Morvai Mihály (MOMUABI), Szücs Ádám (SZAPACI), Verő Anita (VEAPAAB)

**A legrövidebb útkeresés relációs megközelítése nagyméretű gráfokban**

*Tanulmány Jianhua Jun Gao, Ruoming Jin, Jiashuai Zhou, Jeffrey Xu Yu, Xiao Jiang, Tengjiao Wang: Relational Approach for Shortest Path Discovery over Large Graphs*

*cikke alapján.*

*Adatbázisrendszerek elméleti alapjai – ELTE-IK, 2012*

**Absztrakt**

A nagyméretű gráfok gyors növekedését figyelembe véve nem tehetjük fel többé, hogy a gráfok teljes egészében betölthetőek a memóriába, így a lemez alapú gráf műveletek használata elkerülhetetlen. A cikk a legrövidebb útkeresés problémáján keresztül mutat be gráf keresési műveleteket relációs adatbázisokon (RDB).

Mivel a legtöbb gráf kereső lekérdezés iteratív műveletekkel oldható meg, úgymint: határcsúcsok kiválasztása a már elértek halmazából, határcsúcsok kiterjesztése valamint a kiterjesztett és az elért csúcsok összevonása, a szerzők bemutatják a három alapművelet RDB-n megvalósító *FEM* framework-öt. Bevezetésre kerül két új, a kifejezéseket egyszerűsítő valamint a *FEM* hatékonyságát növelő funkció: *window function* és *merge statement* néven. Továbbá bemutatnak két optimalizációs eljárást a legrövidebb útkeresésre a *FEM* framework felhasználásával. Az első egy kétirányú Dijkstra algoritmus, mely csökkenti a keresési teret valamint támogatja a kötegelt feldolgozást. A második egy *SegTable* elnevezésű index, a lokális szegmensek tárolására, mellyel tovább növelhetjük a hatékonyságot. A kísérleti eredmények illusztrálják a relációs megközelítés és az optimalizációs stratégiák jól skálázhatóságát és hatékonyságát.

**Bevezetés**

A gráf adatok nagy ütemű növekedése jelentős technikai kihívásokat jelent. Gráfokat manapság széles körben használnak pl.: szociális hálók, ontológiák, szállítási útvonalak reprezentálására. Ezek a reprezentációk általában nagyméretű gráfokat eredményeznek, melyek nem tölthetők be teljes egészében a memóriába ezért különböző puffer mechanizmusokra van szükség. Ebben az esetben az I/O költség lesz a meghatározó. Fontos szempontok a rugalmas csomópont elérés, valamint a gráf kezelés stabilitása.

A cikk a gráf keresések alapvető feladatán belül is a legrövidebb útkeresés problémájára összpontosít két okból: (1) mivel ez a feladat alapvető követelményként merül fel a legtöbb gráfokat kezelő alkalmazásban, (2) a legrövidebb útkeresés reprezentatív gráf kereső lekérdezés, hasonló kiértékelési mintákkal rendelkezik, mint más lekérdezések.

A jelenlegi lemez alapú módszereket vizsgálva, a szerzők az általános gráf kereső lekérdezésekkel kapcsolatos korlátokkal szembesültek. Bár a külső memória indexelésének tervezése rendelkezésre áll, ám ezek csak síkba rajzolható gráfokra alkalmazhatók [1]. A MapReduce keretrendszer [2, 3] és az open source implementációja, a Hadoop [4] elosztott fájlrendszerrel rendelkező számítógép klasztereken képes a nagy gráfok kezelésére, azonban a séma- és indexkezelő mechanizmusok hiánya miatt a gráf elérés nem elég rugalmas. Ezen kívül a gráfon keletkező dinamikus változások kezelése is gondot okoz. Néhány egyéb gráf művelet, mint a minimumvágás [5] és klikk számítás [6], kiértékelhető a részben betöltött gráfokon, ha az eredmények minősége elméletileg biztosítható vagy közelítő megoldás adható [6, 5]. Ezen metódusok más gráf műveletekre való kiterjesztése azonban nehéz.

Az elmúlt több int 40 évben az RDB kulcsszerepet tölt be az információs rendszerekben. Jól látható, hogy mind az RDB mind a gráfos adatkezelés több átfedő funkcionalitással rendelkezik, pl.: adattárolás, adat pufferelés, elérési műveletek stb. Ezen felül az RDB alapvető tárolási lehetőséget és rugalmas elérést biztosít a gráf számára. Továbbá az RDB már több egyéb komplex adattípushoz is rugalmas kezelést biztosít, mint pl.: XML adatkezelés, szélességi keresés.

A gráf kereső eljárások lefordítása, valamint a hatékonyság növelése RDB környezetben a relációs műveletek és a gráf műveletek között lévő szemantikai eltérések miatt kihívást rejt magában. A gráf keresésben felmerülő adatműveletek komplexitása miatt sokszor van szükség logikai és aritmetikai számításokra, érték egyesítésre valamint kereséshez szükséges információk tárolására. Azonban a gráf és a futásidő az RDB-ben van eltárolva, melyen csak korlátozott műveletek végezhetők, mint a vetítés, kiválasztás, összekapcsolás, aggregálás stb.

A cikk három fő cél köré épül fel:

* Sok gráf kereső lekérdezés által használt iteratív műveletek ­­­­– határcsúcs kiválasztás a már elértekből, határcsúcs kiterjesztés, kiterjesztett és elért csúcsok összefésülése – megvalósítása *FEM* keretrendszerben, három kulcs művelettel: *F*, *E* és *M*-operátor. Kifejezés egyszerűsítése és hatékonyság növelés két új SQL sztenderddel megvalósított funkció bevezetésével: *window function* és *merge statement.* A legrövidebb útkeresés bemutatása a *FEM* keretrendszer használatával.
* Két optimalizációs eljárás bemutatása a legrövidebb útkeresésre *FEM* keretrendszerben. Az első egy kétirányú Dijkstra algoritmus. A kétirányú keresés kisebb kereséséi térrel találja meg a legrövidebb utat. Ha a Dijkstra algoritmusnak megengedjük, hogy az ugyanazon minimális távolsággal rendelkező csúcsokat egy időben terjessze ki, az jobban passzol az RDB kontextushoz. A második egy *SegTable* elnevezésű index bevezetése a lokális, rövidebb szegmensek megőrzésére, valamint a *SegTable* további hatékonyságnövelésre való kihasználása a keresési tér leszűkítésének kiegyensúlyozásával valamint kötegelt kiértékelés támogatásával.
* Széleskörű kísérletek végzése szintetikus és valós életbeli adatokon. Az eredmények megmutatják, hogy a relációs megközelítés jól skálázható módon kezel nagy gráfokat. Továbbá látható az is, hogy az új SQL sztenderd funkciók és az optimalizációs eljárások is szignifikánsan növelik a hatékonyságot.

**Kapcsolódó munkák**

A Dijkstra algoritmus keresési terének szűkítésére a legtöbbet használt módszer a kétirányú Dijkstra [7]. A legrövidebb út több formában is előre kiszámolható, úgymint: landmark index [8, 9], 2-HOP kapcsolat index [10], vagy TEDI index [11]. Ezek az indexek nagyban növelik a hatékonyságot azonban mind feltételezi, hogy a gráf egy az egyben betölthető a memóriába.

A MapReduce keretrendszer [2, 3] és az open source implementációja, a Hadoop [4] jól skálázható eredményt ért el nagymérető gráfok kezelésénél. Korlátokba ütközik azonban a gráfok dinamikus frissítésénél.

Az adatbányászati feladat [12] ás a statisztikai analízis [13] is RDB használatot igényel. XML lekérdezések is SQL-re fordíthatók [ 14, 15, 16].

Az SQL alapú gráf adatbányászat témakört már többen tanulmányozták [17, 18]. A cikk a meglévő megoldásokat kívánja továbbfejleszteni egy általános gráf kereső keretrendszer kifejlesztésével, a hatékonyság növelésével, a keresési tér szűkítésének és az RDB-barát megközelítés egyensúlyának megtalálásával.

**Fogalmak bevezetése**

**Gráf jelölések:**

* gráf, a csúcsok, az élek halmaza.
* Minden egyedi azonosítóval rendelkezik.
* Élek:
* nem negatív élsúly.
* : minimális élsúly
* -ból -be vezető út., ahol az élek: .

A gráfok tárolására két relációs tábla szolgál:

* *TNodes*: csúcsokat tartalmazza –ből. Egyedi azonosítójuk: *nid.*
* *TEdges*: élek tárolása az halmazból. *fid, tid, cost*.

**SQL feature-ök:**



* *window function*: Aggregált eredményt ad vissza. A hagyományos aggregáló függvényekkel szemben nem csak egy eredményt ad vissza. A rendezett halmazokon lévő aggregált eredmény mellett az aggregációból kimaradt attribútumok is elérhetők maradnak egy későbbi *select* utasításhoz. A táblán partíciókat (*window*) képez, és ezekre hívja meg az aggregáló függvényt.
* *merge statement*: Olyan műveletek, mint az *insert, delete, update.* Kiértékelése gyorsabb, mint több módosító művelet egymás után való végrehajtása.

**Relációs FEM keretrendszer legrövidebb útkereséshez**

A legtöbb gráfkereső algoritmus a nagy keresési tér miatt mohó stratégát alkalmaz.

Jelölések: a *k*. iterációban:

* *Ak*: elért csúcsok
* *Fk* : adott feltételnek eleget tévő határcsúcsok (alkalmazás függő, *Fk* ⊆*Ak*)
* *Ek*: kiterjesztett csúcsok a *k*. iterációban (ált. *Fk* szomszédjai)

További jelölések a gráfkereső algoritmusokban:

* *s*: startcsúcs
* *d2s*: start csúcstól való távolságot tároló rekord
* *p2s*: szülőket tároló rekord
* *f*: flag: Dijkstra: azt jelzi kiterjesztett-e már a csúcs, Prim: feszítőfa eleme-e a csúcs
* *T*: Prim algoritmusban lévő feszítőfa

**FEM operátorok és kifejezésük relációs algebrával:**

A fenti jelölésekkel egy *u* csúcs reprezentációja: (*nid, d2s, p2s, f*).

**1. Definíció:****F-operátor** a *k*. kiterjesztés soránaz elért *Ak* csúcsok közül az *Fk* határcsúcsokat adja vissza. .

A *mid* kiszámítása (pl. Dijkstra-nál minimális *d2s*) egy tetszőleges művelettel kiszámítható az F-operátor előtt.

**2. Definíció:****E-operátor** a *k*. kiterjesztés sorána kiterjesztett*Fk*határcsúcsok és a *TEdges* tábla szerinti *Ek* kiterjesztett csúcsokat adja vissza.

*(1)*

*(2)*

**3. Definíció:****M-operátor** *Ek* kiterjesztett csúcsok és az elért *Ak* csúcsok szerinti *Ak+1* csúcsokat adja vissza.

*(1)*

*(2)*

*(3)*

Az *M-*operátorazokat a csúcsokat veszi ki *Ak –*ból, amelyek távolsága nagyobb, mint az újonnan kiterjesztett csúcsoknak *Ek*-ban. Ezután kiveszi azokat a csúcsokat *Ek –*ból, amelyek távolsága nagyobb, mint a megfelelő csúcsoké *Ak* –ban. Végül a kettő uniója adja meg *Ak+1*-et.

A FEM működése az alábbi ábrán látható a k. iterációban:



Példa gráf reprezentációra és az *F*, *E* és *M* operátorok működésére a 2. iterációban:



**Operátorok SQL implementációja:**

A relációs algebrai kifejezések direkt átírása nem lenne hatékony, különösen az *E* és *M*-operátorok esetében.

Az *E*-operátor megvalósításánál használhatjuk a már említett *window function*-t, mely képes visszaadni a nem aggregált attribútumokat is az aggregált eredmény mellett. Ily módon az operátor implementációjakor megspórolhatunk egy a szülőcsúcs kinyerést szolgáló összekapcsolást. Helyette partícionálhatjuk a táblát a *nid*-ek előfordulásaira nézve, rendezhetjük *d2s*-re és vetíthetjük a minimális *d2s*-t a *row\_number* aggregáló függvény segítségével.

Az *M*-operátor esetében két egymást követő *insert* majd *update* művelet helyett használhatjuk a *merge statement-*et, mely ezek kombinációjával hatékonyabb implementációt eredményez.

**Legrövidebb útkeresés algoritmusa FEM használatával:**

Inicializáljuk a *TVisited* táblát;

Amíg a célcsúcs nem terjesztődik ki:

 Meghatározzuk a legkisebb távolságú csúcsot (*mid*);

 Kiterjesztjük a *mid* azonosítójú csúcsot az *F*, *E* és *M* operátorral SQL-el;

 Meghatározzuk a módosult sorokat az adatbázisban;

 Ha a *TVisited* nem módosult:

 KÉSZ;

 Ha a célcsúcs le lett zárva:

 KÉSZ;

 Egyébként lezárjuk a *mid* azonosítójú csúcsot.

Iteratívan visszakeressük a legrövidebb *p* utat SQL-el a *p2s* linkeken.

Visszaadjuk *p*-t;

**Optimalizációs eljárások a legrövidebb útkeresésre**

Két (sokszor konfliktusba kerülő) szempontot veszünk figyelembe: i) a keresési tér leszűkítése általános optimalizációs stratégiával ii) kötegelt feldolgozás támogatása a relációs adatbázis operátorok használata miatt.

Az SQL lekérdezések optimalizálása kötegelt (set-at-a-time) lekérdezéseket igényel, melyek során egy SQL lekérdezésben egyszerre több adatot nyerünk ki, jelentősen csökkentve ezzel az I/O költséget, szemben azzal mintha minden csúcsot egyesével kérdeznénk le (node-at-a-time). Ez azonban jelentősen megnövelheti a keresési teret.

A keresési tér szűkítésére a szerzők kétirányú Dijkstra algoritmus-t használnak, mely a legrövidebb útkeresés feladatnál gyakran használt módszer [7].

**Kétirányú Halmaz Dijkstra Algoritmus:**

Az algoritmus mind a kötegelt adatelérést mind a keresési tér leszűkítését megcélozza. Az alapötlet az, hogy a legrövidebb utat a startcsúcs mellett visszafelé, a célcsúcsból is elkezdi keresni. Ezen kívül egy határcsúcs kiterjesztése helyett az összes lezáratlan csúcsot kiválasztja, melyek ugyanazon minimális távolsággal rendelkeznek, így megvalósul az RDB-barát kötegelt lekérdezés.

Jelölés: *s*: startcsúcs, *t*: célcsúcs

A kétirányú Dijkstra a hagyományos algoritmushoz hasonlóan kifejezhető a *FEM* framework segítségével, a megfelelő rekordokat alkalmazva a célcsúcsra (*p2t*, *d2t*). A kiterjesztéshez az irányt választja, ahol kevesebb határcsúcs szerepel, így csökkentve a köztes csúcsok számát.

Ha *lf*és *lb*jelöli a minimális távolságot a legutolsó előre és hátrafelé való kiterjesztés után, pedig a minimális távolságot *s* és *t* között, akkor az algoritmus leállási feltétele a következő: . (*lf*és *lb* a *TVisited* tábla *d2s*, *d2t* oszlopaiból számolható)

**1.TÉTEL**: *Legyen lf és lb a minimális távolság a legutolsó előre és hátrafelé való kiterjesztés után, a minimális távolság s és t között. Vegyük az előrefelé való kiterjesztést. Egy határcsúcsot akkor kell kiterjeszteni az csúcsba, ha* .

**2.TÉTEL**: *Adott két csúcs s és t egy G gráfban. A kétirányú halmaz Dijkstra algoritmus s-ből t-be vezető legrövidebb út megtalálásához való iterációinak száma, legrosszabb esetben:* min((*s,t*)*/wmin,n*) *ahol* (*s,t*) *az s és t közötti legrövidebb távolság, wmin a minimális élsúly G-ben, n a csúcsok száma.*

**Szelektív útvonal kiterjesztés SegTable felhasználásával:**

Bár a kétirányú halmaz Dijkstra algoritmus kötegelt megoldással találja meg a legrövidebb utat, az összes művelet száma mégis elég magas, ahogy azt a 2. Tételben láthattuk.

A magas műveletigény leküzdésére a szerzők bevezetik a *SegTable*-t, ami az előre kiszámított legrövidebb úton elérhető gráf szegmenseket tárolja a távolságukkal egy adott küszöb alatt, hasonlóan, mint egy legrövidebb útvonal index. A SegTable ily módon azokat szegmenseket tartalmazza, amelyek legnagyobb eséllyel szerepelnek a legrövidebb úton.

**4. Definíció: *SegTable.*** *Legyen G=(V,E) gráf. küszöb. A SegTable két táblát tartalmaz: TOutSegs és TInSegs. TOutSegs a kimenő irányban lévő előreszámított szegmenseket tárolja. TOutSegs minden sora (fid, tid, pid, cost) következőkből áll elő:
(1) a legrövidebb távolság és között, és a csúcs, -ból -be vezető, legrövidebb úton lévő szülője.
(2) az él súlya.*

*TInSegs hasonlóan tárolja az elő feldolgozott szegmenseket a bejövő irányban.*

A SegTable gyakorlatilag meghatároz egy *G’* gráfot, mely tartalmazza az előfeldolgozott szegmenseket valamint az eredeti éleket is *G*-ben.

A ***Szelektív útvonal kiterjesztés*** során a *k.* előrefelé való kiterjesztésnél egy csúcs akkor választódik ki határcsúcsnak, ha az vagy az az összes kiterjesztendő csúcs között minimális. Azaz, olyan csúcsokat terjesztünk ki, amelyeknél nagyobb esély van arra, hogy elkerüljük a felesleges újbóli kiterjesztéseket.

Példa: index küszöb,



Ahogy (c) példa is szemlélteti a k. kiterjesztésnél a legrövidebb út hossza:
, míg az eredeti kötegelt Dijkstra algoritmusnál: . Mivel a legrövidebb út hossza gyorsabban növekszik, az algoritmus megállási feltételét hamarabb elérjük.

**Kétirányú szelektív útvonal kiterjesztés SegTable-el:**

Az alábbi két ábra szemlélteti az algoritmus működését és SQL implementációját. A változónevek az eddigiekben használt jelöléseket követik. Az ezeken kívüli változók: *lf*: legrövidebb út hossza, *nf*: határcsúcsok száma, *fwd*: előre felé történő kiterjesztések száma. Hasonlóan definiáltak az ellenkező irányhoz tartozó változok a megfelelő sorokban. A 6-16- sorban a megállási feltétel teljesüléséig megtörténik a megfelelő irány kiválasztása, majd az *F, E*  és *M* operátorok végrehajtása. Az SQL implementációban a *TEgdes* helyett megjelenik a *TOutSegs* használata.

****

****

**3.TÉTEL**: *Adott egy s startcsúcs és egy t célcsúcs és a SegTable küszöb indexel. Az algoritmus iterációinak száma kisebb, mint* min(*n,*)*.*

**Kísérleti eredmények**

A szerzők külön vizsgálták a különböző lekérdezések időigényét és a SegTable létrehozásával járó költségeket.

A lekérdezések tesztadatairól készült statisztika, valamint a kiértékelések az alábbi ábrán láthatók. Megfigyelték, hogy az *F, E* és *M* operátor közül az *E* operátor kiértékelésének költsége a legnagyobb, az idő 75%-a (6.(b), 6.(c)). A 6.(d)) ábrán összehasonlították a hagyományos SQL eszközöket (TSQL) a két új feature-t (*window function, merge statement*) tartalmazó SQL-el (NSQL). Az eredmény egyértelműen az NSQL-t hozza ki hatékonyabbnak.



A 7.(a) és (b) ábra a cikkben szereplő, különböző eszközöket igénybe vevő (kötegelt feldolgozás, keresési tér szűkítés, SegTable) algoritmusok időigényét hasonlítja össze. Az eredményekből az a következtetés vonható, le, hogy a felvonultatott konfliktusos célok megvalósításánál egyensúlyra érdemes törekedni.

Részletesen vizsgálták az útvonal kiterjesztésnél használt küszöb hatását is mind a lekérdezések, mind az indexméretekre vonatkozóan. Azt találták, hogy sem a túl kicsi sem pedig a túl nagy küszöbérték nem jó, mivel előbbi a keresési tér méretét növeli, utóbbi pedig a szegmensek számát a SegTable-ben és csökkenti a kiterjesztések számát. Az optimális megtalálása további kutatómunka tárgyát képezi.



A 8. (a), (b) és (c) ábra különböző adatbázisok, puffer méretek és indexek függvényében vizsgálja az időigényt. Az eredmények alátámasztják a kifejlesztett módszer jól használhatóságát különböző adatbázis rendszereken (itt *PotgreSQL*). A puffer méret növekedésével közel lineárisan csökken a költség, valamint megfigyelhető, hogy egy küszöb után beáll egy majdnem stabil értékre. Az indexek összehasonlítása alapján a *CluIndex* használata bizonyult a leghatékonyabbnak.

A 8. (d) ábrán a memóriában futó Dijkstra algoritmus futási idejével hasonlították össze a cikkben szereplő módszert. Az eredményből nem meglepő módon kiderül, hogy a relációs megközelítés nem éri el a memóriában futó hatékonyságát. A módszer lényege azonban jól látszik, miszerint az RDB-n alapuló módszer nagyon jól skálázható, stabil, valamint könnyen programozható a gráf kezelési feladatok körében. Épp ezért nagy gráfokra kitűnően alkalmazható.



A 9. ábra a SegTable index méretét és felépítésének idejét elemzi az küszöb függvényében. Nagy több előre kiszámolt szegmenset eredményez, így az index mérete és a konstrukciós idő is az méretével nő. Megfigyelhető emellett, hogy különböző gráfokon a küszöb hatása különböző mértékben érvényesül (9. (b): *GoogleWeb* vs. *DBLP*). 9.(f) ábra ugyancsak az NSQL és TSQL összehasonlítását szemlélteti, ezúttal a konstrukciós idő szempontjából. Az eredmény itt is az NSQL hatékonyabb voltát támasztja alá.

A 9.(g) a puffer méret függvényében mutatja be az index felépítés időigényét. Az eredmény a lekérdezés kiértékeléshez nagyon hasonló.

A 9.(h) ábra a gráf méretének változásában mutatja be a konstrukció költségét. Amellett, hogy nagyobb gráf nagyobb index építést igényel, megállapítható a cikkben szereplő konstrukciós eljárás jól skálázhatósága. A felépítés ideje és a gráf méretének növekedése közel lineáris kapcsolatban áll, annak köszönhetően, hogy a SegTable csak lokális legrövidebb szegmenseket tárol.



**További kutatás terv**

Ahogyaz a cikkben is szerepel, az eredmények vizsgálhatók lehetnének további gráfkereső eljárásokra nézve is. Ezen kívül a hatékonyság növelése érdekében érdemes lenne kihasználni az elosztott adatbázisok nyújtotta előnyöket is. Ezáltal növelni lehetne a skálázhatóságot a gráf méret növekedésével. További vizsgálandó terület lehet a dinamikus változások kezelése a gráfban. A SegTable inkrementális frissítése megoldandó feladat lehet. Az elosztott tárolás elősegítheti a hatékony indexfrissítést is.

További indexek bevezetésének használatával is lehetne gyorsítani a lekérdezéseket.

**Hivatkozások**

[1] D.Hutchinson, A.Maheshwari, and N.Zeh. An external memory data structure for shortest path queries. Discrete Applied Mathematics, 126(1):55–82, 2003.

[2] J.Dean and S.Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, pages 137–150, 2004.

[3] B.Bahmani, K.Chakrabarti, and D.Xin. Fast personalized pagerank on mapreduce. In SIGMOD, pages 973–984, 2011.

[4] Apache Hadoop. <http://hadoop.apache.org>.

[5] C.Aggarwal, Y.Xie, and P.Yu. Gconnect: A connectivity index for massive disk-resident graphs. PVLDB, 2(1):862–873, 2009.

[6] J.Cheng, Y.Ke, A.W.Fu, J.X.Yu, and L.Zhu. Finding maximal cliques in massive networks by h\*-graph. In SIGMOD, pages 447–458, 2010.

[7] D.Wagner and T.Willhalm. Speed-up techniques for shortest-path computations. In STACS, pages 23–36, 2007.

[8] M.Potamias, F.Bonchi, C.Castillo, and A.Gionis. Fastbshortest path distance estimation in large networks. In CIKM, pages 453–470, 2009.

[9] A.Goldberg and C.Harrelson. Computing the shortest path: search meets graph theory. In SODA, pages 156–165, 2005.

[10] E.Cohen, E.Halperin, H.Kaplan, and U.Zwick. Reachability and distance queries via 2-hop labels. In SODA, pages 937–946, 2002.

[11] F. Wei. Tedi: efficient shortest path query answering ongraphs. In SIGMOD, pages 99–110, 2010.

[12] B.Zou, X.Ma, B.Kemme, G.Newton, and D.Precup. Data mining using relational database management systems. In PAKDD, pages 657–667, 2006.

[13] C.Mayfield, J.Neville, and S.Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In SIGMOD, pages 75–86, 2010.

[14] J.Shanmugasundaram, K.Tufte, C.Zhang, G.He, D.DeWitt, and J.Naughton. Relational databases for querying xml documents: Limitations and opportunities. In VLDB, pages

302–314, 1999.

[15] F.Tian, B.Reinwald, H.Pirahesh, T.Mayr, and J.Myllymaki. Implementing a scalable xml publish/subscribe system using a relational database system. In SIGMOD, pages 479–490, 2004.

[16] M.Benedikt, C.Chan, W.Fan, R.Rastogi, S.Zheng, and A.Zhou. Dtd-directed publishing with attribute translation grammars. In VLDB, pages 838–849, 2002.

[17] S.Srihari, S.Chandrashekar, and S.Parthasarathy. A framework for sql-based mining of large graphs on relational databases. In PAKDD, pages 160–167, 2010.

[18] C.Wang, W.Wang, J.Pei, Y.Zhu, and B.Shi. Scalable mining of large disk-based graph databases. In SIGKDD, pages 316–325, 2004.