UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Maja Remic

Problem polnjenja košev v povezavi z omejitvijo kardinalnosti

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Borut Robič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01735/2011
Datum: 15.03.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MAJA REMIC**

Naslov: **PROBLEM POLNJENJA KOŠEV V POVEZAVI Z OMEJITVIJO KARDINALNOSTI**

CARDINALITY CONSTRAINED BIN PACKING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Preglejte razne oblike problema polnjenja košev, sorodne in izpeljane probleme in njihove časovne zahtevnosti. Predstavite aproksimacijske algoritme za problem polnjenja košev, v izbrane med njimi vgradite dodatno omejitev kardinalnosti in jih implementirajte. Eksperimentalno primerjajte implementirane algoritme z algoritmi, ki so bili razviti posebej za reševanje problema polnjenja košev z omejitvijo kardinalnosti.

Mentor:

prof. dr. Borut Robič

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Maja Remic,

z vpisno številko 63040144,

sem avtor/-ica diplomskega dela z naslovom:

Problem polnjenja košev v povezavi z omejitvijo kardinalnosti

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom prof. dr. Boruta Robiča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 13.6.2011 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se

mentorju prof. dr. Borutu Robiču za vse, kar me je naučil, in za pomoč pri pisanju diplomske naloge,

prof. dr. Janezu Žerovniku za idejo, nasvete in pregled

ter

Gašperju Žerovniku.

Kazalo

Seznam uporabljenih kratic in simbolov

AF Any Fit (poljubno ujemanje)

BF Best Fit (najboljše ujemanje)

BFD Best Fit Decreasing (padajoče najboljše ujemanje)

FF First Fit (prvo ujemanje)

FFD First Fit Decreasing (padajoče prvo ujemanje)

NF Next Fit (naslednje ujemanje)

RFF Refined First Fit (izboljšano prvo ujemanje)

WF Worst Fit (najslabše ujemanje)

Povzetek

Polnjenje košev je NP-težek optimizacijski problem, kako predmete danih velikosti razporediti v minimalno število košev omejene kapacitete. Poleg osnovnega problema obstaja še veliko drugih različic. Pri omejitvi kardinalnosti postavimo dodatno omejitev, da število predmetov v košu ne sme preseči določene meje N_{max}.

Opisani so nekateri znani algoritmi za reševanje splošnega problema polnjenja košev, pa tudi trije specifični algoritmi za reševanje problema z omejitvijo kardinalnosti. Algoritma FFD in RFF ter Zhangov algoritem so primerjani s tremi specifičnimi algoritmi na naključnih seznamih z 0%, 10%, 30% in 50% velikih predmetov. Preučeno je tudi obnašanje vseh algoritmov ob povečevanju N_{max}.

Rezultati kažejo, da na seznamih z nizkim deležem velikih predmetov vsi trije specifični algoritmi dosegajo boljše rezultate od splošnih. Eden od specifičnih algoritmov dosega boljše ali primerljive rezultate tudi na seznamih z visokim deležem velikih predmetov in je zato posebej zanimiv. Obnašanje ob povečevanju N_{max} kaže, da so vsi trije specifični algoritmi uporabni tudi za reševanje splošnega problema polnjenja košev.

Ključne besede:

problem polnjenja košev z omejitvijo kardinalnosti, aproksimacijski algoritmi, primerjava, FFD, RFF, Zhangov algoritem.

Abstract

Bin packing is an optimizational NP-hard problem of packing items of given sizes into minimum number of capacity-limited bins. Besides the basic problem, numerous other variants of bin packing exist. The cardinality constrained bin packing adds an additional constraint that the number of items in a bin must not exceed a given limit N_{max}.

Some well-known algorithms for solving the general bin packing problem are described, along with three specific algorithms for solving the cardinality constrained bin packing problem. The FFD, RFF and Zhang's algorithms are compared to the three specific algorithms on random lists of items with 0%, 10%, 30% and 50% of large items. The behaviour of all algorithms when N_{max} increases is also studied.

Results show that all three specific algorithms outperform the general algorithms on lists with low percentage of large items. One of the specific algorithms performs better or equally even on lists with high percentage of large items and is therefore of significant interest. The behaviour when N_{max} increases shows that all three specific algorithms can be used for solving the general bin packing problem as well.

Key words:

cardinality constrained bin packing problem, approximation algorithms, comparison, FFD, RFF, Zhang's algorithm.

Poglavje 1

Uvod

V vsakodnevnem življenju se pogosto srečujemo z željo ali potrebo po prihranku časa, prostora ali sredstev. Postavljamo si vprašanja, podobna naslednjim:

- Kako razporediti svoje osebne predmete v omare, ki so nam na razpolago? Posebno ljudje, ki živijo v stanovanjih, vedo, da gre za precej pereč problem, saj se zdi, da omar ni nikoli dovolj za vse stvari, ki jih želimo v njih hraniti.
- Kako prepeljati tovor na novo lokacijo s čim manjšim številom tovornjakov? Ta problem je lahko celo odločujoč dejavnik pri uspešnosti podjetja, saj stroški transporta niso zanemarljivi.
- Kako razporediti oglase v televizijskem bloku oglasov? Seveda si želimo prikazati kar največ oglasov v čim manj blokih, saj nam to zagotavlja največ oglaševalskih sredstev.

Poleg naštetih se pojavljajo še mnogi drugi problemi, povezani z razporejanjem, od razporejanja dela na strojih v proizvodnji do sestavljanja šolskih urnikov, od urejanja letalskega, avtobusnega in železniškega transporta do razreza neke surovine standardnih mer (npr. lesa) na želeno količino kosov različnih velikosti, od alokacije pomnilnika pri ostranjevanju do razvrščanja člankov na časopisne strani.

Jasno je, da rešitve vseh teh problemov lahko bistveno izboljšajo marsikateri aspekt kvalitete našega življenja. Problem *polnjenja košev*, s katerim se ukvarja ta diplomska naloga, ponuja dober model za rešitve takih in podobnih vprašanj.

Čeprav se zdijo vprašanja enostavna, sploh glede na pogostost njihovega pojavljanja v vsakdanjem življenju, se izkaže, da gre za težek problem, čas,

potreben za izračun natančne rešitve, pa zelo hitro postane neobvladljiv. Zato je nujno iskanje pristopov, ki na račun točnosti problem rešijo v primerno kratkem času, ob tem pa še vedno ponujajo zadovoljivo natančno rešitev.

Znanost se s problemom polnjenja košev intenzivno ukvarja že od zgodnjih 1970-tih. Problem je služil kot eden od temeljev za razvoj metod za analizo uspešnosti aproksimacijskih algoritmov, npr. performančnega kvocienta za najslabši primer in obnašanja algoritma v povprečnem primeru.

Poglavje 2

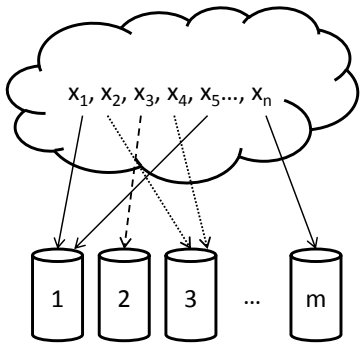
Problem polnjenja košev

2.1 Osnovni problem

Za začetek bomo predstavili in definirali osnovni problem polnjenja košev. Ogledali si bomo tudi, kako zahteven je ta problem s stališča časa, potrebnega za izračun natančne rešitve.

2.1.1 Definicija

Problem polnjenja košev (angl. *bin packing problem*) lahko opišemo na sledeč način: Imamo seznam n predmetov z določenimi velikostmi, ki jih želimo razporediti v koše z omejeno kapaciteto C_{max}. Pri razporejanju želimo doseči minimalno število košev m, tako da skupna velikost predmetov v nobenem košu ne preseže njegove kapacitete. Opisani problem ponazarja slika 2.1.



Slika 2.1: Shema polnjenja košev.

Razvidno je, da mora biti velikost posameznega predmeta nenegativna (smiselno je tudi, da je večja od 0) in ne sme preseči kapacitete koša. Predmeta ne moremo razdeliti v več košev.

Brez izgube splošnosti lahko zahtevamo $C_{max} = 1$, kar omeji velikost posameznega predmeta na vrednosti med 0 in 1. Takšna definicija problema je običajna v literaturi in se je bomo držali tudi v tem delu.

Definirajmo problem bolj formalno (povzeto po [3]):

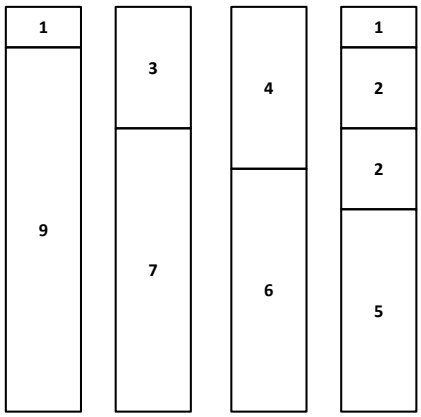
- Dan je seznam nenegativnih števil $x_1, \ldots, x_n \leq 1$.
- Iščemo $m \in \mathbb{N}$ in razdelitev $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$, da velja

$$\sum_{i:f(i)=j} x_i \leq 1 \qquad \text{za vsak } j \in \{1, \ldots, m\},$$

  pri čemer je m minimalen.

Vidimo, da problem sodi med *optimizacijske probleme*, t.j. probleme, pri katerih iščemo najboljšo možno rešitev neke naloge.

Polnjenje košev si oglejmo še na preprostem primeru, ki ga prikazuje slika 2.2. Zaradi lažje predstavljivosti so uporabljena cela števila in koši s kapaciteto, večjo od 1.



Slika 2.2: Primer problema polnjenja košev. Dan je seznam predmetov $X = (3, 6, 2, 1, 5, 7, 2, 4, 1, 9)$ in $C_{max} = 10$. Prikazana rešitev z $m = 4$ koši je optimalna.

2.1.2 Zahtevnost problema

Preden se lotimo zahtevnosti problema polnjenja košev, na kratko ponovimo pomembne definicije iz teorije zahtevnosti računskih problemov. Začnimo z dvema slavnima razredoma *odločitvenih problemov*, t.j. problemov, ki z da ali ne odgovarjajo na neko vprašanje:

P predstavlja razred problemov, ki jih lahko deterministično rešimo v polinomsko omejenem času. Ti problemi veljajo za računsko lahke.

NP predstavlja razred problemov, ki jih lahko v polinomsko omejenem času rešimo nedeterministično (rešitev uganemo), lahko pa v polinomsko omejenem času deterministično preverimo rešitev problema.

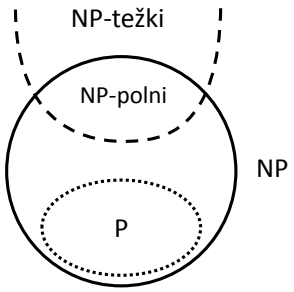
Spomnimo se še naslednjih treh definicij zahtevnosti problemov:

NP-težek problem : Za problem L_0 rečemo, da je NP-težek, če lahko vsak problem $L \in$ NP prevedemo na L_0 v polinomskem času.

NP-poln problem: Problem L_0 je NP-poln, če velja, da je NP-težek in da je hkrati $L_0 \in$ NP.

Krepko NP-poln problem: Problem je krepko NP-poln, če velja, da ostane NP-poln, čeprav vse njegove numerične parametre omejimo na polinomsko dolžino.

Razmerja med opisanimi pojmi bolj nazorno prikazuje slika 2.3.



Slika 2.3: Razmerja med razredi P, NP, NP-težki in NP-polni ob predpostavki P $\neq$ NP.

Dokazovanje NP-polnosti (NP-težkosti) poteka s pomočjo prevedbe nekega drugega problema, za katerega vemo, da je NP-poln (NP-težek), na naš problem.

Ker ne poznamo algoritma, ki bi lahko probleme iz razreda NP rešil v polinomskem času, se predpostavlja neenakost teh dveh razredov, torej P $\neq$ NP. Zavedati pa se moramo, da ta neenakost še ni bila dokazana (ali ovržena), zato vprašanje ostaja odprto. Polinomsko omejen čas se smatra za razumno rastočega in posledično obvladljivega, medtem ko je eksponenten čas neobvladljiv, zato sta ta dva razreda posebej zanimiva.

Problem polnjenja košev v svoji odločitveni obliki je NP-poln (pravzaprav je tudi krepko NP-poln). Velja naslednji izrek:

Izrek 2.1.1. *Naslednji problem je NP-poln: Pri danem primerku problema polnjenja košev I odloči, ali je I rešljiv z dvema košema.*

Dokaz. Dokaz izreka poteka s pomočjo prevedbe problema razdelitve (angl. *partition problem*), ki je NP-poln, na zgornji problem. Pri problemu razdelitve nas zanima, ali lahko dano množico celih števil razdelimo na dve podmnožici, tako da bosta vsoti števil v vsaki podmnožici enaki. Dokaz je povzet po [3]: Ob danem primerku problema razdelitve $c_1, \ldots, c_n$, sestavimo problem polnjenja košev, tako da je:

$$x_i = \frac{2c_i}{\sum_{j=1}^{n} c_j}.$$

Dva koša zadostujeta natanko takrat, ko obstaja podmnožica $S \subseteq 1, \ldots, n$, da velja $\sum_{j \in S} c_j = \sum_{j \notin S} c_j$. Zgornja sestava x_i namreč zagotavlja, da bo vsota vseh predmetov v seznamu natanko 2. Ker je kapaciteta posameznega koša omejena na 1, mora biti, če želimo uporabiti le 2 koša, vsota predmetov v vsakem košu natanko 1. To pa je možno le v primeru, ko obstaja podmnožica S. □

Ker je njegova odločitvena oblika NP-poln problem, je polnjenje košev, kakor še veliko drugih praktičnih problemov, NP-težek optimizacijski problem.

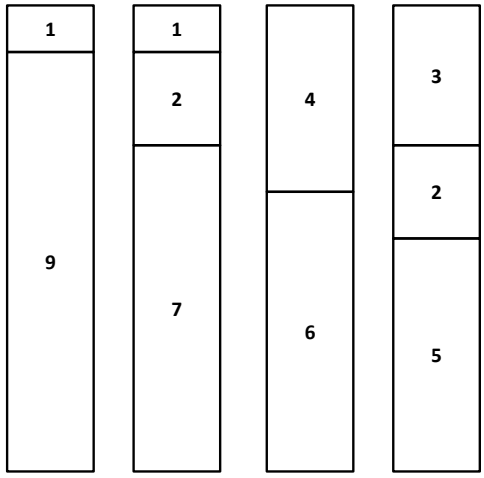
2.2 Izpeljani problemi

V poglavju 2.1 smo si ogledali najosnovnejši problem polnjenja košev. Poleg njega pa obstaja še veliko izpeljanih problemov z dodatnimi zahtevami ali omejitvami. Nekatere med njimi (in še zdaleč ne edine) si bomo pogledali v nadaljevanju.

2.2.1 Omejitev kardinalnosti

Pri omejitvi kardinalnosti[1] (angl. *cardinality constrained bin packing*) osnovnemu problemu polnjenja košev dodamo zahtevo, da število predmetov v posameznem košu ne preseže maksimalnega dovoljenega števila predmetov v košu N_{max}.

Za primer poglejmo, ali lahko nalogo s slike 2.2 preuredimo tako, da zadosti dodatni omejitvi, da so v vsakem košu lahko največ trije predmeti. Rešitev prikazuje slika 2.4.



Slika 2.4: Primer polnjenja košev z omejitvijo kardinalnosti. Uporabljen je isti primer kot na sliki 2.2 z dodatno omejitvijo $N_{max} = 3$. Rešitev je še vedno optimalna.

Kot primer praktične uporabe naj navedemo problem polnitve vsebnikov z izrabljenim jedrskim gorivom, predstavljen v [11]: V trajnem odlagališču izrabljenega jedrskega goriva se gorivni elementi hranijo v posebnih vsebnikih (angl. *canisters*). Seveda želimo minimizirati število vsebnikov, saj večje število predstavlja višje stroške odlagališča, upoštevati pa moramo tudi, da lahko v vsak vsebnik spravimo le omejeno število elementov. Za rešitev problema uporabimo algoritme za reševanje problema polnjenja košev z omejitvijo kardinalnosti.

Ker se bomo s to vrsto izpeljanega problema podrobneje ukvarjali v praktičnem delu naloge, je prav, da problem polnjenja košev z omejitvijo kardinalnosti

[1]Kardinalnost končne množice je število elementov v množici.

še formaliziramo:

- Dan je seznam nenegativnih števil $x_1, \ldots, x_n \leq 1$ in maksimalno število predmetov v posameznem košu $N_{max} \in \mathbb{N}$.
- Iščemo $m \in \mathbb{N}$ in razpored $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$, da velja

  $$\sum_{i:f(i)=j} x_i \leq 1 \qquad \text{in} \qquad |\{x_i | f(i) = j\}| \leq N_{max} \qquad \text{za vsak } j \in \{1, \ldots, m\},$$

  pri čemer je m minimalen.

Problem polnjenja košev z omejitvijo kardinalnosti ostaja NP-težek, ravno tako kot osnovni problem. Formalni dokaz lahko najdemo v [11].

2.2.2 Omejitev razredov

Pri omejitvi razredov (angl. *class constrained bin packing*) vsakemu predmetu x_i dodamo še razred (oziroma barvo) $q_i \in \mathbb{N}$ iz nabora razredov Q. Osnovni problem dodatno omejimo z zahtevo, da število različnih razredov v posameznem košu ne preseže določenega števila $Q_{max} \in \mathbb{N}$. Zaradi smiselnosti dodatne omejitve se navadno zahteva $Q_{max} < |Q|$.

Primer uporabe polnjenja košev z omejitvijo razredov je področje videa na zahtevo, kjer želimo konstruirati strežnik, ki bo zmogel obdelati največje število zahtev glede na pričakovano popularnost filmov v bazi [8].

Z aplikativnega stališča so zanimive tudi naslednje variante omejitve razredov:

- V košu so lahko samo predmeti enega razreda. Primer se pojavi pri ločevanju odpadkov, kjer imamo koše za steklo, embalažo in papir.
- Dovoljene so samo kombinacije nekaterih razredov znotraj enega koša. Primer iz vsakodnevnega nakupovanja: v isti vrečki ni zaželeno nositi živil in pralnega praška.

2.2.3 Dvo- in večdimenzionalno polnjenje košev

Pri klasičnem dvodimenzionalnem (angl. *two-dimensional*) problemu polnjenja košev moramo dane predmete x_i pravokotne oblike razporediti v najmanjše število enakih košev kvadratne oblike. V osnovni problem torej dodamo še eno dimenzijo.

Podobno pri večdimenzionalnem (angl. *multi-dimensional*) problemu polnjenja košev dodamo več dimenzij. Posebej pogosta je tridimenzionalna oblika, pri kateri upoštevamo dolžino, širino in višino predmetov ter košev.

Težji problemi s tega področja dovoljujejo tudi rotacijo predmetov znotraj koša, kar znatno poveča težavnost iskanja zadovoljive rešitve, saj močno poveča prostor rešitev. Možno je tudi, da so predmeti nepravilnih oblik. Posebej tridimenzionalni problem ima veliko področij aplikacij, npr. pri razporejanju predmetov na tovornjake, v omare ipd.

2.2.4 Koši različnih velikosti

Pri tej varianti polnjenja košev (angl. *variable size bin packing*) dovoljujemo koše različnih velikosti. Navadno imajo različno veliki koši različno ceno, zato se naloga iz minimiziranja števila košev spremeni v minimiziranje skupne cene uporabljenih košev.

Kot primer navedimo razporejanje poslov med procesorje različnih zmogljivosti in posledično cen. Želimo, da so vsi posli končani v zadosti kratkem času, pri čemer želimo, da je cena procesorjev, ki jih potrebujemo za izvršitev vseh poslov, najmanjša možna.

2.3 Podobni problemi

Za konec tega poglavja si oglejmo še dva znana problema, ki se močno navezujeta na problem polnjenja košev.

2.3.1 Problem nahrbtnika

Problem nahrbtnika (angl. *knapsack problem*) je NP-težek optimizacijski problem, saj je odločitvena oblika NP-poln problem [5]. Imamo n predmetov z določenimi velikostmi (težami) in vrednostmi (cenami). Z njimi želimo napolniti nahrbtnik omejene nosilnosti C_{max}, tako da se le-ta ne bo strgal in bomo dosegli največjo ceno v njem.

Problem nahrbtnika (glej [6]) lahko torej formuliramo kot:

- Dana je množica predmetov $X = \{x_i | i = 1, \ldots, n\}$. Za vsak predmet x_i poznamo njegovo velikost $c_i \in \mathbb{Z}^+$ in vrednost $p_i \in \mathbb{Z}^+$. Poznamo tudi nosilnost nahrbtnika $C_{max} \in \mathbb{N}$.

- Želimo poiskati množico $Y \subseteq X$, da velja:

$$\sum_{x_i \in Y} c_i \leq C_{max} \qquad \text{in}$$

$$\sum_{x_i \in Y} p_i \text{ je maksimalna.}$$

Podobnost s polnjenjem košev je očitna, če si zamislimo, da lahko uporabimo več nahrbtnikov. Ta problem je znan pod imenom problem maksimalne kardinalnosti polnjenja košev (angl. *maximum cardinality bin packing problem*): Na voljo imamo M_{max} košev kapacitete C_{max} in n predmetov z določenimi težami, želimo pa maksimizirati število predmetov, ki jih lahko spravimo v koše, ne da bi presegli dano kapaciteto in število košev.

2.3.2 Problem razreza

Problem razreza (angl. *cutting-stock problem*) lahko opišemo na sledeči način: Na zalogi imamo neomejeno število kosov nekega materiala (npr. lesenih desk) standardnih dolžin $L_1, L_2, \ldots, L_k$. Želimo izpolniti naročila s strank, pri čemer vsako naročilo sestavljajo zahteve po n_i kosih dolžine l_i za $i = 1, \ldots, s$, tako da bomo porabili kar najmanj materiala oziroma da bo odpadnega materiala čim manj.

Če število različnih standardnih dolžin materiala omejimo na $k = 1$, postane problem razreza zelo podoben osnovnemu problemu polnjenja košev (tudi brez te zahteve pa obstaja podobnost s polnjenjem košev različnih velikosti). Glavna razlika med problemoma je, da imamo pri polnjenju košev predmete z veliko različnimi velikostmi, pri razrezu pa imamo predmete z le nekaj različnimi velikostmi. Razlog za uporabo različnih klasifikacij za ta dva problema je verjetno v tem, da so se za njuno reševanje tradicionalno uporabljale drugačne metode [1].

Poglavje 3

Aproksimacijski algoritmi

3.1 Lastnosti

Zaradi NP-težkosti problema polnjenja košev in splošnega prepričanja, da polinomski algoritem za iskanje rešitve za NP-težke probleme ne obstaja, je posebno pri problemih večjih razsežnosti nesmiselno stremeti k natančni rešitvi, saj bi njeno iskanje vzelo veliko preveč časa. Zato se raje zadovoljimo s približno rešitvijo, pri kateri na račun natančnosti pridobimo na hitrosti reševanja. Algoritmu, ki vrača približno rešitev nekega problema v okviru znanih maksimalnih napak, pravimo *aproksimacijski algoritem*.

3.1.1 Osnovni pojmi

Glede na način razporejanja predmetov in vpogled, ki ga imajo v prihajajoče predmete, lahko algoritme razdelimo v tri skupine:

1. *Sprotni algoritmi* (angl. *on-line*): Ti algoritmi razporejajo predmete v takem vrstnem redu, kot predmeti prihajajo, brez poznavanja naslednjih predmetov. Odločitev je dokončna: ko je predmet enkrat razporejen v koš, ga ni več možno premakniti.

2. *Naknadni algoritmi* (angl. *off-line*): Ti algoritmi pred začetkom razporejanja poznajo vse predmete v seznamu in jih po potrebi predhodno uredijo.

3. *Polsprotni algoritmi* (angl. *semi on-line*): Pri teh algoritmih so zahteve glede na sprotne algoritme nekoliko omiljene. Lahko je nekaj predmetov znanih vnaprej (prihajajo v paketih), lahko je dovoljeno premikanje nekaterih predmetov po razporeditvi v koš ipd.

Za kasnejše potrebe definirajmo še naslednja pojma:

Odprt koš je koš, v katerega je dovoljeno dodajati nove predmete. Tak koš ne sme biti poln.

Zaprt koš je koš, v katerega novih predmetov ni več dovoljeno dodajati. Tak koš je lahko poln, ni pa to obvezno.

Algoritem mora koš označiti za zaprtega, ko je le-ta poln, lahko pa ga za zaprtega označi že prej.

3.1.2 Kakovost algoritmov

Seveda se postavlja vprašanje, kolikšno je odstopanje rešitev aproksimacijskih algoritmov od optimalne rešitve OPT(I), ki predstavlja najmanjše število košev, ki jih potrebujemo za rešitev primerka problema polnjenja košev I. Zagotovo velja:

$$\mathrm{OPT(I)} \geq \lceil \sum_{i=1}^{n} x_i \rceil. \tag{3.1}$$

Oglejmo si dve meri za ocenjevanje kakovosti aproksimacijskih algoritmov:

Absolutni količnik najslabšega primera (angl. *absolute worst-case ratio*) R_A za aproksimacijski algoritem A, ki rešuje problem polnjenja košev, je podan kot:

$$R_A = \sup_I \{\mathrm{A(I)/OPT(I)}\}, \tag{3.2}$$

kjer A(I) označuje število košev, ki jih uporablja rešitev algoritma A za primerek I.

Asimptotični količnik najslabšega primera (angl. *asymptotic worst-case ratio*) R_A^∞ za algoritem A je podan kot:

$$R_A^\infty = \lim_{k \to \infty} \sup_I \{\mathrm{A(I)/OPT(I)} | \mathrm{OPT(I)} \geq k\}. \tag{3.3}$$

Za vsak algoritem očitno velja naslednje razmerje med asimptotičnim in absolutnim količnikom najslabšega primera:

$$R_A \geq R_A^\infty. \tag{3.4}$$

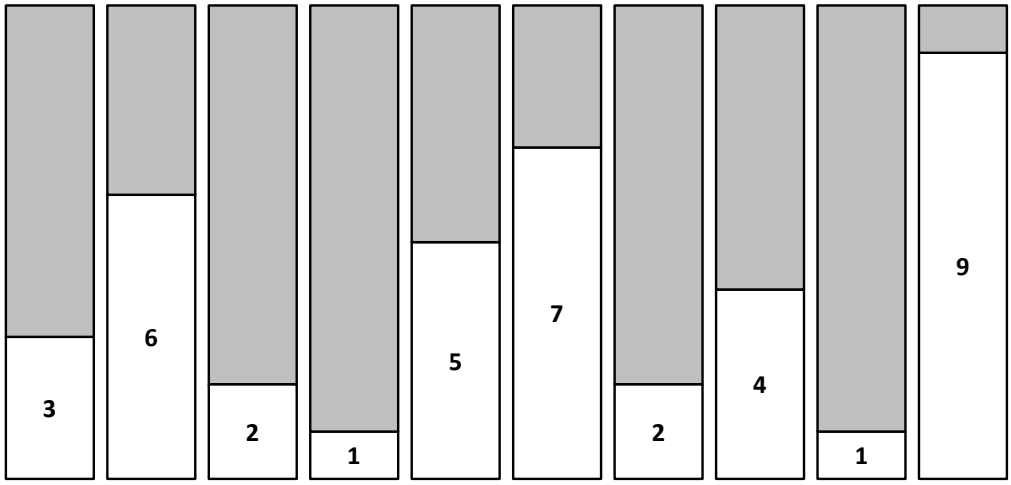
Za aproksimacijski algoritem A problema P pravimo, da je *r-aproksimacijski algoritem* za P, če obstaja konstanta $r \geq 1$, da je $R_A \leq r$ za vsak primerek problema I.

Izrek 3.1.1. *Za problem polnjenja košev ne obstaja ρ-aproksimacijski algoritem za $\rho < \frac{3}{2}$, razen če je* P = NP.

Dokaz. Izrek je posledica izreka 2.1.1 in je povzet po [7]. Privzemimo, da obstaja ρ-aproksimacijski algoritem za $\rho < \frac{3}{2}$. Spomnimo se problema razdelitve, ki smo ga v omenjenem izreku uporabili za dokaz NP-polnosti: za ta problem je OPT(I) = 2. Naš algoritem mora torej vrniti rešitev z $m < \rho\text{OPT(I)} = 3$ koši, torej lahko uporablja samo 2 koša. Tak algoritem bi našel optimalno rešitev in s tem natančno rešil problem razdelitve. □

3.2 Trivialna rešitev

Kakšen algoritem bi rešil problem polnjenja košev, kot je opisan v 2.1? Gotovo si lahko zamislimo preprost in precej naiven algoritem, ki vsak predmet razporedi v svoj koš. Rezultat na primeru s slike 2.2 (ta primer bomo uporabljali tudi za vse ostale algoritme) prikazuje slika 3.1. Siva polja označujejo prazne dele košev.



Slika 3.1: Rezultati primera s slike 2.2, če uporabimo trivialno rešitev, ki vsak predmet razvrsti v svoj koš.

Seveda nam je takoj jasno, da bodo rešitve tega algoritma zelo daleč od optimalnih, saj bo algoritem vsakič potreboval natanko n košev. V nadaljevanju si zato poglejmo nekaj preprostih algoritmov, ki dosežejo boljši rezultat.

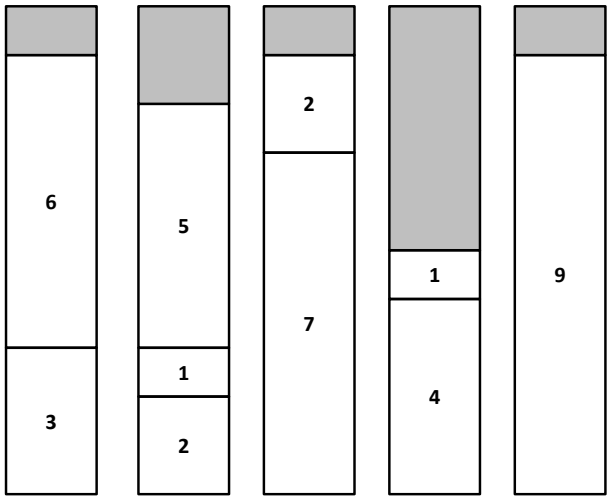
3.3 Sprotni algoritmi

Predvidevamo lahko, da sprotni aproksimacijski algoritmi ponujajo manjšo stopnjo optimizacije zaradi pomanjkanja vpogleda v naslednje predmete seznama. Vendar v določenih primerih iz realnega življenja nimamo možnosti uporabe algoritmov, ki zahtevajo poznavanje vseh predmetov pred začetkom razvrščanja.

Prva dva algoritma iz tega razdelka sodita med t.i. *požrešne algoritme*, kar pomeni, da na vsakem koraku izbereta možnost, ki se trenutno zdi najboljša.

3.3.1 Naslednje ujemanje

Algoritem NF (iz angl. *Next Fit*) poskuša nov predmet uvrstiti v trenutno odprti koš. Če to ni možno, trenutni koš zapre, odpre nov koš in predmet uvrsti vanj. V vsakem trenutku je torej odprt natanko en koš. Primer delovanja algoritma prikazuje slika 3.2.



Slika 3.2: Primer s slike 2.2 za algoritem NF. Vidimo, da NF v tem primeru potrebuje en koš več od optimalnega.

Algoritem NF ima linearno časovno zahtevnost $O(n)$ in konstantno prostorsko zahtevnost $O(1)$. Glede kvalitete rešitve pa veljajo naslednje trditve (povzeto po [2]):

1. Za vsak primerek I velja: $\mathrm{NF(I)} \leq 2\mathrm{OPT(I)} - 1$.

2. $R_{NF}^{\infty} = 2$.

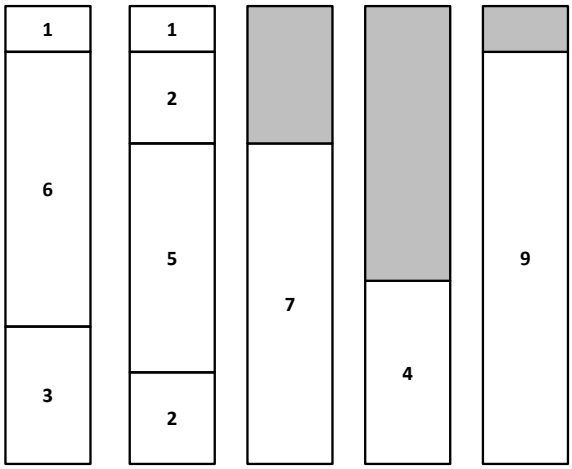
3. Količnik se izboljša, ko se velikost največjega predmeta v seznamu manjša.

3.3.2 Prvo ujemanje

Glavna pomanjkljivost algoritma NF je v tem, da algoritem prehitro zapre koš, ki še ni docela poln: zapre ga takoj, ko nekega predmeta ne more uvrstiti vanj, čeprav je v košu morda še dovolj prostora za kak naslednji predmet. To pomanjkljivost poskuša odpraviti algoritem FF (iz angl. *First Fit*).

Algoritem FF poskuša nov predmet uvrstiti v prvega od odprtih košev, ki je dovolj velik za predmet. Če takega koša ni, algoritem odpre nov koš, pri tem pa ne zapre nobenega od prejšnjih. Vsi koši torej ostajajo odprti do izteka algoritma, razen seveda tistih, ki jih uspemo napolniti do vrha.

Primer delovanja prikazuje slika 3.3.



Slika 3.3: Primer s slike 2.2 za algoritem FF. V tem primeru algoritem FF doseže enako število košev kot NF, vendar to ni pravilo.

Ob ustrezni implementaciji lahko za algoritem FF dosežemo časovno zahtevnost $O(n \log n)$ [2]. Njegova prostorska zahtevnost je linearna $O(n)$. Vidimo, da sta obe zahtevnosti večji od zahtevnosti algoritma NF. Glede kvalitete rešitve pa velja naslednje (povzeto po [2]):

1. Za vsak primerek I velja: $\mathrm{FF(I)} \leq \lceil \frac{17}{10} \mathrm{OPT(I)} \rceil$.

2. $R_{FF}^{\infty} = \frac{17}{10}$ in $R_{FF} \leq \frac{7}{4}$.

3. Obnašanje v najslabšem primeru se močno izboljša, če so velikosti vseh predmetov manjše od $\frac{1}{2}$.

3.3.3 Omejitve ujemanj

Če za odločanje, v katerega od odprtih košev z dovolj prostora bomo uvrstili nov predmet, uporabimo drugačno pravilo, si lahko zamislimo še precej algoritmov, ki so podobni algoritmu FF. Omenimo dva primera:

1. *Algoritem BF* (iz angl. *Best Fit*) predmet razvrsti v tisti koš, v katerem po dodajanju tega predmeta ostane najmanj prostora, torej izbere najboljše ujemanje.

2. *Algoritem WF* (iz angl. *Worst Fit*) predmet razvrsti v tisti koš, v katerem po dodajanju ostane največ prostora.

Vsem tem algoritmom je skupna ena lastnost: Nov koš odprejo samo, če predmeta ne morejo uvrstiti v nobenega od že odprtih košev. Lahko rečemo, da gre za *poljubno ujemanje*, algoritme s to lastnostjo pa včasih označujemo kot *algoritme AF* (iz angl. *Any Fit*).

Zanima nas, kakšne so razlike med temi algoritmi s stališča kvalitete njihovih rešitev. Poglejmo si naslednji izrek:

Izrek 3.3.1. *Če je A algoritem AF, potem velja:* $R_{FF}^{\infty} \leq R_{A}^{\infty} \leq R_{NF}^{\infty}$.*[2]*

Ugotovitev ni spodbudna, saj pove, da katerikoli sprotni algoritem s poljubnim ujemanjem ne more doseči rezultata, ki bi bil boljši od algoritma FF. V nadaljevanju si bomo zato pogledali, kako je rezultat možno izboljšati.

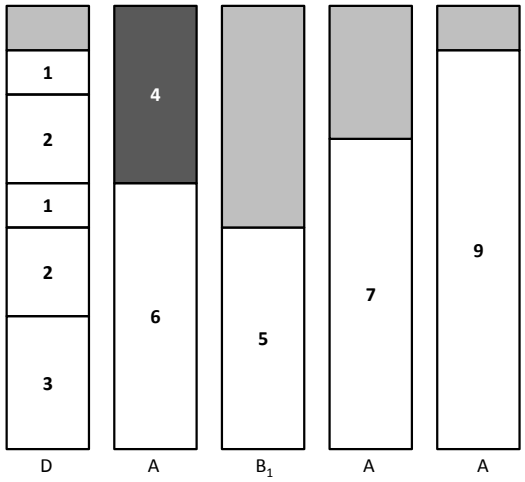
3.3.4 Izboljšave sprotnih algoritmov

Značilnost algoritmov, ki uporabljajo poljubno ujemanje, je, da ne odprejo novega koša, dokler lahko predmet uvrstijo v katerega od že odprtih, kar pa, kot pove izrek 3.3.1, ni ključ do približanja optimalni rešitvi.

Kronološko prvi algoritem, ki je presegel algoritem FF, je *algoritem RFF* (iz angl. *Refined First Fit*), predstavljen v članku [9]. RFF razdeli predmete glede na njihovo velikost v skupine, ki ustrezajo intervalom $D = (0, \frac{1}{3}]$, $B_2 = (\frac{1}{3}, \frac{2}{5}]$, $B_1 = (\frac{2}{5}, \frac{1}{2}]$ in $A = (\frac{1}{2}, 1]$, in podobno razdeli tudi koše. Nato z uporabo algoritma FF razporedi predmete iz vsake skupine v koše iste skupine. Izjema

je skupina B_2, kjer je vsak 6. predmet obravnavan drugače: razporejen je v enega od obstoječih košev iz skupine A ali pa se zanj odpre nov koš iz te skupine. Na ta način se doseže, da se občasno odpre nov koš, v katerega lahko kasneje razporedimo predmet, ki je večji od $\frac{1}{2}$. V omenjenem članku je tudi pokazano, da je časovna zahtevnost algoritma RFF $O(n \log n)$ in da velja $R^{\infty}_{RFF} = \frac{5}{3}$. Prostorska zahtevnost je seveda $O(n)$.

Delovanje algoritma RFF prikazuje slika 3.4. Uporabljeni so bili prilagojeni intervali $D = (0, 3]$, $B_2 = (3, 4]$, $B_1 = (4, 5]$ in $A = (5, 10]$. V želji po boljši ilustraciji delovanja algoritma smo edini predmet iz skupine B_2 obravnavali, kot bi bil 6. po vrsti.



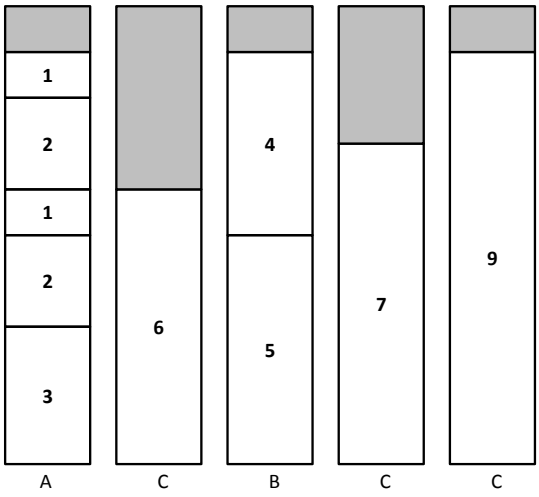
Slika 3.4: Primer s slike 2.2 za algoritem RFF. Vsak uporabljeni koš ima označeno skupino, ki ji pripada. Predmet iz skupine B_2 je obarvan temno sivo.

Zanimiv je tudi algoritem $Harmonic_K$, opisan v članku [4]. Algoritem je podoben zgoraj opisanemu RFF, njegova posebnost pa je v tem, da uporablja omejen prostor. Z izjemo algoritma NF vsi do sedaj opisani algoritmi uporabljajo neomejen prostor, saj košev ne zaprejo, dokler ti niso polni. V praksi si tega pogosto ne moremo privoščiti, zato je omejitev števila košev, ki jih algoritem uporablja v nekem trenutku, pogosto nujna. Harmonic$_K$ uporablja K intervalov I_k, tako da je $I_k = (\frac{1}{k+1}, \frac{1}{k}]$ za $1 \leq k < K$ in $I_K = (0, \frac{1}{K}]$. Predmeti so razdeljeni v skupine glede na to, v kateri interval sodijo po velikosti. Tudi koši so razdeljeni na tak način, pri čemer je v vsakem trenutku lahko odprt le en koš iz posamezne skupine (torej skupno največ K košev). Algoritem predmete

razporeja na sledeči način: Če je v košu še prostor, predmet razporedi v koš, ki ustreza skupini predmeta, sicer pa ta koš zapre, odpre novega in predmet uvrsti vanj. Prostorska zahtevnost algoritma je torej $O(K)$, časovna pa $O(n)$. Dokazano je, da za Harmonic$_K$ velja:

$$\lim_{K\to\infty} R_{H_K}^{\infty} = T_{\infty} = 1.69103\ldots [2]$$

Delovanje algoritma Harmonic$_K$ prikazuje slika 3.5. Uporabljeni so bili prilagojeni intervali $A = (0, 3]$, $B = (3, 5]$ in $C = (5, 10]$.



Slika 3.5: Primer s slike 2.2 za algoritem Harmonic$_K$ pri $K = 3$.

Omenjena algoritma sta služila za osnovo mnogim zapletenejšim algoritmom, ki so se trudili izboljšati asimptotični količnik najslabšega primera. Trenutno najboljšo spodnjo mejo predstavlja naslednji izrek:

Izrek 3.3.2. *Za vsak sprotni algoritem A velja $R_A^{\infty} \geq 1.540$. [2]*

3.4 Naknadni algoritmi

Pri sprotnih algoritmih smo spoznali, da največji problem predstavljajo *veliki* predmeti, t.j. predmeti, ki so večji od $\frac{1}{2}$, saj jih težko uvrstimo v koš, sploh če je že delno zapolnjen. Zato se pojavi ideja, da bi velike predmete obravnavali najprej, preostali prostor pa poskusili zapolniti z manjšimi. Na tem principu temeljijo algoritmi, ki jih bomo spoznali v nadaljevanju.

3.4.1 Padajoče prvo in padajoče najboljše ujemanje

Algoritma FFD (iz angl. *First Fit Decreasing*) in BFD (iz angl. *Best Fit Decreasing*) sta izpeljanki prej predstavljenih algoritmov FF in BF. Za oba je značilno, da predmete najprej uredimo po velikosti v padajočem vrstnem redu, nato pa nad dobljenim seznamom izvedemo algoritem FF oziroma BF.

Če se spet spomnimo primera s slike 2.2, bi oba algoritma dosegla identično, optimalno rešitev (ki je že prikazana na omenjeni sliki). Prostorska in časovna zahtevnost ostaneta nespremenjeni glede na sprotna algoritma.

Za algoritma FFD in BFD velja tudi (povzeto po [2]):

1. $R_{FFD}^{\infty} = R_{BFD}^{\infty} = \frac{11}{9}$.
2. $R_{FFD} = R_{BFD} = \frac{3}{2}$.
3. Obstajajo seznami, kjer BFD doseže boljše rezultate kot FFD, obstajajo pa tudi seznami, kjer je FFD boljši.

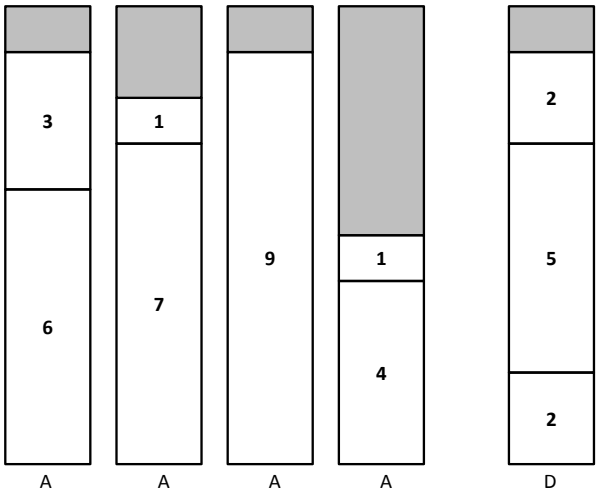
Hitro se postavi vprašanje, ali lahko z drugačnimi algoritmi dosežemo boljše rezultate. Odgovor je da, vendar ima izboljšanje svojo ceno, in sicer povečanje časovne zahtevnosti preko linearne in preko $O(n \log n)$, kar je pogosto nesprejemljivo.

3.4.2 Zhangov algoritem

V članku [10] predstavljeni algoritem je zanimiv, ker ima linearno časovno zahtevnost. Deluje na sledeči način: Na začetku velike predmete razporedimo vsakega v svoj koš. Koše, ki še niso polni, označimo kot *aktivne* in odprte ter jih oštevilčimo v poljubnem vrstnem redu. Preostale predmete razvrstimo po naslednjih pravilih:

1. Če obstaja odprt aktiven koš, poskušamo predmet uvrstiti v koš z najnižjim indeksom. Če to ni mogoče, ta koš zapremo in uporabimo *dodaten* koš:
   (a) Če je dodaten koš že odprt in je v njem dovolj prostora, predmet uvrstimo vanj.
   (b) Drugače dodaten koš zapremo, za predmet odpremo novega in ga označimo kot dodatnega.
2. Če odprt aktiven koš ne obstaja, odpremo nov koš, vanj uvrstimo predmet in ga označimo za aktivnega.

Delovanje Zhangovega algoritma prikazuje slika 3.6. Aktivni koši so oštevilčeni v naraščajočem vrstnem redu. Za algoritem velja $R_Z = \frac{3}{2}$.



Slika 3.6: Primer s slike 2.2 za Zhangov algoritem. A in D označujeta aktivne oziroma dodatne koše.

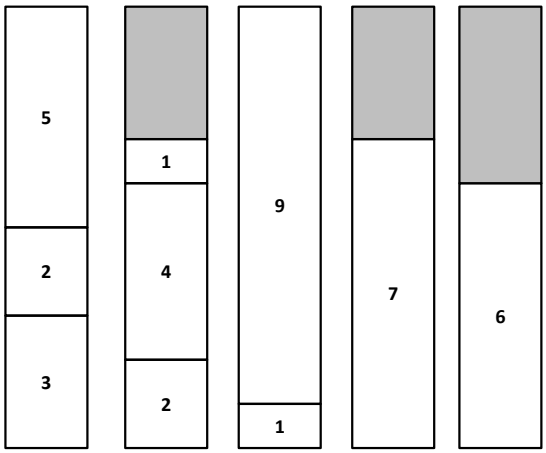
Algoritem lahko modificiramo, da uporablja konstanten prostor: Na začetku odpremo samo en koš, v katerega uvrstimo prvi velik predmet. Ko ta koš zapremo, odpremo naslednjega, v katerega najprej uvrstimo naslednji velik predmet. Tako sta v vsakem trenutku odprta največ dva koša: en aktivni in en dodatni. Opisani algoritem je pravzaprav polsprotni, saj moramo vnaprej poznati le velike predmete.

3.5 Algoritmi za polnjenje košev z omejitvijo kardinalnosti

V članku [11] so predstavljeni trije algoritmi, ki so prirejeni posebej za problem polnjenja košev z omejitvijo kardinalnosti. Vsem algoritmom je skupno, da seznam predmetov najprej sortirajo po padajočem vrstnem redu (tako kot na primer algoritem FFD). V vsakem koraku je izbran nabor N_{max} predmetov, s katerimi se napolni en koš. S C je označena preostala kapaciteta trenutnega koša. Pri izboru prvega predmeta torej velja $C = C_{max}$, z vsakim dodanim predmetom pa se C zmanjša za velikost le-tega. Opisani postopek se ponavlja, dokler niso razvrščeni vsi predmeti.

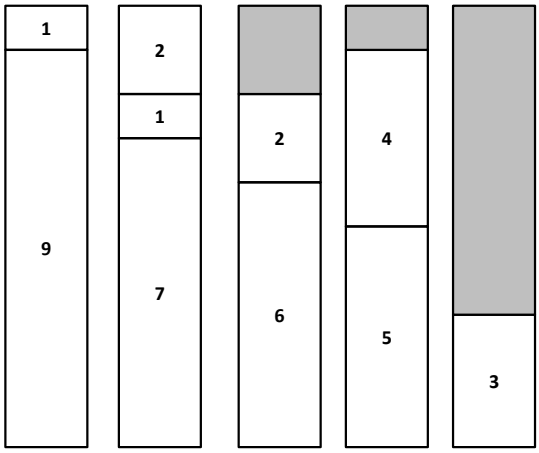
Algoritmi za izbor nabora predmetov uporabljajo sledeče kriterije:

- Algoritem 1 vsakič izbere največji predmet, za katerega velja $x_i \leq C/k$, pri čemer je $k = N_{max}, \ldots, 1$. Delovanje algoritma prikazuje slika 3.7.
- Algoritem 2 za prvi predmet izbere največji predmet, za katerega velja $x_i \leq C$, ostalih $N_{max} - 1$ predmetov pa izbere tako, da vsakič izbere največji predmet, za katerega velja $x_i \leq C/k$, $k = N_{max} - 1, \ldots, 1$. Delovanje algoritma prikazuje slika 3.8.
- Algoritem 3 prvih $\lceil N_{max}/2 \rceil$ predmetov izbere tako, da za vsakega od njih izbere naključni predmet, ki zadošča pogoju $x_i \leq C$. Preostale predmete pa izbere tako, da za vsakega od njih izbere največji predmet, ki zadošča pogoju $x_i \leq C/k$, $k = \lfloor N_{max}/2 \rfloor, \ldots, 1$.



Slika 3.7: Primer s slike 2.2 za Algoritem 1, pri čemer je $N_{max} = 3$. V prvem koraku algoritem izbere predmete, ki ustrezajo pogojem $x_i \leq \frac{10}{3}$, $x_i \leq \frac{7}{2}$ in $x_i \leq \frac{5}{1}$.

Vsi trije algoritmi sodijo med naknadne algoritme, saj za delovanje zahtevajo urejen seznam predmetov. Hitro lahko vidimo, da je prostorska zahtevnost vseh algoritmov konstantna, saj v vsakem koraku zapolnijo en koš, ki ga kasneje ne uporabljajo več. Časovna zahtevnost teh treh algoritmov pa je ob primerni implementaciji $O(n \log n)$.



Slika 3.8: Primer s slike 2.2 za Algoritem 2, pri čemer je $N_{max} = 3$. V prvem koraku algoritem izbere predmete, ki ustrezajo pogojem $x_i \leq 10$, $x_i \leq \frac{1}{2}$ (noben predmet ne ustreza pogoju) in $x_i \leq \frac{1}{1}$.

Poglavje 4

Praktični preizkus

Sedaj je čas, da spoznane algoritme za reševanje problema polnjenja košev preizkusimo tudi v praksi. Osnovni namen eksperimenta je raziskati, kolikšen vpliv ima omejitev kardinalnosti na kakovost rešitve algoritmov. So algoritmi, razviti posebej za problem polnjenja košev z omejitvijo kardinalnosti, zaradi svoje specifičnosti boljši od algoritmov za reševanje osnovnega problema? To je vprašanje, na katerega bomo skušali odgovoriti v nadaljevanju.

4.1 Prilagoditve algoritmov

Za potrebe eksperimenta so bili poleg vseh treh algoritmov iz članka [11] izbrani še algoritma FFD in RFF ter Zhangov algoritem. Ker slednji rešujejo osnovni problem polnjenja košev, jih je bilo potrebno nekoliko prilagoditi za dodatno omejitev kardinalnosti. Modifikacije so bile sledeče:

- Algoritem FFD predmet uvrsti v prvi koš, v katerem je dovolj prostora *in* v katerem je število predmetov manjše od N_{max}.
- Podobno algoritem RFF predmet uvrsti v prvi koš ustrezne skupine, v katerem je dovolj prostora *in* dovolj malo predmetov.
- Zhangov algoritem (aktivni ali dodatni) koš zapre tudi v primeru, ko je v njem še dovolj prostora za trenutni predmet, vendar je bilo doseženo maksimalno dovoljeno število predmetov v košu.

Vsi algoritmi so kot vhodni podatek prejeli po velikosti (padajoče) urejen seznam, čeprav algoritem RFF tega ne zahteva. Algoritmi so bili implementirani v programskem okolju Matlab (glej dodatek A).

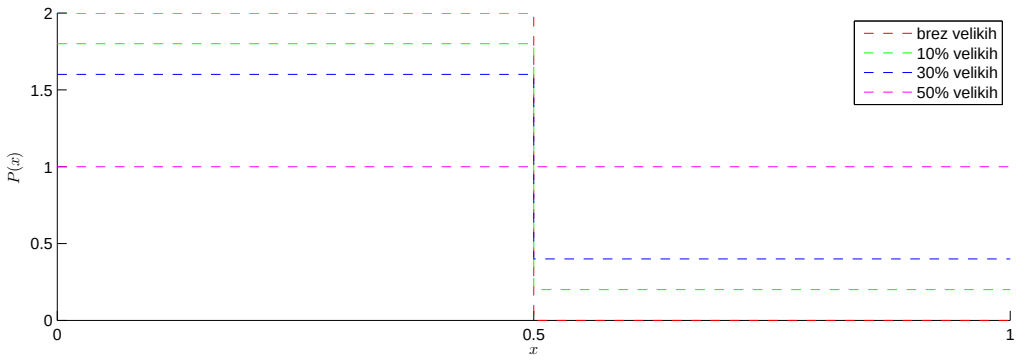
4.2 Eksperimentalna primerjava algoritmov

Prvi eksperiment je služil za medsebojno primerjavo algoritmov s stališča kakovosti rešitve. Zanimalo nas je, kako se algoritmi, razviti posebej za reševanje problema polnjenja košev z omejitvijo kardinalnosti, obnesejo v primerjavi z algoritmi za reševanje splošnega problema. Podoben eksperiment je bil opravljen že v članku [11], vendar z dvema pomembnima razlikama: Za primerjavo s splošnimi algoritmi je bil uporabljen samo algoritem FFD, seznami predmetov, na katerih je preizkus potekal, pa niso vsebovali velikih elementov.

Kot v omenjenem eksperimentu je bila tudi tokrat uporabljena vrednost $N_{max} = 4$. Za kapaciteto košev je bila izbrana $C_{max} = 1$ kot najbolj splošna. Ker smo v teoretičnem delu videli, da največji izziv algoritmom predstavljajo veliki predmeti, je naš preizkus potekal na štirih različnih tipih seznamov predmetov, vseh dolžine $n = 100$:

- seznami brez velikih predmetov,
- seznami z 10% velikih predmetov,
- seznami s 30% velikih predmetov in
- seznami s 50% velikih predmetov.

Vsi seznami so bili generirani z uporabo Matlabovega psevdonaključnega generatorja števil, ki uporablja enakomerno porazdelitev, pri čemer je bila upoštevana zgornja sestava seznamov. Porazdelitev prikazuje tudi slika 4.1. Eksperiment je bil ponovljen 10000-krat, vsakič nad novo generiranimi seznami.



Slika 4.1: Porazdelitev velikosti predmetov za posamezne tipe seznamov.

Za vsak seznam je bila izračunana teoretična spodnja meja za problem polnjenja košev z omejitvijo kardinalnosti:

$$\mathrm{OPT(I)} = \max\left(\lceil\frac{\sum_{i=1}^{n} x_i}{C_{max}}\rceil, \lceil\frac{n}{N_{max}}\rceil\right), \tag{4.1}$$

ki je služila kot približek za optimalno rešitev. Da bi določili kakovost posameznega algoritma, smo uporabili razmerje med rešitvijo algoritma in optimalno rešitvijo A(I) / OPT(I).

Rezultate testiranj prikazujejo slike 4.2 do 4.5 (zaradi večje preglednosti je prikazanih samo prvih 500 iteracij), povprečne vrednosti pa so zbrane v tabeli 4.1.

povprečje	**brez velikih**	**10% velikih**	**30% velikih**	**50% velikih**
FFD	1.1273	1.0933	1.0348	1.0418
RFF	1.3318	1.3199	1.3251	1.3323
Zhang	1.3606	1.3530	1.2680	1.2581
Algoritem 1	1.0262	1.0698	1.1421	1.2059
Algoritem 2	1.0493	1.0552	1.0504	1.0440
Algoritem 3	1.0256	1.0571	1.0992	1.1434

Tabela 4.1: Povprečne vrednosti kvocienta A(I)/OPT(I) za izbrane algoritme pri različno sestavljenih seznamih.

Kot je razvidno iz slik, se rezultati od seznama do seznama precej razlikujejo, tako če pogledamo isti algoritem, kot če primerjamo različne algoritme. Tako obnašanje je pričakovano, saj za vsak algoritem obstajajo seznami, ki so zanj bolj ali manj ugodni, prav tako pa obstajajo seznami, ki so ugodni za nekatere algoritme, za druge pa ne. Obnašanje algoritmov na enem samem seznamu nam zato pove bolj malo, mnogo bolj zgovorno pa je povprečje.

Rezultati za sezname brez velikih predmetov so konsistentni z rezultati predhodnih testiranj iz članka [11], kar daje našemu eksperimentu dodatno težo. Pri seznamih s 50% velikih predmetov se postavlja vprašanje, ali je uporabljeni približek 4.1 za OPT(I) še ustrezen. Zaradi definicije velikih predmetov je namreč jasno, da za vsak velik predmet v seznamu potrebujemo nov koš. Število velikih predmetov posledično neposredno vpliva na optimalno število potrebnih košev, kar bi bilo pri tako velikem deležu velikih predmetov dobro upoštevati za bolj pravilno razmerje A(I) / OPT(I).

Rezultati eksperimenta dobro pokažejo, kako se kakovost rešitve posameznega algoritma spreminja glede na delež velikih predmetov. Opazimo lahko, da

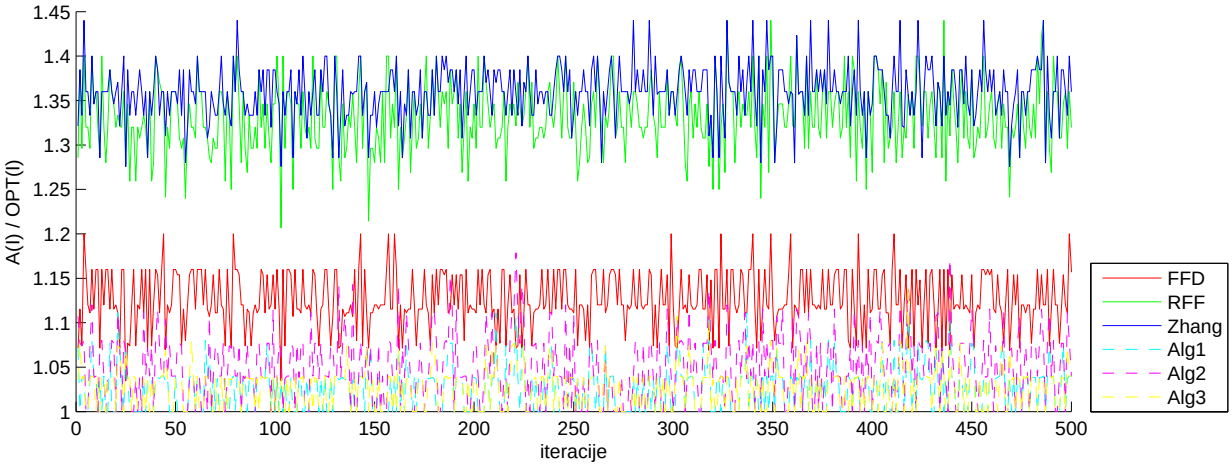
Zhangov algoritem največ pridobi z višjim deležem velikih predmetov, medtem ko so pri algoritmu RFF razlike nesignifikantne in vpliv deleža velikih predmetov najmanjši. Podobno kot za Zhangov algoritem velja tudi za algoritem FFD, le da je izboljšanje nekoliko manjše.

Pri vplivu deleža velikih predmetov na algoritme za reševanje problema polnjenja košev z omejitvijo kardinalnosti naš eksperiment pokaže na pomembno razliko, ki je bila pri predhodnih testiranjih spregledana. Tako Algoritem 1 kot Algoritem 3 namreč močno poslabšata kakovost svoje rešitve, ko se delež velikih predmetov veča. Razlog za poslabšanje je v tem, da oba algoritma velike predmete poskušata uvrstiti v koš šele na koncu, kar se izkaže za slab pristop, saj je koš, napolnjen na tak način, pogosto že preveč poln, da bi lahko sprejel velik predmet. Algoritem 3 je ob velikem deležu velikih predmetov nekoliko boljši zaradi svoje naključnosti, ki mu omogoča, da kot prvi predmet kdaj izbere tudi enega izmed velikih predmetov. Nasprotno pa je Algoritem 2 veliko manj občutljiv na velike predmete, saj za prvi predmet vedno izbere največjega možnega, preostanek kapacitete koša pa poskuša zapolniti z manjšimi predmeti.

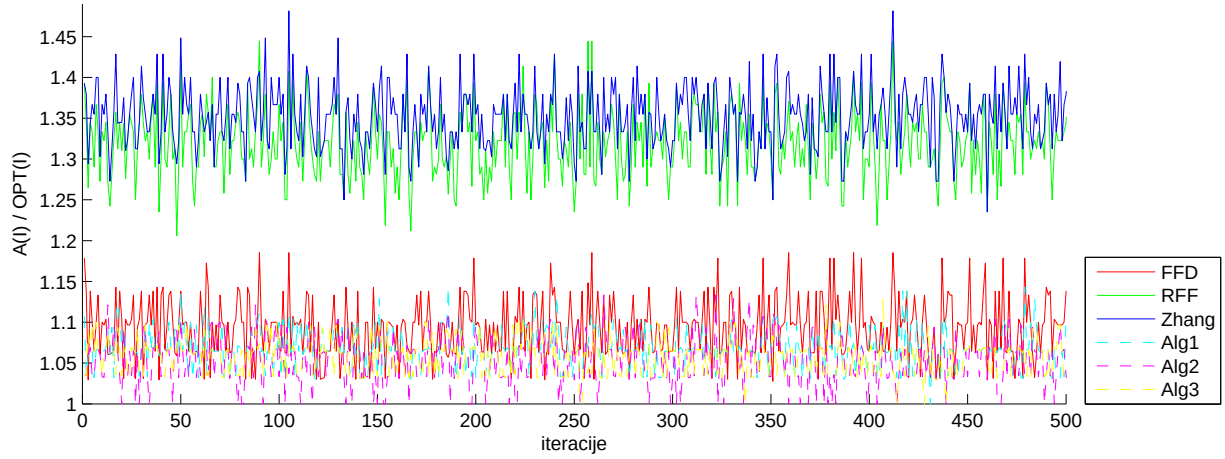
Če primerjamo algoritme med sabo, hitro vidimo, da se Zhangov algoritem in algoritem RFF odrežeta slabše od algoritma FFD. Za algoritem RFF je takšno obnašanje delno pričakovano, saj je bil zamišljen kot sprotni algoritem, vsi ostali pa so naknadni. Slabši dosežki Zhangovega algoritma so verjetno posledica linearne časovne zahtevnosti.

Vsi trije algoritmi za polnjenje košev z omejitvijo kardinalnosti dosežejo boljše rezultate od algoritma FFD za sezname z malo (do 10%) ali nič velikimi predmeti. Najbolj zanimiv pa je Algoritem 2, ki ostaja konkurenčen algoritmu FFD tudi ob povečevanju deleža velikih predmetov v seznamu.

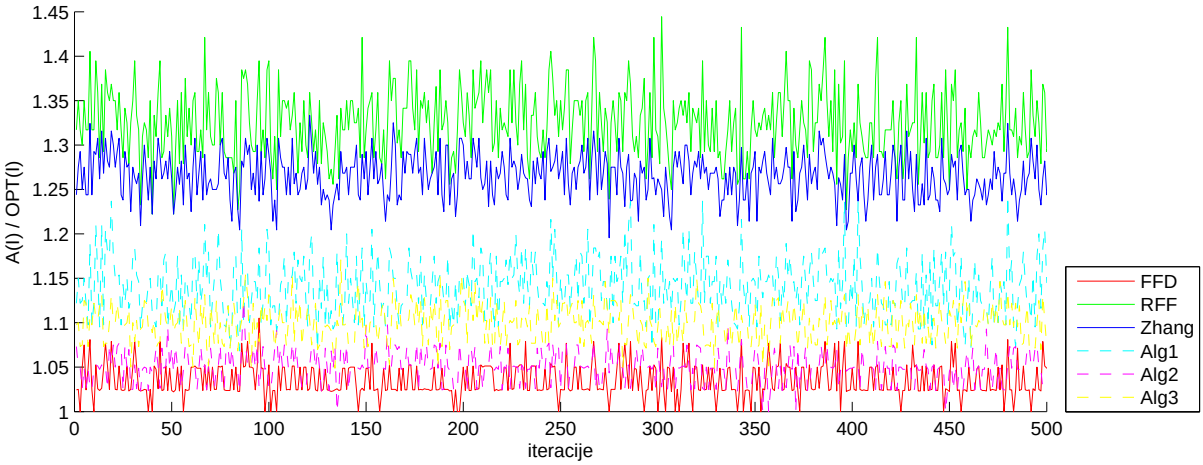
Kot zanimivost še omenimo, da je bil v začetni fazi testiranja izveden tudi eksperiment s seznami z enakimi deleži velikih predmetov, vendar sestavljenimi iz celih števil z intervala (0, 10]. Rezultati niso navedeni, ker so bili trendi spreminjanja kvalitete rešitve glede na delež velikih predmetov v seznamu in medsebojnih primerjav algoritmov podobni. Razlika pa je bila v povprečnem razmerju A(I) / OPT(I), ki je bilo pri celoštevilskih seznamih manjše, kar kaže na to, da se z omejitvijo na cela števila (sploh z relativno majhnega intervala) lažje približamo optimalni rešitvi zaradi mnogo manjšega prostora rešitev.



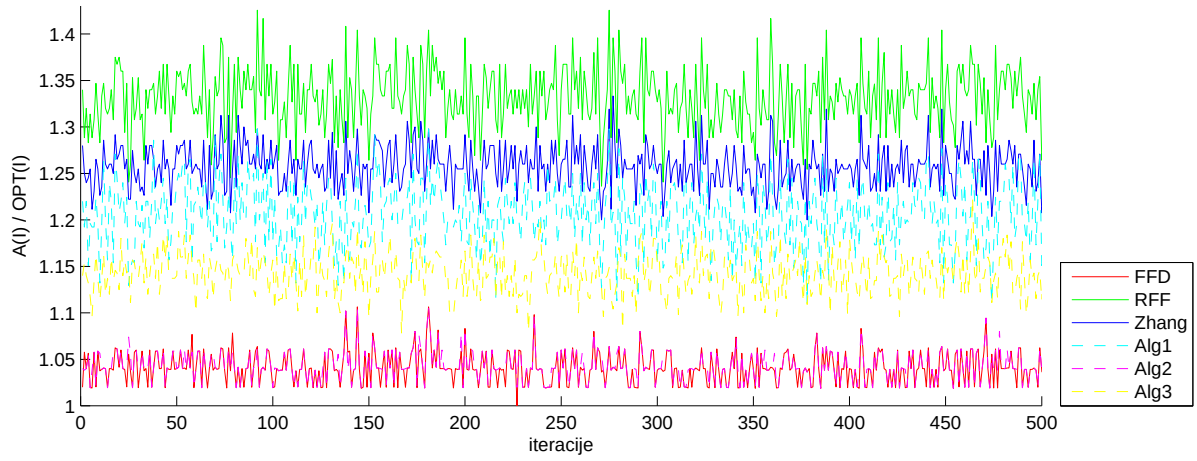
Slika 4.2: Rezultati eksperimentalne primerjave algoritmov za sezname brez velikih predmetov.



Slika 4.3: Rezultati eksperimentalne primerjave algoritmov za sezname z 10% velikih predmetov.



Slika 4.4: Rezultati eksperimentalne primerjave algoritmov za sezname s 30% velikih predmetov.

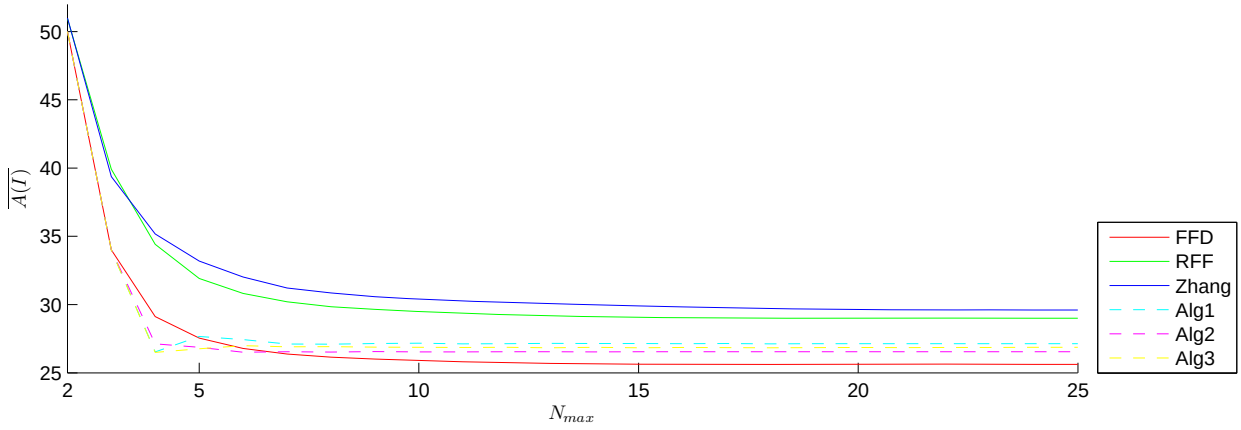


Slika 4.5: Rezultati eksperimentalne primerjave algoritmov za sezname s 50% velikih predmetov.

4.3 Povečevanje dovoljenega števila predmetov v košu

Z drugim eksperimentom smo želeli preveriti, kako se izbrani algoritmi obnašajo, ko povečujemo največje dovoljeno število predmetov v enem košu. Ob $N_{max} \geq n$ omejitev kardinalnosti namreč nima več vpliva, zato se problem polnjenja košev z omejitvijo kardinalnosti prevede kar na osnovni problem.

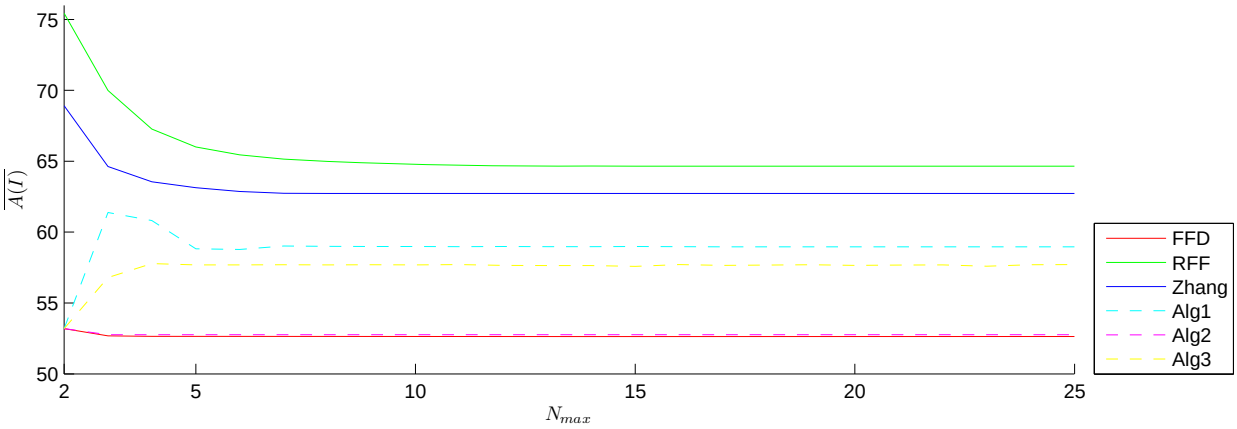
Za potrebe eksperimenta smo generirali sezname predmetov dolžine $n = 100$ pri $C_{max} = 1$. Uporabili smo dva tipa seznamov: sezname brez velikih predmetov in sezname s 50% velikih predmetov, ki so bili generirani na isti način kot pri prvem eksperimentu. Dovoljeno število predmetov v košu smo spreminjali kot $N_{max} = 2, 3, \ldots, \frac{n}{4}$. Meja $\frac{n}{4}$ se je v začetnih fazah eksperimenta izkazala za dovolj veliko za dosego konvergence, zato N_{max} nismo povečevali preko nje. Eksperiment je bil ponovljen 1000-krat, rezultati pa so predstavljeni na slikah 4.6 in 4.7. Prikazano je povprečno število košev v odvisnosti od maksimalnega dovoljenega števila predmetov v košu.



Slika 4.6: Rezultati povečevanja N_{max} za sezname brez velikih predmetov.

Kot pričakovano se kvaliteta rešitve (vsaj v večini primerov) izboljšuje ob povečevanju N_{max}, dokler ne dosežemo meje, ko N_{max} nima več vpliva in omejujoči faktor postane kapaciteta posameznega koša. Kako hitro dosežemo konvergenco, je odvisno od deleža velikih predmetov v seznamu: pri velikem deležu ima N_{max} mnogo manjši vpliv, zato je konvergenca dosežena hitreje, kar se pokaže pri seznamih s 50% velikih predmetov.

Zanimivo je, da pri seznamih brez velikih predmetov vsi trije algoritmi, ki privzeto upoštevajo omejitev kardinalnosti, dosežejo najboljše rezultate ravno



Slika 4.7: Rezultati povečevanja N_{max} za sezname s 50% velikih predmetov.

pri $N_{max} = 4$, kar je nastavitev, ki je bila uporabljena pri predhodnih testiranjih, z večanjem N_{max} pa se njihovo delovanje nekoliko poslabša. Zakaj prihaja do takega obnašanja, bi bilo morda pametno podrobneje raziskati.

Podobno se zgodi pri seznamih s 50% velikih predmetov, kjer pri Algoritmu 1 in Algoritmu 3 pri majhnih N_{max} prihaja do nepričakovanih skokov v kvaliteti rešitve. Tako obnašanje lahko razložimo s samim delovanjem obeh algoritmov, saj pri majhnem N_{max} koš začnemo polniti z večjimi predmeti kot pri velikem.

Opazimo lahko, da povečevanje N_{max} nima velikega vpliva na kvaliteto rešitve za izbrane algoritme. Algoritmi za reševanje splošnega problema tudi pri neomejujočem N_{max} ne dajejo boljših rešitev kot algoritmi, prirejeni za omejitev kardinalnosti. Izjema je algoritem FFD, ki pri večjem N_{max} doseže nekoliko boljše rezultate pri seznamih brez velikih predmetov, kar pa je lahko zgolj naključje, saj je bil eksperiment opravljen na relativno majhnem številu seznamov.

Poglavje 5

Zaključek

Kot smo videli v teoretičnem delu, je polnjenje košev problem z veliko možnostmi praktične uporabe. Zaradi njegove zahtevnosti je smiselno razvijati aproksimacijske algoritme za reševanje tega problema, ki nam ob sprejemljivi časovni zahtevnosti zagotavljajo dovolj kvalitetno približno rešitev.

V pričujočem diplomskem delu smo se ukvarjali s primerjavo aproksimacijskih algoritmov za reševanje bolj specifičnega problema polnjenja košev z omejitvijo kardinalnosti z aproksimacijskimi algoritmi za reševanje splošnega problema. Kot so pokazali eksperimenti, so vsi trije specifični algoritmi v povprečju boljši od uporabljenih splošnih algoritmov za sezname brez velikih predmetov ali z majhnim deležem le-teh, in to ob primerljivi časovni zahtevnosti. V takih primerih je torej priporočljiva uporaba specifičnih algoritmov. V primeru seznamov z večjim deležem velikih predmetov pa se od preizkušenih algoritmov v povprečju najbolje obnese splošni algoritem FFD, vendar specifični Algoritem 2 ne zaostaja veliko, kar ga dela zanimivega tudi za take sezname.

Zanimivo je, da lahko s predstavljenimi specifičnimi algoritmi brez poslabšanja kvalitete rešitve rešujemo tudi splošni problem polnjenja košev, seveda ob primerno nastavljenem največjem dovoljenem številu predmetov v košu. Kot najboljši izmed trojice se spet izkaže Algoritem 2. Povzamemo torej lahko, da je Algoritem 2 v splošnem zelo dobra alternativa algoritmu FFD, v določenih primerih pa ga celo prekaša in je zato bolj priporočljiv za sezname z majhnim deležem velikih predmetov.

Dodatek A

Uporabljeni algoritmi

A.1 Algoritem FFD

```
function [ m ] = FFD( X, Cmax, Nmax )
   Ckosi = 0; % seznam zasedenosti košev
   Nkosi = 0; % seznam števila predmetov v koših

   for i=1:numel(X)
      [~, idxs] = find(Ckosi + X(i) <= Cmax);
      idx = find(Nkosi(idxs) < Nmax, 1, 'first');
      if numel(idx) == 0
         Ckosi = [Ckosi 0];
         Nkosi = [Nkosi 0];
         idx = numel(Ckosi);
      else
         idx = idxs(idx);
      end
      Ckosi(idx) = Ckosi(idx) + X(i);
      Nkosi(idx) = Nkosi(idx) + 1;
   end
   m = numel(Ckosi);
end
```

A.2 Algoritem RFF

```
function [ m ] = RFF( X, Cmax, Nmax )
```

```
   B2 = 6; % konstanta za skupino B2
   B2_count = 0; % števec za skupino B2
   m = 0;
% meje intervalov
   lim_A = 0.5 * Cmax;
   lim_B1 = 0.4 * Cmax;
   lim_B2 = 1/3 * Cmax;
% seznami zasedenosti košev
   C_A = 0;
   C_B1 = 0;
   C_B2 = 0;
   C_D = 0;
% seznami števila predmetov v koših
   N_A = 0;
   N_B1 = 0;
   N_B2 = 0;
   N_D = 0;
   for i=1:numel(X) % vsak predmet razporedimo v koš
      if X(i) > lim_B2 & & X(i) <= lim_B1
         B2_count = B2_count + 1;
      end
      % za skupino A
      if X(i) > lim_A || (B2_count == B2 & & X(i) > lim_B2 & & X(i) <= lim_B1)
         if B2_count == B2
            B2_count = 0; % ponastavitev števca
         end
         [~, idxs] = find(C_A + X(i) <= Cmax);
         idx = find(N_A(idxs) < Nmax, 1, 'first');
         if numel(idx) == 0 %
            C_A = [C_A 0];
            N_A = [N_A 0];
            idx = numel(C_A);
         else
            idx = idxs(idx);
         end
         C_A(idx) = C_A(idx) + X(i);
         N_A(idx) = N_A(idx) + 1;
      % za skupino B1
      elseif X(i) > lim_B1
```

```
        [~, idxs] = find(C_B1 + X(i) <= Cmax);
        idx = find(N_B1(idxs) < Nmax, 1, 'first');
        if numel(idx) == 0 %
           C_B1 = [C_B1 0];
           N_B1 = [N_B1 0];
           idx = numel(C_B1);
        else
           idx = idxs(idx);
        end
        C_B1(idx) = C_B1(idx) + X(i);
        N_B1(idx) = N_B1(idx) + 1;
     % za skupino B2
     elseif X(i) > lim_B2
        [~, idxs] = find(C_B2 + X(i) <= Cmax);
        idx = find(N_B2(idxs) < Nmax, 1, 'first');
        if numel(idx) == 0
           C_B2 = [C_B2 0];
           N_B2 = [N_B2 0];
           idx = numel(C_B2);
        else
           idx = idxs(idx);
        end
        C_B2(idx) = C_B2(idx) + X(i);
        N_B2(idx) = N_B2(idx) + 1;
     % za skupino D
     else
        [~, idxs] = find(C_D + X(i) <= Cmax);
        idx = find(N_D(idxs) < Nmax, 1, 'first');
        if numel(idx) == 0
           C_D = [C_D 0];
           N_D = [N_D 0];
           idx = numel(C_D);
        else
           idx = idxs(idx);
        end
        C_D(idx) = C_D(idx) + X(i);
        N_D(idx) = N_D(idx) + 1;
     end
  end
```

```
   % končno število uporabljenih košev
   if C_A(1) > 0
      m = m + numel(C_A);
   end
   if C_B1(1) > 0
      m = m + numel(C_B1);
   end
   if C_B2(1) > 0
      m = m + numel(C_B2);
   end
   if C_D(1) > 0
      m = m + numel(C_D);
   end
end
```

A.3 Zhangov algoritem

```
function [ m ] = Zhang( X, Cmax, Nmax )
   % velike predmete razporedimo v aktivne koše
   Ckosi = X(X > 0.5 * Cmax); % seznam zasedenosti košev
   Nkosi = ones(1, numel(Ckosi)); %seznam števila predmetov v koših
   Akosi = ones(1, numel(Ckosi)); % seznam odprtih košev
   Cdod = 0;
   Ndod = 0;
   num_dod = 1;
   for i=numel(Ckosi)+1:numel(X)
      idx = find(Akosi == 1, 1, 'first');
      if numel(idx) == 0
         Ckosi = [Ckosi X(i)];
         Nkosi = [Nkosi 1];
         Akosi = [Akosi 1];
      elseif Ckosi(idx) + X(i) > Cmax || Nkosi(idx) == Nmax
         Akosi(idx) = 0;
         if Cdod + X(i) <= Cmax & & Ndod < Nmax
            Cdod = Cdod + X(i);
            Ndod = Ndod + 1;
         else
            num_dod = num_dod + 1;
```

```
            Cdod = X(i);
            Ndod = 1;
         end
      else
         Ckosi(idx) = Ckosi(idx) + X(i);
         Nkosi(idx) = Nkosi(idx) + 1;
      end
   end
   m = numel(Ckosi) + num_dod;
   if Cdod == 0 % dodatni koš ni bil nikoli uporabljen
      m = m - 1;
   end
```

A.4 Algoritem 1

```
function [ m ] = Alg1( X, Cmax, Nmax )
   m = 0;
   while numel(X) > 0
      C = Cmax;
      m = m + 1;
      for i = Nmax:-1:1
         idx = find(X <= C/i, 1, 'first');
         if numel(idx) > 0
            C = C - X(idx);
            X(idx) = [];
         end
      end
   end
end
```

A.5 Algoritem 2

```
function [ m ] = Alg2( X, Cmax, Nmax )
   m = 0;
   while numel(X) > 0
      m = m + 1;
      idx = find(X <= Cmax, 1, 'first');
      C = Cmax - X(idx);
```

```
      X(idx) = [];
      for i = Nmax-1:-1:1
         idx = find(X <= C/i, 1, 'first');
         if numel(idx) > 0
            C = C - X(idx);
            X(idx) = [];
         end
      end
   end
end
```

A.6 Algoritem 3

```
function [ m ] = Alg3( X, Cmax, Nmax )
   m = 0;
   k = floor(Nmax/2);
   while numel(X) > 0
      m = m + 1;
      C = Cmax;
      for i = Nmax:-1:k+1
         idxs = find(X <= C);
         if numel(idxs) > 0
            idx = randi(numel(idxs), 1, 1);
            C = C - X(idxs(idx));
            X(idxs(idx)) = [];
         end
      end
      for i = k:-1:1
         idx = find(X <= C/i, 1, 'first');
         if numel(idx) > 0
            C = C - X(idx);
            X(idx) = [];
         end
      end
   end
end
```

Slike

Tabele

Literatura

[1] J. M. V. de Carvalho, "LP models for bin packing and cutting stock problems," *European Journal of Operational Research*, št. 141, str. 253–273, 2002.

[2] E. G. Coffman, M. R. Garey, D. S. Johnson, "Approximation algorithms for bin-packing: A survey," v zborniku *Approximation Algorithms for NP-Hard Problems*, PWS, Boston, 1997, str. 46-93.

[3] B. Korte, J. Vygen, *Combinatorial Optimization*: Theory and Algorithms, 4. izdaja, Berlin: Springer, 2008, pogl. 18.

[4] C. C. Lee, D. T. Lee, "A simple on-line packing algorithm," *Journal of ACM*, št. 3, zv. 32, str. 562-572, 1985.

[5] B. Robič, *Aproksimacijski algoritmi*, 2. izdaja, Ljubljana: Fakulteta za računalništvo in informatiko, 2008, pogl. 1, 2, 3 in 6.

[6] B. Vilfan, *Osnovni algoritmi*, 12. izdaja, Ljubljana: Založba FE in FRI, 2002, pogl. 6.

[7] V. V. Vazirani, *Approximation Algorithms*, 1. izdaja, Berlin: Springer, 2001, pogl. 9.

[8] E. C. Xavier, F. K. Miyazawa, "The Class Constrained Bin Packing Problem with applications to Video-on-Demand," *Theoretical Computer Science*, št. 1-3, zv. 393, str. 240-259, 2008.

[9] A. C. Yao, "New algorithms for bin packing," *Journal of ACM*, št. 2, zv. 27, str. 207-227, 1980.

[10] G. Zhang, X. Cai, C. K. Wong, "Linear-time-approximation algorithms for bin packing," *Operation Research Letters*, št. 5, zv. 26, str. 217-222, 2000.

[11] G. Žerovnik, J. Žerovnik, "Constructive heuristics for the canister filling problem," *Central European Journal of Operations Research*, DOI 10.1007/s10100-010-0164-5 , 2010.