Tema Algoritmi Aproximativi

Neculae Andrei-Fabian, Grupa 252

1 Knapsack

Problema 1

Fie S un șir de numere naturale $\{s_1, s_2, \ldots, s_n\}$ și K un număr natural, cu $K \geq s_i$ pentru orice i între 1 și n. Vrem să determinăm suma **maximă**, dar care să fie $\leq K$, ce poate fi formată din elementele din S (numerele pot fi luate cel mult o singură dată).

a) Scrieți un algoritm pseudo-polinomial care găsește suma optimă. Indicați complexitatea de timp/spațiu a algoritmului propus de voi și justificați de ce acesta este corect (de ce soluția găsită este optimă)

Algoritm propus

```
int max_sum(const std::vector<int>& a_S, int a_K)
{
    std::vector<int> dp(a_K + 1, 0);
    dp[0] = 1;
    for (int i = 1; i <= static_cast<int>(a_S.size()); ++i)
    {
        for (int j = a_K - a_S[i]; j >= 0; --j)
        {
            if (dp[j])
            {
                dp[j + a_S[i]] = 1;
            }
        }
    }

    int answer{ a_K };
    while (!dp[answer])
    {
        --answer;
    }
    return answer;
}
```

Complexitate timp/spatiu

- Algoritmul are complexitatea de timp $O(N \cdot K)$, unde N este numărul de elemente din S.
- Complexitatea de spațiu este $O(K)$.

Corectitudine

- Observam ca fiecare element s_i va fi adaugat la toate sumele posibile obtinute din submultimea $\{s_1, s_2, \ldots, s_{i-1}\}$.
- La pasul 0, vom seta 0 ca raspuns corect. Putem face acest lucru intrucat numerele noastre sunt pozitive.

- La pasul 1, vom seta $(0 + s_1)$ ca raspuns corect $\iff$ $(0 + s_1) \leq K$.
- In mod similar, pentru $\forall$ i $\in [2, n]$ si pentru $\forall S_{precedenta}$ obtinuta, vom seta $S_{precedenta} + s_i$ ca raspuns corect $\iff$ $S_{precedenta} + s_i \leq K$.
- Raspunsul final va fi cel mai mare numar cu valoarea 1 din intervalul $[0, K]$. Stim sigur ca acest rapsuns este optim, intrucat setam cu 1 toate sumele pe care le putem obtine cu elementele date.

b) Scrieți un algoritm aproximativ care calculează o sumă cel puțin pe jumătate de mare ca cea optimă, dar rulează în timp O(n) și complexitate spațiu O(1).

Algoritm propus

```
int max_sum()
{
    int K{};
    std::cin >> K;
    int s_i{}, answer{ 0 };
    while (std::cin >> s_i)
    {
        if (answer + s_i <= K)
        {
            answer += s_i;
        }
        else
        {
            answer = std::max(answer, s_i);
        }
    }
    return answer;
}
```

Complexitate timp/spatiu

- Algoritmul are complexitatea de timp O(N), unde N este numărul de elemente din S.
- Complexitatea de spațiu este O(1).

Corectitudine

- Fie ALG solutia propusa si OPT solutia optima. Exista doua cazuri:
  - $\sum_{i=1}^{n} s_i \leq K$, caz in care ALG = OPT (i.e. nu se intra deloc pe ramura else).
  - Se intra pe ramura else, adica $answer + s_i \geq K$. Cu alte cuvinte, ori $answer$, ori s_i sunt $\geq \frac{K}{2}$, si atunci selectam maximul dintre acestea. Asadar, ALG $\geq \frac{K}{2}$ (1). De asemenea, stim ca $K \geq$ OPT (2). Din (1) si (2) $\implies$ ALG $\geq \frac{K}{2} \geq \frac{OPT}{2} \implies$ OPT $\leq 2 \cdot$ ALG.

2 Load Balance

Problema 1

Fie o iterație a problemei Load Balancing (cursul 2, slide-ul 16) pentru 2 mașini. La seminarul de algoritmi aproximativi unul dintre studenți propune un algoritm de rezolvare și susține că acesta este 1.1 aproximativ. El rulează algoritmul pe un set de n activități și obține o încărcătură de 80 pe una dintre mașini, respectiv 120 pe cealaltă. Este posibil ca factorul lui de aproximare să fie corect...

a) ...ținând cont că rezultatul obținut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 100?

Demonstratie

- Fie $\{30, 60, 90, 20\}$ un set de activitati cu timpii cel mult 100.
- Observam ca OPT este 110, obtinut din {90} si {30, 60, 20} sau {90, 20} si {30, 60}.
- De asemenea, ALG respecta conditiile din cerinta, avand configuratiile {30, 90} si {60, 20}; si genereaza, evident, 120.
- Cum $110 \cdot 1.1 = 121 > 120$, putem afirma ca algoritmul propus ar putea fi 1.1-aproximativ.

b) ...ținând cont că rezultatul obținut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 10?

Demonstratie

- Stiind ca pentru $\forall i \in \{1, \ldots, n\}, t_i \leq 10$ si ca OPT aloca cat mai echilibrat activitatile, putem observa ca diferenta absoluta de incarcatura dintre cele doua masini va fi de maxim 10.
- Pentru a putea avea un algoritm 1.1-aproximativ, ar trebui ca diferenta absoluta dintre masini sa fie maxim $10 \cdot 1.1 = 11$.
- Insa, stiind ca masinile noastre au incarcatura 80, respectiv 120, observam ca $|80-120| = |-40| = 40 > 11$.
- In concluzie, algoritmul propus nu poate fi 1.1-aproximativ.

Problema 3

Fie algoritmul *Ordered-Scheduling Algorithm*, care implică algoritmul descris anterior la care adăugăm o preprocesare cu care sortăm descrescător activitățile după timpul de desfășurare. Th.2 afirmă că acest algoritm este $\frac{3}{2}$-aproximativ. Arătați că acest factor de aproximare poate fi îmbunătățit la $\frac{3}{2} - \frac{1}{2 \cdot m}$ (unde m este numărul de calculatoare pe care se pot executa activități).

Notatii

- k = indicele masinii cu load-ul maxim in urma rularii algoritmului, i.e. ALG = $load(k)$
- q = ultimul job asignat masinii k
- $load'(M)$ = load-ul masinii M dupa asignarea primelor q-1 job-uri, dar nu si job-ul q

Demonstratie

- Conform demonstratiilor de la curs:
  - $load'(k) \leq \frac{1}{m} \sum_{i=1}^{m} load'(i) = \frac{1}{m} \cdot \sum_{j=1}^{q-1} t_j \leq \frac{1}{m}(\sum_{j=1}^{n} t_j - t_q)$
  - $t_q < \frac{1}{2}(t_m + t_{m+1}) \leq \frac{1}{2} \cdot$ OPT
- ALG = $load'(k) + t_q \leq \frac{1}{m}(\sum_{j=1}^{n} t_j - t_q) + t_q = \frac{1}{m} \sum_{j=1}^{n} t_j + t_q \cdot (1 - \frac{1}{m}) < \frac{1}{m} \sum_{j=1}^{n} t_j + \frac{1}{2}(t_m + t_{m+1}) \cdot (1 - \frac{1}{m})$
- $\frac{1}{m} \sum_{i=1}^{n} t_i + \frac{1}{2}(t_m + t_{m+1}) \cdot (1 - \frac{1}{m}) \leq \text{OPT} + \frac{1}{2} \cdot \text{OPT} \cdot (1 - \frac{1}{m}) = \text{OPT} + \frac{\text{OPT}}{2} - \frac{\text{OPT}}{2 \cdot m} = (\frac{3}{2} - \frac{1}{2 \cdot m}) \cdot \text{OPT}$
- Asadar, din cele doua inegalitati deducem ca ALG $\leq (\frac{3}{2} - \frac{1}{2 \cdot m}) \cdot$ OPT, deci algoritmul este de fapt $(\frac{3}{2} - \frac{1}{2 \cdot m})$-aproximativ.

3 Travelling Salesman Problem

Problema 1

Considerăm o variantă a TSP, în care toate muchiile din graf au ponderea 1 sau 2.

a) Arătați că problema rămâne NP-hard pentru aceste instanțe.

Notatii

- $G(V, E)$ = un graf initial oarecare, neponderat
- Construim un graf ponderat complet $G'(V', E')$, dupa cum urmeaza:
  - $V' = V$
  - Pentru fiecare $e \in E$, ponderea muchiei va fi 1.
  - Pentru fiecare $e \notin E$, ponderea muchiei va fi 2.

Demonstratie

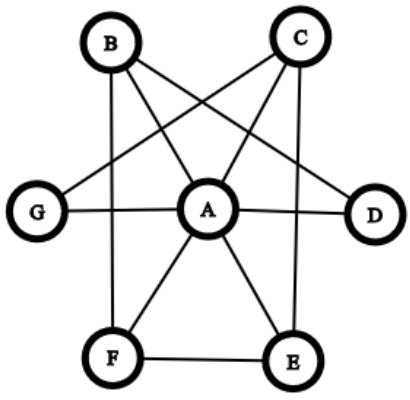
- Presupunem prin reducere la absurd ca problema noastra nu este NP-hard, asadar avem un algoritm polinomial determinist, care ne ofera solutia optima.
- Observam ca, in urma aplicarii algoritmului, vom obtine costul minim egal cu N doar daca graful G este hamiltonian, altfel costul minim va fi cel putin $(n-1)+2 = n+1$.
- Asadar, observam ca algoritmul nostru, care are timp polinomial, ne ajuta sa determinam daca un graf are un ciclu Hamiltonian. Insa, algoritmul de determinare a unui ciclu Hamiltonian este NP-hard, asadar ajungem la o contradictie.
- In concluzie, problema ramane NP-hard.

b) Arătați că aceste ponderi satisfac în continuare inegalitatea triunghiului.

Demonstratie

- Observam ca exista doar 4 cazuri posibile:
  - $\{1,1,1\} \to 1+1 = 2 \geq 1$ (A)
  - $\{1,1,2\} \to 1+1 = 2 \geq 2 \wedge 1+2 = 3 \geq 2$ (A)
  - $\{1,2,2\} \to 1+2 = 3 \geq 2 \wedge 2+2 = 4 \geq 1$ (A)
  - $\{2,2,2\} \to 2+2 = 4 \geq 2$ (A)
- Asadar, ponderile satisfac in continuare inegalitatea triungiului.

c) Algoritmul descris în curs (cursul 3, slide-urile 18-19) este un algoritm 2-aproximativ pentru forma generală a TSP (pentru instanțele care respectă inegalitatea triunghiului). Verificați dacă în această variantă a problemei, algoritmul din curs este chiar $\frac{3}{2}$-aproximativ.



Notatii

- $G(V, E)$ = graful prezentat in imaginea de mai sus
- $V = \{A, B, C, D, E, F, G\}$
- E = muchiile prezente in graf, cu pondere 1, si cele care nu apar in graf, cu pondere 2 (au fost omise pentru simplitate)

Demonstratie

- Observam ca solutia data de OPT este $A \to G \to C \to E \to F \to B \to D \to A$, care are cost $7 \cdot 1 = 7$.
- Daca alegem nodul A pe post de radacina, am obtine un arbore in care restul nodurilor sunt frunze.
- Aplicand algoritmul din curs, am obtine ciclul $A \to B \to C \to D \to E \to F \to G \to A$, care are cost $3 \cdot 1 + 4 \cdot 2 = 3 + 8 = 11$.
- Cum $11 > \frac{3}{2} \cdot 7 = 10.5$, deducem ca algoritmul din curs nu poate fi $\frac{3}{2}$-aproximativ.

4 Vertex Cover

Problema 1

Fie $X = \{x_1, x_2, \ldots, x_n\}$ o mulțime de variabile boolene. Numim formulă booleană (peste mulțimea X) în *Conjunctive Normal Form* (CNF) o expresie de forma $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ unde fiecare predicat (clauză) C_i este o disjuncție a unui număr de variabile (este alcătuit din mai multe variabile cu simbolul "$\vee$" - *logical or* - între ele). Exemplu de astfel de expresie:

$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_7) \wedge (x_1 \vee x_5 \vee x_6) \wedge (x_2 \vee x_5 \vee x_7)$

Evident că orice astfel de expresie va fi evaluată ca true dacă toate elementele lui X iau valoarea true. Ne interesează în schimb să aflăm numărul minim de elemente din X care trebuie să aibă valoarea true astfel încât toată expresia să fie true. Fie următorul algoritm pentru problema de mai sus în varianta în care fiecare clauză are exact trei variabile (numită *3CNF*):

Greedy-3CNF

1. Fie $C = \{C_1, \ldots, C_m\}$ mulțimea de predicate, $X = \{x_1, \ldots, x_n\}$ mulțimea de variabile.
2. Cât timp $C \neq \emptyset$ execută:
   a) Alegem aleator $C_j \in C$.
   b) Fie x_i una dintre variabilele din C_j.
   c) $x_i \leftarrow$ true
   d) Eliminăm din C toate predicatele care îl conțin pe x_i
3. Soluția constă din variabilele pe care le-am setat ca true pe parcursul execuției algoritmului

a) Este algoritmul descris mai sus un algoritm aproximativ? În cazul afirmativ, determinați factorul de aproximare (*worst case*) al algoritmului. Altfel, justificați de ce nu este aproximativ.

Notatii

- $C_1 = (x_1 \vee x_2 \vee x_3)$
- $C_2 = (x_1 \vee x_4 \vee x_5)$
- $C_3 = (x_1 \vee x_6 \vee x_7)$

Demonstratie

- Observam ca pentru $C = C_1 \wedge C_2 \wedge C_3$, OPT = 1, fiind indeajuns sa il facem pe x_1 true.
- In exemplul oferit, ALG poate selecta, spre exemplu, x_2, x_4 si x_6, obtinand astfel 3, insa putem observa ca aceasta valoare continua sa creasca cu cat adaugam termeni de forma $C_i = (x_1 \vee x_{2 \cdot i} \vee x_{2 \cdot i+1})$.
- Asadar, pentru un numar k de astfel de clauze, ALG ne poate da k, in timp ce OPT ramane egal cu 1.
- In concluzie, algoritmul este m-aproximativ.

b) Modificați algoritmul de mai sus astfel încât acesta să fie un algoritm 3-aproximativ pentru problema inițială (și justificați de ce se obține acest factor în urma modificărilor voastre).

Pseudocod propus

1. Fie $C = \{C_1, \ldots, C_m\}$ mulțimea de predicate, $X = \{x_1, \ldots, x_n\}$ mulțimea de variabile.
2. Cât timp $C \neq \emptyset$ execută:
   a) Alegem aleator $C_j \in C$.
   b) $x_i \leftarrow true, \forall x_i \in C_j$
   c) Eliminăm din C toate predicatele care îl conțin pe $x_i, \forall x_i \in C_j$
3. Soluția constă din variabilele pe care le-am setat ca true pe parcursul execuției algoritmului

Demonstratie

- Stim ca OPT minimizeaza numarul de variabile.
- Asadar, putem presupune ca OPT selecteaza la fiecare pas o singura variabila, cea care acopera cele mai multe predicate.
- In schimb, ALG selecteaza cate 3 variabile la fiecare pas.
- Prin urmare, la pasul k, in cel mai rau caz, OPT va genera raspunsul k, iar ALG va genera $3 \cdot k$.
- In concluzie, putem spune ca algoritmul propus este 3-aproximativ.

c) Reformulați problema de mai sus sub forma unei probleme de programare liniară.

Reformulare

- Problema data poate fi reformulata ca o problema liniara astfel:
- Fie multimea de variabile $X = \{x_1, x_2, \ldots, x_n\}$ si multimea de predicate in forma 3CNF $C = \{C_1, C_2, \ldots, C_n\}$, unde:
  1. $x_i \in [0, 1], \forall i \in \{1, \ldots, n\}$
  2. $\forall C_i \in C, C_i = (x_{i,1} \vee x_{i,2} \vee x_{i,3})$, avem $x_{i,1} + x_{i,2} + x_{i,3} \geq 1$
- Minimizati $\sum_{i=1}^{n} x_i$

d) Dați o soluție 3-aproximativă care să rezolve problema de programare liniară formulată la subpunctul anterior.

Pseudocod propus

- Fie $C = \{C_1, \ldots, C_m\}$ mulțimea de predicate, $X = \{x_1, \ldots, x_n\}$ mulțimea de variabile.
- Folosim algoritmul Simplex, obtinand rezultatele in X.

- $\forall i \in \{1, \ldots, n\}$, alegem 1 daca $x_i \geq \frac{1}{3}$, altfel alegem 0.
- Solutia va fi suma alegerilor, i.e. $\sum_{i=1}^{n} \begin{cases} 1, & \text{daca } x_i \geq \frac{1}{3} \\ 0, & \text{daca } x_i < \frac{1}{3} \end{cases}$

Demonstratie

- $\text{ALG} = \sum_{i=1}^{n} \begin{cases} 1, & \text{daca } x_i \geq \frac{1}{3} \\ 0, & \text{daca } x_i < \frac{1}{3} \end{cases} \leq \sum_{i=1}^{n} 3 \cdot x_i = 3 \cdot \sum_{i=1}^{n} x_i \leq 3 \cdot OPT$
- Asadar, algoritmul propus este o 3-aproximare a solutiei optime.