UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Kališnik

Preslikovanje enot filmskih podnapisov

Razvoj učinkovite metode z uporabo preiskovalnih algoritmov in semantično analizo

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2012

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01804/2012
Datum: 15.03.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR KALIŠNIK**

Naslov: **PRESLIKOVANJE ENOT FILMSKIH PODNAPISOV: RAZVOJ UČINKOVITE METODE Z UPORABO PREISKOVALNIH ALGORITMOV IN SEMANTIČNO ANALIZO**

MAPPING BETWEEN SUBTITLE UNITS IN MOVIE SUBTITLES: DESIGN OF AN EFFICIENT METHOD BY USING SEARCH ALGORITHMS AND SEMANTIC ANALYSIS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Naloga je poiskati in implementirati algoritme za učinkovito preslikovanje enot podnapisov med podnapisi. Preslikovanje enot podnapisov se naj osredotoča tako na strukturne lastnosti, kot so časi in trajanja prikaza ter število besed posamezne enote, kot tudi vsebinske lastnosti, kot je vsebina posameznih enot. Semantična analiza podnapisov naj deluje s podnapisi v različnih jezikih.

Mentor:
akad. prof. dr. Ivan Bratko

Dekan:
prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU
diplomskega dela

Spodaj podpisani *Gregor Kališnik*, z vpisno številko *63070041* sem avtor diplomskega dela z naslovom:

Preslikovanje enot filmskih podnapisov: Razvoj učinkovite metode z uporabo preiskovalnih algoritmov in semantično analizo

S svojim podpisom zagotavljam, da sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratkota, da so elektronska oblika diplomskega dela, naslov, povzetek ter ključne besede identični s tiskano obliko diplomskega dela in soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 17.9.2012 Podpis avtorja:

Zahvala

Zahvaljujem se družini in prijateljem, ki so mi pomagali pri popravljanju slovničnih napak. Zahvaljujem se tudi mentorju akad. prof. dr. Ivanu Bratku za vodenje pri mojem diplomskem delu.

Svojemu Abacusu.

Kazalo

Povzetek

Abstract

1 Uvod **1**
1.1 Problem 1
1.2 Struktura podnapisov 2
1.3 Opis preslikovanja 5
1.4 Uporaba 8
1.5 Pregledni prikaz lastnega pristopa 9
1.6 Povzetek rezultatov 10
1.7 Zgradba diplomske naloge 11

2 Pregled obstoječih pristopov **13**
2.1 Usklajevanje stavkov 13
2.2 Dynamic Time Warping 15

3 Opis postopkov **17**
3.1 Preslikovanje z bloki 17
3.2 Prostor stanj 23
3.3 Klasifikator 30
3.4 Semantično primerjanje 32
3.5 Metrike ocenjevanja 39
3.6 Zahtevnostni razredi 41

4 Izbira parametrov klasifikatorja **43**
4.1 Metoda podpornih vektorjev 43
4.2 Odločitveno drevo 45

5 Rezultati poskusov **47**
5.1 Meritve preslikovalnikov brez semantičnega primerjanja 48

5.2 Meritve preslikovalnikov s semantično podobnostjo kot priklic prevoda 50
5.3 Meritve preslikovalnikov s semantično podobnostjo kot razmerje končnih ločil 51
5.4 Najboljši in najučinkovitejši preslikovalnik 51

6 Sklep **55**
6.1 Ocena razvitih metod 55
6.2 Nadaljnje delo 56

Literatura **61**

Povzetek

V diplomski nalogi sem iskal rešitev problema preslikovanja enot dveh različnih podnapisov za isti film s pomočjo obstoječih algoritmov za iskanje najkrajših poti. Za zmanjševanje prostora preiskovanja sem uporabil klasifikatorje iz strojnega učenja in semantično analizo. Preslikave se lahko uporabi za izgradnjo poravnanih korpusov, prilagajanje podnapisov različnim izdajam filma in na koncu tudi razvrščanju podnapisov v skupine med seboj sorodnih podnapisov. Razvita metoda v veliki meri zadošča potrebam praktične uporabe. S primerno nastavitvijo parametrov metoda doseže izmerjeno F oceno 88,83%, časovno učinkovitejša različica, ki je za prakso primernejša, pa 72,64%.

Ključne besede:
preiskovalni algoritmi, klasifikatorji, SVM, odločitveno drevo, požrešno preiskovanje, A*, podnapisi za filme in epizode serij

Abstract

In this bachelor thesis I have developed a solution for the mapping problem between two different subtitles for a movie or episode of a series by using algorithms for finding shortest paths. I have narrowed the search space with classifiers and semantic analysis. Maps can be used for building aligned corpora, adjusting subtitles to different movie releases and finally for clustering subtitles into groups of replaceable subtitles. Developed method is in most cases usable in practice. With properly configured settings the method achieves F score of 88.83% while it's time efficient and more useful in practice version has F score of 72.64%.

Keywords:
search algorithms, classifiers, SVM, decision tree, greedy search, A*, subtitles for movies and episodes of series

Poglavje 1

Uvod

Podnapisi gledalcem predstavljajo pomemben pripomoček pri ogledu filma ali serije, saj si z njihovo pomočjo pomagajo pri lažjem razumevanju tujejezičnih filmskih vsebin. Še posebej to velja za podnapise prilagojene gluhim in naglušnim, katerim predstavljajo nepogrešljiv pripomoček pri splošnem razumevanju filma.

Filmske podnapise lahko izkoristimo tudi v računalništvu, saj lahko predstavljajo zelo pomemben vir pri izgradnji različnih sistemov, namenjenih samodejnemu prevajanju [22]. Pri vzdrževanju urejenosti baze podnapisov je potrebno imeti mehanizem, sposoben zaznati in ovrednotiti različna razmerja med samimi podnapisi filmov in epizod.

V diplomski nalogi sem se osredotočil na rešitev problema določanja temeljnih postopkov, s pomočjo katerih bi lahko ugotavljali razmerja med posameznimi filmskimi podnapisi[1], gradili poravnane korpuse, analizirali in primerjali jezike, razvili pripomočke za prevajalce, itd.

1.1 Problem

Zaradi različnih standardov post-produkcijske obdelave filma prihaja do časovnih razlik (npr. PAL in NTSC), pri čemer dobimo več oblik istega filma. Iz tega razloga je potrebno imeti podnapise prilagojene posamezni izdaji filma. Izdaje filmov se lahko razlikujejo tudi v vsebini. Obstaja lahko izdaja filma za predvajanje na televiziji, izdaja prilagojena državi, izdaja predvajana v kinih ter razširjene izdaje filma z dodatno vsebino.

Podnapisi za isto izdajo filma se lahko med seboj razlikujejo tudi vsebinsko. V podnapisu so lahko dodatne enote za gluhe in naglušne, dialogi iz

[1] V diplomski nalogi obravnavam tudi podnapise epizod, a zaradi lažjega izražanja omenjam le filmske podnapise.

Oblika zapisa podnapisa	Delež
SubRip	86,59%
MicroDVD	11,28%
Sub Station Alpha	0,82%

Tabela 1.1: Seznam treh najbolj zastopanih oblik podnapisov z deleži na strani Podnapisi.NET

ozadja, nekateri prevajalci prevedejo tudi besedila pesmi, itd.

Zaradi časovnih in vsebinskih razlik je težko vsebinsko povezati enote dveh podnapisov, saj brez te povezave se ne da graditi poravnanih korpusov in iskati sorodnostih povezav med podnapisi.

1.2 Struktura podnapisov

Podnapis je sestavljen iz enot podnapisa, vsaka enota vsebuje vsaj eno vrstico besedila, začetni in končni čas, časovni interval prikaza. Zaradi omejene širine zaslona je besedilo enote podnapisa lahko razdeljeno v več vrstic. Pri obdelavi podnapisov se osredotočam na vsebino, tako me ne zanima, kako so enote podnapisa prikazane in se zaradi tega osredotočam le na enote podnapisa.

Za shranjevanje podnapisov obstaja več oblik zapisov, od enostavnejših, v katerih se shranjuje le golo besedilo in časovne informacije, do zelo zapletenih, ki vsebujejo dodatne informacije, kot so koordinate v 2D in 3D prostoru, barve, slike itd. V tabeli 1.1 so prikazani deleži treh najbolj zastopanih oblik podnapisov, ki so na voljo na spletni strani Podnapisi.NET [6].

1.2.1 SubRip

Oblika podnapisa SubRip je bila narejena s strani avtorja istoimenske aplikacije SubRip. Aplikacija je namenjena pretvorbi podnapisov v slikah[2] v besedilno obliko (OCR), SubRip [5].

Oblika podnapisa SubRip je zelo preprosta, vsebuje le časovne podatke in golo besedilo brez naprednejših stilističnih opisov [4]. Sicer ima možnost določanja stilov besedila celotni enoti podnapisa, kot so barve, ležeči, krepki in podčrtani tisk, vendar se to le redko uporablja. Oblika SubRip je razširjena zaradi svoje enostavnosti, saj jo je možno urejati v preprostem urejevalniku besedil, kot je Beležnica.

V izpisu 1 je prikazan primer oblike podnapisa SubRip. Vsaka enota podnapisa ima podano zaporedno številko prikaza, pod njo sta podana čas

[2]Običajno so podnapisi na DVD medijih v slikovni obliki.

začetka ter konca prikaza. Čas je podan v urah, minutah in sekundah. Pri sekundah se uporablja zapis z decimalno vejico. Pod časom se nato nahaja samo besedilo enote podnapisa.

```
117
00:11:24,567 --> 00:11:28,196
Saj ga poznaš. Vsi govorijo
o tem. -Vesel bo.

118
00:11:29,047 --> 00:11:32,642
Povabil je pol Šajerske,
ostali pa bodo vseeno prišli.

119
00:11:36,767 --> 00:11:42,080
Življenje v Šajerski teče
naprej, kot v preteklem veku.

120
00:11:43,247 --> 00:11:47,604
Veliko je vzponov in padcev.
Spremembe pridejo počasi.

121
00:11:48,567 --> 00:11:50,364
Če sploh pridejo.
```

Izpis 1: Primer podnapisa za film „Gospodar prstanov: Bratovščina prstana“ v obliki zapisa SubRip

1.2.2 MicroDVD

Obliko podnapisa MicroDVD uporablja istoimenska aplikacija MicroDVD [2] za predvajanje DVD-jev s podnapisi, ki se ne nahajajo na DVD nosilcu. Vsaka DVD izdaja ima namreč priložene podnapise in takratni predvajalniki niso omogočali možnosti predvajanja z drugimi podnapisi. Avtorji so skupaj z aplikacijo MicroDVD ustvarili tudi svojo obliko podnapisa.

Glavna lastnost oblike MicroDVD je v opisu časa. Čas prikaza enote podnapisa je podan v *sličicah* in ne v sekundah, kar dodatno oteži primerjanje podnapisov MicroDVD z ostalimi oblikami, saj je potrebno narediti pre-

tvorbo v sekunde. Ni točno znano, zakaj so se avtorji odločili za določanje prikaza enote podnapisa zaporedno številko sličice namesto časa podanega v sekundah. Možen razlog bi lahko bil zaradi enostavnejšega in hitrejšega preverjanja trenutno prikazane sličice kot pa izračun predvajanega filma v sekundah.

Pri pretvorbi je potrebno poznati pravo hitrost prikazovanja sličic (FPS), katero pa ni vedno enostavno ugotoviti. Oblika MicroDVD je ena bolj enostavnih oblik in prav tako kot oblika SubRip omogoča določanje osnovnih stilov enotam podnapisa.

MicroDVD oblika ima na začetku vsake enote podnapisa znotraj zavitih oklepajev zapisano začetno ter končno sličico prikaza na ekran. Za časovnimi podatki nato pride golo besedilo. Primer podnapisa v obliki MicroDVD prikazuje izpis 2.

```
1 {17110}{17167}Saj poznaš Bilba.|Celoten kraj je v nemiru.
2 {17172}{17218}No, to bi ga moralo zadovoljiti.
3 {17222}{17268}Pol dežele je povabljene.
4 {17273}{17337}Ostali pa bodo ravno tako prišli.
5 {17412}{17491}In tako gre življenje v|deželi Shire
     naprej...
6 {17496}{17568}...zelo podobno kot v|prejšnji dobi...
7 {17575}{17707}...s polno vzponi in padci|in zelo
     počasnimi spremembami.
8 {17711}{17757}Če sploh so.
```

Izpis 2: Primer podnapisa za film „Gospodar prstanov: Bratovščina prstana“ v obliki zapisa MicroDVD

1.2.3 Sub Station Alpha

Oblika podnapisa Sub Station Alpha, krajše SSA, je ena od bolj zapletenih oblik zapisa. Specifikacija oblike SSA zajema kar 20 strani [3]. Podobno kot druge oblike zapisov je bila tudi SSA narejena skupaj z istoimensko aplikacijo za urejanje podnapisov.

SSA omogoča zelo široko paleto nastavitev, efektov, stilov itd. V grobem je podnapis ločen v sekcije [3]. Primer podnapisa v obliki SSA je prikazan v izpisu 3. Iz primera je razvidno, da je ta oblika zelo zapletena, zaradi česar je oteženo ročno urejanje podnapisov te oblike.

Glava

Glava je namenjena osnovnim informacijam, začenši z naslovom podnapisa, naslovom izvorne skripte in različico aplikacije s katero je bil podnapis shranjen. V glavi se nahaja tudi sekcija, namenjena definicijam oblik enot podnapisa (npr. font, velikost besedila, barva, ipd.).

Vsebina

Vsebina podnapisa se nahaja v sekciji „Events“ in sicer se vsaka enota podnapisa začne z besedo „Dialogue“, sledijo osnovne informacije o prikazu enote podnapisa. Začetek in konec prikaza, stil, poravnava, itd. Definicija enote podnapisa se konča s samim besedilom.

```
1  [Script Info]
2  Title: <untitled>
3  Original Script: <unknown>
4  ScriptType: v4.00
5
6  [V4 Styles]
7  Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, TertiaryColour,
       BackColour, Bold, Italic, BorderStyle, Outline, Shadow, Alignment, MarginL,
       MarginR, MarginV, AlphaLevel, Encoding
8  Style: Default,Tahoma,24,16777215,16777215,16777215,12632256,-1,0,1,1,1,6,30, 30,415,0,0
9
10 [Events]
11 Format: Marked, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text
12 Dialogue: Marked=0,0:00:20.84,0:00:25.19,Default,NTP,0000,0000,0000,!Effect,Pred
       davnimi časi\Nv daljni galaksiji ...
13 Dialogue: Marked=0,0:00:27.76,0:00:32.51,Default,NTP,0000,0000,0000,!Effect,VOJNA ZVEZD
14 Dialogue: Marked=0,0:00:39.32,0:00:45.15,Default,NTP,0000,0000,0000,!Effect,4.
       epizoda\NNOVO UPANJE
```

Izpis 3: Primer podnapisa za film Vojna zvezd 4. epizoda v obliki zapisa Sub Station Alpha

1.3 Opis preslikovanja

Preslikavo sem definiral kot množico povezanih enot podnapisa para podnapisov (leva in desna stran preslikave oz. levi in desni podnapis). Da je povezava enote levega podnapisa z enoto desnega podnapisa smiselna, se morata vsebini enot ujemati. Usklajenost podnapisov s filmom je uporabna lastnost, zaradi česar imajo enote podnapisov za isti film podobne (ali v določenem razmerju) čase prikazov, kljub različnemu jeziku. Torej, enote podnapisa z isto vsebino (v pomenskem smislu) se pojavijo ob istem času ter imajo približno enak čas trajanja. Seveda moramo pri iskanju povezav paziti pri uporabi teh časovnih lastnosti, saj se, kot sem že opisal, podnapisi lahko med seboj razlikujejo v časih, lahko so zamaknjeni ali prilagojeni za izdajo filma z drugačno hitrostjo menjavanja sličic.

Pri povezovanju enot podnapisa moramo paziti tudi na nekaj drugih, vsebinskih, značilnosti. Zaradi lastnosti jezika oziroma potrebe po zgoščevanju vsebine za lažje razumevanje so lahko enote podnapisa razdvojene ali združene. Vsebinske razlike se lahko odražajo tudi kot delež neprevedene vsebine[3]. Zaradi naštetih značilnosti je potrebno določiti sledeča pravila, ki jih mora preslikovalnik pri iskanju povezav med enotami dveh podnapisov predvideti:

1. povezava enote podnapisa z enoto drugega podnapisa (1:1),
2. povezava enote podnapisa z več enotami drugega podnapisa (1:m),
3. povezava več enot podnapisa z eno enoto drugega podnapisa (m:1),
4. povezava več enot podnapisa z več enotami drugega podnapisa (m:n),
5. enota ali več enot podnapisa je lahko tudi brez povezave (1:0 ali 0:1),
6. vrstni red povezanih enot obeh podnapisov mora biti enak.

Prvo pravilo je preprosta. Za enoto na levi strani preslikave obstaja enota na desni strani, kar je značilno za izpeljave podnapisa, kot so popravki in priredbe[4]. Naslednja tri pravila potrebujejo malo več razmisleka. Prevajalec lahko pri prevajanju dodatno razdrobi ali celo združi posamezne enote podnapisa. Zaradi teh (ra)združitev enot je potrebno omogočati povezovanje ne le ene enote na levi strani z eno enoto na desni strani preslikave, ampak tudi večjega števila enot podnapisa, za kar sem dodal še drugo, tretjo in četrto pravilo. Primerne povezave glede na našteta pravila so prikazane na sliki 1.1.

Peto pravilo je dodana zaradi vsebinskih sprememb. Te spremembe se nahajajo pri drugačnih oblikah podnapisov[5] in filmov[6]. Brez petega pravila ne bi bilo mogoče preslikovanje med različnimi oblikami podnapisov, recimo med navadnim podnapisom in podnapisom za gluhe in naglušne.

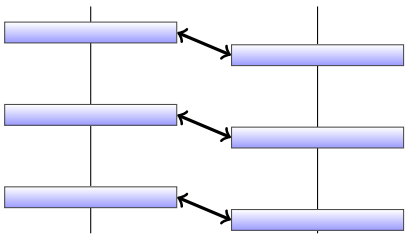
Zadnje pravilo doda še potrebo po ohranjanju vrstnega reda na obeh straneh preslikave. Vzemimo neko povezavo med dvema enotama podnapisa. Šesto pravilo predvideva, da se naslednja enota podnapisa, recimo na levi strani preslikave, ne sme povezati s prejšnjo enoto podnapisa na desni strani preslikave. Slika 1.1f prikazuje eno od povezav, ki krši šesto pravilo.

[3] Neprevedeni dialogi v ozadju, ...

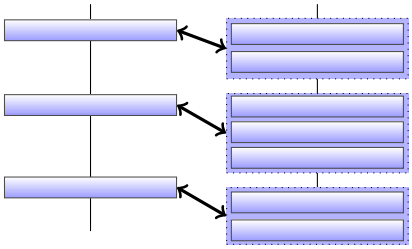
[4] Pri podnapisih se za priredbo podnapisa smatra kot prilagoditev na drugo obliko filma

[5] Podnapisi za gluhe in naglušne, podnapisi z vsebovanimi prevodi pesmic, neprevedene vrstice, itd.

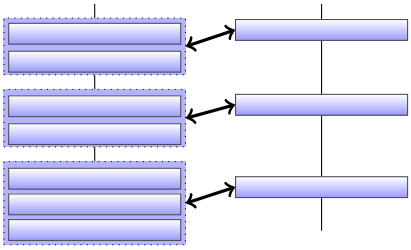
[6] Televizijska izdaja, razširjena izdaja, itd.



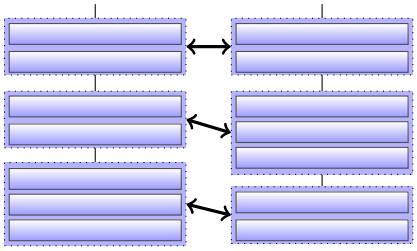
(a) Povezave po 1. pravilu



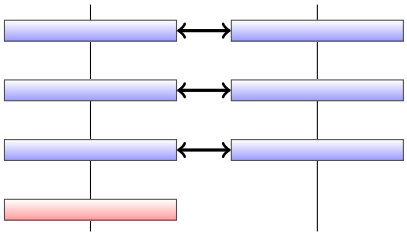
(b) Povezave po 2. pravilu



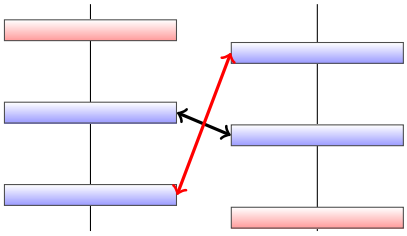
(c) Povezave po 3. pravilu



(d) Povezave po 4. pravilu

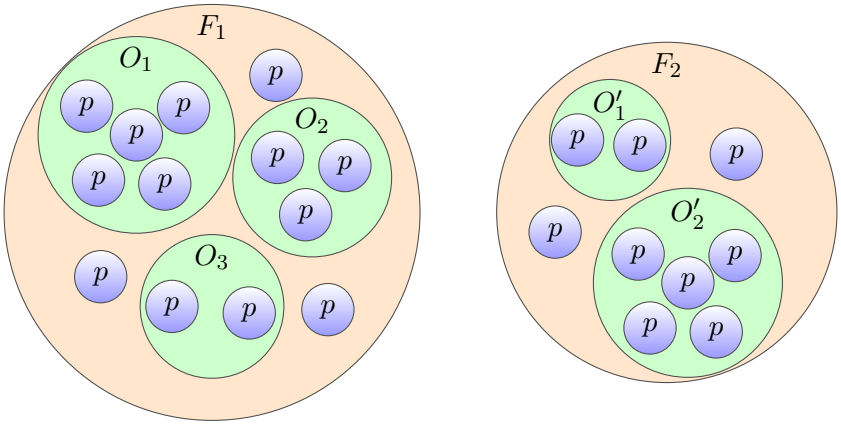


(e) Nepovezana enota podnapisa po 5. pravilu



(f) Nedovoljena povezava zaradi 6. pravila

Slika 1.1: Primeri povezav, ki jih dopuščajo vse pravila



Slika 1.2: Prikaz skupin podnapisov za isto obliko filmov z množicami, kjer sta F_1 in F_2 množici podnapisov za isti film, O_1, O_2, ... množice podnapisov za isto obliko filma in na koncu še posamezni podnapisi p

1.4 Uporaba

1.4.1 Razvrščanje podnapisov v skupine sorodnih podnapisov

Sistem za preslikovanje sem razvil v sodelovanju z ekipo Podnapisi.NET. Sistem bi uporabili za razvrščanje podnapisov v skupine med seboj zamenljivih podnapisov oziroma množice sorodnih podnapisov. S tako razvrščeno bazo podnapisov bo olajšano iskanje primernih podnapisov uporabnikom kot tudi zagotovil potrebam uredniških funkcij in raznih jezikovnih analiz.

Hkrati bo omogočen tudi razvoj programske opreme za strojno identifikacijo filmskega medija, pa naj bo medij le datoteka na trdem disku ali DVD izvod filma. Za strojno identifikacijo filma bi potrebovali vsaj časovne lastnosti podnapisov, torej sistem bi uporabil ali podnapise ali časovne intervale govora, dobljene s pomočjo obstoječih „voice activity detection“ [18] sistemov.

Za lažjo predstavo razvrščanja podnapisov v skupine in njen pomen, si oglejmo sliko 1.2. Na tej sliki je prikazan primer skupin za isto obliko (O_1, O_2, ..., O'_2) filmov F_1 in F_2. Kot je prikazano na sliki, obstaja možnost osamelcev, ki lahko ustrezajo istemu filmu, lahko pa so tudi napačno identificirani podnapisi[7].

[7] Z identifikacijo podnapisa je mišljeno določanje pripadnost podnapisa določenemu filmu.

1.4.2 Prilagajanje podnapisov

S pomočjo znane preslikave med dvema podnapisoma je podnapis mogoče prilagoditi poljubni obliki filma, namreč, preslikane enote podnapisa lahko poravnamo na prave časovne intervale, odvečne enote pa preprosto odstranimo. Problem nastane pri manjkajočih enotah podnapisa, katere je potrebno nadomestiti z novimi.

Prilagajanje podnapisov bi bilo uporabno za navadnega uporabnika, saj bi lahko uporabil obstoječi podnapis za katerokoli izdajo pri želeni izdaji filma, kot primer, uporabnik ima DVD izdajo s slovenskimi podnapisi, medtem ko BlueRay izdaja nima slovenskega prevoda. S prilagajanjem podnapisa z DVD izdaje BlueRay izdaji filma bi lahko užival v gledanju filma s slovenskim prevodom na BlueRay izdaji.

Prilagajanje podnapisa bi bilo uporabno tudi za prevajalce. Ob novi, vsebinsko drugačni, izdaji filma bi lahko prevajalec prilagodil prej prevedeni podnapis na novo izdajo filma, ter si s tem olajšal delo in si skrajšal čas potreben za prilagajanje obstoječega prevoda novi različici izdaje. Potrebni čas za prilagajanje podnapisa bi se za prevajalca zmanjšal, ker ne bi imel potrebe po ročnem poravnavanju časovnih intervalov posameznih enot podnapisa. Hkrati bi prevajalec dobil tudi že vnaprej določene časovne intervale manjkajočih enot podnapisa, katere bi „zapolnil“ z novimi prevodi.

1.4.3 Gradnja poravnanih korpusov

Podnapisi so časovno usklajeni s filmom, zato so med seboj usklajeni tudi podnapisi v različnih jezikih. Na ta način se lahko s pomočjo dveh podnapisov v različnih jezikih gradi poravnane korpuse. Zaradi različnih oblik filmov nastane problem pri gradnji poravnanih korpusov, saj se ne razlikujejo le po časovnih lastnostih, temveč tudi po vsebini (npr. razširjene izdaje filmov, podnapisi za slušno prizadete, različno goščenje vsebine[8]), zato je potrebno poiskati povezave med enotami podnapisa z zelo podobno vsebino in iz teh povezav zgraditi poravnane korpuse.

1.5 Pregledni prikaz lastnega pristopa

Do sedaj sem opisal preslikovanje posameznih enot podnapisa. Moj pristop preslikuje skupine enot podnapisa, imenovane kot bloki. Posamezna povezava vsebuje dve skupini blokov, ena je z levega podnapisa, druga z desnega

[8] Vsebino podnapisa se lahko med drugim zgosti z izpuščanjem prevajalcu nepomembnih vsebin.

podnapisa.

Problem preslikovanja sem predstavil kot preiskovalni problem, ki ga rešujem z uporabo požrešnega preiskovanja in preiskovalnega algoritma A*. Preiskovalni prostor je zelo velik, saj je za posamezen blok potrebno preveriti veliko možnih povezav. Prostor stanj omejujem z uporabo klasifikatorja. V diplomski nalogi sem uporabil odločitveno drevo in metodo podpornih vektorjev. Za cenovno funkcijo sem določil povprečje med posameznimi ocenami podobnosti oziroma primernosti povezave:

- razmerje trajanja obeh strani povezave,
- relativno število možnih nadaljnjih povezav,
- razmerje števil blokov obeh skupin povezave,
- relativna zakasnitev povezave,
- relativna dolžina vseh preskočenih enot podnapisa,
- semantična podobnost.

Za zadnjo oceno podobnosti (semantična podobnost) sem izdelal dva semantična primerjalnika. Eden je zasnovan na metodi primerjanja stavkov. Zaradi lažje izvedbe sem implementiral primerjalnik, ki primerja število pojavitev znakov klicaj in vprašaj. Drugi semantični primerjalnik primerjan vsebine povezav kot priklic (recall) prevoda. Prevod se izvede tako, da za vsako besedo naredi množico možnih prevodov. Kot uspešen prevod se smatra takrat, ko katerokoli besedo iz množice možnih prevodov najde na drugi strani povezave.

Za prevajanje sem potreboval dvo-jezični slovar. Zaradi posebnosti jezika v podnapisih, jezik v filmih je zelo tematsko obarvan, sem se odločil za lastno izvedbo enostavnega algoritma za gradnjo slovarjev. Z ustvarjenih slovarjem sem nato izvajal poskuse.

1.6 Povzetek rezultatov

Meritve poskusov so pokazale boljše rezultate preslikovalnikov s preiskovalnim algoritmom A*, a so zato potrebovale več časa. Za najboljši preslikovalnik z F oceno 88,83% in s povprečnim časom iskanja 989,69s se je izkazal preslikovalnik z odločitvenim drevesom, preiskovalnim algoritmom A* in s primerjanjem ločil kot semantični primerjalnik.

Z upoštevanjem še porabljenega časa za iskanje posamezne preslikave se je za najboljši preslikovalnik izkazal preslikovalnik z odločitvenim drevesom,

požrešnim preiskovanjem in brez semantičnega primerjalnika. Slednji preslikovalnik je dosegel F oceno 72,64% s povprečnim časom iskanja 57,50s.

Razviti preslikovalnik je za primer uporabe pri gradnji poravnanih korpusov ali iskanju sorodnih podnapisov primeren. Podnapisov je veliko in z določanjem ostrih kriterijev pri preslikavi še zmeraj lahko pridemo do zadovoljive količine podatkov za izgradnjo poravnanih korpusov. Za iskanje sorodnih podnapisov se pa primerja le podnapise za isto izdajo filmov, kjer učinkovitejši preslikovalnik deluje z 85,98% F oceno[9]. Za prilagajanje podnapisov za drugo izdajo filma pa je razviti preslikovalnik neprimeren, saj bi za tako uporabo potreboval točnejše rezultate.

1.7 Zgradba diplomske naloge

V drugem poglavju opišem obstoječe pristope za usklajevanje dveh podnapisov. Nadaljujem s podrobnim opisom lastnega pristopa usklajevanja podnapisov z iskanjem preslikave. Četrto poglavje opisuje postopek in izbrane parametre za klasifikatorja odločitveno drevo in metodo podpornih vektorjev. Rezultate poskusov predstavim v petem poglavju in diplomsko nalogo zaključim z zadnjim, sklepnim poglavjem v katerem predstavim tudi nekaj idej za nadaljnji razvoj pristopa.

[9]Podnapisi za isto izdajo filma in brez večjih vsebinskih razlik (dodatnih enot zaradi gluhih in naglušnih, ipd.) praviloma spadajo v enostavnejši zahtevnostni razred, ki je opisan kasneje v diplomski nalogi

Poglavje 2

Pregled obstoječih pristopov

Glavna motivacija za nastanek obstoječih pristopov za usklajevanje dveh filmskih podnapisov je gradnja poravnanih korpusov [16, 15, 20, 22, 13, 21], zato vsi trenutni pristopi vsebujejo postopke usklajevanja stavkov, ki se uporabljajo pri njihovi izgradnji.

Prvi od teh pristopov vsebuje velik del ročnega usklajevanja enot podnapisov [16]. V članku sta avtorja opisala nekaj napotkov, dobljenih s poskusi pri samem usklajevanju, kot je zamik za 500 ms ter poravnavo enot s pomočjo samostalnikov.

V kasnejših člankih so avtorji predlagali samodejne ali polovično samodejne postopke. Ti postopki se delijo na usklajevanje s pomočjo opornih točk (anchor points) [20, 21], DTW (Dynamic Time Warping) [15, 22] in z nadgradnjo algoritma za usklajevanje stavkov dvojezičnih korpusov s pomočjo dinamičnega programiranja po principu Gale and Church 1993 [8, 13].

Obstoječe pristope sem združil v dve idejni skupini. Pristopi prve skupine delujejo po ideji usklajevanja stavkov, medtem ko pristopa druge skupine uporabljata algoritem DTW za maksimizacijo podobnosti ali minimizacijo razdalje iskane preslikave med dvema podnapisoma.

2.1 Usklajevanje stavkov

Problem usklajevanja para podnapisov lahko razumemo tudi kot problem usklajevanja neporavnanega dvojezičnega korpusa [20, 13]. Tako lahko uporabimo algoritme za usklajevanje stavkov.

Algoritmi usklajevanja stavkov uporabljajo raznorazne lastnosti, kot so primerjanje dolžin v znakih, uporaba preprostih dvojezičnih slovarjev itd. Za uporabo teh algoritmov v usklajevanju para podnapisov je potrebno dodati še časovno komponento podnapisov.

Če sta podnapisa med seboj časovno usklajena, časovna komponenta podnapisa zelo pripomore k natančnosti usklajevanja, vendar pa je zaradi velikega števila izdaj filmov precej podnapisov med seboj časovno neusklajenih. Ta problem se rešuje s pomočjo opornih točk [20, 21] ali z dopolnitvijo obstoječega algoritma za usklajevanje stavkov [13].

2.1.1 Oporne točke

Za časovno uskladitev para podnapisov si lahko pomagamo z opornimi točkami (Anchor points). Glavna ideja te metode je v iskanju vsaj dveh opornih točk, s katerimi se izračuna tako zakasnitev t_o kot koeficient razlike[1] c z enačbi [20]:

$$\begin{aligned} c &= \frac{t'_1 - t'_2}{t_1 - t_2} \\ t_o &= t'_2 - t_2 \cdot c \end{aligned}$$

Kjer sta t_1 in t_2 časa opornih točk v izvornem (source)[2] oziroma levem podnapisu ter t'_1 in t'_2 časa opornih točk v ponornem oziroma desnem podnapisu. S tem dvema parametroma je mogoče časovno usklajevati par podnapisov.

Pri usklajevanju z opornimi točkami nastane pomembno vprašanje izbora le-teh. Lahko se jih izbere popolnoma ali delno ročno [20]. Avtor tega pristopa je definiral hevristično oceno, s katero se oceni dobljeno uskladitev stavkov in z njo omeji ročno določanje opornih točk le na tiste pare podnapisov, ki potrebujejo časovno usklajevanje. Ocena o je definirana kot:

$$o = \frac{|N| + 1}{|P| + 1}$$

kjer je N množica nepraznih povezav in P množica praznih povezav. Prazne povezave so tiste povezave, ki imajo stavek le na eni strani. Neprazne povezave pa povezujejo stavek ene strani z vsaj enim stavkom druge strani.

Uporabnik metode sam presodi, nad katero vrednostjo bo ročno določal oporne točke. Avtor metode je za prag izbral vrednost dve[3]. Glavna ideja za to hevristično oceno je v predpostavki, da bi stavčno usklajevanje para podnapisov s podobno vsebino moralo ustvariti veliko število nepraznih povezav.

[1] Pri dveh podnapisih z različno hitrostjo menjavanja sličic pride do razlik za neko konstanto.

[2] V literaturi so podnapisi navedeni kot izvorni ali ponorni, v moji diplomski nalogi uporabljam terminologijo leva in desna stran preslikave.

[3] Dva krat več praznih povezav kot nepraznih povezav.

Obstaja tudi možnost uporabe dvojezičnega slovarja za samodejni izbor opornih točk [21]. Pri tej metodi je avtor z uporabo dvojezičnega slovarja določil večje število množic opornih točk in nato s prej definirano hevristično oceno izbral najboljšo množico opornih točk.

2.1.2 Dopolnitev algoritma za usklajevanje stavkov

Časovnemu usklajevanju podnapisov se lahko izognemo z dopolnitvijo algoritma za usklajevanje stavkov [13]. Avtor tega članka je k obstoječi meri podobnosti dodal še časovno komponento podnapisa.

V izvornem algoritmu usklajevanja stavkov je bila za mero podobnosti uporabljena relativna dolžina stavka v znakih [8]. Pri dopolnjenem algoritmu se je k meri podobnosti dodalo še trajanje enote podnapisa. Celotna mera podobnosti ima torej dve komponenti, časovno in vsebinsko.

2.2 Dynamic Time Warping

Preslikovanja enot podnapisa se lahko lotimo tudi kot optimizacijski problem. Obstajata dva pristopa, z maksimizacijo podobnosti [15] in z minimizacijo razdalje [22]. Oba pristopa uporabljata semantično analizo za izračun mere podobnosti oziroma razdalje.

2.2.1 Maksimizacija podobnosti

Prva metoda preslikovanja enot podnapisa, ki je uporabljala algoritem DTW, je bila maksimizacija mere podobnosti. Kot mero podobnosti so vzeli F oceno prevoda. F oceno so izračunali tako, da so prevedli posamezno besedo, prešteli so koliko prevedenih besed se nahaja na drugi strani preslikave ter izračunali priklic (recall) in preciznost (precision).

2.2.2 Minimizacija razdalje

V drugem članku so problem obrnili in raje uporabili metodo minimizacije mere razdalje. Pri izračunu razdalje RFDM[4] so uporabili slovar $D_{L_2L_1}$ iz jezika L_2 v jezik L_1, ki je za dano besedo w zgeneriral množico prevodov. Tako so za enoto podnapisa $S_j^{L_2}$ z besedami $\left\{ w_{j,1}^{L_2}, w_{j,2}^{L_2}, \ldots, w_{j,M_j^{L_2}}^{L_2} \right\}$ pridobili

[4] „Relative-frequency based distance metric“

vrečo besed (bag of words):

$$B_j = \left\{ D_{L_2L_1}(w_{j,1}^{L_2}), D_{L_2L_1}(w_{j,2}^{L_2}), \ldots, D_{L_2L_1}(w_{j,M_j^{L_2}}^{L_2}) \right\}$$

Definirajmo še enoto $S_i^{L_1}$ druge strani preslikave z besedami

$$W_i = \left\{ w_{i,1}^{L_1}, w_{i,2}^{L_1}, \ldots, w_{j,M_i^{L_1}}^{L_1} \right\}$$

Tako imamo v drugem podnapisu z N_2 enotami množico prevodov $\{B_1, B_2, \ldots, B_{N_2}\}$, v katerem je N_w edinstvenih besed. Vsaka beseda v množici prevodov w_k se pojavi C_{w_k}-krat v celotni zbirki prevodov ($k = 1, \ldots, N_w$). Mera RFDM [22] je izračunana po sledeči enačbi:

$$RFDM(S_i^{L_1}, S_j^{L_2}) = \left(\sum_{k=1}^{M_i^{L_1}} \frac{I_{w_{i,k}^{L_1}}}{C_{w_{i,k}^{L_1}}} \right)^{-1}$$

kjer je

$$I_{w_{i,k}^{L_1}} = \begin{cases} 1; w_{i,k}^{L_1} \in W_i \cap B_j \\ 0; \text{sicer} \end{cases}$$

Spremenljivka $I_{w_{i,k}^{L_1}}$ je indikator, če je beseda w vsebovana tako v množici besed enote prvega podnapisa W_i, kot v množici prevodov B_j enote drugega podnapisa. Zaradi uporabe inverza spremenljivke $C_{w_{i,k}^{L_1}}$ je tako vpliv zelo pogostih besed, kot so na primer vezniki, zmanjšan.

Poglavje 3

Opis postopkov

V prejšnjih poglavjih sem opisal glavne uporabnosti za preslikovalnik in glavne težave, ki jih mora premagati. V tem poglavju bom opisal glavne komponente in zamisli implementacije takega preslikovalnika.

Semantična analiza se je izkazala za zelo zapleteno in časovno potratno področje, zato sem se odločil, da bom opisal več metod vsebinskega primerjanja in jih tudi preizkusil.

3.1 Preslikovanje z bloki

Povezovanje enot podnapisa ima nekaj težav. Enota podnapisa je zelo majhna enota, ki sama po sebi ne nosi veliko uporabnih informacij za razločevanje posameznih enot in njihovih povezav z drugo stranjo preslikave. Težava pri povezovanju enot je tudi veliko število enot v posameznem podnapisu, predstavlja velik računski zalogaj, še posebej za mojo implementacijo. Zato sem raje določil bloke kot skupino enot podnapisa, ki so v časovni soseščini. Za enote podnapisa v isti časovni soseščini velja, da so premori med njimi manjši od premorov z enotami zunaj te časovne soseščine. Preslikovanje z enega v drug podnapis sem določil kot povezovanje blokov, natančneje, skupine blokov. Na sliki 3.1 so prikazane razlike in podobnosti med obema pristopoma.

Problem povezovanja blokov je enak kot pri povezovanju enot podnapisa, a bloki so večji in iz njih se da izpeljati več informacij za uporabo pri klasifikaciji pravilnih oziroma nepravilnih povezav. Njihovo število v podnapisih je tudi občutno manjše, namesto okoli 1.000 enot podnapisa imamo le okoli 80 blokov na podnapis. Zaradi enakih lastnosti blokovnih povezav in povezav enot podnapisa se pravila za blokovno preslikovanje prevede v sledeča pravila:

B1 blok enot v blok enot,

B2 blok enot v več blokov enot,

B3 več blokov enot v en blok enot,

B4 več blokov enot v več blokov enot,

B5 blok ali več blokov enot je lahko tudi brez povezave,

B6 velja časovna linearnost.

Zaradi časovne linearnosti lahko v skupine blokov dajemo le zaporedja blokov brez preskakovanja. Časovna linearnost nas tudi omejuje pri možnosti povezovanja skupin blokov ene strani v drugo stran, križne povezave pa so zaradi časovne linearnosti prepovedane.

Slika 1.1 sicer prikazuje primerne povezave na nivoju enot podnapisa, vendar enako velja tudi za bloke. Edina razlika med enotinim in blokovnim preslikovanjem je v razumevanju problema. Pri preslikovanju enot podnapisa se ukvarjamo s posamezno enoto, medtem ko se pri blokovnem preslikovanju ukvarjamo s skupino enot podnapisa hkrati. Če blok razumemo kot dialog, lahko blokovnemu preslikovalniku rečemo tudi preslikovalnik dialogov.

3.1.1 Bloki in njihova gradnja

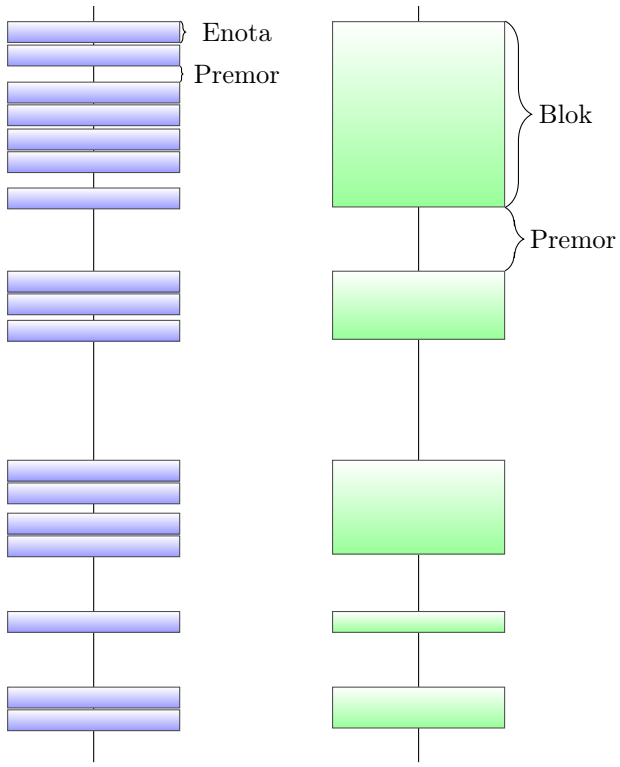
Enote podnapisa se dodaja v blok, dokler je premor med zadnjo enoto v bloku in naslednjo enoto manjši od mejne vrednosti *gap_limit*. Vrednost te spremenljivke izračunamo po enačbi:

$$gap\_limit = \overline{g} + \sigma(g)$$

Kjer je g seznam premorov med enotami podnapisa.

Zaradi uporabe standardnega odklona imamo manjše težave z razlikami v trajanju med filmi in epizodami. Posamezne epizode serije so zaradi narave televizije krajše od filmov in kot take so premori med dialogi manjši. Z manjšimi premori med dialogi se pa tudi zmanjša standardni odklon premorov med enotami podnapisa. V filmih so premori med dialogi daljši, zato je tudi standardni odklon premorov večji. S standardnim odklonom pridobimo nekakšno mero, s katero si lahko pomagamo pri gradnji dovolj velikega števila blokov.

S povprečjem preprečim preveliko razdrobljenost blokov pri epizodah z enakomernimi premori. S prevelikim številom blokov se približamo primeru preslikave enot, medtem ko s premajhnim številom blokov izgubljamo informacije. Celoten algoritem za gradnjo blokov je podan v izpisu 4.



Slika 3.1: Različni predstavitvi podnapisa

```
def create_blocks(subtitle):
  gaps = [subtitle.lines[i].start - subtitle.lines[i - 1].end
          for i in xrange(1, len(subtitle.lines))]
  gap_limit = mean(gaps) + std(gaps)

  blocks = []
  block = Block()
  block.add_line(subtitle.lines[0])
  blocks.append(block)
  for line in subtitle.lines[1:]:
    if line.start - block.end > gap_limit:
      block = Block()
      blocks.append(block)
    block.add_line(line)
  return blocks
```

Izpis 4: Psevdokoda gradnje blokov

3.1.2 Blokovno primerjanje

Blokovno primerjanje sestoji predvsem iz zakasnitev povezav. Zakasnitev povezav je definirana z razliko med začetnimi časi blokov ($B(b)$) obeh strani povezave:

$$d(b, b') = B(b) - B(b')$$

Iz vseh zakasnitev se nato lahko izračuna povprečje, standardni odklon itd. Za povprečje in standardni odklon sem uporabljal bodisi absolutno vrednost bodisi kvadrat zakasnitve in se s tem znebil negativnega predznaka. Našteti podatki izražajo razmerja med podnapisoma.

Ob predpostavki, da je standardni odklon kvadratov zakasnitev[1] enak 0 in s povprečnim zamikom 0s, lahko trdimo, da sta podnapisa sorodna, torej zamenljiva. Ob visokem standardnem odklonu kvadratov zakasnitev pa postaja primerjava vedno manj smiselna.

Razredi in podrazredi preslikav

Preslikave podnapisov se v glavnem delijo v dva različna razreda:

Stopničasto zamaknjen Glavni pomen stopničasto zamaknjenih primerjav je, da zakasnitve nimajo nobene posebne zakonitosti. Poimenoval sem jih stopničasti zaradi možnosti enakih zakasnitev na določenem

[1] Kvadrat zakasnitve je potreben zaradi predznaka, lahko bi se uporabilo absolutne zakasnitve.

intervalu povezav. Skrajna primera stopničasto zamaknjenih primerjav sta primerjava z enakimi zakasnitvami in primerjava z različnimi zakasnitvami za vsako povezavo.

Različna hitrost menjave sličic Izdaje filmov se zaradi tehničnih lastnosti kodiranja lahko razlikujejo tudi v hitrosti menjave sličic. Ta razlika se odrazi v zakasnitvah, določenih z neko konstanto c. Konstanta je določena z razliko med hitrostjo menjavanja sličic (FPS) obeh oblik. Torej, zamiki povezave d_i so izraženi z začetkom leve (z_i) in desne (z'_i) skupine blokov s sledečo enačbo:

$$d_i = z_i \cdot c - z'_i$$

Stopničasto zamaknjene preslikave lahko delimo še na dva pomembna podrazreda:

Identičen Identična preslikava nima nobenih zakasnitev. Ponavadi pride do identičnih preslikav takrat, ko se primerja izvorni podnapis z njegovo priredbo ali s prevodom. S pomočjo tega podrazreda primerjav bi lahko do neke mere izdelali način določanja „sorodstvenih“ vezi med podnapisi.

Zamaknjen Podrazred zamaknjenih preslikav je zelo podoben podrazredu identičnih primerjav, saj imata oba podrazreda pri vseh povezavah enake zakasnitve. Razlika je v tem, da zamaknjene preslikave nimajo ničelnih zakasnitev.

Na sliki 3.2 so prikazani primeri za oba razreda in dodatni delitvi razreda stopničasto zamaknjenih preslikav. Graf na sliki 3.3 ponazarja razlike v zakasnitvah med njimi. Primeri so zaradi lažje upodobitve izmišljeni.

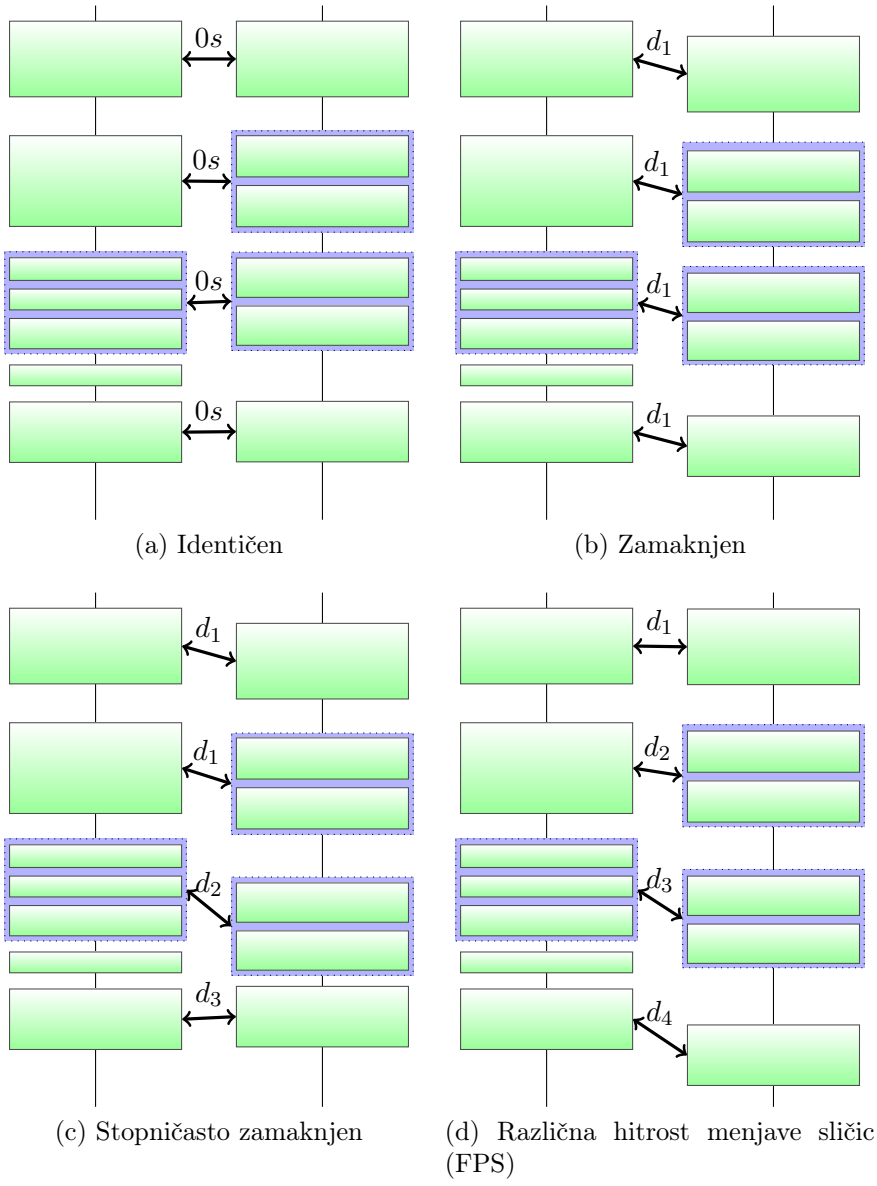
Pokritost

Pri preslikavah je pomembna tudi metrika pokritosti. Pokritost sem določil kot odstotek preslikanih trajanj in je definirana s sledečo enačbo nad preslikavo m:

$$C(m) = \frac{\sum_{(b,b')\in L} D(b) + D(b')}{\sum_{b\in B_l} D(b) + \sum_{b'\in B_r} D(b')}$$

V enačbi nastopajo množica povezav L, množici levih (B_l) ter desnih (B_r) blokov in funkcija trajanja bloka v sekundah $D(b)$. Kot je vidno iz enačbe, pokritost predstavlja razmerje vsote trajanj povezanih blokov in vsote trajanj vseh[2] blokov leve in desne strani preslikave. Glavni namen metrike je ugotavljanje verodostojnosti preslikave.

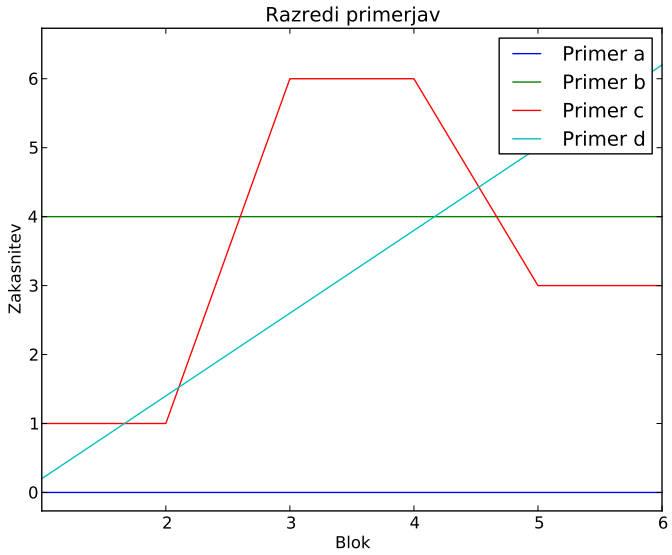
[2]Tako povezanih kot nepovezanih blokov.



(a) Identičen (b) Zamaknjen

(c) Stopničasto zamaknjen (d) Različna hitrost menjave sličic (FPS)

Slika 3.2: Prikaz štirih skupin preslikav



Slika 3.3: Graf zakasnitev štirih razredov primerjav s slike 3.2

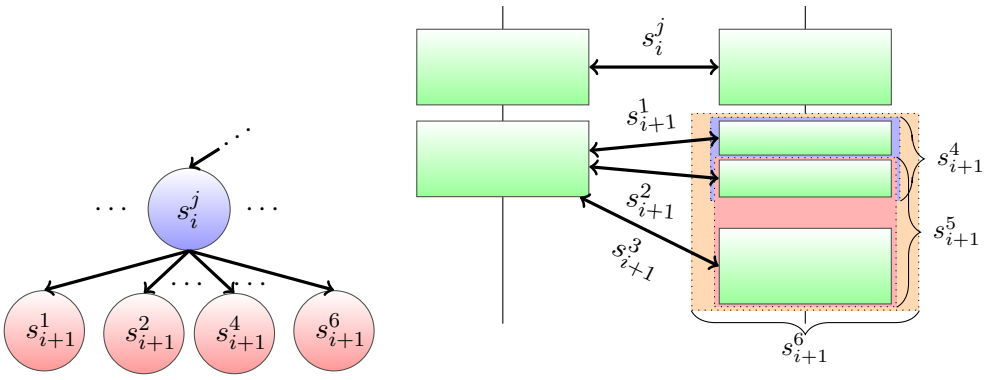
Primer razvrščanja v skupine med seboj zamenljivih podnapisov

Za primer recimo, da želimo podnapise razvrstiti v skupine med seboj zamenljivih podnapisov. Za izbran podnapis bi si izbrali primerjave z visoko pokritostjo[3] ter nizkim standardnim odklonom kvadratnih razlik med začetnimi časi skupin blokov. Vrednost zgornje meje standardnega odklona je odvisna od želene tolerance. Edini kriterij, ki še ostane, je povprečni zamik. Povprečni zamik bi se v tem primeru gibal okoli $300ms$, ki predstavlja zgornjo mejo, kjer gledalec še ne opazi zakasnitve.

3.2 Prostor stanj

Za delovanje preiskovalnega algoritma je potrebno določiti prostor stanj v katerem se išče najboljšo preslikavo. Stanje sem določil kot povezavo med dvema skupinama[4] blokov. V nadaljevanju diplomske naloge označujem stanja z s_i^j, kjer i označuje zaporedno število *nivoja v preiskovalnem drevesu* ali *zaporedno povezavo v preslikavi*. S črko j označujem zaporedno število kombinacije povezave znotraj istega nivoja. Pri prikazu in zapisu poti zaporedno število kombinacije j zanemarim, ker je postopek iskanja pri rezultatu

[3]Empirično sem ugotovil, da se pokritost za identične podnapise nahaja okoli 98%.
[4]Za vsako stran preslikave svojo skupino blokov.



(a) Prostor stanj (b) Blokovni prikaz

Slika 3.4: Prikaz vseh naslednikov stanja s_i v prostoru stanj kot v blokovnem prikazu

nezanimiv. Začetno stanje s_0 ima poseben pomen, ker predstavlja ničelno povezavo, ki je dodano, da omogoči uporabo preiskovalnega algoritma.

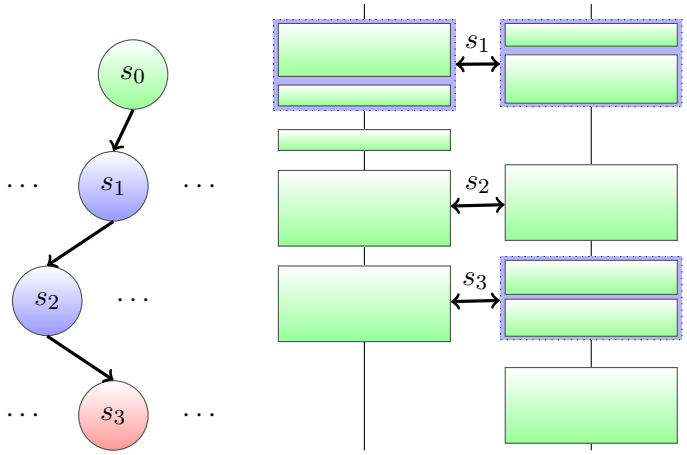
Nasledniki obiskanega stanja s_i so stanja povezanih skupin blokov ($s_{i+1}^j, j = 1, 2, 3, \dots$), ki se nahajajo za bloki zastopanimi v obiskanem stanju s_i. Slika 3.4 prikazuje konkreten primer, kjer imamo na koncu para preiskovanih podnapisov prikazano povezavo dveh blokov (3.4b), predstavljeno s stanjem s_i^j v prostoru stanj (3.4a). Na sliki so prikazane tudi različne možne povezave preostalih, še nepovezanih blokov. Skupine blokov na desni strani sem označil z obarvanimi pravokotniki, kjer modra barva predstavlja povezavo opisano s stanjem s_{i+1}^4, rdeča barva predstavlja povezavo opisano s stanjem s_{i+1}^5 in oranžna barva predstavlja povezavo opisano s stanjem s_{i+1}^6. Vse skupine blokov se povežejo z istim blokom na levi strani preslikave. Stanja brez naslednikov sem označil za končna stanja. S tako določenimi nasledniki stanj sem samo s prostorom stanj zadostil pravilu **B6**.

Izbrana pot stanj predstavlja celotno preslikavo. To pot se lahko uporabi za medsebojno primerjavo dveh podnapisov, saj vsebuje le povezane bloke. Na sliki 3.5 je prikazan primer take preslikave.

3.2.1 Ustvarjanje naslednikov

Zaradi različnih vrst preslikav[5] je potrebno ustvariti veliko število možnih poti. Vzemimo za primer preslikavo para podnapisov, oba s 60 bloki, katerih začetno stanje s_0 bo imelo približno $3,35 \cdot 10^6$ možnih poti. Do tako velikega števila možnosti pride zaradi množice različnih kombinacij povezav skupin

[5]Pravila **B1** do **B5**



(a) Prostor stanj (b) Blokovni prikaz

Slika 3.5: Primer preslikave

blokov na obeh straneh. Število možnih kombinacij skupin blokov na srečo omeji pravilo **B6**, ki zahteva, da se lahko v skupino blokov združi le zaporedni vrstni red blokov brez preskakovanja. Število kombinacij za podani primer se tako izračuna s sledečo vrsto:

$$1 + 2 + 3 \cdots + 60 = \frac{60 \cdot (60 + 1)}{2} = 1.830$$

Vsako skupino blokov je potrebno na eni strani povezati z vsako skupino blokov na drugi strani, torej imamo 1.830^2 (3.348.900) povezav. Posplošeno, vsako stanje s_i^j ima:

$$P(s_i^j) = \frac{N(s_i^j) \cdot (N(s_i^j) + 1)}{2} \cdot \frac{M(s_i^j) \cdot (M(s_i^j) + 1)}{2}$$

naslednikov, kjer funkciji N in M vrneta število skupin blokov na vsaki strani. Preiskovalni algoritem potemtakem preiskuje prostor z

$$\sum_{s_i^j \in \mathbb{S}} P(s_i^j)$$

stanji.

V praksi je nesmiselno pregledati vse možne kombinacije povezav, saj je to časovno potratno. Bolj pomembne so povezave blizu izhajajoče povezave, za smiselno omejitev pa sem določil, da lahko preiskovanje preskoči do največ *pet* blokov ter združi do največ *deset* blokov. S tem sem omejil število novih

stanj na:

$$\left(9 \cdot 5 + \frac{5 \cdot (5+1)}{2}\right)^2 = (45 \cdot 15)^2 = 60^2 = 3.600$$

Celoten algoritem namenjen ustvarjanju kombinacij skupin blokov za posamezno stran preslikave je v izpisu 5.

```
def block_combinations(blocks):
  max_skipped = 5
  for max_combined in range(1, 10):
    skipped = 0
    combination = []
    for block in blocks:
      combination.append(block)
      if max_combined == len(combination):
        yield subset
        # Pomakni za eno naprej
        combination = combination[1:]
        skipped += 1
        if skipped >= max_skipped:
          break
```

Izpis 5: Algoritem za ustvarjanje kombinacij skupin blokov

3.2.2 Preiskovanje

Pri preslikavah želimo čim manj združenih povezav, saj s tem zmanjšujemo ločljivost in s tem natančnost. Na sliki 3.6 je prikazana preslikava, kjer sta dve povezavi združeni v eno. Take združitve niso primerne, ker z njimi izgubljamo informacijo. Kot sem napisal že na primeru primerjave podnapisov s preslikavami, želim s pomočjo povezav priti do razlik v začetnih časih. Več kot jih je, bolj natančno lahko ugotovimo kako se dva podnapisa med seboj razlikujeta. Na isti sliki je prikazana tudi dobra preslikava, kjer je povezava opisana s stanjem s_2 razbita na povezavi opisani s stanjema s_2 in s_3. Prekomerno povezovanje blokov rešim z iskanjem najdaljših poti.

Zaradi stroge definicije naslednikov stanj so lahko nasledniki le tista stanja, ki predstavljajo povezave naslednjih blokov. Tako pridem do acikličnega grafa in lahko pri preiskovanju uporabim algoritme za iskanje najkrajših poti. Z določanjem negativnih uteži prevedem problem iskanja najkrajših poti v problem iskanja najdaljših poti.

Zaradi pravila **B5** se med nasledniki nahajajo ne le najbližje povezave, ampak vse možne kombinacije povezav. Iz tega razloga lahko pridem do

preslikav z velikim številom preskočenih povezav. Z iskanjem najdaljših poti se tej težavi spretno izognem. Na sliki 3.7 je prikazan primer take preslikave. Zaradi te lastnosti pride tudi do pojavitve istih stanj na različnih poteh.

Preizkusil sem preiskovalna algoritma požrešno iskanje (Greedy search) in A*. Glavna ideja za požrešnim iskanjem je v predpostavki dobro določenih cen ter dobrega klasifikatorja primernih stanj. Ob veljavni predpostavki ni potrebno preiskati drugih možnih poti. S požrešnim preiskovanjem lahko dosežem nizke čase iskanja in, v primeru veljavne predpostavke, ob tem ne izgubim veliko na pravilnosti preslikave.

Drugi preiskovalni algoritem, katerega sem preizkusil, je A*, ki sicer zagotovo najde najoptimalnejšo pot ob dopustni hevristiki (admissible heuristic) [10], a pri tem porabi več časa. Uporaba A* se mi zdi smiselna v primeru, če zgornja predpostavka ne drži. V takem primeru je potrebno preiskati več možnih poti, a ob pravilno izbrani hevristiki in cenah pa bo A* našla najoptimalnejšo pot in s tem najboljšo preslikavo.

Izbira hevristike

Za hevristiko pri preiskovalnem algoritmu A* sem izbral možno število stanj pS določeno z enačbo 3.1, ki je enako številu preostalih nepovezanih blokov za zadnjim blokom iz obiskanega stanja s_i. Tako preštejem število blokov na vsaki strani posebej (C_l in C_r) in za možno število stanj vzamem manjše število blokov. Poudariti je tudi potrebno, da se za hevristiko uporablja *negativna* vrednost možnega števila stanj.

$$pS(s_i) = min(C_l, C_r) \tag{3.1}$$

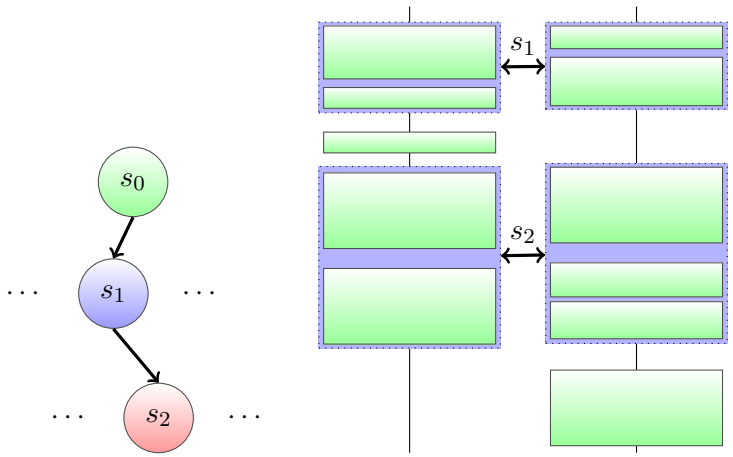
Izbrana hevristika je tudi dopustna, saj bo njena vrednost vedno manjša ali enaka pravi ceni poti. To je zaradi tega, ker povezava potrebuje vsaj en blok na vsaki strani je zgornja meja povezav enaka ali manjša od števila preostalih blokov. Do manjšega števila povezav pa lahko pride zaradi neprimernosti povezave, če je na eni strani blok, ki na drugi strani nima nobenega kandidata, s katerim bi se lahko povezal. Primeri takih blokov nastanejo pri vsebinskih razlikah med podnapisi in/ali različnih izdajah filma.

Izbira cen

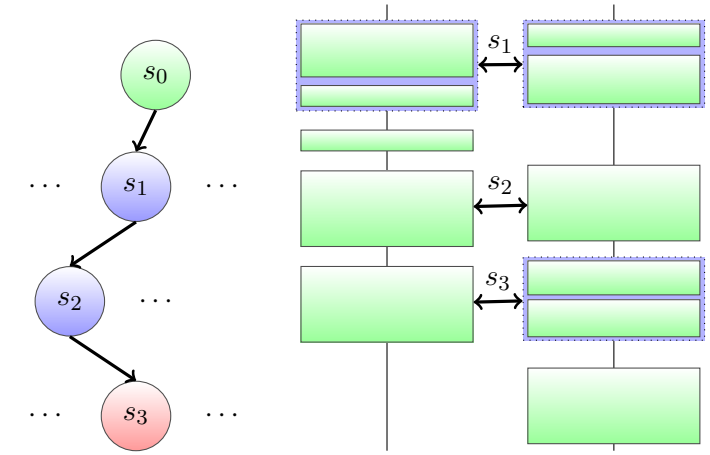
Za ceno povezave s_i^j leve strani skupine blokov $g_l(s_i^j)$ z desno stranjo skupine blokov $g_r(s_i^j)$ sem določil povprečje različnih hevrističnih ocen podobnosti:

pD razmerje trajanja obeh strani povezave (enačba 3.2),

rM relativno število možnih nadaljnjih povezav (enačba 3.3),

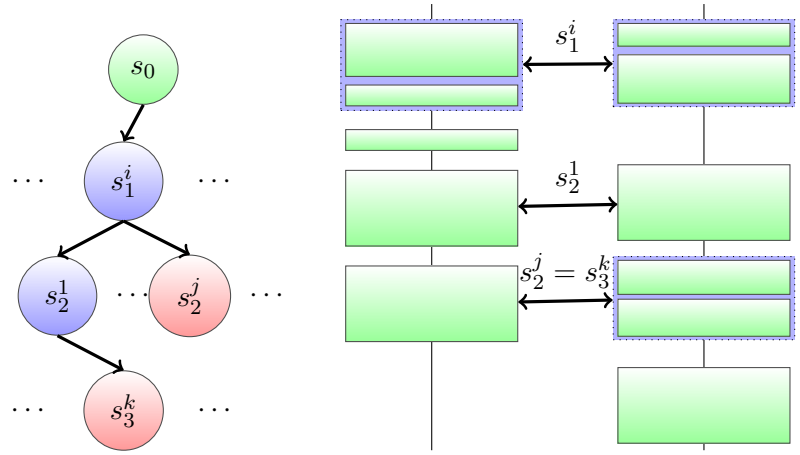


(a) Prikaz slabše poti (b) Blokovni prikaz slabe preslikave



(c) Prikaz dobre poti (d) Blokovni prikaz dobre preslikave

Slika 3.6: Primer daljše in krajše poti



(a) Prikaz grafa, kjer sta stanji s_2^j in s_3^k enaki

(b) Blokovni prikaz dveh možnih preslikav

Slika 3.7: Primer pojavitve istega stanja v dveh poteh

pC razmerje števil blokov obeh skupin povezave (enačba 3.4),

rD relativna zakasnitev povezave (enačba 3.5),

rL relativna dolžina vseh preskočenih enot (enačba 3.6),

S semantična podobnost.

Vsem določenim ocenam je skupno to, da so njihove vrednosti na intervalu med 0 in 1, zato je nekatere potrebno normalizirati.

Razmerje trajanja obeh strani povezave pD je izračunana z deljenjem trajanja ($l(x)$) najkrajše skupine blokov s trajanjem najdaljše skupine blokov v povezavi. Glavna ideja za to oceno je v predpostavki, da manjša ko je razlika trajanj obeh skupin blokov, bolj sta si podobni.

$$pD(s_i^j) = \frac{\min(l(g_l(s_i^j)), l(g_r(s_i^j)))}{\max(l(g_l(s_i^j)), l(g_r(s_i^j)))} \tag{3.2}$$

Cena povezave mora vsebovati tudi komponento iskanja najdaljših poti. Ta del cene je najbolj pomemben za požrešno preiskovanje, ker lahko predčasno zaključi z iskanjem. Relativno število možnih nadaljnjih povezav je izračunano enako kot pri hevristiki (enačba 3.1), le da je normalizirana glede na ostale možne poti ($s_i^k \in \mathbb{C}_{s_{i-1}}$).

$$rM(s_i^j) = \frac{pS(s_i^j)}{\max_{s_i^k \in \mathbb{C}_{s_{i-1}}}(pS(s_i^k))} \tag{3.3}$$

Kot pri razmerju trajanj obeh skupin blokov v povezavi, je ideja za razmerjem števila blokov obeh skupin (c_l in c_r) enaka. Manjša kot je razlika v številu blokov obeh skupin, bolj sta si skupini podobni. Namen te ocene je preprečitev prekomernega združevanja blokov.

$$pC(s_i^j) = \frac{\min(c_l(s_i^j), c_r(s_i^j))}{\max(c_l(s_i^j), c_r(s_i^j))} \tag{3.4}$$

Relativna zakasnitev rD v enačbi 3.5 je izračunana tako, da se zakasnitev povezave $d(s_i^j)$ deli z največjo zakasnitvijo v naboru možnih poti. Tako dobimo mero razdalje in če jo odštejemo od ena, dobimo mero podobnosti.

$$rD(s_i^j) = 1 - \frac{d(s_i^j)}{\max_{s_i^k \in \mathbb{C}_{s_{i-1}}}(d(s_i^k))} \tag{3.5}$$

V ceni povezave je potrebno dodati tudi mero s katero bi se preiskovalni algoritem odvračalo od prekomernega preskakovanja blokov. To se doseže z relativno dolžino vseh preskočenih enot podnapisa. Sešteje se trajanje vseh preskočenih enot $v_l \in \mathbb{V}_{s_i^j}$, tako leve kot desne strani preslikave. Nato se normalizira z največjim seštevkom trajanj preskočenih enot, ki jih opravijo stanja v naboru možnih poti. Da dobim mero podobnosti, dobljeno vrednost odštejem od ena.

$$rL(s_i^j) = 1 - \frac{\sum_{v_l \in \mathbb{V}_{s_i^j}} l(v_l)}{\max_{s_i^k \in \mathbb{C}_{s_{i-1}}}(\sum_{v_l \in \mathbb{V}_{s_i^k}} l(v_l))} \tag{3.6}$$

Semantična podobnost se izračuna z različnimi metodami, ki so opisani v nadaljevanju diplomske naloge. Opravil sem meritve tako z uporabo semantične analize kot tudi brez nje.

3.3 Klasifikator

Tudi zmanjšano število kombinacij je preveliko za smiselno preiskovanje. Znotraj možnih poti se nahaja veliko število povezav, med katerimi pa je le ena primerna. Za učenje klasifikatorja uporabim značilke pridobljene iz povezav. Klasifikator iz množice vseh možnih povezav izlušči le primerne kandidate za nadaljevanje postopka iskanja preslikave.

Povezava blokov nosi veliko značilk, katerih nekaj primerov lahko vidimo na sliki 3.8. Za učenje klasifikatorja in napovedovanje primernosti povezave sem izbral sledeče značilke:

Premori med enotami podnapisa na levi in desni strani: Dva atributa, en za levo in drug za desno stran preslikave. Atributa izražata vsoto trajanj vseh premorov med enotami.

Povprečno trajanje: Povprečno trajanje obeh skupin blokov.

Absolutni zamik: Absolutna razlika med začetnimi časi obeh skupin blokov.

Premori pred in za povezavo: Štirje atributi predstavljajo premor pred in za obema skupinama blokov.

Za klasifikator sem izbral odločitveno drevo in metodo podpornih vektorjev (SVM). Glavna prednost odločitvenega drevesa je enostavna interpretacija modela in enostavnejša uporaba, saj lahko dosežeš dobre rezultate brez mrežnega iskanja (grid search). SVM kot statistični model [9] je težje razumljiv in potrebuje večje število primerkov. Za SVM je potrebno narediti mrežno iskanje parametrov, sicer z njim ne moreš dobiti dobrih rezultatov. Dodatna zanimiva lastnost SVM je možnost izgradnje modela in njegova uporaba na grafičnih procesorjih (GPU) [7].

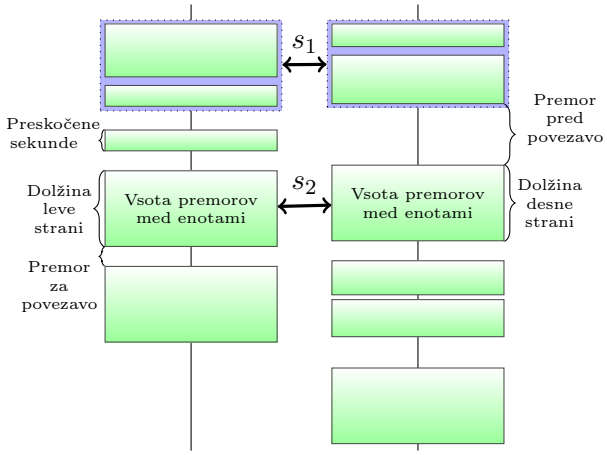
Za boljše delovanje SVM klasifikatorja je potrebno vse vrednosti atributov normirati [12]. Odločil sem se, da bom normirane vrednosti atributov uporabil tudi pri odločitvenem drevesu.

```
1  def construct_feature_list(link):
2    return (
3      sum(link.gaps['left']),
4      sum(link.gaps['right']),
5      link.duration['left'] + link.duration['right'] / 2.,
6      abs(link.duration['left'] - link.duration['right']),
7      min(link.wrapping_gaps['left']['before'], 1e4),
8      min(link.wrapping_gaps['left']['after'], 1e4),
9      min(link.wrapping_gaps['right']['before'], 1e4),
10     min(link.wrapping_gaps['right']['after'], 1e4),
11   )
```

Izpis 6: Izračun vektorja atributov

3.3.1 Učni in testni primeri

Klasifikator napoveduje primernost posamezne povezave, kar pomeni, da ena preslikava vsebuje več učnih primerov. Začel sem z ročnim preslikovanjem podnapisov in povezal 23 preslikav.



Slika 3.8: Nekaj značilk povezav v blokovnem prikazu, kjer stanje s_2 predstavlja trenutno obiskano stanje

Ročno narejene preslikave vsebujejo le primerne primerke povezav. Ker klasifikator potrebuje tudi množico negativnih primerkov, sem jih naredil s preprostim algoritmom, ki je izbral dva naključna podnapisa za različen film in ju preslikal z izbiranjem naključnih povezav. Pare podnapisov za različne filme sem izbiral, da preprečim lažne negativne primerke. S pomočjo opisanega algoritma sem naredil 288 preslikav.

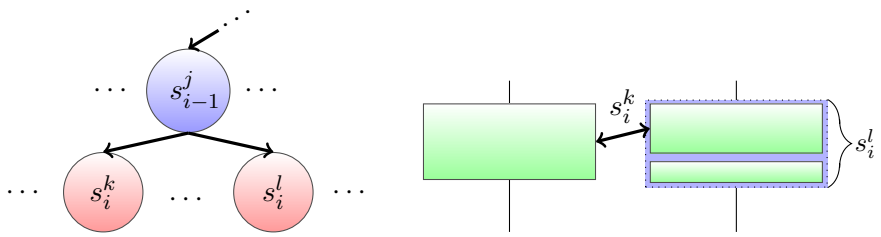
Na koncu sem ocenjeval preslikave naključnih parov podnapisov za isti film ter jih dodajal v učno množico. S tem sem želel dodati tudi specifične lažne pozitivne primere. Takih preslikav je 48, med njimi se nahajajo tudi pravilne preslikave.

Naključnih preslikav je več kot ročno preverjenih zaradi gostote povezav. Ročno preverjene preslikave imajo v povprečju okoli 80 povezav, medtem ko naključne preslikave imajo manj kot 30. Zaradi potrebe po približno enaki zastopanosti obeh razredov (primeren, neprimeren), sem dodal več naključno narejenih preslikav.

Zbrano množico preslikav sem uporabil za iskanje parametrov klasifikatorja. Za verodostojne meritve sem nato zbral še 99 novih preslikav za testno množico.

3.4 Semantično primerjanje

S semantičnim primerjanjem želim pridobiti podobnost povezave s katerimi bi pohitril in izboljšal iskanje preslikav. Glavna ideja je v iskanju primernosti, ne samo na nivoju časovnih lastnosti, ampak tudi na semantični podobnosti.



(a) Prikaz v prostoru stanj (b) Blokovni prikaz

Slika 3.9: Primer dveh primernih povezav, kjer ima s_i^l večjo vsebinsko podobnost

Želja je, da bi sistem najprej preiskal tiste poti, pri katerih so si povezave bolj vsebinsko podobne.

Pričakovano bi uporaba vsebinske primerjave izboljšale preslikovalnik na lokalnih odločitvah. Tak primer je prikazan na sliki 3.9. Glavna značilnost takih odločitev je v primernosti obeh povezav. S pomočjo semantične primerjave lahko na tak način dosežem izboljšavo pri odločanju.

Do sedaj sem se večinoma skliceval na levo in desno stran preslikave. Zaradi želje po manjšem številu slovarjev, sem pri semantični analizi definiral *izvorno* (stran s katere se prevaja) ter *ponorno* (stran v katero se prevaja) stran preslikave. Na ta način je potrebno zgraditi le polovico slovarjev. V diplomski nalogi sem se osredotočil le na slovenski in angleški jezik in namesto potrebe po angleško-slovenskih *in* slovensko-angleških slovarjih, potrebujem le slovensko-angleški slovar.

3.4.1 Podobnost kot priklic prevoda

V članku Building Parallel Corpora from Movies [15] so za vsebinsko primerjanje uporabili F oceno prevoda. Tako sem prišel na idejo, da bi nekaj podobnega uporabil tudi sam. Razlika, ki jo imam je v naboru ponornih besed. V članku so primerjali enoto z enoto podnapisa, medtem ko pri mojemu pristopu primerjam skupine blokov.

Postopek sem prilagodil tako, da vsako enoto izvorne strani preslikave primerjam z *okoliškimi* enotami ponorne strani preslikave. Zaradi te razlike ne morem uporabiti F ocene, zato sem uporabil le priklic prevoda. Problem nastane zaradi števila ponornih besed; moj postopek ima večji nabor le-teh in si z uporabo F ocene v splošnem zmanjšam količnik podobnosti (v F oceni je prisotna tudi natančnost prevoda, ki je odvisna od števila ponornih besed). Priklic prevoda v temu primeru lahko poimenujemo tudi kot razmerje med številom najdenih prevodov s številom izvornih besed.

Vsebinska primerjava poteka tako, da se za vsako enoto podnapisa izbere nabor izvornih besed, katere se primerja z naborom ponornih besed iz bližnje časovne okolice. Za vsako izvorno besedo iz nabora izvornih besed v slovenščini se vzame množico možnih prevodov v angleščini. Potem se za vsako besedo v ponornem jeziku preveri, če obstaja v množici prevodov, v primeru če se, se izvorno besedo jemlje kot najdeno v ponorni strani preslikave. Vsebinska podobnost se nato izračuna z enačbo 3.7, kjer nastopajo:

- *seznam* besed W_s,
- množica ponornih besed W_t,
- funkcija prevajanja $D(w)$, ki vrne množico prevodov besede w,
- funkcija $\delta(W_1, W_2)$, ki vrne 1, če je v $W_1 \cap W_2$ vsaj ena beseda.

$$\text{sim} = \frac{\sum_{w_s \in W_s} \delta\left(D(w_s), W_t\right)}{|W_s|} \tag{3.7}$$

Slovar besed

Metoda semantičnega primerjanja z F oceno potrebuje slovar za prevajanje posameznih besed. Ker nisem našel primernega slovensko-angleškega slovarja in tudi zaradi posebnosti jezika v podnapisih [13], sem se odločil za gradnjo lastnega slovensko-angleškega slovarja.

Za potrebe moje diplomske naloge potrebujem slovensko-slovenski, angleško-angleški in slovensko-angleški slovar. Za gradnjo teh treh slovarjev sem uporabil 5000 ustvarjenih preslikav, pri katerih sem zahteval vsaj 98% pokritost in standardni odklon zakasnitev manjši od 0,25s. Z določenima zahtevama sem zagotovil primernost preslikav za uporabo pri gradnji dvojezičnih slovarjev.

Iz preslikav sem vzel vse povezave skupin blokov. Za vsako besedo izvornega jezika sem določil seznam besed ponornega jezika skupaj s frekvenco in mero inverzne frekvence dokumentov (idf - inverse document frequency) [14]. Kot dokument se smatra izvorna beseda (N_{w_s} je število izvornih besed). Torej, večkrat kot se ponorna beseda w_t pojavi kot možni prevod (df_{w_t}), manjša je njena mera idf. Izračun mere idf za ponorno besedo w_t je prikazana v enačbi 3.8. K vsaki besedi izvornega jezika sem dodajal besede, ki se pojavijo v časovni okolici enote podnapisa v kateri se beseda nahaja. Na sliki 3.10 je prikazan izbor možnih prevodov.

$$idf(w_t) = \log\left(\frac{N_{w_s}}{df_{w_t}}\right) \tag{3.8}$$

Za vsako besedo je zgrajena tabela možnih prevodov skupaj z njihovo frekvenco f_{w_t,w_s} ter mero idf. Sedaj je za vsako izvorno besedo potrebno izbrati manjši izbor prevodov. Najprej seznam besed razvrstim po skupni frekvenci f_{w_t,w_s} iz katere vzamem prvih n_c besed. Izbranim besedam zmnožim njihove skupne frekvence z njihovimi merami idf ($f * idf$). Na ta način zmanjšam pomen pogostih besed in povečam pomen redkih besed. Izbrane besede razvrstim padajoče po pravkar izračunani meri $f * idf$ in vzamem $f * idf$ vrednost prve besede. V nabor prevodov dam vse besede, ki imajo mero $f * idf$ večjo od $u \cdot f * idf_{\max}$, kjer je u utež manjša od 1. Opisani algoritem za gradnjo slovarjev ima nastavljiva parametra n_c ter u.

Glavni pomen parametra n_c je v izločevanju premalo zastopanih besed v tabeli možnih prevodov izvorne besede. Z mero $f * idf$ želim povzpeti najbolj „podobne“ si besede na vrh seznama, sicer bi lahko uporabil kakšno od znanih mer podobnosti, kot sta recimo skupna informacijska vrednost (mutual information score) in koeficient Dice (Dice coefficient) [19]. Težava pri uporabi takih mer je v premajhni zastopanosti posameznih izvornih besed. Za primer si poglejmo besedi *hiša* in *house*. Frekvenca besede *hiša* je 775, medtem ko je frekvenca besede *house* 29.878. Razlog za tako veliko razliko v frekvencah je v različnih oblikah besede *hiša*. To težavo bi lahko rešil z lematizacijo besed (lemmatization), za katero potrebujemo „lematizatorje“, ki pa niso na voljo za vse jezike.

Za lažje razumevanje si poglejmo delovanje na konkretnem primeru besede *drevesa*[6]. Tabela 3.1 prikazuje seznam možnih prevodov, razvrščenih padajoče po skupni frekvenci f_{w_t,w_s}. V tem primeru sem za vrednost parametra n_c določil sedem, za u pa 50%. Črta med besedama *in* ter *are* v tabeli 3.1 ponazarja ločnico; vse kar je nad to črto, je izbrano za „naslednji krog“, kar je pod to črto, je odstranjeno. Tabela 3.2 ponazarja besede, razvrščene padajoče po $f * idf$, besedi *tree* in *trees* sta izbrani kot prevod besede *drevesa*. Pri besedi *tree* še velja:

$$f * idf_{tree} > 0,5 \cdot f * idf_{trees}$$

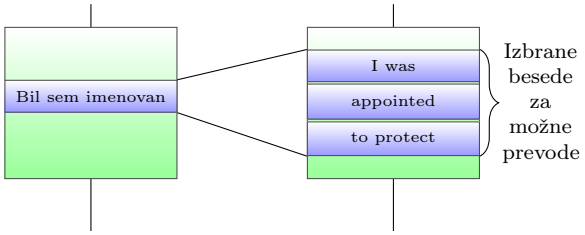
$$263,60 > 0,5 \cdot 356.91 = 178,46$$

Medtem ko pri besedi *the* ta neenačba ne velja več:

$$f * idf_{the} > 0,5 \cdot f * idf_{trees}$$

$$62,58 \not> 178,46$$

[6]Frekvence besed so dobljene iz zbirke 5.000 preslikav



Slika 3.10: Izbor besed ponornega jezika za možne prevode besede v izvornem jeziku

Beseda	f_{w_t,w_s}	Mera idf	$f * idf$
the	138	0,45	62,58
you	62	0,59	36,55
to	60	0,69	41,42
and	54	0,88	47,62
trees	52	6,86	356,91
tree	44	5,99	263,60
in	42	1,12	46,96
are	40	1,21	48,40
it	39	0,96	37,52
is	35	0,58	20,13
not	34	0,95	32,39
of	31	0,94	29,13
that	28	1,06	29,63
for	25	1,49	37,27
all	22	1,83	40,34
they	21	1,93	40,50
will	20	1,77	35,50
your	20	1,65	32,95
back	19	2,73	51,96

Tabela 3.1: Seznam možnih prevodov za besedo *drevesa*, besede so razvrščene padajoče po f_{w_t,w_s}

Beseda	f_{w_t,w_s}	Mera idf	$f * idf$
trees	52	6,86	356,91
tree	44	5,99	263,60
the	138	0,45	62,58
and	54	0,88	47,62
in	42	1,12	46,96
to	60	0,69	41,42
you	62	0,59	36,55

Tabela 3.2: Prečiščen seznam možnih prevodov za besedo *drevesa*, besede so razvrščene padajoče po $f * idf$

3.4.2 Hevristično razmerje vrst stavkov

Podnapisi so prevodi dialogov v filmih, zato so stavki v podnapisih večinoma kratki. Za prevode podnapisov je značilno tudi to, da je pri prevodu do neke mere potrebno zagotoviti vrstni red stavkov. Zaradi teh značilnosti sem prišel na idejo primerjanja vrst stavkov. Odločil sem se, da za določanje vrst stavkov uporabim le končna ločila. Odrekel sem se tudi uporabi pik, saj se pika uporablja v velikem številu primerov kot ne-končno ločilo in sem se s to odločitvijo izognil možnemu šumu.

Algoritem prešteje klicaje in vprašaje za vsako stran preslikave posebej. Nato za vsako skupino končnih ločil deli manjše število pojavitev skupine ločil z večjim, s čemer pridobim delež posameznih vrst stavkov. Na koncu naredim še obteženo vsoto vseh pojavitev končnih ločil. Pojavitve ločil obtežim z njeno zastopanostjo v povezavi. Izpis 7 prikazuje algoritem za izračun podobnosti na podlagi končnih ločil. Razlika med enačbo 3.9 ter opisanim algoritmom je v zaščiti pred ničlami v imenovalcu. V fazi razvoja sem ugotovil, da uporaba glajenja z dodajanjem dodatnega primera (add one smoothing)

poslabša rezultate.

$$\begin{aligned}
n_!(s_i^j) =& n_{!_l}(s_i^j) + n_{!_r}(s_i^j) \\
n_?(s_i^j) =& n_{?_l}(s_i^j) + n_{?_r}(s_i^j) \\
n_p(s_i^j) =& n_!(s_i^j) + n_?(s_i^j) \\
w_!(s_i^j) =& \frac{n_!}{n_p}(s_i^j) \\
w_?(s_i^j) =& \frac{n_?}{n_p}(s_i^j) \\
\mathrm{sim}(s_i^j) =& w_!(s_i^j) \cdot \frac{\min(n_{!_l}(s_i^j), n_{!_r}(s_i^j))}{\max(n_{!_l}(s_i^j), n_{!_r}(s_i^j))} + w_?(s_i^j) \cdot \frac{\min(n_{?_l}(s_i^j), n_{?_r}(s_i^j))}{\max(n_{?_l}(s_i^j), n_{?_r}(s_i^j))}
\end{aligned} \tag{3.9}$$

Za bolj natančen izračun deleža oblik stavkov bi skupaj z bolj naprednim razčlenjevanjem stavkov potreboval tudi klasifikator, s katerim bi določali vrsto stavka. S pomočjo takega načina bi lahko primerjal ne le dve obliki stavkov, ampak tudi druge, kot so povedni, nikalni, itd.

```
def sim(left_text, right_text):
  left_exclamation_mark = float(left_text.count('!'))
  left_question_mark = float(left_text.count('?'))
  right_exclamation_mark = float(right_text.count('!'))
  right_question_mark = float(right_text.count('?'))
  exclamation_total = left_exclamation_mark + right_exclamation_mark
  question_total = left_question_mark + right_question_mark
  total = exclamation_total + question_total

  exclamation = min(left_exclamation_mark, right_exclamation_mark)
  if left_exclamation_mark + right_exclamation_mark > 0:
    exclamation /= max(left_exclamation_mark, right_exclamation_mark)
  question = min(left_question_mark, right_question_mark)
  if left_question_mark + right_question_mark > 0:
    question /= max(left_question_mark, right_question_mark)

  if total > 0:
    return  exclamation_total / total * exclamation \
          + question_total / total * question
  else:
    return 1.
```

Izpis 7: Algoritem za izračun vsebinske podobnosti s primerjanjem ločil

3.5 Metrike ocenjevanja

Za ocenjevanje preslikovalnika je potrebno določiti način primerjanja rezultata sistema z vnaprej določenimi rezultati. Na nekakšen način je potrebno tudi oceniti vpliv posameznih komponent sistema. Glavni komponenti ocene sta klasifikator primernosti stanj in sam preiskovalni algoritem. Najbolj pomemben del preiskovalnega algoritma je določitev cen in hevristike za posamezne poti med stanji.

3.5.1 Primerjanje z referenčnimi preslikavami

Za ocenjevanje delovanja preslikovalnika sem uporabil F oceno. Za izračun F ocene je potrebno izračunati natančnost (precision) P in priklic (recall) R, za kar potrebujem metodologijo določanja resničnih in lažnih pozitivnih ter resničnih in lažnih negativnih primerov. Matrika zamenjav (confusion matrix) je prikazana v tabeli 3.3.

$$P = \frac{TP}{TP+FP} \tag{3.10}$$

$$R = \frac{TP}{TP+FP} \tag{3.11}$$

$$F = \frac{2 \cdot P \cdot R}{P+R} \tag{3.12}$$

Za resnično pozitivne primere sem določil tiste povezave, ki se nahajajo v referenčnih preslikavah (enačba 3.13). Če povezava v referenčni preslikavi ni prisotna v izračunani preslikavi, se to smatra kot lažno negativen primer (enačba 3.16). Lažno pozitivni primeri so tiste povezave izračunane preslikave, ki se v referenčni preslikavi ne pojavijo (enačba 3.14). Za vse nepovezane bloke, tako v referenčni, kot tudi v izračunani preslikavi, se jih obravnava kot resnično negativni primeri (enačba 3.15). Na sliki 3.11 je upodobljen primer metrik.

$$TP = |\mathbb{S}_c \cap \mathbb{S}_r| \tag{3.13}$$

$$FP = |\mathbb{S}_c \setminus \mathbb{S}_r| \tag{3.14}$$

$$TN = |\mathbb{B} \setminus \{b; \forall s \in \mathbb{S}_c \cup \mathbb{S}_r \wedge b \in \mathbb{B}_s\}| \tag{3.15}$$

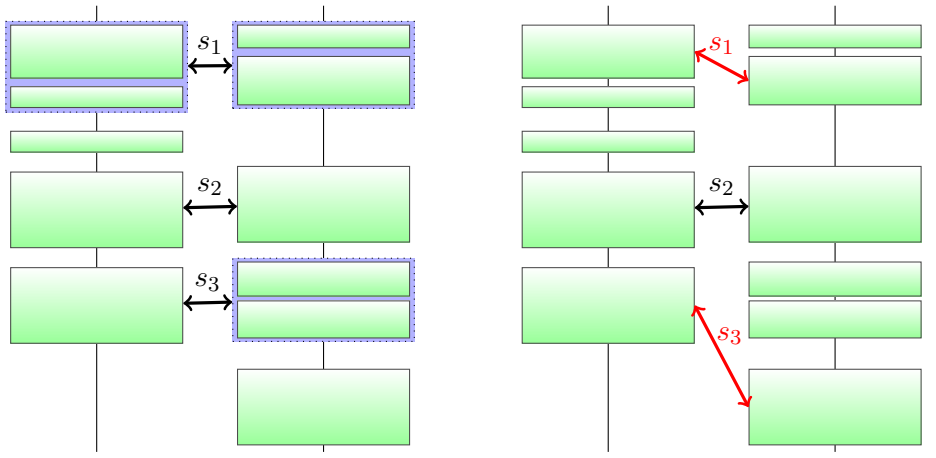
$$FN = |\mathbb{S}_r \setminus \mathbb{S}_c| \tag{3.16}$$

3.5.2 Ocenjevanje učinkovitosti

Za ocenjevanje učinkovitosti preslikovalnika sem meril povprečni čas iskanja preslikave, povprečno število obiskanih stanj ter povprečno razmerje izbranih od možnih naslednikov. Povprečni čas izračuna preslikave je preprosta

		Napovedani razred: Primerna povezava	Napovedani razred: Neprimerna povezava
Dejanski razred	Primerna povezava	Resnično pozitiven (TP)	Lažno negativen (FN)
	Neprimerna povezava	Lažno pozitiven (FP)	Resnično negativen (TN)

Tabela 3.3: Matrika zamenjav



(a) Pravilna preslikava (tri resnično pozitivni primeri in dva resnično negativna primera)

(b) Preslikava z napakami (en resnično pozitiven primer, dva lažno pozitivna primera, en resnično negativen primer in dva lažno negativna primera)

Slika 3.11: Primer dveh preslikav z izračunanimi metrikami

metrika, s katero želim oceniti hitrost preslikovalnika. S pomočjo te metrike sem lahko ocenil vpliv spremembe določenega podsistema na izvajalni čas celotnega postopka.

Druga, zelo pomembna, metrika je povprečno število obiskanih stanj. Preslikave se gradijo s pomočjo preiskovalnega algoritma. Tako je v primeru preiskovalnega algoritma A* pomembno vedeti tudi, koliko stanj je pri gradnji preslikave obiskal. Več stanj je preiskovanje pregledalo, dlje časa potrebuje za iskanje najboljše preslikave.

Zadnja metrika za ocenjevanje učinkovitosti preslikovalnika je merjenje števila izbranih stanj. Pri vsakem obiskanemu stanju se ustvari večje število možnih naslednikov. Število teh možnih stanj se zmanjša z uporabo klasifikatorja, ki oceni, katera stanja so primerna. Manjše kot je to razmerje, manj možnosti je potrebno pregledati, kar poveča učinkovitost celotnega postopka.

3.6 Zahtevnostni razredi

V času razvoja sem opazil vzorec napak preslikovalnika, saj je v določenih primerih izbral napačno odločitev in preiskal prevelik del prostora stanj. Te napačne odločitve povzročajo različni dejavniki, kot so na primer veliko število podobnih si blokov znotraj majhne okolice, velika razdrobljenost blokov, itd.

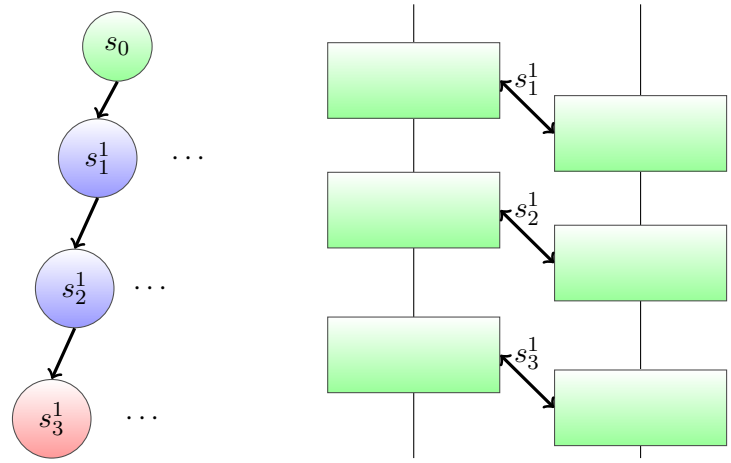
Vpliv vseh teh dejavnikov je težko oceniti, zato sem se odločil za uporabo hevristične ocene. Pri pregledovanju preslikav se je izkazalo, da standardni odklon absolutnih zakasnitev, razmerje števila blokov posamezne strani preslikave (C, enačba 3.17) ter pokritost same preslikave odražajo zahtevnost preslikave. S to ugotovitvijo sem vpeljal intervale za vsak kriterij posebej in pravilo določanja zahtevnostnega razreda na osnovi teh treh kriterijev. Pravilo je preprosto, vsak kriterij določi zahtevnostni razred, med izbranimi se izbere najzahtevnejši razred.

Določil sem štiri zahtevnostne razrede. Enostavni zahtevnostni razred predstavlja tiste preslikave, katere se lahko reši s preprostim algoritmom - izberi prvi nepovezani blok na levi in ga poveži s prvim nepovezanim blokom na desni strani preslikave. Primer take preslikave je na sliki 3.12. Lahek razred predstavlja preslikave, ki so za preslikovalnik enostavni. Naslednja dva razreda predstavljata odmik po zahtevnosti od lahkega zahtevnostnega razreda, pri čemer se zelo težak zahtevnostni razred lahko smatra kot razred preslikav, ki so že skoraj nedoločljivi. Intervali posameznih zahtevnostnih razredov ter njihova zastopanost v testni množici je določena v tabeli 3.4.

$$C(M) = \frac{\min(|\mathbb{B}_{M,l}|, |\mathbb{B}_{M,r}|)}{\max(|\mathbb{B}_{M,l}|, |\mathbb{B}_{M,r}|)} \tag{3.17}$$

Zahtevnostni razred	Intervali			Število preslikav v množici
	Standardni odklon	Razmerje števil blokov	Pokritost	
Enostaven	0s - 5s	- 100%	- 100%	5
Lahek	5s - 10s	100% - 80%	100% - 95%	35
Težak	50s - 10s	80% - 60%	95% - 90%	31
Zelo težak	- 50s	60% -	90% -	28

Tabela 3.4: Zahtevnostni razredi



(a) Predstavitev s prostorom stanj

(b) Blokovna predstavitev

Slika 3.12: Primer primerjave v enostavnem zahtevnostnem razredu

Zahtevnostni razredi nam omogočajo tudi lažji vpogled nad katerimi preslikavami imamo težave. Ugotovimo tudi lahko kakšen vpliv ima določena sprememba nad različnimi zahtevnostnimi razredi.

Poglavje 4

Izbira parametrov klasifikatorja

Za uporabo klasifikatorjev je potrebno poiskati optimalne parametre, kar še posebej velja za metode podpornih vektorjev [12]. Iskanja parametrov sem se lotil z mrežnim iskanjem (grid search). Za vsak parameter sem vzel po dvajset vzorcev, zaradi časovnih omejitev nisem izbral višje ločljivosti.

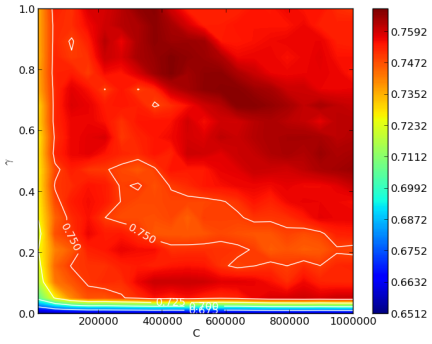
Za iskanje parametrov sem uporabil učno množico. Za vsak par parametrov sem izvedel tako prečno preverjanje klasifikatorja, kot tudi celoten postopek preslikovanja. Pri iskanju preslikav sem uporabil požrešno preiskovanje, za ceno sem uporabil le podobnosti na podlagi časovnih lastnosti. Meril sem F oceno, pri preslikovalniku pa še povprečno razmerje med primernimi in možnimi povezavami. Z merjenjem razmerja primernih povezav sem raziskoval kakšna je povezava med številom primernih povezav in F oceno.

4.1 Metoda podpornih vektorjev

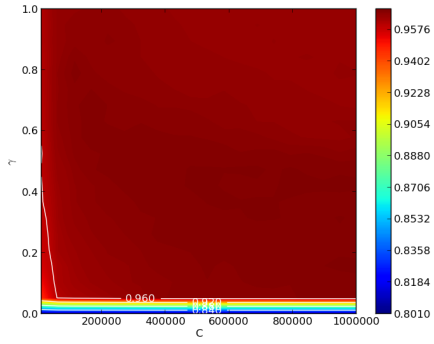
Pri izbiri začetnega intervala parametrov sem pri parametru γ izbral interval od nič do ena, saj sem iz privzete vrednosti uporabljene implementacije SVM, ki je $\frac{1}{n_a}$, kjer je n_a število uporabljenih atributov [17], sklepal, da je za parameter γ bolje, da je manjše od 1. Avtorji uporabljene implementacije SVM-ja predlagajo vrednost parametra C nad 10.000. Na podlagi predloga sem se za parameter C odločil za interval od 10.000 do 1.000.000.

Meritev

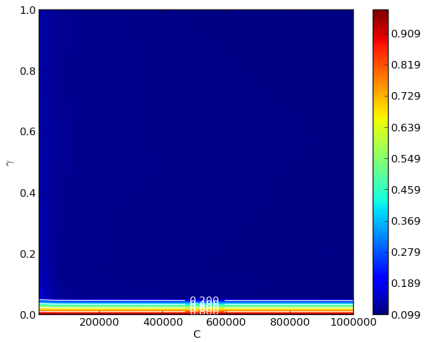
Rezultati meritve so prikazani na slikah 4.1. Zanimivo je, da se F ocena preslikovalnika okoli najvišje točke ne spreminja, razen ob nizkih vrednostih parametra γ. Na podlagi te slike sem se odločil, da zaključim z iskanjem parametrov in izberem parametre najvišje točke ($F = 0,77$) grafa 4.1a, ki



(a) F ocena preslikovalnika



(b) F ocena klasifikatorja



(c) Povprečni delež primernih povezav

Slika 4.1: Rezultati mrežnega iskanja parametrov za klasifikator SVM na intervalih $C = [10.000, 1.000.000], \gamma = (0, 1]$

sta:

- $C = 426842$,
- $\gamma = 0,78947$.

Razlika med F ocenami klasifikatorja in povprečnimi deleži primernih povezav je še manjša, kot je vidno na grafih 4.1b in 4.1c. Največji vpliv ima parameter γ, ki more biti večji od 0,05. Iz grafov je tudi razvidno, da s padanjem parametra C pada tudi F ocena preslikovalnika. Na grafu 4.1c se vidi majhno rast z zmanjševanjem parametra C.

4.2 Odločitveno drevo

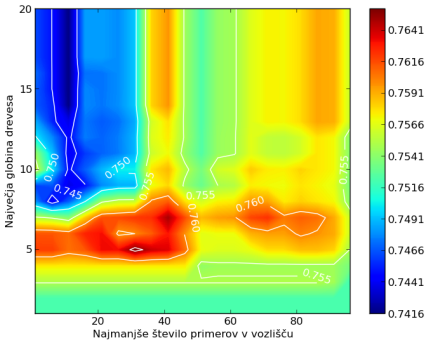
Izbrana implementacija odločitvenega drevesa ima dva parametra [1], najmanjše število primerov v vozlišču ($\min_N$) in največja globina drevesa ($\max_d$). Za najmanjše število primerov v vozlišču sem si izbral interval od 1 do 95, z večjim številom bi verjetno že preveč omejil odločitveno drevo. Za največjo globino drevesa sem določil interval od 1 do 20.

Meritev

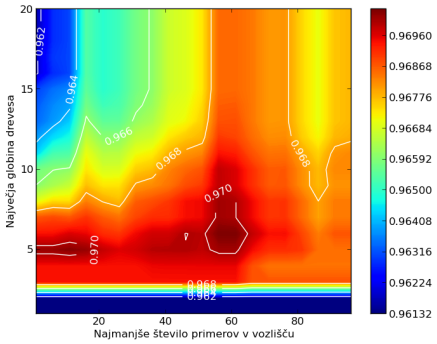
Rezultati meritev za odločitveno drevo so prikazani na slikah 4.2. Tako kot pri klasifikatorju SVM, se tudi F ocene različnih parametrov od preslikovalnika z odločitvenim drevesom (prikazan na grafu 4.2a) ne razlikujejo veliko. Razlika med grafi rezultatov klasifikatorja SVM in odločitvenega drevesa je le v manjših spremembah F ocene. Najvišja točka ($F = 0,77$) na grafu 4.2a je pri parametrih:

- $\min_N = 31$,
- $\max_d = 5$.

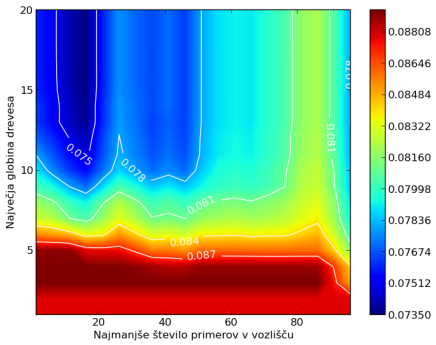
Ob primerjavi grafov 4.1 in 4.2, opazimo, da ima odločitveno drevo v splošnem boljše rezultate od SVM. SVM pa je v najvišji točki dosegel za dva promila boljši rezultat, kar pa lahko zanemarimo. Kako se celoten preslikovalnik obnaša pri enem in drugem klasifikatorju so pokazale meritve nad ločeno testno množico.



(a) F ocena preslikovalnika



(b) F ocena klasifikatorja



(c) Povprečni delež primernih povezav

Slika 4.2: Rezultati mrežnega iskanja parametrov za odločitveno drevo nad intervalom parametra $\max_d = [0 - 20]$ in parametra $\min_N = (0 - 95]$ s korakom 5

Poglavje 5

Rezultati poskusov

Poskuse preslikovalnika sem izvajal nad ločeno testno množico z 99 preslikavami. V tabeli 3.4 je prikazana zastopanost posameznega zahtevnostnega razreda znotraj testne množice. Za vsako kombinacijo klasifikatorja, preiskovalnega algoritma in algoritma semantičnega primerjanja sem izvedel meritev in opazoval sledeče metrike:

- F ocena,
- povprečni čas,
- povprečno število obiskanih stanj,
- povprečno število izbranih stanj.

Ob izbiranju prave poti ima preslikovalnik na voljo več možnih poti. Klasifikatorjeva naloga znotraj preslikovalnika je izbor primernih kandidatov iz množice vseh možnih stanj. Povprečno število izbranih stanj je torej povprečno število kandidatov, ki jih je klasifikator izbral kot možne kandidate in prikazuje za koliko zmanjša množico možnih poti. Tukaj naj spomnim, da je zaradi problema največje število možnih stanj 3.600.

Preiskovalni algoritem raziskuje prostor in odvisno od števila poti ter njihovih cen se lahko preišče manjši ali večji del določenega prostora. Z metriko povprečnega števila obiskanih stanj lahko opazujemo kako se obnaša preiskovalni algoritem. Glede na to, da požrešno preiskovanje pri vsakem stanju izbere le eno pot in se ne vrača nazaj, je ta metrika uporabna predvsem za preiskovalni algoritem A*.

Za določanje učinkovitosti preslikovalnika merim še metriki F ocene in povprečen čas potreben za iskanje preslikave med dvema podnapisoma. Z F oceno lahko ocenimo, kako dobro deluje preslikovalnik. Ker mora biti preslikovalnik uporaben tudi v praksi za primer razvrščanje v skupine sorodnih

podnapisov, je potrebno meriti čas, potreben za iskanje preslikave med dvema podnapisoma.

5.1 Meritve preslikovalnikov brez semantičnega primerjanja

Začel sem s preslikovalniki brez semantičnega primerjanja, kjer sem opazoval povprečno število izbranih stanj. Ta metrika je odvisna predvsem od klasifikatorja in je nisem opazoval pri ostalih dveh preslikovalnikih s semantičnim primerjalnikom. Rezultati meritev so prikazani v tabeli 5.1.

Pri povprečnemu številu izbranih stanj opazimo, da klasifikator SVM izbere nekaj več kandidatov kot odločitveno drevo, vendar glede na to, da je pri vsakem stanju na voljo do 3.600 možnih novih stanj, postane okoli 30 izbranih stanj več zanemarljivo. Druga zanimiva stvar je, da ima enostaven zahtevnostni razred več izbranih stanj kot ostali zahtevnostni razredi. Razlog je verjetno v manjšem številu primerkov v enostavnem zahtevnostnem razredu, ki so verjetno vsi sestavljeni pretežno iz filmov in imajo zaradi tega globlje preiskovalno drevo.

Povprečno število obiskanih stanj pokaže, da preiskovalni algoritem A* preišče večji del prostora. Pri enostavnem zahtevnostnem razredu pride do rešitve s preiskovalnim algoritmom A* zelo hitro, kljub temu, da je preiskovalno drevo v povprečju globlje za 60 stanj, kar je razvidno iz meritev preslikovalnikov s požrešnim algoritmom preiskovanja. Iz tega lahko sklepam, da je kriterij za določanje zahtevnosti preslikav primeren. Še ena zanimivost, ki jo lahko vidimo v rezultatih je, da mora preslikovalnik z odločitvenim drevesom preiskati manjši prostor kot s klasifikatorjem SVM.

Povprečni časi izvajanja sovpadajo s povprečnim številom obiskanih stanj. Iz primerjave obeh preslikovalnikov s požrešnim preiskovanjem se vidi, da klasifikator SVM v splošnem potrebuje več časa kot odločitveno drevo, navkljub dejstvu, da oba preiščeta prostor enako.

F ocene preslikovalnikov pokažejo, da ima preiskovalni algoritem A* boljše rezultate kot požrešno preiskovanje, kar je pričakovano. So pa rezultati požrešnega preiskovanja zadovoljivi in bistveno hitrejši.

		Zahtevnostni razred	F ocena	Povprečni čas	Povprečno število obiskanih stanj	Povprečno število izbranih stanj
SVM	A*	Enostaven	100%	170,56s	113,80	199
		Lahek	92,90%	1015,34s	786,23	139
		Težak	79,18%	2039,40s	1659,68	137
		Zelo težak	82,96%	2422,56s	1927,29	132
		Skupaj	86,15%	1691,34s	1348,49	136
	Požrešno preiskovanje	Enostaven	100%	102.28s	60,00	184
		Lahek	84,18%	82.46s	47,17	146
		Težak	61,31%	62,03s	33,52	126
		Zelo težak	70,51%	85,34s	47,46	135
		Skupaj	73,95%	77,88s	43,63	140
Odločitveno drevo	A*	Enostaven	100%	119,06s	105,80	160
		Lahek	93,73%	621,16s	788,31	119
		Težak	80,20%	1030,17s	1393,39	113
		Zelo težak	83,95%	1194,39s	1621,68	101
		Skupaj	87,04%	886,00s	1179,01	110
	Požrešno preiskovanje	Enostaven	100%	72,90s	60,00	151
		Lahek	85,98%	59,07s	46,83	120
		Težak	58,48%	47,23s	33,84	107
		Zelo težak	66,12%	64,17s	45,96	103
		Skupaj	72,64%	57,50s	43,18	114

Tabela 5.1: Rezultati poskusov brez semantičnega primerjanja.

		Zahtevnostni razred	F ocena	Povprečni čas	Povprečno število obiskanih stanj
SVM	A*	Enostaven	100%	2316,61s	1385,60
		Lahek	91,84%	2834,06s	1896,91
		Težak	80,83%	3436,29s	2430,00
		Zelo težak	79,82%	4502,57s	3167,43
		Skupaj	85,41%	3468,40s	2397,35
	Požrešno preiskovanje	Enostaven	100%	372,20s	60,00
		Lahek	87,58%	258,98s	46,80
		Težak	59,50%	200,98s	33,97
		Zelo težak	75,13%	275,34s	48,89
		Skupaj	79,02%	251,16s	44,04
Odločitveno drevo	A*	Enostaven	100%	1201,63s	1149,40
		Lahek	92,16%	1540,69s	1742,66
		Težak	82.82%	1888.27s	2214,84
		Zelo težak	80,91%	2176,34s	2550,29
		Skupaj	80,32%	1812,18s	2088,97
	Požrešno preiskovanje	Enostaven	100%	329,69s	60,00
		Lahek	87,34%	238,05s	46,29
		Težak	71,88%	203,93s	35,74
		Zelo težak	75,02%	235,85s	47,82
		Skupaj	79,65%	231,37s	44,11

Tabela 5.2: Rezultati poskusov s semantično metodo „podobnost kot priklic prevoda“.

5.2 Meritve preslikovalnikov s semantično podobnostjo kot priklic prevoda

Nadaljeval sem z meritvami preslikovalnikov s semantičnim primerjanjem s podobnostjo kot priklicem prevoda. Od te meritve dalje nisem več opazoval metrike „povprečno število izbranih stanj“, saj se zaradi uporabe semantičnega primerjalnika ne spremeni. Rezultati so zapisani v tabeli 5.2.

Opisana razmerja ostajajo enaka kot pri prejšnjem sklopu meritev. Razlika se pojavi v višjih F ocenah pri vseh preslikovalnikih, kot tudi v višjih povprečnih časih in večjem povprečju obiskanih stanj pri preiskovalnem algoritmu A*.

		Zahtevnostni razred	F ocena	Povprečni čas	Povprečno število obiskanih stanj
SVM	A*	Enostaven	100%	154,07s	100,60
		Lahek	93,70%	1304,11s	937,49
		Težak	81,75%	2512,57s	1863,06
		Zelo težak	84,35%	2787,22s	2117,61
		Skupaj	87,63%	2043,90s	1518,82
	Požrešno preiskovanje	Enostaven	100%	104,50s	60,00
		Lahek	83,74%	77,99s	44,57
		Težak	69,62%	58,71s	32,19
		Zelo težak	69,30%	81,00s	43,61
		Skupaj	76,06%	74,14s	41,20
Odločitveno drevo	A*	Enostaven	100%	110,38s	91,20
		Lahek	94,05%	718,43s	925,51
		Težak	84,05%	1165,95s	1572,00
		Zelo težak	85,62%	1290,62s	1741,43
		Skupaj	88,83%	989,69s	1316,58
	Požrešno preiskovanje	Enostaven	100%	88,53s	60,00
		Lahek	83,63%	68,04s	43,89
		Težak	70,02%	55,30s	33,42
		Zelo težak	68,36%	68,96s	42,43
		Skupaj	75,87%	65,35s	41,01

Tabela 5.3: Rezultati poskusov s semantično metodo „hevristično razmerje vrst stavkov“.

5.3 Meritve preslikovalnikov s semantično podobnostjo kot razmerje končnih ločil

Za zadnji sklop meritev sem izbral preslikovalnike z razmerjem končnih ločil (oziroma hevristično razmerje vrst stavkov). Rezultati so zapisani v tabeli 5.3.

Rezultati teh meritev kažejo, da se preslikovalniki z uporabo razmerja končnih ločil obnašajo malo bolje kot brez semantičnega primerjalnika, a za to potrebujejo več časa, poveča pa se tudi preiskovani prostor pri A*.

5.4 Najboljši in najučinkovitejši preslikovalnik

Do sedaj sem predstavil rezultate meritev in opisal, kako se različni preslikovalniki razlikujejo med seboj. Za uporabo preslikovalnika v praksi je

potrebno izbrati najboljšega ali najučinkovitejšega, odvisno od namena uporabe. Pri uporabi kjer je rezultat pomembnejši od učinkovitejše porabe časa, potrebujemo preslikovalnik z najvišjo F oceno. V primerih uporabe kjer je pomemben tudi čas, pa je potrebno uporabiti najučinkovitejši preslikovalnik. Na grafih 5.1 in 5.2 so podsistemi preslikovalnikov označeni s kraticami oziroma besedami:

DT uporaba klasifikatporja odločitveno drevo,

SVM uporaba klasifikatorja SVM,

A* uporaba preiskovalnega algoritma A*,

Požrešno uporaba preiskovalnega algoritma požrešno preiskovanje,

brez semantike brez uporabe semantičnega primerjanja,

s slovarjem uporaba semantičnega primerjalnika podobnost kot priklic prevoda,

s primerjanjem ločil uporaba semantičnega primerjalnika podobnost kot razmerje ločil.

5.4.1 Najboljši preslikovalnik

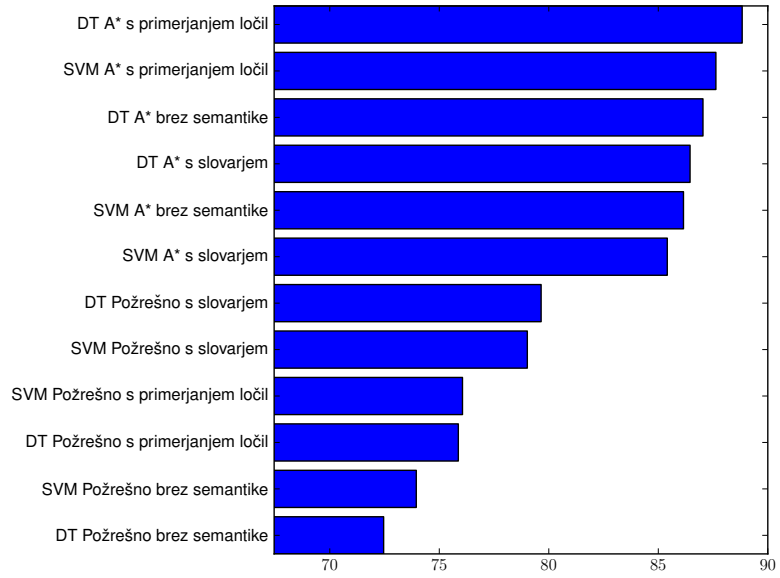
Iz grafov F ocen preslikovalnikov na sliki 5.1 lahko razberemo, da je najboljši preslikovalnik odločitveno drevo s preiskovalnim algoritmom A* in semantičnim primerjalnikom podobnost kot razmerje ločil. Iz grafa vidimo tudi, da so vsi preslikovalniki s preiskovalnim algoritmom A* boljši od preslikovalnikov s požrešnim preiskovanjem.

V skupini preslikovalnikov s požrešnim preiskovanjem kot boljša izstopata preslikovalnika s semantičnim primerjalnikom podobnost kot priklic prevoda. Očitno je tudi to, da se pri požrešnem preiskovanju odločitveno drevo bolje obnese s semantičnim primerjalnikom kot brez semantične primerjave.

Odločitveno drevo deluje bolje tudi s preiskovalnim algoritmom A*. Če pogledamo zgornje štiri preslikovalnike na grafu, vidimo večinoma le preslikovalnike z odločitvenim drevesom. Semantični primerjalnik podobnost kot priklic prevoda poslabša rezultate preslikovalnikov s preiskovalnim algoritmom A*, kar se vidi iz rezultatov.

5.4.2 Najučinkovitejši preslikovalnik

Za iskanje najučinkovitejšega preslikovalnika sem uporabil razmerje med F oceno ter povprečnim časom, potrebnim za iskanje preslikave. Ker absolutne

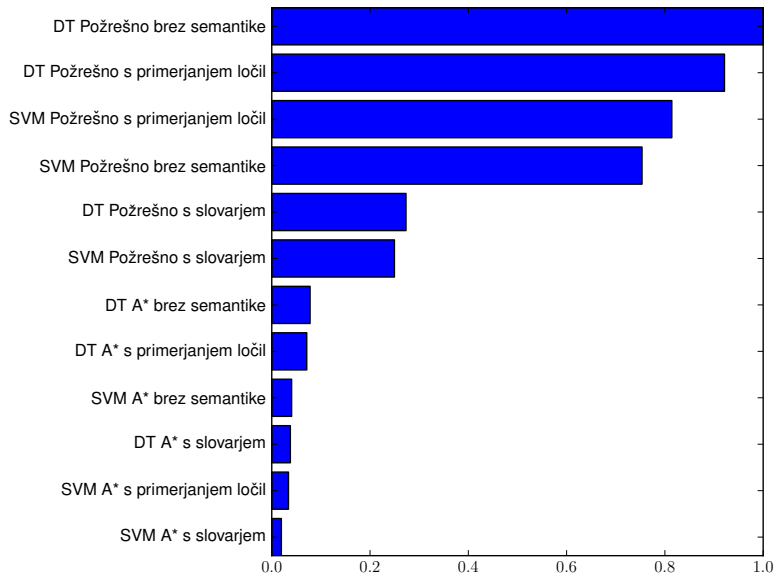


Slika 5.1: Graf F ocen vseh preslikovalnikov, razvrščenih od boljšega proti slabšemu.

vrednosti tega razmerja ne predstavljajo ničesar, sem vse vrednosti normaliziral, zato je skala v grafu 5.2 od 0 do 1.

Najučinkovitejši preslikovalnik je preslikovalnik z odločitvenim drevesom, požrešnim preiskovanjem in brez semantičnega primerjanja. Takoj za njim sledi preslikovalnik s semantičnim primerjalnikom, podobnost kot razmerje ločil. Zanimivo je, da sta preslikovalnika z istim semantičnim primerjalnikom, a s klasifikatorjem SVM, slabša.

Za razliko od grafa 5.1 so sedaj na vrhu preslikovalniki s požrešnim preiskovanjem. Razlog za to je v izredni potratnosti in majhni izboljšavi rezultatov z uporabo preiskovalnega algoritma A*.



Slika 5.2: Graf razmerij cen in učinkovitosti vseh preslikovalnikov, razvrščeni od boljšega proti slabšemu.

Poglavje 6

Sklep

Pri razvoju preslikovalnika sem uporabil tako časovne lastnosti blokov kot tudi semantično primerjavo. Za iskanje preslikave med dvema podnapisoma sem določil prostor stanj, v katerem eno stanje predstavlja eno povezavo blokov obeh strani preslikave, v cenovno funkcijo pa vpeljal različne kriterije časovnih lastnoti. Za zmanjševanje števila možnih poti sem preizkusil in uporabil dva klasifikatorja. Razvil sem dve semantični metodi primerjanja, ki sem ju tudi preizkusil in z njima dopolnil cenovno funkcijo.

Rezultati poskusov so pokazali zelo velik vpliv preiskovalnega algoritma na rezultat. Glede na F oceno so razlike med klasifikatorji ter semantičnim primerjalnikom razmeroma majhne. Drugače je pri povprečnem času iskanja preslikave, saj semantični primerjalnik poveča povprečni čas, kar še posebej velja za semantično primerjanje s podobnostjo kot priklic prevoda. Kot učinkovit se je izkazal preslikovalnik z odločitvenim drevesom, požrešnim preiskovanjem in brez semantične primerjave.

6.1 Ocena razvitih metod

Preslikovalnik brez semantičnega primerjalnika deluje dobro in je primeren za uporabo pri razvrščanju podnapisov v skupine sorodnih podnapisov. To velja predvsem zato, ker so si sorodni podnapisi že tako podobni, da spadajo v enostaven ali lahek zahtevnostni razred, kjer ima preslikovalnik dobre rezultate.

Podobno velja tudi za gradnjo poravnanih korpusov, ker lahko z določanjem kriterija standardnega odklona ter pokritosti izberemo le najboljše preslikave. Potrebna je dobra poravnava stavkov, za kar že obstajajo algoritmi [8, 11].

Razviti preslikovalniki so slabi za prilagajanje podnapisov, saj imajo te-

žave pri iskanju preslikave med podnapisi, ki imajo manjše vsebinske razlike. Tak primer je iskanje preslikave med podnapisom prilagojenim gluhim in navadnim podnapisom.

Preslikovalniki s semantičnim primerjalnikom niso izpolnili pričakovanj. Sicer izboljšajo delovanje celotnega preslikovalnika, a je izboljšava pri metodi primerjanja ločil zanemarljiva, medtem ko se pri metodi podobnosti kot priklic prevoda še dodatno upočasni iskanje preslikave. Pri uporabi semantične primerjave sem pričakoval boljše rezultate.

Problemi so že v sami implementaciji semantičnega primerjalnika. Težava primerjanja ločil je predvsem v premajhnem številu različnih pomenskih kategorij. Z uporabo kakšnega naprednega sistema za klasifikacijo vrst stavkov [23] bi ta metoda lahko bila boljša.

Glavni problem semantične metode s podobnostjo kot priklic prevoda je v enosmernem ocenjevanju, ki jo prinaša metrika priklica prevoda. Zaradi večjega okna na ponorni strani nisem mogel uporabiti F ocene, kot je uporabljena v članku [15]. Ta omejitev pomeni, da se večjemu naboru stanj poda podobno ali celo enako ceno in tako kvečjemu razširi preiskani prostor. Z bolj dodelanim ali celo drugačnim algoritmom za ocenjevanje podobnosti vsebine, ki bi preverila vsebinsko podobnost v obe smeri, bi lahko bilo delovanje preslikovalnika boljše.

6.2 Nadaljnje delo

Preslikovalnik se lahko izboljša z bolj dovršeno cenovno funkcijo. Trenutna cenovna funkcija je povprečje različnih mer podobnosti. Boljša možnost bi lahko bila obteženo povprečje, a je potrebno poiskati prave obtežitve mer podobnosti.

Oba semantična primerjalnika imata še veliko možnih izboljšav. Pri podobnosti kot priklic prevoda je potrebno dopolniti primerjalni algoritem do te mere, da bo preverjal vsebinsko podobnost z obeh strani. Za to bo potrebno narediti za vsak par jezikov po dva dvo-jezična slovarja in pri primerjanju narediti prevod v eno in nato še v drugo smer. Za podobnost pa bi vzel povprečje obeh podobnosti.

Pri primerjavi ločil bi bilo bolje narediti primerjalnik stavkov in primerjati različne vrste stavkov s pomočjo klasifikatorja. Namesto vrste stavka bi se lahko primerjalo, kaj stavek opisuje, ali je to lastno, obče ime, kraj, čas in drugo [23].

Pri preiskovalnem algoritmu A* sem za končna stanja določil vsa stanja, ki nimajo naslednikov. Z dodatnim ocenjevanjem primernosti preslikave v trenutno končnem stanju bi lahko izboljšal rezultat preslikovalnika, seveda

pa se bi čas za iskanje preslikave povečal, vendar bi bila za primere uporabe, kjer je potrebno narediti manjše število preslikav in želimo čim boljši rezultat, taka izboljšava primerna.

Literatura

[1] Decision Trees - scikit-learn. Dostopno na http://scikit-learn.org/stable/modules/tree.html.

[2] Micro DVD Player. Dostopno na http://www.tiasoft.de/mdvdp/.

[3] Sub Station Alpha v4.00+ Script Format. Dostopno na http://moodub.free.fr/video/ass-specs.doc.

[4] .SRT [SubRip] file format specification needed, April 2004. Dostopno na http://forum.doom9.org/showthread.php?p=470941#post470941.

[5] Ripping subtitles from video files using SubRip, Junij 2005. Dostopno na http://zuggy.wz.cz/guides/video.htm.

[6] Formats - Podnapisi.NET, Junij 2012. Dostopno na https://www.podnapisi.net/wiki/wiki/Formats.

[7] Austin Carpenter. CUSVM: A CUDA implementation of Support Vector Classification and Regression. Januar 2009. Dostopno na http://patternsonascreen.net/cuSVMDesc.pdf.

[8] Kenneth Ward Church. Char_align: A Program for Aligning Parallel Texts at the Character Level. Objavljeno v Lenhart K. Schubert, editor, *ACL*, strani 1–8. ACL, 1993. Dostopno na http://dblp.uni-trier.de/db/conf/acl/acl93.html#Church93.

[9] Corinna Cortes in Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018. Dostopno na http://dx.doi.org/10.1007/BF00994018.

[10] Rina Dechter in Judea Pearl. Generalized best-first search strategies and the optimality of A*. *J. ACM*, 32(3):505–536, Julij 1985.

[11] William A. Gale in Kenneth W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 1993.

[12] Chih-Wei Hsu, Chih-Chung Chang, in Chih-Jen Lin. A Practical Guide to Support Vector Classification. Tehnično poročilo, Department of Computer Science, National Taiwan University, 2003. Dostopno na http://www.csie.ntu.edu.tw/~cjlin/papers.html.

[13] Einav Itamar in Alon Itai. Using Movie Subtitles for Creating a Large-Scale Bilingual Corpora. Objavljeno v *LREC*. European Language Resources Association, 2008. Dostopno na http://dblp.uni-trier.de/db/conf/lrec/lrec2008.html#ItamarI08.

[14] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[15] Caroline Lavecchia, Kamel Smaïli, in David Langlois. Building Parallel Corpora from Movies. Objavljeno v *The 4th International Workshop on Natural Language Processing and Cognitive Science - NLPCS 2007*, Funchal, Madeira, Portugal, Junij 2007. Dostopno na http://hal.inria.fr/inria-00155787.

[16] Giguet E Mangeot M. Multilingual Aligned Corpora From Movie Subtitles. Tehnično poročilo, Condillac-LISTIC, 2005. Dostopno na http://jibiki.univ-savoie.fr/~mangeot/Publications/Subtitles_MM-EG.pdf.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, in E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] J. Ramirez, J. M. Gorriz, in J. C. Segura. *Voice Activity Detection. Fundamentals and Speech Recognition System Robustness*. InTech, Junij 2007.

[19] Jörg Tiedemann. Automatical Lexicon Extraction from Aligned Bilingual Corpora. diplomska naloga, Otto-von-Guericke-Universität Magdeburg, 1991. Dostopno na http://ccl.pku.edu.cn/doubtfire/CorpusLinguistics/Bilingual_Corpus/Automatical_Lexicon_Extraction/Automatical%20Lexicon%20Extraction%20from%20Aligned%20Bilingual%20Corpora.htm.

[20] Jörg Tiedemann. Improved sentence alignment for movie subtitles. Objavljeno v *In Proceedings of RANLP, Borovets*, 2007.

[21] Jörg Tiedemann. Synchronizing Translated Movie Subtitles. Objavljeno v *LREC*. European Language Resources Association, 2008. Dostopno na http://dblp.uni-trier.de/db/conf/lrec/lrec2008.html#Tiedemann08.

[22] Andreas Tsiartas, Prasanta Kumar Ghosh, Panayiotis G. Georgiou, in Shrikanth S. Narayanan. Context-driven automatic bilingual movie subtitle alignment. Objavljeno v *INTERSPEECH*, strani 444–447. ISCA, 2009. Dostopno na http://dblp.uni-trier.de/db/conf/interspeech/interspeech2009.html#TsiartasGGN09.

[23] Menno Van Zaanen, Luiz Augusto Pizzato, in Diego Mollá. Classifying Sentences using Induced Structure. Objavljeno v *String Processing and Information Retrieval: 12th International Conference, SPIRE 2005*, strani 139–150. Springer-Verlag, 2005.