Masarykova univerzita
Fakulta informatiky

Vyhľadávanie najbližších vektorov s využitím knižnice FLANN

Bakalárska práca

Tomáš Durčák

Brno, Jar 2015

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Tomáš Durčák

Vedúci práce: RNDr. David Novák, Ph.D.

Poďakovanie

I would like to thank my supervisor . . .

Zhrnutie

The aim of the bachelor work is to provide . . .

Kľúčové slová

keyword1, keyword2 . . .

Obsah

1 Úvod

V posledných dvoch desaťročiach sa do popredia stále viac a viac dostáva potreba vyhľadávania informácií v multimediálnych dátach, čo má za následok rapídny rozvoj technológií a metód v oblasti počítačového videnia (computer vision). Takýmito multimediálnymi dátami môže byť napríklad databáza obsahujúca množstvo obrázkov (rádovo v tisícoch, prípadne miliónoch). Bežnou požiadavkou užívateľa je vyhľadať v takejto databáze obrázky, ktoré sú podobné dotazu, teda obrázku, ktorý systému užívateľ ponúkne. Od tejto podobnosti sa očakáva, že výsledné obrázky vrátené ako výsledok budú obsahovať objekty podobné objektom na dotazovanom obrázku. Ďalej sa očakáva, že vyhľadávanie podobných obrázkov bude efektívne v rýchlosti výpočtu a podobnosti medzi obrázkami.

Problém nájdenia najbližšieho suseda (nearest neighbor search) má veľký význam pre rôzne aplikácie ako napríklad: rozpoznávanie obrazu, kompresiu dát, rozpoznávanie vzorov a ich klasifikáciu, strojové učenie, štatistiku a analýzu dát. Avšak, riešenie tohto problému vo vysoko dimenzionálnych priestoroch sa zdá byť zložitý problém. Toto viedlo k zvýšeniu záujmu o triedy vyhľadávacích algoritmov, ktoré dokážu riešiť tento problém rýchlejšie ako bežné algoritmy.

Pre túto úlohu existujú rôzne vyhľadávacie indexy a za jedno z najlepších riešení je považovaná knižnica FLANN[1]. Cieľom bakalárskej práce bolo túto knižnicu otestovať na rôznych dátach dodaných vedúcim a vyhodnotiť efektivitu vyhľadávania. Výsledky boli porovnané s výkonom indexovaných štruktúr vyvinutých v laboratóriu DISA[2].

Plus tu dopíšem stručný popis jednotlivých kapitol..

---

1. Fast Library for Approximate Nearest Neighbors
2. Laboratory of Data Intensive Systems and Applications, MU Fakulta informatiky, Brno

2 Základné pojmy

2.1 Vektorový priestor

Vektorový priestor $\Omega = D_1 D_2 ... D_n$ má dimenziu n. Objekt (resp. vektor alebo bod) $\mathcal{O} = [a_1, a_2, ..., a_n]$ patriaci do vektorového priestoru je jednoznačne určený svojimi súradnicami $a_i \in D_i, 1 \leq i \leq n$, ktorých je práve n. D_i je doménou dimenzie a určuje množinu hodnôt(resp. vlastností), ktoré môže príslušná vektorová súradnica nadobúdať [1].

Vektorový priestor Ω, definujeme nad určitým poľom $\mathbf{P}$, s význačným prvkom $\mathbf{0}$ a dvomi binárnymi operáciami, sčítaním $+ : \Omega \times \Omega \to \Omega$ a násobením $\cdot : \mathbf{P} \times \Omega \to \Omega$, takými že platí:

$$\forall u, v, w \in \Omega : (u + v) + w = u + (v + w)$$

$$\exists \mathbf{0} \in \Omega : u + \mathbf{0} = \mathbf{0} + u = u$$

$$\forall u \in \Omega \; \exists \; u \in \Omega : u + (-u) = \mathbf{0}$$

$$\forall a, b \in \mathbf{P} \; \exists \; u \in \Omega : (a.b).u = a.(b.u)$$

$$\forall u \in \Omega : 1.u = u \text{ kde } 1 \in \mathbf{P} \text{ je jednotkový prvok z } \mathbf{P}$$

$$\forall a, b \in \mathbf{P} \; \exists \; u \in \Omega : (a + b).u = a.u + b.u$$

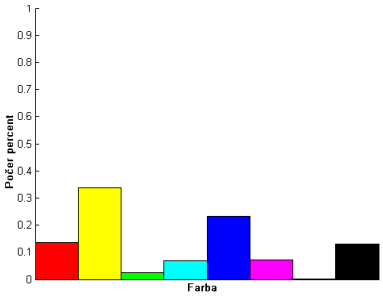
$$\forall a \in \mathbf{P} \; \exists \; u, v \in \Omega : a.(u + v) = a.u + a.v$$

Poznámka: Vektorový podpriestor je tiež vektorový priestor.

Príklad: Obrázok (resp. jeho histogram farieb) môže byť vektorom vo vektorovom priestore a mať súradnice podľa počtu pixelov z každého farebného odtieňu. Potom vektor z histogramu farieb je:

$$\{(\alpha_1, \beta_1), ...(\alpha_M, \beta_M)\}$$

kde M je počet farebných odtieňov.
Iným príkladom môže byť dokument reprezentovaný ako vektor v m-rozmernom priestore príznakov, ktoré zodpovedajú jednotlivým slovám – tzv.termom. Množina n dokumentov je reprezentovaná ako



Obr. 2.1: Histogram farieb s ôsmimi farebnými odtieňmi. Zdroj:[2]

matica $n \times m$. Neplnovýznamové slová (pomocne slovesá, spojky...) sa zvyčajne odstránia [3].

Normovaný vektorový priestor
Normovaný vektorový priestor je vektorový priestor nad $\mathbb{R}^n$, v ktorom je každému vektoru x priradené reálne číslo - **norma**, vyjadrujúca dĺžku vektora x. **Normou** na reálnom vektorovom priestore V rozumieme ľubovoľné zobrazenie $V \to \mathbb{R}$, ktoré vektoru $x \in V$ priradí reálne číslo $\|x\|$, také, že pre všetky $x, y \in V$ a ľubovoľné $c \in \mathbb{R}$ platí:

$$\|x + y\| \leq \|x\| + \|y\|$$
$$\|cx\| = |c|\|x\|$$
$$\|x\| = 0 \Rightarrow x = 0$$

Reálny vektorový priestor s normou teda nazývame normovaný priestor. Intuitívne je to vektorový priestor, v ktorom možno merať dĺžky vektorov. *Vzdialenosť bodov* a, b vo vektorovom priestore V s normou $\|\cdot\|$ nazývame dĺžku vektora $a - b$, teda číslo $\|a - b\|$.

2.2 Metrický priestor

Metrický priestor je množina, na ktorej je definovaná vzdialenosť pre všetky prvky z množiny. Táto vzdialenosť sa nazýva metrika. Metriku môžme definovať ako funkciu, ktorá určuje vzdialenosť medzi dvomi objektami.

Definícia: Nech $X \neq 0$ je množina. Definujme zobrazenie $d : X \times X \to R$, ktoré spĺňa nasledujúce vlastnosti:

$$\begin{aligned} \text{Pre všetky } x, y, z \in X \qquad & d(x,x) = 0 \\ & d(x,y) > 0 \qquad x \neq y \\ & d(x,y) = d(y,x) \\ & d(x,z) + d(z,y) \geq d(x,y) \end{aligned}$$

Množinu X nazývame **základnou množinou**, zobrazenie d **metrikou** a dvojicu (X, d) **metrickým priestorom**. V tej istej množine môžme definovať rôzne metriky. Metrika d musí byť vždy nezáporná [4].

Príklad: Vzdialenosť dvoch bodov na priamke v $\mathbf{R}$ $d = |x - y|$ alebo v rovine v $\mathbf{R^2}$ $d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

2.3 Metrické vs. vektorové priestory

Hoci pojmy *vektorový* a *metrický priestor* sú úplne odlišné, niektoré známe priestory ako napríklad $\mathbb{R}^3$, sú súčasne vektorové priestory aj metrické priestory.

Všeobecne platí, že v metrickom priestore niesu definované špeciálne operácie sčítania $+$ a skalárneho súčinu $\cdot$, ktoré sú vo vektorovom priestore. Na druhej strane, vo všeobecnosti vektorový priestor nemusí mať definovaný pojem "vzdialenosť".

Normovaný vektorový priestor je automaticky metrický priestor, definovaním vzdialenosti medzi dvomi objektami. Každý metrický priestor však nemusí byť vektorovým priestorom.

2.4 Vzdialenostné funkcie

Vzdialenostné funkcie predstavujú hlavný spôsob určenia blízkosti alebo podobnosti dvoch objektov v určitej doméne. Táto vzdialenosť môže byť definovaná rôzne. Najčastejšie v závislosti od použitého

dátového typu alebo účelu aplikácie. Vzdialenostné funkcie rozdeľujeme na dva základne typy podľa charakteru ich návratovej hodnoty:

- **diskrétne –** vracajú len vopred určenú množinu hodnôt malého rozsahu, napr. len hodnotu 1 alebo -1
- **spojité –** rozsah vrátenej množiny hodnôt je veľmi veľký alebo aj nekonečno, napr. hodnoty z intervalu $[0, 1]$

Ďalej si popíšeme najčastejšie používané vzdialenostné funkcie. Jednu z nich budeme používať aj v testovaní a porovnávaní testovaných knižníc [5, 6].

2.4.1 Minkowského vzdialenosť

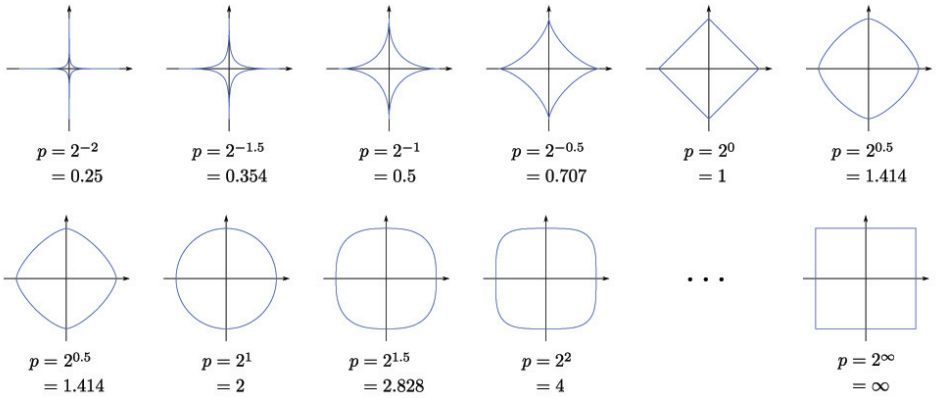
Minkowského vzdialenosť je generalizovaná metrika, ktorá zahŕňa tzv. L_p metriky v zovšeobecnenej forme. Minkowskí ju definoval takto:

$$L_p((x_1, ..., x_n), (y_1, ..., y_n)) = \sqrt[p]{\left(\sum_{i=1}^{n} \mid x_i - y_i \mid^p\right)}$$

Rovnica udáva vzdialenosť medzi dvomi objektami v n-dimenzionálnom priestore pre $p \geq 1$ (ak by bolo $p < 1$ nešlo by o metriku) [5].

Medzi najznámejšie a najpoužívanejšie metriky patria:

- L_1 – Manhattanská metrika vyjadruje najkratšiu vzdialenosť, ktorú musíme prejsť aby sme sa v meste dostali z jedného bodu do druhého. Je pomenovaná podľa pravouhlého systému ulíc v meste New York.
- L_2 – Euklidovská metrika vyjadruje najkratšiu vzdušnú vzdialenosť medzi dvomi objektmi. L_2 je použitá aj pri testoch.
- L_∞ – Čebyševova metrika, nazýva sa tiež maximálna alebo šachovnicová vzdialenosť, pretože na šachovnici, je to minimálny počet ťahov potrebných na prejdenie kráľom z jedného štvorca na iný.



Obr. 2.2: Znázornenie L_p- metrík v R^2 podľa parametra p. Zdroj:[7]

2.5 Vyhľadávanie v metrických a vektorových priestoroch

Základnou charakteristikou vyhľadávania vo vektorových a metrických priestoroch je neurčitosť. Týka sa to hlavne problému **Nájdenia najbližšieho suseda** (ang. Nearest neighbor search) [8]. Tento problém obecne definujeme takto:
Je daná množina bodov $P = \{p_1, ..., p_n\}$ v priestore X a dotazované body $q \in M$, nájdite najbližšie body z P ku q.
Pod pojmom **najbližší sused** rozumieme objekt alebo bod podobný dotazovanému bodu. Nepožadujeme aby boli výsledky vyhľadávania vždy na 100% presné, ale aby výpočet skončil v čo najkratšom čase. Pred vyhľadávaním sa musíme rozhodnúť, ako chceme určovať blízkosť objektov, akú metriku použijeme. V našich testoch bola použitá L_2 metrika, čiže klasická Euklidovská metrika, ktorá je vhodná pre husto vzorkované dáta. Ďalej sa musíme rozhodnúť, koľko najbližších susedov budeme hľadať. Existujú 3 základné druhy dotazov:

- **Dotaz na najbližšieho suseda**- hľadáme najbližší najpodobnejší objekt k dotazu, výsledkom je 1 objekt ku každému dotazu.

Definícia:

$$NN(q) = \{x \in X, \forall y \in X | d(q,x) \leq d(q,y)\}$$

Výsledkom je bod x, ktorého vzdialenosť od dotazu q je najmenšia spomedzi všetkých bodov z množiny X.

- **Dotaz na k-najbližších susedov**- niekedy nám nemusí stačiť len 1 najbližší objekt, preto hľadáme k susedov, ktorý sú mu podobný.

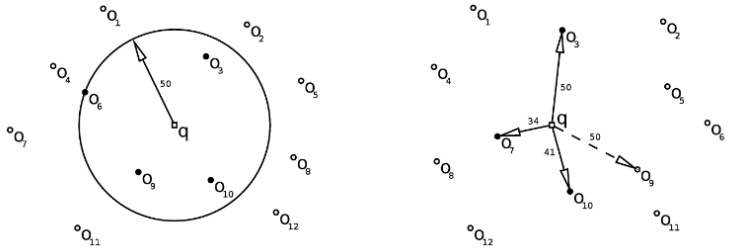
  Definícia:

  $$kNN(q) = \{A | A \subseteq X, |A| = k, \forall x \in A, \forall y \in X - A, d(q,x) \leq d(q,y)\}$$

- **Rozsahový dotaz**- použijeme, keď chceme nájsť všetkých najbližších susedov, ktorý su od dotazovaného bodu q vzdialený maximálne zvolenú hodnotu r (polomer vyhľadávania).

  Definícia:

  $$rNN(q, X, r) = \{x \in X | d(q,x) \leq r\}$$



Obr. 2.3: Znázornenie kNN a rNN v L_2-metrike. Zdroj:[9]

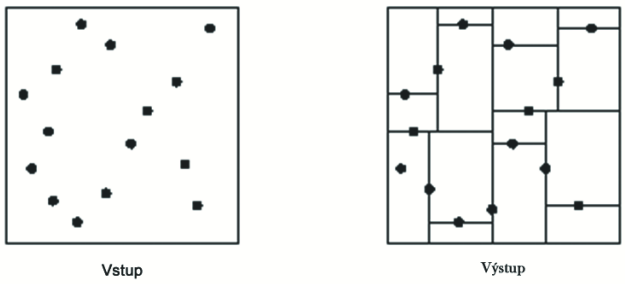
2.6 Indexové štruktúry

Pri vyhľadávaní v metrických alebo vektorových priestoroch je vhodné dáta rozdeliť do menších podmnožín a niektoré dáta pri prezeraní vynechať, čím sa môže odpoveď na dotaz značne urýchliť. Na roztriedenie dát používame špeciálne indexové štruktúry. Dve z nich si v podkapitolách popíšeme.

2.6.1 KD-stromy

KD = k-dimenzionálny strom [10] je binárny strom slúžiaci na reprezentáciu priestorových dát vyšších dimenzií vo vektorovom priestore. Dáta sú v strome rozdeľované podľa viac-dimenzionálnych kľúčov, ktorými sú zvyčajne súradnice bodov alebo vektorov.

Vytvorenie KD-stromu
Vstup: Množina objektov v k-dimenzionálnom priestore.
Problém: Vytvoriť strom, ktorý rozdeľuje priestor polrovinami, tzn. každý objekt je vo svojom vlastnom kvádri. Viď 2.4



Obr. 2.4: Rozdelenie priestoru na jednotlivé polroviny

Samotná konštrukcia stromu:

1. V každej úrovni stromu vyberáme postupne jeden z koordinátov $\{x_1, x_2, ..., x_k\}$ ako základ rozdeľovania ostatných objektov. Napríklad v koreni stromu sa rozhodujeme podľa x_1 a ako v binárnom vyhľadávacom strome, všetky objekty s hodnotou koordinátu x_1 menšou ako hodnota koreňového uzlu budú v ľavom pod-strome a s väčšou alebo rovnou hodnotou budú v pravom pod-strom

2. V ďalšej pod-úrovniach stromu triedime objekty postupne podľa ďalších koordinátov, keď už bude mať strom hĺbku k, začíname porovnávať opäť od začiatku podľa $x_1, ...$

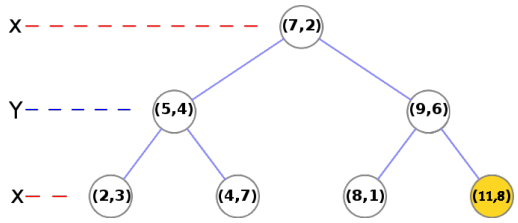
3. Ak by sme vkladali objekty do stromu v náhodnom poradí, strom by bol nevyvážený, preto sa vkladané objekty vyberajú

podľa priemernej hodnoty všetkých hodnôt na danom koordináte alebo podľa mediánu

4. Po log(n) krokoch sa rozklad zastaví, každý objekt má svoje miesto

Vkladanie objektu do stromu:
Príklad: Do stromu na Obr. 2.5 vložíme objekt s kľúčom (11,8)

- V koreni sa podľa koordinátu x porovná s uzlom (7,2), kedže 11>7 ide doprava
- V ďalšej úrovní sa porovná podľa y s uzlom (9,6), kedže 8>6 ide doprava, tento uzol je prázdny, preto sa na toto miesto uloží objekt (11,8)



Obr. 2.5: Vkladanie objektu do KD-stromu

Vyhľadávanie v KD-strome
KD-strom umožňuje vyhľadávať najbližšie body. Na Obr. 2.4 je znázornené ako strom rozdelí vektorový priestor na menšie obdĺžniky (polroviny). Nájdenie najbližšieho objektu teda znamená nájdenie oblasti, v ktorej je dotaz q a prehľadanie okolitých oblastí, kde by sa ešte mohol nachádzať ešte bližší bod.
Algoritmus:

- Nájde sa bunka c obsahujúca objekt q
- q je ohraničená nejakým objektom p (to ale nemusí byť najbližší objekt)
- Nájdeme všetky najbližšie bunky c' vo vzdialenosti $d(p, q)$

- Otestujeme či c' neobsahuje bližší objekt

Použitie:
KD-stromy sa používajú hlavne na indexovanie objektov s nižšou dimenziou. Pri vyšších dimenziách sa začne prejavovať tzv. *prekliatie dimenzionality*, ktoré spôsobuje neefektívnosť pri vyhľadávaní.

2.6.2 M-stromy

Pre indexovanie objektov v metrických priestoroch sa často využíva dátová štruktúra M-strom [1]. Štruktúru stromu tvorí opäť n-arny strom ako u KD-stromu. Rozdiel spočíva v obsahu uzlov a listov stromu. Samotné objekty sú uložené len v listových uzloch stromu. Nelistové uzly obsahujú tzv. **smerovacie objekty**.

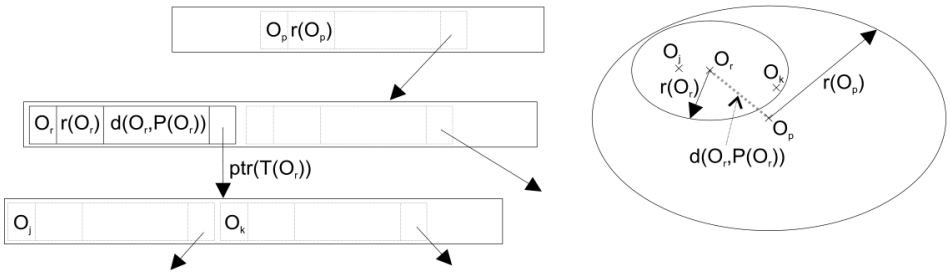
Každý smerovací objekt v uzle obsahuje:

- Samotný objekt O_r, ktorý môže byť *virtuálny* – vypočítaný pre účely M-stromu alebo *skutočný* – uložený v niektorom liste $T(O_r)$
- Odkaz $ptr(T(O_r))$ n svoj pod-strom $T(O_r)$
- Polomer $r(O_r)$
- Vzdialenosť $d(O_r, P(O_r))$ od svojho rodičovského smerovacieho objektu $P(O_r)$

Listové uzly vypadajú podobne, ale miesto odkazu na svoj pod-strom a polomeru obsahujú $oid(O_j)$ – identifikátor celého objektu.

Princíp hierarchie M-stromu spočíva rovnako ako u KD-stromu v rozdelení priestoru na menšie regióny, ktoré ale nemusia byť nutne disjunktné ako je to u KD-stromu. K rozdeleniu slúžia smerovacie objekty, pričom všetky objekty v listových uzloch, ktoré obsahuje pod-strom $T(O_r)$ smerovacieho objektu O_r, sú vo vzdialenosti maximálne $r(O_r)$ od O_r. Teda:

$$\forall O_i \in T(O_r) \qquad d(O_r, O_i) \leq r(O_r)$$



Obr. 2.6: Uzly M-stromu obsahujúce objekty (vľavo) a regióny smerovacích objektov znázornené v metrickom priestore (vpravo)
Zdroj: [1]

Na obrázku 2.6 je znázornená štruktúra stromu a vzťahy medzi smerovacími objektami.

Vytvorenie M-stromu:
Rovnaká množina objektov môže byť v M-strome indexovaná mnohými rôznymi spôsobmi, pričom neporušíme ani jeden z axiomov metriky alebo vlastnosti M-stromov. Avšak každý z týchto spôsobov nemusí byť práve ideálny z hľadiska efektivity operácií vykonávaných nad M-stromom. Obecne hľadáme také spôsoby konštrukcie stromov, aby zložitosť dotazovanie bola minimálna. Efektívnosť vyhľadávania v značnej miere ovplyvňuje výber smerovacích objektov tzv. *pivotov*, ktoré potrebujeme na rozdelenie metrického priestoru pri vytváraní stromu. Dôležitý je vyber pivotov, ktorí sú vo vhodnej vzdialenosti od ostatných objektov. Dobrí pivoti by mali byť navzájom ďaleko od seba a taktiež by sa mali nachádzať ďaleko od ostatných objektov v metrickom priestore.

Vyhľadávanie v M-strome:
V M-stromoch môžme vyhľadávať *k-najbližších susedov* ako aj pomocou *rozsahového dotazu*.

Pri vyhľadávaní k-najbližších susedov je dôležitá prioritná rada, z ktorej postupne vyberáme objekty na vyhľadávanie v strome $T(O_r)$. Každý objekt v rade má tvar:
$(prt(T(O_r)), d_{min}(T(O_r)))$. Prvý člen je už známy ukazateľ na pod-

strom, druhý člen je spodná hranica medzi dotazovaným objektom Q a každým objektom O_j z $T(O_r)$.

Použitie:
M-stromy požívame, keď dáta nie je možné indexovať vo vektorovom priestore, alebo keď sú vektory dát príliš dlhé (dimenzia > 20). Metrický priestor umožňuje definovať špeciálne metriky a tým rôzne interpretovať vzdialenosť (resp. podobnosť) objektov na základe ich vlastností.

3 Knižnica FLANN

FLANN[1] je knižnica poskytujúca rýchly programátorsky nástroj, ktorý slúži na *vyhľadávanie najbližších susedov*. Pre tento účel obsahuje kolekciu nástrojov (algoritmov) a tiež systém automatického výberu najlepšieho algoritmu a optimálnych parametrov použitých pri vyhľadávaní najbližších susedov. Aby bolo vyhľadávanie čo najrýchlejšie, je knižnica FLANN napísaná v programovacom jazyku C++ a je možné ju použiť aj pri programovaní v jazykoch C, Matlab, Python a Ruby. Knižnica je voľne šíriteľná pod licenciou BSD[2] [11].

V ďalších podkapitolách si popíšeme algoritmy *The Multiple Randomized kd-trees*[3], *The Priority Search K-Means Tree*[4] a *The Hierarchical Clustering Tree*[5], ktoré sú obsiahnuté v knižnici FLANN a javia sa najrýchlejšie z pomedzi existujúcich algoritmov na vyhľadávanie najbližších susedov.

3.1 Algoritmus Náhodných KD-stromov

Klasické algoritmy využívajúce KD-stromy sú vhodné na vyhľadávanie najbližších susedov v nízko-dimenzionálnych dátach, avšak pre vyššie dimenzie vykazujú značný pokles výkonu. Tento problém viedol k vývoju nového a vylepšeného algoritmu využívajúceho KD-stromy. [11]

Algoritmus je založený na stavaní mnohopočetných náhodných KD-stromov, ktoré sú prehľadávané paralelne. Originálne KD-stromy pri vytváraní delia dáta na polovicu v každej úrovni stromu vždy postupne podľa dimenzií zaradom. Na porovnanie náhodne KD-stromy vyberajú deliacu dimenziu náhodne z prvých D dimenzií, kde majú dáta najväčší rozptyl. Knižnica FLANN používa v imple-

1. Z anglického: Fast Library for Approximate Nearest Neighbors
2. BSD licencia sa používa pre voľne šíriteľný softvér, je pomenovaná podľa Berkeley Software Distribution, Unixový operačný systém
3. Algoritmus náhodných KD-stromov
4. Algoritmus pre prioritné K-means stromy
5. Algoritmus pre Hierarchické zhlukovacie stromy

mentácií algoritmu hodnotu $D = 5$, ktorá sa pri testoch ukázala ako optimálna. Pri vyhľadávaní sa používa prioritná rada skrz všetky randomizované KD-stromy. Objekty v tejto rade sú zoradené podľa zvyšujúcej sa vzdialenosti od deliacej hranice v každej úrovni, čo zvyšuje rýchlosť vyhľadávania. Každý objekt, ktorý už bol porovnaný s dotazom v niektorom strome, je označený a v ďalších stromoch sa už neporovnáva. Presnosť aproximácie vyhľadávania sa reguluje nastavením maximálneho počtu listových uzlov (skrz všetky stromy), ktoré sa pri vyhľadávaní skontrolujú [11].

3.2 Algoritmus pre Prioritné K-means stromy

Pri niektorých typoch dát, môže byť efektívnejší algoritmus K-means stromov, zvlášť ak je prioritou vysoká presnosť výsledkov. Algoritmus prioritných K-means stromov sa snaží lepšie využívať prirodzenú štruktúru vstupných dát. Na rozdiel od randomizovaných KD-stromov, delí a zoskupuje dáta do skupín za základe vzdialeností skrz všetky dimenzie, pričom KD-stromy využívajú na delenie v každej úrovni len jeden rozmer [11].

Opis algoritmu:
Pri tvorení K-means stromu sa dáta delia v každej úrovni do K rôznych regiónov pomocou *K-means zhlukovacieho algoritmu*. Rovnaká metóda sa rekurzívne aplikuje na každú novú skupinu dát. Delenie končí, keď počet objektov v každom regióne je menší ako hodnota K [11].

Výhľadávanie v strome:
K-means strom sa prehľadáva postupne od koreňa k najbližšiemu listu. Vetvy, ktoré boli pri prechádzaní stromu preskočené sa ukladajú do prioritnej rady, zoradené podľa vzdialenosti k dotazovanému objektu. Algoritmus potom ešte skontroluje pod-stromy uložené v tejto rade [11].

Počet regiónov K, na ktoré sa vstupné dáta delia pri vytváraní stromu určuje parameter algoritmu nazvaný *počet vetiev stromu*. Výber optimálneho K ovplyvňuje rýchlosť a správnosť vyhľadávania. Ďalším parametrom je I_{max} a určuje maximálny počet iterácií, ktoré

sa vykonajú pri delení stromu na regióny. Menší počet iterácií urýchli stavbu stromu, ale zníži presnosť vyhľadávania. Posledným parametrom je výber algoritmu, ktorý určí ako sa bude počítať rozhodovacia hranica pri delení objektov do regiónov. K dispozícií mame: *náhodný výber*, *Gonzalesov algoritmus* (výber objektov, ktoré su ďaleko od seba)[11] alebo *KMeans++ algoritmus* [12].

3.3 Algoritmus pre Hierarchické zhlukovacie stromy

Prechádzajúce dva algoritmy sú určené predovšetkým na vyhľadávanie najbližších susedov vo vektorových dátach. Niesu však vhodné na vyhľadávanie v binárnych dátach. Pre tento účel bol vyvinutý nový algoritmus, ktorý využíva novú dátovú štruktúru: *hierarchické zhukovacie stromy* [13].

Vytváranie hierarchického zhlukovacieho stromu je podobné K-means stromu. Zo vstupných dát sa náhodne vyberie *K zhlukovacích centier* a objekty sa rozdelia do *K* zhlukov podľa toho, ku ktorému zhlukovaciemu centru sú najbližšie. Hodnota *K* je vstupný parameter algoritmu. Delenie potom prebieha rekurzívne na všetkých nových zhlukoch, až kým nie je počet objektov v zhluku menši ako *K*. Vtedy sa vytvoria listové uzly stromu [13].

Na rozdiel od algoritmu pre prioritné K-means stromy, nevytvárame zo vstupných dát len jeden strom, ale celý les stromov, v ktorom sa vyhľadáva paralelne. Aby bolo vytváranie stromov rýchlejšie, je výhodné vyberať zhukovacie centrá náhodne, pričom nedochádza k výraznému poklesu presnosti. Vyhľadávanie je podobné algoritmu pre K-means stromy [13].

Pred vytváraním stromu môžme nastaviť *rozvetvenie stromu (K)*, *počet stromov*, *maximálny počet objektov v listových uzloch* a *algoritmus pre výber prvých K zhukovacích centier* [13].

3.4 Automatický výber vyhľadávacích parametrov

Výber správneho algoritmu a optimálnych parametrov pri vyhľadávanie najbližších susedov nie je triviálny problém. Správny voľba závisí od niekoľkých faktorov ako napríklad: štruktúra vstupných dát a zvolená presnosť vyhľadávania. Každý z možných algoritmov má množinu nastaviteľných parametrov, ktoré v značnej miere ovplyvňujú výsledky vyhľadávania. Je to napríklad počet náhodných stromov pri použití KD-stromu alebo počet vetiev každého uzla pri hierarchickom K-means strome [14].

Výber algoritmu je optimalizačný problém, ktorým sa snažíme nájsť najlepšie riešenie. Cenu výpočtu počítame ako kombináciu času potrebného na vyhľadávanie dotazov, času za ktorý sa vytvorí vyhľadávací index (strom) a pamäťovej záťaže. Každý z týchto faktorov má inú váhu v závislosti na aplikácií a použití. Ak vytvárame vyhľadávací index len raz, ale vyhľadávanie prebieha mnohokrát, je dôležitejší čas vyhľadávania ako čas potrebný na vytvorenie indexu. Niekedy vyhľadávame v indexe len raz, napríklad pri on-line aplikáciach, vtedy potrebujeme aby sa index vytvoril čo najrýchlejšie. Môžu nastať aj situácie, kedy potrebuje čo najmenšie pamäťová zaťaženie.Váha pamäťového zaťaženie je w_m a váha času vytvorenie indexu je w_b. Celkovú cenu počítame pomocou:

$$\text{cost} = \frac{s + w_b b}{(s + w_b b)_{opt}} + w_m m \tag{3.1}$$

kde s je čas vyhľadávania a b je čas potrebný na vytvorenie vyhľadávacieho indexu. Parameter $m = m_t / m_d$ reprezentuje pomer pamäte použitej na uloženie indexu (m_t) a uloženia dát (m_d). Parameter w_b reguluje čas potrebný na vytvorenie indexu vzhľadom ku času vyhľadávania. Ak nastavíme $w_b = 0$, znamená to, že chceme čo najrýchlejšie vyhľadávať a nezaujíma nás rýchlosť vytvorenia indexu. Nastavením $w_b = 1$, priradíme času vyhľadávania a vytvorenia indexu rovnakú dôležitosť. Podobne w_m nastavuje prioritu medzi časom (čas vyhľadávania vytvorenia indexu) a pamäťou, ktorú využíva vyhľadávací index. Hodnota $w_m < 1$ kladie väčšiu váhu na čas nehľadiac na pamäť a hodnota $w_m > 1$ kladie prioritu na množstvo

využitej pamäte. Nastavením $w_m = 1$ priradíme rovnakú prioritu času aj využitiu pamäte [14].

Výber najlepšieho algoritmu a optimálnych parametrov sa vykonáva v dvoch krokoch: V prvom kroku vyberáme algoritmus s najlepšími výsledkami podľa rovnice 3.1. Pri testovaní sa využívajú hodnoty z množiny {1,4,8,16,32} ako počet náhodných KD-stromov, {1,5,10,15} ako počet iterácií a {16,32,64,128,256} ako počet vetiev v K-means strome. V druhom kroku sa využíva Nelder-Mead downhill simplex metóda [15], pomocou ktorej sa určia optimálne parametre pre algoritmus vybraný v prvom kroku. Optimalizácia môže byť počítaná na celých vstupných dátach, alebo len na časti. Typicky stačí $5 - 10\,\%$ a výsledky sa stále veľmi blížia optimálnym hodnotám (ak sú dáta podobnej štruktúry) [14].

3.5 Použitie knižnice FLANN v C++

Jadro knižnice FLANN je naprogramované v programovacom jazyku C++, preto je na jej samotnú kompiláciu potrebný C++ prekladač. Knižnicu je možne použiť s Linuxovými distribúciami, ale aj Windowsom a OS X. Pre viac-jadrovú podporu je nutné mať nainštalovanú knižnicu OpenMP. Predkompilovanú knižnicu FLANN je možné použiť aj pomocou knižnice PCL (Point Cloud Library)[6], do ktorej je implementovaná jedna zo starších verzií. V ďalších častiach si popíšeme niektoré základné triedy a funkcie knižnice [16].

Na ukladanie dát slúži trieda `Matrix<ElementType>`. Ukladáme do nej vstupné objekty, dotazy a výsledky vyhľadávania(vzdialenosti a ID objektov). FLANN potrebuje mať všetky dáta v operačnej pamäti. Na načítanie dát, môže byť použitá knižnica HDF5[7], ktorá musí byť nainštalovaná pred kompiláciou knižnice FLANN.

Po načítaní dát, je potrebné vytvoriť vyhľadávací index a nastaviť parametre. Na tento účel slúži trieda `flann::Index<T>`. V konštruktore triedy sa nastavuje typ vzdialenostnej funkcie a typ vyhľadávacieho algoritmu. Medzi zakladné typy patrí:

6. http://www.pointclouds.org
7. https://hdfgroup.org/HDF5/

- **LinearIndexParams** Určený pre lineárne vyhľadávanie.
- **KDTreeIndexParams** Index s týmto typom vytvorí randomizované KD-stromy, ktoré sa prehľadávajú paralelne. V štruktúre je možné nastaviť počet stromov (`trees`), ktoré sa majú vytvoriť.
- **KMeansIndexParams** Index s týmto typom vytvorí hierarchický k-means strom.
- **CompositeIndexParams** Index s týmto typom je kombináciou randomizovaných KD-stromov a hierarchického K-means stromu.
- **KDTreeSingleIndexParams** Index s týmto typom vytvorí jeden KD-strom, ktorý je optimalizovaný na vyhľadávanie v nízkodimenzionálnych dátach (napríklad 2D a 3D priestor).
- **KDTreeCuda3dIndexParams** Index s týmto typom vytvorí jeden KD-strom. Vytvorenie a vyhľadávanie v strome prebieha na CUDA kompatibilných grafických kartách. Tento typ indexu sa hodí na vyľadávanie veľkého počtu dotazov.
- **HierarchicalClusteringIndexParams** Index s týmto typom použíje na vyhľadávanie hierarchické zhlukovacie stromy.
- **AutotunedIndexParams** Tento typ indexu je určený pre automatický výber najlepšej vyhľadávacej štruktúry a jej optimálnych parametrov.

Po výbere parametrov sa index vytvorí volaním metódy `buildIndex`. Vytvorenie indexu zvyčajne trvá dlhšiu dobu, preto je možné pre opätovné použitie vytvorený index uložiť do súboru. Objekty sa dajú do indexu pridávať a odoberať aj po jeho vytvorení pomocou metód: `addPoints` a `removePoint`.

Na samotné vyhľadávanie najbližších objektov k dotazom slúžia dve základné metódy:

- **flann::Index::knnSearch** Slúži na vyhľadávanie K-najbližších susedov pre množinu dotazovaných objektov. Metóda má parameter `checks`, ktorý určuje počet listových uzlov, ktoré sa

majú skontrolovať a môže byť nastavená na špeciálne hodnoty:
`FLANN_CHECKS_UNLIMITED` (skontroluje všetky listové uzly),
`FLANN_CHECKS_AUTOTUNED` (použije vypočítanú hodnotu, ak bol index vytvorený automaticky).

- **flann::Index::radiusSearch** Metóda na rozsahové vyhľadávanie najbližších objektov do vzdialenosti polomeru `radius` [16].

4 Porovnanie knižnice FLANN a M-indexu

Našim cieľom je porovnanie výkonu dvoch programovacích knižníc určených na vyhľadávanie najbližších susedov. Meranie výkonu programu zvyčajne zahŕňa niekoľko aspektov. Pre programy na vyhľadávanie najbližších susedov je to:

- celkový čas potrebný na vytvorenie indexu
- čas samotného vyhľadávania
- presnosť vyhľadávania

Pre správnosť a férovosť porovnania je taktiež dôležite dodržať niekoľko zásad:

- Porovnávať výkon indexov podobnej veľkosti, napr. 4GB a 5GB. Zvyčajne nieje možne vytvoriť indexy úplne rovnakej veľkosti. Nie je však vhodné porovnávať 4GB index s 50GB indexom.
- Ak počítame zrýchlenie oproti lineárnemu vyhľadávaniu, ktorá ma presnosť 100% a čas vyhľadávania 100s, nemôžeme povedať, že vyhľadávanie s iným indexom, ktoré má presnosť 50% a trvá 10s má zrýchlenie 10x, pretože presnosť nieje rovnaká ani podobná. Reálne môžme predpokladať, že vyhľadávanie hrubou silou bude trvať 50s, ak ma byť presnosť 50%. Takže index je asi len 5x rýchlejší.
- Pri porovnávaní sa často zabúda na zohľadnenie času, za ktorý sa vytvorí index, aj keď to môže byť jeden z najdôležitejších porovnávacích faktorov.

4.1 Nastavenie parametrov

Testovacie dáta: súbory 100K, 300K, 500K a 1M 4096-dimenzionálnych vektorov, textový formát v zip archíve, každý objekt je uložený na dvoch riadkoch, prvý riadok obsahuje ID objektu(10 miestne číslo) samotný vektor je na ďalšom riadku atď.

Dotazované objekty: 1000 náhodne vybraných objektov z testovaných dát.

Groundtruth(správne výsledky): 1000 najbližších susedov spočítaných pre každý dotaz, osobitne pre všetky testovacie dáta. Formát: 2 riadky pre každý dotaz, prvý riadok ID dotazu, druhý riadok 1000 párov(L2 vzdialenosť od dotazu, ID objektu), výsledky pre každý dotaz sú usporiadané podľa vzdialenosti.

Parametre vyhľadávania:
k=1, 10, 100 ,1000
rôzne nastavenia presnosti vyhľadávania

Merania: Presnosť vyhľadávania určuje:

$$\text{recall} = \frac{\text{odpoveď} \cap \text{groundtruth}}{k}$$

Meriame čas pre jeden dotaz, ako priemer z 1000 dotazov v [ms].

4.2 Výsledky

5 Záver

Literatúra

[1] M. Krátký, T. Skopal, and V. Snáše. Porovnání některých metod pro vyhledávání a indexování multimediálních dat. In *Zbornik konference Kybernetika - história, perspektívy, teória a prax*, pages 118 – 134. Žilinská univerzita v Žiline, 2003.

[2] MultiMedia LLC. Color analysis. `https://www.clear.rice.edu/elec301/Projects02/artSpy/color.html`, 2015.

[3] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[4] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[5] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006.

[6] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.

[7] Wikipedia. Minkowski distance. `http://en.wikipedia.org/wiki/Minkowski_distance`, 2015. [Online; 18-April-2015].

[8] Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible*. PhD thesis, Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2009.

[9] Pavel Jurkas. Fqt: indexování metrických prostoru. `https://is.muni.cz/auth/th/72641/fi_b/bc.pdf`, 2007.

[10] Ashraf M. Kibriya and Eibe Frank. An empirical comparison of exact nearest neighbour algorithms. In *Proc 11th European*

Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, pages 140–151. Springer, 2007.

[11] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[12] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[13] Marius Muja and David G. Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV)*, pages 404–410, 2012.

[14] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[15] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[16] Marius Muja and David G. Lowe. *FLANN - Fast Library for Approximate Nearest Neighbors User Manual*. UBC.