Poopćeni problem dodjeljivanja

Ćustić, Porin

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Porin Ćustić

POOPĆENI PROBLEM DODJELJIVANJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, srpanj, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Za mamu i tatu.
Hvala vam na svemu.

[0, 1]

Sadržaj

Uvod

Tijekom godina istraživanja iz područja kombinatorne optimizacije i teorijske računarske znanosti otišla su toliko daleko da i najveći stručnjaci iz tog područja teško mogu pratiti njihov razvoj. Međutim, neki problemi su preživjeli do danas, postali fundamentalni problemi područja te su i dalje zanimljivi mnogobrojnim znanstvenicima. Ovaj diplomski rad bavi se jednim takvim problemom, poopćenim problemom dodjeljivanja.

Pretpostavimo filmski scenarij - zamislite da ste lopov. Spretno ste provalili u muzej, došli do velikog sefa u kojem su najskuplji izlošci muzeja i spremni ste pograbiti razne vrijedne predmete. No, suočavate se s problemom - imate malu naprtnjaču a predmeta je mnogo. Željeli biste znati koje predmete odabrati da stanu u naprtnjaču, a da zajedno što više vrijede. Taj problem je poznat kao problem naprtnjače (eng. *Knapsack Problem*).

No, možemo se pitati što se događa kada nemamo samo jednu naprtnjaču nego više njih te kada težine i vrijednosti predmeta koje ćemo uzeti ovise i o naprtnjačama u koje ćemo ih staviti. Takav problem se naziva poopćeni problem dodjeljivanja (eng. *Generalized Assignment Problem*) i to je problem kojim se bavi ovaj diplomski rad.

U prvom poglavlju ćemo navesti neke osnovne matematičke pojmove i pojmove teorije grafova koje će nam biti potrebne za razmatranje problema. U drugom poglavlju ćemo opisati i analizirati linearni problem dodjeljivanja (eng. *Linear Assignment Problem*), navesti neke algoritme za njegovo rješavanje i njihovu složenost. U trećem poglavlju opisat ćemo problem naprtnjače, navesti i analizirati neke algoritme za rješavanje problema te na kraju poglavlja dati pregled još nekih verzija problema naprtnjače. U četvrtom i petom poglavlju ćemo se baviti samim poopćenim problemom dodjeljivanja - opisat ćemo i analizirati problem te navesti i pobliže objasniti nekoliko algoritama za njegovo rješavanje. Također ćemo reći nešto više o implementaciji algoritama obrađenih u radu te navesti neke eksperimentalne rezultate.

Poglavlje 1

Osnovni matematički pojmovi

U ovom poglavlju navodimo neke osnovne pojmove iz teorije grafova i druge matematičke pojmove koji će nam biti nužni za razumijevanje problema koje obrađuje ovaj diplomski rad.

Definicija 1.1. ***Graf*** *je uređeni par* (V, E) *gdje je* V *neprazan skup elemenata koje nazivamo vrhovima, a* $E \subseteq V \times V$*. Elemente skupa* E *nazivamo lukovima.*

Ovakva definicija grafa podrazumijeva usmjerene bridove, no u radu ćemo najčešće pod pojmom graf smatrati neusmjereni graf, odnosno smatrat ćemo da je E simetrična relacija. Obično se u literaturi lukovi neusmjerenih grafova nazivaju bridovima, pa ćemo i mi u ovom radu koristiti taj naziv. Posebno ćemo istaknuti ako bude riječ o usmjerenom grafu.
Kažemo da su dva vrha u grafu susjedna ako postoji brid koji ih spaja. Formalnije:

Definicija 1.2. *Vrh* u *je* ***susjedan*** *(incidentan) vrhu* v *ako vrijedi* $(u, v) \in E$*.*

Definicija 1.3. ***Podgraf*** *grafa* $G = (V, E)$ *je graf* $G' = (V', E')$ *za koji vrijedi* $V' \subseteq V$ *te* $E' \subseteq E$ *takav da za svaki* $(a, b) \in E'$ *vrijedi da su* $a, b \in V'$*.*

Definicija 1.4. ***Put*** *(šetnja) u grafu* $G = (V, E)$ *je niz vrhova* $(v_1, v_2, ..., v_k)$ *pri čemu su vrhovi* v_i *i* v_{i+1} *susjedni za sve* $i = 1, 2, ..., k - 1$*.*

Definicija 1.5. *Za graf* $G = (V, E)$ *gdje je skup* V *disjunktna unija skupova* U_1 *i* U_2 *kažemo da je* ***bipartitan graf*** *ako svaki brid iz* E *spaja vrh iz* U_1 *s vrhom iz* U_2*.*

U radu ćemo za bipartitan graf koristiti oznaku formata $G = (U_1, U_2; E)$.

Definicija 1.6. ***Sparivanje*** M *u grafu* $G = (V, E)$ *je podskup bridova grafa* G *za koji vrijedi da je svaki vrh od* G *vezan s najviše jednim bridom iz* M*.*

Opisno, hiperravninu definiramo kao podskup nekog prostora dimenzije za 1 manje od tog prostor. Tako je za prostor dimenzije $n = 2$ hiperravnina je pravac, a za prostor dimenzije $n = 3$, hiperravnina je dvodimenzionalna ravnina.

Definicija 1.7. ***Politop*** *je ograničeno konveksno područje n-dimenzionalnog prostora omeđeno s konačno mnogo hiperravnina.*

Definicija 1.8. ***Konveksna kombinacija*** *je linearna kombinacija točaka iz afinog prostora u kojoj su svi koeficijenti nenegativni i čiji je zbroj jednak* 1.

Poglavlje 2

Linearni problem dodjeljivanja

2.1 Opis problema

Pretpostavimo da imamo n poslova koje treba odraditi te n radnika koje trebamo platiti da odrade te poslove. Kako su svi poslovi i radnici različiti, za svaki par radnika i posla (i, j) znamo koliko će nas koštati da radnik i odradi posao j. Cilj nam je svakom radniku dodijeliti po jedan posao, tako da nam trošak bude minimalan.

Označimo radnike brojevima $1, 2, \ldots, n$, poslove također brojevima $1, 2, \ldots, n$, te neka je φ permutacija skupa $\{1, 2, \ldots, n\}$. Tada vidimo da permutacija φ označava jedan način dodjeljivanja poslova radnicima, pri čemu $\varphi(i) = j$ za $i, j \in \{1, 2, \ldots, n\}$ označava da smo radniku i dodijelili posao j.

Neka je $C = (c_{ij})$ matrica troška, to jest $n \times n$ matrica gdje c_{ij} označava trošak potreban da radnik i odradi posao j. Ako uzmemo da je naš ukupni trošak zbroj svih pojedinih troškova, te ako sa S_n označimo skup svih permutacija skupa $\{1, 2, \ldots, n\}$, onda naš problem možemo formalno zapisati kao

$$\min_{\varphi \in S_n} \sum_{i=1}^{n} c_{i\varphi(i)} \tag{2.1}$$

Time smo definirali linearni problem dodjeljivanja (eng. *Linear Sum Assignment Problem*) gdje je ciljna funkcija zadana sumom.

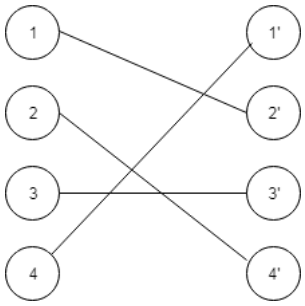
Svaku permutaciju φ možemo na jedinstven način prikazati kao permutacijsku $n \times n$ matricu $X_\varphi = (x_{ij})$ pri čemu vrijedi

$$x_{ij} = \begin{cases} 1 & \text{ako } \varphi(i) = j \\ 0 & \text{inače} \end{cases}$$

Tada permutaciju (dodjeljivanje) možemo prikazati na tri načina:

$$\varphi = (2 \quad 4 \quad 3 \quad 1)$$

$$X_\varphi = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



Slika 2.1: Prikaz permutacije na tri načina

Lako vidimo da permutacijsku matricu X_φ karakteriziraju sljedeća svojstva

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n) \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n) \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, n) \tag{2.4}$$

Naime, svojstvo (2.4) kaže da su svi elementi matrice 0 ili 1, a onda svojstva (2.2) i (2.3) kažu da se u svakom retku i stupcu, respektivno, element 1 pojavljuje točno jednom. Uvjeti (2.2) – (2.4) se nazivaju ograničenja dodjeljivanja (eng. *assignment constraints*).

Ako permutaciju φ prikažemo kroz permutacijsku matricu $X = (x_{ij})$, onda naš problem možemo zapisati na sljedeći način

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.5}$$

uz uvjete

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n) \tag{2.6}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n) \tag{2.7}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j = 1, 2, \ldots, n) \tag{2.8}$$

Linearni problem dodjeljivanja može se gledati i na drugi način - kroz termine teorije grafova. Neka je $G = (U, V; E)$ bipartitan graf za kojeg vrijedi $|U| = |V| = n$. Sparivanje M za koje vrijedi da je svaki vrh grafa G vezan s točno jednim bridom iz M zovemo savršeno sparivanje (eng. *perfect matching*). Poistovjetimo sada vrhove iz skupa U s radnicima, a vrhove iz V s poslovima, te svakom bridu $(i, j) \in E$ pridijelimo težinu koja odgovara iznosu c_{ij}, odnosno trošku da radnik i odradi posao j. Sada naš linearni problem dodjeljivanja postaje problem pronalaska savršenog sparivanja minimalne težine u bipartitnom grafu (eng. *Minimum-cost Weighted Bipartite Matching*).

U ovom radu obrađujemo linearni problem dodjeljivanja u kojemu je konačni trošak izražen kao suma troškova združenih radnika i poslova, međutim to nije jedina mogućnost. Možemo zamisliti situaciju u kojoj su naši radnici računalni procesori, poslovi računalni programi, odnosno sljedovi računalnih instrukcija, a trošak vrijeme potrebno za izvršavanje tih programa. Budući da bi te zadatke mogli izvršavati paralelno, konačan trošak bi bio izražen kao maksimum vremena izvršavanja svih programa. Ovaj problem se u stručnoj literaturi naziva *Linear Bottleneck Assignment Problem*. Problem možemo formalno zapisati kao

$$\min_{\varphi \in S_n} \max_{1 \le i \le n} c_{i\varphi(i)}$$

Ako opet permutaciju φ zapišemo kao permutacijsku matricu $X = (x_{ij})$, onda ovaj problem možemo zapisati kao

$$\min \max_{1 \le i, j \le n} c_{ij} x_{ij}$$

uz uvjete

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n)$$

$$x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, n)$$

2.2 Analiza problema

U prošlom potpoglavlju predstavili smo linearni problem dodjeljivanja kroz termine teorije grafova. Ono što nas zanima, prije nego što pokušamo riješiti problem, jest kako ćemo znati postoji li uopće rješenje problema, odnosno, postoji ili ijedno savršeno sparivanje u našem bipartitnom grafu.

Za vrh $i \in U$, s $N(i)$ označimo skup svih njegovih susjeda, to jest, svih vrhova $j \in V$ koji su spojeni s vrhom i. To možemo proširiti i na svaki podskup U' od U sa $N(U') = \bigcup_{i \in U'} N(i)$

Teorem 2.1. (Hall [13]) Neka je $G = (U, V; E)$ bipartitan graf za kojeg vrijedi $|U| = |V|$. Tada postoji savršeno sparivanje u G ako i samo ako za svaki podskup U' od U vrijedi

$$|U'| \leq |N(U')|$$

Ovaj teorem je poznat i kao teorem o ženidbi (eng. *Marriage theorem*).

U nastavku ovog potpoglavlja navodimo rezultate koji će nam omogućiti da linearni problem dodjeljivanja prikažemo kao problem linearnog programiranja te iskoristimo metode rješavanja linarnog programa za rješavanje našeg problema.

Definicija 2.2. *Dvostruko stohastička matrica $X = (x_{ij})$ je $n \times n$ matrica sa sljedećim svojstvima*

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n)$$

$$x_{ij} \geq 0 \quad (i, j = 1, 2, \ldots, n)$$

Očito je svaka permutacijska matrica dvostruko stohastička pa se svako dodjeljivanje φ može prikazati dvostruko stohastičkom matricom u kojoj je $x_{ij} \in \{0, 1\}$ za $i, j = 1, 2, \ldots, n$. Skup svih dvostruko stohastičkih matrica P_A se naziva politop dodjeljivanja ili Birkhoffov politop.

Teorem 2.3. (Birkhoff [5]) Svaka dvostruko stohastička matrica može se zapisati kao konveksna kombinacija permutacijskih matrica. Specijalno, svaki vrh politopa dodjeljivanja odgovara nekoj permutacijskoj matrici.

Uzmimo sad $m \times n$ matricu A i politop $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ i promotrimo linearni program oblika

$$\min\{c'x : Ax = b, x \geq 0\}$$

Pretpostavimo da je matrica A punog ranga m, odnosno da su redovi matrice linearno nezavisni. Podskup B koji sadrži m linearno nezavisnih stupaca matrice A zovemo baza. Stupci iz B se mogu spojiti da formiraju baznu matricu A_B. Slično, komponente x_j vektora x za koje je stupac j u B možemo iskoristiti da formiramo vektor x_B. Očito je tada bazna matrica regularna $m \times m$ matrica pa jednadžba $A_B x_B = b$ ima jedinstveno rješenje x_B. Za B kažemo da je moguća baza ako $x_B \geq 0$. Rješenje x_B i $x_j = 0$ za $j \notin B$ zovemo bazno rješenje. Ako je B moguća baza, za pripadno bazno rješenje kažemo da je to i moguće rješenje te znamo da ono odgovara vrhovima politopa P.

Središnji teorem linearnog programiranja kaže da, ako postoji konačno optimalno rješenje linearnog programa, onda postoji vrh pripadnog politopa $\{x : Ax = b, x \geq 0\}$ u kojem se to optimalno rješenje postiže ([7]). Sada nam Birkhoffov teorem omogućava da uvjet

$$x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, n) \tag{2.9}$$

zamijenimo s

$$0 \leq x_{ij} \leq 1 \quad (i, j = 1, 2, \ldots, n) \tag{2.10}$$

zbog činjenice da svako moguće bazno rješenje linearnog programa

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.11}$$

uz uvjete

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n) \tag{2.12}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n) \tag{2.13}$$

$$0 \leq x_{ij} \leq 1 \quad (i, j = 1, 2, \ldots, n) \tag{2.14}$$

odgovara vrhu politopa dodjeljivanja, odnosno, svako bazno rješenje je permutacijska matrica koja zadovoljava naš početni uvjet (2.9) . Ovime smo omogućili da linearni problem dodjeljivanja možemo rješavati tehnikama linearnog programiranja.

2.3 Algoritmi za rješavanje

Prvi algoritam za rješavanje linearnog problema dodjeljivanja je predstavio Easterfield 1946. godine ([12]). Bio je to algoritam vremenske složenosti $O(2^n n^2)$. Prvi polinomijalni algoritmi za rješavanje problema bili su primal-dual algoritmi. Najpoznatiji među njima, poznata mađarska metoda koju je predstavio Kuhn ([17],[18]) sredinom 50-ih godina prošlog stoljeća, u originalu rješava problem u $O(n^4)$ vremena. Kasnije varijacije mađarske metode, koje koriste izračunavanje najkraćih puteva, vremenske složenosti $O(n^3)$ dugo su vremena bile najefikasnije praktično rješenje linearnog problema dodjeljivanja. Prvi primal simplex algoritam za rješavanje predstavio je Cunningham 1976. godine ([10]), a tek je 1993. Akgul ([1]) predstavio $O(n^3)$ primal simplex algoritam. Prvi dual algoritam predstavili su Dinic i Kronrod ([11]) 1969. godine. Uz sve navedene algoritme postoje i algoritmi čija je vremenska složenost manja od $O(n^3)$, a istraživanja vezana uz linearni problem dodjeljivanja aktivna su i danas.

U ovom potpoglavlju opisat ćemo $O(n^4)$ verziju mađarske metode koja koristi termine linearnog programiranja i prikaz problema terminima teorije grafova.

Ako pridružimo dualne varijable u_i i v_j ograničenjima dodjeljivanja (2.12) i (2.13) respektivno, dobivamo dualni problem našeg problema

$$\max \quad \left(\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \right) \tag{2.15}$$

uz uvjet

$$u_i + v_j \leq c_{ij} \quad (i, j = 1, 2, \ldots, n) \tag{2.16}$$

Po teoremu komplementarne labavosti (eng. *complementary slackness*, vidi [23]) slijedi da je par mogućih rješenja za primal i dual problem optimalan ako i samo ako vrijedi

$$x_{ij}(c_{ij} - u_i - v_j) = 0 \quad (i, j = 1, 2, \ldots, n) \tag{2.17}$$

Sada možemo definirati

$$\overline{c_{ij}} = c_{ij} - u_i - v_j \quad (i, j = 1, 2, \ldots, n) \tag{2.18}$$

U terminima linearnog programiranja se elementi $\overline{c_{ij}}$ nazivaju reduciranim troškovima. Time smo dobili i transformaciju matrice troška C u matricu $\overline{C}$. Ta transformacija je specijalni slučaj nečega što se naziva *dopuštena transformacija* ([6]). Sada imamo

$$\sum_{i=1}^{n} \sum_{j=1}^{n} (c_{ij} - u_i - v_j) x_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} - \sum_{i=1}^{n} u_i \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} v_j \sum_{i=1}^{n} x_{ij} =$$

$$= \text{zbog (2.12) i (2.13)} = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n} u_i - \sum_{j=1}^{n} v_j \tag{2.19}$$

Vidimo da se za svako moguće rješenje X, ciljne funkcije našeg problema zadane matricom C i $\overline{C}$ razlikuju za konstantu $\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$.

Kao početni korak mađarske metode izvodi se predprocesiranje u kojem se dolazi do mogućeg dual i parcijalnog primal rješenja. Jednu takvu osnovnu metodu vremenske složenosti $O(n^2)$ navodimo ovdje. Za $j = 1, 2, \ldots, n$ uvedimo oznaku

$$row(j) = \begin{cases} i & \text{ako je stupac } j \text{ dodijeljen retku } i \\ 0 & \text{ako stupac } j \text{ nije dodijeljen} \end{cases}$$

Očito vrijedi veza

$$row(j) = i \iff \varphi(i) = j$$

Algoritam 1 Predprocesiranje

1: **for** $i = 1$ **to** n **do**:
2: **for** $j = 1$ **to** n **do**:
3: $x_{ij} = 0$
4: **for** $i = 1$ **to** n **do**:
5: $u_i = \min\{c_{ij} : j = 1, 2, \ldots, n\}$
6: **for** $j = 1$ **to** n **do**:
7: $v_j = \min\{c_{ij} - u_i : i = 1, 2, \ldots, n\}$
8: **for** $j = 1$ **to** n **do**:
9: $row(j) = 0$
10: **for** $i = 1$ **to** n **do**:
11: **for** $j = 1$ **to** n **do**:
12: **if** $row(j) = 0$ **and** $c_{ij} - u_i - v_j == 0$ **then**:
13: $x_{ij} = 1$
14: $row(j) = i$
15: **break**

U linijama $1 - 3$ inicijaliziramo matricu dodjeljivanja. Potom u linijama $4 - 5$ postavljamo u_i na minimalnu vrijednost matrice troška u retku i. U linijama $6 - 7$ postavljamo v_j na minimalnu vrijednost izraza $c_{ij} - u_i$ u stupcu j. U linijama $10 - 15$ prolazimo kroz matricu, provjeravamo na kojim smo mjestima matrice troška došli do nule, i ako taj stupac nije dodjeljen ni jednom retku, dodijelimo stupac j retku i. Ono što u ovom predprocesiranju radimo je to da svakom retku najprije oduzmemo najmanji element u tom retku, a onda

svakom stupcu oduzmemo najmanji element u tom stupcu, nakon oduzimanja po retcima. Time je transformirana matrica troška, sve njezine vrijednosti su i dalje nenegativne, a nule u matrici daju kandidate za optimalno primal rješenje. Može se dogoditi da u svakom retku i svakom stupcu nakon predprocesiranja bude točno jedna nula čime bismo odmah našli optimalno rješenje. Također, vidimo da vrijednosti u_i i v_j zadovoljavaju uvjet (2.16).

Primjer 2.4. *Imamo zadanu matricu troška C, izračunajmo vrijednosti u_i, v_i te transformiranu matricu $\overline{C}$.*

$$C = \begin{bmatrix} 7 & 9 & 8 & 9 \\ 2 & 8 & 5 & 7 \\ 1 & 6 & 6 & 9 \\ 3 & 6 & 2 & 2 \end{bmatrix}$$

Sada imamo $u = (7, 2, 1, 2)$ i $v = (0, 2, 0, 0)$. Transformirana matrica troška sada izgleda ovako

$$\overline{C} = \begin{bmatrix} \underline{0} & 0 & 1 & 2 \\ 0 & 4 & 3 & 5 \\ 0 & 3 & 5 & 8 \\ 1 & 2 & \underline{0} & 0 \end{bmatrix}$$

U matrici smo podcrtali elemente koje je predprocesiranje moglo iskoristiti za stvaranje parcijalnog dodjeljivanja. Time smo dobili row $= (1, 0, 4, 0)$, odnosno $\varphi = (1, 0, 0, 3)$.

Ideja mađarske metode je sljedeća: krećemo s parcijalnim primal rješenjem (dodjeljivanje u kojem su samo neki vrhovi grafa dodijeljeni) i mogućim dual rješenjem (nizovima u i v koji zadovoljavaju (2.16)), svaka iteracija pokušava povećati broj dodijeljenih vrhova grafa koristeći samo bridove koji imaju reducirani trošak jednak nuli. Ako to uspije dobivamo novo parcijalno dodjeljivanje s jednim novim dodijeljenim vrhom. Ako ne uspije, ažuriramo dualno rješenje tako da dobijemo nove bridove s reduciranim troškom koji je jednak nuli.

Definirajmo nekoliko pojmova vezanih za parcijalna dodjeljivanja koja ćemo koristiti u mađarskoj metodi. *Alternirajući put* je put u grafu čiji su bridovi naizmjence dodijeljeni i nedodijeljeni. *Alternirajuće stablo* iz vrha k je stablo čiji svaki put koji kreće iz k je alternirajući. *Uvećavajući put* je alternirajući put kojemu početni i završni bridovi (a time i vrhovi) nisu dodijeljeni.

Mađarska metoda pokušava povećati broj dodijeljenih vrhova tako što traži uvećavajući put na parcijalnom bipartitnom grafu $G^0 = (U, V; E^0)$ pri čemu E^0 sadrži samo one bridove (i, j) za koje u tom trenutku vrijedi $\overline{c_{ij}} = 0$. Ako nađe takav put P onda metoda poveća broj dodijeljenih vrhova tako što zamijeni dodijeljene i nedodijeljene bridove u P. Preciznije, metoda postavlja $x_{ij} = 1$ za $\lfloor |P|/2 \rfloor + 1$ nedodijeljeni brid (i, j) iz P te $x_{ij} = 0$ za $\lfloor |P|/2 \rfloor$ dodijeljeni brid (i, j) iz P. Time smo očito povećali broj dodijeljenih vrhova za 1.

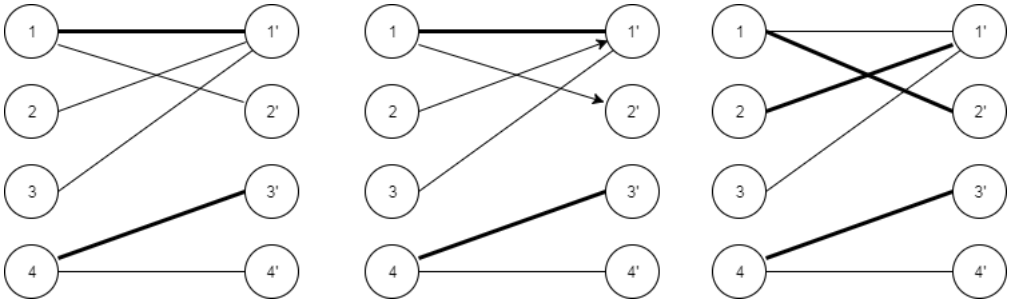
Sada navodimo algoritam koji traži uvećavajući put iz nedodijeljenog vrha $k \in U$ postepenim širenjem alternirajućeg stabla iz k. U svakoj iteraciji vrh je *označen* ako pripada putu iz k. Označeni vrh može biti *skeniran* ako je iskorišten za povećanje alternirajućeg stabla. U ovoj implementaciji označavanje i skeniranje vrhova iz U se podudaraju. U LV spremamo trenutno označene vrhove iz V, dok u SU i SV spremamo trenutno skenirane vrhove iz U odnosno V.

Algoritam 2 Alterniraj(k)

$SU = LV = SV = \emptyset$
$fail$ = false, $sink = 0$, $i = k$
while $fail ==$ false **and** $sink == 0$ **do**:
    $SU = SU \cup \{i\}$
    for all $j \in V \setminus LV : c_{ij} - u_i - v_j == 0$ **do**:
        $pred_j = i$
        $LV = LV \cup \{j\}$
    if $LV \setminus SV == \emptyset$ **then**:
        $fail$ = true
    else:
        j = neki vrh $\in LV \setminus SV$
        $SV = SV \cup \{j\}$
        if $row(j) == 0$ **then**:
            $sink = j$
        else:
            $i = row(j)$
return *sink*

Svaka iteracija se sastoji od dvije faze. Najprije se vrh $i \in U$ označi i skenira te se alternirajuće stablo poveća tako da svim vrhovima $j \in V$ za koje je $(i, j) \in E^0$ postavi da im je prethodnik upravo i. U drugoj fazi se označeni neskenirani vrh $j \in V$ skenira dodavajući jedinstveni brid $(row(j), j)$ uvećavajućem putu te $row(j)$ postaje kandidat od kojeg se kreće u sljedećoj iteraciji. Do prekida izvršavanja algoritma dolazi u dva slučaja: (i) ako odabrani vrh j iz druge faze nije već dodijeljen ($row(j) = 0$) - tada smo došli do uvećavajućeg puta od k do j; (ii) ako V ne sadrži ni jedan označen neskeniran vrh - tada trenutno stablo ne možemo proširiti.

Svaka iteracija glavne petlje algoritma zahtjeva $O(n)$ vremena, a u svakoj iteraciji se odabire novi vrh $i \in U$ koji se koristi u sljedećoj iteraciji. Slijedi da je ukupna složenost $O(n^2)$.



Slika 2.2: Primjer 2.5.

Primjer 2.5. *Nastavljamo sa primjerom 2.4. Imamo matricu reduciranog troška* $\overline{C}$ *i nizove u i v, te uzmimo da je* $k = 2$*. Algoritam Alterniraj(k) nam daje:*

$SU = LV = SV = \emptyset$, $fail = false$, $sink = 0$;

$i = 2 : SU = \{2\}, pred_1 = 2, LV = \{1\}$

$j = 1 : SV = \{1\}$;

$i = 1 : SU = \{2, 1\}, pred_2 = 1, LV = \{1, 2\}$

$j = 2 : SV = \{1, 2\}, sink = 2$;

Dobili smo uvećavajući put koji je prikazan na slici 2.2 *(u sredini). Novo rješenje prikazano na slici* 2.2 *(desno) dobiveno je zamjenom dodijeljenih i nedodijeljenih bridova uvećavajućeg puta.*

Napokon navodimo i samu mađarsku metodu. Kroz fazu predprocesiranja dolazi se do u, v, row i φ varijabli. Označimo još s $\overline{U}$ sve dodijeljene vrhove iz U. Na kraju mađarske metode dobivamo matricu X za koju vrijedi

$$x_{ij} = 1 \iff \varphi(i) = j$$

Promotrimo sada sami algoritam.

Algoritam 3 Mađarska metoda

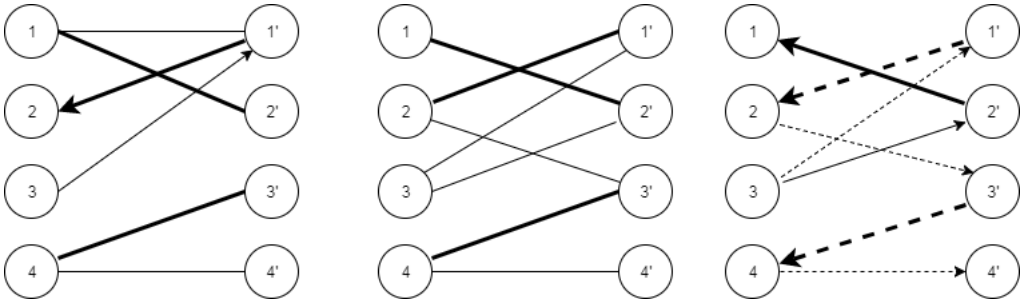
```
1:  while |U̅| < n do:
2:      k = neki vrh iz U \ U̅
3:      while k ∉ U̅ do:
4:          sink = Alterniraj(k)
5:          if sink > 0 then:
6:              U̅ = U̅ ∪ {k}
7:              j = sink
8:              repeat
9:                  i = pred_j
10:                 row(j) = i
11:                 temp = φ(i); φ(i) = j; j = temp;
12:             until i == k
13:         else:
14:             δ = min {c_ij − u_i − v_j : i ∈ SU, j ∈ V \ LV}
15:             for all i ∈ SU do:
16:                 u_i = u_i + δ
17:             for all j ∈ LV do:
18:                 v_j = v_j − δ
```

U svakoj iteraciji mađarske metode odabiremo jedan nedodijeljeni vrh iz U, pozivom metode *Alterniraj*(k) dobivamo uvećavajući put iz tog vrha kojeg u linijama $8-12$ raspletemo tako da zamijenimo dodijeljenje i nedodijeljene bridove tog puta. Time smo povećali kardinalnost našeg parcijalnog dodijeljivanja odnosno dodijelili taj odabrani vrh. Ako nismo u mogućnosti naći uvećavajući put, ažuriramo dualno rješenje (linije $13-18$) čime omogućavamo da u sljedećem koraku metode *Alterniraj*(k) dođemo do uvećavajućeg puta te nastavimo dalje s povećanjem broja dodijeljenih vrhova. Sljedeći rezultat pokazuje da će ovakvo ažuriranje dualnog rješenja stvarno omogućiti nalazak novog uvećavajućeg puta.

Propozicija 2.6. *Ažuriranje dualnog rješenja mađarske metode je takvo da svaki put barem jedan novi brid koji veže označeni vrh iz U s neoznačenim vrhom iz V uđe u E^0*

Dokaz. Ako je i označeni vrh iz U onda je $i \in SU$. Ako je j jedan neoznačeni vrh iz V onda $j \notin LV$. Tada po liniji 16 za te i i j imamo $\overline{c_{ij}} = \overline{c_{ij}} - \delta$. Po definiciji od δ iz linije 14 svi reducirani troškovi su nenegativni, a za barem jedan od njih je trošak jednak nuli čime on ulazi u skup E^0. □

Vanjska petlja mađarske metode se izvrši $O(n)$ puta, a procedura *Alterniraj*(k) i dualno ažuriranje također $O(n)$ puta. Budući da su obje te procedure složenosti $O(n^2)$ slijedi da je ukupna složenost mađarske metode $O(n^4)$.



Slika 2.3: Primjer 2.7.

Primjer 2.7. *Nastavljamo s primjerom* 2.5. *Kroz predprocesiranje smo došli do* $u = (7, 2, 1, 2)$, $v = (0, 2, 0, 0)$, $row = (1, 0, 4, 0)$, $\varphi = (1, 0, 0, 3)$ *te* $\overline{U} = \{1, 4\}$.

Najprije pozivamo proceduru Alternira_j(2) *pri čemu dobivamo* $sink = 2$ *te* $pred = (2, 1, -, -)$. *Povećavamo primalno rješenje na* $\overline{U} = \{1, 4, 2\}$, $row = (2, 1, 4, 0)$ *te* $\varphi = (2, 1, 0, 3)$.

Potom se poziva procedura Alternira_j(3) *koja ne uspijeva naći novi uvećavajući put. Ažuriramo dualno rješenje na* $\delta = 3$, $u = (7, 5, 4, 2)$, $v = (-3, 2, 0, 0)$ *te*

$$\overline{C} = \begin{bmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 5 \\ 4 & 2 & 0 & 0 \end{bmatrix}$$

Na slici 2.3 *(u sredini) se vidi novi bipartitni parcijalni graf* G^0 *u kojemu deblje linije označavaju trenutačno parcijalno dodijeljivanje. Ponovno pozivamo Alternira_j*(3) *koja nam ovog puta daje uvećavajući put (označen isprekidanim strelicama na slici* 2.3 *(desno)).*

Sada mađarska metoda uspijeva dodijeliti i posljednji vrh pa imamo $\overline{U} = \{1, 4, 2, 3\}$, $row = (3, 1, 2, 4)$, $\varphi = (2, 3, 1, 4)$. *Sada imamo matricu dodijeljivanja X koja predstavlja optimalno rješenje i za koju je* $x_{12} = x_{23} = x_{31} = x_{44} = 1$, *dok su na ostalim mjestima nule.*

Poglavlje 3

Problem naprtnjače

3.1 Opis problema

Sjetimo se filmskog scenarija iz uvoda: vi ste lopov koji je upravo ušetao u sef nekog poznatog muzeja. Sa sobom imate samo jednu naprtnjaču, a pred vama se nalazi mnoštvo vrijednih predmeta. Nažalost, ne možete ih sve ponijeti sa sobom jer vam naprtnjača ipak nije toliko velika. Morate izabrati predmete koji će stati u naprtnjaču, ali tako da njihova ukupna vrijednost bude što veća. Taj problem se naziva problem naprtnjače (eng. *Knapsack Problem*).

Problem naprtnjače jedan je od najpoznatijih problema iz područja kombinatorne optimizacije, a predmet je istraživanja već više od 120 godina ([21]). Sam problem, ili njegove varijacije, možemo naći u raznim područjima: u ekonomiji, kriptografiji, računarskoj znanosti, teoriji složenosti i primijenjenoj matematici.

Formalno definirajmo problem. Pretpostavimo da imamo naprtnjaču kapaciteta c, te n predmeta. Za svaki predmet označimo s p_i profit, a sa w_i težinu predmeta i, pri čemu je $i \in \{1, 2, \ldots, n\}$. Neka je vektor $x = (x_1, x_2, \ldots, x_n)$ definiran na sljedeći način

$$x_i = \begin{cases} 1 & \text{ako smo u naprtnjaču stavili predmet } i \\ 0 & \text{inače} \end{cases} \tag{3.1}$$

Tada naš problem naprtnjače postaje problem maksimizacije izraza

$$\sum_{i=1}^{n} p_i x_i \tag{3.2}$$

uz ograničenje

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{3.3}$$

U problemu pretpostavljamo da su c te p_i i w_i za sve $i \in \{1, 2, \ldots, n\}$ pozitivni cijeli brojevi. Također, vrijede sljedeće pretpostavke kojima ćemo se riješiti nekih rubnih slučajeva:

- $n \geq 2$ - inače imamo slučaj $n = 1$ kojeg trivijalno rješimo provjerom stane li predmet u naprtnjaču
- $w_i \leq c,\ i \in \{1, 2, \ldots, n\}$ - očito ni jedan predmet kojemu je težina veća od kapaciteta naprtnjače ne možemo staviti u naprtnjaču
- $\sum_{i=1}^{n} w_i > c$ - ako je zbroj težina svih predmeta manja od kapaciteta naprtnjače onda ćemo jednostavno sve predmete staviti u naprtnjaču

Za kraj ovog opisnog dijela dajemo još jednu primjenu problema naprtnjače. Naime, uveli smo problem naprtnjače kroz problem "pakiranja", ali možemo ga gledati i kroz problem "rezanja". Zamislimo da radimo u tvornici za preradu drva. Imamo veliki komad drveta duljine c koji moramo izrezati na manje komade. U tvornici već imamo predefiniran popis standardnih veličina w_i komada drveta koje možemo prodati za iznos p_i. Problem naprtnjače tada postaje problem rezanja tog velikog komada drveta na način koji će maksimizirati profit tvornice.

3.2 Analiza i složenost problema

Kao što ćemo vidjeti u potpoglavlju 3.4, postoje razne verzije problema naprtnjače. Verzija koju smo opisali i koju ćemo proučavati naziva se još i *0-1 problem naprtnjače* što označava da imamo točno jedan primjerak svakog predmeta te da možemo ili staviti predmet u naprtnjaču ili ga ne staviti, odnosno nema rezanja i stavljanja samo dijela nekog predmeta.

Za razliku od linearnog problema dodjeljivanja, problem naprtnjače nema polinomijalan algoritam za rješavanje, već je to problem iz klase *NP-teških* problema. Promotrimo najprije verziju problema naprtnjače prikazanog kroz problem odluke, a ne optimizacije odnosno traženja maksimalnog profita predmeta u naprtnjači.

Neka je zadan problem kao u potpoglavlju 3.1 te neka imamo i nenegativnu vrijednost k. Problem odluke tada glasi: možemo li odabrati predmete tako da im je ukupna težina manja od c te ukupna vrijednost barem k. Preciznije, želimo naći vektor $(x_1, x_2, \ldots, x_n)$ tako da $x_i \in \{0, 1\}$ i vrijedi

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{3.4}$$

i

$$\sum_{i=1}^{n} p_i x_i \geq k \tag{3.5}$$

Propozicija 3.1. *Problem odluke problema naprtnjače je NP-potpun problem.*

Dokaz. Pokažimo najprije da je problem u klasi *NP*. Jednostavno možemo vidjeti da, ako imamo neki vektor $(x_1, x_2, \ldots, x_n)$, u polinomijalnom vremenu možemo izračunati sume iz (3.4) i (3.5) i provjeriti vrijede li ti uvjeti.

Sad želimo pokazati da neki *NP-potpun* problem možemo svesti na naš problem odluke problema naprtnjače. Koristimo problem SUBSET-SUM za koji znamo da je *NP-potpun* ([16]). SUBSET-SUM problem glasi: neka su zadani nenegativni cijeli brojevi $s_1, s_2, \ldots, s_n, t$. Pitamo se postoji li podskup brojeva $s_1, s_2, \ldots, s_n$ čija je suma jednaka t.

Pretpostavimo sada da imamo neku instancu problema SUBSET-SUM. Tada možemo stvoriti instancu problema naprtnjače tako da postavimo $w_i = p_i = s_i$ za sve $i \in \{1, 2, \ldots, n\}$ te $c = k = t$. Sada vidimo da za tu instancu problema naprtnjače vrijede uvjeti (3.4) i (3.5) ako i samo ako u početnoj instanci SUBSET-SUM problema postoji podskup brojeva čija je suma t. Očito je svođenje problema SUBSET-SUM na problem naprtnjače polinomijalne složenosti, čime dokazujemo da je problem naprtnjače *NP-potpun* problem.

□

Propozicija 3.2. *Optimizacijska verzija problema naprtnjače je NP-težak problem.*

Dokaz. Pretpostavimo da imamo polinomijalni algoritam koji rješava optimizacijski problem naprtnjače. Tada bi svaki problem odluke problema naprtnjače mogli riješiti u polinomijalnom vremenu na način da izračunamo optimalno rješenje, odnosno nađemo maksimalni profit predmeta u naprtnjači, i za svaki k iz problema odluke jednostavno provjerimo je li on veći od profita dobivenog optimalnim rješenjem. Zbog činjenice da je problem odluke *NP-potpun*, slijedilo bi da je $P = NP$.

Međutim, optimizacijska verzija problema naprtnjače nije u klasi *NP* već po samoj definiciji klase *NP*, pa slijedi da je optimizacijska verzija *NP-težak* problem. □

3.3 Algoritmi za rješavanje

Iako smo pokazali da je problem naprtnjače *NP-težak*, poznati su efikasni algoritmi za njegovo rješavanje. Dvije najpoznatije metode za rješavanje su *dinamičko programiranje* i *branch-and-bound*. U pedesetima je Bellman ([4]) razvio prvu metodu za rješavanje

korištenjem dinamičkog programiranja. Krajem šezdesetih je Kolesar ([15]) prvi eksperimentirao sa korištenjem branch-and-bound metode, dok su Horowitz i Sahni ([14]) u sedamdesetima razvili poznati branch-and-bound algoritam. Kasnije su se razvile metode koje u rješavanju koriste dinamičko programiranje i branch-and-bound metodu.

Jedna jednostavna ideja pri rješavanju je da gledamo koji predmet donosi više vrijednosti po jedinici težine i prema tome odabiremo predmete koji više vrijede dok imamo mjesta u naprtnjači. Zbog toga nam je zgodno gledati sortirane i (po potrebi) renumerirane predmete takve da vrijedi

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \frac{p_2}{w_2} \tag{3.6}$$

Predmete jednostavno sortiramo u vremenu $O(n \log n)$. Sada navodimo takav pohlepni algoritam.

Algoritam 4 Pohlepni algoritam

```
1:  SortirajPredmete(n)
2:  c̄ = c
3:  total = 0
4:  for i = 1 to n do:
5:      x_i = 0
6:  for i = 1 to n do:
7:      if w_i ≤ c̄ then:
8:          x_i = 1
9:          c̄ = c̄ − w_i
10:         total = total + p_i
11:     else
12:         x_i = 0
13: return total
```

Algoritam nam vraća varijablu *total* u kojoj je spremljen profit predmeta koji su odabrani, dok je u vektoru $x = (x_1, x_2, \ldots, x_n)$ zapisano koje smo predmete odabrali. U sljedećem primjeru ćemo pokazati da ovaj algoritam nije dobar, odnosno da ne daje optimalno rješenje.

Primjer 3.3. *Neka je $k > 2$ neki prirodni broj. Uzmimo da je kapacitet naše naprtnjače $c = k$ te $n = 2$. Za predmete neka vrijedi $w_1 = p_1 = 1$ te $w_2 = k$ i $p_2 = k - 1$. Sada očito vrijedi $\frac{1}{1} > \frac{k-1}{k}$. Zbog toga bi pohlepni algoritam u prvom koraku uzeo prvi predmet, došao do drugog predmeta i vidio da on više ne stane u naprtnjaču te vratio kao rješenje $total = 1$. Očito je da to nije optimalno rješenje jer uz odabir drugog predmeta imamo $total = k - 1 > 1$*

Vidimo da odstupanje rješenja dobivenog pohlepnim algoritmom od optimalnog rješenja može biti vrlo veliko. Postoje rezultati koji poboljšavaju ovaj pohlepni algoritam i ograničavaju odstupanje, ali svejedno on ne daje uvijek optimalan rezultat.

Sada ćemo navesti algoritam dinamičkog programiranja koji nam daje optimalno rješenje problema naprtnjače. Neka su zadani cijeli brojevi m ($1 \leq m \leq n$) i $\overline{c}$ ($0 \leq \overline{c} \leq c$). Promatramo podinstancu problema naprtnjače s predmetima $1, 2, \ldots, m$ i kapacitetom naprtnjače $\overline{c}$, te s $f_m(\overline{c})$ označimo optimalno rješenje tog problema

$$f_m(\overline{c}) = \max\left\{ \sum_{j=1}^{m} p_j x_j : \sum_{j=1}^{m} w_j x_j \leq \overline{c}, x_j \in \{0,1\} \text{ za } j = 1, 2, \ldots, m \right\} \tag{3.7}$$

Algoritam koji promatramo temelji se na rekurziji: najprije dajemo njen bazni slučaj s jednim predmetom

$$f_1(\overline{c}) = \begin{cases} 0 & \text{za } \overline{c} = 0, 1, \ldots, w_1 - 1 \\ p_1 & \text{za } \overline{c} = w_1, \ldots, c \end{cases}$$

U sljedećim koracima, za $1 < m \leq n$, promatramo f_{m-1}: ako naš predmet stane u dosad napunjenu naprtnjaču, odlučujemo isplati li se dodati ga, odnosno je li optimalno rješenje "bolje" s njim ili bez njega. Preciznije, imamo rekurziju:

$$f_m(\overline{c}) = \begin{cases} f_{m-1}(\overline{c}) & \text{za } \overline{c} = 0, 1, \ldots, w_m - 1 \\ \max\left(f_{m-1}(\overline{c}) \, , \, f_{m-1}(\overline{c} - w_m) + p_m \right) & \text{za } \overline{c} = w_1, \ldots, c \end{cases} \tag{3.8}$$

Optimalno rješenje problema zapisano je u $f_n(c)$. Iz rekurzije lako dobijemo algoritam koji optimalna rješenja podinstanci sprema u matricu dp kako ih ne bi morao više puta izračunavati.

Algoritam 5 Algoritam dinamičkog programiranja

```
1:  for i = 1 to n do:
2:      for j = 1 to c do:
3:          dp[i, j] = 0
4:  for j = 1 to c do:
5:      dp[0, j] = 0
6:  for i = 1 to n do:
7:      for j = 1 to c do:
8:          if w_i > j then:
9:              dp[i, j] = dp[i − 1, j]
10:         else:
11:             dp[i, j] = max(dp[i − 1, j] , dp[i − 1, j − w_i] + p_i)
12: return dp[n, c]
```

Lako se vidi da su vremenska i prostorna složenost ovog algoritma $O(nc)$. Na prvi pogled se čini da smo našli polinomijalan algoritam koji rješava problem iako smo ranije rekli da je problem naprtnjače *NP-težak*. Međutim, ovaj algoritam nije polinomijalan već *pseudo-polinomijalan.* To znači da je on polinomijalan u vrijednosti ulaznih podataka ali ne i duljini ulaznih podataka. Preciznije, kod problema naprtnjače vrijedi da je algoritam ovisan o duljini ulaznog podatka c, odnosno ako označimo sa $b = \log c$ duljinu od c, vrijedi da je vremenska složenost ovog algoritma zapravo $O(n \cdot 2^b)$ pa ipak stoji tvrdnja da je problem *NP-težak*.

3.4 Druge verzije problema

Osim *0-1 problema naprtnjače* postoje i druge verzije ovog problema koje su također dosta istraživane. U ovom ćemo potpoglavlju navesti neke od njih.

Ograničeni problem naprtnjače (eng. *Bounded Knapsack Problem*) je problem u kojem svakog predmeta može biti više, ali ipak ograničeni broj. Možemo zamisliti situaciju da je naš lopov iz uvoda u nekoj trgovini i da od svakog predmeta ima više komada na raspolaganju. Preciznije, uz već definirane p_i, w_i i c, imamo i niz pozitivnih cijelih brojeva $b_1, b_2, \ldots, b_n$, a sam problem glasi

$$\max \sum_{i=1}^{n} p_i x_i \tag{3.9}$$

uz ograničenja

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{3.10}$$

$$0 \leq x_i \leq b_i, \quad x_i \in \mathbb{Z}, \quad (i = 1, 2, \ldots, n) \tag{3.11}$$

Ograničeni problem naprtnjače je također *NP-težak*. Ako svaki primjerak predmeta zapišemo kao poseban predmet onda problem možemo riješiti koristeći algoritam dinamičkog programiranja iz prošlog potpoglavlja.

Neograničeni problem naprtnjače (eng. *Unbounded Knapsack Problem*) je problem sličan ograničenom problemu naprtnjače, samo što sada na raspolaganju imamo beskonačno mnogo komada svakog predmeta. Problem glasi

$$\max \sum_{i=1}^{n} p_i x_i \tag{3.12}$$

uz ograničenja

$$\sum_{i=1}^{n} w_i x_i \leq c \tag{3.13}$$

$$x_i \geq 0, \quad x_i \in \mathbb{Z}, \quad (i = 1, 2, \ldots, n) \tag{3.14}$$

Neograničeni problem naprtnjače je također *NP-težak* a može se riješiti algoritmom dinamičkog programiranja sličnim onom iz prošlog potpoglavlja.

Problem naprtnjače višestrukog izbora (eng. *Multiple-Choice Knapsack Problem*) je problem ruksaka u kojem su predmeti podjeljeni u više skupina te moramo odabrati točno jedan predmet iz svake skupine. Neka su $N_1, N_2, \ldots, N_m$ disjunktni skupovi predmeta te uvedimo matricu $X = (x_{ij})$ pri čemu $x_{ij} = 1$ označava da smo odabrali j-ti predmet iz N_i-tog skupa predmeta. Sada problem glasi

$$\max \sum_{i=1}^{m} \sum_{j \in N_i} p_{ij} x_{ij} \tag{3.15}$$

uz ograničenja

$$\sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq c \tag{3.16}$$

$$\sum_{j \in N_i} x_{ij} = 1 \quad (i = 1, 2, \ldots, m) \tag{3.17}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, 2, \ldots, m, \quad j \in N_i) \tag{3.18}$$

Ovaj problem je također *NP-težak*.

Problem višestrukih naprtnjača je problem sličan *0-1 problemu naprtnjače* samo što sada imamo m ($m \leq n$) naprtnjača. Sve su oznake iste, samo sa $c_1, c_2, \ldots, c_m$ označavamo kapacitete naprtnjača. Također imamo matricu $X = (x_{ij})$ za koju vrijedi $x_{ij} = 1$ ako i samo ako smo predmet j stavili u i-tu naprtnjaču. Problem se definira na sljedeći način

$$\max \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{3.19}$$

uz ograničenja

$$\sum_{j=1}^{n} w_j x_{ij} \leq c_i \quad (i = 1, 2, \ldots, m) \tag{3.20}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \quad (j = 1, 2, \ldots, n) \tag{3.21}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n) \tag{3.22}$$

Problem je također *NP-težak*.

Poglavlje 4

Poopćeni problem dodjeljivanja

4.1 Opis problema

Na kraju prošlog poglavlja naveli smo nekoliko različitih verzija problema naprtnjače, no ono što će nas u ovom poglavlju zanimati je poopćenje tog problema koje se naziva *poopćeni problem dodjeljivanja* (eng. *Generalized Assignment Problem*). Problem je sljedeći: zadano je n predmeta i m naprtnjača, a zadatak je za svaki predmet odabrati naprtnjaču u koju ćemo ga staviti tako da ukupan profit bude maksimalan, a da ne prekoračimo kapacitet ni jedne naprtnjače. Preciznije, ako sa c_i označimo kapacitet pojedine naprtnjače, sa p_{ij} profit predmeta j ako smo ga stavili u naprtnjaču i, te s w_{ij} težinu predmeta j ako smo ga stavili u naprtnjaču i, onda problem glasi

$$\max z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{4.1}$$

uz ograničenja

$$\sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i \quad (i = 1, 2, \ldots, m) \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad (j = 1, 2, \ldots, n) \tag{4.3}$$

$$x_{ij} \in \{0, 1\} \quad (i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n) \tag{4.4}$$

Kao i prije, $x_{ij} = 1$ označava da smo predmet j stavili u naprtnjaču i. Primijetimo da uvjet (4.3) zahtjeva da svi predmeti budu raspoređeni. Kao i dosad pretpostavljamo da su svi p_{ij}, w_{ij} i c_i nenegativni cijeli brojevi. To možemo pretpostaviti bez smanjenja općenitosti jer se ostali slučajevi jednostavnim transformacijama mogu svesti na nenegativne cijele brojeve.

Poopćeni problem dodjeljivanja se često javlja u praksi - kao samostalni problem ili kao potproblem u područjima raspoređivanja resursa, preusmjeravanja vozila i proizvodnih sustava.

Za kraj ovog potpoglavlja recimo nešto o vezi poopćenog problema dodjeljivanja i problema koje smo obrađivali u prethodnim poglavljima. Lako se vidi da ako postavimo $m = 1$, odnosno ako imamo samo jednu naprtnjaču te uvjet (4.3) zamjenimo sa uvjetom da je ta suma manja ili jednaka 1, dobivamo upravo problem naprtnjače. S druge strane, ako postavimo $c_i = w_{ij} = 1$ za sve $i = 1, \ldots, m$ te $j = 1, \ldots, n$ dobit ćemo linearni problem dodjeljivanja. Možda se čini da imamo mali problem jer smo na ovaj način dobili maksimizacijski linearni problem dodjeljivanja, dok smo u drugom poglavlju predstavili minimizacijski problem, ali to zapravo ne stvara nikakav problem. Naime, maksimizacijski linearni problem dodjeljivanja možemo transformirati u minimizacijski tako da svaki iznos profita pomnožimo s -1 i dodamo mu najveći profit p_{max}, tj. stavimo $p'_{ij} = -p_{ij} + p_{max}$. Tada će optimalno rješenje minimizacijskog linearnog problema dodjeljivanja s matricom troška $P' = (p'_{ij})$ odgovarati optimalnom rješenju maksimizacijskog problema s matricom troška $P = (p_{ij})$. Ukupna maksimizacijska suma problema tada će biti jednaka $total_{max} = -total_{min} + n \cdot p_{max}$.

4.2 Analiza problema

Jedna stvar koju smo istaknuli kod opisa problema je uvjet (4.3), odnosno uvjet da svaki predmet mora biti stavljen u neku naprtnjaču. Zbog toga vrijedi da instanca poopćenog problema dodjeljivanja uopće ne mora imati valjano rješenje. Štoviše, pitanje postojanja valjanog rješenja poopćenog problema dodjeljivanja je *NP-potpun* problem što pokazuje sljedeća propozicija.

Propozicija 4.1. *Problem postojanja valjanog rješenja poopćenog problema dodjeljivanja je NP-potpun problem*

Dokaz. Ako imamo neku instancu problema lako vidimo da je matrica $X = (x_{ij})$ jedan certifikat za taj problem. Očito se u polinomnom vremenu mogu ispitati ograničenja (4.2)–(4.4) pa po teoremu o certifikatu slijedi da je taj problem u klasi *NP*.

Sada trebamo pokazati da je neki *NP-potpun* problem moguće svesti na naš problem postojanja valjanog rješenja poopćenog problema dodjeljivanja. Za to ćemo iskoristiti problem PARTICIJE. Problem PARTICIJE glasi: imamo multiskup $W = \{w_1, \ldots, w_n\}$ od n elemenata. Pitanje je postoji li podskup $T \subseteq W$ takav da vrijedi

$$\sum_{w \in T} w = \sum_{w \in W \setminus T} w$$

Problem PARTICIJE je *NP-potpun* (vidi [16]). Pretpostavimo sada da imamo instancu problema PARTICIJE, to jest neki multiskup $W = \{w_1, \ldots, w_n\}$. Iz njega možemo konstruirati instancu poopćenog problema dodjeljivanja na sljedeći način: neka je $m = 2$, $w_{1j} = w_{2j} = w_j$ za $j = 1, \ldots, n$ i neka je $c_1 = c_2 = \frac{1}{2}\sum_{j=1}^{n} w_j$. Sada vidimo da za ovu instancu problema postoji valjano rješenje ako i samo ako za danu instancu problema PARTICIJE postoji rješenje, odnosno skup T. Iz toga slijedi da je problem postojanja valjanog rješenja poopćenog problema dodjeljivanja *NP-potpun* problem. □

Zbog činjenice da je već problem postojanja valjanog rješenja *NP-potpun* te zbog definicije klase *NP* slijedi da je poopćeni problem dodjeljivanja *NP-težak* problem.

Sada ćemo navesti relaksaciju uvjeta problema koja će nam omogućiti da jednostavno izračunamo gornju među na rezultat. Najprije uvedimo oznake $N = \{1, \ldots, n\}$ i $M = \{1, \ldots, m\}$. Uvjet koji ćemo relaksirati je (4.2) i to na način da ga zamjenimo s

$$w_{ij}x_{ij} \leq c_i \quad i \in M, j \in N \tag{4.5}$$

Sada se optimalno rješenje $\overline{X} = (\overline{x_{ij}})$ relaksiranog problema dobiva određivanjem

$$i(j) = \arg\max\{p_{ij} : i \in M, w_{ij} \leq c_i\}$$

za svaki $j \in N$, te postavljanjem $\overline{x_{i(j)j}} = 1$ i $\overline{x_{ij}} = 0$ za sve $i \in M \setminus \{i(j)\}$. Ovo zapravo znači da ćemo za svaki predmet gledati samo u koju naprtnjaču ga je najbolje staviti, uz uvjet da stane u tu naprtnjaču.

Sada imamo prvu gornju među

$$U_0 = \sum_{j=1}^{n} p_{i(j)j} \tag{4.6}$$

koju možemo dodatno poboljšati. Neka je

$$N_i = \{j \in N : \overline{x_{ij}} = 1\} \quad i \in M$$
$$d_i = \sum_{j \in N} w_{ij} - c_i \quad i \in M$$
$$M' = \{i \in M : d_i > 0\}$$
$$N' = \bigcup_{i \in M'} N_i$$

N_i za naprtnjaču i označava skup predmeta koji se nalaze u njoj, d_i je razlika između težina predmeta koji su u naprtnjači i te njezinog kapaciteta, M' je skup naprtnjača koje su prešle svoj kapacitet, a N' je skup predmeta koji se nalaze u naprtnjačama koje su prešle svoj kapacitet. Sada izraz

$$q_j = p_{i(j)j} - max_2\{p_{ij} : i \in M, w_{ij} \leq c_i\}, \quad j \in N'$$

gdje s max_2 označavamo drugi maksimum u tom skupu, daje najmanju razliku u profitu koja će nastati ako predmet j koji je trenutačno u naprtnjači iz M' prebacimo negdje drugdje. Sada vidimo da je za svaki $i \in M'$ donja međa na gubitak profita dana rješavanjem minimizacijskog $0-1$ problema naprtnjače, u oznaci KP_i, danog na sljedeći način

$$\min v_i = \sum_{j \in N_i} q_j y_{ij}$$

uz ograničenja

$$\sum_{j \in N_i} w_{ij} y_{ij} \geq d_i$$

$$y_{ij} \in \{0, 1\}, \quad j \in N_i$$

gdje s $y_{ij} = 1$ označavamo da smo predmet j izvadili iz naprtnjače i.
Sada dobivamo poboljšanu gornju među iz (4.6)

$$U_1 = U_0 - \sum_{i \in M'} v_i \tag{4.7}$$

koju su dali Ross i Soland ([22]). Martello i Toth su 1981. godine, Fisher, Jaikumar i Van Wassenhove 1986., te Jörnsten i Näsberg također 1986. godine, koristeći neke druge relaksacije problema, došli do novih gornjih međa (za sve vidi [20]).

4.3 Algoritmi za rješavanje

U prošlom potpoglavlju smo rekli da je poopćeni problem dodjeljivanja *NP-težak* pa stoga znamo da ne postoji polinomijalan algoritam koji daje optimalno rješenje. Ipak, u ovom potpoglavlju ćemo ukratko opisati jedan nepolinomijalni algoritam koji daje optimalno rješenje, te navesti dva algoritma koji daju jedno moguće, ali ne nužno optimalno rješenje.

Najčešća metoda za egzaktno rješavanje poopćenog problema dodjeljivanja je *branch-and-bound* metoda s pretraživanjem u dubinu (eng. *depth-first branch-and-bound*). Godine 1975. Ross i Soland ([22]) dali su jedan takav algoritam koji koristi gornju među (4.7). Algoritam računa gornju među U_1 na svakom čvoru stabla pretraživanja i informacije dobivene računanjem koristi za odabir varijable po kojoj će granati stablo. Preciznije, varijabla za grananje x_{i*j*} bit će odabrana među onima koji imaju $y_{ij} = 0$ ($i \in M', j \in N'$) te za koju je izraz

$$\frac{q_j}{w_{ij}/(c_i - \sum_{k=1}^{n} w_{ik}x_{ik})}$$

maksimalan. Tako dolazimo do predmeta j^* kojeg je *dobro* staviti u naprtnjaču i^*, s obzirom na razliku u profitu kod premještanja predmeta i preostali prostor u naprtnjači. Grananje se tada postiže generiranjem dva podstabla s postavljenim $x_{i^*j^*} = 1$ te $x_{i^*j^*} = 0$.

Sada navodimo dva aproksimacijska algoritma koji u polinomijalnom vremenu daju jedno moguće rješenje poopćenog problema dodjeljivanja. Prvo ćemo navesti algoritam *MTHG* (*Martello Toth Heuristic for Generalized Assignment Problem*) kojeg su dali Martello i Toth 1981. godine ([19]), a potom jedno njegovo poboljšanje pod nazivom *MTRG* (*Martello Toth Reduction of Generalized Assignment Problem*).

S f_{ij} označimo *mjeru poželjnosti* stavljanja predmeta j u naprtnjaču i. Algoritam *MTHG* ne određuje strogo što bi bila ta mjera poželjnosti, ali ćemo kasnije navesti neke mogućnosti za f_{ij}. Iterativno prolazimo kroz sve predmete koje još nismo stavili u neku naprtnjaču te tražimo predmet j koji ima najveću razliku između najvećeg i drugog najvećeg elementa f_{ij} ($i \in M$). Tada taj predmet j stavimo upravo u pripadnu naprtnjaču i za koju je vrijednost f_{ij} maksimalna. U drugom dijelu algoritma se trenutačno rješenje popravlja kroz niz koraka u kojima za svaki predmet provjeravamo postoji li neka naprtnjača u koju bi bilo bolje staviti taj predmet. Ako na kraju izvršavanja algoritma vrijedi $feas = False$, nismo došli do valjanog rješenja. Inače je vrijednost rješenja zapisana u z, dok je raspored predmeta po naprtnjačama zapisan u nizu (y_j), pri čemu $y_j = i$ znači da smo predmet j stavili u naprtnjaču i. Sada navodimo algoritam.

Algoritam 6 MTHG

$M = \{1, \ldots, m\}$
$N = \{1, \ldots, n\}$
$feas = True$
for $i = 1$ **do to** m:
    $\overline{c_i} = c_i$
$z = 0$
while $N \neq \emptyset$ **and** $feas == True$ **do**:
    $d^* = -\infty$
    for all $j \in N$ **do**:
        $F_j = \{i \in M : w_{ij} \leq \overline{c_i}\}$
        if $F_j == \emptyset$ **then**:
            $feas = False$
        else:
            $i' = \arg\max\{f_{ij} : i \in F_j\}$
            if $F_j \setminus \{i'\} == \emptyset$ **then**:
                $d = \infty$
            else:
                $d = f_{i'j} - max_2\{f_{ij} : i \in F_j\}$
            if $d > d^*$ **then**:
                $d^* = d$
                $i^* = i'$
                $j^* = j$
    if $feas == True$ **then**:
        $y_{j^*} = i^*$
        $z = z + p_{i^*j^*}$
        $\overline{c_{i^*}} = \overline{c_{i^*}} - w_{i^*j^*}$
        $N = N \setminus \{j^*\}$
if $feas == False$ **then**:
    return $False$
else:
    for $j = 1$ **to** n **do**:
        $i' = y_j$
        $A = \{p_{ij} : i \in M \setminus \{i'\},\ w_{ij} < \overline{c_i}\}$
        if $A \neq \emptyset$ **then**:
            $p_{i''j} = \max A$
            if $p_{i''j} > p_{i'j}$ **then**:
                $y_j = i''$
                $z = z - p_{i'j} + p_{i''j}$
                $\overline{c_{i'}} = \overline{c_{i'}} + w_{i'j}$
                $\overline{c_{i''}} = \overline{c_{i''}} - w_{i''j}$
return z

Algoritam se može implementirati tako da se prije početka za svaki predmet j vrijednosti f_{ij} ($i \in M$), za koje vrijedi $w_{ij} \leq \overline{c_i} = c_i$, sortiraju silazno tako da nam u svakoj iteraciji maksimum i drugi maksimum od f_{ij} budu odmah dostupni. Za to nam je potrebno $O(nm \log m)$ vremena. Središnja *while* petlja izvršava $O(n)$ koraka, a svaki korak također zahtjeva $O(n)$ vremena pa je njezino ukupno vrijeme izvršavanja $O(n^2)$. Budući da se stavljanjem predmeta u naprtnjaču iznos $\overline{c_i}$ smanjuje, moguće je da prethodno sortirani niz elemenata f_{ij} više neće biti valjan. No s obzirom na to da se maksimumi mogu samo smanjiti slijedi da je algoritmu ukupno potrebno $O(n^2)$ vremena za provjeru i ažuriranje maksimuma. Lako se vidi da je složenost drugog dijela algoritma $O(nm)$, pa je ukupna složenost ovog algoritma $O(nm \log m + n^2)$.

Martello i Toth su eksperimentalnim rezultatima pokazali da se dobri rezultati mogu dobiti ako se za mjeru poželjnosti f_{ij} odabere nešto od sljedećeg:

(a) $f_{ij} = p_{ij}$ (sa ovim odabirom se može drugi dio algoritma preskočiti)

(b) $f_{ij} = p_{ij}/w_{ij}$

(c) $f_{ij} = -w_{ij}$

(d) $f_{ij} = -w_{ij}/c_i$

Sljedeći algoritam kojeg navodimo, pod nazivom *MTRG*, omogućit će nam da za neka valjana rješenja problema saznamo da su ona ujedno i optimalna. Algoritam prima jedno valjano rješenje (y_j) s vrijednošću z, gornju među U_0 koju smo naveli kod analize problema te pripadne vrijednosti $i(j) = \arg\max\{p_{ij} : i \in M,\ w_{ij} \leq c_i\}$. To valjano rješenje možemo dobiti korištenjem algoritma $MTHG$ kojeg smo naveli ili bilo kojeg drugog koji daje neko valjano rješenje. Počinjemo s matricom $X = (x_{ij})$ kojoj su svi elementi neki broj različit od 0 i 1. Algoritam radi tako da fiksira na 0 sve vrijednosti x_{ij} koje bi, ako bi ih postavio na 1 u izračunavanju U_0, smanjile vrijednost gornje međe na neku vrijednost ne veću od z, odnosno ako bi vrijedilo $z \geq U_0 - p_{i(j)j} + p_{ij}$ (naravno prepostavljamo da krećemo s $z < U_0$). Ako za neki j vrijedi da su svi x_{ij} osim jednog, recimo x_{i*j}, fiksirani na 0, onda fiksiramo $x_{i*j} = 1$. Ako za neki j vrijedi da su svi $x_{ij} = 0$, onda je valjano rješenje (y_j) optimalno. Na početku pretpostavljamo da su $\overline{c_i} = c_i$ za sve $i \in M$.

Algoritam 7 MTRG

```
1: opt = False
2: j = 0
3: while j < n and opt == False do:
4:     j = j + 1
5:     k_j = 0
6:     for i = 1 to m do:
7:         if z ≥ U_0 − p_{i(j)j} + p_{ij} or w_{ij} > c̄_i then:
8:             x_{ij} = 0
9:             k_j = k_j + 1
10:        else:
11:            i* = i
12:    if k_j == m − 1 then:
13:        x_{i*j} = 1
14:        c̄_{i*} = c̄_{i*} − w_{i*j}
15:    else:
16:        if k_j == m then:
17:            opt = True
18: return opt
```

Lako se vidi da je vremenska složenost ovog algoritma $O(nm)$. Kada algoritam fiksira neki element na 1 i time smanji vrijednost nekog kapaciteta $\overline{c_i}$, možemo pokušati reducirati problem ponovnim pokretanjem algoritma. Budući da se maksimalno n elemenata može postaviti na 1, slijedi da bi ukupna složenost s ponovnim pokretanjem bila $O(n^2m)$.

Poglavlje 5

Implementacije i rezultati

U ovom poglavlju ćemo reći nešto više o samoj implementaciji nekih algoritama koje smo obrađivali u ovom radu te navesti neke eksperimentalne rezultate. Svi programi su pisani u programskom jeziku *Python 2.7*, a testirani su na osobnom računalu sa *Linux Mint* operacijskim sustavom. Računalo je opremljeno dvojezgrenim *Intel(R) Core(TM) i3-550 3.20GHz* procesorom sa 8 GB radne memorije.

Navedimo prvo implementaciju algoritma dinamičkog programiranja za rješavanje problema naprtnjače. Oznake su jednake onima iz poglavlja 3. Sam ulaz izgleda ovako: u prvom redu se nalaze kapacitet naprtnjače i broj predmeta odvojeni razmakom. U svakom od sljedećih n redova nalaze se dvije brojke odvojene razmakom koje označavaju težinu i vrijednost pojedinog predmeta. Na kraju izvršavanja ispiše se ukupan ostvareni profit.

```
first_line = raw_input().split()

c = int(first_line[0])
n = int(first_line[1])

dp = [[0 for j in range(c+1)] for i in range(n+1)]
w = [0 for i in range(n+1)]
p = [0 for i in range(n+1)]

for i in range(1,n+1):
  temp = raw_input().split()
  w[i] = int(temp[0])
  p[i] = int(temp[1])

for i in range(1,n+1):
```

```
  for j in range(1,c+1):
    if w[i] > j:
      dp[i][j] = dp[i-1][j]
    else:
      dp[i][j] = max(dp[i-1][j], dp[i-1][j-w[i]] + p[i])

print dp[n][c]
```

Implementacija je testirana na četiri grupe test podatka ([8]). Svaka grupa se sastojala od 5 različitih primjera te je na svakom primjeru mjereno vrijeme izvršavanja. Prosjeci vremena izvršavanja mogu se vidjeti u sljedećoj tablici.

Kapacitet naprtnjače	**Broj predmeta**	**Prosječno vrijeme izvršavanja** [s]
165	10	0.00202
750	10	0.01014
1280836	5	3.07106
6404180	24	90.73978

Tablica 5.1: Problem naprtnjače

Možemo vidjeti da vremena izvršavanja prate vremensku složenost $O(n \cdot 2^{\log c})$ navedenu u potpoglavlju 3.3.

Sada navodimo implementaciju algoritma *MTHG* za rješavanje poopćenog problema dodjeljivanja. Oznake su slične onima iz poglavlja 5. Jedine promjene su elementi $\overline{c}$, i', i'' , $i*$, $j*$, $d*$ koje smo u implementaciji redom nazvali *c_dash*, *i_apos*, *i_double_apos*, *i_star*, *j_star*, *d_star*. Funkcije *initialize_matrix* i *initialize_fij* su pomoćne funkcije za inicijalizaciju matrica. Funkcija *initialize_fij* prima i argument koji određuje na koji će se način određivati vrijednosti f_{ij} iz algoritma. Funkcije *arg_max* i *max_2* su pomoćne funkcije za traženje argumenta maksimuma i drugog maksimuma. Ulaz algoritma izgleda ovako: u prvom redu se unosi broj naprtnjača i broj predmeta odvojeni razmakom. U svakom od sljedećih m redaka se unosi kapacitet svake naprtnjače. Nakon toga se unosi matrica w koja sadrži težine predmeta te se nakon nje unosi matrica p koja sadrži profit svakog predmeta. Program na kraju ispiše ukupan profit koji smo ostvarili ili *'Nismo nasli rjesenje'* ako nije uspio naći valjano rješenje.

```
def initialize_matrix(m,n):
  matrix = [0]
  for i in range(1,m+1):
    matrix.append([0])
    for j in range(1,n+1):
      matrix[i].append(0)
  return matrix

def initialize_fij(m,n,option='d'):
  global p, w, c
  matrix = [0]
  for i in range(1,m+1):
    matrix.append([0])
    for j in range(1,n+1):
      if option == 'a':
        matrix[i].append(p[i][j])
      elif option == 'b':
        matrix[i].append(p[i][j] / float(w[i][j]))
      elif option == 'c':
        matrix[i].append(-w[i][j])
      else:
        matrix[i].append(-w[i][j] / float(c[i]))
  return matrix

def arg_max(f,j,F_j):
  arg = F_j[0]
  maks = f[arg][j]
  for i in F_j:
    if f[i][j] > maks:
      maks = f[i][j]
      arg = i
  return arg

def max_2(f,j,F_j):
  temp = []
  for i in F_j:
    temp.append(f[i][j])
  temp.sort(reverse=True)
```

```
  return temp[1]

first_line = raw_input().split()

m = int(first_line[0])
n = int(first_line[1])

c = [0 for i in range(m+1)]
for i in range(1,m+1):
  c[i] = int(raw_input())

w = initialize_matrix(m,n)
for i in range(1,m+1):
  temp = raw_input().split()
  for j in range(1,n+1):
    w[i][j] = int(temp[j-1])

p = initialize_matrix(m,n)
for i in range(1,m+1):
  temp = raw_input().split()
  for j in range(1,n+1):
    p[i][j] = int(temp[j-1])

M = set(range(1,m+1))
N = set(range(1,n+1))
feas = True
y = [0 for i in range(n+1)]
c_dash = [c[i] for i in range(m+1)]
z = 0
f = initialize_fij(m,n)

while len(N) != 0 and feas == True:
  d_star = -float('inf')

  for j in list(N):
    F_j = []
    for i in list(M):
      if w[i][j] <= c_dash[i]:
        F_j.append(i)
```

```
    if len(F_j) == 0:
      feas = False
    else:
      i_apos = arg_max(f,j,F_j)
      if len(F_j) == 1:
        d = float('inf')
      else:
        d = f[i_apos][j] - max_2(f,j,F_j)
      if d > d_star:
        d_star = d
        i_star = i_apos
        j_star = j

  if feas == True:
    y[j_star] = i_star
    z = z + p[i_star][j_star]
    c_dash[i_star] = c_dash[i_star] - w[i_star][j_star]
    N.remove(j_star)

if feas == False:
  print "Nismo nasli rjesenje"
else:
  for j in range(1,n+1):
    i_apos = y[j]

    temp_M = list(M - {i_apos})
    A_ind = []
    for i in temp_M:
      if w[i][j] < c_dash[i]:
        A_ind.append(i)

    if len(A_ind) != 0:
      i_double_apos = arg_max(p,j,A_ind)
      if p[i_double_apos][j] > p[i_apos][j]:
        y[j] = i_double_apos
        z = z - p[i_apos][j]+p[i_double_apos][j]
        c_dash[i_apos] = c_dash[i_apos]+w[i_apos][j]
        c_dash[i_double_apos] = c_dash[i_double_apos]-
```

```
                                                    w[i_double_apos][j]
print z
```

Sada navodimo tablicu sa eksperimentalnim rezultatima ovog algoritma. Algoritam je testiran na tri grupe podataka sa po pet primjera (test primjeri preuzeti sa [2]). Za svaku grupu navodimo prosječno vrijeme potrebno za izvršavanje i za koliko je posto nađeno rješenje lošije od optimalnog.

Broj naprtnjača	**Broj predmeta**	**Prosječno vrijeme izvršavanja** [s]	**Prosječno lošiji od optimalnog**
5	20	0.00187	5.4%
8	40	0.00516	4.2%
10	60	0.01613	3%

Tablica 5.2: MTHG - Poopćeni problem dodjeljivanja

Također i ovdje vidimo da vremena izvršavanja prate teorijsku složenost iskazanu u potpoglavlju 4.3.

Za kraj ovog poglavlja napravit ćemo usporedbu između implementiranog algoritma za poopćeni problem dodjeljivanja i mađarske metode. Naime, već smo u potpoglavlju 4.1 naveli da je poopćeni problem dodjeljivanja generalizacija linearnog problema dodjeljivanja pa iz toga slijedi da bi sa *MTHG* algoritmom trebali moći riješiti i instance linearnog problema dodjeljivanja. Kao implementaciju mađarske metode koristit ćemo *Python* modul *Munkres* ([9]). U sljedećoj tablici možemo vidjeti usporedbu rezultata mađarske metode i *MTHG* algoritma. U linearnom problemu dodjeljivanja je broj radnika i poslova jednak te on odgovara broju naprtnjača i predmeta u *MTHG* algoritmu i taj je broj naveden u prvom stupcu tablice. Dva algoritma su testirana na pet primjera različite veličine (test primjeri preuzeti sa [3]). Za svaki test primjer navedeno je trajanje oba algoritma i koliko je posto rješenje nađeno *MTHG* algoritmom lošije od onog optimalnog, nađenog mađarskom metodom.

Broj radnika (poslova)	**MTHG prosječno lošiji od optimalnog**	**MTHG - vrijeme izvršavanja** [s]	**Munkres - vrijeme izvršavanja** [s]
100	0.73%	0.42187	0.47195
200	0.47%	3.20306	2.23677
300	0.23%	10.94475	7.51635
400	0.15%	26.55649	12.58284
700	0.15%	140.94444	41.9242

Tablica 5.3: MTHG vs. mađarska metoda

Vidimo da *MTHG* algoritam daje odlične rezultate te da se oni poboljšavaju kako problem raste. Ipak, vidimo i da je vrijeme izvršavanja *MTHG* algoritma znatno duže pa ga se ne isplati koristiti za rješavanje linearnog problema dodjeljivanja kad već imamo efikasan algoritam koji daje optimalno rješenje.

Napomena. Posljednje testiranje je izvršeno na računalu koje je opremljeno dvojezgrenim Intel(R) Core(TM) i5-3317U 1.70GHz procesorom sa 4 GB radne memorije i na operacijskom sustavu Windows(R) 7.

Bibliografija

[1] M. Akgül, *A genuinely polynomial primal simplex algorithm for the assignment*, Discr. Appl. Math. **45** (1993), 93–115.

[2] J.E. Beasley, *Generalised Assignment Problem data*, OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html`.

[3] J.E. Beasley, *Linear Assignment Problem data*, OR-Library, `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/assigninfo.html`.

[4] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[5] G. Birkhoff, *Tres observaciones sobre el algebra lineal*, Revista Facultad de Ciencias Exactas, Puras y Aplicadas Universidad Nacional de Tucuman, Serie A (Matematicas y Fisica Teorica) **5** (1946), 147–151.

[6] R. Burkard, *Admissible transformations and assignment problems*, Kombinatorische Optimierung (2007), 3–5.

[7] R. Burkard, M. Dell'Amico i S. Martello, *Assignment Problems*, SIAM, 2009.

[8] J. Burkardt, *Data for the 01 Knapsack Problem*, Florida State University (FSU), `https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html`.

[9] B. Clapper, *Munkres - Python module*, `https://pypi.python.org/pypi/munkres`.

[10] W.H. Cunningham, *A network simplex method*, Math. Program. **11** (1976), 105–116.

[11] E.A. Dinic i M.A Kronrod, *An algorithm for the solution of the assignment problem*, Sov. Math. Dokl. **10** (1969), 1324–1326.

[12] T.E. Easterfield, *A combinatorial algorithm*, J. London Math. Soc. **21** (1946), 219–226.

[13] P. Hall, *On representatives of subsets*, J. London Math. Soc. **s1-10** (1935), 26–30.

[14] E. Horowitz i S. Sahni, *Computing Partitions with Applications to the Knapsack Problem*, J. ACM **21** (1974), 277–292.

[15] P. J. Kolesar, *A branch and bound algorithm for the knapsack problem*, Management Science **136** (1967).

[16] B. Korte i J. Vygen, *Combinatorial Optimization: theory and algorithms*, Springer, 2006.

[17] H.W. Kuhn, *The Hungarian method for the assignment problem*, Naval Res. Log. Quart. **2** (1955), 83–97.

[18] H.W. Kuhn, *Variants of the Hungarian method for the assignment problem*, Naval Res. Log. Quart. **3** (1956), 253–258.

[19] S. Martello i P. Toth, *An algorithm for the generalized assignment problem*, Operational research **81** (1981), 589–603.

[20] S. Martello i P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Inc., 1990.

[21] G. B. Mathews, *On the partition of numbers*, Proceedings of the London Mathematical Society **28** (1897), 486—-490.

[22] G. T. Ross i R.M. Soland, *A branch and bound algorithm for the generalized assignment problem*, Mathematical Programming **8** (1975).

[23] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, 1986.

Sažetak

Zamislite da ste lopov koji se nalazi u sefu nekog poznatog muzeja. Namjera vam je ukrasti što vrjednije predmete, ali dolazite do problema - na raspolaganju imate malu naprtnjaču. Želite odabrati predmete koji će stati u naprtnjaču, a da im ukupna vrijednost bude što veća.

Možete zamisliti i jedan legalan scenarij: imate n poslova i isto toliko radnika. Svakom radniku trebate platiti da odradi po jedan posao, a svaki od njih ima drugačiju cijenu za pojedini posao. Naravno, cilj vam je dodijeliti poslove tako da vam trošak bude što manji.

Poopćeni problem dodjeljivanja je generalizacija ova dva problema. On pretpostavlja da imate n predmeta i m naprtnjača, te da vrijednost i težina svakog predmeta ovisi i o naprtnjači u koju ćete ga staviti. Cilj vam je rasporediti sve predmete po naprtnjačama na takav način da ukupna vrijednost bude što veća, a da se ne prekorači kapacitet ni jedne naprtnjače.

Ovaj diplomski rad se bavi upravo tim trima problemima. Svaki problem ćemo opisati, analizirati i navesti neke algoritme za njihovo rješavanje. Opisat ćemo i način na koji su problem naprtnjače i linearni problem dodjeljivanja povezani sa poopćenim problemom dodjeljivanja. Na kraju rada ćemo reći nešto više i o implementaciji algoritama za rješavanje ovih problema te navesti neke eksperimentalne rezultate.

Summary

Imagine yourself in an action movie scene: you are a thief inside a museum vault. You are there to steal as much valuable items as you can, but there is a problem - you only have a small knapsack. You want to choose items that will fit into your knapsack, but will also have the highest possible value.

You can imagine another situation: you have n number of jobs that need to be done by n workers. For every job, every worker has a price which you need to pay in order for him to get that job done. You need to assign one job to every worker with the objective of minimizing the total cost.

Generalized Assignment Problem is a generalization of these two problems. You are given n number of items and m number of knapsacks, with the value and size of each item depending on the knapsack you will put it in. Your goal is to maximize the total value by placing each item in one of the knapsacks, making sure none of the knapsacks is overfilled.

This master thesis deals with exactly these problems. We will describe and analyze all three problems and give a few algorithms that solve them. We will also describe the connection between Linear Assignment Problem, Knapsack Problem and Generalized Assignment Problem. At the end we will present implementation of some algorithms that are discussed throughout the thesis.

Životopis

Rođen sam 28. listopada 1993. godine u Zadru gdje sam i odrastao. Od početka osnovne škole privlači me matematika, što je utjecalo na to da upišem prirodoslovno-matematički razred u Gimnaziji Jurja Barakovića. Tijekom sva četiri razreda srednje škole ostvarivao sam odličan uspjeh, te sam sudjelovao na raznim natjecanjima iz matematike i informatike.

Moje zanimanje za matematiku je presudilo da 2012. godine upišem Preddiplomski studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu. Tijekom preddiplomskog studija moje zanimanje za računarstvo dodatno se produbilo, te sam većinu računarskih kolegija s lakoćom pohađao i polagao. Zbog toga sam 2015. godine upisao Diplomski studij Računarstva i matematike. Tijekom cijelog studija najveći fokus i interes iz područja računarstva bilo mi je teorijsko računarstvo te napredni algoritmi i strukture podataka, što je uvelike utjecalo na moj izbor mentora i teme diplomskog rada.

Tijekom studija, dvije godine bio sam član studentske udruge eSTUDENT, u sklopu koje sam sudjelovao u organizaciji TEDxUniversityofZagreb konferencije te App Start Contesta, studentskog natjecanja u izradi mobilnih aplikacija. Od prosinca 2016. godine sam, uz fakultetske obaveze, u tvrtki ReversingLabs zaposlen kao Student Backend Developer.