Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Tesi di laurea

Algoritmi per la ricerca per somiglianza in dati a bassa dimensionalità

Relatore:
Prof. Francesco Silvestri

Laureando:
Yi Jian Qiu

Anno Accademico 2021/2022

Indice

Sommario

Il problema del Near Neighbor (NN) è uno dei problemi più studiati nell'analisi dei dati, infatti negli ultimi decenni si è visto una enorme crescita nella ricerca di soluzioni che possano risolvere efficientemente il problema o in modo approssimato. Dato un insieme di dati rappresentati nello spazio e definito uno tra essi, la questione consiste nel individuare un dato che abbia una certa somiglianza al dato iniziale, ovvero trovare un dato che sia fisicamente vicino all'interno della rappresentazione.

La ricerca per somiglianza risulta un problema alquanto interessante in quanto le diverse soluzioni vengono adoperate in molti settori. Infatti, il problema è soggetto di continuo studio poiché le soluzioni fin'ora trovate si dividono in due gruppi, bassa ed alta dimensionalità. In particolare, si sta cercando di trovare un algoritmo che possa unire le soluzioni delle due categorie.

Questa tesi si concentra sul problema della bassa dimensionalità e si illustrano alcune soluzioni adoperate tutt'oggi.

Capitolo 1

Introduzione

1.1 Near Neighbor Search

Il Near Neighbor Search (NNS), è un problema di ottimizzazione in cui i dati possono essere rappresentati come punti all'interno dello spazio e consiste nel trovare un elemento, dato un insieme di punti, più vicino ad un determinato punto chiamato anche *query point*. In modo più formale il problema può essere definito come: dato un set di punti S in uno spazio metrico[1] di dimensione d e una query point q, trovare il punto p vicino a q all'interno di S. Il concetto di vicino può essere molto vago: in questa tesi, si definiscono vicini due elementi se la loro distanza euclidea è minore di un particolare valore r passato all'algoritmo come parametro.

Questo problema ha portato alla realizzazione di diverse varianti di algoritmi in quanto è utilizzato in molti campi, dall'intelligenza artificiale al campo medico, come ad esempio il riconoscimento dei pattern, computer vision, sequenziamento del DNA, l'online dating o analisi dei cluster.

Sfortunatamente, l'NNS soffre della maledizione della dimensionalità (*curse of dimensionality*). Il problema si presenta quando si decide di incrementare la dimensione dello spazio in cui si sta lavorando, da questo incremento ne consegue una dispersione dei dati che non consente di lavorare correttamente, quindi bisogna aumentare il numero dei dati sottoposti allo studio in maniera esponenziale, in modo da avere dei risultati affidabili. Dato questo motivo si possono suddividere in due macrocategorie, in base alla dimensionalità dello spazio, gli algoritmi per il Near Neighbor Search, ovvero a basse e

[1] uno spazio metrico è un insieme non vuoto di elementi in cui è stata definita una metrica, nota anche come distanza (in questo documento viene utilizzato lo spazio euclideo e quindi la distanza euclidea).

alte dimensioni. Questa tesi si concentrerà su alcuni degli algoritmi a basse dimensioni, precisamente verrà descritta la struttura dati utilizzata e poi verrà spiegata la modalità di ricerca.

Nel corso degli anni sono stati individuate diverse soluzioni che variano principalmente sulla struttura dati utilizzata, perché la qualità delle varie configurazioni cambia. La qualità di un determinato algoritmo è stabilito in primo luogo dal tempo di esecuzione delle interrogazioni, chiamate anche query, e in secondo luogo dallo spazio che viene utilizzato per memorizzare la struttura.

1.2 NNS a basse dimensioni

In questa tesi si presentano alcuni algoritmi utilizzati per la risoluzione del problema quando lo spazio di lavoro è considerato basso, in particolare le soluzioni risponderanno ad una versione esatta e a quella approssimata. I due problemi risultano abbastanza simili, in quanto si basano su strutture dati che derivano dall'albero, però si differenziano nell'individuazione del vicino, infatti, l'algoritmo esatto funziona meglio attraverso la ricerca di un range quadrato, invece la versione approssimata adopera un range circolare.

Nel problema esatto la ricerca avviene attraverso l'individuazione dei punti all'interno di un range quadrato, che viene calcolato partendo da un parametro r, ovvero il raggio della palla circoscritta dal quadrato. Invece, la versione approssimata utilizza il raggio a cui si aggiunge un margine in modo da poter individuare possibili vicini che siano leggermente più distanti dalla query point.

La tesi è strutturata in due parti, la prima si concentra sulla spiegazione dell'algoritmo preciso studiando le strutture dati e le rispettive interrogazioni, differenziandoli in base alla dimensione dello spazio in cui si sta lavorando. La seconda parte si dedica allo studio della versione approssimata, specificando alcune delle strutture dati adoperate e successivamente l'algoritmo utilizzato.

Capitolo 2

Near Neighbor preciso

In questo capitolo verranno trattate le strutture dati e gli algoritmi utilizzati per la ricerca, in particolare verrà adoperato l'albero binario bilanciato per le interrogazioni in una dimensione, il *kd-tree* quando lo spazio avrà una dimensione maggiore o uguale a due, inoltre verrà presentata un'alternativa chiamata *range-tree*, che, come si vedrà, migliorerà alcune caratteristiche del *kd-tree* peggiorandone però altre.

2.1 Ricerca vicino per somiglianza

L'algoritmo esatto sfrutta la ricerca dei punti all'interno di un range lineare, nello spazio monodimensionale, range quadrato, se lo spazio è bidimensionale e cubico o ipercubico se lo spazio è maggiore o uguale a tre. Nel caso in cui la dimensione è uguale a uno, vengono forniti all'algoritmo direttamente gli estremi dell'intervallo in cui cercare il vicino. La situazione risulta più interessante quando lo spazio ha due o più dimensioni, in questi casi, si passa all'algoritmo il raggio r della palla in cui si vuole trovare un vicino (circonferenza se si tratta di uno spazio bidimensionale). In questa spiegazione si definisce il caso in cui la dimensione è due, per le situazioni in cui la dimensione spaziale è maggiore, il ragionamento è analogo solo che si utilizzeranno il cubo e la sfera. Quindi, a partire dal raggio, si definisce la regione quadrata che circoscrive la circonferenza, determinando solitamente il centro del quadrato che equivale al centro della circonferenza e la lunghezza del lato, in modo da poter ricavare gli intervalli in cui cercare i punti. Determinati i punti che si trovano all'interno dell'area quadrata, si controllano le distanza dei punti dalla query point q e si restituisce il primo punto che abbia distanza minore di r, invece, i punti esterni alla circonferenza vengono scartati.

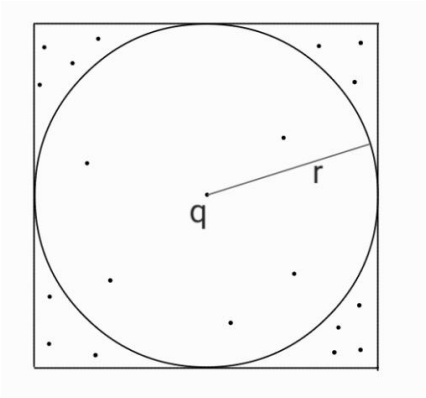

Figura 2.1: Situazione in cui all'interno della circonferenza ci sono punti del set, viene restituito uno qualsiasi.

Nel peggiore dei casi può capitare che all'interno della circonferenza non sia presente nessun punto perché si trovano negli angoli del quadrato, vedi figura 2.2.

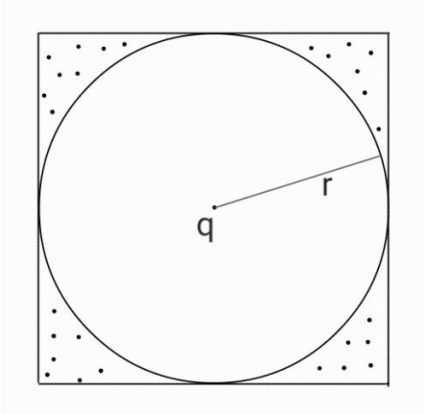

Figura 2.2: Possibile caso in cui non sono presenti punti dentro la circonferenza.

Le ricerche consistono, quindi, nel individuare i punti del set S che si trovano all'interno del range e individuare il primo che rispetti il vincolo sulla distanza da q.

2.2 Algoritmo per interrogazioni in una dimensione

Prima di definire l'algoritmo per la ricerca di nello spazio monodimensionale, si introduce l'albero binario che viene utilizzato per memorizzare i dati.

2.2.1 Albero binario bilanciato per la ricerca

L'albero binario è una particolare struttura dati che risponde alla definizione di "albero ordinato di grado due" con alcune proprietà aggiuntive ovvero:

- I figli possono chiamarsi sinistro o destro.
- In un nodo, il figlio sinistro precede il destro nell'ordinamento posizionale

Solitamente, i nodi dell'albero binario non hanno obbligatoriamente tutti e due i figli, cioè possono avere solo il figlio sinistro o solo quello destro, di conseguenza non è corretto chiamarli come primo o secondo figlio. Si denotano allora, il sottoalbero sinistro e sottoalbero destro, i sottoalberi aventi rispettivamente radice nel figlio sinistro e destro di v. Un albero binario si chiama *proprio* se nessun nodo ha un solo figlio, ovvero tutti i nodi hanno il figlio sinistro e destro, in modo complementare si denota l'albero binario *improprio*, se esso contiene nodi che possiedono un solo figlio. Inoltre, l'albero si dice triangolare se ha il massimo numero di nodi in ciascuno dei suoi livelli, questa condizione consente l'albero di godere di alcune proprietà:

- Se ha altezza h, allora avrà 2^h foglie che si troveranno allo stesso livello h.
- Tra gli alberi di altezza h, quello triangolare ha il massimo numero di foglie e il numero massimo di nodi interni $(2^h - 1)$.
- Tutti i nodi interni hanno due figli, quindi è anche un albero proprio.
- Il livello d avrà al massimo 2^d nodi.

L'albero binario di ricerca, invece, è un particolare albero binario proprio, non vuoto, che possiede ulteriori proprietà:

- Ogni nodo interno v contiene un elemento $x(v)$ che appartiene ad un insieme totalmente ordinato.
- Gli elementi memorizzati nel sottoalbero sinistro del nodo v sono non maggiori di $x(v)$.
- Gli elementi del sottoalbero destro sono non minori di $x(v)$.

Infine, l'albero binario bilanciato di ricerca, come si intuisce dal nome, è una sottocategoria degli alberi bilanciati (sezione 2.2) che però assume le caratteristiche dell'albero binario di ricerca.

2.2.2 Algoritmo di ricerca

Nello spazio monodimensionale, il set di punti possono essere rappresentati dai numeri reali e di conseguenza le interrogazioni consisteranno in un intervallo di numeri, delimitato da due valori [x e x']. Dato un set di punti S sulla retta reale, è possibile realizzare la ricerca di un range di valori utilizzando la struttura dati appena descritta, l'albero binario bilanciato.

Le foglie dell'albero binario T memorizzano i punti di S e i nodi interni, invece, tengono traccia dei valori di divisione utilizzati per orientarsi nel set di punti. Si denota il valore di divisione x_v il valore memorizzato al nodo v, si assume, quindi, che il sottoalbero destro contenga tutti i valori maggiori di x_v, e che il sottoalbero sinistro abbia i punti con valore più piccolo o uguale a x_v. Inoltre, si indicano con μ e μ', , le foglie che delimitano la fine della ricerca all'interno del range. I punti nell'intervallo $[x : x']$ saranno i valori memorizzati tra le due foglie μ e μ' (possono fare parte dell'intervallo). Vedi un possibile esempio nella figura: 2.3.

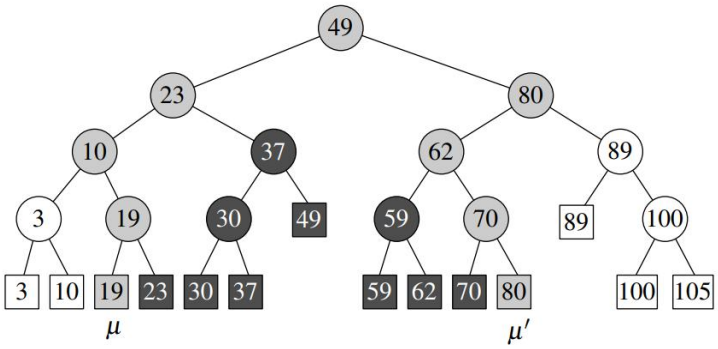

Figura 2.3: Nell'esempio il range richiesto è [18:77] e come si è detto, μ e μ' possono appartenere o no all'intervallo.. Immagine tratta da [1].

Per individuare le foglie interessate, si interrogano le radici dei sottoalberi, in particolare, i sottoalberi che si selezionano hanno radice al nodo v che si trova tra due percorsi di ricerca (i due sottoalberi), i cui genitori fanno parte del percorso. Prima di tutto bisogna trovare il nodo v_{split} dove il percorso per x e x' si divide, quindi partendo dalla radice di T, si controlla che il valore del nodo rientri all'interno dell'intervallo, se non fosse così, allora si assegna come nodo v il figlio sinistro e poi quello destro, si prosegue quindi ricorsivamente in questo modo, finché non si arriva alla foglia o si individua il nodo $v_{split} \in [x : x']$.

A partire dal nodo v_{split}, si segue il percorso di ricerca di x e in ogni nodo in cui si va a sinistra, si riportano tutte le foglie del sottoalbero a destra, perchè esso si troverà

all'interno dell'intervallo richiesto. In modo complementare si riportano le foglie del sottoalbero a sinistra quando, nel percorso di ricerca di x', si seleziona il figlio destro. Alla fine della ricerca, si va a controllare quali foglie appartengano effettivamente al range di valori richiesto, da questo insieme, quindi si seleziona uno dei valori la cui distanza dal punto di interrogazione sia minore di r, così da individuare uno dei vicini come richiesto nel problema del NNS.

L'albero binario di ricerca ci consente di adoperare una memoria $O(n)$ impiegando un tempo $O(n \log n)$ (vedi sezione 3.2.1). Più interessante, invece, è il tempo per le interrogazioni, infatti nel peggiore dei casi i punti richiesti sono tutti i punti del set S che richiederebbe un tempo $\Theta(n)$. Questo risultato potrebbe sembrare negativo ma $\Theta(n)$ non può essere evitato se si vogliono tutti i punti del set, inoltre non sarebbe neanche necessario una struttura dati per la memorizzazione dei punti. Invece, nelle interrogazioni in cui il range è un sottoinsieme dei punti, si indica con k il numero di valori da riportare, quindi servirà un tempo $O(k)$ per leggere tutti i punti nell'intervallo. Inoltre, dato che il tempo speso in ogni nodo è $O(1)$, e i cammini percorsi sono lunghi $O(\log n)$, il tempo totale trascorso nei nodi sarà $O(\log n)$. Quindi il tempo totale per l'interrogazione è $O(\log n + k)$.

Nel problema del NNS, il tempo utilizzato è sempre $O(\log n + k)$ in quanto, dopo aver trovato l'intervallo in cui cercare il range, bisogna scandire tutti i k punti e restituire il primo la cui distanza dal query point è minore di r.

2.3 Ricerca in due dimensioni

Nello spazio a due dimensioni i punti del set P giacciono su un piano e di conseguenza le interrogazioni consistono nel cercare i punti del set in range quadrati $[x : x'] \times [y : y']$ (sarà denominato come *query*). Perciò, un punto $p = (p_x, p_y)$ appartiene al range se e solo se $p_x \in [x : x']$ e $p_y \in [y : y']$. Si può notare che le interrogazioni in due dimensioni corrispondono a due interrogazioni in una dimensione, in particolare una lungo l'asse x e una sull'asse y. Nella ricerca in una dimensione si cercava di dividere il set di punti in due insiemi di circa ugual dimensione, in questo caso, dato che i punti hanno due valori(x e y) si andrà prima a dividere il set prima lungo la coordinata x per poi passare alla divisione della coordinata y.

2.3.1 Cosa è un kd-tree

La struttura dati che verrà utilizzata, per le interrogazioni in due dimensioni, è il *kd-tree* abbreviazione di *k-dimensional tree*, dove k indica la dimensione dell'albero, dunque in questo caso si tratta di un kd-tree a due dimensioni. La costruzione consiste nel dividere, come detto precedentemente, le coordinate in modo alternato utilizzando delle linee verticali e orizzontali che saranno memorizzati nei rispettivi nodi. In particolare, nella radice dell'albero si memorizza la linea verticale l_1 che divide il set S in due sottoinsiemi di simili dimensioni. Successivamente, si denotano con S_s i punti che si trovano alla sinistra della linea, ovvero i punti le cui coordinate x sono minori di l_1, e si etichettano i punti situati a destra della linea con S_d; ovviamente S_s e S_d saranno memorizzati nei rispettivi sottoalberi della radice. A seguire, si dimezza S_s in due sottoinsiemi attraverso una linea orizzontale l_2 e i punti che hanno coordinate y minore di l_2 saranno memorizzati nel sottoalbero sinistro di l_2, invece, i punti che si trovano sopra la linea orizzontale saranno salvati nel sottoalbero destro. Nello stesso modo avviene la divisione di S_d, in cui viene utilizzata una retta orizzontale l_3 e memorizzato i punti al di sopra della riga nel sottoalbero destro e i punti sotto ad essa, nel sottoalbero sinistro. Questo procedimento viene eseguito ricorsivamente finché in tutti i sottoinsiemi saranno presenti un solo punto. Può capitare che le linee di divisione passino proprio su alcuni punti del set, quindi si assume che i punti che giacciono sulle righe verticali appartengano al sottoinsieme sinistro, e i punti che si trovano sulle righe orizzontali facciano parte dell'insieme collocato sotto alla riga.

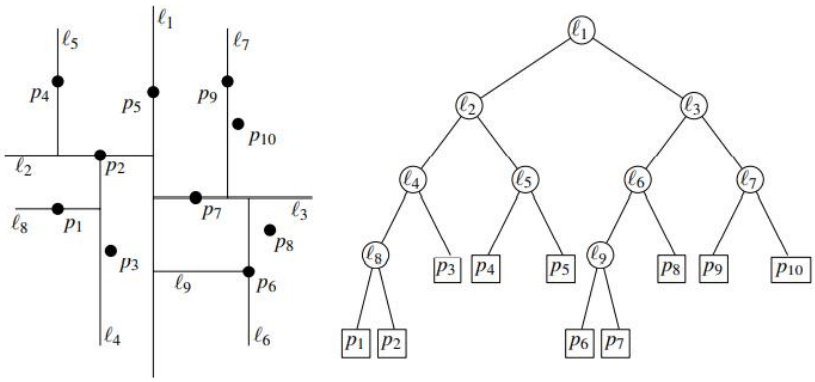

Figura 2.4: Esempio di kd-tree. Immagine tratta da [1].

Nei passaggi precedenti, si utilizza la convenzione in cui la mediana di un set di n debba essere definito come la linea tracciata sul $\lceil n/2 \rceil$-esimo numero più piccolo, questo significa che la media tra due numeri è quello più piccolo.

Il passo che richiede più tempo nella costruzione è la ricerca della linea di divisione in base alla coordinata x o y. Il miglior approccio è quello di riordinare l'insieme dei punti su entrambe le coordinate cosi che il set S venga passato come due liste ordinate. In questo modo, utilizzando le liste, è più semplice trovare la mediana sulle coordinate x e y in un tempo lineare, e il tempo totale richiesto per ordinare i punti quindi è $O(n \log n)$. Passando ora allo spazio occupato, si ricorda che le foglie ed i nodi interni del kd-tree occupano ciascuno $O(1)$ e sapendo che i punti di S vengono memorizzati nelle foglie dell'albero, si afferma, quindi, che lo spazio utilizzato per la struttura dati è $O(n)$.

2.3.2 Interrogazioni sul kd-tree

Dato un set S di n punti, la linea di divisione che taglia il piano verticalmente, viene memorizzata nella radice e il figlio sinistro corrisponde al semipiano che si trova a sinistra della linea, di conseguenza il semipiano a destra viene rappresentato dal figlio destro. Gli altri nodi dell'albero corrispondono anch'essi a regioni del piano. In generale, si definisce $regione(v)$ la regione rettangolare (a volte illimitata su uno o più lati) rappresentata dal nodo v, quindi la regione del nodo radice sarà l'intero piano.

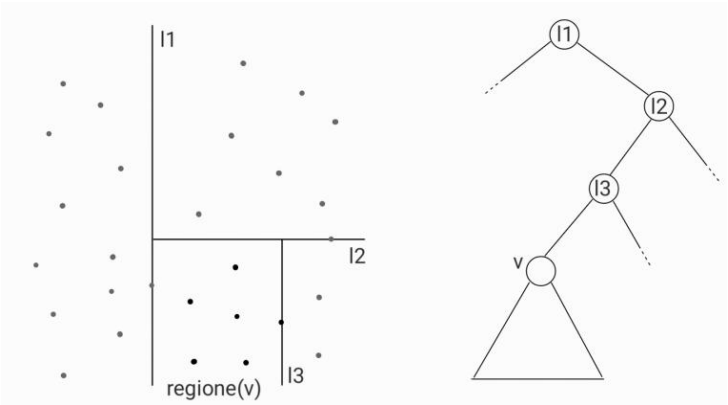

Figura 2.5: Esempio di divisione delle regioni e rappresentazione dei nodi corrispondenti. Immagine tratta da [1].

Si osserva che i punti sono salvati nel sottoalbero τ che ha come radice v se appartengono alla $regione(v)$, di conseguenza, nelle interrogazioni, si visiterà il sottoalbero τ se e solo se il range quadrato interseca l'area della $regione(v)$. Si avranno 3 possibili casi:

1. Se si arriva alle foglie, bisogna controllare se appartengono al range di interrogazione.
2. Se la regione è interamente contenuta nella query, si riportano tutti i punti memorizzati nel sottoalbero.

3. Se la regione interseca parzialmente il range ricercato, si scenderà di un livello l'albero kd-tree.

Al secondo e il terzo punto bisognerà studiare i casi per il sottoalbero destro e sinistro in modo che la chiamata ricorsiva si sviluppi nella regione interessata. L'interrogazione all'interno di un kd-tree, contenente n punti, può essere realizzato in un tempo $O(\sqrt{n}+k)$, dove k è il numero di punti riportati.

Nella ricerca del vicino, dato il parametro r, ovvero la distanza massima in cui bisogna trovare un punto vicino alla query point q, il tempo per la ricerca è uguale al tempo per l'interrogazione del range rettangolare ($O(\sqrt{n}+k)$), in quanto dopo aver trovato i punti da riportare, si controlla che la distanza da q sia minore di r.

2.3.3 Il range-tree, una valida alternativa

L'utilizzo di un kd-tree per le interrogazioni richiede un tempo $O(\sqrt{n}+k)$, perciò quando il numero di punti all'interno del range di interrogazione è piccolo, il tempo di ricerca risulta relativamente alto. Si introduce quindi una nuova struttura dati chiamata *range tree* che fornisce un tempo di interrogazione minore ($O(\log^2 n + k)$), però il prezzo da pagare è l'aumento dello spazio utilizzato che passa da $O(n)$ a $O(n \log n)$.

Sia S un set di n punti sul piano lo si vuole elaborare per le interrogazioni su range quadrati $[x : x'] \times [y : y']$. In primo luogo si trovano i punti che hanno coordinata x all'interno del range $[x : x']$ e questo corrisponde all'interrogazione di un intervallo di valori in una dimensione, come si è visto precedentemente (sezione 3.2) è sufficiente l'utilizzo di un albero binario di ricerca. L'algoritmo richiedeva di trovare il nodo di divisione v_{split}, ovvero il nodo in cui il percorso si divideva. Anche in questo caso, una volta trovato il nodo di divisione, dal figlio sinistro si continua la ricerca con x e ogni volta che la ricerca porta al sottoalbero sinistro, si restituiscono tutti i punti contenuti nel sottoalbero destro. Lo stesso ragionamento vale per il figlio destro di v_{split}, infatti nella ricerca di x' ogni volta che si sceglie il sottolabero destro, vengono restituiti tutti i punti del sinistro. Infine, si controllano i punti μ e μ' per verificare la loro appartenenza all'intervallo. In questo modo si individuano i $O(\log n)$ sottoalberi disgiunti che contengono i punti le cui coordinate x appartengono al range $[x : x']$.

Si denota ora come *sottoinsieme canonico C(v)* del nodo v, il sottoinsieme di punti memorizzati nelle foglie del sottoalbero che ha come radice il nodo v, ad esempio il sottoinsieme canonico della radice dell'albero è l'intero set S e il sottoinsieme canonico di

una foglia è il punto memorizzato in essa. Come è stato visto appena precedentemente, i punti che hanno coordinata x appartenente all'intervallo richiesto, possono essere espressi come l'unione disgiunto di $O(\log n)$ sottoinsiemi canonici. I sottoinsiemi $S(v)$ interessanti saranno quelli in cui i punti hanno coordinata y che appartiene al range $[y : y']$ desiderato. La richiesta può essere tradotta come l'interrogazione in una dimensione, in cui viene utilizzato l'albero binario di ricerca sulle coordinate y dei punti in $S(v)$. Questa premessa consente di introdurre quindi una nuova struttura dati per le ricerche di range rettangolari all'interno di un insieme di n punti sul piano. La struttura dati prende il nome di *range tree* ed è organizzata in questo modo:

- La struttura principale è un albero binario bilanciato di ricerca T costruito in base alle coordinate x dei punti di S.
- Per i nodi interno o le foglie di T, il sottoinsieme canonico $S(v)$ è memorizzato in un albero binario bilanciato di ricerca $T_{assoc}(v)$ (chiamato *struttura associata*) tenendo conto delle coordinate y dei punti. Quindi il nodo v di T contiene un puntatore che indirizza alla radice di $T_{assoc}(v)$.

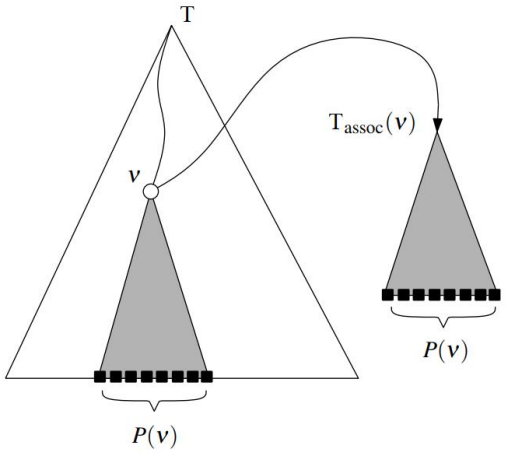

Figura 2.6: Esempio di range-tree. A sinistra l'albero principale, costruito in base alla coordinate x, a destra una delle strutture associate, realizzata secondo le coordinate y. Immagine tratta da [1].

La costruzione del range-tree può essere effettuata attraverso un algoritmo ricorsivo che ha come parametro S ovvero il set di punti ordinati in base alla coordinata x. L'algoritmo può essere riassunto nei tre punti salienti:

1. Se il set S è vuoto, non è necessario costruire l'albero.

2. Se il set S passato come parametro contiene un solo punto, allora si crea una foglia v che memorizza il punto e viene aggiunta alla struttura associata $T_{assoc}(v)$.

3. Altrimenti si divide S in due sottoinsiemi utilizzando la mediana x_{mid}, in particolare S_s e S_d che contengono rispettivamente i punti che hanno coordinata x minore o uguali e maggiori di x_{mid}. Si effettua così la chiamata ricorsive su questi nuovi insiemi, inoltre si crea il nodo v che memorizza x_{mid} e la struttura associata ad essa.

La costruzione di un range-tree, su un set S di n punti, richiede uno spazio $O(n \log n)$ per essere costruito. Infatti, un punto p viene salvato nella struttura associata dei nodi che si trovano sul percorso, ovvero p è memorizzato in esattamente una struttura associata ad ogni livello dell'albero. Dato che il range-tree occupa uno spazio lineare in una dimensione, ne segue che anche le strutture associate utilizzino uno spazio $O(n)$. Perciò lo spazio totale richiesto è proprio $O(n \log n)$.

L'algoritmo per l'interrogazione seleziona prima di tutto i $O(\log n)$ sottoinsiemi canonici che contengono i punti le cui coordinate $x \in [x : x']$, questo può essere realizzato attraverso una ricerca in una dimensione che richiede un tempo $O(\log n)$. Da questi sottoinsiemi si selezionano i punti le cui coordinate $y \in [y : y']$, adoperando nuovamente l'algoritmo per le ricerche monodimensionali sulle strutture associate che contengono i sottoinsiemi canonici, impiegando quindi $O(\log n)$. La ricerca di un range quadrato all'interno del range-tree, quindi, richiede un tempo $O(\log^2 n + k)$ dove k sono i punti restituiti, ovvero i punti contenuti all'interno della regione richiesta.

Nella ricerca del vicino, dato il parametro r, vale la stessa logica del kd-tree quindi il tempo per la ricerca del vicino equivale all'individuare i punti della regione rettangolare all'interno del range-tree, perché per ogni punto trovato si controlla che la distanza di q dal punto riportato sia minore di r.

In conclusione si riassumono quindi i tempi e gli spazi richiesti per le interrogazioni utilizzando le due diverse strutture dati.

	spazio	tempo costruzione	tempo interrogazione
kd-tree	$O(n)$	$O(n \log n)$	$O(\sqrt{n} + k)$
range-tree	$O(n \log n)$	$O(n \log n)$	$O(\log^2 n + k)$

2.4 Interrogazioni in tre o più dimensioni

Le interrogazioni in spazi a 3 dimensioni o maggiori possono essere realizzate utilizzando anche in questi casi il range-tree o il kd-tree. La costruzione delle strutture dati è molto

simile agli alberi adoperati per lo spazio bidimensionale.

2.4.1 Range-tree

Sia S un set di n punti nello spazio di dimensione d, si costruisce l'albero binario bilanciato di ricerca in base alla prima coordinata dei punti. L'albero principale T prende il nome di albero di primo livello e il sottoinsieme canonico $S(v)$ del nodo v dell'albero principale, contiene i punti memorizzati nelle foglie del sottoalbero che ha come radice v. Per ogni nodo v si va a costruire la struttura associata $T_{assoc}(v)$ per i punti $P(v)$ e questo è l'albero di secondo livello che consiste in un range-tree $T1$ di dimensione $(d-1)$ limitato alle $d-1$ coordinate. $T1$ viene costruito in modo ricorsivo nello stesso modo, ovvero è un albero binario bilanciato di ricerca realizzato sulla seconda coordinata dei punti, in cui ciascun nodo punta ad un range-tree $T2$ di dimensione $(d-2)$ ristretto alle $d-2$ coordinate, contenente i punti del sottoalbero.

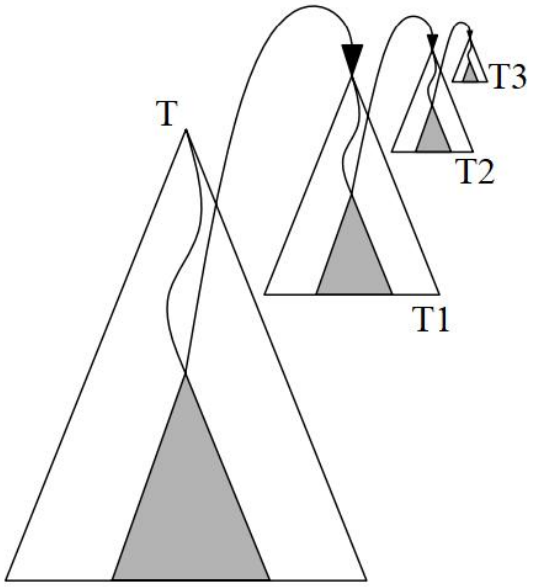

Figura 2.7: Esempio di range-tree in uno spazio di quattro dimensioni. Immagine tratta da [1].

La ricorsione si ferma quando si raggiunge l'ultima coordinata del punto, in questa ultima iterazione, i punti sono memorizzati in un range-tree di dimensione 1.

L'algoritmo per l'interrogazione risulta anch'esso molto simile alla versione bidimenisonale, infatti si utilizza l'albero di primo livello per individuare i $O(\log n)$ nodi i quali sottoinsiemi canonici contengono tutti i punti che hanno la prima coordinata all'interno del

range cercato. Questi sottoinsiemi vengono successivamente interrogati nella corrispondente struttura di secondo livello $T1$. Per ogni albero di secondo livello, si selezionano i $O(\log n)$ sottoinsiemi canonici, da ciò si deduce che al secondo livello ci saranno $O(\log^2 n)$ alberi da analizzare. Essi contengono tutti i punti che hanno la prima e la seconda coordinata all'interno del range richiesto. Gli alberi di terzo livello, che memorizzano a loro volta altri sottoinsiemi canonici, vengono interrogati in modo che la terza coordinata dei punti rientri nel range. Il procedimento ricorsivo continua fino a quando non si giunge all'albero mododimensionale, in questa struttura si trovano i punti la cui ultima coordinata rientra nell'intervallo richiesto e vengono riportati.

Attraverso l'utilizzo del range-tree per la memorizzazione di un set di n punti in uno spazio di d dimensioni, dove $d \geq 2$, l'albero occupa uno spazio $O(n \log^{d-1} n)$ e viene costruito in un tempo $O(n \log^{d-1} n)$. L'algoritmo per la ricerca restituisce i punti all'interno della range in un tempo $O(\log^d n + k)$, dove k è il numero dei punti riportati.

2.4.2 Kd-tree

Il kd-tree costruito sugli n del set S è simile all'albero bidimensionale, la differenza evidente è che bisogna considerare più coordinate per un unico punto. Alla radice, si divide il set in base alla prima coordinata dei punti, ovvero si divide S in due sottoinsiemi di ugual dimensione utilizzando un iperpiano che sia perpendicolare all'asse x_1. Ai figli della radice, la partizione avviene in base alla seconda coordinata, i nodi di profondità due, quindi i figli dei figli, la suddivisione coinvolgerà la terza coordinata e così via. Alla profondità $d-1$ si dividerà il set secondo l'ultima coordinata; l'iterazione successiva, quindi alla profondità d dell'albero, la partizione ripartirà dalla prima coordinata. Questa ricorsione si ferma quando il sottoinsieme conterrà un solo punto che verrà memorizzato all'interno della foglia. Dato che si tratta di un kd-tree in d dimensioni per un set di n punti, l'albero binario avrà n foglie e, di conseguenza, occuperà uno spazio $O(n)$. Il tempo di costruzione invece sarà ancora $O(n \log n)$.

Analogamente allo spazio monodimensionale, anche in questo caso ogni nodo dell'albero corrisponde ad una regione dello spazio e perciò l'algoritmo visita i nodi la cui regione interseca il range ricercato e, inoltre, l'algoritmo attraversa l'intero sottoalbero che hanno come radice i nodi la cui regione è contenuta interamente all'interno della regione interrogata in modo da riportare tutte le foglie, ovvero tutti i suoi punti. Il tempo per l'interrogazione è limitato da $O(n^{1-\frac{1}{d}} + k)$, dove d è la dimensione dello spazio in cui si

sta lavorando e k è il numero dei punti trovati. Come si può notare il limite rappresenta un numero efficiente se la dimensione dello spazio è piccolo (da qui basse dimensioni), infatti con l'aumentare di d, aumenta il valore $\frac{1}{d}$ che tenderà ad essere sempre più piccolo, portando il limite a $O(n+k)$, questo corrisponde a visitare tutti i punti del set risultando non più efficiente nella ricerca del vicino (*curse of dimensionality*).

	spazio	tempo costruzione	tempo interrogazione
kd-tree	$O(n)$	$O(n \log n)$	$O(n^{1-\frac{1}{d}} + k)$
range-tree	$O(n \log^{d-1} n)$	$O(n \log^{d-1} n)$	$O(\log^{d} n + k)$

Sia nell'utilizzo del kd-tree che del range-tree, dopo aver individuato i punti del set che appartengono al range richiesto, si restituisce uno dei punti la cui distanza da q è minore o uguale a r, così da trovare un vicino per il NNS.

Capitolo 3

Near Neighbor approssimato

3.1 Introduzione alle griglie

3.2 L'albero

Prima di definire la struttura dati su cui si poggia questo algoritmo, ovvero il quadtree, bisogna introdurre il modello su cui esso si basa, ovvero l'albero fissando alcune definizioni.

L'albero T è un modello astratto per rappresentare una struttura gerarchica costituita da un insieme di elementi chiamati nodi o vertici, tra loro esiste una relazione gerarchica di tipo genitore/figlio.

Se l'albero non è vuoto, esiste un solo nodo che non ha genitore ovvero il nodo radice, in inglese *root*, invece tutti gli altri nodi hanno ciascuno uno e un solo genitore q, *parent*. I nodi che hanno come genitore q vengono chiamati figli di q (*children*), inoltre i nodi che non hanno figli prendono il nome di foglie (*leaf*). Il collegamento tra genitore e figlio viene effettuato attraverso l'utilizzo di un segmento o arco chiamato ramo, ovviamente un genitore può avere più di un figlio ma un figlio non può avere più genitori. I dati, che possono essere di qualsiasi tipo, sono memorizzati all'interno dei nodi e attraverso i rami si definisce la relazione tra loro.

Dato un albero T, il sottoalbero avente come radice $v \in T$, è l'albero formato da tutti e soli i discendenti di v.

Bisogna ricordare che un albero per poter essere chiamato tale, l'insieme dei suoi nodi e rami deve essere connesso.

Si definiscono, inoltre altri termini che verranno utilizzati successivamente, tra questi si definisce il nodo a come *antenato (ancestor)* del nodo b se e solo se a coinci-

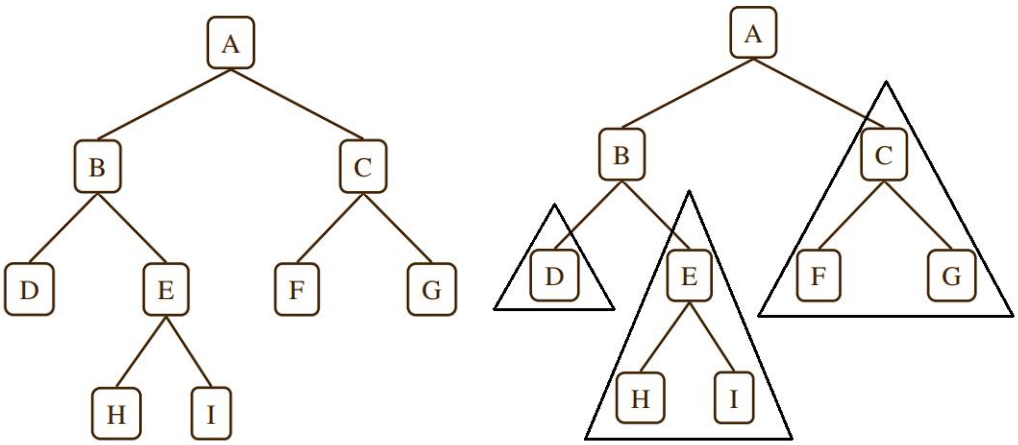

Figura 3.1: Esempio di albero a sinistra con dei possibili sottoalberi a destra.

de con b oppure a è un antenato del genitore di b. Da qui si stabilisce l'acronimo LCA(*Lowest Common Ancestor*) il quale indica il nodo con profondità massima che è antenato dei nodi presi in considerazione.

La profondità di un nodo v è il numero degli antenati di v escluso se stesso, l'altezza invece è il massimo delle lunghezze dei suoi cammini che vanno dalla radice alle sue foglie. Un albero quindi, si dice bilanciato in altezza, se le altezze dei suoi nodi differiscono tra loro al massimo di un'unità.

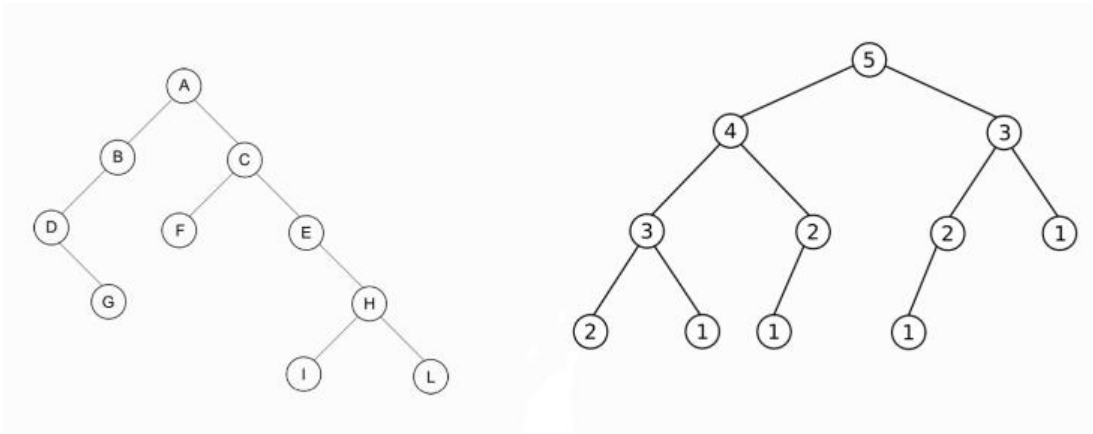

Figura 3.2: A sinistra un esempio di albero non bilanciato, a destra invece un possibile albero bilanciato. Immagini tratte da [**http://alessiorolleri.wikidot.com/albero**].

3.3 Quadtree e le sue varianti

In questa sezione verrà spiegata la struttura che sarà utilizzata per la memorizzazione dei dati chiamata *quadtree*. Prima di procedere alla definizione del quadtree, si specificano

alcuni termini che verranno utilizzati successivamente.

3.3.1 Alcune definizioni

Dato un set di punti S, si suppone che tutti i punti siano racchiusi all'interno di una regione quadrata che sarà il punto di partenza su cui si andrà a sviluppare il quadtree. La regione quadrata unitaria iniziale sarà iterativamente divisa in regioni sempre più piccole che continueranno ad avere la forma quadrata. I quattro vertici di un quadrato si chiamano angoli, i segmenti che collegano i vertici sono chiamati lati, invece prendono il nome di margine i lati dei quadrati che sono contenuti nei lati di regioni più grandi (vedi fig.3.3). Due aree che condividono un margine assumono il nome di vicini(*neighbors*).

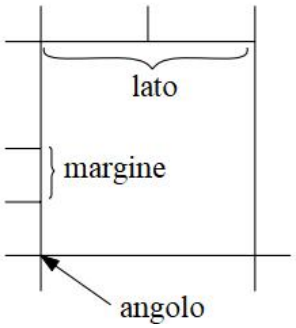

Figura 3.3: Esempi .di lato, margine e angolo. Immagine tratta da [1].

3.3.2 Il quadtree

Sia P una partizione di una unità quadrata e un set di punti S, un quadtree è un albero con una radice in cui ogni nodo interno (tutti i nodi che hanno figli) ha quattro figli. Ogni nodo corrisponde ad un quadrato, ovvero la radice rappresenta il quadrato iniziale che contiene tutti i punti del set S e se un nodo v possiede figli, allora essi corrisponderanno ai quattro quadranti che si formano dividendo la cella con un segmento orizzontale ed uno verticale, tracciati in modo da collegare i punti medi dei lati opposti del quadrato, così da ottenere i quadranti di dimensione equivalente.

Come è stato detto precedentemente, l'albero e in questo caso il quadtree può memorizzare diverse tipologie di dati. Perciò ora si precisa la definizione di quadtree per un set di punti S all'interno di un quadrato unitario C. Si posso avere tre situazioni:

- Se S è vuoto non serve nessun quadtree per rappresentarlo

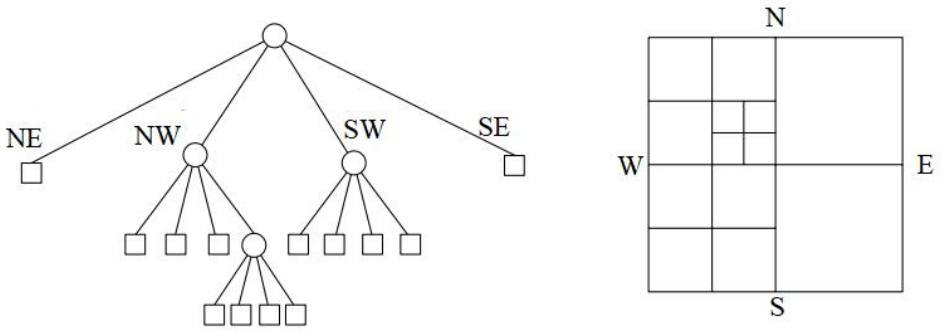

Figura 3.4: Esempio di quadtree e delle possibili suddivisioni. Immagine tratta da [1].

- Se all'interno di S è presente un solo punto, allora il quadtree sarà raffigurato da un solo nodo che sarà anche radice dell'albero.
- Altrimenti, siano I, II, III, IV, i quattro quadranti di C ed inoltre si denotano x_{mid}, y_{mid} i segmenti verticali ed orizzontali che permettono di ottenere i quadranti. Nella figura 3.5 si può notare come i punti che giacciono sui segmenti x_{mid} e y_{mid} sono stati assegnati ai diversi quadranti, in modo da escludere eventuali dubbi.

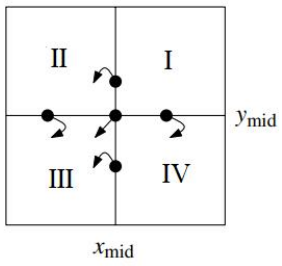

Figura 3.5: Classificazione dei punti sui lati. Immagine tratta da [1].

Questi tre casi permettono quindi di realizzare un algoritmo ricorsivo per la creazione del quadtree partendo dal set di punti e dal quadrato iniziale che li contiene tutti. Solitamente la regione iniziale viene passata come input, se così non fosse, è possibile calcolarla in un tempo lineare controllando i valori massimi e minimi delle coordinate dei punti.

Non bisogna dare per scontato che i punti siano equamente distribuiti all'interno della regione quadrata, infatti essi potrebbero trovarsi tutti all'interno di un unico quadrate e rendere quindi l'albero abbastanza sbilanciato. Perciò non è possibile definire la profondità e la dimensione del quadtree a partire dal set di punti, ma come si può intuire, la profondità dipende dalla distanza tra i punti , infatti la profondità di un quadtree è al massimo $\log(\frac{s}{c}) + \frac{3}{2}$, dove c è la distanza minima tra due punti del set e s è la lunghezza laterale della regione quadrata iniziale.

La dimensione di un quadtree e il tempo per la costruzione dipendono quindi dalla profondità dell'albero e dal numero di punti all'interno del set S. Infatti, un quadtree di profondità d, che memorizza un set di n punti avrà $O((d+1)n)$ nodi e può essere realizzato in un tempo $O((d+1)n)$, questo assumendo che i dati siano già ordinati per ogni coordinata.

Si dimostra che il numero di nodi interni ad un quadtree è legato al numero di foglie dll'albero, infatti si può notare che $n_f = 3n_i + 1$, dove n_f è il numero di foglie dell'albero, invece n_i è il numero di nodi interni. Inoltre, i quadrati rappresentati dai nodi che si trovano allo stesso livello, sono disgiunti e compongono il quadrato unitario iniziale. Questo significa che il numero di nodi interni a una data profondità è al massimo n, da qui, infatti, data una profondità si sommeranno i nodi di ogni livello arrivando a $O((d+1)n)$, è presente il $+1$ perché bisogna tenere conto del nodo radice che ha profondità 0. Lo step che necessita più tempo, nell'algoritmo ricorsivo della costruzione dell'albero, è nella distribuzione dei punti all'interno del quadrato corrente. Quindi il tempo speso per un nodo interno è lineare al numero di punti all'interno della regione associata al nodo. Dato che ogni livello ha al massimo n nodi ne segue che il tempo di costruzione limitato da $O((d+1)n)$.

3.3.3 Quadtree bilanciato

Un quadtree, come affermato precedentemente, può essere molto sbilanciato e per questo si è pensato di definire una versione bilanciata chiamato proprio *quadtree bilanciato*. Questo adattamento, si differenza dal precedente per un vincolo, cioè le celle vicine, quindi che condividono un lato o una parte, differiscono di al massimo un fattore due. In altre parole, ogni lato di un quadrato contiene al massimo due margini e quindi al massimo due lati dei quadrati vicini. Dal nuovo vincolo ne consegue che le foglie adiacenti dell'albero si trovano allo stesso livello di profondità oppure differiscono per lo più di un livello.

Dato un set S di n punti, si indica con τ il quadtree costruito su di esso, la versione bilanciata dell'albero che possiede $O(m)$ nodi, può essere costruita in un tempo $O((d+1)m)$.

3.3.4 Quadtree compresso

Si introduce ora il quadtree compresso T (*compressed quadtree*) che verrà utilizzato nell'algoritmo per la ricerca per somiglianza. Il quadtree compresso può essere visto come il

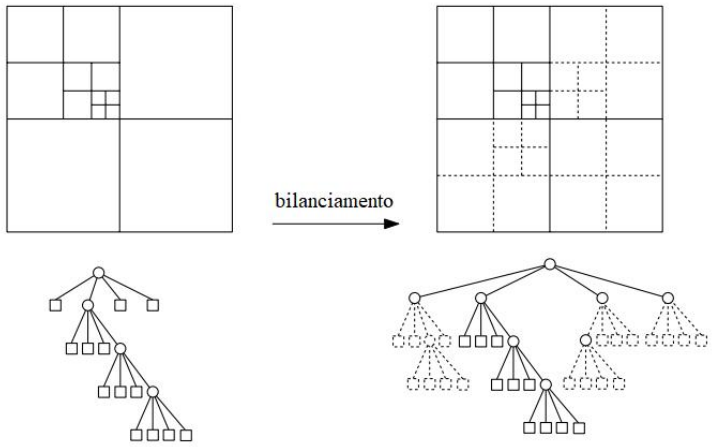

Figura 3.6: Esempio di bilanciamento del quadtree. Immagine tratta da [1].

complementare del quadtree bilanciato, ovvero nella versione bilanciata si cerca di aumentare il numero di nodi in modo da non avere differenze di profondità tra livelli maggiori di uno. In questo modo però si rischia di memorizzare molti nodi vuoti in quanto rappresentano regioni quadrate in cui non sono presenti punti del set. Attraverso la versione compressa si riesce a diminuire drasticamente il numero di nodi andando a memorizzare solamente i sottoalberi che hanno dati interessanti, quindi si definisce il quadtree compresso identificando i quadrati che devono essere presi in considerazione partendo dal quadtree standard. Si definisce come *quadrato interessante* del quadtree la regione che è rappresentato dal nodo radice, o la regione che contiene due o più figli non vuoti. Di conseguenza qualsiasi regione u che contiene due o più punti, include un quadrato interessante w(che può essere u stesso o un'area più piccola), allora w sarà il LCA nel quadtree che conterrà i punti di u. Il quadtree compresso memorizza esplicitamente solo il quadrato interessante, ignorando quelli non interessanti e rimuovendo i figli vuoti dal quadtree standard. Inoltre per ogni u si memorizzano 2^d puntatori bidirezionali per ogni quadrante di u (d è la dimensione dello spazio in cui si sta lavorando). Questi puntatori hanno tre possibili configurazioni:

- Se il quadrante contiene due o più punti, il puntatore si riferisce al più grande quadrato interessante all'interno del quadrante.
- Se il quadrante contiene un solo punto, esso sarà puntato dal puntatore.
- Se il quadrante è vuoto, il puntatore sarà *NULL*, ovvero non punterà a niente.

Quindi, un quadtree compresso per un set di n punti avrà dimensione $O(n)$ dato che l'albero avrà tanti nodi quanti saranno i punti, ma nel caso peggiore l'altezza dell'albero è

$O(n)$ che è comunque inefficiente in quanto è una lista e necessita di una scansione lineare per individuare un punto.

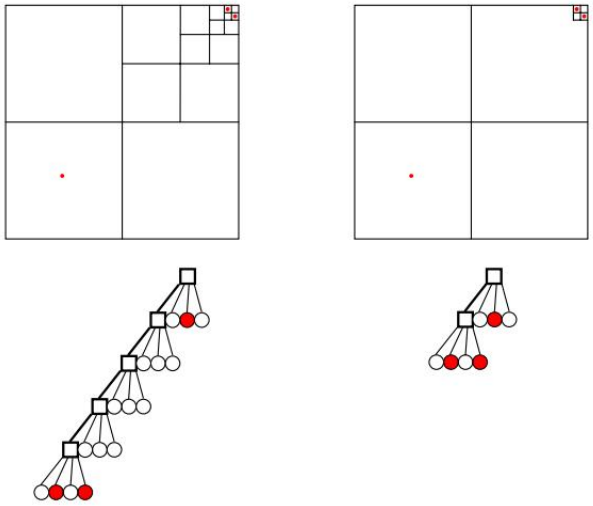

Figura 3.7: A sinistra viene rappresentato un quadtree che memorizza i tre punti all'interno del quadrato, a destra invece è presente la sua versione compressa. Le foglie vuote (ritratto dai cerchi vuoti) raffigurano i puntatori a *NULL*. Immagine tratta da [3].

.

Si assume che il quadtree esegua le seguenti operazioni in un tempo $O(1)$:

- Dato un punto x e un quadrato c, decidere se il punto appartiene alla regione, e se si, a quale quadrante appartiene.
- Dato un quadrante di c contenente un quadrato interessante w e un punto x che non appartiene ad essa, trovare il più grande w all'interno del quadrante.
- Dato un quadrante di c contenenti due punti x e y, trovare il più ampio quadrato interessante all'interno del quadrante.

L'interrogazione standard all'interno del quadtree, consiste nel localizzare la regione che contiene un dato punto x. Tale ricerca inizia dalla radice dell'albero e prosegue attraverso i puntatori tra genitore-figlio, restituendo alla fine il più piccolo quadrato interessante $w(x)$ all'interno del quadtree che contiene la posizione di x. Si ricorda che $w(x)$ può essere una foglia o un nodo interno i quali figli non coprono la regione di x. Se il quadrante di $w(x)$ copre l'area di x e contiene un solo punto (x stesso), si è trovato così il punto nel quadtree. Altrimenti x non appartiene all'albero, ma si è trovato il più piccolo quadrato interessante di T che include la locazione di x. La ricerca procede dalla radice

fino alle foglie e richiede un tempo $O(1)$ per le elaborazioni in ciascun livello, quindi, il tempo richiesto è $O(n)$ nei casi peggiori.

3.4 Algoritmo approssimato per ricerca per somiglianza

Sia S un set di n punti nello spazio $\mathbf{R}^d$, si vuole elaborare S in maniera che, data una query point q, si possa restituire un punto vicino a q all'interno di S abbastanza velocemente. In questa sezione verrà trattata una versione approssimata, ovvero non si richiede un punto vicino (la cui distanza da q sia minore di r) ma si specifica un parametro $\varepsilon > 0$ in modo da rispondere alla ricerca per somiglianza in dati con una versione $(1+\varepsilon)-approssimata$.

Quindi, in modo più formale, dato un set S contenuto in $\mathbf{R}^d$ e il punto q, si denota con $\tilde{q}$ il punto contenuto in S che sia vicino a q. Inoltre, si indica con $d_S(q)$ la distanza tra q e il punto più vicino ad esso, in questo esempio sarà $d_S(q) = \|q - \tilde{q}\|$. Il punto x, che appartiene all'insieme S, è un $(1 + \varepsilon) - approximate\ near\ neighbor$ se la sua distanza è minore o uguale alla distanza del punto più vicino sommando il margine ε, ovvero se $\|q - x\| < (1 + \varepsilon)d_S(q)$. Questa definizione può essere scritta in modo più compatta affermando che x è $(1 + \varepsilon) - ANN$ di q. Nella figura sottostante si può notare che l'aggiunta del parametro $\varepsilon > 0$ consente di trovare anche punti che non siano così vicini a q, ma che si trovino entro un range più largo.

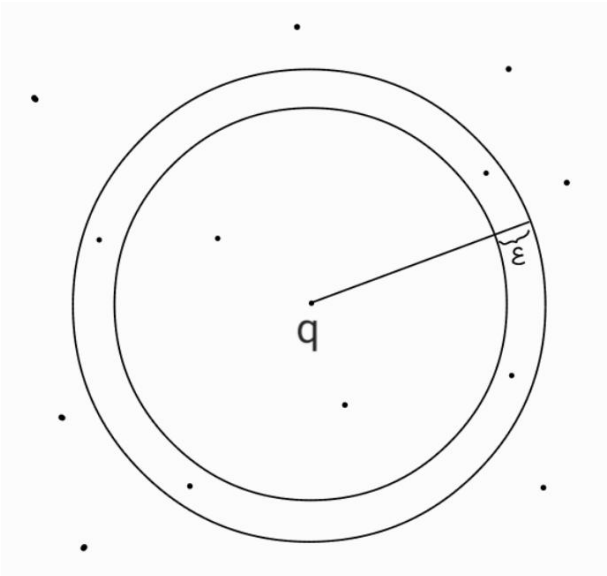

Figura 3.8: Aggiungendo ε, è aumentato il numero dei possibili punti candidati come vicini.

3.4.1 L'albero separatore dell'anello

Si introduce ora un algoritmo "grezzo" che ricerca il vicino per somiglianza, essa utilizza l'albero binario come struttura dati e introduce un nuovo elemento, ovvero *l'albero separatore dell'anello t (t-ring separator tree).*

La palla di centro m e raggio r si denota come $B(m,r)$ e si chiama $B'(m,r)$ il suo complementare ovvero la regione esterna alla palla. L'anello R è la differenza tra due palle concentriche in cui un raggio è maggiore dell'altro, precisamente $R = B(m,r_2)\backslash B(m,r_1)$ dove $r_2 > r_1$. Per semplicità si chiamerà la palla più grande B_{out} e B_{in} la più piccola, inoltre con S_{out} si indicano i punti che si trovano all'esterno della palla di raggio r_2 ovvero $S_{out} = S \cap B'_{out}$, invece $S_{in} = S \cap B_{in}$ denota tutti i punti che si trovano all'interno della palla di raggio r_1 (vedi figura 3.9).

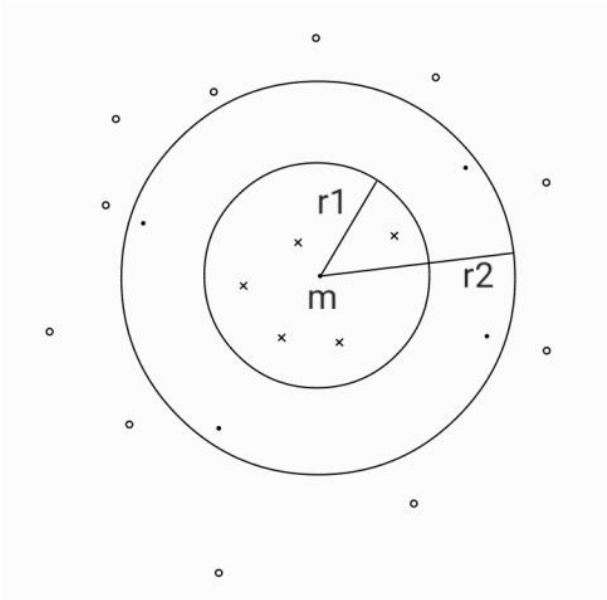

Figura 3.9: le 'x' formano l'insieme S_{in}, gli 'o' rappresentano l'insieme S_{out}.

Il *t-ring separator* di un sottoinsieme $P \subseteq S$ è la coppia (x,r) dove x è un punto del sottoinsieme P e r è un numero reale maggiore di 0 che soddisfa la seguente condizione: $|B_P(x,r)| \geq t|S|$ e $|B_P(x,r)| \leq (1+t)|S|$, in cui B_P indica la palla che contiene i punti del sottoinsieme P. Di conseguenza, il *t-ring separator tree* è l'albero binario dove ogni nodo ha un'etichetta $P \subseteq S$ costruito in modo ricorsivo secondo il seguente procedimento: la radice dell'albero ha etichetta $P = S$, un nodo etichettato da P è una foglia se $|P| = 1$ ed ha due figli se $|P| \geq 2$. In questo ultimo caso, si considera (x,r) il *t-ring separator* di P e lo si aggiunge all'etichetta del nodo (ad esempio "$P,(x,r)$" dove x è un punto di P,

P è un sottoinsieme di di S e $r > 0$. I figli di un nodo sono: uno un figlio interno la quale l'etichetta è $P_i = B(x, 2r)$ e uno un figlio esterno la cui etichetta è $P_e = P \backslash B(x, r)$.

Un albero binario T che ha come foglie i punti del set S, si denota come *l'albero separatore dell'anello* t per S se ogni nodo v che appartiene a T, è associato a una palla b_v, tale che tutti i punti considerati come interni siano $S_{in} = S_v \cap b_v$ dove S_v sono tutti i punti del set S presenti nelle foglie del sottoalbero che ha come radice il nodo v e $b_v = B(p_v, r_v)$ (palla di raggio r_v e centro p_v). Inoltre, tutti gli altri punti di S_v si trovano all'esterno della palla allargata $B(p_v, (1 + t)r_v)$ e sono memorizzati negli altri figli di v.

Quindi, dato un *t-ring separator tree* T, si riesce a rispondere alle interrogazioni $(1 + \frac{4}{t}) - ANN$ in un tempo $O(depth(T))$ (depth(T) è la profondità dell'albero), inoltre dato un set S di n punti nello spazio $\mathbf{R}^d$, il $(\frac{1}{n}) - ring\ separator\ tree$ di S può essere calcolato in un tempo $O(n \log n)$.

I punti nel caso precedente sono i punti che non presentano nessuna limitazione della loro collazione nello spazio, invece ora si introduce un nuovo algoritmo che utilizza il quadtree per memorizzare i punti, raggruppandoli all'interno di una regione quadrata.

Sia S un insieme di n punti contenuti all'interno di un ipercubo nello spazio $\mathbf{R}^d$ e si denota un quadtree Q che contiene gli n punti, dove $diam(S) = \Omega(1)$, cioè la distanza massima tra due punti qualsiasi dell'insieme è un numero. Sia q il query point tale che la distanza dal punto più vicino sia maggiore di r, ovvero $\|\tilde{q} - q\| \geq r$. Successivamente, dato un parametro $\varepsilon > 0$, si riesce a calcolare in $(1+\varepsilon) - ANN$ a q in un tempo $(O(\frac{1}{\varepsilon^d} + \log \frac{1}{r}))$, per far sì che questo limite sia rispettato, bisogna assumere che la distanza tra i due punti più vicini all'interno del set P sia $\mu = CP(P)$ (closest pair all'interno di S). Allora l'algoritmo non cerca mai nelle celle che hanno diametro minore di $\mu/2$ dato che tutti i nodi con diametro$\leq \frac{\mu}{2}$ sono foglie.

3.4.2 ANN a basse dimensioni

Si definisce ora una versione approssimata dell'algoritmo di ricerca del vicino per somiglianza nel caso in cui la dimensione sia bassa. Dato un insieme di n punti in $\mathbf{R}^d$ contenuti in un ipercubo, si costruisce il relativo compressed quadtree $\tilde{T}$ e il ring separator tree T_R di S. A partire dal ring tree T_R e da una query point q si calcola un punto u che appartiene al set tale che la sua distanza da q sia compresa tra la distanza del punto più vicino $d_S(q)$ e questa moltiplicata per $(4n + 1)$. Matematicamente

$$d_P(q) \leq \|uq\| \leq (4n+1)d_P(q)$$

Si dividono le due uguaglianze in modo da formare un sistema, così ottenendo:

$$\begin{cases} \|uq\| \geq d_P(q) \\ \|uq\| \leq (4n+1)d_P(q) \end{cases}$$

Se si eseguono due semplici sostituzioni, in particolare $R = \|uq\|$ e $R = \frac{\|uq\|}{(4n+1)}$ si ottengono cosi le due disequazioni:

$$\begin{cases} R \geq d_P(q) \\ r \leq d_P(q) \end{cases}$$

Compattando il sistema in un'unica espressione, si ottiene

$$r \leq d_P(q) \leq R$$

Quindi, da questa espressione si scopre che la distanza del vicino è limitato dai due valori r e R. Successivamente, si denota $l = \lceil \log R \rceil$ e C come l'insieme delle celle che hanno distanza minore di R dalla query point q. Per ogni cella $c \in C$, si calcola il nodo v che appartiene al quadtree $\tilde{T}$ tale che l'intersezione del set S e la cella c corrisponda ai punti di S nel sottoalbero del nodo v, ovvero S_v. Queste elaborazioni possono essere eseguite in un tempo $O(n \log n)$. Infine, dato una query point q e un parametro $1 \geq \varepsilon > 0$,il tempo per la ricerca del $(1+\varepsilon) - ANN$ a q è $O(\frac{1}{\varepsilon^d} + \log n)$.

Capitolo 4

Conclusioni

In questa tesi si sono quindi discusse alcuni degli algoritmi utilizzati ancora oggi in diversi campi, ovviamente ne esistono altri che rappresentano delle valide alternative. Gli algoritmi mostrati non presentano nessun codice, in quanto si è preferito spiegare a parole il loro funzionamento, infatti è sufficiente tradurre la spiegazione per ottenere il pseudocodice. In questo modo è anche possibile realizzare l'algoritmo in qualsiasi linguaggio si voglia perché consente di superare le barriere sintattiche.

Il problema del NNS, come detto precedentemente, non si limita solo alle basse dimensioni, infatti risulta molto più complesso e interessante quando la dimensione dello spazio aumenta. Fortunatamente, sono stati già trovati algoritmi per la ricerca in alta dimensionalità che possono rappresentare un intrigante argomento da studiare in un possibile futuro.

L'obiettivo che si sta cercando di realizzare ora, che risulta anch'esso complesso e allo stesso tempo stimolante, è l'unione degli algoritmi a bassa e alta dimensione, in modo da avere un'unica soluzione per i due casi. Questo problema potrebbe rappresentare un valido argomento da analizzare per una futura tesi magistrale.

Bibliografia

[1] Berg, M. D., Kreveld, M. V., Overmars, M., & Schwarzkopf, O. (1997). Computational geometry. In Computational geometry (pp. 1-17). Springer, Berlin, Heidelberg.

[2] Har-Peled, S. (2008). Geometric approximation algorithms. Lecture Notes for CS598, UIUC.

[3] Eppstein, D., Goodrich, M. T., & Sun, J. Z. (2005, June). The skip quadtree: a simple dynamic data structure for multidimensional data. In Proceedings of the twenty-first annual symposium on Computational geometry (pp. 296-305).

[4] Wikipedia contributors. "Nearest neighbor search." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 Mar. 2022. Web. 30 May. 2022.

[5] Abdullah, A., Moeller, J., & Venkatasubramanian, S. (2012, June). Approximate Bregman near neighbors in sublinear time: Beyond the triangle inequality. In Proceedings of the twenty-eighth annual symposium on Computational geometry (pp. 31-40).

[6] Krauthgamer, R., & Lee, J. R. (2005). The black-box complexity of nearest-neighbor search. Theoretical Computer Science, 348(2-3), 262-276.