Laboratorij za CAD - LECAD
Fakulteta za strojništvo
Univerza v Ljubljani

OPTIMIZACIJA RAZREZA LINIJSKIH NOSILCEV

interno poročilo 1/1-2000

Avtorji:

Leon Kos

Ljubljana, januar 2000

Povzetek

V poročilu so prikazani postopki za določitev razreza jeklenih profilov z upoštevanjem specifičnih zahtev naročnika kot so: povezava programa na bazo podatkov, dodatek za razrez, upoštevanje ostankov profilov, ki so na skladišču in optimizacija naročila dolžine in števila profilov. Prikazani so predvsem potrebni prijemi z razlago zahtevnosti in lastnostmi.

Kazalo

Poglavje 1

Uvod

Jeklene profile režemo na željene dolžine iz standardnih dolžin, ker je to cenejše in omogoča večjo fleksibilnost proizvodnje. Razrez profilov se določa že pri samem projektiranju, saj profili običajno niso na zalogi ali pa je skladiščenje predrago/prezahtevno. Ko so znane dimenzije profilov se pripravi pregled, kaj je potrebno naročiti za proizvodnjo in za to je potrebno pripraviti podroben načrt proizvodnje. Upoštevati je potrebno tudi to, da bo na skladišču mogoče dobiti tudi že razrezane profile, ki so prišli v skladišče kot sprememba v projektu, napaka pri razrezu, regularni odrezki standardnih profilov in viški naročil standardnih dolžin. Predvideno je, da se ves material, ki je na skladišču uporabi pri oprimizaciji razreza in s tem na bi se zmanjšala zaloga skladišča in hkrati bolje uporabil material.

Do sedaj se je združevanje manjših elementov v večje opravljal ročno s tem, da so bile večje dolžine naročene in ne razrezane iz standardnih. Tak postopek je drag, časovno zamuden in neoptimalen. Z razvojem programa želimo rešiti vse naštete težave in pridobiti na konkurenčni sposobnosti, ki jo prinese predvidena avtomatizacija proizvodnje.

Optimizacija razreza zahteva v praksi več različnih postopkov optimizacije in reševanja specifičnih zahtev, ki so prikazane v nadaljevanju.

Osnovni problem razreza je sestavljen iz več podproblemov, ki imajo v teoretičnem obravnavanji svoje ime. Reševanje problema je odvisno tudi še od vrste, velikosti vhodnih podatkov in tudi od vrste izhodnih podatkov. Optimizacijski problem definira naslednje:

> Izhod algoritma je opis podatkov, ki so v neki meri optimalni glede na vhodne podatke.

Celoštevilčna optimizacija je v primerjavi z zvezno veliko zahtevnejše. Očitno je, da algoritem potrebuje dva vira za rešitev problema: *čas* in *prostor*. Čas se običajno meri v številu sprememb stanj v algoritmu od začetka do končanja. Prostor je običajno definiran kot maksimalni prosto potreben za začasne podatke, ki jih algoritem potrebuje preko celotnega računanja.

1.1 Kompleksnost algoritmov in problemov

Količina virov, ki jih različni algoritmi potrebujejo za reševanje provlema, spada v domeno *teorije kompleksnosti*. Ta teorija govori o *časovni kompleksnosti* in o *prostorski zahtevnosti*. Poudariti je potrebno, da kompleksnost algoritma in kompleksnost problema podajata dva povsem različna koncepta.

Kompleksnost algoritma je najdaljši čas potreben za algoritem, da reši problem podane velikosti. To pomeni, da je kompleksnost algoritme merjena za t.i. *wors case*. Torej za najslabši primer, ki je lahko podan. Za primer algoritma s kompleksnostjo največ $O(x^2$ bo potreboval štiri krat več časa, če se bo količina podatkov podvojila.

Kompleksnost problema je definirana kot kompleksnost najboljšega algoritma za dani problem. To pa nujno ne pomeni najboljšega *znanega* algoritma. Res pa je, da je problem toliko zahteven kolikor je zahteven najboljši znani algoritem za reševanje problema. Lahko se zgodi tudi, da obstaja dokaz o obstoju algoritma, ki rešuje problem bolje od obstoječih, čeprav sam algoritem v praksi še ne obstaja.

Vsi problem itorej niso enako zahtevni. Nekateri problemi so težki in imajo visoko kompleksnost. Če čas potreben za rešitev problema raste eksponentno z velikostjo primera, takrat je problem težek, saj bo verjetno celo najboljši algoritem neuporaben na realih primerih. V praksi obstaja precej problemov, ki pripadajo težavnostni stopnji $\mathcal{P}$ in imajo polinomski čas kompleksnosti. Večina podproblemov, ki so povevezani z razrezom ima težavnost $\mathcal{NP}$.

1.2 Iskanje optimuma

Problemi, ki jih rešujemo pri razrezu so zelo zahtevni. Pravzaprav so nerešljivi v splošnem (za poljubne primere poljubne velikosti), razen če ni $\mathcal{P} = \mathcal{NP}$. Ko je evidentno, da je problem eksponentne narave je uporabno, če se dokažne njegova $\mathcal{NP}$ zahtevnost. In če je dokaz za $\mathcal{NP}$ zahtevnost težko najti, potem bodo smernice dokaza kazale tudi pot do optimalnega algoritma. Iskanje *polinomskega* algoritma zateva zrvhano mera optimizma.

Za $\mathcal{NP}$ probleme je možno določiti mejo uporabnosti, ki za dane velikosti primerov v uporabnem času dajo rezultate.

Če ni očitne zgornje meje za velikost primerov, ki se rešujejo z $\mathcal{NP}$ algoritmom, ali kadar je najboljša rešitev algoritma nedosegljiva v ustreznem času zaradi eksplozije možnosti, moramo opustiti iskanje optimuma. Žalostno kot to morda izgleda, to niti ni tako resno kot izgleda. V praksi večina problemov ne zahteva optimuma. Zašelena je le dobra rešitev problema. S takim pogledom na problem je iskanje dobrega aproksimacijskega algoritma povcem legitimno. Tako obstajajo aproksimacijski algoritmi, ki ne zagotavljajo globalnega optimuma za vsak podan primer, vendar še vedno dajejo rešitve blizu optimuma.

1.3 Hevristika

Ker je večina problemov znanih že vrsto let in ker ima vsak optimizacijski problem tudi praktično vrednost, je bilo razvitih tudi več hevristik, ki pristopajo k reševanju problema na recept. Ti postopki, ki jih hevristika podaja pa temeljijo na *zdravi pameti*.

Čeprav je težko definirati hevristiko in bi bila taka definicija precej subjektivna, lahko rešemo, da je to način reševanja problema, ki se izogiba očitnim napakam. Izkaže se, da so hevristike lahko zelo različne kar se tiče učinkovitosti. Nekatere so zelo neučinkovite ne glede na njihov intuitivni pristop k reševanju problema.

Ekstremni primer hevristike so *ekspertni sistemi*, ki rešujejo problem na *bazi znanja*, ki izkazuje znanje strokovnjaka pri reševanju problema. Tak pristop pa teži k temu, da je reševanje vsaj tako dobro, kot to delajo strokovnjaki na tem področju in ne da bi dali rezultate, ki so boljši od rezultata strokovnjaka. Reševanje na ta način običajno niti ne vpeljuje cenilno funkcijo za primerjanje rešitev.

1.4 Opis vsebine poročila

Problem optimalnega sestavljanja liniskij elementov z upoštevanjem vseh specifičnih zahtev razreza in skladišča je sestavljen iz več teroretičnih pristopov. Osnovni problem BPP (*Bin Packing Problem*) v poglavju 2 je tako dopolnjenen še s problemom MSSP (*Multiple Subset Sum Problem* v poglavju 4, ki omogoča uporabo skladišča. Obe metodi morata upoštevati še dodatek za razrez, kar je enostavno rešljivo za obe metodi v poglavju 3. Celoten problem optimiranja za razrez je bil izveden na realnih testnih podatkih. Analiza numeričnih rezultatov je podana v poglavju 5. Podrobneje pa so rezultati podani tudi v dodatku. V zaključku so komentirani predvsem rezultati, ki jih bo potrebno podrobneje razdelati pri samem uvajanju programa.

Poglavje 2

Osnovna rešitev: metoda polnjenja posod

Problem polnjenja posode je definiran takole: Za podano končno množico števil O (velikosti elementov) in konstantno velikost posode C poiščimo minimalno število posod tako, da vsota vseh elementov (podmnožica O) v posamezni posodi ne presega konstante C.

Ta problem optimizacije ima težavnostno stopnjo $\mathcal{NP}$ (N!), kjer je N ševilo elementov, ki jih je potrebno razvrstiti v posode.

Razrez profilov, barvanje zemljevidov, pakiranje datotek na diskete, razporeditev procesov na več procesorjev je le del velike družine, ki spada v *Bin Packing Problem* ali krajše BPP.

Ker je zahtevnostna stopnja problema BPP za permutacijsko reševanje prevelika je upotrebno uvesti hevristični način razporeditve elementov v posode. Obstaja več pristopov pri reševanju, ki spadaja domeno operacijsih raziskav, kot tudi umetne inteligence in umetnega življenja.

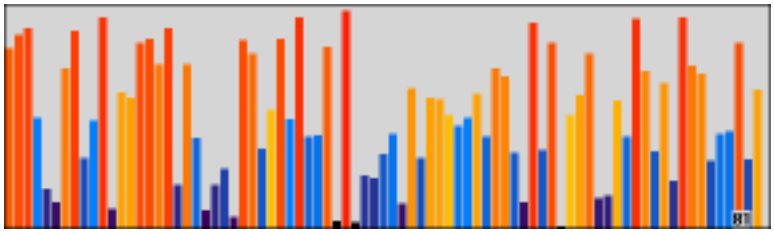
V praksi se uporabljajo deterministični pristopi s hevristično optimizacijo in pa genetski algoritmi iskanja.

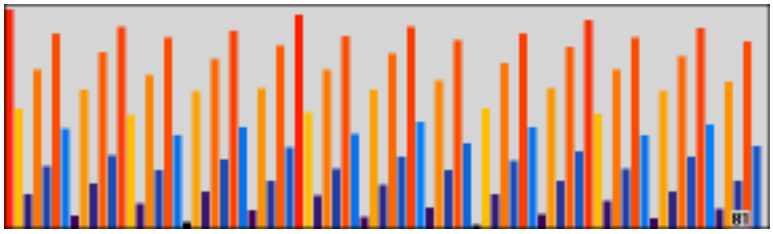
Spodnja meja potrebnega števila posod se lahko določi z enačbo

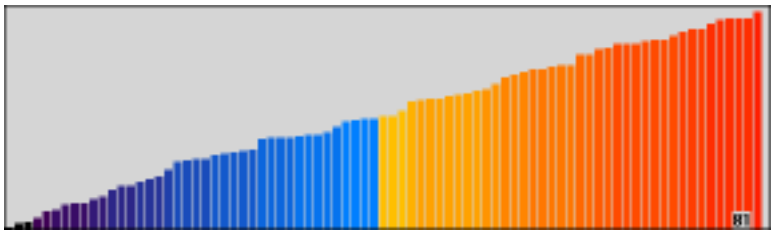
$$LB = \frac{\sum x_i}{C} \qquad i \in \{1, \ldots, N\} \tag{2.1}$$

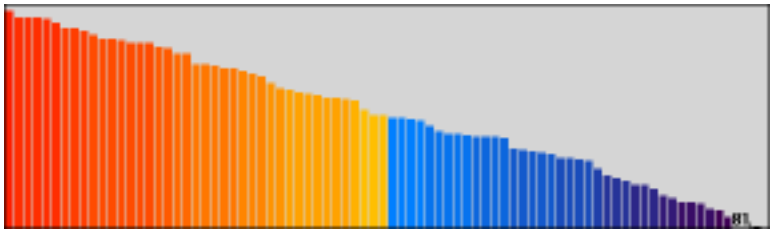
2.1 Hevristični algoritmi

Hevristični aproksimativni algoritmi brez naknadnega razporejanja pakirajo elemente v posodo tako, da jemljejo elemente enega za drugim in ga spravljajo v ustrezno prazno posodo. V kakšnem vrstnem redu se elementi jemljejo iz zalogovnika vpliva tudi na kvaliteto pakiranja. V praksi poznamo naslednje razporeditve elementov:

naključna 

regulrana 

naraščajoča 

padajoča 

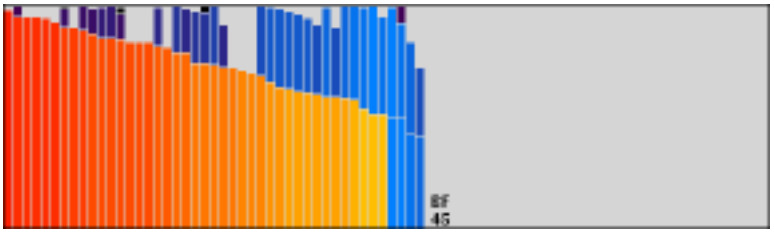
Pakiranje v posode se začne z zajemanjem elementov in postavljanje v posodo z različnimi postopki. Potreben ča za postavljanje elementov v posode je večinoma linearen in zato so to najhitrejše metode pakiranja.

V praksi je običajno najbolje, da elemente razporedimo po podajočem vrstnem redu. Tako najprej pakiramo velike elementa in nato vedno manjše. Primer takega pakiranje po metodi *Best Fit* je prikazan na sliki 2.1

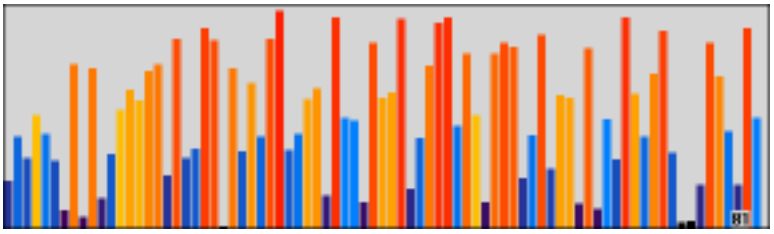
Element, ki se vzame iz zalogovnika se na določen način vstavi v dovolj prazno posodo. Če v obstoječih posodah ni prostora, se kreira nova posoda velikosti konstante C. Primeri pakiranja z različnimi postopki so podani za elemente razporejene tako kot kaže slika 2.1.

2.1.1 First Fit

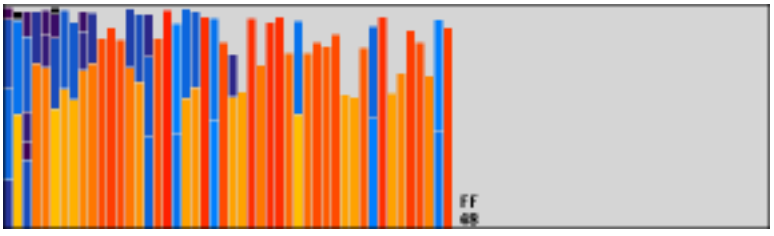
First fit pakiranje postavi element v prvo najbolj levo posodo, ki še ima dovolj prostora. š



Slika 2.1: Best Fit Descending razporeditev elementov v posode



Slika 2.2: Začetna razporeditev elementov v zalogovniku



Slika 2.3: First Fit pakiranje

First Fit Descending (FFD) je primer dobrega algoritma za probleme težavnosti $\mathcal{NP}$. Za vsak primer I hevristika FFD poda rešitev, ki ima

$$FFD(I) = \frac{11}{9}OPT(I) + 4posode \tag{2.2}$$

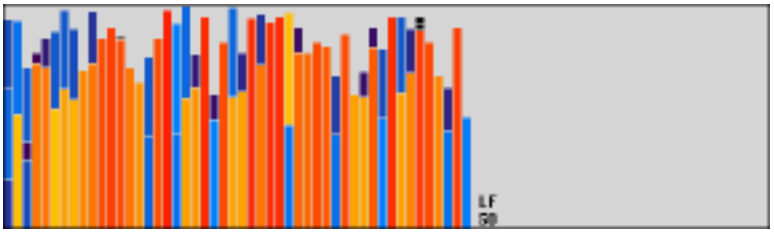
Za veliko večino poljubno velikih BPP primerov pa bo hevristika FFD podala rešitev v

$$FFD(I) = \frac{11}{9}OPT(I)posodah, \tag{2.3}$$

kar pomeni, da hevristika FFD ne bo uporabila več kot 22% posod v primerjavi z optimalno rešitvijo.

2.1.2 Last Fit

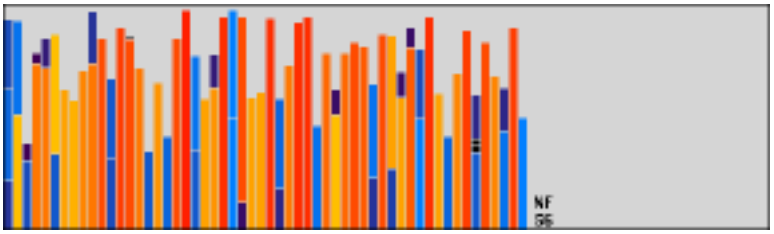
First fit pakiranje postavi element v prvo najbolj desno posodo, ki še ima dovolj prostora.



Slika 2.4: Last Fit pakiranje

2.1.3 Next Fit

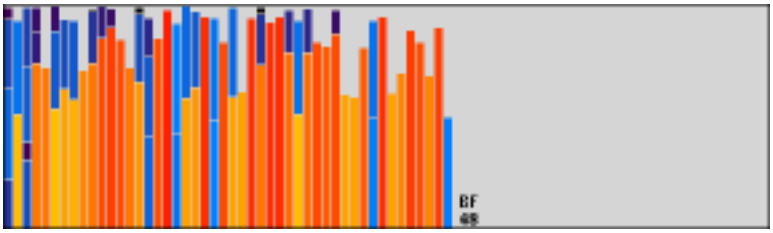
Next fit pakiranje postavi element v najbolj desno posodo. Če prostora ni dovolj začne novo posodo.



Slika 2.5: Next Fit pakiranje

2.1.4 Best Fit

Best fit pakiranje postavi element v posodo, ki najbolj zapolni eno od obstoječih posod.



Slika 2.6: Best Fit pakiranje

2.1.5 Exact Fit

Deterministični algoritem z enostavno hevristiko[3], ki se jo pokazala dokaj uspešna v primerjavi z genetskimi algoritmi. Postopek vključuje tudi lokalno optimizacijo in zato bolje pakira elemente od prej predstavljenih postopkov. Postopek pakiranja bi lahko v kratkem podali z naslednjim algoritmom:

1. Vzemi novo posodo in jo polni z elementi tako dolgo, dokler ne presežeš 1/3 posode
2. Preostanek posode poskusi napolniti tako, da v celoti zapolniš posodo. To lahko narediš z enim elementom, kombinacijo dveh ali treh elementov.
3. Če ne obstaja nobena kombinacija, ki bi popolnoma zapolnila preostanek, ga zmanjšaj za ena in ponavljaj postopek 2, dokler ne bo uspeha.
4. Postopek ponavljaj dokler imaš elemente na razpolago

Postopek je relativno hiter. Preiskuje pa večjo količino prostora in zato ima tudi večje možnosti pri določitvi pakirnega razporeda.

2.2 Skupinski genetski algoritmi

Pri tako velikem iskalnem prostoru, kot je BPP se kot uspešni pogažejo tudi genetski algoritmi, ki z enostavnimi operacijami kot sta križanje in mutacija genov v kromosomu, generirajo nove rešitve. Genetski algoritmi imajo širok spekter uporabe. Pri BPP pa je potrebno upoštevati specifične lastnosti problema, da dosežemo pri križanju dveh rešitev boljše naslednike od staršev. Zato se pri BPP kot gen v kromosomu ne uporablja element ampak posoda, ki vsebuje posodo. Tak način tvorjenja kromosoma (rešitve) se imenuje *Grouping Genetic Algorithm* [5] ali krajše GGA.

GGA se razlikuje od klasičnih GA v dveh pomembnih pogledih. Kodirna shema, ki se uporablja omogoca osnovne dele s katerimi GA deluje. Drugič, za podano kodiranje se uporabljajo posebni genetski operatorji, ki delujejo na kromosomih.

2.2.1 Kodiranje

Če bi uporabili standardno kodiranje kromosoma s predmeti, bi imeli težave pri križanju, saj bi se izgubila informacija o pakiranju. Skupinsko kodiranje rešuje te težave. Za konkreten prikaz skupinskega kodiranja vzemimo naslednji primer razporeditve šestih predmetov označenih od 0 do 5 v posode označene s črkami

```
012345
ADBCEB
```

kar pomeni, da je predmet 0 v posodi označeni z A, 1 v D, 2 in 5 v B, 3 v C. Kromosom predstavljen s skupinami je tako zapisan kot `BECDA`. Vsebino posameznega gena (posode) v kromosomu (skupina posod, ki vsbuje vse elemente) se tako dobi s povezovalno tabelo (*lookup table*). Tako je,

$$A = \{0\}, B = \{2, 5\}, C = \{3\}, D = \{1\}, E = \{4\}.$$

Kromosom bi bil lahko drugače tudi zapisan kot

$$\{0\}\{2, 5\}\{3\}\{1\}\{4\}.$$

Pomembno je poudariti, da genetski operator deluje na skupinskem delu kromosoma. To pa pomeni, da morajo genetski opreratorji delovati s kromosomi spremenljive dolžine. Iz samega kodiranja je razvidno, da razporeditev genov v kromosomu ne vpliva na rešitev. Tako kromosoma BECDA in ABCDE predstavljata isto rešitev.

2.2.2 Križanje

Križanje je operacija pri keterih se lastnosti staršev prenesejo na naslednike. S podanimi strogimi omejitvami in cenilno funkcijo se generirajo skupine (rešitve), ki ne bodo generirale napačnih ali preslabih naslednikov. Postopek križanja bo potekal po naslednjem vzorcu:

1. Izberi dva naključna mesta razdelitve kromosoma pri obeh starših. Za primer imejmo naslednja dva kromosoma:

   ```
   ABCDEF
   abcd
   ```

   ki, ju razsekamo na

```
A|BCD|EF
ab|cd|
```

2. Vstavimo vsebino med rezoma drugega starša na prvi rez prvega starša. Injekiranje tako naredi nov kromosom

```
AcdBCDEF
```

3. Nekateri predmeti v genih se pojavijo dvakrat in jih je zato potrebno odstraniti iz rešitve. Predpostavimo, da se pojavijo elementi ki smo jih injektirali z posodama `c` in `d` tudi v posodah `C, E` in `F`. Te posode izločimo in pri tem nam ostane

```
AcdBD
```

4. Če je potrebno, popravi kromosom tako, da zadovoljiš stroge omejitve in optimiraš cenilno funkcijo. Na tej stopnji se aplicira lokalna hevristika, ki iz nerazporejene predmete poskuša čimbolje razvrstiti v posode. Tu se lahko uporabi polnjenje FFD ali pa dominacijki kriterij[4] z iskanjem zamenjav (HGGA) podobno kot pri *Exact Fit*.

5. Apliciraj točki od 2. do 4. z zamenjano vlogo staršev s ciljem, da generiraš še drugega naslednika.

Iz postopka je razvidno, da GGA promovira obetajoče skupine z dedovanjem. V dodatku A.5 so podani hitrostni testi, ki kažejo da je hitrost dokaj uporabna v primerjavi z EF, če se uporabi FFD za lokalno optimizacijo. Rezultati v [3] kažejo bistveno povečanje potrebnega časa za izračun (do 400 krat) če se uporabi še dominacijki kriterij HGGA. HGGA pa daje nekoliko boljše rešitve kot EF in bi jo bilo mogoče smiselno uporabiti, če bodo testi pokazali uporabne časovne rezultate. Primerno bi bilo tudi določiti velikost problema, ki se rešuje z HGGA in uporabiti GGA z FFD pri večjih problemih.

2.3 Cenilna funkcija

Enostavnoi štetje števila posod ni primerno za praktično uporabo, saj razlike v pakiranju niso toliko razvidne, ker je lahko zadnja posoda malo ali zelo napolnjena. Želeli bi imeti pregled nad celotno kvaliteto pakiranja tako, da promoviramo bolj napolnjene posode. Cenilno funkcijo, ki jo želimo maksimirati, definiramo kot

$$f_{BPP} = \frac{\sum_{i=1..N}(F_i/C)^k}{N} \tag{2.4}$$

kjer je

N	število posod uporabljenih v rešitvi
F_i	vsota velikosti elementov v posodi i
C	kapaciteta posode
k	konstanta, $k > 1$

Konstanta k podaja koncentracijo na zelo dobro napolnjene posode v primerjavi s slabse napolnjenimi pododami. Večji kot je k, bolj promoviramo elitne skupine v primerjavi z približno enako napolnjenimi posodami.

Cenilna funkcija se uporablja predvsem za določitev konca iterativnega iskanja naslednikov. Ko je razlika v naslednikih dovolj mejhna, se preiskovanje prostora zaključi.

Poglavje 3

Dopolnilna rešitev: dodatek za razrez

Pri razrezu profilov je potrebno upoštevati še širino reza, saj lahko to bistveno vpliva na samo razrez, če je rezov več. Upoštevanje dodatka za razrez je rešljivo tako, da vsakemu elementu pred optimiranjem dodamo še dodatek za razrez. Ker pa pri sestavljanju takih elementov zadnji element ni potrebno odrezati je potrebno zagotoviti, da element še vedno pride v poštev pri optimizaciji. To pa zagotovimo tako, da kapaciteto posode oz. velikost profila povečamo za dodatek. Primer sestava z dodatki kaže slika 3. Zgornji del slike predstavlja sestavljene elemente, spodnji pa profil povečan za dodatek razreza.

Slika 3.1: Dodatek zazrez je potrebno dodati tako elementom, kot tudi kapaciteti profila

Problem razreza profilov se tako lahko enostavno transformira v problem polnjenja posod BPP s tem, da vsem profilom in kapaciteti C prištejemo dodatek za razrez. Po rešitvi BPP se lahko izvede obratna transformacija tako, da se dodatek elementom odšteje.

Poglavje 4

Dopolnilna rešitev za splošno skladišče: metoda polnjenja vreč

Problem polnjenja vreč (*Multiple Knapsack Problem*) je problem tatu, ki želi v vreče, ki jih je prinesel s seboj napolniti s čimbolj vrednimi predmeti. Za razliko od BPP je pri MKP[8] omejena velikost vreč, ki so lahko različne in omejeno je število vreč. Predmetov je volumsko več, kot pa je kapaciteta vreč, sicer bi bil problem trivialno rešljiv in bi tat odnesel vse v vrečah, ki jih ima.

Obravnavamo problem kjer n predmetov pakiramo v m vreč različnih kapacitet $c_i, i = 1, \ldots, m$. Vsak predmet j ima dodeljen dobiček p_j in težo w_j. Problem je izbrati m ločenih podmnožic predmetov, tako da vsaka podmnožica i pristoji v vrečo i in da je celotni dobiček čim večji.

Formalno lahko definiramo problem MKP z

$$\begin{aligned} \text{maksimiraj} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \\ \text{glede na} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n \end{aligned} \tag{4.1}$$

Kadar je $x_{ij} = 1$ če je predmet i dodeljen vreči j, sicer je $x_{ij} = 0$. Običajno se predpostavi, da so koeficienti p_j, w_j in c_i pozitivna cela števila. Da se izognemo trivialnim primerom predpostavimo še

$$w_j \leq \max_{i=1,\ldots,m} c_i \quad za \quad j = 1, \ldots, n, \tag{4.2}$$

$$c_i \geq \min_{j=1,\ldots,n} w_j \quad za \quad i = 1, \ldots, m, \tag{4.3}$$

$$\sum_{j=1}^{n} w_j \geq c_i \quad za \quad i = 1, \ldots, m. \tag{4.4}$$

Prva predpostavka zagotavlja, da vsak predmet j gre vsaj v eno vrečo, sicer ga lahko odstranimo iz problema. Če je omejitev (4.3) kršena, potem lahko odstranimo vrečo, saj noben element ne gre v vrečo. Omejitev (4.4) onemogoča trivialno rešitev, ko vsi predmeti gredo v eno od vreč.

4.1 Točni algoritmi

V strgem smislu ima MKP težavnostno stopnjo $\mathcal{NP}$ in bi zato dinamično programiranje rezultiralo s striktno eksponentnim časom računanja. Večina algoritmov[6] tako temelji na *branch-and-bound* načinu iskanja rešitve. Za relativno veliko razmerje n/m obstajajo točne rešitve[8]. Tako je možno za veliko število predmetov $n = 100000$ izračunati točno rešitev v delčku sekunde. Polje iskanja se pri tej metodi zmanjša z določitvijo zgornjih mej, ki pa jih še zmanjčamo z izračunom serije *Subset-Sum Problemov* na vseh vrečah.

4.2 Problem večkratnega seštevka podmnožic

Če v problemu polnjenja vreč, ki ga formulira enačba 4.1, predpostavimo $p_j = w_j$, dobimo posebni primer MKP, ki ga imenujemo *problem večkratnega seštevka podmnoži*. *Multiple Subset Sum Problem* (MSSP). Problem MSSP lahko formuliramo kot

$$\begin{aligned}
\text{maksimiraj} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{ij} \\
\text{glede na} \quad & \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \quad i = 1, \ldots, m, \\
& \sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1, \ldots, n, \\
& x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, \; j = 1, \ldots, n
\end{aligned} \tag{4.5}$$

z enakimi predpostavkami kot pri problemu MKP. Tu je dobiček enakovreden količini predmetov, ki jih polnimo v vreče.

4.3 Aproksimativni algoritmi

Poleg točnih algoritmov je možno razviti tudi različne aproksimativne algoritme, ki delujejo na podobnih osnovah kot pri BPP. Ravno tako je možno uporabiti genetske algoritme. Zahtevnost MSSP pri aproksimativnih algoritmih je običajno

polinomska[1]. Pri algoritmu $H^{\frac{3}{4}}$ [2] je zahtevnost $O(m^2 + n)$. Kvaliteta polnjenja je odvisna od vhodnih podatkov in velikosti vreč.

4.4 Razrez linijskih elementov z upoštevanjem elementov v skladišču

Ob razrezu profilov so lahko ostanki, kljub optimiranju na različne standardne dolžine, dovolj veliki, da se jih splača shraniti v skladišče in jih uporabiti pri naslednjem naročilu (optimiranju). Optimiranje z upoštevanjem skladišča vsebuje naslednje bistvene korake:

1. Za določen ident. je potrebno ugotoviti razpon dolžin (min/max). Iz naročila izberemo vse tiste idente, ki ustrezajo razponu dolžin na skladišču.

2. Če je razmerje n/m dovolj veliko in je $m \leq 12$ lahko uporabimo točno metodo MKP oziroma MSSP. Pri večjem številu elementov na sklavišču m se uporabi PTAS aproksimativna shema. Pri zelo velikem številu m je potrebno izvesti BFD ali genetsko polnjenje vreč. Elemente, ki ni bilo možno uporabiti za polnjenje vreč, se vrne v nedodeljeni izbor za naslednji korak.

3. Z metodo BPP preostale nedodeljene elemente pakiramo v standardne dolžine tako, da za vsako standardno dolžino izvedemo BPP in nato izberemo najustreznejšo za naročilo.

4. Če je standardna dolžina, ki je bila v postopku BPP izbrana bila uporabljena tudi iz skladišča, se problem relaksira tako, da se standardna dolžina izločin iz postopka MKP in se uporabi za BPP. Celoten postopek se ponovi in preveri, če bo dal boljši rezultat, kot je bil pred relaksacijo.

Poglavje 5

Numerični rezultati

5.1 BPP

Pakiranje je bilo izvedeno z različnimi aproksimativnimi algoritmi. Exact Fit daje dobre rezultate v primernem času. Izdelan je bil tudi genetski algoritem za BPP s FFD lokalno optimizacijo , ki naj bi bil kvalitetnejši z daljšim časom optimizacije. Hitrostni rezultati genetskega algoritma so bili nepričakovano dobri, zato bi bilo smiselno uvesti hibridno metodo za še boljše pakiranje z dominacijsko lokalno optimizacijo[7].

Ostanki so bili običajno pod 3%, če ne upoštevamo zelo velike ostanke, ki se jih bo uporabilo pri MSSP v naslednjih optimizacijah. Hitrostne in kvalitativne rezultate, katerih del je videti tudi v dodatku A, je bilo težko primerjati na testnih poadtkih, zaradi manjhnega števila vzorcev. Merodajni so le hitrostni rezultati, kvaliteto pakiranja pa bo potrebno še podrobneje raziskati, saj so podatki specifični v tem smislu, da je opaziti, večje število posod z majhnim številom elementov.

V prilogi so vsi aproskimativni alogoritmi na primeru 3070251 dosegli najmanjše teroretično število posod, saj je skupna dolžina profilov 230130 mm, kar pri 20-ih profilih dolžine 12000mm prinese skupnega 9870mm ostanka.

$$N_{bin,theo} = \text{maxint}\frac{\sum_{i=1}^{N} x_i}{c} = \frac{230130}{12000} = 19.2 \tag{5.1}$$

Enaka je tudi ocena kvakitete pakiranja

$$q = \frac{9870}{20 \cdot 12000}100\% = 4.1\%$$

če se ne upošteva velike elemente, ki bi jih lahko uporabili pri naslednjih optimizacijah.

5.2 MSSP

Pri analizi ostankov in vhodnih podatkov je opaziti, da je velika večina ostankov tako majhnih, da so to pravzaprav odpadki, saj se jih ni dalo uporabiti pri naslednjih

optimizacijah. Zaradi manjšega števila optimizacijo tudi ni bilo možno preveriti MSSP na realnih ostankih. Zato smo ostanke nekaj različnih optimizacij združili v en MSPP problem in s tem dobili zahtevnejši MSPP na realnih ostankih z realimi vhodnimi podatki. Hitrosti za manjše število m so bili dokaj hitri (glej dodatek B).

Izvedena pa je bila le točna rešitev. Ker se predvideva, da skladišče ne bo hranilo večjega števila ostankov, saj se bodo ostanki uporabljali sprotni pri vsaki naslednji optimizaciji, je smiselno uporabiti točno metodo MSSP.

Če se bo pokazala drugačna praksa v skladišču in bi m naraščalo, bo potrebno uvesti aproksimativne metode MSSP (polinomske ali linearne)v odvisnosti od števila in velikosti podatkov. Iz testnih rezultatov je tudi videti, da se običajno ob sprotnem upoštevanju praznenja skladišča, izvajajo le skoraj trivialne rešitve optimizacije MSSP, saj so manjši elementi običajno enako veliki.

Poglavje 6

Zaključek

Predstavljeni postopki so pokazali dovolj učinkovite prijeme za zahteve, ki jih projektna naloga predpisuje. Numerični rezultati su dali zzadovoljive kvalitetne in časovne rezultate na ciljni arhitekturi (PC Celeron 450 MHz). Izvedene so bile tudi hitrostne primerjave med PC in različnimi SGI računalniki, kar je prikazano v dodatku C.

Predpostavlja se, da bo skladišče majhno in bo shranjevalo le relevantne ostanke. Iz danih testnih podatkov ni bilo mogoče zaključiti, kako bo skladišče delovalo na dolgi rok, saj je bilo uporabljeno le pet testnih naročil. Vprašanje tudi je, kako bo, če se bo naročevalo več standardnih profilov, kot jih je potrebno za naročilo. Lahko se zgodi, da je na skladišču kar nekaj standardnih dolžin, ki pa se slabo pakirajo in bi bilo mogoče pametneje naročiti nove standardne profile, ki bi se bolje pakirali, standardne profile pa bi na skladišču pustili. Podobno vprašanje bi lahko zastavili tudi za elemente v skladišču. ki smo jih optimirali z MSSP, ostanki pa so relativno veliki. Mogoče bi bilo dobro uvesti kakšne kriterije tudi za take primere ali prepustiti odločitev uporabniku.

Podrobnejši kriteriji razen max 3% ostanka pri izbiri najustreznejše dolžine niso bili podani. Lahko pa se zgodi, da se bodo ob uporabi programa pokazali primernejši kriteriji, kot je npr. profit oziroma razmerje cena/dolžino profila v povezavi s kakovostjo ostanka. Predvidevam, da je pri optimizaciji potrebno omogočiti uporabniku tudi možnost izbire optimizacije suboptimalnih standardnih dolžin s tem da izbere, katere standardne dolžine dopušča za optimizacijo.

Literatura

[1] Alberto Capara, hans Kepler, and Ulrich Pferschy. The multiple subset sum problem. Technical report, Faculty of Economics, University of Graz, 12 1998.

[2] Alberto Caprara, hans Kepler, and Ulrich Pferschy. A PTAS for the multiple subset sum problem with different knapsack capacities. Technical report, University of Graz, Austria, 1999.

[3] Philip A. Djang and Paul R. Finch. Solving one dimension bin packing problems. *submited to Journal of Heuristics*, june 1998.

[4] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(2):5–30, 1996.

[5] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Wiley Computer Books, 1998.

[6] Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, 1999. accepted for publication.

[7] Silvano Martello and Paolo Tooth. *Knapsack Problems, Algorithms and Computer Implementations*, chapter 8, pages 221–245. John Wiley & Sons, England, 1990.

[8] David Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, 1995.

Dodatek A

Primerjalne hitrosti med algoritmi BPP

Priloženi rezultati metod so bili računani na na SGI ONIX štiriprocesorskem računalniku, vendar se je uporabljal le en procesor pri iskanju rešitve.

A.1 First Fit Descending

Vhodni podatki profilov za ident. št. 3070262 urejeni po padajočem vrstnem redu.

```
6266<40>,6164<39>,6164<38>,6164<37>,6164<36>,5474<35>,
5470<34>,4972<33>,4972<32>,4972<31>,4750<30>,4585<29>,
4382<28>,4358<27>,4308<26>,4308<25>,3959<24>,3765<23>,
3765<22>,3765<21>,3080<16>,3080<17>,3080<18>,3080<19>,
3080<20>,2510<15>,2220<14>,2033<13>,2000<0>,2000<1>,
2000<2>,2000<3>,2000<4>,2000<5>,2000<6>,2000<7>,
2000<8>,2000<9>,2000<10>,2000<11>,2000<12>
```

Rezultat pakiranja po metodi *First Fit*.

```
Bin: 0 11740(12000) 6266<40>,5474<35>
Bin: 1 11634(12000) 6164<39>,5470<34>
Bin: 2 11136(12000) 6164<38>,4972<33>
Bin: 3 11136(12000) 6164<37>,4972<32>
Bin: 4 11136(12000) 6164<36>,4972<31>
Bin: 5 11845(12000) 4750<30>,4585<29>,2510<15>
Bin: 6 11820(12000) 4382<28>,4358<27>,3080<16>
Bin: 7 11696(12000) 4308<26>,4308<25>,3080<17>
Bin: 8 11489(12000) 3959<24>,3765<23>,3765<22>
Bin: 9 11958(12000) 3765<21>,3080<18>,3080<19>,2033<13>
Bin: 10 11300(12000) 3080<20>,2220<14>,2000<0>,2000<1>,
                     2000<2>
```

```
Bin: 11 12000(12000) 2000<3>,2000<4>,2000<5>,2000<6>,
                     2000<7>,2000<8>
Bin: 12 8000(12000) 2000<9>,2000<10>,2000<11>,2000<12>
```

Podatki urejeni po velikosti za ident 3070251 in rezultat pakiranja na 20 profilov. Cas potreben za izracun 0.002 sekunde.

```
7000<66>,7000<65>,7000<64>,7000<63>,6480<62>,6480<61>,
6450<60>,6450<59>,450<58>,6450<57>,6278<56>,6278<55>,
6212<54>,6160<53>,6000<50>,6000<51>,6000<52>,5700<49>,
5700<48>,5500<47>,5450<46>,5000<45>,5000<44>,4363<38>,
4363<39>,4363<40>,4363<41>,4363<42>,4363<43>,4000<37>,
4000<36>,3550<33>,3550<34>,3550<35>,2560<32>,2510<31>,
2500<30>,2500<29>,2500<28>,2500<27>,2500<26>,2500<25>,
2460<24>,2034<23>,2000<21>,2000<22>,700<0>,700<1>,
700<2>,700<3>,700<4>,700<5>,700<6>,700<7>,700<8>,700<9>,
700<10>,700<11>,700<12>,700<13>,700<14>,700<15>,700<16>,
700<17>,700<18>,700<19>,700<20>

Bin:0(12000+0)0[7000] 21[5000]
Bin:1(12000+0)1[7000] 22[5000]
Bin:2(11363+637)2[7000] 23[4363]
Bin:3(11363+637)3[7000] 24[4363]
Bin:4(11980+20)4[6480] 19[5500]
Bin:5(11930+70)5[6480] 20[5450]
Bin:6(11513+487)6[6450] 25[4363] 46[700]
Bin:7(11513+487)7[6450] 26[4363] 47[700]
Bin:8(11513+487)8[6450] 27[4363] 48[700]
Bin:9(11513+487)9[6450] 28[4363] 49[700]
Bin:10(11978+22)10[6278] 17[5700]
Bin:11(11978+22)11[6278] 18[5700]
Bin:12(11612+388)12[6212] 29[4000] 50[700] 51[700]
Bin:13(11560+440)13[6160] 30[4000] 52[700] 53[700]
Bin:14(12000+0)14[6000] 15[6000]
Bin:15(11584+416)16[6000] 31[3550] 43[2034]
Bin:16(11660+340)32[3550] 33[3550] 34[2560] 44[2000]
Bin:17(11410+590)35[2510] 36[2500] 37[2500] 38[2500]
                  54[700] 55[700]
Bin:18(11960+40)39[2500] 40[2500] 41[2500] 42[2460] 45[2000]
Bin:19(7700+4300)56[700] 57[700] 58[700] 59[700] 60[700]
            61[700] 62[700] 63[700] 64[700] 65[700] 66[700]
0.002u 0.007s 0:00.00 0.0% 0+0k 0+0io 0pf+0w
```

Ostanki urejeni po velikosti: 4300, 637, 637, 590, 487, 487, 487, 487, 440, 416, 388, 340, 70, 40, 22, 22, 20, 0, 0, 0, Skupaj: 9870

A.2 Exact Fit

Ident št. 3070251 je potreboval po metodi *Exact Fit* 1.566 sekunde. Potrebno število palic je 20, tako kot pri metodi FirstFit.

```
 Ident: 3070251  Bin capacity: 12000     Cuting addition: 0
Total elements: 67
(700)(700)(700)(700)(700)(700)(700)(700)(700)(700)(700)
(700)(700)(700)(700)(700)(700)(700)(700)(700)(700)
(2000)(2000)(2034)(2460)(2500)(2500)(2500)(2500)(2500)(2500)
(2510)(2560)(3550)(3550)(3550)(4000)(4000)(4363)(4363)(4363)
(4363)(4363)(4363)(5000)(5000)(5450)(5500)(5700)(5700)(6000)
(6000)(6000)(6160)(6212)(6278)(6278)(6450)(6450)(6450)(6450)
(6480)(6480)(7000)(7000)(7000)(7000)
7000+5000=12000 + 0 odpada
7000+5000=12000 + 0 odpada
7000+2500+2500=12000 + 0 odpada
7000+2500+2500=12000 + 0 odpada
6480+5500=11980 + 20 odpada
6480+5450=11930 + 70 odpada
6450+2000+3550=12000 + 0 odpada
6450+2000+3550=12000 + 0 odpada
6450+700+700+4000=11850 + 150 odpada
6450+700+700+4000=11850 + 150 odpada
6278+700+2460+2560=11998 + 2 odpada
6278+700+2500+2510=11988 + 12 odpada
6212+700+700+4363=11975 + 25 odpada
6160+700+700+4363=11923 + 77 odpada
6000+6000=12000 + 0 odpada
6000+700+700+4363=11763 + 237 odpada
5700+700+2034+3550=11984 + 16 odpada
5700+700+700+4363=11463 + 537 odpada
4363+700+2500+4363=11926 + 74 odpada
700+700+700+700+700=12000 + 0 odpada
1.566u 0.009s 0:01.60 97.5% 0+0k 0+0io 0pf+0w
```

A.3 Last Fit Descending

```
Bin:0(10550+1450)0[7000] 31[3550]
Bin:1(11000+1000)1[7000] 30[4000]
Bin:2(11000+1000)2[7000] 29[4000]
Bin:3(11363+637)3[7000] 28[4363]
Bin:4(10843+1157)4[6480] 27[4363]
Bin:5(11543+457)5[6480] 26[4363] 66[700]
```

```
Bin:6(11513+487)6[6450] 25[4363] 65[700]
Bin:7(11513+487)7[6450] 24[4363] 64[700]
Bin:8(11513+487)8[6450] 23[4363] 63[700]
Bin:9(11450+550)9[6450] 22[5000]
Bin:10(11978+22)10[6278] 21[5000] 62[700]
Bin:11(11728+272)11[6278] 20[5450]
Bin:12(11712+288)12[6212] 19[5500]
Bin:13(11860+140)13[6160] 18[5700]
Bin:14(12000+0)14[6000] 15[6000]
Bin:15(11700+300)16[6000] 17[5700]
Bin:16(11660+340)32[3550] 33[3550] 34[2560] 44[2000]
Bin:17(11410+590)35[2510] 36[2500] 37[2500] 38[2500]
                 60[700] 61[700]
Bin:18(11994+6)39[2500] 40[2500] 41[2500] 42[2460] 43[2034]
Bin:19(11800+200)45[2000] 46[700] 47[700] 48[700] 49[700]
                  50[700] 51[700] 52[700] 53[700] 54[700]
                 55[700] 56[700] 57[700] 58[700] 59[700]
```

Ostanki urejeni po velikosti: 1450, 1157, 1000, 1000, 637, 590, 550, 487, 487, 487, 457, 340, 300, 288, 272, 200, 140, 22, 6, 0. Skupaj: 9870.

A.4 Best Fit Descending

```
Bin:0(12000+0)0[7000] 21[5000]
Bin:1(12000+0)1[7000] 22[5000]
Bin:2(11363+637)2[7000] 23[4363]
Bin:3(11363+637)3[7000] 24[4363]
Bin:4(11980+20)4[6480] 19[5500]
Bin:5(11930+70)5[6480] 20[5450]
Bin:6(11513+487)6[6450] 25[4363] 46[700]
Bin:7(11513+487)7[6450] 26[4363] 47[700]
Bin:8(11513+487)8[6450] 27[4363] 48[700]
Bin:9(11513+487)9[6450] 28[4363] 49[700]
Bin:10(11978+22)10[6278] 17[5700]
Bin:11(11978+22)11[6278] 18[5700]
Bin:12(11612+388)12[6212] 29[4000] 50[700] 51[700]
Bin:13(11560+440)13[6160] 30[4000] 52[700] 53[700]
Bin:14(12000+0)14[6000] 15[6000]
Bin:15(11550+450)16[6000] 31[3550] 45[2000]
Bin:16(11660+340)32[3550] 33[3550] 34[2560] 44[2000]
Bin:17(11410+590)35[2510] 36[2500] 37[2500] 38[2500]
                  54[700] 55[700]
Bin:18(11994+6)39[2500] 40[2500] 41[2500] 42[2460] 43[2034]
Bin:19(7700+4300)56[700] 57[700] 58[700] 59[700] 60[700]
```

```
                61[700] 62[700] 63[700] 64[700] 65[700] 66[700]
```

Ostanki urejeni po velikosti: 4300, 637, 637, 590, 487, 487, 487, 487, 450, 440, 388, 340, 70, 22, 22, 20, 6, 0, 0, 0. Skupaj: 9870.

Vsi ostanki so kratki in jih lahko okarakteriziramo kot odpadek

$$\frac{6630}{21 * 12000} = 2.63\% \, ostanka$$

A.5 Grouping Genetic Algorithm

Skupinski gentski algoritem s vstavljanjem po metodi FFD.

```
Bin:0(11950+50)0[7000] 31[3550] 46[700] 62[700]
Bin:1(11363+637)3[7000] 28[4363]
Bin:2(11994+6)39[2500] 40[2500] 41[2500] 42[2460] 43[2034]
Bin:3(11560+440)13[6160] 30[4000] 52[700] 53[700]
Bin:4(11700+300)16[6000] 21[5000] 49[700]
Bin:5(11800+200)1[7000] 45[2000] 50[700] 51[700] 54[700]
                55[700]
Bin:6(11513+487)8[6450] 23[4363] 63[700]
Bin:7(11450+550)9[6450] 22[5000]
Bin:8(11978+22)10[6278] 17[5700]
Bin:9(11978+22)11[6278] 18[5700]
Bin:10(11543+457)5[6480] 26[4363] 66[700]
Bin:11(11513+487)6[6450] 25[4363] 65[700]
Bin:12(11712+288)12[6212] 19[5500]
Bin:13(12000+0)14[6000] 15[6000]
Bin:14(11660+340)32[3550] 33[3550] 34[2560] 44[2000]
Bin:15(11410+590)35[2510] 36[2500] 37[2500] 38[2500]
                60[700] 61[700]
Bin:16(8950+3050)20[5450] 47[700] 56[700] 57[700]
                58[700] 59[700]
Bin:17(11543+457)4[6480] 27[4363] 48[700]
Bin:18(11513+487)7[6450] 24[4363] 64[700]
Bin:19(11000+1000)2[7000] 29[4000]
0.003u 0.008s 0:00.01 0.0% 0+0k 0+0io 0pf+0w
```

Ostanki urejeni po vekikosti: 3050, 1000, 637, 590, 550, 487, 487, 487, 457, 457, 440, 340, 300, 288, 200, 50, 22, 22, 6, 0. Skupaj: 9870.

Dodatek B

Polnjenje vreč

Prikazani so rezultati polnjenja vreč za naročilo 3070251. Skladišče vsebuje naslednje dolžine elementov: 765, 800, 800, 850, 930, 1020, 1111, 1111, 1111, 1111, 1294,1320 Rezultat pakiranja je seveda zelo enostaven:

```
 n = 21, m = 12
knapsack 1 [765] = [700] + 65 ostanka
knapsack 2 [800] = [700] + 100 ostanka
knapsack 3 [800] = [700] + 100 ostanka
knapsack 4 [850] = [700] + 150 ostanka
knapsack 5 [930] = [700] + 230 ostanka
knapsack 6 [1020] = [700] + 320 ostanka
knapsack 7 [1111] = [700] + 411 ostanka
knapsack 8 [1111] = [700] + 411 ostanka
knapsack 9 [1111] = [700] + 411 ostanka
knapsack 10 [1111] = [700] + 411 ostanka
knapsack 11 [1294] = [700] + 594 ostanka
knapsack 12 [1320] = [700] + 620 ostanka
0.001u 0.005s 0:00.00 0.0% 0+0k 0+0io 0pf+0w
```

Vidimo, da smo sicer uporabili kar nekaj 700 mm elementov ampak s tem smo tudi sproducirali dokaj neuporabne dolžine novih ostankov. Hitrostni rezutati zo podani za Onyx2.

Povečanje razpona ostankov na skladišču pomeni tudi povečanje možnih kandidatov za vreče.

```
700, 700, 700, 700, 700, 700, 700, 700, 700, 700,
700, 700, 700, 700, 700, 700, 700, 700, 700, 700,
700, 2000, 2000, 2034, 2460, 2500, 2500, 2500,
2500, 2500, 2500, 2510, 2560, 3550, 3550, 3550,
4000, 4000, 4363,4363, 4363, 4363, 4363, 4363,
5000, 5000, 5450, 5500
```

```
n = 48, m = 16
knapsack 1 [765] = [700] + 65 ostanka
knapsack 2 [800] = [700] + 100 ostanka
knapsack 3 [800] = [700] + 100 ostanka
knapsack 4 [850] = [700] + 150 ostanka
knapsack 5 [930] = [700] + 230 ostanka
knapsack 6 [1020] = [700] + 320 ostanka
knapsack 7 [1111] = [700] + 411 ostanka
knapsack 8 [1111] = [700] + 411 ostanka
knapsack 9 [1111] = [700] + 411 ostanka
knapsack 10 [1111] = [700] + 411 ostanka
knapsack 11 [1294] = [700] + 594 ostanka
knapsack 12 [1320] = [700] + 620 ostanka
knapsack 13 [3002] = [700][700][700][700] + 202 ostanka
knapsack 14 [5300] = [700][700][700][700][2500] + 0 ostanka
knapsack 15 [6531] = [2510][4000] + 21 ostanka
knapsack 16 [11200] = [700][2000][2000][2500][4000]+0 ost.
0.011u 0.006s 0:00.01 100.0% 0+0k 0+0io 0pf+0w
```

Dodatek C

Primerjalne hitrosti med računalniki

Meritve hitrosti so podane zaradi lažjega primerjanja pri razvoju programov za pakiranje. Program ne uporablja veliko spomina in le celoštevilčno aritmetiko. Izkazalo se je, da je PC s Celeronom 450 MHz hitrejši od SGI Indigo2 in le za melenkost počasnejši od SGI Onyx. Seveda pa na večprocesorskih računalnikih niso bili izkoriščeni vsi procesorji ampak le eden.

```
Indigo2 R4400  150 MHz: mipspro -O3
1233566677 : 55
7123356676 : 54
7123367566 : 48
6123756673 : 46
5.949u 0.038s 0:06.14 97.2% 0+0k 3+0io

Indigo2 MIPS10000 195 MHz impact : -64 -mips4 -O3
1233566677 : 55
7123356676 : 54
7123367566 : 48
6123756673 : 46
2.676u 0.020s 0:02.77 97.1% 0+0k 0+0io 0pf+0w

Onyx 4 250MHz MIPS R10000(IP27) with MIPS R10010 FPU:
            -mips4 -n32 -O3
1233566677 : 55
7123356676 : 54
7123367566 : 48
6123756673 : 46
1.982u 0.008s 0:01.99 99.4% 0+0k 0+0io 0pf+0w

Linux: Celeron 450 g++ -O3
```

```
1233566677 : 55
7123356676 : 54
7123367566 : 48
6123756673 : 46

real    0m2.368s
user    0m2.370s
sys     0m0.000s

trimo2: comparission

g++ -O3 -Wall -I./STL    trimo2.cc   -o trimo2
Celeron 450:
leon@achilles:~/Current/trimo$ time ./trimo2
1233566677 : 55
1233576676 : 54
1236375766 : 48
1236576637 : 46

real    0m0.111s
user    0m0.110s
sys     0m0.000s

Onyx:
1233566677 : 55
1233576676 : 54
1236375766 : 48
1236576637 : 46
0.105u 0.007s 0:00.11 90.9% 0+0k 0+0io 0pf+0w
```