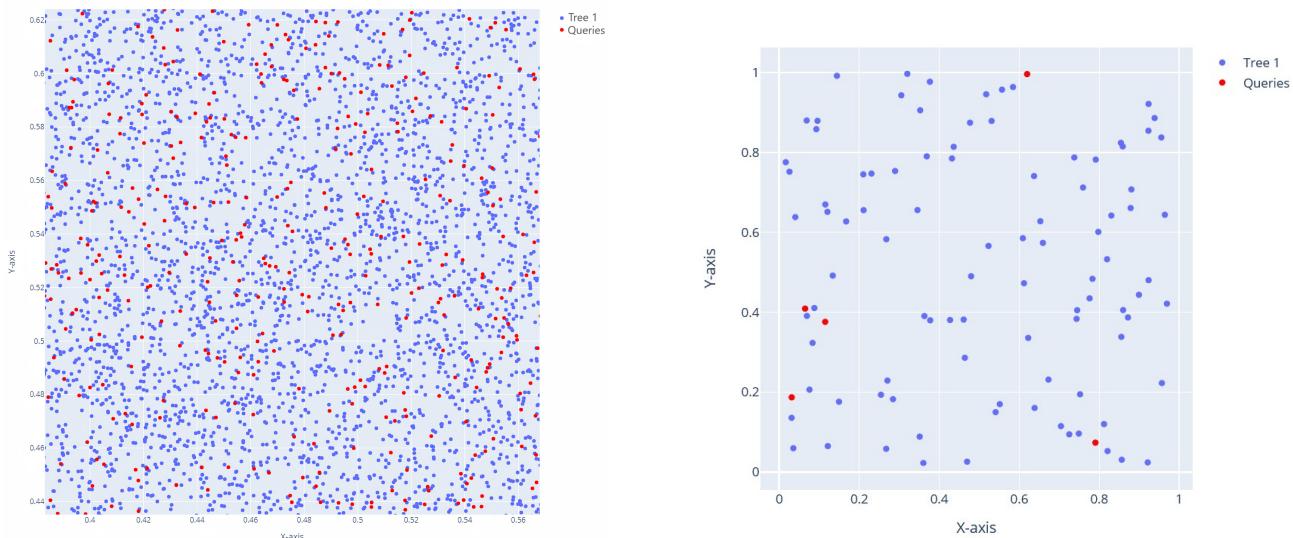
Parameters: info, tipo

```

1: Bb
2: for i  $\leftarrow$  0 to k do
3:   Bb.minPoint[i]  $\leftarrow$  0.0
4:   Bb.maxPoint[i]  $\leftarrow$  1.0
5: end for
6: Increment n                                 $\triangleright$  Incrementa el nombre de nodes de l'arbre en 1
7: root  $\leftarrow$  i_insert(this  $\rightarrow$  root, info, 0, tipo, Bb)           $\triangleright$  Inserta la clau info a l'arbre amb node arrel root

```

Figures

Figura B.1: Partició del pla $[0, 10] \times [0, 10]$ Figura B.2: Representacions 2D d'arbres k -d i el conjunt de *queries* aplicades

Gràfiques

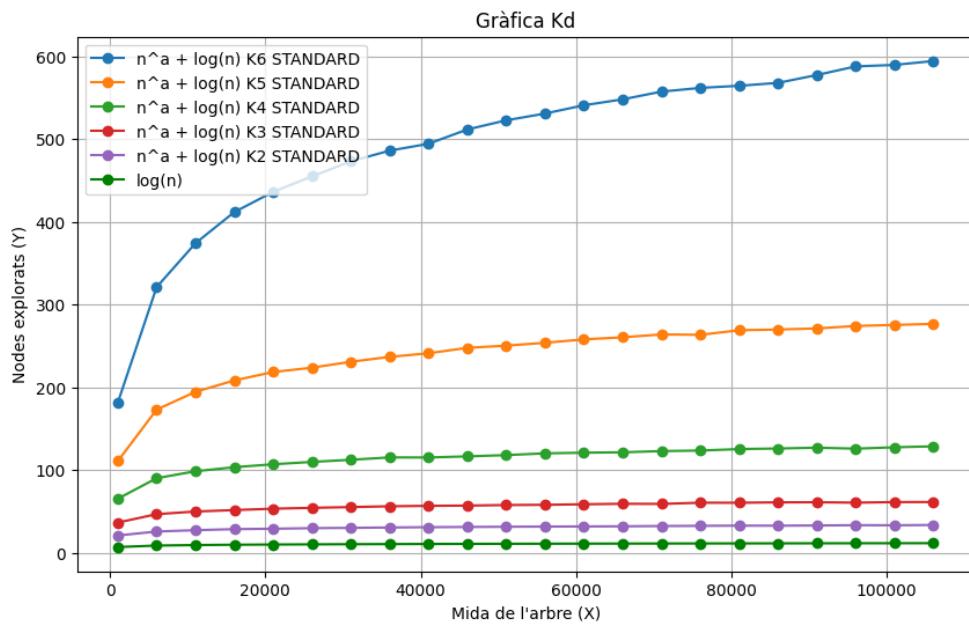


Figura C.1: Cost mitja en funció d' n pel metode *Standard* per a $k \in [2, 4, 5, 6]$.

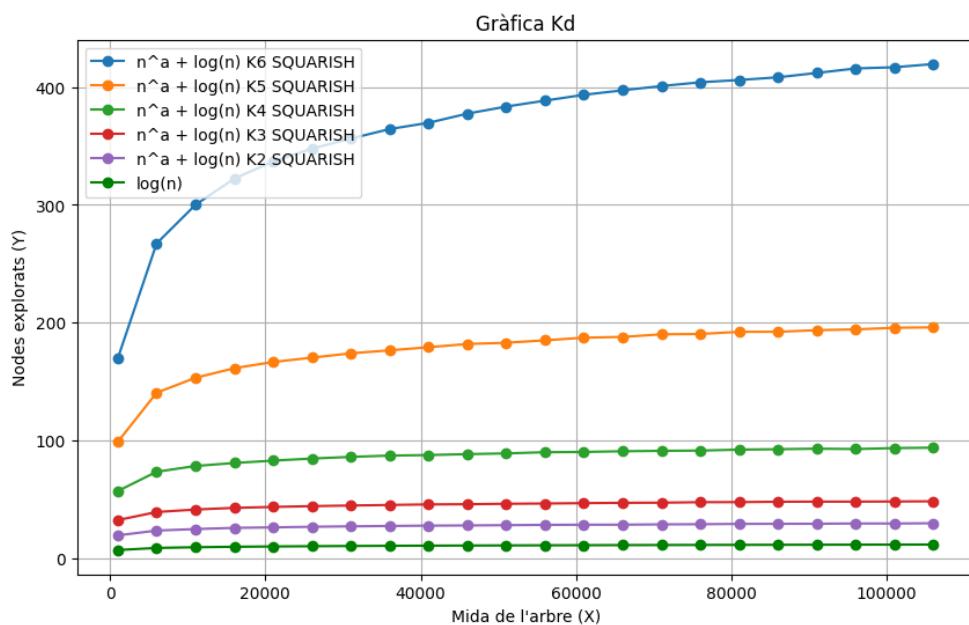


Figura C.2: Cost mitja en funció d' n pel metode *Squarish* per a $k \in [2, 4, 5, 6]$.

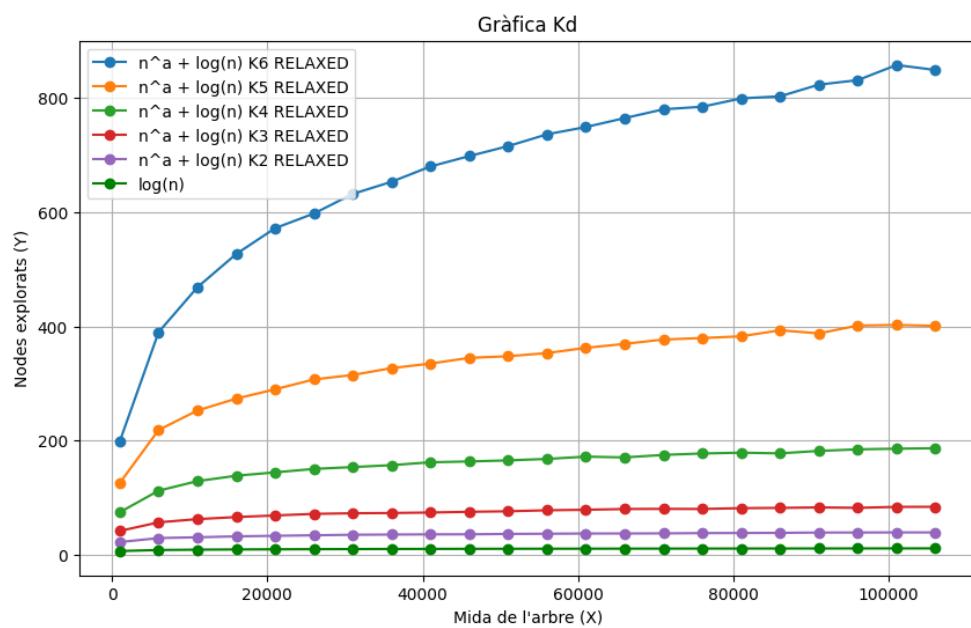


Figura C.3: Cost mitja en funció d' n pel metode *Relaxed* per a $k \in [2, 4, 5, 6]$.

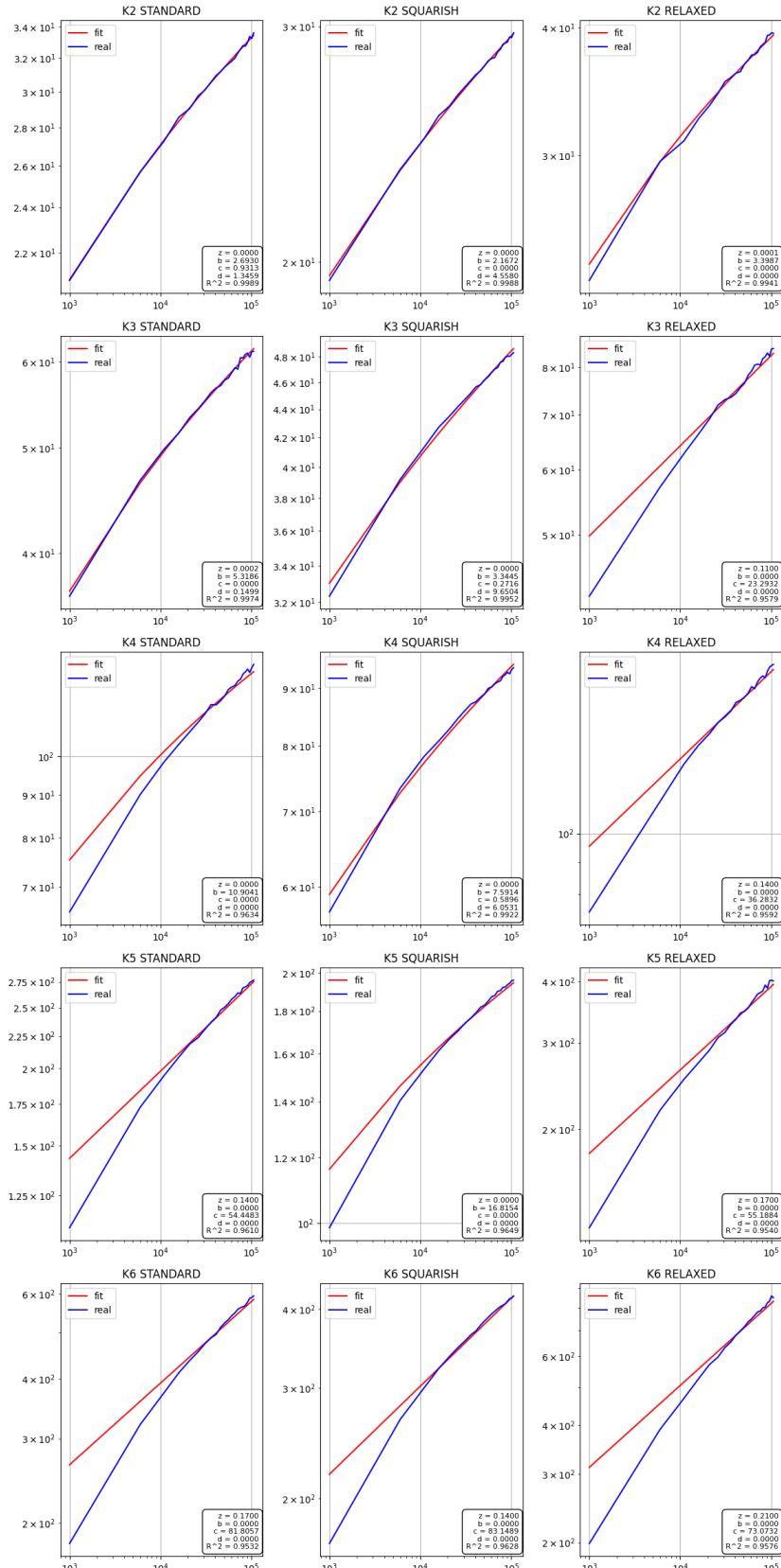


Figura C.4: Estimació de l' ζ (al gràfic com a z) pels mètodes *Standard*, *Squarish* i *Relaxed* per a $k \in [2, 3, 4, 5, 6]$.

Bibliografia

- [1] Bentley, J.L., Friedman J.H.: *Data structures for range searching.* ACM Computing Surveys (CSUR), 11(4):397–409, 1979.
- [2] Friedman, Jerome H., Jon Louis Bentley i Raphael Ari Finkel: *An Algorithm for Finding Best Matches in Logarithmic Expected Time.* ACM Trans. Math. Softw., 3(3):209–226, sep 1977, ISSN 0098-3500. <https://doi.org/10.1145/355744.355745>.
- [3] *Eventos que desencadenan flujos de trabajo - Documentación de GitHub.* <https://docs.github.com/es/actions/using-workflows/events-that-trigger-workflows>.
- [4] *Scipy.optimize.curve_fit — SciPy V1.11.3 Manual.* https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html.
- [5] Miller, Steven J.: *The Method of Least Squares.* https://web.williams.edu/Mathematics/sjmiller/public_html/BrownClasses/54/handouts/MethodLeastSquares.pdf.
- [6] Pons Crespo, Maria Mercè: *Design, Analysis and Implementation of New Variants of Kd-trees.* Tesi de Doctorat, UPC, Facultat d'Informàtica de Barcelona, Departament de Llenguatges i Sistemes Informàtics, Sep 2010. <http://hdl.handle.net/2099.1/11309>.
- [7] google: *GitHub - Google/GoogleTest: GoogleTest - Google Testing and Mocking Framework.* <https://github.com/google/googletest>.
- [8] Samet, Hanan: *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, Massachusetts, 1990.
- [9] Chen, Yewang, Lida Zhou, Yi Tang, Jai Puneet Singh, Nizar Bouguila, Cheng Wang, Huazhen Wang i Jixiang Du: *Fast neighbor search by using revised k-d tree.* Information Sciences, 472:145–162, 2019, ISSN 0020-0255. <https://www.sciencedirect.com/science/article/pii/S0020025518307126>.
- [10] Sridharan, Ramesh: *Linear Regression - Statistics for Research Projects.* <https://www.mit.edu/~6.s085/notes/lecture3.pdf>, 2013.
- [11] Bejar, Javier: *Regresión lineal.* <https://www.cs.upc.edu/~bejar/apa/teoria/APA3a-LinearRegression.pdf>, 2022.