Alkalmazott Matematikai Lapok **37** (2020), 71–85.
DOI: 10.37070/AML.2020.37.1.03

EGY GYAKORI ZÁRT MINTÁKAT ÉS GYAKORI GENERÁTOROKAT KERESŐ VERTIKÁLIS ALGORITMUS

SZATHMÁRY LÁSZLÓ

A gyakori minták feltárására az *Apriori* a legismertebb algoritmus. Azonban az érdekes asszociációs szabályok előállításához kellenek még a gyakori zárt minták (GyZM-k) és a gyakori generátorok is (GyG-ok), melyek a gyakori minták részhalmazai. A *Zart* egy *Apriori*-szerű algoritmus, mely képes kiszűrni a gyakori minták közül a GyZM-kat és a GyG-okat. Viszont ismeretes, hogy a vertikális mintakereső algoritmusok általában jobban teljesítenek, mint az *Apriori*-szerű szintenkénti algoritmusok. Az *Eclat* egy másik jól ismert vertikális mintakereső, mely ugyanazt a kimenetet állítja elő, mint az *Apriori*, vagyis egy adathalmazban megkeresi az összes gyakori mintát. Ebben a cikkben az *Eclat* egy kiterjesztését, az ún. *Eclog* algoritmust mutatjuk be, amely képes kiszűrni a gyakori minták közül a GyZM-kat és a GyG-okat. Az algoritmusunk egy menetben (utófeldolgozás nélkül) tárja fel egy adathalmaz ekvivalencia-osztályait. A teszteredmények azt mutatják, hogy az *Eclog* nagyon jól teljesít, különösen a sűrű, erősen korrelált adathalmazok esetén.

1. Bevezetés

Az adatbányászatban a gyakori minták kinyerése és az asszociációs szabályok feltárása fontos szerepet játszik [1]. A gyakori minták nagy száma miatt számos tömörített reprezentációt ajánlottak, melyek közül a gyakori generátorok (GyG-ok) és a gyakori zárt minták (GyZM-k) a legismertebbek [2, 3, 4]. A szakirodalomban több módszer is létezik a GyG-ok és a GyZM-k egyidejű kinyerésére, de ezen algoritmusok többsége szintenkénti megközelítést alkalmaz [5, 6]. Viszont a tapasztalat azt mutatja, hogy a mélységi elven működő algoritmusok általában jobban teljesítenek, mint a szintenkénti megfelelőik. Ebben a cikkben egy egylépcsős (utófeldolgozást nem igénylő), mélységi elven működő, vertikális algoritmust mutatunk be a GyG-ok és a GyZM-k együttes kinyerésére. Az algoritmusunk kimenetéből triviális módon lehet érdekes asszociációs szabályokat, pl. minimális nem redundáns szabályokat generálni. A minimális nem redundáns szabályokról és azok előállításáról egy korábbi cikkünkben írtunk [7]. Jelen cikkünk egy korábban angolul megjelent közleményünk [8] kibővített változata.

Ezen cikk a következőképpen épül fel. A 2. részben áttekintjük a mintakereséssel kapcsolatos alapvető fogalmakat és definíciókat. A 3. részben az *Eclog* algoritmust mutatjuk be részletesen. A pszeudokód után egy konkrét példán keresztül szemléltetjük az algoritmus működését. A 4. részben az algoritmus futási idejét mérjük össze három másik módszerrel. Végül az 5. részben a jövőbeli tervek következnek, majd a konklúziókkal zárjuk a cikket.

2. Fogalmak és definíciók

Vegyük a következő 4×6-os méretű adathalmazt: $\mathcal{D} = \{(1,\ ACDE),\ (2,\ ABCDE),\ (3,\ ABE),\ (4,\ BEF)\}$. A cikk további részében erre a példára $\boldsymbol{\mathcal{D}}$ adathalmaz néven fogunk hivatkozni.

Legyen $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ objektumok (vagy tranzakciók) egy halmaza, $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ attribútumoknak egy halmaza, $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ pedig egy bináris reláció. Az $\mathcal{A}$ halmaz egy tetszőleges részhalmazát *mintának* nevezzük. Minden tranzakció rendelkezik egy egyedi azonosítóval (*tid*), s a tranzakció-azonosítók egy tetszőleges halmazát *tidset*nek hívjuk. Ha egy tidset valamennyi tranzakciója tartalmaz egy X mintát, akkor a tidsetet az X minta *képének* is nevezzük, s $t(X)$-szel jelöljük. Pl. a $\mathcal{D}$ adathalmazban az $\{A, B\}$ minta képe a $\{2, 3\}$, azaz egyszerűsített jelölést alkalmazva $t(AB) = 23$. Egy minta *hossza* alatt a minta kardinalitását értjük, egy k darab attribútumot tartalmazó mintát pedig k-hosszúságú mintának nevezünk. Egy X minta (abszolút) *support* értéke – jelölése $supp(X)$ – az X minta képének a mérete, azaz $supp(X) = |t(X)|$. Vagyis a support azt mutatja meg, hogy egy minta hány tranzakcióban van jelen. Egy X minta *gyakori*, ha a support értéke nem kisebb, mint egy adott *minimum support* küszöbérték (jelölése min_supp), azaz $supp(X) \geq min\_supp$. Két minta $X, Z \subseteq \mathcal{A}$ ekvivalens ($X \cong Z$), ha a képük megegyezik, vagyis $t(X) = t(Z)$. Egy P mintával ekvivalens minták halmazát a P minta ekvivalenciaosztályának nevezzük: $[P] = \{Q \subseteq \mathcal{A} \mid P \cong Q\}$. Egy ekvivalenciaosztálynak egyetlen maximális eleme van (zárt minta), s több minimális eleme is lehet (generátorok[1]) [9]. A generátorok a szabad minták egy speciális esetét képviselik [10].

2.1. Definíció. Egy minta *zárt*, ha nincs vele azonos support értékű valódi szuperhalmaza.

2.2. Definíció. Egy minta (minimális) *generátor*, ha nincs vele azonos support értékű valódi részhalmaza.

[1] A szakirodalomban ezen mintákat többféleképpen is nevezik: kulcsminták, minimális generátorok, szabad minták, kulcsgenerátorok stb.

A *lezárási* operátor egy X mintához az X ekvivalenciaosztályában található maximális elemet rendeli hozzá. A hozzárendelt elemet X *lezártjának* nevezzük, s $\gamma(X)$-szel jelöljük. Természetesen egy zárt X esetén $X = \gamma(X)$.

2.1. Példa. A $\mathcal{D}$ adathalmazban B és C generátorok, melyek lezártjai: $\gamma(B) = BE$, és $\gamma(C) = ACDE$ (lásd még 1. ábra).

2.1. Tulajdonság. Legyen $X \subseteq \mathcal{A}$. Ha X generátor, akkor $\forall Y \subseteq X$, Y is generátor. Hasonlóképp, ha X nem generátor, akkor $\forall Z \supseteq X$, Z sem generátor.

A GyZM-k és a GyG-ok jól ismert tömör reprezentációi [11] a gyakori mintáknak, s együtt alkotják az érvényes asszociációs szabályok nem redundáns bázisait (pl. a *generikus bázist*) [3].

3. Generátorok és zárt minták kiszűrése a gyakori minták közül mélységi bejárás alkalmazásával

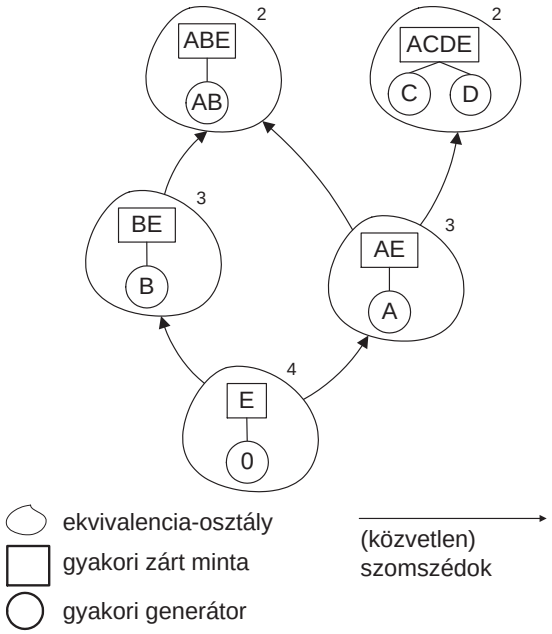
Az *Eclog* algoritmusunk, mely jelen cikkünk központi témája, egy vertikális mintakereső algoritmus, mellyel fel lehet fedezni egy adathalmazban a gyakori ekvivalenciaosztályokat. Ebben a fejezetben először egy áttekintést nyújtunk az algoritmusról. Ezt követően megvizsgálunk néhány vertikális elven működő algoritmust, név szerint az *Eclat*ot [12] és a *Talky*t [13]. Végül részletesen is bemutatjuk az *Eclog* algoritmust.

3.1. Az Eclog általános áttekintése

Az *Eclog* a *Talky* [13] algoritmusra épül, míg a *Talky* az *Eclat* [12] egy módosított változata. Az *Eclat* és a *Talky* ugyanazt a kimenetet produkálják, vagyis egy adathalmaz összes gyakori mintáját keresik meg. Azonban a *Talky* egy eltérő bejárási stratégiát alkalmaz, az ún. fordított preorder módszert. Ez a bejárás jobbról balra halad, s rendelkezik egy speciális tulajdonsággal: amikor elérünk egy X mintához, addigra már felfedeztük X valamennyi részhalmazát. Ennek eredményeképpen ez a bejárás felhasználható a GyG-ok hatékony kiszűrésére. A bejárás során az *Eclog* a GyZM-kat is kiszűri, és hozzárendeli őket a megfelelő GyG-okhoz. Így mire az *Eclog* befejezi a futását, addigra feltárta az összes gyakori ekvivalenciaosztályt. A GyG-ok hatékony módon való kiszűrésére az algoritmus a fordított preorder stratégiára támaszkodik, melyet a következő alfejezetekben mutatunk be.

3.2. Vertikális mintakeresés

A mintakereső algoritmusok – akár az összes gyakori mintát keresik vagy csak a GyZM-kat – általában két osztályba sorolhatók: vannak a szélességi



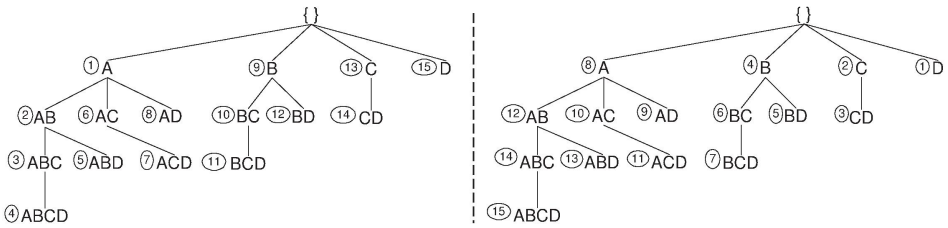
1. ábra. A $\mathcal{D}$ adathalmaz ekvivalenciaosztályai $min\_supp = 2$ küszöbérték mellett. A support értékek az osztályok jobb felső sarkában láthatók. Az E minta generátora az üres halmaz.

keresők, ill. vannak a mélységi keresők. A szélességi elven működő algoritmusok, egész pontosan az *Apriori*-szerű algoritmusok [1], a keresési teret szintenként járják be, kihasználva a minták gyakoriságának antimonotonitását[2]. A mélységi algoritmusok – pl. *Closet* [14] – ezzel szemben a keresési teret egy prefixfába szervezik (lásd 2. ábra). Ezen algoritmusok közül a *vertikális* elven működők az adathalmazt (minta, tidset) párokként reprezentálják ($\{(i, t(i))|i \in \mathcal{A}\}$), mellyel el tudják kerülni az adathalmaz költséges többszöri végigolvasását.

Az *Eclat* [12] volt az első olyan gyakori mintákat kereső algoritmus, amely sikeresen kombinálta a mélységi keresést és az adathalmaz vertikális reprezentációját, s ezt egy fa adatszerkezettel, egy ún. IT-fával oldotta meg, melyben a csúcsok $X \times t(X)$ párok.[3] Az *Eclat* az IT-fát mélységi módon, preorder stratégiával, balról jobbra haladva járja be [12, 15] (lásd 2. ábra).

[2]Az antimonotonitás elve szerint ha egy minta nem gyakori, akkor annak egyetlen szuperhalmaza sem lehet gyakori [1].

[3]Az IT-fa esetén az IT az **i**temset-**t**idset rövidítése, ami arra utal, hogy egy csúcsban egy minta és a hozzá tartozó tidset helyezkedik el.



2. ábra. Bal oldal: az *Eclat* preorder bejárása; jobb oldal: a *Talky* fordított preorder bejárása. A bejárási sorrend a körökben van feltüntetve.

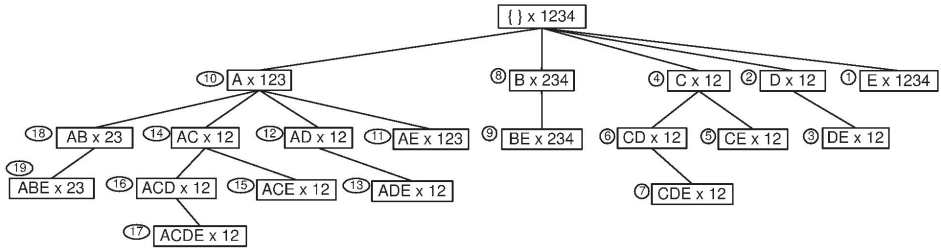
3.3. Fordított preorder bejárás

Az *Eclog* a *Talky* algoritmust terjeszti ki oly módon, hogy a gyakori minták közül kiszűri a GyG-okat és a GyZM-kat. Vagyis az *Eclog* megvizsgál minden újonnan talált gyakori mintát, hogy generátor-e. Ahhoz, hogy ezt a tesztet hatékonyan tudjuk elvégezni, egy X minta esetén már X összes részhalmazát érintenünk kellett. A korábban látott 2.1. tulajdonság alapján egy minta csak akkor lehet generátor, ha az összes részhalmaza is generátor. Egy X minta generátor státuszának megállapításához ezért szükségünk lesz az összes részhalmazának előzetes ismeretére. A szélességi keresést alkalmazó keresőalgoritmusok triviális módon biztosítják a minták helyes sorrendben való feltárását. Viszont a kívánt sorrendet egy mélységi keresésen alapuló algoritmussal is el lehet érni, bár ekkor egy speciális bejárásra van szükség.

A kombinatorikus algoritmusok esetén gyakori elvárás, hogy a keresési tér bejárása során egy adott X mintát az összes részhalmaza *után* dolgozzunk fel. A szintenkénti algoritmusok triviális módon elégítik ki ezt a feltételt. A *fordított preorder bejárást* [16] alapul véve, az adathalmaz attribútumait fordított lexikografikus sorrendbe tesszük (E, D, C stb.). Ha az attribútumok numerikus megfelelőit növekvő sorrendben követjük, akkor egy mélységi, jobbról balra haladó prefixfa bejárást kapunk, ahol a prefixfa a keresési teret reprezentálja. Mivel a csúcsokat a gyermekcsúcsok előtt dolgozzuk fel, ezért *pre*order (és nem *post*order) bejárásról beszélünk.

Összefoglalva: a módszerünk az IT-fát preorder módon járja be jobbról balra. Így, ha veszünk egy tetszőleges X mintát az IT-fában, a bejárás garantálni fogja, hogy az X csúcs *előtt* már az X összes részhalmazát tartalmazó csúcsot felfedeztük.

3.1. Példa. A 2. ábrán látható egy összehasonlítás a kétféle bejárásról. Bal oldalt az *Eclat* preorder bejárása, míg jobb oldalt a *Talky* fordított preorder bejárása szerepel.



3. ábra. A *Talky* algoritmus végrehajtása a $\mathcal{D}$ adathalmazon $min\_supp = 2$ küszöbérték mellett. A csúcsok feldolgozási sorrendje a körökben van feltüntetve.

3.4. A Talky algoritmus

A *Talky* egy vertikális, gyakori mintákat kereső algoritmus, amely a felépített IT-fát mélységi módon, fordított preorder bejárással dolgozza fel (lásd 3. ábra). A $\mathcal{D}$ adathalmazunkból $min\_supp = 2$ küszöbérték mellett a *Talky* a következő 19 gyakori mintát fedezi fel ebben a sorrendben[4]: E (4), D (2), DE (2), C (2), CE (2), CD (2), CDE (2), B (3), BE (3), A (3), AE (3), AD (2), ADE (2), AC (2), ACE (2), ACD (2), $ACDE$ (2), AB (2) és ABE (2).

3.5. Az Eclog részletes bemutatása

Ebben az alfejezetben az *Eclog* algoritmust mutatjuk be részletesen. Mint már korábban láttuk, az *Eclog* alapja a *Talky*. Az *Eclog* az IT-fát fordított preorder módon járja be (lásd 3. ábra), s a gyakori minták feltárása közben kiszűri a GyG-okat és a GyZM-kat. Az *Eclog* a generátorokat hozzákapcsolja a lezártjukhoz, így kimenetként a gyakori ekvivalenciaosztályok listáját állítja elő (lásd 1. táblázat).

Az *Eclog* egy hash táblázatot épít fel (lásd 1. táblázat). A hash táblázat kulcsa egy tidset, míg a kulcshoz tartozó érték egy sor objektum. A táblázat egy sora egy ekvivalenciaosztályt reprezentál, s a következő mezőkből épül fel: **(1)** tidset (definíció szerint egy ekvivalenciaosztály valamennyi elemének azonos a tidset-je), **(2)** generátorok (egy ekvivalenciaosztály minimális elemei), **(3)** ekvivalenciaosztály tagok (egy ekvivalenciaosztály azon elemei, melyek se nem generátorok, se nem zárt minták), **(4)** lezárt (egy ekvivalenciaosztály maximális eleme; ilyenből minden osztályban egy és csakis egy van), ill. **(5)** support (ez a tidset kardinalitása).

Az algoritmus a következőképpen működik. Amikor az algoritmus feltár egy új gyakori mintát az IT-fában, akkor megnézi, hogy a minta egy már korábban felfedezett ekvivalenciaosztályhoz tartozik-e, vagyis megnézzük, hogy a minta tidsetje szerepel-e a hash táblázatban. Ha nincs benne a hash táblázatban, akkor

[4]A zárójelben a support értékek szerepelnek.

1. táblázat. Az *Eclog* egy ilyen táblázatot épít fel, ami valójában egy hash táblázat, ahol a kulcs egy tidset, az érték pedig a táblázat egy sora.

tidset	generátorok	ekv. osztály tagok (opcionális)	lezárt	support
1234	$\emptyset$	E	E	4
12	D, C	DE, CE, CD, CDE, AD, ADE, AC, ACE, ACD, $ACDE$	$ACDE$	2
234	B	BE	BE	3
123	A	AE	AE	3
23	AB	ABE	ABE	2

egy új ekvivalenciaosztályba tartozik, s így a hash táblázatba egy új sort szúrunk be. Ha a tidsetje benne van a hash táblázatban, akkor két eset lehetséges. Jelöljük R-rel azt a sort, amelynek tidsetje megegyezik az aktuális minta tidset értékével, vagyis R azt az ekvivalenciaosztályt reprezentálja, amelybe az aktuális minta tartozik.

Első eset. Az aktuális mintának van egy valódi részhalmaza az R sor „generátorok" mezőjében. Ez azt jelenti, hogy a minta nem generátor, de ebbe az ekvivalenciaosztályba tartozik. Az R „lezárt" mezőjét frissíteni kell a mintával. Ez a frissítés a következőből áll: ha a minta nagyobb, mint az eddig regisztrált lezárt, akkor a lezártat lecseréljük az új mintára. Mint ismeretes, egy ekvivalenciaosztály lezártja az ekvivalenciaosztály legnagyobb eleme, ami egy egyedi elem (vagyis egy ekvivalenciaosztályban pontosan egy ilyen elem van). Mivel az *Eclog* megkeresi az összes gyakori mintát, ezért az algoritmus befejeződésekor a „lezárt" mezőben garantált módon minden sorban az adott ekvivalenciaosztály lezártja fog szerepelni. Opcionálisan egy ekvivalenciaosztály azon elemeit, amelyek nem generátorok, össze lehet gyűjteni az „ekv. osztály tagok" mezőben. A 1. táblázatban ezt a mezőt a könnyebb érthetőség kedvéért tüntettük fel, de ha csak a GyG-okra és a GyZM-kra van szükségünk, akkor az implementáció során ezt a mezőt üresen lehet hagyni.

Második eset. Az aktuális mintának nincs valódi részhalmaza az R sor „generátorok" mezőjében. Ez azt jelenti, hogy a minta az ekvivalenciaosztály egy új generátora, így hozzáadjuk a „generátorok" mezőhöz. Ha a generátor nagyobb, mint a sor lezártja, akkor a „lezárt" mezőt frissíteni kell (az előző esetben leírtak szerint).

Amikor az algoritmus befejezi a futását, a „lezárt" mezőben lévő minták mindegyike végleges, vagyis az ekvivalenciaosztályok tényleges lezártjait tartalmazzák. Az *Eclog* pszeudokódját az 1. algoritmus szemlélteti.

2. táblázat. Az *Eclog* hash táblázatának inicializálása.

tidset	generátorok	ekv. osztály tagok (opcionális)	lezárt	support
1234	$\emptyset$		$\emptyset$	4

3. táblázat. Az *Eclog* hash táblázatának állapota az E, D és DE minták beszúrása után.

tidset	generátorok	ekv. osztály tagok (opcionális)	lezárt	support
1234	$\emptyset$	E	E	4
12	D	DE	DE	2

3.2. Példa. A $\mathcal{D}$ adathalmazunk egy kissé speciális esetnek tekinthető, ui. az adathalmaz E oszlopa telített. Ez azt jelenti, hogy az E minta nem generátor, mivel van egy vele azonos support értékű részhalmaza, nevezetesen az üres halmaz. Definíció szerint az üres halmaz valamennyi sorban jelen van, így a support értéke 100%. Ezek alapján a hash táblázatot a 2. táblázatban látható módon inicializáljuk. A hash táblázat valójában bármilyen input adathalmaz esetén inicializálható így, de ha az adathalmazban nincs telített oszlop, akkor az üres halmaz lezártja végig az üres halmaz fog maradni.

Ezután az algoritmus elkezdi felsorolni a $\mathcal{D}$ adathalmaz 19 gyakori mintáját a *Talky* bejárási stratégiáját alkalmazva (az adathalmaz gyakori mintáit a 3.4-es alfejezetben tüntettük fel). Az első csúcs az $E \times 1234$. Az 1234 tidset már egy létező kulcs a hash táblázatban. Az E mintának van egy vele azonos support értékű részhalmaza (az üres halmaz), így az E mintát hozzáadjuk az „ekv. osztály tagok" oszlophoz, ill. frissítjük a „lezárt" mezőt az E mintával. A következő gyakori minta a $D \times 12$. Mivel a 12 még nincs benne a hash táblázatban, ezért a táblázatba egy új sort kell beszúrni. A következő csúcs a $DE \times 12$. Mivel az 12 tidset már szerepel a hash táblázatban, ezért a DE egy már felfedezett ekvivalenciaosztályhoz tartozik. Van neki egy valódi részhalmaza, a D, így a DE-t hozzáadjuk az „ekv. osztály tagok"-hoz. Mivel a DE nagyobb, mint a jelenleg regisztrált lezárt (D), ezért a „lezárt" mezőt frissítjük a DE mintával. A hash táblázat aktuális állapota a 3. táblázatban található.

Az algoritmus hátralévő részének részletes bemutatását kihagyjuk. Az *Eclog* kimenete – a végeredmény – az 1. táblázatban látható.

Algoritmus 1 (az *Eclog* pszeudokódja):

hashTable: az 1. táblázatban látható hash táblázat; kezdetben üres

```
1)  // inicializálás
2)  sor.tidset ← {legnagyobb tidset}   // a példában: 1234
3)  sor.generatorok ← ∅   // üres halmaz hozzáadása
4)  sor.lezart ← ∅
5)  sor.support ← |sor.tidset|   // kardinalitás
6)  hashTable.add(sor)
7)
8)  // fő blokk
9)  indítsuk el a Talky algoritmust s az aktuális csúcsot rendeljük hozzá
    az akt nevű változóhoz
10) {
11)     if akt.tidset nincs benne a hashTable-ben:
12)         sor.tidset ← akt.tidset
13)         sor.generatorok.add(akt.minta)
14)         sor.lezart ← akt.minta
15)         sor.support ← |sor.tidset|
16)         hashTable.add(sor)
17)     else:
18)         sor ← hashTable.get(akt.tidset)
19)         if akt.minta-nak van egy valódi részhalmaza a sor.generatorok-ban:
20)             sor.ekv_oszt_tagok.add(akt.minta)   // opcionális
21)             if |akt.minta| > |sor.lezart|:
22)                 sor.lezart ← akt.minta
23)             endif
24)         else:
25)             sor.generatorok.add(akt.minta)
26)             if |akt.minta| > |sor.lezart|:
27)                 sor.lezart ← akt.minta
28)             endif
29)         endif
30)     endif
31) }
```

4. Teszteredmények

Tesztjeink során az *Eclog* algoritmust az *Eclat*, *Zart* és *LCM+GrGrowth* algoritmusokkal mértük össze. Az *Eclog*, *Eclat* és *Zart* algoritmusokat Javaban implementáltuk a CORON adatbányász platformban [17].[5] Mivel az *Eclog* alapja az *Eclat*, ezért az összehasonlításba betettük az *Eclat*ot is, ui. kíváncsiak voltunk, hogy az *Eclog* extra műveletei okoznak-e érezhető teljesítménycsökkenést. A *Zart* [7] egy szintenkénti algoritmus, mely a *Pascal*-algoritmus [9] kiterjesztése. A *Zart*, köszönhetően az ún. mintaszámláló következtetésének, nagyon hatékonyan tudja megkeresni a gyakori ekvivalenciaosztályokat. Az *LCM* az egyik leggyorsabb GyZM-kat kereső algoritmus [18], míg a *GrGrowth* az egyik leghatékonyabb GyG-okat előállító algoritmus [19]. Az *Eclog*gal való tisztességes összehasonlítás kedvéért összefogtuk a két algoritmust, s az eredményre *LCM+GrGrowth* néven hivatkozunk. Ez a kombinált algoritmus három lépést valósít meg: kigyűjti a GyZM-kat az *LCM*-mel; kigyűjti a GyG-okat a *GrGrowth*-szal; majd végül egy utófeldolgozás során összepárosítja a GyG-okat és a GyZM-kat, így a kimenet a gyakori ekvivalenciaosztályok listája lesz. Az *LCM* és a *GrGrowth* algoritmusok esetén a szerzők eredeti forráskódját használtuk fel. Az *LCM* C nyelven, míg a *GrGrowth* C++ nyelven lett implementálva.

A teszteket egy Intel Quad Core Xeon 2,33 GHz-es gépen végeztük el Ubuntu GNU/Linux operációs rendszer alatt. Az adott gép 8 GB RAM-mal rendelkezett. Valamennyi válaszidő valós idő, melyet a Unix rendszereken használt *time* paranccsal mértünk le a bemenet és a kimenet között. A tesztekhez a következő adathalmazokat használtuk fel: T20I6D100K, C20D10K, C73D10K és MUSHROOMS. A T20[6] egy ritka adathalmaz, s felépítésében a bevásárlóközpontok adatbázisaira hasonlít, melyek általában gyengén korrelált adatokat tartalmaznak. A C20 és C73 egy népszámlálás részadatait tartalmazza, míg a MUSHROOMS[7] különböző gombák jellemzőit írja le. Ez utóbbi három adathalmaz sűrű és erősen korrelált.

Az említett négy algoritmus futási idejét a 4. táblázatban foglaltuk össze. A táblázatban szintén megtalálható a gyakori minták, a gyakori generátorok, ill. a gyakori zárt minták száma. Az *Eclat* megkeresi, s a kimenetében megjeleníti az összes gyakori mintát. Az *Eclog* és a *Zart* ugyan bejárja a keresési térben az összes gyakori mintát, de a kimenetükben már csak a GyG/GyZM párokat jelenítik meg. Az *LCM+GrGrowth* lecsökkenti a keresési teret a GyZM-kra és a GyG-okra, és az *Eclog*hoz és a *Zart*hoz hasonlóan csak a GyG/GyZM párokat jeleníti meg a kimenetében.

[5] http://coron.loria.fr
[6] http://www.almaden.ibm.com/software/quest/Resources/
[7] http://kdd.ics.uci.edu/

4. táblázat. Az *Eclog* futási ideje, ill. egyéb statisztikák (gyakori minták (GyM-k) száma, gyakori generátorok (GyG-ok) száma, gyakori zárt minták (GyZM-k) száma).

min_supp	futási idő (mp.)				# GyM	# GyG	# GyZM
	Eclog	Eclat	Zart	LCM+ GrGrowth			
T20I6D100K							
10%	0,97	0,97	0,80	0,36	7	7	7
0.75%	1,26	1,24	9,64	1,64	4710	4710	4710
0.50%	1,79	1,75	16,83	2,32	26 836	26 305	26 208
0.25%	5,25	4,43	41,82	7,31	155 163	149 447	149 217
C20D10K							
30%	0,49	0,57	2,13	0,29	5319	967	951
20%	0,63	0,74	3,85	0,44	20 239	2671	2519
10%	0,83	1,10	7,72	0,67	89 883	9331	8777
5%	1,14	2,49	14,64	0,87	352 611	23 051	21 213
2%	2,63	9,33	45,53	1,41	1 741 883	57 659	50 729
C73D10K							
95%	0,68	0,69	1,84	0,24	1007	121	93
90%	0,88	0,99	13,99	0,39	13 463	1368	942
85%	0,94	1,24	35,56	0,49	46 575	3513	2359
Mushrooms							
40%	0,36	0,38	0,72	0,17	505	153	124
30%	0,42	0,46	1,22	0,22	2587	544	425
15%	0,65	1,07	3,83	0,43	99 079	3084	2210
10%	1,23	3,33	13,47	0,56	600 817	7585	4850

A T20 szintetikus adathalmaz a bevásárlóközpontok adatbázisait utánozza, így egy ritka és gyengén korrelált adathalmazról van szó. Ezen adathalmazban kevés a gyakori minta, s majdnem valamennyi gyakori minta GyZM és GyG egyszerre, ami azt jelenti, hogy a legtöbb ekvivalenciaosztály egyelemű. Az *Eclog* és *Eclat* hasonló módon viselkedik, bár az *Eclog* extra műveletei nagyon kis mértékben lassítanak az algoritmuson. Az *Eclog* sokkal jobban teljesít a *Zart*nál, s ez a különbség különösen akkor látványos, amikor a *min_supp* érték nagyon alacsonyra van állítva. Mivel az *LCM+GrGrowth* algoritmusnak főleg egyelemű ekvivalenciaosztályokkal kell dolgoznia, ezért körülbelül ugyanazt a keresési teret kell bejárnia, mint az *Eclog*nak. A tesztek alapján az *LCM+GrGrowth* egy kicsit lassabban teljesít, mint az *Eclog*.

A C20, C73 és Mushrooms adathalmazokban a GyG-ok és a GyZM-k száma sokkal kevesebb, mint az összes gyakori minta száma. Az *Eclog* és *Eclat* hasonlóan teljesít magas *min_supp* érték esetén. Azonban ahogy csökkentjük a küszöbértéket, úgy nő a két algoritmus futási ideje közti különbség. Ez az *Eclat* nagyobb méretű kimenetével magyarázható. Például a C20 adathalmazon $min\_supp = 2\%$ esetén

az *Eclat* 1 741 883 gyakori mintát produkál kimenetként, míg az *Eclog* csupán 57 659 GyG-t és 50 729 GyZM-t. Habár a két algoritmus ugyanazt a keresési teret járja be, ill. az *Eclog* még extra szűréseket is végez, az *Eclat* esetén alacsony *min_supp* mellett nagyon megnő az I/O műveletek száma, ami negatív hatással van az algoritmus teljesítményére. Az *Eclog* és *Zart* esetén a különbség még a korábbi T20-nál is szembetűnőbb. Amint látható, az *LCM+GrGrowth* sűrű adathalmazok esetén egy kicsit jobban teljesít az *Eclog*nál. Ez azzal a ténnyel magyarázható, hogy az *LCM+GrGrowth*-nak sokkal kisebb keresési teret kell bejárnia.

Összefoglalásképpen azt mondhatjuk, hogy ritka adathalmazok esetén az *Eclog* extra jellemzői nem okoznak szinte semmiféle teljesítménycsökkenést az *Eclat*hoz képest, míg sűrű adathalmazok esetén az *Eclog* jobb teljesítményt nyújt az *Eclat*nál, köszönhetően a kisebb kimenetének. Az *Eclog* minden esetben felülmúlja a *Zart* teljesítményét, ami összhangban áll azzal az általános nézettel, mely szerint a vertikális algoritmusok általában hatékonyabbak, mint a szintenkénti algoritmusok. Az *Eclog*ot szintén összehasonlítottuk az egyik leghatékonyabb algoritmussal, az *LCM+GrGrowth*-szal. A ritka adathalmazok esetén az *Eclog* volt gyorsabb, míg a sűrű adathalmazoknál az *LCM+GrGrowth* egy kicsit hatékonyabbnak bizonyult, de az *Eclog* és az *LCM+GrGrowth* közti különbség nem túl jelentős. Az *LCM+GrGrowth* kicsit jobb teljesítménye a következőkkel magyarázható: **(a)** az *LCM* és a *GrGrowth* a mintakereső algoritmusok között a leghatékonyabb megoldások közé tartoznak, **(b)** C/C++ -ban lettek implementálva, ill. **(c)** sűrű adathalmazok esetén egy sokkal kisebb keresési teret kell feltárniuk. Mindent egybevetve úgy érezzük, hogy az *Eclog*nak nincs oka szégyenkeznie az *LCM+GrGrowth* mellett.

5. Konklúzió és jövőbeli tervek

Ebben a cikkben egy vertikális, mélységi elven működő algoritmust mutattunk be, mely kimenetként a GyG/GyZM párokat állítja elő, vagyis egy adathalmazban a gyakori ekvivalenciaosztályokat tárja fel. Az *Eclog*ot az első lépésnek szántuk egy egylépcsős, vertikális elven működő, GyG-okat + GyZM-kat kereső algoritmus tervezéséhez vezető úton. Ehhez egy ismert, gyakori mintákat kereső algoritmust vettünk alapul, amit kiegészítettünk a GyZM-k és a GyG-ok szűrésének a képességével, s azt is sikerült elérnünk, hogy a GyG-ok menet közben rendelődnek hozzá a megfelelő GyZM-khoz, így nincs szükség utófeldolgozásra. Az *Eclog* algoritmus a jövőben könnyen felkészíthető asszociációs szabályok bányászatára is, hiszen a jelenlegi kimenetéből már most is triviális módon előállítható például a generikus bázis [3]. Ha az *Eclog* kimenetét kombináljuk egy olyan algoritmussal, amely képes megállapítani az ekvivalenciaosztályok közti közvetlen és tranzitív kapcsolatokat, akkor egy komplett megoldást kaphatunk az ún. minimális nem redundáns szabályok előállításához [3].

A közeljövőben olyan stratégiákat szeretnénk megvizsgálni, melyekkel csökkenteni lehetne az *Eclog* keresési terét. Ezzel függ össze az az érdekes kérdés,

hogy hogyan lehetne úgy meghatározni egy ekvivalenciaosztály lezártját, hogy nem állítjuk elő az adott ekvivalenciaosztály összes elemét. Egy lehetséges út lehetne a mintaszámláló következtetés [9] alkalmazása, míg egy másik lehetőség az lenne, ha egy ekvivalenciaosztályban csak a kanonikus GyG-okat keresnénk meg, s ezekből állítanánk elő az ekvivalenciaosztály lezártját.

Köszönetnyilvánítás

A publikáció elkészítését az EFOP-3.6.1-16-2016-00022. számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

Hivatkozások

[1] Agrawal, R. and Srikant, R.: *Fast Algorithms for Mining Association Rules in Large Databases*, In: Proc. of the 20th Intl. Conf. on Very Large Data Bases (VLDB '94), San Francisco, CA, Morgan Kaufmann, pp. 487-499 (1994).

[2] Bastide, Y., Taouil, R., Pasquier, N., Stumme, G. and Lakhal, L.: *Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets*, In: Proc. of the Computational Logic (CL '00). Volume 1861 of LNAI., Springer, pp. 972-986 (2000). DOI: 10.1007/3-540-44957-4_65

[3] Kryszkiewicz, M.: *Concise Representations of Association Rules*, In: Proc. of the ESF Exploratory Workshop on Pattern Detection and Discovery, pp. 92-109 (2002). DOI: 10.1007/3-540-45728-3_8

[4] Aggarwal, C.C. and Han, J., eds.: *Frequent Pattern Mining*, Springer International Publishing (2014). DOI: 10.1007/978-3-319-07821-2

[5] Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L.: *Discovering Frequent Closed Itemsets for Association Rules*, In: Proc. of the 7th Intl. Conf. on Database Theory (ICDT '99), Jerusalem, Israel, pp. 398-416 (1999). DOI: 10.1007/3-540-49257-7_25

[6] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N. and Lakhal, L.: *Computing Iceberg Concept Lattices with Titanic*, Data and Knowl. Eng., Vol. **42** No. **2**, pp. 189-222 (2002). DOI: 10.1016/S0169-023X(02)00057-5

[7] Szathmary, L.: *ZART: egy multifunkcionális mintakereső algoritmus*, Alkalmazott Matematikai Lapok, Vol. **27** No. **2**, pp. 107-122 (2010).

[8] Szathmary, L.: *Finding frequent closed itemsets with an extended version of the Eclat algorithm.* Annales Mathematicae et Informaticae, Vol. **48**, pp. 75-82 (2018).

[9] Bastide, Y., Taouil, R., Pasquier, N., Stumme, G. and Lakhal, L.: *Mining frequent patterns with counting inference*, SIGKDD Explor. Newsl, Vol. **2** No. **2**, pp. 66-75 (2000). DOI: 10.1145/380995.381017

[10] Boulicaut, J.F., Bykowski, A. and Rigotti, C.: *Free-Sets: A Condensed Representation of Boolean Data for the Approximation of Frequency Queries*, Data Mining and Knowledge Discovery, Vol. **7** No. **1**, pp. 5-22 (2003). DOI: 10.1023/A:1021571501451

[11] Calders, T., Rigotti, C. and Boulicaut, J.F.: *A Survey on Condensed Representations for Frequent Sets*, In Boulicaut, J.F., Raedt, L.D., Mannila, H., eds.: Constraint-Based Mining and Inductive Databases, Volume 3848 of Lecture Notes in Computer Science., Springer, pp. 64-80 (2004). DOI: 10.1007/11615576_4

[12] Zaki, M.J., Parthasarathy, S., Ogihara, M. and Li, W.: *New Algorithms for Fast Discovery of Association Rules*, In: Proc. of the 3rd Intl. Conf. on Knowledge Discovery in Databases, pp. 283-286 (1997). DOI: 10.1007/978-1-4615-5669-5_1

[13] Szathmary, L., Valtchev, P., Napoli, A. and Godin, R.: *Efficient Vertical Mining of Frequent Closures and Generators*, In: Proc. of the 8th Intl. Symposium on Intelligent Data Analysis (IDA '09). Volume 5772 of LNCS., Lyon, France, Springer, pp. 393-404 (2009). DOI: 10.1007/978-3-642-03915-7_34

[14] Pei, J., Han, J. and Mao, R.: *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21-30 (2000).

[15] Zaki, M.J.: *Scalable Algorithms for Association Mining*, IEEE Transactions on Knowledge and Data Engineering, Vol. **12** No. **3**, pp. 372-390 (2000). DOI: 10.1109/69.846291

[16] Calders, T. and Goethals, B.: *Depth-first non-derivable itemset mining*, In: Proc. of the SIAM Intl. Conf. on Data Mining (SDM '05), Newport Beach, USA. (2005). DOI: 10.1137/1.9781611972757.23

[17] Szathmary, L.: *Symbolic Data Mining Methods with the Coron Platform*. PhD Thesis in Computer Science, Univ. Henri Poincaré - Nancy 1, France (2006).

[18] Uno, T., Kiyomi, M. and Arimura, H.: *LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets*, In Jr., R.J.B., Goethals, B., Zaki, M.J., eds.: FIMI. Volume 126 of CEUR Workshop Proceedings., CEUR-WS.org (2004).

[19] Liu, G., Li, J. and Wong, L.: *A new concise representation of frequent itemsets using generators and a positive border*. Knowl. Inf. Syst., Vol. **17** No. **1**, pp. 35-56 (2008). DOI: 10.1007/s10115-007-0111-5

(*Beérkezett: 2019. február 22.*)

Szathmáry László 1977-ben született Debrecenben. A Debreceni Egyetemen 2001-ben szerzett programtervező-informatikus / angol-magyar szakfordító diplomát. Informatikai PhD-fokozatát a francia Henri Poincaré Egyetemen szerezte meg 2006-ban, Nancyban. Ezután posztdoktori ösztöndíjat nyert Kanadába a montreáli UQAM egyetemre. Magyarországra 2012-ben tért vissza, s azóta a Debreceni Egyetem Informatikai Karán dolgozik. Jelenlegi beosztása: habilitált egyetemi docens. Főbb kutatási területei: szimbolikus adatbányászat, mesterséges intelligencia, tudásreprezentáció.

SZATHMÁRY LÁSZLÓ
Debreceni Egyetem, Informatikai Kar
Információ Technológia Tanszék
4002 Debrecen, Pf. 400
szathmary.laszlo@inf.unideb.hu

A VERTICAL ALGORITHM FOR FINDING FREQUENT CLOSED ITEMSETS AND FREQUENT GENERATORS

László Szathmáry

Apriori is the most well-known algorithm for finding frequent itemsets (FIs) in a dataset. However, for generating interesting association rules, we also need the so-called frequent closed itemsets (FCIs) and the frequent generators (FGs), where FCIs and FGs form a subset of FIs. *Zart* is an *Apriori*-like algorithm that can filter FCIs and FGs among FIs. However, it is known that vertical itemset mining algorithms outperform the *Apriori*-like levelwise algorithms. *Eclat* is another well-known vertical miner that can produce the same output as *Apriori*, i.e. it also finds the FIs in a dataset. Here we propose an extension of *Eclat*, called *Eclog* that can filter FCIs and FGs among FIs. The proposed algorithm is a single-pass algorithm and it explores the frequent equivalence classes in a dataset. Experimental results show that *Eclog* performs very well, especially on dense, highly-correlated datasets.