Aproksimacijski algoritmi - klase složenosti i dizajn

Dujić, Marko

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Marko Dujić

APROKSIMACIJSKI ALGORITMI - KLASE SLOŽENOSTI I DIZAJN

Diplomski rad

Voditelj rada:
Izv.prof.dr.sc.Mladen Vuković

Zagreb, studeni 2018.

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. _____

Sadržaj

Uvod **1**

1 Optimizacijski problemi **2**
1.1 Analiza složenosti problema 2
1.2 Klase složenosti P, NP, PSPACE i EXPTIME 5
1.3 Redukcije 9
1.4 Optimizacijski problemi - definicija i primjeri 13
1.4.1 0 - 1 problem ruksaka 14
1.4.2 Problem trgovačkog putnika 15
1.4.3 Maksimalni k-obojiv podgraf 16
1.4.4 Minimalno bojenje grafa 16

2 Klase PO i NPO **18**

3 Dizajnerske tehnike za aproksimacijske algoritme **23**
3.1 Pohlepna metoda 24
3.1.1 Pohlepni algoritam za 0 - 1 PROBLEM RUKSAKA 24
3.1.2 Pohlepni algoritam za problem MAKSIMALAN NEZAVISAN SKUP 26
3.1.3 Pohlepni algoritam za problem TRGOVAČKOG PUTNIKA . 27
3.2 Sekvencijalni algoritmi za problem particije 28
3.2.1 Raspored poslova na jednake strojeve 29
3.2.2 Sekvencijalni algoritmi za problem MINIMALNO GRUPIRANJE 30
3.2.3 Sekvencijalni algoritmi za problem MINIMALNO BOJENJE GRAFA 32
3.3 Lokalno pretraživanje 34
3.3.1 Algoritmi lokalnog pretraživanja za problem MAKSIMALNI REZ 34
3.3.2 Algoritmi lokalnog pretraživanja za problem TRGOVAČKOG PUTNIKA 35

3.4 Algoritmi bazirani na linearnom programiranju 36
3.4.1 Zaokruživanje rješenja cjelobrojnog linearnog programiranja . 36
3.4.2 Primal-dual algoritmi 37
3.5 Dinamičko programiranje 38

Bibliografija **42**

Uvod

U matematici se svakodnevno susrećemo s mnogim problemima za koje ne postoji egzaktni način rješavanja ili je njihova prostorna ili vremenska složenost rješavanja velika. Za takve probleme postoje metode rješavanja koje približno ili pak točno u nekim slučajevima rješavaju te probleme. Takvi problemi nazivaju se optimizacijski problemi i njima ćemo se baviti u ovom diplomskom radu. Rad je podijeljen u tri dijela.

U prvom dijelu objasnit ćemo najprije pojmove Turingov stroj i algoritam kao osnovu za sve čime ćemo se dalje baviti. Turingov stroj nam je potreban da bismo objasnili pojam algoritma, a algoritmi su nam potrebni da bismo riješili neki problem. Probleme ćemo klasificirati u dvije skupine. Za jedne ćemo navesti samo glavne karakteristike, dok je drugima posvećen ostatak diplomskog rada. Problemi se mogu transformirati u druge probleme tako da rješenjem jednog dobijemo rješenje drugog problema. Te transformacije ćemo nazivati redukcije i bit će nam potrebna u većini dokaza. U zadnjoj točki prvog dijela definirat ćemo optimizacijske probleme što je i tema ovog diplomskog rada.

U drugom dijelu diplomskog rada definiramo dvije skupine, odnosno klase, u koje svrstavamo optimizacijske probleme. Ovaj dio se sastoji uglavnom od teorema i dokaza koji vrijede za te dvije klase. Svakom optimizacijskom problemu pridružit ćemo tri potproblema koja ćemo koristiti u dokazima. Objasnit ćemo vezu tih klasa s klasama koje su karakteristične za probleme opisane u prvom dijelu. Uz to ćemo još i definirati NP-teške probleme čijim dizajniranjem se bavimo u trećem dijelu.

U trećem dijelu diplomskog rada opisat ćemo neke dizajnerske tehnike za algoritme koji aproksimacijom pokušavaju doći do optimalnog rješenja. Navest ćemo algoritme za neke NP-teške probleme te opisati koji je odnos između tako dobivenih rješenja i optimalnih rješenja. Na taj način ćemo znati koliko je određena metoda "dobra" za rješavanje nekog problema.

Poglavlje 1

Optimizacijski problemi

U ovom poglavlju najprije ćemo objasniti pojam Turingovog stroja, algoritma i problema. Nećemo formalno definirati pojmove algoritma i problema već ćemo ih opisati pomoću Turingovog stroja. Definirat ćemo vremensku i prostornu složenost nekog problema. Objasnit ćemo što su problemi odluke te odlučivi i neodlučivi problemi. Bavit ćemo se klasama složenosti. Problemi se mogu svesti na neki drugi problem koristeći redukciju. Definirat ćemo redukciju u Karpovom i Turingovom smislu. Zatim ćemo definirati optimizacijske probleme i opisati neke od njih.

1.1 Analiza složenosti problema

Teorijski model izračunavanja kojim ćemo se ovdje baviti bit će Turingov stroj. Zamišljamo ga kao stroj koji ima beskonačnu traku registara i glavu koja se po toj traci kreće (pomiče se lijevo, desno ili ostaje na mjestu) dok ne stane te čita podatke s trake i zapisuje nove. Po formalnoj definiciji stroj ima konačan skup stanja, konačan skup znakova s kojima radi i koji nazivamo abeceda, početno stanje, skup završnih stanja te funkciju prijelaza koja opisuje prijelaz između stanja.

Imamo deterministički i nedeterministički Turingov stroj. Kod determinističkog Turingovog stroja iz jednog stanja možemo preći u samo jedno drugo stanje. Kod nedeterminističkog taj prijelaz nije jedinstven. Primjerice, ako se na traci nalazimo u nekom registru gdje je zapisan određeni znak i u određenom smo stanju, glavu Turingovog stroja možemo pomaknuti lijevo, desno ili ostati na mjestu i zapisati točno određeni znak te prijeći u drugo (ili isto) stanje. Nedeterministički stroj dozvoljava da za neko stanje u kojem se nalazimo, možemo prijeći u različita stanja (ili zapisati različite znakove te pomaknuti glavu lijevo ili desno) odnosno pruža nam mogućnost izbora.

Pod pojmom problema podrazumijevat ćemo zapravo skup rješenja tog problema. Promatrat ćemo dva tipa problema: probleme odluke i optimizacijske probleme. U sljedećim točkama opisat ćemo te probleme. Ovdje su nam bitna samo njihova rješenja i kako ih možemo povezati s Turingovim strojem.

Ako imamo problem odluke, skup njegovih rješenja možemo opisati koristeći formalni jezik. Prvo definiramo alfabet $\mathcal{A}$ kao neprazan skup znakova (simbola). Riječ alfabeta je svaki konačan niz danog alfabeta. Skup svih riječi nad alfabetom $\mathcal{A}$ označavamo s $\mathcal{A}^\star$. Kažemo da je skup $\mathcal{L}$ jezik ako vrijedi $\mathcal{L} \subseteq \mathcal{A}^\star$. Svaki skup rješenja problema možemo poistovjetiti s nekim jezikom $\mathcal{L}$. Ako imamo zadani problem za neke općenite ulazne podatke, instanca problema bit će isti taj problem sa zadanim vrijednostima ulaznih podataka. Sada za neku instancu problema Turingov stroj može odrediti pripada li ona skupu rješenja problema, odnosno jeziku koji taj stroj odlučuje.

Kasnije ćemo pokazati da se optimizacijski problem može "pretvoriti" u odgovarajući problem odluke pa se njegovo rješavanje svodi na rješavanje problema odluke.

Za neke probleme rješenje postoji, za neke ne postoji. Ako postoji rješenje, tada Turingov stroj obavlja konačan broj prijelaza između stanja i zapisivanja na traku da dođe u završno stanje. U suprotnom, ako ne postoji rješenje, Turingov stroj će raditi beskonačno odnosno neće doći u završno stanje. Za postupak koji Turingov stroj koristi da dođe u završno stanje ili pak radi beskonačno koristit ćemo riječ algoritam. Nećemo formalno definirati algoritam već ćemo ga koristiti na opisani način. U skladu s tim da Turingov stroj može za neke ulaze stati ili raditi beskonačno, imat ćemo algoritme koji staju ili rade beskonačno, odnosno ne staju. Stoga imamo sljedeća dva slučaja: postoji algoritam koji za svaku instancu problema staje ili postoji neka instanca danog problema za koju algoritam radi beskonačno. Za prvi slučaj ćemo reći da algoritam za dani problem staje, a za drugi da algoritam ne staje ili radi beskonačno.

U daljnjim razmatranjima pretpostavljamo da ako imamo zadan problem, tada algoritam staje za sve instance tog problema. Algoritme koji ne staju nećemo razmatrati.

Sada možemo reći da algoritam rješava neki problem ako je potreban konačan broj koraka Turingovom stroju da dođe u završno stanje, za svaku instancu problema. Taj broj koraka je vrijeme izvršavanja algoritma. Pritom uzimamo da je korak algoritma jedan prijelaz između dva stanja (ista ili različita).

U skladu s deterministčkim i nedeterminističkim Turingovim strojevima imamo determinističke i nedeterminističke algoritme.

Svaki algoritam koji rješava neki problem ima određeno vrijeme izvršavanja i memoriju koju pritom zauzima (ako izvršavamo algoritam na računalu). Ta dva aspekta složenosti algoritma zovemo vremenska i prostorna složenost. Pritom nam

neće biti bitno točno vrijeme izvršavanja ili prostor koji zauzima algoritam u računalu za sve moguće ulazne podatke. Te vrijednosti je uglavnom teško izračunati pa ćemo ih približno određivati. Za približno određivanje bitno je kako se mijenja vrijeme izvršavanja ako količina ulaznih podataka raste. Uzimat ćemo u obzir vremensku ili prostornu složenost u najgorem slučaju i rjeđe u prosječnom slučaju. Analiza najgoreg slučaja daje nam sa sigurnošću gornju (vremensku ili prostornu) granicu izvršavanja algoritma. Definiramo sada vremensku i prostornu složenost u najgorem slučaju.

Definicija 1.1. *Neka je $\hat{t}_{\mathcal{A}}(x)$ vrijeme izvršavanja algoritma $\mathcal{A}$ za ulazni podatak x. Vremenska složenost algoritma $\mathcal{A}$ u najgorem slučaju je dana s*

$$t_{\mathcal{A}}(n) = max\{\hat{t}_{\mathcal{A}}(x) \,|\, \forall x\colon |x| \leq n\},\ n \in \mathbb{N}.$$

Slično definiramo prostornu složenost u najgorem slučaju.

Definicija 1.2. *Neka je $\hat{s}_{\mathcal{A}}(x)$ prostor potreban za izvršenje algoritma $\mathcal{A}$ za ulazni podatak x. Prostorna složenost algoritma $\mathcal{A}$ u najgorem slučaju je dana s*

$$s_{\mathcal{A}}(n) = max\{\hat{s}_{\mathcal{A}}(x) \,|\, \forall x\colon |x| \leq n\},\ n \in \mathbb{N}$$

Asimptotskom analizom određivat ćemo približnu složenost algoritma za velike n kao ulazne podatke.

Definicija 1.3 ($\mathcal{O}$ notacija)**.** *Neka su f i g funkcije, $f, g\colon \mathbb{N} \to \mathbb{R}^+$. Pišemo da je $f(n) = \mathcal{O}(g(n))$ ako postoje $c > 0$ i prirodan broj n_0 takvi da za svaki prirodan broj $n \geq n_0$ vrijedi*

$$f(n) \leq c\, g(n).$$

Definicija 1.4 (Ω notacija)**.** *Neka su f i g funkcije, $f, g\colon \mathbb{N} \to \mathbb{R}^+$. Pišemo da je $f(n) = \Omega(g(n))$ ako postoje $c > 0$ i prirodan broj n_0 takvi da za svaki prirodan broj $n \geq n_0$ vrijedi*

$$f(n) \geq c\, g(n).$$

Definicija 1.5 (Θ notacija)**.** *Neka su f i g funkcije, $f, g\colon \mathbb{N} \to \mathbb{R}^+$. Ako istovremeno vrijedi $f(n) = \mathcal{O}(g(n))$ i $f(n) = \Omega(g(n))$, tj. ako postoje $b, c > 0$ i prirodan broj n_0 takvi da za svaki prirodan broj $n \geq n_0$ vrijedi*

$$b\, g(n) \leq f(n) \leq c\, g(n).$$

Tada pišemo $f(n) = \Theta(g(n))$.

Definicija 1.6. *Za algoritam $\mathcal{A}$ kažemo da ima vremensku složenost:*

- $\mathcal{O}(g(n))$ *ako je* $t_{\mathcal{A}}(n) = \mathcal{O}(g(n))$;
- $\Omega(g(n))$ *ako je* $t_{\mathcal{A}}(n) = \Omega(g(n))$;
- $\Theta(g(n))$ *ako je* $t_{\mathcal{A}}(n) = \Theta(g(n))$.

Analogne su definicije za slučaj prostorne složenosti.

1.2 Klase složenosti P, NP, PSPACE i EXPTIME

U ovoj točki proučavamo probleme koji za skup rješenja imaju samo pozitivan i negativan odgovor, odnosno "da" ili "ne". Takvi problemi i algoritmi za njihovo rješavanje imaju različite vremenske i prostorne složenosti. Da bismo proučavali relativne odnose između njihovih složenosti, svrstavamo ih u određenu klasu složenosti. Postoji hijerarhija između klasa složenosti pa pomoću njih za neki problem možemo reći da je složeniji od drugog. Sada ćemo objasniti probleme odluke kojima ćemo se baviti u ovoj točki.

Za problem $\mathcal{P}$ kažemo da je problem odluke ako skup $I_{\mathcal{P}}$ svih instanci problema $\mathcal{P}$ možemo podijeliti u dva skupa: skup $Y_{\mathcal{P}}$ pozitivnih instanci i skup $N_{\mathcal{P}}$ negativnih instanci. Pozitivne instance problema su sve instance koje kao rješenje problema daju pozitivan odgovor "da". Negativne instance problema daju rezultat "ne".

Napomena. *U prošloj točki smo istaknuli da pod problemom $\mathcal{P}$ smatramo zapravo skup svih rješenja tog problema odnosno skup svih pozitivnih instanci $Y_{\mathcal{P}}$. Stoga možemo reći da je problem $\mathcal{P}$ zapravo skup jednak skupu $Y_{\mathcal{P}}$. Koristimo oznaku $Y_{\mathcal{P}}$ kada želimo istaknuti razliku između pozitivnih i negativnih instanci.*

Sada navodimo dva primjera problema odluke.

Primjer 1. *Problem SAT sastoji se od određivanja je li proizvoljna konjunktivna normalna forma (kraće knf) ispunjiva. Problem točnije definiramo:*

$$SAT = \{F : F \text{ je ispunjiva knf}\}.$$

Ne mora za svaki problem odluke postojati algoritam koji ga rješava. U sljedećem primjeru je definiran problem za koji algoritam ne staje.

Primjer 2. *Za dani računalni program i ulaz za taj program, pitanje koje se postavlja je završava li program u konačnom vremenu ili radi beskonačno. Ovaj problem odluke naziva se još halting problem. Preciznije definiramo taj problem s:*

$$HALT_{TM} = \{(M, w) : M \text{ je algoritam koji staje za ulazni podatak } w\}$$

Još je Turing dokazao da ne postoji algoritam koji rješava navedeni problem. Vidi [4].

Naveli smo dva primjera problema odluke. Za jedan problem postoji algoritam koji ga rješava, a za drugi ne postoji. S obzirom na to, problem odluke može biti odlučiv ili neodlučiv problem.

Definicija 1.7. *Za problem odluke $\mathcal{P}$ kažemo da je odlučiv ako postoji algoritam koji ga rješava, odnosno ako algoritam staje za svaki $x \in I_{\mathcal{P}}$ kao ulazni podatak, te vraća vrijednost "da" ako $x \in Y_{\mathcal{P}}$ i "ne" ako $x \in N_{\mathcal{P}}$. Problem odluke je neodlučiv ako nije odlučiv.*

Sada kad smo objasnili što su problemi odluke možemo promotriti hijerarhiju među tim problemima. Glavni cilj teorije složenosti algoritama je proučavanje skupova problema i njihova podjela s obzirom na složenost izvršavanja (vremensku, prostornu). Ta podjela će biti po klasama složenosti. Klasa složenosti je zapravo skup svih problema odluke koji mogu biti riješeni koristeći određenu količinu nekog računskog resursa (u našem slučaju prostora ili vremena). Sada ćemo definirati neke poznate klase složenosti.

Definicija 1.8. *Za bilo koju funkciju $f : \mathbb{N} \to \mathbb{N}$ s DTIME(f(n)), (DSPACE(f(n))) označavamo skup svih problema odluke rješivih determinističkim algoritmom s vremenskom (prostornom) složenošću $\mathcal{O}(f(n))$.*

Tu je navedena stroga podjela problema odluke s obzirom radi li se o prostornoj ili vremenskoj složenosti. Nas će zanimati finija podjela ovisno o kojoj funkciji f (iz prethodne definicije) se radi. Ovdje definiramo tri klase složenosti: P, PSPACE i EXPTIME.

1. klasa problema rješivih u polinomnom vremenu:

$$\mathrm{P} = \bigcup_{k=0}^{\infty} \mathrm{DTIME}(n^k)$$

2. klasa problema rješivih u polinomnom prostoru:

$$\mathrm{PSPACE} = \bigcup_{k=0}^{\infty} \mathrm{DSPACE}(n^k)$$

3. klasa problema rješivih u eksponencijalnom vremenu:

$$\mathrm{EXPTIME} = \bigcup_{k=0}^{\infty} \mathrm{DTIME}(2^{n^k})$$

Klasa složenosti P sadrži mnogo problema koji su efikasno rješivi nekim polinomno vremenski složenim algoritmom. Navodimo jedan primjer problema koji pripada klasi P.

Primjer 3. *Definiramo problem egzistencije puta između dva čvora u grafu:*

$$PUT = \{(G, x, y) : G \text{ je graf }, x, y \in G, \text{ postoji put od } x \text{ do } y\}$$

Vrijedi da je $PUT \in P$.

Još neki problemi koji pripadaju klasi P su: problem određivanja najveće zajedničke mjere dvaju prirodnih brojeva, problem egzistencije rješenja sistema linearnih algebarskih jednadžbi, problem određivanja istinitosti formule logike sudova ako su zadane vrijednosti propozicionalnih varijabli te problem određivanja prostosti broja.

Poznato je da vrijedi sljedeće:

$$\mathrm{P} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME}$$

Dosad smo koristili samo deterministične algoritme za rješavanje problema. Za njih je svojstveno da za svako stanje u kojem se algoritam nalazi postoji samo jedno izlazno stanje. Postoje također i nedeterministički algoritmi koji imaju sposobnost grananja neke odluke kako bi se ispitalo više različitih mogućnosti. Najpoznatiji primjer problema rješivog polinomnim nedeterminističkim algoritmom je problem SAT. U svakom koraku nedeterministički algoritam uzima jedan od $2^{|V|}$ kandidata za vrijednosti propozicionalnih varijabli logike sudova pri čemu je $|V|$ broj propozicionalnih varijabli logike sudova u formuli i ispituje istinitost formule logike sudova.

Definicija 1.9. *Za proizvoljnu funkciju* $f : \mathbb{N} \to \mathbb{N}$ *s NTIME(f(n)) označavamo skup svih problema odluke rješivih nedeterminističkim algoritmom s vremenskom složenošću* $\mathcal{O}(f(n))$.

Sada uvodimo novu klasu složenosti. Svi problemi odluke u toj klasi bit će rješivi u polinomnom vremenu nedeterminističkim algoritmom. Tu klasu ćemo označavati s NP i za nju vrijedi:

$$\mathrm{NP} = \bigcup_{k=0}^{\infty} \mathrm{NTIME}(n^k).$$

Već smo na početku ove točke definirali najpoznatiji problem koji pripada ovoj klasi - problem SAT. Još neki primjeri takvih problema su: problem egzistencije Hamiltonovog puta u grafu, problem egzistencije nezavisnog skupa u grafu, problem faktorizacije prirodnih brojeva te problem cjelobrojnog linearnog programiranja.

Imamo još jednu inkluziju koja je očita.

$$\mathrm{P} \subseteq \mathrm{NP}$$

Svaki deterministički algoritam je specijalni slučaj nedeterminističkog jer imamo samo jednu granu izračunavanja. Dokaz da vrijedi obratna inkluzija zasad nije poznat, a u slučaju da ta inkluzija vrijedi, za sve probleme iz NP postojao bi polinomni deterministički algoritam koji ih rješava.

1.3 Redukcije

Redukcija je osnovni alat za transformaciju jednog problema u drugi problem. Redukcija problema $\mathcal{P}_1$ na problem $\mathcal{P}_2$ predstavlja metodu rješavanja problema $\mathcal{P}_1$ koristeći algoritam za $\mathcal{P}_2$. Postoji mnogo vrsta redukcija međutim ovdje ćemo definirati Karpovu i Turingovu redukciju. Neka su redom $\mathcal{P}_1$ i $\mathcal{P}_2$ proizvoljni problemi te $Y_{\mathcal{P}_1}$ i $Y_{\mathcal{P}_2}$ skupovi pozitivnih instanci tih problema. Sada definiramo Karpovu redukciju.

Definicija 1.10. *Za odlučiv problem $\mathcal{P}_1$ kažemo da je reducibilan u Karpovom smislu na problem $\mathcal{P}_2$ ako postoji algoritam $\mathcal{R}$ koji svaku instancu x problema $\mathcal{P}_1$ transformira u instancu y problema $\mathcal{P}_2$ tako da je $x \in Y_{\mathcal{P}_1}$ ako i samo ako $y \in Y_{\mathcal{P}_2}$. Za $\mathcal{R}$ kažemo da je Karpova redukcija problema $\mathcal{P}_1$ na $\mathcal{P}_2$ i pišemo $\mathcal{P}_1 \leq_m \mathcal{P}_2$. Ako vrijedi $\mathcal{P}_1 \leq_m \mathcal{P}_2$ i $\mathcal{P}_2 \leq_m \mathcal{P}_1$, kažemo da su $\mathcal{P}_1$ i $\mathcal{P}_2$ ekvivalentni u Karpovom smislu. To označavamo s $\mathcal{P}_1 \equiv_m \mathcal{P}_2$.*

U slučaju da je algoritam $\mathcal{R}$ iz prethodne definicije polinomne vremenske složenosti, radi se o polinomno vremenski složenoj Karpovoj redukciji (Oznaka: $\mathcal{P}_1 \leq_m^p \mathcal{P}_2$). Sada navodimo primjer Karpove redukcije jednog problema na drugi.

Primjer 4. *Problemi na kojima ćemo ilustrirati Karpovu redukciju su problem Hamiltonovog puta i problem Hamiltonovog ciklusa. Prvo opisujemo ta dva pojma. Ako su dani usmjereni graf $G = (V, E)$ i vrhovi $s, t \in V$, kažemo da postoji Hamiltonov put između s i t ako postoji put od s do t koji prolazi kroz sve vrhove grafa točno jednom. Ako je dan usmjereni graf $G = (V, E)$, kažemo da postoji Hamiltonov ciklus u grafu G ako postoji ciklus u grafu G koji prolazi kroz sve vrhove grafa točno jednom.*

Problemi nalaženja Hamiltonovog puta i ciklusa su problemi odluke odnosno pitanje je za dani graf postoji li Hamiltonov put ili ciklus u njemu. Problem Hamiltonovog puta ćemo označavati s $\mathcal{P}_1$, a problem Hamiltonovog ciklusa s $\mathcal{P}_2$. Sada ćemo pokazati da vrijedi $\mathcal{P}_1 \leq_m^p \mathcal{P}_2$. Trebamo definirati polinomno vremenski složen algoritam $\mathcal{R}$ koji će transformirati svaku pozitivnu instancu $x \in Y_{\mathcal{P}_1}$ problema $\mathcal{P}_1$ u pozitivnu instancu $y \in Y_{\mathcal{P}_2}$ problema $\mathcal{P}_2$ i obratno. Instanca problema $\mathcal{P}_1$ je oblika $(G = (V, E), s, t)$ gdje je G graf te s i t vrhovi, a instanca problema $\mathcal{P}_2$ je graf G. Algoritam $\mathcal{R}$ definiramo tako da transformira instancu $(G = (V, E), s, t)$ problema $\mathcal{P}_1$ u instancu $(G' = (V \cup \{u\}, E \cup \{(u, s), (t, u)\})$ problema $\mathcal{P}_2$, gdje je u novi čvor različit od svih čvorova iz V.

Sada trebamo pokazati da je algoritam $\mathcal{R}$ polinomne vremenske složenosti te da vrijede sljedeće tvrdnje:

1. *ako je $x \in Y_{\mathcal{P}_1}$ ulazni podatak za algoritam $\mathcal{R}$, tada je izlazna vrijednost algoritma $y \in Y_{\mathcal{P}_2}$*
2. *ako je $y \in Y_{\mathcal{P}_2}$ ulazni podatak za algoritam $\mathcal{R}$, tada je izlazna vrijednost algoritma $x \in Y_{\mathcal{P}_1}$*

Polinoman broj koraka je potreban da se doda novi vrh u graf G pa je algoritam $\mathcal{R}$ polinomne vremenske složenosti.

Sada pokazujemo da vrijede gornje dvije tvrdnje. Ako uzmemo instancu $x \in Y_{\mathcal{P}_1}$ za ulazni podatak algoritma $\mathcal{R}$, znamo da postoji Hamiltonov put $(v_0 = s, v_1, \ldots, v_n = t)$ u grafu G. Sada je izlazna vrijednost y algoritma $\mathcal{R}$ graf G' koji sadrži Hamiltonov ciklus $(u, v_0 = s, v_1, \ldots, v_n = t, u)$ pa je $y \in Y_{\mathcal{P}_2}$. Obratno, ako je $y \in Y_{\mathcal{P}_2}$ ulazni podatak za algoritam $\mathcal{R}$, znamo da u grafu G' postoji Hamiltonov ciklus koji u sebi sadrži bridove (u, s) i (t, u). Sada primjenom algoritma $\mathcal{R}$ izbacujemo vrh u iz ciklusa te dobivamo da je izlazna vrijednost x algoritma $\mathcal{R}$ graf G koji sadrži Hamiltonov put od s do t pa vrijedi $x \in Y_{\mathcal{P}_1}$.

Slijedi da je problem Hamiltonovog puta reducibilan u Karpovom smislu na problem Hamiltonovog ciklusa.

Postoji i općenitiji tip redukcije, koji možemo primijeniti i na probleme koji nisu odlučivi. To je tzv. Turingova redukcija. Da bismo definirali taj tip redukcije, iskoristit ćemo malo drugačiji Turingov stroj. Zamišljamo ga kao standardni Turingov stroj koji sadrži i tzv.*"crnu kutiju"* (eng. black box) koju nazivamo prorokom. U toj "kutiji" stroj će odlučivati neki problem odluke kao da se radi o jednoj instrukciji u algoritmu, odnosno za to je potreban jedan korak algoritma. Uz probleme odluke, u "kutiji" se mogu nalaziti i neodlučivi problemi, te optimizacijski problemi. Turingovom stroju kako smo ga opisali u prvoj točki, dodali bismo još jednu traku, koju nazivamo prorokom, za simuliranje "kutije" da bismo dobili rješenje nekog potproblema početnog problema. Zbog toga ovakav stroj se naziva **Turingov stroj s prorokom** za taj potproblem. Za taj stroj nije bitna vremenska i prostorna složenost problema koji su u "kutiji" što je bitno za bolje razumijevanje klasa složenosti i dokaze nekih teorema vezanih uz njih. Sada preciznije definiramo Turingov stroj s prorokom za neki problem.

Definicija 1.11. *Turingov stroj s prorokom za problem $\mathcal{P}$ (oznaka $\mathcal{T}_p$) je Turingov stroj koji ima još jednu dodatnu traku (s pokretnom glavom) koju zovemo prorokom i dva dodatna stanja: stanje zahtjeva q_z i stanje odgovora q_o. Stroj $\mathcal{T}_p$ prelazi u stanje q_z kada je u radu stroja $\mathcal{T}_p$ potrebno rješenje problema $\mathcal{P}$. Tada se sadržaj trake proroka interpretira kao instanca problema $\mathcal{P}$. Zatim se taj sadržaj mijenja rješenjem problema $\mathcal{P}$ za tu instancu i stroj $\mathcal{T}_p$ prelazi u stanje q_o. Sada se dobiveno rješenje s trake proroka koristi u daljnjem radu stroja $\mathcal{T}_p$ koji prelazi iz stanja q_o u neko drugo stanje. Uzima se da su za taj dio rada Turingovog stroja s prorokom za problem $\mathcal{P}$ potrebna samo dva prijelaza između stanja.*

Slično kao za Turingov stroj opisujemo algoritam s prorokom za neki problem. Znamo da za svaki algoritam $\mathcal{A}$ postoji odgovarajući Turingov stroj $\mathcal{T}_\mathcal{A}$. Ako je $\mathcal{T}_\mathcal{A}$ zapravo Turingov stroj s prorokom za neki problem $\mathcal{P}$, tada je i pripadajući algoritam $\mathcal{A}$ zapravo algoritam s prorokom za problem $\mathcal{P}$.

Objasnili smo algoritam s prorokom za neki problem da bismo sada mogli definirati redukciju problema u Turingovom smislu. Takva redukcija jednog problema na drugi postoji ako algoritam za prvi problem pretpostavlja da je rješenje drugog problema poznato i pritom ga koristi.

Definicija 1.12. *Problem $\mathcal{P}_1$ je reducibilan u Turingovom smislu na problem $\mathcal{P}_2$ ako postoji algoritam $\mathcal{R}$ za problem $\mathcal{P}_1$ s prorokom za $\mathcal{P}_2$. Tada kažemo da je $\mathcal{R}$ Turingova redukcija problema $\mathcal{P}_1$ na $\mathcal{P}_2$ i pišemo $\mathcal{P}_1 \leq_T \mathcal{P}_2$. Ako vrijedi $\mathcal{P}_1 \leq_T \mathcal{P}_2$ i $\mathcal{P}_2 \leq_T \mathcal{P}_1$, kažemo da su $\mathcal{P}_1$ i $\mathcal{P}_2$ ekvivalentni u Turingovom smislu. To označavamo s $\mathcal{P}_1 \equiv_T \mathcal{P}_2$.*

Kao i kod redukcija u Karpovom smislu, posebno promatramo polinomno vremenski složenu Turingovu redukciju (algoritam $\mathcal{R}$ iz prethodne definicije je polinomne vremenske složenosti). Tada pišemo $\mathcal{P}_1 \leq_T^p \mathcal{P}_2$. Jednostavnije rečeno, postoji polinomno vremenski složena Turingova redukcija problema $\mathcal{P}_1$ na problem $\mathcal{P}_2$ ako postoji polinomno vremenski složen algoritam za problem $\mathcal{P}_1$ s prorokom za problem $\mathcal{P}_2$. Vremenska i prostorna složenost potproblema koji je na traci proroka se zanemaruju pri računanju složenosti problema. Sada ćemo dati primjer Turingove redukcije jednog problema na drugi.

Primjer 5. *Prvo ćemo definirati problem $A_{TM} = \{$(M,x) : M je algoritam koji staje za ulazni podatak x i izlazna vrijednost algoritma je "da"$\}$. Sada ćemo pokazati da je problem A_{TM} reducibilan u Turingovom smislu na problem $HALT_{TM}$ definiran u prethodnoj točki. Dat ćemo algoritam za problem A_{TM} s prorokom za problem $HALT_{TM}$. Za bilo koji algoritam M i ulazni podatak x treba odrediti je li izlazna vrijednost algoritma "da" ili "ne".*
Ako vrijedi $(M, x) \in HALT_{TM}$ tada pokrenemo algoritam M s ulaznim podatkom x i imamo dva slučaja:

1. *ako je izlazna vrijednost algoritma "da", tada je $(M, x) \in A_{TM}$*
2. *u suprotnom, ako je izlazna vrijednost algoritma "ne", tada $(M, x) \notin A_{TM}$.*

Ako vrijedi $(M, x) \notin HALT_{TM}$, tada $(M, x) \notin A_{TM}$.
Za dani algoritam potreban je polinoman broj koraka da stane i vrati izlaznu vrijednost te stoga vrijedi $A_{TM} \leq_T^p HALT_{TM}$.

Sada kad smo definirali Karpovu i Turingovu redukciju možemo objasniti koji je odnos među njima. Ako su zadani problemi $\mathcal{P}_1$ i $\mathcal{P}_2$ i postoji Karpova redukcija problema $\mathcal{P}_1$ na problem $\mathcal{P}_2$, tada postoji i Turingova redukcija problema $\mathcal{P}_1$ na problem

$\mathcal{P}_2$. Kako vrijedi $\mathcal{P}_1 \leq_m \mathcal{P}_2$, postoji algoritam koji svaku pozitivnu instancu problema $\mathcal{P}_1$ transformira u pozitivnu instancu problema $\mathcal{P}_2$, te negativnu u negativnu. Sada postoji algoritam za problem $\mathcal{P}_1$ s prorokom za problem $\mathcal{P}_2$ jer na svakoj instanci x_1 problema $\mathcal{P}_1$ možemo iskoristiti proroka za problem $\mathcal{P}_2$ na odgovarajućoj instanci x_2 problema $\mathcal{P}_2$. Na taj način dobit ćemo rješenje problema $\mathcal{P}_1$ te vrijedi $\mathcal{P}_1 \leq_T \mathcal{P}_2$.

Obratan smjer općenito ne vrijedi, odnosno postoji Turingova redukcija koja nije Karpova.

Koristeći redukciju uvodimo definiciju NP-potpunih problema, ali prije toga uvodimo dvije pomoćne definicije: zatvorenost klasa složenosti i potpunost problema odluke. Pritom u tim definicijama relacija $\leq_r$ može biti Karpova ili Turingova redukcija. Za klasu složenosti uzimamo da je bilo koji skup problema odluke.

Definicija 1.13. *Klasa složenosti $\mathcal{C}$ je zatvorena s obzirom na relaciju $\leq_r$ ako za svaki par problema odluke $\mathcal{P}_1$, $\mathcal{P}_2$, takvih da $\mathcal{P}_1 \leq_r \mathcal{P}_2$ i $\mathcal{P}_2 \in \mathcal{C}$ vrijedi da je $\mathcal{P}_1 \in \mathcal{C}$.*

Definicija 1.14. *Neka je $\mathcal{C}$ neka klasa složenosti. Kažemo da je problem odluke $\mathcal{P} \in \mathcal{C}$ potpun u $\mathcal{C}$ s obzirom na neku relaciju $\leq_r$ ako za svaki problem odluke $\mathcal{P}_1 \in \mathcal{C}$ vrijedi $\mathcal{P}_1 \leq_r \mathcal{P}$.*

Sada definiramo NP-potpune probleme.

Definicija 1.15. *Problem odluke $\mathcal{P}$ je NP-potpun ako je potpun u NP s obzirom na relaciju $\leq_m^p$, tj. $\mathcal{P} \in NP$ i za bilo koji problem odluke $\mathcal{P}_1 \in NP$ vrijedi $\mathcal{P}_1 \leq_m^p \mathcal{P}$.*

Dok za probleme u klasi P imamo efikasne algoritme za njihovo rješavanje, danas nisu poznati efikasni algoritmi za probleme koji su NP-potpuni. Najpoznatiji primjer problema koji je *NP-potpun* je problem SAT. Vidi [7].

Kako bi se dokazalo da je neki NP problem $\mathcal{P}$ ujedno i NP-potpun, dovoljno je pokazati da postoji polinomna Karpova redukcija nekog NP-potpunog problema na $\mathcal{P}$. Još uvijek otvoreno pitanje u teoriji računarstva je vrijedi li P=NP. U slučaju da postoji polinomno vremenski složen algoritam koji rješava neki NP-potpun problem, tada bismo znali da za svaki NP problem postoji algoritam polinomne vremenske složenosti.

1.4 Optimizacijski problemi - definicija i primjeri

Dosad smo se uglavnom bavili problemima odluke. Sada ćemo se posvetiti definiciji optimizacijskih problema i najpoznatijim primjerima takvih. To su problemi koji ne pripadaju problemima odluke i čiji cilj je pronaći najbolje (najbliže egzaktnom) rješenje od svih mogućih rješenja. Svaki optimizacijski problem ima odgovarajući problem odluke što ćemo kasnije objasniti. Prvo uvodimo definiciju tih problema.

Definicija 1.16. *Optimizacijski problem $\mathcal{P}$ je uređena petorka $(I_\mathcal{P}, SOL_\mathcal{P}, \mathcal{R}, m_\mathcal{P}, goal_\mathcal{P})$ gdje je:*

1. *$I_\mathcal{P}$ neprazan skup čije elemente nazivamo instance problema $\mathcal{P}$;*
2. *$\mathcal{R}$ je neprazan skup čije elemente nazivamo dopustiva rješenja;*
3. *$SOL_\mathcal{P}$ je funkcija s domenom $I_\mathcal{P}$ i kodomenom $\mathcal{P}(\mathcal{R})$, tj. $SOL_\mathcal{P} : I_\mathcal{P} \to \mathcal{P}(\mathcal{R})$;*
4. *$m_\mathcal{P}$ je funkcija čiji su argumenti uređeni parovi oblika (x, y) gdje je $x \in I_\mathcal{P}$ i $y \in SOL_\mathcal{P}(x)$ te ima kodomenu $\mathbb{N}$; funkciju $m_\mathcal{P}$ nazivamo mjerna funkcija;*
5. *$goal_\mathcal{P} \in \{\min, \max\}$ gdje su* min *i* max *funkcije minimuma odnosno maksimuma na nekom skupu.*

Za danu instancu $x \in I_\mathcal{P}$, ne zanima nas skup $SOL_\mathcal{P}(x)$ svih dopustivih rješenja instance x nego podskup tog skupa koji ćemo označavati sa $SOL^\star_\mathcal{P}(x)$ i za čije elemente vrijedi sljedeće:

$$\forall y \in SOL^\star_\mathcal{P}(x) \quad m_\mathcal{P}(x, y) = goal_\mathcal{P}\{v | v = m_\mathcal{P}(x, z) \wedge z \in SOL_\mathcal{P}(x)\}$$

Skup $SOL^\star_\mathcal{P}(x)$ ćemo zvati skup optimalnih rješenja instance $x \in I_\mathcal{P}$. Mjeru $m_\mathcal{P}(x, y)$ optimalnog rješenja $y \in SOL^\star_\mathcal{P}(x)$ ćemo kraće označavati s $m^\star_\mathcal{P}(x)$ jer su mjere svih optimalnih rješenja iz skupa $SOL^\star_\mathcal{P}(x)$ jednake.

Primjer 6. *Neka je zadan graf $G = (V, E)$. Problem MINIMALNI POKRIVAČ VRHOVA sastoji se od pronalaženja minimalnog skupa U takvog da je za svaki brid $(v_i, v_j) \in E$ barem jedan od vrhova v_i, v_j u skupu U. Koristeći definiciju optimizacijskog problema, dani problem definiran je na sljedeći način:*

1. *$I_\mathcal{P} = \{G = (V, E) \mid G$ je graf$\}$*
2. *$SOL_\mathcal{P}(I_\mathcal{P}) = \{U \subseteq V \mid \forall (v_i, v_j) \in E \ (v_i \in U \vee v_j \in U)\}$*
3. *$m_\mathcal{P}(G, U) = |U|$*
4. *$goal_\mathcal{P} = \min$*

Sada se skup $SOL^{\star}_{\mathcal{P}}(G_1)$ svih optimalnih rješenja instance nekog grafa G_1 sastoji od svih podskupova U skupa svih vrhova V takvih da vrijedi:

$$|U| = \min\{v : v = |U_1| \wedge U_1 \in SOL_{\mathcal{P}}(G_1)\}$$

Svakom se optimizacijskom problemu $\mathcal{P}$ može pridružiti **problem odluke** $\mathcal{P}_D$. Za danu instancu problema $x \in I_{\mathcal{P}}$ i prirodan broj k, problem odluke je odrediti vrijedi li $m^{\star}_{\mathcal{P}}(x) \geq k$ (ako je $goal_{\mathcal{P}} = \max$) ili $m^{\star}_{\mathcal{P}}(x) \leq k$ (ako je $goal_{\mathcal{P}} = \min$).

Svakom se optimizacijskom problemu uz problem odluke mogu pridružiti još dva problema.

Konstruktivni problem - za instancu problema $x \in I_{\mathcal{P}}$ treba odrediti jedno optimalno rješenje $y \in SOL^{\star}_{\mathcal{P}}(x)$ i mjeru $m^{\star}_{\mathcal{P}}(x)$. Ovaj problem ćemo označavati s $\mathcal{P}_C$.

Evaluacijski problem - za instancu problema $x \in I_{\mathcal{P}}$ treba odrediti mjeru $m^{\star}_{\mathcal{P}}(x)$. Oznaka za ovaj problem bit će $\mathcal{P}_E$.

Treća varijanta problema je prije navedeni **problem odluke** $\mathcal{P}_D$.

Sada bi problem odluke za naš gornji primjer bilo pitanje: postoji li pokrivač vrhova na grafu G veličine manje ili jednake k ($k \in \mathbb{N}$) odnosno da vrijedi $|U| \leq k$? Za ovaj problem odluke se može pokazati da je NP-potpun. Postoji velik broj optimizacijskih problema među kojima su najpoznatiji: problem trgovačkog putnika, problem ruksaka, problem nalaženja maksimalnog k-obojivog podgrafa, problem minimalnog bojenja grafa...U sljedećim točkama detaljnije ćemo objasniti neke optimizacijske probleme.

1.4.1 0 - 1 problem ruksaka

Neka je dano n predmeta $y_0, \ldots, y_{n-1}$ pri čemu svaki od njih ima težinu w_i i cijenu p_i, $i = 0, \ldots, n-1$. Potrebno je utvrditi koje je predmete moguće odabrati tako da suma njihovih težina ne prelazi zadani prirodni broj k, a da je njihova ukupna vrijednost maksimalna. Tada je rješenje n-torka $X = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$ takva da je

$$w(X) = \sum_{i=0}^{n-1} x_i w_i \leq k$$

$$P(X) = \sum_{i=0}^{n-1} x_i p_i \rightarrow max$$

Uređenu n-torku interpretiramo ovako : ako smo i-ti predmet stavili u ruksak, $x_i = 1$, u suprotnom $x_i = 0$. Funkcija $w : \{0,1\}^n \rightarrow \mathbb{N}$ naziva se težinska funkcija i njome

računamo ukupnu sumu težina odabrane n-torke. Funkcija $P : \{0,1\}^n \to \mathbb{N}$ je funkcija profita.
0 - 1 problem ruksaka možemo formalno definirati na sljedeći način:

1. Instanca problema: skup predmeta $y_0, \ldots, y_{n-1}$ s težinama w_i i cijenama p_i, $i = 0, \ldots, n-1$, kapacitet ruksaka $k \in \mathbb{N}$; označimo instancu s x

2. Skup rješenja: podskup skupa svih predmeta $\{y_0, \ldots, y_{n-1}\}$ pri čemu je suma težina odabranih predmeta manja ili jednaka prirodnom broju k

3. Mjera: $m_{\mathcal{P}}(x) = \sum_{i=0}^{n-1} x_i p_i$ pri čemu je $x_i = 1$ ako je predmet odabran, inače 0

4. $goal_{\mathcal{P}} = \max$

Postoji još varijanti ovog problema. Broj kopija određenog predmeta koji stavljamo u ruksak ne mora biti 0 ili 1 već je ograničen nekim nenegativnim brojem. U drugoj varijanti taj broj ne mora biti ograničen već možemo staviti koliko god istih predmeta hoćemo u ruksak.

1.4.2 Problem trgovačkog putnika

Zadani problem glasi: Dani su skup gradova (mreža gradova) i udaljenosti između svaka dva grada. Problem je pronaći najkraći mogući put koji obilazi sve gradove točno jednom i vraća se u početnu točku.

Problem ima svoju reprezentaciju pomoću potpunog grafa. Graf ima n vrhova (gradova) s pozitivnim težinama bridova (udaljenostima). Dopustivo rješenje problema je svaki podgraf koji sadrži sve vrhove grafa i ciklički je.

Problem trgovačkog putnika možemo formalno definirati na sljedeći način:

1. Instanca problema: uređeni par $(\{c_1, \ldots, c_n\}, M)$ gdje je $\{c_1, \ldots, c_n\}$ zadani skup gradova i $M \in M_{n\times n}(\mathbb{Z}^+)$; elemente matrice M na mjestu (i, j) označavamo s $D(c_i, c_j)$; instancu označimo s x

2. Skup rješenja: neka permutacija skupa $\{c_1, \ldots, c_n\}$

3. Mjera: $m_{\mathcal{P}}(x) = \sum_{k=1}^{n-1} D(c_k, c_{k+1}) + D(c_n, c_1)$

4. $goal_{\mathcal{P}} = \min$

Danas ne znamo postoji li algoritam koji bi rješavao dani problem u polinomnom vremenu već se on efikasno rješava aproksimativnim metodama. Najjednostavnija metoda je metoda "grubom silom" generiranja svih permutacija gradova uz provjeru koja permutacija ima najkraću udaljenost obilaska svih gradova. Međutim ta metoda nije efikasna jer bi za veći broj gradova provjere jako dugo trajale. Kako bismo malo poboljšali efikasnost, koristimo pohlepnu metodu i lokalno pretraživanje. Navedenim metodama rješavanja ovog problema više ćemo se posvetiti u trećem poglavlju.

1.4.3 Maksimalni k-obojiv podgraf

Prvo navodimo definiciju nezavisnog skupa u grafu $G = (V, E)$ jer nam je potrebna u sljedeća dva primjera. Skup $S \subseteq V$ je nezavisan ako nikoja dva vrha iz S nisu povezani bridom u G.

Neka su zadani graf $G = (V, E)$ i $k \in \mathbb{N}$. Tada je k-bojenje vrhova od G funkcija $c : V \to \{1, 2, \ldots, k\}$ koja svakom vrhu pridruži jednu od k boja. Ako je $c(v) = i$, kažemo da je vrh v obojen bojom i. Pravilno k-bojenje grafa bez petlji je rastav $(C_1, C_2, \ldots, C_k)$ od V na k disjunktnih skupova $C_i = \{v \in V \mid c(v) = i\}$ (mogu biti i prazni) koji su nezavisni. Ograničujemo se samo na jednostavne grafove bez petlji. Graf je k-obojiv ako dopušta pravilno k-bojenje.

Još nam je potrebna definicija podgrafa. Za graf $G' = (V', E')$ kažemo da je podgraf grafa $G = (V, E)$ ako je $V' \subseteq V$, te je $E' \subseteq E \cap V' \times V'$. Specijalno, podgraf G' u našem slučaju imat će skup vrhova V' jednak skupu vrhova V početnog grafa G. Sada formalno definiramo problem maksimalnog k-obojivog podgrafa.

1. Instanca problema: graf $G = (V, E)$, $k \in \mathbb{N}$; označimo instancu s x
2. Skup rješenja: podskup $E' \subseteq E$ takav da je podgraf $G' = (V, E')$ k-obojiv
3. Mjera: $m_{\mathcal{P}}(x)$ = kardinalnost bridova podgrafa, tj. $|E'|$
4. $goal_{\mathcal{P}} = \max$

1.4.4 Minimalno bojenje grafa

Neka je zadan graf $G = (V, E)$. Treba odrediti koliko je boja najmanje potrebno da obojimo sve vrhove grafa G, a da pritom nikoja dva vrha povezana bridom ne budu obojana istom bojom. Navedeni problem zovemo minimalno bojenje grafa. Formalno ga možemo definirati na sljedeći način:

1. Instanca problema: graf $G = (V, E)$; označimo instancu s x

2. Skup rješenja: particija skupa vrhova V na k disjunktnih skupova $V_1, V_2, \ldots, V_k$ takve da je svaki V_i nezavisan u G

3. Mjera: $m_{\mathcal{P}}(x) =$ broj disjunktnih nezavisnih skupova, tj. k

4. $goal_{\mathcal{P}} = \min$

Poglavlje 2

Klase PO i NPO

Kao što se problemi odluke svrstavaju u određene klase složenosti, tako i za optimizacijske probleme imamo odgovarajuću klasifikaciju. Od velike važnosti je odrediti postoji li algoritam polinomne vremenske složenosti koji za svaku instancu problema određuje optimalno rješenje i njegovu vrijednost. Prije nego uvedemo definiciju klase složenosti NPO za optimizacijske probleme, objasnit ćemo jedno svojstvo problema iz klase NP koje će nam biti potrebno u definiciji NPO. Za svaki problem iz klase NP lako je provjeriti je li nešto njegovo rješenje. Ta tvrdnja slijedi iz teorema o certifikatu:

Teorem 2.1. *Za svaki problem odluke $\mathcal{P}$ vrijedi:*

$$\mathcal{P} \in NP \Leftrightarrow (\exists \mathcal{R} \in P)(\exists k \in \mathbb{N}) \\ \mathcal{P} = \{x : (\exists c)(|c| = \mathcal{O}(|x|^k) \wedge \mathcal{R}(x, c))\}.$$

Instancu problema c nazivamo certifikat za instancu x, a problem čiji skup rješenja sadrži sve certifikate nazivamo certifikat za problem $\mathcal{P}$.

U prethodnoj definiciji instance problema $\mathcal{R} \in P$ su uređeni parovi oblika (x, c) pri čemu je x neka instanca problema $\mathcal{P}$, a c certifikat za instancu x.

Slijedi definicija klase NPO. S kraticom $SOL_{\mathcal{P}}$ ćemo u sljedećoj definiciji označavati skup svih rješenja optimizacijskog problema $\mathcal{P}$, odnosno sliku funkcije $SOL_{\mathcal{P}}$.

Definicija 2.1. *Optimizacijski problem $\mathcal{P} = (I_{\mathcal{P}}, SOL_{\mathcal{P}}, \mathcal{R}, m_{\mathcal{P}}, goal_{\mathcal{P}})$ pripada klasi NPO ako vrijedi sljedeće:*

1. *skup instanci $I_{\mathcal{P}}$ je prepoznatljiv (Turing-prepoznatljiv) u polinomnom vremenu;*
2. *postoji polinom q takav da za svaku instancu $x \in I_{\mathcal{P}}$ i svaki $y \in SOL_{\mathcal{P}}(x)$ vrijedi $|y| \leq q(|x|)$ i obratno, za svaki y takav da je $|y| \leq q(|x|)$ u polinomnom*

vremenu je odlučivo vrijedi li $y \in SOL_{\mathcal{P}}(x)$, odnosno po teoremu o certifikatu $SOL_{\mathcal{P}} \in NP$;

3. *postoji algoritam koji u polinomnom vremenu određuje vrijednost mjere $m_{\mathcal{P}}$.*

Sada navodimo primjer problema koji pripada klasi NPO.

Primjer 7. *Problem MINIMALNI POKRIVAČ VRHOVA pripada klasi NPO jer vrijedi:*

1. *Skup instanci odnosno skup neusmjerenih grafova je prepoznatljiv u polinomnom vremenu.*

2. *Svako moguće rješenje problema je podskup skupa vrhova pa sigurno ima manje vrhova nego instanca. Za polinom q možemo uzeti da je identiteta. S druge strane da bi $U \subseteq V$ bilo moguće rješenje moramo provjeriti je li svaki brid u E incidentan s barem jednim vrhom u U. To možemo napraviti u polinomnom vremenu.*

3. *Ako je U moguće rješenje, mjeru $m_{\mathcal{P}}$ ($|U|$) možemo odrediti u polinomnom vremenu.*

Postoji odnos između klasa NPO i NP ako promatramo odgovarajući problem odluke $\mathcal{P}_D$ za problem $\mathcal{P}$ iz klase NPO. To dokazujemo sljedećim teoremom.

Teorem 2.2. *Za svaki optimizacijski problem $\mathcal{P} \in NPO$ odgovarajući problem odluke $\mathcal{P}_D$ pripada klasi NP.*

Dokaz. Pretpostavimo da je problem $\mathcal{P}$ problem maksimizacije. Dokaz za problem minimizacije je sasvim analogan. Neka je q polinom takav da za svaku instancu $x \in I_{\mathcal{P}}$ i svaki $y \in SOL_{\mathcal{P}}(x)$ vrijedi $|y| \leq q(|x|)$ i obratno, za svaki y takav da je $|y| \leq q(|x|)$ u polinomnom vremenu je odlučivo vrijedi li $y \in SOL_{\mathcal{P}}(x)$. Za danu instancu $x \in I_{\mathcal{P}}$ i pozitivni cijeli broj k, konstruiramo nedeterministički Turingov stroj M koji odlučuje odgovarajući problem odluke $\mathcal{P}_D$ u polinomnom vremenu:

1. stroj M redom kreira svaku riječ y danog alfabeta za koju vrijedi $|y| \leq q(|x|)$;

2. M nedeterministički provjerava je li $y \in SOL_{\mathcal{P}}(x)$, ako nije, stroj završava u stanju odbijanja;

3. ako $y \in SOL_{\mathcal{P}}(x)$, tada M računa $m_{\mathcal{P}}(x, y)$ i provjerava vrijedi li $m_{\mathcal{P}}(x, y) \geq k$;

4. ako je prethodna provjera pozitivna, tada M završava u stanju prihvaćanja, inače u stanju odbijanja

Lako je provjeriti da za svaki od ovih koraka Turingov stroj koristi polinomno vrijeme izvršavanja. □

Sada ćemo definirati optimizacijske probleme iz klase NPO za čije rješavanje postoji polinomno vremenski složen algoritam.

Definicija 2.2. *Optimizacijski problem $\mathcal{P}$ pripada klasi PO ako pripada klasi NPO i postoji polinomno vremenski složen algoritam $\mathcal{A}$ takav da, za svaku instancu $x \in I_{\mathcal{P}}$, vraća optimalno rješenje $y \in SOL^{\star}_{\mathcal{P}}(x)$ zajedno s pripadnom mjerom $m^{\star}_{\mathcal{P}}(x)$.*

Problem MINIMALNI PUT u kojem za dana 2 vrha u grafu tražimo minimalni put između njih pripada klasi PO. Skoro svi zanimljivi optimizacijski problemi pripadaju klasi NPO. Problemi MINIMALNI POKRIVAČ VRHOVA, MINIMALNO BOJENJE GRAFA i problem TRGOVAČKOG PUTNIKA pripadaju klasi NPO. Toj klasi pripadaju mnogi drugi optimizacijski problemi na grafovima, problemi pakiranja i rasporeda te problemi cjelobrojnog i binarnog linearnog programiranja. Za te probleme nije poznato pripadaju li klasi PO jer nije poznat nijedan polinomno vremenski složen algoritam koji ih rješava. Pitanje PO = NPO je ekvivalentno pitanju P = NP odnosno može se dokazati da pozitivan odgovor na jedno pitanje povlači pozitivan odgovor na drugo pitanje i obrnuto.

Definicija 2.3. *Za optimizacijski problem $\mathcal{P}$ kažemo da je NP-težak ako za svaki problem odluke $\mathcal{P}' \in NP$ vrijedi $\mathcal{P}' \leq^p_T \mathcal{P}_D$, pri čemu je $\mathcal{P}_D$ pripadajući problem odluke za problem $\mathcal{P}$. To znači da za problem $\mathcal{P}'$ postoji polinomno vremenski složen algoritam s prorokom za problem $\mathcal{P}_D$ takav da rješava problem $\mathcal{P}'$.*

Prethodna definicija vrijedi i za NP-teške probleme odluke ako umjesto Turingove redukcije iskoristimo Karpovu redukciju.

Zaključujemo da je optimizacijski problem NP-težak ako mu je vremenska složenost barem kao svakom problemu iz klase NP. Kao posljedica NP-potpunosti, da bismo dokazali da je optimizacijski problem $\mathcal{P}$ NP-težak, dovoljno je pokazati da vrijedi $\mathcal{P}' \leq^p_T \mathcal{P}_D$ za neki NP-potpun problem $\mathcal{P}'$.

Teorem 2.3. *Neka je problem problem $\mathcal{P} \in NPO$. Ako je pripadajući problem odluke $\mathcal{P}_D$ NP-potpun, tada je $\mathcal{P}$ NP-težak.*

Dokaz. Kako je $\mathcal{P}_D$ NP-potpun, vrijedi $\mathcal{P}' \leq^p_m \mathcal{P}_D$ za svaki $\mathcal{P}' \in$ NP. Karpova redukcija je specijalan slučaj Turingove redukcije i vrijedi da je relacija $\leq^p_T$ tranzitivna pa slijedi da je $\mathcal{P}' \leq^p_T \mathcal{P}$. □

Korolar 2.4. *Ako je $PO = NPO$, tada vrijedi $P = NP$.*

Primjeri NP-teških problema su MINIMALNI POKRIVAČ VRHOVA, MINIMALNO BOJENJE GRAFA i problem TRGOVAČKOG PUTNIKA. Prethodni rezultati nam daju neke odnose između problema odluke i optimizacijskih problema. Sada ćemo proučiti odnose između tri različita pristupa optimizacijskom problemu. Prije smo spomenuli da su to evaluacijski i konstruktivni problem te problem odluke.

Teorem 2.5. *Neka je $\mathcal{P} \in NPO$. Tada vrijedi : $\mathcal{P}_D \equiv_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C$.*

Dokaz. Pokažimo prvo da vrijedi $\mathcal{P}_D \leq_T^p \mathcal{P}_E$. Neka je $x \in I_{\mathcal{P}}$ i $k \in \mathbb{Z}^+$. Algoritam koji rješava problem odluke $\mathcal{P}_D$ može iskoristiti proroka za problem $\mathcal{P}_E$ da izračuna vrijednost $m^\star(x)$ i usporediti tu vrijednost s k.

Slično je za relaciju $\mathcal{P}_E \leq_T^p \mathcal{P}_C$. Neka je sada $x \in I_{\mathcal{P}}$. Algoritam koji rješava problem $\mathcal{P}_E$ može iskoristiti proroka za problem $\mathcal{P}_C$ da dobije vrijednost $m^\star(x)$, što je rješenje problema $\mathcal{P}_E$.

Još nam preostaje za dokazati relacija $\mathcal{P}_E \leq_T^p \mathcal{P}_D$. Kako $\mathcal{P}$ pripada klasi NPO, za svaki $x \in I_{\mathcal{P}}$ i $y \in SOL(x)$, vrijednost $m_{\mathcal{P}}(x, y)$ je ograničena s $M = 2^{p(|x|)}$, za neki polinom p. Stoga, koristeći binarno pretraživanje, evaluacijski problem može biti riješen koristeći najviše $log(M) = p(|x|)$ poziva proroka za problem $\mathcal{P}_D$. □

Danas nije poznato postoji li općenito polinomno vremenski složena Turingova redukcija konstruktivnog problema $\mathcal{P}_C$ na evaluacijski problem $\mathcal{P}_E$. Sljedeći teorem pokazuje da kada je za neki NPO problem njegov odgovarajući problem odluke NP-potpun, konstruktivni i evaluacijski problem, te problem odluke su ekvivalentni.

Teorem 2.6. *Neka je $\mathcal{P}$ neki NPO problem čiji odgovarajući problem odluke $\mathcal{P}_D$ je NP-potpun. Tada vrijedi: $\mathcal{P}_C \leq_T^p \mathcal{P}_D$.*

Dokaz. Radi određenosti, uzmimo da je $\mathcal{P}$ problem maksimizacije. Da bismo dokazali teorem, naći ćemo NPO problem $\mathcal{P}'$ takav da vrijedi $\mathcal{P}_C \leq_T^p \mathcal{P}'_D$. Tada koristeći činjenicu da je problem $\mathcal{P}_D$ jedan NP-potpun problem imamo da vrijedi $\mathcal{P}'_D \leq_T^p \mathcal{P}_D$ (slijedi iz $\mathcal{P}'_D \leq_m^p \mathcal{P}_D$).

Problem $\mathcal{P}'$ definiramo kao problem $\mathcal{P}$ s jedinom razlikom u mjernoj funkciji. Mjerna funkcija $m_{\mathcal{P}'}$ je definirana na način koji ćemo sada opisati. Neka je p polinom iz definicije NPO problema koji je gornja međa za $|y|$ pri čemu su $y \in SOL_{\mathcal{P}}(x)$ i $x \in I_{\mathcal{P}}$. S $\lambda(y)$ označimo redno mjesto rješenja y u leksikografskom poretku svih rješenja (kriterij za poredak je vrijednost mjerne funkcije $m_{\mathcal{P}}$). Za bilo koju instancu $x \in I_{\mathcal{P}'} = I_{\mathcal{P}}$ i rješenje $y \in SOL_{\mathcal{P}'}(x) = SOL_{\mathcal{P}}(x)$ definiramo $m_{\mathcal{P}'}(x, y) = 2^{p(|x|)+1} m_{\mathcal{P}}(x, y) + \lambda(y)$.

Primijetimo da, za svaku instancu x problema $\mathcal{P}'$ i par različitih rješenja $y_1, y_2 \in SOL_{\mathcal{P}'}(x)$, vrijedi $m_{\mathcal{P}'}(x, y_1) \neq m_{\mathcal{P}'}(x, y_2)$. Slijedi da skup rješenja dobivenih od instance x možemo sortirati uzlazno po vrijednosti mjerne funkcije $m_{\mathcal{P}'}$ pri čemu za svaka

dva rješenja vrijedi $m_{\mathcal{P}'}(x, y_i) < m_{\mathcal{P}'}(x, y_{i+1})$. Ovisno radi li se o problemu minimizacije ili maksimizacije, uzmemo minimalni ili maksimalni element iz tog skupa. Taj element je jedinstven pa stoga postoji jedinstveno optimalno rješenje $y \in SOL^{\star}_{\mathcal{P}'}(x)$. Iz nejednakosti $m_{\mathcal{P}'}(x, y_1) > m_{\mathcal{P}'}(x, y_2)$ slijedi nejednakost $m_{\mathcal{P}}(x, y_1) \geq m_{\mathcal{P}}(x, y_2)$ i stoga imamo da je $y \in SOL^{\star}_{\mathcal{P}}(x)$. Ovdje možemo zaključiti da vrijedi $\mathcal{P}_C \leq^p_T \mathcal{P}'_C$.

Optimalno rješenje $y \in SOL^{\star}_{\mathcal{P}'}(x)$ za neku instancu $x \in I_{\mathcal{P}'}$ može se dobiti u polinomnom vremenu korištenjem algoritma s prorokom za problem $\mathcal{P}'_E$. Tada za dobivenu vrijednost $m^{\star}_{\mathcal{P}'}(x)$, računanjem ostatka pri dijeljenju $m^{\star}_{\mathcal{P}'}(x)$ s $2^{p(|x|)+1}$ dobivamo $\lambda(y)$. Iz te vrijednosti iščitamo poziciju rješenja y u leksikografskom poretku rješenja pa i samo rješenje y. Sada slijedi relacija $\mathcal{P}'_C \leq^p_T \mathcal{P}'_E$.

Znamo da vrijedi $\mathcal{P}'_E \leq^p_T \mathcal{P}'_D$. Kako je $\mathcal{P}'_D \in NP$ i $\mathcal{P}_D$ je NP-potpun iz pretpostavke, slijedi da je $\mathcal{P}'_D \leq^p_T \mathcal{P}_D$. Koristeći tranzitivnost Turingove redukcije i relacije koje smo dokazali, vrijedi $\mathcal{P}_C \leq^p_T \mathcal{P}_D$. □

Još uvijek je otvoreno pitanje postoji li NPO problem $\mathcal{P}$ čiji odgovarajući konstruktivni problem $\mathcal{P}_C$ je "teži" od evaluacijskog problema $\mathcal{P}_E$.

Poglavlje 3

Dizajnerske tehnike za aproksimacijske algoritme

U prethodnom poglavlju spomenuli smo neke optimizacijske probleme koji su NP-teški. Ne zna se postoji li efikasan (polinomno vremenski složen) algoritam koji ih rješava. U takvim slučajevima pokušavamo pronaći algoritam koji nalazi rješenje približno optimalnom rješenju. Ako imamo zadanu instancu x optimizacijskog problema $\mathcal{P}$, za dopustivo rješenje $y \in SOL_{\mathcal{P}}(x)$ kažemo da je aproksimacijsko rješenje. Pritom algoritam koji uvijek daje dopustivo rješenje nazivamo aproksimacijski algoritam.

Uočimo da u većini slučajeva nije teško naći polinomno vremenski složen aproksimacijski algoritam koji daje trivijalno rješenje. Primjerice, za problem MINIMALNO BOJENJE GRAFA dovoljno je obojati svaki vrh različitom bojom i dobili bismo jedno dopustivo rješenje. Međutim, nas zanimaju algoritmi koji daju rješenja koja nisu jako "daleka" od optimalnih rješenja.

U ovom poglavlju opisat ćemo najpoznatije tehnike dizajniranja aproksimacijskih algoritama. Osvrnut ćemo se na dizajniranje efikasnih algoritama za NP-teške probleme. U svakoj točki ovog poglavlja razmatrat ćemo jednu od algoritamskih strategija koja se koristi u dizajnu aproksimacijskih algoritama. Svaka od tih strategija može se primijeniti na mnogo problema. Primijenit ćemo te strategije na neke od poznatih problema. Pritom ćemo za svako rješenje dobiveno nekom od strategija dati relaciju između tog rješenja i optimalnog rješenja.

3.1 Pohlepna metoda

Prvi način nalaženja algoritma za određeni problem bit će tzv. pohlepna metoda. Ta metoda se bazira na odabiru lokalno optimalnog rješenja u svakom koraku algoritma nadajući se da će na taj način doći do globalnog optimuma. Na početku se svi kandidati instance (npr. bridovi grafa) sortiraju po određenom kriteriju i zatim se "gradi" skup S (na početku je prazan) od kandidata instance. Pritom imamo funkciju koja provjerava je li skup S izabranih kandidata rješenje zadanog problema. Ta funkcija daje odgovor *da* ili *ne* na pitanje je li odabrani skup S rješenje problema (npr. funkcija provjerava je li skup bridova put između dva zadana vrha u grafu). Još imamo funkciju cilja koja daje vrijednost svakog rješenja problema. To je funkcija koju optimiziramo (maksimiziramo ili minimiziramo). U svakom koraku algoritma promatra se jedan kandidat instance. Ovisno o tome zadovoljava li lokalni uvjet optimalnosti, odnosno ako je najbolji izbor u tom trenutku, taj se element dodaje ili ne dodaje u skup S. Odluka hoće li element biti dodan u skup S ovisi samo o elementima koji su dodani u taj skup, tj. ne ovisi o elementima koje nismo dosad promatrali. Različita rješenja možemo dobiti ovisno o kriteriju po kojem su elementi instance na početku sortirani. Ova se metoda koristi za rješavanje problema optimizacije. Primijenit ćemo tu metodu na 0 - 1 PROBLEM RUKSAKA, problem MAKSIMALAN NEZAVISAN SKUP i problem TRGOVAČKOG PUTNIKA.

3.1.1 Pohlepni algoritam za 0 - 1 PROBLEM RUKSAKA

Zadano je n predmeta pri čemu svaki predmet ima svoju cijenu p_i i težinu w_i. Treba odabrati predmete i staviti ih u ruksak tako da bude maksimalna moguća cijena odabranih predmeta, a da pritom njihova težina ne prelazi težinu M koju ruksak može podnijeti. Za algoritam je prvo potrebno sortirati predmete po padajućem poretku, a kriterij je omjer p_i/w_i. Za sortiranje vremenska složenost je $\mathcal{O}(n \log n)$, a za algoritam odabira predmeta vremenska složenost je $\mathcal{O}(n)$ pa imamo da je ukupna vremenska složenost $\mathcal{O}(n \log n)$. Algoritme za određene probleme navodit ćemo u obliku pseudokoda. U pseudokodu ćemo koristiti standardne naredbe while, for, if i ostale koje su sastavni dio svakog programerskog koda. Sada navodimo pohlepni algoritam za opisani problem:

Input : Skup X koji sadrži n predmeta, za svaki $x_i \in$ X zadane su vrijednosti p_i, w_i i prirodni broj M;
Output: Podskup $Y \subseteq X$ takav da je $\sum_{x_i \in Y} w_i \leq M$;
begin
sortiraj po padajućem poretku predmete iz X po omjeru p_i/w_i
(neka je sada $(x_1, x_2, \ldots, x_n)$ sortirani niz)
$Y := \emptyset$;
for $i := 1$ *to* n **do**
if $M \geq w_i$ **then**
begin
$Y := Y \cup \{x_i\}$;
$M := M - w_i$
end
end
end
return Y
end

Sljedeći teorem pokazuje odnos između rješenja dobivenog pohlepnim algoritmom i optimalnog rješenja.

Teorem 3.1. *Neka je s $m_{Gr}(x)$ dana mjera za rješenje koje algoritam pronađe ako se primijeni na instancu problema x i neka je p_{max} najveća vrijednost među svim predmetima iz instance problema x. Sada za danu instancu x problema 0 - 1 PROBLEM RUKSAKA, neka je $m_H(x) = \max(p_{max}, m_{Gr}(x))$. Tada $m_H(x)$ zadovoljava sljedeću nejednakost:*

$$m^\star(x)/m_H(x) < 2.$$

Dokaz. Neka je j redni broj prvog predmeta koji nije stavljen u ruksak gore navedenim pohlepnim algoritmom s ulazom $x = (x_1, \ldots, x_n)$. Trenutni profit u ruksaku tada je:

$$\overline{p_j} = \sum_{i=1}^{j-1} p_i \leq m_{Gr}(x)$$

i ukupna težina predmeta u ruksaku je:

$$\overline{a_j} = \sum_{i=1}^{j-1} a_i \leq b.$$

Prvo ćemo pokazati da bilo koje optimalno rješenje dane instance mora zadovoljavati sljedeću nejednakost: $m^\star(x) < \overline{p_j} + p_j$. Imamo da su predmeti sortirani silazno

po omjeru p_i/a_i pa slijedi da zamjenom bilo kojeg podskupa odabranih predmeta $x_1, \ldots, x_{j-1}$ s bilo kojim podskupom neodrabranih predmeta $x_j, \ldots, x_n$, koji neće povećati sumu $\overline{a_j}$, također neće povećati ni ukupni profit $\overline{p_j}$ u ruksaku. Stoga je optimalno rješenje $m^\star(x)$ s donje stranje ograničeno zbrojem $\overline{p_j}$ i maksimalnog profita koji se može dobiti popunjavanjem ostatka slobodnog mjesta u ruksaku s predmetima čiji je omjer cijene i težine najviše p_j/a_j. Slobodnih mjesta ima još $b - \overline{a_j}$. Vrijedi $\overline{a_j} + a_j > b$ odnosno $a_j > b - \overline{a_j}$ pa stoga slijedi: $m^\star(x) \leq \overline{p_j} + (b - \overline{a_j})p_j/a_j < \overline{p_j} + p_j$.

Sada promatramo dva slučaja. Ako je $p_j \leq \overline{p_j}$, tada vrijedi:

$$m^\star(x) < 2\overline{p_j} \leq 2m_{Gr}(x) \leq 2m_H(x).$$

Ako je $p_j > \overline{p_j}$, tada je $p_{max} > \overline{p_j}$. U tom slučaju vrijedi:

$$m^\star(x) \leq \overline{p_j} + p_j \leq \overline{p_j} + p_{max} < 2p_{max} \leq 2m_H(x).$$

□

3.1.2 Pohlepni algoritam za problem MAKSIMALAN NEZAVISAN SKUP

Prvo ćemo definirati navedeni problem:

1. Instanca problema: Graf $G = (V, E)$
2. Skup rješenja: podskup $V' \subseteq V$ takav da, za svaki $(u, v) \in E$ ili $u \notin V'$ ili $v \notin V'$.
3. Mjera: kardinalnost nezavisnog skupa V', tj $|V'|$.
4. Maksimizacijski problem

Ako želimo iskoristiti pohlepni algoritam za ovaj problem, potrebno je sortirati vrhove po određenom kriteriju. Taj kriterij bit će stupanj vrha. U poretku će prvi biti vrhovi manjeg stupnja pa vrhovi većeg stupnja. Kad dodajemo određeni vrh u skup rješenja, svi njegovi susjedi se ne moraju promatrati u sljedećim koracima. Zaključujemo da odabirom vrhova manjeg stupnja, dolazimo do mogućeg maksimalnog nezavisnog skupa (ne mora nužno biti maksimalan jer je pohlepni algoritam).

Algoritam je dan na sljedeći način:

Input : Graf $G = (V, E)$
Output: nezavisan skup $V' \subseteq V$;
begin
  $V' := \emptyset$;
  $U := V$;
  while U *nije prazan* **do**
    begin
      $x :=$ vrh minimalnog stupnja u grafu induciranim skupom U;
      $V := V' \cup \{x\}$;
      izbaci x i sve susjede od x iz U
    end
  end
  return V'
end

Za dani graf G s n vrhova i m bridova, neka je $\delta = m/n$. Prethodni algoritam s ulazom G, nalazi nezavisan skup kardinalnosti $m_{Gr}(x)$ takav da vrijedi:

$$m^\star(G)/m_{Gr}(G) \leq \delta + 1.$$

Dokaz ove tvrdnje može se pronaći u [1].

3.1.3 Pohlepni algoritam za problem TRGOVAČKOG PUTNIKA

Instanca ovog problema može se poistovjetiti s potpunim grafom $G = (V, E)$ s pozitivnim težinama na bridovima. Dopustiva rješenja problema su svi podskupovi $I \subseteq E$ takvi da je graf (V, I) ciklus. Ideja pohlepnog algoritma je prvo pronaći Hamiltonov put (put na grafu koji prolazi kroz svaki vrh točno jednom) i zatim povezati bridom zadnji i prvi vrh tog puta.
Da bismo našli Hamiltonov put, prvo izaberemo proizvoljni vrh (grad) c_{i_1} u grafu koji će biti početak puta. Zatim izvršavamo $n-1$ iteracija petlje koju imamo u algoritmu, pri čemu je n broj vrhova u grafu. U prvoj iteraciji izaberemo vrh c_{i_2} takav da je težina brida (duljina puta) između c_{i_1} i c_{i_2} minimalna među svim bridovima kojima je jedan vrh c_{i_1}. Tada brid (c_{i_1}, c_{i_2}) dodamo u put koji je na početku prazan i sad ga "gradimo". U r-toj iteraciji petlje kada imamo put $(c_{i_1}, c_{i_2}, \ldots, c_{i_r})$, izabiremo vrh $c_{i_{r+1}}$ takav da je brid $(c_{i_r}, c_{i_{r+1}})$ minimalne težine među svim bridovima kojima je jedan vrh c_{i_r}, a drugi nije dio puta $(c_{i_1}, c_{i_2}, \ldots, c_{i_r})$. Sada vrh $c_{i_{r+1}}$ dodamo putu $(c_{i_1}, c_{i_2}, \ldots, c_{i_r})$. Nakon $n-1$ iteracija, Hamiltonov put $(c_{i_1}, c_{i_2}, \ldots, c_{i_n})$ je konstruiran. Još samo treba izvršiti zadnji korak algoritma, treba dodati brid (c_{i_n}, c_{i_1}) Hamiltonovom putu da bi bio zatvoren ciklus. Kako je odgovarajući podgraf

sastavljen na prethodni način potpun, algoritam je sigurno pronašao jedno dopustivo rješenje. Prethodni algoritam zovemo još **algoritam najbližeg susjeda** (eng. *Nearest neighbor*) jer u svakom koraku tražimo susjeda koji je najbliži danom vrhu, čime smo potkrijepili pohlepnost algoritma. Danas nije poznata precizna relacija koja bi opisala kvalitetu rješenja dobivenog prethodnim algoritmom u usporedbi s optimalnim rješenjem.

3.2 Sekvencijalni algoritmi za problem particije

U ovoj točki obradit ćemo probleme particije odnosno probleme čija su dopustiva rješenja particije skupa kandidata (elemenata) instance $I = \{x_1, x_2, \ldots, x_n\}$. Sekvencijalni algoritam za problem particije prvo sortira kandidate instance po određenom kriteriju. Zatim algoritam "gradi" particiju sekvencijalno. Opća shema algoritama ovog tipa izgleda ovako:

Input : skup kandidata instance I
Output: particija P skupa I
begin
  sortiraj elementa skupa I (po određenom kriteriju)
  (neka je $(x_1, x_2, \ldots, x_n)$ sortirani skup I)
  $P := \{\{x_1\}\}$;
  for $i := 2$ *to* n **do**
    if x_i *se može dodati određenom skupu* p *u particiji* P **then**
      dodaj x_i u p
    else
      $P := P \cup \{\{x_i\}\}$;
    end
  end
  return P
end

Primijetimo da u prethodnom algoritmu, kad promatramo element x_i sortiranog skupa I, ne mogu se elementi $x_j, j < i$ prebacivati u druge podskupove particije P.

Da bismo dizajnirali sekvencijalni algoritam za određeni problem particije, prvo trebamo odrediti kriterij po kojem ćemo sortirati elemente instance, a zatim kriterij po kojem ćemo elemente dodavati u particiju. Kao u slučaju pohlepnih algoritama, postoji sigurno jedan kriterij po kojem bi sortirali elemente instance, a da pritom

daje optimalno rješenje. Međutim, unaprijed ne znamo koji bi to kriterij mogao biti pa uzimamo kriterij koji bi mogao dati približno dobra aproksimacijska rješenja.

U sljedećim točkama analizirat ćemo sekvencijalne algoritme za probleme MINIMALNI RASPORED POSLOVA NA JEDNAKE STROJEVE, MINIMALNO GRUPIRANJE i MINIMALNO BOJENJE GRAFA.

3.2.1 Raspored poslova na jednake strojeve

Prvo definiramo problem MINIMALNI RASPORED POSLOVA NA JEDNAKE STROJEVE:

1. Instanca problema: skup poslova T, p jednakih strojeva, duljina izvođenja l_j svakog posla $t_j \in T$
2. Skup rješenja: raspored poslova iz T na p strojeva, tj. funkcija $f : T \to [1 \dots p]$
3. Mjera: $\max\limits_{i \in [1 \dots p]} \sum\limits_{t_j \in T : f(t_j) = i} l_j$
4. Minimizacijski problem

Cilj je u ovoj točki navesti algoritam za problem rasporeda određenog broja poslova p na jednake strojeve tako da vrijeme izvršavanja svih poslova bude minimalno. Pritom se na svakom stroju prvo izvršava jedan posao, a onda drugi, odnosno serijski se izvršavaju. Ovaj problem je NP-težak čak i u slučaju kad je $p = 2$.

Pretpostavimo da je poznat poredak po kojem ćemo dodjeljivati poslove strojevima. Sada nam je potreban kriterij po kojem će algoritam dodjeljivati neki posao određenom stroju. Najjedostavnije rješenje je dodijeliti posao stroju koji trenutno ima najkraće vrijeme izvršavanja svih poslova koji su pridodani njemu. Stoga uzmimo da je prvih $j-1$ poslova dodijeljeno određenim strojevima i neka je s $A_i(j-1)$ označeno ukupno vrijeme izvršavanja poslova pridijeljenih i-tom stroju (tj. $A_i(j-1) = \sum_{1 \le k \le j-1 : f(t_k) = i} l_k$). Tad j-ti posao pridodajemo stroju s minimalnim vremenom izvršavanja odnosno minimalnom vrijednošću $A_i(j-1)$. Ovaj algoritam nazivamo **algoritam rasporeda popisa** (eng. *List scheduling*). Sada slijedi teorem koji pokazuje odnos između rješenja problema dobivenog ovim algoritmom i optimalnog rješenja.

Teorem 3.2. *Za danu instancu x problema MINIMALNI RASPORED POSLOVA NA JEDNAKE STROJEVE s p strojeva, za bilo koji poredak poslova po kojem ćemo ih pridruživati strojevima, algoritam rasporeda popisa nalazi rješenje s mjerom $m_{RP}(x)$ takvom da vrijedi:*

$$m_{RP}(x)/m^\star(x) \le 2 - \frac{1}{p}.$$

Dokaz. Označimo s W sumu vremena izvršavanja svih poslova, tj. $W = \sum_{k=1}^{|T|} l_k$. Mjera optimalnog rješenja zadovoljava sljedeću nejednakost: $m^\star(x) \geq W/p$. Raspored pridruživanja poslova strojevima nije bitan. Strojevi imaju redne brojeve od 1 do $|T|$.

Sada želimo ograničiti mjeru $m_{RP}(x)$ dobiveno algoritmom rasporeda popisa. Neka je h-ti po redu stroj onaj koji ima najduže ukupno vrijeme izvršavanja poslova pridruženih njemu, tj. $A_h(|T|) = m_{RP}(x)$, i neka je j redni broj zadnjeg posla pridruženog tom stroju. Kako je posao t_j pridružen stroju koji trenutno ima najmanje ukupno vrijeme izvršavanja pridruženih poslova, tada je ukupno vrijeme izvršavanja na ostalim strojevima barem $A_h(|T|) - l_j$. Stoga vrijedi $W \geq p(A_h(|T|) - l_j) + l_j$ i nejednakost za mjeru $m_{RP}(x)$:

$$m_{RP}(x) = A_h(|T|) \leq \frac{W}{p} + \frac{(p-1)l_j}{p}.$$

Kako vrijedi $m^\star(x) \geq W/p$ te $m^\star(x) \geq l_j$, slijedi:

$$m_{RP}(x) \leq \frac{W}{p} + \frac{(p-1)l_j}{p} \leq m^\star(x) + \frac{p-1}{p} m^\star(x) = (2 - \frac{1}{p}) m^\star(x).$$

□

3.2.2 Sekvencijalni algoritmi za problem MINIMALNO GRUPIRANJE

Definiramo problem MINIMALNO GRUPIRANJE:

1. Instanca problema: konačni multiskup I racionalnih brojeva $\{a_1, a_2, \ldots, a_n\}$ pri čemu vrijedi $a_i \in <0, 1]$ za $i = 1, \ldots, n$
2. Skup rješenja: particija $\{B_1, B_2, \ldots, B_k\}$ skupa I tako da vrijedi $\sum_{a_i \in B_j} a_i \leq 1$ za $j = 1, \ldots, k$
3. Mjera: kardinalost particije, tj. k
4. Minimizacijski problem

Gornji problem se može poistovijetiti s problemom iz svakodnevnog života. Imamo određeni broj predmeta koji imaju svoju težinu i pritom sve težine ne prelaze određenu granicu (težina između 0 i 1). Također imamo sanduke u koje treba te predmete staviti, a pritom i svaki sanduk ima određenu nosivost predmeta (nosivost je 1). Problem je sličan problemu ruksaka.

Jednostavan sekvencijalan algoritam za ovaj problem je **algoritam popunjavanja sljedećeg** (eng. *Next fit*). Algoritam uzima elemente instance i razmatra ih onim poretkom kojim su dani na početku, tj. nema sortiranja po određenom kriteriju. Prvi element (predmet) a_1 stavimo u sanduk B_1. Neka je sada B_j zadnji iskorišteni predmet kada algoritam razmatra predmet a_i. Sada po algoritmu predmet a_i ide u sanduk B_j ako u tom sanduku ima dovoljno mjesta, tj. ako dodamo a_i u B_j nećemo preskočiti granicu nosivosti sanduka. Ako nema dovoljno mjesta, predmet ide u sanduk B_{j+1}. Sljedeći teorem pokazuje odnos između rješenja dobivenog navedenim algoritmom i optimalnog rješenja.

Teorem 3.3. *Za danu instancu x problema MINIMALNO GRUPIRANJE, algoritam popunjavanja sljedećeg nalazi rješenje s mjerom $m_{SP}(x)$ takvo da vrijedi:*

$$m_{SP}(x)/m^\star(x) < 2.$$

Dokaz. Označimo s A sumu težina svih predmeta, tj. $A = \sum_{i=1}^{n} a_i$. Ako uzmemo rješenje dobiveno algoritmom, znamo da za svaki par uzastopnih sanduka koje smo odabrali vrijedi da je suma težina predmeta u njima veća od 1. Stoga vrijedi da je broj iskorištenih sanduka manji od $2\lceil A \rceil$.

S druge strane, kako je broj iskorištenih sanduka u svakom dopustivom rješenju jednak najmanje sumi težina svih predmeta, slijedi da je $m^\star(x) \geq \lceil A \rceil$. Sada slijedi nejednakost $m_{SP}(x) < 2m^\star(x)$.

□

Nedostatak gornjeg algoritma je da predmet koji razmatra pokušava staviti samo u zadnji sanduk zanemarujući sanduke prije njega. Zato navodimo novi algoritam koji nazivamo **algoritam popunjavanja prvog** (eng. *First fit*). U tom algoritmu predmet a_i se stavlja u prvi sanduk koji ima dovoljno mjesta za njega odnosno prvi koji će izdržati još i njegovu težinu. Ako takav sanduk ne postoji, dodajemo novi i u njega stavljamo predmet a_i.

Algoritam popunjavanja prvog je bolji od *algoritma popunjavanja sljedećeg* što pokazuje rezultat: ako je s $m_{PP}(x)$ dana mjera koja se postiže za rješenje koje nalazi *algoritam popunjavanja prvog*, tada vrijedi:

$$m_{PP}(x) \leq 1.7m^\star(x) + 2.$$

Dokaz ove tvrdnje može se pronaći u [1].

Još bolji algoritam za problem MINIMALNO GRUPIRANJE je **algoritam popunjavanja prvog s padajućim poretkom** (eng. *First fit decreasing*). Ovaj algoritam prvo sortira elemente instance (predmete u našem slučaju) u padajućem

poretku po njihovim težinama. Nakon sortiranja algoritam postupa isto kao *algoritam popunjavanja prvog*.

Može se pokazati da za danu instancu x problema MINIMALNO GRUPIRANJE, *algoritam popunjavanja prvog s padajućim poretkom* nalazi rješenje s mjerom $m_{PPP}(x)$ takvom da vrijedi:

$$m_{PPP}(x) \leq 1.5m^{\star}(x) + 1.$$

Dokaz ove tvrdnje može se pronaći u [1].

Postoji i bolji algoritam od prethodnog za problem MINIMALNO GRUPIRANJE. Riječ je o **algoritmu popunjavanja najboljeg s padajućim poretkom** (eng. *Best fit decreasing*). Algoritam prvo sortira predmete po padajućem poretku uzevši u obzir njihove težine. Razlika u odnosu na *algoritam popunjavanja prvog s padajućim poretkom* je u izboru u koji će se sanduk predmet a_i staviti. Dok se kod *algoritma popunjavanja prvog s padajućim poretkom* predmet stavlja u prvi sanduk koji može podnijeti taj predmet, kod ovog algoritma predmet se pokušava staviti u sanduk s minimalnom preostalom nosivošću. Na ovaj način algoritam pokušava smanjiti fragmentaciju povećavajući broj sanduka koji su popunjeni skoro do maksimuma.

Rješenje dobiveno *algoritmom popunjavanja najboljeg s padajućim poretkom* u nekim slučajevima može biti lošije odnosno ne tako blizu optimalnom rješenju kao rješenje dobiveno *algoritmom popunjavanja prvog s padajućim poretkom*.

S druge strane, za neke instance problema *algoritam popunjavanja najboljeg s padajućim poretkom* nalazi optimalno rješenje dok drugi algoritam ne nalazi.

3.2.3 Sekvencijalni algoritmi za problem MINIMALNO BOJENJE GRAFA

Ovaj problem smo već definirali u prvom poglavlju. Da bismo primijenili sekvencijalni algoritam za ovaj problem, prvo treba odrediti poredak po kojem će algoritam razmatrati vrhove grafa. Pretpostavimo da je $(v_1, v_2, \ldots, v_n)$ dobiveni poredak. Vrh v_1 će biti obojan bojom 1 i zatim ostali vrhovi na način koji ćemo sada opisati. Kada algoritam razmatra vrh v_i i pritom su iskorištene boje $1, 2, \ldots, k$, vrh v_i se pokuša obojiti jednom od boja $1, 2, \ldots, k$. Ako se ne može obojati nijednom od iskorištenih boja, nova boja $k+1$ se uzima i njom se oboji vrh v_i. Algoritam ponavlja prethodni korak sve dok svi vrhovi ne budu obojani.

Koliko je dobro dobiveno rješenje, ovisi o poretku po kojem je algoritam razmatrao vrhove. Mi ćemo razmotriti dva kriterija za sortiranje vrhova. Jedan je kriterij da se vrhovi sortiraju po padajućem poretku u odnosu na njihove stupnjeve. Drugi kriterij je da se vrhovi obrađuju po principu najmanjeg zadnjeg što će biti opisano kasnije.

Analizirat ćemo broj boja koje algoritam treba iskoristiti kao funkciju stupnja vrhova. Naime, za poredak vrhova $(v_1, v_2, \ldots, v_n)$ grafa G, neka je s G_i dan graf induciran vrhovima $\{v_1, v_2, \ldots, v_i\}$. Očito je $G_n = G$. Neka je k_i broj boja koje algoritam treba iskoristiti na grafu G_i kako bi došao do rješenja, a $d_i(v)$ je stupanj vrha v u grafu G_i.

Za bilo koji poredak vrhova u grafu, sekvencijalni algoritam za bojenje grafa koristi najviše $\Delta + 1$ boja da oboji sve vrhove grafa G, gdje je Δ najveći stupanj među svim vrhovima grafa G.

Algoritam koji pri analizi koristi padajući poredak vrhova s obzirom na stupanj vrha naziva se **algoritam padajućeg stupnja** (eng. *Decreasing degree*). Broj boja koje ovaj algoritam iskoristi može biti puno veći nego u optimalnom rješenju. Naime, postoji 2-obojiv graf s $2n$ vrhova i maksimalnim stupnjem vrha jednakim $n - 1$ za koji *algoritam padajućeg stupnja* iskoristi n boja da bi dobio rješenje.

Mnogo je kriterija za početni poredak vrhova po kojem će ih algoritam analizirati, a koji bi poboljšali loše ponašanje prethodnog algoritma. Sljedeći kriterij za poredak vrhova definiran je na sljedeći način: v_n, zadnji vrh koji će algoritam analizirati, je vrh s najmanjim stupnjem u grafu G. Poredak ostalih vrhova dobijemo ovako: nakon što su vrhovi $v_{i+1}, \ldots, v_n$ dodani u poredak, v_i je vrh s najmanjim stupnjem u podgrafu induciranom s $V - \{v_{i+1}, \ldots, v_n\}$. Algoritam koji koristi ovako dobiveni poredak vrhova pri analizi nazivamo **algoritam najmanjeg zadnjeg** (eng. *Smallest last*).

Za broj boja potrebnih sekvencijalnom algoritmu za bojenje grafa vrijedi sljedeća nejednakost:

$$k_n \leq 1 + \max_{1 \leq i \leq n} (d_i(v_i)).$$

Dokaz ove tvrdnje može se pronaći u [1].

Poredak vrhova koji koristi *algoritma najmanjeg zadnjeg* minimizira gornju granicu u prethodnoj nejednakosti jer vrijedi, za svaki $1 \leq 1 \leq n$, $d_i(v_i) = \min_{v_j \in G_i} d_i(v_j)$. Kao i prethodni algoritam, i za ovaj se može pokazati da ne nalazi "dovoljno dobra" rješenja za neke grafove. Međutim na planarnim grafovima nalazi zadovoljavajuća rješenja.

Algoritam najmanjeg zadnjeg oboji (da vrijede uvjeti problema MINIMALNO BOJENJE GRAFA) planarni graf s najviše 6 boja. Iz ove tvrdnje slijedi rezultat:

postoji polinomno vremenski složen algoritam $\mathcal{A}$ za bojenje grafa takav da, primijenjen na planarni graf G, pronalazi rješenje s mjerom $m_{\mathcal{A}}(G)$ tako da vrijedi:

$$m_{\mathcal{A}}(G)/m^{\star}(G) \leq 2.$$

Dokaz ove tvrdnje može se pronaći u [1].

3.3 Lokalno pretraživanje

Algoritmi lokalnog pretraživanja počinju s nekim početnim rješenjem problema (dobivenog nekim drugim algoritmom) i iterativno poboljšavaju trenutno rješenje preko boljeg "susjednog" rješenja.

Grubo govoreći, ako je y dopustivo rješenje problema, tada je z susjedno rješenje ako se ne razlikuje značajno od rješenja y. Naprimjer, ako se radi o problemu POKRIVAČ VRHOVA i poznat je jedan pokrivač vrhova kao rješenje problema, tada je susjedno rješenje također pokrivač vrhova koji se razlikuje u jednom vrhu od prethodnog rješenja.

Za dopustivo rješenje y, algoritmi lokalnog pretraživanja traže susjedno rješenje s boljom mjerom. Kada nema boljeg susjednog rješenja od rješenja y, tada algoritam staje u lokalnom optimumu i y se uzima kao rješenje problema.

Da bismo primijenili algoritam lokalnog pretraživanja na određenom problemu, prvo je potreban algoritam koji će pronaći početno dopustivo rješenje problema i strukturu susjedstva za bilo koje dopustivo rješenje. U većini slučajeva to će biti trivijalno što ćemo pokazati u sljedećem primjeru.

3.3.1 Algoritmi lokalnog pretraživanja za problem MAKSIMALNI REZ

U ovoj točki dat ćemo algoritam lokalnog pretraživanja za problem MAKSIMALNI REZ:

1. Instanca problema: graf $G = (V, E)$
2. Skup rješenja: particija skupa V na disjunktne skupove V_1 i V_2
3. Mjera: kardinalnost reza, tj. broj bridova s jednim vrhom u skupu V_1 i drugim vrhom u skupu V_2
4. Maksimizacijski problem

Prvo trebamo definirati na koji način ćemo zadati početno dopustivo rješenje i strukturu susjedstva svakog dopustivog rješenja. U slučaju ovog problema, odabir početnog rješenja je trivijalan: $V_1 = \emptyset$ i $V_2 = V$ je dopustivo rješenje. Nadalje definiramo strukturu susjedstva $\mathcal{N}$: susjedstvo rješenja (V_1, V_2) sastoji se od particija V_{1k}, V_{2k}, za $k = 1, 2, \ldots, |V|$, tako da vrijedi:

1. ako vrh $v_k \in V_1$, tada:

$$V_{1k} = V_1 / \{v_k\} \quad i \quad V_{2k} = V_2 \cup \{v_k\};$$

2. ako vrh $v_k \notin V_1$, tada:

$$V_{1k} = V_1 \cup \{v_k\} \quad i \quad V_{2k} = V_2/\{v_k\};$$

Bitno svojstvo ove strukture susjedstva je da svako lokalno optimalno rješenje ima mjeru koja je po vrijednosti barem pola optimalnog rješenja. Preciznije rečeno, za danu instancu x problema MAKSIMALNI REZ, neka je (V_1, V_2) lokalni optimum s obzirom na strukturu susjedstva $\mathcal{N}$ i neka je $m_{\mathcal{N}}(x)$ mjera za taj optimum. Tada vrijedi:

$$m^\star(x)/m_{\mathcal{N}}(x) \leq 2.$$

Dokaz ove tvrdnje može se pronaći u [1].

3.3.2 Algoritmi lokalnog pretraživanja za problem TRGOVAČKOG PUTNIKA

Nažalost, za većinu optimizacijskih problema prethodna nejednakost za odnos rješenja dobivenog ovim algoritmom i optimalnog rješenja ne vrijedi. Međutim algoritmi lokalnog pretraživanja su vrlo praktični za primjenu kod većine optimizacijskih problema. Naprimjer, u slučaju problema TRGOVAČKOG PUTNIKA postoje algoritmi lokalnog pretraživanja koji pronalaze vrlo dobra aproksimacijska rješenja u razumnom vremenu za neke vrlo velike instance problema (iako za neke instance algoritam se može loše ponašati).

Početno dopustivo rješenje problema TRGOVAČKOG PUTNIKA je bilo koja permutacija n gradova. Stoga možemo uzeti da je identiteta $(c_1, c_2, \ldots, c_n)$ pri čemu su $c_i, i = 1, \ldots, n$ gradovi, početno rješenje s kojim će algoritam lokalnog pretraživanja početi tražiti bolja rješenja. U drugom slučaju, možemo za početno rješenje uzeti rješenje dobiveno nekim drugim algoritmom (*algoritam najbližeg susjeda*).

Struktura susjedstva za problem TRGOVAČKOG PUTNIKA je tzv. *2-opt* struktura koja je opisana na sljedeći način: za dani obilazak I svih gradova, zamjenom dva brida (x, y) i (v, z) u I s bridovima (x, v) i (y, z) dobivamo novi obilazak I'. Ova zamjena bridova naziva se *2-potez* (eng. *2-move*). *2-opt* struktura susjedstva rješenja I sastoji se od svih rješenja koja se mogu dobiti iz I korištenjem *2-poteza*.

Zapažamo da, prema definiranoj strukturi susjedstva, bilo koje rješenje ima $\mathcal{O}(n^2)$ susjednih rješenja. Stoga traženje boljeg rješenja susjednih trenutnom rješenju zahtjeva najviše $\mathcal{O}(n^2)$ operacija.

Eksperimentalne provjere su pokazale da kvaliteta rješenja dobivenog korištenjem *2-opt* strukture susjedstva ovisi o početno izabranom rješenju.

3.4 Algoritmi bazirani na linearnom programiranju

Linearno programiranje je među najefikasnijim područjima operacijskih istraživanja i primjenjuje se na mnogo problema. To je jednostavniji način da opišemo složenije odnose pomoću linearnih nejednadžbi i linearne funkcije koju treba maksimizirati ili minimizirati.

Problem linearnog programiranja može se riješiti u polinomnom vremenu, međutim za njegovu početnu formulaciju je potrebna veća vremenska složenost ako se radi o nekom složenom optimizacijskom problemu. Najpoznatiji primjer linearnog programiranja je cjelobrojno linearno programiranje.

3.4.1 Zaokruživanje rješenja cjelobrojnog linearnog programiranja

Za dani sustav nejednadžbi koji opisuje problem cjelobrojnog linearnog programiranja (CLP), izbacivanjem uvjeta cjelobrojnosti na varijable, dobivamo novi sustav nejednadžbi čije rješenje možemo dobiti u polinomnom vremenu. Optimalno rješenje dobiveno na ovaj način može se iskoristiti kao početno rješenje za CLP zaokruživanjem vrijednosti varijabli koje ne zadovoljavaju uvjet cjelobrojnosti.

Promotrit ćemo problem MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA kao problem CLP. U odnosu na problem MINIMALNI POKRIVAČ VRHOVA, jedina razlika je što svaki vrh v_i u ovom problemu ima nenegativnu težinu c_i i rješenje problema je pokrivač vrhova s minimalnom težinom.

Za dani težinski graf $G = (V, E)$, problem MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA se može formulirati na sljedeći način:

$$\begin{aligned} &\min \sum_{v_i \in V} c_i x_i \\ &x_i + x_j \geq 1, \quad \forall (v_i, v_j) \in E \\ &x_i \in \{0, 1\}, \quad \forall v_i \in V \end{aligned}$$

Ovako formuliran problem označavat ćemo s $\mathrm{CLP}_{PV}(G)$. Neka je LP_{PV} problem dobiven odbacivanjem uvjeta cjelobrojnosti i uvođenjem uvjeta nenegativnosti ($x_i \geq 0$ za svaki $v_i \in V$). Još s $x^\star(G)$ označimo optimalno rješenje problema LP_{PV}. Sada ćemo navesti algoritam za nalaženje dopustivog rješenja V' za problem MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA. U dopustivo rješenje uključujemo sve vrhove čije vrijednosti iznose barem 0.5 nakon svih koraka algoritma.

Input : graf G = (V,E) s nenegativnim težinama vrhova
Output: pokrivač vrhova V' za G
begin
CLP$_{PV}$ je formulacija problema pomoću cjelobrojnog linearnog programiranja
LP$_{PV}$ je formulacija problema dobivena iz CLP$_{PV}$ izbacivanjem uvjeta cjelobrojnosti varijabli
$x^\star(G)$ je optimalno rješenje problema LP$_{PV}$
$V' := \{v_i | x_i^\star(G) \geq 0.5\}$;
return P
end

Može se pokazati da za dani graf G s nenegativnim težinama vrhova, prethodni algoritam nalazi dopustivo rješenje problema MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA s mjerom $m_{LP}(G)$ takvom da vrijedi:

$$m_{LP}(G)/m^\star(G) \leq 2.$$

Dokaz ove tvrdnje može se pronaći u [1].

3.4.2 Primal-dual algoritmi

Svaki problem linearnog programiranja (primal) ima svoj dualni oblik (dual). Ako je primal (LP) problem maksimizacije, onda je njegov dual (DLP) problem minimizacije, i obrnuto. Formulirat ćemo primal-dual algoritam za problem MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA. Za dani težinski graf $G = (V, E)$, dual prije formuliranog problema LP_{VC} je sljedeći problem DLP_{VC}:

$$\begin{aligned} &\max \sum_{v_i, v_j \in E} y_{ij} \\ &\sum_{j:(v_i,v_j)\in E} y_{ij} \leq c_i, \quad \forall v_i \in V \\ &y_{ij} \geq 0, \quad \forall (v_i, v_j) \in E \end{aligned}$$

U slučaju da su svi y_{ij} nula u problemu DLP_{VC}, tada je to dopustivo rješenje ovog problema s mjerom 0. Primal-dual algoritam počinje s tim rješenjem i konstruira pokrivač vrhova poboljšavajući trenutno dualno rješenje. Sada navodimo primal-dual algoritam za MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA.

Input : graf G = (V,E) s nenegativnim težinama vrhova
Output: pokrivač vrhova V' za G
begin
CLP$_{PV}$ je formulacija problema pomoću cjelobrojnog linearnog programiranja
LP$_{PV}$ je formulacija problema dobivena iz CLP$_{PV}$ izbacivanjem uvjeta cjelobrojnosti varijabli
DLP$_{VC}$ je dual problema LP_{VC}
for *svaka dualna varijabla* y_{ij} **do**
$y_{ij} := 0$
end
$V' := \emptyset$;
while V' *nije pokrivač vrhova za* G **do**
begin
neka je (v_i, v_j) brid za koji ni v_i ni v_j nisu u V'; povećavaj y_{ij} sve dok i i j ne dosegnu maksimume indeksa koji se koriste u uvjetima problema DLP$_{VC}$; **if** $\sum_{k:(v_i,v_k)\in E} y_{ik} = c_i$ **then**
$V' := V' \cup \{v_i\}$
else
$V' := V' \cup \{v_j\}$
end
end
end
return V'
end

Može se pokazati da za dani graf G s nenegativnim težinama, gornji algoritam nalazi dopustivo rješenje problema MINIMALNI POKRIVAČ TEŽINSKIH VRHOVA s mjerom m_{PD} takvom da vrijedi:

$$m_{PD}(G)/m^{\star}(G) \leq 2.$$

Dokaz ove tvrdnje može se pronaći u [1].

3.5 Dinamičko programiranje

Ova algoritamska tehnika se koristi za rješavanje kompleksnih optimizacijskih problema tako da se riješe potproblemi tog problema koji su manje složenosti. Problemi moraju imati optimalnu podstrukturu što znači da se rješenje problema može dobiti kombiniranjem optimalnih rješenja potproblema. Svaki potproblem se riješi samo jednom i to se rješenje spremi da bi se kasnije iskoristilo za veći problem.

Algoritmi dinamičkog programiranja ne generiraju nove potprobleme već se stari rješavaju ponovo. Postoje 2 pristupa u rješavanju problema: **odozgo prema dolje** (eng. *bottom-up*) i **odozdo prema gore** (eng. *bottom-up*) pristup. Kod prvog pristupa problem se rastavlja na potprobleme. Zatim se potproblemi riješe i pamte se njihova rješenja. Ovaj pristup predstavlja kombiniranje rekurzije i memoriranja rješenja. Kod drugog pristupa svi se potproblemi redom rješavanju i koriste za nalaženje većih. Ovaj je pristup bolji zbog štednje memorijskog prostora. Ponekad je teško odrediti koji su sve potproblemi potrebni za traženje rješenja danog problema.

Tipični primjeri primjene dinamičkog programiranja su: izračunavanje Fibonaccijevih brojeva, "matrični problemi", pronalaženje najduže zajedničke podsekvence...

U ovoj točki primijenit ćemo tehniku dinamičkog programiranja na
0 - 1 PROBLEM RUKSAKA. Da bismo primijenili navedenu tehniku, potrebno je definirati potprobleme. Znamo da za instancu 0 - 1 PROBLEMA RUKSAKA imamo kapacitet ruksaka $b \in \mathbb{Z}^+$ i konačan skup X od n predmeta takav da su za svaki predmet $x_i \in X$ poznati cijena $p_i \in \mathbb{Z}^+$ i težina $a_i \in \mathbb{Z}^+$.

Za bilo koji prirodni broj k takav da $1 \leq k \leq n$ i za bilo koji prirodni broj p takav da $0 \leq p \leq \sum_{i=1}^{n} p_i$, promatramo problem nalaženja podskupa skupa $\{x_1, \ldots, x_k\}$ takvog da ima minimalnu težinu, manju ili jednaku b, i ukupnu cijenu jednaku p. S $M^\star(k,p)$ označavat ćemo optimalno rješenje prethodno definiranog problema i s $S^\star(k,p)$ odgovarajuću optimalnu cijenu. Kada $M^\star(k,p)$ nije definirano, uzimamo da je $S^\star(k,p) = 1 + \sum_{i=1}^{n} a_i$.

Očito je $M^\star(1,0) = \emptyset$, $M^\star(1,p_1) = \{x_1\}$ i $M^\star(1,p)$ nije definirano za bilo koji $p \neq p_1$. Za $k \in \mathbb{N}$, $2 \leq k \leq n$ i za $p \in \mathbb{N}$, $0 \leq p \leq \sum_{i=1}^{n} p_i$ sljedećom se rekurzijom računa $M^\star(k,p)$ za bilo koje k i p:

$$M^\star(k,p) = \begin{cases} M^\star(k-1, p-p_k) \cup \{x_k\}, & \text{ako je } p_k \leq p, M^\star(k-1,p-p_k) \\ & \text{je definirano, } S^\star(k-1,p) \text{ je barem} \\ & S^\star(k-1,p-p_k) + a_k \text{ i} \\ & S^\star(k-1,p-p_k) + a_k \leq b, \\ M^\star(k-1,p), & \text{inače} \end{cases}$$

Najbolji izbor podskupa od $\{x_1, \ldots, x_k\}$ koji ima ukupnu cijenu p može biti jedno od sljedećeg:

1. najbolji izbor podskupa od $\{x_1, \ldots, x_{k-1}\}$ koji ima ukupnu cijenu $p - p_k$ i predmet x_k
2. najbolji izbor podskupa od $\{x_1, \ldots, x_{k-1}\}$ koji ima ukupnu cijenu p

Kako najbolji izbor podskupa od $\{x_1, \ldots, x_k\}$ koji ima ukupnu cijenu p ili sadrži ili ne sadrži x_k, jedan od tih dvaju izbora mora biti pravi.

Iz gornje formule za računanje $M^\star(k,p)$ dobivamo sljedeći algoritam za 0 - 1 PROBLEM RUKSAKA:

Input : skup X od n predmeta, za svaki $x_i \in X$ poznate su vrijednosti p_i, a_i, kapacitet ruksaka b
Output: podskup $Y \subseteq X$ takav da $\sum_{x_i \in Y} a_i \leq b$
begin
  for $p := 0$ *to* $\sum_{x_i \in Y} a_i$ **do**
    begin
      $M^\star(1,p) :=$ *undefined*;
      $S^\star(1,p) := 1 + \sum_{i=1}^{n} a_i$;
    end
  end
  $M^\star(1,0) := \emptyset$;
  $S^\star(1,0) := 0$;
  $M^\star(1,p_1) := \{x_1\}$;
  $S^\star(1,p_1) := a_1$;
  for $k := 2$ *to* n **do**
    for $p := 0$ *to* $\sum_{i=1}^{n} p_i$ **do**
      begin
        if $p_k \leq p$ *and* $(M^\star(k-1,p-p_k) \neq$ *undefined*$)$ *and*
         $S^\star(k-1,p-p_k) + a_k \leq S^\star(k-1,p)$ *and*
         $S^\star(k-1,p-p_k) + a_k \leq b$ **then**
          begin
            $M^\star(k,p) := M^\star(k-1,p-p_k) \cup \{x_k\}$;
            $S^\star(k,p) := S^\star(k-1,p-p_k) + a_k$;
          end
        else
          begin
            $M^\star(k,p) := M^\star(k-1,p)$;
            $S^\star(k,p) := S^\star(k-1,p)$;
          end
      end
  end
  $p^\star :=$ *maksimalni* p *takav da* $M^\star(n,p) \neq$ *undefined*;
  return $M^\star(n,p^\star)$
end

Može se pokazati da za danu instancu 0 - 1 PROBLEMA RUKSAKA s n predmeta, prethodni algoritam nalazi optimalno rješenje u vremenskoj složenosti $\mathcal{O}(n \sum_{i=1}^{n} p_i)$ gdje p_i označava cijenu i-tog predmeta. Dokaz ove tvrdnje može se pronaći u [1].

Bibliografija

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. *Complexity and Approximation*, Springer, 2003.

[2] J. Hromkovič. *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation and Heuristics*, Springer, 2004.

[3] C. H. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1994.

[4] M. Sipser. *Introduction to the Theory of Computation*, Cengage Learning 2013.

[5] E. Terzi. *Problems and Algorithms for Sequence Segmentations*, Helsinki University, 2006.

[6] V. V. Vazirani. *Approximation Algorithms*, Springer, 2004.

[7] M. Vuković. *Složenost algoritama*, nastavni materijal, PMF-MO Zagreb, 2015. https://www.math.pmf.unizg.hr/sites/default/files/pictures/00-sa-skripta-2015-lipanj-web.pdf

[8] R. A. Wilson. *Graphs, Colourings and the Four-Colour Theorem*, Oxford University Press, 2004.

Sažetak

U ovom diplomskom radu definirali smo optimizacijske probleme i objasnili neke od metoda za rješavanje tih problema. Da bismo mogli bolje proučiti i razumjeti optimizacijske probleme, potrebni su nam bili problemi odluke. Svakom se optimizacijskom problemu može pridružiti odgovarajući problem odluke. Ta nam je činjenica bila korisna u većini dokaza vezanih za klase složenosti za optimizacijske probleme: PO i NPO. Kao što probleme odluke možemo klasificirati u određenu klasu složenosti, tako i optimizacijske probleme svrstavamo u jednu od navedenih klasa. Definirali smo i opisali neke od poznatih optimizacijskih problema među kojima su problem TRGOVAČKOG PUTNIKA i 0 - 1 PROBLEM RUKSAKA. Ti problemi su NP-teški te za njih postoji određena aproksimacijska tehnika rješavanja kojom pokušavamo doći do optimalnog rješenja. Među tim tehnikama opisali smo sljedeće: pohlepna metoda, sekvencijalna metoda za problem particije, lokalno pretraživanje, linearno programiranje i dinamičko programiranje.

Ovdje je opisan samo jedan dio optimizacijskih problema te za njih postoji još dublja podjela i analiza. Postoji mnogo optimizacijskih problema međutim ovdje su dani primjeri s kojima se uglavnom susrećemo u svakodnevnom životu.

Summary

In this graduation paper we defined optimization problems and explained some of the methods for solving those problems. For better understanding and exploring of optimization problems, we first introduced the decision problems. There is a way how some decision problem could be joined with an optimization problem. This statement was useful in many theorems and proofs related to the complexity classes for optimization problems : PO and NPO. We defined and explained some of the well-known optimization problems like travelling salesman problem and maximum knapsack problem. This problems are NP-hard and there exist certain approximation techniques for getting the optimal solution. Some of the methods we explained are: greedy method, sequential method for partitioning the problem, local search, linear programming and dynamic programming.

In this paper we covered just a part of the optimisation problems suggesting there exists a deeper analysis regarding optimisation problems. Although there are many optimisation problems, in this paper we focused on explaining some of the examples from the daily life.

Životopis

Rođen sam 25. srpnja 1993. godine u Novoj Bili, općina Travnik, Republika BiH. Odrastao sam u Grubišnom Polju s majkom, ocem (koji je preminuo 2003. godine), te dvije sestre. Osnovnu i srednju školu pohađao sam u Srednjoj školi Bartola Kašića, Grubišno Polje. Opću gimnaziju završio sam 2012. godine. Zatim sam se upisao na Prirodoslovno-matematički fakultet - Matematički odsjek u Zagrebu. Preddiplomski studij završio sam 2015. godine nakon kojeg sam upisao diplomski studij Računarstvo i matematika na istom fakultetu.