Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Emil Avdič

Razporejanje predavanj na konferenci

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: dr. Andrej Brodnik
Somentor: dr. Gregor Papa

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

NP-polni problemi v računalništvu in informatiki predstavljajo izredno pomembno poglavje in njihov pomen se s prihodom ogromnih količin podatkov samo še krepi. Namreč, obdelave podatkov, ki zahtevajo več kot linearno časovno zahtevnost so v praksi pogosto neizvedljive, saj nimamo ne potrebnih virov ne časa. To pomeni, da predstavljajo nedeterministično polinomski problemi še dodatni izziv.

V diplomski nalogi predlagajte primerne hevristične rešitve za izdelavo urnika sej znanstvene konference. Urnik naj predavanja razvrsti v seje in pri tem upošteva praktične zahteve o velikosti sej, številu sej in homogenosti sej (predavanja z isto tematiko). Predlagane hevristike primerjajte glede na oceno razporeditve predavanj v seje.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Emil Avdič sem avtor diplomskega dela z naslovom:

Razporejanje predavanj na konferenci (angl. *Task scheduling at the conference*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika in somentorstvom dr. Gregorja Papa,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 10. marca 2016 Podpis avtorja:

Zahvala

Zahvala gre mentorju dr. Andreju Brodniku za vse nasvete pri pisanju diplomske naloge in za vso strokovno pomoč, ki mi jo je nudil. Zahvaljujem se tudi somentorju z inštituta Jožef Stefan dr. Gregorju Papa za ideje in strokovno pomoč pri praktičnih zadevah.

Posebej se zahvaljujem še mojemu dekletu Niki in njeni družini za pomoč pri diplomski nalogi in za lektorsko delo. Na koncu bi se rad zahvalil še moji družini za podporo med celotnim študijem.

Kazalo

Slike

Algoritmi

Seznam uporabljenih kratic

kratica	angleško	slovensko
NP	nondeterministic polynomial time	nedeterministično polinomski čas
XML	extensible markup language	razširljiv označevalni jezik

Povzetek

Naslov: Razporejanje predavanj na konferenci

V nalogi bomo predstavili nekaj algoritmov, ki jih je mogoče uporabiti za izdelavo urnika konference. Najprej predstavimo t.i. požrešni algoritem. Temu sledijo še zahtevnejši algoritmi, ki izhajajo iz problema pokritja in problema nahrbtnika. Vse tri algoritme testiramo na treh, različno zgrajenih konferencah, s pripadajočimi članki. Prvo testno konferenco smo kreirali povsem poljubno z izmišljenimi podatki. Druga testna konferenca je bila psevdo naključna, ki jo program zgenerira sam glede na parametre, ki jih podamo. Tretja je konferenca, kjer podatke dobimo iz dejanskih preteklih konferenc ali iz prihajajoče konference.

Izbrane nabore člankov znotraj testnih konferenc smo uredili v seje s pomočjo treh omenjenih algoritmov. Oceno urejanja člankov konference smo ocenili preko izračuna kromatičnosti in praznih prostorov v seji. Glede na naše kriterije ocenjevanja se za najboljšega izkaže algoritem s hevristiko nahrbtnika, ki izhaja iz algoritma prvega prileganja. Na koncu predstavimo rezultate primerjave.

Ključne besede: generiranje urnika, NP-problem, problem nahrbtnika, prolem pokritja, požrešni algoritem.

Abstract

Title: Task scheduling at the conference

We will be introducing a few algorithms, that can be used for creating a schedule of a conference. Firstly we introduce greedy algorithm. We follow up with a few more complicated algorithms, that originate from the bin packing problem and the set cover problem. All three algorithms are then tested on three differently build conferences with their corresponding articles. First conference is created randomly with fictional data. Second conference is created with a pseudo random function. Third conference has real life data from a conference that is yet to come. Selected sets articles are sorted with the help of previously mentioned algorithms. Quality of algorithms is measured through a function of chromaticity and empty spaces in sessions. According to our criteria, the best algorithm is the one with the heuristics of the bin packing problem, coming from the first fit algorithm. At the end we present the results of testing.

Keywords: generating schedule, NP-problem, backsack problem, set cover problem, greedy algorithm.

Poglavje 1

Uvod

Osnova za nalogo je bila spletna aplikacija, ki je bila napisana za android naprave, ki uporabnikom omogoča vpogled v urnik konference. Ker je aplikacija napisana brez uporabniškega vmesnika za urejanje, je za vsako konferenco potrebno spreminjati XML dokument, ki jo opisuje. To osnovno aplikacijo smo najprej nadgradili tako, da shranjuje vse podatke o konferenci v podatkovno bazo in tako odstrani potrebo po XML dokumentu. Nato smo aplikacijo nadgradili še z uporabniškim vmesnikom, ki uporabnikom omogoča, da preko spleta vnašajo in spreminjajo podatke konference in izdelajo urnik konference. Tako lahko preko omenjenega spletnega vmesnika urnik konference spreminjajo tudi ljudje, ki niso vešči dela z XML dokumenti. Z našimi nadgradnjami osnovne aplikacije smo rešili zgolj problem vizualizacije in urejanja konference. Še vedno pa je potrebno ročno izdelati urnik konference, torej razporediti vse članke konference v seje, pri čemer je potrebno upoštevati določene kriterije. Izdelava programa za avtomatsko razvrščanje znanstvenih člankov v urnik konference je tema te diplomske naloge.

1.1 Struktura diplomske naloge

Prvo poglavje diplomske naloge je uvod, kjer je pojasnjeno ozadje in ideja za diplomsko nalogo.

V drugem poglavje podrobno opišemo problem, ki ga v diplomski nalogi raziskujemo ter parametre za opis konference in kakovostno funkcijo, ki pove, kako dobra je ureditev. Predstavimo več algoritmov, ki rešujejo podoben primer.

V tretjem poglavju razložimo implementacijo vseh algoritmov, ki so opisani v drugem poglavju. Razložimo še vse funkcije, ki so bile definirane za implementacijo problema in rešitve.

V četrtem poglavju predstavimo empirične rezultate. Pokažemo, kako različni algoritmi razporedijo konference. Na grafih predstavimo uspešnost različnih algoritmov na različno velikih konferencah.

Poglavje 2

Problem urnika in rešitve

2.1 Urnik za konferenco

Sestavljanje urnika je v osnovi zapleteno opravilo. Potrebno je najti ustrezen algoritem, ki bi veliko število člankov različnih tematik uredil na najboljši način, da so članki iste tematike v isti seji in da v seji ni praznih prostorov. Želimo si najti kar najbolj preprost algoritem, ki dobro deluje pri velikem in majhnem številu člankov.

Če za postavitev člankov v urnik ne bi bilo nobenih zahtev, bi bilo generiranje urnika zelo preprosto. Vse članke bi enega za drugim postavili v urnik, vendar bi bil tak urnik zelo naključen in neuporaben. Pri izdelavi urnika imamo nekaj zahtev in kriterijev:

- Želje udeležencev konference so navadno take, da je potrebno v najkrašjem času slišati kar največ predstavitev člankov s podobno tematiko na enem mestu, kar pomeni, da želimo imeti članke z enako temo enega za drugim.

- Želje organizatorja so, sestaviti tak urnik, kjer je natančno določeno število sej. Slednje so seveda odvisne od števila predavalnic, ki so na voljo in časa trajanja konference.

- Dodatna zahteva je navadno tudi, da imamo v primeru predavatelja, ki

ima več člankov, njegove predstavitve ali skupaj ali pa čim bolj narazen, vendar pa v našem primeru tega kriterija nismo upoštevali.

Kot lahko sklepamo iz naštetih kriterijev za izdelavo dobrega urnika seje, člankov ne uvrščamo neposredno v urnik, ampak jih prej razporedimo v seje. Dolžino seje določi organizator. Seja ima vnaprej določene teme, glede na katere so članki lahko vanjo uvrščeni.

Vsak članek ima vsaj eno ali več tem. Več tem kot jih članek ima, lažje ga je razvrstiti v sejo, saj ga lahko razporedimo v več sej. Če ima naprimer članek teme Algoritmi, Optimizacija, Rekurzivne funkcije, ga lahko razporedimo v sejo, ki ima določeno temo ali Algoritmi ali Optimizacija ali Rekurzivne funkcije. Kakšen je vrstni red člankov znotraj seje, ni bil predmet našega dela.

2.2 Priprava urnika

Napisati preprost algoritem, ki izdela urnik v primeru, ko nimamo omejitev, ni zapleteno. V našem primeru pa želimo napisati algoritem z vhodnimi parametri, ki so za vsako konferenco različni. Konference se med seboj lahko razlikujejo po številu člankov, številu različnih tem, velikostjo sej in številu vseh sej.

Konferenco lahko opišemo z naslednjimi parametri:

- n: število vseh člankov, ki jih razvrščamo,
- k: število vseh različnih tem člankov konference,
- p: število člankov, ki jih lahko razvrstimo v eno sejo,
- s: število vseh sej na konferenci.

Število člankov in število različnih tem sta parametra, ki sta odvisna od posamezne konference. Velika konferenca ima navadno tudi veliko člankov, vendar pa ni nujno, da število člankov vpliva na število tem v konferenci.

Lahko imamo primer velikega števila člankov in zgolj nekaj tem, ali pa malo število člankov z veliko različnimi temami. Vsekakor pa je število člankov in število tem določeno za vsako konferenco vnaprej.

Število člankov in tem sta parametra, ki bistveno manj vplivata na naš problem, kot število in velikost sej. Slednja namreč določata, kako morajo biti članki razporejeni. Število sej je odvisno od števila člankov v konferenci in ne sme preseči maksimalnega števila sej, ki ga določi organizator konference. Omejeno je z enačbo $\lceil n/p \rceil \leq \sigma_{min} \leq s \leq \sigma_{max} \leq n$. Ne more biti manjše, kolikor jih je minimalno potrebno za razporeditev vseh člankov $\lceil n/p \rceil$ in ni smiselno, da jih je več, kot bi jih bilo potrebno v primeru postavitve vsakega članka posebej v svojo sejo (n). σ_{min} in σ_{max} je minimalno in maksimalno število sej, ki ga določi organizator. Ta določi tudi velikost sej, kar neposredno vpliva na minimalno število sej. Večja kot je seja, manjše je število potrebnih sej. Manjše kot so seje, več jih potrebujemo za razporeditev vseh člankov vanje.

2.2.1 Kakovostna funkcija

Za ocenjevanje dobre razporejenosti člankov v seje in posledično dobrega urnika, potrebujemo objektivno oceno, ki jo dobimo s kakovostno funkcijo.

$$f() = \sum_{i=1}^{s} h_i() + k_i() \tag{2.1}$$

Oceno urnika določimo s kakovostno funkcijo 2.1, kjer je $h()$ ocena kromatičnosti i ter $k()$ število praznih prostorov v seji, zaradi katerih se lahko poveča kromatičnost znotraj ostalih sej. Funkcija deluje aditivno, kar pomeni, da vsak prazen prostor in višja kromatičnost poviša oceno. Ker ne želimo imeti praznih prostorov in visoke kromatičnosti, je najbolje ocenjen algoritem tisti, katerega razporeditev je ocenjena najnižje.

Kromatičnost opisuje število različnih tem v eni seji in je opisana s funkcijo $h(x_1) = x_1^3 * c_1$, kjer je c_1 utež, ki nam poveča intenzivnost kromatičnosti in x_1 število različnih tem v seji. Najmanjša in zatorej najboljša kromatičnost

je v seji z eno temo, najslabša pa je v seji s članki, kjer ima vsak svojo temo. Najboljšo kromatičnost na konferenci dosežemo, če imajo vsi članki v isti seji enako temo. Do take razporeditve lahko vedno pridemo, če le dovolj povečamo število sej in če članke, ki po temi ne ustrezajo v določeno sejo, razporedimo v dodatne seje. Take dodatne seje bodo najverjetneje imele prazne prostore, česar si ne želimo. Prazne prostore v seji ocenimo s funkcijo $k(x_2) = (x_2 * logx) * c_2$, kjer je c_2 utež in x_2 število praznih prostorov v seji.

Ker želimo med seboj primerjati konference različnih velikosti (torej z različnim številom člankov), je potrebno sam parameter velikosti konference pri ocenjevanju primerno utežiti. To naredimo tako, da vsako oceno delimo s številom člankov, ki smo jih razvrstili, po formuli $\frac{g}{c_3}$, kjer je g ocena in c_3 število vseh člankov. Algoritem, ki mu kakovostna funkcija določi najboljšo oceno, torej kar najmanjšo kromatičnost in najmanj praznih prostorov, je za dano konferenco najboljši.

S kakovostno funkcijo bomo ocenili vsak posamezen algoritem za izbrani primer konference. Rezultate ocenjevanja bomo primerjali in obrazložili.

2.3 Požrešni algoritem

Za izdelavo urnika konference bomo uporabili tri različne algoritme. Prvi med njimi je t. i. požrešni algoritem. Gre za algoritme, ki naredijo tak korak, da obeta najboljšo rešitev v določenem trenutku. Na problem ne gledajo v celoti in ponavadi rešitve zaradi tega niso optimalne, vendar pa so ti algoritmi navadno hitrejši. Nasprotje so počasnejši dinamični algoritmi, ki rešujejo problem v celoti in skoraj vedno pridejo do optimalne rešitve [9].

V osnovi imajo programi za implementacijo požrešnega algoritma 5 komponent:

1. Množica kandidatov, iz katerih se generira rešitev.
2. Funkcijo, ki izbira najboljšega kandidata za rešitev.
3. Izvedljivostno funkcijo, ki ocenjuje, če kandidat prispeva k rešitvi.

4. Ocenjevalno funkcijo, ki določi vrednosti rešitve.

5. Rešitveno funkcijo, ki sporoči, ko najdemo rešitev.

Pri požrešni izbiri algoritem izbira zgolj glede na trenutne razmere, to pomeni, da izbere korak, ki najbolje prispeva k rešitvi v trenutnem stanju sistema. Ob tem algoritem ne upošteva vpliva trenutne izbire na morebitne bodoče izbire. Algoritem problem razčleni na manjše komponente in vsako komponento rešuje posebej, ne da bi ob tem upošteval morebitne rešitve drugih komponent. Na ta način algoritem vedno pride do neke rešitve, ki je vsaj v določenih primerih dokaj blizu optimalne rešitve.

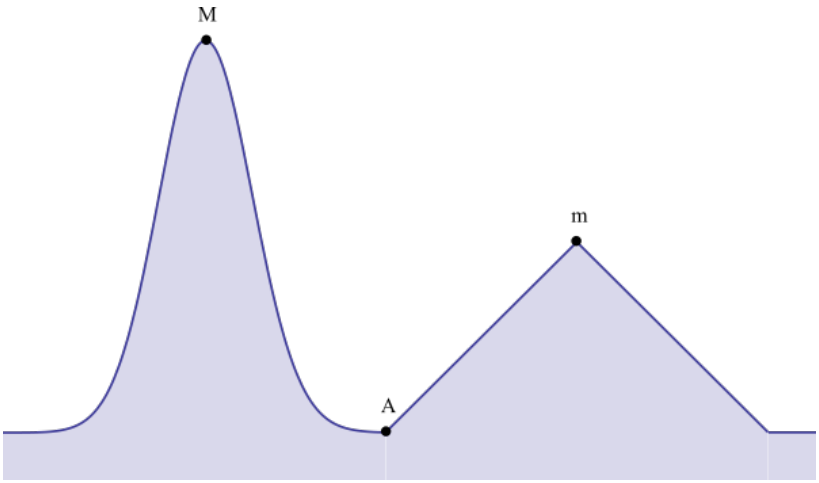
Običajno požrešni algoritmi dobro delujejo pri problemih, ki imajo optimalno podstrukturo. Problem ima optimalno podstrukturo, če je optimalna rešitev problema posledica optimalne rešitve podproblemov [9]. Torej so problemi, ki imajo optimalno podstrukturo, zelo primerni za reševanje s pomočjo požrešnih algoritmov.

Od požrešnih algoritmov pa se razlikuje dinamično programiranje, pri katerem se problem gleda kot celota in se pri novih odločitvah uporabi znanje, ki ga pridobi pri odločanju v prejšnjih korakih. Lahko gre celo za korak nazaj, da prejšnji korak spremeni na bolje. Dinamičnega programiranja nismo uporabili, vendar je to način, ki bi lahko dobro rešil problem, tako da bi po razvrstitvi člankov v seje, dodatno premikal članke med sejami, da bi prišel do želenega rezultata.

2.3.1 Neprimerni problemi za požrešni algoritem

Za nekatere probleme lahko požrešni algoritem najde najslabšo rešitev, sploh za tiste, ki nimajo optimalne podstrukture. Primer je problem potujočega trgovca [3], kjer želimo v primeru seznama mest in razdalj med dvojicama mest najti najkrajšo možno pot. To naredimo tako, da obiščemo vsako mesto natanko enkrat in se na koncu vrnemo v mesto, kjer smo začeli. Hevristika požrešnega algoritma pri takem primeru sledi strategiji, da na vsakem koraku obiščemo novo neobiskano mesto, ki je najbližje trenutnemu položaju.

Kako neprimeren je požrešni algoritem za reševanje takega problema lahko vidimo iz tega, da za določeno kombinacijo mest in razdalj med njimi požrešni algoritem dejansko najde najdaljšo možno pot.



Slika 2.1: Primer, kjer požrešni algoritem ne najde najboljše rešitve.

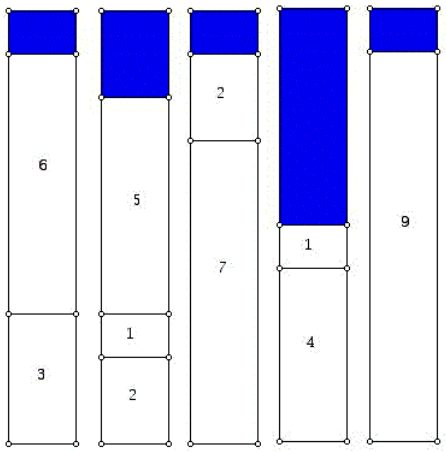
Obstajajo tudi drugi primeri, kjer požrešni algoritmi ne najdejo najboljših rešitev. Kot denimo v primeru na sliki 2.1, ko začnemo v točki A in želimo najti najvišjo točko na grafu. Požrešni algoritem deluje tako, da pogleda, v katero smer moramo iti, da se za več premaknemo v višino. Torej bi se algoritem usmeril proti točki m, ker je naklon od točke A proti točki m višji od naklona proti točki M. Pa vendar je točka M višja od točke m in bi bila optimalna rešitev usmeritev proti točki M.

2.4 Problem nahrbtnika in algoritmi za njegovo reševanje

Kot smo videli v poglavju 2.3, požrešni algoritem v določenih primerih ne daje optimalnih rešitev, zato smo iskali boljše algoritme, ki rešujejo podobne probleme, kot je naš. Bistvo problema nahrbtnika je razporejanje objektov različnih velikosti v koše fiksne in enake velikosti V, in sicer tako, da porabimo najmanjše možno število košev [6].

Problem nahrbtnika spada v skupino NP polnih problemov. Problem je NP poln, ko spada med NP probleme in hkrati med NP težke probleme. Z oznako NP označujemo vse tiste odločitvene probleme, katerih instance, kjer je odgovor >>da<<, imajo dokaz, da je odgovor res >>da<<. Ti dokazi morajo biti dokazljivi v polinomskem času z deterministčnim Turingovim strojem [1]. Z oznako NP težek označujemo vse tiste probleme, ki so vsaj toliko težki, kot najtežji problem v NP [10]. Problem A je NP težek, ko za vsak problem B v NP obstaja polinomska redukcija časa iz B v A [5].

Čeprav so koši podobni kot seje iz našega problema, razvrščanje člankov v seje ni popolnoma identično razvrščanju objektov v koše. Članki so vedno enakih velikosti, to pa ne velja za objekte, ki jih želimo razvrstiti v koše. Naslednja razlika pa je, da želimo pri generiranju urnika članke z isto temo razvrstiti v seje s to temo, ali prevedeno v jezik problema nahrbtnika, v točno določen koš. Pri problemu nahrbtnika elementi med seboj niso povezani in jih razvrstimo v katerikoli koš.



Slika 2.2: Primer problema nahrbtnika.

Čeprav se problema razlikujeta, lahko hevristiko iz algoritmov, ki se jih v osnovi uporablja za problem nahrbtnika, uporabimo tudi pri problemu za izdelave urnika.

Problem nahrbtnika se lahko predstavi na več načinov, kot so polnjenje zabojnikov, polnjenje tovornjakov z omejitvijo teže, generiranje varnostnih kopij na spominski medij.

Za reševanje problema nahrbtnika je na voljo nekaj preprostih hevrističnih algoritmov, ki jih predstavimo v nadaljevanju.

2.4.1 Naslednje prileganje

Naslednje prileganje je najpreprostejši hevristični algoritem za reševanje problema nahrbtnika. Algoritem vzame prvi koš $j = 1$ in prvi predmet $i = 1$. Če je v košu j prostor za predmet i, ga vanj uvrsti in gre na naslednji predmet $i + 1$. Če prostora v košu j ni, preveri naslednji koš $j + 1$, ob tem pa ne pogleda prejšnjih košev (torej košev $< j$). Časovna zahtevnost algoritma je majhna, in sicer $O(n * s)$, kjer je n število elementov in s število košev. Algoritem ne najde vedno optimalne rešitve in tako ostane veliko košev, kjer je še prostor. Algoritem ne gleda košev, ki jih pusti za seboj, in tako skoraj nikoli ne dobi optimalnega rezultata, kar pušča prostor za izboljšanje.

Izrek 2.1 *Naslednje prileganje v najslabšem primeru da rešitev, ki je za dvakrat večja od optimalne rešitve za problem nahrbtnika in ima časovno zahtevnost $O(n)$[8].*

```
Data: Nerazdeljeni elementi
Result: Razdeljeni elementi v koše
for element v seznamu do
    if ustreza v trenutni koš then
        vstavimo v koš;
    else
        pojdi na naslednji koš in vstavi;
    end
end
```

Algoritem 1: Algoritem naslednjega prileganja.

2.4.2 Prvo prileganje

Algoritem prvo prileganje spada v skupino požrešnih algoritmov in je v osnovi nadgradnja algoritma naslednje prileganje. Zelo hitro najde rešitev blizu optimalne. Časovna zahtevnost algoritma je $O(n * \log n)$, pri čemer je n število

vseh elementov, ki jih želimo razvrstiti. Glavna razlika med algoritmoma naslednjega prileganja in prvega prileganja je ta, da algoritem prvo prileganje pri razvrščanju elementov preveri tudi prejšnje koše, v katere smo že uvrščali elemente.

```
Data: Nerazdeljeni elementi
Result: Razdeljeni elementi v koše
for element v seznamu do
    for vsi koši, ki smo jih že uporabili do
        if ustreza v koš then
            vstavimo v koš;
        end
    end
    if elementa nismo uvrstili v obstoječe koše then
        naredimo nov koš in vanj vstavimo element;
    end
end
```

Algoritem 2: Algoritem prvega prileganja.

Enako kot algoritem naslednjega prileganja najslabši rezultat algoritma prvega prileganja razvrsti elemente v 2-krat več košev kot optimalen algoritem, vendar pa v povprečju daje boljše rezultate.

Izrek 2.2 *Prvo prileganje v najslabšem primeru da rešitev, ki je za dvakrat večja od optimalne rešitve za problem nahrbtnika in ima časovno zahtevnost* $O(n \log n)$.

Da algoritem naslednjega prileganja tudi pri najslabšem rezultatu ne more razvrstiti elementov v več kot 2-krat več košev kot optimalna rešitev, je razvidno iz naslednjega razmisleka. Predpostavimo, da algoritem elemente razporedi v več kot 2-krat več košev, kot bi jih pri optimalnem rezultatu. Če vzamemo katerakoli 2 zaporedna koša, morajo njim pripadajoči elementi skupaj porabiti vsaj V prostora, kjer je V velikost koša, saj bi bil drugače za

oba elementa dovolj samo en koš. Ker imamo le optimalno parov in enega več, imamo skupaj več kot optimalno V elementov, kar ni mogoče.

Iz same narave problema nahrbtnika lahko sklepamo, da na to, kako dobro bodo algoritmi razporedili elemente, močno vpliva začetna razvrstitev elementov. Začetna razporeditev elementov namreč vpliva na vrstni red razporejanja. Določeni avtorji celo dokazujejo, da vedno obstaja taka začetna razporeditev elementov, ki omogoča, da algoritem pride do optimalnega rezultata [6].

Algoritem prvega prileganja lahko denimo zelo izboljšamo, če razvrstimo elemente najprej po velikosti v padajočem zaporedju. Sicer algoritem tudi v tem primeru, ko najprej razporeja večje elemente, še vedno ne najde optimalne rešitve. V primerih večjega števila elementov pa nam lahko začetno razporejanje elementov privede do slabše rešitve.

Drugi med algoritmi za reševanje našega problema izdelave urnika bo tako algoritem, katerega hevristika bo zelo podobna algoritmu prvega prileganja. Uporabili bomo tudi optimizacijo razporeditve elementov po velikosti.

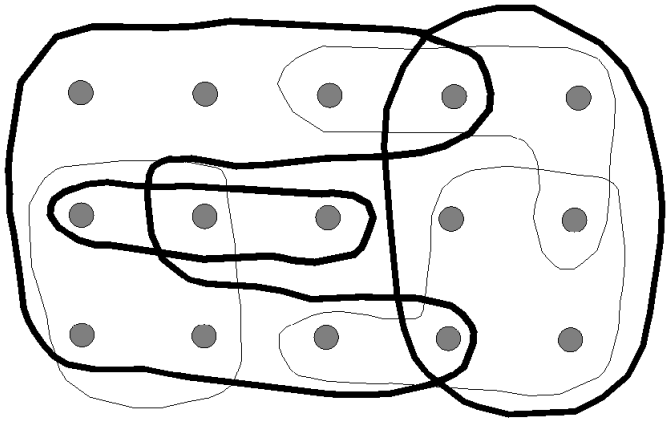
2.5 Problem pokritja in algoritmi za njegovo reševanje

Problem pokritja množic je eno klasičnih vprašanj v kombinatoriki, teoriji zahtevnosti in računalniški znanosti. Je eden od Karpovih 21 NP-polnih problemov. Dokaz, da spada v probleme NP polnosti je Karp podal leta 1972 [4].

Pri problemu pokritja imamo množico elementov in množico podmnožic, ki so zgrajene iz elementov. Podmnožice so vnaprej definirane. Bistvo rešitve problema je, da najdemo najmanjšo množico podmnožic, ki še pokriva vse elemente iz osnovne množice elementov. Podmnožice pa se morajo med seboj kar najmanj pokrivati.

Na sliki 2.3 imamo različne elemente označene kot pike, ki tvorijo osnovno množico elementov in različne podmnožice, ki jih pokrivajo. Tri podmnožice,

ki so označene z odebeljeno črto, pokrivajo vse elemente. Pri danih podmnožicah je to najboljša izbira.



Slika 2.3: Primer pokritja.

Bolj formalna definicija problema: Definiramo univerzalno množico U in družino S podmnožic iz U. Pokritje je poddružina $C \subseteq S$, katerih unija je univerzalna množica U. V problemu, ki je za nas zanimiv, imamo vhodni par (U, S) in želimo najti pokritje z množicami, ki porabi najmanj podmnožic iz družine S.

Obstaja požrešni algoritem za izračun pokritja, ki izbira množice po pravilu: na vsakem koraku izbere množico, ki vsebuje največje število nepokritih elementov. Tak algoritem lahko najde rešitev, ki se približa razmerju $H(n)$, kjer je n velikost množice, ki jo želimo pokriti. Z drugimi besedami: najde pokritje, ki je lahko za $H(n)$ večje kot minimalno, kjer je $H(n)$ n-to harmonično število ($H(n) = \sum_{k=1}^{n} 1/k \leq \ln n + 1$)[2]. Raziskave kažejo, da je ravno požrešni algoritem lahko najboljša polinomska aproksimacija za manjše probleme pokritja [7].

Problem pokritja je mogoče primerjati z razvrščanjem člankov v seje. Univerzalna množica so vsi članki, podmnožice pa so članki z istimi temami, iz katerih izbiramo. Problema se sicer preveč razlikujeta, da bi lahko uporabili enake algoritme, vendar lahko za naš problem uporabimo hevristiko, ki

je uporabljena za reševanje problema pokritja.

Poglavje 3

Implementacije

V prejšnjih poglavjih smo govorili le o teoretičnem ozadju algoritmov. Za analizo različnih algoritmov je bilo potrebno implementirati ogrodje, ki omogoča shranjevanje člankov in razporejanje v konferenci podobno podatkovno strukturo. Celotno ogrodje in algoritmi so implementirani v programskem jeziku Java. Uporabljen IDE je Netbeans, kjer smo algoritme testirali in ocenili rezultate.

3.1 Ogrodje

Za implementacijo je bilo najprej potrebno definirati podatkovne strukture, ki najbolj opredeljujejo konferenco.

Definirali smo tri različne razrede:

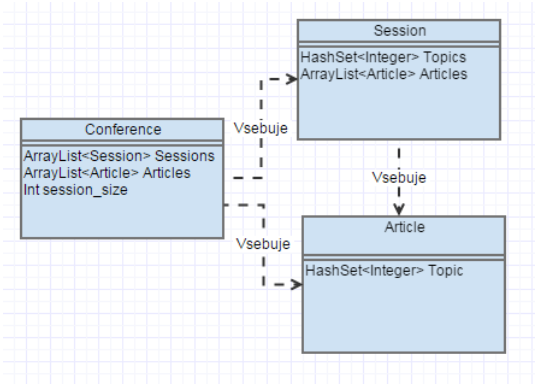
Article
: Je osnovni razred, ki definira članek. V tem razredu je definiran seznam za vse različne teme, ki so vezane na članek. Shranjene so kot številke, ker za potrebe te raziskave ne potrebujemo dejanskih imen tem. Razred ima dva različna konstruktorja. Oba dodajata teme v seznam. Prvi dodaja po eno temo, drugi pa sprejme tabelo ter doda vse elemente iz tabele v seznam.

Session

Je razred, ki definira sejo. V njem je definirana množica tem seje in seznam vseh člankov v seji. Število člankov v seji ni definirano v tem razredu, vendar se kasneje omeji v Conference razredu. Session ima dva konstruktorja. Prvi je brez parametrov in samo inicializira spremenljivke. Drugi sprejme povezani seznam člankov in jih shrani.

Conference

Je glavni razred, ki opredeljuje celotno konferenco. Definirana ima dva seznama, v katerih so shranjene vse seje in vsi članki, ki jih želimo razvrstiti. Poleg seznamov ima shranjeno še velikost seje (število člankov, ki jih seja lahko sprejme). Conference ima tri konstruktorje. Prvi sprejme razred Conference in se uporablja za kreiranje nove enake instance. Drugi in tretji sprejmeta seznam člankov. Razlika je, da v enem lahko z dodatnim parametrom spremenimo število člankov, ki jih seja lahko prejme, torej lahko spremenimo velikost seje. V temu razredu so definirane vse funkcije, ki razporedijo članke v seje glede na parametre, ki jim jih podamo.



Slika 3.1: Zgradba ogrodja za algoritme.

Poleg treh glavnih razredov, ki opisujejo konference, obstaja še razred, ki vse skupaj povezuje in zažene algoritme. Imenovali ga bomo Main. Definirane ima tri glavne funkcije, ki so razložene v podpoglavjih, in funkcijo, ki

prebere podatke o konferenci iz datoteke.

3.1.1 Algoritem za naključno generiranje konferenc

Kot smo že napovedali, smo za testiranje naših algoritmov poleg izmišljene in realne konference uporabili tudi psevdo naključno konferenco, ki smo jo dobili z uporabo Algoritma za naključno generiranje konferenci (gl. Algoritem 3). Algoritem sprejme tri parametre:

- število člankov n, ki jih želimo zgenerirati,
- maksimalno število tem m, ki jih želimo v člankih in
- število različnih tem k, katere se razdeli v članke.

Parametri so med seboj neodvisni. Omejeni so navzdol, saj ne bi bilo smiselno imeti manj kot en članek ali temo. Algoritem deluje s časovno zahtevnostjo $O(n * k)$. Rezultat algoritma je instanca objekta Conference, v katerem so v povezanem seznamu shranjeni vsi naključno zgenerirani članki.

Data: n, m, k
Result: Članki z naključno razporejenimi temami
for *i = 0; i < n; i++* **do**
  rand = psevdo naključno število med 1 in m;
  for *j = 0; j < rand; j++* **do**
    artikel = dodajamo v artikel po eno temo, ki je številka med 1 in k ;
  end
  dodamo artikel v skupno podatkovno strukturo;
end
return Conference(Artikli)

Algoritem 3: Algoritem za generiranje naključne konference.

3.1.2 Algoritem za ocenjevanje razporeditve

Za medsebojno primerjavo rezultatov, dobljenih z uporabo različnih testnih konferenc, smo uporabili Algoritem za ocenjevanje razporeditve (gl. Algoritem 4). Algoritem za svoje delo potrebuje 3 elemente:

- k: konferenco, ki jo želimo oceniti,
- c_1 utež za kromatičnost in
- c_2 utež za prazne prostore v seji.

Data: k, c_1, c_2
Result: Ocena konference
for *vse seje v konferenci* **do**
  kromatičnost $chrom = 0$;
  prazni prostori $prazni = 0$;
  for *vsak članek v seji* **do**
    pogledamo, če ustreza večinski temi seje;
    if *članek ne ustreza v sejo* **then**
      $chrom$++;
    end
  end
  if *število člankov v seji* $! = p$ **then**
    $prazni = p-$ število člankov v seji;
  end
  ocena kromatičnosti = $h(chrom)$;
  ocena praznih prostorov = $k(prazni)$;
  ocena = ocena kromatičnosti + ocena praznih prostorov;
end
return ocena;

Algoritem 4: Algoritem za ocenjevanje konference.

Konferenco podajamo v obliki instance Conference, v kateri so članki že urejeni v povezanem seznamu. Druga dva elementa pa sta uteži, s pomočjo katerih lahko pri ocenjevanju poudarimo posamezen vidik ocene. Če želimo, da je poudarek na enakosti tem v seji, povečamo c_1 in če želimo poudariti pomembnost polnih sej, povečamo c_2. Uteži se uporablja v kakovostni funkciji (gl. Formula 2.1), ki je opisana v poglavju 2.2.1. Za drugačne namene bi se funkcijo lahko spremenilo problemu primerno.

3.2 Osnovni algoritem za razporejanje člankov

Osnovni algoritem (gl.Algoritem 5) je bil napisan po požrešni metodi. V primeru, ko za generiranje urnika uporabimo algoritem iz skupine požrešnih algoritmov, je algoritem sorazmerno preprost, kar je tudi osnovna značilnost slednjih algoritmov. Kot vsak požrešni algoritem je tudi ta kratkoviden in ne vidi problema v celoti. Prvi članek se uvrsti v prazno sejo, ki se ji določi enake teme, kot jih ima članek. Preostale članke, ki so definirani v konferenci, poskuša najprej uvrstiti v prvo sejo, ki je na voljo. Preden uvrsti članek v sejo, preveri če le-ta ustreza seji. Članek seji ustreza, če je presek tem članka in seje neprazna množica. Ko se članek uvrsti v sejo, se seji določi novo temo, ki je presek tem članka in seje. Če članka ne moremo uvrstiti v prvo sejo, pogledamo naslednjo in tako dalje, dokler ne pridemo do konca vseh že določenih sej. Če članka nismo uvrstili, naredimo novo sejo, ki ima teme enake članku, katerega smo vanjo uvrstili. Kriterija za uvrstitev v sejo sta dva:

- seja mora imeti prazen prostor: prazen prostor ima, če je število člankov v seji manjše od maksimalnega števila člankov v seji, ki je določeno v Conference objektu,
- teme članka morajo ustrezati temam seje: kriterij je izpolnjen, če v preseku tem članka in seje obstaja vsaj ena tema.

Težave nastanejo, če izpolnimo maksimalno število sej in nimamo seje,

v katero bi se članek lahko uvrstil. V takem primeru članek uvrstimo v prvo sejo, ki še ima prostor in ne spremenimo teme seje. Veliko število takih člankov v eni seji pomeni veliko kromatičnost in s tem slabšo oceno razporeditve.

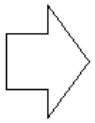
```
Data: Nerazdeljeni članki s temami
Result: Razdeljeni članki v seje
while še imamo članke v seznamu do
    if že obstaja seja then
        for seje s članki do
            if presek tem članka in seje != {} then
                if seja še ni polna then
                    vstavimo članek v sejo;
                    nov seznam tem seje je presek tem članka in seje;
                end
            end
        end
        if nobena seja ne ustreza && število sej < maksimalnega
         števila sej then
            naredimo novo sejo in vanjo vstavimo članek;
        else
            vstavimo v prvo sejo, ki ni polna;
        end
    else
        naredimo novo sejo in vanjo vstavimo članek s temo;
    end
end
```

Algoritem 5: Osnovni algoritem.

V primeru manjše konference z manj članki se osnovni algoritem izkaže za dokaj ustreznega. Rezultat je blizu optimalnega, kar se vidi na sliki 3.2. Uporabili smo konferenco z 20 članki, ki smo jih razvrstili v štiri različne teme. Na koncu nam algoritem razdeli članke v 8 sej, kjer so v vsaki seji le članki, ki imajo vsaj eno enako temo. Algoritem ima časovno zahtevnost

$O(n * s)$, kjer je n število vseh člankov in s je število sej, ki jih porabimo.

```
public static void main(String[] args) {
    ArrayList<Article> articles = new ArrayList<Article>();
    articles.add(new Article(new int[]{1,2,3,4}));
    articles.add(new Article(new int[]{1,2,4}));
    articles.add(new Article(new int[]{1,2,3}));
    articles.add(new Article(new int[]{1,3,4}));
    articles.add(new Article(new int[]{1}));
    articles.add(new Article(new int[]{2}));
    articles.add(new Article(new int[]{3,4}));
    articles.add(new Article(new int[]{1,2}));
    articles.add(new Article(new int[]{4}));
    articles.add(new Article(new int[]{3,4}));
    articles.add(new Article(new int[]{2,4}));
    articles.add(new Article(new int[]{1,2}));
    articles.add(new Article(new int[]{1,3}));
    articles.add(new Article(new int[]{2,3}));
    articles.add(new Article(new int[]{4}));
    articles.add(new Article(new int[]{3,4}));
    articles.add(new Article(new int[]{2,4}));
    articles.add(new Article(new int[]{1,2,3,4}));
    articles.add(new Article(new int[]{4}));
    articles.add(new Article(new int[]{2}));
```



```
Session articles: 3
[1, 2, 3, 4]
[1, 2, 4]
[1, 2, 3]
---------
Session articles: 3
[1, 3, 4]
[1]
[1, 2]
---------
Session articles: 3
[2]
[2, 4]
[1, 2]
---------
Session articles: 3
[3, 4]
[4]
[3, 4]
---------
Session articles: 3
[1, 3]
[2, 3]
[3, 4]
---------
Session articles: 3
[4]
[2, 4]
[1, 2, 3, 4]
---------
Session articles: 1
[4]
---------
Session articles: 1
[2]
```

Slika 3.2: Primer konference z 20 članki, ki jih požrešni algoritem razdeli v 6 sej po 3 članke in 2 seji po en članek.

3.3 Nadgradnje

Kot je bilo že povedano, spada naš osnovni algoritem v skupino požrešnih algoritmov. Naša želja je, da bi algoritem deloval bolj globalno ter tako našel boljšo rešitev. Ob primerjanju našega problema s problemom nahrbtnika in s problemom pokritja smo odkrili podobnosti in uporabili hevristiko iz teh problemov, da bi lahko osnovni algoritem izboljšali. Oba izboljšana algoritma imata še vedno časovno zahtevnost $O(n * s)$

3.3.1 Hevristika nahrbtnika

Problem nahrbtnika in njegove osnovne rešitve so bile opisane v poglavju 2.4. Pri problemu nahrbtnika vedno obstaja razporeditev elementov, za katero tudi požrešni algoritem najde optimalno rešitev[6]. Vendar pa te razporeditve, ki ima za posledico optimalno rešitev, ne moremo najti, če ne raziščemo vseh možnih razporeditev. Izkaže se, da je za izboljšanje rezultata potrebno uporabiti preprosto razvrstitev elementov po velikosti padajoče. Pri problemu nahrbtnika so to elementi, ki imajo največjo >>velikost<<. Tako bomo najprej razporedili največje elemente, ostale elemente pa dodajali majhnim prostorom, ki so ostali poleg velikih elementov. Ker je problem zelo podoben našemu, smo se odločili, da tudi v našem primeru pred uvrščanjem člankov v seje, članke najprej razvrstimo po >>velikosti<<. Ob tem je >>velikost<< število tem posameznega članka. Članki z večjim številom tem nam omogočijo več izbire za razporejanje ostalih člankov, zato je bistveno, da te najprej razporedimo. To je pravzaprav posledica dejstva, da ob uvrstitvi prvega članka v sejo, ta članek določi temo seje. Tako nam veliko število tem v članku poveča možnosti za sprejetje ostalih manjših (z manj temami) člankov v sejo.

Velikost pri problemu nahrbtnika torej lahko prevedemo v število tem, ki jih ima en članek. Tako se vse članke v konferenci najprej razporedi po številu tem padajoče. Rezultati v poglavju 4 pokažejo, da algoritem (gl. Algoritem 6), kateremu pred razporejanjem uredimo članke po velikosti, v našem primeru skoraj vedno najde dobro razporeditev in se tudi izkaže za najboljšega izmed testiranih algoritmov.

Problem nastane, če imamo konferenco, v kateri imajo vsi članki enako število tem. Pri taki konferenci ne moremo razporediti člankov po številu tem in tako nimamo napredka, zato je bila potrebna izjema. Ko zaženemo algoritem za urejanje po številu tem, moramo urediti tudi članke z enakim številom tem in jih razporediti tako, da so članki z enako temo skupaj. S tem preprečimo, da bi bila konference s članki z enakim številom tem neurejena pred razporejanjem v seje. Namesto imen tem smo za označbe tem uporabili

kar cela števila, saj je tako program za razporejanje hitrejši.

```
Data: Nerazdeljeni članki s temami
Result: Razdeljeni članki v seje
while !urejeno do
    for za vsak članek i in i + 1 do
        if število tem članka i > število tem članka i + 1 then
            zamenjamo pozicijo i in i + 1 ;
        end
    end
end
while !urejeno do
    for vse članke do
        if število člankov i == 1 && število člankov i + 1 == 1 then
            if številka teme v članku i > številka teme v članku i + 1
             then
                zamenjamo pozicijo i in i + 1 ;
            end
        end
    end
end
Uporabi Osnovni algoritem z urejenimi članki;
```

Algoritem 6: Algoritem za urejanje s hevristiko problema nahrbtnika.

3.3.2 Hevristika pokritja

Nadgraditev našega osnovnega algoritma s hevristiko problema nahrbtnika je zelo dobra, vendar nas zanima, če obstaja še kakšna, ki je mogoče boljša ali hitrejša, denimo z uporabo rešitev za problem pokritja. Celoten problem pokritja je opisan v poglavju 2.5. Kot je opisano na koncu poglavja, se naš problem in problem pokritja preveč razlikujeta in lahko uporabimo samo nekatere elemente pokritja. V osnovi želimo izbrati najmanjše podmnožice, ki pokrijejo vse elemente, vendar pa se med seboj ne prekrivajo. Začnemo

lahko z izbiranjem velikih množic, ki pokrijejo veliko elementov, vendar je v tem primeru verjetnost, da se množice prekrivajo večja. Lahko pa izberemo manjše množice, ki pokrijejo manj elementov, vendar je verjetnost, da se med seboj prekrivajo, manjša. Ker želimo najti najmanjše podmnožice, ki še pokrijejo vse elemente, začnemo z manjšimi podmnožicami.

Kot smo že povedali v poglavju o pokritju, lahko podmnožice prevedemo v članke s temami. Vendar nam za razliko od hevristike nahrbtnika, kjer je smiselna predhodna razporeditev člankov po velikosti padajoče, hevristika pokritja narekuje, da članke predhodno razporedimo po velikosti naraščajoče. Torej bo naš osnovni algoritem najprej začel razporejati članke z najmanjšim številom tem. Enak algoritem za urejanje, kot smo ga uporabili pri urejanju s hevristiko nahrbtnika, uporabimo za ureditev člankov po številu tem padajoče.

Data: Nerazdeljeni članki s temami
Result: Razdeljeni članki v seje
while *!urejeno* **do**
  for *za vsak članek i in i + 1* **do**
    if *število tem članka i < število tem članka i + 1* **then**
      zamenjamo pozicijo i in i + 1;
    end
  end
end
Osnovni algoritem z urejeni članki;

Algoritem 7: Algoritem za urejanje s hevristiko problema pokritja.

Algoritem s hevristiko pokritja (gl. Algoritem 7) deluje zelo podobno, kot algoritem 6. Razlikujeta se v načinu ureditve pred začetkom razporejanja. Pri algoritmu s hevristiko pokritja tudi nismo dodali izjeme za konference, kjer imajo vsi članki po eno temo, saj bi bil rezultat ocenjen podobno ali celo enako. Razlike bi se pojavile, če bi imeli na koncu premalo sej za razvrstitev vseh člankov in bi se tako članki razvrstili v seje, ki nimajo enake teme. V takem primeru bi ureditev povzročila spremembo v oceni, vendar bi bil

rezultat boljši ali slabši le po naključju.

Poglavje 4

Empirični rezultati

Testiranje naših algoritmov na različnih konferencah pokaže zanimive rezultate. Obstajajo konference in različne ureditve člankov, ki povzročijo, da je ocena razporeditve vseh algoritmov enaka. To pa ne pomeni, da so članki razvrščeni enako.

4.1 Testne konference

Uporabili bomo različne konference. Prva je manjša z $n = 20$ članki in $k = 11$ različnimi temami. Članki s temami so pred razporejanjem naslednji: [1, 2, 3, 4], [1, 2, 4], [1, 2, 3], [1, 3, 4], [1], [2], [3, 4], [1, 2], [4], [3, 4], [2, 4], [1, 2], [1, 3], [2, 3], [4], [3, 4], [2, 4] , [1, 2, 3, 4], [4], [2]. Vsak oklepaj predstavlja en članek in vsaka številka v članku predstavlja eno temo. Druga ima realne podatke, in sicer $n = 101$ člankov in $k = 11$ različnih tem. Uporabila se bo za preverjanje algoritmov na konferenci, ki se bo v realnosti zgodila. Da preverimo, kako se obnašajo algoritmi na velikih konferencah, bomo uporabili psevdo naključne konference s $n = 1000$ članki in $k = 15$ temami. V zadnjem podpoglavju bomo uporabili še naključno konferenco s parametri realne.

4.2 Testiranje

Za motivacijo pogledamo najprej preprost primer, da bomo kasneje lahko bolje razumeli zahtevnejše primere. Uporabili bomo osnovni algoritem (gl. Algoritem 5), ki ga testiramo na manjši konferenci. Konferenca je bila definirana v prejšnjem podpoglavju. Algoritmu za parametre nastavimo število sej $s = 8$ in velikost seje $p = 3$.

Članke nam razporedi tako:

- Seja 1: [1, 2, 3, 4], [1, 2, 4], [1, 2, 3],
- Seja 2: [1, 3, 4], [1], [1, 2],
- Seja 3: [2], [2, 4], [1, 2],
- Seja 4: [3, 4], [4], [3, 4],
- Seja 5: [1, 3], [2, 3], [3, 4],
- Seja 6: [4], [2, 4], [1, 2, 3, 4],
- Seja 7: [4],
- Seja 8: [2].

Zgoraj opazimo, da so se članki razporedili v 8 sej, od katerih je šest polnih, dve pa imata le po en članek. Ker smo podali preveliko število sej $s = 8$, je algoritem zadnja dva članka uvrstil vsakega v svojo sejo. Če bi želeli imeti zadnja dva članka v skupni seji, bi bilo potrebno podati število sej $s = 7$. Z zmanjšanjem števila sej s bi prisilili algoritem, da razvrsti zadnji članek k nepolni seji. Iz tega lahko sklepamo, da število sej s vpliva na razporejenost in posledično na oceno razporeditve. V podpoglavju 4.2.1 bomo testirali vpliv števila sej s na oceno.

Ker smo ugotovili, da število sej s direktno vpliva na razporeditev člankov v sejo, testiramo tudi vpliv velikosti seje p. Na istem algoritmu in konferenci spremenimo parametre in nastavimo velikost seje na $p = 5$, števila sej $s = 8$ pa ne spreminjamo:

- Seja 1: [1, 2, 3, 4], [1, 2, 4], [1, 2, 3], [1, 3, 4], [1],
- Seja 2: [2], [1, 2], [2, 4], [1, 2], [2, 3],
- Seja 3: [3, 4], [4], [3, 4], [4], [3, 4],
- Seja 4: [1, 3], [1, 2, 3, 4],
- Seja 5: [2, 4], [4],
- Seja 6: [2],
- Seja 7:,
- Seja 8:.

Opazimo lahko, da imamo le 6 uporabljenih sej, od katerih so 3 polne in 3 nepolne. Večja velikost seje p nam je zmanjšala minimalno število sej $s = \lceil \frac{n}{p} \rceil$ in povečala število nepolnih sej v razporeditvi. To nam pokaže, da tudi velikost seje p vpliva na različno razporeditev konference in posledično na oceno. V istem podpoglavju kot je navedeno zgoraj, bomo testirali tudi vpliv velikosti seje p na oceno.

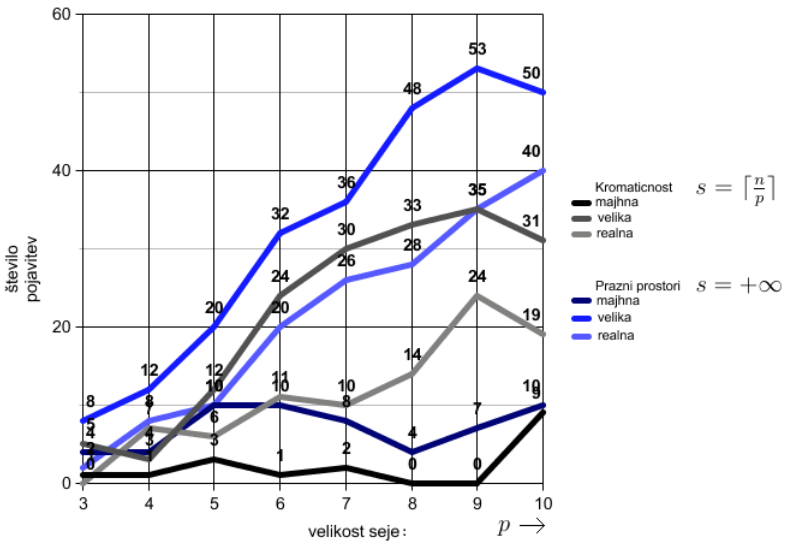
4.2.1 Vpliv parametrov na oceno kakovostne funkcije

V tem podpoglavju prikazujemo vpliv velikosti seje p in števila sej s na oceno razporeditve. Ocene, ki jih bomo prikazovali v grafih in jih primerjali, so dobljene s kakovostno funkcijo $f()$ (gl. Enačbo 2.1) opisano v poglavju 2.2.1.

Za najboljše ocene je najprej potrebno preučiti, kako se posamezni del kakovostne funkcije obnaša glede na parametre, ki jih podamo algoritmu, ne glede na utež, ki jo v kakovostni funkciji $f()$ lahko spreminjamo. Za ta namen bomo vsak del funkcije izpostavili in vsakega posebej testirali. Praznim prostorom $k()$ se bomo izognili z zmanjšanem velikosti seje p na minimalno $\lceil \frac{n}{p} \rceil$ za razporeditev vseh člankov. S tem bomo algoritem prisilili, da uporabi le toliko sej, kot jih je potrebno za razporeditev vseh člankov v

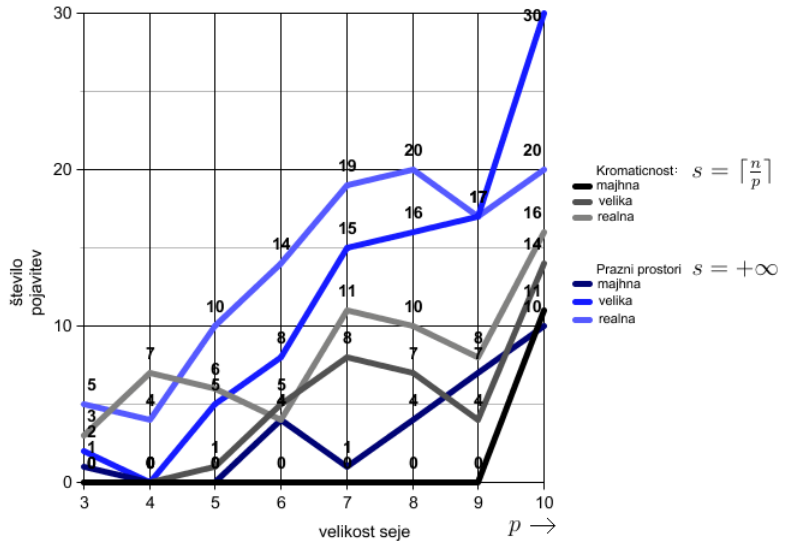
seje in zapolni vse prazne prostore. Če število člankov n ni deljivo z velikostjo seje p , nam bo v zadnji seji vedno ostalo nekaj praznih prostorov, ki jih bomo za namen raziskave kromatičnosti $h()$ ignorirali. Pri minimalnem številu sej $s = \lceil \frac{n}{p} \rceil$ je vpliv kromatičnosti $h()$ k oceni $f()$ največji. (gl. Slika 4.1 Kromatičnost). Da bi dobili največji vpliv praznih prostorov $k()$ je bilo potrebno odstraniti kromatičnost $h()$. S povečevanjem števila sej $s = +\infty$ smo povzročili, da se je članek, ki ni imel skupne teme s sejo, razvrstil v prazno sejo (gl. 4.1 Prazni prostori). Ker je vsak članek v seji le s članki enake teme, se kromatičnost izniči in na kakovostno funkcijo vplivajo le prazni prostori $f() = \sum_{i=1}^{s} k_i()$.

S spreminjanjem števila s in velikosti sej p lahko nadzorujemo vpliv posameznega dela kakovostne funkcije na oceno. Izziv je najti kombinacijo števila sej s in velikosti sej p, da je ocena najboljša.

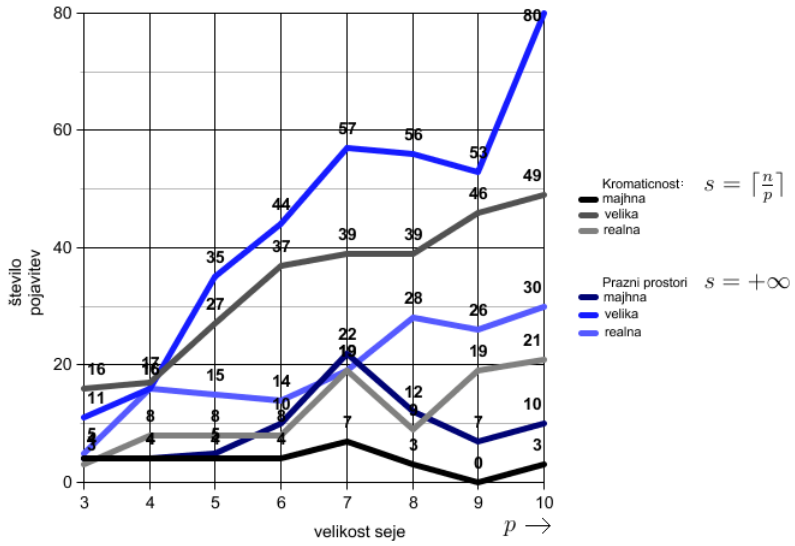


Slika 4.1: Vpliv posameznega dela kakovostne funkcije pri osnovnem algoritmu.

Na slikah 4.1, 4.2 in 4.3 je prikazano število člankov v sejah z različno temo (kromatičnost) in število praznih prostorov. Testirali smo vse algoritme na vseh konferencah. Število pojavitev na grafih nam pove, kolikokrat se prazen prostor pojavi (modra črta) in kolikokrat se pojavi članek, ki ne sodi v svojo



Slika 4.2: Vpliv posameznega dela kakovostne funkcije pri algoritmu s hevristiko nahrbtnika.



Slika 4.3: Vpliv posameznega dela kakovostne funkcije pri algoritmu s hevristiko pokritja.

sejo. Z drugimi besedami lahko za zadnji primer rečemo, da se mu poviša stopnja kromatičnosti (črna črta).

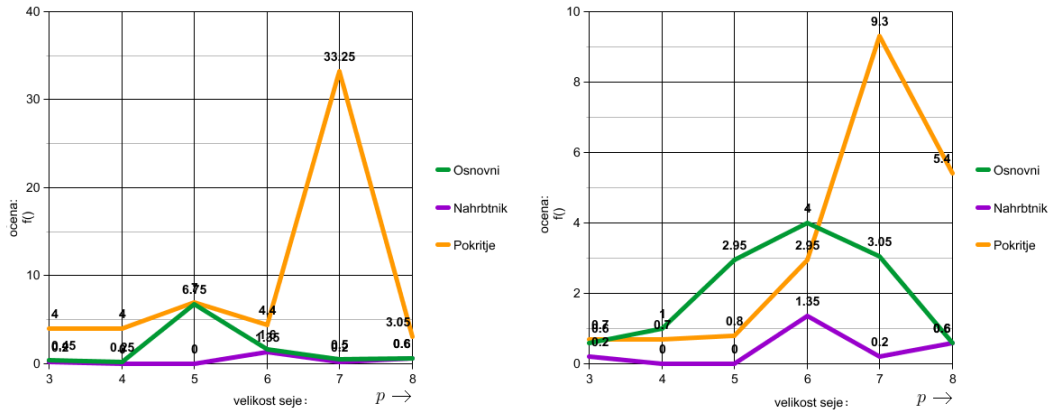
Opazimo, da je vpliv velikosti seje p na oceno $f()$ enak pri vseh velikostih konferenc n. Število pojavitev praznih sej $k()$ in kromatičnost $h()$ sta najvišja ob velikih sejah, iz česar lahko sklepamo, da velike seje povzročijo višjo kromatičnost in večjo pojavitev praznih sej ter posledično tudi visoko oceno. Obratno pa je najmanj praznih prostorov in najnižja kromatičnost pri majhnih sejah, kar pomeni, da so najbolje ocenjene razporeditve z manjšimi sejami. Ugotovitev je pričakovana, saj majhne seje povzročijo večje minimalno število sej $\lceil \frac{n}{p} \rceil$ in posledično večjo izbiro pri razporejanju člankov. S svojo velikostjo p pa omejujejo tudi večje nepravilnosti (prazne prostore in kromatičnost) v razporeditvi.

4.2.2 Primerjava algoritmov

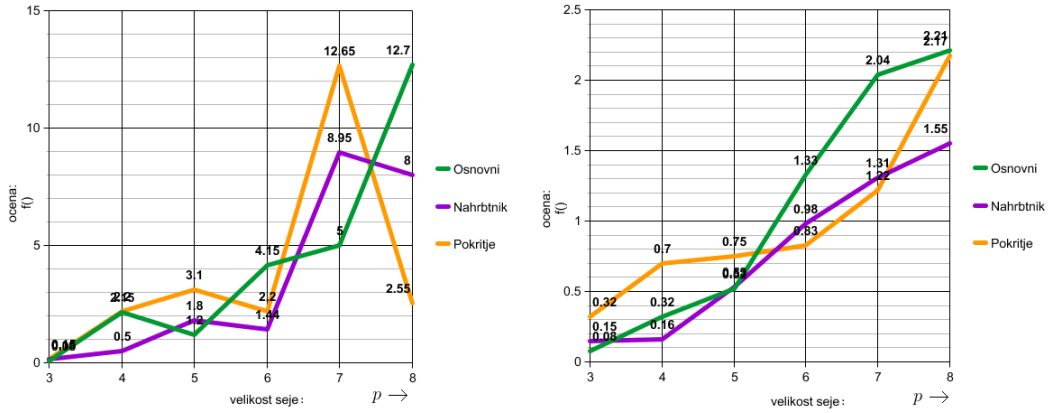
V prejšnjem podpoglavju smo primerjali vpliv parametrov na oceno $f()$. V tem podpoglavju pa bomo primerjali oceno rezultata za različne algoritme ob postavitvi enakih parametrov. Različne kombinacije parametrov za algoritem lahko zelo vplivajo na izbor najboljšega algoritma za konferenco. Najboljši algoritem bomo izbrali tako, da bomo algoritmom spreminjali parametre in jih testirali na vseh konferencah. Za algoritem, ki bo imel v naših primerih največkrat najboljšo oceno, lahko rečemo, da najverjetneje poda najboljšo razporeditev.

Na slikah 4.4, 4.5 in 4.6 so prikazane primerjave vseh algoritmov. Uporabljene konference so opisane v podpoglavju 4.1. Spreminjali smo velikost seje p, število sej pa smo nastavili na $s = \lceil \frac{n}{p} \rceil$ in $s = +\infty$. Grafi potrdijo, da parametri zelo vplivajo na oceno, saj se v določenih razmerah lahko za najboljšega izkaže tudi algoritem, ki v večini primerov poda najslabše ocenjeno razporeditev. To lahko vidimo na sliki 4.5, kjer ima algoritem s hevristiko pokritja najboljšo oceno $f()$ pri velikosti seje $p = 8$ in $s = \lceil \frac{n}{p} \rceil$. Če pogledamo ostale grafe, lahko opazimo, da se na večini primerov izkaže za najslabšega.

Na grafih izstopa algoritem s hevristiko nahrbtnika. Opazimo, da so njegove ocene v večini primerov najnižje. Še vedno pa obstajajo razmere, kjer se izkaže za slabšega, kot lahko vidimo na sliki 4.5, kjer pri velikosti



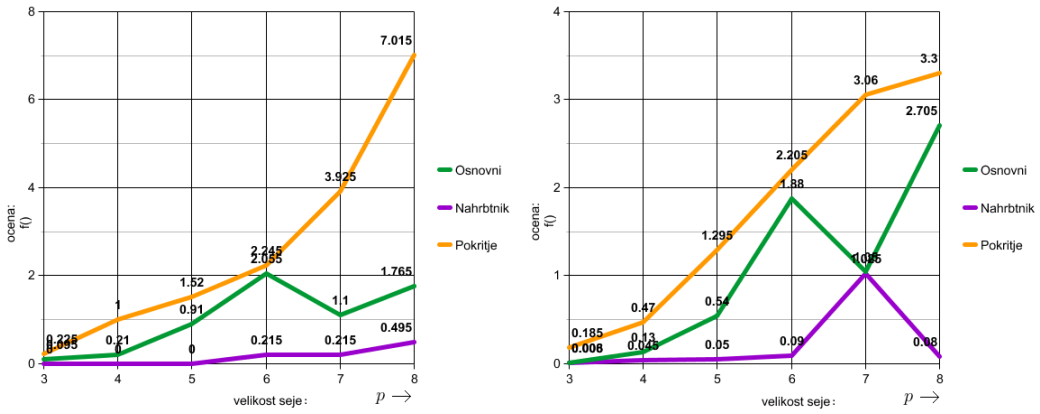
Slika 4.4: Vpliv velikosti seje na oceno vseh algoritmov manjše konference. Levo je število sej $s = \lceil \frac{n}{p} \rceil$. Desno je število sej $s = +\infty$.



Slika 4.5: Vpliv velikosti seje na oceno vseh algoritmov realne konference. Levo je število sej $s = \lceil \frac{n}{p} \rceil$. Desno je število sej $s = +\infty$.

seje $s = 7$ in $s = 8$ ni ocenjen najboljše. Ne moremo reči, da je algoritem s hevristiko nahrbtnika v splošnem najboljši algoritem, lahko pa trdimo, da se algoritem s hevristiko nahrbtnika na naših primerih največkrat izkaže za najboljšega.

Rezultat je dokaj pričakovan, saj algoritem s hevristiko nahrbtnika najprej razporedi članke z veliko temami, ki omogočijo veliko različnih možnosti za nadaljnje majhne članke. Druga dva algoritma se pri določenih nastavljenih parametrih približata, vendar pa pri večini nastavitev ne prideta blizu.



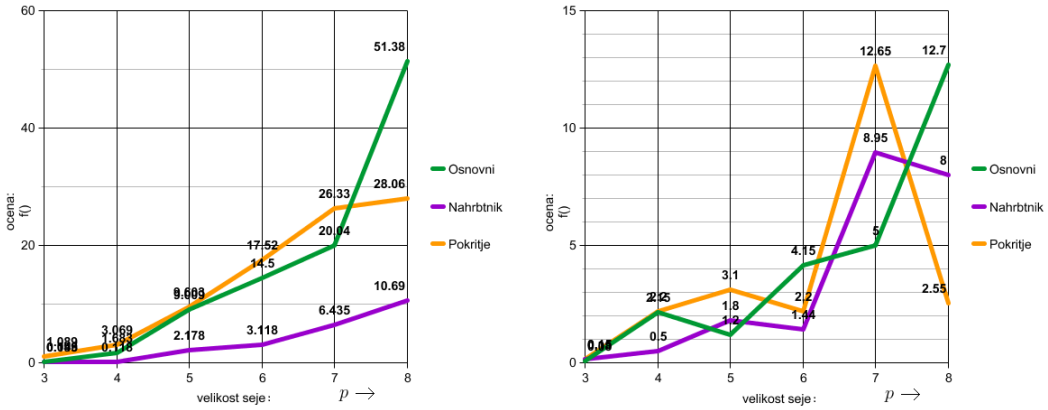
Slika 4.6: Vpliv velikosti seje na oceno vseh algoritmov večje psevdo naključne konference. Levo je število sej $s = \lceil \frac{n}{p} \rceil$. Desno je število sej $s = +\infty$.

Zaradi dodatne hevristike pokritja bi sklepali, da je osnovni algoritem slabši od algoritma s hevristiko pokritja, vendar urejanje člankov z manj temami povzroči, da imamo seje z manj temami in tako manj možnosti za razporejanje v teme.

4.2.3 Primerjava algoritmov na različno velikih konferencah

V prejšnjem podpoglavju smo lahko na sliki 4.6, kjer smo testirali na veliki konferenci, opazili, da so algoritmi ocenjeni bolj enotno in je konstantno najboljši le en. V našem primeru je to algoritem s hevristiko nahrbtnika. Za boljšo primerjavo algoritmov smo jih testirali še na več različnih velikih psevdo naključnih konferencah. Pred testiranjem velikih konferenc, za občutek primerjajmo še psevdo naključno zgenerirano konferenco velikosti realne, ki smo jo uporabljali v prejšnjih podpoglavjih. Taka konferenca bo imela $k = 11$ in $n = 101$ člankov, kateri bodo imeli 3 teme. Na sliki 4.7 vidimo, da se ocene algoritmov pri enaki velikosti sej zelo razlikujejo. Pri generirani konferenci (levo) opazimo, da so povprečne ocene zelo visoke v primerjavi z ocenami realne konference. Iz tega lahko sklepamo, da rezultati

testiranja na velikih konferencah, ki jih bomo dobili v nadaljevanju, ne bodo imitirali realnih konferenc enake velikosti. Še vedno pa je vredno preveriti, kako se algoritmi obnašajo na psevdo naključnih velikih konferencah.

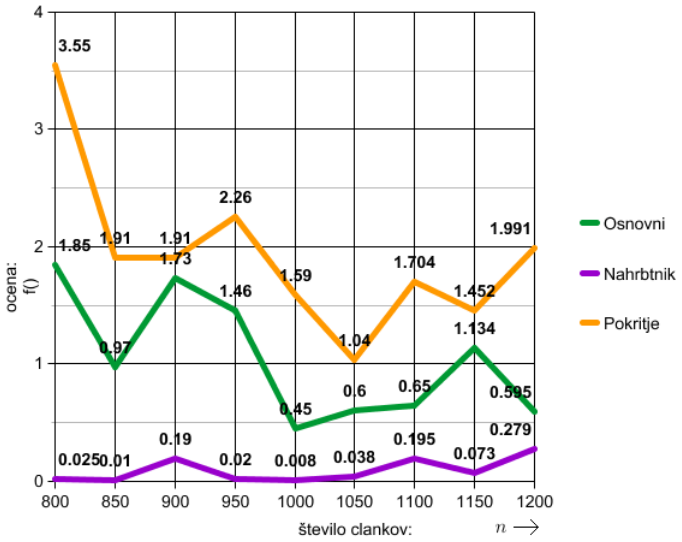


Slika 4.7: Primerjava ocen rezultatov vseh algoritmov psevdo naključno zgenerirane (levo) in realne konference (desno).

Na sliki 4.8 predstavimo 9 različnih psevdo naključnih velikih konferenc. Prva konferenca ima $n = 800$ člankov in vsaka naslednja $n = n+50; n < 1200$. Članki imajo največ 5 tem, ki so naključno izbrane $k = 15$. Generirali smo jih z Algoritmom za naključno generiranje konferenc (gl. Algoritem 3).

Za parametre algoritmov smo nastavili velikost seje $p = 6$ in $s = \lceil \frac{n}{p} \rceil$. Ker $n \nmid p$, bomo vedno imeli eno sejo, ki ne bo polna. Tako bo kakovostna funkcija ocenjevala kromatičnost in število praznih prostorov.

Ocene (gl. Slika 4.8 Pokritje) algoritma s hevristiko pokritja so ponovno najslabše. Najboljše ocene pa je enako kot pri manjših konferencah dobil algoritem s hevristiko nahrbtnika (gl. Slika 4.8 Nahrbtnik). Pri nekaterih konferencah se ocene osnovnega algoritma (gl. Slika 4.8 Osnovni) približajo, vendar so še vedno slabše kot pri algoritmu s hevristiko pokritja.



Slika 4.8: Primerjava ocen rezultatov vseh algoritmov različnih konferenc.

Poglavje 5

Zaključek

Na začetku dela smo predstavili problem razporejanja urnika na konferencah. V drugem poglavju smo opisali parametre p, s, n, k in kakovostno funkcijo $f()$, ki se uporablja za ocenjevanje razporeditev. Predstavili smo podobne probleme in jih primerjali z našim.

V nadaljevanju smo opisali funkcije, razrede in algoritme, ki smo jih kasneje uporabili za testiranje. Bolj podrobno smo razložili hevristiko požrešnega algoritma, hevristiko za reševanje problema nahrbtnika in hevristiko za reševanje problema pokritja. Vse hevristike smo uporabili v algoritmih. Vsi algoritmi delujejo s časovno zahtevnostjo $O(n * s)$.

Na koncu smo hevristike razporejanja člankov v seje testirali. Testiranje smo opravili na treh različnih konferencah. Prva se je uporabila pri testiranju algoritmov na manjših konferencah. Druga je bila uporabljena za primerjavo algoritmov na konferenci z realnimi podatki. Tretja je bila psevdo naključno zgenerirana konferenca, ki smo jo uporabili za primerjanje algoritmov pri razporejanju velikih konferenc. Ker smo za primerjavo potrebovali več velikih konferenc, se je psevdo naključno konferenco zgeneriralo večkrat. Vsako razporeditev smo ocenili s kakovostno funkcijo $f()$, ki je ocenjevala kromatičnost $h()$ in število praznih prostorov v sejah $k()$. Naši rezultati kažejo, da je glede na našo kakovostno funkcijo najbolje ocenjen algoritem s hevristiko nahrbtnika. Ko smo preučili še vlogo parametrov, smo prišli do pričakovanega

zaključka, da je najbolj pomemben parameter velikost seje p, ker najbolj vpliva na končno oceno razporeditve.

Za razporejanje predavanj na konferenci lahko uporabimo različne algoritme z različnimi hevristikami. V naši nalogi smo jih opisali le nekaj in jih med seboj primerjali. Potrebno bi bilo testirati več različnih algoritmov na več različnih realnih konferencah, ki bi nas morda pripeljali do boljših rezultatov. Čeprav rezultati naših testiranj kažejo, da najboljše ocene dobi algoritem s hevristiko nahrbtnika, so ob spremenjenih parametrih ali drugačni začetni razporeditvi člankov, lahko tudi drugi algoritmi bolje ocenjeni. Na vprašanje, kateri algoritem je najboljši za razporejanje predavanj na konferenci, nam ni uspelo odgovoriti, vendar smo prišli do algoritmov, ki bi s pravimi parametri za določeno konferenco to lahko bili.

Poglavje 6

Literatura

[1] M.H. Alsuwaiyel. Design techniques and analysis. *Algorithms*, 10, 1999.

[2] Václav Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[3] Gregory Gutin, Anders Yeo, and Alexey Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117(1):81–86, 2002.

[4] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[5] Donald Ervin Knuth. Postscript about NP-hard problems. *ACM SIGACT News*, 6(2):15–16, 1974.

[6] Rhyd Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers & Operations Research*, 36(7):2295–2310, 2009.

[7] Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 435–441. ACM, 1996.

[8] Alexander Souza. Combinatorial alghorithms. Lecture notes, Humboldt University Berlin, Winter Term 2011.

[9] H. Cormen Thomas, E. Leiserson Charles, L. Rivest Ronald, and Stein Clifford. *Introduction to Alghoritms*. MIT Press and McGraw-Hill, 2001.

[10] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A): algorithms and complexity*. Mit Press, 1991.