Alla scoperta del Cover Tree

Per la ricerca dei Nearest Neighbors

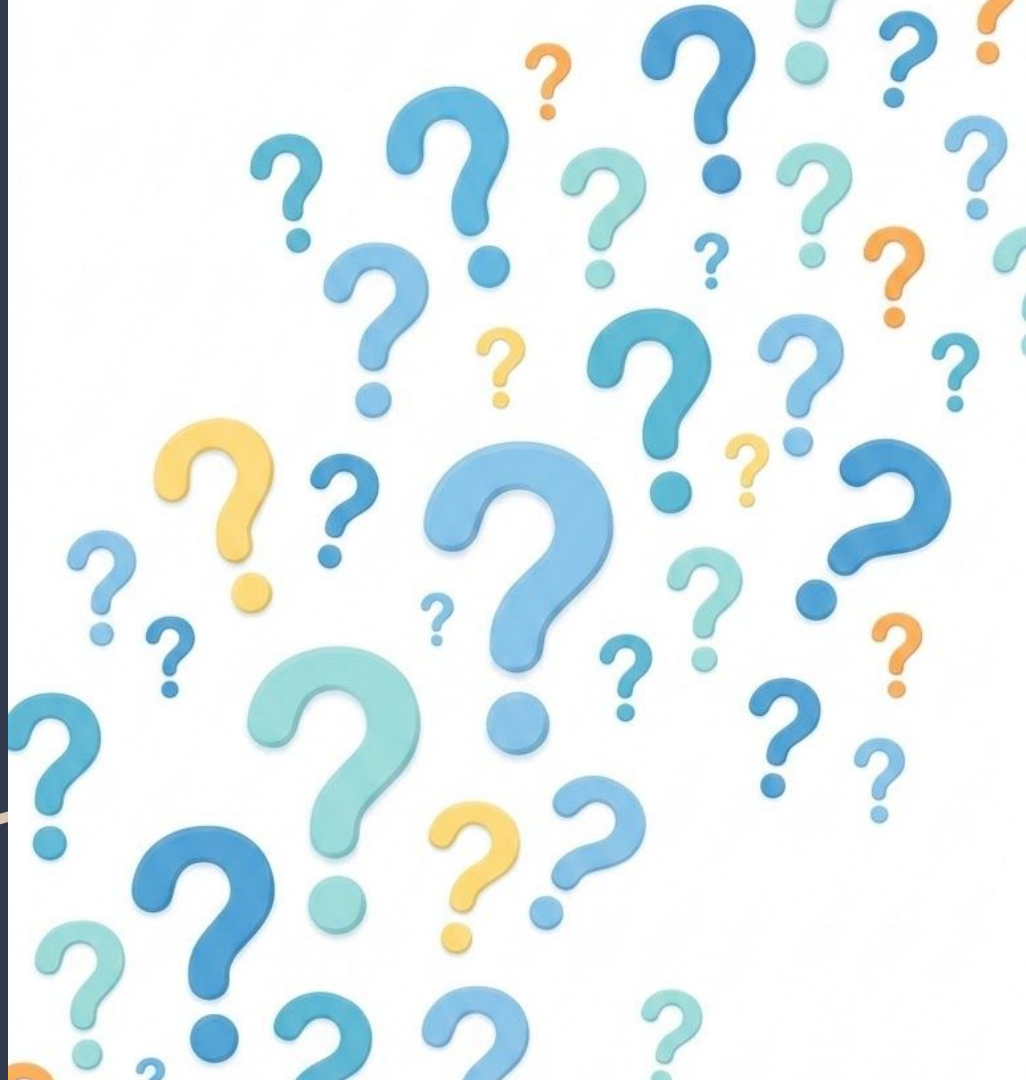
Marco Tessari ortogonalista
Andrea A. Pietracaprina mentore


1. LA RICERCA DEI NEAREST NEIGHBORS
2. COVER TREE
3. COSTRUZIONE
4. RICERCA

Roadmap

La ricerca dei Nearest Neighbors

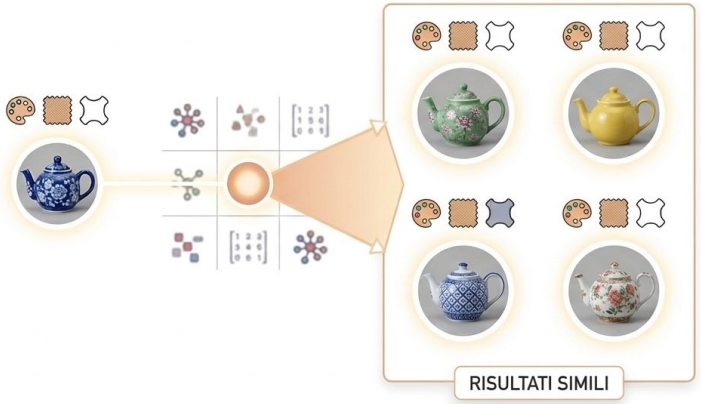
Esempi
Il problema
Possibili soluzioni
Approcci comuni



Esempi

Reverse image search

Quando si cerca una immagine sul web, ne vengono estratte le caratteristiche per poi trovarne l'origine o altre **immagini che siano simili** a quella cercata



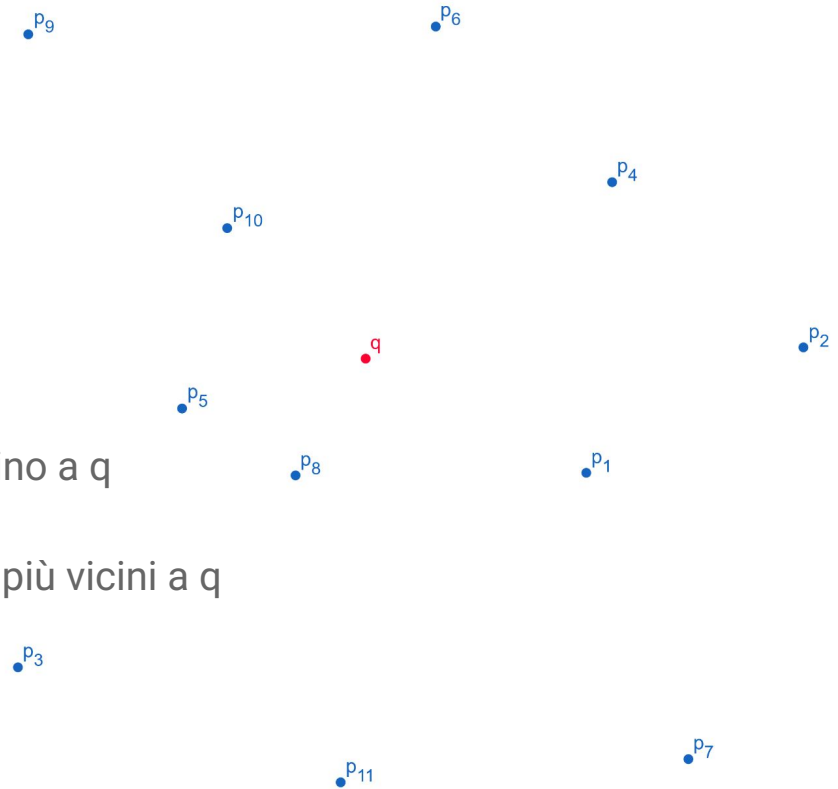
Classificazione

Classifichiamo un punto assegnandogli l'**etichetta più frequente** tra i suoi vicini più prossimi

Il problema

In uno spazio metrico (X, d) sono dati:

- un insieme di riferimento R = {p_1, p_2, …, p_n}
- un punto q



Nearest Neighbor (**NN**): trovare il punto in R più vicino a q

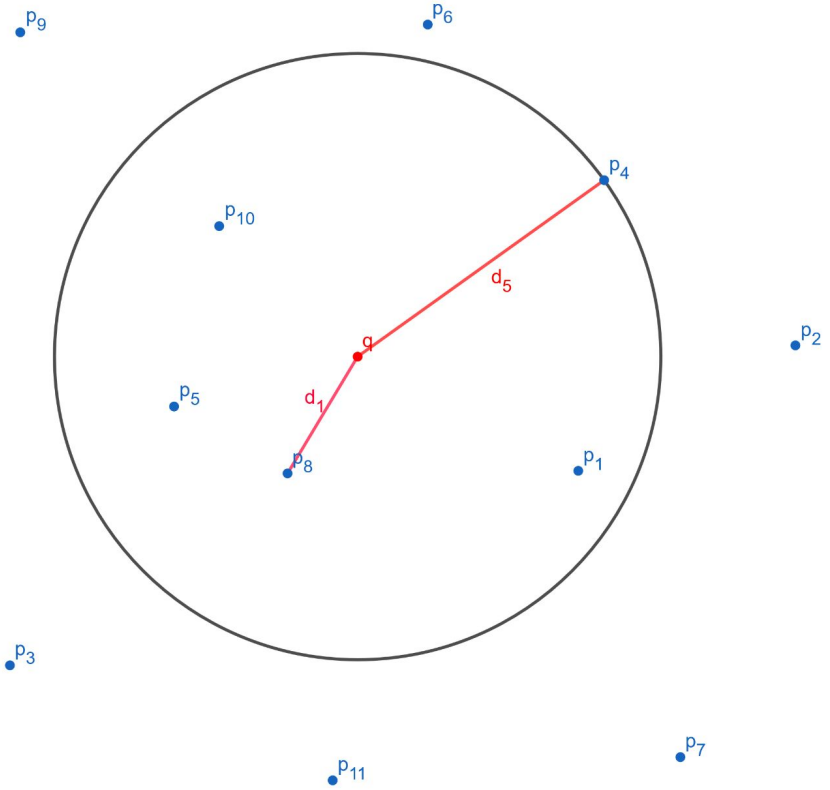
k-Nearest Neighbors (**NN_k**): trovare i k>0 punti in R più vicini a q

Il problema

Siano $\min_{p \in R} d(p, q) = d_1 \leq d_2 \leq \ldots \leq d_n = \max_{p \in R} d(p, q)$
le distanze da q ordinate dei punti in R

NN (q, R) $\in \arg\min_{p \in R} d(p, q)$

NN_k(q, R) $= \{p \in R \mid d(p, q) \leq d_k\}$



Facile, no?

Possibili soluzioni

Semplici ma inefficienti algoritmi per NN e NN_k

Nearest Neighbor:
Scansionare R per trovare il NN di q **$O(n)$**

k-Nearest Neighbors:
Usare una max-heap con chiave d(p, q) e capacità k **$O(n \log k)$**

La notazione ***O*(·) nasconde** il costo del **calcolo della distanza**, che è $O(D)$ se $X = \mathbb{R}^D$

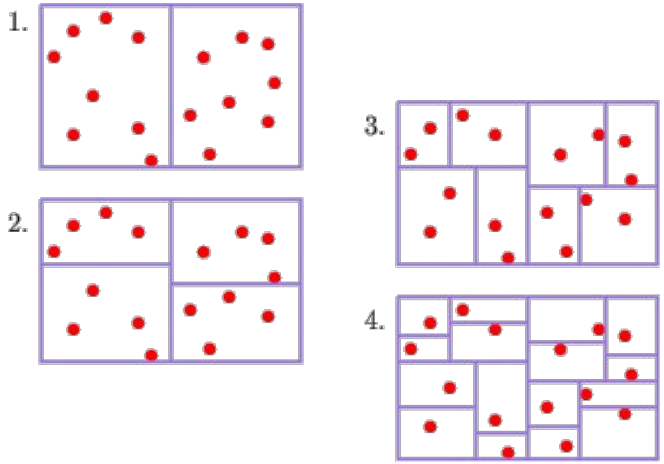
Prestazioni spesso inaccettabili, si preferisce usare **strutture dati apposite**

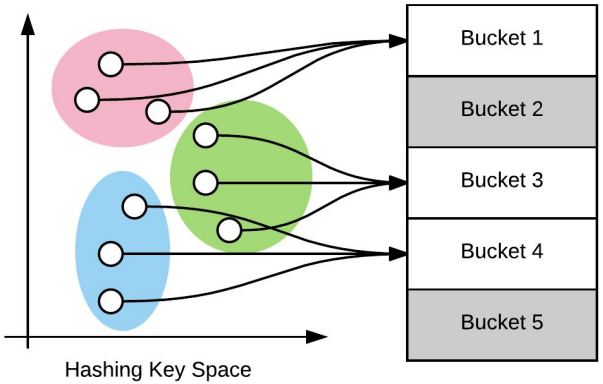
Approcci comuni

K-d Tree
Struttura ad albero per la **ricerca esatta**

Prestazioni NN: ***O*(n)**

Curse of dimensionality





Locality Sensitive Hashing
Tecnica utilizzata per la **ricerca approssimata** ed **esatta**

Prestazioni: $O(n^{\rho})$ con $0<\rho<1$

Uso di **spazio potenzialmente elevato**

Cover Tree

Background ed evoluzione
Proprietà fondamentali
Proprietà di Contenimento
Cover Tree compresso

Background ed evoluzione

Nata da un'evoluzione di idee maturate nel tempo

2004

Navigating nets: Simple algorithms for proximity search

[Extended Abstract]

Robert Krauthgamer * James R. Lee †

Cover Trees for Nearest Neighbor

Alina Beygelzimer BEYGEL@US.IBM.COM
IBM Thomas J. Watson Research Center, Hawthorne, NY 10532
Sham Kakade SHAM@TTI-C.ORG
TTI-Chicago, 1427 E 60th Street, Chicago, IL 60637
John Langford
TTI-Chicago, 1427 E 60th Street, Chicago, IL 60637

2006
>1200 citazioni su Google Scholar

A New Near-linear Time Algorithm For k-Nearest Neighbor Search Using a Compressed Cover Tree

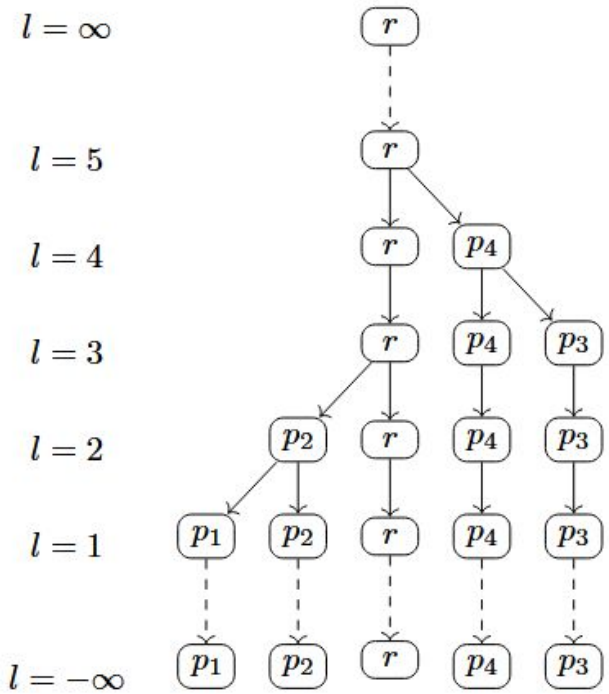
Yury Elkin [1] **Vitaliy Kurlin** [1]

2023

Il mio lavoro da ortogonalista

Studio del *Compressed Cover Tree*, semplificando le notazioni usate ove adeguato, e riscrivendo gli algoritmi principali per renderli più leggibili

Proprietà fondamentali

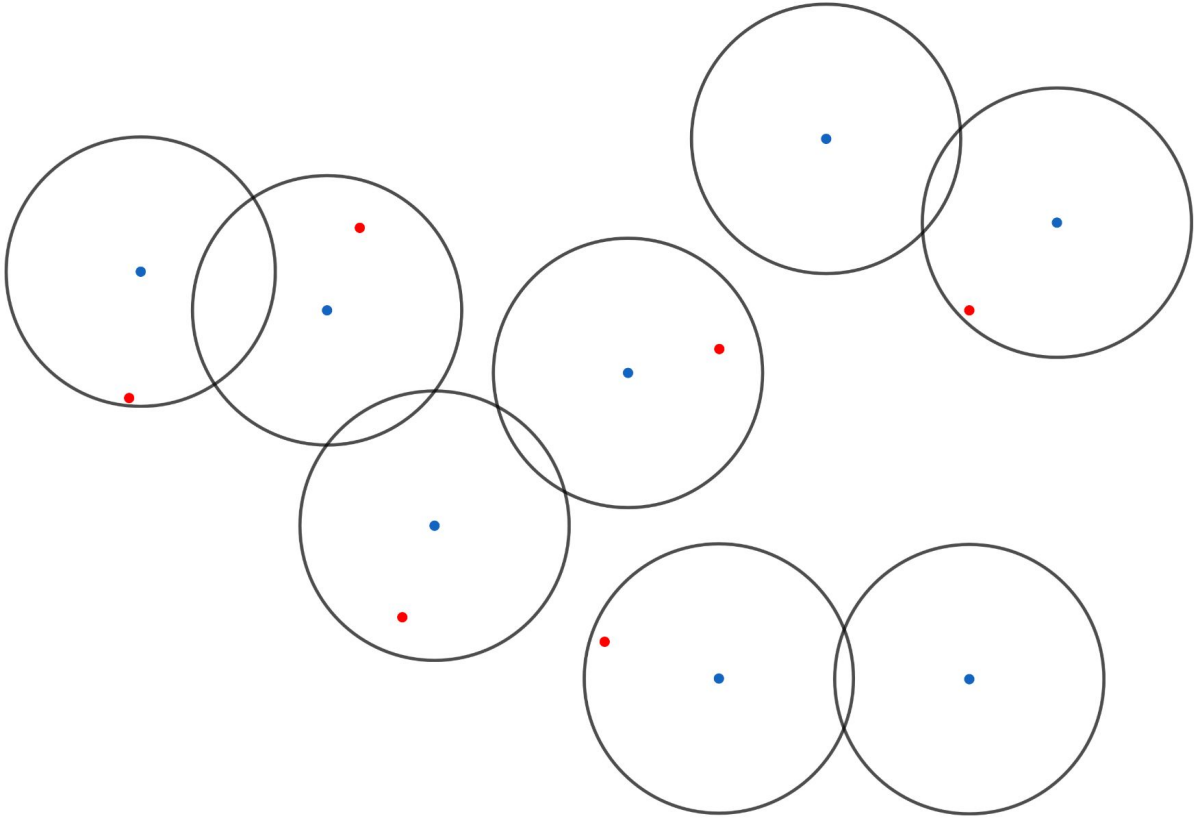


Albero concettuale: ogni punto compare infinite volte

Ogni nodo rappresenta un **punto p** ∈ R
ed una **regione** dello spazio controllata da p

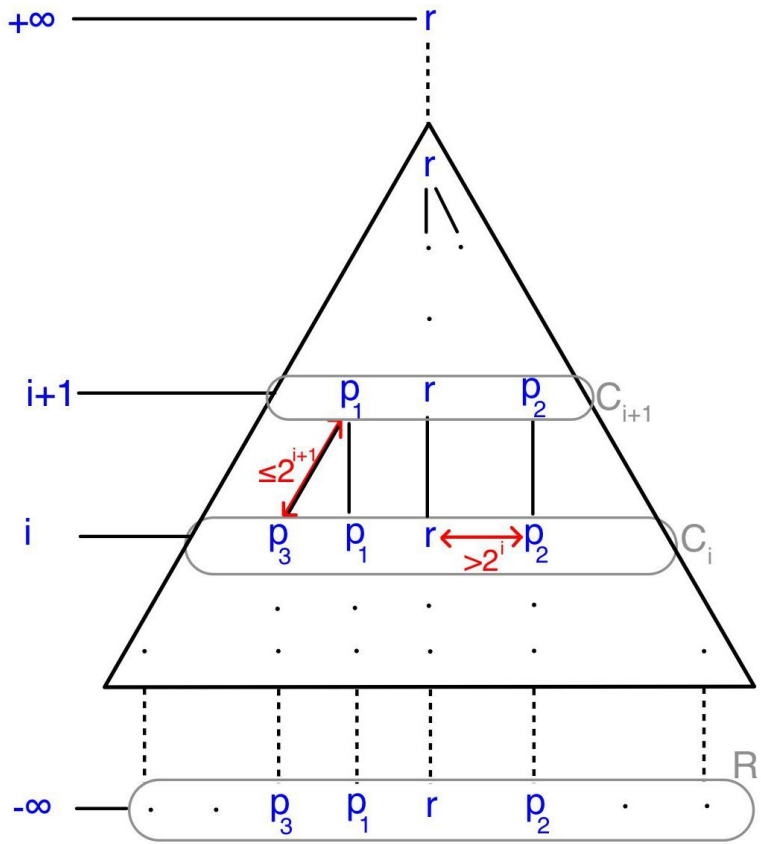
La **grandezza** della regione **dipende dal livello** del nodo

La regione **contiene i figli** del nodo,
ma non altri nodi allo stesso livello



Struttura legata alle distanze

Proprietà fondamentali



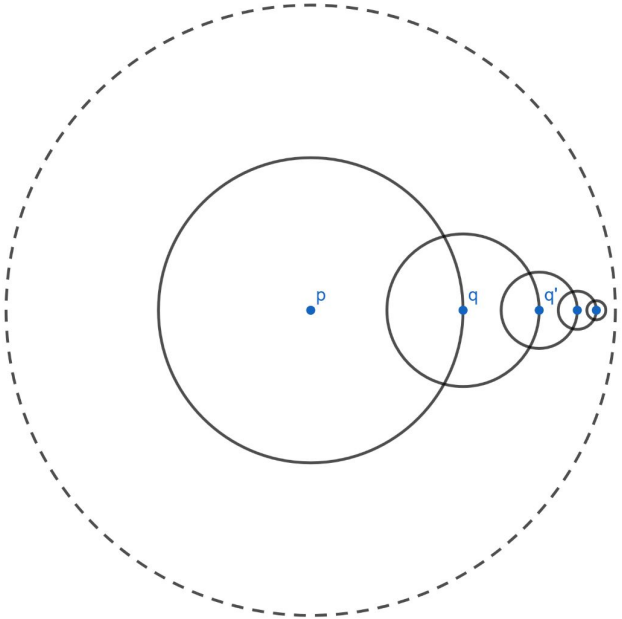
Per ogni livello i, un **cover set** $C_i \subseteq R$ è l'insieme che racchiude tutti i nodi a livello i

- **Nesting**: $\forall$ i: $\mathbf{C_i \supseteq C_{i+1}}$
- **Separation**: $\forall$ **p,q** $\in C_i$ distinti: $\mathbf{d(p, q) > 2^i}$
- **Covering**: $\forall$ q $\in C_i$ esiste **p = parent(q)** $\in C_{i+1}$ tale che $\mathbf{d(p, q) \leq 2^{i+1}}$

Un **punto r** $\in$ R viene scelto come **radice** dell'albero
Per i livelli speciali $\pm\infty$ valgono $C_{+\infty} = \{r\}$ e $C_{-\infty} = R$

Proprietà di Contenimento

Covering dal punto di vista del padre:
$\forall\ p \in C_i$ se esiste $\mathbf{q \in C_{i-1}}$ **con genitore p**
allora vale $\mathbf{d(p, q) \leq 2^i}$



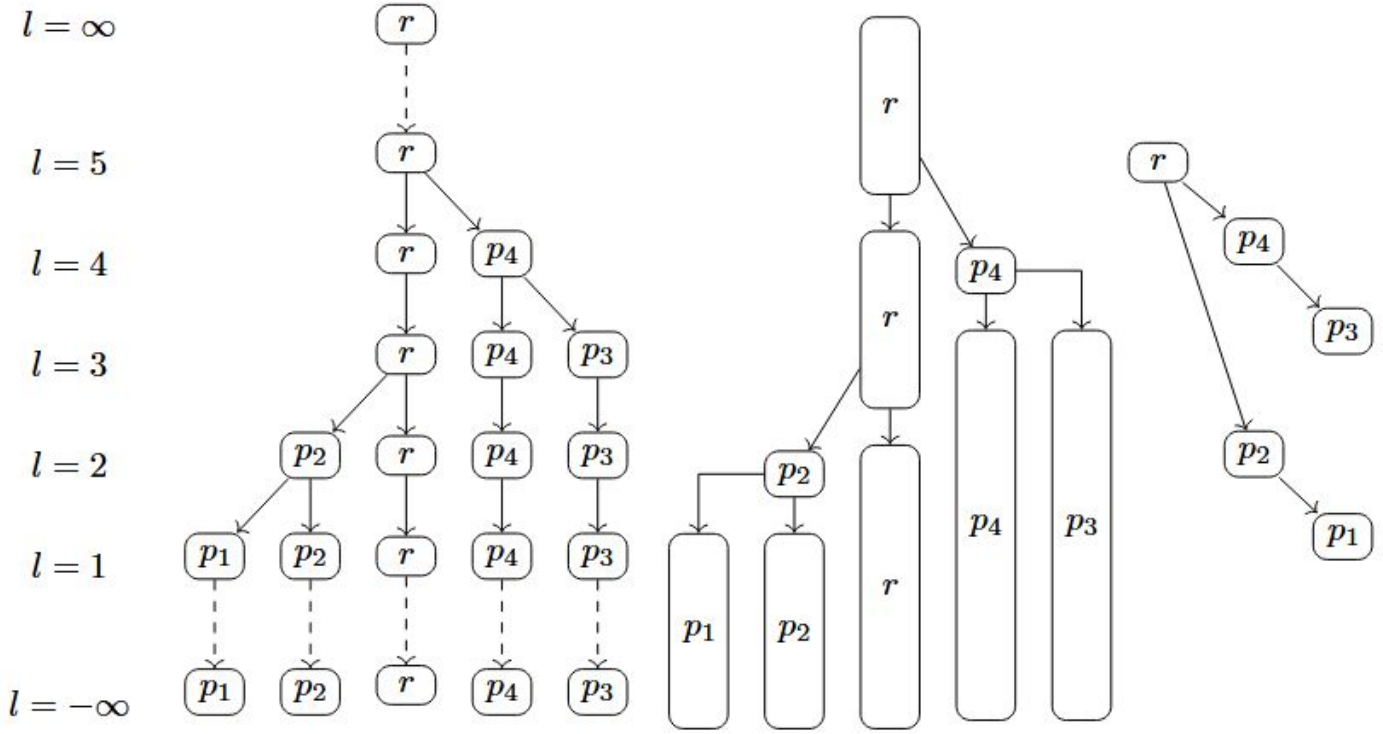
Proprietà di **Contenimento**:
se $p \in C_i$ allora $\forall$ q discendente di p vale $\mathbf{d(p, q) \leq 2^{i+1}}$

Proprietà **fondamentale** usata dagli algoritmi

Implicita (concettuale)
∞ nodi

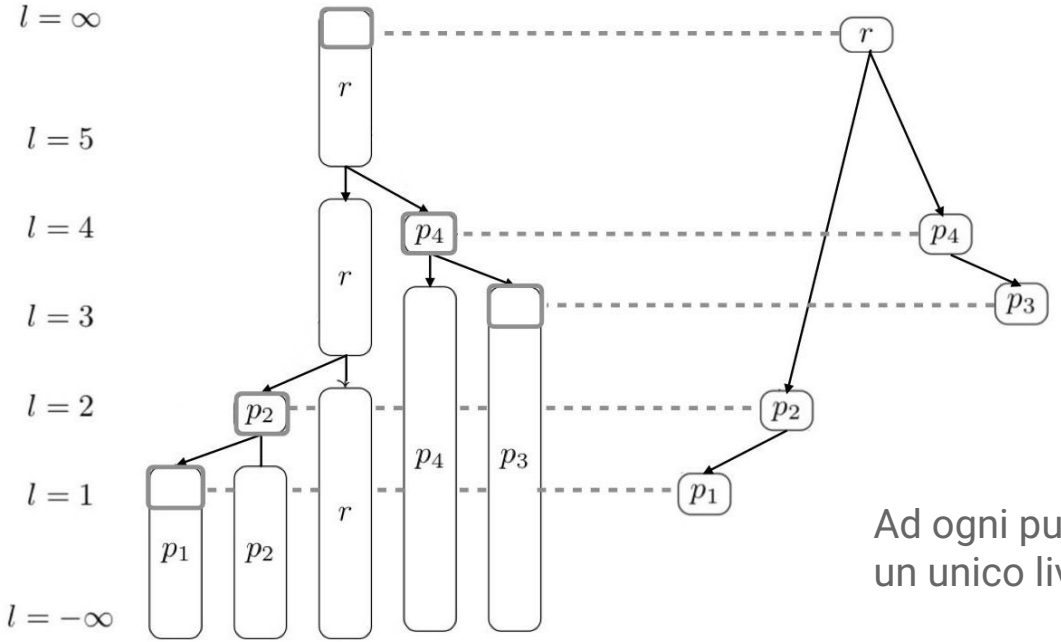
Esplicita
O(n) nodi

Compressa
n nodi



Cover Tree compresso

Idea:
Si tiene traccia solo del primo livello in cui ogni punto compare
Ogni nodo tiene traccia dei figli nei livelli sottostanti



Ad ogni punto p viene assegnato un unico livello l(p) ∈ ℤ ∪ {+∞}

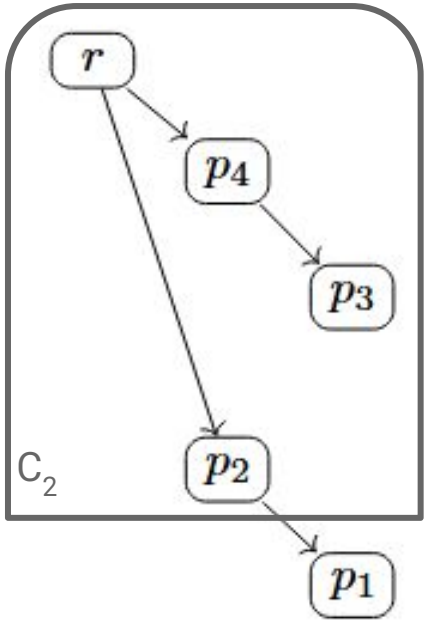
Cover Tree compresso

Cover sets: $C_i = \{p \in R \mid l(p) \geq i\}$

Nesting soddisfatta per definizione
Covering e *Separation* verificate durante la costruzione

Punti di R ↔ nodi dell'albero

Spazio totale ***O*(n)**



$l(r) = +\infty$

$l(p_4) = 4$

$l(p_3) = 3$

$l(p_2) = 2$

$l(p_1) = 1$

Costruzione

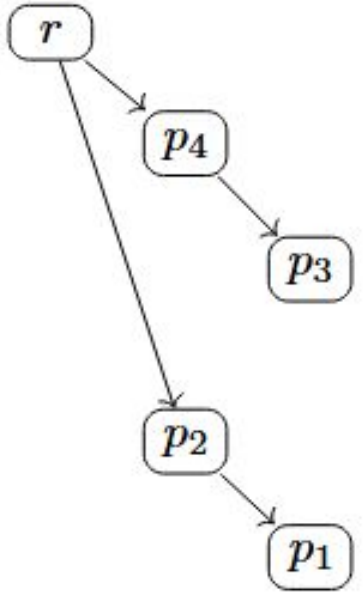
Notazioni utili
Algoritmo
Ricerca degli antenati
Ricerca del genitore
Complessità

Notazioni utili

Dati un punto p ed un livello i ≤ l(p), definiamo

Children(p, i) = insieme dei figli di p al livello i

Next(p, i) = il più alto livello sotto i in cui sono presenti figli di p,
-∞ altrimenti



l(r) = +∞

$l(p_4) = 4$

$l(p_3) = 3$

$l(p_2) = 2$

$l(p_1) = 1$

Estensione ad un insieme A di punti:

Children(A, i) = $\cup_{a \in A}$ Children(p, i)
Next(A, i) = $\max_{a \in A}$ Next(a, i)

Saltiamo da un livello al successivo usando Next(A, i)

Algoritmo

Si parte con un ordinamento arbitrario di R = $\{p_1, p_2, \ldots, p_n\}$

Costruiamo i cCT $T_1, T_2, \ldots, T_i$ sui punti $\{p_1, p_2, \ldots, p_i\}$, aggiungendo un punto alla volta

Algoritmo:
$T_1 \leftarrow$ cCT con radice r = p_1
for i = 2 … n : $T_i \leftarrow$ addPoint(T_{i-1}, p_i)
return T_n

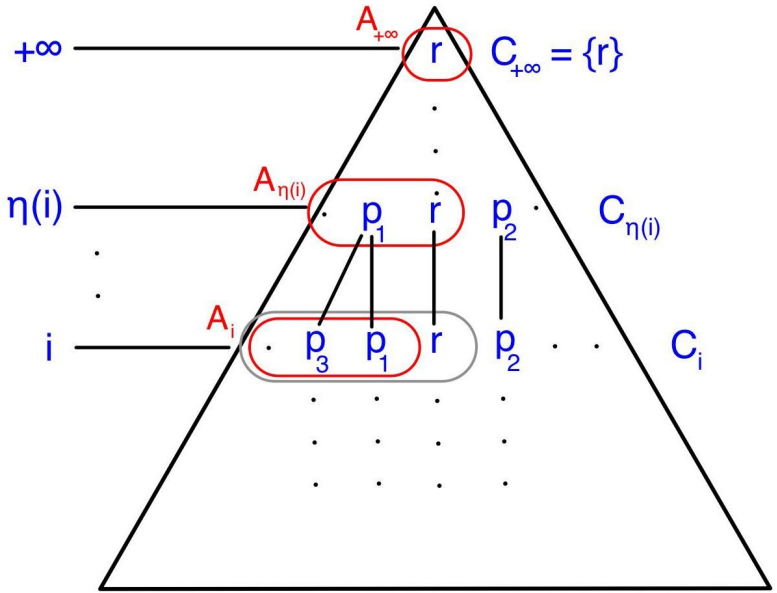
Il cCT finale è costruito su R

L'aggiunta in sé di un nuovo punto è la parte complessa, e avviene in **due fasi**:

- Ricerca dei possibili antenati
- Ricerca del genitore tra gli antenati

Ricerca degli antenati

addPoint(T, p):



Fase 1
Identifichiamo possibili antenati di p, partendo dalla radice

A_i: **possibili antenati di p a livello i** (nodi attivi)

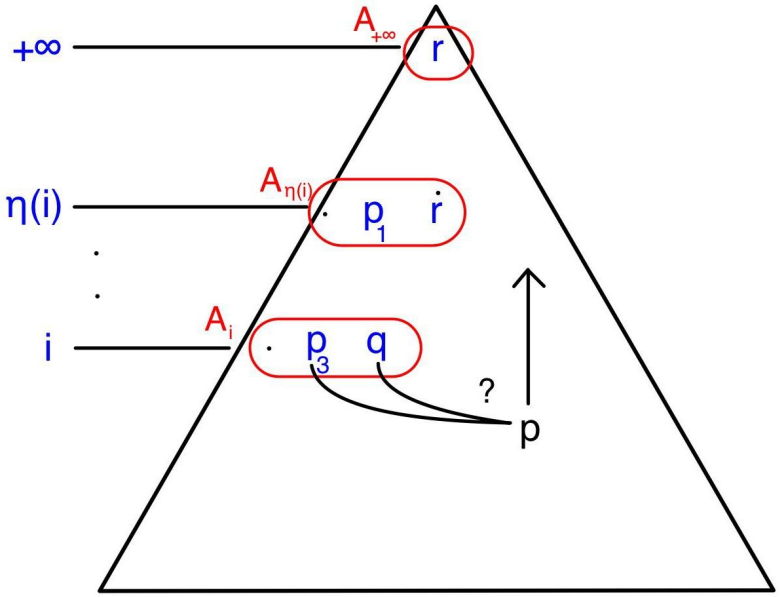
Due operazioni:

1. Espansione del livello precedente
   $E_i = A_{\eta(i)} \cup \text{Children}(A_{\eta(i)}, i)$
2. Calcolo dei nodi attivi
   $A_i = \{q \in E_i \mid d(p, q) \leq 2^{i+1}\}$ (*Contenimento*)

Quando non ci sono nuovi livelli da esplorare oppure se A_i è vuoto, comincia la fase 2

Ricerca del genitore

addPoint(T, p):



Fase 2

Ricerca di un genitore q valido per p tra gli antenati

Sia $\mathbf{q_i \in A_i}$ il nodo che **minimizza d(p, q_i)**

q_i: possibile genitore

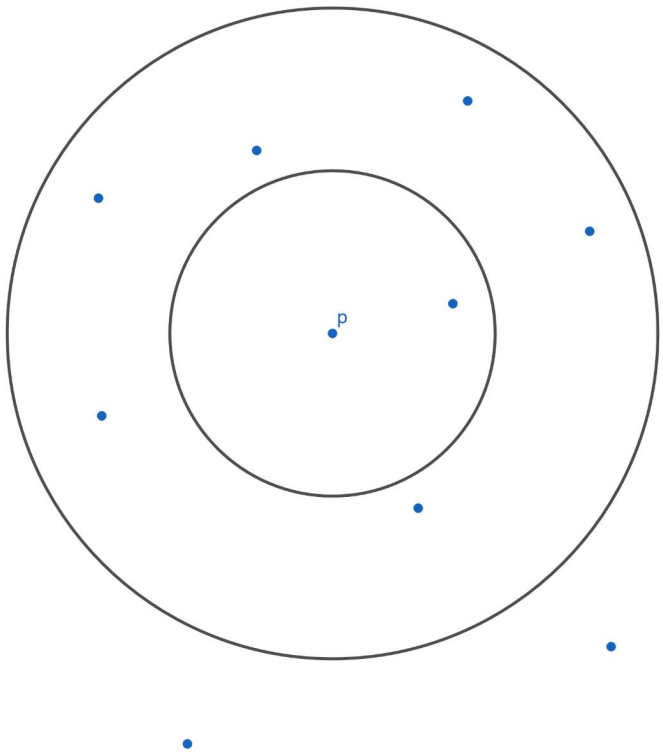
Parto dall'ultimo livello esplorato nella fase 1,
e risalgo fino a che

$d(p, q_i) \leq 2^i$ (*Covering*)

Assegno **q_i = parent(p)**

l(p) = massimo livello che soddisfa la *Separation*

Complessità



L'**expansion constant** c(R) è una costante intrinseca ad R
Esprime l'aumento del numero di punti contenuti in una palla centrata in p ∈ R se ne raddoppio il raggio

Definizione:
Sia $B_R(p, r)$ l'insieme dei punti di R con distanza da p non maggiore di r
c(R) è il **massimo rapporto** tra $|\mathbf{B_R(p, 2r)}|$ e $|\mathbf{B_R(p, r)}|$, considerando ogni possibile p ∈ R e r > 0

Approccio beyond worst-case per l'analisi della complessità

Complessità

L_p = numero di livelli attraversati dalla fase 1 aggiungendo $p \in R$

- $\mathbf{L_p = O(c^{O(1)} \log n)}$
- $|\mathbf{A_i}| = \mathbf{O(c^{O(1)})}$ per ogni livello i visitato

Loop principale: n-1 iterazioni, ognuna compie le due fasi
Fase 1: L_p iterazioni, ciascuna richiede tempo $O(c^{O(1)})$
Fase 2: $O(L_p)$ iterazioni, ciascuna richiede tempo $O(c^{O(1)})$

Totale: $O(n \max_p (\text{fase 1} + \text{fase 2})) = \mathbf{O(c^{O(1)} n \log n)}$

Un **K-d Tree** si costruisce in tempo $O(n \log n)$, ma **solo per spazi euclidei** con dimensionalità costante
Il **Cover Tree** si applica a **spazi metrici in generale**

Ricerca

Struttura dell’algoritmo
Algoritmo per il NN
Ragionamento geometrico
Complessità
Altri parametri

Struttura dell'algoritmo

Gli algoritmi per i problemi NN e NN_k sono simili, **ci concentriamo** quindi **sul NN**

Algoritmo di mia ideazione, modificando quello del NN_k descritto nel paper

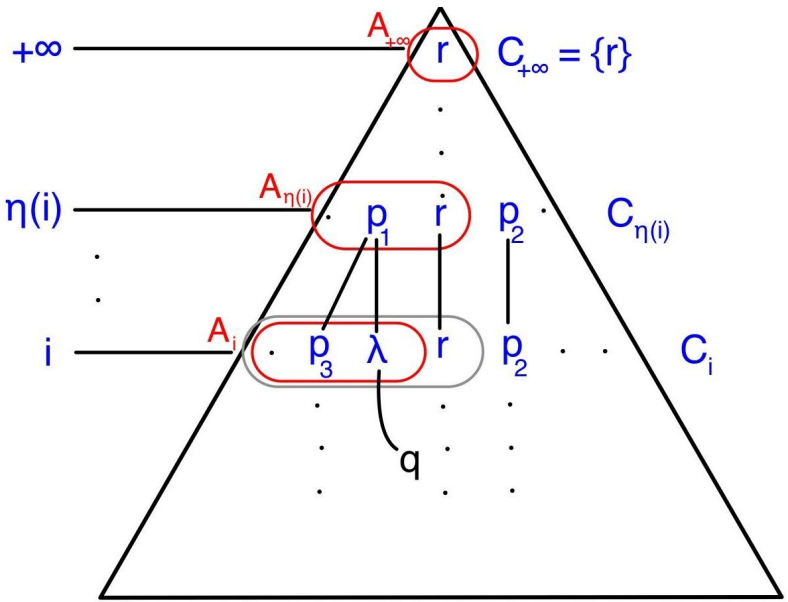
Idea:

Simile alla fase 1 della costruzione

Discendo l'albero e **considero solo nodi** che potrebbe contenere il **NN di q nel loro sottoalbero**

Ad ogni livello espando i nodi attivi del livello precedente ed escludo quelli troppo lontani

Algoritmo per il NN



Identifichiamo possibili antenati del NN di q, partendo dalla radice

A_i: possibili antenati del NN di q a livello i (nodi attivi)

Tre operazioni:

1. Espansione del livello precedente
   $E_i = A_{\eta(i)} \cup$ Children($A_{\eta(i)}$, i)
2. Stimo il NN di q
   λ = NN di q in E_i
3. Calcolo dei nodi attivi
   $A_i = \{p \in E_i \mid d(p, q) \leq d(q, \lambda) + 2^{i+1}\}$

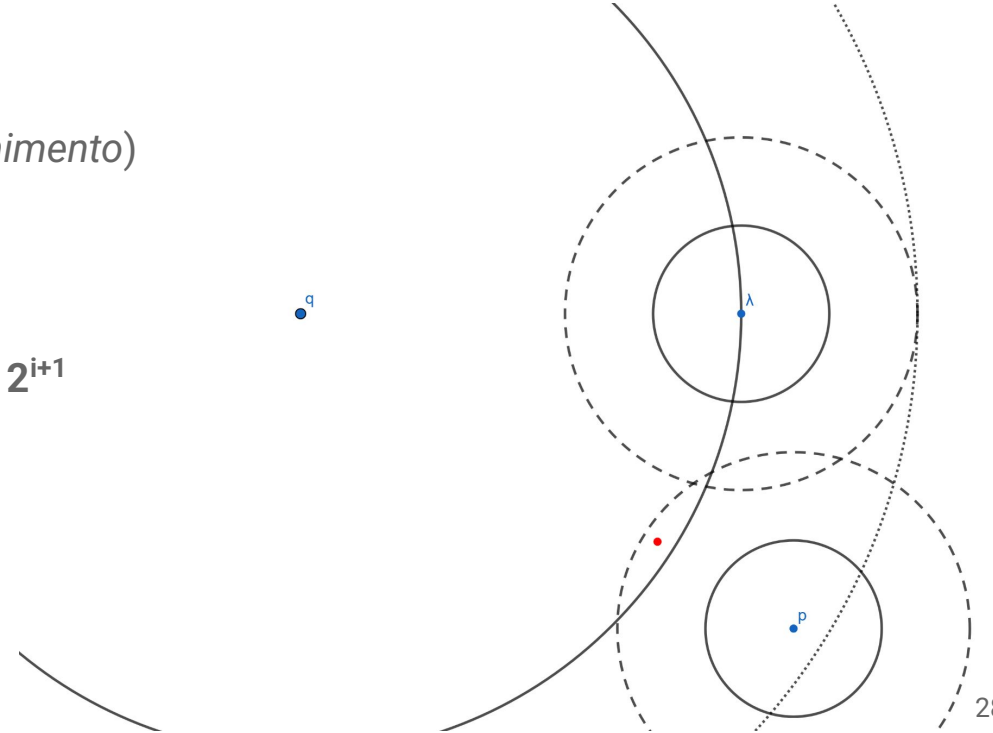
Disceso tutto l'albero **restituisco λ** come NN di q

Ragionamento geometrico

λ stima il NN al livello i, ma non escludiamo altri punti che potrebbero avere discendenti più vicini a q di λ

Raggio di ricerca: $d(q, \lambda) + 2^{i+1}$ (*Contenimento*)

Interrompo prima la discesa quando $d(q, \lambda) > 2^{i+1}$
per garantire un taglio significativo del raggio di ricerca per ogni livello



Complessità

L_q = numero di livelli attraversati

- $\mathbf{L_q = O(c^{O(1)} \log n)}$
- $\mathbf{|A_i| = O(c^{O(1)})}$ per ogni livello i visitato

L_q iterazioni: ciascuna richiede tempo $O(c^{O(1)})$
A fine ciclo compio una volta sola operazioni $O(c^{O(1)})$

Totale: $O(L_q\, c^{O(1)} + c^{O(1)}) = \mathbf{O(c^{O(1)} \log n)}$

Per insiemi R con c(R ∪ {q}) piccolo le **prestazioni** sono **eccellenti**

Altri parametri

- **Minimized expansion constant $c_m(R)$**
  minimo valore di c(A) con A ⊇ R
- **Aspect ratio Δ(R)**
  Rapporto tra la distanza massima e minima dei punti in R
- **Doubling dimension**
  Misura di dimensionalità dello spazio metrico (*X*, d) ⊇ R

Per esempio, il numero di livelli occupati da un cCT è *O*(log Δ(R))

La complessità di ricerca del NN è anche esprimibile come $\boldsymbol{O(c^{O(1)}(c_m{}^{O(1)} \log n + c))}$

Usare altre costanti permette di ampliare lo spettro di istanze per cui l'algoritmo è efficiente

Conclusioni

Il compressed Cover Tree è adeguato per la ricerca del Nearest Neighbor

Con un approccio beyond worst-case, se l'expansion constant è $O(1)$ le complessità sono

- Costruzione: $\boldsymbol{O(n \log n)}$
- Ricerca del NN: $\boldsymbol{O(\log n)}$

Modificando l'algoritmo di ricerca, si risolve anche $NN_k(q, R)$
con complessità $\boldsymbol{O(c^{O(1)} \log k \, (c_m{}^{O(1)} \log n + c\, k))}$

Esistono analisi fatte in funzione della doubling dimension, ma la ricerca è ancora attiva

Grazie per l’attenzione

Domande?