IFJ. GYENESEI ATTILA

Egy hatékony megoldás a bolthálózatok döntéstámogatási problémáira: szekvencia mintázatok keresése

1. Bevezetés

„A számítógépek a bölcsesség forrását ígérték nekünk, azonban csak az adatok áradatát szállítják”- e megállapítás arra az alapvető szükségre világít rá, hogy a kutatási, termelési, üzleti folyamatokból származó adatokat információvá kell alakítani ahhoz, hogy megfelelő következtetéseket vonhassunk le belőlük, és ezek alapján döntéseket hozhassunk. Az adatbányászat tulajdonképpen összefüggés-keresés multidimenzionális térben.

Dolgozatom célja az adatbányászat egyik módszerének, a szekvencia mintázatok keresésének bemutatása és az eddig ismert algoritmusoknál egy optimálisabb és gyorsabb saját eljárás ismertetése. A módszer adott elemek együtt-előfordulását, mintázatát keresi az adatbázisban. A dolgozatban bemutatott megoldás az összes kombinációt számbavevő, sorozatokkal dolgozó eljárásokkal szemben kevesebb műveletet végrehajtó, irányított gráfokat alkalmaz.

A probléma aktualitását az adja, hogy a vezetői információs rendszereknél a beérkező adatok elemi szintűek, elemi módon tároltak az operatív rendszerekben, míg a vezetők globális rálátást igényelnek vállalatuk működésére.

Dolgozatom eredménye hozzájárul ahhoz, hogy az üzleti életben felhasználható – előzőleg ismeretlen – információt gyűjtsünk nagy adatbázisokból. A pénzügyi terület mellett az eljárás jól hasznosítható a tudományos kutatásban is, ahol nagy mennyiségű mérési, kísérleti adatot kell elemezni.

2. Mi az adatbányászat?

Az adatbányászat az üzleti életben és a tudományos kutatásokban is a ‘90-es évek elején indult el hódító pályáján. A tudományos kutatásokban elsősorban a

mesterséges intelligencia, azon belül is a neurális hálózatok vitték előre. Az üzleti életben a marketing volt eleinte a fő területe, mivel itt a legnagyobb az információáramlás sebessége, ezek a cégek próbáltak először információkat szerezni a vásárlópiacról, annak szokásairól. Korábban az adatokat a munka gyorsítása érdekében tárolták elektronikusan, a köztük lévő összefüggéseket (az információt) a hagyományos adatelemzési módszerek nem tették láthatóvá. A ‘90-es évek elején a cégek rájöttek, hogy az évtizedek óta gyûjtött adatokkal nem tudják üzletvitelüket hatékonyságát növelni.

Mára már minden, nagy mennyiségű adattal dolgozó területen nélkülözhetetlen. Néhány alkalmazási lehetősége az üzleti életben:
- célzott marketing: ügyfél célcsoportok meghatározása,
- ügyfél elégedettség-vizsgálat: kérdőívekre és demográfiai adatokra alapozva,
- értékesítési előrejelzés: korábbi forgalmi adatokra alapozva,
- üzemi és ügyviteli folyamatok optimalizálása,
- hitelkérelem-elbírálás,
- visszaélés felderítés: biztosítási csalások, hitelkártya visszaélések,
- befektetési portfólió elemzés, árfolyam-előrejelzés,
- beruházás menedzsment,
- raktárkészlet tervezés és optimalizálás,
- beszállító és ügyfélminősítés,
- számítógépek és hálózatok terhelésének előrejelzése, feladatütemezés.

Az adatbányászati módszerek csak részben új eljárások: például minta keresés adatokban, feltáró jellegű elemzések, előrejelzések. Támogatják vagy automatizálják a mintakeresési folyamatokat és főleg nagy adathalmazokra használhatók.

Az adatbányászat az alábbi elemző eszköztárak felhasználását jelenti:
- vizualizáció,
- magasszintű statisztikai módszerek,
- következtető rendszerek,
- neurális hálózatok.

Vizualizáció alatt a két- és háromdimeniós grafikonokat, tudományos és üzleti diagrammokat, speciális ábrázolásokat, térképeket, térinformatikai eszközök, valamint a multimédia használatát értjük.

A statisztikai eljárások között nem csak a szorosan vett matematikai módszerek találhatók. Ide tartoznak a statisztikai próbák, klaszter-, faktor-, diszkriminancia analízis, többdimenziós skálázás, lineáris és nemlineáris modellek,

kontingenciatáblák, preferencia térképek mellett például az idősorok elemzése, lineáris és nemlineáris regresszió-analízis, lineáris és nemlineáris programozás, az ökonometriai modellek, és szimulációs egyéb speciális eljárások.

3. Szekvencia mintázatok keresése

3.1. A probléma ismertetése

Az adatbányászat a legtöbb nagy bolthálózat döntéstámogatási problémáira hatékony megoldást jelent [1]. Az ilyen egységek egy vásárlás során tárolják a tranzakció dátumát, azoknak a tételeknek (termékeknek) a kódját, amelyet a vásárló ebben a tranzakcióban vásárolt. Nagyon gyakran az adatrekordot még növeli a vásárló azonosítója, például amikor készpénz helyett hitel- , illetve törzsvásárló- kártyával fizet.

Az ilyen adatbázisokon mutatjuk be a szekvencia mintázatok keresésének lehetséges megoldásait. Egy jellemző példa a mintázatra, mikor a vásárlók a „Csillagok háborúja” után a „A birodalom visszavág”, majd a „Jedi visszatér” című filmeket vásárolják egymás után. (Meg kell említenünk, hogy ezek a filmek külön-külön is megnézhetők, nem szükséges a „Jedi visszatér” megértéséhez az előző kettőt ismernünk.) Azok a vásárlók, akik más filmekkel együtt ezeket a filmeket is megvásárolták, támogatják ezt a szekvencia mintázatot.

3.2. A probléma formalizálása

Adott egy vásárló tranzakciókból álló *D* adatbázisunk, melyben minden tranzakció vásárlói azonosítóból, a vásárlás időpontjából és a vásárolt termékekből áll.
A következő kikötések tesszük:

1. egy vásárló egy vásárlási időegység alatt csak egyszer vásárolhat,
2. nem figyeljük a vásárolt termékek mennyiségét, minden terméket egy tranzakció során egy egységnek veszünk.

Az egy tranzakció során vett termékek halmazát *tranzakciótételnek*, a *tranzakciótételek* listáját pedig *sorozatnak* nevezzük. Az általánosság megsértése nélkül minden egyes *tételt* egy egész számmal fogunk jelölni. Tehát egy *i tranzakciótételt* $(i_1 i_2 \text{K}\ i_m)$-el jelölünk, ahol minden i_j egy tételt (terméket) jelent, egy *s sorozatot* pedig $(s_1 s_2 \text{K}\ s_n)$-el jelölünk, ahol minden s_j egy *tranzakciótételt* jelöl.

Definíció: Egy $\langle b_1 b_2 \text{K}\ b_m \rangle$ sorozat *tartalmaz* egy $\langle a_1 a_2 \text{K}\ a_n \rangle$ sorozatot, ha léteznek tetszõleg $(i_1 < i_2 < \text{K} < i_n)$ egész értékek úgy, hogy $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \text{K}, a_n \subseteq b_{i_n}$.
Jelölés: A fenti *tartalmaz* relációt „π” jelöljük.

Példa: A $\langle (7)(3\ \ 8)(9)(4\ \ 5\ \ 6)(8) \rangle$ sorozat tartalmazza a $\langle (3)(4\ \ 5)(8) \rangle$ sorozatot, azaz $\langle (3)(4\ \ 5)(8) \rangle \pi \langle (7)(3\ \ 8)(9)(4\ \ 5\ \ 6)(8) \rangle$, mivel $(3) \subseteq (3\ \ 8)$, $(4\ \ 5) \subseteq (4\ \ 5\ \ 6)$ és $(8) \subseteq (8)$.
Megjegyzés: A $\langle (3\ \ 5) \rangle$ sorozat nem tartalmazza a $\langle (3)(5) \rangle$ sorozatot, azaz $\langle (3)(5) \rangle \not\pi \langle (3\ \ 5) \rangle$ és fordítva, mivel az egyik sorozatban *3*-as és *5*-ös terméket külön tranzakció során vettük, míg a másikban egyben.

Definíció: Azt mondjuk, hogy az *s sorozat maximális*, ha nem létezik olyan sorozat, amelyik az *s* sorozatot tartalmazza.

Minden vásárlói tranzakciót együtt is tekinthetünk egy sorozatnak, ahol minden egyes tranzakciótétel termékek halmazából áll, és a tranzakciótételek listája tranzakció idő szerint növekvő sorrendbe vannak rendezve. Az ilyen sorozatot *vásárló sorozatnak* nevezzük.

Definíció: Legyenek T_1, T_2, K, T_n olyan vásárlói tranzakció idők, amelyek növekvő sorrendbe vannak rendezve. A vásárolt termékek halmazát - a T_i tranzakció időben - jelölje $\mathit{tranzakciótétel}(T_i)$.
Egy $\langle \mathit{tranzakciótétel}(T_1)\ \mathit{tranzakciótétel}(T_2)\ \text{K}\ \mathit{tranzakciótétel}(T_n) \rangle$ sorozatot *vásárló sorozatnak* nevezünk.

Definíció: Azt mondjuk, hogy egy vásárló *támogatja* az *s* sorozatot, ha *vásárló sorozata* tartalmazza az *s* sorozatot.

Feladatunk, hogy az összes olyan maximális sorozatot megtaláljuk, ami egy felhasználó által meghatározott minimális támogatottsággal rendelkezik. Minden ilyen feltételnek eleget tevő maximális sorozat reprezentál egy szekvencia mintázatot.

Definíció: Minden minimális támogatottságot kielégítő szekvenciát *támogatott szekvenciának* nevezünk.

Példa: Tekintsük az 1. ábrán lévő adatbázist. Az adatbázis minden egyes rekordja egy tranzakciót tárol, tranzakció idő szerint rendezve. Például a 4.

tranzakcióban a 2-es vásárló június 20-án egy 40-es, egy 60-as és egy 70-es kóddal jelölt terméket vásárolt egymás után, de egy tranzakcióban.

Vásárlás ideje	Vásárló azonosító	Vásárolt termékek
1998. június 10.	2	10, 20
1998. június 12.	5	90
1998. június 15.	2	30
1998. június 20.	2	40, 60, 70
1998. június 25.	4	30
1998. június 25.	3	30, 50, 70
1998. június 25.	1	30
1998. június 30.	1	90
1998. június 30.	4	40, 70
1998. július 25.	4	90

1. ábra

Vásárló azonosító	Vásárlás ideje	Vásárolt termékek
1	1998. június 25.	30
1	1998. június 30.	90
2	1998. június 10.	10, 20
2	1998. június 15.	30
2	1998. június 20.	40, 60, 70
3	1998. június 25.	30, 50, 70
4	1998. június 25.	30
4	1998. június 30.	40, 70
4	1998. július 25.	90
5	1998. június 12.	90

2. ábra

A 2. ábrán az adatbázisunk már vásárló azonosító és tranzakció idő szerint rendezett. Írjuk fel a vásárló sorozatokat (3. ábra).

Vásárló azonosító	Vásárló sorozat
1	⟨(30)(90)⟩
2	⟨(10 20)(30)(40 60 70)⟩
3	⟨(30 50 70)⟩
4	⟨(30)(40 70)(90)⟩
5	⟨(90)⟩

3. ábra

Látható, hogy a 2-es kóddal jelölt vásárló a vizsgált időszak során három tranzakciót hajtott végre. Az első vásárlásnál a 10-es, 20-as kóddal jelölt termékeket, a másodiknál csak a 30-as és a harmadik vásárlásnál a 40-es, 50-es és 60-as kóddal jelölt termékekből vásárolt.
Ha a vásárlói támogatottság például 25%, akkor azokat a mintázatokat keressük, amelyeket legalább két vásárló támogat az ötbõl. Ezek a ⟨(30)⟩, ⟨(40)⟩, ⟨(70)⟩, ⟨(90)⟩, ⟨(30)(40)⟩, ⟨(30)(70)⟩, ⟨(30)(90)⟩, ⟨(40)(70)⟩ és a ⟨(30)(40 70)⟩, de például a ⟨(10 20)(30)⟩ nem elégíti ki a fenti feltételt, mivel csak a 2-es kóddal jelölt vásárló támogatja. A kapott mintázatok közül azonban csak azok a megoldások, melyek maximálisak is. Két ilyen mintázat van: a ⟨(30)(90)⟩ és a ⟨(30)(40 70)⟩, amint ezt a 4. ábra is mutatja:

Szekvencia mintázatok legalább 25%-os támogatottságnál
⟨(30)(90)⟩
⟨(30)(40 70)⟩

4. ábra

3.3. A megoldás szakaszainak ismertetése

A szekvencia mintázatok keresésének megoldásmenetét a következő öt szakaszra bonthatjuk:

1) rendezési szakasz, ahol az eredeti adatbázist vásárló azonosító és tranzakció idő szerint rendezzük,
2) támogatottak szakasza, ahol meghatározzuk azokat a tételeket, amelyek minimális támogatottsággal bírnak,
3) transzformációs szakasz, ahol egy transzformációs adatbázist fogunk felépíteni, melyben már csak a támogatottak szerepelnek,

4) szekvencia szakasz, ahol a három algoritmussal megkeressük a szekvencia mintázatokat,
5) maximális szakasz, ahol a kapott szekvencia mintázatok közül is a maximálisakat kutatjuk fel.

A szakaszok részletes ismertetése előtt definiáljuk még a következőket:
Definíció: Egy *sorozat hosszán* a sorozatban lévő *tranzakciótételek* számát értjük. Egy *k* hosszú sorozatot *k-sorozatnak* nevezünk.

Definíció: Az *i tranzakciótétel* támogatott, ha van olyan vásárló, aki az *i* tranzakciótételben lévő minden tételt megvásárolta.
Megjegyzés: A *tranzakciótétel(i)* és a 1 hosszú $\langle i \rangle$ sorozatok megegyeznek.
Definíció: A minimális támogatottságú *tételeket* és az azokból felépülő tranzakciótételeket, melyek minimális támogatottsággal bírnak *támogatottaknak* nevezzük.
Megjegyzés: Minden *támogatott szekvenciában* lévő *tranzakciótétel* minimális támogatottságú.

1. *Rendezési szakasz:* Ebben a lépésben a kezdeti *D* adatbázist a vásárló azonosító, mint elsődleges kulcs szerint, és tranzakció idő, mint másodlagos kulcs szerint rendezzük. Azaz az eredeti adatbázist *vásárló sorozat* adatbázissá alakítjuk. A 3. ábra egy ilyen adatbázist mutat be.

2. *Támogatottak szakasza:* Ebben a részben először megkeressük az összes *M* minimális támogatottságú tranzakciótételt [2] [3] [4]. Egyidejűleg az összes *minimálisan támogatott 1-sorozatokat* is megtaláljuk, mivel $\{\langle m \rangle | m \in M\}$. A talált *támogatottakat* megfeleltetjük egy egész számnak. A példánkban a *támogatottak* a $\langle (30) \rangle$, $\langle (40) \rangle$, $\langle (70) \rangle$, $\langle (40)(70) \rangle$ és $\langle (90) \rangle$. Egy lehetséges megfeleltetést az 5. ábra mutat. Indok erre a megfeleltetésre, hogy két egyszerű támogatottat konstans időben össze tudunk hasonlítani, és a keresési idő is csökken, ha egy sorozat támogatását keressük egy vásárlói sorozatban.

Támogatottak	Megfeleltetés
(30)	1
(40)	2
(70)	3
(40 70)	4
(90)	5

5. ábra

3. <u>*Transzformációs szakasz:*</u> Ebben a szakaszban a *vásárló sorozatokból* felépítünk egy transzformációs adatbázist, melyben minden tranzakció már csak *támogatottat* fog tartamazni.
   A vásárló sorozatokon végighaladva a következőképpen járunk el:
   a) Ha egy tranzakciótétel nincs a támogatottak között, akkor azt a tranzakciótételt nem vesszük fel a traszformációs adatbázisba.
   b) Ha van olyan vásárló sorozat, amelyben nincs olyan tranzakciótétel, ami benne lenne a támogatottak között, akkor ezt a vásárlót se vegyük fel az új adatbázisba. (a vásárlók száma ettől még nem változik).

   A transzformációs adatbázisban az így kapott vásárló sorozatok a támogatottak listájából áll, melyet $\{m_1, m_2, \text{K}, m_n\}$ jelölünk, ahol minden m_j egy támogatott.
   Az ilyen transzformált adatbázist D_T-vel jelöljük.
   A példánkban kapott transzformációs adatbázist mutatja a 6. ábra. A 2-es vásárló sorozatából a $(10\ \ 20)$ tranzakciótételt nem vettük fel a transzformált adatbázisunkba, mivel nincs a támogatottak között (nem minimálisan támogatott), valamint a $(40\ \ 60\ \ 70)$ tranzakciótételből a $\{(40),(70),(40\ \ 70)\}$ támogatott listát kaptuk, melyben a 60-as tétel nem szerepel, mert nincs meg a minimális támogatottsága.

Vásárló azonosító	Eredeti vásárló sorozat	Transzformált vásárló sorozat	Megfelelt. után
1	$\langle(30)(90)\rangle$	$\langle\{(30)\}\{(90)\}\rangle$	$\langle\{1\}\{5\}\rangle$
2	$\langle(10\ \ 20)(30)(40\ \ 60\ \ 70)\rangle$	$\langle\{(30)\}\{(40),(70),(40\ \ 70)\}\rangle$	$\langle\{1\}\{2,3,4\}\rangle$
3	$\langle(30\ \ 50\ \ 70)\rangle$	$\langle\{(30),(70)\}\rangle$	$\langle\{1,3\}\rangle$
4	$\langle(30)(40\ \ 70)(90)\rangle$	$\langle\{(30)\}\{(40),(70),(40\ \ 70)\}\{(90)\}\rangle$	$\langle\{1\}\{2,3,4\}\{5\}\rangle$
5	$\langle(90)\rangle$	$\langle\{(90)\}\rangle$	$\langle\{5\}\rangle$

6. ábra

4. <u>*Szekvencia szakasz:*</u> Ebben a részben megkeressük a szekvencia mintázatokat. Ezt a 3.4.-es pontban mutatjuk be.

5. <u>*Maximális szakasz:*</u> Az előző részben megtalált szekvencia mintázatok közül fogjuk ebben a szakaszban a maximálisat kiválasztani. Az AprioriAll algoritmuson kívül ezt a lépést a szekvencia szakasszal kombináljuk.
   Mivel egy maximális sorozatot úgy definiáltunk, hogy egy *s sorozat maximális*, ha nem létezik olyan sorozat, amelyik az *s* sorozatot tartalmazza, ezért feladatunk a kapott mintázatok közül azok törlése, amelyek nem maximálisak.

Tekintsük a lehetséges *S* sorozatok közül a leghosszabbat. Legyen ennek hossza *n*. Ennek az *S* sorozat részsorozatainak törlését a következő egyszerű algoritmussal tehetjük meg:

for ($k = n$; $k > 1$; k --) **do**
  for minden s_k *k-sorozat-ra* **do**
    Összes s_k törlése az *S*-ből

3.4. Szekvencia szakasz

Ebben a szakaszban határozzuk meg azokat a szekvencia mintázatokat, melyek minimális támogatottsággal rendelkeznek. A szakirodalom alapján bemutatunk két algoritmust, melyek az Apriori családból származnak [3]. Ezen algoritmusok jellemzője, hogy többszörös vizsgálatot végeznek az adatokkal.

Az Apriori algoritmus két típusa a AprioriAll és az AprioriSome [3] [5]. Az AprioriAll algoritmus az első vizsgálatban veszi azon *1-sorozatokat*, melyek minimális támogatottsággal rendelkeznek. Ezeket a támogatottak szakaszában állapítottuk meg. Ezekből elindulva, az összes lehetséges párosítást elvégezve létrehoz lehetséges *2-sorozatokat*, melyeket *jelölt sorozatoknak* nevez. Minden jelölt sorozatnak megkeresi a támogatottságát az összes vásárlói sorozaton. Ha ez nem éri el a minimális támogatottságot, akkor törli a jelölt sorozatból. Ha minden új jelöltre a vizsgálatot elvégezte, akkor új jelölteket alapul véve ismételi meg az iterációt. Az iteráció akkor fejeződik be, mikor nem tud új jelölteket előállítani. Az AprioriAll algoritmus tehát az összes támogatott sorozatot felkutatja, beleértve a nem maximálisokat is. Az AprioriSome algoritmus egy felhasználó által megadott függvény szerint nem minden jelöltgenerálás után végzi el a vizsgálatot, hanem a függvényben megadott sorozat hosszúságoknál. Ez az algoritmus abból indul ki, hogy jobb inkább több, esetleg nem támogatott sorozatokat generálni, mint minden egyes generálás után a nagy adatbázisban a jelöltek támogatását kutatni.

Az Apriori algoritmusok után bemutatunk egy harmadik algoritmust, ami az előző kettőhöz képest a sorozatokkal dolgozó eljárásokkal szemben kevesebb műveletet végrehajtó, irányított gráfokat alkalmaz.

3.4.1. AprioriAll algoritmus

Az algoritmust a 7. ábra mutatja. L_k jelöli a támogatott *k-sorozatokat* és C_k a jelölt *k-sorozatokat*. Minden egyes vizsgálat során az előző vizsgálatban kapott támogatott jelöltekből generálja a következő lehetséges jelölteket. Ha vége egy vizsgálatnak, akkor a támogatottságukat ellenőrzi és amelyeké nem éri el a minimálisat, azt töröli a jelöltek közül. Az első vizsgálatban az *1-sorozatokkal* kezdi a vizsgálatot.

L_1 = {*1-sorozatok* a támogatottak közül}
for (k = 2; $L_{k-1} \neq 0$; k++) **do**
begin
C_k = Új jelöltek generálása az L_{k-1}-ből (lásd 3.4.1.1.)
for minden *c vásárló sorozat*-ra az adatbázisban **do**
C_k jelöltek számlálóinak növelése, ha benne vannak *c*-ben
L_k = C_k azon jelöltjei, amelyek minimális támogatottak
end
Válasz = Sorozatok Y_k L_k-ban

7. ábra: AprioriAll algoritmus

3.4.1.1. Apriori jelöltgenerálás

Ebben a függvényben a *(k-1)-sorozatokból* generálja a *k* hosszúságú jelölteket. A generálás úgy történik, hogy az L_{k-1}-ben lévő sorozatokat összekapcsolja az L_{k-1}-ben lévő sorozatokkal a következőképpen:

insert into C_k
select $p.támogatottak_1, p.támogatottak_2, \mathrm{K}, p.támogatottak_{k-1}, q.támogatottak_{k-1}$
from L_{k-1} p, L_{k-1} q
where $p.támogatottak_1 = q.támogatottak_2, \mathrm{K}, p.támogatottak_{k-2} = q.támogatottak_{k-2}$;

Ezután törli az összes olyan $c \in C_k$ jelölt sorozatot, amely sorozat *(k-1)* hosszú részsorozata nincs benne L_{k-1}-ben:

for minden $c \in C_k$ sorozat-ra **do**
for minden *c*-nek, *s (k-1)-részsorozatára* **do**

if ($s \notin L_{k-1}$) **then**
  delete c **from** C_k ;

Sorozat	Támogatottsága
$\langle 1 \quad 2 \quad 3 \rangle$	2
$\langle 1 \quad 2 \quad 4 \rangle$	2
$\langle 1 \quad 3 \quad 4 \rangle$	3
$\langle 1 \quad 3 \quad 5 \rangle$	2
$\langle 2 \quad 3 \quad 4 \rangle$	2

8. ábra

Tekintsük példaként a 8. ábrát, ahol *3-sorozatokat* láthatunk az L_3 halmazban. Ha ezeket inputként megkapja a függvény, akkor a 9. ábrán lévő sorozatokat adja az összekapcsolás után. Például az $\langle 1 \quad 2 \quad 4 \quad 3 \rangle$-at úgy kapta, hogy az $\langle 1 \quad 2 \quad 4 \rangle$-et és az $\langle 1 \quad 2 \quad 3 \rangle$-at összekapcsolta:

$\langle 1 \quad 2 \quad 3 \quad 4 \rangle$
$\langle 1 \quad 2 \quad 4 \quad 3 \rangle$
$\langle 1 \quad 3 \quad 4 \quad 5 \rangle$
$\langle 1 \quad 3 \quad 5 \quad 4 \rangle$

9. ábra

A kapott sorozatok közül ezután törli az összes olyan sorozatot, amely sorozat részsorozata nincs benne L_3-ban. A 10. ábra mutatja, hogy egyetlen olyan sorozat van, amelynek mindegyik részsorozata benne van az L_3-ban. Például az $\langle 1 \quad 2 \quad 4 \quad 3 \rangle$ sorozatot azért kellett törölnie, mert a $\langle 2 \quad 4 \quad 3 \rangle$ részsorozata nincs L_3-ban.

$\langle 1 \quad 2 \quad 3 \quad 4 \rangle$

10. ábra

Példa: Tekintsük a 11. ábrán lévő vásárló sorozatokat. Ebben a példában nem mutatjuk meg az eredeti adatbázist. A vásárló sorozatok már transzformálva vannak, így minden vásárló sorozat csak támogatottakat tartalmaz. A támogatottakat megfeleltettük egy egész számnak. A minimális támogatottság legyen 40%, azaz legalább 2 vásárlónak kell támogatnia a keresett szekvenciákat.

⟨{1 5}{2}{3}{4}⟩
⟨{1}{3}{4}{3 5}⟩
⟨{1}{2}{3}{4}⟩
⟨{1}{3}{5}⟩
⟨{4}{5}⟩

11. ábra: Vásárló sorozatok

Az első vizsgálat során felírjuk az egy hosszú sorozatok támogatottságát, amit a 12. ábra mutat, majd az algoritmus lépéseit folytatva a következő 12.,13.,14.,15. ábrán látható megoldásokat kapjuk:

Sorozat	Támogatottság
⟨1⟩	3
⟨2⟩	2
⟨3⟩	4
⟨4⟩	4
⟨5⟩	2

12. ábra: Támogatott *1-sorozatok*

Sorozat	Támogatottság
⟨1 2⟩	2
⟨1 3⟩	4
⟨1 4⟩	3
⟨1 5⟩	3
⟨2 3⟩	2
⟨2 4⟩	2
⟨3 4⟩	3
⟨3 5⟩	2
⟨4 5⟩	2

13. ábra: Támogatott *2-sorozatok*

Sorozat	Támogatottság
⟨1 2 3⟩	2
⟨1 2 4⟩	2
⟨1 3 4⟩	3
⟨1 3 5⟩	2
⟨2 3 4⟩	2

14. ábra: Támogatott *3-sorozatok*

Sorozat	Támogatottság
⟨1 2 3 4⟩	2

15. ábra: Támogatott *4-sorozatok*

Az ötödik vizsgálat során már nem tudunk jelölteket állítani. A maximális szakaszban leírt algoritmus végrehajtása után, azaz a nem maximális sorozatok törlése után a 16. ábrán látható megoldásokat kapjuk:

Sorozat	Támogatottság
$\langle 1 \quad 2 \quad 3 \quad 4 \rangle$	2
$\langle 1 \quad 3 \quad 5 \rangle$	2
$\langle 4 \quad 5 \rangle$	2

16. ábra: Maximálisan támogatott sorozatok

3.4.2. AprioriSome algoritmus

Az algoritmus a 17. ábrán látható. Különbség az AprioriAll algoritmushoz képest, hogy ez egy előrehaladó és egy hátrahaladó szakaszból áll. Az előrehaladó szakaszban minden jelölt generálást elvégez, de csak bizonyos hosszúságú sorozatoknak a támogatottságát vizsgálja, a többiét majd a hátrahaladó szakaszban. Például az 1, 2, 4 és 6 hosszúságú sorozatokat az előrehaladó szakaszban, míg a 3 és 5 hosszúakat a hátrahaladó szakaszban vizsgálja. Ezt egy *next* függvény szabályozhatja, mely bemenetként megkapja a legutolsó lépésben vizsgált sorozat hosszát és kiadja, hogy a következő lépésben milyen hosszú sorozatokat vizsgáljon az algoritmus. Egy speciális eset, amikor *next*(k) = k+1, ahol k volt az utolsó jelölthosszúság, amikre a támogatottságot megvizsgálta, mivel ez éppen az AprioriAll algoritmust adja (tetszõleges hosszú jelöltekre elvégezzük a támogatásvizsgálatot).
Jelölés: hit_k jelölje a támogatott *k-sorozatok* és a jelölt *k-sorozatok* arányát: $hit_k = \frac{|L_k|}{|C_k|}$.

A *next* függvényre egy példa lehet a következő:

```
function next(k : integer)
begin
        if (hit_k < 0.666)    return k+1
        elseif (hit_k < 0.75) return k+2
        elseif (hit_k < 0.80) return k+3
        elseif (hit_k < 0.85) return k+4
        else                  return k+5
end
```

Új jelölt generálásához az AprioriAll algoritmusnál ismertetett algoritmust használja. Ha a k-dik vizsgálatnál az L_{k-1}-ben nincsenek minimális támogatású

jelöltek, mert azt a lépést kihagyta, akkor a C_{k-1}-ben lévõ jelölteket használja C_k generálásához. Ez megtehetõ, mivel $C_{k-1} \supseteq L_{k-1}$.

```
// Előrehaladó szakasz
L1 = {támogatott 1-sorozatok}
C1 = L1
utolsó = 1 // az utoljára megszámolt C_utolsó
for (k = 2; C_{k-1} ≠ ∅ and L_utolsó ≠ ∅; k++) do
      begin
      if ( L_{k-1} ismert) then
      C_k = Új jelöltgenerálás L_{k-1}-bõl
   else
      C_k = Új jelöltgenerálás C_{k-1}-bõl
   if (k == next(utolsó)) then begin
         for minden c vásárló sorozat-ra az adatbázisban do
               C_k jelöltek számlálóinak növelése, ha benne vannak c-ben
         L_k = C_k azon jelöltjei, amelyek minimális támogatottak
         utolsó = k
         end
   end
// Hátrahaladó szakasz
for (k--; k >=1; k--) do
   if ( L_k nincs meghatározva az előrehaladó szakaszban) then begin
   Az összes C_k-ban lévõ sorozat törlése, amit L_i tartalmaz, i > k
   for minden c vásárló sorozat-ra a D_T-ben do
         C_k jelöltek számlálóinak növelése, ha benne vannak c-ben
   L_k = C_k azon jelöltjei, amelyek minimális támogatottak
   end
   else begin
   Az összes olyan sorozat törlése L_k-ban, amit L_i tartalmaz, i > k
   end
Válasz = Sorozatok Y_k L_k-ban
```

17. ábra: AprioriSome algoritmus

A hátrahaladó szakaszban elõször törli az összes (megvizsgált) minimális sorozatot, amik nem maximálisak valamint a nem vizsgált részsorozatokat, csak utána vizsgálja meg az eddig még nem vizsgált jelöltek támogatottságát.

Példa: Oldjuk meg az AprioriAll algoritmusnál használt 11. ábrán lévő feladatot. A 12. ábrán a minimális vizsgálat után láthatók a támogatott *1-sorozatok*. Az egyszerűség kedvéért, legyen $next(k) = 2k$. A második vizsgálatban is elvégezzük a minimális támogatottságnak a vizsgálatát és a feltételnek eleget tevőket felírjuk a C_2-ből az L_2-be (13. ábra). A C_3-ban levő jelölteket a 18. ábra mutatja. Viszont ebben a lépésben nem vizsgáljuk meg a jelöltek támogatását, hanem folytatjuk a C_4 jelöltek generálásával a C_3-ból. A C_4 jelöltek támogatását szintén megvizsgáljuk és a megfelelőket az L_4 halmazba tesszük (15. ábra). A hátrahaladó szakaszban az L_4-ből nem törlünk, mert itt nincs hosszabb a többinél, ezért része sem lehet, hanem megyünk a C_3 jelöltjeihez. Töröljük ebből azokat a részsorozatokat, melyek az L_4 sorozatok valamelyikének része. A „maradékokat” mutatja a 19. ábra.

$\langle 1 \quad 2 \quad 3 \rangle$
$\langle 1 \quad 2 \quad 4 \rangle$
$\langle 1 \quad 3 \quad 4 \rangle$
$\langle 1 \quad 3 \quad 5 \rangle$
$\langle 2 \quad 3 \quad 4 \rangle$
$\langle 3 \quad 4 \quad 5 \rangle$

18. ábra

Ezek közül csak az $\langle 1 \quad 3 \quad 5 \rangle$ sorozat maximális *3-sorozat*. A következő lépésben a $\langle 4 \quad 5 \rangle$ sorozat kivételével minden *2-sorozatot* törlünk, mert részsorozatok, csak úgy mint az 1 hosszú sorozatok az L_1-ben. Így megkaptuk az $\langle 1 \quad 2 \quad 3 \quad 4 \rangle$, $\langle 1 \quad 3 \quad 5 \rangle$ és $\langle 4 \quad 5 \rangle$ maximális sorozatokat.

$\langle 1 \quad 3 \quad 5 \rangle$
$\langle 3 \quad 4 \quad 5 \rangle$

19. ábra

Ebben a példában az AprioriSome csak két 3 hosszú sorozatokat vizsgált, szemben az AprioriAll-nál 6 vizsgálattal. Viszont így sokkal több minimális támogatással nem rendelkező sorozatokkal generál újabbakat. A generálást azonban gyorsabban végre tudja hajtani, mint nagy adatbázisban minden egyes jelölt támogatásának a vizsgálatát.

3.4.3. Feladat megoldása irányított gráfokkal

A gráfok rendkívül gyakran használt struktúrák, a gráfokkal foglalkozó algoritmusok alapvető szerepet játszanak a számítástudományban. Számos érdekes probléma fogalmazható és oldható meg gráfok segítségével. Ebben a részben a szekvencia mintázatok keresését mutatjuk be irányított gráfokkal.

3.4.3.1. Vásárlói sorozat ábrázolása irányított gráfokkal

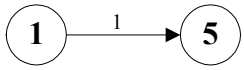
Definíció: A $\vec{G}$ irányított gráf egy négyes: (V, E, K, B). V és E halmazok, ahol V elemeit pontoknak vagy csúcsoknak, E elemeit éleknek nevezzük. K és B két reláció V és E között. Minden e élre a $\{v \in V : vKe\}$ és $\{v \in V : vBe\}$ halmazok egyeleműek. Ha $\{vBe\}$, akkor azt mondjuk, hogy v az e él végpontja vagy e bevezet v-be. Ha u az e él kezdőpontja és v a végpontja, akkor azt mondjuk, hogy az e él egy uv él; e u-ból v-be megy.

Lényeges különbség az Apriori algoritmusokhoz képest, hogy míg az Apriori algoritmusok sorozatonként ábrázolják az vásárlói sorozatokat, addig a gráf algoritmusok vásárlói gráfokat építenek fel a transzformációs adatbázisban.

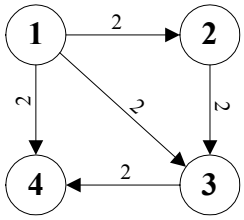
Transzformált vásárló sorozat	Megfeleltetés után
⟨{(30)}{(90)}⟩	⟨{1}{5}⟩
⟨{(30)}{(40),(70),(40 70)}⟩	⟨{1}{2,3,4}⟩
⟨{(30),(70)}⟩	⟨{1,3}⟩
⟨{(30)}{(40),(70),(40 70)}{(90)}⟩	⟨{1}{2,3,4}{5}⟩
⟨{(90)}⟩	⟨{5}⟩

20. ábra

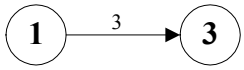
Írjuk fel a 20. ábrán látható vásárló sorozatokat egy-egy gráfba, ahol a támogatottak lesznek a csúcspontok és az élekre a vásárlónak az azonosítóját írjuk.



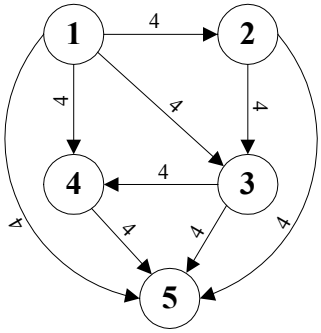
1-es vásárló *vásárló sorozata*



2-es vásárló *vásárló sorozata*

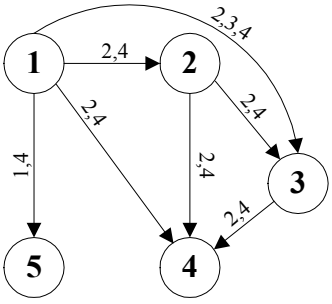


3-as vásárló *vásárló sorozata*



4-es vásárló *vásárló sorozata*

Az 5-ös vásárló vásárló sorozata egy csúcspontnak felel meg, amiből és amibe nem megy él, mivel a vásárló csak egy tranzakciót hajtott végre. A 21. ábrán a vásárló sorozatokat egyetlen gráffal reprezentáljuk, ahol az éleken azon vásárlók azonosítója szerepel, akik az adott „utat” támogatják. Csak azokat az éleket vesszük fel a gráfba, amelyek minimális támogatottsággal rendelkeznek. Ha a minimális támogatottság 25%, akkor például a 4-es és az 5-ös csúcs között nem vezet él, mert csak egy vásárló támogatja ezt az utat a minimális kettő helyett:



21. ábra

3.4.3.2. Az algoritmus

Az algoritmus feladata, hogy a vásárlói gráfokból felírt, csak minimális éleket

tartalmazó gráfban megtaláljuk a minimális támogatottságú utakat. Két algoritmust fogunk bemutatni, az egyik minden csúcspontból elindul és megkeresi a csúcsból induló minimális támogatottságú utakat, míg a másik csak abból a csúcspontból indul el, amelyiken még nem járt és az út keresése során nem lép olyan élre, melyet már egy másik csúcspontból indított keresés felhasznált.

Két módszert szokás használni egy $G=(V,E)$ gráf ábrázolására: az egyikben szomszédsági listákkal, a másikban csúcsmátrixszal (szomszédsági mátrixszal) adjuk meg a gráfot. Mindkét algoritmus során feltesszük, hogy a bemeneti gráfot szomszédsági listákkal ábrázoljuk.

3.4.3.2.1. Gyenge algoritmus

Az algoritmus a 22. ábrán látható. A vásárlói $G=(V,E)$ gráf szomszédsági listás ábrázolása esetén egy *Szomszéd* tömböt használunk. Ez $|V|$ darab listából áll, és a *Szomszéd* tömbben minden csúcshoz egy lista tartozik. Minden $u \in V$ csúcs esetén, a $Szomszéd[u]$ szomszédsági lista tartalmazza az összes olyan *v* csúcsot, amelyre létezik $(u,v) \in E$ él. Azaz: $Szomszéd[u]$ elemei *u* csúcs *G*-beli szomszédjai.

GYENGE(G):
for minden $k \in V[G]$ csúcsra **do**
Előd[*v*] ← NIL
for minden $k \in V[G]$ csúcsra **do**
begin
Mutat ← HÁTRA
GYENGE-BEJÁR(*k*)
end

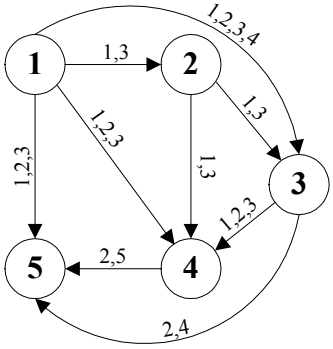
GYENGE-BEJÁR(*u*):
for minden $v \in Szomszéd[u]$ csúcsra **do**
begin
if $k=u$ **then** $T_{C_k} \leftarrow T_{(u,v)}$
if $T_{C_k} \overset{\min.}{=} T_{(u,v)}$ **then**
begin
Előd[*v*] ← *u*, *Előző_u* ← *u*
$C_k \leftarrow (C_k \cup (u,v))$
$T_{C_k} \leftarrow (T_{C_k} \cap T_{(u,v)})$

```
            Mutat ← ELÕRE
            GYENGE-BEJÁR(v)
            end
        end
    if Mutat = ELŐRE then
        begin
        L_k ← C_k
        T_Lk ← T_Ck
        Mutat ← HÁTRA
        end
    if k ≠ u then
        C_k ← (C_k / (Elõzõ_u, u))
```

22. ábra

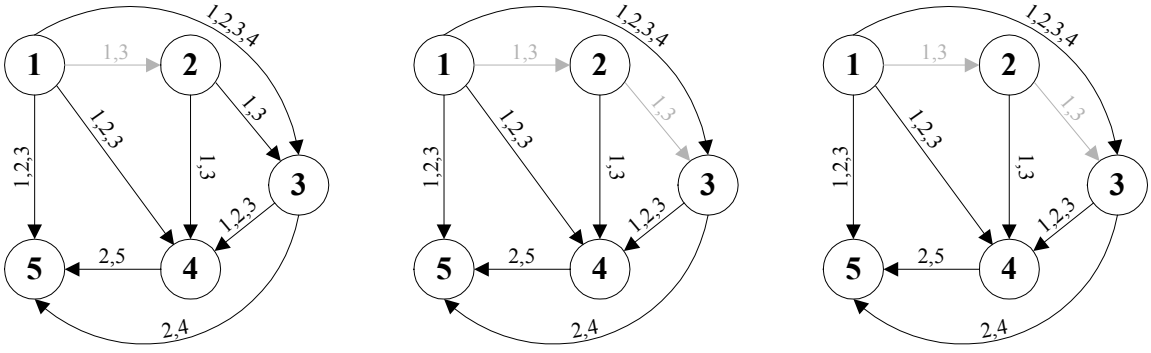
Az algoritmus minden $k \in V[G]$ esetén a G éleit szisztematikusan megvizsgálja, és így rátalál minden k-ból elérhető csúcsra. Elérhető csúcsnak nem egy pont szomszédait nevezzük, hanem azokat a szomszédokat, melyek minimális támogatottsága megegyezik az addig megtett út támogatottságával.A keresés során az utoljára elért, új élekkel rendelkező v csúcsból kivezető, még nem vizsgált éleket derítjük fel. Ha v-hez tartozó összes élt megvizsgáltuk, akkor a keresés „visszalép”, és megvizsgálja annak a csúcsnak a kivezető éleit, amelyből v-t elértük. Ezt addig folytatja, amíg el nem éri az összes csúcsot, amely elérhető az eredeti kezdő csúcsból. Visszalépés előtt egyrészt a kezdő csúcsból addig megtett utat az L_k halmazba írjuk (ezt a C_k halmaz tárolja), ami az algoritmus befejezése után az összes lehetséges minimális támogatottságú utat tartalmazni fogja, másrészt a T_{L_k} halmazba azokat a vásárló azonosítókat írjuk (ezt a T_{C_k} halmaz tárolja), akik támogatták ezt az utat. Ha a keresés elérte az összes olyan csúcsot, ami az eredeti csúcspontból elérhető, akkor vesszük a következő $k \in V[G]$ kezdő csúcspontot.

Példa: A 23. ábra a 11. ábrán látható vásárlói sorozat gráfját prezentálja. A GYENGE algoritmus működésének szemléltetését a 24. ábra mutatja.

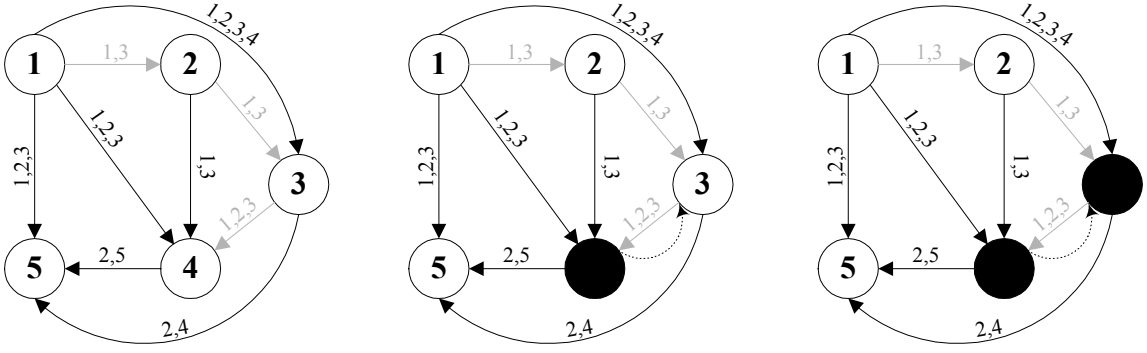


23. ábra

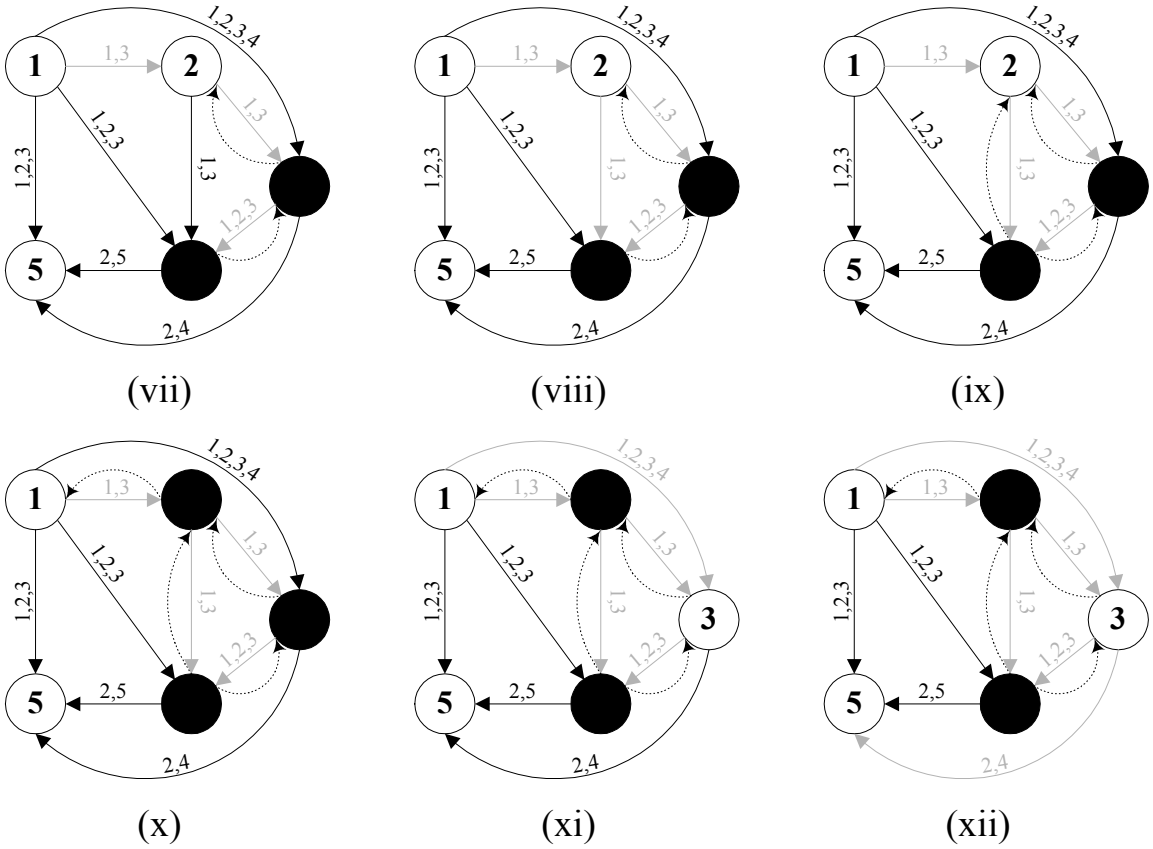
A 24. ábrán az 1-es csúcspontból kiinduló keresést láthatjuk tizenkét lépésben. Azt a csúcspontot, ahol éppen tart a keresés „bepöttyöztük” és azokat az éleket, amelyen halad az algoritmus, elhalványítottuk. Visszafelé lépegetésnél a pontozott nyilat használtuk. Például a



(i) (ii) (iii)



(iv) (v) (vi)



(vii) (viii) (ix)

(x) (xi) (xii)

24. ábra

(iv) lépésben értünk el a 4-es csúcsponthoz, aminek nincs olyan szomszédja, ahová folytathatnánk az utunkat (az 5-ös csúcspontba azért nem mehetünk, mert az $\langle 1234 \rangle$ C_k út T_{C_k} támogatottsága (1,3), míg a (45) él támogatottsága (2,5)), a megtett utat beírjuk az L_k halmazba, a támogatottságát a T_{L_k} -ba, és visszalépünk a 3-as csúcsponthoz.

A keresés lépéseit csak a tizenkettedik lépésig mutattuk be. Ha a keresés az 1-es csúcspontból elérhető összes utat felfedezi, utána vesszük a többi csúcspontot és azokra is végrehajtjuk az algoritmust. A különböző csúcspontokból kiindulva a következő minimális támogatottságú utakat kapjuk (sorrendbe téve egy lehetséges végrehajtás során):

Út	Vevők (akik támogatják)
$\langle 1 \quad 2 \quad 3 \quad 4 \rangle$	1,3
$\langle 1 \quad 2 \quad 4 \rangle$	1,3
$\langle 1 \quad 3 \quad 5 \rangle$	2,4
$\langle 1 \quad 4 \rangle$	1,2,3
$\langle 1 \quad 5 \rangle$	1,2,3

1-es csúcspontból kiindulva

Út	Vevők (akik támogatják)
$\langle 2 \quad 3 \quad 4 \rangle$	1,3
$\langle 2 \quad 4 \rangle$	1,3

2-es csúcspontból kiindulva

Út	Vevők (akik támogatják)
$\langle 3 \quad 4 \rangle$	1,2,3
$\langle 3 \quad 5 \rangle$	2,4

3-as csúcspontból kiindulva

Út	Vevők (akik támogatják)
$\langle 4 \quad 5 \rangle$	2,5

4-es csúcspontból kiindulva

Megjegyzés: Ha kiszűrjük a nem maximális utakat, akkor valóban az $\langle 1 \quad 2 \quad 3 \quad 4 \rangle$, $\langle 1 \quad 3 \quad 5 \rangle$ és $\langle 4 \quad 5 \rangle$ megoldásokat kapjuk.

Látható a GYENGE algoritmus pazarlása: mivel minden olyan csúcsponton áthalad, amin már járt, sok utat feleslegesen tesz meg. Ezt a pazarlást fogja megszüntetni az ERŐS algoritmus.

3.4.3.2.2. Erős algoritmus

Az algoritmust a 25. ábra mutatja. A GYENGE kereséshez hasonlóan, ha egy már elért *u* csúcs szomszédsági listájának vizsgálata során elérünk egy *v* csúcsot, akkor a keresés *u*-t feljegyzi, mint elődjét, azaz *Előd*[*v*] értékét u-ra állítja. A keresés által elõállított elõd részgráf több fából állhat, hiszen a keresést többször hajthatjuk végre különbözõ kezdõ csúcsokból kiindulva. Az *elõd részgráf* definíciója $G_{\text{Elõd}} = (V, E_{\text{Elõd}})$, ahol $E_{\text{Elõd}} = \{(\text{Elõd}[v], v) : v \in V$ és $\text{Elõd}[v] \neq NIL\}$. Az elõd részgráf egy *erdõ*, mely több fát tartalmaz. $E_{\text{Elõd}}$ éleit *fa éleknek* nevezzük.

ERŐS(G):

for minden $k \in V[G]$ csúcsra **do**
begin
szín[*k*] ← FEHÉR
Előd[*v*] ← NIL
end

```
    for minden k ∈ V[G] csúcsra do
      if szín[k] = FEHÉR then
          begin
          Mutat ← HÁTRA
          ERŐS-BEJÁR(k)
          end

ERŐS-BEJÁR(u):
    szín[u] ← SZÜRKE
    for minden v ∈ Szomszéd[u] csúcsra do
      begin
      if k = u then T_{C_k} ← T_{(u,v)}
      if T_{C_k} (min.) ≠ T_{(u,v)} then
          szín[u] ← FEHÉR
      else
          begin
          if szín[v] = FEHÉR then
              begin
              Előd[v] ← u, Előző_u ← u
              C_k ← (C_k Y (u,v))
              T_{C_k} ← (T_{C_k} I T_{(u,v)})
              Mutat ← ELŐRE
              ERŐS-BEJÁR(v)
              end
          if szín[v] = FEKETE then
              begin
              if w ∈ Szomszéd[v] = ∅ then
                  begin
                  C_k ← (C_k Y (u,v))
                  Előző_u ← u
                  Mutat ← ELŐRE
                  end
              if w ∈ Szomszéd[v] ≠ ∅ then
                  begin
                  C_k ← (C_k Y (u,v))
                  for w ∈ Szomszéd[v] csúcsra do
                      if T_{C_k} (min.) = T_{L_w} then
                          begin
```

```
                                    L_k ← (C_k Y L_w)
                                    T_{L_k} ← (T_{C_k} I T_{L_w})
                                    Előző_u ← u
                                    Mutat ← HÁTRA
                                    end
                            C_k ← (C_k / (Elõzõ_u, u))
                            end
                        end
                    end
            if szín[u] = SZÜRKE then
                szín[u] ← FEKETE
            if Mutat = ELŐRE then
                begin
                L_k ← C_k
                T_{L_k} ← T_{C_k}
                Mutat ← HÁTRA
                end
            if k ≠ u then
                C_k ← (C_k / (Elõzõ_u, u))
```

25. ábra

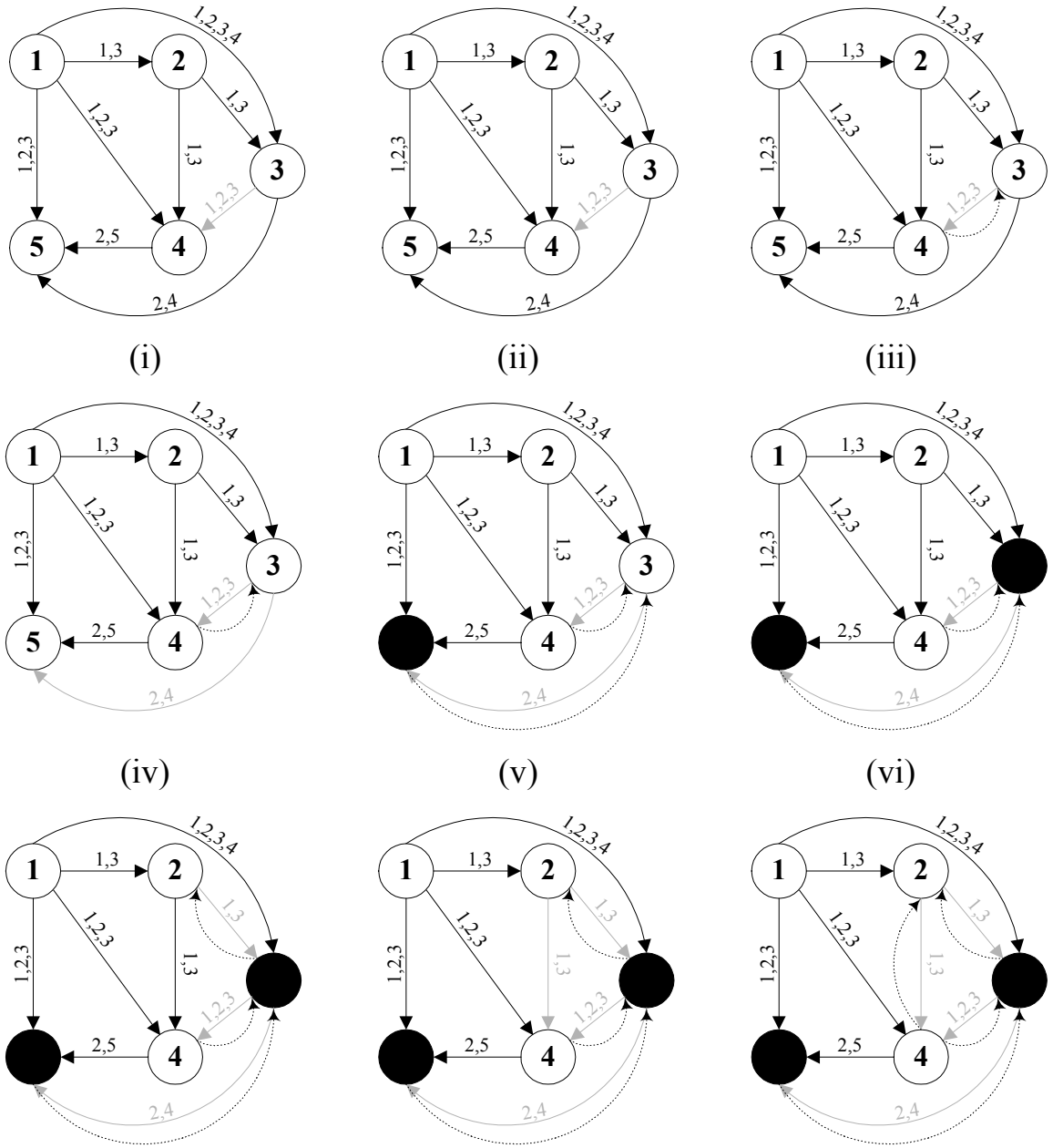
A csúcsok állapotait színekkel különböztetjük meg. Kezdetben minden csúcs fehér, amikor elérünk egy csúcsot, szürkére színezzük, és befeketítjük, ha elhagytuk, azaz amikor a szomszédsági listájának minden elemét megvizsgáltuk.

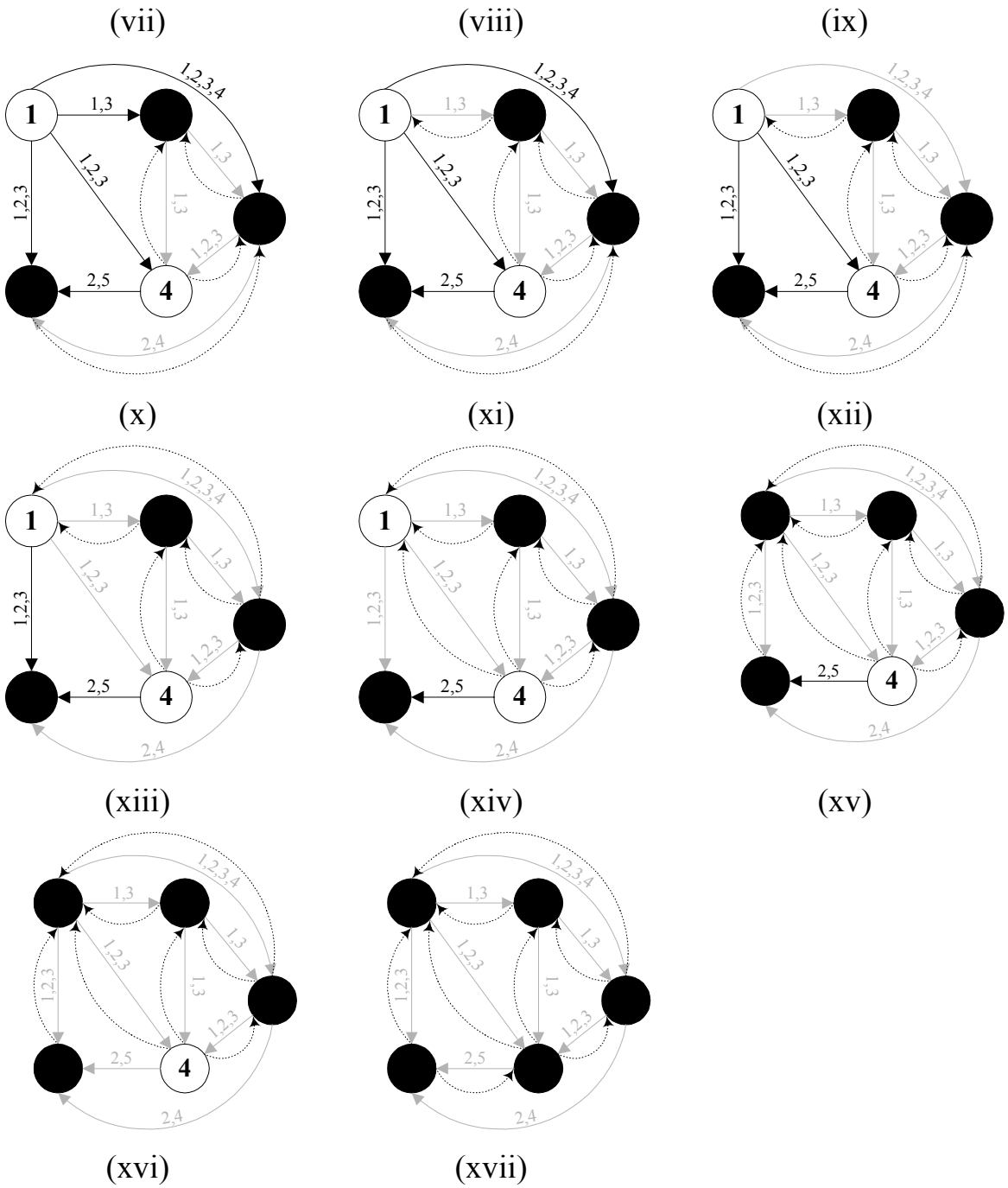
Az ERŐS első öt sorában minden csúcs színét fehérre állítjuk és az *Előd* értékek kezdetben NIL-ek lesznek. A 6-11. sorokban V csúcsain haladunk végig, és ha fehér csúcshoz jutunk, az ebből elérhető csúcsokat bejárjuk ERŐS-BEJÁR segítségével. Valahányszor ERŐS-BEJÁR(k)-t meghívjuk a 10. sorban, a csúcs az erdő egy új fájának gyökere lesz.

ERŐS-BEJÁR(u) hívásakor az u csúcs fehér. Az első sorban u színét szürkére változtatjuk. A 2-39. sorokban megvizsgáljuk u összes v szomszédját. Ha az eddigi C_k út és az (u,v) él támogatottsága nem minimális, akkor az u csúcspont színét vissza kell állítanunk fehérre, hogy egy későbbi keresés során ezen az (u,v) élen is közlekedhessünk. A 9-16. sorokban a v alatti részt bejárjuk rekurzív módon, ha v fehér, valamint ha az eddigi C_k út és az (u,v) él támogatottsága minimális. Ha v fekete, akkor ez azt jelenti, hogy arra már korábban jártunk,

most csak azt kell megvizsgálnunk, hogy a *v* szomszédjait tartalmazó út támogatottsága és az az eddig bejárt C_k út támogatottsága minimális-e. Ha igen, akkor C_k utunkhoz hozzáfűzzük a *v* szomszédjait tartalmazó utat. Ha nem, vagy *v*-nek nincs szomszédja, akkor csak az (*u*,*v*) élet írjuk hozzá eddigi utunkhoz. Végül, miután az *u*-ból kivezető összes élt megvizsgáltuk, a 41-50. sorokban *u* színét feketére állítjuk (feltéve, ha közben nem írtuk át fehérre), és feljegyezzük L_k-ba a kezdő csúcsból megtett utat, valamint T_{L_k}-ba ennek támogatottságát.

Példa: Járjuk be az ERŐS algoritmussal a 23. ábrán látható vásárló sorozat gráfját. Az algoritmus működését a 26. ábra mutatja tizenhét lépésben.





26. ábra

Először válasszuk a 3-as csúcspontot és induljunk el a 4-be (i). Itt vizsgáljuk meg a 4-es szomszédait. Egyetlen szomszédjába, az 5-ös csúcspontba vezető út támogatottsága nem felel meg az eddig megtett út minimális támogatottságának, ezért állítsuk vissza a 4-es színét fehérre, majd jöjjünk vissza a 3-as csúcspontba (iii). A 3-as csúcspontból már csak az 5-ösbe tudunk menni (színét szürkére állítjuk) (iv), ennek támogatottsága megfelelő, ezért színét váltsuk feketére és menjünk vissza a kiinduló 3-asba (v). Mivel a 3-as csúcspont minden szomszédját be tudtuk járni, ezért állítsuk ezt is feketére. Válasszunk egy új *k*

kezdő csúcspontot. Legyen ez a 2-es, a színét állítsuk szürkére (vi). A 2-es szomszédai a 3-as és a 4-es csúcspontok. Menjünk először a 3-as csúcspontba (vii). Mivel ennek a színe fekete, ezért nézzük meg az L_k halmazban a 3-asból kiinduló lehetséges utakat és ezek támogatottságait. A 4-es felé vezető út megfelelő, ezért vegyük fel az $\langle 2 \quad 3 \quad 4 \rangle$ utat az L_k halmazba és lépjünk vissza a 2-es csúcspontba. Az keresés további lépéseit az ábrán követhetjük nyomon.

4. A kapott eredmények, az algoritmusok további elemzése

A bemutatott algoritmusokat bemeneteik méretével és futási idejével hasonlítjuk össze.

4.1. Az algoritmusok bemenetének méretei

Mind az Apriori algoritmusok, mind az irányított gráf algoritmusok a transzformációs adatbázisból indulnak el, de lényeges különbség a két algoritmus család között, hogy míg az Apriori algoritmusok sorozatonként ábrázolják az vásárlói sorozatokat, addig a gráf algoritmusok vásárlói gráfokat építenek fel.

Két módszert szokás használni egy $G = (V, E)$ gráf ábrázolására: az egyikben szomszédsági listákkal, a másikban szomszédsági mátrixszal adjuk meg a gráfot. A szomszédsági listákon alapuló reprezentációt akkor érdemes használnunk, ha az élek száma sokkal kisebb, mint a csúcspontok négyzete. A szomszédsági mátrixos prezentáció akkor előnyösebb, ha gyorsan el kell döntenünk, hogy két csúcsot összeköt-e él, vagy az élek száma közelít a csúcspontok négyzetéhez. A szomszédsági listás ábrázoláshoz szükséges tárterület mérete $O(\max(V, E)) = O(V + E)$, mivel a szomszédsági listák hosszainak összege az élek számával egyenlő és egy (u, v) élt úgy ábrázolunk, hogy v-t felvesszük a $Szomszéd[u]$ listába. Ennek az ábrázolásnak azonban hátránya, hogy nehéz eldönteni, egy (u, v) él szerepel-e a gráfban, hiszen ehhez a $Szomszéd[u]$ szomszédsági listában kell v-t keresni. Ez a hátrány kiküszöbölhető szomszédsági mátrix használatával, ez azonban növeli a szükséges tárterület méretét, mivel ez $|V|$ csúcsszámok esetén $\Theta(V^2)$ tárterületet foglal le, függetlenül a gráf éleinek számától.

Mindkét ábrázolás esetén fel kell tüntetnünk az éleket támogató vásárlók azonosítóját, melyet szomszédsági mátrixoknál könnyen megtehetünk, ha a

mátrix *u* sorába és *v* oszlopába írjuk, szomszédsági listák esetén pedig a *v* csúcs mellett tároljuk *u* szomszédsági listájában.

A sorozatokat tároló transzfomációs adatbázis nem végzi el a fenti rendszerezést, csak bemásolja a transzformált vásárlói sorozatokat az adatbázisba [5]. Ezt az előnyt azonban a mintázatok keresésénél sokszorosan fizeti meg végrehajtási idővel, amit az AprioriSome, a mintázatok keresésének lecsökkentésével az AprioriAll-hoz képest mérsékelni tud.

4.2. Az algoritmusok futási ideje

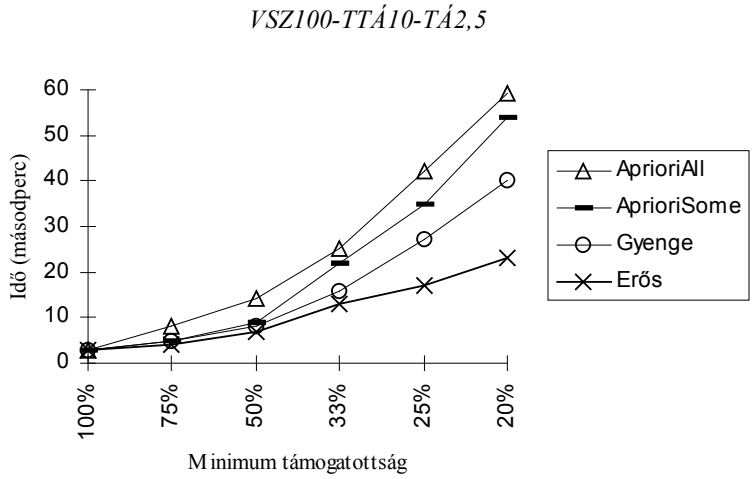
Az Apriori és a gráf algoritmusok közötti leglényegesebb különbség, hogy míg az Apriori algoritmusok olyan jelölt mintázatokat generálhatnak, aminek a támogatása nem ér el egy megadott értéket, addig a gráf algoritmusok ilyeneket nem kapnak, mivel csak azokon az éleken járják be a gráfot, amelyeknek ez a támogatottsága megvan. Így minden egyes új hosszúságú jelölt állításnál az ideiglenes tároló méretét növeli (ami nagy adatbázisnál jelentős lehet), míg a gráfoknál nem.
Vizsgáljuk meg a gráf algoritmusok futási idejét. A GYENGE algoritmus 1-2. és 3-7. sorainak, valamint az ERŐS algoritmus 1-5. és 6-11. sorainak futási ideje $\Theta(V)$, ha eltekintünk a meghívott GYENGE-BEJÁR, illetve ERŐS-BEJÁR eljárás végrehajtásához szükséges időtől. Egy optimális kezdő csúcspont választásnál az ERŐS-BEJÁR eljárást pontosan egyszer hívjuk meg minden egyes $v \in V$ csúcsra, hiszen csak fehér csúcsra hívható meg, és az eljárás elsőként szürkére festi a csúcsot. (Ha a kezdő csúcs választása optimális volt, akkor az eljárás ezt nem színezi vissza fehérre.) Az ERŐS-BEJÁR(*v*) egyszeri végrehajtása során a 2-40. sorok, illetve a GYENGE-BEJÁR(*v*) 1-12. sorok ciklus iterációszáma: $|Szomszéd[v]|$. Ugyanakkor $\sum_{v \in V} |Szomszéd[v]| = \Theta(E)$, így ERŐS-BEJÁR 2-40. sorai végrehajtásához felhasznált összidő $\Theta(E)$. Ezért az ERŐS algoritmus futási ideje (a fent említett csúcspont választásnál) $\Theta(V + E)$. A GYENGE algoritmusnál ez a futási idő annyiban változik, hogy itt is minden csúcs szomszédját megvizsgáljuk legalább egyszer, viszont a támogatottságtól függően ezt a vizsgálatot többször is elvégezhetjük. Ennek száma legrosszabb esetben $|V|$ lehet. Hasonló a helyzet az ERŐS algoritmusnál, ha a kezdő csúcspontot nem optimálisan választottuk.

A futási időt erre a célra elkészített program segítségével is teszteltük, melyben minden algoritmust egy mesterségesen generált adatbázisra futtattunk le. A 27. ábra mutatja a program paramétereit.

$\lvert VSZ \rvert$	Vásárlók száma
$\lvert TTÁ \rvert$	Vásárló sorozatokban a tranzakcióktételek átlaga
$\lvert TÁ \rvert$	Tranzakciótételekben levő termékek átlaga

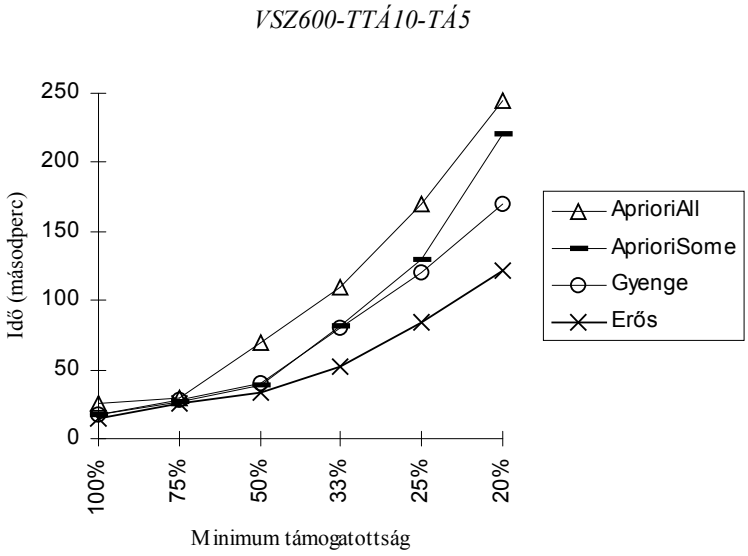
27. ábra

A futási időket a 28. és 29. ábrák mutatják. A 28. ábrán 100 vásárló esetére vizsgáltuk a futási időt, ahol átlagosan minden vásárlói sorozat 10 tranzakciótételből állt. A tranzakciótételekben átlagosan 2,5 termék volt. A vizsgálatot hat különböző vevőtámogatásra néztük meg. Az AprioriSome algoritmusnál a *next* függvényt olyan lépésközzel hívtuk meg, hogy minél jobb futási időt kapjunk. Az algoritmusok tesztelése során megállapítottuk, hogy elméletünket a gyakorlat is alátámasztotta, és a gráf algoritmusok sokkal gyorsabban végezték el a szekvencia mintázatok keresését, mint az Apriori algoritmusok.



28. ábra

A 29. ábrán már 600 vásárló esetére vizsgáltuk a futási időt, ahol átlagosan minden vásárlói sorozat 10 tranzakciótételből állt. A tranzakciótételekben itt már átlagosan 5 termék volt. Érdekes volt megfigyelni, hogy mind a gráf, mind az Apriori algoritmusok 33% vevőtámogatás alatt egyre többet dolgoztak. Sajnos a program nem tette lehetővé, de a közeljövő egyik kihívása, hogy tesztelést milliós adatrekordokra is elvégezhessük.


VSZ600-TTÁ10-TÁ5
250
200
150
100
50
0
Idő (másodperc)
100%
75%
50%
33%
25%
20%
Minimum támogatottság
AprioriAll
AprioriSome
Gyenge
Erős

29. ábra

IRODALOM

[1] M. Stonebrake et al. The DBMS research at crossroads. In *Proc. of the VLDB Conference*, Dublin, August 1993.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Washington, D.C., May 1993.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994.

[4] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In *KDD-94*: AAAI Workshop on Knowledge Discovery in Databases, July 1994.

[5] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *Proc. of the 11th International Conference*, Taipei, Taiwan, March 1995.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns: Generalizations and Performance Improvements. *Research Report RJ 9994*, IBM Almaden Research Center, San Jose, California, December 1995.

[7] IBM QUEST Data Mining Project
http://www.almaden.ibm.com/cs/quest/index.html

[8] Andásfai Béla: Gráfelmélet. Polygon, 1994.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest: Algoritmusok. Műszaki Könyvkiadó, 1997.

[10] Hajnal Péter: Gráfelmélet. Polygon, 1997.