Olli Peltomaa

Automaattisen verkkoharavoinnin menetelmät ja haasteet

Tietotekniikan kandidaatintutkielma

11. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Olli Peltomaa

Yhteystiedot: olli.m.a.peltomaa@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Automaattisen verkkoharavoinnin menetelmät ja haasteet

Title in English: Methods and challenges of automatic web scraping

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 23+0

Tiivistelmä: Verkkoharavointi on tekniikka, jota käyttämällä voidaan kerätä tietoa internetistä ohjelmallisesti ja sitä voidaan hyödyntää moniin tieteellisiin ja kaupallisiin tarkoituksiin. Verkkoharavointiohjelmat voivat kuitenkin kohdata monenlaisia haasteita, jotka saattavat pakottaa kehittäjän päivittämään haravointiohjelmaa toistuvasti. Kirjallisuuden perusteella käyttöliittymättömät selaimet ja koneoppimisalgoritmit tuottavat yhdessä parhaiten erilaisia haasteita sietävän ohjelman. Verkkoharavoinnin ala on altis nopeille muutoksille, mutta nykyisen kirjallisuuden perusteella koneoppimiseen perustuvissa algoritmeissa on kenties eniten tutkittavaa.

Avainsanat: Verkkoharavointi, verkkosivut, CAPTCHA, internet, päätön selain

Abstract: Web scraping is a technique that can be used to gather information from the Internet programmatically and it can be used for many scientific and commercial purposes. However, web scrapers can face a variety of challenges that may force the developer to update the scraper repeatedly. Based on the literature, headless browsers and machine learning algorithms together produce the best scrapers that tolerates different challenges. The field of web scraping is prone to rapid changes, but based on the current literature, algorithms based on machine learning have perhaps the most research to do.

Keywords: Web scraping, websites, CAPTCHA, internet, headless browser

Kuviot

Sisällys

1 Johdanto

Verkkoharavointi (engl. web scraping) on prosessi, jolla viitataan useimmiten ohjelmalliseen tiedon keräämiseen verkkosivuilta (Hillen 2019). Sen käyttö on yleistynyt viime vuosina niin, että nykyään jopa puolet verkkosivujen kävijämääristä on automaattisten ohjelmien generoimaa liikennettä (Rahman ja Tomar 2021). Boegershausen, Borah ja Stephen (2021) kuvailevat vapaana verkossa olevaa dataa tiedon kultakaivokseksi, mikä voi selittää verkkoharavoinnin suosiota. Verkkoharavoinnilla kerättyä tietoa käytetäänkin monissa kaupallisissa ja tieteellisissä tarkoituksissa. Sen käyttökohteita voivat olla esimerkiksi hintojen vertailu, markkinointi (Hillen 2019) ja mielipidenalyysi (Kaur 2022).

Verkkosivun haravoiminen voi kuitenkin olla sivuston käyttöehtojen vastaista (Krotov, Johnson ja Silva 2020). Tämä on johtanut erilaisten haravointia estävien tekniikoiden käyttöönottoon verkkosivuilla, kuten IP-osoitteiden estoihin (Marques ym. 2018), joita mm. käsitellään enemmän luvussa kolme. Jotta haravointiohjelman toiminta olisi mahdollisimman varmaa, on ohjelman kehittäjän huomioitava edellä mainitun kaltaiset haasteet.

Tämän kirjallisuuskatsauksen tarkoituksena on pohtia parhaita menetelmiä itsenäisten verkkoharavointiohjelmien toteuttamiseen, sekä valottaa verkkoharavoinnin suurimpia haasteita. Itsenäiset haravointiohjelmat tarkoittavat tämän tutkielman kontekstissa ohjelmia, jotka sietävät mahdollisia verkkosivun muutoksia ja ohittavat ilman ihmisen väliintuloa yleisimmät haravoinninestomenetelmät. Tutkielmassa myös pyritään tunnistamaan alan nykytilan perusteella tulevaisuuden tutkimuskohteita.

Tutkielma koostuu kahdesta teorialuvusta, joista ensimmäinen käsittelee erilaisia verkkoharavoinnin menetelmiä, jotka ovat jaettu perinteisiin menetelmiin ja moderneihin tekniikoihin. Tämän jälkeen kolmannessa luvussa esitellään erilaisia verkkoharavoinnin haasteita aloittaen rakenteellisista ongelmista ja lopettaen haravoinnin torjuntatapoihin. Luvussa neljä pohditaan edellä mainittuihin haasteisiin ratkaisuja erilaisilla haravointimenetelmillä ja niiden yhdistelmillä.

2 Verkkoharavoinnin menetelmät

Verkkoharavoinnilla voidaan tarkoittaa myös ihmisen tekemää tekstin kopioimista manuaalisesti hiirtä käyttäen (Sirisuriya 2015), mutta tässä tutkitelmassa keskitytään ohjelmalliseen haravointiin. Alun perin haravointia suoritettiin lähes yksinomaan tekemällä HTTP-pyyntö haluamalleen verkkosivulle ja tutkimalla ohjelmallisesti vastauksena saatua HTML-tiedostoa (Gheorghe, Mihai ja Dârdală. 2018). Gheorghe, Mihai ja Dârdală. kuitenkin mainitsevat, että verkkosivujen kehityttyä on monimutkaisemmille haravointitekniikoille ollut tarvetta. HTML-tiedostoa voidaan tutkia esimerkiksi vertailemalla merkkijonoja käyttäen apuna säännöllisiä lausekkeita, jäsentämällä teksti takaisin HTML DOM-rakenteeseen tai käyttämällä XML-polkukieltä eli XPathia (Darmawan ym. 2022). Edellä mainittujen tapojen jälkeen syvennymme nykyaikaisimpiin haravointimenetelmiin kuten käyttöliittymättömiin selaimiin sekä koneoppimiseen perustuviin menetelmiin.

2.1 Perinteiset menetelmät

Kenties vanhin edelleen yleisessä käytössä oleva keino haravoida tietoa HTML-tiedostoista on säännölliset lausekkeet. Thompson (1968) esitteli jo 1960-luvulla algoritmin, joka etsi tekstistä ennalta määriteltyjä merkkijonoja käyttäen Kleenen ym. (1956) esittelemää teoriaa säännöllisistä tapahtumista (engl. regular events) ja lausekkeista (engl. regular expression). Säännölliset lausekkeet, joihin jatkossa viitataan lyhenteellä RegEx, ovat merkkijonoja, jotka määrittelevät hakumallin, jossa tuntemattomat merkit voidaan esitellä metamerkkeinä (Campesato 2018). Metamerkeillä tarkoitetaan Campesaton mukaan merkkejä, jotka kuvaavat jotakin muuta kuin merkkiä itseään. Tällä tavoin voidaan etsiä tekstistä erilaisia kombinaatioita, jotka vastaavat annettua hakumallia. Esimerkiksi metamerkki `"."` tarkoittaa RegEx:issä mitä tahansa merkkiä (Jarmul ja Lawson 2017) ja metamerkillä `"^"` voidaan etsiä rivejä tekstistä, jotka alkavat sitä seuraavalla merkillä tai merkkijonolla (Campesato 2018).

Päätös käyttää RegEx:ia verkkoharavoinnissa riippuu siitä, onko haettava informaatio yleistettävissä säännölliseen lausekkeeseen (Munzert ym. 2015). Esimerkiksi, jos haettava informaatio on esitetty sivulla epäyhteneväisissä HTML-rakenteissa, saattaa olla vaikeaa tuottaa

RegEx:ia, joka löytäisi kaiken halutun datan. Samoin myös rakenteeltaan liian yhteneväinen verkkosivu voi tuottaa ongelmia. Jos kaikki sivun teksti on HTML-dokumentissa esimerkiksi <**p**> -elementtien sisällä ilman erottavia tunnisteita, RegEx ei välttämättä pysty suodattamaan pois epäolennaista tietoa (Munzert ym. 2015).

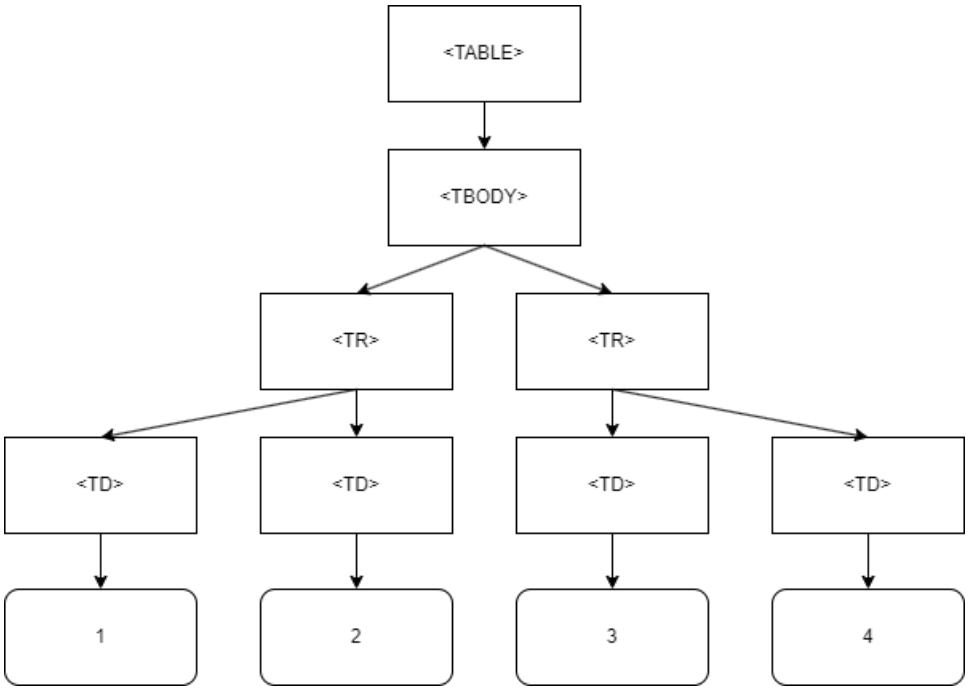
Koska verkkosivuilla on HTML-elementtien muodostama rakenne, voi toisinaan olla hyödyllisempää käyttää tätä rakennetta apuna tiedon löytämiseksi. Sen sijaan, että tutkittaisiin tekstimutoista dataa kuten RegEx:issä, voidaan sivuja tutkia rakenteisesti käyttämällä HTML DOM:ia. HTML DOM on lyhenne englannin kielen sanoista Hypertext Markup Language Document Object Model (Gheorghe, Mihai ja Dârdală. 2018). Se on ohjelmointirajapinta verkkosivun rakenteen määrittäville HTML-tiedostoille, ja sitä käytetään HTML-elementtien muuttamiseen, lisäämiseen, poistamiseen tai hakemiseen (Wood ym. 1998). HTML DOM:issa määritellään kaikkien verkkosivun elementtien objektit ja niiden ominaisuudet sekä metodit niiden saamiseksi (Darmawan ym. 2022). Darmawanin ym. mukaan verkkoselaimen ei tarvitse tuntea DOM:ia esittääkseen HTML-tiedostoja, mutta Javascript pääsee sen avulla käsiksi verkkosivun elementteihin.

Ero tavalliseen HTML-rakenteeseen on se, että kun HTML:ssä elementit ovat sisäkkäin, kuten kuviossa 1, HTML DOM:ia kuvataan useimmiten puurakenteena (Wood ym. 1998). Woodin ym. mukaan HTML DOM:ssa on ainakin yksi juurielementti, joka sisältää verkkosivun rakenteen. Tämä rakenne voi sisältää sisäkkäisiä elementtejä kuten kuviossa 2, jota voidaan käyttää hyväksi verkkoharavoinnissa. Tämän sisäkkäisen rakenteen avulla voidaan navigoida puun alkioiden läpi, kunnes ohjelma löytää halutun elementin, jonka sisältämä data voidaan ottaa talteen (Gheorghe, Mihai ja Dârdală. 2018).

Toinen tapa etsiä tietoa rakenteisesti verkkosivulta on käyttää apunaan XPathia. XPath (lyhennys sanoista XML Path Language) on kyselykieli, jonka pääasiallisena tarkoituksena on valita osia XML-dokumentista, mutta sitä voidaan käyttää myös tavallisiin merkkijono-, numero- ja totuusarvomuuttujaoperaatioihin (Clark, DeRose ym. 1999). Kuten HTML DOM, myös XPath kuvaa Clarkin, DeRosen ym. mukaan XML-dokumentin puurakenteena. He sanovat tämän rakenteen sisältävän solmuja, jotka voivat olla tyypiltään elementti-, ominaisuus- tai tekstisolmuja. Gunawan ym. (2019) mainitsevat, että XPath:ia voidaan käyttää myös HTML-tiedostojen kanssa, ja siten se sopii käytettäväksi myös verkkoharavoinnissa. Sen

```
<TABLE>
<TBODY>
<TR>
<TD>1</TD>
<TD>2</TD>
</TR>
<TR>
<TD>3</TD>
<TD>4</TD>
</TR>
</TBODY>
</TABLE>
```

Kuvio 1. Taulukko HTML-tiedostossa, perustuu Woodin ym. (1998) kuvioon.



Kuvio 2. Taulukko HTML DOM objektina, perustuu Woodin ym. (1998) kuvioon.

kenties hyödyllisin ominaisuus tiedon haravoimisessa Gunawanin ym. mukaan on sijaintipolut, joiden avulla voidaan valita dokumentista tiettyjä solmuja.

Sijaintipolut valitsevat joukon solmuja suhteessa kontekstisolmuun (Clark, DeRose ym. 1999), joka tarkoittaa sillä hetkellä tarkasteltavaa solmua (Clark ym. 1999). Sijaintipolkuja käyttämällä voidaan valita verkkosivuilta elementtejä, jotka ovat kontekstisolmun lapsi- tai vanhempisolmuja, tietyn ominaisuuden omaavia solmuja tai ovat elementiltään tietynlaisia (Clark, DeRose ym. 1999). Esimerkiksi, Clarkin, DeRosen ym. mukaan kyselyllä `chapter[title="Introduction"]` valitaan kaikki kontekstisolmun `chapter` -elementtiä olevat lapsisolmut, joilla on yksi tai useampi `title` -elementti lapsisolmunaan, joiden merkkijonoarvona on "Introduction".

Kolmas keino rakenteen hyödyntämiseen tiedon etsinnässä on CSS:n valitsimien käyttö. CSS (engl. Cascading Style Sheets) on kieli, jolla voidaan muokata HTML-sivun ulkoasua. Ulkoasua voidaan muokata CSS:ssa käyttämällä hyödyksi valitsimia, joilla voidaan asettaa tyyli yksittäiselle elementille tai joukolle elementtejä (Meyer 2006). Valitsimien avulla voidaan valita HTML-elementtejä niiden tunnisteiden (id), luokan, tyypin tai attribuutin perusteella (Kanaoka ja Toyama 2015). Esimerkiksi valitakseen kaikki `strong` -elemetit, jotka ovat `h1` -elementin lapsielementtejä, voidaan ne CSS:n valitsimilla valita seuraavasti: `h1 > strong` (Meyer 2006). Samoja valitsimia voidaan käyttää hyödyksi myös verkkoharavoinnissa, sillä niiden avulla voidaan valita halutut elementit verkkosivuilta, ja hakea niiden sisältämä data (Darmawan ym. 2022).

2.2 Modernit ratkaisut

Nykyisin niin kutsuttujen käyttöliittymättömien- tai päättömien selaimien (engl. headless browser) käyttö on yleistynyt verkkoharavoinnissa. Gheorghe, Mihai ja Dârdală. (2018) mukaan päättömillä selaimilla voidaan imitoida ihmisen vuorovaikutusta verkkosivulla, kuten hiiren painallusta ja sivun selaamista. Suositut päättömät selaimet, kuten Selenium ja Puppeteer ovat pääasiassa kehitetty verkkosivujen testaamiseen (“SeleniumLibrary” 2022; “Puppeteer” 2023), mutta niitä voidaan hyödyntää myös verkkoharavoinnissa. Päätön selain poikkeaa 2.1 luvussa esitellyistä menetelmistä siten, että pelkän HTML-sivun hakemisen sijasta

päätön selain lataa kaikki asiakaspuolen skriptit aivan kuin tavallinen selain (Qudus Khan ym. 2021).

Monet päättömät selaimet käyttävät hyödykseen jo aikaisemmin luvussa 2.1 esiteltyjä tiedon haravointimenetelmiä. Esimerkiksi suositut päättömät selaimet kuten Puppeteer ja Selenium tukevat verkkosivun elementtien valitsemista käyttäen CSS-valitsimia ja XPathia ("Puppeteer" 2023; "SeleniumLibrary" 2022). Datan valitsemisen ja painikkeiden painamisen lisäksi mm. Puppeteer-kirjasto mahdollistaa muitakin hyödyllisiä ominaisuuksia, kuten näyttökuvien ottamisen, sivun muuttamisen PDF-tiedostoksi ja kirjautumislomakkeiden täyttämisen ("Puppeteer" 2023).

Aiemmin esiteltyjen haravointitekniikoiden lisäksi on viime vuosina ollut kehitteillä yhä enemmän koneoppimiseen perustuvia menetelmiä (Khder 2021). Tällaisia kehitteillä olevia ratkaisuja ovat olleet muun muassa konenäköä hyödyntävät ohjelmat, jotka pyrkivät tunnistamaan ja keräämään tietoa verkkosivuilta visuaalisesti, kuten ihmisetkin (Sirisuriya 2015). Eräs tapa, jolla tietokone voi tunnistaa asioita visuaalisesti, on käyttää tekstintunnistusalgoritmeja (engl. Optical Character Recognition). Tekstintunnistusalgoritmit, joihin viitataan jatkossa lyhenteellä OCR, ovat koneoppimiseen perustuvia menetelmiä kuvan sisällön ymmärtämiseksi (Serrao, Salunke ja Mathur 2013).

OCR:n toiminta on jaettu kolmeen vaiheeseen: esikäsittelyyn, segmentointiin ja tunnistukseen (Serrao, Salunke ja Mathur 2013). Esikäsittelyvaiheessa kuvasta pyritään Serraon, Salunken ja Mathurin mukaan siivoamaan tarpeettomat yksityiskohdat, jotta vain tärkeintä informaatiota käsitellään myöhemmissä vaiheissa. Heidän mukaansa segmentoinnissa kuvasta yritetään tunnistaa kiinnostavat kohteet eli ROI:t (engl. Region of Interest), jotka voivat olla esimerkiksi kirjaimia. Viimeisessä vaiheessa, eli tunnistuksessa, he sanovat kuvasta pyrittävän tunnistamaan esim. kirjain mahdollisista eri vaihtoehdoista.

Perinteisten haravointimenetelmien taakkana on se, että niiden avulla tehdyt ohjelmat eivät kestä verkkosivujen rakenteen muutoksia (Qudus Khan ym. 2021). Tämän ongelman ratkaisemiseksi kannattaa tutkia koneoppimisen mahdollisuuksia tarkemmin. Nykykirjallisuus kuitenkin käsittelee koneoppimista verkkoharavoinnin kontekstissa lähinnä erilaisten CAPTCHA-testien ohittamisessa, ei niinkään tiedon etsimisessä. Khder (2021) mainitsee

tutkimuksessaan, että koneoppimisen hyödyntäminen voi johtaa haravointitekniikoihin, jotka oppivat aikaisemmasta kokemuksesta ja siten osaavat automaattisesti mukautua moniin verkkosivuihin. Koneoppimiseen perustuviin haravointiohjelmiin keskittyvää avointa kirjallisuutta on kuitenkin ollut vaikeasti saatavilla, mikä korostaa tulevaisuuden tutkimuksen tarvetta.

3 Verkkoharavoinnin haasteet

Verkkoharavointi ei aina ole täysin suoraviivaista, sillä toisinaan tiedon saamiseksi sivuilta on jouduttu ohittamaan monenlaisia haasteita. Ongelmia aiheuttavat useimmiten verkkosivun rakenne ja sen muutokset, mutta toisinaan haasteet on varta vasten suunniteltu estämään automaattisten ohjelmien pääsyä sivulle. Esimerkiksi verkkokaupoille verkkoharavointia pidetään yhtenä suurimmista uhista, sillä kaupan hintojen haravointi voi vähentää myyntiä ja laskea sivun näkyvyyttä hakukoneissa (Rahman ja Tomar 2021). Aina verkkoharavointi ei siis ole toivottua, minkä vuoksi sitä vastaan on kehitetty erilaisia suojausmekanismeja. Tässä luvussa esitellään ensin verkkoharavoinnin yleisiä haasteita, jotka liittyvät verkkosivujen rakenteisiin, jonka jälkeen syvennytään varsinaisiin automaattisia ohjelmia vastaan kehitettyihin suojausmekanismeihin.

3.1 Rakenteelliset ongelmat

Yksi tavallisimmista haasteista, joita verkkoharavointiohjelmat kohtaavat, on verkkosivun uudistumisesta johtuva rakenteen muutos. Jos verkkosivun rakenne muuttuu, yksinkertainen haravointiohjelma ei välttämättä enää kykene löytämään samoja tietoja kuin aiemmin. Tällöin haravointiohjelman kehittäjä joutuu päivittämään ohjelmaansa vastaamaan uuden sivun rakennetta (Qudus Khan ym. 2021). Ongelman yleisyydestä kertoo se, että esimerkiksi Pereira (2022) mainitsee joutuneensa päivittämään haravointiohjelmiensa lähdekoodeja kuusi kertaa viiden kuukauden aikana, kun hän haravoi tietoa matkailualan verkkosivuilta. Pereiran havaitsemaa ongelmaa tukee myös se, että Ezzat, G Abd ElJalil ja Othman (2019) saivat selville, että heidän tutkimuksessaan mukana olevista matkatoimistoista jopa 50% saattoi päivittää sivujaan päivittäin.

Verkkosivujen päivittyminen ei kuitenkaan ole ainut tilanne, jolloin sivun rakenne voi muuttua. Jo 2000-luvun alussa Lam ja Ngan (2006), mainitsivat dynaamisten verkkosivujen määrän kasvavan valtavasti. Lamin ja Nganin mukaan monet verkkosivut ovatkin yhteydessä tietokantoihin, joista sisältöä ladataan sivulle. Sen lisäksi, että tietoja ladataan palvelimelta sivulle, voidaan Javascriptiä käyttämällä manipuloida HTML-elementtejä ja siten luoda,

poistaa tai muokata elementtejä myös asiakaspuolella (Ullman 2012). Tämä tarkoittaa sitä, että pelkkä HTML-tiedosto itsessään ei aina sisällä kaikkea haluttua tietoa (Jarmul ja Lawson 2017). Tälläisiä sivuja voidaan kutsua dynaamisiksi verkkosivuiksi.

Päivittyvien ja dynaamisten verkkosivujen sijaan Gallagher ja Beveridge (2022) mainitsevat verkkosivujen epäyhtenäisen luonteen ensimmäisenä luetellessaan verkkoharavoinnin teknisiä haasteita. Haasteita aiheuttaa esimerkiksi se, jos verkkosivu käyttää HTML-elementeissään epäsäännöllisesti CSS-luokkia samalla kun haravointiohjelma käyttää haravoinnissa apunaan CSS-valitsimia. Tämä ei kuitenkaan täysin estä haravointia, kuten aikaisemmin esitellyt rakenteelliset haasteet. Gallagher ja Beveridge sanovat tämän vain johtavan dataan, joka ei ole johdonmukaista tai täydellistä. Heidän mukaansa tämä on ongelma lähinnä vain sivuilla, joiden kehityksessä ei ole käytetty frontend-kirjastoja. Tällaisten sivujen määrä on kuitenkin ollut jo pitkään vähentymässä. Laazirin ym. (2019) mukaan useimmat verkkosivut käyttävät jo jonkinlaista frontend-kirjastoa, jonka avulla voidaan helpommin luoda standardisoituja ja hyvin jäsenneltyjä sivuja.

3.2 Haravoinnin torjuntatavat

Tavallisille verkon käyttäjille CAPTCHA:t (engl. Completely Automated Public Turing Test To Tell Computers and Humans Apart) ovat kenties tutuin esimerkki keinoista, joilla automaattisten ohjelmien pääsyä verkkosivuille rajoitetaan. Ensimmäinen CAPTCHA, joka esiteltiin vuonna 1997, generoi kuvan satunnaisesta tekstistä, jonka käyttäjän tuli syöttää tekstikenttään jatkaakseen eteenpäin (Moradi ja Keyvanpour 2015). Myöhemmin CAPTCHA:sta on tehty monenlaisia variaatioita, mutta Moradi ja Keyvanpour ovat karkeasti jaoteltuna saaneet sisällytettyä ne kolmeen pääkategoriaan: visuaalisiin, ei-visuaalisiin ja hybridimallisiin CAPTCHA:hin.

Moradi ja Keyvanpour (2015) kuitenkin mainitsevat, että vaikka ääniin perustuvat CAPTCHA:t käyttävät hyödykseen tietokoneen heikkoutta tunnistaa sanoja taustahälyn joukosta, ovat ääniin perustuvat CAPTCHA:t kuitenkin haasteellisia myös käyttäjille. Moradin ja Keyvanpourin mukaan kaikissa tilanteissa ei esimerkiksi ole mahdollista käyttää kaiuttimia tai kuulokkeita, ja eri kuulovamman asteisille henkilöille ei välttämättä ole mahdollista ratkaista

ääneen perustuvia CAPTCHA:ja. Visuaaliset CAPTCHA:t ovatkin tästä syystä yleisempiä, ja tässä tutkielmassa keskitytään jatkossa niihin, kun puhutaan CAPTCHA:ista.

Nykyisin Googlen tarjoama reCAPTCHA on yleisin käytössä olevan CAPTCHA:n muoto (Sivakorn, Polakis ja Keromytis 2016). Sivakornin, Polakisin ja Keromytisin mukaan reCAPTCHA esittää ensin valintaruudun otsikolla "I'm not a robot", josta painamalla reCAPTCHA suorittaa riskianalyysin sivulle pyrkijästä. He mainitsevat riskianalyysin koostuvan mm. pyrkijän selainhistorian -ja ympäristön arvioinnista. Tämän perusteella käyttäjä joko päästetään sivustolle tai hänelle tarjotaan visuaalinen CAPTCHA ratkaistavaksi.

Tavallisille verkon selaajille CAPTCHA:aa näkymättömämpi, mutta haravointiohjelmien kannalta konkreettinen haaste on palvelimien käyttölokien valvonta. Sillä tarkoitetaan tapaa tunnistaa ylimääräistä liikennettä, ihmiselle epätyypillistä käyttäytymistä tai HTTP-pyyntöjä, joiden muoto ei ole samanlainen kuin ihmisten ohjaamalla selaimella (Gheorghe, Mihai ja Dârdală. 2018). Tätä valvontaa voidaan Gheorghen, Mihain ja Dârdalan mukaan suorittaa manuaalisesti tai automaattisesti erilaisten palomuurisovellusten avulla. He mainitsevat myös, että IP-osoitteet, jotka jäävät kiinni valvonnassa, voidaan estää joko väliaikaisesti tai pysyvästi epätoivotun haravoinnin estämiseksi.

Toinen ihmiskäyttäjälle näkymätön keino estää verkkoharavointia on käyttää niin kutsuttuja hunajapurkkeja (engl. honeypot). Hunajapurkeiksi kutsutaan menetelmiä, joiden tarkoituksena on harhauttaa verkkoharavat tuhlaamaan aikaa ja resursseja hämäykseksi tarkoitettuihin kohteisiin (Cohen 2006). Hunajapurkki voi olla esimerkiksi HTML-sivun rakenteessa olevan kirjautumislomakkeen kenttä, joka on CSS:ssä asetettu piilotetuksi (Moradi ja Keyvanpour 2015). Moradin ja Keyvanpourin mukaan tälläiseen kenttään ihmiskäyttäjä ei voi kirjoittaa tietoa, mutta raakaa HTML-sivua tutkiva haravointiohjelma taas pääsee siihen käsiksi. Moradi ja Keyvanpour sanovat, että jos tällaiseen lomakkeen kenttään on lisätty tekstiä, voidaan olettaa, että käyttäjä ei ole ihminen.

4 Menetelmiä haasteiden ohittamiseksi

Verkkoharavointiin liittyy monia haasteita, joten on vaikea valita yhtä tapaa ohittamaan kaikkia edellä mainittuja ongelmia. Qudus Khan ym. (2021) kuitenkin pyrkivät artikkelissaan ratkaisemaan näistä useimmat käyttäen algoritmia, jonka he kehittivät Unity Asset Store Web Scrapper (UAWS) -nimiseen ohjelmaan. UAWS-ohjelman algoritmi on Qudus Khanin ym. mukaan aikavaativuudeltaan lineaarinen, joka sopii hyvin suurille datamäärille. Se myös pyrkii heidän mukaansa mukautumaan verkkosivun rakenteen muutoksiin, jolloin verkkosivujen päivitysten jälkeen ei tarvitse muuttaa haravointiohjelman toimintaa.

UAWS-algoritmin tiedonkeruuprosessi on jaettu kahteen osaan. Ensin se generoi listan sivuista, joilla sen tulee vierailla, ja sen jälkeen käy jokaisen sivun läpi yksitellen haravoiden halutut tiedot talteen (Qudus Khan ym. 2021). Tallennettujen tietoja perusteella, halutuille hakusanoille voidaan tunnistaa alku-ja lopputunnisteita, joiden avulla samat tiedot voidaan etsiä sivulta uudestaan myös verkkosivun rakenteen muututtua. Alkutunniste voi olla esimerkiksi `<td>First name:` ja lopputunniste `</td>`. Qudus Khan ym. toteavat, että vaikka kyseiset tunnisteet voivat usein vaihtua verkkosivun rakenteen muutoksessa, niiden attribuuttien arvot usein eivät. Näitä attribuuttien arvoja heidän algoritminsa käyttää hyväkseen etsiäkseen halutut tiedot muuttuneesta rakenteesta ja tallentaakseen niille uudet tunnisteet.

Qudus Khan ym. (2021) mainitsevat verkkoharavoinnin haasteina HTML DOM:in ja XPathin suuren muistin käytön, sekä XPath:iin ja RegEx:iin perustuvien ohjelmien pitkän suoritusajan. Aihe tuntuu kuitenkin olevan kiistelty, sillä Darmawan ym. (2022) saivat tutkimuksessaan selville, että XPath oli nopeampi kuin HTML DOM, CSS-valitsimet ja RegEx. UAWS-ohjelman algoritmissa käytettiin edellä mainittujen tekniikoiden nopeudesta tai hitaudesta riippumatta päätöntä selainta, sillä sen avulla voidaan ratkaista kaksi yleistä ongelmaa. Ensimmäiseksi, palvelin kohtelee sitä kuin ihmisen ohjaamaa selainta, jolloin palvelin todennäköisemmin hyväksyy sen pääsyn sivulle (Qudus Khan ym. 2021). Toiseksi, luvussa 2.2 mainitaan päättömän selaimen lataavan asiakaspuolen skriptit. Qudus Khanin ym. mukaan nämä lataavat dynaamisen sisällön sivulle, jolloin kaikki halutut tiedot ovat saatavilla.

Haravoinnin estomenetelmistä Qudus Khan ym. ovat ohittaneet hunajapurkit päätöntä selain-

ta käyttämällä. Kuten 2.2 luvussa todettu, päättömät selaimet lataavat asiakaspuolen skriptit aivan kuin tavallisetkin selaimet. Tällöin 3.2 luvussa kuvaillut CSS:n avulla piilotetut elementit eivät ole haravointiohjelman saatavilla, toisin kuin perinteisesti tutkimalla HTTP-pyynnöllä saatua HTML-tiedostoa. Qudus Khan ym. eivät kuitenkaan olleet algoritmissaan ottaneet huomioon CAPTCHA:ja. Edellä esiteltyä algoritmia voisi tämän epäkohdan ratkaisemiseksi jatkojalostaa käyttämään hyödykseen luvussa 2.2 esiteltyä OCR:a. OCR:aa käyttämällä voidaan ohittaa testit, jotka pyytävät kirjoittamaan vääristyneen sanan kuvasta tai valitsemaan kuvia, jotka sisältävät tiettyjä asioita.

Mikäli sivusto käyttää luvussa 3.2 esiteltyä reCAPTCHA:a, voidaan testi ohittaa jopa pelkällä valintalaatikon klikkauksella. Sivakorn, Polakis ja Keromytis (2016) esittelivät tutkimuksessaan tavan välttyä kokonaan visuaalisilta CAPTCHA:ilta generoimalla evästeitä, jotka vaikuttavat tulevan tavalliselta käyttäjältä. Kuten luvussa 3.2 todettu, reCAPTCHA päästää sivulle ilman ylimääräisiä testejä, jos pyrkijä on riskianalyysissa todettu vaarattomaksi. Sivakornin, Polakisin ja Keromytisin generoimat uudet evästeet useimmiten päästettiin sivulle ongelmitta, kunhan niiden oli annettu ensin ikääntyä jonkin aikaa ennen niiden käyttöä. Sivakorn, Polakis ja Keromytis kuitenkin raportoivat löydöksistään reCAPTCHA:n ohittamiseksi Googlelle, joka sittemmin on muuttanut riskianalyysin toimintaa. Generoitujen evästeiden toimivuudesta tänä päivänä ei siis ole varmuutta, mikä kasvattaa tarvetta visuaalisten CAPTCHA:jen ratkaiseville algoritmeille.

CAPTCHA:t kuitenkin onnistuvat nykyisin hyvin erottelemaan ihmiset ohjelmista niiden ratkaisunopeuden perusteella (Chiapponi ym. 2022). Chiapponin ym. mukaan raja-arvo ihmisen ja ohjelman CAPTCHA:n ratkaisunopeudelle voidaan saada verkkoharavointiohjelmien ratkaisunopeuden normaalijakaumasta. Tällöin tietyn ajan alittavat tai ylittävät suoritukset voidaan katsoa ihmisen tekemiksi. Chiapponi ym. kuitenkin mainitsevat, että nykyisin monet haravointiohjelmat ovat turvautuneet niin kutsuttuihin CAPTCHA-farmeihin, joissa ihmiset ratkaisevat CAPTCHA:ja maksua vastaan ohjelmien puolesta. Tämä johtaa ratkaisunopeuksiin, jotka antavat ymmärtää sivulle pyrkijän olevan ihminen.

Haravointiohjelma voi välttyä jäämästä luvussa 3.2 esitellyn käyttölokien valvonnan hampaisiin ottamalla huomioon haravoinnin hyvät käytännöt (engl. crawling politness policy). Haravoinnin hyvät käytännöt huomioimalla voidaan säästää verkkosivun palvelinta ylikuor-

mitukselta mm. implementoimalla haravointiohjelmaan aikaviive pyyntöjen välille (Gheorghe, Mihai ja Dârdală. 2018). Eettisten haravointiohjelmien tulisikin noudattaa sääntöjä, joita verkkosivun ylläpitäjä on asettanut robots.txt-tiedostossa (Sun, Zhuang ja Giles 2007). Robots.txt-tiedostolla, tai vaihtoehtoisesti Robots-metatunnisteella, voidaan tietyillä sivun osilla vierailut joko sallia tai kieltää käyttämällä `Disallow` ja `Allow` direktiivejä (Yang ja Liao 2010). Yang ja Liao mainitsevat myös `crawl-delay` direktiivin, jolla voidaan ohjeistaa haravointiohjelmaa odottamaan tietyn aikaa ennen kuin se pyytää sivua uudetaan. Näitä ohjeita seuraamalla haravointiohjelma simuloi ihmiskäyttäytymistä ja toimii hyvien tapojen mukaisesti (Gheorghe, Mihai ja Dârdală. 2018). Tällöin sivuston ylläpitäjällä ei ole syytä IP-osoitteen estolle.

5 Yhteenveto

Verkkosivujen ja verkkoharavoiden välillä on käyty taistelua jo pitkään. Aina, kun yksi taistelun osapuoli löytää uuden tavan päihittää vastustajansa keinot, toinen löytää jo uuden ratkaisun sen voittamiseksi. Tästä esimerkkinä voidaan pitää CAPTCHA-farmeja, joita verkkoharavat ovat alkaneet käyttää hyödykseen CAPTCHA:jen kehityttyä tunnistamaan haravointiohjelmat tehokkaammin. Ala on siis altis nopeille muutoksille, mistä johtuen kaikki tässä tutkielmassa käsitellyt menetelmät tai haasteet eivät välttämättä ole relevantteja enää muutaman vuoden päästä.

Tämänhetkisen tiedon valossa varmin tapa toteuttaa mahdollisimman itsenäisiä ja toimintavarmoja verkkoharavointiohjelmia on käyttää hyödykseen päättömiä selaimia ja koneoppimista. Luvussa 4 esitelty, UAWS-ohjelman algoritmi on jo ratkaissut suurimman osan luvussa 3 luetelluista haasteista. UAWS:n algoritmi mm. saa käyttöönsä koko verkkosivun dynaamisen sisällön, sillä päätön selain lataa pelkän HTML-sivun lisäksi asiakaspuolen skriptit. Päätöntä selainta myös kohdellaan kuten tavallista selainta, joten IP-osoitteiden estoilta voidaan välttyä sitä käyttämällä. Haravointiohjelman tulee tämän lisäksi myös muistaa lukea mahdolliset ohjeet robots.txt-tiedostosta, jotta ohjelman IP-osoite ei joudu estolistalle liian nopeiden pyyntöjen tai kiellettyjen sivujen haravoinnin vuoksi.

Koneoppimisalgoritmit, kuten OCR, ovat kirjallisuuden perusteella paras tunnettu tapa ratkaista CAPTCHA-testejä ilman välikäsiä. Niiden toimintavarmuus on kuitenkin heikentynyt testien kehityttyä, minkä vuoksi maksullisiin CAPTCHA-farmeihin on jouduttu turvautumaan. Kirjallisuuden perusteella haravointiohjelmat ja haravoinnin estomenetelmät ovat sykleittäin toistensa niskan päällä, joten on hyvin mahdollista, että algoritmien kehityttyä nykyiset CAPTCHA:tkin voidaan ratkaista uusien koneoppimismallien avulla. Koneoppimisen hyödyntämistä varsinaiseen tiedon etsimiseen ja keräämiseen ei vielä ole tutkittu kovinkaan paljoa, vaan sitä on enemmänkin hyödynnetty erilaisten testien ratkomiseen. Koneoppimisen saralla siis näyttäisi olevan vielä tulevaisuudessa tutkittavaa myös tiedon keräämisen alueella.

Lähteet

Boegershausen, Johannes, Abhishek Borah ja Andrew T Stephen. 2021. “Fields of gold: Web scraping for consumer research”. *Marketing Science Institute Working Paper Series,* 1–58.

Campesato, Oswald. 2018. *Regular Expressions.* Mercury Learning & Information. https://search-ebscohost-com.ezproxy.jyu.fi/login.aspx?direct=true&db=e000xww&AN=1826926&site=ehost-live.

Chiapponi, E., M. Dacier, O. Thonnard, M. Fangar, M. Mattsson ja V. Rigal. 2022. “An industrial perspective on web scraping characteristics and open issues”. Teoksessa *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S),* 5–8. https://doi.org/10.1109/DSN-S54099.2022.00012.

Clark, James, ym. 1999. “Xsl transformations (xslt)”. *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/xslt* 103. Viitattu 21. huhtikuuta 2023. https://www.w3.org/TR/1999/REC-xslt-19991116.

Clark, James, Steve DeRose ym. 1999. *XML path language (XPath).* Viitattu 21. huhtikuuta 2023. https://www.w3.org/TR/1999/REC-xpath-19991116/.

Cohen, Fred. 2006. “The use of deception techniques: Honeypots and decoys”. *Handbook of Information Security* 3 (1): 646–655.

Darmawan, I., M. Maulana, R. Gunawan ja N. Widiyasono. 2022. “Evaluating Web Scraping Performance Using XPath, CSS Selector, Regular Expression, and HTML DOM With Multiprocessing Technical Applications”. *JOIV : International Journal on Informatics Visualization* 6 (4): 904. https://doi.org/10.30630/joiv.6.4.1525.

Ezzat, Mohamed, Sabreen G Abd ElJalil ja Asmaa Othman. 2019. “The Role of Egyptian Travel Agencies’ Websites in Increasing Tourist’s Online Trust”. *International Journal of Tourism and Hospitality Management* 2 (1): 108–125. https://doi.org/10.21608/IJTHM.2019.52149.

Gallagher, John R, ja Aaron Beveridge. 2022. “Project-oriented web scraping in technical communication research”. *Journal of Business and Technical Communication* 36 (2): 231–250. https://doi.org/10.1177/10506519211064619.

Gheorghe, M., F. Mihai ja M. Dârdală. 2018. “Modern techniques of web scraping for data scientists”. *Romanian Journal of Human - Computer Interaction* 11 (1): 63–75.

Gunawan, Rohmat, Alam Rahmatulloh, Irfan Darmawan ja Firman Firdaus. 2019. “Comparison of web scraping techniques: regular expression, HTML DOM and Xpath”. Teoksessa *2018 International Conference on Industrial Enterprise and System Engineering (ICoIESE 2018),* 283–287. Atlantis Press. https://doi.org/10.2991/icoiese-18.2019.50.

Hillen, J. 2019. “Web scraping for food price research”. *British Food Journal* 121 (12): 3350–3361. https://doi.org/10.1108/BFJ-02-2019-0081.

Jarmul, K., ja R. Lawson. 2017. *Python Web Scraping - Second Edition : Successfully Scrape Data From Any Website with the Power of Python 3.x.* Nide Second edition. Packt Publishing. ISBN: 9781786462589. https://search-ebscohost-com.ezproxy.jyu.fi/login.aspx?direct=true&db=e000xww&AN=1528142&site=ehost-live.

Kanaoka, Kei, ja Motomichi Toyama. 2015. “Browser GUI for generating web data extraction rules in Ducky”. Teoksessa *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services,* 1–5. https://doi.org/10.1145/2837185.2837262.

Kaur, P. 2022. “Sentiment analysis using web scraping for live news data with machine learning algorithms”. *Materials today : proceedings* 65:3333–3341. https://doi.org/10.1016/j.matpr.2022.05.409.

Khder, Moaiad Ahmad. 2021. “Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application.” *International Journal of Advances in Soft Computing & Its Applications* 13 (3). https://doi.org/10.15849/IJASCA.211128.11.

Kleene, Stephen C, ym. 1956. “Representation of events in nerve nets and finite automata”. *Automata studies* 34:3–41. https://doi.org/10.1515/9781400882618-002.

Krotov, V., L. Johnson ja L. Silva. 2020. “Tutorial: Legality and Ethics of Web Scraping”. *Communications of the Association for Information Systems* 47:555–581. https://doi.org/10.17705/1CAIS.04724.

Laaziri, Majida, Khaoula Benmoussa, Samira Khoulji, Kerkeb Mohamed Larbi ja Abir El Yamami. 2019. “Analyzing bootsrap and foundation font-end frameworks: a comparative study”. *International Journal of Electrical and Computer Engineering (IJECE)* 9 (1): 713–722. https://doi.org/10.11591/ijece.v9i1.pp713-722.

Lam, Kam-Yiu, ja Chris CH Ngan. 2006. “Temporal pre-fetching of dynamic web pages”. *Information Systems* 31 (3): 149–169. https://doi.org/10.1016/j.is.2004.11.007.

Marques, P., Z. Dabbabi, M-M. Mironesc, O. Thonnard, A. Bessani, F. Buontempo ja I. Gashi. 2018. “Using Diverse Detectors for Detecting Malicious Web Scraping Activity”. *Paper presented at the IEEE/IFIP International Conference on Dependable Systems and Networks,* https://doi.org/10.1109/DSN-W.2018.00033.

Meyer, Eric A. 2006. *CSS: The Definitive Guide: The Definitive Guide*. "O’Reilly Media, Inc.".

Moradi, Mohammad, ja MohammadReza Keyvanpour. 2015. “CAPTCHA and its Alternatives: A Review”. *Security and Communication Networks* 8 (12): 2135–2156. https://doi.org/10.1002/sec.1157.

Munzert, S., C. Rubba, P. Meißner, D. Nyhuis ja P. Meißner. 2015. *Automated Data Collection with R : A Practical Guide to Web Scraping and Text Mining*. John Wiley & Sons, Incorporated. https://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=1824310.

Pereira, S. 2022. “Automated web scraping and data visualisation for tourism based on popular accommodation platforms”, viitattu 22. huhtikuuta 2023. https://hdl.handle.net/1822/81399.

“Puppeteer”. 2023. Viitattu 28. maaliskuuta 2023. https://pptr.dev/.

Qudus Khan, Fazal, Georgios Tsaramirsis, Naimat Ullah, Mohamed Nazmudeen, Sadeeq Jan ja Awais Ahmad. 2021. “Smart algorithmic based web crawling and scraping with template autoupdate capabilities”. *Concurrency and Computation: Practice and Experience* 33 (22): e6042. https://doi.org/10.1002/cpe.6042.

Rahman, R. U., ja D. S. Tomar. 2021. “Threats of price scraping on e-commerce websites: attack model and its detection using neural network”. *Journal of Computer Virology and Hacking Techniques volume* 3:75–89. https://doi.org/10.1007/s11416-020-00368-6.

“SeleniumLibrary”. 2022. Viitattu 28. maaliskuuta 2023. https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html.

Serrao, Merrill, Shanu Salunke ja Amrita Mathur. 2013. “Cracking CAPTCHAS for cash: a review of CAPTCHA crackers”. *International Journal of Engineering Research & Technology* 2 (1): 1–5. https://www.ijert.org/cracking-captchas-for-cash-a-review-of-captcha-crackers.

Sirisuriya, De S. 2015. “A Comparative Study on Web Scraping”. *Proceedings of 8th International Research Conference of KDU. General Sir John Kotelawala Defence University,* 135–140. Viitattu 26. tammikuuta 2023. http://ir.kdu.ac.lk/handle/345/1051.

Sivakorn, Suphannee, Jason Polakis ja Angelos D Keromytis. 2016. “I’m not a human: Breaking the Google reCAPTCHA”. *Black Hat* 14:1–12.

Sun, Yang, Ziming Zhuang ja C Lee Giles. 2007. “A large-scale study of robots. txt”. Teoksessa *Proceedings of the 16th international conference on World Wide Web,* 1123–1124. https://doi.org/10.1145/1242572.1242726.

Thompson, Ken. 1968. “Programming techniques: Regular expression search algorithm”. *Communications of the ACM* 11 (6): 419–422. https://doi.org/10.1145/363347.363387.

Ullman, Larry. 2012. *Modern JavaScript: Develop and Design.* Peachpit Press.

Wood, Lauren, Arnaud Le Hors, Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Gavin Nicol, Jonathan Robie, Robert Sutor ym. 1998. “Document object model (dom) level 1 specification”. *W3C recommendation* 1. https://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/.

Yang, Chyan, ja Hsien-Jyh Liao. 2010. “Using the Robots. txt and Robots Meta tags to implement online copyright and a related amendment”. *Library hi tech* 28 (1): 94–106. https://doi.org/10.1108/07378831011026715.