Štetje rešitev klasičnega problema nahrbtnika

Poročilo pri predmetu Finančni praktikum

Kristina Vatovec in Žan Mikola

Januar 2021

Kazalo

Slike

1 Uvod

Napisano poročilo je sklepni del projektne naloge pri predmetu Finančni praktikum.

Projektna naloga se nanaša na problem štetja rešitev klasičnega problema nahrbtnika. Pri delu sva se oprla na članek Štefankovič idr. (2012).

V prvem delu poročila je opisan problem in predstavljen algoritma za dan problem. Glavni del projektne naloge je bila implementacija algoritma v razdelku 2 . Algoritem sva napisala v programskem jeziku Python .

Sledil je eksperimentalni del, kjer sva poskušala ugotoviti ali pri raličnih vhodnih podatkih prihaja do sprememb časovne zahtevnosti. Algoritem sva preizkusila pri različnem številu predmetov, poskusov in pa vrednostih epsilona. Sklepi so povzeti v poglavju 3.

Da bi bila izkušnja reševanje problema za samega uporabnika bolj zanimiva sva v okviru projektne naloge dodala grafični uporabniški vmesnik.

2 Opis problema

Podanih je n elementov z celoštevilskimi težami $w_1, ..., w_n$ in kapaciteto C, ki je prav tako celo število Privzemimo štetje klasičnega problema nahrbtnika. S pomočjo algoritma, ki temelji na dinamičnem programiranju želimo poiskati oceno za število rešitev problema znotraj relativne napake $1 \pm \epsilon$ v polinomskem času n in $1/\epsilon$.

Preden nadaljujemo, omenimo naslednji izrek, ki je bistvenega pomena pri samem problemu.

Izrek 1. *Podane so teže $w_1, ..., w_n$ in kapaciteta C pri problemu nahrbtnika. Naj bo Z število rešitev problema. Obstaja deterministični algoritem, ki za vsak $\epsilon \in [0, 1]$ vrne $Z^{'}$ za katerega velja $Z \leq Z^{'} \leq Z(1 + \epsilon)$.*

Poglejmo si še funkcijo $T : \{0, \ldots, n\} \times \{0, \ldots, s\} \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$, ki je definirana v spodnjem algoritmu.

Vhod: Celaštevila $w_1, ..., w_n$, C in $\epsilon > 0$.

1. Postavimo $T[0, 0] = 0$ and $T[o, j] = \infty$ za $j > 0$.
2. Postavimo $Q = (1 + \epsilon/(n + 1))$ in $s = \lceil nlog_Q 2 \rceil$.
3. Za $i = 1 \rightarrow n$, za $j = 0 \rightarrow s$, postavim

$$T[i, j] = \ min_{\alpha \in [0,1]} max \begin{cases} T[i-1, \lceil j + log_Q \alpha \rceil], \\ T[i-1, \lceil j + log_Q (1-\alpha) \rceil] + w_i, \end{cases}$$

   kjer po dogovoru velja $T[i - 1, k] = 0$ za $k < 0$.

4. Naj bo
$$j^{'} := max\{j : T[n, j] \leq C\}.$$
5. Izhod $Z^{'} := Q^{j^{'}+1}$

Pri iskanju minimuna funkcije T v odvisnost od parametra $\alpha \in [0, 1]$, je v resnici dovolj gledati le α, ki ustrezajo vrednostim diskretne množice S. Za $j \in \{0, 1, \ldots, s\}$, je množiča $S = S_1 \cup S_2$, kjer $S_1 = \{Q^{-j}, \ldots, Q^0\}$ in $S_2 = \{1 - Q^0, \ldots, 1 - Q^{-j}\}$.

Izkaže se, da izhodni podatek Z' zadošča zgoraj napisanem izreku, kar pa je tisto, kar si želimo pri algoritmu za naš problem.

V tem algoritmu smo uporabili funkcijo T, vendar na prvi pogled ni povsem jasno kaj pomeni vrednost, ki jo vrne. Fukcija je T je torej aproksimacijska funkcija funkcije τ. Funkcija $\tau : \{0, \ldots, n\} \times \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\pm\infty\}$, kjer $\tau(i, a)$ vrne najmanjši C, Pri katerem obstaja vsaj a rešitev problema nahrbtnika z težami $\omega_1 \ldots \omega_n$ in kapaciteto C. Fukcijo τ ne znamo efektivno izračunati, zato definiramo fukcijo T.

Torej točno število rešitev problema nahrbtnika je

$$Z = max\{a : \tau(n, a) \leq C\}$$

3 Algoritem

Kot sva že omenila v uvodu, je bil glavni del projektne naloge implementacija zgornjega algitma.

Celotna kodo s kometarji sva dodala na repozitorij. Napisana je v programskem jeziku pythonin jo lahko poiščete v datoteki pod imenom *program.py*.

Spodaj pa si lahko ogledate glavno fukncijo v sami kodi.

```
def resitve(teze,C, epsilon):
    teze = [x for x in teze if x <=C]
    n = len(teze)

    Q = (1 + (epsilon / (n + 1)))
    s = math.ceil(n * math.log(2,Q))
    T = [[0 for stvari in range(s+1)] for row in range(n+1)]
    for j in range(1,s+1):
        T[0][j] = float('inf')
    for i in range(1,n+1):
        for j in range(s+1):
            vrednosti = []
            for k in range(j,-1,-1):
                alpha1 = Q **(-k)
                if math.floor(j + math.log(alpha1,Q)) < 0:
                    prva1 = 0
                else:
                    prva1 = T[i-1][math.floor(j + math.log(alpha1,Q))]
                if 1- alpha1 !=0:
                    if math.floor(j + math.log(1-alpha1,Q)) < 0:
                        druga1 = 0 + teze[i-1]
                if 1-alpha1 == 0:
                    druga1 = 0 + teze[i-1]
                elif math.floor(j + math.log(1-alpha1,Q)) > 0:
                    druga1 = T[i-1][math.floor(j + math.log(1-alpha1,Q))] + teze[i-1]
                vrednosti.append(max(prva1,druga1))
```

```
            alpha2 = 1-(Q ** (-k))
            if math.floor(j + math.log(1-alpha2,Q)) < 0:
                druga2 = 0 + teze[i-1]
            else:
                druga2 = T[i-1][math.floor(j + math.log(1-alpha2,Q))] + teze[i-1]
            if alpha2 !=0:
                if math.floor(j + math.log(alpha2,Q)) < 0:
                    prva2 = 0
            if alpha2 == 0:
                prva2 = 0
            elif math.floor(j + math.log(alpha2,Q)) > 0:
                prva2 = T[i-1][math.floor(j + math.log(alpha2,Q))]
            vrednosti.append(max(prva2,druga2))
        T[i][j] = min(vrednosti)
mozni_j_crta = []
for j in range(s+1):
    if T[n][j] <= C:
        mozni_j_crta.append(j)
if mozni_j_crta == []:
    return 1
j_crta = mozni_j_crta[-1]
resitev = Q ** (j_crta + 1)
return resitev
```

4 Časovna zahtevnost in generirani podatki

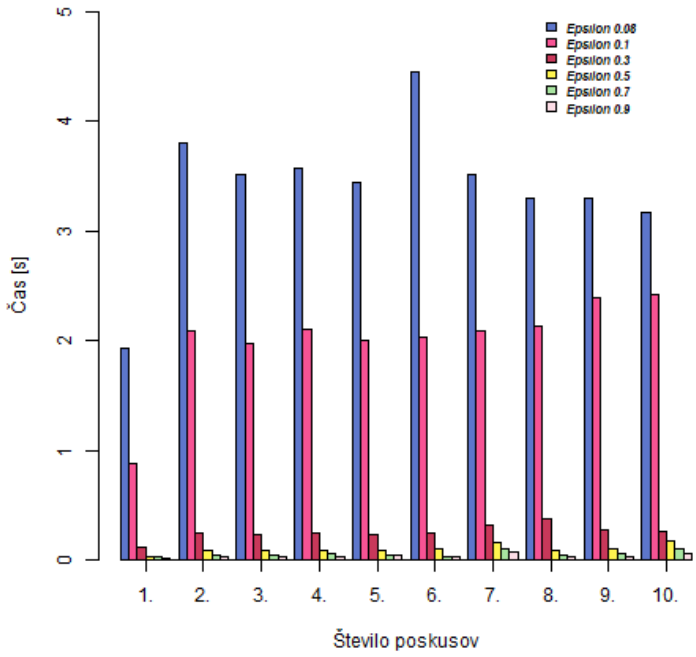
V eksperimentalnem delu sva poskušala ugotoviti ali pri raličnih vhodnih podatkih prihaja do sprememb časovne zahtevnosti. Bolj natančno, zanimala naju je hitrost izračunane rešitve pri različnih vrendostih epsilona.
Najprej sva sestavila kodo za generiranje podatkov, ki jo lahko najdete v datoteki *program.py*. Koda je napisana na način, da podatke na koncu shrani v excelovo datoteko.

Generirane podatkov lahko razdelimo v tri skupin in sicer glede na število predmetov, ki jih imamo v nahrbtniku. V prvem nahrbtniku je 5 predmetov, v drugem 10 in v tretjem 20. Nato pa sva si izbrala nekaj vrednosti epsilonov in za vsako vrednost naredila 10 poskusov.

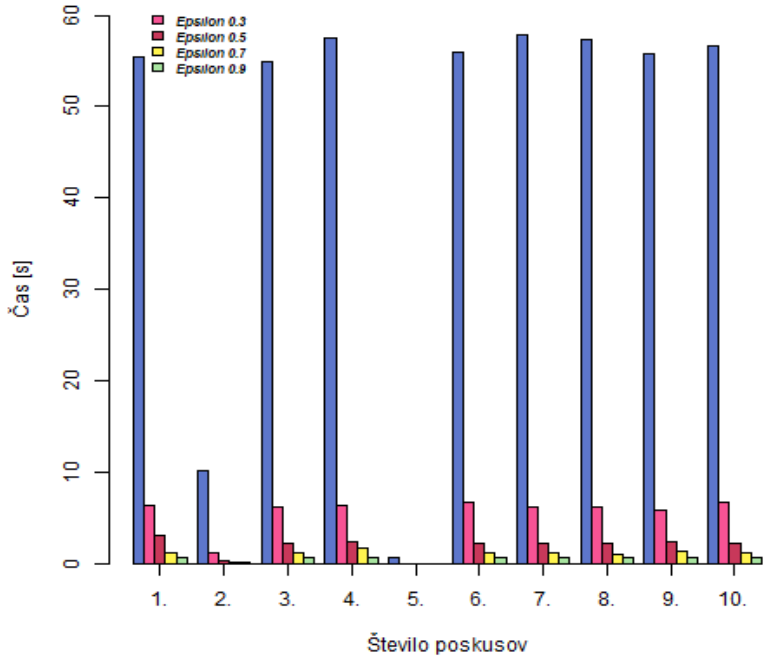
Ruzultati so v skladu s tem kar sva pričakovala. Bolj natančno kot želimo rešitev, torej manjši epislon kot si izberemo več časa je potrebnega. Seveda pa čas narašča tudi z naraščanjem števila predmetov v nahrbtniku.

Rezultate so zbrani v spodnjih grafih.

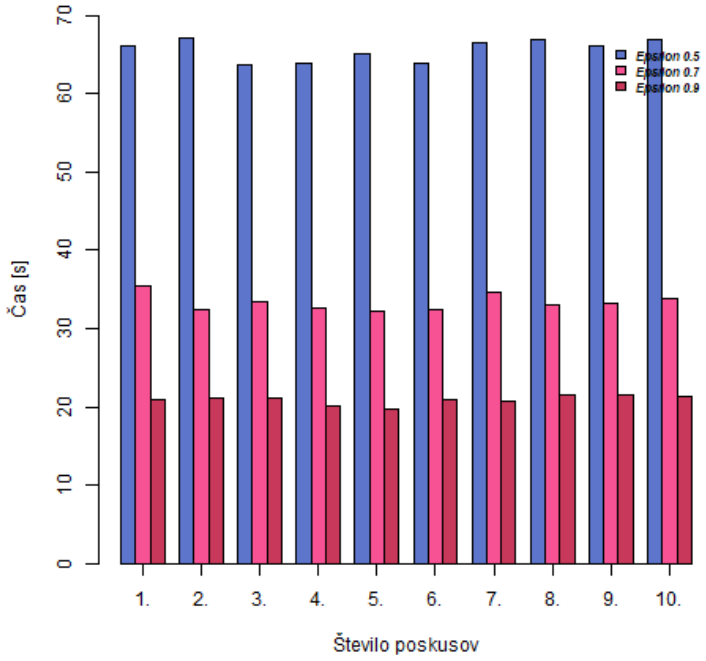
Slika 1: Eksperiment : 5 predmetov in 10 poskusov



Slika 2: Eksperiment :10 predemtov in 10 poskusov



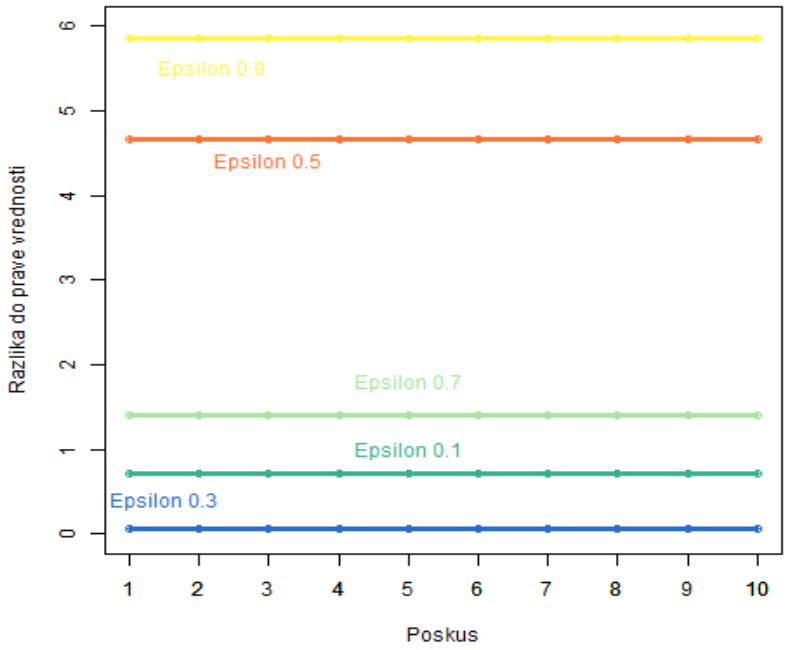
Slika 3: Eksperiment: 20 predmetov in 20 poskusov



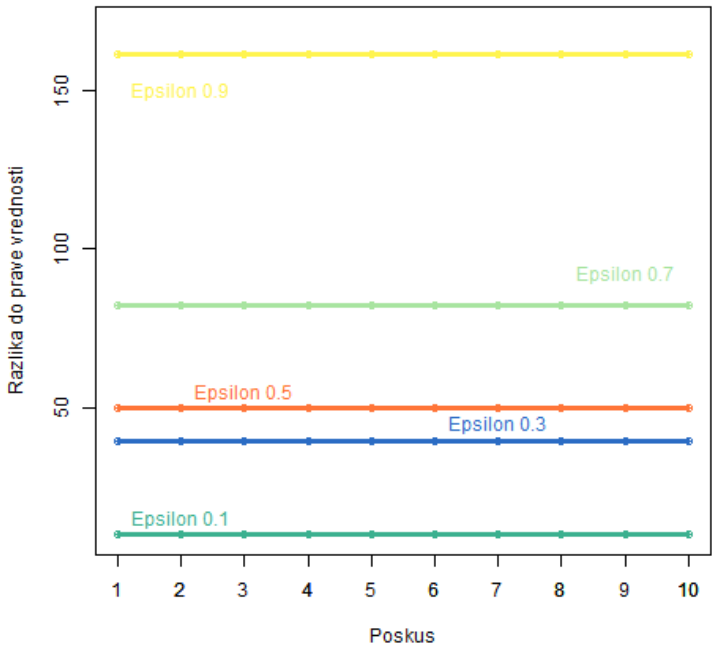
V zadnjem delu eksperimentalnega dela pa sva še naredila primerjavo točnih vrednosti problema in pa izračunane približke števila rešitev in prišla do zanimive ugotovitve.

V tem delu eksperimenta sva izbrala nahrbtnik 10 predmetov in nahrbtnik 20 predmetov. Rezultati so pokazali, da je odstopanje enako ne glede na teže predmetov v naboru desetih oziroma 20 predmetov.

Slika 4: Razlika med točno vrednostjo in približkom za 10 predmetov



Slika 5: Razlika med točno vrednostjo in približkom za 20 predmetov



5 Viri

- Štefankovič, Vempala, Vigoda (2012). A deterministic polynomial time approximation scheme for counting knapsack solution.