CONTRIBUŢII PRIVIND MODELAREA ŞI OPTIMIZAREA ŞIRURILOR DE AŞTEPTARE ÎN SPITAL

Gabriel-Cristian Ene

Spitalul de urgență-Târgovişte

e-mail: enegabrielcristian@yahoo.com

Cristian Eremia

Universitatea Politehnica Bucureşti

e-mail: cristian_valentin2003@yahoo.com

Rezumat: Lucrarea prezintă rezultate ale cercetării ştiinţifice doctorale cu privire la modelarea şirurilor de aşteptare şi optimizarea timpului de ocupare a locurilor în spital de către pacienţii care aşteaptă în spital pentru a fi serviţi .

Cuvinte cheie: modelare, optimizare, şiruri de aşteptare pacienţi, servire, spitale.

Abstract: The paper presents results on the modeling and optimization of hospital waiting lists for inpatient admission.

Keywords: modeling, optimization, lines of waiting patients, hospitals.

1. Introducere

Modelarea şi simularea proceselor specifice din sistemul medical nu constituie un scop în sine, ci vizează analiza fenomenelor din sistem spre a facilita utilizarea cât mai eficace a suportului informatic de nivel înalt în adoptarea deciziilor privitoare la managementul activităţii spitaliceşti respective sau a unui fenomen, cum ar fi fenomenul riscului în oncologie, analiza supravieţuirii, etc. Optimizarea servirii unui şir de pacienţi, caracterizaţi prin durate individuale diferite a servirii lor, conduce la ideea găsirii ordinii de servire a pacienţilor care asigură un consum minim de timp pentru ocuparea locurilor din spital.

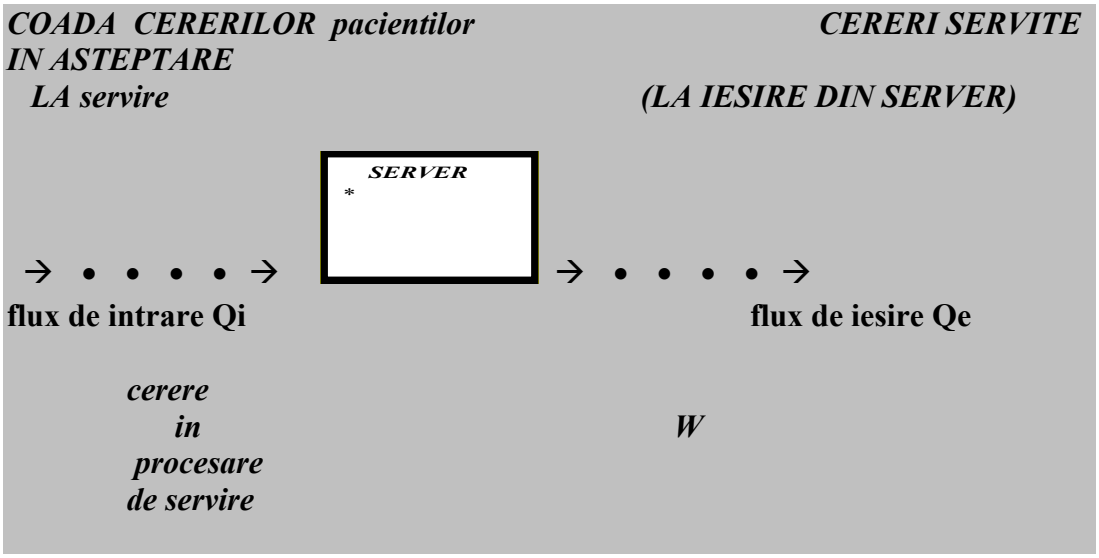

Figura 1. Sistem de servire a pacienţilor cu un singur server

În modelarea sistemelor de servire a pacienţilor prin intermediul unui singur server se presupune că la un moment dat to poate avea loc sosirea şi instalarea în coadă de aşteptare doar a unui singur pacient, urmând ca în continuare, la momente aleatoare de timp să apară, în mod similar, tot câte un singur pacient instalându-se în coadă de aşteptare (fig. 1). Elementele aleatoare în acest proces de formare a cozii sunt momentele to, t1, t2..... la care sosesc şi se instalează în coadă de aşteptare pacienţii care trebuie serviţi într-o anumită ordine (de exemplu în ordinea „primul venit→primul servit" sau orice altă ordine de servire dinainte stabilită, spre exemplu după gravitatea stării pacientului, etc.) [1]. Abstractizând, putem considera că, aceste

obiecte instalate în coadă de aşteptare pot fi de natură fizică sau informaţională şi de o mare diversitate. În abordarea sistemică a cozii de aşteptare se face abstracţie de natura fizică ori informaţională a acestor obiecte. Pentru generalitate, în continuare, aceste obiecte vor fi denumite generic „cereri” .

2. Modelul instalării în coadă a pacienţilor şi simularea servirii

Una din caracteristicile cozii de aşteptare este *fluxul de cereri.* Acest flux exprimă numărul cererilor sosite şi instalate în coadă în unitatea de timp şi reprezintă *mărimea de intrare a sistemului numit coadă de aşteptare*. Fluxul de cereri de intrare constituie o caracteristică a cozii de aşteptare şi are şi o interpretare concretă deoarece exprimă numărul de pacienţi sosiţi şi instalaţi în unitatea de timp în coadă de aşteptare. Deoarece un eveniment discret se produce de unul singur la un moment aleator considerăm fluxul suficient de mic încât, în intervalul de timp Δt, poate sosi la coadă cel mult o singură cerere [2]. Este evident că şansele ca în acest interval să apară o nouă cerere sunt cu atât mai mici cu cât intervalul de timp Δt este mai scurt. De aceea, vom presupune că, în cazul lui Δt mic, probabilitatea apariţiei unei cereri este proporţională cu lungimea acestui interval de timp adică probabilitatea acestui eveniment elementar este,

$$p = (\lambda \Delta t) \qquad (1)$$

în care $\boldsymbol{\lambda}$ este un număr pozitiv ce caracterizează densitatea (frecvenţa sosirii cererilor) şi reprezintă un parametru al modelului sistemului de servire a cererilor din coada de aşteptare. Ne propunem în continuare să stabilim care este probabilitatea ca la momentul curent de timp t numărul de cereri ***x(t)*** instalate în coadă să aibă valoarea ***k***, fixată în prealabil.

2.1 Formarea cozii

Modelul matematic pe care urmărim să-l construim trebuie să permită să se calculeze şirul de valori discrete (între 0 şi 1) ale probabilităţilor: $P(x=0,t=n\Delta t)$ ca în intervalul t „să nu apară nicio cerere”, $P(x=1,t=n\Delta t)$ ca în intervalul t „să apară o singură cerere”, $P(x=2,t=n\Delta t)$ ca în intervalul t „să apară x=2 cereri”, etc. … ca în intervalul t „să apară x=k cereri”.

La intrarea sistemului de servire (SS) sosesc mereu pacienţi pe măsura trecerii timpului t. Presupunem că intensitatea fluxului de sosire a clienţilor nu este prea mare, astfel că într-un interval mic Δt de timp nu poate sosi mai mult de un client. Evident, cu cât este mai mic intervalul Δt cu atât sunt mai puţine şanse ca în acest interval să apară un client care să se instaleze în coadă. De aceea, vom considera că probabilitatea p de apariţie a unui client (într-un interval mic de timp Δt) este proporţională cu lungimea acestui interval. Această probabilitate de apariţie p a unui client este aproximativ egală cu $p \cong \lambda \bullet \Delta t$, în care λ este un număr care caracterizează densitatea de apariţie a clienţilor (adică numărul de clienţi care apar în unitatea de timp).

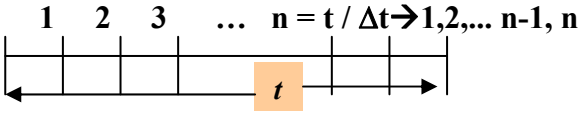

Figura 2. Divizarea intervalului de timp t în n segmente de lungime Δt

Fie u(t) numărul de clienţi la momentul t. Determinăm valoarea probabilităţii ca la momentul t numărul clienţilor să fie k. Împărţim intervalul (0,t) în segmente Δt atât de mici astfel încât într-un interval Δt să poată apărea practic cel mult un singur client (probabilitatea care prin definiţie este $\lambda \bullet \Delta t$). Se formulează acum o nouă întrebare: ”care este probabilitatea ca în cazul a n intervale $t = n \bullet \Delta t$, numărul de clienţi sosiţi la intrarea SS să fie unul singur ?”

Pentru aceasta divizăm intervalul t în n segmente ca în figurile 1 şi 2.

Dacă notăm cu e_n evenimentul apariției unui pacient în intervalul n, acest eveniment are probabilitatea $p(e_n)= \lambda \bullet \Delta t$. Notam cu $\bar{e}_n$ probabilitățile de neapariție $1-p(e_n)$. Astfel, în intervalul t are loc un şir de evenimente aleatoare independente. În modelul prezentat mai sus se are în vedere renumita regulă FIFO de servire a pacienților, în ordinea sosirii lor. În continuare se consideră cazul în care au fost stocate toate cererile sosite, după care se trece la tratarea cererilor într-o ordine astfel aleasă încât *timpul folosit* de server pentru servirea tuturor cererilor din mulțimea stocată să fie *minim.*

2.2 Simularea sistemului de servire a cererilor

Principalele ipoteze de lucru se referă la tipul distribuției probabilității intervalului de timp dintre două sosiri consecutive de clienți şi la durata de execuție a servirii unui client .

Fluxul de cereri la intrare este dat prin funcția de distribuție exponențială care este descrisă în [1]**.** *Timpul de execuție* τ_j pentru programul de rezolvare a cererii **j** de către server este o mărime aleatoare dată prin probabilitatea Pk (k= 1,2,…,s) (există s programe de prelucrare) deoarece sunt s soluționări pentru cererea j. Prin urmare avem:

$$\sum_{i=1}^{s} P_i = 1$$

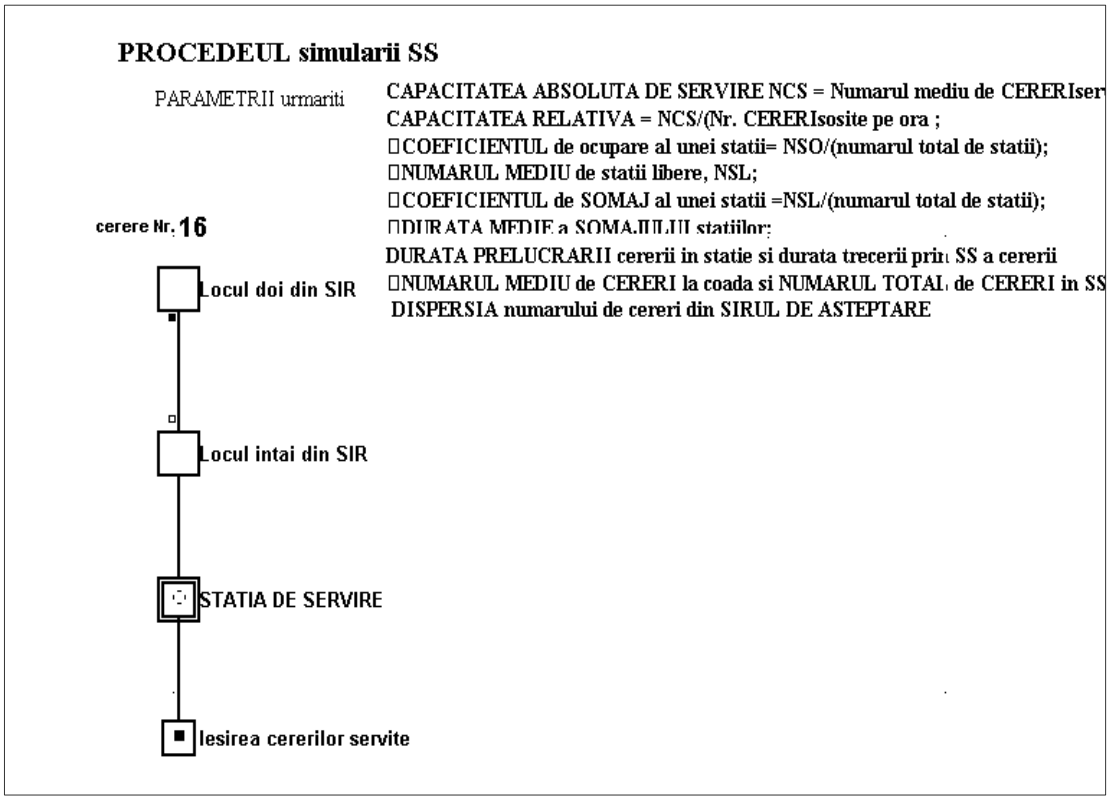

Figura 3. Parametrii sistemului de servire evaluați prin simulare

În pseudocodul algoritmului de simulare şi în descrierea algoritmului sunt utilizate următoarele notații:

tM – Timpul curent de modelare – care creşte prin incrementarea cu valori discrete

tj – momentul lansării cererii în sistem = 1, timpul de execuție al programului apelat de cererea j

τ j – timpul curent pentru programul j incrementat discret cu Δ, adică $\bar{\tau}$ j = n Δ

T – limita superioară a timpului rezervat simulării SS

r – indicele acelei cereri care este prima în şirul de aşteptare (prima din coadă)

τ *j - unde j=(1,s) timpul de execuţie al programului apelat de cererea j

■**Pas 1** ***Se generează momentul*** **t j** ***şi se determină*** τ * **j**. ***Se trece la Pas 2***
■**Pas 2** ***Se verifică*** DACA ***coada este*** GOALA:
–***Dacă*** DA ***se execută Pas 3***
–***Dacă*** NU ***se execută Pas 4***
■**Pas 3** **t M = t j**
■**Pas 4** ***Se instalează la coadă cererea*** **j** ***şi se trece la Pas 5***
■**Pas 5** ***Se verifică DACA*** **t M + $\Delta \leq$ t j :**
–***DACA*** DA, ***se execută Pas 7***
–***DACA*** NU, ***se execută Pas 6***
■**Pas 6** ***Se face*** **j = j +1** ***şi se revine la Pas 1***
■***Pas 8 Se verifică DACA t M $\leq$ t r :***
■***DACA DA, se trece la Pas 9***
■***DACA NU, se trece la Pas 10***
■***Pas 9 t M= t r se trece la Pas 10***
■***Pas 10 Se verifică DACA*** $\bar{\tau}$ ***r = 0***
■***DACA NU se trece la Pas 11***
■***DACA DA se trece la Pas 12***
■***Pas 11*** τ ***a in s??? aştept r = t M1 - t r se trece la Pas 12***
■***Pas 12*** ***Se verifică DACA*** τ ***r +*** Δ ***<*** τ **** r***
■***DACA DA se trece la Pas 13***
■***DACA NU se trece la Pas 15***
■***Pas 13*** $\bar{\tau}$ ***r =*** τ $\bar{r}$ ***+*** Δ ***şi se trece la Pas 14***
■***Pas 14*** ***t M = t M +*** Δ ***şi se trece la Pas 17***
■***Pas 15*** ***t M = t M +*** τ ****r şi se trece la Pas 16***
■***Pas 16*** τ **** raspr = t M - t r şi se trece la Pas 17***
■***Pas 17*** ***Se verifică t M < T***
■***DACA DA se trece la Pas 18***
■***DACA NU se trece la Pas 19***
■***Pas 18*** ***Refacerea cozii – se trece la Pas 5***
■***Pas 19 Afişare rezultat***
■***Pas 20 STOP***

Figura 4. Pseudocodul simulării Sistemului de servire (SS)

Algoritmul de simulare a SS, prezentat în figura 4, este compus din două etape: prima etapă (formată din paşii 1… 6) se referă la organizarea pacienților în şirul de aşteptare iar etapa a doua este destinată simulării servirii, adică rezolvării cererilor acestora [3].

În continuare sunt explicitați patru dintre cei mai importanți paşi din algoritmul de simulare a sistemului de servire de tip monocanal prezentat în figura 4.

La ***Pas 1***, algoritmul de mai sus imită momentele t j de sosire a cererilor şi timpul τ *j de execuție al programului apelat de cererea j.

La ***Pas 2*** se verifică dacă în momentul sosirii cererii j coada este goală cu scopul de a stabili care cerere este "prima" la coadă (adică stabileşte începutul cozii). În ambele situații cererea este instalată în şir la coadă (***Pas 4*** simulează instalarea).

Pasul 3 a fost introdus pentru adăugarea timpului t_M curent de modelare care creşte cu valori discrete până la momentul sosirii t j.

Pasul 5 verifică dacă în intervalul de divizare Δ nu au sosit cumva şi alte cereri (fără Pas 5 se pot pierde o parte din cereri). Astfel, paşii ***Pas1, Pas2, …Pas6*** organizează cererile la coadă.

A doua etapă din algoritmul de simulare al unui SS de tip monocanal prezentat în figura 4 conține paşii 7... 20 ai algoritmului. În continuare sunt explicitați cei mai importanți paşi dintre aceştia.

Pasul 7 alocă procedurii (programului) **"r"** un segment **Δ**.

Pasul 8 verifică dacă nu cumva timpul curent de simulare $\mathbf{t_M}$ nu a rămas în urma momentului $\mathbf{t_r}$ de apariție în sistem a cererii care este "prima" la coadă. Dacă cumva $\mathbf{t_M < t_r}$ atunci la ***Pas 9*** timpul curent al modelării este "tras" până la $\mathbf{t_r}$.

Pasul 10 verifică dacă procedurii cu numărul "**r**" i s-a alocat măcar un segment de timp **Δ**. Dacă condiția ***Pas 10*** se verifică (adică nu se obține timpul dorit) atunci se adaugă un **Δ**. Dacă **NU** se verifică condiția de la acest pas, atunci la ***Pasul 11*** se calculează durata aşteptării τ^a pentru începerea servirii cererii cu numărul **"r".**

Pasul 12 verifică dacă programul (procedura) cu numărul **"r"** apelată de cererea **"r"** are timp suficient ca în segmentul **Δ** alocat să poată termina execuția. Dacă condiția de la ***Pasul 12*** se verifică, atunci programul lansat în execuție de cererea **"r"** se termină în timp util. Dacă la ***Pasul 12*** **NU** se verifică condiția (adică timpul alocat execuției nu este suficient), atunci timpul $\bar{\tau}_r$ de execuție al programului **"r"** şi timpul curent de modelare $\mathbf{t_M}$ sunt măriți (***Paşii 13*** **şi** ***14***), iar dacă NU (deci programul reuşeşte să fie executat în intervalul **Δ)** şi la ***Pasul 15*** timpul de modelare este mărit până la valoarea realmente necesară pentru terminarea execuției programului **"r"** (acest interval poate fi mai mic ca **Δ** deoarece($\tau^*_r - \bar{\tau}_r$) ≤ **Δ**). Prin aceasta se realizează o economisire a timpului de modelare.

Pasul 16 calculează intervalul de timp τ^{rasp}_r – după care cererea deservită (**r**) obține răspunsul (rezolvarea).

Pasul 17 verifică atingerea cotei superioare **T** a intervalului de modelare. Dacă această cotă nu este atinsă, atunci se reconstruieşte coada prin deplasarea cu un loc a cererilor. Semnificația ultimilor paşi este clară din pseudocod.

3. Optimizarea servirii

3.1 Dependența dintre durata aşteptării, respectiv a ocupării serverului şi ordinea servirii cererilor

În această secțiune se are în vedere situația în care avem la dispoziție o mulțime finită de cereri W={1,2,3, . . . ,n}care pot fi servite într-o ordine convenabilă oarecare. Activitatea concretă, într-un spital, a evidențiat faptul că durata rezolvării unei cereri nu este aceeaşi pentru orice cerere din W. Se poate considera că fiecare cerere necesită pentru servire o durată diferită, dar cunoscută, de timp necesar pentru servire. Astfel, cererilor din mulțimea W le corespund duratele de prelucrare: t1, t2, t3 , ... , tn. Este evident că durată aşteptării în spital până la terminarea servirii cererii k va fi *Tk, k=1,2,3,....,n* diferită pentru fiecare cerere din W şi care depinde de ordinea în care cererile din W vor fi servite. Astfel, dacă într-o primă situație se păstrează ordinea din W, aceste durate vor fi:

$$T1(1)=t1;\ T2(1)=T1(1)+t2;\ T3(1)=T2(1)+t3;\ \ldots;Tn(1)=Tn\text{-}1(1)+tn \qquad (2)$$

Pentru această situație notată cu **1** durata procesării tuturor cererilor din W este:

$$T(1)=\sum_{k=1}^{k=n} Tk(1) \qquad (3)$$

Pentru o situație următoare notată cu **2** în care ordinea s-a schimbat prin inversarea primelor două cereri din W avem :

T1(2)=t2;T2(2)=t1+ t2 ;T3(2)=t1+ t2+ t3;...Tn(2)=t1+ t2+ t3+ , ...,+ tn-1+ tn .

Durata procesării T(2), a tuturor cererilor (care corespunde timpului până la externarea tuturor pacienților) din **W** în acest caz, este diferită de T(1):

$$T(2)=\sum_{k=1}^{k=n} Tk(2) \neq T(1)$$

Numărul total de astfel de variante (care se pot obține prin permutările cererilor din W) este **N=n!** în care **n** este cardinalul mulțimii W. Problema determinării ordinii de servire care să asigure cea mai scurtă durată de procesare a tuturor cererilor din W presupune calculul celor N durate corespunzătoare celor **n!** variante rezultate prin permutări şi găsirea valorii optime T*, prin căutarea duratei minime din mulțimea celor N= **n!** durate calculate pentru toate situațiile posibile:

$$T^* = \min[T(1), T(2), \ldots . T(k) \ldots, T(N)] \tag{4}$$

Asemenea problemă este specifică algoritmilor de ***optimizare combinatorială*** dintre care cel mai simplu şi mai potrivit, în acest caz, este un algoritm de tip *greedy* [1]. *Combinatorica* este un domeniu reprezentativ al matematicilor care, în esență, se ocupă cu studiul "aranjamentelor" obiectelor unei mulțimi finite, conform unei structuri date. În problemele combinatoriale, prioritare sunt chestiunile de existență şi de numărare:

- ***există un anumit tip particular de aranjament?;***
- ***câte asemenea aranjamente se pot forma?***

Caracteristic problemelor combinatoriale este faptul că numărul acestor aranjamente este finit. Dacă aranjamentele unei probleme combinatoriale pot fi comparate între ele, pe baza unui **criteriu de apreciere** apare o problemă de ***optimizare combinatorială în care trebuie să se obțină răspuns la întrebarea:***

Care este cel mai bun aranjament din punctul de vedere al criteriului respectiv?

Răspunsul la această întrebare adaptată problemei optimizării servirii care asigură timpul cel mai mic de ocupare a serverului este dat în secțiunea următoare.

3.2 Generarea prin permutări a variantelor ordinii servirii

Pentru implementarea pe calculator a algoritmului de rezolvare a problemei de optimizare (4) este necesară generarea celor N= n! permutări ale cererilor din mulțimea W pentru a putea calcula T(1), T(2), ... , T(N) din (4). Pentru generarea permutărilor există mai mulți algoritmi dar toți aceştia furnizează n! variante şi algoritmul de optimizare bazat pe permutări necesită un timp de cel puțin ordinul ***O(n!)***[4]**.** Pentru a ne putea imagina cum creşte complexitatea algoritmului odată cu creşterea lui *n* facem observația că dacă mulțimea W conține trei cereri, numărul permutărilor este 3!=6 iar dacă numărul de cereri creşte de circa trei ori, numărul de permutări creşte de peste un milion de ori (numărul de cereri fiind zece cereri 10! numărul permutărilor este de aproximativ 3 milioane şi jumătate). Pentru ilustrare prezentăm în continuare modul de aplicare, în minimizarea timpului de aşteptare la coadă, a unui cunoscut algoritm de optimizare combinatorială denumit ***greedy*** **(**greedy = lacom). Prezentarea algoritmului se face în următoarele condiții:

- ***Se presupune capacitatea de servire limitată (un singur bloc operator, un singur echipament de iradiere, etc.)***

- ***Se presupune că durata servirii (durata tratament t) diferă de la pacient la pacient, dar este cunoscută apriori.***
- ***Timpul total T consumat de n pacienți în cadrul spitalului este egal cu: T=timpul T1 consumat de primul pacient + T2 consumat de al 2-lea pacient + … +Tn consumat de ultimul pacient.***
- ***Schimbarea ordinii servirii, prin permutări, permite obținerea variantei de şir de aşteptare în care T este minim dar, atenție: Nr. de permutări N=n!***

Pentru generarea permutărilor există mai mulți algoritmi, dar toți aceştia necesită un timp de cel puțin ordinul ***O(n!)*** [2]**.** Pentru ilustrare prezentăm în continuare modul de aplicare, în minimizarea timpului de aşteptare la coadă, a unui cunoscut algoritm de optimizare combinatorială denumit ***greedy*** **(**greedy = lacom). Algoritmii *greedy* sunt în general simpli şi sunt folosiți la probleme de optimizare, cum ar fi: ***să se găsească cea mai bună ordine de tratare de către un server a unor cereri de complexitate diferită şi care aşteaptă să fie servite***. Pentru a rezolva această problemă de optimizare, căutam o soluție posibilă care să minimizeze valoarea funcției obiectiv. Un algoritm *greedy construieşte soluția pas cu pas*. Inițial, mulțimea candidaților selectați este vidă. *Greedy* presupune o reordonare a cererilor rezultând o nouă mulțime a celor *n cereri care aşteaptă să fie servite astfel încât durata servirii tuturor cererilor să fie minimă*. Este foarte dificilă scrierea formei generale a unei probleme rezolvabile folosind tehnica Greedy. *Tehnica Greedy conduce la timp de calcul polinomial.* Motivul care conduce la acest timp de calcul, ține de mecanismul tehnicii. Să presupunem că mulțimea din care se face alegerea are ***n*** elemente şi că soluția are tot ***n*** elemente (caz maxim). Se fac ***n*** alegeri, la fiecare alegere se fac ***n*** teste, rezultă un algoritm cu timp $\mathbf{O(n^2)}$**.** De multe ori, este necesar ca elementele mulțimii C să fie sortate, pentru ca apoi să alegem din acestea, iar sortarea necesită un timp minim $\boldsymbol{O(n \times \log_2 n)}$**.** Însă sortarea se efectuează la început. Prin urmare, acest timp se adună, deci nu influențează rezultatul. ***Dacă algoritmul greedy funcționează corect, prima soluție găsită va fi totodată o soluție optimă a problemei***. Soluția optimă nu este în mod necesar unică: se poate ca funcția obiectiv să aibă aceeaşi valoare optimă pentru mai multe soluții posibile. Algoritmul este "lacom" şi la fiecare pas, alege ***cel mai bun candidat*** la momentul respectiv, fără să-i pese de viitor. Dacă un candidat este inclus în soluție, el rămâne acolo; dacă un candidat este exclus din soluție, el nu va mai fi niciodată reconsiderat. Asemenea unui întreprinzător rudimentar care urmăreşte câştigul imediat în dauna celui de perspectivă, un algoritm greedy acționează simplist. Totuşi, ca şi în afaceri, o astfel de metodă poate da rezultate foarte bune tocmai datorită simplității ei. Un algoritm greedy nu duce deci întotdeauna la soluția optimă, sau la o soluție. Este doar un principiu general, urmând ca, pentru fiecare caz în parte, să determinăm dacă obținem sau nu soluția optimă.

Pentru ilustrarea celor prezentate se consideră un exemplu simplu care nu implică utilizarea tehnicii de calcul pentru rezolvare. În continuare sunt prezentate două exemple în care există o singură stație de servire care trebuie să satisfacă cererile a *n* clienți. Timpul de servire necesar fiecărui pacient este diferit şi cunoscut în prealabil:

- pentru pacientul *i* este necesar un timp t_i, $1 > i > n$.
- dorim să minimizăm timpul cumulat total de aşteptare în incinta spitalului de toți cei n pacienți. Acest timp cumulat este exprimat de următoarea relație: $T = \sum_{k=1}^{k=n} Tk$, în care Tk este durata aşteptării până când serverul ajunge să termine servirea pacientului *k.* Aceasta este acelaşi lucru cu minimizarea timpului mediu de aşteptare, care este T/n.

Exemplul 1, dacă avem în total trei pacienți atunci, $t_1 = 5$ este timpul, exprimat în unități convenționale, necesar rezolvării cererii primului pacient, al doilea $t_2 = 9$ unități convenționale, necesar rezolvării cererii acestuia iar pentru ultimul pacient din şir, durata servirii cererii lui este $t_3 = 3$.

t_1=5 t_2=9 t_3=3.

Ordinea	***Timpul T de asteptare***			
1 2 3	5+(5+9)+(5+9+3)	=	36	
1 3 2	5+(5+3)+(5+3+9)	=	30	
2 1 3	9+(9+5)+(9+5+3)	=	40	
2 3 1	9+(9+3)+(9+3+5)	=	41	
3 1 2	3+(3+5)+(3+5+9)	=	**28**	**← optim**
3 2 1	3+(3+9)+(3+9+5)	=	32	

Figura 5. Ordinea servirii celor 3 pacienți obținută prin permutări și compararea duratelor servirii în cele șase cazuri rezultate

Dacă pacienții sunt serviți în această ordine (ordinea inițială a sosirii) se zice că servirea se face conform regulii ***„primul sosit → primul servit”*** cunoscută și sub denumirea de **FIFO** (F*irst Input → First Output)*. În acest prim exemplu ilustrăm situația în care nu aplicăm greedy, ci căutăm soluția în mulțimea celor 6 variante obținute prin aranjamente pe mulțimea celor 3 cereri. Această procedură este ilustrată în figura 3. Din această figură rezultă că sunt posibile șase ordini de servire a clienților menționați, care au fost obținute prin permutări. În primul caz, când se aplică regula de servire FIFO, clientul k=1 este servit primul și el poate părăsi spitalul la momentul T1=5, iar clientul k=2 așteaptă până este servit clientul k=1 și apoi este servit și el, iar timpul lui de așteptare este T2=14, clientul k=3 așteaptă până sunt serviți clienții k= 1, k= 2 și apoi este servit și el, deci timpul lui de așteptare este T3=17. Timpul total de așteptare cumulat ***Tfifo*** al celor trei pacienți rezultă că suma acestor intervale de timp (prima linie din tabel): T1+T2+T3=5+14+17=36.

După cum rezultă din figura 5, ordinea cea mai convenabilă în care trebuie serviți cei trei pacienți este 3→1→2 deoarece în acest caz **T** are valoarea cea mai mică, T=28. Dacă numărul de pacienți este >10 procedeul prezentat este prohibitiv.

În lucrare se propune utilizarea unui algoritm de optimizare de tip “greedy” în 2 pași (fig. 6). În acest caz din figura 6 “LĂCOMIA“ algoritmului *greedy* rezidă în faptul că, el căutând minimul, se repede din primul pas la varianta în care pacienții sunt ordonați crescător pentru servire.

- Se dă vectorul: to=[5 9 3] ;
  - **Pas 1**: Calculez **Tgreedy** și încep cu **sortarea** în ordine crescătoare a lui to
  - ***[tg ig]=sort(to)***; furnizează noua ordine ***ig*** și noile valori ***tg*** în ordine crescătoare
- ig ordinea servirii = noua ordine
  - **pas 2**: calculez așteptarea Tg pentru toți pacienții în noua ordine
  - ***for k=1:n***
  - ***Tg(k) =Tg(k-1)+tg(k)***
  - ***end***
  - ***T=sum(Tg)*** calculez întârzierea cumulată T
  - ***Tgreedy=T***

Figura 6. Program matlab de implementare a Algoritmului greedy în 2 paşi în cazul din ex. 1

În concluzie, ordinea optimă pentru greedy este: începând cu acei pacienţi care au durate de tratament minim şi continuând în mod similar ca în exemplul următor:

Exemplul 2*: GREEDY acţionând conform ordinii optime.* Se consideră ca exemplu un şir de 4 pacienţi la care *durata de tratament a celor n=4 pacienţi ÎN ORDINEA SOSIRII este exprimată prin vectorul:*

t = [5 , 9 , 20 , 3]

a cărei lungime este L = n = 4 şi în care primul sosit necesită timpul de servire t(1)=3, următorul are timpul de servire t(2)=20, etc.

Algoritmul ***GREEDY acţionând conform ordinii optime*** este prezentat în figura 7.

T=0; L=n

PAS1: ***din vectorul t se tratează pacientul cu durata ce mai mică*** t_m

PAS2: se calculează timpul staţionarii în spital a pacientului tratat T =T+t_m **şi se elimină din t elementul cu durata cea mai mică: L=L-1**

DACA **(L -1)>0 se revine la PAS1**

ALTFEL **stop.**

Figura 7. Algoritmul greedy în cazul din exemplul 2

Pasul 1 şi pasul 2 ai algoritmului din figura 7 se vor executa de patru ori. La prima execuţie L=4, astfel că pacientul cu timp minim de servire este cel cu timpul cel mai mic de servire adică **min{**5, 9, 20, 3}**=3**. Acesta este servit şi externat după ***T(1)=3.*** Lungimea L este decrementată cu o unitate iar vectorul t devine t={5, 9, 20}. La a doua execuţie este servit pacientul cu timp de servire **min{**5, 9, 20}=5 care este externat deci după T(2)=T(1)+5=8, etc. Următorii doi pacienţi sunt externaţi după intervalul T(3)=T(2)+9=17, respectiv T(4)=T(3)+20=37. Astfel timpul minim de ocupare a paturilor de spital de către pacienţii din şir este ***Tgreedy=3+8+17+37=65.*** Dacă pacienţii ar fi fost serviţi nu în ordinea ***greedy,*** ci în ordinea instalării la coadă (fifo= first-input – first-output) timpul ocupării paturilor ***Tfifo=90.*** Timpul maxim de ocupare ar fi ***Tmax*** = 20+29+34+37=130 şi se obţine în cazul în care ordinea servirii este determinată de timpul maxim de servire: 20, 9, 5, 3. Se observă că algoritmul ***greedy*** asigură în acest caz o reducere de sută la sută faţă de timpul maxim.

5. Concluzii

Optimizarea servirii pacienţilor are importanţă economică întrucât minimizează timpul ocupării paturilor în spital de către pacienţii din lista de aşteptare a unei secţii medicale [5]. Se are în vedere o secţie oncologică în care autorul a efectuat cercetarea ştiinţifică în perioada doctoranturii [3]. Algoritmul greedy în 2 paşi, propus în prezenta lucrare este foarte simplu: la fiecare pas se selectează pacientul cu timpul minim de servire din mulţimea de clienţi rămasă. Prin metoda greedy obţinem deci întotdeauna planificarea optimă a pacienţilor. Problema poate fi generalizată pentru un sistem cu mai multe staţii de servire. Aceasta fiind inclusă în programul propriu de viitor al cercetării ştiinţifice postdoctorale.

BIBLIOGRAFIE

1. **GORUNESCU, M.:** Optimal Hospital Beds Allocation Using Queuing Systems. În Proceedings of the 3rd Romanian Conference on Artificial Intelligence and Digital Communications – AIDC2003, Research Notes in Artificial Intelligence and Digital Communications, Craiova: Universitaria Publishing House, 2003, Vol. 103, pp. 131-136, ISBN 978-8419-71-9. [EFA-04]
2. **ENE, G., E. FIROIU, S. ANGELESCU:** Sistem Informatic de procesare evaluare şi monitorizare a datelor screeningului în cancerul colului uterin, A III-a conferință națională de oncologie medicală, Mamaia 9-12, septembrie 2004.
3. **ENE, G.:** Aplicarea în domeniul oncologiei medicale a metodei VaR de evaluare a riscului, Revista Română de Informatică şi Automatică, nr. 2/2007, pp. 63-68.
4. **BĂDULESCU, F., GORUNESCU F.:** Informatica oncologică: metode statistico-informatice în oncologie, Bucureşti: Editura Didactică şi Pedagogică, 2003.