UNIVERSITATEA "POLITEHNICA" DIN BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

REFERAT
COMPLEMENTE DE INFORMATICA
ALGORITM PENTRU
ENUMERAREA SETURILOR FRECVENTE MAXIMAL FOLOSIND DUALIZARE NEREDUNDANTA

Studenţi	**ROŞU-COJOCARU Andrei**
Seria	**C3**
Grupa	**352**

-2005-

CUPRINS

CAPITOLUL 1 – DESCRIEREA ALGORITMULUI

Rezumat

În cadrul acestui referat se va descrie o implementare a unui algoritm pentru enumerarea tuturor seturilor frecvente maximal utilizând dualizare neredundantă care reprezintă o versiune îmbunătăită a algoritmului Gunopulos şi colaboratorii [resursa1]. Algoritmul Gunopulos şi colaboratorii rezolvă multe din problemele de dualizare însă timpul computaţional este foarte mare. S-a intercalat dualizarea cu algoritmul principal şi s-a redus timpul computaţional pentru dualizare cu până la o dualizare. Astfel se reduce şi complexitatea în spaţiu a algoritmului. Mai mult, s-au accelerat prelucrările utilizând densităţi reduse.

1. Introducere

Fie E un set de itemi şi fie T un set de tranzacţii pe E. Pentru un set de itemi $S \subseteq E$, notăm setul de tranzacţii incluzând setul de itemi S cu $X(S)$. Definim frecvenţa lui S prin $|X(S)|$. Pentru o constantă dată α, dacă un set de itemi S satisface relaţia $|X(S)| \geq \alpha$, atunci spunem că S esre frecventă. Un set de itemi frecvent care nu este inclus în nici un alt set de itemi frecvent se spune că este maximal. Un set de itemi care nu este frecvent se spune ca este nefrecvent. Un set de itemi nefrecvent care nu nu include nici un alt set de itemi nefrecvent se spune ca este minimal.

Acest referat descrie o implementare a algoritmului pentru enumerarea tuturor seturilor de itemi frecvente maximal utilizând dualizarea prezentat în detaliu în [resursa2] Algoritmul reprezintă o versiune îmbunătăţită faţă de algoritmul Gunopulos şi colaboratorii [resursa1]. Algoritmul calculeaza seturi de itemi frecvente maximal bazându-se pe calcularea punctelor de intersecţie minime dintr-un hipergraf, calculând setul ţintă minim sau, în alte cuvinte, calculând dualizarea unei funcţii monotone. Algoritmul găseşte toate seturile de itemi minimale care nu sunt incluse in nici un set de itemi frecvent maximal obţinut prin dualizare. Dacă un set de itemi frecvent se găseşte în aceste seturi de itemi minimale, atunci algoritmul gaseşte un set de itemi frecvent maximal nou incluzând incluzând setul de itemi frecvent. În acest fel, algoritmul evită verificarea tuturor seturilor de itemi frecvente. Totuşi, acest algoritm rezolvă probleme de dualizare de multe ori, deoarece nu este rapid pentru motive practice. Mai mult, algoritmul foloseşte algoritmul de dualizare al lui Fredman si Khachiyan [resursa3] despre care se spune că este încet în practică.

S-a îmbunătăţit algoritmul în [resursa2] utilizând algoritmi de dualizare incrementală propuşi de Kavvadias şi Stavropoulos [resursa4] şi Uno [resursa5]. S-a dezvoltat un algoritm prin intercalarea dualizării cu gasirea seturilor de itemi frecvente maximal. Din punct de vedere teoretic, algoritmul rezolvă o problemă dualizare cu dimensiunea $|Bd^+|$, în care Bd^+ reprezintă setul de itemi frecvent maximal, în timp ce algoritmul Gunopulos şi colaboratorii rezolvă probleme de dualizare $|Bd^+|$ cu dimensiuni de la 1 până la $|Bd^+|$. Astfel, se reduce timpul computaţional cu un factor de ordinul $\frac{1}{|Bd^+|}$.

Pentru a reduce şi mai mult timpul computaţional, s-a utilizat algoritmul de dualizare al lui Uno [resursa5]. Timpul computaţional în cazul algoritmului lui Uno a fost determinat din punct de vedere experimental ca fiind liniar cu numărul de ieşiri în condiţiile în care este $O(|E|)$ este complexitatea pentru fiecare ieşire în parte, în timp ce algoritmul lui Kavvadias şi Stavropoulos are o complexitate de $O(|E|^2)$. Aceasta reduce timpul computaţional cu un factor de $\frac{1}{|E|}$. Mai mult, s-a adăugat o îmbunătăţire bazată pe densitatea redusă la intrare. Prin aceasta, timpul computaţional din punct de vedere experimental pentru fiecare ieşire în parte s-a redus la $O(\overline{Bd^+})$, unde $\overline{Bd^+}$ este dimensiunea medie a seturilor de itemi frecvente maximal. În concluzie, s-a redus timpul computaţional cu un factor de ordinul $\frac{\overline{Bd^+}}{|Bd^+| \times |E|^2}$ utilizând o combinare între algoritmul Gunopulos şi colaboratorii şi algoritmul lui Kavvadias şi Stavropoulos.

În continuare, se va descrie algoritmul şi rezultatul computaţional. Secţiunea 2 descrie algoritmul Gunopulos şi colaboratorii, iar secţiunea 3 descrie algoritmul implementat în cazul de faţă şi algoritmul lui Uno. Secţiunea 4 explică care sunt îmbunătăţirile folosind densitatea redusă. Experimentele computaţionale pentru instanţele FIMI'03 sunt prezentate în Capitolul 3 iar concluziile sunt prezentate în capitolul 4.

2. Enumerarea seturilor de itemi frecvente prin dualizare

În această secţiune se va descrie algoritmul lui Gunopulos şi colaboratorii. Explicaţii cu privire la acest algoritm se găsesc şi în resursa 1 însă acestea sunt oferite utilizând termeni generici. În aceasta secţiune, explicaţiile se vor referi strict la domeniul de prelucare de seturi de itemi frecvente.

Fie Bd^- setul de itemi nefrecvent minimal. Pentru o familie de subseturi H a lui E, un set ţintă HS a lui H este un set astfel încât pentru fiecare $S \in H, S \cap HS \neq \phi$. Dacă un set ţintă nu include nici un alt set ţintă, spunem că setul ţintă este minimal. Notăm mulţimea tuturor seturilor ţintă minimale a lui H cu $MHS(H)$. Notăm complementul unui subset S a lui E cu $\overline{S}$. Pentru o familie de subseturi H, notăm mulţimea $\{\overline{S} | S \in H\}$ cu $\overline{H}$.

Există o legătură foarte strânsă între seturile de itemi frecvente maximal şi seturile de itemi nefrecvente minimal prin operaţia de set ţintă minimal.

Propoziţia 1. $Bd^- = MHS(\overline{Bd^+})$

Utilizând propoziţia de mai jos, Gunopulos şi colaboratorii au propus un algoritm denumit *Dualizare şi Avansare* prezentat în Figura 1 pentru a calcula seturile de itemi frecvente maximal.

Propoziţia 2. *Fie* $Bd^+ \subseteq Bd^+$. *Atunci, pentru fiecare* $S \in MHS(\overline{Bd^+})$, *fie* $S \in Bd^-$, *fie S este frecvent (dar nu amândouă).*

Dualizare şi Avansare [resursa1]

1. $Bd^+ := \{go\_up(\phi)\}$.

2. Calculează $MHS(\overline{Bd^+})$.

3. Dacă nici un set din $MHS(\overline{Bd^+})$ *nu este frecvent, ieşirea este* $MHS(\overline{Bd^+})$.

4. Daca există un set frecvent S în $MHS(\overline{Bd^+})$, $Bd^+ := Bd^+ \cup \{go\_up(S)\}$ *şi se revine la pasul 2.*

Figura 1 : Algoritmul Dualizare şi Avansare

În algoritmul de mai sus, *go_up(S)* pentru un subset *S* al lui *E* este un set de itemi frecvent maximal care este calculat după cum urmează :

1. se selectează un element *e* din $\overline{S}$ şi se verifică frecvenţa lui $S \cup \{e\}$.
2. daca setul este frecvent, $S := S \cup \{e\}$ şi se revine la pasul 1.
3. altfel, dacă nu există nici un element *e* din $\overline{S}$ astfel încât $S \cup \{e\}$ să fie frecventă, atunci se întoarce S.

Propoziţia 3. *Numărul verificărilor de frecvenţă în algoritmul "Dualizare şi Avansare" pentru a calcula* Bd^+ *este cel mult* $|Bd^+| \cdot |Bd^-| + |Bd^+| \cdot |E|^2$.

În fapt, algoritmul Gunopulos şi colaboratorii rezolvă probleme de dualizare cu dimensiuni între 1 şi $|Bd^+|$. Deşi se poate termina dualizarea atunci când se găseşte un set de itemi frecvent maximal nou, se poate verifica fiecare set de itemi nefrecvent minimal din nou şi din nou. Acesta este unul din motivele pentru care algoritmul Gunopulos şi colaboratorii nu este rapid în practivă. În secţiunea următoare, se propune un nou algoritm obţinut prin intercalarea operaţiei *go_up* într-un algoritm de dualizare. Algoritmul practic realizază o problemă de dualizare de dimensiune $|Bd^+|$.

3. Descrierea algoritmului

Lema cheie a algoritmului este următoarea :

Lema 1. *Fie* Bd_1^+ *şi* Bd_2^+ *subseturi ale lui* Bd^+. *Dacă* $Bd_1^+ \subseteq Bd_2^+$,

$$MHS(\overline{Bd_1^+}) \cap Bd^- \subseteq MHS(\overline{Bd_2^+}) \cap Bd^-$$

Dacă se presupune că s-a găsit deja seturi ţintă minimale corespunzând lui $\overline{Bd^+}$ a unui subset Bd^+ a seturilor de itemi frecvente maximale, lema de mai sus înseamnă că dacă adăugăm o mulţime de itemi frecventă maximală la Bd^+, orice set ţintă minimal este în continuare un set de itemi nefrecvent minimal. De aceea, dacă se poate folosi un algoritm pentru a vizita fiecare set ţintă minimal bazat pe o adăugare incrementală de seturi de itemi frecvente maximal unul câte unul, nu mai este necesar să se verifice acelaşi set ţintă minimal din nou chiar dacă seturi de itemi frecvente maximal sunt găsite din nou. Algoritmii de dualizare propuşi de Kavvadias şi Stavropoulos [resursa4] şi Uno [resursa5] sunt astfel de tipuri de algoritmi. Utilizând aceşti algoritmi, se reduce numărul de verificări.

Se va prezenta în cele ce urmează algoritmul lui Uno [resursa5] care este o versiune îmbunătăţită a algoritmului Kavvadias şi Stavropoulos [resursa4]. Se vor introduce în cele ce urmează câteva notaţii. Un set de itemi $S \in H$ este denumit critic pentru $e \in hs$, dacă $S \cap hs = \{e\}$. Se va nota o familie de seturi de itemi critice pentru *e* cu *hs* şi *H* cu *crit(e,hs)*.

De notat este faptul că pentru fiecare $e \in mhs$, *crit(e,mhs)* nu este vidă. Se presupune că *H={S1,...,Sm}*, şi fie MHS_i ca fiind $MHS(\{S_0,...,S_i\})(1 \le i \le n)$. Pur şi simplu se va nota *MHS(H)* cu *MHS*. Un set ţintă pentru $\{S_1,...,S_i\}$ este minimal dacă şi numai dacă $crit(e,hs) \cap \{S_1,...,S_i\} \neq \phi$ pentru orice $e \in hs$.

Lema 2. *Pentru orice* $mhs \in MHS_i\ (1 \le i \le n)$*, există doar un set ţintă minimal* $mhs' \in MHS_{i-1}$ *care să satisfacă oricare din următoarele condiţii (dar nu amândouă) :*

- *mhs'=mhs.*
- *mhs'=mhs\{e} unde* $crit(e,mhs) \cap \{S_0,...,S_i\} = \{S_i\}$.

Se denumeşte *mhs'* părintele lui *mhs* şi *mhs* un copil al lui *mhs'*. Din moment ce relaţia părinte-copil nu este ciclică, reprezentarea ei grafică formează o pădure în care fiecare din componentele ei conectate este un compac înrădăcinat la setul ţintă minimal al lui MHS_1. Se vor considera copacii ca drumurile transversale definite pentru toate seturile ţintă minimale ale tuturor mulţimilor MHS_i. Aceste drumuri transversale pot fi urmărite într-o manieră care utilizează căutarea în adâncime generând copiii setului ţintă minimal vizitat la pasul curent, din moment ce se pot enumera toate seturile ţinta minimale ale lui *MHS* în timp liniar direct proporţional cu $\sum_i |MHS_i|$. Deşi $\sum_i |MHS_i|$ poate fi exponenţial faţă de |*MHS*|, aceste cazuri se aşteaptă să fie în practică excepţii. Din punct de vedere experimental $\sum_i |MHS_i|$ este proporţional cu |*MHS*|.

Pentru a găsi copiii unui set ţintă minimal, se utilizează următoarea propoziţie care derivă din lema de mai sus.

Propoziţia 4. *Orice copil mhs' al lui* $mhs \in MHS_i$ *satisface una dintre următoarele condiţii :*

(1) mhs'=mhs

(2) $mhs' = mhs \cup \{e\}$

În particular, nici un mhs nu are un copil care să satisfacă condiţia (1) şi un copil care să satisfacă condiţia (2).

Dacă $mhs \cap S_{i+1} \neq \phi$ atunci $mhs \in MHS_{i+1}$, şi condiţia (1) ţine. Dacă $mhs \cap S_{i+1} = \phi$, atunci $mhs \notin MHS_{i+1}$, şi condiţia (2) poate ţine pentru anumite $e \in S_{i+1}$. Dacă $mhs' = mhs \cup \{e\}$ este un copil al lui *mhs*, atunci pentru orice $e' \in mhs$, există un set de itemi $S_j \in crit(e', mhs)$, $j \le i$ astfel încât $e \notin S_j$.

Pornind de la aceste observaţii, se obţine algoritmul descris în Figura 2.

```
global S_0,...,S_m;
calculeaza_mhs(i,mhs) /* mhs este un set ţintă minimal pentru S_0,...,S_i */
begin
1 if i==m then ieşirea este mhs şi return;
2 else if S_{i+1} ∩ mhs ≠ φ then calculeaza_mhs(i+1,mhs);
  else
  begin
3 for fiecare e ∈ S_{i+1}
4     if pentru fiecare e' ∈ mhs , există un S_j ∈ crit(e', mhs), j ≤ i astfel încât S_j nu conţine e then
5         calculeaza_mhs(i+1, mhs ∪ {e});
  end
  return;
end
```

Figura 2 : Algoritm pentru Enumerarea Seturilor Ţintă Minimale

O iteraţie pentru algoritmul din Figura 2 consumă :

- $O(|mhs|)$ pentru linia 1.
- $O(|S_i \cup mhs|)$ pentru linia 2.
- $O\left((|E|-|mhs|)\times \sum_{e'\in mhs} |crit(e', mhs) \cap \{S_0,...,S_i\}|\right)$ pentru liniile de la 3 la 5, cu excepţia calculului pentru *crit.*

Pentru a calcula *crit* rapid, se va memora *crit(e,mhs)* în memorie şi se vor actualiza atunci când se va genera un apel recursiv. Trebuie reţinut că aceasta va utiliza O(m) memorie. Din moment ce $crit(e', mhs \cup \{e\})$ este obţinut din *crit(e',mhs)* prin înlăturarea seturilor care îl includ pe e (de exemplu, $crit(e', mhs \cup \{e\}) = \{S \mid S \in crit(e', mhs), e' \notin S_{i+1}\}$), $crit(e', mhs \cup \{e\})$ pentru toate *e'* poate fi calculat într-un timp O(*m*). Aşadar timpul computaţional al unei iteraţii este marginit superior de către $O(|E| \times m)$.

Pe baza acestui algoritm de dualizare, s-a dezvoltat un algoritm de enumerare a seturilor de itemi frecvente maximal. Initial, algoritmul consideră intrarea *H* a problemei de dualizare la setul vid. Apoi, algoritmul rezolva dualizarea în acelaşi mod ca algoritmul de mai sus. Când un set ţintă minimal *mhs* este găsit, algoritmul îi verifică frecvenţa. Dacă *mhs* este frecvent, algoritmul găseşte un set de itemi frecvent maximal S care îl include, şi îl adaugă pe $\overline{S}$ la *H* ca un element nou al lui *H*. Acum *mhs* nu este un set ţintă minimal din moment ce $\overline{S} \cap mhs = \phi$. Algoritmul continuă să genereze un apel recursiv pentru a găsi un set ţintă minimal a lui *H* actualizat. În cazul în care *mhs* nu este frecvent, din Lema 1, *mhs* continuă să fie setul ţintă minimal chiar şi atunci când *H* este actualizat. Aşadar, se utilizează algoritmul backtracking pentru a se găsi alte seturi ţintă minimale.

Când algoritmul se termină, $\overline{H}$ este setul de itemi frecvente maximal şi mulţimea tuturor seturilor ţintă minimale pe care algoritmul a găsit-o este mulţimea seturilor de itemi nefrecvente minimal. Arborele recursiv pe care algoritmul l-a generat este un subarbore a arborelui recursiv obţinut prin algoritmul de dualizare al lui Uno care primeşte la intrare $\overline{Bd^+}$ care este mulţimea complementelor mulţimilor de itemi frecvente maximal.

Algoritmul descris în Figura 3 se numeşte *Enumerator de Margini Neredundante*.

Enumerator de Margini Neredundante
global întreg *bdpnum*; mulţimi bd_0^+, bd_1^+...;
main()
begin
bdpnum:=0;
construieşte_bdp(0,mulţimea_vidă);
ieşirea sunt toate bd_j^+ ($0 \le j \le bdpnum$);
end
construieşte_bdp(i,mhs)
begin
if *i==bdpnum* /* set ţintă minimal pentru $\bigcup_{j:=0}^{bdpnum} \overline{bd_j^+}$ este găsit */
then goto 1 **else goto** 2
1. **if** *mhs* nu este frecvent, **return**; /* element nou din Bd^- este găsit */
$bd_{bdpnum}^+ := go\_up2(mhs)$; /* element nou din Bd^+ este găsit */
$bdpnum := bdpnum + 1$; /* se trece la pasul 2 */
2. **if** $\overline{bd_i^+} \cap mhs \neq \phi$ **then** *construieşte_bdp(i+1,mhs);*
else
begin
for fiecare $e \in \overline{bd_i^+}$ **do**
if $bd_i^+ \cup \{e\}$ este un set ţintă minimal pentru $\{\overline{bd_0^+}, \overline{bd_1^+}, ..., \overline{bd_{i-1}^+}\}$
then *construieşte_bdp(i+1,* $mhs \cup \{e\}$*);*
return;
end

Figura 3 : Algoritm pentru a verifica fiecare Set Ţintă Minimal doar o dată

Teorema 1. *Timpul computaţional pentru IBE este* $O(dualizare(\overline{Bd^+}) + |Bd^+|g)$*, unde* $dualizare(\overline{Bd^+})$ *este timpul computaţional al algoritmului lui Uno pentru dualizarea lui* $\overline{Bd^+}$*, iar g este timpul computaţional pentru go_up.*

Trebuie remarcat, de asemenea, că complexitatea în spaţiu a algoritmului IBE este $O\left(\sum_{S \in Bd^+} |S|\right)$ deoarece tot ce este nevoie este de a memora Bd^+ şi odată ce un set din Bd^- este verificat nu mai este nevoie ca el să fie memorat.

Pe de altă parte, algoritmul Gunopulos şi colaboratorii sugerează o utilizare a algoritmului Fredman şi Khachiyan [resursa3] care are nevoie de un spațiu de $O\left(\sum_{S\in\left(Bd^{+}\cup Bd^{-}\right)}|S|\right)$ din moment ce algoritmul are nevoie atât de Bd^{+} cât şi de Bd^{-} în ultima etapă.

4. Utilizarea densității reduse

În această secțiune, se va realiza o îmbunătățire a fazei de dualizare a algoritmului utilizând densitatea redusă pentru *H*. În datele reale, dimensiunile seturilor de itemi frecvente maximal sunt de obicei mici. De obicei, ele sunt mărginite de o constantă. Se va utiliza această structură puțin densă pentru accelerarea algoritmului.

În primul rând, se va considera o modalitate de a reduce timpul computațional al iterațiilor. Pentru aceasta, se va analiza algoritmul descris în Figura 2. Partea de restrângere a prelucrărilor unei iterații este reprezentată de liniile de la 3 la 5 care verifică existența unui set de itemi critic $S_j \in crit(mhs, e'), j < i$ astfel încât $e \notin S_j$. Pentru a verifica această condiție pentru un item $e \notin mhs$, se va utiliza un timp $O\left(\sum_{e'\in mhs}|crit(mhs,e')|\right)$, aşadar verificarea pentru toate $e \notin mhs$ se va realiza într-un timp $O\left(\left(|E|-|mhs|\right)\times\sum_{e'\in mhs}|crit(mhs,e')|\right)$.

În schimb, se va calcula $\bigcup_{S\in crit(mhs,e)}\overline{S}$ pentru fiecare $e \in mhs$. Dacă şi numai dacă $e' \in \bigcup_{S\in crit(mhs,e)}\overline{S}$ pentru toate $e \in mhs$, e' satisface condiția "if" de la linia 4. Pentru a calcula $\bigcup_{S\in crit(mhs,e)}\overline{S}$ pentru toate $e \in mhs$ timpul de calcul va deveni $O\left(\sum_{e\in mhs}\sum_{S\in crit(mhs,e)}\left|\overline{S}\right|\right)$. În cazul algoritmului IBE, $\overline{S}$ reprezintă setul de itemi frecvent maximal din moment ce dimensiunea medie a lui $\left|\overline{S}\right|$ se aşteaptă să fie mică. Dimensiunile seturilor de itemi nefrecvente minimal nu sunt mai mari decat dimensiunea maxima a seturilor de itemi frecvente maximal si de obicei sunt mai mici decât dimensiunea medie a seturilor de itemi frecvente maximal. Aşadar, |*mhs*| este de asemenea aşteptat să fie mic.

În al doilea rând, se reduce numărul de iterații. Pentru $mhs \subseteq E$, vom defini *uncov(mhs)* ca fiind mulțimea $S \in H$ care satisfac relația $S \cap mhs = \phi$. Dacă $mhs \cap S_i \neq \phi$, iterația care are ca intrări *mhs* şi *i* nu face nimic altceva decât să genereze un apel recursiv care îl incrementează pe *i* cu 1. Aceste tipuri de iterații ar trebui să fie evitate. Numai iterațiile care execută liniile 3 până la 5 sunt esențiale. De aceea, în fiecare iterație, îl setăm pe i ca fiind indexul minim pentru *uncov(mhs)*. Ca un rezultat pentru aceasta, nu este necesar de a se executa linia 2 si numărul de iterații este redus de la $\sum_i|MHS_i|$ la $\left|\bigcup_i MHS_i\right|$. Algoritmul îmbunătățit este descris în Figura 4.

```
global S_0,...,S_m;
calculeaza_mhs(i,mhs) /* mhs este un set ţintă minimal pentru S_0,...,S_i */
begin
1 if uncov(mhs)==multimea_vida then ieşirea este mhs şi return;
2 i:= index minim pentru uncov(mhs);
3 for fiecare e ∈ mhs do
4    incrementează contorul pentru itemi în  ⋃_{S∈crit(mhs,e)} S̄   cu 1
  end
5 for fiecare e' ∉ mhs astfel încât contorul este incrementat cu |mhs| do
  /* itemii incluşi în toate  ⋃_{S∈crit(mhs,e)} S̄   */
6 calculeaza_mhs(i+1, mhs ∪ {e});
  return;
end
```

Figura 4 : Algoritmul de Dualizare Îmbunătăţit folosind Densitatea Redusă

În implementarea prezentată în Figura 4, în momentul în care se generează un apel recursiv, se alocă memorie pentru fiecare variabilă utilizată în apelul recursiv. Aşadar memoria necesară de către algoritm este în cel mai rău caz $O(|E| \times m)$. Totusi, din punct de vedere experimental, memoria necesară este întotdeauna liniară şi proporţională cu dimensiunea intrării. Trebuie remarcat faptul că se poate reduce complexitatea din punctul de vedere al memoriei în cel mai defavorabil caz prin câţiva algoritmi sofisticaţi.

CAPITOLUL 2 – DESCRIERE IMPLEMENTARE

Enumerarea seturilor frecvente maximal folosind dualizare neredundanta a fost implementată în limbajul de programare C, fiind împărţită în trei fişiere sursă :

- ibe.c – implementează partea principală a algoritmului şi anume :
- realizează verificarea pentru un set de itemi dacă este frecvent, iar în cazul în care acest setul de itemi este frecvent determină complementul setului de itemi frecvent maximal care include setul de itemi descoperit;
- realizează dualizarea în sensul în care dacă setul de itemi este infrecvent se termină, iar în cazul în care setul de itemi este frecvent, încearcă să determine un set de itemi frecvent maximal care să includă setul de itemi descoperit pe care îl consideră ca ieşire adăugându-l totodată la instanţa dualizării;
- există posibilitatea de a extinde funcţionalitatea oferită de program prin variabila FREQSET_PROBLEM astfel : prin valoarea 1 se specifică determinarea seturilor frecvente, prin valoarea 2 se specifică determinarea mulţimilor închise, prin valoarea 3 se specifică determinarea seturilor de itemi frecvente maximal (valoarea implicită a variabilei este 3);
- există posibilitatea de a controla ieşirea programului prin variabila FREQSET_OUTPUT astfel : prin valoarea 1 se specifică faptul că sunt afişate informaţiile uzuale cu privire la rularea algoritmului, prin valoarea 2 se specifica faptul că sunt afişate informaţii sumare cu privire la rularea programului, prin valoarea 3 se specifica faptul că nu se vor afişa informaţii;

- freqset_.c – conține rutinele implementate de majoritatea algoritmilor care determină mulțimi de itemi frecvente maximal

Sunt definite structuri de date, măşti de biți, macrouri şi funcții auxiliare pentru determinarea mulțimilor de itemi frecvente maximal. De observat este faptul că aceşti algoritmi se bazează pe structuri de date cum ar fi liste care reține salturi care se fac in anumite momente ("jumplist") implementate sub forma unor cozi, vectori de biti şi grafuri (reținute prin matricea de adiacențe) urmărindu-se formarea unor clici. De asemenea, de observat este faptul ca observații frecvent întâlnite sunt sortări de mai multe feluri (sortare rapidă, radix sort), realizarea de permutări

- shdual.c – conține iterațiile algoritmului în varianta îmbunătățită a algoritmului (figura 4), structurile de date utilizate fiind grafuri, vectori de biti, liste care reține salturi care se fac în anumite momente implementate sub forma unor cozi.

Fişierele sursă au fost compilate folosind compilatorul gcc sub linux versiunea 2.95.4 folosindu-se un nivel de optimizare al codului 3 si instanțiindu-se variabilele de care s-a amintit mai sus FREQSET_PROBLEM=3 şi FREQSET_OUTPUT=2. În urma compilării se obține un fişier binar fim_maximal care trebuie să primeasca ca parametri un fisier de intrare (se folosesc fişiere folosite frecvent la implementarile FIM), un prag (în teorie acest prag este de obicei notat cu *s*, în algoritm el este denumit threshold sau support) care nu se da procentual ci ca porțiune din numarul de tranzacții şi, opțional, un fişier de ieşire.

Programul ibe.c a fost modificat astfel încât să se poată determina timpul de rulare al algoritmului (la inceputul şi la sfărşitul funcției main s-au făcut apeluri start_time=time((time_t)NULL), respectiv end_time=time((time_t)NULL) diferenta dintre aceste două valori returnând timpul de rulare al algoritmului în secunde.

În fişierul de ieşire specificat ca parametru (opțional) în linia de comandă sunt specificate seturile de itemi frecvente maximal precum şi frecvența de apariție care trebuie să fie mai mare decât pragul specificat. Pe ecran (sau redirectat în fisier prin ">") se va obține timpul de rulare precum şi diverşi parametri (printre care, de o importanță deosebită este numărul de iterații care în cazul de față - FREQSET_OUTPUT=2 – nu se afişează).

CAPITOLUL 3 – EXPERIMENTE

Experimentele s-au realizat pe o maşină virtuală VMWare rulând un sistem de opeare Linux Debian cu kernel 2.4.25. Maşina virtuală are un acces la memoria fizică a calculatorului de până la 64 M, aşadar în timpul rulării algoritmilor nu s-a consumat mai mult de această valoare. Totuşi, s-a observat că în timpul rulării algoritmilor se consumă foarte mult din timpul de procesor, aşa cum apare în capturile din Task Manager din Figura 5. Maşina gazdă are un procesor AMD Athlon XP 1800+ care ruleaza la o frecvență de 1.54 GHz şi la care maşina virtuală VMWare are acces deplin. Având în vedere aceste caracteristici şi comparând rezultatele experimentelor obținute în acest caz cu rezultatele obținute în [resursa6] se poate trage concluzia că diferențele se încadrează într-o limită acceptabilă, procesorul pe care s-au realizat prelucarile în acest caz fiind mai rapid decât cel din [resursa6], limitările apărând din cauza memoriei de 64 M. Totuşi, aceste limitări nu sunt foarte mari dacă avem în vedere faptul că în [resursa6] se precizează că maximul de memorie consumată în timpul rulării programului este de 110M.

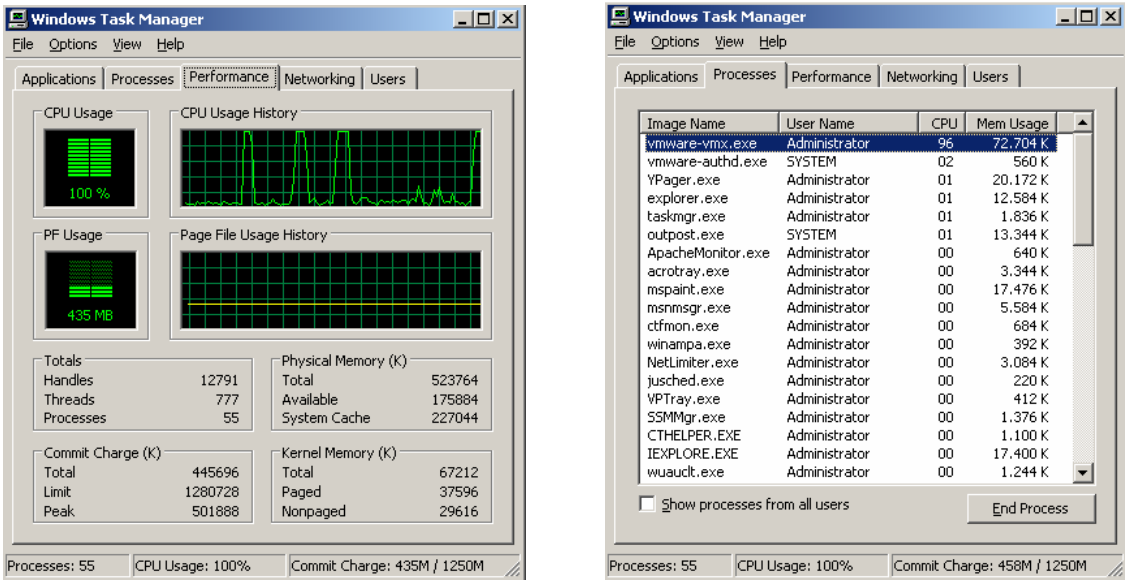


Figura 5 – Timpul de procesor şi memoria ocupată în timpul rulării algoritmului

Totuşi, valoarea de 110M s-a obținut la rularea algoritmului pentru o bază de date care nu a fost utilizată în testele realizate în acest caz.

Seturile de date folosite sunt următoarele :

baza de date	numar itemi	numar tranzacții	dimensiune medie tranzacție
T10I4D100K	1000	100000	10.0
T40I10D100K	1000	100000	39.6
pumsb	7117	49046	74.0
pumsb_star	7117	49046	50.0
chess	76	3196	37.0

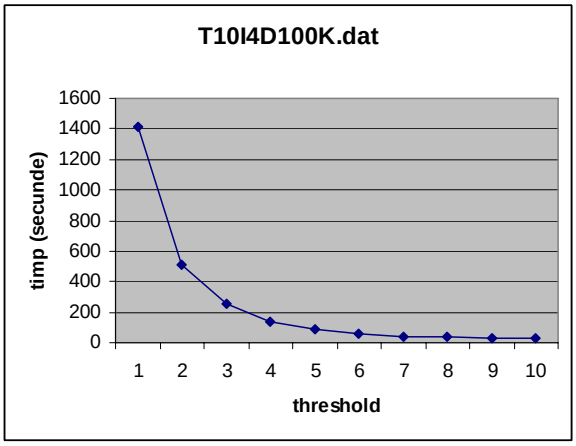
Tabel 1 Seturile de date cu caracteristicile

1. **Baza de date** T10I4D100K.dat

Pentru baza de date T10I4D100K.dat s-au obtinut următorii timpi de rulare :

threshold	timp
10	1414
20	508
30	258
40	138
50	89
60	60
70	44
80	35
90	31
100	26

Tabel 2 Timpii de executie pentru T10I4D100K.dat

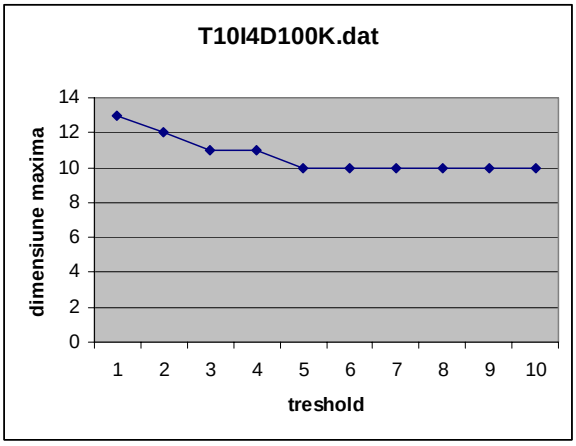


Grafic 1 Dependenta timp – treshold pentru T10I4D100K.dat

In functie de valoarea atribuita lui treshold s-au obtinut multimi cu număr variabil de itemi, asa cum reiese din tabelul urmator:

threshold	dim max a unei multimi	nr de multimi frecvente	exemple de multimi frecvente cu numar maxim de elemente
10	13	127264	16 679 165 88 652 516 486 638 161 510 883 217 529 (22)
20	12	50258	16 679 165 88 652 516 486 638 161 510 217 529 (150)
30	11	27867	764 213 753 136 348 411 54 274 21 239 32 (425)
40	11	17725	764 213 753 136 348 411 54 274 21 239 32 (425)
50	10	12062	42 269 464 922 823 515 147 820 975 196 (119)
60	10	8864	865 777 492 632 706 893 116 523 778 381 (91)
70	10	6853	958 108 486 242 8 75 71 438 684 766 (249)
80	10	5598	896 207 950 461 403 285 75 205 829 529 (258)
90	10	4679	958 108 486 242 8 75 71 438 684 766 (426)
100	10	4054	896 207 950 461 403 285 75 205 829 529 (251)

Tabel 3 Variatia multimilor frecvente in functie de min_sup



Grafic 2 Dependenta numarului maxim de elemente (pentrut mulțimi frecvente) de threshold

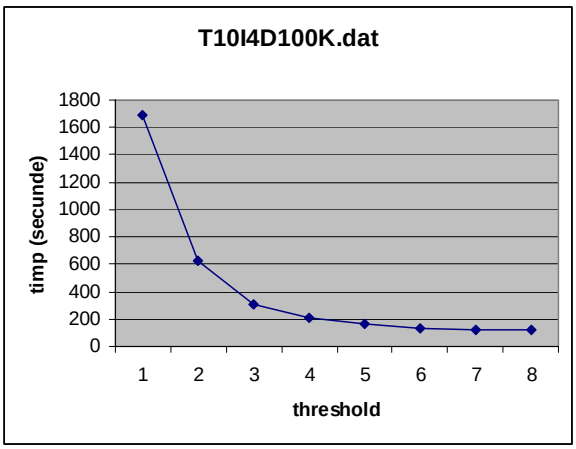
Se observă că incrementarea pragului duce la scăderea timpului de rulare însă se obțin frecvente cu dimensiuni mai mici (nu semnificative însă).

2. **Baza de date** T40I10D100K.dat

Pentru baza de date T40I10D100K.dat s-au obtinut următorii timpi de rulare :

threshold	timp
500	1689
1000	627
1500	310
2000	214
2500	163
3000	137
3500	126
4000	116

Tabel 4 Timpii de executie pentru T40I10D100K.dat

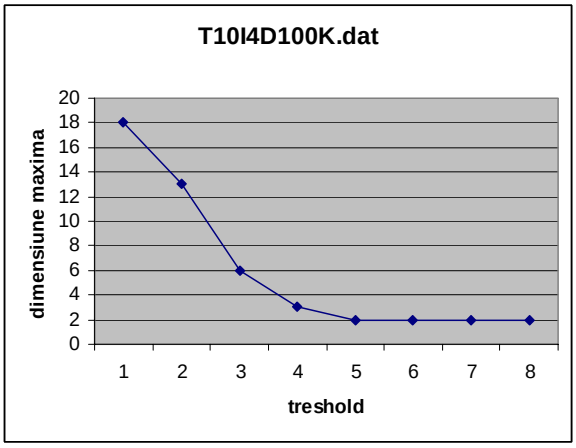


Grafic 3 Dependenta timp – treshold pentru T40I10D100K.dat

In functie de valoarea atribuita lui treshold s-au obtinut multimi cu număr variabil de itemi, asa cum reiese din tabelul urmator:

threshold	dim max a unei multimi	nr de multimi frecvente	exemple de multimi frecvente cu numar maxim de elemente
500	18	65815	683 86 309 90 705 964 640 620 745 735 793 51 43 494 120 438 510 368 (1177)
1000	13	21692	648 881 74 934 375 745 298 710 275 579 809 177 489 (1938)
1500	6	4947	680 544 876 900 618 598 (3168)
2000	3	2015	895 937 368 (4698)
2500	2	1063	450 883 (13324)
3000	2	700	675 937 (11778)
3500	2	505	177 217 (19326)
4000	2	405	438 489 (18491)

Tabel 5 Variatia multimilor frecvente in functie de threshold



Grafic 4 Dependenta numarului maxim de elemente (pentrut mulțimi frecvente) de threshold

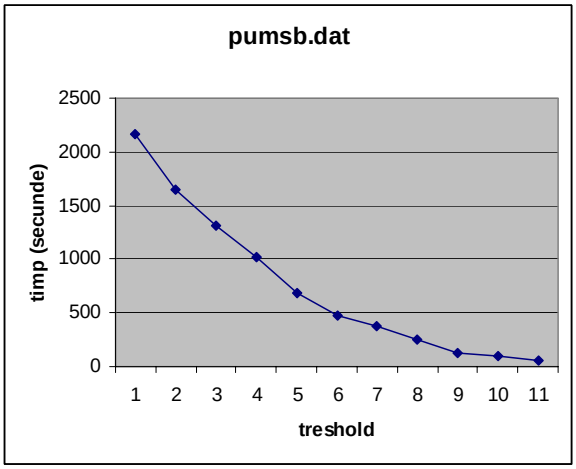
Se observă că incrementarea pragului duce la scăderea timpului de rulare însă se obțin frecvente cu dimensiuni mai mici (calitatea mulțimilor frecvente scade considerabil după cum se poate observa în grafic 4).

3. **Baza de date** pumsb.dat

Pentru baza de date pumsb.dat s-au obtinut următorii timpi de rulare :

threshold	timp
42000	2167
42500	1649
43000	1311
43500	1016
44000	691
44500	469
45000	381
45500	255
46000	126
46500	97
47000	53

Tabel 6 Timpii de executie pentru pumsb.dat

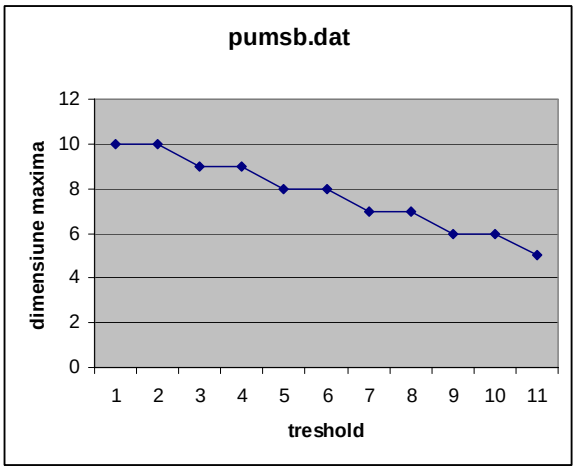


Grafic 5 Dependenta timp – treshold pentru T10I4D100K.dat

In functie de valoarea atribuita lui treshold s-au obtinut multimi cu număr variabil de itemi, asa cum reiese din tabelul urmator:

threshold	dim max a unei multimi	nr de multimi frecvente	exemple de multimi frecvente cu numar maxim de elemente
42000	10	932	7092 4940 4426 4434 4428 184 180 170 7062 4438 (42835)
42500	10	714	4940 4432 4426 4434 4428 184 180 170 7062 4438 (43576)
43000	9	541	4940 4432 4426 4428 184 180 170 7062 4438 (44252)
43500	9	396	4432 4426 4434 4428 184 180 170 7062 4438 (44475)
44000	8	288	4940 4432 4428 184 180 170 7062 4438 (44627)
44500	8	194	4432 4434 4428 184 180 170 7062 4438 (45325)
45000	7	144	4432 4434 184 180 170 7062 4438 (45689)
45500	7	99	4426 4428 184 180 170 7062 4438 (46556)
46000	6	54	4426 4428 184 180 7062 4438 (46917)
46500	6	35	4428 184 180 170 7062 4438 (47310)
47000	5	21	184 180 170 7062 4438 (47815)

Tabel 7 Variatia multimilor frecvente in functie de threshold



Grafic 6 Dependenta numarului maxim de elemente (pentrut mulțimi frecvente) de threshold

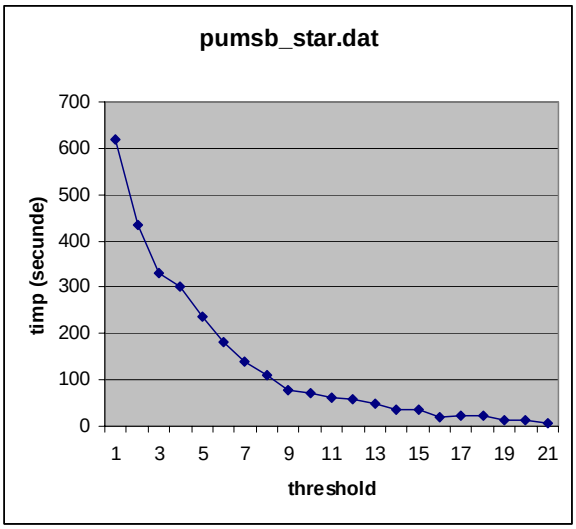
Se observă că incrementarea pragului duce la scăderea timpului de rulare însă se obțin frecvente cu dimensiuni mai mici (scăderea dimensiunii mulmțimilor frecvente se face uniform, proporțional cu incrementarea pragului).

4. **Baza de date** pumsb_star.dat

Pentru baza de date pumsb_star.dat s-au obtinut următorii timpi de rulare :

threshold	timp
10000	620
11000	434
12000	332
13000	302
14000	238
15000	181
16000	140
17000	110
18000	79
19000	70
20000	60
21000	59
22000	48
23000	36
24000	36
25000	20
26000	23
27000	24
28000	14
29000	13
30000	8

Tabel 8 Timpii de executie pentru pumsb_star.dat

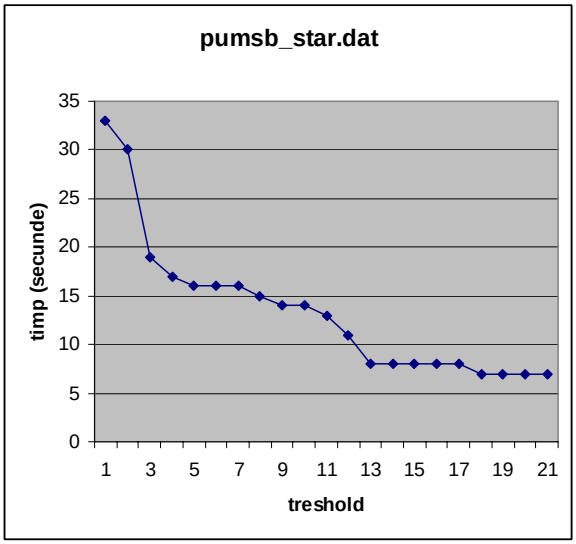


Grafic 7 Dependenta timp – treshold pentru pumsb_star.dat

In functie de valoarea atribuita lui treshold s-au obtinut multimi cu număr variabil de itemi, asa cum reiese din tabelul urmator:

threshold	dim max a unei multimi	nr de multimi frecvente	exemple de multimi frecvente cu numar maxim de elemente
10000	33	1666	4500 4497 154 4524 6866 4945 4517 4494 4491 4409 169 7032 7052 6856 5946 4953 70 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 4503 4937 4933 7072 (11119)
11000	30	1137	4500 4497 154 6866 4945 4517 4494 4491 4409 7032 7052 6856 5946 4953 70 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 4503 4937 4933 7072 (49046)
12000	19	803	7046 7026 7036 4525 4780 4680 4518 4946 6867 4493 277 4496 4937 4933 168 4499 84 4502 161 (13985)
13000	17	608	7032 7052 6856 5946 4953 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 7072 (15223)
14000	16	426	4786 4780 4680 4518 4946 6867 7057 4493 4496 4937 4933 168 4499 84 4502 161 (16840)
15000	16	315	4786 4525 4780 4680 4518 4946 6867 4493 4496 4937 4933 168 4499 84 4502 161 (19793)
16000	16	226	4525 4780 4680 4518 4946 6867 4493 277 4496 4937 4933 168 4499 84 4502 161 (18219)
17000	15	179	6856 5946 4953 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 7072 (19349)
18000	14	120	6856 5946 4953 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 (19349)
19000	14	100	6856 5946 4953 6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 (19349)
20000	13	81	4780 4680 4518 4946 6867 4496 4937 4933 168 4499 84 4502 161 (20837)
21000	11	76	6922 6869 7022 7042 4785 4727 4627 4527 4833 4807 4798 (22277)
22000	8	57	4785 4727 4627 4527 4833 4807 4798 7072 (23221)
23000	8	40	4413 4493 4496 168 4499 84 4502 161 (29349)
24000	8	38	4493 277 4496 168 4499 84 4502 161 (29384)
25000	8	17	4493 277 4496 168 4499 84 4502 161 (31053)
26000	8	21	4493 277 4496 168 4499 84 4502 161 (32200)
27000	7	17	277 4496 168 4499 84 4502 161 (30215)
28000	7	11	277 4496 168 4499 84 4502 161 (34042)
29000	7	9	4493 4496 168 4499 84 4502 161 (32200)
30000	7	4	4493 4496 168 4499 84 4502 161 (49046)

Tabel 9 Variatia multimilor frecvente in functie de threshold



Grafic 8 Dependenta numarului maxim de elemente (pentrut mulțimi frecvente) de threshold

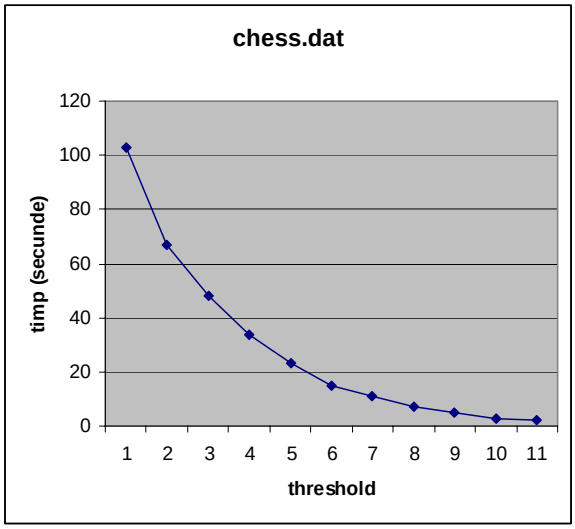
Se observă că incrementarea pragului duce la scăderea timpului de rulare însă se obțin frecvente cu dimensiuni mai mici. Se observă o diferență semnificativă între pragurile 11000 şi 12000 dimensiunea maximă a mulțimilor frecvente scăzând în acest interval cu până la 50%, de aceea, se poate considera că rezultate relevante se obțin sub pragul 11000.

5. **Baza de date** chess.dat

Pentru baza de date chess.dat s-au obtinut următorii timpi de rulare :

threshold	timp
1500	103
1600	67
1700	48
1800	34
1900	23
2000	15
2100	11
2200	8
2300	5
2400	3
2500	2

Tabel 10 Timpii de executie pentru chess.dat

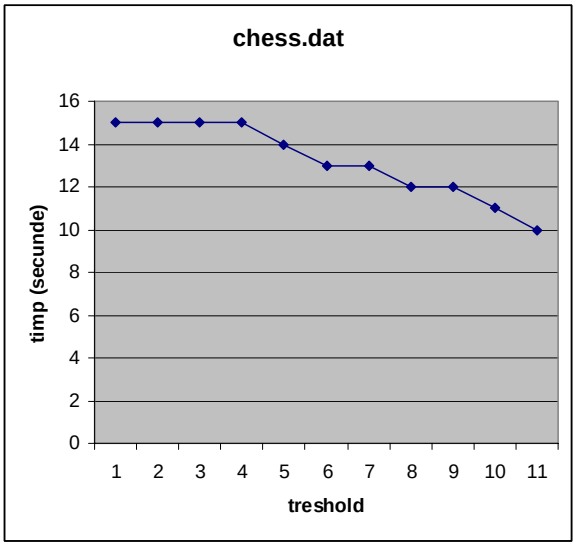


Grafic 9 Dependenta timp – treshold pentru chess.dat

In functie de valoarea atribuita lui treshold s-au obtinut multimi cu număr variabil de itemi, asa cum reiese din tabelul urmator:

threshold	dim max a unei multimi	nr de multimi frecvente	exemple de multimi frecvente cu numar maxim de elemente
1500	15	16303	23 72 31 64 42 3 48 66 7 36 60 40 29 52 58 (1745)
1600	15	11209	64 42 25 5 48 66 56 34 62 36 60 40 29 52 58 (1884)
1700	15	7837	64 42 5 48 66 56 34 62 7 36 60 40 29 52 58 (1819)
1800	15	5393	25 9 5 48 66 56 34 62 7 36 60 40 29 52 58 (2012)
1900	14	3673	3 25 5 48 66 56 34 62 36 60 40 29 52 58 (2149)
2000	13	2419	42 48 66 56 34 62 7 36 60 40 29 52 58 (2080)
2100	13	1610	25 5 48 66 56 34 62 36 60 40 29 52 58 (2322)
2200	12	799	25 48 66 56 34 7 36 60 40 29 52 58 (2361)
2300	12	708	5 48 56 34 62 7 36 60 40 29 52 58 (2517)
2400	11	478	66 56 34 62 7 36 60 40 29 52 58 (2454)
2500	10	292	5 48 66 62 36 60 40 29 52 58 (2762)

Tabel 11 Variatia multimilor frecvente in functie de threshold



Grafic 10 Dependenta numarului maxim de elemente (pentrut mulţimi frecvente) de threshold

Se observă că incrementarea pragului duce la scăderea timpului de rulare însă se obţin frecvente cu dimensiuni mai mici (nu semnificative însă). Aspectul neplăcut în acest caz este acela că scade numărul de mulţimi frecvente semnificativ începând cu pragul 1700, aşadar pentru rezultate semnificative trebuie să se aleagă un prag sub valoarea 1700.

CAPITOLUL 4 – CONCLUZII

După cum s-a constatat din datele experimentale de mai sus, pe măsură ce se scade pragul *s*, numărul de seturi de itemi frecvente maximal va scade, scăzând totodată şi dimensiunea maximă a unui set de itemi frecvent maximal. Odată cu numărul de seturi de itemi frecvente maximal şi cu şi dimensiunea maximă a unui set de itemi frecvent maximal va scade totodată şi complexitatea în timp precum şi complexitatea în spaţiu a algoritmului ceea ce nu este un aspect de neglijat. În practică trebuie să se aleagă dimensiunea pragului care satisface cel mai mult cerinţelor problemei sau să se ajungă la un compromis între calitatea seturilor de itemi frecvente maximal şi timpul de execuţie al algoritmului.

Analizând tabelele prezentate în [resursa6] putem trage concluzia că algoritmul IBE nu este foarte util dacă se doreşte determinarea de seturi de itemi frecvente maximal de dimensiune redusă şi în timp rapid, în acest sens existând algoritmi mai eficienţi cum ar fi Apriori, creştere-FP si Charm însă pe măsură ce se rafinează căutarea în sensul determinării de seturi de itemi frecvente maximal de dimensiune mare si/sau determinarea unui număr semnificativ de seturi de itemi frecvente maximal eficienţa algoritmului IBE creşte în raport cu algoritmii de data-mining amintiţi mai sus astfel încât devine mai util să îl folosim. Este interesant de observat faptul că în toate cazurile şi pe majoritatea seturilor de date algoritmul IBE este mai eficient decât algoritmul Closet care pentru anumite valori devine inpractic de rulat. De asemenea, trebuie remarcat faptul că algoritmul IBE găseşte acelaşi număr de mulţimi închise, de mulţimi frecvente, de mulţimi frecvente maximal şi de mulţimi infrecvente minimal pentru anumite valori ale pragului, aşadar nu se pierd din datele de ieşire odată cu schimbarea setului de date sau cu modificarea parametrilor de intrare.

CAPITOLUL 5 – RESURSE BIBLIOGRAFICE

1. Gunopulos D., Mannila H. şi Saluja S., "Discovering All Most Specific Sentences using Randomized Algorithms", *Proc. of ICDT '97,* pp. 215-229 (1997)
2. Satoh K., Uno. T, "Enumerating Maximal Frequent Sets using Irredundant Dualization", *Lecture Notes in Artificial Intelligence* (Proc. of Discovery Science 2003), Springer-Varlag, pp.192-201, 2003.
3. Fredman M. L. şi Khachiyan L. "On the Complexity of Dualization of Monotone Disjunctive Normal Forms", Journal of Algorithms 21 (3), pp. 618-628 (1996)
4. Kavvadias D. J. and Stavropoulos E. C. "Evaluating an Algorithm for the Transversal Hypergraph Problem", *Algorithm Engeneering*, pp. 72-84 (1999)
5. Uno T. "A Practical Fast algorithm for Enumerating Minimal Set Coverings", *SIGAL83*, Information Processing Society of Japan, pp. 9-16 (2002)
6. Uno T., Satoh K. "Detailed Description of an Algorithm for Enumeration of Maximal Frequent Sets with Irredundant Dualization", *National Institute of Informatics*, 2003
7. http://fimi.cs.helsinki.fi/
8. http://dblab.cs.ucr.edu/
9. www.cs.mu.oz.au/~jbailey/papers/hypergraph.ps
10. http://research.nii.ac.jp/~uno/papers/0311IBE.ppt