Adaptív dinamikus szegmentálás idősorok indexeléséhez

Balassi Márton (FW8FCC)

Englert Péter (PKOAMN)

Tömösy Péter (DAMHR6)

2013. október 22.

Absztrakt

Egy kihívásokkal teli és aktív kutatási területtel, idősorok indexelésével foglalkozik Yang Wang, Peng Wang, Jian Pei, Wei Wang és Sheng Huang A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series című publikációja, melyet a 2013-as Very Large Databases konferencián mutattak be. A cikkben vizsgált fő probléma adatsorok hasonlósági keresése. Ennek hatékony megvalósítására egy DSTree nevű új indexet írnak le, amely jelentősen eltér az eddigi megközelítésektől. Az adatsorok dimenzióját ezek jellemzően egy globális, minden idősornál és indexnél azonos szegmentálás alapján végzik. Az új megközelítés ezzel szemben dinamikusan, adaptív módon alakítja ki a szegmenseket, továbbá az indexet is újszerűen, úgynevezett hasítási stratégiákat alkalmazva építi fel. Ennek elméleti előnyeit kiterjedt empirikus vizsgálatokkal, teszteléssel és mérésekkel támasztják alá. Ezt a cikket szeretnénk most kontextusban elhelyezni, összefoglalni, majd javaslatokat tenni további lehetséges kutatási irányokra.

Tartalomjegyzék

1. Bevezetés

Az idősorok időben egymás után mért értékek sorozatai. Rengeteg különböző tudományterületen találkozhatunk velük, orvosi alkalmazásoktól (EKG, EEG) kezdve szenzorhálózatokon keresztül a pénzügyi alkalmazásokig (tőzsde). Manapság nap mint nap hatalmas mennyiségű ilyen jellegű adat termelődik a világban. Nem meglepő tehát, hogy népszerű, és sok kihívást rejtő kutatási terület az idősorok tárolása és elemzése. Az első igazán jelentős cikkek a témában az 1990-es évek közepén jelentek meg [6], az innen számított az első évtized kutatási eredményeiről már kiváló összefoglalók születtek [8], azonban még ma is aktív kutatási terület.

Ha különböző megoldásokat szeretnénk összehasonlítani, ahhoz először is tudnunk kell, hogy milyen elemzéseket, lekérdezéseket szeretnénk az idősorokon végrehajtani. Sok alkalmazásban van szükség például hasonlósági keresésre. Ehhez először definiálnunk kell egy $D(\cdot,\cdot)$ hasonlósági mértéket két idősor között, ez jellemzően az euklideszi távolság. A feladat ekkor idősorok egy adott $\mathscr{TS}$ halmazán, adott Q idősor és $0 < \varepsilon \in \mathbb{R}$ érték mellett, hogy megkeressük azokat az S idősorokat, melyekre $D(Q,S) \leq \varepsilon$. Ha a hasonlósági mérték az euklideszi távolság, az idősorok pedig azonos hosszúságúak, akkor ezt a feladatot teljes illesztésnek (*whole matching*) hívjuk. Ha a Q idősor rövidebb az adatbázisban ($\mathscr{TS}$ halmaz) találhatóaknál, és olyan idősorokat keresünk, amelyeknek valamely részsorozata ε távolságon belül van Q-tól, akkor a részsorozat illesztés (*subsequence matching*) problémáról beszélünk.

Ez az összefoglaló Yang Wang, Peng Wang, Jian Pei, Wei Wang és Sheng Huang A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series című publikációja [13] alapján készült, amelyet a 2013-as Very Large Databases konferencián mutattak be. A továbbiakban erről a cikkről lesz szó. Témája egy adatszerkezet idősorok indexelésére, amelynek segítségével hatékonyan megvalósítható a teljes illesztés. Az adatszerkezet segítségével könnyen adhatunk ajánlást a megfelelő ε megválasztásához is, azáltal, hogy támogatja az adatbázisban tárolt idősorok egy adott idősortól vett távolságáról közelítő hisztogram készítését is.

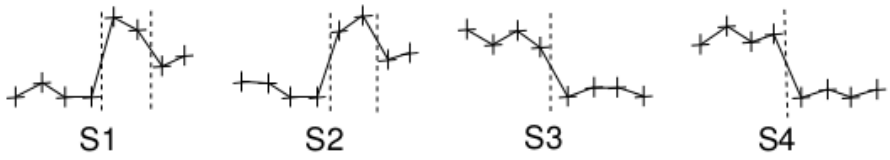
Egy további művelet, amely hatékonyan megvalósítható az általuk leírt reprezentációval, az adatbázisban tárolt idősorok egy adott idősortól vett távolságáról közelítő hisztog-

ram készítése. Utóbbi hasznos lehet például amiatt, hogy a hisztogram alapján könnyebb megválasztani az ε értéket a teljes illesztéshez.

Az idősorokon értelmezett teljes illesztés probléma hatékony kezelésére több különböző indexet találhatunk a szakirodalomban. Két közös vonást is felfedezhetünk bennük. Az egyik az, hogy az idősorok dimenzióját csökkenteni próbálják valamilyen globálisan (minden idősorra egyformán) értelmezett szegmentálással, a másik pedig hogy keresés során alsó korlátok számításával próbálják vágni a keresési fa ágait, csökkenteni annak méretét.

A dimenziócsökkentés megértését könnyíti, ha úgy tekintünk egy n értékből álló idősorra, mint egy pontra az n dimenziós euklideszi térben. Alkalmazható rá például a szinguláris felbontás, a diszkrét Fourier-transzformáció, de illeszthető rá polinom vagy spline is. Ezek (esetleg a kevésbé fontos tagok elhagyása után) egy alacsonyabb dimenziójú közelítését adják az eredeti idősornak. Így már alkalmazhatóak rájuk térbeli indexek, mint például az R-fa [7], melyek a magasabb dimenziójú adatokat nehezen kezelik. Fontos megemlíteni, hogy ezek mind globális módszerek, vagyis minden idősorhoz ugyanúgy számítanak ki egy egységes dimenziójú közelítést, általában annak adott sűrűséggel történő szegmentálása, majd a szegmensek közelítése révén.

A cikkben megjelenő egyik fő ötlet, hogy előnyös lenne dinamikusan, az adattól függően szegmentálni az idősorokat. Ezt szemlélteti a cikkből kiragadott 1. ábra. Az ábrán az $S1$ és $S2$ idősorokat érdemes 3 szegmensre bontani, ha konstanssal jól közelíthető szegmenseket szeretnénk kapni, ugyanakkor az $S3$ és $S4$ idősorok két szegmensre bontásával is alacsony szórású értékeket tartalmazó szegmenseket kapunk. Ilyen módon tömörebb reprezentációt hozhatunk létre anélkül, hogy feláldoznánk a közelítés minőségét.



1. ábra. Példa dinamikusan hatékonyabban szegmentálható idősorokra.

A másik közös vonás a szakirodalomban fellelhető megoldások többsége között, hogy alsó korlátozást használnak a teljes illesztéshez a hasonlósági keresés során. Ennek lényege, hogy létezik egy, a csökkentett dimenziójú reprezentációkon értelmezett $D_{LB}(\cdot,\cdot)$

függvény, melyre igaz, hogy ha $S_i (i \in \{1,2\})$ idősor reprezentációja $\tilde{S}_i (i \in \{1,2\})$, akkor $D_{LB}(\tilde{S}_1, \tilde{S}_2) \leq D(S_1, S_2)$. Egy ilyen függvény ismeretében ha az indexben tárolt $\tilde{S}$ tömör reprezentációra, valamint a lekérdezés Q idősorának hasonló módon tömörített reprezentációjára ($\tilde{Q}$) igaz, hogy $D_{LB}(\tilde{Q}, \tilde{S}) > \varepsilon$, akkor az S által reprezentált idősorokat már nem kell vizsgálnunk, ugyanis azok tényleges távolsága Q-tól biztos, hogy ε-nál nagyobb.

A cikk újításai közé tartozik egyrészt egy elterjedt alsó korlát számítási módszer kiegészítése, pontosabbá tétele, másrészt felső korlátozás használata. A felső korlátozás lényege, hogy létezik egy $D_{UB}(\cdot,\cdot)$ függvény, melyre igaz, hogy minden idősorra (az előző bekezdés jelöléseivel) $D_{UB}(\tilde{S}_1, \tilde{S}_2) \geq D(S_1, S_2)$. Ennek ismeretében ha Q idősorhoz hasonló idősorok keresésre során találunk egy $\tilde{S}$ értéket, amelyre $D_{UB}(\tilde{Q}, \tilde{S}) \leq \varepsilon$, akkor tudjuk, hogy az $\tilde{S}$ által reprezentált összes idősor Q-tól ε távolságon belül van, így része az eredménynek.

2. Irodalmi áttekintés

A teljes illesztés problémával kapcsolatos első eredmények közé talán a legjelentősebb Agrawal és tsai. műve [1], melyben távolságfüggvénynek az euklideszi távolságot, dimenziócsökkentésnek a diszkrét Fourier-transzformációt, a transzformáltak indexelésére pedig R-fát [7] használtak. Faloutsos és tsai. [6] a problémát kiterjesztették a részsorozat illesztésre.

A későbbi cikkek főleg különböző módszerekkel foglalkoztak az idősorok dimenziójának csökkentésére, úgy, hogy a csökkentett dimenziójú adatokhoz lehessen adni jó D_{LB} alulról korlátozó távolságfüggvényt. A használt módszerek közé tartozik például a diszkrét Fourier-transzformáció mellett a szinguláris felbontás [11], a diszkrét wavelet transzformáció [5], a szakaszonként lineáris approximáció [10], a szakaszonkénti aggregáció [14], az adaptív szakaszonként konstans approximáció [4] és a Csebisev-polinomokkal történő közelítés [2].

Az indexelésre a legnépszerűbb eszköz az R-fa, ezt módosítás nélkül [1], vagy az adott csökkentett dimenziójú idősor-reprezentációhoz igazítva [4] is alkalmazzák. Egy meglehetősen új megközelítése az iSAX [12, 3], mely egyben egy reprezentáció (SAX) és egy hozzá tartozó indexelési módszer.

3. Módszerek

3.1. Bővített adaptív szakaszonként konstans approximáció

Először bemutatjuk a cikk által leírt és használt dimenziócsökkentési módszert, a bővített adaptív szakaszonként konstans approximációt (Extended Adaptive Piecewise Constant Approximation, EAPCA), mely nevéhez híven az ismert adaptív szakaszonként konstans approximáció (Adaptive Piecewise Constant Approximation, APCA) egy kiegészítése. Ebben az alfejezetben leírjuk a két reprezentációt, valamint a rajtuk értelmezett, a bevezetőben említett f_{LB} és f_{UB} alsó és felső korlátozó függvényeket, melyek segítségével az idősorok távolságát tudjuk alulról és felülről becsülni.

A továbbiakban legyen adott n pozitív egész szám, valamint $X = (x_1, x_2, \ldots, x_n)$ és $Y = (y_1, y_2, \ldots, y_n)$ n hosszú, valós számokból álló idősorok. X és Y euklideszi távolsága $D(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. A cikk korábbi cikkek bevett gyakorlatára és elemzésire [9] támaszkodva ezt használja távolságmértéknek, így ezentúl két idősor távolsága ($D(X,Y)$) alatt euklideszi távolságukat értjük.

Az APCA reprezentációban egy X idősort $(X_1, X_2, \ldots, X_m)$ szegmenseire bontjuk, ahol $m \leq n$ és $\forall j \in \{1, \ldots, m\} : X_j = (x_{r_{j-1}+1}, \ldots, x_{r_j})$ valamely $0 = r_0 < r_1 < \cdots < r_m = n$ indexsorozatra. Egy X_j szegmens közelítő reprezentációja a (μ_j, r_j) páros lesz, ahol $\mu_j = \frac{\sum_{k=r_{j-1}+1}^{r_j} s_k}{r_j - r_{j-1}}$ a szegmens értékeinek átlaga. X közelítő reprezentációja tehát $\tilde{X} = ((\mu_1, r_1), \ldots, (\mu_m, r_m))$.

Azt mondjuk, hogy $\tilde{X}$ és $\tilde{Y}$ APCA reprezentációk illeszkednek, ha az előző bekezdés jelöléseivel az m és $r_0, \ldots, r_m$ értékeik megegyeznek, vagyis $\tilde{X} = ((\mu_1^X, r_1), \ldots, (\mu_m^X, r_m))$. és $\tilde{Y} = ((\mu_1^Y, r_1), \ldots, (\mu_m^Y, r_m))$. Illeszkedő reprezentációk alapján a 3.1. Lemma segítségével tudunk alsó korlátot számítani két idősorra.

3.1. Lemma. *Adott két egyforma hosszú idősor, X és Y, valamint illeszkedő APCA reprezentációik, $\tilde{X} = ((\mu_1^X, r_1), \ldots, (\mu_m^X, r_m))$ és $\tilde{Y} = ((\mu_1^Y, r_1), \ldots, (\mu_m^Y, r_m))$. Ekkor:*

$$D(X,Y) \geq \sqrt{\sum_{i=1}^{m}(r_i - r_{i-1})(\mu_i^X - \mu_i^Y)^2}$$

Az EAPCA reprezentáció a szegmensek reprezentációját az értékek szórásával egészíti ki, és ennek segítségével egy pontosabb alsó korlátot, valamint egy felső korlátot

is definiál. X idősor EAPCA reprezentációja az APCA reprezentációhoz hasonlóan $\tilde{X} = ((\mu_1, \sigma_1, r_1), \ldots, (\mu_m, \sigma_m, r_m))$ lesz, ahol a megjelenő új $\sigma_i = \sqrt{\frac{\sum_{j=r_{i-1}+1}^{r_i} s_j^2}{r_i - r_{i-1}} - (\frac{\sum_{j=r_{i-1}+1}^{r_i} s_j}{r_i - r_{i-1}})^2}$ értékek a reprezentált szegmens értékeinek szórása. Illeszkedő reprezentációkat is hasonlóan definiálhatunk, alsó és felső korlátot pedig a 3.2. Tétel alapján számíthatunk.

3.2. Tétel. *Adott két egyforma hosszú idősor, X és Y, valamint illeszkedő EAPCA reprezentációik, $\tilde{X} = ((\mu_1^X, \sigma_1^X, r_1), \ldots, (\mu_m^X, \sigma_m^X, r_m))$ és $\tilde{Y} = ((\mu_1^Y, \sigma_1^Y, r_1), \ldots, (\mu_m^Y, \sigma_m^Y, r_m))$. Ekkor:*

$$D(X,Y) \geq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})[(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^X - \sigma_i^Y)^2]}$$

és

$$D(X,Y) \leq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})[(\mu_i^X - \mu_i^Y)^2 + (\sigma_i^X + \sigma_i^Y)^2]}$$

Az alsó korlát nyilvánvalóan pontosabb, hiszen a gyökjel alatt levő összeg minden tagja egy $(r_i - r_{i-1})(\sigma_i^X - \sigma_i^Y)^2$ nemnegatív értékkel van megnövelve. A 3.2. Tétel bizonyítását a cikk [13] részletesen taglalja.

3.2. Korlátok idősorok halmazaitól vett távolságra

Az alsó korlátot EAPCA reprezentáció mellett kiterjeszthetjük idősorok halmazától vett távolságra is. Ez alatt azt értjük, hogy egy idősor a halmaz bármely elemétől vett távolságát, vagyis X idősor és $Y_1, \ldots, Y_l$ egyforma hosszú idősorok esetén a $\min_{1 \leq j \leq l} D(X, Y_j)$ távolságot szeretnénk alulról becsülni.

Legyen tehát X az idősor, amelynek a távolságát az $Y_1, \ldots, Y_l$ idősorokból álló halmaztól alulról szeretnénk becsülni. Legyen $X, Y_1, \ldots, Y_l$ mind egyforma hosszú, továbbá legyenek $\tilde{X} = ((\mu_1^X, \sigma_1^X, r_1), \ldots, (\mu_m^X, \sigma_m^X, r_m))$, $\tilde{Y_1} = ((\mu_1^{Y_1}, \sigma_1^{Y_1}, r_1), \ldots, (\mu_m^{Y_1}, \sigma_m^{Y_1}, r_m))$, $\ldots$, $\tilde{Y_l} = ((\mu_1^{Y_l}, \sigma_1^{Y_l}, r_1), \ldots, (\mu_m^{Y_l}, \sigma_m^{Y_l}, r_m))$ az $X, Y_1, \ldots, Y_l$ idősorok illeszkedő EAPCA reprezentációi. Jelölje az $\tilde{Y_1}, \ldots, \tilde{Y_l}$ halmazban i. szegmens minimális és maximális átlagát $\mu_i^{min} = \min_{1 \leq j \leq l} \mu_i^{Y_j}$ és $\mu_i^{max} = \max_{1 \leq j \leq l} \mu_i^{Y_j}$, szórását pedig $\sigma_i^{min} = \min_{1 \leq j \leq l} \sigma_i^{Y_j}$ és $\sigma_i^{max} = \max_{1 \leq j \leq l} \sigma_i^{Y_j}$.

3.3. Tétel. *Adott $X, Y_1, \ldots, Y_l$ egyforma hosszú idősorok és illeszkedő $\tilde{X}, \tilde{Y_1}, \ldots, \tilde{Y_l}$ EAPCA reprezentációik. Ekkor*

$$\min_{1 \leq j \leq l} D(X, Y_j) \geq \sqrt{\sum_{i=1}^{m} (r_i - r_{i-1})(LB_i^{\mu} + LB_i^{\sigma})^2}$$

és

$$\max_{1\leq j\leq l} D(X,Y_j) \leq \sqrt{\sum_{i=1}^{m}(r_i - r_{i-1})(UB_i^{\mu} + UB_i^{\sigma})^2}$$

ahol

$$LB_i^{\mu} = \begin{cases} (\mu_i^{min} - \mu_i^X)^2 & \text{if } \mu_i^X \leq \mu_i^{min}; \\ 0 & \text{if } \mu_i^{min} < \mu_i^X \leq \mu_i^{max}. \\ (\mu_i^{max} - \mu_i^X)^2 & \text{if } \mu_i^{max} < \mu_i^X; \end{cases}$$

$$LB_i^{\sigma} = \begin{cases} (\sigma_i^{min} - \sigma_i^X)^2 & \text{if } \sigma_i^X \leq \sigma_i^{min}; \\ 0 & \text{if } \sigma_i^{min} < \sigma_i^X \leq \sigma_i^{max}. \\ (\sigma_i^{max} - \sigma_i^X)^2 & \text{if } \sigma_i^{max} < \sigma_i^X; \end{cases}$$

és

$$UB_i^{\mu} = \begin{cases} (\mu_i^{max} - \mu_i^X)^2 & \text{if } \mu_i^X \leq \frac{\mu_i^{min}+\mu_i^{max}}{2}; \\ (\mu_i^{min} - \mu_i^X)^2 & \text{if } \frac{\mu_i^{min}+\mu_i^{max}}{2} < \mu_i^X; \end{cases}$$

$$UB_i^{\sigma} = (\sigma_i^{max} + \sigma_i^X)^2$$

A 3.3. Tétel bizonyítását a cikk [13] részletesen taglalja. A tétel következménye, hogy ha illeszkedő EAPCA reprezentációval adott idősorok halmazától vett távolságot szeretnénk alulról becsülni, ahhoz elég a szegmensek átlagainak és szórásainak minimumát és maximumát számon tartanunk.

3.3. A DSTree index

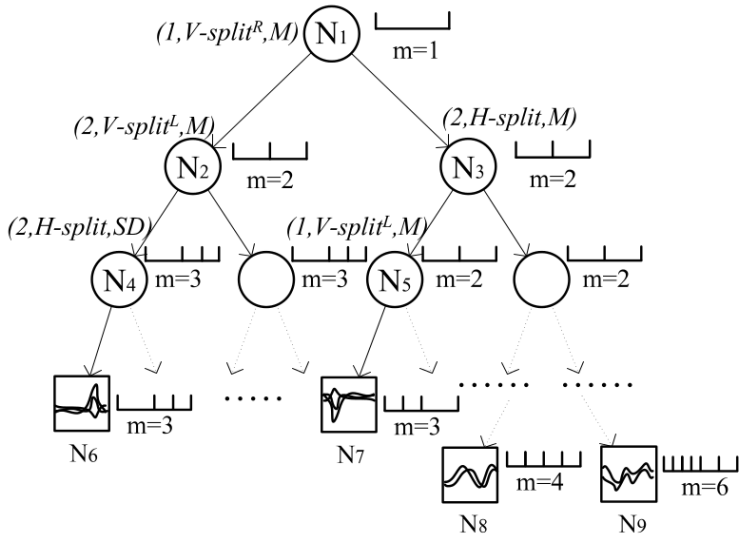
A cikk fő eredménye egy, az EAPCA reprezentációra épülő új index, a DSTree (*dynamic splitting tree*) bevezetése. Ennek leírásához először definiáljuk a reprezentációhoz tartozó szegmentálások között a *finomítás* relációt. Mint láttuk, az EAPCA reprezentáció szegmensekre bontja az idősort. A szegmenseket meghatározza jobb végpontjuk, vagyis a reprezentáció leírásakor használt jelölésekkel $(r_1,\ldots,r_m)$, ahol $0 < r_1 < r_2 < \cdots < r_m = n$, egyértelműen meghatározza a szegmentálást. Adott $SG_1 = (r_1,\ldots,r_m)$ és $SG_2 = (r'_1,\ldots,r'_{m'})$ szegmentálásokra azt mondjuk, hogy SG_2 egylépéses finomítása SG_1-nek, ha $m' = m+1$, és létezik olyan $1 \leq i_0 < m$, hogy $1 \leq i \leq i_0$ esetén $r_i = r'_i$, $i_0 < i$ esetén pedig $r_i = r'_{i+1}$. Jelölése: $SG_1 \prec^1 SG_2$. Azt mondjuk, hogy SG_2 finomítása SG_1-nek, ha létezik $SG'_1,\ldots,SG'_l$ szegmentálások egy olyan sorozata ($l \geq 2$), melyre

$SG_1 = SG'_1 \prec SG'_2 \prec \cdots \prec SG'_l = SG_2$.

A DSTree a 2. ábrán látható módon bináris fa struktúrába szerveződik, egy csúcs a részfájába tartozó idősorok egy indexe. Megkülönböztetünk belső csúcsokat és leveleket. Minden csúcsban tároljuk a következő információkat:

- A csúcs által meghatározott részfában található idősorok C számát.
- Az $SG = (r_1, \ldots, r_m)$ szegmentálását a csúcs által indexelt idősoroknak.
- A felső és alsó korlát számításához felhasználható aggregált $Z = (z_1, \ldots, z_m)$ információt, ahol $z_i = (\mu_i^{min}, \mu_i^{max}, \sigma_i^{min}, \sigma_i^{max})$.

Ezeken kívül a levelek tárolnak egy mutatót egy legfeljebb ψ idősort tároló fájlra, ahol ψ a fa előre megadott *levélkapacitása*, a belső csúcsok pedig tárolják a hasítási stratégiát, amely a 3.3.2. fejezetben lesz kifejtve.



2. ábra. DSTree index

A DSTree minden csúcsára igaz, hogy a részfájában található csúcsokhoz tartozó szegmentálások vagy a saját szegmentálásával azonosak, vagy finomabbak annál. Ez megengedi azt, hogy különböző csúcsok szegmentálása különböző legyen. A szegmensek száma is különbözhet, de ezek egyezése esetén is lehet különbség két szegmentálás között. A 2. ábrán az N_2 és N_4 csúcsok az előbbi, az N_4 és N_7 csúcsok az utóbbi esetre mutatnak példát.

3.3.1. Felépítés

A DSTree felépítése az inicializációval kezdődik, majd az idősorokat egyesével beszúrjuk a fába. A fát úgy inicializáljuk, hogy egyetlen csúcsból, a gyökérből (N_R) álljon, az ahhoz tartozó szegmentálás pedig $SG = (n)$ legyen, vagyis egyetlen szegmensre bontsa az idősorokat az index.

A beszúrás művelet algoritmusa a 3. ábrán látható (a cikkből [13] másolva). Az idősorokat úgy próbáljuk elhelyezni, hogy a hasonló idősorok egy levélbe kerüljenek. Ennek elérése érdekében beszúráskor a gyökértől indulva heurisztikusan a csúcshoz tartozó hasítási stratégia alapján döntjük el, hogy melyik részfájába szúrjuk be az új idősort. Ha levélhez értünk, beszúrjuk az új idősort. Ha ezzel meghaladná az idősorok száma a levél ψ kapacitását, akkor a levelet hasítani kell, aminek következtében a levélből belső csúcs lesz két új levéllel.

Algorithm 1 $N.Insert(X)$: N is a node, X is a time series

```
1:  update Z in node N according to X;
2:  if N is a leaf node then
3:      if C < ψ then                          ▷ N has space to hold X
4:          Append X to data file pointed by N, C = C + 1;
5:      else                                   ▷ C == ψ, no space in N to hold X
6:          Append X to data file pointed by N, C = C + 1;
7:          SP = BestSplit();
8:          Create two children nodes for N;
9:          for each time series Y in N do
10:             N' = N.routeToChild(Y, SP); N'.insert(Y);
11:         end for
12:     end if
13: else
14:     N' = N.routeToChild(X, SP); N'.insert(X);
15: end if
```

3. ábra. Beszúrás DSTree-be

Kritikus lépés az algoritmusban az alkalmazott hasítási stratégia kiválasztása ($BestSplit()$). Ezt, valamint a két utódcsúcs közül a hasítási stratégia alapján választó $routeToChild()$ függvényt a következő fejezet tárgyalja.

3.3.2. Hasítási stratégiák

Egy belső csúcsnál többféle módon lehet eldönteni, hogy melyik idősor melyik részfába kerüljön. Egy ilyen módot hasítási stratégiának hívunk. Amikor egy levélből csúcs lesz, akkor rendelünk hozzá az ott található idősorok alapján egy hasítási stratégiát, majd a későbbi idősorok beszúrásánál a hozzárendelt hasítási stratégiát követjük.

A cikk megkülönböztet vízszintes hasítást (horizontal splitting, H-split) és függőleges hasítást (vertical split, V-split). Az előbbi esetén a szegmentálás nem változik, az utódok szegmentálása ugyanaz marad, míg az utóbbi esetén a szegmentálás finomodik, a szülő csúcs szegmentálásának ugyanazt az egylépéses finomítását fogja tartalmazni mindkét utód.

A cikk kétféle vízszintes hasítási stratégiát használ. Az egyik esetén kiválasztjuk az egyik szegmenst, majd az idősorokat a megfelelő szegmensen felvett átlaguk alapján particionáljuk. Ha az átlag kisebb, mint a minimális és a maximális átlag (a csúcsban tárolt μ_i^{min} és μ_i^{max} értékek) számtani közepe, akkor az egyik részfába kerül az idősor, ha nagyobb vagy egyenlő, akkor a másikba. A másik stratégiai hasonló, csak átlag helyett szórást használunk. A függőleges hasítás a következő módon történik. Az egyik szegmenst kiválasztjuk és megfelezzük, majd a keletkező két szegmens közül valamelyikre vízszintes hasítást alkalmazunk.

Kérdés, hogy hogyan válasszunk stratégiát, illetve stratégián belül szegmenst, amire alkalmazzuk. Célunk, hogy minél hasonlóbb idősorok kerüljenek egy részfába. Ha ezt szem előtt tartva szeretnénk minden lehetséges stratégiához kiszámítani az egy részfába kerülő idősorok páronkénti hasonlóságát, az meglehetősen számításigényes lenne. Éppen ezért a cikk, abból kiindulva, hogy a hasonlósági keresés során kapott alsó és felső korlát különbségét szeretnénk minimalizálni, a következő mértéket vezeti le arra, hogy mennyire előnyös számunkra egy csúcs:

$$Qos = \sum_{i=1}^{m} (r_i - r_{i-1})((\mu_i^{max} - \mu_i^{min})^2 + (\sigma_i^{max})^2)$$

A stratégia választása tehát úgy történik, hogy minden lehetséges stratégiára (azokat minden lehetséges szegmensre alkalmazva) kiszámítjuk a keletkező R jobb és L bal utód Qos értékét, valamint a hasított N csúcsét is, és azt a stratégiát választjuk, amelyre a $Qos(N) - \frac{Qos(R)+Qos(L)}{2}$ érték maximális. Ez a 3. ábrán a $BestSplit()$ függvény. A

routeToChild() a már adott hasítási stratégia alapján mondja meg, hogy egy idősor melyik részfába kerül.

3.3.3. Műveletek

A DSTree kétféle lekérdező műveletet támogat, hasonlósági keresést, amely a leghasonlóbb idősort adja vissza, valamint a távolságok eloszlásának becslését. Előbbiből az egzakt algoritmus mellett létezik egy heurisztikus változat is, mely gyors, és bár jó eséllyel hasonlót, de nem biztos, hogy a leghasonlóbb idősort adja vissza. Egy adott Q idősortól a távolságok eloszlásának becslése egy közelítő hisztogramot kiszámítását takarja.

Algorithm 2 $exactSearch(Q)$

1: **Input**: A query time series Q
2: **Output**: The nearest time series TS with distance D_{bsf}
3: $N_{bsf} = HeuristicSearch(Q)$;
4: $(TS, D_{bsf}) = calcMinDist(N_{bsf}, Q)$;
5: Initialize distance priority queue pq;
6: $pq.Add(N_R, D_{LB}(N_R, Q))$;
7: **while** $!pq.isEmpty()$ **do**
8: $(N_{cur}, LB_{cur}) = pq.PopMin()$;
9: **if** $LB_{cur} > D_{bsf}$ **then**
10: break;
11: **end if**
12: **if** N_{cur} is a leaf node **then**
13: $(X, Dist) = calcMinDist(N_{cur}, Q)$;
14: **if** $Dist < D_{bsf}$ **then**
15: $D_{bsf} = Dist; TS = X$;
16: **end if**
17: **else**
18: **for all** children nodes N' of N_{cur} **do**
19: **if** $D_{LB}(N', Q) < D_{bsf}$ **then**
20: $pq.add(N', D_{LB}(N', Q))$;
21: **end if**
22: **end for**
23: **end if**
24: **end while**
25: return TS,D_{bsf};

4. ábra. Hasonlósági keresés

A heurisztikus hasonlósági keresés a beszúrás mintájára megnézi, hogy melyik levélbe kerülne a lekérdezés Q idősora, majd az ebben a levélben tárolt minden idősorra kiszámítja annak Q-tól vett távolságát, és visszaadja a legközelebbit. A 4. ábrán látható egzakt algoritmus ezt felhasználja arra, hogy gyorsan találjon egy jó közelítő megoldást, amely

alapján jó eséllyel sok ágat tud vágni a keresési fán. A csúcsokban az ismertetett módon tudunk alsó korlátot számítani az általa indexelt csúcsok Q-tól vett távolságára. A bejárás egy prioritási sor alapján történik, mindig a legígéretesebbnek tűnő csúcsból lépünk tovább. Ha ez egy levél, akkor kiszámítjuk a benne tárolt idősoroktól vett pontos távolságot. Ha már a legígéretesebbnek tűnő csúcsról is látszik az alsó korlát alapján, hogy az általa indexelt idősorok távolabb vannak Q-tól, mint az eddig talált legjobb megoldás, akkor befejezhetjük a keresést.

Az 5. ábrán látható a közelítő hisztogram készítő algoritmus. A csúcsokban a Q-tól vett távolságra számított alsó és felső korlát, valamint a csúcs által indexelt idősorok száma alapján, azok összesítésével tudunk becslést adni adott távolság-intervallumon belül levő idősorok számára. A cikkben ennek részletezése mellett egy alternatív megközelítés is felmerül, amely esetén a gyökérből a levelekbe vezető utakat adott α arányban osztó (vagy ehhez legközelebb eső) csúcsokat vesszük figyelembe a hisztogram készítésénél.

Algorithm 3 $Histogram(Q)$

Input: A query time series Q
Output: A distance histogram $Hist$
Initialize a distance range count list L;
Initialize a node stack $Stack$;
$Stack.push(Root, -\infty, +\infty)$;
while $!Stack.isEmpty()$ **do**
    $(N, LB_p, UB_p) = Stack.Pop()$;
    $LB = D_{LB}(N, Q)$;
    $UB = D_{UB}(N, Q)$;
    $LB = max(LB, LB_p)$;
    $UB = min(UB, UB_p)$;
    if N is a leaf node **then**
        $Count = N.C$; $L.add(LB, UB, Count)$;
    else
        for all child node of N, N' **do**
            $Stack.Push(N', LB, UB)$;
        end for
    end if
end while
$Hist = BuildHistogram(L)$;
return $Hist$;

5. ábra. Közelítő hisztogram kiszámítása

4. Eredmények

A leírt adatszerkezetet alapos tesztelésnek vetették alá. A tesztelés során összehasonlítási alapul szolgált kettő különböző indexelési módszer. Az egyik a szakaszonkénti aggregáció [14] (Piecewise Aggregate Approximation, PAA) segítségével csökkenti az idősorok dimenzióját, majd R-fával indexeli azokat. Ezt a szerzők maguk implementálták. A másik az iSAX2.0 [3] index, melynek az eredeti implementációját használták. Az algoritmusok paraméterezését is részletesen taglalja a cikk.

A tesztadatok két fő csoportra oszthatóak: szintetikus és valódi. A szintetikus adatok a következő módszerekkel lettek előállítva:

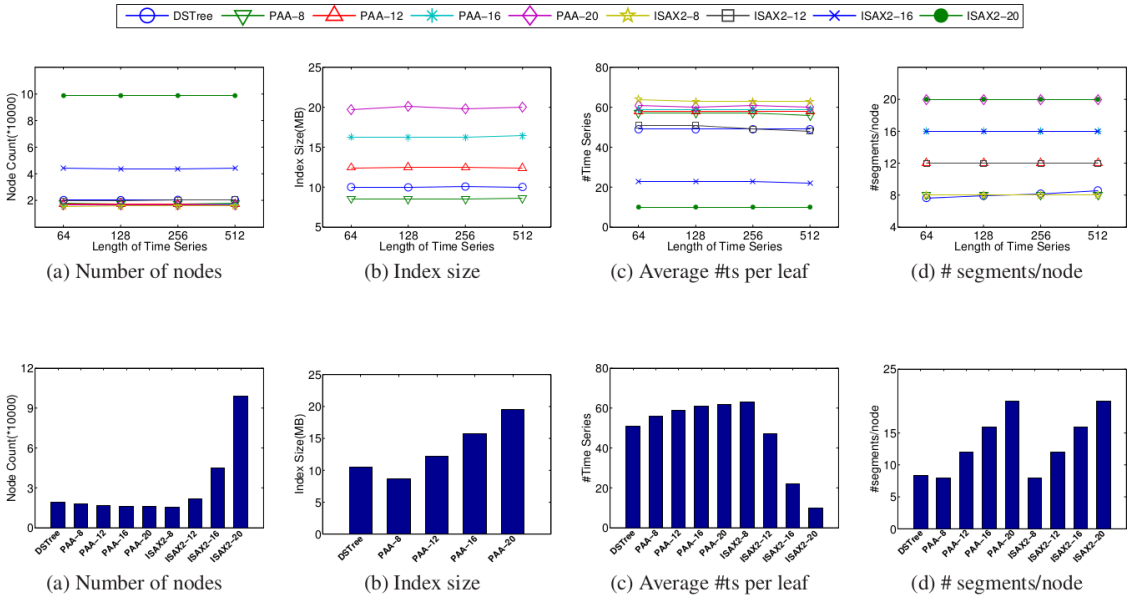
- Véletlen bolyongás, a $[-5,5]$ intervallumból egyenletesen véletlenül választott kezdőponttal, valamint a $[0,2]$ intervallumból egyenletesen véletlenül választott lépéshosszal.
- Normális eloszlással generálódnak az idősor pontjai. Az eloszlás középpontja a $[-5,5]$ intervallumból, szórása pedig a $[0,2]$ intervallumból kerül egyenletesen véletlenül kiválasztásra.
- Az előző módszerrel generálunk legalább 3, legfeljebb 10 idősort, majd ezeket konkatenáljuk. A konkatenált szegmensek száma egyenletesen véletlenül kerül kiválasztásra.
- Több szinuszfüggvény összekeveréséből mintavételezéssel. A függvények periódusa a $[2,10]$ intervallumból, amplitúdója a $[2,10]$ intervallumból, átlaga pedig a $[-5,5]$ intervallumból kerül egyenletesen véletlenül kiválasztásra.

Az előállítási mód szintén egyenletese véletlenül kerül kiválasztásra. Ezzel a módszerrel négy adathalmazt generáltak, egy 64, egy 128, egy 256 és egy 512 érték hosszú idősorokból állót. A skálázhatóság tesztelése során használtak további, akár $200 \cdot 10^6$ idősorból álló szintetikus adathalmazokat is.

A valódi adathalmazok hidak állapotát mérő szenzoroktól származnak. 10^6 idősor lett összegyűjtve több mint 20 féle szenzortól, mint például hőmérők vagy gyorsulásmérők. Minden idősor hossza 256 érték, ez összesen nagyjából $3GB$ adat.

A futtatások mind egy Intel Core i5 2.5GHz processzorral és 4 GB memóriával rendelkező asztali számítógépen történtek, továbbá minden feltüntetett érték 50 futtatás átlaga.

Az első összehasonlítási szempont az index mérete. Konkrétan a csúcsok száma, az index fizikai mérete, az egy levélben tárolt idősorok átlagos száma, valamint egy csúcs szegmentálásának átlagos szegmensszáma. Ezek az értékek láthatóak összehasonlítva a 6. ábrán, felül a szintetikus, alul a valódi adathalmazokon.

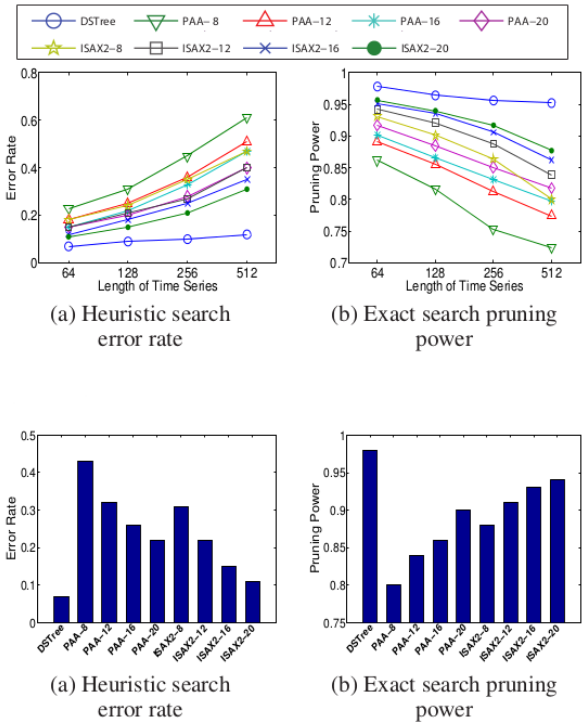


6. ábra. Indexméret összehasonlítása a szintetikus (felül) és valódi (alul) adathalmazokon

A mérési eredmények nem meglepőek, de igazolják a dinamikus hasítási stratégiák hatékonyságát: az átlagos szegmensszám csak mérsékelten nő az idősorok hosszával, a csúcsok száma pedig alacsony marad. Ezek mellett az index magasságáról is készült statisztika. Ennek tanulsága például, hogy bár az iSAX2.0 indexben az átlagos gyökértől levélig vezető úthossz alacsonyabb, mint a DSTree-ben, de maximális úthosszban rosszabbul teljesít.

A következő összehasonlítási szempont a keresés hatékonysága, az ide tartozó méréseket a 7. ábra mutatja. A heurisztikus keresés hatékonyságát az $E = \frac{|D(Q,\overline{X})-D(Q,X)|}{D(Q,X)}$ hibarátával (*error rate*) mérjük, ahol Q a lekérdezés, X az optimális megoldás, $\overline{X}$ pedig a heurisztikus keresés által visszaadott idősor. Az egzakt keresés hatékonyságát a vágási rátával (*pruning power*) mérjük, amely azon idősorok száma, amelyeket a lemezről felolvasásuk és pontos távolságuk kiszámítása nélkül elvetettünk, osztva az összes idősor

számával. 100 lekérdezést végeztek, melyeknek felét az adathalmazból származó, felét a szintetikus adatokhoz hasonlóan generált idősorok tették ki.



(a) Heuristic search error rate

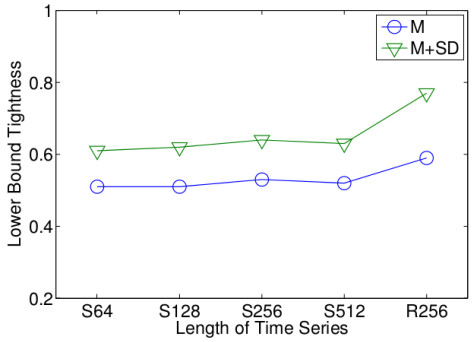
(b) Exact search pruning power

(a) Heuristic search error rate

(b) Exact search pruning power

7. ábra. Keresés hatékonysága szintetikus (felül) és valódi (alul) adathalmazokon

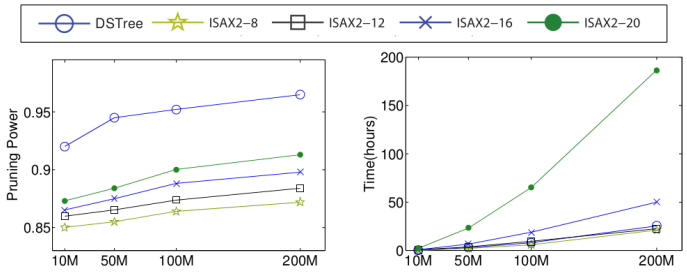
A mérések alapján a DSTree jelentősen jobban teljesít a többi indexnél, heurisztikus és egzakt keresés terén is. Ennek több oka is lehet, például a pontosabb alsó korlát, amelyet a 8. ábra szemléltet, vagy a dinamikus hasítási stratégiák.



8. ábra. APCA (M) és EAPCA (M+SD) alsó korlát értékek összehasonlítása. Az értékek az egzakt keresés teszteléséből származnak, és normálva vannak a közelített távolsággal.

Tesztelve lett a közelítő hisztogram, valamint az index skálázhatósága is. A skálázhatóságot a vágási rátán (*pruning power*) és az index felépítéséhez szükséges időn keresztül

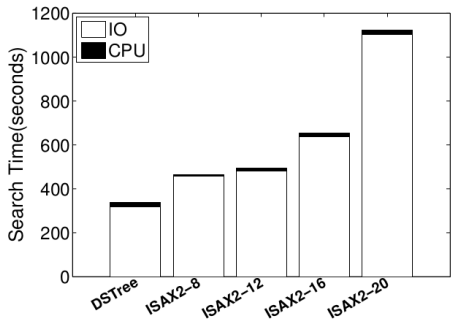
mérték, ezeket hasonlítják össze a 9. ábrán. Az összehasonlítás alapjául az iSAX2.0 index szolgál. Az adathalmazok az iSAX2.0-hoz tartozó szintetikus adatot generáló programmal készültek, 10, 50, 100 illetve 200 millió idősort tartalmaznak. Az idősorok hossza mindegyikben 256, a levelek kapacitása pedig mindenhol $\psi = 5000$.



(a) Pruning power (b) Building time

9. ábra. Skálázhatóság összehasonlítása.

A keresési időt szintén az iSAX2.0 indexszel hasonlítják össze, ezt mutatja a 10. ábra. A tesztadat $200 \cdot 10^6$ db, 256 hosszú idősorból állt, a 100 lekérdezés fele az adatok közt szereplő, fele véletlenszerűen generált idősor. A teljes index minden esetben elfért a memóriában.



10. ábra. Keresési idő összehasonlítása.

A mért futásidő alapján a DSTree egyértelműen hatékonyabb. Hogy könnyebb legyen ennek az okát vizsgálni, a futásidőt processzoridőre (index bejárása, idősorok összehasonlítása) és IO-időre (idősorok beolvasása a merevlemezről) bontjuk. Így látszik, hogy a processzoridő kis hányadát teszi ki a keresésnek minden index esetén, a különbséget az IO műveletekkel töltött idő különbsége okozza. Ennek oka vélhetően a DSTree jobb vágási rátája, valamint az, hogy kevesebb levelet tartalmaz.

5. Diszkusszió

A cikk egyik fő következtetése, hogy idősorok indexelésénél a globális szegmentálás feleslegesen megnövelheti az index méretét, valamint annak minőségén is ronthat; adaptív, dinamikus szegmentálással jelentősen javítani lehet a keresési időn. Ennek alapján meg lehetne vizsgálni az idősorok indexelésének ismert módszereit, hogy lehetséges-e dinamikus szegmentálást alkalmazni a globális szegmentálás helyett. Előnyös lehet továbbá az eddig APCA reprezentációt használó módszerek kipróbálása az új EAPCA reprezentációt és a rajta definiált pontosabb alsó korlátot használva.

A DSTree index felépítése szintén további kutatásra ad lehetőséget. A jelenlegi módszerrel az index hatékonysága függhet az idősorok beszúrási sorrendjétől. Ezzel szemben fel lehetne használni, hogy az összes idősor rendelkezésre áll az index felépítésekor, és ez a többletinformáció esetleg optimálisabb fa felépítését tenné lehetővé. Egy lehetséges módszer például az idősorok hierarchikus klaszterezése, majd a fa ez alapján történő kialakítása. További érdekes kérdés, hogy milyen további hasítási stratégiákat, illetve a stratégiák közötti választásra használható mértéket definiálhatunk, és ezek hatékonysága hogyan viszonyul a cikkben leírtakéhoz.

A végzett mérések a leghasonlóbb idősor megkeresésére koncentráltak, míg a cikk bevezetője a legfeljebb ε távolságra levő idősorok lekérdezését veti fel, mint probléma. Ennek a lekérdezésnek a megvalósítása a DSTree adatszerkezeten triviális, így érdemes lenne ennek hatékonyságát is összevetni az eddig ismert megoldásokkal.

Tekintve, hogy az idősorok indexelése aktív kutatási terület, ajánlatos lenne a különböző módszerek összehasonlítására szabványos kereteket létrehozni. A cikk saját valós adatokon, és saját módszerrel generált szintetikus adatokon végzi az index tesztelését. Az összehasonlításra sok saját maguk által definiált értéket használ, mint például a hibaráta vagy a vágási ráta. Kérdéses, hogy az eredményeket nem befolyásolja-e jelentősen ezek megválasztása, vagy például olyan döntések, hogy a hasonlósági lekérdezések felében a tárolt idősorok közül kerül ki a bemenet. Ezt a bizonytalanságot erősen csökkentené szabvány tesztelésre használható adatok, módszerek és optimalizálandó statisztikák létrehozása.

Hivatkozások

[1] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In DavidB. Lomet, editor, *Foundations of Data Organization and Algorithms*, volume 730 of *Lecture Notes in Computer Science*, pages 69–84. Springer Berlin Heidelberg, 1993.

[2] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 599–610, New York, NY, USA, 2004. ACM.

[3] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0: indexing and mining one billion time series. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 58–67, Washington, DC, USA, 2010. IEEE Computer Society.

[4] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.

[5] Kin-Pong Chan and Ada Wai-chee Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, ICDE '99, pages 126–133, Washington, DC, USA, 1999. IEEE Computer Society.

[6] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[7] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[8] Eamonn Keogh. A decade of progress in indexing and mining large time series databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1268–1268. VLDB Endowment, 2006.

[9] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov.*, 7(4):349–371, October 2003.

[10] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback, 1998.

[11] K. V. Ravi Kanth, Divyakant Agrawal, and Ambuj Singh. Dimensionality reduction for similarity searching in dynamic databases. *SIGMOD Rec.*, 27(2):166–176, June 1998.

[12] Jin Shieh and Eamonn Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 623–631, New York, NY, USA, 2008. ACM.

[13] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. In *Proceedings of the 39th International Conference on Very Large Data Bases*, VLDB '13. VLDB Endowment, 2013.

[14] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 385–394, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.