Spletni pajek - 1. seminarska naloga

Delfina Bariša, Žiga Simončič, Nejc Smrkolj Koželj

2. april 2019

1 Uvod

Spletni pajek (v nadaljevanju pajek) je programska oprema, ki izkorišča medsebojno povezavo spleta za različne namene. Nas bo zanimalo predvsem pridobivanje podatkov. Informacije, ki jih najde na spletu so lahko sestavljene iz spletne strani, slik in drugih podatkovnih oblik.

Poznamo več vrst pajkov, vsi pa so sestavljeni iz naslednjih komponent:

- Začetne strani
- Frontier
- Shramba strani

Začetne strani so strani na katerih bo pajek začel iskati informacije. Če se med informacijami, ki jih pridobi na spletni strani, nahajajo nove strani, te doda v seznam še ne obiskanih strani. Ta seznam imenujemo frontier. Obiskane strani pa shrani v shrambo strani.

Pajki imajo različne strategije po katerih določijo, katero stran bodo naslednjo vzeli iz frontierja. Nas bo zanimal pajek, ki išče v širino. Ta ima forntier implementiran po principu "prvi noter, prvi ven". Tako torej iz vrste vzame tisto stran, ki je najdlje v njej.

V seminarski nalogi bomo implementirali večnitnega spletnega pajka, ki obiskuje samo .gov.si spletne strani po strategiji "prvi noter, prvi ven". V nadaljevanju bomo podrobneje opisali implementacijo takšnega pajka in naredili njegovo analizo.

2 Splošen pregled spletnega pajka

Ob zagonu spletni pajek svojo pot začne z začetnimi povezavami (domenami), ki se zapišejo v frontier. Nato iz frontier-ja pridobi povezave po principu "prvi noter prvi ven", oziroma FIFO. Vsako povezavo obišče, izvede *JavaScript* in izlušči vse povezave (po želji izlušči in shrani tudi slike in datoteke). Povezave ustrezno filtrira (samo *.gov.si* domene) in zavrže duplikate. Ustrezno upošteva ukaze iz morebitnih datotek robots.txt in sitemap. Uporablja tudi kanonikalizacijo povezav. Ko pajek obišče stran, pred luščenjem povezav preveri še vsebinsko podobnost strani z že obiskanimi. To ponavlja, dokler ga ne ustavimo.

Pajek ima tudi možnost izvajanja v več nitih. Napisan je v Python-u. Lahko ga prilagajamo s pomočjo spodnjih parametrov:

- USE_MULTITHREADING: boolean, vklopi paralelizacijo.
- NUMBER_OF_THREADS: integer, število niti v primeru paralelizacije.
- FRESH_START: boolean, če True, pobriše vse zapise iz baze in začne od začetka, drugače nadaljuje.
- DEFAULT_CRAWL_DELAY: integer, privzet čas v sekundah med zahtevami, če robots.txt ne obstaja ali crawl delay ni definiran.
- CRAWLER_EMPTY_FRONTIER_SLEEP: integer, čas spanja v sekundah, če v frontier-ju ni ustreznih povezav.
- MAX_DEPTH: integer, največja globina, do katere bo pajek dodajal povezave v frontier.
- MIN_DEPTH: integer, najmanjša globina, od katere bo pajek jemal povezave iz frontier-ja.

3 Podrobnosti spletnega pajka

3.1 Podatkovna baza

Shemo podatkovne baze *crawldb* smo uporabili, kot je pisalo v navodilih.

V tabeli *site* shranjujemo vse najdene domene (in poddomene, ki jih obravnavamo kot ločene domene), razčlenjeno vsebino datoteke robots.txt in vsebino *sitemap*-a.

Tabelo smo tudi razširili z atributoma *crawl_delay* (tipa *integer*) in *next_access_time* (tipa *integer*).

Atribut *crawl_delay* ima enako vrednost kot v datoteki robots.txt, dodali smo ga zaradi priročnosti in lažjega dostopa do vrednosti. Atribut *next_access_time* predstavlja čas UNIX in nam pove, kdaj lahko izvedemo naslednjo zahtevo na to domeno.

Tabelo *page* uporabljamo kot frontier in shrambo vseh obdelanih povezav/strani. Tipe zapisov ločujemo z atributom *page_type_code*. V splošnem: vrednost *HTML* pomeni obdelano povezavo/stran, *FRONTIER* pomeni, da povezava še ni obdelana, *DUPLICATE*, da je povezava duplikat in ni bila podrobneje pregledana. V tabelo *page* sharanjujemo samo povezave na navadne spletne strani, povezave na slike in dokumente hranimo v drugih tabelah. Tabelo smo tudi razširili z atributi *depth* (tipa *integer*), *hash* (tipa *integer array*) in *processing* (tipa *boolean*). Atribut *depth* nam pove globino, na kateri se povezava nahaja (v kodi ga ne uporabljamo, namenjen je analizi in statistiki), *hash* hrani *minHash* podpis, ki predstavlja vsebino strani, *processing* pa označuje, če je povezava trenutno v obdelavi (sinhronizacija med nitmi).

Ostalih tabel nismo spreminjali. V tabeli *image* hranimo slike, v tabeli *page_data* binarne datoteke in v tabeli *link* povezave med stranmi v tabeli *page*.

3.2 Pridobivanje povezav

Povezave po vrsti pridobivamo iz tabele *page* (frontier) in sicer urejeno po času dostopa (čas dostopa je v našem kontekstu čas, ko smo povezavo našli). Če v danem trenutnku v tabeli ni

ustreznih zapisov (bodisi zaradi *crawl_delay*-a ali ker je zares prazna), bo pajek počakal nekaj sekund in poskusil znova (čas čakanja lahko v kodi nastavimo s parametrom).

Dobljeno povezavo kanonikaliziramo (kot je opisano na prosojnicah) in obiščemo s pomočjo orodja *Selenium*, ki nam požene tudi kodo *JavaScript*. Uporabljamo *Chrome webdriver*. Iz izvorne kode dobljene spletne strani izluščimo vse povezave v značkah <*a*>, ki vsebujejo atribut *href*. Pregledamo tudi vse značke, ki vsebujejo atribut *onClick*. Relativne povezave razširimo na absolutne, če obstaja, za razširitev uporabimo povezavo v znački <*base*>, drugače pa glede na trenutni URL. Iz teh poskušamo pridobiti povezavo do spletne strani, ki se nahaja za kodo oblike *location.href* ali *document.location*.

Izluščene povezave shranimo v tabelo *page* kot tip *FRONTIER*, pred tem pa še zavržemo vse povezave, ki ne pripadajo domeni *.gov.si*. Zavržemo tudi vse duplikate (ujemanje po URL-ju) in upoštevamo ukaze v robots.txt (*disallow, crawl-delay, user-agent, sitemap*).

3.3 Pridobivanje slik, in binarnih datotek

Takoj po pridobivanju povezav iz prenesene spletne strani se pridobi povezave do slik in binarnih datotek.

Za iskanje binarnih datotek iz spletne strani izločimo vse povezave v značkah <*a*>, ki vsebujejo v atributu *href* eno izmed končnic binarnih datotek. Te končnice so definirane v podatkovni bazi v tabeli *data_type*. Trenutna implementacija išče povezave, ki se končajo na eno izmed sledečih končnic: *.pdf*, *.doc*, *.docx*, *.ppt* in *.pptx*.

Za iskanje slik iz spletne strani izločimo vse povezave v značkah <*src*>, ki vsebujejo atribut *scr* pri čemer se mora povezava končati na eno izmed definiranih končnic: *.jpg*, *.jpeg*, *.png*, *.gif*, *.tiff*, *.tif*, *.raw*. Slik, kjer se vsebina poda kar v atribut src preko base64 kodiranja ignoriramo.

Po pridobitvi povezav se datoteke in slike pridobi po enakem postopku kot se pridobi spletno stran. Na URL se s pomočjo Seleniuma pošlje GET zahtevek, vsebino odgovora pa se zapiše v podatkovno bazo.

3.4 Zaznavanje duplikatov

Pajek pri pregledovanju spletnih strani, še posebej tistih s podobno tematiko, velikokrat naleti na spletne strani, ki jih je že obiskal. Takšne strani imenujemo dvojniki. Slednje ne želimo dodajati v frontier, ker bi v obratnem primeru dvakrat ali celo več krat obiskali isto spletno stran in iz njih pridobili že znane informacije. To pa je tako časovno kot prostorsko potratno za pajka. Iz tega razloga želimo dvojnike zaznati ter preprečiti, da bi jih pajek dodal v frontier.

Dvojnike lahko velikokrat zaznamo že po identičnih spletnih naslovih. Te zaznamo tako, da preverimo, ali že obstajajo v frontierju. Spletni naslovi pa so lahko v različnih oblikah, zato jih moramo pred dodajanjem v frontier kanonikalizirati, kar smo tudi implementirali.

Zna pa se tudi zgoditi, da se dve strani nahajata na različnih spletnih naslovih, a imata identično ali skoraj podobno vsebino. Način kako zaznamo dvojnike po vsebini bo opisan v poglavju 3.4.1

3.4.1 Zaznavanje duplikatov po vsebini

Dve spletni strani lahko glede na vsebino zelo enostavno primerjamo:

- s točno primerjavo njune HTML vsebine ali
- pa tabelo v podatkovni bazi dopolnimo s hash vrednostjo HTML vsebine in dve strani primerjamo gleda na njuni hash vrednosti.

Slabost teh dveh načinov je, da zaznata samo popolnoma enake strani. Tiste, ki so zelo podobne (se razlikujeta minimalno) pa ne bosta zaznala kot dvojnikov. Kar pomeni, da obstaja možnost, da bomo v frontier dodali podvojeno stran in posledično iz dveh strani izluščili praktično skoraj enake informacije

Tovrstne slabosti smo se želeli izogniti in smo se zato odločili za zaznavanje dvojnikov gleda na vsebino s pomočjo minHash metode lokalno občutljivega razprševanja in Jaccard razdalje.

Razložimo, kako smo implementirali iskanje dvojnikov glede na vsebino:

1. Za vsako spletno stran najprej pridobimo vsebino. Da bi dosegli natančnejšo primerjavo, s pomočjo knjižnice BeutifulSoup izluščimo vsebino HTML strani brez HTML oznak.

2. Nato iz dobljenega niza znakov Q, ustvarimo seznam S, ki po vrsti vsebuje podnize iz Q dolžine 8. Razlog za izbiro te dolžine, se skriva v povprečni dolžini slovenske besede, ki je ravno 8.

3. Vsakemu nizu iz S priredimo hash vrednost in dobimo seznam hash vrednosti.

4. Nato na seznamu S uporabimo minHas metodo in ustvarimo minHas podpis.

   MinHash metoda deluje tako, da s pomočjo razpršitvene funkcije preslika 32 bitno celo število v drugo celo število.

   Enačba 1 prikazuje razpršitveno funkcijo, ki smo jo uporabili.

   $$h(x) = (ax + b)\%c \tag{1}$$

   Kjer a in b predstavljata naključni vrednosti, ki sta manjši od maksimalne vrednosti x-a. c pa predstavlja praštevilo, ki je za malenkost večje od maksimalne vrednosti x-a. Tako bo razpršitvena funkcija za različne izbire a in b, ustvarila drugačno naključno preslikavo vrednosti. Kar pomeni, da lahko izračunamo poljubno število naključnih razpršitvenih funkcij. Zelo pomembno je, da za vse vsebine strani uporabimo iste razpršitvene funkcije, torej iste vrednosti a in b.

   Da bo primerjava čim bolj natančna, smo v naši implementaciji uporabili 250 razpršitvenih funkcij. Za vsako razpršitveno funkcijo smo naredili sledeče:

   (a) Na vsakem elementu v tabeli S uporabimo razpršitveno funkcijo.

   (b) Izmed vseh izračunanih vrednosti v izberemo minimalno (te vrednosti shranjujemo v tabelo). Od tod izvira ime minHash.

Ko pridemo čez vse funkcije dobimo, tabelo dolžine 250. Ta predstavlja minHas podpis trenutne vsebine strani.

5. Nato pridobimo minHash podpise vseh že pregledanih strani. MinHash podpis trenutne strani s pomočjo Jaccard razdalje primerjamo z vsakim izmed pridobljenih podpisov. Formulo za Jaccardovo razdaljo prikazuje enačba 2.

$$razdalja = \frac{minHasPodpisTrenutne \cap minHasPodpisPridobljene}{minHasPodpisTrenutne \cup minHasPodpisPridobljene} \tag{2}$$

Jaccardova razdalja vrne vrednost med 0 in 1. Ta nam pove kako podobni sta si strani (bližje kot je 1,bolj podobni sta si). Iz tega razloga, moramo določit primerno mejo, ki bo realno določal od katere vrednosti naprej, dve strani označimo kot dvojnici. Ker smo uporabili veliko razpršitvenih funkcij, je bila Jaccardova razdalja primerno višja in smo zato mejo v naši implementaciji postavili na 99 %.

Primer 1: Pokažimo, da naša implementacija preverjanja dvojnikov po vsebini zares zazna, da sta dve strani, ki imata različni povezavi, po vsebini enaki.

- Prva stran: https://e-uprava.gov.si/
- Druga stran: https://e-uprava.gov.si/si
- Jaccardova razdalja: 1.0
- Enaka hasha: Da

Torej naša implementacija zazna dve popolnoma enaki strani.

Kot smo že omenili, lahko dve popolnoma enaki strani zaznamo tudi s točno primerjavo njune HTML vsebine ali s primerjavo njunih hash vrednosti HTML vsebin. Zato bomo v naslednjem primeru pokazali še, da naša implementacija zazna kot dvojnici tudi dve strani, ki se razlikujeta zelo malo, kar zgornja dva načina ne moreta.

Primer 2: Pokažimo, da naša implementacija preverjanja dvojnikov po vsebini zares zazna, da sta dve strani, ki imata različni povezavi in se zelo malo razlikujeta po vsebini dvojnici.

- Prva stran: https://sitebulb.com/hints/indexability/canonical-hints/canonical-points-to-a-noindex-url/
- Druga stran: https://sitebulb.com/hints/indexability/canonical-hints/canonical-points-to-a-noindex-nofollow-url/
- Jaccardova razdalja: 0.968503937007874
- Enaka hasha: Ne

Ti dve strani se razlikujeta samo po parih stavkih. Vse ostale informacije pa so enake. Kot je razvidno, naša implementacija zazna ti dve strani za različni in sicer, se razlikujeta za približno 0.315 %. S točno primerjavo njune HTML vsebine ali s primerjavo njunih hash vrednosti HTML vsebin, pa bi bili ti dve strani označeni kot drugačni.

3.5 Robots.txt

Lastniki spletnih strani podajo "robotom", v našem primeru spletnim pajkom, navodila za svojo spletno stran s pomočjo robots.txt datoteke. Na ta način preprečijo brskanje po ne želenih vsebinah, datotekah in imenikih.

Robost.txt datoteka lahko vsebuje več različnih direktiv. Za nas so bile pomembne naslednje:

- **User-agent**: Vsebuje podatek za katere "robote"veljajo direktive, ki sledijo.
- **Disallow**: Vsebuje naslov spletne strani ali dela spletne strani, do katere "robot"ne sme dostopati
- **Allow**: Se uporabi, ko želi lastnik sporočiti "robotu", da lahko dostopa do kakšnega podimenika spletne strani, ki se nahaja pod direktivo Disallow.
- **Crawl-delay**: Vsebuje podatek na koliko sekund lahko "robot"dostopa do spletne strani. Uporablja se zato, da prepreči preobremenitev spletne strani.
- **Sitemap**: Poda povezavo do XML datoteke, ki vsebuje seznam spletnih naslovov, kateri se nahajajo na spletni strani.

Robots.txt se nahaja v izvornem imeniku spletne strani. Zato njegov obstoj preverimo ob vsakem dodajanju nove domene. Če robot.txt obstaja, se sprehodimo po njem in preverimo ali obstaja direktiva *User-agent: **. V primeru, ko obstaja, pregledamo direktive, ki sledijo za njo.

Vsebino direktiv *Allow* in *Disallow*, shranimo v obliki slovarja. Pred dodajanjem strani v frontier URL strani preverimo najprej z nizi v direktivi *Allow*, nato pa še z nizi v *Disallow*. Če se URL ujema s kateremkoli nizom v *Allow*, URL brezpogojno dodamo v frontier, drugače pa ga preverimo še z nizi v *Disallow*, kjer ga ob ujemanju ne dodamo v frontier.

Če obstajajo direktive *Sitemap*, vrednosti iz njih, torej spletne povezave do XML datotek, shranimo v obliki seznama. Nato obiščemo vsako spletno povezavo iz seznama in izluščimo vse povezave. Nakar jih dodamo v frontier.

3.6 Implementacija večnitnosti

Večnitnost smo implementirali z razredom *Process*, ki je vključen v knjižnici *multiprocessing*. Implementirali smo klasični paralelni sistem, kjer glavna nit upravlja delovanje ostalih niti. Njena naloga je zagon posameznih niti ter, v primeru predčasne ustavitve, sprožiti signal za ustavitev.

Niti se med seboj ne potrebujejo sinhronizirati, saj je delo posamezne niti neodvisno od drugih niti. Edina kritična sekcija je pisanje v bazo, saj bi lahko ob pisanju večih niti prišlo do podvajanja zapisov v site tabeli ter večkratnega procesiranja robots.txt. Za reševanje problema kritične sekcije smo uporabili ključavnico, ki je implementirana v razredu *Lock* zgoraj omenjene knjižnice.

4 Problemi pri implementaciji

4.1 Problem s paralelizacijo v Python-u

Python ima drugačno poimenovanje programskih konstruktov kot ostali bolj uporabljeni programski jeziki. *Multithreading* v Pythonu pomeni večnitenje, pri čemer bodo vse niti tekle na enemu procesu zaporedno po principu koroutin.

Za dejansko sočasno izvajanje je potrebno uporabiti večprocesiranje (*Multiprocessing*). Pri tem je razlika od ostalih jezikov ta, da se zažene več fizičnih procesov, kjer se vsakemu procesu poda začeten okvir spremenljivk. Posledično to pomeni, da je komunikacija med procesi skoraj nemogoča.

Problem smo imeli pri implementaciji *crawl-delay*-a, kjer smo na koncu bili primorani implementirati zakasnitev na nivoju baze s pomočjo dodatnega atributa.

4.2 Problem z zaznavo duplikatov po vsebini

Število najdenih duplikatov je relativno veliko.

Zato smo postavili mejo Jaccardove razdalje na 99 %. Kar pomeni, sta dve spletni strani zaznani kot dvojnici, če se razlikujeta za največ 0.9 %. Drugače povedano, se razlikujeta minimalno.

Razlog za izbiro visoke meje leži tudi v uporabi velikega števila hash funkcij, ki omogočajo natančnejše primerjave in posledično zahtevajo višjo mejo.

Velikemu številu dvojnikov so pripomogle tudi strani, ki so zahtevale certifikat. Ker pajek ni podal zahtevanega cetifikata, ni mogel dostopati do njih. Posledično je bila njihova vsebina prazna, torej so imele vrednost null in so se obravnavale kot dvojniki. V tabeli page najdemo tudi nekaj povezav na datoteke za katere končnic ne preverjamo posebej in imajo tudi te zapisi html_content vrednost null.

5 Statistika in analiza rezultatov

V zadnjih tednih so na spletni strani podatki.gov.si (morda tudi drugje) nekajkrat spremenili robots.txt.

Tabela 1 prikazuje statistiko strani, ki jih je pridobil pajek, omejen na osnovne štiri domene (e-prostor, e-uprava, evem, podatki). Tabeli 2 in 3 prikazujeta podrobnejšo statistiko slik in dokumentov. Pajek je bil zagnan približno 40 ur in pol oziroma 2440 minut (originalno 2500 minut, vendar se je vmes prestavila ura).

Referenčna hitrost pajka Apatche Nutch je 290 minut za okoli 3000 strani. To je okoli 5,8 sekund na stran. Na osnovnih štirih domenah je naš pajek s 4 nitmi za 16386 strani potreboval 2440 minut, kar nanese 8,9 strani na sekundo. To vključuje ekstrakcijo slik in dokumentov. Predvidevamo, da je takšna razlika tudi zaradi počasnosti Pythona v primerjavi z Javo, v kateri je napisan Apatche Nutch. Implementacija paralelizacije v Pythonu ("multiprocessing") zelo verjetno tudi vnese kar nekaj "overheada".

Domena	HTML	DUPLICATE	IMAGE	BINARY
e-prostor.gov.si	223	118	132	32
e-uprava.gov.si	2077	2951	1901	1
evem.gov.si	5665	1254	27818	227
podatki.gov.si	8421	374	32087	0
SKUPAJ	16386	4697	61938	260

Tabela 1: Rezultati na začetnih štirih domenah.

Domena	PNG	JPG	GIF	UNKNOWN	SKUPAJ
e-prostor.gov.si	56	54	9	13	132
e-uprava.gov.si	1888	10	0	3	1901
evem.gov.si	27725	78	9	6	27818
podatki.gov.si	8091	11	23970	15	32087
SKUPAJ	37760	153	23988	37	61938

Tabela 2: Statistika najdenih slik začetnih štirih domenah.

Največji dejavnik, ki je prispeval k tako velikemu času pa je ogromno število strani na domeni podatki.gov.si, saj je to praktično iskalnik po bazi podatkov, kjer lahko iščemo po različnih parametrih. V drugi polovici teka, nekje od dvajsete ure dalje, so v frontier-ju ostali le zapisi iz te domene, ki pa ima crawl-delay nastavljen na 10 sekund. To pomeni, da je kljub paralelizaciji teoretičen najhitrejši mogoč čas 10 sekund na stran.

Število slik je tako veliko zato, ker ne preverjamo duplikatov. To nam omogoča, da izračunamo povprečno število slik na spletno stran. V tem primeru je to približno 3,8 slike na stran.

Pajek je dosegel globino 6.

Enako velja za dokumente, ki jih je v tem primeru 0,016 na stran.

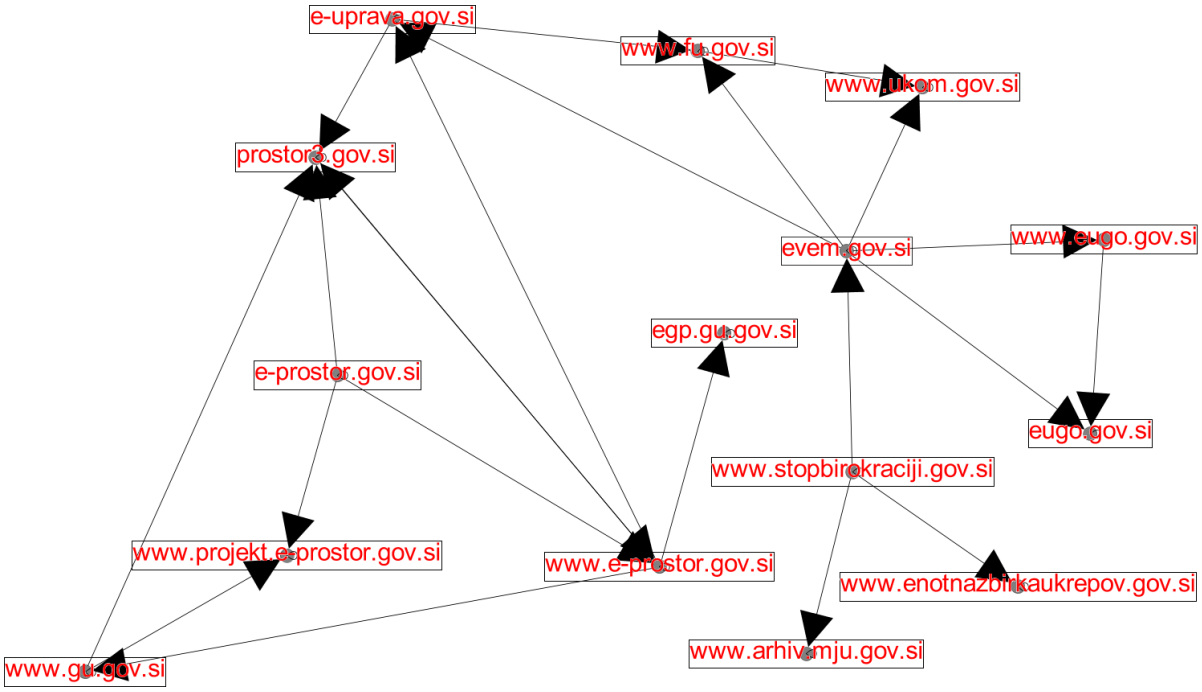
Tabela 4 prikazuje statistiko pajka z začetnimi osnovnimi štirimi domenami in še štirim dodatnimi domenami. V tem primeru slik in dokumentov nismo iskali. Pajek je tekel na 16 nitih, 1444 minut, kar je malenkost več kot 24 ur oziroma 1 dan. Poleg osnovnih osmih domen je našel še 12 dodatnih (pod)domen. Prišel je do globine 10 (to je največja globina z oznako HTML, v frontier-ju so še manjše). Upošteval je samo domene s končnico .gov.si. Hitrost pajka je bila v tem primeru 6,6 sekund na stran, kar je precej boljše od rezultata zgoraj. Rezultat

Domena	PDF	DOC	DOCX	SKUPAJ
e-prostor.gov.si	32	0	0	32
e-uprava.gov.si	1	0	0	1
evem.gov.si	126	96	5	227
podatki.gov.si	0	0	0	0
SKUPAJ	159	96	5	260

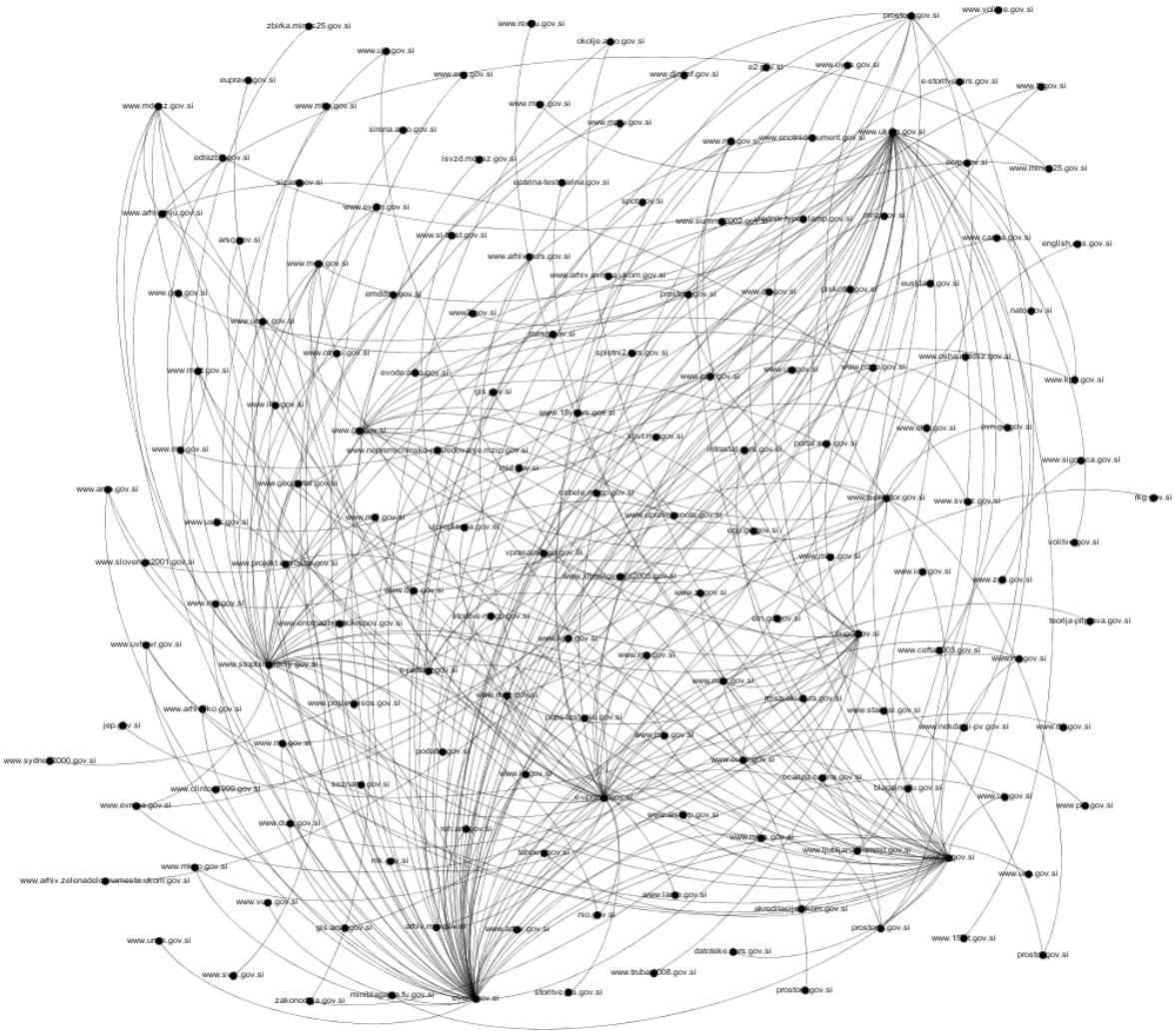
Tabela 3: Statistika najdenih dokumentov na začetnih štirih domenah.

Domena	HTML	DUPLICATE
evem.gov.si *	4853	1051
e-prostor.gov.si *	233	123
e-uprava.gov.si *	2060	2917
podatki.gov.si *	1	0
stopbirokraciji.gov.si *	420	93
ukom.gov.si *	2069	905
gu.gov.si *	1087	1167
fu.gov.si *	1038	1491
prostor3.gov.si	21	15
enotnazbirkaukrepov.gov.si	184	5
ovn.gu.gov.si	7	0
projekt.e-prostor.gov.si	7	0
gis.gov.si	1	1
kpv.gov.si	1	16
arhiv.mju.gov.si	33	1
eugo.gov.si	1077	9
prostor-s.gov.si	9	0
prostor4.gov.si	42	2
vprasalnik.gu.gov.si	2	0
egp.gu.gov.si	7	2
SKUPAJ	13152	7798

Tabela 4: Rezultati z osmimi začetnimi domenami (*).



Slika 1: Tabela povezovanja domen pri čemer se gleda le obiskane strani



Slika 2: Tabela povezovanja vseh domen, tako obiskane kot še neobiskane

je boljši predvsem zato, ker se bile vse niti vseskozi polno izrabljene, razbremenjene pa so bile tudi ekstrakcije slik in dokumentov.

Na sliki 1 lahko vidimo, kako se medsebojno povezujejo domene že obiskanih domen. Na sliki 2 lahko vidimo kako se povezujejo vse domene, tako obiskane kot tiste, ki so še v frontierju.

6 Zaključek

V poročilu smo opisali delovanje našega spletnega pajka. Naredili smo splošen pregled funkcionalnosti nato pa smo opisali podrobnosti. Opisali smo tudi glavne probleme, na katere smo naleteli pri implementaciji pajka. Poročilo smo zaključili z statistiko in predstavitvijo rezultatov.