UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Simon Birk

PREPOZNAVANJE GRAFA IZ SLIKE

Diplomska naloga
na univerzitetnem študiju

Mentor: doc. dr. Bojan Kverh

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01721/2010
Datum: 03.12.2010

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

Kandidat: **SIMON BIRK**

Naslov: **PREPOZNAVANJE GRAFA IZ SLIKE**
GRAPH RECOGNITION FROM IMAGE

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Digitalizacija dokumentov je pomembna disciplina računalniškega vida. Sistemi za optično prepoznavo pisave so v uporabi že nekaj časa. Manj pozornosti pa je bilo namenjeno prepoznavanju različnih diagramov in skic. Razvijte sistem, ki bo sposoben iz slike prepoznati graf. Sistem naj bo sposoben razbrati oznake vozlišč, povezave med vozlišči in dolžine povezav. Omogoča naj tudi izvoz prepoznanega grafa v XML datoteko. Za posamezne dele sistema razvijte nove ali prilagodite obstoječe algoritme s področja računalniškega vida. Sistem testirajte na različnih na roko narisanih grafih in predstavite rezultate v analitični obliki.

Mentor:

doc. dr. Bojan Kverh

Kverh

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Zahvala

Zahvalil bi se svojemu mentorju, doc. dr. Bojanu Kverhu, za njegovo prijazno pomoč in nasvete pri izdelavi te diplomske naloge. Velika zahvala gre moji Martini, ki me je spodbujala, in mi svetovala. Zahvala gre tudi moji družini, ki mi je v času pisanja diplomske naloge nudila potrebno moralno podporo.

Kazalo

Seznam uporabljenih kratic in simbolov

OCR	Optical Character Recognition
ICR	Intelligent Character Recognition
OMR	Optical Mark Recognition
HT	Hough Transformation, Houghova transformacija
XML	eXtensible Markup Language – razširljiv označevalni jezik
blob	množica točk iste barve v sliki, ki so med seboj povezane 4 ali 8-povezljive
line thinning	postopek tanjšanja robov
threshold	pragovna vrednost pri binarizaciji slike
pixel	picture element - piksel, točka v sliki

Povzetek

Optična prepoznava dokumentov predstavlja pomemben del področja računalniškega vida. Dokument je običajno predstavljen kot množica slik, ki jo sistem za prepoznavo dokumentov ločeno obdela. Pri prepoznavi vsako sliko razdeli na posamezne elemente, glede na to kaj predstavljajo. Za vsak element je potreben drugačen pristop. Razvili smo aplikacijo, ki predstavlja enega takih pristopov. Aplikacija je sposobna iz slike, ki vsebuje graf, razbrati vozlišča in povezave. Vsakemu vozlišču določi lego, velikost in oznako, vsaki povezavi pa potek, usmerjenost in utež. Glede na smeri povezav se določi usmerjenost prepoznanega grafa. Končni rezultat aplikacije je struktura, shranjena v obliki XML (eXtensible Markup Language), ki je zaradi širše uporabe primerna za nadaljnjo obdelavo z različnimi algoritmi.

Ključne besede

optična prepoznava dokumentov, OCR, prepoznava grafov, topologija grafov

Abstract

Optical recognition of documents constitutes an important part of computer vision. A document is usually presented as a sequence of images. This sequence is then processed by a document recognition system. During recognition process each image is separated into various elements. Each type of element requires a different approach to recognition procedure. We have developed an application that represents one of such approaches. The application recognizes a graph containing nodes and edges from an image. For each node location, size and label are recognized. For each edge location, direction and weight are recognized. Given the directions of edges the orientation of identified graph can be determined. As the result of recognition a structure is stored in the form of XML (eXtensible Markup Language), which is widely used and is therefore suitable for further processing by various algorithms.

Keywords

optical document recognition, OCR, graph recognition, graph topology

1. Uvod

Optična prepoznava predstavlja zanimivo področje računalniškega vida. Obsega različne pristope in postopke, pri katerih iz slike prepoznavamo vsebino. Običajno gre za tiskano, tipkano ali ročno pisano besedilo, ki smo ga pridobili s pomočjo optičnega bralnika oz. skenerja. Z razvojem optične prepoznave so se razvili različni sistemi optične prepoznave, ki so zasnovani tako, da so sposobni obdelave velike množice dokumentov. Poljuben dokument je predstavljen kot sekvenca slik, ki jih sistem obdela v določenem vrstnem redu. Gre predvsem za postopek digitalizacije dokumentov in njihovo nadaljnjo uporabo. Poleg osnovnih tehnik prepoznave so uporabniki algoritme prilagodili svojim potrebam in zasnovali tudi svoje pristope in dopolnitve.

Pri obdelavi dokumenta se najprej srečamo s postopkom predobdelave. Najprej vhodno sliko pretvorimo v binarno. Nad dobljeno sliko izvedemo segmentacijo, s pomočjo katere določimo položaj in velikost posameznih elementov slike. Elementi lahko predstavljajo besedilo, fotografije, grafike, grafe, končne avtomate, ali pa na primer matematične grafe funkcij. Od tu dalje se izvaja optična prepoznava posameznih elementov. Ker so sistemi običajno usmerjeni bolj v prepoznavo besedila, se posamezni grafični elementi običajno le kopirajo v nov dokument. Vsak algoritem prepoznave besedila uporablja nek opis znakov, s katerim si pomaga pri prepoznavi. Podobno lahko za poljuben element sestavimo opis, ki ga nato algoritem za prepoznavo uporabi. Sistem optične prepoznave dokumentov, ki je zgrajen modularno, omogoča vgradnjo modulov za prepoznavo različnih tipov elementov. Tako imamo module za besedilo, tabele, graf, diagrame potekov, ki kot rezultat vrnejo besedilo, tabelo z vsebino, strukturo grafa ali diagrama poteka.

Razvili smo samostojno aplikacijo za optično prepoznavo grafov, ki jih srečujemo na področju teorije grafov. Graf vsebuje vozlišča in povezave med njimi. Vhodna slika lahko vsebuje s programom ali pa na roke narisan matematični graf. Pri risanju je potrebno paziti, da so velikosti vozlišč približno enake, da so vozlišča v obliki krožnice in da se oznake ter uteži ne dotikajo vozlišč in povezav. Kot rezultat postopka prepoznave aplikacija sestavi shemo in jo

shrani v XML datoteko. Poleg tega ustvari tudi novo sliko, na kateri prikaže prepoznano strukturo oz. graf.

V naslednjem poglavju je razložena domena grafov in postopki, ki se uporabljajo na področju sistemov za prepoznavo dokumentov. V poglavju 2.2 je opisano, kakšen vhod program za prepoznavo pričakuje in kakšne so njegove omejitve. Sledi poglavje, kjer so podrobno opisani osnovni postopki, filtri in transformacije, ki smo jih med postopkom prepoznave uporabili. V poglavju Prepoznava grafa je razložen podroben postopek, ki ga uporabljamo za prepoznavo. V naslednjem poglavju so podani primeri in rezultati uporabe aplikacije. Sledijo sklepne ugotovitve in razprava o dobljenih rezultatih. V dodatku je podan opis pomembnejših razredov, ki jih v aplikaciji uporabljamo.

2. Teoretično ozadje

2.1. Graf in teorija grafov

Graf je struktura v diskretni matematiki, s katero lahko ponazorimo omrežje (ceste, železnice, sistem kanalov, molekul, ...) [8]. Graf je sestavljen iz vozlišč ali točk (kraji, postaje, računalniki, atomi, ...) in povezav (ceste, žice, kanali, vezi ...), ki lahko nosijo različne lastnosti. Matematična disciplina, ki proučuje lastnost grafov se imenuje teorija grafov.

Definicija:

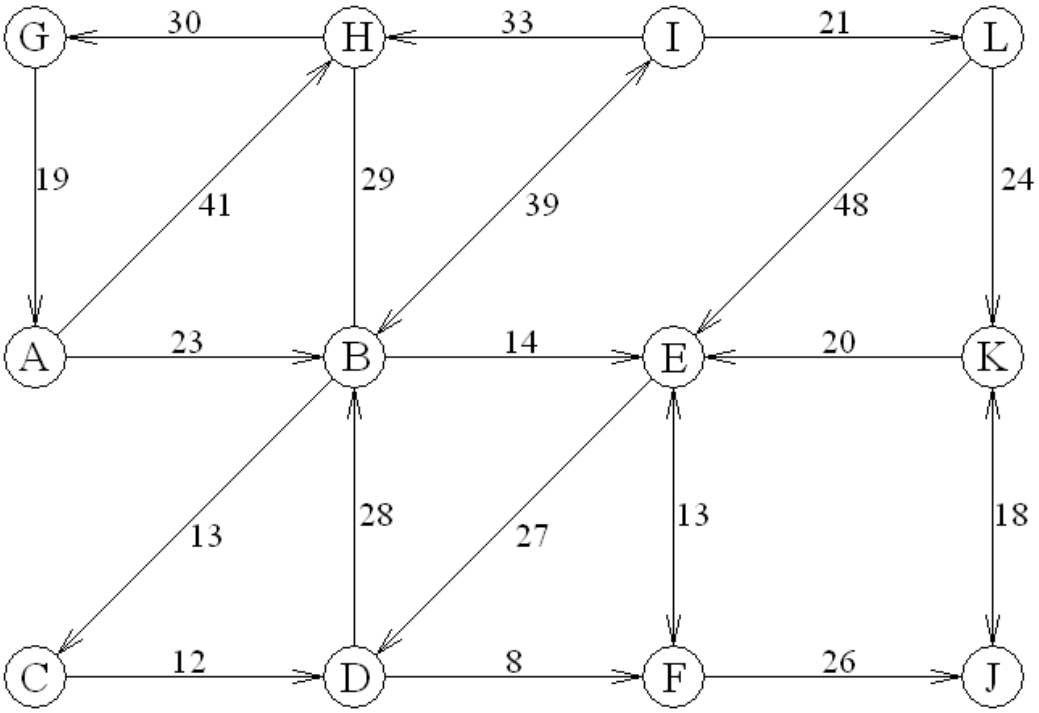
Graf $G = (V, E, i, r)$ je matematična struktura, pri kateri je

- V razred vozlišč,
- E razred povezav,
- $i : E \to V$, preslikava začetek, ki določi krajišče polpovezave,
- $r : E \to E$, involucija obrat brez negibnih točk, ki polpovezavi določi njeno nasprotno polpovezavo.

Involucija je preslikava, ki je sama sebi inverz. Negibna točka x preslikave $f : S \to S$ je element množice S, ki se preslika f nase: $x = f(x)$. Povezava je (neurejeni) par nasprotnih polpovezav $\{u, v\}$. Krajišči take povezave sta vozlišči $i(u)$ in $i(v)$. Če sta enaki, je povezava zanka. Za vsako vozlišče $v \in V$ je moč praslike $\left|I^{-1}(v)\right|$ valenca ali stopnja vozlišča v. Označujemo jo z $val(v)$ oz. $\deg(v)$ ali $d(v)$.

Graf ima lahko v splošnem zanke in vzporedne povezave (ali večkratne povezave). Graf brez zank in vzporednih povezav je enostaven. Predstavimo ga lahko z irefleksivno simetrično relacijo na vozliščih. Za nek graf G pravimo tudi, da je ravninski, če ga je moč narisati v ravnini tako, da se povezave ne sekajo, razen v krajiščih.

V delu obravnavamo prepoznavo ravninskih grafov s poljubno stopnjo vozlišč.



Slika 2.1: Primer računalniško skiciranega usmerjenega grafa.

2.2. Sistemi za digitalizacijo dokumentov

Algoritmi in metode, ki bi omogočile, da bi računalniški sistemi lahko "videli" in razpoznavali objekte tako kot človek, so še danes izziv mnogim raziskovalcem in znanstvenikom v različnih disciplinah [7]. Optično razpoznavanje znakov (v nadaljevanju OCR) je eno najbolj zanimivih področij tovrstnih raziskav.

OCR sistemi so zaradi avtomatičnega in hitrejšega zajemanja podatkov do danes povečali popularnost in uporabnost sistemov za upodabljanje dokumentov. Način uporabe je seveda odvisen od namena posameznih aplikacij. Današnji sistemi za upravljanje z dokumenti imajo navadno OCR že implementiran. Včasih pa se razvije potreba po lastni integraciji OCR funkcij, na primer v primerih, ko primarni OCR sistem ne podpira razpoznavanja znakov, ki so značilni za določeno jezikovno področje. V slovenskem jeziku so to na primer znaki č, ž in š.

2.2.1. OCR sistemi

Optično razpoznavanje znakov glede na vrsto zapisa delimo na:

- Razpoznavanje ene vrste pisave (*ang.* Fixed-font character recognition), ki je vnaprej znana. Tu prevladuje uporaba primerjalnih metod.

- Razpoznavanje poljubne pisave (*ang.* Omnifont character recognition), ki zajema razpoznavanje natipkanega teksta in je čim bolj neodvisno od oblike pisave. Omogočalo naj bi tudi interakcijo z uporabnikom, kar pomeni učenje znakov in stilov pisav. Sodobni sistemi nudijo tudi avtomatično odpravljanje nekaterih napak, seveda ob pomoči raznih slovarjev, s katerimi si pomagajo. Prevladujejo metode strukturne oziroma topološke analize.
- ICR *(ang*. Intelligent Character Recognition) – inteligentno razpoznavanje znakov, ki se navezuje na razpoznavanje ročne pisave (*ang.* handwriting recognition). Večinoma se uporabljajo metode na osnovi nevronskih mrež.
- OMR (*ang.* Optical Mark Recognition) – razpoznavanje označevanj na obrazcih, npr. anketah kjer je izhodni rezultat "označeno" ali pa "neoznačeno".
- razpoznavanje črtnih kod (*ang.* Barcode recognition)

OCR sisteme lahko delimo tudi glede na namen ali pa uporabljene metode. Poznamo dve veliki skupini metod za razpoznavanje: **matrično primerjalno** (*ang.* pattern matching, matrix matching, template matching) in **strukturno razvrščevalno metodo** (*ang.* structural classification, topological analysis). Znane so še metode na osnovi verjetnostne teorije in pa seveda metode na osnovi nevronskih mrež. Velikokrat se metode v sistemih med seboj prepletajo in tako dobimo hibridne strukture.

2.2.2. Osnovni koraki prepoznavanja dokumentov

2.2.2.1. Predobdelava slike

Pred samim procesom razpoznavanja je potrebno vhodni dokument primerno transformirati. Iz slike izločimo razne motnje (npr. posamezni izolirani slikovni elementi), ki so nastale ali zaradi nekvalitetnega tiska ali pa napak pri skeniranju. Nato se izvede izločitev nepotrebnih delov, kot so črte, okvirji, ozadja in podobno. Naslednja faza je analiza oziroma dekompozicija slike. Gre za razčlenitev vsebine slike na bloke, pri čemer je pomembna predvsem določitev tipa bloka: tekst, tabela, grafični element, črtna koda, itd. [2]. Tipi blokov so zelo pomembni za kasnejšo segmentacijo in razpoznavanje besedila in grafik.

2.2.2.2. Segmentacija

Segmentacija poteka v več korakih. Prvi korak je že v prejšnjem razdelku omenjena dekompozicija dokumenta. S segmentacijo razgradimo sliko posameznega bloka na posamezne znake, ki jih kasneje ločeno razpoznamo. Težave nastopijo predvsem v primeru

"zlepljenih" (*ang.* touching characters) ali prelomljenih znakov [2] ter delno prekrivajočih se območij znakov, kar se dogaja predvsem pri ročni pisavi. Dodatni nivoji in algoritmi segmentacije poskušajo te napake odpraviti, saj v nasprotnem primeru pride do hipersegmentacije (več znakov združenih v enega) ali podsegmentacije znakov.

Metode segmentacije so na primer izmenjevanje vertikalne in horizontalne projekcije, strategije pa se delijo v tri skupine: *top-down, bottom-up* in hibridne [1]. Običajno se ob segmentaciji izvajajo še procesi normalizacije znaka na standardno velikost in včasih tudi tanjšanje znakov na debelino enega slikovnega elementa (*ang.* thinning algorithm). Avtorji so pri opisovanju in delitvi procesov same predobdelave in segmentacije slike pogosto različnih mnenj.

2.2.2.3. Iskanje značilk znaka

V tem koraku znake podrobno definiramo in jim določimo lastnosti, oziroma značilke (*ang*. feature extraction) in jim priredimo vrednosti v obliki vektorja značilk, ki so odvisne od same metode razpoznavanja. Definicija značilk je bistvenega pomena za uspeh metode, poleg velike odvisnosti od vrste pisave oziroma znaka. Ti vektorji značilk so podlaga za naslednji korak, razvrščanje.

2.2.2.4. Razvrščanje

Razvrščanje je v večini izvedeno s klasifikacijo vrednosti značilk. Rezultat izpeljave primerjamo z referenčnimi vzorci posameznih razredov in vzorec razvrstimo v tisti razred, kamor spada referenčni vzorec z najmanjšo razdaljo, oziroma z največjo mero ujemanja. Končni rezultat razpoznavanja je oznaka razreda (znak), v katerega je vzorec razvrščen.

2.2.2.5. Končna obdelava

Poleg človekovega preverjanja rezultata razpoznavanja, sistemi vsebujejo različne metode avtomatičnega popravljanja oz. verifikacije rezultata. Prva možnost je preverjanje obstoja besede v slovarju (*ang*. spell-checking). Verifikacija lahko poteka na osnovi sistema pravil, ki definirajo bolj in manj verjetna zaporedja znakov in ostale lastnosti jezika. Tovrstne tehnike se seveda uporabljajo le v primeru nezanesljivo razpoznanega znaka. Uporabno je na primer barvno označevanje "sumljivo" razpoznanih znakov.

Ob koncu obdelave se rezultati še primerno oblikujejo in shranijo.

3. Uporabljene tehnike in filtri

3.1. Pretvorba v sivinsko in binarno sliko

Sivinska slika (*ang*. grayscale image) predstavlja digitalno sliko, kjer vsaka točka oz. piksel (*ang*. pixel) nosi le informacijo o intenziteti [11]. Sliko sestavljajo odtenki sive barve, kjer je najmanj intenziven odtenek črne, najbolj intenziven pa odtenek bele barve (Slika 3.1). Pretvorba barvne v sivinsko sliko poteka tako, da se za vsako točko $t(r,g,b)$ v sliki izračuna intenziteta $I(t)$ kot obtežena vsota

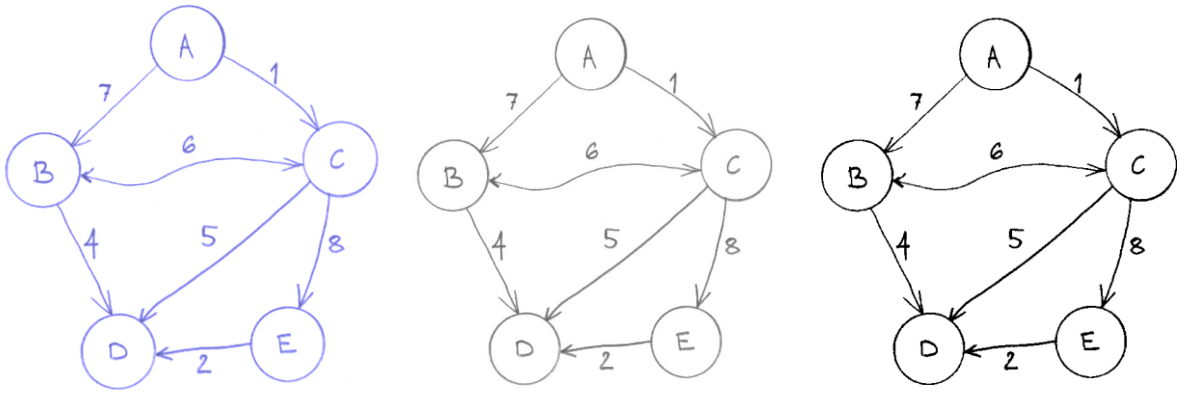
$$I(t) = c_r \cdot red(t) + c_g \cdot green(t) + c_b \cdot blue(t), \tag{1}$$

kjer so

$$c_r = 0.2125,\ c_g = 0.7154,\ c_b = 0.0721 \tag{2}$$

uteženi koeficienti z vrednostmi med 0 in 1.

Binarna slika (*ang*. bilevel image, binary image) je digitalna slika, kjer lahko vsaka točka zavzame le dve različni vrednosti [12]. Običajno za prikaz uporabljamo črno in belo barvo. Postopek binarizacije (*ang.* binarization) sivinsko sliko pretvori v binarno. Pogosto se uporablja pri pripravi vhoda v OCR [9], ki vhod običajno zahtevajo binarno oz. črno-belo sliko.

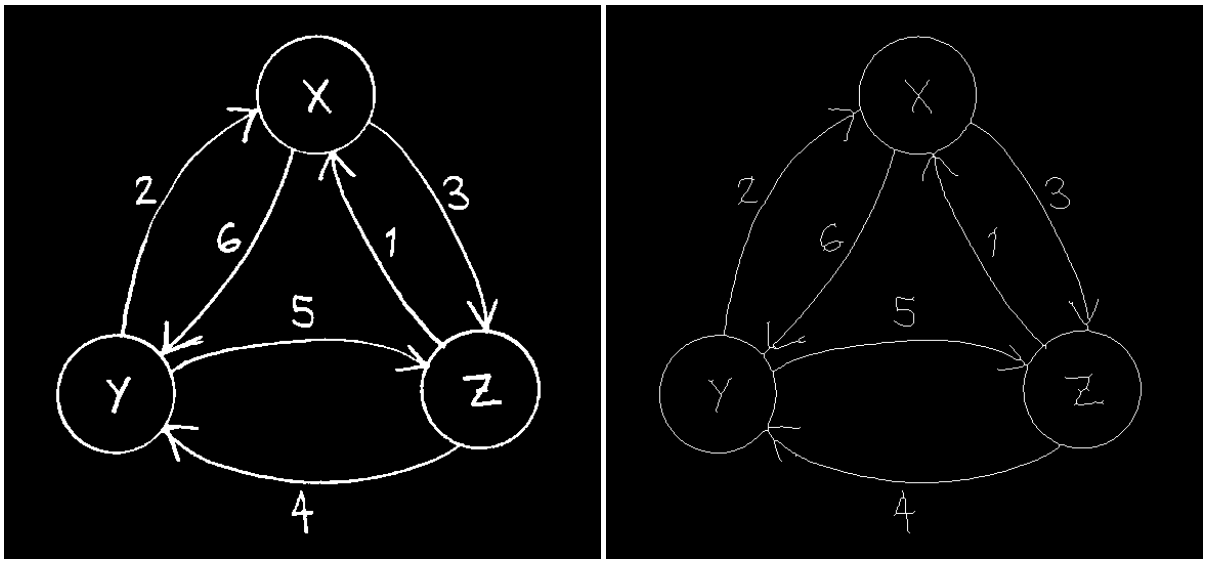


Slika 3.1: Pretvorba barvne slike v sivinsko in nato v binarno sliko.

Najenostavnejši postopek binarizacije je uporaba upragovanja (*ang*. thresholding) oz. pragovne vrednosti. Vrednost točk določimo tako, da točke z večjo vrednostjo od pragu označimo kot bele, ostale pa kot črne oz. ozadje. V večini primerov je binarizacija težko izvedljiva samo z enim pragom. V takih primerih si pomagamo z naprednejšimi postopki, s katerimi različna področja slik obravnavamo ločeno. Imeli smo slike z malo šuma, zato napredne tehnike binarizacije niso bile potrebne. Uporabili smo razred `SISThreshold`, ki na osnovi statistike slike določi pragovno vrednost [5].

3.2. Tanjšanje robov

Postopek tanjšanja robov (*ang.* line thinning) se uporablja na črno-belih slikah. Iz vhodne slike tako postopoma dobimo novo sliko, ki vsebuje stanjšane robove (Slika 3.2).

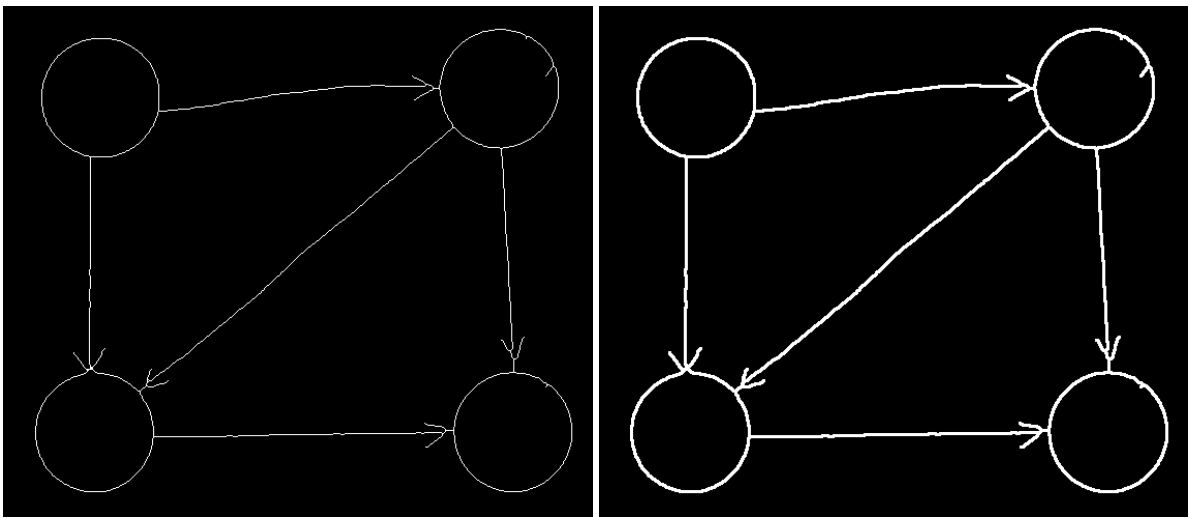


Slika 3.2: Slika pred (levo) in po (desno) postopku tanjšanja robov.

Za postopek tanjšanja robov smo uporabili razred `FiltersSequence`, ki omogoča definiranje in izvajanje zaporedja več filtrov nad vhodom. Izhod filtra v sekvenci je vhod naslednjemu filtru v sekvenci. Definirali smo 8 različnih filtrov dimenzije 3x3 in jih dodali v sekvenco in jo večkrat izvedli. Uporabili smo filter `HitAndMiss`, ki omogoča način tanjšanja robov. Sekvenco smo ponovili tolikokrat, da se je vhodna slika trenutne iteracije popolnoma ujela z izhodno sliko prejšnje iteracije. V povprečju smo potrebovali 5 – 8 ponovitev sekvence. Največje število ponovitev sekvence smo omejili na 20, delno zaradi časovne kompleksnosti predvsem pa zaradi sklepa, da tako debelih črt v slikah ne bo.

3.3. Debeljenje robov

V postopku debeljenja robov (*ang.* dilation) vsaki točki v vhodni sliki pripišemo maksimalno vrednost njenih sosedov. Če imamo npr. bel graf na črnem ozadju, s tem postopkom odebelimo povezave in vozlišča (Slika 3.3).



Slika 3.3: Slika pred (levo) in po (desno) postopku debeljenja robov.

V postopku prepoznavanja grafa smo za debeljenje uporabili filter `Binary-Dilatation3x3`, ki implementira postopek dilatacije oz. v našem primeru debeljenja robov nad binarno sliko.

3.4. Označevanje povezanih delov slike

Označevanje povezanih delov (*ang.* blob) je postopek, ki združuje točke na podlagi funkcije povezljivosti (*ang.* pixel connectivity) [6]. Na primer, vse točke v povezanem delu slike si delijo neko lastnost. To je lahko podobna intenziteta točk (pri sivinskih slikah), ali pa enaka barva točk (pri binarnih slikah).

Funkcija povezljivosti opisuje relacijo med dvema ali več točkami. Da točke povežemo, morajo biti izpolnjeni določeni pogoji. Da funkcijo povezljivosti formuliramo, moramo vpeljati *notacijo sosednosti*. Za točko $p(x, y)$ je množica točk N_4 podana z opisom

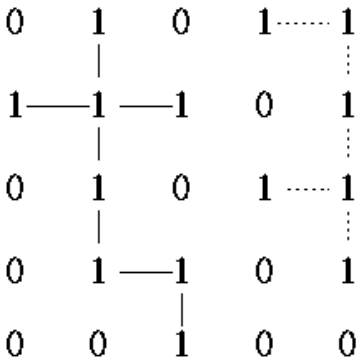
$$N_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\} \quad (3)$$

in se imenuje njena 4-sosednost (*ang.* 4-neighbors). Podobno definiramo 8-sosednost (*ang.* 8-neighbors)

$$N_8(p) = N_4 \bigcup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}. \quad (4)$$

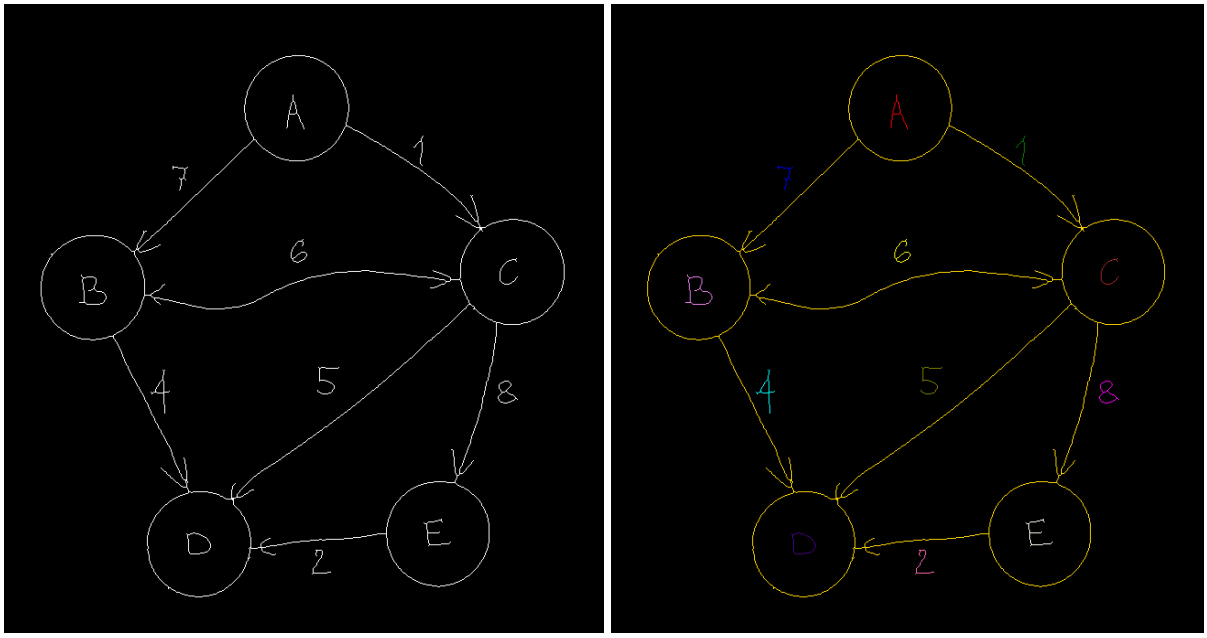
Iz tega potem definiramo 4- in 8-povezljivost (*ang*. connectivity):

Za neki točki p in q v sliki velja, da sta 4-povezana, če je q v množici $N_4(p)$ in 8-povezana, če je q v $N_8(p)$.



Slika 3.4 Dva povezana dela na osnovi 4 – povezljivosti.

Primer označevanja povezanih delov slike prikazuje Slika 3.5, kjer je vsak povezan del slike označen z drugačno barvo.

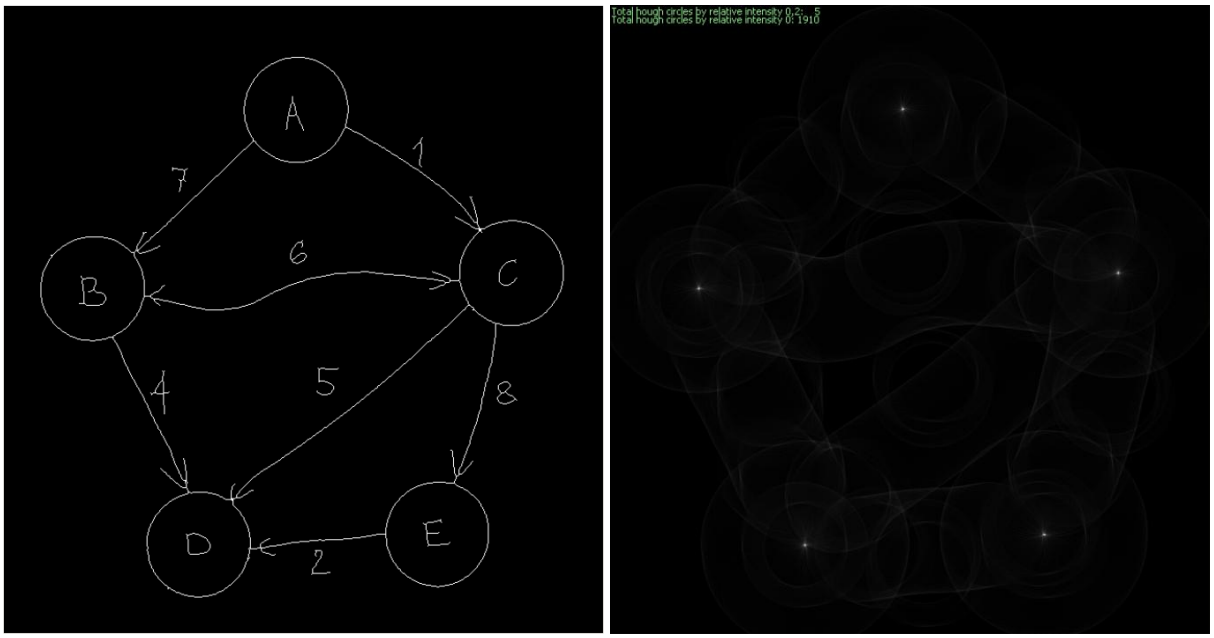


Slika 3.5: Slika pred (levo) in po (desno) postopku označevanja povezanih delov

Za postopek ekstrakcije smo uporabili razred `BlobCounter`, ki vsebuje vse potrebno za iskanje lastnosti delov slike, kot so na primer položaj, velikost in polnost. Ekstrakcija in označevanje različnih povezanih in nepovezanih delov oz. komponent je eden osnovnih postopkov v večini aplikacij, ki se uporabljajo za (avtomatsko) analizo slik [4].

3.5. Houghova transformacija

Houghova transformacija (*ang.* Hough transformation) oz. HT se širše uporablja za iskanje premic in vseh krivulj, ki jih je moč parametrizirati. Ideja transformacije je preslikava težkega problema iskanja vzorcev v enostaven problem iskanja vrhov v prostoru parametrov krivulje [3]. Kot vhod se zahteva slika z najdenimi robovi (Slika 3.6).



Slika 3.6: Vhodna slika (levo) in rezultat Houghove transformacije (desno), kjer bele pike predstavljajo središča vozlišč.

Uporabili smo transformacijo, prilagojeno za iskanje krožnic. Implementirana je v razredu `HoughCircleTransformation`, ki ga uporabljamo za potrditev posameznih kandidatov za vozlišča.

3.6. Detekcija robov

Robne točke oz. robovi so točke, okoli katerih se vrednosti v sliki nenadno spremenijo [3]. S pojmom roba opišemo povezane verige točk oz. fragmente v sliki.

Uporabili smo Sobelov detektor, čigar jedro predstavlja dvoje separabilnih jeder:

$$S_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ in } S_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{5}$$

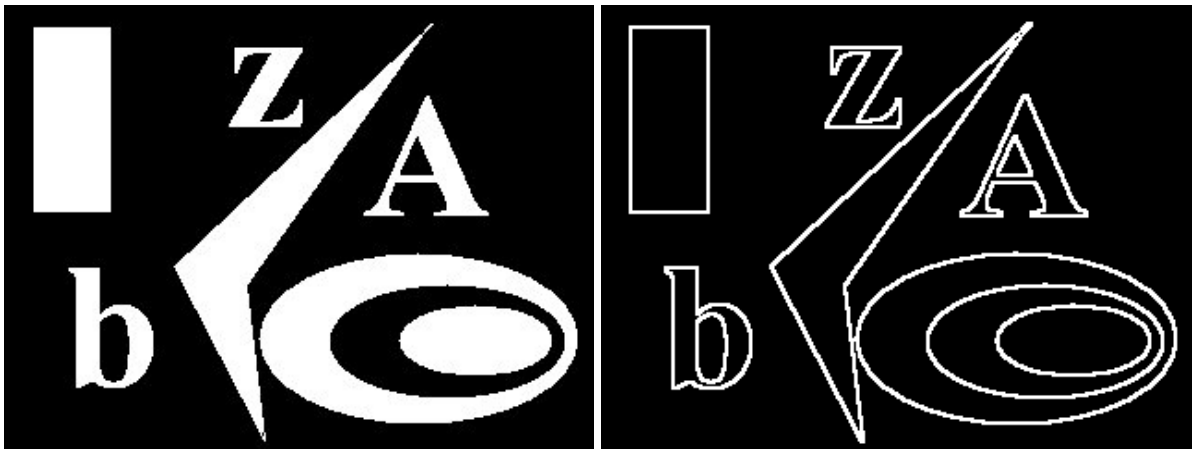
ki jih konvoluiramo z vhodno sliko I, in dobimo

$$G_x = S_x * I \text{ in } G_y = S_y * I . \quad (6)$$

Moč roba v točki (x, y) je

$$G(x, y) = \sqrt{G_x^2(x, y) + G_y^2(x, y)}. \quad (7)$$

Vse točke, za katere velja $G(x, y) \geq \tau$, označimo kot robove (Slika 3.7).



Slika 3.7 Pred (levo) in po (desno) postopku detekcije robov.

Detektor robov je zasnovan v razredu `SobelEdgeDetector`. Uporabljamo ga pri pripravi posameznih kandidatov, preden jih preverimo, če dejansko predstavljajo vozlišča.

3.7. Optična prepoznava besedila

Za postopek prepoznave oznak vozlišč in uteži povezav smo uporabili OCR knjižnico `tessnet2`. Je odprtokodna knjižnica, ki podpira optično prepoznavo znakov. Prirejena je za uporabo v .NET okolju in uporablja t.i. "*Tesseract engine*", ki je knjižnica, napisana v jeziku C++. Prvotno jo je razvilo podjetje HP, danes pa jo razvija Google pod licenco Apache License. Knjižnica je brezplačna in njena uporaba enostavna. Uporabimo lahko več jezikov, imamo pa tudi način delovanja "*numeric mode*", ki omogoča iskanje števil.

Kot vhod algoritem OCR potrebuje binarno sliko, vrne pa množico prepoznanih znakov, ki jih moramo potem sami združiti v besedilo. Knjižnico uporabljamo za prepoznavo oznak vozlišč (prepoznava besedila) in vrednosti uteži (prepoznava števil) povezav.

4. Prepoznava grafa

4.1. Omejitve

Aplikacija je usmerjena predvsem obdelavi prostoročno narisanih grafov, uspešno pa prepozna tudi računalniško skicirane grafe. Postopek prepoznavanja pravilno deluje, če se držimo določenih pravil. V ta namen smo definirali omejitve, ki temelje na osnovnih lastnostih grafov.

Omejitve za vozlišča:

- Oznaka znotraj vozlišča se ne sme dotikati oboda vozlišča.
- Obod vozlišča mora imeti obliko krožnice. Največje dovoljeno odstopanje med višino in širino vozlišč je 10 %.
- Obod vozlišča ne sme biti prekinjen.

Omejitve za povezave:

- Povezave ne smejo biti prekinjene in se ne smejo sekati med seboj.
- Utež se povezave ne sme dotikati.
- Povezava mora biti daljša od dveh premerov vozlišča.
- Višina znakov posamezne uteži naj bo večja od ¼ velikosti vozlišča in hkrati manjša od dvojne velikosti vozlišča.

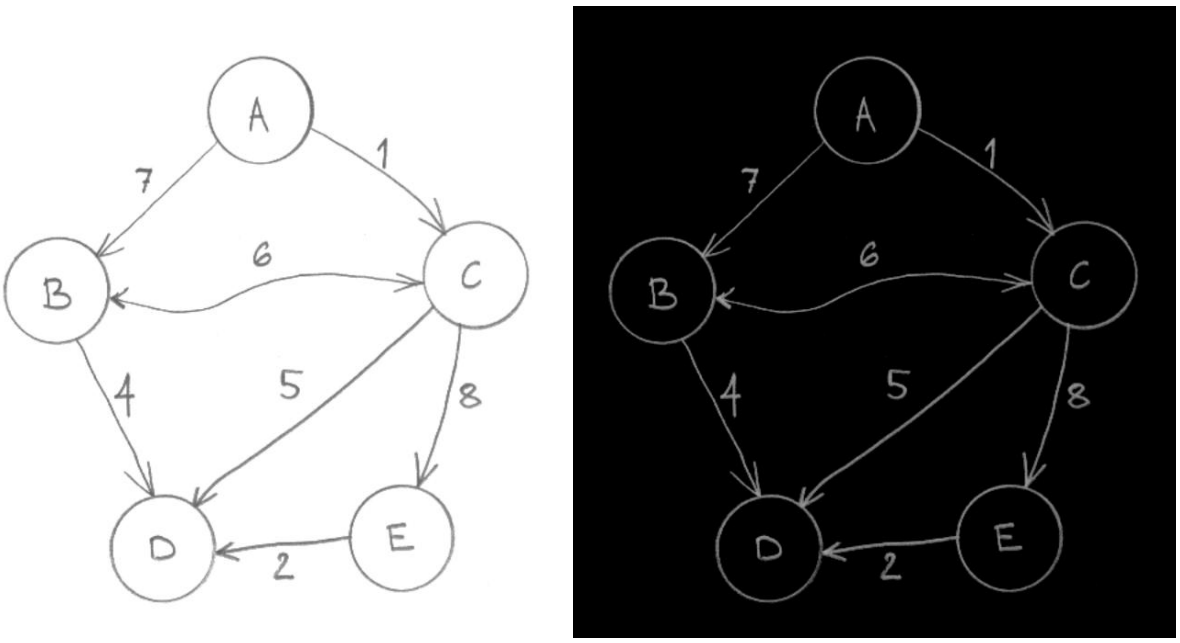
Poleg omejitev pa je priporočljivo upoštevati naslednje nasvete:

- Utež naj bo blizu povezave, h kateri pripada.
- Usmerjena krajišča povezav naj bodo narisana jasno.
- Obodi vozlišč in krivulje povezav naj bodo odebeljeni.

4.2. Priprava vhodne slike

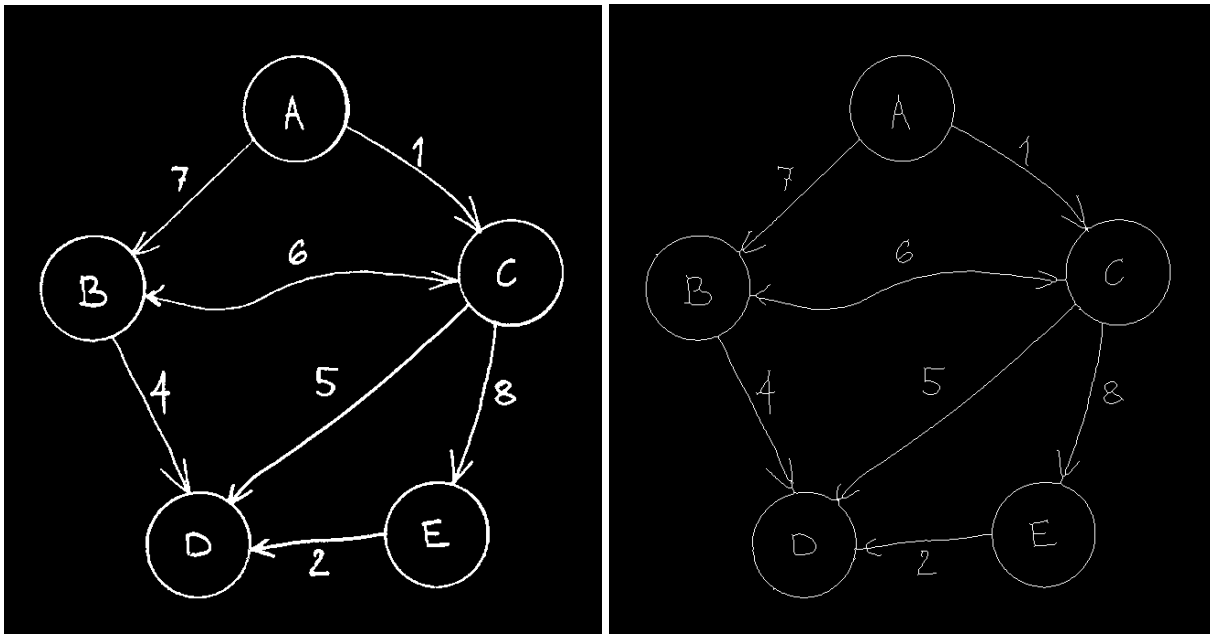
Za postopek prepoznave je potrebno pripraviti binarno in sivinsko različico vhodne slike. Če je vhodna slika že v sivinah, pripravimo samo binarno. Za pretvorbo v sivinsko sliko uporabljamo razred `Grayscale`, pretvorbo v binarno sliko pa izvedemo z razredom `SISThreshold` (Poglavje 3.1).

Za delovanje algoritma prepoznave vozlišč potrebujemo binarno sliko, kjer je graf v ospredju, ozadje pa je črno. V primeru, ko je graf v ozadju, je potrebno izračunati njegov negativ. Barvo ozadja se ugotovi tako, da se iz slike naključno izbere 100 točk in se jih po barvi klasificira med bele in črne. V primeru, ko je več točk belih, sklepamo, da gre za belo ozadje in vhodni sliki priredimo njen negativ (Slika 4.1).



Slika 4.1: Vhodna slika za prepoznavo, kjer je graf predstavljen s črno barvo kot ozadje (levo) in negativ slike, kjer je graf bele barve (desno).

Binarno sliko je potrebno normalizirati oz. pretvoriti v neko začetno obliko. S tem rešimo tudi problem, ko se debeline črt v grafu razlikujejo. Začetna oblika se pridobi z večkratno ponovitvijo postopka tanjšanja robov (Poglavje 3.2). Dobimo sliko z debelinami črt enega slikovnega elementa (Slika 4.2).



Slika 4.2: Graf pred (levo) in po (desno) postopku tanjšanja robov.

4.3. Prepoznava vozlišč

4.3.1. Položaj

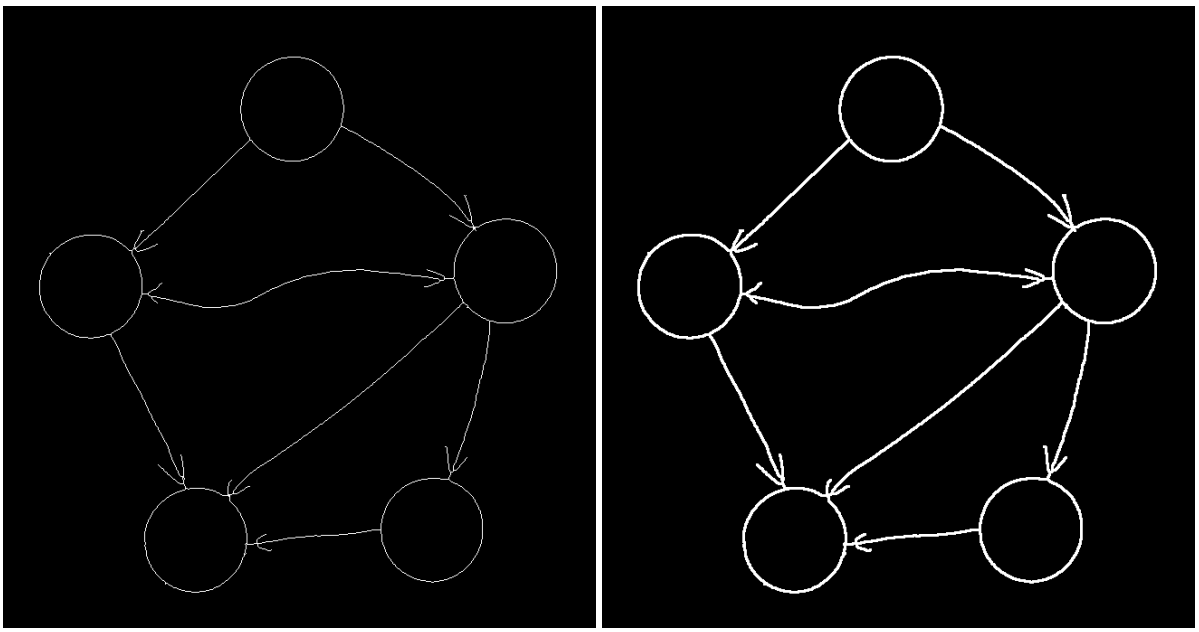
Za iskanje položaja in velikosti vozlišč najprej izvedemo postopek, s katerim iz slike izločimo skelet grafa. Če skeleta ni mogoče določiti, se prepoznava grafa ne more nadaljevati.

Zasnovali smo naslednji postopek:

1. S pomočjo razreda `BlobCounter` (Poglavje 3.4) ustvari seznam vseh povezanih elementov (v nadaljevanju blobov). Ti elementi so:
    - znak – del oznake vozlišča,
    - število – del uteži povezave,
    - povezan skelet z obodi vozlišč in povezavami med njimi,
    - razne pike in robovi različnih velikosti, ki predstavljajo šum.
2. Iz seznama izbere tiste blobe, ki so po višini večji od polovice višine slike in po širini širši od polovice širine slike.
3. Izmed blobov izbere najmanjšega po velikosti, ker le ta predstavlja skelet grafa (Slika 4.3, levo).

S prvim korakom dobimo kandidate za vozlišča. V drugem ohranimo samo dovolj velike. Ker je v sliki prisoten šum, se lahko zgodi, da blob z največjo površino ne predstavlja skeleta

grafa, zato v tretjem koraku izberemo najmanjšega iz drugega koraka. Kot šum se pogosto pojavijo odtenki robov skeniranih listov, ki so po velikosti lahko večji od skeleta samega grafa. Rezultat postopka je tako skelet grafa (Slika 4.3, levo).



Slika 4.3: Skelet grafa pred (levo) in po (desno) debeljenju robov.

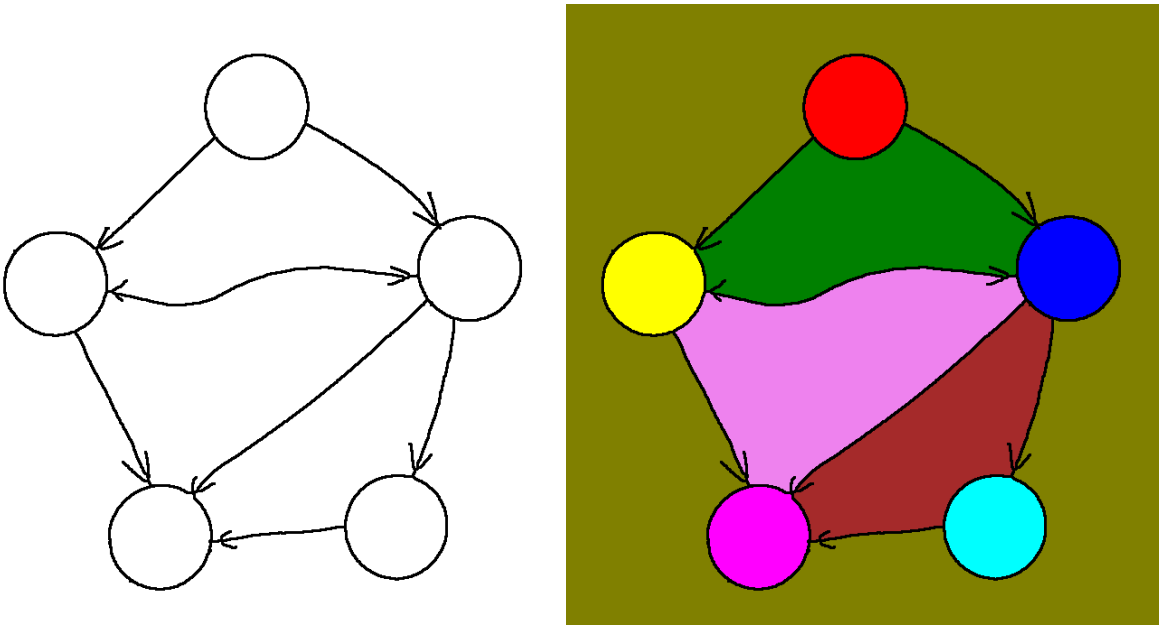
Da določimo točno lego vozlišč, potrebujemo položaj notranjih delov vozlišč.

Nadaljujemo s sledečim postopkom:

1. S postopkom dilatacije (Poglavje 3.3) v sliki odebeli vse črte (Slika 4.3, desno).
2. Sliko pretvori v njen negativ (Slika 4.4, levo).
3. S pomočjo razreda `BlobCounter` ustvari seznam vseh blobov (Slika 4.4, desno)

Prvi korak je potreben, ker se pri ekstrakciji blobov uporablja povezovanje točk z uporabo 8-povezljivosti (Poglavje 3.4). Če robovi ne bi bili odebeljeni, bi bil rezultat ekstrakcije en sam blob v velikosti slike. Drugi korak spremeni pomembna polja v belo barvo, torej ospredje. Tretji korak ustvari seznam kandidatov za vozlišča, ki jih je potrebno preveriti. Ti so:

- notranjost vozlišč,
- notranja lica oz. področja med vozlišči,
- zunanje lice oz. področje izven grafa.



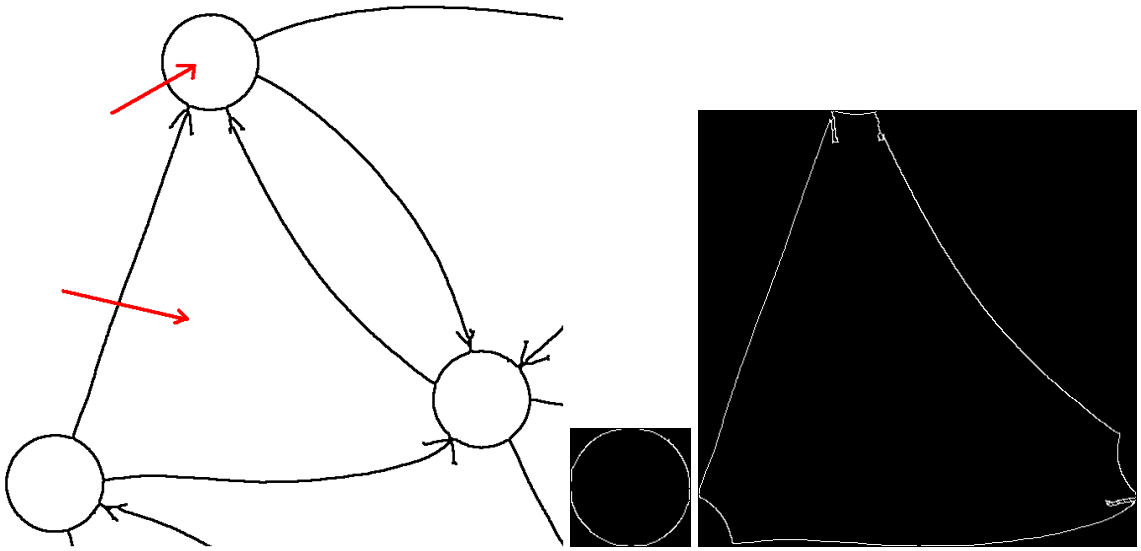
Slika 4.4: Negativ grafa iz Slika 4.3 in predstavitev barvanja blobov (za lepšo ponazoritev).

Nad množico blobov izvedemo naslednji postopek:

1. Odstrani vse blobe, ki so po $širina(b) < 10 točk$ ali $višina(b) < 10 točk$.
2. Odstrani vse blobe, za katere velja $širina(b) > \frac{širina(I)}{2}$ in $višina(b) > \frac{višina(I)}{2}$, kjer je I vhodna slika.
3. Za preostale blobe b preveri pogoja:
   a. če $širina(b) > višina(b)$, potem mora veljati $\frac{širina(b)}{višina(b)} < 1{,}1$
   b. če $višina(b) > širina(b)$, potem mora veljati $\frac{višina(b)}{širina(b)} < 1{,}1$

   in odstrani vse blobe, za katere to ne velja.

Prvi korak odpravi šum, ki ga predstavljajo majhne pike, drugi korak pa odstrani velika lica in zunanje lice grafa. Tretji korak zahteva podobnost bloba dimenzijam kvadrata, kjer dopuščamo 10 odstotno odstopanje. Dovoljeno vrednost odstopanja smo določili s testiranjem. Po tem postopku imamo torej samo tiste blobe, ki zadoščajo zgornjim pogojem in so torej podobni kvadratu (Slika 4.5).



Slika 4.5: Primer skeleta in dveh kandidatov za vozlišče, ki izpoljujeta pogoje in sta po dimenzijah podobna kvadratu.

Zgoraj sta navedena dva kandidata za vozlišče na črni podlagi. Levi kandidat predstavlja primer vozlišča, ki ga iščemo, desni pa notranje lice, ki je po naključju takih dimenzij in ga moramo izločiti. Med temi kandidati ločimo tako, da uporabimo detekcijo robov in Houghovo transformacijo.

Nad vsakim kandidatom se izvede postopek, ki:

1. z uporabo Sobelovega detektorja poišče robove nad sliko kandidata (Poglavje 3.6),
2. izvede enkratno tanjšanje robov,
3. nad sliko I iz 2. točke izvede Houghovo transformacijo za krožnice (Poglavje 3.5) s polmerom $r = \lfloor (širina(I) + višina(I)) / 4 \rfloor$,
4. izmed najdenih krožnic odstrani tiste, ki imajo relativno intenziteto pod 0,75,
5. če med preostalimi krožnicami velja, da je intenziteta zadnje krožnice enaka 1, potem kandidata oz. blob doda v končno množico kandidatov za vozlišča.

S Sobelovim detektorjem in tanjšanjem robov naredimo sliko primernejšo za Houghovo transformacijo. S testiranjem smo ugotovili, da je v primeru, ko je šlo za vozlišče, obstajala samo ena oz. le nekaj krožnic v sliki bloba, ki so imele enako relativno intenziteto oz. število glasov (*ang*. votes), zato zadnja točka oz. pogoj učinkovito ločuje med pravimi in napačnimi kandidati za vozlišča. V primeru, ko ni šlo za vozlišče, se je pri iskanju krožnic vedno razvrstilo več krožnic različnih intezitet.

Končno množico kandidatov uporabimo za kreiranje vozlišč. Vozlišča predstavimo z razredom `Domain.NodeCandidate` (Dodatek). Položaj vozlišča določimo iz lege

kandidata, velikost vozlišča pa iz njegove velikosti. Tako je množica vozlišč pripravljena za algoritem, ki izvede prepoznavo oznak.

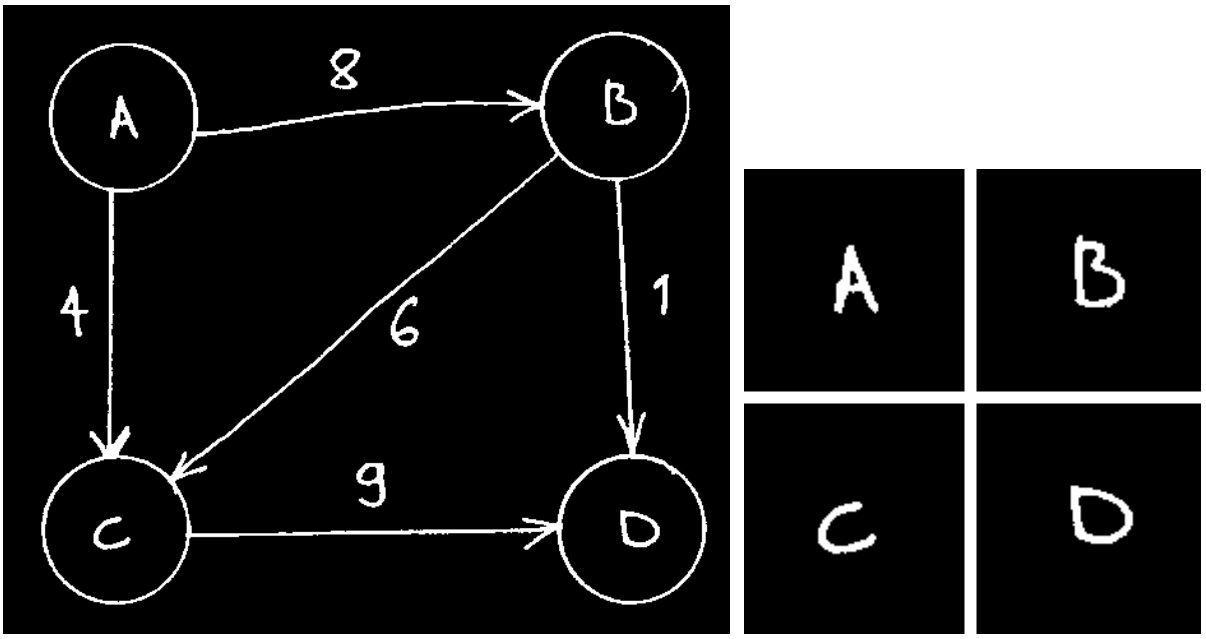
4.3.2. Oznaka

Osnovna lastnost oznak je, da se nahajajo v notranjosti vozlišč. Sestavlja jih poljubno zaporedje črk. Ker poznamo vse položaje vozlišč, je iskanje oznak enostavno. Uporabimo binarno sliko grafa in seznam vozlišč.

Postopek prepoznave oznak vozlišč:

1. s pomočjo razreda `BlobCounter` (Poglavje 3.4) iz vhodne binarne slike ustvari seznam vseh blobov,
2. za posamezno vozlišče:
   2.1. izbere blobe, ki ležijo znotraj vozlišča,
   2.2. združi slike izbranih blobov med seboj v sliko oznake (Slika 4.6),
3. za posamezno vozlišče:
   3.1. izvedi prepoznavo besedila iz slike oznake (Poglavje 3.7),
   3.2. prepoznano besedilo shrani v oznako vozlišča.

Pogosto se zaradi občutljivosti prepoznave znakov zgodi, da prepoznano besedila vsebuje tudi znake, ki niso črke (na primer ';'. '/' ipd.). Odločili smo se, da uporabimo vse znake, ki jih je algoritem prepoznal in jih pripišemo vozlišču.



Slika 4.6: Vhodni graf in dobljene slike oznak vozlišč, ki se uporabijo za prepoznavo.

4.4. Prepoznava povezav

4.4.1. Položaj

Vsaka povezava se začne in konča v nekem vozlišču. Ker smo vozlišča že določili, je potrebno preveriti oz. najti vse povezave, ki med temi vozlišči obstajajo. Za prepoznavanje potrebujemo sliko skeleta grafa, saj poleg vozlišč vsebuje tudi vse povezave.

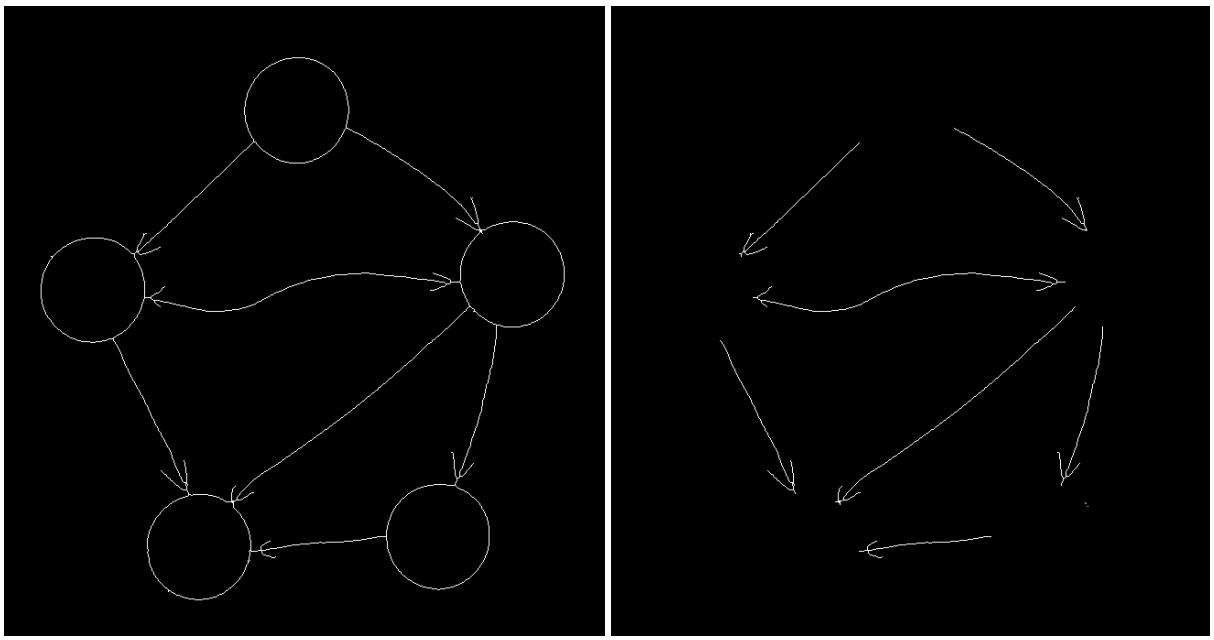
Najprej smo razvijali postopek, pri katerem smo neko vozlišče vzeli kot izhodišče. Iz vozlišča smo iskali poti, ki vodijo do drugih vozlišč. Ker se je izkazalo, da je iskanje začetka povezav zelo neroden postopek, smo uporabili drugačen pristop.

Ugotovili smo, da si lahko pomagamo tako, da vozlišča s prerisovanjem odstranimo iz skeleta (Slika 4.7). Tako ostanejo povezave, ki jih moramo ločiti od šuma in jim določiti natančno lego, krajišča usmerjenost in uteži.

Postopek prepoznave:

1. vsa vozlišča preriše s polnimi krožnicami črne barve (Slika 4.7),
2. iz slike brez vozlišč s pomočjo razreda `BlobCounter` (Poglavje 3.4) ustvari seznam blobov.

Seznam blobov predstavlja vse povezave in nekaj ostankov povezav in obodov vozlišč, ki smo jih pridelali z brisanjem vozlišč. Blob omejuje potek povezave iz vseh 4-ih strani, povezava pa se dotika vsakega izmed njih.



Slika 4.7: Vhodna slika skeleta pred in po brisanju vozlišč.

Za vsak blob iz seznama izvedemo naslednji potopek prečnega prereza (Slika 4.8), ki:

1. preveri, če velja $širina(b) > višina(b)$:

   1.1. določi izhodiščne točke: $T_0(širina(b)/2, 0)$, $T_1((širina(b)/2)-1, 0)$ in $T_2((širina(b)/2)+1, 0)$,

   1.2. za vsako od izhodiščnih točk T_0, T_1, T_2 preveri vrednosti točk, kjer je $y \in 0, višina(b)-1$ in vzame prvi y, za katerega je točka bela

2. preveri, če velja $širina(b) \leq višina(b)$:

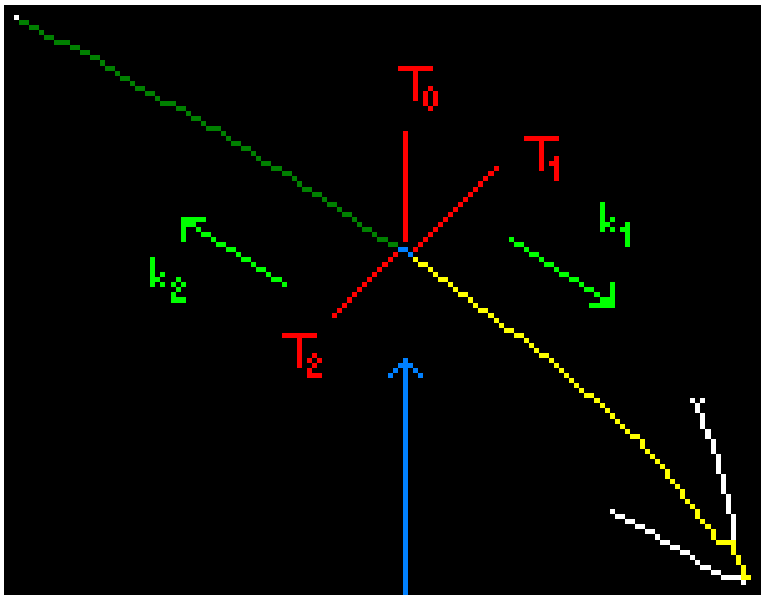
   2.1. določi izhodiščne točke $T_0(0, višina(b)/2)$, $T_1(0, (višina(b)/2)-1)$ in $T_2(0, (višina(b)/2)+1)$

   2.2. za vsako od izhodiščnih točk T_0, T_1, T_2 preveri vrednosti točk, kjer je $x \in 0, širina(b)-1$ in vzame prvi x, za katerega je točka bela

3. dobljene točke T_0', T_1', T_2' iz koraka 1.2 ali 2.2 shrani

4. iz točk T_0' in T_1' izračuna smerni koeficient k_1, iz T_0' in T_2' pa smerni koeficient k_2

Izvedemo torej prerez bloba po dolžini ali širini in vzamemo prvo točko, ki je bele barve. Poleg te točke poiščemo tudi njenega levega in desnega soseda. S temi točkami izračunamo začetna koeficienta za smer gibanja, s katerima si pomagamo pri določanju natančne lege povezave.



Slika 4.8: Grafični prikaz podrobnosti povezave med določanjem lege.

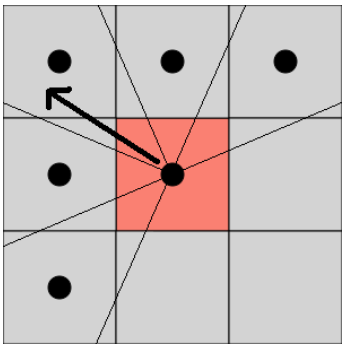
Slika 4.8 prikazuje modro označeno smer iskanja točk in najdene izhodiščne točke. Smeri začetnih smernih koeficientov k_1 in k_2 sta označeni s svetlo zeleno barvo. Rumeni del povezave predstavlja točke potovanja proti desnemu koncu povezave z začetnim smernim koeficientom k_1. Temno-zeleni del pa podobno predstavlja množico točk, ki smo dobili s potovanjem v proti levemu koncu smeri z začetnim koeficientom k_2.

Za potovanje v smereh k_1 in k_2 izvedemo naslednji postopek, ki v vsakem koraku:

1. izračuna novo smer gibanja z upoštevanjem zgodovine,
2. sestavi množico kandidatov za naslednjo točko,
3. izbere prvo belo točko iz te množice:
   3.1. doda točko v množico točk povezave,
   3.2. izvede premik v dodano točko in se vrne v nov korak,
4. če v množici ni belih točk, prekine iskanje.

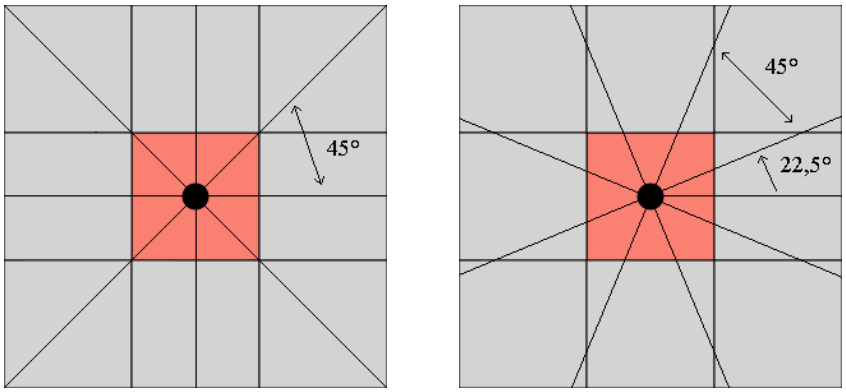
Rezultat postopka predstavlja Slika 4.8, kjer sta levi in desni del premice obarvana. Smer se izračuna iz trenutne točke in točke izpred 20 korakov, kar smo določili s testiranjem. Če zgodovine ne upoštevamo, se v nekaterih primerih zgodi, da algoritem potovanja po povezavi krene s poti, zato je zgodovino potrebno upoštevati, da se algoritem obdrži v pravi smeri. Do odstopanja lahko pride tudi zaradi šuma, ki v obliki točk tik ob premici nastane kot posledica postopkov binarizacije in tanjšanja robov.

Pri gibanju po povezavi v 2. točki je zelo pomembna funkcija, ki glede na točko in smer sestavi seznam točk oz. kandidatov za naslednjo točko. Za poljubno smer vrne 5 točk. Prva sovpada s smerjo gibanja, druga in tretja pa sta njena soseda, četrta in peta točka pa soseda druge in tretje točke (Slika 4.9).



Slika 4.9: Prikaz točke v sredini in izračunane smeri gibanja. Pike prikazujejo, katere sosedne točke izbere funkcija.

Ker ima točka 8 sosedov, smo prostor smeri razdelili na 8 intervalov širine 45°. Ker na točko gledamo kot kvadrat, je bolj ustrezno, da zaloge vrednosti intervalov zamaknemo za polovico širine posameznega intervala, kar znese 22,5°. Tako je smer gibanja usklajena s položajem sosednjih točk (Slika 4.10).



Slika 4.10: Razdelitev prostora kotov na 8 intervalov. Levo je razdelitev, kjer začnemo pri kotu 0°, desno pa so intervali zamaknjeni za 22,5°.

S postopkom prepoznave obeh delov povezave smo tako dobili dve množici točk, ki predstavljata levi in desni del povezave. Da se dokončno oceni, ali gre za povezavo ali ne, nadaljujemo s postopkom, ki:

1. združi množici obeh delov povezave (Slika 4.11),
2. če je v množici manjše število točk, kot je velikost premera vozlišča, se postopek prekine in ponovi od začetka za naslednjo povezavo,
3. za vsako vozlišče iz seznama vozlišč:

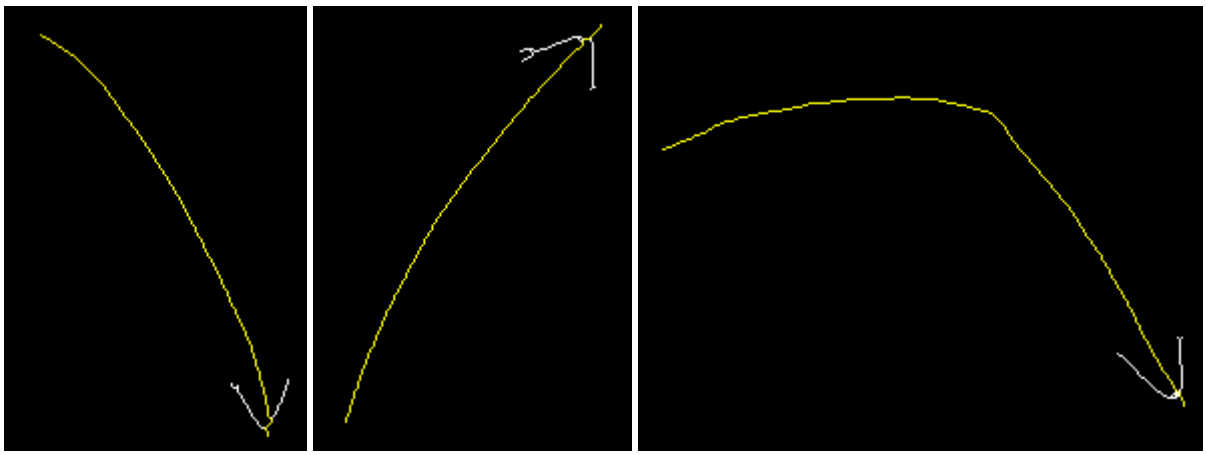
   3.1. izračuna razdaljo d_1 med vozliščem in prvim koncem povezave:

      3.1.1. če velja $d_1 < 1{,}5 \cdot \bar{r}$, kjer je $\bar{r}$ povprečni polmer vozlišč potem nastavi vozlišče prvemu koncu povezave,

   3.2. izračuna razdaljo d_2 med vozliščem in drugim koncem povezave:

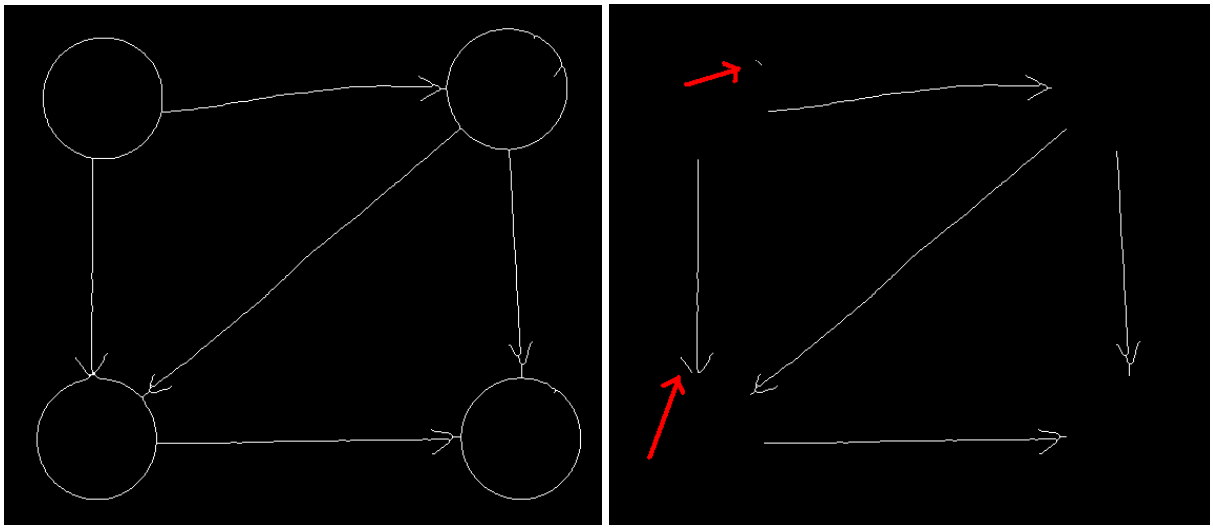
      3.2.1. če velja $d_2 < 1{,}5 \cdot \bar{r}$, kjer je $\bar{r}$ povprečni polmer vozlišč potem nastavi vozlišče drugemu koncu povezav,

4. povezavo shrani v seznam prepoznanih povezav in postopek ponovi od začetka za naslednjo povezavo.



Slika 4.11: Slike prepoznanih povezav, kjer so rumeno obarvane vse točke, ki smo jih obiskali pri določanju lege povezave.

Spodnjo omejitev dolžine povezave smo določili s testiranjem. Tako smo odstranili tudi ostanke puščic usmerjenih povezav in ostanke vozlišč po brisanju (Slika 4.12, desno). Za bližino vozlišč krajišču povezave smo s testiranjem določili vrednost $1{,}5 \cdot \bar{r}$. Izkazalo se je, da temu pogoju vedno zadosti samo eno vozlišče.

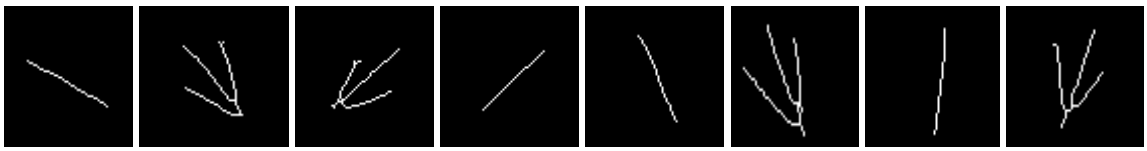


Slika 4.12: Skelet pred in po brisanju vozlišč. Spodaj vidimo ostanek puščice usmerjene povezave, zgoraj pa majhen ostanek, ki je pripadal vozlišču.

4.4.2. Usmerjenost

Usmeritev povezave je določena z njenima krajiščema. Krajišče lahko predstavlja črta oz. puščica. Skupaj to pomeni štiri različne usmeritve povezave. Za določitev povezave algoritem potrebuje seznam povezav in skelet grafa. Določanje krajišč povezave se izvaja ločeno.

Krajišče neusmerjene povezave sestavlja glavni krak, ki je del povezave. Pri usmerjenem krajišču pa gre za tri dele (Slika 4.13). Poleg glavnega tu obstajata še dva kraka, ki pa se s povezavo stikata v krajišču pod nekim kotom.



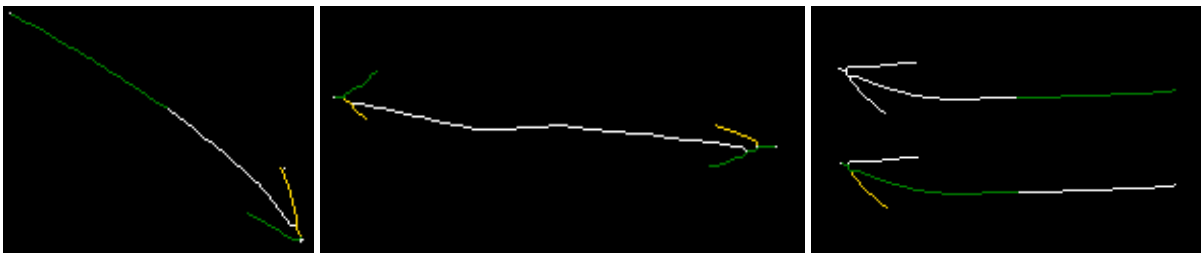
Slika 4.13: Različna krajišča povezav.

Prvi postopek, ki smo ga implementirali, je temeljil na uporabi Houghove transformacije za črte. Ker pa imamo tu manjše fragmente povezav in manjše sličice krajišč, se transformacija ni dobro obnesla, predvsem pa je bilo potrebnega veliko dodatnega dela, zato smo iskali drugačno rešitev. Razvili smo postopek, ki deluje na podoben način kot iskanje položaja povezave. Ker je potrebno preveriti obe krajišči, se postopek izvede za vsako krajišče posebej.

Postopek prepoznave usmeritve povezave:

1. izračuna naklon α povezave v krajišču,
2. izračuna naklona $\alpha_1 = \alpha + 35°$ in $\alpha_2 = \alpha - 35°$ za dodatna kraka krajišča,
3. za vsakega od krakov:
   - 3.1. sestavi množico kandidatov za naslednjo točko,
   - 3.2. izbere prvo belo točko iz te množice:
     - 3.2.1. doda točko v množico točk kraka,
     - 3.2.2. izvede premik v dodano točko,
   - 3.3. če število dodanih točk za krak preseže polovico dolžine povezave ali če v množici kandidatov za naslednjo točko ni belih točk, prekine iskanje,
4. če dolžina obeh krakov presega polovico dolžine povezave, shrani stanje povezave kot neusmerjeno,
5. izračuna delež točk, ki so vsebovane tudi v povezavi,
   - 5.1. če deleža točk v obeh krajiščih presegata polovico vseh točk, shrani stanje povezave kot neusmerjeno,
6. shrani stanje povezave kot usmerjeno.

Začetni naklon potrebujemo, da določimo kota puščic. Smeri krakov α_1 in α_2 se med potovanjem ne spreminjata. Točke se dodajajo enako kot pri povezavah. Ko v seznamu ni več točk bele barve, ali pa ko smo dosegli polovico dolžine povezave, smo dosegli robni pogoj.



Slika 4.14: Trije primeri prepoznanih krajišč povezav.

Slika 4.14 predstavlja točke, ki jih je algoritem izbral pri določanju krakov krajišč. Rumena in zelena barva predstavljata prvi in drugi krak krajišč. Levi in desni primer prikazujeta povezavi, kjer eno krajišče ni usmerjeno. Polovica povezave je obarvana zeleno, kar pomeni, da kraka ne obstajata in krajišče ni usmerjeno. Srednji primer predstavlja povezavo, kjer se jasno vidi vse štiri krake. Desni primer prikazuje levo krajišče, kjer je zeleni krak krenil s poti. Odločili smo se, da krajišče v takem primeru še vedo velja za usmerjeno.

Ko smo za neko povezavo preverili obe krajišči, glede na njuni stanji definiramo usmerjenost povezave in jo shranimo.

4.4.3. Utež

Vsaka utež v grafu je sestavljena iz ene ali več števk in pripada eni povezavi. Da utež pripišemo neki povezavi, je potrebno ugotoviti, kje v sliki se nahaja, katere števke jo sestavljajo ter kakšno vrednost predstavlja.

Postopek prepoznave uteži:

1. s pomočjo razreda `BlobCounter` (Poglavje 3.4) iz vhodne binarne slike ustvari seznam vseh blobov,
2. izbere blobe, za katere velja, da:

   2.1. se blob ne nahaja znotraj vozlišča,

   2.2. $širina(b) < \frac{širina(I)}{2}$ in $višina(b) < \frac{višina(I)}{2}$, kjer je I vhodna slika,

   2.3. $višina(b) < 4 \cdot \overline{r}$, kjer je $\overline{r}$ povprečni polmer vozlišča,

   2.4. $višina(b) > \frac{1}{4}\overline{r}$, kjer je $\overline{r}$ povprečni polmer vozlišča.

Zgornji pogoji pod 2. točko izločijo skelet, oznake vozlišč in vse manjše bele pike ali lise, ki predstavljajo šum. Preostale blobe pretvorimo v kandidate za uteži.

Kandidate nato združujemo med seboj po pravilu:

- če med bloboma b_1 in b_2 velja $d(b_1,b_2) < \bar{r}$, kjer je $d(b_1,b_2)$ oddaljenost med bloboma in $\bar{r}$ povprečni polmer vozlišč, potem bloba združi.

Če je oddaljenost med dvema kandidatoma za utež manjša od povprečnega polmera vozlišč v grafu, ju združimo. Kot rezultat dobimo množico uteži, ki jih pripišemo povezavam.

Slika 4.15: Posamezne uteži, ki med seboj razlikujejo po velikosti

Postopek pripisovanja uteži vsaki uteži poišče povezavo, ki ji je najbližja.

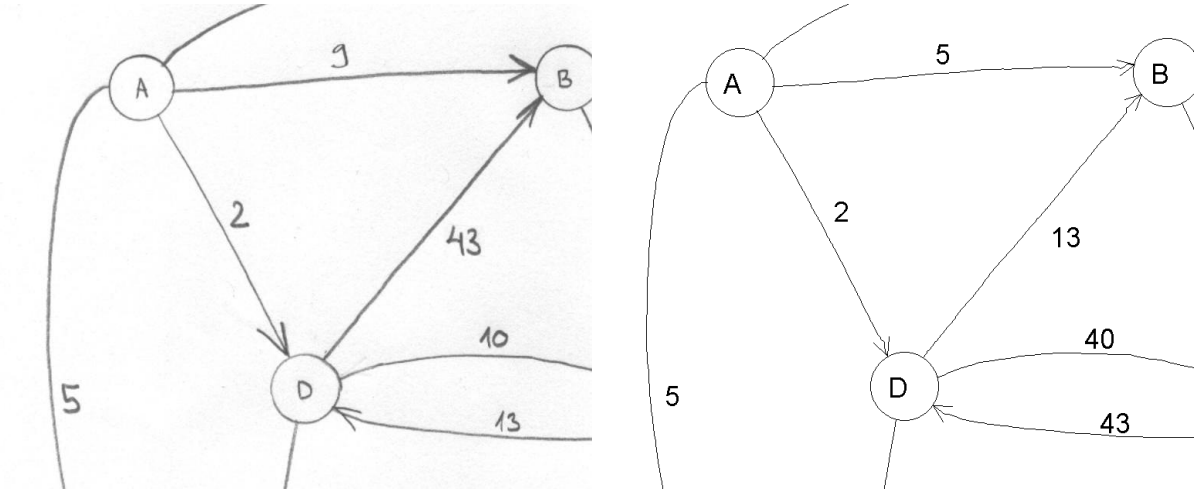
Za vsako utež:

1. za vsako povezavo:
   1.1. poišče točko, ki je najbližje uteži,
   1.2. če je povezava trenutno najbližja, shrani razdaljo in povezavo,
2. najbližji (shranjeni) povezavi pripiše utež

Ko imamo uteži pripisane povezavam, jih je potrebno še prepoznati. Za prepoznavo uporabljamo OCR knjižnico Tessnet, ki omogoča način iskanja po številkah.

Postopek prepoznave števk za vsako sliko uteži:

1. izvede prepoznavo znakov,
2. odstrani znake, ki niso številke,
3. shrani preostale znake v povezavo, kot vrednost uteži (Slika 4.16).

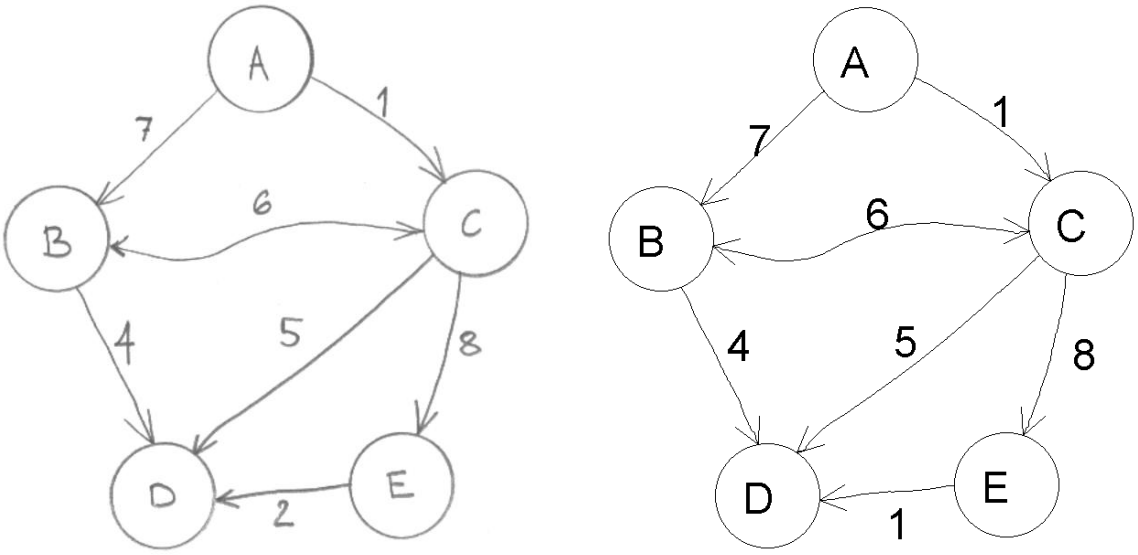


Slika 4.16: Uteži in povezave pred in po prepoznavi znakov.

Zaradi občutljivosti postopka prepoznave znakov in ročne pisave prihaja tu do veliko napak pri prepoznavi števk. Število napak se zmanjša, če pišemo števke v obliki podobni računalniškemu zapisu.

4.5. Vizualizacija in shranjevanje rezultatov

Program ob koncu prepoznave shrani graf v digitalno sliko, tako da lahko uspešnost algoritma takoj ocenimo (Slika 4.17).



Slika 4.17: Vhodna slika in slika z rezultati.

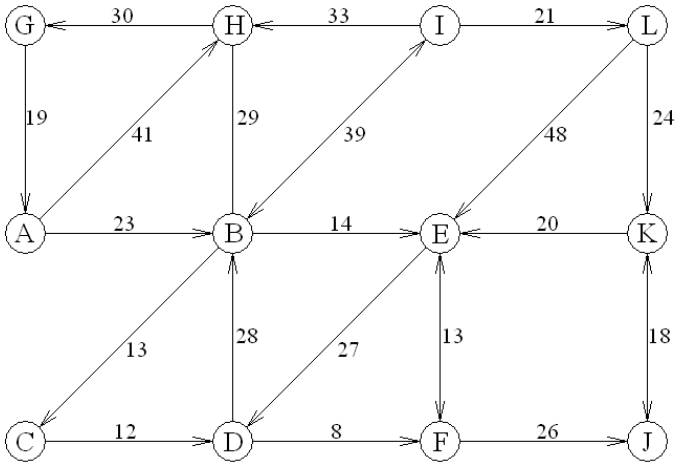
Celotni postopek prepoznave obsega tudi izvoz strukture v jezik XML (ang. *eXtensible Markup Language*) [10], ki omogoča zapis poljubne hierarhične strukture in s tem poljubno mnogo nivojev gnezdenja. Vsak nivo lahko označimo s poljubnim imenom.

Prvi nivo predstavlja oznaka *"Graph"* z atributi, ki vsebuje seznam vozlišč *"NodeList"* in seznam povezav *"EdgeList"*. Seznam vozlišč vsebuje oznake *"Node"* z atributi, kot sta lega in oznaka itd. V seznamu povezav so *"Edge"* oznake z atributi, kot so utež, usmerjenost, prvo vozlišče, drugo vozlišče itd.

5. Primeri uporabe

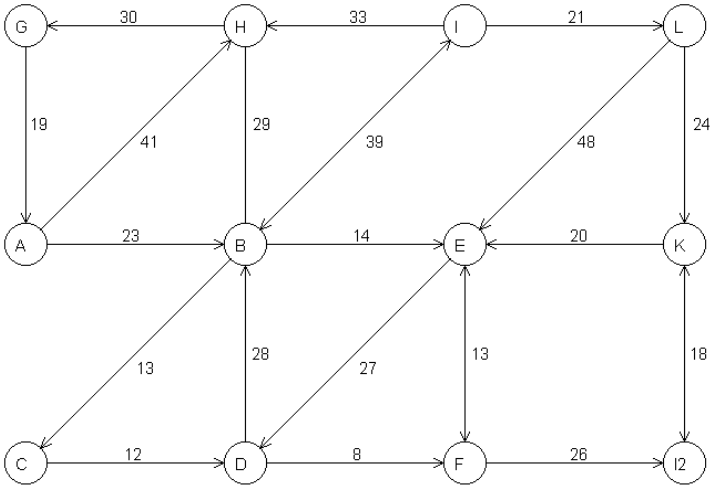
Delovanje algoritma za prepoznavo grafa iz slike smo preizkusili na različnih primerih. Tako smo ugotavljali in beležili različne pristope k oblikovanju grafa za dosego boljših rezultatov. Iz dobljenih rezultatov smo oblikovali navodila za risanje grafov, ki so podrobneje opisana v poglavju 4.1 Omejitve.

Slika 5.1 predstavlja idealen vhod, narisan s pomočjo računalnika in enostaven za prepoznavo. Slika je dimenzije 640 x 480, ima 12 vozlišč in 20 usmerjenih povezav. Uporabili smo pisavo Times New Roman.



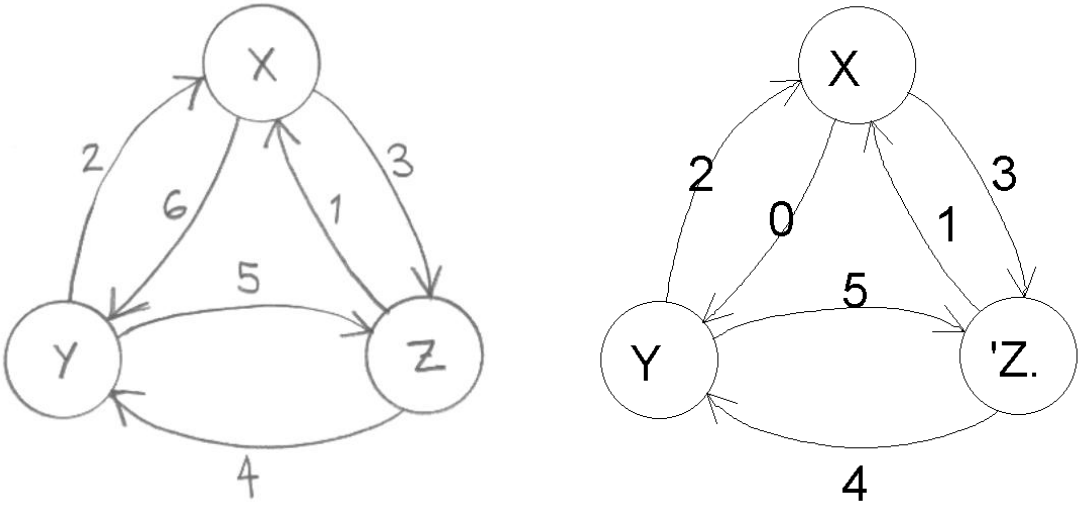
Slika 5.1: Vhodna slika računalniško skiciranega grafa.

Ker je primer enostaven in brez šuma, pri prepoznavi grafa skoraj ni bilo napak. Zabeležili smo eno napako pri oznaki vozlišča "J". Vse povezave v grafu so debeline ene pike, zato tanjšanje robov ni bilo potrebno. Vse uteži so bile kljub majhni dimenziji vhodne slike zaradi računalniške pisave uspešno prepoznane (Slika 5.2). Ker je oblika povezav enostavna (vse so ravne) in so vsa krajišča simetrična, algoritem za prepoznavo krajišč ni naredil napake.



Slika 5.2: Izhodna slika računalniško skiciranega grafa.

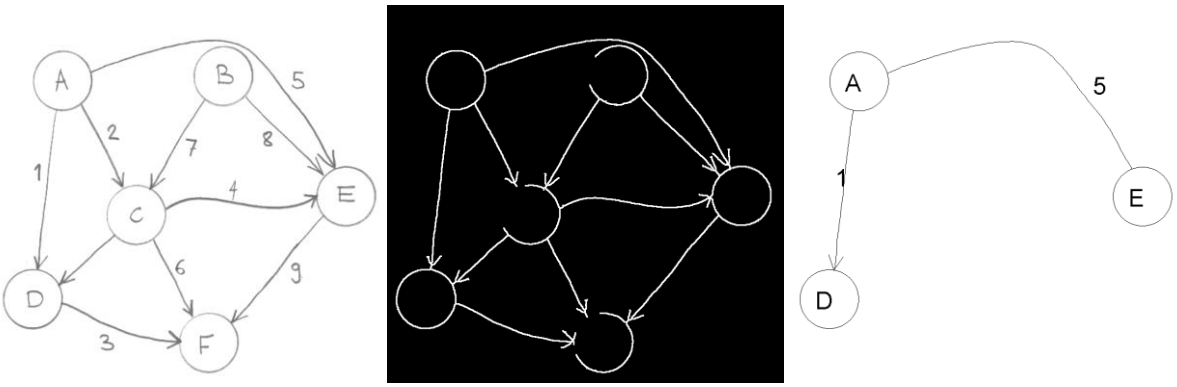
Naslednji primer pod Slika 5.3 predstavlja enostaven graf, kjer smo pri risanju pazili na oblikovanju znakov in števk, da so podobni računalniškemu zapisu znakov. Algoritem je uspešno prepoznal vsa vozlišča in povezave med njimi. Tudi krajišča povezav so bila uspešno določena. Pri prepoznavi uteži se je pojavila napaka na povezavi "X→Y". Opazili smo tudi napačno prepoznavo oznake vozlišča "Z", kjer je algoritem zaznal še druga dva znaka.



Slika 5.3: Vhodna in izhodna slika enostavnega grafa.

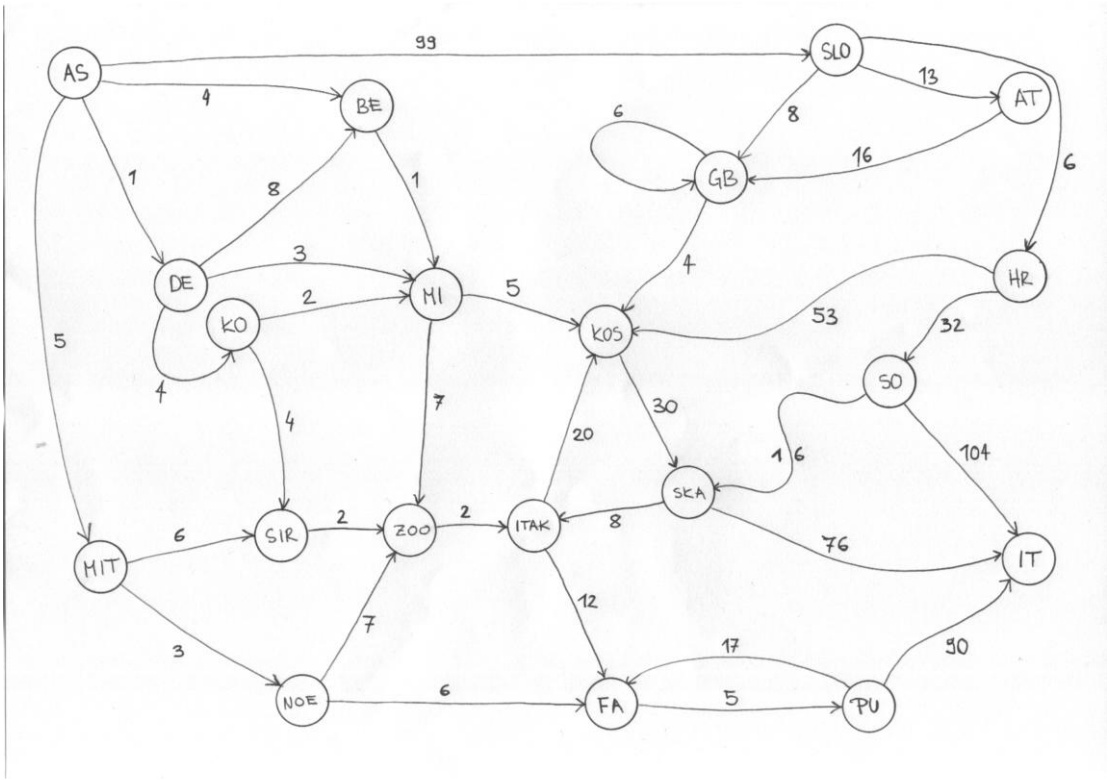
Naslednji primer (Slika 5.4) predstavlja rezultat prepoznave, ki je posledica nenatančnega risanja s pisalom, ki ni dovolj kontrastno glede na podlago. Ker je bil obod vozlišč "C", "B" in "F" premalo izrazit, se pri binarizaciji ni ohranil v celoti. Zato teh vozlišč pri prepoznavi algoritem ni našel. V nadaljevanju zato tudi povezave ki vodijo do teh vozlišč, niso bile

prepoznane. Poleg manjkajočih vozlišč opazimo tudi napačno oceno krajišča pri vozlišču "E" (Slika 5.4).



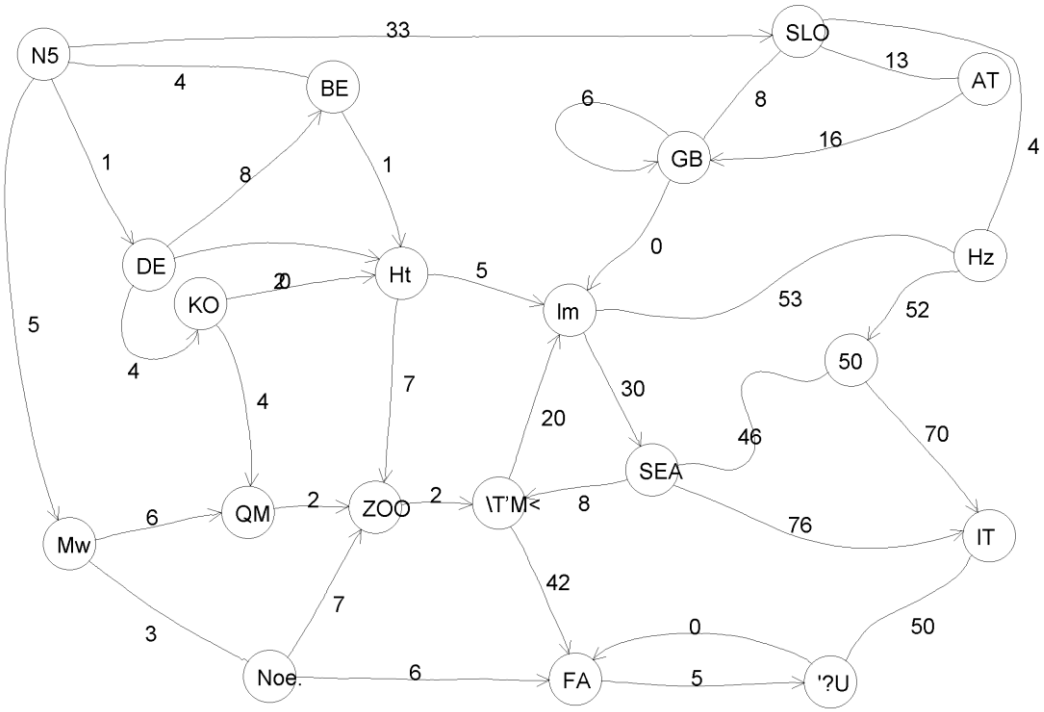
Slika 5.4: Vhodna slika, skelet grafa po binarizaciji in izhodna slika pomanjkljivo prepoznanega grafa.

Slika 5.5 predstavlja primer ročno narisanega grafa večje velikosti z 20 vozlišči in preko 30 usmerjenimi povezavami. Vhodna slika je bila velikosti približno 2300 x 1700 točk. Na sliki levo opazimo rob lista, ki je posledica skeniranja. Ker gre za večjo sliko, je bilo tudi delovanje prepoznave počasnejše.



Slika 5.5: Vhodna slika grafa z veliko vozlišči.

Pri prepoznavi grafa so bila vsa vozlišča in povezave prepoznane (Slika 5.6). Prišlo je do pričakovanih napak pri prepoznavi oznak vozlišč in uteži povezav. Program je prepoznal tudi zanko vozlišča "GB". Pri oznakah opazimo, da so poleg črk prepoznani tudi drugi znaki, na primer pika pri "Noe." Pri določanju smeri je prišlo do petih napak, na primer med vozliščema "SLO" in "Hz". Program uteži "3" med vozliščema "DE" in "Ht" ni prepoznal, ker se je na vhodni sliki po postopku binarizacije utež (Slika 5.5) dotikala povezave.



Slika 5.6: Izhodna slika grafa z veliko vozlišči.

6. Sklepne ugotovitve

Postopek prepoznave je uspešen, če se pri oblikovanju vhoda držimo omejitev, kot so ukrivljenost povezav, oblika krajišč, velikost uteži glede na vozlišča, dotikanje utež povezavam, itd. Največja odstopanja v postopku prepoznave so pričakovano predstavljale napake pri optični prepoznavi uteži in oznak, vendar je večji delež napak prisotnih pri prepoznavi uteži. Ugotovili smo, da je zelo odvisno, kako oblikujemo posamezne znake in števke. S pravilno obliko smo na primer lahko oblikovali vhod, pri katerem je bilo v postopku prepoznave zelo malo napak. V primeru ko prepoznava skeleta grafa ni bila uspešna, ni bilo smisla nadaljevati, ker smo dobili prazen rezultat. V nekaj primerih se je zgodilo, da vsa vozlišča niso bila prepoznana, ker obod nekaterih vozlišč ni imel obliko krožnice (Slika 5.4) ali pa ni bil dovolj kontrasten glede na podlago. V tem primeru smo prepoznali samo medsebojne povezave med prepoznanimi vozlišči. Nekatere povezave zaradi svoje ukrivljenosti niso bile prepoznane. Do tega je prihajalo zato, ker pri določanju lege povezave uporabljamo zgodovino, ki omejuje največjo ukrivljenost povezav glede na velikost vozlišč.

Pri implementaciji smo imeli nekaj težav s preslikavami, zato smo napisali knjižnico funkcij, s katerimi smo računali kote, presečišča in razdalje med točkami. Težavo z izhodiščno obliko vhodne slike smo odpravili z uporabo metodo tanjšanja povezav. Nastalo sliko smo nato uporabili za prepoznavo vozlišč. Prvi pristop za iskanje vozlišč je bila uporaba Houghove transformacije za krožnice, ki pa zahteva definicijo polmera krožnic. Ker pa se velikost vozlišč od slike do slike razlikuje, smo rešitev iskali v drugačnem pristopu, ki ne zahteva velikosti vozlišč. Ugotovili smo, da z uporabo ekstrakcije povezanih elementov lahko izluščimo sliko skeleta in od tu dalje poiščemo vozlišča. Pri določanju krajišč povezav smo si najprej pomagali s Houghovo transformacijo za premice, a se ta ni izkazala za uspešno, zato smo razvili drug postopek. Postopek temelji na algoritmu, ki od krajišč povezave potuje po krakih krajišča. Izkazal se je za dovolj učinkovitega in s sorazmerno malo napakami.

Aplikacija bi lahko z prilagoditvijo postopka prepoznave omogočala različne razširitve, ki jo naredijo uporabno tudi za reševanje drugih problemov. Tako bi lahko obseg grafov razširili tudi na neravninske grafe. To bi dosegli z nadgradnjo prepoznavanja povezav tako, da bi

zaznali tudi sekajoče se povezave. Glede na podobnost grafov z avtomati bi lahko opisali in prepoznali tudi različne končne deterministične oz. nedeterministične avtomate. Tu bi na primer vozlišča predstavljala množico stanj, povezave pa prehodno funkcijo in črke abecede. Podobno bi bilo, če bi prepoznali vozlišča v obliki štirikotnika ali elipse, ker bi na podoben način aplikacijo lahko priredili tako, da bi lahko prepoznala diagrame potekov.

Literatura

[1] S.Deng, S. Latifi, E. Regentova, "Document segmentation using polynomial spline wavelets", Pattern Recognition, št. 34, pp. 2533 – 2545, 2001

[2] C. Strouthopoulos, N. Papamarkos, C. Chamzas, "Identification of Text-Only Areas in Mixed-Type Documents", *Engng. Applic. Artif. Intell.*, Vol. 10, No. 4, pp. 387 – 401, 1997

[3] E. Trucco, A. Verri, *Introductory Techniques for 3-D Computer Vision*, New Yersey: Prentice Hall, 1998, pogl. 4.2 in 5.2.

[4] (2010) AForge.NET Framework, Connected Components Labeling, Dostopno na: http://www.aforgenet.com/framework/docs/html/240525ea-c114-8b0a-f294-508aae3e95eb.htm

[5] (2010) AForge.NET Framework, SISThreshold, Dostopno na: http://www.aforgenet.com/framework/docs/html/39e861e0-e4bb-7e09-c067-6cbda5d646f3.htm

[6] (2003) Connected Components Labeling, Dostopno na: http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm

[7] (2010) Optical character recognition, Dostopno na: http://en.wikipedia.org/wiki/Optical_character_recognition

[8] (2010) Graf (diskretna matematika). Dostopno na: http://wiki.fmf.uni-lj.si/wiki/Graf

[9] (2010) Image Binarization. Dostopno na: https://www.research.ibm.com/haifa/projects/image/glt/binar.html

[10] (2010) Extensible Markup Language (XML), Dostopno na: http://en.wikipedia.org/wiki/XML

[11] (2010) Grayscale, Dostopno na: http://en.wikipedia.org/wiki/Grayscale

[12] (2010) Binary image, Dostopno na: http://en.wikipedia.org/wiki/Binary_image

Dodatek

Struktura programa z razredi

Program je zgrajen modularno. Razredi so izpeljani iz t.i. osnovnih (*ang.* base class) razredov, ki določajo njihovo strukturo in medsebojno povezanost. Razred `GraphRecognizer` ima na primer definirana razreda `EdgeRecognizer` in `NodeRecognizer`, ki ju mora uporabnik definirati, če želi napisati aplikacijo, ki bi ustrezala tej strukturi osnovnih razredov. Domenski razredi predstavljajo povezave in vozlišča, abstraktni pa definirajo strukturo oz. relacije med razredi.

Pomožni razredi, ki v nadaljevanju niso definirani:

- `GraphUtil`, ki vsebuje matematične funkcije za računanje kotov, presečišč in razdalj med točkami in premicami itd.
- `BasicImageProcessor`, ki vsebuje vse funkcije za delo z grafičnimi elementi, kot so: pretvorba v sivinsko sliko, negativ slike, tanjšanje robov in Houghova transformacija.
- `GraphExporter`, ki vsebuje vse potrebno za izvoz strukture grafa v XML datoteko.

Domenski razredi

```
// domenski razred Edge - povezava
public class Edge
{
    public Node Node1;
    public Node Node2;
    public int Weight;
    public System.Drawing.Point Node1ClosestPoint;
    public System.Drawing.Point Node2ClosestPoint;
    public System.Drawing.Point Node1ClosestPointOnNode;
    public System.Drawing.Point Node2ClosestPointOnNode;
    public string WeightString;
    public Const.Enum.EdgeDirection Direction;
    public List<System.Drawing.Point> Points;
    public System.Drawing.Point WeightLocation;
```

```
    public Edge();
    public override string ToString();
    public string ToDirectedString();
}

// domenski razred Node - vozlišče
public class Node
{
    public Point Location;
    public string Name;
    public List<Edge> Edges;
    public int Radius;
    public Rectangle Rectangle;

    public Node();
    public Node(string name, Point location);
    public void AddEdge(Edge edge);
    public void RemoveEdge(Edge edge);
    public override string ToString();
}
// domenski razred Graph - graf
public class Graph
{
    public List<Node> Nodes;
    public List<Edge> Edges;

    public Graph();
    public void AddNode(Node node);
    public void AddEdge(Edge edge);
    public void LinkEdgesToNodes();
    public override string ToString();
}
```

Pomožni razredi

```
// pomožni razred NodeCandidate - kandidat za vozlišče
public class NodeCandidate : Node
{
    public Point AvgLocation;
    public List<Point> Locations;
    public int Count;

    public Bitmap SourceBitmap;
    public NodeCandidate();
    public override string ToString();
}

// pomožni razred EdgeCandidate - kandidat za povezavo
public class EdgeCandidate : Edge
{
    public Bitmap Bitmap;
    public Bitmap WeightBitmap;
    public List<EdgeWeightCandidate> WeightCandidates;
    public Bitmap End1Bitmap;
    public Bitmap End2Bitmap;
    public Point[] End1RectCoordinates;
    public Point[] End2RectCoordinates;

    public EdgeCandidate();
```

```
    public override string ToString();
}

// pomožni EdgeWeightCandidate - kandidat za utež vozlišča
public class EdgeWeightCandidate
{
    public List<System.Drawing.Rectangle> Rects;
    public List<System.Drawing.Bitmap> Images;
    public List<EdgeCandidate> EdgeCandidates;
    public int ClosestDistance;
    public int Width;
    public int Height;

    public System.Drawing.Point Center();
    public int MinX();
    public int MaxX();
    public int MinY();
    public int MaxY();
}
```

Abstraktni razredi

```
// abstraktni razred GraphRecognizer - prepoznava grafa
public abstract class GraphRecognizer
{
    public Bitmap GraphSourceBitmap;
    public Domain.Graph Graph;

    public GraphRecognizer();
    public GraphRecognizer(Bitmap graphBitmap);

    // abstraktni razredi, ki morajo biti definirani
    public abstract GraphRecognizerBase.NodeRecognizer NodeRecognizer;
    public abstract GraphRecognizerBase.EdgeRecognizer EdgeRecognizer;
    public abstract GraphRecognizerBase.Domain.Graph RecognizeGraph();
}

// abstraktni razred NodeRecognizer - prepoznava vozlišč
public abstract class NodeRecognizer
{
    public GraphDataContainer GraphData;
    public NodeRecognizer();
    public NodeRecognizer(GraphDataContainer GraphData);

    // abstraktni razredi, ki morajo biti definirani
    public abstract NodeLocationRecognizer NodeLocationRecognizer;
    public abstract NodeNameRecognizer NodeNameRecognizer;
    public abstract List<GraphRecognizerBase.Domain.Node> RecognizeNodes();
}

// abstraktni razred NodeLocationRecognizer - prepoznava položaja vozlišč
public abstract class NodeLocationRecognizer
{
    public GraphDataContainer GraphData;
    public NodeLocationRecognizer();
    public NodeLocationRecognizer(GraphDataContainer GraphData);

    // abstraktni razred, ki mora biti definiran
    public abstract List<GraphRecognizerBase.Domain.NodeCandidate>
        RecogniseNodeLocations();
}
```

```
// abstraktni razred NodeNameRecognizer - prepoznava oznake vozlišč
public abstract class NodeNameRecognizer
{
    public GraphDataContainer GraphData;
    public NodeNameRecognizer();
    public NodeNameRecognizer(GraphDataContainer GraphData);

    // abstraktni razred, ki mora biti definiran
    public abstract List<GraphRecognizerBase.Domain.NodeCandidate>
        RecogniseNodeNames();
}

// abstraktni razred EdgeRecognizer - prepoznava povezav
public abstract class EdgeRecognizer
{
    public GraphDataContainer GraphData;
    public EdgeRecognizer();
    public EdgeRecognizer(GraphDataContainer GraphData);

    // abstraktni razredi, ki morajo biti definirani
    public abstract EdgeLocationRecognizer EdgeLocationRecognizer;
    public abstract EdgeWeightRecognizer EdgeWeightRecognizer;
    public abstract List<Domain.Edge> RecognizeEdges();
}

// abstraktni razred EdgeLocationRecognizer - prepoznava položaja povezav
public abstract class EdgeLocationRecognizer
{
    public GraphDataContainer GraphData;
    public EdgeLocationRecognizer();
    public EdgeLocationRecognizer(GraphDataContainer GraphData);

    // abstraktni razred, ki mora biti definiran
    public abstract List<GraphRecognizerBase.Domain.EdgeCandidate>
        RecognizeEdgeLocations();
}

// abstraktni razred EdgeDirectionRecognizer - prepoznava usmeritve povezav
public abstract class EdgeDirectionRecognizer
{
    public GraphDataContainer GraphData;
    public EdgeDirectionRecognizer();
    public EdgeDirectionRecognizer(GraphDataContainer GraphData);

    // abstraktni razred, ki mora biti definiran
    public abstract List<GraphRecognizerBase.Domain.EdgeCandidate>
        RecogniseEdgeDirections(
            List<GraphRecognizerBase.Domain.EdgeCandidate> edgeCandidates);
}

// abstraktni razred EdgeWeightRecognizer - prepoznava uteži povezav
public abstract class EdgeWeightRecognizer
{
    public GraphDataContainer GraphData;
    public EdgeWeightRecognizer();
    public EdgeWeightRecognizer(GraphDataContainer GraphData);

    // abstraktni razred, ki mora biti definiran
    public abstract List<GraphRecognizerBase.Domain.EdgeCandidate>
        RecognizeEdgeWeights();
}
```

Izjava o avtorstvu

Izjavljam, da sem diplomsko nalogo izdelal samostojno pod vodstvom mentorja doc. dr. Bojana Kverha. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.

Simon Birk