# Probabilităţi şi satistică - Cursul 7

Olariu E. Florentin

Aprilie, 2017

1 Introducere
   Clasificarea algoritmilor aleatori

2 Algoritmi Las Vegas
   Evaluarea unui game-tree
   RandomizedQuickSort
   Un algoritm aleator pentru determinarea medianei

3 Algoritmi Monte Carlo
   Verificarea înmulţirii matricilor
   Un algoritm aleator pentru min-cut

4 Metoda probabilistică
   Problema satisfiabilităţii
   Aplicaţii în teoria grafurilor

5 Bibliography

- Algoritmii pot fi clasificaţi drept algoritmi *determinişti* şi *nedeterminişti*.
- Această clasificare a dat naştere şi la binecunoscutele clase de complexitate: **P** şi **NP**.
- O definiţie simplificată a unui algoritm este următoarea: o intrare (input) care este procesată şi transformată într-o ieşire (output).
- Un *algoritm determinist* este un algoritm pentru care rezultatul (ieşirea) este acelaşi pentru orice intrare, având acelaşi comportament în orice execuţie.
- Un *algoritm nedeterminist* este un algoritm care poate da rezultate diferite pentru acelaşi input, având eventual, comportamente diferite la execuţii diferite (e.g. valorile variabilelor pot fi diferite).
- Diferenţa esenţială constă în aceea că un proces nedeterminist este influenţat de *alegerile* care pot fi făcute.

- Un algoritm determinist trebuie să rezolve (întotdeauna) corect o problemă şi cât de repede posibil (de obicei se cere ca numărul de paşi/operaţii să polinomial în dimensiunea inputului).
- Exemple de algoritmi nedeterminişti: algoritmii concurenţi şi *algoritmii aleatori*.
- Un algoritm aleator primeşte la intrare şi o sursă de numere aleatoare care îi permite să facă alegeri aleatoare în timpul execuţiei. Cel care îl proiectează încearcă să arate ca se comportă "corect" pentru fiecare input.

**Definition 1**

*Un* **algoritm aleator** *este un algoritm care în cursul execuţiei face anumite alegeri probabilistice.*

- Aceste alegeri probabilistice constau în generarea de valori ale unei variabile aleatoare. Aceste valori sunt implicate în calculele pe care algoritmul se presupune că le face.

- Algoritmii aleatori sunt cei mai cunoscuţi algoritmi de tip nedeterminist, iar randomizarea a devenit o abordare standard în proiectarea algoritmilor datorită simplităţii şi vitezei sporite.
- Acest tip de aplicaţii ale teoriei probabilităţilor în informatică a devenit frecvent in ultimele decade.
- În domenii precum comunicaţiile, criptografia şi optimizarea discretă randomizarea şi metodele probabilistice au devenit instrumente uzuale de investigaţie:
    - Protocolul Ethernet foloseşte alegeri aleatoare când accesează mediul de comunicaţie.
    - Testarea primalităţii (în criptografie) foloseşte alegeri probabilistice.
    - Unele probleme **NP**-dificile pot fi rezolvate pentru majoritatea intrărilor de către algoritmi aleatori.

- *Structuri de date*: sortare, statistici ordonate, căutare.
- *Identităţi algebrice*: verificarea identităţii polinoamelor şi matricilor.
- *Teoria (algoritmică a) grafurilor*: arbori parţiali de cost minim, drumuri de cost minim, tăieturi de pondere minimă.
- Numărare şi enumerare: permanentul unei matrici, numărarea structurilor combinatoriale.
- Calcul parale şi distribuit: evitarea blocajului (deadlock), consens distribuit.
- *Demonstraţii probabilistice existenţiale*: dovezi că un anumit obiect combinatorial există cu probabilitate nenulă printre obiectele unui spaţiu probabilistic.

- Studiul variabilelor aleatoare asociate unui algoritm aleator este utilizat pentru a analiza eficienţa şi probabilitatea de a greşi ale algoritmului.
- Clasificarea fundamentală a algoritmilor aleatori: algoritmi care rezolvă corect (întotdeauna) problema (asociată) şi algoritmi care greşesc.
- Algoritmii **Monte Carlo** au o probabilitate pozitivă de a greşi.
- Algoritmii **Las Vegas** nu greşesc niciodată (probabilitatea lor de a greşi este nulă).

- Algoritmii Las Vegas sunt algoritmi aleatori care garantează că fiecare output este corect (este soluţie a problemei pe care algoritmul încearcă să o rezolve).
- Literatura de specialitate a stabilit o definiţie puţin diferită a sintagmei "output corect" permiţând algoritmului să dea şi răspunsul " Nu ştiu."

**Definition 2**

*Fie $A$ un algoritm aleator căruia îi este permis răspunsul "?". $A$ este numit* **algoritm Las Vegas** *pentru calculul funcţiei $F$ dacă, pentru orice intrare $x$ (orice argument al lui $F$),*

(i) $P(A(x) = F(x)) \geqslant 1/2.$

(ii) $P(A(x) = "?") = 1 - P(A(x) = F(x)) \leqslant 1/2.$

- Pentru algoritmii Las Vegas ne interesează de obicei complexitatea medie: timpul mediu de execuţie şi spaţiul mediu de memorie folosit.
- Aceasta este posibil pentru că diferite execuţii ale algoritmului (pentru o aceeaşi intrare) au diferite caracteristice (în termeni de timp CPU şi spaţiu de memorie).
- Căutăm să descriem algoritmi Las Vegas cu timp de execuţie mediu mărginit (de obicei de un polinom).
- În definiţia de mai sus constanta $1/2$ poate fi schimbată cu orice altă constantă $\epsilon \in (0, 1)$, după cum o arată următorul rezultat.

### Proposition 1

*Fie $\epsilon \in (0,1)$ şi $A$ un algoritm aleator care calculează funcţia $F$ aşa încât*

$$P\left(A(x) = F(x)\right) \geqslant \epsilon \text{ şi}$$

$$P\left(A(x) = \text{"?"}\right) = 1 - P\left(A(x) = F(x)\right)$$

*Fie, pentru $k \in \mathbb{N}^*$, $A_k$, un algoritm aleator care pentru orice intrare $x$ execută în mod independent algoritmul $A$ de $k$ ori asupra lui $x$. Există un $k$ astfel ca $P\left(A_k(x) = F(x)\right) \geqslant 1/2$.*

- Un *game tree* este un arbore cu rădăcină în care nodurile interne aflate la distanţă pară faţă de rădăcină sunt etichetate cu MIN (rădăcină are o etichetă MIN), iar restul nodurilor interne sunt etichetate cu MAX. Fiecărei frunze îi este asociată o valoare - un număr real.
- Evaluarea arborelui se face astfel: fiecare frunzăreturnează valoarea asociată ei, fiecare nod MAX returnează cea mai mare valoare a unuia dintre fii, iar fiecare nod MIN returnează cea mai mică valoare a unuia dintre fii.
- Dat un game-tree, *problem evaluării* constă în a determina valoarea returnată de rădăcină.
- Acest tip de evaluare are un rol important în IA (în probleme game-playing).

- Vom limita analiza noastră la cazul special când în frunze valorile sunt biţi (astfel un nod MIN este o operaţie logică AND, iar un nod MAX are o operaţie logică OR).
- Considerăm un arbore binar complet cu adâncimea $2h$ şi $n = 4^h$ frunze.
- Este uşor de văzut că un algoritm determinist va fi forţat să citească toate frunzele: într-un nod AND primul copil evaluat poate returna valoarea 1, iar într-un nod OR valoarea 0 - în amândouă cazurile algoritmul este forţat sa evalueze ambii descendenţi.
- Algoritmul aleator evaluează recursiv un descendent ales la întâmplare al nodului curent. Dacă valoarea rezultată nu determină valoarea nodului se evaluează şi celălalt descendent.

**RandomEval**($x$) {
  **if**($x.operation$ = **AND**){
    **choose** *uniformly at random a child* $u$;
    **if**(**RandomEval**($u$) == **1**) {
      **let** $v$ *the other child*;
      **return RandomEval**($v$);
    }
    **return RandomEval**($u$);
  }
⋮
}

### Theorem 1.1

*Pentru un game-tree cu adâncimea $2h$, costul mediu al evaluării este cel mult $3^h$.*

Proof. Considerăm mai întâi un nod OR care returnează 1 cu două frunze (cazul cel mai nefavorabil pentru un algoritm determinist constă într-o frunză 0 şi o frunză 1); cu probabilitate $1/2$ algoritmul alege mai întâi frunza cu valoare 0, numărul mediu de paşi este $1/2 \cdot 1 + 1/2 \cdot 2 = 3/2$; în mod similar pentru un nod AND care returnează 0 cu două frunze, numărul mediu de paşi este cel mult $3/2$.

Evident numărul de paşi este 2 pentru un nod AND evaluat la 1 şi pentru un nod OR evaluat la 0 - nu există vreo economie. Câştigul este că, spre exemplu, într-un nod intern AND care returnează 1 amândoi copii OR trebuie să returneze 1 ceea ce este cazul "bun" pentru OR.

Folosim metoda inducţiei.

Evaluăm mai întâi numărul de paşi în cazul $h = 1$,i. e., un arbore cu rădăcina AND care are doi copii OR, fiecare având câte două frunze drept copii.

Dacă rădăcina returnează 0, atunci numărul de evaluări este $1 + 2 = 3$ (1 pentru unul dintre copii şi 2 pentru frunzele acestuia).

Dacă rădăcina returnează 1, numărul de evaluări este $3/2+3/2 = 3$ (amândoi copiii sunt în cazul "bun" pentru noduri OR).

Considerăm acum un nod OR ai cărui copii AND sunt rădăcinile câte unui arbore cu adâncimea $2(h-1)$. Dacă rădăcina OR returnează 1, atunci cel puţin unul dintre copii returnează 1. Cu probabilitate $1/2$ acest copil este ales mai întâi şi, cu probabilitate $1/2$, amândoi subarborii sunt evaluaţi. Costul mediu este cel mult

$$\frac{1}{2} \cdot 3^{h-1} + \frac{1}{2} \cdot 2 \cdot 3^{h-1} = \frac{3}{2} 3^{h-1}.$$

Dacă rădăcina OR returnează 0, amândoi copiii vor trebui evaluaţi ceea ce impune un cost de cel puţin $2 \cdot 3^{h-1}$.

Considerăm acum un nod AND rădăcină a unui arbore cu adâncimea $2h$. Dacă rădăcina returnează 0, atunci cel puţin unul dintre copiii OR ai rădăcinii returnează 0; cu probabilitate $1/2$ acest copil este ales mai întâi şi costul mediu este cel mult

$$2 \cdot 3^{h-1} + \frac{1}{2} \cdot \frac{3}{2} \cdot 3^{h-1} \leqslant 3^h.$$

Dacă rădăcina returnează 1, amândoi copiii trebuie evaluaţi impunând un cost de cel mult

$$2 \cdot \frac{3}{2} \cdot 3^{h-1} = 3^h. \blacksquare$$

- Dacă $N$ este numărul de frunze al unui astfel de arbore, atunci numărul mediu de paşi (i. e., numărul mediu de noduri evaluate) este cel mult $N^{\log_4 3} = N^c$ ($c < 0.8$).

- Considerăm problema the problem sortării elementelor unei mulţimi $S$ (pe care există o ordine totală) doar prin compararea perechilor de elemente.
- Dacă am putea găsi un element $x$ al lui $S$ aşa încât jumătate dintre elemenetele lui $S$ să fie mai mici ca $x$, atunci am putea utiliza următoarea procedură:
  - partiţionează $S \setminus \{x\}$ în două submulţimi $S_1$ şi $S_2$, unde $S_1$ constă din acele elemente care sunt mai mici decât $x$, iar $S_2$ din restul elementelor.
  - sortează recursiv $S_1$ şi $S_2$, apoi returnează elementele lui $S_1$ în ordine crescătoare, urmate de $x$ şi apoi de elementele lui $S_2$ în ordine crescătoare.

- Dacă am putea determina $x$ în $cn$ paşi (pentru o constantă $c$), atunci am putea partiţiona $S \setminus \{x\}$ în $(n-1)$ paşi comparând fiecare element cu $x$.
- Notăm cu $T(n)$ numărul de paşi pentru a asemenea procedură (în cazul cel mai nefavorabil) pentru $n = |S|$.
- $T(n)$ este dat de următoarea recurenţă

$$T(N) \leqslant 2\,T(n/2) + (c+1)n.$$

- Soluţia acestei recurenţe este $T(n) \leqslant \tilde{c}n \log n$ (pentru o constantă $\tilde{c}$) ceea ce dă un timp de execuţie de $\mathcal{O}(n \log n)$.
- Complexitatea timp rămâne aceeaşi chiar dacă $x$ nu împarte exact (ci doar aproximativ) în două mulţimea $S$.

- Întrebarea este cât de repede poate fi găsit un astfel de $x$?
- Un răspuns simplu este să alegem la întâmplare un astfel de $x$ din $S$; ceea ce rezultă este algoritmul RandomizedQuickSort - un exemplu de algoritm aleator de tip Las Vegas.

**RandQuickSort**$(S)$ **{**

**choose** $x \in S$ *independently and uniformly at* **random;**

// $x$ **se numeşte pivot.**

$S_1 \leftarrow \{y \in S \,:\, y < x\}$;

$S_2 \leftarrow \{y \in S \,:\, y > x\}$;

**return** [**RandQuickSort**$(S_1)$, $x$, **RandQuickSort**$(S_2)$]; **}**

- Suntem interesaţi să număram comparaţiile - acesta este costul dominant al algoritmului.

- Să presupunem că ordinea totală a elementelor lui $S$ este $x_1 < x_2 < \ldots < x_n$.
- În cazul cel mai nefavorabil complexitatea timp a algoritmului este $\mathcal{O}(n^2)$ (când, de exemplu pentru intrarea $x_1 = n, x_2 = n-1, \ldots, x_n = 1$, când pivotul este întotdeauna în prima poziţie).
- Suntem interesaţi în calcul numărului mediu de comparaţii.
- Numărul total de comparaţii, $X$, este evident o variablă aleatoare. Notăm cu $X_{ij}$ următoarea variabilă Bernoulli

$$X_{ij} = \begin{cases} 1, & x_i \text{ and } x_j \text{ sunt comparate în timpul execuţiei} \\ 0, & \text{altfel} \end{cases}$$

$X$ poate fi scrisă ca $X = \sum_{i=1}^{n} \sum_{j>i} X_{ij}$.

**Theorem 2.1**

*Pentru orice intrare numărul mediu de comparaţii făcute de RandQuickSort este* $2n \ln n + \mathcal{O}(n)$.

Proof. Dacă $p_{ij}$ este probabilitatea ca $x_i$ şi $x_j$ să fie comparate în timpul execuţiei algoritmului,

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j>i} \mathbb{E}[X_{ij}] = \sum_{i=1}^{n} \sum_{j>i} p_{ij},$$

$x_i$ şi $x_j$ sunt comparate dacă şi numai dacă $x_i$ sau $x_j$ este primul pivot din mulţimea $S^{ij} = \{x_i, x_{i+1}, \ldots, x_j\}$.
Pivotul fiind ales independent şi uniform aleator din fiecare submulţime, $p_{ij} = 2/(j - i + 1)$, pentru că orice element din submulţime are aceeaşi şansă să fie ales.

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1} = 2\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{j-i+1} =$$

$$= 2\sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{1}{k} = 2\sum_{k=1}^{n} \sum_{i=1}^{n-k+1} \frac{1}{k} = 2\sum_{k=1}^{n} \frac{n-k+1}{k} =$$

$$= (2n+2)\sum_{k=2}^{n} \frac{1}{k} - 4n = 2n\ln n + \mathcal{O}(n). \blacksquare$$

## Definition 3

*Dată o mulţime total ordonată cu $n$ elemente, $S$, mediana lui $S$ este un element $m \in S$ astfel încât cel puţin $\lfloor n/2 \rfloor$ elemente din $S$ sunt mai mici sau egale cu $m$ şi cel puţin $\lfloor n/2 \rfloor + 1$ elemente din $S$ sunt mai mari sau egale cu $m$.*

- Mediana poate fi găsită, determinist, în $\mathcal{O}(n \log n)$ paşi prin sortare şi alegerea elementului adecvat din mulţimea sortată; există, de asemeni un algoritm determinist mai complex, liniar, pentru determinarea medianei ([Blum73]).
- Vom prezenta un algoritm simplu, aleator, de complexitate liniară a cărui idee de bază este de a găsi două elemente apropiate în ordonarea crescătoare a lui $S$ şi care conţin mediana între ele.
- Acest algoritm este tot de tip Las Vegas.

**RandMedian**($S$) {

**let** $R$ *be the multi-set of* $\lceil \sqrt[4]{n^3} \rceil$ *chosen elements of* $S$ *independently and uniformly at* **random** *with replacement*;

**sort** $R$; // **se utilizează un algoritm optim determinist în** $\mathcal{O}(|R| \log |R|)$ **paşi**

**let** $d \leftarrow$ *the* $\lfloor \sqrt[4]{n^3} - \sqrt{n} \rfloor$*th smallest element in the sorted order of* $R$;

**let** $u \leftarrow$ *the* $\lfloor \sqrt[4]{n^3} + \sqrt{n} \rfloor$*th smallest element in the sorted order of* $R$;

$C \leftarrow \{x \in S \,:\, d \leqslant x \leqslant u\}$;

$l_d = |\{x \in S \,:\, x < d\}|$; $l_u = |\{x \in S \,:\, x > u\}|$;

**if**($l_d > n/2$ **or** $l_u > n/2$ **or** $|C| > 4\lceil \sqrt[4]{n^3} \rceil$)

  **return** *"no median found"*;

**sort** $C$;

**return** *the* $(\lfloor n/2 \rfloor - l_d + 1)$*th smallest element in the sorted order of* $C$; }

- Alegerea cardinalului mulţimii $R$ şi a lui $d$ şi $u$ sunt făcute pentru a garanta că
  - (i) $C$ include median lui $S$ cu probabilitate mare şi că
  - (ii) $C$ este suficient de mică încât să poată fi sortată în timp subliniar cu probabilitate mare.

**Theorem 3.1**

*Algoritmul aleator pentru determinarea medianei are o complexitate liniară şi oferă un răspuns corect.*

Proof. Algoritmul ar oferi un răspuns negativ dacă şi numai dacă mediana nu aparţine mulţimii $C$ când $l_d \leqslant n/2$, $l_u \leqslant n/2$ şi $|C| \leqslant 4\lceil \sqrt[4]{n^3} \rceil$; aceasta înseamnă că $l_d > n/2$ sau $l_u > n/2$. În ceea ce priveşte complexitatea timp, sortarea elementelor lui $C$ se poate face în $\mathcal{O}(\sqrt[4]{n^3} \log \sqrt[4]{n^3} = \mathcal{O}(n)$. ∎

- Următorul rezultat, dat aici fără demonstraţie (puteţi consulta [Mitzenmacher65]), încheie analiza algoritmului.

**Theorem 3.2**

*Probabilitatea ca algoritmul aleator pentru determinarea medianei să nu găsească mediana este cel mult* $1/\sqrt[4]{n}$.

- Algoritmii Monte Carlo sunt algoritmi aleatori care nu pot garanta un răspuns corect (nu oferă întotdeauna o soluţie a problemei).

**Definition 4**

*Un algoritm aleator $A$ este numit* **algoritm Monte Carlo** *pentru calculul funcţiei $F$ dacă, pentru orice intrare $x$ (orice argument al lui $F$),*

$$P\left(A(x) = F(x)\right) \geqslant 1/2.$$

- Astfel de algoritmi apar adeseori în probleme de optimizare. Foarte relevant pentru un algoritm Monte Carlo este că probabilitatea de a greşi este majorată.

- Ca şi în definiţia algorimilor Las Vegas constanta $1/2$ poate fi înlocuită prin orice altă constantă $\epsilon \in (0, 1)$.
- *Amplificarea* este operaţia de repetare independentă a execuţiei algoritmului până când probabilitatea erorii scade suficient de mult. Este ca şi cum am încerca să transformăm un algoritm Monte Carlo într-unul Las Vegas.
- Să presupunem că probabilitatea erorii unui algoritm $A$ este cel mult $\epsilon$ şi că repetăm în mod independent algoritmul de $k$ ori; probabilitatea de a obţine numai răspunsuri eronate este cel mult $\epsilon^k$,
- Cum $\lim_{k \to \infty} \epsilon^k = 0$, pentru valori mari ale lui $k$, putem face această probabilitate oricât de mică.
- Atenţie: valori mari ale lui $k$ pot mări foarte mult complexitatea timp.

- Să presupunem că avem trei matrici pătratice de ordin $n$, $A$, $B$ şi $C$; pentru simplificare vom presupune că operaţiile sunt modulo 2.
- Dorim să verificăm dacă $A \cdot B = C$. Aceasta este o problemă de decizie (se poate răspunde doar cu "da" sau "nu").
- O metodă constă în a înmulţi $A$ cu $B$ după care se compară rezultatul cu $C$; înmulţirea matricilor necesită $\mathcal{O}(n^3)$ timp sau, utilizând algoritmi mai sofisticaţi, $\mathcal{O}(n^{2.37})$.
- Descriem un algoritm aleator care permite verificarea mai rapidă cu riscul de a primi un răspuns greşit, dar cu probabilitate scăzută.

**RandomVerify**$(A, B, C)$ **{**

**generate** $r_1, r_2, \ldots, r_n \in \{0, 1\}$ *independently at uniformly at random*;

$r \leftarrow (r_1, r_2, \ldots, r_n)$;

**if**$(ABr = Cr)$

**return** *"yes"*;

**return** *"no"*; **}**

- Când algoritmul returnează "nu" suntem siguri că $A \cdot B \neq C$. Singura eroare pe care algoritmul o poate face este aceea de a returna "da" dar răspunsul să fie, de fapt, "nu".
- Vom majora această probabilitate de a greşi cu ajutorul următorului rezultat.

**Theorem 1.1**

*Pentru* $r \in \{0, 1\}^n$*, ales ca în algoritmul anterior,* $P(ABr = Cr) \leqslant \frac{1}{2}$*, ştiind că* $A \cdot B \neq C$*.* $\left(P(ABr = Cr | A \cdot B \neq C) \leqslant \frac{1}{2}.\right)$

Proof. Cum $D = AB - C \neq 0$ - putem presupune că $d_{1n} \neq 0$; dacă $Dr = 0$, atunci

$$\sum_{i=1}^{n} d_{1i} r_i = 0 \Rightarrow r_n = -\frac{\sum_{i=1}^{n-1} d_{1i} r_i}{d_{1n}}.$$

După ce am ales $r_1, \ldots, r_{n-1}$ există două posibile valori pentru $r_n$, deci probabilitatea ca egalitatea de mai sus să aibă loc este cel mult $1/2$.

$$P(ABr = Cr) =$$

$$= \sum_{(x_1, \ldots x_{n-1}) \in \{0,1\}^n} P\left[(ABr = Cr) \cap \{r_i = x_i \;:\; i = \overline{1, n-1}\}\right]$$

$$\leqslant \sum_{(x_1, \ldots x_{n-1}) \in \{0,1\}^n} P\left[\left(r_n = -\frac{\sum_{i=1}^{n-1} d_{1i} r_i}{d_{1n}}\right) \cap \{r_i = x_i \;:\; i = \overline{1, n-1}\}\right]$$

$$= \sum_{(x_1,\ldots x_{n-1})\in\{0,1\}^n} P\left[\left(r_n = -\frac{\sum_{i=1}^{n-1} d_{1i} r_i}{d_{1n}}\right)\right] P\left(\cap\{r_i = x_i \ : \ i = \overline{1, n-1}\right.$$

$$\leqslant \sum_{(x_1,\ldots x_{n-1})\in\{0,1\}^n} \frac{1}{2} P\left(\cap\{r_i = x_i \ : \ i = \overline{1, n-1}\}\right) = \frac{1}{2}. \blacksquare$$

- Algoritm constă în generarea a $n$ numere aleatoare uniforme şi în trei înmulţiri matrice=vector ($\mathcal{O}(n^2)$). Complexitatea timp a algorimtului este deci $\mathcal{O}(n^2)$.
- Dacă executăm în mod independent de $k$ ori algoritmul obţinem o probabilitate de agreşi de cel mult $2^{-k}$.

- Fie $G = (V, E, w)$ un multigraf, unde $w : E \to \mathbb{N}^*$ determină multiplicitatea muchiilor (altfel spus avem un graf cu ponderi întregi pe muchii).
- O *tăietură* (*cut*) în $G$ este generată de o bipartiţie a lui $V$, $(A, B)$, iar costul ei este $w(A, B) = \sum_{e \in E(A,B)} w(e)$, unde $E(A, B) = \{uv \in E : u \in A, v \in B\}$ sunt muchiile tăieturii.
- Problema *tăieturii de cost minim* (*min-cut problem*) cere să se determine o tăietură de cost $w$ minim în $G$. Pentru rezolvarea acestei probleme există un algoritm determinist[1] de complexitate timp $\mathcal{O}(nm + n^2 \log n)$ (prin contrast probelma *max-cut* este **NP**-hard).
- Analizăm în această secţiune un algoritm aleator pentru o tăietură minimă bazat pe contracţia muchiilor unui graf.

[1] Algoritmul Stoer-Wagner.

- Operaţia de bază, $contraction(u, v)$, înlocuieşte în graful curent nodurile $u$ şi $v$ cu un nod nou $z$ asignează mulţimea muchiilor incidente lui $z$ ca fiind muchiile incidente cu $u$ şi $v$ (altele decât cele dintre $u$ şi $v$).
- în timpul contracţiei muchiile incidente cu $u$ şi $v$ cu un capăt comun sunt reţinute ca muchii multiple.

**RandMinCut**$(G\ ,\ w)$ {
  **while**$(|V(G)| > 2)$ {
    **choose** $e = uv \in E(G)$ *independently and uniformly at* **random;**
    $(G, w) \leftarrow contraction(u, v)$;
  }
  **return** $E(G)$; }

- După fiecare pas numărul de noduri scade cu unul şi orice tăietură în noul graf corespunde unei tăieturi anterioare.
- După fiecare pas anumite tăieturi din graful original pot dispărea, dar algoritmul se termină cu o mulţime de muchii care reprezintă o tăietură din graful original.
- O strategie utilă este aceea de a termina algoritmul cu $k \geqslant 2$ noduri în loc de 2 şi după aceea se poate aplica un algoritm determinist pentru a determina o tăietură de cost minim în graful rămas.

**RandMinCut**$(G\,,\ w,\ k)$ **{**
  **while**$(|V(G)| > k)$ **{**
    **choose** $e = uv \in E(G)$ *independently and uniformly at* **random;**
    $(G, w) \leftarrow contraction(u, v)$;
  **}**
  **return** $E(G)$; **}**

## Theorem 2.1

*Algoritmul aleator* **RandMinCut**$(G\ ,\ w,\ k)$ *are o complexitate timp polinomială şi determină o tăietură minimă cu probabilitate de cel puţin* $\frac{k(k-1)}{n(n-1)}$.

Proof. Fiecare contractţie poate fi executată în $\mathcal{O}(n)$ paşi (de ce?); astfel, complexitatea algoritmului **RandMinCut**$(G\ ,\ w,\ k)$ este $\mathcal{O}(kn)$.

Dacă într-un multigraf cu $n$ noduri avem o tăietură de cost minim $h$, atunci gradul minim al unui nod este cel puţin $h$, deci numărul de muchii din $G$ este cel puţin $hn/2$; probabilitatea ca o muchi care se află într-o tăietură de cost minim să fie contractată va fi cel mult $\frac{h}{hn/2} = \frac{2}{n}$.

Fie $C$ o tăietură de cost minim. Estimăm probabilitatea ca algoritmul să nu returneze $C$.

Dacă $C$ ese returnată, atunci nici una dintre muchiile lui $C$ nu a fost contractată.

Fie $A_i$ evenimentul ca o muchie a lui $C$ a fost contractată în pasul $i$ şi $B_i$ evenimentul ca nicio muchie a lui $C$ nu a fost contractată în primele $(i-1)$ iteraţii. Numărul de noduri după $i$ iteraţii este $(n-i)$, deci

$$P(\overline{A}_1) \geqslant 1 - \frac{2}{n}, P(\overline{A}_i | B_{i-1}) \geqslant 1 - \frac{2}{n-i+1}.$$

Pe de altă parte,

$$P(B_i) = P(\overline{A}_i \cap B_{i-1}) = P(\overline{A}_i | B_{i-1}) P(B_{i-1}), \forall i \leqslant k.$$

$$P(B_k) = P(\overline{A}_1) \cdot P(\overline{A}_2 | B_1) \cdot \ldots \cdot P(\overline{A}_k | B_{k-1}) \geqslant$$

$$\geqslant \prod_{i=1}^{n-k} \left(1 - \frac{2}{n-i+1}\right) \geqslant \frac{k(k-1)}{n(n-1)}. \blacksquare$$

- Pentru $k = 2$ algoritmul **RandMinCut**$(G\ ,\ w)$ garantează o probabilitate a erorii de cel mult$[1 - 2/n(n-1)] \geqslant (1 - 2/n^2)$;
- Putem reduce această probabilitate repetând execuţia algoritmului: dacă îl executăm de $n^2/2$ ori (mărind complexitate timp la $\mathcal{O}(n^4)$) probabilitatea erorii este cel mult

$$\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}} < \frac{1}{e}.$$

- Dacă executăm de $n(n-1)\ln n$ ori algoritmul (mărind complexitate timp la $\mathcal{O}(n^4 \log n)$) probabilitatea erorii este cel mult

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)\ln n} < e^{-2\ln n} = \frac{1}{n^2}.$$

- Marele avantaj al acestui algoritm este simplitatea lui.

- Metoda probabilistică este o tehnică folosită pentru demonstrarea existenţei unor obiecte matematice (combinatorii) cu anumite proprietăţi.
- Ideea de bază a acestei metode constă în a demonstra că probabilitatea existenţei unui obiect cu proprietăţile cerute este strict pozitivă, ceea ce înseamnă că un asemenea obiect există.
- Pentru aceasta, construim un spaţiu de probabilitate peste mulţimea obiectelor implicate şi arătăm ca probabilitatea obiectului respectiv este nenulă.

## Theorem 0.1

*(Principiul mediei) Dacă $X$ este o variabilă aleatoare discretă cu $M[X] \geqslant \alpha$, atunci $P\{X \geqslant \alpha\} > 0$.*

Primele exemple ale aplicării metodei probabiliste sunt legate de chestiunea satisfiabilităţii.

## Proposition 2

*Fie $\mathcal{F} = \{C_1, C_2, \ldots, C_m\}$ o familie de $m$ clauze. Există o asignare a valorilor de adevăr a variabilelor booleene implicate, astfel ca numărul de clauze satisfăcute să fie cel puţin*

$$\sum_{i=1}^{m} \left(1 - 2^{-|C_i|}\right) \geqslant m \left(1 - 2^{-h}\right),$$

*unde $h = \min\limits_{1 \leqslant i \leqslant m} |C_i|$.*

**proof:** Imaginăm următorul experiment aleator abstract: fiecărei variabile booleene $x$ îi asignăm independent valoarea 1 (*adevărat*) sau 0 (*fals*) cu acceaşi probabilitate 0.5. Definim variabilele aleatoare

$$X_i = \begin{cases} 1, & \text{dacă } C_i \text{ este adevărată} \\ 0, & \text{dacă } C_i \text{ este falsă} \end{cases}$$

Probabilitatea ca să fie adevărată clauza $C_i$ este egală cu probabilitatea ca măcar unul dintre cei $l_i$ literali ai ei să fie adevărat, adică $\left(1 - 2^{-|C_i|}\right)$. Numărul de clauze satisfăcute este egal cu $X = \sum_{i=1}^{n} X_i$ şi, atunci

$$M[X] = \sum_{i=1} M[X_i] = \sum_{i=1} \left(1 - 2^{-|C_i|}\right) \geqslant m\left(1 - 2^{-h}\right).$$

Concluzia urmează conform Teoremei 0.1. ∎

## Corollary 1.1

*Orice instanţă a problemei k-SAT cu un număr de clauze mai mic strict decât $2^k$ este satisfiabilă. (O instanţă a problemei k-SAT are în fiecare clauză exact k literali.)*

Dem. Reluând argumentul de mai sus, cu $|C_i| = k$, $\forall i = \overline{1, m}$:

$$M[X] = \sum_{i=1} M[X_i] = \sum_{i=1} \left(1 - 2^{-|C_i|}\right) = m\left(1 - 2^{-k}\right) > m - 1,$$

obţinem de aici că $P\{X \geqslant m\} = P\{X > m - 1\} > 0$ - deci există o asignare a valorilor de adevăr care să satisfacă toate cele $m$ clauze. ■

- Marginea indicată în acest rezultat este cea mai bună posibilă deoarece putem defini o instanţă a problemei $k$-SAT cu $2^k$ clauze care să fie nesatisfiabilă: de exemplu familia tuturor clauzelor având $k$ literali definiţ peste o mulţime de $k$ variabile booleene.
- În mod similar se poate demonstra

## Proposition 3

*O familie de clauze $\mathcal{F} = \{C_1, \ldots, C_m\}$ este satisfiabilă dacă*

$$\sum_{i=1}^{m} 2^{-|C_i|} < 1.$$

- Următoarele două rezultate sunt aplicaţii ale metodei probabiliste în teoria grafurilor. Introducem aceste rezultate cu două definiţii mai degrabă informale.
- Fie $G = (V, E)$ un graf, o mulţime de noduri $S \subseteq V$ se numeşte *stabilă* dacă oricare două noduri din $S$ nu sunt adiacente: $uv \notin E, \forall\, u, v \in S$. Dacă $(A, B)$ este o partiţie a nodurilor lui $G$, *tăietura* generată de această partiţie este mulţimea de muchii în cross între $A$ şi $B$:

$$E(A, B) = \{uv \in E \,:\, u \in A, v \in B\}.$$

- Este de menţionat că atât problema determinării unei mulţimi stabile de cardinal maxim cât şi cea a determinării unei tăieturi cu număr maxim de muchii sunt probleme **NP**-hard.

**Proposition 4**

*Fie $G = (V, E)$ un graf cu $n$ noduri şi $m$ muchii. Există o bipartiţie $(A, B)$ a lui $G$ astfel încât*

$$|E(A, B)| \geqslant \frac{m}{2}.$$

Dem. Considerăm următorul algoritm aleator care construieşte o bipartiţie $(A, B)$ a lui $V$:

$A \leftarrow \varnothing$;

**for** $(i = \overline{1, n})$

  **cu probabilitate** $\frac{1}{2}$ **adaugă** $v$ **la** $A$;

$B \leftarrow V \setminus A$;

Pentru fiecare muchie $uv \in E$ definim o variabilă Bernoulli

$$X_{uv} = \begin{cases} 1, & \text{dacă } uv \in E(A,B), \\ 0, & \text{dacă } uv \notin E(A,B) \end{cases}$$

Evident că

$$P\{X_{uv} = 1\} = P\{(u \in A \text{ şi } v \in B) \text{ sau } (u \in B \text{ şi } v \in A)\} =$$

$$= P\{u \in A\} \cdot P\{v \in B\} + P\{u \in B\} \cdot P\{v \in A\} = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}.$$

În plus, $X = \sum_{uv \in E} X_{uv}$ este o variabilă aleatoare care numără câte muchii conţine tăietura generată de bipartiţia $(A,B)$ şi, atunci

$$M[X] = \sum_{uv \in E} M[X_{uv}] = \frac{m}{2} \Rightarrow P\left\{X \geqslant \frac{m}{2}\right\} > 0.$$

Astfel, va trebuie să existe o partiţie cu proprietăţile din enunţ. ■

## Proposition 5

*Fie $G = (V, E)$ un graf cu $n$ noduri şi $m \geqslant n/2$ muchii. Atunci $G$ are o mulţime stabilă (sau independentă) de noduri de cardinal cel puţin $n^2/m$.*

Dem. Fie $d = \dfrac{2m}{n}$ gradul mediu în $G$ $\left(\sum_{v \in V} d_G(v) = 2m\right)$; deoarece $m \geqslant n/2$, avem $(1 - d^{-1}) \geqslant 0$. Construim o mulţime stabilă folosind următorul algoritm aleator ( $V = \{v_1, v_2, \ldots, v_n\}$):

**1. for** $(i = \overline{1, n})$

**cu probabilitate** $\left(1 - d^{-1}\right)$ **şterge nodul** $v_i$;

// **se elimină şi toate muchiile incidente cu** $v$.

**2. for** $(e = uv \in E)$

**şterge muchia** $e$ **şi nodul** $u$ **sau** $v$;

După execuţia acestui algoritm, nodurile rămase formează o mulţime independentă. După pasul 1 un nod rămâne în graf cu probabilitate $d^{-1}$; fie $X_i$ o variabilă egală cu 1 dacă nodul $v_i$ a rămas în graf şi 0 altfel. Notăm $X = \sum_{i=1} X_i$ numărul de noduri rămase după pasul 1. Deoarece fiecare $X_i$ este o variabilă Bernoulli cu media $d^{-1}$, obţinem $M[X] = nd^{-1}$.

Fie acum $Y_{uv}$, $uv \in E$, un alt set de variabile Bernoulli: $Y_{uv} = 1$ dacă muchia $uv$ rămâne în graf după pasul 1 şi 0 altfel. $Y = \sum_{uv \in E} Y_{uv}$ este numărul de muchii rămase în graf după primul pas.

Evident că $P\{Y_{uv} = 1\} = d^{-2}$ (probabilitatea ca nici $u$, nici $v$ să nu fie şterse), deci $M[Y_{uv}] = d^{-2}$ şi $M[Y] = md^{-2} = \dfrac{n}{2d}$.

Fie $Z$ numărul de noduri rămase în graf după pasul 2. Deoarece $Z \geqslant X - Y$ (mai sunt şterse încă cel mult $Y$ noduri):

$$M[Z] \geqslant M[X] - M[Y] = \frac{n}{d} - \frac{n}{2d} = \frac{n}{d} = \frac{n^2}{2m}. \blacksquare$$

- Următorul rezultat foloseşte noţiunea de *turneu* (un digraf $D = (V, A)$ cu proprietatea că între orice două noduri există exact un singur arc: $\forall\, u, v \in V$, $|\{\vec{uv}, \vec{vu}\} \cap A| = 1$).
- Denumirea vine din aceea că nodurile pot fi asimilate unor jucători şi fiecare pereche de jucători se confruntă o singură dată: $\vec{uv} \in A$ numai dacă $u$ îl bate pe $v$.
- Un turneu $D$ are proprietatea $P_k$ dacă pentru orice mulţime de $k$ jucători există un alt jucător care îi bate pe toţi ($k < |V|$).
- Pentru un $k$ dat, există un turneu cu proprietatea $S_k$? Metoda oferă şi o idee despre câte noduri trebuie să aibă turneul.

## Proposition 6

*Fie $k \in \mathbb{N}^*$, dacă $\binom{n}{k}(1-2^{-k})^{k-n} < 1$, atunci există un turneu cu $n$ noduri care să aibă proprietatea $P_k$.*

Dem. Construim mai întâi un turneu aleator astfel:

$A \leftarrow \varnothing$;

**for**$(\{u, v\} \subseteq V)$

**cu probabilitate** $\frac{1}{2}$ **adaugă** $\vec{uv}$ **sau** $\vec{vu}$ **la** $A$;

Fie $M \subseteq V$, $|M| = k$; probabilitatea ca un nod $v \notin M$ să domine toate nodurile din $M$ este $2^{-k}$, deci probabilitatea ca să nu le domine pe toate cele din $M$ este $(1-2^{-k})$. Evenimentele că două noduri distincte $u$, respectiv $v \neq u$ nu domină nodurile din $M$ sunt independente, astfel probabilitatea ca mulţimea $M$ să nu fie dominată de nici un nod din afara ei este $(1-2^{-k})^{n-k}$.

Acum putem estima probabilitatea ca nici o mulţime $S$, de cardinal $k$, să nu fie dominată:

$$\binom{n}{k}(1-2^{-k})^{n-k} < 1,$$

deci probabilitatea ca măcar una dintre mulţimile de cardinal $k$ să fie dominate este nenulă.

Fie acum $f(k)$ numărul minim de noduri ale unui turneu care are proprietatea $P_k$. Se poate arăta ([Alon08]) că $f(k) = \mathcal{O}(k^2 2^k)$. ∎

End

- Alon, N., J. H. Spencer, *The probabilistic method*, Wiley, 2008.
- Bertsekas, D. P., J. N. Tsitsiklis, *Introduction to Probability*, Athena Scietific, 2002.
- Blum, M., R W. Floyd. V. Pratt, R. L. Rivest, R. E. Tarjan, *Time bounds for selection*, J. of Comp. and Sys. Sci. 7, pp. 448-461,1973.
- Karger, D., *Global min-cuts in RNC and other ramifications of a simple min-cut algorithm*, ACM-SIAM Symp. on Discr. Alg. 4, pp 21-30, 1993.
- Mitzenmacher, M., E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, 1995.

- Motwani, R., P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 2005.
- Spencer, J. H., *Ten lectures on the probabilistic method*, SIAM, 1994.