Gyakori elemhalmazok

Bankó Tibor

June 9, 2010

Tartalom

A kiindulási probléma

A gazdaságnak a legnagyobb profitot általában az együtt vásárolt termékek jelentik (például a sör és a pelenka). Így nagyon fontos ezen termékhalmazok megkeresése.

A gyakori elemhalmaz fogalma

Legyen adott:

- termékek egy halmaza: $A = \{a_1, a_2, ...a_m\}$
- tranzakciók egy halmaza: $T = \{t_1, t_2, ...t_m\}$, ahol $t_j \subseteq A$. Ezt a T-t *bemeneti sorozatnak* elemeit pedig *tranzakcióknak* hívjuk. A tranzakció felfogható egy vásárlás során megvett terméekeknek.

Egy $\alpha \subseteq A$ halmaz *fedése* azon tranzakciók halmaza, melyek azt tartalmazzák. Ekkor α támogatottsága ($supp(\alpha)$) a fedésének számossága. Azok a fontos halmazok, melyek sok tranzakcióban előfordulnak, hiszen ezeket gyakran vásárolták együtt. Emiatt definiálva van egy *min_supp* (támogatottsági küszöb), és csak az ennél nagyobb támogatottságú halmazokkal kell foglalkozni. Ezeket hívjuk *gyakori elemhalmazoknak*.

A gyakori elemhalmaz fogalma (folyt.)

Az A elemein definiálva van egy rendezés. A tranzakciók elemeit ennek megfelelően tartjuk rendezve.
Ritka elemeknek nevezzük azokat az elemeket, melyek egyetlen gyakori elemhalmaznak sem elemei.
Szűrt tranzakciókhoz úgy juthatunk, hogy elhagyjuk a tranzakciókból a ritka elemeket. Ezek úgy sem számítanak, és így kisebb tranzakciókhoz jutunk, amelyek meggyorsítják az algoritmusok futását. Nagyon fontos, hogy gyakori elemhalmaz minden részhalmaza is gyakori, hiszen azok legalább annyi tranzakcióban megtalálhatóak.

Tranzakciók tárolási módjai

A tranzakciókat és elemeket id-val látjuk el.

- *Horizontális adatbázis:* Minden tranzakcióhoz tároljuk az elemeinek listáját.
- *Vertikális adatbázis:* Minden elemhez tároljuk egy listában a tranzakciókat, melyek őt tartalmazzák.
- *Relációs adatbázis:* A tábla minden sora egyetlen elem - tranzakció párt tartalmaz.

Tartalom

Tartalom

Gyakori elemhalmazok egy speciális esetben

A következő sql lekérdezéssel a kételemű gyakori elemhalmazok kereshetőek meg, amennyiben azokat relációs adatbázisban tároljuk.

```
SELECT I.elem, J.elem, COUNT(I.tranzakció)
    FROM tranzakciók I,tranzakciók J
WHERE I.tranzakció = J.tranzakció AND I.elem < J.elem
    GROUP BY I.elem, J.elem
    HAVING COUNT(I.tranzakció) >= min_supp
```

Az általános esetet megoldó algoritmusok

Egy algoritmust *teljesnek* nevezünk, ha minden gyakori elemhalmazt megtatlál. *Helyesnek* akkor nevezzük, ha csak a gyakori elemhalmazokat találja meg. Ilyen értelemben az előző sql lekérdezés se nem helyes se nem teljes.
Az gyakori elemhalmazokat kinyerő algoritmusok(GYEK) általában 3 lépést ismételgetnek.

- Jelölteket állítanak elő.
- Meghatározzák a jelöltek támogatottságát.
- Kiválogatják a gyakoriakat.

A különbség a keresési tér bejárásában (a jelöltek előállításában) és a támogatottság meghatározásában van.

Tartalom

Az Apriori algoritmus

$l = 0$
$J_l = \{0\}$
`while` $|J_l| \neq 0$ `do`

$$\texttt{támogatottság\_meghatározás}(T, J_l)$$
$$Gy_l = \texttt{gyakoriak\_kiválogatása}(J_l, min\_supp)$$
$$J_{l+1} = \texttt{jelölt\_eloállítás}(Gy_l)$$
$$l = l + 1$$

`end while`
`return GY`
Egy szélességi bejárást valósít meg a lehetséges elemhalmazokat reprezentáló fában, hiszen minden iterációban próbál egyre nagyobb méretű gyakori elemhalmazokat keresni.

Jelöltek előállítása

Két n elemű halmazból(Legyenek ezek: l_1, l_2. Ezeket *genenrátoroknak* hívjuk) egy $n+1$ méretű lesz előállítva. A következő tulajdonságok figylemebevételével:

- l_1 a deiniált rendezésnek megfelelően megeleőzi a l_2-t
- l_1 és l_2 $l-1$ elemű prefixei megegyeznek.

Az így kapott $n+1$ elemű halmazról kell eldönteni, hogy gyakori-e. Ennek szükséges feltétele, hogy mind az $n+1$ db n elemű részhalmaza gyakori legyen. Ezt le kell ellenőrizni mielőtt gyakorinak jelöljük.

A jelöltek támogatottságának meghatározása

A jelöltek elemszáma alapján különböző eljárásokat kell használni:

- **1 elem esetén:** minden elemhez hozzárendelünk egy int változót kezdetben 0 értékkel. A tranzakciókat végigolvassuk és minden eleme esetén növeljük az ahhoz tartozó számlálót.
- **2 elem esetén:** Tfh. n gyakori elemünk van elemünk van. Ekkor érdemes felvenni egy $n * n$ méretű kezdetben csupa 0 elemet tartalmazó tömböt, melynek csak a DNY-ÉK irányú átló felett levő elemeit vesszük figyelembe. Ekkor az $< l, k >$ $(l < k)$ párhoz tartozó értéket a tömb [l][k-l] eleme tartalmazza. Az egyes tranzakciók végigolvasása folyamán minden előállítható párra meg kell növelni a tömb megfelelő elemét.

A jelöltek támogatottságának meghatározása

- **2-nél több elem esetén:** Érdemes a jelölteket tartalmazó fát egy *szófává* alakítani. A szófa élei a megfelelő elemek. A pontjai pedig az egyedhalmazoknak felelnek meg. A gyökér az üres halmaznak. A többi elem, pedig annak a rendezett halmaznak, amely elemek azokon az éleken vannak, melyeken át az ponthoz el lehet jutni a gyökérből. Egy csomópont egyes gyerekei balról jobbra oly sorrendben vannak, ahogy a csomópontot bővítő elem lexiografikus rendezése adja vagy ennek fordítottja vagy pedig valamilyen más szempont szerint van rendezve (lsd. részletetesebben a későbbiekben).
  Az algoritmus:
  A tranzakciókon belül az elemek is a lexiogafikus rendezésnek megfelelően vannak. Egy tranzakciónál végig kell lépkedni a fában oly módon, hogy minden olyan jelölt támogatottságát növeljük, melyet a tranzakció tartalmaz. Ha a tranzakció hossza t, akkor $\frac{t^2}{2}$ pontot kell bejárni a fában.

Apriori szófával

A szófát nem csak a támogatottság meghatározásához használhatjuk fel, hanem a jelölt előállításnál is. Ugyanis a szófa olyan levelei, melyek testvérek is megfelelnek a jelölt előállítás során említett genenrátoroknak. Az új jelöltet csak a bal oldalibb alá kell felvenni. Így elkerülve, hogy az algoritmusban egy elem kétszer is megjelenjen.
Tehát az *algoritmus* a futása során egyetlen szófát épít. A szófa kezdetben az üres halmazt jelentő jelentő egyetlen csúcsból áll, melynek gyerekei a gyakori elemhalmazok. Ezek gyerekei a kételemű gyakoriak. Ezek után érdemes a 3 elemű jelölteket az előbb említett módon előállítani. Tehát a fa levelei a jelölteknek felelnek meg. Ezek támogatottságát meg kell határozni. A ritka halmazoknak megfelelő leveleket (vagyis amelyek támogatottsága nem éri el a*min_supp*-ot) le kell törölni...

A rendezés hatása a szófa méretére

Törekedni kell a minél kisebb fára, mert rengeteg (akár exponenciálisan sok) gyakori elemhalmaz is lehet. A minimális szófa megtalálása neház feladat.

Tétel

A minimális szófa megkeresése NP-nehéz probléma.

A tapasztalatok szerint a gyakoriság szerint csökkenő sorrend adja a legkedvezőbbe méretet. De mégsem ezt érdemes használni, hanem a gyakoriság szerinti növekvő sorrendet. Ennek az előnyei:

- Egy csomópontból kevesebb él indul ki.
- Valamint a gyökér közelében vannak a ritka elemek.

A fenti tulajdonságok azért előnyösek, mert így a tranzakciók nem jutnak olyan mélyre a fában és nem érintenek annyi levelet.

További optimalizációk

- **Ritka jelöltek törlése:** A ritka jelöltek törléséhez nem kell még egyszer bejárni a szófát, hanem az új jelöltek előállításánál is ráérünk megtenni.
- **Zsákutca nyesés:** Felesleges tárolni azokat a csúcsokat, melyek egyik gyereke sem gyakori egyedhalmaz, hiszen ezek csak a memóriát foglalják. Persze vigyázni kell, nehogy túl korán töröljük. Addig nem sznad, ameddig nem állt elő az összes olyan jelölt, aminek részhalmaza lehet.

Tranzakciók szűrése

Fontos, hogy a tranzakciókból szűrjük a nem releváns elemeket vagyis azokat, amik már nem lehetnek elemei az aktuális méretű jelölteknek. Hiszen ez lassítja a támogatottság meghatározást.
Az ötletek:

- Minden tranzakcióból töröljük a ritka elemeket.
- Az algoritmus l. iterációjában (ekkor a jelöltek l méretűek) törölni kell azokat a t tranzakciókat, melyek elemszáma nem nagyobb, mint l. Hiszen ekkor ez a tranzakció már nem tartalmaz olyan elemet, ami a következő iterációban jelölt lesz.
- Töröljük azt a tranzakciót, mely nem tartalmaz jelöltet. Ez úgysem adhat már nagyobb méretű gyakori elemhalmazt, hiszen, akkor kellett volna jelöltet tartalmaznia.

Tranzakciók szűrése (folyt.)

- Töröljük a tranzakció azon elemeit, melyek nem elemei egyetlen olyan jelöltnek sem, amit a tranzakció tartalmaz. Ugyanis, ezek az elemek nagyobb méretű gyakori elemhalmazoknak sem lehetnek elemei. Ha így a tranzakció számossága kisebb egyenlő l-lel, akkor töröljük a tranzakciót is.
- Töröljük a tranzakció azon elemeit, melyek nem elemei legalább l db olyan jelöltnek, amit tartalmaz a tranzakció. Hiszen ekkor ezek az elemek megintcsak nem lehetnek $l + 1$ méretű gyakori elemhalmazok elemei. Az elemek eltávolítása után használhatjuk a második ötletet.

Tartalom

Az Eclat algoritmus

Az Eclat algoritmus az üres mintából kiindulva egy mélységi keresést hajt végre, ellentétben az Apriori szélességi keresésével. Mindig egyetlen jelöltet állít elő, melynek meg is határozza a támogatottságát. Egy l számosságú jelölt előállítása során kettő $l-1$ számosságút használ fel, hasonlóan az Apriorihoz. A támogatottság meghatározása azonban máshogy történik. Ehhez be kell vezetni az ún. *TID-halmazokat*. A TID-halmazt elemek egy halmazához rendeljük hozzá. A TID-halmaz megadja azokat a tranzakció azonosítókat, amelyekben az adott halmaz minden eleme megtalálható. Így a jelölt támogatottsága generátorainak TID-halmazainak metszetének számosságával egyezik meg.

Tartalom

FP-Growth algoritmus

A keresési tér bejárásá megegyezik az Eclat algoritmuséval. De a támogatottságok meghatározása eltér. Ehhez az algoritmus felhasználja az elemhalmazok vetítését. A T tranzakcióhalmaz P elemhalmazra való vetítését(T | P) úgy kapjuk, hogy a vesszük T P-t tartalmazó elemeit, és töröljük belőlük P-t. Az algoritmus egy rekurziós lépésében három dolgot ismétel. A gyakori elemek megkeresése után előállítja azokból a szűrt tranzakciókat. Majd a szűrt tranzakciókat vetíti az egyes gyakori elemekre és mindegyik így előállt szűrt tranzakcióra új rekurzív hívást indít. A rekurzív hívásnál átadja az eddigi gyakori elemeket is így bővíti a gyakori elemhalmazokat.

Tartalom

A teljesítmények elemzése

2004-ben rendeztek egy versenyt az GYEK algoritmusok összehasolítására.
Sebesség tekintetében az Eclat és FP-growth módosításai vitték a pálmát.
Memórihasználatban az Apriori volt a nyerő.