SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Juraj Brigljević

PROBLEM MAKSIMALNOG TEžINSKOG NEZAVISNOG SKUPA U GRAFU

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan, 2023.

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. _____

Per aspera ad astra

Sadržaj

Uvod

Danas su poznati brojni problemi na grafovima koji imaju primjenu u stvarnom životu. Jedan od njih je i problem maksimalnog težinskog nezavisnog skupa u grafu. Kao što i ime navodi, problem se sastoji od pronalaska nezavisnog skupa vrhova u težinskome grafu, koji ima najveću težinu. Taj problem ima primjenu u slaganju bioloških mreža i u stvaranju rasporeda za upisivanje podataka na disk. On je također usko vezan uz još dva optimizacijska problema: problem maksimalne težinske klike i problem minimalnog težinskog vršnog pokrivača.

U ovome radu proučit ćemo teorijsku pozadinu problema maksimalnog težinskog nezavisnog skupa i njemu sličnih problema. Također, razmotrit ćemo i implementirati algoritme koji efikasno rješavaju problem u slučaju općenitih grafova i u slučaju stabala.

U prvome poglavlju zapisane su osnovne definicije potrebne za razumijevanje grafova te složenosti algoritama. U drugome poglavlju proučavamo neke teorijske rezultate koji povezuju navedene probleme i koji opisuju njihovu složenost u vremenskome smislu. U trećemu poglavlju proučavamo dva algoritma za rješavanje problema maksimalnog težinskog nezavisnog skupa. U slučaju općenitih grafova proučavamo jednu heuristiku, dok u slučaju stabala razmotrit ćemo egzaktni i optimalni algoritam. U četvrtome poglavlju osvrćemo se na implementacijske detalje ovih dvaju algoritama. U petome poglavlju testiramo algoritme na nekim primjerima i proučavamo njihove rezultate.

Poglavlje 1

Teorijska pozadina

Cilj ovoga poglavlja jest uvesti osnovne pojmove koji su potrebni za definiciju problema maksimalnog težinskog nezavisnog skupa i za pokazivanje da on pripada klasi NP-teških problema.

1.1 Pojmovi iz teorije grafova

Za početak promotrimo osnovne definicije u teoriji grafova. Za detaljniji uvod u grafove može se pogledati u [5] i [6].

Definicija 1.1.1. *Neusmjereni graf* G je par $G = (V, E)$, gdje je $V \neq \emptyset$ konačan skup, čije elemente zovemo *vrhovi*, a E je skup dvočlanih podskupova skupa V, čije elemente zovemo bridovi.

Ako je $e \in E$, $e = \{a, b\}$ neki brid, kažemo da su vrhovi a i b *incidentni* s bridom e i tada su vrhovi a i b međusobno *susjedni* vrhovi.

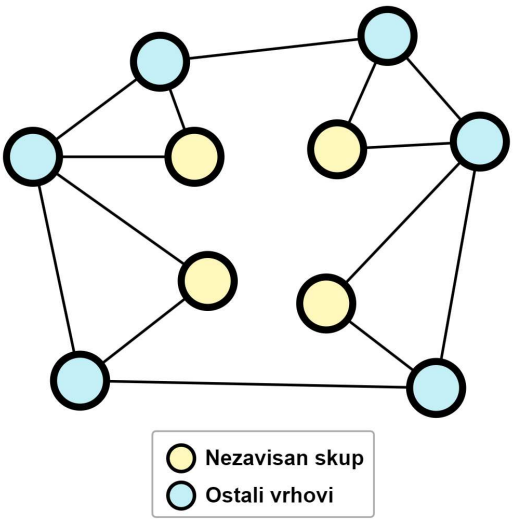
Ako uz graf G imamo još i funkciju $\omega\colon V \to \mathbb{N}$, onda par (G, ω) zovemo *neusmjereni težinski graf*. Za vrh $v \in V$, vrijednost $\omega(v)$ zovemo *težina vrha* v.

Definicija 1.1.2. Za graf $G' = (V', E')$ kažemo da je *podgraf* grafa $G = (V, E)$ ako je $V' \subseteq V$ i $E' \subseteq E$.

Za definiciju problema potrebno je odrediti pojam *nezavisnog skupa vrhova* u grafu. Osim toga, vidjet ćemo da je taj pojam usko vezan uz pojmove *klika* i *vršni pokrivač* u grafu.

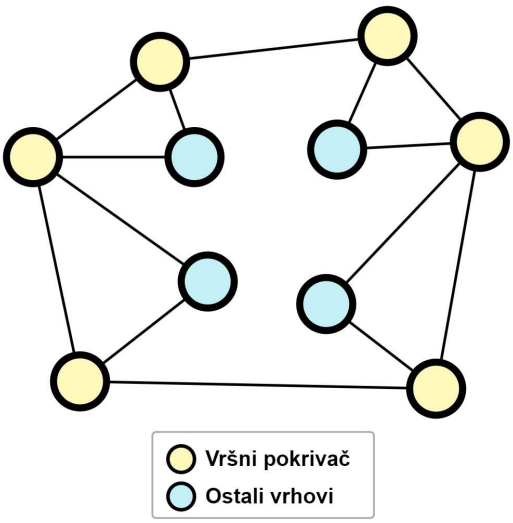
Definicija 1.1.3. Neka je $G = (V, E)$ graf.
Nezavisan skup vrhova je skup vrhova $V' \subseteq V$ t.d. $(\forall u, v \in V')\ (\{u, v\} \notin E)$.
Vršni pokrivač je skup vrhova $V' \subseteq V$ t.d. $(\forall \{u, v\} \in E)\ (u \in V' \text{ ili } v \in V')$.
Klika je skup vrhova $V' \subseteq V$ t.d. $(\forall u, v \in V')\ (\{u, v\} \in E)$.

Primjer 1.1.4. Promotrimo primjere skupova iz definicije 1.1.3:



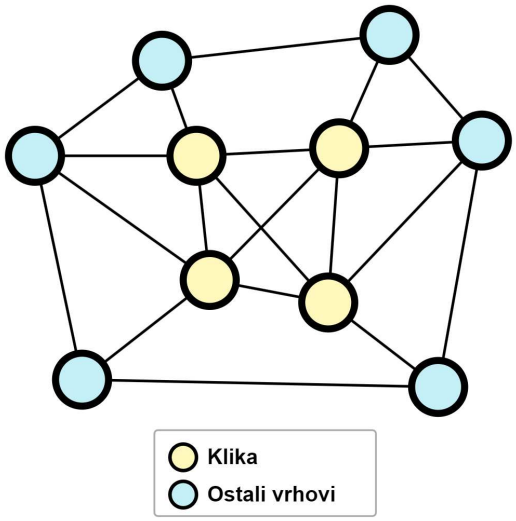
Slika 1.1: Primjer nezavisnog skupa vrhova

Na slici 1.1 možemo vidjeti primjer jednog nezavisnog skupa vrhova $N \in V$ u grafu. Primijetimo da je taj nezavisan skup vrhova također *maksimalan*, tj. ne postoji nezavisan skup $N' \in V$ koji sadrži više vrhova od N.



Slika 1.2: Primjer vršnog pokrivača

Slika 1.2 pokazuje primjer vršnog pokrivača. Uočimo da je taj skup vrhova upravo komplement nezavisnog skupa sa slike 1.1. Ta činjenica vrijedi općenito i dokazana je u 1.1.5.



Slika 1.3: Primjer klike

Lema 1.1.5. *Neka je $G = (V, E)$ graf i $V' \subseteq V$. Tada je ekvivalentno:*

1. *V' je nezavisan skup vrhova u grafu G*
2. *V' je klika u komplementarnom grafu G^c od G, gdje je $G^c = (V, E^c)$, a $E^c = \{\{u, v\} \mid u, v \in V \text{ i } \{u, v\} \notin E\}$*
3. *$V \setminus V'$ je vršni pokrivač u grafu G.*

Dokaz.

$\boxed{1. \Leftrightarrow 2.}$ Neka je V' nezavisan skup vrhova u grafu G. Tada ne postoji brid između nikoja dva vrha u V'. Kako je graf G^c komplementaran grafu G, među svaka dva vrha iz V' postoji brid u grafu G^c, tj. V' je klika u G^c. Obrat je analogan.

$\boxed{1. \Rightarrow 3.}$ Neka je V' nezavisan skup vrhova u grafu $G = (V, E)$. Tada ne postoji brid između nikoja dva vrha u V'. To znači da nikoji vrh iz V' nije incidentan s nekim bridom iz E. Dakle, svaki brid iz E je incidentan s nekim vrhom iz $V \setminus V'$, tj. $V \setminus V'$ je vršni pokrivač u grafu G.

1. $\Leftarrow$ 3. Neka je $V \setminus V'$ vršni pokrivač u grafu G, za neki $V' \subseteq V$,. Pretpostavimo da skup V' nije nezavisan u G. To znači da postoji brid $\{u, v\} \in E$, za neke $u, v \in V'$. No tada $V \setminus V'$ nije vršni pokrivač u grafu G, jer $u, v \notin V \setminus V'$, a $\{u, v\} \in E$. Dakle, V' je nezavisan skup u grafu G.

Sada očito vrijedi $2 \Leftrightarrow 1 \Leftrightarrow 3$. □

Definicija 1.1.6. Neka je $e_1, \ldots, e_n$ niz bridova u grafu G, za $n \in \mathbb{N}$. Ako postoje vrhovi $v_0, \ldots, v_n$ t.d. $e_i = \{v_{i-1}, v_i\}$, za $i = 1, \ldots, n$, kažemo da je niz bridova $e_1, \ldots, e_n$ *šetnja* između vrhova v_0 i v_n u grafu G. Ako su svi bridovi e_i međusobno različiti, onda tu šetnju nazivamo *staza*, a ako su uz to i svi vrhovi v_j međusobno različiti, onda tu stazu zovemo *put*.

Šetnja je *zatvorena* ako su joj početni i završni vrh jednaki. *Duljina* šetnje je broj njezinih vrhova. *Ciklus* je zatvoreni put duljine barem 1.

Kažemo da je graf *povezan* ako između svaka njegova dva vrha postoji šetnja. *Stablo* je povezan graf bez ciklusa.

1.2 Turingovi strojevi

Da bismo uopće mogli govoriti formalno o složenosti algoritama, moramo uvesti neki pojam apstraktnoga računala, a to će biti upravo *Turingovi strojevi*. Detaljnije o njima i o složenosti algoritama može se vidjeti u [11].

Definicija 1.2.1. *Deterministični Turingov stroj* je uređena sedmorka $(Q, \Sigma, \Gamma, \delta, q_0, q_\checkmark, q_x)$, gdje je:

- Q konačan neprazan skup čije elemente nazivamo *stanja*;
- Σ konačan skup koji nazivamo *alfabet*, a njegove elemente *znakovi*. Vrijedi da tzv. prazni simbol $\textvisiblespace \notin \Sigma$;
- Γ je konačan skup koji nazivamo *radna abeceda*. Vrijedi da je $\textvisiblespace \in \Gamma$ te $\Sigma \subset \Gamma$;
- $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ je proizvoljna funkcija koju nazivamo *funkcija prijelaza*;
- $q_0 \in Q$ i nazivamo ga *početno stanje*;
- $q_\checkmark \in Q$ i nazivamo ga *stanje prihvaćanja*;
- $q_x \in Q$ i nazivamo ga *stanje odbijanja*;

Napomena 1.2.2. Tehnički gledano, gornja definicija je definicija deterministočnog Turingova odlučitelja, ali radi jednostavnosti koristimo se izrazom stroj, skraćeno: DTS. Također, u nastavku teksta pretpostavljamo da se DTS T sastoji upravo od gornjih elemenata.

Definicija 1.2.3. Kažemo da DTS T *prepoznaje riječ* $w \in \Gamma^*$ ako postoji konačan niz parova $(q_0, s_0), \ldots, (q_m, s_m) \in Q \times \Gamma$, konačan niz simbola $I_1 \ldots, I_m \in \{-1, 0, 1\}$ i $m \in \mathbb{N}$, takvi da vrijedi:

1. q_0 je upravo početno stanje stroja T, a s_0 je prvi lijevi simbol riječi w;
2. $\forall j \in \{0, \ldots, m-1\}$ vrijedi $\delta(q_j, s_j) = (r_{j+1}, s_{j+1}, I_{j+1})$ i $q_j \notin \{q_\checkmark, q_x\}$;
3. $q_m = q_\checkmark$.

Za proizvoljan DTS T s $L(T)$ označavamo skup svih riječi koje T prepoznaje. Kažemo da neki DTS T *prepoznaje jezik* L ako je $L = L(T)$. Za neki jezik kažemo da je *prepoznatljiv* ako postoji DTS koji ga prepoznaje.

Kažemo da T staje s nekim ulazom ako postoje nizovi kao u gornjoj definiciji, gdje je $q_m \in \{q_\checkmark, q_x\}$. Ako T prepoznaje riječ, još kažemo da je *prihvaća*, a ako staje u stanju q_x, kažemo da *odbija* riječ.

Definicija 1.2.4. Za neki jezik $L \subseteq \Gamma^*$ kažemo da je *odlučiv* ako postoji DTS T koji ga prepoznaje i $(\forall w \in \Gamma^* \setminus L)$ stoj T s ulazom w staje u stanju q_x. Ako jezik nije odlučiv, kažemo da je *neodlučiv*.

Pomoću Turingovih odlučitelja možemo definirati probleme odluke - kod njih za neki kôd TO-a T i ulaznu riječ w odlučujemo pripada li ona skupu rješenja ili ne. Još moramo definirati i *kodiranje stroja* T.

Definicija 1.2.5. Neka je T proizvoljan TS gdje je $Q = \{q_0, q_1, \ldots, q_m, q_{m+1}, q_{m+2}\}$, pri čemu je $q_{m+1} = q_\checkmark$ i $q_{m+2} = q_x$. *Kodiranje* stroja T definiramo pomoću nizova nula i jedinica, na sljedeći način:

- svakom stanju q_i pridružujemo niz od $i + 1$ znakova nula;
- simbolu -1 (oznaka za pomak glave stroja ulijevo) pridružujemo 0, simbolu 1 pridružujemo 00, a simbolu 0 pridružujemo 000;
- ako je $\Gamma = \{s_1, \ldots, s_n\}$, onda simbolu s_i pridružujemo niz od i nula.

Ako je $\delta(q, s) = (q', s', I)$, za $q, q' \in Q$, $s, s' \in \Gamma$ i $I \in \{-1, 0, 1\}$, onda uređenu petorku (q, s, q', s', I) nazivamo *konfiguracija* TS-a T. Svakoj konfiguraciji pridružujemo niz nula i jedinica na sljedeći način:

"kod stanja q" 1 "kod simbola s" 1 "kod stanja q'" 1 "kod simbola s'" 1 "kod simbola I".

Budući da je svaki TS jedinstveno određen konačnim nizom konfiguracija $k_1, \ldots, k_p$ (zato što su skupovi Q i Γ konačni) i zato što možemo zadati da je niz konfiguracija uređen leksikografski, Turingovu stroju T na jedinstven način pridružujemo sljedeći kod:

1 "kod konfiguracije k_1" 1 "kod konfiguracije k_2" 1 ... 1 "kod konfiguracije k_p" 1,

koji nazivamo *kod Turingova stroja* T, a označavamo ga s $\langle T \rangle$. Ako je T TS i w neka riječ, onda s $\langle T, w \rangle$ označavamo kod stroja T koji na ulazu ima riječ w.

Deterministični strojevi su relativno prirodan pojam, ali za definiciju klase NP ipak će nam biti potreban kompleksniji stroj, a to je upravo *nederministični Turingov stroj*.

Definicija 1.2.6. *Nedeterministični Turingov stroj* (NTS) je uređena sedmorka $(Q, \Sigma, \Gamma, \Delta, q_0, q_\checkmark, q_x)$, gdje svi simboli osim Δ (δ u DTS-u) odgovaraju onima u DTS-u. Funkcija prijelaza ima sljedeći oblik: $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$.

Napomena 1.2.7. Ostale definicije vezane za nedeterminističke Turingove strojeve (npr. prepoznavanje riječi, prepoznavanje jezika i sl.) analogne su onima kod determinističnih Turingovih strojeva.

Napomena 1.2.8. Uočimo da se ove definicije odnose na pojam jezika, odnosno skup nekih riječi. Probleme kojima se bavimo u ovome radu definirat ćemo kao probleme *odluke*. To su problemi kod kojih se pitamo za koji skup ulaznih podataka problem daje odgovor "da". Uočimo da je pronalaženje odgovora "da" na pitanje odluke ekvivalentno prepoznavanju pripada li zadana riječ nekom jeziku. Stoga, za potrebe razmatranja teorije vremenske složenosti algoritama nastavljamo s pojmom jezika, dok ćemo se prilikom definiranja samog problema maksimalnog težinskog nezavisnog skupa prebaciti na pojam odluke.

1.3 Vremenska složenost algoritama

Sada možemo definirati klase vremenske složenosti naših problema. Za početak moramo vidjeti kako brojimo korake naših strojeva.

Definicija 1.3.1. Neka je T jednotračni TS, neka su $q, q' \in Q$, $s, s' \in \Gamma$ i $I \in \{-1, 0, 1\}$. Uređenu petorku (q, q', s, s', I) nazivamo *korak stroja* T ako vrijedi $\delta(q, s) = (q', s', I)$, odnosno ako vrijedi $(q, q', s, s', I) \in \Delta$ kada se radi o NTS-u.

Neka je T neki DTS koji staje za svaki ulazni podatak. *Vremenska složenost* stroja T je funkcija $time_T : \mathbb{N} \to \mathbb{N}$, gdje je $time_T(n)$ maksimalni broj koraka koje stroj T napravi za svaki ulazni podatak duljine n. *Vremenska složenost* NTS-a je analogna, ali pritom funkcija $time_T$ uzima u obzir sve grane izračunavanja NTS-a.

Ako je $time_T$ vremenska složenost stroja T, onda kažemo da je T *$time_T$-vremenski složen*.

Definicija 1.3.2. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$ dvije funkcije. Kažemo da je funkcija g *asimptotska gornja međa* za funkciju f ako postoje $c > 0$ i $n_0 \in \mathbb{N}$ takvi da $(\forall n \geq n_0)\ f(n) \leq cg(n)$. Oznaka: $f(n) \in O(g(n))$.

Definicija 1.3.3. Kažemo da je TS T *vremenski polinomni* ako postoji neki polinom f takav da vrijedi $time_T(n) \in O(f(n))$.

Definicija 1.3.3 vrijedi i za DTS-ove i za NTS-ove. Naravno, znajući da nedeterminističnost intuitivno omogućava istovremeno praćenje svih grana izračunavanja nekoga stroja, za očekivati je da će problemi koje odlučuju NTS-ovi biti kompleksniji i vremenski složeniji.

Definicija 1.3.4. Neka je $f : \mathbb{N} \to \mathbb{R}^+$ proizvoljna funkcija. Klasa vremenske složenosti $DTIME(f(n))$ je skup svih jezika koji su odlučivi nekim $O(f(n))$-vremenski složenim jednotračnim DTS-om.

Klasa vremenske složenosti $NTIME(f(n))$ je skup svih jezika koji su odlučivi nekim $O(f(n))$-vremenski složenim NTS-om.

Definicija 1.3.5. S $PTIME$, ili samo P, označavmo klasu svih jezika koji su odlučivi nekim DTS-om vremenske složenosti $O(n^k)$, $\forall k \in \mathbb{N}$. Dakle, vrijedi $P = \bigcup_{k\in\mathbb{N}} DTIME(n^k)$.

S $NPTIME$, ili samo NP, označavamo klasu svih jezika koji su odlučivi nekim NTS-om vremenske složenosti $O(n^k)$, $\forall k \in \mathbb{N}$. Dakle, vrijedi $NP = \bigcup_{k\in\mathbb{N}} NTIME(n^k)$.

Sada kada znamo za klasu NP, možemo proučiti gdje su smješteni nama relevantni problemi. No, kako je središnji pojam ovoga rada problem maksimizacije, potreban nam je još i malo drugačiji pojam. To će biti upravo pojam *NP-teških* problema, koji na neki način smješta sve probleme u klasi NP na istu razinu. Naime, kada bismo za neki NP-težak problem pronašli polinomijalni algoritam, takav bismo imali i za svaki problem u klasi NP. Kako takav algoritam još nije poznat, a nije dokazano ni suprotno, pitanje $P \stackrel{?}{=} NP$ i dalje stoji otvoreno.

Definicija 1.3.6. Kažemo da je neka funkcija $f : \Gamma_1^* \to \Gamma_2^*$ *vremenski polinomno izračunljiva* ako postoji polinomno vremenski složen TS koji za svaku ulaznu riječ $w \in \Gamma_1^*$ na traku ispisuje $f(w)$.

Kažemo da je jezik $L_1 \subseteq \Gamma_1^*$ *polinomno reducibilan* na jezik $L_2 \subseteq \Gamma_2^*$ ako postoji vremenski polinomno izračunljiva funkcija $f\colon \Gamma_1^* \to \Gamma_2^*$, takva da za svaki $w \in \Gamma_1^*$ vrijedi $w \in L_1 \iff f(w) \in L_2$.

Ako je jezik L_1 polinomno reducibilan na jezik L_2, onda to označavamo s $L_1 \leq_p L_2$.

Lema 1.3.7. *Za proizvoljne jezike L_1, L_2 i L_3 takve da je $L_1 \leq_p L_2$ i $L_2 \leq_p L_3$ vrijedi $L_1 \leq_p L_3$.*

Dokaz. Neka su $L_1 \subseteq \Gamma_1^*$, $L_2 \subseteq \Gamma_2^*$ i $L_3 \subseteq \Gamma_3^*$ prozivoljni jezici takvi da je $L_1 \leq_p L_2$ i $L_2 \leq_p L_3$. To znači da postoje vremenski polinomno izračunljive funkcije $f\colon \Gamma_1^* \to \Gamma_2^*$ i $g\colon \Gamma_2^* \to \Gamma_3^*$ takve da za svaku $w \in \Gamma_1^*$ vrijedi $w \in L_1 \iff f(w) \in L_2$ i za svaku $v \in \Gamma_2^*$ vrijedi $v \in L_2 \iff g(v) \in L_3$. Neka je $w \in \Gamma_1^*$, tada je $w \in L_1 \iff f(w) \in L_2 \iff g(f(w)) \in L_3$, tj. $L_1 \leq_p L_3$. □

Definicija 1.3.8. Kažemo da je jezik L *NP-težak* ako vrijedi: $(\forall L' \in NP)\ L' \leq_p L$.

Kažemo da je jezik L *NP-potpun* ako vrijedi: $L \in NP$ i L je NP-težak.

Za kraj navodim teorem o certifikatu, koji olakšava pokazivanje da određeni problem pripada klasi NP. Za dokaz pogledajte u [11].

Teorem 1.3.9 (O certifikatu)**.**

Za svaki jezik L vrijedi:

$$L \in NP \iff (\exists R \in P)(\exists k \in \mathbb{N})\ L = \{x \mid (\exists c)(|c| \in O(|x|^k) \wedge (x, c) \in R)\}.$$

1.4 Logika sudova

Jedan od osnovnih problema koji su NP-potpuni jest upravo problem ispunjivosti formule logike sudova - SAT. No da bismo formalno mogli definirati taj problem, trebaju nam neke definicije iz same logike sudova.

Definicija 1.4.1. *Alfabet logike sudova* je unija skupova A_1, A_2 i A_3, gdje je:

$A_1 = \{P_0, P_1, P_2, \dots\}$ prebrojiv skup čije elemente nazivamo *propozicionalne varijable*,

$A_2 = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ skup logičkih veznika,

$A_3 = \{(,)\}$ skup pomoćnih simbola.

Definicija 1.4.2. *Atomarna formula* je svaka propozicionalna varijabla. *Formula logike sudova* definira se rekurzivno:

1. svaka atomarna formula je formula;

2. ako su A i B formule, onda su i riječi $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$ i $(A \leftrightarrow B)$ formule;

3. riječ alfabeta logike sudova je formula ako je nastala primjenom konačno mnogo koraka uvjeta 1 i 2.

Definicija 1.4.3. Svako preslikavanje $I\colon \{P_0, P_1, \dots\} \to \{0, 1\}$ nazivamo *interpretacija*. Vrijednost interpretacije I na proizvoljnoj formuli definiramo rekurzivno ovako:

$$\begin{aligned} I(\neg A) = 1 &\iff I(A) = 0; \\ I(A \wedge B) = 1 &\iff I(A) = 1 \wedge I(B) = 1; \\ I(A \vee B) = 1 &\iff I(A) = 1 \vee I(B) = 1; \\ I(A \to B) = 1 &\iff I(A) = 0 \vee I(B) = 1; \\ I(A \leftrightarrow B) = 1 &\iff I(A) = I(B). \end{aligned}$$

Definicija 1.4.4. Za fomulu F logike sudova kažemo da je *ispunjiva* ako postoji interpretacija I takva da vrijedi $I(F) = 1$.

Definicija 1.4.5. Atomarnu formulu i njezinu negaciju nazivamo *literal*. Formulu oblika $A_1 \wedge A_2 \wedge \cdots \wedge A_n$ nazivamo *konjunkcija*, a formulu oblika $A_1 \vee A_2 \vee \cdots \vee A_n$ nazivamo *disjunkcija*, gdje su A_i proizvoljne formule.

Elementarna disjunkcija je disjunkcija literala. *Konjunktivna normalna forma*, ili samo *knf*, je konjunkcija elementarnih disjunkcija.

Poglavlje 2

Problem maksimalnog težinskog nezavisnog skupa

Sada kada znamo osnovne pojmove koji su nam potrebni, možemo definirati središnji problem ovoga rada i njemu bliske probleme.

2.1 Definicije relevantnih problema

Definicija 2.1.1. Definirajmo neke probleme:

- SAT je problem koji odgovara na pitanje je li neka formula F logike sudova u knf ispunjiva.
- 3-SAT je problem koji odgovara na pitanje je li neka formula F logike sudova koja je u 3-knf ispunjiva (3-knf je knf u kojoj svaka elementarna disjunkcija sadrži točno 3 literala).
- CLIQUE je problem koji za neusmjereni graf G i neki broj k odlučuje postoji li u grafu G klika veličine barem k.
- VC je problem koji za neusmjereni graf G i neki broj k odlučuje postoji li u grafu G vršni pokrivač veličine najviše k.
- IS je problem koji za neusmjereni graf G i neki broj k odlučuje postoji li u grafu G nezavisni skup veličine barem k.

Definicija 2.1.2. Problem *maksimalnog nezavisnog skupa u grafu* (*eng. maximum independent set*, u nastavku: MIS) u neusmjerenom grafu G problem je pronalaska najvećeg nezavisnog skupa vrhova u G.

Problem *maksimalnog težinskog nezavisnog skupa u grafu* (*eng. maximum weight independent set*, u nastavku: MWIS) u neusmjerenom težinskom grafu (G, ω) problem je pronalaska nezavisnog skupa vrhova u G koji ima maksimalnu težinu, pri čemu je težina nezavisnog skupa jednaka zbroju težina svih njegovih vrhova.

Kako pojam NP-teških problema definiramo samo za probleme odluke, za problem MWIS, koji je problem maksimizacije, ne možemo izravno reći da pripada klasi NP-teških problema. No, kada bismo za problem maksimizacije imali polinomni algoritam, onda je takav i njemu pripadni problem odluke MIS. Naime, problem MIS tada možemo riješiti tako da u polinomnom vremenu riješimo MWIS i onda se pitamo odgovara li njegovo rješenje onome u problemu MIS. To znači da ako je problem MIS NP-težak, onda i njemu pripadni problem maksimizacije MWIS možemo tako shvatiti, jer ima upravo traženo svojstvo da kada bi bio polinoman, onda bi i svaki problem odluke u klasi NP bio takav.

Za probleme maksimizacije mogu se definirati i pripadni pojmovi složenosti, ali kako to nije područje ovoga rada navodim literaturu u kojoj se može uvjeriti u takvo što [4].

Prije nego što nastavimo s proučavanjem problema iz teorijske perspektive, objasnio bih jedan primjer u kojemu se on koristi. Danas u digitaliziranome svijetu imamo sve veću potrebu za generiranjem i skladištenjem izrazito velike količine podataka. Stoga postoje brojne podatkovne centrale koje pohranjuju i pružaju pristup raznim skupovima podataka. Za to se koriste veliki brojevi rotirajućih diskova koji troše veliku količinu energije, a očekuje se da će s napretkom tehnologije cijena te energije biti glavni trošak. Dakle, možemo se pitati postoje li načini da se energija uštedi prilikom korištenja takvih postrojenja.

Postoje razne tehnike za očuvanje energije, a velik dio njih temelji se na ideji da se brzina okretanja samih diskova smanjuje do stanja niske potrošnje nakon određenog perioda neaktivnosti. Jedan od načina da se ovo postigne jest da se napravi raspored zahtjeva za zapisom na disk tako da potrošnja bude što manja. Taj se problem može modelirati upravo kao problem MWIS. Naime, ako zahtjeve shvatimo kao vrhove u grafu, pridružimo im težine koje predstavljaju uštedu u energiji, a povežemo bridovima one zahtjeve koji se ne mogu izvršiti istovremeno, onda upravo tražimo nezavisni skup u grafu koji ima maksimalnu težinu. Za detaljniji opis samoga problema može se proučiti [3].

2.2 Vremenska složenost problema MWIS

Teorem 2.2.1 predstavlja jedan od najbitnijih rezultata teorije složenosti algoritama, a njegov dokaz može se naći u [11]. U navedenoj literaturi može se naći i dokaz teorema 2.2.2, gdje se polinomijalnim svođenjem problema SAT na problem 3-SAT pokazuje da je i 3-SAT NP-potpun.

Teorem 2.2.1. *(Cook-Levinov teorem)*
Problem SAT je NP-potpun.

Teorem 2.2.2.

Problem 3-SAT je NP-potpun.

Pogledajmo sada kako nam teorem 2.2.2 može poslužiti da bismo proučili vremensku složenost nama relevantnih problema.

Teorem 2.2.3.

1. *3-SAT* $\leq_p$ *CLIQUE.*
2. *CLIQUE* $\leq_p$ *VC.*
3. *CLIQUE* $\leq_p$ *IS.*

Dokaz. 1. Pokažimo 3-SAT $\leq_p$ CLIQUE. Neka je F neka 3-knf koja sadrži bar k elementarnih disjunkcija. Za formulu F konstruiramo graf G na sljedeći način: vrhove čine svi literali formule F, gdje pritom više vrhova može biti označeno istim literalom. Bridovima su povezani svi vrhovi osim: vrhova koji predstavljaju literale iz iste elementarne disjunkcije, i vrhova koji su označeni komplementarnim literalima (dakle P i $\neg P$).

Želimo pokazati da vrijedi $F \in SAT \iff$ graf G sadrži kliku veličine bar k.

Pretpostavimo da je 3-knf F s bar k elementarnih disjunkcija ispunjiva. Neka je I neka interpretacija takva da je $I(F) = 1$. To znači da u svakoj elementarnoj disjunkciji postoji bar jedan literal P za koji je $I(P) = 1$. Iz svake od barem k elementarnih disjunkcija odaberemo po jedan takav. Sada je jasno da upravo vrhovi koji odgovaraju tim literalima tvore jednu kliku veličine bar k u grafu G. Naime, u G su povezani svi vrhovi, osim onih iz iste elementarne disjunkcije (a mi odabiremo samo po jedan literal iz svake), i osim onih komplementarnih (a mi takve nikada nećemo odabrati jer ne može istovremeno vrijediti $I(P) = 1$ i $I(\neg P) = 1$).

Pretpostavimo da graf G sadrži kliku K veličine bar k. Definirajmo parcijalnu interpretaciju $I\colon Var(F) \to \{0, 1\}$ na sljedeći način:

$$I(P_i) := \begin{cases} 1, & \text{ako je } P_i \in K \\ 0, & \text{inače.} \end{cases} \tag{2.1}$$

Kako po pretpostavci F sadrži bar k elementarnih disjunkcija, a K je klika veličine bar k, mora vrijediti da je u svakoj elementarnoj disjunkciji bar jedan literal L takav da je $I(L) = 1$ (to su upravo literali koji su predstavljeni vrhovima iz klike K). Sada je jasno da je $I(F) = 1$, odnosno F je ispunjiva.

Još je pitanje je li ovo svođenje vremenski polinomno. Neka je $f := |F|$ duljina formule F. Jasno je da je $v := |Var(F)| \leq f$ pa graf G ima točno v vrhova. Znamo da je broj bridova u G manji ili jednak $v^2 \leq f^2$. Dakle, da bismo konstruirali graf G, moramo proći cijelom formulom da popišemo sve vrhove u vremenu $O(f)$, bridove (kao parove vrhova) zatim zapišemo u vremenu $O(f^2)$. Još trebamo maknuti sve bridove iz iste elementarne disjunkcije, što možemo učiniti još jednim prolaskom po formuli u vremenu $O(f)$, i prolaskom po svim bridovima kako bismo vidjeli povezuju li neki komplementarne literale u vremenu $O(f^2)$. Ukupno je to $O(f^2)$, dakle vremenski polinomno.

2. Pokažimo CLIQUE $\leq_p$ VC. Za ovo će nam poslužiti lema 1.1.5 po kojoj u grafu $G = (V, E)$ vrijedi da je $V' \subseteq V$ je klika u komplementarnom grafu $G^c \iff V \setminus V'$ je vršni pokrivač u grafu G. Zbog te leme jasno vrijedi da G^c ima kliku V' veličine bar $k \iff G$ ima vršni pokrivač $V \setminus V'$ veličine najviše $|V| - k$.

Još je pitanje je li ovo svođenje polinomno. Komplementiranje grafa zahtijeva prolazak po svim bridovima u $V \times V$ i izbacivanje onih u G te dodavanje onih koji nisu u G, u vremenu $O(|V|^2)$. Komplementiranje vrhova nekog podskupa $V' \subseteq V$ slično zahtijeva vrijeme $O(|V|)$. Ako pretpostavimo da je zapis broja k binaran, razlika $|V| - k$ se jasno dobije vremenu $O(|V|)$. Dakle, svođenje je očito polinomno.

3. Pokažimo CLIQUE $\leq_p$ IS. Opet se pozivamo na rezultat leme 1.1.5, tj. na ekvivalenciju koja kaže da je u grafu $G = (V, E)$ neki skup $V' \subseteq V$ nezavisan $\iff$ je V' klika u komplementarnom grafu G^c. Sada vrijedi da G^c ima kliku veličine bar $k \iff G$ ima nezavisan skup veličine bar k.

U svođenju opet imamo komplementiranje grafa u vremenu $O(|V|^2)$, dok je skup vrhova koji predstavlja rješenje upravo isti. Sveukupno, svođenje je očito polinomno. □

Korolar 2.2.4.

- *Problem CLIQUE je NP-potpun.*
- *Problem VC je NP-potpun.*
- *Problem IS je NP-potpun.*

Dokaz. Koristeći se teoremima 2.2.2 i 2.2.3 jednostavno zaključujemo da je svaki od navedenih problema NP-potpun. Naime, relacija $\leq_p$ je tranzitivna po lemi 1.3.7 pa vrijedi da (3-SAT$\leq_p$CLIQUE$\leq_p$IS, VC)$\Rightarrow$(3-SAT$\leq_p$CLIQUE, IS, VC), a znamo da je 3-SAT NP-težak kao jedan NP-potpun problem pa su takvi i CLIQUE, IS i VC. S obzirom na to da problemi CLIQUE, VC i IS pripadaju klasi NP, sva tri problema su NP-potpuni. Činjenica da ovi problemi pripadaju klasi NP jednostavno slijedi iz teorema o certifikatu 1.3.9. Kao primjer pogledajmo pripada li problem IS klasi NP. Zanima nas ako dobijemo neki nezavisni skup kao certifikat (čija je duljina sigurno polinomna jer je zapis nekog podskupa

skupa svih vrhova grafa nužno kraći ili jednak zapisu cijelog grafa), možemo li u polinomnom vremenu provjeriti je li navedeni skup nezavisan? Naravno, to se jednostavno postigne prolaskom svih bridova i uvjeravanjem da nijedan od njih nije između vrhova našeg certifikata, a taj je postupak polinoman u duljini zapisa grafa. Za probleme CLIQUE i VC na analogan način dobije se da pripadaju klasi NP. □

Kao što smo već napomenuli, probleme optimizacije možemo smatrati NP-teškima ako bi njihovo vremenski polinomno rješenje značilo da su svi problemi u NP rješivi u polinomnom vremenu na deterministočnom stroju. Pogledajmo što nam to znači za probleme MIS i MWIS.

Korolar 2.2.5.

- *Problem MIS je NP-težak.*
- *Problem MWIS je NP-težak.*

Dokaz. Neka je G neki graf i $k \in \mathbb{N}$ neki broj. Pretpostavimo da u polinomnome vremenu možemo pronaći rješenje problema MIS. To znači da u polinomnome vremenu možemo odrediti najveći nezavisni skup N' u proizvoljnome grafu G'. Da bismo sada odlučili ima li graf G nezavisni skup veličine barem k (što je upravo odgovor na problem IS), moramo vidjeti je li $k \leq |N|$, ako je N najveći nezavisni skup grafa G. Ako to ne vrijedi, tj. ako najveći nezavisni skup u grafu G ima manje od k vrhova, onda očito u G ne postoji nezavisni skup veličine bar k. A ako to vrijedi, onda je upravo N traženi nezavisni skup. Dakle, polinomno rješenje problema MIS daje nam polinomno rješenje problema IS, za koji znamo da je NP-težak po korolaru 2.2.4. Stoga, problem MIS je NP-težak.

Slično vrijedi i u slučaju problema MWIS. Naime, neka je G neki graf. Pretpostavimo da u polinomnome vremenu možemo pronaći rješenje problema MWIS. To znači da u polinomnome vremenu možemo odrediti nezavisni skup N' u proizvoljnome težinskom grafu (G', ω'), s najvećom mogućom težinom. Sada najveći nezavisni skup u grafu G (dakle rješenje problema MIS) možemo pronaći tako da shvatimo graf $G = (V, E)$ kao težinski graf s težinskom funkcijom ω, za koju je $\omega(v) = 1$, $\forall v \in V$. Sada je jasno da je N maksimalni nezavisni težinski skup u $(G, \omega) \iff N$ je maksimalni nezavisni skup u G, tj. polinomno rješenje problema MWIS daje nam polinomno rješenje problema MIS. Stoga, problem MWIS je također NP-težak. □

Pitanje koje se postavlja jest koje su posljedice gornjih teorema? Kako smo već napomenuli, nije poznat odgovor na pitanje $P \stackrel{?}{=} NP$, a na temelju te činjenice i korolara 2.2.5 ne očekujemo da postoji efikasni egzaktni algoritam za rješavanje problema MWIS na općenitome grafu (odnosno algoritam koji pronalazi optimalno rješenje u polinomnom vremenu). Ali, moguće je da postoji efikasni egzaktni algoritam za neke posebne vrste grafova. U idućim poglavljima se zbog toga osvrćemo na proučavanje približnog algoritma

(heuristike) za rješavanje problema MWIS na općenitom grafu, a vidjet ćemo da jedna posebna vrsta grafova (stabla) imaju efikasan i egzaktan algoritam za rješavanje problema MWIS.

Kao što smo pokazali vezu među problemima MWIS, MIS i IS, tako se mogu pokazati i analogne veze među problemima MWVC (*eng. minimum weight* VC), MVC (*eng. minimum* VC) i VC, odnosno među problemima MWCLIQUE (*eng. maximum weight* CLIQUE), MCLIQUE (*eng. maximum* CLIQUE) i CLIQUE. Imajući na umu lemu 1.1.5, iz rješenja problema MWIS jednostavno možemo dobiti rješenja problema MWVC i MWCLIQUE, odnosno iz rješenja problema MIS dobivamo rješenja problema MVC i MCLIQUE. Dakle, algoritmi za rješavanje problema MWIS koje ćemo proučavati mogu se primijeniti i za rješavanje problema MWVC i MWCLIQUE.

Poglavlje 3

Algoritmi za rješavanje problema MWIS

3.1 Algoritam za rješavanje problema MWIS na općenitim grafovima

U općenitome slučaju promatrat ćemo jedan algoritam ponavljanog lokalnog pretraživanja (*eng. iterated local search*, ILS) uz varijabilni spust korištenjem susjedstava (*eng. variable neighborhood descent*, VND). Navedene paradigme opisane su u [7] i [8].

Ponavljano lokalno pretraživanje

U ILS heuristici želimo istražiti područje rješenja tako što skočimo na neko rješenje koje je slično trenutnome i u njegovu susjedstvu potražimo boljeg kandidata. Na zadanome rješenju s^* provedemo korak `Perturbiraj` koji nas dovede do nekog posrednog rješenja s'. Na tom posrednom rješenju zatim pozovemo metodu `LokalnoPretraživanje`, koja nam ponudi novog kandidata za rješenje $s_{kandidat}$. Ako kandidat zadovoljava `UvjetPrihvaćanja`, on postaje novi s^* s kojim krećemo u sljedeći korak heuristike, a inače ponavljamo postupak s istim s^*.

Ovaj postupak prikazan je pseudokodom 3.1.1. Na koji je način svaki od koraka implementiran ovisi o primjeni, no generalno ideja je sljedeća:

- metoda `GenerirajPočetnoRješenje` nam daje neko nasumično ili pohlepno početno rješenje na kojemu možemo provoditi postupak;
- `LokalnoPretraživanje` pokušava poboljšati trenutno rješenje traženjem boljega u njegovu susjedstvu;

- kako ne bismo zapeli u lokalnom optimumu, pozivamo metodu `Perturbiraj` da bismo što bolje istražili područje rješenja;

- i na kraju definiramo `UvjetPrihvaćanja`, pomoću kojega određujemo je li naš kandidat dovoljno dobar da bismo s njim nastavili algoritam.

Pseudokod 3.1.1: Ponavljano lokalno pretraživanje - ILS

```
1 ILS
2     s_0 = GenerirajPočetnoRješenje();
3     s* = LokalnoPretraživanje(s_0);
4     dok ( uvjet zaustavljanja nije zadovoljen ) :
5         s' = Perturbiraj(s*);
6         s_kandidat = LokalnoPretraživanje(s');
7         s* = UvjetPrihvaćanja();
```

Kako je svaka od ovih metoda implementirana vidjet ćemo kasnije.

Varijabilni spust korištenjem susjedstava

Ključni cilj varijabilnog spusta korištenjem susjedstava jest pronalazak lokalnog optimuma istraživanjem niza susjedstava trenutnog rješenja. Susjedstva ovise o samome problemu i definiraju se unaprijed. Novo rješenje prihvaća se isključivo ako je bolje od prethodnoga, stoga je ova tehnika izrazito jednostavna te često ne zahtijeva definiranje dodatnih parametara.

VND nastoji iskoristiti sljedeća opažanja:

- lokalni optimum jednoga susjedstva nije nužno lokalni optimum i nekoga drugog susjedstva;

- globalni optimum je neki lokalni optimum s obzirom na sva moguća susjedstva

- za mnoge probleme lokalni optimumi nekih susjedstava često su relativno bliski.

Dakle, ovim postupkom trudimo se pronaći što bolji optimum promatrajući razna susjedstva koja možda imaju različite optimume. Pritom imamo na umu da neko od susjedstava sigurno sadrži globalni optimum, a o njemu često možemo dobiti bar nekakvu informaciju iz sličnih susjedstava jer su ona često relativno bliska. Pseudokod 3.1.2 prikazuje postupak.

Pseudokod 3.1.2: Varijabilni spust korištenjem susjedstava - VND

ulazni podatci: broj susjedstava l_{max}

```
1  VND
2      s = GenerirajPočetnoRješenje();
3      dok ( u prethodnoj iteraciji nalaziš bolje rješenje ) :
4          l = 1;
5          dok ( l ≠ l_max ) :
6              N_l(s) = Susjedstvo(l, s);
7              pronađi najboljeg susjeda s' ∈ N_l(s);
8              ako ( s' je bolje rješenje od s ) :
9                  s = s';
10                 l = 1
11             inače : l += 1;
```

U našem primjeru imat ćemo dva susjedstva, a ostatak postupka u potupnosti se poklapa sa pseudokodom 3.1.2.

Heuristika ILS-VND

Sada možemo proučiti kako u paradigmu ILS možemo uključiti heuristiku VND da bismo efikasno pretražili područje rješenja problema MWIS. Ova metoda opisana je u [9]. Pseudokod 3.1.3 prikazuje cijeli postupak.

Algoritam kao ulazne podatke prima težinski neusmjereni graf $G = (V, E, w)$ i maksimalni broj iteracija *maxIter*. Također, algoritam prima četiri parametara $c_1, \ldots, c_4$, koje trebamo prilagoditi da bismo imali dobar omjer između kvalitete rješenja i vremena izvršavanja programa. Algoritam vraća maksimalni nezavisni skup S^*, čiju težinu saznajemo pozivanjem metode `Težina(`S^*`)`.

Inicijalizacija. Postupak započinjemo nasumičnom inicijalizacijom rješenja u liniji 2. Početni skup S_0 generiramo uniformnim dodavanjem nasumičnih vrhova $v \in V$. S_0 punimo vrhovima dok više nema slobodnih, tj. $(\nexists v \in V \setminus S_0)(\forall u \in S_0)((u, v) \notin E)$, čime smo osigurali da je naš skup valjan neproširiv nezavisan skup.

Glavni dio algoritma jest petlja *while*, koja počinje u liniji 8. Ona upravo prati ranije opisanu paradigmu ILS. Naime, u liniji 9 perturbiramo trenutno rješenje. Zatim na njemu pozovemo postupak `LokalnoPretraživanje` u liniji 10, a u našem slučaju to je upravo heuristika VND. I na kraju, u liniji 11, odlučimo koje ćemo rješenje zadržati metodom `Prihvati()`.

Opišimo kako su implementirane gore navedene metode u algoritmu ILS-VND za problem MWIS.

Pseudokod 3.1.3: ILS-VND algoritam za problem MWIS

ulazni podatci: težinski neusmjereni graf $G = (V, E, w)$, maksimalni broj iteracija *maxIter*, parametri c_1, c_2, c_3 i c_4
izlazni podatci: maksimalni nezavisni skup S^*

```
1  ILS-VND (G, maxIter)
2      S_0 = Inicijaliziraj(G);
3      S = LokalnoPretraživanje(S_0, G);
4      S* = S;
5      lokalni_w = Težina(S);
6      iter = 1;
7      i = 1;
8      dok ( iter ≤ maxIter ) :
9          S' = Perturbiraj(c_1, S, G);
10         S' = LokalnoPretraživanje(S', G);
11         (S, S*, i, lokalni_w) = Prihvati(S, S*, S', i, lokalni_w, G, c_2, c_3, c_4);
12         iter += 1;
13     vrati S*
```

Perturbacija. Metoda `Perturbiraj(`k, S, G`)` modificira trenutno rješenje tako što u njega ubaci $k \in \mathbb{N}$ uniformno odabranih nasumičnih vrhova koji nisu u rješenju. Kako bi dobiveni skup i dalje bio nezavisan, uz to izbaci i sve susjedne vrhove novododanih k vrhova. S obzirom na to da je moguće da sada postoji strogo veći nezavisni skup koji sadrži S, u njega želimo dodati uniformno odabrane nasumične vrhove dok više ne bude proširiv (do nezavisnog skupa koji ga sadrži).

Lokalno pretraživanje. Kako bismo poboljšali trenutno rješenje, pozivamo metodu `LokalnoPretraživanje(`S, G`)`, koja se koristi VND heuristikom. Kako znamo da nam je za definiciju te heuristike potreban skup susjedstava, opisat ćemo par takvih struktura za problem MWIS: $(\omega, 1)$-zamjenu ($\mathcal{N}_1$) i $(1, 2)$-zamjenu ($\mathcal{N}_2$). Kao što smo već vidjeli u generalnom opisu VND-a, redom odabiremo novo susjedstvo da bismo nastavili potragu za boljim rješenjem, kad god trenutno susjedstvo ne uspije poboljšati trenutno rješenje. Inače, kada nađemo poboljšano rješenje, vraćamo se na prvo susjedstvo i ponavljamo postupak. Na kraju, iz istog razloga kao i kod perturbacije, u rješenje dodajemo slobodne vrhove do maksimalnosti. Pseudokod ove metode vidljiv je u 3.1.4.

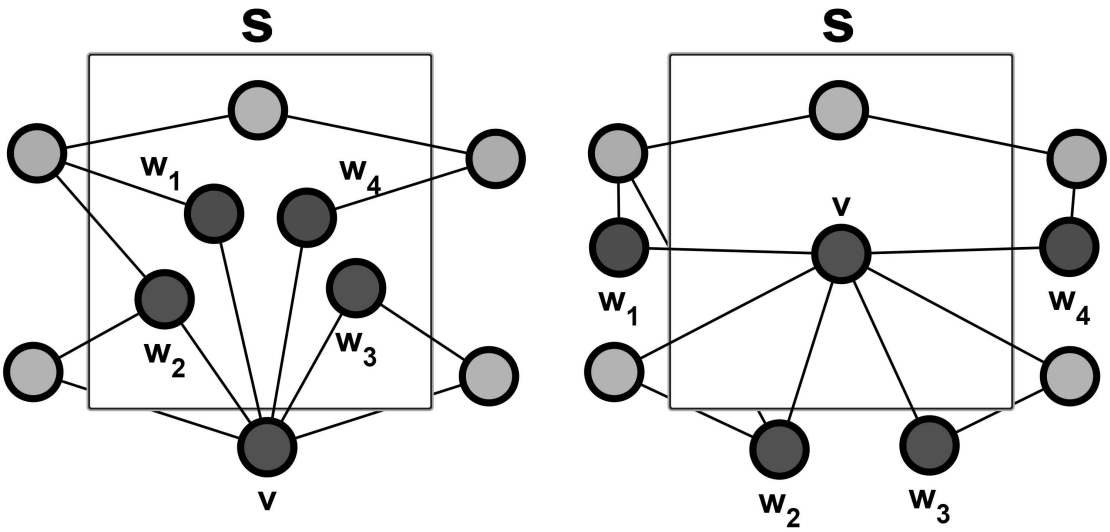
Susjedstvo $(\omega, 1)$-zamjena ubacuje jedan vrh $v \in V \setminus S$ u rješenje i onda briše njegovih $\omega = |N_S(v)|$ susjednih vrhova u rješenju. Stoga, da bi novo rješenje bilo poboljšano, težina umetnutog vrha $w(v)$ mora biti veća od sume težina izbačenih vrhova $\Sigma_{y \in N_S(v)} w(y)$.

Korisno je uočiti da nekog kandidata za $(\omega, 1)$-zamjenu možemo provjeriti u vremenu

Pseudokod 3.1.4: Metoda LokalnoPretraživanje

ulazni podatci: maksimalni nezavisni skup S i težinski neusmjereni graf $G = (V, E, w)$
izlazni podatci: maksimalni nezavisni skup S

```
1  LokalnoPretraživanje (S, G)
2      k = 1;
3      dok ( k ≤ 2 ) :
4          S' = Poboljšanje(k, S, G);
5          ako ( Težina(S') ≤ Težina(S) ) : k += 1 ;
6          inače :
7              k = 1;
8              S = S';
9              S = DodajSlobodneVrhove(S, G);
10     vrati S
```

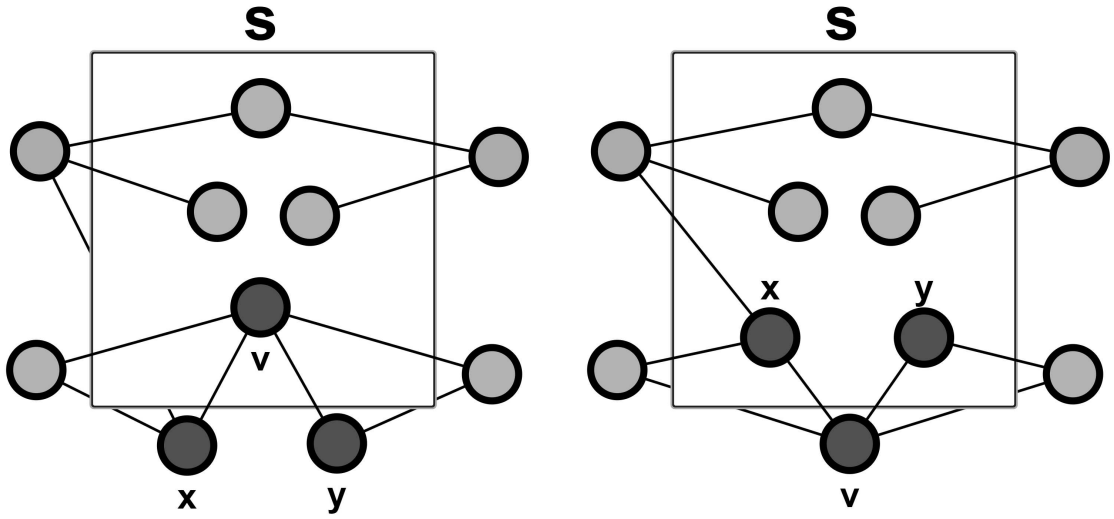


(a) Prije zamjene (b) Poslije zamjene

Slika 3.1: $(\omega, 1)$-zamjena $\mathcal{N}_1$

$O(1)$. Naime, neka je μ polje koje za svaki vrh $v_i \in V$ pamti razliku između njegove težine i sume težina njegovih susjeda, dakle $\mu(v_i) = w(v_i) - \Sigma_{y \in N_S(v_i)} w(y)$. Kad god mijenjamo rješenje, moramo promijeniti i polje μ na sljedeći način: kada ubacujemo/izbacujemo vrh $v \in V$ u/iz S-a, za svakog susjeda $y \in N_V(v)$ smanjimo/povećamo $\mu(y)$ za $w(v)$. Za zadano polje μ, kako bismo pronašli poboljšanje rješenja u $(\omega, 1)$-zamjeni, dovoljno je pogledati polje vrhova koji nisu u rješenju i pronaći neki od njih, $v \in V \setminus S$, za koji vrijedi $\mu(v) > 0$. Dakle, cijela $(\omega, 1)$-zamjena može se izvršiti u vremenu $O(|V \setminus S|)$.

Susjedstvo $(1, 2)$-zamjena sastoji se od izbacivanja jednog vrha $v \in S$ i ubacivanja dvaju vrhova $x, y \in V \setminus S$ u S. Poboljšanje se, jasno, događa u slučaju kada je $w(v) < w(x) + w(y)$.



(a) Prije zamjene (b) Poslije zamjene

Slika 3.2: $(1, 2)$-zamjena $\mathcal{N}_2$

Kako bismo osigurali raznolikost, ako je neki vrh v izbačen iz rješenja tijekom postupka perturbacije, smijemo ga ubaciti u rješenje tek nakon što smo proučili sve $(\omega, 1)$-zamjene koje ga ne uključuju. S druge strane, ako je v ubačen u rješenje tijekom perturbacije, onda ga ne smijemo izbaciti sve dok ne razmotrimo sve $(1, 2)$-zamjene koje ga ne uključuju.

Slike su izrađene pomoću internetske stranice kojoj se može pristupiti u [12].

Uvjeti prihvaćanja. Metoda `Prihvati(`S, S^*, S', i, *lokalni_w*, G, c_2, c_3, c_4`)` brine se da tijekom pretraživanja područja rješenja imamo dobar omjer između istraživanja i eksploatacije. Naime, kandidat za rješenje S' uvijek je prihvaćen kao novo trenutno rješenje ako je bolji od trenutnoga rješenja S, tj. ako je `Težina(`S'`)` $\leq$ `Težina(`S`)`. No, ponekad bismo htjeli prihvatiti i lošija rješenja ako shvatimo da već neko vrijeme nismo pronašli bolje. Stoga, pamtit ćemo lokalni brojač i koji vraćamo na početno stanje kad god je rješenje S poboljšano. Vrijednost brojača i smanjuje se ako je najbolja lokalna težina poboljšana

Pseudokod 3.1.5: Metoda Prihvati

ulazni podatci: trenutno rješenje S, najbolje rješenje S^*, kandidat za rješenje S', lokalni brojač i, najbolja lokalna težina *lokalni_w*, težinski neusmjereni graf G, parametri c_2, c_3 i c_4

izlazni podatci: novo trenutno rješenje S, novo najbolje rješenje S^*, promijenjeni lokalni brojač i, nova najbolja lokalna težina *lokalni_w*

```
1  Prihvati (S, S*, S', i, lokalni_w, G, c2, c3, c4)
2      ako ( Težina(S) ≤ Težina(S') ):
3          S = S';
4          i = 1;
5          ako ( lokalni_w ≤ Težina(S) ):
6              lokalni_w = Težina(S);
7              i −= |S|/c2;
8          ako ( Težina(S*) ≤ Težina(S) ):
9              S* = S;
10             i −= |S| c3
11     inače, ako ( i ≤ |S|/c2 ): i += 1 ;
12     inače:
13         lokalni_w = Težina(S);
14         S = Perturbiraj(c4, S, G);
15         i = 1;
16     vrati S, S*, i, lokalni_w
```

ili ako je pronađeno rješenje bolje od trenutnog najboljeg S^*. S druge strane, brojač ćemo povećati ako rješenje S nije poboljšano. Kada utvrdimo da za brojač vrijedi $i > |S|/c_2$, tada njegovu vrijednost vratimo na početnu i na rješenju pokrenemo metodu perturbacije. Time smo odlučili napustiti neki lokalni optimum trenutnoga rješenja u nadi da ćemo pronaći neki dalji, do kojega ne bismo mogli stići lokalnim pretraživanjem, ali koji možda ima bolji optimum. Pseudokod je vidljiv u 3.1.5.

Kako je ovaj algoritam heuristika, bitno je napomenuti da vraća približno rješenje, no ne nužno optimalno. Za primjer pogledajte u poglavlje 5.

3.2 Algoritam za rješavanje problema MWIS na stablima

Promotrimo što se događa s problemom kada se ograničimo na grafove koji čine stabla. Kao što je opisano u [2], vidjet ćemo jedan algoritam temeljen na paradigmi dinamičkoga programiranja, koji ima vremenesku složenost $O(|V|)$. Za razliku od heuristike ILS-VND, ovaj algoritam je egzaktan, tj. uvijek daje optimalno rješenje problema MWIS na stablima.

Neka je G stablo i neka je P_i podstablo stabla G s korijenom u vrhu $i \in V$. Definiramo sljedeće vrijednosti za svaki vrh i:

$$M(i) = \max\{w(I) \mid I \text{ je nezavisni skup u } P_i \text{ i } i \in I\},$$
$$M'(i) = \max\{w(I) \mid I \text{ je nezavisni skup u } P_i \text{ i } i \notin I\}.$$

Ove vrijednosti upravo predstavljaju najveće težine nezavisnih skupova u podstablu P_i koji, redom, sadrže i ne sadrže vrh i. Uočimo da se ove vrijednost mogu izračunati rekurzivno, na sljedeći način:

$$M(i) = w(i) + \sum_j M'(j), \text{ gdje je } j \text{ dijete vrha } i,$$
$$M'(i) = \sum_j \max\{M(j), M'(j)\}, \text{ gdje je } j \text{ dijete vrha } i,$$
$$M(list) = w(list) \text{ i}$$
$$M'(list) = 0.$$

Naime, vrijednost $M(i)$ je upravo maksimum svih nezavisnih skupova u P_i koji sadrže vrh i, zbog čega nijedan od tih nezavisnih skupova ne smije sadržavati nijedno od djece vrha i, a vrijednosti takvih skupova upravo odgovaraju vrijednosti $M'(i)$. Slično, ako naš nezavisni skup ne sadrži vrh i, onda vrijednost $M'(i)$ dobijemo upravo tako da uzmemo maksimum težina podstabala sve djece j, koja sadrže ili ne sadrže sam korijen j, jer time sigurno ne gubimo svojstvo nezavisnosti s obzirom na to da su različita podstabla s korijenom na istoj dubini međusobno disjunktna.

Jednom kada izračunamo vrijednosti $M(i)$ i $M'(i)$ za svaki vrh i, MWIS I možemo odrediti tako da se krećemo od korijena stabla G prema njegovim listovima. Korijen ubacujemo u skup I ako je $M(korijen) > M'(korijen)$. Svaki vrh j koji nije korijen uključujemo u I ako njegov roditelj nije u I i ako je $M(j) > M'(j)$. Vrijednosti M i M' za svaki vrh u potpunosti određuju nalazi li se on u nezavisnom skupu I ili ne, stoga je ovaj algoritam točan.

Promotrimo još i pseudokod ovoga algoritma te proučimo njegovu složenost.

Pseudokod 3.2.1: MWIS u stablu

ulazni podatci: težinsko neusmjereno stablo $G = (V, E, w)$
izlazni podatci: maksimalni nezavisni skup I

```
1  Tree (G)
2      za svaki ( list j u stablu G ) :
3          M(j) = w(j);
4          M'(j) = 0;
5      za svaki ( vrh i koji nije list u stablu G ) :
6          M(i) = Σ_j M'(j), gdje je j dijete vrha i;
7          M'(i) = Σ_j max{M(j), M'(j)}, gdje je j dijete vrha i;
8      I = ∅;
9      ako ( M(korijen) > M'(korijen) ) : I = I ∪ {korijen};
10     za svaki ( vrh j koji nije list u stablu G ) :
11         ako ( roditelj vrha j nije u I i M(j) > M'(j) ) : I = I ∪ {j} ;
12     vrati I
```

Uočimo da u pseudokodu 3.2.1 točno dvaput pregledavamo svaki vrh, jedanput u prvim dvjema petljama i drugi put u trećoj petlji. Stoga je vremenska složenost ovoga algoritma upravo $O(|V|)$. S obzirom na to da u najgorem slučaju MWIS može imati najviše $|V| - 1$ vrhova (korijen nije u skupu, a svi ostali vrhovi su njegova djeca i jesu u skupu), ovaj problem ima donju vremensku ogradu $\Omega(|V|)$, što znači da je ovaj algoritam optimalan do na konstantu.

Poglavlje 4

Implementacija

Programskome kodu može se pristupiti na GitHub-u:
`https://github.com/jbrigljevic/diplomski`.

4.1 Implementacija algoritma ILS-VND

Graf je implementiran kao lista susjedstava (s tim da je umjesto prave liste korištena struktura podataka std::vector iz standardne biblioteke C++-a, radi bolje prostorne lokalnosti). Vrhove grafa indeksiramo od 0 do nekog $n \in \mathbb{N}$ pa u nekom polju (std::vector) jednostavno pamtimo težinu svakog vrha.

Po uzoru na implementaciju koja je opisana u [1], naše rješenje S opisat ćemo pomoću permutacije svih vrhova, podijeljenih u tri bloka. Prvi blok sadržavat će $|S|$ vrhova koji se nalaze u rješenju, drugi blok čuvat će slobodne vrhove, a zadnji blok držat će vezane vrhove. Slobodni vrhovi su oni koji nemaju nijednog susjeda u rješenju, dok su vezani oni koji imaju bar jednog. Veličine prvih dvaju blokova pamtimo eksplicitno. Također, htjeli bismo moći efikasno micati vrhove iz jednog u drugi blok, stoga ćemo za svaki vrh dodatno pamtiti i njegovu poziciju u permutaciji. A kako bismo dodatno htjeli za svaki vrh znati je li slobodan ili vezan, pamtit ćemo još i broj njegovih susjeda koji se nalaze u rješenju. Sada je micanje vrhova iz jednog bloka u drugi izrazito jednostavno i izvedivo u konstantnome vremenu (samo treba pogledati u polje pozicija kamo koji vrh ide, a kako je polje implementirano kao std::vector, pristup svakom elementu je vremenski konstantan).

Kako bi svi ovi podatci bili ispravni, prilikom ubacivanja ili izbacivanja nekog vrha iz rješenja moramo promijeniti i informacije o rješenju, o samome vrhu i o njegovim susjedima. No zauzvrat sljedeće operacije možemo izvesti u konstantnome vremenu:

- provjeri je li rješenje neproširivo (do većeg nezavisnog skupa koji sadrži trenutno rješenje),

- provjeri je li vrh unutar rješenja,
- odredi vezanost nekog vrha za rješenje (dakle broj njemu susjednih vrhova unutar rješenja) i
- odaberi nasumični vrh uniformno.

Uz prethodno navedenu vremensku konstantnost micanja vrhova po polju rješenja, dodatno moramo i proći svim njegovim susjedima. Naime, za svakog susjeda moramo promijeniti njegovu vezanost, povećati ili smanjiti ovisno o tome dodaje li se ili izbacuje li se vrh, a ovisno o toj vezanosti možda moramo i pomaknuti vrh iz slobodnog bloka u vezani, ili obratno. Stoga, ukupna složenost ubacivanja ili izbacivanja jednog vrha je $O(|V|)$, jer je vrh u najgorem slučaju susjedan sa svim ostalima u grafu.

Već smo vidjeli kako efikasno pretražiti susjedstvo $(\omega, 1)$-zamjena, a ovakva implementacija rješenja dozvoljava nam sljedeći postupak za pretraživanje susjedstva $(1, 2)$-zamjena.

Provjeravat ćemo za svaki $x \in S$ je li dobar kandidat za $(1, 2)$-zamjenu (a to je onaj koji poveća težinu rješenja). U svakoj $(1, 2)$-zamjeni koja izbacuje x, vrhovi koje ubacujemo u rješenje moraju biti njegovi susjedi koji su vezani isključivo za x.

Iz tog razloga gradimo polje $L(x)$ koje se sastoji od svih vrhova vezanih isključivo za x. Među svim kandidatima u $L(x)$ tražimo dva različita nesusjedna vrha. To radimo tako da za fiksni vrh $v \in L(x)$ gledamo postoji li vrh $w \in L(x)$, $v \neq w$, koji ne pripada skupu $N(v)$, polju susjedstva vrha v. Ako pretpostavimo da su vrhovi sortirani po svojem nazivu (što možemo osigurati prije izvršavanja samog algoritma), takav par možemo pronaći prolazeći poljima $L(x)$ i $N(v)$ istovremeno. Naime, svi elementi od $L(x)$ bi se trebali pojaviti u istome redosljedu unutar $N(v)$. Ako nije tako, onda je nedostajući vrh upravo w koji tražimo.

Ovaj postupak pretraži cijelo susjedstvo $(1, 2)$-zamjena u vremenu $O(|V|^2)$. Naime, u najgorem slučaju moramo pregledati sve vrhove x unutar rješenja u vremenu $O(|V|)$. Za svaki od tih vrhova gradimo polje $L(x)$ u vremenu $O(st(x))$, što je omeđeno s $O(|V|)$. S obzirom na to da polja $L(x)$ i $N(v)$ pretražujemo istovremeno, to radimo opet u vremenu $O(st(v))$, što je opet omeđeno s $O(|V|)$. Ukupno imamo $O(|V|) \cdot 2 \cdot O(|V|) = O(|V|^2)$.

Sva navedena polja također su implementirana kao std::vector. Ona su objedinjena u jednu klasu *Solution*, a u svakom trenutku algoritma pamtimo tri objekta te klase: trenutno rješenje, najbolje rješenje i novo rješenje.

Promotrimo još i složenost ove heuristike. Kako algoritam zahtijeva neki uvjet zaustavljanja, u našem slučaju maksimalni broj iteracija, ocijenit ćemo jednu iteraciju algoritma. Ona se sastoji od triju metoda: `Perturbiraj`, `LokalnoPretraživanje` i `Prihvati`.

Kako nam implementacija rješenja S omogućava odabir nasumičnog vrha u konstantnome vremenu, metodu `Perturbiraj(`k, S, G`)` možemo ocijeniti na sljedeći način. Točno k nasumičnih vrhova odaberemo i ubacimo u vremenu $O(k|V|) = O(|V|)$, za svaki od kojih moramo dodatno izbaciti najviše $O(|V|)$ vrhova, gdje izbacivanje svakog izvršimo u vremenu $O(|V|)$, dakle ukupno je to $O(|V|^2)$.

Metodu `LokalnoPretraživanje` nažalost ne možemo a priori ocijeniti. Naime, unaprijed ne znamo koliko puta će se u svakome susjedstvu tražiti rješenje, jer ovisno o tome nađe li se bolje ponovno će se ići u potragu za boljim. Tako da samo držimo na umu da pretraživanje susjedstva $(\omega, 1)$-zamjena možemo izvršiti u vremenu $O(|V \setminus S|)$, dok se pretraživanje susjedstva $(1, 2)$-zamjena može izvršiti u vremenu $O(|V|^2)$. Svako pretraživanje susjedstva dodatno zahtijeva i korak proširivanja, koji u najgorem slučaju dodaje sve vrhove u rješenje (jasno je da je ovo izazito loša ocjena u prosječnom slučaju) u vremenu $O(|V|^2)$. Sve u svemu, gornja ograda za pretraživanje nekog susjedstva jednom je sigurno $O(|V|^2)$.

Još nam ostaje i metoda `Prihvati`, u kojoj je jedini značajan korak dodatan korak perturbacije, opet u vremenu $O(|V|^2)$.

4.2 Implementacija algoritma na stablima

Graf je implementiran sasvim isto kao i u slučaju općenitih grafova. Pretpostavlja se da je zato sam primjer grafa uistinu stablo, i to takvo da su na većim dubinama veće oznake vrhova. Također se pretpostavlja da su vrhovi u listi susjedstva sortirani uzlazno, jer je tada vrh na prvom mjestu liste upravo roditelj, dok su svi ostali djeca nekog vrha. Polja M i M' implementirana su kao std::vector, dok je rješenje I oblika std::set.

Poglavlje 5

Testiranje

5.1 ILS-VND algoritam

Provedimo sada testove da vidimo kako funkcioniraju ovi algoritmi u praksi. Započnimo s ILS-VND-om, testovi su provedeni na dobro poznatim mjerodavnim primjercima iz skupova podataka DIMACS i BHOSLIB, prikupljenih sa stranice [10]. U ovim primjercima, svakom vrhu i pridružujemo težinu $(i \mod 200) + 1$, kako su pridružene težine vrhovima u literaturi [9]. Kako su ovi podatci namijenjeni za rješavanje problema CLIQUE, testove provodimo na komplementima navedenih primjera.

Testovi su provedeni na procesoru Ryzen 5 3600 s frekvencijom 3.6 GHz/4.2 GHz (Turbo Core) i 16 GB RAM-a, na Windows 10 OS-u. Algoritam je pozvan na svakome primjerku sto puta od kojih je onda uzeta njihova srednja vrijednost. Algoritam je napisan u C++-u i kompiliran s GCC 13.1.0 i optimizacijskom zastavicom '-O3'. Što se tiče parametara navedenih u pseudokodovima, najveći broj iteracija postavljen je na 2 milijuna, $c_1 = 1$, $c_2 = 3$, $c_3 = 4$, a $c_4 = 2$. Iz literature [9] preuzeti su podaci o najboljim poznatim rješenjima za ovaj problem. Izvršavanje koda završeno je prilikom dostizanja tih rezultata kako bi se skratilo ukupno vrijeme potrebno za testiranje svih primjera. U svakom izvršavanju, osim vremena izvršavanja i broja iteracija do maksimalnog broja iteracija (ili do najboljeg poznatog rješenja iz literature), zapamćeni su vrijeme izvršavanja i broj iteracija do najboljeg pronađenog rješenja. Rezultati su vidljivi u tablicama 5.1 i 5.2.

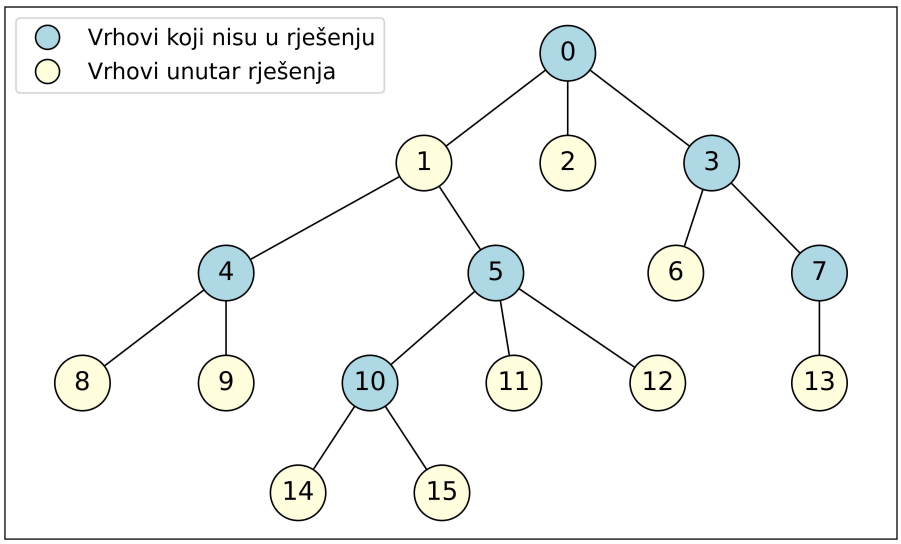
Što se tiče testnog skupa BHOSLIB, u njemu se broj vrhova u grafu kreće između 450 i 1534, gdje primjeri s prefiksom 'frb30' imaju najmanje vrhova, a s povećanjem broja u prefiksu povećava se i broj vrhova u grafu. Jasno vidimo kako veći broj vrhova utječe na povećanje vremena izvršavanja i povećanje broja potrebnih iteracija. Također u nekim primjerima vidimo značajnu razliku između ukupnog broja iteracija i broja iteracija do najboljeg pronađenog rješenja. Ova činjenica nam govori da bismo osim maksimalnog broja iteracija mogli dodati i uvjet zaustavljanja koji prati koliko iteracija nije pronađeno bolje

rješenje i zaustavlja algoritam kada se prijeđe neki prag. Također, promatrajući prosječnu težinu rješenja i najbolje pronađeno rješenje vidimo da algoritam prilično konzistentno vraća izrazito slična rješenja, a uspoređujući te rezultate s najboljim poznatim rješenjima iz literature možemo zaključiti da se ono najčešće poklapa s najboljim rješenjem koje je algoritam ILS-VND pronašao.

Isti zaključci vrijede i u testnom skupu DIMACS. Ono što bih u njemu istaknuo jest primjerak 'C4000-5' koji ima 4000 vrhova, a u kojemu vidimo da se ukupno vrijeme izvršavanja i vrijeme do najboljeg pronađenog rješenja ne razlikuju značajno. Također, u njemu broj iteracija najčešće ni ne stiže do određenog maksimuma. Za razliku od njega, imamo primjerak 'hamming10-2' s 1024 vrhova, koji izrazito brzo pronađe dobar lokalni maksimum, ali onda zapne u njemu i izvršavanje se nastavi do maksimalnog broja iteracija bez promjene. Kao što je i ranije napomenuto, postoje i primjeri za koje znamo koje je njihovo optimalno rješenje, no algoritam ga ne pronalazi uvijek, npr. 'brock800-4' ima poznato optimalno rješenje 2971 (pronađeno egzaktnim rješavačem CPLEX-om, pogledajte u [9]), dok je prosjek naše heuristike nešto ispod te vrijednosti. Dakle, algoritam često pronalazi izrazito dobro rješenja, kako u grafovima s malim brojem vrhova, tako i u onima s velikim brojem vrhova. Veći broj vrhova utječe na brzinu izvršavanja i broj iteracija, ali ne nužno. Naime, kao što smo vidjeli postoje grafovi s velikim brojem vrhova koji brzo pronađu najbolje poznato rješenje, dok neki drugi mogu zapeti u lokalnom maksimumu. Uspoređujući pronađene rezultate s rezultatima iz [9], možemo se uvjeriti da algoritam najčešće pronalazi najbolje poznato rješenje, a ponekad i ono optimalno, kada je takvo poznato.

5.2 Algoritam na stablima

Kako je ovaj algoritam relativno trivijalan i izuzetno brz, pogledajmo samo na jednostavnome primjeru rezultat koji je dao. Težine su iste kao i kod ILS-VND algoritma. Grafički prikaz ostvaren je pomoću Python 3.10.1.



Slika 5.1: Primjer rješenja problema MWIS na stablima

BHOSLIB						
Naziv	Prosječno vrijeme (s)	Prosječan broj iteracija	Prosječno vrijeme do najboljeg rješenja (s)	Prosječan broj iteracija do najboljeg rješenja	Prosječna težina rješenja	Najbolje pronađeno rješenje
frb30-15-1	0.52	47270	0.52	47270	2990	2990
frb30-15-2	0.19	18286	0.19	18286	3006	3006
frb30-15-3	0.18	17511	0.18	17511	2995	2995
frb30-15-4	0.09	8809	0.09	8809	3032	3032
frb30-15-5	0.07	6084	0.07	6084	3011	3011
frb35-17-1	3.67	284743	3.67	284743	3650	3650
frb35-17-2	6.77	522213	5.83	450065	3736.84	3738
frb35-17-3	0.55	40291	0.55	40291	3716	3716
frb35-17-4	1.91	141188	1.91	141188	3683	3683
frb35-17-5	0.77	59153	0.77	59153	3686	3686
frb40-19-1	19.42	1247571	12.36	794363	4060.88	4063
frb40-19-2	2.55	170837	2.55	170837	4112	4112
frb40-19-3	10.90	718326	9.54	628688	4114.7	4115
frb40-19-4	3.74	253913	3.74	253913	4136	4136
frb40-19-5	2.05	136986	2.05	136986	4118	4118
frb45-21-1	21.13	1212954	14.56	835672	4755.75	4760
frb45-21-2	10.28	578686	9.05	509767	4782.94	4784
frb45-21-3	4.51	263132	4.50	263132	4765	4765
frb45-21-4	1.31	76300	1.31	76300	4799	4799
frb45-21-5	10.85	593691	10.27	561906	4778.69	4779
frb50-23-1	37.38	1890027	14.55	735887	5483.41	5494
frb50-23-2	34.44	1680804	16.02	782352	5453.83	5462
frb50-23-3	35.40	1789229	18.10	914791	5482.57	5486
frb50-23-4	39.99	2000000	13.68	683684	5444.7	5453
frb50-23-5	23.96	1164857	17.54	852743	5490.6	5498
frb53-24-1	23.72	1051117	17.70	784175	5663.76	5670
frb53-24-2	37.10	1722687	19.77	918357	5694.49	5707
frb53-24-3	39.02	1793751	22.15	1018176	5629.87	5655
frb53-24-4	38.64	1748691	14.45	653346	5700.85	5714
frb53-24-5	42.91	1960014	20.03	915065	5646.04	5659

BHOSLIB						
Naziv	Prosjek (s)	Ukupno iteracija	Prosječno vrijeme do najboljeg rješenja (s)	Broj iteracija do najboljeg rješenja	Prosječna težina rješenja	Najbolje nađeno rješenje
frb59-26-1	47.53	1877759	22.96	907549	6572.14	6591
frb59-26-2	50.60	1957446	22.62	875471	6585.34	6645
frb59-26-3	51.76	1963280	19.97	757479	6580.99	6608
frb59-26-4	44.66	1742687	18.47	720147	6569.67	6592
frb59-26-5	51.89	1993920	25.90	995212	6551.11	6584

Tablica 5.1: Rezultati na podacima iz skupa BHOSLIB

DIMACS						
Naziv	Prosjek (s)	Ukupno iteracija	Prosječno vrijeme do najboljeg rješenja (s)	Broj iteracija do najboljeg rješenja	Prosječna težina rješenja	Najbolje nađeno rješenje
brock200-1	< 0.01	90	< 0.01	92	2821	2821
brock200-2	< 0.01	58	< 0.01	58	1428	1428
brock200-3	< 0.01	89	< 0.01	89	2062	2062
brock200-4	< 0.01	159	< 0.01	160	2107	2107
brock400-1	< 0.01	944	< 0.01	944	3422	3422
brock400-2	0.01	1178	0.01	1178	3350	3350
brock400-3	0.03	3105	0.03	3105	3471	3471
brock400-4	0.20	20451	0.20	20451	3626	3626
brock800-1	0.02	1192	0.02	1192	3121	3121
brock800-2	0.08	4436	0.08	4436	3043	3043
brock800-3	0.05	3128	0.05	3128	3076	3076
brock800-4	30.10	1688756	5.55	311225	2970.31	2971
c-fat200-1	< 0.01	31	< 0.01	32	1284	1284
c-fat200-2	< 0.01	18	< 0.01	21	2411	2411
c-fat200-5	< 0.01	5	< 0.01	6	5887	5887
c-fat500-1	< 0.01	92	< 0.01	92	1354	1354
c-fat500-10	< 0.01	7	< 0.01	8	11586	11586
c-fat500-2	< 0.01	47	< 0.01	47	2628	2628
c-fat500-5	< 0.01	13	< 0.01	14	5841	5841
C1000-9	1.66	90540	1.66	90540	9254	9254
C125-9	< 0.01	184	< 0.01	186	2529	2529
C2000-5	2.43	49542	2.43	49542	2466	2466
C2000-9	56.86	1808500	21.30	677618	10968	10999
C250-9	0.03	3611	0.03	3611	5092	5092
C4000-5	63.41	613978	59.36	574685	2791.56	2792
C500-9	0.09	8019	0.09	8019	6955	6955
DSJC1000-5	0.03	1148	0.03	1148	2186	2186
DSJC500-5	0.02	1415	0.02	1415	1725	1725
gen200-p0-9-44	< 0.01	90	< 0.01	90	5043	5043
gen200-p0-9-55	0.04	6461	0.04	6508	5416	5416
gen400-p0-9-55	0.03	2606	0.03	2606	6718	6718
gen400-p0-9-65	0.06	6279	0.06	6279	6940	6940
gen400-p0-9-75	0.01	834	< 0.01	834	8006	8006

DIMACS						
Naziv	Prosjek (s)	Ukupno iteracija	Prosječno vrijeme do najboljeg rješenja (s)	Broj iteracija do najboljeg rješenja	Prosječna težina rješenja	Najbolje nađeno rješenje
hamming10-2	30.37	940685	0.02	3592	50418	50512
hamming10-4	3.96	196720	3.96	196720	5129	5129
hamming6-2	< 0.01	1	< 0.01	794	1072	1072
hamming6-4	< 0.01	1	< 0.01	2	134	134
hamming8-2	7.82	900004	< 0.01	17	10886	10976
hamming8-4	< 0.01	2	< 0.01	3	1472	1472
johnson16-2-4	< 0.01	30	< 0.01	30	548	548
johnson32-2-4	< 0.01	220	< 0.01	220	2033	2033
johnson8-2-4	< 0.01	5	< 0.01	6	66	66
johnson8-4-4	< 0.01	1	< 0.01	1	511	511
keller4	< 0.01	97	< 0.01	97	1153	1153
keller5	0.04	2464	0.04	2464	3317	3317
keller6	20.55	298321	20.55	298321	8062	8062
MANN-a27	0.89	38927	0.89	38927	12283	12283
MANN-a45	114.97	1843932	47.45	760827	34263.2	34265
p-hat1000-1	0.05	1608	0.05	1608	1514	1514
p-hat1000-2	< 0.01	63	< 0.01	63	5777	5777
p-hat1000-3	0.03	805	0.03	805	8111	8111
p-hat1500-1	0.07	1138	0.07	1138	1619	1619
p-hat1500-2	0.02	314	0.02	314	7360	7360
p-hat1500-3	0.05	898	0.05	898	10321	10321
p-hat300-3	< 0.01	106	< 0.01	106	3774	3774
p-hat500-1	< 0.01	180	< 0.01	180	1231	1231
p-hat500-2	< 0.01	76	< 0.01	76	3920	3920
p-hat500-3	< 0.01	248	< 0.01	252	5375	5375
p-hat700-1	< 0.01	136	< 0.01	137	1441	1441
p-hat700-2	< 0.01	46	< 0.01	46	5290	5290
p-hat700-3	< 0.01	326	< 0.01	326	7565	7565
san1000	0.14	424	0.14	426	1716	1716
san200-0-7-1	< 0.01	329	< 0.01	332	3370	3370
san200-0-7-2	< 0.01	366	< 0.01	369	2422	2422
san200-0-9-1	0.14	9897	0.14	10140	6825	6825
san200-0-9-2	< 0.01	679	< 0.01	803	6082	6082
san200-0-9-3	< 0.01	815	< 0.01	815	4748	4748

DIMACS						
Naziv	Prosjek (s)	Ukupno iteracija	Prosječno vrijeme do najboljeg rješenja (s)	Broj iteracija do najboljeg rješenja	Prosječna težina rješenja	Najbolje nađeno rješenje
san400-0-5-1	0.03	406	0.03	406	1455	1455
san400-0-7-1	1.82	36664	1.82	36664	3941	3941
san400-0-7-2	0.73	15505	0.73	15505	3110	3110
san400-0-7-3	0.04	1444	0.04	1444	2771	2771
san400-0-9-1	18.79	642420	7.56	262586	9486.25	9776
sanr200-0-7	< 0.01	161	< 0.01	161	2325	2325
sanr200-0-9	< 0.01	113	< 0.01	113	5126	5126
sanr400-0-5	< 0.01	251	< 0.01	251	1835	1835
sanr400-0-7	< 0.01	505	< 0.01	505	2992	2992

Tablica 5.2: Rezultati na podacima iz skupa DIMACS

Zaključak

U ovome radu bavili smo se proučavanjem problema MWIS s teorijskog i s praktičnog stajališta. Istražili smo vremensku složenost problema IS i nekih njegovih optimizacijskih varijacija te smo ih usporedili sa sličnim problemima na grafovima, a to su VC i CLIQUE. Zaključili smo da je problem MWIS NP-težak, zbog čega se ne očekuje da postoji efikasan egzaktan algoritam za njegovo rješavanje. Iz tog razloga, u praktičnom dijelu rada, posegnuli smo približnome rješenju problema u slučaju općenitih grafova, pomoću heuristike ILS-VND. Heuristika se koristi paradigmama ILS i VND te dozvoljava brzu izgradnju početnog rješenja u čijim se raznim susjedstvima zatim traže bolja rješenja. Taj algoritam testiran je na testnim podacima iz baza podataka BHOSLIB i DIMACS te su rezultati uspoređeni s onima iz literature. Pokazalo se da heuristika može izrazito brzo pronaći dobra rješenja, a vrlo često i optimalna ili najbolja poznata rješenja. Unatoč tome, jasno je da su neka rješenja samo približna i da nema garancije za njihovim dostizanjem.

Osim heuristike ILS-VND, proučili smo i postoji li bolje rješenje na jednoj posebnoj vrsti grafova, a to su stabla. Zaključili smo da koristeći se paradigmom dinamičkog programiranja možemo definirati egzaktni algoritam koji rješava problem MWIS na stablima, s linearnom vremenskom složenošću u broju vrhova stabla. Pokazali smo da je algoritam optimalan, do na konstantu, u smislu vremena izvršavanja, a njegova implementacija je izrazito jednostavna.

Kao što smo komentirali, heuristika ILS-VND mogla bi se dodatno prilagoditi dodavanjem drugih uvjeta zaustavljanja, kao što je npr. broj iteracija bez promjene rješenja, ovisno o tome je li nam potrebno približno rješenje u kratkom roku, ili želimo li pronaći što bolje rješenje, trošeći više vremena. Također, koraci lokalnog pretraživanja, perturbacije i prihvaćanja svi se mogu prilagoditi potrebama. Perturbacija može mijenjati rješenje u većoj ili manjoj mjeri. Uvjeti prihvaćanja mogu se prilagoditi ovisno o tome koliko kvalitetno rješenje tražimo i koliko vremena želimo potrošiti istražujući neko susjedstvo. Dok u korak lokalnog pretraživanja možemo dodati i razna druga susjedstva koja se možda brže pretražuju, ili imaju bolja rješenja. Sve ove tri točke mjesto su za potencijalno poboljšanje.

Bibliografija

[1] D. Andrade, M. Resende i R. Werneck, *Fast Local Search for the Maximum Independent Set Problem*, Journal of Heuristics, sv. 18, svibanj 2008, str. 220–234.

[2] G. H. Chen, M. T. Kuo i J. P. Sheu, *An Optimal Time Algorithm for Finding a Maximum Weight Independent Set in a Tree*, BIT, sv. 28, 1988, str. 353–356.

[3] J. Chou, J. Kim i D. Rotem, *Energy-Aware Scheduling in Disk Storage Systems*, Proceedings - International Conference on Distributed Computing Systems, srpanj 2011, str. 423 – 433.

[4] M. R. Garey i D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, str. 109–120, W. H. Freeman, New York, 1979.

[5] J. L. Gross, J. Yellen i P. Zhang (ur.), *Handbook of Graph Theory*, Chapman and Hall/CRC, 2013.

[6] D. Jungnickel, *Graphs, Networks and Algorithms*, Springer, 2013.

[7] H. R. Lourenço, O. C. Martin i T. Stützle, *Handbook of Metaheuristics*, pogl. Iterated Local Search: Framework and Applications, str. 363–397, Springer US, Boston, MA, 2010.

[8] J. Moreno-Pérez, N. Mladenovic, B. Melian i I. Amo, *Variable Neighbourhood Search*, Inteligencia Artificial, sv. 7, siječanj 2006, str. 71–86.

[9] B. Nogueira, R. Pinheiro i A. Subramanian, *A hybrid iterated local search heuristic for the maximum weight independent set problem*, Optimization Letters, sv. 12, svibanj 2018, str. 567–583.

[10] Ryan A. Rossi i Nesreen K. Ahmed, *The Network Data Repository with Interactive Graph Analytics and Visualization*, AAAI, 2015, `https://networkrepository.com`, posjećena 11. 8. 2023.

[11] M. Vuković, *Složenost algoritama - predavanja i vježbe*, 2020, https://www.pmf.unizg.hr/_download/repository/SA-skripta-2019.pdf, posjećena 30. 8. 2023.

[12] yWorks, *yEd Live*, 2023, https://www.yworks.com/yed-live/, posjećena 21. 8. 2023.

Sažetak

Problem maksimalnog težinskog nezavisnog skupa u grafu jedan je NP-težak problem optimizacije. On ima primjene u stvarnome svijetu, koje zahtijevaju brze algoritme koji pronalaze kvalitetna rješenja. Osim toga, u bliskoj je vezi s problemima maksimalne težinske klike i maksimalnog težinskog vršnog pokrivača.

U ovome radu definirani su navedeni problemi i iskazane su njihove međusobne veze. Proučena je njihova vremenska složenost u teorijskom smislu i iskazani su relevantni rezultati. Kako je problem maksimalnog težinskog nezavisnog skupa NP-težak, u njegovu rješavanju posežemo za heuristikom kako bismo na brz način pronašli kvalitetna rješenja u općenitom slučaju, a proučavamo i kako se problem može egzaktno riješiti u slučaju stabla.

Heuristika koju smo proučili temeljena je na dvjema paradigmama: ponavljanom lokalnom pretraživanju i varijabilnim spustom uz korištenje susjedstava. Ponavljano lokalno pretraživanje heuristika je u kojoj se trudimo pronaći sve bolje rješenje pretraživanjem neke lokalne okoline trenutnoga rješenja. Ideja je da se na taj način trudimo doseći lokalni optimum pod pretpostavkom određene neprekidnosti područja pretraživanja. Kako na taj način možemo zapeti u lokalnom optimumu, u navedeni postupak uključujemo korak perturbacije koji nastoji pronaći ostale lokalne optimume. Da bismo uspjeli efikasno istražiti blizinu trenutnoga rješenja, potreban nam je dobar pojam bliskih rješnja. Iz tog razloga posežemo varijabilnom spustu uz korištenje susjedstava, u kojemu definiramo par susjedstava unutar kojih tražimo bolja rješenja od trenutnoga. Navedeni algoritam je implementiran i testiran na bazama podataka BHOSLIB i DIMACS, na kojima je uočeno da daje kvalitetna rješenja u izrazito malome vremenu.

Problem se može ograničiti na stabla i u tom slučaju ima rješenje vremenske složenosti $\Omega(|V|)$, gdje je $|V|$ broj vrhova u grafu. Algoritam koji ga rješava koristi se paradigmom dinamičkoga programiranja i ima izrazito jednostavnu implementaciju.

Summary

The maximum weight independent set is an NP-hard optimization problem. It has some real-world applications, which require quick algorithms that find quality solutions. Also, it is in a close relationship with the maximum weight clique problem and the maximum weight vertex cover problem.

In this thesis, the listed problems are defined and their relationships are expressed and proven. Their time complexity is studied in a theoretical sense and the relevant results are expressed. Since the maximum weight independent set is NP-hard, quality solutions in the general case are found by solving it in a heuristic approach, while the problem is also solved exactly in the case of trees.

The studied heuristic is based on two paradigms: iterated local search and variable neighborhood descent. Iterated local search is a heuristic in which we try to find better solutions by searching the local neighborhood of the current solution. The idea is that by doing so we try to reach a local optimum under the assumption of continuity of the search area. Since this can lead to the algorithm getting stuck in a local optimum, a perturbation step is included which attempts to find other local optima. In order to efficiently search the vicinity of the current solution, a good notion of similar solutions is required. For that goal, variable neighborhood descent is defined alongside two notions of neighborhoods which are deemed to contain similar solutions. The algorithm is implemented and tested on the BHOSLIB and DIMACS databases, where it is observed that it quickly finds quality solutions.

The problem can be restricted to trees, where its time complexity is $\Omega(|V|)$, where $|V|$ is the number of vertices in the graph. The algorithm which solves it uses a dynamic programming approach and has a simple implementation.

Životopis

Rođen sam 22. ožujka 2000. godine u Velikoj Gorici, gdje sam završio osnovnu školu. Pohađao sam XV. gimnaziju u Zagrebu od 2014. do 2018. godine. 2018. godine upisujem preddiplomski studij matematike na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. 2021. godine završavam navedeni studij i upisujem diplomski studij Računarstvo i matematika na istome fakultetu.