

## Bicriterial Optimization of Software

Prof. dr. Ion IVAN, prep. Cătălin BOJA  
Catedra de Informatică Economică, A.S.E. București

*There are defined two optimum criteria for software analysis. For each criterion there are defined solutions in order to reach a minimum level. There are analyzed the effects of pursuing one objective over the other one. There is developed an aggregate function for which it is determined the two criteria composed level. Based on this value it is selected the optimum solution.*

**Keywords:** software, optimization, criteria, analysis.

### 1 Criterii de optimizare

**1** *Minimizarea spațiului de memorie* ocupat de datele inițiale reprezintă un obiectiv deosebit de important în dezvoltarea de software. Pentru problema P se consideră datele inițiale  $D_1, D_2, \dots, D_n$  caracterizate prin volum, calitate și formă de stocare.

Datele inițiale sunt definite sub forma unor cuvinte de lungime  $L_i$  separate printr-un simbol special. În final datele de intrare se constituie într-un șir S de  $n$  cuvinte de lungime  $L_T$ .

$$L_T = \sum_{i=1}^n L_i + (n-1) \text{ și } Lg(S) = L_T$$

Indicatorul  $L_T$  este dat de numărul de separatori dintre cuvinte în ipoteza că două cuvinte sunt separate printr-un caracter de separare, la care se adaugă o unitate.

În memoria externă sau în memoria internă șirul S de cuvinte are o lungime  $LB_T$  exprimată ca număr de baiți.

Baza de date naturală BDN memorează în format ASCII elementele șirului S așa cum este el definit fără a se efectua modificări.

A optimiza datele de intrare revine la a minimiza spațiul de memorie exprimat ca șiruri de baiți fără a pierde informație. Se găsește un mijloc de transformare a șirului S într-un șir S' astfel încât lungimea șirului S' să corespundă relației:

$$Lg(S') \ll Lg(S), Lg(S') = L'_T$$

unde  $Lg()$  este funcția lungime a unui șir de caractere.

**Minimizarea volumului de calcule** revine la a transforma o secvență  $S_{cv}$  într-o secvență  $S'_{cv}$  astfel încât  $V(S_{cv}) > V(S'_{cv})$  unde  $V()$

este funcția de calcul a volumului de cicluri mașină necesare pentru execuția secvenței  $S_{cv}$  de instrucțiuni.

Dacă se consideră secvențele  $S_{cv}^{(0)}, S_{cv}^{(1)}, \dots, S_{cv}^{(k)}$  și dacă  $V(S_{cv}^{(0)}) > V(S_{cv}^{(1)}) > \dots > V(S_{cv}^{(k)})$

se spune că s-a derulat un proces de optimizare, secvența  $S_{cv}^{(k)}$  este cea mai bună prin comparare cu secvențele  $S_{cv}^{(0)}, S_{cv}^{(1)}, \dots, S_{cv}^{(k-1)}$

construite. Dacă se vor construi alte secvențe, procesul de evaluare pentru a alege secvența optimă se reia. Se vorbește despre un proces de optimizare parțial întrucât nu există un lanț incomplet de secvențe. În timp lanțul de secvențe se construiește și se reevaluează componentele pentru a se identifica secvența cea mai bună. Și în cazul datelor de intrare, variantele de stocare indică la un moment dat modul optim de stocare. Dacă se definesc noi metode, procesul se reia, obținând un alt mod optim de stocare a datelor.

Procesul de optimizarea a aplicației software din punctul de vedere al spațiului de stocare și a volumului de prelucrări reprezintă un ciclu finit de etape pentru că:

- producătorii nu dispun de resurse nelimitate; fiecare proiect de acest tip are un termen limită și un buget în care proiectul trebuie să se încadreze; în această situație se încearcă găsirea celei mai bune soluții care să maximizeze nivelul de calitate al aplicației și care să minimizeze necesarul de resurse;
- tehnologia hardware și software impune limite de dezvoltare; cu toate că dezvoltarea din domeniul IT are un ritm accelerat, unele dintre soluțiile existente limitează mulțimea de utilizatori prin costurile ridicate de achizi-

ție sau implementare; de exemplu aplicațiile ce permit simularea de fenomene meteorologice au grad de performanță ridicat însă sunt foarte puține locurile în care ele sunt implementate, datorită în primul rând costurilor foarte mari.

## 2. Modalități pentru minimizarea spațiului de memorie ocupat de datele de intrare

Metode utilizate în acest proces sunt:

- implementarea tipurilor de date corespunzătoare; alegerea tipului de dată corespunzător se face pe baza unei analize a mulțimii valorilor de intrare și a situațiile prelucrate de aplicația software; de exemplu implementarea unei variabile care ia valori în mulțimea  $\{0,1\}$  este definită ca fiind de tip *char* ocupând doar un octet;
- analiza și utilizarea structurilor de date adecvate; în cele mai multe cazuri se folosește mai mult spațiu decât este necesar datorită ușurinței implementării lucrului cu anumite structuri de date și datorită lipsei unei analize care să precedă implementarea și care să indice cel mai potrivit mod de a stoca datele astfel încât spațiul ocupat să fie minim;
- reutilizarea spațiului de memorie prin reinițializări de variabile diferite referite sub același nume;
- alegerea modului de stocare a datelor de intrare; de exemplu, stocarea informațiilor referitoare la mulțimea studenților dintr-o facultate se realizează prin intermediul fișierelor relative; utilizarea acestei metode prezintă avantaje pentru operațiile de citire a datelor despre un student, însă este inefficientă din punctul de vedere al minimizării spațiului de memorie pentru că multe zone de date nu sunt folosite; dacă numărul matricol al studenților are valoarea maximă 10000, și dacă poziționarea datelor se face după această valoare distinctă vor exista în fișierul respectiv tot atâtea zone de memorie de dimensiune egală cu spațiul ocupat de datele unui singur student; în cazul care mulțimea studenților activi este cu mult mai mică decât valoarea maximă a numărului matricol vor exista numeroase zone din fișier care nu sunt utilizate;
- compactarea zonei de memorie; se utilizează metode și algoritmi de compresie a da-

telor; cu toate că se obține un spațiu de memorie ocupat minimizat, procesul are efecte negative asupra volumului de prelucrare din aplicație pentru că presupune o creștere a efortului de prelucrare prin repetarea operațiilor de compresie/decompresie a datelor la fiecare citire/scriere;

- suprapunere informații pe aceeași zonă de date; limbajul de programare C are implementată facilitatea de a defini structuri de date de tip *union* în care membrii sunt definiți pe aceeași zonă de date; la nivel de date de intrare se analizează setul de valori și se definesc zone din baza de date ce conțin aceeași valoare pentru mai multe câmpuri.

## 3. Modalități pentru minimizarea volumului de prelucrare

Procesul urmărește reducerea numărului de cicluri mașină executate prin minimizarea nivelului de instrucțiuni procesate și a complexității secvențelor de cod. Prin reducerea numărului de prelucrări se scade timpul total de execuție deoarece acesta este strict dependent de numărul de cicluri mașină.

Metode de minimizare a volumului de prelucrări urmăresc optimizarea:

- codului sursă;
- algoritmilor implementați; urmărește găsirea de algoritmi de complexități reduse care să conducă la doritele dorite, dar care să genereze cât mai puține prelucrări; alegerea algoritmului se face în funcție de complexitatea acestuia, indicatorul fiind notat cu  $O(\ )$ ; de exemplu dacă se dorește căutarea unei valori într-o mulțime este mult mai eficient să se implementeze operația pe arbori binari de căutare echilibrați și atunci complexitatea este  $O(\log_2 N)$ ; dacă operația este implementată pe liste, atunci complexitatea algoritmului este  $O(N)$ , fapt care va genera mult mai multe prelucrări.

Optimizarea pe cod sursă include:

- eliminarea subexpresiilor comune; evaluarea expresiei  
float e;  
$$e = (a^2 + b^2 + c^2 + d^2) / (a^2 + b^2 + c^2 + d^2 - 1)$$
  
$$* \sqrt{(a^2 + b^2 + c^2 + d^2)}$$
;

utilizând codul:

```
float e = (a*a + b*b + c*c + d*d) / (a*a + b*b + c*c + d*d) * sqrt(a*a + b*b + c*c + d*d);
```

devine

```
float expr = (a*a + b*b + c*c + d*d);
float e = expr / expr * sqrt(expr);
```

- eliminarea invarianților:

```
for(int i=0; i<n; i++) { int a =7;
s+= x[i]; }
```

secvența devine

```
int a = 7;
for(int i=0; i<n; i++) s+= x[i];
```

- eliminarea codului mort prin transformarea secvenței:

```
bool vb = true;
if(vb) then cout<<"Mesaj: true";
else cout<<"Mesaj: false";
```

în care instrucțiunile de pe ramura *else* nu se realizează niciodată, într-o nouă secvență:

```
cout<<"Mesaj: true";
```

- reducerea expresiilor indiciale:

```
a[i][j][k] = 0;
```

prin utilizarea formelor de adresare bazate pe pointeri:

```
*(*(a+i)+j)+k)=0;
```

- regruparea structurilor de control, în care secvența:

```
for(int i=0; i<n; i++) s1 += x[i];
for(int i=0; i<n; i++) s2 = x[i]*x[i];
```

se înlocuiește cu secvența

```
for(int i=0; i<n; i++) {s1 += x[i];
s2 = x[i] * x[i];}
```

Ideea de a optimiza se concretizează prin modificări în programe care reduc volumul de prelucrări. Volumul de prelucrări,  $V$ , dat ca număr de iterații se calculează pentru secvența:

```
S = 0;
for(int i=0; i<n; i++)
S+= x[i];
```

Volumul de prelucrări  $V$ , este  $V = 3n+1$  iterații.

Îmbunătățirea secvențelor conduce la micșorarea volumului de instrucțiuni. De exemplu, pentru generarea matricei unitate:

```
for(int i=0; i<n; i++)
for(int j=0; j<m; j++)
a[i][j] = 0;
for(i=0; i<n; i++)
a[i][i] = 1;
```

care are un volum  $V_1 = 2n^2 + 3n$ .

Secvența îmbunătățită este

```
for(int i=0; i<n; i++)
for(int j=0; j<m; j++)
a[i][j] = (i == j)
```

cu un volum  $V_2 = n + n^2$

Comparând secvențele  $S_1$  și  $S_2$  rezultă că secvența  $S_2$  este mai performantă. Sunt situații în care se urmărește scăderea gradului de complexitate.

Pentru eliminarea subexpresiilor:

$$e = \frac{a*a + b*b + c*c}{a*a + b*b + c*c + d*d - 1}$$

complexitatea este:

$$C_1 = 18 * \log_2 18 + 19 * \log_2 19 = 8,14$$

Dacă înlocuim elementele subexpresiilor cu:

```
x = a*a + b*b + c*c + d*d
e = x / (x-1)
```

atunci complexitatea întregii secvențe este:

$$C_2 = 13 * \log_2 13 + 12 * \log_2 12 = 7,28$$

Inegalitatea  $C_1 > C_2$  conduce la concluzia că prin eliminarea subexpresiilor se obține scăderea complexității.

De exemplu secvența:

```
for(i=0; i<n; i++) Sx += x[i];
for(i=0; i<n; i++) Sy += y[i];
```

are volumul de prelucrări egal cu  $V_1 = 4n$

Secvența:

```
for(i=0; i<n; i++)
{
Sx += x[i];
Sy += y[i];
}
```

are volumul  $V_2 = 3n$ , iar relația  $V_1 < V_2$  conduce la concluzia că optimizarea reduce volumul de prelucrări.

#### 4. Corelația dintre volumul de date și volumul de prelucrări

Crearea bazei de date în formă naturală presupune introducerea de la tastatură a datelor.

Volumul de prelucrări, efortul, depind de operatori și aici se împarte durata în:

- *durata specifică operatorilor*, în care se analizează timpul necesar executării rutinelor de prelucrare a datelor și aducerea lor în formatul specificat; de asemenea, se cuantifică durata necesară obținerii rezultatelor intermediare și finale;

- *durata de scriere pe suport*, care are o pondere însemnată din timpul total de execuție al aplicației și este influențată de caracte-

risticile componentelor hardware; dacă se analizează o aplicație distribuită atunci se iau în calcul și parametrii transmiterii datelor în rețea.

În continuare, problematica optimizării este legată strict de ceea ce se prelucerează prin program și fără a lua în considerare ceea ce consumă operatorul. Apar și probleme de optimizare, care fac obiectul altei abordări. Se presupune că operatorii care introduc date au randament constantă, iar datele introduse sunt complete și corecte. Se construiește programul PP. Datele se introduc o singură dată în baza de date naturală, BDN și se utilizează de  $N$  ori într-un interval de timp dat,  $\Delta T$ .

*Varianta 0:* se creează BDN și se efectuează prelucrările pentru a obține rezultatele. Programul are:

*Secv1* – introducerea date operator,  $V_1$ ;

*Secv2* – validare și scriere date pe suport,  $V_2$ ;

*Secv3* – citire date de pe suport,  $V_3$ ;

*Secv4* – prelucrare date,  $V_4$ ;

*Secv5* – afișare rezultate,  $V_5$ .

unde  $V_1, V_2, V_3, V_4$  și  $V_5$  reprezintă volumul de prelucrări ca număr de cicluri mașină necesare execuției secvențelor.

Volumul total de prelucrări este  $V_{(0)}$  cicluri mașină, iar baza de date ocupă zona de memorie de lungime  $L_{BDN}$  octeți.

*Varianta 1:* se creează baza de date BDN și se produc modificări în program pentru a optimiza programul ca număr de cicluri mașină. Programul conține:

*Secv1* – introducerea date operator,  $V_1$ ;

*Secv2* – validări și scriere date,  $V_2$ ;

*Secv3* – citire date de pe suport,  $V_3$ ;

*(Secv4)<sub>modificată</sub>* – prelucrare date,  $V_4$ ;

*Secv5* – afișare rezultate,  $V_5$ .

Varianta aceasta conține un volum de prelucrări egal cu  $V_{(1)}$  cicluri mașină, iar  $V_{(0)} > V_{(1)}$ . Contribuția la reducerea ciclurilor mașină revine secvenței *Secv4<sub>modificată</sub>*.

Zona ocupată de datele de intrare rămâne neschimbată,  $L_{BDN}$ .

$$V_{(0)} = V_1 + V_2 + k*(V_3 + V_4 + V_5)$$

unde  $k$  reprezintă numărul de rulări a programului obținerea de rapoarte.

$$V_{(1)} = V_1 + V_2 + k*(V_3 + V_4 + V_5)$$

*Varianta 2:* se execută compresia de date și decompresia la fiecare execuție a programului. Se introduc secvențele:

*Secv2-0* – pentru compresie înainte de a fi scrise pe suportul de memorare; se obține un spațiu de memorie ocupat de dimensiune redusă;

*Secv3-0* – pentru decompresia datelor; după citirea datelor, acestea sunt decomprimate pentru a fi prelucrate;

*Secv5-0* – pentru compresia datelor înainte de a rescrie baza de date; odată ce au fost obținute rezultatele dorite, iar sesiunea de lucru se încheie, datele obținute sunt compresate și scrise în memorie.

Varianta 2 conduce la  $V_{(2)}$  cicluri mașină:

$$V_{(2)} = (V_1 + V_{2-0} + V_2) + k*(V_{3-0} + V_3 + V_4 + V_{5-0} + V_5)$$

Se reduce lungimea zonei de memorie ocupată de BD la  $L_{BDCompresat} < L_{BDN}$ , iar volumul de prelucrări este  $V_{(2)} > V_{(1)}$ .

Dacă  $V_2 > V_0 > V_1$  rezultă că efectul minimizării secvenței de prelucrare este depășit cu mult de efectul compresiei de date, obținând un volum de prelucrări aferent lucrului cu baza de date naturală.

Rezultă că efortul optimizării prelucrărilor este atenuat de compresia/decompresia datelor, fără a depăși volumul prelucrării cu baza de date naturală.

În cazul în care se continuă optimizarea secvențelor de prelucrare obținându-se  $V_{(3)}$  este important să se obțină inegalitatea:

$$V_{(0)} > V_{(2)} > V_{(3)}$$

adică optimizarea prelucrărilor să compenseze efectul compresiei/decompresiei.

Pentru exemplificare se consideră problema evaluării expresiei:

$$S = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n x_{ijk}$$

*Varianta 0* – preia datele de la tastatură, le validează și le scrie ca șiruri de caractere; la prelucrare se iau datele de pe suport, se convertesc spre întreg, se însumează și se afișează rezultatele; lungimea bazei de date este egală cu  $L_{BD} = 9*n^3$  pentru că fiecare valoare este memorată de către un șir cu nouă caractere; volumul de prelucrări este  $V_0$ .

$V_1$  – număr cicluri mașină executate pentru operația de citire a datelor; figura 1 exemplifică rutina de citire a datelor prin implementarea unui subprogram scris în limbajul de programare C++;

```
void citireNr(char nr[9])
{ printf("\n Introduceți numărul:");
  scanf("%5s",nr);
}
```

**Fig.1.** Subprogram pentru citirea ca șir de caractere a unui număr format din maxim 9 cifre.

$V_2$  – volum prelucrări pentru scriere date în baza de date; se consideră că fiecare număr este scris în memorie după ce a fost citit; înainte de a fi scris în baza de date, șirul de caractere este validat pentru a se verifica dacă reprezintă cifrele unui număr sau nu; figura 2 descrie prelucrările efectuate în această etapă; pentru a exemplifica, baza de date este reprezentată de un fișier binar;

```
void scriereBD(char nr[9], FILE *pBD)
{ unsigned int i;
  char flag = 0;
  for(i=0;i<strlen(nr);i++)
  { if (isalpha(nr[i]))
    { flag = 1;
      break;
    }
  }
  if (!flag) fwrite(nr,9,1,pBD);
  else printf("\n Nu este numar");
}
```

**Fig.2.** Subprogram pentru validarea unui număr scris ca șir de caractere și salvarea acestuia în baza de date.

$V_3$  – număr de cicluri mașină executate pentru citirea datelor necesare de pe suportul pe care sunt memorate; se consideră că numărul de elemente din baza de date este cunoscut și este egal cu  $n^3$ ; rutina descrisă în figura 3 construiește un masiv tridimensional pe baza datelor din fișier; odată citit, fiecare șir de caractere este convertit într-o valoare întregă;

```
int*** citireBD(int n, FILE *pBD)
{ int i,j,k;
  char nr[9];
  int ***masiv = AlocareMasiv3D(n);
```

```
for(i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
{ fread(nr,9,1,pBD);
  masiv[i][j][k] = atoi(nr);
}
return masiv;
}
```

**Fig.3.** Subprogram pentru citirea datelor din baza de date și memorarea lor sub formă de masiv tridimensional.

$V_4$  – descrie volumul de prelucrări realizate pentru efectuarea adunării prin intermediul a trei cicluri imbricate; rutina scrisă în limbajul C este descrisă în figura 4;

```
int suma(int *** masiv, int n)
{ int S=0;
  int i,j,k;
  for(i=0;i<n;i++)
  for(j=0;j<n;j++)
  for(k=0;k<n;k++)
    S = S + masiv[i][j][k];
  return S;
}
```

**Fig.4.** Subprogram pentru determinarea sumei valorilor masivului tridimensional.

$V_5$  – scrierea datelor rezultatelor pe suport; prelucrările sunt descrise în subprogramul din figura 5.

```
void scriereBD2(int *** masiv, int n, FILE *pBD)
{ int i,j,k;
  char nr[9];
  for(i=0;i<n;i++)
  for(j=0;j<n;j++)
  for(k=0;k<n;k++)
  { itoa(masiv[i][j][k],nr,10);
    fwrite(nr,9,1,pBD);
  }
}
```

**Fig.5.** Rutină pentru scrierea masivului tridimensional în baza de date.

În cazul în care se procedează la ameliorarea procesului de prelucrare se obțin volumele de prelucrări din varianta 1.

*Varianta 1* – preia datele de la tastatură ca șiruri de caractere, le validează și face conversia spre întreg; lungimea bazei de date este în acest caz  $L_{BD} = 4 \cdot n^3$  pentru că fiecare valoare este de tip întreg și ocupă patru baiți, zona fiind suficientă pentru a stoca o valoare numerică cu maxim zece cifre; în această si-

tuatie volumul de prelucrări este dat de:

$V_1$  – număr cicluri mașină executate pentru operația de citire a datelor;

$V_2$  – volum prelucrări pentru scriere date în baza de date;

$V_{2-0}$  – număr de cicluri mașină necesare pentru conversia șirului de caractere la o valoare întreagă înainte de a fi scris; figura 6 descrie instrucțiunile asociate volumului de prelucrări  $V_2 + V_{2-0}$ ;

```
void scriereBD_v1(char nr[9], FILE
*pBD)
{ int valoare;
  unsigned int i;
  char flag = 0;
  for(i=0;i<strlen(nr);i++)
  { if (isalpha(nr[i]))
    { flag = 1;
      break;
    }
  }
  if (!flag)
  { valoare = atoi(nr);
    fwrite(&valoare,sizeof(int),1,pBD);
  }
  else printf("\n Nu este numar");
}
```

**Fig.6.** Subprogram pentru validarea unui număr scris ca șir de caractere și salvarea acestuia în baza de date ca valoare întreagă.

$V_{3-0}$  – pentru citire masiv tridimensional; rutina este optimizată, deoarece nu mai execută conversia numărului din șir de caractere la valoare întreagă;

$V_4$  – descrie volumul de prelucrări realizate pentru efectuarea adunării;

$V_{5-0}$  – scriere date în baza de date; de asemenea se elimină transformarea valorii numerice în reprezentare pe șir de caractere.

Volumul total este

$$V_1 = V_1 + V_2 + V_{2-0} + k*(V_{3-0} + V_4 + V_{5-0}).$$

Efecte optimizării sunt date de faptul că s-a obținut reducerea spațiului de la  $9*n^3$  baiți la  $4*n^3$  baiți, unde  $n^3$  reprezintă dimensiunea masivului. Minimizarea spațiului de memorare s-a obținut prin reducerea la patru a numărului de octeți pe care este scrisă valoarea întreagă.

Se verifică dacă  $V_0 > V_1$  și dacă relația este adevărată, atunci înseamnă că efectul conver-

siei e anulat de optimizarea prelucrărilor.

Dacă  $V_1 > V_0$  înseamnă că efectul optimizării prelucrărilor este mai slab decât volumul de prelucrări introdus în conversiile datelor.

Se observă că minimizarea spațiului de memorie e însoțită de introducerea de secvențe de prelucrare și deci de creștere a volumului de prelucrări. Din alt punct de vedere minimizarea prelucrărilor are ca efect scăderea volumului de cicluri mașină.

Efectele sunt în cele mai multe cazuri contradictorii. Cele două criterii nu sunt complementare iar procesele ce conduc la minimizarea nivelurilor lor se influențează reciproc bazându-se pe măsuri ce scad nivelul unei caracteristici și implicit determină creșterea valorii pentru cealaltă caracteristică.

În cazul în care se dorește minimizarea volumului de prelucrări prin controlul redundanței, crește spațiul de memorie rezervat de aplicație în vederea efectuării prelucrărilor.

Existența unui unic loc unde se află o informație impune repetarea operației de citire a acelui articol ori de câte ori este nevoie de informație. Minimizarea redundanței conduce la creșterea volumului de operații de citire.

Pentru minimizarea prelucrărilor, anumite informații trebuie să fie memorate în mai multe locuri, ceea ce conduce la creșterea lungimii zonei de memorare. Mai mult, inițializările presupun efectuarea de operații în toate punctele, nu într-un singur punct, fapt ce generează creșterea volumului de prelucrări.

## 5. Minimizarea costului total

Pentru implementarea unui program apar cheltuielile:

$C_b$  – utilizare hardware care includ prelucrările pe  $10^a$  cicluri mașină;

$C_m$  – stocarea un interval  $\Delta T$  pe parcursul căroră se execută cele  $k$  rulări ale programului, repartizate pe  $x$  Mb.

Costul total  $CT_0$  al programului PP pentru varianta 0 este:

$$CT_0 = C_m * L_{BDN} + C_b * k*(V_1 + V_2 + V_3 + V_4 + V_5)$$

Costul total  $CT_1$  al programului PP pentru varianta 1 este:

$$CT_1 = C_m * L_{\text{conversie}} + C_b * k * (V_1 + V_{2-0} + V_3 + V_{4-0} + V_5)$$

Se verifică dacă  $CT_0 > CT_1$ , atunci optimizarea are efect pozitiv. Dacă  $CT_1 > CT_0$  rezultă că optimizarea utilizării memoriei elimină efectele optimizării prelucrărilor.

Întrucât practica diferențiază importanța criteriilor s-a trecut la stabilirea unor coeficienți de importanță pentru cele două criterii. Folosind un eșantion reprezentativ format din 40 de programatori cu experiență în limbajele de programare Java și C++ s-a obținut coeficientul de importanță  $p = 0,46$  pentru optimizarea spațiului de memorie și  $q = 0,54$  coeficientul de importanță pentru optimizarea volumului prelucrărilor.

În acest context:

$$CT'_1 = p * C_m * L_{\text{BDCompresat}} + q * C_b * k * (V_1 + V_{2-0} + V_3 + V_{4-0} + V_5)$$

$$CT'_0 = p * C_m * L_{\text{BDT}} + q * C_b * k * (V_1 + V_2 + V_3 + V_4 + V_5)$$

De exemplu, dacă pentru datele analizate se iau în considerare valorile:

$$p = 0,46; q = 0,54;$$

$$L_{\text{BDT}} = 1000;$$

$$(V_1 + V_{2-0} + V_3 + V_{4-0} + V_5) = 5000 \text{ cicluri mașină};$$

$$C_m = 10;$$

$$C_b = 100;$$

$$L_{\text{BDComprimata}} = 200;$$

$$(V_1 + V_2 + V_3 + V_4 + V_5) = 1500 \text{ cicluri mașină};$$

se obține:

$$CT_0 = 0,46 * 10 * 1000 + 0,54 * 100 * 5000 = 4600 + 270000 = 274600 \text{ unități volum}$$

$$CT_1 = 0,46 * 10 * 200 + 0,54 * 100 * 1500 = 920 + 81000 = 81920 \text{ unități volum}$$

Deci  $CT_0 \gg CT_1$

Fără a aplica cele două ponderi, se obțin pe indicatorii  $CT_0$  și  $CT_1$  valorile:

$$CT_0 = 10000 + 500000 = 510000 \text{ unități volum}$$

$$CT_1 = 2000 + 150000 = 157000 \text{ unități volum}$$

Deci  $CT_0 \gg CT_1$ .

Un alt caz este cel în care se înregistrează valorile:

$$C_m = 1000$$

$$L_{\text{BDN}} = 3000$$

$$C_b = 3500$$

$$(V_1 + V_2 + V_3 + V_4 + V_5) = 900 \text{ cicluri mașină}$$

$$(V_1 + V_{2-0} + V_3 + V_{4-0} + V_5) = 800 \text{ cicluri mașină}$$

$$CT_0 = 1000 * 3000 + 3500 * 900 = 6150000 \text{ unități volum}$$

$$CT'_0 = 0,46 * 3000000 + 0,54 * 3150000 =$$

$$1380000 + 1701000 = 3081000 \text{ unități volum}$$

$$CT_1 = 1000 * 3400 + 3500 * 800 = 3400000 + 2800000 = 6200000 \text{ unități volum}$$

$$CT'_1 = 0,46 * 3400000 + 0,54 * 2800000 =$$

$$1564000 + 1512000 = 3076000 \text{ unități volum}$$

Minimizarea costului total vrea să realizeze o agregare a celor două funcții de minimizarea lungimii memoriei și minimizarea volumului operațiilor de prelucrare prin intermediul costurilor totale:

$$CT = CM + CP$$

unde:

$CM$  – cost stocare în memorie;

$CP$  – cost prelucrare.

și prin costul total modificat aplicând ponderile date de coeficienții de importanță:

$$CT' = p * CN + q * CP$$

Acestea oferind o imagine mai bună care ține seama de percepția utilizatorilor a celor două laturi: capacitatea de memorare și viteza de prelucrare.

## 6. Concluzii

Luarea în considerare a două criterii reprezentă o etapă importantă pentru optimizarea multicriterială a software.

Celor două criterii alese permit măsurarea riguroasă a efectelor ca modificări a lungimii zonelor de memorie, respectiv reducerea numărului de cicluri mașină.

Dacă se consideră metodele de reducere a volumului de date  $MD_1, MD_2, \dots, MD_n$  și metode de reducere a volumului de prelucrări  $MP_1, MP_2, \dots, MP_m$  rezultă  $m * n$  variante de combinare a efectelor, pentru fiecare pereche

( $MD_i$ ,  $MP_j$ ) măsurându-se costul total  $CT_{ij}$ , respectiv, costul total corectat,  $CT'_{ij}$ .

Dacă pentru aceeași pereche ( $MD_i$ ,  $MP_j$ ) se obține relația  $CT_{ij} = CT'_{ij}$  rezultă situația stabilă în care costul economic coincide cu costul perceput de utilizatori.

Trebuie realizate măsurări sistematice pentru a identifica situațiile în care metodele nu conduc spre optimizare, rezultând eliminări de linii și de coloane din matricea formată din elementele ( $MD_i$ ,  $MP_j$ ). Astfel se îmbunătățește procesul de optimizare a produsului software prin reducerea la minimum a numărului de perechi, ( $MD_i$ ,  $MP_j$ ), pentru care se impune realizarea practică a soluției și evaluarea acesteia.

Se generează o problemă de minimizare a costului total ce include variantele dezvoltate.

### Bibliografie

[\*\*\*\*04] \*\*\*\* - *Optimization of Java*,

[http://www.factindex.com/o/op/optimization\\_of\\_java.html](http://www.factindex.com/o/op/optimization_of_java.html), 2004.

[BOJA05] Cătălin BOJA – Software Multicriterial Optimization, The Proceedings of the Seventh International Conference of Informatics in Economy, May 2005, Academy of Economic Studies, Bucharest, Romania, Infoec Printing House, pp. 1068 – 1074, ISBN 973-8360-04-8.

[CURTI99] Petru CURTICAPEAN – *Metode de cuantificare a efectelor optimizării sistemelor de programe*, Referat prezentat în cadrul pregătirii Tezei de Doctorat, ASE, București, 1999.

[DUMI04] Eugen DUMITRASCU – *Algoritmi de optimizare a produselor software*, Referat prezentat în cadrul pregătirii Tezei de Doctorat, ASE, București, 2004.

[FARL02] Jim FARLEY – *Java distributed computing*, O'Reilly Printing House, ISBN 1-56592-206-9E, electronic format.

[IEEE94] IEEE Standards Collection – *Software Engineering, Std. 1061-1992 IEEE standard for software quality metrics methodology*, Published by The Institute of Electrical and Electronics Engineers, New York, 1994.

[ISO91] ISO/IEC 9126 International Standard - *Information Technology - Software product evaluation - Quality characteristics and guidelines for their use*, 1991, Geneva, Switzerland.

[IVAN05a] Ion IVAN, Cătălin BOJA – Empirical Software Optimization, Revista Informatică Economică, vol. IX, nr. 2/2005, Infoec Printing House, Bucharest, 2005, ISSN 1453 – 1305, pp 43 – 50.

[IVAN05b] Ion IVAN, Cătălin BOJA – Global Software Optimization, pp. 205 – 214, Information Systems & Operations Management – ISOM no. 3 Workshop, Aprilie 20 – 21, 2005, Romanian American University Master of Economic Informatics, Academy of Economic Studies Master BRIE, Bucharest, ProUniversalis Printing House, ISBN 973-87166-8-3.

[IVAN05c] Ion IVAN, Cătălin BOJA – Collaborative systems components empirical software optimization effect assessment, The proceedings of The International Workshop on COLLABORATIVE SUPPORT SYSTEMS IN BUSINESS AND EDUCATION, October 28 – 29, 2005, Cluj-Napoca, Romania, Risoprint Printing House, Cluj-Napoca, pp 152 – 189, ISBN 973-651-008-9, Romania, 2005.

[IVAN04] Ion IVAN, Cătălin BOJA – *Metode Statistice în Analiza Software*, Editura ASE, București, 2004, ISBN 973-594-498-7.

[IVAN02] Ion IVAN, Paul POCATILU, Doru CAZAN – *Limbaje de asamblare*, Editura ASE, București, 2002.

[IVAN83] Ion IVAN, S. COMAN, Al. BALOG, R. ARHIRE – *Tehnici de evaluare a efectelor optimizării de programe*, Revista de statistică, nr. 3, 7, 1983.

[IVAN96] Ion IVAN, Cristian CODREANU – *Optimizarea programelor assembler*, Raport de cercetare, București, 1996.

[MOGO02] Cristina MOGOS – *Optimizarea sistemelor de programe*, Lucrare de diplomă, ASE, București, 2002.

[SCHA73] Martin SCHAEFER – *A mathematical theory of global program optimization*, Prentice-Hall, 1973.