Térbeli-szöveges hasonlósági összekapcsolások

Tanulmány: Panagiootis Bouros, Shen Ge és Nikos Mamoulis: Spatio-Textual Similarity Joins c. cikke alapján.

Gergácz Dániel (FAZXPD)
Nagy Tamás (SAZIP1)
Tomcsik Bence Tibor (EVSD6N)

Absztrakt

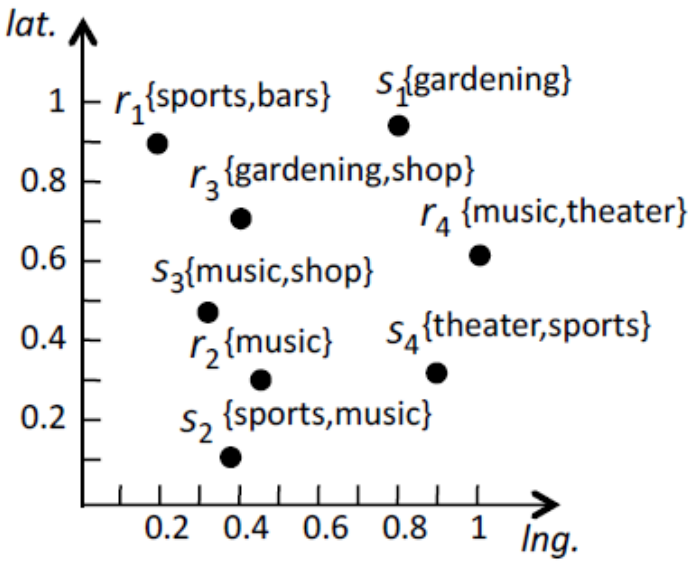
Adott objektumok egy gyűjteménye, amelyek egyaránt tartalmaznak térbeli és szöveges információkat, a térbeli-szöveges hasonlósági összekapcsolás előállítja az objektumok azon párjait, amelyek térben közeliek és szövegesen hasonlóak.
Példaként tekintsünk egy közösségi hálót, térben és szövegesen címkézett személyekkel. Hasznos feladat lehet olyan párokat találni akik helyileg közel vannak egymáshoz és a profiljukban nagy az átfedés (pl.: közös érdeklődési körük van).

A tanulmányban bemutatásra kerülnek a korszerű térbeli- és a halmaz hasonlósági összekapcsolások, továbbá olyan hatékony algoritmusok, amelyek figyelembe veszik a térbeli és a szöveges megszorításokat is. Ezt követően olyan kötegelt adatfeldolgozó technikákról lesz szó, amelyek az újabb algoritmusok hatékonyságát növelik. A tanulmány végén valós és mesterséges adathalmazokon végzett mérések összehasonlításával látható lesz, hogy a cikk szerzői alapján kidolgozott technikák nagyságrenddel gyorsabbak a kiindulási megoldásoknál.

Bevezetés

Az évek során az adatbázisok egyre bonyolultabbakká váltak azáltal, hogy az entitások különböző típusú kiegészítő információkkal címkézhetők fel, pl.: kulcsszavak és térbeli pozíciókkal.
Az utóbbi időben növekvő érdeklődés figyelhető meg mind a kutatási-, mind az ipari környezetben az iránt, hogy a teret, mint egy újabb dimenziót, a meglévő adatok rendezéséhez és bennük szöveges kereséshez használják.
Ezzel az irányzattal összhangban vizsgálják a cikk írói a térbeli-szöveges hasonlósági összekapcsolás (ST-SJOIN) típusú lekérdezések kiértékelését.

Adott objektumok két gyűjteménye, R és S, amelyek térbeli és szöveges információkat tartalmaznak. Az ST-SJOIN előállítja $R \times S$ J részhalmazát amelyben minden $(r, s) \in J$-re, r térben közel van s-hez egy távolság küszöb alapján ($dist_l(r, s) \leq \epsilon$, ahol $dist_l$ a pozíciók közötti távolságot jelöli) és a halmaz tartalmának hasonlósága r és s között átlép egy θ küszöböt ($sim_t(r, s) \geq \theta$, ahol sim_t a szöveges hasonlóságot jelöli).



Az 1. ábrán látható négy férfi (r_1-től r_4-ig) és négy nő (s_1-től s_4-ig), akiket a pozíciójuk és az érdeklődési körül alapján kapcsolhatunk össze.
Feltételként adjuk meg, hogy a lehetséges pároknak legfeljebb $\epsilon = 0.3$-as euklideszi távolságúnak ($dist_l$) és legalább $\theta = 0.5$-ös Jaccard hasonlóságúnak kell lennie. Ekkor az összekapcsolás eredménye $\{(r_2, s_2), (r_2, s_3)\}$.

Alkalmazások:
Az ST-SJOIN számos területen alkalmazható, ahol az entitásokhoz térbeli és szöveges információ kapcsolódik.
Néhány példa:

1. **Személyes adatbázisok**
   Ahogy a fenti példából is kiderült, a közösségi hálók az azonos profillal és pozícióval rendelkező személyek számára használhatják az összekapcsolást. Az eredmény "ajánlásként" ismert a legtöbb közösségi oldalon (pl.: "Kit ismerhetek" funkció).
   Ügyfeleket tartalmazó adatbázisban olyan csoportok megtalálása akik közel laknak egymáshoz és azonos a profiljuk. Ez az eredmény felhasználható direkt marketing célból.
2. **Redundáns és "piszkos" adat**
   Adattömörítés és tisztítás a halmaz-hasonlósági összekapcsolás tipikus alkalmazása. Alap esetben az adatok szöveges hasonlósága van figyelembe véve, de a hatékonyság növelhető ha ki tudjuk szűrni a közel azonos entitásokat (pl.: egy kép amely "híd"-ként van címkézve, a világ minden hídját figyelembe veheti, kivéve, ha nem csak a szöveget, hanem a pozíciót is használjuk az összekapcsoláskor).
3. **POI-val (point of interest) rendelkező adatbázisok**
   Olyan üzleteket tud összekapcsolni amelyek közös jellemzőkkel, témával foglalkoznak (pl.: kínai étterem) és közel vannak egymáshoz. Ezt felhasználva a kollaboráció adta lehetőségek széles körben kihasználhatóak (hirdetések, piackutatás, kedvezmények, beszerzési együttműködés).

Ebből a néhány példából is kiderült, hogy a lehetőségek száma jóval túlmutat az elvárthoz képest.

Kapcsolódó munkák

Ahogy arról korábban szó esett, számos cikk született a térbeli-szöveges hasonlósági lekérdezésekről az utóbbi időben [2, 3, 4, 5, 6]. Ezen lekérdezések bemenete egy l térbeli pozíció és kifejezések egy K halmaza és a cél, hogy egy R gyűjteményből objektumokat találjunk meg a lekérdezés által, amelyek térben közel vannak l-hez és szövegesen hasonlóak K-val.

A cikk írói megjegyzik, hogy tudomásuk szerint csak egyetlen olyan cikk van [7] a szöveges-térbeli hasonlósági összekapcsolásoknál, ahol a probléma eltérően van megfogalmazva:
Adott két adathalmaz R és S, minden R-ben lévő objektumra a legjobb S-beli párja adott. A cikk erre a lekérdezésre próbál hatékony megoldásokat találni. A cikk az $R = S$ esetet vizsgálja, mert ez a legjobban bemutatható a lekérdezéseken, de megjegyzik, hogy könnyedén kiterjeszthető az $R \neq S$ általános esetre.

Probléma definíció

Egy térbeli-szöveges x objektumot az $(x.id, x.loc, x.text)$ hármas ír le, amely az azonosító, a pozíció, és x szöveges leírásának modellezésére szolgál.
$x.loc$ a két dimenziós térből veszi az értékeit, $x.text$ kifejezések egy halmaza, amelyek egy véges globális szótárból $T = \{t_1, t_2, \cdots, t_n\}$ veszik az értékeiket. Minden t kifejezés az $x.text$-ben súlyozott (alapértelmezett súly az 1, a súlyozatlan halmazok esetén), amely t x-hez viszonyított jelentőségét hivatott modellezni.
x objektum mérete, $|x|$, az $x.text$-ben lévő kifejezések számát jelenti.
Minden térbeli-szöveges x és y objektumhoz megadható a térbeli távolságuk, $dist_l(x, y)$ $x.loc$ és $y.loc$ alapján, valamint a szöveges hasonlóságuk, $sim_t(x, y)$, amely a halmaz hasonlóság $x.text$ és $y.text$ halmazok között (az átfedések száma, Jaccard hasonlóság).

A tanulmány hátralévő részében feltesszük, hogy x és y objektum közötti térbeli távolság a pozíciójuk euklideszi távolsága, $dist_l(x, y) = dist(x.loc, y.loc)$ és a szöveges hasonlóságuk egyenlő a Jaccard hasonlósággal, $sim_t(x, y) = \frac{|x.text \cap y.text|}{|x.text \cup y.text|}$.

Definíció:

Adott R-beli térbeli-szöveges objektumok egy gyűjteménye, a térbeli-szöveges hasonlósági összekapcsolás (ST-SJOIN) olyan R-beli objektum párokat állít elő, amelyek mind térben közeliek egymáshoz, mind szövegesen hasonlóak.

Formálisan, adott egy térbeli ϵ távolság küszöb és egy θ szöveges hasonlósági küszöb, amelyekre ST-SJOIN(R, ϵ, θ) előállítja az összes (x, y) párt, $x, y \in R$, amelyekre $dist_l(x, y) \leq \epsilon$ és $sim_t(x, y) \geq \theta$.

Halmaz hasonlósági összekapcsolások háttere

Egy hatékony módja a szöveges hasonlósági összekapcsolás számításának az invertált fájlok [8] használata. Egy gyűjtemény R objektumainak az invertált fájlja egy olyan index amely egy globális T szótárban lévő összes t kifejezést egy $x \in R$-beli objektumok (amely tartalmazza t-t) postings listájával (az indexek listája) azonosít.

Tegyük fel, hogy létezik R-nek egy ilyen indexe. Ahhoz, hogy kiszámoljuk az összekapcsolást minden $x \in R$-re, először minden $t \in x.text$ L_t postings listáján végzünk próbát és összegyűjtjük x minden $y \in L_t$ objektummal való egybeesését az $O_x[y]$ halmazba. Így előállítjuk egy x objektumhoz az összes (x, y) lehetséges párok halmazát. Ezek után vesszük az összes $y \in O_x$-et, $O_x[y] > 0$, és kiszámítjuk $sim_t(x, y)$ értékét. Ha $sim_t(x, y) \geq \theta$ akkor az (x, y) pár hozzá lesz adva az összekapcsolás eredményéhez.
A fő probléma ezzel az eljárással az, hogy a gyakori kifejezések postings listái nagyon hosszúak lehetnek, ebből adódóan, nagy számú lehetséges párok generálódnak.
Ennek érdekében egy szűrő segítségével csökkenteni kell a lehetséges párok számát.

Prefix szűrő:
Először kanonizálnunk kell minden x objektumot:

Sorba rendezzük $x.text$-ben lévő kifejezéseket egy globális $\mathcal{O}$ szerint, amely az $x.text$-ben lévő legritkább kifejezéseket előrébb helyezi. Majd vesszük $x.text\ ppref(x)$ prefixét (prefix próba). $ppref(x)$ l_p^x hossza $|x|$-től, a hasonlósági függvénytől és a hasonlósági küszöbtől függ.
A jelen esetben, ahol Jaccard hasonlóságot használunk és fenntartunk egy θ küszöböt, $l_p^x = |x| - \lceil \theta \cdot |x| \rceil + 1$.
Figyelembe véve a prefix szűrő alapelvét, ahhoz, hogy két objektum, x és y, hasonló legyen, $ppref(x)$-nek és $ppref(y)$-nak legalább egy közös kifejezést kell tartalmaznia. Így minden x-hez, hogy előállítsuk O_x lehetséges párjait, csak a $ppref(x)$-ben lévő kifejezéseket kell végigjárnunk és próbálnunk.
Ez kisebb számú lehetséges párokhoz vezet és jelentősen csökkenti a lekönyvelendő műveletek számát.

ALL-PAIRS [9] algoritmus

Prefix szűrő elvén alapul, továbbá az invertált index méretének csökkentésére is lehetőséget ad.
Az algoritmus a gyűjtemény objektumait méretük szerint növekvő sorrendben vizsgálja. Minden x objektumra, ALL-PAIRS minden $t \in \text{ppref}(x)$ kifejezés L_t postings listáján végez próbát. Az objektumok sorrendje miatt,csak azokat a kifejezéseket kell indexelnie, amik az $x.text$ index prefixében vannak ($ipref(x)$), ahelyett, hogy a $ppref(x)$-et indexelné.

PPJOIN algoritmus

A PPJOIN algoritmus kibővíti az ALL-PAIRS algoritmust egy pozíció- és egy suffix szűrővel. Pozíciós információ alatt egy kanonizált x objektum $x.text$ halmazában lévő kifejezés pozícióját értjük.

Adott két objektum x és y, a pozíciós szűrő alapötlete, hogy az $O(x,y)$ átfedéseikre egy $\bar{O}$ felsőkorlátot számoljunk. Ha $\bar{O}$ kisebb mint x és y minimális átfedése, amelyet a θ határküszöb által követelünk meg, akkor nyugodtan eldobhatjuk az (x,y) párt.
A suffix filter esetében, a PPJOIN algoritmus az "oszd meg és uralkodj" módon működik x és y objektumok suffixein, azért, hogy egy $\underline{H}$ alsó korlátot számoljon a Hamming távolságukból ($H(x,y)$). Ha $\underline{H}$ nagyobb mint a maximális Hamming távolság, amely a pár által van megkövetelve, hogy teljesüljön θ, akkor nyugodtan eldobhajtuk (x,y) párt.

Algorithm 1: PPJOIN(R, θ)

input : R is a collection of objects sorted by the increasing order of their sizes - each object is canonicalized by a global ordering $\mathcal{O}$; a textual similarity threshold θ
output : the set J of all object pairs (x, y), such that $x, y \in R$ and $sim_t(x, y) \geq \theta$

```
1  foreach term t ∈ T do
2      L_t ← ∅
3  foreach object x ∈ R do
4      ℓ_p^x ← |x| − ⌈θ · |x|⌉ + 1;                    // Probe prefix length
5      ℓ_i^x ← |x| − ⌈(2θ/(θ+1)) · |x|⌉ + 1;            // Index prefix length
6      for pos_x = 1 to ℓ_p^x do
7          t ← term of x.text at position pos_x;
8          foreach entry ⟨y, pos_y⟩ ∈ L_t such that |y| ≥ θ · |x| do
9              if QualifyPositionalFilter(x, pos_x, y, pos_y) and
                 QualifySuffixFilter(x, pos_x, y, pos_y) then
10                 O_x[y] ← O_x[y] + 1;                // Increase overlap
11             else
12                 O_x[y] ← −∞;                        // Prune pair
13         if pos_x ≤ ℓ_i^x then
14             L_t ← L_t ∪ {⟨x, pos_x⟩};               // Build/extend index
15     Verify(x, O_x, J);
16 return J;
```

Térbeli-szöveges hasonlósági összekapcsolások

Ahhoz, hogy a PPJOIN algoritmust arra használjuk, hogy ST-SJOIN-t számítsunk ki, minden olyan y-t el kell vetnünk, amely a távolságra vonatkozó megszorítást nem teljesíti x-szel szemben.
Ezt az ötletet alkalmazza a PPJ algoritmus.

Algorithm 2: PPJ(R, ϵ, θ)

input : R is a collection of spatio-textual objects sorted by the increasing order of their sizes - each object is canonicalized by a global ordering $\mathcal{O}$; a spatial distance threshold ϵ; a textual similarity threshold θ
output : the set J of all object pairs (x, y), such that $x, y \in R$, $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$

... // Lines 1-7 in Algorithm 1
8 **foreach** *entry* $\langle y, pos_y \rangle \in L_t$ **such that** $dist_l(x, y) \leq \epsilon$ **and** $|y| \geq \theta \cdot |x|$ **do**
... // Lines 9-16 in Algorithm 1

A PPJOIN-hoz képest a PPJ algoritmus még egy szűrőt használ a 8. sorban, azért, hogy két objektum, x és y, között leellenőrizze a térbeli távolságot. Megjegyzendő, hogy a térbeli távolság szűrő sokkal kevesebb költséggel jár mint a pozíciós- és suffix szűrő és éppen ezért kell elsőként alkalmazni.
Annak érdekében, hogy a PPJ-nél hatékonyabb eljárásokat állítsunk elő, ki kell használnunk az ST-SJOIN "természetét". Ahhoz, hogy azonosítsunk, minden R-beli x objektumhoz, minden szóba jövő (x, y) párt hatékonyan. ki kell használnunk az ϵ térbeli távolság küszöböt, azért, hogy csak azokat az y objektumokat vegyük figyelembe, amelyekre $dist_l(x, y) \leq \epsilon$.

A legfőbb hátránya a PPJ algoritmusnak, hogy nem képes csupán ezeket az objektumokat vizsgálni (pl.: minden t kifejezésnél amely az $x.text$ szűrő próbájában van, PPJ az összes L_t postings listában lévő y objektumot vizsgálja, függetlenül attól, hogy y milyen messze van x-től a térben).
PPJ előbb említett hiányossága teszi szükségessé egy olyan térbeli indexelő eljárás bevezetését, amely gyorsan kiritkítja a lehetséges párok számát a térbeli távolságuk alapján. A következőkben erről lesz szó.

Dinamikus rács particionálás

Adott egy ST-SJOIN(R, ϵ, θ) lekérdezés egy R adatgyűjteményen, R kétdimenziós terének dinamikus felosztása G_R. A rácsban minden négyzetrács kiterjedése minden dimenzióban megegyezik az ϵ térbeli távolság küszöbbel, ezért a rács nem előre kiszámított, hanem dinamikusan van meghatározva a lekérdezés paraméterei által.

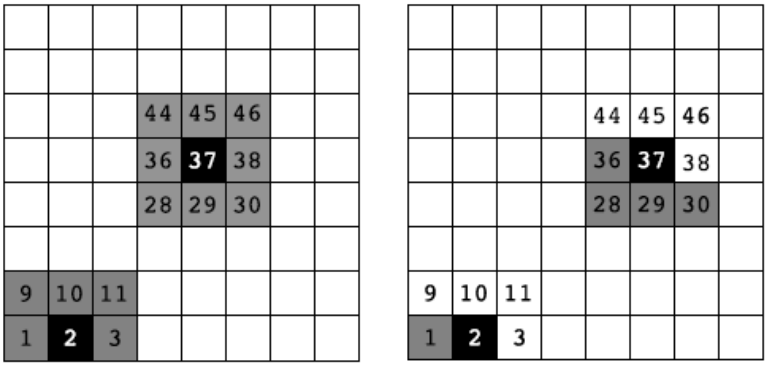


Figure 3: Employing dynamic grid partitioning: (a) PPJ-I defines cell intervals, (b) PPJ-C identifies join cells

Tegyük fel, hogy van egy vizsgálandó x objektum amelynek az azonosítója 37. Ahhoz, hogy kiválogassuk a lehetséges (x, y) párokat, csak azokat az objektumokat kell vizsgálnunk, amelyek a 37-es cellában és a mellette lévő 8 szomszédos cellában vannak. Általánosan, legfeljebb 9 cella van vizsgálva minden egyes x objektum esetében. A szomszédos cellák az objektumok egy bővebb halmazát tartalmazzák, x-től legfeljebb ϵ távolságra; ezért továbbra is alkalmazni kell a térbeli távolság szűrőt minden lehetséges pár esetén ezekben a cellákban. Az ábrán a cellák sorfolytonosan és lentről felfelé vannak számozva.

A következő két fejezetben a PPJ algoritmus két kibővítéséről lesz szó amelyek kihasználják a dinamikus rács felosztást, hogy felgyorsítsák az ST-SJOIN számítási sebességét.

Fontos megjegyezni, hogy a továbbiakban egy $(x.id, x.loc, x.text, x.cid)$ négyest használunk egy x objektum modellezésére, ahol $x.cid$ a (dinamikusan meghatározott) rácsazonosító, amely tartalmazza $x.loc$-ot.

PPJ-I algoritmus

- Az összekapcsolás kiértékelése előtt, az adott ϵ távolság küszöb alapján, PPJ-I előállítja a tér rácsfelosztását. Minden c rácscella esetén, PPJ-I meghatároz legfeljebb 9, c-vel szomszédos cellát és az azonosítójukból legfeljebb 3 intervallumot (innen a '-I' az algoritmus nevében). Ezek az intervallumok határozzák meg a térbeli régióját minden c-beli objektumnak. Pl.: az előző ábrán lévő 37-es cella esetében PPJ-I előállítja a [28, 30], [36, 38], [44, 46] intervallumokat, míg a 2-es cella esetében csak két intervallumot [1, 3] és [9, 11] állít elő.
- PPJ-I megtartja minden postings lista $\langle y, pos_y \rangle$ elemeit, $y.cid$ szerinti növekvő sorrendben.
- Egy úgynevezett "lightweight" cella-index is megtartásra kerül, minden L_t postings lista után. A cella-index minden egyes (cid) cella esetén tartalmaz egy $\langle cid, p_{cid} \rangle$ bejegyzést, amelynek L_t-beli objektumai vannak. p_{cid} egy olyan pointer amely

hozzáférést nyújt az L_t-beli $\langle y, pos_y \rangle$ első bejegyzéshez, amelynél $y.cid = cid$. Legrosszabb esetben a cella-index mérete megegyezik a rácsban lévő cellák számával. Gyakorlatban ez sokkal kisebb mint maga a lista.

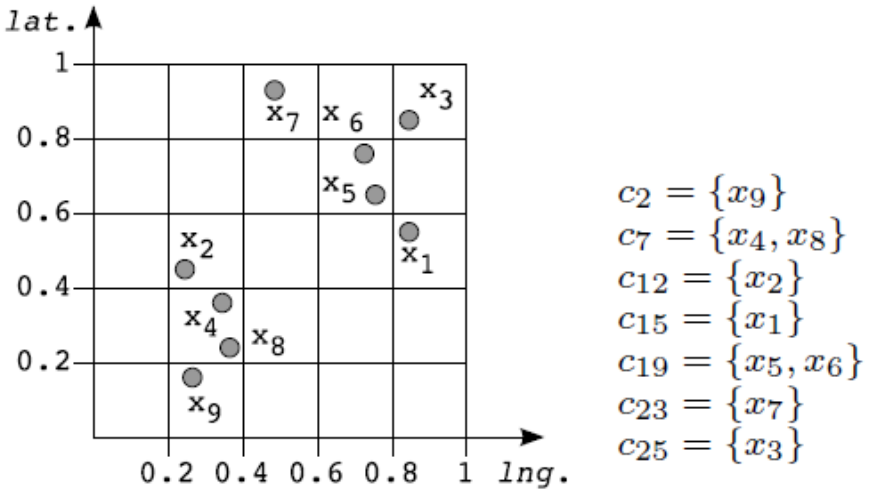
Algorithm 3: PPJ-I(R, ϵ, θ)

input : R is a collection of spatio-textual objects sorted by the increasing order of their sizes - each object is canonicalized by a global ordering $\mathcal{O}$; a spatial distance threshold ϵ; a textual similarity threshold θ

output : the set J of all object pairs (x, y), such that $x, y \in R$, $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$

```
1  G_R ← ConstructGridPartitioning(R, ε);
2  foreach cell c in G_R do
3      I[c] ← DefineCellIntervals(c, G_R);
   ... // Lines 1-7 in Algorithm 1
11 foreach interval i[c_min, c_max] ∈ I[c] do
12     foreach entry ⟨y, pos_y⟩ ∈ L_t such that c_min ≤ y.cid ≤ c_max and
       dist_l(x, y) ≤ ε and |y| ≥ θ · |x| do
           ... // Lines 9-14 in Algorithm 1
19     Verify(x, O_x, J)
20 return J;
```

Példa:



$c_2 = \{x_9\}$
$c_7 = \{x_4, x_8\}$
$c_{12} = \{x_2\}$
$c_{15} = \{x_1\}$
$c_{19} = \{x_5, x_6\}$
$c_{23} = \{x_7\}$
$c_{25} = \{x_3\}$

Figure 4: Dynamic grid partition G_R for the collection R in Figure 2, with $\epsilon = 0.2$

A PPJ-I algoritmus létrehozza a rácsfelosztást (25 cella, $\epsilon = 0.2$). Majd minden megvizsgált x obejktumra az $x.cid$ dinamikusan meghatározásra kerül és minden $t \in x.text$ kifejezéshez a megfelelő L_t-n keresést hajt végre a calla-index segítségével azért, hogy megtalálja a postingokat a szomszédos cellákban. Végül x beszúrásra kerül L_t-be.

A szűrő fázis folyamán PPJ-I a rács segítségével csupán 6 pár vesz figyelembe a 13-ból amiket PPJ vett figyelembe:
$(x5, x3), (x6, x3)$ a D kifejezés esetén, $(x6, x5)$ a C kifejezés esetén és $(x8, x4), (x9, x4), (x9, x8)$ az A kifejezés esetén,

PPJ-C algoritmus

PPJ és PPJ-I algoritmusok egy R-beli gyűjtemény objektumait méretük szerinti növekvő sorrendben vizsgálják.
PPJ-C algoritmus főbb jellemezői:

- PPJ-C, hasonlóan PPJ-I-hez, dinamikus rácsfelosztást használ. Ezt követően R objektumai a cella-azonosítójuk növekvő sorrendje szerint vannak megvizsgálja (innen a '-C' az algoritmus nevében). Tehát az összekapcsolás előtt, PPJ-C elsősorban a cella-azonosító szerint, másodsorban a méret szerint rendezi az objektumokat.
- Minden egyes c cellához, PPJ-C, cellák egy $A[c]$ halmazát azonosítja, amely tartalmazza az összes y objektumot, amely össze lenne kapcsolva minden x objektummal c-ben. $A[c]$ magát a c cellát is tartalmazza valamint minden c-vel szomszédos, nála kisebb azonosítóval rendelkező cellákat is. Ezért minden szomszédos cellapár csak egyszer van figyelembe véve. (Pl.: a korábbi ábrán a 37-es cellát csak ugyanabban a cellában lévő objektumokkal, valamint a szomszédos cellákban lévőkkel kell összekapcsolni: $A[37] = \{28, 29, 30, 36, 37\}$ A többi szomszédos de nagyobb azonosítóval rendelkező cellák később lesznek összekapcsolva x objektummal).
- Ellentétben PPJ-I-vel (és PPJ-vel), PPJ-C egy invertált indexet épít ki minden c rácscella számára, egy globális index helyett. Így minden t kifejezéshez a globális szótárban van egy $c.L_t$ postings lista minden c cellához, ami azt eredményezi, hogy c tartalmaz legalább egy x objektumot $t \in ipref(x)$ esetén. Ez a séma előnyben van az előző kettővel szemben a tárhely kapacitás szempontjából, mivel PPJ-C eldobja a c cellát és a tartalmát minden c' cella után, amelyre $A[c']$ tartalmazza c-t, míg PPJ-I megtartja az összes megvizsgált, indexelt adatot, amíg az algoritmus véget nem ér.

Algorithm 4: PPJ-C(R, ϵ, θ)

input : R is a collection of spatio-textual objects sorted by the increasing order of their sizes - each object is canonicalized by a global ordering $\mathcal{O}$; a spatial distance threshold ϵ; a textual similarity threshold θ
output : the set J of all object pairs (x, y), such that $x, y \in R$, $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$

```
1 G_R ← ConstructGridPartitioning(R, ε);
2 foreach cell c in G_R do
3     A[c] ← IdentifyJoinCells(G_R, c);
4     foreach cell c' in A[c] do
5         J ← J ∪ PPJ(c, c', ε, θ);
6 return J;
```

PPJ-R algoritmus

Tegyük fel, hogy az objektumok egy T_R R-fa által vannak indexelve a térben és adott PPJ-R eljárás, amely a PPJ-t egészíti ki azzal, hogy a térbeli indexeken dolgozik.
Az algoritmus elsősorban egy térbeli ϵ távolságú összekapcsolásként működik. Az R fa két csúcsát, N_x és N_y, kapja bemenetként (első alkalommal, $N_x = N_y$ a fa gyökere).

Ha N_x és N_y nem levél csúcsok, akkor PPJ-R az összes bejegyzés párra, $(e_x, e_y) \in N_x \times N_y$, amelyekre a minimális távolság e_x és e_y minimális befoglaló téglalapjára legfeljebb az ST-SJOIN ϵ küszöbének térbeli távolságával egyenlő. Ezek a bejegyzések olyan objektum párokhoz vezetnek, amelyek teljesítik az összekötés térbeli megszorításait, ezáltal az algoritmus rekurzívan fut az (e_x, e_y) által mutatott csúcspárokra. Ha N_x és N_y levél csúcsok, akkor PPJ-R összekapcsolja őket.

Az algoritmusban szereplő $E_\epsilon(\cdot)$ azt jelenti, hogy $MBR(N_x)$ és $MBR(N_y)$ (Minimal bounding rectangle) ki van terjesztve ϵ-nal minden dimenzióban és irányban és az A metszetük kiszámításra kerül. Minden $e_x \in N_x$ bejegyzés ($e_y \in N_y$) amely az $MBR(e_x)$ ($MBR(e_y)$) által nem metszik egymást, eldobásra kerülnek. Ezt követően a megmaradt bejegyzések az alacsonyabb x-dimenziós határjaik szerint vannak sorrendezve.
Végül a metszetük összekapcsolása, ami I-vel van jelölve, egy síkbeli-pásztázó heurisztika szerint van kiszámítva [11]. I tartalmaz minden (e_x, e_y) párokat, amelyekre $e_x \in N_x$, $e_y \in N_y$ és a távolság köztük minden dimenzióban kisebb vagy egyenlő mint ϵ.

Ez az eljárás nagyon hasonlít a PPJ-C algoritmushoz, kivéve, hogy térbeli felosztások amelyek össze vannak kapcsolva, az R-fa struktúrája által vannak meghatározva és nem a dinamikus rácsfelosztás által, valamint az R-fa térbeli partíciók párjainak megtalálására szolgál amelyeket PPJ által össze kellene kapcsolni.

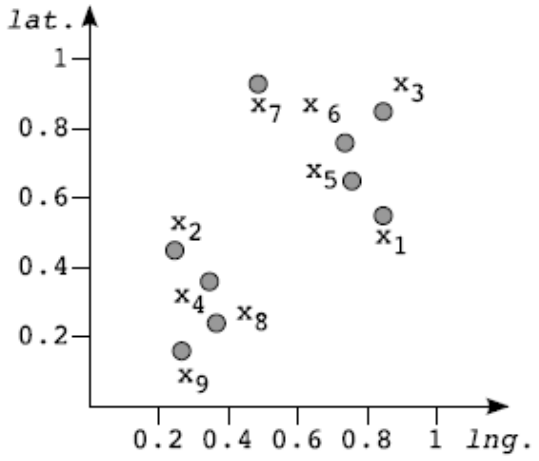
Algorithm 5: PPJ-R$(T_R, N_x, N_y, \epsilon, \theta)$

input : an R-tree T_R indexing a collection of objects R; two nodes N_x and N_y at the same level of T_R; a spatial distance threshold ϵ; a textual similarity threshold θ
output : the set J of all object pairs (x, y), such that $x, y \in R$, $dist_l(x, y) \leq \epsilon$ and $sim_t(x, y) \geq \theta$

1 **if** N_x **and** N_y *are leaf nodes* **then**
2 $J \leftarrow J \cup$ PPJ$(N_x, N_y, \epsilon, \theta)$;
3 $A \leftarrow$ MBR$(E_\epsilon(N_x)) \cap$ MBR$(E_\epsilon(N_y))$; // Space Restriction
4 $M_x = \{$MBR$(e_x) \mid (e_x \in N_y) \wedge (MBR(e_x) \cap A \neq \emptyset)\}$;
5 $M_y = \{E_\epsilon($MBR$(e_y)) \mid (e_y \in N_y) \wedge (MBR(e_y) \cap A \neq \emptyset)\}$;
6 Sort(M_x); Sort(M_y);
7 $I \leftarrow$ PlaneSweepIntersectionTest(M_x, M_y); // Plane Sweep
8 **foreach** *pair of entries* $(e_x, e_y) \in I$ **such that** $dist_l(e_x, e_y) \leq \epsilon$ **do**
9 $J \leftarrow J \cup$ PPJ-R$(T_R, e_x.ptr, e_y.ptr, \epsilon, \theta)$; // Recursion
10 **return** J;

Objektumok csoportosítása

Tekintsük objektumok egy R gyűjteményét (ábra) továbbá egy szöveges hasonlósági összekapcsoló lekérdezést R-en egy $\theta = 0.7$ küszöbbel.



x_1: $\{B, C\}$
x_2: $\{E, F\}$
x_3: $\{D, E, F\}$
x_4: $\{A, B, E, F\}$
x_5: $\{C, D, E, F\}$
x_6: $\{C, D, E, F\}$
x_7: $\{A, B, C, D, F\}$
x_8: $\{A, B, D, E, F\}$
x_9: $\{A, B, C, D, E\}$

Figure 2: A collection of spatio-textual objects

Az alábbi táblázat tartalmazza minden x objektumra a $ppref(x)$-t, az $l_p^x = |x| - \lceil \theta \cdot |x| \rceil + 1$ formula alkalmazásával.

Megfigyelhető, hogy számos objektum között van azonos prefix próba. Példaként, amikor az x_4, x_7, x_8 és x_9-et vizsgáljuk, akkor akármelyik algoritmus előállítaná az L_A és L_B postings listákat, majd kiszámolná és összegezné ugyanazt az O_x átfedést az összes objektumokra. Továbbá, mivel $ipref(x)$ részhalmaza $ppref(x)$, az algoritmus ugyanazt a prefix indexet indexelné 4 alkalommal is.

Másik oldalról vizsgálva, tegyük fel, hogy először csoportosítjuk az összes objektumot amelynek megegyezik a prefix próbája, majd alkalmazzuk ugyanazt az összekapcsoló algoritmust a megegyező prefixű objektumok csoportján. Ez a megközelítés kikerülné a felesleges műveleteket továbbá ami még fontosabb, hogy lehetővé tenné számunkra, hogy nagy mértékben csökkentsük az objektumokat.

Table 4: Probe prefixes for the spatio-textual objects in Figure 2

object x	$x.text$	$ppref(x,), \theta = 0.7$
x_1	$\{B, C\}$	$\{B\}$
x_2	$\{E, F\}$	$\{E\}$
x_3	$\{D, E, F\}$	$\{D\}$
x_4	$\{A, B, E, F\}$	$\{A, B\}$
x_5	$\{C, D, E, F\}$	$\{C, D\}$
x_6	$\{C, D, E, F\}$	$\{C, D\}$
x_7	$\{A, B, C, D, F\}$	$\{A, B\}$
x_8	$\{A, B, D, E, F\}$	$\{A, B\}$
x_9	$\{A, B, C, D, E\}$	$\{A, B\}$

Példa:

Tekintsük a következő két csoportot, $\{x_5, x_6\}$ és $\{x_4, x_7, x_8, x_9\}$. Mivel a prefix próbája ezeknek a csoportoknak, $\{C, D\}$ és $\{A, B\}$, nem tartalmaz közös kifejezést, egyenesen eldobhatjuk az összes lehetséges objektum párokat az által, hogy egyszer megvizsgáljuk a csoport objektumait. Ellenben PPJOIN-nak 5 párt kellene ellenőriznie: $(x_5, x_4), (x_6, x_4), (x_7, x_5), (x_8, x_5), (x_9, x_5)$.

Másrészről PPJOIN kiegészítése ezzel a csoportosító technikával, két hiányosságot vezet be:

- Elsőként azt, hogy további költséget vezet be az ellenőrző fázis alatt, hogy kifejtse az objektumokat minden csoport párjai esetén amelyek kielégítik az átfedési küszöböt és azonosítják a végső eredményeket.
- Másodikként azt, hogy a gyűjtemény aktuális objektumait tekintve, a csoportosításon alapú PPJOIN megváltoztathatja a vizsgálat sorrendjét. Más szóval, csoport objektumainak méret szerinti vizsgálata nem egyezik meg az aktuális objektumok méret szerinti vizsgálatával, mivel a különböző hosszúságú objektumoknak megegyezhet a prefixe. Ennek a problémának a kiküszöbölésére figyelembe kell venni az objektumok méretét amikor csoportokat határozunk meg.

Egész pontosan, egy csoport olyan objektumokat tartalmaz, amelyek tartalmaznak közös prefix próbákat és megegyezik a méretük.

Eredmények

Eredmények elemzéséhez használt adatok:

- **FLICKR:** fényképek egy gyűjteménye a Flickr nevű internetes oldalról, amelyek New York városról készültek az elmúlt két évben. 1,505,243 objektumot tartalmaz 726,958 kifejezésből álló szótárral. Minden egyes fényképre a címkék és a cím elemeinek uniója adja a kép szöveges leírását. Az átlagos súlyozott mérete egy ilyen objektumnak 10.5.
- **POI-USCA és POI-AU** két POI gyűjteménye és üzleti felsorolása USA, Kalifornia államnak valamint Ausztráliának. Minden hely esetén, kigyűjtésre került az elhelyezkedésük és a címkéik, kategóriáik, alkategóriáik uniója mint a szöveges leírás. POI-USCA 1,511,837 objektumot tartalmaz egy 16,048 kifejezésből álló szótárral, míg POI-AU 696,212 objektumot és 2,633 kifejezést tartalmaz. Kettő között a fő különbség az objektumaik között lévő eloszlás a két dimenziós térben. Ausztrália földrajza miatt az objektumok meglehetősen csoportosítottak. Az átlagos súlyozott mérete egy ilyen objektumnak 4.4 POI-USCA esetén és 4.7 POI-AU esetén.
- Továbbá **mesterséges adathalmazok**at is használnak a cikkben az eredmények összehasonlításakor. ahol változtatják $|R|$ objektumainak számát egy gyűjteményben, valamint a globális szótár $|T|$ méretét.

"Csoportosítani vagy nem csoportosítani"

Az első mérésnél a csoportosítás optimalizáló hatását vizsgáljuk futás közben az összekapcsolási eljárásokon.

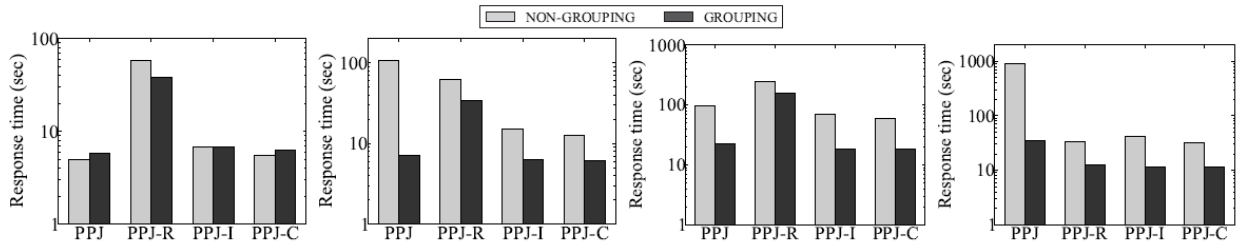


Figure 6: Grouping versus Non-Grouping

Mindegyik algoritmus (PPJ, PPJ-R, PPJ-I, PPJ-C) esetén egymással szembeállítjuk a csoportosított és a nem csoportosított verzióját a korábban definiált adathalmazokon (FLICKR, POI-USCA, POI-AU és mesterséges adathalmaz).

A grafikonok azt mutatják, hogy az algoritmusok csoportosító verziója hasonló teljesítményt produkál a nem-csoportosító verzióhoz képest a legrosszabb esetekben, de a legjobb esetekben kimagaslóan jobban teljesít. Látható az is, hogy a FLICKR esetén nem

tűnik hatásosnak a csoportosító változat, mert ezeknél az adatoknál a csoportok nagy része egyelemű lesz.
Összességében a csoportosítás használata az eljárásokban legalább olyan hatékony mint amikor nincsenek használva.

Látható még egy mérés arra, hogy a csoportosításnak milyen hatása van egy egyszerű szöveges hasonlósági összekapcsolás esetén.

Table 6: Applying the grouping technique to PPJOIN for textual similarity joins: response time (sec)

	GROUPING		NON-GROUPING	
θ	FLICKR	POIS-USCA	FLICKR	POIS-USCA
0.6	67.24	570.97	72	1929.79
0.7	29.01	308.53	30.72	985.1
0.8	12.25	220.4	13.19	700.08
0.9	5.88	181.83	6.21	589.45

Az eredmény azt mutatja, hogy a csoportosítási optimalizáció jelentősen megnöveli a korszerű szöveges hasonlósági összekapcsoló algoritmus teljesítményét.

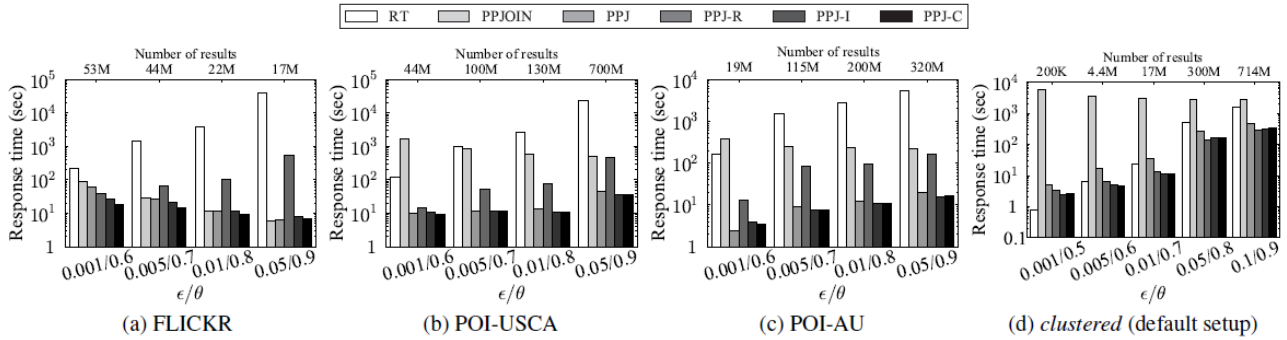
Összehasonlítás az alap algoritmusokkal

Ebben a részben az ST-SJOIN eljárások vannak összevetve két ismert algoritmussal, amelyek eltérően bánnak az összekapcsolás mindegyik dimenziójával: tér és szöveg.

Az RT eljárás egy gyűjtemény objektumait indexeli egy R-fa használatával. ST-SJOIN kiszámításához egy térbeli ϵ távolságú összekapcsolást hajt végre és minden lehetséges (x, y) párra kiszámítja azok $sim_t(x, y)$ értékét és ellenőrzi, hogy $sim_t(x, y) \geq \theta$.

A PPJOIN eljárás egy szöveges hasonlósági összekapcsolást hajt végre, de az ellenőrző fázisban azt is leellenőrzi, hogy $dist_l(x, y) \leq \epsilon$ fennáll e minden lehetséges (x, y) pár esetén.

A PPJ algoritmus két dologban különbözik PPJOIN-tól:
PPJ hamarabb elvégzi a $dist_l(x, y) \leq \epsilon$ ellenőrzést a szűrő fázis folyamán, valamint ahogy az ST-SJOIN eljárásokban, PPJ alkalmazza a csoportosítási heurisztikát a keresés hatékonysága érdekében.



(a) FLICKR (b) POI-USCA (c) POI-AU (d) *clustered* (default setup)

Figure 7: Comparison with RT and PPJOIN baseline methods

Megfigyelhető, hogy az *RT* eljárás válaszideje nő, miközben a PPJOIN esetében csökken. Az is megfigyelhető, hogy a PPJ, PPJ-R, PPJ-I és PPJ-C algoritmusok válaszideje szintén nő, mivel figyelembe veszik a probléma térbeli dimenzióját, szemben a PPJOIN-nal amely ϵ-t csak a szöveges szűrőn átmenő párok ellenőrzésekor veszi figyelembe.

PPJ-R gyenge teljesítménye annak tudható be, hogy az algoritmus által használt R-fa nagy számú levél csúcsokat hoz létre sokkal kisebb kiterjedéssel mint ϵ. Ez azt jelenti, hogy nagy számú levél csúcs párok vannak ϵ távolságon belül, amiket PPJ-R-nek össze kell kapcsolnia. Továbbá a levél csúcsok sokkal kevesebb objektumot tartalmaznak egy PPJ-I/PPJ-C-ben lévő cellához képest, ezért a PPJ-R-beli csoportosítás haszna erősen korlátozott.

ST-SJOIN eljárások összehasonlítása

Objektumok számának változtatása $|R|$

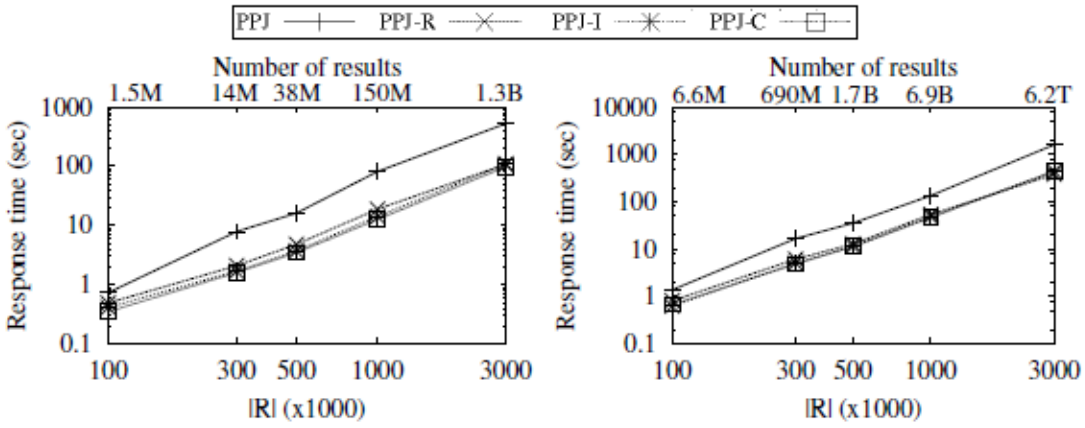


Figure 8: Synthetic collections: varying the number of objects $|R|$, with $|T| = 50{,}000$, $\epsilon = 0.01$, $\theta = 0.7$

Megfigyelhető, hogy PPJ-C nyújtja a legjobb teljesítményt. Valamint, ahogy várható volt, PPJ válaszideje mindig magasabb mint a többi eljárásé, mivel nem alkalmazza a térbeli indexelő technikát, ami kihasználná ϵ-t, hogy a lehetséges párok számát csökkenteni lehessen.

A szótár méretének változtatása $|T|$

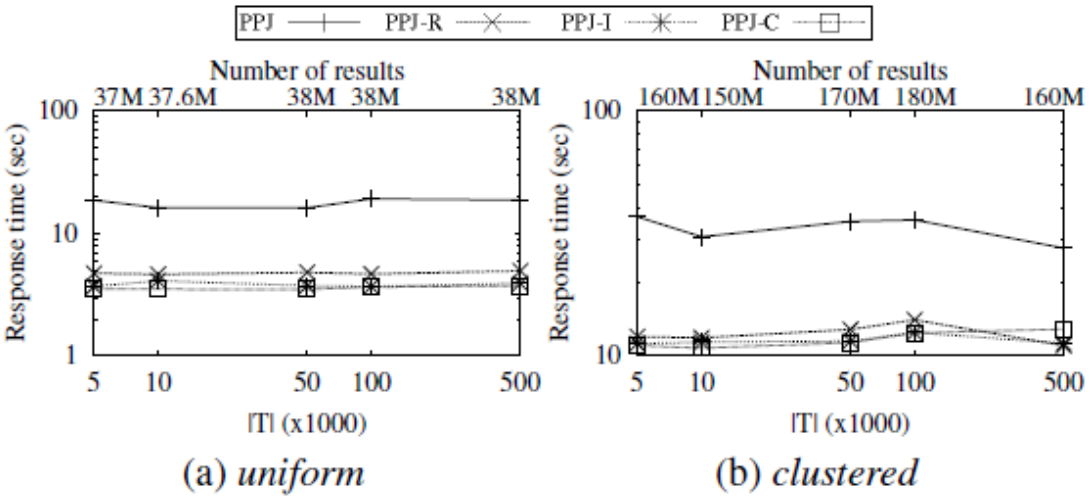


Figure 9: Synthetic collections: varying the size of dictionary $|T|$, with $|R| = 500{,}000$, $\epsilon = 0.01$, $\theta = 0.7$

Az tapasztalható, hogy az eljárásokat nem érinti annyira jelentősen $|T|$ értékének növelése. Az eljárások viselkedése konzisztens az összekapcsolási eredménye számával. Az előző esethez hasonlóan, PPJ-C nyújtja ismét a legjobb teljesítményt és PPJ a legkevésbé hatékony.

A térbeli ϵ távolság küszöb változtatása

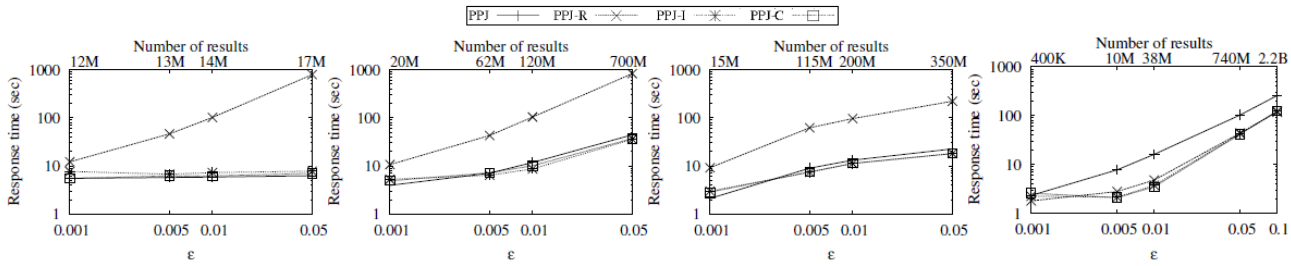
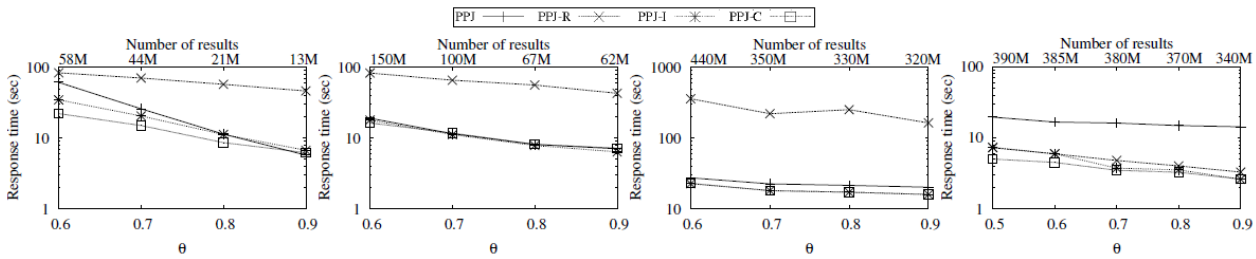


Figure 10: Varying the spatial distance threshold ϵ

ϵ növelésével az összekapcsolásnak egyre több eredménye lesz, ezért mindegyik eljárás válaszideje nő. A PPJ-R hasonlóan teljesít mint a PPJ-C ebben az esetben, mivel ezeknek az eljárásoknak hasonló térbeli felosztásuk van.

A szöveges hasonlósági θ küszöb változtatása



(a) FLICKR $\epsilon = 0.005$ (b) POI-USCA $\epsilon = 0.005$ (c) POI-AU $\epsilon = 0.05$ (d) *uniform* default setup

Figure 11: Varying the textual similarity threshold θ

θ növelésével mindegyik eljárás válaszideje csökken, mivel az összekapcsolási eredmények csökkennek. PPJ-C a leghatékonyabb eljárás, míg a PPJ-R és PPJ a legkevésbé hatékonyabb eljárások.

Összefoglalás

Ebben a tanulmányban a térbeli-szöveges hasonlósági összekapcsolást vizsgáltuk meg (ST-SJOIN), egy olyan lekérdezést amelyet széles körben lehet alkalmazni. Olyan eljárások kerültek bemutatásra amelyek a korszerű térbeli és szöveges összekapcsolásokat egyesítik.

A tanulmány során a PPJ-C algoritmus bizonyult a legjobbnak, amely dinamikusan osztja fel a teret az összekapcsolás térbeli feltétele alapján, majd szétválogatja egy szöveges feltétel alapján, csak szomszédos rácscella párokra.

Kitértünk a csoportosítási optimalizáló technikákra, amelyek növelik minden szöveges és térbeli-szöveges összekapcsolás eljárás teljesítményét.

A cikk írói megjegyezték, hogy a jövőben eltérő típusú térbeli-szöveges összekapcsolások mérésével és alkalmazásával terveznek foglalkozni. Továbbá azt is megjegyezték, hogy annak ellenére, hogy a PPJ-C algoritmus volt a leghatékonyabb, tovább lehetne fejleszteni azzal ha egy eloszott környezetben alkalmaznák az eljárást.

Irodalomjegyzék:

[1] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis: Spatio-Textual Similarity Joins

[2] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. PVLDB, 3(1):373–384, 2010.

[3] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. PVLDB, 2(1):337–348, 2009.

[4] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In ICDE, 2008.

[5] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. IEEE Trans. Knowl. Data Eng., 23(4):585–599, 2011.

[6] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørv°ag. Efficient processing of top-k spatial keyword queries. In SSTD, 2011.

[7] J. Ballesteros, A. Cary, and N. Rishe. Spsjoin: parallel spatial similarity joins. In GIS, pages 481–484, 2011.

[8] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. ACM Trans. Database Syst., 23(4):453–490, 1998.

[9] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In WWW, 2007.

[10] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near duplicate detection. ACM Trans. Database Syst., 36(3):15, 2011.

[11] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r trees. In SIGMOD Conference, 1993.