Naključnost v pokrivanju košev

(Randomness in Bin Covering)

Jakob Beber

June 30, 2025

Abstract

V članku predstavimo sprotno različico problema pokrivanja košev in možne strategije za njegovo reševanje. Kasneje se osredotočimo na strategijo naslednji usrezen, pri kateri ugotavljamo, da je analiza najslabšega primera preveč pesimistična za praktično uporabo. Da bi to raziskali, smo razvili spletno platformo za izvajanje in preizkušanje strategij ter različnih generatorjev zaporedij. V naših empiričnih rezultatih prikažemo razliko med teoretično najslabšim in "povprečnim" primerom uspešnosti za strategijo DNF.

1 Uvod

Pri *problemu pokrivanja košev* imamo podano zaporedje predmetov velikosti med 0 in 1, ki jih moramo porazdeliti v koše tako, da jih pokrijemo čim več. Pri tem je koš pokrit, če je seštevek velikosti predmetov odloženih vanj, vsaj 1.

Vsebino naloge predstavljamo z izmišljenim problemom, s katerim bo lažje razumeti namen, raziskovanje in rezultate te naloge.

1.1 Motivacijski primer

V vasi Kosevnice so vaščani zgradili poseben super računalnik. Računalnik je poseben v tem, da lahko privarčuje ogromno energije, če na njem deluje kar se da veliko procesorjev. Da lahko procesor deluje, mora biti vsaj minimalno obremenjen. Vsi vaščani si želijo uporabljati super računalnik, a bi radi hkrati plačali najcenejšo ceno za uporabo. Zato so pooblastili župana, da sestavi program, ki bo razporejal delovne procese po računalniku. Županov program mora delovne procese porazdeliti po super računalniku tako, da bo v kateremkoli trenutku delovalo kar se da veliko procesorjev. Tako bo župan zagotovil nizke cene uporabe računalnika za vse vaščane.

Ta izmišljen problem dobro modelira tudi probleme v resničnem svetu, kjer moramo porazdeliti podatke med največ možnih procesorjev, kjer procesor za delovanje potrebuje minimalno količino podatkov.

2 Definicija problema in opredelitev

Formalno definirajmo *problem pokrivanja košev*, ki ga rešujemo v motivacijskem primeru.

V problemu pokrivanja košev imamo podano zaporedje $\sigma = (v_1, v_2, ..., v_i)$ predmetov $v_i \in (0, 1)$ Predmete v_i iz zaporedja σ zlagamo v koše $K_1, K_2, ..., K_j$, kjer je $1 \leq j < |\sigma|$. Polnost koša je skupna velikost predmetov v njem

$$pol(K) = \sum_{v \in K} v.$$

Če je polnost koša vsaj 1, koš postane *pokrit*.

Pokrit koš pomeni, da ima procesor na super računalnku vsaj minimalno dela, hkrati pa župan želi, da deluje čim več procesorjev. Problem, ki ga rešujemo pri *problemu pokrivanja košev*, je razporeditev predmetov v koše tako, da je število pokritih košev čim večje.

2.1 Optimizacijski problem

Za vsako vhodno zaporedje predmetov (σ), obstaja rešitev, ki razvrsti predmete po koših tako, da je število pokritih košev maksimalno. Taki rešitvi rečemo *natančna rešitev*, algoritmu, ki pa bi to uspel doseči pa *optimalni* algoritem in ga označimo z OPT. Zaradi računske težavnosti problema (problem pokrivanja košev je NP-težak [1]), je optimalni algoritem nedosegljiv. Ta zapletenost nas sili v iskanje alternativnih pristopov, kot so hevristične strategije, da bi učinkovito našli dobre rešitve.

2.2 Hevristična strategija

Hevristična strategija je algoritem, ki je zasnovan tako, da najde zadovoljivo rešitev. Ta vrsta strategije ne zagotavlja optimalnega rezultata, ampak je njen cilj najti kakovostno rešitev v praktičnem času.

2.3 Konkurenčno razmerje

Da lahko hevristično strategijo ovrednotimo, predstavimo *konkurenčno razmerje*. To razmerje nam pove, kako dobra je naša strategija v primerjavi z optimalno strategijo.

Označimo našo strategijo z A. Število pokritih košev, ki nam jih strategija A vrne za dano zaporedje σ, označimo z $A(\sigma)$. Strategijo A ovrednotimo z njenim *konkurenčnostnim* razmerjem R, ki je vrednost, katera zadošča neenakosti

$$A(\sigma) \geq R * OPT(\sigma) - C, \tag{1}$$

za vsa končna zaporedja σ in neko konstanto C. Če je $C = 0$, rečemo da je konkurenčno razmerje *strogo* (angl. *strict*), drugače pa je *asimptotično*. Konkurenčno razmerje je pozitivno realno število $R \leq 1$, kjer $R = 1$ pomeni, da je strategija A optimalna.

2.4 Sprotni problem

Županov program v motivacijskem problemu mora delovne procese razporediti takoj v določen procesor na računalniku. Če predmeti prihajajo eden za drugim in moramo vsak predmet razporediti v koš takoj, ko pride na vrsto, govorimo o *sprotni različici* (angl. *online version*) problema pokrivanja košev. Zaporedju σ dodamo tri metode: $odpriZaporedje$, $naslednjiPredmet$ in $konecZaporedja$. Z metodo $odpriZaporedje$ zaporedje pripravimo za branje, metoda $naslednjiPredmet$ nam dá naslednji predmet v zaporedju, metoda $konecZaporedja$ sporoči, če smo na koncu zaporedja. Za razliko od osnovne različice, moramo pri sprotni različici koš najprej odpreti, če želimo vanj odlagati predmete. Ko koš odpremo, ima polnost 0, in rečemo, da je *odprt*. Predmet moramo odložiti v odprt koš, preden pride naslednji predmet. Koš je odprt, dokler ga ne zapremo. Zapremo pa ga, ko postane pokrit. Zaprtega koša ne moremo več odpirati. Odloženih predmetov v koših ne smemo več premikati.

V nadaljevanju naloge se bomo osredotočili le na *sprotno različico* problema.

2.5 Orakel

V sprotni različici problema pokrivanja košev ne vemo, ne koliko, ne kakšne predmete še prejmemo. Zaradi premočnega nasprotnika je podan problem za strategijo preveč omejujoč, ki ji zato razširimo delovanje. Naši pomankljivi strategiji dovolimo, da prejme nasvet za delovanje, ki ji ga dá *Orakel* [2].

Orakel je vsevedni dobri mož, ki pomaga strategiji. Ker *Orakel* pozna celotno zaporedje predmetov σ, lahko pred začetkom delovanja našemu algoritmu posreduje vnaprej določene informacije. Takšni obliki pomoči rečemo *nasvet na traku* (angl. *advice-on-tape model*). Strategija lahko kadar koli med izvajanjem bere iz tega traku in se pri tem odloča. Število bitov, ki jih strategija prebere s traku, je opredeljeno kot *svetovalna kompleksnost* [3]. S tem modelom lahko določimo, koliko informacij o prihodnosti potrebujemo, da bi zagotovo dosegli boljše konkurenčno razmerje.

3 Strategije reševanja problema

V motiacijskem primeru mora župan sestaviti program, ki bo čim boljše razporejal prihajajoče delovne procese na procesorje super računalnika. V temu poglavju predstavimo tri različne pristope reševanja, od preprostih strategij, do strategije, ki uporablja pomoč.

3.1 Naslednji ustrezen

Strategija naslednji ustrezen (angl. *Dual Next Fit* oz. *DNF*) je bila prva predlagana strategija za rešitev problema pokrivanja košev. Izvira iz svojega duala Next Fit, kjer je bila uporabljena v problemu pakiranja košev. Assmann *in sodelavci* [1] so dokazali, da je konkurenčnost strategije 1/2. Strategijo bi uspeli razviti tudi sami, saj je izpeljava zelo enostavna.

3.1.1 Delovanje strategije

Delovanje strategije na vhodnem zaporedju σ opišemo v naslednjih korakih:

Najprej `odpriZaporedje` σ.
Nato dokler ni `konecZaporedja` σ, delaj:

1. KORAK: Vzemi `naslednjiPredmet` v iz zaporedja σ.
2. KORAK: Če ni nobenega odprtega koša, odpri nov koš. Postavi predmet v v odprti koš in ga zapri, če postane pokrit.

Opazimo, da imamo pri tej strategiji vedno odprt en koš.

3.2 Harmonična strategija

Harmonična strategija (angl. *Dual Harmonic* oz. *DH*) je naslednja obravnavana strategija. Ideja, ki stoji za strategijo je, da lahko prihajajoče predmete klasificiramo po velikosti. Izberemo poljubni parameter k, ki porazdeli predmete po velikosti v k skupin:

$$\left(1, \frac{1}{2}\right], \left(\frac{1}{2}, \frac{1}{3}\right], \ldots, \left(\frac{1}{k-1}, \frac{1}{k}\right], \left(\frac{1}{k}, 0\right) \tag{2}$$

Konkurenčnost strategije je $1/2$, kot so to dokazali Epstein *in sodelavci* [4].

3.2.1 Delovanje strategije

Iz zaporedja σ prejmemo predmet in ga na podlagi njegove velikosti razvrstimo v pripadajočo skupino. Med delovanjem imamo vedno odprtih natanko k košev, po enega na skupino. V določen koš polnimo predmete samo iz enake skupine. Za polnenje košev znotraj vsake skupine uporabimo strategijo naslednji ustrezen - v koš odložimo predmet, ko je koš pokrit, ga zapremo in odpremo novega. Opazimo, da je koš skupine

$$\left(\frac{1}{i-1}, \frac{1}{i}\right], \ 1 < i \leq k,$$

poln natanko tedaj, ko je v njem i predmetov, katerih skupna velikost je največ $i/(i-1)$. Posledično je višek največ

$$\frac{i}{i-1} - 1 = \frac{1}{i-1} \ .$$

3.3 Strategija z natančno pomočjo

Da lahko sestavimo to strategijo, predstavimo $odpriZaporedje \rightarrow < nasvet >$, obliko pomoči strategiji, ki jo imenujemo *nasvet na traku*. To pomoč bo nudil *Orakel*, ki pozna celotno vhodno zaporedje. Podano pomoč lahko v strategiji uporabimo kadarkoli in kolikokratkoli si želimo skozi reševanje problema.

3.3.1 Nasvet na traku

Strategija *Oraklovo* pomoč posebej obravnava v skupini $(1, 1/2]$, in sicer polovico predmetov, ki so največji, je smiselno, da so sami v košu in ne po dva, kot predlaga strategija naslednji ustrezni. Tako *Orakel* za vhodno zaporedje σ strategiji posreduje nasvet na traku $< m, x_m >$. *Orakel* prešteje vse predmete v zaporedju, ki imajo velikost večjo od $1/2$ in polovico vseh preštetih predmetov predstavlja m. Velikost najmanjšega predmeta od največjih m predmetov pa predstavlja x_m. Z zapisom samo polovice vseh predmetov, večjih od $1/2$, strategiji omogočimo, da z ostalo polovico predmetov vedno uspe pokriti vsaj m košev.

3.3.2 Opis strategije

Strategija uporabi $< m, x_m >$, da odpre m košev, ki jim rečemo *kritični koši*. Prvotno kritičnim košem nastavimo *navidezno polnost* veliko x_m. V kritične koše lahko odložimo le en predmet velikosti $\geq x_m$. Ko tak predmet odložimo v kritični koš, se navidezna polnost koša posodobi na dejansko velikost odloženega predmeta. Prav tako izberemo poljuben parameter k, ki odpre k košev z enako porazdelitvijo predmetov po velikosti v k skupin, kot to stori harmonična strategija v (2). Nadaljevanje opišemo sledeče:

> Najprej `odpriZaporedje` $\sigma \rightarrow < m, x_m >$.
> Nato dokler ni `konecZaporedja` σ, delaj:
>
> 1. KORAK: Vzemi `naslednjiPredmet` v iz zaporedja σ.
> 2. KORAK: Če je v večji ali enak x_m, položi predmet v v naslednji kritični koš, ki še nima takega predmeta in posodobi navidezno polnost koša.
> 3. KORAK: Če je v manjši od x_m in večji ali enak $1/k$, položi predmet v njegov pripadajoči velikostni razred koša z uporabo strategije naslednji ustrezen.
> 4. KORAK: Če je v manjši od $1/k$, položi predmet v kritični koš, ki še nima navidezne polnosti večje od 1, in posodobi navidezno polnost. Če imajo vsi kritični koši navidezno polnost vsaj 1, položi predmet v koš k z uporabo strategje naslednji ustrezen.

To ponavljamo, dokler na vhodnem zaporedju ne dosežemo *konecZaporedja*.

Predstavljena strategija boljše razporeja prihajajoče predmete in tako uspe pokriti večje število košev. Konkurenčnost opisane strategije A predstavimo v obliki neenačbe (1), katero so dokazali Brodnik *in sodelavci* [3]

$$A(\sigma) \geq \frac{2}{3} * OPT(\sigma) - \frac{173}{60}.$$

4 Analiza DNF

Strategija naslednji ustrezni je izjemno preprosta strategija, a je zaradi slabe konkurenčnosti pogosto prezrta. Oglejmo si, kako nasprotnik (angl. adversary) uspe strategijo DNF prisiliti v $R = 1/2$, in kako lahko konkurenčnost podrobneje raziskujemo, če nasprotnika nekoliko omejimo.

4.1 Slabo zaporedje

Če pri kateremkoli ustvarjenem zaporedju uredimo predmete po velikosti od največjega do najmanjšega, strategijo naslednji ustrezen (angl. Dual Next Fit) prisilimo v najslabšo konkurenčnost za tisto specifično ustvarjeno zaporedje.

Ker se konkurenčnost strategij meri kot minimum skozi vsa možna zaporedja σ, pa se moramo pri ustvarjanju takšnega zaporedja držati naslednjih omejitev:

1. Zaporedje σ je dolžine $|\sigma| = i$, kjer imamo i predmetov.
2. Imamo $i/2 = n$ predmetov, velikosti $1/n$. Rečemo jim majhni predmeti.
3. Hkrati imamo $i/2 = n$ predmetov velikosti $1 - 1/n$, ki jim rečemo veliki predmeti.

Optimalna rešitev takega zaporedja je $OPT(\sigma) = n$. Če zaporedje uredimo od največjega predmeta do najmanjšega, pri strategiji naslednji ustrezen dosežemo konkurenčnost $DNF(\sigma) = n/2$.

4.2 Naključna zaporedja

V primerih iz realnega sveta, so predmeti v zaporedjih velikokrat bolj naključni. Težko se zgodi, da bomo prejemali nekaj časa samo velike predmete, na koncu pa samo najmanjše. Ti predmeti so pogosto pomešani med sabo.

Prav tako so omejitve z velikostjo majhnih predmetov nerazumne. Ko zaporedoma razvrščamo predmete v koše, ne poznamo dolžine zaporedja, zato v praktičnem primeru velikost majhnih košev ni odvisna od dolžine zaporedja. V empiričnem primeru definiramo majhne predmete z velikostjo med 0 in 0.3 in tako nismo odvisni od dolžine zaporedja.

Distribucija majhnih in velikih predmetov je v "slabem" zaporedju modelirana v škodo strategiji, a v resničnem svetu redko vidimo take distribucije.

Tako dobimo zaporedja, ki so v naravi pogostejša in jih je smiselno bolj podrobno analizirati.

5 Platforma

Da lahko predstavljene strategije ter poljubna zaporedja testiramo na dejanskih primerih, smo razvili spletno platformo "Bin Covering Interactive" [5]. Spletna platforma vsebuje možnost testiranja ali implementacijo strategij, generatorja zaporedji ali poljubno skripto. Za testiranje delovanja oziroma razvoj potrebujemo znanje Python programskega jezika ter razumeti arhitekturo delovanja platforme.

5.1 Arhitektura platforme

Na platformi sta vnaprej pripravljena dva osnovna razreda: strategija in generator (glej sliko 1). Oba razreda sta samo ogrodje, na katerih lahko razvijemo poljubno strategijo ali ustvarjalca zaporedji. Vsak razred pa vsebuje 4 obvezne metode za delovanje: *init*(), *start*(), *next*() in *stop*(). S temi javnimi metodami lahko razreda med seboj komunicirata in ne razkrivata drug drugemu informacij.

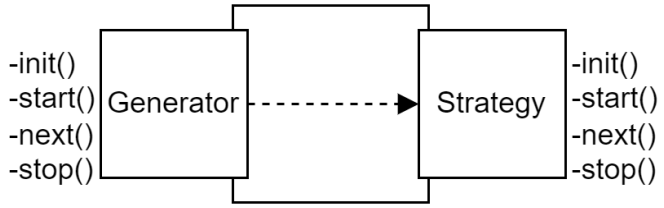

Figure 1: Delovanje strategije naslednji ustrezen.

Sami smo implementirali strategij DNF in DH ter poseben generator predmetov, kjer vemo, kakšen je rezultat strategije $OPT(\sigma)$. Zaradi razvoja generatorja in celotne platforme, smo obsežno primerjali konkurenčna razmerja DNF strategije na različnih vhodnih zaporedji. Rezultate predstavimo v šestem poglavju.

5.2 Implementacija strategije DNF

Psevdokoko za implementacijo strategije predstavimo v algoritmu 1. Uporabimo metodo *next*() iz razreda *Generator*, ki pošlje predmet iz zaporedja σ. Predmet odlagamo v *bin_load*. Vsakič, ko predmet odložimo, preverimo, ali je *bin_load* postal pokrit. Če *bin_load* postane pokrit, ga prištejemo k parametru $full\_bins$ in ponastavimo *bin_load* na 0. Z metodo *stop*() ustavimo delovanje teh dveh korakov in vrnemo parameter $full\_bins$.

Algorithm 1: Generator Class Implementation

Input: Class *Generator*
Output: Number of full bins

```
1  Function init():
       Data: bin_load ← 0
       Data: full_bins ← 0
2  Function start(generator):
3      self.gen ← generator
4  Function next():
5      bin_load ← bin_load + item
6      if bin_load > BIN_COVER_LOAD then
7          full_bins ← full_bins + 1
8          bin_load ← 0
9  Function stop():
10     return full_bins
```

5.3 Implementacija generatorja predmetov

Implementacija generatorja je na voljo na spletni platformi. Z psevdokodo se bomo posvetili najbolj pomembni metodi v razredu, *generate_numbers*, ki jo predstavimo v algoritmu 2. Metoda *generate_numbers* sprejme parameter *bins_covered*, ki pove, koliko košev bo vrnila strategija $OPT(\sigma)$ za ustvarjeno zaporedje σ. Predmete zaporedja naključno generiramo in jih odštevamo od polnega koša. Ko koš doseže mejo "praznine" (to definiramo s sprejemljivko *threshold*) določimo velikost še zadnjega predmeta, ki bo izpraznil koš. Z metodo razreda *Generator.next*(), pošilja generator po en predmet iz zaporedja σ. Ko smo prejeli vse predmete, pokličemo metodo *Generator.stop*(), da prejmemo celo število pokritih košev.

Algorithm 2: Generate Numbers Function

```
Input: Integer bins_covered
Output: List of generated numbers numbers
Function generate_numbers(bins_covered):
    numbers ← []
    bin_load ← _max_load
    i ← 0
    threshold ← 0.3 ;                          // fixed threshold
    while i < bins_covered do
        random_number ←
          get_number((0.3 * (_max_load - 1)), _max_load - 1)
        bin_load ← bin_load − random_number
        if bin_load < (threshold ∗ _max_load) then
            numbers.append(random_number)
            numbers.append(bin_load)
            bin_load ← _max_load
            i ← i + 1
        else
            numbers.append(random_number)
    return numbers
```

6 Rezultati

V obsegu te naloge smo se osredotočili, kako se strategija nasledni ustrezni obnaša na naključnih zaporedjih σ. V tabeli 1 lahko vidimo, kakšne možnosti generiranja predmetov smo podali razredu *Generator*. S temi možnostmi, smo ustvarili vhodna zaporedja, nato pa na njih naredili transpozicije. Transpozicije smo opravili tako, da niti en predmet ni ostal na istem mestu v zaporedju.

V testiranjih nas je zanimalo, kako se povprečje konkurenčnosti spreminja, s povečevanjem velikosti zaporedja $|\sigma|$. Rezultati (glej tabelo 2) so nas nekoliko presenetili. Kljub povečevanju $|\sigma|$ in številu permutacij, narejenih na zaporedju, lahko vidimo, da se konkurenčnost strategije ne spreminja veliko. Opazimo tudi, da je konkurenčnost strategije naslednji usrezni tudi veliko boljša z naključnimi vhodnimi zaporedji.

Table 1: Možnosti izbire predmetov v ustvarjanju zaporedja

Predmet velikosti v %	Možnost izbrane dane velikosti v %
0 - 30	50
30 - 70	8,3
70 - 100	41,7

Table 2: Rezultat strategije naslednji ustrezen, glede na OPT

$OPT(\sigma)$	$\lvert\sigma\rvert$	št. permutacij	$avg(DNF(\sigma))$	avg in %
300	841	100	228,42	76,14
600	1682	100	458,35	76,3916667
1200	3393	100	915,67	76,3058333
2400	6713	100	1828,57	76,1904167
4800	13517	100	3655,13	76,1485417
5000	14192	100	3814,55	76,291
10000	28434	100	7635,69	76,3569
20000	56607	100	15263,58	76,3179
10000	28301	1000	7630,389	76,3039
10000	28419	1000	7634,684	76,34684
20000	56462	1000	15248,224	76,24112
40000	113218	100	30524,96	76,3124
50000	142030	1000	38170,936	76,341872
80000	226920	100	61071,49	76,3393625
100000	283457	100	76336,49	76,33649
100000	283717	1000	76340,721	76,340721
300000	850230	100	228969,36	76,32312
500000	1416167	1000	381605,289	76,3210578

7 Zaključek

Kljub našim empiričnim ugotovitvam, ki kažejo veliko boljše konkurenčno razmerje za naključna zaporedja, ostaja matematični dokaz strožjih mej DNF strategije odprt problem. Spletna platforma nam ponuja dragoceno orodje za izvajanje in ocenjevanje, tako obstoječih, kot tudi novih strategij pri različnih porazdelitvah zaporedij.

Literatura

[1] Susan F Assmann, David S. Johnson, Daniel J. Kleitman, and JY-T Leung. On a dual version of the one-dimensional bin packing problem. *Journal of algorithms*, 5(4):502–525, 1984.

[2] Joan Boyar, Lene M Favrholdt, Shahin Kamali, and Kim S Larsen. Online bin covering with advice. *Algorithmica*, 83:795–821, 2021.

[3] Andrej Brodnik, Bengt J Nilsson, and Gordana Vujovic. Online bin covering with exact parameter advice. *arXiv preprint arXiv:2309.13647*, 2023.

[4] Leah Epstein, Lene M Favrholdt, and Jens S Kohrt. Comparing online algorithms for bin packing problems. *Journal of Scheduling*, 15:13–21, 2012.

[5] J. Beber. Bin covering. https://github.com/jakobb01/BinCovering/tree/main/web_interface, 2025. Accessed: 2025-06-28.