Preferenčné vyhľadávanie v metrických priestoroch

Martin Šumák, Peter Gurský

Ústav informatiky PF, Univerzita Pavla Jozefa Šafárika v Košiciach,
Moyzesova 16, 040 01 Košice
martin.sumak@netkosice.sk, gursky@upjs.sk

Abstrakt. Súčasné algoritmy na vyhľadávanie najlepších *n* objektov nemajú podporu atribútov vyjadrujúcich vzdialenosť objektov od používateľom zvoleného bodu v metrickom priestore. Optimálne vyhľadávacie techniky ako 3P-NRA vyžadujú zasielanie objektov a hodnôt ich atribútov v prúde pre každý atribút v poradí od najlepších po najhoršie. Štandardné typy vyhľadávania nad metrickými priestormi, ako sú rozsahové dopyty alebo kNN dopyty, toto neumožňujú. V tomto článku predstavíme prúdovú verziu metódy najbližších susedov a jej vylepšenie umožňujúce definovať preferenciu nie len pre blízke, ale aj pre stredne vzdialené či vzdialené objekty od zvoleného fixovaného bodu.

Kľúčové slová: M-strom, najlepších *k* objektov, preferencie

1 Úvod

Pri súčasnom raste objemu spracovávaných dát sa čoraz viac dostávajú do pozornosti dopyty, ktoré poskytnú používateľom iba niekoľko pre nich najlepších objektov. Typickým príkladom sú fulltextové vyhľadávače, ktoré zobrazia iba prvých cca 10 najrelevantnejších stránok. Zobrazovanie najlepších *n* objektov sa však presadzuje aj v iných doménach ako sú multimédiá, hotely, reality, pracovné ponuky či elektronika.

To, ktoré objekty sú lepšie, je dané hodnotami ich vlastností (atribútov). Pred začiatkom vyhľadávania je potrebné vedieť, ktoré hodnoty používateľ v každom atribúte preferuje a ktorý atribút je preňho ako dôležitý. Niekedy sa pridáva aj niekoľko ostrých obmedzení na hodnoty atribútov.

Súčasné známe spôsoby vyhľadávania najlepších *n* objektov sa sústreďujú na spracovanie ordinálnych (prirodzene zotriediteľných) atribútov ako sú cena, veľkosť, množstvo alebo kvalita [2,3,4,5]. Ostatné typy atribútov majú iba selektívnu funkciu (ostré obmedzenie) [8] napr. byty v Bratislave, hotely v Tatrách, fotoaparáty v čiernej farbe a pod. V tomto článku je našim cieľom schopnosť vyjadriť preferencie ako „blízko Štrbského Plesa“, „zhruba 5 až 10 km od centra Košíc“ alebo „ďaleko od New Yorku“.

Veľmi efektívnym algoritmom na vyhľadávanie najlepších *n* objektov je 3P-NRA[3]. Tento algoritmus číta niekoľko prúdov dát – za každý atribút jeden. Každý z prúdov zasiela ohodnotené identifikátory objektov zotriedené podľa preferencie používateľa pre daný atribút. Na to aby našiel najlepších *n* objektov prečíta z jednotlivých prúdov dát dopredu neznáme počty objektov a ich ohodnotení v príslušnom atribúte.

V tomto článku predstavujeme spôsob, ktorým sa dá vyhľadávanie najlepších *n* objektov rozšíriť o spracovanie metrických atribútov. Metrickým atribútom je

ohodnotenie (metrickej) vzdialenosti objektov od používateľom zvoleného bodu, ktorý budeme nazývať kotvový bod. Na to, aby sme to vedeli zabezpečiť, potrebujeme vedieť poskytnúť algoritmu 3P-NRA prúd objektov zotriedený od najlepších po najhoršie v danom metrickom atribúte. Tento typ atribútov nie je vo všeobecnosti prirodzene zotriediteľný. Zotriedenie je možné až v čase otázky, keď sa dozvieme preferencie používateľa. Triedenie všetkých objektov podľa metrického atribútu je pri väčších dátach porovnateľné so zotriedením všetkých objektov podľa celkovej používateľovej preferencie zahrňujúcej aj preferencie k hodnotám ostatných atribútov.

Namiesto toho využijeme indexovaciu štruktúru M-strom popísanú v kapitole 2 a predstavíme algoritmy, pomocou ktorých sa dá vytvárať prúd zotriedených objektov (sekcie 2.1 a 2.2). Naším hlavným cieľom je minimalizovanie času spracovania dát a minimalizovanie počtu prístupov na disk ku štruktúre M-stromu pri návrate iba prvých k objektov a to aj bez znalosti čísla k dopredu. Minimalizovanie počtu prístupov je užitočné z toho dôvodu, že algoritmus na hľadanie najlepších n objektov takmer vždy ukončí svoj beh ešte pred tým, než využije všetky dáta z prúdov. Málo preferované objekty sa tak vôbec nemusia spracovávať.

Nad M-stromom sú typické dva druhy dopytov [1, 7]. Prvý je rozsahový dopyt, ktorý vráti nezotriedenú množinu bodov nachádzajúcich sa v danej viacrozmernej guli. Druhým typom je kNN dopyt (kNN = k najbližších susedov), ktorý vráti zoznam k bodov zotriedených od najbližšieho po najvzdialenejší od daného kotvového bodu (popis kNN je v sekcii 2.3). Žiaľ žiaden z týchto dopytov nie je vhodný pre naše účely. kNN dopyt by sa dal použiť pri preferovaní bodov blízkych ku danému bodu, akurát že číslo k musí byť známe dopredu. Na simulovanie prúdu dát bez známej využiteľnej dĺžky by sme museli za k zvoliť počet všetkých bodov, čo by nesplňovalo našu požiadavku pre minimalizovanie počtu prístupov.

Zasielanie prúdu dát už bolo vyriešené pre R-stromy [6]. Tento algoritmus však neumožňuje definíciu preferencií pre iné ako najbližšie objekty ku kotvovému bodu. Ostatné typy indexov nad metrickými atribútmi sme neskúmali.

Aby sme overili efektivitu našich prístupov, v kapitole 3 uvádzame výsledky experimentov nad 6 kolekciami umelo vytvorených bodov s rôznym rozdelením a 1 kolekciou reálnych dát predstavujúcich GPS súradnice priradených k ponukám na predaj bytov alebo domov owrapovaných zo stránky www.byt.sk.

Hlavnými prínosmi tohto článku sú:

- Vytvorenie algoritmu Sort query nad M-stromom umožňujúceho vytvárať prúd bodov metrického priestoru od najbližších po najvzdialenejšie. Ďalej jeho vylepšenú verziu t.j. algoritmus Fuzzy sort query, pomocou ktorého získavame body (objekty) v poradí od najlepších po najhoršie podľa používateľových preferencií vyjadrených fuzzy funkciou ohodnocujúcou vzdialenosť objektu od kotvového bodu. Obidva algoritmy minimalizujú počet prístupov vzhľadom na počet bodov spracovaných z prúdu.
- Algoritmus SortQuery experimentálne porovnávame s algoritmom kNN (k najbližších susedov) nad 6 kolekciami dát s rôznym rozdelením a 1 kolekciou reálnych dát.

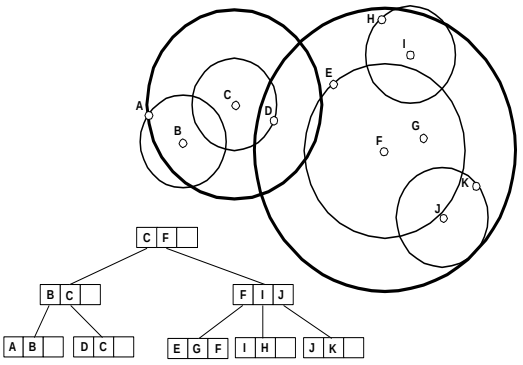
2 M-strom

M-strom je indexovacia štruktúra navrhnutá pre uchovávanie objektov, ktoré môžeme reprezentovať bodmi v metrickom priestore. M-strom poskytuje nezávislosť od metrického priestoru, ktorého body (objekty) chceme v ňom uchovávať. Ak chceme použiť na indexovanie objektov M-strom, stačí ak poznáme funkciu na počítanie vzdialenosti medzi ľubovoľnými dvoma objektmi, ktorá je metrikou. Výhodou tohto prístupu je napríklad aj to, že za metriku môžeme považovať nielen klasickú vzdialenosť v euklidovskom priestore ale aj napríklad čas potrebný na presun medzi každými dvoma bodmi. Konkrétnou metrikou, ktorú M-stromu poskytneme a typom objektov, ktoré budeme v strome uchovávať definujeme metrický priestor.

M-strom je štruktúrou, ktorá poskytuje stránkovanie jednotlivých uzlov na pevný disk, dynamické pridávanie nových objektov a navyše je vyvážený. Všetky indexované objekty sú v M-strome udržované v listoch, ostatné (vnútorné) uzly stromu slúžia na hierarchické zoskupovanie uzlov nižšej úrovne do samostatných celkov – uzlov vyššej úrovne. Každý uzol M-stromu si môžeme predstaviť ako guľu daného metrického priestoru. Listy obsahujú body uchovávané v strome, vnútorné uzly obsahujú uzly nižšej úrovne. Uzly majú jednotnú kapacitu a jednotné minimálne zaplnenie, ktoré musia dosahovať všetky uzly s výnimkou koreňa stromu. Formálne môžeme uzly popísať nasledujúcim spôsobom:

list: $\mathbf{N} = \langle c(\mathbf{N}), r(\mathbf{N}), p(P(\mathbf{N})), \{\langle O_i, d(O_i, c(\mathbf{N}))\rangle : i \in \{1, 2, \ldots, q(\mathbf{N})\}\}\rangle$
vnútorný uzol: $\mathbf{N} = \langle c(\mathbf{N}), r(\mathbf{N}), p(P(\mathbf{N})), \{\langle c(\mathbf{N}_i), d(c(\mathbf{N}_i), c(\mathbf{N})), r(\mathbf{N}_i), p(\mathbf{N}_i)\rangle : i \in \{1, 2, \ldots, q(\mathbf{N})\}\}\rangle$

kde **N** je uzol, c(**N**) je stred uzla vybraný pre list spomedzi jeho objektov O_i a pre vnútorný uzol spomedzi stredov jeho potomkov c($\mathbf{N}_i$). P(**N**) je rodičovský uzol uzla **N**, p(**N**) je smerník na uzol **N**, d je metrika (vzdialenostná funkcia medzi objektami) a r(**N**) je polomer uzla **N**. q(**N**) je počet objektov listu **N**, resp. pre vnútorný uzol počet poduzlov uzla **N**. Každý uzol teda udržiava ku všetkým svojim objektom aj ich vzdialenosť od svojho stredu. Vnútorné uzly udržiavajú pre každý svoj poduzol ešte navyše smerník na daný poduzol a jeho polomer. Príklad M-stromu pre dvojrozmerný euklidovský priestor znázorňuje obrázok 1.



Obr. 1. Príklad M-stromu a grafické znázornenie bodov a uzlov

Ako sme už spomínali v úvode, na to, aby sme mohli efektívne pracovať s metrickými atribútmi pri hľadaní najlepších *n* objektov, potrebujeme generovať prúd bodov od najlepších po najhoršie podľa preferencie daného používateľa. Uvedieme dve verzie algoritmu Sort query, ktorý takýto prúd generuje. Prvá verzia umožňuje vytvárať prúd bodov od najbližších po najvzdialenejšie od kotvového bodu (napr. hotely blízko pracovného stretnutia, byty blízko pracoviska a pod.). Pomocou druhej verzie t.j. Fuzzy sort query je možné definovať aj preferencie pre ľubovoľné vzdialenosti od kotvového bodu (byty blízko mesta ale radšej ďalej od hlučného centra alebo hotel ďaleko od letiska).

2.1 Sort query

Algoritmus Sort query má jediný vstup a to ľubovoľný bod daného metrického priestoru, t.j. kotvový bod, ktorý budeme označovať Q. Cieľom algoritmu je postupne vrátiť na výstup body uložené v strome v poradí od najbližšieho po najvzdialenejší od Q. Algoritmus sa skladá z dvoch fáz: inicializačnej a iteračnej. Inicializačná fáza naštartuje výpočet, a iteračná fáza vráti každým svojím zavolaním jeden bod zo stromu. Prvé volanie iteračnej fázy vráti najbližší bod stromu od Q. Druhé volanie iteračnej fázy vráti druhý najbližší bod atď. Iteračnú fázu môžeme volať dovtedy, kým postupne nevráti v danom poradí všetky body stromu. Algoritmus v iteračnej fáze vždy prečíta minimálny nutný počet uzlov stromu, aby dokázal korektne vrátiť nasledujúci objekt v správnom poradí podľa vzdialenosti. Je zrejmé, že po prvých *k* volaniach iteračnej fázy získame postupne rovnaký výstup, ktorý vráti dopyt kNN.

Pre popísanie obidvoch fáz algoritmu zavedieme pojem pracovný zoznam, ktorý algoritmus pre svoj výpočet používa. Pracovný zoznam obsahuje prvky dvojakého typu:

bodový prvok: $\langle O, d(O, Q), d(O, Q)\rangle$
uzlový prvok: $\langle p(\mathbf{N}), \max\{d(c(\mathbf{N}), Q) - r(\mathbf{N}), 0\}, d(c(\mathbf{N}), Q) + r(\mathbf{N})\rangle$

Kde O je ľubovoľný bod v strome, **N** je ľubovoľný uzol stromu a Q je kotvový bod. Bodovým prvkom pracovného zoznamu je teda usporiadaná trojica: bod stromu O a jeho minimálna a maximálna vzdialenosť od kotvového bodu Q. U bodového prvku je druhá zložka zhodná s treťou. Uzlovým prvkom je usporiadaná trojica: smerník na uzol **N**, minimálna a maximálna vzdialenosť kruhu uzla **N** od bodu Q. Minimálna vzdialenosť je najmenšia možná vzdialenosť od kotvového bodu, ktorú môže nejaký bod podstromu uzla **N** dosiahnuť. Analogický význam má maximálna vzdialenosť. V popise algoritmu je uvedené, že vkladanie prvkov do pracovného zoznamu má byť vykonané tak, aby bol zoznam zotriedený. Presne to znamená, že prvky majú byť usporiadané vzostupne podľa minimálnej vzdialenosti od kotvového bodu Q. Prvky s rovnakou minimálnou vzdialenosťou majú byť medzi sebou usporiadané vzostupne podľa maximálnej vzdialenosti. Ešte treba dodať (pre špeciálny prípad nulového polomeru uzla), že usporiadanie má byť také, aby bodové prvky boli umiestnené v zozname vždy pred uzlovými prvkami s rovnakou minimálnou vzdialenosťou.

Algoritmus Sort query:
Inicializačná fáza:

```
sortQueryInitialization(kotvovýBod Q)
   1.  Nech pracovný zoznam je prázdny.
   2.  Nech uzol N je koreň stromu.
   3.  Vlož do pracovného zoznamu prvok <p(N), 0, 0>
```

Iteračná fáza:

```
sortQueryNext()
   1. Nech trojica <x, y, z> je prvý prvok pracovného zoznamu,
      odober z neho tento prvok.
   2. Ak x je objekt vráť na výstup objekt x a hodnotu y a
      skonči.
   3. Ak x je smerník na uzol
      a. Nech N je uzol, na ktorý ukazuje smerník x, načítaj
         uzol N z disku.
      b. Ak N je list, tak pre každý objekt Oi listu N pridaj do
         pracovného zoznamu prvok <Oi, d(Oi, Q), d(Oi, Q)> tak,
         aby bol pracovný zoznam zotriedený.
      c. Ak N je vnútorný uzol, tak pre každý poduzol Ni uzla N
         pridaj do pracovného zoznamu prvok
          <p(Ni), max{d(c(Ni),Q)-r(Ni), 0}, d(c(Ni),Q) + r(Ni)>
         tak, aby bol pracovný zoznam zotriedený.
      d. Pokračuj krokom č. 1.
```

Korektnosť algoritmu je zaručená spôsobom práce s pracovným zoznamom. Inicializačná fáza algoritmu zaručí, že pracovný zoznam berie do úvahy všetky body v strome. V iteračnej fáze, ak dôjde k odobratiu uzlového prvku z pracovného zoznamu, krok 3.b. resp 3.c. iteračnej fázy opäť zaručí, že pracovný zoznam berie do úvahy všetky body v strome, ktoré ešte neboli vrátené na výstup krokom 2 iteračnej fázy. Poradie prvkov pracovného zoznamu zase zaručí, že bod vrátený v kroku 2 iteračnej fázy je spomedzi všetkých bodov stromu, ktoré neboli ešte týmto krokom vrátené, najbližšie ku kotvovému bodu Q.

2.2 Fuzzy sort query

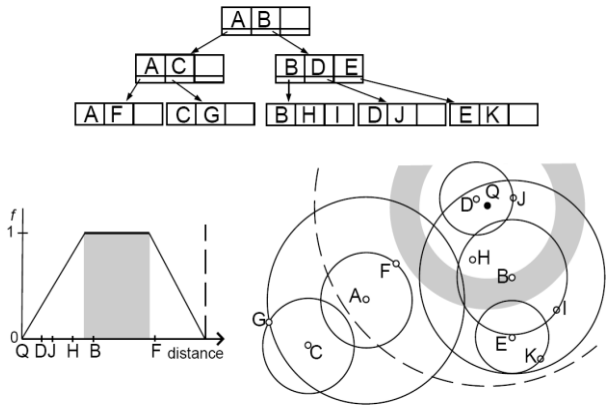
Jemnou modifikáciou prvkov pracovného zoznamu dosiahneme, že algoritmus Sort query bude vracať body stromu v poradí od najlepšieho po najhorší podľa ohodnotenia ich vzdialenosti od kotvového bodu fuzzy funkciou. Fuzzy funkcia je prostriedkom na vyjadrenie preferencií k vzdialenostiam od kotvového bodu. Jej definičným oborom je vzdialenosť od kotvového bodu a oborom hodnôt čísla v intervale [0,1] kde 1 znamená veľmi preferovanú hodnotu a 0 nepreferovanú (alebo neprijateľnú) hodnotu. Nech teda pracovný zoznam obsahuje takto modifikované prvky:

bodový prvok: $\langle O, f(d(O, Q)), f(d(O, Q))\rangle$
uzlový prvok: $\langle p(\mathbf{N}), f_{best}(d(c(\mathbf{N}), Q) - r(\mathbf{N}), d(c(\mathbf{N}), Q) + r(\mathbf{N})),$
$f_{worst}(d(c(\mathbf{N}), Q) - r(\mathbf{N}), d(c(\mathbf{N}), Q) + r(\mathbf{N}))\rangle$

kde funkcia f je príslušná fuzzy funkcia a počíta fuzzy hodnotu vzdialenosti od kotvového bodu. Funkcia f_{best} vyjadruje maximum fuzzy funkcie na intervale a funkcia f_{worst} počíta minimum fuzzy funkcie na intervale. Prvá zložka prvkov ostala nezmenená. Druhá a tretia zložka ostala nezmenená len významovo. Sú to v poradí najlepšia a najhoršia fuzzy hodnota vzdialenosti bodu resp. uzla od kotvového bodu. Druhá a tretia zložka je u bodového prvku fuzzy hodnota vzdialenosti objektu O od kotvového bodu Q. U uzlového prvku je druhá zložka najlepšia (najväčšia) fuzzy hodnota vzdialenosti od Q, ktorú môže nejaký bod podstromu uzla **N** dosiahnuť. Tretia zložka je u uzlového prvku najhoršia (najmenšia) fuzzy hodnota vzdialenosti od Q, ktorú môže nejaký bod podstromu uzla **N** dosiahnuť.

Keďže chceme posielať body od najlepších po najhoršie, tak najlepšie body už nie sú vo všeobecnosti najbližšie, ale body O vyššou hodnotou f(d(O, Q)). Pracovný zoznam bude preto udržiavaný v opačnom, teda zostupnom poradí podľa druhej zložky svojich prvkov (v prípade jednoduchého Sort query boli lepšie body s menšou vzdialenosťou od Q, teraz sú lepšie body s väčšou fuzzy hodnotou). V rámci rovnakej druhej zložky budú prvky usporiadané tiež zostupne podľa tretej zložky. Opäť je potrebné zabezpečiť to, aby bodové prvky boli v zozname umiestnené pred uzlovými prvkami s rovnakou druhou zložkou. Pri použití fuzzy funkcií, ktoré sú konštantné na relatívne dlhých intervaloch, má táto požiadavka veľký význam. Vtedy totiž môže nastať to, že množstvo bodových a uzlových prvkov sa bude zhodovať v druhej aj v tretej zložke.

Korektnosť algoritmu s použitím fuzzy funkcie je opäť zachovaná vďaka poradiu, v akom sú jednotlivé prvky pracovného zoznamu udržiavané. Fuzzy funkcia môže byť pritom ľubovoľná, nie sú na ňu pre použitie vo Fuzzy sort query kladené žiadne obmedzenia.



Obr 2. Preferencia pre "strednú" vzdialenosť od kotvového bodu Q

Príklad 1. Pre ilustráciu fungovania algoritmu Fuzzy sort query uvažujme situáciu na obrázku 2. V M-strome je uložených 11 objektov A až K. Vstupom pre algoritmus je kotvový bod Q a fuzzy funkcia f s maximálnou hodnotou na intervale zvýraznenom sivou farbou a nulovou hodnotou za čiarkovanou čiarou. Pre jednoduchosť nazvime uzly rovnakým písmenom ako označujeme ich stredy a pridajme im index úrovne, na ktorej sa v strome nachádzajú. Nech koreň stromu je na úrovni číslo 1. Po inicializačnej fáze je zrejmé, že pracovný zoznam bude vyzerať nasledovne: [<$p(A_2)$,

1.0, 0.0>, <p(B_2), 1.0, 0.0>]. Obidva prvky zoznamu sa zhodujú v druhej a tretej zložke, teda ich poradie môže byť ľubovoľné. Prvé volanie sortQueryNext() vykoná nasledujúce kroky. Odoberie prvok <p(A_2), 1.0, 0.0> z pracovného zoznamu. Prvou zložkou odobraného prvku je smerník, teda do pracovného zoznamu pribudnú na správne miesto podľa poradia nové prvky a pracovný zoznam bude vyzerať takto: [<p(B_2), 1.0, 0.0>, <p(A_3), 0.9, 0.0>, <p(C_3), 0.0, 0.0>]. Opäť odoberieme prvý prvok z pracovného zoznamu a keďže ani teraz jeho prvá zložka ešte nie je bod tak po pridaní nových prvkov bude pracovný zoznam vyzerať nasledovne: [<p(B_3), 1.0, 0.2>, <p(E_3), 1.0, 0.1>, <p(A_3), 0.9, 0.0>, <p(D_3), 0.8, 0.0>, <p(C_3), 0.0, 0.0>]. Keďže prvá zložka prvého prvku pracovného zoznamu stále nie je bod, cyklus sa opakuje a dostaneme nasledujúci pracovný zoznam: [<B, 1.0, 1.0>, <p(E_3), 1.0, 0.1>, <p(A_3), 0.9, 0.0>, <H, 0.8, 0.8>, <p(D_3), 0.8, 0.0>, <I, 0.7, 0.7>, <p(C_3), 0.0, 0.0>]. Teraz odoberieme prvý prvok z pracovného zoznamu a keďže jeho prvá zložka je objekt B, vrátime ho na výstup aj s jeho hodnotou 1.0.

2.3 kNN query

Pri experimentovaní sme potrebovali naše nové algoritmy porovnať s nejakým známym algoritmom. Ako sme už spomínali, výstup totožný s algoritmom Sort query generuje aj štandardný algoritmus kNN. Jediným rozdielom je to, že kNN má dopredu dané, koľko objektov je potrebných dať na výstup. V algoritme Sort query to znamená k zavolaní funkcie `sortQueryNext()`. Aby si čitateľ urobil komplexnú predstavu uvedieme aj pôvodný algoritmus kNN.

Algoritmus kNN má dva vstupy: kotvový bod Q a prirodzené číslo *k*. Výstupom algoritmu je zoznam *k* najbližších bodov v strome voči kotvovému bodu Q zotriedených podľa ich vzdialenosti od Q. Algoritmus používa dva zoznamy. Nazvime ich výsledný a pracovný zoznam. Výsledný zoznam má pevne stanovený počet prvkov na k a udržiava prvky dvojakého typu:

bodový prvok: <O, d(O, Q)>
uzlový prvok: < p(**N**), d(c(**N**), Q) + r(**N**)>

kde O je ľubovoľný bod v strome, Q je kotvový bod a **N** je ľubovoľný uzol stromu. Bodový prvok je usporiadaná dvojica skladajúca sa z bodu a jeho vzdialenosti ku Q. Uzlový prvok je usporiadaná dvojica, ktorej prvá zložka je smerník na uzol **N** a druhá zložka je maximálna vzdialenosť od Q, ktorú môže nejaký bod podstromu uzla **N** dosiahnuť.

Pracovný zoznam obsahuje prvky jedného typu:

uzlový prvok: <p(**N**), max{d(c(**N**), Q) – r(**N**), 0}>

kde **N** je ľubovoľný uzol stromu a Q je kotvový bod. Prvkom pracovného zoznamu je usporiadaná dvojica: smerník na uzol **N** a minimálna vzdialenosť od Q, ktorú môže nejaký bod podstromu uzla **N** dosiahnuť.

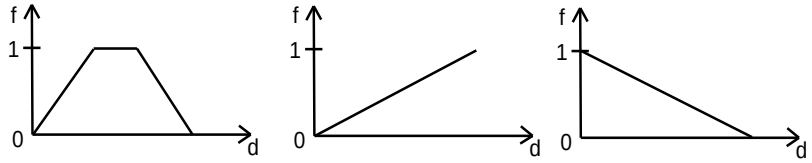
Obidva zoznamy budú v priebehu výpočtu udržiavané zotriedené vzostupne podľa druhej zložky svojich prvkov.

```
kNearestNeighboursQuery(kotvovýBod Q)
  1. Inicializuj výsledný zoznam k prvkami <null,∞>.
  2. Nech uzol N je koreň stromu.
  3. Vlož do pracovného zoznamu prvok <p(N),∞>.
  4. Pokiaľ pracovný zoznam nie je prázdny rob:
    a.Nech dvojica <x, y> je prvý prvok pracovného zoznamu,
      odober z neho tento prvok.
    b.Ak výsledný zoznam obsahuje prvok <x, z>, kde z je
      ľubovoľné číslo, odober prvok <x, z> z výsledného
      zoznamu a pridaj do výsledného zoznamu prvok <null, ∞> na
      poslednú (k-tu) pozíciu.
    c.Nech U je uzol, na ktorý ukazuje smerník x, načítaj uzol
      U z disku.
    d.Ak U je list tak pre každý objekt O_i listu U urob:
      I.Nech dvojica <v, w> je práve k-ty prvok výsledného
        zoznamu. Ak (|d(Q, c(U)) − d(O_i, c(U))| ≤ w) a d(O_i, Q)
        ≤ w tak:
         1.odober prvok <v, w> z výsledného zoznamu,
         2.vlož do výsledného zoznamu prvok <O_i, d(O_i, Q)> tak,
           aby bol výsledný zoznam zotriedený.
         3.Nech dvojica <v, w> je práve k-ty prvok výsledného
           zoznamu. Odober z pracovného zoznamu všetky prvky
           <a, b>, pre ktoré platí: b > w.
    e.Ak U je vnútorný uzol tak pre každý jeho poduzol U_i urob:
      I.Nech dvojica <v, w> je práve k-ty prvok výsledného
        zoznamu. Ak (|d(Q, c(U)) − d(c(U_i), c(U))| ≤ w) a
        max{d(c(U_i), Q) − r(U_i), 0} ≤ w tak:
         1.Vlož do pracovného zoznamu prvok <p(U_i),
           max{d(c(U_i), Q) − r(U_i), 0}> tak, aby bol pracovný
           zoznam zotriedený.
         2.Ak d(c(U_i), Q) + r(U_i) ≤ w:
           a.Odober prvok <v, w> z výsledného zoznamu.
           b.Vlož do výsledného zoznamu prvok <p(U_i), d(c(U_i),
             Q) + r(U_i)> tak, aby bol výsledný zoznam
             zotriedený.
           c.Nech dvojica <v, w> je práve k-ty prvok výsledného
             zoznamu. Odober z pracovného zoznamu všetky prvky
             <a, b>, pre ktoré platí: b > w.
  5. Vráť výsledný zoznam na výstup a skonči.
```

Poznamenajme, že aj z algoritmu kNN by sa dalo urobiť rozšírenie, ktoré by počítalo *k* bodov stromu v poradí od najlepšieho po najhorší podľa preferencií k vzdialenostiam definovaných fuzzy funkciou. My však k hľadaniu najlepších *k* objektov potrebujeme prúdovú verziu, takže sa tejto modifikácii venovať nebudeme.

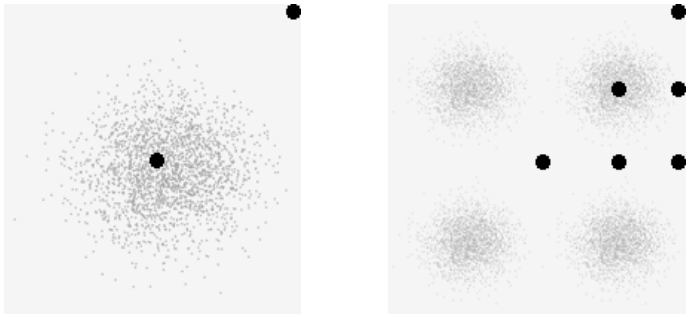
3 Experimenty

V testoch sme merali čas výpočtu algoritmov. Porovnávali sme kNN query, Sort query a Fuzzy sort query. U Fuzzy sort query sme použili postupne 3 fuzzy funkcie znázornené na obrázku 3.



Obr. 3. Fuzzy funkcie kopec, rastúca a klesajúca použité v dopytoch

Testy algoritmov sme vykonali dohromady na 7 rôznych sadách bodov. Šesť z nich bolo vygenerovaných, jedna sada obsahovala reálne dáta.



Obr. 4. Rozloženie dát pri normálnych rozdeleniach v 2D a kotvové body dopytov

Popíšme najprv vygenerované dáta. Dve sady obsahovali náhodne vygenerované body s rovnomerným rozložením pravdepodobnosti, pričom jedna obsahovala body z dvojrozmerného euklidovského priestoru, druhá z trojrozmerného euklidovského priestoru. Ďalšie dve sady bodov boli vygenerované s normálnym rozložením, jedna pre dvojrozmerný priestor, druhá pre trojrozmerný. Posledné dve sady vygenerovaných dát sa tiež líšia len dimenziou. Obsahovali body vygenerované spojením štyroch resp. ôsmych normálnych rozložení ako znázorňuje obrázok 4. Všetky vygenerované sady obsahovali 200000 bodov.

Reálne dáta tvorilo 18743 bodov na zemskom povrchu. Všetky tieto body predstavovali byty na území Slovenskej republiky, prevažne vo väčších mestách.

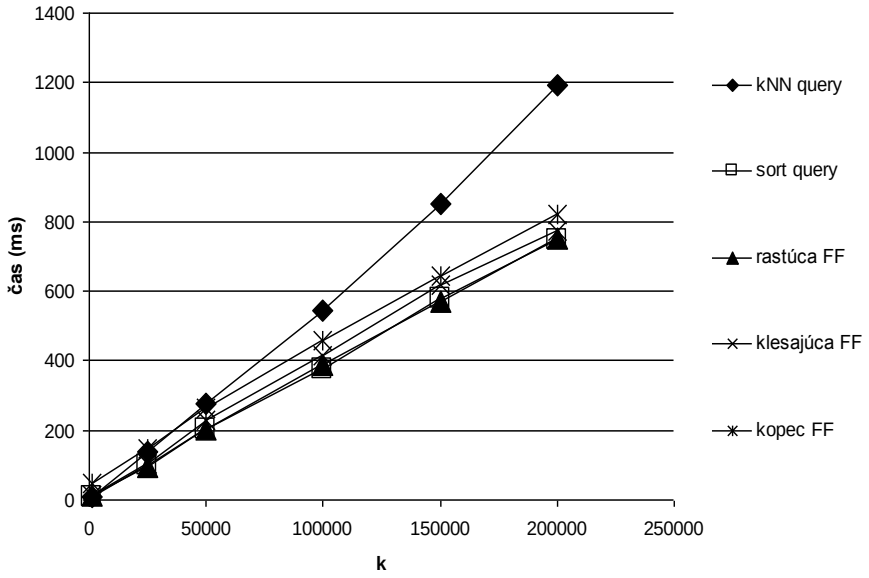
Pre body dvojrozmerného priestoru a body reálnych dát bola použitá kapacita uzla 2048 bytov, pre body trojrozmerného priestoru 2560 čo znamenalo kapacitu uzlov zhruba 70 bodov.

Body boli do stromu pridávané spôsobom „multi way leaf choice“ popísanom v [7]. Po pridaní všetkých bodov do stromu bol na strom aplikovaný algoritmus „generalized slim down“ popísaný v [7], ktorý zlepší kvalitu indexu.

Uvedené zoznamy, ktoré algoritmy používajú sú implementované javovskou triedou TreeSet, ktorá zaručuje usporiadanosť a logaritmickú zložitosť pridávania a vyhľadávania objektov.

Pre rovnomerne vygenerované dáta bol použitý jeden kotvový bod. Pre dáta s normálnymi rozloženiami sú použité kotvové body znázornené na obrázku 4. U reálnych dát boli použité tri kotvové objekty a to mestá Bratislava, Banská Bystrica a Košice. Pre každú sadu dát a zvolený kotvový bod bolo vykonaných päť sérií dopytov.

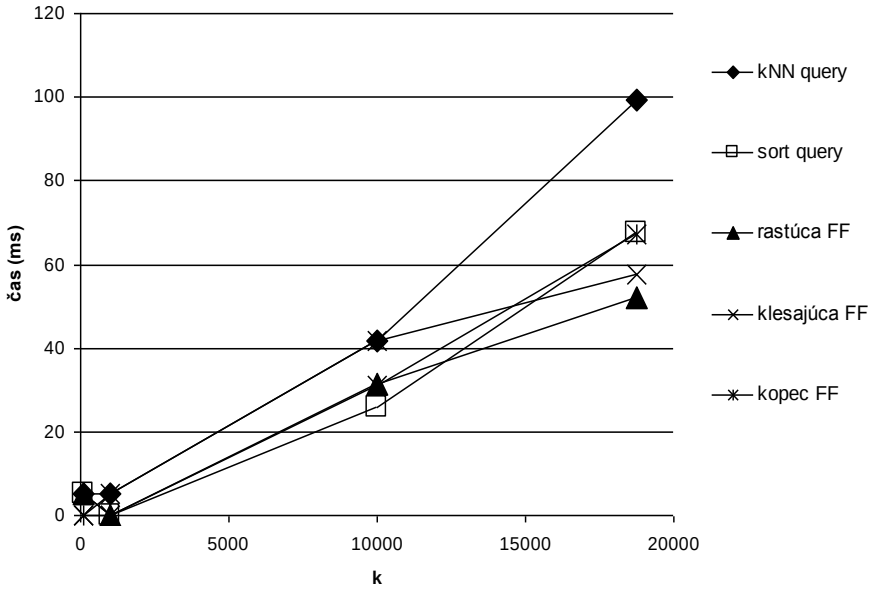
Postupne boli vykonané série dopytov pre kNN query, Sort query a Fuzzy sort query s troma spomínanými fuzzy funkciami. Série dopytov nad vygenerovanými dátami obsahujú 6 dopytov líšiacich sa počtom získaných bodov. Ten bol postupne 1000, 25000, 50000, 100000, 150000, 200000. Série dopytov nad reálnymi dátami obsahovali 4 dopyty postupne pre počty získaných bodov 100, 1000, 10000, 18743.



Obr. 5. Priemerné časy nad vygenerovanými dátami

Dohromady bolo vykonaných 540 dopytov nad vygenerovanými dátami a 60 dopytov nad reálnymi dátami. Obrázok 5 znázorňuje priemerné časy výpočtu všetkých dopytov daného algoritmu nad vygenerovanými dátami v závislosti od počtu získaných bodov. Podobne obrázok 6 obsahuje priemerné časy výpočtov nad reálnymi dátami. Konkrétne časy použité k výpočtu priemerných hodnôt sa vo všetkých prípadoch líšili len minimálne resp. nepodstatne. Môžeme pozorovať, že algoritmus Sort query je vo všetkých svojich variantoch rýchlejší ako algoritmus kNN query. Je to spôsobené tým, že algoritmus kNN query na rozdiel od Sort query pracuje s dvoma zoznamami. Prečítanie vnútorných uzlov často spôsobuje zápis nových prvkov do oboch zoznamov (kroky 4.e.I.1 a 4.e.I.2.b v kNN) a vyhľadávanie vo výslednom zozname (krok 4.b).

Počet prístupov na disk bol u algoritmov kNN query a Sort query za rovnakých podmienok zhodný. U algoritmu Fuzzy sort query s použitím uvedených fuzzy funkcií neboli v počte prístupov na disk pozorované žiadne relevantné rozdiely voči algoritmom kNN resp. Sort query.



Obr. 6. Priemerné časy nad reálnymi dátami

4 Záver

V tomto článku sme sa venovali podpore metrických atribútov pri vyhľadávaní najlepších *n* objektov. Náš nový prístup tak umožňuje pracovať s expresívnejšími používateľskými preferenciami, v ktorých už metrické atribúty nepôsobia iba ako ostré obmedzenia. Aby sme to dokázali, prezentovali sme algoritmus Sort query a jeho fuzzy rozšírenie, ktoré poskytujú prúd bodov od najlepších po najhoršie vzhľadom na používateľove preferencie k vzdialenosti od kotvového bodu.

V experimentoch na umelých aj reálnych dátach sme ukázali, že aj zložitejšie preferencie (t.j. fuzzy funkcie) nespôsobujú takmer žiadne oneskorenie pri generovaní prúdu. Algoritmus Sort query bol v našich testoch rýchlejší ako algoritmus kNN, ktorý generuje rovnaký výstup (aj keď kNN je schopný vrátiť iba presne *k* bodov).

Práca bola čiastočne podporená slovenskými projektmi NAZOU a VEGA 1/3129/06.

Referencie

1. Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In Proceedings of the 23rd Athens Intern. Conf. on VLDB, pages 426–435. Morgan Kaufmann, 1997
2. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In Proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
3. Gurský P.: Algoritmy na vyhľadávanie najlepších *k* objektov bez priameho prístupu. Proceedings of Znalosti 2006, pages 95-105, 2006
4. Gurský P., Horváth T., Novotný R., Vaneková V., Vojtáš P.: UPRE: User preference based search system. In IEEE WI 2006
5. Gurský P., Vojtáš P.: Multikriteriálne vyhľadávanie najlepších objektov s podporou viacerých užívateľov. Proceedings of Znalosti 2007.
6. Hjaltason G., Samet H.: Incremental Distance Join Algorithms for Spatial Databases, Proceedings of the 1998 ACM SIGMOD Intl. Conference on Management of Data, Seattle,WA, June 1998, pp. 237–248
7. Skopal T.: Metric Indexing in Information Retrieval, Ph.D. thesis, VŠB-Technical University of Ostrava, 2004
8. Xin D., Han J, Chang K.: Progressive and Selective Merge: Computing Top-K with Ad-Hoc Ranking Functions. In SIGMOD 2007.
9. Skopal T., Pokorný J., Krátký M., Snášel V.: Revisiting M-Tree Building Principles. In ADBIS 2003

Annotation:

A preference search in metric spaces

Current algorithms of searching the top *n* objects include no support for attributes that express object's distance from user defined point in metric space. Optimal search techniques like 3P-NRA require for each attribute a stream of objects ranked from the best to the worst with respect to user preferences. Standard algorithms over metric spaces for range queries or kNN queries do not have a stream as a result. In this paper we propose the stream version of nearest neighbours queries and its improvement with support of the preferences for middle distance objects as well as for far objects from the fixed anchor point.